Path planning using A*

David FILLIAT - ENSTA Paris

13 décembre 2021

1 Introduction

In this practical work, we will work on the A* path planning algorithm. For this, we will use the python code available on the course Moodle. The provided script `pathfind.py` makes it possible to read a map from an image file and plan a path between two points. The provided code requires the installation of the `numpy`[1], `matplotlib`[2] and `opencv-python`[3] python packages.

Upload your report as a pdf file that includes your answers to the questions and the code you wrote on the Moodle.

2 Heuristic influence

In this part, we will study the role of the heuristic, by keeping the default map, start and goal positions and modifying the `heuristic_weight` parameter in the beginning of the file.

Question 1 : With parameter `heuristic_weight = 0.0` what is the algorithm that is executed ? What are the advantages and disadvantages with respect to the version with `heuristic_weight = 1.0` ?

Question 2 : With parameter `heuristic_weight = 5.0` what is the result of the algorithm ? What are the advantages and disadvantages with respect to the version with `heuristic_weight = 1.0` ? What is the theoretical reason for this behavior ?

3 Influence of the environment

Question 3 : Compare the performance of the algorithm with `heuristic_weight = 0.0` and with `heuristic_weight = 1.0` in the different environments provided with the code. What do you see ? In which cases does the A* algorithm bring the most gain ?

4 Weighted nodes

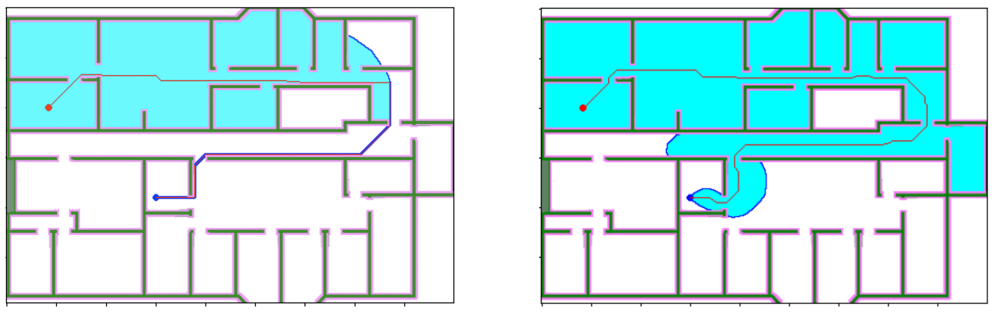

FIGURE 1 – Shortest path without node cost (left) and with a penalty for proximity of obstacles (right)

In the default configuration, no weight is associated with the nodes themselves. All nodes are therefore treated the same, and the optimal path will pass near obstacles and through narrow passages (Fig. 1, left).

1. https://numpy.org/
2. https://matplotlib.org/
3. https://opencv.org/

Question 4 : Modify the weight of the nodes so that the paths do not pass too close to obstacles (Fig. 1, right). You can use the opencv `cv2.distanceTransform(map, cv2.DIST_L2, 3)` function which makes it possible to compute, for each free position of the map, the distance to the nearest obstacle. Illustrate the trajectories obtained on the environment proposed by default in the script.