机器人技术与实践 · 太空机械臂大作业

徐屹寒　周子为　余凯翔　许泽琛　朱益辉

2025 年 12 月 15 日

1　正运动学解

1.1　DH 参数表

根据机械臂结构图推导，DH 参数配置如下 (以 A 为基座)：

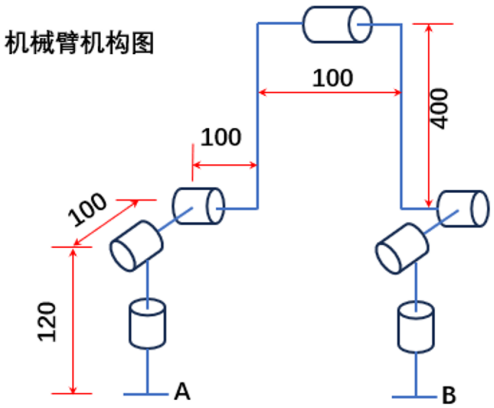

图 1: 机械臂结构图

表 1: DH 参数表

Link (i)	α_i	a_i	d_i	θ_i
1	$-90°$	0	120	$\theta_1(0°)$
2	$90°$	0	100	$\theta_2(90°)$
3	$0°$	400	150	$\theta_3(180°)$
4	$0°$	400	150	$\theta_4(180°)$
5	$90°$	0	0	θ_5
6	$90°$	0	100	$\theta_6(90°)$
7	$0°$	0	120	θ_7

1.2 正运动学解析解

正运动学解析解代码如下：

Listing 1: 正运动学代码

```python
import numpy as np
c = np.cos
s = np.sin
P = np.pi

def Tran(alpha, a, d, theta):
    # Transformation matrix calculation
    return np.array([
        [c(theta), -s(theta)*c(alpha), s(theta)*s(alpha),
           a*c(theta)],
        [s(theta), c(theta)*c(alpha), -c(theta)*s(alpha),
           a*s(theta)],
        [0, s(alpha), c(alpha), d],
        [0, 0, 0, 1]
    ])

def forward(q, base):
    if base == "R":
        t1 = q[0]
        t2 = q[1]
        t3 = q[2]
        t4 = -q[3]
        t5 = -q[4]
        t6 = -q[5]
        t7 = -q[6]

        T_01 = Tran(-P/2, 0, 0.12, t1 + P)
        T_12 = Tran(P/2, 0, 0.1, t2 + P/2)
        T_23 = Tran(0, 0.4, 0.15, t3 + P)
        T_34 = Tran(0, 0.4, 0.15, t4 + P)
        T_45 = Tran(P/2, 0, 0, t5)
        T_56 = Tran(P/2, 0, 0.1, t6 + P/2)
        T_67 = Tran(0, 0, 0.12, t7)

    elif base == "L":
        # Left base logic (omitted for brevity, similar to R)
```

```
        t1 = q[6]
        t2 = q[5]
        t3 = q[4]
        t4 = -q[3]
        t5 = -q[2]
        t6 = -q[1]
        t7 = -q[0]

        T_01 = Tran(-P/2, 0, 0.12, t1 + P)
        T_12 = Tran(P/2, 0, 0.1, t2 + P/2)
        T_23 = Tran(0, 0.4, 0.15, t3 + P)
        T_34 = Tran(0, 0.4, 0.15, t4 + P)
        T_45 = Tran(P/2, 0, 0, t5)
        T_56 = Tran(P/2, 0, 0.1, t6 + P/2)
        T_67 = Tran(0, 0, 0.12, t7)
        pass

    return T_01 @ T_12 @ T_23 @ T_34 @ T_45 @ T_56 @ T_67
def printT(T):
    print(T)
    print(getcoords(T))
    print(getdegrees(T))
q=np.zeros(7)
printT(forward(q, "R"))
```

