

# 机器人路径规划

大作业

张 文泰

学号：21009101463

## 一、程序分析

### A \* 算法

```
double AstarPathFinder::getHeu(GridNodePtr node1, GridNodePtr node2) {
    // using digonal distance and one type of tie_breaker.
    double h;
    Eigen::Vector3i start_index = node1->index;
    Eigen::Vector3i end_index = node2->index;

    double distance[3];
    distance[0] = abs((double)(start_index(0) - end_index(0)));
    distance[1] = abs((double)(start_index(1) - end_index(1)));
    distance[2] = abs((double)(start_index(2) - end_index(2)));
    std::sort(distance, distance + 3);
    h = distance[0] + distance[1] + distance[2] + (std::sqrt(3.0) - 3) * distance[0] +
        (std::sqrt(2.0) - 2) * distance[1];
    return h;
}
```

### getHeu 函数

该函数在 A \* 算法中用于计算启发函数——用于估计从当前节点到目标节点的最短路径代价,它接收两个指向 *GridNode* 类型对象的指针参数 *node1* 和 *node2*,下面是该函数的大致思路:

1. 获取当前节点 *node1* 和目标节点 *node2* 的索引,这些索引表示节点在三维空间中的位置。
2. 计算节点 *node1* 和 *node2* 在三个维度上的距离。这里使用了绝对差值来计算距离。
3. 对计算得到的距离进行排序,以确保最大的距离值在数组的最后。这里采用的排序方式是标准库中的 *std::sort* 函数。
4. 将三个维度上的距离合并为一个启发式估计值 *h*。实现方式是将三个维度上的距离相加,并且加入了一个 *tie\_breaker*。*tie\_breaker* 的作用是在启发式估计值相等的情况下,优先选择距离更远但是在对角线上的节点。
5. 返回计算得到的启发式估计值 *h*。

```

void AstarPathFinder::AstarGraphSearch(Vector3d start_pt, Vector3d end_pt) {
    ros::Time time_1 = ros::Time::now();
    Vector3i start_idx = coord2gridIndex(start_pt);
    Vector3i end_idx = coord2gridIndex(end_pt);
    goalIdx = end_idx;
    start_pt = gridIndex2coord(start_idx);
    end_pt = gridIndex2coord(end_idx);
    GridNodePtr startPtr = GridNodeMap[start_idx(0)][start_idx(1)][start_idx(2)];
    GridNodePtr endPtr = GridNodeMap[end_idx(0)][end_idx(1)][end_idx(2)];
    openSet.clear();
    GridNodePtr currentPtr = NULL;
    GridNodePtr neighborPtr = NULL;
    startPtr->gScore = 0;
    startPtr->fScore = getHeu(startPtr, endPtr);
    startPtr->id = 1;
    startPtr->coord = start_pt;
    openSet.insert(make_pair(startPtr->fScore, startPtr));
    double tentative_gScore;
    vector<GridNodePtr> neighborPtrSets;
    vector<double> edgeCostSets;
    while (!openSet.empty()) {
        currentPtr = openSet.begin()->second;
        currentPtr->id = -1;
        openSet.erase(openSet.begin());
        Eigen::Vector3i current_idx = currentPtr->index;
        AstarGetSucc(currentPtr, neighborPtrSets, edgeCostSets);
        for (int i = 0; i < (int)neighborPtrSets.size(); i++) {
            neighborPtr = neighborPtrSets[i];
            double gh = currentPtr->gScore + edgeCostSets[i];
            double fh = gh + getHeu(neighborPtr, endPtr);
            if (neighborPtr->id == 0) {
                neighborPtr->gScore = gh;
                neighborPtr->fScore = fh;
                neighborPtr->cameFrom = currentPtr;
                neighborPtr->nodeMapIt = openSet.insert(make_pair(neighborPtr->fScore, neighborPtr));
                if (neighborPtr->index == goalIdx) {
                    ros::Time time_2 = ros::Time::now();
                    terminatePtr = neighborPtr;
                    ROS_WARN("[A*]{sucess} Time in A* is %f ms, path cost if %f m", (time_2 -
time_1).toSec() * 1000.0, currentPtr->gScore * resolution);
                    return;
                }
            }
            else {
                neighborPtr->id = 1;
                continue;
            }
        }
        else if (neighborPtr->id == 1) {
            if (neighborPtr->gScore > gh){
                neighborPtr->gScore = gh;
                neighborPtr->fScore = fh;
                neighborPtr->cameFrom = currentPtr;
                openSet.erase(neighborPtr->nodeMapIt);
                neighborPtr->nodeMapIt = openSet.insert(make_pair(neighborPtr->fScore,
neighborPtr));
            }
        }
        else {
            continue;
        }
    }
    }
    ros::Time time_2 = ros::Time::now();
    if ((time_2 - time_1).toSec() > 0.1)
        ROS_WARN("Time consume in Astar path finding is %f",
            (time_2 - time_1).toSec());
}

