深蓝学院
shenlanxueyuan.com

移动机器人运动规划第二章学习分享

主讲人 JT

纲要

深蓝学院
shenlanxueyuan.com

- 第一部分：算法
- 第二部分：仿真实现
- 第三部分：Q&A

算法

- $Dijkstra$

$$g(n) = g(m) + Cost_{mn}$$

- A^*

$$f(n) = g(n) + h(n)$$

启发函数h的选择：

$$Manhattan: h(n) = \Delta x + \Delta y + \Delta z$$

$$Euclidean: h(n) = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$$

$$Diagonal: h(n) = \Delta x + \Delta y + \Delta z + \left(\sqrt{3} - 3\right)\min(\Delta x, \Delta y, \Delta z) + \left(\sqrt{2} - 2\right)[\mathrm{mid}(\Delta x, \Delta y, \Delta z) - \min(\Delta x, \Delta y, \Delta z)]$$

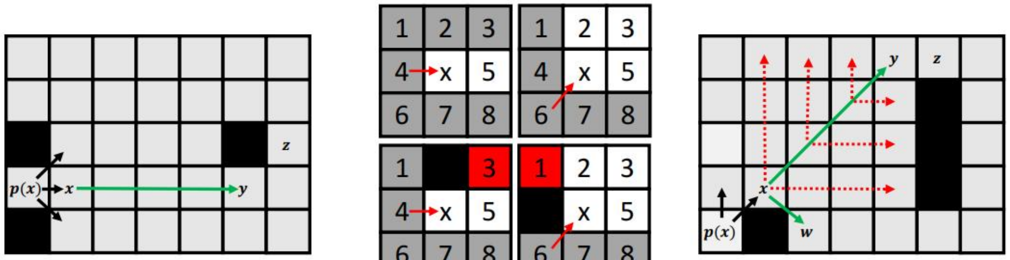
算法

- *JPS*

基于A^*改进，一种扩展搜索节点的策略，提升复杂环境下的搜索效率。（具体见课件）

Jumping Rules

Look Ahead Rule



Jumping Straight

Jumping Diagonally

- Recursively apply straight pruning rule and identify y as a jump point successor of **x**. This node is interesting because it has a neighbor z that cannot be reached optimally except by a path that visits **x** then **y**.
- Recursively apply the diagonal pruning rule and identify y as a jump point successor of **x**.
- Before each diagonal step we first recurse straight. Only if both straight recursions fail to identify a jump point do we step diagonally again.
- Node w, a forced neighbor of **x**, is expanded as normal. (also push into the open list, the priority queue)

算法

- A^*算法在工程应用上的trick

A^*算法在实际搜索的过程中会有很多代价相同的路径，这会降低搜索的效率，核心思想是打破路径间的对称性。

$$h = h + cross \times 0.001$$

$$cross = abs(\Delta x_1 \Delta y_2 - \Delta x_2 \Delta y_1 + \Delta x_1 \Delta z_2 - \Delta x_2 \Delta z_1 + \Delta y_1 \Delta z_2 - \Delta y_2 \Delta z_1)$$

