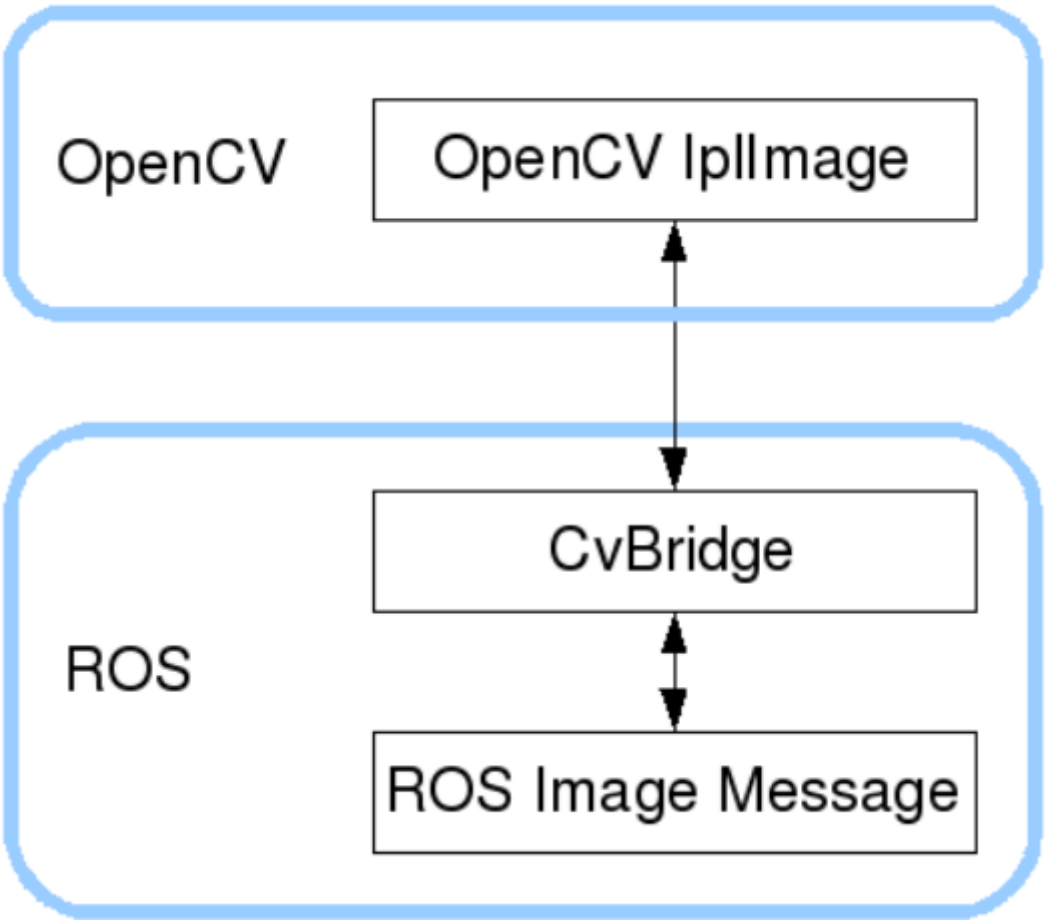
6.ROS+Opencv basics

ROS transmits images using its own sensor_msgs/Image message format, which does not allow direct image processing. However, the provided CvBridge library can perfectly convert and convert image data formats. CvBridge is a ROS library that acts as a bridge between ROS and OpenCV.

The image data conversion between OpenCV and ROS is shown in the following figure:



This lesson uses three case studies to demonstrate how to use CvBridge for data transformation.

