第一部分：基础题部分

题目 1：坐标系转换

1.1 问题分析

需根据无人机（机体系 B）的世界系姿态，结合执行器相对机体系的运动（圆锥运动），计算机体系到执行器系的旋转矩阵，最终得到执行器在世界系下的姿态四元数并绘图。

1.2 核心思路

- **坐标系旋转关系**：世界系（W）→机体系（B）的旋转由无人机四元数描述，机体系（B）→执行器系（D）的旋转由给定矩阵 ${}^{B}R_{D}$ 描述，因此世界系（W）→执行器系（D）的旋转矩阵为：

$$ {}^{W}R_{D} = {}^{W}R_{B} \cdot {}^{B}R_{D} $$

（${}^{W}R_{B}$ 由无人机四元数转换得到）
- **四元数处理规则**：旋转矩阵转四元数后需归一化，且保证四元数实部 $q_w > 0$（若小于 0 则整体取反）。

1.3 计算步骤

- **读取无人机姿态数据**：从 tracking.csv 中读取时间 t 及无人机四元数（$q_x\ q_y\ q_z\ q_w$）。
- **计算 机体系到执行器系旋转矩阵 ${}^{B}R_{D}$**：

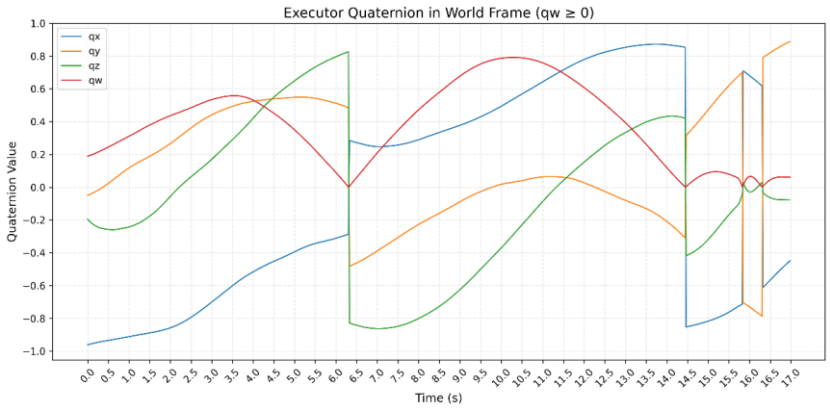
$$ {}^{B}R_{D}=\begin{bmatrix} cos(\omega t) & -sin(\omega t)cos\alpha & sin(\omega t)sin\alpha \\ sin(\omega t) & cos(\omega t)cos\alpha & -cos(\omega t)sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix} $$

代入$\omega = 0.5rad/s$、$\alpha = \pi/12$
- **计算世界系→执行器系旋转矩阵 ${}^{W}R_{D}$**：将无人机四元数转换为旋转矩阵 ${}^{W}R_{D}$，与 ${}^{B}R_{D}$ 做矩阵乘法。
- **旋转矩阵转四元数并处理**：将 ${}^{W}R_{D}$ 转换为四元数，归一化后确保$q_w > 0$。

1.4 代码实现

见附录 1：executor_quaternion.py

1.5 结果曲线



题目 2：开放题（规划/控制/动力学）

2.1 A* 轨迹规划的启发式与路径简化

1. tie_breaker 对开集拓展顺序与路径平滑性的影响

在 A *算法中，启发函数和 h(n)用于估计节点 n 到目标的代价，总代价 (f(n) = g(n) + h(n)) (g(n)为起点到节点 n 的实际代价)。**当两个节点的 f(n) 相等时（即出现“平局”）**，算法可能随机选择其中一个，导致路径出现“走之”形震荡（zig-zag）。

现有代码中提到的 tie_breaker（如 1.01）通过轻微放大启发式代价(h'(n) = tie_breaker * h(n))，使原本 f(n)相等的节点产生微小差异。具体影响如下：

- **开集拓展顺序**：放大后的 h'(n)会让更接近目标方向的节点优先被选中，减少了同等代价下的随机选择，使搜索更偏向“直接朝向目标”的路径。
- **路径平滑性**：减少了因平局导致的迂回震荡，路径更接近直线，平滑性提升。

2. 让无人机偏好平面飞行的修改方案

若想减少不必要的高度变化（即偏好 z 轴方向的平稳），可从**启发函数**或**边权**（实际代价）两方面修改：

方案 1：修改启发函数（使偏向于平面方向）

在计算启发式 h(n)时，降低高度方向（z 轴）的权重，使算法更倾向于在平面内规划路径。例如：

```
double Astarpath::getHeu(MappingNodePtr node1, MappingNodePtr node2) {
  double dx = abs(node1->index(0) - node2->index(0));
  double dy = abs(node1->index(1) - node2->index(1));
  double dz = abs(node1->index(2) - node2->index(2));

  // 平面方向权重为1.0，高度方向权重降低（如0.5）
  double heu = (dx + dy) + 0.5 * dz;
  double tie_breaker = 1.01;
  return tie_breaker * heu;
}
```

