注意xml文件下的注释格式为：

<!-- 内容 -->

但为了方便，我的代码注释为c语言的//

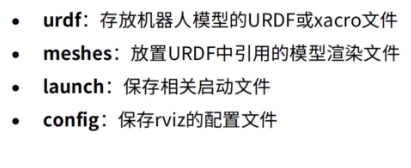
至于源码附在了文章的末尾

1.创建功能包，初始化环境

catkin\_create\_pkg mbot\_description urdf xacro

2.配置环境

3.在src文件夹下新建urdf、meshes、launch、config文件夹



4.在urdf文件里面新建一个mbot.urdf文件

5.代码解读：

<?xml version="1.0" ?>

<robot name="mbot">

<!-- ///////////////////////////////////// COLOR ///////////////////////////////////////////// -->

<material name="Black">

<color rgba="0 0 0 1"/>

</material>

<material name="White">

<color rgba="1 1 1 0.95"/>

</material>

<material name="Blue">

<color rgba="0 0 1 1"/>

</material>

<material name="Yellow">

<color rgba="1 0.4 0 1"/>

</material>

<!-- ///////////////////////////////////// BASE ///////////////////////////////////////////// -->

<link name="base\_link">

<visual>

//位置信息

<origin xyz=" 0 0 0" rpy="0 0 0" />

//外观形状信息

<geometry>

<cylinder length="0.16" radius="0.20"/>

</geometry>

//下方是材料，特指颜色，而对颜色的定义Yellow事先已定义放在了开头，即第十四行

<material name="Yellow"/>

</visual>

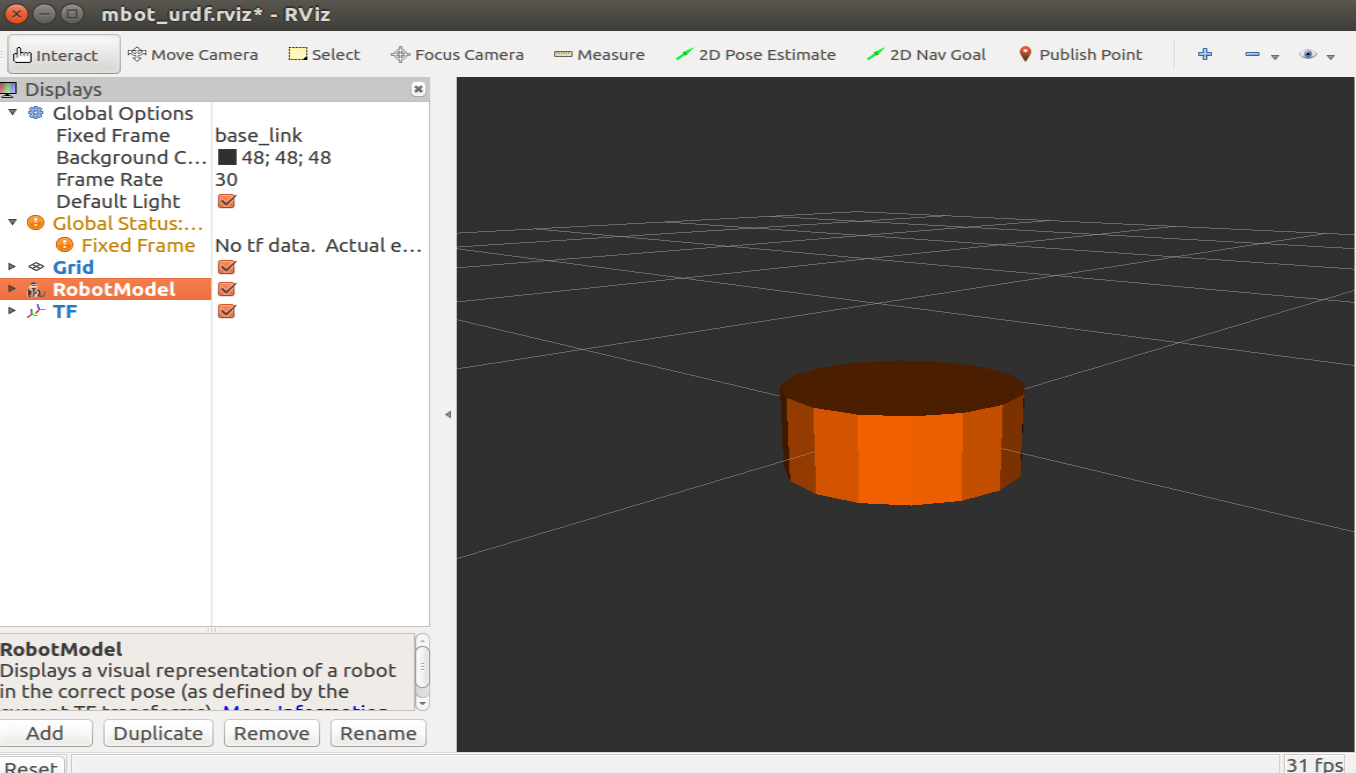
</link>

//到此处位置，已经完成了底盘的。

创建

上述已经完成了小车底盘的创建。把中文都删去，运行时。

效果如下图：底座



6.在launch文件里面放入启动文件display\_mbot\_urdf.launch

内容如下：

<launch>

<!-- 设置机器人模型路径参数 -->

<param name="robot\_description" textfile="$(find mbot\_description)/urdf/mbot.urdf" />

<!-- 运行joint\_state\_publisher节点，发布机器人的关节状态 -->

<node name="joint\_state\_publisher\_gui" pkg="joint\_state\_publisher\_gui" type="joint\_state\_publisher\_gui" />

<!-- 运行robot\_state\_publisher节点，发布tf -->

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

<!-- 运行rviz可视化界面 -->

<node name="rviz" pkg="rviz" type="rviz" args="-d $(find mbot\_description)/config/mbot\_urdf.rviz" required="true" />

</launch>

7.编译一下工作环境：

catkin\_make

source devel/setup.zsh

