127.OpenMv_Cam库函数指引

2025年3月10日 16:35

【1】图像的获取与显示

```
import sensor # 引入感光元件的模块，用于操作摄像头传感器
import time # 引入时间模块，用于控制时间延迟和FPS计算

# 初始化传感器
sensor.reset()  # Reset and initialize the sensor. 重置
sensor.set_pixformat(sensor.RGB565)  # Set pixel format
设置像素格式为RGB565（或者灰度）
sensor.set_framesize(sensor.QVGA)  # Set frame size to
QVGA（320x240分辨率）
sensor.set_vflip(True)  # 垂直方向翻转。根据实际
#！！！重要：不同摄像头是否需要镜像，根据实际情况定。
sensor.set_hmirror(True)  # 水平方向反转。根据实
#！！！重要：不同摄像头是否需要镜像，根据实际情况定。
#****************************如果不需要镜像就注释
sensor.skip_frames(time=2000)  # Wait for se
clock = time.clock()  # Create a clock objec
于追踪FPS（每秒帧数）



while True:
    clock.tick()  # Update the FPS clock. 更
    img = sensor.snapshot()  # Take a pictur
返回图像
    print(clock.fps())  # 输出当前的帧率（FPS）
```

举个栗子

Take a chestnut

- 【max_mindata】YuanXin_to_Color.py
- All_in.py
- Color_FindTest.py
- Color_to_YuanXin.py
- GonChuan_Sai.py
- helloworld_1.py
- Rgb565_color_tracking.py
- untitled_code.py
- YuanXin_FindTest.py
- YuanXin_to_Color.py
- YuZhi_FindTest.py

然后查看摄像头是否需要反转、代码添加到循环之前。

```
# 注意是否有下面两句根据自己摄像头调整
sensor.set_vflip(True)  #垂直方向翻转 根据自己摄像头和模块安装位置调整 ！！！重要不同摄像头是否需要镜像根据实际情况定，如果不
需要镜像需要注释掉
sensor.set_hmirror(True) #水平方向反转 根据自己摄像头和模块安装位置调整 ！！！重要不同摄像头是否需要镜像根据实际情况定，如果
不需要镜像需要注释掉
```

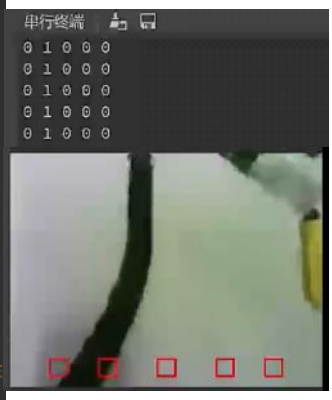
【2】感兴趣区域LAB识别and串行终端输出 openmv视觉循迹

```
import sensor
import pyb

#GROUND_THRESHOLD 阈值参数 通过工具->机器视觉->阈值编辑器
#sensor.set_contrast(1)#设置相机图像对比度。-3至+3。
roi1 = [( 20,  105, 10, 10),
        ( 45,  105, 10, 10),
        ( 75,  105, 10, 10),
        (105,  105, 10, 10),
        (130,  105, 10, 10)]

led = pyb.LED(1)
led.on()

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.set_vflip(True)  #垂直方向翻转
sensor.skip_frames(time=2000)
sensor.set_auto_whitebal(True)#自动白平衡模式
sensor.set_auto_gain(False)#关闭自动增益模式
#lcd.init() #初始化lcd屏幕

GROUND_THRESHOLD=(0, 30, -22, 23, -128, 80)

while(True):
    data=0
    blob1=None
    blob2=None
    blob3=None
    blob4=None
    blob5=None
    flag = [0,0,0,0,0]
    img = sensor.snapshot().lens_corr(strength = 1.7 , zoom = 1.0)#对获取到的图像执行镜头校正
    blob1 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[0])
    blob2 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[1])
    blob3 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[2])
    blob4 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[3])
    blob5 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[4])

    if blob1:#如果roi1区域内找到阈值色块 就会赋值flag[0]为1
        flag[0] = 1
    if blob2:
        flag[1] = 1
    if blob3:
        flag[2] = 1
    if blob4:
        flag[3] = 1
```

上面的代码识别后结果输出在串行终端还是不够直观，我们可以增加输出在屏幕上也显示识别结果

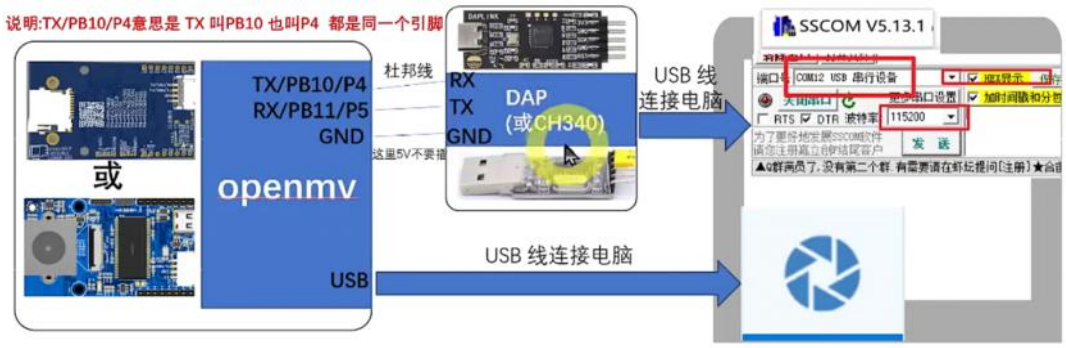
我们通过增加下面代码实现把结果输出在缓冲图像上，这样我们就可以直接通过IDE或者后面通过增加屏幕LCD显示，来直接看到我们识别的结果。

```
    # 遍历所有感兴趣的区域，并绘制矩形框及其识别结果
    for i, rec in enumerate(roi1):
        img.draw_rectangle(rec, color=(255, 0, 0))  # 绘制矩形框
        # 根据flag显示识别结果
        result_text = str(flag[i])  # 显示 1 或 0
        #rec 中的 rec[0] 对应的是矩形框左上角的 横坐标（x），而 rec[1] 对应的是矩形框左上角
的 纵坐标（y）
        text_y_position = rec[1] - 15  # 调整文本显示位置，使其位于矩形框的上方
        img.draw_string(rec[0], text_y_position, result_text, color=(255, 255,
255), scale=2)  # 在矩形框内绘制文本
```



```
    if blob5:
        flag[4] = 1
    print(flag[0],flag[1],flag[2],flag[3],flag[4])#数据打印串行终端

    for rec in roi1:#遍历all感兴趣的区域roi1绘制color=(255,0,0)颜色
        img.draw_rectangle(rec, color=(255,0,0))
        #lcd.display(img)   #图像显示

```

openmv发送数据帧与Stm32通信

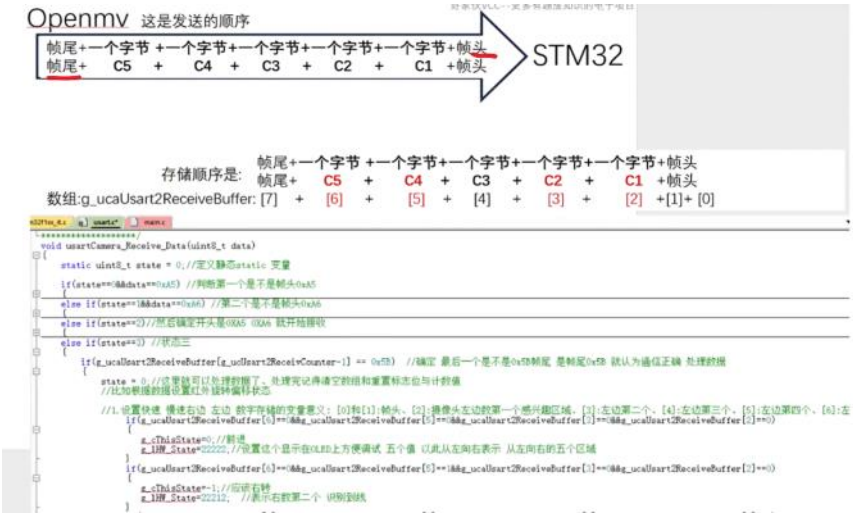



```
import pyb, sensor, image, math, time
from pyb import UART
import ustruct
from image import SEARCH_EX, SEARCH_DS
import time
import sensor, lcd
# 导入所需的库和模块
```

```
uart = UART(3,115200,bits=8, parity=None, stop=1, timeout_char = 1000)#初始化串口
def send_five_uchar(c1,c2,c3,c4,c5):#功能发送五个无符号字符（unsigned char）
    global uart;
    data = ustruct.pack("<BBBBBBBB",#使用了 ustruct.pack() 函数将这些数据打包为二进制格式。使用 "<BBBBBBBB" 作为格式字符串来指定要打包的数据的类型和顺序：
                        0xA5,
                        0xA6,
                        c1,
                        c2,
                        c3,
                        c4,
                        c5,
                        0x5B
                        )
    uart.write(data);#uart.write(data) 将打包好的二进制数据帧写入 UART 发送缓冲区，从而将数据通过串口发送出去
    print(data)#通过 print(data) 打印发送的数据到串行终端，方便调试和确认发送的内容。
    send_five_uchar(flag[0],flag[1],flag[2],flag[3],flag[4])#把五个数据通过串口发送出去、发送五个无符号字符。
```

```python
import pyb
from pyb import UART
import ustruct
import sensor

uart = UART(3,115200,bits=8, parity=None, stop=1, timeout_char = 1000)#初始化串口

roi1 = [( 20,  105, 10, 10),
        ( 45,  105, 10, 10),
        ( 75,  105, 10, 10),
        (105,  105, 10, 10),
        (130,  105, 10, 10)]

led = pyb.LED(1)
led.on()

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.set_vflip(True)  #垂直方向翻转
sensor.skip_frames(time=2000)
sensor.set_auto_whitebal(True)#自动白平衡模式
sensor.set_auto_gain(False)#关闭自动增益模式

GROUND_THRESHOLD=(0, 30, -22, 23, -128, 80)


def send_five_uchar(c1,c2,c3,c4,c5):#功能发送五个无符号字符（unsigned char）
    global uart;
    data = ustruct.pack("<BBBBBBBB",
                        0xA5,
                        0xA6,
                        c1,
                        c2,
                        c3,
                        c4,
                        c5,
                        0x5B
                        )
    uart.write(data);
    print(data)

GROUND_THRESHOLD=(0, 30, -22, 23, -128, 80)

while(True):
    data=0
    blob1=None
    blob2=None
    blob3=None
    blob4=None
    blob5=None
    flag = [0,0,0,0,0]
    img = sensor.snapshot().lens_corr(strength = 1.7 , zoom = 1.0)#对获取到的图像执行镜头校正
    blob1 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[0])
    blob2 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[1])
    blob3 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[2])
    blob4 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[3])
    blob5 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[4])

    if blob1:#如果roi1区域内找到阈值色块 就会赋值flag[0]为1
        flag[0] = 1
    if blob2:
        flag[1] = 1
    if blob3:
        flag[2] = 1
    if blob4:
        flag[3] = 1
    if blob5:
        flag[4] = 1
    send_five_uchar(flag[0],flag[1],flag[2],flag[3],flag[4])

    for i, rec in enumerate(roi1):
        img.draw_rectangle(rec, color=(255,0,0))
        result_text = str(flag[i])
        text_y_position = rec[1] - 15
        img.draw_string(rec[0], text_y_position, result_text, color=(255, 255,
        255), scale=2)

```

Stm32接受发过来的数据帧，并显示在loed上




```
void usartCamera_Receive_Data(uint8_t data)
{
    static uint8_t state = 0;//定义静态static 变量

    if(state==0&&data==0xA5) //判断第一个是不是帧头0xA5
    {
    else if(state==1&&data==0xA6) //第二个是不是帧头0xA6
    {
    else if(state==2)//然后确定开头是0XA5 0XA6 就开始接收
    {
    else if(state==3) //状态三
    {
        if(g_ucaUsart2ReceiveBuffer[g_ucUsart2ReceiveCounter-1] == 0x5B)  //确定 最后一个是不是0x5B帧尾 是帧尾0x5B 就认为通信正确 处理数据
        {
            state = 0;//这里就可以处理数据了、处理完记得清空数组和重置标志位与计数值
            //比如根据数据设置红外旋转偏移状态

            //1.设置快速 慢速右边 左边 数字存储的变量意义：[0]和[1]:帧头，[2]:摄像头左边数第一个感兴趣区域，[3]:左边第二个，[4]:左边第三个，[5]:左边第四个，[6]:左
                if(g_ucaUsart2ReceiveBuffer[0]==0&&g_ucaUsart2ReceiveBuffer[1]==0&&g_ucaUsart2ReceiveBuffer[2]==0)
                {
                  g_cThisState=0;//解透
                  g_iHW_State=22222;//设置这个显示在OLED上方便调试 五个值 以此从左向右表示 从左向右的五个区域
                }
                if(g_ucaUsart2ReceiveBuffer[0]==0&&g_ucaUsart2ReceiveBuffer[1]==0&&g_ucaUsart2ReceiveBuffer[2]==0)
                {
                  g_cThisState=-1;//应该右转
                  g_iHW_State=22212;  //表示右数第二个 识别到线
                }
```

