AI 视觉套件

用户手册

(技术开发文档)

文档版本：V1.102

发布日期：2021/10/08

免责申明

在法律允许的最大范围内，本手册所描述的产品（含其硬件、软件、固件等）均“按照现状”提供，可能存在瑕疵、错误或故障，勤牛创智不提供任何形式的明示或默示保证，亦不对使用本手册或使用本公司产品导致的任何特殊、偶然或间接的损害进行赔偿。在使用本产品前详细阅读本使用手册及网上发布的相关技术文档并了解相关信息，确保在充分了解产品相关知识的前提下使用本产品。

本产品的使用者有责任确保遵循相关国家的切实可行的法律法规，确保在勤牛创智机械臂的使用中不存在任何重大危险。

版本修订说明

时间	版本号	修订记录
2021.03.30	V1.002 版	使用扩展板连接 AI 视觉与 Mirobot
2021.04.21	V1.101 版	套件结构变化，更改组装文档，优化例程
2021.10.08	V1.102 版	完善程序调试细节

北京勤牛创智科技有限公司

地址：北京市海淀区清华东路16号3号楼中关村能源与安全科技园1603室

网址：cn.wlkata.com

目录

1. 产品简介

AI 视觉套件（以下简称“视觉套件”）是 Mirobot 机械臂的重要配件。视觉套件可以很好的配合机械臂完成颜色识别分拣，图像识别处理等工作，有了它的帮助，可以让机械臂变得更加“智能”，更容易处理复杂任务。

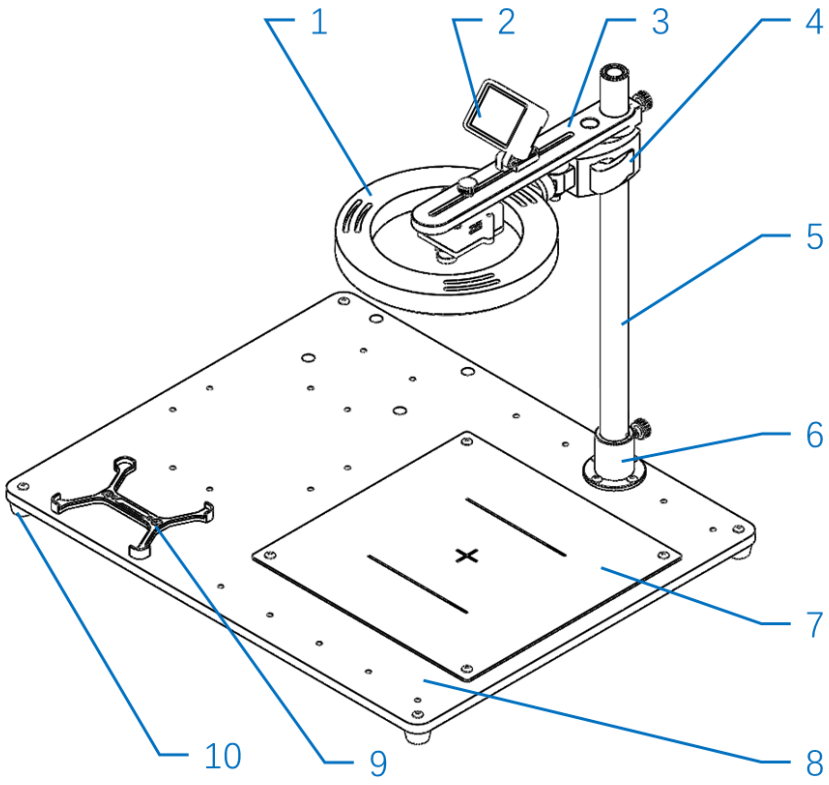


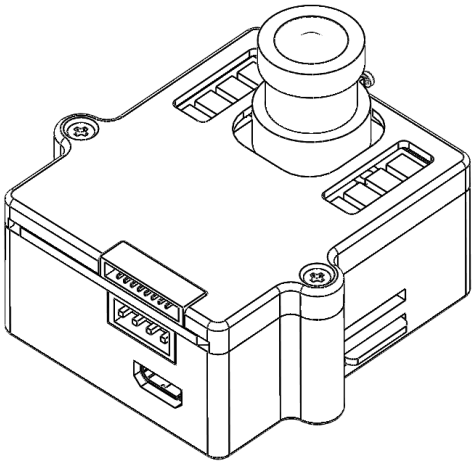
图 1：AI 视觉套件

1.补光灯　2.监视屏幕　3.支架横梁　4.补光灯支架　5.立柱　6.立柱连接件

7.标定板　8.底板　9.控制器支架　10.脚垫

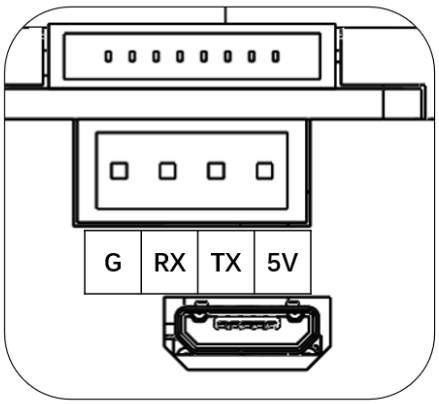
2. AI 视觉套件组件简介

- **AI 视觉模块**



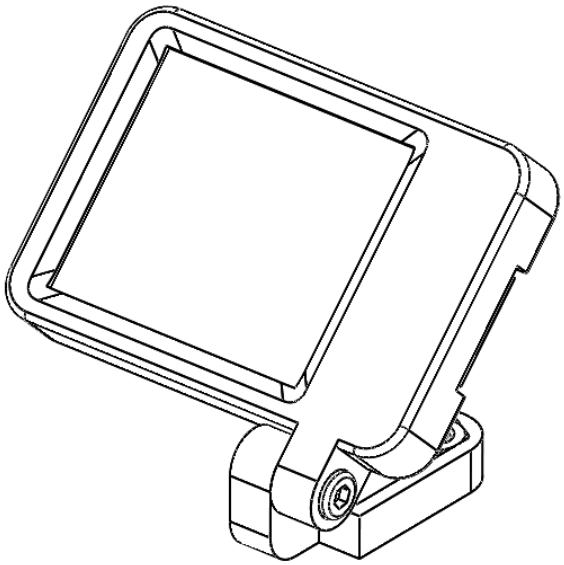
功能：获取图像信息，实现寻找色块、人脸检测、眼球跟踪、边缘检测、标志跟踪等功能，可以用来实现非法入侵检测、产品的残次品筛选、跟踪固定的标记物等。使用者仅需要写一些简单的 Python 代码，即可轻松的完成各种机器视觉相关的任务。

接口：串口（XH2.54-4），LCD 接口，MicroUSB 接口



配件：TF 卡；无畸变镜头

- **监视屏幕**



屏幕类型：1.8”TFT　LCD

水平分辨率：128 像素（28.03mm）0.18mm 像素间距

垂直分辨率：160 像素（35.04mm）0.18mm 像素间距

显示颜色：64K 16-bit RGB565

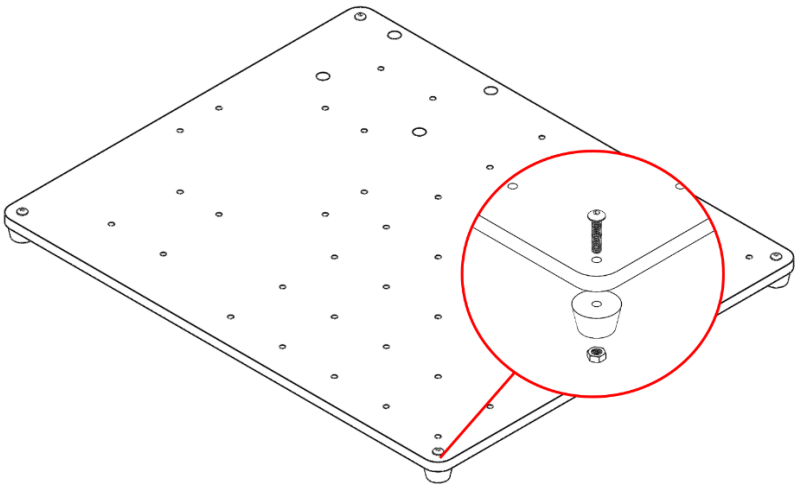
3. AI 视觉套件快速入门

3.1 结构组装

- **安装立柱**

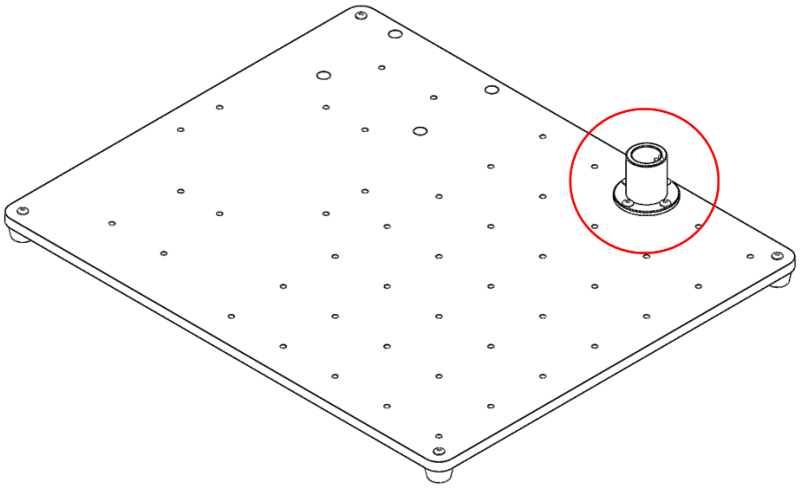
（1）安装脚垫

将底板取出，揭去底板表面保护膜，**磨砂面朝上**（避免因亚克力底板反光，对颜色识别产生干扰），使用 M4-18 圆头内六角螺栓、螺母将脚垫固定在底板四角；



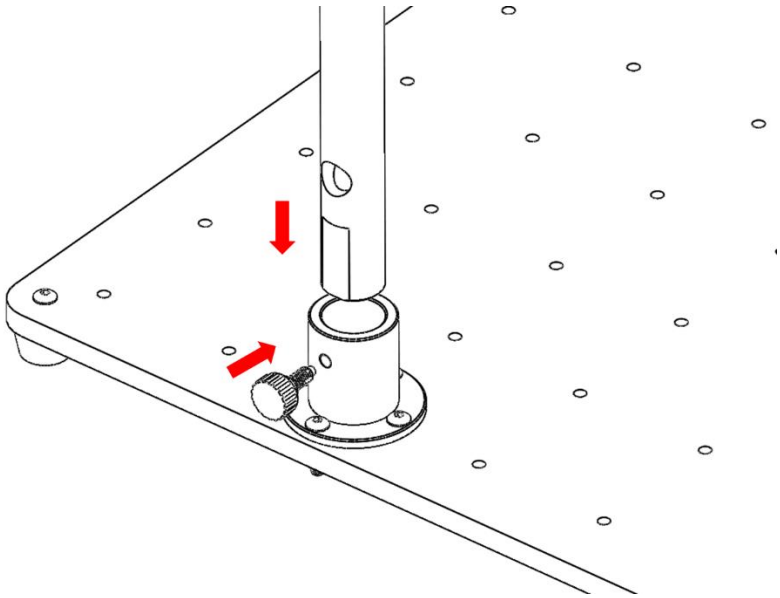
（2）安装立柱连接件（法兰）

使用 M4-18 圆头内六角螺栓、螺母将立柱连接件（法兰）固定在图示位置（注意：法兰侧面紧固螺钉螺纹孔朝向底板底边）。



（2）安装立柱

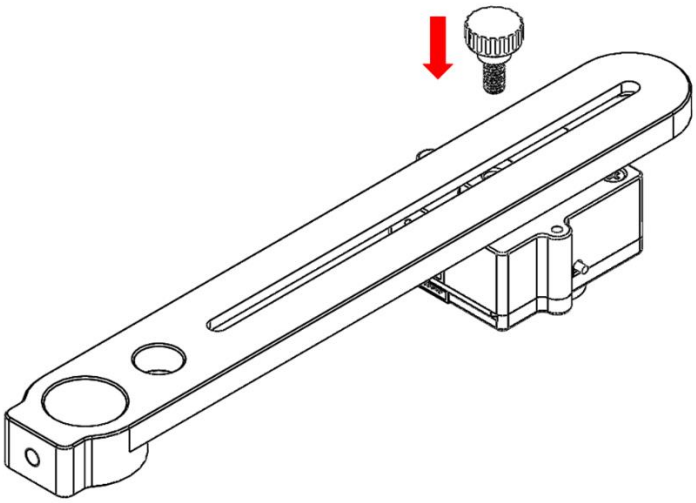
将立柱竖直插入立柱连接件，并使用 M5-12 尼龙缓冲螺钉顶紧，（注意：立柱侧边开孔一侧朝下，立柱为空心圆管，此孔用于串口线穿出）



