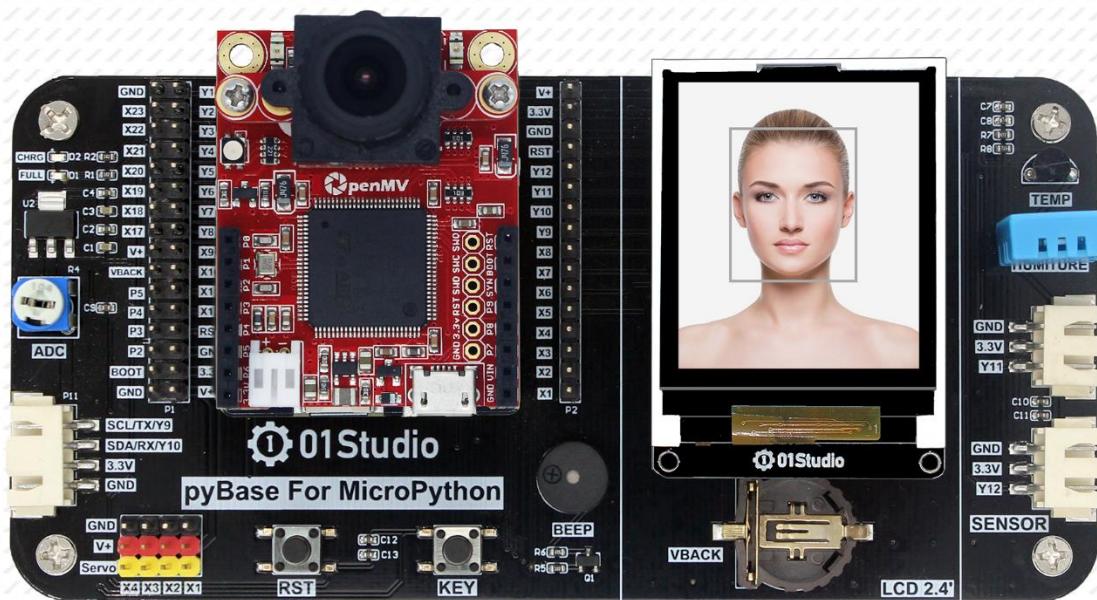


# MicroPython 从0到1

用python做嵌入式编程

(基于OpenMV4平台)

01Studio团队 编著

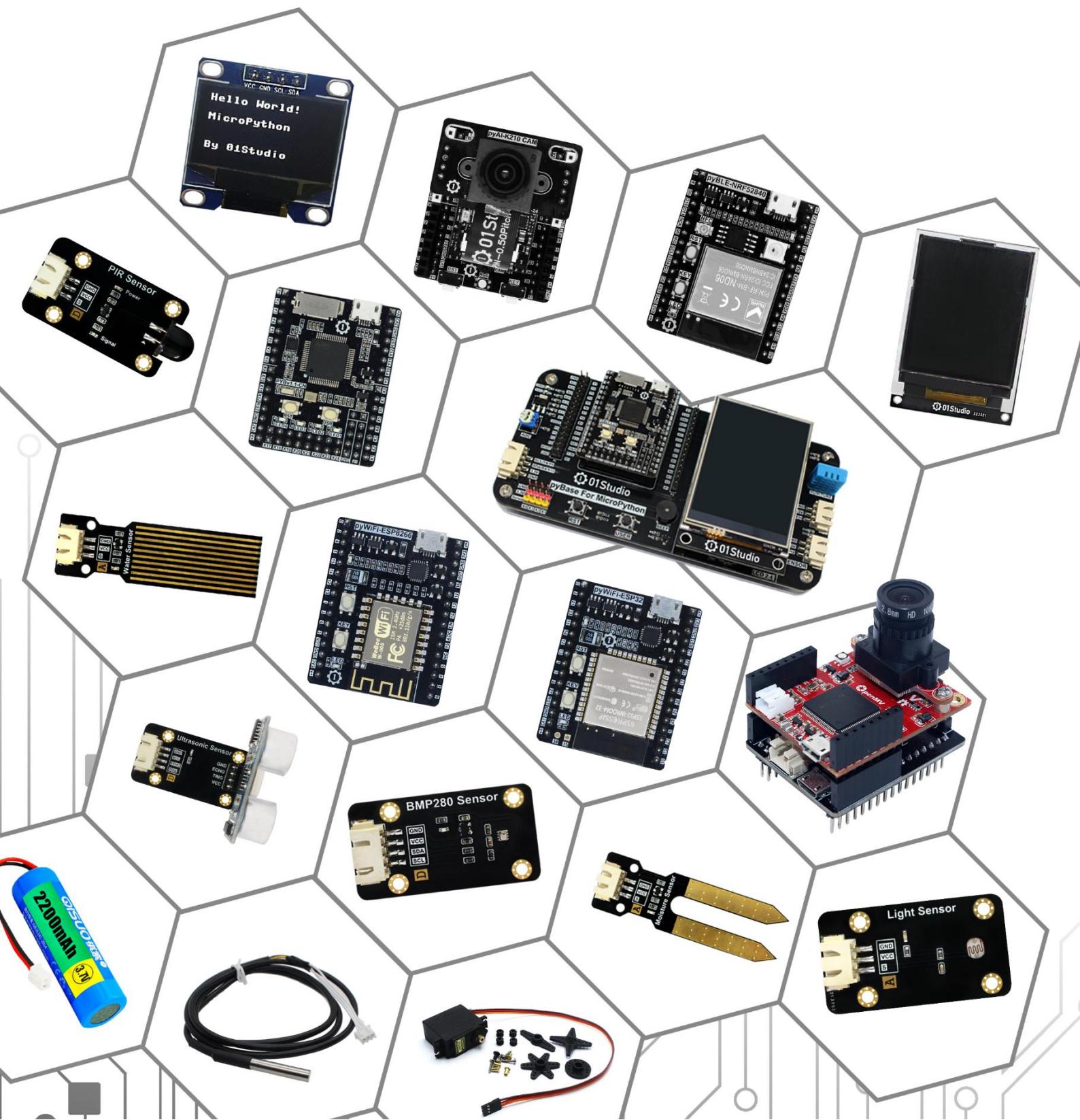


 01Studio

-让编程变得简单有趣-

# MicroPython

## 产品家族



# 前言

## 为什么学习 MicroPython?

单片机嵌入式编程经历了汇编、C 语言的发展历程，可以说是一次编程革命，其背后的原因是单片机的速度越来越快，集成度越来越高。而这一趋势并没有停止，摩尔定律仍然适用。在未来，单片机上很可能直接跑机器语言。

在 2014 年，MicroPython 在英国诞生了，对于电子爱好者来说无疑拉开了新时代的序幕，用 python 这个每年用户量不断增长的编程语言来开发嵌入式，加上无数开源的函数模块，让嵌入式开发变得从未如此的简单。

MicroPython 致力于兼容 Python。因此，我们在学习完 MicroPython 后除了可以开发有趣的电子产品外，还可以继续深入使用 Python 语言去开发后台、人工智能等领域。

## 为什么要写《MicroPython 从 0 到 1》教程？

对于初学者，常常需要浪费大量时间来搭建开发环境和安装应用软件，以及需要从不同渠道寻找开发资料。MicroPython 作为新兴的东西，市面上的资料更是参差不齐，被搞得头昏目眩。

“让编程变得简单有趣”作为我们 01Studio 团队的使命，我们一直在寻找最优的学习方法。力求以最简单易懂的方式来讲述 MicroPython 的学习方法，从开发环境快速建立、基础实验、传感器实验到项目进阶实验。《MicroPython 从 0 到 1》诞生了。相比于传统的出版图书，我们会以更平易近人的口吻讲解实验，配上精美的插图，每一行代码都是自己的亲身经历，目的就是让大家更好的入门 MicroPython，将精力放在编程和开发中去。

## 为什么要打造 MicroPython 学习套件？

MicroPython 在中国是一个新兴的东西，前途无限，但是网上的学习模块套件参差不齐，大多是复制官方开源的开发板来设计，用过的就知道，外国的电路设计跟国内的风格很不同，甚至常常让初学者钻牛角尖。为此，01Studio 团队特意打造的中国风的 MicroPython 开发套件，《MicroPython 从 0 到 1》上的例程也是基于此板开发的，每个例程都能直接跑起。通过一系列系统学习，你甚至可

以用它来完成你的比赛或项目。

### 为什么要打造 01Studio 社区论坛？

早在数年前我们打造了 WeBee 品牌，专注物联网开发，到现在已经服务了数万名学生、教师和工程师等相关人员。早期依靠着群来维护，但随着开发者的数量增加，我们发现仅依靠群或者技术支持人员已经不能满足需求。

社区论坛是很好的学习交流地方，在这里你可以学习到前人的经验，可以通过搜索内容来解决问题。为什么全世界很火的开源硬件都是由外国人做起来，我们认为很重要的一个点是源于开源和分享精神。大部分开发者都会想着从论坛或网站解决自身问题而不愿意奉献，而我们希望 01Studio 社区是一个大家乐于发表文章和分享心得的地方。我们始终始终坚持开源原则，包括书籍内容、所有例程代码和部分硬件模块的开源。

期待你加入 01Studio 社区大家庭，成为我们的一份子，一起成长。

【社区链接: [www.01Studio.org](http://www.01Studio.org)】

【微信公众号: 01Studio 社区】

01Studio

2019. 4 于深圳

## 版本说明

版本	日期	作者	更新内容
v1.0	2021-2-5	Jackey	1. 第1版（重构）

## 版权声明

《MicroPython 从 0 到 1》由 **01Studio** 团队打造，已于深圳市版权局注册备案，任何单位或个人引用相关文字、图片或相关内容请注明出处【**01Studio**】，否则我们将保留追究相关法律责任的权利。

