

ROS机器人创新实践

第四讲 机器人感知

高斌

目录/Contents

01

机器视觉

02

机器语音



01

机器视觉



1.机器视觉——机器人应用案例





1.机器视觉——ROS摄像头

- 当我们把摄像头通过USB与计算机连接之后，下一步便是通过ROS来调取摄像头图像了。不过，在ROS系统中，USB摄像头并不被原生支持，我们需要自己安装驱动包来获取摄像头图像并发布出去，这样image_view包才能够获取图像并显示出来。
- 下载并编译usb_cam包：

```
$ cd ~/catkin_make/src
$ git clone https://github.com/bosch-ros-pkg/usb_cam.git
$ cd ..
$ catkin_make
```

```
spark@spark: ~/catkin_make
-- ==> add_subdirectory(usb_cam)
-- Found PkgConfig: /usr/bin/pkg-config (found version "0.29.1")
-- Checking for module 'libavcodec'
--   Found libavcodec, version 56.60.100
-- Checking for module 'libswscale'
--   Found libswscale, version 3.1.101
-- Configuring done
-- Generating done
-- Build files have been written to: /home/spark/catkin_make/build
####
#### Running command: "make -j4 -l4" in "/home/spark/catkin_make/build"
####
Scanning dependencies of target usb_cam
[ 25%] Building CXX object usb_cam/CMakeFiles/usb_cam.dir/src/usb_cam.cpp.o
[ 50%] Linking CXX shared library /home/spark/catkin_make/devel/lib/libusb_cam.so
[ 50%] Built target usb_cam
Scanning dependencies of target usb_cam_node
[ 75%] Building CXX object usb_cam/CMakeFiles/usb_cam_node.dir/nodes/usb_cam_node.cpp.o
[100%] Linking CXX executable /home/spark/catkin_make/devel/lib/usb_cam/usb_cam_node
[100%] Built target usb_cam_node
spark@spark:~/catkin_make$
```

1. 机器视觉——ROS摄像头

```
$ 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.机器视觉——ROS摄像头

Launch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="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>
```



1.机器视觉——ROS摄像头

获取图像

- 可以看到，脚本中开启了两个节点，一个是usb_cam自身的节点，它附带了几个参数以确定图像的来源及格式；另一个是image_view节点，用于显示获取到的图像数据。其中，video_device设定图像来源的设备，在Ubuntu中，可以通过以下命令查看当前系统中可用的视频设备：

```
$ ls /dev/ |grep video
```

```
spark@spark:~/catkin_make$ ls /dev/ |grep video
video0
video1
video2
spark@spark:~/catkin_make$
```




1.机器视觉——ROS摄像头

ROS中的图像数据（二维）

- 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

```
→ ~ rosmg 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.机器视觉——ROS摄像头

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.机器视觉——ROS摄像头

RGBD

- 除了常见的色彩摄像头之外，还有深度摄像头也是机器人常用的一种传感器，它能在一定的范围内探测前方物体距离摄像头的距离，例如：**Kinect**、**Xtion**、**realsense**等。使用的测距方法也主要是时间飞行（**ToF**）及结构光等，大家可自行了解下，这里不做过多讨论。
- 以我们**spark**自带的深度摄像头作为讲解，当插入摄像头，应可以看到ID为**2b05:0401**的设备。

```
spark@spark:~/spark$ lsusb
Bus 001 Device 002: ID 8087:07e6 Intel Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 002 Device 006: ID 2bc5:0401
Bus 002 Device 005: ID 1a86:7523 QinHeng Electronics HL-340 USB-Serial adapter
Bus 002 Device 003: ID 05e3:0608 Genesys Logic, Inc. Hub
Bus 002 Device 004: ID 8087:07dc Intel Corp.
Bus 002 Device 002: ID 05e3:0608 Genesys Logic, Inc. Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```



1.机器视觉——ROS摄像头

连接realsense

首先下载librealsense,有条件在[GitHub](#)下载,快速下载用[Gitee](#)
在/home下进入librealsense,并安装依赖项

➤ 安装SDK

```
git clone -b v2.50.0 https://github.com/IntelRealSense/librealsense.git
sudo apt-get install libudev-dev pkg-config libgtk-3-dev
sudo apt-get install libusb-1.0-o-dev pkg-config
sudo apt-get install libglfw3-dev
sudo apt-get install libssl-dev

./scripts/setup_udev_rules.sh
./scripts/patch-realsense-ubuntu-lts.sh
mkdir build
cd build
cmake ../ -DCMAKE_BUILD_TYPE=Release -DBUILD_EXAMPLES= true
sudo make
sudo make install
```





1.机器视觉——ROS摄像头

连接realsense

- ROS功能包安装
- realsense-ros 要和 librealsense 版本匹配，比如 realsense-ros 4.03支持的 Realsense SDK 为 librealsense 2.50 .0

```
cd ~/catkin_ws/src  
git clone https://github.com/IntelRealSense/realsense-ros.git  
cd realsense-ros/realsense2_camera git checkout `git tag | sort -  
V | grep -P "\d+\.\d+\.\d+" | tail -1`  
sudo apt-get install ros-neotic-ddynamic-reconfigure  
cd ~/catkin_ws && catkin_make
```

ROS Wrapper for Intel® RealSense™ Devices

These are packages for using Intel RealSense cameras (D400 series SR300 camera and T265 Tracking Module) with ROS.

This version supports Kinetic, Melodic and Noetic distributions.

