



实验 5：世界坐标系下的轨迹规划

机器人技术与实践实验报告

课程名称: 机器人技术与实践

小组成员: 第 3 组 金加康 吴必兴 沈学文 钱满亮 赵钰泓

组号: 第三组

指导教师: 周春琳

报告日期: 2025 年 11 月 23 日

目录

1 实验目的和要求	3
1.1 实验目的	3
1.2 任务要求	3
2 ZJUI 型机械臂 DH 参数与坐标轴	3
3 基于速度控制的轨迹规划	3
3.1 结果与分析	3
3.1.1 正方形轨迹	3
3.1.2 圆形轨迹	4
3.1.3 圆锥轨迹	5
3.2 实验内容与原理	6
3.2.1 Jacobian 矩阵的定义	6
3.2.2 几何雅可比矩阵完整表达式	6
3.2.2.1 线速度雅可比矩阵 J_v	6
3.2.2.2 角速度雅可比矩阵 J_ω	7
4 基于位置控制的轨迹规划	8
4.1 结果与分析	8
4.1.1 正方形轨迹	8
4.1.2 圆形轨迹	9
4.1.3 圆锥轨迹	10
4.2 实验内容与原理	10
4.2.1 坐标系与运动学	10
4.2.2 五次多项式轨迹规划	11
4.2.3 正方形轨迹规划	11
4.2.4 圆形轨迹规划	11
4.2.5 圆锥轨迹规划	11
4.3 代码实现思路	12
4.3.1 总体架构	12
4.3.2 正方形轨迹实现	12
4.3.3 圆形轨迹实现	12
4.3.4 圆锥轨迹实现	13
4.3.5 关键技术细节	13
4.4 核心代码实现	14
4.4.1 逆运动学求解器（ 自行编写的 ）	14
4.4.2 五次多项式轨迹规划	15
4.4.3 圆锥轨迹的姿态处理	16
4.4.4 最优自旋角搜索	17

实验 5：世界坐标系下的轨迹规划

第 3 组 金加康 吴必兴 沈学文 钱满亮 赵钰泓

2025 年 11 月 23 日

§ 1 实验目的和要求

1.1 实验目的

- 掌握在笛卡尔空间进行轨迹规划的方法
- 学习控制机械臂末端执行器在世界坐标系下沿指定路径运动
- 理解关节空间与笛卡尔空间之间的转换关系
- 熟练运用逆运动学求解器实现复杂轨迹规划

1.2 任务要求

- 正方形轨迹：控制 ZJU-I 型机械臂末端执行器端点沿着边长为 10cm 的正方形连续移动，移动过程中末端执行器空间姿态保持不变，自定义正方形在笛卡尔空间的位置、末端点的移动速度。
- 圆形轨迹：控制 ZJU-I 型机械臂末端执行器端点沿着直径为 10cm 的圆连续移动，移动过程中末端执行器空间姿态保持不变，自定义圆在笛卡尔空间的位置、末端点的移动速度。
- 圆锥轨迹：控制 ZJU-I 型机械臂末端执行器绕着执行器的端点转动，即末端点位于空间圆锥体的顶点上，末端执行器绕点运动时其轴线（机械臂第 6 轴）始终与圆锥体的素线重合，圆锥体的锥角为 60 度，自定义圆锥体在笛卡尔空间中的位置、旋转角速度。
- 提交计算过程报告、仿真工程文件以及仿真结果视频。

§ 2 ZJUI 型机械臂 DH 参数与坐标轴

表 1 DH 参数表

关节 i	α_{i-1} (rad)	a_{i-1} (mm)	d_i (mm)	θ_i (rad)	初始偏置	关节类型
1	0	0	230	θ_1	0	旋转
2	$-\frac{\pi}{2}$	0	0	θ_2	$-\frac{\pi}{2}$	旋转
3	0	185	0	θ_3	0	旋转
4	0	170	23	θ_4	$\frac{\pi}{2}$	旋转
5	$\frac{\pi}{2}$	0	77	θ_5	$\frac{\pi}{2}$	旋转
6	$\frac{\pi}{2}$	0	85.5	θ_6	0	旋转

