**基于A\*算法的无人车路径规划**

**实验报告**

|  |  |
| --- | --- |
| **班级** | **9191062302** |
| **学号** |  |
| **姓名** |  |

目录

[1 实验目的 3](#_Toc125903706)

[2 实验内容 3](#_Toc125903707)

[3 实验代码 3](#_Toc125903708)

[4 实验结果 8](#_Toc125903709)

# 实验目的

通过实验掌握A\*智能搜素算法及其在无人车路径规划中的应用。提升使用智能搜索算法解决实际问题的能力。

# 实验内容

生成一个NxN的二维网格，随机指定一些格子为障碍， 并设定左上角有辆无人车（占一个格子）要去右下角，使用A\*算法为该无人车计算起点到终点的不撞到障碍的最优路径。

# 实验代码

启发函数f(n)= g(n) + h(n)，g(n)为当前节点到起点的距离，h(n)为当前节点到终点的距离。

*import* math

*import* random  
*import* matplotlib.pyplot *as* plt  
  
  
*class* AStar:  
 *# 初始化  
 def* \_\_init\_\_(*self*, ox, oy, edge, radius):  
 *self*.min\_x = *None  
 self*.min\_y = *None  
 self*.max\_x = *None  
 self*.max\_y = *None  
 self*.x\_width = *None  
 self*.y\_width = *None  
 self*.obstacle\_map = *None  
  
 self*.edge = edge *# 网格边长  
 self*.radius = radius  
 *self*.calc\_obstacle\_map(ox, oy) *# 绘制  
 self*.motion = *self*.get\_motion\_model() *# 无人车运动方向  
  
 class* Node:  
 *def* \_\_init\_\_(*self*, x, y, cost, parent\_index):  
 *self*.x = x  
 *self*.y = y  
 *self*.cost = cost  
 *self*.parent\_index = parent\_index  
  
 *def* \_\_str\_\_(*self*):  
 *return* str(*self*.x) + ',' + str(*self*.y) + ',' + str(*self*.cost) + ',' + str(*self*.parent\_index)  
  
 *# 寻找最优路径，返回最短路径  
 def* planning(*self*, sx, sy, gx, gy):  
 start\_node = *self*.Node(*self*.calc\_xy\_index(sx, *self*.min\_x),  
 *self*.calc\_xy\_index(sy, *self*.min\_y), 0.0, -1)  
 goal\_node = *self*.Node(*self*.calc\_xy\_index(gx, *self*.min\_x),  
 *self*.calc\_xy\_index(gy, *self*.min\_y), 0.0, -1)  
 *# 保存进入集合节点和待计算节点* open\_set, closed\_set = dict(), dict()  
 open\_set[*self*.calc\_index(start\_node)] = start\_node  
 *while True*:  
 *# f(n)= g(n) + h(n)* c\_id = min(open\_set, key=*lambda* o: open\_set[o].cost + *self*.calc\_heuristic(goal\_node, open\_set[o]))  
 current = open\_set[c\_id]  
 *# plt.plot(self.calc\_position(current.x),  
 # self.calc\_position(current.y), 'xc')  
 # plt.pause(0.0001)  
 if* current.x == goal\_node.x *and* current.y == goal\_node.y:  
 goal\_node.cost = current.cost  
 goal\_node.parent\_index = current.parent\_index  
 *break  
 del* open\_set[c\_id]  
 closed\_set[c\_id] = current  
  
 *# 遍历邻接节点  
 for* move\_x, move\_y, move\_cost *in self*.motion:  
 node = *self*.Node(current.x + move\_x,  
 current.y + move\_y,  
 current.cost + move\_cost, c\_id)  
 n\_id = *self*.calc\_index(node)  
 *if* n\_id *in* closed\_set:  
 *continue  
 # 判断邻接节点  
 if not self*.verify\_node(node):  
 *continue  
 if* n\_id *not in* open\_set:  
 open\_set[n\_id] = node  
 *else*:  
 *# 更新  
 if* node.cost <= open\_set[n\_id].cost:  
 open\_set[n\_id] = node  
 rx, ry = *self*.calc\_final\_path(goal\_node, closed\_set)  
 *return* rx, ry  
  
 @staticmethod  
 *def* calc\_heuristic(n1, n2): *# n1终点，n2当前网格  
 return* math.hypot(n1.x - n2.x, n1.y - n2.y) *# 当前网格和终点距离  
  
 # 无人车每次能向周围移动8个网格移动* @staticmethod  
 *def* get\_motion\_model():  
 *# [dx, dy, cost]* motion = [[1, 0, 1], *# 右* [0, 1, 1], *# 上* [-1, 0, 1], *# 左* [0, -1, 1], *# 下* [-1, -1, math.sqrt(2)], *# 左下* [-1, 1, math.sqrt(2)], *# 左上* [1, -1, math.sqrt(2)], *# 右下* [1, 1, math.sqrt(2)]] *# 右上  
 return* motion  
  