在这一段代码中，注意到我们在一些部分取了角度的相反数，这是因为仿真文件里规定的旋转方向和我们先前制定的 DH 参数表规定的旋转方向相反；而我们同时设立了 L_based 和 R_based，这是机械臂在旋转时第一步为以初始右臂为基准、第二步时为初始左臂为基准。

将全 0 角度代入验证：

```
[[ 1.00000000e+00  3.21507022e-32 -1.22464680e-16  3.00000000e-01]
 [-1.12481984e-32 -1.00000000e+00 -2.44929360e-16 -9.87804622e-17]
 [-1.22464680e-16  2.44929360e-16 -1.00000000e+00  1.38777878e-17]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]
XYZ:
[ 3.00000000e-01 -9.87804622e-17  1.38777878e-17]
aby:
(np.float64(-180.0), np.float64(-7.0167092985348775e-15), np.float64(6.444742937248654e-31))
```

图 2: 全 0 角度正运动学结果

2 逆运动学解

2.1 求解 θ_1

令：

$$T_6 = {}^0T_1^{-1} \cdot T_{end} \cdot {}^6T_7^{-1} \tag{1}$$

根据上述矩阵方程，取 (1,4) 项相等。令：

$$\begin{cases} m_1 = d_7 a_x - p_x \\ n_1 = -d_7 a_y + p_y \\ C_1 = -d_3 - d_4 \end{cases} \tag{2}$$

得到方程：

$$(d_7 a_x - p_x)c_1 - (-d_7 a_y + p_y)s_1 = -d_3 - d_4 \tag{3}$$

解得：

$$\theta_1 = \operatorname{atan2}(m_1, n_1) - \operatorname{atan2}(C_1, \pm\sqrt{m_1^2 + n_1^2 - C_1^2}) \tag{4}$$

2.2 求解 θ_7

同样对上述矩阵方程，取 (1,2) 项相等得到：

$$(-n_x c_1 - n_y s_1)s_7 + (-o_x c_1 - o_y s_1)c_7 = 0 \tag{5}$$

可以解得：

$$\theta_7 = -\operatorname{atan2}(o_x c_1 + o_y s_1, n_x c_1 + n_y s_1) \tag{6}$$

2.3 求解 θ_6

取 (1,1) 和 (1,3) 项相等，解得：

$$\theta_6 = \operatorname{atan2}(a_x c_1 + a_y s_1, n_x c_1 c_7 + n_y s_1 c_7 - o_x s_7 c_1 - o_y s_1 s_7) \tag{7}$$

2.4 求解 θ_4

对于矩阵方程：

$${}^1T_4 = {}^0T_1^{-1} \cdot T_{end} \cdot {}^6T_7^{-1} \cdot {}^5T_6^{-1} \cdot {}^4T_5^{-1} \tag{8}$$

取 (2,4) 和 (3,4) 项相等得到：

$$\begin{cases} a_3 c_3 - a_4 c_{34} = -a_z d_7 - d_1 - d_6(n_z s_7 + o_z c_7) + p_z \\ -a_3 s_3 + a_4 s_{34} + d_2 = d_6[(n_x s_1 - n_y c_1)s_7 + (o_x s_1 - o_y c_1)c_7] + d_7(a_x s_1 - a_y c_1) - p_x s_1 + p_y c_1 \end{cases} \tag{9}$$

令方程右边分别为 m_4 和 n_4（需减去 d_2），平方和得到：

$$a_3^2 + a_4^2 - 2a_3a_4(c_3c_{34} + s_3s_{34}) = m_4^2 + n_4^2 \tag{10}$$

利用 $c_4 = c_3c_{34} + s_3s_{34}$，解得：

$$c_4 = \frac{m_4^2 + n_4^2 - a_3^2 - a_4^2}{2a_3a_4} \tag{11}$$

$$\theta_4 = \pm \arccos\left(\frac{m_4^2 + n_4^2 - a_3^2 - a_4^2}{2a_3a_4}\right) \tag{12}$$

