EE368 · Robotic Motion and Control

Jacobian

Name / Yiwen Ying
Weiyu Chen

May 7, 2025

1 jacobian.py with comments

```
#!/usr/bin/env python3
###
import math
import numpy as np
import rospy
from sensor_msgs.msg import JointState
from geometry_msgs.msg import Point

class Link:
    """
    Link类表示机器人的单个连杆
    使用DH参数来描述连杆的几何特性
    """
    def __init__(self, dh_params):
        # dh_params: [alpha, a, d, theta_offset]
        self.dh_params_ = dh_params

    def transformation_matrix(self, theta):
        """
        计算DH参数下的变换矩阵
        theta: 当前关节角度
        返回: 4x4的齐次变换矩阵
        """
        alpha = self.dh_params_[0] # 关节扭角
        a = self.dh_params_[1]     # 连杆长度
        d = self.dh_params_[2]     # 连杆偏距
        theta = theta+self.dh_params_[3] # 关节角偏移量

        # 计算三角函数值
        st = math.sin(theta)
        ct = math.cos(theta)
        sa = math.sin(alpha)
        ca = math.cos(alpha)

        # 构建DH变换矩阵
        trans = np.array([[ct, -st, 0, a],
                          [st*ca, ct * ca, - sa, -sa * d],
                          [st*sa, ct * sa,   ca,  ca * d],
                          [0, 0, 0, 1]])
        return trans

    @staticmethod
    def basic_jacobian(trans, ee_pos):
        """
        计算基本雅可比矩阵列
        trans: 当前连杆的变换矩阵
        ee_pos: 末端执行器的位置
        返回: 该关节对应的雅可比矩阵列
        """
```

```
        # 提取当前连杆的位置和z轴方向
        pos = np.array([trans[0, 3], trans[1, 3], trans[2, 3]])
        z_axis = np.array([trans[0, 2], trans[1, 2], trans[2, 2]])

        # 计算线速度和角速度分量
        basic_jacobian = np.hstack((np.cross(z_axis, ee_pos - pos), z_axis))
        return basic_jacobian

class NLinkArm:
    """
    NLinkArm类表示由多个连杆组成的机械臂
    实现了正向运动学、逆向运动学等功能
    """
    def __init__(self, dh_params_list) -> None:
        # 根据DH参数列表创建连杆对象
        self.link_list = []
        for i in range(len(dh_params_list)):
            self.link_list.append(Link(dh_params_list[i]))

    def transformation_matrix(self, thetas):
        """
        计算整个机械臂的变换矩阵
        thetas: 所有关节的角度列表
        返回: 从基座标系到末端执行器的总变换矩阵
        """
        trans = np.identity(4)
        for i in range(len(self.link_list)):
            trans = np.dot(trans, self.link_list[i].transformation_matrix(thetas[i]))
        return trans

    def forward_kinematics(self, thetas):
        """
        计算正向运动学
        thetas: 关节角度列表
        返回: 位置和欧拉角
        """
        trans = self.transformation_matrix(thetas)
        x = trans[0, 3]
        y = trans[1, 3]
        z = trans[2, 3]

        # 计算欧拉角
        alpha, beta, gamma = self.euler_angle(thetas)
        return [x, y, z, alpha, beta, gamma]

    def euler_angle(self, thetas):
        """
        从变换矩阵计算欧拉角
        thetas: 关节角度列表
        返回: ZYZ欧拉角
        """
        trans = self.transformation_matrix(thetas)

        # 计算alpha角 (绕Z轴旋转)
        alpha = math.atan2(trans[1][2], trans[0][2])
        # 确保alpha在[-pi/2, pi/2]范围内
        if not (-math.pi / 2 <= alpha <= math.pi / 2):
            alpha = math.atan2(trans[1][2], trans[0][2]) + math.pi
        if not (-math.pi / 2 <= alpha <= math.pi / 2):
```

```python
            alpha = math.atan2(trans[1][2], trans[0][2]) - math.pi

        # 计算beta角（绕新Y轴旋转）
        beta = math.atan2(
            trans[0][2] * math.cos(alpha) + trans[1][2] * math.sin(alpha),
            trans[2][2])

        # 计算gamma角（绕新Z轴旋转）
        gamma = math.atan2(
            -trans[0][0] * math.sin(alpha) + trans[1][0] * math.cos(alpha),
            -trans[0][1] * math.sin(alpha) + trans[1][1] * math.cos(alpha))

        return alpha, beta, gamma

    def inverse_kinematics(self, ref_ee_pose):
        """
        计算逆向运动学
        使用雅可比矩阵的数值迭代法求解
