



# 《移动机器人控制与规划》期末作业报告

上课时间 : \_\_\_\_\_  
授课教师 : \_\_\_\_\_  
姓名 : \_\_\_\_\_  
学号 : \_\_\_\_\_  
日期 : \_\_\_\_\_

2025 - 2026 学年第一学期  
吕熙敏  
陈宝伊  
23354048  
2026年1月17日

# 目 录

注：点击章节名称即可跳转

## 第一部分：documents 题目报告

---

- 1 题目 1：坐标系转换
  - 1.1 问题分析
  - 1.2 计算思路与推导
  - 1.3 实验结果
- 2 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
  - 2.2 SO(3) 位置控制器的力姿态生成
  - 2.3 动力学建模与约束
- 3 题目3：无人机微分平坦性（Optional）
  - 3.1 问题分析
  - 3.2 计算思路与推导
    - 3.2.1 轨迹方程
    - 3.2.2 速度与加速度计算
    - 3.2.3 构造机体坐标系
    - 3.2.4 旋转矩阵与四元数
  - 3.3 实现细节
    - 3.3.1 代码结构
    - 3.3.2 主要函数
  - 3.4 实验结果

## 第二部分：code 任务报告

---

- 1 实验环境说明
- 2 任务一：补全四旋翼飞机的动力学模型
  - 2.1 补全代码
    - 2.1.1 线加速度 v\_dot
    - 2.1.2 角加速度 omega\_dot
  - 2.2 编译代码
  - 2.3 测试无人机悬停
- 3 任务二：补充无人机的前端规划模块
  - 3.1 启发式函数
  - 3.2 A\* 主循环
  - 3.3 路径追溯
- 4 任务三：控制器参数调试
  - 4.1 调参思路
    - 4.1.1 参数分析
    - 4.1.2 调参过程
  - 4.2 测试过程及结果
- 5 任务四：附加题优化
  - 5.1 路径剪枝优化
  - 5.2 让无人机偏好平面飞行优化
  - 5.3 策略参数微调
    - 5.3.1 重规划频率

5.3.2 路径简化程度

5.3.3 优化效果

5.4 重规划抖动问题修复

5.5 后端轨迹优化：基于曲率的时间分配策略

5.6 前端规划改进：Theta\* 算法

5.6.1 算法选择

5.6.2 实现优化

5.6.3 改进效果

# 第一部分：documents 题目报告

## 1 题目 1：坐标系转换

### 1.1 问题分析

本题需要我们计算执行器在世界坐标系下的姿态（以四元数表示），并绘制出四元数的变化曲线。分析题目可知，系统包含三个主要的坐标系：

- **世界坐标系 (World Frame,  $\mathcal{W}$ )**：惯性参考系。
- **机体坐标系 (Body Frame,  $\mathcal{B}$ )**：固连于无人机本体，其相对于世界坐标系的姿态由 `tracking.csv` 中的四元数序列给出。
- **执行器坐标系 (Device Frame,  $\mathcal{D}$ )**：固连于末端执行器，其相对于机体坐标系的运动为已知的周期性圆锥运动。

### 1.2 计算思路与推导

为了求得末端执行器在世界坐标系下的姿态  ${}^{\mathcal{W}}q_{\mathcal{D}}$ ，我们采用旋转矩阵链式法则进行推导。

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

其中：

- ${}^{\mathcal{W}}R_{\mathcal{B}}$  表示无人机机体相对于世界的旋转矩阵。
- ${}^{\mathcal{B}}R_{\mathcal{D}}$  表示执行器相对于机体的旋转矩阵。

具体计算步骤如下

1. 机体姿态：从给定的轨迹文件中读取每一时刻的四元数  $q_{wb} = [q_x, q_y, q_z, q_w]^T$ ，并利用 `scipy.spatial.transform` 库函数将其转换为旋转矩阵  ${}^{\mathcal{B}}R_{\mathcal{B}}(t)$ 。

2. 相对运动：根据题目给出的公式（1），执行器固连坐标系相对于机体系统的姿态变化矩阵：

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

其中， $\omega = 0.5 \text{ rad/s}$ ,  $\alpha = \pi/12$ 。

3. 合成与四元数转换：根据旋转矩阵链式法则  ${}^{\mathcal{W}}R_{\mathcal{D}} = {}^{\mathcal{W}}R_{\mathcal{B}} \cdot {}^{\mathcal{B}}R_{\mathcal{D}}$ ，计算矩阵乘积得到  ${}^{\mathcal{W}}R_{\mathcal{D}}(t)$ ，随后将其逆变换回四元数形式  ${}^{\mathcal{W}}q_{\mathcal{D}}(t)$

4. 四元数连续性处理（归一化）：由于四元数的  $q$  与  $-q$  表示相同的旋转，为保证输出曲线的连续性而符合题目要求，需对结果进行判断：若计算得到的实部  $q_w < 0$ ，则将整个四元数取反，确保  $q_w \geq 0$ 。

### 1.3 实验结果

根据上述思路推导，编写代码 `t1.py` 实现。得到的末端执行器在世界坐标系下的四元数变化曲线如图 1 所示：

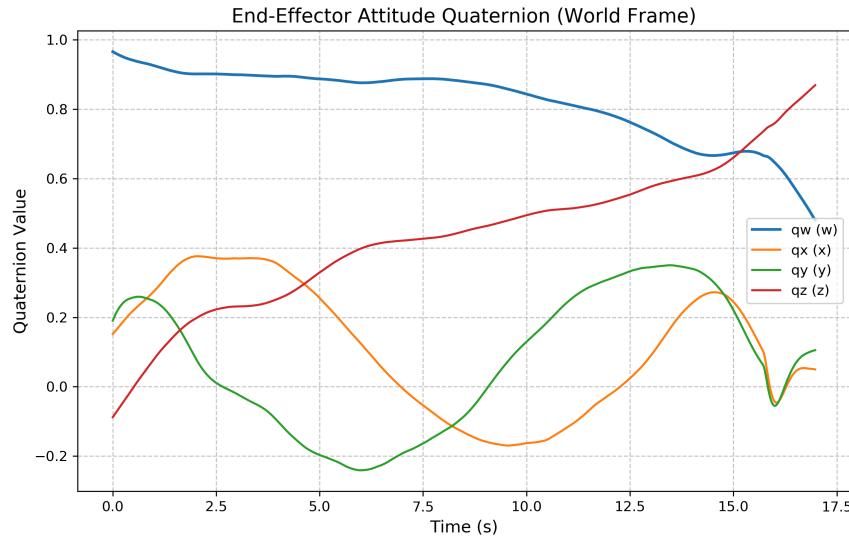


图 1. 四元数各分量变化曲线

## 2 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

现有的 tie\_breaker 会轻微放大启发式以减少“走之”路径。请解释这种写法对开集拓展顺序与路径平滑性的影响；

tie\_breaker 的核心原理是通过在启发式函数  $h(n)$  上乘上一个微小的系数  $(1 + p)$ ，使得算法在面临多个  $f$  值相同的节点选择时，优先选择离目标更近（即原始  $h$  值更小）的节点，从而赋予搜索过程更明确的方向性。