## 目录

### 第一部分 MicroPython 基础知识 ..... 8

第 1 章 MicroPython 简介 .....	8
1.1 MicroPython 是什么 .....	8
1.2 MicroPython 支持的微控制器平台 .....	9
1.3 MicroPython 相关学习资料 .....	11
1.3.1 01Studio 社区 .....	11
1.3.2 MicroPython 文档（中文） .....	12
1.3.3 MicroPython 官方网站 .....	13
1.4 MicroPython 开发套件介绍 .....	14
1.4.1 OpenMV4 平台 .....	15
1.4.2 pyBase .....	21
1.4.3 IOT/通讯模块 .....	23
1.4.4 拓展配件 .....	25
第 2 章 Python 基础知识 .....	31
2.1 原始数据类型和运算符 .....	31
2.2 变量和集合 .....	36
2.3 流程控制和迭代器 .....	42
2.4 函数 .....	46
2.5 类 .....	49
2.6 模块 .....	51
2.7 高级用法 .....	52

### 第二部分 基于 OpenMV4 平台 ..... 54

第 3 章 开发环境快速建立 .....	55
3.1 基于 Windows .....	55
3.1.1 安装开发软件 OpenMV IDE .....	55
3.1.2 开发套件使用 .....	57
3.2 基于 Mac OS .....	68
3.2.1 安装开发软件 OpenMV IDE .....	68
3.3 基于 Linux（树莓派） .....	69
第 4 章 基础实验 .....	70
4.1 点亮第一个 LED .....	71
4.2 流水灯 .....	76
4.3 GPIO（按键） .....	81
4.4 外部中断 .....	86
4.5 定时器 .....	91
4.6 I2C 总线（OLED 显示屏） .....	94
4.7 RTC 实时时钟 .....	101
4.8 ADC .....	107

4.9 DAC.....	113
<b>第 5 章 机器视觉 .....</b>	<b>121</b>
5.1 摄像头应用 .....	121
5.2 画图 .....	127
5.3 特征检测 .....	133
5.3.1 边缘检测 .....	134
5.3.2 圆形识别 .....	138
5.3.3 线段识别 .....	142
5.3.4 直线识别 .....	146
5.3.5 矩形识别 .....	150
5.3.6 特征点识别 .....	154
5.3.7 快速线性回归（巡线） .....	161
5.4 颜色追踪 .....	167
5.4.1 RGB565 彩色自动追踪.....	168
5.4.2 机器人巡线（实线巡线） .....	175
5.4.3 单个颜色识别 .....	183
5.5 人脸检测 .....	189
5.5.1 人脸检测 .....	190
5.5.2 人脸追踪 .....	196
5.6 眼球追踪 .....	205
5.6.1 眼球检测 .....	206
5.6.2 瞳孔追踪 .....	211
5.7 图片拍摄 .....	217
5.7.1 普通拍摄 .....	218
5.7.2 人脸检测拍摄 .....	222
5.7.3 移动物体抓拍 .....	227
5.8 视频录制 .....	233
5.8.1 普通 GIF 视频录制 .....	234
5.8.2 人脸检测 GIF 视频录制 .....	239
5.8.3 移动物体 GIF 视频录制 .....	244
5.9 图像滤波 .....	250
5.9.1 颜色二值化滤波 .....	251
5.9.2 锐化滤波 .....	256
5.9.3 图像翻转 .....	260
<b>第 6 章 拓展模块 .....</b>	<b>265</b>
6.1 LCD 显示屏 .....	266
6.2 舵机 .....	270
6.3 多路舵机模块 .....	280
6.4 WiFi 模块 .....	285
6.4.1 连接无线路由器 .....	286
6.4.2 Socket 通信 .....	290
6.4.3 MQTT 通信.....	301
6.4.4 无线视频传输 .....	314
6.5 摄像头模块 .....	317

6.5.1 MT9V034 全局快门.....	318
6.5.2 FLIR Lepton 红外热成像.....	324
6.6 摄像头镜头 .....	329
6.6.1 广角镜头 .....	330
6.6.2 长焦镜头 .....	332
6.6.3 无畸变镜头 .....	334
6.6.4 手动调焦镜头 .....	336
第 7 章 项目应用 .....	338
7.1 照相机 .....	338

# 第一部分 MicroPython 基础知识

## 第1章 MicroPython 简介

### 1.1 MicroPython 是什么

第一次接触 MicroPython 的时候，我就想这是个什么玩意，从字面意思来看，就是 Micro 加 Python。难道是阉割版的 Python？阉割后可以在微控制器上面跑？当然你也可以这么理解，我们来看看官方的说明：

“MicroPython 是 Python 3 编程语言的精简高效实现，包括 Python 标准库的一小部分，并且经过优化，可以在 Microcontrollers（微控制器）和有限的环境中运行。

MicroPython 包含许多高级功能，如交互式提示，任意精度整数，闭包，列表理解，生成器，异常处理等。然而它非常紧凑，可以在 256k 的代码空间和 16k 的 RAM 内运行。

MicroPython 旨在尽可能与普通 Python 兼容，以便您轻松地将代码从电脑传输到微控制器或者嵌入式系统。”

看完官方说明后，大家应该有所了解，Micropython 是指在微控制器上使用 Python 语言进行编程，学习过单片机和嵌入式开发的小伙伴应该都知道早期的单片机使用汇编语言来编程，随着微处理器的发展，后来逐步被 C 所取代，现在的微处理器集成度越来越高了，那么我们现在可以使用 Python 语言来开发了。

Python 的强大之处是封装了大量的库，开发者直接调用库函数则可以高效地完成大量复杂的开发工作。MicroPython 保留了这一特性，常用功能都封装到库中了，以及一些常用的传感器和组件都编写了专门的驱动，通过调用相关函数，就可以直接控制 LED、按键、伺服电机、PWM、AD/DA、UART、SPI、IIC 以及 DS18B20 温度传感器等等。以往需要花费数天编写才能实现的硬件功能代码，现在基于 MicroPython 开发只要十几分钟甚至几行代码就可以解决。真可谓：“人生苦短，我用 Python 和 MicroPython”。

## 1.2 MicroPython 支持的微控制器平台

MicroPython 到目前为止已经可以在多种嵌入式硬件平台上运行：STM32、ESP8266、ESP32、CC3200、K210 等等。由于项目的开源特性，很多开发者在尝试将其移植到更多平台上。

MicroPython 最早支持的硬件平台是 STM32，开发板名称叫 pyboard。使用的芯片型号是：STM32F405RGT6，该芯片具备 1MB flash 和 196k SRAM，168MHz 主频。

除此之外，上海乐鑫的 WiFi 芯片 ESP8266/ESP32 也非常成熟。用户使用 MicroPython 可以快速开发物联网相关应用，实现 WiFi 无线连接。

除此之外，不少优秀的开源项目也是基于 MicroPython 衍生出来的，如机器视觉界 Arduino 之称的 OpenMV、人工智能芯片 K210 等。随着社区的日益成熟，MicroPython 必定将嵌入式编程推向新的高度。本书主要是围绕 OpenMV4 平台来进行编写。

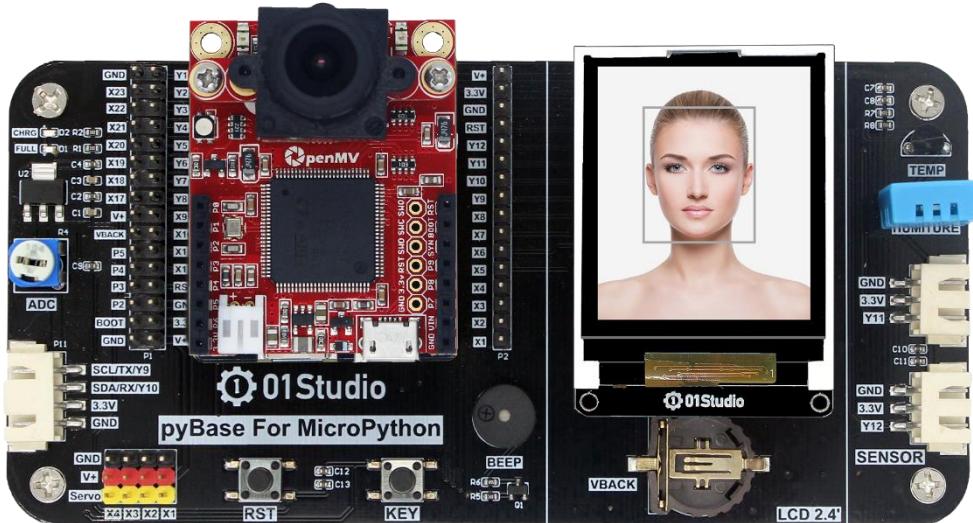


图 1-1 pyAI-OpenMV4 开发套件

以下是 01Studio 其它 micropython 开发平台：

<b>STM32 平台</b>	<b>ESP8266 平台</b>	<b>ESP32 平台</b>	<b>NFR52840 平台</b>	<b>OpenMV4 平台</b>	<b>K210 平台</b>
pyBoard v1.1-CN	pyWiFi-ESP8266	pyWIFI-ESP32	pyBLE-NRF52840	pyAI-OpenMV4	pyAI-K210
					
					

图 1-2 01Studio MicroPython 系列开发平台

## 1.3 MicroPython 相关学习资料

### 1.3.1 01Studio 社区

【社区网址: [www.01Studio.org](http://www.01Studio.org)】

01Studio 社区是 MicroPython 开发者交流的社区论坛, 我们以极简风格设计, 开发者在学习过程中遇到问题可以到论坛搜索或者发帖提问, 以提高学习效率。01Studio 团队也会在社区定期发布学习资源。