$$\Delta x_1 = abs\left(x_n - x_g\right) \Delta x_2 = abs(x_s - x_n)$$

$$\Delta y_1 = abs\left(y_n - y_g\right) \Delta y_2 = abs(y_s - y_n)$$

$$\Delta z_1 = abs\left(z_n - z_g\right) \Delta z_2 = abs(z_s - z_n)$$

纲要

深蓝学院
shenlanxueyuan.com

- 第一部分：算法
- **第二部分：仿真实现**
- 第三部分：Q&A

程序

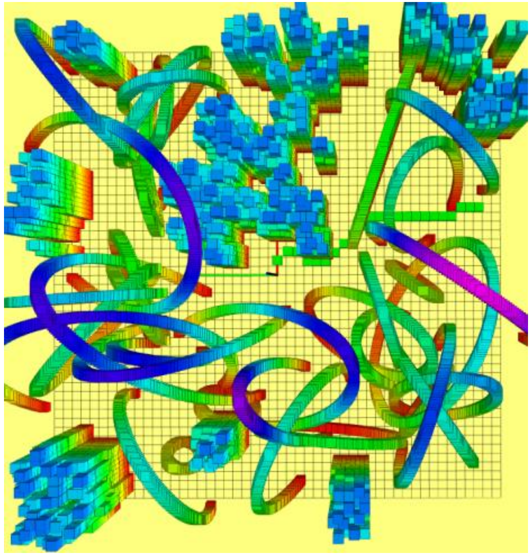
深蓝学院
shenlanxueyuan.com

- A^*算法框架

```
startNode.g = 0;
startNode.h = heuristicFunc(startNode, goalNode);
startNode.f = startNode.g + startNode.h;
open.insert(startNode);//起点加入open集合中
while(open.empty() == false){
    currentNode = open.begin();//从open中弹出f(n)值最小的节点n;
    if(currentNode == goalNode){//如果到了目标节点
        goalNode.fatherNode = currentNode.fatherNode;//目标节点的父节点替换为当前节点，用于回溯轨迹
        break;
    }
    close.insert(currentNode);//将n加入close队列;
    allNeighborNodes = getNeighborNodes(currentNode);//扩展n节点的邻居neighbors;
    for(neighbor in allNeighborNodes){
        if(neighbor not in open && neighbor not in close){//如果邻居节点是没有访问过的节点
            neighbor.g = currentNode.g + distance(neighbor, currentNode);//更新g值
            neighbor.h = heuristicFunc(neighbor, goalNode);//更新h值
            neighbor.f = neighbor.g + neighbor.h;//更新f值
            neighbor.fatherNode = currentNode;//更新父节点
            open.insert(neighbor);
        }
        else if(neighbor in open){//如果邻居节点是在open集合中的节点
            if(neighbor.g >= currentNode.g + distance(neighbor, currentNode)){//如果邻居节点原本的g大于当前节点的g加上当前节点到邻居节点的距离
                neighbor.g = currentNode.g + distance(neighbor, currentNode);//更新g值
                neighbor.h = heuristicFunc(neighbor, goalNode);//更新h值
                neighbor.f = neighbor.g + neighbor.h;//更新h值
                neighbor.fatherNode = currentNode;//更新父节点
            }
        }
        else if(neighbor in close){
            continue;//对于已经在close集合中的节点不作处理，即跳过;
        }
    }
}
path = backtrack(goalNode);
```

程序

●A^*运行效果



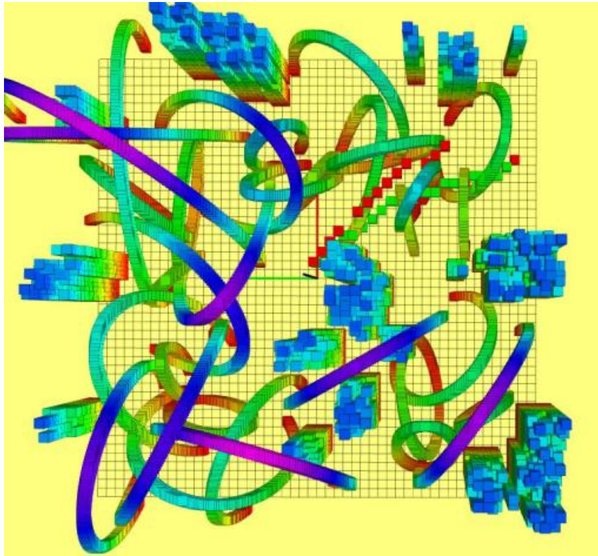
```
[ WARN] [1667206917.710253048]: 3D Goal Set
[ INFO] [1667206917.721034701]: [node] receive the planning target
[ WARN] [1667206917.722677686]: [ManhattanHeu A*]{sucess}  Time in A*  is 0.0986
88 ms, path cost if 1.163826 m
[ WARN] [1667206917.723258150]: visited_nodes size : 50
[ WARN] [1667206917.725409592]: [EuclideanHeu A*]{sucess}  Time in A*  is 1.3793
98 ms, path cost if 1.099547 m
[ WARN] [1667206917.726359106]: visited_nodes size : 844
[ WARN] [1667206917.727274119]: DiagnalHeu [A*]{sucess}  Time in A*  is 0.150093
 ms, path cost if 1.140394 m
[ WARN] [1667206917.728217347]: visited_nodes size : 57
```

```
[ WARN] [1667218151.866045304]: 3D Goal Set
[ INFO] [1667218151.873664241]: [node] receive the planning target
[ WARN] [1667218151.875891930]: [EuclideanHeu A* without tieBreaker]{sucess}  Ti
me in A*  is 1.114020 ms, path cost if 0.973697 m
[ WARN] [1667218151.877711640]: visited_nodes size : 653
[ WARN] [1667218151.879661942]: [EuclideanHeu A* with tieBreaker]{sucess}  Time
in A*  is 1.064995 ms, path cost if 0.973697 m
[ WARN] [1667218151.880697396]: visited_nodes size : 603
```

