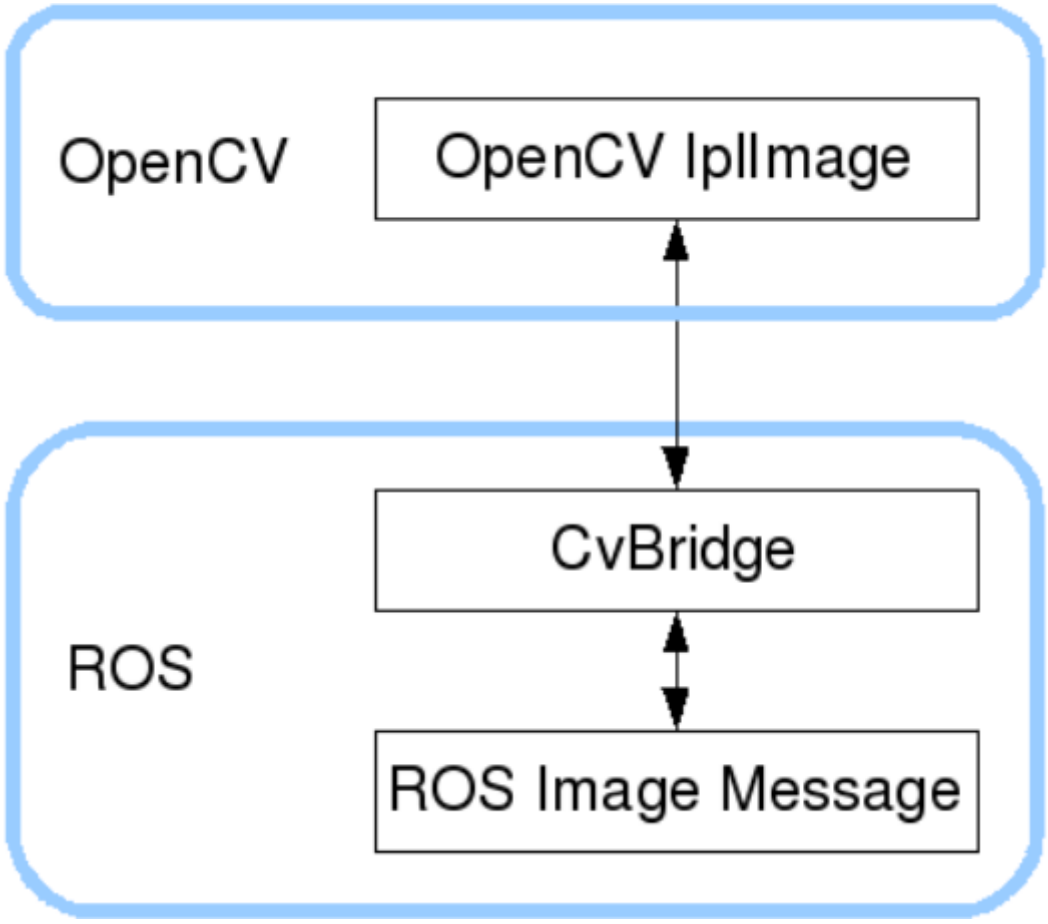
4.ROS+opencv application

This lesson takes the Astra camera as an example. Ordinary cameras are similar.

ROS transmits images in its own sensor_msgs/Image message format and cannot directly perform image processing, but the provided [CvBridge] can perfectly convert and be converted image data formats. [CvBridge] is a ROS library, equivalent to the 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. Astra camera

Before driving the depth camera, the astra camera device needs to be recognized on the host machine; when entering the docker container, the astra device needs to be mounted to recognize the camera in the docker container. The supporting host has already set up an environment and does not require additional configuration. If it is on a new host, you need to add a rule file. The adding method is very simple. Copy the /etc/udev.rules.d/56-orbbecusb.rules file under the host to the /etc/udev.rules.d directory in the new environment, and then restart it.

