FINDING AN OPTIMAL FREE-OBSTACLE PATH FOR A MOBILE ROBOT USING A* ALGORITHM

[1]AMER S. ALHARTHI, [2]SAAD B. ALOTAIBI

King Abdulaziz for Science and Technology (KACST), National Center for Robotics and Automation System
E-mail: [1]aharthi@kacst.edu.sa

Abstract - The path planning is a fundamental problem in robotics, it consists to find an optimal free-obstacle path for a robot from a starting position to a given destination (a goal). There are many approaches to solve this problem, one of the most common and widely known is the A Star search. In this paper we will implement the A* searching technique to solve the path planning problem. We assume the environment of the robot is represented by a grid map in which the initial and goal states (current location and the destination) are clearly localized on the map. Some cells of the map contains obstacles the robot cannot traverse them to reach the goal. The robot can move in different directions (horizontally or vertically) from a cell to another neighbor free cell. In the following sections we will describe how to implement A* algorithm on static environment with randomly obstacles.

Keywords - Path Planning, A* Algorithem, Free-Obstcale

I. INTRODUCTION

Path planning is an important and essential process in the field of robotics. It is the planning for sequence of steps that needs to be followed by the moving robot to reach its final destination. A moving robot can reach its final destination in many different paths. The planning must be carried out to select the path that is shortest, efficient and optimal. Various different algorithms are available to perform the path planning process. Some of the popular algorithms are Breath-first, Dijkstra, A*, D*, ARA*, AD* and many others. All these algorithms works on graph mapping approach and finds the shortest route by selecting the nearest neighbor through a series of iteration. The iteration methods are different for different cases and are suitable for a particular situation. Each of the algorithms has its own merits and demerits. Some algorithm works well for static obstacles whereas others are good for dynamic obstacles or both. Some algorithm follows global path planning whereas the others follow local path planning. The global path planning requires the entire information of the planning plane and performs the planning with respect to the information. The local path planning does not have any information of planning plane and the path planning is carried out online while the robot is moving.

The A* algorithm also known as the static algorithm is perfectly suitable for the robot movement in plane having static obstacles. It always aims for the shortest path and takes the weightage of all the nearby cells before following the path[1]. The idea of the paper is to use the A* algorithm to find the optimal path between the current and goal state for a mobile platform. In addition, it should find it by decreasing the number of steps between current and goal states. Path planning problem has many methods to find the solution from 200 years, but those methods are not strong enough also, they are not easy to adapt with any dynamic change in the problem formulation as most of them based on the concept of space configuration. The biggest problem of the methods are the need for extra resources (size, time and memory) to get an optimal solution. A* has been recognized as one of the most strong searching algorithm for complex and misbehaved optimization problems and can work with static and dynamic environment, furthermore it inheres parallel search techniques. the problem will be formulated as a robot moving in environment and want to find a path along with static obstacles from a starting point to ending point using optimization criteria which is Shortest Path, moreover it allows four-neighbor movements with low complexities [1]. In the following section, we will discuss the commonly path planning algorithms.

II. RELATED WORK

Based on the path planning algorithms that used in real environment (Static or Dynamic), we discuss some of methods in path planning of mobile robots. The path planning is the set of principles as followed by mobile robots to ensure collision free and shortest path movement for both dynamic as well as the static obstacles. Many different algorithms are available to ensure the path planning. One of the proposed method, the authors have taken the commonly available honeybee movement as an analogy and proposed a path planning algorithm. The proposed algorithm (Artificial Bee Colony algorithm) is experimented for both static as well as dynamic obstacles under certain conditions. The authors proposed solution in two cases namely, path planning in static and dynamic environments. The algorithm is proposed for a 2D environment with grids. Each of the cell joining four grid points is called node. In the first case (static environment), A 2D environment with multiple nodes are considered and a population size is initialized with equal number of employed and

onlooker bee. The number of paths (Pt) is assigned as half of population size. At start of the algorithm, Pt is determined randomly. An objective function is determined for each Pt. A fitness function is estimated for each Pt by taking the inverse of the objective function. The fitness value is criteria for the obstacle free shortest patch. At every instant 8-neighbour rule is used check the next possible obstacle free paths and objective function is again determined. This process is followed until the required destination is met with shortest time frame. In second case (dynamic environment), the number of paths as determined for the first case are arranged in decreasing order. The sorting is carried out depending on the fitness value. The best path is determined by considering the dynamic obstacles. If the fitness value is increased beyond a specific value then that particular path is discarded and the remaining ones are considered[1].

