ROS机械臂开发

——3. ROS机器人视觉应用

目录

目录

- 1. ROS摄像头驱动

1. ROS摄像头驱动

常用传感器的ROS驱动可参考：http://wiki.ros.org/Sensors

1. ROS摄像头驱动 —— 连接摄像头

$ sudo apt-get install ros-kinetic-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 —— 连接摄像头

```xml
<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 —— ROS中的图像数据（二维图像）

- 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

```
➜  ~ rosmsg 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
```

1080*720分辨率的摄像头产生一帧图像的数据大小是：3*1080*720=2764800字节，即2.7648MB

1. ROS摄像头驱动 —— ROS中的图像数据（二维图像）

压缩图像消息

```
➜  ~ 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

1. ROS摄像头驱动 —— 连接Kinect

$ sudo apt-get install ros-kinetic-freenect-*

$ git clone https://github.com/avin2/SensorKinect.git

$ cd SensorKinect/Bin

$ tar xvf SensorKinect093-Bin-Linux-x86-v5.1.2.1.tar.bz2 (32bit)

$ tar xvf SensorKinect093-Bin-Linux-x64-v5.1.2.1.tar.bz2 (64bit)

$ sudo ./install.sh （在解压出来的文件夹路径下）

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 —— 连接Kinect

```xml
<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 —— ROS中的图像数据（三维图像）

- 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：一列数据的字节步长；
- data：点云数据的存储数组，总字节大小为 row_step* height；
- is_dense：是否有无效点。

```
➜  ~ rosmsg 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 —— 连接Realsense

➢ 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

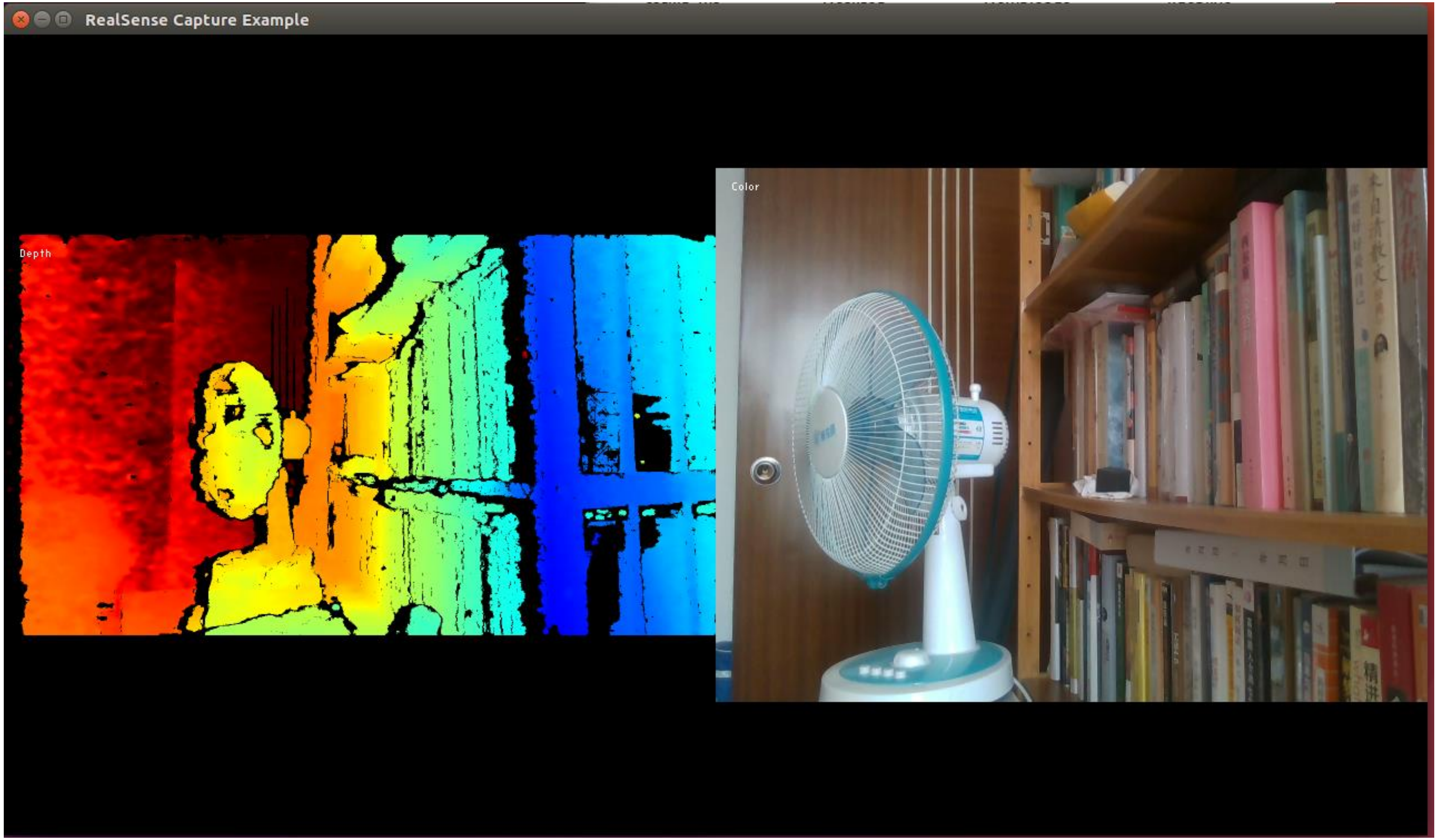
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrcsource ~/.bashrc

参考链接：

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

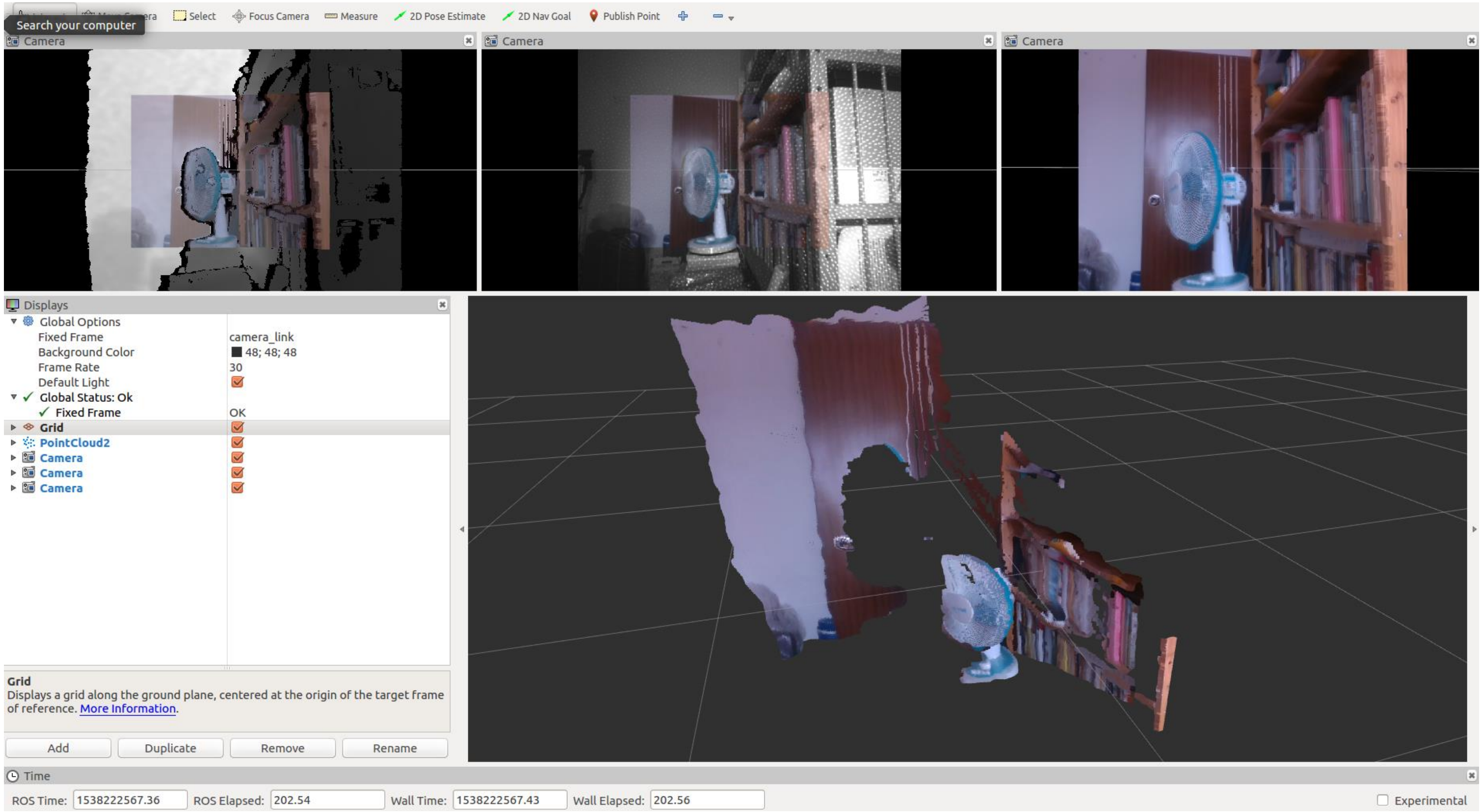
https://blog.csdn.net/u012926144/article/details/80761342