- 对开集拓展顺序影响：**标准的 A\* 会遍历所有  $f$  值相同的节点，导致搜索空间呈“圆形”扩散。引入 tie\_breaker 后，在  $f$  值相近的情况下，算法会优先选择  $h$  值更小（即更靠近终点）的节点。这使得算法的行为在一定程度上向深度优先搜索靠拢，能够显著减少开集扩展的节点数量，提高搜索效率。
- 对路径平滑性影响：**在栅格地图中，由于打破了不同路径间的“平局”，算法倾向于选择一条确定的方向延伸，而不是在等价的栅格间左右横挑。这有效减少了路径中的“之”字形转折，使生成的原始路径更加平直。

如果想让无人机更偏好平面飞行（减少不必要的高度变化），你会如何修改启发或边权？说明对可行性和最优性的影响。

#### 修改边权重 ( $g$ score)

在 AstarGetSucc 函数中计算邻居节点间的 edgeCost 时，对 Z 轴分量进行加权：

$$\text{Cost}(n_{current}, n_{next}) = \sqrt{(x_{next} - x_{curr})^2 + (y_{next} - y_{curr})^2 + (\lambda \cdot (z_{next} - z_{curr}))^2}$$

#### 修改启发式 ( $h$ score)

在 getHeu 函数中，对 Z 轴距离差进行加权：

$$h(n) = \sqrt{(x_{goal} - x_n)^2 + (y_{goal} - y_n)^2 + (\lambda \cdot (z_{goal} - z_n))^2}$$

#### 对可行性和最优性的影响

- 可行性：**不会影响路径的可行性。引入权重  $\lambda$  只是改变了图搜索中各条边的代价，并没有切断图的连通性。原

本可通行的区域依然是连通的，只是通过 Z 轴变化的路径代价变得更高了。只要起点和终点之间存在物理上无碰撞的路径，A\* 算法依然能找到它。

- 最优性：A\* 算法将不再保证找到原代价函数中欧几里得距离最短的路径。无人机可能会为了避免高度变化而选择绕远路。但如果定义“最优”为能耗最小或平稳度最高，那么算法找到的路径在新的代价函数下仍是最优的。
- 由实际修改代码可知，需要同时修改  $g$  和  $h$  才能获得理想效果。如果只修改其中之一，会破坏算法的最优性，甚至发生无视障碍物的情况。从理论分析，仅修改启发式  $h$  而不修改边权  $g$ ，也可能导致  $h(n) > \text{real\_cost}(n)$ ，违反了 A\* 的可采纳性条件。

pathSimplify 采用递归的道格拉斯–普克 (Douglas–Peucker) 思路，按 `path_resolution` 过滤路径节点。结合仿真器的动力学 (加速度与姿态约束)，讨论过大或过小的 `path_resolution` 可能分别导致的跟踪误差与安全性问题。

根据道格拉斯–普克萨算法，首先找到路径中偏离“首尾连线”最远的点，然后以这个点为中心拆分路径为两段，再递归简化子路径。在简化时根据阈值 `path_resolution` 来判断是否移除中间点：点到直线的垂直距离超过该值则保留，否则移除中间点。

#### `path_resolution` 过大

路径被过度简化，原本平滑的曲线变成了由长直线段连接的折线。路径中可能出现明显的尖锐转折点，或者无法绕开障碍物。

1. **跟踪误差：**四旋翼无法瞬间改变速度方向，必须先改变姿态产生水平分力。在仿真器代码中限制了最大推力，进而也就限制了最大加速度，说明无人机无法瞬间大幅度改变方向，无法跟踪超出限制范围的急转弯。当遇到超出限制范围的急转弯时，由于推力饱和，无人机无法提供足够的向心力。结果是无人机在拐弯处发生过冲，实际轨迹会滑向弯道外侧，形成一条偏离原路径的弧线。
2. **安全性：**在障碍物密集的场景下，这种过冲会导致无人机撞上原本规划路径外侧的障碍物。

#### `path_resolution` 过小

路径点过于密集，保留了规划器生成时的抖动或噪声，需要无人机频繁转向达到路径点。

##### 1. 跟踪误差：

在仿真器代码 `Quadrotor.cpp` 中可知，姿态和电机响应受限于转动惯量 `J` 与电机时间常数 `motor_time_constant`。代码 `motor_rpm_dot = (input_ - cur_state.motor_rpm) / motor_time_constant;` 说明电机转速无法突变，存在物理滞后。现实控制中同理。

密集的路径点如果不平滑，要求控制器以极高的频率改变姿态。由于电机响应有一定时间的滞后，且机身有惯性，无人机来不及响应如此高频的指令，导致实际轨迹相对于规划轨迹产生相位滞后。

2. **安全性：**控制器为了达到路径点频率改变姿态，会输出剧烈波动的 `input` (PWM信号)，导致电机频繁在加速和减速间切换，可能导致电机过热或烧毁。而且高频激励可能激发机架结构共振，引起机身剧烈抖动。

## 2.2 SO(3) 位置控制器的力姿态生成

在 `code/src/quadrotor_simulator/so3_control/src/SO3Control.cpp` 中，`calculateControl` 将位置/速度误差、期望加速度前馈、重力补偿和与机体测得加速度相关的  $k_a$  项叠加。

在 `SO3Control.cpp` 的 `calculateControl` 函数中，控制律公式如下：

$$\mathbf{F} = \mathbf{K}_x(\mathbf{p}_d - \mathbf{p}) + \mathbf{K}_v(\mathbf{v}_d - \mathbf{v}) + m\mathbf{a}_d + m\mathbf{K}_a(\mathbf{a}_d - \mathbf{a}) + mg\mathbf{e}_3$$

## 各项在飞行中的作用分析：

- **位置误差反馈  $K_x(\mathbf{p}_d - \mathbf{p})$** : PD 控制中的比例项，根据当前位置与期望位置的偏差产生恢复力，如同虚拟弹簧，保证无人机能够收敛到期望轨迹上。
- **速度误差反馈  $K_v(\mathbf{v}_d - \mathbf{v})$** : PD 控制中的微分项，提供阻尼作用，抑制系统超调和振荡，增强系统在动态跟踪时的稳定性。
- **加速度前馈  $m\mathbf{a}_d$** : 前馈控制项，根据期望轨迹的加速度直接计算所需的力，使系统在不产生误差的情况下也能响应高动态机动，显著减小跟踪滞后。
- **重力补偿  $mge_3$** : 在 Z 轴方向提供恒定的升力以抵消重力，使悬停或水平飞行成为系统的平衡点。
- **加速度误差项  $m\mathbf{K}_a(\mathbf{a}_d - \mathbf{a})$** : 利用机载 IMU 测量的实际加速度进行反馈。该项可以看作是对未建模动力学（如空气阻力）和外部扰动（如阵风）的快速修正，增强系统的鲁棒性。

