

# 1. ROS+OpenCV Basics

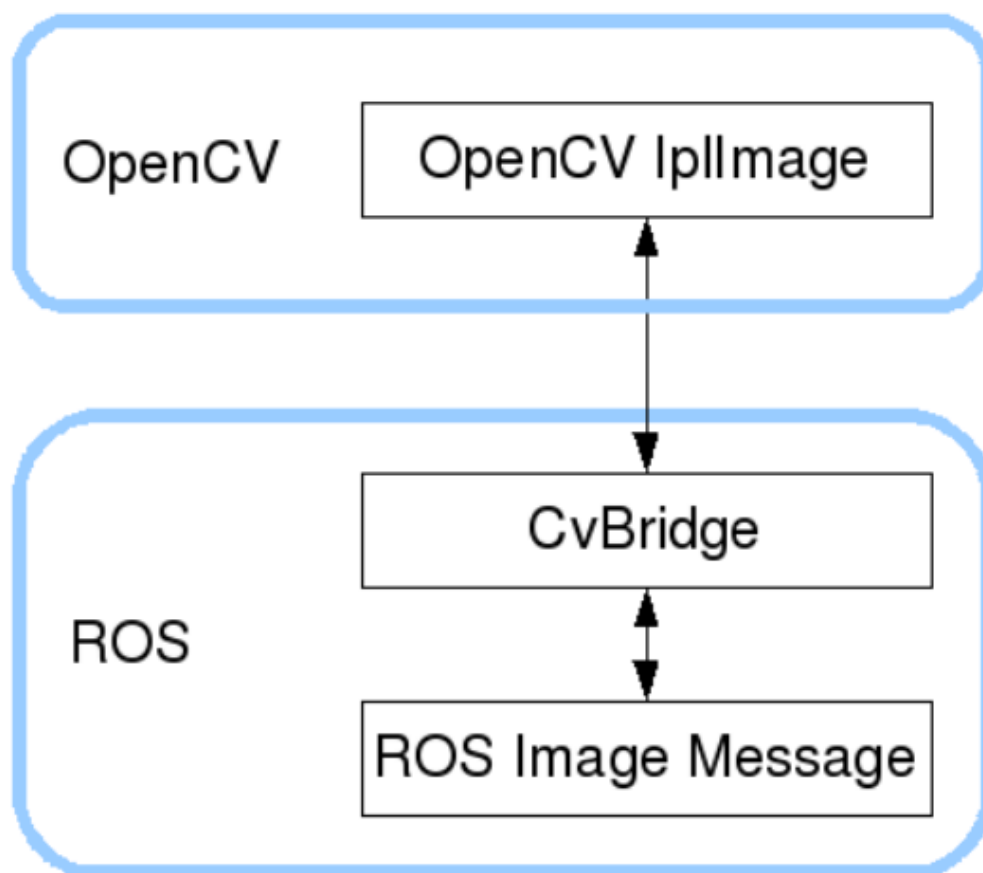
---

Function package location: ~/ascam\_ws/src/ascam\_visual

## 1.1 Overview

ROS has integrated Opencv3.0 or higher versions during the installation process, so there is almost no need to consider the installation configuration. 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.

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



Although the installation configuration does not require too much consideration, the use 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

[cv\_bridge]: image conversion dependency package.

```
<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>
```

- CMakeLists.txt

This file has a lot of configuration content. For specific content, please check the source file. The file location is: ~/ascam\_ws/src/ascam\_visual/CMakeLists.txt

