TP3 Path planning using A*

Yan CHEN

December 2020

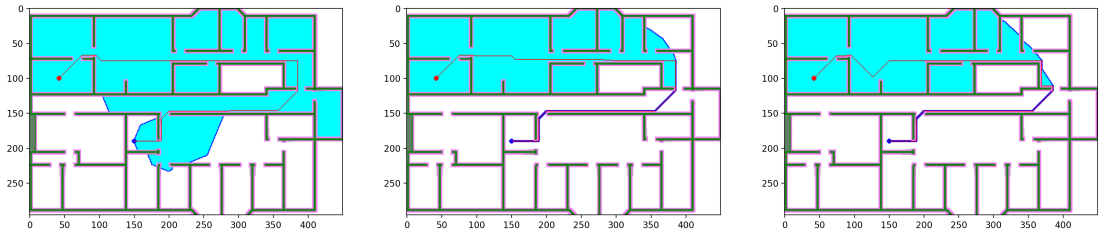
1 Introduction

In this TP we will talk about the A* path planning algorithm. A* algorithm judges the priority of every nodes according to the function as fallows :

$$f(n) = g(n) + h(n) \tag{1}$$

where the function $f(n)$ present the general priority of node n. $g(n)$ present the priority of node n toward to original point. $h(n)$ present the priority of node n toward to goal. Their values are smaller, their priorities are higher. And the function $h(n)$ is called heuristic influence.

2 Heuristic influence



(a) heuristic weight = 0 (b) heuristic weight = 1 (c) heuristic weight = 5

FIGURE 1 – Heuristic weight

Heuristic weight is a parameter of the function $h(n)$. The value is larger, the heuristic function has a bigger influence to the priority of the algorithm. When the heuristic weight is zero in the Figure , the A* algorithm is like Dijkstra algorithm. Because the heuristic function hasn't be considered. When the heuristic weight is 1, smaller zone is searched and the computation time is less. However the length

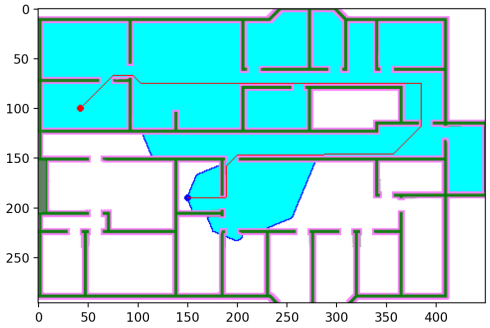
of path is longer. But it's not evident here. When the heuristic weight is 5, the searched zone is smaller relative to the heuristic weight=1. However, the length of path is longer and the computation time is longer.

Heuristic Weight	Computation Time / s	Path length
0	1.237	685.966
1	0.905	685.966
5	0.965	705.019

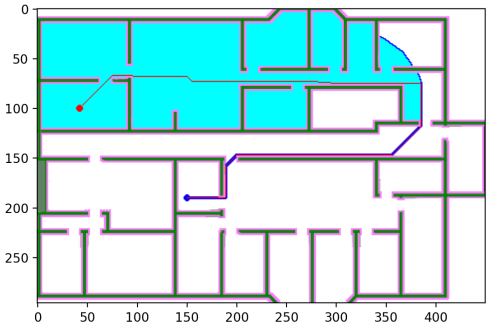
FIGURE 2 – Performance under different heuristic weights

If $h(n)$ is always less than or equal to the cost of node n to the goal, the A* algorithm is guaranteed to find the shortest path. But when the value of $h(n)$ is smaller, the more nodes the algorithm will traverse, which also leads to the slower computation. If $h(n)$ is exactly equal to the cost of node n to goal, then the A* algorithm will find the best path and it will be fast. Unfortunately, this is not possible in all scenarios. This is because it is difficult to figure out exactly how far to the goal until the robot reach it. If the value of $h(n)$ is larger than the cost of node n to the goal, the A* algorithm is not guaranteed to find the shortest path, though it will be faster.