```

## AstarGraphSearch 函数

这段代码实现了 A\*算法的核心逻辑，用于在给定起点和终点的情况下，在地图上搜索最短路径。代码大致逻辑如下：

1. 记录起点和终点对应的栅格坐标，并将目标索引设置为终点的索引。然后，将起点和终点的位置重新计算（似乎是为了确保它们在栅格中的位置是准确的）。
2. 初始化起点和终点节点，这里假设它们已经在之前的地图初始化中被创建。
3. 清空待弹出点集，并定义要弹出的当前节点和邻居节点。
4. 计算起点的启发式函数值，并将其加入开集。
  - a) 进入主循环，直到开集为空：
  - b) 弹出开集中最大  $f$  值的节点，并标记为闭集。
  - c) 获取当前节点的邻居节点集合。
  - d) 遍历邻居节点集合：
    - i. 如果邻居节点是自由节点：
    - ii. 计算邻居节点的  $g$  和  $f$  值，并加入开集。
    - iii. 如果该邻居节点是目标节点，则搜索成功，记录终点并返回。
    - iv. 否则，将邻居节点标记为开集。
  - e) 如果邻居节点已在开集中：
    - i. 如果通过当前节点到达邻居节点的路径更短，则更新邻居节点的  $g$  和  $f$  值，并重新排序开集。
    - ii. 如果邻居节点已在闭集中，则不做处理。
5. 如果搜索失败，记录时间消耗。

简单地说，这段代码主要实现的是 A\*算法的搜索过程，其中通过开集和闭

集来管理节点的状态，并在搜索过程中不断更新节点的代价值以寻找最短路径。

## RDP 算法简化轨迹

```
vector<Vector3d> AstarPathFinder::pathSimplify(const vector<Vector3d> &path, double path_resolution)
{
    vector<Vector3d> subPath;
    if (path.size() <= 2)
    {
        // 如果轨迹中的点数小于2，则直接返回原来的轨迹
        subPath = path;
    }
    else
    {
        const Vector3d first = path[0]; //定义首点
        const Vector3d last = path[path.size() - 1]; //定义尾点
        int flag = 0; //标记距离最大的点的下标
        double disSquare = 0;
        for (int i = 1; i < path.size() - 1; i++) {
            double temp = disP2L(first, last, path[i]);
            if (temp > disSquare) { //记录最大距离及编号
                disSquare = temp;
                flag = i;
            }
        }

        if (disSquare < path_resolution) { //判断值与阈值的关系,阈值自己设定
            subPath.push_back(first); //如果小于阈值，则保留首尾点
            subPath.push_back(last);
            //用于存储留下来的点,是最后的成果
        }
        else { //否则分成两段
            vector<Vector3d> recResults1 = pathSimplify(vector<Vector3d>(path.begin(), path.begin() +
            flag), path_resolution);
            vector<Vector3d> recResults2 = pathSimplify(vector<Vector3d>(path.begin() + flag,
            path.end()), path_resolution);
            subPath.insert(subPath.end(), recResults1.begin(), recResults1.end() - 1);
            subPath.insert(subPath.end(), recResults2.begin(), recResults2.end());
        }
    }
    return subPath;
}
```

### pathSimplify 函数

这段代码实现了 RDP 算法，用于简化路径。其主要逻辑是通过找到距离最大点来决定路径的主要形状，并不断递归分割和简化路径，以达到减少路径点数的目的。其主要逻辑如下：

1. 检查：如果路径中的点数少于或等于 2，则路径不需要简化，直接返回原路径。
2. 提取首尾点：取出路径第一个点和最后一个点，作为简化路径的首尾点。
3. 寻找最大偏离点：遍历路径中的所有中间点，计算每个点到首尾点连线的距离，找出距离最大的点并记录其下标和距离。
4. 判断是否需要进一步简化：如果最大距离小于给定的阈值，认为路径在此范围内可以简化为一条直线，只保留首尾点作为简化路径。如果最大

