

南京宇微电子科技有限公司 FKMV CAM H7 R2 相机产品 V1.0

2025 年 1 月 21 日修订

FU KUN

版本修订

时间	版本号	修订内容
2025 年 1 月 21 日	V1.0	初版

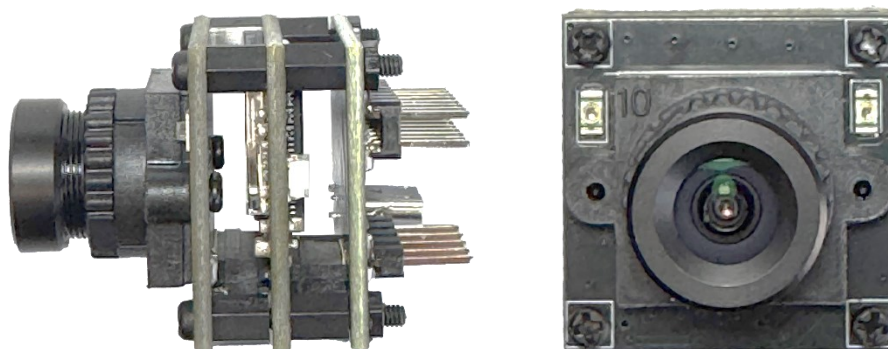
客户须知

本文档为产品使用参考所编写，文档版本可能随时更新，恕不另行通知。本文中提供的所有使用方法、说明及建议仅供参考，不构成任何承诺或保证。使用本产品及本文档内容所产生的结果，由用户自行承担风险。本公司对因使用本文档或产品而导致的任何直接或间接损失，不承担任何责任。

一、FKMV CAM H7 R2 相机

这是一款全新设计的超小型相机，专为高性能嵌入式视觉应用打造。这款相机在保持强大功能的同时，显著减小了尺寸，为您的项目提供更灵活的集成选择。采用 3 层 PCB 堆叠的小型化设计，每片 PCB 尺寸仅有 2.6cm*2.6cm，使用绝缘的尼龙柱连接，可独立安装使用，适用小型化场景。同时，配备转接板，可以接支持入多种外设，如补光板、屏幕、wifi 等模组。

主芯片选用原装 STM32H743VIT6 主控，PCB 采用 6 层板设计，信号质量好，散热性能优异，可长时间持续工作。相机选用原装 ov7725（非拆机传感器），选用大厂无畸变镜头，增加了两颗高强度红外 LED 灯珠，质量保证，价格优惠。（镜头默认配有高截止质量的红外滤波片）



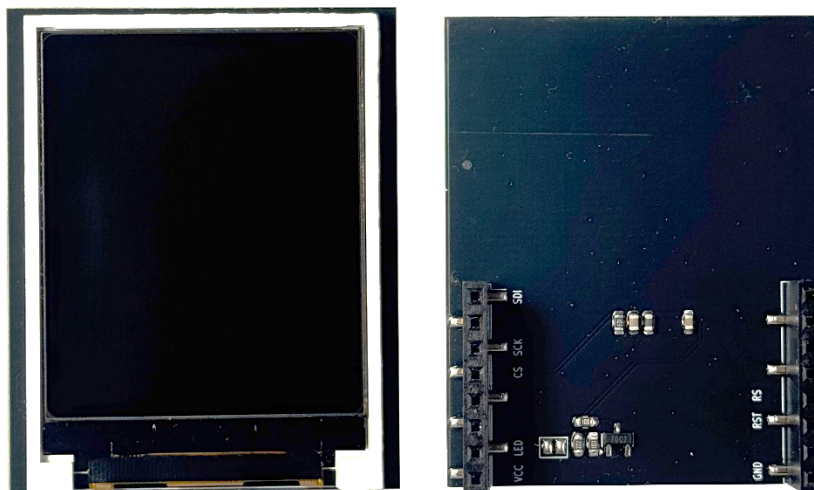
参数	
镜头	2.8mm 无畸变镜头
图像传感器	OV7725
控制器	STM32H743 vit6
接口	SPI/IIC/CAN/UART/DAC/USB
存储容量	≤32G（TF-Card）
指示灯	三色 RGB led 灯
补光灯	850nm 红外 led 灯
重量	17g（含镜头帽）
尺寸	26mm*26mm*41mm（含尾部针角）
供电要求	支持 USB 5V，电池 4.2V

二、相机配件

目前推出的配件有 3 种，TFT-LCD 显示屏、补光灯模块和桥接版，因为需要同时兼容 OpenMV4 H7 相机主板，所以配件模块设计比较大，与体积更小巧的 FKMV4 H7 连接需要一块桥接板。

2.1 TFT-LCD

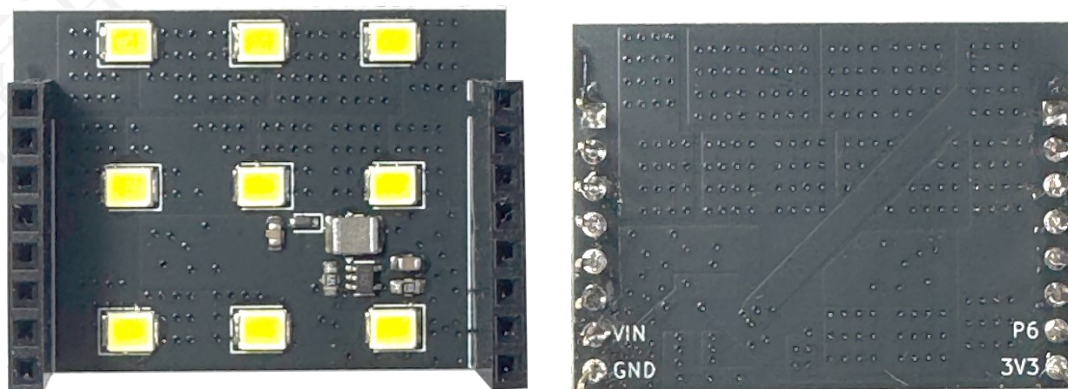
TFT LCD（薄膜晶体管液晶显示器）是一种广泛用于显示具有鲜艳色彩和清晰细节的高质量视觉效果 LCD 技术，这款 1.8 寸显示屏采用了 ST7735 驱动芯片驱动，支持通过四线 SPI 接口与 MCU 交互。当与专为图像处理而设计的 FKMV 集成时，它可以实时显示摄像头输出，从而增强 FKMV 系统在基于视觉的机器人、物体检测和图像流等应用中的可用性。



参数	
模块尺寸	宽：39mm，长：48mm
分辨率	128*RGB*160
驱动 IC 型号	ST7735S
背光类型	三星 LED*2，电流：30Ma
电压	工作电压 $V_{cc}=2.8\sim3.3V$
视角	12:00 o'clock
接口类型	MCU 串口 SPI 4 线
功耗	0.11W
工作温度	$-20^{\circ}C\sim+70^{\circ}C$
存储温度	$-80^{\circ}C\sim+80^{\circ}C$

