# 机器人感知

机器人感知是指机器人使用各类传感器来感知周围的环境。



**相机**

相机相当于机器人的眼睛。从相机获得的图像对于识别机器人周围的环境非常有用。例如，利用相机图像的对象识别和脸部识别；使用两台相机（立体相机）从两个不同图像之间的差异获得的距离值；利用距离值生成3维地图的Visual-SLAM；单眼相机VisualSLAM；利用从彩色图像获得的颜色信息的颜色识别；跟踪特定对象的对象跟踪。

相机种类非常多，本节中将用USB摄像头来进行说明。USB摄像头意味着它是支持USB的视频录制设备。另一个名称是USB video device class（UVC）。所以，官方的名字是“UVC相机”，但本节中将其称为常用的“USB摄像头”。

相机的接口并不只有USB。某些相机具有可连接到网络的功能。通常连接到局域网或WiFi，将视频数据以视频流形式传输到网络。这些相机应该被称为网络摄像头。此外，有些摄像机使用FireWire(IEEE1394接口）进行高速传输，主要用于需要高速传输图像的研究目的。FireWire标准在大多数常见的电路板上无法找到，但它是由苹果公司开发的，因此主要用于苹果产品。

**深度相机**（Microsoft的Kinect、Intel的RealSense等）



**激光雷达**



## ROS中使用USB摄像头

USB摄像头相关功能包

USB摄像头是目前使用非常普遍的一类摄像头，如笔记本内置的摄像头等。在ROS中使用这类设备非常容易，ROS为我们提供了与USB摄像头相关的各种功能包。有关更多信息，请参阅ROS Wiki的“传感器/摄像机”类别（http://wiki.ros.org/Sensors/Cameras）。在此让我们看看几种功能包。

■ **libuvc-camera** 这是用于采用UVC标准的相机的接口功能包。（开发者：Ken Tossell）  
■ **uvc-camera** 因为有相对详细的相机设置功能，所以非常方便。此外，如果有两个相机，所以考虑使用立体相机，那么这将是一个比较合适的功能包。  
■ **usb-cam 这是使用非常简单的摄像头驱动程序**。（开发者：Benjamin Pitzer）  
■ freenect-camera, openni-camera, openni2-camera 所有这三个功能包名称中都有相机，但它们都是深度相机（如Kinect或Xtion）的功能包。这些传感器也被称为RGB-D相机，因为它们也包含彩色相机。如果要利用彩色图像，则需要使用这些功能包。  
■ camera1394 它是使用FireWire（IEEE 1394接口）的相机的驱动程序。  
■ prosilica-camera 它被用于AVS的prosilica相机，它被广泛用于研究目的。  
■ pointgrey-camera-driver 它是Point Grey Research公司的Point Gray相机的一个驱动程序，被广泛用于科研。  
■ **camera-calibration** James Bowman和Patrick Mihelich开发了一个应用了OpenCV的校准功能的相机校准功能包。许多相机相关的功能包需要这个功能包。

