Algorithmic Agent for Effective Mobile Robot Navigation in an Unknown Environment

A. Francy Golda
B.E., Final Year Student
amalagolda@gmail.com

S. Aridha
B.E., Final Year Student
aridhasrini@gmail.com

D. Elakkiya
B.E., Final Year Student
elakkiya.d@gmail.com

Dept. Computer Science & Engineering
Thiagarajar College of Engineering, Madurai 625 015, Tamil Nadu, India

Abstract* - *The proposed algorithm for mobile robot supports for the optimal shortest path navigation in an unknown maze-type environment. The approach to find the shortest path through flood fill algorithm considers only the static maze-type environment. The proposed strategy deals with the dynamicity (moving obstacles) and plans the path accordingly. While exploring the maze by wall following, it considers also the number and the position of walls in-between to reach the goal by taking local optimal decisions.

Keywords- Mobile robot navigation; flood fill; dynamic path planning; maze traversal.

I. INTRODUCTION

Nowadays, mobile robots are applied in military, mining and various other fields. The mobile robots which are highly efficient are used to reduce the risks of humans in mining and so on. Robots need to know about the structure of their environment to be able to fulfill complex tasks [1]. Anyway, in rescue situations resulting from earthquakes, this knowledge is not available a priori. Therefore, the robots have to build their own maps while localizing themselves. The algorithm is intended for use in applications such as search-and-rescue operations and emergency environment monitoring. Developing accurate environment maps is one of the most difficult challenges for autonomous navigation systems. An accurate representation of the environment [3] improves the capabilities of the robot in navigation and dealing with dynamic objects. The pre-requisite of the task is correct map creation while maintaining an accurate estimate of the pose of the moving robot. This is commonly referred to as the Simultaneous Localization and Mapping (SLAM) problem. The inclusion of dynamic objects can cause difficulties to SLAM, and therefore may require additional information to simplify the problem.

The flood fill algorithm proposed by *Bellman* deals with static obstacles in an indoor environment. Our proposed algorithm focuses on the static and dynamic movement of the obstacles in an unknown, indoor environment. The 'unknown' environment is one which the robot doesn't aware of it. The robot explores the environment from source to destination and then traces backwards in the shortest path to the source. When an obstacle is encountered, it takes the next shortest path. Since we deal with the dynamic obstacles, the neighbors of the new and the old positions of the particular obstacle have to be flooded again by using the same flood fill algorithm.

II. FLOOD FILL ALGORITHM

Flood-fill algorithm is based on the modified depth first algorithm. The flood-fill algorithm [2] is used primarily for path planning. It is used to plan an optimal (shortest) path to the nearest unexplored cell in the event that a repeated state is detected and it is also used to plan a path back "home" when the goal is reached. Furthermore, the flood-fill guarantees completeness in that it is exhaustive in determining the existence of a potential solution; however, it can only plan a path within the confines of the "known" environment. The shortest path is found from destination to source. The robot explores the maze from source to destination by wall following with the following steps.

Consider the maze in Fig.1

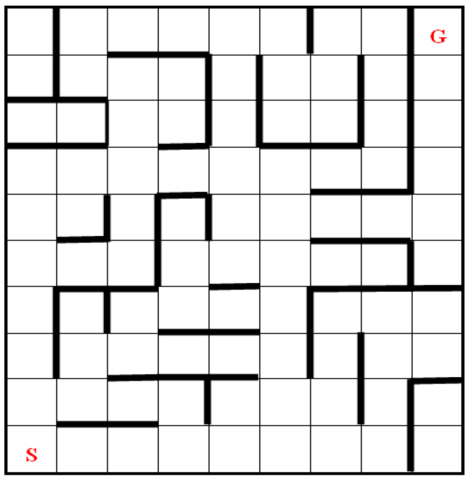

Fig. 1

Step 1: Labeling the grid cells (Fig. 2)

The label of each cell denotes the shortest distance from the source to the particular cell. Initially, each grid cells are labeled with -1. While the robot explores the maze, flooding starts.

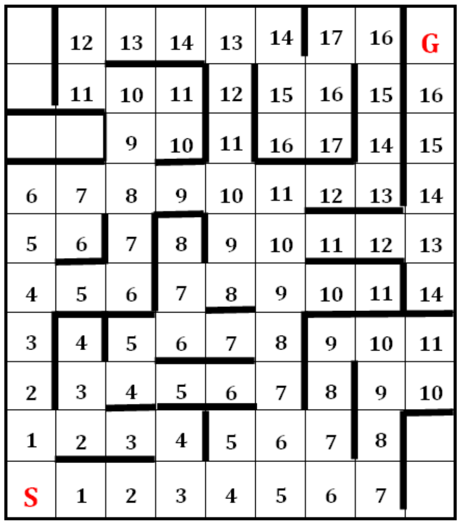

Figure . 2

Source is labeled as zero. Accessible grid cells adjacent to the source are labeled with a distance 1 and the cells that can't be accessed (because of the presence of walls) are left undisturbed (-1) this flooding repeats until we reach the goal. If the sensors find that the three sides are walls, then the agent decide to backtrack the path till it finds the grid point covered by a minimum of 1 wall. This is illustrated as

```
An empty stack S => to hold the neighbor cells
Push source cell onto S
Do
    Cell = pop from S
    If (Cell is the goal) then
        Empty the stack
        Exit => Goal is reached
    Else If ((Cell is not yet visited) and
                (Cell can be accessible)) then
        Mark Cell as visited and Label the Cell
        Push the 4 neighbors of Cell to S
    End If
While (S is non-empty)
```

