title: openmv学习之旅-2 top: false cover: false toc: true mathjax: true date: 2019-08-31 16:26:39 password: summary: tags: 机器视觉 categories: 机器视觉

大家好，我是杰杰。

从上一篇openmv的学习中openmv学习之旅-1

我们可以很简单运用micropython在openmv上做我们想做的事情。

Python这个东西用起来是很简单的，， 下面来说说改善色块追踪的算法

先做个改善前的分析吧：

改善前：

```
API: find_blobs
```

`thresholds` 是颜色的阈值。这个参数是一个列表，可以包含多个颜色。在返回的色块对象blob可以调用code方法，来判断是什么颜色的色块。

`roi` 是“感兴趣区”。

`x_stride` 就是查找的色块的x方向上最小宽度的像素，默认为2。

`y_stride` 就是查找的色块的y方向上最小宽度的像素，默认为1。

`area_threshold` 面积阈值，如果色块被框起来的面积小于这个值，会被过滤掉。

`pixels_threshold` 像素个数阈值，如果色块像素数量小于这个值，会被过滤掉

`merge` 合并，如果设置为True，那么合并所有重叠的blob为一个。 注意：这会合并所有的blob，无论是什么颜色的。如果你想混淆多种颜色的blob，只需要分别调用不同颜色阈值的find_blobs。

```
blobs = img.find_blobs([red])
```

`find_blobs` 对象返回的是多个 `blob` 的列表。 而一个 `blobs` 列表里包含很多 `blob` 对象， `blobs` 对象就是色块，每个 `blobs` 对象包含一个色块的信息。

blob有多个方法：

`blob.rect()` 返回这个色块的外框——矩形元组(x, y, w, h)，可以直接在image.draw_rectangle中使用。

`blob.x()` 返回色块的外框的x坐标（int），也可以通过blob[0]来获取。

`blob.y()` 返回色块的外框的y坐标（int），也可以通过blob[1]来获取。

`blob.w()` 返回色块的外框的宽度w（int），也可以通过blob[2]来获取。

`blob.h()` 返回色块的外框的高度h（int），也可以通过blob[3]来获取。

`blob.pixels()` 返回色块的像素数量（int），也可以通过blob[4]来获取。

`blob.cx()` 返回色块的外框的中心x坐标（int），也可以通过blob[5]来获取。

`blob.cy()` 返回色块的外框的中心y坐标（int），也可以通过blob[6]来获取。

`blob.rotation()` 返回色块的旋转角度（单位为弧度）（float）。如果色块类似一个铅笔，那么这个值为0-180°。如果色块是一个圆，那么这个值是无用的。如果色块完全没有对称性，那么你会得到0-360°，也可以通过blob[7]来获取。