1. ROS摄像头驱动 —— 连接Realsense



Realsense SDK example

1. ROS摄像头驱动 —— 连接Realsense



点云显示

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

目录

➢2. 摄像头内参标定

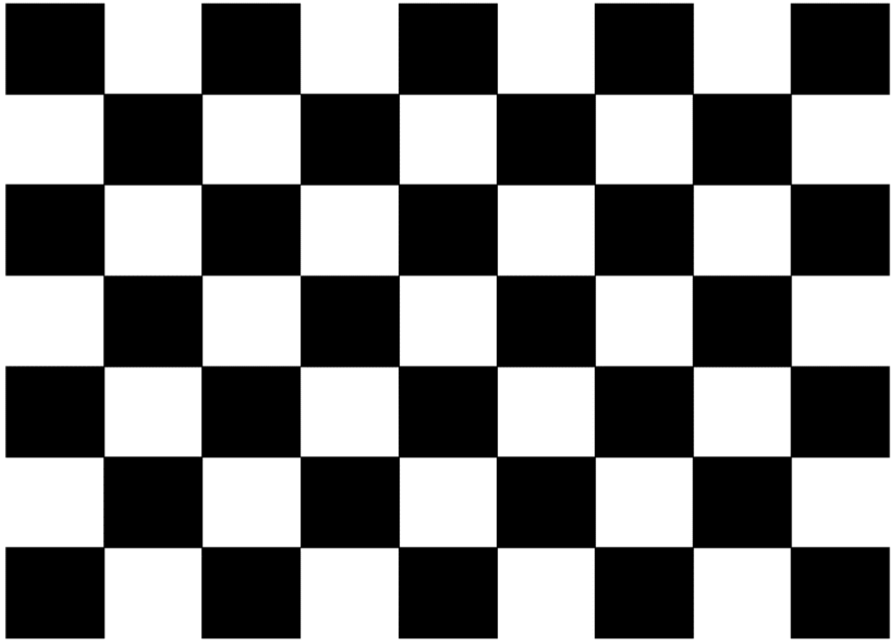
2.摄像头内参标定

➢ 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-kinetic-camera-calibration
```



棋盘格标定靶

2.摄像头内参标定

摄像头标定流程

➢ 启动摄像头

$ roslaunch robot_vision usb_cam.launch

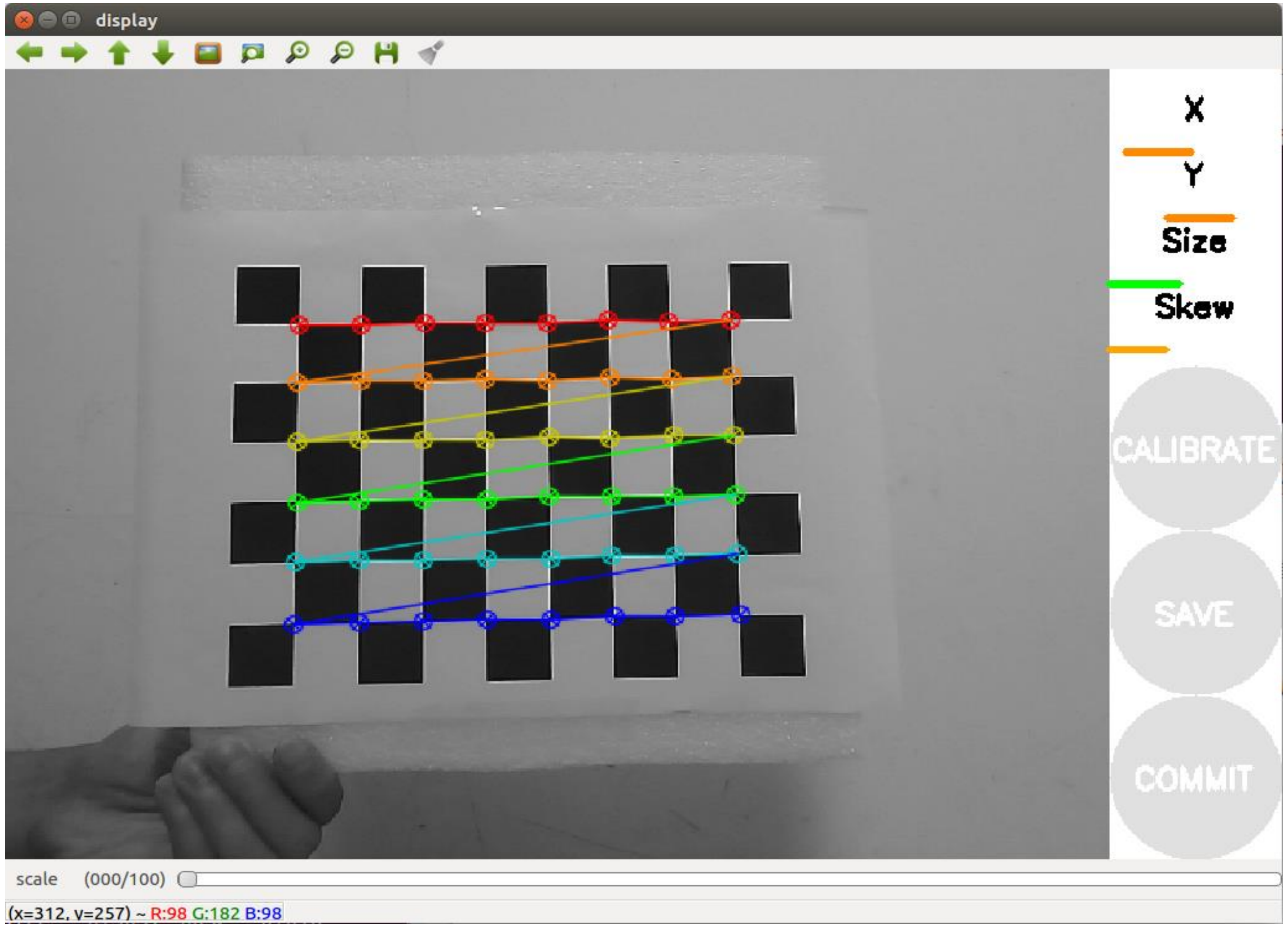
➢ 启动标定包

$ 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.摄像头内参标定

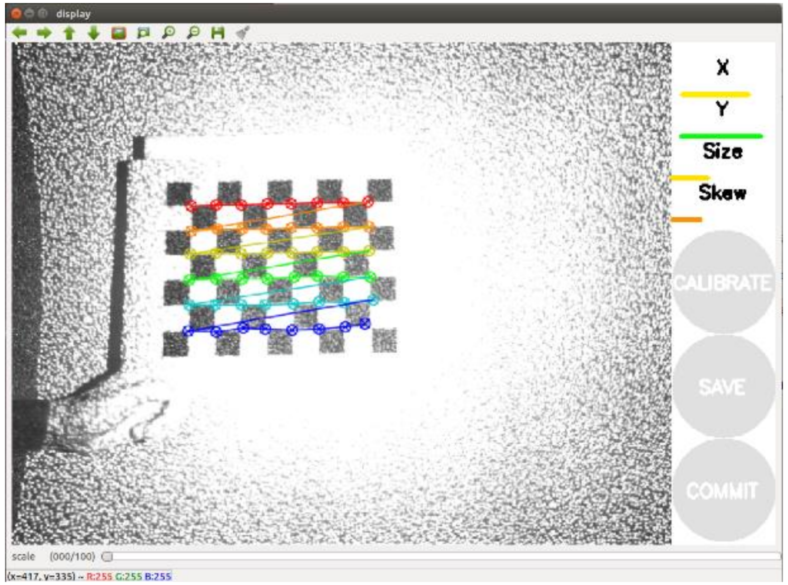
- X：标定靶在摄像头视野中的左右移动；
- Y：标定靶在摄像头视野中的上下移动；
- Size：标定靶在摄像头视野中的前后移动；
- Skew：标定靶在摄像头视野中的倾斜转动。



标定过程

2.摄像头内参标定

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 --si

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 8

--square 0.024

2.摄像头内参标定

摄像头如何使用标定文件？