可以直接使用usb\_cam功能包驱动。usb\_cam功能包是针对V4L协议USB摄像头的ROS驱动包，。

1、安装usb\_cam功能包

$sudo apt install ros-noetic-usb-cam

$sudo apt install **ros-noetic-image-view /rqt\_image\_view /rviz**

安装路径为：

/opt/ros/noetic/share/usb\_cam

/opt/ros/noetic/share/image\_view

**核心节点是usb\_cam\_node节点。**

2、连接USB摄像头到PC端的USB口，通过以下命令启动摄像头：

$roslaunch usb\_cam usb\_cam-test.launch

或者

$rosrun usb\_cam usb\_cam\_node

$**rqt\_image\_vew**

使用rostopic list命令查看发布的话题消息：

$rostopic list

$ /usb\_cam/image\_raw

$rostopic info /usb\_cam/image\_raw

$sensor\_msgs/Image

3、使用usb\_cam和**image\_view**

$roslaunch usb\_cam usb\_cam-test.launch

或者

$rosrun usb\_cam usb\_cam\_node

查看图像信息

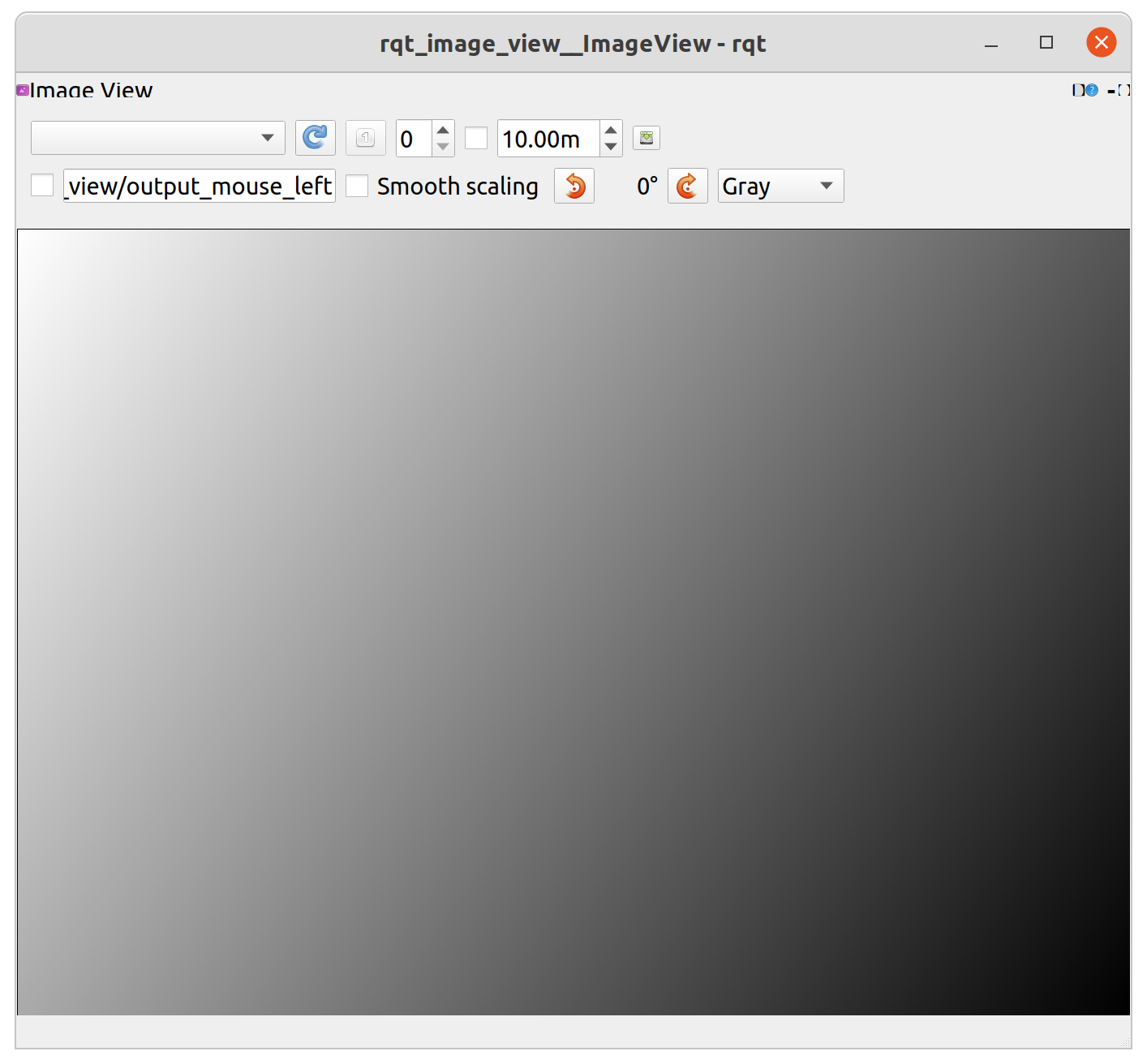
在ROS中可以使用三种方式来查看图像信息：

1、

$rosrun image\_view image\_view **image:=/usb\_cam/image\_raw**

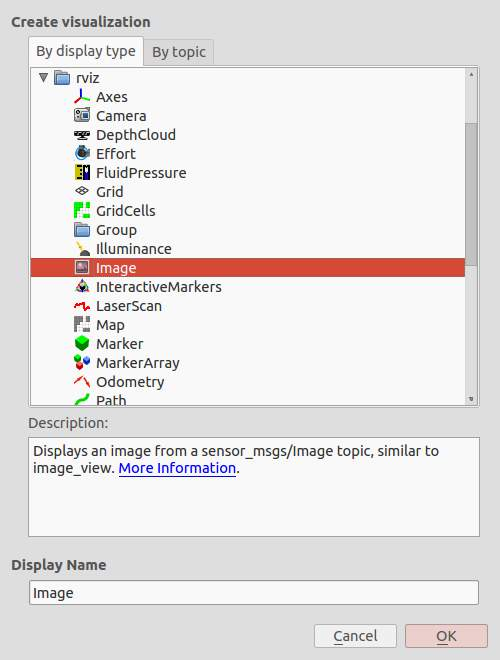
2、ROS还为我们提供了rqt的插件用于图像显示，可以通过如下命令行语句运行：

$ rqt\_image\_view



3、使用rviz来查看图像信息

RViz运行后，先更改“Displays”选项。单击RViz左下方的[Add]，在[By display type]选项卡中选择[Image]，以此加载图像显示功能。然后将[Image] → [Image Topic]的值设置为响应的话题消息。



## 摄像头标定

摄像头为什么要标定？

摄像头这种精密仪器对光学器件的要求较高，生成的物体图像往往会发生畸变，为避免数据源造成的误差，需要对摄像头的参数进行标定。

在进行摄像头标定之前，先安装标定功能包：

$sudo apt install ros-noetic-camera-calibartion

具体的标定过程：

1、启动roscore

2、启动摄像头节点 rosrun usb\_cam usb\_cam\_node

3、启动标定节点

rosrun camera\_calibration cameracalibrator.py --size 8x6 --square 0.024 image:=/usb\_cam/image\_raw **camera:=/usb\_cam**

参数解释：

size：标定棋盘格的内部角点个数，共有6行，每行有8个内部角点；

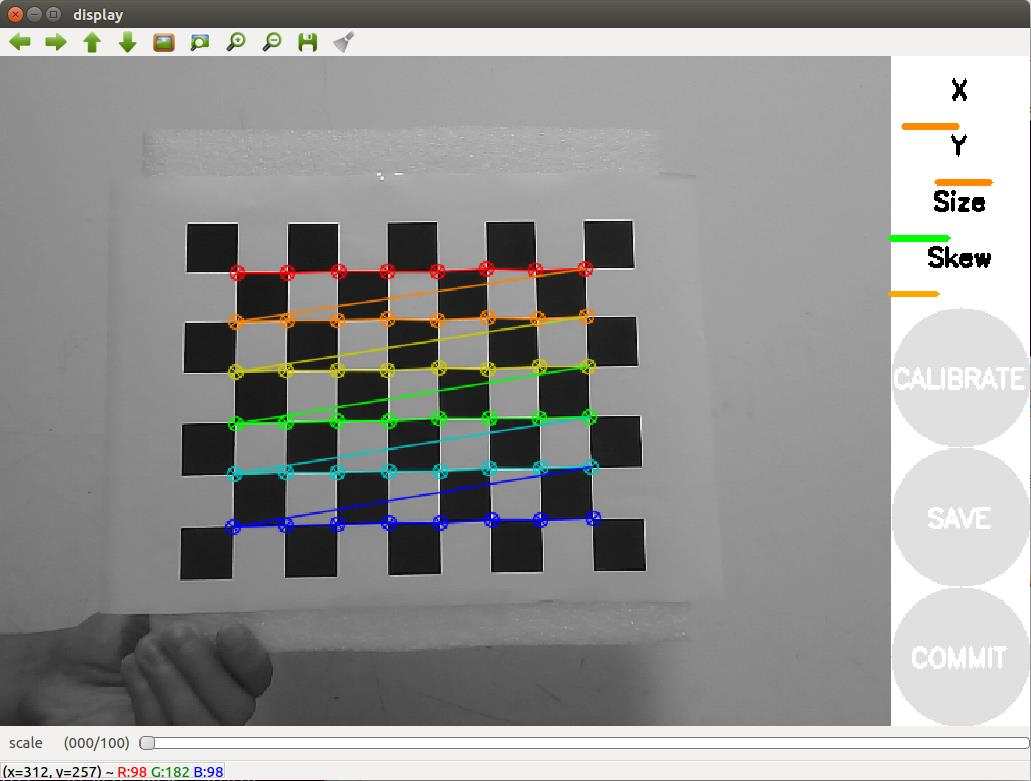
square：对应每个棋盘格的边长，单位是：米

image和camera：设置摄像头发布的图像话题

摄像机的校准是以一个由黑白方块组成的棋盘为基准进行的，如下图所示。可以从本文的附件中下载8×6国际象棋棋盘，并打印出来后将其贴到一个平坦的纸板上。有时也会打印成超过1米的棋盘，但这里用的是A4纸。作为参考，8×6棋盘横向有9个方块，所以有8个交叉点，而竖向有7个方块，有6个交叉点，所以它被称为8×6棋盘。