说明各项在飞行中的作用，并解释为何在大误差或传感器噪声下需要对 `ka` 做截断或限幅。

代码中对 `ka` 进行了截断：当总误差 `totalError` 大于 3 时，强制 `ka = 0`。

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

原因如下：

1. 加速度计测量值通常包含高频噪声和振动。如果不加限制地引入加速度反馈，噪声会被控制器放大，导致电机高频抖动，引起机体震荡甚至系统失稳。
2. 当跟踪误差过大时（例如系统初始化或遭受剧烈撞击），线性控制律计算出的期望加速度可能极其巨大。此时若强行加入加速度反馈，会导致计算出的推力方向剧烈变化，甚至使姿态解算出现奇异。
3. 在大误差阶段，主要依靠位置和速度项 ( $K_x, K_v$ ) 控制；只有在误差较小、进入精细跟踪阶段时，引入加速度项 ( $K_a$ ) 才能有效提升精度。

控制器会根据总力方向生成姿态，并将倾角限制在约  $45^\circ$ 。若更重的机体需要保持相似的加速度跟踪，你会如何同时调整质量参数与 `kx/kv` 增益？讨论对推力大小、姿态角度以及系统稳定性的预期影响。

## 参数调整策略

假设机体质量  $m$  增加，为了保持相似的加速度跟踪性能，保持闭环系统的自然频率  $\omega_n$  和阻尼比  $\zeta$  不变，需要做如下调整：

1. **质量参数 (`mass_`)**: 更新为新的实际质量。因为代码中前馈项 `mass_ * des_acc` 和重力补偿 `mass_ * g_` 显式依赖质量。如果此参数偏小，无人机会因为重力补偿不足掉高。
2. **反馈增益 ( $K_x, K_v$ )**: 按比例增大

在代码实现中，反馈力直接计算为  $F_{fb} = K_x e_p + K_v e_v$ 。根据牛顿定律  $\ddot{x} = F/m$ 。若要保持加速度  $\ddot{x}$  的响应特性不变，例如当  $m$  变为  $2m$  时，所需的力  $F$  也要变为  $2F$ 。因此， $K_x$  和  $K_v$  应乘以相同的倍率，随质量线性增加。如果不调整  $K_x, K_v$ ，只调整 `mass_`，系统响应会变慢，轨迹跟踪误差会变大。

## 对系统特性的预期影响

1. **推力大小**: 推力会显著增加。悬停推力直接由  $mg$  决定，质量加倍，悬停推力也会加倍。
2. **姿态角度**: 在跟踪相同轨迹时，姿态角基本保持不变。姿态角由总推力向量的方向决定 ( $\theta \approx \arctan(F_{horizontal}/F_{vertical})$ )。当质量加倍且增益正确缩放后，水平分力（用于水平加速）和垂直分力（用于抵消重力）都同时加倍。向量的模长变了，但向量的方向不变。
3. **系统稳定性**:

- 若正确调整增益：稳定性保持不变，动态响应与轻量机体一致。
- 若不调整增益：系统带宽下降。对于较重的机体，原有的  $K_x$  显得太小，会导致抗干扰能力变差。但同时，由于惯性大，系统对高频噪声的敏感度可能会略微降低。

那能不能结合两个代码再回答一次第一问：聚焦无人机动力学本身的建模与约束：结合仿真器（四旋翼刚体+重力+推力）说明质量、惯量与气动阻尼的建模简化如何影响控制分配；

## 2.3 动力学建模与约束

聚焦无人机动力学本身的建模与约束：结合仿真器（四旋翼刚体+重力+推力）说明质量、惯量与气动阻尼的建模简化如何影响控制分配；

在无人机控制系统中，控制分配依赖于动力学模型将期望的合外力  $F$  和力矩  $\tau$  映射到四个电机的转速  $\omega_i$ 。简化的影响如下：

### 质量

仿真器中的平动动力学方程为  $m\ddot{p} = RF_T e_3 - mge_3 - F_{drag}$ 。控制器通常简化为  $F_{drag} \approx 0$ 。控制分配时，期望推力  $F_{des}$  直接转化为油门。若模型质量参数不准，计算出的悬停油门基准将产生偏差，直接导致垂直方向的推力过大或过小，进而影响垂直轨迹的跟踪速度。

### 惯量

转动动力学方程为  $J\dot{\omega} = \tau - \omega \times J\omega$ 。通常简化假设  $J$  为对角矩阵（忽略非对角元素的惯性积）。然而，实际无人机若重心不完全在几何中心，非对角项存在。这种简化会导致轴间耦合。例如，控制器只想进行俯仰（Pitch）操作，但由于惯量耦合，实际会激发轻微的滚转（Roll），导致控制分配解算出的电机指令无法完美实现纯解耦运动。

### 气动阻尼

通常模型忽略与速度成正比的空气阻力项 ( $-Dv$ )。在高速运动时，这一简化会导致前馈力不足。控制分配仅根据  $F = ma$  提供加速力，而未分配额外的力来克服空气阻力，导致实际加速度小于期望值。

如果质量、阻尼估计有误，预期在轨迹跟踪中会出现什么可观测现象（如稳态偏差、过冲、振荡）？

如果在 **SO3Control**（主要为 PD 控制 + 前馈，无积分项）的条件下：

1. 质量估计有误 ( $m_{est} \neq m_{real}$ )：出现Z轴稳态偏差。
  - 若质量估计小了，重力补偿不足，无人机会在期望高度下方平衡
  - 若质量估计大了，重力补偿过量，无人机会在期望高度上方平衡。
2. 阻尼估计有误（或忽略阻尼）：发生轨迹跟踪滞后和动态下的稳态误差。当无人机以恒定速度  $v$  飞行时，实际受力包含阻力  $F_{drag} \approx -k_d v$ 。控制器若忽略此项，仅提供  $F = 0$  的水平分力。结果是无人机速度将下降，直到位置误差产生的反馈力  $K_p e_p$  增大到足以抵消空气阻力。会表现为无人机在水平方向上始终落后于期望轨迹。若强行增大反馈增益  $K_p$  来减小滞后，可能会引发高频振荡或过冲。

若想在模型中加入简单的气动阻力，你会在控制层（SO3Control）还是规划层（轨迹限速）做出调整？说明各自的优缺点，并给出一个调整参数的思路。

加入简单的线性气动阻力模型，假设为  $F_{drag} = -Dv$ ：

1. 控制层调整（SO3Control）：在计算总推力时，增加前馈补偿项，公式调整为：`force_ = ... + mass_* des_acc + D * des_vel`。
  - **优点：高性能。**主动抵消阻力，使得无人机能更精确地跟踪高速、高动态轨迹，减少跟踪误差。
  - **缺点：依赖模型参数。**需要精确测定阻力系数  $D$ 。若  $D$  估计过大，会导致系统不稳定（正反馈）。且会增

加控制器的计算复杂度。

2. 规划层调整（轨迹限速）：在生成期望轨迹时，限制最大速度  $v_{max}$  和最大加速度  $a_{max}$ 。

- **优点：**安全且鲁棒。不需要修改底层控制律，避免了因参数错误导致的失稳风险。通过降低速度，自然减小了空气阻力的影响（阻力与速度平方或一次方成正比）。
- **缺点：**牺牲性能。限制了无人机的机动能力，无法发挥其最大飞行潜力。无法从根本上消除阻力带来的跟踪误差，只是减小了其量级。