```
<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="1280" />
        <param name="image_height" value="720" />
        <param name="pixel_format" value="yuyv" />
        <param name="camera_frame_id" value="usb_cam" />
        <param name="io_method" value="mmap"/>

        <param name="camera_info_url" type="string" value="file://$(find robot_vision)/camera_calibration.yaml" />
    </node>

</launch>
```

robot_vision/launch/usb_cam_with_calibration.launch

2.摄像头内参标定

Kinect如何使用标定文件？

```
<launch>

    <!-- Launch the freenect driver -->
    <include file="$(find freenect_launch)/launch/freenect.launch">
        <arg name="publish_tf"                      value="false" />

        <!-- use device registration -->
        <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" />

        <arg name="rgb_camera_info_url" value="file://$(find robot_vision)/kinect_rgb_calibration.yaml" />
        <arg name="depth_camera_info_url" value="file://$(find robot_vision)/kinect_depth_calibration.yaml" />
    </include>

</launch>
```

robot_vision/launch/freenect_with_calibration.launch

目录

➢3. ROS+OpenCV应用

3. ROS+OpenCV应用

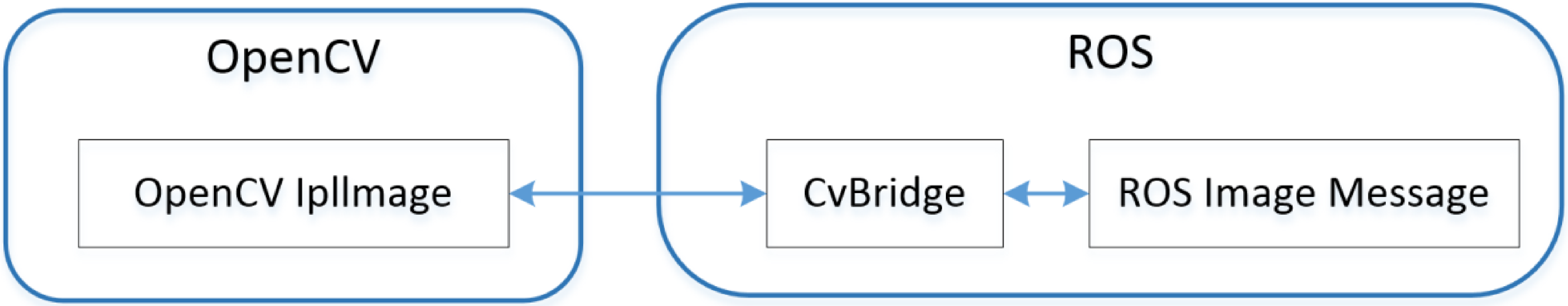
OpenCV是什么？

- 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应用 —— cvbridge

安装OpenCV

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



ROS与OpenCV的集成框架

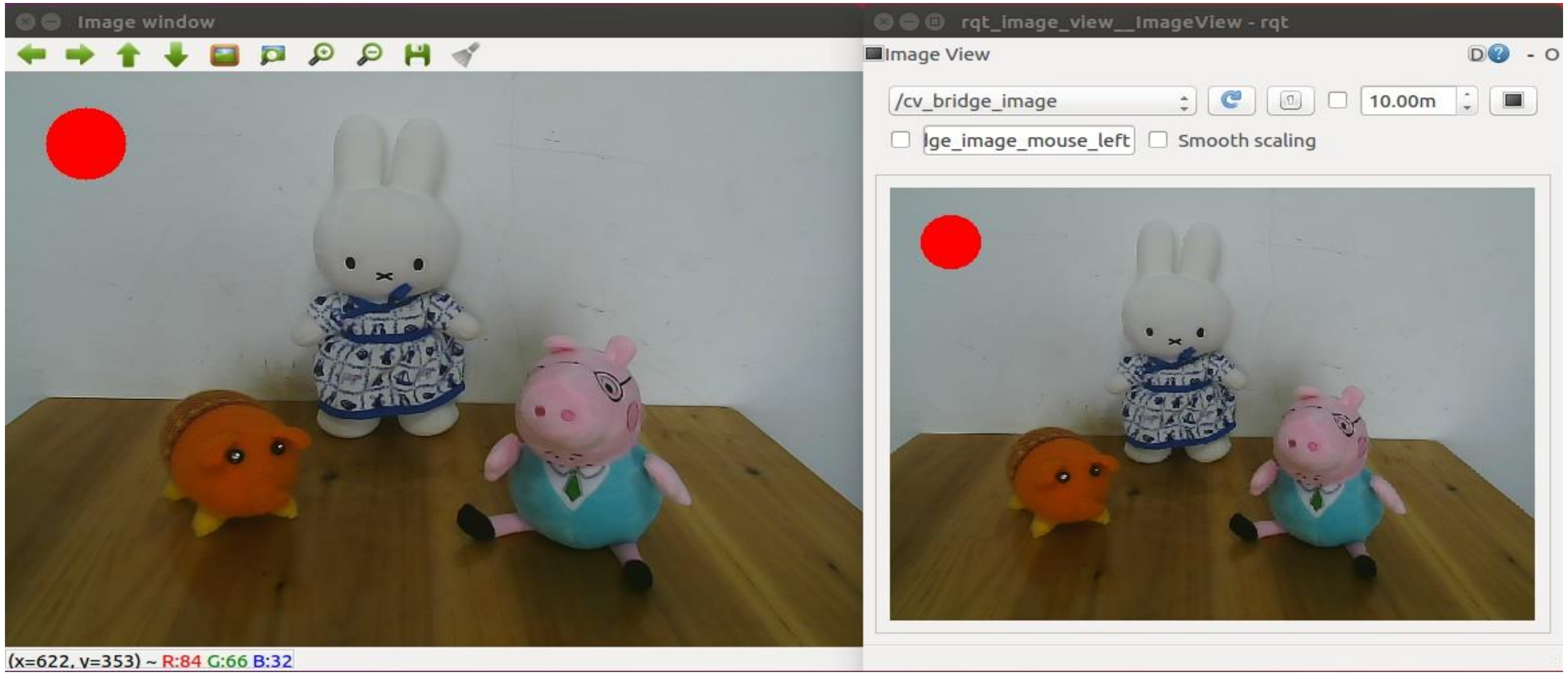
3. ROS+OpenCV应用 —— cvbridge

测试例程

$ roslaunch robot_vision usb_cam.launch

$ rosrun robot_vision cv_bridge_test.py

$ rqt_image_view



3. ROS+OpenCV应用 —— cvbridge

- **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；

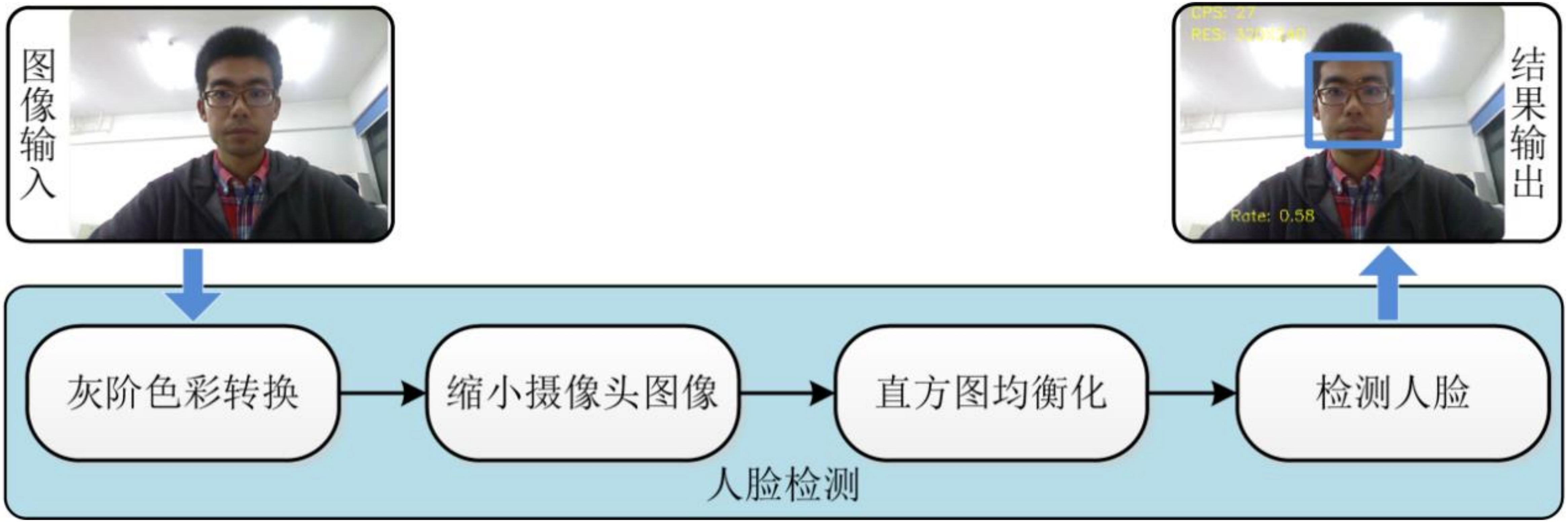
*输入参数：

1. 图像消息流
2. 转换的图像数据格式