4、将标定靶放入到摄像头的视野中，并且左右、上下、前后移动标准靶。



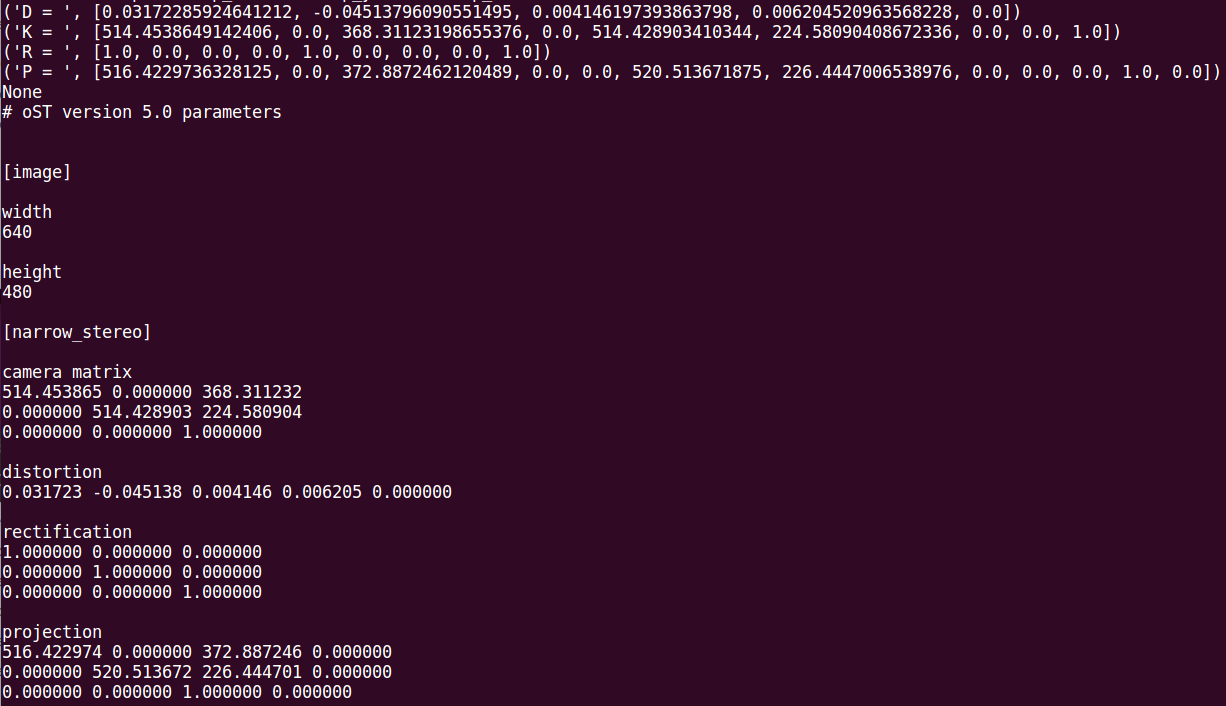
X：标定靶在摄像头视野中的左右移动；

Y：标定靶在摄像头视野中上下移动；

Size：标定靶在摄像头视野中的前后移动；

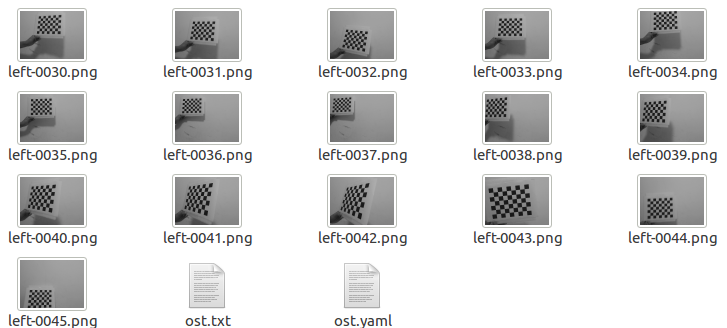
Skew：标定靶在摄像头视野中倾斜转动。

5、标定完成后——“CALIBRAT”变成绿色，可以点击CALIBRATE按钮，持续一段时间后，标定过程计算完成。



标定完成后，“SAVE”按钮和“COMMIT“按钮编程绿色。

点击“SAVE“按钮后，将数据保存至 **/tmp/calibrationdata.tar.gz**文件中，解压缩文件后，生成一下图像文件，ost.yaml文件，将ost.yaml复制到功能包下面。



6、使用标定文件

在launch文件中加载标定文件

<param name="camera\_info\_url" type="string" value=file://$(find 功能包名/camera\_calibration.yaml/>

## 在ROS中使用OpenCV

OpenCV是一个开源的计算机视觉函数库。它是由Intel公司在1999年发起的，已经成为一个全球流行的计算机视觉编程资源，可以运行在Linux、Windows和macOS等操作系统上。OpenCV由一系列的C函数和C++类构成，提供了C++、Python、MATALB等编程语言接口。同时，OpenCV可以直接访问硬件摄像头，并且还提供了一个简单GUI系统——highgui。

为了利用OpenCV的功能，ROS提供了ROS格式的消息和OpenCV格式的图像之间转换的桥梁。

除了能够适应OpenCV类，ROS还为处理图像定制了**发布和订阅**功能。ROS的image\_transport框架包括**管理图像传输的类和节点**。由于图像可能让image\_transport通信带宽不堪重负，所以尽可能降低网络负担就变得非常重要了。

首先，如果一个image\_transport发布的“图像”话题消息没有订阅者，这个话题上的图像将不会被发布；其次，image\_trnasport发布和订阅可以自动执行编码和解码以降低网络带宽消耗。

1、如何在ROS中使用OpenCV

2、如何在ROS中将发布图像消息

在安装ROS的noetic版本时，已经为我们安装OpenCV 4，在ROS的节点中可以直接调用OpenCV 4的API函数进行图像处理。

OpenCV 4 的头文件默认安装目录：/usr/include/opencv4。

OpenCV 4的库文件默认安装目录：/usr/lib/x86\_64-linux-gnu/。

如果沒有安裝OpenCV，則可以使用如下命令安裝：

$sudo apt install **ros-noetic-vision-opencv** **libopencv-dev** **python-opencv**

ROS中的消息类型Image

在OpenCV中使用cv::Mat格式来存储图像。

在ROS中的sensor\_msgs中提供了一种Image消息类型：

$rosmsg info **sensor\_msgs**/Image

std\_msgs/Header header

uint32 seq

time stamp

string frame\_id

uint32 height //图像的高度

uint32 width //图像的宽度

string encoding //图像的编码格式，包含RGB、YUV等常用格式

uint8 is\_bigendian //图像数据的大小端存储模式

uint32 step //一行图像数据的字节数量，作为数据的步长参数

uint8[] data //存储图像数据的数组，大小为step×height个字节

对于1080×720分辨率的摄像头产生一副图像的数据大小是：3×1080×720 = 2764800字节，即2.7648MB.

