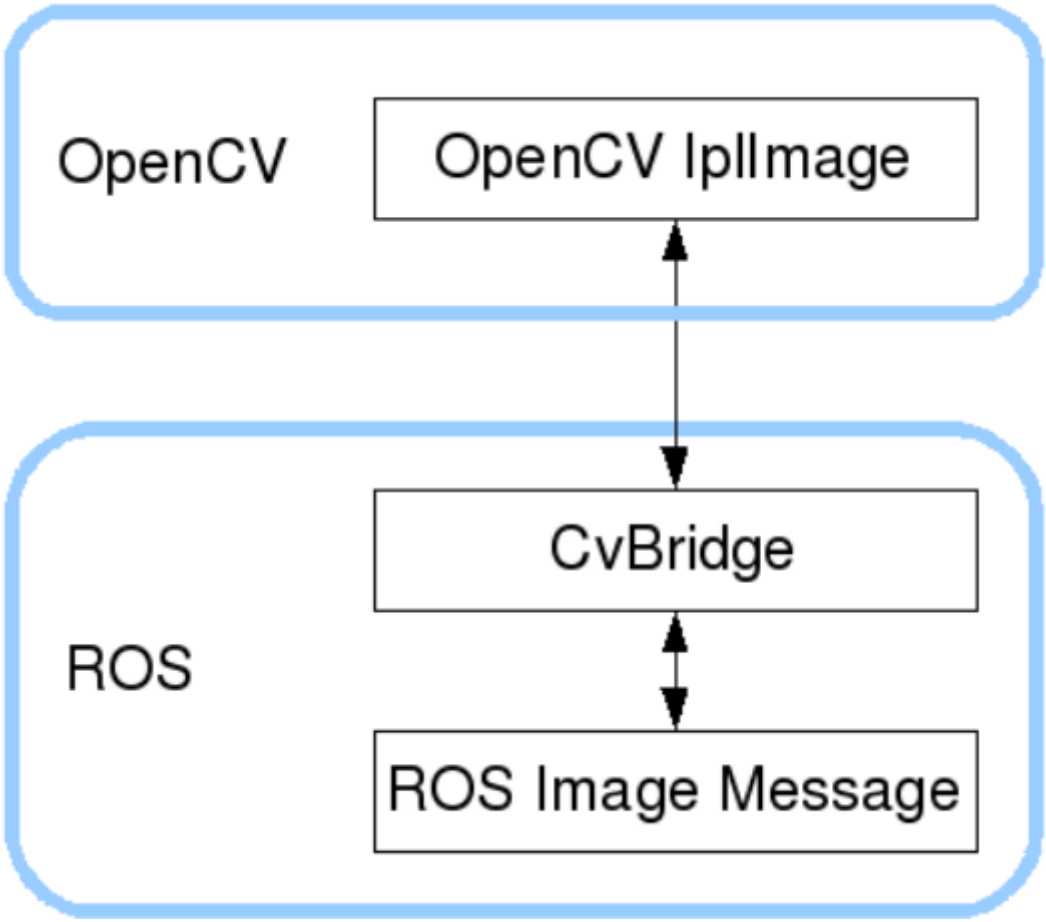
2、ROS+opencv application

This lesson takes Astra cameras as an example, similar to regular cameras.

ROS with its own “sensor_ The msgs/Image” message format transmits images, which cannot be directly processed, but the provided 【CvBridge】can perfectly convert and convert image data formats. 【CvBridge】 is an ROS library that serves as a bridge between ROS and Opencv.

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



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

1、Astra camera

Before driving a depth camera, it is necessary for the host to be able to recognize the Astra camera device; When entering the Docker container, you need to mount this Astra device to recognize the camera in the Docker container. The supporting host has already been built in an environment and does not require additional configuration. If it is on a new host, a rule file needs to be added. The addition method is very simple. Copy the "/etc/udev. rules. d/56 orbbecusb. rules" file from the host computer to the "/etc/udev. rules. d" directory in the new environment, and then restart it once.

