8. ROS+Opencv foundation

This lesson takes the Astra camera as an example, which is similar to ordinary cameras.

8.1. Overview

Wiki：http://wiki.ros.org/cv_bridge/

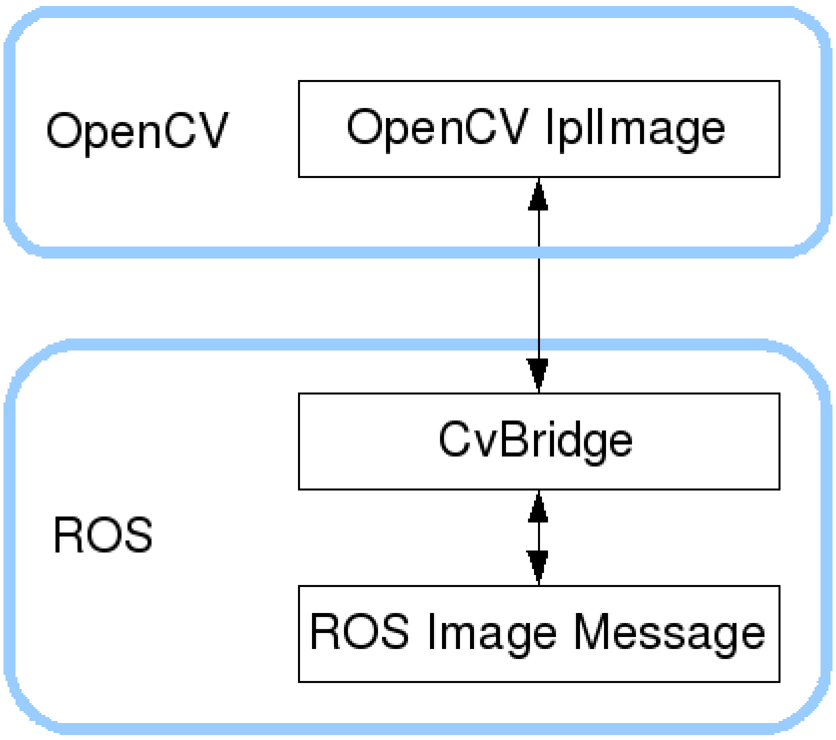
Teaching：http://wiki.ros.org/cv_bridge/Tutorials

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

Feature pack location：~/yahboomcar_ws/src/yahboomcar_visual

ROS has already integrated Opencv3.0 or above during the installation process, so the installation configuration hardly requires too much consideration. 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], it also requires [Opencv] and [PCL], so it is 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

There are many configuration contents in this file. For details, please refer to the source file.