提供了另外一种压缩图像消息类型：

rosmsg show sensor\_msgs/**CompressedImage**

std\_msgs/Header header

uint32 seq;

time stamp;

string frame\_id

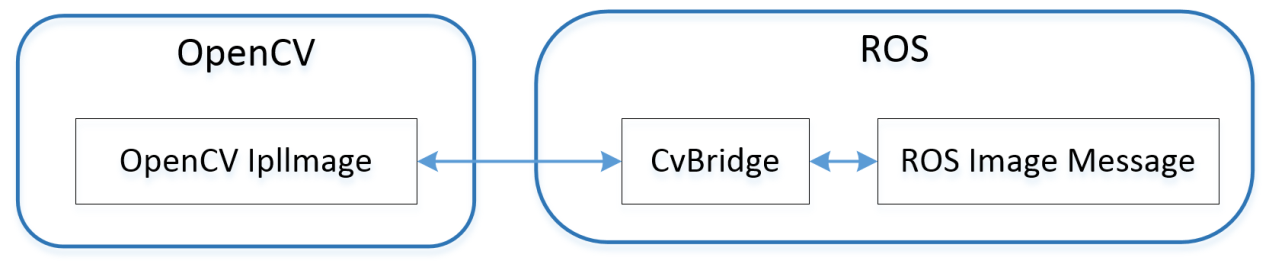
string format

uint8[] data

format: 图像的压缩编码格式（jpeg、png、bmp）

data: 存储图像数据数组

ROS为开发者提供了与OpenCV的接口功能包——cv\_bridge。



通过这个功能包可以将ROS中图像数据转换为OpenCV格式（cv::Mat格式）的图像，并调用OpenCV库进行各种图像处理；也可以将OpenCV处理后的数据转换成ROS格式的图像，通过话题进行发布，实现各节点之间的图像传输。

namespace cv\_bridge {

class CvImage

{

public:

std\_msgs::Header header;

std::string encoding;

cv::Mat image;

};

typedef boost::shared\_ptr<CvImage> CvImagePtr;

typedef boost::shared\_ptr<CvImage const> CvImageConstPtr;

}

## 使用CvBridge转换ROS Images和OpenCV Images

**CvBridge提供了两种转换方式，分别为复制（copy）和共享（share）。**

**1、将ROS的sensor\_msgs/Image消息转换到CvImage，并使用OpenCV函数进行图像处理**

**如果想要修改数据，就需要复制ROS的消息数据**

CvImagePtr **toCvCopy**(const sensor\_msgs::ImageConstPtr& source,

const std::string& encoding = std::string());

CvImagePtr **toCvCopy**(const sensor\_msgs::Image& source,

const std::string& encoding = std::string());

不需要修改数据，可以安全第共享ROS消息的数据，而无需复制。

CvImageConstPtr **toCvShare**(const sensor\_msgs::ImageConstPtr& source,

const std::string& encoding = std::string());

CvImageConstPtr **toCvShare**(const sensor\_msgs::Image& source,

const boost::shared\_ptr<void const>& tracked\_object, const std::string& encoding = std::string());

函数的输入是一个**图像指针**和一个可选的**图像编码**参数用于规定目标CvImage的编码。

toCvCopy函数会从ROS消息中拷贝一个图像数据，不管如何修改CvImage类的内容都不会影响源数据。

对于某些常用的编码，CvBridge提供了可选的color或pixel depth的转换，要想使用这个特性，需要将编码指定为一下格式之一：

* mono8: CV\_8UC1, grayscale image ——灰度图像
* 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

其中mono8和bgr8是大多数OpenCV函数所期望的图像编码格式。

**2、将OpenCV 图像转换到ROS的图像消息**

要转换OpenCV格式图像为ROS图像消息，可以使用**CvImage类**的成员函数toImageMsg()成员函数：

class CvImage

{

sensor\_msgs::ImagePtr toImageMsg() const;

// Overload mainly intended for aggregate messages that contain

// a sensor\_msgs::Image as a member.

void toImageMsg(sensor\_msgs::Image& ros\_image) const;

};

案例1需求——在ROS中使用OpenCV

在ROS节点读取一副图片，并将图像转换为灰度图。

具体的实现流程：

1. 创建功能包image\_converter，并添加依赖项roscpp、rospy、std\_msgs、**cv\_bridge、image\_transport、sensor\_msgs。**
2. 在功能包的src目录下新建image\_converter.cpp文件。
3. 修改CMakeList.txt配置文件

**find\_package(OpenCV REQUIRED)**

include\_directories(

**${OpenCV\_INCLUDE\_DIRS}**

${catkin\_INCLUDE\_DIRS}

)

add\_executable(**image\_converter** src/image\_converter.cpp)

add\_dependencies(**image\_converter** ${${PROJECT\_NAME}\_EXPORTED\_TARGETS}

${catkin\_EXPORTED\_TARGETS})

target\_link\_libraries(**image\_converter**

${catkin\_LIBRARIES}

)

1. 编写源文件image\_converter.cpp

#include <ros/ros.h>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/types\_c.h>

static const std::string OPENCV\_WINDOW = "Image Window";

static const std::string INPUT\_WINDOW = "输入图片";

static const std::string OUTPUT\_WINDOW = "输出图片";

int main(int argc, char\*\* argv)

{

ros::init(argc, argv, "image\_converter");

//读取静态的图像，

cv::Mat img\_in = **cv::imread**("/home/lingmx/HIT.jpg");

if(img\_in.empty())

{

ROS\_ERROR("Read the picture failed!");

return -1;

}

cv::Mat img\_out;

cv::cvtColor(img\_in, img\_out, CV\_RGB2GRAY);