1. Depth camera

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

1.1.1 Start the camera

Terminal input,

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

1.1.2 View camera topics

Terminal input,

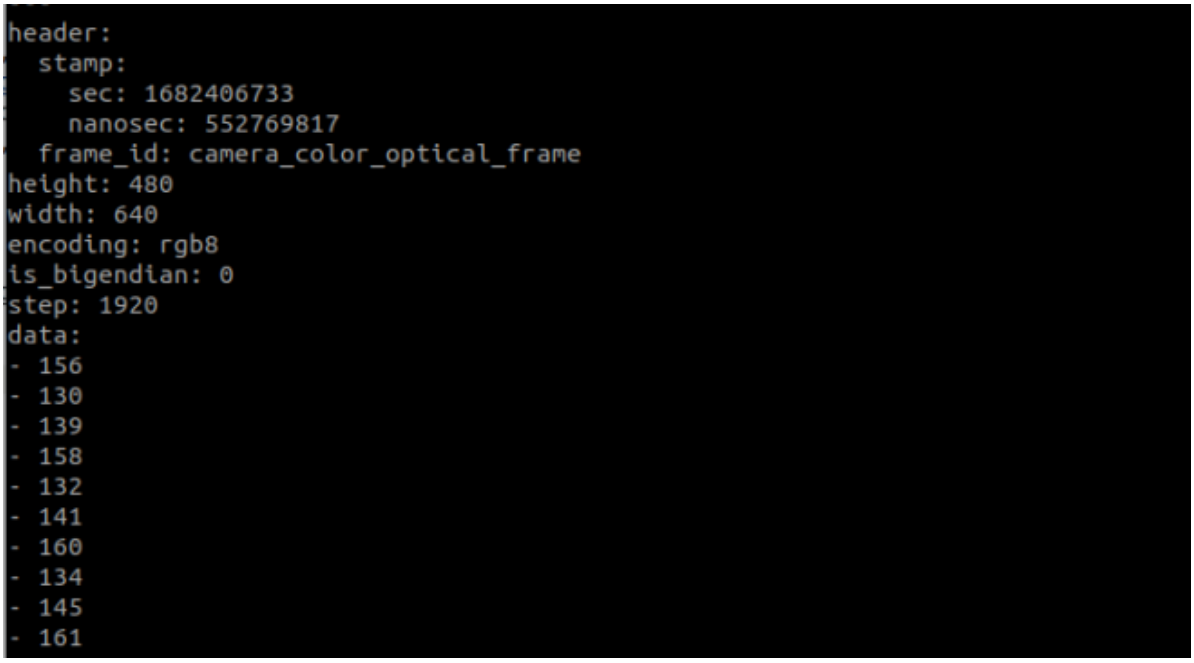
```
ros2 topic list
```

```
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 focus is on image data topics. Here, we only parse the topic information for RGB color images and depth images. Use the following commands to view their respective data information: Enter the command in the terminal.

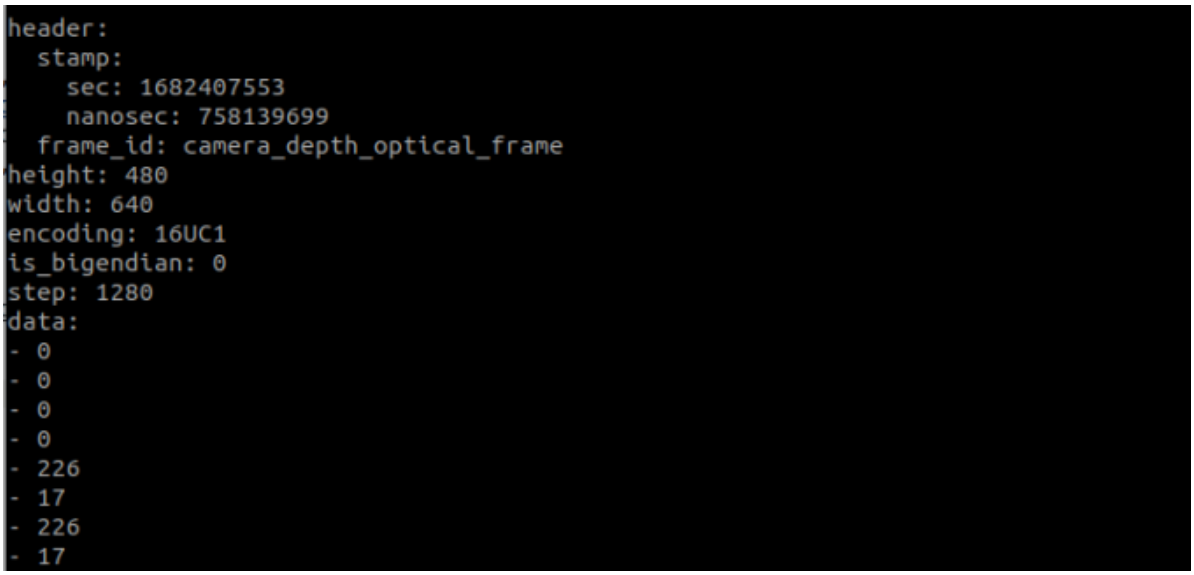
```
#View RGB image topic data content
ros2 topic echo /camera/color/image_raw
#View Depth image topic data content
ros2 topic echo /camera/depth/image_raw
```