8.2、Astra

8.2.1. Start Astra Camera

```
roslaunch orbbec_camera astraproplus.launch
```

View threads

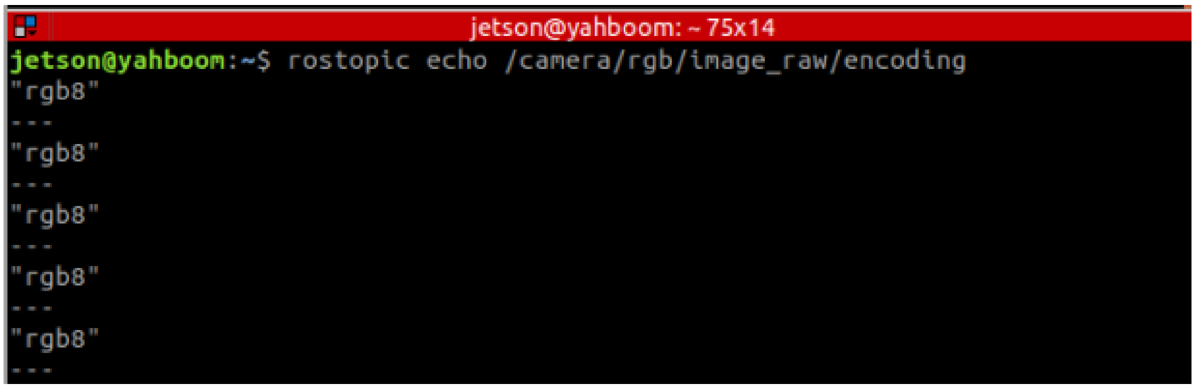
```
rostopic list
```

You can see a lot of topics, just a few commonly used in this section

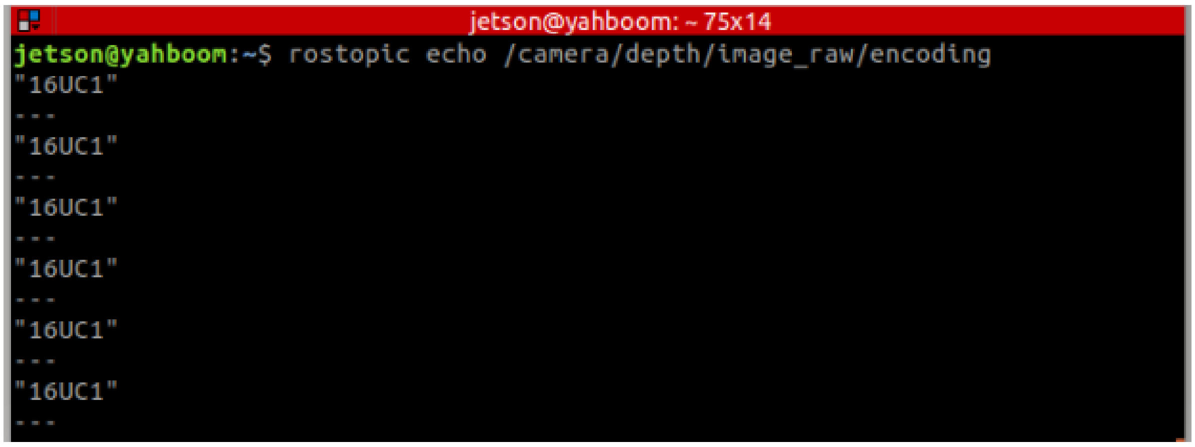
topic name	type of data
/camera/depth/image_raw	sensor_msgs/Image
/camera/rgb/image_raw	sensor_msgs/Image
/camera/ir/image_raw	sensor_msgs/Image

Check the encoding format of the topic: rostopic echo +【topic】+encoding, for example

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



```
jetson@yahboom: ~ 75x14
jetson@yahboom:~$ rostopic echo /camera/rgb/image_raw/encoding
"rgb8"
---
"rgb8"
---
"rgb8"
---
"rgb8"
---
"rgb8"
---
```



```
jetson@yahboom: ~ 75x14
jetson@yahboom:~$ rostopic echo /camera/depth/image_raw/encoding
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
"16UC1"
---
```

8.2.2. Start the color map subscription node

```
roslaunch yahboomcar_visual astra_get_rgb.launch version:=cpp
```

- version parameter: optional [py, cpp] different codes have the same effect.

View node graph

```
rqt_graph
```

When you open the node graph, densely packed nodes and relationships between nodes will appear. At this time, we are using the part linked to the topic [/camera/rgb/image_raw], and [/astra_rgb_Image_cpp] is the node we wrote.

- py code analysis

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

```
sub = rospy.Subscriber("/camera/rgb/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.

