Path Planning Algorithm for a Robot in a Labyrinth

Milena Karova[1], Ivaylo Penev[2], Ventsislav Nikolov[3] and Danislav Zhelyazkov[4]

Abstract – **The paper presents an algorithm for path planning for a robot in a labyrinth. The algorithm uses an image, obtained by a camera. The image is processed and converted to a matrix, presenting the labyrinth with obstacles and walls. Afterwards an algorithm, based on the Dijkstra's algorithm, is applied to find the shortest path in the labyrinth. As opposed to the classical Dijkstra's algorithm, the presented algorithm compares the size of the robot to the size of an obstacle. The implementation of the algorithm is described and the obtained results are presented.**

Keywords – **Path planning, Robot, Labyrinth, Dijkstra algorithm, Wave moving process.**

I. Introduction

In the past the orientation of a human which is in a new unknown place has been difficult. That is why the researchers have spent some time for walking, orientation and map creating for the unknown areas. Thus the geographic maps emerged to facilitate the orientation in unknown location. Later the orbital satellites have been invented and used to take pictures and automatically map the ground surface.

After the invention of computers, information technologies and programming tools the humans are trying to make a perfect electronic anthropoid called robot. Artificial intelligence is intended to be introduced in it to make it like a human. In order such intelligence to be qualitative and reliable it should be able to produce at least some thinking operations and, for example, orientation.

Many algorithms and methods have been studied for planning path of robots [5]. Great attention has been given to Genetic algorithms [2], A* algorithm [3], as well as other naturally inspired optimization algorithms [4].

In most cases the map of the environment (with obstacles and possible paths) is obtained through the sensors of the robot [1]. This approach serves well for movement of robots on terrestrial surfaces, but could be inapplicable if the robot's sensors could not be used or sensors usage is difficult. For example this is the case with movement in air or underwater areas.

This paper describes an application of a project in which a robot, labyrinth and camera are used. The camera shots and sends the pictures to the robot which is moving through the labyrinth. If the labyrinth is changed the camera informs the robot. Thus the following operations should be implemented: analyzing of the labyrinth image, finding of the optimal path, making of the path map, moving simulation. The main goal is to find the shortest path in a labyrinth given by a picture. The walls, obstructions, the robot and the exit of the labyrinth are the input data and should be marked. The output of the system is the labyrinth printed in a file and animated simulation of the robot moving.

The application goes in processing mode when a picture is selected.

[1]Milena Karova is with the Department of Computer Science at Technical University of Varna, Bulgaria, E-mail:mkarova@ieee.bg
[2]Ivaylo Penev is with the Department of Computer Science at Technical University of Varna, Bulgaria, E-mail: ivailo.penev@tu-varna.bg
[3]Ventsislav Nikolov is with the Department of Computer Science at Technical University of Varna, Bulgaria, E-mail: v.nikolov@tu-varna.bg
[4]Danislav Zhelyazkov is a student at the Department of Computer Science at Technical University of Varna, Bulgaria, E-mail: d.zhelyazkov.7331@gmail.com

II. Processing Mode

A. Image processing and building of virtual labyrinth

A picture of the labyrinth is obtained by a camera, attached to the robot. Additionally on the picture the initial position of the robot is marked by a red square and the exit of the labyrinth is marked by a green square (fig. 1).

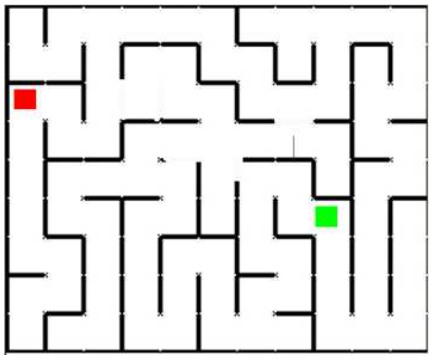

Fig.1 Picture of a labyrinth

Table I
Symbol according

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'

In this stage every pixel of the image is analyzed and a map is created as a two-dimensional array. Every pixel is transformed to a symbol according to Table 1. The pixels' coordinates correspond to the symbols positions in the map.