距离大于阈值，说明路径不能简化为一条直线，需要进一步处理。

5. 递归简化路径：将路径在最大偏离点处分成两段，对每段路径分别递归调用简化函数，简化每段子路径，将两段简化后的路径合并为一条完整的简化路径。
6. 合并结果：合并时，避免重复添加交界点，确保简化路径中的点是唯一且按顺序排列的。
7. 返回简化路径：最终返回简化后的路径，包含保留下来的关键点，达到减少路径点数的目的。

RDP 算法通过递归分割路径，仅保留对整体形状影响最大的关键点，从而高效简化路径，同时保持路径的主要形状特征，具有高效、灵活、易实现和广泛适用的优点。

### Minisnap 轨迹优化

```
void trajOptimization(Eigen::MatrixXd path) {
    MatrixXd vel = MatrixXd::Zero(2, 3);
    MatrixXd acc = MatrixXd::Zero(2, 3);
    vel.row(0) = start_vel;
    _polyTime = timeAllocation(path);
    _polyCoeff =
        _trajGene->PolyQPGeneration(_dev_order, path, vel, acc, _polyTime, slover);
    int unsafe_segment;
    unsafe_segment = _astar_path_finder->safeCheck(_polyCoeff, _polyTime);
    unsafe_segment = -1;
    MatrixXd repath = path;
    int count = 0;
    while (unsafe_segment != -1) {
        std::cout << unsafe_segment << std::endl;
        Eigen::Vector3d start = repath.row(unsafe_segment);
        Eigen::Vector3d end = repath.row(unsafe_segment + 1);
        Eigen::Vector3d mid = (start + end) / 2;
        Eigen::MatrixXd tmp(int(repath.rows() + 1), 3);
        std::cout << mid << std::endl;
        tmp.block(0, 0, unsafe_segment + 1, 3) = repath.block(0, 0, unsafe_segment + 1, 3);
        tmp.row(unsafe_segment + 1) = mid;
        tmp.block(unsafe_segment + 2, 0, int(repath.rows()) - unsafe_segment - 1, 3) =
            repath.block(unsafe_segment + 1, 0, int(repath.rows()) - unsafe_segment - 1, 3);
        repath = tmp;
        _polyTime = timeAllocation(repath);
        _polyCoeff =
            _trajGene->PolyQPGeneration(_dev_order, repath, vel, acc, _polyTime, slover);
        unsafe_segment = _astar_path_finder->safeCheck(_polyCoeff, _polyTime);
    }
    visPath(repath);
    visTrajectory(_polyCoeff, _polyTime);
}
```

### trajOptimization 函数

这段代码实现了一个轨迹优化函数，用于确保生成的轨迹在空间中是安全的。

该函数接受一个路径矩阵 path，通过多项式拟合生成轨迹，并检查其安全性。如果发现不安全段，则通过细分路径并重新生成轨迹，直到所有段都安全。

```
VectorXd timeAllocation(MatrixXd Path) {
    VectorXd time(Path.rows() - 1);
    double t_scope = 2.0 * _Vel / _Acc;
    double distance_acc = 1.0 / 2.0 * _Acc * t_scope * t_scope * 2.0;
    for (int k = 0; k < Path.rows() - 1; ++k) {
        Vector3d delta = Path.row(k) - Path.row(k + 1);
        double d = std::sqrt(delta.dot(delta));

        if (d <= distance_acc) {
            time(k) = std::sqrt(d / _Acc);
        }
        else {
            time(k) = t_scope + (d - distance_acc) / _Vel;
        }
    }
    return time;
}
```

timeAllocation 函数

该函数用于为路径中的各段分配时间，它根据路径段长度、最大速度和最大加速度，为路径的每一段分配合适的时间。其主要优点在于能合理考虑加速、匀速和减速三个阶段，使得时间分配更为精准和高效。

```
int AstarPathFinder::safeCheck(MatrixXd polyCoeff, VectorXd time) {
    int unsafe_segment = -1;
    Vector3d pos;
    for (int i = 0; i < time.size(); i++) {
        for (double t = 0.0; t < time(i); t += 0.01) {
            pos = getPosPoly(polyCoeff, i, t);
            if (isOccupied(coord2gridIndex(pos))) {
                return i;
            }
        }
    }
    return unsafe_segment;
}
```

