2012 IEEE 36th International Conference on Computer Software and Applications Workshops

An Image Processing Approach to Solve Labyrinth Discovery Robotics Problem

Behnam Rahnama
European University of Lefke
Dept. of Computer Engineering
Gemikonagi-KKTC,Turkey
behnam@rahnama.com

Atilla Elçi
Sulymen Demirel University
Dep. Of Computer & Educational Tech.
Isparta, Turkey
atilla.elci@gmail.com

Shadi metani
European University of Lefke
Dept. of Computer Engineering
Gemikonagi-KKTC,Turkey
Sha2003de@hotmail.com

Abstract— **Maze solving using multiple algorithms is one of the important problems in the last years, maze solving problem is to determine the path of a mobile robot from its source position to a destination position through a workspace populated with obstacles, in addition to finding the shortest path among the solutions. Autonomous robotics is a field with wide-reaching applications, from bomb-sniffing robots to autonomous devices for finding humans in wreckage to home automation; many people are interested in low-power, high speed, reliable solutions. we will introduce an algorithm to solve mazes and avoid robots to go through the long process, this algorithm is based on image processing and shortest path algorithm, the algorithm will work efficiently because of the preprocessing on maze's image data rather than going through the maze cell by cell. Therefore, it focuses on the entire maze rather than the current part that an agent is in it. This approach will give the robot preplanning time before falling in mistakes like loops and being trapped in a local minimum of dead ends of labyrinth besides saving time, because the entire maze map will be known before the navigation starts.**

Keywords: maze solving, maze solving algorithms, image processing, image rotating, image scaling, and edge detection.

I. INTRODUCTION

Essentially, you have a maze made up of a 16 by 16 grid of cells. Mice must find their way from a predetermined starting position to the finish. The robot will need to keep track of where it is, discover walls as it explores, map out the maze and detect when it has reached the goal using the shortest path when the maze has multiple solutions.

There are a number of different maze solving algorithms, that is, automated methods for the solving of mazes. To find the shortest path between the start and the end is an important issue, which the researchers care about while developing an algorithm. Traffic control is a real life example of maze solving technology, to find the shortest path for the ambulance or fire fighters.

Robotics competitions have interested the engineering community for many years. Robots compete on an international scale hundreds of times a year, in events as varied as the annual MIT robot competition and the BEAM robotics games. One of the competitions with the richest history is the MicroMouse competition. Micromice are small, autonomous devices designed to solve mazes quickly, and efficiently. The MicroMouse competition has existed for almost 20 years in the United States, and even longer in Japan. It has changed little since its inception. The goal of the contest is simple: the robot must navigate from a corner of a maze to the center as quickly as possible. The actual final score for a robot is primarily a function of the total time in the maze and the time of the fastest run. MicroMouse mazes are typically designed as a lattice of square posts with slots in the sides to accommodate modular wall units.

II. LITERATURE REVIEW

Up to now, many approaches have been proposed to solve this problem, Institute of Electrical and Electronic Engineering (IEEE) holds an international MicroMouse competition each year, the aim is to develop robotic system that can solve the maze and find the shortest path between the start and the end points. One of the earliest systems developed by a group of students from California State University in 2002 introduced a solving for a maze using high quality and price equipments, the idea is to fit an ordinary radio-controlled toy car with a Charge-Coupled Device (CCD) camera and a transmitter, image processing and control of the car are accomplished by a personal computer (PC), so the image of the road captured by the camera and transferred to the computer that process and digitalize the image and introduce the drive command that transfer to the robot again (A vision-guided autonomous vehicle: an alternative micromouse competition, 1997). Data structure called a sparse direction map was proposed to solve maze, this map contains directions used to guide agents from any position to the goal, maps were evolved using a variant of an elitist Simple Genetic Algorithm, this method successfully evolve maps of three types of mazes of varying size and complexity but not allows the shortest path (Evolving sparse direction maps for maze path finding, 2004). In 2003 an 8-geometry maze routing algorithm for planning shortest path in an image workspace with obstacles of arbitrary shapes presented, starting from a top view of a workspace with obstacles, the so-called free workspace is first obtained by virtually expanding the obstacles in the image, they prove the efficiency of the algorithm with simulation results, but this algorithm will not work without

