

OpenMV 机器视觉开发指南

武汉无名创新科技有限公司

一、前言

OpenMV 是基于 MicroPython 的嵌入式机器视觉模块，目标是成为机器视觉界的“Arduino”。它成本低易拓展，开发环境友好，除了用于图像处理外，还可以用 Python 调用其硬件资源，进行 I/O 控制，与现实世界进行交互。

你想学习机器视觉，却不知从何入手？在傅立叶变换，小波变换等一系列信号与系统的强烈攻势之下，挣扎抑或挫败？

封装各种算法细节，不需要直接跟底层代码打交道。

用户友好的 PythonAPI，让你开始的时候，就享受机器视觉带给你的愉悦，逐层剥开底层算法实现原理，代码开源，随意修改底层源码，编译固件。

非计算机专业，想在自己的机器人或者小车上加载机器视觉模块，却担心自己学不懂？电赛来袭，慌忙准备，没时间系统学习机器视觉？

来用 OpenMV 吧，快速入手，不需要专业的背景知识。

几行代码，轻松搞定。例程丰富，也许你需要做的只是改一下参数，让你的机器人开启视觉智能。

二、OpenMV 简介及开发语言介绍

2.1、OpenMV 简介

Openmv 是使用 STM32F765VI ARMCortex M7 处理器，216MHz，512KBRAM，2Mbf1ash，所有的 I/O 引脚输出 3.3V 并且 5V 兼容。这个处理器有以下的 IO 接口：

全速 USB(12Mbps)接口，连接到电脑。当插入 OpenMV 摄像头后，你的电脑会出现一个虚拟 COM 端口和一个“U 盘”。SD 卡槽拥有 100Mbps 读写，这允许你的 OpenMV 摄像头录制视频，和把机器视觉的素材从 SD 卡提取出来。一个 SPI 总线高达 54Mbps 速度，允许你简单的把图像流数据传给 LCD 扩展板，WiFi 扩展板，或者其他控制器。一个 I2C 总线，CAN 总线，和一个异步串口总线(TX/RX)，用来链接其他控制器或者传感器。一个 12-bitADC 和一个 12-bitDAC。3 个 I/O 引脚用于舵机控制。所有的 IO 口都可以用于中断和 PWM(板子上有 10 个 I/O 引脚)。一个 RGBLED（三色），两个高亮的 850nm IR LED（红外）。

OV7725 感光元件在 80FPS 下可以处理 640x480 分辨率 8-bit 灰度图或者 320x240

分辨率 16-bit RGB565 彩色图像。当分辨率低于 320×240 可以达到 120FPS。大多数简单的算法运行在 30FPS 一下。

OpenMV1 的像素是 30 万，OpenMV2 的像素是 200 万，OpenMV3 又改为了 30 万像素。即便使用了 STM32 最好的芯片 (STM32F7)，在处理图像的时候，面对大量的计算，也难免会有些吃力。OpenMV3 之所以使用低像素，是因为在清晰度和性能之间做了一个折中。30 万像素，在处理一些图像细节的时候会比较吃力，但是巡线，颜色追踪还是没有问题的。

2.2、应用

目前 OpenMV 摄像头可以用来做一下的事情 (未来会更多)：

Frame Differencing 帧差分算法。

您可以使用 OpenMV Cam 上的帧差分算法来查看场景中的运动情况。帧差分算法可以将 OpenMV 用于安全应用。

Color Tracking 颜色追踪

您可以使用 OpenMV 在图像中一次检测多达 16 种颜色 (实际上永远不会想要找到超过 4 种颜色)，并且每种颜色都可以有任意数量的不同的斑点。OpenMV 会告诉您每个 Blob 的位置，大小，中心和方向。使用颜色跟踪，您的 OpenMV Cam 可以进行编程，以跟踪太阳，线跟踪，目标跟踪等等 (视频演示详见：<https://singtown.com/video/>)。

Marker Tracking 标记跟踪

您可以使用 OpenMV Cam 来检测颜色组的颜色，而不是单独的颜色。这允许您在对象上放置颜色标签 (2 种或多种颜色的标签)，OpenMV 会获取标签对象的内容 (视频演示详见：<https://singtown.com/video/>)。

Face Detection 人脸检测

您可以使用 OpenMV Cam (或任何通用对象) 检测脸。OpenMV 摄像头可以处理 Haar 模板进行通用对象检测，并配有内置的 Frontal Face 模板和 Eye Haar 模板来检测人脸和眼睛 Eye Tracking 眼动跟踪。您可以使用眼动跟踪来检测某人的注视方向。您可以使用它来控制机器人。眼睛跟踪检测瞳孔的位置，同时检测图像中是否有眼睛。

Optical Flow 光流

您可以使用光流来检测您的 OpenMV 摄像机面前的画面。例如，您可以使用四旋翼上的光流定点来保证在空中的稳定性。

QR Code Detection/Decoding 二维码检测/解码

您可以使用 OpenMV Cam 在其视野中读取 QR 码。通过 QR 码检测/解码，您可以使智

能机器人能够读取环境中的标签。 **请关注我们后期的代码更新**

Data Matrix Detection/Decoding 矩阵码检测/解码

OpenMV Cam M7 也可以检测和解码矩阵码（2D 条形码）。 **请关注我们后期的代码更新。**

Linear Barcode Decoding 条形码

OpenMV Cam M7 还可以处理 1D 条形码。他可以解码 EAN2、EAN5、EAN8、UPCE、ISBN10、UPCA、EAN13、ISBN13、I25、DATABA、DARABAR_EXP、CODABAR、CODE39、CODE93 和 CODE128（视频演示详见：<https://singtown.com/video/>）。

AprilTag Tracking 标记跟踪

甚至比上面的 QR 码更好，OpenMV Cam M7 也可以追踪 160x120 的 AprilTags，高达约 12FPS。AprilTags 是旋转不变，尺度不变，剪切不变和照明不变的最先进的基准标记（视频演示详见：<https://singtown.com/video/>）。

Line Detection 直线检测

OpenMV Cam 可以在几乎跑满帧率的情况下，快速完成无限长的直线检测。而且，也可以找到非无限长的线段或者使用霍夫变换检测间断的线（视频演示详见：<https://singtown.com/video/>）。

Template Matching 模板匹配

您可以使用 OpenMV 模板匹配来检测视野中是否有模板相似的图片。例如，可以使用模板匹配来查找 PCB 上的标记，或读取显示器上的已知数字。

Image Capture 图像捕捉

您可以使用 OpenMV 捕获高达 320x240 RGB565（或 640x480 灰度）BMP/JPG/PPM/PGM 图像。可以直接在 Python 脚本中控制如何捕获图像。最重要的是，使用机器视觉的算法，进行绘制直线，绘制字符，然后保存。

Video Recording 视频录制

您可以使用 OpenMV 摄像机记录多达 320x240 RGB565（或 640x480 灰度）JPEG 视频或 GIF 图像。您可以在 Python 脚本中直接控制如何将每个视频帧记录，并完全控制视频录制的开始和结束。而且，像拍摄图像一样，您可以使用机器视觉的算法，进行绘制直线，绘制字符，然后保存。最后，所有上述功能都可以混合 IO 引脚的控制，来配合你自己的自定义应用，以与现实世界交谈。

2.3、Python 开发语言入门

廖学峰-小白的 Python 新手教程: <https://www.liaoxuefeng.com/wiki/0014316089557264a6b348958f449949df42a6d3a2e542c000>。

廖学峰大大的 Python 教程, 我见过的写得最好的 Python 开发教程, 个人的 Python 入门就是从廖学峰的教程开始的。笨办法学 Python: <https://wizzardforcel.gitbooks.io/lpthw/content/>。此书有大量编程任务, 在 Python 中通过练习和记忆等技巧慢慢建设和建立技能, 然后应用它们解决越来越困难的问题。

