ICSE 2010

PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON SCIENCE AND ENGINEERING

Volume - 1

Electronics
Electrical Power
Information Technology
Engg: Physics

Sedona Hotel, Yangon, Myanmar
December 2-3, 2010

PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON SCIENCE AND ENGINEERING

Organized by
Ministry of Science and Technology

DECEMBER 2-3, 2010

SEDONA HOTEL, YANGON, MYANMAR

Model Based Path Finding Algorithm for Intelligent Mobile Robot

Kyawt Kyawt Yee[#1], Zaw Min Aung[#2], Hla Myo Tun[#3]

[#]*Department of Electronic Engineering, Mandalay Technological University*
Mandalay, Myanmar

[1]kkyece@gmail.com

[3]hlamyotun.mtu@gmail.com

[*] *Department of Technical and Vocational Education, Ministry of Science and Technology*
Myanmar

[2]zawminaung@gmail.com

***Abstract*— In this paper, a model-based path finding algorithm is presented. The proposed algorithm aims to provide a most reasonable trajectory for an intelligent mobile robot in dangerous area where human cannot enter. This algorithm allows the robot to navigate through static obstacles, and finding the optimum shortest path in order to get the target without collision in the predefined map. The path finding strategy is designed with A* search algorithm. The shortest path result is shown using MATLAB Simulation, Version 7.7.0471 (2008b).**

***Keywords*— Model-based, Path Finding, Mobile Vehicle, Optimum Shortest Path, A* Search Algorithm**

I. INTRODUCTION

Autonomous robots which work without human operators are required in many applications. In order to achieve tasks, autonomous robots have to be intelligent and should decide their own action. When the autonomous robot decides its action, it is necessary to plan optimally depending on their tasks. More, it is necessary to find a collision free path minimizing a cost such as time, energy and distance. When an autonomous robot moves from a point to a target point in its given environment, it is necessary to find an optimal or feasible path avoiding obstacles in its way. One of the main areas of research, in order to achieve successful autonomous robots, is path finding. If the environment is known static terrain and it generates a path in advance it said to be off-line algorithm called model-based algorithm. It is said to be on-line if it is capable of producing a new path in response to environmental changes [1] [2] [6].

II. THE LITERATURE OF THE PATH FINDING ALGORITHM

Path finding generally refers to the plotting, by a computer application, of the shortest route between two points. The path finding problem comes down to graph search problem ultimately. Blind search and heuristic search is the two mainstream categories of Graph search .In order to find the target, the methods of blind search are by means of exhaustive search in a specific sequence without considering the specific knowledge of the issue. The most typical way is the Dijkstra algorithm [1], which is used to find single source shortest path (one point to multipoint of the shortest distance). Its principle is repeatedly examines the closest not-yet-expanded vertex, adding its vertices to a set of vertices to be visited and expands outwards from the starting point to the goal. However, blind searching is inefficient, high space-time complexity, tending to be trapped in combinatorial explosion, as a result the heuristic search has gradually become the mainstream method. Heuristic method use heuristic information to select the most promising node from current set to expand the path, which greatly improves the search efficiency and intelligence. Currently, A * algorithm [5] is the most widely used heuristic search algorithm. It not only has the benefits of Dijkstra algorithm, like completeness and admissibility but also has the merits of BFS (Best-First-Search) algorithm [3], for example, high efficiency, low space-time complexity, shortest distance and so on. However, when facing the specific areas and various forms of the complex map, A * algorithm also exposes many deficiencies [4].

III. A STAR (A*) SEARCH PATH FINDING ALGORITHM

Navigating a terrain and finding the shortest path to a target location is one of the fundamental problems in path finding. While there are many approaches to this problem, one of the most common and widely known is the A star search.