978-0-7695-4758-9/12 $26.00 © 2012 IEEE
DOI 10.1109/COMPSACW.2012.114

the top view of the work space which is not allowed in an international competition (A new maze routing approach for path planning of a mobile robot, 2003). Jianping Cai, Xuting Wan, Meimei Huo and Jianzhong Wu proposed a maze exploring algorithm named "Partition-Central Algorithm", used to find the shortest path in a micromouse competition maze, this algorithm based on dividing a standard maze into multiple partitions and in each partition different search strategy is adopted according to the current absolute direction of the micromouse, this algorithm performed more feature than classic central algorithm (An Algorithm of Micromouse Maze Solving, 2010). One of the latest researches lead by Swati Mishra and Pankaj Bande in 2008 produces several algorithms to solve mazes, begin with very basic Wall Follower logic to solve the maze, and gradually improves the algorithm to accurately solve the maze in shortest time with some more intelligence, the algorithm is developed up to some sophisticated level as Flood-Fill algorithm, and then gave comparison between them as a result they said if we don't have any time and hardware constraint we can effectively use the Djikstra's algorithm, but if both are the constraints then Flood Fill would be superior to others, if we do not wish to have any complex calculation to embed in the system, we can stick to the left/right wall follower, but for this we need to have a previous knowledge of the maze, whether it is right-walled or left-walled. Thus, the Flood Fill is by far the most effective of all (Maze Solving Algorithms for Micro Mouse, 2008).

high efficiency algorithm developed by group of researchers in 2008, this algorithm is using Center-First-Theorem which is based on selecting the direction which points to the center first other than other directions at a bifurcation junction, after finding the short paths among the labyrinth a comparison process starts to find the shortest one, this algorithm avoids the solver from the complex work of recording the whole routing information of the maze and so reduces the errors, improve the efficiency and makes the programming of route searching become easily (A high efficiency Center-First-Routing-Theorem algorithm of micro-mouse, 2008). One of the developed methods is assigning and manipulating artificial potentials to provide locally optimized path choices while maintaining the integrity of the potentials, The basic algorithm is improved by retaining information of the number of decisions that have been made (A potential maze solving algorithm for a micromouse robot, 1995), the methods of optimal control is also used to find the shortest path (Shortest distance paths for wheeled mobile robots, 1998). In 2010 a Constructing Shortest Path Maps for planning turn-constrained Collision-Free paths for robotic vehicles with limited turn-radius has been discussed, if we want to compute the shortest path between the source vertex and a target vertex such that the path does not contain sharp turns, then the standard Shortest Path Map SPM cannot be used, so they construct the CSPM for a collection of convex polygonal obstacles such that it can be used to compute the shortest path that does not contain sharp turns. (Generalization of Shortest Path Map, 2010).

In the last years, there has been a greater interest in systems of multiple autonomous robots for the accomplishment of cooperative tasks. In 2005 a research describe the implementation and development of a distributed architecture for the programming and control of a team of coordinated heterogeneous mobile robots, which are able to collaborate among them and with people in the accomplishment of tasks of services in daily environments, different robots, each one with its own configuration, are more flexible, robust and cost effective. Moreover, the tasks to achieve may be too complex for one single robot, whereas they can be effectively done by multiple robots (A framework for the development of cooperative robotic applications, 2005). In 2007 research by A. Elci, and B. Rahnama Multi-agent system based on semantic web services used to discover the labyrinth was developed; these distributed standalone processors as individual agents use the web service to achieve the connectivity, each robot as an agent communicate with others about position, context, discovered area or walls cause the robots to find the shortest path (Human-Robot Interactive Communication Using Semantic Web Tech. in Design and Implementation of Collaboratively Working Robots, 2007), After that, they developed a Decision-making algorithm on collaborative labyrinth discovery robots using Semantic Web technology, they utilize both the Open World Assumption (OWA) and Closed World Assumption (CWA), then comparison and a strategy provided in deciding on selecting either of CWA/OWA to power decision-making and reasoning (Defining a Strategy to Select Either of Closed/Open World Assumptions on Semantic Robots, 2008). Recently the path planning approach in the cooperative robotics system attracts the researchers, due to the characteristics such as recombine ability and self-adaptability of cooperative robots group, the problem of path planning for robot group is formalized as Multiple Travelling