- **安装 AI 视觉模块、屏幕、补光灯**

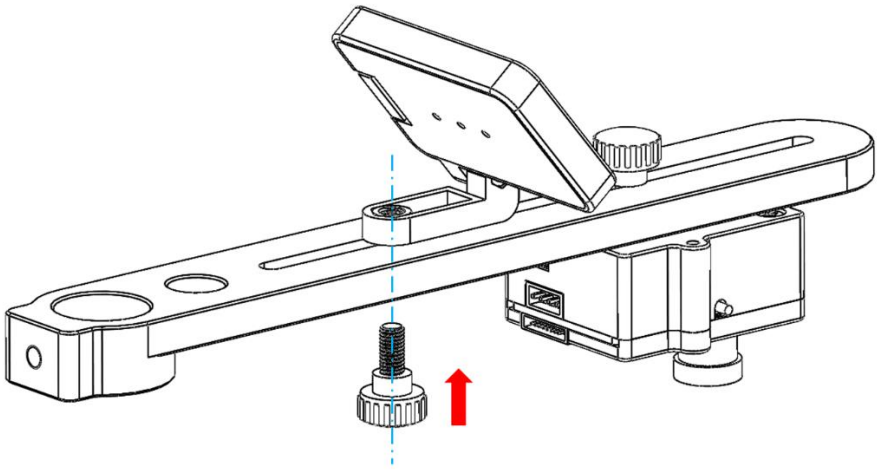
（1）固定 AI 视觉模块

支架如图方向摆放，将 AI 视觉模块卡入支架的槽口内，注意摆放方向，排线插座一侧朝向支架圆孔，确定位置后用 M5-10 手拧螺钉固定。



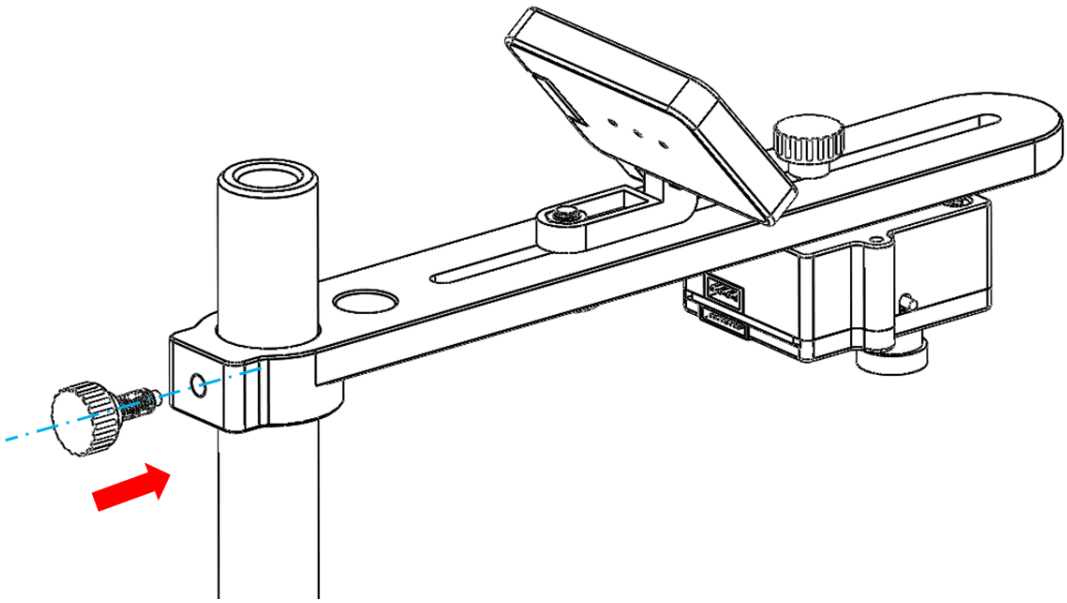
（2）固定监视屏幕

屏幕按图示位置安装，用 M5-10 手拧螺钉固定。



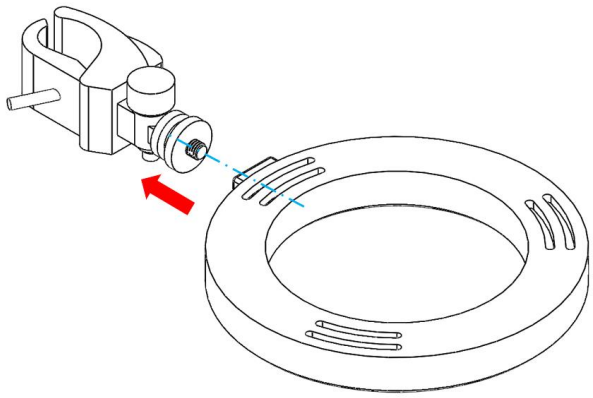
（3）安装支架

支架的圆孔与立柱相配合，并使用 M5-12 尼龙缓冲螺钉固定。



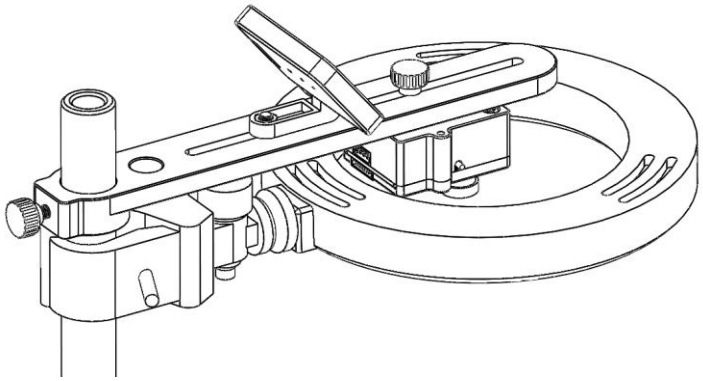
（4）组装补光灯支架

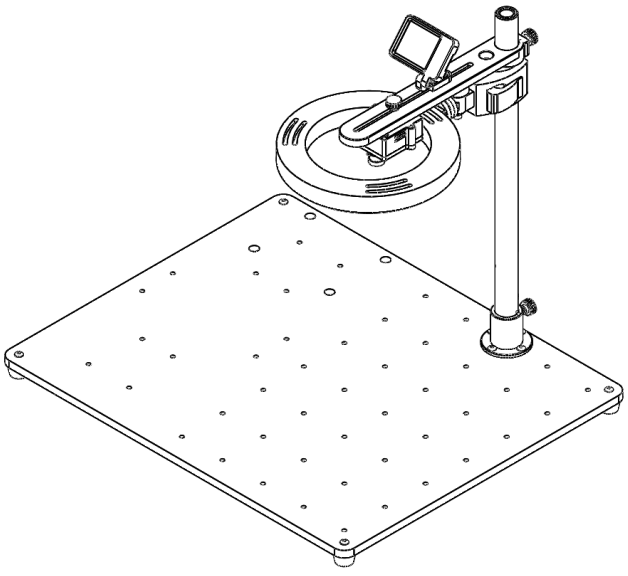
将补光灯与支架连接。



（3）固定补光灯（选装）

将补光灯靠近支架横梁安装，使用快拆螺丝固定。





视觉套件基本结构组装完成，其余配件根据需求选择安装。

3.2 安装 OpenMV IDE

● 软件下载

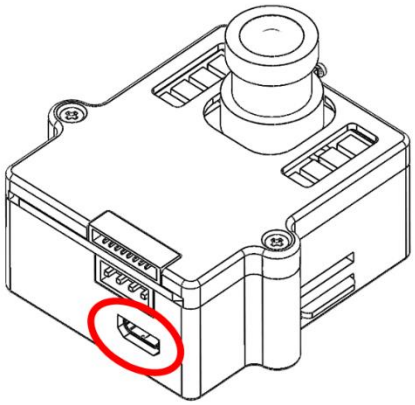
https://singtown.com/openmv-download/

● 软件安装

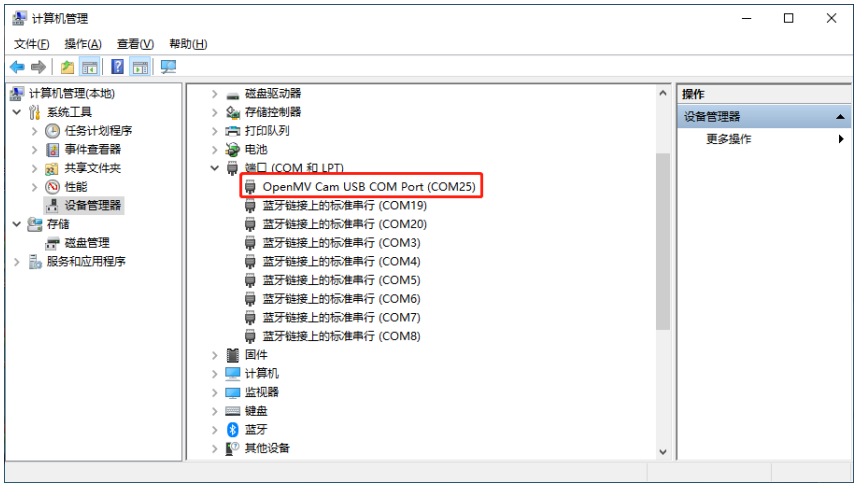
根据提示进行安装；

● 设备连接

（1）将 Micro-USB 数据线插接 AI 视觉模块（下图红圈位置）并连接到计算机 USB 接口；

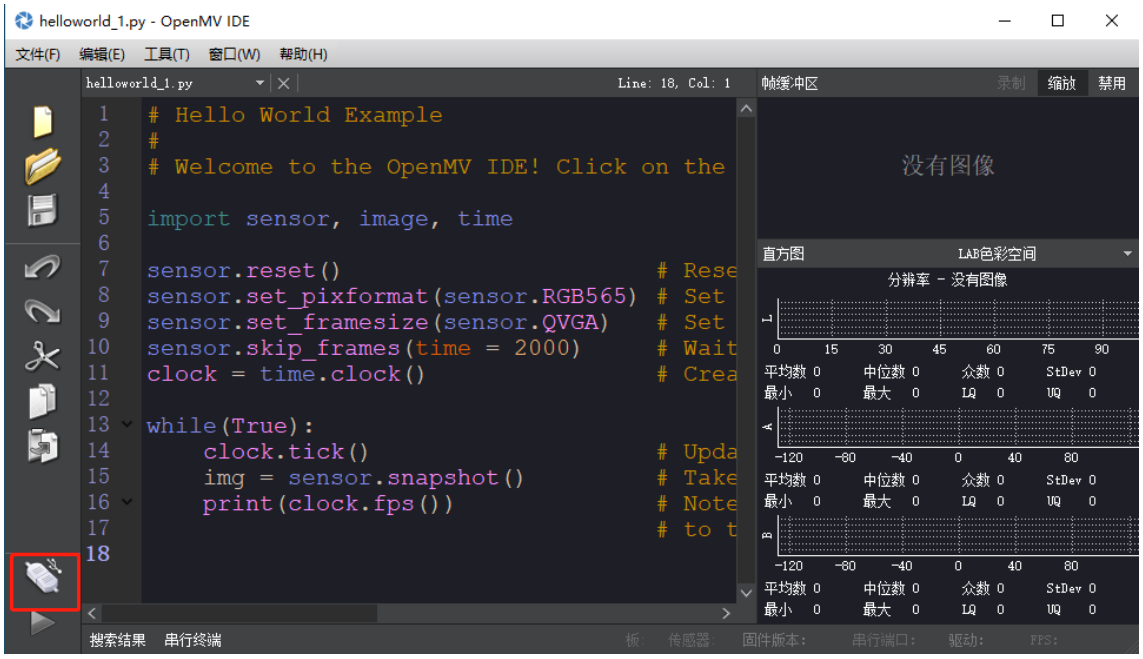


（2）打开设备管理器查看是否识别到视觉模块；

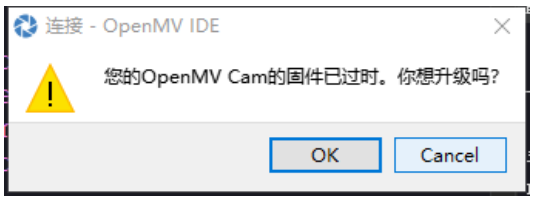


（3）双击 OpenMV IDE 图标打开软件；

（4）点击“连接”按钮，连接设备；

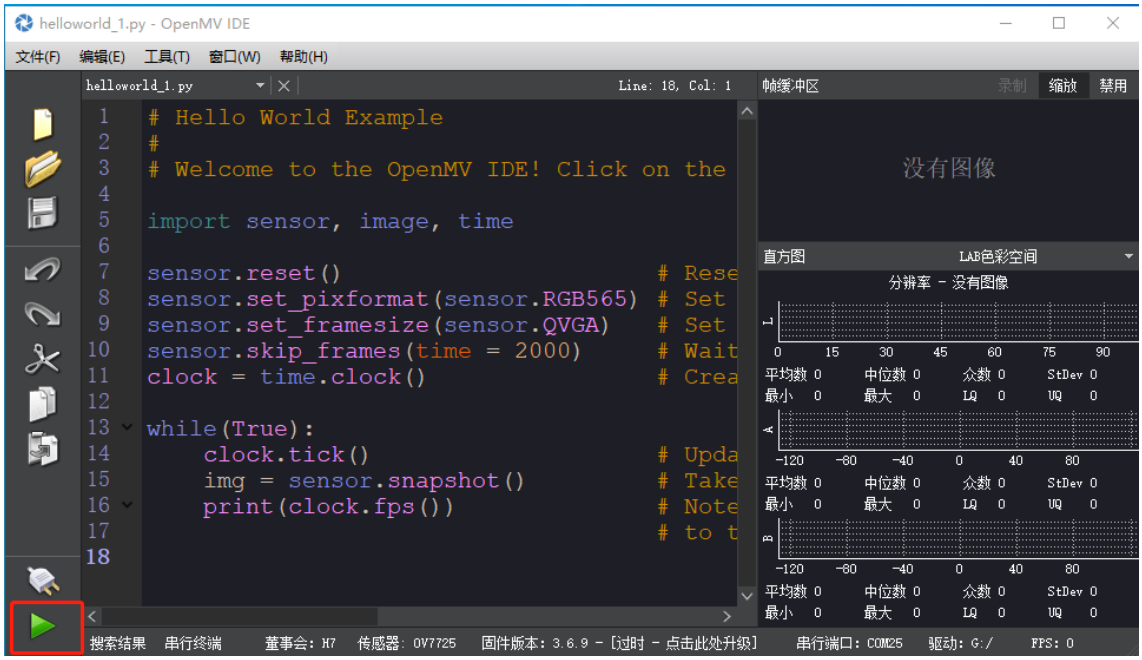


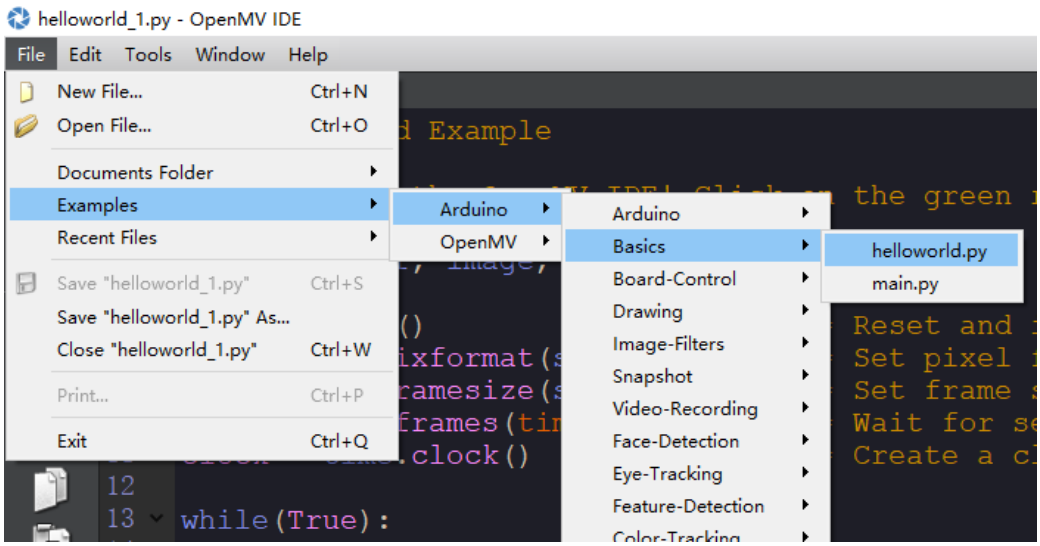