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, i.e.:
$(Xr_{min}, Yr_{min} : Xr_{max}, Yr_{max});(Xe_{min}, Ye_{min} : Xe_{max}, Ye_{max})$.

The width of the exit and robot are also calculated:
$k=max((Xr_{max}-Xr_{min}),(Yr_{max}-Yr_{min}),(Xe_{max}-Xe_{min}),(Ye_{max}-Ye_{min}))$.

B. Optimal path in the virtual labyrinth

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.

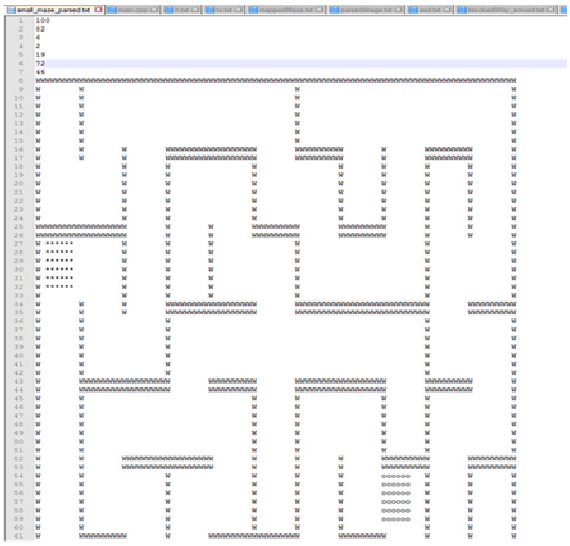

Fig.2 Virtual labyrinth shown as specially ordered symbols

III. FINDING THE OPTIMAL PATH IN THE LABYRINTH

This is the main step in the application logic. Finding the path is based on the well-known Dijkstra 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 '<'. The robot follows these symbols to move to the exit, i.e. the symbols mark the back trace for the robot.

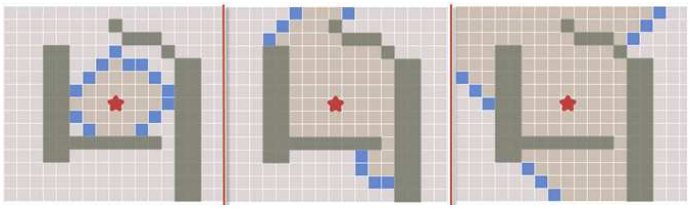

Fig. 3 The wave moving

The algorithm consists of the following basic steps:

- Creation of a queue – a set of points;
- Extraction of elements from the queue;
- For each point all neighbor points are checked. 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:

```
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();
}
```

IV. EXPERIMENTS - ANIMATION WITH SIMULATED MOVING

The application prints as a text file the marked virtual labyrinth. The file contains information for the size of the labyrinth and the robot as well as the coordinates of the start and end points. The walls and obstructions are shown as 'W' character, the path to the final is shown with the symbols '^', '>', 'v', '<' and the final is shown as 'o'. This file can be exported and easily used by any other software application.

The application is implemented using the Java Swing technology for user interfaces. When the process is completed a maze monitor is shown as a panel in the main user window sharing the virtual labyrinth. This monitor is responsible for the reverse transformation of the characters map to a graphical picture. A maze command executor is also used to move the robot according to the markers over which it is currently placed. If there are not markers in the robot position then this means that the target is reached or there is no path in the labyrinth. In these cases the robot cannot move any more. The executor moves the robot by a timer until a situation with an impossible move is reached (Fig. 4).

