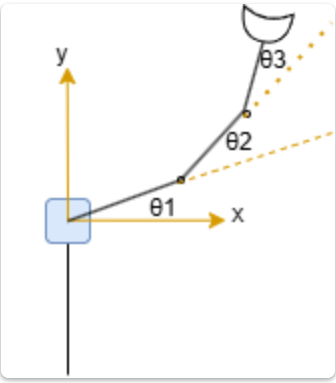
WBC三自由度机械臂教程

项目参考：[# 基于零空间方法（NUB）的全身控制（WBC）的简单实现]((32 封私信 / 80 条消息) 基于零空间方法（NUB）的全身控制（WBC）的简单实现 - 知乎)
本项目基于gazebo实现上述帖子中的三自由度机械臂WBC控制项目
笔者使用的环境
ubuntu20.04，ROS noetic
在阅读本教材前，请先仔细阅读完WBC学习笔记.pdf，本教材只作为实践练习使用。
该机械臂简图由下方所示：



在一个直线轨道上有一个滑块，沿着y轴自由运动，滑块上有平面3R机械臂，各连杆长度均为1m，定义刚性连接到滑块上，且与世界坐标系方向相同的坐标系为滑块坐标系。设计3R机械臂控制系统，满足以下要求：

- 控制系统输入：滑块在轨道上的位置 y_h，机械臂各关节位置

$$\boldsymbol{\theta} = [\theta_1 \quad \theta_2 \quad \theta_3]^T$$

  。
- 控制系统输出：机械臂各个关节的速度 $\dot{\boldsymbol{\theta}}$。
- 控制任务1：让机械臂末端位置保持在世界坐标系下点

$$\boldsymbol{A} = [2 \quad 0]^T$$

- 控制任务2：让机械臂末端朝向x轴。

传统控制思路：
在滑块坐标系上，机械臂末端的期望位姿为：

$$\boldsymbol{x}_d = \begin{bmatrix} 2 \\ -y_h \\ 0 \end{bmatrix}$$

在滑块坐标系下，机械臂末端的运动学正解为：

$$\boldsymbol{x} = \begin{bmatrix} l_1c_1 + l_2c_{12} + l_3c_{123} \\ l_1s_1 + l_2s_{12} + l_3s_{123} \\ \theta_1 + \theta_2 + \theta_3 \end{bmatrix}$$

使用PD控制器，有

$$\dot{\boldsymbol{x}}^{des} = k_p(\boldsymbol{x_d} - \boldsymbol{x}) + k_d\dot{\boldsymbol{x}}^{cur}$$

将运动学正解微分有雅可比矩阵为

$$\dot{\boldsymbol{x}} = J\dot{\theta}$$

于是可以得到关节速度的输出

$$\dot{\theta} = J^{+}\dot{\boldsymbol{x}}$$

，J^{+}为**伪逆矩阵**，当J有逆则为逆，当J无逆则为满足

$$\dot{\theta} = J^{+}\dot{\boldsymbol{x}}$$

的最小二乘解。这种方式，在机械臂期望位置超出机械臂可达工作空间，系统将会非常扭曲。

但是当我们使用WBC后，将位置控制定为高优先级，姿态控制定为低优先级，则可一定程度上解决上述问题。
由**WBC学习笔记.pdf**知，最终的关节角速度应为：

$$\dot{q}_2 = \dot{q}_1 + (J_2N_1)^{\dagger}(\dot{x}_2 - J_2\dot{q}_1)$$

而

$$\dot{q}_1 = J_1^{\dagger}\dot{x}_1$$

在项目中，最后函数`wbc_calculate(self):`的答案为：

```
# WBC控制的实现
# 1. PD控制器参数（仅保留Kp/Kd，无阻尼伪逆）
task1_Kp = 10
task1_Kd = 0.8
task2_Kp = 100
task2_Kd = 0.4

theta1 = self.joint_angles[0]
theta2 = self.joint_angles[1]
theta3 = self.joint_angles[2]

q_vel = np.array([self.joint_vels[0], self.joint_vels[1],
self.joint_vels[2]]).reshape(-1, 1)
```

```
theta12 = theta1 + theta2
theta123 = theta1 + theta2 + theta3
s1 = math.sin(theta1)
s12 = math.sin(theta12)
s123 = math.sin(theta123)
c1 = math.cos(theta1)
c12 = math.cos(theta12)
c123 = math.cos(theta123)

# 计算雅可比矩阵并计算其伪逆，以及期待的任务空间的速度（基于PD控制器）
J_1 = np.array([
    [-s1 - s12 - s123, -s12 - s123, -s123],
    [c1 + c12 + c123,  c12 + c123,   c123]
])
J_2 = np.array([[1.0, 1.0, 1.0]])

x_1 = np.array([c1 + c12 + c123, s1 + s12 + s123]).reshape(-1, 1)
x_1d = np.array([2.0, -self.y_h]).reshape(-1, 1)
x_2 = np.array([theta123]).reshape(-1, 1)
x_2d = np.array([0.0]).reshape(-1, 1)

x_vel_1 = J_1 @ q_vel
x_vel_2 = J_2 @ q_vel

x_dot_1 = task1_Kp * (x_1d - x_1) - task1_Kd * x_vel_1
x_dot_2 = task2_Kp * (x_2d - x_2) - task2_Kd * x_vel_2

J_pinv1 = np.linalg.pinv(J_1)
J_pinv2 = np.linalg.pinv(J_2)

I3 = np.eye(3)
theta_dot = J_pinv1 @ x_dot_1 \
          + (I3 - J_pinv1 @ J_1) @ J_pinv2 @ (x_dot_2 - J_2 @ J_pinv1 @ x_dot_1)

max_vel = 2.0
theta_dot[0, 0] = np.clip(theta_dot[0, 0], -max_vel, max_vel)
theta_dot[1, 0] = np.clip(theta_dot[1, 0], -max_vel, max_vel)
theta_dot[2, 0] = np.clip(theta_dot[2, 0], -max_vel, max_vel)

self.JointVel[0] = theta_dot[0, 0]
self.JointVel[1] = theta_dot[1, 0]
self.JointVel[2] = theta_dot[2, 0]

# 调试打印（可选，查看PD效果）
rospy.loginfo_throttle(1.0,
```

```
    f"任务1误差: {np.linalg.norm(x_1d - x_1):.2f} | 末端速度:
{np.linalg.norm(x_vel_1):.2f}\n"
    f"任务2误差: {np.linalg.norm(x_2d - x_2):.2f} | 角度和速度:
{np.linalg.norm(x_vel_2):.2f}"
)
```