

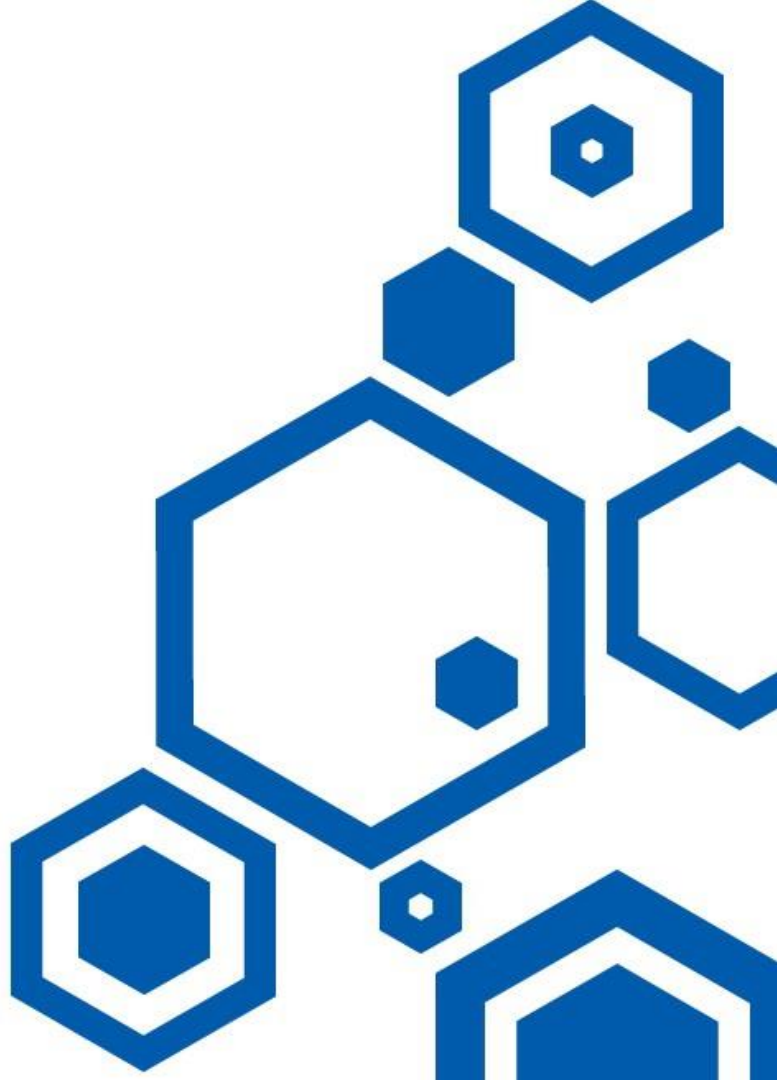


深蓝学院
shenlanxueyuan.com

第二章作业思路分享



主讲人 丁成科



- 1. Matlab部分思路分析
- 2. ros A*部分思路分析

1 把Start_node加入OPENLIST

2 重复以下步骤:

- 1 找到openlist中f最小的节点n
- 2 把节点n从openlist移除, 加入closedlist
- 3 扩散节点n的邻近节点m:
 - a. 若m为障碍物或在closed_list或越界, 则continue下一节点
 - b. 若m不在openlist中则加入openlist中, 并记录父节点、f,h,g等值
 - c. 若m已在openlist中, 则比较旧有的g值和新g值,
若新g值值更小, 则更新节点的数据, 包括父节点、h,g,f等