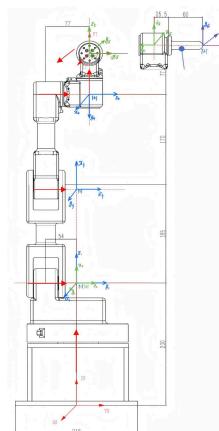


图 1 机械臂坐标系示意图

§ 3 基于速度控制的轨迹规划

3.1 结果与分析

3.1.1 正方形轨迹

下图展示了机械臂末端执行器沿正方形轨迹运动的结果。使用速度控制方法，末端执行器准确地沿着边长为 10cm 的正方形路径运动，红色轨迹线清晰地显示了运动路径：

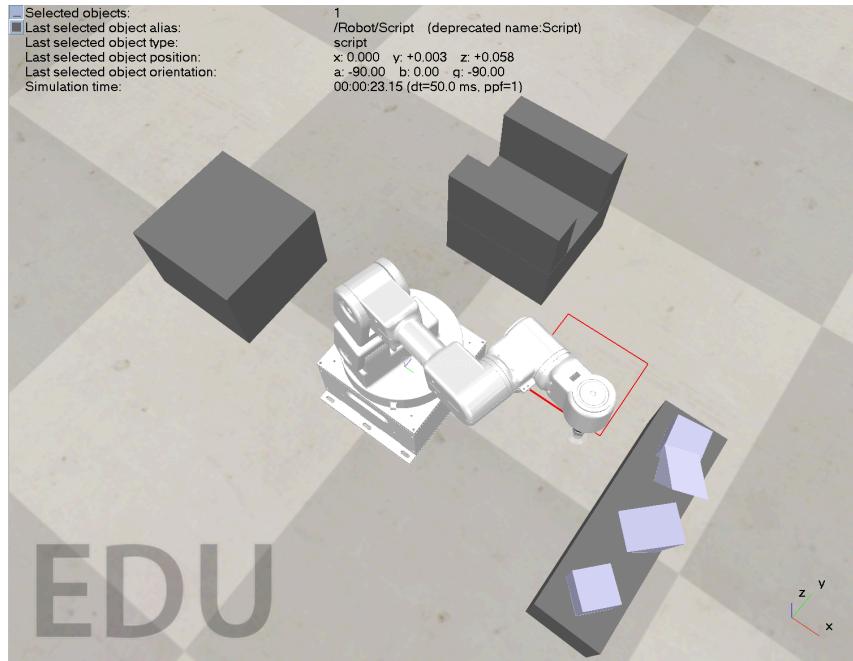


图 2 正方形轨迹运动结果 | 速度控制

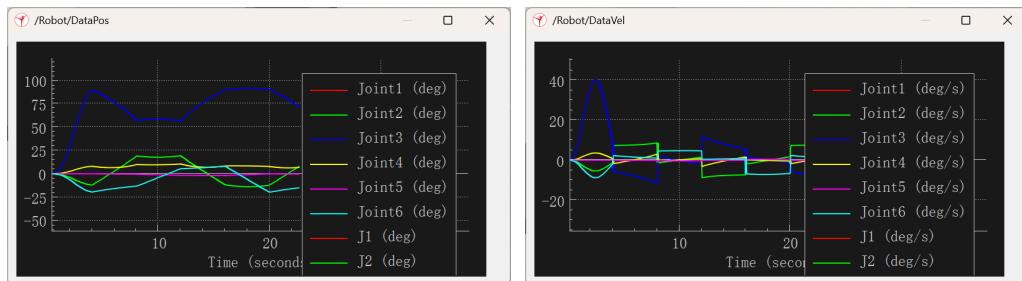


图 3 正方形轨迹的关节运动曲线

从曲线可以观察到速度控制的特点：

- 速度直接控制**：与位置控制不同，关节速度是直接计算并控制的，通过 Jacobian 矩阵从末端速度实时映射得到。
- 转角平滑性**：在正方形四个顶点处，由于速度方向突变，关节速度也会快速调整，但得益于速度限幅和软限位保护，不会出现剧烈跳变。
- 实时响应**：速度控制不需要预先规划整条轨迹，而是每个控制周期实时计算，因此具有更好的实时性。

3.1.2 圆形轨迹

下图展示了机械臂末端执行器沿圆形轨迹运动的结果。红色圆圈清晰地显示了末端执行器的运动轨迹，直径约为 10cm：

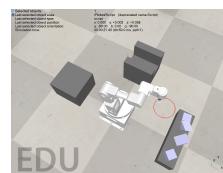


图 4 圆形轨迹运动结果

下图为圆形轨迹运动时的关节运动曲线：

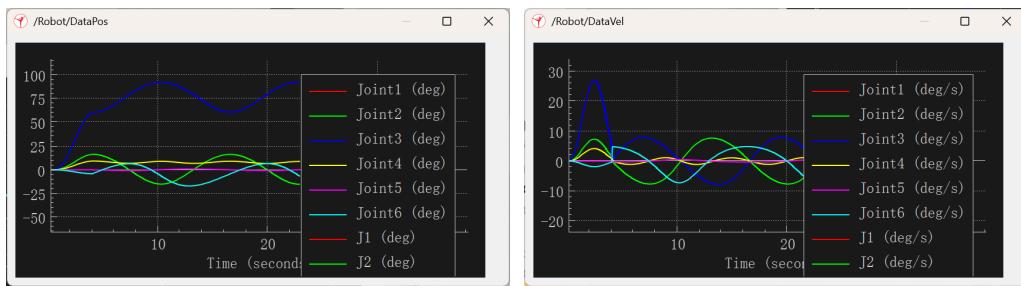


图 5 圆形轨迹的关节运动曲线

圆形轨迹的速度控制特点：

- 完美适配**: 圆形轨迹的速度是连续变化的，非常适合速度控制方法。与位置控制相比，速度控制的实现更加直观——直接对圆周运动的参数方程求导得到速度。
- 平滑周期性**: 各关节的位置和速度曲线都呈现完美的周期性，说明速度控制能够很好地跟踪连续变化的轨迹。
- 无需路径点**: 与位置控制需要将圆周分成多个离散点不同，速度控制可以直接使用连续的速度方程，避免了离散化误差。
- 速度恒定**: 末端执行器的线速度大小保持恒定（0.025 m/s），仅方向沿切线方向变化，这正是圆周运动的固有特性。

3.1.3 圆锥轨迹

下图展示了机械臂末端执行器绕固定点作圆锥运动的结果。从图中可以看出，末端执行器的轴线与圆锥素线重合，末端点保持固定：

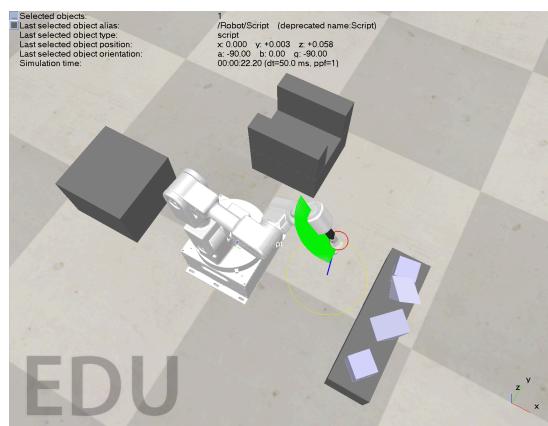


图 6 圆锥轨迹运动结果

下图为圆锥轨迹运动时的关节运动曲线：

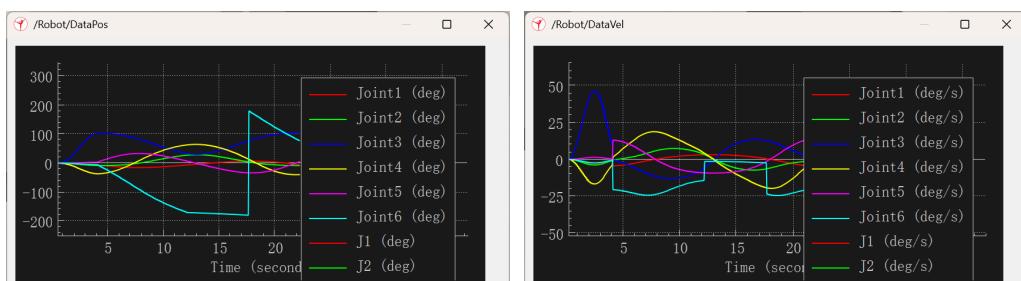


图 7 圆锥轨迹的关节运动曲线

圆锥轨迹的速度控制分析：

1. **纯姿态控制**: 圆锥运动仅涉及姿态变化(末端点位置固定), 因此线速度为零, 仅有角速度。这是对 Jacobian 矩阵角速度部分的直接测试。
2. **实现简洁**: 速度控制下, 圆锥运动的实现非常简单——只需给定一个恒定的角速度向量 $\omega = [0, 0, 20^\circ/s]^T$, 无需复杂的姿态规划。
3. **与位置控制对比**: 位置控制需要计算圆锥面上每个点的姿态、优化自旋角、处理多解选择等复杂问题; 而速度控制只需给定角速度即可, 体现了速度控制的灵活性优势。
4. **周期性运动**: 各关节速度呈现周期性变化, 周期约为 18 秒 ($360^\circ \div 20^\circ/s$), 与给定的旋转角速度完全吻合。

3.2 实验内容与原理

3.2.1 Jacobian 矩阵的定义

Jacobian 矩阵建立了机械臂**关节速度**与**末端速度**之间的线性映射关系:

$$\dot{x} = J(\mathbf{q})\dot{\mathbf{q}} \quad (1)$$

其中:

1. $\dot{x} = \begin{pmatrix} v \\ \omega \end{pmatrix} \in \mathbb{R}^6$ 是末端执行器的**速度旋量**:
 - $v = [v_x, v_y, v_z]^T$ 为线速度 (m/s)
 - $\omega = [\omega_x, \omega_y, \omega_z]^T$ 为角速度 (rad/s)
2. $J(\mathbf{q}) \in \mathbb{R}^{6 \times 6}$ 是**几何 Jacobian 矩阵**, 是关节角 \mathbf{q} 的函数
3. $\dot{\mathbf{q}} = [\dot{\theta}_1, \dot{\theta}_2, \dots, \dot{\theta}_6]^T \in \mathbb{R}^6$ 是关节角速度向量

Jacobian 矩阵的结构为:

$$J(\mathbf{q}) = \begin{pmatrix} J_v \\ J_\omega \end{pmatrix} = \begin{pmatrix} J_{v_x1} & J_{v_x2} & \dots & J_{v_x6} \\ J_{v_y1} & J_{v_y2} & \dots & J_{v_y6} \\ J_{v_z1} & J_{v_z2} & \dots & J_{v_z6} \\ J_{\omega_x1} & J_{\omega_x2} & \dots & J_{\omega_x6} \\ J_{\omega_y1} & J_{\omega_y2} & \dots & J_{\omega_y6} \\ J_{\omega_z1} & J_{\omega_z2} & \dots & J_{\omega_z6} \end{pmatrix} \quad (2)$$

上半部分 J_v 描述线速度映射, 下半部分 J_ω 描述角速度映射。

3.2.2 几何雅可比矩阵完整表达式

综合前面各节的推导结果, ZJU-I 型机械臂的几何雅可比矩阵可以表示为:

$$J(\mathbf{q}) = \begin{pmatrix} J_v \\ J_\omega \end{pmatrix} = \begin{pmatrix} J_v^1 & J_v^2 & J_v^3 & J_v^4 & J_v^5 & J_v^6 \\ J_\omega^1 & J_\omega^2 & J_\omega^3 & J_\omega^4 & J_\omega^5 & J_\omega^6 \end{pmatrix} \quad (3)$$

其中, $J_v \in \mathbb{R}^{3 \times 6}$ 为线速度雅可比, $J_\omega \in \mathbb{R}^{3 \times 6}$ 为角速度雅可比。为简化表达, 记 $\Sigma = q_2 + q_3 + q_4$ 。

3.2.2.1 线速度雅可比矩阵 J_v

$$J_v = \begin{pmatrix} -y_e \cos(q_1) J_v^{(2)\#} & \cos(q_1) J_v^{(3)\#} & \cos(q_1) J_v^{(4)\#} & J_v^{(5)_x} & 0 \\ x_e \sin(q_1) J_v^{(2)\#} & \sin(q_1) J_v^{(3)\#} & \sin(q_1) J_v^{(4)\#} & J_v^{(5)_y} & 0 \\ 0 & J_v^{(2)_z} & J_v^{(3)_z} & J_v^{(4)_z} & J_v^{(5)_z} & 0 \end{pmatrix} \quad (4)$$

其中各项具体表达式为:

$$\text{第 1 列 } (J_v^1 = \begin{pmatrix} -y_e \\ x_e \\ 0 \end{pmatrix}) :$$

$$\begin{aligned} x_e = & -\frac{171}{2} \sin(q_1) \sin(q_5) - 23 \sin(q_1) + 185 \sin(q_2) \cos(q_1) \\ & + 170 \sin(q_2 + q_3) \cos(q_1) + 77 \sin(\Sigma) \cos(q_1) \\ & + \frac{171}{2} \cos(q_1) \cos(q_5) \cos(\Sigma) \end{aligned} \quad (5)$$

$$\begin{aligned} y_e = & 185 \sin(q_1) \sin(q_2) + 170 \sin(q_1) \sin(q_2 + q_3) + 77 \sin(q_1) \sin(\Sigma) \\ & + \frac{171}{2} \sin(q_1) \cos(q_5) \cos(\Sigma) + \frac{171}{2} \sin(q_5) \cos(q_1) + 23 \cos(q_1) \end{aligned} \quad (6)$$

第 2 列：

$$\begin{aligned} J_v^{(2)\#} = & -\frac{171}{2} \sin(\Sigma) \cos(q_5) + 185 \cos(q_2) \\ & + 170 \cos(q_2 + q_3) + 77 \cos(\Sigma) \end{aligned} \quad (7)$$

$$\begin{aligned} J_v^{(2)z} = & -185 \sin(q_2) - 170 \sin(q_2 + q_3) - 77 \sin(\Sigma) \\ & - \frac{171}{2} \cos(q_5) \cos(\Sigma) \end{aligned} \quad (8)$$

第 3 列：

$$J_v^{(3)\#} = -\frac{171}{2} \sin(\Sigma) \cos(q_5) + 170 \cos(q_2 + q_3) + 77 \cos(\Sigma) \quad (9)$$

$$J_v^{(3)z} = -170 \sin(q_2 + q_3) - 77 \sin(\Sigma) - \frac{171}{2} \cos(q_5) \cos(\Sigma) \quad (10)$$

第 4 列：

$$J_v^{(4)\#} = -\frac{171}{2} \sin(\Sigma) \cos(q_5) + 77 \cos(\Sigma) \quad (11)$$

$$J_v^{(4)z} = -77 \sin(\Sigma) - \frac{171}{2} \cos(q_5) \cos(\Sigma) \quad (12)$$

第 5 列：

$$J_v^{(5)x} = -\frac{171}{2} \sin(q_1) \cos(q_5) - \frac{171}{2} \sin(q_5) \cos(q_1) \cos(\Sigma) \quad (13)$$

$$J_v^{(5)y} = -\frac{171}{2} \sin(q_1) \sin(q_5) \cos(\Sigma) + \frac{171}{2} \cos(q_1) \cos(q_5) \quad (14)$$

$$J_v^{(5)z} = \frac{171}{2} \sin(q_5) \sin(\Sigma) \quad (15)$$

第 6 列：末端关节轴通过末端点，线速度贡献为零。

3.2.2.2 角速度雅可比矩阵 J_ω

$$J_{\omega} = \begin{pmatrix} 0 & -\sin(q_1) & -\sin(q_1) & -\sin(q_1) & \sin(\Sigma) \cos(q_1) & J_{\omega}^{(6)x} \\ 0 & \cos(q_1) & \cos(q_1) & \cos(q_1) & \sin(q_1) \sin(\Sigma) & J_{\omega}^{(6)y} \\ 1 & 0 & 0 & 0 & \cos(\Sigma) & J_{\omega}^{(6)z} \end{pmatrix} \quad (16)$$