2.2 补光灯模块

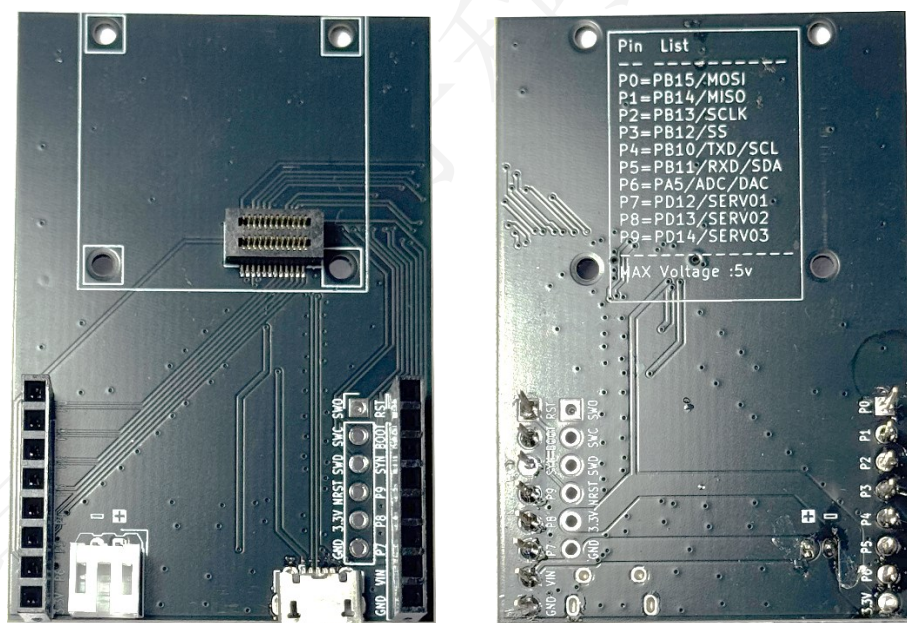
补光灯模块是一款专为 FKMV 摄像头设计的附加板，旨在扩展其在机器人和自动化应用方面的功能。该模块配备高强度 LED，可在弱光环境下提供增强照明，非常适合在昏暗条件下执行基于视觉的任务，例如物体检测、线路跟踪和人脸识别。光强度可以通过编程控制，允许根据环境的照明需求进行动态调整。该模块结构紧凑且易于集成，是对 FKMV 生态系统的补充，可为计算机视觉项目提供更好的性能和多功能性支撑。



参数	
LED 类型	高强度白色 LED/黄色胶体
LED 数量	9 个 LED
照明角度	约 120°
光强度	额定 26LM 可通过 PWM 或软件控制进行调节
色温	6000~7000K
输入电压	3.3V 或 5V（取决于与 OpenMV 板的兼容性）
功耗	约为 1.8W（当所有 LED 都处于全亮度时）
控制接口	PWM
尺寸	26mm*36mm
工作温度	-20℃~+85℃
存储温度	-20℃~+85℃

2.3 桥接板

TFT LCD（薄膜晶体管液晶显示器）是一种广泛用于显示具有鲜艳色彩和清晰细节的高质量视觉效果 LCD 技术，这款 1.8 寸显示屏采用了 ST7735 驱动芯片驱动，支持通过四线 SPI 接口与 MCU 交互。当与专为图像处理而设计的 FKMV 集成时，它可以实时显示摄像头输出，从而增强 FKMV 系统在基于视觉的机器人、物体检测和图像流等应用中的可用性。



参数	
模块尺寸	宽：39mm，长：48mm
接口	所有 FKMV 引出接口
功能	连接 FKMV 与其它外设

三、使用方法

3.1 相机组装

3.2 软件下载

电脑端打开下载网址：[Download – OpenMV](#)，找到下载界面，选择下载适配您电脑的软件安装包，如果因为网络问题您无法下载成功，我们为您准备了安装包，分享地址如下：

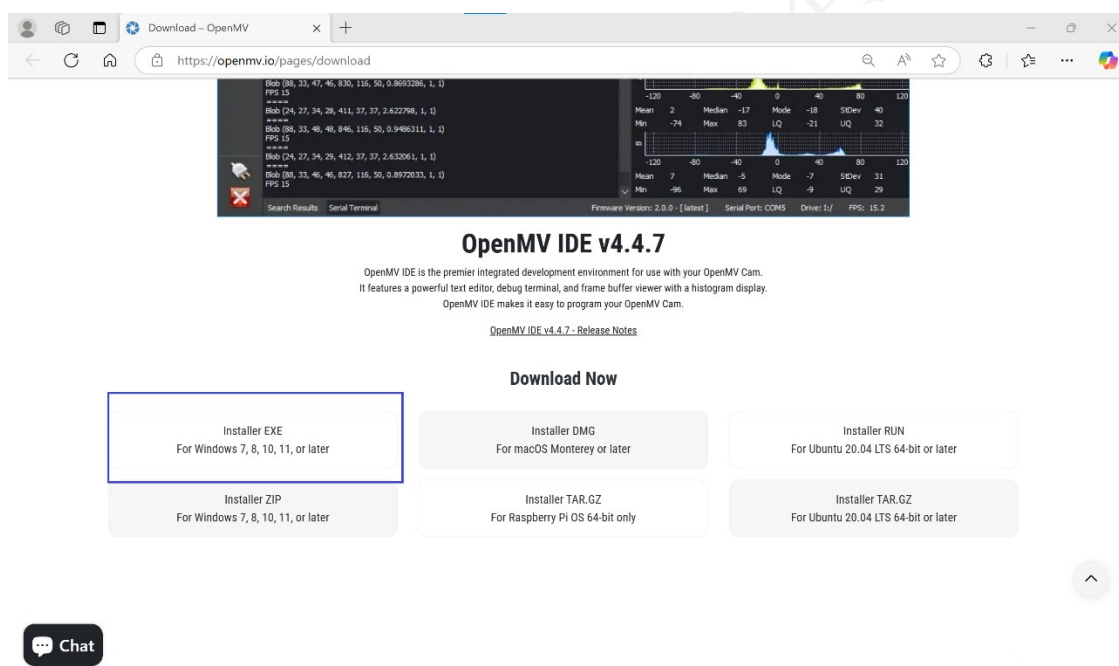
阿里云盘：<https://www.alipan.com/s/TbSUSwsg9et> 提取码: 37ze