Let's first take a screenshot of a frame of RGB color image information and see.



This section explains the basic information of the image, including an important value, **encoding**, which is **rgb8**. This value indicates that the encoding format of this image frame is rgb8, and this will be needed when performing data conversion later.

Similarly, below is image data information for a single frame of the depth image.



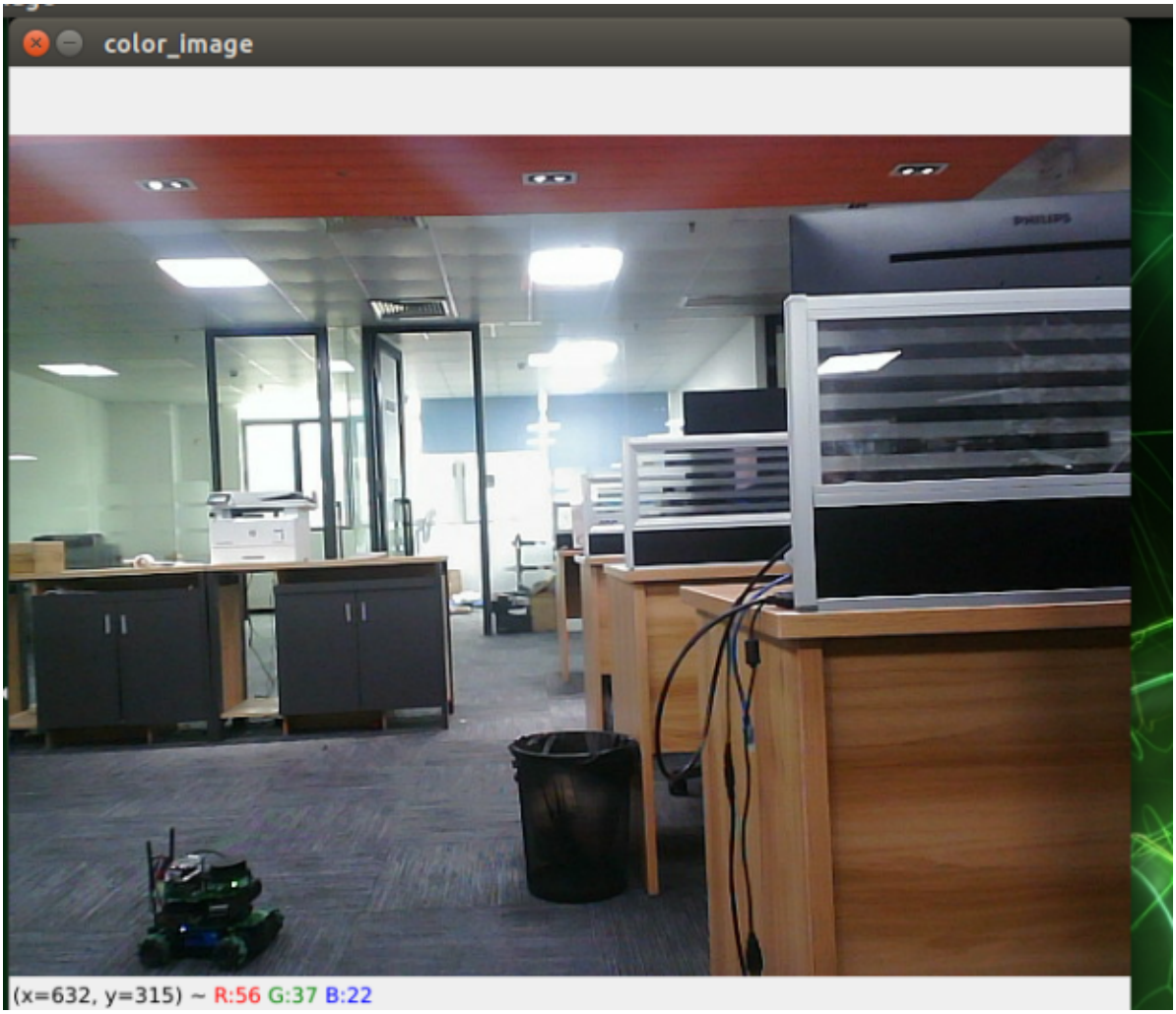
The encoding value here is **16UC1**.

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

2.1. Run the command

Terminal input,

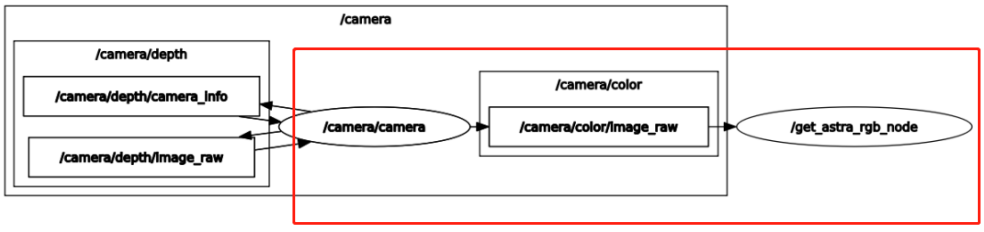
```
#RGB color image display node
ros2 run dofbot_pro_vision astra_rgb_image
```



2.2 Viewing the Node Communication Diagram

Enter the following in the terminal:

```
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
```

As shown in section 2.2, the `/get_astra_rgb_node` subscribes to the `/camera/color/image_raw` topic, and then converts the topic data into image data for publication. The code is as follows:

```
#Import the opecv library and the cv_bridge library
import cv2 as cv
from cv_bridge import CvBridge
# Create a CvBridge object

self.bridge = CvBridge()

# Define a subscriber to subscribe to RGB color image topic data published by the
depth camera node

self.sub_img = self.create_subscription(Image, '/camera/color/image_raw',
self.handleTopic, 100)

# Convert msg to image data, where bgr8 is the image encoding format

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

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

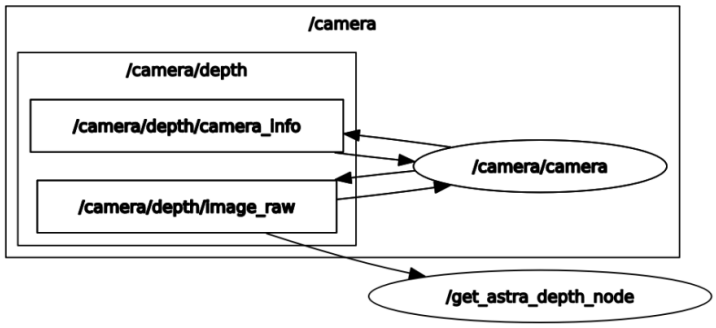
3.1. Run the command

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

3.2 Viewing the Node Communication Diagram

Enter the following in the terminal:

```
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 for RGB color image display. It subscribes to the topic data `/camera/depth/image_raw` published by the depth camera node, and then converts the data into image data through data transformation. The code is as follows:

```
# Import the opecv and cv_bridge libraries
import cv2 as cv
from cv_bridge import CvBridge
# Create a CvBridge object
self.bridge = CvBridge()
# Define a subscriber to subscribe to depth image topic data published by the
depth camera node
self.sub_img
=self.create_subscription(Image,'/camera/depth/image_raw',self.handleTopic,10)
# Convert msg to 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 the command

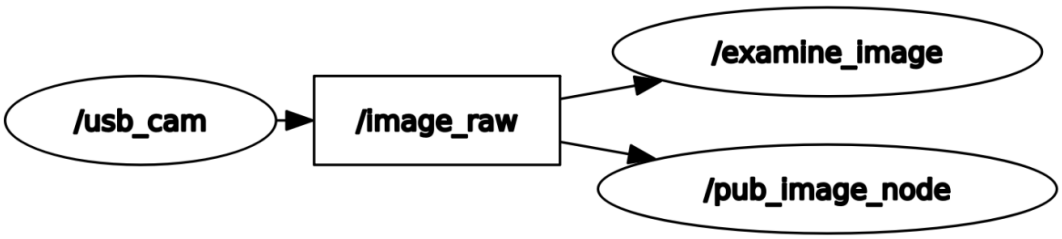
If an error occurs, it means that the roslaunch command has occupied the video device. In this case, you need to unplug the device to resolve the issue!

```
# Run the dofbot_pro_vision pub_image topic data node

ros2 run dofbot_pro_vision pub_image

# Run the usb camera topic node

ros2 launch usb_cam camera.launch.py
```

