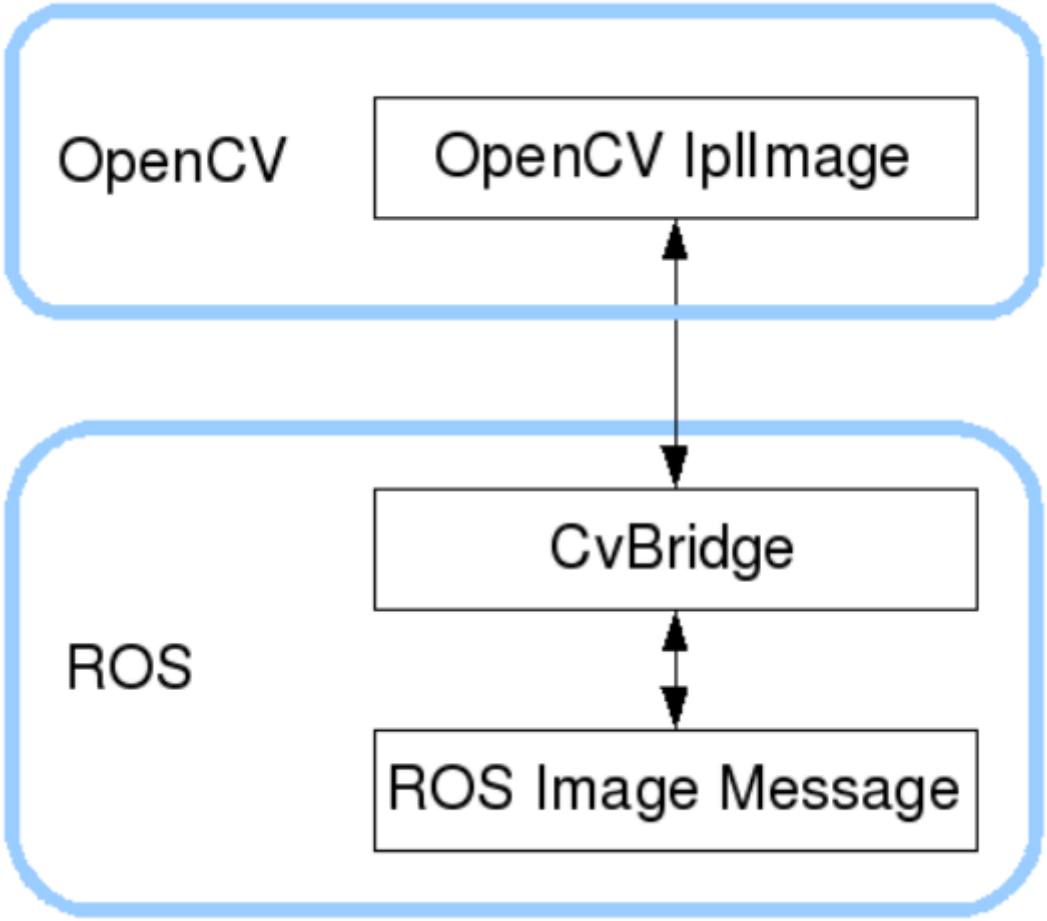
1.ROS+Opencv application

This lesson takes a monocular camera as an example.

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

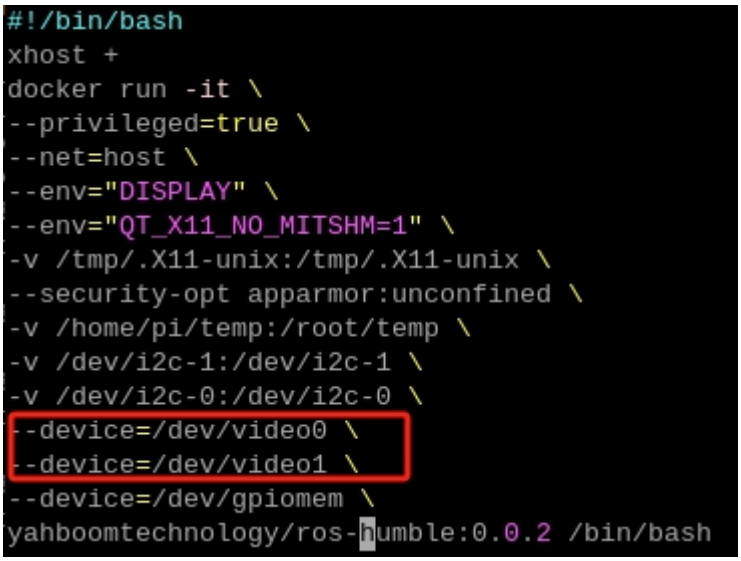


This lesson uses three cases to show how to use CvBridge for data conversion.