```
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格式额数据转换成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()
```

robot_vision/scripts/cv_bridge_test.py

3. ROS+OpenCV应用 —— 人脸识别



基于Haar特征的级联分类器对象检测算法

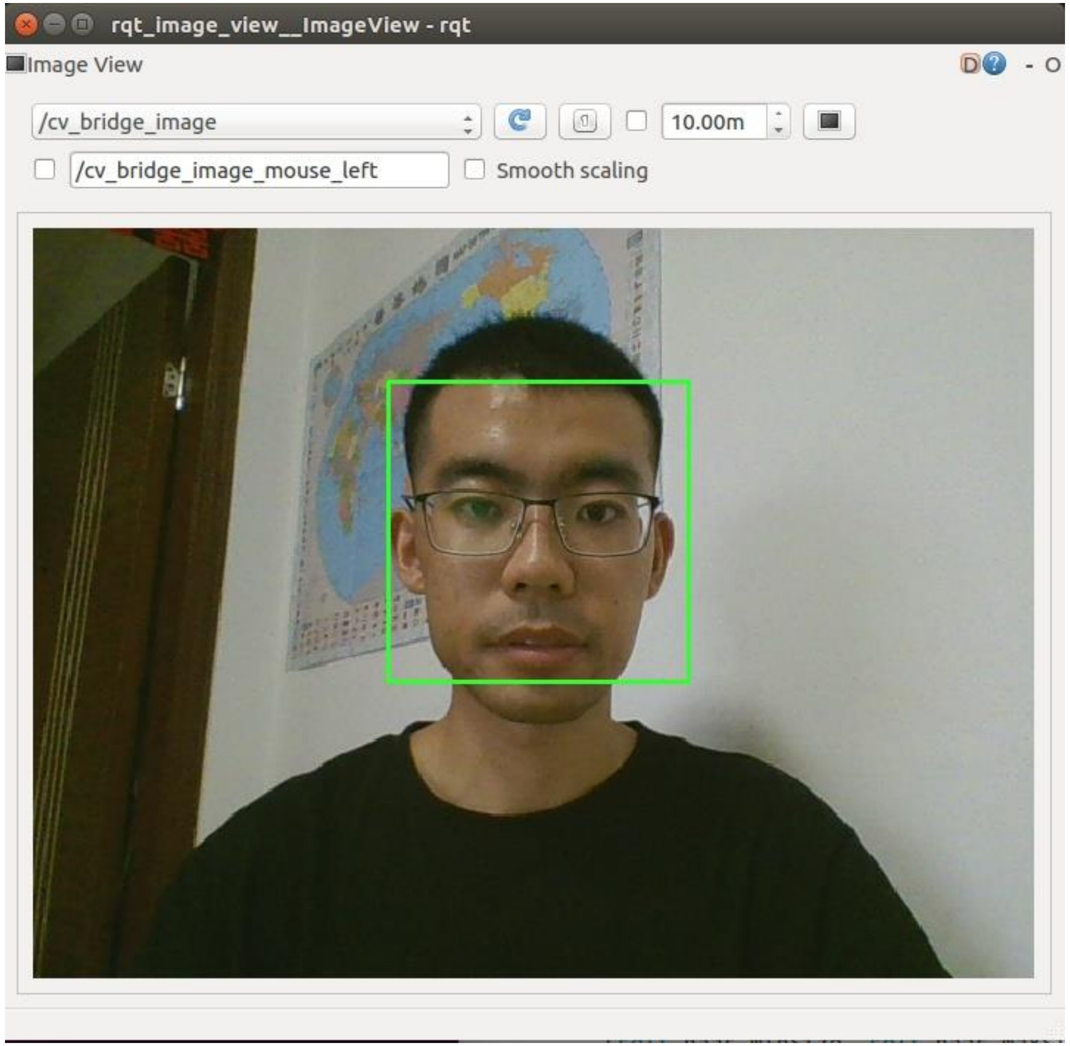
3. ROS+OpenCV应用 —— 人脸识别

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应用 —— 人脸识别

➢ 初始化部分：

完成ROS节点、图像、识别参数的设置。

➢ ROS图像回调函数：

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
```