【3】指定区域获取阈值和单独实现

```
# 设定阈值范围变量 后面会更新到这里的
threshold = [0, 0, 0, 0, 0, 0]  # LAB色彩通道的阈值 [Lmin, Lmax, Amin, Amax, Bmin, Bmax]

#****************[0]-获取指定位置阈值-控制阈值计算只执行一次的标志********************
threshold_calculated = False #控制阈值计算只执行一次的标志
threshold_roi = (80, 60, 20, 20)  # 设定ROI，(x, y, w, h)格式# 设定要分析的区域
target_roi = (120, 80, 20, 20)  # 设定目标区域，(x, y, w, h)格式，用于后续判断是否满足阈值
#****************[1]-获取指定位置阈值-阈值获取函数                ********************
# 封装为函数，识别指定区域的阈值
def get_threshold(roi):
    # 循环多次（默认150次）更新阈值
    threshold = [0, 0, 0, 0, 0, 0]  # LAB色彩通道的阈值 [Lmin, Lmax, Amin, Amax, Bmin, Bmax]
    for _ in range(150):
        img = sensor.snapshot()
        # 获取指定区域的颜色直方图
        hist = img.get_histogram(roi=roi)
        img.draw_rectangle(roi, color=(0, 255, 0), thickness=2)  # 使用绿色（0, 255, 0），厚度为2# 在图像上绘制绿色矩形框标识采集区域
        # 在绿色矩形框上方显示“采集计算阈值中...”并加上省略号
        img.draw_string(roi[0], roi[1] - 10, "Collecting Threshold...", color=(0, 255, 0), scale=1)
        # 获取L、A、B三个通道的5%和95%分位值
        lo = hist.get_percentile(0.05)  # 获取5%分位值，表示颜色分布的下边界
        hi = hist.get_percentile(0.95)  # 获取95%分位值，表示颜色分布的上边界
        print("采集计算阈值中...请等待")  # 打印检查结果，1表示满足，0表示不满足
        # 输出lo和hi的值
#        print(f"5% Percentile (lo): L={lo.l_value()} A={lo.a_value()} B={lo.b_value()}")
#        print(f"95% Percentile (hi): L={hi.l_value()} A={hi.a_value()} B={hi.b_value()}")
        # L通道的最小值和最大值平均后作为新的阈值
        threshold[0] = (threshold[0] + lo.l_value()) // 2  # L通道的最小值
        threshold[1] = (threshold[1] + hi.l_value()) // 2  # L通道的最大值
        # A通道的最小值和最大值平均后作为新的阈值
        threshold[2] = (threshold[2] + lo.a_value()) // 2  # A通道的最小值
        threshold[3] = (threshold[3] + hi.a_value()) // 2  # A通道的最大值
        # B通道的最小值和最大值平均后作为新的阈值
        threshold[4] = (threshold[4] + lo.b_value()) // 2  # B通道的最小值
        threshold[5] = (threshold[5] + hi.b_value()) // 2  # B通道的最大值

    print(f"计算阈值的位置区域是 ROI Info: x={roi[0]}, y={roi[1]}, width={roi[2]}, height={roi[3]}")  # 输出roi区域的信息
    # 打印每个通道的阈值信息
    print("计算出的阈值  Threshold: Lmin={0} Lmax={1}, Amin={2} Amax={3}, Bmin={4} Bmax={5}".format(
        threshold[0], threshold[1], threshold[2], threshold[3], threshold[4], threshold[5]
    ))

    # 返回计算得到的阈值列表，包含L、A、B三个通道的最小值和最大值
    return threshold  # 返回最终的阈值数组
```

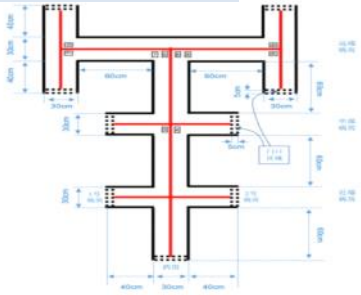
```
while(True):
    # 捕获图像
    img = sensor.snapshot()

#*****************[2]-获取指定位置阈值-进行阈值计算的内容*******************
    if not threshold_calculated:# 仅在阈值未计算时进行计算
        # 调用函数获取指定区域的阈值
        threshold = get_threshold(threshold_roi)

        # 设置阈值计算完成的标志
        threshold_calculated = True

#    img.draw_rectangle(threshold_roi, color=(0, 255, 0), thickness=2)  # 使用绿色
(0, 255, 0)，厚度为2

    # 检查目标区域是否满足阈值条件
    blobs  = img.find_blobs([threshold], roi=target_roi)
    if blobs:#如果roi1区域内找到阈值色块 就会赋值flag[0]为1
        result = 1

    else :
        result = 0


    # 在目标区域上绘制矩形框
    img.draw_rectangle(target_roi, color=(255, 0, 0), thickness=2)  # 使用红色
(255, 0, 0)，厚度为2
    print("Target region check result: ", result)  # 打印检查结果，1表示满足，0表示不
满足
    # 延时，避免输出过于频繁
```

【4】增加串口输出获取指定区域阈值

```
import pyb
from pyb import UART
import ustruct
import sensor

uart = UART(3,115200,bits=8, parity=None, stop=1, timeout_char = 1000)#初始化串口

roi1 = [( 20,  105, 10, 10),
        ( 45,  105, 10, 10),
        ( 75,  105, 10, 10),
        (105,  105, 10, 10),
        (130,  105, 10, 10)]

led = pyb.LED(1)
led.on()

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.set_vflip(True)  #垂直方向翻转
sensor.skip_frames(time=2000)
sensor.set_auto_whitebal(True)#自动白平衡模式
sensor.set_auto_gain(False)#关闭自动增益模式

GROUND_THRESHOLD=(0, 30, -22, 23, -128, 80)
threshold_calculated = False

def get_threshold(roi):
    threshold = [0, 0, 0, 0, 0, 0]
    for _ in range(150):
        img = sensor.snapshot()
        hist = img.get_histogram(roi=roi)
        img.draw_rectangle(roi, color=(0, 255, 0), thickness=2)
        img.draw_string(roi[0], roi[1] - 10, "Collecting Threshold...",\
        color=(0, 255, 0), scale=1)
        lo = hist.get_percentile(0.05)
        hi = hist.get_percentile(0.95)
        print("采集计算阈值中...请等待")
        threshold[0] = (threshold[0] + lo.l_value()) // 2  # L通道的最小值
        threshold[1] = (threshold[1] + hi.l_value()) // 2  # L通道的最大值
        threshold[2] = (threshold[2] + lo.a_value()) // 2  # A通道的最小值
        threshold[3] = (threshold[3] + hi.a_value()) // 2  # A通道的最大值
        threshold[4] = (threshold[4] + lo.b_value()) // 2  # B通道的最小值
        threshold[5] = (threshold[5] + hi.b_value()) // 2  # B通道的最大值
    print(f"计算阈值的位置区域是 ROI Info: x={roi[0]}, y={roi[1]}, width\
    ={roi[2]}, height={roi[3]}")
    print("计算出的阈值  Threshold: Lmin={0} Lmax={1}, Amin={2} Amax={3},\
    Bmin={4} Bmax={5}".format(threshold[0], threshold[1], threshold[2],\
    threshold[3], threshold[4], threshold[5]))
    return threshold


def send_five_uchar(c1,c2,c3,c4,c5):#功能发送五个无符号字符（unsigned char）
    global uart;
    data = ustruct.pack("<BBBBBBBB",
                        0xA5,
```

```python
def send_five_uchar(c1,c2,c3,c4,c5):#功能发送五个无符号字符（unsigned char）
    global uart;
    data = ustruct.pack("<BBBBBBBB",
                        0xA5,
                        0xA6,
                        c1,
                        c2,
                        c3,
                        c4,
                        c5,
                        0x5B
                        )
    uart.write(data);
    print(data)

GROUND_THRESHOLD=(0, 30, -22, 23, -128, 80)

while(True):
    data=0
    blob1=None
    blob2=None
    blob3=None
    blob4=None
    blob5=None
    flag = [0,0,0,0,0]

    if not threshold_calculated:
        GROUND_THRESHOLD = get_threshold(roi1[2])
        threshold_calculated = True

    img = sensor.snapshot().lens_corr(strength = 1.7 , zoom = 1.0)#对获取到的图像执行镜头校正
    blob1 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[0])
    blob2 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[1])
    blob3 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[2])
    blob4 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[3])
    blob5 = img.find_blobs([GROUND_THRESHOLD], roi=roi1[4])

    if blob1:#如果roi1区域内找到阈值色块 就会赋值flag[0]为1
        flag[0] = 1
    if blob2:
        flag[1] = 1
    if blob3:
        flag[2] = 1
    if blob4:
        flag[3] = 1
    if blob5:
        flag[4] = 1
    print(flag[0],flag[1],flag[2],flag[3],flag[4])
    # send_five_uchar(flag[0],flag[1],flag[2],flag[3],flag[4])

    for i, rec in enumerate(roi1):
        img.draw_rectangle(rec, color=(255,0,0))
        result_text = str(flag[i])
        text_y_position = rec[1] - 15
        img.draw_string(rec[0], text_y_position, result_text, color=(255, 255,
        255), scale=2)
```

【5】识别十字路口（单片机判断串口指令）



根据串口输出的感兴趣区域的识别结果，来判断是否是路口

调整适合的感兴趣区域大小，如果有两个及其以上是1 那么就可以判定是路口。

【6】识别形状（追小球小车）

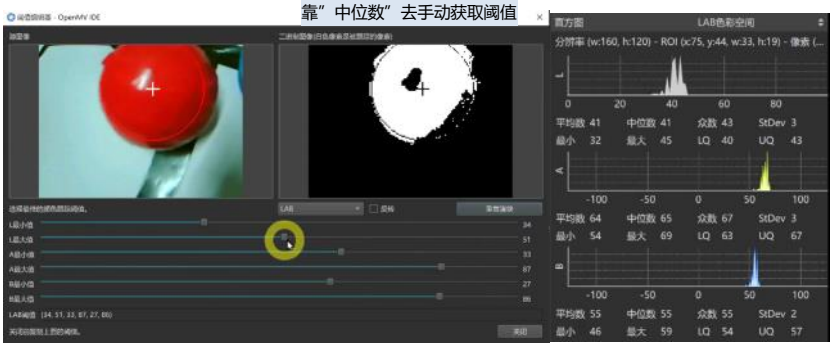
```python
import sensor, image, time
# 初始化传感器
sensor.reset()

# 设置图像格式为RGB565，RGB格式相较于灰度图像速度更快
sensor.set_pixformat(sensor.RGB565)

# 设置图像分辨率为QQVGA（160x120），适用于快速处理
sensor.set_framesize(sensor.QQVGA)

# ************************** 如果不需要镜像就注释掉以下代码
**************************
# 如果摄像头模块的安装方向需要翻转，可以启用以下设置来进行镜像操作：
sensor.set_vflip(True)  # 设置垂直翻转，如果摄像头安装方向需要垂直翻转，启用此选项
sensor.skip_frames(time=2000)

# 创建时钟对象以便计算帧率
clock = time.clock()

while(True):
    # 每次循环都调用 tick() 更新时钟，计算帧率
    clock.tick()

    # 获取当前图像，并进行镜头畸变校正（参数1.8为校正系数，适用于大部分情况）
    img = sensor.snapshot().lens_corr(1.8)

    # 调整阈值和半径范围参数，以提高检测效果和性能
    for c in img.find_circles(
        threshold=3500,  # 设置霍夫变换的阈值。值越大，只有强度更高的圆会被检测到。此值可以
根据实际场景调整。较高的值意味着需要更明显的圆形才能被检测到
        x_margin=10,  # x方向的合并误差范围，增大会使相近的圆合并。
        y_margin=10,  # y方向的合并误差范围，增大会使相近的圆合并。
        r_margin=10,  # 半径方向的合并误差范围，增大会使半径相近的圆合并。
        r_min=2,  # 设置检测的最小圆半径
        r_max=100,  # 设置检测的最大圆半径 单位是像素。100 像素对应多少毫米是一个动态计算的
问题，需要根据具体的摄像头视场角、分辨率和物体距离来调整。 具体可以借助AI计算和了解
        r_step=2  # 设置检测半径时的步长，步长越小，检测的圆的精度越高，但性能消耗较大
    ):
        # 绘制圆形，(255, 0, 0)表示圆的颜色为红色
        img.draw_circle(c.x(), c.y(), c.r(), color=(255, 0, 0))

        # 打印圆的信息，包括圆心坐标和半径
        print("Circle found: x = {}, y = {}, radius = {}, magnitude =
{}".format(c.x(), c.y(), c.r(), c.magnitude()))

    # 输出当前帧率，帮助调试和评估图像处理的性能
    print("FPS: %f" % clock.fps())
```

```python
import sensor, time

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.set_vflip(True)  #垂直方向翻转
sensor.skip_frames(time=2000)
sensor.set_auto_whitebal(True)#自动白平衡模式
sensor.set_auto_gain(False)#关闭自动增益模式
clock = time.clock()

while(True):
# 每次循环都调用 tick() 更新时钟，计算帧率
    clock.tick()
    img = sensor.snapshot().lens_corr(1.8)
    for c in img.find_circles(
        threshold=3500,  #设置霍夫变换的阈值,值越大,强度更高的圆才会被检测到\
        x_margin=10,  # x方向的合并误差范围，增大会使相近的圆合并。\
        y_margin=10,  # y方向的合并误差范围，增大会使相近的圆合并。\
        r_margin=10,  # 半径方向的合并误差范围，增大会使半径相近的圆合并。\
        r_min=2,  # 设置检测的最小圆半径\
        r_max=100,  # 设置检测的最大圆半径 单位是像素。100 像素对应多少毫米\
        r_step=2  # 设置检测半径时的步长，步长越小，检测的圆的精度越高，但性能消耗较大\
    ):
        img.draw_circle(c.x(), c.y(), c.r(), color=(255, 0, 0))

        print("Circle found: x = {}, y = {}, radius = {}, magnitude = \
        {}".format(c.x(), c.y(), c.r(), c.magnitude()))

    # 输出当前帧率，帮助调试和评估图像处理的性能
    print("FPS: %f" % clock.fps())
```