百度网盘：<https://pan.baidu.com/s/1sBfvTnzIGi24zeOvT3r21g?pwd=t84q> 提取码: t84q


下面我们介绍在 Win10 系统环境下和 Ubuntu 20.04 系统环境下的安装流程。

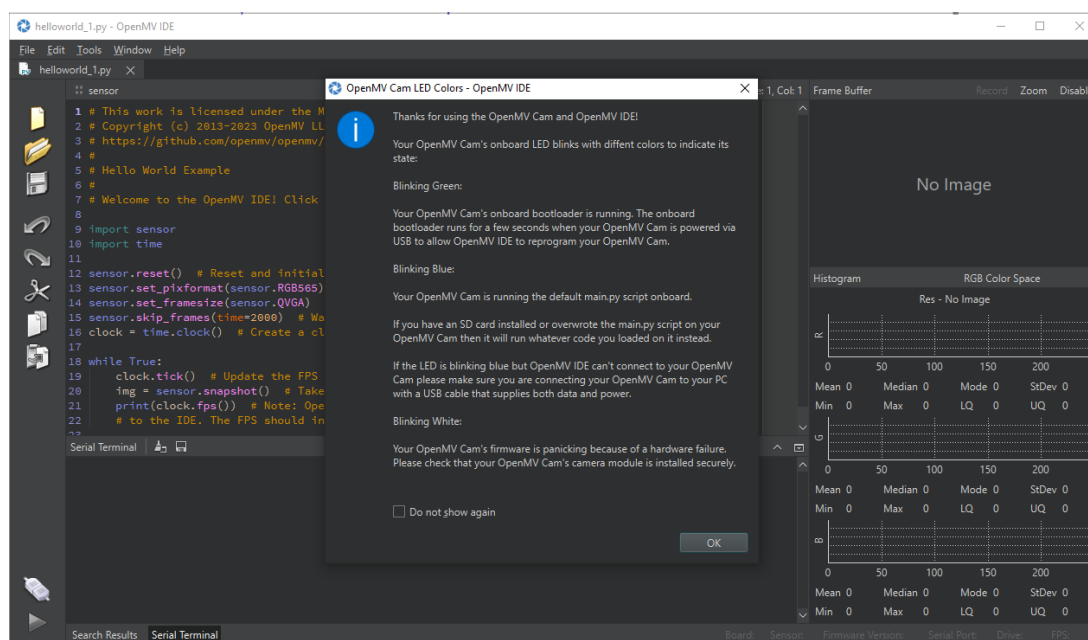
3.2.1 Win10 系统

第一步：点击 Installer EXE For Windows 7,8,10,11 or later，浏览器会执行下载。

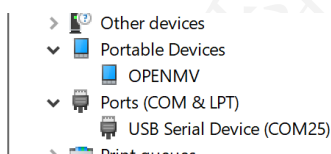


第二步：左键双击安装包，选择您希望安装的位置，并同意相关的条款，按照提示一路 NEXT，最后执行安装。安装时间与您的电脑配置有关，约 3 分钟，最后点击完成。

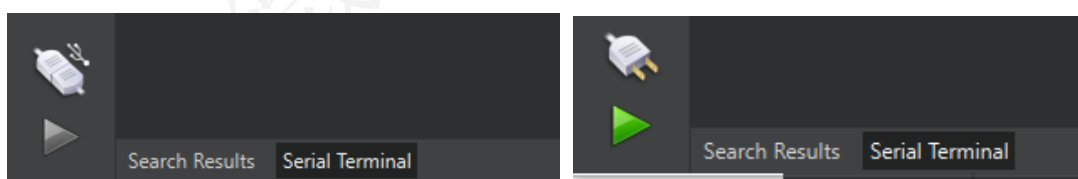
第三步：在桌面找到快捷方式图标 ，双击打开



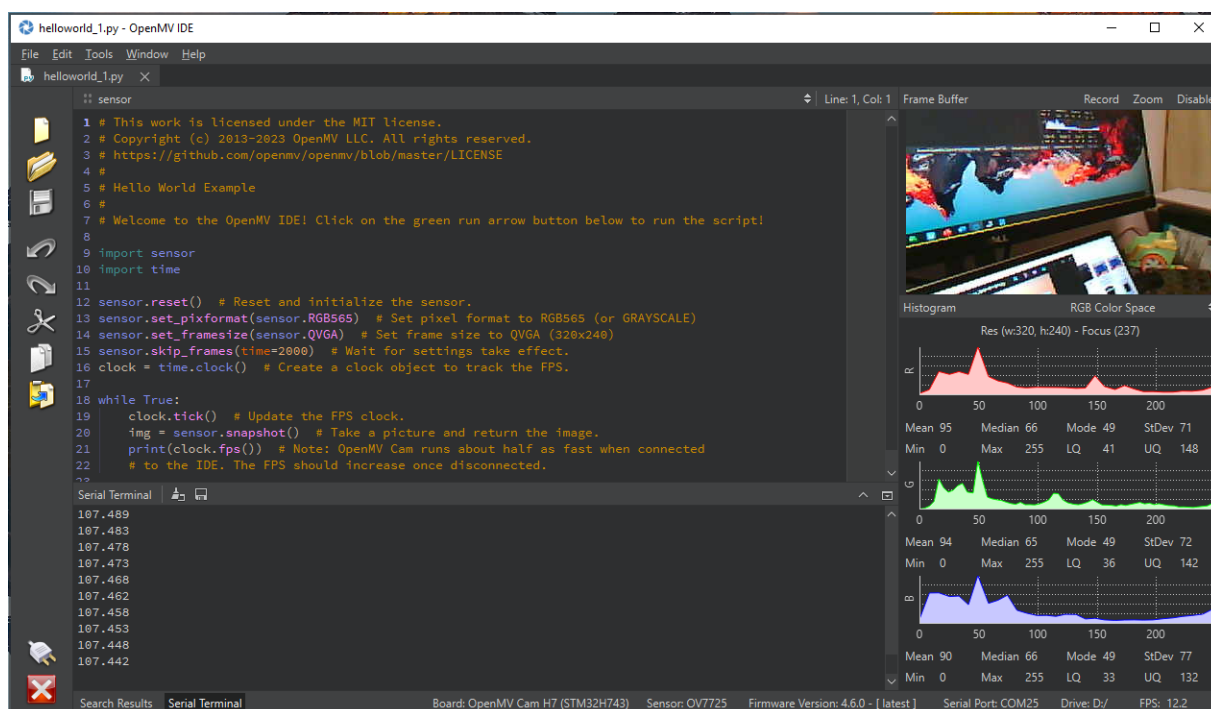
第四步：使用数据线将 FKMV 相机接入到电脑，相机与电脑连接后会有连接提示音，同时在设备管理器中的“便携式设备”中会出现 OPENMV，在“端口”中会出现一个新的 USB 串口设备，串口号 COMxx 不同电脑设备分配的会有不同。



