移动机器人规划与控制期末文档作业报告

王相轩　　23354154

2025 年 12 月 22 日

目录

1 题目1：坐标系转换

1.1 已知条件

- 无人机在世界坐标系 $\{\mathcal{W}\}$ 下的姿态（由 `tracking.csv` 提供）以四元数 ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ 表示。
- 末端执行器固连坐标系 $\{\mathcal{D}\}$ 相对于无人机机体系 $\{\mathcal{B}\}$ 的姿态变化由旋转矩阵 ${}^{\mathcal{B}}\mathbf{R}_{\mathcal{D}}$ 描述，其矩阵元由公式(1)给出：

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

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

1.2 目标

计算末端执行器在世界坐标系 $\{\mathcal{W}\}$ 下的姿态，同样以四元数 ${}^{\mathcal{W}}_{\mathcal{D}}\mathbf{q}$ 表示。

1.3 计算步骤

计算过程的核心是坐标系的链式变换。世界系到末端执行器系的旋转矩阵可以通过机体系进行传递。

1.3.1 计算世界系到机体系的旋转矩阵

首先，将给定的无人机本体四元数 ${}^{\mathcal{W}}_{\mathcal{B}}\mathbf{q}$ 转换为对应的旋转矩阵 ${}^{\mathcal{W}}\mathbf{R}_{\mathcal{B}}$。设四元数为 $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$，其对应的旋转矩阵为：

$$ {}^{\mathcal{W}}\mathbf{R}_{\mathcal{B}} = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y - q_wq_z) & 2(q_xq_z + q_wq_y) \\ 2(q_xq_y + q_wq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z - q_wq_x) \\ 2(q_xq_z - q_wq_y) & 2(q_yq_z + q_wq_x) & 1-2(q_x^2+q_y^2) \end{bmatrix} $$

对于 `tracking.csv` 中的每一个时间点 t_i，根据对应的四元数值计算得到 ${}^{\mathcal{W}}\mathbf{R}_{\mathcal{B}}(t_i)$。

在代码实现中，通常使用现成的库函数来完成这一转换以确保数值稳定性。

```python
# 使用 scipy.spatial.transform.Rotation
from scipy.spatial.transform import Rotation as R
# q_drone 为 [qx, qy, qz, qw] 顺序
rot_drone = R.from_quat([qx[i], qy[i], qz[i], qw[i]])
W_R_B = rot_drone.as_matrix()
```

1.3.2 计算机体系到末端执行器系的旋转矩阵

根据题目给出的公式(1)，直接代入参数 ω 和 α，以及当前时间 t_i，计算矩阵 ${}^{\mathcal{B}}\mathbf{R}_{\mathcal{D}}(t_i)$。

```python
def B_R_D(t_val):
    cw = np.cos(omega * t_val)
    sw = np.sin(omega * t_val)
    ca = np.cos(alpha)
```

```
    sa = np.sin(alpha)
    return np.array([
        [cw, -sw*ca, sw*sa],
        [sw, cw*ca, -cw*sa],
        [0, sa, ca]
    ])
```

1.3.3 计算世界系到末端执行器系的旋转矩阵

通过矩阵连乘，将上述两个旋转矩阵结合起来，得到世界坐标系 $\{\mathcal{W}\}$ 到末端执行器坐标系 $\{\mathcal{D}\}$ 的旋转矩阵：

$$^{\mathcal{W}}\mathbf{R}_{\mathcal{D}}(t_i) = {}^{\mathcal{W}}\mathbf{R}_{\mathcal{B}}(t_i) \cdot {}^{\mathcal{B}}\mathbf{R}_{\mathcal{D}}(t_i)$$

这里的乘法顺序是关键。因为 $^{\mathcal{B}}\mathbf{R}_{\mathcal{D}}$ 将向量从 $\{\mathcal{D}\}$ 系变换到 $\{\mathcal{B}\}$ 系，而 $^{\mathcal{W}}\mathbf{R}_{\mathcal{B}}$ 将向量从 $\{\mathcal{B}\}$ 系变换到 $\{\mathcal{W}\}$ 系。因此，连乘 $^{\mathcal{W}}\mathbf{R}_{\mathcal{B}} \cdot {}^{\mathcal{B}}\mathbf{R}_{\mathcal{D}}$ 实现了从 $\{\mathcal{D}\}$ 系到 $\{\mathcal{W}\}$ 系的变换，即构成了 $^{\mathcal{W}}\mathbf{R}_{\mathcal{D}}$。

```
# 矩阵乘法计算世界系到末端执行器系的旋转
W_R_D = W_R_B @ B_R_D(t_i)
```