方案 2：修改边权（增加高度变化的代价）

在计算节点间实际移动代价(g(n)时，对高度方向的移动增加惩罚。例如在 AstarGetSucc 中调整边权：

```
// 原：sqrt(dx² + dy² + dz²)
// 修改为：平面距离 + 高度惩罚
edgeCostSets.push_back(sqrt(dx*dx + dy*dy) + 2.0 * abs(dz));
```

对可行性和最优性的影响

- **可行性**：两种修改均不会影响可行性。只要存在从起点到目标的路径，算法仍能找到一条可行路径。
- **最优性**：
  - 方案 1（启发函数修改）：若高度方向权重降低后仍满足 (h(n) ≤实际代价)（可采纳性），则最优性保留；若权重过低导致 h(n)过度低估，可能会丢失最优性。
  - 方案 2（边权修改）：本质是改变了“最优路径”的定义（将高度变化视为更高代价），因此会找到在新代价定义下的最优路径，但可能偏离原欧氏距离意义下的最优路径。

两种方案均能引导无人机减少高度变化，方案 2 的效果更直接（明确惩罚高度移动），但对

最优性的改变更显著；方案 1 更温和，若参数合理可在偏好平面飞行的同时保留接近原最优性的路径。

3.pathSimplify 函数的道格拉斯 - 普克（Douglas-Peucker）实现分析

代码中 pathSimplify 函数通过递归方式实现路径简化，核心逻辑如下：

- **递归分割**：计算路径中所有点到首尾点连线的垂直距离，找到距离最大的点（dmax）。
- **阈值判断**：若 dmax > path_resolution，则以该点为界分割路径并递归简化；否则保留首尾点，丢弃中间点。
- **目标**：在保证路径形状近似的前提下，减少冗余节点，降低轨迹跟踪复杂度。

path_resolution 过大的影响

(1)跟踪误差增大

- **动力学约束冲突**：无人机存在加速度和姿态角限制（如最大俯仰角、滚转角）。当 path_resolution 过大时，简化后的路径可能出现急剧转折，导致：
  - 实际轨迹无法完全跟踪简化路径（因加速度或姿态角超过物理极限），产生显著偏移误差。
  - 简化路径丢失原路径的平滑过渡段，无人机在转折处需剧烈调整姿态，引发超调或震荡。

(2)安全性下降

- **障碍物碰撞风险**：原路径可能通过微调绕开障碍物，但若 path_resolution 过大，因中间避障节点被过滤，简化后的路径可能直接穿过障碍物区域。
- **路径可行性降低**：简化后的路径可能包含无人机动力学无法实现的“跳跃式”转折，导致实际飞行中轨迹与规划路径偏差，误入危险区域。

path_resolution 过小的影响

(1)跟踪效率降低，间接增大误差

- **节点冗余过多**：过小的 path_resolution 会保留大量密集节点，导致：
  - 轨迹跟踪算法需频繁调整控制量，受传感器噪声和执行器延迟影响，累积误差可能增大。
  - 计算负担加重，实时性下降，间接影响跟踪精度。

(2)安全性冗余，但存在潜在风险

- **优势**：路径更接近原始轨迹，可最大限度保留避障细节，降低碰撞风险。
- **潜在问题**：过度密集的节点可能导致轨迹“过于曲折”，无人机频繁小幅度调整姿态，增加能量消耗和机械磨损；同时，微小的路径波动可能被动力学约束放大，引发不必要的姿态抖动，影响稳定性。

小结

- **过大的 path_resolution**：主要风险是路径过度简化，违反无人机动力学约束，导致跟踪误差剧增和碰撞风险。
- **过小的 path_resolution**：主要问题是节点冗余，降低跟踪效率并可能引发姿态抖动，但安全性相对更高。

合理的 path_resolution 需结合无人机的动力学参数（如最大加速度、最大姿态角）设定，通常应略小于无人机在安全加速度下可平稳跟踪的最小转弯半径对应的距离阈值。

2.2 SO(3)位置控制器的力姿态生成

1. 各项在飞行中的作用分析

- **位置/速度误差项（kx.asDiagonal() * (des_pos - pos_) + kv.asDiagonal() * (des_vel - vel_)）这是比例-微分**（PD）控制项，用于修正实际位置、速度与期望状态的偏差。
  - 位置误差项（kx 增益）：当无人机偏离期望位置时，产生与误差成正比的力，推动无人机向期望位置移动，减少静态偏差。
  - 速度误差项（kv 增益）：当实际速度与期望速度不符时，产生阻尼力，抑制超调与震荡，提高系统稳定性。

通过位置偏差（des_pos - pos_）和速度偏差（des_vel - vel_）分别与增益 kx、kv 相乘后叠加，直接修正无人机实际状态与期望状态的偏差。位置误差项推动无人机向期望位置移动，减少静态偏差；速度误差项则提供阻尼作用，抑制运动过程中的超调与震荡，提升系统稳定性。