第五步：点击 IDE 左下角的连接按钮，左边的状态会变为右边的状态，因为 FKMV 不是 OpenMV 官方的设备，所以会有提示框弹出，按照您的需要执行就好了。



第六步：当您处理好上一步的提示问题，就可以点击绿色的小按钮实现连接，这时 FKMV 相机会执行 IDE 默认生成的程序。



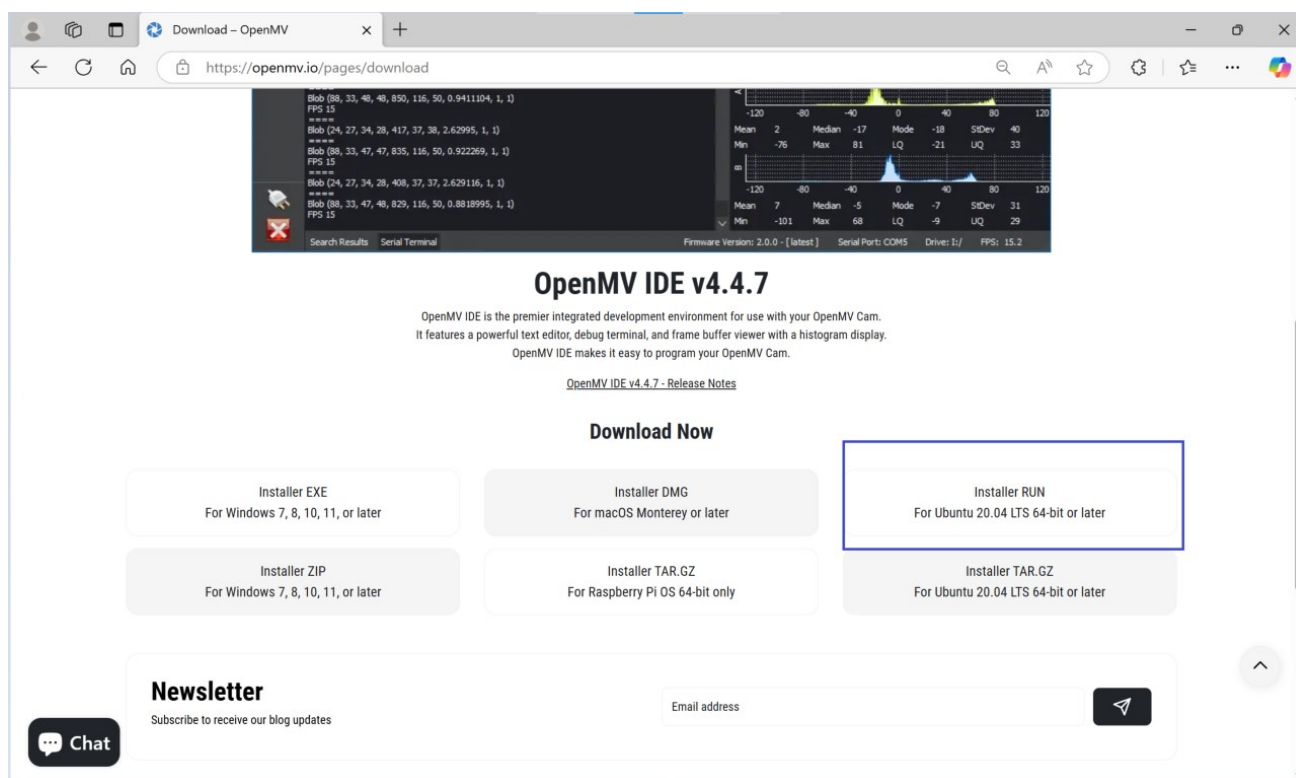
以上就是在 Win10 系统中安装 IDE 的基本流程，IDE 的使用方式及案例尝鲜请查阅第 4 节相关内容。

3.2.2 Ubuntu 20.04 系统

第一步：点击 Installer RUN For Ubuntu 20.04 LTS 64-bit or later，浏览器会执行下载，如果因为网络问题您无法下载成功，我们为您准备了安装包在百度网盘和阿里云盘，分享地址如下：

阿里云盘：<https://www.alipan.com/s/TbSUSwsg9et> 提取码: 37ze

百度网盘：<https://pan.baidu.com/s/1sBfvTnzIGi24zeOvT3r21g?pwd=t84q> 提取码: t84q



第二步：使用鼠标左键单击打开终端，或者使用键盘快捷键“Ctrl+T”打开终端。

第三步：使用 `cd` 命令找到到您下载的安装包位置，依次执行 `sudo apt update` 和 `sudo apt upgrade`。执行 `python3 --version`，若的结果显示没有安装 `python`，则执行 `sudo apt install python3 -y`，否则这一步结束。

第四步：在终端依次执行命令“`chmod +x openmv-ide-linux-*.run`”和“`./openmv-ide-linux-*.run`”，标黄色位置应是您下载的安装包名称，如 `openmv-ide-linux-x86_64-4.4.7.run`。

第五步：按照提示执行安装即可，安装完成后打开 IDE，使用附赠的数据线接入 FKMV 相机，即可正常使用。一些例程的使用介绍，请查阅 3.3 节。

3.3 运行例程

我们以 Win10 平台为例，展示一些 IDE 自带的 demo 例程。

3.3.1 IDE 显示图像

路径 `File/Examples/Helloworld/helloworld.py`

```
# This work is licensed under the MIT license.
# Copyright (c) 2013-2023 OpenMV LLC. All rights reserved.
# https://github.com/openmv/openmv/blob/master/LICENSE
#
# Hello World Example
#
```

```
# Welcome to the OpenMV IDE! Click on the green run arrow button below to run the script!
```

```
import sensor
import time

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

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

这段代码是一个简单的 OpenMV 示例脚本，用于展示如何使用 OpenMV 的图像传感器库进行基本操作。该程序初始化摄像头模块，并设置像素格式和帧大小。每次循环中拍摄一张图片，并打印当前帧率。是一个“Hello World”级别的 OpenMV 示例，非常适合入门学习。以下是代码的详细分析：

3.3.1.1 导入库