safeCheck 函数

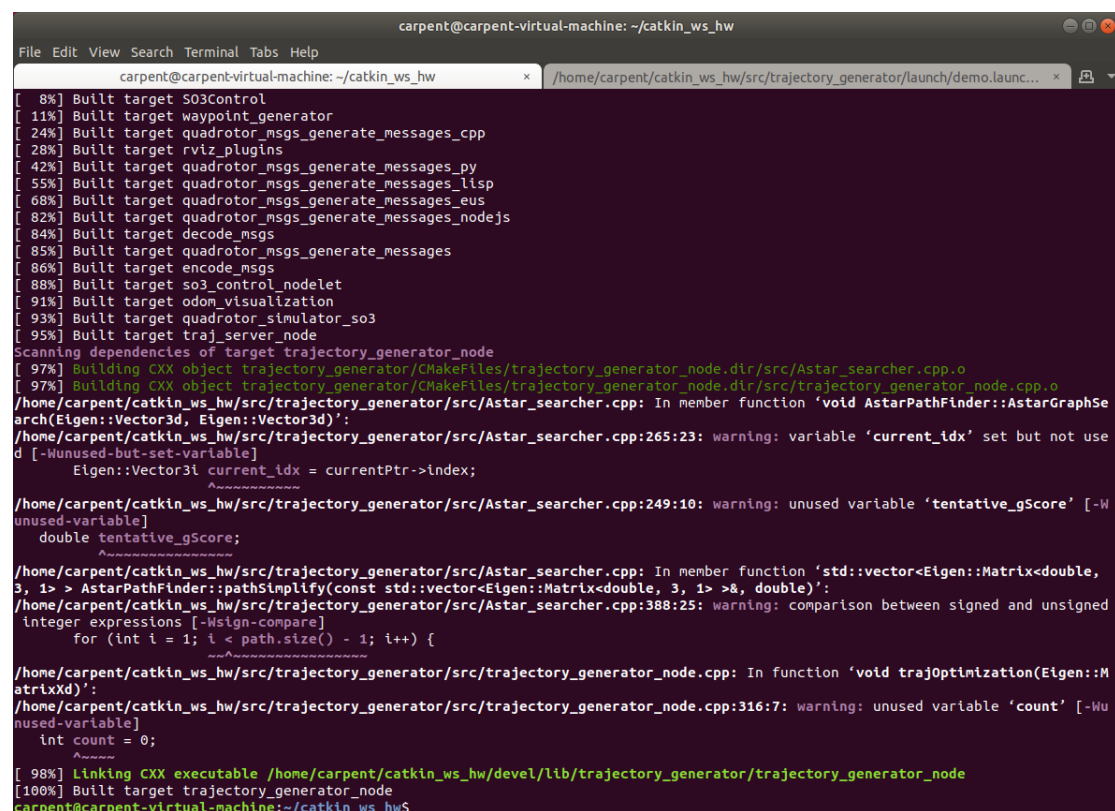
safeCheck 函数通过遍历轨迹的每一段和每段内的多个时间点，检查轨迹是否安全。如果任何位置位于障碍物中，函数会返回该段的索引，否则返回 -1 表示安全。

示整个轨迹安全。这种方法保证了生成的轨迹能够避开障碍物，确保路径的安全性。函数关键点在于：

1. 时间步长：时间步长 0.01 是检查轨迹的精细程度。
2. 轨迹位置计算：getPosPoly 函数通过多项式系数计算出当前时间的轨迹位置。
3. 位置检查：isOccupied 函数结合 coord2gridIndex 转换，将轨迹位置映射到网格上，并检查该位置是否有障碍物。

Minisnap 轨迹优化通过最小化加速度的高次导数来实现路径的平滑化和优化，确保机器人的运动平稳且安全，其优点包括生成的路径平滑度高、加速度变化小、适应性强等。

