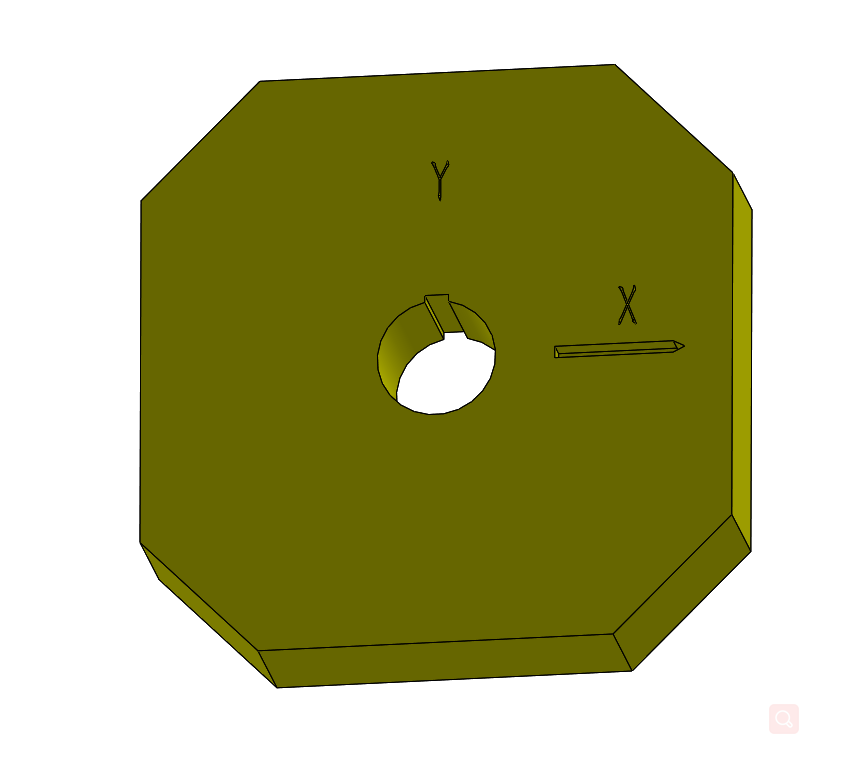
作者被ros的ros\_control控制器和robot\_hw，还有剖gazebo仿真环境等一些列奇奇怪怪的东西折磨的不成人形，所以今天痛定思痛，从0开始一步一步配置这些东西。

## 制图

不管怎么样，我们都应该先画一个模型，因为这不是重点，所以就很随意的画了一个2自由度的连杆，要注意的是这个不是咱们经常说的二连杆。

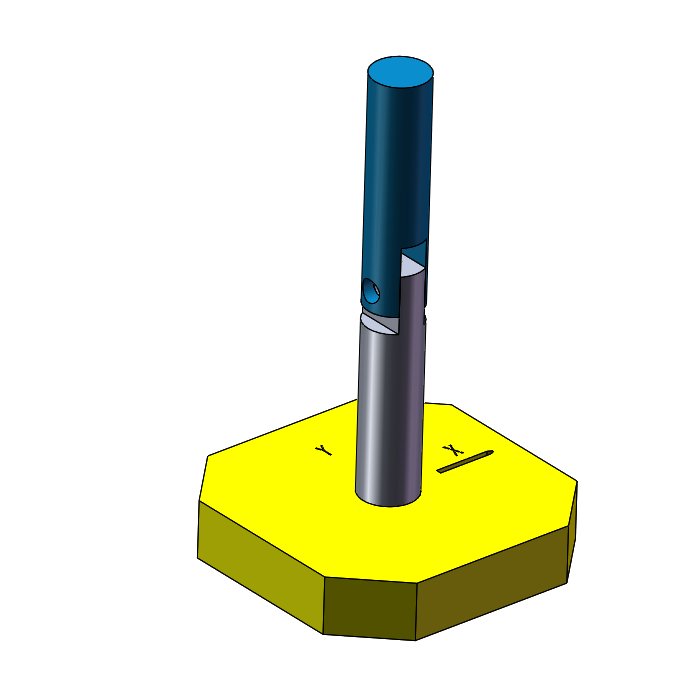
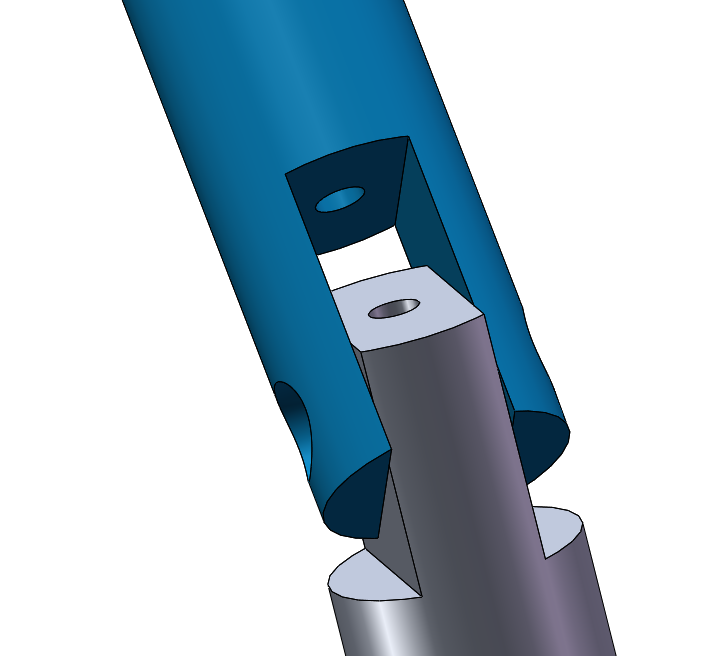


现在这个2自由度杆可以随意拖动，但是我们等下生成urdf的过程中不能让它可以随意拖动。所以需要给它画一些约束才行。



给基座模型标一下xy轴的方向，要不然待会转来转去就晕了。中间的长得像键槽的东西也是随便画来做配合的。

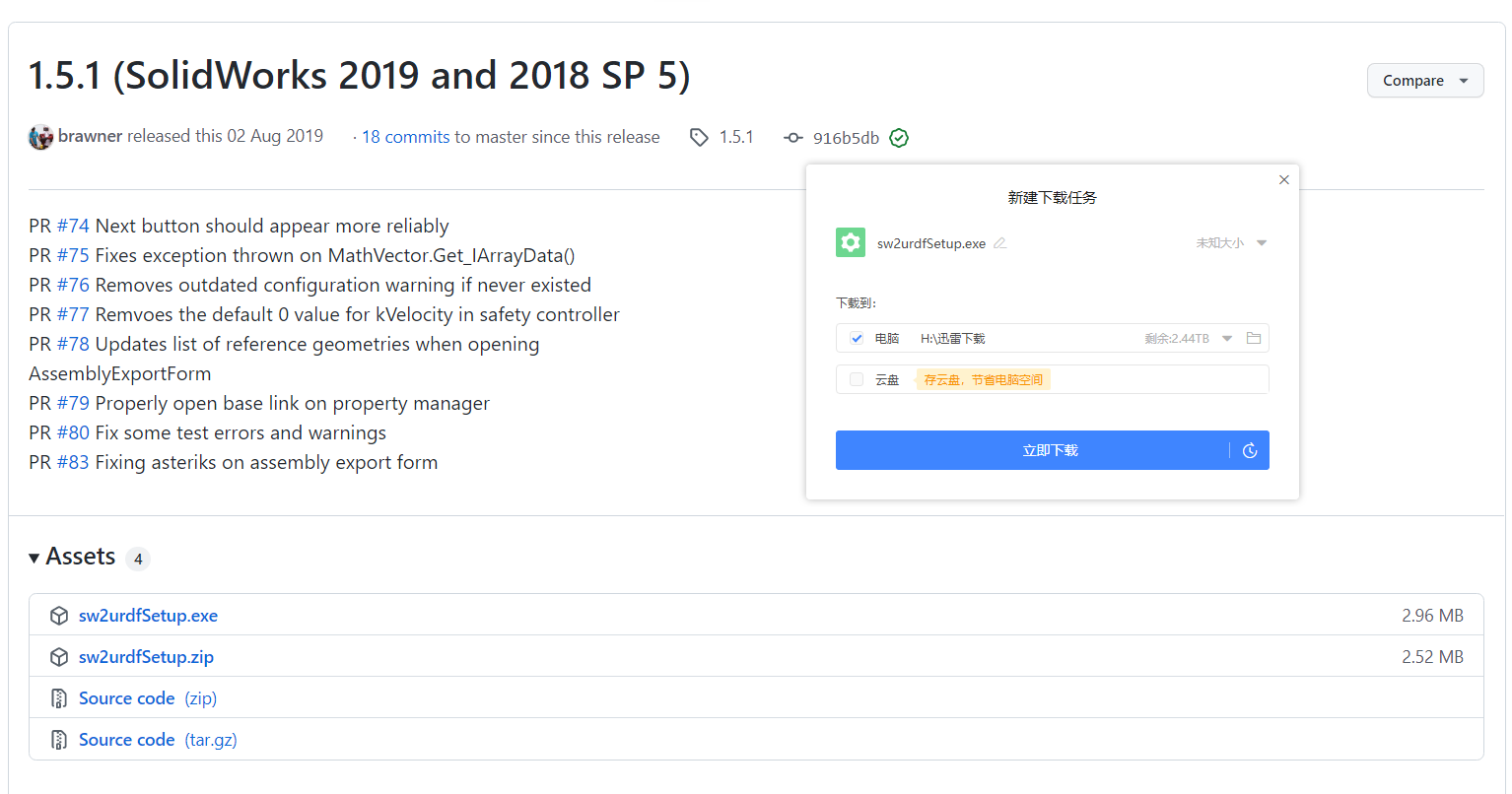
剩下的两根杆也是同样的道理，中间切除一点，我一般是第几个joint就切直径多大的圆柱。如果尺寸大的话一般就个位数是joint的名字，比如第一个joint我一般就切一个21mm或者31mm或者101mm，这样子在配合的过程中就不会乱。



## 配置urdf

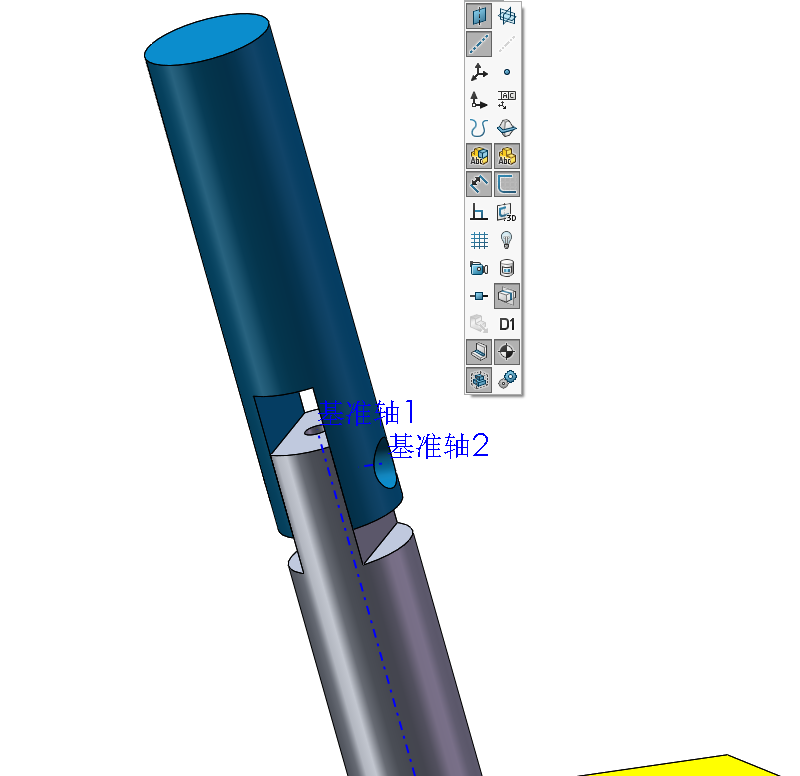
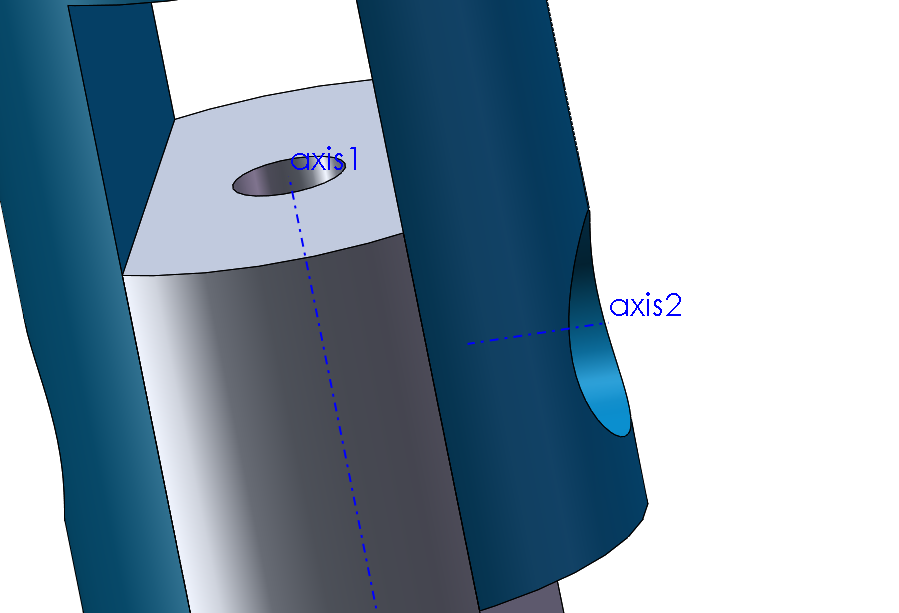
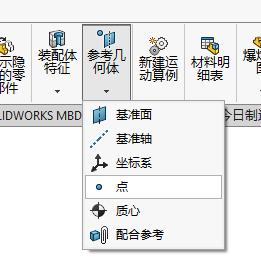
当你配合到机器人的初始状体之后，我们就可以通过urdf插件来生成urdf了。编者发现urdf插件忘记安装了，所以现场装一下。