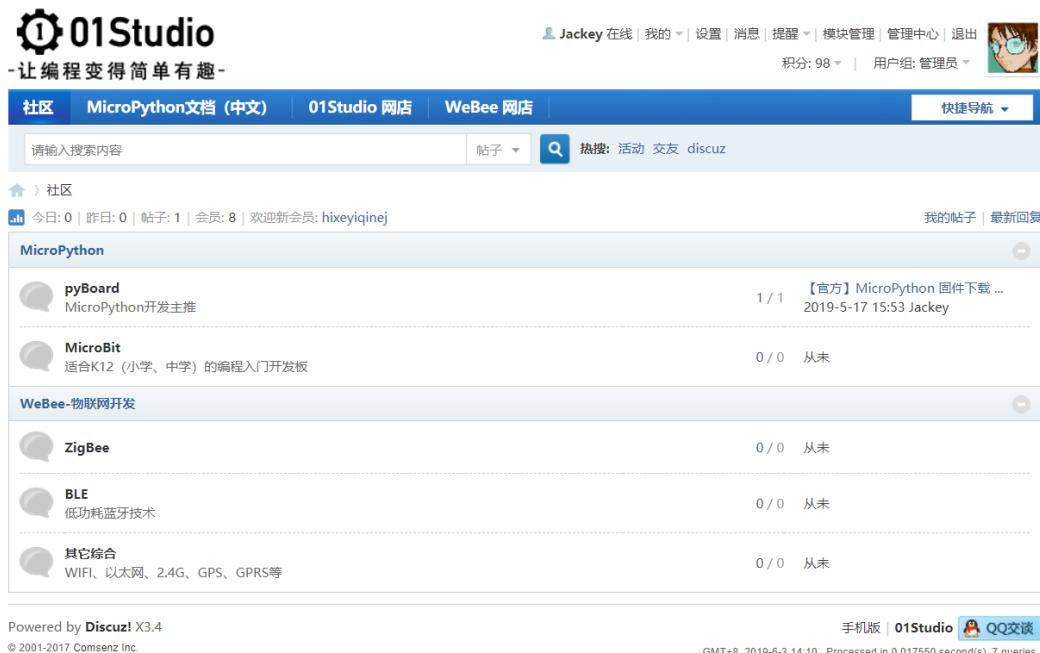


图 1-3 01Studio 社区

### 1.3.2 MicroPython 文档（中文）

限于篇幅，本教程部分实验只介绍关键的函数和模块应用，如果在学习过程中希望深入了解 MicroPython 函数和模块，请查阅：

MicroPython 文档（中文） 【网址：[docs.micropython.01studio.org](https://docs.micropython.01studio.org)】



图 1-4 文档在社区的位置

该文档是 01Studio 团队在维护的官方文档中文翻译版，我们力求保持与官方文档保持实时同步，降低开发者的学习门槛。



图 1-5 MicroPython 文档 (中文)

### 1.3.3 MicroPython 官方网站

【网址：[www.micropython.org](http://www.micropython.org)】

英文版官网有官方文档(DOCS)和英文论坛，适合比较英语比较好的小伙伴。

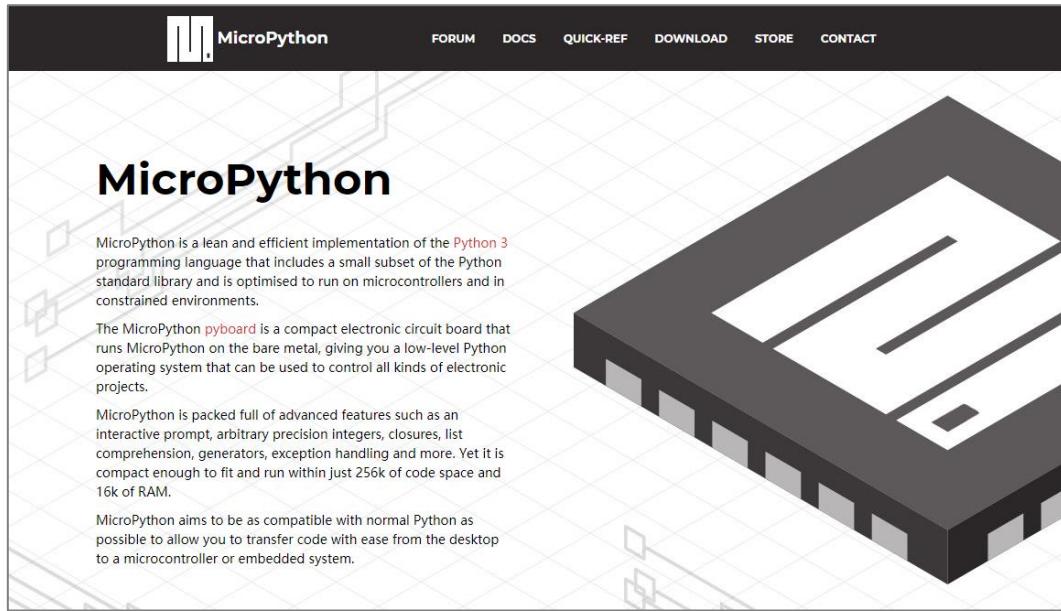


图 1-6 MicroPython 官网

## 1.4 MicroPython 开发套件介绍

为了让广大电子爱好者更方便地学习 MicroPython，01Studio 团队打造了一系列本土化的高性价比学习套件和周边模块。当前已支持 STM32 平台、ESP8266 平台、ESP32 平台、NRF52840 平台、OpenMV 平台、K210 平台以及周边传感器模块和配件外设。我们的开发平台采用核心板+底板形式设计，保留了 MicroPython 官方的兼容性，同时使开发者可以更好的连接外设，进行更多扩展性实验。

《MicroPython 从 0 到 1》上的例程也是基于本学习平台开发的，我们承诺资源会不断更新，保证所有代码程序能直接跑起。毫不夸张地说：你甚至可以将本教材的例程和实践应用在自己的产品研发和项目开发中去。

## 1.4.1 OpenMV4 平台

### 1.4.1.1 pyAI-OpenMV4

pyAI-OpenMV4 主控芯片使用 STM32H743VIT6，所以也是可以说是 STM32 平台，实现原理跟 pyBoard 有点类似。这是 O1Studio 在保留官方 OpenMV4 不变的基础上增加转接板实现，具体改进如下：