## 二、实验结果

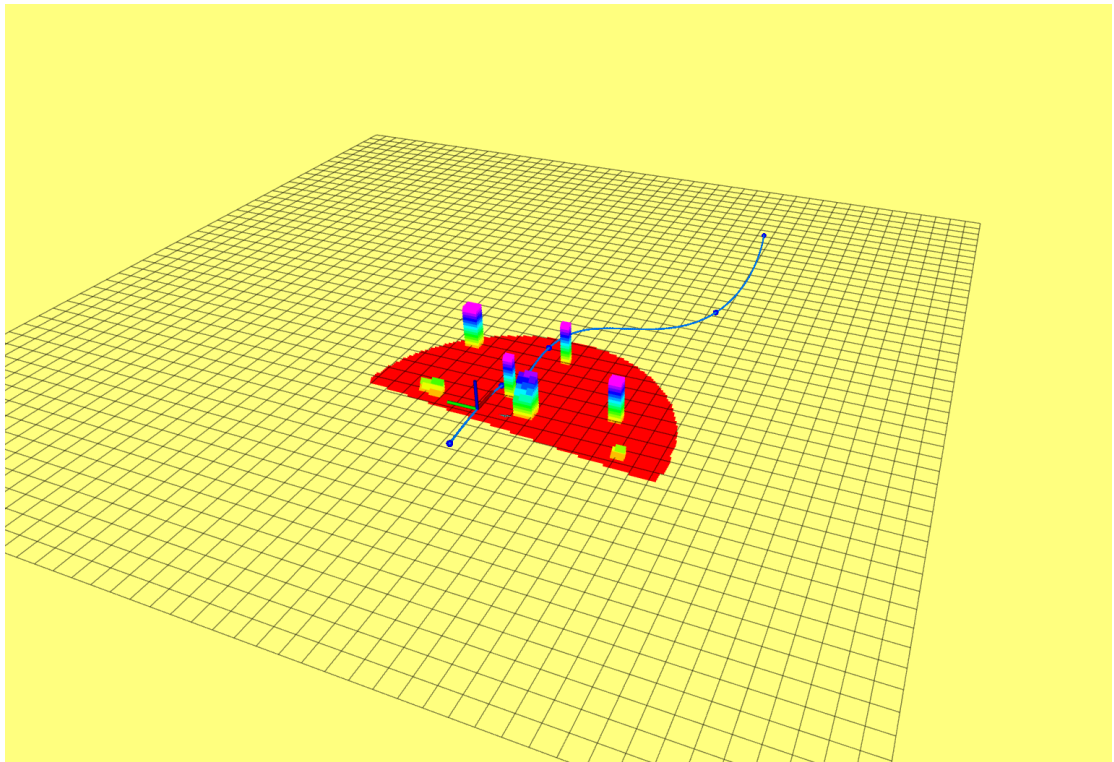


```
carpent@carpent-virtual-machine: ~/catkin_ws_hw
File Edit View Search Terminal Tabs Help
carpent@carpent-virtual-machine: ~/catkin_ws_hw x /home/carpet/catkin_ws_hw/src/trajectory_generator/launch/demo.launch... x
[ 8%] Built target S03Control
[ 11%] Built target waypoint_generator
[ 24%] Built target quadrotor_msgs_generate_messages_cpp
[ 28%] Built target rviz_plugins
[ 42%] Built target quadrotor_msgs_generate_messages_py
[ 55%] Built target quadrotor_msgs_generate_messages_lisp
[ 68%] Built target quadrotor_msgs_generate_messages_eus
[ 82%] Built target quadrotor_msgs_generate_messages_nodejs
[ 84%] Built target decode_msgs
[ 85%] Built target quadrotor_msgs_generate_messages
[ 86%] Built target encode_msgs
[ 88%] Built target so3_control_nodelet
[ 91%] Built target odom_visualization
[ 93%] Built target quadrotor_simulator_so3
[ 95%] Built target traj_server_node
Scanning dependencies of target trajectory_generator_node
[ 97%] Building CXX object trajectory_generator/CMakeFiles/trajectory_generator_node.dir/src/Astar_searcher.cpp.o
[ 97%] Building CXX object trajectory_generator/CMakeFiles/trajectory_generator_node.dir/src/trajectory_generator_node.cpp.o
/home/carpet/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp: In member function 'void AstarPathFinder::AstarGraphSearch(Eigen::Vector3d, Eigen::Vector3d)':
/home/carpet/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:265:23: warning: variable 'current_idx' set but not used [-Wunused-but-set-variable]
    Eigen::Vector3i current_idx = currentPtr->index;
                        ^
/home/carpet/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:249:10: warning: unused variable 'tentative_gScore' [-Wunused-variable]
    double tentative_gScore;
    ^
/home/carpet/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp: In member function 'std::vector<Eigen::Matrix<double, 3, 1> > AstarPathFinder::pathSimplify(const std::vector<Eigen::Matrix<double, 3, 1> >&, double)':
/home/carpet/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:388:25: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
    for (int i = 1; i < path.size() - 1; i++) {
                        ^
/home/carpet/catkin_ws_hw/src/trajectory_generator/src/trajectory_generator_node.cpp: In function 'void trajOptimization(Eigen::Matrix<double, 3, 1>*)':
/home/carpet/catkin_ws_hw/src/trajectory_generator/src/trajectory_generator_node.cpp:316:7: warning: unused variable 'count' [-Wunused-variable]
    int count = 0;
    ^
[ 98%] Linking CXX executable /home/carpet/catkin_ws_hw/devel/lib/trajectory_generator/trajectory_generator_node
[100%] Built target trajectory_generator_node
carpent@carpent-virtual-machine:~/catkin_ws_hw$
```