## 参数调整思路

选择在控制层进行前馈补偿，参数  $D$ （阻力系数）的调整思路如下：

1. **开环测试：**让无人机在仿真器中以不同的恒定速度  $v$  平飞。
2. **数据记录：**记录维持该速度所需的额外推力分量（即除去  $ma$  之外的力）。
3. **拟合：**根据  $F_{extra} \approx D \cdot v$ （线性）或  $F_{extra} \approx k \cdot v^2$ （二次），通过最小二乘法拟合出阻力系数  $D$  或  $k$ 。
4. **验证：**将拟合出的  $D$  写入 **S03Control**，观察高速飞行时的位置跟踪误差是否显著减小。

## 3 题目3：无人机微分平坦性（Optional）

### 3.1 问题分析

四旋翼无人机是一个微分平坦系统，其平坦输出为位置  $(x, y, z)$  和偏航角  $\psi$ 。这意味着系统的所有状态和控制输入都可以通过平坦输出及其有限阶导数表示。

本题旨在基于微分平坦性特性，由给定的位置轨迹反解四旋翼无人机的姿态。

1. **轨迹与约束：**给定世界坐标系下的双纽线轨迹，且规定偏航角始终与运动速度方向保持对齐。
2. **坐标系构建：**采用题目定义的“前-左-上”机体坐标系。需利用轨迹的加速度矢量确定推力方向（即  $z_B$  轴），结合航向向量构建完整的旋转矩阵。
3. **输出要求：**解算结果需转换为四元数形式，并严格满足归一化及  $qw \geq 0$  的约束，最终按时间步长输出离散化的姿态数据。

### 3.2 计算思路与推导

根据四旋翼的微分平坦性理论，系统的全状态可由平坦输出  $\sigma = [x, y, z, \psi]^T$  及其导数代数确定。本题中，位置轨迹  $[x(t), y(t)]$  已知 ( $z(t) = 10$ )，偏航角  $\psi$  由速度方向决定。具体解算步骤如下：

#### 3.2.1 轨迹方程

题目给定的双纽线轨迹（世界坐标系）：

$$\begin{cases} x = \frac{10 \cos t}{1 + \sin^2 t} \\ y = \frac{10 \sin t \cos t}{1 + \sin^2 t} \\ z = 10 \end{cases}$$

其中  $t \in [0, 2\pi]$ 。

#### 3.2.2 速度与加速度计算

对给定的轨迹方程进行一阶和二阶求导，得到速度  $\mathbf{v}$  和加速度  $\mathbf{a}$ ：

$$\mathbf{p} = \begin{bmatrix} x(t) \\ y(t) \\ 10 \end{bmatrix}, \quad \mathbf{v} = \dot{\mathbf{p}} = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ 0 \end{bmatrix}, \quad \mathbf{a} = \ddot{\mathbf{p}} = \begin{bmatrix} \ddot{x}(t) \\ \ddot{y}(t) \\ 0 \end{bmatrix}$$

### 3.2.3 构造机体坐标系

根据牛顿第二定律，四旋翼的推力  $T$  沿机体  $z$  轴产生，合力提供了重力补偿和运动加速度：

$$T\mathbf{z}_B = m(\mathbf{a} + g\mathbf{z}_W)$$

其中  $\mathbf{z}_W = [0, 0, 1]^T$  为世界系重力方向。对合加速度向量归一化，即得到机体  $z$  轴单位向量：

$$\mathbf{z}_B = \frac{\mathbf{a} + g\mathbf{z}_W}{\|\mathbf{a} + g\mathbf{z}_W\|} = \text{normalize} \left( \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ g \end{bmatrix} \right)$$

题目要求偏航角  $\psi$  始终与速度方向对齐，故：

$$\psi(t) = \text{atan2}(\dot{y}(t), \dot{x}(t))$$

据此构造期望的航向方向向量  $\vec{n}_\psi$ （即速度方向在水平面的投影）：

$$\vec{n}_\psi = \begin{bmatrix} \cos \psi \\ \sin \psi \\ 0 \end{bmatrix}$$

依据题目定义的“前-左-上”（FLU）机体坐标系，我们利用已确定的  $\mathbf{z}_B$  与航向向量  $\vec{n}_\psi$  构建正交基：

1. 确定  $y_B$  轴 (Left)：机体左侧方向应垂直于“推力方向”和“航向方向”所构成的平面：

$$\mathbf{y}_B = \text{normalize}(\mathbf{z}_B \times \vec{n}_\psi)$$

2. 确定  $x_B$  轴 (Front)：根据右手定则，由  $y_B$  和  $z_B$  叉乘得到机体前方：

$$\mathbf{x}_B = \mathbf{y}_B \times \mathbf{z}_B$$

### 3.2.4 旋转矩阵与四元数

将上述基向量组合得到旋转矩阵  $R_{WB} = [\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B]$ 。最后将  $R_{WB}$  转换为四元数  $\mathbf{q} = [w, x, y, z]^T$ 。为满足作业的唯一性约束，做如下处理：

- 归一化：保证  $\|\mathbf{q}\| = 1$ 。
- 符号修正：若  $q_w < 0$ ，则  $\mathbf{q} \leftarrow -\mathbf{q}$ ，确保  $q_w \geq 0$ 。

## 3.3 实现细节

题目要求创建 ROS Package，编写代码实现解题过程并可视化结果。

### 3.3.1 代码结构

```
1 quadrotor_df/
2   ├── CMakeLists.txt
3   └── package.xml
4   └── src/
5     └── calculate_attitude.cpp  # 主计算程序
6   └── scripts/
7     └── plot_quaternion.py      # 绘图脚本
```

### 3.3.2 主要函数

- `getPosition(t)`：计算给定时刻的位置
- `getVelocity(t)`：计算速度
- `getAcceleration(t)`：计算加速度
- `calculateAttitude(t)`：计算姿态四元数

