实验一.基于颜色特征的目标识别与追踪实验

一. 搭建实验平台

- 实验环境：Ubuntu14.04
- 硬件设备：计算机、计算机自带摄像头
- 软件相关库：OpenCV-2.4.13
- 测试方式：打开终端，在video.py所在目录下输入命令`python ./video.py`,弹出摄像头画面，证明安装成功。
- 注意事项：在安装OpenCV之前最好安装ROS，不然很多依赖包可能无法正确安装而导致错误。

二. 理解实验代码

1.实验源码text.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
import cv2
import video

# 定义常量，生成颜色模板时需要用到
LOWER_BLUE = np.array([0., 60., 32.])  # 蓝色的下界
UPPER_BLUE = np.array([180., 255., 255.])  # 蓝色的上界



class App(object):
    def __init__(self, color):
        self.cam = video.create_capture(0)  # 捕获摄像头设备并创建一个对象
        self.frame = None
        cv2.namedWindow('camshift')  # cv2窗口对话框名称
        self.hist_roi = None
        self.selection = None
        self.tracking_state = 0
        self.hide_background = False  # 是否需要隐藏背景，默认显示

        if color == 'blue':
            self.flag = 'blue'
            self.roi = cv2.imread('blue.jpg')  # 读取blue.jpg作为感兴趣的区域

    # 初始化状态参数
    def start(self):
        self.selection = (0, 0, 640, 480)  # 选取该区域作为颜色识别检测区域
        self.tracking_state = 1  # 是否需要跟踪检测
```

```
    def get_mask(self, hsv_image, color='blue'):
        # 获得hsv_image对应颜色的蒙板
        if color not in ['blue', 'green', 'red']:
            return cv2.inRange(hsv_image, np.array([0., 0., 0.]),
                               np.array([255., 255., 255.]))
        elif color == 'blue':
            return cv2.inRange(hsv_image, LOWER_BLUE, UPPER_BLUE)


    def show_hist(self):
        # 展示图片的直方图
        bin_count = self.hist_roi.shape[0]
        bin_w = 24
        img = np.zeros((256, bin_count * bin_w, 3), np.uint8)
        for i in xrange(bin_count):
            h = int(self.hist_roi[i])
            cv2.rectangle(img, (i * bin_w + 2, 255), ((i + 1) * bin_w - 2, 255 - h),
                          (int(180.0 * i / bin_count), 255, 255), -1)
        img = cv2.cvtColor(img, cv2.COLOR_HSV2BGR)
        cv2.imshow('hist', img)

    def run(self):
        roi = self.roi  # 获取ROI
        self.start()
        while True:
            ret, self.frame = self.cam.read()  # 调用摄像头读取图像
            vis = self.frame.copy()  # 创建相机图像副本

            # 将当前帧从RGB格式转换为HSV格式
            hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV)

            # 获得当前hsv图像的掩模
            mask = self.get_mask(hsv, color=self.flag)

            if self.selection:
                x0, y0, x1, y1 = self.selection
                self.track_window = (x0, y0, x1, y1)  # 追踪子区域

                # 对ROI进行颜色格式转换和阈值限制
                hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
                mask_roi = self.get_mask(hsv_roi, color=self.flag)
                # 绘制ROI图像的一维直方图
                hist_roi = cv2.calcHist([hsv_roi], [0], mask_roi, [16], [0, 180])
                # 对hist做归一化
                cv2.normalize(hist_roi, hist_roi, 0, 255, cv2.NORM_MINMAX)

                # 将hist向量reshape为1列并存入self.hist中
                self.hist_roi = hist_roi.reshape(-1)
                self.show_hist()

                # 可见区域
                # 目标图的待搜索区域;注意此时x,y的顺序
```

```
                # 像素坐标系:x轴向右,y轴向下,即x轴对应图像宽度,有多少列像素,y轴对应图像高度，代表行
                vis_roi = vis[y0:y1, x0:x1]
                cv2.bitwise_not(vis_roi, vis_roi)  # 对每个像素进行二进制取反操作