【7】识别颜色（追小球小车）

```python
import sensor  # 导入 OpenMV 的 sensor 库，用于与摄像头交互
import time  # 导入 time 库，用于计时
import math  # 导入 math 库，用于数学计算，特别是角度转换
# 设置红色的颜色阈值范围 (L Min, L Max, A Min, A Max, B Min, B Max)
color_threshold = (30, 100, 15, 127, 15, 127)  # 红色的色彩空间阈值，可以根据需要调整阈
值范围

# 摄像头初始化设置
sensor.reset()  # 重置摄像头模块，确保初始化清晰
sensor.set_pixformat(sensor.RGB565)  # 设置图像像素格式为 RGB565 格式，适合色彩处理
sensor.set_framesize(sensor.QQVGA)  # 设置图像分辨率为 QVGA (320x240)。分辨率越高，图
像质量越好，但处理速度可能会较慢

# ************************** 如果不需要镜像就注释掉以下代码
**************************
# 摄像头的镜像和翻转操作，根据摄像头模块的安装方向决定是否需要
sensor.set_vflip(True)  # 设置垂直翻转。如果摄像头安装方向需要翻转，启用此选项
sensor.skip_frames(time=2000)  # 跳过前几帧，确保摄像头稳定，避免由于启动时摄像头状态不稳
定产生错误
sensor.set_auto_gain(False)  # 关闭自动增益，确保颜色跟踪不受环境光影响，避免图像质量变化
sensor.set_auto_whitebal(False)  # 关闭自动白平衡，确保颜色跟踪不受环境光和白平衡影响

clock = time.clock()  # 创建一个 clock 对象，用于计算帧率 (FPS)，帮助评估性能

while True:  # 循环执行，持续进行颜色跟踪
    clock.tick()  # 计时一帧的处理时间，帮助计算帧率（FPS）
    img = sensor.snapshot()  # 捕获当前帧图像，获取实时的摄像头图像

    # 使用 find_blobs 查找颜色区域，返回符合条件的 blob
    # 这里使用了上面定义的红色阈值，根据阈值对图像进行颜色区域检测
    for blob in img.find_blobs(
        [color_threshold],  # 使用当前定义的颜色阈值进行颜色跟踪，目标为红色区域
        pixels_threshold=200,  # 过滤掉小的 blob，最小像素数为 200。较小的区域可能不值得
跟踪
        area_threshold=200,  # 过滤掉面积过小的区域，较小的区域通常不被认为是有效的目标
        merge=True,  # 合并重叠的 blob（如果多个检测到的区域重叠，它们会被合并为一个）
    ):
        # 对于延伸性较强的区域（非圆形区域），进行边缘和轴线的绘制
        if blob.elongation() > 0.5:  # 如果区域的延伸性大于 0.5（接近矩形），则认为其为
较为规则的区域
            img.draw_edges(blob.min_corners(), color=(255, 0, 0))  # 绘制区域的最小
矩形边缘，使用红色进行标记
            img.draw_line(blob.major_axis_line(), color=(0, 255, 0))  # 绘制主轴
```

靠”中位数”去手动获取阈值




```python
import sensor, time

color_threshold = (30, 100, 15, 127, 15, 127)

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.set_vflip(True)
sensor.skip_frames(time=2000)
sensor.set_auto_gain(False)      #关闭自动增益，让颜色跟踪不受环境光影响，避免图像质量变化
sensor.set_auto_whitebal(False) #关闭自动白平衡，确保颜色跟踪不受环境光和白平衡影响
clock = time.clock()

while True:
    clock.tick()
    img = sensor.snapshot()

    for blob in img.find_blobs(
        [color_threshold],
        pixels_threshold=200,
        area_threshold=200,
        merge=True,
    ):

        if blob.elongation() > 0.5:
            img.draw_edges(blob.min_corners(), color=(255, 0, 0))
            img.draw_line(blob.major_axis_line(), color=(0, 255, 0))
```

```
        if blob.elongation() > 0.5:  # 如果区域的延伸性大于 0.5（接近矩形），则认为其为较为规则的区域
            img.draw_edges(blob.min_corners(), color=(255, 0, 0))  # 绘制区域的最小矩形边缘，使用红色进行标记
            img.draw_line(blob.major_axis_line(), color=(0, 255, 0))  # 绘制主轴线，使用绿色进行标记
            img.draw_line(blob.minor_axis_line(), color=(0, 0, 255))  # 绘制副轴线，使用蓝色进行标记

        else :  # 这是blob.elongation() 小于等于0.5是圆形或者正方形了
            # 绘制圆形
            # 使用宽度和高度中的较小值来作为圆形的半径
            radius = int(min(blob.w(), blob.h()) / 2)  # 计算半径，选择宽度和高度中的较小值作为半径
            img.draw_circle(blob.cx(), blob.cy(), radius, color=(255, 0, 0))  # 绘制红色圆形

            print("Circle found: x = {}, y = {}, radius = {}, ".format(blob.cx(), blob.cy(), radius))

        img.draw_cross(blob.cx(), blob.cy())  # 绘制中心交叉十字，表示颜色区域的中心位置


    print(clock.fps())  # 打印每秒帧数（FPS），用于调试性能，查看处理速度，确保帧率足够稳定
```

```
    ):

        if blob.elongation() > 0.5:
            img.draw_edges(blob.min_corners(), color=(255, 0, 0))
            img.draw_line(blob.major_axis_line(), color=(0, 255, 0))
            img.draw_line(blob.minor_axis_line(), color=(0, 0, 255))

        else :
            radius = int(min(blob.w(), blob.h()) / 2)
            img.draw_circle(blob.cx(), blob.cy(), radius, color=(255, 0, 0))

            print("Circle found: x = {}, y = {}, radius = {}, \
            ".format(blob.cx(), blob.cy(), radius))

        img.draw_cross(blob.cx(), blob.cy())
    print(clock.fps())
```

【8】结合篇@先识别颜色，后识别形状

```
import sensor, image, time
#教程作者:好家伙VCC
#欢迎交流群QQ: 771027961 作者邮箱: 1930299709@qq.com
#更多教程B站主页:[好家伙VCC的个人空间-好家伙VCC个人主页-哔哩哔哩视频](https://space.bilibili.com/434192043)
#淘宝主页链接:[首页-好家伙VCC-淘宝网](https://shop415231378.taobao.com)
#更多嵌入式手把手教程-尽在好家伙VCC
# 定义颜色阈值（L, A, B），用于识别红色
# L通道：亮度值，较小表示较暗的颜色
# A通道：绿色与红色的色差，红色偏大
# B通道：蓝色与黄色的色差，红色偏小
color_threshold = (0, 100, 0, 127, 0, 127)  # (L_min, L_max, A_min, A_max, B_min, B_max)

# 初始化摄像头模块
sensor.reset()  # 重置摄像头，确保设备正常工作
sensor.set_pixformat(sensor.RGB565)  # 设置摄像头的像素格式为RGB565，每个像素16位色深
sensor.set_framesize(sensor.QQVGA)  # 设置摄像头的分辨率为QQVGA（160x120），适合快速处理

# ************************** 如果不需要镜像就注释掉以下代码 **************************
# 摄像头镜像和翻转设置，根据摄像头的安装方向调整
sensor.set_vflip(True)  # 设置垂直翻转，适用于摄像头上下安装的情况
sensor.set_hmirror(True)  # 设置水平翻转，适用于摄像头左右安装的情况
# ************************** 如果不需要镜像就注释掉以上代码 **************************

sensor.skip_frames(time = 2000)  # 跳过前几帧的图像，确保图像稳定后再开始处理
sensor.set_auto_gain(False)  # 必须关闭自动增益，防止影响颜色追踪
sensor.set_auto_whitebal(False)  # 必须关闭自动白平衡，防止影响颜色追踪

# 创建一个时钟对象，用于计算和控制帧率
clock = time.clock()

# 主循环，不断获取摄像头图像并进行处理
while(True):
    clock.tick()  # 计时当前帧的处理时间，计算帧率

    # 获取当前图像并进行镜头畸变校正，纠正因镜头产生的畸变
    img = sensor.snapshot().lens_corr(1.8)  # 1.8是畸变系数，适当调整可以改善图像质量

    # 使用霍夫变换查找圆形，并返回找到的圆的信息
    for c in img.find_circles(
        threshold = 2500,  # 设置圆形检测的阈值。较高的值意味着需要更明显的圆形才能被检测到
        x_margin = 10,     # 圆心的X坐标允许的误差范围
        y_margin = 10,     # 圆心的Y坐标允许的误差范围
        r_margin = 10,     # 圆半径的允许误差范围
        r_min = 2,         # 圆的最小半径 单位为像素。
        r_max = 100,       # 圆的最大半径 单位为像素。100 像素对应多少毫米是一个动态计算的问题，需要根据具体的摄像头视场角、分辨率和物体距离来调整。
        r_step = 2         # 圆半径变化的步长 单位为像素。
    ):
        # 计算圆形的外接矩形区域，这样可以方便获取圆的统计信息
        area = (c.x() - c.r(), c.y() - c.r(), 2 * c.r(), 2 * c.r())  # (x, y, width, height)

        # 获取该区域内的像素颜色统计信息
        statistics = img.get_statistics(roi=area)  # 获取外接矩形区域的像素统计信息（颜色分布）

        # 打印该区域的颜色统计数据，用于调试
        #print(statistics)

        # 判断该区域是否为红色圆
        # 使用L、A、B通道的众数来判断颜色是否符合红色范围
        if (
            color_threshold[0] < statistics.l_mode() < color_threshold[1] and  # L通道的众数应小于100，表示较暗的颜色
            color_threshold[2] < statistics.a_mode() < color_threshold[3] and  #
```

```
import sensor, time

color_threshold = (30, 100, 15, 127, 15, 127)

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.set_vflip(True)
sensor.skip_frames(time=2000)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)
clock = time.clock()

while True:
    clock.tick()
    img = sensor.snapshot().lens_corr(1.8)

    for c in img.find_circles(
        threshold = 2500,
        x_margin = 10,
        y_margin = 10,
        r_margin = 10,
        r_min = 2,
        r_max = 100,
        r_step = 2
    ):
        area = (c.x() - c.r(), c.y() - c.r(), 2 * c.r(), 2 * c.r())
        statistics = img.get_statistics(roi=area)

        if (
            color_threshold[0] < statistics.l_mode() < color_threshold[1] and
            color_threshold[2] < statistics.a_mode() < color_threshold[3] and
            color_threshold[4] < statistics.b_mode() < color_threshold[5]
        ):
            img.draw_circle(c.x(), c.y(), c.r(), color=(192, 255, 0))
            print("Circle found: x = {}, y = {}, radius = {}".format(c.x(), c.y(), c.r()))
        else:
            img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))
    print("FPS %f" % clock.fps())
```

```
        # 使用L、A、B通道的众数来判断颜色是否符合红色范围
        if (
            color_threshold[0] < statistics.l_mode() < color_threshold[1] and  #
L通道的众数应小于100，表示较暗的颜色
            color_threshold[2] < statistics.a_mode() < color_threshold[3] and  #
A通道的众数应小于127，表示偏红色
            color_threshold[4] < statistics.b_mode() < color_threshold[5]      #
B通道的众数应小于127，表示偏蓝色
        ):
            # 如果该区域是红色的圆形，用银光绿色的圆框标记该圆形
            img.draw_circle(c.x(), c.y(), c.r(), color=(192, 255, 0))  # 银光绿色
圆框
            print("Circle found: x = {}, y = {}, radius = {}".format(c.x(),
c.y(), c.r()))
        else:
            # 如果不是红色圆形，用白色矩形框标记该区域
            img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))  # 白色

    # 打印当前帧率（每秒帧数），便于调试性能
    print("FPS %f" % clock.fps())  # 输出当前的帧率
```

【9】结合篇@先识别颜色，再识别形状

```
import sensor, image, time
red_threshold = [(35, 69, 41, 127, -14, 57)]  # (L_min, L_max, A_min, A_max,
B_min, B_max)

# 初始化摄像头模块
sensor.reset()  # 重置摄像头，准备工作
sensor.set_pixformat(sensor.RGB565)  # 设置摄像头的像素格式为RGB565（16位色深）
sensor.set_framesize(sensor.QQVGA)  # 设置摄像头的分辨率为QQVGA（160x120）

# ************************** 如果不需要镜像就注释掉以下代码
**************************
# 摄像头的镜像和翻转操作，根据摄像头模块的安装方向决定是否需要
sensor.set_vflip(True)  # 设置垂直翻转。如果摄像头安装方向需要翻转，启用此选项
sensor.set_hmirror(True)  # 设置水平翻转。如果摄像头安装方向需要翻转，启用此选项
# ************************** 如果不需要镜像就注释掉以上代码
**************************

sensor.skip_frames(time=2000)  # 跳过前几帧的图像，确保图像稳定，防止刚开机时图像不清晰
sensor.set_auto_gain(False)  # 关闭自动增益，用于颜色追踪，避免自动增益影响颜色识别
sensor.set_auto_whitebal(False)  # 关闭自动白平衡，用于颜色追踪，避免自动白平衡影响颜色识
别

# 创建一个时钟对象，用于计算和控制帧率
clock = time.clock()  # 用于控制循环的时间和帧率

while(True):
    clock.tick()  # 计时当前帧的处理时间，更新帧率
    # 获取当前图像，并进行镜头畸变校正
    img = sensor.snapshot().lens_corr(1.8)  # 纠正镜头畸变，1.8是畸变系数，用于保证图像
无畸变

    # 颜色过滤：提取红色区域
    # 使用颜色阈值过滤红色区域，返回的是一组与红色相匹配的区域（blobs）
    # 这里使用了red_threshold来限制颜色范围，只识别红色区域，像素数阈值为100，区域最小面积为
100
    blobs = img.find_blobs(red_threshold, pixels_threshold=100,
area_threshold=100, merge=True)

    # 对每个找到的红色区域进行处理
    for blob in blobs:
        # blob.rect 是该颜色区域的边界矩形，格式为 (x, y, w, h)
        # x, y是矩形的左上角坐标，w是宽度，h是高度
        # 在该颜色区域内进一步进行形状检测（如圆形）
        area = (blob.x(), blob.y(), blob.w(), blob.h())  # 获取区域的矩形边界框

        # 获取该区域内的颜色统计信息（统计颜色分布），帮助分析区域内的颜色均衡
        # stats 变量包含了图像区域内的平均色调等信息
        statistics = img.get_statistics(roi=area)  # 获取区域内颜色的统计信息
        #print(statistics)  # 打印该区域的颜色统计信息，便于调试和查看

        circles = img.find_circles(
            threshold=2500,  # 设置圆形检测的阈值，较高的值意味着更明显的圆形才能被检测到
            x_margin=10,     # 圆心的X坐标允许的误差范围
            y_margin=10,     # 圆心的Y坐标允许的误差范围
            r_margin=10,     # 圆半径的允许误差范围
            r_min=2,         # 圆的最小半径（像素），根据需求可以调整
            r_max=100,       # 圆的最大半径（像素），根据需求可以调整
            r_step=2         # 圆半径变化的步长（像素）
        )

        # 检查霍夫变换返回的圆形列表
        # circles是一个包含多个圆的列表，每个圆有圆心坐标(x, y)和半径r
        for c in circles:
            # 检查圆形的检测结果是否确实位于红色区域内，避免误判
            # 判断该圆的中心是否位于红色区域的矩形内
            if blob.x() <= c.x() <= blob.x() + blob.w() and blob.y() <= c.y() <=
blob.y() + blob.h():
                # 如果检测到圆形且圆形位于红色区域内，绘制圆框
                img.draw_circle(c.x(), c.y(), c.r(), color=(192, 255, 0))  # 用绿
色绘制圆形
                print("Circle found: x = {}, y = {}, radius = {}".format(c.x(),
c.y(), c.r()))
            else:
                # 如果圆形不在红色区域内，绘制白色圆框
                img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))  # 白
色
    # 打印当前帧率（每秒帧数），便于调试性能
```

```
import sensor, time

red_threshold = [(30, 100, 15, 127, 15, 127)]

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.set_vflip(True)
sensor.skip_frames(time=2000)
sensor.set_auto_gain(False)      #关闭自动增益，让颜色跟踪不受环境光影响，避免图像质量变化
sensor.set_auto_whitebal(False) #关闭自动白平衡，确保颜色跟踪不受环境光和白平衡影响
clock = time.clock()

while True:
    clock.tick()
    img = sensor.snapshot().lens_corr(1.8)

    blobs = img.find_blobs(red_threshold, pixels_threshold=100,\
    area_threshold=100, merge=True)

    for blob in blobs:
        area = (blob.x(), blob.y(), blob.w(), blob.h())
        statistics = img.get_statistics(roi=area)
        circles = img.find_circles(
            threshold = 2500,
            x_margin = 10,
            y_margin = 10,
            r_margin = 10,
            r_min = 2,
            r_max = 100,
            r_step = 2)
        for c in circles:
            if blob.x() <= c.x() <= blob.x() + blob.w() and\
            blob.y() <= c.y() <= blob.y() + blob.h():
                img.draw_circle(c.x(), c.y(), c.r(), color=(192, 255, 0))
                print("Circle found: x = {}, y = {}, radius = {}".format(c.x(),\
                c.y(), c.r()))
            else:
                img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))


    print("FPS %f" % clock.fps())