1.1. Start the camera

Take starting the Astrapro plus camera as an example. After entering the docker container, enter in the terminal,

```
ros2 launch astra_camera astro_pro_plus.launch.xml
```

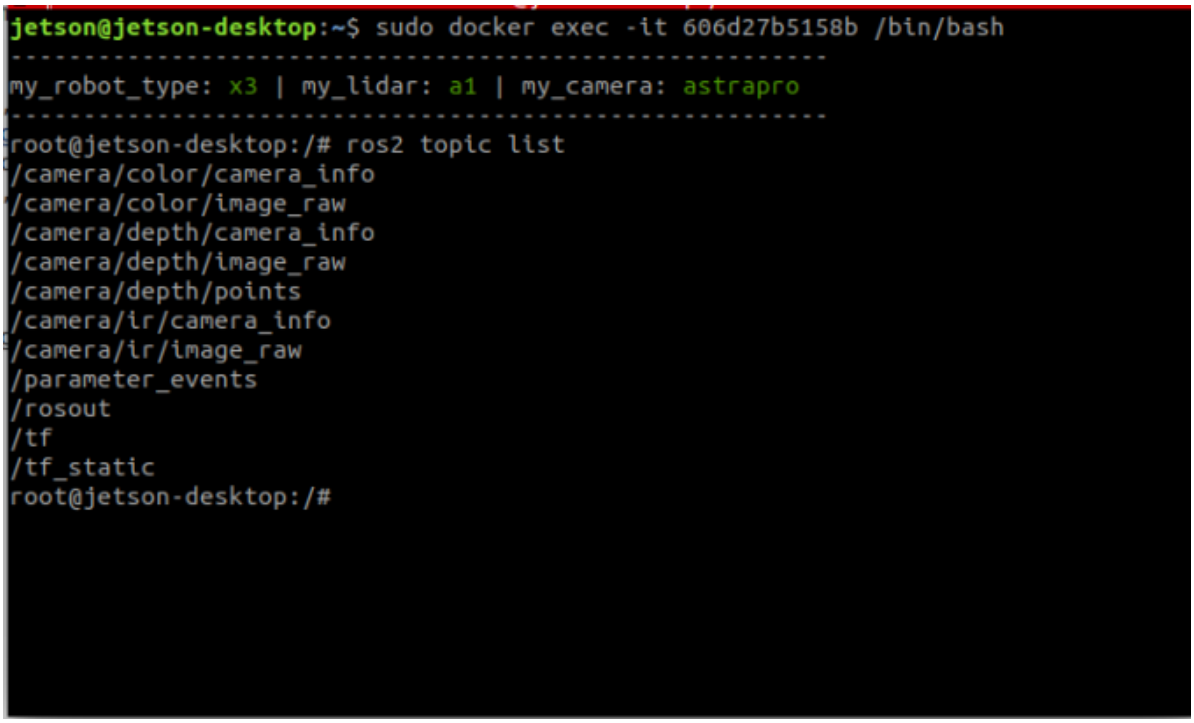
The corresponding camera model startup is shown in the table below.

launch文件	相机型号
ros2 launch astra_camera astra_pro.launch.xml	Astrapro
ros2 launch astra_camera astro_pro_plus.launch.xml	Astraproplus
ros2 launch astra_camera astra.launch.xml	Astramini

1.2. View camera topics

doke terminal input,

```
ros2 topic list
```



The main topic is the image data topic. Here we only parse the RGB color image and Depth depth image topic information. Use the following command to view the respective data information and enter it in the docke 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
```

First intercept a frame of RGB color image information and take a look.

```
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,This will need to be referenced when doing data conversion later.