                # 在vis中，置mask中为0的对应位置也为0
                vis[mask == 0] = 0

            if self.tracking_state == 1:
                self.selection = None  # 取消ROI模板

                # 反向投影法
                prob = cv2.calcBackProject([hsv], [0], self.hist_roi, [0, 180], 1)
                prob &= mask  # 与mask进行与运算 得到所求颜色的直方图概率分布

                # CamShift算法迭代终止条件：达到最大迭代次数或者达到收敛阈值
                criteria_term = (cv2.TERM_CRITERIA_EPS|cv2.TERM_CRITERIA_COUNT, 10, 1)

                # camshift算法根据反向投影图计算目标颜色的质心，实现对目标颜色的跟踪；
                # 同时返回搜索窗信息，用于下一次迭代中调整搜索窗的位置和大小
                # track_box存储搜索窗的状态信息(圆心坐标，长/短轴，角度)
                track_box, self.track_window = cv2.CamShift(prob, self.track_window,
criteria_term)

                if track_box[1][1] <= 1:
                    # 如果没有检测到 重置检测状态
                    self.start()
                else:
                    # 检测到目标颜色
                    if self.hide_background:
                        # 如果需要隐藏背景， 使用prob直方概率分布图替换vis图像
                        vis[:] = prob[..., np.newaxis]
                    try:
                        '''
                        track_box: [[center, axes], [angle, startAngle], endAngle]
                        '''
                        # 在track_box内部绘制椭圆图像
                        # 设置搜索窗的属性：(0,255,255)在BGR空间中为黄色,2为椭圆线圈像素宽度
                        a = str(track_box[0][0])+" "+str(track_box[0][1])+
                            " "+str(round(track_box[1][0],2))\+
                            " "+str(round(track_box[1][1],2))+
                            " "+str(round(track_box[2],2))+"\r\n"
                        print a
                    except:
                        print track_box

            cv2.imshow('camshift', vis)

            ch = 0xFF & cv2.waitKey(5)  # 保留返回值的低8位
            if ch == 27 or ch == ord('q'):  # 27对应ESC，即按ESC键退出
                break
            if ch == ord('b'):  # 输入b改变是否需要显示背景
                self.hide_background = not self.hide_background
            if ch == ord('r'):   # 重新开始检测颜色
```

```
                self.start()
        cv2.destroyAllWindows()


if __name__ == '__main__':
    import sys

    try:
        print sys.argv[1]
        color = sys.argv[1]
    except IndexError:
        # 命令行参数未指定检测的颜色
        color = 'blue'

    a = App(color)
    a.run()
```

2.分析实验源码

（1）init初始化函数分析

```
def __init__(self, color):
        self.cam = video.create_capture(0)  # 捕获摄像头设备并创建一个对象
        self.frame = None  # 定义帧
        cv2.namedWindow('camshift')  # cv2窗口对话框名称
        self.hist_roi = None
        self.selection = None
        self.tracking_state = 0
        self.hide_background = False  # 是否需要隐藏背景，默认显示

        if color == 'red':
            self.flag = 'red'
            self.roi = cv2.imread('red.jpg')  # 读取red.jpg作为感兴趣的区域
        elif color == 'green':
            self.flag = 'green'
            self.roi = cv2.imread('green.jpg')  # 读取green.jpg作为感兴趣的区域
        else:
            self.flag = 'blue'
            self.roi = cv2.imread('blue.jpg')  # 读取blue.jpg作为感兴趣的区域
```

> init函数是在执行`a = App(color)`时自动调用的初始化函数，主要对一些参数的初始化以及图像的读取，本实验用的时蓝色图像，所以color='blue'。通过cv的imread函数把blue.jpg图像读取进来并存放到roi里面，也就是roi存放的是例图（感兴趣的区域）。

（2）run函数分析

- 获取roi图像，`roi = self.roi`
- 调用start函数，`self.start()`
- 调用相机图像并创建副本。