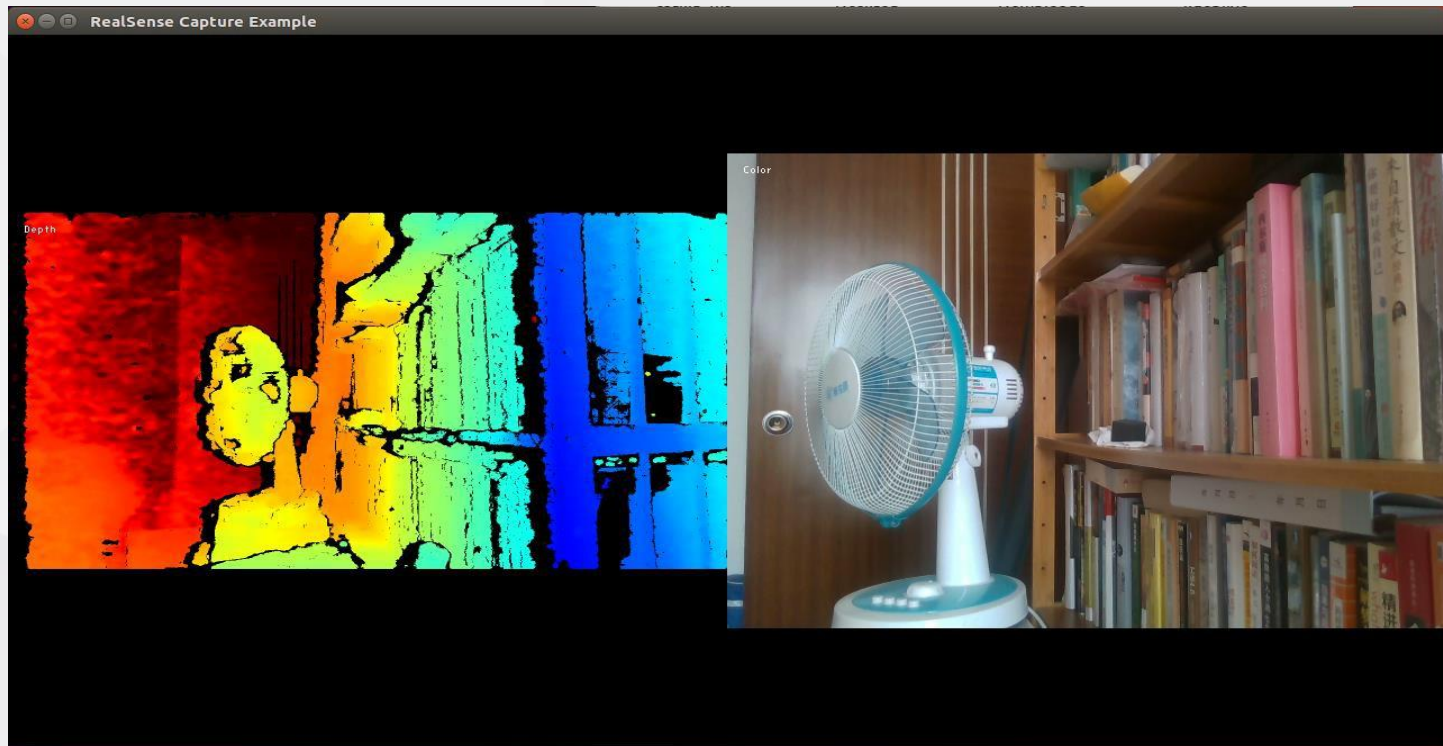
For running in ROS2 environment please switch to the [ros2 branch](#).

LibRealSense2 supported version: v2.50.0 (see [realsense2_camera release notes](#))