III. PROPOSED APPROACH

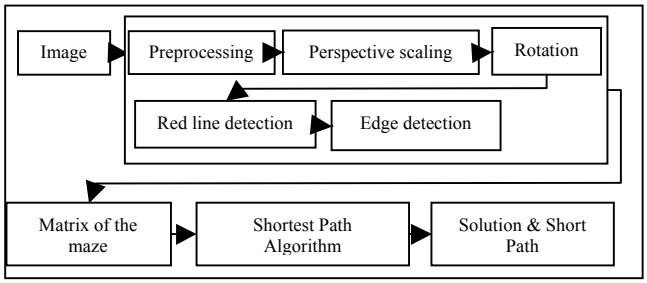

Figure 1 Proposed algorithm

A. Capturing and Merging Images

We used web camera to take photos from different positions around the maze, we captured photos from the four corners of the maze. From each corner while the angle of view is 90, we took 7 photos with 15 degree between them. As in figure 2 shows.

Figure 2 Sample of photos captured from corner of the maze.

After that as a preprocessing phase to prepare the input image, we used Microsoft ICE in merging these photos to create panoramic photo of the entire maze, Microsoft Image Composite Editor is an advanced panoramic image stitcher. Given a set of overlapping photographs of a scene shot from a single camera location, the application creates a high-resolution panorama that seamlessly combines the original images. Figures 3 is samples of panoramic photos we created.

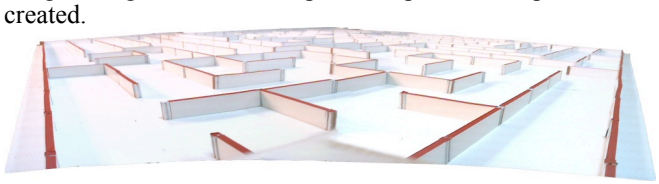

Figure 3 Result after merging the photos of left up corner.

B. *Image Perspective Phase*

By examining the maze image we see in figure 3, we notice that the fare parts of the maze (walls) smaller and not clear like the nearer parts, and this is normal because of perspective problem of the image, in this phase we will solve this problem by scaling the nearer parts to be Proportional with the farer parts.

Bilinear interpolation method implemented using image processing toolbox in Matlab, Figure 4 shows the maze image scaled from the down two corners, first a special transformation structure built to scale the image from the down two corners, then the image transformed according to that structure, scaling the two down corners or reducing the width of the image from one side of the image solve the perspective problem, the precision of the image fixed by adjusting the scaling values to be somewhere between the two corners to give good quality, after scaling, the empty area usually filled with black pixels, which eliminated by cropping the image, or by easily removing the pixels that have value of black color.

In the next codes, by the first statement the original image corners coordination set, after that corners transferred to the expected place by multiplying the original places with value less than one, this will scale the corners position, after applying imtransform function, by maketform function a spatial transformation structure created, finally the last function transfers the image to new coordinate by scaling the corners to new positions. Figure 4 result image after applying these codes.

```
b_corner = [0 0; wd 0; 0 ht; wd ht];
t_corner = [0 0; wd 0; wd*0.46 ht; wd*0.543 ht];
T = maketform('projective',b_corner,t_corner);
perspective im = imtransform(base_im,T,'bilinear');.
```