2.5 求解 θ_3

展开 c_{34} 和 s_{34}：

$$\begin{cases} c_{34} = \cos(\theta_3 + \theta_4) = c_3c_4 - s_3s_4 \\ s_{34} = \sin(\theta_3 + \theta_4) = s_3c_4 + c_3s_4 \end{cases} \tag{13}$$

代入前述方程组整理得：

$$\begin{cases} (a_3 - a_4c_4)c_3 + a_4s_4s_3 = m_4 \\ (-a_3 + a_4c_4)s_3 + a_4s_4c_3 = n_4 \end{cases} \tag{14}$$

令 $m_3 = a_3 - a_4c_4, n_3 = a_4s_4$，解得：

$$\theta_3 = \text{atan2}(n_3m_4 - m_3n_4, m_3m_4 + n_3n_4) \tag{15}$$

2.6 求解 θ_5

借助 $\theta_3 + \theta_4 + \theta_5$ 得到。取 (2,2) 和 (3,2) 项相等，得到：

$$\begin{cases} s_{345} = -n_zs_7 - o_zc_7 \\ c_{345} = (n_xs_1 - n_yc_1)s_7 + (o_xs_1 - o_yc_1)c_7 \end{cases} \tag{16}$$

则：

$$\theta_3 + \theta_4 + \theta_5 = \text{atan2}(s_{345}, c_{345}) \tag{17}$$

进而求出 θ_5。

2.7 验证

对于上面正运动学角度全 0 的姿态进行反解，可以看到成功解出了全 0 角。

Listing 2: 逆运动学代码

```
CD3=np.array([0.3, 0, 0, P, 0, 0]) # x,y,z,roll,pitch,yaw
q1=solve(CD3, "R")
print("q1:")
print(q1)
T1=forward(q1, "R")
printT(T1)
print("/n")
```

```
(base) PS C:\Users\zyh\Desktop> python .\正逆运动学.py
q1:
[0. 0. 0. 0. 0. 0.]
XYZ:
[ 3.00000000e-01 -9.87804622e-17  1.38777878e-17]
aby:
(np.float64(-180.0), np.float64(-7.0167092985348775e-15), np.float64(6.444742937248654e-31))


q2:
[0. 0. 0. 0. 0. 0.]
XYZ:
[ 3.00000000e-01 -9.87804622e-17  1.38777878e-17]
aby:
(np.float64(-180.0), np.float64(-7.0167092985348775e-15), np.float64(6.444742937248654e-31))
```

图 3: 逆运动学结果

3 轨迹规划

3.1 五次多项式轨迹规划

3.1.1 方法介绍

五次多项式轨迹规划通过以下公式计算关节角度的轨迹:

$$q(t) = a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \tag{18}$$

其中 $q(t)$ 是关节角度在时间 t 的位置，系数 $(a_5, a_4, ..., a_0)$ 由起点、终点的位移、速度和加速度约束确定。

3.1.2 起终点五次多项式

我们首先实现了从起点到终点的五次多项式轨迹规划。计算轨迹系数需要以下约束:

- 起点位置 $q(0)$ 和终点位置 $q(T)$
- 起点速度 $\dot{q}(0)$ 和终点速度 $\dot{q}(T)$

- 起点加速度 $\ddot{q}(0)$ 和终点加速度 $\ddot{q}(T)$

代码实现如下:

Listing 3: 起终点五次多项式系数计算

```
def compute_quintic_curve_coefficients(start_pos, end_pos,
    start_vel, end_vel, duration):
    time_matrix = np.array([
        [0, 0, 0, 0, 0, 1],
        [duration**5, duration**4, duration**3, duration**2,
            duration, 1],
        [0, 0, 0, 0, 1, 0],
        [5*duration**4, 4*duration**3, 3*duration**2, 2*duration, 1,
            0],
        [0, 0, 0, 2, 0, 0],
        [20*duration**3, 12*duration**2, 6*duration, 2, 0, 0],
    ])
    # 时间矩阵的逆
    inv_time_matrix = np.linalg.inv(time_matrix)
    # 计算轨迹系数
    coeffs = np.dot(
        inv_time_matrix,
        np.array([start_pos, end_pos, start_vel, end_vel, 0, 0]).T
    )
    return coeffs
```

