# Robot Path Planning Algorithm

Velko Naumov, Milena Karova, Danislav Zhelyazkov, Mariana Todorova, Ivaylo Penev, Ventsislav Nikolov, and Vilian Petkov

***Abstract*—**This paper presents an improvement of a classic Dijkstra algorithm to the domain of sampling based motion. The algorithm uses an image, obtained by a camera. The algorithm processes the image to convert it into a matrix, presenting the labyrinth with obstacles and walls. Afterwards the algorithm finds the shortest path to a final target in the labyrinth. In contrast to the classical Dijkstra's algorithm, the presented algorithm compares the size of the robot to the size of the obstacles on the way. A simulation of the algorithm is developed to visualize the movement of the robot. Experimental results, obtained by the simulation, are presented. The potential of the proposed results is apparent both in terms of reliability and quality of solutions found.

***Keywords*—**Dijkstra algorithm, Labyrinth, Mobile robot, Path planning, Wave moving process.

## I. Introduction

MOTION planning is one of the significant tasks in intelligent control of mobile robots.

Motion planning of the robot is often decomposed into path planning and trajectory planning. The aim of the trajectory planning is to schedule the movement of a mobile robot along the planned path [8]. One of the critical problems for the mobile robots is path planning which is still an open one to be studied extensively. Path planning allows robots find the optimal path between two points.

Path planning research covers a wide area of robotics research because it enhances robotic navigation systems in both static and dynamic environments. With the perfect path planning system, mobile robots can navigate by itself without human intervention to reach the targeted destination [3].

The basic steps in path planning are:

The work presented in this paper was supported within the Startup Scientific Project № 1, 2015, Technical University of Varna, "Research and development of algorithms for control of mobile robots under extreme conditions in virtual reality".

V. Naumov is with the Department of Automation of Manufacturing, Technical University of Varna, Bulgaria, e-mail: velko.naumov@abv.bg

M. Karova is with the Department of Computer Science and Technology, Technical University of Varna, Bulgaria, e-mail: mkarova@ieee.bg

D. Zhelyazkov is with the Department of Computer Science and Technology, Technical University of Varna, Bulgaria, e-mail: d.zhelyazkov.7331@gmail.com

M. Todorova is with the Department of Automation of Manufacturing, Technical University of Varna, Bulgaria, e-mail: mgtodorova@tu-varna.bg

I. Penev is with the Department of Computer Science and Technology, Technical University of Varna, Bulgaria, e-mail: ivailo.penev@tu-varna.bg

V. Nikolov is with the Department of Computer Science and Technology, Technical University of Varna, Bulgaria, e-mail: v.nikolov@tu-varna.bg

V. Petkov is with the Department of Automation of Manufacturing, Technical University of Varna, Bulgaria, e-mail: v.petkov@tu-varna.bg

- First step. Choosing a map representation that is appropriate to the application;
- Second step. Reducing the robot to a point-mass, which allows planning in the configuration space.

This allows the application of generic shortest path finding algorithms, which have applications in a large variety of domains. Algorithms to find a shortest path are important not only in robotics, but also in network routing, video games and gene sequencing.

Therefore a big number of algorithms and methods have been researched for finding of the shortest path between the start and the goal points. The approaches can broadly be categorized into on-line and off-line techniques. Some of the commonly used algorithms are: Djikstra algorithm, A* algorithm [7], wavefront-based planners, breadth-first search (BFS), depth-first search (DFS), rapidly exploring random trees and etc [3], [5]. They are shortly summarized below.

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 have been checked, the algorithm proceeds to the vertex with the next lowest cost [6].

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 is 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.

An extension of A* that addresses the problem of expensive re-planning when obstacles appear in the path of the robot, is known as D*. Unlike A*, D* starts from the goal vertex and has the ability to change the costs of parts of the path that include an obstacle. This allows D* to re-plan around an obstacle while maintaining most of the already calculated path. A* and D* are computationally complex when the search space is large.