安装地址：[sw\_urdf\_exporter - ROS Wiki](http://wiki.ros.org/sw_urdf_exporter)

因为我用的是2018版本的，所以选择了1.5.1。下载安装很简单，而且其实软件很小。之前配置URDF的时候，模型比较复杂，上千个零件，天天炸，搞得我一度很崩溃。后来发现是同一个模型在装配体中不能出现两次。举个例子，我有一个螺栓的模型，我如果将这同一个螺栓放到不同的地方，就会有好多个这样的螺栓插到不同的孔里，这是装配体里就会有好多个螺栓，那么这个模型必炸。所以一个简单的办法就是将每一个部件（没有相对运动的装配体）先搞成一个装配体，然后再另存为一个零件，没错，如果你打开装配体，另存为一个零件，它就变成了一个零件。再重新将每个部件装配起来，但是，左轮和右轮要保存为不同的文件中，不能我一个轮子装到左边，然后随配合复制或者直接复制到右边去。这样也会炸。正常的做法应当是复制一份文件，取不同的名字，然后再加进去。**请你认真看这段文字，要不然你的模型必炸！！！！！**

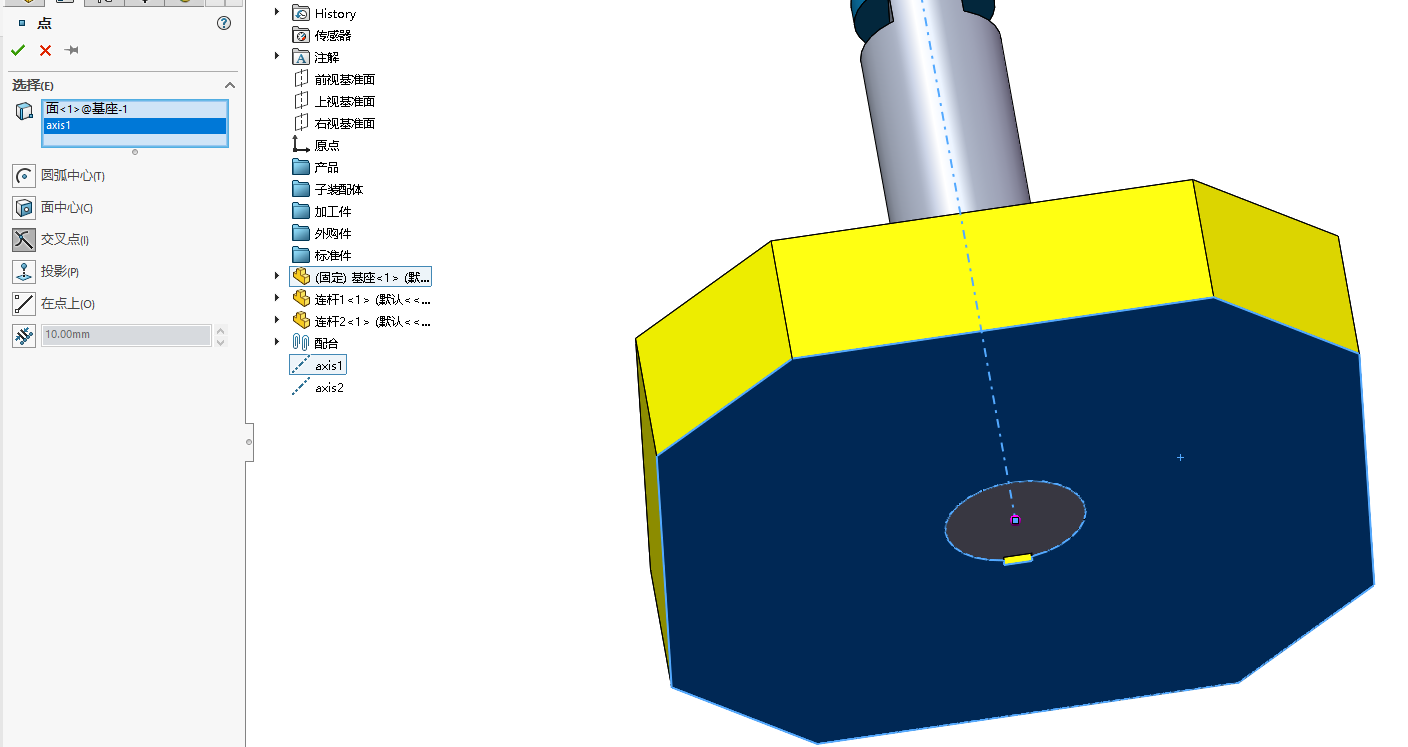
下面开始配置URDF的坐标系，我建议你们使用改进DH法，具体的理论可以自己去搜索。在改进DH法建系后，计算雅可比矩阵和动力学方程的时候就表示向量就不用从坐标系原点向后指了。标准DH法后面的事情很容易让人晕掉。

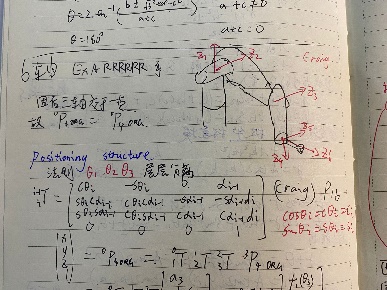
### 建立基准轴

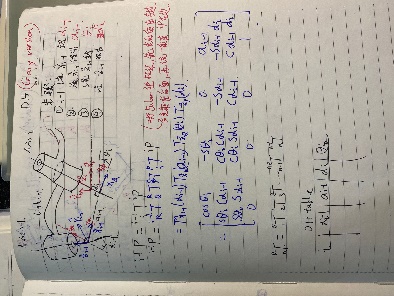
  

将每一个旋转轴标识出来。记得改成英文，要不然你的模型必炸！！！

然后就是要插入每个坐标系原点的点。

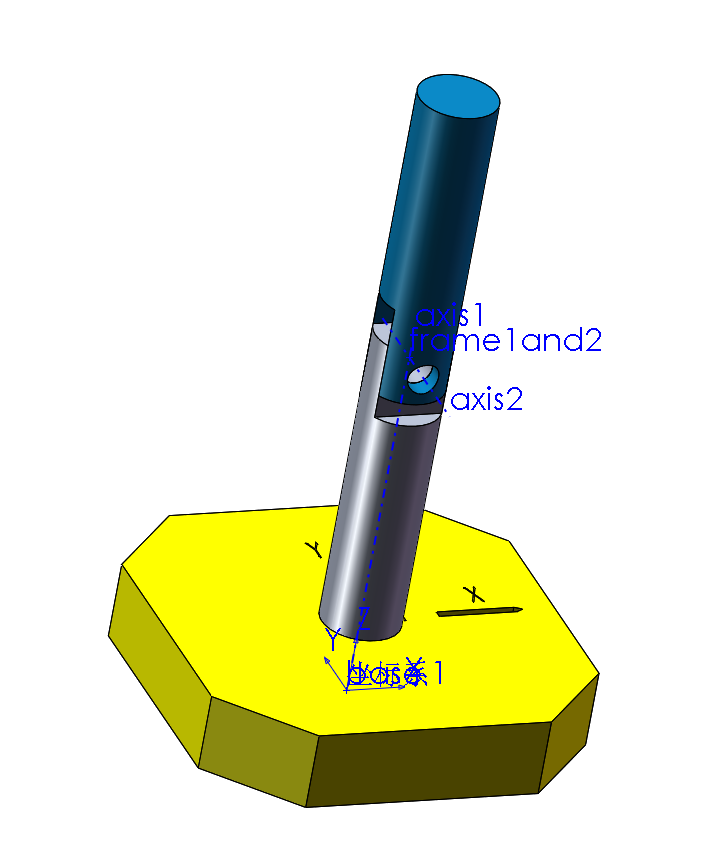
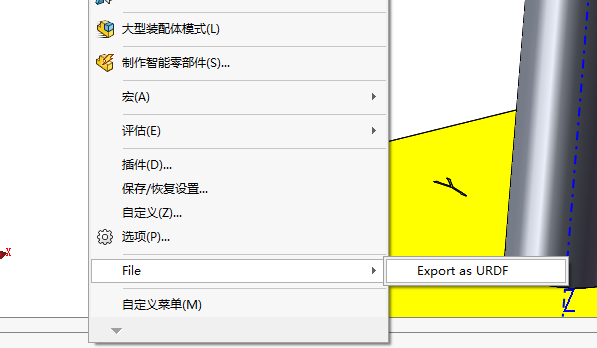
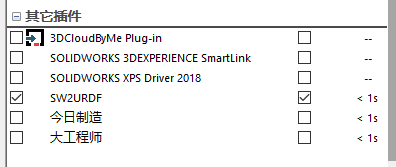
建系方法大概就是这样：





