目录

[Optimal A\*在三种不同场景下生成的路径截图 1](#_Toc68206451)

[采用的Heuristic Function 1](#_Toc68206452)

[Dijkstra VS Normal A\* VS Optimal A\* 2](#_Toc68206453)

[搜索方法 2](#_Toc68206454)

[数据样本： 2](#_Toc68206455)

[结果分析 2](#_Toc68206456)

[程序补充说明 2](#_Toc68206457)

[Optimal A\* VS JPS 3](#_Toc68206458)

[采用的Heuristic Function 4](#_Toc68206459)

[两种路径搜索方法的对比 4](#_Toc68206460)

[数据背景： 4](#_Toc68206461)

[数据分析 5](#_Toc68206462)

[遇到的问题和解决方法 5](#_Toc68206463)

## Optimal A\*在三种不同场景下生成的路径截图

图表, 折线图

描述已自动生成图表, 散点图

描述已自动生成图表, 散点图

描述已自动生成

### 采用的Heuristic Function

Diagonal heuristic

文本, 信件

描述已自动生成

## Dijkstra VS Normal A\* VS Optimal A\*

搜索方法：Dijkstra, 普通A\*, 采用最优解作为启发方程的optimal A\*

数据样本：在三种不同的复杂度场景下，基于三种搜索方法各进行五次路径生成，记录每次路径生成时探索的节点个数并取平均值。

图表, 条形图, 瀑布图

描述已自动生成

### 结果分析

在复杂度不高的场景下，搜索方法采用的启发方程越接近真实最短路径即h(n) ->h\*(n)，其生成最优路径的效率越高。

然而在复杂场景下，没有统一的能够准确计算真实最短路径的理论解，而且三种方法的启发式方程的结果都与远小于真实最短路径h(n) << h\*(n), 因此生成路径的消耗相差不大。

在机器人真实的落地运用中，要考虑普适性，上述中三种算法，Dijkstra’s在简单场景下效率不高，在这次对比中采用的最优解的方法只适用于二维栅格地图，对于其他的地图需要重新计算理论最优解，也比较耗时耗力，因此我觉得A\*普适性更高。

### 程序补充说明

在matlab作业中，我自定义了一个启发式方程脚本heuristic.m, 这个函数可以根据调用时输入的option变量，计算并输出相对应的启发式方程结果。

0->Dijkstra’s; 1-> normal A\*; 2->optimal A\*

图形用户界面, 文本

描述已自动生成

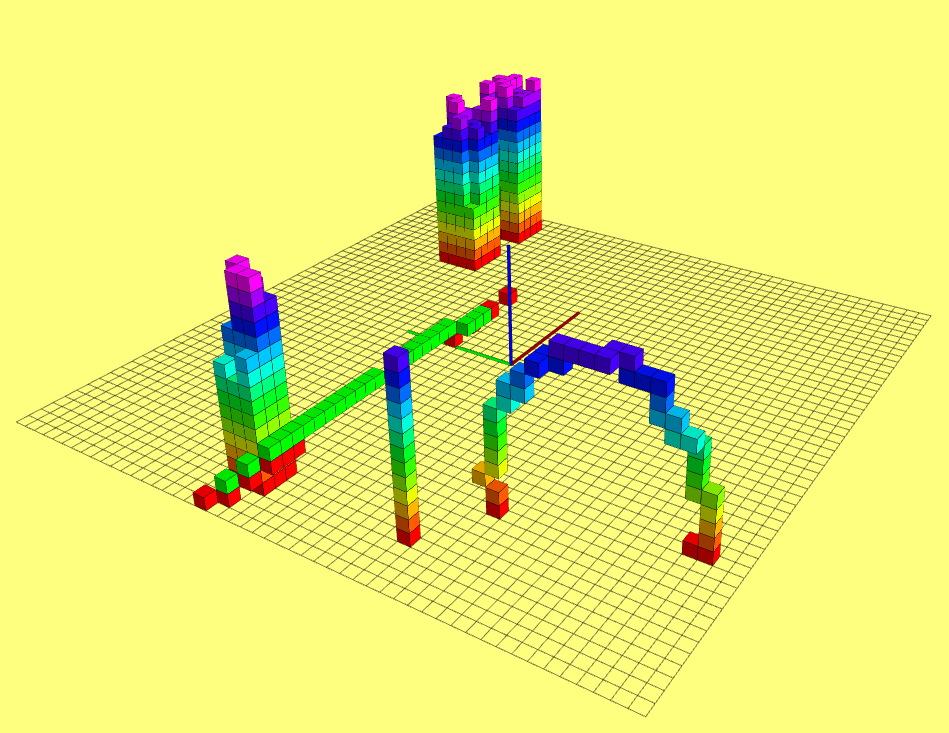
## Optimal A\* VS JPS

算法流程

日程表

中度可信度描述已自动生成

实现结果



### 采用的Heuristic Function

三维栅格地图下，最短路径问题的最优解：Diagnal heuristic 3d + simple tie breaker

 double hn = (sqrt(3)-sqrt(2))\*dmin + (sqrt(2)-1)\*dmid + dmax;

  //simple tie breaker

hn = hn\*(1+1/1000);

## 两种路径搜索方法的对比

### 数据背景：

两个场景

Sparse：circle\_num: 1; obs\_num: 5

Dense：circle\_num: 50; obs\_num: 1000

数据：对于四个目标点（尽量在顶点附近（-5,-5,0), (5,5,0),(-5, 5, 5), (5,-5,5)），生成路径所遍历的节点的个数，及所用时间的平均值。

两组对比：

对比1：A\* VS JPS without tie breaker

表格

描述已自动生成

对比2：A\* VS JPS with tie breaker

表格

描述已自动生成

### 数据分析

A\*和JPS区别：

**A\*在稀疏环境下，表现要好于JPS**，这也符合理论，因为JPS花费大量时间在搜索边界上了，而不是专注向目标前进。

相反，**在复杂场景下，JPS表现更优**，而且在实际中遇到过40ms VS 4ms的情况，当时的目标点与起始点之间有很多障碍物，可以表明越是在复杂的场景下，JPS的效果越好。理论上解释，就是得益于它跳跃的特点从而节省了遍历中间无用节点的时间，而且复杂的场景的边界一般很容易触及，这也弥补了JPS相对于A\*的不足。

Tie Breaker的作用

就结果来说，tie breaker在复杂的场景下发挥着比较好的作用，但在环境简单情况下并没有那么好。这有可能是数据的原因：对比的时候地图有差异(因为需要重新编译运行)，或者是tie breaker设置的过于简单。

### 遇到的问题和解决方法

**Multimap的遍历** 🡪 运用同类型的iterator遍历map，并于first, second的方式指向元素。

**Index和coord**两种坐标表示方法之间的区别，计算edgeCost和heuristic cost时使用的坐标不一致导致的各种问题。🡪 在loop中统一使用index，显示在地图上时用coord。

**三维中对角启发函数的计算** 🡪 先假设起始点沿3d对角线方向移动直到和终点在同一平面上，将三维问题简化为二维问题。