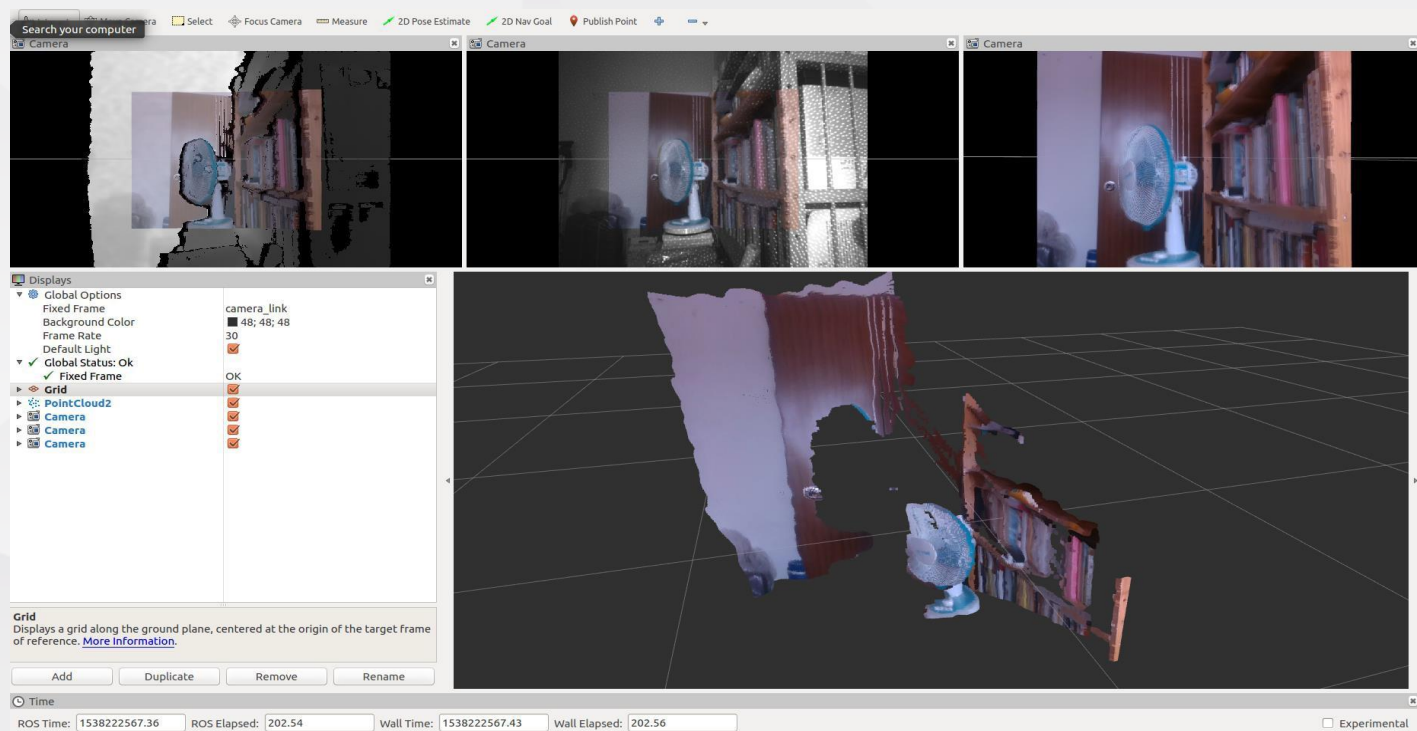


1.机器视觉——ROS摄像头



Realsense SDK example

1. 机器视觉——ROS摄像头



点云显示

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




1.机器视觉——ROS摄像头

ROS中的图像数据（三维）

- 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。

```
→ ~ rosmg 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
```

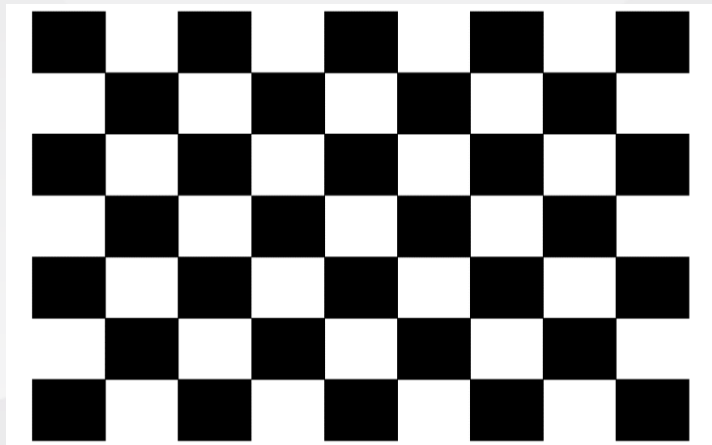
点云单帧数据量也很大, 如果使用分布式网络传输, 需要考虑能否满足数据的传输要求, 或者针对数据进行压缩。



1.机器视觉——摄像头内参标定

摄像头为什么要标定？

- 由于大部分摄像头经过镜头获取图像之后会产生失真，对我们后期的图像处理造成误差，所以，我们需要首先对此摄像头进行标定，以此获取摄像头内参。标定方法主要是使用一种已知的标定图案，通过对不同角度的视图进行辨识来实现。在这里我们使用经典的棋盘图案来做标定，当然也有其他的标定图案，不过不在我们本次的讨论范围，大家有兴趣可自行查询。



棋盘格标定靶



1.机器视觉——摄像头内参标定

- 我们使用`camera_calibration`包进行标定，不过在标定之前我们需要打开摄像头并将图像发布出来，并不需要开启`image_view`包来显示图像，所以，我们需要精简上一步所使用的`launch`文件，去掉开启`image_view`包的脚本。

```
<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>
</launch>
```



1.机器视觉——摄像头内参标定

摄像头内参标定流程

➤ 启动摄像头

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

➤ 启动标定包

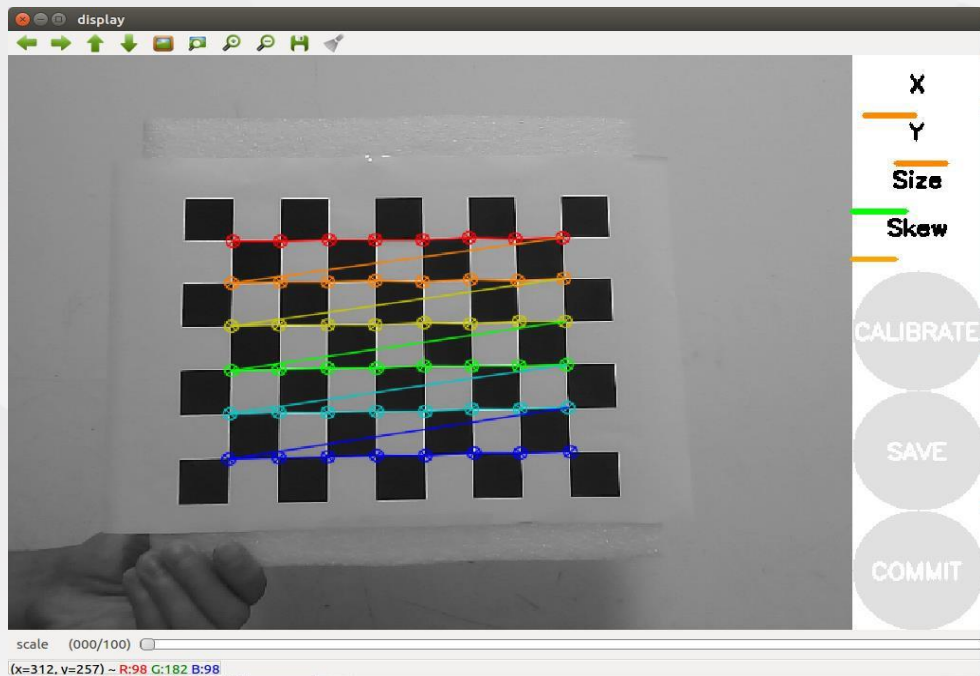
```
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.024  
image:=/usb_cam/image_raw camera:=/usb_cam
```