## 1.2. Start the camera

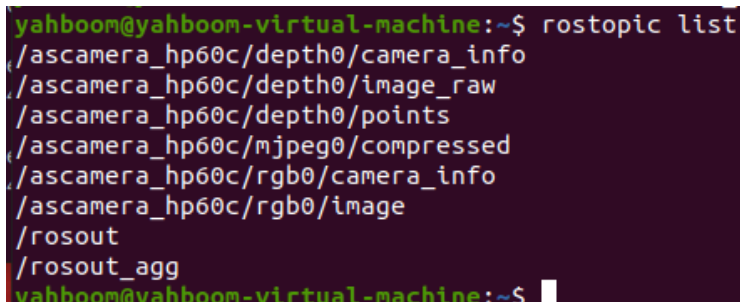
### 1.2.1. Start the camera

Terminal input,

```
roslaunch ascamera hp60c.launch
```

View the topic,

```
rostopic list
```



```
yahboom@yahboom-virtual-machine:~$ rostopic list
/ascamera_hp60c/depth0/camera_info
/ascamera_hp60c/depth0/image_raw
/ascamera_hp60c/depth0/points
/ascamera_hp60c/mjpeg0/compressed
/ascamera_hp60c/rgb0/camera_info
/ascamera_hp60c/rgb0/image
/rosout
/rosout_agg
yahboom@yahboom-virtual-machine:~$
```

The following are commonly used,

/ascamera\_hp60c/rgb0/image: RGB color image topic

/ascamera\_hp60c/depth0/image\_raw: Depth depth image topic

/ascamera\_hp60c/depth0/points: Depth point cloud data topic

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

```
rostopic echo /ascamera_hp60c/rgb0/image/encoding
rostopic echo /ascamera_hp60c/depth0/image_raw/encoding
```

```

yahboom@yahboom-virtual-machine:~$ rostopic echo /ascamera_hp60c/rgb0/image/encoding
"bgr8"
---
"bgr8"
---
"bgr8"
---

yahboom@yahboom-virtual-machine:~$ rostopic echo /ascamera_hp60c/depth0/image_raw/encoding
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"

```

### 1.2.2, Start the color image subscription node

```

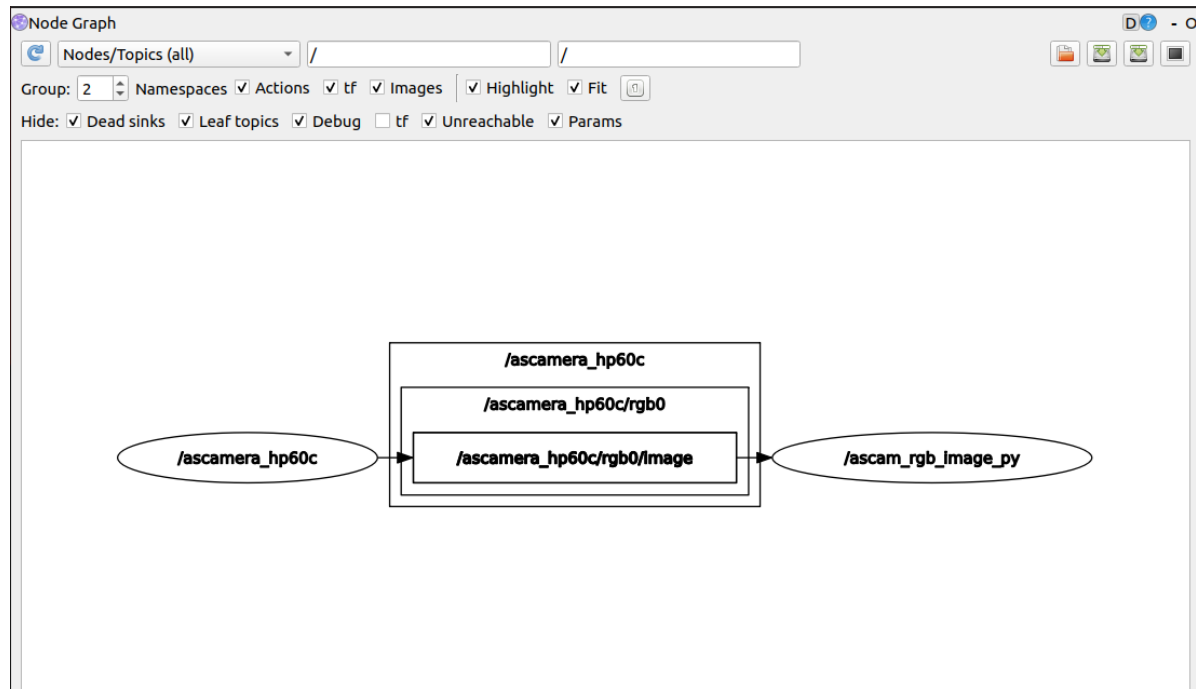
roslaunch ascam_visual ascam_rgb_image.py # py
roslaunch ascam_visual ascam_rgb_image # C++