1.3.4 将旋转矩阵转换为四元数

将步骤3中得到的 $^{\mathcal{W}}\mathbf{R}_{\mathcal{D}}(t_i)$ 转换回四元数形式 $^{\mathcal{W}}_{\mathcal{D}}\mathbf{q}(t_i)$。转换算法如下：设旋转矩阵为 $\mathbf{R} = [r_{ij}], i, j = 1, 2, 3$。首先计算四元数的标量部分 q_w 和向量部分 q_x, q_y, q_z：

$$\begin{aligned} q_w &= \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}} \\ q_x &= \frac{r_{32} - r_{23}}{4q_w} \\ q_y &= \frac{r_{13} - r_{31}}{4q_w} \\ q_z &= \frac{r_{21} - r_{12}}{4q_w} \end{aligned}$$

此公式在 q_w 不为零时稳定。在实际编程中，通常采用一种数值更稳定的方法，即比较矩阵的迹 $(r_{11} + r_{22} + r_{33})$ 和对角线元素，选择最大的那个来避免除以一个很小的数。使用库函数可以自动处理这些问题。

```
# 将旋转矩阵转换为四元数 返回( [x, y, z, w] 顺序)
q = R.from_matrix(W_R_D).as_quat()
# 调整为 [w, x, y, z] 顺序
q_final = [q[3], q[0], q[1], q[2]]
```

1.3.5 四元数后处理（保证连续性与规范性）

1. **归一化：** 对计算得到的四元数进行归一化处理，确保其模长为1：

$$\mathbf{q}_{\text{norm}} = \frac{\mathbf{q}}{\|\mathbf{q}\|} = \frac{[q_w, q_x, q_y, q_z]^T}{\sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}}$$

2. **保证 $q_w \geq 0$：** 检查归一化后的四元数。如果 $q_w < 0$，则将整个四元数取反：

$$\mathbf{q}_{\text{final}} = \begin{cases} \mathbf{q}_{\text{norm}}, & \text{if } q_w \geq 0 \\ -\mathbf{q}_{\text{norm}}, & \text{if } q_w < 0 \end{cases}$$

这一操作是为了保证四元数时间序列的连续性，因为 $\mathbf{q}$ 和 $-\mathbf{q}$ 代表同一个旋转，但符号翻转可能导致相邻时刻的四元数发生跳变。

```python
# 归一化
q_final = q_final / np.linalg.norm(q_final)
# 保证 q_w >= 0
if q_final[0] < 0:
    q_final = -q_final
```

1.4 总结

通过以上五个步骤，对于每个给定的时间点 t_i，都可以计算出其时末端执行器在世界坐标系下的、满足连续性要求的归一化四元数姿态 ${}^{\mathcal{W}}_{\mathcal{D}}\mathbf{q}(t_i)$。将所有时间点的计算结果串联，即可得到执行器在世界坐标系下的姿态变化曲线，同时进行四元数归一化验证，结果如下：