（5）弹出的对话框选择“取消”升级（**固件升级后设备不可用**）；



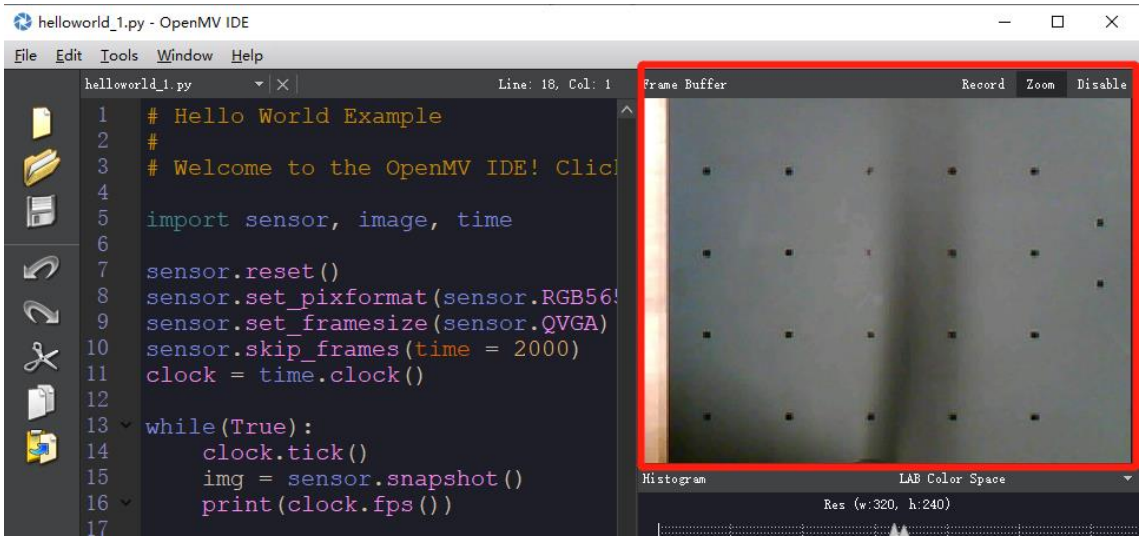
● 运行测试程序

（1）点击右下角“开始”按钮运行当前“helloworld_1.py”，若当前界面非此程序，请按下图所示路径打开“helloworld_1.py”；

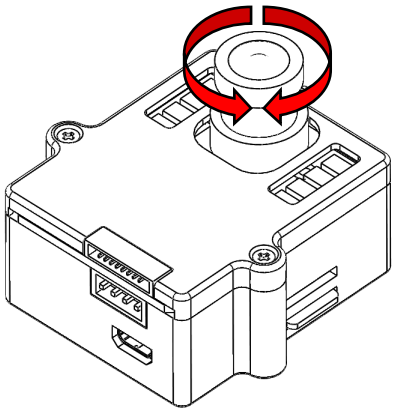




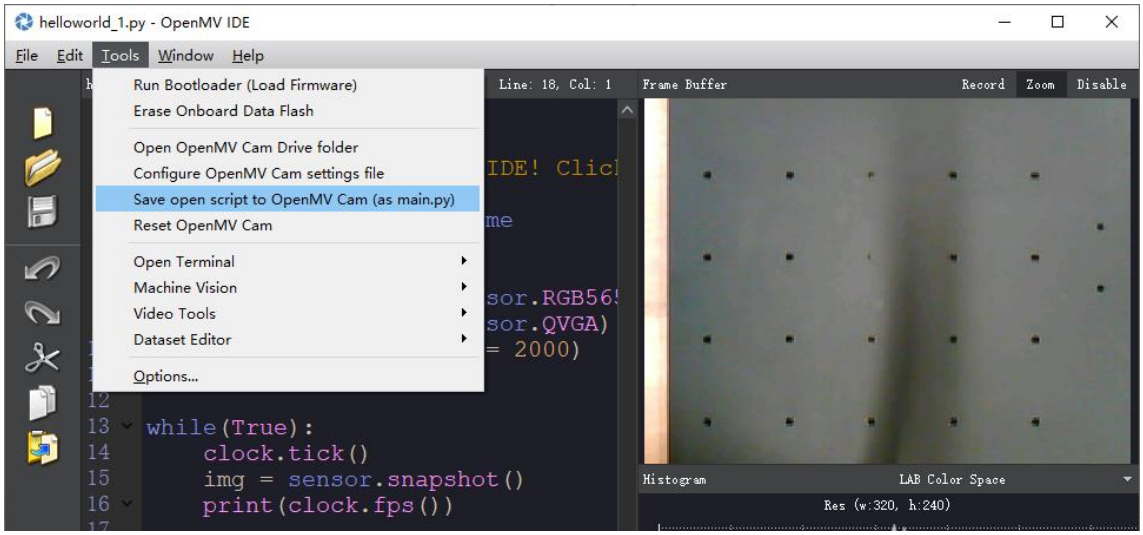
（2）开始运行后，即可在界面右上方的帧缓冲区看到图像。



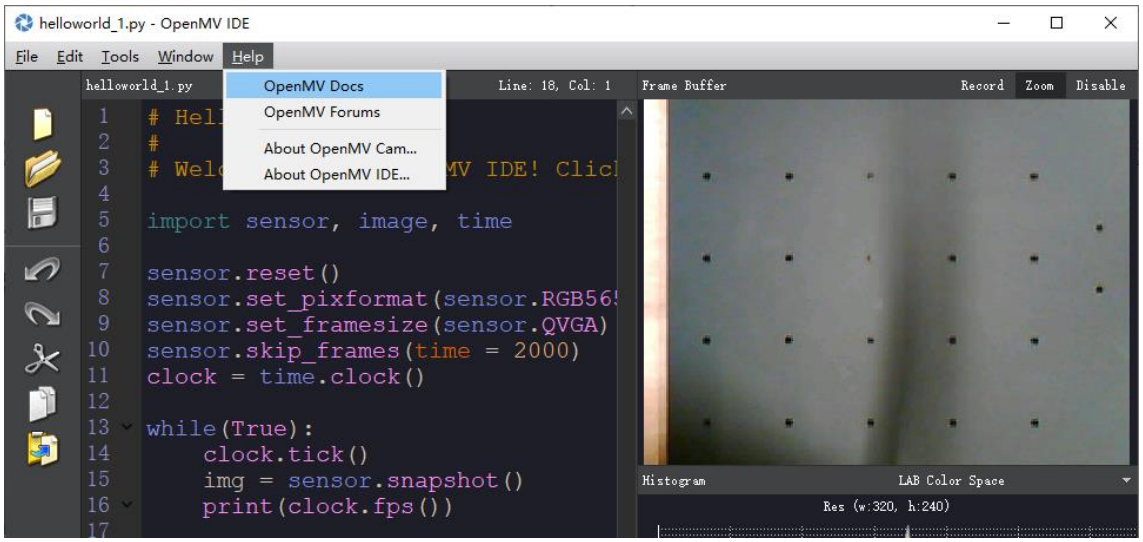
如果图像不清晰，请手动旋转镜头，调整镜头的位置。



（8）如需离线运行，可将程序保存至视觉模块；



（9）OpenMV IDE 资料可参考相关文档



3.3 示例（一）

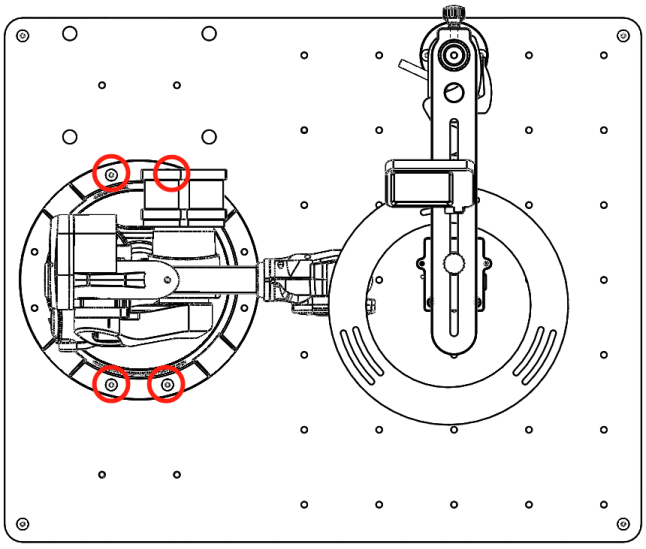
本示例讲解视觉模块的按钮、板载补光灯与监视屏幕的使用；

● 结构组装

视觉模块板载补光灯需要通过串口接口供电，因此本案例将使用机械臂控制器为视觉模块供电。

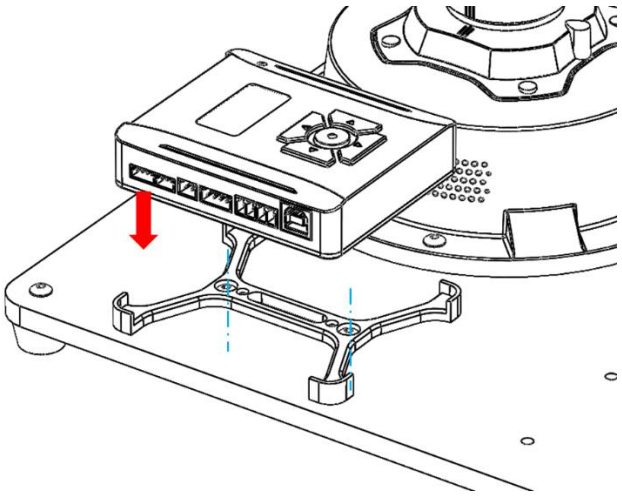
（1）固定机械臂

在图示位置使用 M4-18 圆头内六角螺栓、螺母，将机械臂固定至底板；



（2）安放机械臂控制器

将控制器支架使用 M4-18 圆头内六角螺栓、螺母固定至底板，并将控制器放置其上；



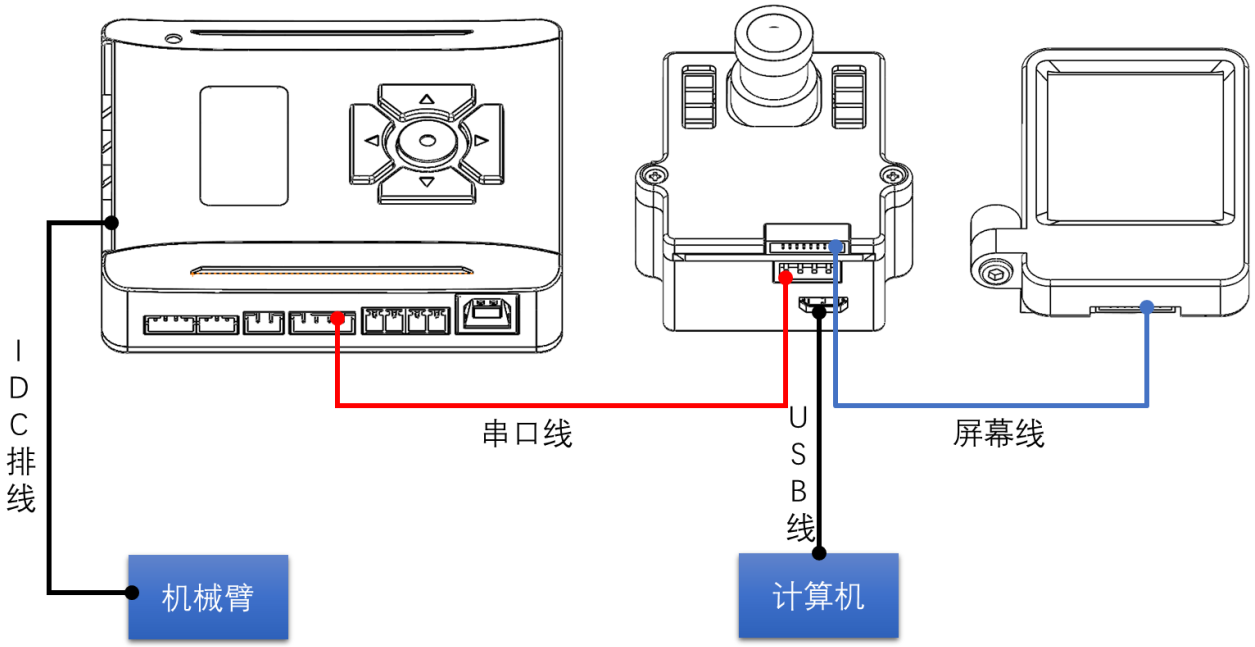
● 硬件连接

（1）控制器使用“IDC 排线”连接至机械臂底座；

（2）使用“串口线”将视觉模块连接至多功能控制器；

（3）监视屏幕连接至视觉模块；

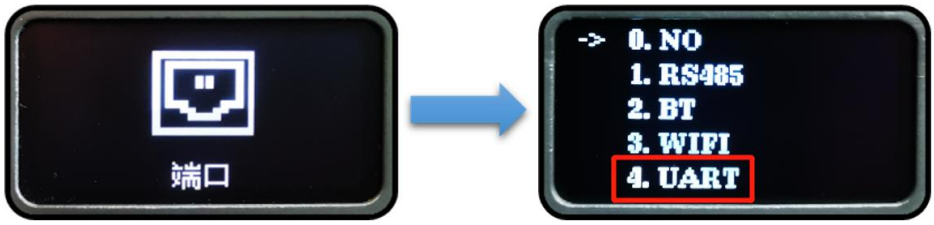
（4）视觉模块使用“micro-USB 数据线”连接至计算机；



● 控制器设置

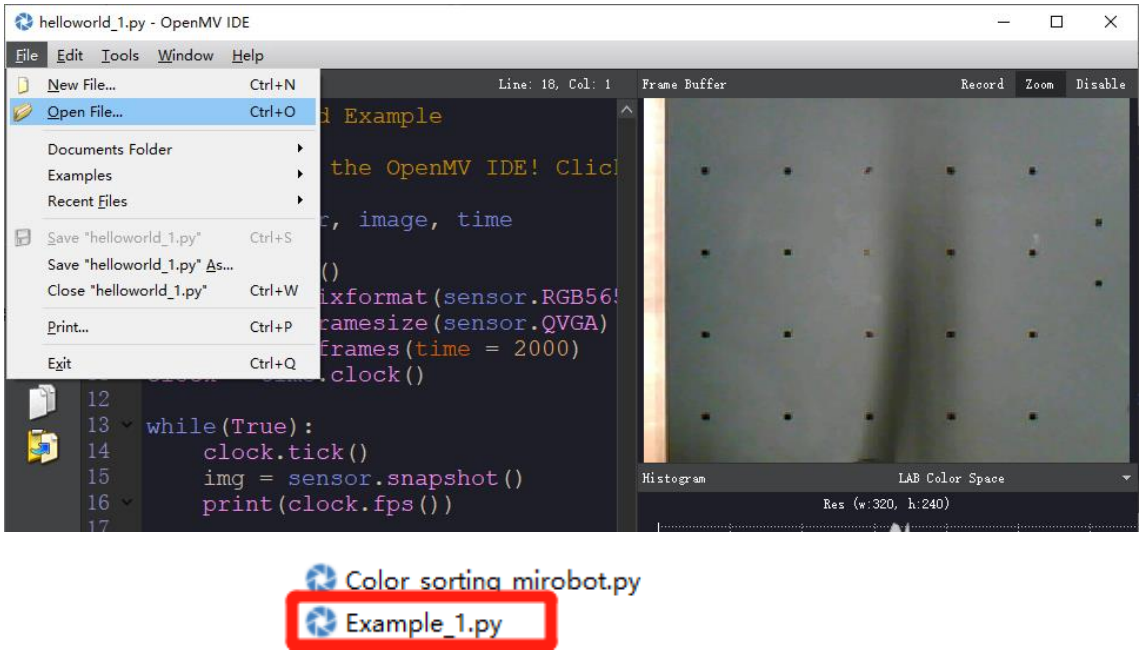
（1）设置端口为“UART”