3.1.3 带中间速度约束的三点五次多项式轨迹

为了在三点之间规划轨迹，并保证中间点的速度连续性，我们设计了带中间速度约束的轨迹规划方法。其关键在于同时满足以下条件:

- 起点、终点的位移、速度、加速度约束。
- 中间点的位移和速度约束。

代码实现如下:

Listing 4: 带中间速度约束的轨迹规划

```
def compute_quintic_curve_with_mid_vel(start_pos, mid_pos, end_pos,
    mid_vel,time_start_mid, time_mid_end):
    time_matrix = np.array([
        [0, 0, 0, 0, 0, 1],
```

```
        [time_start_mid**5, time_start_mid**4, time_start_mid**3,
            time_start_mid**2, time_start_mid, 1],
        [time_mid_end**5, time_mid_end**4, time_mid_end**3,
            time_mid_end**2, time_mid_end, 1],
        [0, 0, 0, 0, 1, 0],
        [5*time_start_mid**4, 4*time_start_mid**3,
            3*time_start_mid**2, 2*time_start_mid, 1, 0],
        [5*time_mid_end**4, 4*time_mid_end**3, 3*time_mid_end**2,
            2*time_mid_end, 1, 0],
    ])
    # 时间矩阵的逆
    inv_time_matrix = np.linalg.inv(time_matrix)
    # 计算轨迹系数
    coeffs = np.dot(inv_time_matrix, np.array([start_pos, mid_pos,
        end_pos, 0, mid_vel, 0]).T)
    return coeffs
```

通过这个函数，我们可以在三点之间生成平滑的轨迹，并保证中间点的速度约束。

3.2 轨迹执行

在得到五次多项式的轨迹系数后，我们可以根据时间动态计算关节的位置。轨迹执行的核心是将时间代入五次多项式，计算关节的当前位置 $q(t)$。

代码实现如下:

Listing 5: 轨迹执行函数

```
def execute_quintic_curve(coeffs, time):
    # 时间向量
    time_vector = np.array([time**5, time**4, time**3, time**2,
        time, 1])
    # 计算关节位置
    joint_positions = np.dot(coeffs, time_vector)
    return joint_positions
```

中间速度约束的轨迹，也是类似如上代码。

4 机械臂运动控制

为了实现机械臂在 CoppeliaSim 仿真环境中的运动控制,我们编写了 `RobotSimulator` 类，主要完成机械臂的初始化、基座切换、关节控制等功能。该类通过远程 API 与仿真环境交互，控制机械臂的各个关节执行精确运动。

4.1 基座切换

在机械臂运动过程中,左右基座切换是运动控制中的关键环节。我们通过 `switch_base()` 方法，实现机械臂基座从左（“A”）到右（“B”）的切换。

基座切换步骤:

1. 获取当前关节位置。
2. 根据目标基座调整所有关节的父子关系。
3. 重置连接矩阵，确保关节旋转角度与仿真环境保持一致。
4. 设置关节角度为反向值，以适应新基座。

代码实现:

Listing 6: 基座切换代码

```
def switch_base(self, base):
    if self.base == base:
        return
    current_positions = np.array([self.sim.getJointPosition(joint)
       for joint in self.joints])
    if base == "A":
        self._switch_to_base("left", current_positions)
    elif base == "B":
        self._switch_to_base("right", current_positions)
    self.base = base
```