字非常的丑，大家凑活看一下。建议自行百度寻找。

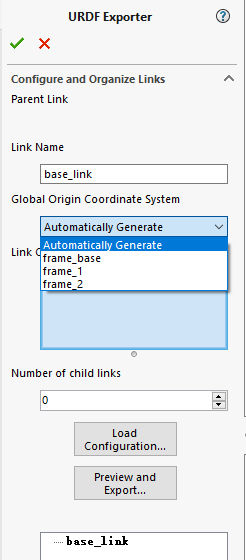
建系的过程中感觉好像哪里不太对，就把整个杆拧了90度，如图所示：

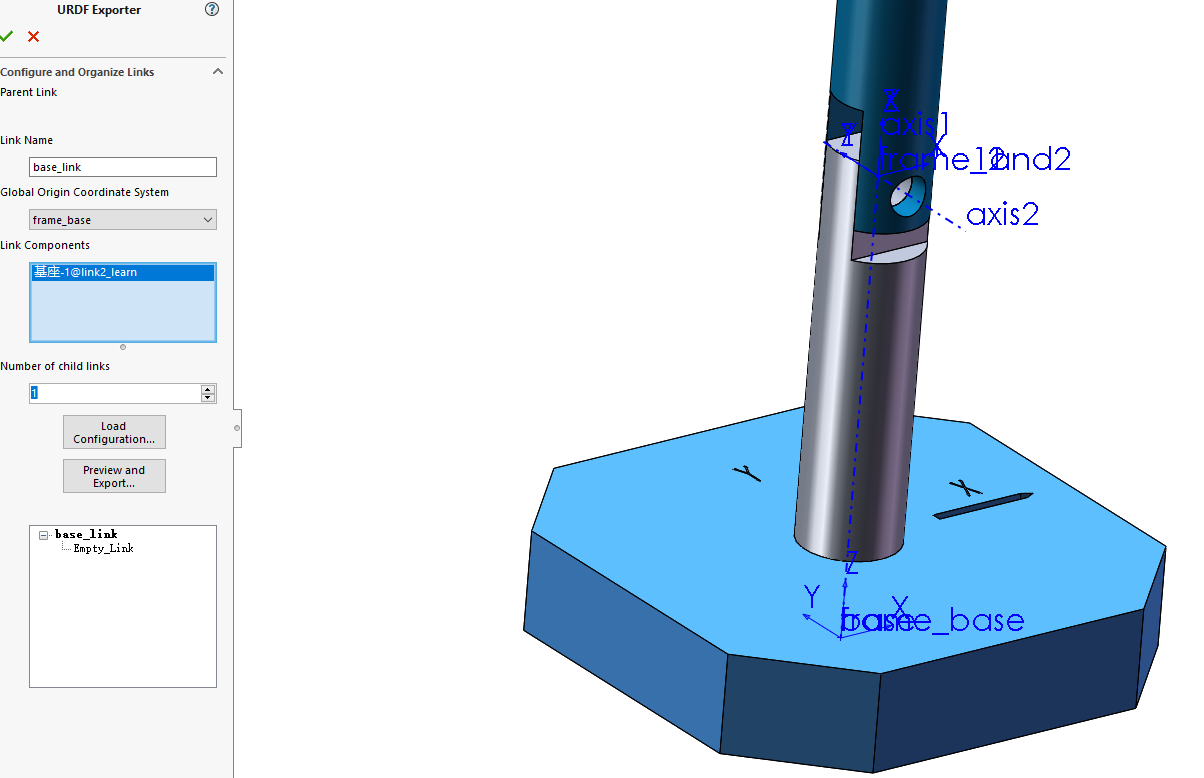
  

当然这个过程不影响操作也不影响计算，只不过是一种标准罢了。

当笔者准备开始配置的时候，才发现urdf插件竟然启动不了！！！！百度了一下发现是安装了今日制造等插件，这些插件会自动关掉其他插件。卸载了就好了

首先配置base\_link：base\_link的坐标系选择frame\_base，模型选择基座，子link数量为1

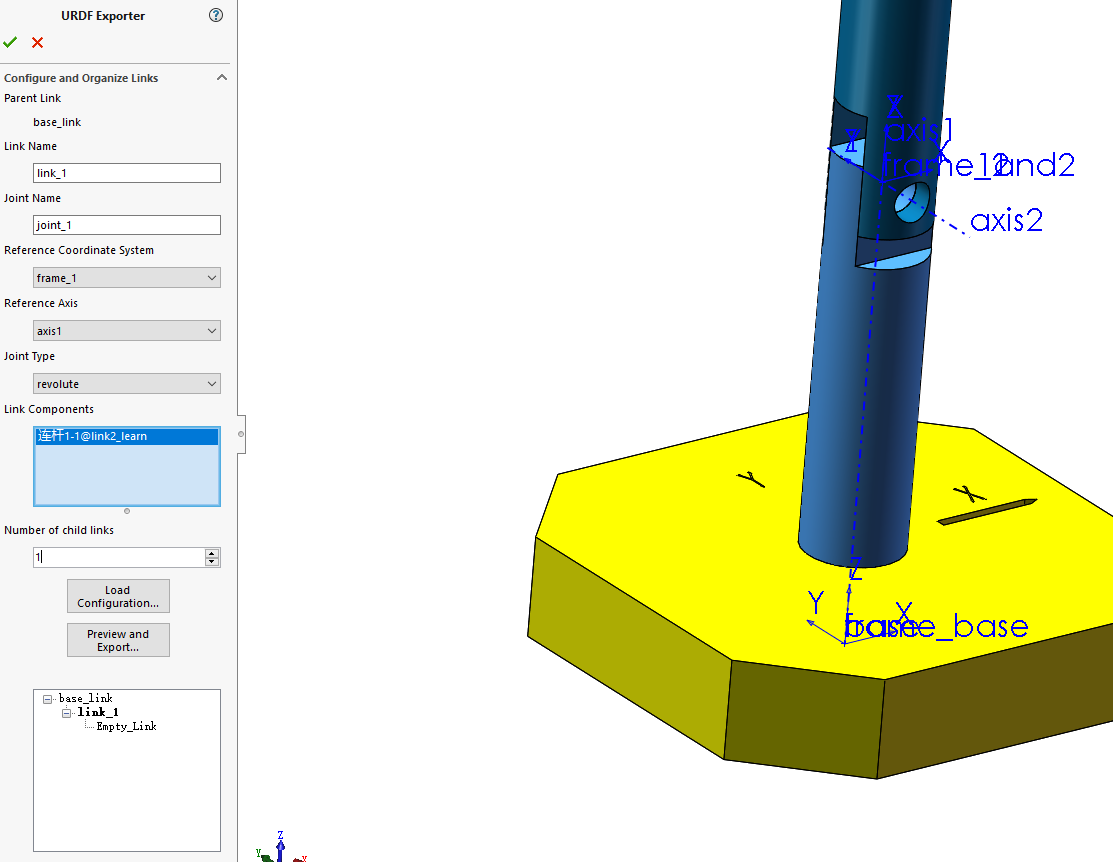
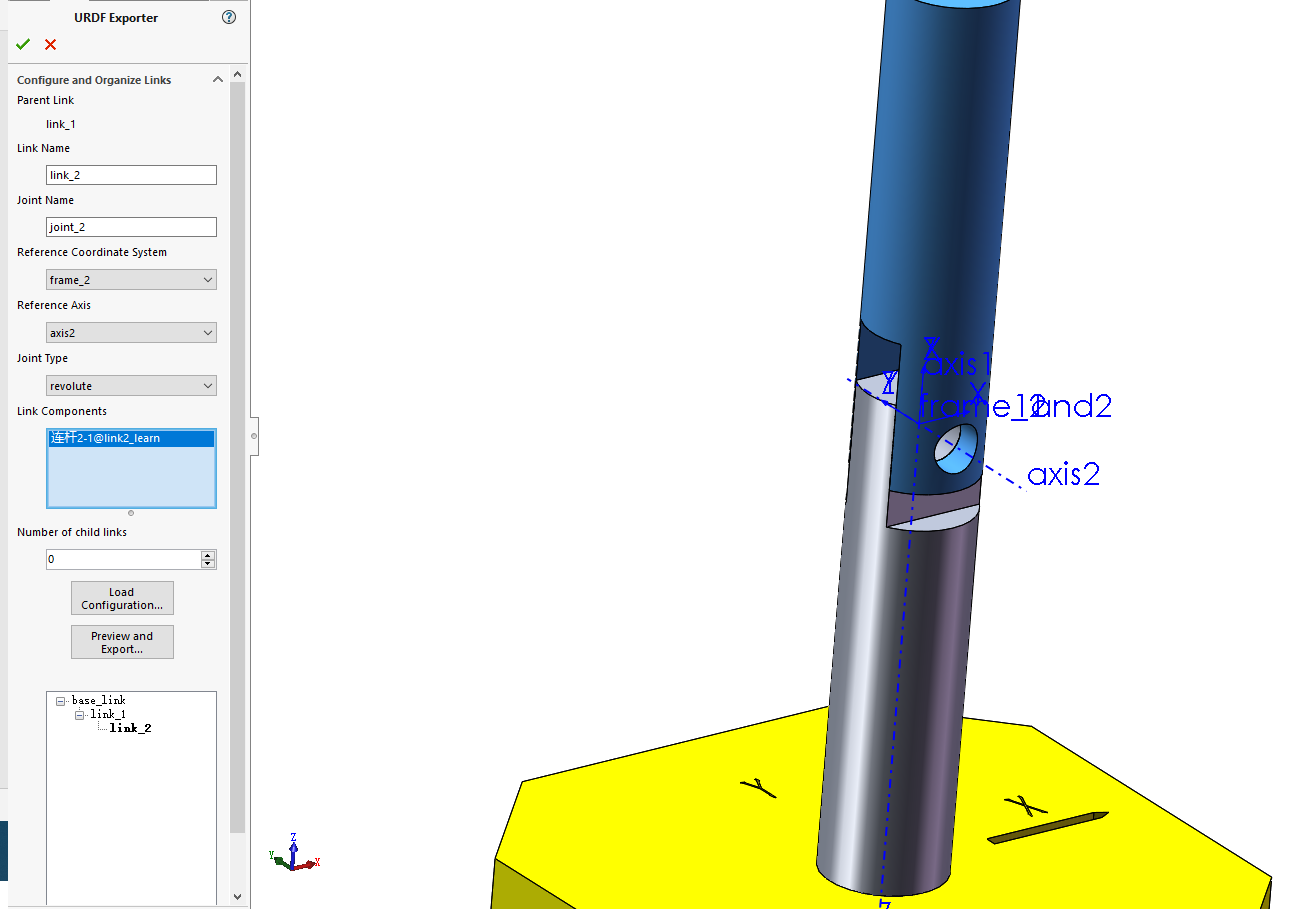




配置link\_1的时候我们可以看到多出了很多选项，

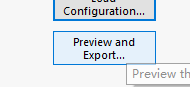
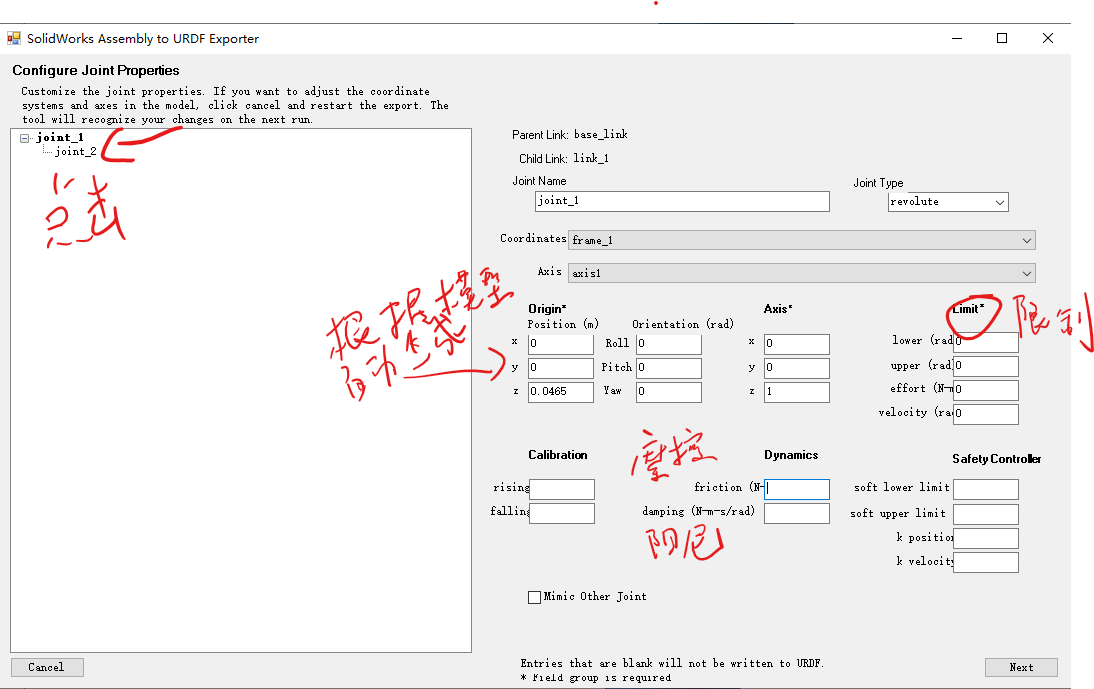
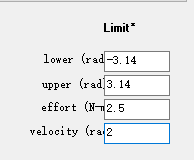
Linkname我们填写link\_1,这里的joint是link\_1相对于父link旋转的关节，我们称它为joint\_1。

旋转轴和坐标系我们分别选择对应的，一定不要自动识别。Joint type这里是旋转关节，我们就用revolute，后面还有一个link，所以我们的child link数量选择1。