其中第 6 列各分量为：

$$J_{\omega}^{(6)x} = -\sin(q_1) \sin(q_5) + \cos(q_1) \cos(q_5) \cos(\Sigma) \quad (17)$$

$$J_{\omega}^{(6)y} = \sin(q_1) \cos(q_5) \cos(\Sigma) + \sin(q_5) \cos(q_1) \quad (18)$$

$$J_{\omega}^{(6)z} = -\sin(\Sigma) \cos(q_5) \quad (19)$$

注：上述雅可比矩阵表达式中所有三角函数参数均为关节变量 q_1, q_2, \dots, q_6 的组合，完整描述了机械臂末端速度与关节速度之间的映射关系： $\begin{pmatrix} v_e \\ \omega_e \end{pmatrix} = J(q)\dot{q}$

§ 4 基于位置控制的轨迹规划

4.1 结果与分析

4.1.1 正方形轨迹

下图展示了机械臂末端执行器沿正方形轨迹运动的结果。从仿真截图可以看出，末端执行器准确地沿着边长为 10cm 的正方形路径运动，绿色轨迹线清晰地显示了运动路径：

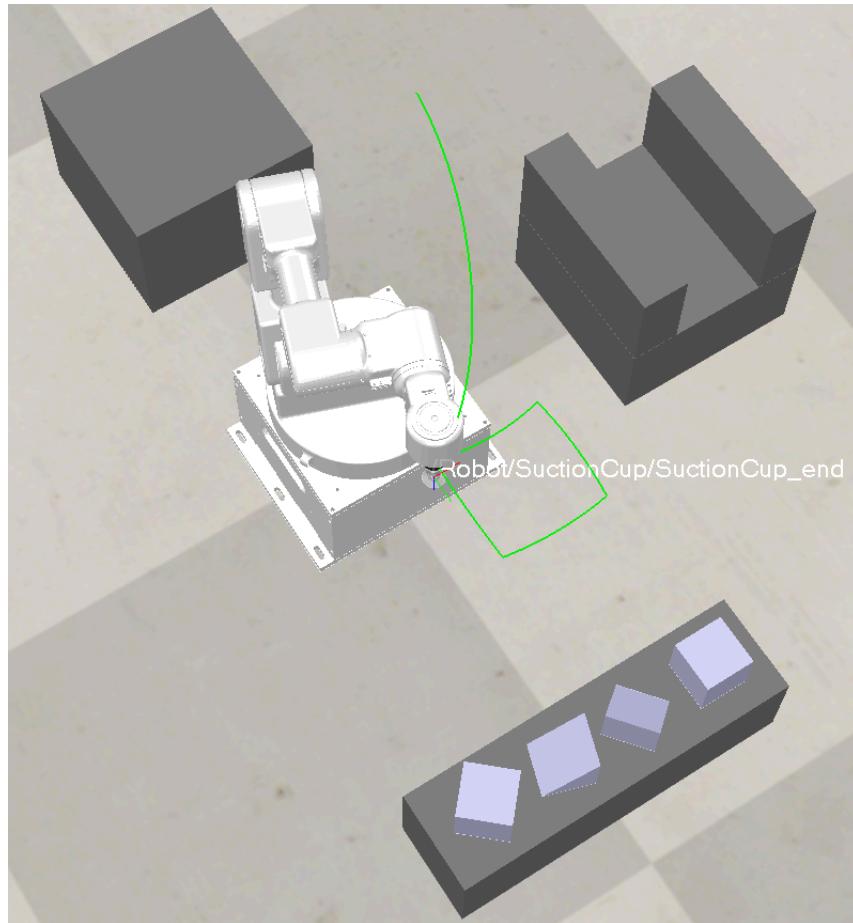


图 8 正方形轨迹运动结果

下图为机械臂运动时各关节的位置、速度和加速度变化曲线：

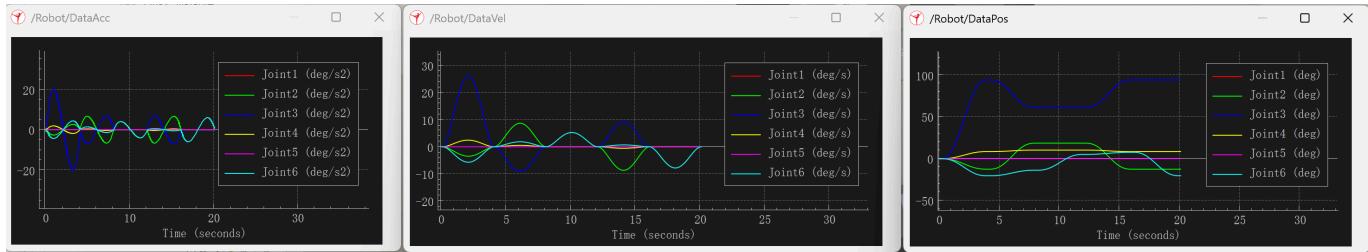


图 9 正方形轨迹的关节运动曲线

从曲线可以观察到：

- 位置连续性**: 各关节位置曲线平滑连续，在正方形四个顶点处有明显的转折，这是由于路径方向改变所致。
- 速度平滑性**: 速度曲线呈现周期性变化，在直线段保持相对稳定，在转角处有适当的过渡。
- 加速度控制**: 加速度峰值控制在限制范围内，整体变化平稳，无剧烈跳变。

