

ROS理论与实践

—— 第7讲：机器视觉处理



主讲人 胡春旭







机器人博客“古月居”博主

《ROS机器人开发实践》作者

武汉精锋微控科技有限公司 联合创始人

华中科技大学 人工智能与自动化学院 硕士



-  1. ROS摄像头驱动及数据接口
-  2. 摄像头参数标定
-  3. ROS+OpenCV图像处理方法及案例
-  4. ROS+Tensorflow物体识别方法及案例



1. ROS摄像头驱动及数据接口



1. ROS摄像头驱动及数据接口





1. ROS摄像头驱动及数据接口

```
$ sudo apt-get install ros-melodic-usb-cam
$ roslaunch usb_cam usb_cam-test.launch
$ rqt_image_view
```



usb_cam功能包中的话题

	名称	类型	描述
Topic发布	~<camera_name>/image	sensor_msgs/Image	发布图像数据

usb_cam功能包中的参数

参数	类型	默认值	描述
~video_device	string	"/dev/video0"	摄像头设备号
~image_width	int	640	图像横向分辨率
~image_height	int	480	图像纵向分辨率
~pixel_format	string	"mjpeg"	像素编码, 可选值: mjpeg, yuyv, uyvy
~io_method	string	"mmap"	IO通道, 可选值: mmap, read, userptr
~camera_frame_id	string	"head_camera"	摄像头坐标系
~framerate	int	30	帧率
~brightness	int	32	亮度, 0~255
~saturation	int	32	饱和度, 0~255
~contrast	int	32	对比度, 0~255
~sharpness	int	22	清晰度, 0~255
~autofocus	bool	false	自动对焦
~focus	int	51	焦点 (非自动对焦状态下有效)
~camera_info_url	string	-	摄像头校准文件路径
~camera_name	string	"head_camera"	摄像头名称



1. ROS摄像头驱动及数据接口

```
<launch>
  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="yuyv" />
    <param name="camera_frame_id" value="usb_cam" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
    <remap from="image" to="/usb_cam/image_raw"/>
    <param name="autosize" value="true" />
  </node>
</launch>
```

usb_cam-test.launch



1. ROS摄像头驱动及数据接口

- **Header**: 消息头, 包含消息序号, 时间戳和绑定坐标系;
- **height**: 图像的纵向分辨率;
- **width**: 图像的横向分辨率;
- **encoding**: 图像的编码格式, 包含RGB、YUV等常用格式, 不涉及图像压缩编码;
- **is_bigendian**: 图像数据的大小端存储模式;
- **step**: 一行图像数据的字节数量, 作为数据的步长参数;
- **data**: 存储图像数据的数组, 大小为step*height个字节

```
→ ~ rosmmsg show sensor_msgs/Image
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

1280*720分辨率的摄像头产生一帧图像的数据大小是: $3 \times 1280 \times 720 = 2764800$ 字节, 即2.7648MB



1. ROS摄像头驱动及数据接口

压缩图像消息

```
→ ~ rosmmsg 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



1. ROS摄像头驱动及数据接口



➤ 安装驱动

```
$ sudo apt-get install ros-melodic-freenect-*  
$ git clone https://github.com/avin2/SensorKinect.git  
$ cd SensorKinect/Bin  
$ tar xvf SensorKinect093-Bin-Linux-x64-v5.1.2.1.tar.bz2  
$ sudo ./install.sh （在解压出来的文件夹路径下）
```

➤ 运行

```
$ roslaunch freenect_launch freenect.launch
```

freenect_camera功能包中的话题和服务

	名称	类型	描述
Topic 发布	rgb/camera_info	sensor_msgs/CameraInfo	RGB相机校准信息
	rgb/image_raw	sensor_msgs/Image	RGB相机图像数据
	depth/camera_info	sensor_msgs/CameraInfo	深度相机校准信息
	depth/image_raw	sensor_msgs/Image	深度相机数据
	depth_registered/camera_info	sensor_msgs/CameraInfo	配准后的深度相机校准信息
	depth_registered/image_raw	sensor_msgs/Image	配准后的深度相机数据
	ir/camera_info	sensor_msgs/CameraInfo	红外相机校准信息
	ir/image_raw	sensor_msgs/Image	红外相机数据
	projector/camera_info	sensor_msgs/CameraInfo	深度相机校准信息
	/diagnostics	diagnostic_msgs/DiagnosticArray	传感器诊断信息
Services	rgb/set_camera_info	sensor_msgs/SetCameraInfo	设置RGB相机的校准信息
	ir/set_camera_info	sensor_msgs/SetCameraInfo	设置红外相机的校准信息



1. ROS摄像头驱动及数据接口

```
<launch>

  <!-- 启动freenect驱动 -->
  <include file="$(find freenect_launch)/launch/freenect.launch">
    <arg name="publish_tf" value="false" />
    <arg name="depth_registration" value="true" />
    <arg name="rgb_processing" value="true" />
    <arg name="ir_processing" value="false" />
    <arg name="depth_processing" value="false" />
    <arg name="depth_registered_processing" value="true" />
    <arg name="disparity_processing" value="false" />
    <arg name="disparity_registered_processing" value="false" />
    <arg name="sw_registered_processing" value="false" />
    <arg name="hw_registered_processing" value="true" />
  </include>
</launch>
```

freenect.launch



1. ROS摄像头驱动及数据接口

- **height**: 点云图像的纵向分辨率;
- **width**: 点云图像的横向分辨率;
- **fields**: 每个点的数据类型;
- **is_bigendian**: 数据的大小端存储模式;
- **point_step**: 单点的数据字节步长;
- **row_step**: 一行数据的字节步长;
- **data**: 点云数据的存储数组, 总字节大小为 $\text{row_step} * \text{height}$;
- **is_dense**: 是否有无效点。

```
→ ~ rosmmsg show sensor_msgs/PointCloud2
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
sensor_msgs/PointField[] fields
  uint8 INT8=1
  uint8 UINT8=2
  uint8 INT16=3
  uint8 UINT16=4
  uint8 INT32=5
  uint8 UINT32=6
  uint8 FLOAT32=7
  uint8 FLOAT64=8
  string name
  uint32 offset
  uint8 datatype
  uint32 count
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

点云单帧数据量也很大, 如果使用分布式网络传输, 需要考虑能否满足数据的传输要求, 或者针对数据进行压缩。



1. ROS摄像头驱动及数据接口

➤ 安装SDK (<https://github.com/intel-ros/realsense/releases>)