        ref_ee_pose: 目标末端执行器位姿 [x, y, z, alpha, beta, gamma]
        返回: 关节角度列表
        """
        thetas = [0, 0, 0, 0, 0, 0]  # 初始关节角度
        for cnt in range(500):  # 最大迭代次数
            # 计算当前位姿
            ee_pose = self.forward_kinematics(thetas)
            diff_pose = np.array(ref_ee_pose) - ee_pose

            # 计算基本雅可比矩阵
            basic_jacobian_mat = self.basic_jacobian(thetas)
            alpha, beta, gamma = self.euler_angle(thetas)

            # 计算角速度到欧拉角速度的转换矩阵
            K_zyz = np.array(
                [[0, -math.sin(alpha), math.cos(alpha) * math.sin(beta)],
                 [0, math.cos(alpha), math.sin(alpha) * math.sin(beta)],
                 [1, 0, math.cos(beta)]])
            K_alpha = np.identity(6)
            K_alpha[3:, 3:] = K_zyz

            # 计算关节角度的增量
            theta_dot = np.dot(
                np.dot(np.linalg.pinv(basic_jacobian_mat), K_alpha),
                np.array(diff_pose))
            thetas = thetas + theta_dot / 100. # 使用小增量更新关节角度
        return thetas

    def basic_jacobian(self, thetas):
        """
        计算完整的雅可比矩阵
        thetas: 关节角度列表
        返回: 6xn的雅可比矩阵，n为关节数量
        """
        ee_pos = self.forward_kinematics(thetas)[0:3] # 末端执行器位置
        basic_jacobian_mat = []
        trans = np.identity(4)
        for i in range(len(self.link_list)):
            trans = np.dot(trans, self.link_list[i].transformation_matrix(thetas[i]))
            basic_jacobian_mat.append(self.link_list[i].basic_jacobian(trans, ee_pos))
        return np.array(basic_jacobian_mat).T
```

```python

if __name__ == "__main__":
    """
    创建ROS节点，发布工具的位姿、速度和力信息
    """
    # 初始化ROS节点
    rospy.init_node("jacobian_test")
    # 创建发布者
    tool_pose_pub = rospy.Publisher("/tool_pose_cartesian", Point, queue_size=1)
    tool_velocity_pub = rospy.Publisher("/tool_velocity_cartesian", Point, queue_size=1)
    tool_force_pub = rospy.Publisher("/tool_force_cartesian", Point, queue_size=1)

    # 机械臂的DH参数
    dh_params_list = np.array([[0, 0, 243.3/1000, 0],
                               [math.pi/2, 0, 10/1000, 0+math.pi/2],
                               [math.pi, 280/1000, 0, 0+math.pi/2],
                               [math.pi/2, 0, 245/1000, 0+math.pi/2],
                               [math.pi/2, 0, 57/1000, 0],
                               [-math.pi/2, 0, 235/1000, 0-math.pi/2]])
    # 创建机械臂对象
    gen3_lite = NLinkArm(dh_params_list)

    while not rospy.is_shutdown():
        # 获取关节状态
        feedback = rospy.wait_for_message("/my_gen3_lite/joint_states", JointState)
        thetas = feedback.position[0:6]       # 关节角度
        velocities = feedback.velocity[0:6]   # 关节速度
        torques = feedback.effort[0:6]        # 关节力矩

        # 计算ee的位姿、速度和力
        tool_pose = gen3_lite.forward_kinematics(thetas)
        J = gen3_lite.basic_jacobian(thetas)
        tool_velocity = J.dot(velocities)      # ee速度 = 雅可比矩阵 × 关节速度
        tool_force = np.linalg.pinv(J.T).dot(torques)  # ee力 = 雅可比矩阵转置逆 × 关节力矩

        # 创建并发布消息
        # ee位姿
        tool_pose_msg = Point()
        tool_pose_msg.x = tool_pose[0]
        tool_pose_msg.y = tool_pose[1]
        tool_pose_msg.z = tool_pose[2]

        # ee速度
        tool_velocity_msg = Point()
        tool_velocity_msg.x = tool_velocity[0]
        tool_velocity_msg.y = tool_velocity[1]
        tool_velocity_msg.z = tool_velocity[2]

        # ee力
        tool_force_msg = Point()
        tool_force_msg.x = tool_force[0]
        tool_force_msg.y = tool_force[1]
        tool_force_msg.z = tool_force[2]

        # 发布
        tool_pose_pub.publish(tool_pose_msg)
        tool_velocity_pub.publish(tool_velocity_msg)
        tool_force_pub.publish(tool_force_msg)

```

```
        # 打印
        print(f"joint position: {thetas}")
        print(f"joint velocity: {velocities}")
        print(f"joint torque: {torques}")

        print(f"tool position: {tool_pose}")
        print(f"tool velocity: {tool_velocity}")
```

The codes are correct.

We can use D-H Table to transform the links:

$$T = \begin{bmatrix} \cos(\theta+\theta_{\text{offset}}) & -\sin(\theta+\theta_{\text{offset}}) & 0 & a \\ \sin(\theta+\theta_{\text{offset}})\cos(\alpha) & \cos(\theta+\theta_{\text{offset}})\cos(\alpha) & -\sin(\alpha) & -d\sin(\alpha) \\ \sin(\theta+\theta_{\text{offset}})\sin(\alpha) & \cos(\theta+\theta_{\text{offset}})\sin(\alpha) & \cos(\alpha) & d\cos(\alpha) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

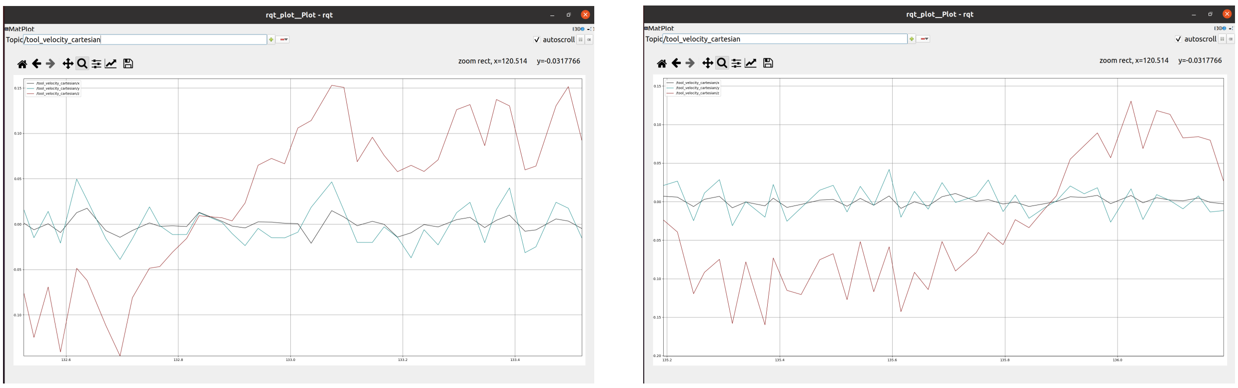
The Jacobian can be computed by

$$J_i = \begin{bmatrix} \mathbf{z}_{i-1} \times (\mathbf{p}_e - \mathbf{p}_i) & \mathbf{z}_{i-1} \end{bmatrix} \tag{2}$$

Using arithmetic iteration, we can compute inverse kinematics:

$$\begin{aligned} \Delta\theta &= J^{\dagger} K_{\text{ZYX}}(\mathbf{p}_{\text{ref}} - \mathbf{p}_{\text{current}}) \\ \theta_{\text{new}} &= \theta_{\text{old}} + \frac{\Delta\theta}{100} \end{aligned} \tag{3}$$

2 Velocity Curve

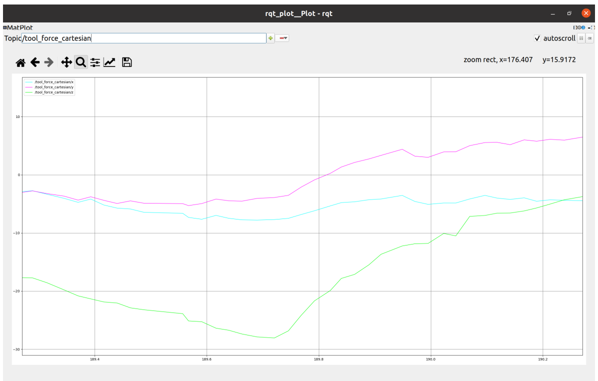


(a) (b)

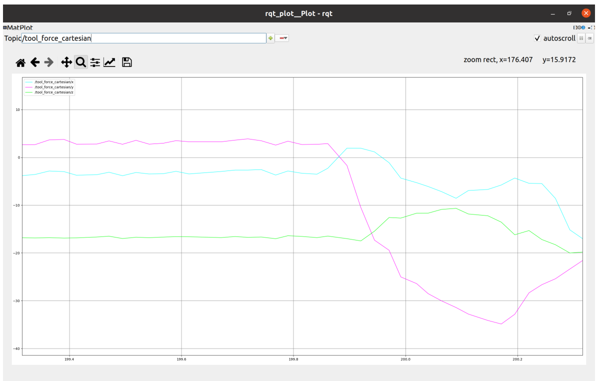
Figure 1: Velocity Curve

Topic: */tool_velocity_cartesian* publishes the linear velocity of the end-effector in Cartesian space.

3 Force Curve



(a)



(b)

Figure 2: Force Curve

Topic: $/tool\_force\_cartesian$ publishes the equivalent linear force of the end-effector in Cartesian space.