深蓝学院
shenlanxueyuan.com

第二章作业思路分享

主讲人 ******

目录

- MATLAB部分
- ROS部分

A*算法流程

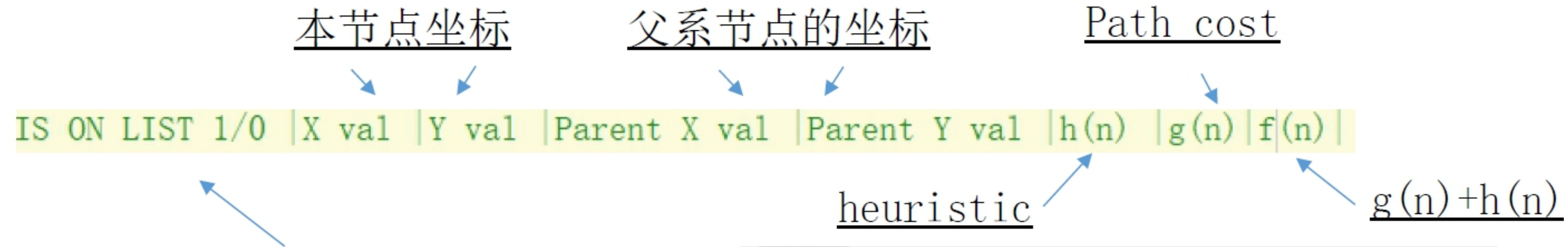
- Maintain a priority queue to store all the nodes to be expanded
- The heuristic function h(n) for all nodes are pre-defined
- The priority queue is initialized with the start state X_S
- Assign $g(X_S)=0$, and g(n)=infinite for all other nodes in the graph
- Loop
  - If the queue is empty, return FALSE; break;
  - Remove the node “n” with the lowest $f(n)=g(n)+h(n)$ from the priority queue
  - Mark node “n” as expanded
  - If the node “n” is the goal state, return TRUE; break;
  - For all unexpanded neighbors “m” of node “n”
    - If g(m) = infinite
      - g(m)= g(n) + Cnm
      - Push node “m” into the queue
    - If $g(m) > g(n) + C_{nm}$
      - g(m)= g(n) + Cnm
  - end
- End Loop

Only difference comparing to Dijkstra’ s algorithm

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



本节点坐标

父系节点的坐标

Path cost

IS ON LIST 1/0 |X val |Y val |Parent X val |Parent Y val |h(n) |g(n)|f(n)|

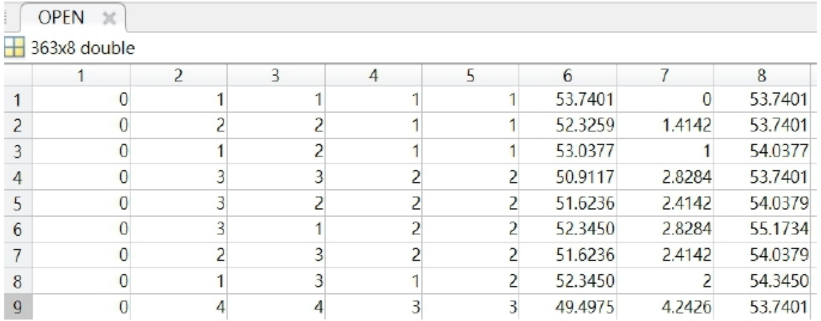
heuristic

g(n)+h(n)

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

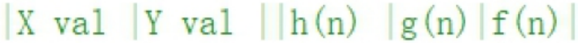


363x8 double

	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

MATLAB部分

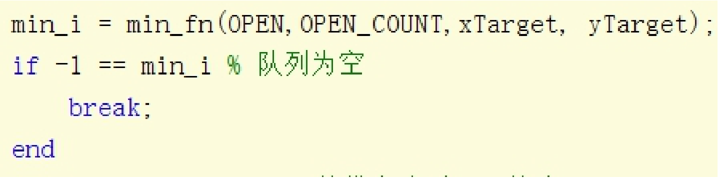
- **min_fn:** 查找在OPEN中所有未被扩展的节点中(即在OPEN中第一列为1的元素)，其f值最小的节点，返回该节点在OPEN中所在的行数。若OPEN中的节点全被扩展了(则在OPEN中第一列元素全为0)，则返回-1。
- **expand_array:** 对当前节点进行expand，注意不会expand越界点、障碍物点和已被扩展点。最后返回neighbors点的坐标和h,g,f，结构如下，注意函数返回值为一个矩阵(n*5)，n为可neighbors的数量。



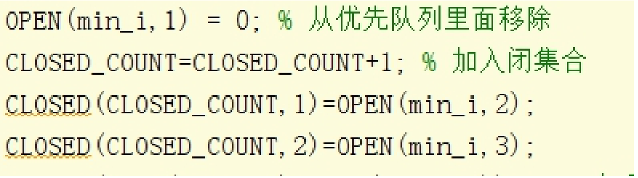
```
|X val |Y val ||h(n) |g(n)|f(n)|
```