1.1.1、Start camera

Taking starting the Astrapro camera as an example, after entering the Docker container, the terminal inputs,

```
ros2 launch astra_camera astra_pro.launch.xml
```

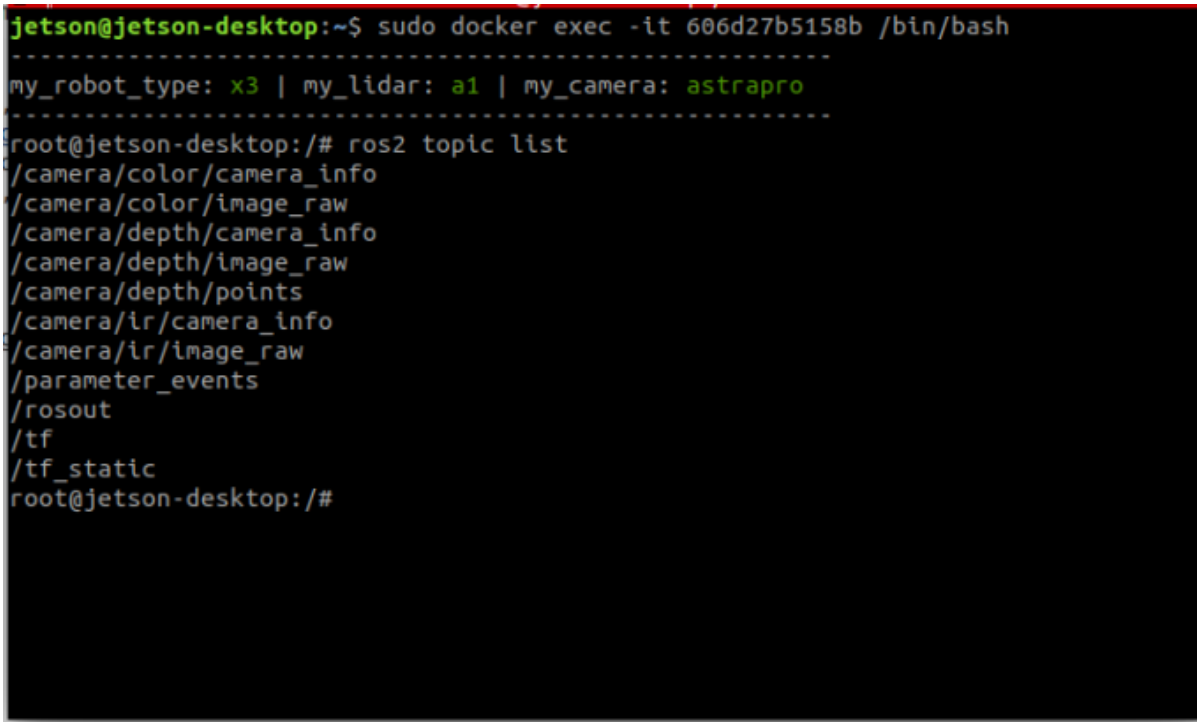
The corresponding camera model startup is shown in the table below,

Lanunch file	Camera model
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

1.1.2、View camera topics

Dock terminal input,

```
ros2 topic list
```



The main focus is on the topic of image data. Here, we only analyze the topic information of RGB color images and depth images. Use the following command to view the respective data information, and input the Dock terminal,

```
#View RGB image topic data content
ros2 topic echo /camera/color/image_raw
#Viewing Depth Image Topic Data Content
ros2 topic echo /camera/depth/image_raw
```

First, take 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
```


First, take a frame of RGB color image information and take a look. This explains the basic information of the image, an important value, 【encoding 】, where the value is 【 rgb8 】. This value indicates that the encoding format of this frame of image is rgb8, which needs to be referenced when performing data conversion later。

Similarly, below is the 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
```