- (1) 兼容 pyBoard 接口；
- (2) 板载锂电池输入接口和充电电路；
- (3) 引出复位和功能按键，布局合理，丝印清晰。



图 1-7 pyAI-OpenMV4



图 1-8 pyAI-OpenMV4 转接板

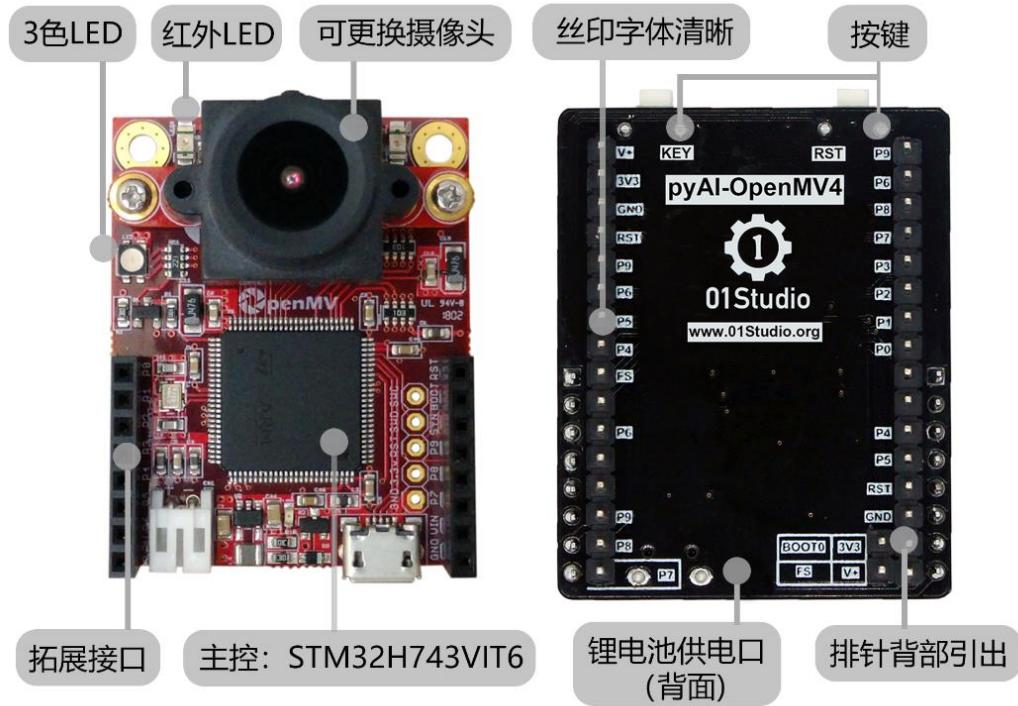


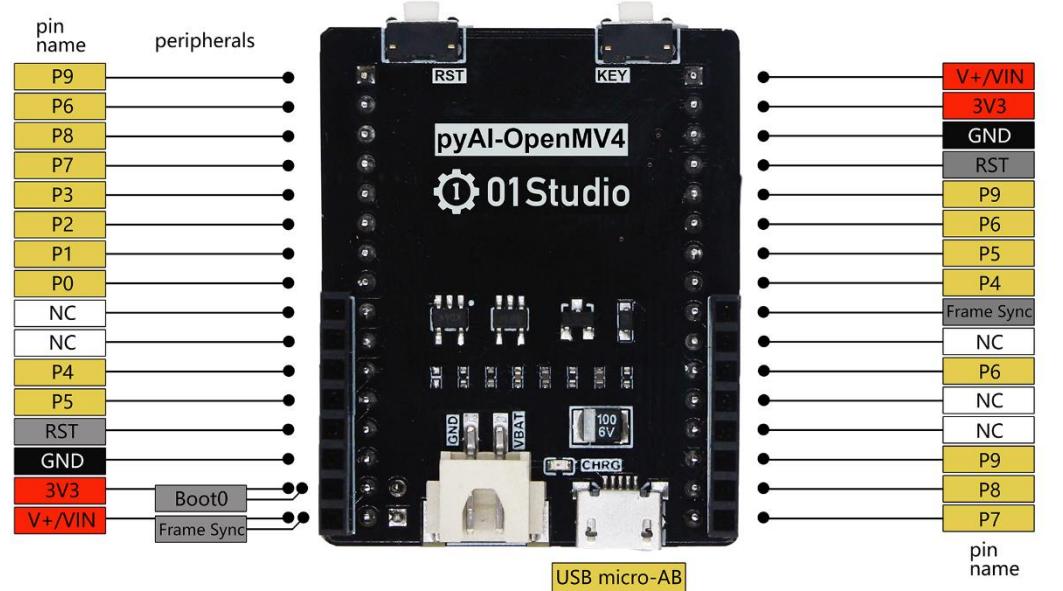
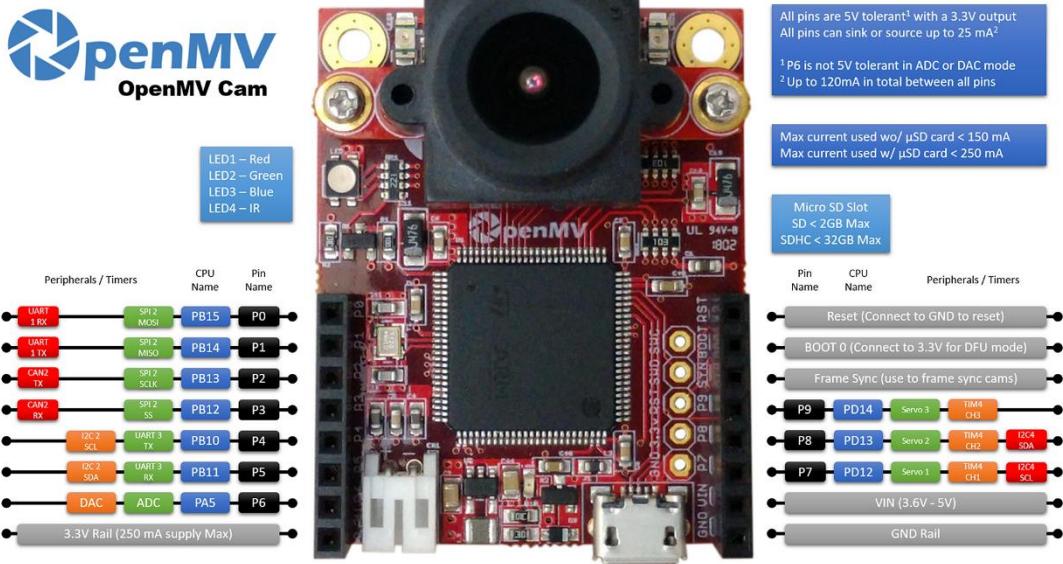
图 1-9 pyAI-OpenMV4 (正反面)

### 功能参数

V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 500mA
VBAT	锂电池输入（板载 FET 保护电路和充电电路）
LED	LED1-红灯，LED2-绿灯，LED3-蓝灯，LED4-红外
按键	RST-复位，KEY-引脚 “P9”
SD 卡	最大支持 32GB (SDHC)
BOOT0	如果连接到 3V3，复位后进入 DFU 模式

表 1-1 pyAI-OpenMV4 参数

# OpenMV Cam H7 - OV7725



## pyAI-OpenMV4

MicroPython

 [www.01Studio.org](http://www.01Studio.org)

V+: 3.6-6V输入 (当由USB供电时电压为5V, 建议使用USB口供电)

3V3: 3.3V输出, 最大电流500mA

VBAT: 锂电池输入 (板载FET保护电路和充电电路)

LED: LED1-红灯, LED2-绿灯, LED3-蓝灯, LED4-红外

按键: RST-复位, KEY-引脚" P9"

SD卡: 最大支持32GB(SDHC)

BOOT0: 如果连接到3V3, 复位后进入DFU模式

图 1-10 pyAI-OpenMV4 引脚图

### 1.4.1.2 pyAI-OpenMV4 Plus

pyAI-OpenMV4 Plus 相对于 pyAI-OpenMV4 性能有所提升，主控芯片使用 STM32H743IJK6，外挂 32MB SDRAM 和 32MB Flash。

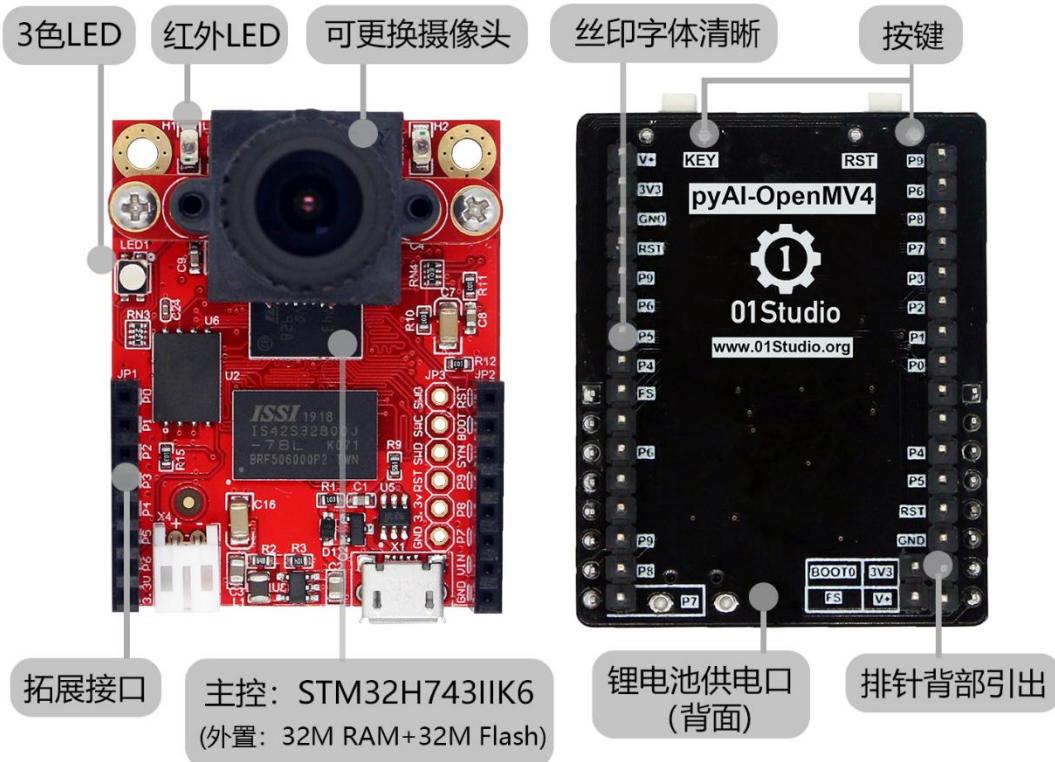


图 1-11 pyAI-OpenMV4 Plus

VS	pyAI-OpenMV4	PyAI-OpenMV4 Plus
标配摄像头	ov7725(30万像素)	ov5640(500万像素)
RAM	1MB	1MB+32MB SDRAM
FLASH	2M	2M+32MB QSPI Flash