//Show the image

cv::namedWindow("输入图片");

cv::imshow("输入图片",img\_in);

cv::namedWindow("输出图片");

cv::imshow("输出图片", img\_out);

cv::waitKey(0);

}

案例2需求：

使用OpenCV从USB摄像头获取图像数据，并发布。

#include <ros/ros.h>

#include <image\_transport/image\_transport.h>

#include <opencv2/highgui/highgui.hpp>

#include <cv\_bridge/cv\_bridge.h>

#include <sstream>

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);

/\*

对比ROS中发布订阅消息时创建的发布者和订阅者

ros::Publisher pub = nh.advertise<std\_msgs::String>("topic", 100);

ros::Subscriber sub = nh.subscriber<std\_msgs::String>("topic1", 100, callback);

\*/

cv::VideoCapture cap(0);

if (!cap.isOpened()) {

ROS\_INFO("cannot open video device\n");

return 1;

}

**cv::Mat** frame;

**sensor\_msgs::ImagePtr msg**;

ros::Rate loop\_rate(10);//以10ms间隔发送图片

while (ros::ok()) {

**cap >> frame;**

if (!frame.empty()) {

msg = cv\_bridge::CvImage(std\_msgs::Header(), "bgr8", frame).toImageMsg();

pub.publish(msg);

}

ROS\_INFO("runnning!");

ros::spinOnce();

loop\_rate.sleep();//与ros::Rate loop\_rate相对应,休息10ms

}

return 0;

}

案例3：使用C++类实现图像转换



发布方

#include <ros/ros.h>

//ROS中有关图像传输的头文件

#include <image\_transport/image\_transport.h>

#include <sensor\_msgs/image\_encodings.h>

#include <cv\_bridge/cv\_bridge.h>

//OpenCV的头文件

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

int main(int argc, char \*argv[])

{

ros::init(argc, argv, "image\_publisher");

cv::String filename = "/home/lingmx/HIT.jpg";

if(argc == 2)

{

filename = argv[1];

}

cv::Mat img\_in = cv::imread(filename);

if(img\_in.empty())

{

ROS\_ERROR("Read the picture failed!");

return -1;

}

//2.创建图像发布者

ros::NodeHandle nh;

image\_transport::ImageTransport transport(nh);

image\_transport::Publisher image\_pub =

transport.advertise("**/camera/image\_raw**", 1);

//3.图像转换

cv\_bridge::CvImage cvi;

cvi.header.stamp = ros::Time::now();

cvi.header.frame\_id = "image";

cvi.encoding = "**bgr8**";

cvi.image = img\_in;

sensor\_msgs::Image im;

cvi.toImageMsg(im);

ros::Rate loop\_rate(10);

while(ros::ok())

{

//4. 发布图像

image\_pub.publish(im);

ros::spinOnce();

loop\_rate.sleep();

}

/\* code \*/

return 0;

}

#include <ros/ros.h>

#include <image\_transport/image\_transport.h>

#include <cv\_bridge/cv\_bridge.h>

#include <sensor\_msgs/image\_encodings.h>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

static const std::string OPENCV\_WINDOW = "Image window";

class ImageConverter

{

ros::NodeHandle nh\_;

image\_transport::ImageTransport it\_;

image\_transport::Subscriber **image\_sub\_**;

image\_transport::Publisher **image\_pub\_**;

public:

ImageConverter()

: it\_(nh\_)

{

// Subscrive to input video feed and publish output video feed

image\_sub\_ = **it\_**.subscribe("/camera/image\_raw", 1,

&ImageConverter::**imageCb**, this);

image\_pub\_ = **it\_**.advertise("/image\_converter/output\_video", 1);

cv::namedWindow(OPENCV\_WINDOW);

}

~ImageConverter()

{

cv::destroyWindow(OPENCV\_WINDOW);

}

void imageCb(const sensor\_msgs::ImageConstPtr& **msg**)

{

cv\_bridge::CvImagePtr **cv\_ptr**;

try

{

cv\_ptr = **cv\_bridge::****toCvCopy**(msg, sensor\_msgs::image\_encodings::BGR8);

}

catch (cv\_bridge::Exception& e)

{

ROS\_ERROR("cv\_bridge exception: %s", e.what());

return;

}

// Draw an example circle on the video stream

if (cv\_ptr->image.rows > 60 && cv\_ptr->image.cols > 60)

cv::circle(cv\_ptr->image, cv::Point(50, 50), 10, CV\_RGB(255,0,0));

// Update GUI Window

cv::imshow(OPENCV\_WINDOW, cv\_ptr->image);

cv::waitKey(3);

// Output modified video stream

**image\_pub\_**.publish(cv\_ptr->toImageMsg());

}

};

int main(int argc, char\*\* argv)

{

ros::init(argc, argv, "image\_converter");

ImageConverter ic(nh);

ros::spin();

return 0;

}

注意：

1、image\_transport::ImageTransport it\_的构造函数需要的输入参数为ros::NodeHandle nh;

2、图像的发布者和订阅者的实例化对象是ImageTransport::advertise，ImageTransport::subscribe完成的。

将以下的发布者和订阅者的创建方式：

#include <ros/ros.h>

void imageCallback(const sensor\_msgs::ImageConstPtr& msg)

{

// ...

}

ros::NodeHandle nh;

ros::Subscriber sub = nh.subscribe("in\_image\_topic", 1, imageCallback);

ros::Publisher pub = nh.advertise<sensor\_msgs::Image>("out\_image\_topic", 1);

替换为：

// Use the image\_transport classes instead.

#include <ros/ros.h>

#include <image\_transport/image\_transport.h>

void imageCallback(const sensor\_msgs::ImageConstPtr& msg)

{

// ...

