Artificial Intelligence in Mobile Robots

Lab 5 - Path planning

By *Tobias L & Özgun M*

Lab Assistant *Ali Abdul Khaliq*

Örebro University

October 7, 2016

Student information

Tobias Lindvall
870603-6657
tobiaslindwall@gmail.com

Özgun Mirtchev
920321-2379
ozgun.mirtchev@gmail.com

Report handed in: 2016-10-07

1 Overview

The objectives of this lab is to implement and test a search algorithm which will be used for path planning to be used by the ePuck. To be able to move from a start position to a goal position a set of problems needs to be solved, specifically which paths are allowed to be taken and at the same time avoiding obstacles along the way. Ultimately the end goal is to find the closest path between the start and the goal. The algorithm which is going to be used specifically for this lab is Breadth-First Search (BFS). BFS searches a space by layers and moves outwards from the goal to the start position, while at the same marking the spaces it has found and how many steps are required to get to a specific point in the space. The specific functions which are going to be implemented are Search() (BFS), MarkCell() and Plan() (Backtracking to find a path).

2 Tasks

2.1 Task 1 - Implement Search() and MarkCell()

- Pseudo code for both the Search() and MarkCell() procedures was given in the lecture slides. This pseudo code was used for implementing the functions into the source code.
- For modelling the virtual room where the Search() procedure were to be applied, some data structure was needed. At first, a simple 2D array (int map[i][j]) was used where "i" and "j" describe the coordinate of a node and the int value of the element describes the node's state. This array was later replaced by the structure "Map" that was given in the base code.
- For holding the adjacent nodes that were to be marked with info and eventually explored, the structure "List" was used.

Code for this task can be found in appendix: Search and MarkCell.

2.2 Task 2 - Map and Search()-simulating

Three different maps were used for testing Search(). The maps were expressed in a text file and read into the program by the function CreateMapFromFile(). The Search() results were saved by using the function SaveMapToFile().

```
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
@@@  S                                                     @@@
@@@                                                       @@@
@@@                                                       @@@
@@@                                                       @@@
@@@                                                       @@@
@@@                                                       @@@
@@@ ### ### ### ### ### ### ### ### ### ### ###          @@@
@@@                                                       @@@
@@@                                                       @@@
@@@                                                       @@@
@@@                                                       @@@
@@@                                                       @@@
@@@  G                                                    @@@
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
```

(a) First map

```
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
@@@  S   33  32  31  30  29  28  27  26  25  24  23  24 @@@
@@@  33  32  31  30  29  28  27  26  25  24  23  22  23 @@@
@@@  32  31  30  29  28  27  26  25  24  23  22  21  22 @@@
@@@  31  30  29  28  27  26  25  24  23  22  21  20  21 @@@
@@@  30  29  28  27  26  25  24  23  22  21  20  19  20 @@@
@@@  29  28  27  26  25  24  23  22  21  20  19  18  19 @@@
@@@ ### ### ### ### ### ### ### ### ### ### ###  17  18 @@@
@@@   5   6   7   8   9  10  11  12  13  14  15  16  17 @@@
@@@   4   5   6   7   8   9  10  11  12  13  14  15  16 @@@
@@@   3   4   5   6   7   8   9  10  11  12  13  14  15 @@@
@@@   2   3   4   5   6   7   8   9  10  11  12  13  14 @@@
@@@   1   2   3   4   5   6   7   8   9  10  11  12  13 @@@
@@@  G   1   2   3   4   5   6   7   8   9  10  11  12 @@@
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
```

(b) Searched first map

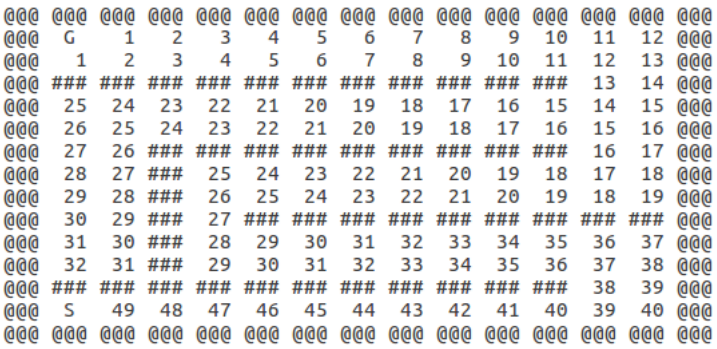
```
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
@@@                                     ###                 @@@
@@@                                     ###                 @@@
@@@                                     ###  S              @@@
@@@           ### ### ### ###           ### ### ###        @@@
@@@                     ###                                 @@@
@@@                     ###                                 @@@
@@@                     ###           ### ### ### ### ###  @@@
@@@                     ###                                 @@@
@@@                     ###                                 @@@
@@@       ### ### ###   ###                                 @@@
@@@            G  ###   ### ### ### ### ###                 @@@
@@@               ###                                       @@@
@@@               ###                                       @@@
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
```

(c) Second map

```
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
@@@  12  13  14  15  16  17  18  19  20 ###           28 @@@
@@@  11  12  13  14  15  16  17  18  19 ###       28  27 @@@
@@@  10  11  12  13  14  15  16  17  18 ###  S    27  26 @@@
@@@   9  10 ### ### ### ###  17  18  19 ### ### ###  25 @@@
@@@   8   9  10  11  12 ###  18  19  20  21  22  23  24 @@@
@@@   7   8   9  10  11 ###  19  20  21  22  23  24  25 @@@
@@@   6   7   8   9  10 ###  20  21 ### ### ### ### ### @@@
@@@   5   6   7   8   9 ###  21  22  23  22  21  22  23 @@@
@@@   4   5   6   7   8 ###  22  23  22  21  20  21  22 @@@
@@@   3 ### ### ###   9 ###  23  22  21  20  19  20  21 @@@
@@@   2   1  G  ###  10 ### ### ### ### ###  18  19  20 @@@
@@@   3   2   1 ###  11  12  13  14  15  16  17  18  19 @@@
@@@   4   3   2 ###  12  13  14  15  16  17  18  19  20 @@@
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
```

(d) Searched second map

(e) Third map



(f) Searched third map

2.3 Task 3 - Implement Plan()

The Plan() procedure was also implemented with help from pseudo code that was given in the lecture slides.

Code for this task can be found in appendix: Plan.

2.4 Task 4 - Map and Plan()-simulating

The three maps from Task 2 were used here as well. The Plan() function was modified to write the chosen path to the map, expressed as "*". Time measurements was also done to see difference from one map to another.

```
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
@@@  S   33  32  31  30  29  28  27  26  25  24  23  24 @@@
@@@  *   32  31  30  29  28  27  26  25  24  23  22  23 @@@
@@@  *   31  30  29  28  27  26  25  24  23  22  21  22 @@@
@@@  *   30  29  28  27  26  25  24  23  22  21  20  21 @@@
@@@  *   29  28  27  26  25  24  23  22  21  20  19  20 @@@
@@@  *   *   *   *   *   *   *   *   *   *   *   *   19 @@@
@@@ ### ### ### ### ### ### ### ### ### ### ###  *   18 @@@
@@@   5   6   7   8   9  10  11  12  13  14  15  *   17 @@@
@@@   4   5   6   7   8   9  10  11  12  13  14  *   16 @@@
@@@   3   4   5   6   7   8   9  10  11  12  13  *   15 @@@
@@@   2   3   4   5   6   7   8   9  10  11  12  *   14 @@@
@@@   1   2   3   4   5   6   7   8   9  10  11  *   13 @@@
@@@  G   *   *   *   *   *   *   *   *   *   *   *   12 @@@
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
```

(a) Planned first map

```
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
@@@  12  13  14  15  16  17  18  19  20 ###         28 @@@
@@@  11  12  13  14  15  16  17  18  19 ###      28  27 @@@
@@@  10  *   *   *   *   *   *   *   *  ###  S   *   *  @@@
@@@   9  *  ### ### ### ###  17  18  *  ### ### ###  *  @@@
@@@   8  *   10  11  12 ###  18  19  *   *   *   *   *  @@@
@@@   7  *    9  10  11 ###  19  20  21  22  23  24  25 @@@
@@@   6  *    8   9  10 ###  20  21 ### ### ### ### ### @@@
@@@   5  *    7   8   9 ###  21  22  23  22  21  22  23 @@@
@@@  *   *    6   7   8 ###  22  23  22  21  20  21  22 @@@
@@@  *  ### ### ###   9 ###  23  22  21  20  19  20  21 @@@
@@@  *   *   G  ###  10 ### ### ### ### ###  18  19  20 @@@
@@@   3   2   1 ###  11  12  13  14  15  16  17  18  19 @@@
@@@   4   3   2 ###  12  13  14  15  16  17  18  19  20 @@@
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
```

(b) Planned second map

```
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
@@@  G   *   *   *   *   *   *   *   *   *   *   *   12 @@@
@@@   1   2   3   4   5   6   7   8   9  10  11  *   13 @@@
@@@ ### ### ### ### ### ### ### ### ### ### ###  *   14 @@@
@@@  25  24  23  22  21  20  19  18  17  16  15  *   15 @@@
@@@  26  25  24  23  22  21  20  19  18  17  16  *   16 @@@
@@@  27  26 ### ### ### ### ### ### ### ### ###  *   17 @@@
@@@  28  27 ###  *   *   *   *   *   *   *   *   *   18 @@@
@@@  29  28 ###  *   25  24  23  22  21  20  19  18  19 @@@
@@@  30  29 ###  *  ### ### ### ### ### ### ### ### ### @@@
@@@  31  30 ###  *   *   *   *   *   *   *   *   *   37 @@@
@@@  32  31 ###  29  30  31  32  33  34  35  36  *   38 @@@
@@@ ### ### ### ### ### ### ### ### ### ### ###  *   39 @@@
@@@  S   *   *   *   *   *   *   *   *   *   *   *   40 @@@
@@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
```

(c) Planned third map

3 Conclusions

Search

When we tried to implement the Search-function, the first problem we encountered was to choose a map to be used to represent the grid on which the ePuck was going to calculate the path on. Our first solution was to use a simple 2D array, which would display the values of the cells. This worked out to be good, but we realised that using the Map from the maps.h library file was a better idea, since it had some convenient functions that could be used to manipulate the map.

The implementation of the Search-function was otherwise fairly straight-forward and was done fast.

Plan

Implementing the Plan-function from pseudo-code was also straight-forward. It works well and gives a path with the minimum possible steps to reach a goal.

We also decided to measure the time it takes for the Plan-function to compute a path. It was purely out of interest and was not a required task.

- First map: ~3.16 s
- Second map: ~2.63 s
- Third map: ~2.48 s

If we look at the times and compare them with the structure of each map, a possible conclusion would be that the more obstacles a map has, the less time it takes for the algorithm to find a path. Vice versa, it also takes longer to find a path if there are more nodes to search. This is evident if you look at the difference between the third map which has many more obstacles than the first map. Notice that we used a 20 ms delay in our search loop. The reason for this is that we wanted to print the map while searching and the delay made it possible to follow the progress visually. This causes the times to have abnormally high values, but still comparable to each other.

Search and MarkCell

```
void MarkCell(int i, int j, int value)
{
    Cell c = {i, j, 0.0};
    switch (map[i][j])
    {
    case -1: // Initial position
        Push(&TheQueue, c);
        break;
    case -2: // Unexplored c
        map[i][j] = value;
        Push(&TheQueue, c);
        break;
    case -3: // Obstacle
        break;
    case -4: // Out of boundaries
        break;
    default: // Already explored
        break;
    }
}

void Search(Cell start, Cell goal)
{
    ClearList(&TheQueue, FIFO);
    Push(&TheQueue, goal);
    while (!IsListEmpty(&TheQueue))
    {
        Cell current = Pop(&TheQueue);
        if (current.i == start.i && current.j == start.j)
            break;
        int dist = map[current.i][current.j] + 1;
        MarkCell(current.i, current.j - 1, dist);
        MarkCell(current.i, current.j + 1, dist);
        MarkCell(current.i - 1, current.j, dist);
        MarkCell(current.i + 1, current.j, dist);
        PrintMap(&TheMap);
        Sleep(20);
    }
}
```

Plan

```
void Plan(Cell start, Cell goal)
{
    Search(start, goal);
    Cell curr = start;
    Cell next;
    int it = 0;
    printf("(%d, %d)\n", curr.i, curr.j);
    do
    {
        next = GetNextNode(curr);          // Extracted into a function
           because of readability
        printf("(%d, %d) (%d)\n", next.i, next.j, map[next.i][next.j]);
        if (it)
            map[curr.i][curr.j] = -6;      // Marks the path on the map
               with "*"
        PathCoords[it] = next;             // Saves the path to global
           array
      it++;
        curr = next;
    } while (map[curr.i][curr.j] != 0);
}
```