

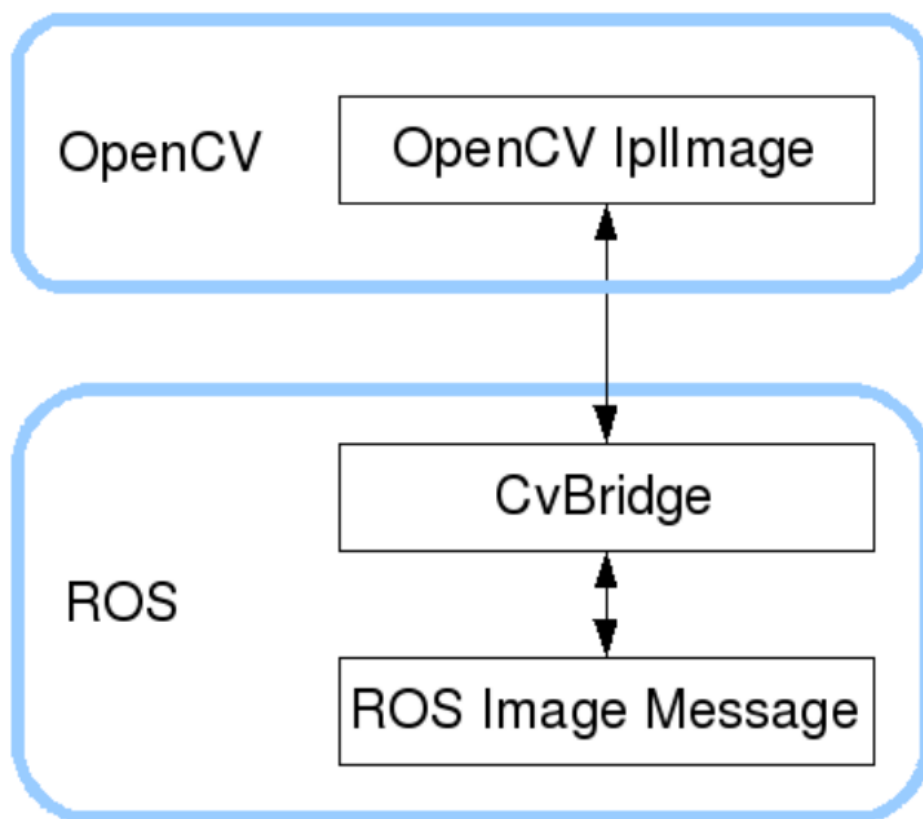
ROS+Opencv basics

Code path: ~/orbbec_ws/src/astra_visual

1. Overview

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 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:



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:

```
<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>
<exec_depend>image_transport</exec_depend>
```

【cv_bridge】：Image conversion dependency package.

- CMakeLists.txt

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

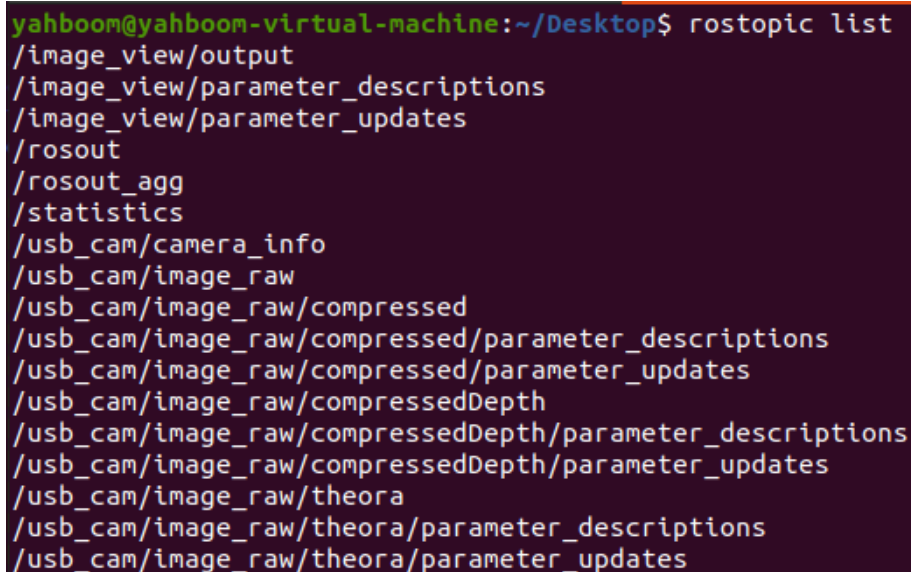
2. Start USB camera

Input following command:

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

View topics

```
rostopic list
```



```
yahboom@yahboom-virtual-machine:~/Desktop$ rostopic list
/image_view/output
/image_view/parameter_descriptions
/image_view/parameter_updates
/rosout
/rosout_agg
/statistics
/usb_cam/camera_info
/usb_cam/image_raw
/usb_cam/image_raw/compressed
/usb_cam/image_raw/compressed/parameter_descriptions
/usb_cam/image_raw/compressed/parameter_updates
/usb_cam/image_raw/compressedDepth
/usb_cam/image_raw/compressedDepth/parameter_descriptions
/usb_cam/image_raw/compressedDepth/parameter_updates
/usb_cam/image_raw/theora
/usb_cam/image_raw/theora/parameter_descriptions
/usb_cam/image_raw/theora/parameter_updates
```

The commonly used ones are the following,

/Camera/color/image_Raw: RGB color image topic

/Camera/depth/image_Raw: Depth depth image topic

/Camera/ir/image_Raw: IR infrared image topic

/Camera/depth/points: topic of deep point cloud data

View the encoding format of the topic: robust echo+[topic]+encoding.

For example:

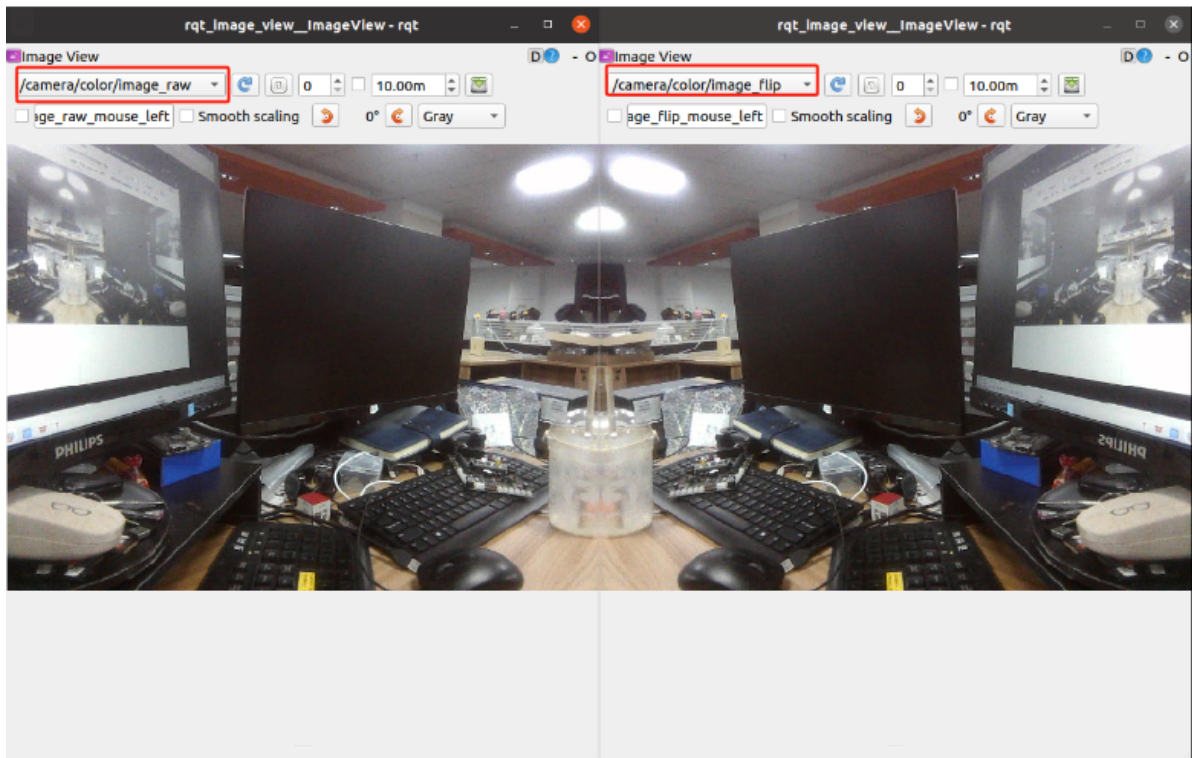
```
rostopic echo /camera/color/image_raw/encoding
rostopic echo /camera/depth/image_raw/encoding
```

```
yahboom@yahboom-virtual-machine:~$ rostopic echo /camera/color/image_raw/encoding
"rgb8"
---
"rgb8"
---
"rgb8"
---
"rgb8"
---
"rgb8"
```

```
yahboom@yahboom-virtual-machine:~$ rostopic echo /camera/depth/image_raw/encoding
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
```