控制器主界面，按下控制器“↓”按钮，进入“端口”菜单，选择“UART”；

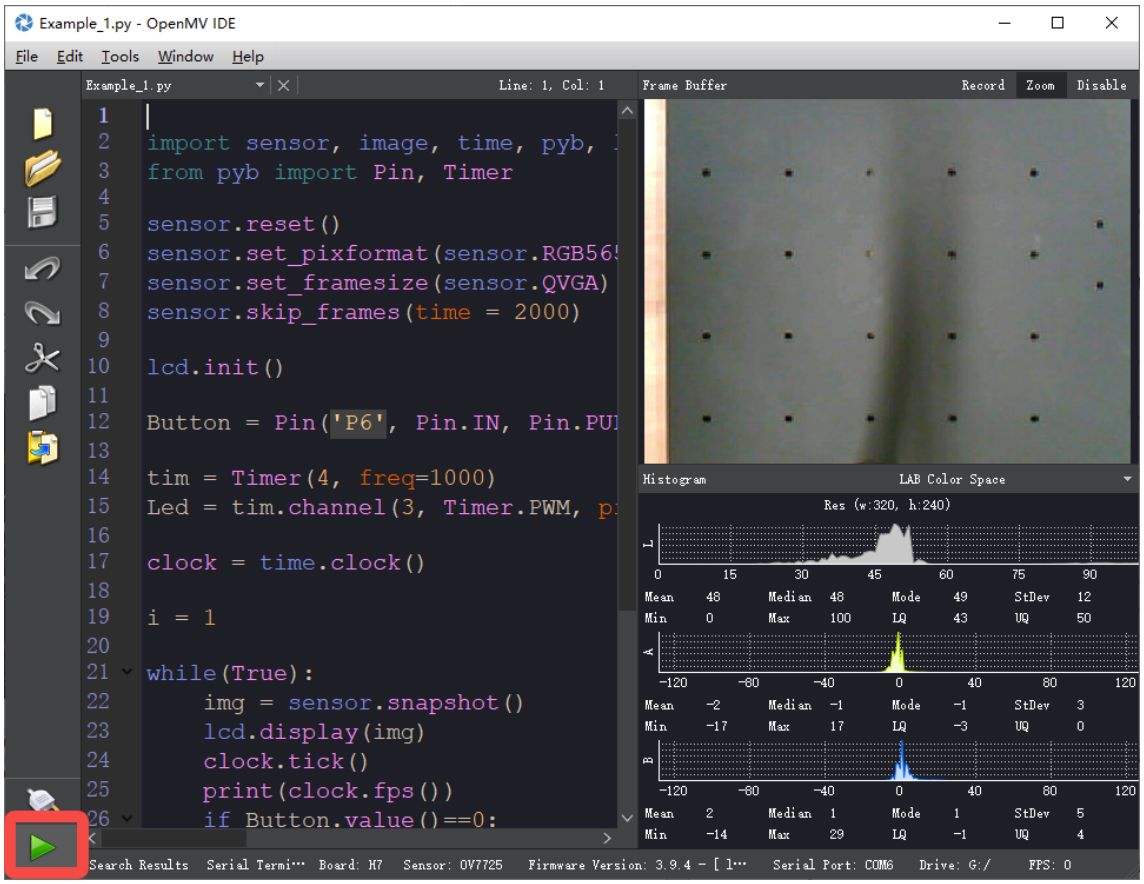


- **运行程序**

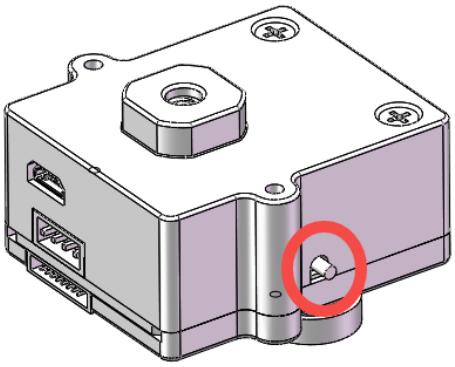
（1）打开文件“Example_1.py”；



（2）点击右下角“开始”按钮运行当前程序；



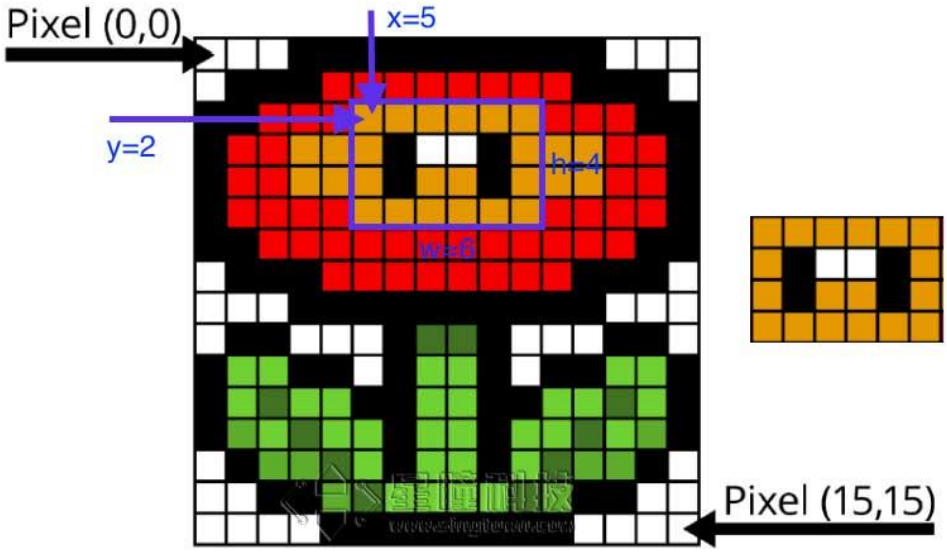
（3）按下模块上的按钮，控制板载补光灯打开或关闭。



（4）由于 LCD 屏的分辨率（128×160）小于当前设置的图像分辨率（QVGA (320x240)），因此程序中指定一区域（lcd_roi = (x, y, w, h)）使 LCD 屏幕居中显示视觉模块捕捉到的图像；

帧缓冲区图像

LCD 屏显示图像



x:ROI 区域中左上角的 x 坐标　　y:ROI 区域中左上角的 y 坐标

w:ROI 的宽度　　h:ROI 的高度

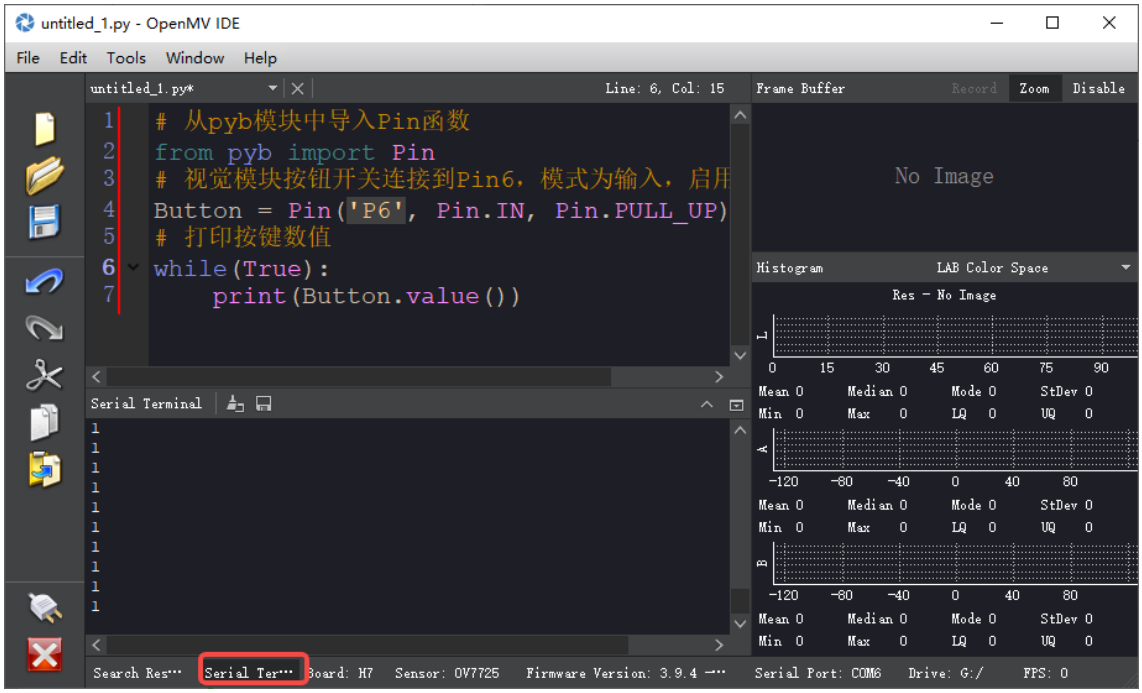
● 代码解析（文件：Example_1.py）

（1）按钮开关控制

这个例子展示了如何使用 OpenMV 的 I/O 引脚，“P6”为板载按钮引脚，将编辑框内的内容全部删掉，换成以下代码运行。

```
# 从 pyb 模块中导入 Pin 函数
from pyb import Pin
# 视觉模块按钮开关连接到Pin6，模式为输入，启用上拉电阻
Button = Pin('P6', Pin.IN, Pin.PULL_UP)
# 打印按键数值
while(True):
    print(Button.value())
```

运行程序后，点击最下方的 Serial Terminal（红框位置），会弹出终端窗口。即可查看返回的按键值，松开时为“1”，按下时为“0”



注意：按钮开关与 LCD 屏幕同时使用时，LCD 代码必须在按钮引脚初始化代码之前，否则程序离线运行时按钮不可用；

```
# 屏幕初始化
lcd.init()

# 设置按键引脚
Key = Pin('P6', Pin.IN, Pin.PULL_UP)
```

（2）补光灯 pwm 控制

这个例子展示了如何使用 OpenMV 的 PWM，“P9”为补光灯引脚，将编辑框内的内容全部删掉，换成以下代码运行。

```
import time
from pyb import Pin, Timer
# 使用定时器 4 创建一个定时器对象- 以 1000Hz 触发
tim = Timer(4, freq=1000)
# 配置 Pin9 的初始脉宽值百分比为 20
Led = tim.channel(3, Timer.PWM, pin=Pin("P9"), pulse_width_percent=20)

while (True):
    time.sleep_ms(1000)
```

补光灯脉宽值百分比范围为 0~100，为防止补光灯过热影响寿命，建议长时间开启时，百分比小于 60。

(3) LCD 屏使用

这个例子展示了如何使用监视屏幕，将编辑框内的内容全部删掉，换成以下代码运行。

```
import sensor, image

sensor.reset()                          # 初始化摄像头，reset()是sensor 模块里面的函数
sensor.set_pixformat(sensor.RGB565)     # 设置图像色彩格式，有RGB565 色彩图和GRAYSCALE 灰度图两种
sensor.set_framesize(sensor.QVGA)       # 将图像大小设置为QVGA (320x240)
sensor.skip_frames(time = 2000)         # 等待设置生效

lcd.init()                              # 初始化 lcd 屏幕

    x = (sensor.width() - lcd.width()) / 2
    y = (sensor.height() - lcd.height()) / 2
    lcd_roi = (int(x), int(y), lcd.width(), lcd.height())# 设置LCD 屏幕显示区域

    while(True):
        img = sensor.snapshot()             # 拍一张照片并返回图像。
        lcd.display(img,roi = lcd_roi)      # LCD 屏居中显示缓冲区图像
```

(4) 文件：Example_1.py 全部代码

```
import sensor, image, time, pyb, lcd    # 引入此例程依赖的模块，sensor 是与摄像头参数设置相关的模块，image 是图像处理相关的模块
from pyb import Pin, Timer

sensor.reset()                          # 初始化摄像头，reset()是sensor 模块里面的函数
sensor.set_pixformat(sensor.RGB565)     # 设置图像色彩格式，有RGB565 色彩图和GRAYSCALE 灰度图两种
sensor.set_framesize(sensor.QVGA)       # 将图像大小设置为QVGA (320x240)
sensor.skip_frames(time = 2000)         # 等待设置生效

lcd.init()                              # 初始化 lcd 屏幕

    x = (sensor.width() - lcd.width()) / 2
    y = (sensor.height() - lcd.height()) / 2
```

```
13.   lcd_roi = (int(x), int(y), lcd.width(), lcd.height())# 设置 LCD 屏幕显示区域
14.
15.   Button = Pin('P6', Pin.IN, Pin.PULL_UP)  # 视觉模块按钮开关连接到 Pin6，设置开关闭合，引脚置低
16.
17.   tim = Timer(4, freq=1000)                # 使用定时器 4 创建一个定时器对象-以 1000Hz 触发
18.   Led = tim.channel(3, Timer.PWM, pin=Pin("P9"), pulse_width_percent=0)
  #配置 Pin9 的初始脉宽值百分比为 0
19.
20.   clock = time.clock()                     # 创建一个时钟对象
21.
22.   i = 1                                    # 赋值变量
23.
24.   while(True):                             # python while 循环，一定不要忘记加冒号“：”
25.       clock.tick()                         # 更新 FPS 帧率时钟
26.       img = sensor.snapshot()              # 拍一张照片并返回图像。
27.       print(clock.fps())                   # 打印当前的帧率
28.       lcd.display(img,roi = lcd_roi)       # LCD 屏居中显示缓冲区图像
29.
30.       if Button.value()==0:                # 判断按钮按下
31.           while Button.value()==0:         # 等待按钮松开
32.               pass
33.           if i:
34.               Led.pulse_width_percent(50)  # 设置 LED 引脚 PWM 占空比 50%，打开板载补光灯，范围（0-100）
35.               i=0
36.           else:
37.               Led.pulse_width_percent(0)   # 设置 LED 引脚 PWM 占空比 0，关闭板载补光灯
38.               i=1
```

3.4 示例（二）——颜色分拣

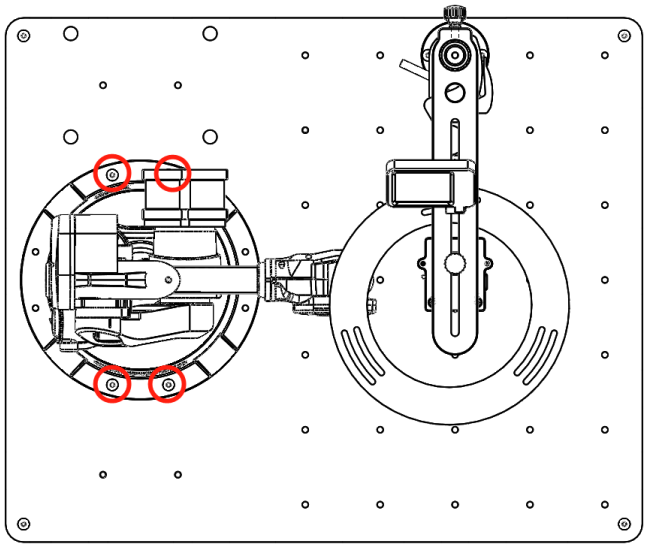
该示例将通过 AI 视觉模块识别彩色物块，提炼出颜色及位置信息，并将识别信息转换为 G 代码，控制机械臂抓取木块，实现木块的颜色分拣。

