**单目USB摄像和Intel Realsense D435i 头+ Apriltag\_ros配置和实现对相机姿态的估计**

**前言**

配置平台：

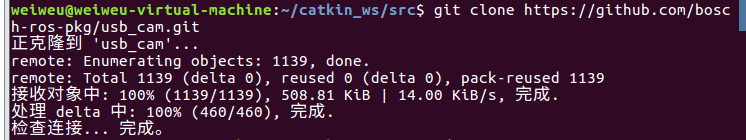
1. ROS版本：noetic/melodic
2. Ubuntu：20.04LTS/18.04LTS

**一：AprilTag\_ros的配置**

**1.1 usb摄像头的安装和使用**

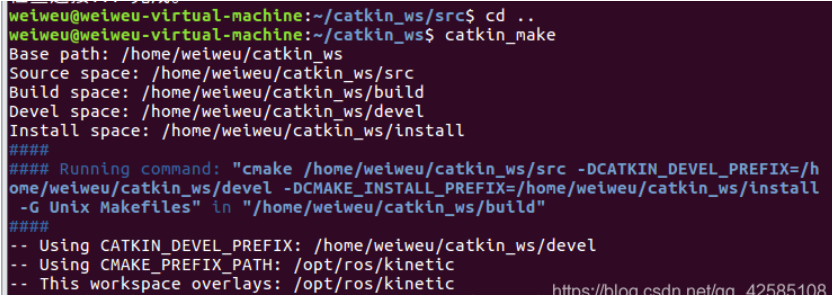
1. **下载源码**

|  |
| --- |
| Bash cd catkin\_ws/src #安装在一个ROS工作空间上 git clone https://github.com/bosch-ros-pkg/usb\_cam.git #下载源码 |



2. **编译工作空间**

|  |
| --- |
| Bash cd .. *#回到工作空间* ctakin\_make *#进行编译* |



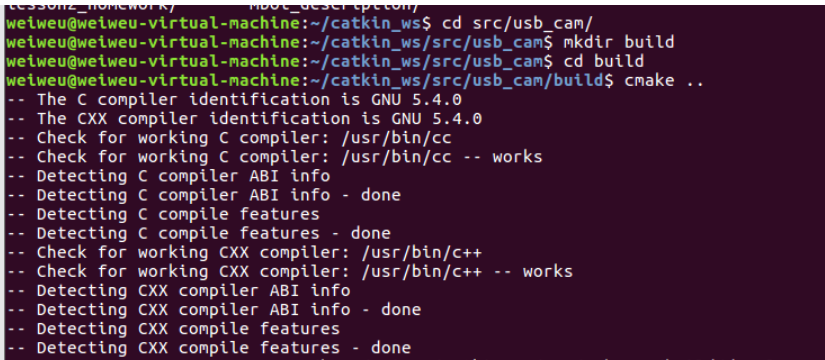
3. **添加usb\_cam属性文件（用于终端窗口可检索到usb\_cam功能包）**