- `rotationMatrixToQuaternion()` : 旋转矩阵转四元数

### 3.4 实验结果

根据计算得到的 `df_quaternion.csv` 的结果绘制出四元数的变化曲线如图 2 和图 3 所示

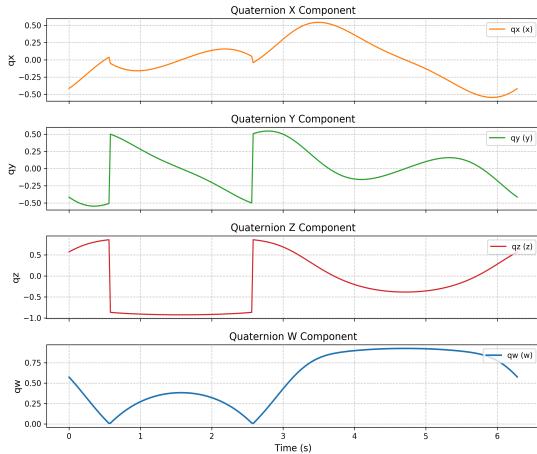


图 2. 四元数分量图

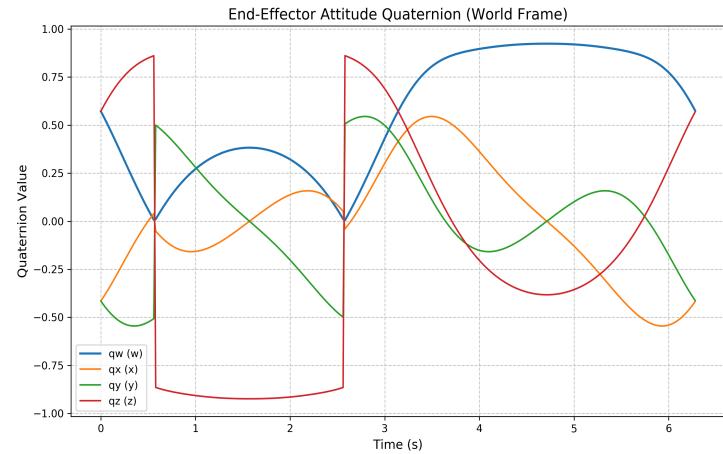


图 3. 合并趋势图

## 第二部分：code 任务报告

### 1 实验环境说明

实验未使用课程提供的 VMware 虚拟机镜像（Ubuntu 20.04）标准环境，使用物理机双系统 + Docker 容器化的部署方案。参考课程虚拟环境说明，具体配置如下：

- 操作系统: Ubuntu 22.04 LTS
- 运行环境: Docker 容器环境
- ROS 版本: ROS Noetic Ninjemys
- 环境构建说明: 由于 Ubuntu 22.04 原生支持 ROS 2 (Humble)，不再直接支持 ROS 1 (Noetic)。为确保与课程要求的 ROS Noetic 环境严格一致，通过 Docker 部署了独立的 ROS Noetic 容器。经测试，该环境与课程提供的 Ubuntu 20.04 虚拟机环境在代码编译、仿真运行及话题通信上完全兼容，实验结果有效。

### 2 任务一：补全四旋翼飞机的动力学模型

#### 2.1 补全代码

`Quadrotor.cpp` 中描述了四旋翼的动力学模型，负责模拟真实无人机的物理特性、计算无人机的运动、为控制器提供反馈，任务要求我们补充其中的线加速度和角加速度的计算代码。

##### 2.1.1 线加速度 v\_dot

根据牛顿第二定律，加速度 = 所有力的合力 / 质量。阅读代码可以发现需要考虑的力有：重力 `mass_*g`、推力 `thrust`、空气阻力 `resistance`、外力 `external_force_`，得到计算公式：

$$\dot{v} = \frac{1}{m}(F_{gravity} + F_{thrust} + F_{resistance} + F_{external})$$

其中需要注意：重力需要转化为向量与其他力统一形式；推力沿机体z轴方向，需要 `* R.col(2)` 转换到世界坐标系。补充代码如下：

```
1 | v_dot = -g_ * Eigen::Vector3d(0, 0, 1) + thrust * R.col(2) / mass_
2 |         - resistance * vnorm / mass_ + external_force_ / mass_;
```

### 2.1.2 角加速度 omega\_dot

根据欧拉方程：

$$\dot{\omega} = J^{-1}(\tau - \omega \times J\omega + \tau_{external})$$

阅读代码可知：

- `J_.inverse()`：惯性矩阵的逆
- `moments`：螺旋桨产生的力矩
- `cur_state.omega.cross(J_ * cur_state.omega)`：科里奥利力矩
- `external_moment_`：外部力矩

补充代码如下：

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

## 2.2 编译代码

```
1 | cd code
2 | catkin_make
```

编译结果：

```
1 | ...
2 | [100%] Built target traj_server_node
```

## 2.3 测试无人机悬停

```
1 | source devel/setup.bash
2 | roslaunch trajectory_generator test_control.launch
```

由于使用Docker，容器被拒绝了访问显示器权限，可视化窗口创建失败。在终端运行以下命令，添加权限。

```
1 | bonnie@bonnie:~$ xhost +local:root
2 | non-network local connections being added to access control list
```

Rviz 可视化中无人机已经停在空中约 3 米的位置上，测试通过。

# 3 任务二：补充无人机的前端规划模块

## 3.1 启发式函数

任务要求补全 `Astar_searcher.cpp` 中的 `getHeu()` 函数。

1. 距离表示方式：在本项目中，由于无人机可以在 3D 空间自由移动（包含对角方向），使用欧几里得距离。

```
1 | double heu = sqrt(dx*dx + dy*dy + dz*dz);
```

2. Tiebreaker 方法：通过 documents 作业题目 1 可知，Tiebreaker 通过微调启发式值来打破多个节点的 f 值相同的

平局，减少搜索节点数量，使路径更平滑。实现选择使用缩放因子法 `tie_breaker`。

```
1 | double tie_breaker = 1.0 + 1.0 / 10000.0;
2 | heu = heu * tie_breaker;
```

## 3.2 A\* 主循环

任务要求补全 A\* 算法的主循环部分。主要分为以下几个部分：

1. 弹出  $f$  值最小的节点：从 Open Set 中弹出  $f(n)$  值最小的节点  $n_{curr}$  (`currentPtr`) 作为当前扩展节点，并将其标记为已访问，加入 Closed Set，避免重复处理。

```
1 | currentPtr = Openset.begin()->second;
2 | Openset.erase(Openset.begin());
3 | currentPtr->id = -1; // 移入 Closed List
```

2. 判断是否到达终点：通过检查 `index` 与 `goalIdx` 是否一致，检查  $n_{curr}$  是否为目标节点。若是，则搜索结束，并将该指针将作为后续路径回溯过程的起始锚点 (`terminatePtr = currentPtr`)；否则继续执行。

```
1 | if (currentPtr->index == goalIdx) {
2 |     terminatePtr = currentPtr;
3 |     ROS_WARN("[A*] Goal reached!");
4 |     return true;
5 | }
```

3. 扩展当前节点：调用 `AstarGetSucc` 函数获取当前节点周围的邻居节点集合。

```
1 | AstarGetSucc(currentPtr, neighborPtrSets, edgeCostSets);
```

4. 处理未访问的邻居节点：再遍历所有邻居节点时，对于没有被访问过 (`id==0`) 节点，需要初始化其  $g$  值与  $h$  值，并设置父节点为  $n_{curr}$ ，计算  $f = g + h$  后加入 Open Set。

```
1 | neighborPtr->g_score = tentative_g_score;
2 | neighborPtr->f_score = tentative_g_score + getHeu(neighborPtr, endPtr);
3 | neighborPtr->Father = currentPtr;
4 | neighborPtr->id = 1;
5 | Openset.insert(make_pair(neighborPtr->f_score, neighborPtr));
```

## 3.3 路径追溯

任务要求补充完"返回 A\* 寻找到的路径"的部分。`getPath()` 函数在 A\* 搜索成功后被调用。此时：

`terminatePtr` 指向终点节点；每个节点都有 `Father` 指针指向其父节点，通过不断回溯 `Father`，可以得到从起点到终点的完整路径。

原有的代码已经将路径节点从终点到起点的顺序存入 `front_path`。那么需要补充将 `front_path` 中的路径反转，并转换为坐标向量返回：

```
1 | // 将起点也加入路径
2 | terminatePtr->coord = gridIndex2coord(terminatePtr->index);
3 | front_path.push_back(terminatePtr);
4 |
5 | // 反转路径 (从终点到起点 -> 从起点到终点)
6 | reverse(front_path.begin(), front_path.end());
7 |
8 | // 提取坐标向量
9 | for(const auto& node : front_path) {
10 |     path.push_back(node->coord);
11 | }
```