通过该功能，机械臂能够灵活调整工作空间中的位置，扩大了机械臂的可操作范围。

4.2 运动控制

运动控制的核心在于以一定时间间隔调整各关节角至目标位置，从而完成机械臂的任务动作。我们实现了 `move()` 方法，将目标关节位置数组传递到仿真环境: 。

Listing 7: 运动控制代码

```
def move(self, joint_positions):
    if self.base == "A":
        self._set_joint_positions(joint_positions)
    elif self.base == "B":
        self._set_joint_positions(joint_positions[::-1])
```

仿真循环控制逻辑如下，根据当前仿真时间判断当前运动阶段，并执行关节角度调整与基座切换:

Listing 8: 仿真主循环

```
# 仿真循环控制
while (current_time := robot_simulator.sim.getSimulationTime()) < total_time:
    if current_time < t01:
        robot_simulator.switch_base("B")
        joint_q = execute_quintic_curve(coeffs_01, current_time)
    elif current_time < t01 + t12:
        robot_simulator.switch_base("A")
        joint_q = execute_quintic_curve(coeffs_12, current_time - t01)

    robot_simulator.move(joint_q)
    robot_simulator.client.step()
```

5 仿真结果展示

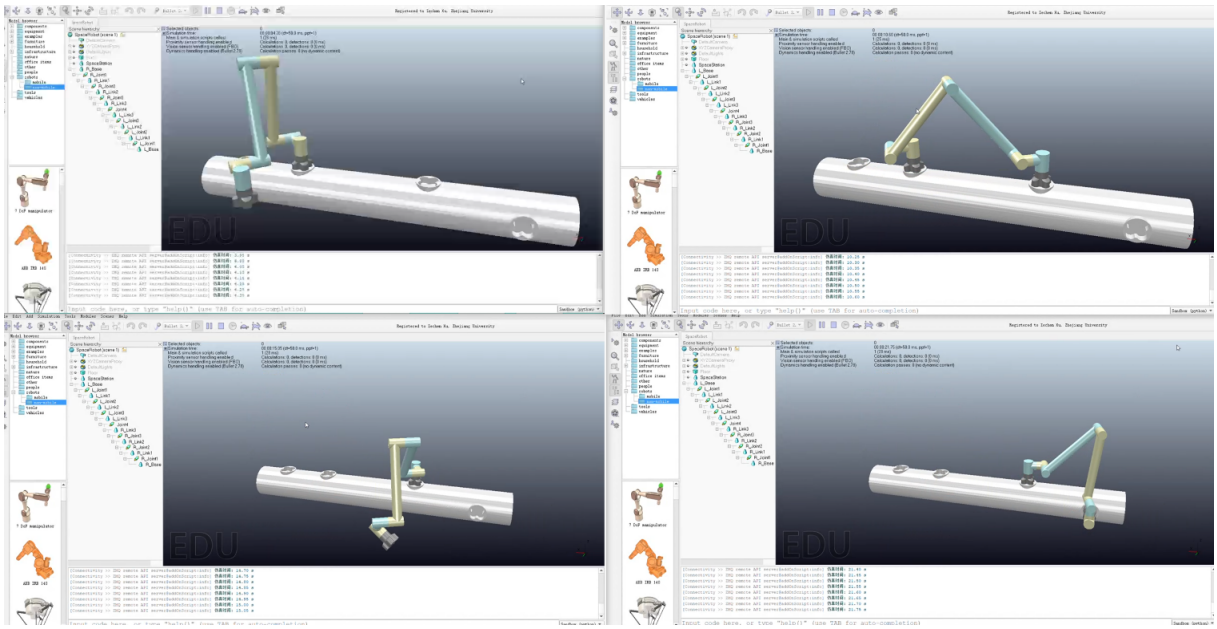

图 4: 机械臂运动仿真过程截图

通过仿真结果可以看出，机械臂在整个运动过程中实现了平滑、准确的动态控制，满足了预期的运动规划与目标位置要求。