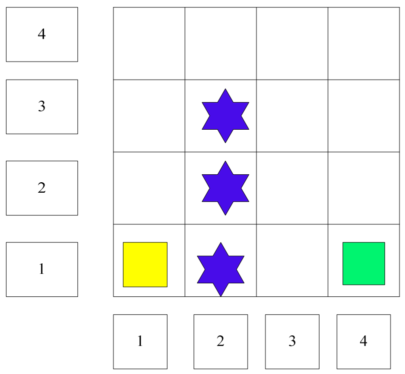

Fig. 1 An example of terrain before finding the path

The robot (yellow square) has to traverse the terrain and reach the target (the green square). However this method is computationally intensive and does not always guarantee the best path to the target. Give some information regarding the location of the target to make an educated guess. For example if the robot knows its target lies to the east, explore squares to the east of your current location.

The A* uses the distance between the current location and the target and moves to the square that has the smallest distance. It evaluates squares (henceforth called a .node.) by combining h(n), the distance(cost) to that node and g(n), the distance(cost) to get from that node to the goal node. The total cost f (n) = g (n) +h (n) is calculated for each successor node and the node with the smallest cost f (n) is selected as a successor.

1) Determining the Cost of the shortest path

The distance between two nodes is simply determined by calculating the straight distance between the two nodes. Though this might not the true distance (given the fact that there are obstacles that need to be circumnavigated), it never overestimates the actual distance. It can be shown that as long as cost is never overestimated, the algorithm is admissible i.e. it generates the optimal path. Let us consider the case for a simple 4X4 Matrix.

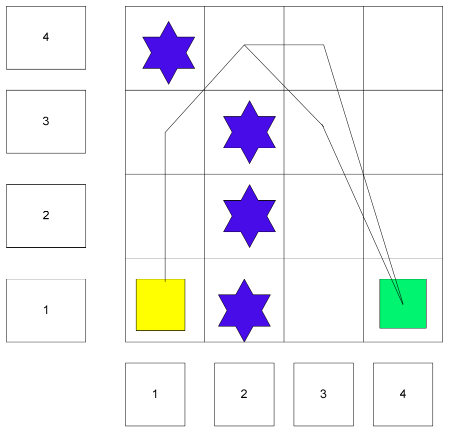

Fig. 2 Finding the shortest path using A*

The start position is (1, 1). The successive node (only one in this case is (1, 2). There is no ambiguity, until the Robot reaches node (2, 4). Here there are two nodes (3, 4) and (3, 3). The successor node can be determined by evaluating the cost to the target from both the nodes.

F (n) for node (3, 3)

h (2,1) = 1 + 1 + sqrt(2)

{Distance from N (1, 1) -> N (1, 2) -> N (1, 3) ->N (2, 1)}

f (n) = h (2, 1) + 1.414 (assuming each square is 1X1 units)

g (n) = sqrt ((4 - 3) ^2 + (1-3)^2) = 2.23

f (n)= 3.44+H(2,1)

f (n) for node (3,4)

h (n)= h(2,1) + 1

g(n)=sqrt ((4-3)^2 + (1-4)^2) = 4.16

f(n) = 4.16 + h(2,1).

Now f(n) for (3,3) has been found to be the smallest of the two, hence the successor node is f(n).The robot can now move to the node (3, 3) and continue expanding the successor nodes as above, until the target node is reached.

2) Dead End of the terrain

This is done by maintaining two lists OPEN and CLOSED. The list OPEN stores all successive paths that are yet to be explored while list CLOSED stores all paths that have been explored. The list OPEN also stores the parent node of each node. This is used at the end to trace the path from the target to the start position, thus generating the optimal route. Consider the figure below. The start node has 2 successors (2, 1) and (1, 2). From the initial calculation (2, 1) is chosen and the robot travels along that node, however ones it reaches the dead end, it discards the node (2, 1) and takes the second successor (1, 2) and explores that route. Once the target node is reached the parent nodes are found and tracked back to the start node to get the complete path.

In the above example N(4,3) -> N(3,4)->N(2,3)->N(1,2)->N(1,1) gives the shortest path.

