Note: Diamondback introduces a new C++ cv_bridge API. Make sure you have selected the correct distribution above.

Please ask about problems and questions regarding this tutorial on answers.ros.org (http://answers.ros.org). Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Converting between ROS images and OpenCV images (C++)

Description: This tutorial describes how to interface ROS and OpenCV by converting ROS images into OpenCV images, and vice versa, using cv_bridge. Included is a sample node that can be used as a template for your own node.

Keywords: image, images, OpenCV, cvbridge, CvBridge

Tutorial Level: INTERMEDIATE

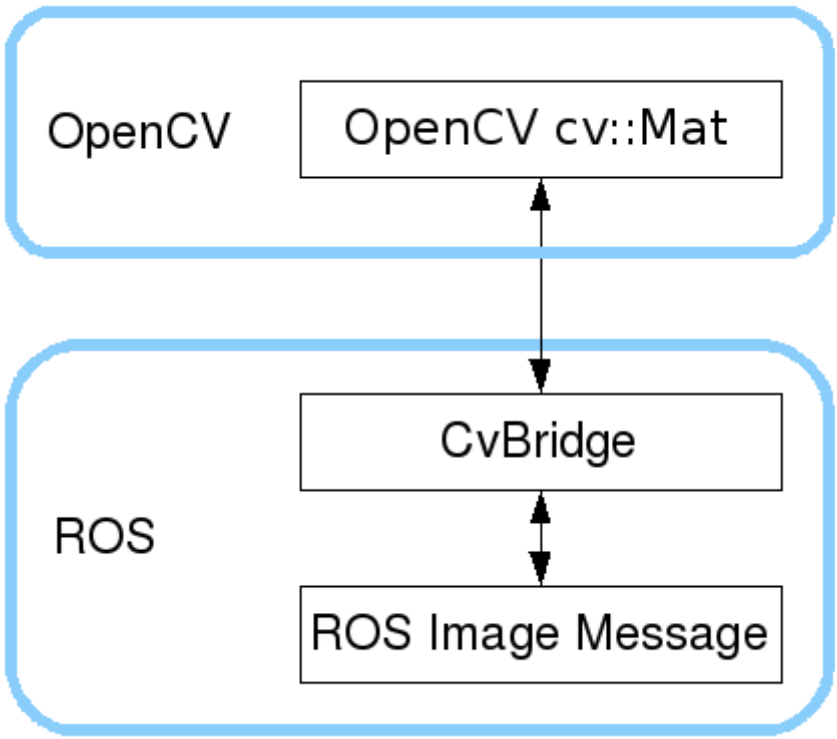
Next Tutorial: Converting between ROS images and OpenCV images (Python) (/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython)

1. Concepts

ROS passes around images in its own sensor_msgs/Image (http://docs.ros.org/api/sensor_msgs/html/msg/Image.html) message format, but many users will want to use images in conjunction with OpenCV. `CvBridge` is a ROS library that provides an interface between ROS and OpenCV. `CvBridge` can be found in the cv_bridge (/cv_bridge) package in the vision_opencv (/vision_opencv) stack.

In this tutorial, you will learn how to write a node that uses `CvBridge` to convert ROS images into OpenCV cv::Mat (http://opencv.willowgarage.com/documentation/cpp/core_basic_structures.html#mat) format.

You will also learn how to convert OpenCV images to ROS format to be published over ROS.



1.1 Migration from codes written in C-Turtle or earlier

There was a major api change ROS Diamondback (/diamondback) regarding `OpenCV`, of which backward compatibility has been maintained for a while but removed in newer distro (eg. hydro (/hydro)), e.g. `sensor_msgs/CvBridge`. See the design decision (/cv_bridge_redesign). Also this QA (http://answers.ros.org/question/9765/how-to-convert-cvmat-to-sensor_msgsimageptr/?answer=14282#post-id-14282) is helpful.