}

ros::NodeHandle nh;

image\_transport::ImageTransport it(nh);

image\_transport::Subscriber sub = it.subscribe("in\_image\_base\_topic", 1, imageCallback);

image\_transport::Publisher pub = it.advertise("out\_image\_base\_topic", 1);

image\_transport publishers advertise individual ROS Topics for each availa

3、上述代码的含义是：节点作为订阅者接收了sensor\_msgs::Image消息,在回调函数中，需要对图像数据做些处理，并将处理后数据发布出去。

4、CvImagePtr通俗地说，就是CvImage \*CvImagePtr;

#!/usr/bin/env python

from \_\_future\_\_ import print\_function

import roslib

import sys

import rospy

import cv2

from std\_msgs.msg import String

from sensor\_msgs.msg import Image

from cv\_bridge import CvBridge, CvBridgeError

class image\_converter:

**def \_\_init\_\_(self):**

self.image\_pub = rospy.Publisher("image\_topic\_2",Image)

self.bridge = CvBridge()

self.image\_sub = rospy.Subscriber("image\_topic",Image,self.callback)

**def callback(self,data):**

try:

cv\_image = self.bridge.imgmsg\_to\_cv2(data, "bgr8")

except CvBridgeError as e:

print(e)

#shape函数用来返回图像的高度、宽度以及像素的通道数

(rows,cols,channels) = cv\_image.shape

if cols > 60 and rows > 60 :

cv2.circle(cv\_image, (50,50), 10, 255)

cv2.imshow("Image window", cv\_image)

cv2.waitKey(3)

try:

self.image\_pub.publish(self.bridge.cv2\_to\_imgmsg(cv\_image, "bgr8"))

except CvBridgeError as e:

print(e)

**def main(args):**

ic = image\_converter()

rospy.init\_node('image\_converter', anonymous=True)

try:

rospy.spin()

except KeyboardInterrupt:

print("Shutting down")

cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

main(sys.argv)

**案例3：**

/\*

需求：

获得一副图像，寻找其中的红色像素（一个像素点中红色成分远大于蓝色和绿色成分的和，即认定其为红色像素点），将其颜色修改为白色（RGB=255,255,255），并计算红色像素点的中心，在其中心点出显示蓝色方块。

\*/

#include <ros/ros.h>

#include <image\_transport/image\_transport.h>

#include <cv\_bridge/cv\_bridge.h>

#include <sensor\_msgs/image\_encodings.h>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

static const std::string OPENCV\_WINDOW = "OpenCV display window";

using namespace std;

int g\_redratio = 10; //threshold to decide if a pixel qualifies as dominantly "red"

class ImageConverter {

    ros::NodeHandle nh\_;

    image\_transport::ImageTransport it\_;

    image\_transport::Subscriber image\_sub\_;

    image\_transport::Publisher image\_pub\_;

public:

    ImageConverter(ros::NodeHandle &nodehandle)

    : it\_(nh\_) {

        // Subscribe to input video feed and publish output video feed

        image\_sub\_ = it\_.subscribe("simple\_camera/image\_raw", 1,

                &ImageConverter::imageCb, this);

        image\_pub\_ = it\_.advertise("/image\_converter/output\_video", 1);

        cv::namedWindow(OPENCV\_WINDOW);

    }

    ~ImageConverter() {

        cv::destroyWindow(OPENCV\_WINDOW);

    }

    //image comes in as a ROS message, but gets converted to an OpenCV type

    void imageCb(const sensor\_msgs::ImageConstPtr& msg) {

        cv\_bridge::CvImagePtr cv\_ptr; //OpenCV data type

        try {

            cv\_ptr = cv\_bridge::toCvCopy(msg, sensor\_msgs::image\_encodings::BGR8);

        } catch (cv\_bridge::Exception& e) {

            ROS\_ERROR("cv\_bridge exception: %s", e.what());

            return;

        }

        // 查找红色限速点，并将其转化为黑色，否则转化为白色

        int npix = 0; //count the red pixels

        int isum = 0; //accumulate the column values of red pixels

        int jsum = 0; //accumulate the row values of red pixels

int redval, blueval, greenval, testval;

// OpenCV representation of an RGB pixel

cv::Vec3b rgbpix;

//遍历所有的像素点

        for (int i = 0; i < cv\_ptr->image.cols; i++) {

            for (int j = 0; j < cv\_ptr->image.rows; j++) {

//提取一个像素点

**rgbpix = cv\_ptr->image.at<cv::Vec3b>(j, i);**

                //examine intensity of R, G and B components (0 to 255)

                redval = rgbpix[2] + 1; //add 1, to avoid divide by zero

                blueval = rgbpix[0] + 1;

                greenval = rgbpix[1] + 1;

                //比较像素点的红色成分与蓝色和绿色成分之和

                testval = redval / (blueval + greenval);

                //如果红色成分足够大，认定其为红色像素点，置为白色

                if (testval > g\_redratio) {

                    cv\_ptr->image.at<cv::Vec3b>(j, i)[0] = 255;

                    cv\_ptr->image.at<cv::Vec3b>(j, i)[1] = 255;

                    cv\_ptr->image.at<cv::Vec3b>(j, i)[2] = 255;

                    npix++; //note that found another red pixel

                    isum += i; //accumulate row and col index vals

                    jsum += j;

                }

//否则置为黑色

else {

                    cv\_ptr->image.at<cv::Vec3b>(j, i)[0] = 0;

                    cv\_ptr->image.at<cv::Vec3b>(j, i)[1] = 0;

                    cv\_ptr->image.at<cv::Vec3b>(j, i)[2] = 0;

                }

            }

        }

//以上的过程相当于将彩色图像转换为黑白图像

        cv::Mat gray\_image,contours;

        //convert the color image to grayscale:

        cv::cvtColor(cv\_ptr->image, gray\_image, CV\_BGR2GRAY);

        //use Canny filter to find edges in grayscale image;

        //put result in "contours"; low and high thresh are tunable params

        cv::Canny(gray\_image,// gray-level image

        contours, // output contours

        125,// low threshold

        350);// high threshold

        //cout << "npix: " << npix << endl;

        //paint in a blue square at the centroid:

        int half\_box = 5; // choose size of box to paint

        int i\_centroid, j\_centroid;

        double x\_centroid, y\_centroid;

        if (npix > 0) {

            i\_centroid = isum / npix; // average value of u component of red pixels

            j\_centroid = jsum / npix; // avg v component

            x\_centroid = ((double) isum)/((double) npix); //floating-pt version

            y\_centroid = ((double) jsum)/((double) npix);

            ROS\_INFO("u\_avg: %f; v\_avg: %f",x\_centroid,y\_centroid);

            //cout << "i\_avg: " << i\_centroid << endl; //i,j centroid of red pixels

            //cout << "j\_avg: " << j\_centroid << endl;

**for (int i\_box = i\_centroid - half\_box; i\_box <= i\_centroid + half\_box; i\_box++) {**

**for (int j\_box = j\_centroid - half\_box; j\_box <= j\_centroid + half\_box; j\_box++) {**

**//make sure indices fit within the image**

**if ((i\_box >= 0)&&(j\_box >= 0)&&(i\_box < cv\_ptr->image.cols)&&(j\_box < cv\_ptr->image.rows))**

**{**

**cv\_ptr->image.at<cv::Vec3b>(j\_box, i\_box)[0] = 255;                          cv\_ptr->image.at<cv::Vec3b>(j\_box, i\_box)[1] = 0;**

**cv\_ptr->image.at<cv::Vec3b>(j\_box, i\_box)[2] = 0;**

**}**

**}**

**}**

        }

        // Update GUI Window; this will display processed images on the open-cv viewer.