```
$ mkdir build
```

```
$ cd build
```

```
$ cmake ..
```

```
$ make
```

```
$ sudo make install
```



➤ 安装ROS驱动 (<https://github.com/IntelRealSense/librealsense/releases>)

```
$ catkin_make -DCATKIN_ENABLE_TESTING=False -DCMAKE_BUILD_TYPE=Release
```

```
$ catkin_make install
```

```
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrcsource ~/.bashrc
```

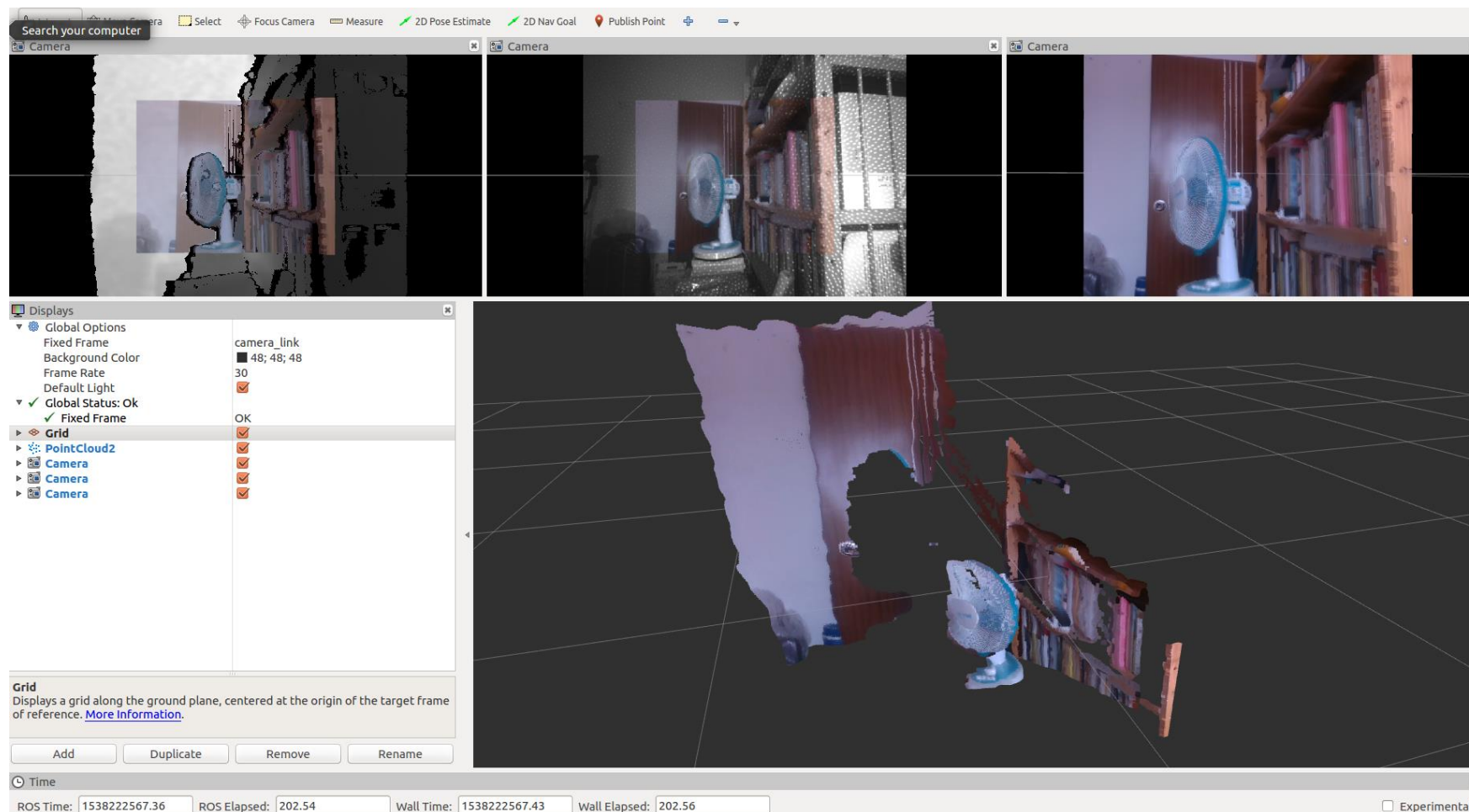
参考链接:

<https://github.com/IntelRealSense/librealsense/blob/master/doc/installation.md>

<https://blog.csdn.net/u012926144/article/details/80761342>



1. ROS摄像头驱动及数据接口



点云显示

```
$ roslaunch realsense2_camera rs_rgbd.launch  
$ rosrn rviz rviz
```



2. 摄像头参数标定



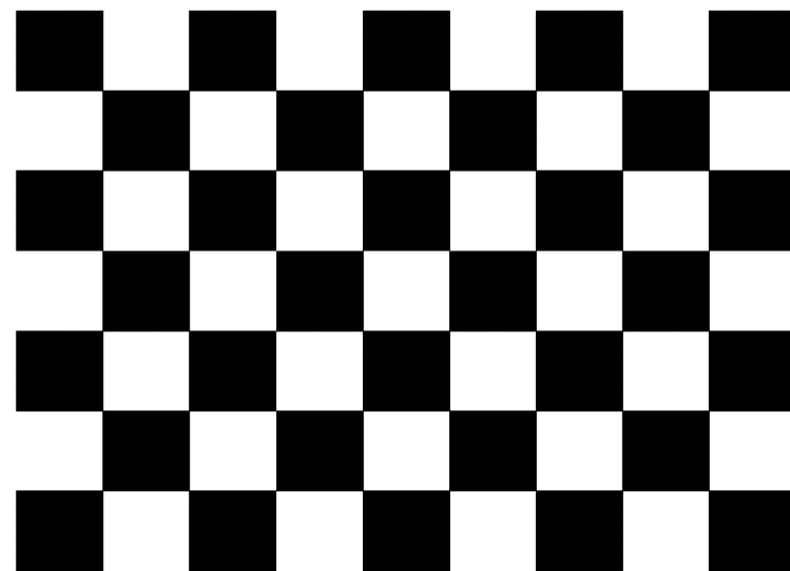
2. 摄像头参数标定

➤ 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，由于摄像头内部与外部的一些原因，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，需要针对摄像头的参数进行标定。

安装标定功能包