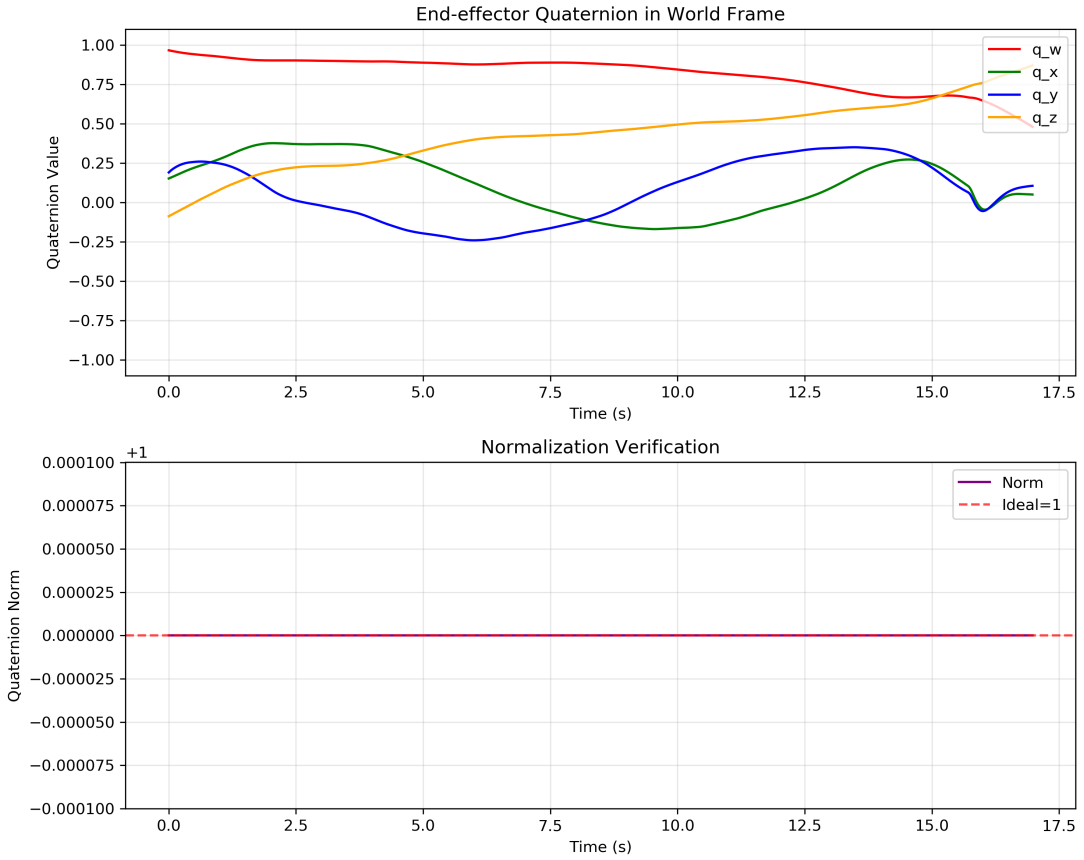

图 1: 变化曲线与归一化验证

2 题目2：开放题解答

2.1 A*轨迹规划的启发式与路径简化

2.1.1 启发式函数与tie_breaker的影响分析

现有tie_breaker的作用机制：

在A*算法中，启发式函数$h(n)$用于估计从节点n到目标节点的代价。tie_breaker通过轻微放大启发式值来影响开集节点的扩展顺序：

$$\begin{aligned} f_{\text{new}}(n) &= f(n) \times (1+\epsilon) \\ &= [g(n)+h(n)] \times (1+\epsilon) \end{aligned}$$

其中ϵ为很小的正数（如0.001）。

关于其对开集拓展顺序的影响：首先可以打破平局，当多个节点具有相同的f值时，tie_breaker使得距离起点更近（g值更小）的节点具有更小的调整后f值；其次可以减少”走之”路径：通过优先扩展靠近起点的节点，算法更倾向于生成直线路径，减少不必要的方向变化。

其次，其对路径平滑性的影响包括：减少路径中的锯齿状波动，提高路径的平滑度；但在获得更实用的飞行路径的同时可能牺牲理论上的最短路径最优性。

偏好平面飞行的修改方案：

为减少不必要的高度变化，可修改启发式函数：

```
double 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));

    // 增加高度变化的惩罚权重
    double height_penalty = 2.0;
    double h = dx + dy + height_penalty * dz;

    // tie_breaker
    double tie_breaker = 1.0 + 1.0/1000.0;
    return h * tie_breaker;
}
```

或修改边权计算：

```
// 在函数中AstarGetSucc
edgeCostSets.push_back(sqrt(dx*dx + dy*dy + height_penalty*dz*dz));
```

对可行性和最优性的影响分析：可行性方面，修改后仍能保证找到可行路径，因为障碍物检测逻辑不变；最优性方面，牺牲几何最短路径最优性，但获得更符合无人机动力学特性的实用路径；计算效率方面，可能增加扩展节点数量，但改善了路径质量。