3. Start the color picture subscription node

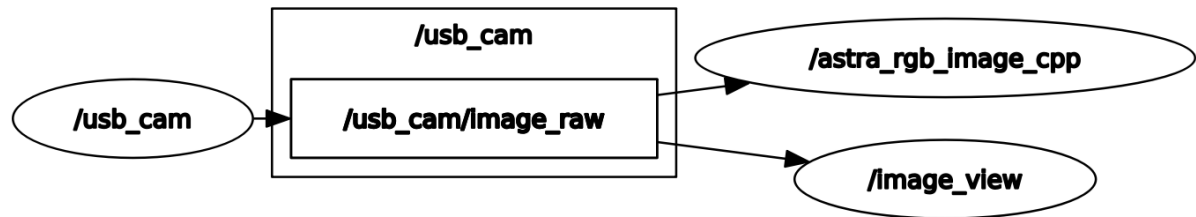
```
roslaunch astra_visual astra_rgb_image.py # py
roslaunch astra_visual astra_rgb_image # C++
```



View the node graph

Input following command:

```
rqt_graph
```



- py code analysis

Create a subscriber: The subscription topic is `["/camera/color/image_raw"]`, the data type is `[Image]`, and the callback function `topic()`

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

Use `CvBridge` to convert data. What you need to pay attention to here is the encoding format. If the encoding format is incorrect, the converted image will have problems.

```
bridge = CvBridge()
```

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

- c++ code analysis

```
//Create a receiver
ros::Subscriber subscriber =
n.subscribe<sensor_msgs::Image>("/camera/color/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);
```

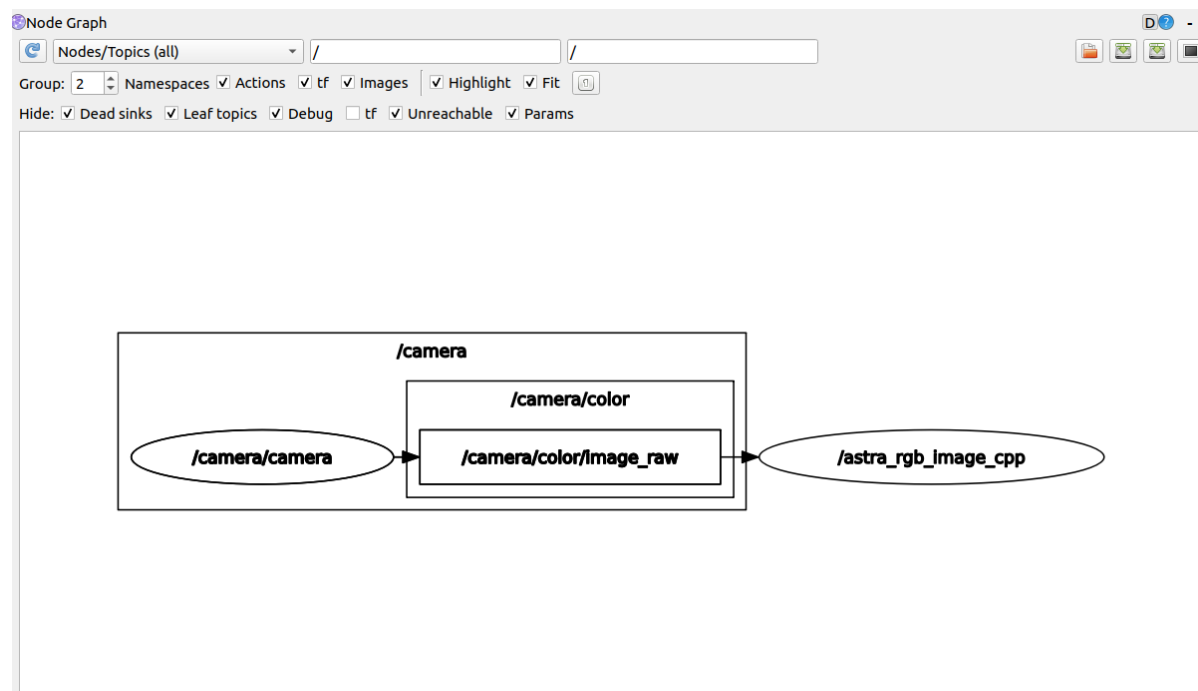
4. Start the depth map subscription node

```
#Select 1 of the following commands to start  
roslaunch astra_visual astra_depth_image.py # py  
roslaunch astra_visual astra_depth_image # C++
```



View node:

rqt_graph



- py code analysis

Create a subscriber: The topic of subscription is `"/camera/depth/image_raw"` , data type `Image` , callback function `topic()` .