```
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>
("/camera/rgb/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);
```

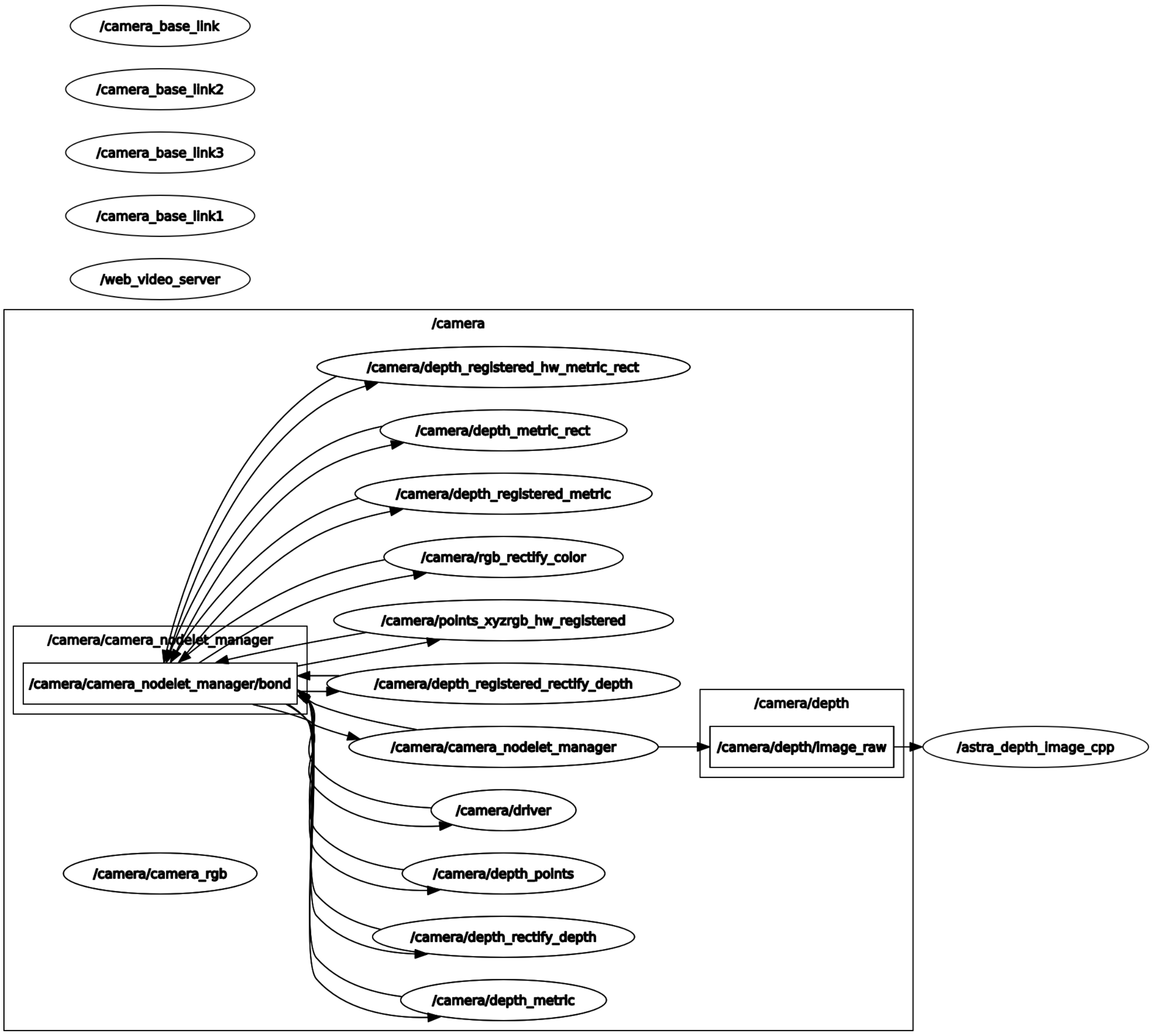
8.2.3. Start the depth graph subscription node

```
roslaunch yahboomcar_visual astra_get_depth.launch version:=cpp
```

View node graph

```
rqt_graph
```

Opening the node graph will show dense nodes and relationships between nodes. At this time, we use the part linked to the [/camera/depth/image_raw] topic. , [/astra_depth_Image_cpp] is the node we wrote.


/camera_base_link
/camera_base_link2
/camera_base_link3
/camera_base_link1
/web_video_server
/camera
/camera/depth_registered_hw_metric_rect
/camera/depth_metric_rect
/camera/depth_registered_metric
/camera/rgb_rectify_color
/camera/points_xyzrgb_hw_registered
/camera/camera_nodelet_manager
/camera/camera_nodelet_manager/bond
/camera/depth_registered_rectify_depth
/camera/camera_nodelet_manager
/camera/depth
/camera/depth/image_raw
/astra_depth_image_cpp
/camera/driver
/camera/camera_rgb
/camera/depth_points
/camera/depth_rectify_depth
/camera/depth_metric

- py code analysis

Create a subscriber: The subscribed topic is ["/camera/depth/image_raw"], the data type is [Image], and the callback function is [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 to 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>
("/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);
```

8.2.4. Start color image inversion

```
roslaunch yahboomcar_visual astra_image_flip.launch
```

image view

```
rqt_image_view
```

- py code analysis

Here the subscriber publisher is created,

1. Create subscribers

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

2. Create publisher

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

```
sub_img = rospy.Subscriber("/camera/rgb/image_raw", Image, topic)
pub_img = rospy.Publisher("/camera/rgb/image_flip", Image, queue_size=10)
```

3. Callback function

```python
# Normal image transfer processing
def topic(msg):
    if not isinstance(msg, Image):
        return
    bridge = CvBridge()
    frame = bridge.imgmsg_to_cv2(msg, "bgr8")
    # Opencv process 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)
```