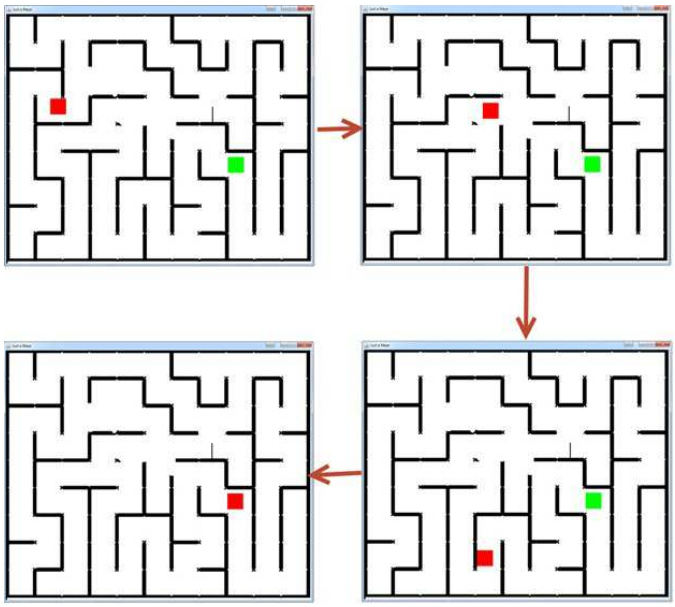

Fig. 4. Animated simulation of the robot movement

V. RESULTS

File with a model of the labyrinth with walls and obstacles and the paths found is obtained as a result of the algorithm (Fig. 5).

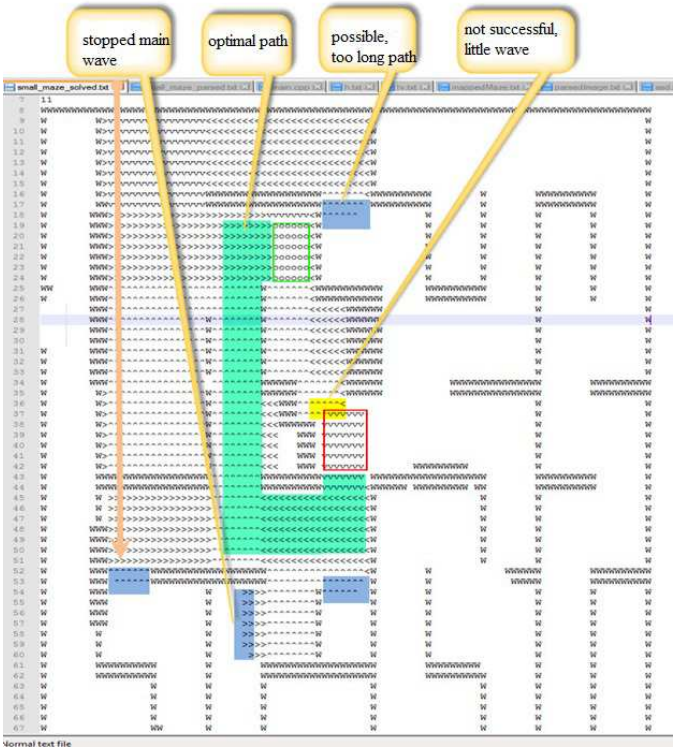

Fig. 5. Model of the labyrinth with found paths

The output file has the following structure:

- Numeric data about:
  - size of the labyrinth;
  - size of the robot and the exit of the labyrinth;
  - coordinates of the start and the exit.
- Symbol map with the following data:
  - walls and obstacles – symbol 'w' is used;
  - path to the exit – symbols '^' , '>' , 'v' , '<' are used.
  - exit – symbol 'o'.

The output, shown on Fig. 5, presents:

- several thin slices;
- unsuccessful little wave;
- successful stop of the wave;
- several possible paths.

A file with this structure could be used by other applications, using the same programming interfaces.

VI. CONCLUSION

For the present we consider significant application of the presented algorithm 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 demonstrated 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.

One of the most important optimizations of the proposed algorithm should be its parallel realization. This could also make possible dynamic changes like manual choice of the robot target and manual change of the labyrinth. Another future task is implementation of another searching 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, pp. 2-9, 2012.

[2] J. Su, 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, pp. 723-727, 2013.

[3] N. Sariff, 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, pp. 183-188, 2006.

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

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