```
$ sudo apt-get install ros-melodic-camera-calibration
```



棋盘格标定靶



2. 摄像头参数标定

摄像头标定流程

➤ 启动摄像头

```
$ roslaunch robot_vision usb_cam.launch
```

➤ 启动标定包

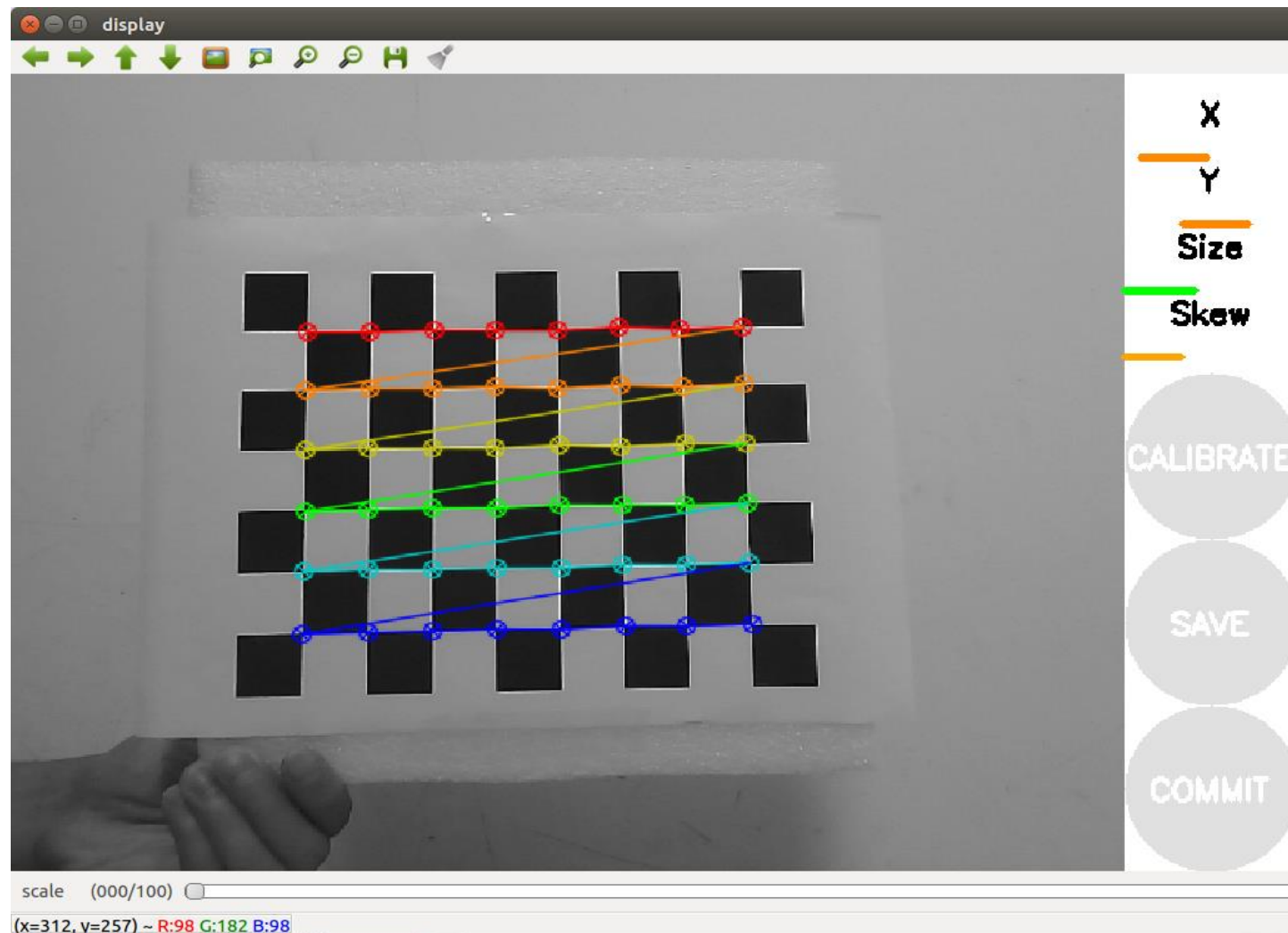
```
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.024  
image:=/usb_cam/image_raw camera:=/usb_cam
```

1. size: 标定棋盘格的内部角点个数，这里使用的棋盘一共有六行，每行有8个内部角点；
2. square: 这个参数对应每个棋盘格的边长，单位是米；
3. image和camera: 设置摄像头发布的图像话题。



2. 摄像头参数标定

- **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的上下移动;
- **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的倾斜转动。



标定过程



2. 摄像头参数标定

Kinect标定流程

➤ 启动Kinect

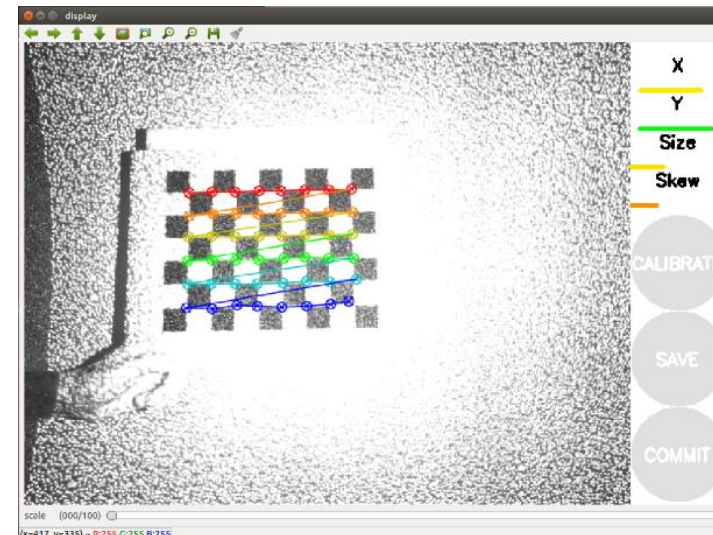
```
$ roslaunch robot_vision freenect.launch
```

➤ 启动彩色摄像头

```
$ rosrun camera_calibration cameracalibrator.py image:=/camera/rgb/image_raw  
camera:=/camera/rgb --size 8x6 --square 0.024
```

➤ 标定红外摄像头