8.启动launch文件：

roslaunch mbot\_description display\_mbot\_urdf.launch

来启动rviz

9.Fix Frame改为base\_link

避免下一次进入rviz还要设置一下rviz，可以保存该文件，左上角file，save as另存为，放入功能包的config中

10.详细的mbot.urdf文件内容：

<?xml version="1.0" ?>

<robot name="mbot">

<!-- ///////////////////////////////////// COLOR ///////////////////////////////////////////// -->

<material name="Black">

<color rgba="0 0 0 1"/>

</material>

<material name="White">

<color rgba="1 1 1 0.95"/>

</material>

<material name="Blue">

<color rgba="0 0 1 1"/>

</material>

<material name="Yellow">

<color rgba="1 0.4 0 1"/>

</material>

<!-- ///////////////////////////////////// BASE ///////////////////////////////////////////// -->

<link name="base\_link">

<visual>

//位置信息

<origin xyz=" 0 0 0" rpy="0 0 0" />

//外观形状信息

<geometry>

//cylinder是圆柱体的意思，length对应高度，radius对应半径

<cylinder length="0.16" radius="0.20"/>

</geometry>

//下方是材料，特指颜色，而对颜色的定义Yellow事先已定义放在了开头，即第十四行

<material name="Yellow"/>

</visual>

</link>

//到此处位置，已经完成了底盘的创建

<!-- ///////////////////////////////////// WHEEL ///////////////////////////////////////////// -->

<link name="left\_wheel\_link">

<visual>

//解释一下这里的1.5707

//轮子我们不能平放，要竖着放，那么它转动才会前移

//但这里的单位都是弧度

//1度=pi/180弧度

//所以90度就是90\*pi/180=1.57...

//xyz是坐标

<origin xyz="0 0 0" rpy="1.5707 0 0" />

<geometry>

<cylinder radius="0.06" length = "0.025"/>

</geometry>

<material name="White"/>

</visual>

</link>

//joint类型无限制旋转，自由旋转

<joint name="left\_wheel\_joint" type="continuous">

<origin xyz="0 0.19 -0.05" rpy="0 0 0"/>

//这里的父子链接要写，表明了joint要链接的两个link

<parent link="base\_link"/>

<child link="left\_wheel\_link"/>

//相对于child link也就是joint link的坐标系:

//对于continuous或者revolute关节, 反映了绕着哪一根轴旋转,

//对于prismatic关节, 反映了沿着哪一根轴移动.

//对于planar关节, 反映了沿着哪个平面(由法向量体现)移动.