2.1.2 路径简化参数的影响分析

path_resolution过大的影响：跟踪误差方面，由下面公式，路径点过少，相邻点间距过大，无人机需要大幅调整姿态：

$$\text{最大跟踪误差} \approx \frac{\text{路径段长度}^2}{8 \times R_{\min}}$$

其中$R_{\min}$为无人机最小转弯半径。

其次在安全性方面，简化后的路径可能过于靠近障碍物，缺乏安全裕度。

path_resolution过小的影响：路径点过多，增加控制器计算复杂度；包含过多细节，导致频繁的姿态调整，且增加了路径数据存储空间。

2.2 SO(3)位置控制器的力姿态生成

2.2.1 控制器各项作用分析

阅读代码可知，控制器力计算公式为：

$$\begin{aligned}\mathbf{F} = &\mathbf{K}_x \circ (\mathbf{p}_d - \mathbf{p}) + \mathbf{K}_v \circ (\mathbf{v}_d - \mathbf{v}) \\ &+ m\mathbf{a}_d + m\mathbf{K}_a \circ (\mathbf{a}_d - \mathbf{a}) + mg\mathbf{e}_z\end{aligned}$$

其各项物理意义为：

项	作用描述
$\mathbf{K}_x \circ (\mathbf{p}_d - \mathbf{p})$	位置误差比例控制，提供向目标收敛的基本力
$\mathbf{K}_v \circ (\mathbf{v}_d - \mathbf{v})$	速度误差微分控制，抑制超调，提高稳定性
$m\mathbf{a}_d$	期望加速度前馈，提前补偿动态变化
$m\mathbf{K}_a \circ (\mathbf{a}_d - \mathbf{a})$	加速度误差补偿，改善跟踪性能
$mg\mathbf{e}_z$	重力补偿，维持高度稳定

由上述公式，在大误差或传感器噪声情况下对ka项截断或限幅的必要性包括：

- 大误差情况下，当$|\mathbf{p}_d - \mathbf{p}| > 3$时，截断ka可以防止控制力饱和：

$$\mathbf{K}_a = [\min(|e_x|, 3) \times 0.2, \min(|e_y|, 3) \times 0.2, \min(|e_z|, 3) \times 0.2]$$

- 考虑传感器噪声：加速度传感器噪声η_a会被放大，因此需要对Ka做限幅：

$$\Delta F_{\text{noise}} = m\mathbf{K}_a \circ \eta_a$$

2.2.2 重载机体参数调整策略

参数调整方案：

```
// 质量更新
setMass(new_mass);

// 增益调整（基于质量比例）
double mass_ratio = new_mass / original_mass;
Eigen::Vector3d new_kx = kx * sqrt(mass_ratio);
Eigen::Vector3d new_kv = kv * sqrt(mass_ratio);
```

调整原理分析：根据牛顿第二定律$F = ma$，推力与质量成正比，当机体更重时，为了保持相似的加速度跟踪，采用平方根缩放保持相似的阻尼比：

$$\zeta = \frac{k_v}{2\sqrt{mk_x}} \Rightarrow \text{保持}\zeta\text{不变需 } k_v \propto \sqrt{m}$$

调整后的预期影响分析如下：

参数	预期影响
推力大小	线性增加：$F_{\text{new}} = F_{\text{orig}} \times \frac{m_{\text{new}}}{m_{\text{orig}}}$
姿态角度	相同机动下变化更平缓（惯性增大）
系统稳定性	适当增益可维持稳定性，但需注意振荡风险
响应速度	略有降低，需重新调整控制参数

2.3 动力学建模与约束

2.3.1 建模简化对控制分配的影响分析

在四旋翼无人机仿真器中，采用”刚体+重力+推力”的简化动力学模型：

$$\text{仿真模型：} m\ddot{\mathbf{p}} = \mathbf{F} - mg\mathbf{e}_z \tag{1}$$

$$\text{实际物理：} m\ddot{\mathbf{p}} = \mathbf{F} - mg\mathbf{e}_z - \mathbf{D}\dot{\mathbf{p}} + \Delta\mathbf{F}_{\text{unmodeled}} \tag{2}$$

①质量：

首先，质量参数m直接影响控制分配中的推力计算：

根据推力分配机制：

$$F_{total} = \|\mathbf{F}\| = m\|\ddot{\mathbf{p}}_d + \mathbf{K}_p(\mathbf{p}_d - \mathbf{p}) + \mathbf{K}_v(\dot{\mathbf{p}}_d - \dot{\mathbf{p}})\| \tag{3}$$

$$f_i = \mathbf{M}^{-1}(i,:) \cdot [F_{total}, \boldsymbol{\tau}_x, \boldsymbol{\tau}_y, \boldsymbol{\tau}_z]^T \tag{4}$$