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

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

```
# Encoding format
encoding = ['16UC1', '32FC1']
# You can switch between different encoding formats to test the effect
frame = bridge.imgmsg_to_cv2(msg, encoding[1])
```

- C++ code analysis

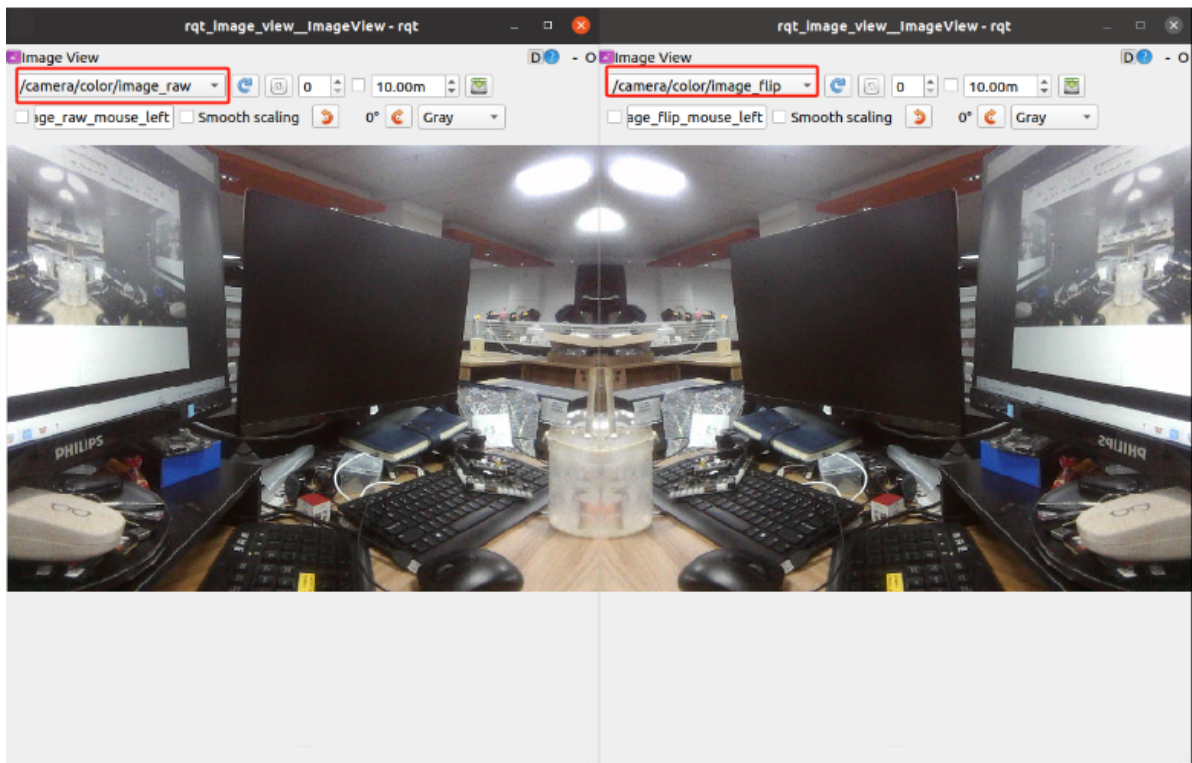
```
//Create a receiver
ros::Subscriber subscriber = n.subscribe<sensor_msgs::Image>
("/camera/depth/image_raw", 10, depth_Callback);
//Create cv_bridge example
cv_bridge::CvImagePtr cv_ptr;
//Data conversion
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::TYPE_16UC1)
```

5. Start color image inversion

```
roslaunch astra_visual astra_image_flip.py # py
```

Image viewing (turn on both), select different topics to display

```
rqt_image_view
```



- code analysis

- 1) Create a subscriber

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

- 2) Create a publisher

The published topic is ["/camera/rgb/image_flip"], the data type [Image], and the 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)
```