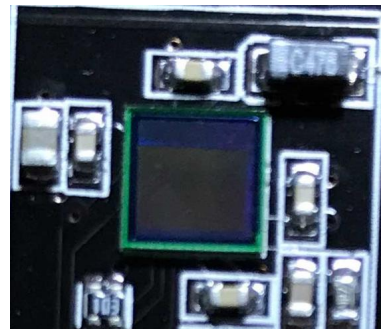
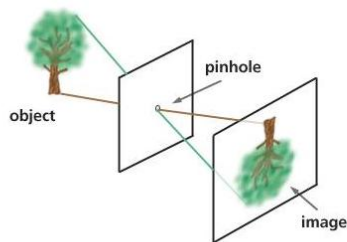
就学习 OpenMV 的角度上来讲, 熟悉 Python 基本语法, 各种数据结构, 控制流语句就可以进行 OpenMV 的学习和使用。

MicroPython: <https://micropython.org/>。

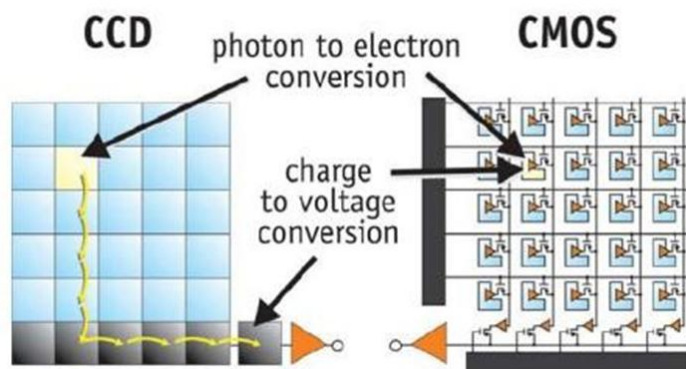
MicroPython 的库函数, 随着各种发行版的使用会有所不同。OpenMV 的硬件资源控制部分, 参阅文档。OpenMV-MicroPython Libraries: <http://docs.openmv.io/library/index.html#python-standard-libraries-and-micro-libraries>。

三、机器视觉常识

摄像头分为数字摄像头和模拟摄像头两大类。数字摄像头可以将视频采集设备产生的模拟视频信号转换成数字信号，进而将其储存在计算机里。模拟摄像头捕捉到的视频信号必须经过特定的视频捕捉卡将模拟信号转换成数字模式，并加以压缩后才可以转换到计算机上运用。数字摄像头可以直接捕捉影像，然后通过串、并口或者 USB 接口传到计算机里。景物=>光学图像=>电学信号=>数字图像信号=>PC 显示景物通过镜头产生光学图像；光学图像再同半导体的图像传感器生成电学信号；电学信号由 A/D 转换器转化为数字图像信号；数字图像信号经由 DSP 处理，在 USB 连接下在 PC 上显示出来。说到底，摄像头就是一个将光学信号转变成电信号的一个装置。在计算机视觉中，最简单的相机模型是小孔成像模型如图所示：



感光芯片（CCD or CMOS die）：



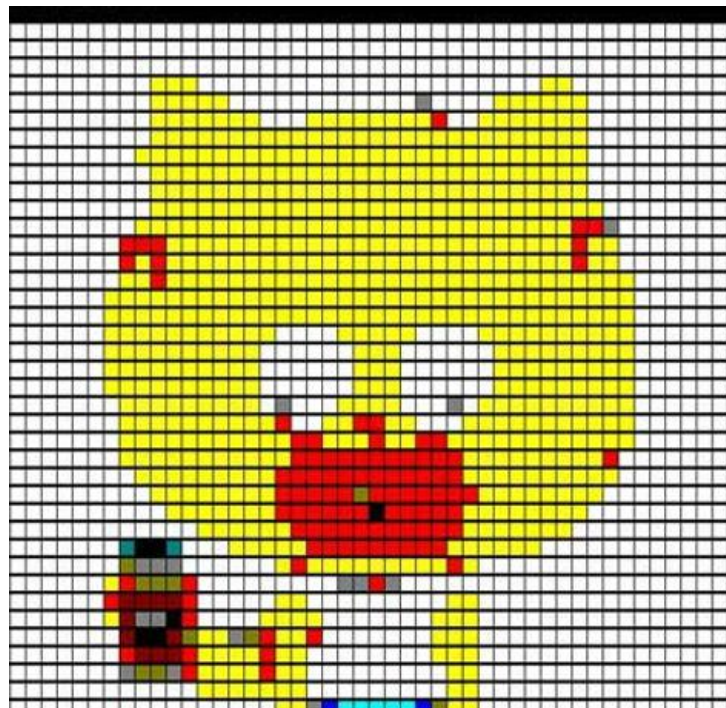
CCD(Charge Coupled Device)电荷耦合组件，用于录像或者图像扫描；灵敏度高，噪声小，信噪比大，但成本高，生产工艺复杂，功耗高。

CMOS(Complementary Metal-Oxide Semiconductor)附加金属氧化物半导体

组件，是低端视频设备；集成度高，功耗低（不到 CCD 的 1/3），成本低，但是噪声大，灵敏度低，对光源要求高。

什么是像素和分辨率

像素，为视频显示的基本单位，译自英文“pixel”，pix 是英语单词 picture 的常用简写，有时亦被称为 pel (picture element)。在很多情况下，它们采用点或者方块显示。每个像素可有各自的颜色值，可采三原色显示，因而又分成红、绿、蓝三种子像素（RGB 色域），或者青、品红、黄和黑（CMYK 色域，印刷行业以及打印机中常见）。照片是一个个取样点的集合，在视频没有经过不正确的/有损的压缩或相机镜头合适的前提下，单位面积内的像素越多代表分辨率越高，所显示的视频就会接近于真实物体。比如有 640*480 个点，每个点就是一个像素，把每个点的像素收集整理起来，就是一副图片，那么这张图片的分辨率就是 640*480：



什么是帧率

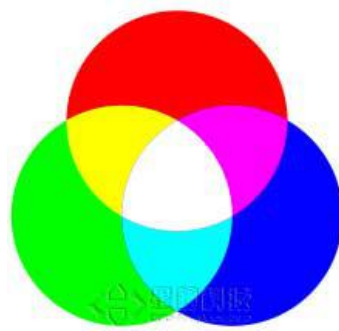
帧率（FPS）就是每秒钟处理的图片数量，如果超过 20 帧，人眼就基本分辨不出卡顿。当然，如果用在机器上，帧率是越高越好的。

注：没有标注均为不传输图像给 IDE，因为这个过程很耗费时间。

Openmv 颜色空间（RGB 颜色空间和 LAB 颜色空间）

RGB 三原色的原理不是物理原因，而是由于人的生理原因造成的。人的眼睛

内有几种辨别颜色的锥形感光细胞，分别对黄绿色、绿色和蓝紫色（或称紫罗兰色）的光最敏感（波长分别为 564、534 和 420 纳米）。



Lab 颜色空间中，L 亮度；a 的正数代表红色，负端代表绿色；b 的正数代表黄色，负端代表蓝色。不像 RGB 和 CMYK 色彩空间，Lab 颜色被设计来接近人类视觉。因此 L 分量可以调整亮度对，修改 a 和 b 分量的输出色阶来做精确的颜色平衡。

其他颜色模型参考：

[小波的世界]HSI 颜色空间及其应用

<http://nkwavelet.blog.163.com/blog/static/22775603820147197503722>

[百度文库] RGB、Lab、YUV、HSI、HSV 等颜色空间的区别

<https://wenku.baidu.com/view/f38c04e69b89680203d82513.html>

图像的整体属性

高度 `height img.height()`

宽度 `width img.width()`

图像的大小 `img.size()`，size 的取值其实是 `width * height`，就是说一共有多少个 pixels

图像格式：`img.format()`，图像格式一共有两种：`sensor.GRAYSCALE:8-bits per/ pixel`、`sensor.RGB565:16-bits per pixel`。