```
import sensor
import time
```

- **sensor**: OpenMV 提供的库，用于控制摄像头模块的行为，例如设置分辨率、拍照等。
- **time**: 标准 Python 库，用于管理时间与帧率。

3.3.1.2 初始化传感器

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

- **sensor.reset()**: 重置摄像头模块，初始化传感器。
- **sensor.set_pixformat(sensor.RGB565)**: 设置像素格式为 RGB565（即每个像素使用 16 位存储颜色）。可选值：sensor.GRAYSCALE（灰度图）或其他支持的格式。
- **sensor.set_framesize(sensor.QVGA)**: 设置帧大小为 QVGA (320x240)。可选值包括 VGA、QQVGA 等。
- **sensor.skip_frames(time=2000)**: 跳过若干帧，等待 2 秒以使设置生效。

- `time.clock()`: 创建一个时钟对象，用于跟踪帧率（FPS）。

3.3.1.3 主循环

while True:

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

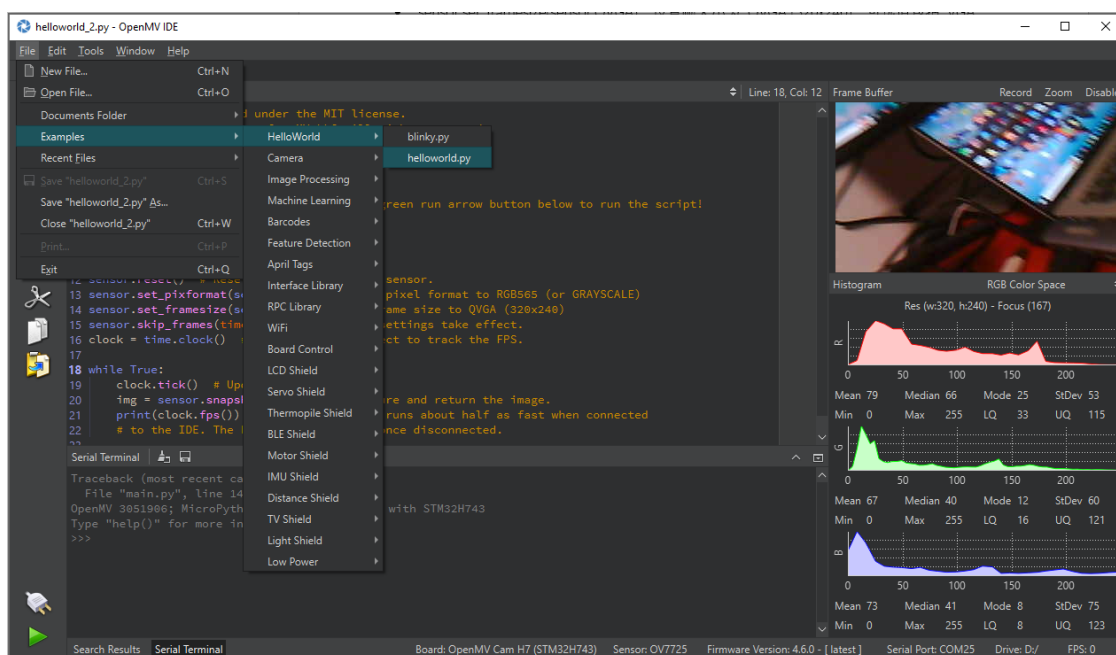
- `clock.tick()`: 记录每次循环的时间间隔，更新 FPS（每秒帧数）。
- `sensor.snapshot()`: 拍摄一帧图像并返回图像对象（可以进行后续处理）。
- `print(clock.fps())`: 打印实时的帧率。注意：当 OpenMV 摄像头连接到 IDE 时，性能会降低，断开连接后帧率会提高。

3.3.1.4 演示

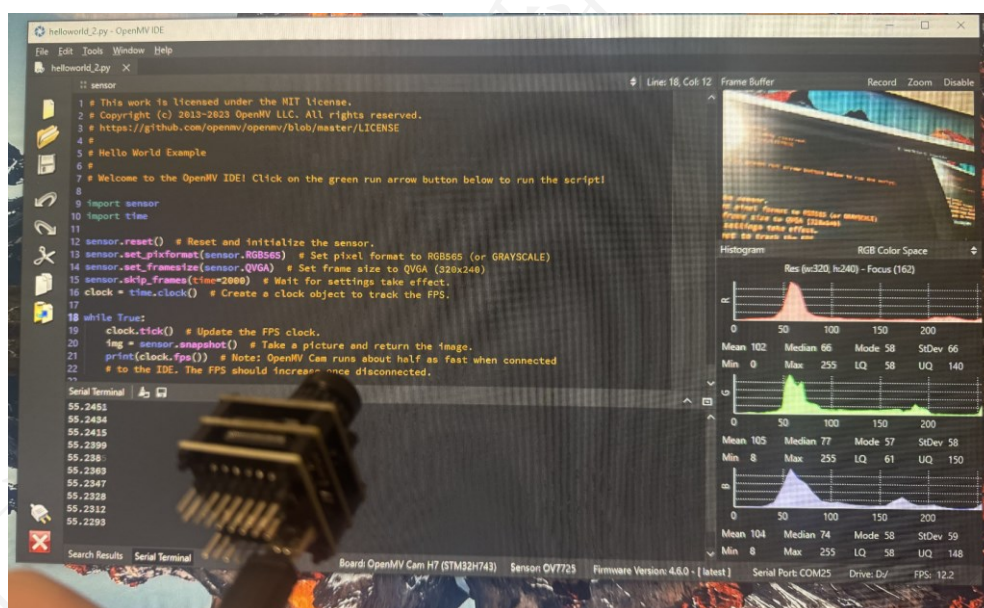
本次演示使用的设备如下图所示



将 FKMV 相机通过附赠的 USB 数据线连接到电脑 USB 口，点击左下角的 USB 连接标志。成功连接后，找到路径为 File/Examples/Helloworld 程序路径，左键点击 `helloworld.py` 代码，IDE 会自动加载此代码到工作区。如未成功连接，以上程序路径是灰色不可选状态。在连接过程中会跳出让您购买许可证的提示，我们建议您根据自身情况，合理支持开源社区的工作。我们这里为了演示方便，就先点击“否”了。



成功加载代码后，点击做下角的绿色三角，执行这段代码。取下镜头盖后，在右上角的捕获框内，就能显示实时捕获的图像了，在左下角点击串口终端，则可以显示实时打印的信息。当您发现图像捕获框内图像不清晰，意味着您需要在保证镜头清洁的前提下，旋转镜头以获得更好的对焦。



3.3.2 TFT LCD 显示图像

路径 File/Examples/LED Shield/lcd.py

```
# This work is licensed under the MIT license.
# Copyright (c) 2013-2023 OpenMV LLC. All rights reserved.
# https://github.com/openmv/openmv/blob/master/LICENSE
```

```
#  
# LCD Example  
#  
# Note: To run this example you will need a LCD Shield for your OpenMV Cam.  
#  
# The LCD Shield allows you to view your OpenMV Cam's frame buffer on the go.  
  
import sensor  
import display  
  
sensor.reset() # Initialize the camera sensor.  
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE  
sensor.set_framesize(sensor.QQVGA2) # Special 128x160 framesize for LCD Shield.  
# Initialize the lcd screen.  
# Note: A DAC or a PWM backlight controller can be used to control the  
# backlight intensity if supported:  
# lcd = display.SPIDisplay(backlight=display.DACBacklight(channel=2))  
# lcd.backlight(25) # 25% intensity  
# Otherwise the default GPIO (on/off) controller is used.  
lcd = display.SPIDisplay()  
  
while True:  
    lcd.write(sensor.snapshot()) # Take a picture and display the image.
```

这是一个 LCD 显示屏示例。需要使用 OpenMV 的 LCD Shield（硬件扩展模块）以实现摄像头帧缓冲区的实时显示。该程序捕获摄像头的实时图像，并通过 LCD Shield 显示。图像分辨率设置为 QQVGA2 (128x160)，匹配 LCD Shield 的屏幕大小。

3.3.2.1 导入库

```
import sensor  
import display
```

- sensor: 用于摄像头的初始化、设置分辨率和捕获帧。
- display: 用于操作 LCD Shield 的显示功能。

3.3.2.2 摄像头初始化

```
sensor.reset() # Initialize the camera sensor.  
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE  
sensor.set_framesize(sensor.QQVGA2) # Special 128x160 framesize for LCD Shield.
```

- sensor.reset(): 初始化摄像头传感器。
- sensor.set_pixformat(sensor.RGB565): 设置像素格式为 RGB565（16 位颜色）。可选值：sensor.GRAYSCALE（灰度图）。

- `sensor.set_framesize(sensor.QQVGA2)`: 设置帧大小为 QQVGA2 (128x160), 这是 LCD Shield 的专用分辨率。

3.3.2.3 LCD 屏幕初始化

```
# Initialize the lcd screen.  
# Note: A DAC or a PWM backlight controller can be used to control the  
# backlight intensity if supported:  
# lcd = display.SPIDisplay(backlight=display.DACBacklight(channel=2))  
# lcd.backlight(25) # 25% intensity  
# Otherwise the default GPIO (on/off) controller is used.  
lcd = display.SPIDisplay()
```

- `display.SPIDisplay()`: 初始化 LCD 屏幕, 使用 SPI 接口进行通信。
- 注释部分提到: 如果支持, 可以使用 DAC 或 PWM 控制背光强度: 如果没有 PWM 控制, 默认使用 GPIO 控制背光 (仅支持开/关)。

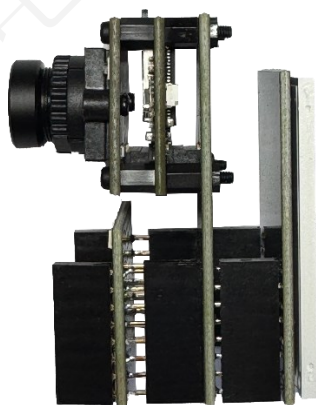
3.3.2.4 主循环