4.2 Viewing the Node Communication Diagram

Enter the following in the terminal:

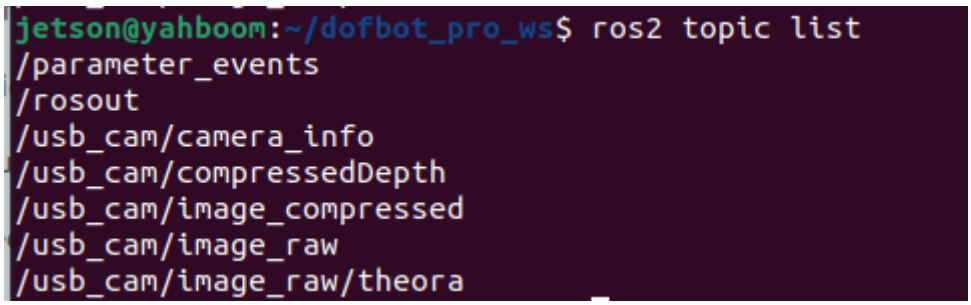
```
ros2 run rqt_graph rqt_graph
```



4.3 Viewing Topic Data

First, query which image topics have been published. Enter the following in the Docker terminal:

```
ros2 topoic list
```

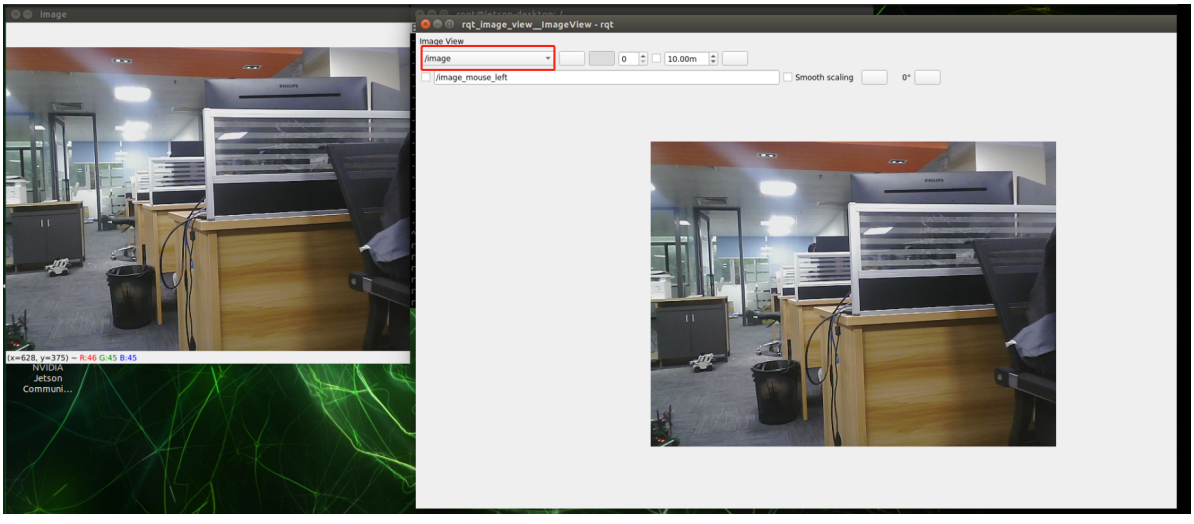


The `/image` part contains the topic data we published. Use the following command to print out the data content of this topic.

```
ros2 topic echo /image
```

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, a format conversion is also performed here to convert the image data into topic data before publishing it. That is, image topic data -> image data -> image topic data.

```
# Import the opecv and cv_bridge libraries
import cv2 as cv
from cv_bridge import CvBridge
# Create a 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 an image topic data publisher
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")
# Convert image data to image topic data (cv2_to_imgmsg) and then publish it
msg = self.bridge.cv2_to_imgmsg(frame, "bgr8")
self.pub_img.publish(msg)
```