## 4 任务三：控制器参数调试

任务要求我们对 `so3_control_nodelet.cpp` 中函数 `position_cmd_callback` 里的 `kx_` 和 `kv_` 参数进行调整。

### 4.1 调参思路

#### 4.1.1 参数分析

同 documents 作业题目 2 中的分析，由代码 `S03Control.cpp` 中位置控制器的力计算公式：

```
1 | force = kx*(pos_des-pos) + kv*(vel_des-vel) + m*acc_des+m*g*[0,0,1]
```

这是典型的 PD 控制器架构。其中：

- **kx (位置增益)**：PD 控制中的比例项。其大小决定了系统对位置误差的响应速度。kx 越大，无人机消除位置误差的动力越足，响应越快；但若过大，会导致系统在目标位置附近产生超调甚至剧烈震荡。
- **kv (速度增益)** PD 控制中的微分项。kv 的主要作用是提供系统稳定性，抑制由 kx 引起的震荡。适当增大 kv 可以减少超调，使收敛过程更平滑；但若过大，会使系统反应变得迟钝。

#### 4.1.2 调参过程

1. **参数对称化处理**：首先将 X 轴与 Y 轴的控制参数统一。由于四旋翼飞行器在水平面具有结构对称性，调整  $k_{x,0} = k_{x,1}, k_{v,0} = k_{v,1}$ ，保证机体在各个水平航向上的响应特性一致。在 xy 方向不统一时明显可以看出无人机存在左右摇摆的现象。
2. **辅助可视化与调试**：为了方便分析控制器在动态过程中的表现（如超调、稳态误差及收敛性），在 `S03Control.cpp` 中增加了数据记录功能，将每一时刻的期望状态、实际状态及控制量输出保存为 .txt 文件。同时编写可视化脚本 `plot_debug.py`。通过生成的位置/速度跟踪曲线、误差分布图及相平面图，可以比单纯观察仿真过程更快速地定位参数问题
3. **迭代调参结果**：依据上述可视化工具，遵循“先调  $k_x$  提升响应速度，后调  $k_v$  抑制振荡”的原则，调整参数。

### 4.2 测试过程及结果

任务要求运行仿真程序后使用评估脚本 `calculate_results.py` 得到分数。测试时发现项目源代码多处地方使用了绝对路径打开 txt 文件，由于没有使用指定虚拟机进行实验，因此将绝对路径 `/home/stuwork/MRPC-2025-homework/` 均替换为 `/home/bonnie/File/noetic_ws/MRPC-2025-homework/`。

且评估脚本中试图写入 `MRPC-2025-homework/solutions/result.txt`，但由于目录不存在发生报错。故运行前手动创建 `solutions` 文件夹与 `redukt.txt` 文件。

调参后得到最终指标与调参前对比如下：

参数组合	kx	kv	RMSE	运行时间	轨迹长度	是否碰撞	得分
初始值	(15.7, 8.7, 6.0)	(6.4, 13.4, 4.0)	~0.089	~69.169	~36.347	0	~38.882
调整后	(15.7, 15.7, 15.0)	(6.4, 6.4, 4.0)	~0.039	~76.260	~34.016	0	~29.804

在调参过程中发现 z 轴存在难以消除的瞬时大幅度跟踪误差。结合可视化图像分析（见图 4 和图 5），实际轨迹在某些时刻无法跟随期望轨迹的剧烈变化，仿真中表现为无人机在某些节点会突然向上飞翻转后回到原规划轨迹。单纯增大  $k_x$  或  $k_v$  已无法解决该问题，且极易引发高频振荡，该问题需在后续优化解决。

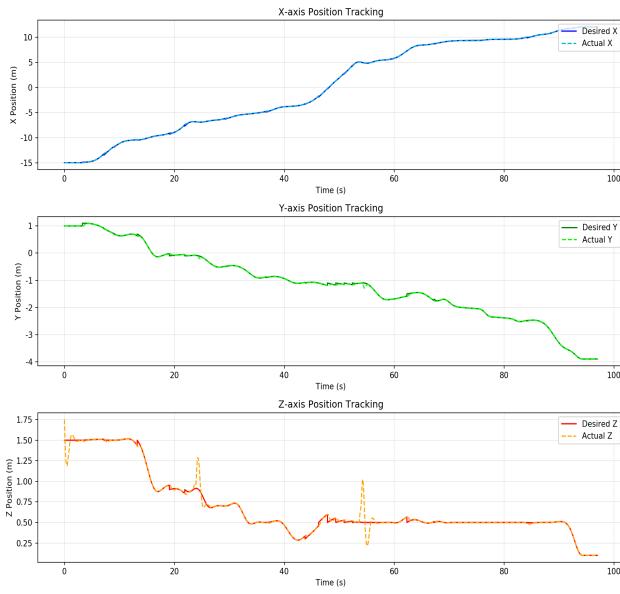


图 4. 位置跟踪图

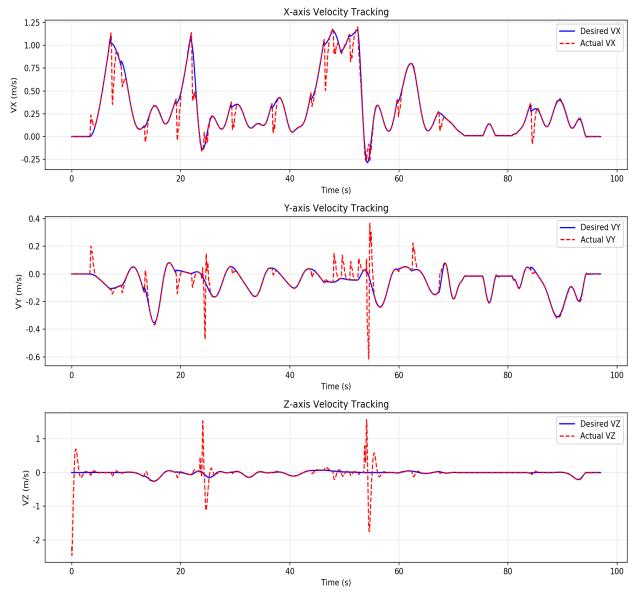


图 5. 速度跟踪图

## 5 任务四：附加题优化

### 5.1 路径剪枝优化

**问题：**根据仿真运行过程可以很轻易地发现，A\* 算法生成的路径中间点过多，特别是在靠近障碍物时会生成大量密集路径点，导致无人机频繁转向前进缓慢，轨迹不够平滑。

**改进思路：**为解决上述问题，设计并实现了一种双阶段路径剪枝算法，目的是保证在无碰撞的前提下，最大程度精简路径点。

1. **第一阶段：视线剪枝** 该阶段基于贪心策略，尝试连接路径上的非相邻点以“拉直”路径。为防止过度剪枝导致的轨迹偏离风险，引入了双重约束：

- **安全约束：**新增 `segmentCollisionFree` 函数对拟合直线进行高分辨率采样检测，确保新生成的直线段完全无碰撞。
- **距离约束：**限制单段路径的最大长度和前瞻点数（参数可调整）。这一改进有效防止了因线段过长导致的多项式轨迹拟合误差过大，避免了无人机在长距离飞行中偏离安全走廊。