- **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。

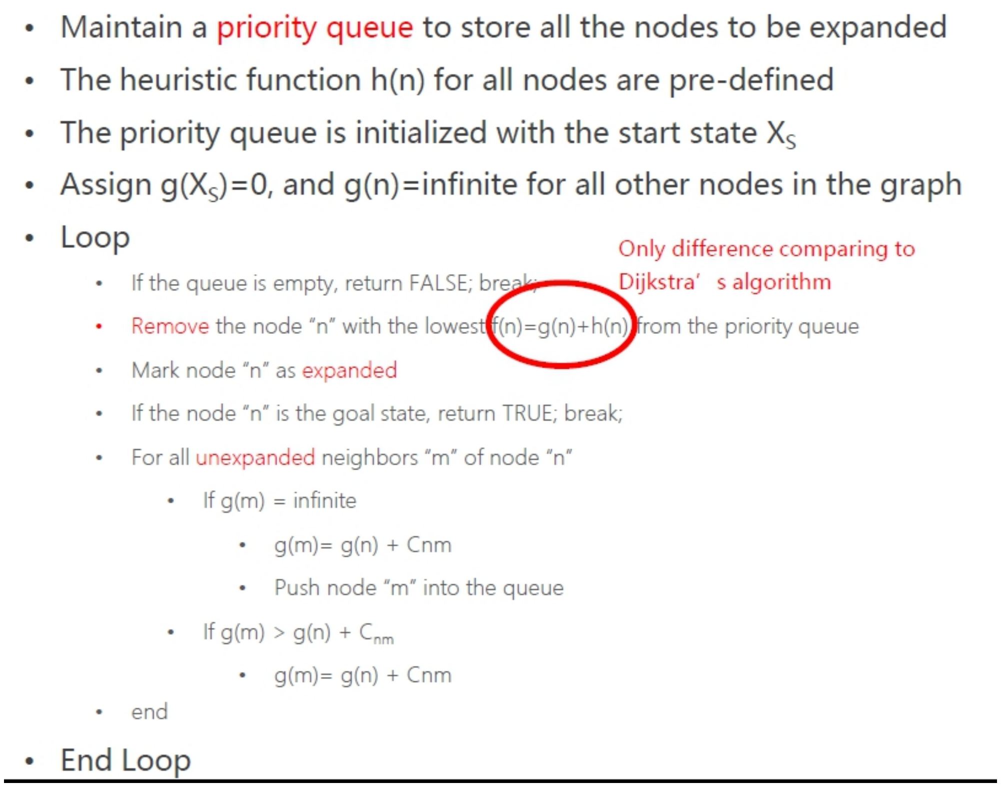
MATLAB部分



```
min_i = min_fn(OPEN,OPEN_COUNT,xTarget, yTarget);
if -1 == min_i % 队列为空
    break;
end
```



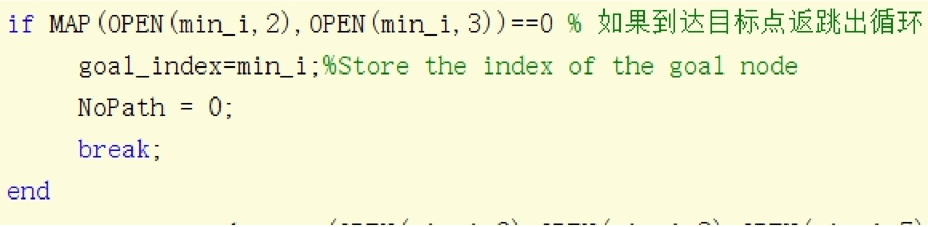
```
OPEN(min_i,1) = 0; % 从优先队列里面移除
CLOSED_COUNT=CLOSED_COUNT+1; % 加入闭集合
CLOSED(CLOSED_COUNT,1)=OPEN(min_i,2);
CLOSED(CLOSED_COUNT,2)=OPEN(min_i,3);
```



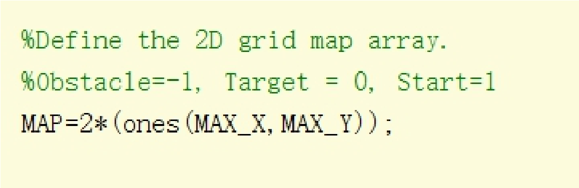
- Maintain a priority queue to store all the nodes to be expanded
- The heuristic function h(n) for all nodes are pre-defined
- The priority queue is initialized with the start state X_S
- Assign $g(X_S)=0$, and g(n)=infinite for all other nodes in the graph
- Loop
    - If the queue is empty, return FALSE; break;
    - Remove the node "n" with the lowest f(n)=g(n)+h(n) from the priority queue
    - Mark node "n" as expanded
    - If the node "n" is the goal state, return TRUE; break;
    - For all unexpanded neighbors "m" of node "n"
        - If g(m) = infinite
            - g(m)= g(n) + Cnm
            - Push node "m" into the queue
        - If $g(m) > g(n) + C_{nm}$
            - g(m)= g(n) + Cnm
    - end
- End Loop

