8. ROS+Opencv foundation

This lesson uses a USB camera as an example.

8.1. Overview

Wiki: http://wiki.ros.org/cv_bridge/

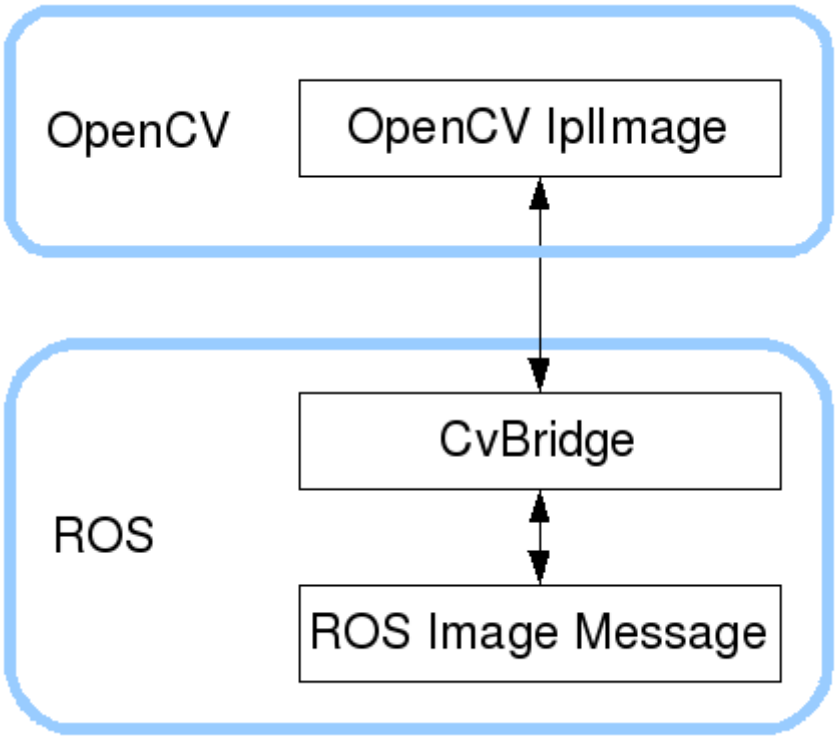
Tutorials: http://wiki.ros.org/cv_bridge/Tutorials

Source code: https://github.com/ros-perception/vision_opencv.git

Function package location: ~/transbot_ws/src/transbot_visual

ROS has already integrated Opencv3.0 and above during the installation process, so there is almost no need to think too much about the installation configuration. ROS uses its own sensor_msgs/Image message format transfers images 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:



Although the installation configuration does not require too much consideration, the usage environment still needs to be configured, mainly the two files [package.xml] and [CMakeLists.txt]. This function package not only uses [CvBridge], but also requires [Opencv] and [PCL], so they are configured together.

- package.xml

Add the following content

```
    <build_depend>sensor_msgs</build_depend>
    <build_export_depend>sensor_msgs</build_export_depend>
    <exec_depend>sensor_msgs</exec_depend>

    <build_depend>std_msgs</build_depend>
    <build_export_depend>std_msgs</build_export_depend>
    <exec_depend>std_msgs</exec_depend>

    <build_depend>cv_bridge</build_depend>
    <build_export_depend>cv_bridge</build_export_depend>
    <exec_depend>cv_bridge</exec_depend>

    <build_depend>transbot_msgs</build_depend>
    <build_export_depend>transbot_msgs</build_export_depend>
    <exec_depend>transbot_msgs</exec_depend>

    <exec_depend>image_transport</exec_depend>
```

[cv_bridge]: Image conversion dependency package.

[transbot_msgs]: Custom message dependency package.

-CMakeLists.txt

This file has a lot of configuration content, please check the source file for specific content.

8.2, USB camera

8.2.1. Start USB camera

jetson motherboard/Raspberry Pi 4B

```
roslaunch usb_cam usb_cam-test.launch
```