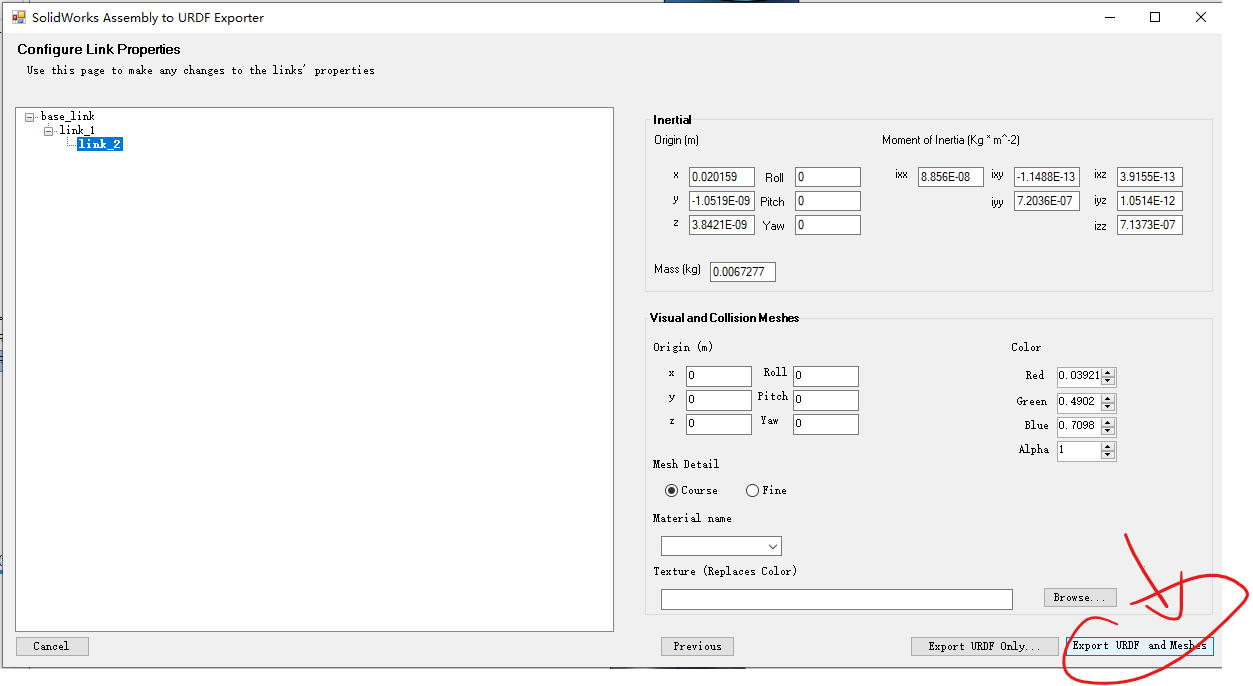
同样link\_2的配置也是一样。配置完建议你点下那个对勾，要不然你手残点错了没保存可就是另一个故事了。（别问我为什么这么熟悉）

配置完之后点这个preview and export，我们来配置关节限位等一系列参数，还有惯量阵等等。当然，这些后期都可以手动修改。

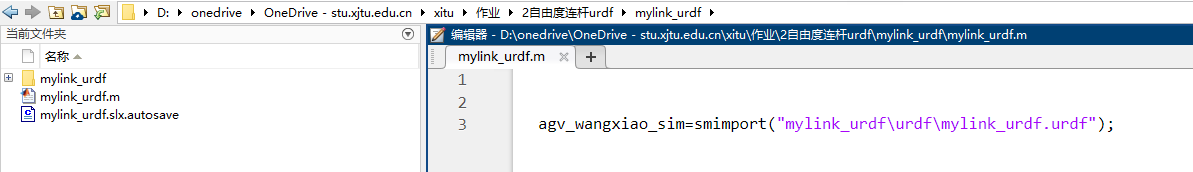
随便填一下关节限位和力矩限制

下一步，检查完没啥问题就点这个：



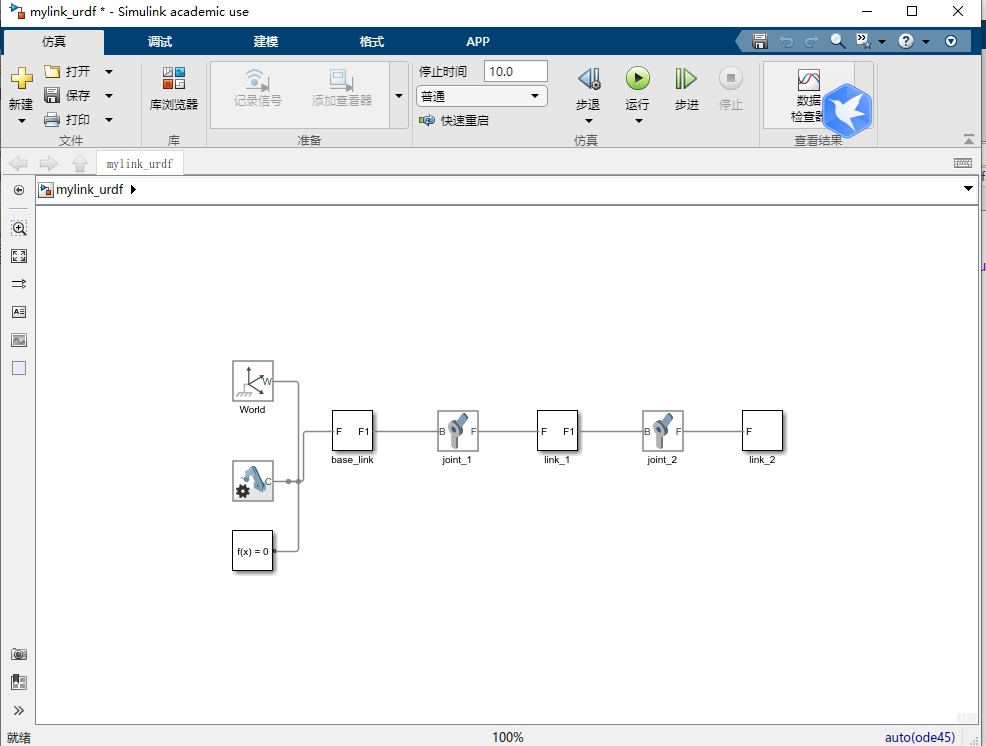
选择完路径和名字，就可以导出了。

我们在matlab中写一行代码：



agv\_wangxiao\_sim=smimport("mylink\_urdf\urdf\mylink\_urdf.urdf");

就可以测试一下你的文件到底能不能用。这里要注意，matlab文件不要写在这个文件夹里面，而且要把工作路径设置到matlab文件相同的路径。这样才能识别urdf文件中的相对路径。不信你可以试试，如果这一步中，你的文件没有炸掉。那就说明你成功了。

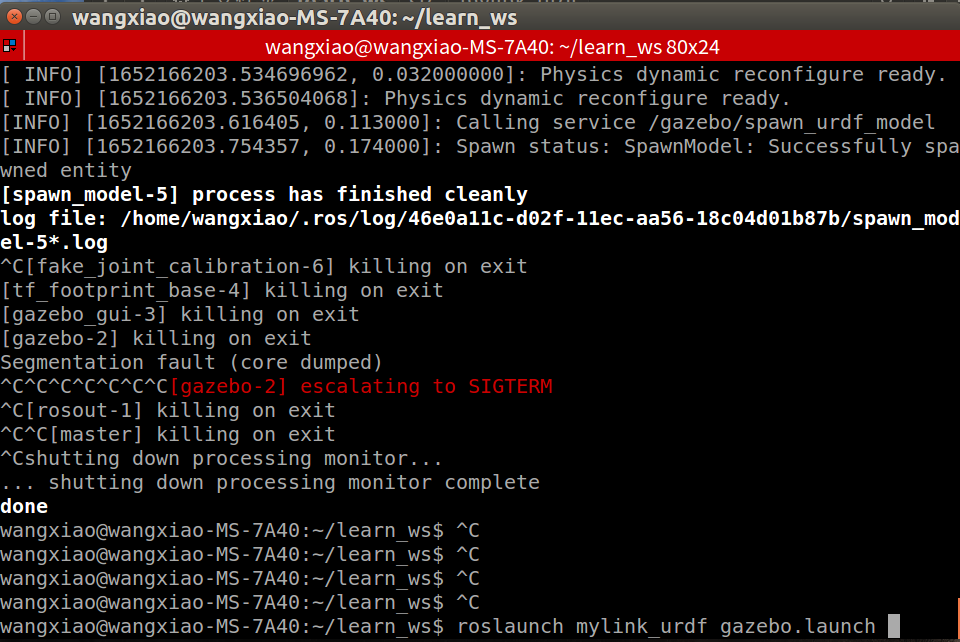
 

把生成的urdf文件夹拷贝到工作区...相信在这里应该不需要解释怎样创建工作空间了把????

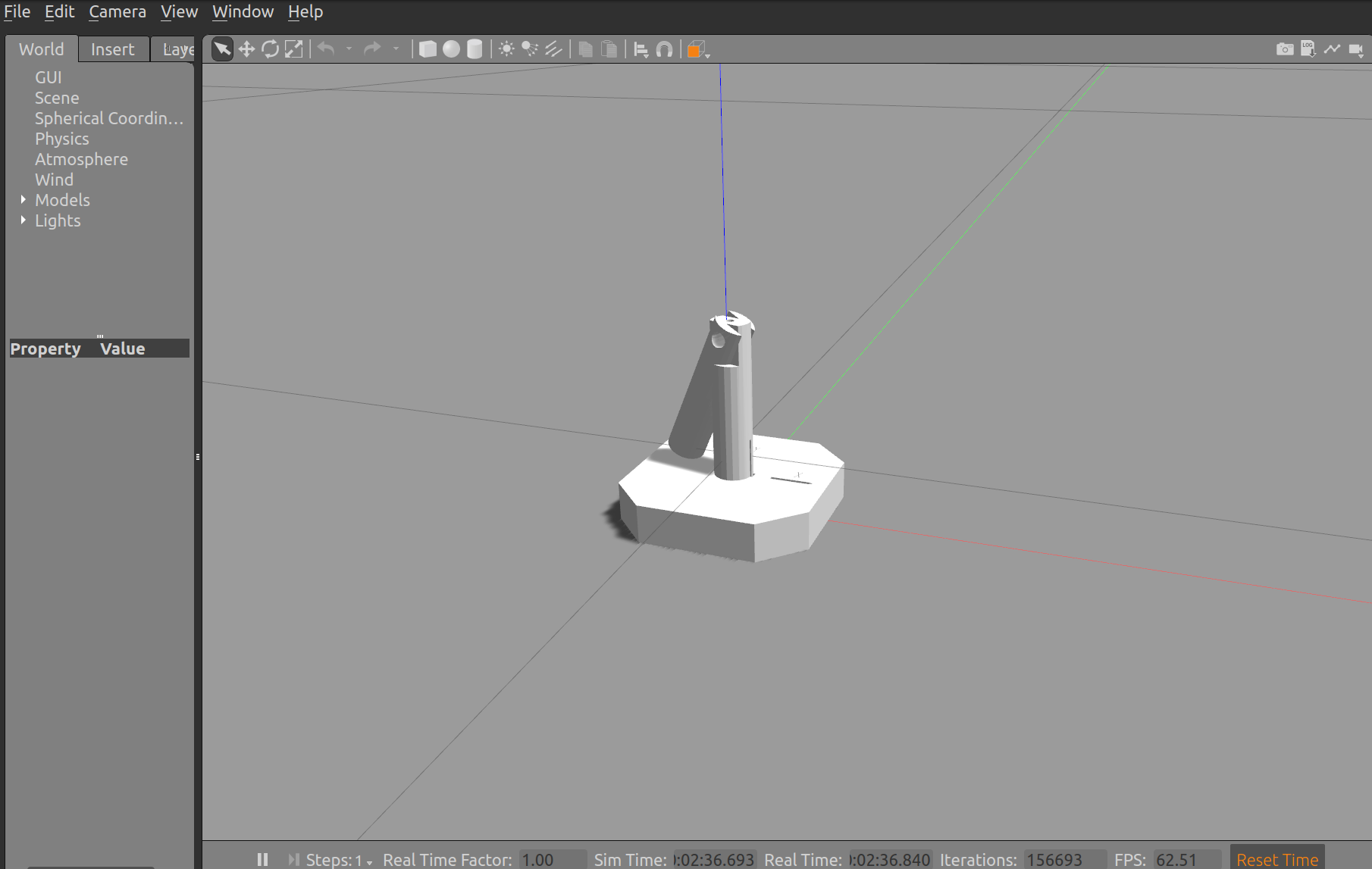
我们自动生成的功能包放到工作区的src文件下,编译一下source一下,或者在.bashrc中配置一下,打开终端,第一次输入命令可能会补全不了,输入:

roslaunch mylink\_urdf gazebo.launch

mylink\_urdf是你自己配置的功能报的名字, gazebo.launch是lanuch文件.



然后就有下图的效果.