(4) .直至以下条件时跳出循环

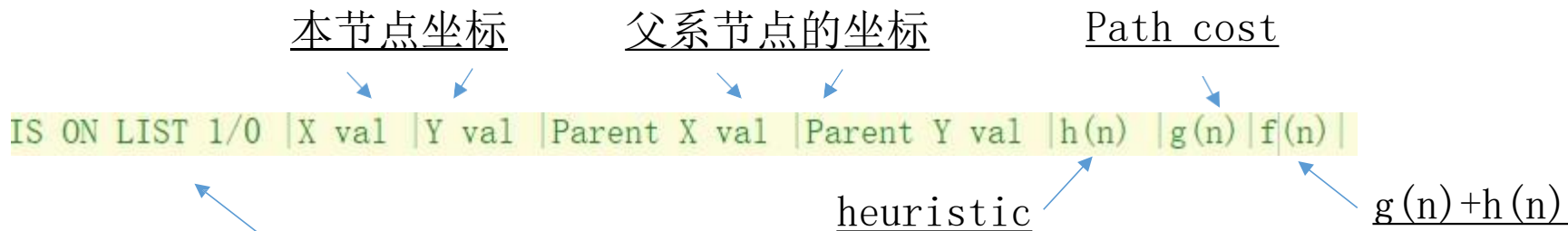
- a. 找到 target_node, 回溯路径
- b. OPENLIST为空, 找不到路径

3 回溯路径

Matlab部分

主要变量：OPEN

1. OPEN的每一行为一个节点，列由以下结构决定



判断是否已被扩展。

1表示在OPEN中没但没被扩展

0表示已扩展，即在CLOSEDLIST中

OPEN								
363x8 double								
	1	2	3	4	5	6	7	8
1	0	1	1	1	1	53.7401	0	53.7401
2	0	2	2	1	1	52.3259	1.4142	53.7401
3	0	1	2	1	1	53.0377	1	54.0377
4	0	3	3	2	2	50.9117	2.8284	53.7401
5	0	3	2	2	2	51.6236	2.4142	54.0379
6	0	3	1	2	2	52.3450	2.8284	55.1734
7	0	2	3	2	2	51.6236	2.4142	54.0379
8	0	1	3	1	2	52.3450	2	54.3450
9	0	4	4	3	3	49.4975	4.2426	53.7401

- **min_fn**: 查找在OPEN中所有未被扩展的节点中(即在OPEN中第一列为1的元素), 其f值最小的节点, 返回该节点在OPEN中所在的行数。若OPEN中的节点全被扩展了(则在OPEN中第一列元素全为0), 则返回-1。
- **expand_array**: 对当前节点进行expand, 注意不会expand越界点、障碍物点和已被扩展点。最后返回neighbors点的坐标和h,g,f, 结构如下, 注意函数返回值为一个矩阵(n*5), n为可neighbors的数量。

```
|X val |Y val ||h(n) |g(n)|f(n)|
```

- **insert_open**: 输入节点x,y,父系节点坐标x,y和f,g,h等数据以产生一行OPEN的数据结构, 这函数主要用于生成新节点数据格式插入OPEN中。

```
while (~isEmpty(OPEN)) % 4(2) 判断OPEN中的节点是否全被扩展了
```

```
% 2(1) 在OPEN中找出f值最小的节点的行索引
```

```
i_min = min_fn(OPEN, OPEN_COUNT, xTarget, yTarget);
```

```
% 2(2) 把节点i_min从open标记为已被弹出, 并加入closedlist
```

```
OPEN(i_min, 1)=0;
```

```
CLOSED_COUNT = CLOSED_COUNT + 1;
```

```
CLOSED(CLOSED_COUNT, 1:2) = OPEN(i_min, 2:3);
```

```
node_x = OPEN(i_min, 2);
```

```
node_y = OPEN(i_min, 3);
```

```
% 4(1) 判断是否找到target_node
```

```
if(node_x == xTarget && node_y == yTarget)
```

```
    NoPath = 0;
```

```
    break;
```

```
end
```

1. 把Start_node加入OPENLIST

2. 重复以下步骤:

1. 找到openlist中f最小的节点n

2. 把节点n从openlist移除, 加入closedlist

3. 扩散节点n的邻近节点m:

a. 若m为障碍物或在closed_list或越界, 则continue下一节点

b. 若m不在openlist中则加入openlist中, 并记录父节点、f,h,g等值

c. 若m已在openlist中, 则比较旧有的g值和新g值,

若新g值更小, 则更新节点的数据, 包括父节点、h,g,f等

4. 直至以下条件时跳出循环

a. 找到 target_node, 回溯路径

b. OPENLIST为空, 找不到路径

isEmpty函数判断在OPEN中第一列是否都为0

isInOpen函数判断在OPEN中是否存在xval,yval坐标点

```
gn = OPEN(i_min, 7);  
% 2(3). 扩散节点node的邻近节点, 返回由邻近节点数据组成的矩阵exp_array  
exp_array = expand_array(node_x, node_y, gn, xTarget, yTarget, CLOSED, MAX_X, MAX_Y);  
n_neighbor = size(exp_array, 1);  
for i = 1:n_neighbor  
    xval = exp_array(i, 1);  
    yval = exp_array(i, 2);  
    hval = exp_array(i, 3);  
    gval = exp_array(i, 4);  
    fval = exp_array(i, 5);  
  
% 3(c)判断expand的节点中是否已在OPEN中, 若是则比较gn值, 择优更新。  
if isInOpen(OPEN, xval, yval)  
    %若在OPEN中, 找出在OPEN的行数, 邻断OPEN中的gn和新neighbor的gn的大小,  
    %可用到node_index函数寻找该点在OPEN的第几行  
    %不提供答案  
end  
else  
    % 3(b). 若不在OPEN中, 把新节点加入OPEN中。可用到insert_open函数  
end  
end
```

1. 把Start_node加入OPENLIST

2. 重复以下步骤:

1. 找到openlist中f最小的节点n

2. 把节点n从openlist移除, 加入closedlist

3. 扩散节点n的邻近节点m:

a. 若m为障碍物或在closed_list或越界, 则continue下一节点

b. 若m不在openlist中则加入openlist中, 并记录父节点、f,h,g等值

c. 若m已在openlist中, 则比较旧有的g值和新g值,
若新g值值更小, 则更新节点的数据, 包括父节点、h,g,f等

4. 直至以下条件时跳出循环

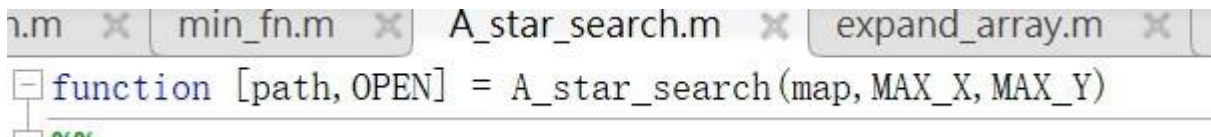
a. 找到 target_node, 回溯路径

b. OPENLIST为空, 找不到路径

- 当找到target后，会把Nopath标记为0，并开始回溯路径
- 由于只在OPEN中记录了父系节点(CLOSEDLIST没有记录父系节点)，
因此需要在OPEN中回溯路径
- 通过从target_node开始找父节点，直至父节点为start_node则完成
- 使用node_index函数获得父系节点的位置
- 查找OPEN的第4,5列获得再之前的父系坐标

```
while true
    path = [xval yval;path];
    n_index = node_index(OPEN, xval, yval);
    xval = OPEN(n_index, 4);
    yval = OPEN(n_index, 5);
    if(xval == xStart && yval == yStart)
        path = [xval yval;path];
        break
    end
end
```


