**OpenMV 学习记录**

*——本学习记录只针对OpenMV4*

**学习人：陈 浩**

**参考网站：book.openmv.cc**

**学习时间：2022-01-10——2020-1-20**

目录

第一节 pyb模块中的外设……………………………………………………………………….………3

1、时间相关函数……………………………………………………………………………………………..……3

2、LED外设控制……………………………………………………………………………………………………3

3、IO口控制…………………………………………………………………………………………………………4

4、舵机控制…………………………………………………………………………………………………………..5

5、定时器………………………………………………………………………………………………………………6

6、PWM…………………………………………………………………………………………………………………6

7、ADC…………………………………………………………………………………………………………………..7

8、DAC…………………………………………………………………………………………………………………..7

9、UART…………………………………………………………………………………………………………………8

10、SPI…………………………………………………………………………………………………………………..8

11、I2C…………………………………………………………………………………………………………………..9

第二节 串口通信……………………………………………………………………………….……………10

1、python中字符串…………………………………………………………………………………………….10

2、json…………………………………………………………………………………………………………………10

3、python生成json……………………………………………………………………………………………..10

4、其他单片机json模块……………………………………………………………………………………..11

5、串口通信…………………………………………………………………………………………………………11

6、串口通信实用例程…………………………………………………………………………………………12

第三节 图像处理……………………………………………………………………………….……………14

1、sensor模块基本用法……………………………………………………………………………………...14

2、time模块基本用法…………………………………………………………………………………………15

3、图像的基本运算……………………………………………………………………………………………..16

4、图像的统计信息……………………………………………………………………………………………..17

5、画图………………………………………………………………………………………………………………..18

6、寻找色块…………………………………………………………………………………………………………19

7、AprilTag标记跟踪…………………………………………………………………………………………..21

8、模块匹配…………………………………………………………………………………………………………25

9、特征点匹配…………………………………………………………………………………………………….26

10、二维码识别…………………………………………………………………………………………………..28

11、条形码识别…………………………………………………………………………………………………..29

12、圆形识别………………………………………………………………………………………………………30

13、其他例程………………………………………………………………………………………………………31

1）、颜色形状同时识别……………………………………………………………………………...31

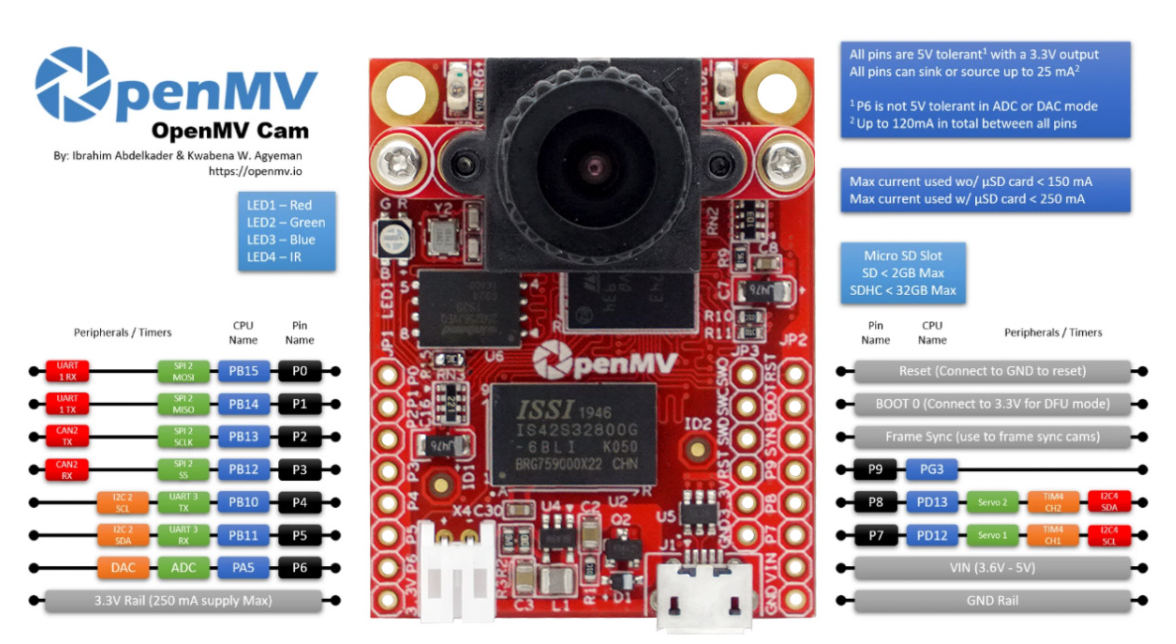
2）、颜色模板匹配同时识别…………………………………………………………………….32

3）、测距…………………………………………………………………………………………………….33

14、线性回归（巡线）………………………………………………………………………………………….34

15、常用函数补充…………………………………………………………………………………………..…37

一、pyb模块中的外设



**1、时间相关函数**

pyb.delay(ms)：延时给定毫秒数

pyb.udelay(us)：延时给定微秒数

pyb.millis()：获取从启动开始计时的毫秒数

**2、LED外设控制**

* RGBLED灯

pyb.LED(1)：红色

pyb.LED(2)：黄色

pyb.LED(3)：蓝色

* 红外LED

pyb.LED(4)

例：

from pyb import LED,delay #导入pyb中LED与delay

red = LED(1) #LED(1)即红灯

green = LED(2)#LED(2)即绿灯

blue = LED(3) #LED(3)即蓝灯

t\_1s = 1000 #定时时间1s

while(True):

#红灯亮1s

red.on() #开红灯

delay(t\_1s)#延时1s

red.off() #关红灯

#绿灯亮1s

green.on() #开绿灯

delay(t\_1s)#延时1s

green.off() #关绿灯

#蓝灯亮1s

blue.on() #开蓝灯

delay(t\_1s)#延时1s

blue.off() #关蓝灯

#黄灯亮1s

red.on() #开红灯

green.on() #开绿灯

delay(t\_1s)#延时1s

red.off() #关红灯

green.off() #关绿灯

#紫灯亮1s

red.on() #开红灯

blue.on() #开蓝灯

delay(t\_1s)#延时1s

red.off() #关红灯

blue.off() #关蓝灯

#青灯亮1s

green.on() #开绿灯

blue.on() #开蓝灯

delay(t\_1s)#延时1s

green.off() #关绿灯

blue.off() #关蓝灯

#白灯亮1s

red.on() #开红灯

green.on() #开绿灯

blue.on() #开蓝灯

delay(t\_1s)#延时1s

red.off() #关红灯

green.off() #关绿灯

blue.off() #关蓝灯

**3、IO口控制（P0~P9）**

* 输出引脚

p0\_out = pyb.Pin(‘P0’,pyb.Pin.OUT\_PP) #设置p0\_out为输出引脚

p0\_out.high() #设置p0\_out引脚输出高电平

p0\_out.low() #设置p0\_out引脚输出低电平

* 输入引脚

p1\_in = pyb.Pin(‘P1’,pyb.Pin.IN,pyb.Pin.PULL\_UP) #设置p1\_in为输出引脚

value = p1\_in.value() #读取p1\_in引脚电平1（3.3V）或0（0V）

例：

from pyb import Pin,LED,delay #导入pyb中Pin,LED,delay模块

p0\_out = Pin('P0',Pin.OUT\_PP) #设置P0引脚为输出引脚

p1\_in = Pin('P1',Pin.IN,Pin.PULL\_UP) #设置P1引脚为输入引脚

red = LED(1)

green = LED(2)

while(True):

#P0作输出引脚

p0\_out.high() #P0引脚输出1（3.3V）

delay(1000)#延时1s

p0\_out.low() #P0引脚输出0（0V）

delay(1000)#延时1s

#P1作输入引脚

value = p1\_in.value()#读取P1引脚电平1（3.3V）或0（0V）

print(value)

if value == 1: #P1为高电平时，打开红灯

green.off()

red.on()

elif value == 0: #P1为低电平时，打开绿灯

red.off()

green.on()