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。



1.机器视觉——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



1.机器视觉——摄像头内参标定

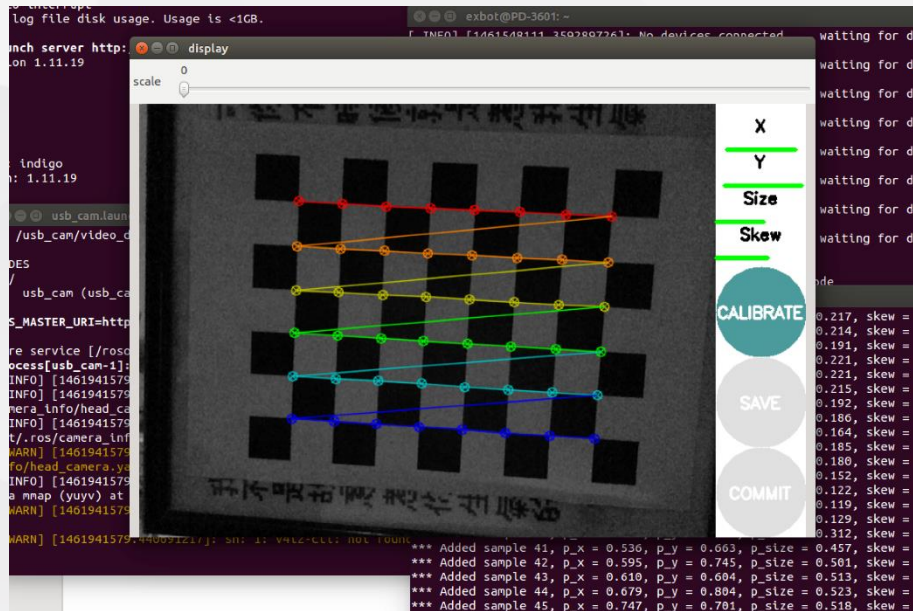
- 界面正常出现后，我们需要手持棋盘板在不同的距离不同的角度位置让程序识别棋盘，所以需要确保棋盘全部保持出现在图像中。大致的几个位置如下图（转自官网）显示：





1.机器视觉——摄像头内参标定

- 在每个不同位置保持片刻，等棋盘被彩色高亮标示后即可移动。在移动过程中可以看到窗口右上角的几个进度条在增长，同时颜色也渐渐趋向绿色。当右侧的calibration按钮亮起之后，表示我们已经采集够了标定所需数据，点击按钮便会自动计算并显示结果。





1.机器视觉——摄像头内参标定

- 关于ROS camera driver中最难以理解的一点 camera_info:
- 首先我们需要了解camera本身的模型参数，具体分为内参与外参，camera还存在不同程度的畸变，描述这些畸变也有对应的模型及其参数。对于双目相机来说，相机成像平面与等本征矩阵参数也需要记录。
- 这些信息被ROS统一整合到了一起，放在camera_info里面与image资料一起发布。
- 最让人难以上手ROS图像处理这一整套方案的地方在于，在ROS系统体系中，一般默认camera driver负责管理这部分信息，但是大部分开源出来的ROS camera driver并没有对这部分做很好的处理，原因很简单，因为硬件种类太多了，即使是硬件厂家如Pointgrey 的官方ROS驱动， camera_info 也需要我们进行手动测定。



1.机器视觉——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



1.机器视觉——ROS+OpenCV应用

OpenCV是什么?

- Open Source ComputerVision 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。