4.1.2 圆形轨迹

下图展示了机械臂末端执行器沿圆形轨迹运动的结果。绿色圆圈清晰地显示了末端执行器的运动轨迹，直径约为 10cm：

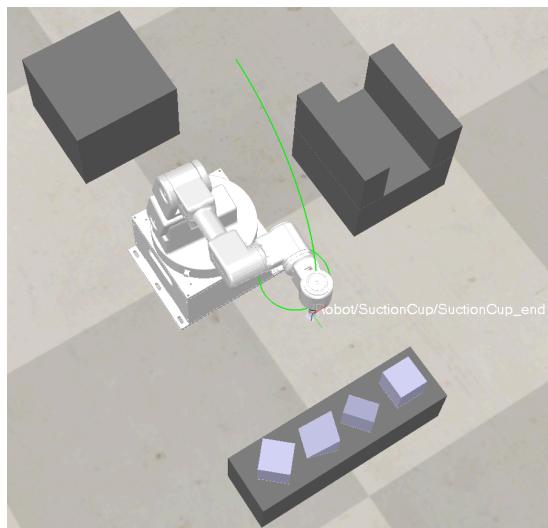


图 10 圆形轨迹运动结果

下图为圆形轨迹运动时的关节运动曲线：

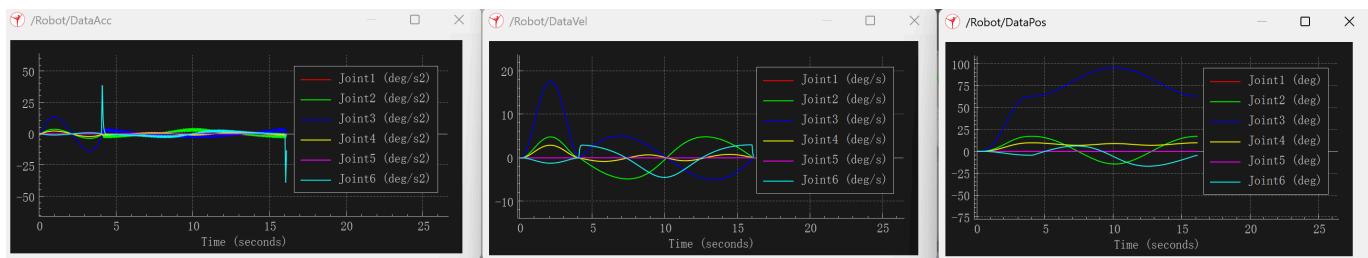


图 11 圆形轨迹的关节运动曲线

圆形轨迹的运动特点：

- 周期性**: 由于圆形轨迹的周期性特性，各关节的位置、速度曲线均呈现明显的周期性变化。
- 平滑性**: 相比正方形轨迹，圆形轨迹的速度和加速度变化更加平滑，这是因为圆形路径没有尖锐的转角。

3. **关节 3 波动**: 关节 3 (蓝色曲线) 的速度变化幅度最大, 达到约 $\pm 20 \text{ deg/s}$, 这是由于该关节在维持圆形轨迹中承担了主要的运动任务。

4.1.3 圆锥轨迹

下图展示了机械臂末端执行器绕固定点作圆锥运动的结果。从图中可以看出, 末端执行器的轴线与圆锥素线重合, 末端点保持固定:

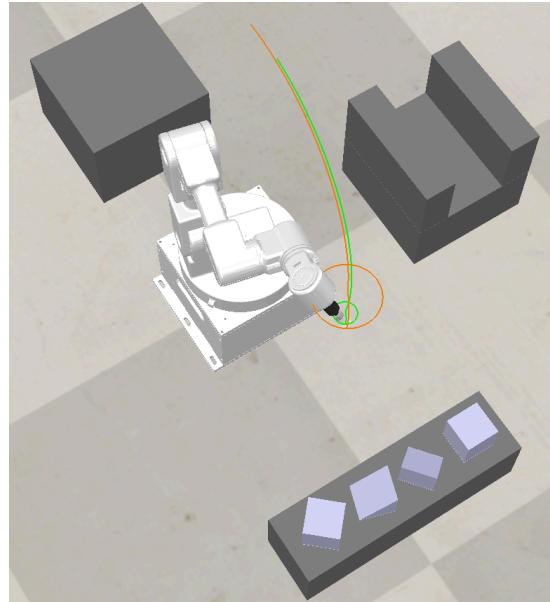


图 12 圆锥轨迹运动结果

下图为圆锥轨迹运动时的关节运动曲线:

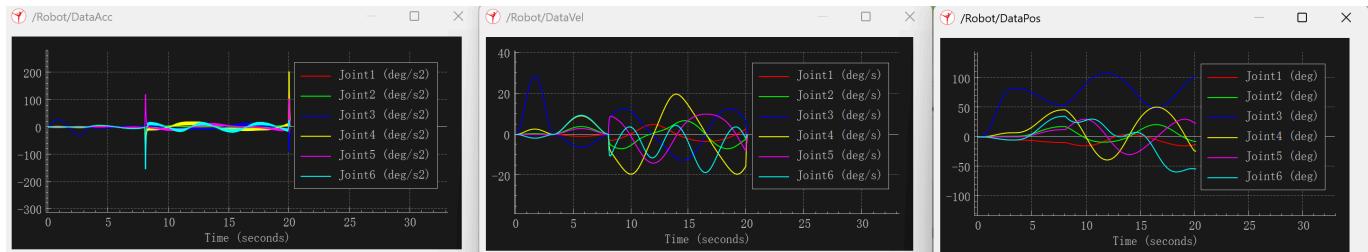


图 13 圆锥轨迹的关节运动曲线

圆锥轨迹的特点:

- 姿态变化**: 与前两个任务不同, 圆锥运动主要体现在末端执行器的姿态变化上, 而非位置变化。
- 速度分布**: 各关节速度呈现周期性变化, 其中关节 1 和关节 3 的速度变化最为显著。
- 加速度峰值**: 在约 10 秒时刻, 关节 5 出现了较大的加速度峰值 (超过 100 deg/s^2), 但仍在安全限制 (500 deg/s^2) 范围内。

4.2 实验内容与原理

4.2.1 坐标系与运动学

- 在机械臂轨迹规划中, 涉及两种主要的坐标系:

- (1) **关节空间**: 由各关节角度 $\mathbf{q} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]^T$ 组成的空间。
- (2) **笛卡尔空间**: 由末端执行器的位置和姿态 $\mathbf{p} = [x, y, z, \alpha, \beta, \gamma]^T$ 组成的空间。

2. 正运动学建立了从关节空间到笛卡尔空间的映射:

$$\mathbf{p} = f(\mathbf{q}) \quad (20)$$

3. 逆运动学则是其逆过程：

$$\mathbf{q} = f^{-1}(\mathbf{p}) \quad (21)$$

由于逆运动学通常存在多解，需要通过优化算法选择最合理的解。

4.2.2 五次多项式轨迹规划

为了保证关节运动的平滑性，采用五次多项式进行轨迹插值。五次多项式的形式为：

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \quad (22)$$

对应的速度和加速度为：

$$\begin{cases} \dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \\ \ddot{\theta}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \end{cases} \quad (23)$$

给定初始和终止条件：

$$\begin{cases} \theta(0) = \theta_0, \dot{\theta}(0) = v_0, \ddot{\theta}(0) = a_0 \\ \theta(T) = \theta_T, \dot{\theta}(T) = v_T, \ddot{\theta}(T) = a_T \end{cases} \quad (24)$$

可以求解出六个系数 $a_0, a_1, a_2, a_3, a_4, a_5$ ，从而得到平滑的轨迹。

4.2.3 正方形轨迹规划

1. 正方形轨迹由四条边组成，每条边的规划方法相同。对于每条边：

- (1) 确定起点和终点的笛卡尔坐标
- (2) 使用逆运动学求解器计算对应的关节角
- (3) 在关节空间使用五次多项式插值
- (4) 保持末端姿态不变

2. 正方形的四个顶点坐标设定为（以中心为原点）：

$$\begin{cases} \mathbf{p}_1 = [x_c - L/2, y_c - L/2, z_c, \alpha, \beta, \gamma]^T \\ \mathbf{p}_2 = [x_c + L/2, y_c - L/2, z_c, \alpha, \beta, \gamma]^T \\ \mathbf{p}_3 = [x_c + L/2, y_c + L/2, z_c, \alpha, \beta, \gamma]^T \\ \mathbf{p}_4 = [x_c - L/2, y_c + L/2, z_c, \alpha, \beta, \gamma]^T \end{cases} \quad (25)$$