Raspberry Pi 5

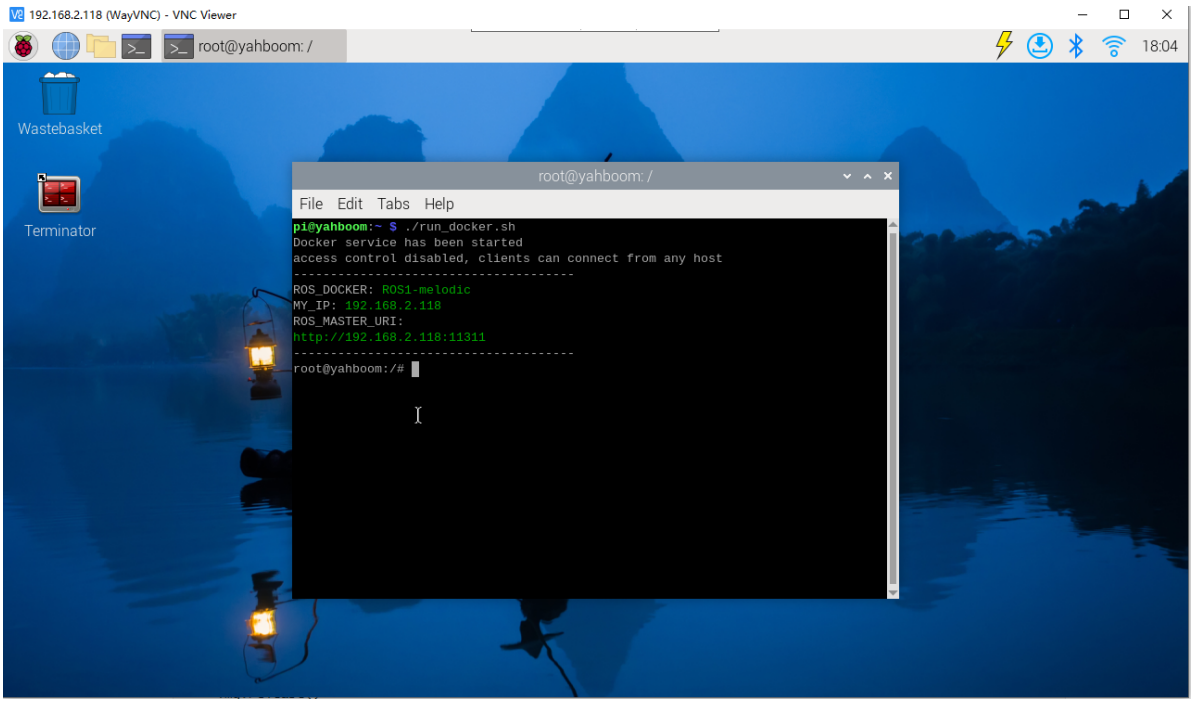
Before running, please confirm that the large program has been permanently closed

Enter docker

Note: If there is a terminal that automatically starts docker, or there is a docker terminal that has been opened, you can directly enter the docker terminal to run the command, and there is no need to manually start docker

Start docker manually

```
./run_docker.sh
```




```
roslaunch usb_cam usb_cam-test.launch
```

View topics

jetson motherboard/Raspberry Pi 4B

```
rostopic list
```

Raspberry Pi 5

Enter the same docker from multiple terminals

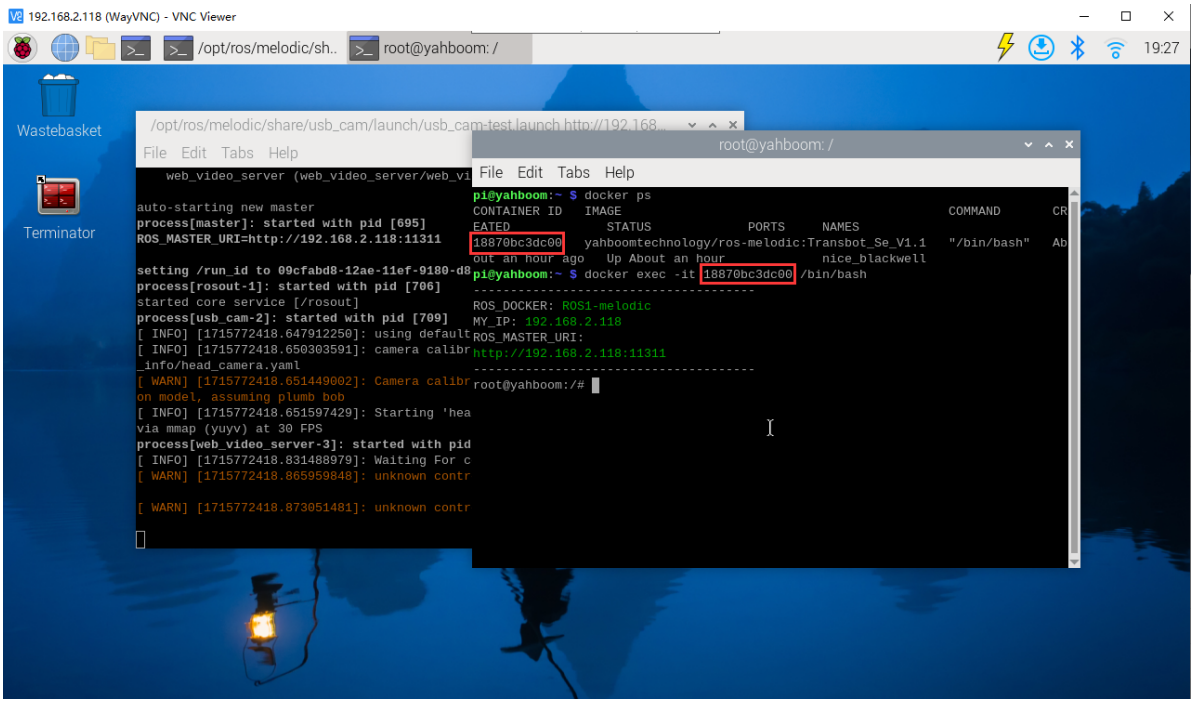
Keep the program of the previous docker terminal running and open a new terminal

Enter the following command

```
docker ps
```

Enter the same docker and use the following 18870bc3dc00 to modify the ID displayed on the actual terminal.

```
docker exec -it 18870bc3dc00 /bin/bash
```




```
rostopic list
```

You can see that there are many topics, here are a few commonly used ones.