Other researchers discuss three of the popularly available graph search algorithms for path planning and presents the results. The algorithms are namely Breadth-first, Dijkstra and A*. Experiments are carried out to determine the shortest path and the situation of multiple goals for a single starting point. Mobile robots generally follow grid based approach to detect the shortest and economic route. The experiments are carried out by assuming certain set of situations. The three graph search algorithms have their own merits and demerits and must be employed depending upon the situations and requirements[2].
The Breadth-first algorithm works on branching method. It starts with the starting cell of the map matrix and locates the nearby cell that is obstacle free and close to the goal. The identification number of the next nearby cell is updated in the OPEN LIST until the goal is reached. The OPEN LIST provides the optimum path upon reaching the goal cell. The total cost is later estimated by multiplying the OPEN LIST path and the cost of the corresponding cells[2].
The Dijkstra algorithm is an upgraded version of the Breadth-first algorithm. It locates the nearby obstacle free cell based on the cell cost. It chooses the least cost nearby cell for movements and updates the OPEN LIST accordingly[2].

A* algorithm is the most efficient and requires a bit more computation for locating the optimum path. For each of the movement it selects the next new neighbor by considering all the previous located nearby neighbors. At any point it considers the just previous cell for selecting the optimum low cost path[2].

III. EXPERIMENTS AND RESULTS

In our work we assume the environment of the robot is represented by a grid map in which the initial and goal states are clearly localized on the map. Some the cells in a map contains obstacles and the robot cannot traverse them to reach the goal. The following figure illustrates the environment (AreaMap) of the robot.

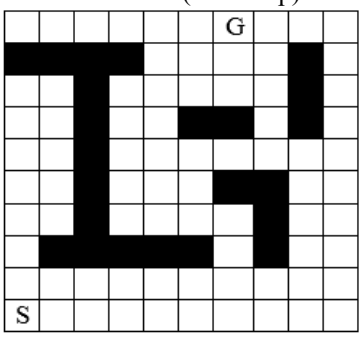


Figure 1

The robot can move in different directions (horizontally or vertically) from a cell to another neighbor free cell. In the following sections we will describe the requirements of A* algorithm that we will implement on our environment.

1- Area Map

The whole map will be represented by Map width, Map height, All map cells (nodes) Start location in map (X, Y) and Goal location in map (X, Y). To start using the map, it must be created first and be will prepared as the following steps:

1. Fill the array representing the map with nodes and put the X and Y for each node.
2. Register each node neighbors (other nodes) from (East, West, North and South).
3. Set start location and remove any old start location and mark the start location as start.
4. Set goal location and remove any old goal location and mark the goal location as goal.

To get distance between any two nodes as it will be required through the way to the goal, and if the two nodes on the same column (X) or the same row (Y), then the distance will be 1.

2- Node

Each node which is a cell in the map has a location determined by column (X) and row (Y) and also has neighbors (other nodes) from East, West, North and South as shown below:

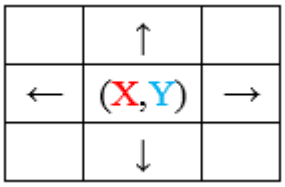


Figure 2

Each node will have its related info and also some information about its neighbors:

1. Its location on X axis (row).
2. Its location on Y axis (column).
3. Is this node the start node or not.
4. Is this node the goal node or not to stop search.
5. Is this node an obstacle (block) node

6. Is this node visited before or not to not visit it again (Boolean visited).
7. Its distance from start node.
8. Its distance based on heuristic from goal.
9. The previous step (node) before visiting this node.
10. The node on the west side of this node (west).
11. The node on the east side of this node (east).
12. The node on the north side of this node (north).
13. The node on the south side of this node (south).
14. List of all neighbors from all sides.

Almost of functions in Node class is a setters and getters functions. The different setters and getters from normal is the ones for west, east, north and south because each setter insert into the neighbors list. Node class implement comparable as it will be so often to compare two nodes with each other, its equal when Xs of nodes equals and Ys equals to each too or not, and also implement equals function to have compareTo function from comparable. To compare must be check if the total distance from current node to goal node less than or greater than the same distance for other node and will return (1, -1 or 0)

1 -> current node distance is greater than the other node.
-1 -> current node distance is less than the other node.
0 -> current node distance equal to the other node.