**4、舵机控制（P7、P8）**

s1 = pyb.Servo(1) #舵机1信号线，即P7引脚

s2 = pyb.Servo(2) #舵机2信号线，即P8引脚

s1.angle(45) #舵机1转动角度控制，范围-90°~90°

s2.angle(45) #舵机2转动角度控制，范围-90°~90°

例：

from pyb import Servo,delay #导入pyb中Servo,delay模块

s1 = Servo(1) #舵机1信号线，即P7引脚

s2 = Servo(2) #舵机2信号线，即P8引脚

while(True):

s1.angle(0) #舵机1转动0度

delay(500) #延时0.5s

s1.angle(45)#舵机1转动45度

delay(500) #延时0.5s

s1.angle(90)#舵机1转动90度

delay(500) #延时0.5s

s2.angle(0)

delay(500)

s2.angle(-45)

delay(500)

s2.angle(-90)

delay(500)

s1.angle(0) #s1复位

s2.angle(0) #s2复位

**5、定时器**

* 选择定时器，并设置触发频率  
  tim2 = Timer(2,freq = 5) #选择定时器2，并以5hz触发，即每200ms触发一次
* 相应定时器中断，在中断函数中调用中断函数

tim2.callback(tick) #触发定时器2中断，调用tick函数

例子：

from pyb import Pin,Timer,LED,delay #引入pyb中Pin,Timer,LED,delay

blue = LED(3) #选择RGB灯中blue色

def tick(timer):

blue.toggle() #切换LED的开关，若LED已开则关闭，若LED已关#则打开

tim2 = Timer(2,freq = 5)#选择定时器2，并以5hz触发，即每200ms触发一#次

tim2.callback(tick) #触发定时器2中断，调用tick函数

while(True):

delay(1000)

**6、PWM**

* ch1 = tim4.channel(1,Timer.PWM,pin = Pin('P7'),pulse\_width\_percent = 10)

1：通道1

Timer.PWM：定时器设置输出为PWM

pin = Pin('P7')：

例：

from pyb import Pin,Timer,delay #引入pyb模块中Pin,Timer,delay

tim4 = Timer(4,freq = 1000) #选择定时器4，并以5hz触发

#P7作通道1，输出占空比为10%的PWM方波

ch1 = tim4.channel(1,Timer.PWM,pin = Pin('P7'),pulse\_width\_percent = 10)

#P8作通道2，输出占空比为20%的PWM方波

ch2 = tim4.channel(2,Timer.PWM,pin = Pin('P8'),pulse\_width\_percent = 20)

while(True):

delay(100)

**7、ADC（模数转换,P6）**

* 设置模数转换引脚

adc\_p6 = ADC(‘P6’)

* 读取相应引脚电平，并转换为数字量

value = adc\_p6.read()

例：

from pyb import Pin,ADC,delay #从pyb模块中引入Pin,ADC,delay

adc\_p6 = ADC('P6') #P6引脚设置为模数转换引脚

#注：只有P6引脚可作模数，数模转换引脚

while(True):

value = adc\_p6.read() #读取P6引脚电压，并转换为数字量，范围#0~4095

voltage = round((value\*3.3)/4095,2) #电压保留小数点后两位

print('ADC = %fV'%voltage) #输出电压

delay(500)

**8、DAC（数模转换，P6）**

* 设置数模转换引脚

dac\_p6 = DAC('P6')

* 输出电压，参数为数字量转换为模拟量

例：

from pyb import DAC,Pin,delay #从pyb模块中引入Pin,DAC,delay

dac\_p6 = DAC('P6') #设置P6引脚为数模转换引脚产生三角波

while(True):

for i in range(256):

dac\_p6.write(i) #输出电压，范围为：0~255，即：0~3.3V

print((i\*3.3)/256)

delay(5)

for i in range(256):

dac\_p6.write(255-i)

print((255-i)\*3.3/256)

delay(5)

**9、UART(UART1、UART3)**

* 配置UART及波特率

uart\_1 = UART(1,9600)

uart\_3 = UART(3,9600)

注：仅有UART1和UART3可用

* UART发送数据

uart\_3.write("Hello World!")

* UART接受字符

uart\_3.read(5) #5意为连续接受五个字节

例：

from pyb import UART,delay #从pyb模块中引入UART,delay

uart\_3 = UART(3,9600) #配置UART3波特率为9600

uart\_1 = UART(1,9600) #配置UART1波特率为9600

while(True): #仅有UART1和UART3可配置

uart\_3.write("Hello World!") #UART3发送字符串

delay(1000)

**10、SPI（SPI2）**

* SPI配置

spi\_2 = SPI(2,SPI.MASTER,baudrate = 200000,polarity = 1,phase = 0)

2：选择SPI2，板子上只有SPI2，因此固定为2

SPI.MASTER：设置为主模式，也可设置为从模式（SPI.SLAVE）

baudrate = 200000：波特率为200000即SCK时钟频率

polarity：可为0或1，表示空闲时时钟线SCK电平

phase：可为0或1，表示在第一或第二时钟沿上采样数据

例：

from pyb import I2C

i2c\_2 = I2C(2,I2C.MASTER,addr = 0x12,baudrate = 400000)

test = i2c\_2.is\_ready(0x12) #test = False或True

i2c\_2.mem\_read(data,0x12,0x00)

**11、IIC**

例：

from machine import I2C, Pin #machine库是软件模拟的I2C协议,所以使用

#任何引脚都可以，但是还是推荐使用上面所说的引脚。

i2c = I2C(sda=Pin('P5'),scl=Pin('P4'))

i2c.scan()

i2c.writeto(0x42, b'123')#向0X42地址中依次写入二进制1、2、3三个字节

i2c.readfrom(0x42, 4) # 从0X42地址依次读出三个字节数据

i2c.readfrom\_mem(0x42, 8, 3)# read 3 bytes from memory of slave 42,

# starting at memory-address 8 in the slave

i2c.writeto\_mem(0x42, 2, b'\x10')# write 1 byte to memory of slave 42

# starting at address 2 in the slave

二、串口通信

**1、python中字符串**

* python中字符串str = “Hello World!” #” “或 ‘ ‘都可以，’’’ ‘’’表示多行字符串
* OpenMV可以直接通过串口发送字符串

例：

from pyb import UART

uart = UART(3, 9600)

str = “Hello World!”

uart.write(str)

**2、json**

json是一种简洁高效的数据交换格式，如：

"[[12,0], [10,12], [22,10], [99,11]]"

可以是这种简单的，把OpenMV采集到的色块x，y坐标发送出去

'''

{

"number":10,

"color" :[255,0,0],

"rate" :0.65

}

'''

也可以是这种复杂的

**3、python中json模块**

OpenMV中由json模块，

可以用json.dumps(obj)和json.loads(str)生成json字符串和解析json字符串

例：

import json

obj\_1 = [[12,0], [10,12], [22,10], [99,11]]

print(json.dumps(obj\_1))

obj\_2 = {

"number":10,

"color" :[255,0,0],

"rate" :0.65

}

print(json.dumps(obj\_2))

输出：

'[[12, 0], [10, 12], [22, 10], [99, 11]]'

'{"color": [255, 0, 0], "number": 10, "rate": 0.65}'

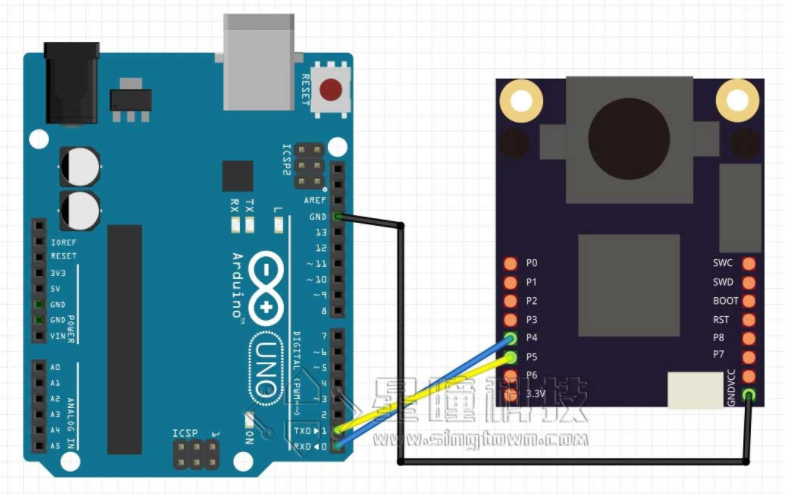
**4、其他单片机的json模块**

STM32: <http://blog.csdn.net/yannanxiu/article/details/52712723>

NodeMCU(esp8266): <https://nodemcu.readthedocs.io/en/master/en/modules/sjson/>

**5、串口通信**

TTL串口至少需要3根线：TXD，RXD，GND。TXD是发送端，RXD是接收端，GND是地线。 连线的时候，需要把OpenMV的RXD连到另一个MCU的TXD，TXD连到RXD。