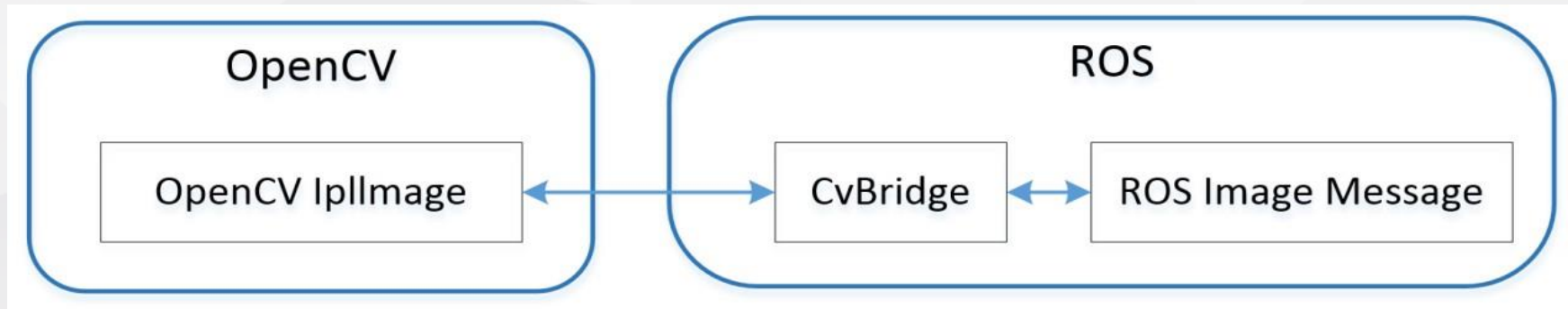




1. 机器视觉——ROS+OpenCV应用

安装OpenCV

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



ROS与OpenCV的集成框架



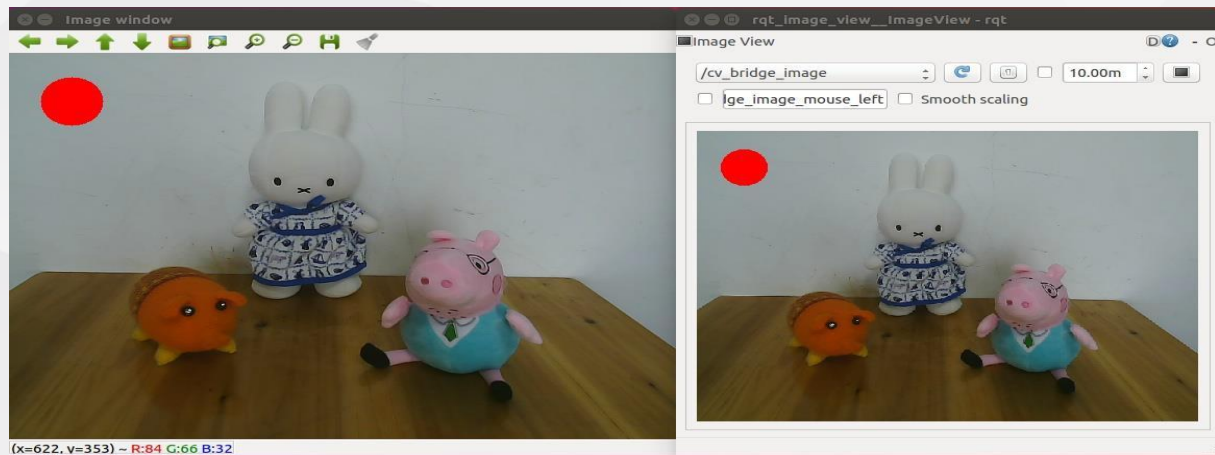
1.机器视觉——ROS+OpenCV应用

cvbridge测试例程

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

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

```
$ rqt_image_view
```





1. 机器视觉——ROS+OpenCV应用

➤ **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()
```



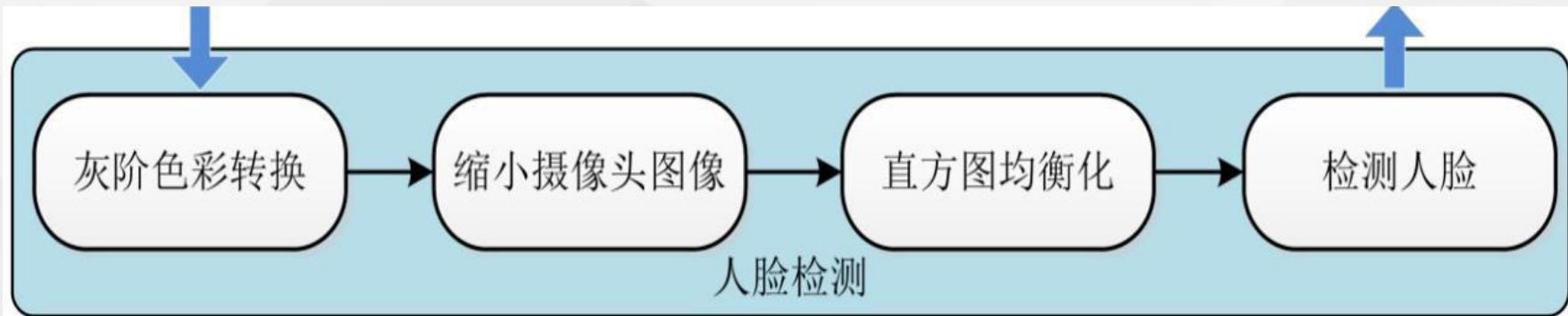
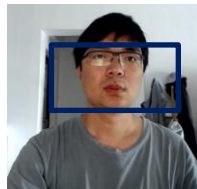
1. 机器视觉——ROS+OpenCV应用

人脸识别

图像输入



图像输出



基于Haar特征的级联分类器对象检测算法



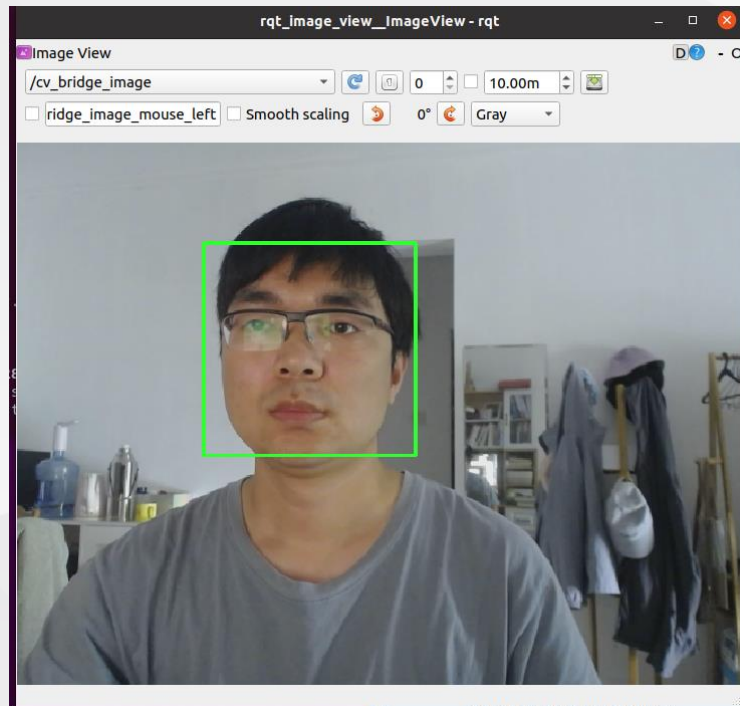
1.机器视觉——ROS+OpenCV应用

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



1.机器视觉——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
```




1.机器视觉——二维码识别



安装二维码识别功能包

\$在ws/src目录下

```
git clone https://github.com/machinekoder/ar_track_alvar.git -b noetic-devel
```

```
# cd ..
```

```
catkin_make -DCMAKE_BUILD_TYPE=Release
```