```

```
            else:
                # 如果圆形不在红色区域内，绘制白色圆框
                img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))  # 白
色
    # 打印当前帧率（每秒帧数），便于调试性能
    # FPS表示每秒处理的帧数，是衡量图像处理效率的标准
    print("FPS %f" % clock.fps())  # 输出当前的帧率（每秒帧数），以便调试图像处理速度
```

【10】结合篇@分别识别形状，颜色（增加最小变化阈值）

```
# 位置和半径变化阈值，只有变化大于该阈值时才更新
position_threshold = 4  # 位置变化的最小阈值 两个值越小小球识别更新的越频繁，值越大小球的细
微运动越不会更新识别
radius_threshold = 4    # 半径变化的最小阈值 可以根据自己的需求测试调节这个两个阈值

# 上一帧的圆心坐标和半径
prev_x, prev_y, prev_r = None, None, None
# ************************** 最小变化阈值滤波 *************************
            # 如果是第一次检测，更新值
            if prev_x is None or prev_y is None or prev_r is None:
                prev_x, prev_y, prev_r = x, y, r  #更新上次的位置值
                # 绘制圆形
                img.draw_circle(x, y, r, color=(192, 255, 0)) #绘制圆为绿色

            else:
                # 判断位置和半径变化是否大于阈值
                x_change = abs(x - prev_x)#计算这次值和上次值 的绝对值
                y_change = abs(y - prev_y)
                r_change = abs(r - prev_r)

                if x_change > position_threshold or y_change >
position_threshold or r_change > radius_threshold:
                    # 变化大于阈值，更新值
                    prev_x, prev_y, prev_r = x, y, r
                    # 绘制更新后的圆形
                    img.draw_circle(x, y, r, color=(192, 255, 0))
                    print("Circle found: x = {}, y = {}, radius = {}".format(x,
y, r))
                else :
                    # 变化小于阈值，绘制上次的坐标和半径
                    img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                    print("Circle found: x = {}, y = {}, radius =
{}".format(prev_x, prev_y, prev_r))

        else:
            # 如果不是红色圆形，用白色矩形框标记该区域
            img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))  # 白色
```

```
import sensor, time

color_threshold = (30, 100, 15, 127, 15, 127)

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.set_vflip(True)
sensor.skip_frames(time=2000)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)
clock = time.clock()

position_threshold = 4
radius_threshold = 4
prev_x, prev_y, prev_r = None, None, None


while True:
    clock.tick()
    img = sensor.snapshot().lens_corr(1.8)

    for c in img.find_circles(
        threshold = 2500,
        x_margin = 10,
        y_margin = 10,
        r_margin = 10,
        r_min = 2,
        r_max = 100,
        r_step = 2
    ):
        area = (c.x() - c.r(), c.y() - c.r(), 2 * c.r(), 2 * c.r())
        statistics = img.get_statistics(roi=area)

        if (
           color_threshold[0] < statistics.l_mode() < color_threshold[1] and
           color_threshold[2] < statistics.a_mode() < color_threshold[3] and
           color_threshold[4] < statistics.b_mode() < color_threshold[5]
        ):
            x, y, r = c.x(), c.y(), c.r()
            if prev_x is None or prev_y is None or prev_r is None:
                prev_x, prev_y, prev_r = x, y, r  #更新上次的位置值
                img.draw_circle(x, y, r, color=(192, 255, 0)) #绘制圆为绿色
            else:
                x_change = abs(x - prev_x)#计算这次值和上次值 的绝对值
                y_change = abs(y - prev_y)
                r_change = abs(r - prev_r)

                if x_change > position_threshold or y_change > \
                position_threshold or r_change > radius_threshold:
                    prev_x, prev_y, prev_r = x, y, r
                    img.draw_circle(x, y, r, color=(192, 255, 0))
                    print("Circle found: x = {}, y = {}, radius = {}".\
                    format(x, y, r))
                else :
                    img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                    print("Circle found: x = {}, y = {}, radius = {}".\
                    format(prev_x, prev_y, prev_r))


        else:
            img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))
    print("FPS %f" % clock.fps())

```

【11】结合篇@分别识别形状，颜色（增加最小变化阈值+最大变化阈值）

```
import sensor, image, time
#教程作者:好家伙VCC
#欢迎交流群QQ: 771027961 作者邮箱: 1930299709@qq.com
#更多教程B站主页:[好家伙VCC的个人空间-好家伙VCC个人主页-哔哩哔哩视频]
(https://space.bilibili.com/434192043)
#淘宝主页链接:[首页-好家伙VCC-淘宝网](https://shop415231378.taobao.com)
#更多嵌入式手把手教程-尽在好家伙VCC



# 定义颜色阈值（L, A, B），用于识别红色
color_threshold = (0, 100, 0, 127, 0, 127)

# 初始化摄像头模块
sensor.reset()  # 重置摄像头，确保设备正常工作
sensor.set_pixformat(sensor.RGB565)  # 设置摄像头的像素格式为RGB565
sensor.set_framesize(sensor.QQVGA)  # 设置分辨率为QQVGA（160x120）

# ************************** 如果不需要镜像就注释掉以下代码
```

```
while True:
    clock.tick()
    img = sensor.snapshot().lens_corr(1.8)

    for c in img.find_circles(
        threshold = 2500,
        x_margin = 10,
        y_margin = 10,
        r_margin = 10,
        r_min = 2,
        r_max = 100,
        r_step = 2
    ):
        area = (c.x() - c.r(), c.y() - c.r(), 2 * c.r(), 2 * c.r())
        statistics = img.get_statistics(roi=area)

        if (
           color_threshold[0] < statistics.l_mode() < color_threshold[1] and
           color_threshold[2] < statistics.a_mode() < color_threshold[3] and
           color_threshold[4] < statistics.b_mode() < color_threshold[5]
        ):
            x, y, r = c.x(), c.y(), c.r()
            if prev_x is None or prev_y is None or prev_r is None:
```

```
sensor.set_pixformat(sensor.RGB565)  # 设置摄像头的像素格式为RGB565
sensor.set_framesize(sensor.QQVGA)  # 设置分辨率为QQVGA（160x120）

# ************************** 如果不需要镜像就注释掉以下代码 **************************
sensor.set_vflip(True)  # 垂直翻转
sensor.set_hmirror(True)  # 水平翻转
# ************************** 如果不需要镜像就注释掉以上代码 **************************

sensor.skip_frames(time=2000)  # 跳过前几帧，确保图像稳定
sensor.set_auto_gain(False)  # 关闭自动增益
sensor.set_auto_whitebal(False)  # 关闭自动白平衡

# 创建时钟对象
clock = time.clock()

# ************************** 最小变化阈值滤波 **************************
# 位置和半径变化阈值
position_threshold = 4  # 位置变化的最小阈值
radius_threshold = 4    # 半径变化的最小阈值
MAX_CHANGE_THRESHOLD = 80  # 最大变化阈值（例如位置或半径变化超过此值时不更新）

# 上一帧的圆心坐标和半径
prev_x, prev_y, prev_r = None, None, None
# ************************** 最小变化阈值滤波 **************************

# 主循环，不断获取摄像头图像并进行处理
while True:
    clock.tick()  # 计时当前帧的处理时间

    # 获取当前图像并进行镜头畸变校正
    img = sensor.snapshot().lens_corr(1.8)

    # 使用霍夫变换查找圆形
    for c in img.find_circles(
        threshold=2500,  # 圆形检测阈值
        x_margin=10,     # 圆心X坐标误差范围
        y_margin=10,     # 圆心Y坐标误差范围
        r_margin=10,     # 圆半径误差范围
        r_min=2,         # 圆的最小半径
        r_max=100,       # 圆的最大半径
        r_step=2         # 圆半径变化步长
    ):
        # 计算圆形的外接矩形区域
        area = (c.x() - c.r(), c.y() - c.r(), 2 * c.r(), 2 * c.r())  # (x, y,
width, height)
        statistics = img.get_statistics(roi=area)

        # 判断该区域是否为红色圆
        if (
            color_threshold[0] < statistics.l_mode() < color_threshold[1] and
            color_threshold[2] < statistics.a_mode() < color_threshold[3] and
            color_threshold[4] < statistics.b_mode() < color_threshold[5]
        ):
            # 获取当前圆心和半径
            x, y, r = c.x(), c.y(), c.r()

            # 如果是第一次检测，更新值
            if prev_x is None or prev_y is None or prev_r is None:
                prev_x, prev_y, prev_r = x, y, r
                img.draw_circle(x, y, r, color=(192, 255, 0))  # 绘制圆形

            else:
                # 计算变化量
                x_change = abs(x - prev_x)
                y_change = abs(y - prev_y)
                r_change = abs(r - prev_r)

                # 判断变化是否大于最小阈值，并且变化小于最大阈值
                if (
                    (x_change > position_threshold or y_change >
position_threshold or r_change > radius_threshold) and
                    (x_change <= MAX_CHANGE_THRESHOLD and y_change <=
MAX_CHANGE_THRESHOLD and r_change <= MAX_CHANGE_THRESHOLD)
                ):
                    # 变化大于最小阈值且不超过最大阈值，更新值
                    prev_x, prev_y, prev_r = x, y, r
                    img.draw_circle(x, y, r, color=(192, 255, 0))
                    print("Circle found: x = {}, y = {}, radius = {}".format(x,
y, r))
                else:
                    # 变化小于阈值，绘制上次的坐标和半径
                    img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                    print("Circle found: x = {}, y = {}, radius =
{}".format(prev_x, prev_y, prev_r))

        else:
            # 如果不是红色圆形，用白色矩形框标记该区域
            img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))  # 白色
```

```
        color_threshold[2] < statistics.a_mode() < color_threshold[3] and
        color_threshold[4] < statistics.b_mode() < color_threshold[5]
    ):
        x, y, r = c.x(), c.y(), c.r()
        if prev_x is None or prev_y is None or prev_r is None:
            prev_x, prev_y, prev_r = x, y, r  #更新上次的位置值
            img.draw_circle(x, y, r, color=(192, 255, 0)) #绘制圆为绿色
        else:
            x_change = abs(x - prev_x)#计算这次值和上次值 的绝对值
            y_change = abs(y - prev_y)
            r_change = abs(r - prev_r)

            if ((x_change > position_threshold or
            y_change > position_threshold or
            r_change > radius_threshold) and
            (x_change <= MAX_CHANGE_THRESHOLD and
            y_change <= MAX_CHANGE_THRESHOLD and
            r_change <= MAX_CHANGE_THRESHOLD)):
                prev_x, prev_y, prev_r = x, y, r
                img.draw_circle(x, y, r, color=(192, 255, 0))
                print("Circle found: x = {}, y = {}, radius = {}".\
                format(x, y, r))
            else :
                img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                print("Circle found: x = {}, y = {}, radius = {}".\
                format(prev_x, prev_y, prev_r))
```

```
# 打印当前帧率
print("FPS %f" % clock.fps())
```