图 1-12 OpenMV4 与 OpenMV4 Plus 主要参数对比

### 1.4.1.3 pyAI-OpenMV4 开发套件

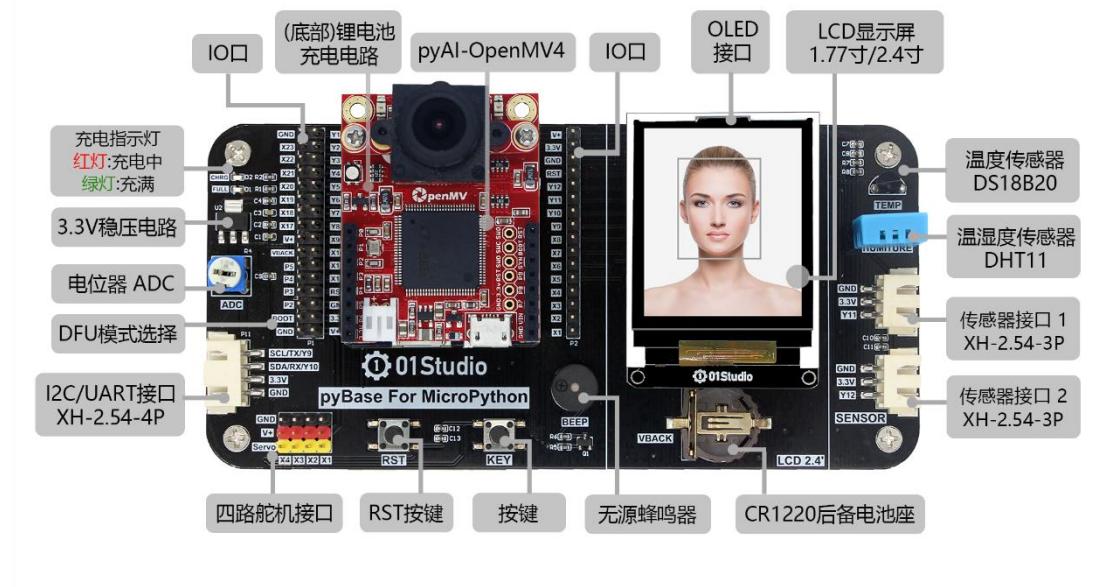


图 1-13 pyAI-OpenMV4 开发套件

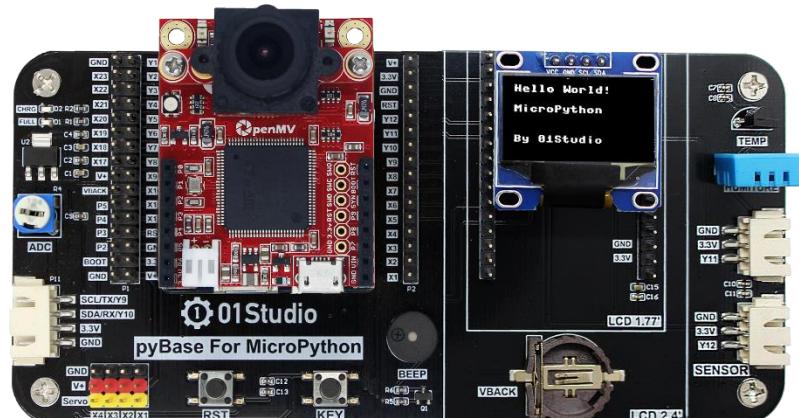


图 1-14 配 OLED 显示屏

主要配置	
核心板	pyAI-OpenMV4
底板	pyBase
显示屏	1.77 寸 LCD / 0.9 寸 OLED 显示屏

表 1-2

### 1.4.2 pyBase

pyBase 是 01Studio 针对各 micropython 开发平台量身定制的底板，可以使用它可以做更多的 MicroPython 实验，pyBase 同时设计了外设接口，扩展性非常强。以下是详细的功能说明：

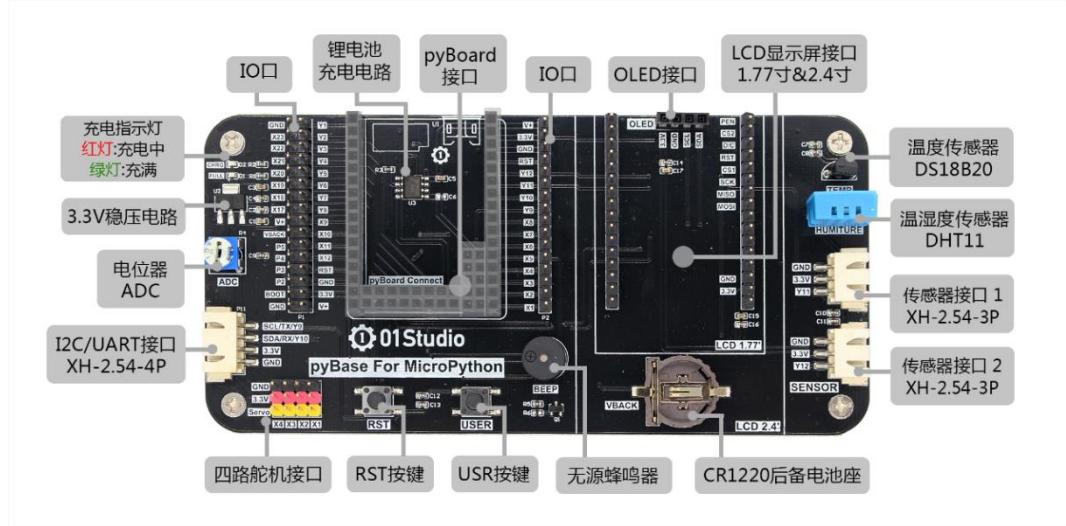


图 1-15 pyBase 功能说明

功能参数	
pyboard 接口	兼容 pyboard v1.1-CN 以及官方的 pyboard
2.54mm 排针	引出 pyboard 全部接口
按键 2 个	引出 RST 和 USR 按键
电位器	ADC 输入
无源蜂鸣器	DAC 输出
纽扣电池座	安装 CR1220 纽扣电池
Servo 接口	舵机接口 X1-X4 (连接舵机需要外接隔离电路)
OLED 接口	4P/0.96 寸/I2C/OLED 显示屏
1.77 寸 LCD 接口	适用于 01Studio 1.77 寸 LCD 显示屏
2.4 寸 LCD 接口	适用于 01Studio 2.4 寸 LCD 显示屏 (带电阻触摸)
温度传感器	DS18B20
温湿度传感器	DHT11
Sensor 接口 1	3P 防呆接口，用于外接传感器

Sensor 接口 1	3P 防呆接口，用于外接传感器
通讯接口	4P 防呆接口，用于外接 UART/I2C 设备
锂电池充电电路	对接入的锂电池进行充电（红灯->充电，绿灯->充满）

表 1-3

### 1.4.3 IOT/通讯模块

pyIOT 开源项目由 01Studio 发起，旨在为市面上成熟的串口物联网模块开发 MicroPython 库，让用户可以使用 python 编程快速实现各类物联网相关应用。

#### 1.4.3.1 pyIOT-BLE TLS01

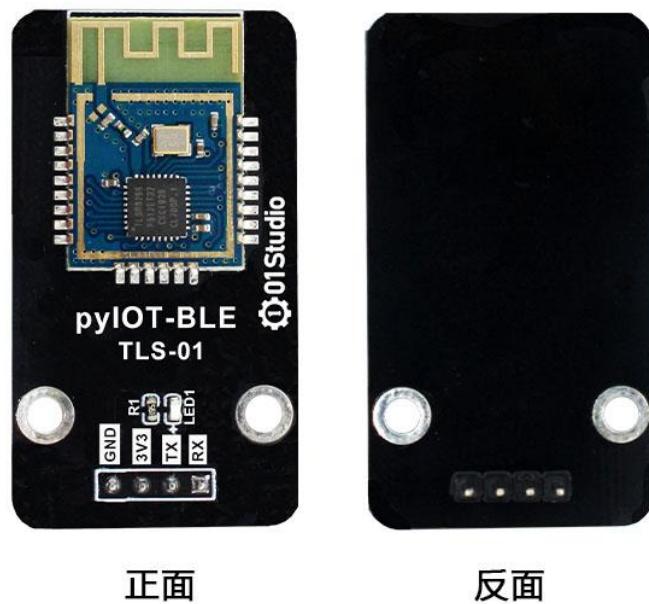


图 1-16

功能参数	
蓝牙主控	TLSR8266
控制方式	UART (串口)
蓝牙版本	BLE 4.0 (从机)
功耗	工作电流: <8.8mA, 广播电流: <1mA
引脚	GND, 3.3V, TX, RX
模块尺寸	4.5*2.5cm

表 1-4

### 1.4.3.2 pyIOT-LORA E22

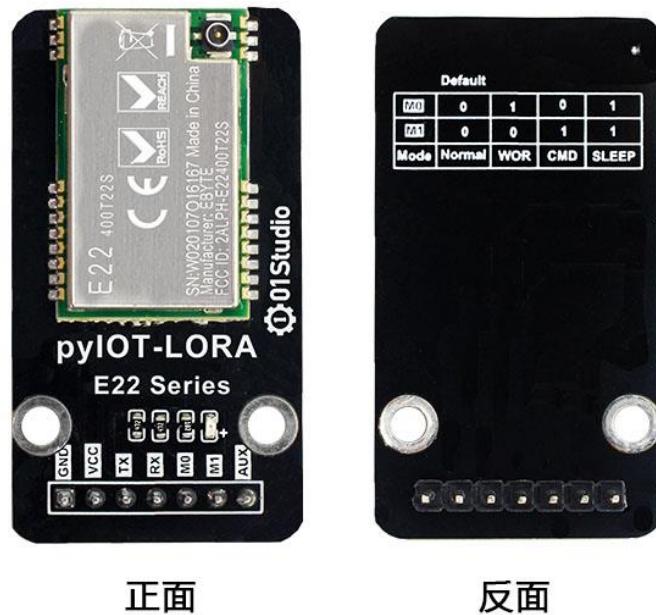


图 1-17

功能参数	
工作频段	410~493MHz (433M)
射频芯片	SX1268
发射功率	22dBm
通讯距离	最大: 5km
通讯接口	UART (串口)
天线	IPEX
发射电流	110mA
供电电压	2.3-5.2V
工作温度	-40 °C - 85 °C
波特率	1200 – 115200 bps (默认 9600)
空中速率	0.3k – 42.5kbps
接收长度	1024 字节 (自动分包)
模块尺寸	4.2*2.5cm

表 1-5

### 1.4.3.3 pyIOT-GPS

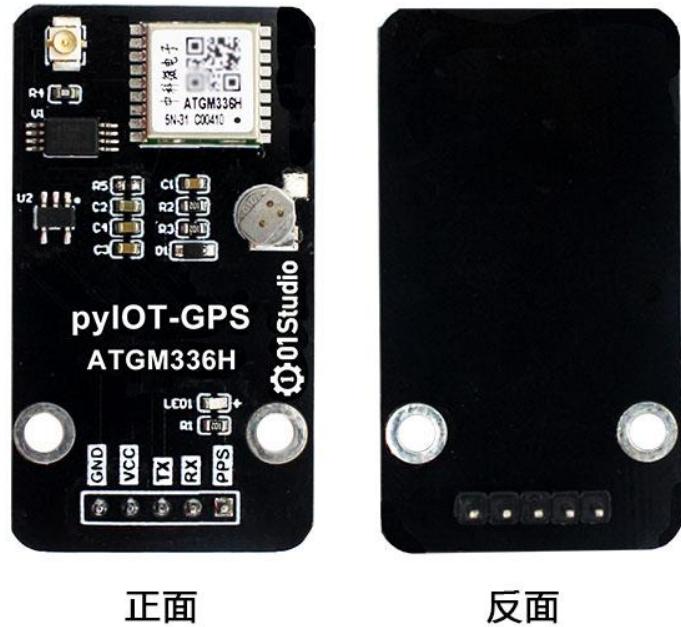


图 1-18

功能参数	
主控芯片	ATGM336H-5N
卫星信号	GPS/BDS/GLONASS
波特率	9600bps
工作电压	3.3V-5V
串口电平	3.3V 或 5V (自适应)
定位精度	2.5m (开阔地点)
功耗	工作: <25mA, 待机: <10uA (@3.3V)
模块尺寸	4.2*2.5cm