Only difference comparing to Dijkstra' s algorithm

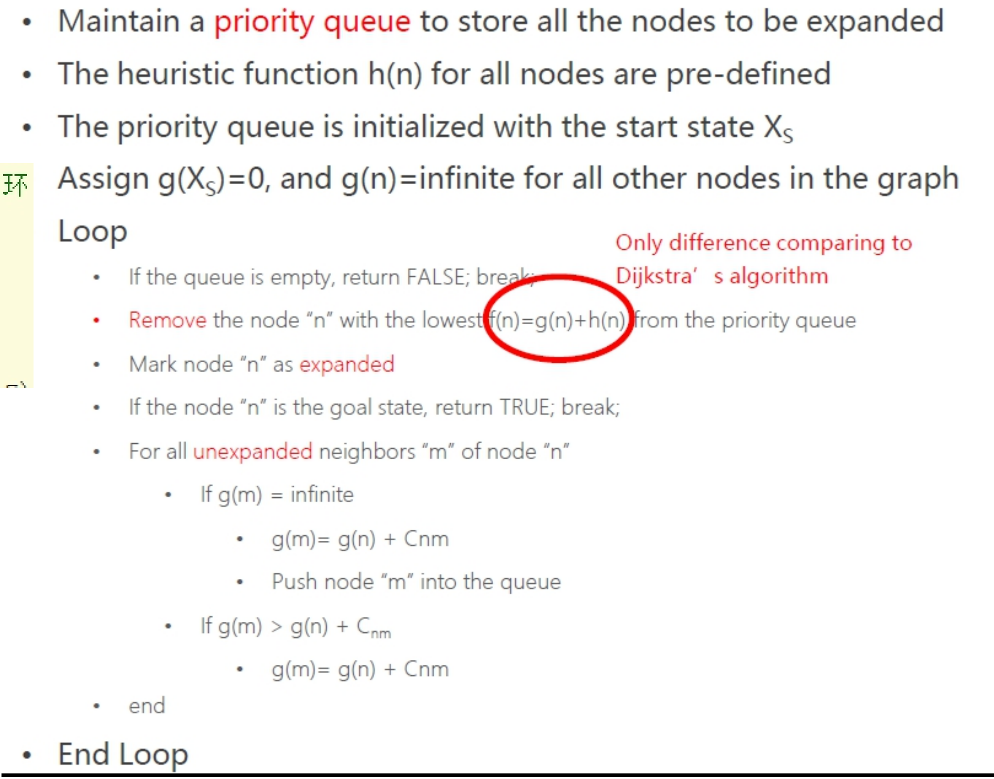
MATLAB部分



```
if MAP(OPEN(min_i,2),OPEN(min_i,3))==0 % 如果到达目标点返跳出循环
     goal_index=min_i;%Store the index of the goal node
     NoPath = 0;
     break;
end
```



```
%Define the 2D grid map array.
%Obstacle=-1, Target = 0, Start=1
MAP=2*(ones(MAX_X,MAX_Y));
```

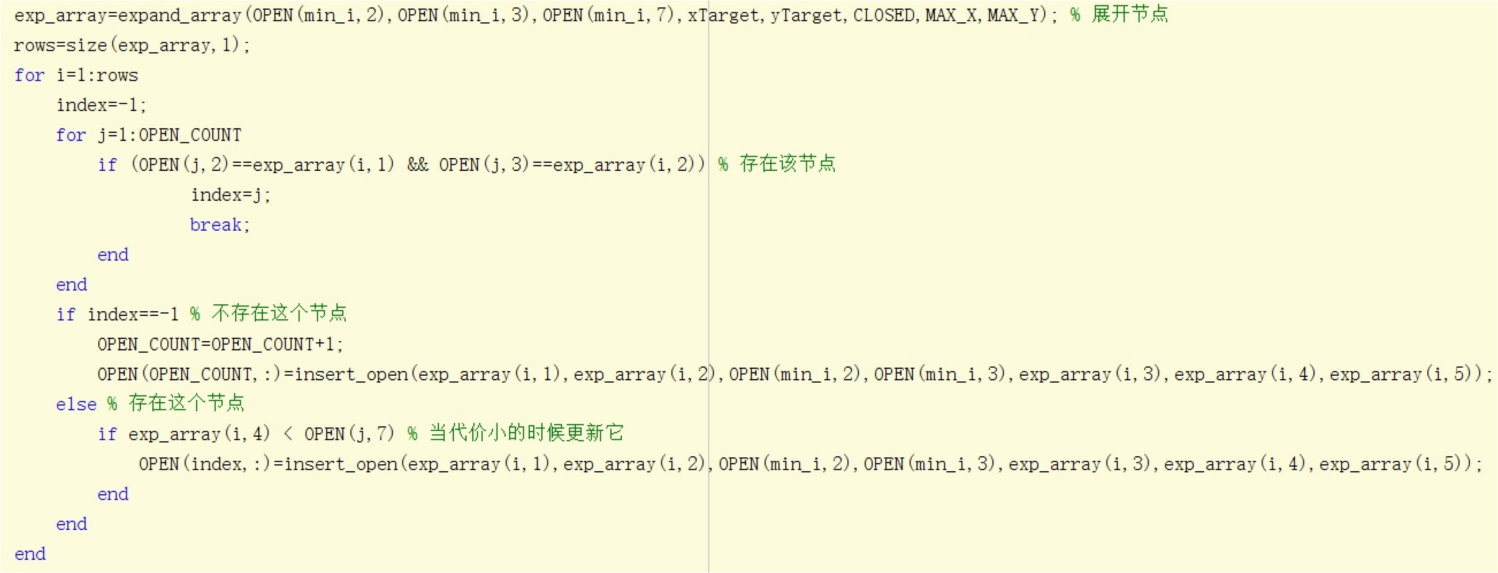


```
• Maintain a priority queue to store all the nodes to be expanded
• The heuristic function $h(n)$ for all nodes are pre-defined
• The priority queue is initialized with the start state $X_S$
• Assign $g(X_S)=0$, and $g(n)$=infinite for all other nodes in the graph
• Loop
    • If the queue is empty, return FALSE; break;
    • Remove the node "n" with the lowest $f(n)=g(n)+h(n)$ from the priority queue
    • Mark node "n" as expanded
    • If the node "n" is the goal state, return TRUE; break;
    • For all unexpanded neighbors "m" of node "n"
        • If $g(m)$ = infinite
            • $g(m)= g(n) + Cnm$
            • Push node "m" into the queue
        • If $g(m) > g(n) + C_{nm}$
            • $g(m)= g(n) + Cnm$
    • end
• End Loop
```