- **期望加速度前馈（mass_ * des_acc）**基于期望加速度直接引入控制力，提前补偿无人机跟踪期望轨迹所需的基础力，减少动态响应滞后。前馈控制能提高系统对快速变化轨迹的跟踪能力，降低反馈控制的负担。
- **重力补偿（mass_ * g_ * Eigen::Vector3d(0, 0, 1)）**无人机在垂直方向需克服重力，该项通过引入与重力方向相反的力，大小为 mass_ * g_，抵消重力影响，使控制器能专注于跟踪轨迹的动态变化，避免重力干扰导致的稳态误差。
- **ka 项（mass_ * ka.asDiagonal() * (des_acc - acc_)）**ka 项的计算基于机体实测加速度与期望加速度的偏差，即期望加速度减去实测加速度。ka 增益的每个分量遵循特定规则：当对应方向的总误差绝对值超过 3 时，该分量为 0；否则为误差绝对值的 0.2 倍。ka 项的作用是补偿模型误差或外部扰动（如气流）导致的加速度偏差，以此进一步提升加速度跟踪精度，且 ka 的取值会根据总误差动态调整。

2. 大误差或传感器噪声下对 ka 项截断/限幅的原因

代码中 ka 的定义为：

```
Eigen::Vector3d ka(fabs(totalError[0]) > 3 ? 0 : (fabs(totalError[0]) * 0.2),
                   fabs(totalError[1]) > 3 ? 0 : (fabs(totalError[1]) * 0.2),
                   fabs(totalError[2]) > 3 ? 0 : (fabs(totalError[2]) * 0.2));
```

- **大误差场景**：若 ka 随误差无限制增大，会导致对应方向的修正力过大，可能引发无人机剧烈运动（如超调量剧增、高频震荡），甚至超出物理执行范围导致失控。
- **传感器噪声场景**：传感器噪声会使 acc_（机体实测加速度）包含高频干扰，导致 des_acc - acc_出现无规律波动。若 ka 不做限制，噪声会被放大并引入控制力，造成无人机抖动，破坏系统稳定性。
- 截断/本质是通过限制 ka 的最大作用，避免过度修正，保证系统在误差较大或噪声干扰时的鲁棒性。

3. 更重机体保持相似加速度跟踪的调整方式及对稳定性的影响

质量参数与 kx/kv 增益的调整方式

- **质量参数调整**：需通过 setMass 方法将 mass_更新为实际增大后的质量。代码中控制力（force_）的计算直接依赖 mass_，例如重力补偿项（mass_ * g_ * Eigen::Vector3d(0, 0, 1)）和期望加速度前馈相关项（mass_ * des_acc、mass_ * ka.asDiagonal() * (des_acc - acc_)）均与质量成正比，准确的质量参数是保证力值计算准确的基础，否则会因力值不足导致加速度跟踪偏差。
- **kx/kv 增益调整**：由于机体质量增大后惯性增加，相同力作用下加速度变化更平缓，

系统动态响应变慢。需适度调大 kx（位置误差增益）和 kv（速度误差增益）：增大 kx 可增强对位置偏差的修正力度，抵消惯性导致的位置跟踪滞后；增大 kv 能提供更强的阻尼，抑制因惯性增加可能引发的超调或震荡，确保系统响应速度与跟踪精度。

对推力大小、姿态角度的影响

- **推力大小**：推力（force_）在参数调整后必然增大。一方面，重力补偿项随质量增大而直接增加；另一方面，期望加速度前馈项和 ka 项补偿力也因 mass_增大而变大。因此，保持相同加速度跟踪时，更重的机体需要更大的总推力来克服重力并提供所需动态加速度。
- **姿态角度**：姿态角度由推力方向决定，水平加速度依赖推力的水平分力。质量增大后，若需维持相同水平加速度，所需水平分力（与质量成正比）增加，需通过增大姿态倾角（倾斜角度）使推力的水平分力（与倾角正弦值成正比）满足需求。但代码中限制推力方向与垂直方向的夹角不超过 45°（theta = M_PI/2 实际对应 90°，结合 cos(theta)逻辑，实际限制为 45°），若所需倾角接近或超过该限制，系统会通过调整推力大小（如代码中解方程重新计算 s 的逻辑）维持角度约束，可能间接影响垂直方向的力平衡。

对系统稳定性的影响

- **正面影响**：质量增大意味着系统惯性增加，对外界瞬时扰动（如气流）的响应更平缓，抗噪声能力增强，运动更稳定。同时，合理调大 kx 和 kv 可补偿惯性导致的响应滞后，维持跟踪精度与动态性能的平衡。
- **潜在风险**：
  - 若 kx/kv 增益过大，可能导致控制力超调，而大质量的惯性会放大震荡持续时间，引发系统不稳定。
  - 当姿态倾角接近 45°限制时，推力的垂直分力（与倾角余弦值成正比）减小，需更大总推力平衡重力，可能导致垂直方向控制余量不足，极端情况下出现高度波动。
  - 更大的推力需求可能接近执行器（如电机）的最大输出极限，若推力饱和，会导致控制失效，破坏系统稳定性。