2. Converting ROS image messages to OpenCV images

`CvBridge` defines a `CvImage` type containing an OpenCV image, its encoding and a ROS header. `CvImage` contains exactly the information sensor_msgs/Image (http://docs.ros.org/api/sensor_msgs/html/msg/Image.html) does, so we can convert either representation to the other. CvImage (/CvImage) class format:

Toggle line numbers

```
 1 namespace cv_bridge {
 2
 3 class CvImage
 4 {
 5 public:
 6   std_msgs::Header header;
 7   std::string encoding;
 8   cv::Mat image;
 9 };
10
11 typedef boost::shared_ptr<CvImage> CvImagePtr;
12 typedef boost::shared_ptr<CvImage const> CvImageConstPtr;
13
14 }
```

When converting a ROS sensor_msgs/Image (http://docs.ros.org/api/sensor_msgs/html/msg/Image.html) message into a `CvImage`, `CvBridge` recognizes two distinct use cases:

1. We want to modify the data in-place. We have to make a copy of the ROS message data.
2. We won't modify the data. We can safely share the data owned by the ROS message instead of copying.

`CvBridge` provides the following functions for converting to `CvImage`:

Toggle line numbers

```
 1 // Case 1: Always copy, returning a mutable CvImage
 2 CvImagePtr toCvCopy(const sensor_msgs::ImageConstPtr& source,
 3                     const std::string& encoding = std::string());
 4 CvImagePtr toCvCopy(const sensor_msgs::Image& source,
 5                     const std::string& encoding = std::string());
 6
 7 // Case 2: Share if possible, returning a const CvImage
 8 CvImageConstPtr toCvShare(const sensor_msgs::ImageConstPtr& source,
 9                           const std::string& encoding = std::string());
10 CvImageConstPtr toCvShare(const sensor_msgs::Image& source,
11                           const boost::shared_ptr<void const>& tracked_ob
ject,
12                           const std::string& encoding = std::string());
```

The input is the image message pointer, as well as an optional `encoding` argument. The encoding refers to the destination `CvImage`.

`toCvCopy` creates a copy of the image data from the ROS message, even when the source and destination encodings match. However, you are free to modify the returned `CvImage`.

`toCvShare` will point the returned `cv::Mat` at the ROS message data, avoiding a copy, if the source and destination encodings match. As long as you hold a copy of the returned `CvImage`, the ROS message data will not be freed. If the encodings do not match, it will allocate a new buffer and perform the conversion. You are not permitted to modify the returned `CvImage`, as it may share data with the

ROS image message, which in turn may be shared with other callbacks. Note: the second overload of `toCvShare` is more convenient when you have a pointer to some other message type (e.g. stereo_msgs/DisparityImage (http://docs.ros.org/api/stereo_msgs/html/msg/DisparityImage.html)) that contains a `sensor_msgs/Image` you want to convert.

If no encoding (or rather, the empty string) is given, the destination image encoding will be the same as the image message encoding. In this case `toCvShare` is guaranteed to not copy the image data. Image encodings can be any one of the following OpenCV image encodings:

- 8UC[1-4]
- 8SC[1-4]
- 16UC[1-4]
- 16SC[1-4]
- 32SC[1-4]
- 32FC[1-4]
- 64FC[1-4]

For popular image encodings, `CvBridge` will optionally **do color or pixel depth conversions as necessary**. To use this feature, specify the encoding to be one of the following strings:

- `mono8`: `CV_8UC1`, grayscale image
- `mono16`: `CV_16UC1`, 16-bit grayscale image
- `bgr8`: `CV_8UC3`, color image with blue-green-red color order
- `rgb8`: `CV_8UC3`, color image with red-green-blue color order
- `bgra8`: `CV_8UC4`, BGR color image with an alpha channel
- `rgba8`: `CV_8UC4`, RGB color image with an alpha channel

Note that `mono8` and `bgr8` are the two image encodings expected by most OpenCV functions.