Only difference comparing to Dijkstra’ s algorithm

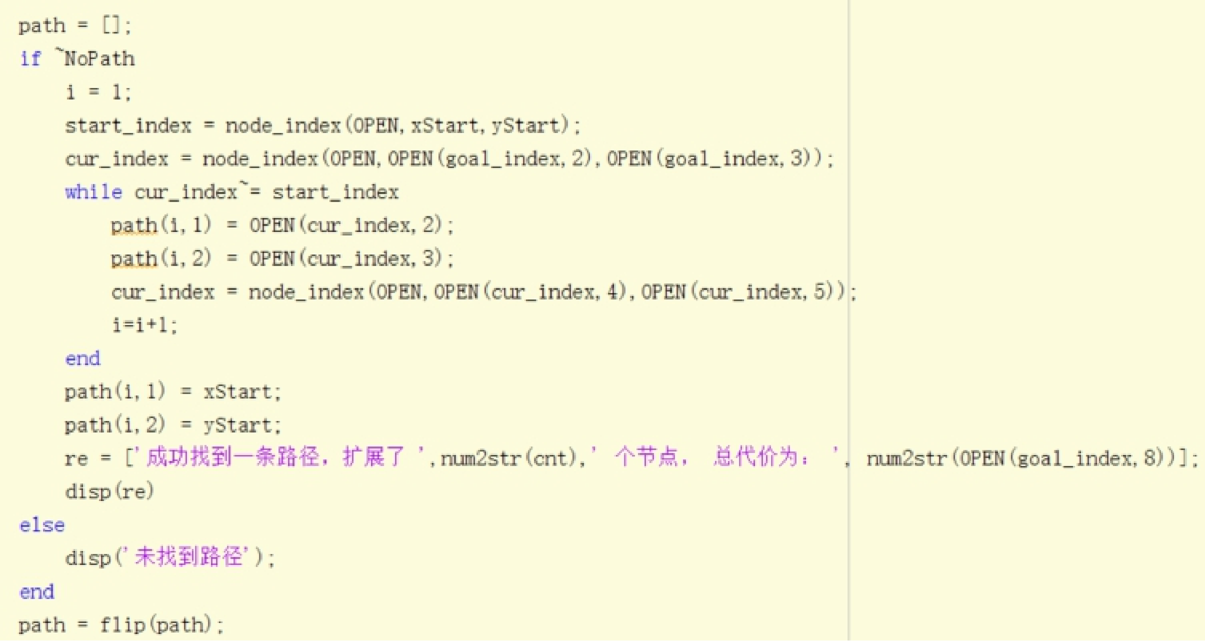
MATLAB部分

- For all unexpanded neighbors “m” of node “n”
  - If g(m) = infinite
    - g(m)= g(n) + Cnm
    - Push node “m” into the queue
  - If g(m) > g(n) + C_{nm}
    - g(m)= g(n) + Cnm
- end

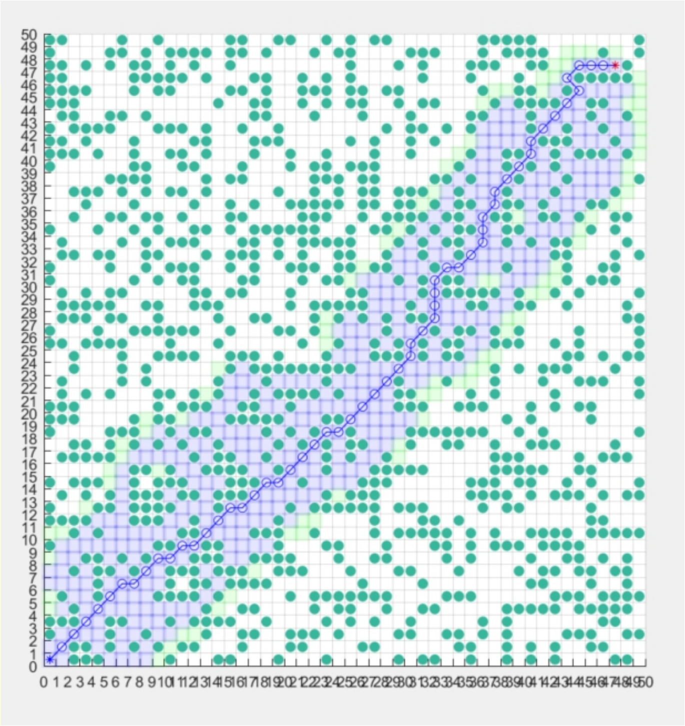


```
exp_array=expand_array(OPEN(min_i,2),OPEN(min_i,3),OPEN(min_i,7),xTarget,yTarget,CLOSED,MAX_X,MAX_Y); % 展开节点
rows=size(exp_array,1);
for i=1:rows
    index=-1;
    for j=1:OPEN_COUNT
        if (OPEN(j,2)==exp_array(i,1) && OPEN(j,3)==exp_array(i,2)) % 存在该节点
                index=j;
                break;
        end
    end
    if index==-1 % 不存在这个节点
        OPEN_COUNT=OPEN_COUNT+1;
        OPEN(OPEN_COUNT,:)=insert_open(exp_array(i,1),exp_array(i,2),OPEN(min_i,2),OPEN(min_i,3),exp_array(i,3),exp_array(i,4),exp_array(i,5));
    else % 存在这个节点
        if exp_array(i,4) < OPEN(j,7) % 当代价小的时候更新它
            OPEN(index,:)=insert_open(exp_array(i,1),exp_array(i,2),OPEN(min_i,2),OPEN(min_i,3),exp_array(i,3),exp_array(i,4),exp_array(i,5));
        end
    end
end
```

