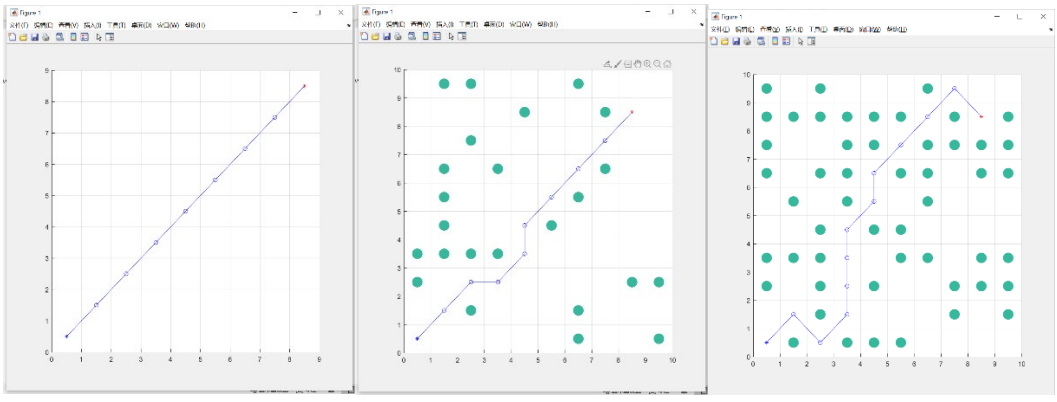
目录

Optimal A*在三种不同场景下生成的路径截图



采用的 Heuristic Function

Diagonal heuristic

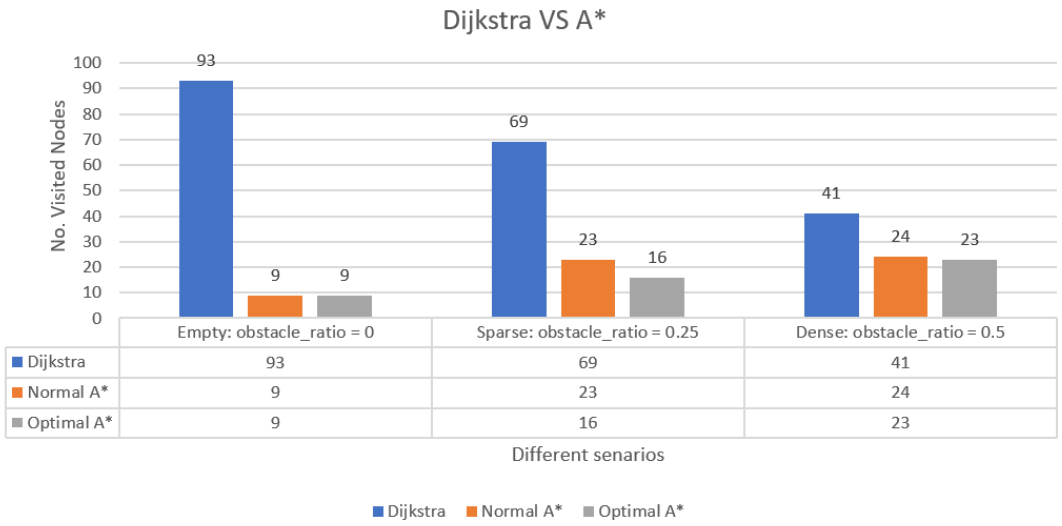
It has the closed-form solution!

dx=abs(node.x −goal.x)
dy=abs(node.y −goal.y)
h=(dx+dy)+(√2−2)*min(dx,dy)