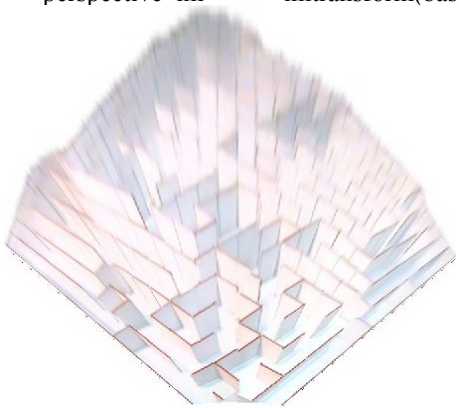

Figure 4 scaled image to solve the perspective problem

C. *Rotation Phase*

The result of the previous phase is an image in diamond shape, to read this image and process the pixels we need to rotate and scale it to be rectangle and one corner must be located in the origin, so this phase will rotate the image with specific angle to be rectangle, in geometry and linear algebra, a rotation is a transformation in a plane or in space that describes the motion of a rigid body around a fixed point. A rotation is different from a translation, which has no fixed points, the rotation applied to transfer the image to rectangle shape and match the x axes on the upper side and y axes by the left side this step is required to be able to read the image pixels easily and got the image matrix.

In Matlab the rotation operator performs a geometric transform which maps the position (x1, y1) of a picture element in an input image onto a position (x2, y2) in an output image by rotating it through a specified angle q about an origin. In most implementations, output locations (x2, y2) which are outside the boundary of the image are ignored, so it filled by black color that eliminate lately. Rotation is most commonly used to improve the visual appearance of an image, although it can be useful as a preprocessor in applications where directional operators are involved (Gonzalez & Woods, 2002).

The rotation operator performs a transformation of the form:

$$x_2 = \cos\theta * (x_1 - x_0) - \sin\theta * (y_1 - y_0) + x_0$$

$$y_2 = \sin\theta * (x_1 - x_0) + \cos\theta * (y_1 - y_0) + y_0$$

Where (x0, y0) are the coordinates of the center of rotation (in the input image), and θ is the angle of rotation with clockwise rotations having positive angles. (Note here that we are working in image coordinates, so the y axis goes downward. Similar rotation formula can be defined for when the y axis goes upward. Even more than the translate operator, the rotation operation produces output locations (x2, y2) which do not fit within the boundaries of the image (as defined by the dimensions of the original input image). In such cases, destination elements which have been mapped outside the image are ignored by most implementations. Pixel locations out of which an image has been rotated are usually filled in with black pixels, we eliminated this area by

cropping the image or by easily removing the pixels that have value of black color, the result is rectangle image of the maze it's left upper corner in the (0, 0) coordinate, and this will make the operation of reading the image easy to perform, in the next figure the result image after rotation of -47 degrees, the image rotated counterclockwise 47 degrees.

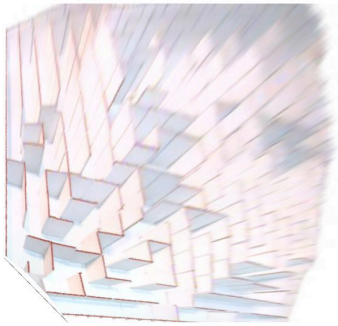

Figure 5 rotated image to solve the coordination problem

The codes below is used to rotate the image, perspective_rotated_im = imrotate(perspective_im,-47) rotates image perspective_im by angle degrees in a clockwise direction around its center point. To rotate the image counterclockwise, specify a positive value for angle. Imrotate makes the output image perspective_rotated_im large enough to contain the entire rotated image. imrotate uses nearest neighbor interpolation and loose method used to make the output image large enough to contain the entire rotated image, setting the values of pixels in B that are outside the rotated image to 0 or black, so the next codes used to change the pixels value with 0 to be 255 that mean white.