Topic name	Data type
/usb_cam/image_raw	sensor_msgs/Image
/usb_cam/image_raw/compressed	sensor_msgs/CompressedImage
/usb_cam/image_raw/compressedDepth	sensor_msgs/CompressedImage
/usb_cam/image_raw/theora	theora_image_transport/Packet

Check the encoding format of the topic: rostopic echo +[topic]+encoding, for example

jetson motherboard/Raspberry Pi 4B

```
rostopic echo /usb_cam/image_raw/encoding
```

Raspberry Pi 5

Enter the same docker from multiple terminals

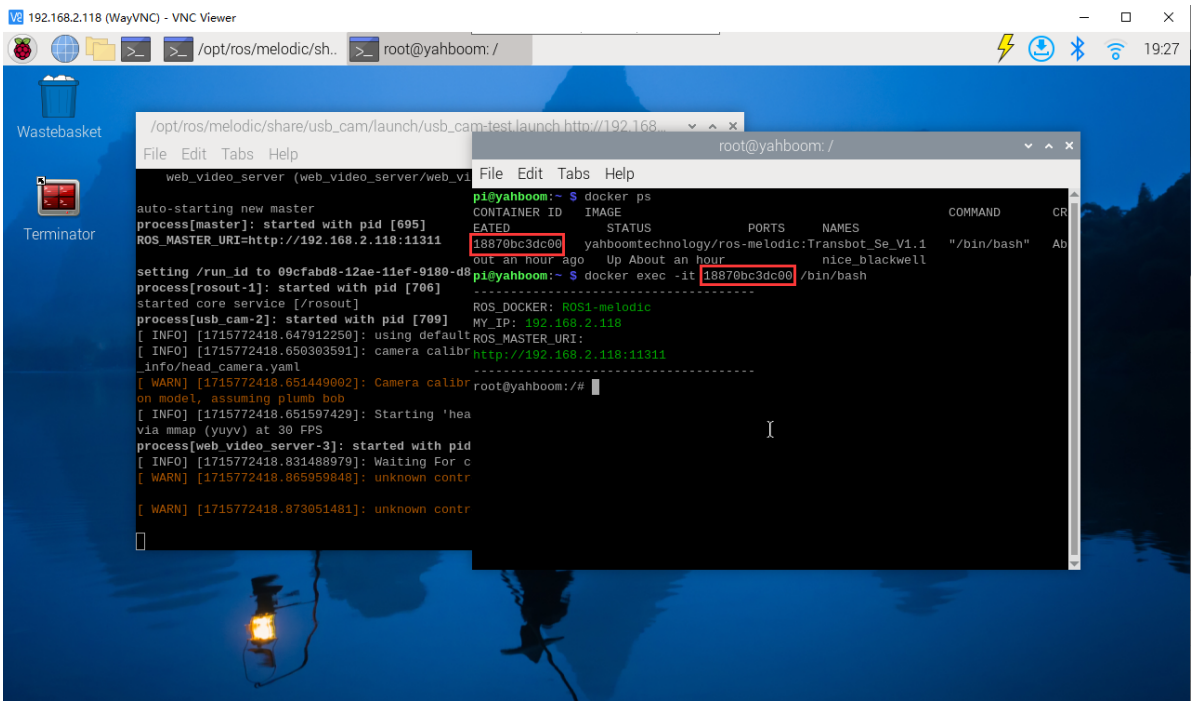
Keep the program of the previous docker terminal running and open a new terminal

Enter the following command

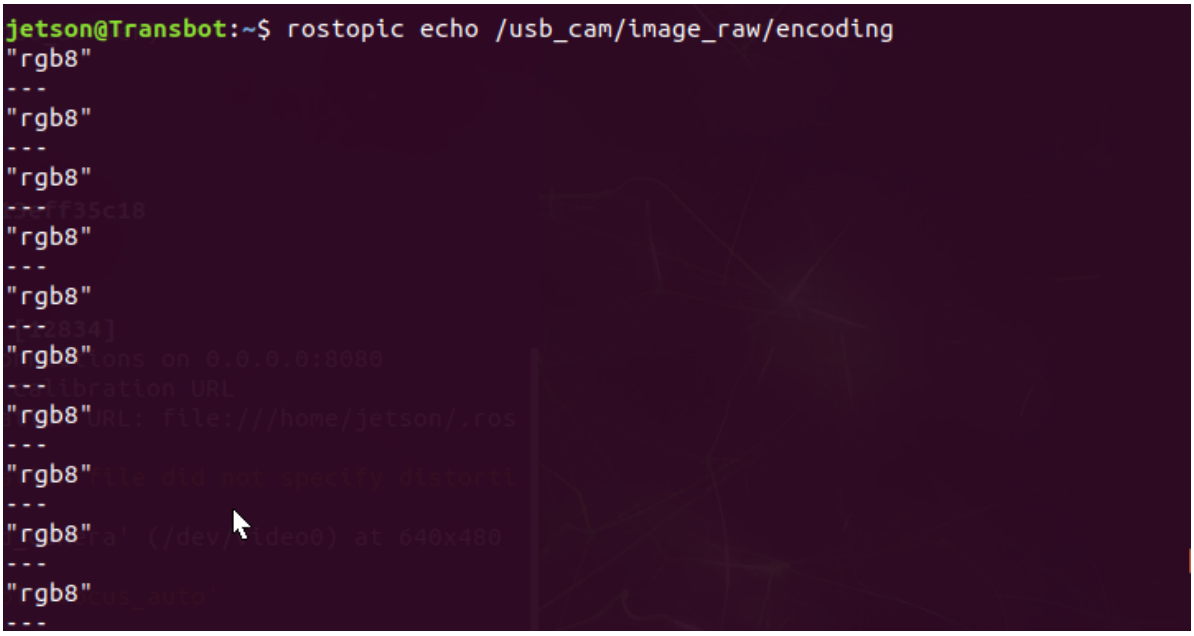
```
docker ps
```

Enter the same docker and use the following 18870bc3dc00 to modify the ID displayed on the actual terminal.