//此处表示的是车轮（驱动轮）沿着y轴转动

<axis xyz="0 1 0"/>

</joint>

<link name="right\_wheel\_link">

<visual>

<origin xyz="0 0 0" rpy="1.5707 0 0" />

<geometry>

<cylinder radius="0.06" length = "0.025"/>

</geometry>

<material name="White"/>

</visual>

</link>

<joint name="right\_wheel\_joint" type="continuous">

<origin xyz="0 -0.19 -0.05" rpy="0 0 0"/>

<parent link="base\_link"/>

<child link="right\_wheel\_link"/>

<axis xyz="0 1 0"/>

</joint>

<!-- ///////////////////////////////////// CASTER //支撑轮 ///////////////////////////////////////////// -->

<link name="front\_caster\_link">

<visual>

<origin xyz="0 0 0" rpy="0 0 0"/>

<geometry>

//圆球sphere

<sphere radius="0.015" />

</geometry>

<material name="Black"/>

</visual>

</link>

<joint name="front\_caster\_joint" type="fixed">

<origin xyz="0.18 0 -0.095" rpy="0 0 0"/>

<parent link="base\_link"/>

<child link="front\_caster\_link"/>

</joint>

<link name="back\_caster\_link">

<visual>

<origin xyz="0 0 0" rpy="0 0 0"/>

<geometry>

<sphere radius="0.015" />

</geometry>

<material name="Black"/>

</visual>

</link>

<joint name="back\_caster\_joint" type="fixed">

<origin xyz="-0.18 0 -0.095" rpy="0 0 0"/>