```
$ rosrun camera_calibration cameracalibrator.py image:=/camera/ir/image_raw  
camera:=/camera/ir --size 8x6 --square 0.024
```





3. ROS+OpenCV图像处理方法及案例

OpenCV是什么？

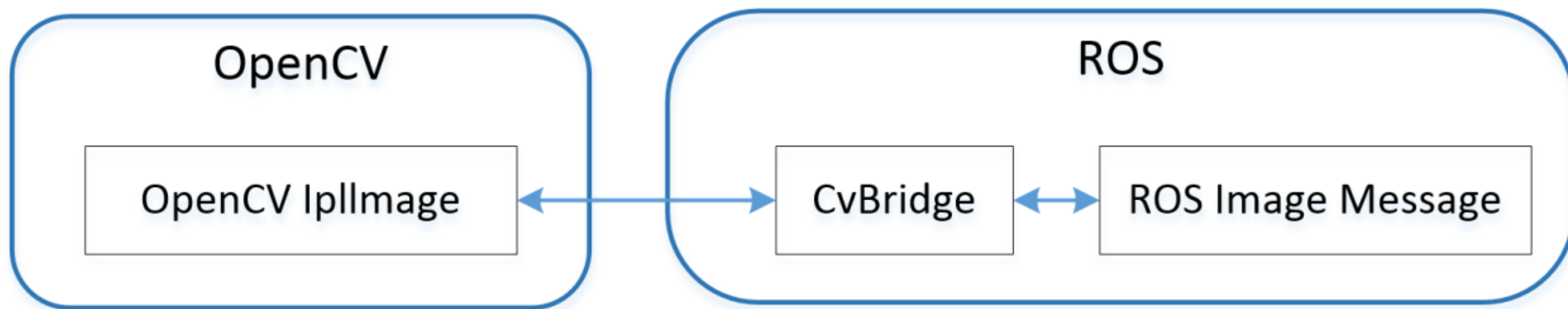
- Open Source Computer Vision Library;
- 基于BSD许可发行的跨平台开源计算机视觉库（Linux、Windows和Mac OS等）；
- 由一系列C函数和少量C++类构成，同时提供C++、Python、Ruby、MATLAB等语言的接口；
- 实现了图像处理和计算机视觉方面的很多通用算法，而且对非商业应用和商业应用都是免费的；
- 可以直接访问硬件摄像头，并且还提供了一个简单的GUI系统——highgui。



3. ROS+OpenCV图像处理方法及案例

安装OpenCV

```
$ sudo apt-get install ros-melodic-vision-opencv libopencv-dev python-opencv
```



ROS与OpenCV的集成框架



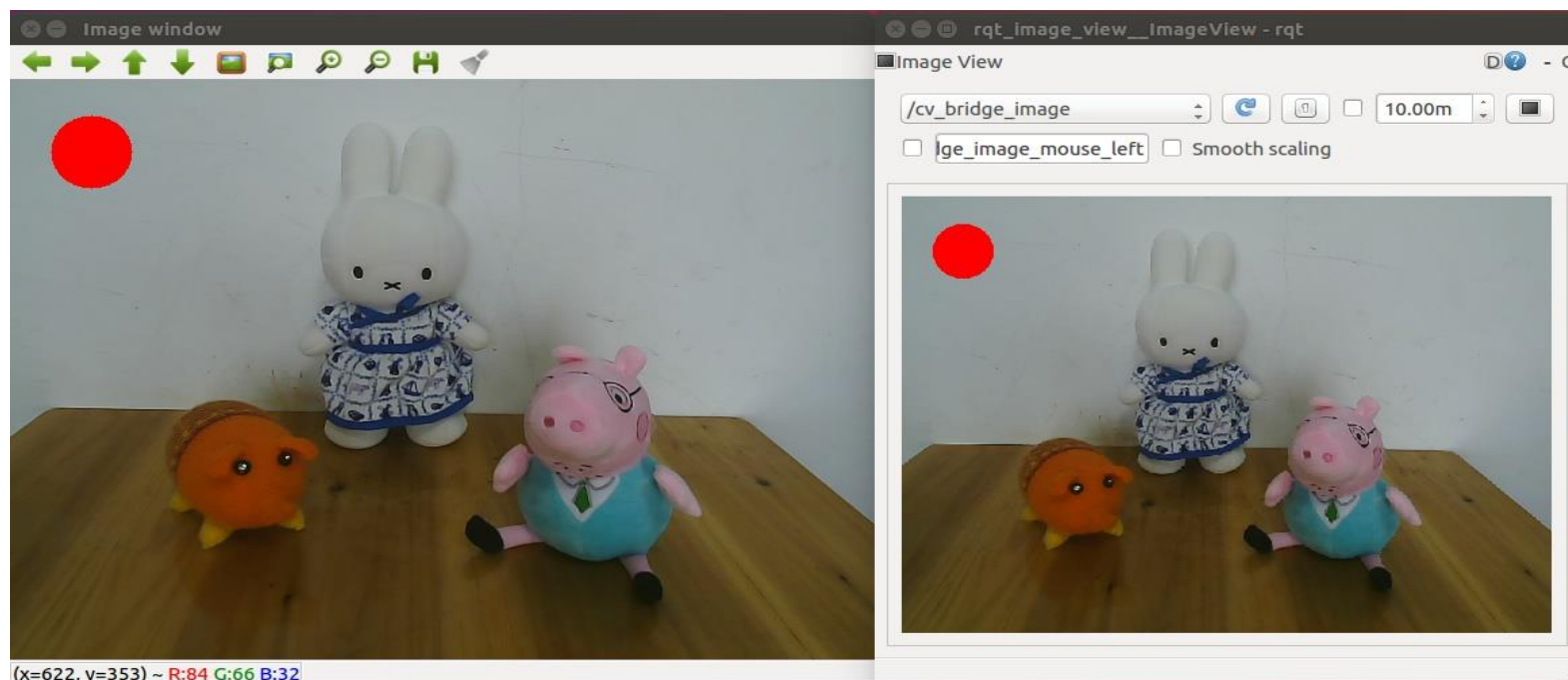
3. ROS+OpenCV图像处理方法及案例

测试例程

```
$ roslaunch robot_vision usb_cam.launch
```

```
$ rosrun robot_vision cv_bridge_test.py
```

```
$ rqt_image_view
```





3. ROS+OpenCV图像处理方法及案例

- `imgmsg_to_cv2()`: 将ROS图像消息转换成OpenCV图像数据;
- `cv2_to_imgmsg()`: 将OpenCV格式的图像数据转换成ROS图像消息;

* 输入参数:

1. 图像消息流
2. 转换的图像数据格式

robot_vision/scripts/cv_bridge_test.py