函数返回数值为整数，取值范围是 0/1：`format0`：灰度图，每个 pixel 占 8 个位 `format1`:RGB 图，每个 pixel 占 16 个位。

此时你已心生疑虑，对于 grayscale 是单个数值，占 8 个位

rgb 应该是其三倍啊，因为其存储了三个值，但是为啥是 16 个位呢详情参见 R

GB565 详解

实验代码:

```
import sensor, image, time
sensor.reset() # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565) # Set pixel format to RGB565 (or GRAYSCALE)
sensor.set_framesize(sensor.QVGA)
clock = time.clock() # Create a clock object to track the FPS. while(True):
    clock.tick() # Update the FPS clock.
    img = sensor.snapshot() # Take a picture and return the image. print("IMG Format")
    print(img.format())
    print("IMG Width")
    print(img.width())
    print("IMG Height")
    print(img.height())
    print("IMG SIZE")
    print(img.size())
```

样例输出

IMG Format 1

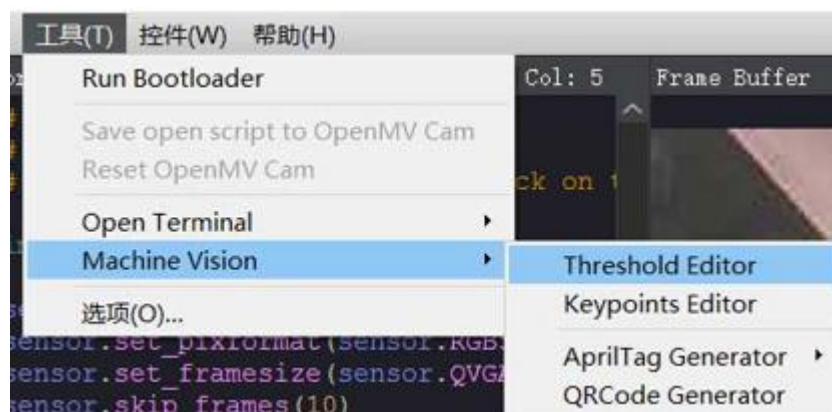
IMG Width 320

IMG Height 240

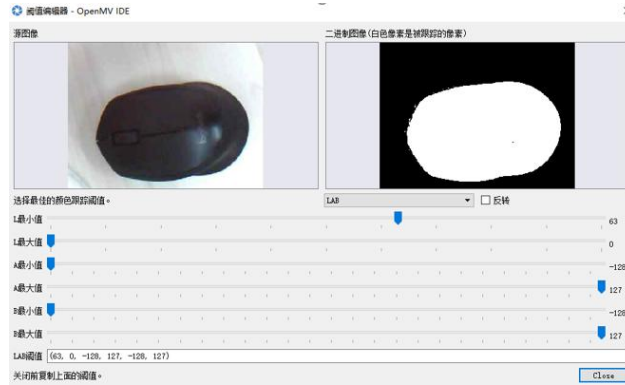
IMG SIZE 153600

阈值选择工具

一个颜色阈值的结构是这样的: red = (minL, maxL, minA, maxA, minB, maxB) 元组里面的数值分别是 L A B 的最大值和最小值。在新版的 IDE, 有更方便的阈值选择工具, 打开工具→Machine Vision→Threshold Editor 见下面。

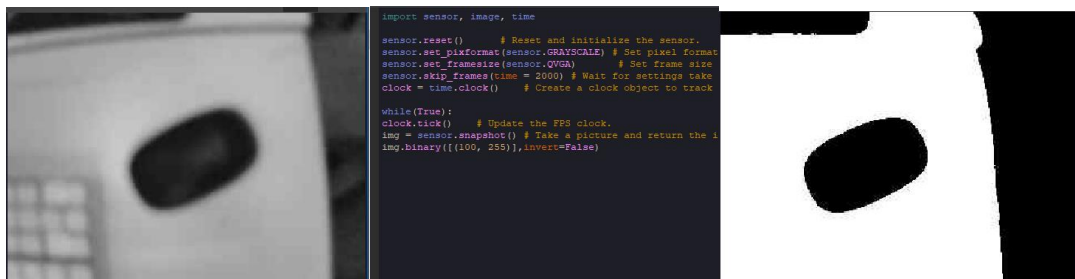


点击 Frame Buffer 可以获取 IDE 中的图像, Image File 可以自己选择一个图像文件。拖动六个滑块, 可以实时的看到阈值的结果, 我们想要的结果就是将我们的目标颜色变成白色, 其他颜色全变为黑色。



二值化 binary

`image.binary(thresholds, invert=False)` thresholds 阈值为一个数组 [...], 数组里面有若干个 tuple 组成, 每个 tuple 都由最大值与最小值组成, (lower, upper), 即上界下界, 如果是 RGB 格式的话, 则需要设定六个阈值 (l_lo, l_hi, a_lo, a_hi, b_lo, b_hi), 在阈值范围内的就设定为 1 (white), 不再阈值范围内的就设定为 0 (black), 原图、实验效果及实验代码:

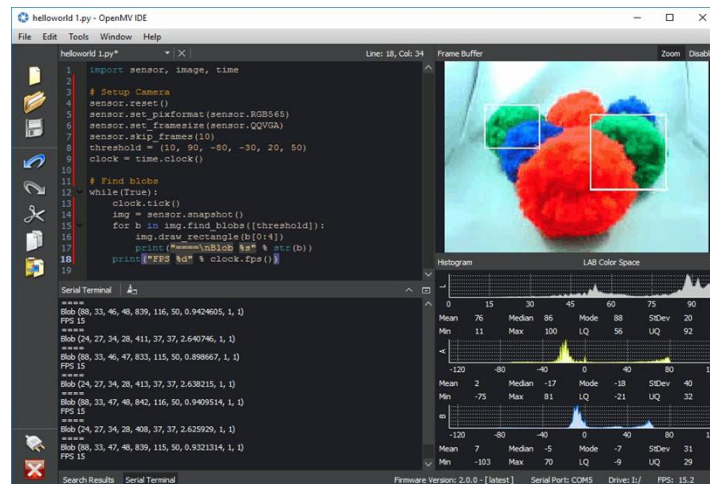


`image.invert()` 反色就是说, 1 (白色) \rightarrow 0 (黑色), 0 (黑色) \rightarrow 1 (白色), 相当于逻辑中的取反操作, 等同于将上面图片黑色部分变成白色, 白色部分变成黑色。

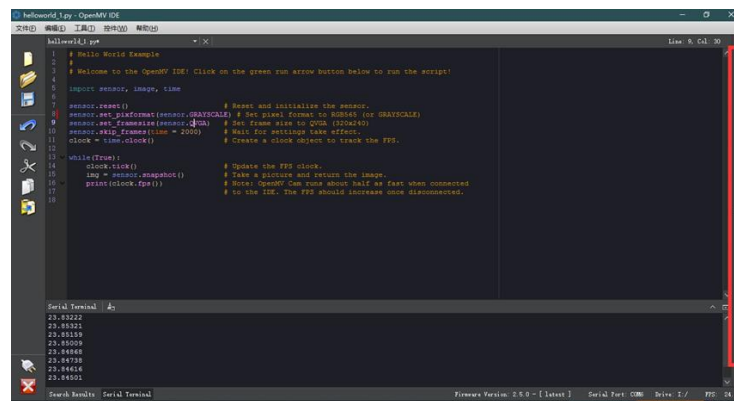
四、Openmv 快速上手

4.1、OpenMV IDE、驱动安装及使用教程

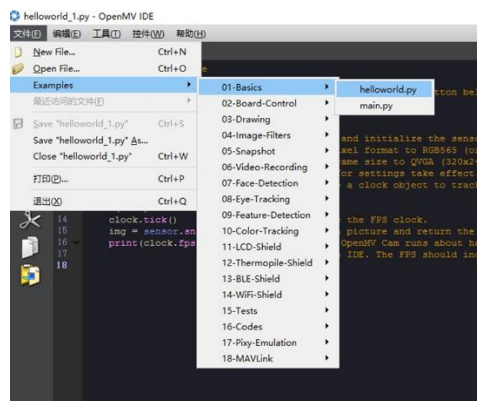
下载 IDE: <https://openmv.io/pages/download>, 当前版本为 v1.7.1 版本, 根据你操作系统的型号选择合适的安装包。因为基于 QT 所以对 Linux 也有很好的支持。IDE 工作台如下图所示。