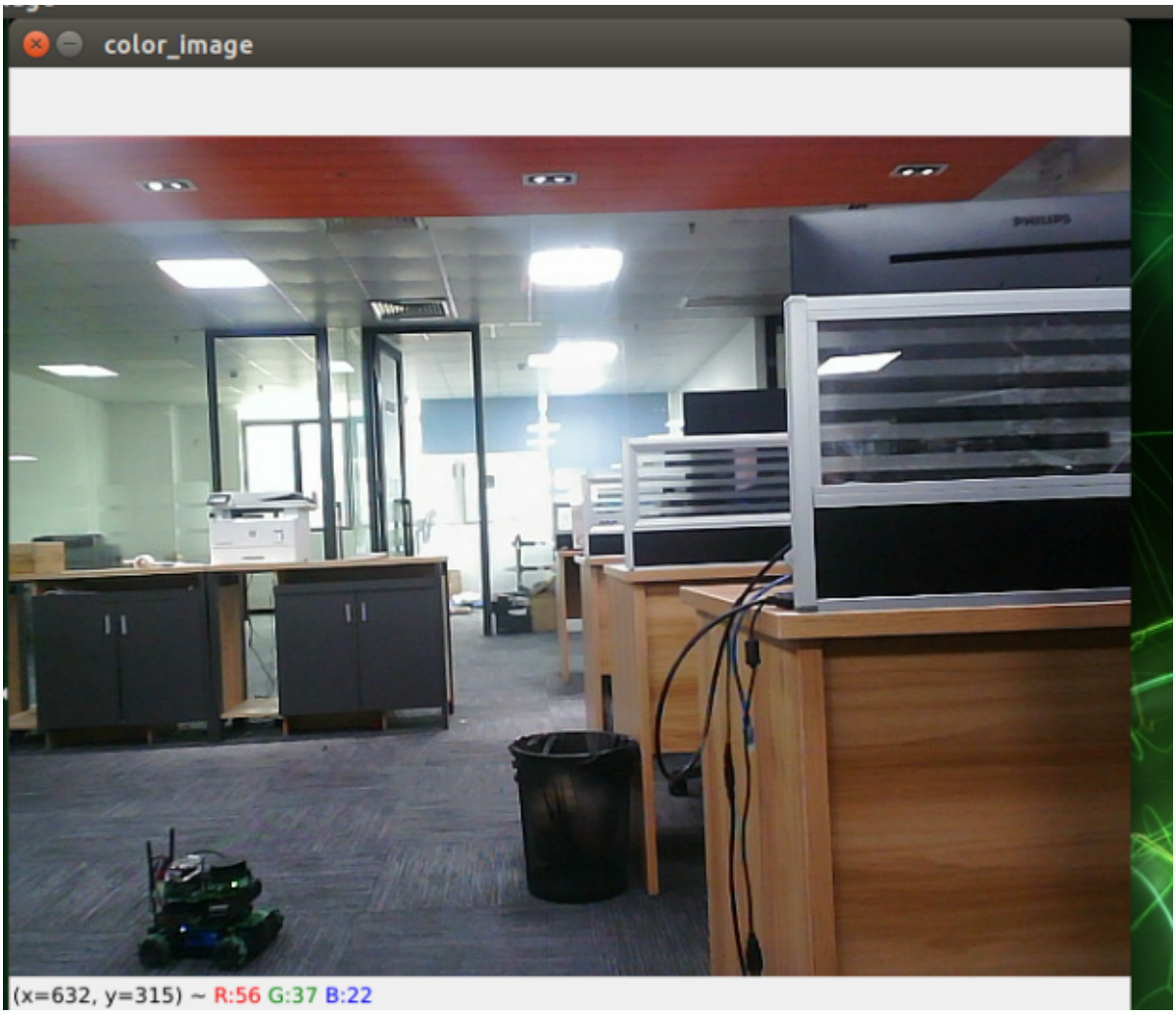

The encoding value here is 【16UC1】.

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

2.1、Run Command

Docker terminal input，

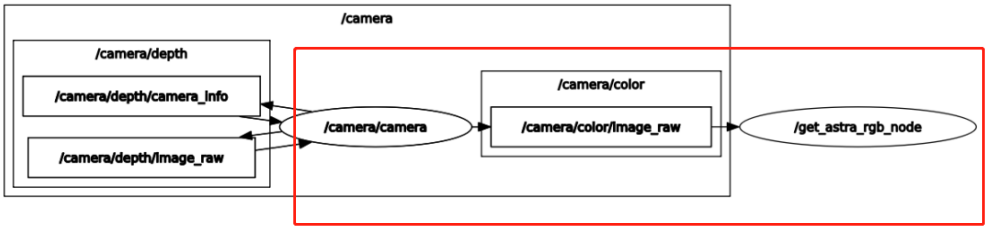
```
#RGB color image display node
ros2 run yahboomcar_visual astra_rgb_image
#Running Astrapro Camera
ros2 launch astra_camera astra_pro.launch.xml
```