The wavefront propagation algorithm has emerged as the dominant method for path planning in discrete grid maps [2]. The strategy is based on the propagation of wavefronts that encode the distance from the robot's current location to any point in its environment. The wavefronts propagate from the

source located on the centre right of the map. Each wavefront generated is designated a higher value than the previous. The shortest path can be determined by selecting any point on the map and then tracing the highest descent of wavefronts back to the source.

BFS is a systematic way of searching a graph: that is, visiting every node. The root of the BFS search tree is the node in the graph at which starts the search. The fringe is the list, initially empty, of all nodes queued for expanding or examining. The visited list is the list, initially empty, of all nodes already visited, which prevents the search from going in circles.

DFS and BFS are very closely related to each other. The big advantage of DFS is that it has much lower memory requirements than BFS, because it is not necessary to store all of the child pointers at each level. The decision to choose one over the other should be based on the type of data and what you are looking for.

A more recent development known as Rapidly-Exploring Random Trees (RRT) addresses this problem by using a randomized approach that aims at quickly exploring a large area of the search space. Although RRT quickly finds some solution, smooth paths usually require additional search algorithms that start from an initial estimate provided by RRT [6].

Great attention has been given to Genetic algorithms [3], [4], [8]. These algorithms are not only capable to find optimal paths that satisfy the optimization criteria but are also adaptable and robust not only in the static but also in the dynamic environments. Compared to traditional approaches, these methods have been proven as robust and effective search techniques that can be used to optimize the robot path planning (RPP) problem.

Path planning in spatial representation often requires the integration of several approaches. This can provide efficient and accurate navigation of a mobile robot.

An algorithm for planning the path of a mobile robot in a labyrinth is presented in this paper. It processes an image, obtained by a camera, to convert it into a matrix, presenting the labyrinth with obstacles and walls. The shortest path to a final target is found using the presented algorithm. It is based on the Dijkstra's algorithm. The difference is that it compares the size of the robot with the size of an obstacle. The presented algorithm is tested in labyrinth with varying sizes. The performed simulations and obtained experimental results are presented and analyzed.

## II. Building Virtual Labyrinth

In the presented algorithm the robot's environment is obtained by a camera (Fig. 1) and has the following properties:

1. every pixel of the image is analyzed and a map is created as a two-dimensional array;
2. every pixel is transformed to a symbol according to Table I;
3. the pixels' coordinates correspond to the symbols positions in the map.

This project is divided into 2 major areas: visual detection of virtual labyrinth and path planning.

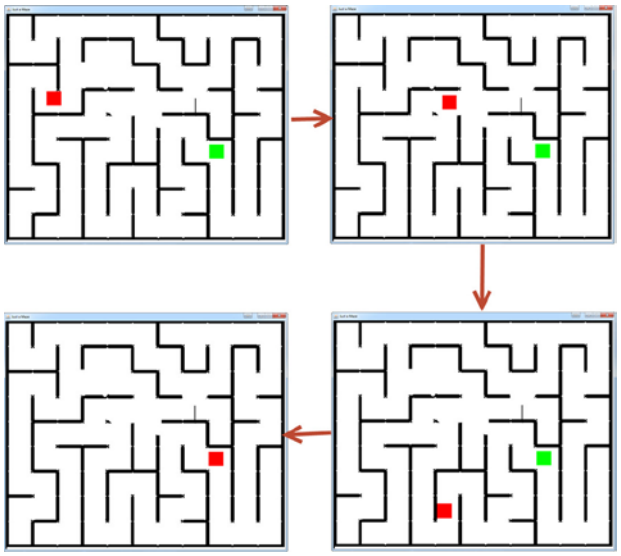

Fig. 1 Labyrinth scenes and robot movement

Table I. Relation between colors and symbols

| RGB | color | meaning | symbol |
|---|---|---|---|
| >200,>200,>200 | light | space | ' ' |
| >200,<100,<100 | nuance red | start / robot | '*' |
| <100,>200,<100 | nuance green | end / exit | 'o' |
| other | other | obstacle / wall | 'W' |