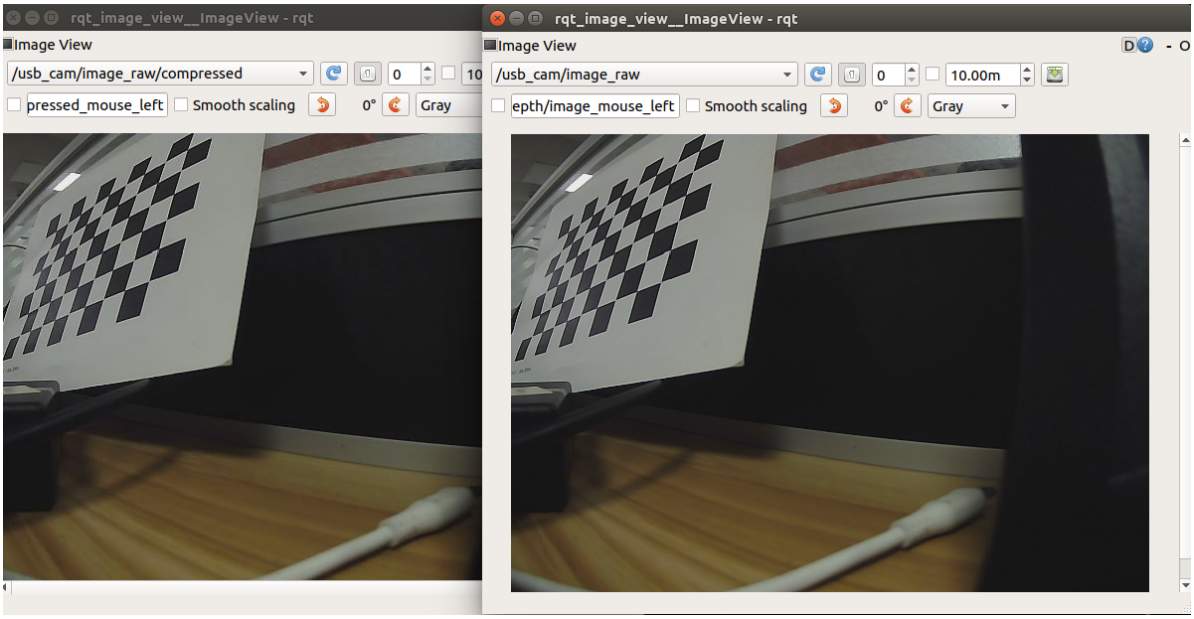
```
docker exec -it 18870bc3dc00 /bin/bash
```




```
rostopic echo /usb_cam/image_raw/encoding
```



Topics followed by [compressed] or [compressedDepth] are compressed topics. When ROS images are transmitted, data packet loss may occur due to factors such as the network, main control running speed, main control running memory, and huge video stream data. Obtain Not on topic. So I have no choice but to subscribe to compressed topics. Open two images at the same time to subscribe to different topic tests. If the device performance is very good and the network is also good, you will not see any changes. Otherwise, you will find that the topic after image compression will be much smoother.



8.2.2. Start the color graph subscription node

jetson motherboard/Raspberry Pi 4B

```
roslaunch transbot_visual usb_cam_image.launch # launch
rosrun transbot_visual usb_cam_image.py # py
rosrun transbot_visual usb_cam_image # C++
```