2.3 动力学建模与约束

1. 质量、惯量与气动阻尼的建模简化对控制分配的影响

四旋翼动力学模型的核心方程为：

- 平移运动：

$$m\ddot{p} = F + mg$$（F为总推力，m 为质量）

- 旋转运动：

$$I\dot{\omega} = \tau - \omega \times I\omega$$（I为惯量矩阵，τ为控制力矩）

建模简化的影响：

- **质量简化**：控制分配中推力与质量直接相关。若忽略质量变化，会导致推力计算偏差，进而影响轨迹跟踪精度。
- **惯量简化**：实际控制中常假设惯量矩阵为对角阵或采用标称值。简化会导致姿态控制的力矩分配不准确，引发姿态响应延迟或超调。
- **气动阻尼忽略**：默认模型未考虑阻尼力时，控制分配仅补偿重力和期望加速度，实际飞行中因阻尼存在会产生未建模扰动，导致速度跟踪误差。

2. 模型参数估计误差的可观测现象

- **质量估计偏大**：计算推力时会低估所需力$F \propto 1/m$，导致轨迹在重力方向（z 轴）出现**稳态偏差**，悬停时高度低于期望，水平方向跟踪滞后。
- **质量估计偏小**：推力过度补偿，导致轨迹**过冲**，尤其在加速阶段表现明显，可能引发振荡。
- **阻尼估计误差**：
  - 低估阻尼（k_v偏小）：系统阻尼不足，轨迹会出现**高频振荡**，尤其在速度突变时。
  - 高估阻尼（k_v偏大）：响应迟滞，跟踪误差增大，可能出现**稳态偏差**。
- **惯量估计误差**：主要影响姿态控制，导致姿态角超调或响应缓慢，间接引发位置轨迹的**周期性振荡**。

3. 在控制层加入气动阻力的实现思路

在 calculateControl 函数中补偿阻尼力，修改力计算方程：

```
// 原力计算方程基础上增加阻尼补偿项
force_.noalias() =
  kx.asDiagonal() * (des_pos - pos_) +
  kv.asDiagonal() * (des_vel - vel_) +
  mass_ * des_acc +
  mass_ * ka.asDiagonal() * (des_acc - acc_) +
  mass_ * g_ * Eigen::Vector3d(0, 0, 1) +
  kd.asDiagonal() * vel_;  // 新增：kd为阻尼系数矩阵，补偿F_d = -kv*v（此处用+kd*v是因为控制量需抵消阻力）
```

参数整定思路：

- 离线辨识：通过阶跃响应实验，测量不同速度下的阻力影响，拟合k_v初始值。
- 在线调整：将k_v设计为自适应参数，例如根据速度误差动态修正：

```
Eigen::Vector3d kd = base_kd + 0.1 * (des_vel - vel_).cwiseAbs();  // 速度误差越大，阻尼补偿越强
```

- 稳定性约束：确保k_v不超过临界值（避免系统过阻尼），限制$k_v < 2\sqrt{kx \cdot m}$。

通过控制层补偿可更直接地应对气动阻尼的动态影响，同时需结合参数辨识和自适应策略提高鲁棒性。

调整策略对比

调整层面	优点	缺点
控制层（SO3Control）	实时补偿，无需修改轨迹，适应动态变化	需精确估计k_v，参数整定复杂，可能引入噪声放大
规划层（轨迹限速）	简化控制逻辑，通过轨迹约束规避阻尼影响	轨迹保守性增加，无法适应阻尼动态变化，灵活性低

第二部分：代码补全部分

任务一：补全四旋翼飞机的动力学模型

3.1 任务分析

本任务旨在补全 Quadrotor.cpp 中的四旋翼动力学模型核心代码，具体需完善线加速度（v_dot）和角加速度（omega_dot）的计算逻辑，确保四旋翼能准确模拟真实飞行状态，并通过悬停测试验证补全效果。

四旋翼的动力学模型需遵循经典物理定律，核心需解决两个关键物理量的计算：

1. **线加速度（v_dot）**：四旋翼的线运动由合外力决定，需综合考虑重力（竖直向下）、电机总推力（沿机体坐标系 z 轴向上，需转换为世界坐标系）、外部干扰力及空气阻力（与运动方向相反），符合牛顿第二定律（合外力 = 质量 × 加速度）。
2. **角加速度（omega_dot）**：四旋翼的姿态运动由合外力矩决定，需考虑电机产生的控制力矩、外部力矩，同时需补偿角速度引发的科里奥利力耦合效应，遵循欧拉方程（角动量定理）。

3.2 补全思路

1. **v_dot 补全逻辑**：基于牛顿第二定律，先将机体坐标系下的总推力$thrust$通过姿态矩阵 R 的第 3 列$R.col(2)$ 转换为世界坐标系向量，再叠加外部力$external_force_$、抵消重力影响（重力向量为$mass_ \cdot (0,0,g_)$），扣除空气阻力$resistance \cdot vnorm$后，除以四旋翼质量$mass_$得到线加速度v_dot。

```
v_dot = (R.col(2) * thrust + external_force_ - mass_ * Eigen::Vector3d(0, 0, g_) - resistance * vnorm) / mass_;
```