```
import rospy
import cv2
from cv_bridge import CvBridge, CvBridgeError
from sensor_msgs.msg import Image

class image_converter:
    def __init__(self):
        # 创建cv_bridge, 声明图像的发布者和订阅者
        self.image_pub = rospy.Publisher("cv_bridge_image", Image, queue_size=1)
        self.bridge = CvBridge()
        self.image_sub = rospy.Subscriber("/usb_cam/image_raw", Image, self.callback)

    def 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")
        except CvBridgeError as e:
            print e

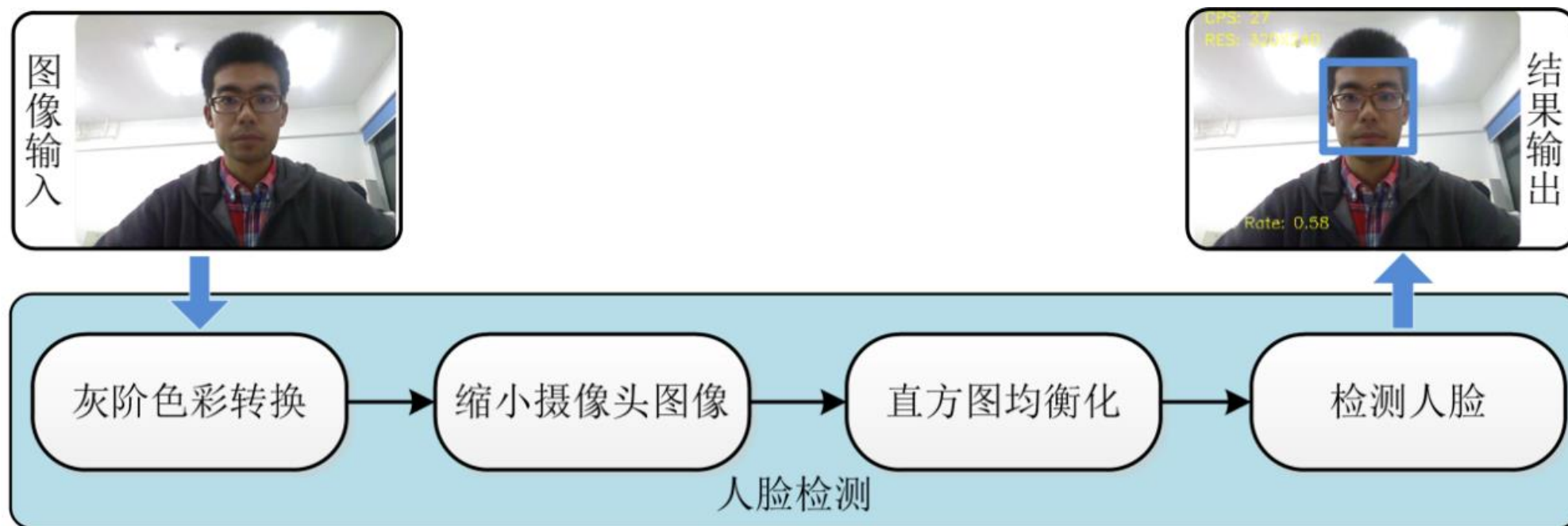
        # 在opencv的显示窗口中绘制一个圆, 作为标记
        (rows, cols, channels) = cv_image.shape
        if cols > 60 and rows > 60 :
            cv2.circle(cv_image, (60, 60), 30, (0,0,255), -1)

        # 显示Opencv格式的图像
        cv2.imshow("Image window", cv_image)
        cv2.waitKey(3)

        # 再将opencv格式数据转换成ros image格式的数据发布
        try:
            self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image, "bgr8"))
        except CvBridgeError as e:
            print e

if __name__ == '__main__':
    try:
        # 初始化ros节点
        rospy.init_node("cv_bridge_test")
        rospy.loginfo("Starting cv_bridge_test node")
        image_converter()
        rospy.spin()
    except KeyboardInterrupt:
        print "Shutting down cv_bridge_test node."
        cv2.destroyAllWindows()
```


3. ROS+OpenCV图像处理方法及案例



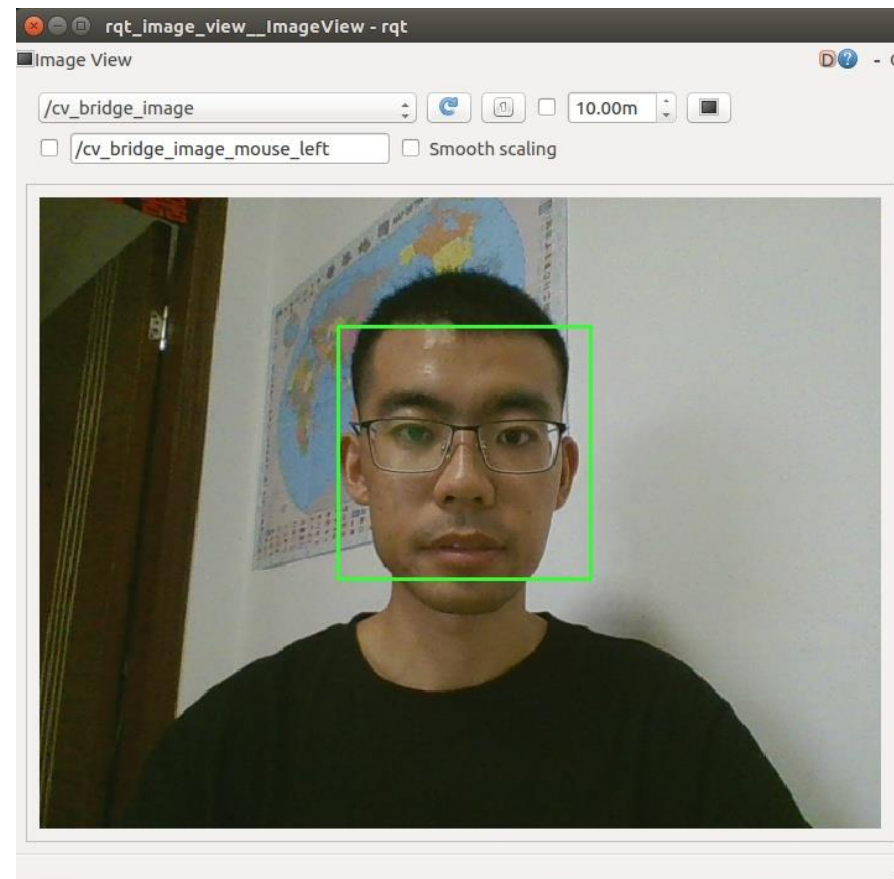
基于Haar特征的级联分类器对象检测算法

启动人脸识别实例