【12】结合篇@引入Kalman滤波器

```
import sensor, image, time
#**********************增加卡尔曼滤波
*******************************************************************************
import sensor, image, time, math
from ulab import numpy as np
color_threshold = (0, 100, 0, 127, 0, 127)  # (L_min, L_max, A_min, A_max,
B_min, B_max)

# 初始化摄像头模块
sensor.reset()  # 重置摄像头，确保设备正常工作
sensor.set_pixformat(sensor.RGB565)  # 设置摄像头的像素格式为RGB565，每个像素16位色深
sensor.set_framesize(sensor.QQVGA)  # 设置摄像头的分辨率为QQVGA（160x120），适合快速处
理

# ***************************** 如果不需要镜像就注释掉以下代码
***************************
# 摄像头镜像和翻转设置，根据摄像头的安装方向调整
sensor.set_vflip(True)  # 设置垂直翻转，适用于摄像头上下安装的情况
sensor.skip_frames(time = 2000)  # 跳过前几帧的图像，确保图像稳定后再开始处理
sensor.set_auto_gain(False)  # 必须关闭自动增益，防止影响颜色追踪
sensor.set_auto_whitebal(False)  # 必须关闭自动白平衡，防止影响颜色追踪

# 创建一个时钟对象，用于计算和控制帧率
clock = time.clock()
position_threshold = 4  # 位置变化的最小阈值 两个值越小小球识别更新的越频繁，值越大小球的细
微运动越不会更新识别
radius_threshold = 4    # 半径变化的最小阈值 可以根据自己的需求测试调节这个两个阈值
MAX_CHANGE_THRESHOLD = 80  # 最大变化阈值（例如位置或半径变化超过此值时不更新）
# 上一帧的圆心坐标和半径
prev_x, prev_y, prev_r = None, None, None
#**********************增加卡尔曼滤波**************************8
Ts = 1/22 #Ts = 1  帧率的倒数
# 状态空间矩阵定义
 # 状态转移矩阵 A，描述系统的状态变化
#A = np.array([[1,0,0,0,Ts,0],[0,1,0,0,0,Ts],[0,0,1,0,0,0],[0,0,0,1,0,0],
[0,0,0,0,1,0],[0,0,0,0,0,1]])
# 改进状态转移矩阵A，将速度部分更明确地包含进来A 是状态转移矩阵，描述了系统状态（位置、速度等）
的变化，该矩阵会把当前的状态转换到下一时刻的状态。
A = np.array([[1, 0, 0, 0, Ts, 0],
              [0, 1, 0, 0, 0, Ts],
              [0, 0, 1, 0, 0, 0],
              [0, 0, 0, 1, 0, 0],
              [0, 0, 0, 0, 1, 0],
              [0, 0, 0, 0, 0, 1]])


 # 观测矩阵 C，描述从状态到观测值的映射关系  C 是观测矩阵，它将状态向量（位置、速度）与观测量
（图像中的矩形框信息）联系起来。这里假设观测量是位置和速度。
C = np.array([[1,0,0,0,0,0],[0,1,0,0,0,0],[0,0,1,0,0,0],[0,0,0,1,0,0],
[0,0,0,0,1,0],[0,0,0,0,0,1]])
# 过程噪声协方差矩阵
# 过程噪声协方差矩阵 Q，用于描述过程的随机噪声
#Q_value = [1e-8 for _ in range(6)]
#Q = np.diag(Q_value) #创建对角矩阵
# 增大过程噪声矩阵Q_value，使得卡尔曼滤波更灵活地应对物体快速运动
#Q 是过程噪声协方差矩阵，用于描述系统过程中的不确定性。矩阵的对角元素表示每个状态变量的噪声水
平。
Q_value = [1e-6 for _ in range(6)]  # 调整过程噪声值Q_value = [1e-6 for _ in
range(6)]
Q = np.diag(Q_value)  # 更新过程噪声协方差矩阵

# 观测噪声协方差矩阵  R 是观测噪声协方差矩阵，表示观测过程中测量误差的大小。
R_value = [1e-6 for _ in range(6)]
R = np.diag(R_value)
# 定义观测量Z
x = 0 #左顶点x坐标
y = 0 #左顶点y坐标
last_frame_x = x #上一帧左顶点x坐标
last_frame_y = y #上一帧左顶点y坐标
w = 0 #矩形框宽度w
h = 0 #矩形框高度h
dx = 0 #左顶点x坐标移动速度
dy = 0 #左顶点y坐标移动速度
Z = np.array([x,y,w,h,dx,dy])
# 定义卡尔曼滤波函数变量
#x_hat = np.array([80,60,30,30,2,2]) # 初始估计的状态值（位置、速度等）
# 初始状态估计：根据实际应用情况调整位置和速度的初值
x_hat = np.array([80, 60, 30, 30, 2, 2])  # 根据你的应用需要调整这些值

x_hat_minus = np.array([0,0,0,0,0,0]) # 初始预测的状态值
p_value = [10 for _ in range(6)] # 状态误差的初始值 p 是状态误差的初始协方差矩阵。
p = np.diag(p_value)# 创建误差协方差矩阵 p
def Kalman_Filter(Z):
    global A,C,Q,R,x_hat,x_hat_minus,p
    # 预测部分
    x_hat_minus = np.dot(A,x_hat)
    p_minus = np.dot(A, np.dot(p, A.T)) + Q
    # 校正部分
    S = np.dot(np.dot(C, p_minus), C.T) + R
    # 选择一个小的正则化项
    regularization_term = 1e-4
```

```
def Kalman_Filter(Z):
    global A,C,Q,R,x_hat,x_hat_minus,p
    # 预测部分
    x_hat_minus = np.dot(A,x_hat)
    p_minus = np.dot(A, np.dot(p, A.T)) + Q
    # 校正部分
    S = np.dot(np.dot(C, p_minus), C.T) + R
    # 选择一个小的正则化项
    regularization_term = 1e-4
    # 正则化 S 矩阵
    S_regularized = S + regularization_term * np.eye(S.shape[0])
    # 计算正则化后的 S 矩阵的逆
    S_inv = np.linalg.inv(S_regularized)
    # 计算卡尔曼增益 K
    K = np.dot(np.dot(p_minus, C.T), S_inv)
    x_hat = x_hat_minus + np.dot(K,(Z - np.dot(C,x_hat_minus)))
    p = np.dot((np.eye(6) - np.dot(K,C)),p_minus)
    return x_hat
last_frame_location = [0 for _ in range(4)] #用于存储上一帧的目标位置，这通常用于目标跟踪和计算目标移动等任务。一个长度为4的列表 last_frame_location，其中每个元素的初始值为 0
last_frame_rect = [0 for _ in range(4)]  #存储上一帧检测到的矩形框坐标 成了一个长度为4的列表 last_frame_rect，并且每个元素的初始值为 0。
box = [0 for _ in range(4)] #生成一个包含四个子列表的列表,存储x1,y1,x2,y2


while(True):
    clock.tick()  # 计时当前帧的处理时间，计算帧率

    # 获取当前图像并进行镜头畸变校正，纠正因镜头产生的畸变
    img = sensor.snapshot().lens_corr(1.8)  # 1.8是畸变系数，适当调整可以改善图像质量

    # 使用霍夫变换查找圆形，并返回找到的圆的信息
    for c in img.find_circles(
        threshold = 2500,  # 设置圆形检测的阈值。较高的值意味着需要更明显的圆形才能被检测到
        x_margin = 10,     # 圆心的X坐标允许的误差范围
        y_margin = 10,     # 圆心的Y坐标允许的误差范围
        r_margin = 10,     # 圆半径的允许误差范围
        r_min = 2,         # 圆的最小半径 单位为像素。
        r_max = 100,       # 圆的最大半径 单位为像素。100 像素对应多少毫米是一个动态计算的问题，需要根据具体的摄像头视场角、分辨率和物体距离来调整。
        r_step = 2         # 圆半径变化的步长 单位为像素。
    ):
        # 计算圆形的外接矩形区域，这样可以方便获取圆的统计信息
        area = (c.x() - c.r(), c.y() - c.r(), 2 * c.r(), 2 * c.r())  # (x, y, width, height)
        # 获取该区域内的像素颜色统计信息
        statistics = img.get_statistics(roi=area)  # 获取外接矩形区域的像素统计信息（颜色分布）

        # 打印该区域的颜色统计数据，用于调试
        #print(statistics)

        # 判断该区域是否为红色圆
        # 使用L、A、B通道的众数来判断颜色是否符合红色范围
        if (
            color_threshold[0] < statistics.l_mode() < color_threshold[1] and  # L通道的众数应小于100，表示较暗的颜色
            color_threshold[2] < statistics.a_mode() < color_threshold[3] and  # A通道的众数应小于127，表示偏红色
            color_threshold[4] < statistics.b_mode() < color_threshold[5]      # B通道的众数应小于127，表示偏蓝色
        ):
            # 获取当前圆心和半径
            x, y, r = c.x(), c.y(), c.r()


            # 如果是第一次检测，更新值
            if prev_x is None or prev_y is None or prev_r is None:
                prev_x, prev_y, prev_r = x, y, r  #更新上次的位置值
                # 绘制圆形
                img.draw_circle(x, y, r, color=(192, 255, 0)) #绘制圆为绿色
                # 输出圆心坐标和半径
                print("第一次检测       First detection: center_x = {}, center_y = {}, radius = {}".format(x, y, r))
                circle_center_x = x
                circle_center_y = y
                circle_radius = r

            else:
                # 判断位置和半径变化是否大于阈值
                x_change = abs(x - prev_x)#计算这次值和上次值 的绝对值
                y_change = abs(y - prev_y)
                r_change = abs(r - prev_r)

                # 判断变化是否大于最小阈值，并且变化小于最大阈值
                if (
                    (x_change > position_threshold or y_change > position_threshold or r_change > radius_threshold) and
                    (x_change <= MAX_CHANGE_THRESHOLD and y_change <= MAX_CHANGE_THRESHOLD and r_change <= MAX_CHANGE_THRESHOLD)
                ):
                    # 变化大于阈值，更新值
                    prev_x, prev_y, prev_r = x, y, r
                    # 绘制更新后的圆形
                    img.draw_circle(x, y, r, color=(192, 255, 0))
                    # 输出圆心坐标和半径
                    print("更新检测       Updated detection: center_x = {}, center_y = {}, radius = {}".format(x, y, r))
                    circle_center_x = x
                    circle_center_y = y
                    circle_radius = r

                else :
                    # 变化小于阈值，绘制上次的坐标和半径
                    img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                    # 输出圆心坐标和半径（使用上次的坐标和半径）
```

```
                # 判断变化是否大于最小阈值，并且变化小于最大阈值
                if (
                    (x_change > position_threshold or y_change >
position_threshold or r_change > radius_threshold) and
                    (x_change <= MAX_CHANGE_THRESHOLD and y_change <=
MAX_CHANGE_THRESHOLD and r_change <= MAX_CHANGE_THRESHOLD)
                ):
                    # 变化大于阈值，更新值
                    prev_x, prev_y, prev_r = x, y, r
                    # 绘制更新后的圆形
                    img.draw_circle(x, y, r, color=(192, 255, 0))
                    # 输出圆心坐标和半径
                    print("更新检测       Updated detection: center_x = {},
center_y = {}, radius = {}".format(x, y, r))
                    circle_center_x = x
                    circle_center_y = y
                    circle_radius = r

                else :
                    # 变化小于阈值，绘制上次的坐标和半径
                    img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                    # 输出圆心坐标和半径（使用上次的坐标和半径）
                    print("没有显著变化 No significant change: center_x = {},
center_y = {}, radius = {}".format(prev_x, prev_y, prev_r))
                    circle_center_x = prev_x
                    circle_center_y = prev_y
                    circle_radius = prev_r


            # 假设圆的中心 (c.x(), c.y()) 和半径 c.r()


            # 计算矩形框的左上角坐标和宽高
            rect_x = int(circle_center_x - circle_radius)  # 矩形框左上角的 x 坐标
            rect_y = int(circle_center_y - circle_radius)  # 矩形框左上角的 y 坐标
            rect_w = int(2 * circle_radius)  # 矩形框的宽度，等于圆的直径
            rect_h = int(2 * circle_radius)  # 矩形框的高度，等于圆的直径

            # 将矩形框的四个参数存储到 rect 数组中
            rect = [rect_x, rect_y, rect_w, rect_h]


            box = [rect[0], rect[1], rect[0] + rect[2], rect[1] + rect[3]] # 计算
标记框的四个角坐标（左上角和右下角）
            x, y, w, h = rect[0], rect[1], rect[2], rect[3]# 获取新的矩形框的坐标 识
别成功的时候这个值是识别值的数据赋值
            dx = (x - last_frame_x) / Ts# 计算x方向的速度，假设 Ts 为时间步长
            dy = (y - last_frame_y) / Ts# 计算y方向的速度
            Z = np.array([x, y, w, h, dx, dy]) # 构造测量向量 Z，包括位置和速度
            x_hat = Kalman_Filter(Z)# 使用卡尔曼滤波器进行状态估计
            last_frame_x, last_frame_y = x, y # 更新上一帧的x，y坐标
            #img.draw_rectangle(last_frame_rect, color = (0, 0, 255))
            #img.draw_rectangle(rect, color = (255, 0, 0)) # 用红色矩形框绘制目标位置
            last_frame_rect = rect# 更新上一帧的矩形框 识别成功的时候
            last_frame_location = box# 更新上一帧的位置 识别成功的时候


        else:
            # 如果不是红色圆形，用白色矩形框标记该区域
            #img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))  # 白色



            # 如果不是红色圆形，用白色矩形框标记该区域
            #img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))  # 白色
            #解释：根据卡尔曼滤波的预测值和目标的速度来更新目标的位置：
            #x_hat[0] 和 x_hat[1] 是预测的位置，x_hat[4] 和 x_hat[5] 是目标的速度（速度
分别对应 dx 和 dy）。
            #Ts 是时间步长。更新后的 x, y 位置是通过加入速度（dx 和 dy）的影响来计算的。
            #w 和 h 是目标的宽度和高度，这些信息没有改变，仍然使用 x_hat[2] 和 x_hat[3]。
            x,y,w,h = (x_hat[0] + (x_hat[4] * Ts)),(x_hat[1] + (x_hat[5] *
Ts)),x_hat[2],x_hat[3]
            #解释：计算目标在 X 方向的速度（dx），通过当前帧的 x 位置与上一帧的
last_frame_x 位置之差来计算，除以时间步长 Ts
            dx = (x - last_frame_x) / Ts
            #解释：计算目标在 Y 方向的速度（dy），通过当前帧的 y 位置与上一帧的
last_frame_y 位置之差来计算，除以时间步长 Ts
            dy = (y - last_frame_y) / Ts
            #解释：将目标的位置、宽高、速度（dx 和 dy）组合成一个状态向量 Z，这个向量将作为卡
尔曼滤波器的输入
            Z = np.array([x, y, w, h, dx, dy])
            #解释：将状态向量 Z 传递给卡尔曼滤波器 Kalman_Filter，以获得更新后的预测状态
x_hat
            x_hat = Kalman_Filter(Z)
            #更新上一帧的目标位置 last_frame_x 和 last_frame_y 为当前帧的目标位置 x 和
y，以便在下一帧计算速度时使用
            last_frame_x, last_frame_y = x, y
            #更新上一帧的目标矩形框 last_frame_rect 为当前帧的矩形框 x, y, w, h。
            last_frame_rect = [x,y,w,h]
            #更新上一帧的目标位置 last_frame_location 为当前帧的目标位置。这是一个矩形框的
坐标，包括左上角 (x, y) 和右下角 (x + w, y + h)
            last_frame_location = [x,y,(x + w),(y + h)]

    predicted_rect = [ # 使用卡尔曼滤波的预测值绘制预测的矩形框
        int(x_hat[0]),# 预测的矩形框左上角x坐标
        int(x_hat[1]),# 预测的矩形框左上角y坐标
        int(x_hat[2]), # 预测的矩形框宽度
        int(x_hat[3]) # 预测的矩形框高度
    ]
    # 绘制矩形框
    img.draw_rectangle(predicted_rect, color=(100, 100, 100))  # 使用灰色绘制矩形框
    #下面是把矩形转化会圆绘制的，我们不需要
#    # 计算圆心坐标和半径
    center_x = int(x_hat[0] + x_hat[2] / 2)  # 圆心x坐标（矩形框左上角x + 宽度的一半）
    center_y = int(x_hat[1] + x_hat[3] / 2)  # 圆心y坐标（矩形框左上角y + 高度的一半）
    radius = int(min(x_hat[2], x_hat[3]) / 2)  # 半径取矩形框宽度和高度的最小值的一半
    # 输出圆心坐标和半径信息
    print("卡尔曼计算       Predicted Circle: center_x = {}, center_y = {}, radius
```

```
#    # 计算圆心坐标和半径
    center_x = int(x_hat[0] + x_hat[2] / 2)  # 圆心x坐标（矩形框左上角x + 宽度的一半）
    center_y = int(x_hat[1] + x_hat[3] / 2)  # 圆心y坐标（矩形框左上角y + 高度的一半）
    radius = int(min(x_hat[2], x_hat[3]) / 2)  # 半径取矩形框宽度和高度的最小值的一半
    # 输出圆心坐标和半径信息
    print("卡尔曼计算        Predicted Circle: center_x = {}, center_y = {}, radius
= {}".format(center_x, center_y, radius))

    # 打印当前帧率（每秒帧数），便于调试性能
    print("FPS %f" % clock.fps())  # 输出当前的帧率
```