```
while True:  
    lcd.write(sensor.snapshot()) # Take a picture and display the image.
```

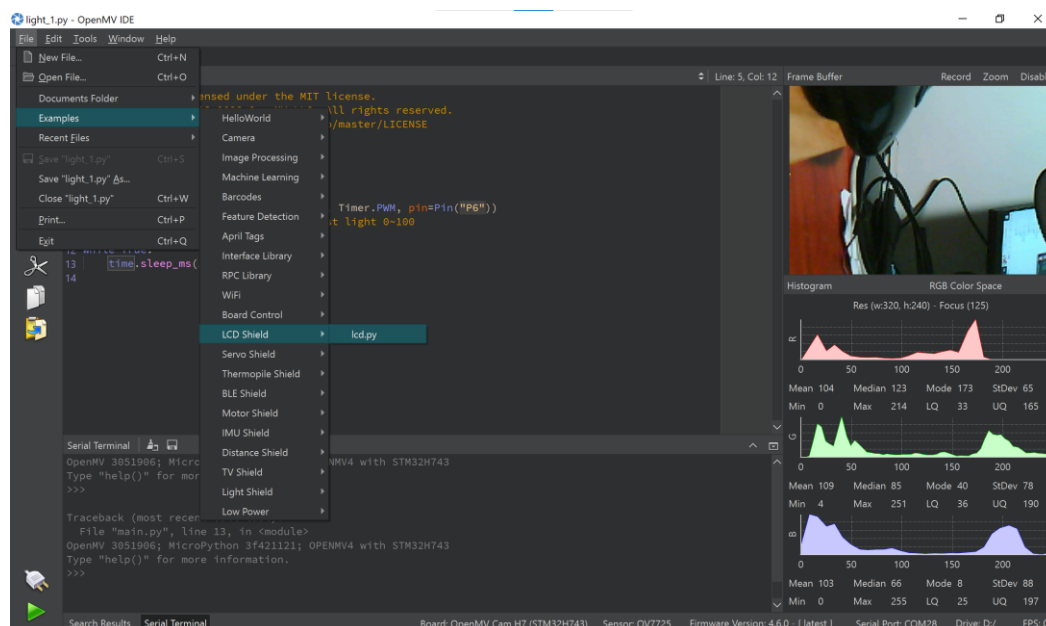
- `sensor.snapshot()`: 捕获一帧图像, 并返回图像数据。
- `lcd.write()`: 将捕获的图像数据写入 LCD 显示屏, 实时显示。

3.3.2.5 演示

本次演示使用的设备如下图所示, 本次实验需要一块转接板, 外接显示屏模组。这种兼容设计, 在保证小型化的基础上, 可以兼容市面上绝大多数 OpenMV 外设。

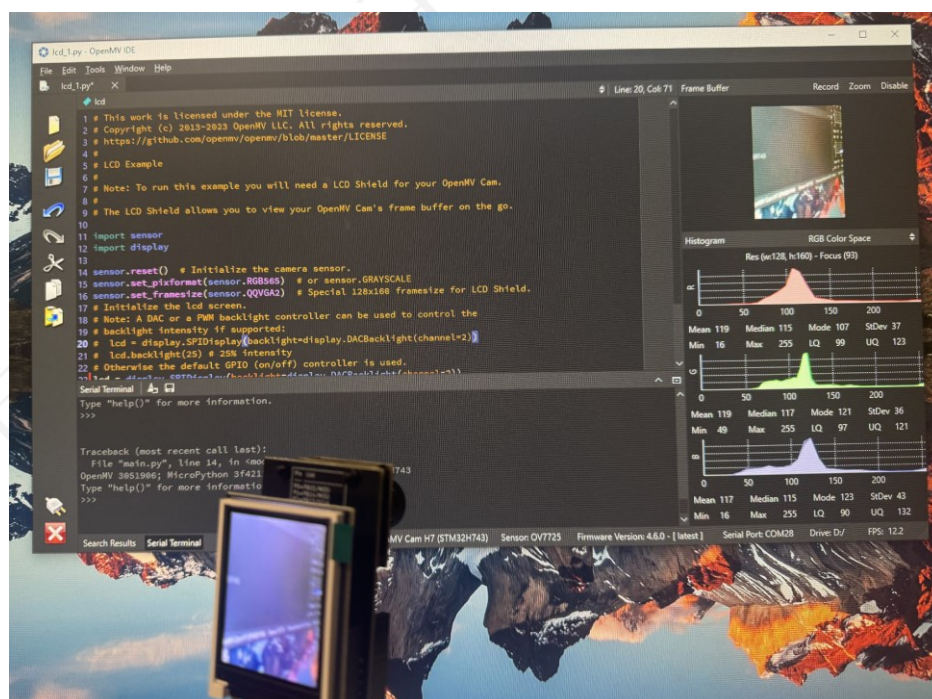


程序的路径如下, 注意事项参见 3.3.1.4 说明。



连接成功后，可以看到 IDE 右上角视频捕获界面与显示屏均有图像显示，但是屏幕亮度比较低。可以通过调整 P6 口的 DAC 输出背光控制电压，从而控制显示屏亮度。我们把 3.3.2.3 LCD 屏幕初始化的代码做如下更改

```
lcd = display.SPIDisplay(backlight=display.DACBacklight(channel=2))
lcd.backlight(75)
```



3.3.3 补光灯

路径 File/Examples/Light Shield/light.py

```
# This work is licensed under the MIT license.
# Copyright (c) 2013-2023 OpenMV LLC. All rights reserved.
# https://github.com/openmv/openmv/blob/master/LICENSE
#
import time
from pyb import Pin, Timer

# 50kHz pin6 timer2 channel1
light = Timer(2, freq=50000).channel(1, Timer.PWM, pin=Pin("P6"))
light.pulse_width_percent(100) # adjust light 0~100

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

这段代码的功能是使用 OpenMV 板控制一根引脚（P6）输出 50kHz 的 PWM 信号，同时可以调整信号的占空比来控制亮度或其他负载特性。初始化定时器 2，设置为 50kHz 的 PWM 输出。在引脚 P6 上生成 PWM 信号，占空比为 0~100%可调。

3.3.3.1 导入库

```
import time
from pyb import Pin, Timer
```

- time: 提供时间相关功能，例如延时（sleep）。
- pyb: OpenMV 板上特有的库，用于低级硬件控制。Pin: 管理引脚功能（GPIO）。Timer: 定时器模块，用于生成 PWM、定时中断等。

3.3.3.2 PWM 信号生成

```
# 50kHz pin6 timer2 channel1
light = Timer(2, freq=50000).channel(1, Timer.PWM, pin=Pin("P6"))
light.pulse_width_percent(100) # adjust light 0~100
```

- Timer(2, freq=50000): 初始化定时器 2，设置频率为 50kHz。
- .channel(1, Timer.PWM, pin=Pin("P6")): 在定时器的通道 1 上创建一个 PWM 输出，绑定到引脚 P6。
- light.pulse_width_percent(100): 设置 PWM 信号的占空比为 100%（完全导通）。占空比范围为 0~100，用于控制信号的高电平时间比例。

3.3.3.3 主循环

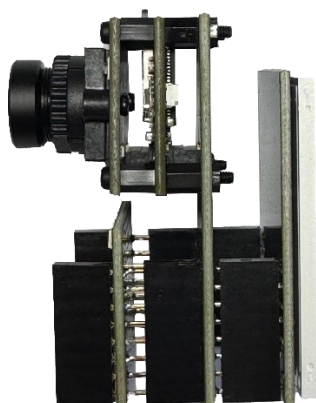
```
while True:
    time.sleep_ms(1000)
```