Finally, `CvBridge` will recognize Bayer pattern encodings as having OpenCV type `8UC1` (8-bit unsigned, one channel). It will not perform conversions to or from Bayer pattern; in a typical ROS system, this is done instead by image_proc (/image_proc). `CvBridge` recognizes the following Bayer encodings:

- `bayer_rggb8`
- `bayer_bggr8`
- `bayer_gbrg8`
- `bayer_grbg8`

3. Converting OpenCV images to ROS image messages

To convert a `CvImage` into a ROS image message, use one the `toImageMsg()` member function:

Toggle line numbers

```
1 class CvImage
2 {
3   sensor_msgs::ImagePtr toImageMsg() const;
4
5   // Overload mainly intended for aggregate messages that contain
6   // a sensor_msgs::Image as a member.
7   void toImageMsg(sensor_msgs::Image& ros_image) const;
8 };
```

If the `CvImage` is one you have allocated yourself, don't forget to fill in the `header` and `encoding` fields.

For an example of allocating one yourself please see the Publishing Images tutorial (/image_transport/Tutorials/PublishingImages).

4. An example ROS node

Here is a node that listens to a ROS image message topic, converts the image into a `cv::Mat`, draws a circle on it and displays the image using OpenCV. The image is then republished over ROS.

In your package.xml and CMakeLists.xml (or when you use catkin_create_pkg (/catkin)), add the following dependencies:

```
sensor_msgs
cv_bridge
roscpp
std_msgs
image_transport
```

Create a image_converter.cpp file in your /src folder and add the following:

Toggle line numbers

```
 1 #include <ros/ros.h>
 2 #include <image_transport/image_transport.h>
 3 #include <cv_bridge/cv_bridge.h>
 4 #include <sensor_msgs/image_encodings.h>
 5 #include <opencv2/imgproc/imgproc.hpp>
 6 #include <opencv2/highgui/highgui.hpp>
 7
 8 static const std::string OPENCV_WINDOW = "Image window";
 9
10 class ImageConverter
11 {
12   ros::NodeHandle nh_;
13   image_transport::ImageTransport it_;
14   image_transport::Subscriber image_sub_;
15   image_transport::Publisher image_pub_;
16
17 public:
18   ImageConverter()
19     : it_(nh_)
20   {
21     // Subscrive to input video feed and publish output video feed
22     image_sub_ = it_.subscribe("/camera/image_raw", 1,
23       &ImageConverter::imageCb, this);
24     image_pub_ = it_.advertise("/image_converter/output_video", 1);
25
26     cv::namedWindow(OPENCV_WINDOW);
27   }
28
29   ~ImageConverter()
30   {
31     cv::destroyWindow(OPENCV_WINDOW);
32   }
33
34   void imageCb(const sensor_msgs::ImageConstPtr& msg)
35   {
36     cv_bridge::CvImagePtr cv_ptr;
37     try
38     {
39       cv_ptr = cv_bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR
8);
40     }
41     catch (cv_bridge::Exception& e)
42     {
43       ROS_ERROR("cv_bridge exception: %s", e.what());
44       return;
45     }
46
47     // Draw an example circle on the video stream
48     if (cv_ptr->image.rows > 60 && cv_ptr->image.cols > 60)
49       cv::circle(cv_ptr->image, cv::Point(50, 50), 10, CV_RGB(255,0,0));
```

```
50
51     // Update GUI Window
52     cv::imshow(OPENCV_WINDOW, cv_ptr->image);
53     cv::waitKey(3);
54
55     // Output modified video stream
56     image_pub_.publish(cv_ptr->toImageMsg());
57   }
58 };
59
60 int main(int argc, char** argv)
61 {
62   ros::init(argc, argv, "image_converter");
63   ImageConverter ic;
64   ros::spin();
65   return 0;
66 }
```

Let's break down the above node:

Error: No code_block found Using image_transport (/image_transport) for publishing and subscribing to images in ROS allows you to subscribe to compressed image streams. Remember to include `image_transport` in your package.xml.