```



View the node graph,

```
rqt_graph
```



- py code analysis

Create a subscriber: the subscribed topic is `["/ascamera_hp60c/rgb0/image"]`, data type `[Image]`, callback function

`[topic()]`

```
sub = rospy.Subscriber("/ascamera_hp60c/rgb0/image", Image, topic)
```

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

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

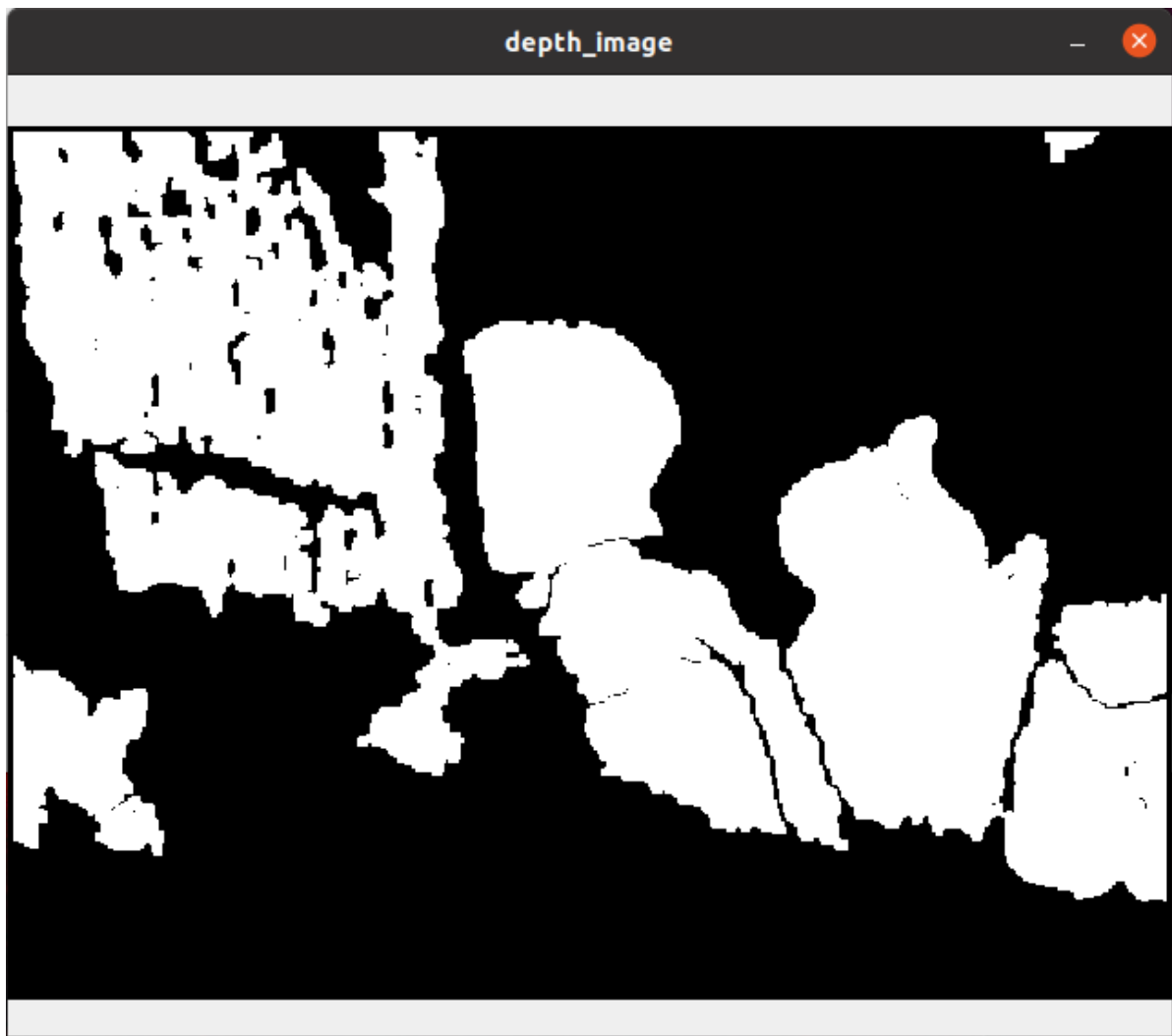
- c++ code analysis