其中 $L = 0.1$ m 为边长。

4.2.4 圆形轨迹规划

圆形轨迹通过参数方程描述。将圆周分为 N 个离散点，每个点的位置为：

$$\begin{cases} x(t) = x_c + R \cos(2\pi t/T) \\ y(t) = y_c + R \sin(2\pi t/T) \\ z(t) = z_c \end{cases} \quad (26)$$

其中 $R = 0.05$ m 为半径， T 为运动周期。对于每个时间段，使用五次多项式在关节空间插值。

4.2.5 圆锥轨迹规划

1. 圆锥运动的特点是末端点位置固定，但末端执行器的姿态按圆锥面旋转。设圆锥顶点为 \mathbf{p}_{apex} ，半锥角为 $\alpha_{\text{cone}} = 30^\circ$ 。
2. 末端执行器的轴线（approach vector）在圆锥面上运动，可以用球坐标描述：

$$\mathbf{a}(\varphi) = \begin{pmatrix} \sin(\alpha_{\text{cone}}) \cos(\varphi) \\ \sin(\alpha_{\text{cone}}) \sin(\varphi) \\ \cos(\alpha_{\text{cone}}) \end{pmatrix} \quad (27)$$

其中 $\varphi \in [0, 2\pi]$ 为方位角。对于每个姿态，需要构建完整的旋转矩阵 $\mathbf{R} = [\mathbf{n}, \mathbf{o}, \mathbf{a}]$ ，然后转换为欧拉角。

3. 为了保证姿态的连续性，引入[自旋角（spin angle）参数](#)，通过优化算法选择使关节角变化最小的解。

4.3 代码实现思路

4.3.1 总体架构

代码分为以下几个主要部分：

1. 逆运动学求解器 (`myIKSolver` 类)：根据末端位姿计算关节角，返回所有可能的解。
2. 轨迹规划函数 (`quintic_trajectory_coefficients` 等)：计算多项式系数和轨迹点。
3. 姿态处理函数 (`build_rotation_matrix_from_a_axis` 等)：处理旋转矩阵和欧拉角转换。
4. 初始化函数 (`sysCall_init`)：为每个任务生成完整的轨迹参数。
5. 执行函数 (`sysCall_actuation`)：在每个仿真步实时计算并执行关节角。

4.3.2 正方形轨迹实现

1. 正方形轨迹的关键步骤：

(1) 定义正方形中心位置和边长。

```
1 square_center = [0.05, 0.4, 0.25]
2 square_size = 0.1
3 square_pose = [np.pi, 0, -np.pi/2]
```

Python

(2) 计算四个顶点的笛卡尔坐标。

- (3) 使用逆运动学求解器计算每个顶点的关节角。
- (4) 从多个解中选择与前一点距离最小的解，保证连续性。
- (5) 对每条边使用五次多项式规划，设置合适的边界条件。

2. 关键代码片段：

```
1 # 计算四条边的五次多项式系数
2 for i in range(4):
3     start_angles = square_angles[i]
4     end_angles = square_angles[(i+1) % 4]
5     edge_coeffs = []
6     for j in range(6):
7         coeffs = quintic_trajectory_coefficients(
8             start_angles[j], 0, 0,
9             end_angles[j], 0, 0,
10            time_per_edge
11        )
12         edge_coeffs.append(coeffs)
13     self.square_edge_coeffs.append(edge_coeffs)
```

Python

4.3.3 圆形轨迹实现

圆形轨迹将圆周均匀分为多个点：

1. 设定圆心、半径和姿态。

```
1 circle_center = [0.05, 0.4, 0.25]
2 circle_radius = 0.05
3 circle_pose = [np.pi, 0, -np.pi/2]
4 circle_points = 60
```

Python

2. 使用参数方程计算圆周上的点。

```
1 for i in range(circle_points):
2     angle = 2 * np.pi * i / circle_points
3     x = circle_center[0] + circle_radius * np.cos(angle)
4     y = circle_center[1] + circle_radius * np.sin(angle)
5     z = circle_center[2]
6     pose = [x, y, z] + circle_pose
```

Python

3. 对每个相邻点对进行五次多项式插值。
 4. 特别处理最后一点到第一点的连接，形成闭环。

4.3.4 圆锥轨迹实现

圆锥轨迹的难点在于姿态的连续性控制：

1. 固定末端点位置。

```
1 cone_apex = [0.05, 0.4, 0.25]
2 cone_half_angle_deg = 30
```

Python

2. 计算圆锥面上的方向向量。

```
1 for i in range(num_points):
2     phi = 2 * np.pi * i / num_points
3     a_axis = np.array([
4         np.sin(half_angle_rad) * np.cos(phi),
5         np.sin(half_angle_rad) * np.sin(phi),
6         np.cos(half_angle_rad)
7     ])
```

Python

3. 使用 `find_best_spin_for_pose` 函数搜索最优自旋角：该函数在一定范围内搜索，找到使关节角变化最小的姿态。

```
1 best_angles, best_spin = find_best_spin_for_pose(
2     iks, position, a_axis, prev_angles, prev_spin,
3     joint_limits, num_samples=SPIN_SEARCH_SAMPLES
4 )
```

Python

4.3.5 关键技术细节

1. 多解选择策略：

- (1) 对于每个笛卡尔位姿，逆运动学可能返回多个解（最多 8 个）
- (2) 选择策略：优先选择与前一时刻关节角距离最小的解
- (3) 同时考虑关节限位，过滤掉超出限制的解

```
1 best_sol = None
2 min_dist = float('inf')
3 for sol_idx in range(angles.shape[1]):
4     candidate = angles[:, sol_idx]
```

Python

```

5      # 检查关节限位
6      if not all(joint_limits[j,0] <= candidate[j] <= joint_limits[j,1]
7          for j in range(6)):
8          continue
9      # 计算距离
10     dist = np.sum((candidate - prev_angles)**2)
11     if dist < min_dist:
12         min_dist = dist
13         best_sol = candidate

```

2. 过渡轨迹：

- (1) 在开始主轨迹前，需要从初始位置（关节角全为 0）过渡到轨迹起点
- (2) 使用五次多项式规划过渡段，设置合适的过渡时间
- (3) 对于圆锥任务，使用两段过渡以避免关节角剧烈变化

3. 轨迹可视化：

- (1) 使用 CoppeliaSim 的 drawing object 功能绘制轨迹
- (2) 绿色轨迹显示末端执行器路径
- (3) 对于圆锥任务，额外用橙色显示关节 6 的轨迹