表 1-6

### 1.4.4 拓展配件

#### 1.4.4.1 OLED 显示屏



图 1-19 OLED 显示屏

功能参数	
供电电压	3.3V
屏幕尺寸	0.9 寸
颜色参数	黑底白字
通讯方式	I2C 总线
接口定义	2.54mm 排针 (4Pin) 【VCC、GND、SCL、SDA】
整体尺寸	2.8*2.8cm

表 1-7

#### 1.4.4.2 1.77 寸 LCD 显示屏

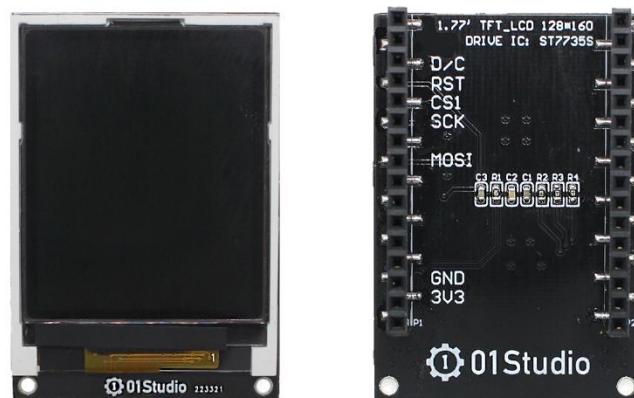


图 1-20 1.77 寸 LCD 显示屏

功能参数	
供电电压	3.3V
屏幕尺寸	1.77 寸
分辨率	128*160
颜色参数	TFT 彩色
驱动芯片	ST7735S
通讯方式	SPI 总线
接口定义	2.54mm 排母（兼容 pyboard v1.1 接口）
整体尺寸	5.0*3.5 cm

表 1-8

#### 1.4.4.3 舵机

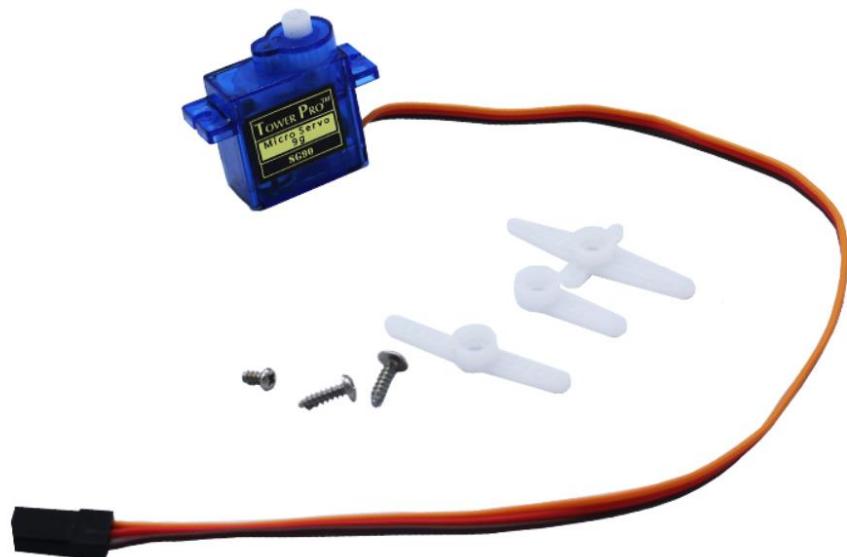


图 1-21 舵机

功能参数	
尺寸	32*30*1.1 mm
重量	15g
扭力	1.6kg/cm
接口	XH2.54 接口 (3Pin) 【GND (黑)、VCC (红)、Single (橙)】
工作电压	3.5-6V
工作温度	-30°C-60 °C
转动角度	180° 和 360° 连续旋转。
应用场景	固定翼、直升机 KT、机器人、机械臂、航模等

表 1-9

#### 1.4.4.4 RGB 灯带

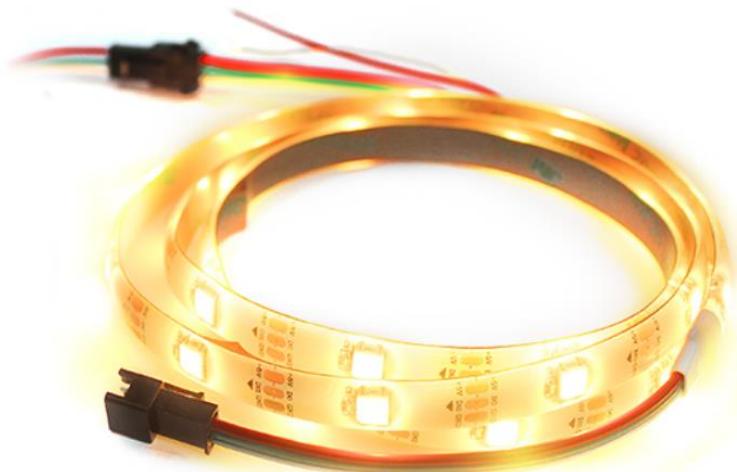


图 1-22 RGB 灯带

功能参数	
灯带长度	1 米
灯珠数量	30
颜色	RGB 全彩
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
驱动 IC	WS2812B
供电电压	3.3-5V
功率	每颗灯珠最高 0.3W
应用场景	流水灯/呼吸灯/节日灯饰布置等

表 1-10

#### 1.4.4.5 USB 转串口 TTL

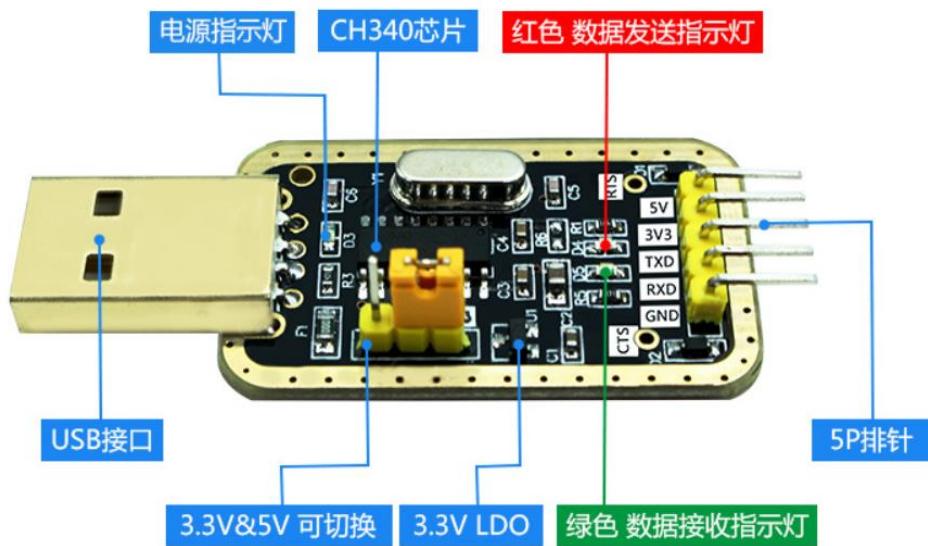


图 1-23 USB 转串口 TTL

功能参数	
驱动芯片	CH340
对外供电	5V 和 3.3V
支持电平	3.3V 和 5V 可切换
引脚接口	2.54MM 排针

表 1-11

## 第2章 Python 基础知识

由于 python 语言是学习 micropython 的基础，为了照顾没有 python 基础的朋友，我们一直在思考应该给大家提供怎么样的教程。Python 相关的书籍和学习资料相信大家能在网上找到不少，所以这里整理给大家的经典 python3 快速学习资料，你甚至可以使用它来当作 python 字典查阅！

【原著：Louie Dinh，翻译：Geoff Liu】

```
# 用井字符开头的是单行注释
```

```
""" 多行字符串用三个引号  
包裹，也常被用来做多  
行注释
```