        //cv::imshow(OPENCV\_WINDOW, cv\_ptr->image); //display processed image

        //cv::imshow(OPENCV\_WINDOW, gray\_image); //display the grayscale image

        cv::imshow(OPENCV\_WINDOW, contours); //display the contours

        cv::waitKey(3); //need waitKey call to update OpenCV image window

        // Also, publish the processed image as a ROS message on a ROS topic

        // can view this stream in ROS with:

        //rosrun image\_view image\_view image:=/image\_converter/output\_video

        image\_pub\_.publish(cv\_ptr->toImageMsg());

    }

}; //end of class definition

int main(int argc, char\*\* argv) {

    ros::init(argc, argv, "red\_pixel\_finder");

    ros::NodeHandle n; //

    ImageConverter ic(n); // instantiate object of class ImageConverter

    //cout << "enter red ratio threshold: (e.g. 10) ";

    //cin >> g\_redratio;

    g\_redratio= 10; //choose a threshold to define what is "red" enough

    ros::Duration timer(0.1);

    double x, y, z;

    while (ros::ok()) {

        ros::spinOnce();

        timer.sleep();

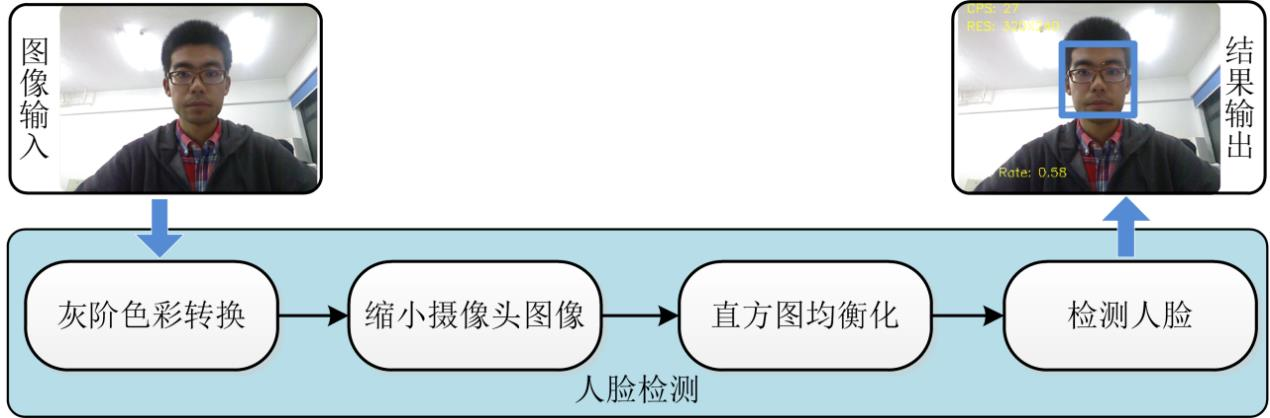
    }

    return 0;

}

案例5：ROS + OpenCV应用示例：人脸识别

基于Haar特征的级联分类器对象检测算法



具体实现过程：

1、启动roscore

2、启动摄像头 rosrun usb\_cam usb\_cam\_node

3、启动人脸识别 roslaunch robot\_vision face\_detector.launch

4、启动rqt\_image\_view

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

import rospy

import cv2

import numpy as np

from sensor\_msgs.msg import Image, RegionOfInterest

from cv\_bridge import CvBridge, CvBridgeError

class faceDetector:

    def \_\_init\_\_(self):

        rospy.on\_shutdown(self.cleanup);

        # 创建cv\_bridge

        self.bridge = CvBridge()

        self.image\_pub = rospy.Publisher("**cv\_bridge\_image**", Image, queue\_size=1)

        # 获取haar特征的级联表的XML文件，文件路径在launch文件中传入

        cascade\_1 = rospy.get\_param("~cascade\_1", "")

        cascade\_2 = rospy.get\_param("~cascade\_2", "")

        # 使用级联表初始化haar特征检测器

        self.cascade\_1 = cv2.CascadeClassifier(cascade\_1)

        self.cascade\_2 = cv2.CascadeClassifier(cascade\_2)

        # 设置级联表的参数，优化人脸识别，可以在launch文件中重新配置

        self.haar\_scaleFactor  = rospy.get\_param("~haar\_scaleFactor", 1.2)

        self.haar\_minNeighbors = rospy.get\_param("~haar\_minNeighbors", 2)

        self.haar\_minSize      = rospy.get\_param("~haar\_minSize", 40)

        self.haar\_maxSize      = rospy.get\_param("~haar\_maxSize", 60)

        self.color = (50, 255, 50)

        # 初始化订阅rgb格式图像数据的订阅者，此处图像topic的话题名可以在launch文件中重映射

        self.image\_sub = rospy.Subscriber("input\_rgb\_image", Image, self.image\_callback, queue\_size=1)

    def image\_callback(self, data):

        # 使用cv\_bridge将ROS的图像数据转换成OpenCV的图像格式

        try:

            cv\_image = self.bridge.imgmsg\_to\_cv2(data, "bgr8")

            frame = np.array(cv\_image, dtype=np.uint8)

        except CvBridgeError as e:

            print(e)

        # 创建灰度图像

        grey\_image = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)rq

        # 创建平衡直方图，减少光线影响

        grey\_image = cv2.equalizeHist(grey\_image)

        # 尝试检测人脸

        faces\_result = self.detect\_face(grey\_image)

        # 在opencv的窗口中框出所有人脸区域

        if len(faces\_result)>0:

            for face in faces\_result:

                x, y, w, h = face

                cv2.rectangle(cv\_image, (x, y), (x+w, y+h), self.color, 2)

        # 将识别后的图像转换成ROS消息并发布

        self.image\_pub.publish(self.bridge.cv2\_to\_imgmsg(cv\_image, "bgr8"))

    def detect\_face(self, input\_image):

        # 首先匹配正面人脸的模型

        if self.cascade\_1:

            faces = self.cascade\_1.detectMultiScale(input\_image,

                    self.haar\_scaleFactor,

                    self.haar\_minNeighbors,

                    cv2.CASCADE\_SCALE\_IMAGE,

                    (self.haar\_minSize, self.haar\_maxSize))