Similarly, below 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
```

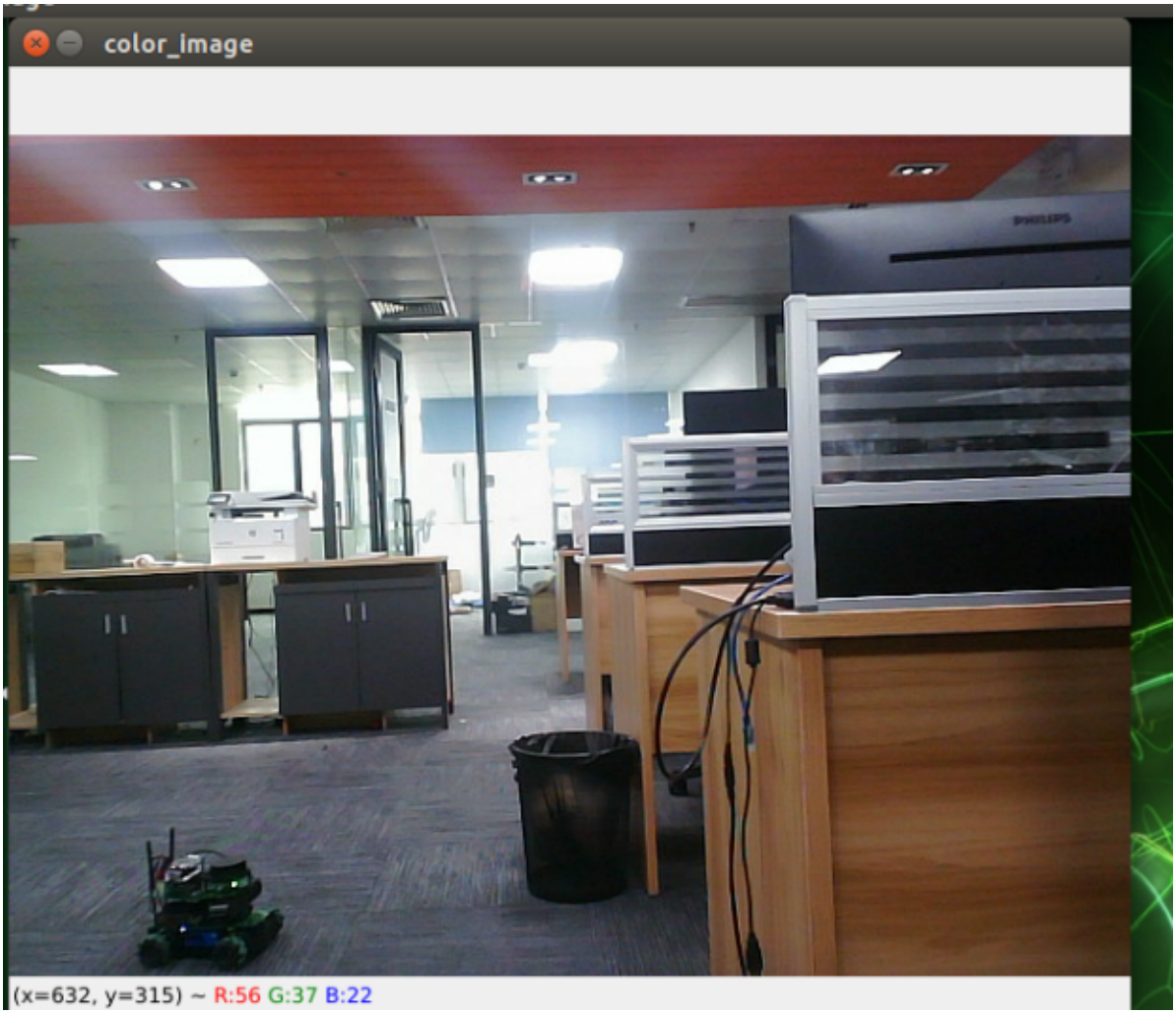
The encoding value here is **16UC1**.

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

2.1. Run command

docker terminal input,

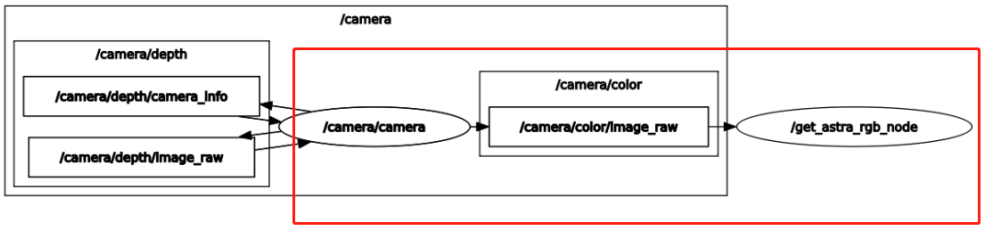
```
#RGB color image display node
ros2 run yahboomcar_visual astra_rgb_image
#Run Astrapro plus camera
ros2 launch astra_camera astro_pro_plus.launch.xml
```



2.2. View node communication diagram

docker terminal input,