● 结构组装

视觉模块板载补光灯需要通过串口接口供电，因此本案例将使用机械臂控制器为视觉模块供电。

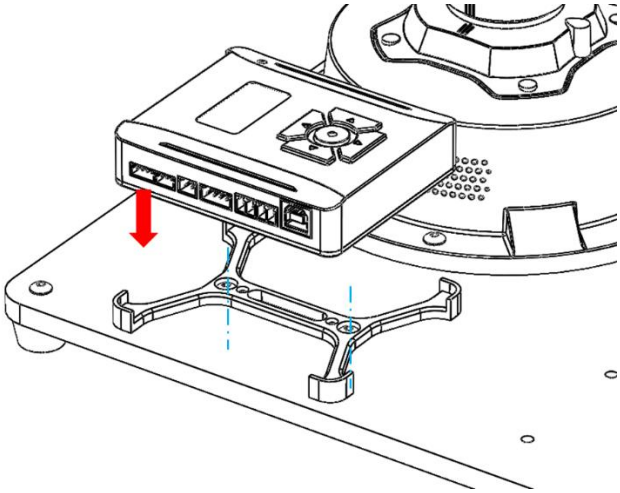
（1）固定机械臂

在图示位置使用 M4-18 圆头内六角螺栓、螺母，将机械臂固定至底板；



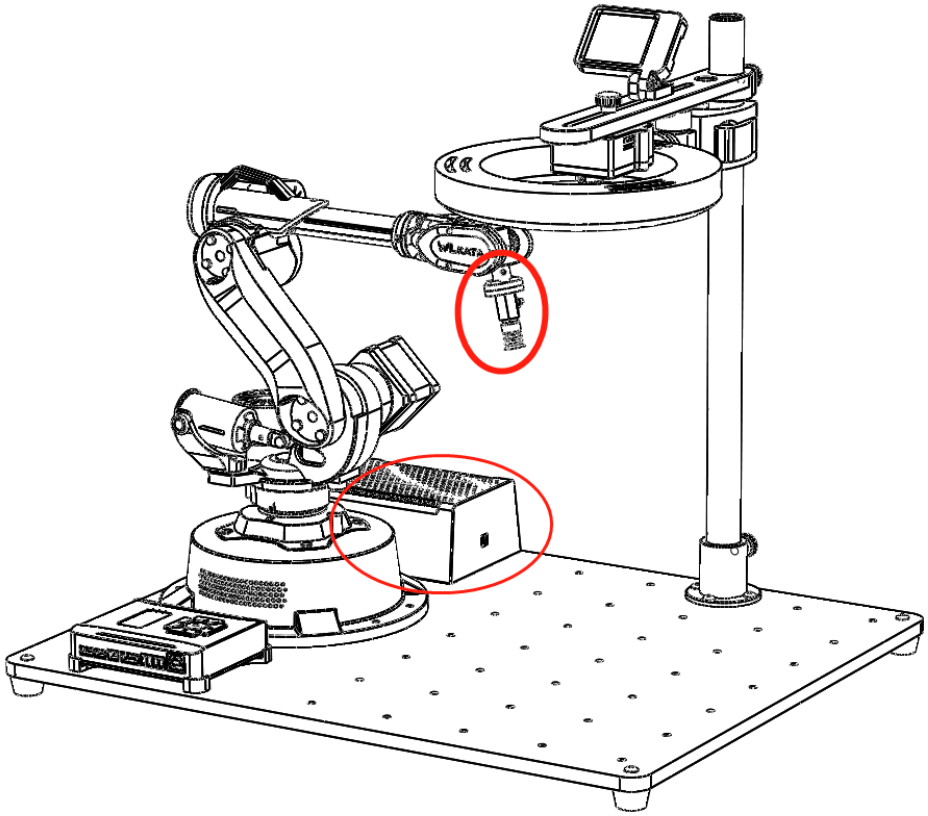
（2）安放机械臂控制器

将控制器支架使用 M4-18 圆头内六角螺栓、螺母固定至底板，并将控制器放置其上；



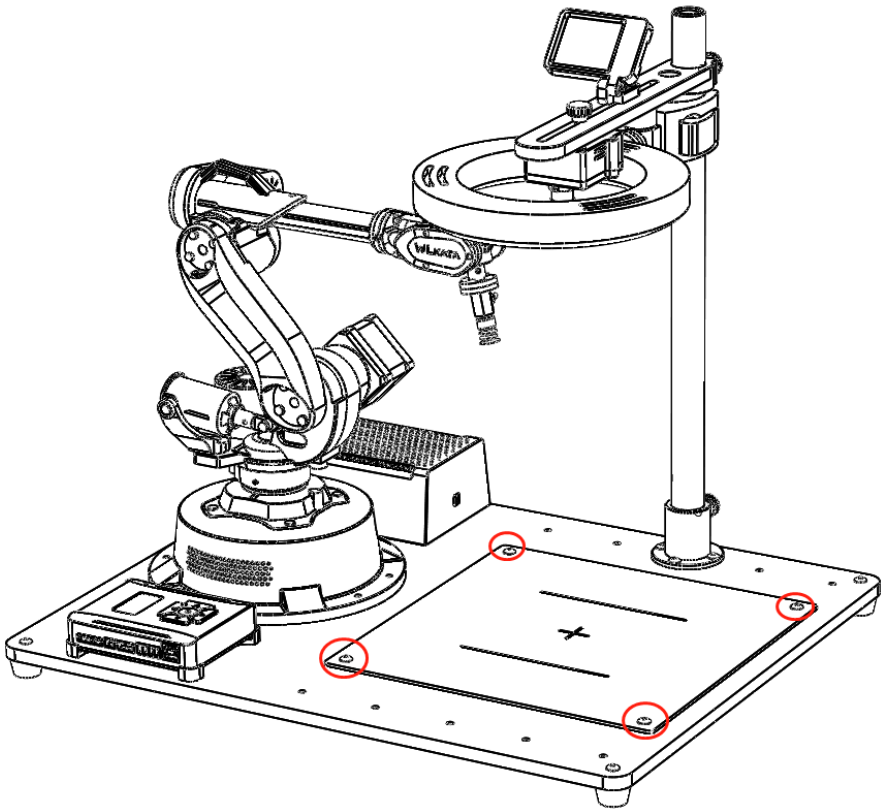
（3）安装气动工具

气泵布置于机械臂一侧，将机械臂末端工具更换为单指吸盘，并连接气管；



（4）固定标定板

标定板使用使用 M4-18 圆头内六角螺栓、螺母固定至底板；



●硬件连接

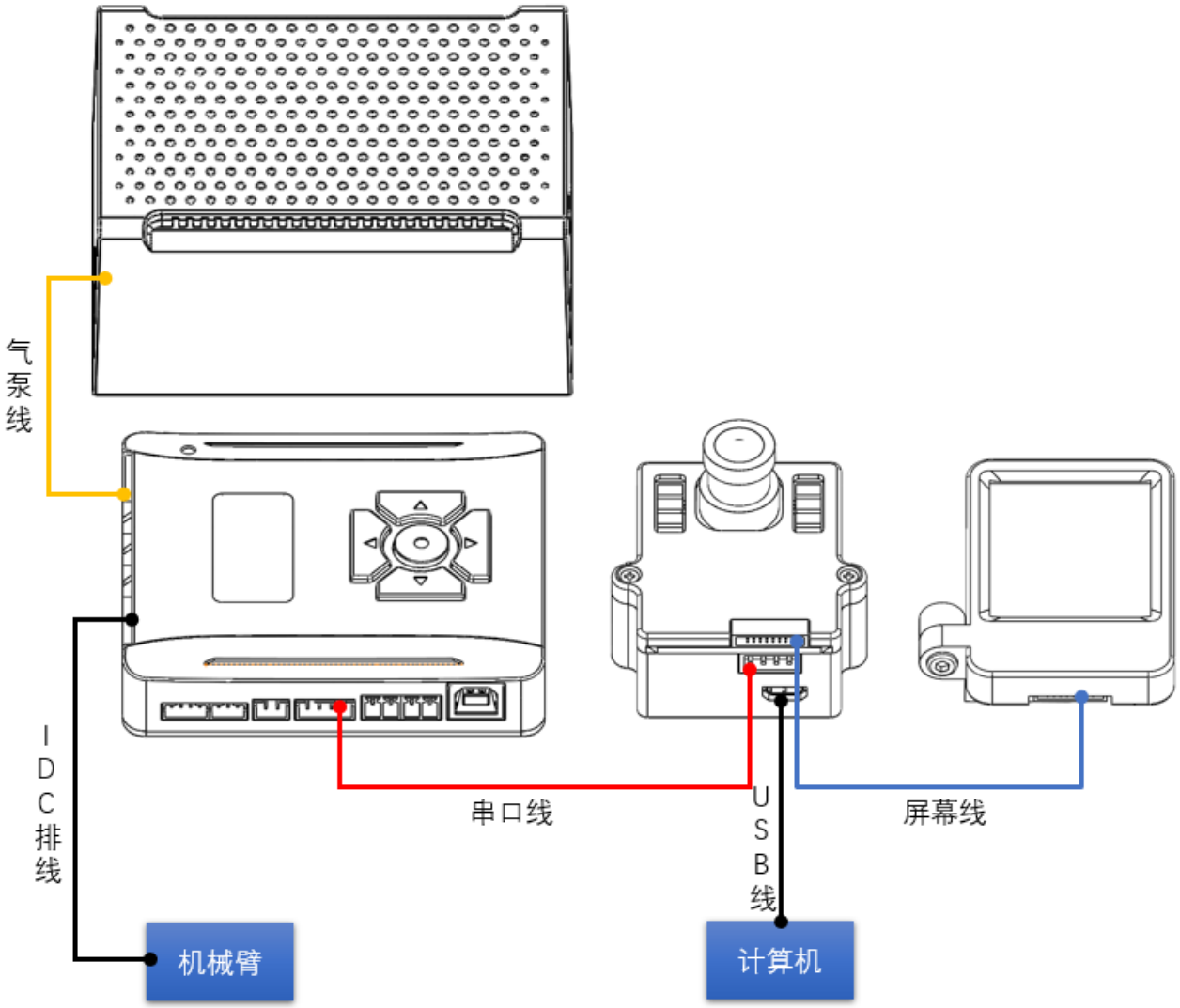
（1）控制器使用“IDC 排线”连接至机械臂底座；

（2）使用“串口线”将视觉模块连接至多功能控制器；

（3）监视屏幕连接至视觉模块；

（4）视觉模块使用“micro-USB 数据线”连接至计算机；

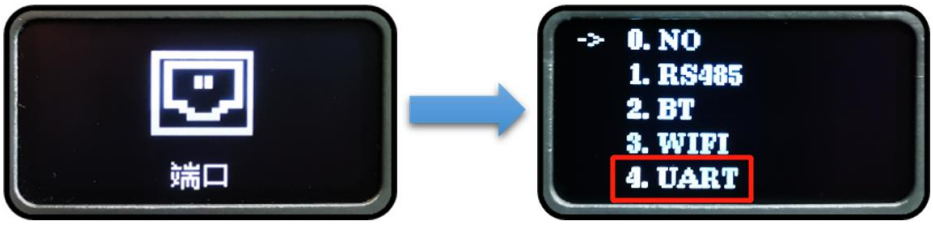
（5）气泵使用自带的“气泵线”连接至控制器黄色 PWM 接口



● 控制器设置

（1）设置端口为“UART”

控制器主界面，按下控制器“↓”按钮，进入“端口”菜单，选择“UART”；



● 更换镜头

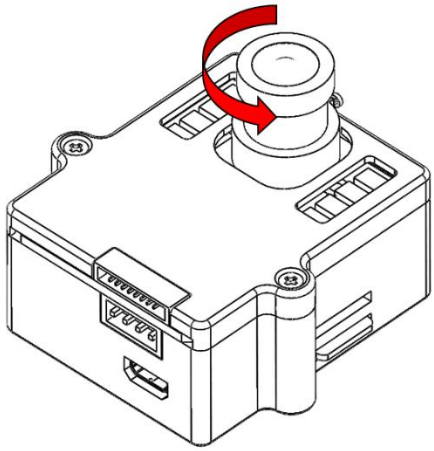
视觉模块标配标准镜头，因为光学原理，在感光芯片上不同的位置，与镜头的距离不同的，简单说就是近大远小，所以在边缘会出现鱼眼效果（桶型畸变）。为了解决这个问题，可以在代码中使用算法来矫正畸变，注：OpenMV 中使用 image.lens_corr(1.8)来矫正 2.8mm 焦距的镜头。也可以直接使用无畸变镜头。无畸变镜头加入了额外的矫正透镜部分，以下为同一高度下两种镜头的图像对比。

标配镜头

无畸变镜头

（1）取下标准镜头

将镜头逆时针旋下，收存好，注意避免触摸镜头及滤片，以免造成污损；



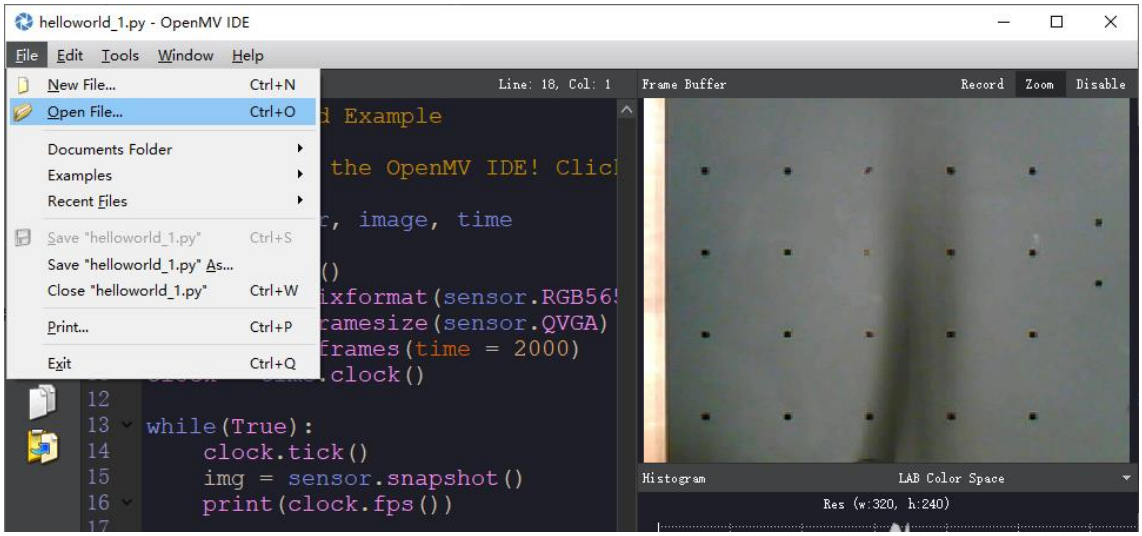
（2）安装无畸变镜头

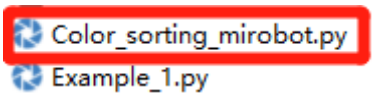
从套装中取出无畸变镜头（如下图所示）安装至镜头座，；

- **运行程序**

（1）加载程序

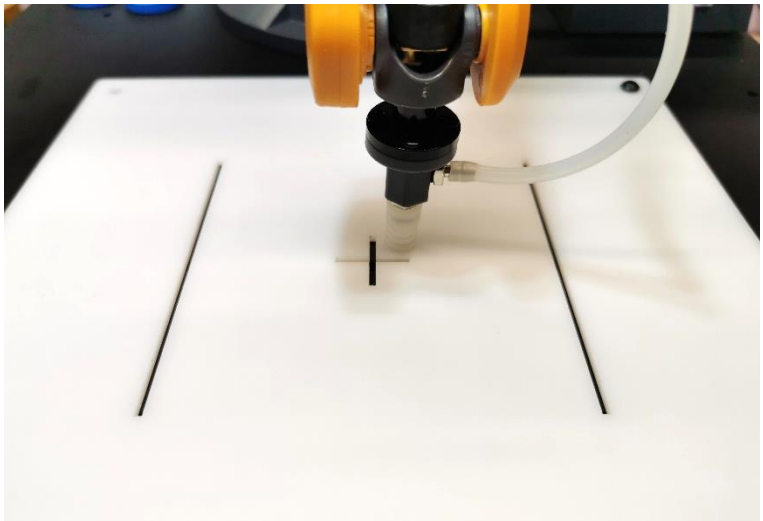
打开文件“Color_sorting_mirobot.py”；





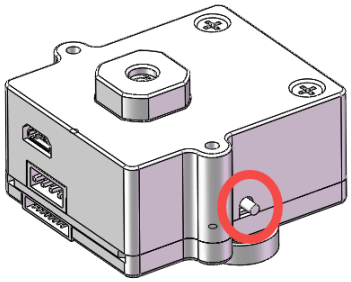
（2）校准坐标

机械臂自身未校准时，末端绝对位置可能存在误差，当机械臂与标定板固定完成时，理论上标定板中心十字位置为机械臂坐标（X210，Y0）位置，当实际可能存在偏差，如下图所示，此时可以通过调整机械臂“$150~$156 参数”进行校准。