`blob.code()` 返回一个16bit数字，每一个bit会对应每一个阈值。

（上面的知识在openmv的官网上都有说明）

下面是说说调用find_blobs来做色块的追踪的原理

它是全幅图像扫描，它有优点也有缺点

先说说优点吧:信息全面，全幅图像的搜索，把所有色块都搜索进来了

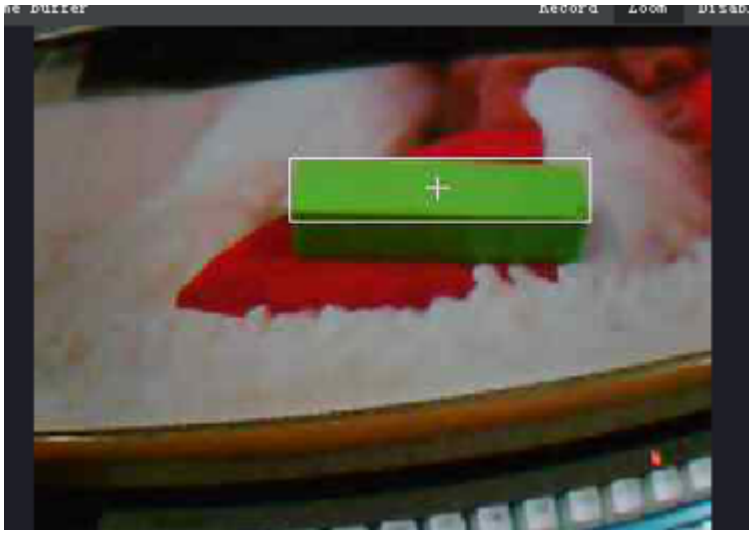
但是缺点也不少：①：运算的速度，是很慢的，有些地方我们根本不需要扫描它。

②：色块的数量，很多时候，会有很多延时差不多的色块过来干扰，导致追踪失败。

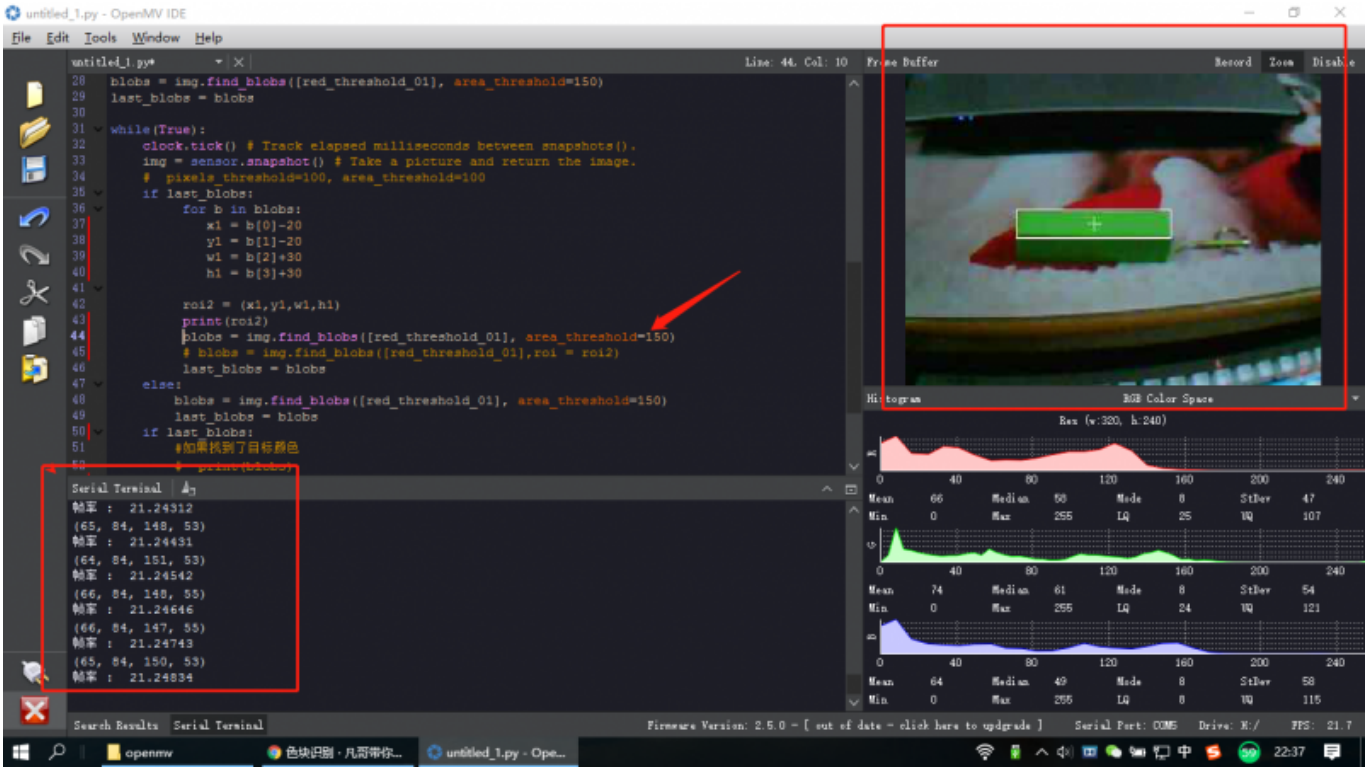
源码：

```
import sensor, image, time
red_threshold_01 = (0, 35, 0, 50, -10, 40)
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(10)
sensor.set_auto_whitebal(False)
clock = time.clock()
while(True):
    clock.tick()
    img = sensor.snapshot()
    blobs = img.find_blobs([red_threshold_01],
                           area_threshold=150)
    if blobs:
    #如果找到了目标颜色
        print(blobs)
        for b in blobs:
        #迭代找到的目标颜色区域
            # Draw a rect around the blob.
            img.draw_rectangle(b[0:4]) # rect
            #用矩形标记出目标颜色区域
            img.draw_cross(b[5], b[6]) # cx, cy
            #在目标颜色区域的中心画十字形标记
    print(clock.fps())
```