【13】All_in结合篇@识别形状，颜色，最小阈值，最大阈值，Kanman滤波，指定区域

3.4.9-识别形状+颜色+最小变化阈值+增加最大变化阈值+卡尔曼滤波与运动估计+通过指定区域获得阈值

我们要增加通过指定区域获得阈值的功能，可以把"3.2.4-通过指定区域获得阈值介绍和单独实现"章节的代码移植进 到我们的"3.4.4章节"

```
import sensor, image, time
#*********************增加卡尔曼滤波
*****************************************************************************
import sensor, image, time, math
from ulab import numpy as np
#教程作者:好家伙VCC
#欢迎交流群QQ: 771027961 作者邮箱: 1930299709@qq.com
#更多教程B站主页:[好家伙VCC的个人空间-好家伙VCC个人主页-哔哩哔哩视频]
(https://space.bilibili.com/434192043)
#淘宝主页链接:[首页-好家伙VCC-淘宝网](https://shop415231378.taobao.com)
#更多嵌入式手把手教程-尽在好家伙VCC
# 定义颜色阈值（L, A, B），用于识别红色
# L通道：亮度值，较小表示较暗的颜色
# A通道：绿色与红色的色差，红色偏大
# B通道：蓝色与黄色的色差，红色偏小
color_threshold = (0, 100, 0, 127, 0, 127)  # (L_min, L_max, A_min, A_max,
B_min, B_max)

# 初始化摄像头模块
sensor.reset()  # 重置摄像头，确保设备正常工作
sensor.set_pixformat(sensor.RGB565)  # 设置摄像头的像素格式为RGB565，每个像素16位色深
sensor.set_framesize(sensor.QQVGA)  # 设置摄像头的分辨率为QQVGA（160x120），适合快速处
理

# *************************** 如果不需要镜像就注释掉以下代码
***************************
# 摄像头镜像和翻转设置，根据摄像头的安装方向调整
sensor.set_vflip(True)  # 设置垂直翻转，适用于摄像头上下安装的情况
sensor.set_hmirror(True)  # 设置水平翻转，适用于摄像头左右安装的情况
# *************************** 如果不需要镜像就注释掉以上代码
***************************


sensor.skip_frames(time = 2000)  # 跳过前几帧的图像，确保图像稳定后再开始处理
sensor.set_auto_gain(False)  # 必须关闭自动增益，防止影响颜色追踪
sensor.set_auto_whitebal(False)  # 必须关闭自动白平衡，防止影响颜色追踪

# 创建一个时钟对象，用于计算和控制帧率
clock = time.clock()

# *************************** 最小变化阈值滤波 ***************************
# 位置和半径变化阈值，只有变化大于该阈值时才更新
position_threshold = 4  # 位置变化的最小阈值 两个值越小小球识别更新的越频繁，值越大小球的细
微运动越不会更新识别
radius_threshold = 4    # 半径变化的最小阈值 可以根据自己的需求测试调节这个两个阈值
MAX_CHANGE_THRESHOLD = 80  # 最大变化阈值（例如位置或半径变化超过此值时不更新）
# 上一帧的圆心坐标和半径
prev_x, prev_y, prev_r = None, None, None
# *************************** 最小变化阈值滤波 ***************************
#****************[0]-获取指定位置阈值*******************
threshold_calculated = False #控制阈值计算只执行一次的标志
threshold_roi = (70,50,20,20) #会通过这个位置获得追踪阈值
# 封装为函数：识别指定区域的阈值
def get_threshold(roi):
    # 循环多次（默认150次）更新阈值
    threshold = [0, 0, 0, 0, 0, 0]  # LAB色彩通道的阈值 [Lmin, Lmax, Amin, Amax,
Bmin, Bmax]
    for _ in range(150):
        img = sensor.snapshot()
        # 获取指定区域的颜色直方图
        hist = img.get_histogram(roi=roi)
        img.draw_rectangle(roi, color=(0, 255, 0), thickness=2)  # 使用绿色（0,
255, 0），厚度为2# 在图像上绘制绿色矩形框标识采集区域
        # 在绿色矩形框上方显示"采集计算阈值中..."并加上省略号
        img.draw_string(roi[0], roi[1] - 10, "Collecting Threshold...", color=
(0, 255, 0), scale=1)
        # 获取L、A、B三个通道的5%和95%分位值
        lo = hist.get_percentile(0.05)  # 获取5%分位值，表示颜色分布的下边界
        hi = hist.get_percentile(0.95)  # 获取95%分位值，表示颜色分布的上边界
        print("采集计算阈值中...请等待")  # 打印检查结果，1表示满足，0表示不满足
        # 输出lo和hi的值
#        print(f"5% Percentile (lo): L={lo.l_value()} A={lo.a_value()} B=
{lo.b_value()}")
#        print(f"95% Percentile (hi): L={hi.l_value()} A={hi.a_value()} B=
{hi.b_value()}")
        # L通道的最小值和最大值平均后作为新的阈值
        threshold[0] = (threshold[0] + lo.l_value()) // 2  # L通道的最小值
        threshold[1] = (threshold[1] + hi.l_value()) // 2  # L通道的最大值
        # A通道的最小值和最大值平均后作为新的阈值
```

```
            #x_hat[0] 和 x_hat[1] 是预测的位置，x_hat[4] 和 x_hat[5] 是目标的速度（速度
分别对应 dx 和 dy）。
            #Ts 是时间步长。更新后的 x, y 位置是通过加入速度（dx 和 dy）的影响来计算的。
            #w 和 h 是目标的宽度和高度，这些信息没有改变，仍然使用 x_hat[2] 和 x_hat[3]。
            x,y,w,h = (x_hat[0] + (x_hat[4] * Ts)),(x_hat[1] + (x_hat[5] *
Ts)),x_hat[2],x_hat[3]
            #解释：计算目标在 X 方向的速度（dx），通过当前帧的 x 位置与上一帧的
last_frame_x 位置之差来计算，除以时间步长 Ts
            dx = (x - last_frame_x) / Ts
            #解释：计算目标在 Y 方向的速度（dy），通过当前帧的 y 位置与上一帧的
last_frame_y 位置之差来计算，除以时间步长 Ts
            dy = (y - last_frame_y) / Ts
            #解释：将目标的位置、宽高、速度（dx 和 dy）组合成一个状态向量 Z，这个向量将作为卡
尔曼滤波器的输入
            Z = np.array([x, y, w, h, dx, dy])
            #解释：将状态向量 Z 传递给卡尔曼滤波器 Kalman_Filter，以获得更新后的预测状态
x_hat
            x_hat = Kalman_Filter(Z)
            #更新上一帧的目标位置 last_frame_x 和 last_frame_y 为当前帧的目标位置 x 和
y，以便在下一帧计算速度时使用
            last_frame_x, last_frame_y = x, y
            #更新上一帧的目标矩形框 last_frame_rect 为当前帧的矩形框 x, y, w, h。
            last_frame_rect = [x,y,w,h]
            #更新上一帧的目标位置 last_frame_location 为当前帧的目标位置。这是一个矩形框的
坐标，包括左上角 (x, y) 和右下角 (x + w, y + h)
            last_frame_location = [x,y,(x + w),(y + h)]

    predicted_rect = [ # 使用卡尔曼滤波的预测值绘制预测的矩形框
        int(x_hat[0]),# 预测的矩形框左上角x坐标
        int(x_hat[1]),# 预测的矩形框左上角y坐标
        int(x_hat[2]), # 预测的矩形框宽度
        int(x_hat[3]) # 预测的矩形框高度
    ]
    # 绘制矩形框
    img.draw_rectangle(predicted_rect, color=(100, 100, 100))  # 使用灰色绘制矩形框
    #下面是把矩形转化会圆绘制的，我们不需要
#    # 计算圆心坐标和半径
    center_x = int(x_hat[0] + x_hat[2] / 2)  # 圆心x坐标（矩形框左上角x + 宽度的一半）
    center_y = int(x_hat[1] + x_hat[3] / 2)  # 圆心y坐标（矩形框左上角y + 高度的一半）
    radius = int(min(x_hat[2], x_hat[3]) / 2)  # 半径取矩形框宽度和高度的最小值的一半
    # 输出圆心坐标和半径信息
    print("卡尔曼计算        Predicted Circle: center_x = {}, center_y = {}, radius
= {}".format(center_x, center_y, radius))

    # 打印当前帧率（每秒帧数），便于调试性能
    print("FPS %f" % clock.fps())  # 输出当前的帧率
```

```
import sensor, time
from ulab import numpy as np

color_threshold = (0, 100, 0, 127, 0, 127)

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.set_vflip(True)
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)
clock = time.clock()

position_threshold = 4
radius_threshold = 4
MAX_CHANGE_THRESHOLD = 80
prev_x, prev_y, prev_r = None, None, None
threshold_calculated = False
threshold_roi = (70,50,20,20)

def get_threshold(roi):
    threshold = [0, 0, 0, 0, 0, 0]
    for _ in range(150):
        img = sensor.snapshot()
        hist = img.get_histogram(roi=roi)
        img.draw_rectangle(roi, color=(0, 255, 0), thickness=2)
        img.draw_string(roi[0], roi[1] - 10, "Collecting Threshold...",\
        color=(0, 255, 0), scale=1)
        lo = hist.get_percentile(0.05)
        hi = hist.get_percentile(0.95)
        print("采集计算阈值中...请等待")
        threshold[0] = (threshold[0] + lo.l_value()) // 2
        threshold[1] = (threshold[1] + hi.l_value()) // 2
```

```
#       print(f"95% Percentile (hi): L={hi.l_value()} A={hi.a_value()} B={
{hi.b_value()}")
        # L通道的最小值和最大值平均后作为新的阈值
        threshold[0] = (threshold[0] + lo.l_value()) // 2  # L通道的最小值
        threshold[1] = (threshold[1] + hi.l_value()) // 2  # L通道的最大值
        # A通道的最小值和最大值平均后作为新的阈值
        threshold[2] = (threshold[2] + lo.a_value()) // 2  # A通道的最小值
        threshold[3] = (threshold[3] + hi.a_value()) // 2  # A通道的最大值
        # B通道的最小值和最大值平均后作为新的阈值
        threshold[4] = (threshold[4] + lo.b_value()) // 2  # B通道的最小值
        threshold[5] = (threshold[5] + hi.b_value()) // 2  # B通道的最大值


    print(f"计算阈值的位置区域是 ROI Info: x={roi[0]}, y={roi[1]}, width={roi[2]},
height={roi[3]}")  # 输出roi区域的信息
    # 打印每个通道的阈值信息
    print("计算出的阈值  Threshold: Lmin={0} Lmax={1}, Amin={2} Amax={3}, Bmin={4}
Bmax={5}".format(
        threshold[0], threshold[1], threshold[2], threshold[3], threshold[4],
threshold[5]
    ))


    # 返回计算得到的阈值列表，包含L、A、B三个通道的最小值和最大值
    return threshold  # 返回最终的阈值数组
#**********************增加卡尔曼滤波**************************8
Ts = 1/22 #Ts = 1  帧率的倒数
# 状态空间矩阵定义
 # 状态转移矩阵 A，描述系统的状态变化
#A = np.array([[1,0,0,0,Ts,0],[0,1,0,0,0,Ts],[0,0,1,0,0,0],[0,0,0,1,0,0],
[0,0,0,0,1,0],[0,0,0,0,0,1]])
# 改进状态转移矩阵A，将速度部分更明确地包含进来A 是状态转移矩阵，描述了系统状态（位置、速度等）
的变化。该矩阵会把当前的状态转换到下一时刻的状态。
A = np.array([[1, 0, 0, 0, Ts, 0],
              [0, 1, 0, 0, 0, Ts],
              [0, 0, 1, 0, 0, 0],
              [0, 0, 0, 1, 0, 0],
              [0, 0, 0, 0, 1, 0],
              [0, 0, 0, 0, 0, 1]])


 # 观测矩阵 C，描述从状态到观测值的映射关系  C 是观测矩阵，它将状态向量（位置、速度）与观测量
（图像中的矩形框信息）联系起来。这里假设观测量是位置和速度。
C = np.array([[1,0,0,0,0,0],[0,1,0,0,0,0],[0,0,1,0,0,0],[0,0,0,1,0,0],
[0,0,0,0,1,0],[0,0,0,0,0,1]])
# 过程噪声协方差矩阵
# 过程噪声协方差矩阵 Q，用于描述过程的随机噪声
#Q_value = [1e-8 for _ in range(6)]
#Q = np.diag(Q_value) #创建对角矩阵
# 增大过程噪声矩阵Q_value，使得卡尔曼滤波更灵活地应对物体快速运动
#Q 是过程噪声协方差矩阵，用于描述系统过程中的不确定性。矩阵的对角元素表示每个状态变量的噪声水
平。
Q_value = [1e-6 for _ in range(6)]  # 调整过程噪声值Q_value = [1e-6 for _ in
range(6)]
Q = np.diag(Q_value)  # 更新过程噪声协方差矩阵

# 观测噪声协方差矩阵  R 是观测噪声协方差矩阵，表示观测过程中测量误差的大小。
R_value = [1e-6 for _ in range(6)]
R = np.diag(R_value)
# 定义观测量Z
x = 0 #左顶点x坐标
y = 0 #左顶点y坐标
last_frame_x = x #上一帧左顶点x坐标
last_frame_y = y #上一帧左顶点y坐标
w = 0 #矩形框宽度w
h = 0 #矩形框高度h
dx = 0 #左顶点x坐标移动速度
dy = 0 #左顶点y坐标移动速度
Z = np.array([x,y,w,h,dx,dy])
# 定义卡尔曼滤波函数变量
#x_hat = np.array([80,60,30,30,2,2]) # 初始估计的状态值（位置、速度等）
x_hat = np.array([80, 60, 30, 30, 2, 2])  # 根据你的应用需要调整这些值

x_hat_minus = np.array([0,0,0,0,0,0]) # 初始预测的状态值
p_value = [10 for _ in range(6)] # 状态误差的初始值 p 是状态误差的初始协方差矩阵。
p = np.diag(p_value)# 创建误差协方差矩阵 p
#**********************增加卡尔曼滤波
*************************************************************************


# 卡尔曼滤波函数
#预测阶段：利用状态转移矩阵和上一状态估计预测当前状态。
#校正阶段：通过卡尔曼增益对预测状态进行校正，使得估计值接近真实值。
#输入 Z：观测值（或测量值），通常是来自外部传感器（例如相机、雷达等）的数据。在这个代码中，Z 是
一个包含目标的位置信息（如矩形框的四个角坐标）的向量，格式为 [x, y, w, h, dx, dy]，其中 x 和
y 是目标的中心位置，w 和 h 是目标的宽度和高度，dx 和 dy 是目标的速度。
#输出 x_hat：更新后的状态估计，包括位置（x, y）、宽度（w, h）、速度（dx, dy）。该值是通过卡尔
曼滤波器的预测和校正步骤计算得到的最优估计。
def Kalman_Filter(Z):
    global A,C,Q,R,x_hat,x_hat_minus,p
    # 预测部分
    x_hat_minus = np.dot(A,x_hat)
    p_minus = np.dot(A, np.dot(p, A.T)) + Q
    # 校正部分
    S = np.dot(np.dot(C, p_minus), C.T) + R
    # 选择一个小的正则化项
    regularization_term = 1e-4
    # 正则化 S 矩阵
    S_regularized = S + regularization_term * np.eye(S.shape[0])
    # 计算正则化后的 S 矩阵的逆
    S_inv = np.linalg.inv(S_regularized)
    # 计算卡尔曼增益 K
    K = np.dot(np.dot(p_minus, C.T), S_inv)
    x_hat = x_hat_minus + np.dot(K,(Z - np.dot(C,x_hat_minus)))
    p = np.dot((np.eye(6) - np.dot(K,C)),p_minus)
    return x_hat
last_frame_location = [0 for _ in range(4)] #用于存储上一帧的目标位置，这通常用于目标跟
踪和计算目标移动等任务。一个长度为4的列表 last_frame_location，其中每个元素的初始值为 0
last_frame_rect = [0 for _ in range(4)]  #存储上一帧检测到的矩形框坐标 成了一个长度为4的
```

```
        color=(0, 255, 0), scale=1)
        lo = hist.get_percentile(0.05)
        hi = hist.get_percentile(0.95)
        print("采集计算阈值中...请等待")
        threshold[0] = (threshold[0] + lo.l_value()) // 2
        threshold[1] = (threshold[1] + hi.l_value()) // 2
        threshold[2] = (threshold[2] + lo.a_value()) // 2
        threshold[3] = (threshold[3] + hi.a_value()) // 2
        threshold[4] = (threshold[4] + lo.b_value()) // 2