```
ret, self.frame = self.cam.read()  # 调用摄像头读取图像
vis = self.frame.copy()  # 创建相机图像副本
```

- 把相机RGB图像转换成HSV格式并获取掩模。mask 是对目标图像进行计算的掩模，这个掩模被设定为蓝色，因为我们需要识别的区域被设定为蓝色，所以掩模的上下界都被设定为蓝色的上下界。

```
hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV) # 将当前帧从RGB格式转换为HSV格式
mask = self.get_mask(hsv, color=self.flag) # 获得当前hsv图像的掩模
```

- 把例图roi转换成HSV格式和取掩模值。

```
hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)  # 将例图从RGB格式转换为HSV格式
mask_roi = self.get_mask(hsv_roi, color=self.flag)  # 获得当前例图的hsv图像的掩模
```

- 绘制roi图像的一维直方图

```
hist_roi = cv2.calcHist([hsv_roi], [0], mask_roi, [16], [0, 180])
```

- 对hist做归一化

```
cv2.normalize(hist_roi, hist_roi, 0, 255, cv2.NORM_MINMAX)
```

- 将hist向量reshape为1列并存入self.hist中,并在屏幕中显示出来

```
self.hist_roi = hist_roi.reshape(-1)
self.show_hist()
```

- 反向投影法

```
prob = cv2.calcBackProject([hsv], [0], self.hist_roi, [0, 180], 1)  # 反向投影法
prob &= mask  # 与mask进行与运算,得到所求颜色的直方图概率分布
```

> 反向投影的理解：
>
> 直方图反向投影通常应用在对比示例图像和目标图像中，对比实际上是提取图像中的某个特征，提取特征的过程就是反向投影的过程。首先把两张图像的亮度值0-255划分区间(注意模型和目标的直方图划分区间要相同)，找到目标图像的像素值所在的区间，再去模型直方图的对应区间找到BIN的值，用该值代替目标图像对应区间内的所有像素值。最后得到的图像矩阵和原矩阵大小相同，但其中的值种类更少。由反向投影的过程可以看出，若在原图像的直方图区间内点越多，替换的像素值就越大，而越大的值在亮度等级上越高，表现在反向投影矩阵中就越亮。最后的结果是灰度图像，需要寻找的特征区域在反向投影图中越亮，其他区域越暗。

- 运用CamShift算法对追踪区域内的图像进行prob检测

```
# CamShift算法的停止条件
criteria_term = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)

# camshift算法根据反向投影图计算目标颜色的质心，实现对目标颜色的跟踪；
# 同时返回搜索窗信息，用于下一次迭代中调整搜索窗的位置和大小
# track_box存储搜索窗的状态信息(圆心坐标，长/短轴，角度)
track_box, self.track_window = cv2.CamShift(prob, self.track_window, criteria_term)
```

> **CamShift算法的理解：**
>
> CamShift算法将meanshift算法扩展到连续图像序列，其将视频的所有帧做meanshift运算，并将上一帧的结果（搜索窗的大小和重心），作为下一帧meanshift算法搜索窗的初始值，不断迭代此过程，最终实现对目标的跟踪。
>
> 本实验中首先使用例图的直方图对视频图像做反向投影，得到颜色概率分布图。然后使用CamShift算法根据得到的向量控制搜索框向目标区域移动。每一帧根据上一帧的结果动态调整绘制的椭圆追踪框的大小、形状和方向。
>
> **CamShift算法的具体过程：**
>
> 1.初始化搜索窗；
>
> 2.计算搜索窗的颜色概率分布（反向投影）；
>
> 3.执行meanshift算法，获取搜索窗新的大小和位置；
>
> 4.在下一帧视频图像中用3中的值重新初始化搜索窗的大小和位置，再跳转到2继续进行。

（3）start函数分析

```
def start(self):
    self.selection = (0, 0, 640, 480)  # 选取该区域作为颜色识别检测区域
    self.tracking_state = 1  # 是否需要跟踪检测
```

> 这个函数主要是对检测区域进行设置以及把追踪状态设置为 1 ；
>
> self.selection这个函数的四个参数分别代表检测区域左上角(x1,y1)和(x2,y2)；
>
> self.tracking_state是用来设置是否跟踪检测，1 代表开启，0 代表关闭。