Graph Construction Algorithm for finding the Shortest Path in a Maze

Milena Karova[1], Ivaylo Penev[2], Mariana Todorova[3], Hristiyan Bobev[4], Neli Kalcheva[5]

Abstract – **The paper presents implementation of an algorithm for movement of a robot from a start position to a final destination in a maze. The algorithm solves two main problems: transformation of the maze into a graph and finding the shortest path from an initial to a final position. The algorithm is implemented as a part of an application, using already generated text image, obtained by a picture of the maze from top. Walls (obstacles), empty spaces, starting position of the robot and the final destination are marked on the image. The maze is presented as a bit set to form a graph which vertexes are valid positions of the robot (i.e. the robot can rotate without touching a wall). The algorithm finds the shortest path, marks movement commands and saves them into a text file.**

Keywords – **Bit Set, Breadth First Search, Graph, Image, Maze, Shortest Path**

I. Introduction

The movement of a robot in a maze is a problem with many applications into lots of areas. Therefore the problem is an object of a broad research interest.

The main problems, which have to be solved, are presentation of the maze in a proper form and implementation of an algorithm for movement of the robot from initial position to a final destination (exit) into the maze. In the existing solutions the robot sensors or the camera of the robot are usually used [1, 3, 4]. Using these devices only a part of the maze could be shot. As a result information for the obstacles (the walls) in the maze is obtained during the robot movement. This approach is suitable, if the robot moves in an unknown or dynamically changing environment, but has some disadvantages:

- Requires permanent recalculation of the movement trajectory;
- Troubles the application of effective algorithms for path finding.

Various shortest path finding algorithms for robot movement in a maze have been researched. Some of the commonly used algorithms are: Dijkstra algorithm, A* algorithm, breadth-first search (BFS), depth-first search (DFS) and etc.

Dijkstra's algorithm is one of the simplest algorithms. Starting from the initial vertex where the path should start, the algorithm marks all direct neighbors of the initial vertex with the cost to get there. It then proceeds from the vertex with the lowest cost to all of its adjacent vertices and marks them with the cost to get to them via itself if this cost is lower. Once all neighbors of a vertex are checked, the algorithm proceeds to the vertex with the next lowest cost [2, 5].

A* is like Dijkstra's algorithm in that it can be used to find a shortest path. A* algorithm is the most popular choice for path finding, because it's fairly flexible and can be used in a wide range of contexts. It is one of a family of graph search algorithms that follow the same structure. These algorithms represent the map as a graph and then find a path in that graph. Depending on the environment, A* algorithm might accomplish search much faster than Dijkstra's algorithm [5].

Graph traversing in depth (Breadth-First-Search or BFS) is algorithm for path searching in trees or graphs [6]. Searching starts from a given vertex in the graph (or from the root in case of a tree) and all the unvisited neighbors are visited first. The algorithm is proposed by E. F. Moore, who used it for finding the shortest path in a maze.

The algorithm, proposed in this paper, is called AlgoHris. It solves two main problems:

- Construction of a graph using the image of the maze, shot from above (for example by the camera of a drone);
- Finding the shortest path from an initial position to a final destination in the graph.

The AlgoHris algorithm is compared to three other algorithms: Backtracking, A* and Genetic Algorithm GAPP.

II. Algorithm AlgoHris formulation nstructions for the authors

The AlgoHris algorithm considers an image of the maze. The image is shot from above, for example by the camera of a drone. Thus AlgoHris is integrated into an application for shooting a maze from above, finding shortest path in the maze and moving a robot from an initial position to a final destination in the maze (Fig. 1).

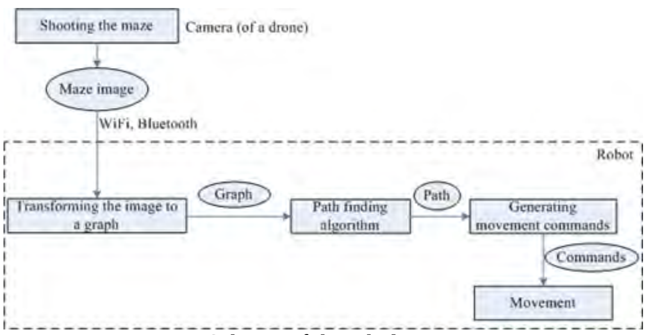


Fig. 1. Schema of the whole project

[1]Milena Karova, [2]Ivaylo Penev, [3]Mariana Todorova, [5]Neli Kalcheva are with the Faculty of Automation and Computing at Technical University of Varna, 1 Studentska str., Varna 9010, Bulgaria, E-mail: mkarova@ieee.bg, ivailo.penev@tu-varna.bg, mgtodorova@yahoo.com, n_kalcheva@abv.bg

[4]Hristiyan Bobev is a student with the Faculty of Automation and Computing at Technical University of Varna, 1 Studentska str., Varna 9010, Bulgaria, E-mail: hristiyan_bobev@abv.bg.

The proposed algorithm consists of two parts:

- Construction of a graph, presenting the maze;
- Finding the shortest path from an initial position to a final destination, using the graph.

A. Graph construction age

The construction of a graph is an essential problem for the robot movement. This problem is hard due to the limited resources of the robot platform (computing power, memory).

The proposed AlgoHris algorithm uses presentation of the maze as a set of characters 0 and 1, where 0 marks free (possible) position for the robot movement and 1 marks busy position (i.e. an obstacle or a wall). Fig. 2 presents a simplified maze and a graph with an initial position and a final destination.

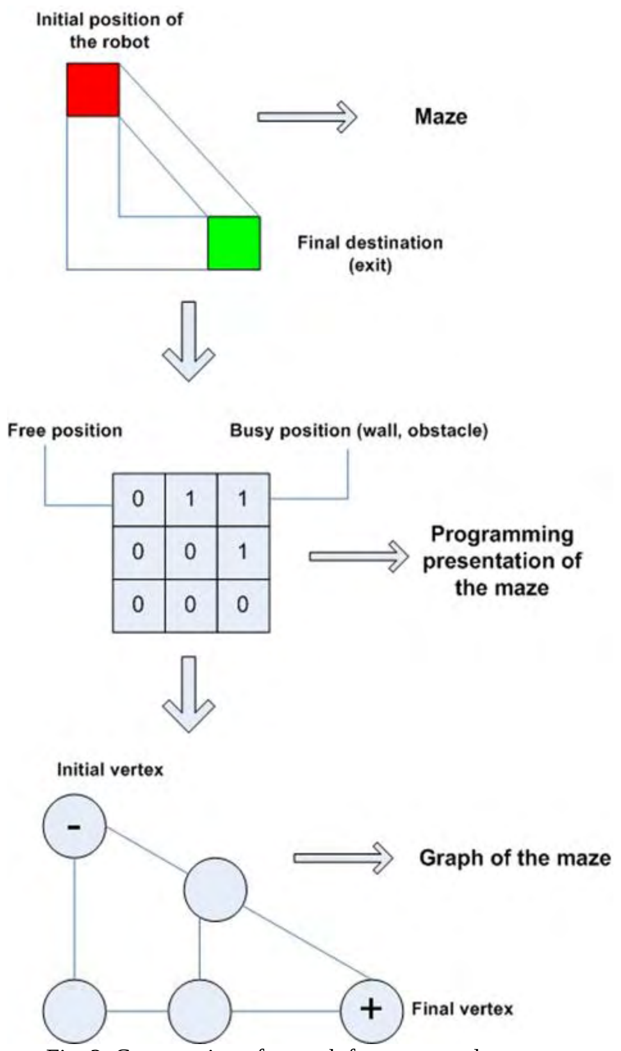


Fig. 2. Construction of a graph for an example maze

The algorithm constructs unweighted, indirect graph, formed by all pixels, which could be a valid position of the robot in the maze. A position is valid, if all pixels up, down, left and right in a distance equal to the robot size are free. Such construction of the graph guarantees, that only the possible routes are presented in the graph. Over this graph various algorithms for path finding could be applied.

The graph construction algorithm uses radius of the circle, described around the robot (Fig. 3). This presentation is used for checking the valid positions of the robot in horizontal and vertical direction.

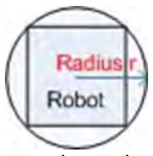


Fig. 3. Robot presentation, using described circle

B. Check for valid position in horizontal direction

The valid positions in horizontal direction are checked according to the radius of the described circle (i.e. according to the robot size) (Fig. 4).

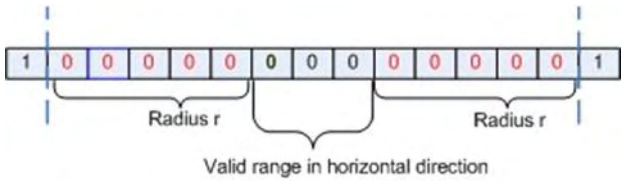


Fig. 4. Valid positions in horizontal direction

If there are no valid positions in horizontal direction, no vertex is added to the graph for these positions (Fig. 5).

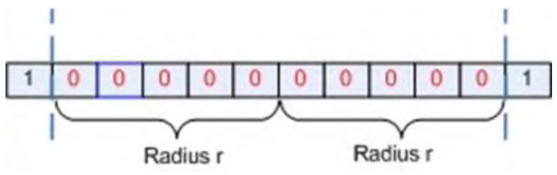
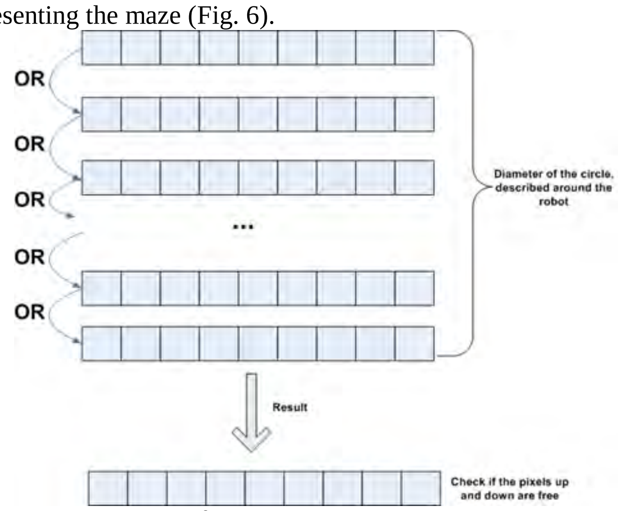


Fig. 5. No valid positions in horizontal direction

C. Check for valid position in vertical direction

The valid positions of the robot in vertical direction are formed by OR operation of each pair of rows from the set, presenting the maze (Fig. 6).



OR
OR
OR
OR
OR
...
Diameter of the circle, described around the robot
Result
Check if the pixels up and down are free

Fig. 6. Check for valid positions in vertical direction

If the OR operation of a pair of neighbor rows results in a character 1, then an obstacle is available on this position and the robot could not be in the position. No vertex is added to the graph for the position (Fig. 7).

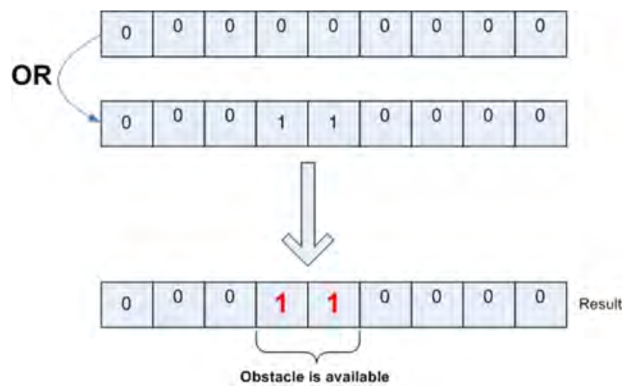


Fig. 7. An obstacle in vertical direction

If the OR operation of a pair of neighbor rows results in characters 0 only, then the positions in vertical direction are free. A vertex is added to the graph (Fig. 8).

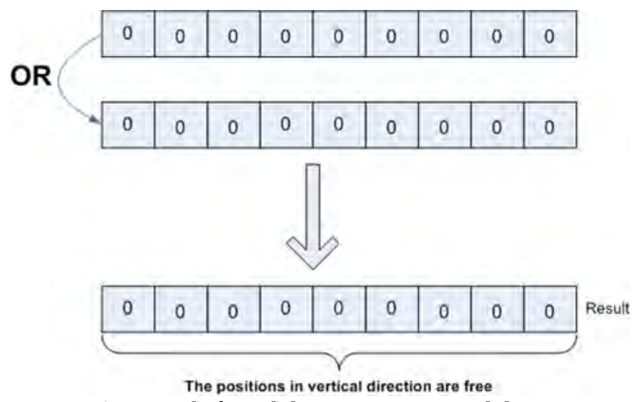


Fig. 8. Example for valid positions in vertical direction

D. Finding the shortest path

If the described graph, containing only the possible positions of movement, is available, an algorithm for finding the shortest path could be applied over the graph.

The algorithm, proposed in this paper, implements breadth first search. The algorithm finds the shortest path between an initial and a final vertex, passing through least number of visited vertices.

The algorithm uses the following data structures:

- Queue of the graph vertices;
- List of the visited vertices;
- List of the possible paths.

First the initial vertex is marked as visited and is added to the queue. Afterwards the neighbor vertices are traversed. If a neighbor vertex is not visited, in the field "previous" of the vertex the number of the previous neighbor is written. The vertex is marked as visited and is added to the queue. The loop continues until the queue is not empty and the final vertex is not visited.

Fig. 9 presents the algorithm execution for an example graph.

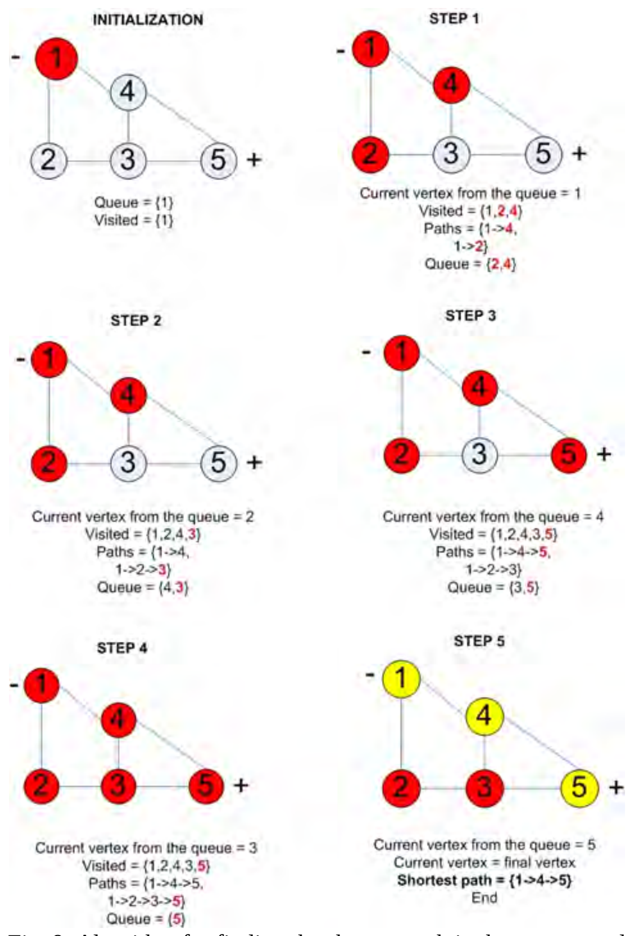

INITIALIZATION
Queue = {1}
Visited = {1}
STEP 1
Current vertex from the queue = 1
Visited = {1,2,4}
Paths = {1->4, 1->2}
Queue = {2,4}
STEP 2
Current vertex from the queue = 2
Visited = {1,2,4,3}
Paths = {1->4, 1->2->3}
Queue = {4,3}
STEP 3
Current vertex from the queue = 4
Visited = {1,2,4,3,5}
Paths = {1->4->5, 1->2->3}
Queue = {3,5}
STEP 4
Current vertex from the queue = 3
Visited = {1,2,4,3,5}
Paths = {1->4->5, 1->2->3->5}
Queue = {5}
STEP 5
Current vertex from the queue = 5
Current vertex = final vertex
Shortest path = {1->4->5}
End

Fig. 9. Algorithm for finding the shortest path in the constructed graph

E. Programming implementation

The most challenging problem in the implementation of the described algorithm is proper presentation of mazes with large dimensions (e.g. 1600x1013 pixels). Such images have to be quickly transformed to a graph, considering the limited memory and computing power in the selected robot platform.

In the implementation of the AlgoHris algorithm the maze is presented as a set of bits. Each bit is a structure of elements with only 2 possible values: 0 (true) or 1 (false). A programming class is realized, emulating an array of Boolean elements. Each element reserves only one bit, which is eight times less than the simple character data type. Each bit could be accessed individually as in ordinary array.

III. Experimental Results

The experiments are carried out in two directions:

- Comparing the times for graph construction for mazes with various dimensions;
- Comparing the algorithm with three other algorithms, using the same dimensions (height and width in pixels) of the maze – Backtracking, A* and genetic algorithm GAPP.

In the first experiment the times for completing the separate steps of the whole algorithm are presented in Table I.

TABLE I
EXPERIMENTAL RESULTS WITH IMAGES OF EXAMPLE MAZES WITH DIFFERENT RESOLUTION

Width of the maze (pixels)	**Height of the maze (pixels)**	**Time for graph construction (ms)**	**Time for shortest path finding (ms)**
480	304	45	5
480	304	45	5
480	304	46	6
1000	820	72	23
1000	820	73	24
1600	1013	105	52

The results show, that the implemented algorithm processes relatively large image (1600x1013 pixels) and finds the shortest path for less than 160 ms.

The data, obtained by the second experiment, are presented on Fig. 10. In the case of small dimensions of the maze there is no significant difference between the times for shortest path finding. The genetic algorithm is comparatively slow and inapplicable to large dimensions of the maze. AlgoHris achieves shorter time than the Backtracking and even than the A* algorithm when increasing the dimensions of the maze.

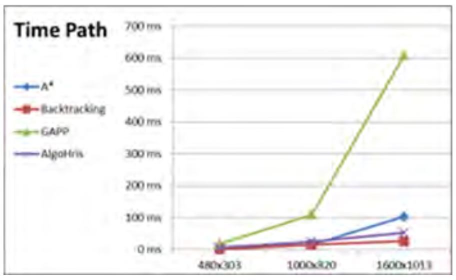


Fig. 10. The shortest path finding time for four different algorithms

On Fig. 11 the approximation function for AlgoHris is shown. The function is

y = 23x – 18.667, R2 = 0.9845

The dependence between the time for path finding and increasing the maze dimensions is proportional and the algorithm is stable.

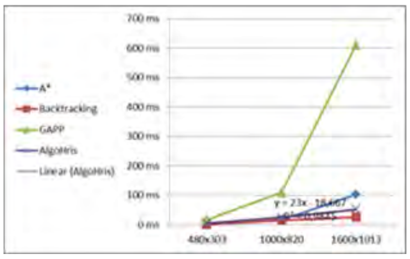


Fig. 11. Trendline correlation for AlgoHris

IV. CONCLUSION

The proposed AlgoHris algorithm for movement of a robot in a maze has the following advantages in comparison to the existing implementations of other algorithms:

- It uses an image of the maze, which makes possible fast processing of the pixels in the image to find a path in the maze;
- The maze is presented as a bit set, which significantly increases the graph construction, containing only the possible paths for movement of the robot.

Although in the current implementation AlgoHris uses the breadth first search method to find the shortest path, other methods for shortest path finding as Dijkstra's algorithm and A* algorithm could be effectively applied over the graph, produced by AlgoHris.

The future work will consider implementation of algorithms for finding paths in mazes, which are not shot in advance. The camera or the sensors of the robot will be used. This way the robot will be able to move in dynamically changing environment.

REFERENCES

[1] Chauhan S., Bajpai A., Collision Free Autonomous Robot Path Planning. *International Journal of Engineering Research & Technology*, Vol. 1, Iss. 8, e-ISSN: 2278-0181, 2012.

[2] Hachour O., The proposed path finding strategy in static unknown environments. *International Journal of Systems Applications, Engineering & Development*, Iss. 4, Vol. 3, 2009.

[3] Hachour O., Path planning of Autonomous Mobile robot. *International Journal of Systems Applications, Engineering & Development*, Iss. 4, Vol. 2, 2008.

[4] Sedgewick R., Wayne K., *Algorithms, 4th Ed*., ISBN-13: 978-0-321-57351-3, Pearson Education, 2011.

[5] Naumov V., Karova M., D. Zhelyazkov, M. Todorova, I. Penev, V. Nikolov, V. Petkov, Robot Path Planning Algorithm, *International Journal of Computers and Communications*, ISSN: 2074-1294, NAUN, 2015.

[6] Correll N., *Introduction to Autonomous Robots*, 1st edition, ISBN-13:978-1493773077, 2014.