A*作业说明

Part 0 准备工作

下载 hw_2，将 src 文件夹中的三个功能包**覆盖第一次作业的三个功能包**（改动较大，建议直接覆盖），并按照第一次作业的流程进行编译。

注意，代码编译没有问题，但是运行后会报错，这和函数未完成有关，待同学们补全代码就不会出现该问题。

Part 1 代码执行流程

见文件：src/grid_path_searcheer/src/demo_node.cpp

主函数 main

```
1.  int main(int argc, char** argv)
2.  {
3.      ......
4.      //订阅到地图信息的回调函数
5.      _map_sub  = nh.subscribe( "map",       1, rcvPointCloudCallBack );
6.      //订阅到终点信息的回调函数
7.      _pts_sub  = nh.subscribe( "waypoints", 1, rcvWaypointsCallback );
8.      ......
9.
10.     //定义了结构体 AstarPathFinder 变量_astar_path_finder，该结构体存储、实现了 Astar 路径规划所需的所有信息和功能
11.     _astar_path_finder  = new AstarPathFinder();
12.     _astar_path_finder  -> initGridMap(_resolution, _map_lower, _map_upper, _max_x_id, _max_y_id, _max_z_id);
13.
14.     //定义了结构体 JPSPathFinder 变量_jps_path_finder，该结构体存储、实现了 JPS 路径规划所需的所有信息和功能
15.     _jps_path_finder    = new JPSPathFinder();
16.     _jps_path_finder    -> initGridMap(_resolution, _map_lower, _map_upper, _max_x_id, _max_y_id, _max_z_id);
17.
18.     ......
19. }
```

回调函数 rcvPointCloudCallBack

```
1.  void rcvPointCloudCallBack(const sensor_msgs::PointCloud2 & pointcloud_map)
2.  {
```

```
    ......
    // 将障碍物信息设置进入栅格化地图，为后续路径规划做准备
    _astar_path_finder->setObs(pt.x, pt.y, pt.z);
    _jps_path_finder->setObs(pt.x, pt.y, pt.z);

    // 可视化地图部分
    ......
    _grid_map_vis_pub.publish(map_vis);
    ......
}
```

回调函数 rcvWaypointsCallback

```
void rcvWaypointsCallback(const nav_msgs::Path & wp)
{
    ......
    //获取交互式界面给出的终点坐标
    target_pt << wp.poses[0].pose.position.x,
                 wp.poses[0].pose.position.y,
                 wp.poses[0].pose.position.z;
    ......
    //输入起点、终点，调用 pathFind 函数
    pathFinding(_start_pt, target_pt);
}
```

路径规划函数 pathFinding

```
void pathFinding(const Vector3d start_pt, const Vector3d target_pt)
{
    //使用 A*进行路径搜索
    _astar_path_finder->AstarGraphSearch(start_pt, target_pt);
    //获取规划的路径
    auto grid_path    = _astar_path_finder->getPath();
    ......
    //可视化结果
    visGridPath (grid_path, false);
    ....
    //为下次规划重置地图
    _astar_path_finder->resetUsedGrids();
    //进行 JPS 路径规划编写时，将_use_jps 的值置为 1 即可
#define _use_jps 0
#if _use_jps
    {
      //使用 JPS 进行路径搜索
      _jps_path_finder -> JPSGraphSearch(start_pt, target_pt);
      ......
    }
#endif
```

```
}
```

Part2 涉及类和结构体的简介

节点表示：用结构体变量 GridNode 表示，存储了节点的坐标、g(n)、f(n)值、父节点指针等信息。

```
struct GridNode
{
    int id;      // 1--> open set, -1 --> closed set
    Eigen::Vector3d coord;
    Eigen::Vector3i dir;   // direction of expanding, only for JPS
    Eigen::Vector3i index;
    double gScore, fScore;
    GridNodePtr cameFrom;
    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(){};
};
```

父类AstarPathFinder

```
class AstarPathFinder
{
    private:

    protected:
        ......
            //open set 实现：用C++ STL 中的 multimap 实现
        std::multimap<double, GridNodePtr> openSet;
            //启发式函数，作业完成
        double getHeu(GridNodePtr node1, GridNodePtr node2);
            //拓展节点函数，作业完成
        void AstarGetSucc(GridNodePtr currentPtr, std::vector<GridNodePtr> & neighborPtrSets, std::
    vector<double> & edgeCostSets);
```

```
        ......
    public:
      ......
          //A*搜索算法函数，作业完成
      void AstarGraphSearch(Eigen::Vector3d start_pt, Eigen::Vector3d end_pt);
      ......
};
```

open set实现：用C++ STL中的multimap实现，multimap将{key,value}当做元素，允许重复元素。multimap根据key的排序准则自动将元素排序，因此使用时只需考虑插入和删除操作即可。

详细信息可以查看以下文档：ttps://zh.cppreference.com/w/cpp/container/multimap

继承类JPSPathFinder

```
class JPSPathFinder: public AstarPathFinder
{
    ......
    //JPS 的拓展节点函数，已经完成
    void JPSGetSucc(GridNodePtr currentPtr, std::vector<GridNodePtr> & neighborPtrSets, std::vector<double> & edgeCostSets);
    ......
    //JPS 搜索算法函数，主体框架和 A*一致，只要用心对照修改，在完成了 A*的基础，使用我们提供的函数接口完成 JPS 难度不大
    void JPSGraphSearch(Eigen::Vector3d start_pt, Eigen::Vector3d end_pt);
};
```

Part 3 任务详情

完成 **src/grid_path_searcheer/src/Astar_searcher.cpp** 下的

```
void AstarPathFinder::AstarGetSucc(...);
double AstarPathFinder::getHeu(...);
void AstarPathFinder::AstarGraphSearch(...);
vector<Vector3d> AstarPathFinder::getPath(...);
```

请仔细阅读代码中的注释，按照STEP 1 – STEP 8 的提示逐步完成。

Part 4 作业提交要求

(1)提交完整可编译运行的程序功能包**grid_path_searcheer**

(2)撰写一篇**不超过2页A4纸**的文档，需要包含以下内容

(3)算法流程、运行结果

(4)对比不同启发式函数（Manhattan、Euclidean、Diagonal Heuristic）对A*运行效率的影响

③对比是否加入Tie Breaker对A*运行效率的影响

④任何完成算法过程中遇到的问题、以及解决方法

⑤（选做）如果完成了JPS，最好附上A*和JPS算法效率的分析（何种情况下A*更优、何种情况下JPS更优？）

⑥（选做）以及其他你认为有趣的内容。

Part 5 拓展练习（选做）

在完成任务（1）的基础上，仿照**void** AstarPathFinder::AstarGraphSearch(...)的写法，补全

src/grid_path_searcheer/src/readonly/JPS_searcher.cpp下的

void JPSPathFinder::JPSGraphSearch(...)，由于JPS和Astar仅在扩展节点时有区别，所以只需要仔细对照，并结合已经写好的**void** JPSPathFinder::JPSGetSucc (...)；完成JPS难度不大。