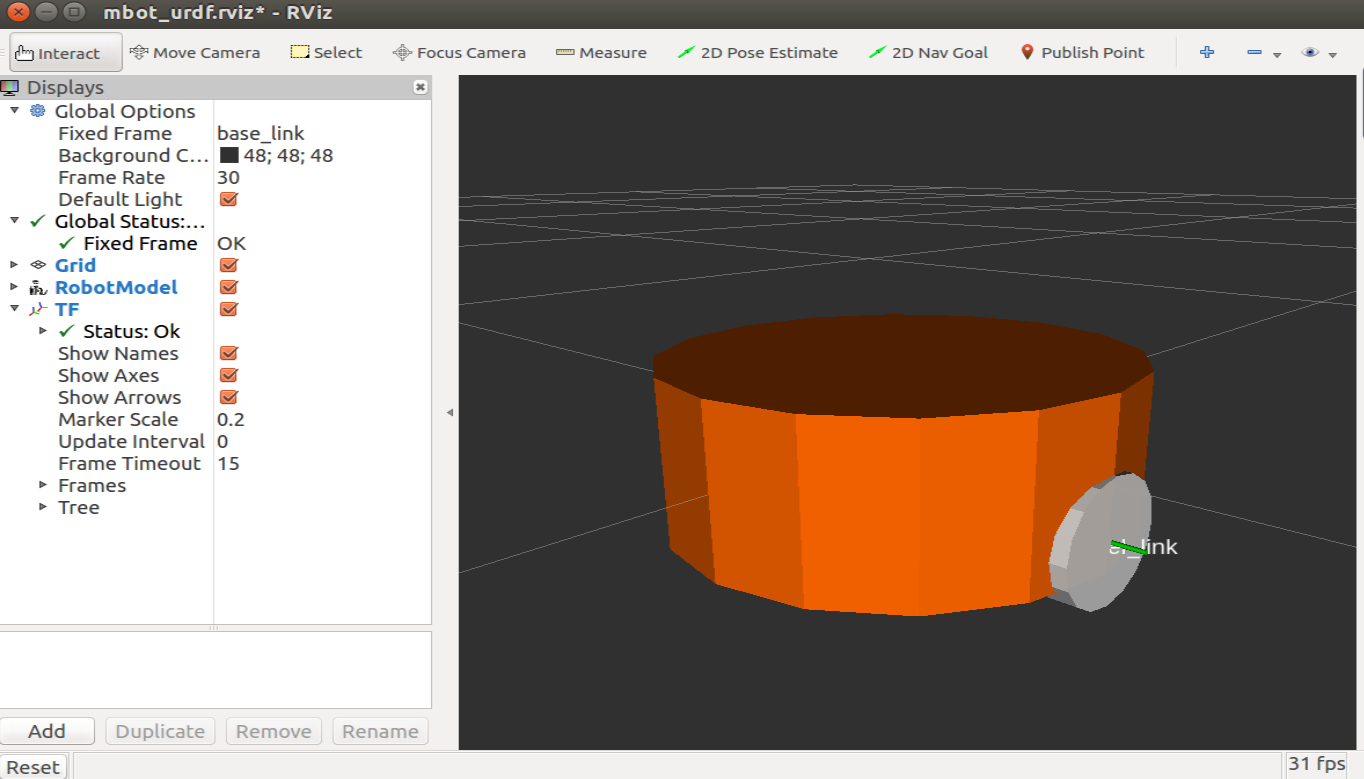
<parent link="base\_link"/>

<child link="back\_caster\_link"/>

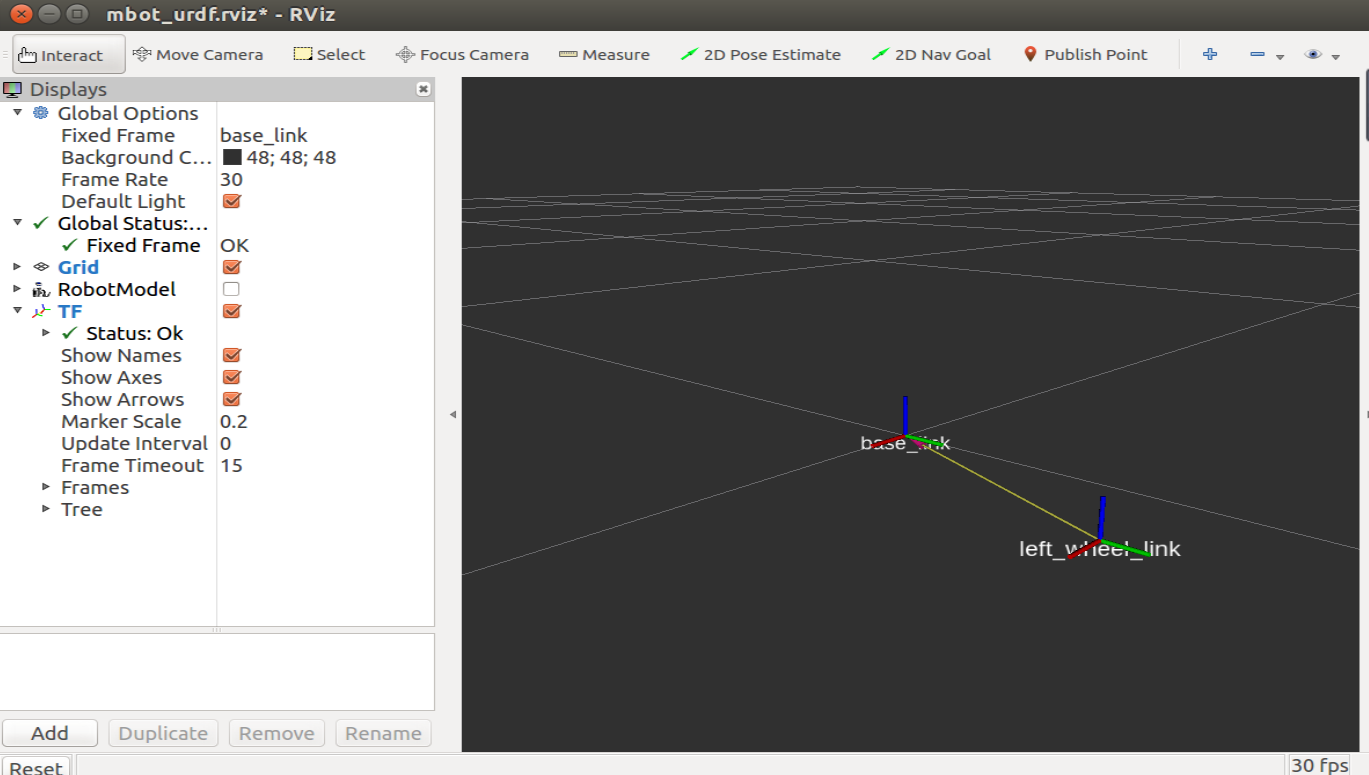
</joint>

</robot>

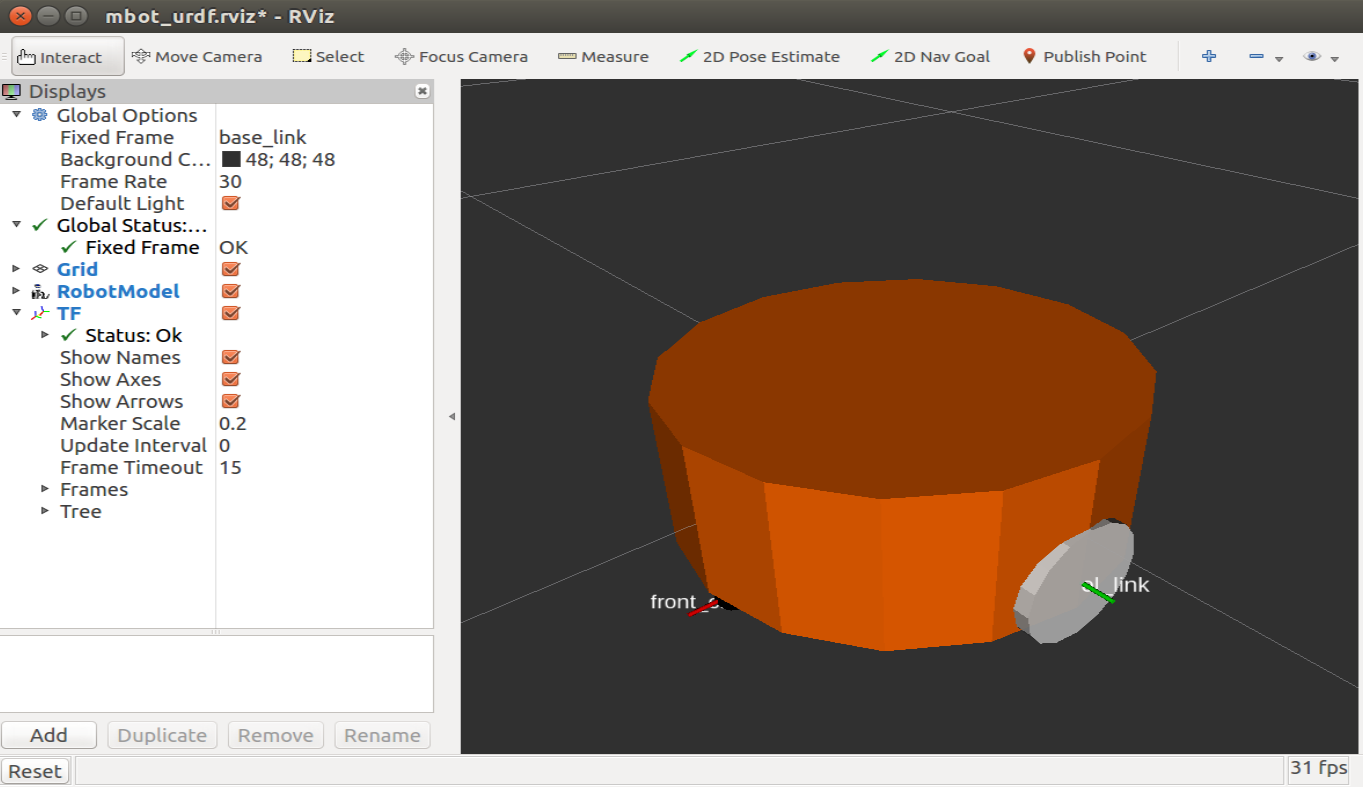
添加了左轮后效果图：



tf坐标图：

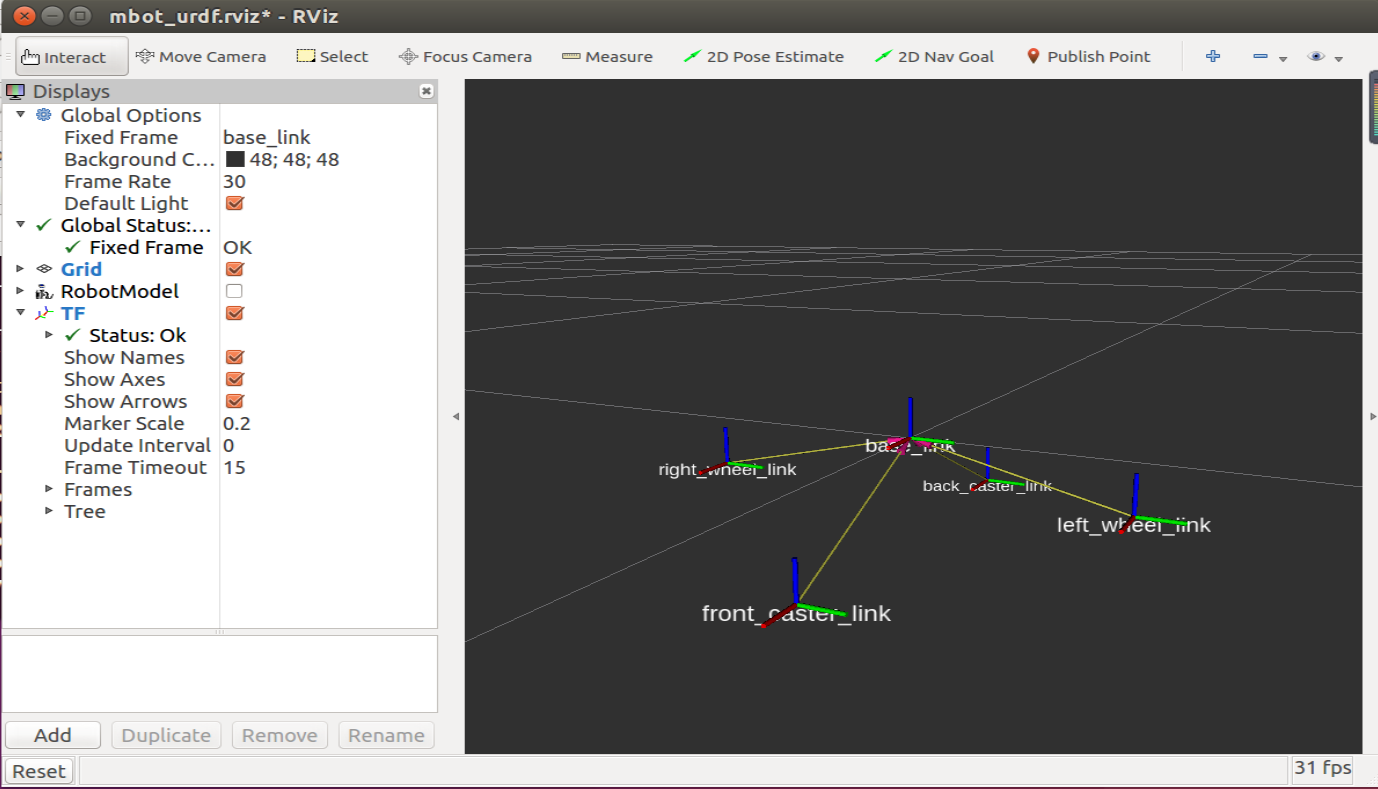


小车平台全景图：





tf坐标图：



接着，我们考虑给小车加一些传感器：比如下面的激光雷达laser

1.在urdf文件里面新建一个urdf文件，并命名为mbot\_with\_laser.urdf

2.代码如下：

在mbot.urdf的基础上添加了如下代码：

<!-- ///////////////////////////////////// LASER ///////////////////////////////////////////// -->

<link name="laser\_link">

<visual>

<origin xyz=" 0 0 0 " rpy="0 0 0" />

<geometry>

<cylinder length="0.05" radius="0.05"/>

</geometry>

<material name="Black"/>

</visual>

</link>

<joint name="laser\_joint" type="fixed">

<origin xyz="0 0 0.105" rpy="0 0 0"/>

<parent link="base\_link"/>

<child link="laser\_link"/>

</joint>

3.调用的launch文件：

<launch>

<!-- 设置机器人模型路径参数 -->

//下面存放的路径和调用的文件

//是唯一和上面小车launch文件不同的地方

<param name="robot\_description" textfile="$(find mbot\_description)/urdf/mbot\_with\_laser.urdf" />

<!-- 运行joint\_state\_publisher节点，发布机器人的关节状态 -->

<node name="joint\_state\_publisher\_gui" pkg="joint\_state\_publisher\_gui" type="joint\_state\_publisher\_gui" />