目录

➢4. 二维码识别

4. 二维码识别

安装二维码识别功能包

```
$ sudo apt-get install ros-kinetic-ar-track-alvar
```

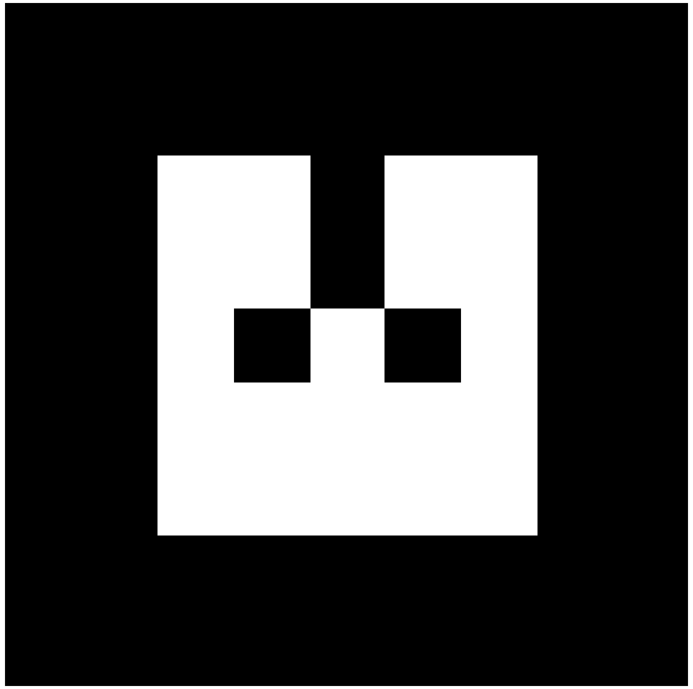
4. 二维码识别

创建二维码

$ rosrun ar_track_alvar createMarker

$ rosrun ar_track_alvar createMarker o

```
➜  ~ rosrun ar_track_alvar createMarker
SampleMarkerCreator
===================

Description:
  This is an example of how to use the 'MarkerData' and 'MarkerArtoolkit'
  classes to generate marker images. This application can be used to
  generate markers and multimarker setups that can be used with
  SampleMarkerDetector and SampleMultiMarker.

Usage:
  /opt/ros/kinetic/lib/ar_track_alvar/createMarker [options] argument

    65535              marker with number 65535
    -f 65535           force hamming(8,4) encoding
    -1 "hello world"   marker with string
    -2 catalog.xml     marker with file reference
    -3 www.vtt.fi      marker with URL
    -u 96              use units corresponding to 1.0 unit per 96 pixels
    -uin               use inches as units (assuming 96 dpi)
    -ucm               use cm's as units (assuming 96 dpi) <default>
    -s 5.0             use marker size 5.0x5.0 units (default 9.0x9.0)
    -r 5               marker content resolution -- 0 uses default
    -m 2.0             marker margin resolution -- 0 uses default
    -a                 use ArToolkit style matrix markers
    -p                 prompt marker placements interactively from the user
```



4. 二维码识别

创建二维码

```
$ roscd robot_vision/config
$ rosrun ar_track_alvar createMarker -s 5 0
$ rosrun ar_track_alvar createMarker -s 5 1
$ rosrun ar_track_alvar createMarker -s 5 2
```

4. 二维码识别

```
<launch>

    <node pkg="tf" type="static_transform_publisher" name="world_to_cam"
          args="0 0 0.5 0 1.57 0 world usb_cam 10" />

    <arg name="marker_size" default="5" />
    <arg name="max_new_marker_error" default="0.08" />
    <arg name="max_track_error" default="0.2" />
    <arg name="cam_image_topic" default="/usb_cam/image_raw" />
    <arg name="cam_info_topic" default="/usb_cam/camera_info" />
    <arg name="output_frame" default="/usb_cam" />

    <node name="ar_track_alvar" pkg="ar_track_alvar" type="individualMarkersNoKinect" respawn="false" output="screen">
        <param name="marker_size"          type="double" value="$(arg marker_size)" />
        <param name="max_new_marker_error" type="double" value="$(arg max_new_marker_error)" />
        <param name="max_track_error"      type="double" value="$(arg max_track_error)" />
        <param name="output_frame"         type="string" value="$(arg output_frame)" />

        <remap from="camera_image"  to="$(arg cam_image_topic)" />
        <remap from="camera_info"   to="$(arg cam_info_topic)" />
    </node>

    <!-- rviz view /-->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find robot_vision)/config/ar_track_camera.rviz"/>

</launch>
```

robot_vision/launch/ar_track_camera.launch

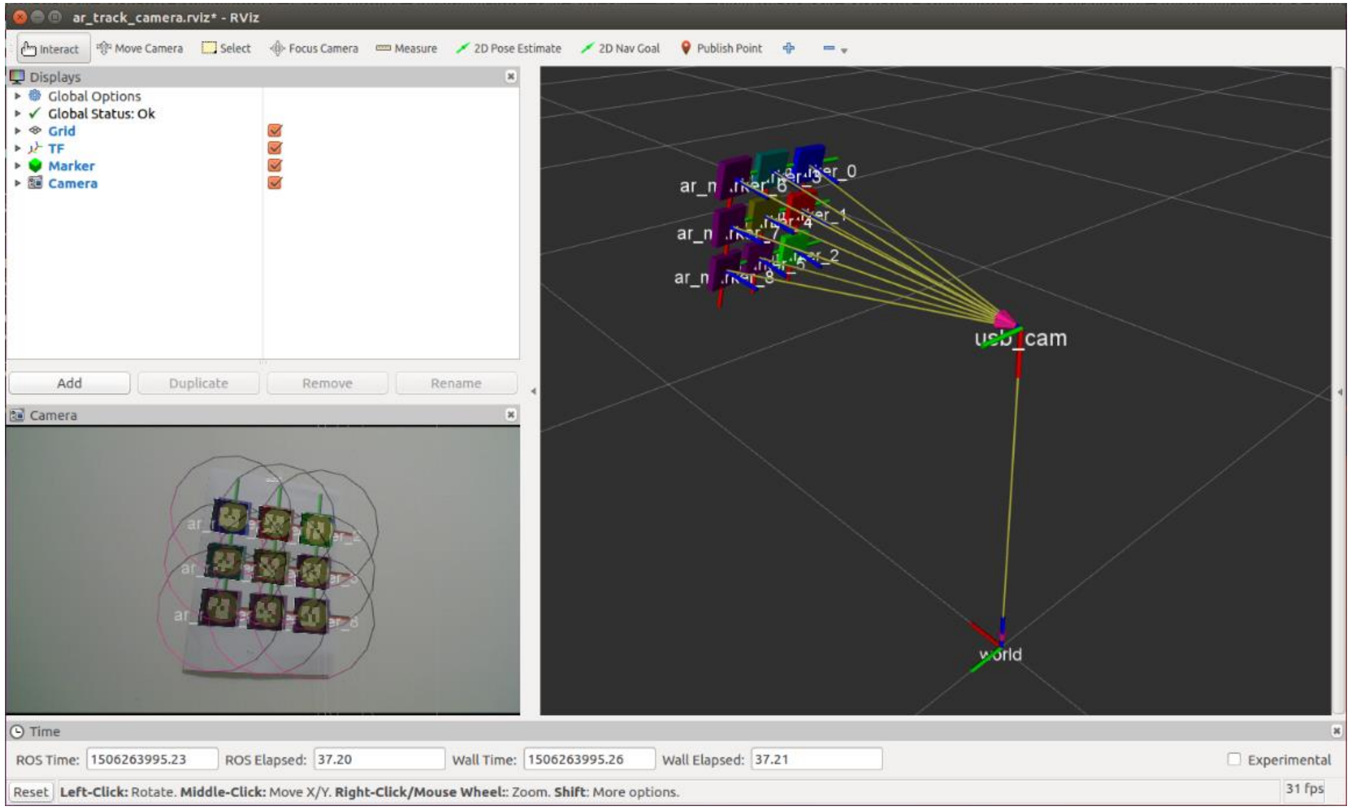
4. 二维码识别

启动摄像头二维码识别示例

$ roslaunch robot_vision usb_cam_with_calibration.launch

$ roslaunch robot_vision ar_track_camera.launch

* 启动摄像头时，需要加载标定文件，否则可能无法识别二维码。



4. 二维码识别

查看识别到的二维码位姿

rostopic echo /ar_pose_marker


```
markers:
  -
    header:
      seq: 0
      stamp:
        secs: 1506264295
        nsecs: 261290716
      frame_id: /usb_cam
    id: 5
    confidence: 0
    pose:
      header:
        seq: 0
        stamp:
          secs: 0
          nsecs:         0
        frame_id: ''
      pose:
        position:
          x: -0.050358386788
          y: -0.0286603534611
          z: 0.493313470257
        orientation:
          x: 0.98092476878
          y: 0.00292648513227
          z: 0.183482106987
          w: 0.0641276078974
```