例：

import pyb #引入pyb模块

uart\_1 = pyb.UART(1,9600) #配置UART 1波特率为9600

uart\_3 = pyb.UART(3,9600) #配置UART 3波特率为9600

while(True):

uart\_1.write("Hello") #UART 1发送数据"Hello"

uart\_3.write("World") #UART 3发送数据"World"

pyb.delay(1000) #延时1s

注意：OpenMV4的UART有两个，分别是UART1和UART3

**6、串口通信实用例程**

传输复杂数据（参考即可）：

import sensor, image, time

from pyb import UART

import json

yellow\_threshold = (65, 100, -10, 6, 24, 51)

sensor.reset()

sensor.set\_pixformat(sensor.RGB565)

sensor.set\_framesize(sensor.QQVGA)

sensor.skip\_frames(10)

sensor.set\_auto\_whitebal(False)

clock = time.clock()

uart = UART(3, 115200)

while(True):

img = sensor.snapshot()

blobs = img.find\_blobs([yellow\_threshold])

if blobs:

print('sum :', len(blobs))

output\_str = json.dumps(blobs)

for b in blobs:

img.draw\_rectangle(b.rect())

img.draw\_cross(b.cx(), b.cy())

print('you send:',output\_str)

uart.write(output\_str+'\n')

else:

print('not found!')

传输精简数据（实用代码）：

import sensor, image, time

from pyb import UART

import json

yellow\_threshold = (65, 100, -10, 6, 24, 51)

sensor.reset()

sensor.set\_pixformat(sensor.RGB565)

sensor.set\_framesize(sensor.QQVGA)

sensor.skip\_frames(10)

sensor.set\_auto\_whitebal(False)

clock = time.clock()

uart = UART(3, 115200)

def find\_max(blobs):

max\_size=0

for blob in blobs:

if blob.pixels() > max\_size:

max\_blob=blob

max\_size = blob.pixels()

return max\_blob

while(True):

img = sensor.snapshot()

blobs = img.find\_blobs([yellow\_threshold])

if blobs:

max\_blob=find\_max(blobs)

print('sum :', len(blobs))

img.draw\_rectangle(max\_blob.rect())

img.draw\_cross(max\_blob.cx(), max\_blob.cy())

output\_str="[%d,%d]" % (max\_blob.cx(),max\_blob.cy()) #方式1

#output\_str=json.dumps([max\_blob.cx(),max\_blob.cy()]) #方式2

print('you send:',output\_str)

uart.write(output\_str+'\r\n')

else:

print('not found!')

二、图像处理

**1、sensor模块基本用法**

* 初始化感光元件：sensor.reset()
* 设置照片格式：sensor.set\_pixformat(参数)

参数：sensor.RGB565（彩色）

sensor.GRAYSCALE（灰色）

* 设置图像大小：sensor.set\_framesize(参数)

参数：sensor.QQQQVGA（40\*30）

sensor.QQQVGA（80\*60）

sensor.QQVGA（160\*120）

sensor.QVGA（320\*240）

sensor.VGA（640\*480）

* 跳过几帧图像或跳过几秒图像：sensor.skip\_frames([n,time])

通过指定n，跳过n张照片，sensor.skip\_frames(10)

通过指定time，跳过time毫秒的帧，sensor.skip\_frames(time = 1000)

若都不指定，sensor.skip\_frames()，则默认跳过300毫秒的帧

* 拍摄一张照片，返回一个image对象：sensor.snapshot()

img = sensor.snapshot()

* 自动增益控制：sensor.set\_auto\_gain(True/False)

sensor.set\_auto\_gain(True) #打开自动增益

sensor.set\_auto\_gain(False) #关闭自动增益

在使用颜色追踪时，需关闭自动增益

* 自动白平衡控制：sensor.set\_auto\_whitebal(True/False)

sensor.set\_auto\_whitebal(True) #打开自动白平衡

sensor.set\_auto\_whitebal(False) #关闭自动白平衡

在使用颜色追踪时，需关闭自动白平衡

* 自动曝光控制：sensor.set\_auto\_exposure(True/False)

sensor.set\_auto\_exposure(True) #打开自动曝光

sensor.set\_auto\_exposure(False) #关闭自动曝光

* 设置窗口ROI（region of interest：感兴趣区）：sensor.set\_windowing(ROI)

sensor.set\_windowing((640,80)) #取中间640\*80区域

* 设置翻转

sensor.set\_hmirror(True) #水平方向翻转

sensor.set\_vflip(True) #垂直方向翻转

例：

import sensor #引入sensor感光元件模块，sensor意为传感器

from pyb import delay

import time #引入time模块

sensor.reset() #初始化感光元件

sensor.set\_pixformat(sensor.RGB565)#设置找照片格式为彩色

sensor.set\_framesize(sensor.QVGA) #设置图像大小为QVGA格式

#########以下代码一般可不加########

#sensor.set\_auto\_gain(True) #打开自动增益（颜色识别时需关闭）

#sensor.set\_auto\_whitebal(True) #打开白平衡（颜色识别时需关闭）

#sensor.set\_auto\_exposure(True) #打开自动曝光

#sensor.set\_windowing((100,100))#选取感兴趣区

#sensor.set\_hmirror(False) #取消水平翻转

#sensor.set\_vflip(False) #取消垂直翻转

################################

sensor.skip\_frames(time = 1000)#跳过启动后1秒内照片

clock\_1 = time.clock() #构造时钟

while(True):

clock\_1.tick() #开始追踪运行时间

img = sensor.snapshot() #拍摄一张照片

print(clock\_1.fps(),clock\_1.avg()) #停止追踪运行时间并返回fps和#平均运行时间

**2、time模块基本用法**

* 延时ms毫秒：time.sleep(ms) #等价于pyb.delay(ms)
* 构造时钟：clock\_1 = time.clock()  
   clock\_1.tick() #开始追踪运行时间

fps = clock\_1.fps() #停止追踪运行时间并返回当前FPS（帧数）

time\_ms = clock\_1.avg() #停止追踪运行时间并换回当前平均运行时间（ms）

例：

import sensor #引入sensor感光元件模块，sensor意为传感器

from pyb import delay

import time #引入time模块

sensor.reset() #初始化感光元件

sensor.set\_pixformat(sensor.RGB565)#设置找照片格式为彩色

sensor.set\_framesize(sensor.QVGA) #设置图像大小为QVGA格式

sensor.skip\_frames(time = 1000) #跳过启动后1秒内照片

clock\_1 = time.clock() #构造时钟

while(True):

clock\_1.tick() #开始追踪运行时间

img = sensor.snapshot() #拍摄一张照片

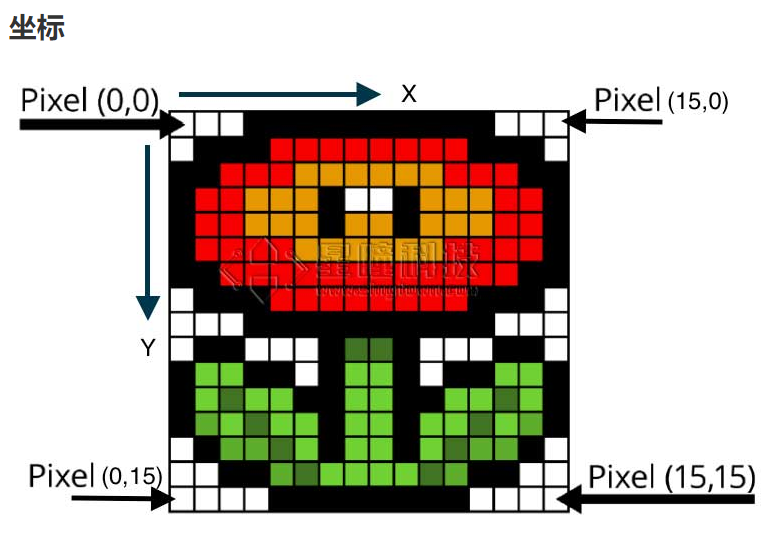
print(clock\_1.fps(),clock\_1.avg()) #停止追踪运行时间并返回fps