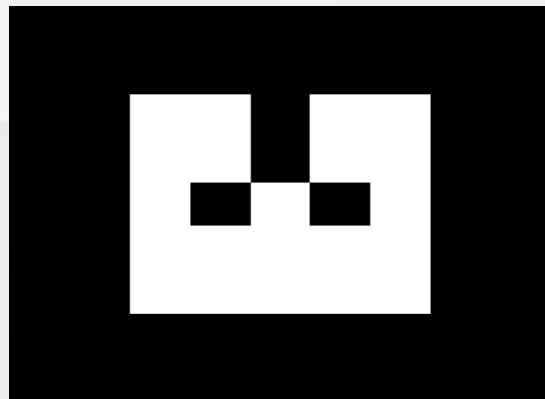

1. 机器视觉——二维码识别

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
-l "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
```





1. 机器视觉——二维码识别

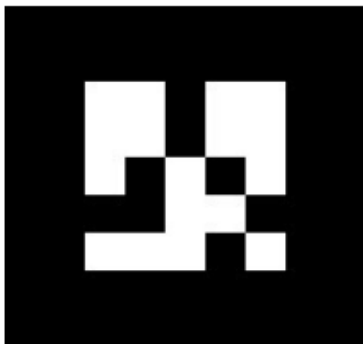
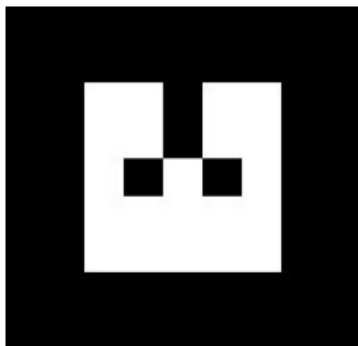
创建二维码

```
$ roscd robot_vision/config
```

```
$ rosrun ar_track_alvar createMarker -s 5 0
```

```
$ rosrun ar_track_alvar createMarker -s 5 1
```

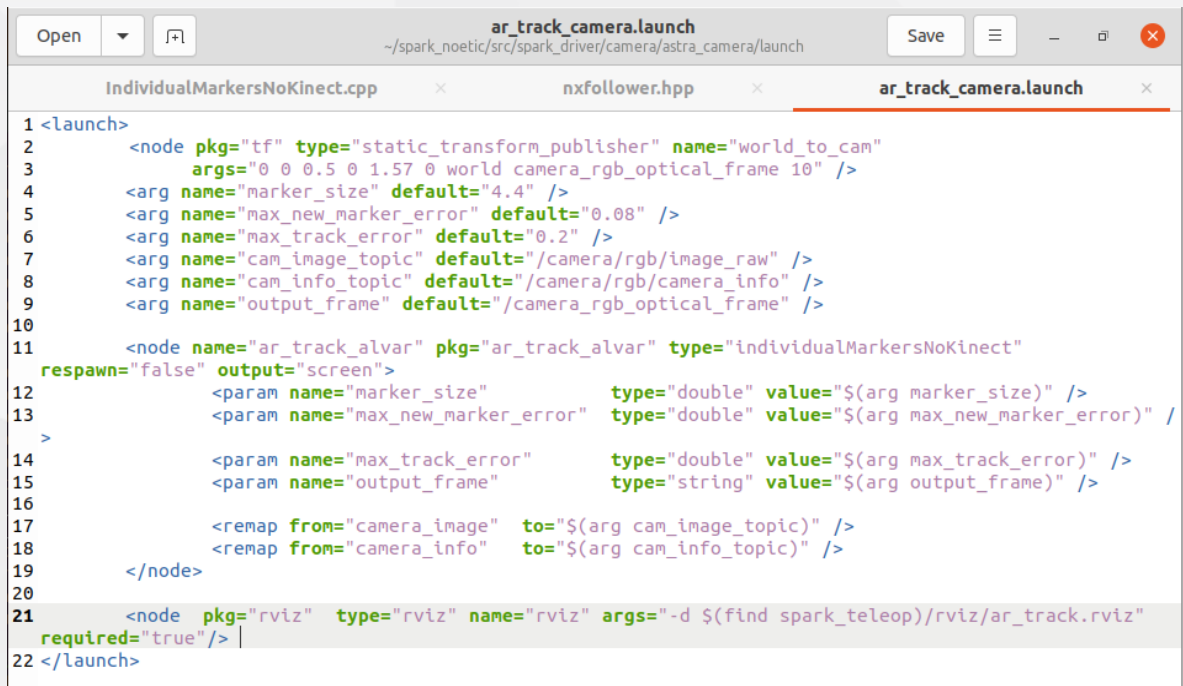
```
$ rosrun ar_track_alvar createMarker -s 5 2
```





1.机器视觉——二维码识别

ar-track-alvar功能包支持USB摄像头或RGB-D摄像头作为识别二维码的视觉传感器，分别对应于individualMarkersNoKinect和individualMarkers这两个不同的识别节点。复制ar-track-alvar功能包launch文件夹中的pr2_indiv_no_kinect.launch文件作为蓝本，针对使用的RGB-D摄像头进行修改设置，重命名为camera_driver/camera/camera_driver_transfer/launch/ar_track_camera.launch:



```
1 <launch>
2   <node pkg="tf" type="static_transform_publisher" name="world_to_cam"
3     args="0 0 0.5 0 1.57 0 world camera_rgb_optical_frame 10" />
4   <arg name="marker_size" default="4.4" />
5   <arg name="max_new_marker_error" default="0.08" />
6   <arg name="max_track_error" default="0.2" />
7   <arg name="cam_image_topic" default="/camera/rgb/image_raw" />
8   <arg name="cam_info_topic" default="/camera/rgb/camera_info" />
9   <arg name="output_frame" default="/camera_rgb_optical_frame" />
10
11   <node name="ar_track_alvar" pkg="ar_track_alvar" type="individualMarkersNoKinect"
12     respawn="false" output="screen">
13     <param name="marker_size" type="double" value="$(arg marker_size)" />
14     <param name="max_new_marker_error" type="double" value="$(arg max_new_marker_error)" />
15     <param name="max_track_error" type="double" value="$(arg max_track_error)" />
16     <param name="output_frame" type="string" value="$(arg output_frame)" />
17     <remap from="camera_image" to="$(arg cam_image_topic)" />
18     <remap from="camera_info" to="$(arg cam_info_topic)" />
19   </node>
20
21   <node pkg="rviz" type="rviz" name="rviz" args="-d $(find spark_teleop)/rviz/ar_track.rviz"
22     required="true"/>
23 </launch>
```

1.机器视觉——二维码识别

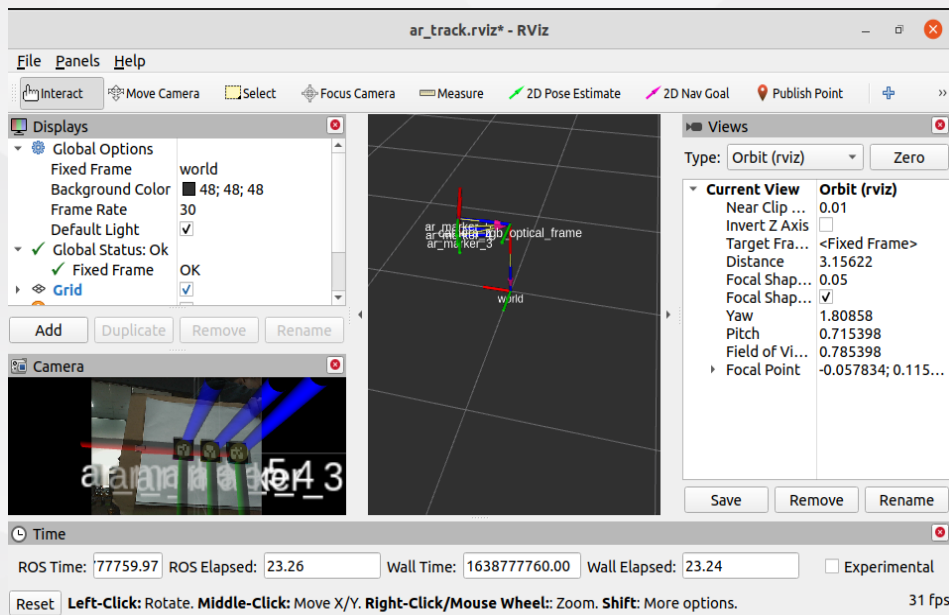
启动摄像头二维码识别示例

