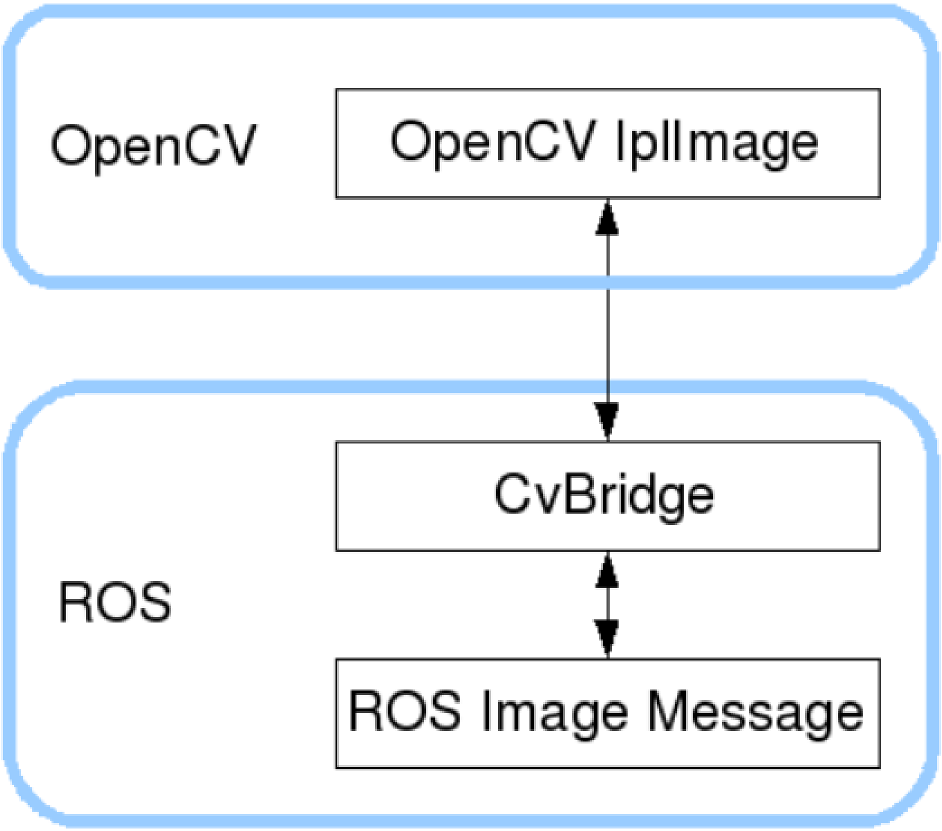
- # 5. ROS+Opencv Basics

Function package location: ~/orbbec_ws/src/astra_visual

5.1. Overview

ROS has integrated Opencv 3.0 and above during the installation process, so there is almost no need to consider the installation configuration. ROS transmits images in its own sensor_msgs/Image message format and cannot directly process images, but the provided [CvBridge] can perfectly convert and be converted image data formats. [CvBridge] is a ROS library, which is equivalent to a bridge between ROS and Opencv.

Opencv and ROS image data conversion is shown in the figure below:



Although the installation configuration does not require too much consideration, the use environment still needs to be configured, mainly the two files [package.xml] and [CMakeLists.txt]. This function package not only uses [CvBridge], but also requires [Opencv] and [PCL], so they are configured together.

- package.xml

[cv_bridge]: image conversion dependency package.

```
<build_depend>sensor_msgs</build_depend>
<build_export_depend>sensor_msgs</build_export_depend>
<exec_depend>sensor_msgs</exec_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>std_msgs</exec_depend>
<build_depend>cv_bridge</build_depend>
<build_export_depend>cv_bridge</build_export_depend>
<exec_depend>cv_bridge</exec_depend>
<exec_depend>image_transport</exec_depend>
```

- CMakeLists.txt

This file has a lot of configuration content. For specific content, please check the source file. The file location is: ~/orbbec_ws/src/astra_visual/CMakeLists.txt

5.2. Start the camera

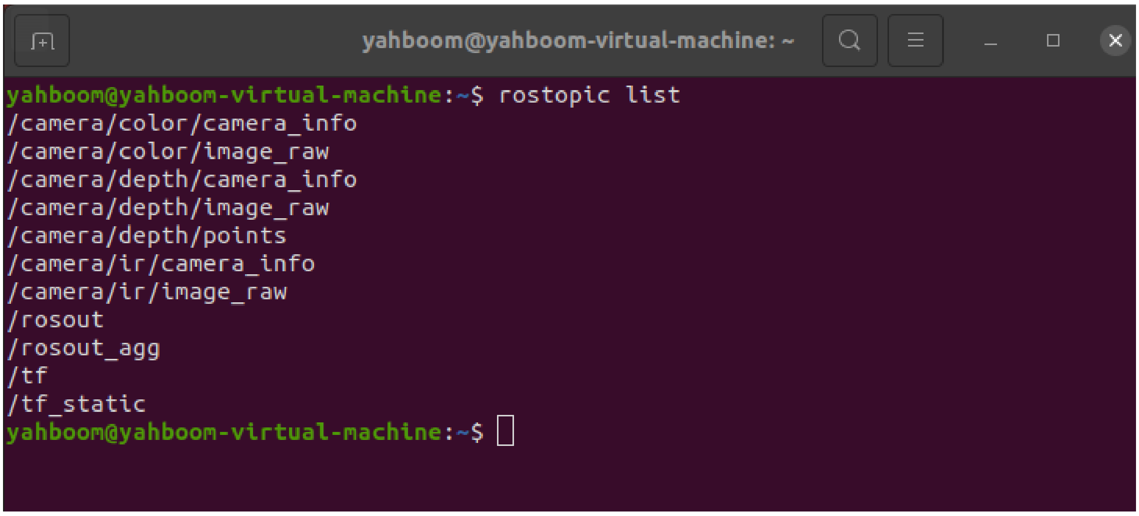
5.2.1. Start the camera

Terminal input,

```
roslaunch orbbec_camera orbbec_camera.launch
```

View the topic,

```
rostopic list
```



```
yahboom@yahboom-virtual-machine:~$ rostopic list
/camera/color/camera_info
/camera/color/image_raw
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/ir/camera_info
/camera/ir/image_raw
/rosout
/rosout_agg
/tf
/tf_static
yahboom@yahboom-virtual-machine:~$ 
```

The following are commonly used,

/camera/color/image_raw: RGB color image topic

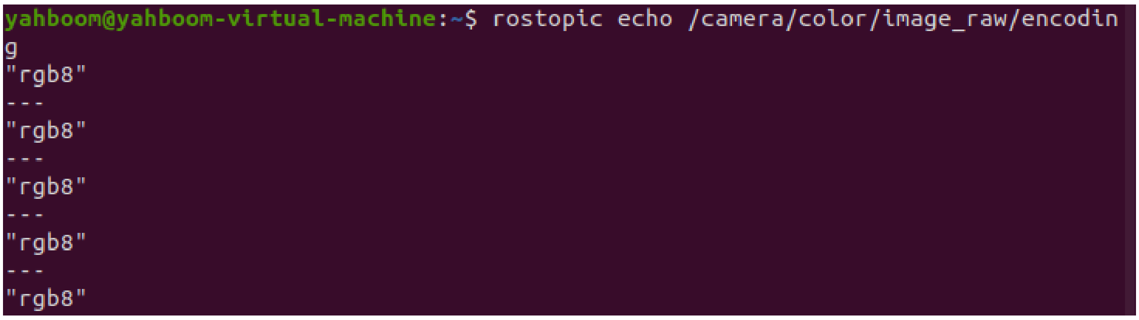
/camera/depth/image_raw: Depth depth image topic

/camera/ir/image_raw: IR infrared image topic

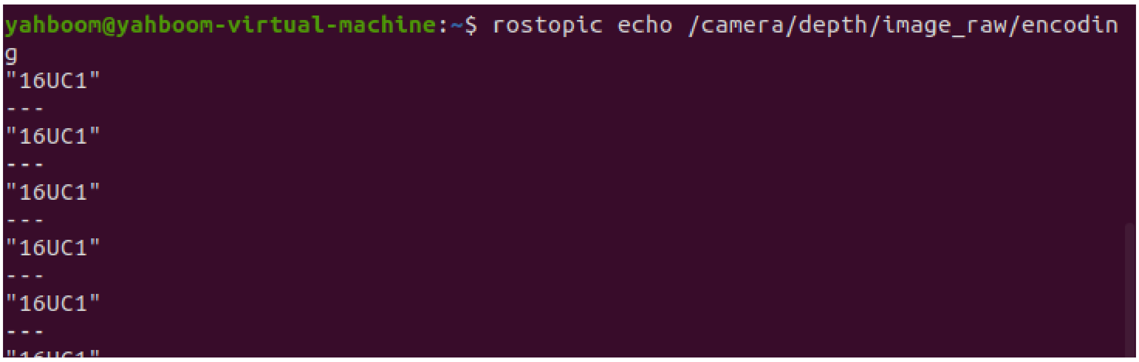
/camera/depth/points: Depth point cloud data topic

View the encoding format of the topic: rostopic echo + [topic] + encoding, for example,

```
rostopic echo /camera/color/image_raw/encoding
rostopic echo /camera/depth/image_raw/encoding
```



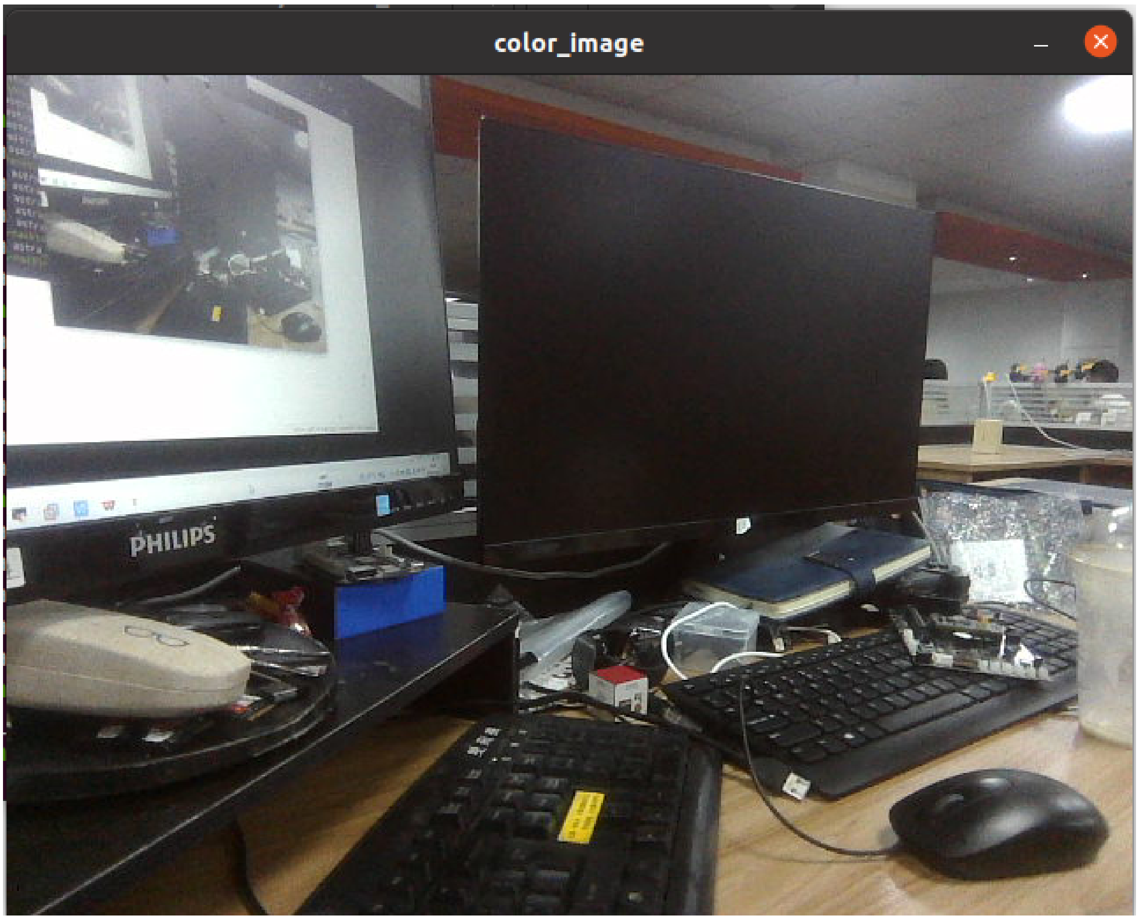
```
yahboom@yahboom-virtual-machine:~$ rostopic echo /camera/color/image_raw/encodin
g
"rgb8"
---
"rgb8"
---
"rgb8"
---
"rgb8"
---
"rgb8"
```



```
yahboom@yahboom-virtual-machine:~$ rostopic echo /camera/depth/image_raw/encodin
g
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
```

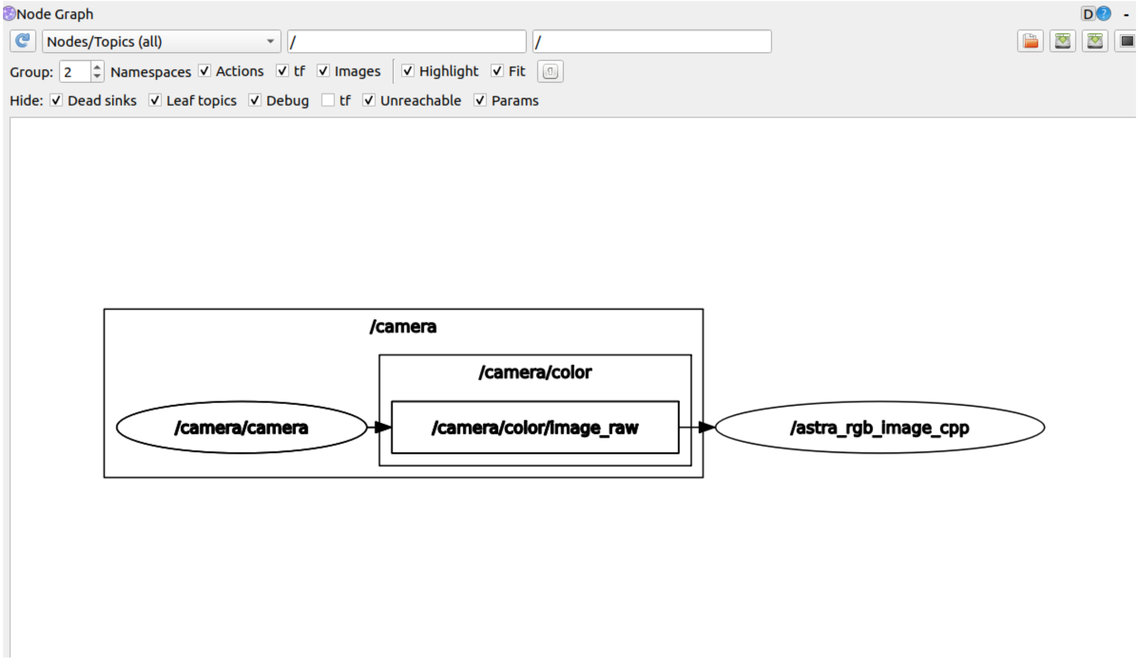
5.2.2, Start the color image subscription node

```
rosrun astra_visual astra_rgb_image.py # py
rosrun astra_visual astra_rgb_image # C++
```



View the node graph,

```
rqt_graph
```



Group: 2 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug □ tf ☑ Unreachable ☑ Params

- py code analysis

Create a subscriber: the subscribed topic is ["/camera/color/image_raw"], data type [Image], callback function

[topic()]

```
sub = rospy.Subscriber("/camera/color/image_raw", Image, topic)
```

Use [CvBridge] to convert data. Here, you should pay attention to the encoding format. If the encoding format is incorrect, the converted image will have problems.

```
bridge = CvBridge()
frame = bridge.imgmsg_to_cv2(msg, "bgr8")
```

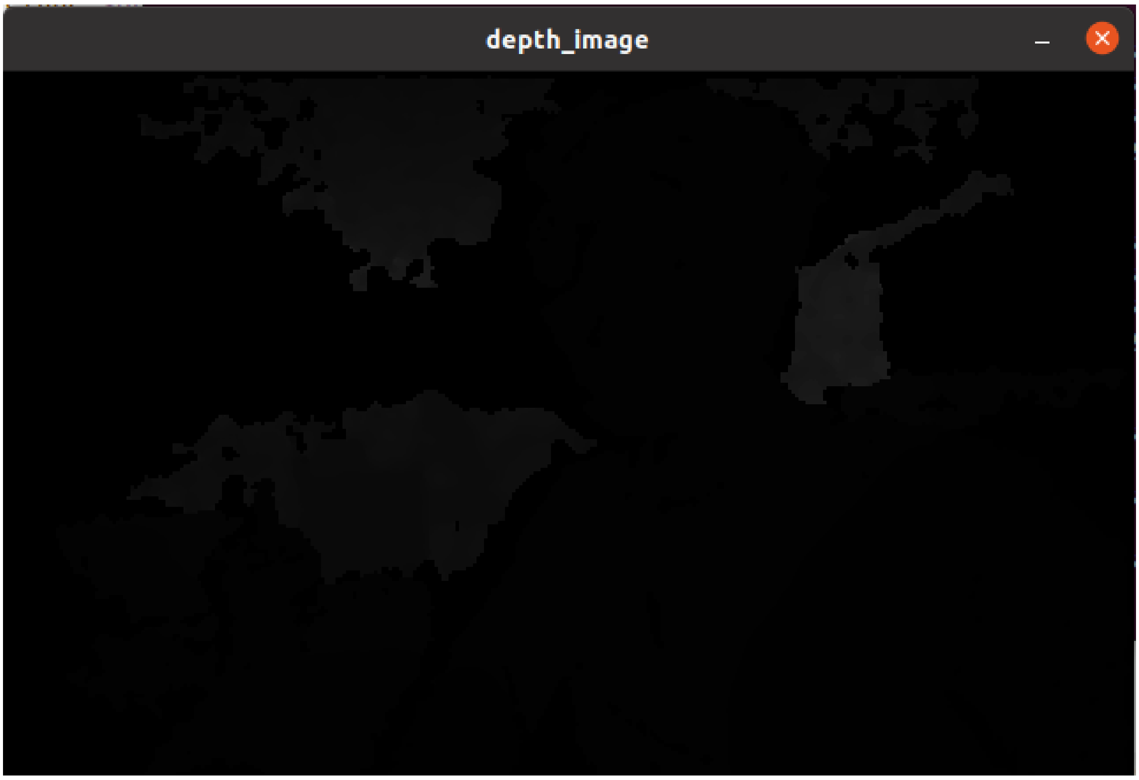
- c++ code analysis

Similar to py code

```
//Create a receiver
ros::Subscriber subscriber =
n.subscribe<sensor_msgs::Image>("/camera/color/image_raw", 10, RGB_Callback);
//Create cv_bridge example
cv_bridge::CvImagePtr cv_ptr;
//Data conversion
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
```

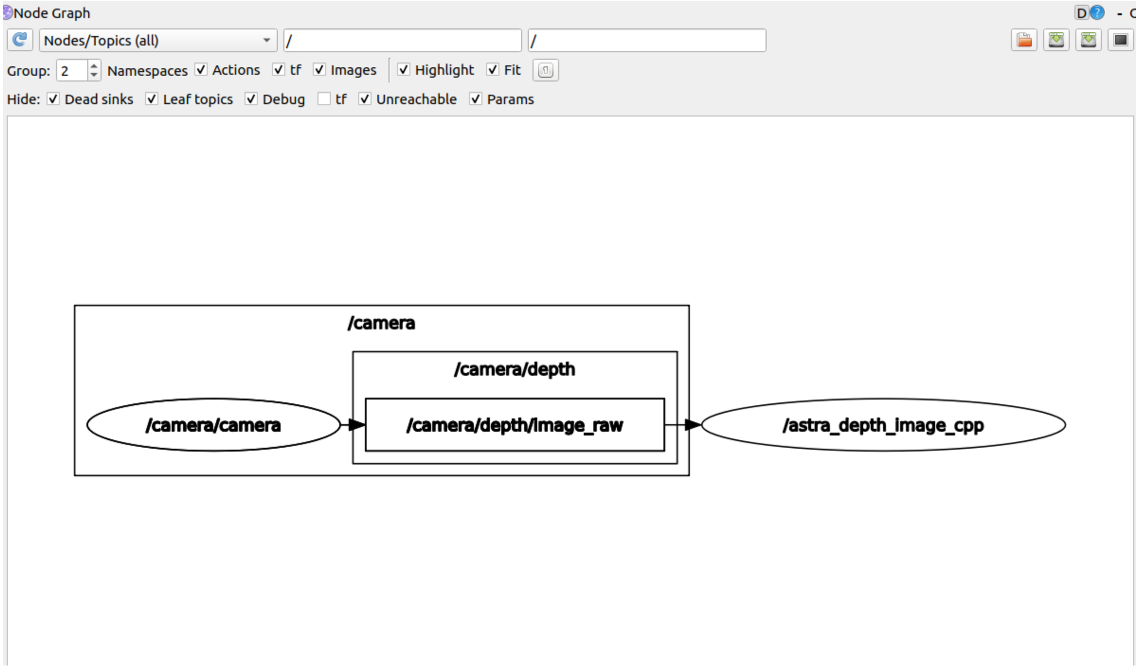
5.2.3, Start the depth map subscription node

```
#Start by selecting 1 of the following commands
rosrun astra_visual astra_depth_image.py # py
rosrun astra_visual astra_depth_image # C++
```



View the node graph,

```
rqt_graph
```


Node Graph

Nodes/Topics (all) / /

Group: 2 Namespaces ☑ Actions ☑ tf ☑ Images ☑ Highlight ☑ Fit

Hide: ☑ Dead sinks ☑ Leaf topics ☑ Debug □ tf ☑ Unreachable ☑ Params

- py code analysis

Create a subscriber: the subscribed topic is ["/camera/depth/image_raw"], the data type is [Image], and the callback function is [topic()]

```
sub = rospy.Subscriber("/camera/depth/image_raw", Image, topic)
```

Use [CvBridge] to convert data. Here, you should pay attention to the encoding format. If the encoding format is incorrect, the converted image will have problems.

```
# Encoding format
encoding = ['16UC1', '32FC1']
# You can switch different encoding formats to test the effect
frame = bridge.imgmsg_to_cv2(msg, encoding[1])
```

- c++ code analysis

Similar to py code,

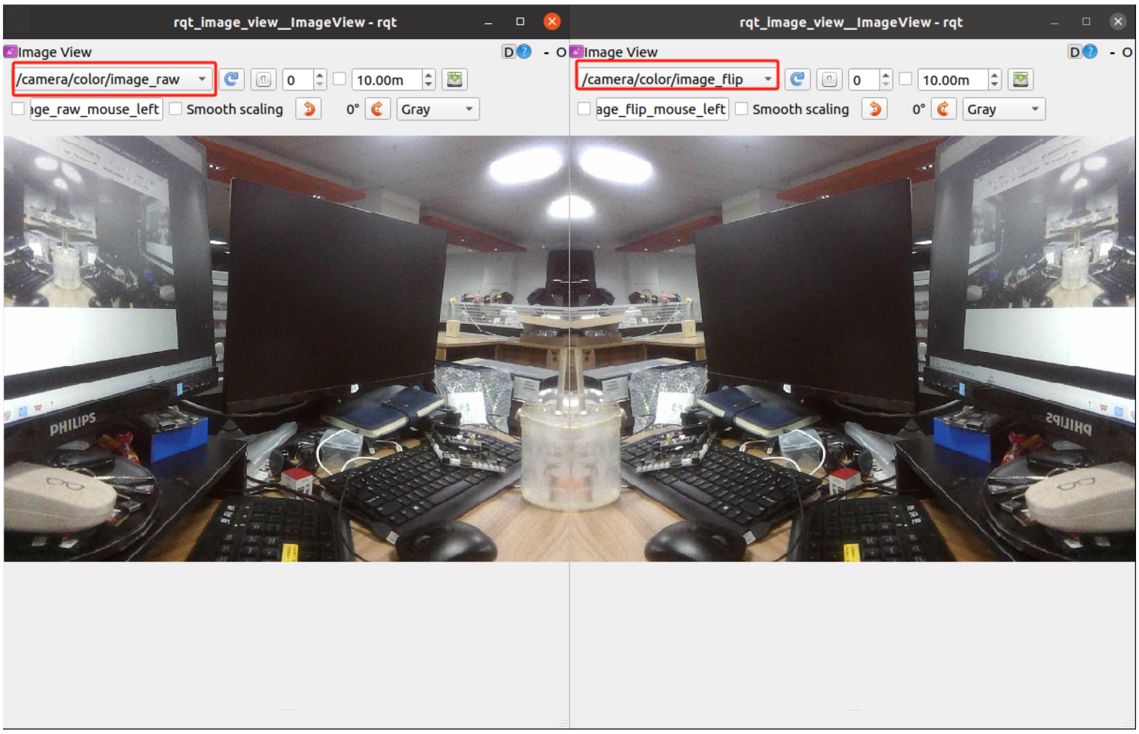
```
//Create a receiver.
ros::Subscriber subscriber = n.subscribe<sensor_msgs::Image>
("/camera/depth/image_raw", 10, depth_Callback);
// Create cv_bridge example
cv_bridge::CvImagePtr cv_ptr;
// Data conversion
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::TYPE_16UC1)
```

5.2.4, Start color image inversion

```
rosrun astra_visual astra_image_flip.py # py
```

Image view (open two), select different topics to display

```
rqt_image_view
```



/camera/color/image_raw

/camera/color/image_flip

- Code analysis

1), create subscribers

The subscribed topic is ["/camera/color/image_raw"], data type [Image], callback function [topic()].

2), create publishers

The published topic is ["/camera/rgb/image_flip"], data type [Image], queue size [10].

3. Callback function

```
# Normal image transmission processing
def topic(msg):
if not isinstance(msg, Image):
return
bridge = CvBridge()
frame = bridge.imgmsg_to_cv2(msg, "bgr8")
# Opencv image processing
frame = cv.resize(frame, (640, 480))
frame = cv.flip(frame, 1)
# opencv mat -> ros msg
msg = bridge.cv2_to_imgmsg(frame, "bgr8")
# Image processing is complete, publish directly
pub_img.publish(msg)
```