#和平均运行时间

**3、图像的基本运算（使用前需引入image模块）**

注意：此处的图像为img = sensor.snapshot()获取的图像

* 图像的坐标



* 获取/设置像素点

获取像素点：img.get\_pixel(x,y)

对于灰度图，返回（x,y）坐标的灰度值

对于彩色图，返回（x,y）坐标的（R,G,B）的元组

设置像素点：img.set\_pixel(x,y,pixel)

对于灰度图，设置（x,y）坐标的灰度值

对于彩色图，设置（x,y）坐标的（R,G,B）值

* 获取图像的宽度、高度及大小

获取图像宽度：img.width()

获取图像高度：img.height()

获取图像大小：img.size() #单位为Byte

* 图像的运算

图像二值化：img.invert()

例：

import sensor #引入sensor（感光元件）库

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565) #设置照片格式为RGB565

sensor.set\_framesize(sensor.QVGA) #设置图像大小为QVGA

sensor.skip\_frames(time = 1000) #跳过1秒内帧数

while(True):

img = sensor.snapshot() #拍摄一张照片

img.invert() #图像二值化

img.set\_pixel(0,0,(255,0,0)) #设置坐标（0，0）处像素为红色（255，#0，0）

print(img.get\_pixel(0,0)) #获取坐标（0，0）处像素点

print(img.width()) #获取图像宽度

print(img.height())#获取图像高度

print(img.format())#获取图像格式

print(img.size()) #获取图像大小

**4、图像的统计信息（statistics：统计）（使用前需引入image模块）**

* 获取图像统计信息：statistics = img.get\_statistics(roi = (x,y,w,h))

x,y：roi（感兴趣区）左上角坐标

w：ROI的宽度

h：ROI的高度

* 对于灰度（GRAYSCALE）照片：

statistics.mean()：返回灰度平均值

statistics.median()：返回灰度中位数

statistics.mode()：返回灰度众数

statistics.stdev()：返回灰度标准差

statistics.min()：返回灰度最小值

statistics.max()：返回灰度最大值

statistics.lq()：返回灰度第一四分数

statistics.uq()：返回灰度第三四分数

* 对于彩色（RGB565）照片：

有L、A、B三个通道

l\_mean()、l\_median()、l\_mode()、l\_stdev()、l\_min()、l\_max()、l\_lq()、l\_uq()

a\_mean()、a\_median()、a\_mode()、a\_stdev()、a\_min()、a\_max()、a\_lq()、a\_uq()

b\_mean()、b\_median()、b\_mode()、b\_stdev()、b\_min()、b\_max()、b\_lq()、b\_uq()

例：

import sensor #引入sensor模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565) #设置照片格式为RGB65

sensor.set\_framesize(sensor.QVGA) #设置照片大小为QVGA

sensor.skip\_frames(time = 1000) #跳过1s内帧数

ROI = (120,160,50,50) #选取（120，160）坐标处宽50高50的感兴趣区

while(True):

img = sensor.snapshot()#拍摄一张图像

Statistics = img.get\_statistics(roi = ROI) #获取感兴趣的统计信息

color\_l = Statistics.mode() #获取感兴趣区L通道上众数

color\_a = Statistics.mode() #获取感兴趣区A通道上众数

color\_b = Statistics.mode() #获取感兴趣区B通道上众数

img.draw\_rectangle(ROI) #在图像上用框画出感兴趣区

print(color\_l,color\_a,color\_b) #打印出L、A、B通道上众数

**5、画图**

* 画线：img.draw\_line(line\_tuple,color = )

line\_tuple：格式（x0，y0，x1，y1），意为（x0，x1）直线，（y0，y1）直线

color：可以是灰度值（0~255），也可以是彩色值（R,G,B）。默认为白色

* 画框：img.draw\_rectangle(rect\_tuple,color = )

rect\_tuple：格式（x，y，w，h），与ROI感兴趣区一致

color：与画线一致

* 画圆：img.draw\_circle(x，y，radius，color = )

x，y：为圆心坐标

radius：为圆半径

color：与画线一致

* 画十字：img.draw\_cross(x，y，size = 5，color = )

x，y：十字中心坐标

size = 5：两寸尺寸

color：与画线一致

* 写字：img.draw\_string(x，y，text，color = )

x，y：写字初始坐标

text：要写的字符串。使用\n使光标移动到下一行

color：与画线一致

例：

import sensor #引入sensor模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565) #设置照片格式为RGB565

sensor.set\_framesize(sensor.QVGA) #设置图像大小为QVGA

sensor.skip\_frames(time = 1000) #跳过1s内帧数

while(True):

img = sensor.snapshot() #拍摄一张图片

#在(10,10)和(30,10)之间画红线

img.draw\_line((10,10,30,10),color = (255,0,0))

#在点(20,20)处画20\*20绿框

img.draw\_rectangle((20,20,20,20),color = (0,255,0))

#在圆心(70,70)处画半径为30蓝圆

img.draw\_circle(70,70,30,color = (0,0,255))

#在坐标(100,100)处画尺寸为15的黄色十字

img.draw\_cross(100,100,size = 15,color = (255,255,0))

#在坐标(120,120)处写紫色的字，#“chenhao”,并换行写“hahaha”

img.draw\_string(120,120,"chenhao\nhahaha",color = (255,0,255))

**6、寻找色块（find\_blobs函数）**

* for blobs in img.find\_blobs(thresholds，roi = ROI，x\_stride = 2，y\_stride = 1，invert = False，area\_threshold = 10，pixels\_threshold = 10，merge = False，margin = 0)
* thresholds：颜色阈值，为一个列表[l\_min,l\_max,a\_min,a\_max,b\_min,b\_max]
* roi = ROI：ROI为感兴趣区，格式为[x，y，w，h]，若选择感兴趣区，则在感兴趣区进行寻找色块，若不指定感兴趣区则在整个图像上进行寻找色块。
* x\_stride = 2：查找色块x方向上最小宽度像素，默认为2
* y\_stride = 1：查找色块y方向上最小宽度像素，默认为1
* invert = False：反转阈值，把阈值意外颜色作为阈值进行查找，如：查找红色意外色块，则在设置颜色阈值时设置为红色，设置inverse = True。
* are\_threshold = 10：面积阈值，若色块被框起来的面积小于这个值，则会被过滤掉
* pixels\_threshold = 10：像素个数阈值，若色块像素个数小于这个值则会被过滤掉。
* merge = False：合并，若merge = True则合并所有颜色阈值为一个，问题：当设置merge = True时，openmv斜对色块功能正常，正对三个色块无法合并
* margin = 0：边界，若设置为5，则两个相同颜色且间距小于5的色块，会被合并，即默认为一个色块。
* 返回参数blobs，即img.find\_blobs函数返回的参数

blobs为一个列表包含多个blob，注：blobs为包含多个色块的列表，blob为某一色块的列表，即blobs = [[blob]，[blob]， [blob]……]

* blob列表中内容

blobs.rect() #返回这个色块的外框——矩形元组（x，y，w，h），可直接在

#img.draw\_rectangle()中使用

blobs.x() #返回色块外框的x坐标

blobs.y() #返回色块外框的y坐标

blobs.w() #返回色块外框的宽度w

blobs.h() #返回色块外框的高度h

blobs.pixels()#返回色块的像素数量

blobs.cx() #返回色块外框的中心坐标x

blobs.cy() #返回色块外框的中心坐标y

blobs.rotation()#返回色块的旋转角度（单位为弧度）

blobs.code() #返回一个16bit数字即4为16进制数，每一位会对应每一个阈值。举个例子：blobs = img.find\_blobs([red, blue, yellow], merge=True)

