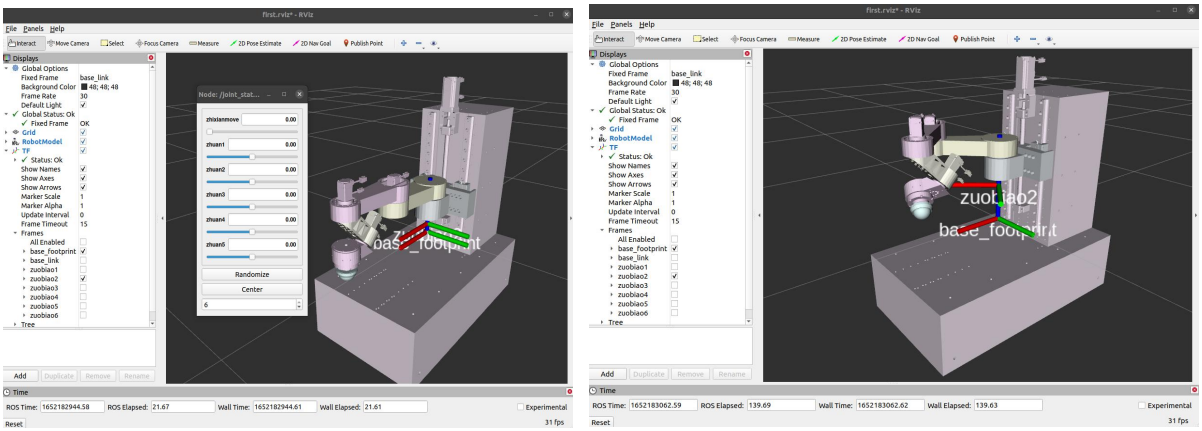
1.生成 urdf 文件

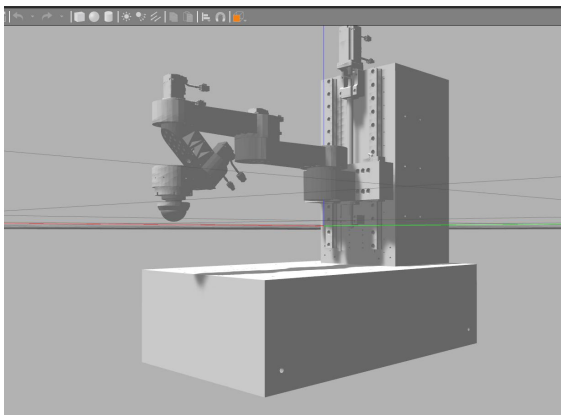
根据 DH 坐标系建立相关的坐标，然后通过 solidworks 进行坐标建立，尤其是需要注重 base_link 的建立，安装导出 urdf 插件并利用 export as URDF 得到涵盖 urdf 模型的功能包，在 ROS 中建立工作空间将功能包放入成功编译，可以发现涵盖了 rviz 和 gazebo 相关的 launch 文件，我们更关心里面的 urdf 文件，其中 joint 标签表示各个零部件的连接关系，此项目将最后关节即转轴设定为 continues 类型，最后通过 check_urdf 指令检查模型的从属关系：

```
robot name is: mingurdf
---------- Successfully Parsed XML ---------------
root Link: base_footprint has 1 child(ren)
    child(1):  base_link
        child(1):  zuobiao1
            child(1):  zuobiao2
                child(1):  zuobiao3
                    child(1):  zuobiao4
                        child(1):  zuobiao5
                            child(1):  zuobiao6
```

我们的是串联机器人，因此该坐标目录是正确的，下面通过 Rviz 显示模型，通过 joint_state_publisher_gui 功能包进行手动节点控制：

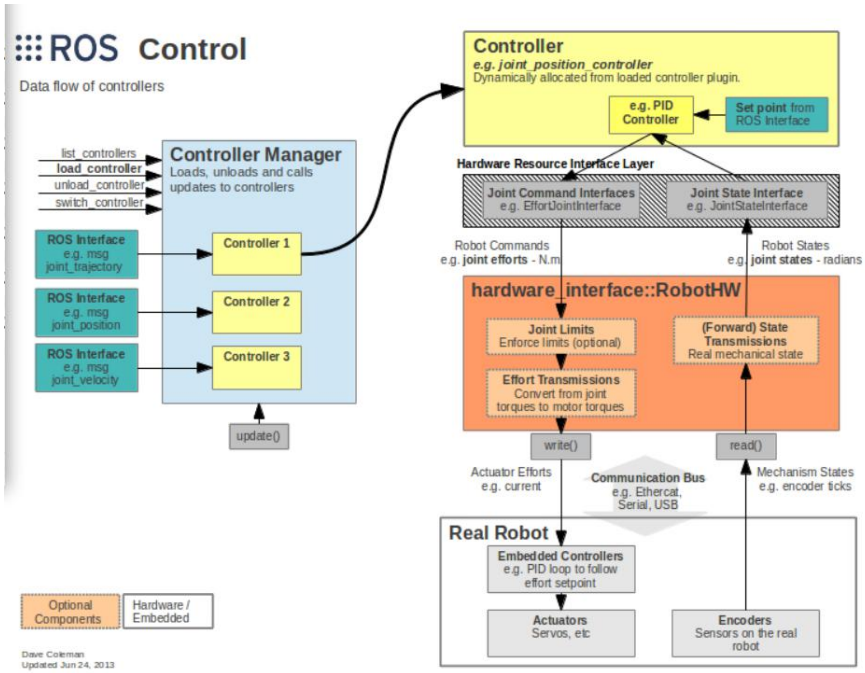


证明 ufdf 模型正确，接下来是 gazebo 测试，由于我们将基坐标设置在了大理石台面上，因此显示的时候大理石台基底在全局坐标之下：



2.消息发布 gazebo 仿真

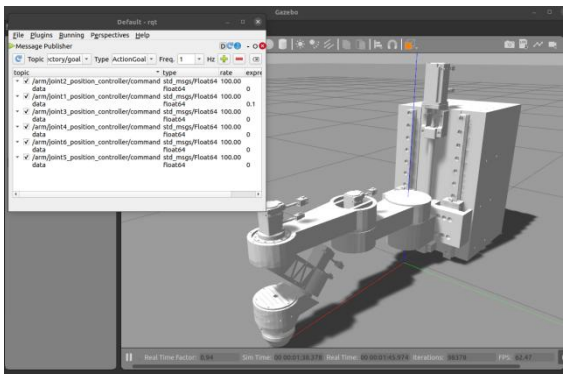
本模型将 base_link 固定，保证大理石台的固定，由于我们定义的 base_link 坐标系是位于大理石台面之上，因此大理石台在全局坐标 z 负向区域，gazebo 仿真考虑了碰撞和各个零部件的质量惯量属性，因而 gazebo 仿真属动力学仿真，moveit！也需要使用 ros_control 接口，ros_control 是一个功能包，可以将非标机械臂的各个关节状态变量作为输入，可以通过控制力矩或者位置等接口实现对机械臂的控制，ros_control 的模式如下：



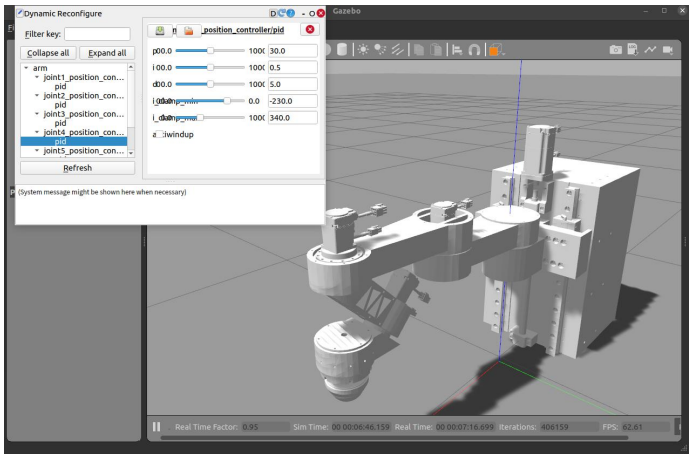
如上图所示，控制器层和真正机械臂之间还有一个硬件层，本文在相关 yaml 文件中进行了控制器的声明，同时在 urdf 文件中添加了硬件层的 transmission 并定义了各个 joint 标签的角度限制等，由于本文采用 gazebo 进行动力学仿真，因此需要在 urdf 文件里加入传动装置也就是执行器并配置相应的控制器，这与实际中的机械臂驱动器控制器的原理是类似的，具体如下：

```
<transmission name="trans_zhixianmove">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="zhixianmove">
        <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
    </joint>
    <actuator name="zhixianmove_motor">
        <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
        <mechanicalReduction>100</mechanicalReduction>
    </actuator>
</transmission>
```

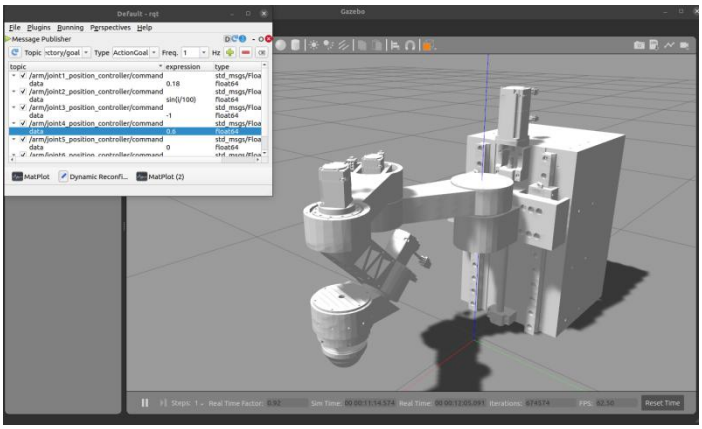
这里采用了力矩接口，并添加了 100 的减速比，这与抛光机械臂实际设计中是保持一致的，在添加 gazebo_ros_control 插件后（也就是定义控制作用域），进行控制器的设置：设置 effort_controllers 控制器下的 JointPositionController 控制器，同时设置相应的 pid 参数，在这通过 rqt_gui 实现参数的调节：



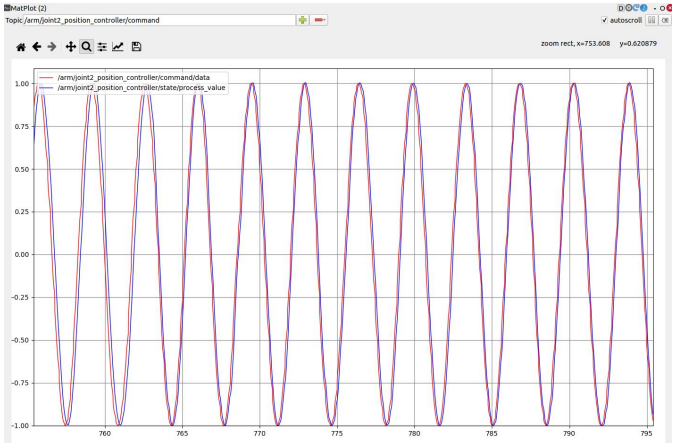
可以通过每个节点的 ros_control 主题进行发布消息，如上图所示，一开始的 gazebo 仿真中模型会随意摆动，这是因为 pid 参数不当（本模型中是因为参数过小）导致的，因此需要进行参数调整（通过 rqt_gui 中的动态配置设置，直到机械臂不再产生抖动为止）：



pid 参数调整完成后，便发布消息，可以通过 rostopic list 查询并发布消息：



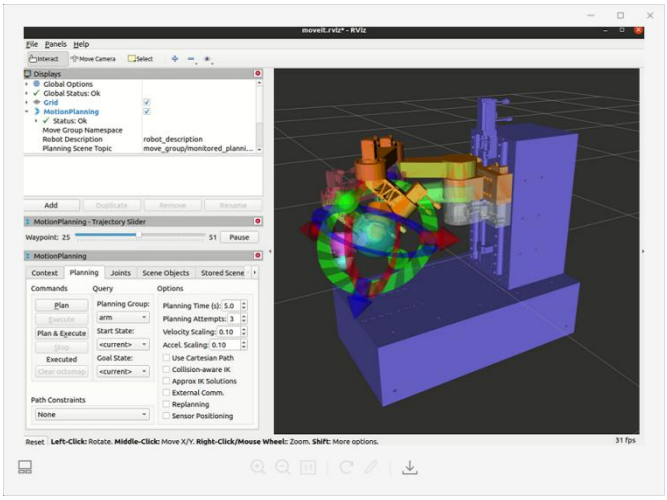
可以通过 rqt 的 plot 找到相应关节主题并绘图，如下图展示的是关节一的相关角度曲线，其中一条线表示命令指令，一个是实际执行的指令值：



上图左部分是初始设置的 pid 参数，可以看到有严重的滞后性，而右部分则是调整后的 pid 参数，可以看到得到的实际运动值和指令值误差降低很多，本项目用的是松下伺服电机，需要通过调节刚性让 pid 参数获得动态调整的不同能力，因此可以通过此仿真获得初始刚性的设置以及 pid 参数的大体情况。

3.Moveit！规划

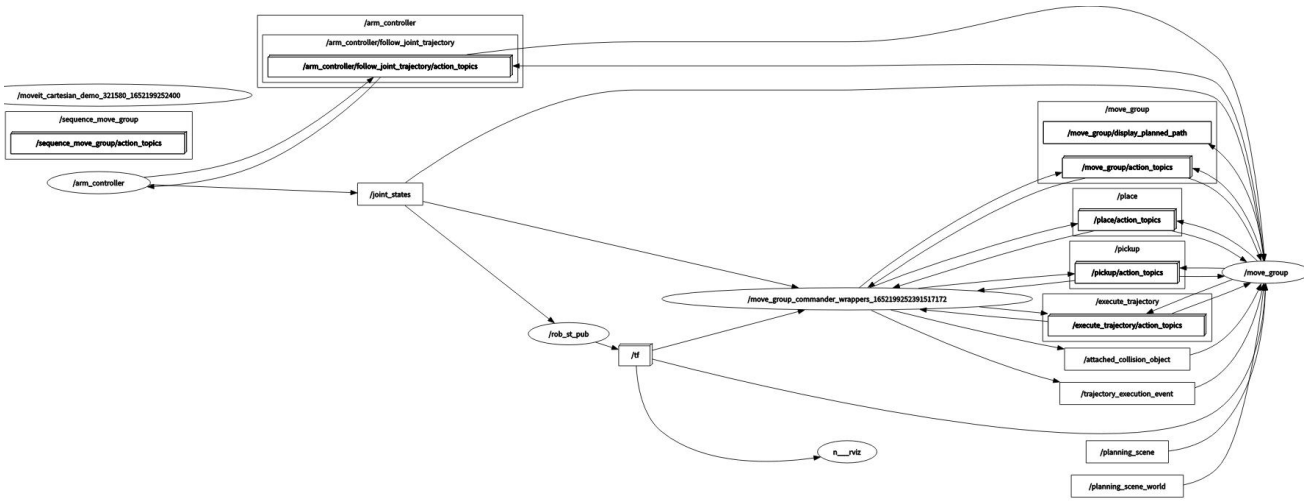
Moveit 的核心是 move_group 组件，负责了所有功能包的串联，我们可以通过相关 launch 文件查看所包含的一系列功能包例如 OMPL 等，关于其配置就不再过多赘述，由于本项目采用的是运动控制卡，运动控制卡已经实现微插补，输入相关角度和时间即可进行关节下的自主规划，因此可以通过 moveit 去模拟控制卡，一方面可以检验笛卡尔空间下的规划是否正确，一方面可以检验运动学逆解是否正确。下图是配置完成后打开 demo 后的展示效果，可以实现拖拽规划。



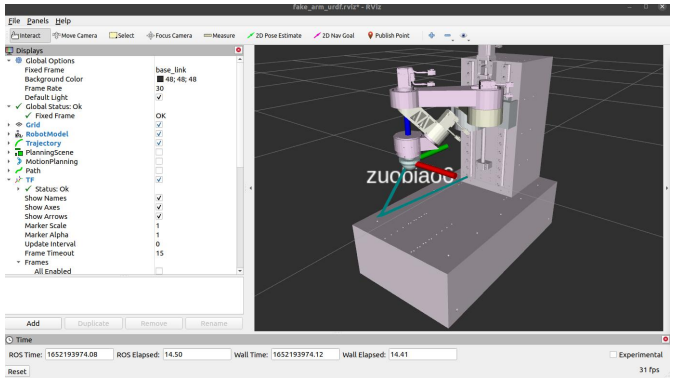
3.1 笛卡尔空间规划

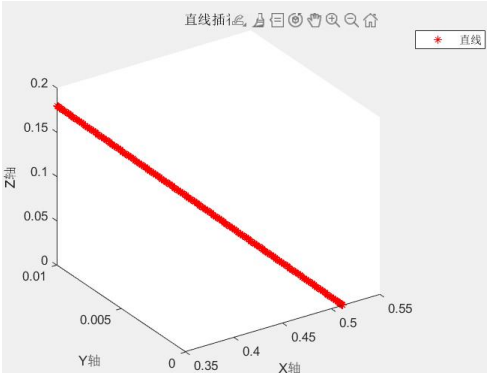
配置好 moveit 的相关参数后，可以看到生成的 srdf 文件里包含了规定的一些姿态组，本文借助 gazebo 可以仿真动力学的特性实现动力学仿真，通过 Rviz 实现规划算法的验证和运动学正逆解的验证，下面先介绍直接使用集成的笛卡尔空间规划相关方法，在前文得到的直线和圆弧插补等曲线在此进行笛卡尔空间下的规划验证。

实现笛卡尔空间规划可以直接利用 Moveit!的相关 python API，代码中本项目同时实现了工作空间下的运动规划和带约束下的笛卡尔空间规划，用两种方式都可以实现对机械臂运动的验证，通过对笛卡尔路径规划的 API——compute_cartesian_path()实现对路径点的录入以及是否考虑避障等。运行中的节点分布图如下所示：

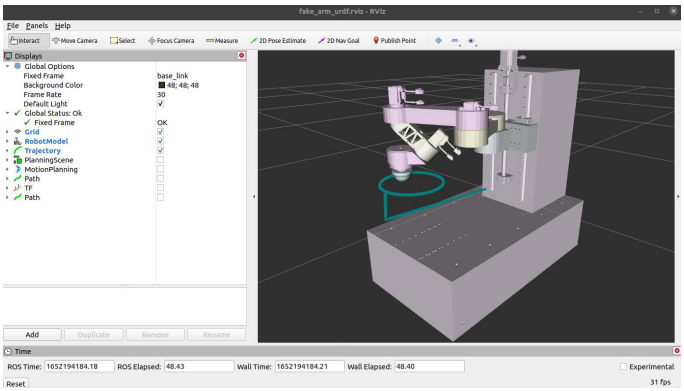


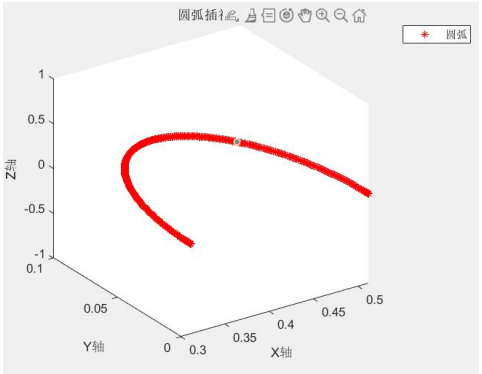
节点图中可以很清晰的看到 move_group 的核心性，下面进行规划。首先根据前文的插补原理，速度规划采用 T 型运动曲线进行规划，规定固定的插补周期得到了指定路径的插补点。定义的起点是[0.512,0,0]，终点是[0.35,0.01,0.18]，使用 MoveGroupCommander 获得机械臂 arm 控制对象组，再通过 compute_cartesian_path 指令定义规划的笛卡尔空间接口，输入本地进行的直线插补数据集，最后得到规划曲线并与 matlab 中对比：



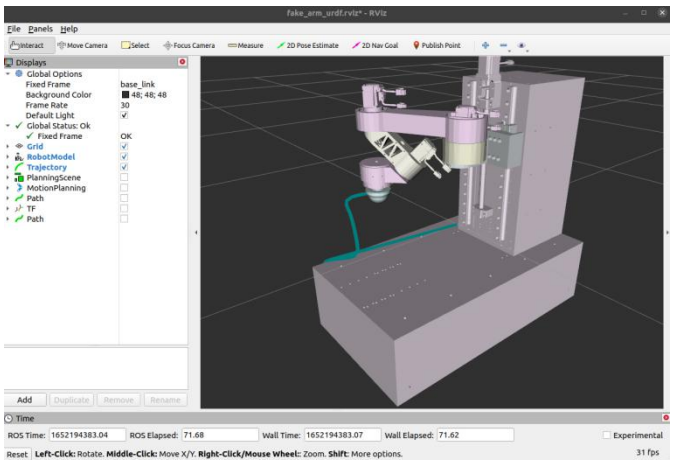


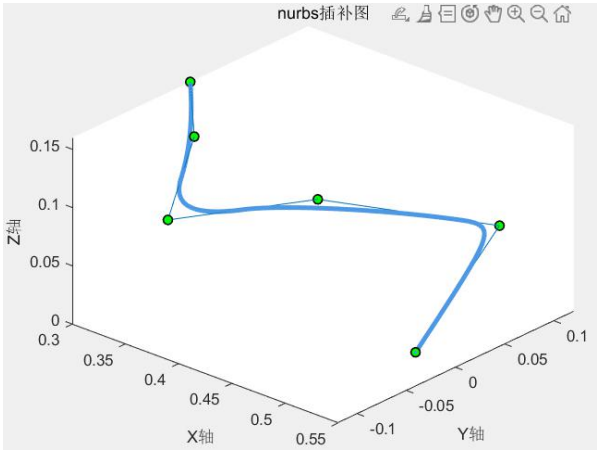
根据前文所述圆插补原理生成一系列的圆弧轨迹曲线：





以及生成的 nurbs 插补曲线及对比曲线：

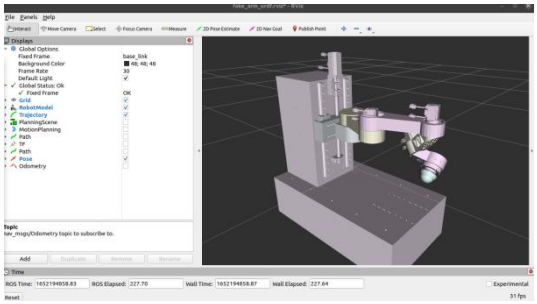




3.2 运动学算法验证

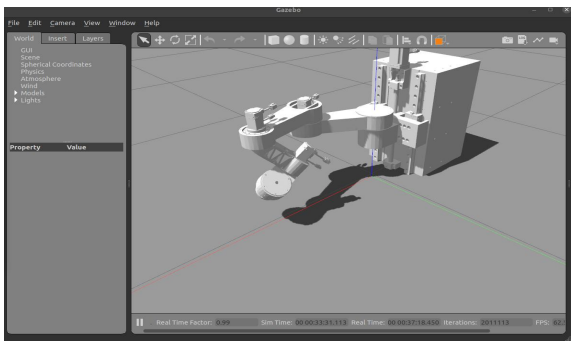
抛光机械臂所采用的运动学算法也就是需要写入 PC 和控制卡通信的有上述三种插补算法以及运动学正逆解算法等，为了更好的模拟仿真同时验证正逆解的正确性，本文通过 ArbotiX 控制器仿真实际的控制器，也就是对每一个关节进行了电机和驱动器的模拟，并通过 action 消息类型和相关插件进行 moveit 和 ArbotiX 的通讯，本文只需要规定 action 的消息队列即可，也就是模拟了控制卡的 PVT 模式路径点输入。

下图所示为起点为[0.48,0.06,0.1]，终点为[0.48,-0.06,0.1]的规划后的轨迹，由于抛光所需要的轨迹是逐层向外的框型轨迹，因此在这进行了多段轨迹规划，下图所示为第一段直线运动，可以看到刀具轴的运动方向保持 Y 轴的负方向：

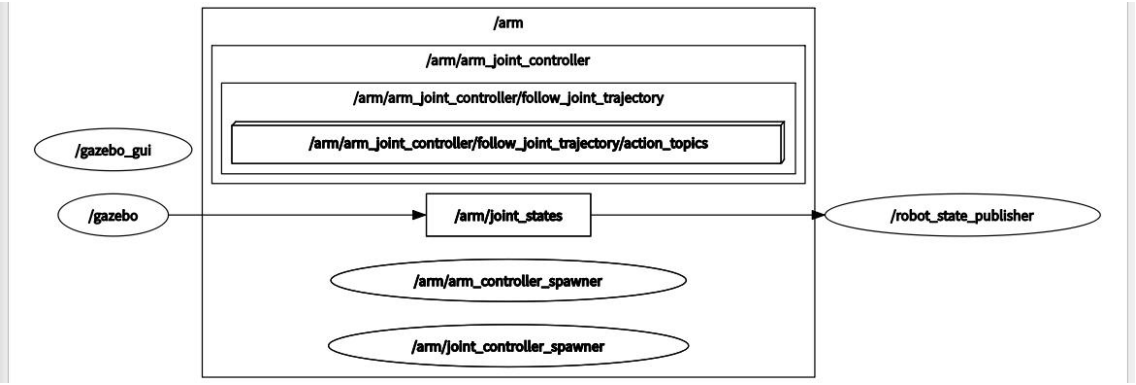


当然除了在 Rviz 中实现对算法的验证，也可以直接在 gazebo 中获得动力学仿真结果，将模型加载进 gazebo 里面后首先通过 joint_state_controller/JointStateController 发布节点信息，同时定义与执行器相连的 effort_controllers/JointTrajectoryController 控制器，这种控制器也是最接近控制卡高级运动功能的控制器模式，本文对每一个关节用的传动装置是 EffortJointInterface，这里与 Rviz 里面不同，Rviz 里面直接采用的位置执行器，最后定义 FollowJointTrajectory 的 action 消息与 ArbotiX 连接从而实现对执行器的控制即可开始仿真。

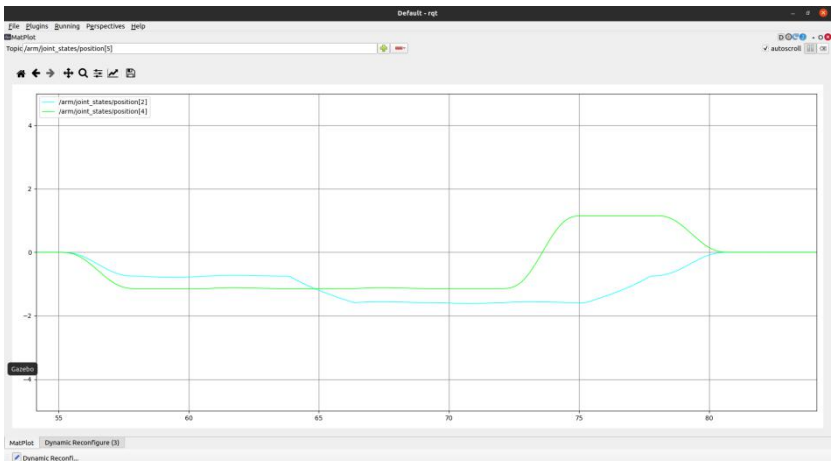
先利用前文调节好的 pid 参数保证防抖，直接写入相关 yaml 文件，下图是正在仿真中的运动状态：



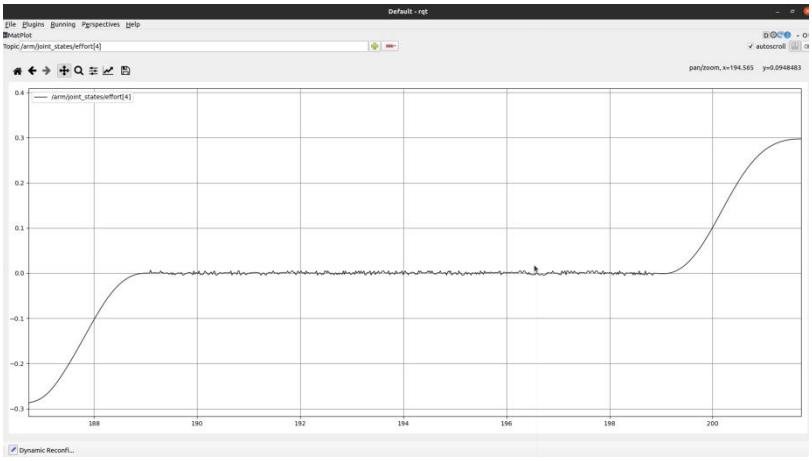
可以查看相关运行节点如下，可以看到并没有笛卡尔空间规划的复杂性，是因为本文直接通过 Arbotix 控制、通过相关 action 类型发布消息，并没有与 moveit 相连，从之前的节点图可以看到是通过配置了 FollowJointTrajectoryAction 接口的相关插件进行连接。



本文先通过 rostopic echo 命令获得 joint_states 的消息类型和数据格式，然后通过 rqtplot 指令获得抛光机械臂的每一个关节的状态信息，如下所示：



关节 2、4 的位置图



关节 3 的力矩图

综上，运动算法验证是正确的，可以写入本项目的软件中具体实现。