- while True: 无限循环，保持程序运行。
- time.sleep_ms(1000): 每次循环延时 1000 毫秒（1 秒）。

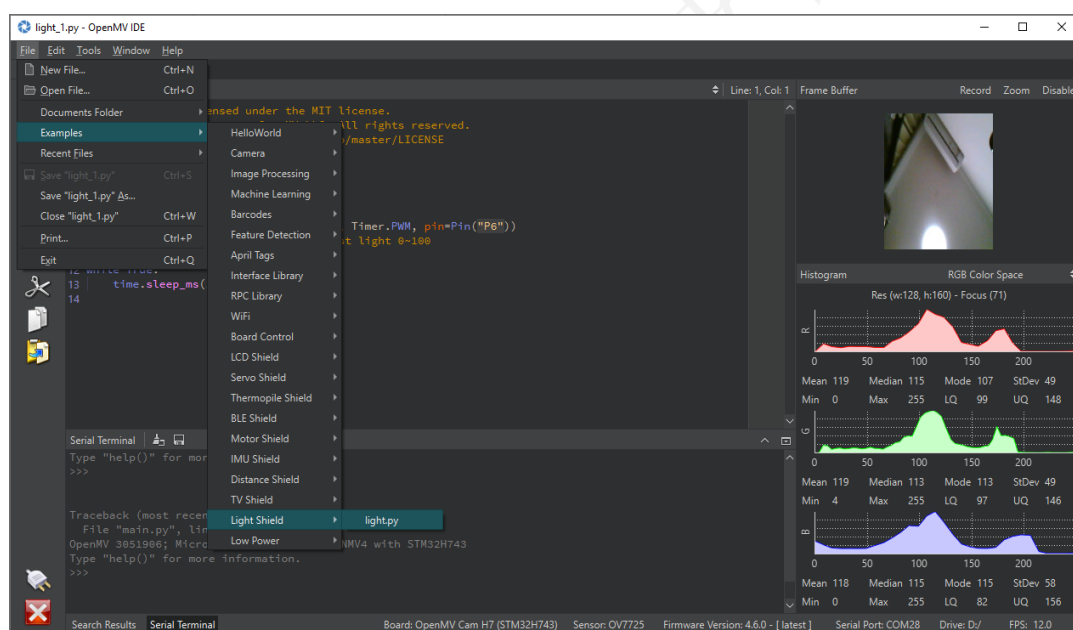
当前代码中，循环内没有其他操作，仅用于保持信号输出。

3.3.3.4 演示

次演示使用的设备如下图所示，本次实验需要一块转接板，外接显示屏模组。这种兼容设计，在保证小型化的基础上，可以兼容市面上绝大多数 OpenMV 外设。



程序的路径如下，注意事项参见 3.3.1.4 说明。



因为补光灯的功率比较大，最大功耗高达 3W 以上，所以仅用 usb 从电脑取电是没办法满足系统运行要求的。为此，需要外接电池供电，FKMV 提供了 2.0mm 电池接口，本次我们使用线性电源供电以演示。因为默认程序提供的是固定亮度输出，为了更直观的演示对亮度的控制，我们对代码做了一下修改

```
import time
from pyb import Pin, Timer

# 初始化定时器和 PWM 信号
# 50kHz, 定时器 2, 通道 1, 使用引脚 P6
```

```
light = Timer(2, freq=50000).channel(1, Timer.PWM, pin=Pin("P6"))

brightness = 0 # 初始亮度
increment = 10 # 每次增加或减少的亮度百分比

while True:
    light.pulse_width_percent(brightness) # 设置当前亮度
    print("Brightness:", brightness) # 打印当前亮度值（可选）
    time.sleep(1) # 每隔一秒调整一次亮度

    # 调整亮度值
    brightness += increment
    if brightness > 100 or brightness < 0: # 反向调整亮度
        increment = -increment # 改变增量方向
    brightness += increment # 防止越界
```

以上程序，使用的 50KHz 频率输出，占空比从 0 到 100，再从 100 到 1，每隔 1 秒变化一次，循环往复，我们录制了演示视频展示

阿里云盘：补光灯循环亮度控制测试.mp4 <https://www.alipan.com/s/1FKhzHFcf24> 提取码: 91vy

百度网盘：补光灯循环亮度控制测试.mp4 链接：

<https://pan.baidu.com/s/1YzSM6QVULtYRFtdJLWS0xg?pwd=epki>

提取码: epki

3.3.5 录制视频

路径 File/Examples/Camera/Video Recording/mjpeg.py

```
# This work is licensed under the MIT license.
# Copyright (c) 2013-2023 OpenMV LLC. All rights reserved.
# https://github.com/openmv/openmv/blob/master/LICENSE
#
# MJPEG Video Recording Example
#
# Note: You will need an SD card to run this demo.
#
# You can use your OpenMV Cam to record mjpeg files. You can either feed the
# recorder object JPEG frames or RGB565/Grayscale frames. Once you've finished
# recording a Mjpeg file you can use VLC to play it. If you are on Ubuntu then
# the built-in video player will work too.

import sensor
import time
import mjpeg
```

```
import machine

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

led = machine.LED("LED_RED")

led.on()
m = mjpeg.Mjpeg("example.mjpeg")

clock = time.clock() # Create a clock object to track the FPS.
for i in range(200):
    clock.tick()
    m.write(sensor.snapshot())
    print(clock.fps())

m.close()
led.off()

raise (Exception("Please reset the camera to see the new file."))
```

这段代码是一个基于 OpenMV 的示例，用于录制 MJPEG 格式的视频并保存为文件。以下是代码的详细分析：

3.3.5.1 导入库

```
import sensor
import time
import mjpeg
import machine
```

- sensor: 控制摄像头传感器。
- time: 提供计时功能（如 FPS 计算）。
- mjpeg: 用于 MJPEG 文件录制。
- machine: 控制硬件（如 LED）。

3.3.5.2 初始化传感器

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

- `sensor.reset()`: 初始化摄像头传感器。
- `sensor.set_pixformat(sensor.RGB565)`: 设置图像的像素格式为 RGB565（可选为 GRAYSCALE）。
- `sensor.set_framesize(sensor.QVGA)`: 设置图像的帧尺寸为 QVGA（320x240 像素）。
- `sensor.skip_frames(time=2000)`: 等待 2 秒，使摄像头完成配置。

3.3.5.3 LED 和 MJPEG 文件初始化

```
led = machine.LED("LED_RED")
```

```
led.on()
```

```
m = mjpeg.Mjpeg("example.mjpeg")
```

- `machine.LED("LED_RED")`: 控制 OpenMV 板上的红色 LED。
- `led.on()`: 打开红色 LED，表示录制开始。
- `mjpeg.Mjpeg("example.mjpeg")`: 创建一个 MJPEG 文件对象，用于将捕获的帧写入文件。

3.3.5.4 LED 和 MJPEG 文件初始化

```
clock = time.clock() # Create a clock object to track the FPS.
```

```
for i in range(200):
```

```
    clock.tick()
```

```
    m.write(sensor.snapshot())
```

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

- `time.clock()`: 用于计算每帧的时间间隔，从而计算帧率 (FPS)。
- `sensor.snapshot()`: 捕获一帧图像。
- `m.write()`: 将捕获的帧写入 MJPEG 文件。
- `print(clock.fps())`: 打印当前帧率。