如果这个色块是红色，那么它的code就是0001，如果是蓝色，那么它的code就是0010。注意：一个blob可能是合并的，如果是红色和蓝色的blob，那么这个blob就是0011。

blobs.count() #若merge = True,则返回整个外框内包围色块数；若merge=False，则固定返回1。

blobs.area() #返回色块外框面积，等于w\*h

blobs.density()#返回色块密度，等于色块像素数除以外框面积

例：

import sensor,time,image #引入sensor,time,image模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565) #设置照片格式为RGB565

sensor.set\_framesize(sensor.QVGA) #设置图像大小为QVGA

sensor.set\_auto\_gain(False) #颜色识别必须关闭自动增益

sensor.set\_auto\_whitebal(False) #颜色识别必须关闭白平衡

sensor.skip\_frames() #跳过0.3s内帧数

clock\_1 = time.clock() #构造时钟对象

thresholds = [(30, 100, 15, 127, 15, 127),

(30, 100, -64, -8, -32, 32),

(68, 15, -5, 30, -75, -31)] #红色,绿色,蓝色阈值

ROI = [0,0,320,240] #感兴趣区

#单颜色识别

while(True):

clock\_1.tick() #开始追踪运行时间

img = sensor.snapshot() #拍摄一张照片

#使用find\_blobs函数，并遍历图像中阈值为红色,面积大于200,像素大

#于200的色块

for blob\_n in img.find\_blobs([thresholds[0]],are\_threshold = 200,pixels\_threshold = 200):

img.draw\_rectangle(blob\_n.rect()) #画框,框出色块

img.draw\_cross(blob\_n.cx(),blob\_n.cy())#画中心为框中心的十字

print(blob\_n.code())

print(clock\_1.fps()) #打印当前帧率

#多颜色识别

while(True):

clock\_1.tick() #开始追踪运行时间

img = sensor.snapshot() #拍摄一张照片

#使用find\_blobs函数，并遍历图像中阈值为红色、绿色、蓝色,面积大于200,像素大于200的色块,merge=True意为合并所有颜色阈值为一个

for blob\_n in img.find\_blobs(thresholds,are\_threshold = 200,pixels\_threshold = 200,merge = True):

img.draw\_rectangle(blob\_n.rect()) #画框,框出色块

img.draw\_cross(blob\_n.cx(),blob\_n.cy())#以框中心画十字

print(blob\_n.code())

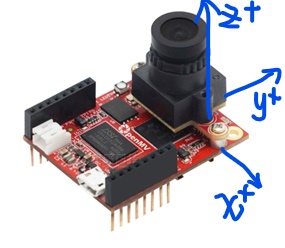
print(clock\_1.fps()) #打印当前帧率

**7、AprilTag标记跟踪（find\_apriltags函数）**





* for apriltags in img.find\_apriltags(roi = ROI,families = img.TAG36H11,fx = f\_x,fy = f\_y,cx = c\_x,cy = c\_y)
* roi = ROI：为感兴趣区，若未指定，则默认为整个图像
* families = img.TAG36H11选择AprilTag种类，若不设置，默认为img.TAG36H11
* fx = f\_x：以像素为单位的相机x方向的焦距
* fy = f\_y：以像素为单位的相机y方向的焦距
* cx = c\_y：图像的x中心位置，即img.width()/2
* cy = c\_y：图像的y中心位置，即img.height()/2
* 返回参数apriltags
* apriltags.corners()#返回图像中AprilTag四个角的坐标元组（x，y）组成的列表，四个角按左上角开始沿顺时针顺序返回
* apriltags.rect()#返回AprilTag外框的矩形元组（x，y，w，h），可直接用于外框的绘制img.draw\_rectangle(apriltags.rect())
* apriltags.x()#返回AprilTag边界框的x坐标
* apriltags.y()#返回AprilTag边界框的y坐标
* apriltags.w()#返回AprilTag边界框的宽度w
* apriltags.h()#返回AprilTag边界框的高度h
* apriltags.id()#返回AprilTag的数字ID，对于常用的AprilTag种类TAG36H11的ID范围为0~586
* apriltags.family()#返回AprilTag种类，返回16则为TAG36H11种类的AprilTags
* apriltags.cx()#返回AprilTag边框中心的x坐标
* apriltags.cy()#返回AprilTag边框中心的y坐标
* apriltags.roration()#返回AprilTag旋转的弧度（单位：弧度）
* apriltags.decision\_margin()#返回AprilTag匹配的色饱和度(0.0~1.0)，1.0最佳
* apriltags.hamming()#返回AprilTag可接受的数位误差数值，对于常用的AprilTag种类TAG36H11最多可接受4位误差
* apriltags.goodness()#返回AprilTag图像的色饱和度(0.0~1.0)，1.0最佳，这个返回值有问题总是为0
* apriltags.x\_translation()#返回距离摄像机x方向的变换，距离的单位未知
* apriltags.y\_translation()#返回距离摄像机y方向的变换，距离的单位未知
* apriltags.z\_translation()#返回距离摄像机z方向的变换，距离的单位未知
* apriltags.x\_rotation()#返回以弧度计的AprilTag在x平面上的旋度
* apriltags.y\_rotation()#返回以弧度计的AprilTag在y平面上的旋度
* apriltags.z\_rotation()#返回以弧度计的AprilTag在z平面上的旋度



* X+：x轴正方向；Y+：y轴正方向；Z+：z轴正方向  
  如已知（x\_translation，y\_translation，z\_translation） = （x0，y0，z0）以及空间实际坐标（x1，y1，z1），则openmv内虚拟坐标系与实际坐标系之间系数为：

令d = sqrt(x1\*\*2 + y1\*\*2 + z1\*\*2)

k = d / sqrt(x0\*\*2 + y0\*\*2 + z0\*\*2)，对于0v5640约等于27.33

例 1：

import sensor,time,image #引入sensor,time模块

sensor.reset() #初始化感光元件

sensor.set\_pixformat(sensor.RGB565)#设置照片格式为RGB565

sensor.set\_framesize(sensor.QQVGA) #设置图像大小为QQVGA(23帧),若设置

#为QVGA则很卡，帧数只有7帧

sensor.set\_auto\_gain(False) #关闭自动增益

sensor.set\_auto\_whitebal(False)#关闭白平衡

sensor.skip\_frames(time = 1000)#跳过1s的帧数

clock\_1 = time.clock() #构造时钟对象

while(True):

clock\_1.tick() #开始追踪运行时间

img = sensor.snapshot() #拍摄一张照片

for apriltags in img.find\_apriltags():

#用红色框框出Apriltags

img.draw\_rectangle(apriltags.rect(),color = (255,0,0))

#在框中央画上绿色十字

img.draw\_cross(apriltags.cx(),apriltags.cy(),color = (0,255,0))

#求出AprilTag旋转角度

degress = apriltags.rotation()\*180/3.14

#打印AprilTag标签的ID；旋转角度；AprilTag四个角坐标，依次为#左上角顺时针取

print(apriltags.id(),degress,apriltags.corners())

print(clock\_1.fps()) #打印当前帧数

例 2：

import sensor,time,image #引入sensor,time模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565)#设置照片格式为RGB565

sensor.set\_framesize(sensor.QQVGA) #设置图像大小为QQVGA(160\*120),若

#设置为QVGA(320\*240)很卡

sensor.skip\_frames(time = 1000) #跳过1秒内不稳定帧数

#设置以像素为单位的相机x方向的焦距，2.8为镜头焦距，3.984为感光元件

#的长，160为图像长度=img.width()

f\_x = (2.8/3.984)\*160

#设置以像素为单位的相机y方向的焦距，2.8为镜头焦距，2.952为感光元件

#的宽，120为图像长度=img.height()

f\_y = (2.8/2.952)\*120

#图像的x中心位置 = img.width()/2

c\_x = 160/2

#图像的y中心位置 = img.height()/2

c\_y = 120/2

clock\_1 = time.clock() #构造时钟对象

#弧度转换为角度函数

def angle(radian):

return radian\*180/3.14

while(True):

clock\_1.tick() #开始追踪运行时间

img = sensor.snapshot() #拍摄一张照片

#使用find\_apriltags函数对图像进行找寻AprilTag

for apriltags in img.find\_apriltags(fx = f\_x,fy = f\_y,cx = c\_x,cy = c\_y):

