个性化实践内容

1. OpenArm 机械臂运动学控制

本学期，我加入了具身智能实验室，负责 OpenArm 七自由度机械臂的运动学控制与轨迹规划。期间，我独立搭建了完整的运动学解算库，完成了包括硬件在环仿真、单臂轨迹规划、半人形机器人整体控制等工作。

1.1. 项目介绍：

七自由度 S-R-S 机械臂是一种与人手臂自然结构产生的自由度一致的机械臂构型，其结构分为肩部，肘部和腕部，其中肩部，肘部与腕部分别由三个相交轴旋转副构成，可以视作一个球铰，肘部由一个旋转副组成，故称为 S-R-S 机械臂（球面铰-旋转副-球面铰）。

OpenArm 就是典型的 S-R-S 冗余机械臂，其作为一款安全、开源、高性能的人形机械臂平台，为物理 AI 研究和工业协作提供了理想解决方案。它在保证安全性的同时，提供了类人操作性能和强大的开发灵活性，使研究人员和开发者能够快速迭代和部署创新应用。

图 1.1.1 OpenArm 基本参数

图 1.1.2 OpenArm 硬件选项

我的整体工作流程按以下思路开展：仿真环境搭建→运动学库构建→单片机移植→单臂运动学解算与轨迹规划→运动学性能分析→半人形机器人整体控制。

1.2. 仿真环境搭建：

我使用的系统是 Linux 的 Ubuntu22.04，仿真环境基于 ROS2，编译器为 VScode。此外，我选择 Rviz 作为可视化机器人模型的平台。

Rviz 是一款三维可视化工具，很好的兼容了各种基于 ROS 软件框架的机器人平台，可以使用 XML 对机器人、周围物体等任何实物进行尺寸、质量、位置、材质、关节等属性的描述，并且在界面中呈现出来。同时，rviz 可通过图形化的方式实时显示机器人传感器的信息、机器人的运动状态、周围环境的变化等。

整体的仿真环境搭建流程如下图所示：

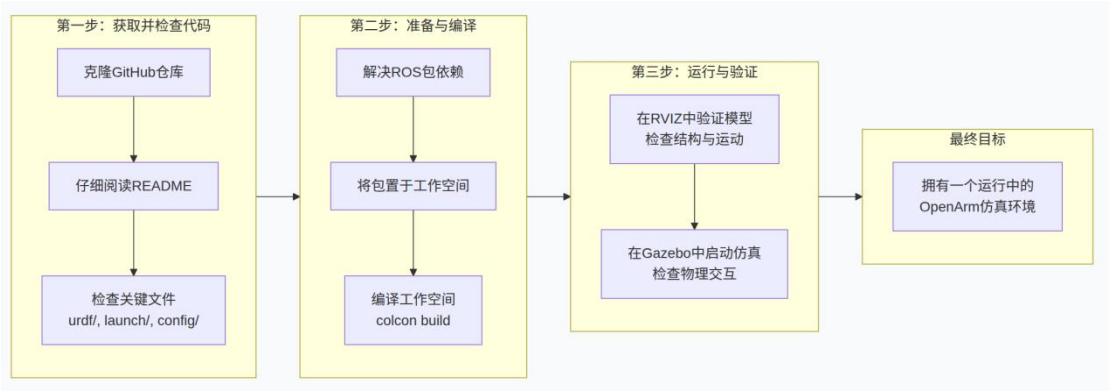


图 1.2.1 仿真环境搭建流程

我调用了 openarm_description 仓库中的 URDF 模型，通过配置启动文件将其可视化在 Rviz 中。为了验证机械臂的运动链路是否完整，我使用图形化 API——GUI 滑块对机械臂的七个关节角（Openarm_joint1~7）进行控制。

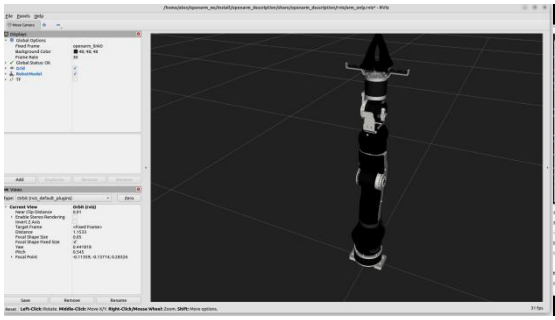


图 1.2.2 Rviz 平台可视化

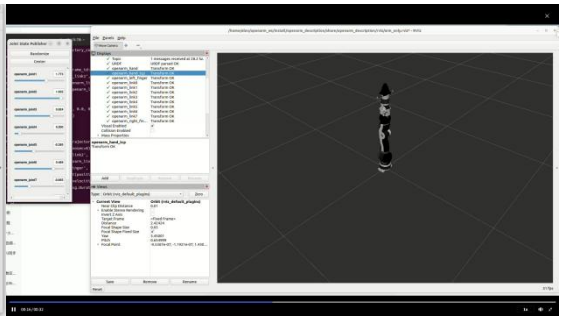


图 1.2.3 GUI 滑块控制

1.3. 运动学库构建：

为了寻找一个支持硬件在环仿真工作的运动学库，我必须使用 C/C++库，其中，KDL 库是一款应用较广且完整的开源库，其作为一个轻量级、跨平台的开源 C++ 库，专为关节型机器人（如机械臂） 设计，提供了一套完整的工具链，用于解决机器人正逆运动学求解、雅克比矩阵计算、轨迹生成、刚体动力学分析等核心问题，是机器人开发中解决“运动规划与控制”核心问题的基础工具库，也是 ROS/ROS2 生态中运动学模块的核心依赖。

集成 KDL 进行正运动学验证是连接“关节空间”与“笛卡尔空间”的关键。然而，由于我最终要实现的目标是使用单片机进行虚拟机械臂控制，单片机无法调用基于 ROS2 的开源运动学库。因此，在验证了 KDL 库后，我对其进行解耦与

提纯，重新搭建了一个专用于单片机的运动学框架。

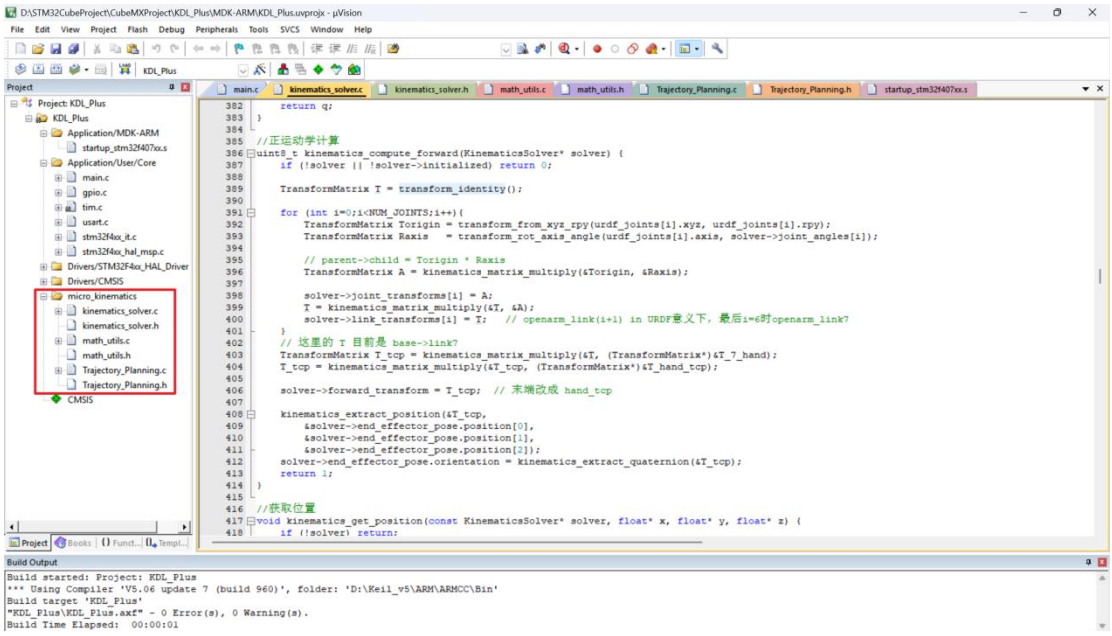


图 1.3.1 重构的 C 语言运动学库

如图 1.3.1，核心运动学文件为 kinematics_solver.c/h，math_utils.c/h，Trajectory_Planning.c/h，分别对应运动学求解器的构建、运动学解算所需的数学函数与工具、轨迹规划任务，总代码数约 1500 行。我使用结构体的形式将核心运动学函数封装在其中，便于调用，可移植性高。

1.4. 单片机移植：

基于重新构建的运动学库，我进行了单片机的算法移植。

众所周知，机械臂运动学解算是一个相对耗费算力的工程，因此我决定使用正点原子探索者开发板，其 MCU 为 STM32F407ZGT6，具有 FPU 浮点运算单元（一种集成在 MCU 中的硬件专用模块，专门用于加速浮点数运算操作）和足够的 RAM（192KB），可以高效处理浮点运算，且拥有丰富的各类接口，可以极大程度方便后续的调试。

图 1.4.1 探索者 V3 硬件资源分布

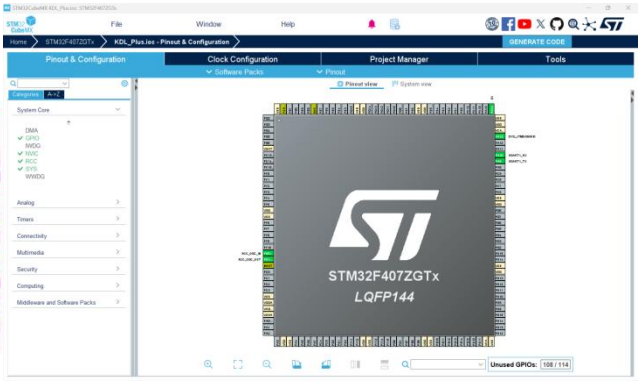


图 1.4.2 引脚资源与配置

我使用的工具链是 STM32CubeMX+Keilv5，是基于 HAL 库的 STM32 代码框架。为提高计算效率，选择使用外部高速晶振，将主频提升至 168MHz，并启用 FPU 浮点计算单元。开启 USART1，使用 DMA 传输的方式将计算结果通过 USBtoTTL 模块上传至 ROS 端。

在开始硬件在环仿真前，我首先对比了原生 KDL 库与单片机解算出的运动学数据，对单片机中的核心运动学代码进行验证，从而避免 ROS 虚拟环境迁移到嵌入式硬件中可能产生的错误，这也是现实工程中一个不可或缺的重要流程。

我在 Keil 与 ROS 中给定一致的机械臂 DH 参数和关节角指令，并用串口捕获输出的解算结果，如下图所示：

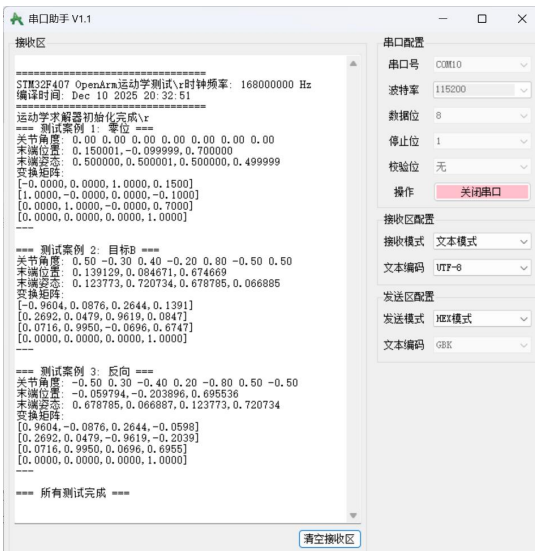


图 1.4.1 重构库的解算结果（STM32 端）

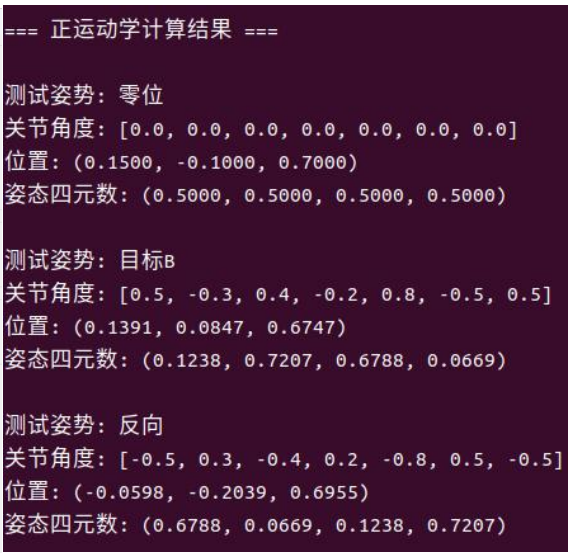


图 1.4.2 KDL 库解算结果（ROS 端）

通过与 ROS 端 KDL 库计算得出的结果相对比，可以发现两种方法算出的机械臂末端位姿数据完全一致，由此验证了重构库的核心运动学算法的正确性。

为了让 ROS 端能接受到 STM32 发送的数据，我创建了一个 Rviz 专用的串口节点脚本 scripts/rviz_serial_controller.py，将关节角数据以 json 格式发送至 Rviz 节点，从而实现对虚拟机械臂的控制。

1.5. 单臂运动学解算与轨迹规划：

在实现硬件在环仿真，成功打破了 STM32 与 ROS 之间的壁垒后，为了进一步加强对机械臂的控制，我使用逆运动学进行机械臂的轨迹规划。

在这里，我使用了 DLS 伪逆法来处理逆运动学求解，通过引入一个阻尼系数 λ，解决奇异性和不稳定性问题。流程是在每个迭代步骤中，计算关节的增量（dq），并更新当前的关节角度，通过正运动学的验证与反馈不断逼近，直到达到目标位置，是冗余机械臂和高精度轨迹跟踪场景的主流解法。

$$J^{+} = J^{T}\left(JJ^{T} + \lambda^{2} I\right)^{-1}$$

图 1.5.1 阻尼最小二乘法雅克比矩阵伪逆公式

为了一步步实现复杂的轨迹规划，我将逆运动求解器的雅可比分为 3×7 位置版本和 6×7 位姿版本。其中，位置版本只考虑末端坐标（x,y,z），位姿版本则是同时考虑坐标和姿态，相应的计算复杂度与计算速度也有算不同。

如下表所示，我为机械臂设计了其中运动模式，分别用于验证正运动学，逆运动学，测试求解器性能等。

模式ID	类型	名称	描述	作用
模式 0	正运动学	关节角正弦波	各关节独立正弦波运动	验证硬件在环是成功
模式 1	正运动学	步进测试	5个固定位置循环切换	验证正运动学精度
模式 2	正运动学	慢速平滑正弦波	低频正弦波，运动更平滑	验证正运动学动态性能
模式 3	逆运动学	末端画圆	XY平面上的圆形轨迹，Z固定	验证平面轨迹跟踪精度
模式 4	逆运动学	末端X轴往返	X方向直线往复运动	验证单轴动态性能
模式 5	逆运动学	末端Y轴往返	Y方向直线往复运动	验证单轴动态性能
模式 6	逆运动学	末端Z轴往返	Z方向直线往复运动	验证垂直方向性能

图 1.5.2 重构的 C 语言运动学库

1.5.1. 3×7 位置版本

对于位置快版本，雅可比矩阵的维度是 3×7，末端误差维度是 e = [dx, dy, dz]，DLS 伪逆也是 3×3。其轨迹规划效果如下：

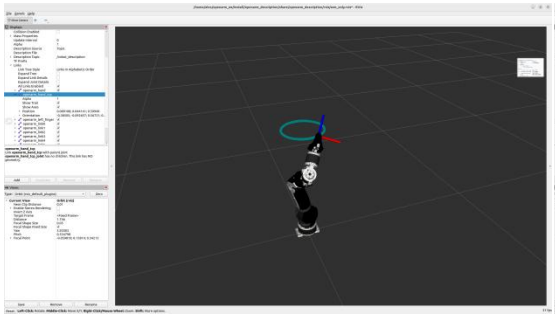


图 1.5.3 case3:XY 平面画圆

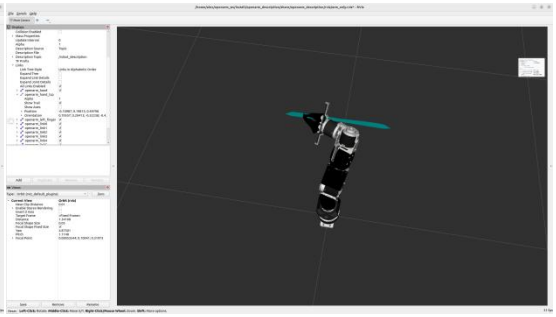


图 1.5.4 case4:沿 X 轴画直线

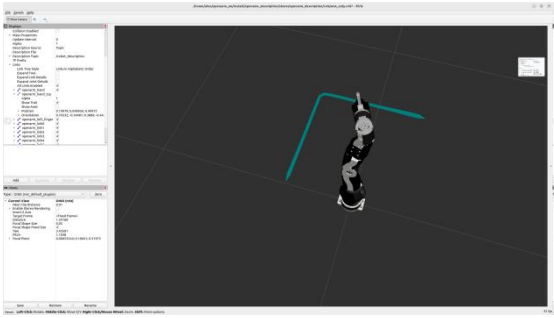


图 1.5.5 case5:沿 Y 轴画直线

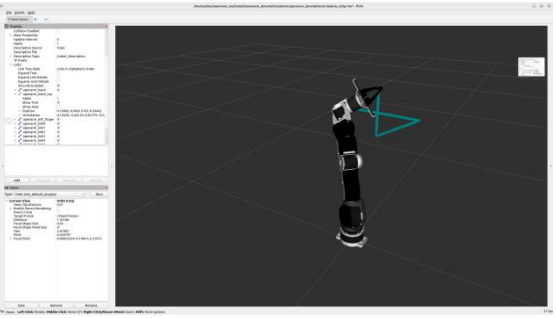


图 1.5.6 case6:沿 Z 轴画直线

1.5.2. 6×7 位姿版本

3*7 位置版本只关注末端执行器的坐标，而不考虑姿态，和现实中的轨迹规划仍有差距。为了实现更加优雅的控制，我进行了算法的升级，将 3*7 坐标雅可比升级为 6*7 位姿雅可比。3×7 位置版本到 6×7 位姿版本的核心变化只有三件事：

1. 雅可比从 3×7 变成 6×7（线速度 + 角速度）

2. 误差从 e=[dx dy dz] 变成 e=[dx dy dz dθx dθy dθz]（姿态用“旋转向量”误差）
3. DLS 伪逆从（3×3）逆变成（6×6）逆

位姿版本的逆运动学控制效果如下：

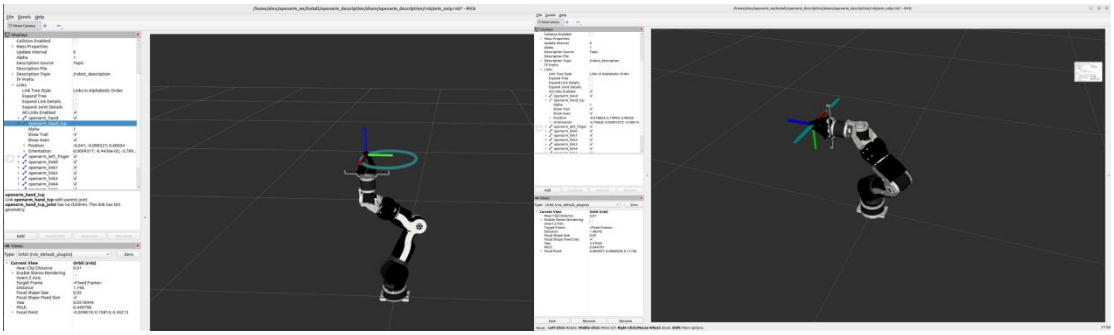


图 1.5.7 case3:XY 平面画圆　　图 1.5.8 case4:沿 X 轴画直线

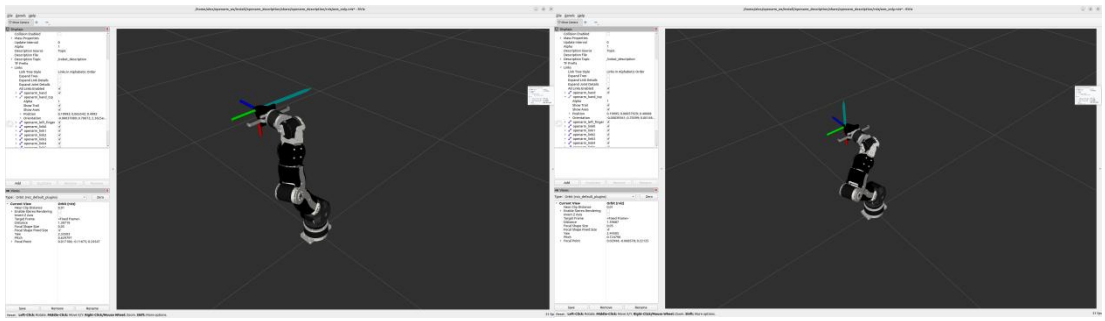


图 1.5.9 case5:沿 Y 轴画直线　　图 1.5.10 case6:沿 Z 轴画直线

通过对比可以观察到，在 6×7 位姿版本的控制下，机械臂末端执行器的姿态会根据目标轨迹进行改变。

至此，我实现了单个 S-R-S 机械臂的完整运动学解算与轨迹规划。

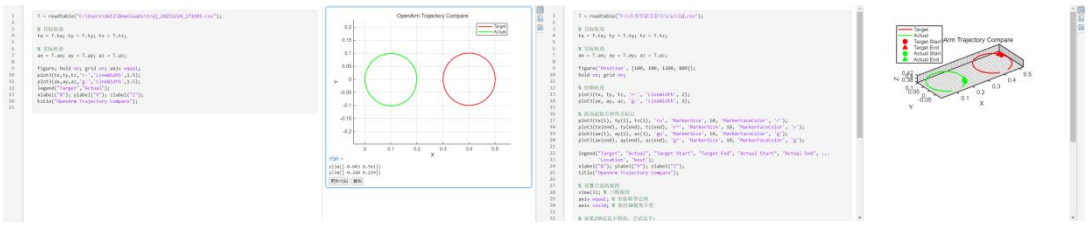
1.6. 运动学性能分析：

为了验证逆运动学算法的精度、效率等性能，我通过记录目标轨迹和实际轨迹的数据，将其在 matlab 上拟合进行对比，从而观察轨迹规划的效果，并通过 STM32 的时钟定时器测出一次逆运动学计算需要的时间来评估效率。

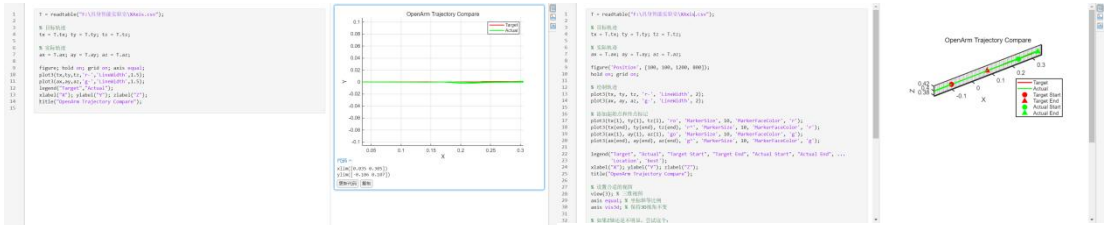
1.6.1. 轨迹精度

我在 ROS 端创建了轨迹生成模块，将机械臂末端的理想位姿与实际位姿数据记录在 CSV 文件中。为了拟合这些数据点，我在 Matlab 中创建了一个可视化脚本，将提取到的数据组拟合为不同颜色的目标轨迹与实际轨迹，拟合效果如下：

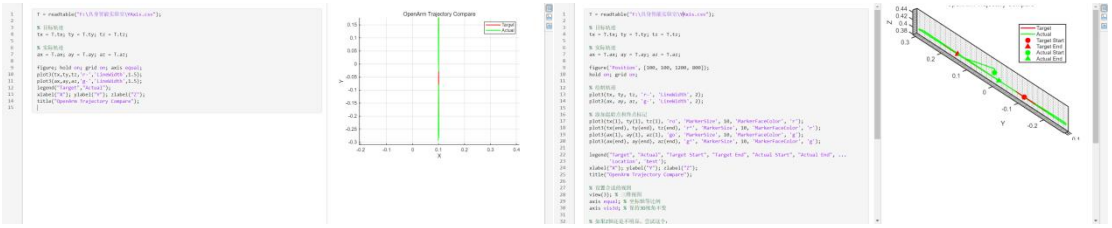
case3:在 XY 平面画圆



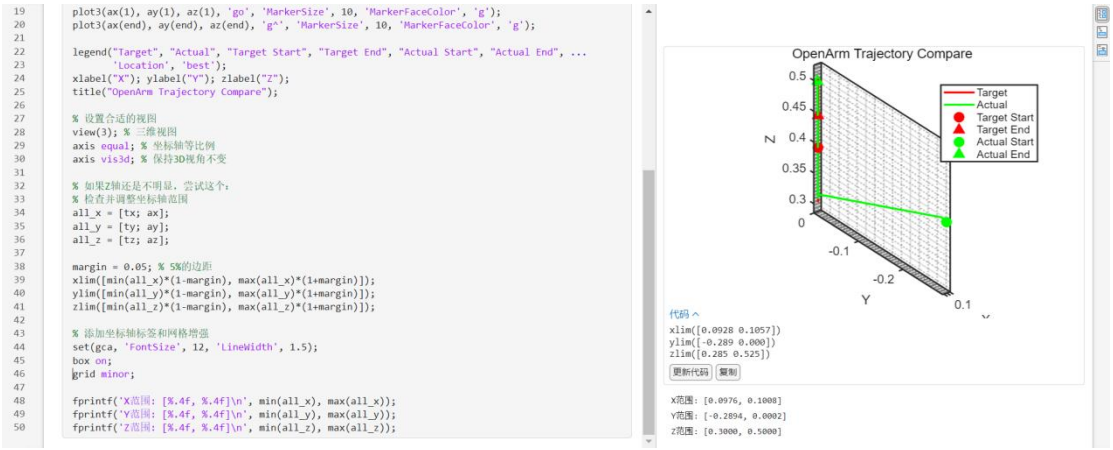
case4:沿 X 轴画直线



case5:沿 Y 轴画直线



case6:沿 Z 轴画直线



1.6.2. 计算效率

单片机都有硬件定时器，可以用于高精度地测量时间。使用硬件定时器的方法通常是精确且不受程序其他部分影响的。STM32F407 的 TIM2/TIM5 是 32 位通用定时器（计数范围 0~0xFFFFFFFF），远超 16 位定时器（0~0xFFFF），既保证高精度又避免频繁溢出。

因此，在 STM32 端，我启用了通用定时器 TIM2，其定时器时钟最高 84MHz（APB1 时钟 42MHz×2），其最高精度可达到 11.9ns，完全足够足够满足此处的测试。

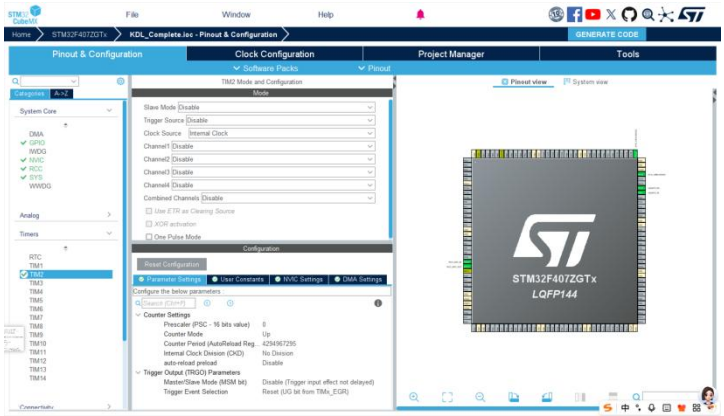


图 1.6.2.1 定时器配置

我使用的计时函数是__HAL_TIM_GET_COUNTER(&htim2)，其定时器计数原理由纯硬件实现，无需中断、无需 MCU 干预，计数过程和代码执行完全独立，精度为硬件级，无软件延迟。

前面我们提到，STM32F407ZGT6 具有专门用于浮点计算的 FPU，能提高浮点运算的效率。在这里为了形成对比，我用串口接收了四种条件（快（慢）版本有（无）FPU）下的逆运动学解算时间，如下所示。

3×7 坐标 IK：

图 1.6.2.2 未启用 FPU

图 1.6.2.3 启用 FPU

6×7 位姿 IK：

图 1.6.2.4 未启用 FPU

图 1.6.2.5 启用 FPU

可以看出：

- 3×7 坐标 IK 不启用 FPU 时单次解算时间大约在 11000~12000us 间；
- 3×7 坐标 IK 启用 FPU 时单次解算时间大约在 4000~5000us 间；
- 6×7 坐标 IK 不启用 FPU 时单次解算时间大约在 14000~40000us 间；
- 6×7 坐标 IK 启用 FPU 时单次解算时间大约在 6000~8000us 间；

根据分析，3×7 坐标 IK 整体相较于 6×7 坐标 IK 更快，是因为其只需要解算位置而不用考虑姿态，维度更低，计算更简洁；而启用 FPU 与否同样对计算效率有较大影响，因为代码中的计算使用了大量的浮点数。

综上，结果符合预期。

1.7. 半人形机器人整体控制：

完成了上述的单臂解算和性能分析流程后，我开始进行完整的 OpenArm 半人形机器人的控制，其核心在于双臂解算与协同控制。我创建了一个 xacro 脚本，将完整的 OpenArm 半人形机器人模型导入 Rviz 中，经过校位和调整后效果如下：

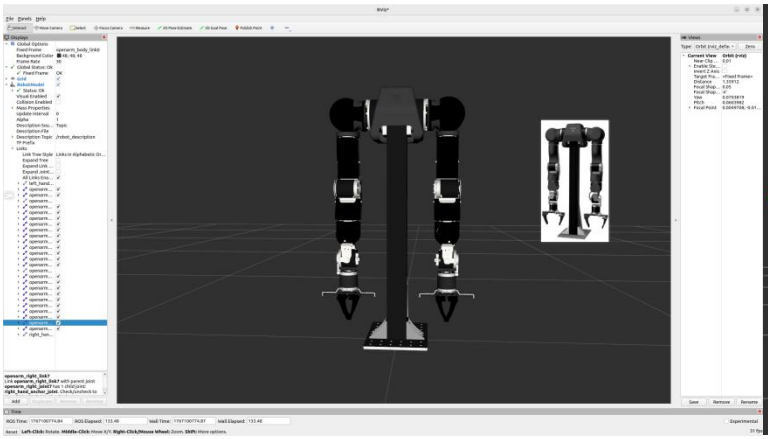


图 1.7.1 OpenArm 半人形机器人

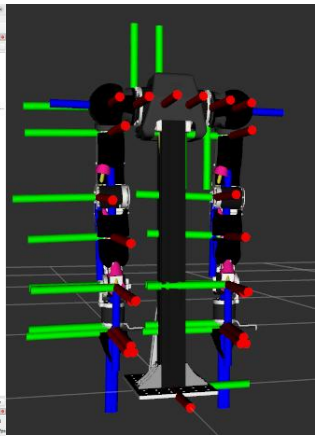


图 1.7.2 关节轴

为了验证其结构完整性与节点是否正确，我同样在单片机控制前先启用了 GUI 滑块控制器。

由于之前在进行单臂解算时，我已经将核心运动学函数都以结构体形式封装在专门的运动学文件中，因此将其扩展为双臂控制的应用较为方便。然而，半人形机器人的整体控制并非简单的将双臂*2。在坐标对齐和双臂的协调控制上，还需要多花功夫。

为了实现对机器人双臂的实时控制，每次发送的 json 数据中需要包含 14 个关节角数据，并且我定义的发送频率为 20Hz，这样的中断频率不利于 MCU 的进程运行，并且容易造成数据丢包或覆盖的现象。于是，我启用了单片机的一个独立外设控制器——DMA（直接存储器访问），核心作用是让数据在“内存”和“外设”之间（或内存和内存之间）直接传输，不仅全程不需要 MCU 参与，还能提高数据的传输速度，非常适用于快速连续的数据传递。

随后，我创建了三个任务对其进行关节角的控制，分别是双臂同向运动、双臂镜像运动、双臂此起彼伏模拟步态运动：

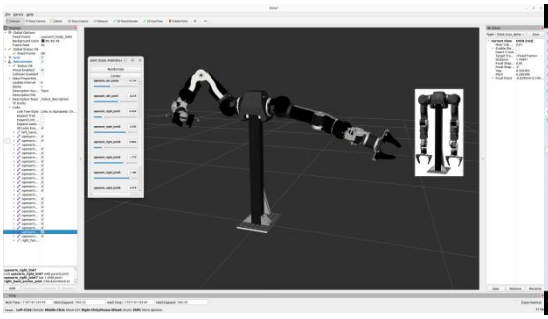


图 1.7.3 GUI 滑块控制

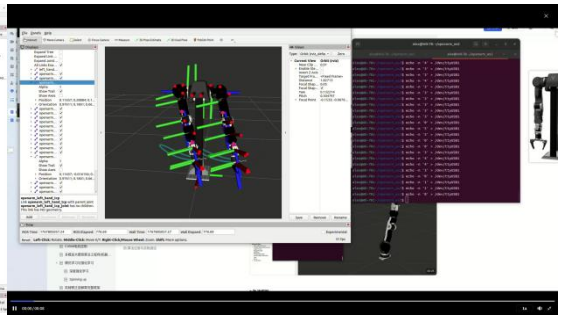


图 1.7.4 双臂同向运动

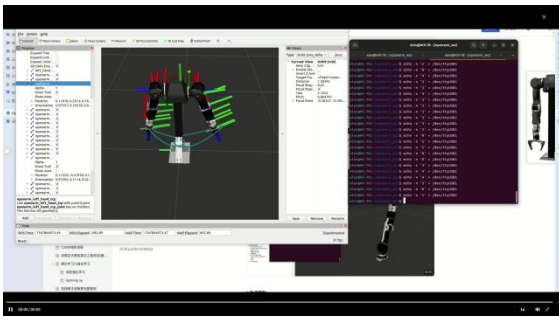


图 1.7.5 双臂镜像运动

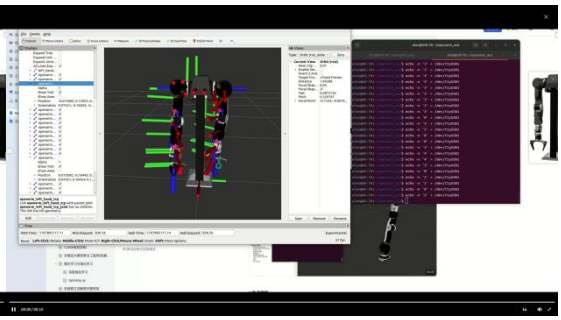


图 1.7.6 模拟步态

后续工作中，我将继续推进半人形机器人的整体轨迹规划与双臂协同控制。