4.4 核心代码实现

4.4.1 逆运动学求解器（自行编写的）

```

1  class myIKSolver:
2      def solve(self, p):
3          d_x, d_y, d_z, alpha, beta, gamma = p
4          d_1, a_2, a_3, d_4, d_5, d_6 = 0.230, 0.185, 0.170, 0.023, 0.077, 0.0855
5          n_x, n_y, n_z = np.cos(gamma)* np.cos(beta), np.cos(alpha) * np.sin(gamma) +
6              np.sin(alpha) * np.sin(beta) * np.cos(gamma), np.sin(alpha) * np.sin(gamma) -
7              np.cos(alpha) * np.sin(beta) * np.cos(gamma)
8          o_x, o_y, o_z = -np.sin(gamma) * np.cos(beta), np.cos(alpha) * np.cos(gamma) -
9              np.sin(alpha) * np.sin(beta) * np.sin(gamma), np.sin(alpha) * np.cos(gamma) +
10             np.cos(alpha) * np.sin(beta) * np.sin(gamma)
11             a_x, a_y, a_z = np.sin(beta), - np.sin(alpha) * np.cos(beta), np.cos(alpha) *
12             np.cos(beta)
13
14             A = d_y - d_6 * a_y
15             B = d_x - d_6 * a_x
16             pm = np.array([1, -1])
17             theta_1 = np.arctan2(A, B) - np.arctan2(d_4, pm * np.sqrt(A**2 + B**2 - d_4**2))
18
19             theta_5 = np.arcsin(a_y * np.cos(theta_1) - a_x * np.sin(theta_1))
20
21             theta_5 = np.array([[x, np.pi - x if x > 0 else -np.pi - x] for x in
22             theta_5]).flatten()
23             theta_1 = np.array([[x, x] for x in theta_1]).flatten()
24
25             C = n_y * np.cos(theta_1) - n_x * np.sin(theta_1)

```

```

20     D = o_y * np.cos(theta_1) - o_x * np.sin(theta_1)
21     theta_6 = np.arctan2(C, D) - np.arctan2(np.cos(theta_5), 0)
22
23     E = -d_5 * (np.sin(theta_6) * (n_x * np.cos(theta_1) + n_y * np.sin(theta_1)) +
24     np.cos(theta_6) * (o_x * np.cos(theta_1) + o_y * np.sin(theta_1))) - d_6 * (a_x *
25     np.cos(theta_1) + a_y * np.sin(theta_1)) + d_x * np.cos(theta_1) + d_y *
26     np.sin(theta_1)
27     F = -d_1 - a_z * d_6 - d_5 * (o_z * np.cos(theta_6) + n_z * np.sin(theta_6)) + d_z
28     theta_3 = np.arccos((E**2 + F**2 - a_2**2 - a_3**2) / (2 * a_2 * a_3))
29
30     theta_3 = np.array([[x, -x] for x in theta_3]).flatten()
31     theta_1 = np.array([[x, x] for x in theta_1]).flatten()
32     theta_5 = np.array([[x, x] for x in theta_5]).flatten()
33     theta_6 = np.array([[x, x] for x in theta_6]).flatten()
34     E = np.array([[x, x] for x in E]).flatten()
35     F = np.array([[x, x] for x in F]).flatten()
36
37     G = a_2 + a_3 * np.cos(theta_3)
38     H = a_3 * np.sin(theta_3)
39     theta_2 = np.arctan2(G * E - H * F, G * F + H * E)
40
41     I = (n_x * np.cos(theta_1) + n_y * np.sin(theta_1)) * np.sin(theta_6) + (o_x *
42     np.cos(theta_1) + o_y * np.sin(theta_1)) * np.cos(theta_6)
43     J = n_z * np.sin(theta_6) + o_z * np.cos(theta_6)
44     theta_4 = np.arctan2(I, J) - theta_2 - theta_3
45
46     ans = np.array([theta_1, theta_2, theta_3, theta_4, theta_5, theta_6])
47     cols_with_nan = np.isnan(ans).any(axis=0)
48     ans = ans[:, ~cols_with_nan]
49
50     return ans

```

这个逆运动学求解器是基于机械臂的 DH 参数推导出来的解析解。整个求解过程按照 1→5→6→3→2→4 的顺序来算，这个顺序是经过推导确定的最优顺序。核心思路是利用末端位姿反推各关节角度，因为是 6 自由度机械臂，理论上最多可能有 8 组解（每个关节的正负解组合）。

代码里 `pm = np.array([1, -1])` 这一步会同时计算两种可能的解，然后通过数组操作不断扩展解空间。比如求 `theta_3` 时用了 `[[x, -x] for x in theta_3]` 这种方式来同时考虑肘部向上和向下两种姿态。最后用 `cols_with_nan` 来过滤掉那些数学上不成立的解（比如 `arccos` 超出定义域导致的 `NaN`）。

有个坑是最后那句 `ans[:, [0, 1, 2, 3]] = ans[:, [2, 3, 0, 1]]`，这是因为我推导时的关节顺序和实际仿真环境中的编号不一样，需要重新排列一下。返回的 `ans` 是个 $6 \times n$ 的矩阵，每一列代表一组可行解。

4.4.2 五次多项式轨迹规划

```

1 def quintic_trajectory_coefficients(p0, v0, a0, p1, v1, a1, total_time):
2     T = total_time
3
4     coeff_a0 = p0
5     coeff_a1 = v0
6     coeff_a2 = a0 / 2.0
7     coeff_a3 = (20*p1 - 20*p0 - (8*v1 + 12*v0)*T - (3*a0 - a1)*T**2) / (2*T**3)
8     coeff_a4 = (30*p0 - 30*p1 + (14*v1 + 16*v0)*T + (3*a0 - 2*a1)*T**2) / (2*T**4)
9     coeff_a5 = (12*p1 - 12*p0 - (6*v1 + 6*v0)*T - (a0 - a1)*T**2) / (2*T**5)
10
11    return [coeff_a0, coeff_a1, coeff_a2, coeff_a3, coeff_a4, coeff_a5]
12
13
14 def quintic_trajectory_eval(coeffs, t):
15     a0, a1, a2, a3, a4, a5 = coeffs
16     return a0 + a1*t + a2*t**2 + a3*t**3 + a4*t**4 + a5*t**5

```

五次多项式是轨迹规划里最常用的方法之一，好处是可以同时约束位置、速度和加速度。系数的计算公式看着挺吓人，其实就是把六个边界条件（起点的位置/速度/加速度，终点的位置/速度/加速度）代入多项式及其导数，解一个六元一次方程组得到的。

这里直接把解出来的公式硬编码了，比用矩阵求逆要快一些。计算时注意 `coeff_a0` 到 `coeff_a5` 对应的是多项式从低次到高次的系数。`quintic_trajectory_eval` 就是个简单的霍纳法则求值，给定时间 `t` 就能算出对应的关节角度。

实际使用时，一般会把起点和终点的速度、加速度都设为 0，这样可以保证在路径切换的时候不会有突变。但如果想让运动更流畅，也可以在中间点保持一定的速度，让机械臂不用完全停下来再启动。