Step 2: Now the robot is in the goal. When it comes time to make a move, the robot must examine all adjacent cells which are not separated by walls and choose the one with the lowest distance value. To find the optimal shortest path the robot plans a path to home through the explored (visited) grid points in the descending order of their labels (from label of the destination (maximum label) to label of source (0)). If the robot finds two or more possible adjacent nodes, then it follows the priority as: *straight ahead, east, west, north or south*.

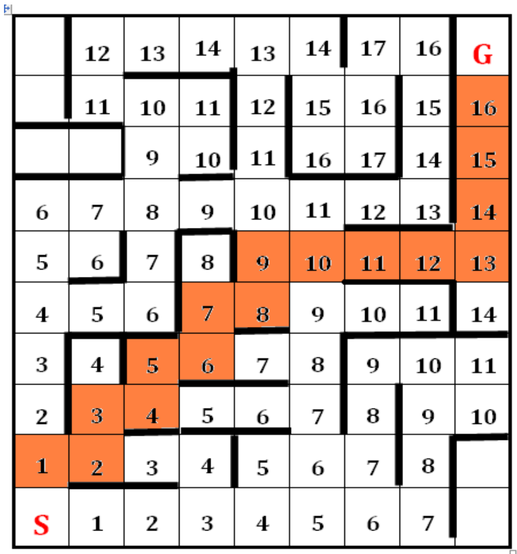

Figure. 3

The mobile robot can go South or West at the grid cell with the label 10 to reach the grid cell with label 9 and traverse the same number of cells on its way to the destination cell. Since turning would take time, the robot will choose to go forward to the West cell. The shaded boxes (in Fig.3) represent the shortest path between the source and the goal as per the flood fill algorithm in static environment.

Consider the same maze in Fig.1 with mobile obstacles and a mobile robot. The flood fill algorithm can be extended to handle the dynamicity.

III. PROPOSED ALGORITHM AGENT

The algorithm is successful only if the proper data structures are used effectively. To implement this algorithm the following are the requirements in terms of mode of storage.

A. Data Structures in Need

Data structures are the containers to hold data. The simplest one is array. It is suitable for static situations. To consider dynamicity the linked lists can be used.

1. To handle the dynamic behavior of the obstacles (walls) the position of the walls has to be noted. The information about the position of walls is collected from the sensor data while the mobile robot explores the environment and stored in a *structure* named *cell_info* as

```
struct cell_info
{
        int label;
        boolean front, back;
        boolean left, right;
} cell[row][column];
```

The variables *row*, *column* denote the number of grid cells in the environment taken horizontally or row wise and vertically or column wise respectively. Each grid cell has an object of this structure to store its information about its label using *label* integer. The boolean values are used to denote the presence of walls on the respective sides. For example, *front* variable has the value of *yes/true* if a wall is present at the front, of the particular grid cell else it has the value of *no/false*. This is the same for other three *back*, *left* and *right* variables. From the boolean values the accessible neighbors for the particular cell can be found. To save memory space we can also use the ***bit-fields*** concept as 8 bits for *label*, a single bit for each boolean variable. Each object occupies only *12 bits*. Thus the space occupied by all the objects is the product of 12 and number of grid cells.

2. The information about the *walls* can be stored in a simple data structure *linked list of linked lists* since an entire linked list is necessary to have the details about the location of a single wall. The nodes contain the grid point number on which the wall lies. The significances of using linked lists are

- To allocate only the necessary memory space (instead of having static fixed size arrays) since memory requirements change, if any change occurs in the environment.
- Allows dynamic updates including the creation and deletion of linked lists (walls) thus saving memory space.
- Can store any type and any number of data in a single node by using *structure* to hold data.

3 .The grid cell environment can be considered as a *two dimensional array* since we consider the indoor environment. This is to save grid point values which will be further used in the nodes of the linked lists to specify the location of walls.

4. It is essential to store the (old and new) neighbors of the walls in an *array* dynamically. For this, we can use the dynamic allocation techniques. The number of neighbors depends upon the length of the wall. It is usually twice its length. The length of the wall can be obtained from the linked lists.

B. Dynamic Planning

Initially, these values are filled for the explored grid cells in their respective objects as per the sensed data about the position of walls. After the initial exploration, the robot plans the shortest path to the source as per the flooded values stored in the *label* variable in each object. Meanwhile, any change may have been happened that is the walls may have changed its positions or some new walls may have appeared or some of the walls may be missing.