因为我们目前没有加入控制器,直接将模型加入到仿真环境中去会像这样软趴趴的...等等,好像有什么奇怪的东西进来了…

我们打开urdf描述文件,现在向urdf中加入变速器\关节电机接口

<transmission name="tran1">

<type>transmission\_interface/SimpleTransmission</type>

<joint name="joint\_1">

<hardwareInterface>hardware\_interface/EffortJointInterface

</hardwareInterface>

</joint>

<actuator name="motor1">

<hardwareInterface>hardware\_interface/EffortJointInterface

</hardwareInterface>

<mechanicalReduction>1</mechanicalReduction>

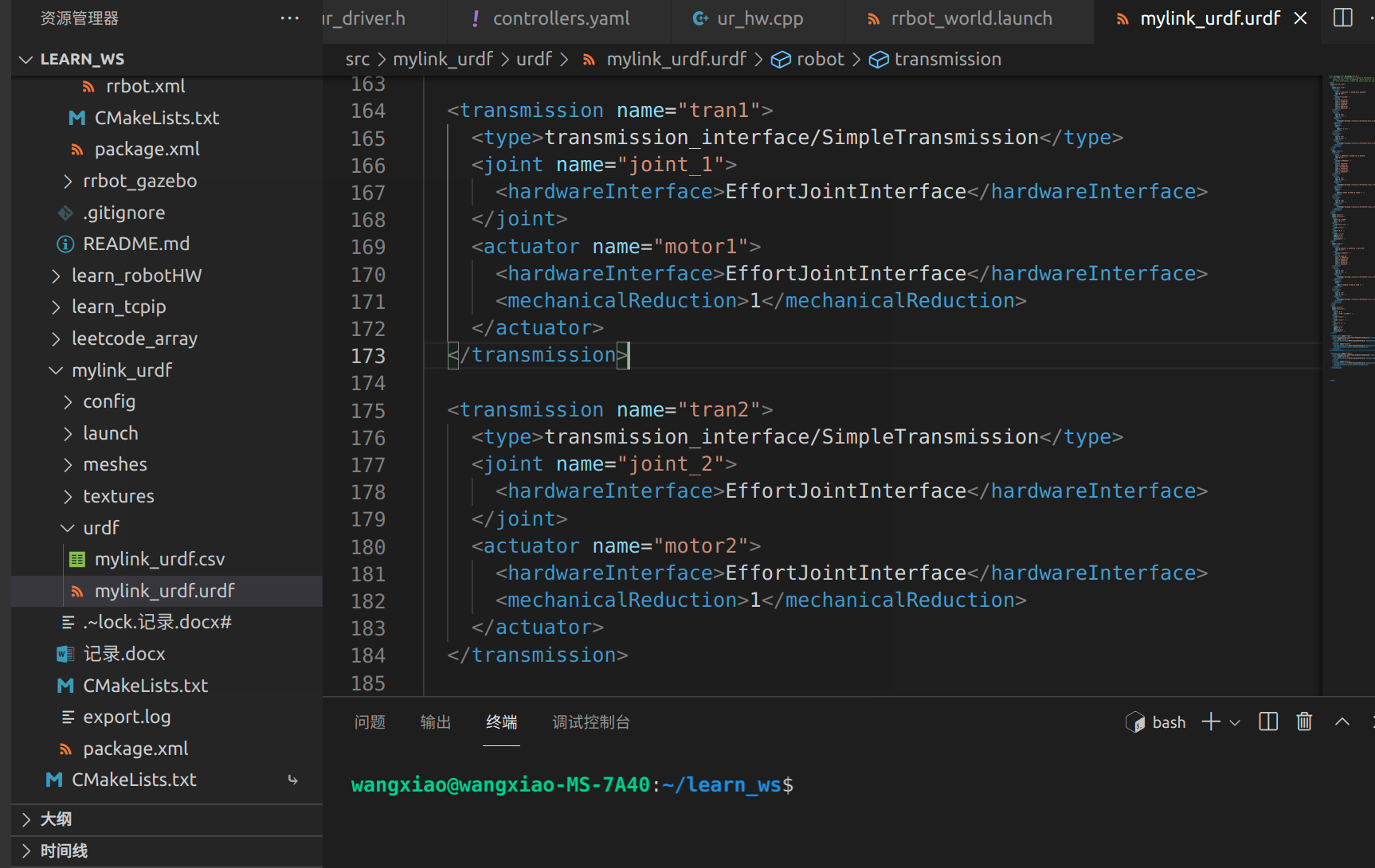
</actuator>

</transmission>

这里要注意的是:type标签中的东西照抄,我也不知道表达的什么含义,反正ros这么复杂,抄就行了.joint标签中写入name

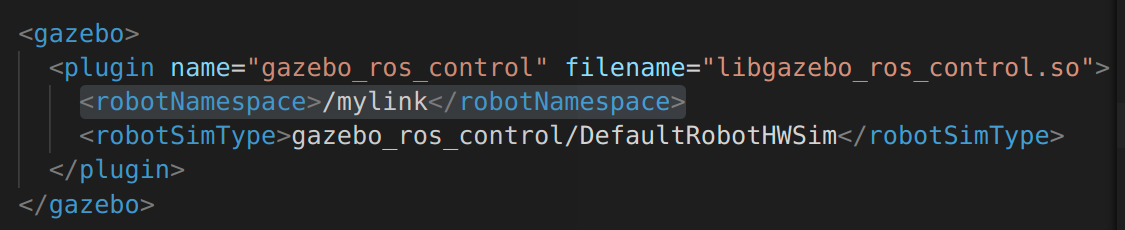
hardwareinterface标签中写入你要加入的硬件控制器类型,你的电机想工作在力矩模式就选择hardware\_interface/EffortJointInterface(这里我改过来了,如果写EffortJointInterface ,运行后会有警告)

详情可参考wiki文档,因为我一般控制器都是自己写,所以很少会用其他的控制接口,一般只用力矩接口或者电流接口.



mechanicalReduction标签是减速比,默认写1就行…

下面加入gazebo的gazebo\_ros\_control标签



<gazebo>

<plugin name="gazebo\_ros\_control" filename="libgazebo\_ros\_control.so">

<robotNamespace>/mylink</robotNamespace>

<robotSimType>gazebo\_ros\_control/DefaultRobotHWSim</robotSimType>

</plugin>

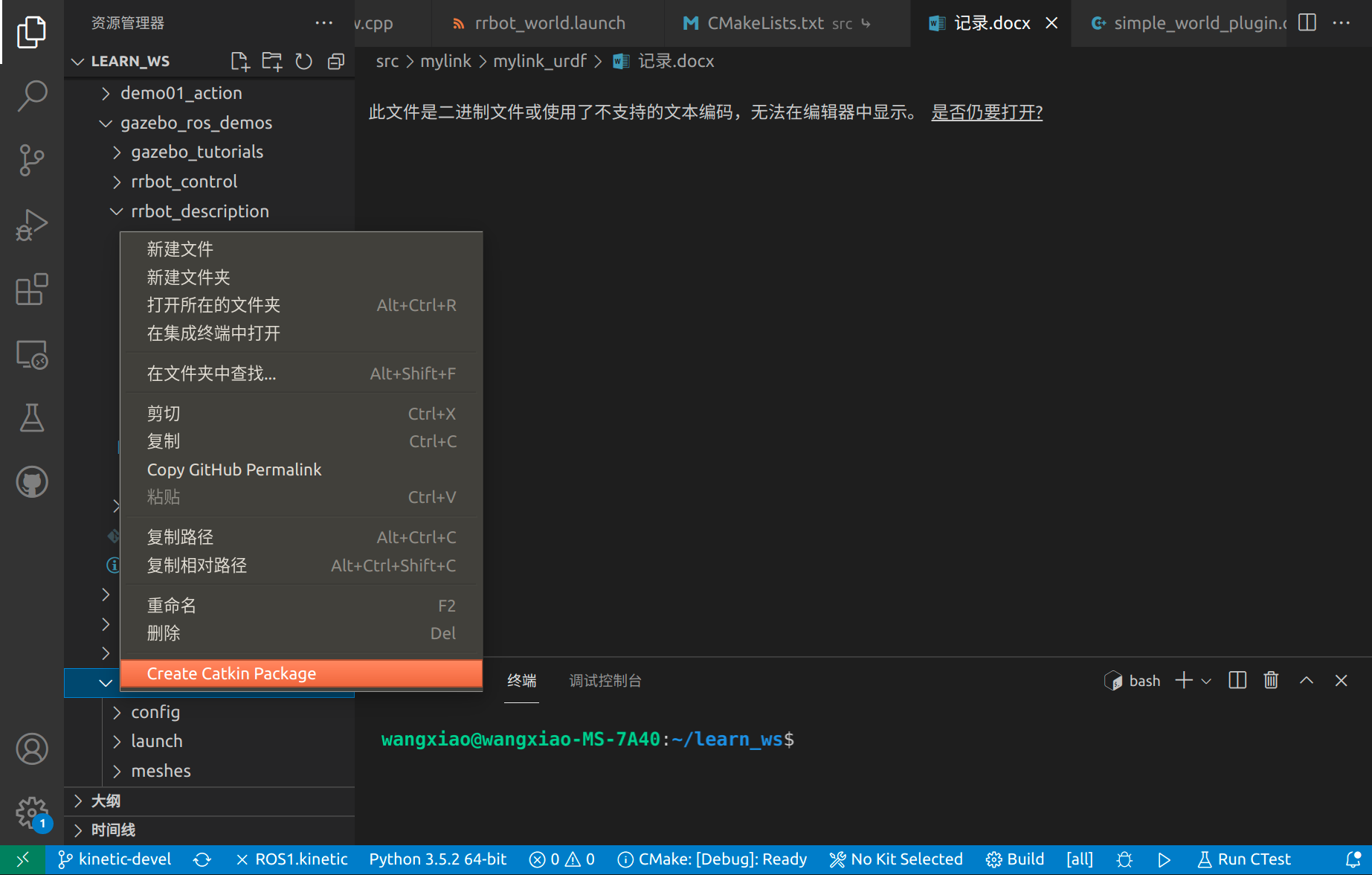
</gazebo>

其中:<robotNamespace>/mylink</robotNamespace>

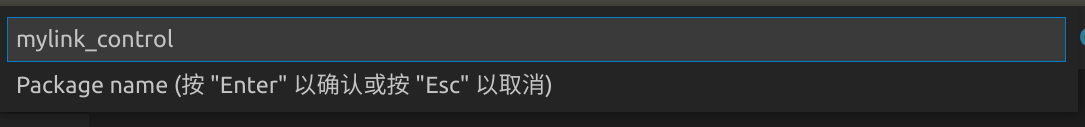
是设置命名空间,要记住这个命名空间,后面要用得到…

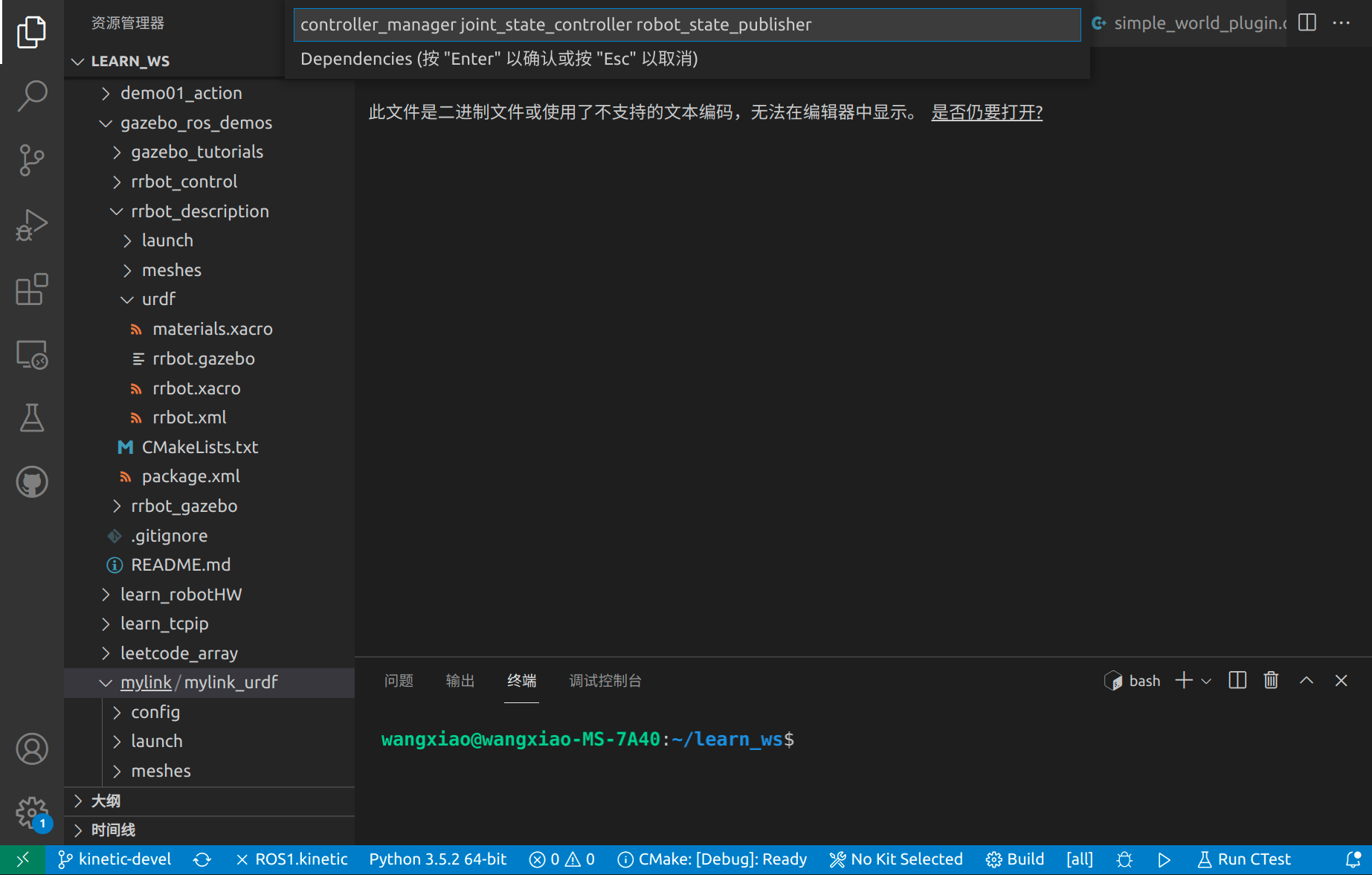
再往后,我们需要新建功能包,但是我的工作空间中有其他的包,所以为了避免混乱,我把与这个link相关的功能包放在同一个文件夹下也就是/learn\_ws/src/mylink/各类功能包