Similar to py code

```
//Create a receiver
ros::Subscriber subscriber =
n.subscribe<sensor_msgs::Image>("/ascamera_hp60c/rgb0/image", 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);
```

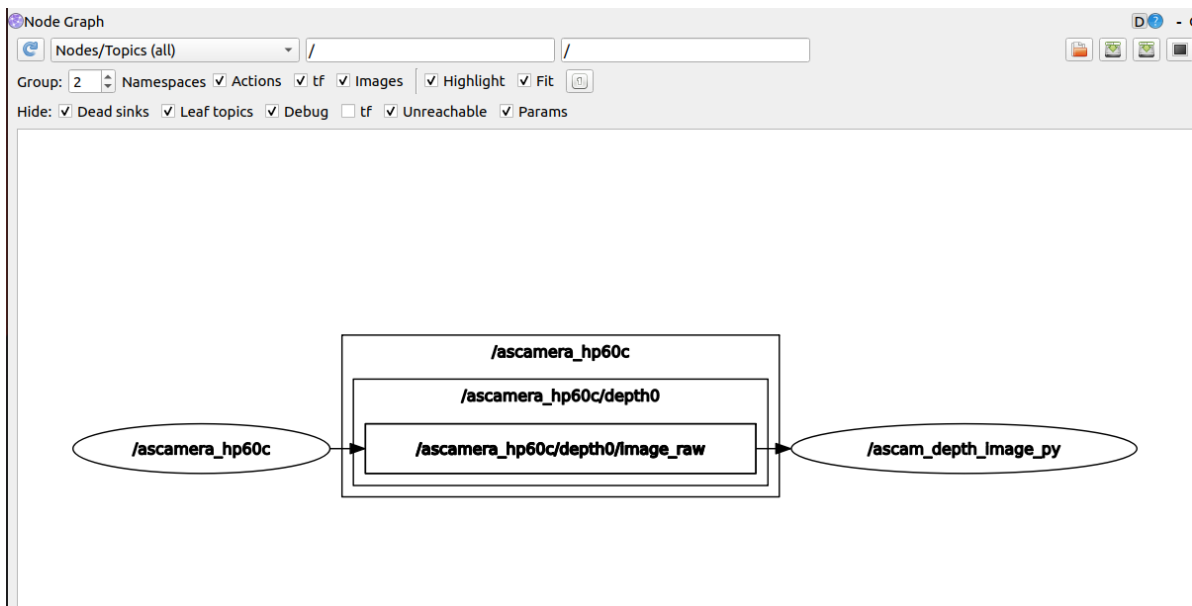
### 1.2.3, Start the depth map subscription node

```
#Start by selecting 1 from the following commands
roslaunch ascam_visual ascam_depth_image.py # py
roslaunch ascam_visual ascam_depth_image # C++
```