        # 如果正面人脸匹配失败，那么就尝试匹配侧面人脸的模型

        if len(faces) == 0 and self.cascade\_2:

            faces = self.cascade\_2.detectMultiScale(input\_image,

                    self.haar\_scaleFactor,

                    self.haar\_minNeighbors,

                    cv2.CASCADE\_SCALE\_IMAGE,

                    (self.haar\_minSize, self.haar\_maxSize))

        return faces

    def cleanup(self):

        print("Shutting down vision node.")

        cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

    try:

        # 初始化ros节点

        rospy.init\_node("face\_detector")

        faceDetector()

        rospy.loginfo("Face detector is started..")

        rospy.loginfo("Please subscribe the ROS image.")

        rospy.spin()

    except KeyboardInterrupt:

        print("Shutting down face detector node.")

        cv2.destroyAllWindows()

## 共享图像数据的例子

在上面的介绍中我们创建了图像的拷贝，但共享图像也很容易：

namespace enc = sensor\_msgs::image\_encodings;

void imageCb(const sensor\_msgs::ImageConstPtr& msg)

{

cv\_bridge::CvImageConstPtr cv\_ptr;

try

{

if (enc::isColor(msg->encoding))

cv\_ptr = cv\_bridge::toCvShare(msg, enc::BGR8);

else

cv\_ptr = cv\_bridge::**toCvShare**(msg, enc::MONO8);

}

catch (cv\_bridge::Exception& e)

{

ROS\_ERROR("cv\_bridge exception: %s", e.what());

return;

}

// Process cv\_ptr->image using OpenCV

}

如果输入图像的编码是"bgr8"，cv\_ptr将会是图像数据的一个别名而非拷贝。如果输入图像不是"bgr8"编码但是可转换为"bgr8"编码（如"mono8"），CvBridge将会为cv\_ptr分配一个新的buffer并执行转换。

在图像转换过程中，通常使用异常捕获语句。

如果没有异常捕获语句的话一行代码就能共享图像了，但可能输入图像的编码无法转换为目标编码而导致节点崩溃。

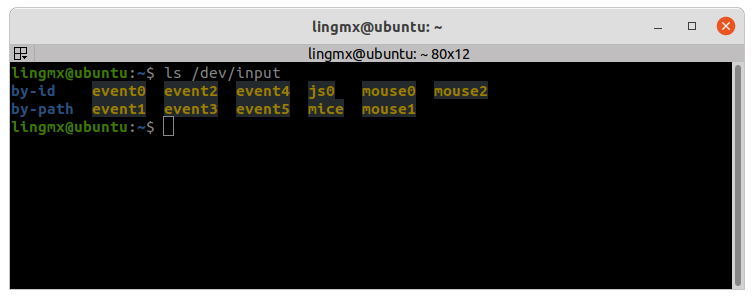
## 在ROS中使用游戏手柄

游戏手柄也是在ROS中常用的一种外部输入设备，我们可以通过游戏手柄来改变速度和方向来远程控制机器人。

**如何安装游戏手柄？**

1. 首先将游戏手柄插到电脑的USB端口，在终端中输入如下指令：

ls /dev/input



其中的js0就代表游戏手柄。

测试操作杆的操作是否有效，输入:

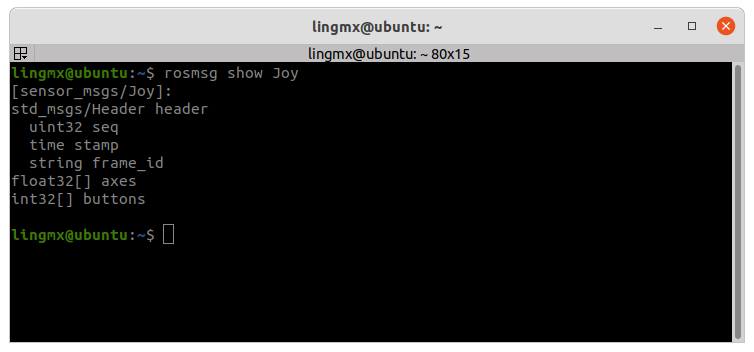
        jstest /dev/input/js0

        然后会在终端中显示操作杆的各个控制值的即时值，操作操作杆，如果每个按键和操作都有效，说明操作杆是正常的。最后在ROS中的节点里进行测试。

从上面的测试中，会发现在前后左右摇动操作杆时，相应改变的数据是Axes中0号和1号位的数据，也是我们最常用的数据，其他按键对应的位置也可以找到，编程的时候就是利用的这些数据位置，所以一定要找到每个按键的对应编号。

打开joy节点：

        rosrun joy joy\_node



1、创建一个功能包，名称为turtle\_teleop\_joy，添加依赖项roscpp rospy std\_msgs sensor\_msgs turtlesim

2、在src目录下，新建文件turtle\_teleop\_joy.cpp

3、编辑文件如下：

#include "ros/ros.h"

#include "geometry\_msgs/Twist.h"

#include "sensor\_msgs/Joy.h"

class Teleop

{

public:

Teleop(ros::NodeHandle nh);

private:

ros::NodeHandle nh\_;

ros::Publisher pub;

ros::Subscriber sub;

void callBack(const sensor\_msgs::Joy::ConstPtr& msg );

};

Teleop::Teleop(ros::NodeHandle nh)

{

nh\_= nh;

ROS\_INFO("Initializing Subscribers");

pub = nh\_.advertise<geometry\_msgs::Twist>(**"turtle1/cmd\_vel"**, 10);

ROS\_INFO("Initializing Subscribers");

sub = nh\_.subscribe<sensor\_msgs::Joy>("**joy**", 10, &**Teleop::callBack**, this);

}

void Teleop::callBack(const sensor\_msgs::Joy::ConstPtr& msg)

{

geometry\_msgs::Twist cmd\_vel;

cmd\_vel.linear.x = msg->axes[1];

cmd\_vel.angular.z = msg->axes[3];

pub.publish(cmd\_vel);

}

int main(int argc, char \*argv[])

{

setlocale(LC\_ALL,"");

ros::init(argc, argv, "teleop\_joy");

ros::NodeHandle nh;

Teleop teleop\_joy(nh);

ros::spin();

/\* code \*/

return 0;

}

<launch>

<node pkg="turtlesim" type="turtlesim\_node" name="turtlesim"/>

<node pkg ="joy" type="joy\_node" name="joy\_node" output="screen"/>

<node pkg="demo\_turtlesim" type="turtle\_teleop\_joy"

name="turtle\_teleop\_joy" output="screen"/>

</launch>