```
$ roslaunch robot_vision usb_cam_with_calibration.launch
```

```
$ roslaunch robot_vision ar_track_camera.launch
```

在显示的图像中，可以看到多个二维码被同时准确识别出来，图像中的二维码上会出现坐标轴，代表识别到的二维码姿态。

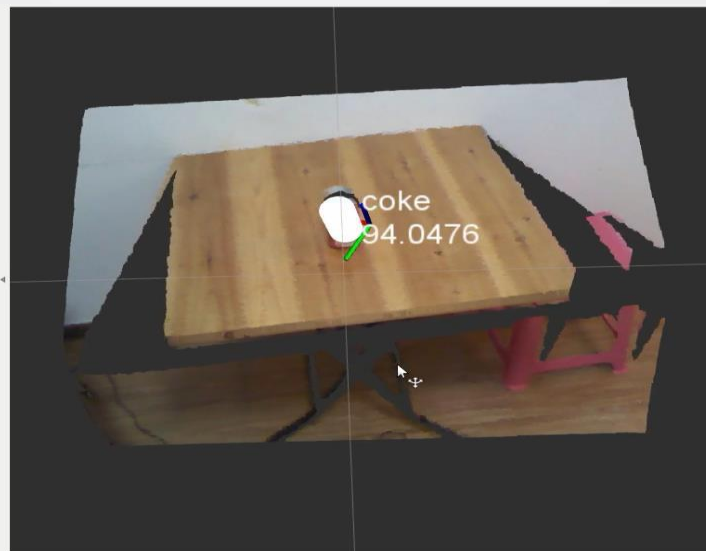
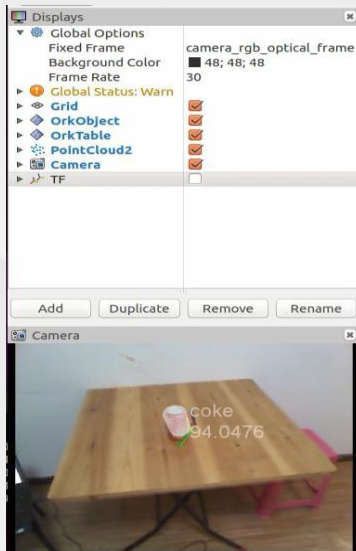
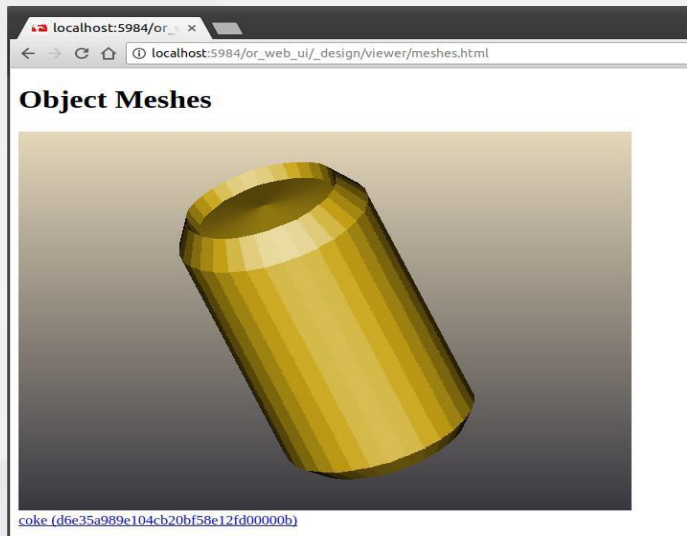
ar_track_alvar功能包不仅可以识别图像中的二维码，而且可以确定二维码的空间位姿。在使用摄像头的情况下，因为二维码尺寸已知，所以根据图像变化可以计算二维码的姿态，还可以计算二维码相对摄像头的空间位置。





1.机器视觉——物体识别

Object Recognition Kitchen (ORK)

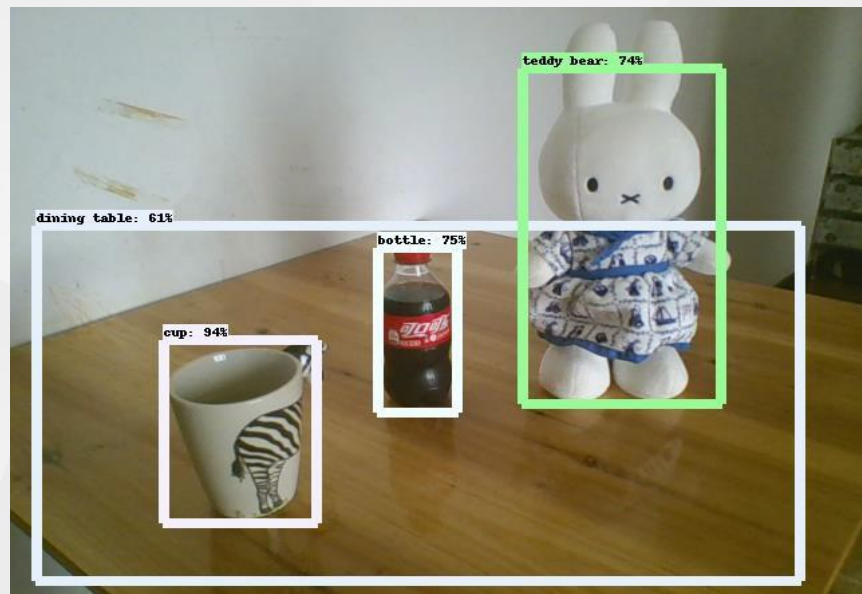


可乐罐识别



1.机器视觉——二维码识别

TensorFlow Object Detection API



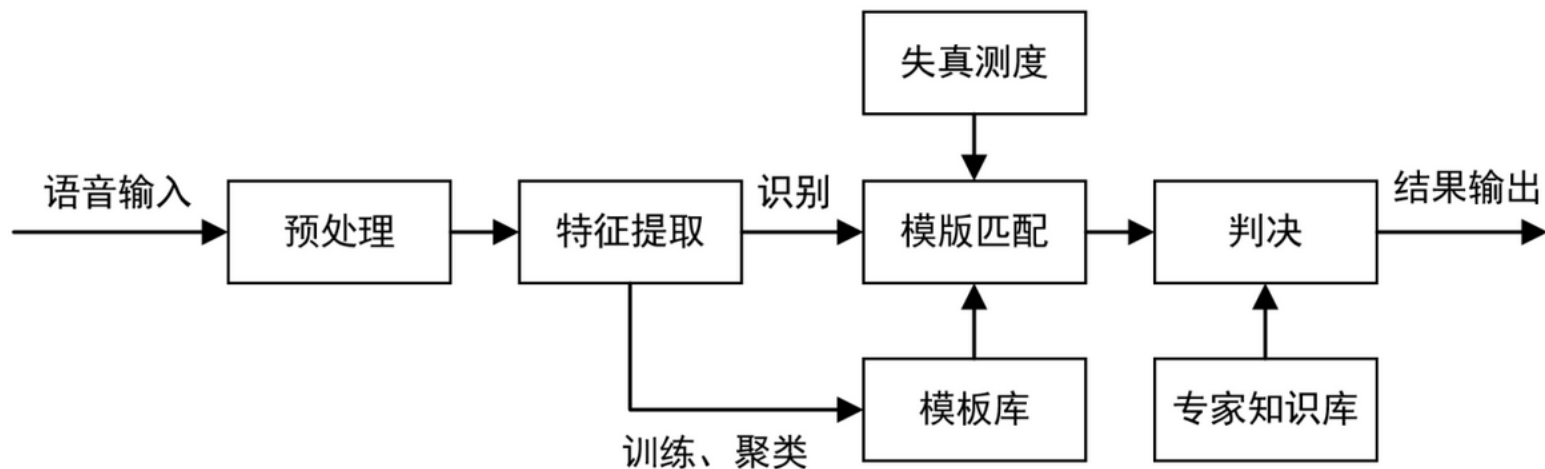


02

机器语音



2.机器语音



语音识别的理论模型



2.机器语音

常用语音功能包

- Pocketsphinx**: 集成CMU Sphinx和Festival开源项目中的代码，实现语音识别的功能
- audio-common**: 提供了文本转语音的功能实现完成"机器人说话"的想法
- AIML**: 人工智能标记语音，Artificial Intelligence Markup Language是一种创建自然语音软件代理的XML语言



THANKS