```
$ roslaunch robot_vision usb_cam.launch
```

```
$ roslaunch robot_vision face_detector.launch
```

```
$ rqt_image_view
```



人脸识别效果



3. ROS+OpenCV图像处理方法及案例

➤ 初始化部分：

完成ROS节点、图像、识别参数的设置。

➤ 回调函数：

将图像转换成OpenCV的数据格式，然后预处理之后开始调用人脸识别的功能函数，最后把识别结果发布。

➤ 人脸识别

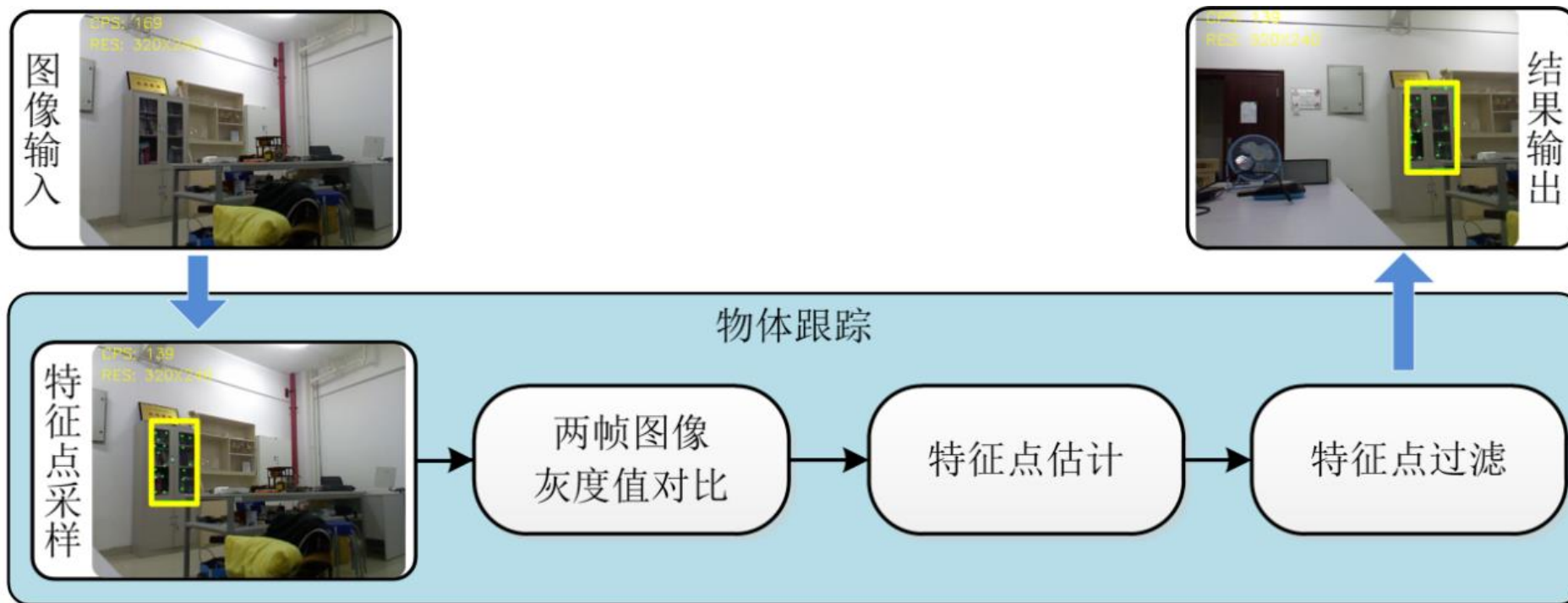
调用OpenCV提供的人脸识别接口，与数据库中的人脸特征进行匹配。

robot_vision/script/face_detector.py

```
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, e:  
        print e  
  
    # 创建灰度图像  
    grey_image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  
  
    # 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
```



3. ROS+OpenCV图像处理方法及案例



跟踪物体的特征点



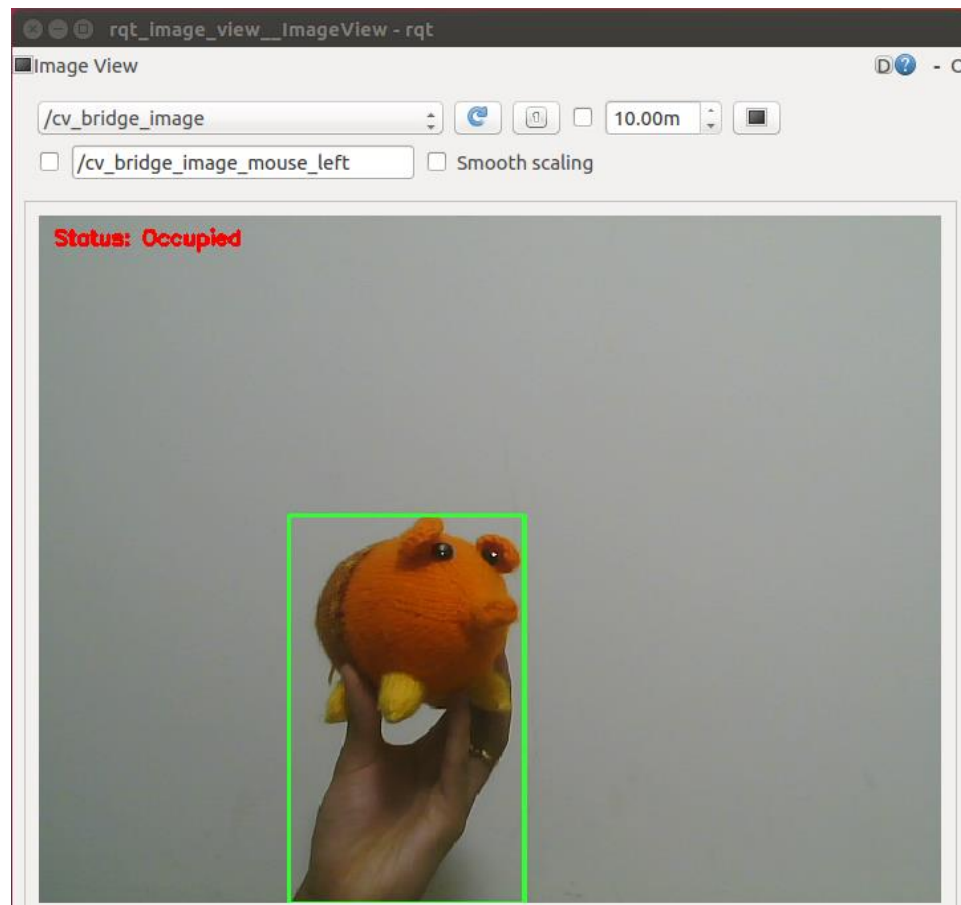
3. ROS+OpenCV图像处理方法及案例

启动物体跟踪实例