1. Monocular camera

Before driving the camera, the host needs to be able to identify the camera device; when entering the docker container, you need to mount this USB device to identify 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 this to the startup file.

```
--device=/dev/video0 \
--device=/dev/video1 \
```



1.1, Start the camera

1.1.1, Source code path

```
cd  ~/opt/ros/humble/share/usb_cam/
```

1.1.2, Installation steps

Take starting a monocular camera as an example, you can use the command to directly download the camera driver file of ros2.

Note: The factory docker image has been installed and does not need to be installed again.

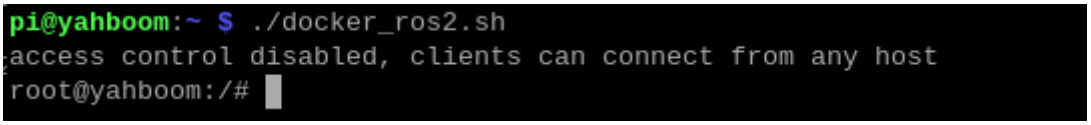
```
sudo apt-get install ros-humble-usb-cam
```

1.1.3, Start the camera

Open a terminal and enter the following command to enter docker,

```
./docker_ros2.sh
```

The following interface appears, indicating that you have successfully entered docker.



Enter the following command to start the camera,

```
ros2 run usb_cam  usb_cam_node_exe
```

1.2. View the camera topic

Open a new terminal, enter the same docker, and change the following da8c4f47020a to the ID displayed in the actual terminal

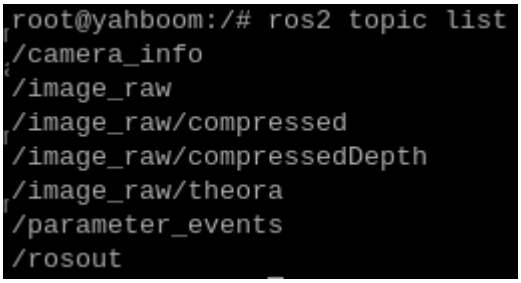
```
docker ps
```

```
docker exec -it da8c4f47020a   /bin/bash
```

Enter the following command to query the node,

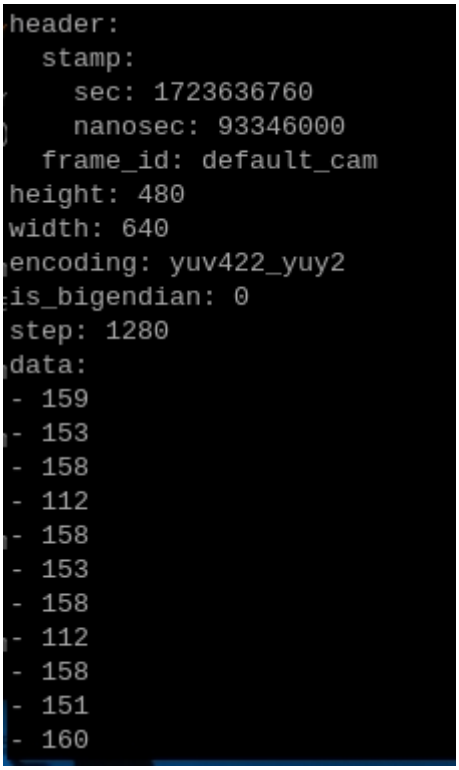
Enter the dock terminal,

```
ros2 topic list
```



The main topic is image data. Here we only parse RGB color images. Use the following commands to view the respective data information. Enter in the dock terminal:

```
#View RGB image topic data content
ros2 topic echo /image_raw
```



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 yuv422, this needs to be referred to when doing data conversion later.