```
ros2 run rqt_graph rqt_graph
```



2.3. Core code analysis

Code reference path,

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_visual/yahboomcar_visual/astra_rgb_image.py
```

It can be seen from 2.2 that the /get_astra_rgb_node node subscribes to the topic of /camera/color/image_raw, and then through data conversion, the topic data is converted into image data and published. 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 published 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

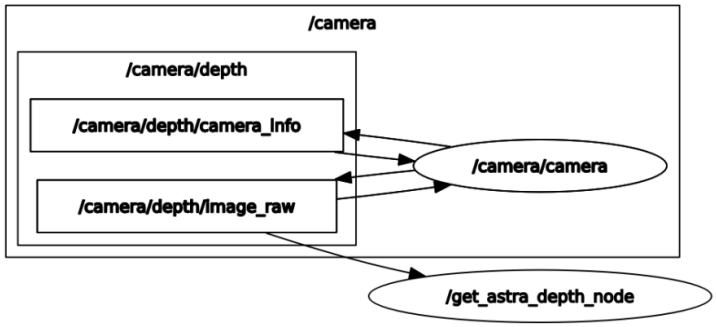
3.1. Run command

```
#RGB color image display node
ros2 run yahboomcar_visual astra_depth_image
#Run Astrapro plus camera
ros2 launch astra_camera astro_pro_plus.launch.xml
```

3.2. View node communication diagram

docker terminal input,

```
ros2 run rqt_graph rqt_graph
```



3.3. Core code analysis

Code reference path,

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_visual/yahboomcar_visual/a
stra_depth_image.py
```

The basic implementation process is the same as 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 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

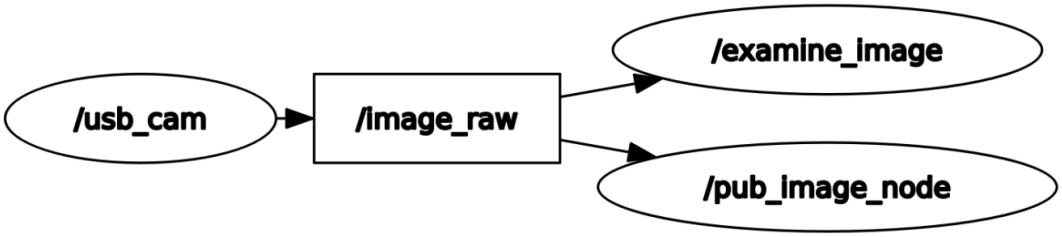
4.1. Run command

```
#Run the publish image topic data node
ros2 run yahboomcar_visual pub_image
#Run the usb camera topic node
ros2 launch usb_cam demo_launch.py
```

4.2. View node communication diagram

docker terminal input,

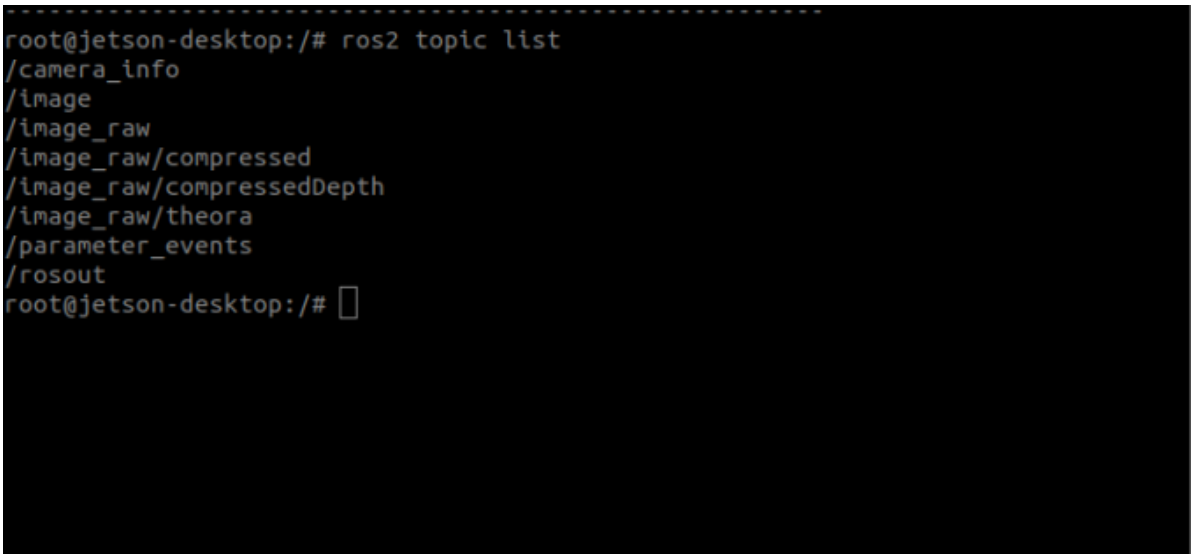
```
ros2 run rqt_graph rqt_graph
```



4.3. View topic data

First check which image topics are published, and enter it in the docker terminal.