        threshold[5] = (threshold[5] + hi.b_value()) // 2
    print(f"计算阈值的位置区域是 ROI Info: x={roi[0]}, y={roi[1]}, \
    width={roi[2]}, height={roi[3]}")
    print("计算出的阈值  Threshold: Lmin={0} Lmax={1}, \
    Amin={2} Amax={3}, Bmin={4} Bmax={5}".format(
    threshold[0], threshold[1], threshold[2], threshold[3], threshold[4], threshold[5]))
    return threshold

Ts = 1/12 #Ts = 1 帧率的倒数
A = np.array([[1, 0, 0, 0, Ts, 0],
              [0, 1, 0, 0, 0, Ts],
              [0, 0, 1, 0, 0, 0],
              [0, 0, 0, 1, 0, 0],
              [0, 0, 0, 0, 1, 0],
              [0, 0, 0, 0, 0, 1]])
C = np.array([[1,0,0,0,0,0],[0,1,0,0,0,0],\
[0,0,1,0,0,0],[0,0,0,1,0,0],[0,0,0,0,1,0],[0,0,0,0,0,1]])
Q_value = [1e-6 for _ in range(6)]
Q = np.diag(Q_value)
R_value = [1e-6 for _ in range(6)]
R = np.diag(R_value)

x = 0
y = 0
last_frame_x = x
last_frame_y = y
w = 0
h = 0
dx = 0
dy = 0
Z = np.array([x,y,w,h,dx,dy])
x_hat = np.array([80, 60, 30, 30, 2, 2])
x_hat_minus = np.array([0,0,0,0,0,0])
p_value = [10 for _ in range(6)]
p = np.diag(p_value)

def Kalman_Filter(Z):
    global A,C,Q,R,x_hat,x_hat_minus,p
    x_hat_minus = np.dot(A,x_hat)
    p_minus = np.dot(A, np.dot(p, A.T)) + Q
    S = np.dot(np.dot(C, p_minus), C.T) + R
    regularization_term = 1e-4
    S_regularized = S + regularization_term * np.eye(S.shape[0])
    S_inv = np.linalg.inv(S_regularized)
    K = np.dot(np.dot(p_minus, C.T), S_inv)
    x_hat = x_hat_minus + np.dot(K,(Z - np.dot(C,x_hat_minus)))
    p = np.dot((np.eye(6) - np.dot(K,C)),p_minus)
    return x_hat
last_frame_location = [0 for _ in range(4)]
last_frame_rect = [0 for _ in range(4)]
box = [0 for _ in range(4)]


while(True):
    clock.tick()
    img = sensor.snapshot().lens_corr(1.8)
    if not threshold_calculated:#获取指定位置阈值-进行阈值计算的内容
        color_threshold = get_threshold(threshold_roi)
        threshold_calculated = True

    for c in img.find_circles(
        threshold = 2500,  # 设置圆形检测的阈值，较高意味着需更明显圆才能被检测到
        x_margin = 10,     # 圆心的X坐标允许的误差范围
        y_margin = 10,     # 圆心的Y坐标允许的误差范围
        r_margin = 10,     # 圆半径的允许误差范围
        r_min = 2,         # 圆的最小半径 单位为像素
        r_max = 100,       # 圆的最大半径 单位为像素
        r_step = 2):       # 圆半径变化的步长 单位像素
        area = (c.x() - c.r(), c.y() - c.r(), 2 * c.r(), 2 * c.r())  # (x, y, width, height)
        statistics = img.get_statistics(roi=area)

        if (
            color_threshold[0] < statistics.l_mode() < color_threshold[1] and
            color_threshold[2] < statistics.a_mode() < color_threshold[3] and
            color_threshold[4] < statistics.b_mode() < color_threshold[5] ):
            x, y, r = c.x(), c.y(), c.r()
            if prev_x is None or prev_y is None or prev_r is None:
                prev_x, prev_y, prev_r = x, y, r  #更新上次的位置值
                img.draw_circle(x, y, r, color=(192, 255, 0)) #绘制圆为绿色
                # 输出圆心坐标和半径
                print("第一次检测       First detection: center_x = {}, \
                center_y = {}, radius = {}".format(x, y, r))
                circle_center_x = x
                circle_center_y = y
                circle_radius = r
            else:
                x_change = abs(x - prev_x)
                y_change = abs(y - prev_y)
                r_change = abs(r - prev_r)
                if (
                    (x_change > position_threshold or y_change > \
                    position_threshold or r_change > radius_threshold) and
                    (x_change <= MAX_CHANGE_THRESHOLD and y_change <= \
                    MAX_CHANGE_THRESHOLD and r_change <= MAX_CHANGE_THRESHOLD)):
                    prev_x, prev_y, prev_r = x, y, r
                    img.draw_circle(x, y, r, color=(192, 255, 0))
                    print("更新检测       Updated detection: center_x = \
                    {}, center_y = {}, radius = {}".format(x, y, r))
                    circle_center_x = x
                    circle_center_y = y
                    circle_radius = r
                else :
                    img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                    print("没有显著变化 No significant change: center_x = \
                    {}, center_y = {}, radius = {}".format(prev_x, prev_y, prev_r))
                    circle_center_x = prev_x
                    circle_center_y = prev_y
```

```
    p = np.dot((np.eye(6) - np.dot(K,C)),p_minus)
    return x_hat
last_frame_location = [0 for _ in range(4)] #用于存储上一帧的目标位置，这通常用于目标跟踪和计算目标移动等任务。一个长度为4的列表 last_frame_location，其中每个元素的初始值为 0
last_frame_rect = [0 for _ in range(4)]  #存储上一帧检测到的矩形框坐标 成了一个长度为4的列表 last_frame_rect，并且每个元素的初始值为 0。
box = [0 for _ in range(4)] #生成一个包含四个子列表的列表,存储x1,y1,x2,y2

# 主循环，不断获取摄像头图像并进行处理
while(True):
    clock.tick()  # 计时当前帧的处理时间，计算帧率

    # 获取当前图像并进行镜头畸变校正，纠正因镜头产生的畸变
    img = sensor.snapshot().lens_corr(1.8)  # 1.8是畸变系数，适当调整可以改善图像质量

#*****************[2]-获取指定位置阈值-进行阈值计算的内容********************
    if not threshold_calculated:# 仅在阈值未计算时进行计算
        # 调用函数获取指定区域的阈值
        color_threshold = get_threshold(threshold_roi)

        # 设置阈值计算完成的标志
        threshold_calculated = True

    # 使用霍夫变换查找圆形，并返回找到的圆的信息
    for c in img.find_circles(
        threshold = 2500,  # 设置圆形检测的阈值。较高的值意味着需要更明显的圆形才能被检测到
        x_margin = 10,      # 圆心的X坐标允许的误差范围
        y_margin = 10,      # 圆心的Y坐标允许的误差范围
        r_margin = 10,      # 圆半径的允许误差范围
        r_min = 2,          # 圆的最小半径 单位为像素。
        r_max = 100,        # 圆的最大半径 单位为像素。100 像素对应多少毫米是一个动态计算的问题，需要根据具体的摄像头视场角、分辨率和物体距离来调整。
        r_step = 2          # 圆半径变化的步长 单位为像素。
    ):
        # 计算圆形的外接矩形区域，这样可以方便获取圆的统计信息
        area = (c.x() - c.r(), c.y() - c.r(), 2 * c.r(), 2 * c.r())  # (x, y, width, height)
        # 获取该区域内的像素颜色统计信息
        statistics = img.get_statistics(roi=area)  # 获取外接矩形区域的像素统计信息（颜色分布）

        # 打印该区域的颜色统计数据，用于调试
        #print(statistics)

        # 判断该区域是否为红色圆
        # 使用L、A、B通道的众数来判断颜色是否符合红色范围
        if (
            color_threshold[0] < statistics.l_mode() < color_threshold[1] and  # L通道的众数应小于100，表示较暗的颜色
            color_threshold[2] < statistics.a_mode() < color_threshold[3] and  # A通道的众数应小于127，表示偏红色
            color_threshold[4] < statistics.b_mode() < color_threshold[5]      # B通道的众数应小于127，表示偏蓝色
        ):
            # 获取当前圆心和半径
            x, y, r = c.x(), c.y(), c.r()


            # 如果是第一次检测，更新值
            if prev_x is None or prev_y is None or prev_r is None:
                prev_x, prev_y, prev_r = x, y, r  #更新上次的位置值
                # 绘制圆形
                img.draw_circle(x, y, r, color=(192, 255, 0)) #绘制圆为绿色
                # 输出圆心坐标和半径
                print("第一次检测       First detection: center_x = {}, center_y = {}, radius = {}".format(x, y, r))
                circle_center_x = x
                circle_center_y = y
                circle_radius = r

            else:
                # 判断位置和半径变化是否大于阈值
                x_change = abs(x - prev_x)#计算这次值和上次值 的绝对值
                y_change = abs(y - prev_y)
                r_change = abs(r - prev_r)

                # 判断变化是否大于最小阈值，并且变化小于最大阈值
                if (
                    (x_change > position_threshold or y_change > position_threshold or r_change > radius_threshold) and
                    (x_change <= MAX_CHANGE_THRESHOLD and y_change <= MAX_CHANGE_THRESHOLD and r_change <= MAX_CHANGE_THRESHOLD)
                ):
                    prev_x, prev_y, prev_r = x, y, r
                    # 绘制更新后的圆形
                    img.draw_circle(x, y, r, color=(192, 255, 0))
                    # 输出圆心坐标和半径
                    print("更新检测       Updated detection: center_x = {}, center_y = {}, radius = {}".format(x, y, r))
                    circle_center_x = x
                    circle_center_y = y
                    circle_radius = r

                else :
                    # 变化小于阈值，绘制上次的坐标和半径
                    img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                    # 输出圆心坐标和半径（使用上次的坐标和半径）
                    print("没有显著变化 No significant change: center_x = {}, center_y = {}, radius = {}".format(prev_x, prev_y, prev_r))
                    circle_center_x = prev_x
                    circle_center_y = prev_y
                    circle_radius = prev_r


            # 假设圆的中心 (c.x(), c.y()) 和半径 c.r()

            # 计算矩形框的左上角坐标和宽高
```

```
                    img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                    print("没有显著变化 No significant change: center_x = \
                    {}, center_y = {}, radius = {}".format(prev_x, prev_y, prev_r))
                    circle_center_x = prev_x
                    circle_center_y = prev_y
                    circle_radius = prev_r
            rect_x = int(circle_center_x - circle_radius)
            rect_y = int(circle_center_y - circle_radius)
            rect_w = int(2 * circle_radius)
            rect_h = int(2 * circle_radius)
            rect = [rect_x, rect_y, rect_w, rect_h]

            box = [rect[0], rect[1], rect[0] + rect[2], rect[1] + rect[3]]
            x, y, w, h = rect[0], rect[1], rect[2], rect[3]
            dx = (x - last_frame_x) / Ts
            dy = (y - last_frame_y) / Ts
            Z = np.array([x, y, w, h, dx, dy])
            x_hat = Kalman_Filter(Z)
            last_frame_x, last_frame_y = x, y
            last_frame_rect = rect
            last_frame_location = box

        else:
            x,y,w,h = (x_hat[0] + (x_hat[4] * Ts)),(x_hat[1] + (x_hat[5] * Ts)),x_hat[2],x_hat[3]
            dx = (x - last_frame_x) / Ts
            dy = (y - last_frame_y) / Ts
            Z = np.array([x, y, w, h, dx, dy])
            x_hat = Kalman_Filter(Z)
            last_frame_x, last_frame_y = x, y
            last_frame_rect = [x,y,w,h]
            last_frame_location = [x,y,(x + w),(y + h)]
        predicted_rect = [
            int(x_hat[0]),
            int(x_hat[1]),
            int(x_hat[2]),
            int(x_hat[3])]
        img.draw_rectangle(predicted_rect, color=(100, 100, 100))
        center_x = int(x_hat[0] + x_hat[2] / 2)
        center_y = int(x_hat[1] + x_hat[3] / 2)
        radius = int(min(x_hat[2], x_hat[3]) / 2)
        print("卡尔曼计算       Predicted Circle: center_x = {}, center_y = \
        {}, radius = {}".format(center_x, center_y, radius))

        print("FPS %f" % clock.fps()) #输出当前的帧率