### 2.1 原始数据类型和运算符

```
#####
## 1. 原始数据类型和运算符
#####

# 整数
3 # => 3

# 算术没有什么出乎意料的
1 + 1 # => 2
8 - 1 # => 7
10 * 2 # => 20
```

```
# 但是除法例外，会自动转换成浮点数
35 / 5 # => 7.0
5 / 3 # => 1.6666666666666667

# 整数除法的结果都是向下取整
5 // 3      # => 1
5.0 // 3.0 # => 1.0 # 浮点数也可以
-5 // 3    # => -2
-5.0 // 3.0 # => -2.0

# 浮点数的运算结果也是浮点数
3 * 2.0 # => 6.0

# 模除
7 % 3 # => 1

# x 的 y 次方
2**4 # => 16

# 用括号决定优先级
(1 + 3) * 2 # => 8

# 布尔值
True
False

# 用 not 取非
not True # => False
not False # => True
```

```
# 逻辑运算符，注意 and 和 or 都是小写
```

```
True and False # => False
```

```
False or True # => True
```

```
# 整数也可以当作布尔值
```

```
0 and 2 # => 0
```

```
-5 or 0 # => -5
```

```
0 == False # => True
```

```
2 == True # => False
```

```
1 == True # => True
```

```
# 用==判断相等
```

```
1 == 1 # => True
```

```
2 == 1 # => False
```

```
# 用!=判断不等
```

```
1 != 1 # => False
```

```
2 != 1 # => True
```

```
# 比较大小
```

```
1 < 10 # => True
```

```
1 > 10 # => False
```

```
2 <= 2 # => True
```

```
2 >= 2 # => True
```

```
# 大小比较可以连起来！
```

```
1 < 2 < 3 # => True
```

```
2 < 3 < 2 # => False
```

```
# 字符串用单引双引都可以
"这是个字符串"
'这也是个字符串'

# 用加号连接字符串
"Hello " + "world!" # => "Hello world!"

# 字符串可以被当作字符列表
"This is a string"[0] # => 'T'

# 用.format 来格式化字符串
"{} can be {}".format("strings", "interpolated")

# 可以重复参数以节省时间
"{0} be nimble, {0} be quick, {0} jump over the {1}".format("Jack",
"candle stick")
# => "Jack be nimble, Jack be quick, Jack jump over the candle stick"

# 如果不想数参数，可以用关键字
"{name} wants to eat {food}".format(name="Bob", food="lasagna")
# => "Bob wants to eat lasagna"

# 如果你的 Python3 程序也要在 Python2.5 以下环境运行，也可以用老式的格式化语法
"%s can be %s the %s way" % ("strings", "interpolated", "old")

# None 是一个对象
None # => None

# 当与 None 进行比较时不要用 ==，要用 is。is 是用来比较两个变量是否指向同一个对象。
```

```
"etc" is None # => False
None is None # => True

# None, 0, 空字符串, 空列表, 空字典都算是 False
# 所有其他值都是 True

bool(0) # => False
bool("") # => False
bool([]) # => False
bool({}) # => False
```

## 2.2 变量和集合

```
#####
## 2. 变量和集合
#####

# print 是内置的打印函数
print("I'm Python. Nice to meet you!")

# 在给变量赋值前不用提前声明
# 传统的变量命名是小写，用下划线分隔单词
some_var = 5
some_var # => 5

# 访问未赋值的变量会抛出异常
# 参考流程控制一段来学习异常处理
some_unknown_var # 抛出 NameError

# 用列表(list)储存序列
li = []
# 创建列表时也可以同时赋给元素
other_li = [4, 5, 6]

# 用 append 在列表最后追加元素
li.append(1)    # li 现在是[1]
li.append(2)    # li 现在是[1, 2]
li.append(4)    # li 现在是[1, 2, 4]
li.append(3)    # li 现在是[1, 2, 4, 3]
# 用 pop 从列表尾部删除
```

```
li.pop()          # => 3 且 li 现在是[1, 2, 4]
# 把 3 再放回去

li.append(3)    # li 变回[1, 2, 4, 3]

# 列表存取跟数组一样

li[0]  # => 1
# 取出最后一个元素

li[-1] # => 3

# 越界存取会造成 IndexError

li[4] # 抛出 IndexError

# 列表有切割语法

li[1:3] # => [2, 4]
# 取尾

li[2:] # => [4, 3]
# 取头

li[:3] # => [1, 2, 4]
# 隔一个取一个

li[::-2] # => [1, 4]
# 倒排列表

li[::-1] # => [3, 4, 2, 1]
# 可以用三个参数的任何组合来构建切割

# li[始:终:步伐]

# 用 del 删除任何一个元素

del li[2] # li is now [1, 2, 3]

# 列表可以相加

# 注意: li 和 other_li 的值都不变
```

```
li + other_li    # => [1, 2, 3, 4, 5, 6]

# 用 extend 拼接列表
li.extend(other_li)  # li 现在是[1, 2, 3, 4, 5, 6]

# 用 in 测试列表是否包含值
1 in li  # => True

# 用 len 取列表长度
len(li)  # => 6

# 元组是不可改变的序列
tup = (1, 2, 3)
tup[0]  # => 1
tup[0] = 3  # 抛出 TypeError

# 列表允许的操作元组大都可以
len(tup)  # => 3
tup + (4, 5, 6)  # => (1, 2, 3, 4, 5, 6)
tup[:2]  # => (1, 2)
2 in tup  # => True

# 可以把元组合成列表解包，赋值给变量
a, b, c = (1, 2, 3)  # 现在 a 是 1, b 是 2, c 是 3
# 元组周围的括号是可以省略的
d, e, f = 4, 5, 6
# 交换两个变量的值就这么简单
e, d = d, e  # 现在 d 是 5, e 是 4
```

```
# 用字典表达映射关系

empty_dict = {}

# 初始化的字典

filled_dict = {"one": 1, "two": 2, "three": 3}

# 用[]取值

filled_dict["one"] # => 1

# 用 keys 获得所有的键。

# 因为 keys 返回一个可迭代对象，所以在这里把结果包在 list 里。我们下面会详细介绍可迭代。

# 注意：字典键的顺序是不定的，你得到的结果可能和以下不同。

list(filled_dict.keys()) # => ["three", "two", "one"]

# 用 values 获得所有的值。跟 keys 一样，要用 list 包起来，顺序也可能不同。

list(filled_dict.values()) # => [3, 2, 1]

# 用 in 测试一个字典是否包含一个键

"one" in filled_dict # => True

1 in filled_dict # => False

# 访问不存在的键会导致 KeyError

filled_dict["four"] # KeyError

# 用 get 来避免 KeyError

filled_dict.get("one") # => 1
```

```

filled_dict.get("four")    # => None
# 当键不存在的时候 get 方法可以返回默认值
filled_dict.get("one", 4)   # => 1
filled_dict.get("four", 4)  # => 4

# setdefault 方法只有当键不存在的时候插入新值
filled_dict.setdefault("five", 5) # filled_dict["five"]设为 5
filled_dict.setdefault("five", 6) # filled_dict["five"]还是 5

# 字典赋值
filled_dict.update({"four":4}) # => {"one": 1, "two": 2, "three": 3,
"four": 4}
filled_dict["four"] = 4 # 另一种赋值方法

# 用 del 删除
del filled_dict["one"] # 从 filled_dict 中把 one 删除

# 用 set 表达集合
empty_set = set()
# 初始化一个集合，语法跟字典相似。
some_set = {1, 1, 2, 2, 3, 4} # some_set 现在是{1, 2, 3, 4}

# 可以把集合赋值于变量
filled_set = some_set

# 为集合添加元素
filled_set.add(5) # filled_set 现在是{1, 2, 3, 4, 5}

# & 取交集

```

```
other_set = {3, 4, 5, 6}
filled_set & other_set    # => {3, 4, 5}

# | 取并集
filled_set | other_set    # => {1, 2, 3, 4, 5, 6}

# - 取补集
{1, 2, 3, 4} - {2, 3, 5}    # => {1, 4}

# in 测试集合是否包含元素
2 in filled_set    # => True
10 in filled_set   # => False
```

## 2.3 流程控制和迭代器

```
#####
## 3. 流程控制和迭代器
#####

# 先随便定义一个变量
some_var = 5

# 这是个 if 语句。注意缩进在 Python 里是有意义的
# 印出"some_var 比 10 小"
if some_var > 10:
    print("some_var 比 10 大")
elif some_var < 10:    # elif 句是可选的
    print("some_var 比 10 小")
else:                  # else 也是可选的
    print("some_var 就是 10")

"""

用 for 循环语句遍历列表
打印:
    dog is a mammal
    cat is a mammal
    mouse is a mammal
"""

for animal in ["dog", "cat", "mouse"]:
    print("{} is a mammal".format(animal))

"""


```

```
"range(number)"返回数字列表从 0 到给的数字
```

```
打印：
```

```
0  
1  
2  
3  
....  
  
for i in range(4):  
    print(i)
```

```
....
```

```
while 循环直到条件不满足
```

```
打印：
```

```
0  
1  
2  
3  
....  
  
x = 0  
  
while x < 4:  
    print(x)  
    x += 1 # x = x + 1 的简写
```

```
# 用 try/except 块处理异常状况
```

```
try:
```

```
    # 用 raise 抛出异常  
    raise IndexError("This is an index error")  
  
except IndexError as e:  
    pass # pass 是无操作，但是应该在这里处理错误  
  
except (TypeError, NameError):
```

```
pass      # 可以同时处理不同类的错误

else:    # else 语句是可选的，必须在所有的 except 之后
    print("All good!")  # 只有当 try 运行完没有错误的时候这句才会运行

# Python 提供一个叫做可迭代(iterable)的基本抽象。一个可迭代对象是可以被当作序列

# 的对象。比如说上面 range 返回的对象就是可迭代的。

filled_dict = {"one": 1, "two": 2, "three": 3}
our_iterable = filled_dict.keys()

print(our_iterable) # => dict_keys(['one', 'two', 'three']), 是一个实现可迭代接口的对象

# 可迭代对象可以遍历

for i in our_iterable:
    print(i)    # 打印 one, two, three

# 但是不可以随机访问

our_iterable[1] # 抛出 TypeError

# 可迭代对象知道怎么生成迭代器

our_iterator = iter(our_iterable)

# 迭代器是一个可以记住遍历的位置的对象

# 用 __next__ 可以取得下一个元素

our_iterator.__next__() # => "one"

# 再一次调取 __next__ 时会记得位置

our_iterator.__next__() # => "two"
```

```
our_iterator.__next__() # => "three"

# 当迭代器所有元素都取出后，会抛出 StopIteration
our_iterator.__next__() # 抛出 StopIteration

# 可以用 list 一次取出迭代器所有的元素
list(filled_dict.keys()) # Returns ["one", "two", "three"]
```