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。



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。

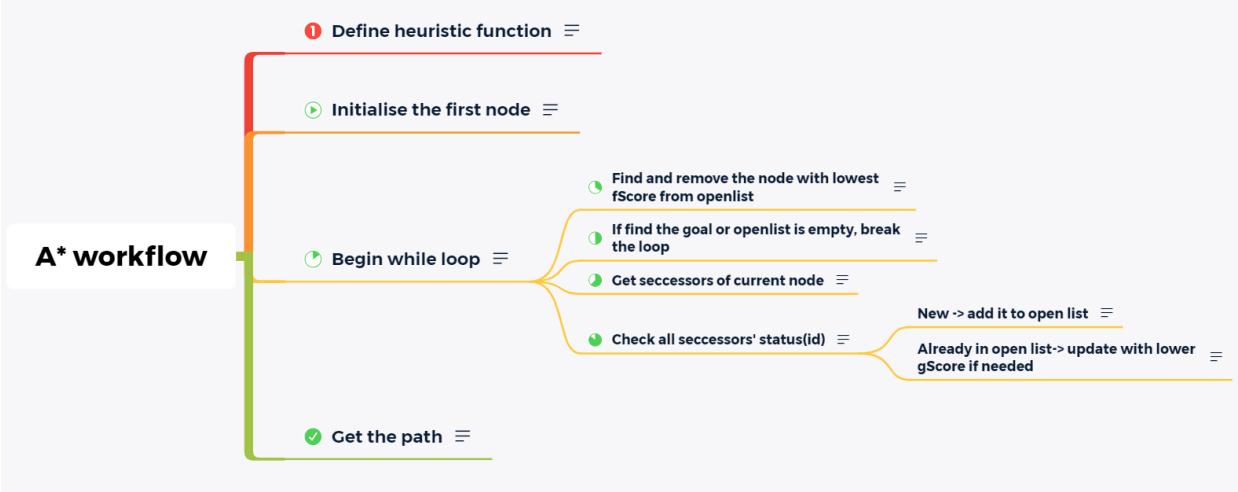
程序补充说明

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*

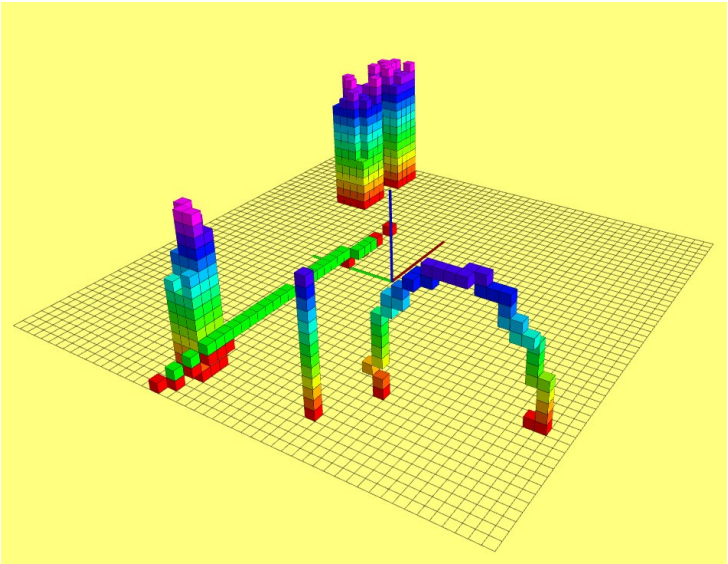
```
function hn = heuristic(x1, y1, x2, y2, option)
%This function calculates heuristic cost according to the option
%
if(option==0)  %Dijstla
    hn = 0;
elseif(option==1)  % A* using Eucliean
    hn=sqrt((x1-x2)^2 + (y1-y2)^2);
elseif(option==2)   % A* using optimal solution
    dx = abs(x2-x1);
    dy = abs(y2-y1);
    hn=dx+dy+(sqrt(2)-2)*min(dx, dy);
end
```

Optimal A* VS JPS

算法流程



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

without tie breaker	Sparse		Dense	
	Visited nodes	Time(ms)	Visited noc	Time(ms)
A*	100	0.96	286	3.41
	96	1.05	101	1.73
	107	1.06	184	2.45
	119	1.28	258	2.34
Ave	105.50	**1.09**	207.25	**2.48**
JPS	16	1.23	163	2.14
	19	1.39	62	0.64
	18	3.31	98	1.07
	46	3.29	58	0.37
Ave	24.75	**2.31**	95.25	**1.06**

对比 2：A* VS JPS with tie breaker

without tie breaker	Sparse		Dense	
	Visited nodes	Time(ms)	Visited noc	Time(ms)
A*	100	0.85	363	3.97
	96	0.96	140	1.72
	189	1.95	145	1.65
	189	1.87	80	1.27
Ave	143.50	**1.41**	182.00	**2.15**
JPS	31	3.01	197	0.5
	3	2.64	83	0.26
	26	3.94	50	0.78
	27	3.94	49	0.8
Ave	21.75	**3.38**	94.75	**0.59**

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。