|  |
| --- |
| Bash source ./devel/setup.bash *#每打开一次工作空间窗口都要* |

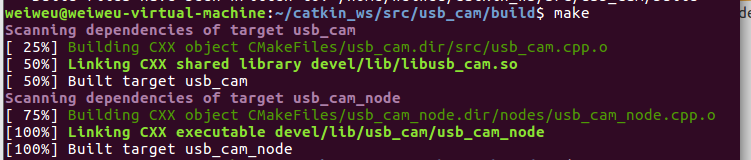
4. **编译usb\_cam源码**

回到工作空间的src文件中，再去usb\_cam中进行编译

|  |
| --- |
| Bash cd usb\_cam mkdir build cd build cmake .. ##注意后面的两点 |



|  |
| --- |
| Bash make |



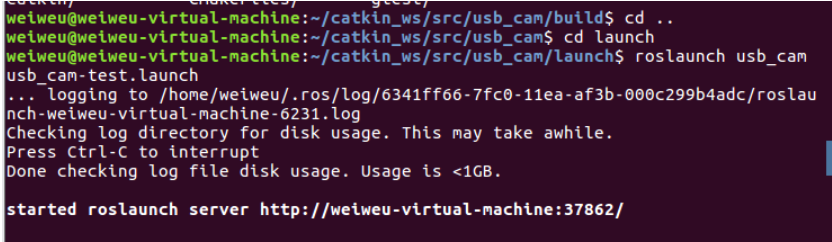
5. **测试USB摄像头**

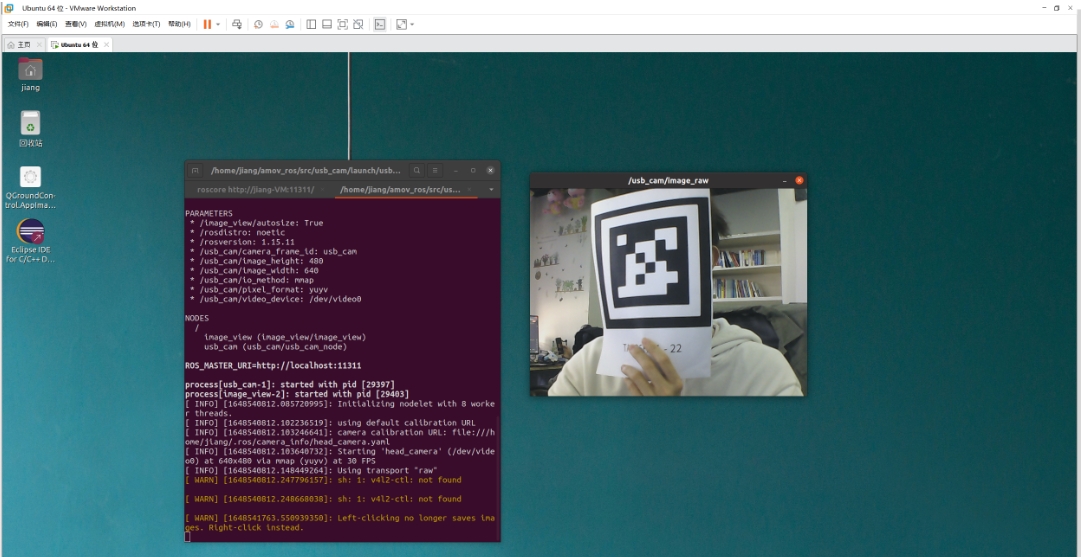
1）运行ROS

|  |
| --- |
| Bash roscore |

1. 回到usb\_cam的launch文件夹中，运行usb\_cam的功能包

|  |
| --- |
| Bash cd launch roslaunch usb\_cam usb\_cam-test.launch |





**1.2 AprilTag\_ros包的安装**

1. **安装依赖库apriltag**

|  |
| --- |
| 安装方法如下。这个注意不要把它放到ROS的工作空间里 |

|  |
| --- |
| Bash git clone https://github.com/AprilRobotics/apriltag.git # 这个不是ros的功能包 |

1. **编译依赖库apriltag**

进入[apriltag](https://so.csdn.net/so/search?q=apriltag&spm=1001.2101.3001.7020) 文件夹中，然后新建文件夹build

|  |
| --- |
| Bash mkdir build cd build cmake .. ## 注意两点 make sudo make install #安装到ros中 |

1. **安装AprilTag\_ros包**

将源码拷贝到你的工作空间中的src工作目录下

|  |
| --- |
| Bash git clone https://github.com/AprilRobotics/apriltag\_ros.git #在src文件夹内 |

1. **回到工作空间进行编译**

|  |
| --- |
| Bash catkin\_make |

**1.3 单目摄像机的标定**

摄像机标定是通过寻找对象在图像与现实世界的**转换数学关系**，找出其定量的联系，从而实现从图像中测量出现实中实际数据的目的，基于此才能实现后面的位姿检测。

1. **安装标定功能包**

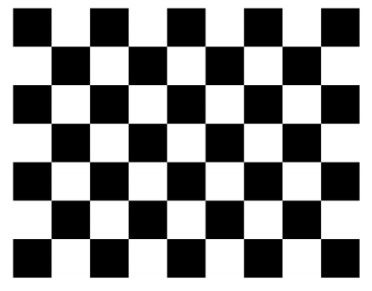
|  |
| --- |
| Bash sudo apt-get install ros-melodic-camera-calibration |

1. **打开摄像头进行标定**

|  |
| --- |
| Bash roslaunch usb\_cam usb\_cam-test.launch #启动摄像头 rosrun camera\_calibration cameracalibrator.py --size 8x6 --square 0.024 image:=/usb\_cam/image\_raw camera:=/usb\_cam |

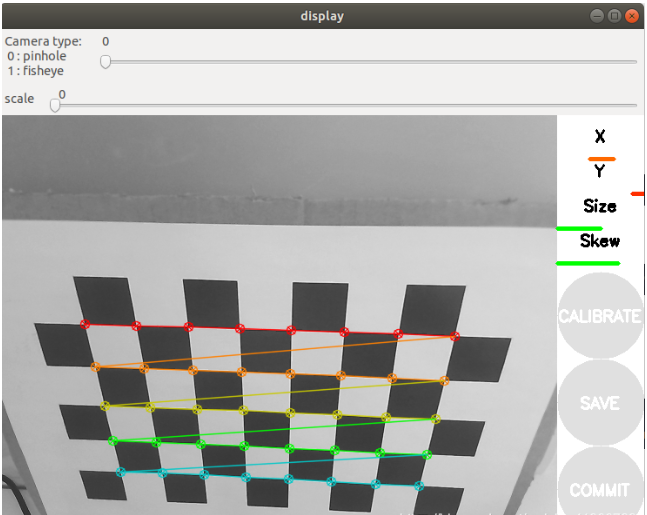
*第二条命令参数说明：  
size：棋盘内交叉点的个数，行\*列  
square：一个格子的边长，单位是m  
image：订阅摄像头发布的图像话题（ROS topic）  
camera：寻找相应的设备相机名（现实情况应该是/dev，仿真的话，不清楚）*

摄像机的校准是以一个由黑白方块组成的棋盘为基准进行的，如图8-8所示。从下面的地址下载8x6国际象棋棋盘，并打印出来后将其贴到一个平坦的纸箱。有时也会打印成超过1米的棋盘，但这里用的是A4纸。作为参考，8x6棋盘横向有9个方块，所以有8个交叉点，而竖向有7个方块，有6个交叉点，所以它被称为8x6棋盘。



**[该类型的内容暂不支持下载]**

1. **开始标定**



上/下/左/右移动摄像头，等各个方向绿条满了之后，标定完成时，CALIBRATE会有颜色，点击它，然后点击save，再点击COMMIT。这是会自动保存标定文件。

点击save 后，标定文件会自动保存在/tmp路径下，文件名为calibrationdata.tar.gz压缩包

1. **输出标定参数**

[1]

|  |
| --- |
| Bash  writing calibration data to /home/jiang/.ros/camera\_info/head\_camera.yaml #ros默认自动找这个路径的文件 |

**用该方法生成的标定参数，此后每次打开相机就会引用head\_camera.yaml 文件**

[2]

修改校准参数文件，把ost.txt文件改名为ost.ini，并使用camera\_calibration\_parsers功能包的convert节点创建相机参数文件（camera.yaml）。

[3]

创建完成后，将其保存在~/.ros/camera\_info/目录中，则ROS中使用的相机相关功能包会引用此信息。

|  |
| --- |
| Bash mv ost.txt ost.ini #修改名字 rosrun camera\_calibration\_parsers convert ost.ini camera.yaml #转换成相机引用信息  mkdir ~/.ros/camera\_info #创建相机信息文件夹 mv camera.yaml ~/.ros/camera\_info/ #移动到该文件下 |

[4]

重新运行相机节点，打印标定信息：

|  |
| --- |
| Bash rostopic echo /camera\_info |

|  |
| --- |
| 摄像机标定完成 |

**二：AprilTag\_ros的使用**

**2.1 AprilTag\_ros的使用（单目相机的使用）**

**2.1.1 编辑运行的配置文件**

进入apriltag\_ros/apriltag\_ros/config文件夹中，会看到两个文件

**1）settings.yaml （配置apriltag算法）**

|  |
| --- |
| Bash tag\_family: 'tag36h11' # options: tagStandard52h13, tagStandard41h12, tag36h11, tag25h9, tag16h5, tagCustom48h12, tagCircle21h7, tagCircle49h12 #支持单一标签类型 tag\_threads: 2 # default: 2 # 设置Tag\_Threads允许核心APRILTAG 2算法的某些部分运行并行计算。 典型的多线程优点和限制适用 tag\_decimate: 1.0 # default: 1.0 #减小图像分辨率 tag\_blur: 0.0 # default: 0.0 #设置tag\_blur> 0模糊图像，tag\_blur < 0锐化图像 tag\_refine\_edges: 1 # default: 1 #增强了计算精度，但消耗了算力 tag\_debug: 0 # default: 0 #1为保存中间图像到~/.ros max\_hamming\_dist: 2 # default: 2 (Tunable parameter with 2 being a good choice - values >=3 consume large amounts of memory. Choose the largest value possible.) # Other parameters publish\_tf: true # default: false #发布tf坐标 |

|  |
| --- |
| 这里有**两点注意**： I. tag\_family:时设置检测的标签类型（二维码图片类型），一般用’tag36h11’ 。 openmv的IDE可以生成这个标签。 II. 如果想发布TF坐标，需要把publish\_tf设置为true。 |

**2）tags.yaml （定义要查找的标签和标签簇）**

|  |
| --- |
| Bash # 这个标签中至少要填入三个参数 standalone\_tags:  [  {id: 1, size: 0.05}, # id 对应于生成标签的ID号，size对应标签的大小  {id: 2, size: 0.05},  {id: 22, size: 0.05},  {id: 45, size: 0.05},    ]  #这个标签是根据大小和位置检测的  tag\_bundles:  [  {  name: 'my\_bundle',  layout:  [  {id: 10, size: 0.05, x: 0.0000, y: 0.0000, z: 0.0, qw: 1.0, qx: 0.0, qy: 0.0, qz: 0.0} # id 对应于生成标签（二维码）的ID号，size对应标签的大小  {id: 22, size: 0.05, x: 0.0000, y: 0.0000, z: 0.0, qw: 1.0, qx: 0.0, qy: 0.0, qz: 0.0}， # 我选用的是22和45号标签  {id: 45, size: 0.05, x: 0.0000, y: 0.0000, z: 0.0, qw: 1.0, qx: 0.0, qy: 0.0, qz: 0.0}， # id 对应于生成标签（二维码）的ID号，size对应标签的大小  ]  }   ] |

注意添加你要设定的二维码标签的ID，和size（单位是 m），后面的 x,y,z 等表示的是坐标

**2.1.2 修改launch 文件**

这里主要是为了启用对应的相机话题（topic）和图像话题（topic）

进入apriltag\_ros/apriltag\_ros/launch文件夹中，会看到三个launch文件，主要修改这个**continuous\_detection.launch文件**

|  |
| --- |
| Bash  <arg name="camera\_name" default="/usb\_cam" /> <arg name="camera\_frame" default="usb\_cam" /> #发布一个坐标系  <arg name="image\_topic" default="image\_raw" /> |

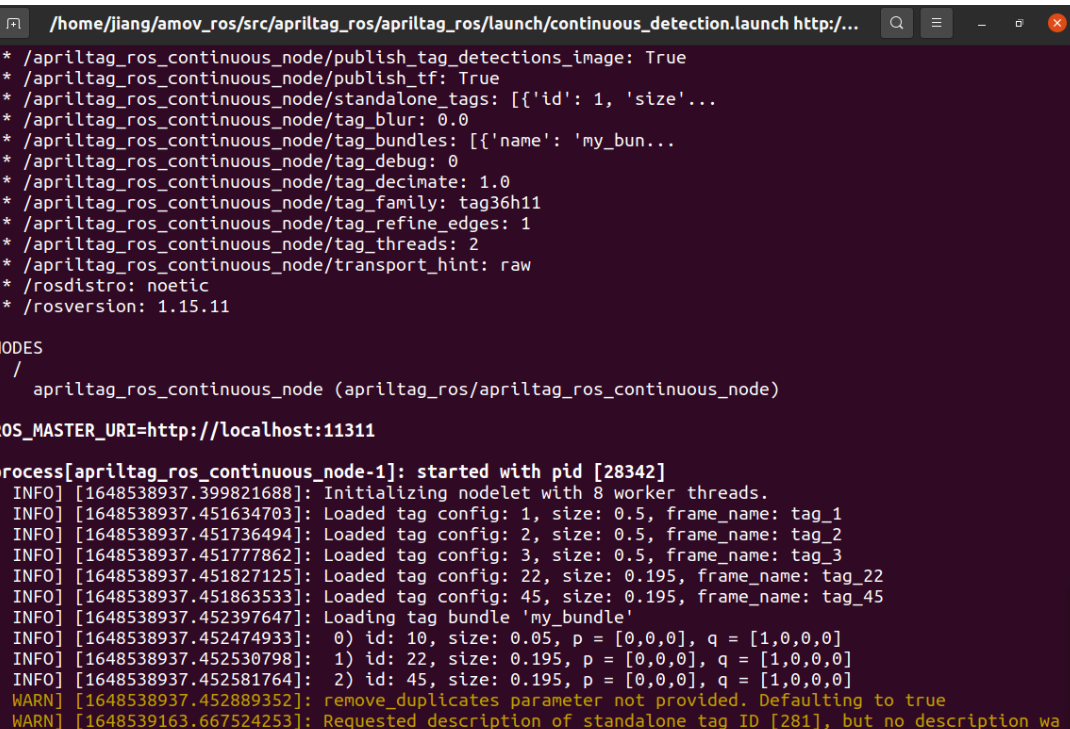
**2.1.3 使用功能包**

**1） 先打开相机**

|  |
| --- |
| Bash roslaunch usb\_cam usb\_cam-test.launch #启动摄像头 |

**2）运行功能包**

|  |
| --- |
| Bash roslaunch apriltag\_ros continuous\_detection.launch |

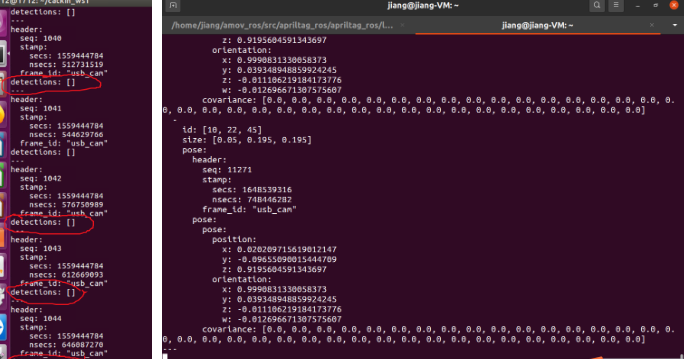


**3）运行标签检测节点**

这是查看检测标签的各个参数，如坐标（xyz），方向（xyzw），成功检测到标签会出现相应的id，size（单位为米）， pose等信息，实时显示相机的位姿信息，注意看下图

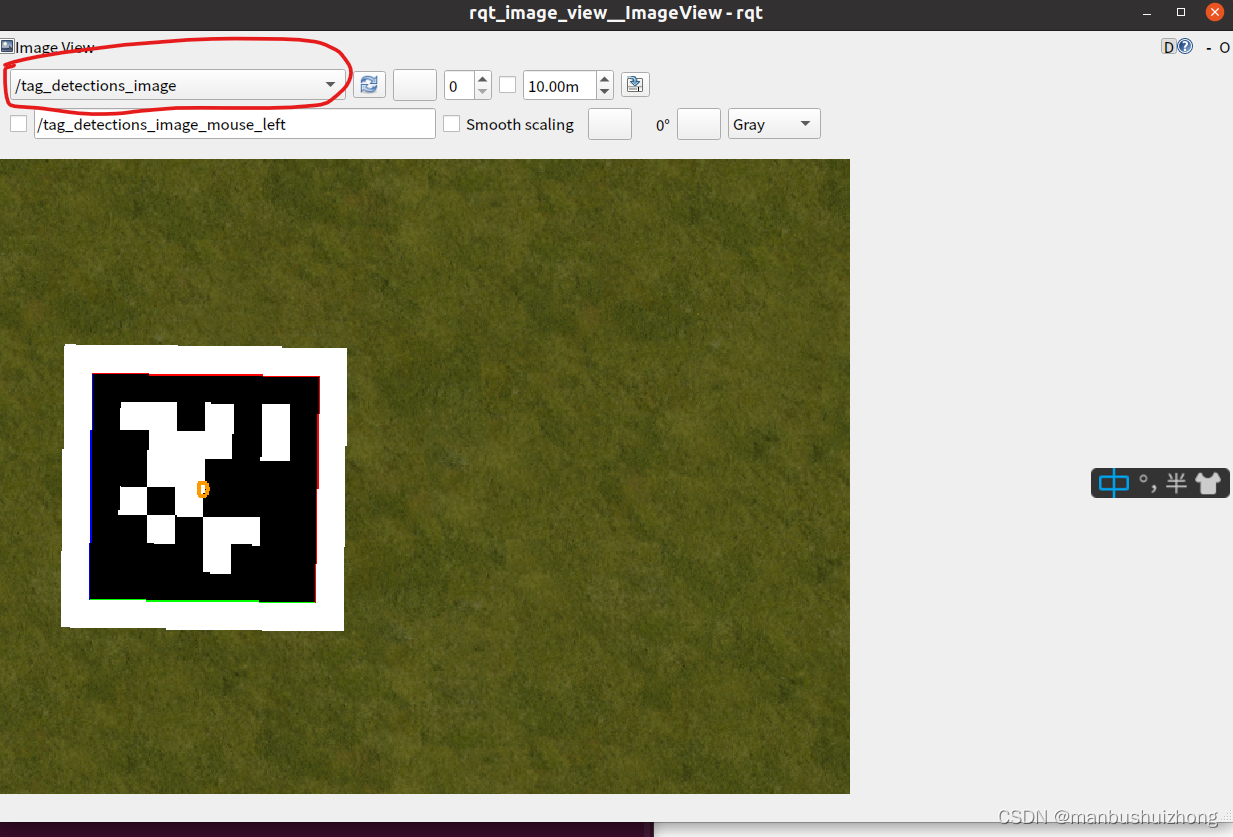
|  |
| --- |
| Bash rostopic echo /tag\_detections |

|  |
| --- |
| 左侧：未检测到 右侧：检测到 |



**4）可视化标签坐标**

运行如下命令，新开一个Terminal，先打开rqt\_image\_view,然后在里面选择tag\_detections\_imgage 话题，就可以看到apr检测的图像



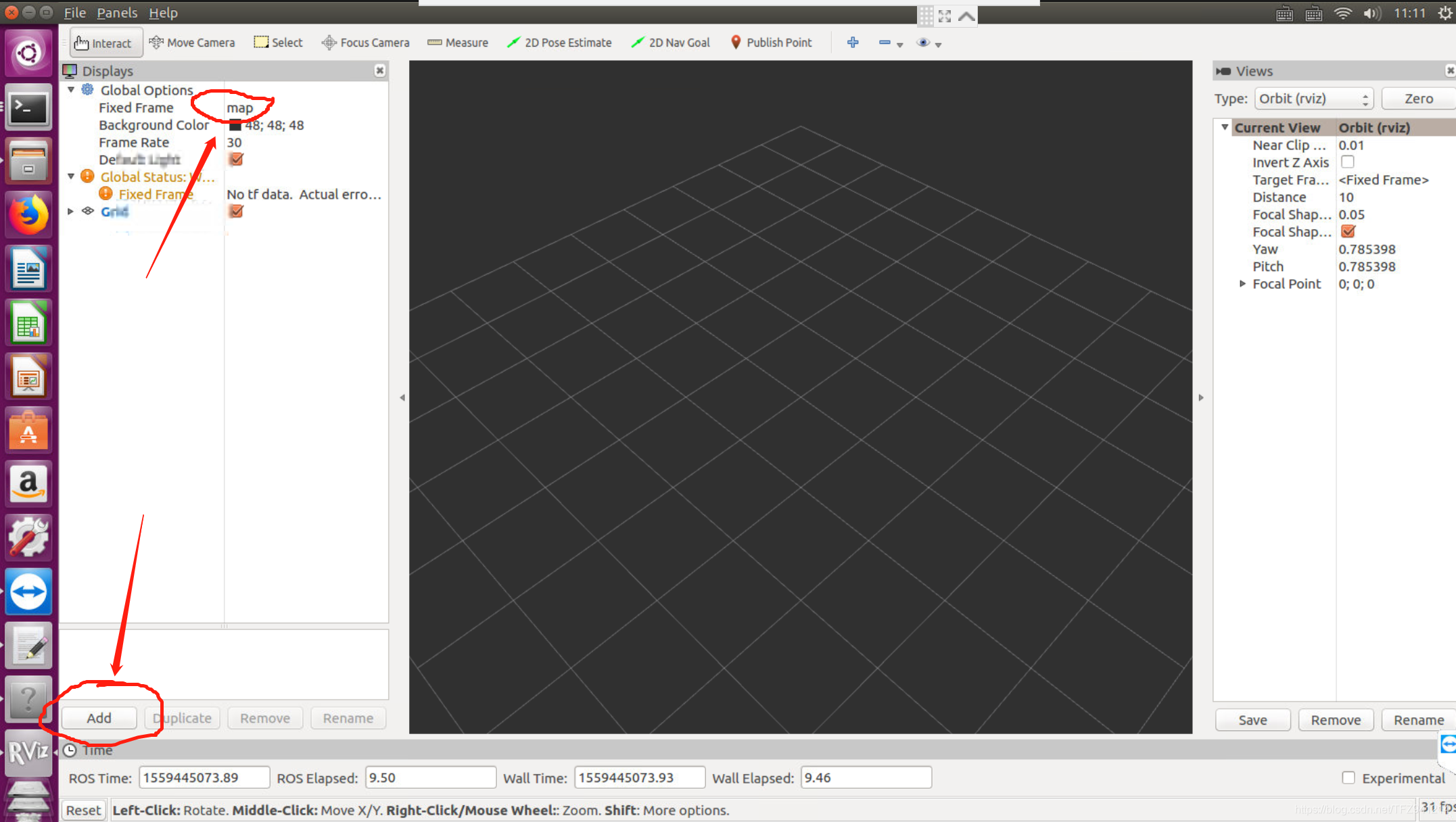
也可以在terminal中输入以下命令，获取话题的具体消息

|  |
| --- |
| Bash rostopic echo /tag\_detections\_image |

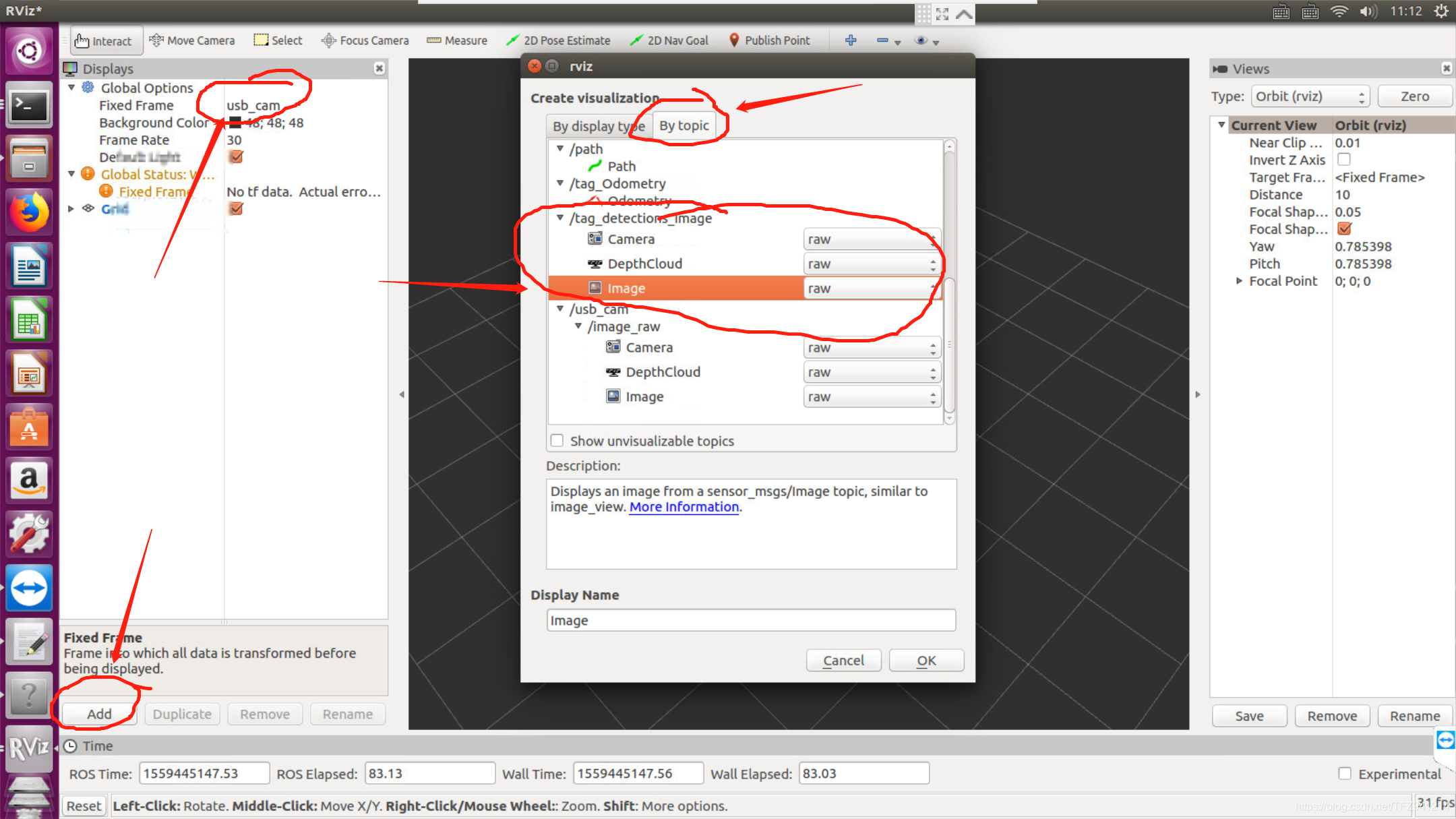
**5）图形化rviz显示**

|  |
| --- |
| Bash rosrun rviz rviz |

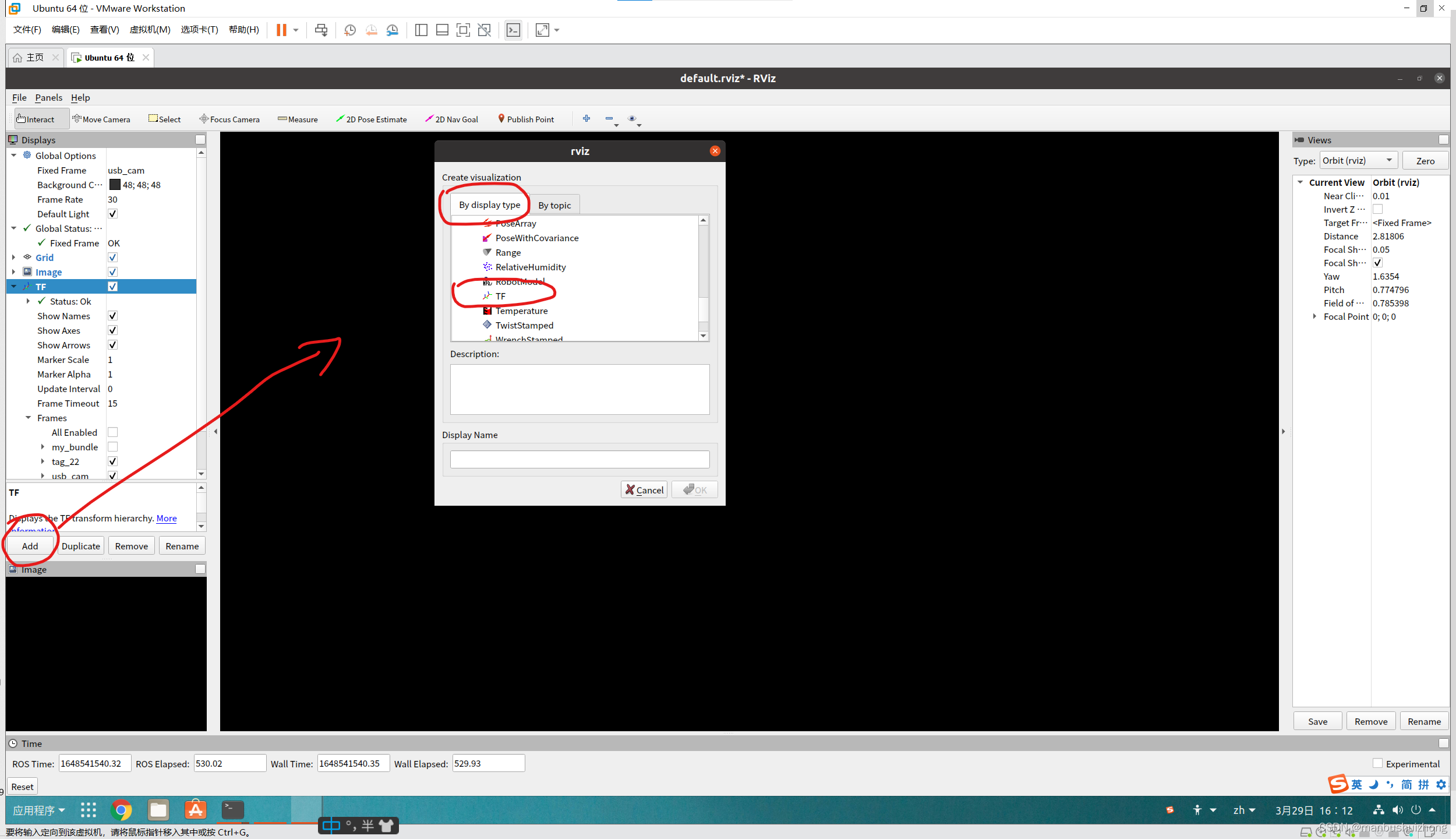
这是rviz 的初始界面。



将红圈的map改为usb\_cam，点击add按钮添加话题，首先把二维码（标签）检测的**话题**添加进去



二维码检测的话题添加进去后在左下角就可以看到被标记出来的二维码标签，再将TF坐标显示（不是话题！）添加进去。

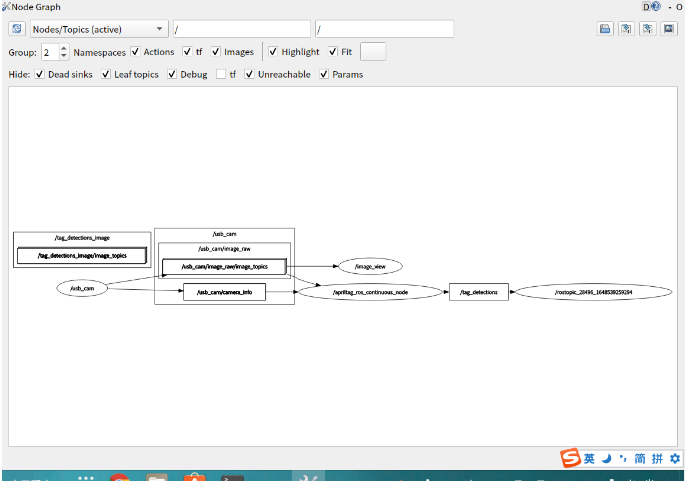


出现下图，然后就可以查看标签位姿了 。

**6）查看话题发布与订阅的关系信息系**

可以查看话题发布与订阅的关系信息系，新开一个Terminal（终端），运行如下命令

|  |
| --- |
| Bash rosrun rqt\_graph rqt\_graph |



**2.2 AprilTag\_ros的使用（D435i的使用）**

**2.2.1 启动realsense来查看有哪些话题(前提是装了D435i的驱动)**

|  |
| --- |
| 将 realsense d435i 连接上电脑，一定要用 usb 3.0 的线，否则无法启动。 |

|  |
| --- |
| Bash $ source devel/setup.bash $ roslaunch realsense2\_camera rs\_camera.launch |

另开一个窗口查看现在激活的话题。

|  |
| --- |
| Bash $ rostopic list |

如果你的realsense正确打开了的话会出现一系列的内容，这里截取了其中一小部分：

|  |
| --- |
| Bash /camera/color/camera\_info /camera/color/image\_raw /camera/color/image\_raw/compressed /camera/color/image\_raw/compressed/parameter\_descriptions |

其中 /camera/color/camera\_info 和 /camera/color/image\_raw 是 apriltag\_ros 所需要的，前者是相机信息，后者是realsense采集到的彩色三通道图片。

|  |
| --- |
| 注意：apriltag\_ros 只能使用三通到图片，有深度的图片是无法使用的。 |

**2.2.2 配置 apriltag\_ros 的配置文件**

1. **配置aprilatg\_ros/config/setting.yaml**

|  |
| --- |
| Bash tag\_family: 'tag36h11'  tag\_threads: 2  tag\_decimate: 1.0  tag\_blur: 0.0  tag\_refine\_edges: 1  tag\_debug: 0  max\_hamming\_dist: 2   publish\_tf: true transport\_hint: "raw" |

**tag\_family**:

*使用的是什么种类的二维码，这个参数一定要正确，否则是无法检测出图像的。如何判断自己二维码是什么类型，在他的链接中 https://github.com/AprilRobotics/apriltag\_ros Tag Size Definition 部分可以看到并排的6个二维码，下面的字符串（如：“Tag36h11” 和 “TagCircle21h7”）就是二维码类型，直接替换即可。其他参数使用默认值即可。*

2. **配置 tags.yaml**

这个文件只有两个标签可以编写。

|  |
| --- |
| Bash standalone\_tags:  [  {id: 001, size: 0.75, name: tag\_1}  ]  tag\_bundles:  [  {  name: 'tag\_1',  layout:  [  {id: 001, size: 0.75, x: 0, y: 0, z: 0, qw: 1, qx: 0, qy: 0, qz: 0}  ]  }  ] standalone\_tags:  [  {id: 001, size: 0.75, name: tag\_1}  ]  tag\_bundles:  [  {  name: 'tag\_1',  layout:  [  {id: 001, size: 0.75, x: 0, y: 0, z: 0, qw: 1, qx: 0, qy: 0, qz: 0}  ]  }  ] |

*这里面写要使用多少个二维码，apriltag\_ros 允许一张图片中出现多个二维码，但一定要明确每个二维码类型，如果想要添加的话这样写就可以，但记得两个标签都要添加:*

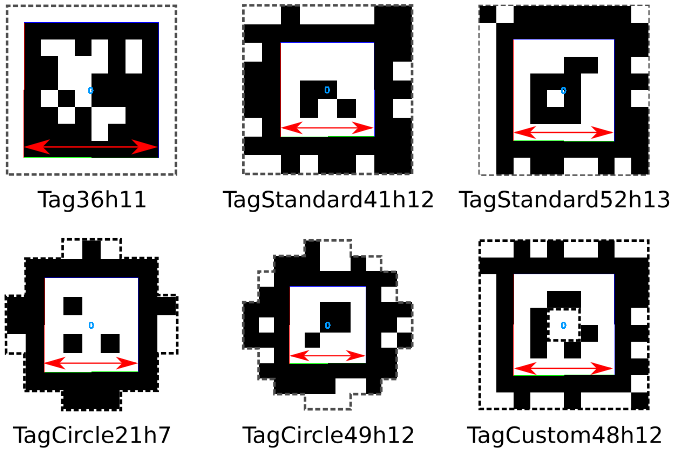
|  |
| --- |
| Bash standalone\_tags:  [  {id: 001, size: 0.75, name: tag\_1},  {id: 002, size: 0.20, name: tag\_2},  ...  ]  tag\_bundles:  [  {  name: 'tag\_1',  layout:  [  {id: 001, size: 0.75, x: 0, y: 0, z: 0, qw: 1, qx: 0, qy: 0, qz: 0}  ]  },  {  name: 'tag\_2',  layout:  [  {id: 002, size: 0.20, x: 0, y: 0, z: 0, qw: 1, qx: 0, qy: 0, qz: 0}  ]  }  ] |

**standalone\_tags解释：**

*id：你给每个二维码的编号，可以从任意数字开始，只要你自己能区分哪个号是哪个二维码就好；*

*size：二维码的长度。这个值是需要手动测量出来的，不同类型的二维码测量方式不同，具体可以看他的链接 https://github.com/AprilRobotics/apriltag\_ros Tag Size Definition 部分，红色箭头就是你需要手动侧脸的二维码长度，单位是米，然后填写到这里；*

*name：和id一样，这是为了更好地区分可以任起；*



**tag\_bundles注释：**

*name：和每个tag保持一致；*

*id：和每个tag保持一致；*

*size：和每个tag保持一致；*

*x,y,z：这个指的是该二维码中心点坐标，在推算相机姿态的时候就是以这个数为坐标原点进行计算的。假设相机垂直于这个中心点1m，那么最后返回的pose中x，y，z=[0,0,1]；*

*qw,qx,qy,qz：四元数，和上面的x,y,z效果差不多，如果二维码贴在了一个斜面上，那么这里一定要修改，否则计算出来的pose就会差出很多。*

3. **修改 continuous\_detection.launch 文件**

打开 apriltag\_ros/launch/continuous\_detection.launch 文件。

需要修改的主要有两个标签：

* arg name=“camera\_name”：相机话题的前缀，如 /camera/color
* arg name=“image\_topic”：相机发布的图像话题，如 image\_raw

这两个值是在 rostopic list中查看相机发布的话题中看见，如果你用的不是realsense，那么需要按照相机包发布出来的话题名修改，这里用的是默认设置的 realsense 相机话题。

注意：“camera\_name” 一定只能用前缀，如果多加了 “/” 会导致算法订阅到的话题变成了 “/camera/color//image\_raw” 这样是不会出数据的，因为后面其实是做了一个字符串拼接：

|  |
| --- |
| Bash <remap from="image\_rect" to="$(arg camera\_name)/$(arg image\_topic)" /> <remap from="camera\_info" to="$(arg camera\_name)/camera\_info" /> |

最终修改如下：

|  |
| --- |
| Bash <launch>  <arg name="launch\_prefix" default="" /> <!-- set to value="gdbserver localhost:10000" for remote debugging -->  <arg name="node\_namespace" default="apriltag\_ros\_continuous\_node" />   <!-- <arg name="camera\_name" default="/camera\_rect" /> -->  <!-- <arg name="image\_topic" default="image\_rect" /> -->  <arg name="camera\_name" default="/camera/color" />  <arg name="image\_topic" default="image\_raw" />   <!-- Set parameters -->  <rosparam command="load" file="$(find apriltag\_ros)/config/settings.yaml" ns="$(arg node\_namespace)" />  <rosparam command="load" file="$(find apriltag\_ros)/config/tags.yaml" ns="$(arg node\_namespace)" />    <node pkg="apriltag\_ros" type="apriltag\_ros\_continuous\_node" name="$(arg node\_namespace)" clear\_params="true" output="screen" launch-prefix="$(arg launch\_prefix)" >  <!-- Remap topics from those used in code to those on the ROS network -->  <remap from="image\_rect" to="$(arg camera\_name)/$(arg image\_topic)" />  <remap from="camera\_info" to="$(arg camera\_name)/camera\_info" />   <param name="publish\_tag\_detections\_image" type="bool" value="true" /> <!-- default: false -->  </node> </launch> |

4. **启动 apriltag\_ros 算法**

|  |
| --- |
| Bash $ source devel/setup.bash $ roslaunch apriltag\_ros continuous\_detection.launch |

如果没有出现红色的报错说明启动成功了，如果有则检查 yaml 文件有没有出现多一个逗号或者省略号之类的。

然后再开一个窗口就可以订阅推算出来的话题：

|  |
| --- |
| Bash $ rostopic echo /tag\_detections |

|  |
| --- |
| 如果只出现这个说明二维码没有在相机视野内，或者前面二维码类型配置错误。 |

|  |
| --- |
| Bash --- header:   seq: 1739  stamp:   secs: 1664460149  nsecs: 591933966  frame\_id: "camera\_color\_optical\_frame" detections: [] --- |

正确检测到的应该会有下面的信息：

|  |
| --- |
| Bash --- header:   seq: 1612  stamp:   secs: 1664460138  nsecs: 717808008  frame\_id: "camera\_color\_optical\_frame" detections:   -   id: [0]  size: [0.075]  pose:   header:   seq: 3482  stamp:   secs: 1664460138  nsecs: 717808008  frame\_id: "camera\_color\_optical\_frame"  pose:   pose:   position:   x: 0.0454747857095  y: -0.107662586406  z: 0.540045888173  orientation:   x: 0.999320052973  y: 0.0235171876449  z: -0.0148883373629  w: -0.0241807986292  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  -   id: [0]  size: [0.078]  pose:   header:   seq: 3482  stamp:   secs: 1664460138  nsecs: 717808008  frame\_id: "camera\_color\_optical\_frame"  pose:   pose:   position:   x: 0.0472937771379  y: -0.111969089862  z: 0.5616477237  orientation:   x: 0.999320052973  y: 0.0235171876449  z: -0.0148883373629  w: -0.0241807986292  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] --- |

|  |
| --- |
| 注意：这里其实是有两个pose，仔细看两个pose标签中的 size 是不一样的，这个其实就是二维码边框一个像素的大小。  【1】如果你需要的是只算黑色部分就选小的pose  【2】如果你需要的是紧挨着黑色部分的白色像素，那就选大的pose。  有点类似内接圆和外切圆的关系 |