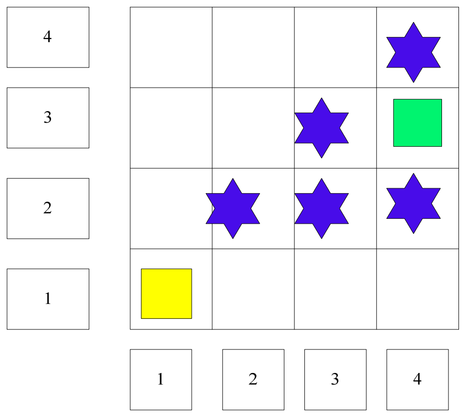

Fig. 3 An example of closed terrain

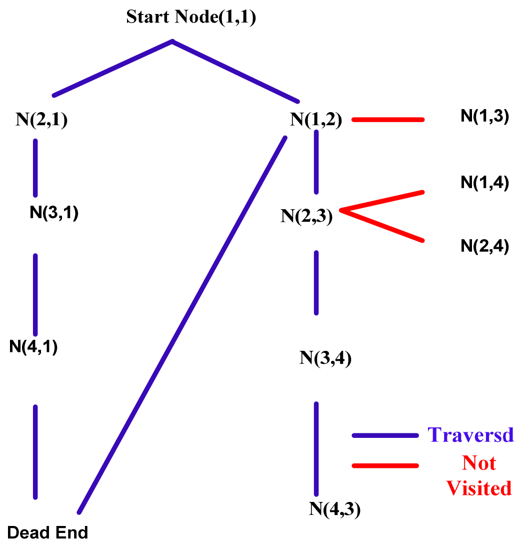

Fig. 4 the selection nodes for the robot tracking

IV. Simulation Results

This paper is to analyze the optimal shortest path to get the necessary target.The algorithm is tested on the simulator, which is developed from MATLAB source code where by the environment is studied in a two dimensional coordinate system. The general flow chart using A* algorithm for the required shortest path is shown in Fig. 5.

There is a predefined map with area in this paper. The area block is defined as x/y dimension. The starting point is x=1, y=1.The target point is x=10, y=10.The algorithm is A* finding the shortest path between the two nodes. The start point is yellow triangle, blue star for obstacles, the green square for the target point and the yellow square is the robot .The map used to test before algorithm start is shown in Fig. 6.In this map the number of nodes are 100 nodes and the number of these nodes to get the shortest path are 13 nodes. These result nodes are shown in table 1.There are 41 obstacles in this predefined map. From this map, there are at least five paths to get the target point. The best result after using algorithm is shown in Fig. 7.The shortest path from the start to the target is described using blue line.

1). Algorithm

The algorithm consists of the following steps.

1) Put the start node on the list OPEN and calculate the cost function f (n). {h (n) = 0; g(n) = distance between the goal and the start position, f(n) = g(n).}

2) Remove from the List OPEN the node with the smallest cost function and put it on CLOSED. This is the node n. (Incase two or more nodes have the cost function, arbitrarily resolve ties. If one of the nodes is the goal node, then select the goal node)

3) If n is the goal node then terminate the algorithm and use the pointers to obtain the solution path. Otherwise, continue.

4) Determine all the successor nodes of n and compute the cost function for each successor not on list CLOSED.

5) Associate with each successor not on list OPEN or CLOSED the cost calculated and put these on the list OPEN, placing pointers to n (n is the parent node).

6) Associate with any successors already on OPEN the smaller of the cost values just calculated and the previous cost value. (min (new f(n), old f(n)))

7) Go to step 2.

2). MATLAB Program

The MATLAB Program consists of the following files.

a) Expan _ array. m: This function takes a node and returns the expanded list of successors, with the calculated fn values. One of the criteria's being none of the successors is on the CLOSED list.

b) Insert_open.m: This function populates the list OPEN with values that have been passed to it. The arguments are xval,yval,parent_xval,parent_yval,hn,gn,fn .

c) Min_fn.m: This function takes the list OPEN as one of its arguments and returns the node with the smallest cost function.

d) Node_index.m: This function returns the index of the location of a node in the list OPEN.