#用红色框框出AprilTag

img.draw\_rectangle(apriltags.rect(),color = (255,0,0))

#用绿色十字标出AprilTag标签中心

img.draw\_cross(apriltags.cx(),apriltags.cy(),color = (0,255,0))

values = [apriltags.x\_translation(),

apriltags.y\_translation(),

apriltags.z\_translation(),

angle(apriltags.x\_rotation()),

angle(apriltags.y\_rotation()),

angle(apriltags.z\_rotation())]

#返回AprilTag空间中x方向坐标，y方向坐标，z方向坐标，x方向#旋转角度，y方向旋转角度，z方向旋转角度

print(values)

print(clock\_1.fps()) #停止追踪运行时间并返回当前帧数

**8、模块匹配（find\_template函数）**

模块匹配find\_template()采用的是ncc算法，只能匹配与模板图片大小和角度基本一致的图案。视野中的目标图案稍微比模板图片大一些或者小一些就可能匹配不成功。

模板匹配适应于摄像头与目标物体之间距离确定，不需要动态移动的情况。比如适应于流水线上特定物体的检测，而不适应于小车追踪一个运动的排球。

* rect = img.find\_template(template,threshold,roi = ROI,step = 2,search = image.SEARCH\_EX)
* template：一个与这个图像对象相匹配的小图像对象，格式为pgm，图像为灰度（GRALSCALE），bmp~pgm在线转换网址<https://convertio.co/zh/bmp-pgm/>
* threshold：浮点数（0.0~1.0），threshold较小时可提高检测速率但增加误报率，threshold可降低误报率但降低检测速率
* roi = ROI：设置感兴趣区，若不设置默认在全屏进行模块匹配
* step = 2：查找模板时需要跳过的像素数量，跳过像素可大大提高算法运行的速度但该方法只适用于SERACH\_EX模式下的算法。
* search = image.SEARCH\_EX：设置搜索模板所用算法，共有两种算法：image.SEARCH\_EX（更快）和image.SEARCH\_DS（更详尽），一般采用image.SEARCH\_EX
* rect = find\_template()返回参数rect
* 返回的参数rect为（x,y,w,h），可直接用img.draw\_rectangle(rect)框出匹配成功的模块
* **注意：模块匹配仅支持灰度图**

例：

import sensor,time,image #引入sensor,time,image模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.GRAYSCALE)#设置照片格式为GRAYSCALE即灰度

sensor.set\_framesize(sensor.QQVGA)#设置图像大小为QQVGA,QVGA内存会爆

sensor.skip\_frames() #不设置参数，默认跳过300ms内帧数

template\_1 = image.Image("/template\_1.pgm") #模板1“陈”

template\_2 = image.Image("/template\_2.pgm") #模板2“浩”

clock\_1 = time.clock() #构造时钟对象

while(True):

clock\_1.tick() #开始追踪时钟运行时间

img = sensor.snapshot() #拍摄一张照片

rect\_1 = img.find\_template(template\_1,0.7,step = 4,search = image.SEARCH\_EX) #调用模块匹配函数，匹配模块1

if rect\_1:

img.draw\_rectangle(rect\_1) #用框框出模块1

rect\_2 = img.find\_template(template\_2,0.7,step = 4,search = image.SEARCH\_EX) #调用模块匹配函数，匹配模块2

if rect\_2:

img.draw\_rectangle(rect\_2) #用框框出模块2

print(clock\_1.fps()) #打印当前帧数

**9、特征点匹配（find\_keypoints函数，match\_descriptor函数）**

利用 FAST/AGAST 算法进行特征提取，并且进行目标追踪，仅支持灰度图。

程序运行最开始的十秒左右出现的物体作为目标特征，请在程序运行的最开始，将目标物体放在摄像头中央识别，直至出现特征角点，证明已经识别记录目标特征。

匹配过程中，如果画面出现十字和矩形框，证明匹配成功。

**寻找特征点函数**

* keypoints = img.find\_keypoints(max\_keypoints = 150,threshold = 10,scale\_factor = 1.2,normalized = True)
* max\_keypoints = 150：一个键点对象所能容纳的键点最大数量。若键点对象过大导致内存问题，应降低该值。
* threshold = 10：控制提取数量的数字阈值，对于默认的AGAST角点检测器，该值应在20左右。
* scale\_factor = 1.2：比例因子（必须大于1），较高的比例因子运行更快，但其图像匹配相应较差，理想值位于1.35~1.5之间
* normalized = True：若为True，在多分辨率下关闭提取键点。若希望算法运行更快，就将其设置为True。
* img.find\_keypoints函数返回参数keypoints
* keypoints保存目标物体的特征，若keypoints = None，则说明无特征，一般用于关键点初始化

**匹配特征点函数**

* kptmatch = img.match\_descriptor(descriptor0,descriptor1,threshold = 70,filter\_outliers = False)
* descriptor0：被匹配对象
* descriptor1：现匹配对象
* threshold = 70：匹配阈值设置，默认为70
* filter\_outliers = False：为ORB键点过滤异常值服务，默认为False
* kptmatch = img.match\_descriptor函数返回对象kptmatch
* kptmatch.rect()#特征对象的矩形框（x,y,w,h）
* kptmatch.cx()#特征对象的中心x坐标
* kptmatch.cy()#特征对象的中心y坐标
* kptmatch.x()#特征对象矩形框的x坐标
* kptmatch.y()#特征对象矩形框的y坐标
* kptmatch.w()#特征对象矩形框的w宽度
* kptmatch.h()#特征对象矩形框的h高度
* kptmatch.count()#匹配特征点的数量
* kptmatch.theta()#特征对象的旋转角度
* kptmatch.match()#匹配关键点的(x，y)元组列表

**注意：特征点匹配仅支持灰度图（GRAYSCALE）**

例：

import sensor,time,image #引入sensor,time,image模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.GRAYSCALE)#设置照片格式为灰度图

sensor.set\_framesize(sensor.QVGA) #设置图像大小为QVGA

sensor.skip\_frames(time = 1000) #跳过1s内不稳定帧数

clock\_1 = time.clock()#构造时钟对象

keypoint\_1 = None #原匹配对象关键点

#画出关键点

def draw\_keypoints(img,keypoint):

if keypoint:

print(keypoint) #打印出原匹配对象关键点参数

img.draw\_keypoints(keypoint)#画出原匹配对象关键点

img = sensor.snapshot() #拍摄一张照片

time.sleep\_ms(2000) #延时2s

#主程序

while(True):

clock\_1.tick() #构造始终对象

img = sensor.snapshot()#拍摄一张照片

if keypoint\_1 == None:

#采集原匹配对象关键点

keypoint\_1 = img.find\_keypoints(max\_keypoints = 150,threshold = 10,scale\_factor = 1.2)

draw\_keypoints(img,keypoint\_1)

else:

#采集现匹配对象关键点

keypoint\_2 = img.find\_keypoints(max\_keypoints = 150,threshold = 10,scale\_factor = 1.2,normalized = True)

if keypoint\_2:

#原匹配对象与现匹配对象关键点进行匹配

match = image.match\_descriptor(keypoint\_1,keypoint\_2)

if match.count() >= 10:

#用框框出现匹配对象

img.draw\_rectangle(match.rect())