效果图：



运算速度：



从拍摄完到扫描完，每秒只能处理二十多帧图像，而且是简单的处理。

虽然脱机运行速度可以快一倍。但是还是比较慢的。

肯定要改进啊。

以下是改进扫描算法的思想：

①：首先做一次全幅图像的扫描，找到我们需要的色块。

②：我们得到色块的信息：如色块的长、宽、及色块的位置

③：我们只扫描色块周围的区域（ROI）

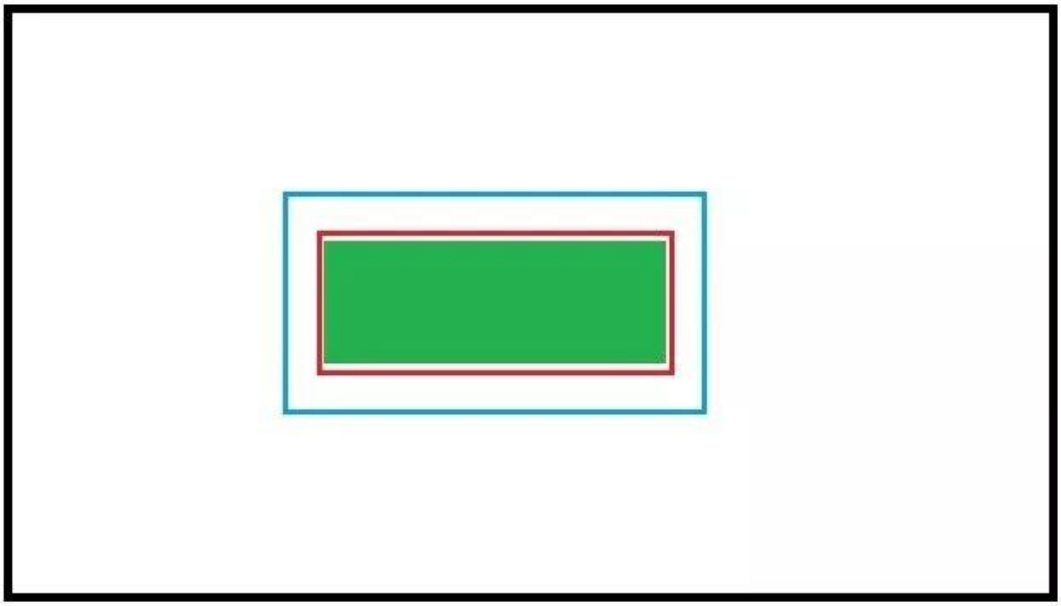
④：假设在新的ROI找不到我们需要的色块，就重新来。

这个方法类似于飞思卡尔的边缘检测。

其实这个改进是非常简单的。因为我们需要的东西在microPython中全部都有。

只需要拿到find_blobs的返回的东西就好啦。

算法示意图



绿色的是我们追踪的色块，

而红色的框是我们第一次全局扫描得到的东西

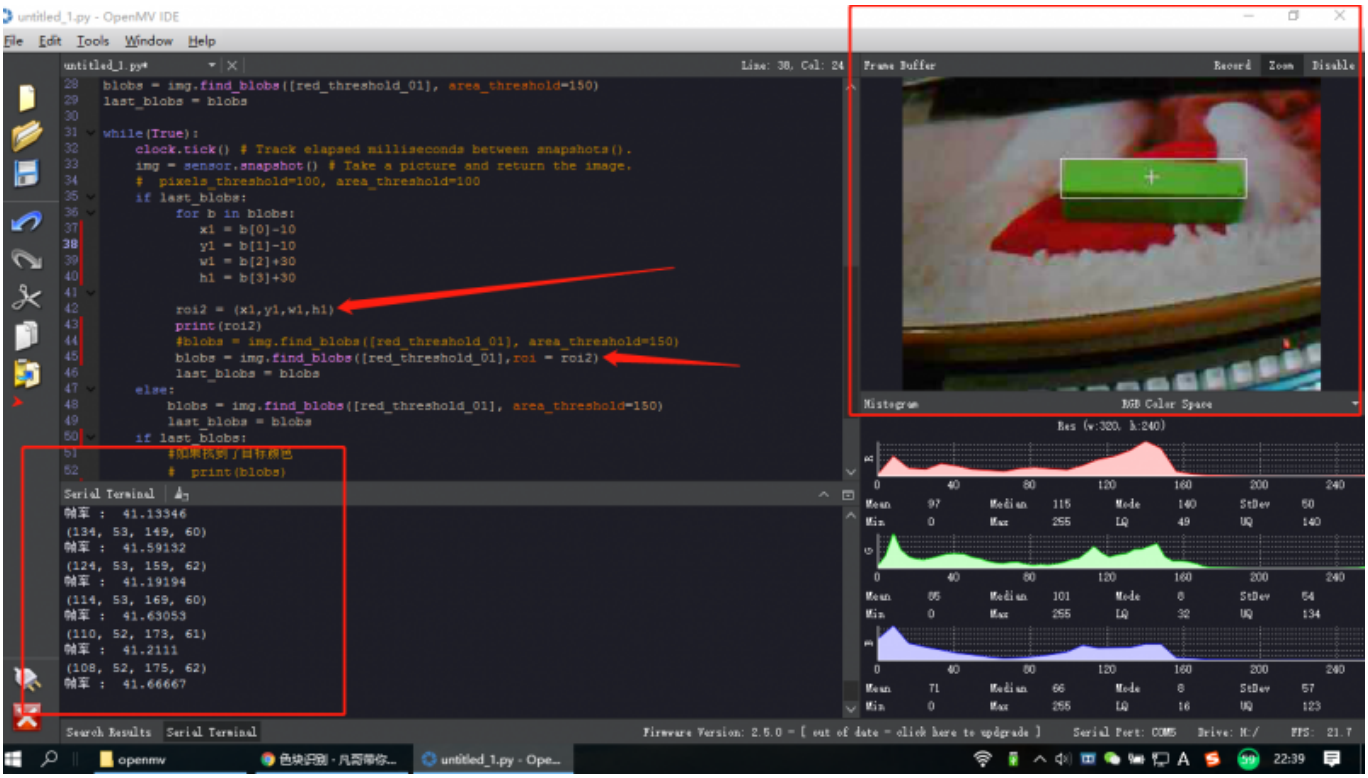
那么我们只需在红色的框之外做一次扫描就能得到绿色块啦

实现的源码

```
 import sensor, image, time
 #red_threshold_01 = (45, 100, -60, 80, 34, 91)
 red_threshold_01 = (21, 29, -72, 6, -11, 17)
 sensor.reset()
 sensor.set_pixformat(sensor.RGB565)
 sensor.set_framesize(sensor.QVGA)
 sensor.skip_frames(20)
 sensor.set_auto_whitebal(False)
 #关闭白平衡。白平衡是默认开启的，在颜色识别中，需要关闭白平衡
clock = time.clock()
img = sensor.snapshot()
blobs = img.find_blobs([red_threshold_01],
                       area_threshold=150)
last_blobs = blobs
while(True):
    clock.tick()
    img = sensor.snapshot()
    if last_blobs:
         for b in blobs:
```