B = imrotate(A,angle,method) rotates image A, using the interpolation method specified by method. Method is a text string that can have one of these values. There are three rotation methods in Matlab 'nearest` neighbor, 'bilinear' interpolation and 'bicubic' interpolation, nearest fast one but produces aliasing effects, the bilinear doe's anti-aliasing, but is slightly slower, the bicubic does anti-aliasing, preserves edges better than bilinear interpolation, but gray levels may slightly overshoot at sharp edges. This is probably the best method for most purposes, but also the slowest.

```
perspective_rotated_im = imrotate(perspective_im,-47,' bicubic ','loose');
blackPixels = perspective_rotated_im == 0;
perspective_rotated_im(blackPixels) = 255;
```

D. Edge Detection

Edge detection performed to detect the walls of the maze, detecting the walls and so the paths is an important step to create the matrix of the image, which can be easily read then using one of the maze solving algorithm. To solve the maze using the maze image we suggest using shortest path algorithms.

Edges = edge(GrayImage,'canny'); takes a grayscale or a binary image GrayImage as its input, and returns a binary image Edges of the same size as GrayImage, with 1's where the function finds edges in GrayImage and 0's elsewhere. By default, edge uses the Sobel method to detect edges.

The operator first convolves the image with a Gaussian kernel to perform the noise reduction (just as the Prewitt operator has the regions of 1s to perform averaging). It then differentiates the image in the two orientations. Rather than perform one convolution for smoothing and a further two for differentiation, the smoothing and differentiation kernels are combined and the whole operation is performed using two convolutions, this technique used to detect the wall of the maze, but problem appeared because of the low edges of the walls appeared as well.

One more phase added to detect the red color over the walls, thus only the red pixel will detected, then when apply the edge detection canny method we will have only the upper side of the walls.

In the figure bellow Image after applying edge detection canny method (a) after applying the red object mask (b) before applying red mask, the first one show only the upper edge of the walls, but the second has noise that is the lower edges.

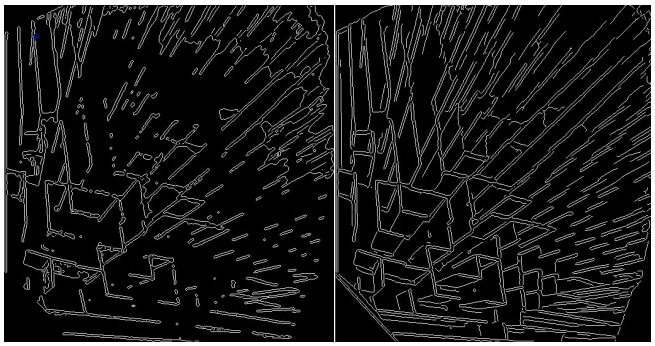

Figure 6 Image after applying edge detection canny method

E. Red color detection

Matlab has the ability to find out exactly how much red, green, and blue content there is in an image. You can find this out by selecting only one color at a time and viewing the image in that color. Red object color mask created using the codes below, the red object mask used to mask out the red-only portions of the RGB maze image, so only the red parts of the walls remained figure 7.

Applying the Canny edge detection algorithm to detect the walls, giving good results because the changes in pixel intensity will be high, thus it's change from black to white, Figure 6 Show the image before and after applying the red color mask.

Figure 7 is the result image after applying the codes with maze image, the noise in this image will not affect the result, that's an averaging function used to calculate the average of surrounding pixel values of one pixel expected to on the wall, if the average more than threshold then a wall should be there.

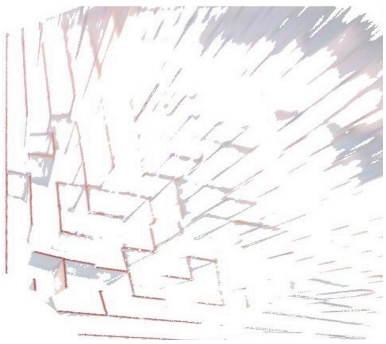

Figure 7 Result after applying the red object mask on the image.