#在现匹配对象中央用十字点出

img.draw\_cross(match.cx(),match.cy())

print(match.count(),match.theta()) #输出现匹配对象匹配

#关键点数以及旋转角度

img.draw\_string(0,0,"fps:%0.2f"%clock\_1.fps())#在图像上显示帧率

**10、二维码识别（find\_qrcodes函数）**

<http://cli.im/>可以用草料二维码生成你想要的内容。

* for qrcodes in img.find\_qrcodes(roi = ROI)
* roi = ROI：设置感兴趣区，若不设置默认为整个图像

注意：为使这一方法成功运行，图像上二维码需比较平展，

使用img.lens\_corr(1.8)函数来消解镜头的桶形畸变或使用无畸变镜头

* for qrcodes in img.find\_qrcodes(roi = ROI)返回参数qrcodes
* qrcodes.corners()#返回一个由该对象的四个角组成的四个元组(x,y)的列表，四个角通常是按照从左上角开始沿顺时针顺序返回的
* qrcodes.rect()#返回二维码边框的矩形元组（x,y,w,h）
* qrcodes.x()#返回二维码边框的x坐标
* qrcodes.y()#返回二维码边框的y坐标
* qrcodes.w()#返回二维码边框的宽度w
* qrcodes.h()#返回二维码边框的长度h
* qrcodes.payload()#返回二维码有效载荷的字符串

例：

import sensor,time,image #引入sensor,time,image模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.GRAYSCALE)#设置照片为灰度图

sensor.set\_framesize(sensor.QVGA) #设置图像格式为QVGA

sensor.skip\_frames() #跳过300ms内帧数

clock\_1 = time.clock() #构造时钟对象

while(True):

clock\_1.tick() #开始追踪运行时间

img = sensor.snapshot()#拍摄一张照片

img.lens\_corr(1.8) #对于二维码需矫正镜头畸变，或使用无畸变镜

#头

#使用find\_qrcodes函数找寻待检测二维码

for qrcodes in img.find\_qrcodes():

img.draw\_rectangle(qrcodes.rect())#用矩形框框出二维码

message = qrcodes.payload() #读取二维码中字符串信息

print(message) #打印字符串中信息

img.draw\_string(0,0,"fps:%0.2f"%clock\_1.fps())#在图像上打印帧率

**11、条形码识别（find\_barcodes函数）**

* for barcodes in img.find\_barcodes(roi = ROI)
* roi = ROI：设置感兴趣区，若不设置默认为整个图像
* for barcodes in img.find\_barcodes(roi = ROI)返回参数barcodes
* barcodes.rect()#返回条形码边框的矩形元组（x,y,w,h）
* barcodes.x()#返回条形码边框的x坐标
* barcodes.y()#返回条形码边框的y坐标
* barcodes.w()#返回条形码边框的宽度w
* barcodes.h()#返回条形码边框的长度h
* barcodes.payload()#返回条形码有效载荷的字符串

**注意：条形识别无需矫正镜头，即无需img.lens\_corr(1.8)**

例：

import sensor,time,image #引入sensor,time,image模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.GRAYSCALE)#设置照片为灰度图

sensor.set\_framesize(sensor.QVGA) #设置图像格式为QVGA

sensor.skip\_frames() #跳过300ms内帧数

clock\_1 = time.clock() #构造时钟对象

while(True):

clock\_1.tick() #开始追踪运行时间

img = sensor.snapshot()#拍摄一张照片

#使用find\_barcodes函数找寻待检测条形码

for barcodes in img.find\_barcodes():

img.draw\_rectangle(barcodes.rect())#用矩形框框出条形码

message = barcodes.payload() #读取条形码中字符串信息

print(message) #打印字符串中信息

img.draw\_string(0,0,"fps:%0.2f"%clock\_1.fps())#在图像上打印帧率

**12、圆形识别（find\_circles）**

* circles = img.find\_circles(roi = ROI,x\_stride = 2,y\_stride = 2,threshold = 2000,x\_margin = 10,y\_margin = 10,r\_margin = 10,r\_min = 2,r\_max = 2,r\_step = 2)
* roi = ROI：设置感兴趣区，若不设置默认为整个图像
* x\_stride = 2：霍夫变换时需要跳过的x像素的数量，若圆较大可增加，默认为2
* y\_stride = 2：霍夫变换时需要跳过的x像素的数量，若圆较大可增加，默认为2
* threshold = 2000：控制从霍夫变换中监测到的圆。只返回大于或等于 threshold 的圆，应用程序的正确的 threshold 值取决于图像
* x\_margin = 10：控制所检测的圆的合并
* y\_margin = 10：控制所检测的圆的合并
* r\_margin = 10：控制所检测的圆的合并
* r\_min = 2：控制检测到的最小圆半径，增加此值可以加速算法，默认为2
* r\_max = 2：控制检测到的最大圆半径，减少此值可以加快算法，默认为 min(roi.w/2, roi.h/2)
* r\_step = 2：控制如何逐步检测半径。默认为2
* 返沪参数circles
* circles.x() #返回圆的x位置
* circles.y() #返回圆的y位置
* circles.r() #返回圆的半径
* circles.magnitude() #返回圆的模

例：

import sensor,image,time #引入sensor,time,image模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565)#设置照片格式为RGB565，灰度图同样可以

sensor.set\_framesize(sensor.QVGA) #设置图像大小为QVGA

sensor.skip\_frames() #跳过300ms内不稳定帧数

clock = time.clock() #构造始终对象

ROI = (0,0,320,160) #设置感兴趣区，即QVGA图像的全屏

while(True):

clock.tick() #开始追踪运行时间

img = sensor.snapshot()#拍摄一张照片

#img.lens\_corr(1.8) #形状识别需矫正镜头，或使用无畸变镜头

#在拍摄的照片中寻找圆

for circle in img.find\_circles(roi = ROI,x\_stride = 2,y\_stride = 2,z\_stride = 2,

threshold = 3500,

x\_margin = 10,y\_margin = 10,z\_margin = 10,

r\_min = 2,r\_max = 320,r\_step = 2):

#若找到圆，则用圆圈出

img.draw\_circle(circle.x(),circle.y(),circle.r())

#若找到圆，打印出圆的半径和模

print(circle.r(),circle.magnitude())

#用黑色在图像上显示当前帧率

img.draw\_string(0,0,"fps:%0.1f"%clock.fps(),color = (0,0,0))

**13、其他例程：**

**1、颜色形状同时识别**

import sensor,time,image #引入sensor,time,image模块

from math import sqrt #从math引入sqrt函数(开平方函数)

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565)#设置图照片格式为RGB565

sensor.set\_framesize(sensor.QVGA) #设置图像大小为QVGA(320\*160)

sensor.skip\_frames() #跳过300ms内初始帧数

clock = time.clock() #构造时钟对象

ROI\_all = (0,0,320,160)#设置感兴趣区，即全屏，若不设置，图像会很卡

while(True):

clock.tick() #开始追踪运行时间

img = sensor.snapshot()#拍摄一张照片

#img.lens\_corr(1.8)

#在图像中寻找圆

for circle in img.find\_circles(roi = ROI\_all,x\_stride = 2,y\_stride = 2,z\_stride = 2,threshld = 3500,

x\_margin = 10,y\_margin = 10,z\_margin = 10,

r\_min = 2,r\_max = 160,r\_step = 2):

#取找到的圆的内的最大正方形

area = (circle.x()-int((sqrt(2)/2)\*circle.r()),

circle.y()-int((sqrt(2)/2)\*circle.r()),

int(sqrt(2)\*circle.r()), int(sqrt(2)\*circle.r()))

#统计最大正方形

statistics = img.get\_statistics(roi = area)

#若最大正方形为红色，则认为该圆为红色，用红色的圆圈出整个圆

if 0<statistics.l\_mode()<100 and 0<statistics.a\_mode()<120 and 0<statistics.b\_mode()<120:

img.draw\_circle(circle.x(),circle.y(),circle.r(),color = (255,0,0))

#否则用矩形框框出该对象

else:

img.draw\_rectangle(area)

#打印当前帧率

print(clock.fps())