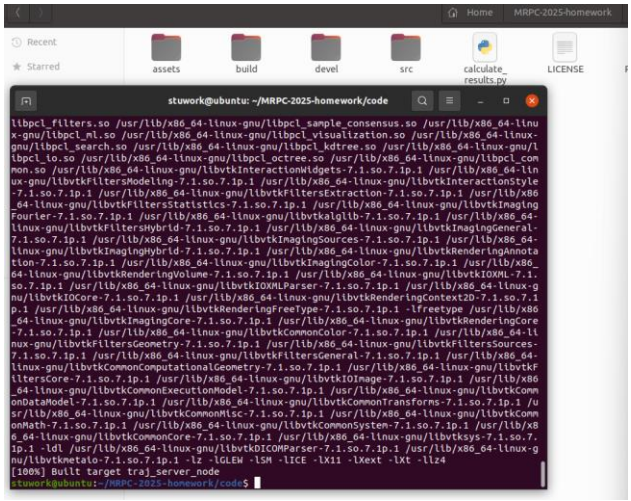
2. **omega_dot 补全逻辑**：基于欧拉方程，先计算控制力矩$moments$与外部力矩$external_moment_$的合力矩，减去科里奥利力耦合项 $cur\_state.omega \times (J\_ \times cur\_state.omega)$（×表示叉乘），再与惯性矩阵的逆$J_.inverse()$相乘，得到角加速度$omega_dot$。

```
omega_dot = J_.inverse() * (moments + external_moment_ - cur_state.omega.cross(J_ * cur_state.omega));
```

3.3 测试过程

步骤 1：编译代码

进入工作空间根目录执行编译命令 catkin_make



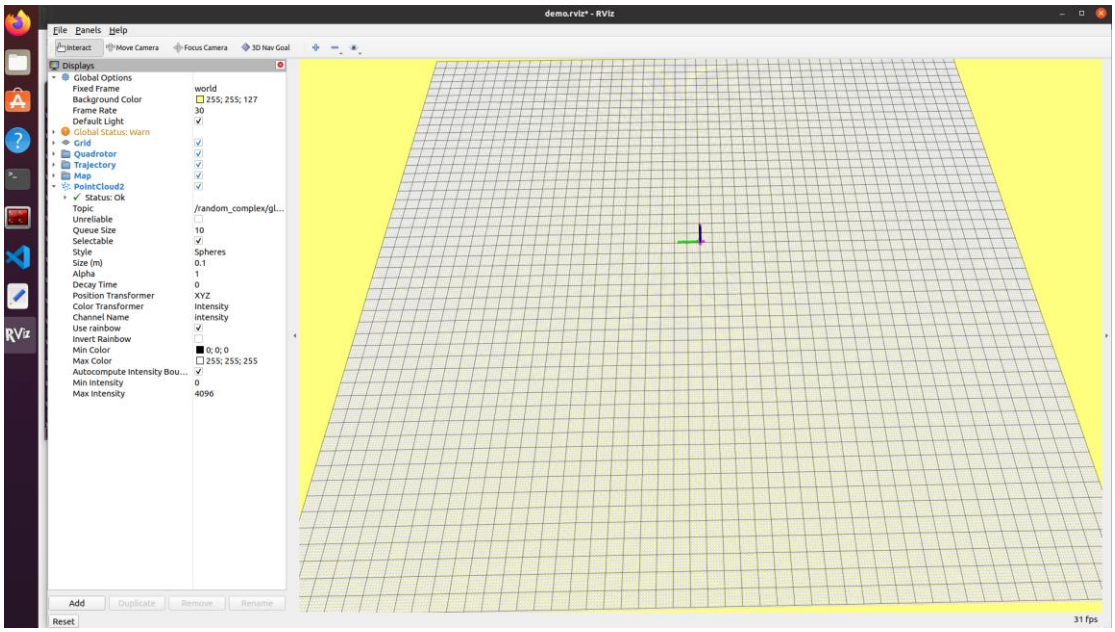
编译过程无报错，最终显示[100%] Built target traj_server_node，表明编译成功。