如果没有图像窗口，从右侧可以拖拽出来。



查看样例代码：



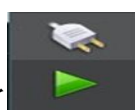
串口连接与代码烧录

点击链接串口，就是左下角那个白色的按钮



连接后高亮，并且显示

绿色的按钮，说明可以运行代码了



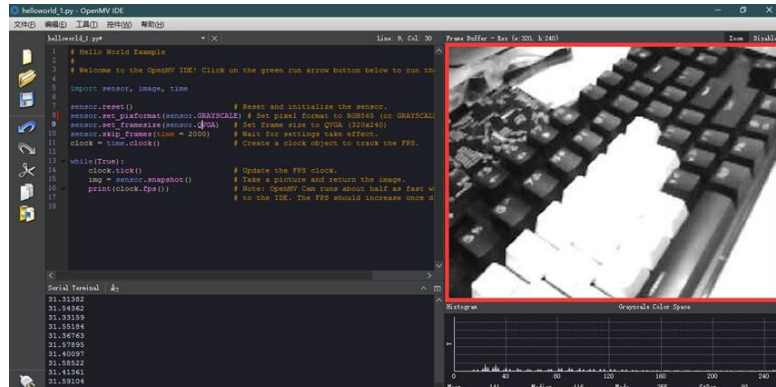
点击绿色的按钮，代码上传给 Open

MV，程序执行，这时在视频区域你可以看到事实的图像了。此时按钮会变成红色

的 X，如果要中断程序，或者改动了代码需要更新，点击 X，然后再重新运行。


视频显示

右上角显示了实时的图像信息，如果你想截图取样的话，直接左键选中一个矩形区域然后右键保存到本地，选择合适的格式。



驱动安装

正常情况下，OpenMV 插入电脑上，会自动安装驱动。目前正常工作在 32 位和 64 位的 win7、win8、win8.1、win10。自动安装后，在设备管理器会出现

一个虚拟串口。 **USB 串行设备 (COM7)**，但是可能驱动不会自动安装，所以需要自己手动安装。这时在设备管理器中会出现一个叹号，表示没有正常安装驱动。



首先下载驱动：<http://pan.baidu.com/s/1pKQd59L> 解压到桌面，然后右键设备管理器中的这个设备，然后点升级驱动：

你希望如何搜索驱动程序软件？

→ 自动搜索更新的驱动程序软件(S)

Windows 将在你的计算机和 Internet 上查找用于相关设备的最新驱动程序软件，除非在设备安装设备中禁用该功能。

→ 浏览计算机以查找驱动程序软件(R)

手动查找并安装驱动程序软件。

选择浏览计算机以查找驱动程序软件

浏览计算机上的驱动程序文件

在以下位置搜索驱动程序软件：

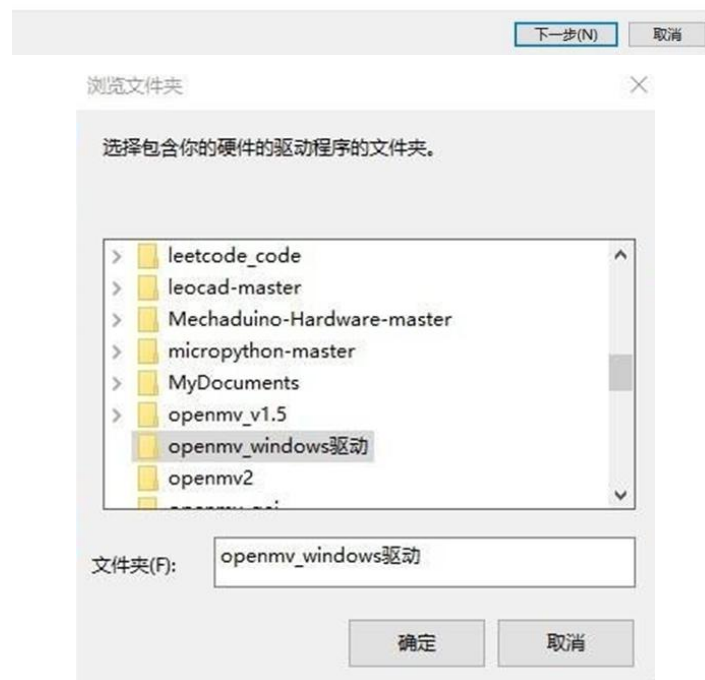
E: [v]

浏览(R)

☒ 包括子文件夹(I)

→ 从计算机的设备驱动程序列表中选择(L)

此列表将显示与该设备兼容的已安装的驱动程序软件，以及与该设备处于同一类别下的所有驱动程序软件。



选中桌面上的 openmv_windows 驱动文件夹，过一会就会正常装好驱动。

浏览计算机上的驱动程序文件

在以下位置搜索驱动程序软件:

C:\Users\kidswong999\Desktop\openmv_windows驱动

浏览(R)...

☒ 包括子文件夹(I)

→ 从计算机的设备驱动程序列表中选择(L)

此列表将显示与该设备兼容的已安装的驱动程序软件，以及与该设备处于同一类别下的所有驱动程序软件。

下一步(N)

取消

这时应该会成功的安装好驱动。但是 但是 但是，如果你遇到下面的情况，就比较麻烦了。

Windows 安装设备的驱动程序软件时遇到一个问题

Windows 已找到设备的驱动程序软件，但在试图安装它时遇到错误。

OpenMV USB Comm Port

系统找不到指定的文件。

如果您知道设备制造商，则可以访问其网站并检查驱动程序软件的支持部分。

这是因为有些系统用了一些精简系统的软件，把一些不常用的驱动删除了，但是 OpenMV 的驱动依赖这些驱动，所以要把他们粘贴回来。

驱动安装失败解决办法

OpenMv 驱动安装失败，90%的情况都是你电脑的问题，精简版操作系统和使用了一些优化软件通常是引起此类问题的原因。OpenMV 驱动解决办法跟 arduino 类似。这是因为精简版的 window 系统删掉了一些不常用的驱动信息引起的，解决方法如下：