**2、颜色模板匹配同时识别**

#注意：本程序识别色块函数返回code有问题，无论什么颜色，总是返回#2！！！

import sensor,time,image #引入sensor,time,image模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565)#设置照片格式为RGB565

sensor.set\_framesize(sensor.QQVGA) #设置图像大小为QQVGA

sensor.set\_auto\_gain(False) #颜色识别需关闭自动增益

sensor.set\_auto\_whitebal(False) #颜色识别需关闭白平衡

sensor.skip\_frames() #跳过300ms内帧数

clock = time.clock() #构造时钟对象

templates = ["/0.pgm","1.pgm"]#匹配模块位置字符串

thresholds = [(30, 100, 15, 127, 15, 127),

(30, 100, -64, -8, -32, 32)] #红色,绿色阈值

while(True):

clock.tick() #开始追踪运行时间

img = sensor.snapshot()#拍摄一张照片

#调用find\_blobs函数寻找面积大于200，像素点大于200的色块

for blob in img.find\_blobs(thresholds,are\_threshold = 200,

pixels\_threshold = 200):

img = img.to\_grayscale() #图像转换为灰度图

#调用模块匹配函数，对模块进行匹配

for template in templates:

rect = img.find\_template(image.Image(template),0.7,

step = 4,search = image.SEARCH\_EX)

if rect:

img.draw\_rectangle(rect) #用矩形框框出匹配成功模块

print(blob.code(),template)#打印对应颜色索引和模块对

#应字符串

img.draw\_string(0,0,"fps:%0.1f"%clock.fps())#在图像上打印当前帧率

**3、测距**

import sensor,time,image #引入sensor,time,image模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565)#设置照片格式为RGB565

sensor.set\_framesize(sensor.QVGA) #设置图像大小为QVGA

sensor.skip\_frames() #跳过0.3s内帧数

clock\_1 = time.clock() #构造时钟对象

red\_threshold = (30, 80, 15, 50, -5, 20)#红色阈值

k = 20\*62 #测距比例系数k = 距离 \* 像素数

#求k方法，让色块在距openmv的20cm处测量其宽度的像素(blob\_n.w())为#62

while(True):

clock\_1.tick() #开始追踪运行时间

img = sensor.snapshot()#拍摄一张照片

#用find\_blobs函数找寻红色色块，且色块面积和像素应大于200

for blob\_n in img.find\_blobs([red\_threshold],are\_threshold = 200,pixels\_threshold = 200):

img.draw\_rectangle(blob\_n.rect()) #用矩形框框出色块

img.draw\_cross(blob\_n.cx(),blob\_n.cy())#在色块中心画十字

length = k / blob\_n.w()#由比例系数k求得色块距openmv长度

print(clock\_1.fps(),length)#打印当前帧率和长度

**14、线性回归（巡线）（get\_regression函数）**

**1）、线性回归函数**

* line = img.find\_regression(thresholds, invert = False, x\_stride = 2, y\_stride = 1, are\_threshold = 10, pixels\_threshold = 10, robust = False)
* thresholds：想要追踪颜色的阈值，对于RGB565图像，阈值为（l\_min, l\_max, a\_min, a\_max, b\_min, b\_max）；对于灰度图像，每个元组需要包含两个值 - 最小灰度值和最大灰度值
* invert = False：反转阈值，若设置为invert = True，则反转阈值
* x\_stride = 2：跳过的x方向像素数（默认即可）
* y\_stride = 1：跳过的y方向像素数（默认即可）
* are\_threshold = 10：面积阈值，如果回归后的边界框区域小于 area\_threshold ，则返回None，即过滤掉（默认即可）
* pixels\_threshold = 10：像素阈值，如果回归后的像素数小于 pixel\_threshold ，则返回None，即过滤掉（默认即可）
* robust = False：robust = False即实用最小二乘法进行，速度较快，但不能处理异常值；robust = True即使用Theil-Sen线性回归算法，速度较慢，但处理异常值性能较好
* 返回参数line
* line.line()：返回直线元组(x1, y1, x2, y2)，可用img.draw\_line(line.line())画线
* line.x1()：返回直线的p1的 x坐标分量
* line.y1：返回直线的p1的 y坐标分量
* line.x2：返回直线的p2 的x分量
* line.y2：返回直线的p2的 y分量
* line.length：返回直线的长度
* line.magnitude()：返回霍夫变换后直线的模，表征线性回归可信度，大于8可信度高
* line.theta()：返回霍夫变换后直线的角度（0~179度）
* line.rho()：返回霍夫变换后直线的p值，即截距

**2）、图像二值化函数**

* img.binary(thresholds, invert = False, zero = False, mask = None, to\_bitmap = False, copy = False)
* thresholds：想要追踪的颜色阈值，对于RGB565图像，thresholds = (l\_min, l\_max, a\_min, a\_max, b\_min, b\_max)；对于灰度图像，每个元组需要包含两个值最小灰度值和最大灰度值
* invert = False：若设置invert = True，则进行反转阈值操作
* 其他参数不用设置即可

例：（自写）

import sensor,time,image #引入sensor,time,image模块

sensor.reset() #重置感光元件

sensor.set\_pixformat(sensor.RGB565)#设置照片格式为RGB565

sensor.set\_framesize(sensor.QQQVGA)#设置图像大小为QQQVGA(提高帧率)

sensor.skip\_frames() #跳过300ms初始帧率

clock = time.clock() #构建时钟对象

thresholds = (5, 70, -23, 15, -57, 0)#线性回归对象的颜色阈值

while(True):

clock.tick() #开始追踪运行时间

#照片二值化，阈值区域为白，其他为黑色

img = sensor.snapshot().binary([thresholds])

#调用线性回归函数，robust = True即使用Theil-Sen线性回归算法，速#度较慢，但处理异常值性能较好

line = img.get\_regression([(100,100)],robust = True)

if line:

#用红色线画出线性回归得到线

img.draw\_line(line.line(),color = (255,0,0))

#打印直线的模，直线的角度，直线的截距

print(line.magnitude(),line.theta(),line.rho())

img.draw\_string(0,0,"fps:%0.2f"%clock.fps())#图像上显示帧率

例：（学习参考）

THRESHOLD = (5, 70, -23, 15, -57, 0)

import sensor, image, time

import car

from pid import PID

rho\_pid = PID(p=0.4, i=0)

theta\_pid = PID(p=0.001, i=0)

sensor.reset()

sensor.set\_vflip(True)

sensor.set\_hmirror(True)

sensor.set\_pixformat(sensor.RGB565)

sensor.set\_framesize(sensor.QQQVGA) # 80x60 (4,800 pixels) - O(N^2)

# max = 2,3040,000.

sensor.skip\_frames(time = 2000)

clock = time.clock()

while(True):

clock.tick()

img = sensor.snapshot().binary([THRESHOLD])

line = img.get\_regression([(100,100)], robust = True)

if (line):

rho\_err = abs(line.rho())-img.width()/2

if line.theta()>90:

theta\_err = line.theta()-180

else:

theta\_err = line.theta()

img.draw\_line(line.line(), color = 127)

print(rho\_err,line.magnitude(),rho\_err)

if line.magnitude()>8:

#if -40<b\_err<40 and -30<t\_err<30:

rho\_output = rho\_pid.get\_pid(rho\_err,1)

theta\_output = theta\_pid.get\_pid(theta\_err,1)

output = rho\_output+theta\_output

car.run(50+output, 50-output)

else:

car.run(0,0)

else:

car.run(50,-50)

pass

#print(clock.fps())

**15、常用函数补充：**

* img = img.to\_grayscale() #将图像转换为灰度图并赋值给原图像
* sensor.snapshot().save(“0.jpg”)

#保存截取到的图片，命名为0，格式为jpg

sensor.snapshot().save(“Test/%s.pgm”%n)

#保存截取到的图片至Test文件夹下，并命名为n.pgm（n可变）

sensor.snapshot().save("Test/t%s/%s.pgm" % (num, n) )

#保存截取到的图片至Test文件夹下的t num文件夹下，num可变，如t1,t2,t3…并命名为n.pgm（n可变）,如0.pgm,1.pgm,2.pgm

* sensor.sleep(True) #感光元件休眠模式，省电！可在调用sensor.sleep(False)清除
* rtc实时时钟

例：

import pyb

rtc = pyb.RTC()

#2022年1月19日周三，18时5分0秒0亚秒

rtc.datetime((2022,1,19,3,18,5,0,0))

while(True):

print(rtc.datetime())

pyb.delay(1000)