```

```
                    prev_x, prev_y, prev_r = x, y, r
                    # 绘制更新后的圆形
                    img.draw_circle(x, y, r, color=(192, 255, 0))
                    # 输出圆心坐标和半径
                    print("更新检测       Updated detection: center_x = {},
    center_y = {}, radius = {}".format(x, y, r))
                    circle_center_x = x
                    circle_center_y = y
                    circle_radius = r

                else :
                    # 变化小于阈值，绘制上次的坐标和半径
                    img.draw_circle(prev_x, prev_y, prev_r, color=(192, 255, 0))
                    # 输出圆心坐标和半径（使用上次的坐标和半径）
                    print("没有显著变化 No significant change: center_x = {},
    center_y = {}, radius = {}".format(prev_x, prev_y, prev_r))
                    circle_center_x = prev_x
                    circle_center_y = prev_y
                    circle_radius = prev_r


            # 假设圆的中心 (c.x(), c.y()) 和半径 c.r()


            # 计算矩形框的左上角坐标和宽高
            rect_x = int(circle_center_x - circle_radius)  # 矩形框左上角的 x 坐标
            rect_y = int(circle_center_y - circle_radius)  # 矩形框左上角的 y 坐标
            rect_w = int(2 * circle_radius)  # 矩形框的宽度，等于圆的直径
            rect_h = int(2 * circle_radius)  # 矩形框的高度，等于圆的直径

            # 将矩形框的四个参数存储到 rect 数组中
            rect = [rect_x, rect_y, rect_w, rect_h]


            box = [rect[0], rect[1], rect[0] + rect[2], rect[1] + rect[3]] # 计算
    标记框的四个角坐标（左上角和右下角）
            x, y, w, h = rect[0], rect[1], rect[2], rect[3]# 获取新的矩形框的坐标 识
    别成功的时候这个值是识别值的数据赋值
            dx = (x - last_frame_x) / Ts# 计算x方向的速度，假设 Ts 为时间步长
            dy = (y - last_frame_y) / Ts# 计算y方向的速度
            Z = np.array([x, y, w, h, dx, dy]) # 构造测量向量 Z，包括位置和速度
            x_hat = Kalman_Filter(Z)# 使用卡尔曼滤波器进行状态估计
            last_frame_x, last_frame_y = x, y # 更新上一帧的x，y坐标
            #img.draw_rectangle(last_frame_rect, color = (0, 0, 255))
            #img.draw_rectangle(rect, color = (255, 0, 0)) # 用红色矩形框绘制目标位置
            last_frame_rect = rect# 更新上一帧的矩形框 识别成功的时候
            last_frame_location = box# 更新上一帧的位置 识别成功的时候



        else:
            # 如果不是红色圆形，用白色矩形框标记该区域
            #img.draw_circle(c.x(), c.y(), c.r(), color=(255, 255, 255))  # 白色
```

【14】圆环识别和执行结构

目的最后是控制小车把东西放在圆环中间

```
import sensor, image, time
from ulab import numpy as np

#教程作者:好家伙VCC
#欢迎交流群QQ: 771027961 作者邮箱: 1930299709@qq.com
#更多教程B站主页:[好家伙VCC的个人空间-好家伙VCC个人主页-哔哩哔哩视频](https://space.bilibili.com/434192043)
#淘宝主页链接:[首页-好家伙VCC-淘宝网](https://shop415231378.taobao.com)
#更多嵌入式手把手教程-尽在好家伙VCC
# ********** 参数配置 **********
# 颜色阈值配置（LAB格式）
color_thresholds = {
    'red':   (0, 100, 0, 127, 0, 127),  # 红色的LAB颜色范围
    'green': (0, 100, -128, -10, 0, 127),  # 绿色的LAB颜色范围
    'blue':  (0, 100, -128, 127, -128, -10)  # 蓝色的LAB颜色范围
}

# 霍夫变换参数
HOUGH_THRESHOLD = 2000   # 圆形检测的灵敏度，值越高要求边缘越明显
MIN_RADIUS = 10          # 检测的最小半径
MAX_RADIUS = 50          # 检测的最大半径

# 卡尔曼滤波参数
TS = 1/60                # 帧时间（假设帧率为60fps）

# 初始化摄像头
sensor.reset()  # 重置摄像头
sensor.set_pixformat(sensor.RGB565)  # 设置像素格式为RGB565
sensor.set_framesize(sensor.QQVGA)  # 设置分辨率为QQVGA（160x120）
sensor.set_vflip(True)  # 垂直翻转图像
sensor.set_hmirror(True)  # 水平翻转图像
```

升降式结构

升降机构中把摄像头安装在升级机构的上端

```
sensor.reset()  # 重置摄像头
sensor.set_pixformat(sensor.RGB565)  # 设置像素格式为RGB565
sensor.set_framesize(sensor.QQVGA)  # 设置分辨率为QQVGA（160x120）
sensor.set_vflip(True)  # 垂直翻转图像
sensor.set_hmirror(True)  # 水平翻转图像
sensor.set_auto_gain(False)  # 关闭自动增益
sensor.set_auto_whitebal(False)  # 关闭自动白平衡
clock = time.clock()  # 创建时钟对象用于计算帧率

# ******************** 卡尔曼滤波器类 ********************
class KalmanFilter:
    def __init__(self, initial_state):
        # 状态转移矩阵
        self.A = np.array([
            [1, 0, 0, 0, TS, 0],  # 位置和速度的状态转移
            [0, 1, 0, 0, 0, TS],
            [0, 0, 1, 0, 0, 0],
            [0, 0, 0, 1, 0, 0],
            [0, 0, 0, 0, 1, 0],
            [0, 0, 0, 0, 0, 1]
        ])
        # 观测矩阵
        self.C = np.eye(6)  # 单位矩阵，表示状态和观测值直接对应
        # 过程噪声协方差矩阵
        self.Q = np.diag([1e-6]*6)  # 过程噪声较小
        # 观测噪声协方差矩阵
        self.R = np.diag([1e-6]*6)  # 观测噪声较小
        # 初始状态
        self.x_hat = initial_state  # 初始状态估计值
        self.p = np.diag([10]*6)  # 初始误差协方差矩阵

    def update(self, Z):
        # 预测步骤
        x_hat_minus = np.dot(self.A, self.x_hat)  # 预测状态
        p_minus = np.dot(self.A, np.dot(self.p, self.A.T)) + self.Q  # 预测误差协方差

        # 更新步骤
        S = np.dot(self.C, np.dot(p_minus, self.C.T)) + self.R  # 计算卡尔曼增益的分母
        S_inv = np.linalg.inv(S + 1e-4*np.eye(6))  # 计算逆矩阵，加入正则化项避免奇异矩阵
        K = np.dot(np.dot(p_minus, self.C.T), S_inv)  # 计算卡尔曼增益
        self.x_hat = x_hat_minus + np.dot(K, (Z - np.dot(self.C, x_hat_minus)))  # 更新状态估计
        self.p = np.dot((np.eye(6) - np.dot(K, self.C)), p_minus)  # 更新误差协方差
        return self.x_hat

# 初始化三个卡尔曼滤波器（分别对应红、绿、蓝）
kf_red = KalmanFilter(np.array([80, 60, 30, 30, 2, 2]))  # 红色滤波器的初始状态
kf_green = KalmanFilter(np.array([80, 60, 30, 30, 2, 2]))  # 绿色滤波器的初始状态
kf_blue = KalmanFilter(np.array([80, 60, 30, 30, 2, 2]))  # 蓝色滤波器的初始状态

# ******************** 主循环 ********************
while True:
    clock.tick()  # 记录当前帧的时间
    img = sensor.snapshot().lens_corr(1.8)  # 获取图像并校正镜头畸变
    # ===== 在画幅中心绘制小圆环标志 =====
    img.draw_circle(80, 60, 5, color=(0, 0, 0), thickness=1)  # 黑色小圆环，半径5像素
    # 分别处理红、绿、蓝三种颜色
    for color, threshold in color_thresholds.items():
        # 第一步：颜色阈值分割，找到当前颜色的区域
        blobs = img.find_blobs([threshold], merge=True, margin=10)  # 查找当前颜色的区域

        if blobs:
            # 取最大的色块
            largest_blob = max(blobs, key=lambda b: b.area())  # 找到面积最大的当前颜色区域
            # 绘制当前颜色的矩形框
            if color == 'red':
                rect_color = (255, 0, 0)  # 红色
            elif color == 'green':
                rect_color = (0, 255, 0)  # 绿色
            else:
                rect_color = (0, 0, 255)  # 蓝色
            img.draw_rectangle(largest_blob.rect(), color=rect_color)  # 绘制矩形框

            # 第二步：在当前颜色的区域内检测圆形
            roi = (largest_blob.x(), largest_blob.y(), largest_blob.w(), largest_blob.h())  # 定义检测区域
            circles = img.find_circles(
                threshold=HOUGH_THRESHOLD,  # 圆形检测的灵敏度
                x_margin=10,  # 圆心x坐标的误差范围
                y_margin=10,  # 圆心y坐标的误差范围
                r_margin=10,  # 半径的误差范围
                r_min=MIN_RADIUS,  # 最小半径
                r_max=MAX_RADIUS,  # 最大半径
                roi=roi  # 限制检测区域为当前颜色区域
            )

            if circles:
                # 筛选同心圆：取半径最大的圆（假设最外层是目标）
                valid_circles = []
                for c in circles:
                    # 检查是否在色块中心附近
                    if abs(c.x() - largest_blob.cx()) < 15 and abs(c.y() - largest_blob.cy()) < 15:
                        valid_circles.append(c)

                if valid_circles:
                    target = max(valid_circles, key=lambda c: c.r())  # 找到半径最大的圆
                    x, y, r = target.x(), target.y(), target.r()  # 获取圆心坐标和半径
                    # 绘制检测到的圆环
                    img.draw_circle(x, y, r, color=(0, 255, 0))  # 绿色圆环

                    # 更新卡尔曼滤波器
                    Z = np.array([x, y, 2*r, 2*r, 0, 0])  # 构造观测值：[x, y, w, h, dx, dy]
                    if color == 'red':
                        state = kf_red.update(Z)  # 更新红色滤波器
                    elif color == 'green':
                        state = kf_green.update(Z)  # 更新绿色滤波器
```

```
            else:
                state = kf_blue.update(Z)  # 更新蓝色滤波器

            # 绘制预测结果
            pred_x = int(state[0])  # 预测的圆心x坐标
            pred_y = int(state[1])  # 预测的圆心y坐标
            pred_r = int(state[2]/2)  # 预测的半径

            # 约束圆心在当前颜色的区域内
            blob_x, blob_y, blob_w, blob_h = largest_blob.rect()  # 获取当前颜色区域的边界
            pred_x = max(blob_x, min(blob_x + blob_w, pred_x))  # 约束x坐标在当前颜色区域内
            pred_y = max(blob_y, min(blob_y + blob_h, pred_y))  # 约束y坐标在当前颜色区域内
            pred_r = min(pred_r, min(blob_w, blob_h) // 2)  # 约束半径不超过当前颜色区域大小

            # 绘制预测的圆心和圆环
            img.draw_cross(pred_x, pred_y, color=rect_color)  # 红色、绿色或蓝色十字标记圆心
            img.draw_circle(pred_x, pred_y, pred_r, color=(255, 255, 0))  # 黄色圆环

    # 打印帧率
    print("FPS:", clock.fps())  # 输出当前帧率
```