2.2、View node communication diagram

Docker terminal input,

```
ros2 run rqt_graph rqt_graph
```



2.3、Core code parsing

Code reference path,

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

As can be seen from 2.2,/get_ Astra_ Rgb_ Node node subscribed to/camera/color/image_ Raw's topic, and then through data conversion, the topic data is converted into car image data for publishing. The code is as follows,

```
#Import opecv library and cv_ Bridge Library
import cv2 as cv
from cv_bridge import CvBridge
#Creating CvBridge Objects
self.bridge = CvBridge()
#Define a subscriber to subscribe to RGB color image topic data published by deep
camera nodes
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 depth image topic information and display depth images

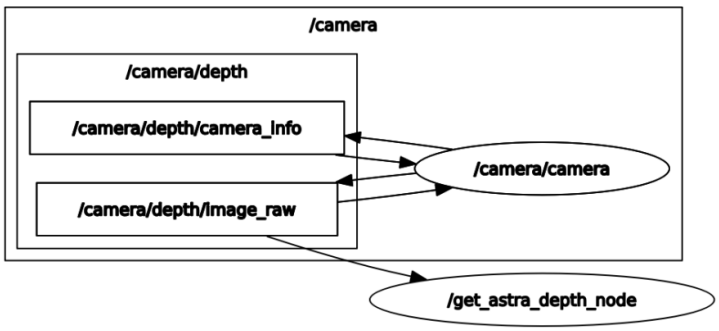
3.1、Run Command

```
#RGB color image display node
ros2 run yahboomcar_visual astra_depth_image
#Running Astrapro Camera
ros2 launch astra_camera astra_pro.launch.xml
```

3.2、View node communication diagram

Docker terminal input,

```
ros2 run rqt_graph rqt_graph
```



3.3、Core code parsing

Code reference path,

```
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_visual/yahboomcar_visual/as
tra_depth_image.py
```

The basic implementation process is the same as RGB color image display, subscribed to the/camera/depth/image published by the depth camera node_ Raw's topic data is then converted into image data through data conversion, with the following code,

```
#Import opecv library and cv_ Bridge Library
import cv2 as cv
from cv_bridge import CvBridge
#Creating CvBridge Objects
self.bridge = CvBridge()
#Define a subscriber to subscribe to depth camera node published depth image
topic data
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 publish the converted image data

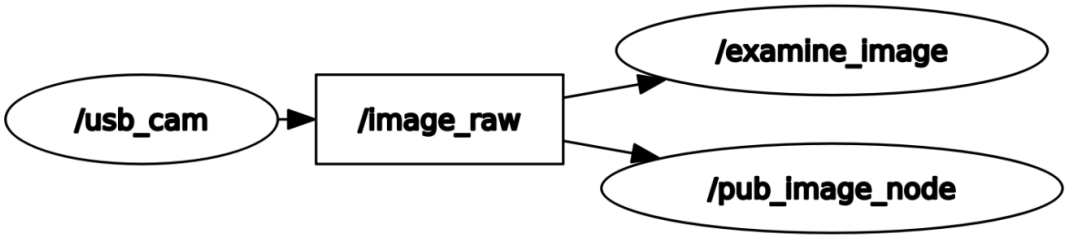
4.1、Run Command

```
#Run and publish image topic data nodes
ros2 run yahboomcar_visual pub_image
#Run USB camera topic node
ros2 launch usb_cam demo_launch.py
```