步骤 2：测试无人机悬停

配置环境变量并启动悬停测试脚本：

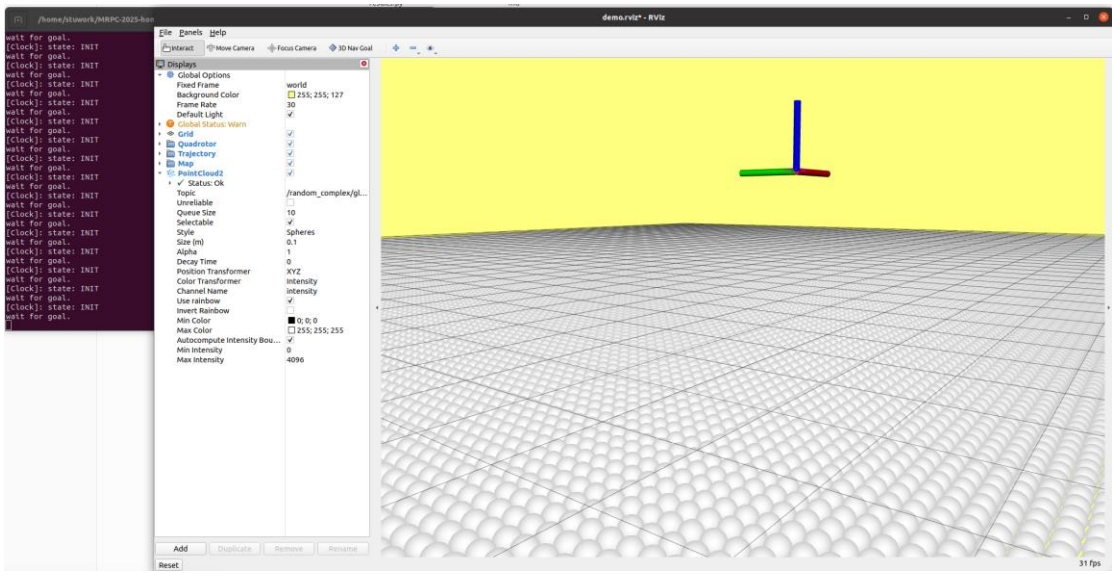
```
source devel/setup.bash
roslaunch trajectory_generator test_control.launch
```

成功弹出 Rviz 可视化界面，界面中四旋翼模型清晰显示，全局状态正常：



步骤 3：验证悬停效果

在 Rviz 界面中观察到，四旋翼稳定悬停于约 3 米高度，位置无明显漂移、姿态保持平稳，说明补全的动力学模型能准确模拟四旋翼的受力与运动关系，满足设计要求。



任务二：补充无人机的前端规划模块

4.1 任务分析

本次任务基于 ROS 框架实现面向无人机的 3D 网格 A*路径规划算法，核心目标是让无人机在已知障碍物的三维环境中，从起点自主规划出一条无碰撞、最短路径至终点。算法需满足三大核心要求：路径最优性（长度最短）、环境安全性（无碰撞）、执行高效性（探索试错次数合理）。

4.2 核心任务补全思路

1. 启发函数（getHeu）补全

核心思路为“保证最优性+减少无效探索”：采用欧式距离作为基础启发值（满足 A*算法最优性的核心条件），叠加极小权重的曼哈顿距离作为 tie-breaker，既避免同等 f_score 节点过多导致的探索效率降低，又不破坏启发值的一致性，确保算法收敛至最短路径。

2. A * 主循环补全

核心思路为“优先级扩展+代价更新”：

- 从 OpenSet 中弹出 f_score 最小的节点，优先探索最优候选节点；
- 扩展节点 26 邻域，过滤无效/被占用节点后计算新路径代价；
- 对未访问节点初始化代价并加入 OpenSet，对已在 OpenSet 的节点仅在新代价更优时更新，保证每个节点的 g_score 始终为当前最优。

3. 路径回溯（getPath）补全

核心思路为“反向遍历+正向输出”：从终点节点开始，通过父节点指针反向遍历至起点，将遍历结果反转后得到从起点到终点的完整路径，确保路径节点顺序符合无人机飞行逻辑。

4.3 算法逻辑理解

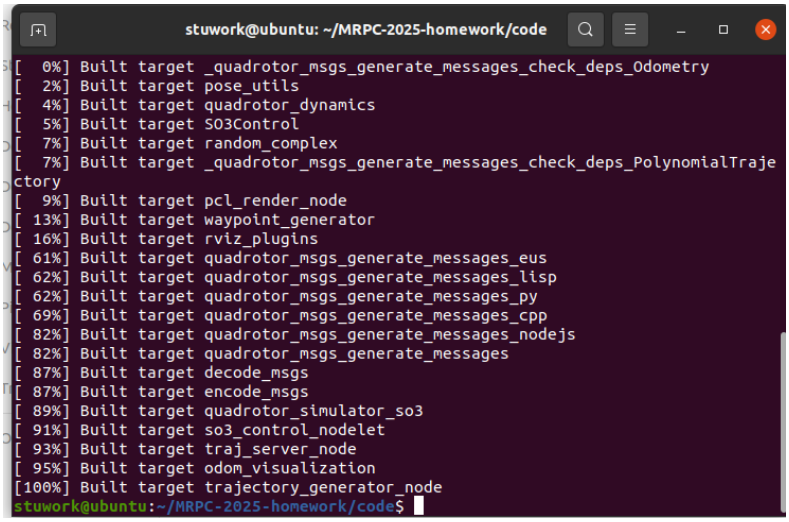
A*算法以“启发式搜索”为核心，通过评估节点的“实际代价（g_score）+预估代价（h_score）”（即 f_score），优先扩展最优节点，最终找到最优路径。核心流程如下：