这样子重新编译发现lanuch文件依旧可以打开,没什么问题.



catkin\_create\_pkg



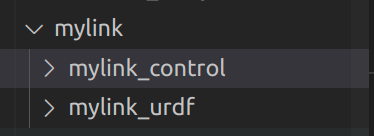


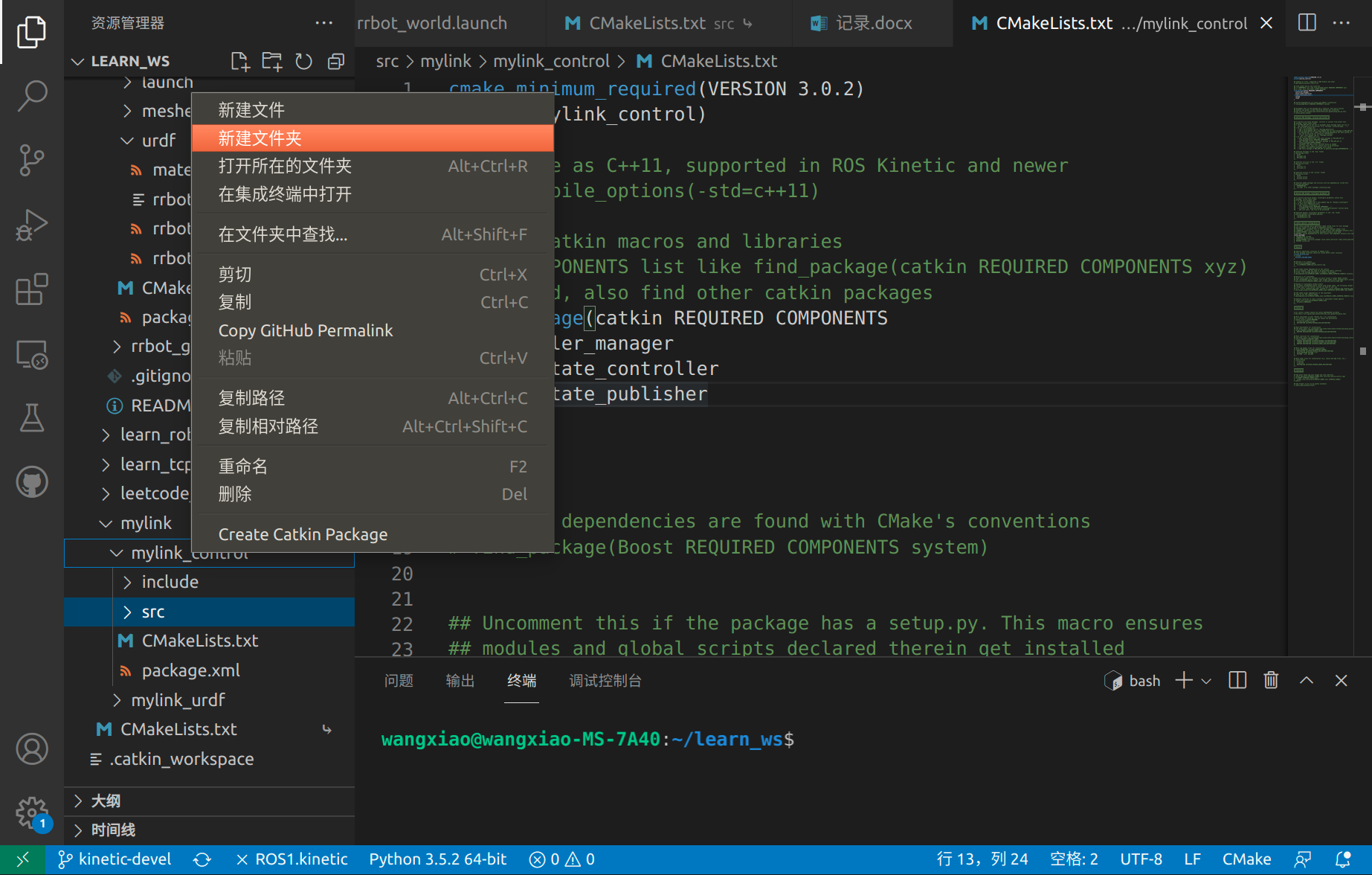
功能包名字:mylink\_control

导入依赖项:

controller\_manager joint\_state\_controller robot\_state\_publisher

搞完还是要手拖一下才能到这种情况





新建一个文件夹叫config和launch

在config文件中配置控制器,注意这里用的python语法,要注意缩进,前面URDF文件中加入的命名空间要与第一行的内容一致.这里的控制器是位置控制器计算出来的力矩指令,也就是反馈回路为位置,经过pid计算得出力矩指令发给机器人….



launch文件如下图所示:



<launch>

<!-- Load joint controller configurations from YAML file to parameter server -->

<rosparam file="$(find mylink\_control)/config/mylink\_control.yaml" command="load"/>

这里是将刚才的yaml文件加载到参数服务器中...

<!-- load the controllers -->

<node name="controller\_spawner" pkg="controller\_manager" type="spawner" respawn="false"

output="screen" ns="/mylink" args="joint1\_position\_controller

joint2\_position\_controller

joint\_state\_controller"/>

这里是将yaml文件中配置的控制器通过控制器管理器controller\_manager启动起来

<!-- convert joint states to TF transforms for rviz, etc -->

<node name="robot\_state\_publisher" pkg="robot\_state\_publisher" type="robot\_state\_publisher"

respawn="false" output="screen">

<remap from="/joint\_states" to="/mylink/joint\_states" />

</node>

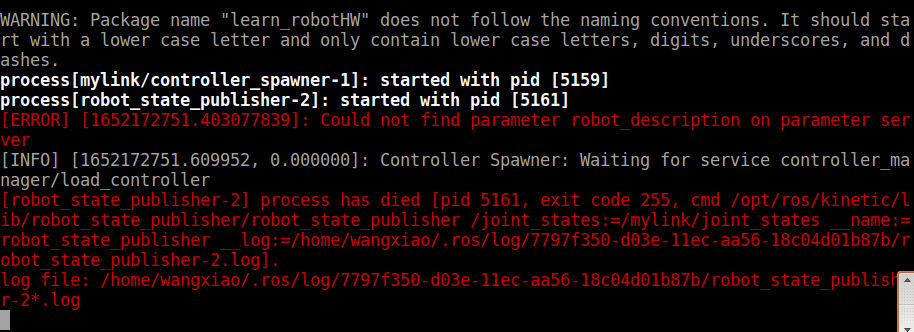
这里是将机器人的状态通过节点发布出来

还做了一个重映射(remap)

</launch>

笔者根据网上的垃圾教程走到这一步,当你运行完启动gazebo.launch文件再启动mylink\_control.launch文件就会发现错误百出……

你发现你根本就运行不了控制器,报错如下:

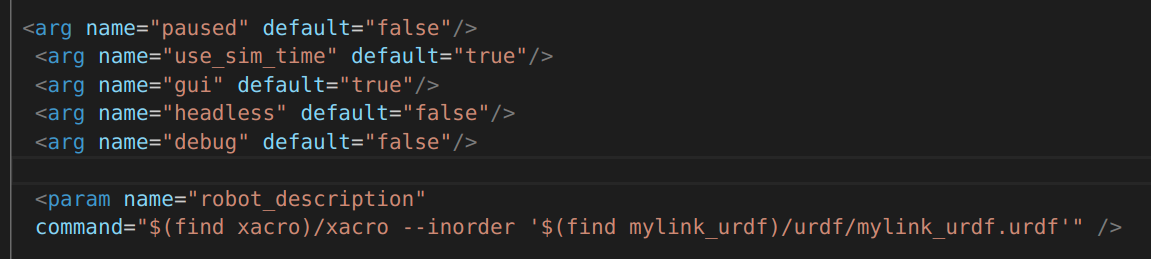


Could not find parameter robot\_description on parameter server

cmd /opt/ros/kinetic/lib/robot\_state\_publisher/robot\_state\_publisher /joint\_states:=/mylink/joint\_states \_\_name:=robot\_state\_publisher

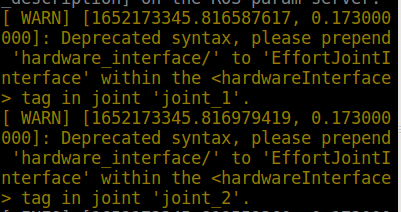
仔细观察报错内容,其实是机器人描述参数没有加载进来,但是仔细想想,机器人的参数,肯定不是在控制器里加载的吧….应该是在进入gazebo的时候把参数加载出来...于是根据这个思路,我去检查了网上能跑通的例程...检查了他们的gazebo.launch启动文件..

发现少了这几行



虽然上面几行参数没有看懂什么意思,但是我觉得应该是需要加上的...或者说问题不大(我注释吊那5行感觉没什么影响 )..关键是最下面这一行,将机器人描述参数加载到参数服务器中,参数名称为:robot\_description

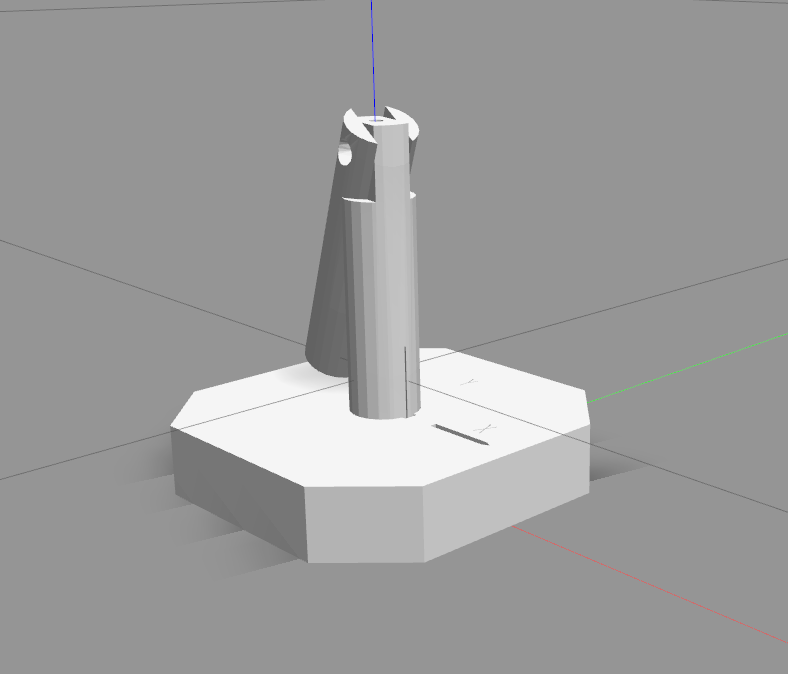
然后再启动一次gazebo发现warning



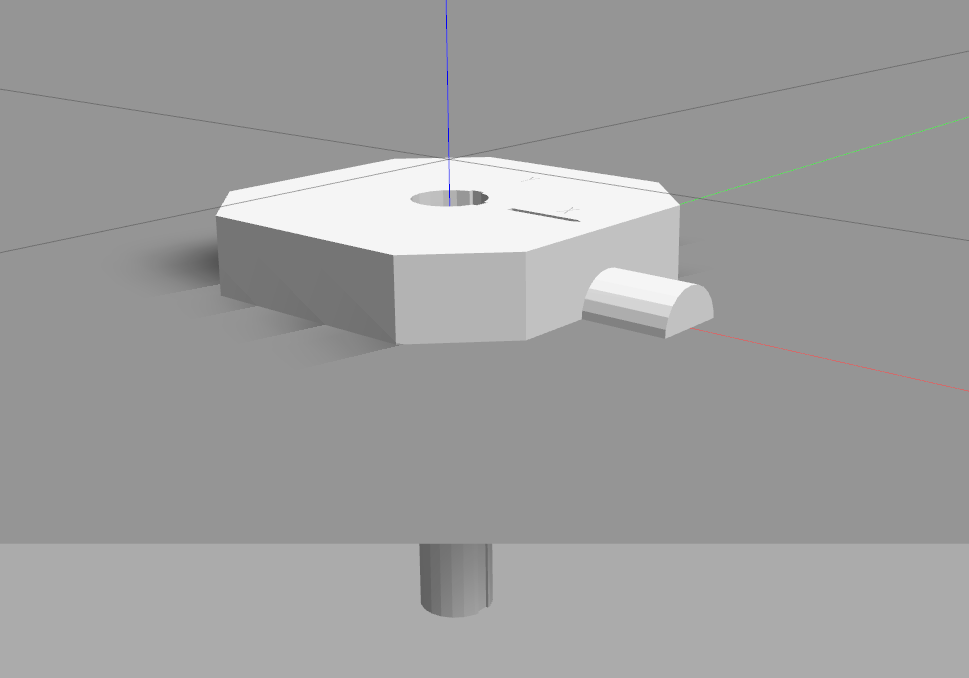
[ WARN] [1652173345.816587617, 0.173000000]: Deprecated syntax, please prepend 'hardware\_interface/' to 'EffortJointInterface' within the <hardwareInterface> tag in joint 'joint\_1'.