3.3.5.5 LED 和 MJPEG 文件初始化

```
m.close()
```

```
led.off()
```

- `m.close()`: 关闭文件，确保所有数据写入到存储介质中。
- `led.off()`: 关闭红色 LED，指示录制结束。

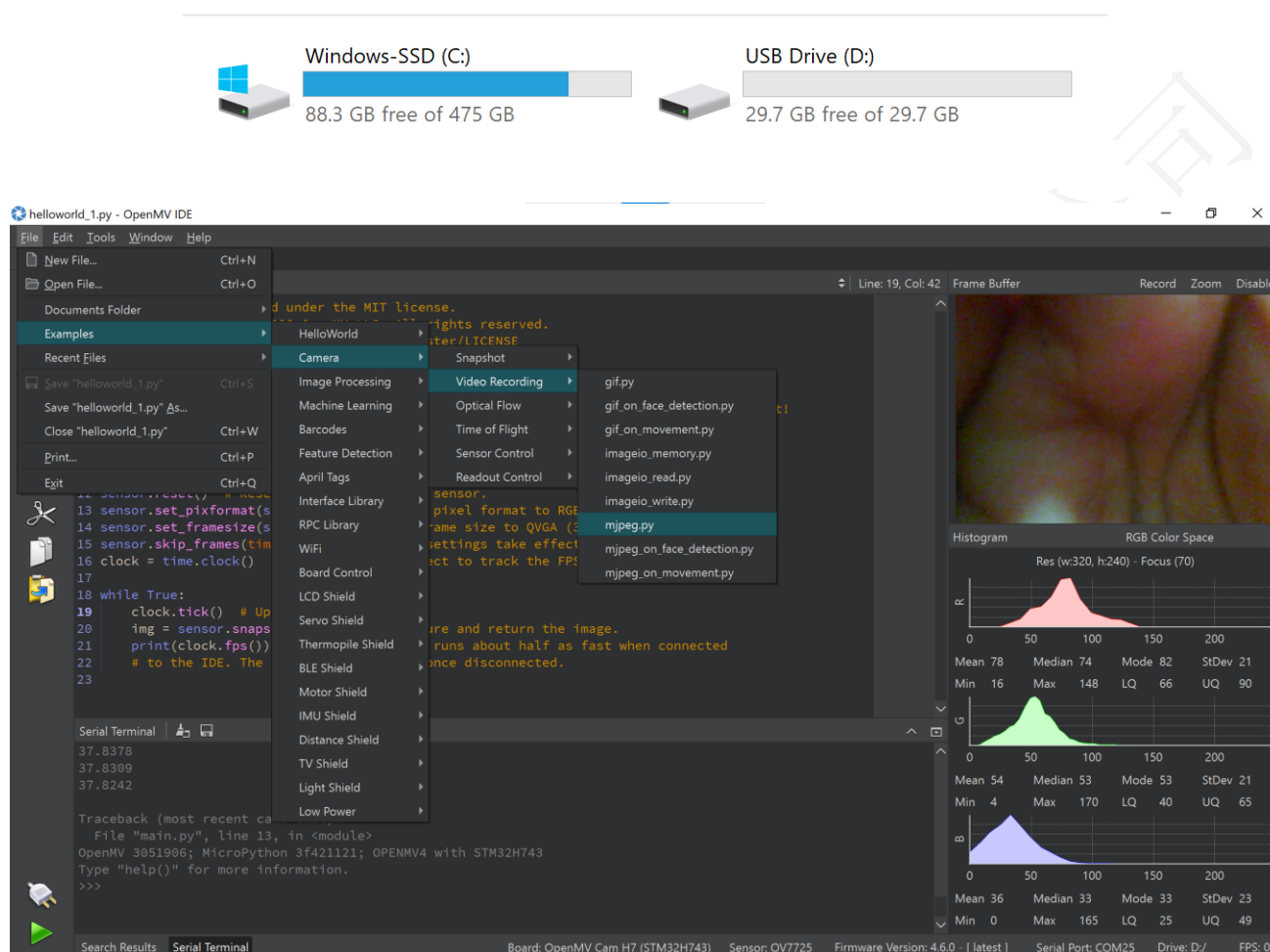
3.3.5.5 异常提示

```
raise (Exception("Please reset the camera to see the new file."))
```

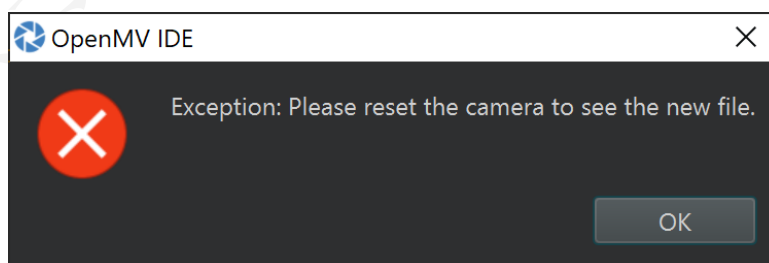
- 提示用户需要重启设备: 在 OpenMV 中，文件写入完成后通常需要重启设备才能访问新文件。

3.3.5.6 演示

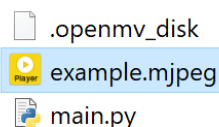
这个演示需要录制一段视频，视频需要存储在 SD 卡中。建议您先插入不大于 32GB 的 SD 卡，然后再接入电脑。如果成功，会生成一个 USB 存储设备，这个存储设备的容量应为 SD 卡的容量。



按照上述路径加载程序后，开始运行这段代码，直至提示



这时我们重新插拔一下 FKMV 设备，在 SD 卡中可以发现多了一个名为“example”的视频文件，双击打开即可观看刚才拍摄的视频。



我们在源码的基础上稍作修改，就可以一直录制下去，可作为监控使用。

改动说明

- 文件名动态生成:使用 `file_index` 变量为每个文件生成唯一的文件名，例如 `example_1.mjpeg`、`example_2.mjpeg`。每次录制 200 帧后，自动切换到下一个文件。
- 循环录制: 使用无限循环 `while True` 实现持续录制。每次录制 200 帧，关闭当前文件后重新开始新文件。
- LED 提示:红色 LED 始终保持亮起状态，表示录制正在进行。
- 文件大小控制:通过限制每个文件录制 200 帧，避免生成单个过大的 MJPEG 文件。

```
import sensor
import time
import mjpeg
import machine

# 初始化传感器
sensor.reset() # 重置并初始化传感器
sensor.set_pixformat(sensor.RGB565) # 设置像素格式为 RGB565 (或 GRAYSCALE)
sensor.set_framesize(sensor.QVGA) # 设置帧尺寸为 QVGA (320x240)
sensor.skip_frames(time=2000) # 等待 2 秒以确保设置生效

# 控制 LED
led = machine.LED("LED_RED")

# 持续录制逻辑
file_index = 1 # 文件编号
led.on() # 打开 LED 指示开始录制

while True:
    # 为每个新文件生成唯一的文件名
    file_name = "example_{}.mjpeg".format(file_index)
    print("Recording to:", file_name)

    # 初始化 MJPEG 文件对象
    m = mjpeg.Mjpeg(file_name)
    clock = time.clock() # 创建时钟对象以跟踪 FPS
```

```
# 每个文件录制 200 帧
for _ in range(200):
    clock.tick()
    m.write(sensor.snapshot()) # 捕获并写入一帧
    print("FPS:", clock.fps())

# 关闭当前文件并准备下一个
m.close()
file_index += 1 # 文件编号递增

# 注意: 此代码没有关闭 LED, 表示设备一直处于录制状态
```

联系我们

联系人: 付坤

联系方式: fukunrcts@163.com