 *# 绘制栅格地图  
 def* calc\_obstacle\_map(*self*, ox, oy):  
 *self*.min\_x = 0  
 *self*.min\_y = 0  
 *self*.max\_x = N  
 *self*.max\_y = N  
 *self*.x\_width = round((*self*.max\_x - *self*.min\_x) / *self*.edge) *# x方向网格个数  
 self*.y\_width = round((*self*.max\_y - *self*.min\_y) / *self*.edge) *# y方向网格个数  
 # 初始化地图  
 self*.obstacle\_map = [[*False for* \_ *in* range(*self*.y\_width)]  
 *for* \_ *in* range(*self*.x\_width)]  
  
 *# 设置障碍物  
 for* ix *in* range(*self*.x\_width):  
 x = *self*.calc\_position(ix)  
 *for* iy *in* range(*self*.y\_width):  
 y = *self*.calc\_position(iy)  
 *for* iox, ioy *in* zip(ox, oy):  
 d = math.hypot(iox - x, ioy - y)  
 *if* d <= *self*.radius:  
 *self*.obstacle\_map[ix][iy] = *True  
 break  
  
 # 根据网格编号计算实际坐标  
 def* calc\_position(*self*, index):  
 pos = index \* *self*.edge  
 *return* pos  
  
 *# 位置坐标转为网格坐标  
 def* calc\_xy\_index(*self*, position, minp):  
 *return* round((position - minp) / *self*.edge)  
  
 *# 给每个网格编号，得到每个网格的key  
 def* calc\_index(*self*, node):  
 *return* node.y \* *self*.x\_width + node.x  
  
 *# 邻居节点是否超出范围  
 def* verify\_node(*self*, node):  
 px = *self*.calc\_position(node.x)  
 py = *self*.calc\_position(node.y)  
 *if* px < *self*.min\_x:  
 *return False  
 if* py < *self*.min\_y:  
 *return False  
 if* px >= *self*.max\_x:  
 *return False  
 if* py >= *self*.max\_y:  
 *return False  
 if self*.obstacle\_map[node.x][node.y]:  
 *return False  
 return True  
  
 # 计算路径  
 def* calc\_final\_path(*self*, goal\_node, closed\_set):  
 rx = [*self*.calc\_position(goal\_node.x)]  
 ry = [*self*.calc\_position(goal\_node.y)]  
 parent\_index = goal\_node.parent\_index  
 *while* parent\_index != -1:  
 n = closed\_set[parent\_index]   
 rx.append(*self*.calc\_position(n.x))   
 ry.append(*self*.calc\_position(n.y))   
 parent\_index = n.parent\_index   
 *return* rx, ry  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 *# 设置障碍物数量* m = 400  
 *# 设置网格大小* N = 60  
 *# ------------------  
 # 起点和终点* sx = 5.0  
 sy = N - 5.0  
 gx = N - 5.0  
 gy = 5.0  
 *# 网格边长* grid\_size = 1.0  
 *# 无人车半径* radius = 1.0  
 *# 生成障碍物位置* ox, oy = [], []  
 *for* i *in* range(0, N + 1): ox.append(i); oy.append(0) *# 下边界  
 for* i *in* range(0, N + 1): ox.append(N); oy.append(i) *# 右边界  
 for* i *in* range(0, N + 1): ox.append(i); oy.append(N) *# 上边界  
 for* i *in* range(0, N + 1): ox.append(0); oy.append(i) *# 左边界  
 while* m > 0:  
 x = random.randint(1, N)  
 y = random.randint(1, N)  
 *if* x == sx *and* y == sy: *continue  
 if* x == gx *and* y == gy: *continue* ox.append(x)  
 oy.append(y)  
 m = m - 1  
  
 plt.plot(ox, oy, label="obstacle", color='k', linestyle='None', marker='.') *# 障碍物黑色* plt.plot(sx, sy, label="starting point", color='g', linestyle='None', marker='x') *# 起点绿色* plt.plot(gx, gy, label="goal point", color='b', linestyle='None', marker='x') *# 终点蓝色* plt.grid(*True*)  
 plt.axis('equal')  
 plt.legend(loc="upper right")  
 dijkstra = AStar(ox, oy, grid\_size, radius)  
 *# 求解路径* rx, ry = dijkstra.planning(sx, sy, gx, gy)  
 *# 绘制路径* plt.plot(rx, ry, label="shortest path", color='r', linestyle='-')  
 plt.legend(loc="upper right")  
 plt.pause(0.1)  
 plt.show()

# 实验结果

输入：网格大小N=60、障碍物数量m=400

输出：