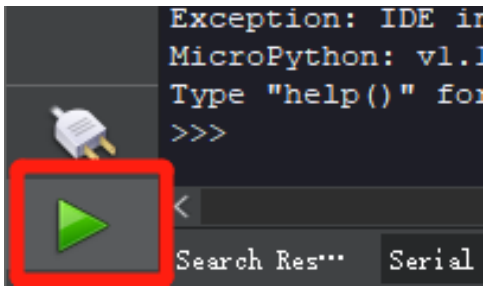


本程序也提供了另一种校准方式，使得机械臂在进行颜色分拣时忽略自身位置误差的影响，详细步骤如下：

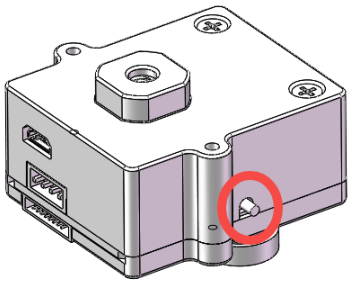
①按住视觉模块侧面的按钮不松开；



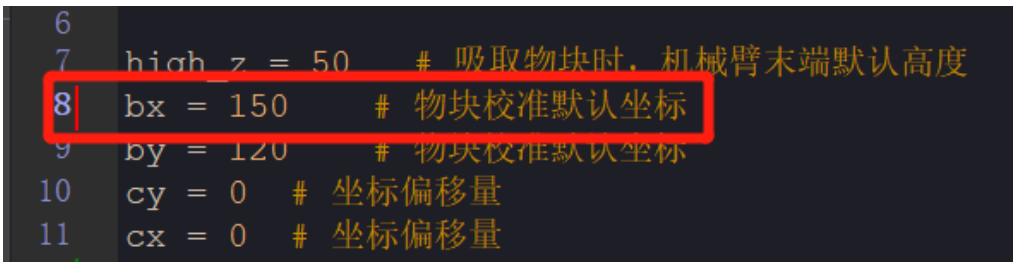
②按下绿色开始键，运行“Color_sorting_mirobot.py”程序；

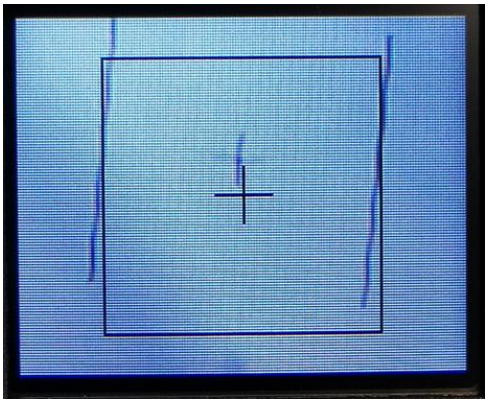


③等待机械臂运动后，松开按钮；

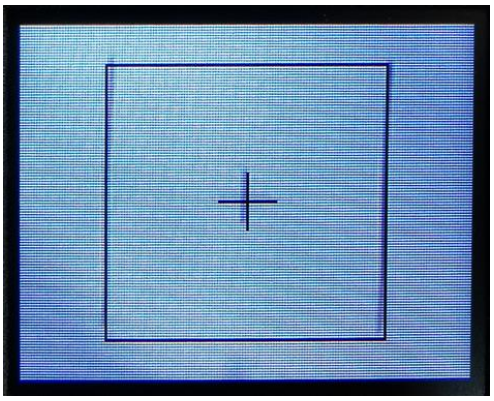


④机械臂复位完成，向一侧移开后，调整视觉模块在支架上的位置，使 LCD 屏幕上的图案与标定板的图案重合，如果图案只能平行，不能重合，请修改下图 bx 数值，并重复之前步骤；





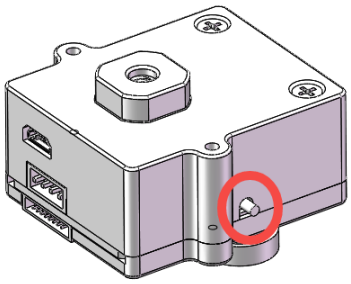
校准前



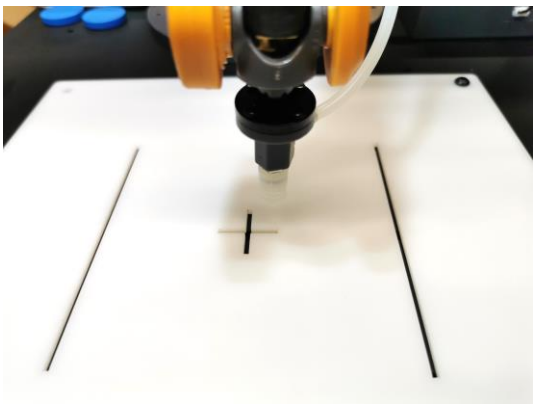
校准后

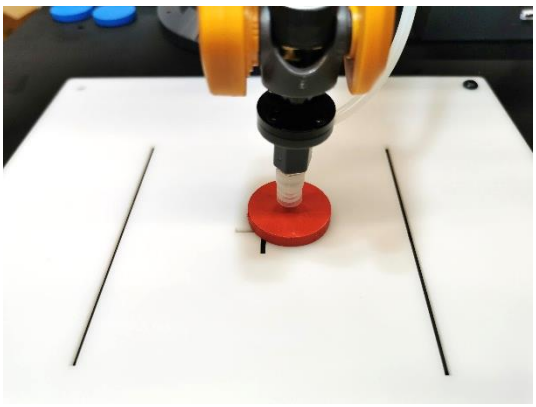
图：LCD 监视屏幕图像

⑤图案重合后，短按一下按钮；

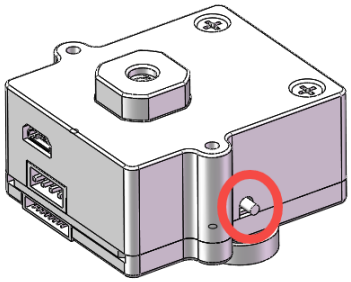


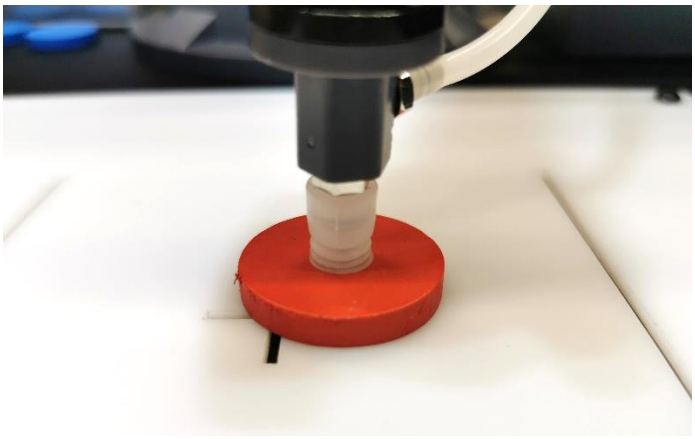
⑥机械臂运动至下图位置，取出一个红色木片，放置于吸盘正下方；





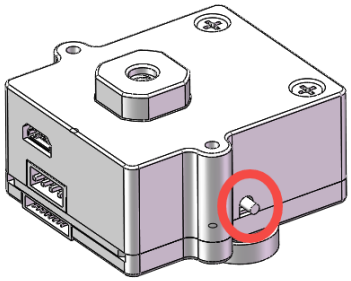
⑦短按按钮，每短按一次，吸盘高度降低 1mm，直至吸盘完全接触物块。





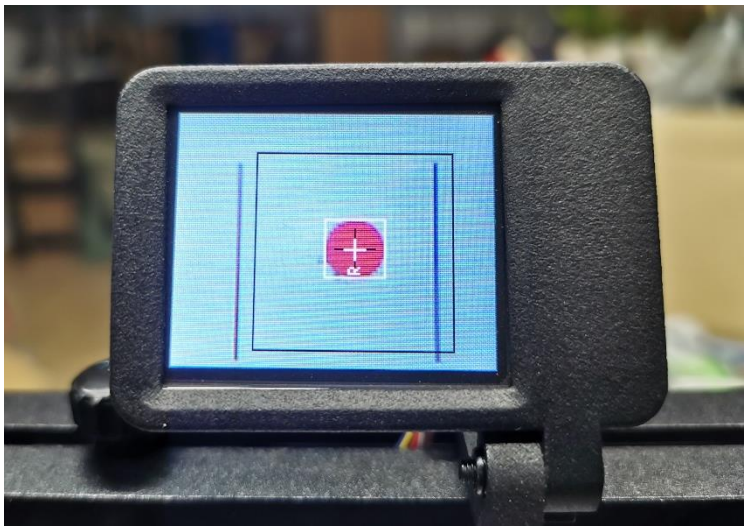
连续短按，每次下降 1mm　　目标状态

⑧长按按钮 **3s** 并松开，保存位置，完成吸盘到物块之间距离的校准。



（3）自动分拣

将木块放置在标定板上（以十字为中心边长 120 的矩形区域是有效识别区域），机械臂即可自动抓取分拣。



● **程序调试**

本程序主要演示 OpenMV 的“寻找色块”功能，这也是 OpenMV 用的最多的功能，通过 find_blobs 函数可以找到色块，下面讨论一下，find_blobs 的细节。

（1）find_blobs 函数

```
image.find_blobs(thresholds, roi=Auto, x_stride=2, y_stride=1, invert=False, area_threshold=10, pixels_threshold=10, merge=False, margin=0, threshold_cb=None, merge_cb=None)
```

这里的参数比较多。

- thresholds 是颜色的阈值，注意：这个参数是一个列表，可以包含多个颜色。如果你只需要一个颜色，那么在这个列表中只需要有一个颜色值，如果你想要多个颜色阈值，那这个列表就需要多个颜色阈值。注意：在返回的色块对象 blob 可以调用 code 方法，来判断是什么颜色的色块。

```
red = (xxx,xxx,xxx,xxx,xxx,xxx)
blue = (xxx,xxx,xxx,xxx,xxx,xxx)
yellow = (xxx,xxx,xxx,xxx,xxx,xxx)

img=sensor.snapshot()
red_blobs = img.find_blobs([red])

color_blobs = img.find_blobs([red,blue, yellow])
```

- roi 是“感兴趣区”。在使用统计信息中已经介绍过了。

  left_roi = [0,0,160,240]

  blobs = img.find_blobs([red],roi=left_roi)

- x_stride 就是查找的色块的 x 方向上最小宽度的像素，默认为 2，如果你只想查找宽度 10 个像素以上的色块，那么就设置这个参数为 10：

  blobs = img.find_blobs([red],x_stride=10)

- y_stride 就是查找的色块的 y 方向上最小宽度的像素，默认为 1，如果你只想查找宽度 5 个像素以上的色块，那么就设置这个参数为 5：

  blobs = img.find_blobs([red],y_stride=5)

- invert 反转阈值，把阈值以外的颜色作为阈值进行查找
- area_threshold 面积阈值，如果色块被框起来的面积小于这个值，会被过滤掉
- pixels_threshold 像素个数阈值，如果色块像素数量小于这个值，会被过滤掉
- merge 合并，如果设置为 True，那么合并所有重叠的 blob 为一个。

  注意：这会合并所有的 blob，无论是什么颜色的。如果你想混淆多种颜色的 blob，只需要分别调用不同颜色阈值的 find_blobs。

（2）阈值

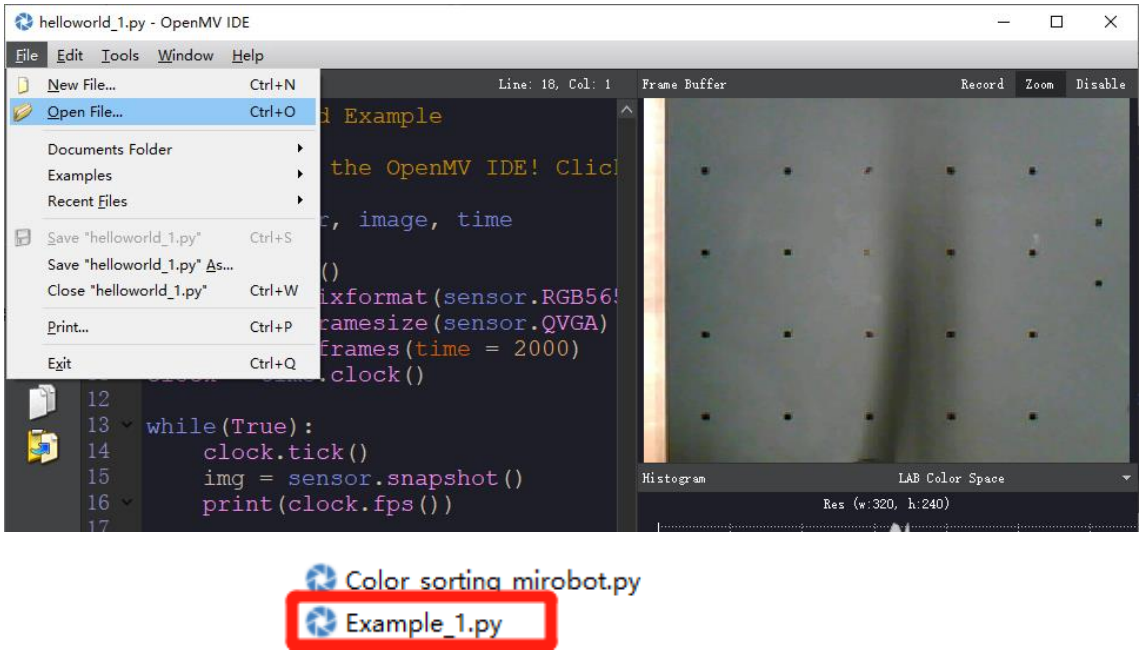
thresholds 是颜色的阈值，一个颜色阈值的结构是这样的：

```
1. red = (minL, maxL, minA, maxA, minB, maxB)
```

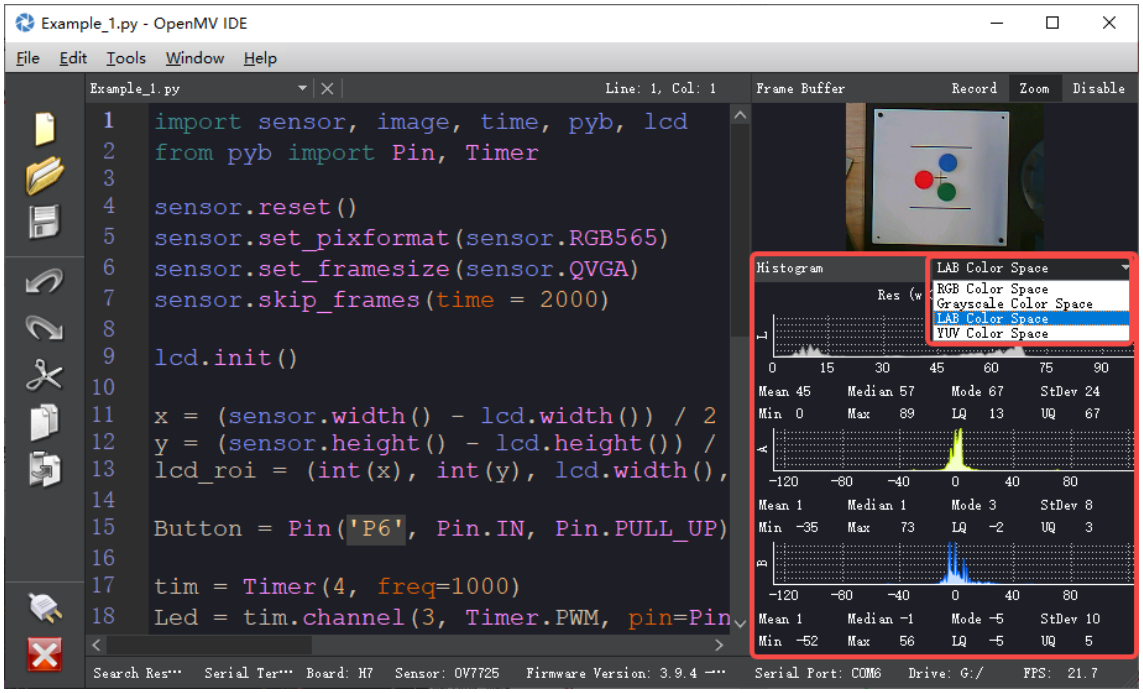
元组里面的数值分别是 L A B 的最大值和最小值。Lab 颜色空间中，L 亮度；a 的正数代表红色，负端代表绿色；b 的正数代表黄色，负端代表兰色。不像 RGB 和 CMYK 色彩空间，Lab 颜色被设计来接近人类视觉。