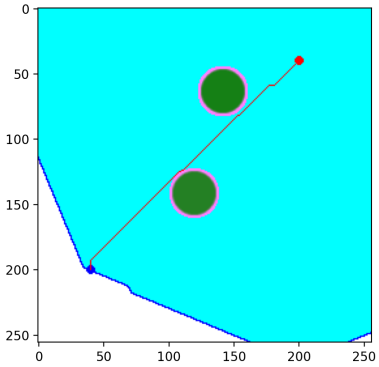
3 Influence of the environment



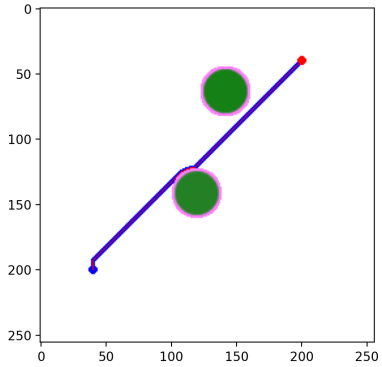
(a) Office : heuristic weight = 0



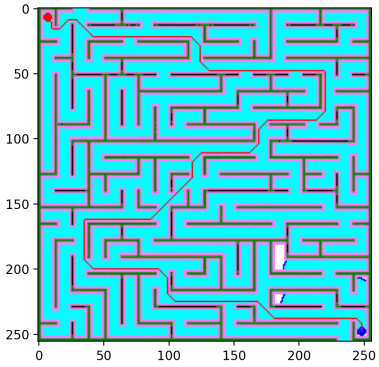
(b) Office : heuristic weight = 1



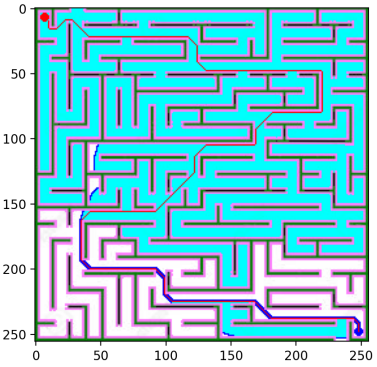
(c) Free space : heuristic weight = 0



(d) Free space : heuristic weight = 1



(e) Lab : heuristic weight = 0



(f) Lab : heuristic weight = 1

FIGURE 3 – Performance compared to different environment

Environment	Heuristic Weight	Computation Time / s	Path length
Office	0	1.237	685.966
	1	0.905	685.966
Free space	0	2.132	230.375
	1	0.015	230.375
Lab	0	0.796	785.747
	1	0.825	785.747

FIGURE 4 – Performance under different environments

Based on the above results, we can find that the heuristic weight is larger, the researched zone is smaller. And theoretically it needs less time to find a shortest path, especially, in the free space. But the gap between the computation time of them decreases with more obstacles in the environment. And when there are a large number of obstacles in the environment, the situation of heuristic=0 dispense even less time than that of heuristic=1, such as in the Labyrinte. Anyway, the two situation have almost found the same shortest path.

4 Weighted nodes

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. So here I modify the weight of nodes. These points are more closed the obstacles, they have a higher weight of cost. The robot will not pass too close to obstacles. So the function of cost becomes as fallow :

$$f(n) = g(n) + h(n) + w(n) \tag{2}$$

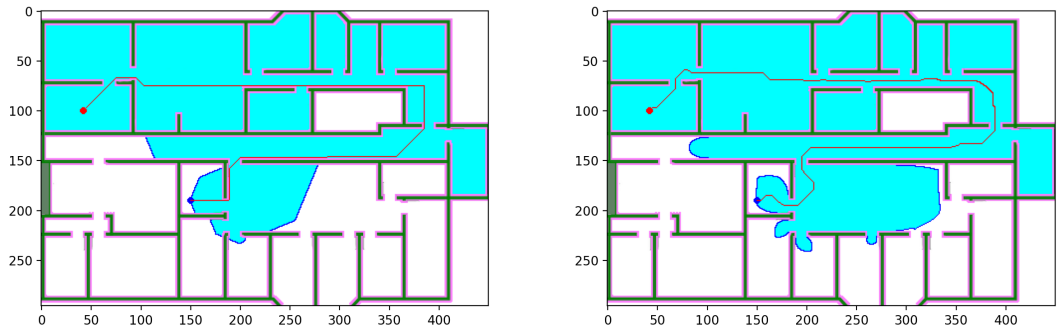
where the $w(n)$ is cost function of nodes.

I add two ranges of codes in the function __init__() that distribute the weight of nodes in the environment.

```
self.dis = cv2.distanceTransform(self.map, cv2.DIST_L2, 3) # calculate thess
    distances bewteen points and points 0 in the map
self.dis_cost = self.dis.max()/(self.dis+1e-5) # thes points neaby the obstacls
     have more cost
```

Then I modify the code in the 156 range.

```
new_cost = cost_so_far[current]+math.dist(current, next)+self.dis_cost[next]
```



(a) left (b) right

FIGURE 5 – Shortest path without node cost (left) and with a penalty for proximity of obstacles (right)

Based on the above result, the method of distributing the cost weight of nodes have a better performance. The robot will not move nearby the obstacles.