- 额外的小功能:
- 把OPEN也作为A_star_search函数的输出



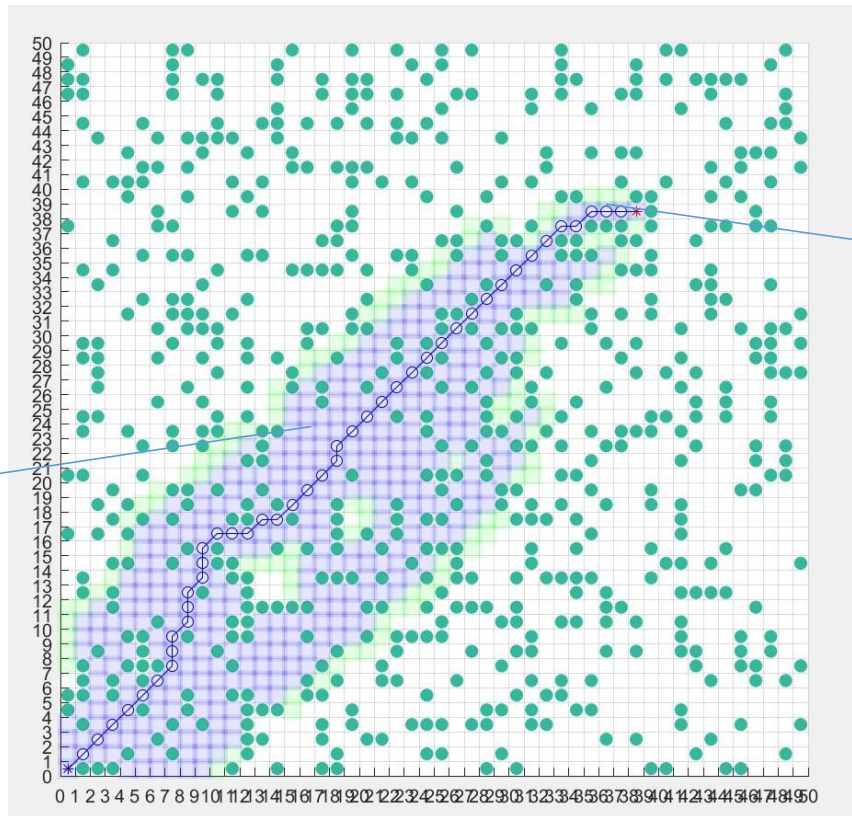
```
function [path, OPEN] = A_star_search(map, MAX_X, MAX_Y)
```

- 输入到visualize_map中，可以直观得出expand的范围。

```
% Waypoint Generator Using the A*  
[path, OPEN] = A_star_search(map, MAX_X, MAX_Y);  
  
% visualize the 2D grid map  
visualize_map(map, path, OPEN);
```

Matlab部分

Expand area



Ros A*部分

节点结构(GridNodePtr):

1 id →判断节点身处位置

(1) . 1→openlist中

(2) . -1→closedlist中

(3) . 0→没被expand

2 Coord 世界坐标

3 Index 栅格坐标

4 gScore 节点path cost

5 fScore = gScore + hScore

6 cameFrom 父系节点

```
typedef GridNode* GridNodePtr;

struct GridNode
{
    int id;           // 1--> open set, -1 --> closed set
    Eigen::Vector3d coord; // world 3D position
    Eigen::Vector3i dir;  // direction of expanding
    Eigen::Vector3i index; // grid 3D position

    double gScore, fScore;
    GridNodePtr cameFrom; // mark the father node
    std::multimap<double, GridNodePtr>::iterator nodeMapIt;

    GridNode(Eigen::Vector3i _index, Eigen::Vector3d _coord){
        id = 0;
        index = _index;
        coord = _coord;
        dir = Eigen::Vector3i::Zero();

        gScore = inf;
        fScore = inf;
        cameFrom = NULL;
    }

    GridNode(){};
    ~GridNode(){};
};
```

- 常用函数: (1). gridIndex2coord() (2). coord2gridIndex()
- Coord 和 index 区别:
 - 实际使用时常常会使用到栅格地图
 - 会把真实世界一定范围的点(coord)假定为同一个栅格(index)(精度问题)
 - 例如resolution=0.2时, 那么可以理解为每隔0.2会有一个新的栅格
 - 代码中使用gridIndex2coord()和coord2gridIndex互换世界坐标和栅格坐标