Step1 首先打开 C:\windows\inf\setupapi.dev.log

这个文件包含了有关即插即用设备和驱动程序安装的信息，当然它也记录你 Arduino 驱动安装失败的原因。打开该文件，滚动到文件末尾附近，你可以看到如下信息：

```

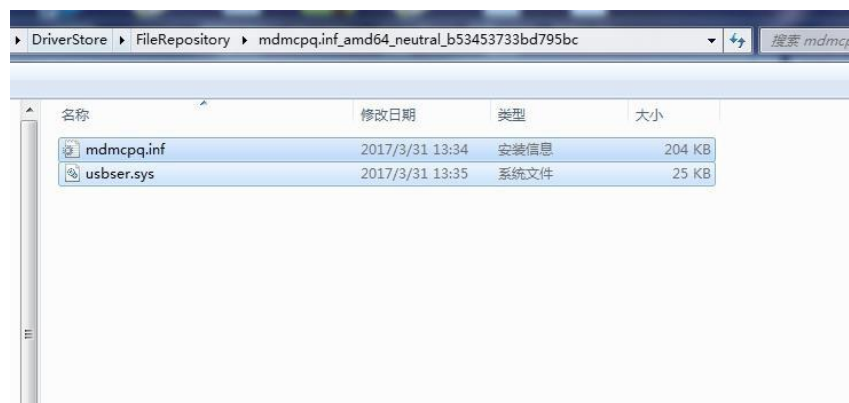
setupapi.devlog - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

dvi: [DIF_INSTALLDEVICEFILES - exit(0x00000000)] 09:26:57.884
dvi: [Firmware file queue...]
dvi: [SCAN_FILE_QUEUE]
dvi: [Scan Flags=820]
dvi: [SPQ_SCAN_PROBE_COPY_QUEUE]
dvi: [SPQ_SCAN_FILE_COMPARISON]
dvi: [SPQ_SCAN_ACTIVATION]
dvi: [Scan number of copy nodes=1]
dvi: [Scan action=200 DoPruning=32]
dvi: [Scan end Validity flags=820 CopyNodes=1]
dvi: [SCAN_FILE_QUEUE exit(0, 0x00000000)]
dvi: [Committing file queue...]
dvi: [Commit file queue]
dvi: [Commit to DelNodes=0 RepNodes=0 CopyNodes=1]
dvi: [SPFILENOTIFY_STARTQUEUE]
dvi: [SPFILENOTIFY_STARTQUEUE - exit(0x00000001)]
dvi: [Commit copy subqueue]
dvi: [subqueue count=1]
dvi: [SPFILENOTIFY_STARTQUEUE]
dvi: [SPFILENOTIFY_STARTQUEUE - exit(0x00000001)]
dvi: [source media:]
dvi: [Description - [windows cd]]
dvi: [SourcePath - [C:\Windows\System32\DriverStore\FileRepository\mdmcpq.inf_amd64_neutral_b53453733bd795bc]]
dvi: [SourceFile - [usbser.sys]]
dvi: [Flags - 0x00000000]
dvi: [SPFILENOTIFY_NEEDMEDIA]
dvi: [SPFILENOTIFY_NEEDMEDIA - exit(0x00000000)]
dvi: [SPFILENOTIFY_NEEDMEDIA - returned 0x00000000]
dvi: [source media: SPFILENOTIFY_NEEDMEDIA - returned 0x00000000]
dvi: [Error 2: The system cannot find the file specified.]
dvi: [Commit copy subqueue exit(0x00000002)]
dvi: [FileQueueCommit aborting!]
dvi: [Error 2: The system cannot find the file specified.]
dvi: [SPFILENOTIFY_ENQUEUE]
dvi: [SPFILENOTIFY_ENQUEUE - exit(0x00000001)]

```

Step2 在 C:\Windows\System32\DriverStore\FileRepository\路径下，新建一个 mdmcpq.inf_amd64_neutral_b53453733bd795bc（这个根据你电脑的提示修改）文件夹

Step3 将丢失文件拷贝到刚新建的文件下后，从新安装驱动，不出意外可以安装成功，缺失驱动文件在网盘中下载 <http://pan.baidu.com/s/1jIxfxoM>



4.2、固件编译、烧录、升级

编译固件及固件升级: <https://github.com/openmv/openmv/wiki> 固件的烧录: <https://singtown.com/video/>

4.3、使用注意事项

OpenMV3 拔插注意事项，OpenMV3 的存储有两种，一种是 Flash，一种是 SD 卡。如果需要频繁的烧录程序，建议配备一个 SD 卡，将程序存放在 SD 卡中。默认是 Flash 在 OpenMV3 拔下来之前需要断开 OpenMV 的串口连接，弹出 Flash 存储。**注意！**尽量别在程序运行的时候按复位键，另外,Flash 易损,平常使用过程中要注意避免热拔插。

五、常用例程

5.1、基本操作

```
import sensor, image, time

sensor.reset() # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565) # Set pixel format to RGB565 (or GRAYSCALE)
sensor.set_framesize(sensor.QVGA) # Set frame size to QVGA (320x240)
sensor.skip_frames(time = 2000) # Wait for settings take effect.
clock = time.clock() # Create a clock object to track the FPS.

while(True):
    clock.tick() # Update the FPS clock.
    img = sensor.snapshot() # Take a picture and return the image.
    print(clock.fps()) # Note: OpenMV Cam runs about half as fast when connected
                        # to the IDE. The FPS should increase once disconnected.
```

用 usb 线与电脑连接后，打开文件——examples——01Basic——helloworl d.py 例程，点击左下角绿色箭头按钮运行。import sensor, image, time 引入此例程依赖的模块，sensor 是与摄像头参数设置相关的模块，image 是图像处理相关的模块，time.clock 时钟控制相关的模块。import 相当于 c 语言的#include<>，模块相当于 c 语言的库。sensor.reset()#初始化摄像头，reset()是 sensor 模块里面的函数；sensor.set_pixformat(sensor.GRAYSCALE)设置图像色彩格式，有 RGB565 色彩图和 GRAYSCALE 灰度图两种；sensor.set_framesize(sensor.QVGA).设置图像像素大小，sensor.QQVGA:160x120，sensor.QQQVGA:80x60，sensor.QQVGA2:128x160（一般用于 LCD 扩展板），sensor.QVGA:320x240，sensor.QQCIF:88x72，sensor.QCIF:176x144，sensor.CIF: 352x288，sensor.skip_frames(10),clock =time.clock()初始化时钟，**注意：**python 中没有大括号{}，以缩进代替{}，而且缩进的 tab 和空格不能混用，一个程序只能用一种缩进（一个 tab 或者四个空格）

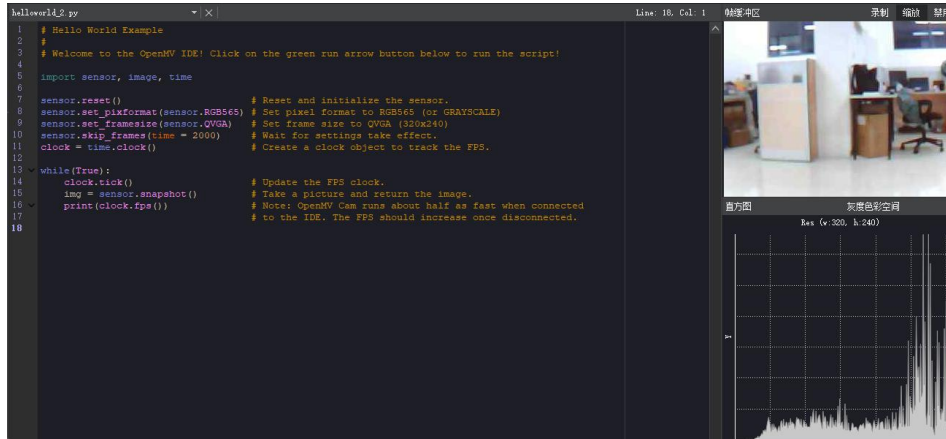
```
while(True):
```

```
#python while 循环，一定不要忘记加冒号“:”
```

```
clock.tick()
```

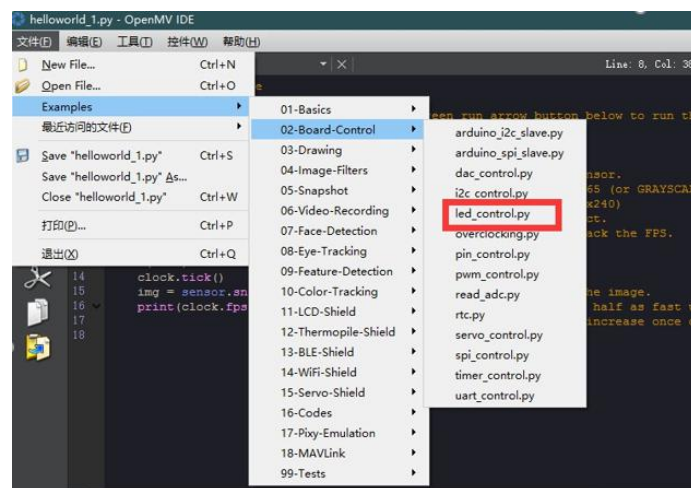
```
img = sensor.snapshot() # Take a picture and return the image.
```

#截取当前图像，存放于变量 img 中。注意 python 中的变量是动态类型，不需要声明定义，直接用即可。print(clock.fps())#打印当前的帧率。然后就可以看到运行效果啦。