- A^*算法的运行效果如上图左边所示；
- 上图右边分别是不同启发函数和是否使用tieBreaker的运行效果。

程序

●运行效果

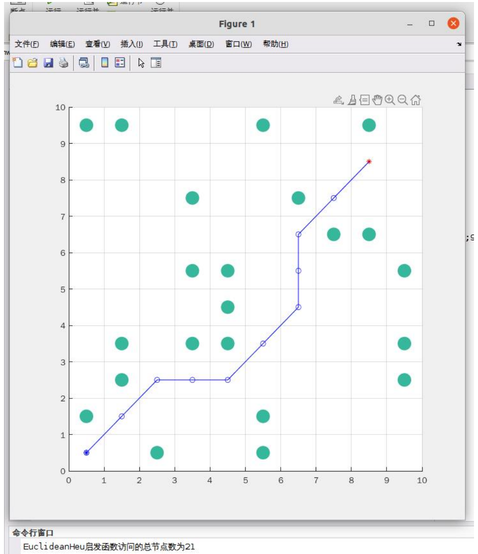


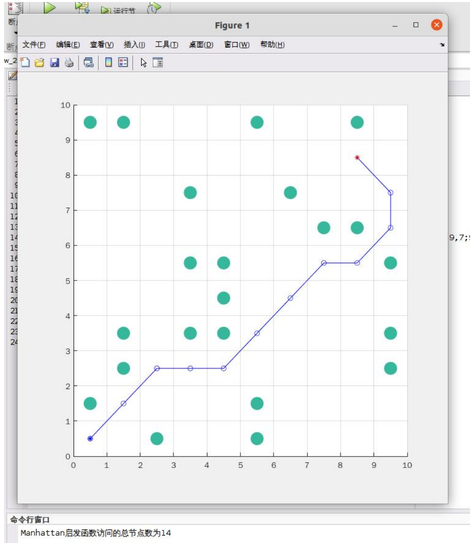
```
[ WARN] [1667221241.837571875]: 3D Goal Set
[ INFO] [1667221241.844091374]: [node] receive the planning target
[ WARN] [1667221241.847724943]: [EuclideanHeu A* without tieBreaker]{sucess}  Ti
me in A*  is 3.514492 ms, path cost if 1.198959 m
[ WARN] [1667221241.848577271]: visited_nodes size : 1063
[ WARN] [1667221241.855886173]: [JPS]{sucess}  Time in JPS  is 6.067146 ms, path
 cost if 4.394793 m
[ WARN] [1667221241.857347856]: visited_nodes size : 3966
```

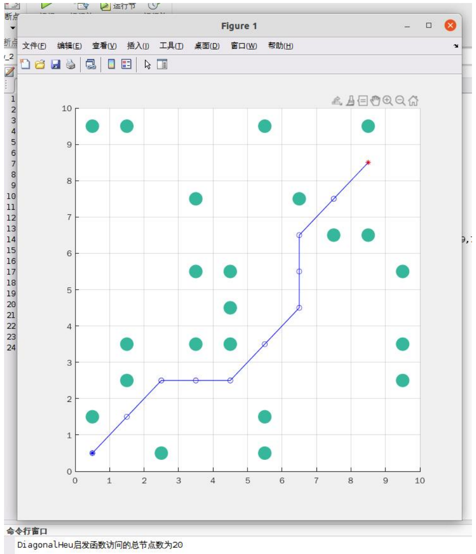
- JPS算法的运行效果如上图左边所示；
- 上图右边是A^*算法和JPS运行结果对比。

程序

●Matlab仿真结果







在matlab仿真下可以更直观地显示出不同启发式函数的路径搜索效果

程序

- 结果分析
  - 使用Manhattan距离作为启发函数得到的路径为非最优，但是访问总结点数最少，因为Manhattan函数值相对较大，更加偏向与贪心。
  - 使用Euclidean距离作为启发函数得到的路径为最优，但是访问的总结点数最多，因为欧式距离为严格最小的h值。
  - 使用对角距离作为启发函数得到路径为最优，且访问的总结点数相对欧式距离少。因为在栅格地图下，对角函数得到的为最小的h值，因此路径最优；对角距离值会大于欧式距离，因此访问节点数少。

作业要求

- 作业内容：提交完整可编译运行的程序功能包grid_path_searcher；撰写一篇说明文档；算法流程及运行效果；对比不同启发函数对A*运行效率的影响
- 评价标准：
  - 及格：补全代码，且A*路径搜索功能正常
  - 良好：在及格的基础上对比不同启发函数对A*效率的影响
  - 优秀：在良好的基础上，加入关于JPS算法的对比，并撰写说明文档，对作业的流程进行详细的描述

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening