## 配置夹具的moveit：

前置工作：

修改夹具的urdf

夹具urdf位于urdf\_source目录下

urdf\_source

└── grippers

├── Allegro

├── Barrett

├── fetch\_gripper

├── franka\_panda

├── gripper\_pyb\_info.pk

├── h5\_hand

├── HumanHand

├── jaco\_robot

├── Manuals

├── panda\_arm.urdf

├── robotiq\_2finger

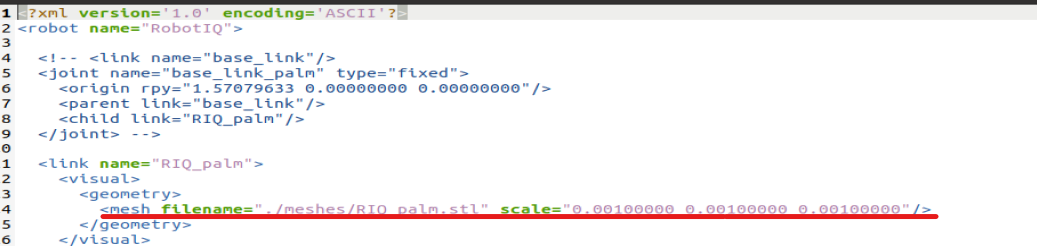
├── robotiq\_3finger

├── sawyer

├── shadow\_hand

└── wsg\_50

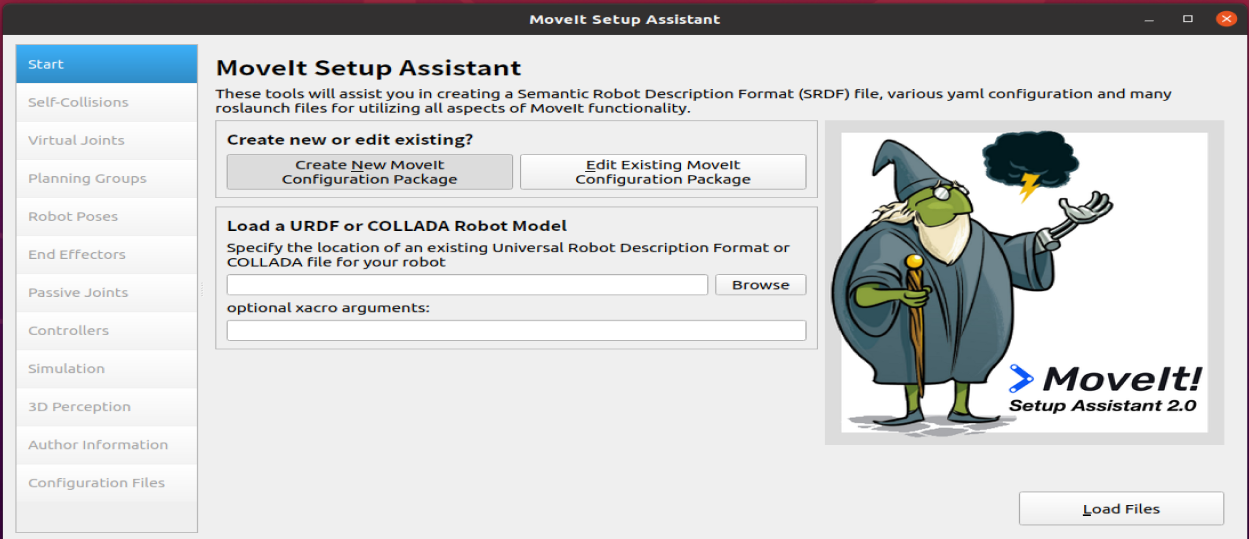
也可于<https://github.com/IRVLUTD/isaac_sim_grasping>下载



对所有./meshes/...修改为当前包名，如当前包名为barrett\_description，则应修改为package://barrett\_description/meshes/...

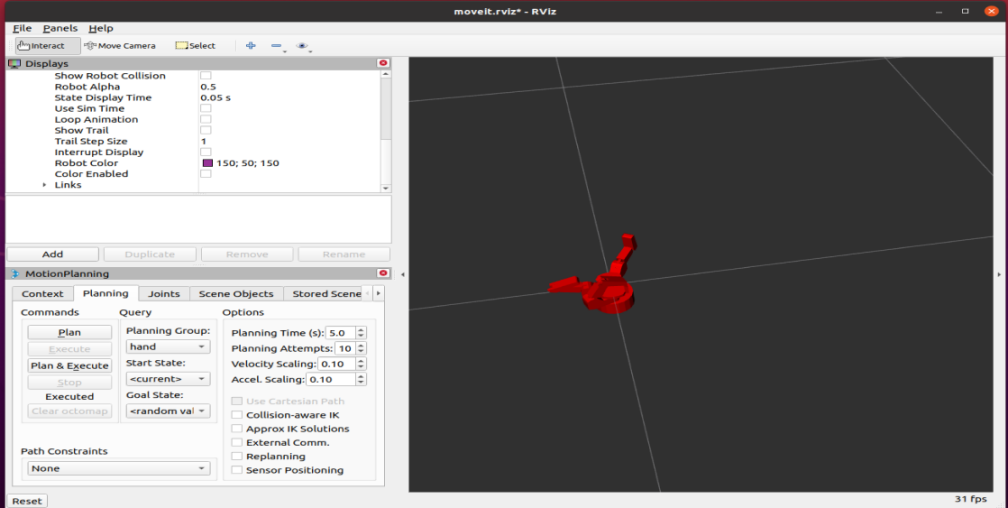
然后在工作区下编译catkin\_make并且source devel/setup.sh

在ros工作区下roslaunch moveit\_setup\_assistant setup\_assistant.launch



后续具体配置见[MoveIt Setup Assistant](https://moveit.github.io/moveit_tutorials/doc/setup_assistant/setup_assistant_tutorial.html)

配置生成新的moveit包后运行demo.launch证明配置成功



在MotionPlanning中可对关节进行控制，也可以在moveit\_assistant的Planning Group中选择配置moveit的不同规划组。

## Moveit连接Isaac sim：

使用上面新生成的moveit配置控制仿真环境中的机械臂

1. franka\_isaac\_execution.launch

把panda\_moveit\_config改成由moveit\_assistant生成的新包名，如new\_robot或者only\_arm

<include file="$(find **panda\_moveit\_config**)/launch/planning\_context.launch">

<arg name="load\_robot\_description" value="true"/>

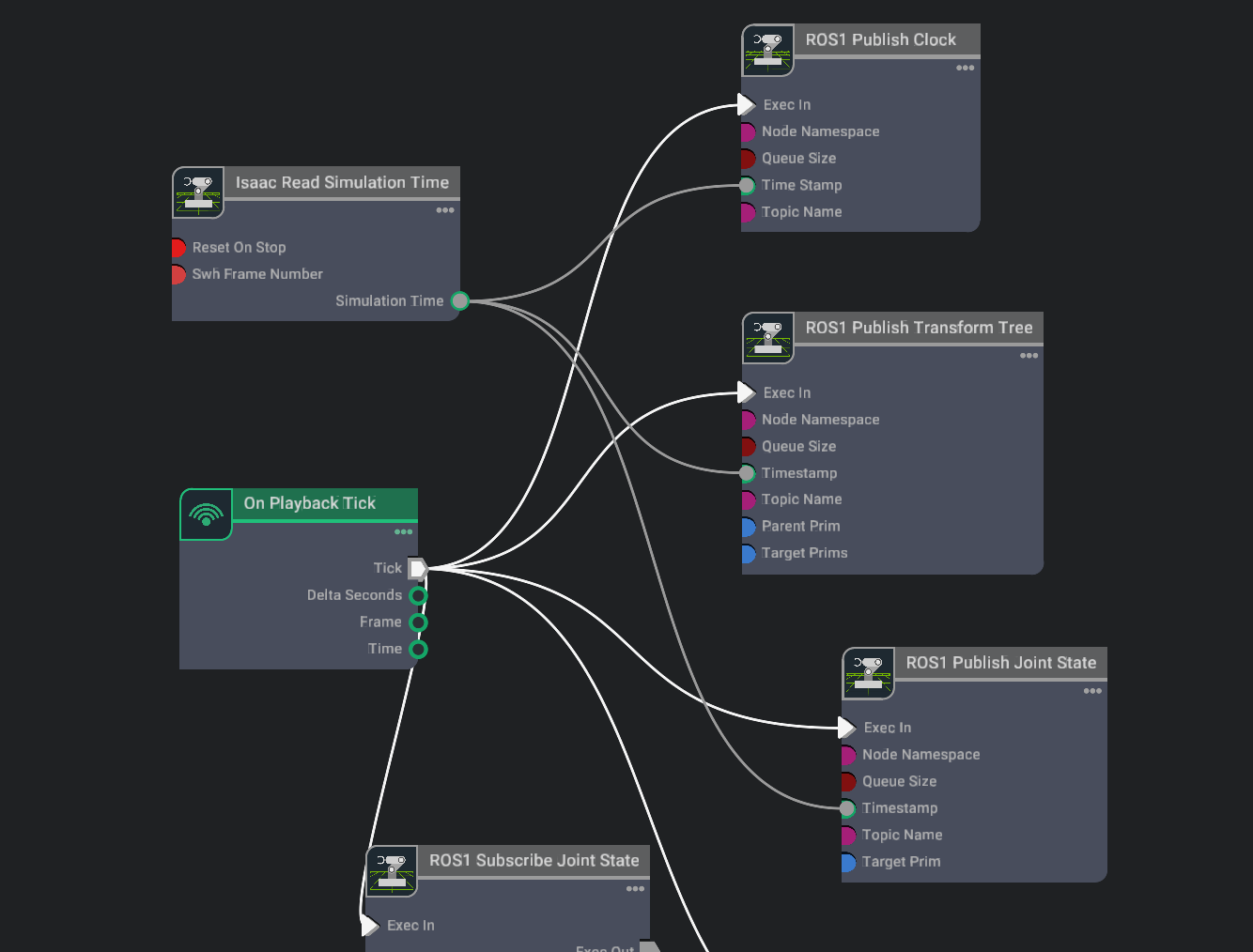
</include>

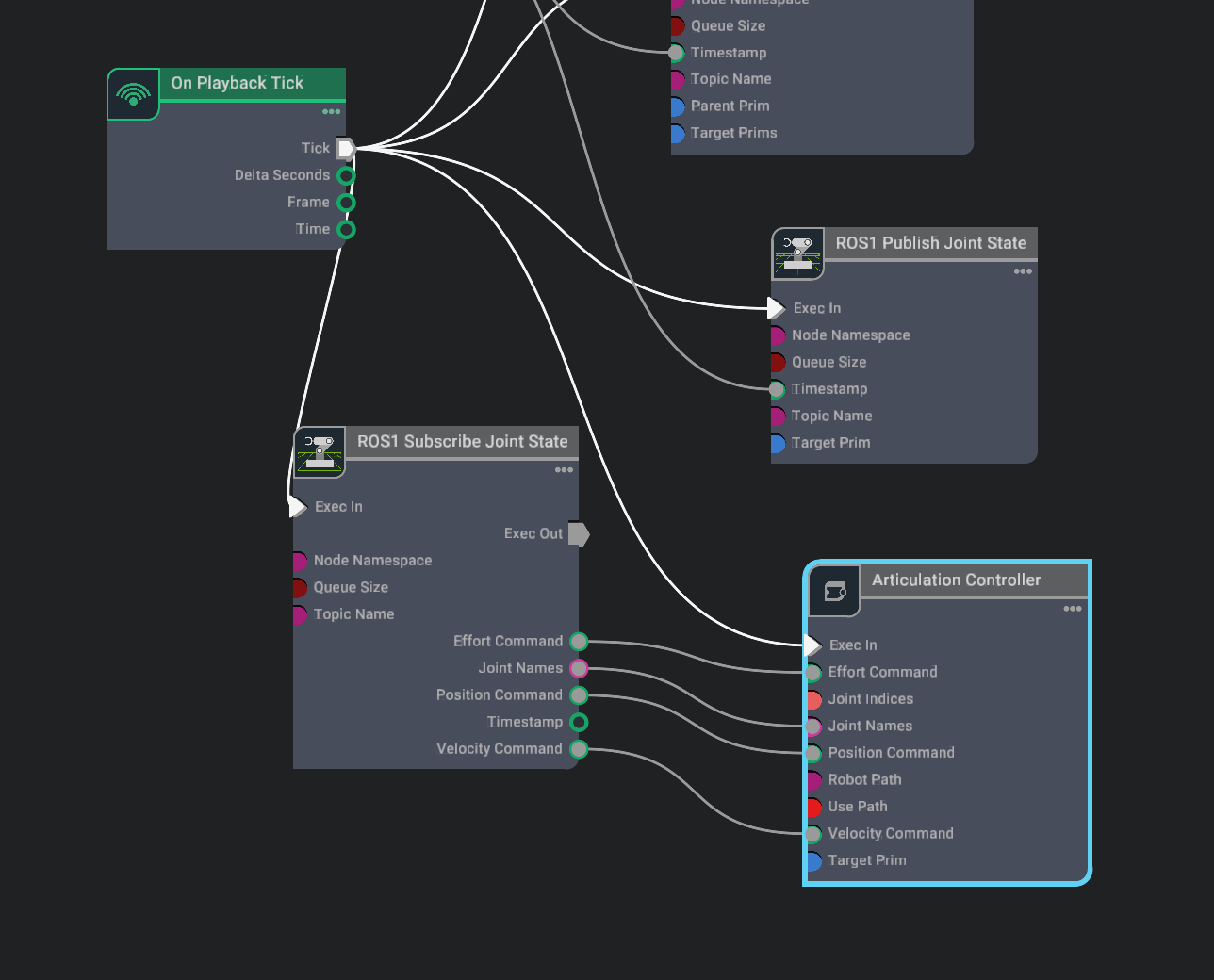
<include file="$(find **panda\_moveit\_config**)/launch/move\_group.launch">

先启动仿真环境，再启动isaac\_moveit文件夹下的launch文件

1. 配置ActionGraph

在Isaac sim中配置Actiongraph来发布ros节点。





设置Targetprim：

ROS1 Publish Transform Tree:.../base\_link

ROS1 Publish Joint State:.../base\_link

Articulation Controller:.../base\_link

配置成功后点击play，在终端中输入rosrun rqt\_tf\_tree rqt\_tf\_tree可以看到如下tf图：

输入rostopic echo /joint\_states

---

header:

seq: 1997

stamp:

secs: 1722500525

nsecs: 907341718

frame\_id: ''

name:

- hand\_joint

- left\_wheel\_joint

- right\_wheel\_joint

- servo\_link1\_joint

- servo\_link2\_joint

- L\_joint1

- L\_joint2

- L\_joint3

- L\_joint4

- L\_joint5

- L\_joint6

- L\_drive\_joint

- L\_left\_finger\_joint

- L\_left\_inner\_knuckle\_joint

- L\_right\_outer\_knuckle\_joint

- L\_right\_finger\_joint

- L\_right\_inner\_knuckle\_joint

- R\_joint1

- R\_joint2

- R\_joint3

- R\_joint4

- R\_joint5

- R\_joint6

- R\_drive\_joint

- R\_left\_finger\_joint

- R\_left\_inner\_knuckle\_joint

- R\_right\_outer\_knuckle\_joint

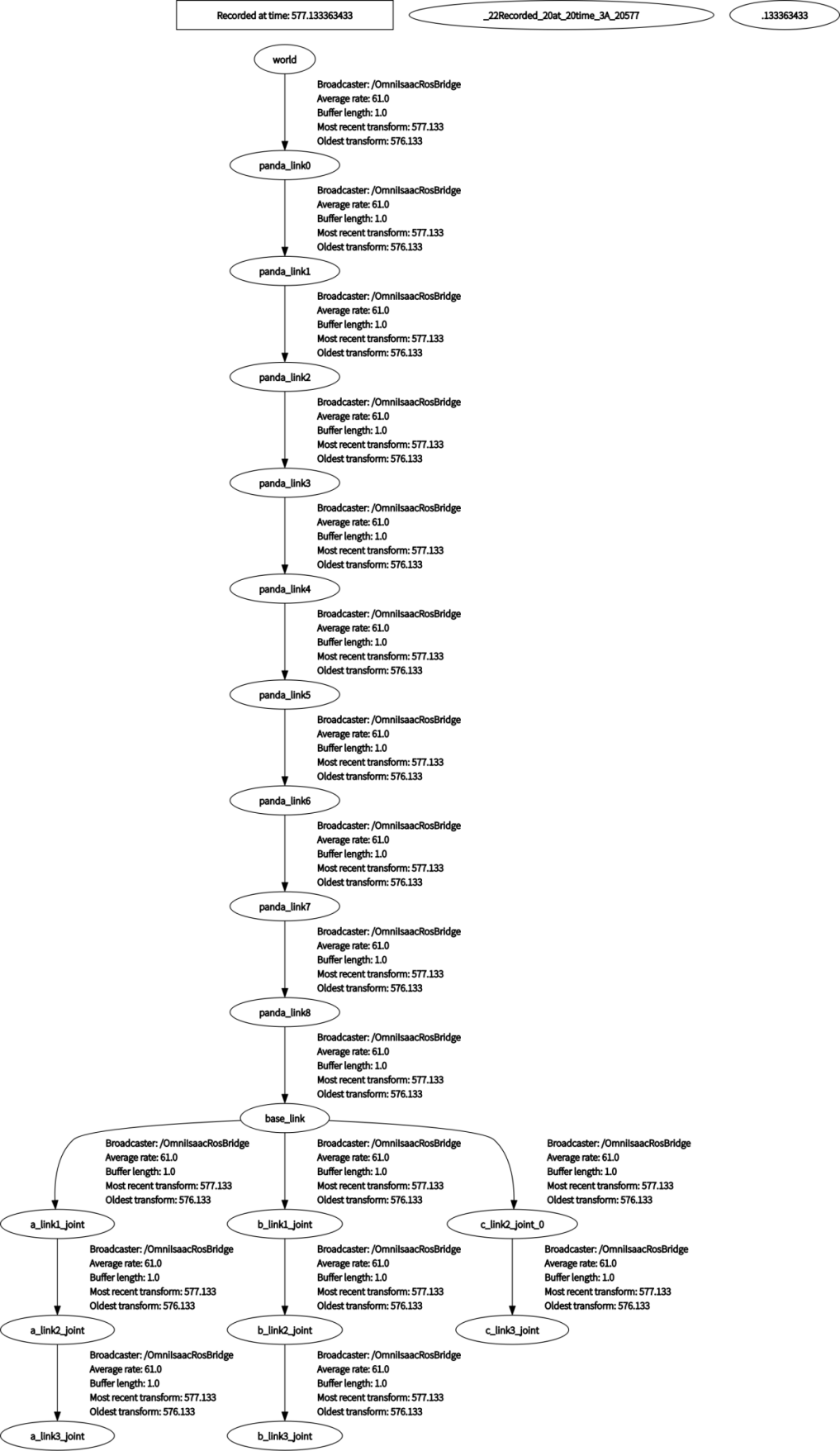
- R\_right\_finger\_joint

- R\_right\_inner\_knuckle\_joint

position: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.4475296347081661, 0.4475296347081661, 0.4475296347081661, 0.4475296347081661, 0.4475296347081661, 0.4475296347081661, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

velocity: []

effort: []