4. 二维码识别

```xml
<launch>

    <node pkg="tf" type="static_transform_publisher" name="world_to_cam"
          args="0 0 0.5 0 1.57 0 world camera_rgb_optical_frame 10" />

    <arg name="marker_size" default="5.0" />
    <arg name="max_new_marker_error" default="0.08" />
    <arg name="max_track_error" default="0.2" />

    <arg name="cam_image_topic" default="/camera/depth_registered/points" />
    <arg name="cam_info_topic" default="/camera/rgb/camera_info" />
    <arg name="output_frame" default="/camera_rgb_optical_frame" />

    <node name="ar_track_alvar" pkg="ar_track_alvar" type="individualMarkers" respawn="false" output="screen">
        <param name="marker_size" type="double" value="$(arg marker_size)" />
        <param name="max_new_marker_error" type="double" value="$(arg max_new_marker_error)" />
        <param name="max_track_error" type="double" value="$(arg max_track_error)" />
        <param name="output_frame" type="string" value="$(arg output_frame)" />

        <remap from="camera_image"  to="$(arg cam_image_topic)" />
        <remap from="camera_info"   to="$(arg cam_info_topic)" />
    </node>

    <!-- rviz view /-->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find robot_vision)/config/ar_track_kinect.rviz"/>

</launch>
```

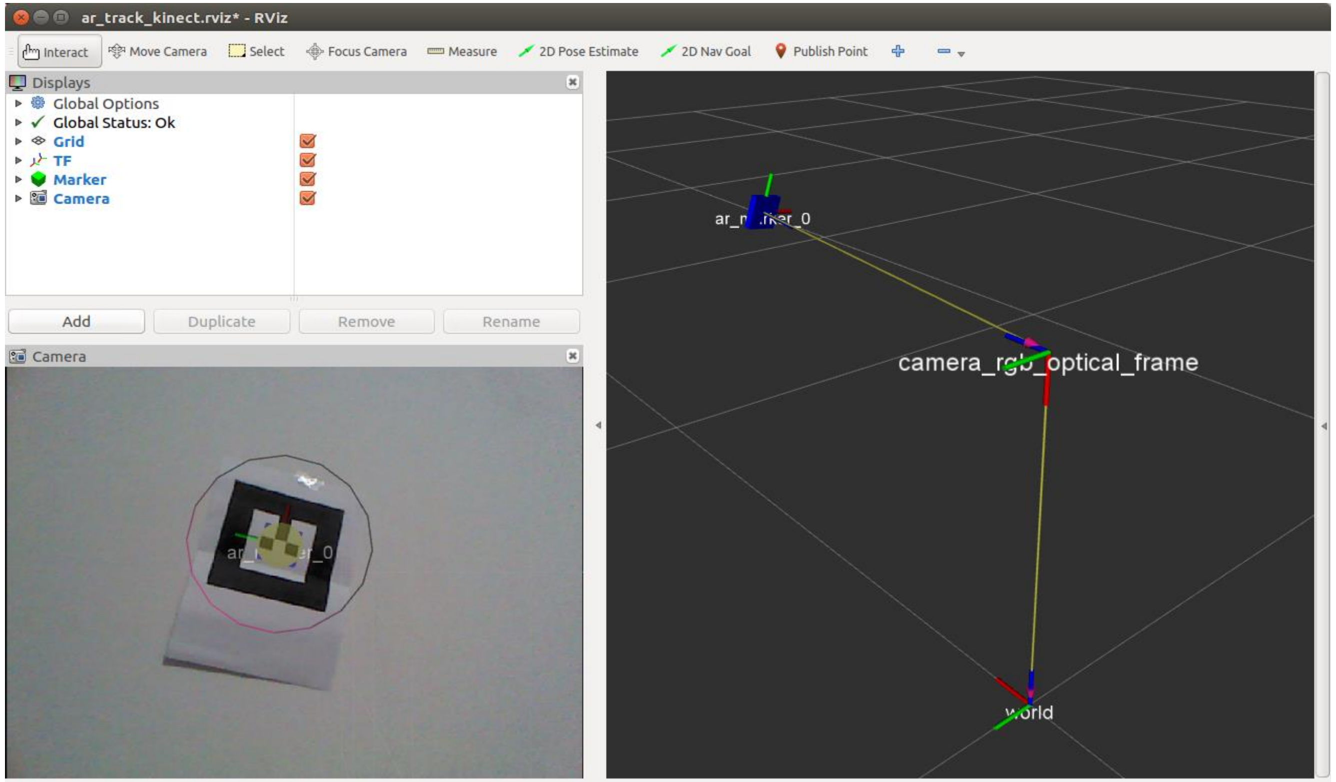
robot_vision/launch/ar_track_kinect.launch