```
ros2 topoic list
```



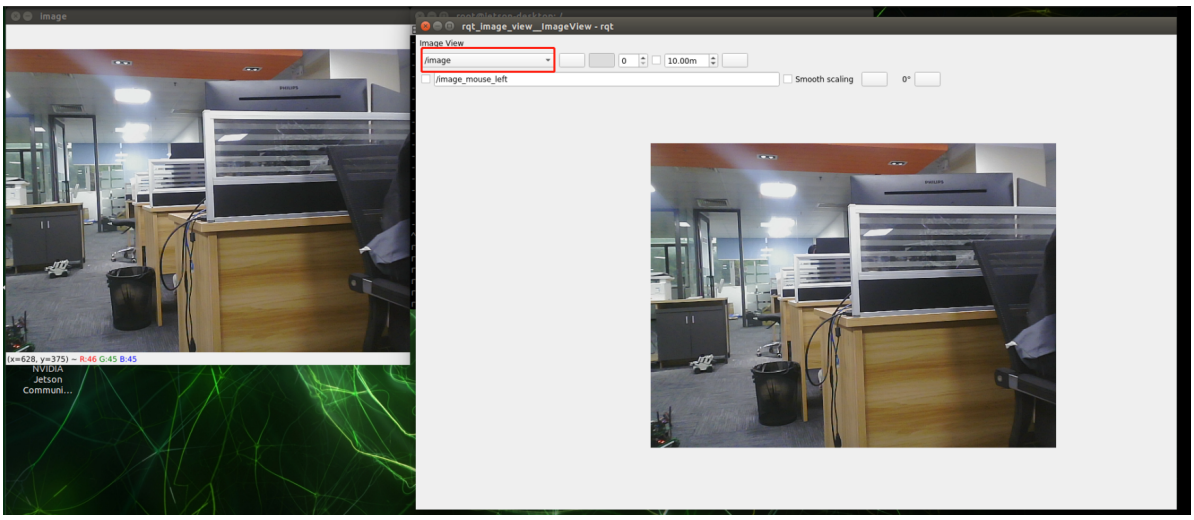
The /image is the topic data we published. Use the following command to print and see the data content of this topic.

```
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 images,

```
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,

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_visual/yahboomcar_visual/pub_image.py
```

The implementation steps are roughly the same as the previous two. The program first subscribes to the topic data of /image_raw, and then converts it into image data. However, it also performs lattice conversion here to convert the image data into topic data, and then publishes it, which is the image topic data. ->Image data->Image topic data.

```python
#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,'/image_raw',self.handleTopic,500)
#Image topic data publisher defined
self.pub_img = self.create_publisher(Image,'/image',500)
#msg is converted into image data imgmsg_to_cv2, where bgr8 is the image encoding
format
frame = self.bridge.imgmsg_to_cv2(msg, "bgr8")
#The image topic data (cv2_to_imgmsg) converted from the image data is then
published.
msg = self.bridge.cv2_to_imgmsg(frame, "bgr8")
self.pub_img.publish(msg)
```