MATLAB部分



```
path = [];
if ~NoPath
    i = 1;
    start_index = node_index(OPEN,xStart,yStart);
    cur_index = node_index(OPEN,OPEN(goal_index,2),OPEN(goal_index,3));
    while cur_index~= start_index
        path(i,1) = OPEN(cur_index,2);
        path(i,2) = OPEN(cur_index,3);
        cur_index = node_index(OPEN,OPEN(cur_index,4),OPEN(cur_index,5));
        i=i+1;
    end
    path(i,1) = xStart;
    path(i,2) = yStart;
    re = ['成功找到一条路径，扩展了 ',num2str(cnt),' 个节点， 总代价为： ', num2str(OPEN(goal_index,8))];
    disp(re)
else
    disp('未找到路径');
end
path = flip(path);
```



ROS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

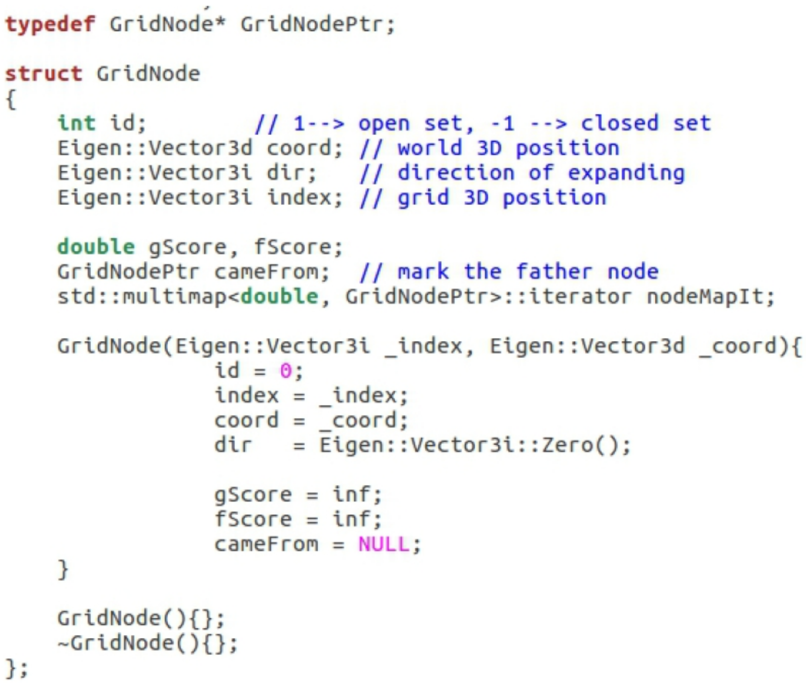
2 Coord 世界坐标

3 Index 栅格坐标

4 gScore 节点path cost

5 fScore = gScore +hScore

6 cameFrom 父系节点

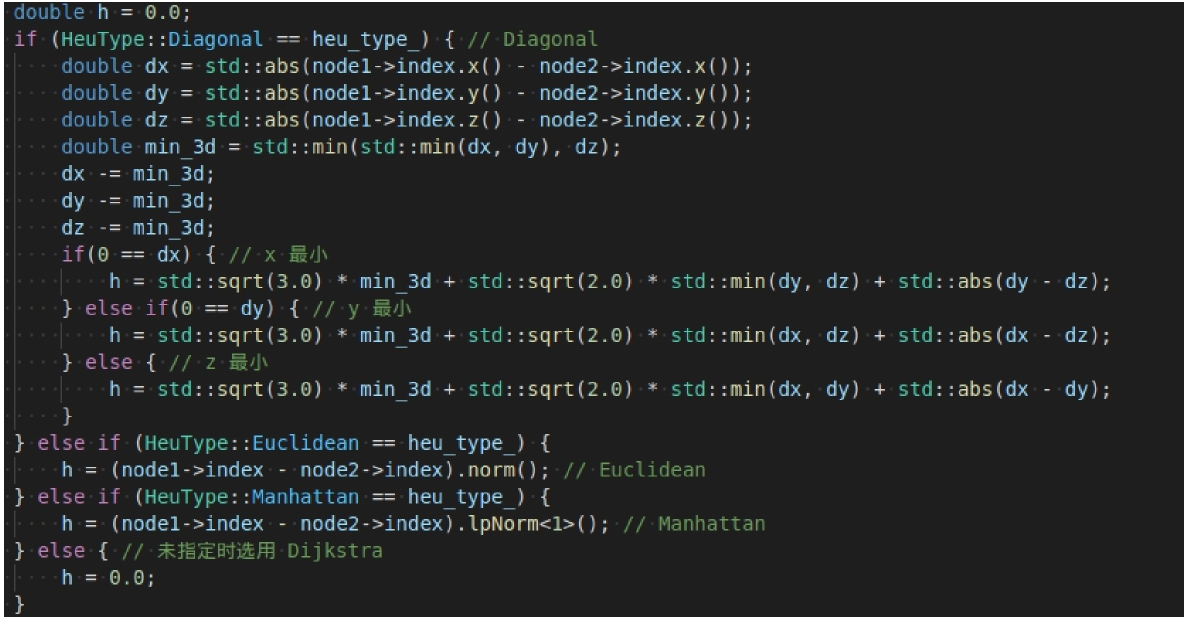


```
typedef GridNode* GridNodePtr;

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(){};
};
```