4. 二维码识别

启动Kinect二维码识别示例

$ roslaunch robot_vision freenect.launch

$ roslaunch robot_vision ar_track_kinect.launch



4. 二维码识别

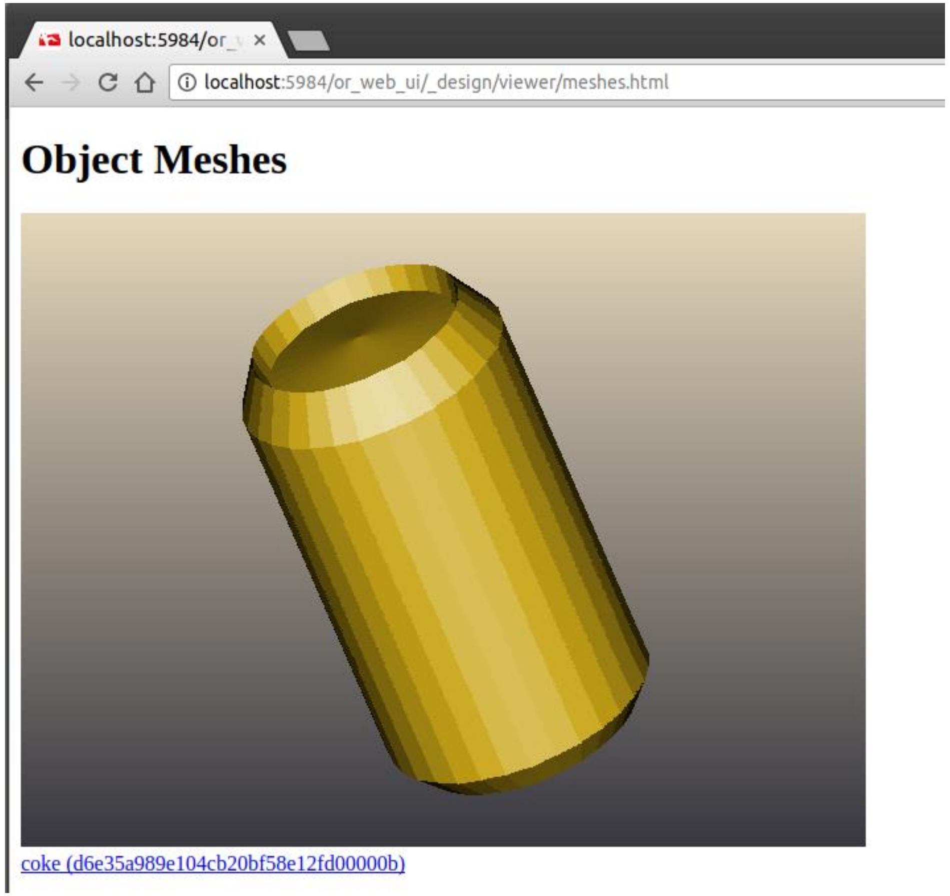

Co., Ltd.
ROBOTIS Manipulator
DYNAMIXEL
070900
ROBOTIS
DYNAMIXEL PRO
LG

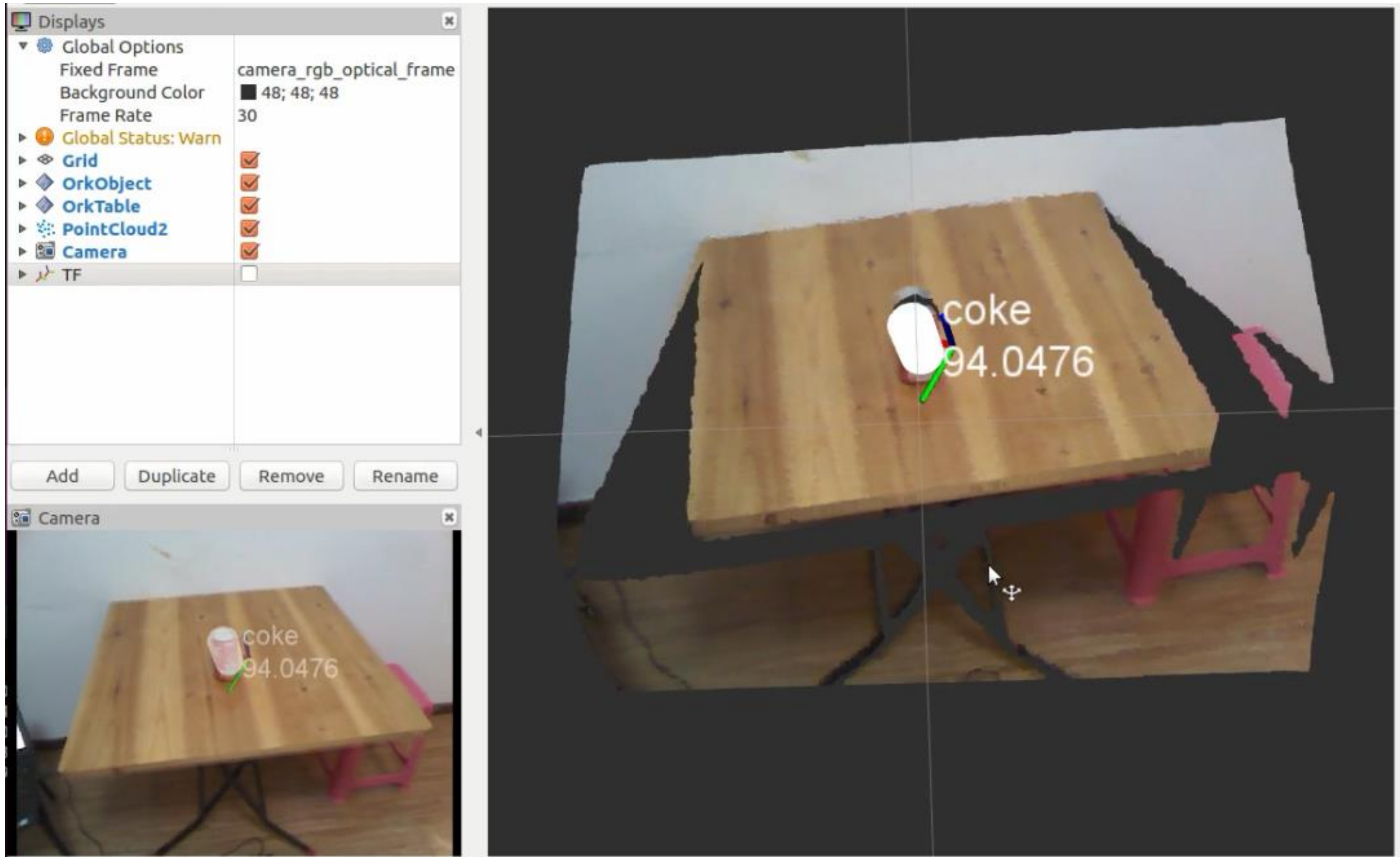
目录

➢5. 物体识别

5. 物体识别—— ORK

Object Recognition Kitchen（ORK）



