Note: This tutorial assumes that you have completed the previous tutorials: ROS tutorials (/ROS/Tutorials), Using CvBridge to Convert Between ROS Images and OpenCV Images (/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages).

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.

Writing a Simple Image Publisher (C++)

Description: This tutorial shows how to publish images using all available transports.

Tutorial Level: BEGINNER

Next Tutorial: Writing a Simple Image Subscriber (/image_transport/Tutorials/SubscribingToImages)

Contents

1. Writing a Simple Image Publisher
    1. The Code
    2. The Code Explained
    3. Adding video stream from a webcam
2. Building your node

catkin | rosbuild

0.1 Writing a Simple Image Publisher

Here we'll create the publisher node which will continually publish an image.

Change to the directory you've created for these tutorials:

(Assuming you have created your package in ~/image_transport_ws)

```
$ cd ~/image_transport_ws/
$ git clone https://github.com/ros-perception/image_common.git
$ mkdir src
$ ln -s `pwd`/image_common/image_transport/tutorial/ ./src/image_transport_tutor
ial
```

0.0.1 The Code

Have a look at `my_publisher.cpp`:

Toggle line numbers

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <opencv2/highgui/highgui.hpp>
#include <cv_bridge/cv_bridge.h>

int main(int argc, char** argv)
{
  ros::init(argc, argv, "image_publisher");
  ros::NodeHandle nh;
  image_transport::ImageTransport it(nh);
  image_transport::Publisher pub = it.advertise("camera/image", 1);
  cv::Mat image = cv::imread(argv[1], CV_LOAD_IMAGE_COLOR);
  cv::waitKey(30);
  sensor_msgs::ImagePtr msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", image).toImageMsg();

  ros::Rate loop_rate(5);
  while (nh.ok()) {
    pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
  }
}
```

0.0.2 The Code Explained

Now, let's break down the code piece by piece. For lines not explained here, review Writing a Simple Publisher and Subscriber (C++) (http://www.ros.org/wiki/ROS/Tutorials/WritingPublisherSubscriber(c++)).

Error: No code_block found `image_transport/image_transport.h` includes everything we need to publish and subscribe to images.

Error: No code_block found These headers will allow us to load an image using OpenCV and convert it to the ROS message format.

Error: No code_block found We create an `ImageTransport` instance, initializing it with our `NodeHandle`. We use methods of `ImageTransport` to create image publishers and subscribers, much as we use methods of `NodeHandle` to create generic ROS publishers and subscribers.

Error: No code_block found Advertise that we are going to be publishing images on the base topic "camera/image". Depending on whether more plugins are built, additional (per-plugin) topics derived from the base topic may also be advertised. The second argument is the size of our publishing queue.

`advertise()` returns an `image_transport::Publisher` object, which serves two purposes: 1) it contains a `publish()` method that lets you publish images onto the base topic it was created with, and 2) when it goes out of scope, it will automatically unadvertise.

Error: No code_block found We load a user-specified (on the command line) color image from disk using OpenCV, then convert it to the ROS type sensor_msgs/Image (http://docs.ros.org/api/sensor_msgs/html/msg/Image.html). See this tutorial

(/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages) for more on ROS-OpenCV image conversion.

Error: No code_block found We broadcast the image to anyone connected to one of our topics, exactly as we would have using a `ros::Publisher`.

0.0.3 Adding video stream from a webcam

The example above requires a path of an image file to be added as a command line parameter (`cv::imread(argv[1], CV_LOAD_IMAGE_COLOR)`). This image will be converted and send as a message to an image subscriber. In most cases however this is not a very practical example since often you are required to handle streaming data (for example: multiple webcameras mounted on a robot record the scene around it and you have to pass the image date to some other node for further analysis). You can modify the example quite easily to make it work with a video device supported by `cv::VideoCapture` (in case it is not you have to handle it accordingly):

Toggle line numbers

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <opencv2/highgui/highgui.hpp>
#include <cv_bridge/cv_bridge.h>
#include <sstream> // for converting the command line parameter to integer

int main(int argc, char** argv)
{
  // Check if video source has been passed as a parameter
  if(argv[1] == NULL) return 1;

  ros::init(argc, argv, "image_publisher");
  ros::NodeHandle nh;
  image_transport::ImageTransport it(nh);
  image_transport::Publisher pub = it.advertise("camera/image", 1);

  // Convert the passed as command line parameter index for the video device to an integer
  std::istringstream video_sourceCmd(argv[1]);
  int video_source;
  // Check if it is indeed a number
  if(!(video_sourceCmd >> video_source)) return 1;

  cv::VideoCapture cap(video_source);
  // Check if video device can be opened with the given index
  if(!cap.isOpened()) return 1;
  cv::Mat frame;
  sensor_msgs::ImagePtr msg;

  ros::Rate loop_rate(5);
  while (nh.ok()) {
    cap >> frame;
    // Check if grabbed frame is actually full with some content
    if(!frame.empty()) {
      msg = cv_bridge::CvImage(std_msgs::Header(), "bgr8", frame).toImageMsg();
      pub.publish(msg);
      cv::waitKey(1);
    }

    ros::spinOnce();
    loop_rate.sleep();
  }
}
```

If you have a single device you do not need to do the whole routine with passing a command line argument (`argv[1]`) and parsing it at all. In this case you can hard-code the index/address of the device and directly pass it to the video capturing structure in OpenCV (example: `cv::VideoCapture(0)` if /dev/video0 is used). In addition multiple checks are also included here to make sure that the publisher does not brake if in case the camera is shut downs or similar issue). If the retrieved frame from the video device is not empty it will be then converted to a ROS message, which will be published by the publisher.

0.1 Building your node

Just run:

```
catkin_make
```

Now let's write a simple image subscriber (/image_transport/Tutorials/SubscribingToImages).