1. Actiongraph代码配置

# Creating a action graph with ROS component nodes

try:

og.Controller.edit(

{"graph\_path": "/ActionGraph", "evaluator\_name": "execution"},

{

og.Controller.Keys.CREATE\_NODES: [ # 创建节点

("OnImpulseEvent", "omni.graph.action.OnImpulseEvent"),

("ReadSimTime", "omni.isaac.core\_nodes.IsaacReadSimulationTime"),

("PublishJointState", "omni.isaac.ros\_bridge.ROS1PublishJointState"),

("SubscribeJointState", "omni.isaac.ros\_bridge.ROS1SubscribeJointState"),

("ArticulationController", "omni.isaac.core\_nodes.IsaacArticulationController"),

("PublishTF", "omni.isaac.ros\_bridge.ROS1PublishTransformTree"),

("PublishClock", "omni.isaac.ros\_bridge.ROS1PublishClock"),

],

og.Controller.Keys.CONNECT: [ # 节点连接

("OnImpulseEvent.outputs:execOut", "PublishJointState.inputs:execIn"),

("OnImpulseEvent.outputs:execOut", "SubscribeJointState.inputs:execIn"),

("OnImpulseEvent.outputs:execOut", "PublishTF.inputs:execIn"),

("OnImpulseEvent.outputs:execOut", "PublishClock.inputs:execIn"),

("OnImpulseEvent.outputs:execOut", "ArticulationController.inputs:execIn"),

("ReadSimTime.outputs:simulationTime", "PublishJointState.inputs:timeStamp"),

("ReadSimTime.outputs:simulationTime", "PublishClock.inputs:timeStamp"),

("ReadSimTime.outputs:simulationTime", "PublishTF.inputs:timeStamp"),

("SubscribeJointState.outputs:jointNames", "ArticulationController.inputs:jointNames"),

("SubscribeJointState.outputs:positionCommand", "ArticulationController.inputs:positionCommand"),

("SubscribeJointState.outputs:velocityCommand", "ArticulationController.inputs:velocityCommand"),

("SubscribeJointState.outputs:effortCommand", "ArticulationController.inputs:effortCommand"),

],

og.Controller.Keys.SET\_VALUES: [ # 设置节点的值

# Setting the /Franka target prim to Articulation Controller node

("ArticulationController.inputs:usePath", True),

("ArticulationController.inputs:robotPath", FRANKA\_STAGE\_PATH),

("PublishJointState.inputs:targetPrim", [usdrt.Sdf.Path(FRANKA\_STAGE\_PATH)]),

("PublishTF.inputs:targetPrims", [usdrt.Sdf.Path(FRANKA\_STAGE\_PATH)]),

],

},

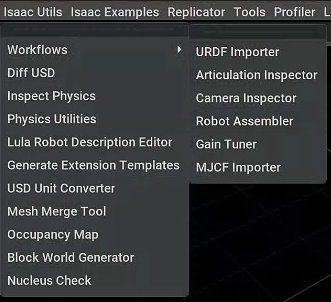
)

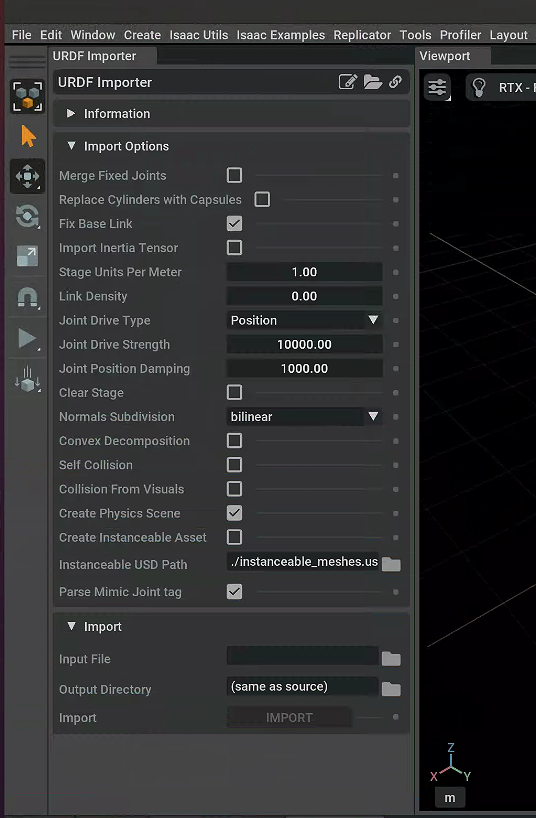
更改蓝色部分为自己的路径

注意：选择targetPrim时要绑定到物体的base\_link上，因为2023.1.1版本的Isaac sim把ArticulationAPI绑定到了base\_link上，而不是物体本身。

1. Urdf->USD:

在Isaac sim的左上角打开Isaac Unit→Workflows→URDF Importer





如果没有urdf只有xacro文件可以使用rosrun xacro xacro yourfile.xacro > filename.urdf进行转化。

1. 机械臂和夹具在仿真环境中拼接：

两种方法：

1. Assemble
2. 手动添加Joint

把夹具移动到机械臂末端合适的位置，在把机械臂末端的link和夹具的base\_link之间添加上joint。Body0是父link，body1是子link

## Ir100：

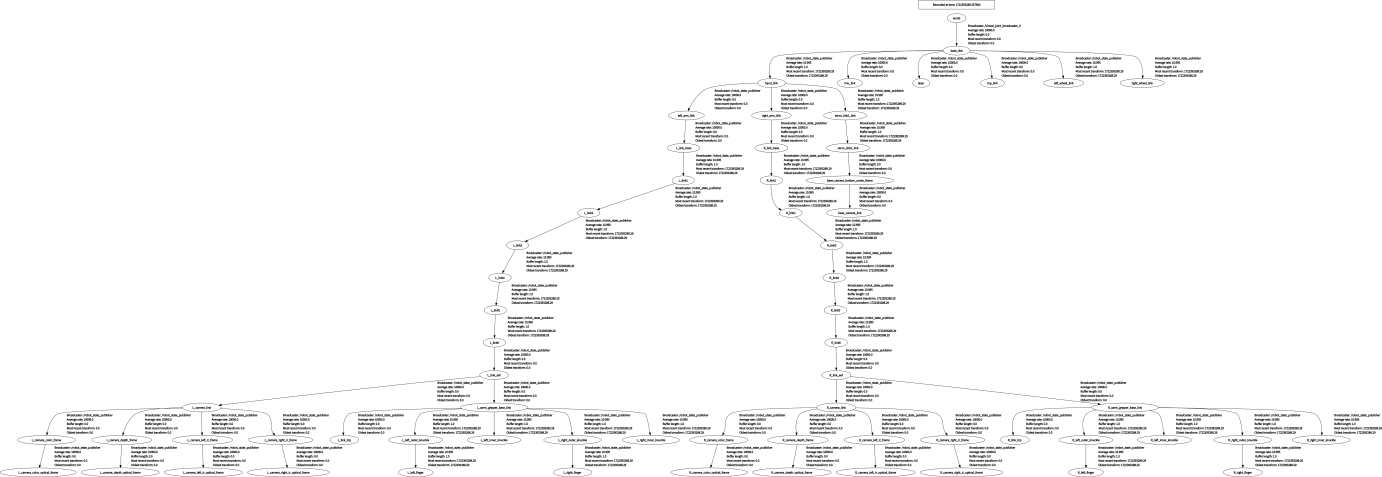
描述文件：



配置好的moveit



Ir100的tf树：



Ir100的USD：

~/shiys/new\_robot.usd

目前一直有问题，还在调。。。

可能的解决方法：

1. 不使用仿真环境发布/tf，用demo.launch发布，接着启动isaac\_moveit目录下的ir100.launch
2. 修改USD，在xacro文件中添加一个真正的base\_link（不含有visual和collision的，用于在仿真环境中添加Articulation Root）→Urdf→USD
3. 。。。

## 一些烂七八糟的：

手改urdf拼接panda\_arm和allergo\_hand：

⬅这是moveit。

⬅这是机械臂和手拼接后的xacro文件，代码里写了注解。

Roslaunch panda\_allergo\_v2 demo.launch