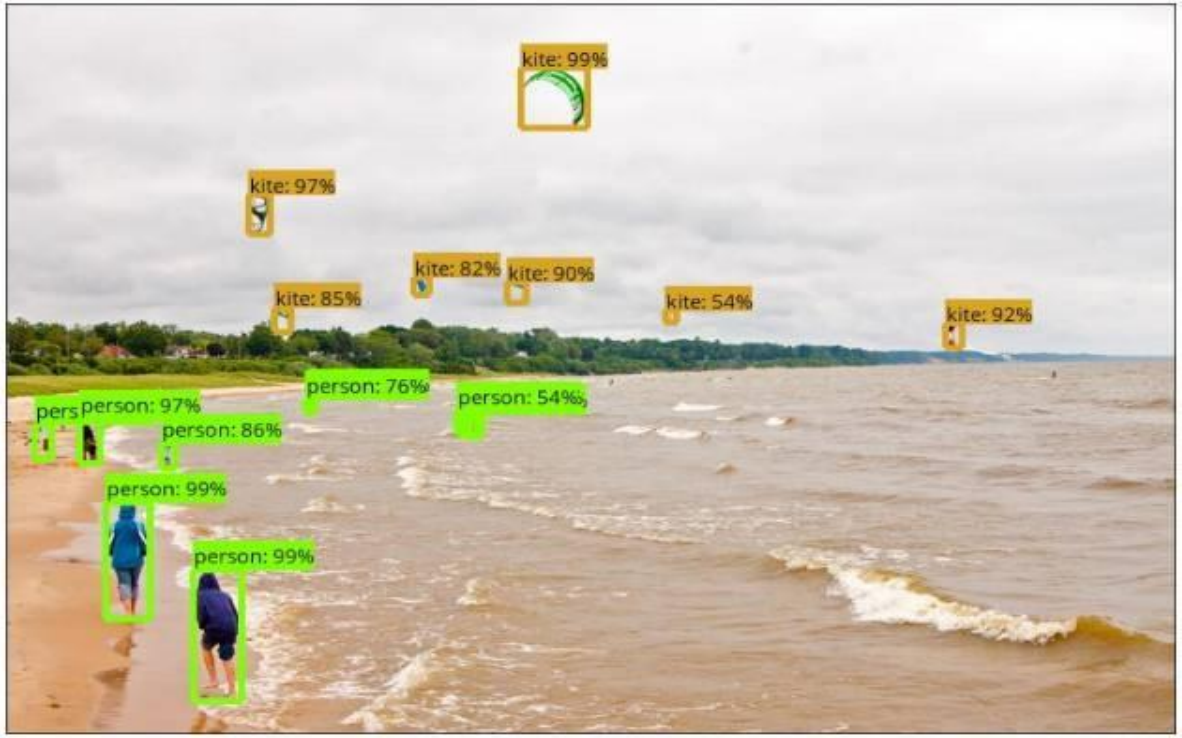


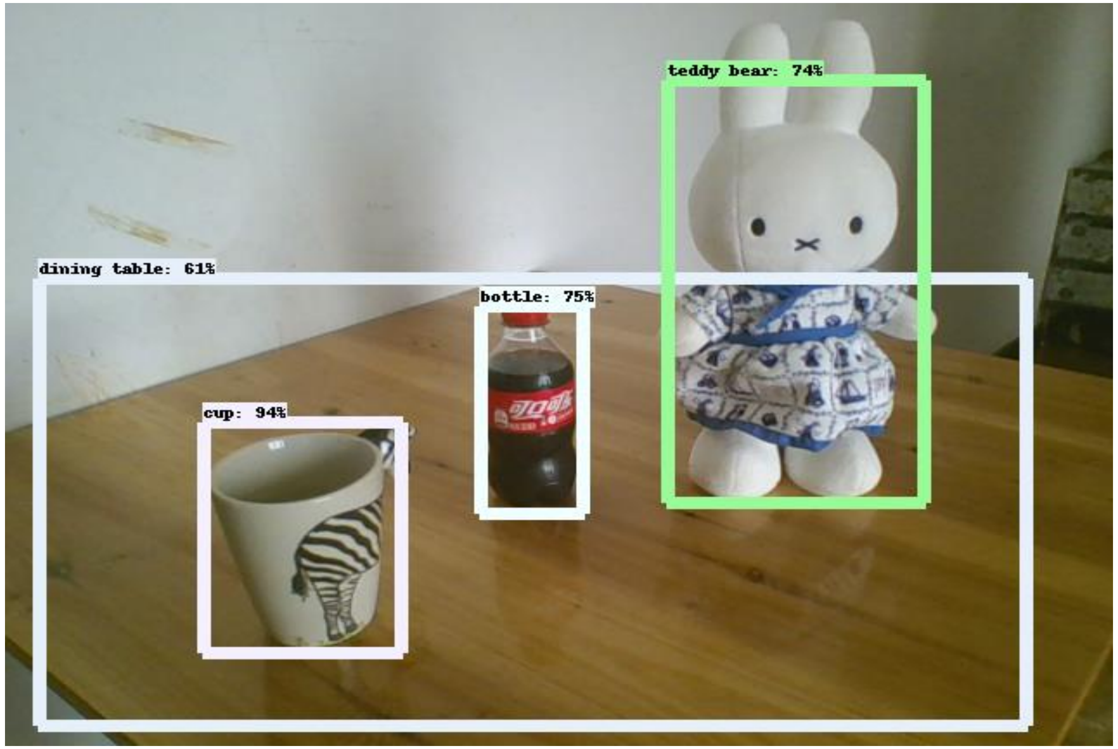
可乐罐识别

* 参考链接：http://wiki.ros.org/object_recognition

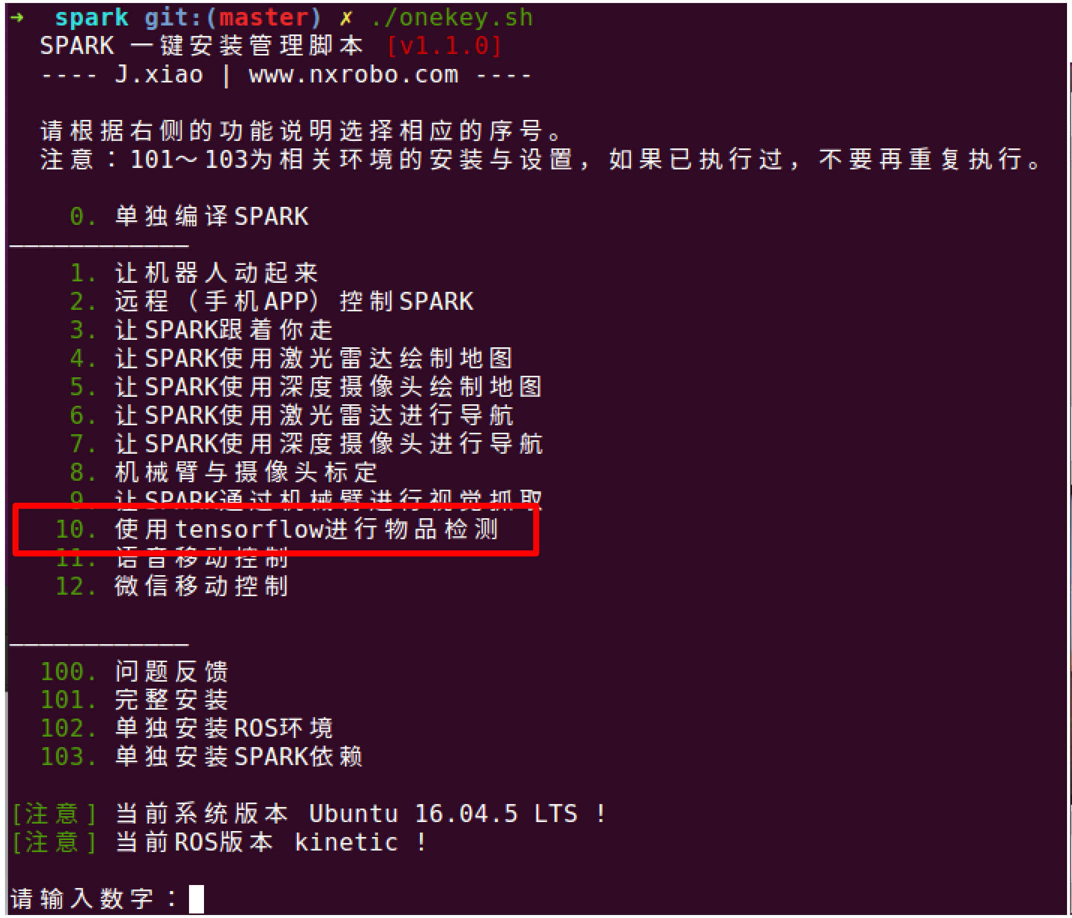
5. 物体识别 —— TensorFlow

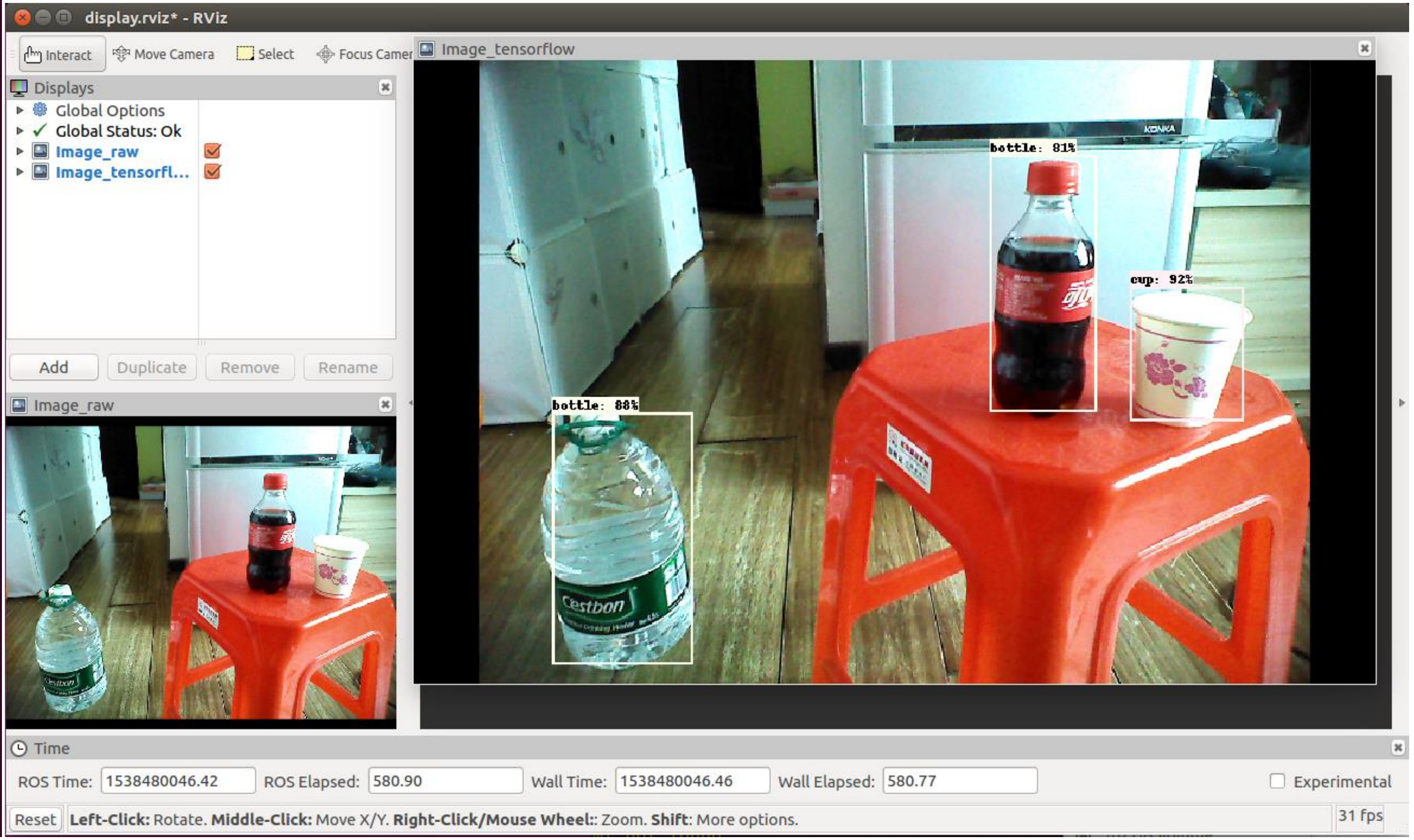
TensorFlow Object Detection API





5. 物体识别 —— TensorFlow





Spark 物体识别

Thank you!