- **初始化**：将起点节点加入 OpenSet，初始化其 g_score 为 0、f_score 为启发值，其余节点状态重置；
- **迭代搜索**：循环弹出 OpenSet 中 f_score 最小的节点，标记为已访问（CloseSet），扩展其 26 邻域有效节点（过滤越界/被占用节点）；
- **代价更新**：对邻域节点计算新的 g_score，若新代价更优则更新 f_score 并记录父节点，将节点加入/更新至 OpenSet；
- **终止判断**：当弹出的节点为终点时，终止搜索并回溯路径；若 OpenSet 为空则搜索失败；
- **路径处理**：从终点反向遍历父节点至起点，反转后输出正向路径，并通过道格拉斯 - 普克算法简化路径，剔除冗余节点。

4.4 测试过程

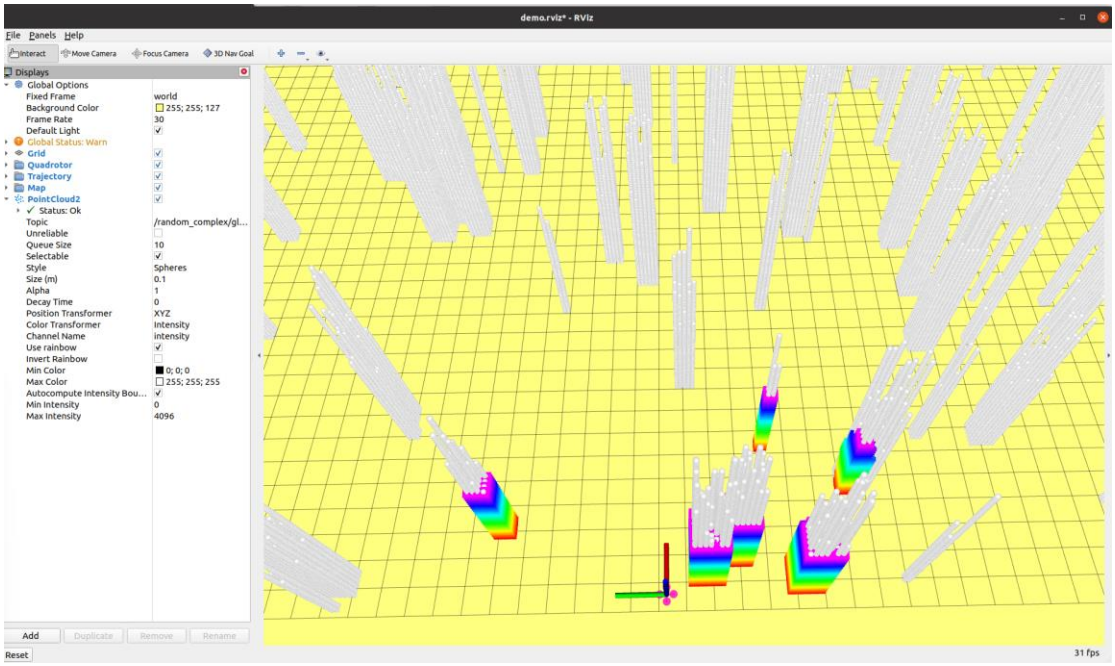
步骤 1：编译代码

进入工作空间根目录执行编译命令 catkin_make



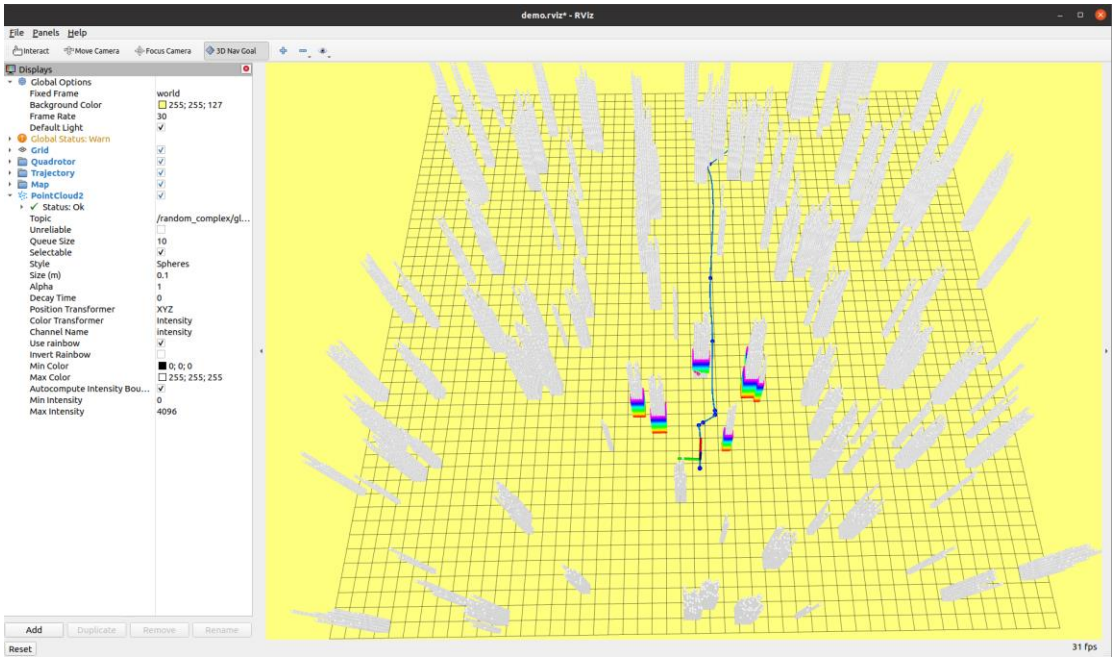
步骤 2：启动仿真

启动 launch 文件，弹出可视化界面：



步骤 3：测试路径规划

点击`3D Nav Goal`，在界面上拖动一下。



运行成功，飞机寻找路径，到达目标点。

任务三：控制器参数调试

附录 1：executor_quaternion.py

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.transform import Rotation as R

# 1. 读取无人机姿态数据
df = pd.read_csv('tracking.csv')
t_list = df['t'].values
qx_b = df['qx'].values  # 无人机四元数虚部x
qy_b = df['qy'].values  # 无人机四元数虚部y
qz_b = df['qz'].values  # 无人机四元数虚部z
qw_b = df['qw'].values  # 无人机四元数实部w

# 2. 题目给定固定参数
omega = 0.5  # 角速度，单位rad/s
alpha = np.pi / 12  # 圆锥半顶角，单位rad
cos_alpha = np.cos(alpha)
sin_alpha = np.sin(alpha)

# 3. 初始化执行器四元数存储列表
exec_qx = []
exec_qy = []
exec_qz = []
exec_qw = []

# 4. 逐时刻计算执行器世界系姿态
for i in range(len(t_list)):
    t = t_list[i]
    # 4.1 无人机四元数转旋转矩阵（世界系→机体系 ^W R_B）
    # scipy库要求输入格式为 (qw, qx, qy, qz)，与csv数据对应
    q_drone = [qw_b[i], qx_b[i], qy_b[i], qz_b[i]]
    R_WB = R.from_quat(q_drone).as_matrix()

    # 4.2 计算机体系→执行器系旋转矩阵 ^B R_D
    cos_omega_t = np.cos(omega * t)
    sin_omega_t = np.sin(omega * t)
    R_BD = np.array([
        [cos_omega_t, -sin_omega_t * cos_alpha, sin_omega_t * sin_alpha],