5.2 点亮 LED 灯

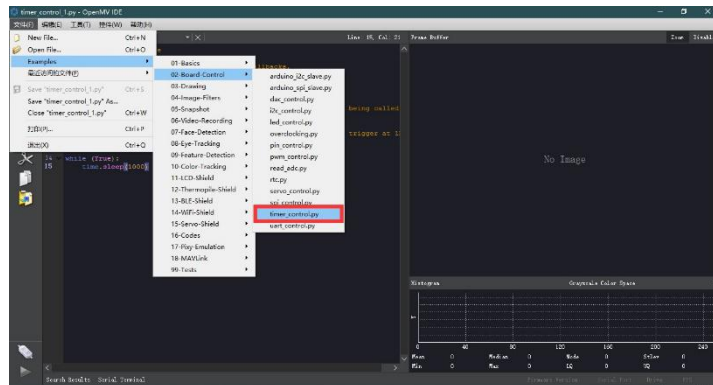
OpenMV 摄像头有一个 RGB LED 和两个红外 LED 灯。您可以单独控制 RGB LED 的红色，绿色和蓝色区段，将两个 IR LED 控制为一个单位。要控制 LED，首先导入 pyb 模块。然后为要控制的特定 LED 创建一个 LED 类对象，`pyb.LED(number)` 创建一个 LED 对象，您可以使用它来控制特定的 LED。通过 `pyb.LED0` 控制红色 RGB LED，“1”控制绿色 RGB LED，“2”控制蓝色 RGB LED，“3”控制两个红外 LED。在创建如上所述的 LED 控制对象后，有三种方法可以调用每个 LED 有 `off()`, `on()`, 与 `toggle()` 三个选项。按照目录打开此文件 `led_control.py`。



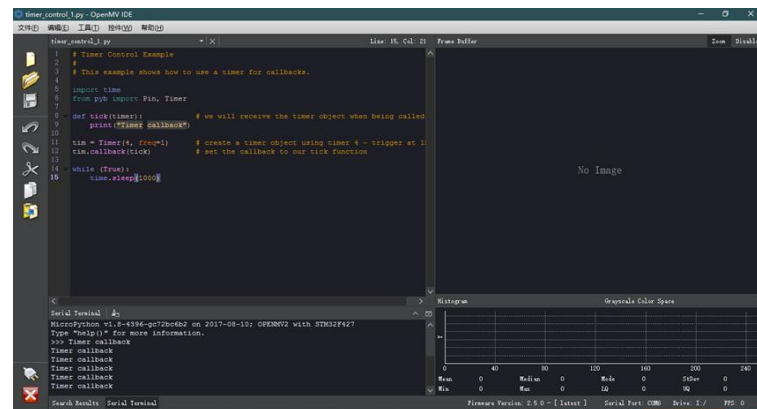
5.3、定时器的使用

官方文档:<http://docs.openmv.io/library/pyb.Timer.html?highlight=timer> 意思也就是 Timer1 用于控制摄像头，Timer5 用于控制舵机驱动，Timer6 用于定 ADC/DAC 读或写，所以，我们编程的时候不能使用这三个。那也就是说，

只可以用定时器 2, 3, 4, 当然, 没用到那些外设的时候, 也就是只用 LED 的时候 14 个定时器都能用。打开图中所示文件。



此程序初始化了定时器4, 频率设置为1hz, 回调函数为tick, 也就是每隔1s 进行一次tick函数, 打印一句Timer callback, 效果如下。



注意: 定时器调用的时候, 不可以开辟新的内存, 这一点很重要, 当你在定时器的回调函数定义一个新的变量或者定义一个字符串的时候, 就会提示有 Memory Error 的错误。同时也意味着如果定时器函数回调期间出现了BUG, 不能返回完整的报错信息。所以我们需要提前申请一块缓存, 用于异常处理。

```
import micropython

micropython.alloc_emergency_exception_buf(100)
```

未加紧急异常缓存的报错信息:

```
uncaught exception in Timer(4) interrupt handler
TypeError:
```

定时器的正确用法:

假定现在有个需求, 需要使用定时器, 阶段性的检查串口是否有数据,

并且将其读入。如何实现？如何克服我们之前说的那个，callback 时不可以申请内存的问题？解决方法是，定时器只判断修改标志位（flag，标志位设置为全局变量 global），然后在 while 循环里，根据标志位，做相应的处理。

```
has_data = False # 初始化为没有数据读入

def is_data_coming(timer):
    global uart
    global has_data

    if uart.any():
        has_data = True
    else:
        has_data = False

timer = Timer(4)
timer.init(freq=10)
timer.callback(led_on_off)

while True:
    # 这里只延时 啥也不干
    if has_data:
        data = uart.readline()
        print(data)
```

5.4、UART 串口通信

更全面请看：<http://docs.micropython.org/en/latest/pyboard/library/pyb.UART.html?highlight=uart>

读写操作：

`uart.read(10)`：读入 10 个字符，返回一个比特对象

`uart.read()`：读取所有的有效字符

`uart.readline()`：读入一行

`uart.readinto(buf)`：读入并且保存在缓存中

`uart.write('abc')`：向串口写入 3 个字符 abc

单个字符的读取与写入：

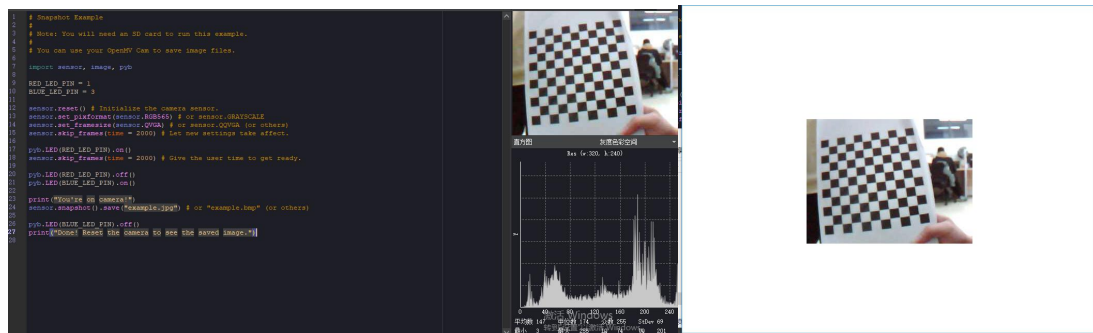
`uart.readchar()`：读入一个字符

`uart.writechar(42)`：写入 ASCII 码为 42 的字符

`uart.any()`：判断串口是否有数据

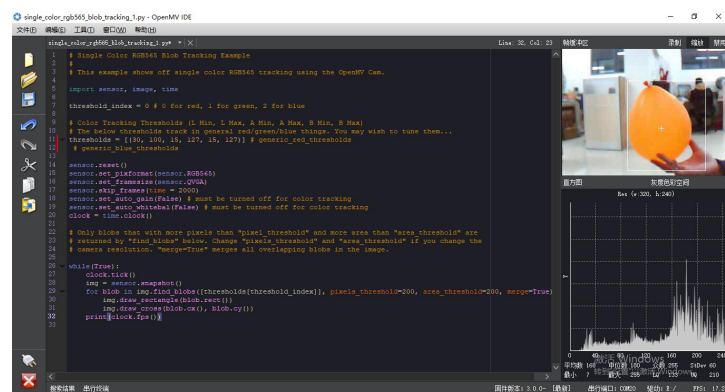
5.5、拍照

当我们要进行摄像头参数的标定时，我们需要使用 openmv 拍摄棋盘图，故使用的例程为 `Snapshot-snapshot.py`，本例程的目标是使用 `save` 函数保存摄像头图片。**注意：**因为 openmv 内存较小，需要外接 SD 卡才能保存图片哦。



