Warsaw University of Technology

Navigation

Global A\* algorithm

Piotr Kurowski

Adam Sobiesiak

Group 87

1. SLAM

SLAM is the Simultaneous Localization And Mapping. It is one of the core problems facing autonomous mobile robots. The core SLAM problem is the requirement that a robot, starting in an unknown environment, explore in order to learn the environment (map), while simultaneously using that map to keep track of the robot’s position (localize) within the environment.

The SLAM problem is difficult for many reasons. Mobile robots are moving sensor platforms – sensors vary in type and capability but all have some limitations and all are noisy to some degree.

2. A\* Algorithm

A\* algorithm solves problems by searching among all possible paths to the goal for the one that incurs the smallest cost (least distance travelled, shortest time).

Assume that robot wants to get from point A to point B. This is schematic map of robot neighborhood.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  | 1 |  |  |  |  |
|  |  |  | 1 |  |  |  |  |
|  | A |  | 1 |  |  | B |  |
|  |  |  | 1 |  |  |  |  |
|  |  |  | 1 |  |  |  |  |
|  |  |  |  |  |  |  |  |

Where: A - start point (robot position), B - goal point, 1 - illegal terrain, green fields - legal terrain

This is scheme of the A\* algorithm for this problem.

* Begin at the starting point A. Add this point to “close point” list. Close Point list is list of squares that you don’t need to look at again for now.
* Look at all the reachable squares adjacent to the starting point, ignoring illegal terrain. Add this points to the “open list”. For each of these squares, save point A as its “parent square”.
* Choose next square with the lowest F cost.

F = G + H

G = the movement cost to move from the starting point A to a given square on the grid, following the path generated to get there.

H = the estimated movement cost to move from that given square on the grid to the goal which is point B. There are couple of methods to calculate this movement cost. We use Manhattan method. In this method robot calculate the total number of squares moved horizontally and vertically to reach the target square from the current square, ignoring diagonal movement, and ignoring any obstacles in the way.

* Path is generated by repeatedly going through our open list and choosing the square with the lowest F score. Drop this choose square from the open list and add it to the closed list.
* Check all of the adjacent squares. Ignoring those that are on the closed list or unwalkable, add squares to the open list if they are not on the open list already. Make the selected square the “parent” of the new squares.
* If an adjacent square is already on the open list, check to see if this path to that square is a better one. In other words, check to see if the G score for that square is lower if we use the current square to get there. If not, don’t do anything.   
      On the other hand, if the G cost of the new path is lower, change the parent of the adjacent square to the selected square (in the diagram above, change the direction of the pointer to point at the selected square). Finally, recalculate both the F and G scores of that square.

 In this example, each horizontal or vertical square move have cost equal 10 and a cost diagonal move have cost 14.

This is path calculate from point A to B by A\* algorithm.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  | 1 |  |  |  |  |
|  |  |  | 1 |  |  |  |  |
|  | A |  | 1 |  |  | B |  |
|  |  |  | 1 |  |  |  |  |
|  |  |  | 1 |  |  |  |  |
|  |  |  |  |  |  |  |  |

A\* algorithm is complete and will always find a solution if one exists.