2. **第二阶段：共线点移除** 在第一阶段的基础上，进一步检查路径点的几何关系。计算中间点到前后两点连线的垂直距离，若距离小于设定阈值（`path_resolution`），则视为共线冗余点予以移除。同时，在此阶段再次执行碰撞检测与长度校验，确保移除操作不会引入新的安全隐患。

**优化效果：**改进后的算法显著减少了路径点数量，生成的轨迹更加直观、平滑。通过引入最大段长限制，有效平衡了“路径简洁性”与“轨迹拟合精度”之间的矛盾，显著提升了无人机的飞行流畅度，减少了运行时间。且路径剪枝后，调参过程中发现的 z 轴的瞬时大幅度跟踪误差也不再出现。

操作	RMSE	运行时间	轨迹长度	是否碰撞	得分
剪枝前	~0.039	~76.260	~34.016	0	~29.804
剪枝后	~0.042	~36.209	~29.976	0	~21.728

## 5.2 让无人机偏好平面飞行优化

**问题及改进思路：**无人机在运动过程中包含不必要的上下移动。受 documents 作业题目 1 启发，通过增加 Z 轴代价权重，让 A\* 算法倾向于在水平面内规划路径。修改的理论基础已在前文详细分析，此处不再过多赘述。具体修改以下两个部分：

- **边代价计算：**在扩展邻居节点计算  $g$  值时，将 Z 轴的位移分量  $dz$  乘以权重  $\lambda$ ，实现中设置  $\lambda = 2.0$ 。
- **启发式函数：**为了保持搜索的一致性并引导搜索方向，`getHeu` 函数中的启发式估计同样应用了该权重。确保了算法在评估“剩余距离”时也能感知到垂直方向的高代价。

**优化效果：**图 6 展示了引入 Z 轴权重前后的飞行高度变化对比。优化后的路径在非必须变高区域保持了更平稳的飞行高度，显著减少了不必要的上下移动的现象，提升了轨迹的平滑度与飞行的稳定性。

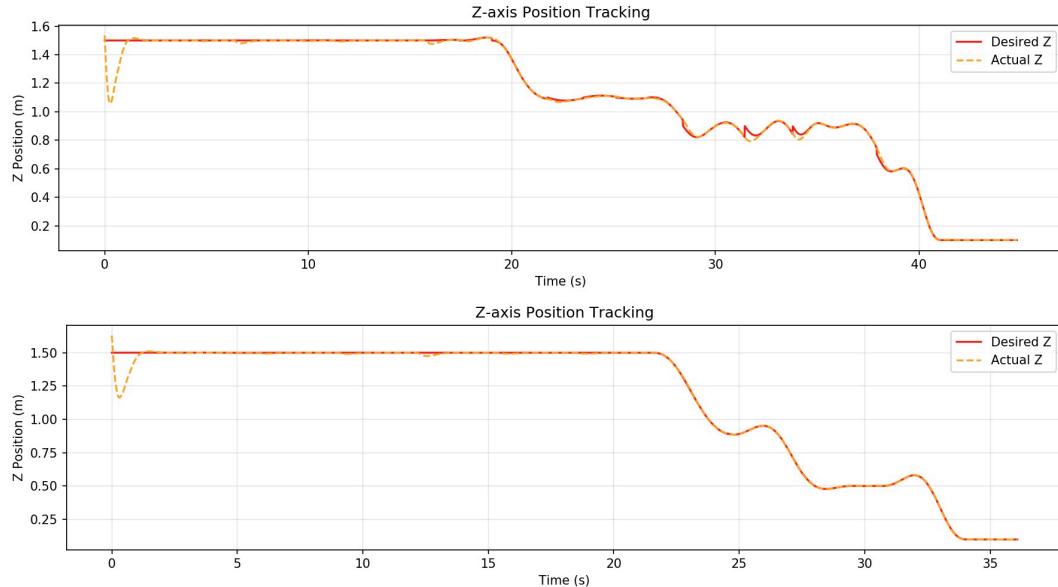


图 6. 引入 Z 轴权重前（上）与引入后（下）的飞行高度变化对比

## 5.3 策略参数微调

为了进一步提升系统的鲁棒性与轨迹平滑度，针对特定的控制策略参数进行了微调优化。

### 5.3.1 重规划频率

- **问题分析：**原始代码设定每隔 1.0 秒强制触发一次轨迹重规划。在静态环境或慢速飞行任务中，过高的重规划频率会导致飞行器频繁切换轨迹段，引发不必要的控制量跳变，进而导致机体抖动。
- **调整方案：**将重规划触发阈值 `t_replan` 从 1.0s 延长至 3.0s。

### 5.3.2 路径简化程度

- **问题分析：**在之前的路径剪枝算法中，前瞻点数限制较为保守，导致在开阔区域仍保留了多余的中间航点，限制了多项式轨迹生成的平滑潜力。
- **调整方案：**将视线剪枝的搜索窗口 `MAX_LOOK_AHEAD` 参数从 10 提升至 15。

### 5.3.3 优化效果

微调参数后，轨迹连续性更好，进一步减少了由于航点切换引起的加减速过程。虽然这在理论上降低了对动态障碍物的响应速度，但在本次作业的静态环境设定下，极大地提升了飞行的平稳性，分数有明显下降：

操作	RMSE	运行时间	轨迹长度	是否碰撞	得分
调整前	~0.042	~36.209	~29.976	0	~21.728
调整后	~0.029	~30.169	~29.626	0	~17.932

## 5.4 重规划抖动问题修复

**问题：**在轨迹重规划触发的瞬间，无人机机体出现明显的抖动与不连续现象。经排查，原始代码在触发重规划时，直接将当前时刻（now）设为新轨迹的起始点。然而，路径计算本身需要消耗时间，导致生成的轨迹在发布时，其定义的“起始时刻”已经成为过去。这种计算耗时与执行时刻的时序错位，使得控制器接收到的首个设定点与当前状态偏差巨大，直接引发机体的剧烈抖动与轨迹断层。

**优化策略：**

- 引入预测性延迟机制：**摒弃“立即执行”的策略，改为“规划未来”。在开始计算轨迹前，预先设定一个**执行延迟**量  $\delta t$ ，并基于未来时刻  $t_{future} = t_{now} + \delta t$  的位置、速度、加速度作为新轨迹的起点。这确保了当计算完成并发布轨迹时，时间刚好到达  $\delta t$ ，控制器能够自然衔接。
- 自适应时间窗口：**针对不同阶段的计算负荷差异，动态调整延迟量  $\delta t$ ：
  - 冷启动阶段：**首次重规划涉及内存分配等初始化操作，耗时较长，将延迟设定为 **200ms**，确保计算充裕。
  - 热运行阶段：**后续重规划计算稳定，将延迟切换为 **30ms**，以保证对环境变化的快速响应。
- 时间戳对齐：**修正通信协议，强制将轨迹消息的时间戳设置为上述预设的未来起始时间，确保规划层、控制层与物理时间轴的严格同步。