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

2.1. Run command

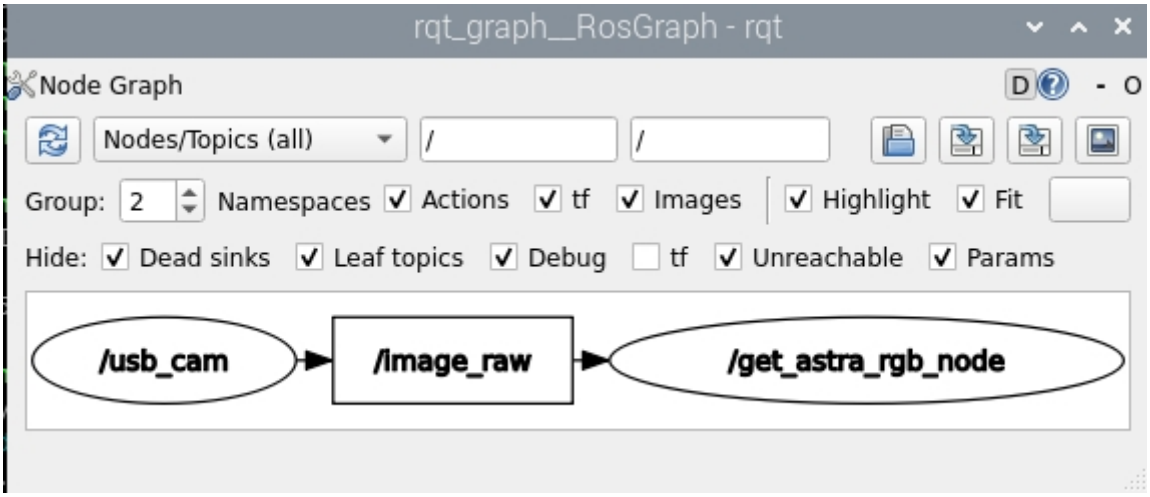
Enter in the docker terminal,

```
ros2 run usb_cam  usb_cam_node_exe
ros2 run yahboomcar_visual astra_rgb_image
```

2.2. View the node communication diagram

Enter in the docker terminal,

```
ros2 run rqt_graph rqt_graph
```



2.3, core code analysis

Code reference path,

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

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

```
Import opecv and cv_bridge libraries
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,'/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 image data and then publish converted image data

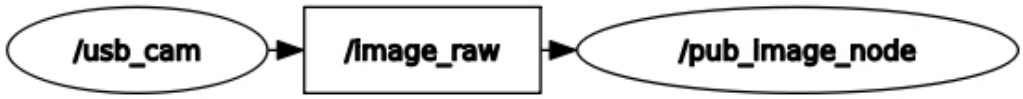
3.1. Run command

```
#Run the publish image topic data node
ros2 run yahboomcar_visual pub_image
#Run the usb camera topic node
ros2 run usb_cam usb_cam_node_exe
```

3.2. View the node communication diagram

Enter in the docker terminal,

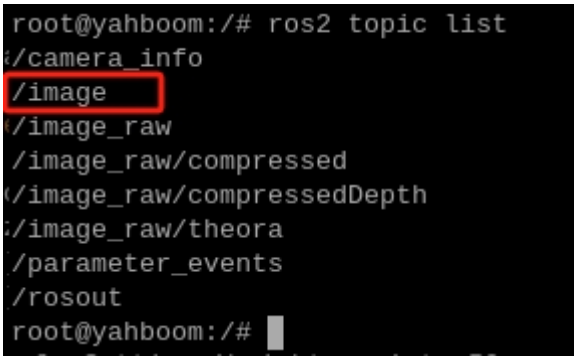
```
ros2 run rqt_graph rqt_graph
```



3.3. View topic data

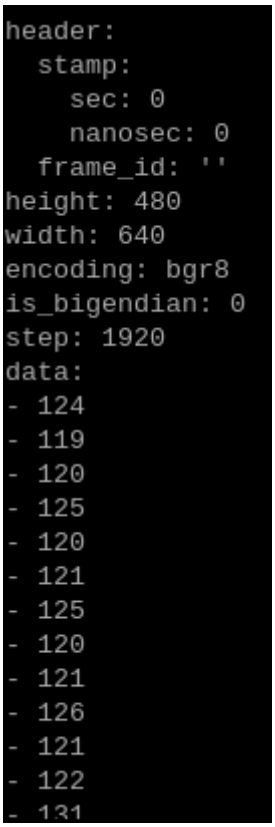
First check which image topics are published, enter in the docker terminal,

```
ros2 topic list
```



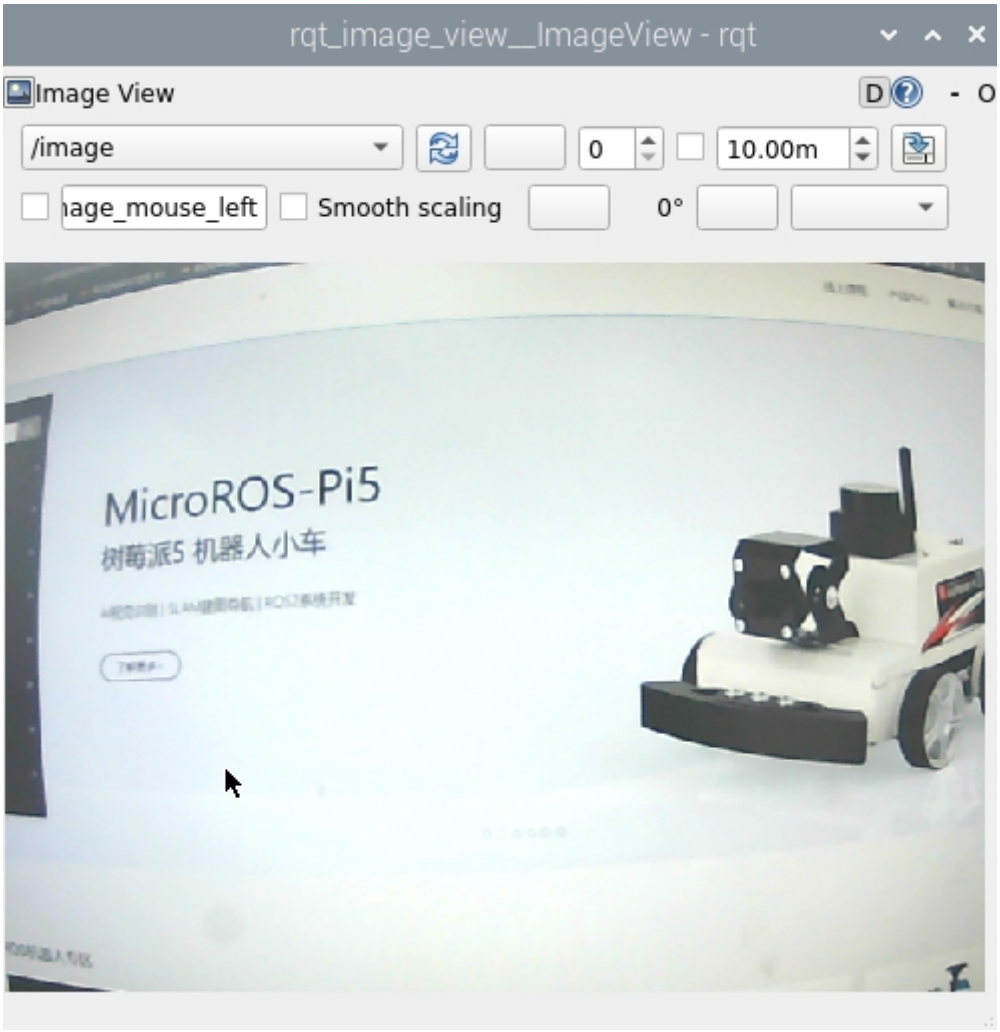
The /image is the topic data we published. Use the following command to print the data content of this topic.

```
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_raw in the upper left corner to view the image.

3.4, core code analysis

Code path,

```
/root/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, a 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,'/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")
#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)
```