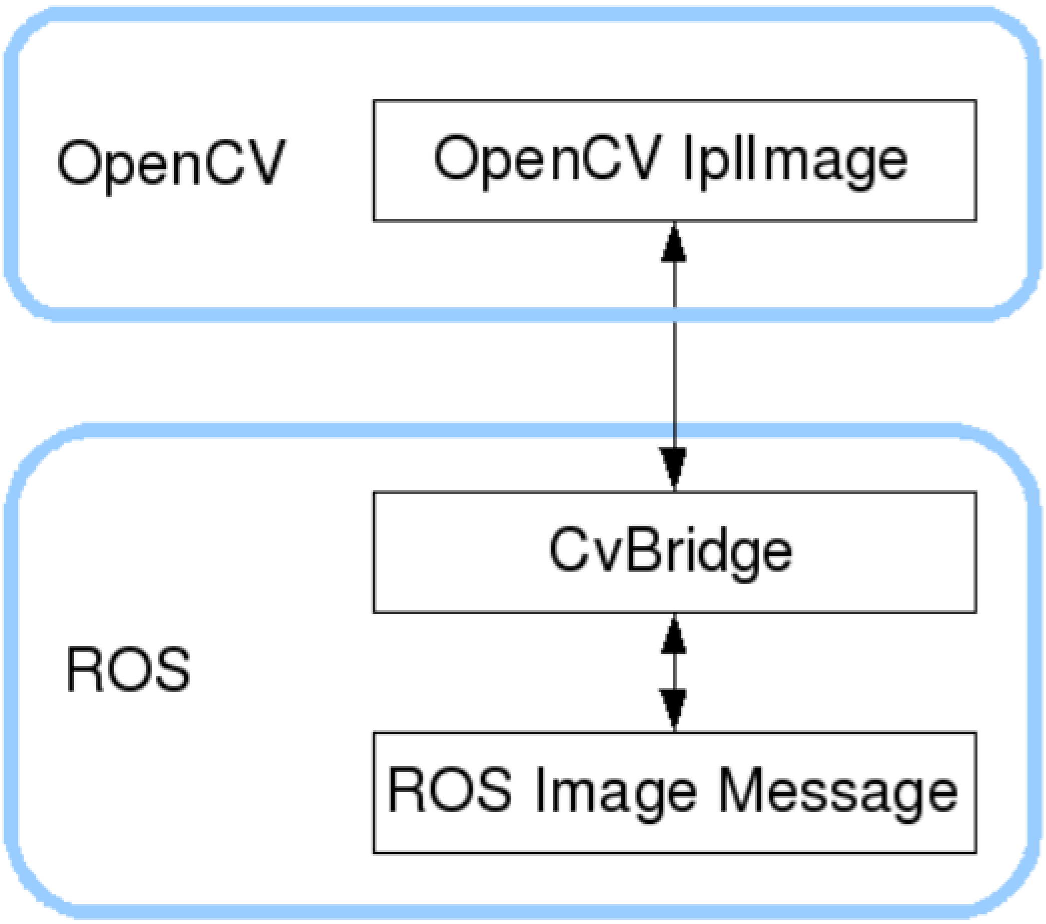
2. ROS+opencv basics

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:



This lesson uses three cases to show how to use CvBridge for data conversion.

1. Depth camera

Before driving the depth camera, the host needs to be able to identify the camera device;

1.1.1. Start the camera

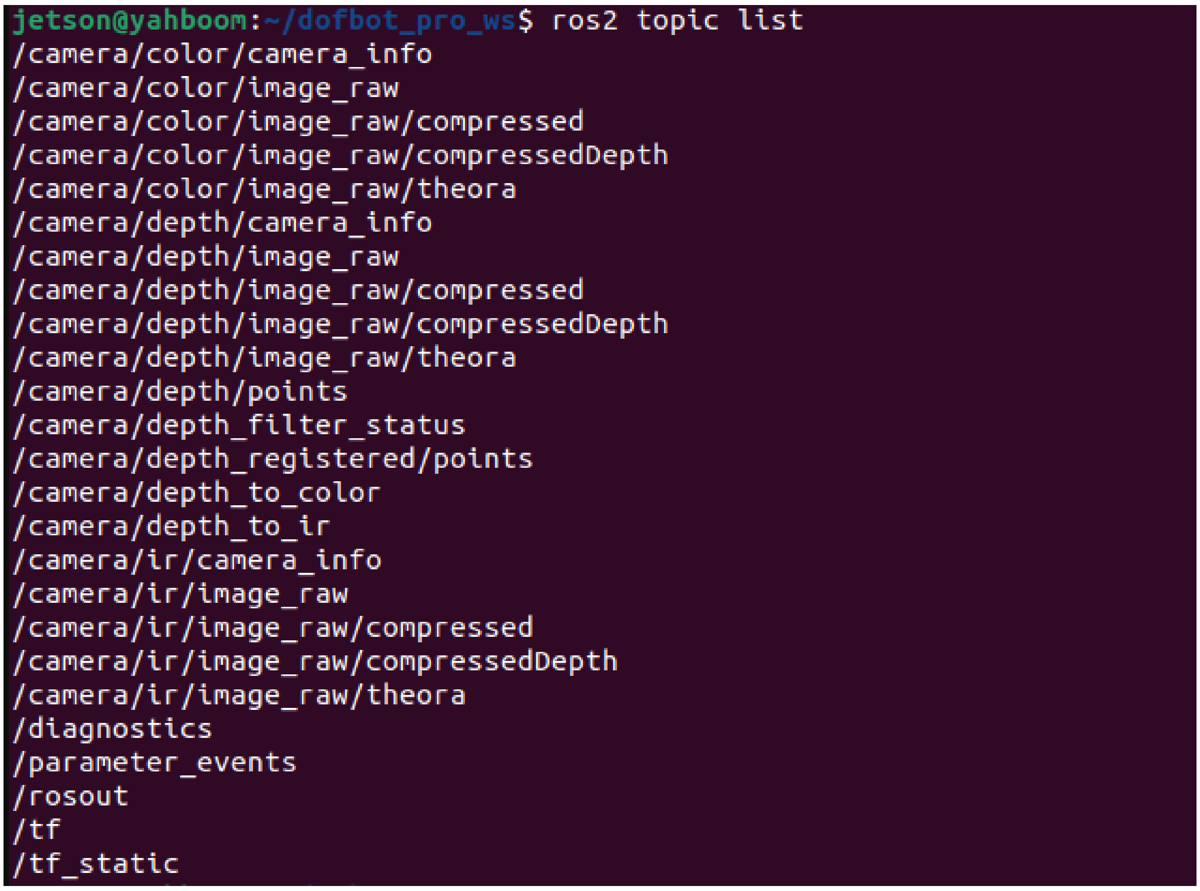
Terminal input,

```
ros2 launch orbbec_camera dabai_dcw2.launch.py
```

1.1.2. View the camera topic

Terminal input,

```
ros2 topic list
```

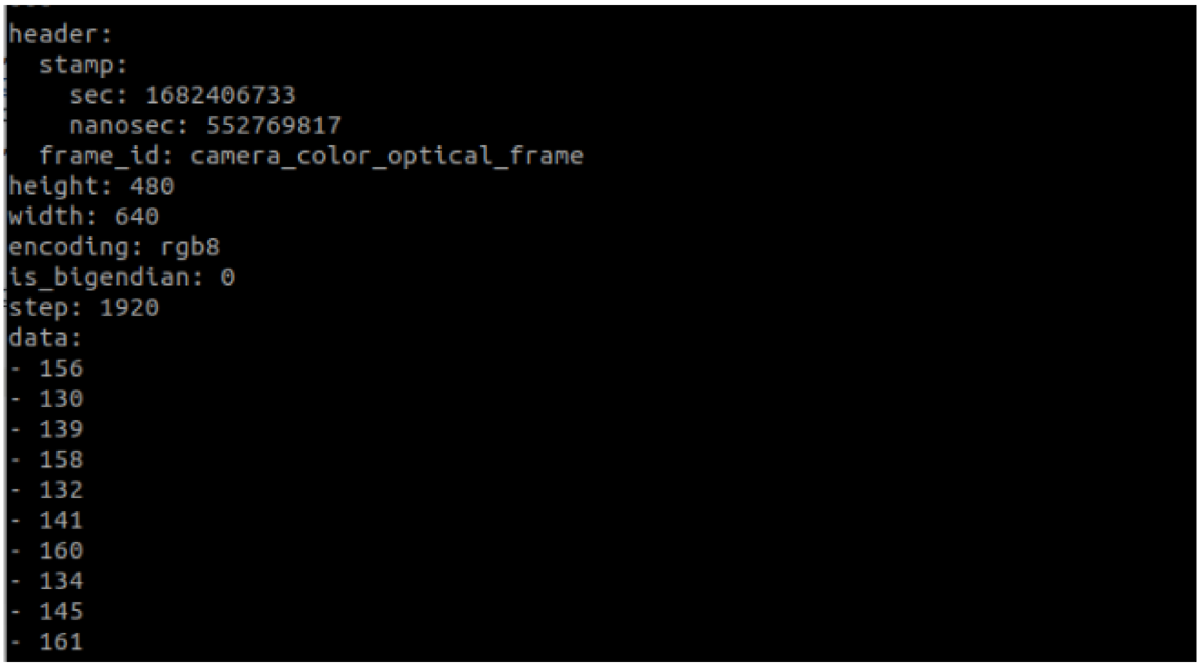


```
jetson@yahboom:~/dofbot_pro_ws$ ros2 topic list
/camera/color/camera_info
/camera/color/image_raw
/camera/color/image_raw/compressed
/camera/color/image_raw/compressedDepth
/camera/color/image_raw/theora
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/image_raw/compressed
/camera/depth/image_raw/compressedDepth
/camera/depth/image_raw/theora
/camera/depth/points
/camera/depth_filter_status
/camera/depth_registered/points
/camera/depth_to_color
/camera/depth_to_ir
/camera/ir/camera_info
/camera/ir/image_raw
/camera/ir/image_raw/compressed
/camera/ir/image_raw/compressedDepth
/camera/ir/image_raw/theora
/diagnostics
/parameter_events
/rosout
/tf
/tf_static
```

The main thing to look at is the image data topic. Here only RGB color image and Depth depth image topic information are parsed. Use the following commands to view their respective data information. Terminal input,

```
#View RGB image topic data content
ros2 topic echo /camera/color/image_raw
#View Depth image topic data content
ros2 topic echo /camera/depth/image_raw
```

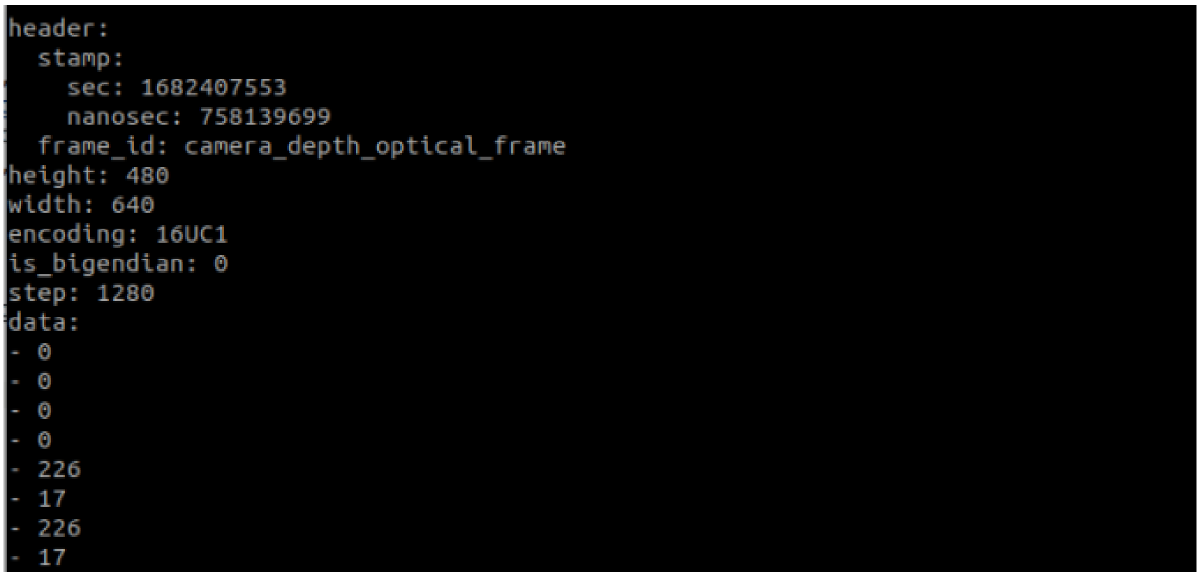
First, take a look at the RGB color image information of a frame.



```
header:
  stamp:
    sec: 1682406733
    nanosec: 552769817
  frame_id: camera_color_optical_frame
height: 480
width: 640
encoding: rgb8
is_bigendian: 0
step: 1920
data:
- 156
- 130
- 139
- 158
- 132
- 141
- 160
- 134
- 145
- 161
```

Here is the basic information of the image, an important value, **encoding**, the value here is **rgb8**, this value indicates that the encoding format of this frame of image is rgb8, which needs to be referred to when doing data conversion later.

Similarly, the following is the image data information of a certain frame of the depth image,



```
header:
  stamp:
    sec: 1682407553
    nanosec: 758139699
  frame_id: camera_depth_optical_frame
height: 480
width: 640
encoding: 16UC1
is_bigendian: 0
step: 1280
data:
- 0
- 0
- 0
- 0
- 226
- 17
- 226
- 17
```

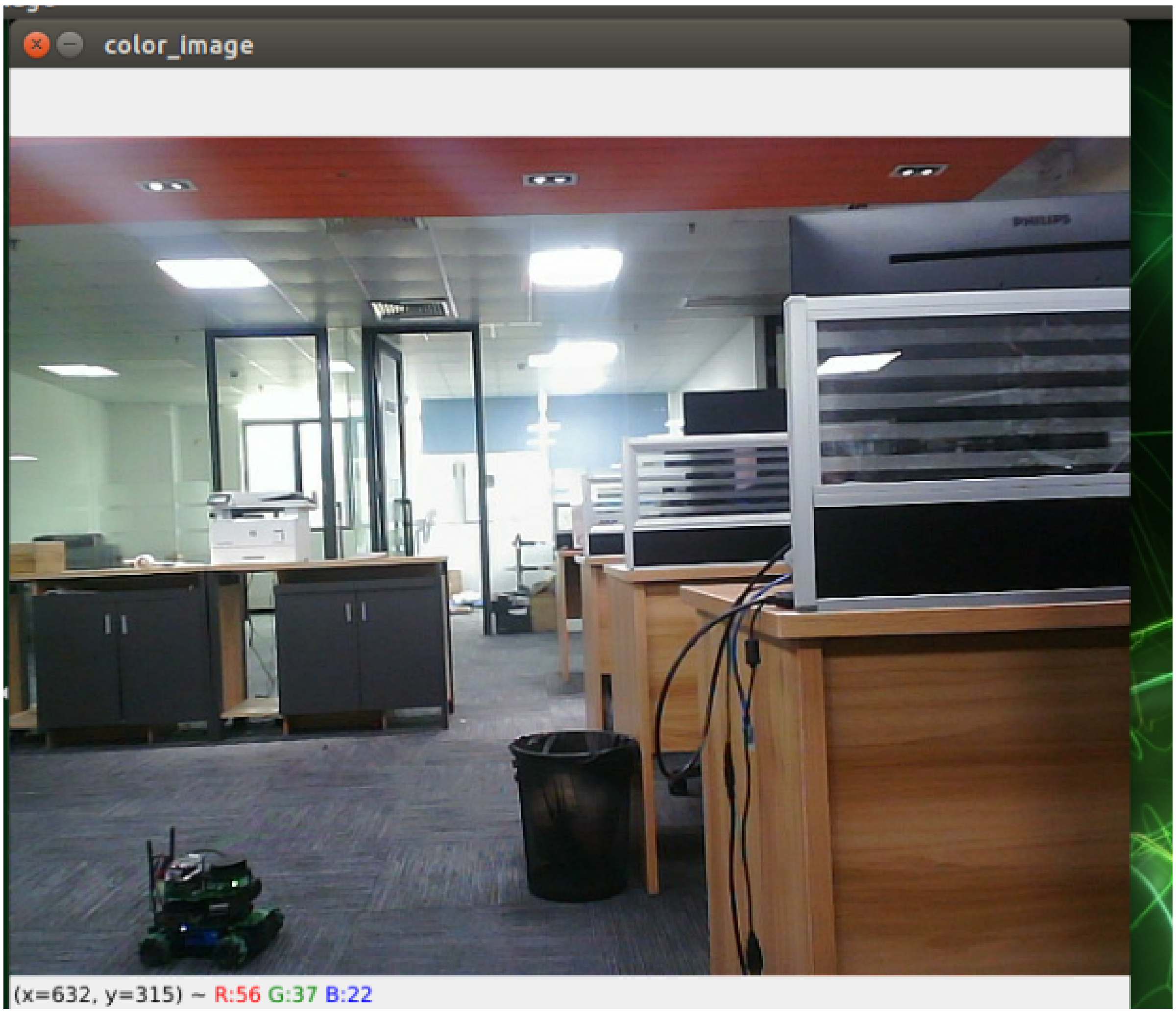
The encoding value here is **16UC1**.

2. Subscribe to RGB image topic information and display RGB image

2.1. Run command

Terminal input,

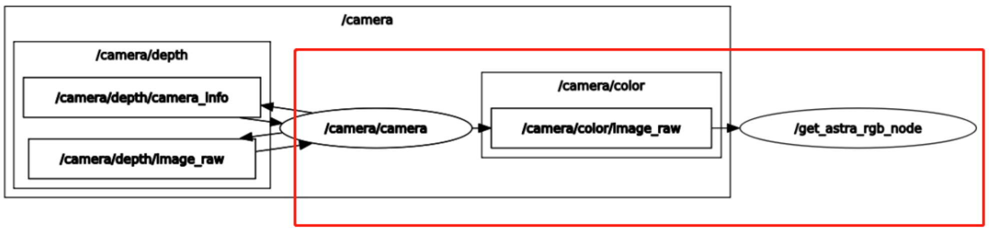
```
#RGB color image display node
ros2 run dofbot_pro_vision astra_rgb_image
```



2.2. View node communication graph

Terminal input,

```
ros2 run rqt_graph rqt_graph
```



2.3, core code analysis

Code reference path,

```
~/dofbot_pro_ws/src/dofbot_pro_vision/dofbot_pro_vision/astra_rgb_image.py
```

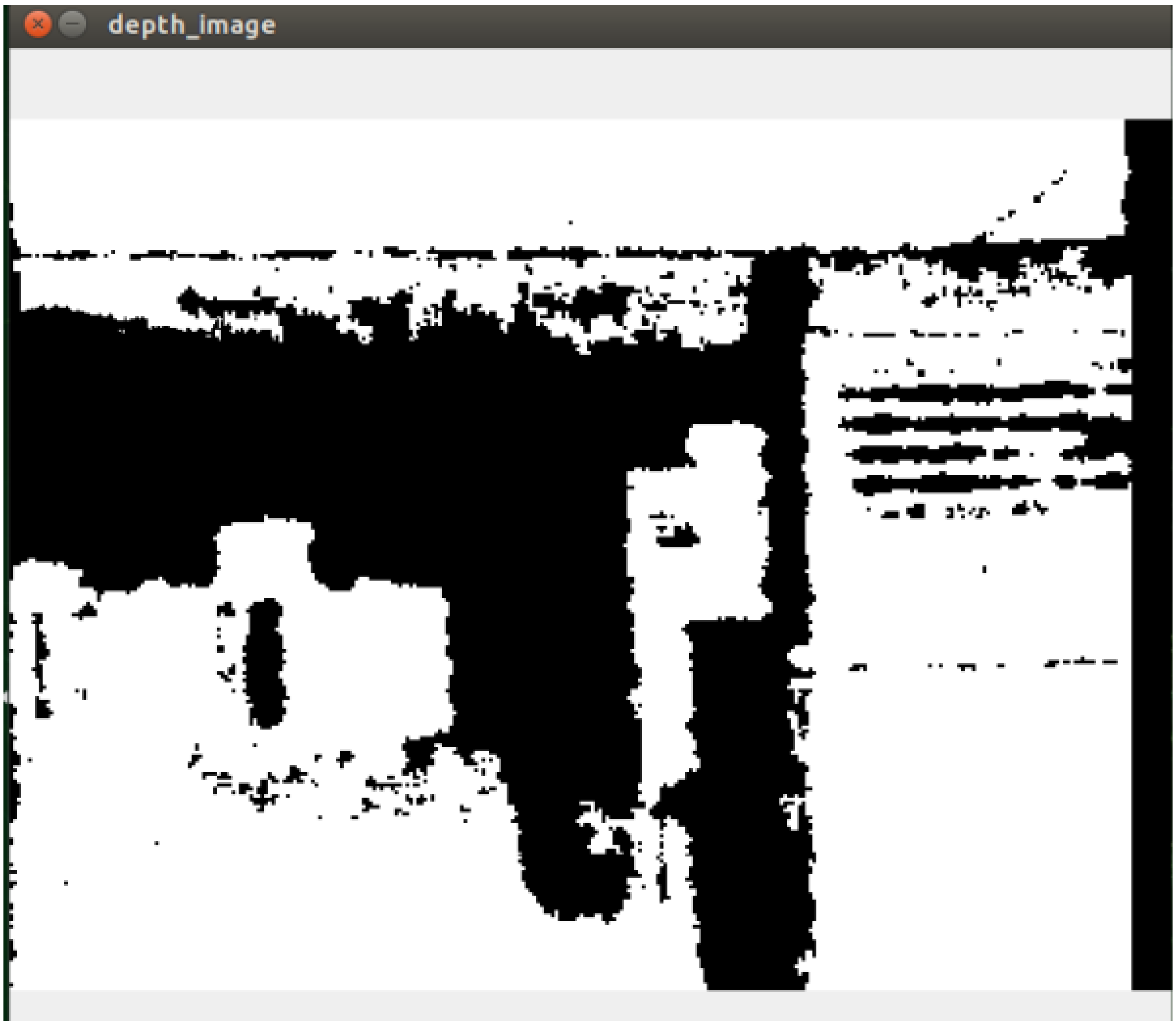
From 2.2, we can see that the /get_astra_rgb_node node subscribes to the topic of /camera/color/image_raw, and then converts the topic data into image data through data conversion. The code is as follows,

```
#Import opecv library and cv_bridge library
import cv2 as cv
from cv_bridge import CvBridge
#Create CvBridge object
self.bridge = CvBridge()
#Define a subscriber to subscribe to the RGB color image topic data released by
the depth camera node
self.sub_img
=self.create_subscription(Image,'/camera/color/image_raw',self.handleTopic,100)
#msg is converted into image data, where bgr8 is the image encoding format
frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
```

3. Subscribe to Depth image topic information and display Depth image

3.1. Run command

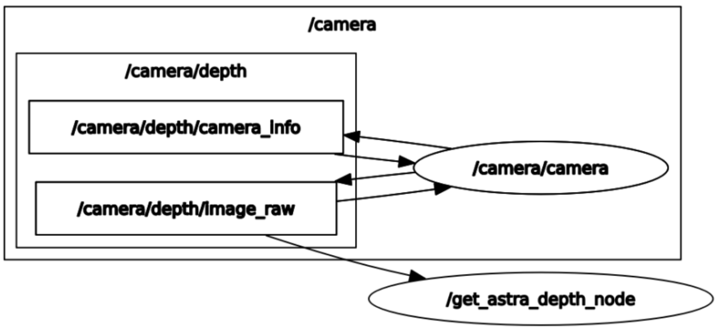
```
#RGB color image display node
ros2 run dofbot_pro_vision astra_depth_image
```



3.2. View node communication graph

Terminal input,

```
ros2 run rqt_graph rqt_graph
```



3.3, core code analysis

Code reference path,

```
~/dofbot_pro_ws/src/dofbot_pro_vision/dofbot_pro_vision/astra_depth_image.py
```

The basic implementation process is the same as the RGB color image display. It subscribes to the topic data of /camera/depth/image_raw published by the depth camera node, and then converts it into image data through data conversion. The code is as follows,

```
#Import opecv library and cv_bridge library
import cv2 as cv
from cv_bridge import CvBridge
#Create CvBridge object
self.bridge = CvBridge()
#Define a subscriber to subscribe to the Depth image topic data published by the
depth camera node
self.sub_img =
self.create_subscription(Image,'/camera/depth/image_raw',self.handleTopic,10)
#msg is converted into image data, where 32FC1 is the image encoding format
frame = self.bridge.imgmsg_to_cv2(msg, "32FC1")
```

4. Subscribe to image data and then publish the converted image data

4.1. Run command

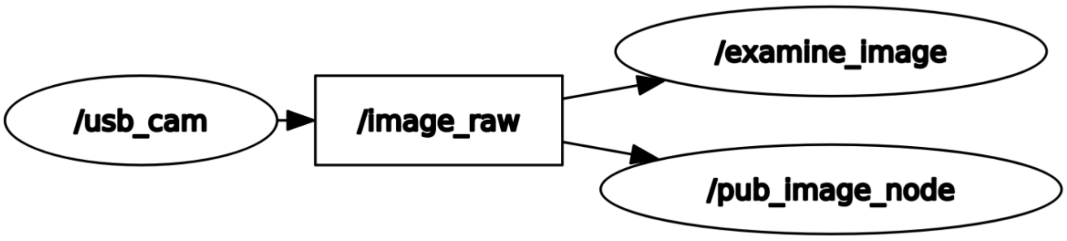
If there is an error, it is because the video device is occupied after running the previous roslaunch. At this time, you need to re-unplug the device to solve the problem!

```
#Run the node to publish image topic data
ros2 run dofbot_pro_vision pub_image
#Run the USB camera topic node
ros2 launch usb_cam camera.launch.py
```

4.2. View the node communication graph

Terminal input,

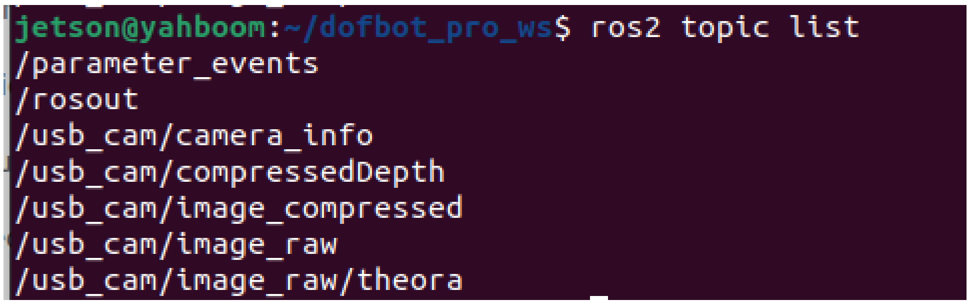
```
ros2 run rqt_graph rqt_graph
```


$$\begin{tikzcd}
 & & \text{/examine\_image} \\
\text{/usb\_cam} \arrow[r] & \boxed{\text{/image\_raw}} \arrow[ur] \arrow[dr] & \\
 & & \text{/pub\_image\_node}
\end{tikzcd}$$

4.3. View topic data

First check which image topics are published, enter the docker terminal,

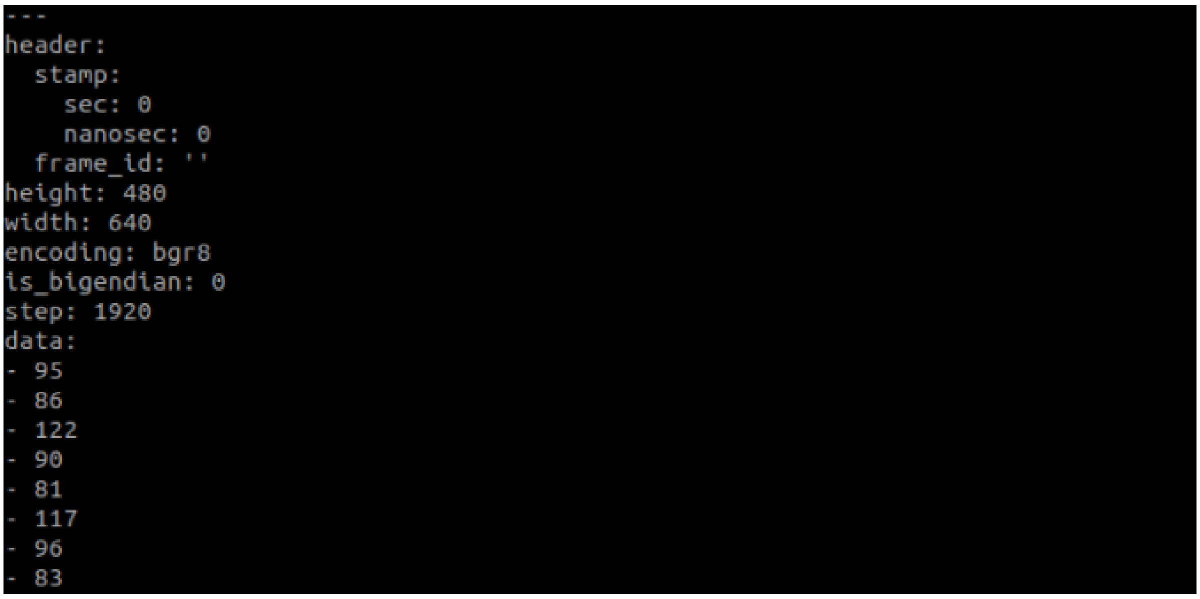
```
ros2 topoic list
```



```
jetson@yahboom:~/dofbot_pro_ws$ ros2 topic list
/parameter_events
/rosout
/usb_cam/camera_info
/usb_cam/compressedDepth
/usb_cam/image_compressed
/usb_cam/image_raw
/usb_cam/image_raw/theora
```

The /image is the topic data we published. Use the following command to print the data content of this topic,

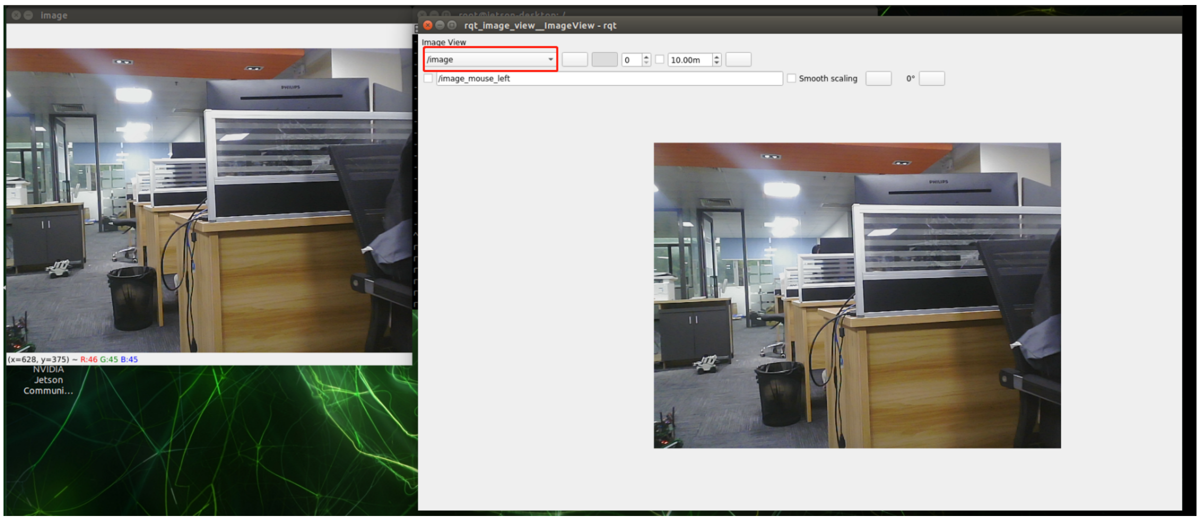
```
ros2 topic echo /image
```



```
---
header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
height: 480
width: 640
encoding: bgr8
is_bigendian: 0
step: 1920
data:
- 95
- 86
- 122
- 90
- 81
- 117
- 96
- 83
```

You can use the rqt_image_view tool to view the image,

```
ros2 run rqt_image_view rqt_image_view
```



After opening, select the topic name/image in the upper left corner to view the image.

4.4, core code analysis

Code path,

```
~/dofbot_pro_ws/src/dofbot_pro_vision/dofbot_pro_vision
```

The implementation steps are roughly the same as the previous two. The program first subscribes to the topic data of /usb_cam/image_raw and then converts it into image data. However, format conversion is also performed here to convert the image data into topic data and then publish it, that is, image topic data->image data->image topic data.

```
#Import opecv library and cv_bridge library
import cv2 as cv
from cv_bridge import CvBridge
#Create CvBridge object
self.bridge = CvBridge()
#Define a subscriber to subscribe to USB image topic data
```

```
self.sub_img =
self.create_subscription(Image,'/usb_cam/image_raw',self.handleTopic,500)
#Define the publisher of image topic data
self.pub_img = self.create_publisher(Image,'/image',500)
#Convert msg to image data imgmsg_to_cv2, where bgr8 is the image encoding format
frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
#Image data converted to image topic data (cv2_to_imgmsg) and then published
msg = self.bridge.cv2_to_imgmsg(frame, "bgr8")
self.pub_img.publish(msg)
```