ROS部分

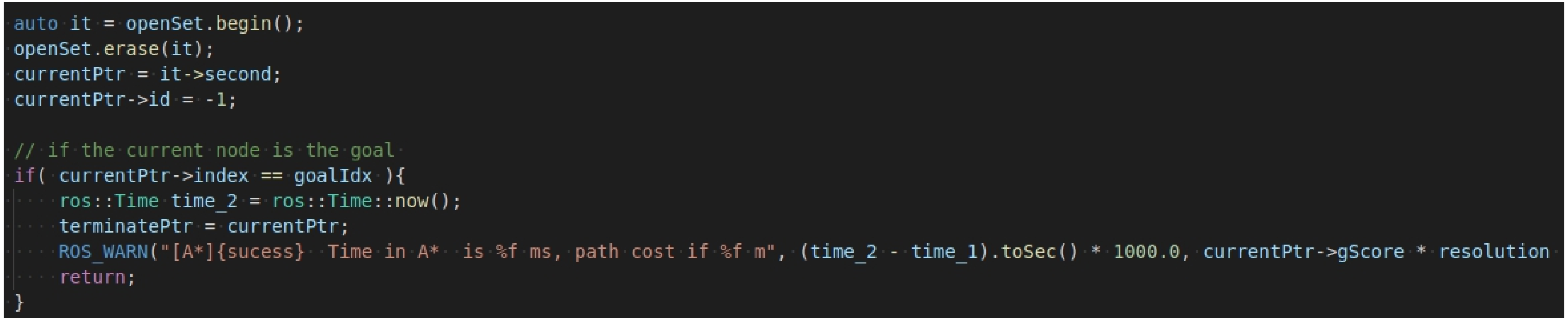
- 常用函数: (3). isOccupied() (4). isFree()
- isOccupied()判断x,y,z点是否在界内，是否是障碍物，若(x,y,z)栅格在界内并且是障碍物，则返回True
- IsFree()则和isOccupied基本相返，若(x,y,z)栅格在界内并且不是障碍物，才返回True
- isOccupied()和IsFree()在代码中多态，注意输入的变量类型。

ROS部分



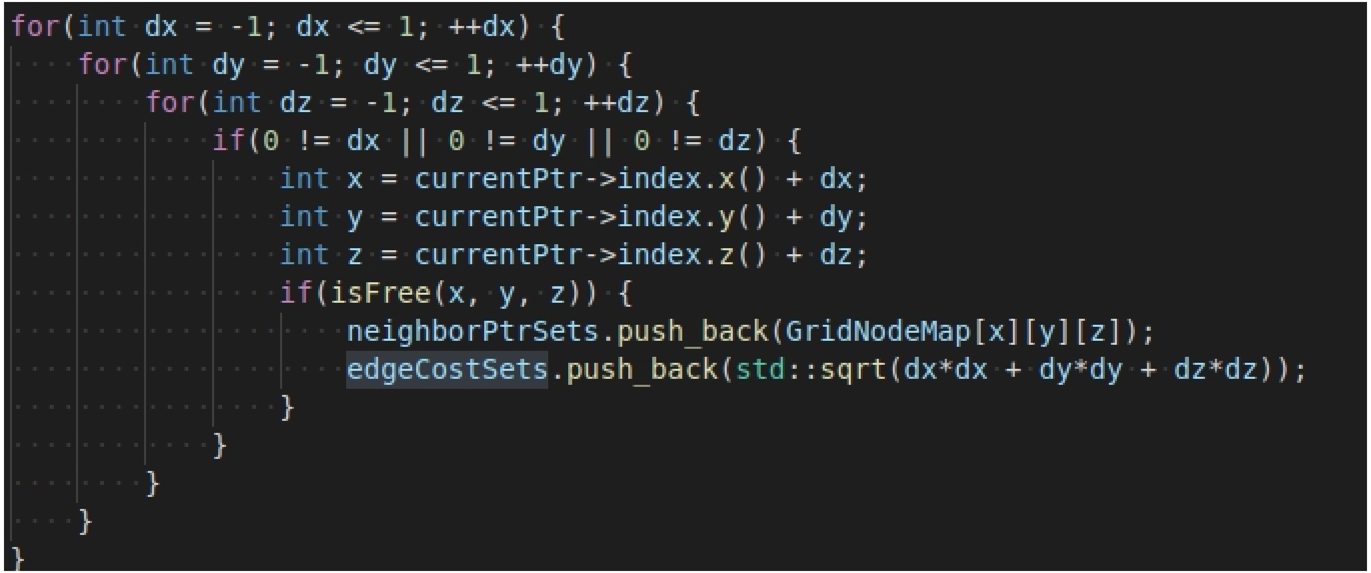
```
double h = 0.0;
if (HeuType::Diagonal == heu_type_) { // Diagonal
    double dx = std::abs(node1->index.x() - node2->index.x());
    double dy = std::abs(node1->index.y() - node2->index.y());
    double dz = std::abs(node1->index.z() - node2->index.z());
    double min_3d = std::min(std::min(dx, dy), dz);
    dx -= min_3d;
    dy -= min_3d;
    dz -= min_3d;
    if(0 == dx) { // x 最小
        h = std::sqrt(3.0) * min_3d + std::sqrt(2.0) * std::min(dy, dz) + std::abs(dy - dz);
    } else if(0 == dy) { // y 最小
        h = std::sqrt(3.0) * min_3d + std::sqrt(2.0) * std::min(dx, dz) + std::abs(dx - dz);
    } else { // z 最小
        h = std::sqrt(3.0) * min_3d + std::sqrt(2.0) * std::min(dx, dy) + std::abs(dx - dy);
    }
} else if (HeuType::Euclidean == heu_type_) {
    h = (node1->index - node2->index).norm(); // Euclidean
} else if (HeuType::Manhattan == heu_type_) {
    h = (node1->index - node2->index).lpNorm<1>(); // Manhattan
} else { // 未指定时选用 Dijkstra
    h = 0.0;
}
```