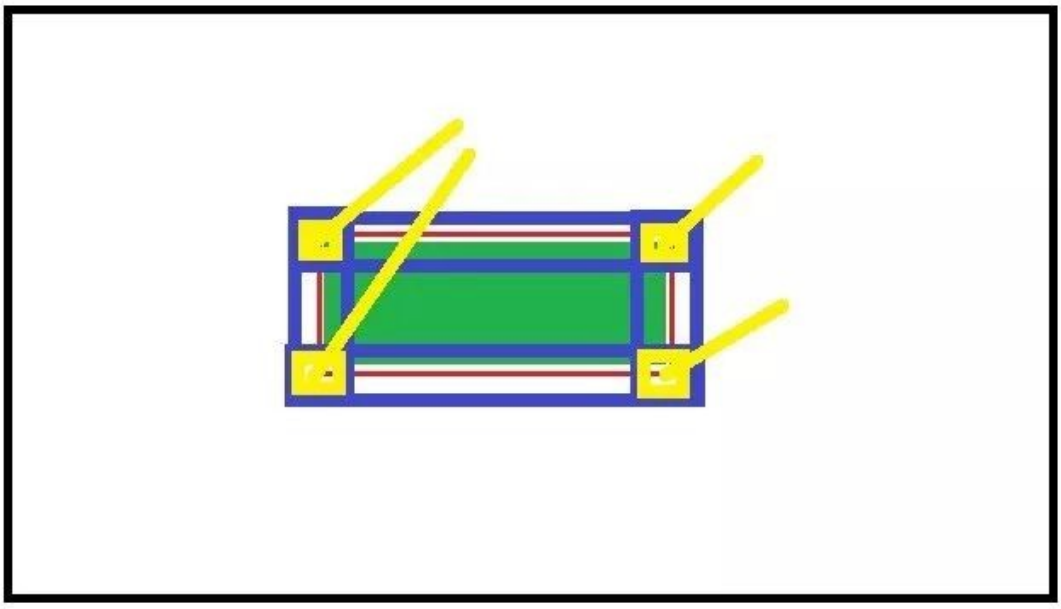
```
            x1 = b[0]-7
            y1 = b[1]-7
            w1 = b[2]+12
            h1 = b[3]+12
            #print(b.cx(),b.cy())
        roi2 = (x1,y1,w1,h1)
        print(roi2)
        blobs = img.find_blobs([red_threshold_01],
                               roi = roi2,
                               area_threshold=1000)
        last_blobs = blobs
    else:
        blobs = img.find_blobs([red_threshold_01],
                               area_threshold=1000)
        last_blobs = blobs
    if last_blobs:
        #如果找到了目标颜色
        #  print(blobs)
        for b in last_blobs:#迭代找到的目标颜色区域
            img.draw_rectangle(b[0:4])
            img.draw_cross(b[5], b[6])
    print("帧率 : ",clock.fps())
```

改善后的帧率的确是快了不少的：基本能快一倍以上，当然，物体在图片越大，处理的速度会越慢。



帧率达到了 45 ，并且，对于运动中的物体，也能很好的追踪出来，减少无关物体的干扰。

按照这个思想，我们还能把这个物体分成四条边来扫描：



那么是不是只需要扫描到这个物体的四条边，并且证明四条边的点都有重合，那么，我们就能知道这个物体是一个整体从而得到物体的位置。。。。

当然，这只是想法。对于程序员任何的功能想法的实现都需要代码的实现，这代码我就不实现了，有兴趣的可以试试。。。

喜欢就关注我吧！

相关代码可以在公众号后台获取。