<!-- 运行robot\_state\_publisher节点，发布tf -->

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

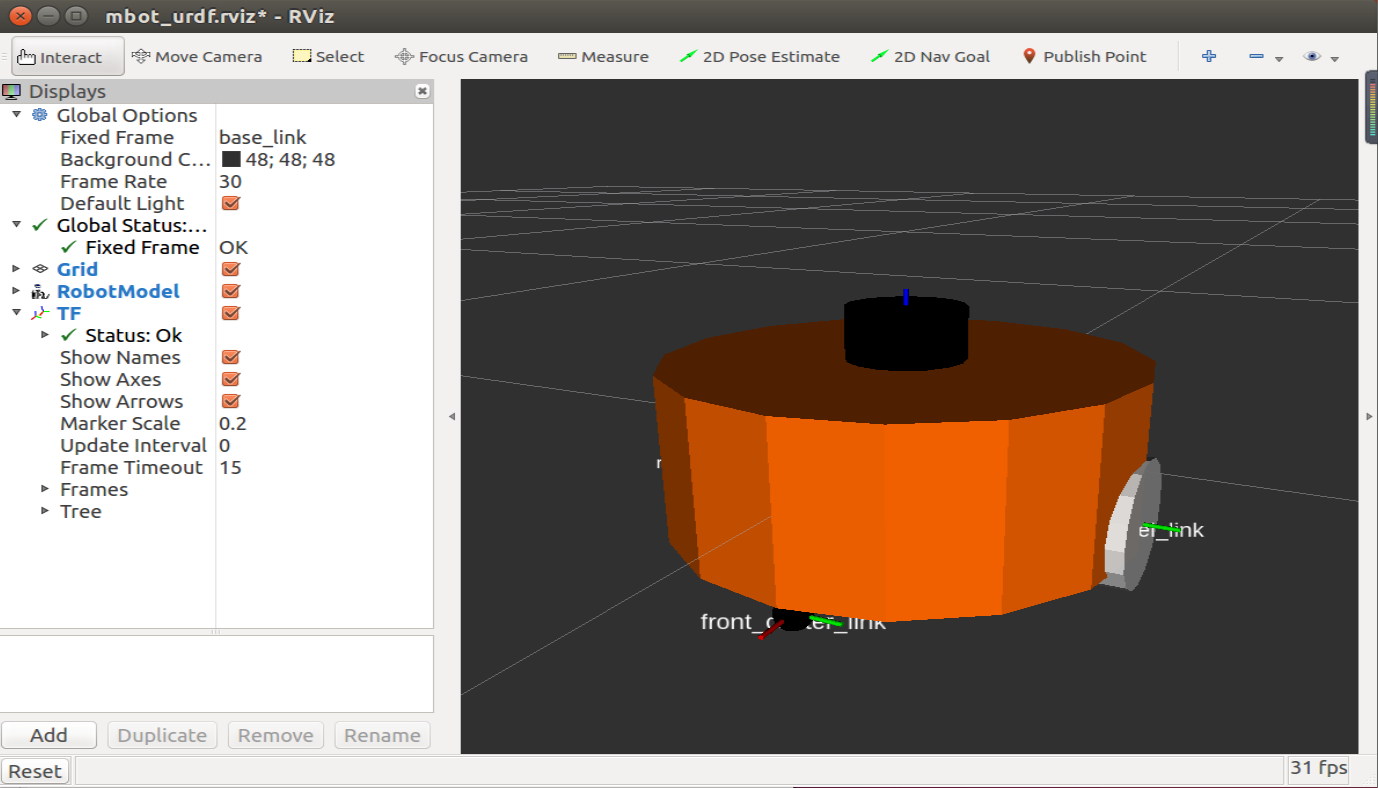
<!-- 运行rviz可视化界面 -->

<node name="rviz" pkg="rviz" type="rviz" args="-d $(find mbot\_description)/config/mbot\_urdf.rviz" required="true" />