When green or red pixel occurred some additional computations are made to find the minimal and maximal values of coordinates of the robot position and the labyrinth exit (Eq.1).

$$(x_{rmin}, y_{rmin} : x_{rmax}, y_{rmax});(x_{emin}, y_{emin} : x_{emax}, y_{emax}) \quad (1)$$

The width of the exit and robot are also calculated (Eq.2).

$$k = max((x_{rmax}-x_{rmin}), (y_{rmax}-y_{rmin}), (x_{emax}-x_{emin}),(y_{emax}-y_{emin})) \quad (2)$$

Thus the algorithm guarantees that the robot will go through wide enough paths.

The previous data is integrated within the programming model of the labyrinth. It is presented as a global object, called data transfer object (DTO), accessible from any other part of the application including all interface implementations. This global object is a pure data object without any functionalities and it is able to self-validate and convert itself to plain text. The validation aims to refuse invalid or incorrect pictures.

The virtual labyrinth can be constructed either by analyzing a picture or by a scanning stream. The virtual labyrinth after its processing is shown in Fig. 2.

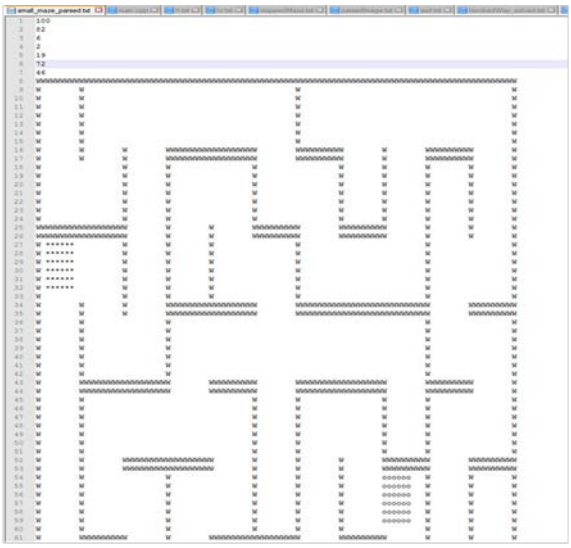

Fig. 2 Virtual labyrinth shown as specially ordered characters

## III. Optimal Path Planning Algorithm

The potential path is formed on version of Dijkstra's algorithm. The difference is, that the presented algorithm compares the size of the robot with the size of an obstacle.

A wave starting from the end of the labyrinth is observed as a final unit that is moving from one point to a next neighbor point. The wave gradually marks all points (units) directed to the final point – Fig.3. This idea is further developed in the application and is called Gasolisation. The difference is that in our case the robot, respectively the final, are with different width compared to the width of the walls and paths. The marking points are replaced with marking lines (sequences of points) with length k. The current traversing line in fact does not search neighbor points but the whole neighbor lines. If the robot is found the main wave stops it's spreading and a new small wave starts to spread trying to mark the robot. If this does not succeed then this means that the area locating the robot is too narrow and then the small wave stops and the main wave continues. Otherwise if the robot is successfully marked the algorithm is completed. If the main wave cannot continue, because the all lines are marked, then an exception is thrown saying that a path is not found. The algorithm works on the characters file, that represents the virtual labyrinth, and the marking is done by using the symbols '^', '>', 'v' and '<'.

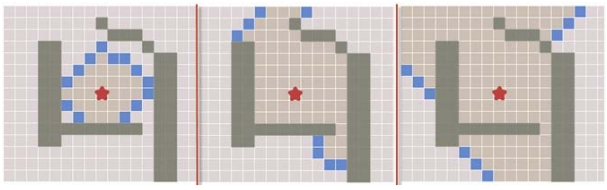

Fig. 3 The wave moving

The algorithm consists of the following basic steps:

- Creation of a queue – a set of ordered points;
- Extraction of elements from the queue;

Checking of all neighbor points for each point. The possible neighbors are in four directions – up, right, down, left.