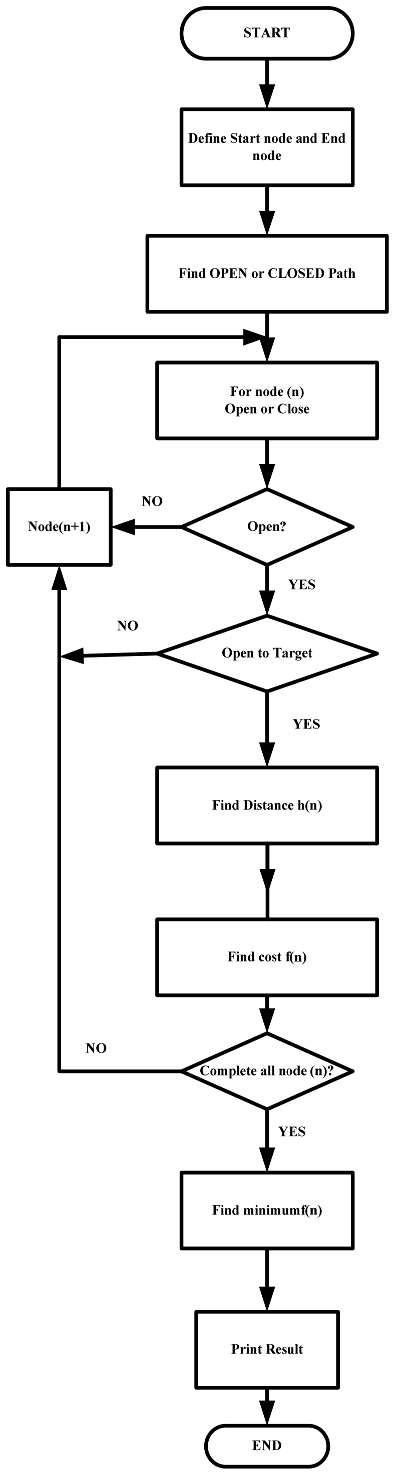

Fig. 5 The proposed flow chart for the shortest path algorithm

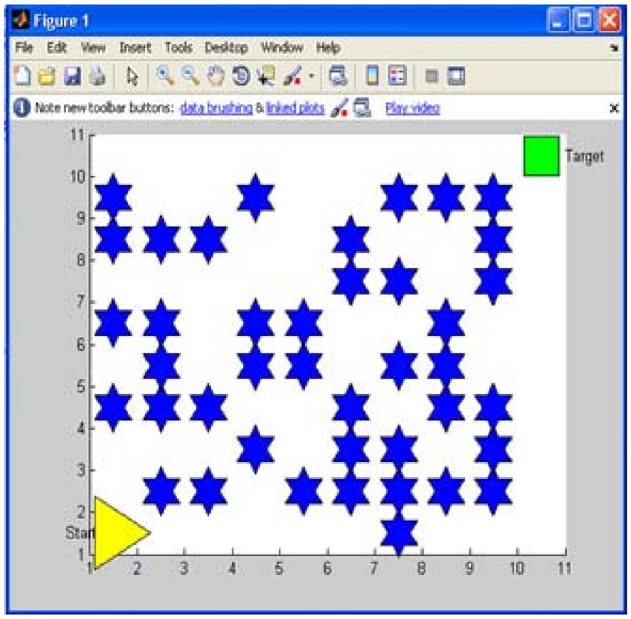
Figure 1
File Edit View Insert Tools Desktop Window Help
Note new toolbar buttons: data brushing & linked plots Play video
Target
Start
1 2 3 4 5 6 7 8 9 10 11