</launch>

当然，这里的雷达我们只使用一个小小的圆柱体来代替的，如果觉得不够逼真，可以去网站上找激光雷达的三维模型文件进行配置

效果图：



接下是有三维模型的kinect传感器配置：

1.在urdf文件夹中创建mbot\_with\_kinect.urdf文件

2.在launch文件中创建mbot\_with\_kinect.launch文件

3.在mbot\_with\_kinect.urdf的基础上添加了如下代码：

<!-- ///////////////////////////////////// KINECT ///////////////////////////////////////////// -->

<link name="kinect\_link">

<visual>

<origin xyz="0 0 0" rpy="0 0 1.5708"/>

<geometry>

<mesh filename="package://mbot\_description/meshes/kinect.dae" />

</geometry>

</visual>

</link>

<joint name="kinect\_joint" type="fixed">

<origin xyz="0.15 0 0.11" rpy="0 0 0"/>

<parent link="base\_link"/>

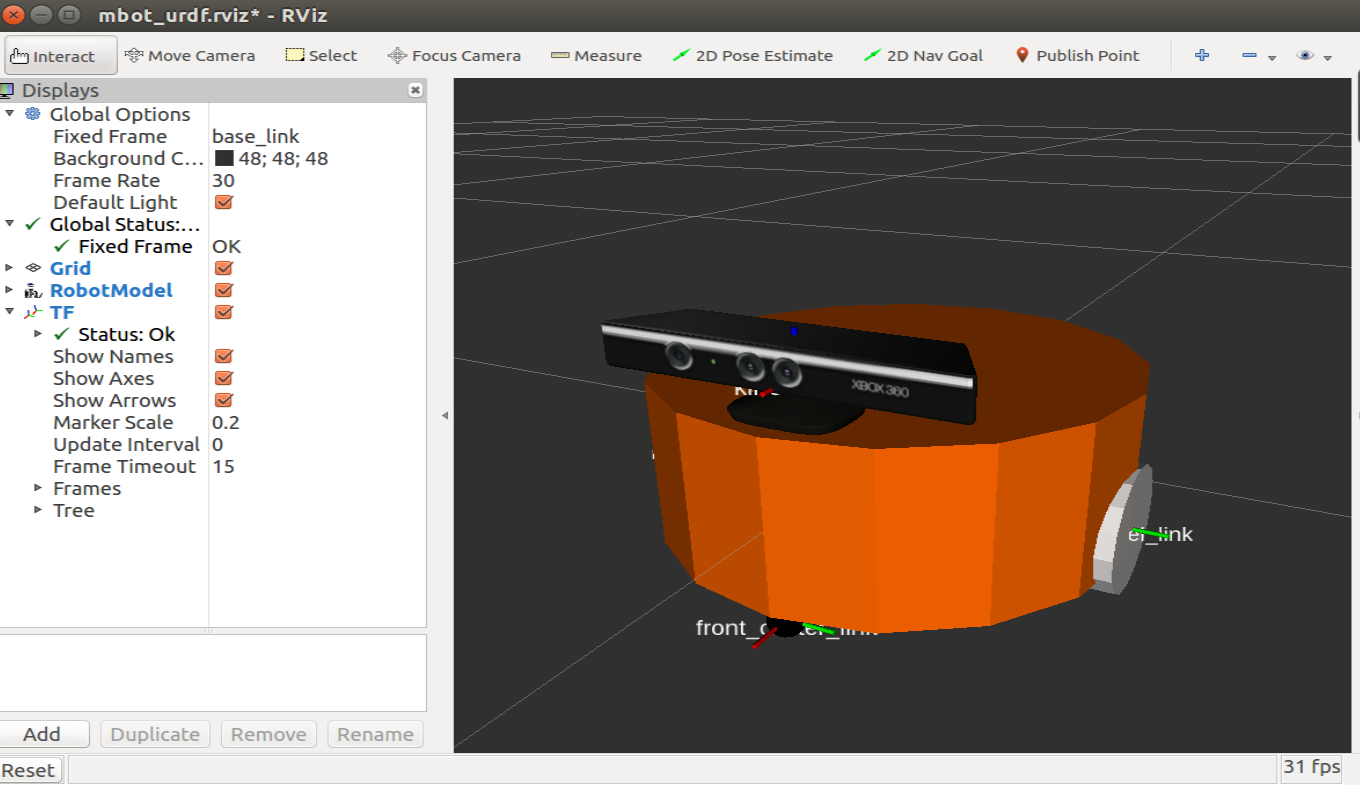
<child link="kinect\_link"/>

</joint>

<mesh filename="package://mbot\_description/meshes/kinect.dae" />

上一行代码就是kinect三维建模文件的引用

效果图如下所示：



后续也可以用solidwork导入urdf文件

建模完成以后可以用指令：

urdf\_to\_graphiz mbot.urdf

来检查标签是否闭合，即标签是否成对出现，也可以检查模型的整体结构

代码链接来自古月居：

码云（国内）：https://gitee.com/guyuehome/guyueclass/tree/main/ros\_basic

github（国外）：https://github.com/guyuehome/guyueclass/tree/main/ros\_basic