IV. Experimental Results

Maze solving ideal based algorithm demonstrates better results than other algorithms (Rahnama, 2009). The propose is to provide the entire maze information to the robot before starting navigation. Solve then navigate technique is the aim of our study; we can have the information of the maze by examining the output image which is produced at the image processing part.

The preprocessing on the maze image before robot starts navigation will give the robot a change to pre-plan the time and select the appropriate method before falling in mistakes like loops and being trapped in local minima. Figure 3.2 Shows experiment result for the ideal based solving algorithm. As seen in this Figure, knowing the maze structure beforehand will improve the navigation performance to a great deal.

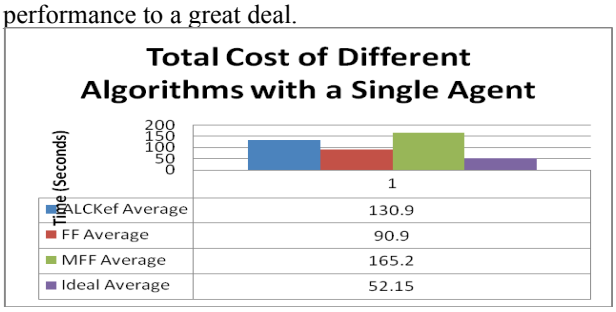

Figure 8 Total Cost of Different Algorithms with a Single Agent.

A. Wall detection

You have a maze made up of a 16 by 16 grid of cells as shown in Figure 9. Mice must find their way from a predetermined starting position to the finish. The robot will need to keep track of where it is, discover walls as it explores, map out the maze and detect when it has reached the goal using the shortest path when the maze has multiple solutions. So, first the robot has to detect the walls, the maze image with size M x N, so divide the width by 16 will give the number of pixels that each cell will have, while dividing the height and the width with 16 will give the coordination of the center pixel of the first cell, then by adding or subtracting the amount of (M/32) will be the coordination of a pixel value on the border of the cell, then by examining this pixel value, if its red then there is wall.

This function return each cell walls (north, east, south, west), the wall set to one if the average value of block pixels around the pixel on the wall more than threshold, and zero if the average les, as we mentioned before and because of the noise in the image we used average function to detect the wall, we will explain the BlockAverage later in this chapter.

This function described here takes the image as an input, first finds cell center points (*x*,*y*), then adds (m/32) to find coordination of point in the wall, (d) decide the wall we searching for, starting with one as north wall and finishing with four as west wall, sending this points (x, y, d) to *BlockAverage* function to return the average of block pixel around that point, if the average is more than threshold then that wall exists and output is then set to one, if not, then no wall exists and output is then set to zero.

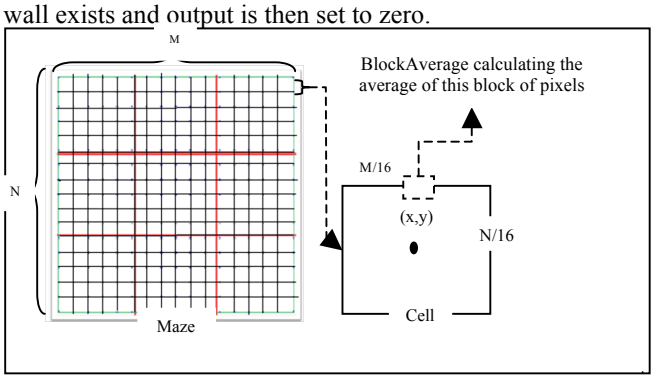

Figure 9 16 x 16 maze and cell with walls (north, east, south and west).

As mentioned before BlockAverage function takes point coordination on the wall and calculates the around pixels (blocks) then returns the average, the average is used to decide if a wall exists or not. Because of the noise in the image not all walls are well detected. That is why an averaging function is used to solve this problem, thus if the average is more than threshold, the existence of a wall at that point is assumed.