[ WARN] [1652173345.816979419, 0.173000000]: Deprecated syntax, please prepend 'hardware\_interface/' to 'EffortJointInterface' within the <hardwareInterface> tag in joint 'joint\_2'.

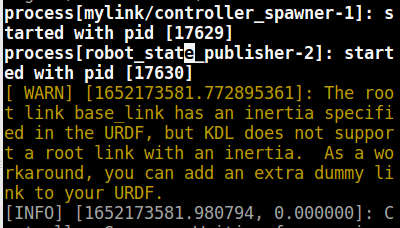
此时机器人变软,不像之前一样立在哪里..



运行mylink\_control.launch文件,结果机器人直接埋到土里……………………………..

然后mylink\_control报warn:

[ WARN] [1652173581.772895361]: The root link base\_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.

如图:

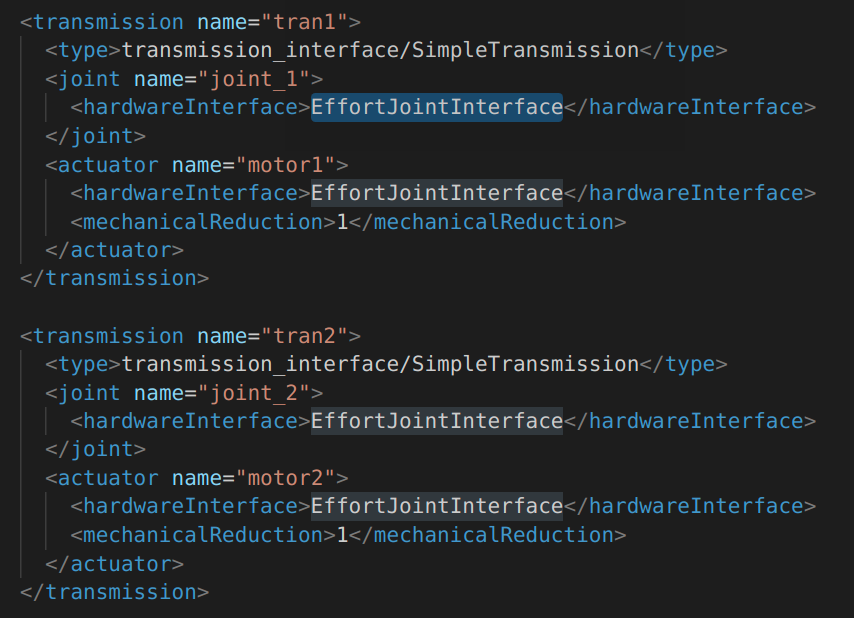
这我直接吐了……………

一个bug一个bug地解决,首先我们看第一个bug

[ WARN] [1652173345.816587617, 0.173000000]: Deprecated syntax, please prepend 'hardware\_interface/' to 'EffortJointInterface' within the <hardwareInterface> tag in joint 'joint\_1'.

[ WARN] [1652173345.816979419, 0.173000000]: Deprecated syntax, please prepend 'hardware\_interface/' to 'EffortJointInterface' within the <hardwareInterface> tag in joint 'joint\_2'.

我百度了一下说我的urdf文件有问题



应该将这里的EffortJointInterface

改成hardware\_interface/EffortJointInterface

好的我们改完之后试一下

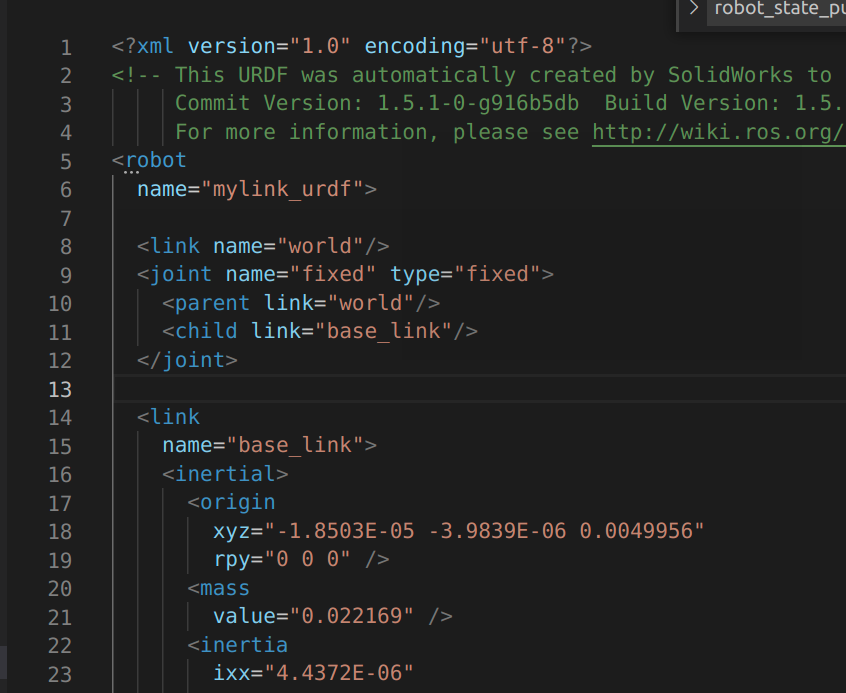
虽然没有了WARN,但是gazebo中模型的表现完全没有变!!!!气死了

于是我开始解决第二个bug…

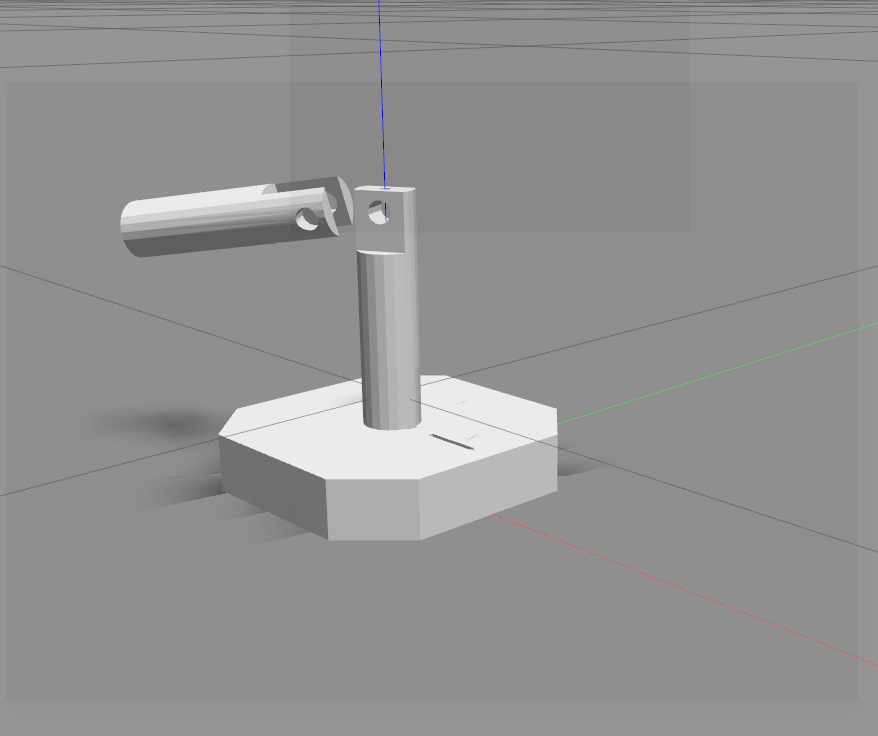
百度搜索了一下

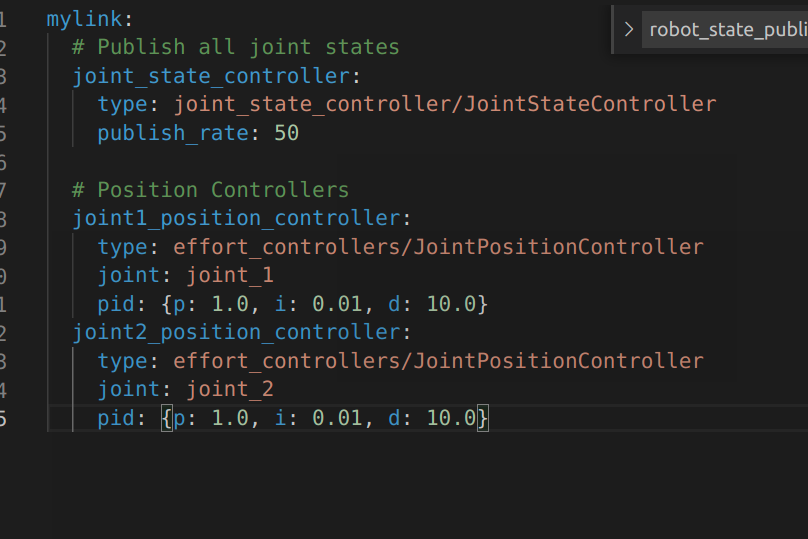
[ WARN] [1652173581.772895361]: The root link base\_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to your URDF.

发现,在urdf文件中需要定义一个没有质量的坐标系才行...比如我的base\_link坐标系是有质量数据的,所以直接加一个world坐标系,和base坐标系重合即可,如图所示,在base\_link前面加一个world坐标系和joint



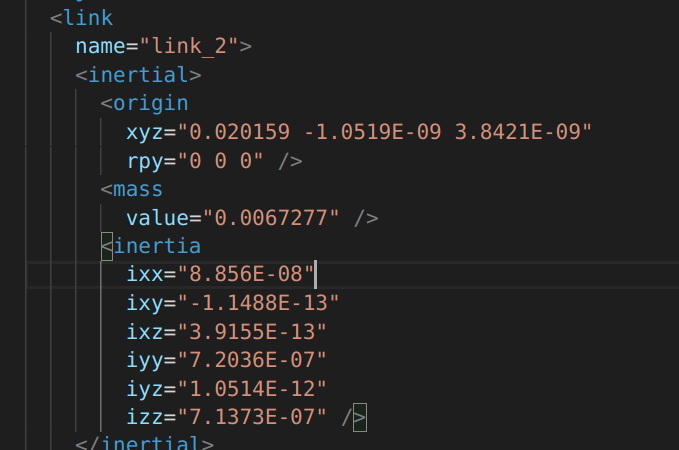
这个时候再运行,发现显示已经正常了,没有警告和错误.但是启动控制器它变得抽搐了起来,我怀疑pid参数调的高了才导致这种情况.

所以我把pid的p值从100调小到1

重新启动gazebo和控制器

发现还是不行,,,我又找了各种各样的问题,发现都不是问题的关键.直到有一次启动控制器后,我发现连杆差点挣脱joint的限制飞了出来,于是我就怀疑控制器给出的命令直接导致连杆的执行器速度过大导致仿真环境解算不出来…….