Raspberry Pi 5

Enter the same docker from multiple terminals

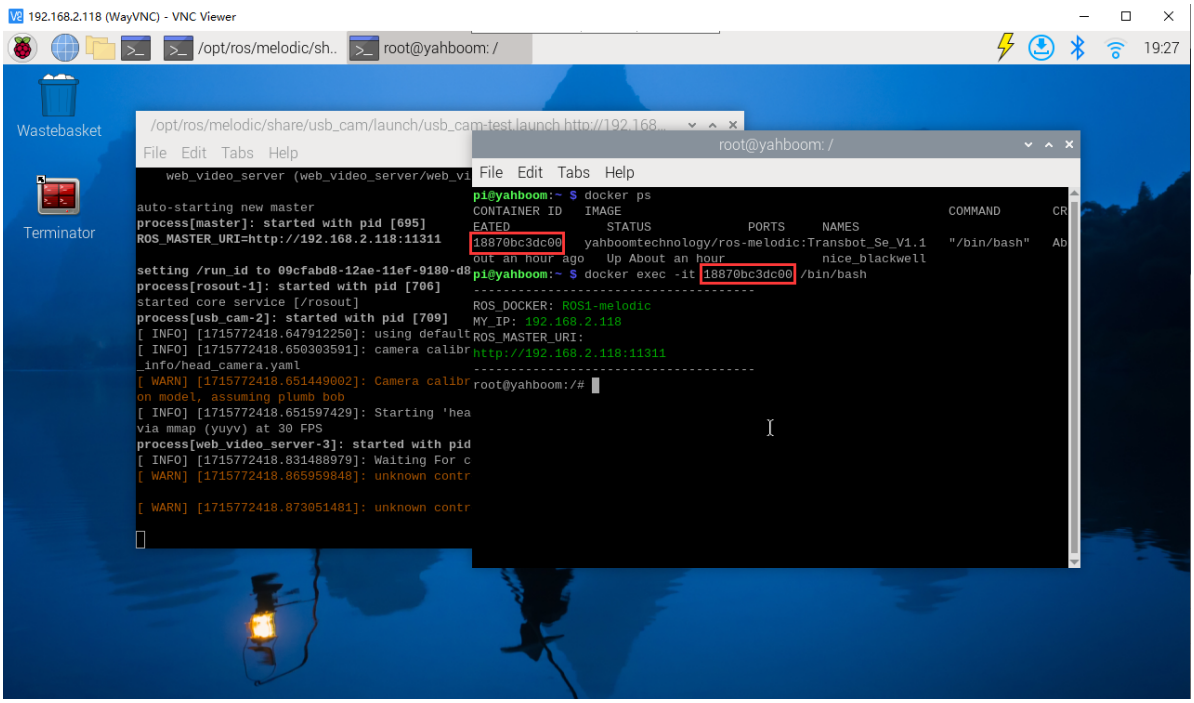
Keep the program of the previous docker terminal running and open a new terminal

Enter the following command

```
docker ps
```

Enter the same docker and use the following 18870bc3dc00 to modify the ID displayed on the actual terminal.

```
docker exec -it 18870bc3dc00 /bin/bash
```




```
roslaunch transbot_visual usb_cam_image.launch # launch
rosrun transbot_visual usb_cam_image.py # py
rosrun transbot_visual usb_cam_image # C++
```

View node graph

jetson motherboard/Raspberry Pi 4B

```
rqt_graph
```

Raspberry Pi 5

Enter the same docker from multiple terminals

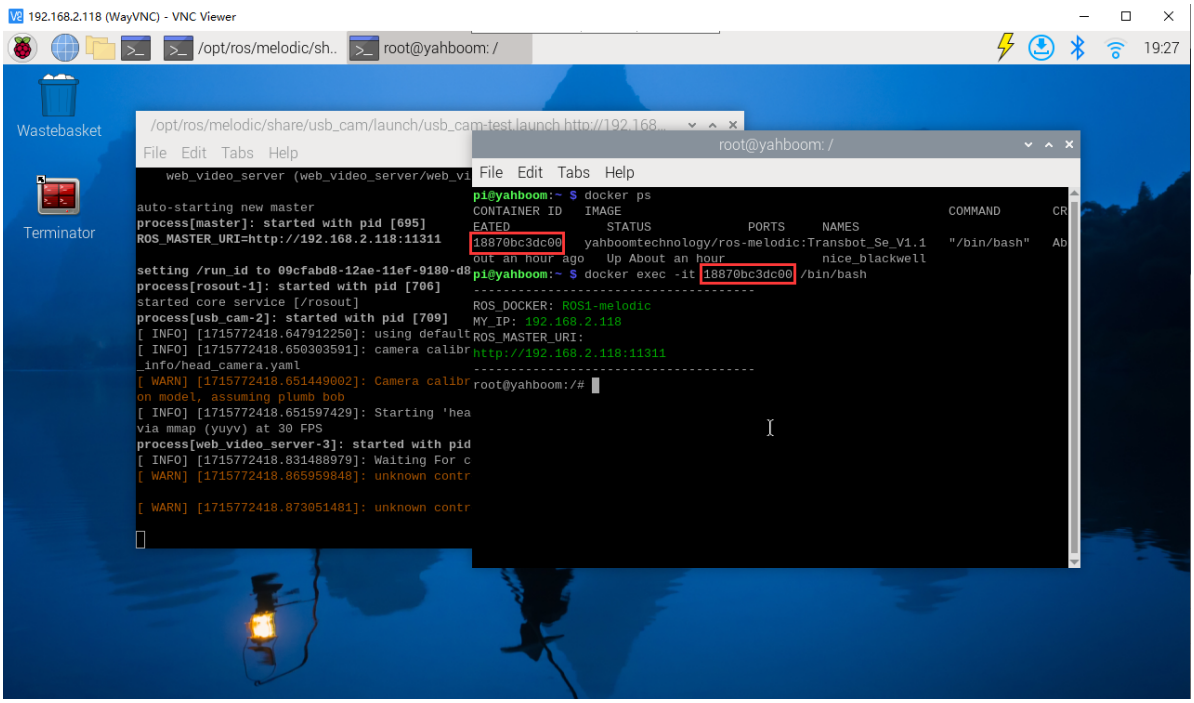
Keep the program of the previous docker terminal running and open a new terminal

Enter the following command

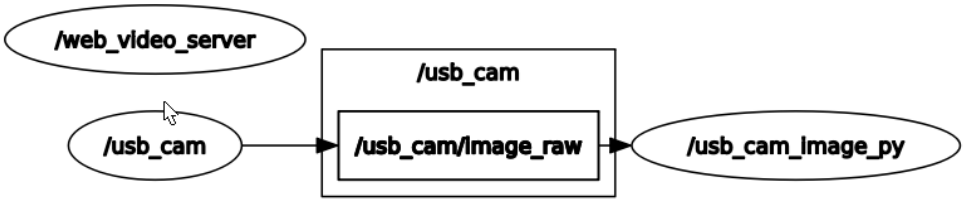
```
docker ps
```

Enter the same docker and use the following 18870bc3dc00 to modify the ID displayed on the actual terminal.

```
docker exec -it 18870bc3dc00 /bin/bash
```




```
rqt_graph
```



There are many nodes here, we mainly look at the one below. [/usb_cam_iamge_py] is the node we wrote.