```
$ roslaunch robot_vision usb_cam.launch
```

```
$ roslaunch robot_vision motion_detector.launch
```

```
$ rqt_image_view
```



物体跟踪效果



3. ROS+OpenCV图像处理方法及案例

➤ 初始化部分：

完成ROS节点、图像、识别参数的设置。

➤ 图像处理：

将图像转换成OpenCV格式；完成图像预处理之后开始针对两帧图像进行比较，基于图像差异识别到运动的物体，最后标识识别结果并发布。

```
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, e:  
        print e  
  
    # 创建灰度图像  
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  
    gray = cv2.GaussianBlur(gray, (21, 21), 0)  
  
    # 使用两帧图像做比较，检测移动物体的区域  
    if self.firstFrame is None:  
        self.firstFrame = gray  
        return  
    frameDelta = cv2.absdiff(self.firstFrame, gray)  
    thresh = cv2.threshold(frameDelta, self.threshold, 255, cv2.THRESH_BINARY)[1]  
  
    thresh = cv2.dilate(thresh, None, iterations=2)  
    binary, cnts, hierarchy= cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  
  
    for c in cnts:  
        # 如果检测到的区域小于设置值，则忽略  
        if cv2.contourArea(c) < self.minArea:  
            continue  
  
        # 在输出画面上框出识别到的物体  
        (x, y, w, h) = cv2.boundingRect(c)  
        cv2.rectangle(frame, (x, y), (x + w, y + h), (50, 255, 50), 2)  
        self.text = "Occupied"  
  
        # 在输出画面上打当前状态和时间戳信息  
        cv2.putText(frame, "Status: {}".format(self.text), (10, 20),  
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 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(frame, "bgr8"))
```

robot_vision/script/face_detector.py/motion_detector.py

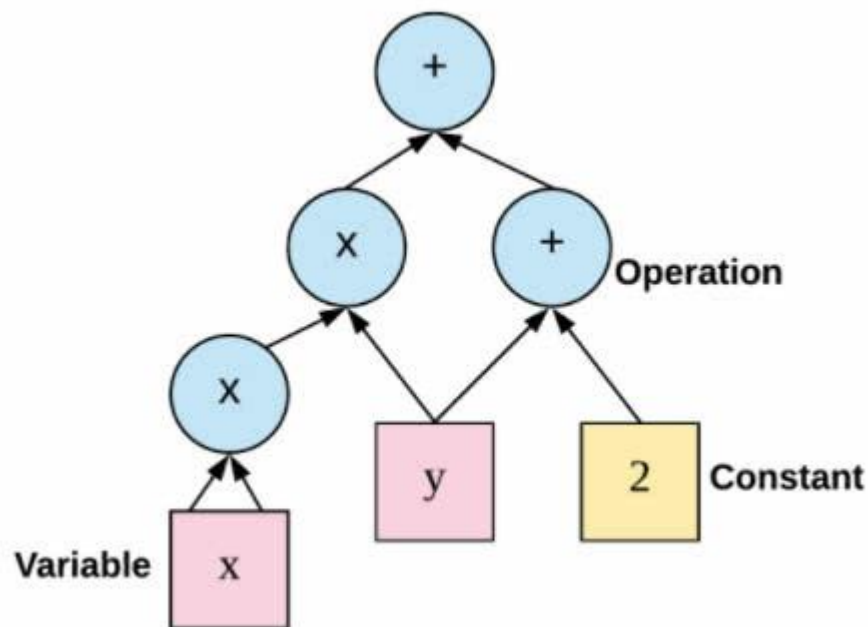


4. ROS+Tensorflow物体识别方法及案例

TensorFlow 是一个用于人工智能的开源神器

TensorFlow是什么？

- 采用数据流图，用于数值计算的开源软件库。
- 节点在图中表示数学操作，线表示在节点间相互联系的多维数据数组，即张量（tensor）。
- 架构灵活，可以在多种平台上展开计算。
- 最初由Google大脑小组（隶属于Google机器学习研究机构）的研究员和工程师们开发出来，用于机器学习和深度神经网络方面的研究，但这个系统的通用性使其也可广泛用于其他计算领域。



4. ROS+Tensorflow物体识别方法及案例

➤ 安装Tensorflow

```
$ sudo apt-get install python-pip python-dev python-virtualenv
```

```
$ virtualenv --system-site-packages ~/tensorflow
```

```
$ source ~/tensorflow/bin/activate
```

```
$ easy_install -U pip
```

```
$ pip install --upgrade tensorflow
```

➤ 安装功能包

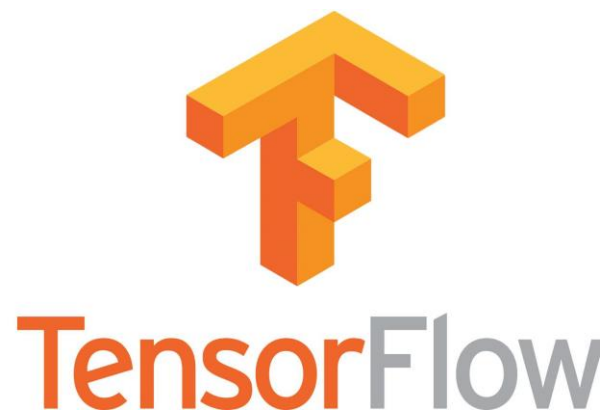
```
$ cd ~/catkin_ws/src
```