在 IDE 的图像里获取这个阈值方法如下：

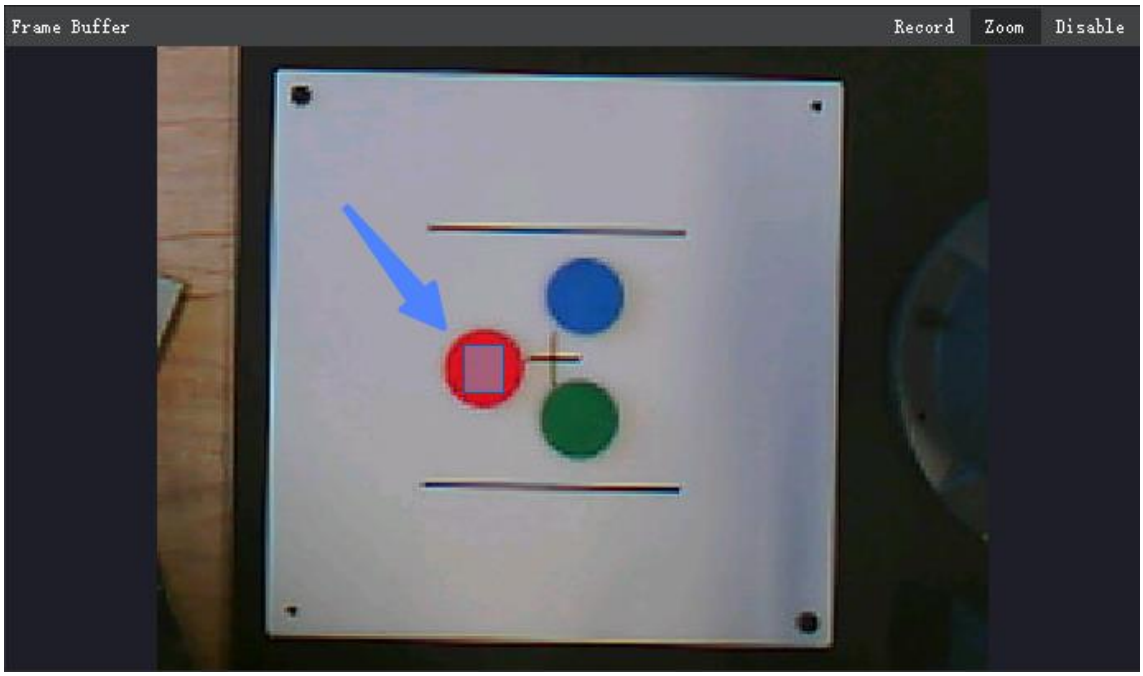
①运行“Example_1.py”



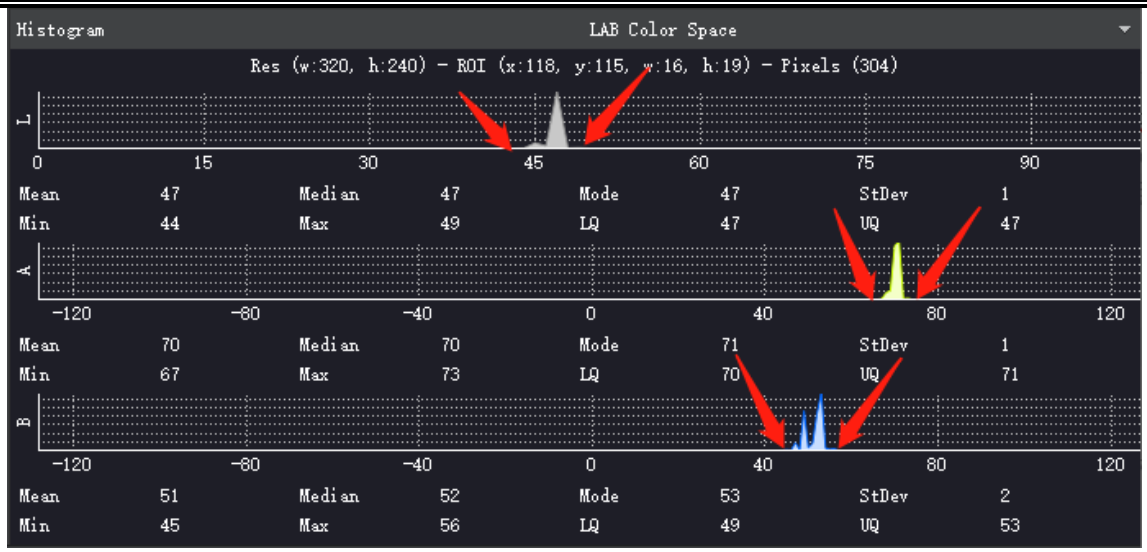
②在 framebuffer 下面的坐标图中，选择 LAB Color Space。



③在 framebuffer 中的目标颜色上左击圈出一个矩形；



④三个坐标图分别表示圈出的矩形区域内的颜色的 LAB 值，选取三个坐标图的最大最小值，即(40, 50, 65, 75, 45, 60)



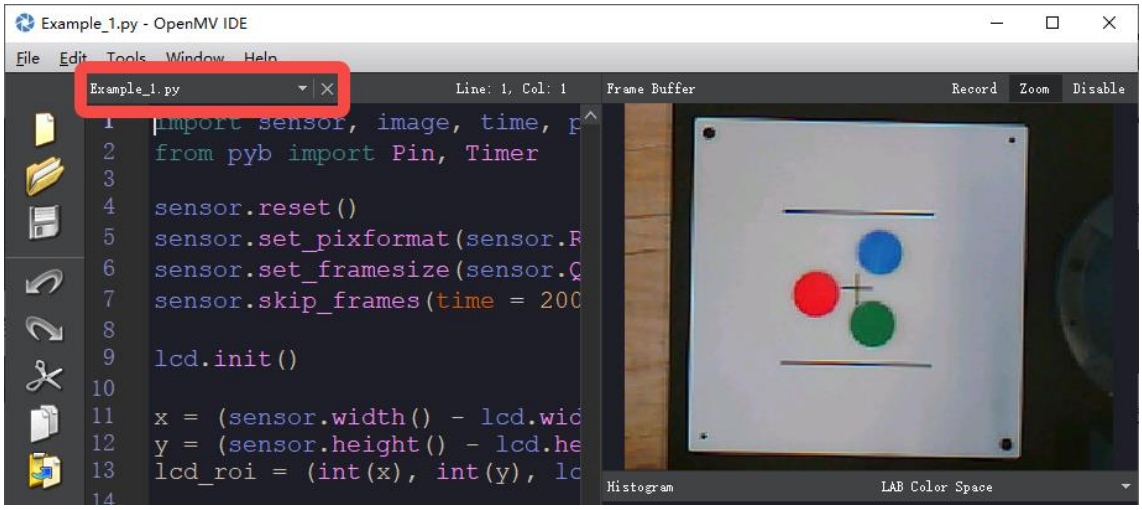
（3）颜色阈值选择工具

Color_sorting_mirobot.py 示例中，机械臂分拣的木块颜色为红、绿、蓝，颜色阈值见下图：

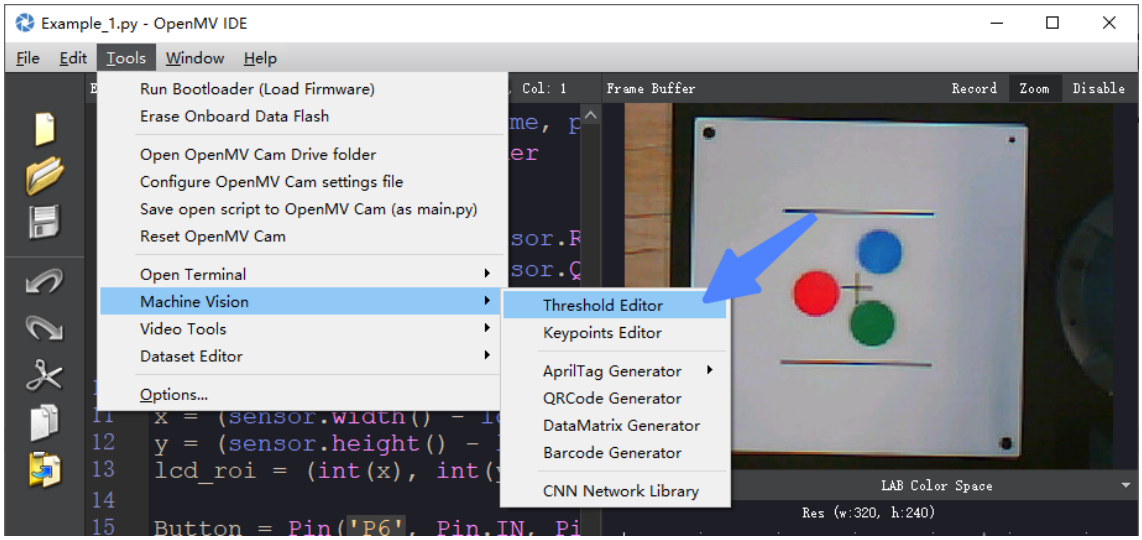
```
# 设置颜色阈值
thresholds = [(0, 100, 25, 127, -128, 127),     #红色
              (0, 100, -128, -18, 11, 127),     #绿色
              (1, 100, -128, 127, -128, -20)]   #蓝色
```

如果程序不能正确分辨出三种颜色，则需使用阈值选择工具修正木块的颜色阈值，操作方法如下：

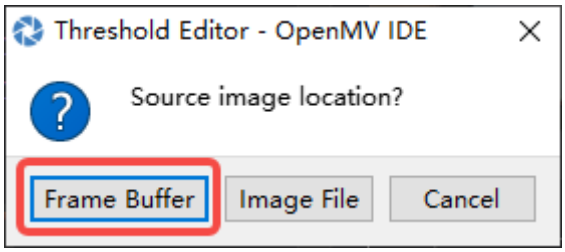
①首先运行 “Example_1.py” 让 IDE 里的 framebuffer 显示图案，按下视觉模块的按钮打开补光灯。



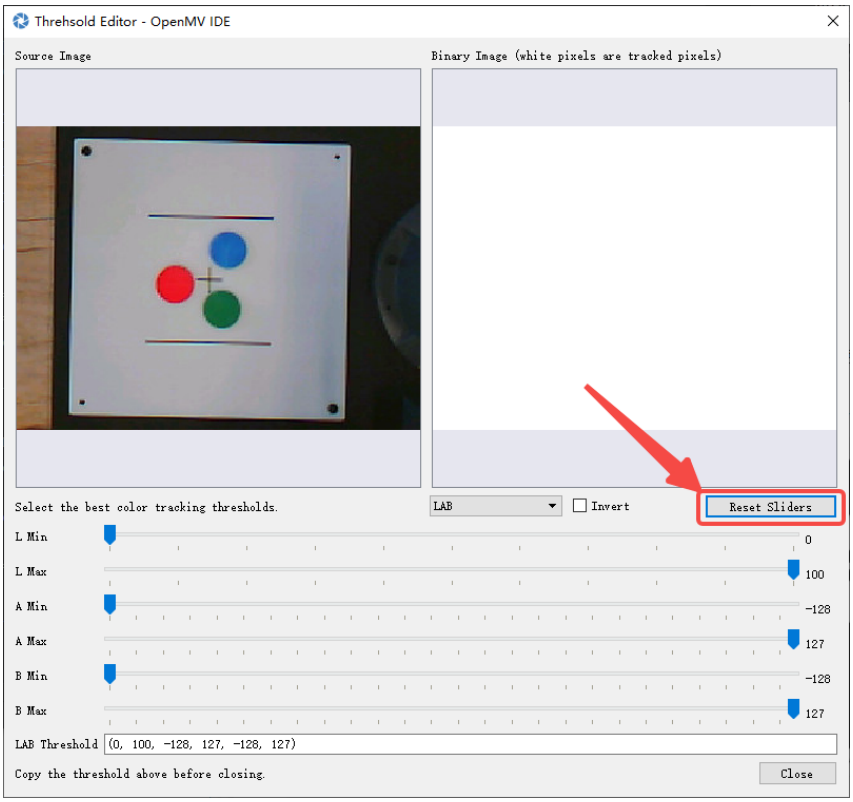
②打开 工具 → Mechine Vision → Threshold Editor



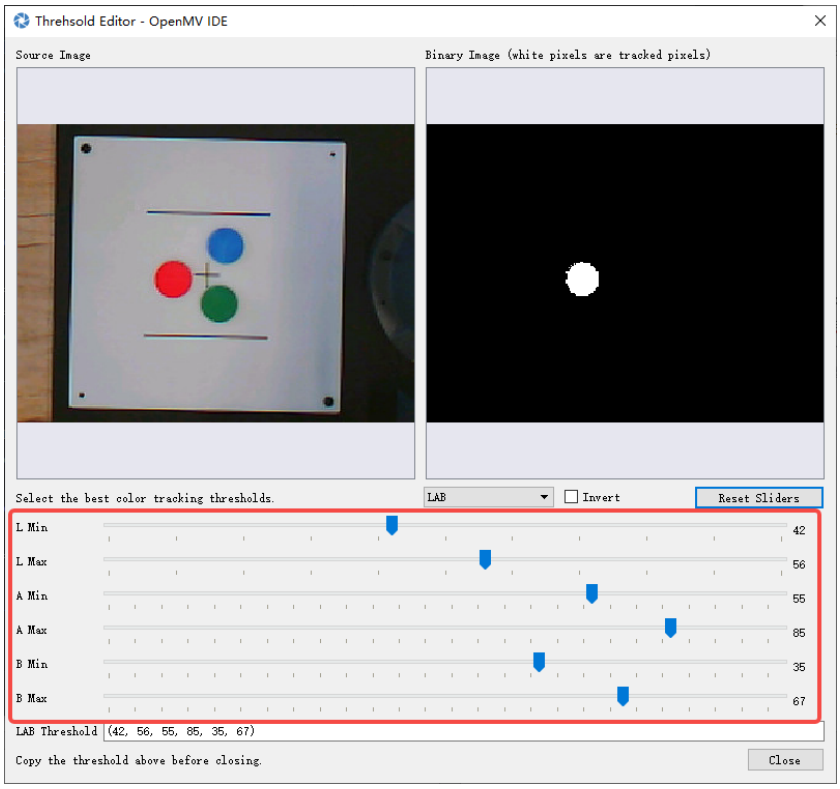
③点击 Frame Buffer 可以获取 IDE 中的图像，Image File 可以自己选择一个图像文件。