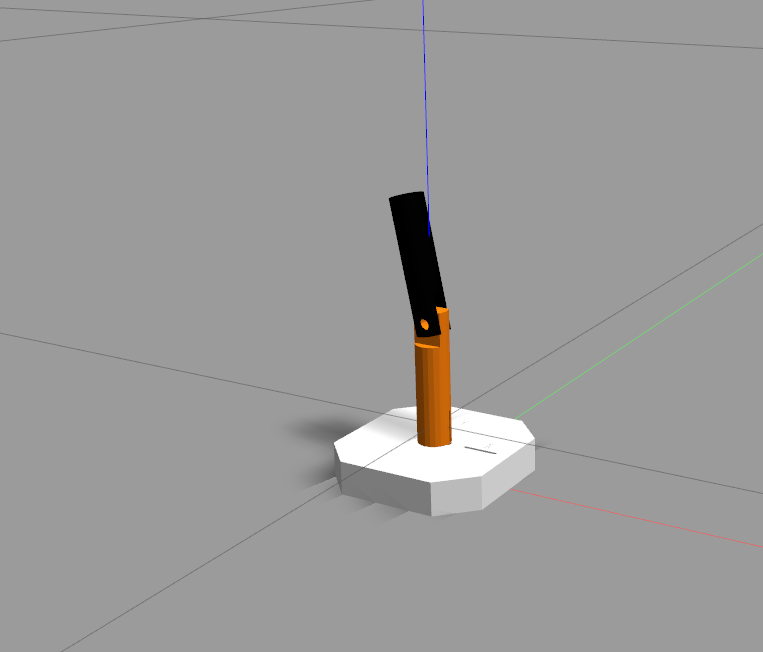
根据牛顿力学,一个是力太大,另一个原因就是质量太小,于是我返回urdf文件中找到质量,



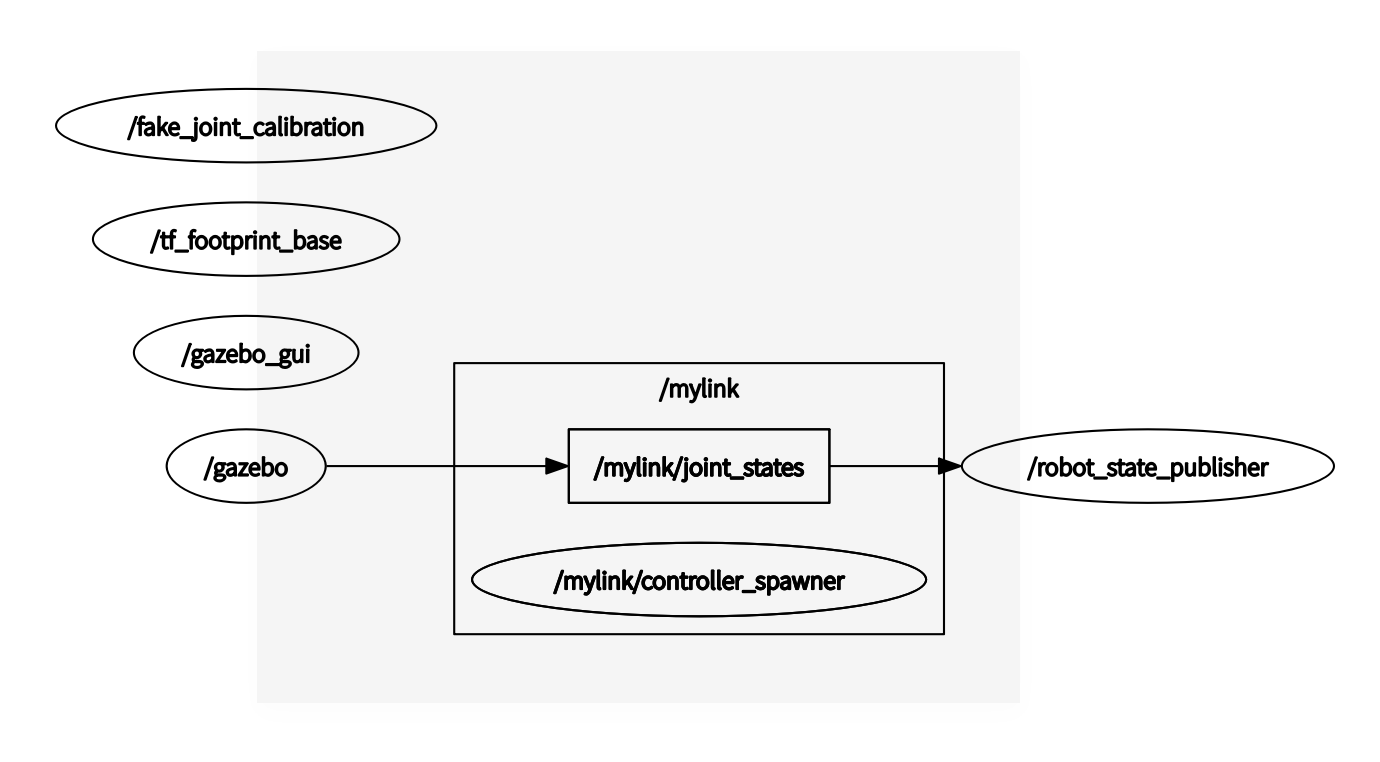
惯性张量都到了负13次方,质量也只有6克,所以我人为将质量放大到1公斤,其它的都加几个数量级...运行后

虽然有点鬼畜,但是至少是跑通了,重新调一下pid

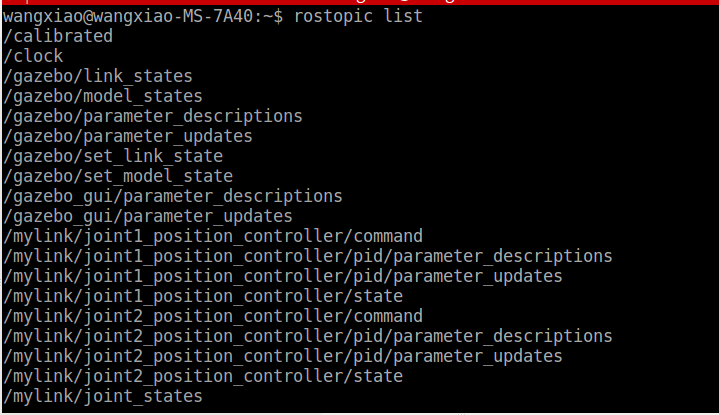
得到下图



使用rqt\_graph查看节点:

然后输入命令rostopiclist

/mylink/joint2\_position\_controller/command

/mylink/joint1\_position\_controller/command

这两个话题就是可以给出命令的话题

我们使用

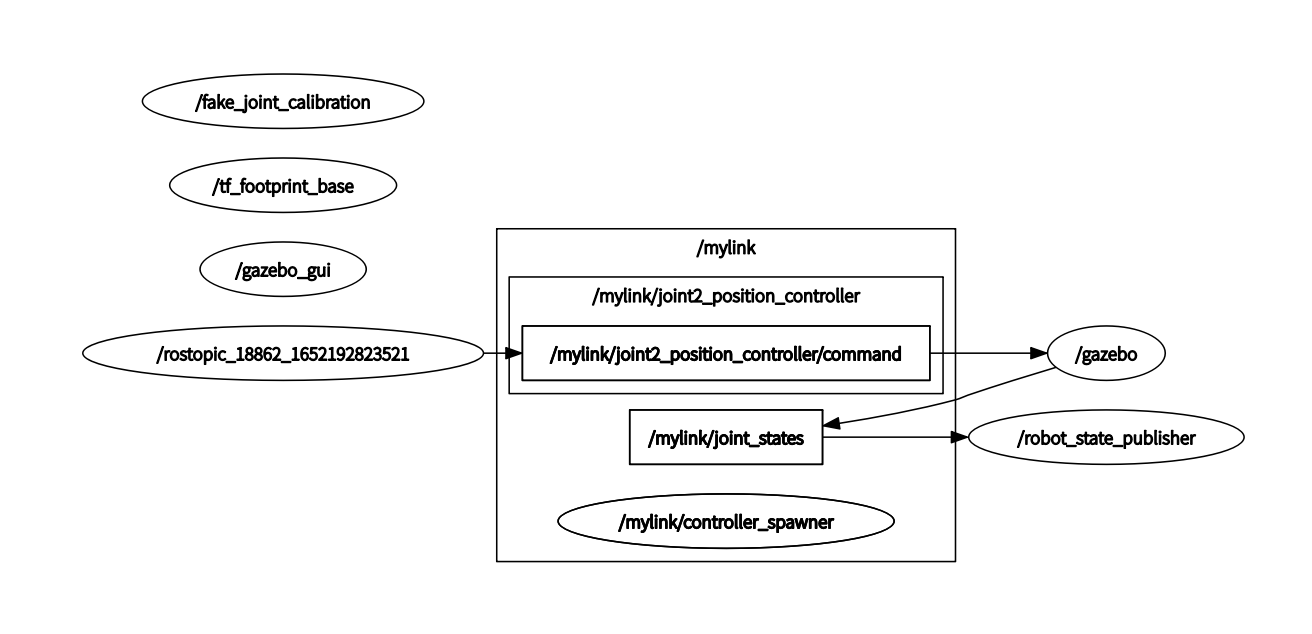
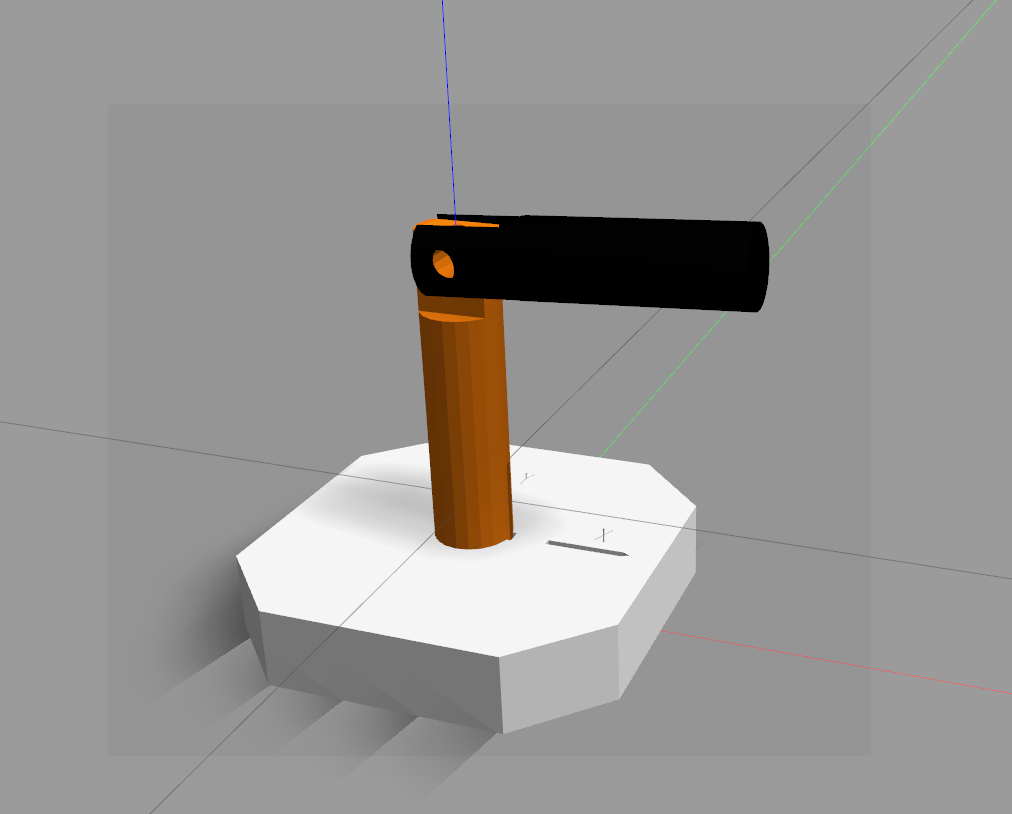
rostopic pub /mylink/joint2\_position\_controller/command std\_msgs/Float64 1.5

命令给控制器发一个指令

于是机械臂运动到了这个状态:

我们更新rqt\_graph得到:

可以说,终于一切正常了…

rqt\_graph显示的节点图传回来一切正常

<https://blog.csdn.net/TIANBOT/article/details/124044371>

[https://blog.csdn.net/wxflamy/article/details/79266726#fnref:foot1](https://blog.csdn.net/wxflamy/article/details/79266726" \l "fnref:foot1)

写一个控制器的步骤

上一节我们使用ros自带的控制器,这一节我们来自己变写一个控制器

新建一个带有必要依赖的ROS包；

编写控制器C++代码；

将C++类注册或者发布为一个插件；

在XML文件中定义插件；

更新package.xml来发布插件；

编写CMakeLists.txt

编译代码

为控制器参数编写配置文件

在gazebo中进行仿真

使用controller manager加载控制器。