4.2、View node communication diagram

Docker terminal input,

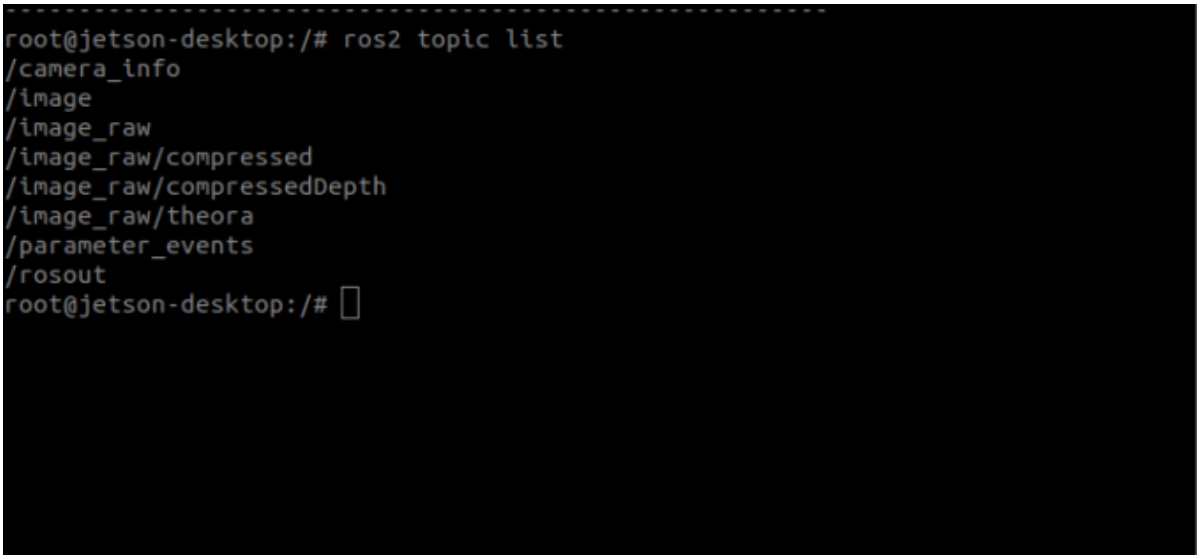
```
ros2 run rqt_graph rqt_graph
```



4.3、Viewing Topic Data

First, check which image topics have been published and input them through the Docker terminal,

```
ros2 topoic list
```



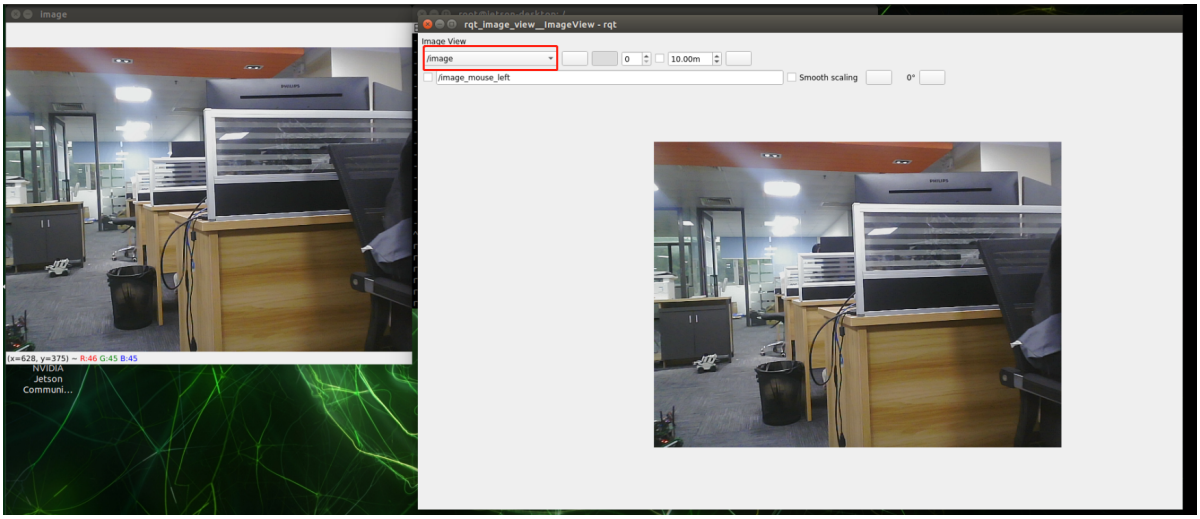
The/image is the topic data we have published. Use the following command to print and view the data content of this topic,

```
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 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 parsing

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, and the program first subscribed to/image_ Raw's topic data is then converted into image data, but here we also perform lattice transformation to convert image data into topic data and publish it, which 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
#Creating CvBridge Objects
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)
#Defined 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")
#The image topic data (cv2uto_imgmsg) converted from image data is then published
msg = self.bridge.cv2_to_imgmsg(frame, "bgr8")
self.pub_img.publish(msg)
```