ROS部分



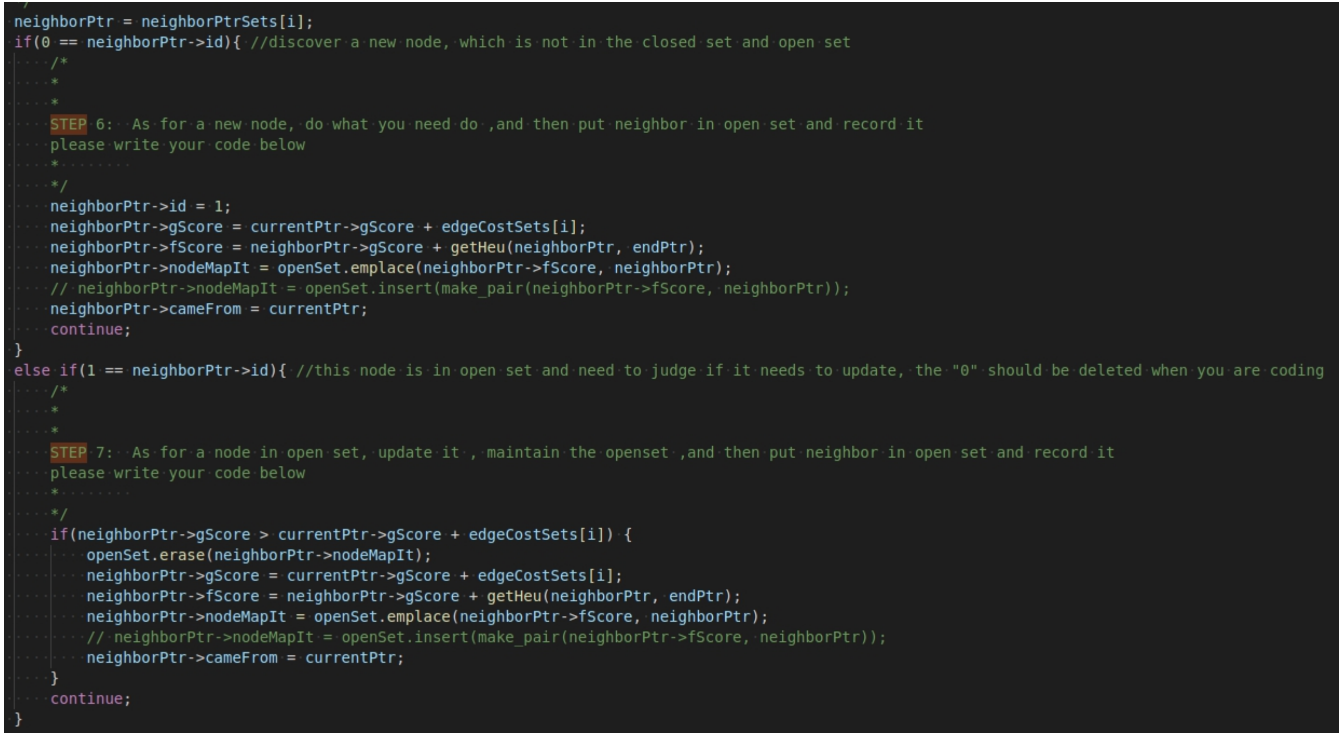
```
auto it = openSet.begin();
openSet.erase(it);
currentPtr = it->second;
currentPtr->id = -1;

// if the current node is the goal
if( currentPtr->index == goalIdx ){
    ros::Time time_2 = ros::Time::now();
    terminatePtr = currentPtr;
    ROS_WARN("[A*]{sucess}  Time in A*  is %f ms, path cost if %f m", (time_2 - time_1).toSec() * 1000.0, currentPtr->gScore * resolution
    return;
}
```

