Autonomous Robots Lab Report 4
Graph Search (A* Algorithm)

Raabid Hussain

April 23, 2016

1 Objective

The objective of this lab work was to implement the A-star graph search algorithm for path planning using MATLAB. We were provided with the list of obstacle vertices and the visible edges (using the RPS algorithm from lab 2). The start and the goal position of the robot were represented as the first and last element respectively in the vertices list. Given this graph of the environment, we had to find the optimal path through the A* algorithm. The optimal path took the euclidean distances as the optimizing parameter. The result was also graphically represented for a better visualization.

2 Introduction

Path planning is one of the key aspects of robotics nowadays. One common approach to this problem is first build a visibility graph. This is done by taking all the vertices in the map and connecting each with all the vertices that are visible to it. Different approaches can be used for that. After the visibility graph has been formed, it is important to select an optimal path from it. One popular approach is the A* algorithm which uses heuristic guide to perform a greedy best first search around the map. Although the algorithm is well noted for its performance and accuracy, there is one huge drawback: it requires a pre-known visibility map to process the map.

A* involves finding a path between the current position of the robot and the goal position, satisfying all the physical constraints. It starts with a

visibility graph containing all the nodes and the edges (possible connections between the nodes). It takes into account all of these edges by first considering the ones that appear to lead the shortest path with the help of a weighted tree like structure. It starts at the start position and keeps on expanding the tree until one of its branch reaches the goal position.

At each instance of expanding the tree, the algorithm determines which of its branches to expand into one or more longer branches based on a heuristic cost function. This heuristic function must never over-estimate the actual cost to get to the goal node. In our case we chose this cost function as the euclidean distance to the goal position from the current node.

3 Implementation of A* Algorithm

MATLAB was used to implement the A* algorithm for path planning. A set of vertices or nodes in the environment were provided. The vertices were input to the RPS algorithm implemented in previous labs to get a list of all the possible edges or paths. Next, both the list of the vertices and the edges were input to the A* function to get the optimum path and its euclidean distance. The start and the goal position were provided as the first and the last vertex respectively in the vertices list.

The algorithm starts by creating two empty lists called the closed and the open list. The closed list is to contain a set of all the nodes that have been evaluated whereas the open list is to contain a set of tentative nodes that need to be evaluated. This open list was initialized later to contain the starting position. For simplicity, both the lists also contained the distance needed to be traversed to reach each node in the list added with its heuristic distance. Also the heuristic distance, that is the euclidean distance from each node to the goal position was pre-calculated and stored in a list. And lastly a sparse matrix was made that contained the distances of each edge in the visibility graph.

The open list was sorted in each iteration according to the distance of each node as the node with the smallest distance after each iteration needs to be added to the closed list. This allowed to just simply push the first element in the open list into the closed list.

The algorithm works by adding the first node into the open list and then into the closed list or directly into the closed list. Then all the nodes that have a connectivity with this starting node are added into the open list in ascending order of their heuristic distance (distance to the goal position added with the distance needed to be traversed along all the nodes to reach the current node). Then the first node was inserted into the closed list and all the nodes having a connectivity with this node was added into the open list. This process is repeated until the stopping conditions are reached.

A catch in this approach is that many times it is found that the path being followed is not actually the optimal one. So whenever updating the open list, we have to check if there is an alternating path to the newly added nodes in the open list that has a shorter distance. This mostly occurs when the node being added is already in the open list. So if such an instance is found, then the path with the smaller heuristic distance is kept and the other discarded.

The algorithm stops when the node marked as goal enters the closed list resulting in a successful result or when the open list is found empty in which case no solution has been found. When the goal position is found to enter the closed list, a simple reverse traversal is done on the closed list to find all the nodes that should be in the optimal path. The result is shown in two forms. First a list containing all the nodes in the optimal path is output along with the physical distance of this optimal path. Secondly, a graphical representation of the map and the optimal path is shown.

3.1 Sample Test-run Results

After writing the MATLAB function, it was time to test the program on different environments. First a simple test environment was tried. The visibility graph was obtained using the rotational plane sweep algorithm for which the code was written in lab 2 of this course. The output of the RPS algorithm was input to the Astar function to get the optimal path nodes and the minimum distance needed to be traversed. The result is shown in figure 1.

```
path =

     1     3     4     7    10

>> minCost

minCost =

   13.3788
```

Figure 1: The output of the algorithm indicating all the nodes in the optimal path and the distance for the optimal path.

A visualization of the result obtained can be seen in figure 2, where the blue line lines represent the output of the RPS algorithm whereas the green line represents the optimal path found.

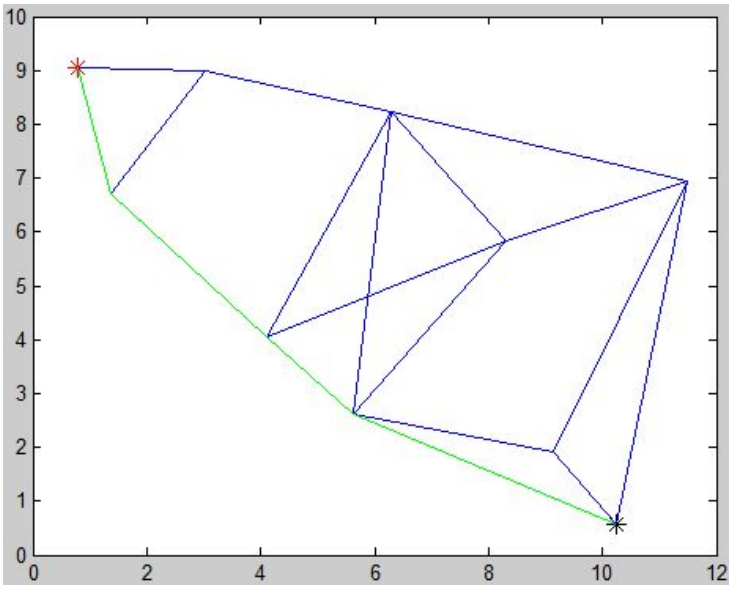

Figure 2: The optimal path found using A* algorithm for a simple map. The blue lines represent the visibility graph, whereas the green lines represents the path found between the start and the goal position.

The program was also tested on large complex environments, a sample output for which is shown in figure 3. Here the blue lines indicate all the possible paths the robot can use, whereas the green line represents the optimal trajectory found using the algorithm.

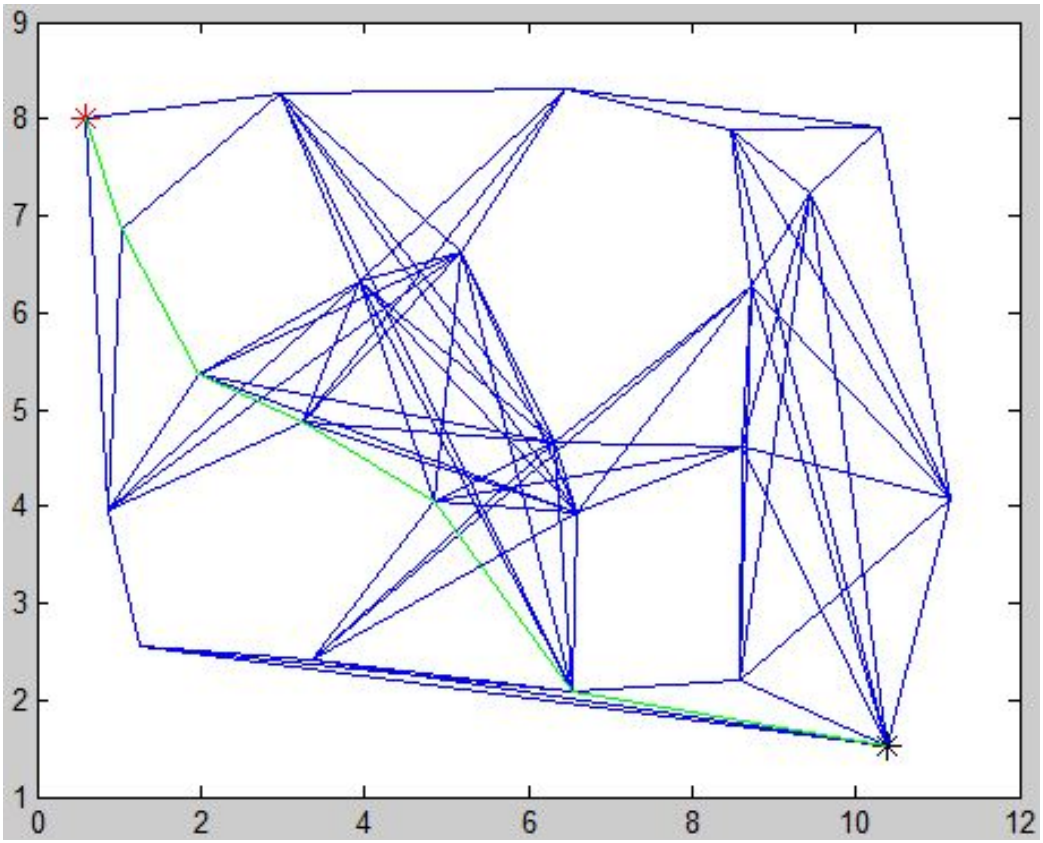

Figure 3: The optimal path found using A* algorithm for a large complex map. The blue lines represent the visibility graph, whereas the green lines represents the path found between the start and the goal position.

4 Conclusion

In this lab assignment, the A* algorithm was successfully implemented using MATLAB. Provided an already known visibility graph, the program can find the optimal path between the start and the goal position, satisfying all physical constraints. It used the euclidean distance as the heuristic cost function. For reference, the pseudo-code provided in the lab instruction manual was followed. The program was tested on various types of environments. All the test were successful.