To manage this change in the environment we propose the following steps after exploration.

1. Verify the *label* values of all the neighbors from their objects of the structure *cell_info*.
2. The mobile robot (in the destination) moves to the nearest neighbor cell according to the value of the *label* if there is no wall in-between them.
3. If the mobile robot find any change in the environment (with the help of sensors), then the flooding has to be repeated in the neighborhood of the location where the change has occurred.

- If the wall has changed its position, the object values of the neighbors of the old and new position of the wall have to be updated.
- If the wall has disappeared, the past neighbor (object) values have to be updated.
- If a new wall appears dynamically, the neighbor (object) values have to be updated.

The boolean (object) values of all the affected neighbors have to be updated first in order to find their *label* values later. The *label* of the cell can be found only if the accessible neighbors of the particular cell are known. The accessible neighbors can be known only from the boolean values of the particular cell. The affected neighbors can be pushed into a *stack* and popped one cell at a time to update its values if necessary. A cell is popped out and its *label* is found from the values of the neighbors as

***label* = min (*label* of accessible neighbors) + 1**

if there is at least one accessible neighbor.

***label* = -1**

if there is no accessible neighbor. Each cell is popped out and their object values are updated. After the update of all the neighbors, the robot takes its path.

C. Collision Avoidance

The robot moves in the local optimal path until it encounters a wall at its path. It is difficult to predict the position and movement of walls in a dynamic environment. If it finds any wall at its path, it takes the next optimal path if available. If the label of the neighbors is in such a way that the robot can't take that path, then the neighbor values are flooded again as per the proposed strategy and then the robot continues its path. The processing time is comparatively shorter than the time taken by the wall to move. So there is less chance of collision. Our future work is to avoid the collision completely and to find the global optimal path.

IV. CONCLUSION

The proposed algorithm agent supports for the robot to reach the place despite moving obstacles or walls. The proposed algorithm will be suited for the personal robot assisting the physically challenging/old age people for fetching the things and the exploration of map with vision will guide the visually challenging people also. The future work can be with the maze to find the shortest path between any two points in the maze in a shorter time.

REFERENCES

[1] Sebastian Thrun.” Learning metric-topological maps for indoor robot navigation.” *Artificial Intelligence*, 99(1):21–71, 1998.

[2] R. Clark, A. El-Osery, K. Wedeward, and S. Bruder “A Navigation and Obstacle Avoidance Algorithm for Mobile Robots Operating in Unknown, Maze-Type Environments” Proc. International Test and Evaluation Association Workshop on Modeling and Simulation, Las Cruces, NM, December 2004

[3] James Ballantyne “Robotic Navigation in Crowded Environments: Key Challenges for Autonomous Navigation Systems” Imperial College London.

[4] Borentein, J. and Y. Koren, “Real-time obstacle avoidance for fast mobile robots in cluttered environments,” Proceedings IEEE International Conference on Robotics and Automation, Cincinnati (OH), USA, May 13-18, 1990, pp. 572 – 577.

[5] Wang, L., “Computational intelligence in autonomous mobile robotics- A review”, Proceedings of the 2002 International Symposium on Micromechatronics and Human Science, Nagoya, Japan, Oct. 20-23, 2002, pp 227 – 235.

[6] An Efficient Dynamic System for Real-Time Robot-Path Planning Allan R. Willms and Simon X. Yang, IEEE transactions on systems, man, and cybernetics – part B: cybernetics, vol. 36, no. 4, August 2006.

[7] Fedor A.kolushev and Alexandar A.Bogdanov,”Multi- agent Optimal Path Planning for Mobile Robots in Environment with Obstacles”.

[8] C. W. Warren, “Fast path planning using modified a* method,” in *Proc.IEEE Int. Conf. Robotics and Automation*, Atlanta, GA, May 1993,pp. 662–667.

[9] “A Robust Layered Control System for a Mobile Robot”, Rodney A.Brooks, Aritificial Intelliegence Laboratory, Massachusetts Institute of Technology, 1985

[10] “Artificial Life and Real Robots”, Rodney A. Brooks, MIT Artificial Intelligence Laboratory, 1992.

[11] Map Building with Mobile Robots in Dynamic EnvironmentsDirk H¨ahnel Rudolph Triebel Wolfram Burgard Sebastian Thrun University of Freiburg, Department of Computer Science, Germany Carnegie Mellon University, School of Computer Science, PA, USA.

[12] Robot Navigation Planning Problems in Dynamic Environments R. Urniezius, S. Bartkevicius Department of Theoretical Electrical Engineering, Kaunas University of Technology,Studentu 48, Lt- 51367 Kaunas, Lithuania, 2008.

[13] Anytime Dynamic A*: An Anytime, Replanning Algorithm Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun School of Computer Science Computer Science Department Carnegie Mellon University Stanford University Pittsburgh, PA, USA Stanford, CA, USA