**优化效果：**一定程度上缓解了重规划瞬间的抖动现象。虽然受限于系统实时性等物理因素，机体抖动未能被完全消除，但由计算时延引发的轨迹跟踪误差相比优化前显著降低。如图 7 所示，右图为优化后效果。

操作	RMSE	运行时间	轨迹长度	是否碰撞	得分
修复前	~0.029	~30.169	~29.626	0	~17.932
修复后	~0.006	~30.070	~30.008	0	~13.362

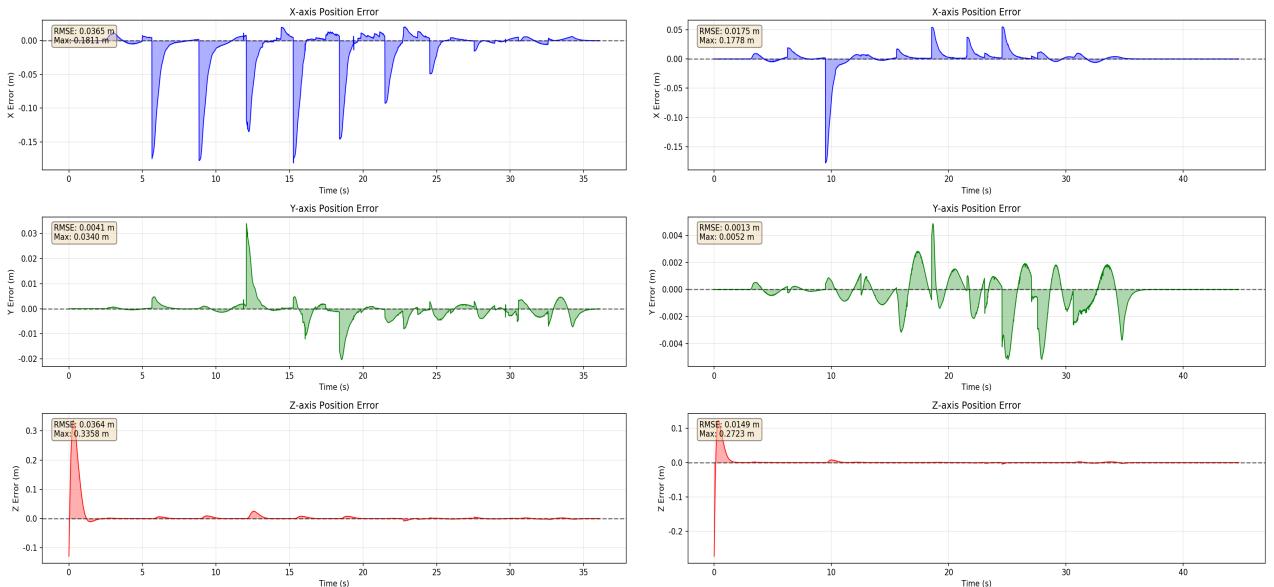


图 7. 优化前（左）与优化后（右）位置跟踪误差对比图

## 5.5 后端轨迹优化：基于曲率的时间分配策略

**策略：**在将几何路径转化为时间参数化的轨迹时，不再单纯依据距离分配时间。引入了基于转角曲率的时间放大因子，对于急转弯路径，动态增加2-4倍（参数可调整）的时间预算。这使得无人机在通过大角度弯道时能自动减速，从而生成符合动力学约束的圆润轨迹，避免了过弯时的超调与碰撞。

**优化效果：**优化后显著消除了绕障转弯时的“甩尾”现象，使轨迹更加平稳。

## 5.6 前端规划改进：Theta\* 算法

### 5.6.1 算法选择

Theta\* 是 A\* 的现代改良版算法，在 A\* 的基础上引入了视线检查机制。在扩展节点时，它允许父节点越过当前邻居节点直接连接到后继节点。这种“任意角度（Any-Angle）”的特性使得生成的路径接近欧几里得空间的最短距离，显著减少了路径转折点，为后端 Minimum Snap 轨迹生成提供了更优质的初值。

### 5.6.2 实现优化

为了解决实际飞行中的抖动、频繁重规划及轨迹不平滑问题，在标准 Theta\* 实现上进行了以下改进：

- 路径后处理：**实现多级路径简化策略。首先，引入“起点去噪”逻辑，强制忽略起点附近（参数可调整）因网格离散化产生的密集微小线段；其次，利用向量点积进行共线检查，剔除直线上冗余的中间点；最后，仅对长距离线段（参数可调整）进行稀疏插值。这确保输出路径仅保留关键几何拐点，大幅降低后端优化的计算量。
- 起步阶段的安全检查：**在轨迹安全性检查（SafeCheck）中增加了“起步豁免”机制。针对重规划初期因控制误差或膨胀半径导致起点可能处于障碍物边缘的情况，强制忽略轨迹前 0.5 秒（参数可调整）内的微小碰撞风险。这一改进彻底解决了因起点误报导致的“死循环重规划”问题，消除了飞行卡顿现象。

### 5.6.3 改进效果

算法	RMSE	运行时间	轨迹长度	是否碰撞	得分
A*	~0.006	~30.070	~30.008	0	~13.362
Theta*	~0.005	~20.329	~28.697	0	~10.953

为了直观评估改进后的几何路径质量，图 8 展示了 A\* 与 Theta\* 算法的路径规划对比。如图所示，改进后的 Theta\* 算法生成的轨迹路径明显更加笔直。

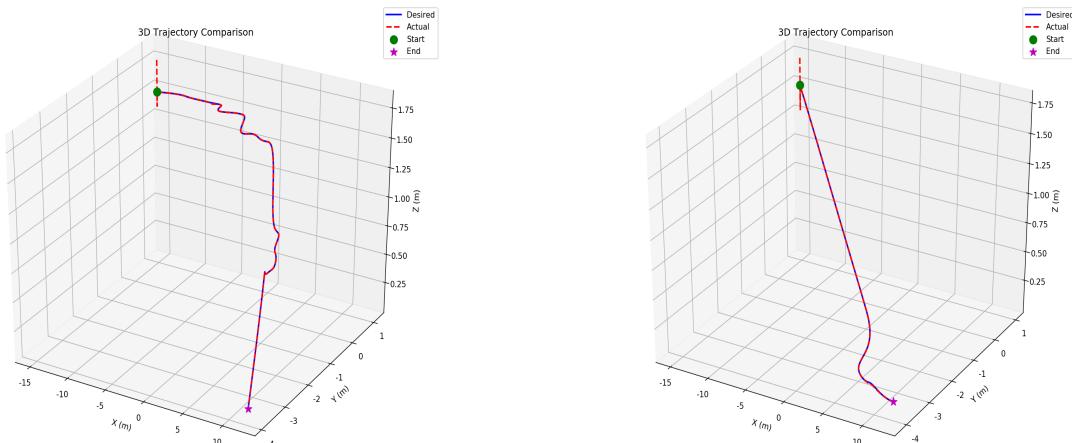


图 7. A\*（左）与 Theta\*（右）算法的路径规划对比