A neighbor point is free, if the following conditions are satisfied:

- the point is not a part of an obstacle;
- the point is not marked.

A free point is marked and added to the queue.

All free points are marked by its neighbors. The order of marking forms the shortest path.

The whole algorithm is as follows (Fig. 4):

```
void FindPath(Dot startDot, Dot finalDot) {
  Queue<Dot> justAQueue;
  justAQueue.Add(finalDot);
  while (Dot currentDot == justAQueue.Pop()) {
    Array<Dot> neighbours = currentDot.GetNeighbours();
    foreach (Dot neighbour in neighbours) {
    if(neighbor.isObstacle || neighbor.isMarked)
      next;
      currentDot.Mark(neighbor);
      if (neighbor == startDot)
        return;
       justAQueue.Add(neighbor);
    }
  }
  throw new NoPathFoundException();
}
```


Fig. 4 Path-finding algorithm

## IV. Simulation Studies and Results

For the simulation of the algorithm an application is created. The input of the application is an image of a labyrinth. The application implements the algorithm to convert the image into a text format and to move an object from an initial position to a final target (exit of the labyrinth).

The algorithm is tested by labyrinths with varying sizes

(different width and height in pixels). Examples of two labyrinths are shown in Fig. 5 and Fig. 6.

The following times are measured for each labyrinth:

- time for labyrinth construction (i.e. converting the image into text format, suitable for processing);
- time for obtaining a solution (i.e. finding a path to the target);
- time for the movement of the robot to reach the target.

The results are summarized in Table II.

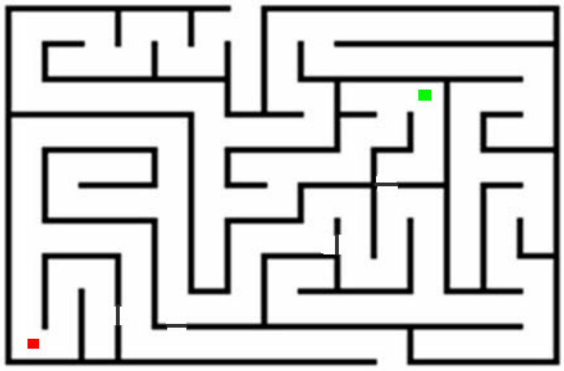

Fig. 5 480x304 pixels labyrinth

Result: The algorithm solves the planning problem (converts the image, finds a path and moves the object to the exit) for 0.2 seconds.

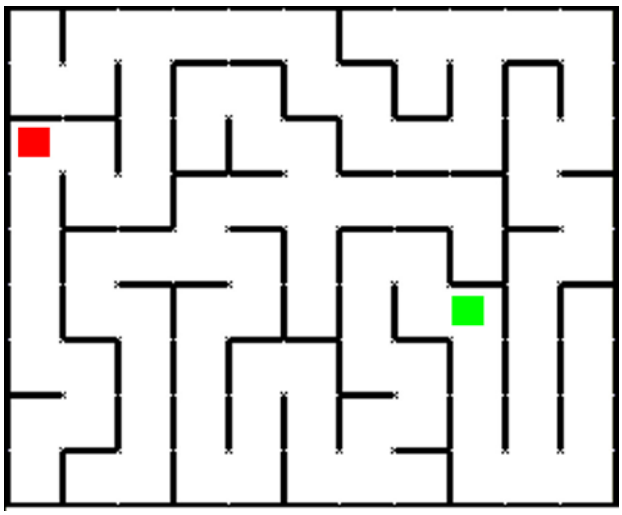

Fig. 6 1000 x 820 pixels labyrinth

Result: The algorithm solves the planning problem for 0.3 seconds.