3- Path

Path is simply the steps to go from start node to goal node in terms of Nodes list (ArrayList<Node> steps). And the most important functions are the ones add to the list and check if a node already in the list or not

- getLength – return the length of path which is the list size.
- getStep – get the node object in a specific index.
- getX – get the X axis value for a node in a specific index.
- getY – get the Y axis value for a node in a specific index.
- appendStep – add step to the end of list.
- prependStep – add step to the start of list and will be used with the previous information from Node.

4- Heurisitic

It is the way of trying to find a solution for any problem by making some compromising (speed, accuracy, optimality or completeness) to find approximate solution for problems don't have exact solution. In other words, heuristic is the way to trying to guess the solution in the way we think after getting a problem or puzzle. Heuristic trying to find the closest node to the each time by applying the following equation:

Distance to goal = (dx)2 + (dy)2
Where d is delta (dx = goalX – currentX) and same for dy

And after getting the distance for each node visited during process, it will be add to a list to pick the next step from, given that Node implements comparable based on the value of heuristic then the list will be sorted after adding new node to find the closest node on the top of list and finally pick the top of the list to work with.

To implement A* algorithm we need to encapsulates all the previous requirements in one unit and will be used directly from the Test class file and also has the full business of algorithm and how it works. A star It is the class that encapsulates all the previous classes in one unit and will be used directly from the Test class file and also has the full business of algorithm and how it works. AStar has inner class called SortedList based on ArrayList to add the following functions which will be useful in Astar:

- getFirst -> get the first entry in the list.
- Add -> add to list and sort the list after addition.

AStart has all the required data to be ready for work based on the previous classes as following:

- AreaMap object -> the map will work on.
- ClosestHeuristic object -> will be used to guess the next step.
- closedList -> list of nodes visited to not visit it again.
- SortedList object -> for open nodes (ready to be visited) sorted using heuristic.
- Path object -> for shortest path formation during the process.

AStar has three main functions to make the algorithms

- reconstructPath -> to form the path after reaching the goal using the previous value in each node, and will go from goal previous node to its previous and so on till reaching the start node which has no previous and during each step will add to the path.

Goal -> previous -> previous -> ….. -> start

- printPath -> will go through the whole map (nested loop) to print it with its obstacles, start, goal and path drawing as following:

  - if the visited is obstacle, then print 'X'
  - else if it is the start node, then print 'S'
  - else if it is the goal node, then print 'G'
  - else if it is in the detected shortest path, then print '0'
  - else print empty space ' '

- calcShortestPath -> important function which start with the start node and get its neighbor to check which way will take to the goal based on

heuristic and doing the same with the selected neighbor till reaching the goal. The chart below identifies the steps to move from node to node and construct the path and finally print the path.

We have developed A* algorithm in Java and implemented it on grid map using NetBeans IDE 8.1. Testing of the algorithm was carried out for three simple situations. The results of the experiment show an excellent performance for getting a shortest path. The following figures (cf. figure 3-5) show the results in three simple situations.

CONCLUSION

In this paper, we discussed the path planning of mobile robots, proposes a method and experiments. We done the proposed method in three different randomly grid map. The initial and goal states are clearly localized on the grid map. Some the cells in a grid map contains obstacles and the robot cannot traverse them to reach the goal. In the end, we have an excellent performance for getting a shortest path using A* algorithm.

ACKNOWLEDGEMENT

This research was supported by King Abdulaziz for Science and Technology (KACST), National Center for Robotics and Automation System.

REFERENCES

[1] I. Al-Taharwa, A. Sheta, and M. Al-Weshah, “A mobile robot path planning using genetic algorithm in static environment,” Journal of Computer Science, vol. 4, no. 4, pp. 341–344, 2008.

[2] Goel, Priyanka; Singh, Devendra, “Efficient ABC Algorithm for Dynamic Path Planning”, International Journal of Computer Applications, February 2014, pp. 15.

[3] T. Y KSEL, A. SEZGIN, “An Implementation of Path Planning Algorithms for Mobile Robots on a Grid Based Map”, Proc of the 4th International Conference on Electrical and Electronics Engineering (ELECO 2005), Bursa, Turkey, 2005, pp. 382-387.

★★★