```
$ git clone https://github.com/Kukanani/vision\_msgs.git
```

```
$ git clone https://github.com/osrf/tensorflow\_object\_detector.git
```

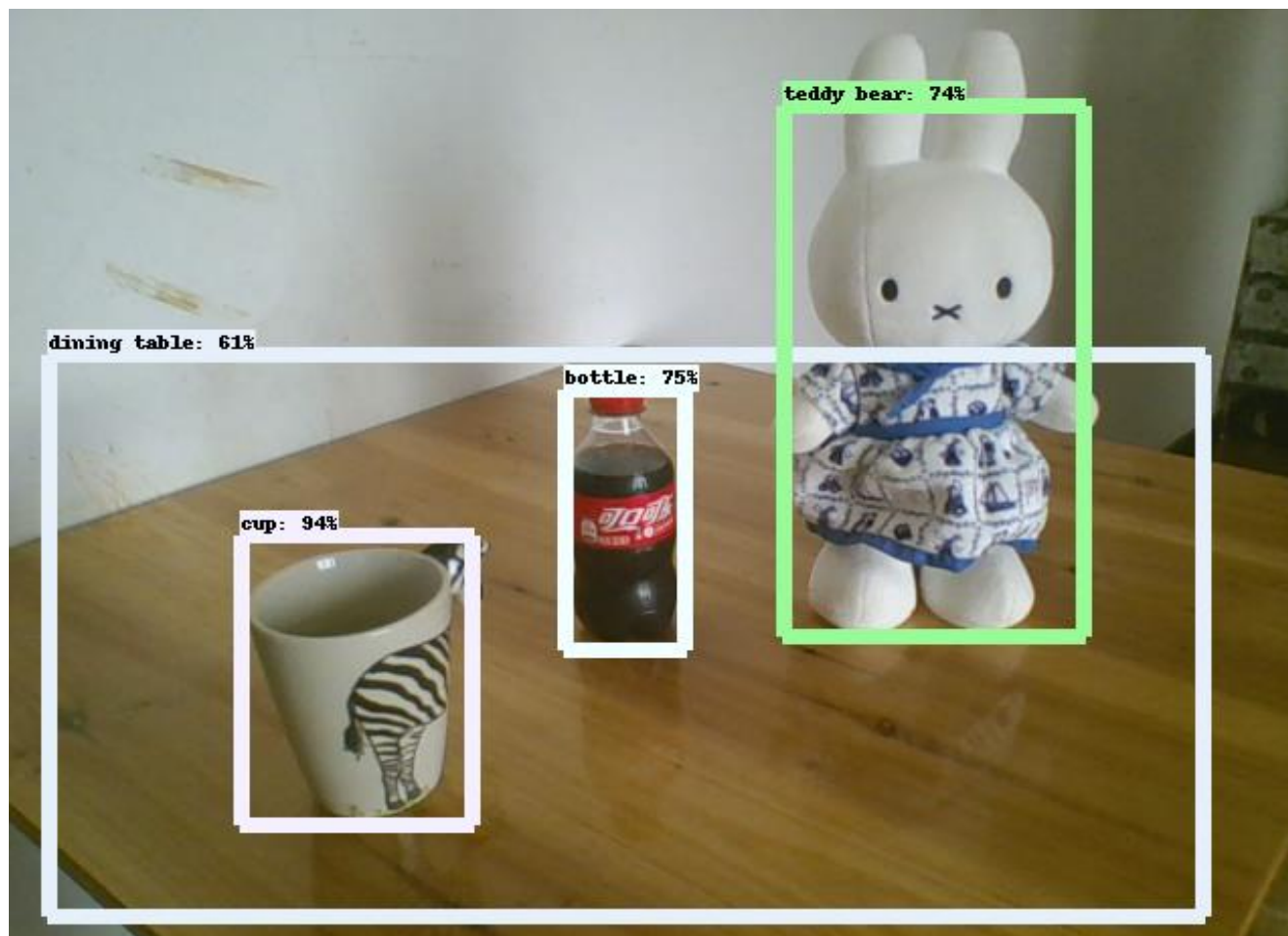
➤ 编译功能包

```
$ cd ~/catkin_ws && catkin_make
```





4. ROS+Tensorflow物体识别方法及案例



物体识别 \$ roslaunch tensorflow_object_detector usb_cam_detector.launch

4. ROS+Tensorflow物体识别方法及案例



基于深度学习的视觉分拣



基于深度学习的字符排序分拣



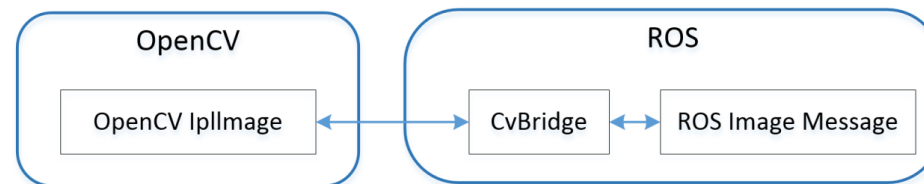
ROS摄像头驱动及数据接口

- RGB摄像头: sensor_msgs/Image
- RGBD摄像头: sensor_msgs/PointCloud2

摄像头参数标定

- camera_calibration

ROS+OpenCV 图像处理方法及案例



ROS+Tensorflow 物体识别方法及案例

- tensorflow_object_detector



1. 使用人脸识别的方式实现以下场景：

- 通过人脸识别的方式，发布速度控制命令，控制上讲作业中实现的仿真机器人运动；
- 例如：人脸向左移动，小车向左转，人脸向前移动，小车前进等。

2. 按照本讲第四节的内容，复现tensorflow物体识别的案例，并实现以下跟随场景：

- 识别一个杯子及其在图像中的位置；
- 根据杯子的识别结果，发布速度控制命令，控制上讲作业中实现的仿真机器人运动；
- 例如：杯子远离摄像头，小车前进，杯子在图像中向左运动，小车主转等。



- ROS cv_bridge wiki
http://wiki.ros.org/cv_bridge
- ROS opencv_apps
http://wiki.ros.org/opencv_apps
- OpenCV教程
<https://www.w3cschool.cn/opencv/opencv-2gnx28u3.html>
- tensorflow_object_detector
https://github.com/osrf/tensorflow_object_detector.git
- TensorFlow Tutorials
<https://tensorflow.google.cn/tutorials/>

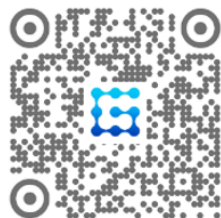




Thank You

怕什么真理无穷，进一寸有一寸的欢喜

更多精彩，欢迎关注



 古月居



 古月春旭