Error: No code_block found Includes the header for `CvBridge` as well as some useful constants and functions related to image encodings (http://www.ros.org/doc/api/sensor_msgs/html/namespacesensor__msgs_1_1image__encodings.html). Remember to include `cv_bridge` in your package.xml.

Error: No code_block found Includes the headers for OpenCV's image processing and GUI modules. Remember to include `opencv2` in your package.xml.

Error: No code_block found Subscribe to an image topic "in" and advertise an image topic "out" using image_transport (/image_transport).

Error: No code_block found OpenCV HighGUI (http://opencv.willowgarage.com/documentation/cpp/highgui__high-level_gui_and_media_i_o.html) calls to create/destroy a display window on start-up/shutdown.

Error: No code_block found In our subscriber callback, we first convert the ROS image message to a `CvImage` suitable for working with OpenCV. Since we're going to draw on the image, we need a mutable copy of it, so we use `toCvCopy()`. `sensor_msgs::image_encodings::BGR8` is simply a constant for "bgr8", but less susceptible to typos.

Note that OpenCV expects color images to use BGR channel order.

You should always wrap your calls to `toCvCopy()` / `toCvShared()` to catch conversion errors as those functions will not check for the validity of your data.

Error: No code_block found Draw a red circle on the image, then show it in the display window.

Error: No code_block found Convert the `CvImage` to a ROS image message and publish it on the "out" topic.

To run the node, you will need an image stream. Run a camera or play a bag file to generate the image stream. Now you can run this node, remapping (/Remapping%20Arguments) "in" to the actual image stream topic.

If you have successfully converted images to OpenCV format, you will see a HighGui window with the name "Image window" and your image+circle displayed.

You can see whether your node is correctly publishing images over ROS using either rostopic (/rostopic) or by viewing the images using image_view (/image_view).

5. Examples of sharing the image data

In the complete example above, we explicitly copied the image data, but sharing (when possible) is equally easy:

Toggle line numbers

```
1 namespace enc = sensor_msgs::image_encodings;
2
3 void imageCb(const sensor_msgs::ImageConstPtr& msg)
4 {
5   cv_bridge::CvImageConstPtr cv_ptr;
6   try
7   {
8     cv_ptr = cv_bridge::toCvShare(msg, enc::BGR8);
9   }
10  catch (cv_bridge::Exception& e)
11  {
12    ROS_ERROR("cv_bridge exception: %s", e.what());
13    return;
14  }
15
16  // Process cv_ptr->image using OpenCV
17 }
```

If the incoming message has "bgr8" encoding, `cv_ptr` will alias its data without making a copy. If it has a different but convertible encoding, say "mono8", `CvBridge` will allocate a new buffer for `cv_ptr` and perform the conversion. Without the exception handling this would only be one line of code, but then an incoming message with a malformed (or unsupported) encoding would bring down the node. For example, if the incoming image is from the `image_raw` topic for a Bayer pattern camera, `CvBridge` will throw an exception because it (intentionally) does not support automatic Bayer-to-color conversion.

A slightly more complicated example:

Toggle line numbers

```
 1 namespace enc = sensor_msgs::image_encodings;
 2
 3 void imageCb(const sensor_msgs::ImageConstPtr& msg)
 4 {
 5   cv_bridge::CvImageConstPtr cv_ptr;
 6   try
 7   {
 8     if (enc::isColor(msg->encoding))
 9       cv_ptr = cv_bridge::toCvShare(msg, enc::BGR8);
10     else
11       cv_ptr = cv_bridge::toCvShare(msg, enc::MONO8);
12   }
13   catch (cv_bridge::Exception& e)
14   {
15     ROS_ERROR("cv_bridge exception: %s", e.what());
16     return;
17   }
18
19   // Process cv_ptr->image using OpenCV
20 }
```

In this case we want to use color if available, otherwise falling back to monochrome. If the incoming image is either "bgr8" or "mono8", we avoid copying data.

Wiki: cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages (last edited 2017-04-20 22:38:09 by AdamAllevato (/AdamAllevato))