B. The maze Matrix

In this phase we will create the binary matrix out of the image, the perspective problem phase fixed the distances between the walls, then filtered the red color only to detect the walls. Finally edge detection method was used to find information related to walls only. The distance between walls fixed is, so that the paths can be easily noticed, but because of the noise that the image has due to the sequence of processing, we created an averaging function that will check the walls around every single cell. If the average of the pixels is more than a threshold, a wall is said to be present.

The north walls of all cells: cell(:,:,1) =

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	0	0	0	0	0	1	0	1	1	1	1	0
0	1	1	1	0	0	1	0	0	0	0	1	1	1	1	0
0	1	1	1	0	1	1	0	0	0	0	1	0	0	1	0
0	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0
0	1	1	0	0	0	0	1	1	1	1	0	1	1	1	0
0	1	0	0	0	0	0	1	1	0	0	0	1	1	1	1
0	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0
0	0	1	0	0	1	1	1	0	1	0	0	0	0	0	0
0	0	0	0	0	1	1	0	1	1	0	0	1	0	0	0
0	0	0	0	1	1	0	0	0	1	0	0	1	1	0	0

0 1 1 1 1 0 0 1 0 0 0 0 1 1 0 0
0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 0
0 1 1 1 0 0 0 0 0 1 1 0 1 1 0 0
0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 0

The following codes create the walls by the matrix we created in the previous section, if the wall is set to one then a line is drawn, the first loops created all walls except the right and the down edges of the maze.

V. CONCLUSION

We introduced an algorithm to solve the maze that avoids the long process to save time and memory. The proposed algorithm is based on image of the maze, the algorithm works efficiently because of the preprocessing on the entire maze`s image data before the robot starts navigation, so the robot will have information about the maze and all the paths, only capturing the image and processing it will be enough to make the robot navigate to the target without searching because the path will be known, this will give the robot preplanning time and selection of the method before falling in mistakes like loops and being trapped in a local minimum.

In this research an algorithm to solve maze discovery problem has been produced, based on an image of the maze, to create and preprocessing the data of the entire maze. This avoids the robot traversing long pathways and saves time and memory. Even if it fails to cover the entire maze it still better then navigating the whole maze cell by cell. Results show that not all wall are detected properly, this means image processing techniques is go to approximate half of the maze, but for complete solution it doesn't give proper solution, so its better if we do it in process as long as the robot approaches to the end try to capture more images so we get more accurate results.

Image processing is not the complete solution, if we had taken an image from top of the maze this would work in a better form, but in this situation it’s not completely appropriate.

REFERENCES

[1] Cong, Y. Z., & Ponnambalam, S. (2009). Mobile robot path planning using ant colony optimization. Advanced Intelligent Mechatronics (pp. 851 - 856). Singapore: IEEE.

[2] Drupal. (n.d.). Micromouse Book. Retrieved august 2010, from Micromouse Online: http://www.micromouseonline.com/book/micromouse-book

[3] Elci, A., & Rahnama, B. (2007). Human-Robot Interactive Communication Using Semantic Web Tech. in Design and Implementation of Collaboratively Working Robots. Robot and Human interactive Communication (pp. 273 - 278). Jeju: IEEE.

[4] Elci, A., Rahnama, B., & Kamran, S. (2008). Defining a Strategy to Select Either of Closed/Open World Assumptions on Semantic Robots. Computer Software and Applications (pp. 417 - 423). Turku: IEEE.

[5] Garro, B., Sossa, H., & Vazquez, R. (2006). Path Planning Optimization Using Bio-Inspirited Algorithms. Artificial Intelligence (pp. 319 - 330). Mexico City, Mexico: IEEE.

[6] Gewali, L., & Roman, V. (2010). Generalization of Shortest Path Map. Information Technology: New Generations (ITNG) (pp. 296 - 300). Las Vegas, NV: IEEE.

[7] Gonzalez, R. C., & Woods, R. E. (2002). Digital Image Proccessing. New jersey: Prentice Hall, Inc.

[8] Gordon, V., & Matley, Z. (2004). Evolving sparse direction maps for maze path finding. Evolutionary Computation (pp. 835 - 838). IEEE.

[9] Goschin, S., Franti, E., Dascalu, M., & Osiceanu, S. (2007). Combine and compare evolutionary robotics and reinforcement Learning as methods of designing autonomous robots. Evolutionary Computation (pp. 1511 - 1516). Singapore: IEEE.

[10] Hassouna, M., Abdel-Hakim, A., & Farag, A. (2005). Robust robotic path planning using level sets. Image Processing (pp. III - 473-6). IEEE.

[11] Hongshe Dang, J. S. (2010). An Efficient Algorithm for Robot Maze-Solving. Xi’an-China: IEEE.

[12] Huh, D.-J., Park, J.-H., Huh, U.-Y., & Kim, H.-i. (2002). Path planning and navigation for autonomous mobile robot. IECON 02 [Industrial Electronics Society (pp. 1538 - 1542). IEEE.

[13] Jan, G. E., Chang, K.-Y., & Parberry, I. (2003). A new maze routing approach for path planning of a mobile robot. Advanced Intelligent Mechatronics, 2003 (pp. 552 - 557). IEEE.

[14] Manjunath, T., Nagaraja, B., Kusagur, A., & Gopala. (2009). Simulation & Implementation of Shortest Path Algorithm with a Mobile Robot Using Configuration Space Approach. Advanced Computer Control (pp. 197 - 201). Singapore: IEEE.

[15] maze. (2010, October 12). Retrieved august 14, 2010, from wikipedia: http://en.wikipedia.org/wiki/Maze

[16] maze solver-theory. (n.d.). Retrieved august 09, 2010, from micromouse: http://www.societyofrobots.com/member_tutorials/book/export/html/94

[17] Mishra, S., & Bande, P. (2008). Maze Solving Algorithms for Micro Mouse. Signal Image Technology and Internet Based Systems (pp. 86 - 93). Bali: IEEE.

[18] Nebot, P., & Cervera, E. (2005). A framework for the development of cooperative robotic applications. Advanced Robotics (pp. 901 - 906). Seattle, WA: IEEE.

[19] Ordonez, C., Jr, E. G., Selekwa, M. F., & Dunlap, D. D. (2008). The virtual wall approach to limit cycle avoidance for unmanned ground vehicles. Robotics and Autonomous Systems , 645-657.

[20] Pullen, W. D. (1996). mazealgorithms. Retrieved october 14, 2010, from think labyrinth: http://www.astrolog.org/labyrnth/algrithm.htm

[21] R. Fisher, S. P. (2003). Sobel Edge Detector. Retrieved from image proccessing learning resources: http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

[22] Su, L., & Tan, M. (2005). A virtual centrifugal force based navigation algorithm for explorative robotic tasks in unknown environments. Robotics and Autonomous Systems , 261-274.

[23] Wang, D. (2008). A linear-time algorithm for computing collision-free path on reconfigurable mesh. Elsevier.

[24] Wang, D., Yu, X., Wan, W., & Xu, H. (2008). A new method of infrared sensor measurement for micromouse control. Audio, Language and Image Processing (pp. 784 - 787). Shanghai: IEEE.

[25] Warren, C. (2002). Global path planning using artificial potential fields. Robotics and Automation (pp. 316 - 321). Scottsdale, AZ : IEEE.

[26] Wu, Y.-R., Tsai, M.-C., & Wang, T.-C. (2005). Maze routing with OPC consideration. Design Automation Conference (pp. 198 - 203). IEEE.

[27] Wyard-Scott, L., & Meng, Q.-H. (1995). A potential maze solving algorithm for a micromouse robot. Communications, Computers, and Signal Processing (pp. 614 - 618). Victoria, BC: IEEE.

Yu, Z., Jinhai, L., Guochang, G., Rubo, Z., & Haiyan, Y. (2002). An implementation of evolutionary computation for path planning of cooperative mobile robots. Intelligent Control and Automation (pp. 1798 - 1802). IEEE.