View the node graph,

rqt\_graph



- py code analysis

Create a subscriber: the subscribed topic is `["/ascamera_hp60c/depth0/image_raw"]`, data type `[Image]`, callback function `[topic()]`

```
sub = rospy.Subscriber("/ascamera_hp60c/depth0/image_raw", Image, topic)
```

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

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

- c++ code analysis

Similar to py code,

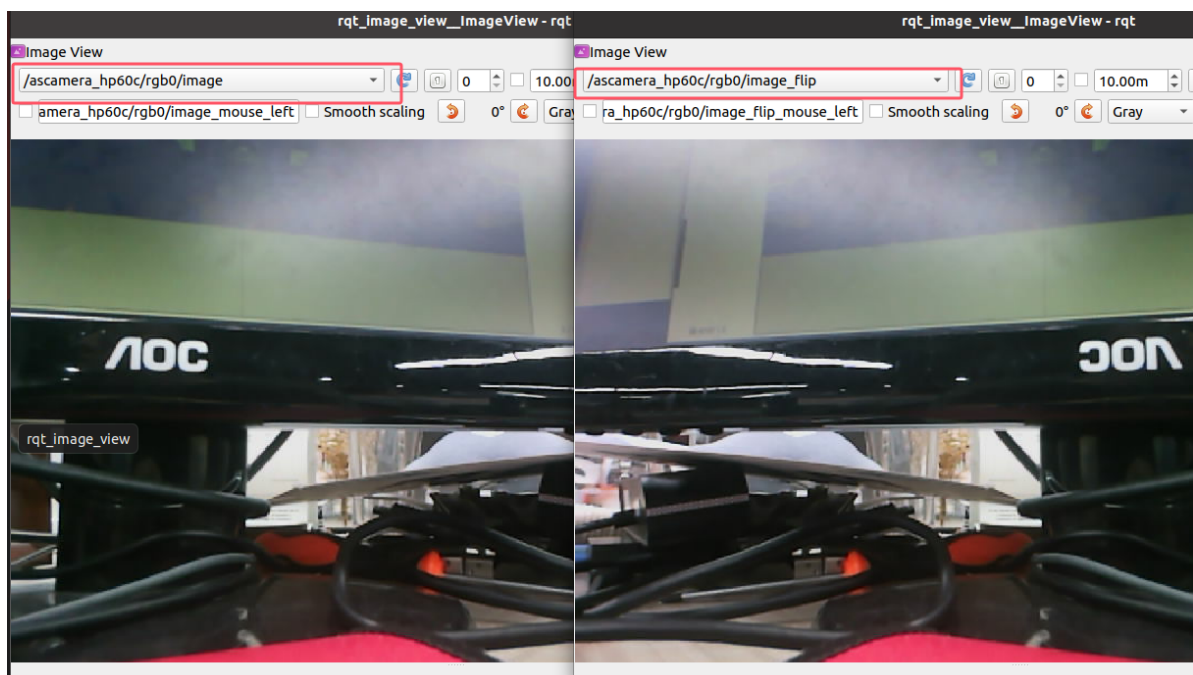
```
//Create a receiver.
ros::Subscriber subscriber = n.subscribe<sensor_msgs::Image>
("/ascamera_hp60c/depth0/image_raw", 10, depth_Callback);
// Create cv_bridge example
cv_bridge::CvImagePtr cv_ptr;
// Data conversion
cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::TYPE_16UC1)
```

## 1.2.4, start color image inversion

```
roslaunch ascam_visual ascam_image_flip.py # py
```

Image view (open two), select different topics to display

```
rqt_image_view
```



- Code analysis

1), create subscribers

The subscribed topic is ["/ascamera\_hp60c/rgb0/image"], data type [Image], callback function [topic()].

2), create publishers

The published topic is ["/ascamera\_hp60c/rgb0/image\_flip"], data type [Image], queue size [10].

3. Callback function

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