Table II. Results from tests of the algorithm with different labyrinths

| Width of the labyrinth (pixels) | Height of the labyrinth (pixels) | Labyrinth construction time (ms) | Time for finding a path (ms) | Whole process time (ms) |
|---|---|---|---|---|
| 100 | 82 | 39 | 12 | 63 |
| 480 | 304 | 110 | 68 | 187 |
| 480 | 304 | 129 | 62 | 201 |
| 480 | 304 | 113 | 105 | 243 |
| 1000 | 820 | 171 | 106 | 300 |
| 1000 | 820 | 187 | 107 | 305 |
| 1600 | 1013 | 304 | 140 | 460 |

## V. Conclusion

The results show, that the algorithm is able to move an object in a labyrinth with large size (1600 x 1013 pixels) for less than 500 ms. Such an image could be obtained by an usual camera (for example the camera of a phone or a tablet).

There are at least two tasks to be performed in the future:

(1)The algorithm should be tested in real circumstances for a real mobile robot to verify the applicability of the proposed method.

(2)The effectiveness and feasibility of the algorithm should be testified by different parameters: labyrinth types, path distance, search speed of the optimal path.

(3)The proposed method can be transformed to dynamic path planning method under an unknown environment.

A significant application of the presented algorithm exists in the field of technology education. The algorithm has been implemented in a real robot platform (in our case LEGO EV3 robot). This way we could demonstrate to students fundamental concepts in computing and automation: path-finding and search algorithms, robot programming, device motion control, finite state machines, others. Furthermore the algorithm is a suitable basis for comparison of different path finding algorithms, for example breadth-first and depth first search with A* algorithm.

## References

[1] A. Rodic, Navigation, “Motion Planning and Control of Autonomous Wheeled Mobile Robots in Labyrinth Type Scenarios”, Volume 8, Number 2, Intelligent Service Robotic Systems, IPSI Journal, Transactions on Internet Research, TIR, ISSN 1820 - 4503, 2012, pp. 2-9.

[2] A. Al-Jumaily, and C. Leung, “Wavefront Propagation and Fuzzy Based Autonomous Navigation”, International Journal of Advanced Robotic Systems, vol. 2, Number 2, ISSN 1729-8806, 2005, pp.093-102.

[3] N. Buniyamin, N. Sariff, W. A. J. Wan Ngah, and Z. Mohamad, “Robot global path planning overview and a variation of ant colony system algorithm”, International Journal Of Mathematics And Computers In Simulation, issue 1, vol. 5, 2011, pp. 9 – 16.

[4] J. Su, and J. Li, “Path Planning for Mobile Robots Based on Genetic Algorithms”, Proceedings of Ninth International Conference on Natural Computation (ICNC), ISBN: 978-1-4673-4714-3, 2013, pp. 723-727.

[5] M. A. H. Ali, M. Mailah, and T. H. Hing, “Path Planning of Mobile Robot for Autonomous Navigation of Road Roundabout Intersection”, International Journal Of Mechanics, issue 4, vol. 6, 2012, pp. 203 – 211.

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

[7] N. Sariff, and N. Buniyamin, “An Overview of Autonomous Mobile Robot Path Planning Algorithms”, Proceedings of 4th Student Conference on Research and Development, ISBN: 1-4244-0527-0, 2006, pp. 183-188.

[8] O. Hachour, Path planning of Autonomous Mobile robot, International Journal of Systems Applications, Engineering & Development iss. 4, vol. 2, 2008.

[9] S. Muldoon, L. Chaomin, F. Shen, and H. Mo, Naturally Inspired Optimization Algorithms as Applied to Mobile Robotic Path Planning, IEEE Symposium on Swarm Intelligence, ISBN: 978-1-4799-4458-3, 1994, pp. 1-6.

[10] http://www.redblobgames.com/pathfinding/a-star/introduction.html.