5.6、颜色追踪

本例程为 `10-Color_Ttracking-blob_detection` 本例程的目标是用 OpenMV 实现颜色识别。openmv 可以多个颜色同时识别。



例如：`green_threshold = (0, 80, -70, -10, -0, 30)` 设置绿色的阈值，括号里面的数值分别是 LAB 的最大值和最小值 (minL, maxL, minA, maxA, minB, maxB)，LAB 的值在图像左侧三个坐标图中选取。如果是灰度图，则只需设置 (min, max) 两个数字即可。

`sensor.reset()`

`sensor.set_pixformat(sensor.RGB565)`

`sensor.set_framesize(sensor.QQVGA)`

```
sensor.skip_frames(10)
```

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()
```

```
while(True):
```

```
    clock.tick()
```

```
    img = sensor.snapshot()
```

```
    blobs = img.find_blobs([green_threshold]) #find_blobs(thresholds, invert=False, roi=Auto), thresholds 为颜色阈值，是一个元组，需要用括号 [] 括起来。invert=1, 反转颜色阈值，invert=False 默认不反转。roi 设置颜色识别的视野区域，roi 是一个元组，roi = (x, y, w, h)，代表从左上顶点(x, y)开始的宽为w 高为h的矩形区域，roi 不设置的话默认为整个图像视野。这个函数返回一个列表，[0]代表识别到的目标颜色区域左上顶点的 x 坐标，[1] 代表左上顶点 y 坐标，[2 代表目标区域的宽，[3] 代表目标区域的高，[4] 代表目标区域像素点的个数，[5] 代表目标区域的中心点 x 坐标，[6] 代表目标区域中心点 y 坐标，[7] 代表目标颜色区域的旋转角度（是弧度值，浮点型，列表其他元素是整型），[8] 代表与此目标区域交叉的目标个数，[9] 代表颜色的编号（它可以用来分辨这个区域是用哪个颜色阈值 threshold 识别出来的）。
```

```
    if blobs:如果找到了目标颜色
```

```
        for b in blobs:迭代找到的目标颜色区域
```

```
            img.draw_rectangle(b[0:4])用矩形标记出目标颜色区域
```

```
            img.draw_cross(b[5], b[6])在目标颜色区域的中心画十字形标
```

记

```
            print(b[5], b[6])输出目标物体中心坐标
```

```
    print(clock.fps())
```

5.7 腐蚀与膨胀

腐蚀操作首先对图像进行分割，二值化，将在阈值内的区域变为白色，阈值外区域变为黑色，再对图像边缘进行侵蚀，侵蚀函数 `erode(size,`

threshold=Auto), size 为 kernal 的大小, 去除边缘相邻处多余的点。threshold 用来设置去除相邻点的个数, threshold 数值越大, 被侵蚀掉的边缘点越多, 边缘旁边白色杂点少; 数值越小, 被侵蚀掉的边缘点越少, 边缘旁边的白色杂点越多。

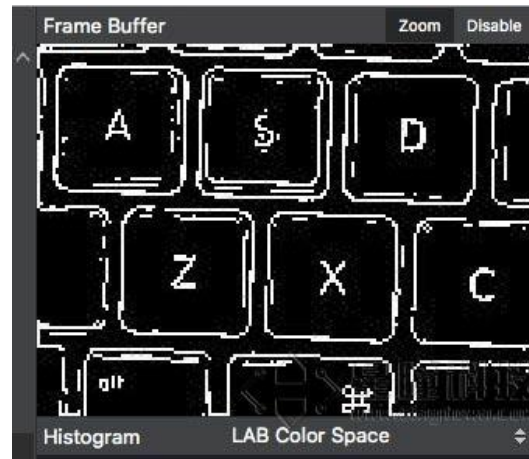
膨胀操作对图像边缘进行膨胀, 膨胀函数 image.dilate(size, threshold=Auto), size 为 kernal 的大小, 使边缘膨胀。threshold 用来设置去除相邻点的个数, threshold 数值越大, 边缘越膨胀; 数值越小, 边缘膨胀的小。具体代码如下所示。

```
while(True):  
    sensor.set_pixformat(sensor.GRAYSCALE)  
    for i in range(20):  
        img = sensor.snapshot()  
        img.binary([grayscale_thres])  
    for i in range(20):  
        img = sensor.snapshot()  
        img.binary([grayscale_thres])  
        img.dilate(2)  
    sensor.set_pixformat(sensor.RGB565)  
    for i in range(20):  
        img = sensor.snapshot()  
        img.binary([rgb565_thres])  
        img.erode(2)  
    for i in range(20):  
        img = sensor.snapshot()  
        img.binary([rgb565_thres])  
        img.dilate(2)
```

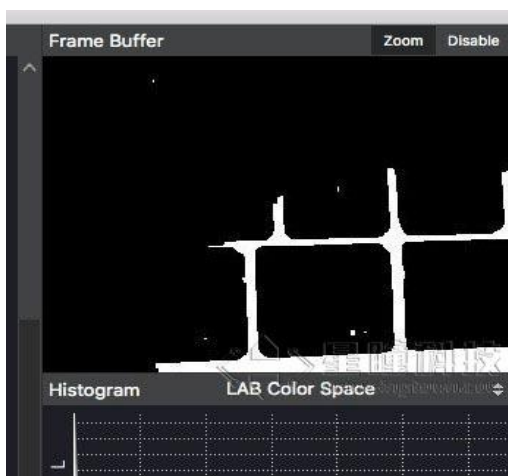
实现效果如下所示。



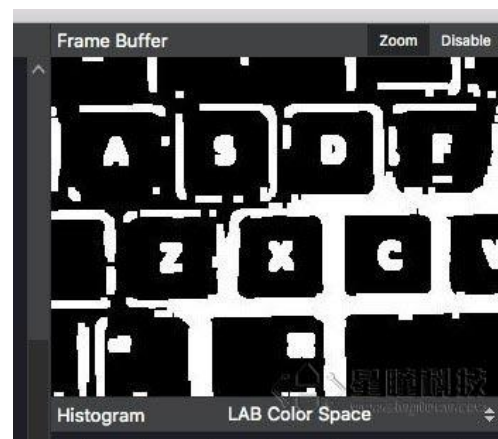
原图



边缘检测后的原图



erode 函数腐蚀后的原图



dilate 函数膨胀后的原图

膨胀与腐蚀

http://blog.csdn.net/poem_qianmo/article/details/23710721

开运算、闭运算、形态学梯度、顶帽、黑帽合辑

http://blog.csdn.net/poem_qianmo/article/details/24599073

5.8 条形码和二维码检测

条形码检测可以在 OpenMV Cam 的 OV7725 相机模块的 640x480 分辨率下运行。条形码检测也将在 RGB565 模式下工作，但分辨率较低。也就是说，条形码检测需要更高的分辨率才能正常工作，因此应始终以 640x480 的灰度运行：

<http://book.openmv.cc/example/16-Codes/find-barcodes.html>

二 维 码 检 测 :

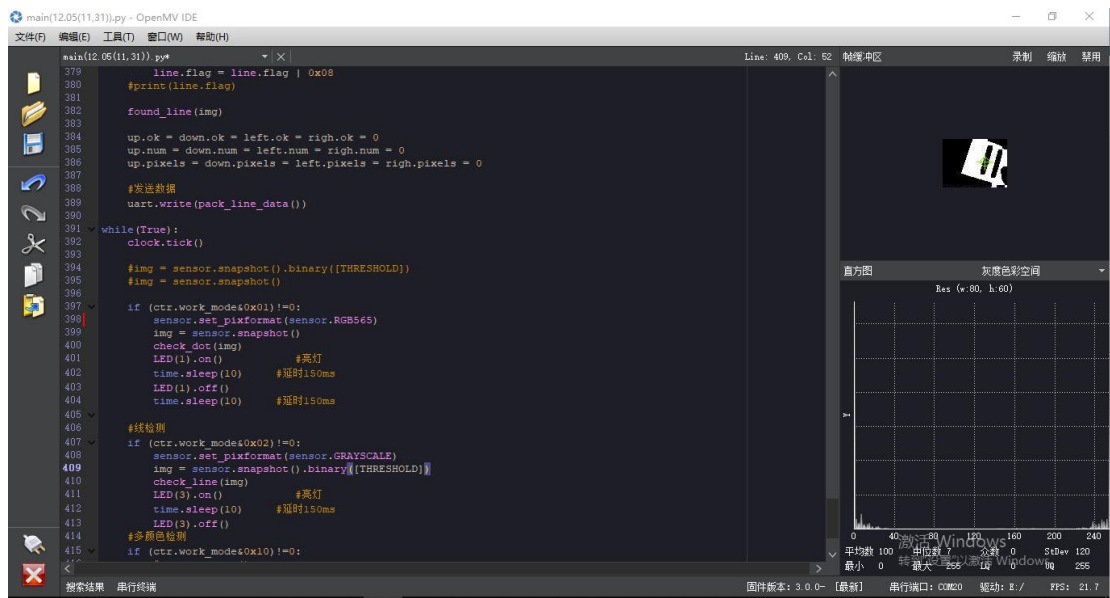
<http://book.openmv.cc/example/16-Codes/qrcodes-with-lens-zoom.html>

六、电赛教程

6.1、黑点或色块检测

色块坐标输出思路：先初始化摄像头等传感器模块，然后利用前面介绍的阈值编辑器调节需要识别色块的阈值大小，再调用用 `img.find_blobs()` 查找色块，找到不同的色块后，使用 `dot.pixels < blob.pixels()` 比较得出最大的色块像素点，最后调用 `blob.cx()`，`blob.cy()` 得出 `x, y` 的坐标，将得到的坐标值减去整个图像的中点像素坐标值即可得到需要色块的相对于图像中心的坐标偏差值。再用 `uart.write(pack_dot_data())`（串口发送详见代码部分）函数发送黑点或者色块的坐标，部分代码如下：