因此，质量误差的传播效应包括：

- 推力大小偏差：$\Delta F = |m - \hat{m}| \cdot (\|\ddot{\mathbf{p}}_d\| + g)$

- 重力补偿误差：导致高度方向的稳态误差 $\mathbf{e}_{ss,z} = \frac{(m-\hat{m})g}{k_{p,z}}$
- 能量分配不均：某些电机过载而其他电机利用率不足

②惯量：

其次，惯量矩阵$\mathbf{I}$决定姿态控制的力矩分配：

期望力矩计算公式：

$$\boldsymbol{\tau}_d = \hat{\mathbf{I}}\dot{\boldsymbol{\omega}}_d + \boldsymbol{\omega} \times \hat{\mathbf{I}}\boldsymbol{\omega} + \mathbf{K}_R\mathbf{e}_R + \mathbf{K}_\omega\mathbf{e}_\omega \tag{5}$$

惯量误差影响计算：

$$\Delta\boldsymbol{\tau} = (\mathbf{I} - \hat{\mathbf{I}})\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I} - \hat{\mathbf{I}})\boldsymbol{\omega} \tag{6}$$

综上，惯量误差的影响包括：

- 交叉耦合：俯仰和横滚运动间产生非期望的耦合力矩
- 动态响应失真：系统表现出欠阻尼（振荡）或过阻尼（响应迟缓）
- 控制分配矩阵敏感性：惯量误差通过分配矩阵$\mathbf{M}$放大到各电机

③气动阻尼：

气动阻尼忽略时，忽略阻尼项$\mathbf{D}\dot{\mathbf{p}}$导致速度相关的力未补偿：

$$\mathbf{F}_{damp} = -\mathbf{D}\dot{\mathbf{p}} = -\begin{bmatrix} d_x & 0 & 0 \\ 0 & d_y & 0 \\ 0 & 0 & d_z \end{bmatrix}\dot{\mathbf{p}} \tag{7}$$

具体表现包括：匀速飞行需持续推力，但控制器未包含此项补偿；加速后由于缺乏阻尼，实际速度超过期望值而出现速度过冲；刹车距离加长，减速时需要额外时间克服惯性等。

④估计有误出现的可观测现象：

现象	产生原因
稳态位置偏差	质量估计错误，重力补偿不准确
速度过冲	阻尼忽略或速度增益过小
持续振荡	惯量估计错误导致欠阻尼
响应延迟	模型过于简化，未考虑系统惯性
能量消耗异常	气动阻力被忽略，实际功耗更高

2.3.2 气动阻力补偿策略

控制层补偿方案：

```
// 添加线性阻力模型
Eigen::Vector3d drag_force = -kv_drag * vel_;
force_.noalias() = ... + drag_force; // 添加到总力中

// 参数在线辨识
void onlineDragIdentification() {
    // 在匀速飞行阶段采集数据
    // F = mg + Dv 水平匀速时()
    // 使用最小二乘法拟合矩阵D
}
```

规划层调整方案：

- 速度约束：$v_{\max} = \sqrt{\frac{F_{\max} - mg}{k_v}}$
- 加速度曲线优化，考虑阻力影响

两种方案比较：

特性	控制层调整	规划层调整
实时性	实时补偿，自适应性强	离线计算，无法适应变化
精度	可精确补偿动态效应	近似补偿，相对保守
计算复杂度	需要在线计算，复杂度较高	计算简单，预处理完成
适用场景	动态环境，变化条件	已知固定环境
实现难度	较高，需传感器支持	较低，路径规划阶段完成

策略选择与理由： 基于无人机在实际应用中的动态环境特性和飞行安全要求，我选择控制层补偿方案。虽然该方案实现复杂度较高，但其实时自适应能力能够有效应对飞行过程中的气流变化、载重变动等不确定因素。规划层调整虽然实现简单，但其离线特性无法适应实际飞行中的动态环境变化，在安全性方面存在较大局限。控制层补偿通过在线参数辨识和实时调整，能够确保无人机在各种复杂环境下的精确跟踪性能和飞行安全，这一优势在工程实践中具有决定性意义。

参数调整流程：

步骤1：基准测试（无阻力补偿）

步骤2：匀速飞行数据采集

步骤3：最小二乘参数辨识

步骤4：验证与微调

步骤5：在线自适应更新