        [sin_omega_t, cos_omega_t * cos_alpha, -cos_omega_t * sin_alpha],
        [0, sin_alpha, cos_alpha]
    ])

    # 4.3 计算世界系→执行器系旋转矩阵 ^W R_D = ^W R_B × ^B R_D
    R_WD = R_WB @ R_BD

    # 4.4 旋转矩阵转四元数（输出格式：qx, qy, qz, qw）
    q_exec = R.from_matrix(R_WD).as_quat()
```

```
    # 4.5 第一步：四元数归一化
    q_exec = q_exec / np.linalg.norm(q_exec)

    # 4.6 第二步：qw<0时整体取反
    if q_exec[3] < 0:
        q_exec = -q_exec # 整体取反：qx→-qx, qy→-qy, qz→-qz, qw→-qw

    # 存储全量四元数分量
    exec_qx.append(q_exec[0])
    exec_qy.append(q_exec[1])
    exec_qz.append(q_exec[2])
    exec_qw.append(q_exec[3])

# 5. 绘制四元数变化曲线
plt.figure(figsize=(12, 6))
plt.plot(t_list, exec_qx, label='qx', linewidth=1.2, color='#1f77b4')
plt.plot(t_list, exec_qy, label='qy', linewidth=1.2, color='#ff7f0e')
plt.plot(t_list, exec_qz, label='qz', linewidth=1.2, color='#2ca02c')
plt.plot(t_list, exec_qw, label='qw', linewidth=1.2, color='#d62728')

# 图表格式优化（满足报告清晰度要求）
plt.xlabel('Time (s)', fontsize=12)
plt.ylabel('Quaternion Value', fontsize=12)
plt.title('Executor Quaternion in World Frame (qw ≥ 0)', fontsize=14)
plt.legend(loc='best', fontsize=10)
plt.grid(alpha=0.3, linestyle='--')
plt.xticks(np.arange(t_list.min(), t_list.max() + 0.5, 0.5), rotation=45)  # 每0.5秒一个tick，清晰易读
plt.yticks(np.arange(-1.0, 1.1, 0.2))
plt.tight_layout()

# 保存图片
plt.savefig('executor_quaternion_curve.png', dpi=300, bbox_inches='tight')
plt.show()

# 6. 保存结果数据
result_df = pd.DataFrame({
    't': t_list,
    'qx': exec_qx,
    'qy': exec_qy,
    'qz': exec_qz,
    'qw': exec_qw
})
result_df.to_csv('executor_quaternion_result.csv', index=False)
```