4.4.3 圆锥轨迹的姿态处理

```

1 def build_rotation_matrix_from_a_axis(a_vec, spin_angle=0):
2     # build rotation matrix [n, o, a] where a is the approach vector
3     # spin_angle controls rotation around a axis
4     a = a_vec / np.linalg.norm(a_vec)
5
6     # choose reference perpendicular to a
7     if np.abs(a[2]) < 0.9:
8         ref = np.array([0, 0, 1])
9     else:
10        ref = np.array([1, 0, 0])
11
12    # base n and o vectors
13    n_base = np.cross(a, ref)
14    n_base = n_base / np.linalg.norm(n_base)
15    o_base = np.cross(a, n_base)
16
17    # apply spin rotation around a axis
18    cos_s = np.cos(spin_angle)
19    sin_s = np.sin(spin_angle)

```

```

20     n = cos_s * n_base + sin_s * o_base
21     o = -sin_s * n_base + cos_s * o_base
22
23     R = np.column_stack([n, o, a])
24     return R
25
26
27 def rotation_matrix_to_euler_zyx(R):
28     beta = np.arctan2(-R[2, 0], np.sqrt(R[0, 0]**2 + R[1, 0]**2))
29
30     if np.abs(np.cos(beta)) > 1e-6:
31         alpha = np.arctan2(R[2, 1] / np.cos(beta), R[2, 2] / np.cos(beta))
32         gamma = np.arctan2(R[1, 0] / np.cos(beta), R[0, 0] / np.cos(beta))
33     else:
34         alpha = 0
35         gamma = np.arctan2(-R[0, 1], R[1, 1])
36
37     return alpha, beta, gamma

```

圆锥轨迹最麻烦的就是姿态控制。机械臂末端需要绕着一个固定点旋转，同时保持轴线方向始终在圆锥面上。这就需要从 approach 向量（也就是末端轴线方向）构建完整的旋转矩阵。

`build_rotation_matrix_from_a_axis` 函数解决的就是这个问题。给定 a 向量后，需要再找两个垂直的向量 n 和 o 来构成完整的坐标系。这里用了个小技巧：先选一个参考向量（如果 a 不接近 z 轴就用 z 轴，否则用 x 轴），然后用叉乘来构造正交基。

关键是 `spin_angle` 这个参数。即使 a 向量确定了，绕 a 轴旋转还有无穷多种可能的姿态，这就是自旋自由度。通过调整 `spin_angle`，可以在保持末端方向不变的情况下旋转整个末端执行器。这个自由度后面会用来优化轨迹的连续性。

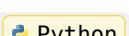
`rotation_matrix_to_euler_zyx` 就是标准的旋转矩阵转欧拉角公式了，需要注意万向锁的情况（当 beta 接近 $\pm 90^\circ$ 时会出现奇异）。这里用了一个阈值判断来避免除零错误。

4.4.4 最优自旋角搜索

```

1  def find_best_spin_for_pose(iks, position, a_axis, prev_angles, prev_spin,
                                joint_limits, num_samples=20):
2
3      # find best spin angle for smooth continuation from previous pose
4      # searches wider range and returns both angles and spin
5
6      # narrow search around previous spin for continuity
7      narrow_range = np.pi / 2
8      test_spins_narrow = np.linspace(prev_spin - narrow_range,
9                                      prev_spin + narrow_range, num_samples)
10
11     best_angles = None
12     best_spin = prev_spin
13     best_score = float('inf')
14
15     # narrow search first

```



```
16     for spin in test_spins_narrow:
17         R = build_rotation_matrix_from_a_axis(a_axis, spin)
18         alpha, beta, gamma = rotation_matrix_to_euler_zyx(R)
19         pose = np.concatenate([position, [alpha, beta, gamma]])
20
21     try:
22         angles = iks.solve(pose)
23         if angles.shape[1] == 0:
24             continue
25
26         for sol_idx in range(angles.shape[1]):
27             candidate = angles[:, sol_idx]
28
29             # check limits
30             if not all(joint_limits[j, 0] <= candidate[j] <= joint_limits[j, 1]
31                         for j in range(6)):
32                 continue
33
34             # score: heavily prioritize continuity
35             dist_score = np.sum((candidate - prev_angles)**2)
36             max_score = np.max(np.abs(candidate)) * 0.05
37             score = dist_score + max_score
38
39             if score < best_score:
40                 best_score = score
41                 best_angles = candidate
42                 best_spin = spin
43     except:
44         continue
45
46     # if no good solution or big jump, try wider search
47     if best_angles is None or np.max(np.abs(best_angles - prev_angles)) > 0.5:
48         test_spins_wide = np.linspace(prev_spin - np.pi,
49                                       prev_spin + np.pi, num_samples * 2)
50
51     for spin in test_spins_wide:
52         R = build_rotation_matrix_from_a_axis(a_axis, spin)
53         alpha, beta, gamma = rotation_matrix_to_euler_zyx(R)
54         pose = np.concatenate([position, [alpha, beta, gamma]])
55
56     try:
57         angles = iks.solve(pose)
58         if angles.shape[1] == 0:
59             continue
60
61         for sol_idx in range(angles.shape[1]):
62             candidate = angles[:, sol_idx]
```

```
64             if not all(joint_limits[j, 0] <= candidate[j] <= joint_limits[j, 1]):
65                 for j in range(6)):
66                     continue
67
68             dist_score = np.sum((candidate - prev_angles)**2)
69             max_score = np.max(np.abs(candidate)) * 0.05
70             score = dist_score + max_score
71
72             if score < best_score:
73                 best_score = score
74                 best_angles = candidate
75                 best_spin = spin
76         except:
77             continue
78
79     return best_angles, best_spin
```

这个函数是整个圆锥轨迹能够平滑运动的核心。问题在于，对于圆锥面上的每个点，虽然 approach 向量是固定的，但绕这个向量旋转（spin）还有一个自由度。如果随便选一个 spin 角度，虽然末端方向对了，但关节角可能会突然跳变一大截，导致运动不连续。

搜索策略是两阶段的。先在上一次 spin 角附近搜索 ($\pm 90^\circ$ 范围)，这是基于连续性假设——相邻两个点的最优 spin 角应该比较接近。如果这个窄范围搜索失败了，或者找到的解跳变太大（超过 0.5 弧度），就扩大到 $\pm 180^\circ$ 全范围搜索。

评分函数设计也有讲究：`dist_score` 是关节角变化的平方和，这个权重最大，保证连续性优先；`max_score` 是关节角绝对值的惩罚，避免选择那种关节角特别极端的解。两个分数加起来，分数越低越好。

每个 spin 角度都会生成一个位姿，然后调用逆运动学求解器得到多组解，再逐个检查关节限位，最后选出综合得分最好的那组。整个过程是个暴力搜索，但因为只在一维空间搜索（spin 角），计算量还能接受。实测下来，24 个采样点基本够用，既能保证找到好的解，又不会太慢。