```
#点检测函数
def check_dot(img):
    for blob in img.find_blobs(thresholds, pixels_threshold=150, area_threshold=150, merge=True, margin=5):
        if dot.pixels < blob.pixels(): #寻找最大的黑点
            img.binary(thresholds)
            img.erode(2)
            dot.pixels = blob.pixels()
            dot.x = blob.cx()
            dot.y = blob.cy()
            dot.ok = 1
            img.draw_cross(dot.x, dot.y, color=127, size=10)
            img.draw_circle(dot.x, dot.y, 5, color=127)
```





点检测数据解读:

Bit1	AA	帧头
Bit2	AF	帧头
Bit3	F2	点数据标志位
Bit4	07	有效数据位
Bit5	00	Dot.x 坐标高八位
Bit6	3B	Dot.x 坐标低八位
Bit7	00	Dot.y 坐标高八位
Bit8	06	Dot.x 坐标低八位
Bit9	0C	像素数高八位
Bit10	F2	像素数低八位
Bit11	01	点数据标志位 (有数据为 1, 没有数据为 0)
Bit12	00	保留

1. dot_x : 点的 X 坐标 (数据范围: 0-80, 单位: 像素点)。
2. dot_y : 点的 Y 坐标 (数据范围: 0-40, 单位: 像素点)。
3. flag: 是否检测到点 (1: 是, 0: 否)。

7.2、无人机巡线

巡线部分使用霍夫变换进行线段的拟合,然后将图像部分分成4个 roi 区域,根据标志位来判断直线在每个区域的位置,来判断线段是一条线还是两条,是横线还是竖线。如果是两条线,是十字相交还是 T 字相交或者是 7 字相交。霍夫换巡线如下所示。

```
def found_line(img):
    singleline_check.flager = img.get_regression([(255,255)], robust = True)
    if (singleline_check.flager):
        #print(clock.fps())
        singleline_check.rho_err = abs(singleline_check.flager.rho()-0)
        if singleline_check.flager.theta()>90:
            singleline_check.theta_err = singleline_check.flager.theta()-0
        else:
            singleline_check.theta_err = singleline_check.flager.theta()-0
        img.draw_line(singleline_check.flager.line(), color = 127)
        print(singleline_check.theta_err)

def check_line(img):
    fine_border(img,up,up_roi)
    fine_border(img,down,down_roi)
    fine_border(img,left,left_roi)
    fine_border(img,right,right_roi)

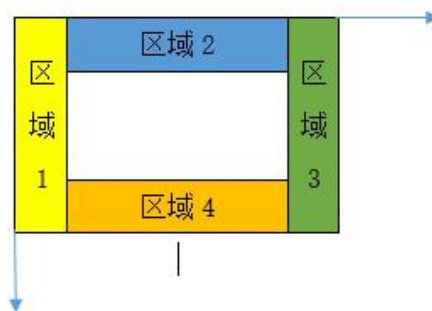
    line.flag = 0
    if up.ok:
        line.flag = line.flag | 0x01
    if down.ok:
        line.flag = line.flag | 0x02
    if left.ok:
        line.flag = line.flag | 0x04
    if right.ok:
        line.flag = line.flag | 0x08
    #print(line.flag)

    found_line(img)

    up.ok = down.ok = left.ok = right.ok = 0
    up.num = down.num = left.num = right.num = 0
    up.pixels = down.pixels = left.pixels = right.pixels = 0

    #发送数据
    uart.write(pack_line_data())
```

4 个 roi 区域的划分图如下所示:



线检测数据解读:

Bit1	AA	帧头
Bit2	AF	帧头

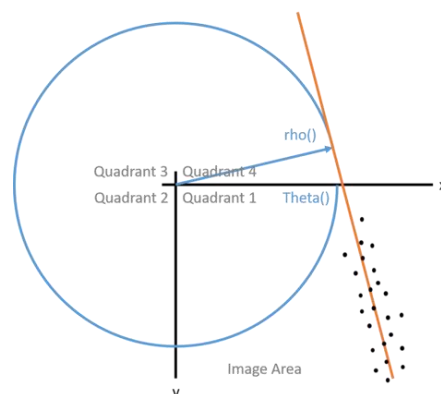
Bit3	F3	线数据标志位
Bit4	07	有效数据位
Bit5	00	singleline_check.rho_err 高八位
Bit6	3B	singleline_check.rho_err 低八位
Bit7	00	singleline_check.theta_err 高八位
Bit8	06	singleline_check.theta_err 低八位
Bit9	01	线数据标志位（有数据为 1，没有数据为 0）
Bit10	F2	校验和
Bit11	00	保留
Bit12	00	保留

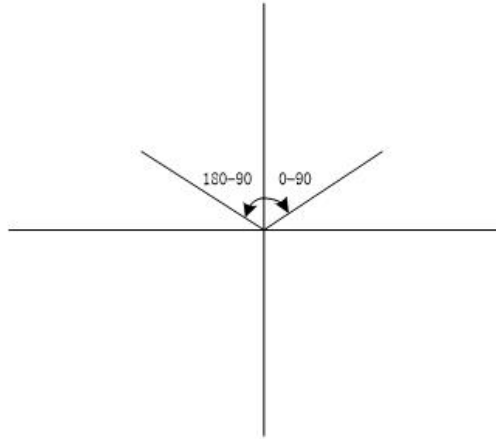
Flag 标志位各位描述：

bit7-4	保留
bit3	1: 有线穿过视窗右边 0: 没线穿过视窗右边
bit2	1: 有线穿过视窗左边 0: 没线穿过视窗左边
Bit1	1: 有线穿过视窗下边 0: 没线穿过视窗下边
Bit0	1: 有线穿过视窗上边 0: 没线穿过视窗上边

线检测数据解读：

- 1、singleline_check.rho_err：线偏离中线的偏差（数据范围：0-80，单位像素点）。
- 2、singleline_check.theta_err：线偏离中线的倾角（数据范围：0-180，单位：度）。





3、line.flag: 是否有目标线穿过视窗中上下左右四个边的标志。