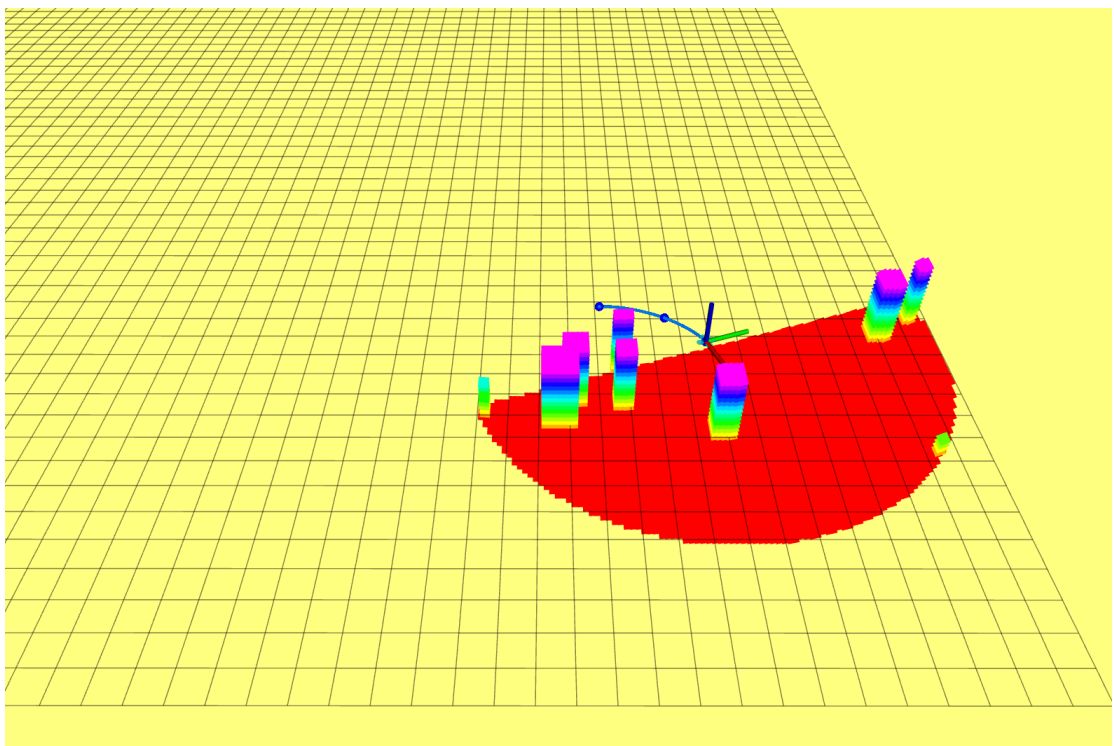
程序运行成功



如图所示，程序成功运行，地图显示在 RViz 中。接下来使用 3 nav goal 工具在图中选取终点，运行结果如图所示：

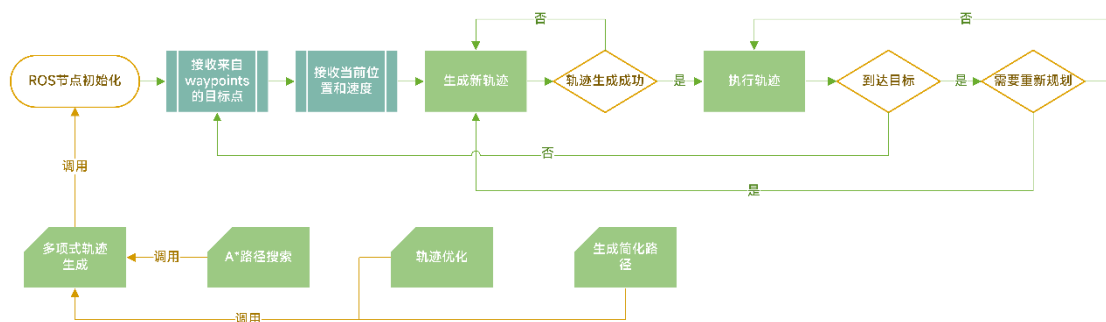


程序开始运行



轨迹运行结果

下面是本实验中的程序运行流程图：



流程图

### 三、 实验中遇到的问题及解决方案

```
carpent@carpent-virtual-machine: ~/catkin_ws_hw
File Edit View Search Terminal Help
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/trajectory_generator_node.cpp:316:7: warning: unused variable 'count' [-Wunused-variable]
    int count = 0;
    ^
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp: In member function 'void AstarPathFinder::AstarGraphSearch(Eigen::Vector3d, Eigen::Vector3d)':
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:259:42: error: qualified-id in declaration before '(' token
    vector<Vector3d> AstarPathFinder::getPath() {
                                         ^
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:275:31: error: qualified-id in declaration before '(' token
    double AstarPathFinder::disP2L(const Vector3d first, const Vector3d last, const Vector3d third)
                                ^
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:285:47: error: qualified-id in declaration before '(' token
    vector<Vector3d> AstarPathFinder::pathSimplify(const vector<Vector3d> &path,
                                                  ^
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:298:37: error: qualified-id in declaration before '(' token
    Vector3d AstarPathFinder::getPosPoly(MatrixXd polyCoeff, int k, double t) {
                                    ^
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:318:31: error: qualified-id in declaration before '(' token
    int AstarPathFinder::safeCheck(MatrixXd polyCoeff, VectorXd time) {
                                ^
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:215:15: warning: unused variable 'currentPtr' [-Wunused-variable]
    GridNodePtr currentPtr = NULL;
                  ^
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:216:15: warning: unused variable 'neighborPtr' [-Wunused-variable]
    GridNodePtr neighborPtr = NULL;
                  ^
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:239:10: warning: unused variable 'tentative_gScore' [-Wunused-variable]
    double tentative_gScore;
           ^
/home/carpent/catkin_ws_hw/src/trajectory_generator/src/Astar_searcher.cpp:326:1: error: expected '}' at end of input
}
^
trajectory_generator/CMakeFiles/trajectory_generator_node.dir/build.make:110: recipe for target 'trajectory_generator/CMakeFiles/trajectory_generator_node.dir/src/Astar_searcher.cpp.o' failed
```

报错现象

如图所示，我在第一次编译工作空间时出现了报错，阅读报错信息之后意识到，是由未补全代码导致，在完成代码补全之后解决了问题。

#### 四、 实验感悟与课程总结

在本学期的课程学习以及完成大作业的过程中,我收获了许多宝贵的经验和感悟,不仅学习了课内专业知识,还深刻理解了路径规划在机器人、无人机等领域的重要性。通过实践,我掌握了 A\*算法、Minisnap 轨迹优化等技术,加深了对这些技术原理和实现方法的理解。

不仅如此,几次课程作业的学习与锻炼,加深了我对 C++、MATLAB 编程语言的掌握程度,锻炼了自己的问题解决能力和编程技能,并可以使用它们实现路径规划算法,将其应用于实际场景中。除此之外,还包括对于虚拟机的使用以及 Ubuntu 系统的认识,我之前只用 VMware 来搭建饥荒联机版的云服务器,这次课程作业让我对虚拟机的认识更深一层,同时还让我认识了此前从未接触过的 Ubuntu 操作系统,极大地拓宽了我的视野。

最重要的是,通过这次大作业,我意识到路径规划不仅是一项技术,更是一种思维方式,需要综合考虑问题的复杂性和多样性,不断探索和创新。因此,我将继续学习和探索路径规划领域的知识,努力提升自己的能力,为未来的科技创新和应用做出更多贡献。

最后要说的是,非常感谢常晶老师与智斌亮老师在本学期的课程学习过程中对我的指导与帮助,我的成长得益于两位老师的悉心付出,对两位老师致以衷心的感谢。