- 常用函数: (3). isOccupied() (4). isFree()
- isOccupied()判断x,y,z点是否在界内，是否是障碍物，若(x,y,z)栅格在界内并且是障碍物，则返回True
- IsFree()则和isOccupied基本相反，若(x,y,z)栅格在界内并且不是障碍物，才返回True
- isOccupied()和IsFree()在代码中多态，注意输入的变量类型。

getHeu函数

定义点1(x1,y1,z1)和点2(x2,y2,z2),

- $dx = |x1 - x2|$; $dy = |y1 - y2|$; $dz = |z1 - z2|$

- Manhattan = $dx + dy + dz$

- Euclidean = $\sqrt{dx^2 + dy^2 + dz^2}$

```
double detal_x,detal_y,detal_z;  
detal_x = abs(node2->index[0]-node1->index[0]);  
detal_y = abs(node2->index[1]-node1->index[1]);  
detal_z = abs(node2->index[2]-node1->index[2]);  
  
return detal_x+detal_y+detal_z;
```

- Diagonal = $dx + dy + dz - (3 - \sqrt{3}) * \min(dx, dy, dz) - (2 - \sqrt{2}) * (middle(dx, dy, dz) - \min(dx, dy, dz))$

AstarGetSucc函数

- 对节点x,y,z进行26连通的expand
- 能否成为neighbor主要判断以下内容:

- 不扩展本体
- 扩展点不越界
- 不是障碍物点
- 也不希望是id=-1的点

```
for(int i = -1; i < 2; i++){
    for(int j = -1; j < 2; j++){
        for(int k = -1; k < 2; k++){
            if(i!=0 || j!=0 || k!=0){

                int tem_x = i+currentPtr->index[0];
                int tem_y = j+currentPtr->index[1];
                int tem_z = k+currentPtr->index[2];

                if(isFree(tem_x,tem_y,tem_z)){
                    if(GridNodeMap[tem_x][tem_y][tem_z]->id != -1){
                        neighborPtrSets.push_back(GridNodeMap[tem_x][tem_y][tem_z]);
                        edgeCostSets.push_back(pow((i)*(i)+(j)*(j)+(k)*(k),0.5));
                    }
                }
            }
        }
    }
}
```

注意不要new，否则会不清楚该节点的状态(id)，
在后面不能判断是否在OPENLIST中

A*主循环

- OPENSET使用mutimap，记录了fScore和对应的GridNodePtr。
- 由于mutimap会根据key值自动排序，fScore最小的节点可以直接使用begin()获得。弹出后把该指针的id变为-1，并在OPENSET中erase该元素，以保证其后begin()出来的节点一定是fScore最小而且id不是-1。

```
currentPtr = openSet.begin()->second;  
currentPtr->id = -1;  
openSet.erase(openSet.begin());
```


A*主循环

- 对neighbor节点进行分析:
- 1. 若 $id = 0$ ，则表示还没进入过OPENSET中，则在OPENSET中直接更新
- 2. 若 $id = 1$ ，则表示有相同坐目标节点已在OPENSET中，这时候比较新节点的g值和旧有g值，若新g值更优则对节点数据进行更新

```
else if(neighborPtr -> id == 1){  
    if(gCost < neighborPtr->gScore){  
        neighborPtr->fScore = fCost;  
        neighborPtr->gScore = gCost;  
        neighborPtr->cameFrom = currentPtr;  
        openSet.insert(make_pair(neighborPtr->fScore, neighborPtr));  
    }  
    continue;  
}
```

id = 1时的参考代码

路径回溯

- 方法基本同matlab。通过不断查找父系节点，直至父系节点为初始点时结束
- while判断结束的依据可以是当父系节点是空指针时，也可以是gScore=0时
- 这里给出一种基本的实现方式

```
vector<Vector3d> AstarPathFinder::getPath()
{
    vector<Vector3d> path;
    while(terminatePtr != NULL)
    {
        int x = terminatePtr->index(0);
        int y = terminatePtr->index(1);
        int z = terminatePtr->index(2);
        path.push_back(GridNodeMap[x][y][z]->coord);
        terminatePtr = terminatePtr->cameFrom;
    }

    reverse(path.begin(),path.end());

    return path;
}
```



深蓝学院
shenlanxueyuan.com

感谢各位聆听!
Thanks for Listening