- py code analysis

Create a subscriber: The topic of subscription is ["/usb_cam/image_raw"], data type [Image], callback function [topic()]

```
sub = rospy.Subscriber("/usb_cam/image_raw", Image, topic)
```

Use [CvBridge] for data conversion. What you need to pay attention to here is the encoding format. If the encoding format is incorrect, the converted image will have problems.

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

- c++ code analysis

Similar to py code

```
//Create a receiver.
ros::Subscriber subscriber = n.subscribe<sensor_msgs::Image>
("/usb_cam/image_raw", 10, RGB_Callback);
//Create cv_bridge example
cv_bridge::CvImagePtr cv_ptr;
// data conversion
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
```

8.2.3. Start color image inversion

jetson motherboard/Raspberry Pi 4B

```
roslaunch transbot_visual usb_cam_flip.launch # launch
rosrun transbot_visual usb_cam_flip.py # py
```

Raspberry Pi 5

Enter the same docker from multiple terminals

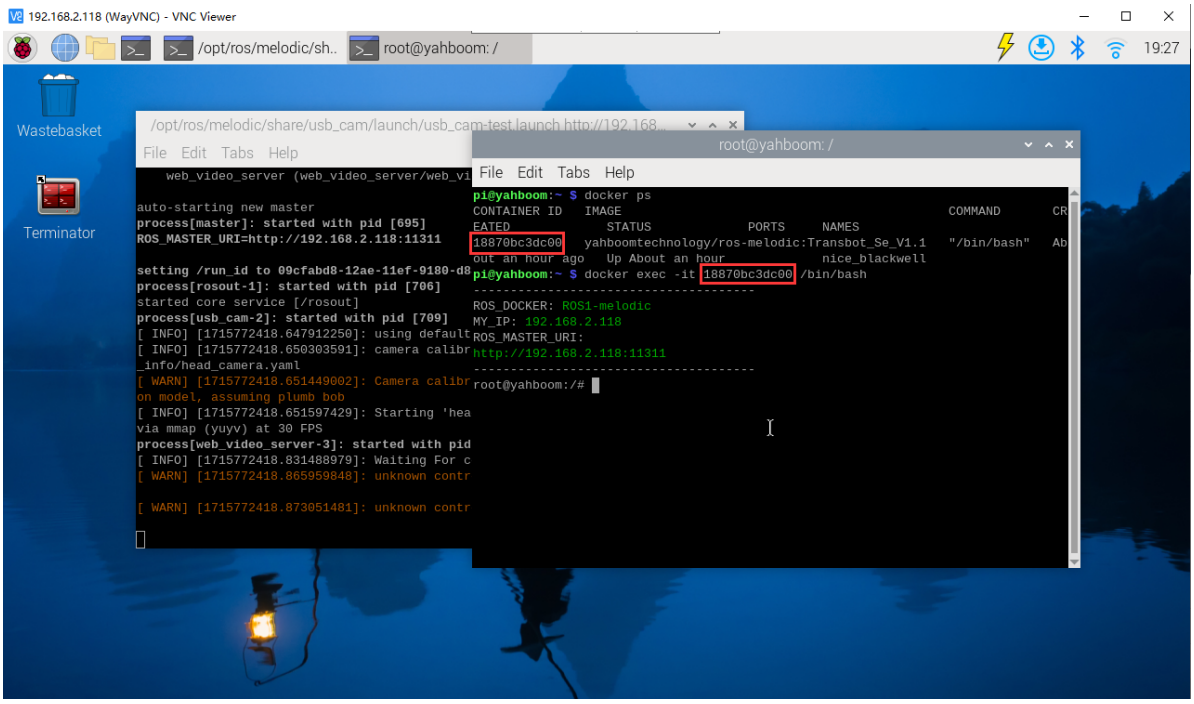
Keep the program of the previous docker terminal running and open a new terminal

Enter the following command

```
docker ps
```

Enter the same docker and use the following 18870bc3dc00 to modify the ID displayed on the actual terminal.

```
docker exec -it 18870bc3dc00 /bin/bash
```




```
roslaunch transbot_visual usb_cam_flip.launch # launch
rosrun transbot_visual usb_cam_flip.py # py
```