ROS部分



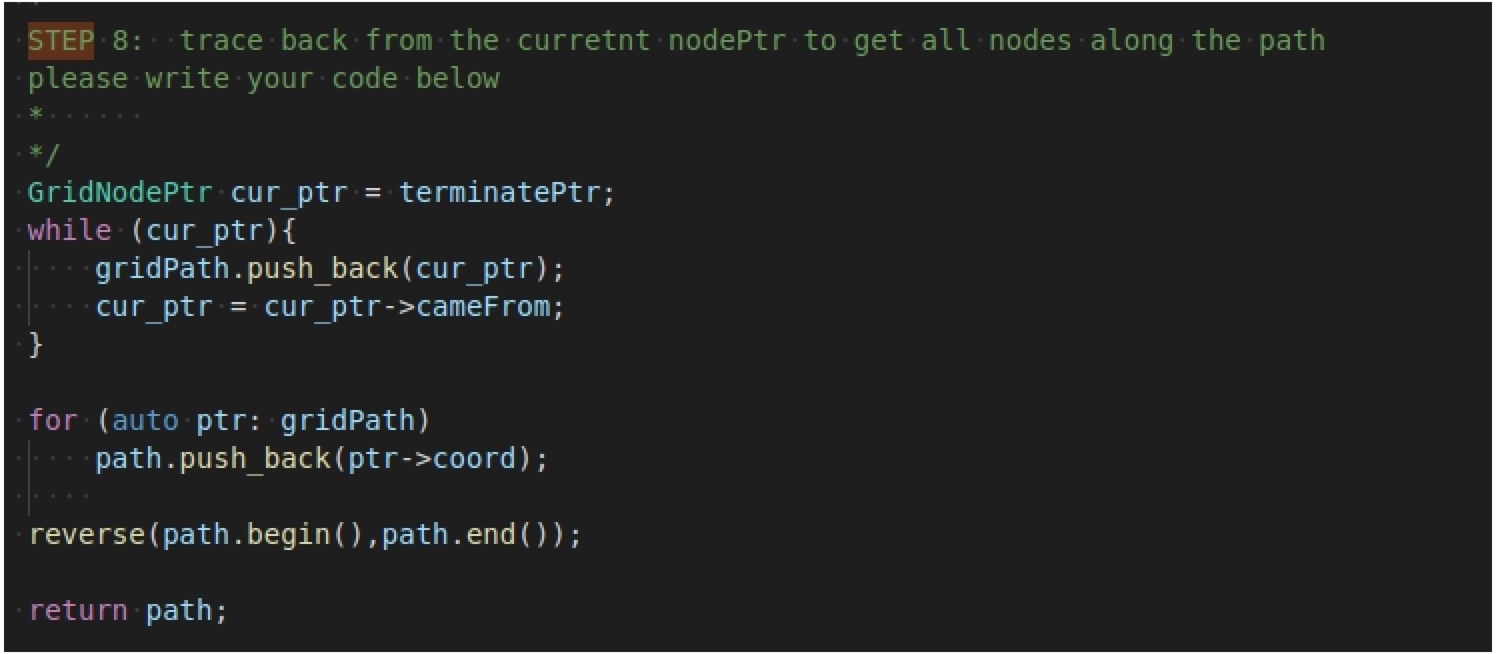
```
for(int dx = -1; dx <= 1; ++dx) {
    for(int dy = -1; dy <= 1; ++dy) {
        for(int dz = -1; dz <= 1; ++dz) {
            if(0 != dx || 0 != dy || 0 != dz) {
                int x = currentPtr->index.x() + dx;
                int y = currentPtr->index.y() + dy;
                int z = currentPtr->index.z() + dz;
                if(isFree(x, y, z)) {
                    neighborPtrSets.push_back(GridNodeMap[x][y][z]);
                    edgeCostSets.push_back(std::sqrt(dx*dx + dy*dy + dz*dz));
                }
            }
        }
    }
}
```

ROS部分



```
neighborPtr = neighborPtrSets[i];
if(0 == neighborPtr->id){ //discover a new node, which is not in the closed set and open set
    /*
    *
    *
    STEP 6:  As for a new node, do what you need do ,and then put neighbor in open set and record it
    please write your code below
    *        
    */
    neighborPtr->id = 1;
    neighborPtr->gScore = currentPtr->gScore + edgeCostSets[i];
    neighborPtr->fScore = neighborPtr->gScore + getHeu(neighborPtr, endPtr);
    neighborPtr->nodeMapIt = openSet.emplace(neighborPtr->fScore, neighborPtr);
    // neighborPtr->nodeMapIt = openSet.insert(make_pair(neighborPtr->fScore, neighborPtr));
    neighborPtr->cameFrom = currentPtr;
    continue;
}
else if(1 == neighborPtr->id){ //this node is in open set and need to judge if it needs to update, the "0" should be deleted when you are coding
    /*
    *
    *
    STEP 7:  As for a node in open set, update it , maintain the openset ,and then put neighbor in open set and record it
    please write your code below
    *        
    */
    if(neighborPtr->gScore > currentPtr->gScore + edgeCostSets[i]) {
        openSet.erase(neighborPtr->nodeMapIt);
        neighborPtr->gScore = currentPtr->gScore + edgeCostSets[i];
        neighborPtr->fScore = neighborPtr->gScore + getHeu(neighborPtr, endPtr);
        neighborPtr->nodeMapIt = openSet.emplace(neighborPtr->fScore, neighborPtr);
        // neighborPtr->nodeMapIt = openSet.insert(make_pair(neighborPtr->fScore, neighborPtr));
        neighborPtr->cameFrom = currentPtr;
    }
    continue;
}
```

ROS部分



```
STEP 8:  trace back from the curretnt nodePtr to get all nodes along the path
please write your code below
*      
*/
GridNodePtr cur_ptr = terminatePtr;
while (cur_ptr){
    gridPath.push_back(cur_ptr);
    cur_ptr = cur_ptr->cameFrom;
}

for (auto ptr: gridPath)
    path.push_back(ptr->coord);
    
reverse(path.begin(),path.end());

return path;
```

ROS部分

有无Tie Breaker	搜索遍历的节点数（个）	运行时间（ms）	路径代价（m）
无	61	0.413702	6.905382
有	27	0.202895	6.905382

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening