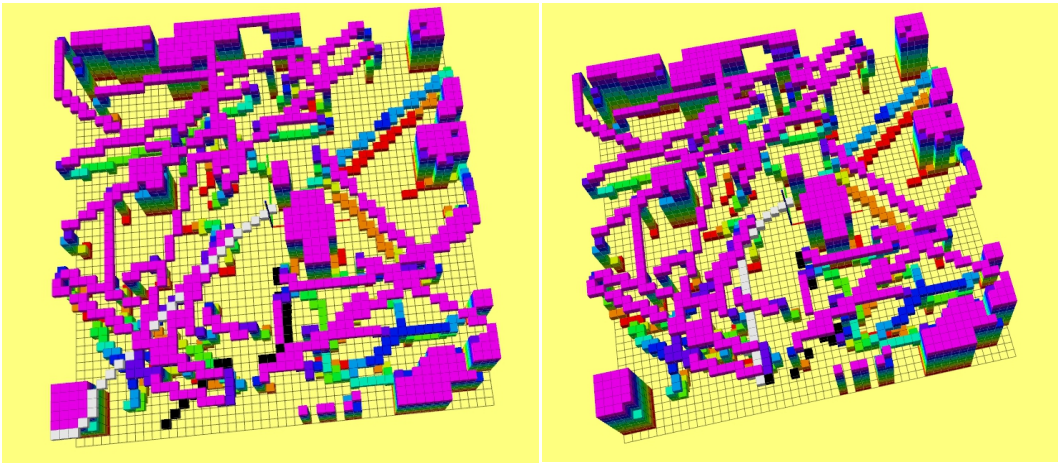
HomeWork 2

一、成果展示



白色估计为 A*，黑色轨迹为 JPS。

```
NODES
  /
    demo_node (grid_path_searcher/demo_node)
    random_complex (grid_path_searcher/random_complex)
    waypoint_generator (waypoint_generator/waypoint_generator)

ROS_MASTER_URI=http://192.168.3.27:11311

process[demo_node-1]: started with pid [11961]
process[random_complex-2]: started with pid [11962]
process[waypoint_generator-3]: started with pid [11967]
[ INFO] [1657088934.215309350]: [node] receive the planning target
[ INFO] [1657088934.217262909]: start AstarGraphSearch
[ WARN] [1657088934.222200426]: [A*]{sucess}  Time in A*  is 4.861115 ms, path cost if 9.699485 m
[ INFO] [1657088934.222270811]: AstarGraphSearch over
[ INFO] [1657088934.222304392]: beigin astar getPath~~~~
[ INFO] [1657088934.222342904]: astar getPath over~~~~
[ WARN] [1657088934.222985095]: visited_nodes size : 3474
[ INFO] [1657088934.223131156]: astar visGridPath over~~
[ INFO] [1657088934.223808205]: start JPSGraphSearch~
[ WARN] [1657088934.224216867]: [JPS]{sucess} Time in JPS is 0.343935 ms, path cost if 11.096910 m
[ INFO] [1657088934.224249515]: JPSGraphSearch over~
[ INFO] [1657088934.224278823]: beigin JPS getPath~~~~
[ INFO] [1657088934.224311890]: JPS getPath over~~~~
[ WARN] [1657088934.224862854]: visited_nodes size : 88
[ INFO] [1657088934.224924430]: JPS visGridPath over~~
```

二、部分代码分析

1. getHeu()获取 h 值

```
#define Manhattan 1;
#define Euclidean 2;
#define Diagonal  3;
#define Dijkstra  0;
double AstarPathFinder::getHeu(GridNodePtr node1, GridNodePtr node2)
{
    /*
    choose possible heuristic function you want
    Manhattan, Euclidean, Diagonal, or 0 (Dijkstra)
    Remember tie_breaker learned in lecture, add it here ?
    */
   double d_x = abs(double(node2->index(0)-node1->index(0)));
   double d_y = abs(double(node2->index(1)-node1->index(1)));
   double d_z = abs(double(node2->index(2)-node1->index(2)));

//可选函数模型
   int h_function=Euclidean;// Manhattan, Euclidean, Diagonal, Dijkstra
   double h=0.0;
   switch(h_function)
   {
   case 1 :  //Manhattan
     {h=abs(d_x+d_y+d_z);}
     break;

   case 2 ://Euclidean
      {h=sqrt(pow(d_x ,2.0)+pow(d_y,2.0)+ pow(d_z,2.0)); }
     // double h=(node1->index-node2->index).norm();
      break;
   case 3 ://Diagonal   ? ? ? ? ?
      {

      }
      break;

   case 0 ://Dijkstra
       break;

   }
   bool use_Tie_Breaker=true;
   if(use_Tie_Breaker)
   {
     float p=1.0/1000;
     h=h*(1+p);
   }

    return h;
}
```

根据不同的函数值，指定不同的策略。当 h 等于 0 时，A*算法就会退化为 Dijkstra 算法。欧式距离为斜边，曼哈顿距离为直角边之和。Tie_Breaker 的使用是针对 f 值相同，路径不同的情况。轻微改变 h，以改变 f 值，以打破对称性。

2. AstargetSucc()获取拓展节点

```
// 3D 理论是扩展周围27-1个节点
for (int x=currentPtr->index.x()-1;x<=currentPtr->index.x()+1;x++)
{
    for (int y =currentPtr->index.y()-1 ; y <=currentPtr->index.y()+1; y++)
    {
        for (int z =currentPtr->index.z()-1; z <=currentPtr->index.z()+1; z++)
        {
            //排除自身
            if(x!=currentPtr->index.x() || y!=currentPtr->index.y() ||z!=currentPtr->index.z())
            {
                if(isFree(x,y,z))
                {
                    auto roundNodeptr=GridNodeMap[x][y][z];
                    neighborPtrSets.push_back(roundNodeptr);

            //  注意此处的edgeCostSets的存储内容及作用
                    if(abs(x-currentPtr->coord[0])+abs(y-currentPtr->coord[1])+abs(z-currentPtr->coord[2])==1.0)
                    {
                     edgeCostSets.push_back(1.0);
                    }else if(abs(x-currentPtr->coord[0])+abs(y-currentPtr->coord[1])+abs(z-currentPtr->coord[2])==2.0)
                    {
                     edgeCostSets.push_back(double(sqrt(2)));
                    }else
                    {
                     edgeCostSets.push_back(double(sqrt(3)));
                    }
                }

            }

        }

    }
}
```

将获取的拓展节点存入 neighborPtrSets，把父节点与拓展节点之间的距离存入 edgeCostSets。

三、对比分析

将地图生成器的随机种子去掉，控制每次的地图和起始点设为相同，进行各算法的比较，结果如下：

	Different Algorithm				
	Euclidean	Manhattan	Dijkstra	Diagonal	Diagonal with Tie Breaker
time(ms)	4.385	0.187	19.285		
Path cost(m)	8.660	8.660	8.660		
Visited nodes	2021	26	22591		

不同启发函数对 A*的搜索效率对比：

1）搜索速度： Manhattan>Euclidean>Dijkstra

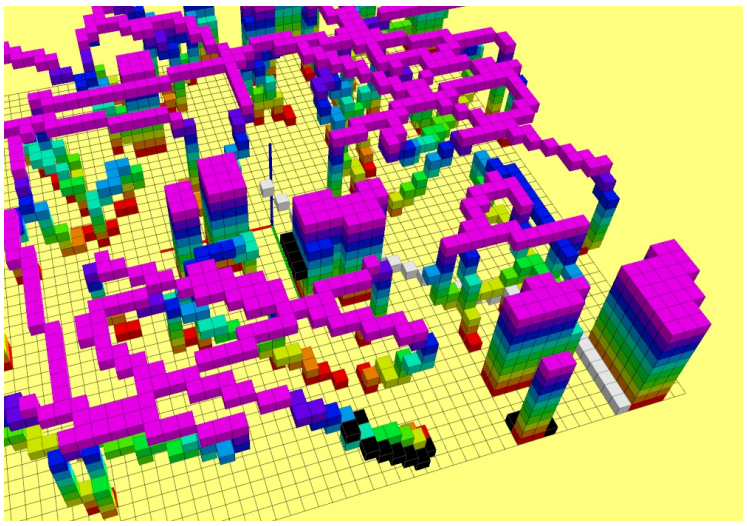
2）遍历节点数：Dijkstra>Euclidean>Manhattan

3）路径长度：长度基本相同

加入 Tie Breaker 的作用：

针对 f 值相同，路径不同的情况。轻微改变 h，以改变 f 值，打破对称性。使用 Tie Breaker 可减少了搜索过程中遍历的节点数目，降低运行的时间，提高效率。

A*与 JPS 对比：利用多组数据进行对比：



A*			
	time(ms)	Path cost(m)	Visited nodes
test1	2.966	8.660	1985
test2	3.275	8.660	2014
test3	4.834	9.006	3666
JPS			
test1	0.254	11.999	105
test2	0.151	9.717	16
test3	0.114	9.653	17

JPS 在实验障碍较多的地图中，运行时间和遍历节点数量都优于 A*的。