Image viewing

jetson motherboard/Raspberry Pi 4B

```
rqt_image_view
```

Raspberry Pi 5

Enter the same docker from multiple terminals

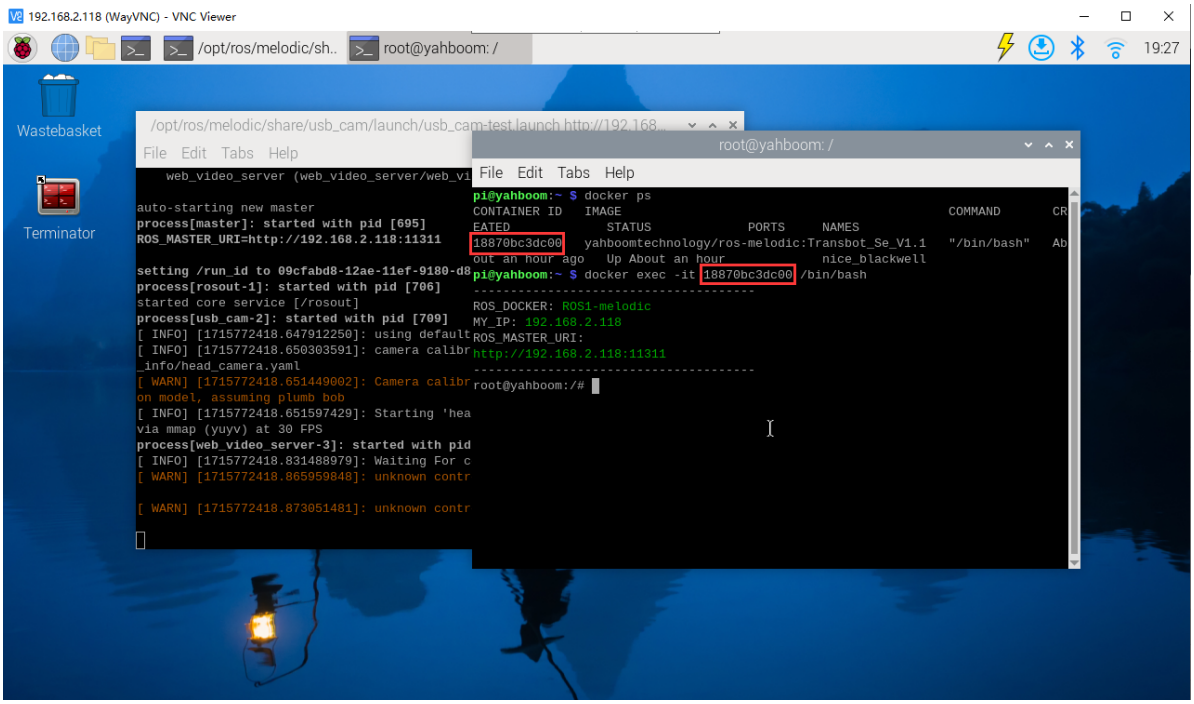
Keep the program of the previous docker terminal running and open a new terminal

Enter the following command

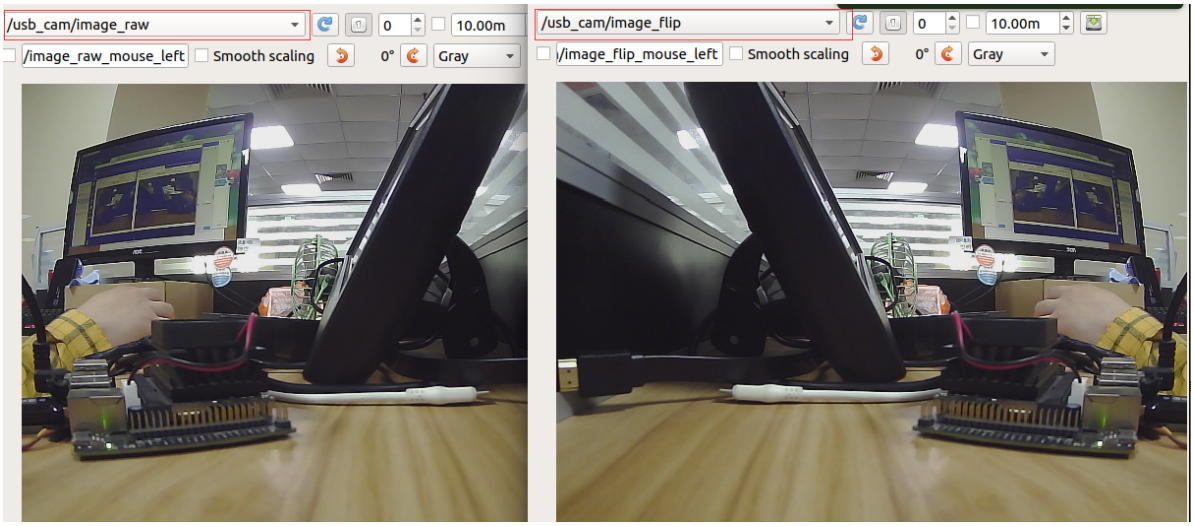
```
docker ps
```

Enter the same docker and use the following 18870bc3dc00 to modify the ID displayed on the actual terminal.

```
docker exec -it 18870bc3dc00 /bin/bash
```




```
rqt_image_view
```



- py code analysis

Two subscribers and two publishers are created here, one processing general image data and one processing compressed image data.

1. Create a subscriber

The subscribed topic is ["/usb_cam/image_raw"], the data type [Image], and the callback function [topic()].

The subscribed topic is ["/usb_cam/image_raw/compressed"], the data type [CompressedImage], and the callback function [compressed_topic()].

2. Create publisher

The published topic is ["/usb_cam/image_flip"], the data type [Image], and the queue size [10].

The published topic is ["/usb_cam/image_flip/compressed"], the data type [CompressedImage], and the queue size [10].

```
sub_img = rospy.Subscriber("/usb_cam/image_raw", Image, topic)
pub_img = rospy.Publisher("/usb_cam/image_flip", Image, queue_size=10)
sub_comimg = rospy.Subscriber("/usb_cam/image_raw/compressed", CompressedImage,
compressed_topic)
pub_comimg = rospy.Publisher("/usb_cam/image_flip/compressed", CompressedImage,
queue_size=10)
```

3. Callback function

```
# Normal image transfer processing
def topic(msg):
     if not isinstance(msg, Image):
         return
     bridge = CvBridge()
     frame = bridge.imgmsg_to_cv2(msg, "bgr8")
     # Opencv processing images
     frame = cv.resize(frame, (640, 480))
     frame = cv.flip(frame, 1)
     # opencv mat -> ros msg
     msg = bridge.cv2_to_imgmsg(frame, "bgr8")
     # After image processing is completed, publish it directly
     pub_img.publish(msg)

# Compressed image transmission processing
def compressed_topic(msg):
     if not isinstance(msg, CompressedImage): return
     bridge = CvBridge()
     frame = bridge.compressed_imgmsg_to_cv2(msg, "bgr8")
     # Opencv processing images
     frame = cv.resize(frame, (640, 480))
     frame = cv.flip(frame, 1)
     # Create CompressedIamge
     msg = CompressedImage()
     msg.header.stamp = rospy.Time.now()
     # Image data conversion
     msg.data = np.array(cv.imencode('.jpg', frame)[1]).tostring()
```

```
        # After image processing is completed, publish it directly
        pub_comimg.publish(msg)
```

8.2.4. Release image

The main thing is to subscribe to the USB camera image topic and publish the image topic. It is relatively simple and will not be introduced again. The startup command is as follows:

jetson motherboard/Raspberry Pi 4B

```
roslaunch transbot_visual usb_cam_to_image.launch # launch
rosrun transbot_visual usb_cam_to_image.py # py
```

Raspberry Pi 5

Enter the same docker from multiple terminals

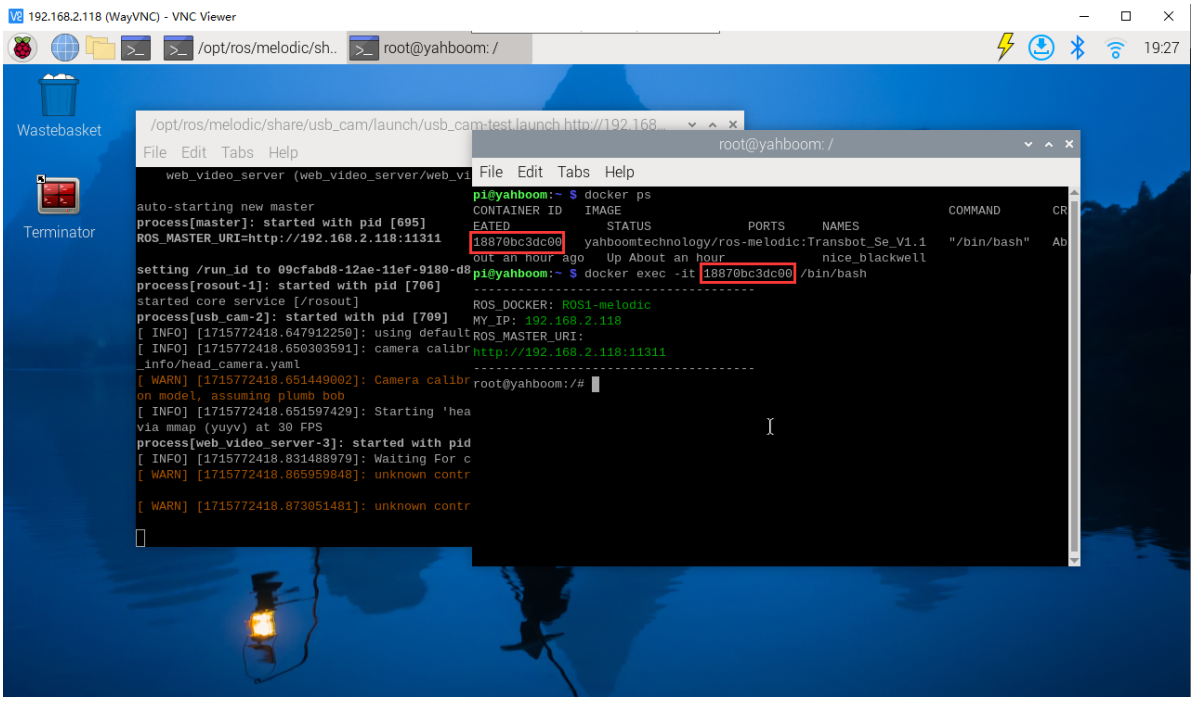
Keep the program of the previous docker terminal running and open a new terminal

Enter the following command

```
docker ps
```

Enter the same docker and use the following 18870bc3dc00 to modify the ID displayed on the actual terminal.

```
docker exec -it 18870bc3dc00 /bin/bash
```




```
roslaunch transbot_visual usb_cam_to_image.launch # launch
rosrun transbot_visual usb_cam_to_image.py # py
```