▷ Start Point

✶ Obstacle

🟩 Target

🟨 Robot

Fig. 6 The predefined map used to test before algorithm start

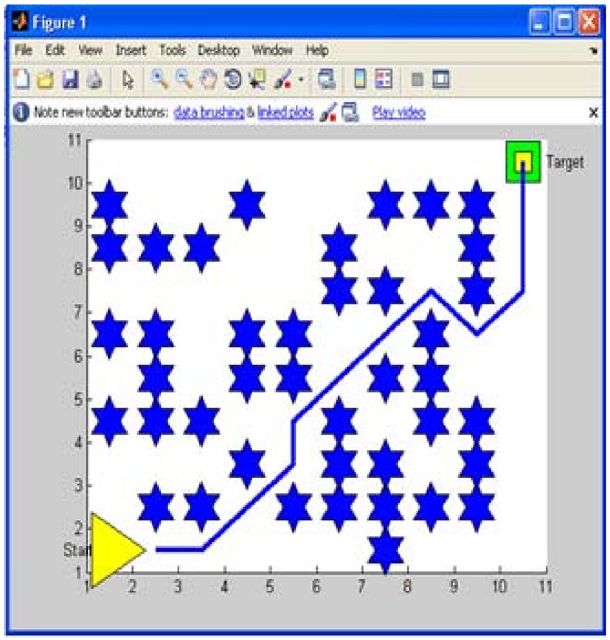
Figure 1
File Edit View Insert Tools Desktop Window Help
Note new toolbar buttons: data brushing & linked plots Play video
Target
Start
1 2 3 4 5 6 7 8 9 10 11

▷ Start Point

✶ Obstacle

🟩 Target

🟨 Robot

Fig. 7 The simulation shortest path result using A* algorithm

TABLE I
THE SELECTED NODES FOR THE OPTIMUM SHORTEST PATH

Node No.	Node coordinate
Start	(1,1)
2	(2,1)
3	(3,1)
4	(4,2)
5	(5,3)
6	(5,4)
7	(6,5)
8	(7,6)
9	(8,7)
10	(9,6)
11	(10,7)
12	(10,8)
13	(10,9)
Target	(10,10)

V. CONCLUSIONS

In this research, a map is created before finding the path shown in Fig.6 and the shortest path result is illustrated in Fig. 7 using MATLAB simulation program. Path finding is an important problem in many applications. This paper has proposed a path finding algorithm for mine detection robot that is targeted to be used by the locals of mine-infested areas to check for the safety of their surroundings. Several techniques exist for path finding, but the most predominant today are based on Dijkstra's Algorithm. Improvements over this original algorithm are generally in the area of faster search or the ability to reuse previous information. Faster search is mainly achieved by heuristics, as with A*search algorithm. This faster algorithm can reduce the number of nodes required than other algorithm. This can calculate the distance between the current node and the target before the robot reach the necessary target. Therefore A* achieves better performance by using heuristics.Using A* search algorithm, the best shortest path can get without wasting time and energy. This algorithm is more reliable fort his research work.

ACKNOWLEDGMENT

The author is very grateful to her supervisor Dr. Zaw Min Aung, Director General, Department of Technical and Vocational Education, for his guidance, permission and helpful suggestion to complete on this paper. The author also would like to thank Dr. Zaw Min Naing for his valuable suggestions and comments in doing this research. Also the author would like to express thank to Dr. Hla Myo Tun (Head and Associate Professor in Department of Electronic Engineering, Mandalay Technological University) for his patience, kindness and suggestions.Special thanks to my parents and my husband for moral support to finish this paper.

REFERENCES

[1] T.C. Liang, J.S. Liu, G.T. Hung, and Y.Z. Chang, "Practical and flexible path planning for car-like mobile robot using maximal-curvature cubic spiral," *Robotics and Autonomous System*Vol. 52, 2005, pp. 312-335.

[2] K. Konolige, “A gradient method for real-time robot control,” *IEEE/R*SJ Int. Conf. Intelligent Robots and Systems, pp.639-646, 2000.

[3] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings IEEE ICRA*, 1994.

[4] Patrick Lester. A* path_nding for beginners, April 2004. Retrieved May 2005 from

[5] http://www.policyalmanac.org/games/aStarTutorial.htm.

[6] Sanjiv Singh, Reid Simmons, Trey Smith, Anthony (Tony) Stentz, Vandi Verma, Alex Yahja, and Kurt Schwehr. Recent progress in local and global traversability for planetary rovers. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2000*. IEEE, April 2000.