8. Basics of ROS+Opencv

This lesson takes the Astra camera as an example, and ordinary cameras are similar.

8.1. Overview

Wiki: http://wiki.ros.org/cv_bridge/

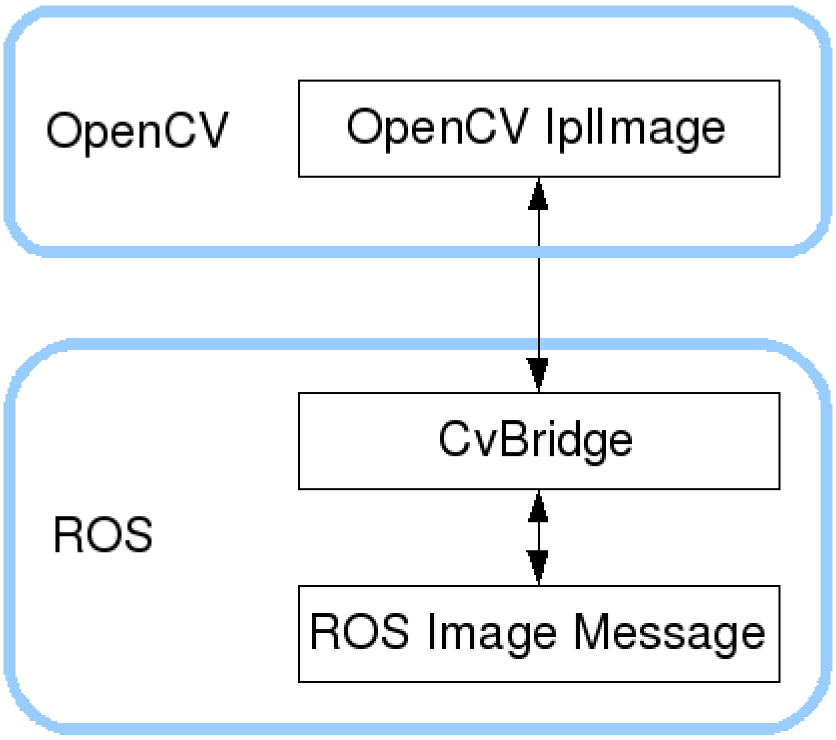
Tutorials: http://wiki.ros.org/cv_bridge/Tutorials

Source code: https://github.com/ros-perception/vision_opencv.git

Function package location: ~/yahboomcar_ws/src/yahboomcar_visual

ROS has integrated Opencv 3.0 and above during the installation process, so there is almost no need to consider the installation and 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.

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



Although the installation configuration does not require 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

Add the following content

```
  <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>
```

【cv_bridge】: Image conversion dependency package.

- CMakeLists.txt

This file has a lot of configuration content, please check the source file for specific content.

8.2, Astra

8.2.1, Start Astra camera/Gemin335 camera

```
sudo supervisorctl stop ChassisServer
#Start Astra camera
roslaunch orbbec_camera astra_pro2.launch
#Gemin335 camera
roslaunch orbbec_camera gemini_330_series.launch
```

View topic

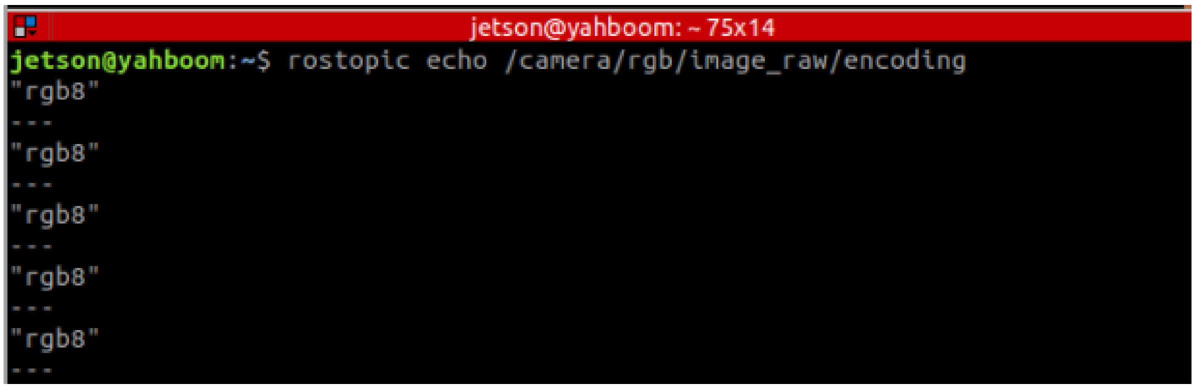
```
rostopic list
```

You can see that there are many topics, and only a few are commonly used in this section

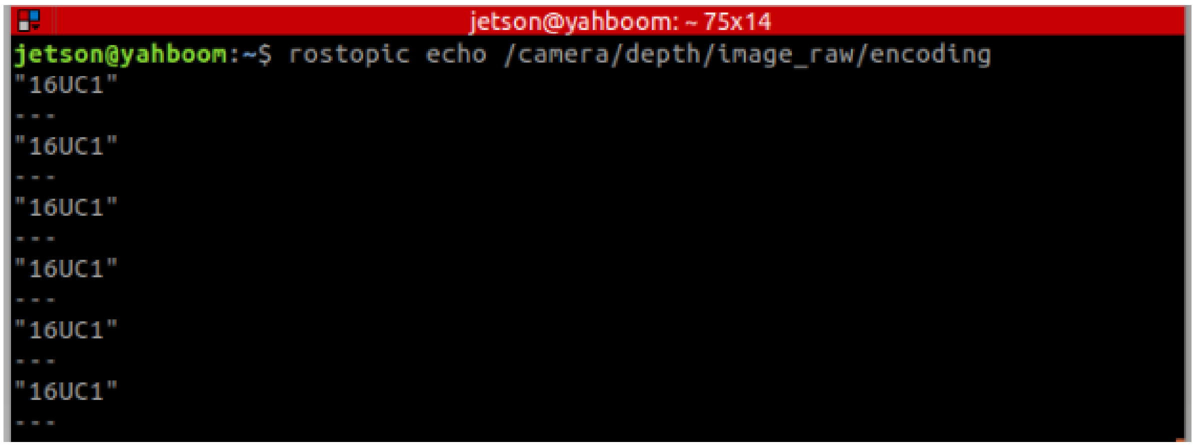
Topic name	Data type
/camera/depth/image_raw	sensor_msgs/Image
/camera/depth/image	sensor_msgs/Image
/camera/color/image_raw	sensor_msgs/Image
/camera/depth/image_raw/compressedDepth	sensor_msgs/CompressedImage
/camera/color/image_raw/compressed	sensor_msgs/CompressedImage

Check 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
```

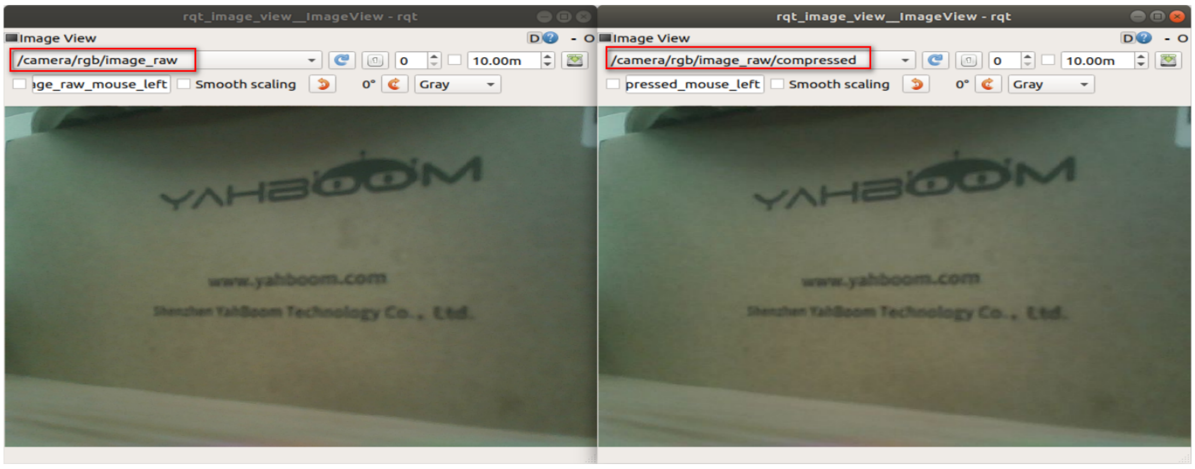


```
jetson@yahboom: ~ 75x14
jetson@yahboom:~$ rostopic echo /camera/rgb/image_raw/encoding
"rgb8"
---
"rgb8"
---
"rgb8"
---
"rgb8"
---
"rgb8"
---
```



```
jetson@yahboom: ~ 75x14
jetson@yahboom:~$ rostopic echo /camera/depth/image_raw/encoding
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
```

Topics followed by [compressed] or [compressedDepth] are compressed topics. When ROS transmits images, data packets may be lost due to factors such as the network, the main control running speed, the main control running memory, and the huge video stream data. Therefore, I can only subscribe to compressed topics. Open two images at the same time to subscribe to different topics for testing. If the device performance is good and the network is also good, no changes will be seen. Otherwise, you will find that the topic will be much smoother after the image is compressed.



8.2.2. Start the color map subscription node

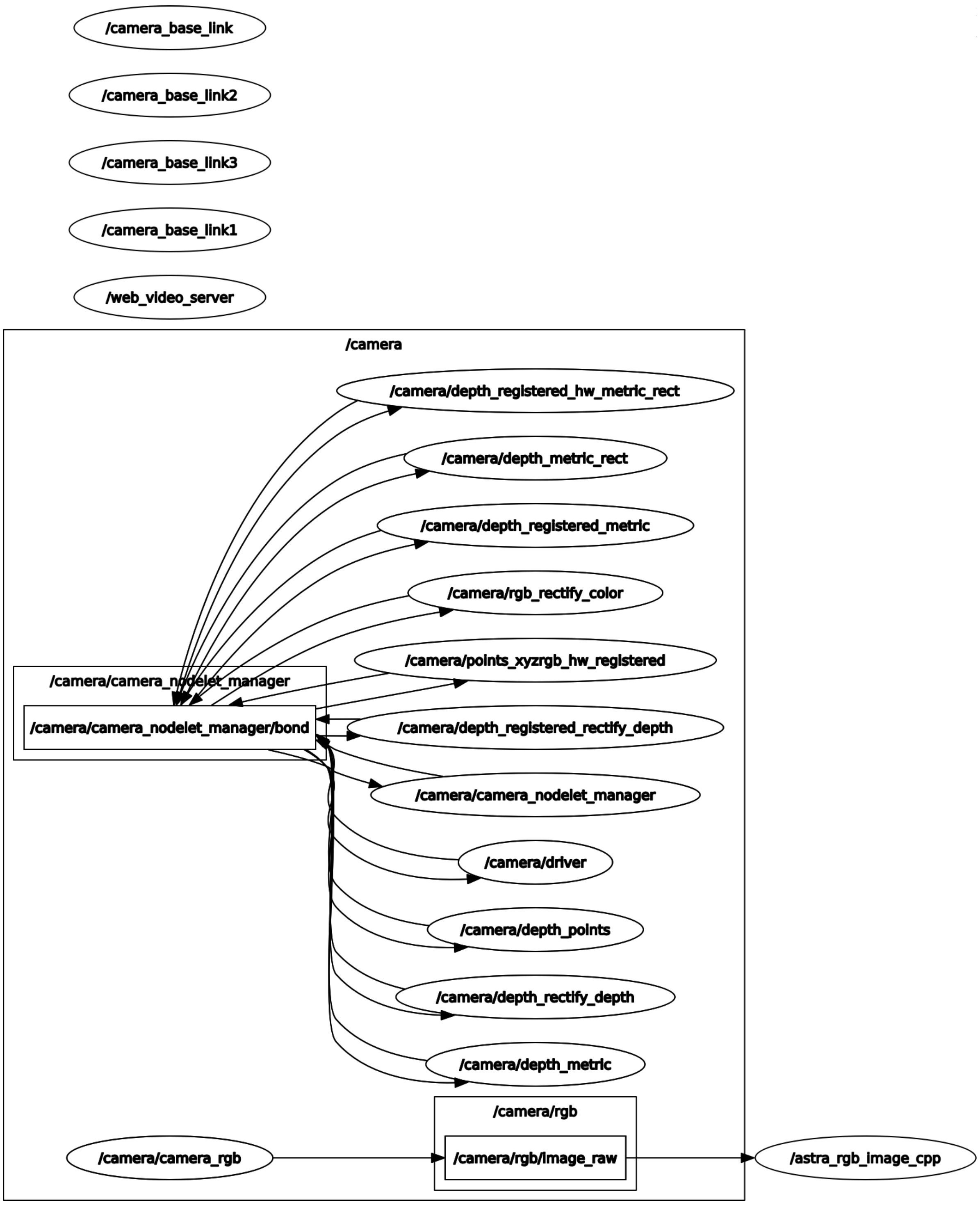
```
roslaunch yahboom_navrobo_visual astra_get_rgb.launch version:=cpp
```

- version parameter: optional [py, cpp] different codes, same effect.

View Node Graph

```
rqt_graph
```

When you open the node graph, you will see densely packed nodes and the relationships between them. At this point, we are using the part linked to the topic [/camera/color/image_raw], and [/astra_color_Image_cpp] is the node we wrote.



/camera_base_link
/camera_base_link2
/camera_base_link3
/camera_base_link1
/web_video_server
/camera
/camera/depth_registered_hw_metric_rect
/camera/depth_metric_rect
/camera/depth_registered_metric
/camera/rgb_rectify_color
/camera/points_xyzrgb_hw_registered
/camera/camera_nodelet_manager
/camera/camera_nodelet_manager/bond
/camera/depth_registered_rectify_depth
/camera/camera_nodelet_manager
/camera/driver
/camera/depth_points
/camera/depth_rectify_depth
/camera/depth_metric
/camera/rgb
/camera/camera_rgb
/camera/rgb/image_raw
/astra_rgb_image_cpp

- py code analysis

Create subscriber: subscribe to the topic ["/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 need to pay attention to the encoding format. If the encoding format is incorrect, the converted image will have problems.

```
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);
```

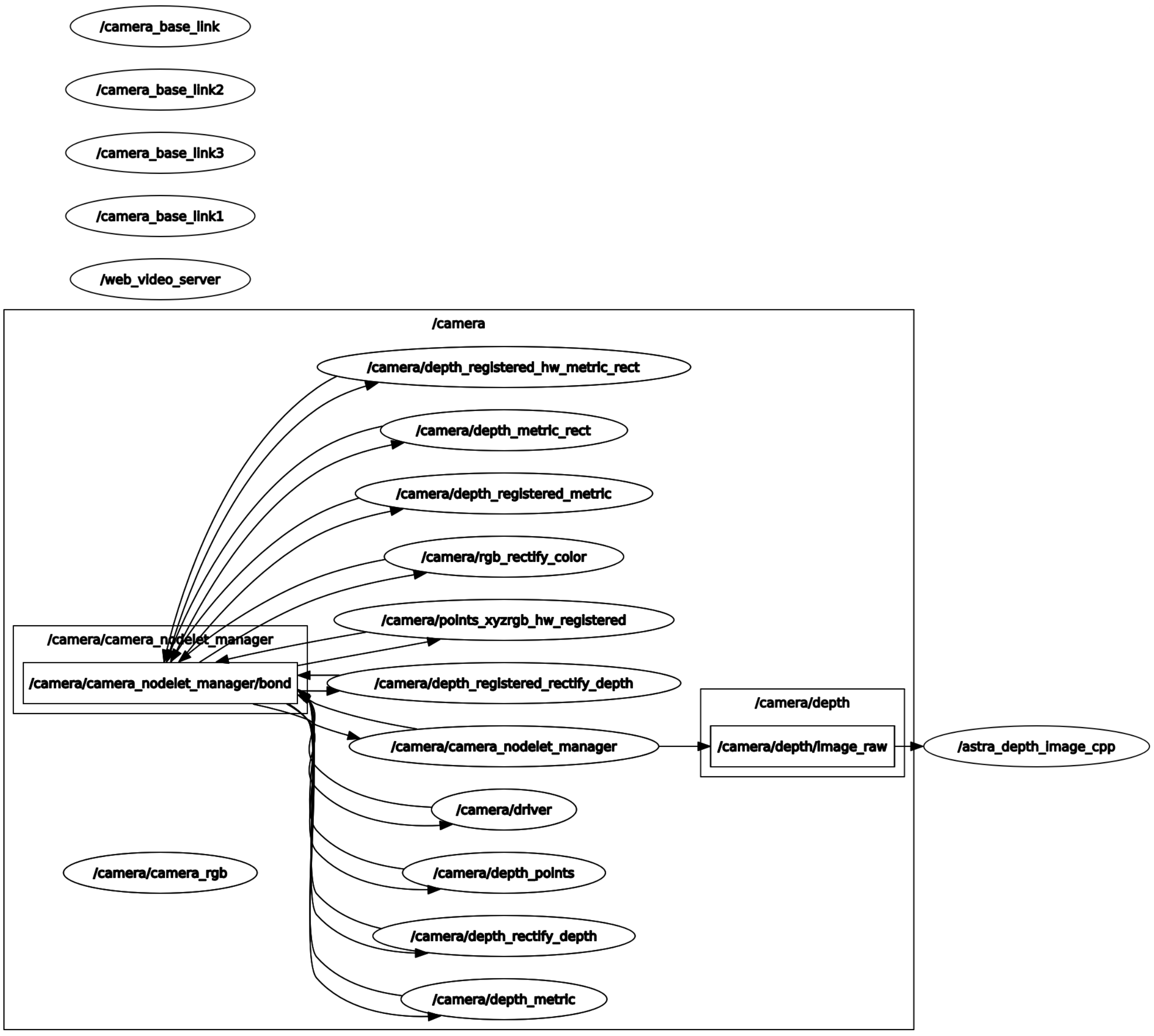
8.2.3. Start the depth map subscription node

```
roslaunch yahboom_navrobo_visual astra_get_depth.launch version:=cpp
```

View Node Graph

```
rqt_graph
```

When you open the node graph, you will see a lot of nodes and their relationships. At this point, we are using the part that is linked to the topic [/camera/depth/image_raw]. [/astra_depth_Image_cpp] is the node we wrote.


/camera_base_link
/camera_base_link2
/camera_base_link3
/camera_base_link1
/web_video_server
/camera
/camera/depth_registered_hw_metric_rect
/camera/depth_metric_rect
/camera/depth_registered_metric
/camera/rgb_rectify_color
/camera/points_xyzrgb_hw_registered
/camera/camera_nodelet_manager
/camera/camera_nodelet_manager/bond
/camera/depth_registered_rectify_depth
/camera/camera_nodelet_manager
/camera/depth
/camera/depth/image_raw
/astra_depth_Image_cpp
/camera/driver
/camera/camera_rgb
/camera/depth_points
/camera/depth_rectify_depth
/camera/depth_metric

- py code analysis

Create subscriber: subscribe to the topic ["/camera/depth/image_raw"], data type [Image], callback function [topic()]

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

Use [CvBridge] to convert data. Here you need to 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

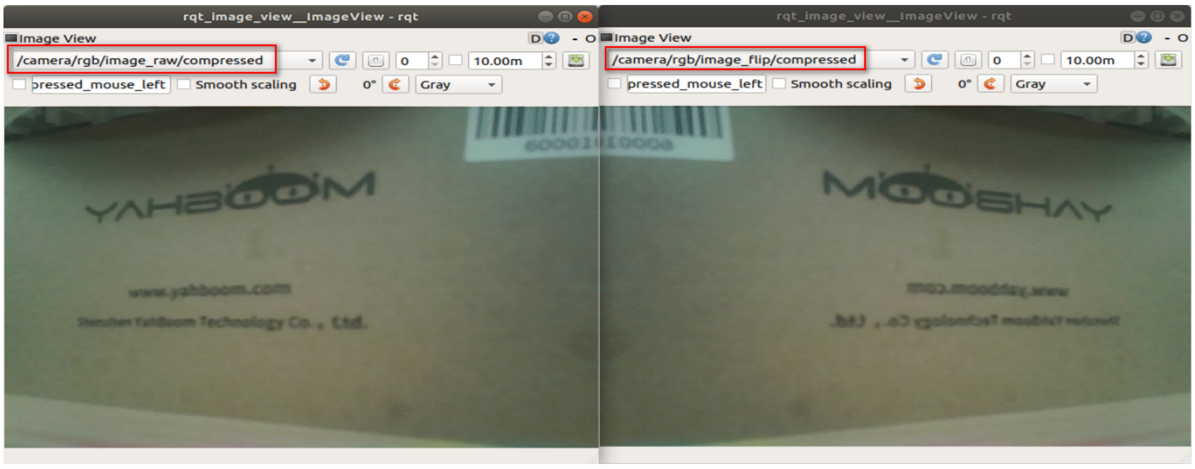
```
//Create a receiver.
ros::Subscriber subscriber = n.subscribe<sensor_msgs::Image>
("/camera/depth/image_raw", 10, depth_Callback);
//Create cv_bridge example
cv_bridge::CvImagePtr cv_ptr;
//Data conversion
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::TYPE_16UC1);
```

8.2.4. Start color image inversion

```
roslaunch yahboom_navrobo_visual astra_image_flip.launch
```

Image Viewer

```
rqt_image_view
```


rqt_image_view__ImageView - rqt

/camera/rgb/image_raw/compressed

/camera/rgb/image_flip/compressed

- py code analysis

Two subscribers and two publishers are created here, one for general image data and the other for compressed image data.

1) Create subscriber

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

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

2) Create publisher

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

The published topic is ["/camera/color/image_flip/compressed"], data type [CompressedImage], queue size [10].

```
sub_img = rospy.Subscriber("/camera/color/image_raw", Image, topic)
pub_img = rospy.Publisher("/camera/color/image_flip", Image, queue_size=10)
sub_comimg = rospy.Subscriber("/camera/color/image_raw/compressed",
CompressedImage, compressed_topic)
pub_comimg = rospy.Publisher("/camera/color/image_flip/compressed",
CompressedImage, 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 processes the image
    frame = cv.resize(frame, (640, 480))
    frame = cv.flip(frame, 1)
    # opencv mat ->  ros msg
    msg = bridge.cv2_to_imgmsg(frame, "bgr8")
    # After the image is processed, publish it directly
    pub_img.publish(msg)

# Compressed image transmission processing
def compressed_topic(msg):
    if not isinstance(msg, CompressedImage): return
    bridge = CvBridge()
    frame = bridge.compressed_imgmsg_to_cv2(msg, "bgr8")
    # Opencv processes the image
    frame = cv.resize(frame, (640, 480))
    frame = cv.flip(frame, 1)
    # Create CompressedIamge
    msg = CompressedImage()
    msg.header.stamp = rospy.Time.now()
    # Image data conversion
    msg.data = np.array(cv.imencode('.jpg', frame)[1]).tostring()
    # After the image is processed, publish it directly
    pub_comimg.publish(msg)
```