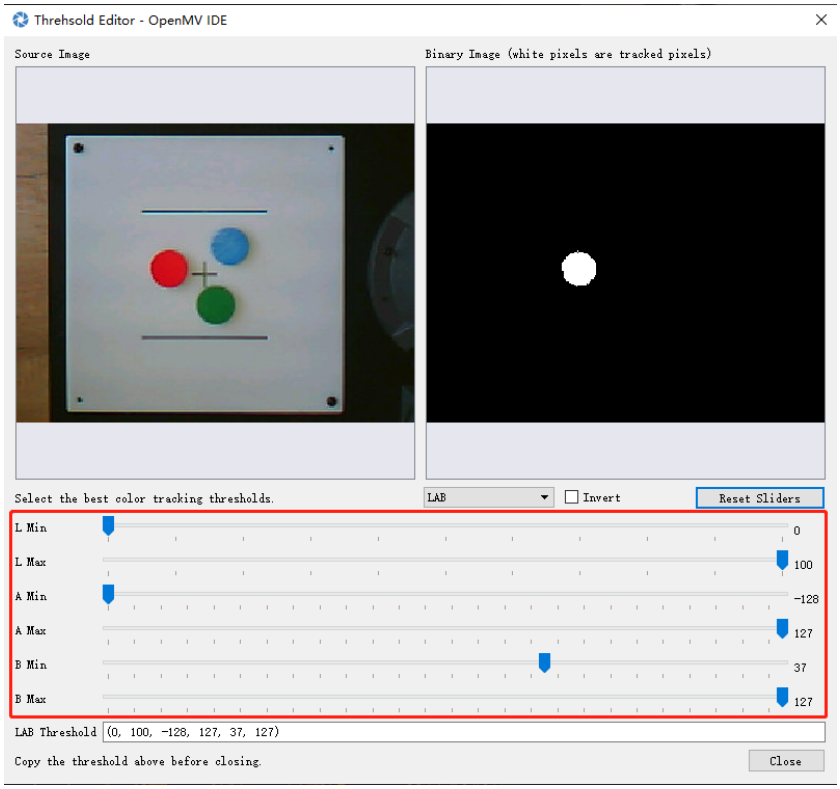


④重置滑块；

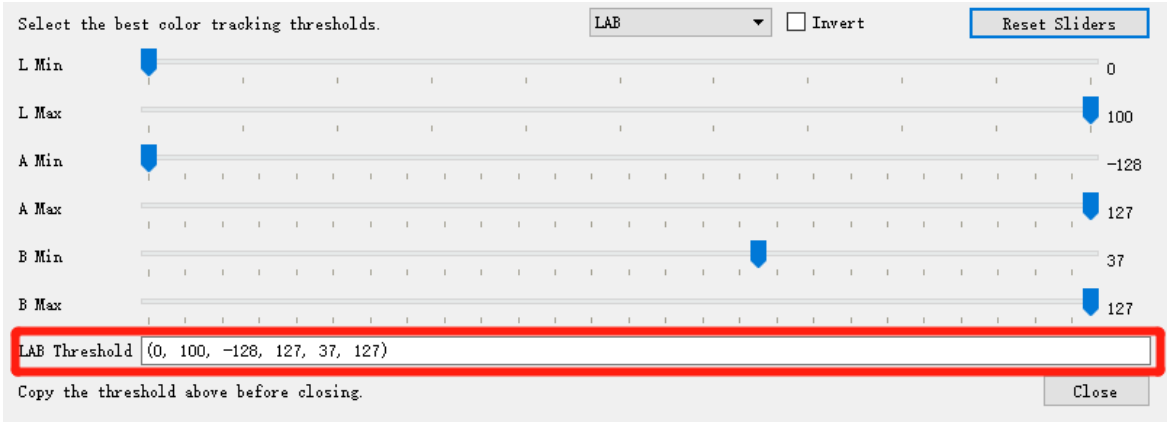


⑤以识别红色木块的 LAB 值为例，依次拖动 6 个滑块，直至将我们的目标颜色变成白色，其他颜色全变为黑色。注意：将目标颜色调整至白色后，应尽可能的增大颜色阈值范围，以增加颜色判断的容错能力，下面两张图片应优先选择第二张图片的阈值区间。





⑥这样就得到了目标的 LAB 阈值，复制后替换程序中 thresholds 对应数值即可；



（4）blob 色块对象

find_blobs 对象返回的是多个 blob 的列表。（注意区分 blobs 和 blob，这只是一个名字，用来区分多个色块，和一个色块）。列表类似与 C 语言的数组，一个 blobs 列表里包含很多 blob 对象，blobs 对象就是色块，每个 blobs 对象包含一个色块的信息。

blob 有多个方法：

- blob.rect() 返回这个色块的外框——矩形元组(x, y, w, h)，可以直接在 image.draw_rectangle 中使用。
- blob.x() 返回色块的外框的 x 坐标（int），也可以通过 blob[0]来获取。
- blob.y() 返回色块的外框的 y 坐标（int），也可以通过 blob[1]来获取。
- blob.w() 返回色块的外框的宽度 w（int），也可以通过 blob[2]来获取。
- blob.h() 返回色块的外框的高度 h（int），也可以通过 blob[3]来获取。
- blob.pixels() 返回色块的像素数量（int），也可以通过 blob[4]来获取。
- blob.cx() 返回色块的外框的中心 x 坐标（int），也可以通过 blob[5]来获取。
- blob.cy() 返回色块的外框的中心 y 坐标（int），也可以通过 blob[6]来获取。
- blob.rotation() 返回色块的旋转角度（单位为弧度）（float）。如果色块类似一个铅笔，那么这个值为 0~180°。如果色块是一个圆，那么这个值是无用的。如果色块完全没有对称性，那么你会得到 0~360°，也可以通过 blob[7]来获取。
- blob.code() 返回一个 16bit 数字，每一个 bit 会对应每一个阈值。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。这个功能可以用于查找颜色代码。也可以通过 blob[8]来获取。

- blob.count() 如果 merge=True，那么就会有多个 blob 被合并到一个 blob，这个函数返回的就是这个的数量。如果 merge=False，那么返回值总是 1。也可以通过 blob[9]来获取。
- blob.area() 返回色块的外框的面积。应该等于(w * h)
- blob.density() 返回色块的密度。这等于色块的像素数除以外框的区域。如果密度较低，那么说明目标锁定的不是很好。比如，识别一个红色的圆，返回的 blob.pixels()是目标圆的像素点数，blob.area()是圆的外接正方形的面积。

● 代码解析（文件：Color_sorting_mirobot.py）

更多解析请参考程序注释：

```
import sensor, image, time, math, lcd, os, pyb
from pyb import UART, Pin, Timer


value = './dfg.txt' # 校准信息储存位置

high_z = 50   # 吸取物块时，机械臂末端默认高度
bx = 150    # 物块校准默认坐标
by = 120    # 物块校准默认坐标
cy = 0  # 坐标偏移量
cx = 0  # 坐标偏移量

# 设置颜色阈值
thresholds = [(0, 100, 25, 127, -128, 127),    #红色
              (0, 100, -128, -18, 11, 127),    #绿色
              (1, 100, -128, 127, -128, -20)]  #蓝色

# 等待机械臂运动完成
def mirobot_wait_finish():
    inByte = ''
    print('wait')
    while inByte.find('>'):
        while uart.any() > 0:
            inByte = uart.readline().decode()
       #lcd.display(sensor.snapshot())
    #print(inByte)  # 打印机械臂返回数据，用于调试开发
    print('finish')
```

```
28.
29. pyb.delay(1000)  # 延时 1s
30.
31. # 设置串口端口
32. uart = UART(3, 115200)
33.
34. # 设置板载补光灯亮度
35. tim = Timer(4, freq=1000)           # 使用定时器 4 创建一个定时器对象-以 1000Hz 触发
36. Led = tim.channel(3, Timer.PWM, pin=Pin("P9"), pulse_width_percent=50)  # 配置 Pin9 的初始脉宽值百分比为 50(打开板载补光灯）
37.
38. # 摄像头初始化
39. sensor.reset()                          # 初始化摄像头
40. sensor.set_pixformat(sensor.RGB565)     # 设置图像色彩格式为 RGB565 色彩图
41. sensor.set_framesize(sensor.QVGA)       # 将图像大小设置为 QVGA (320x240)
42. sensor.skip_frames(time = 2000)         # 等待设置生效
43. sensor.set_auto_gain(False)             # 如果使用颜色识别，必须关闭
44. sensor.set_auto_whitebal(False)         # 如果使用颜色识别，必须关闭
45.
46. # 屏幕初始化
47. lcd.init()
48. print('LCD INIT')
49. x = (sensor.width() - lcd.width()) / 2
50. y = (sensor.height() - lcd.height()) / 2
51. lcd_roi = (int(x), int(y), lcd.width(), lcd.height())# 设置 LCD 屏幕显示区域
52.
53. # 设置按键引脚
54. Key = Pin('P6', Pin.IN, Pin.PULL_UP)
55. keyvalue = Key.value()          # 读取按钮状态
56. pyb.delay(1000)
57.
58. # 机械臂初始化
59. uart.write("$40 = 1\n")         # 设置机械臂打开回文
60. uart.write("$H\n")              # 机械臂复位
61. mirobot_wait_finish()          # 等待完成
62. pyb.delay(2000)
63. uart.write("M20 G90 G00 F2000\n")   # 设置机械臂笛卡尔运动模式
64. uart.write("M3S0\n")                # 关闭气泵
65. uart.write("X150 Y-150\n")          # 移开机械臂，防止挡住摄像头
66. mirobot_wait_finish()               # 等待动作完成
67.
68.
69.
```

```
70. # 如果开机时按钮按下，校准模式
71. if keyvalue==0:
72.     # 校准图像位置，等待直到按钮被按下
73.     while Key.value():
74.         img = sensor.snapshot()
75.         img.draw_cross(bx, by, color = (0, 0, 0), size = 10, thickness = 1)
    # 画十字
76.         img.draw_rectangle(bx-50, by-
50, 100,100, color = (0, 0, 0), thickness = 1, fill = False)       # 画边长为100
的方框
77.         lcd.display(img,roi = lcd_roi)       # LCD 屏居中显示缓冲区图像
78.     uart.write('X210Y0'+'z'+str(high_z)+'\n')       # 机械臂末端移动至
X210 Y0 Z50，此处为标定板中心正上方
79.
80.     keytime = 0                             # 将红色圆片放在标定板中心
81.     # 校准吸盘高度，等待按钮被长按
82.     while(True):
83.         img = sensor.snapshot()
84.         lcd.display(img,roi = lcd_roi)       # LCD 屏居中显示缓冲区图像
85.         while Key.value()==0:
86.             keytime+=1
87.             pyb.delay(1)
88.         if keytime in range(20,500):     # 短按按钮，机械臂末端下移1mm
89.             high_z-=1
90.             uart.write('z'+str(high_z)+'\n') # 打印高度
91.             if high_z < -10:
92.                 print('error\n')
93.                 break
94.         elif keytime > 2000:             # 长按确认高度，退出校准
95.             uart.write("Z120\n")
96.             uart.write("X150Y-150\n")# 移开机械臂，防止挡住摄像头
97.             print("X150Y-150Z120")
98.             mirobot_wait_finish()        # 等待动作完成
99.             # 自动校准
100.             while(True):
101.                 find = 0
102.                 img = sensor.snapshot()
103.                 for blob in img.find_blobs(thresholds, x_stride=5, y_stride=
5,pixels_threshold=50):     #识别红色块
104.                     if blob.code() == 1: # 识别到红色块后，将识别到的物块中心坐
标设为校准的默认坐标
105.                         img.draw_string(blob.x(), blob.y() + 10, 'R')
106.                         img.draw_cross(blob.cx(), blob.cy())
107.                         img.draw_rectangle(blob.rect())
```

```
                        bx=blob.cx()
                        by=blob.cy()
                        print('colour: ',blob.cx(), blob.cy())
                        find = 1
                        break
                if find == 1:
                    break
                lcd.display(img,roi = lcd_roi)
            pyb.delay(1000)
            # 记录校准信息
            f = open(value,'w')
            f.write(str(high_z)+'\n')
            f.write(str(bx)+'\n')
            f.write(str(by)+'\n')
            print('Variable writing...')
            print('Initial height: ',high_z)
            print('x Relative coordinates: ',bx)
            print('y Relative coordinates: ',by)
            f.close()
            break
        keytime = 0
# 读取校准信息
try:
    f = open(value,'r')
    high_z = int(f.readline())
    bx = int(f.readline())
    by = int(f.readline())
    print('Variable writing...')
    print('Initial height: ',high_z)
    print('x Relative coordinates: ',bx)
    print('y Relative coordinates: ',by)
# 否则写入默认参数
except:
    f = open(value,'w')
    f.write(str(high_z)+'\n')
    f.write(str(bx)+'\n')
    f.write(str(by)+'\n')
    print('Variable writing...')
    print('Initial height: ',high_z)
    print('x Relative coordinates: ',bx)
    print('y Relative coordinates: ',by)
    f.close()
else:
    f.close()
```

```
152.
153.
154.
155. # 设置分拣区域
156. color_roi = (bx-60, by-60, 120, 120)
157.
158. # 分拣物块
159. while(True):
160.     img = sensor.snapshot()                  # 获取图像
161.     img.draw_cross(bx, by, color = (0, 0, 0), size = 10, thickness = 1)
162.     img.draw_rectangle(bx-50, by-
50, 100,100, color = (0, 0, 0), thickness = 1, fill = False)
163.     # 在color_roi 范围内查找色块
164.     for blob in img.find_blobs(thresholds, roi = color_roi, x_stride=15, y_s
tride=15,pixels_threshold=150):
165.         color = ''
166.         print(str(blob.code()))             # 打印阈值代号
167.         if blob.code() == 1:
168.             color = 'R'
169.             img.draw_string(blob.x(), blob.y() + 10, 'R')
170.         elif blob.code() == 2:
171.             color = 'G'
172.             img.draw_string(blob.x(), blob.y() + 10, 'G')
173.         elif blob.code() == 4:
174.             color = 'B'
175.             img.draw_string(blob.x(), blob.y() + 10, 'B')
176.
177.         img.draw_cross(blob.cx(), blob.cy()) # 绘制十字
178.         img.draw_rectangle(blob.rect())      # 绘制方框
179.         lcd.display(img,roi = lcd_roi)
180.         print('Center coordinates: ',bx, by)
181.         print('Color coordinates: ',blob.cx(), blob.cy())
182.
183.         # 颜色坐标转换（图像左上角为坐标原点）
184.         y = 0+(blob.cy()-by+cy)
185.         x = 210+(bx-blob.cx()+cx)
186.         print('Relative coordinates: ',x, y)
187.
188.         # 发送指令到机械臂
189.         output_str="x%d y%d z100\n" % (x, y)
190.         print('Output coordinates: '+output_str)
191.         uart.write(output_str)                # 移动吸盘到色块正上方
192.         mirobot_wait_finish()                 # 等待运动完成
193.         uart.write('Z'+str(high_z)+'\n')      # 降低吸盘高度
```

```
194.            uart.write(" M3 S1000"+'\n')            # 打开气泵
195.            mirobot_wait_finish()                   # 等待运动完成
196.            uart.write("M20 G00 G90 Z120\n")                    # 提高吸盘高度
197.            mirobot_wait_finish()                   # 等待运动完成
198.            if color == 'R':                        # 移动吸盘到存放区
199.                uart.write("X70Y-150\n")
200.            elif color == 'B':
201.                uart.write("X210Y-150\n")
202.            elif color == 'G':
203.                uart.write("X140Y-150\n")
204.            mirobot_wait_finish()                   # 等待运动完成
205.            uart.write("M3S0\n")                    # 关闭气泵
206.            uart.write("X150 Y-150 Z150\n")          # 移开机械臂，防止挡住摄像头
207.            break
```

官方公众号