## 2.4 函数

```
#####
## 4. 函数
#####

# 用 def 定义新函数
def add(x, y):
    print("x is {} and y is {}".format(x, y))
    return x + y    # 用 return 语句返回

# 调用函数
add(5, 6)    # => 印出"x is 5 and y is 6"并且返回 11

# 也可以用关键字参数来调用函数
add(y=6, x=5)    # 关键字参数可以用任何顺序

# 我们可以定义一个可变参数函数
def varargs(*args):
    return args

varargs(1, 2, 3)    # => (1, 2, 3)

# 我们也可以定义一个关键字可变参数函数
def keyword_args(**kwargs):
    return kwargs
```

```

# 我们来看看结果是什么:

keyword_args(big="foot", loch="ness")  # => {"big": "foot", "loch": "ness"}


# 这两种可变参数可以混着用

def all_the_args(*args, **kwargs):
    print(args)
    print(kwargs)
    """
all_the_args(1, 2, a=3, b=4) prints:
(1, 2)
{"a": 3, "b": 4}
"""

# 调用可变参数函数时可以做跟上面相反的，用*展开序列，用**展开字典。

args = (1, 2, 3, 4)
kwargs = {"a": 3, "b": 4}

all_the_args(*args)  # 相当于 foo(1, 2, 3, 4)
all_the_args(**kwargs)  # 相当于 foo(a=3, b=4)
all_the_args(*args, **kwargs)  # 相当于 foo(1, 2, 3, 4, a=3, b=4)


# 函数作用域

x = 5


def setX(num):
    # 局部作用域的 x 和全局域的 x 是不同的
    x = num # => 43
    print (x) # => 43

```

```
def setGlobalX(num):  
    global x  
    print (x) # => 5  
    x = num # 现在全局域的 x 被赋值  
    print (x) # => 6  
  
setX(43)  
setGlobalX(6)  
  
# 函数在 Python 是一等公民  
def create_adder(x):  
    def adder(y):  
        return x + y  
    return adder  
  
add_10 = create_adder(10)  
add_10(3) # => 13  
  
# 也有匿名函数  
(lambda x: x > 2)(3) # => True  
  
# 内置的高阶函数  
map(add_10, [1, 2, 3]) # => [11, 12, 13]  
filter(lambda x: x > 5, [3, 4, 5, 6, 7]) # => [6, 7]  
  
# 用列表推导式可以简化映射和过滤。列表推导式的返回值是另一个列表。  
[add_10(i) for i in [1, 2, 3]] # => [11, 12, 13]  
[x for x in [3, 4, 5, 6, 7] if x > 5] # => [6, 7]
```

## 2.5 类

```
#####
## 5. 类
#####

# 定义一个继承 object 的类
class Human(object):

    # 类属性，被所有此类的实例共用。
    species = "H. sapiens"

    # 构造方法，当实例被初始化时被调用。注意名字前后的双下划线，这是表明这个属
    # 性或方法对 Python 有特殊意义，但是允许用户自行定义。你自己取名时不应该用
    这

    # 种格式。

    def __init__(self, name):
        # Assign the argument to the instance's name attribute
        self.name = name

    # 实例方法，第一个参数总是 self，就是这个实例对象
    def say(self, msg):
        return "{name}: {message}".format(name=self.name, message=msg)

    # 类方法，被所有此类的实例共用。第一个参数是这个类对象。
    @classmethod
    def get_species(cls):
        return cls.species
```

```
# 静态方法。调用时没有实例或类的绑定。  
  
@staticmethod  
  
def grunt():  
    return "*grunt*"  
  
# 构造一个实例  
  
i = Human(name="Ian")  
print(i.say("hi"))      # 印出 "Ian: hi"  
  
j = Human("Joel")  
print(j.say("hello"))  # 印出 "Joel: hello"  
  
# 调用一个类方法  
  
i.get_species()      # => "H. sapiens"  
  
# 改一个共用的类属性  
  
Human.species = "H. neanderthalensis"  
i.get_species()      # => "H. neanderthalensis"  
j.get_species()      # => "H. neanderthalensis"  
  
# 调用静态方法  
  
Human.grunt()       # => "*grunt*"
```

## 2.6 模块

```
#####
## 6. 模块
#####

# 用 import 导入模块

import math

print(math.sqrt(16)) # => 4.0


# 也可以从模块中导入个别值

from math import ceil, floor

print(ceil(3.7)) # => 4.0
print(floor(3.7)) # => 3.0


# 可以导入一个模块中所有值

# 警告：不建议这么做

from math import *


# 如此缩写模块名字

import math as m

math.sqrt(16) == m.sqrt(16) # => True


# Python 模块其实就是普通的 Python 文件。你可以自己写，然后导入，

# 模块的名字就是文件的名字。

# 你可以这样列出一个模块里所有的值

import math

dir(math)
```

## 2.7 高级用法

```
#####
## 7. 高级用法
#####

# 用生成器(generators)方便地写惰性运算

def double_numbers(iterable):
    for i in iterable:
        yield i + i

# 生成器只有在需要时才计算下一个值。它们每一次循环只生成一个值，而不是把所有的
# 值全部算好。
#
# range 的返回值也是一个生成器，不然一个 1 到 900000000 的列表会花很多时间和内
# 存。
#
# 如果你想用一个 Python 的关键字当作变量名，可以加一个下划线来区分。

range_ = range(1, 900000000)
# 当找到一个 >=30 的结果就会停
# 这意味着 `double_numbers` 不会生成大于 30 的数。
for i in double_numbers(range_):
    print(i)
    if i >= 30:
        break

# 装饰器(decorators)
# 这个例子中，beg 装饰 say
```

```
# beg 会先调用 say。如果返回的 say_please 为真， beg 会改变返回的字符串。
from functools import wraps

def beg(target_function):
    @wraps(target_function)
    def wrapper(*args, **kwargs):
        msg, say_please = target_function(*args, **kwargs)
        if say_please:
            return "{} {}".format(msg, "Please! I am poor :(")
        return msg

    return wrapper

@beg
def say(say_please=False):
    msg = "Can you buy me a beer?"
    return msg, say_please

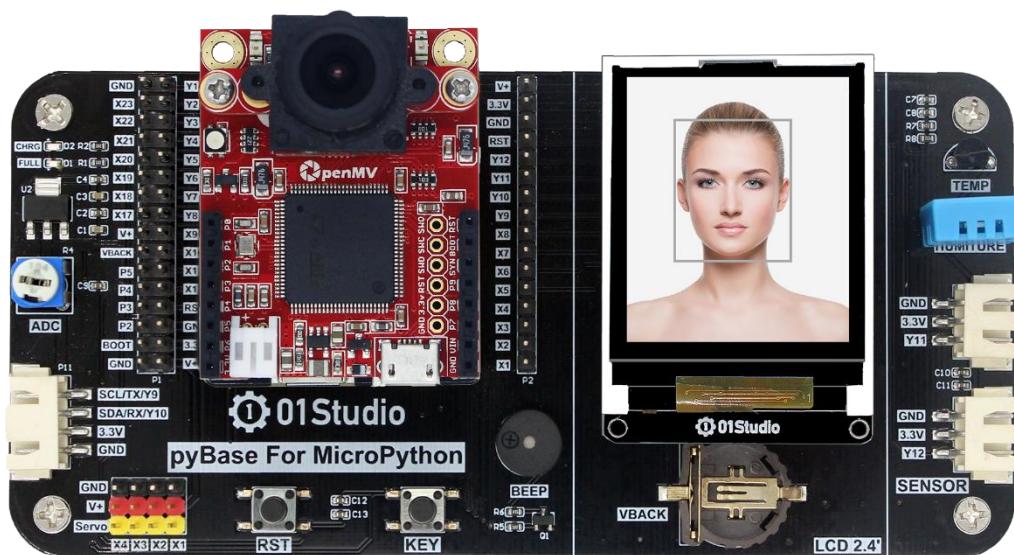
print(say()) # Can you buy me a beer?
print(say(say_please=True)) # Can you buy me a beer? Please! I am
poor :(
```

## 第二部分 基于 OpenMV4 平台

OpenMV（开源机器视觉）是美国克里斯·安德森及其团队基于 MicroPython 发起的开源项目，该项目旨在创建低成本，可扩展，使用 Python 驱动的机器视觉模块，旨在成为“机器视觉的 Arduino”。开源目标是使机器视觉算法更接近制造商和业余爱好者。该开源项目完成了困难且耗时的算法工作，为我们的创作留出了更多时间！

OpenMV Cam 就像一个功能强大的 Arduino，带有一个用 Python 编程的板载摄像头。我们可以轻松地在 OpenMV Cam 所看到的图像上运行机器视觉算法，因此您可以在几秒钟内跟踪颜色，检测面部等等，然后在实际环境中控制 I / O 引脚。

如果你学习了前面的平台，那么恭喜你，对于 OpenMV 你会非常快速上手。如果你没有学习过任何一个 MicroPython 平台那也没关系，你可以借助 OpenMV4 平台的学习从而入门 MicroPython。



pyAI-OpenMV4 开发套件

## 第3章 开发环境快速建立

### 3.1 基于 Windows

#### 3.1.1 安装开发软件 OpenMV IDE

由于 OpenMV 拥有成熟的团队，他们为该平台量身定做了 IDE，可以在官网下载，我们使用该 IDE 可以轻松进行开发。而且拥有 Windows、Mac OS、Linux、树莓派等版本。

下载地址：<https://openmv.io/pages/download>，下载界面如下图。用户可以根据自己系统选择合适的版本下载安装！

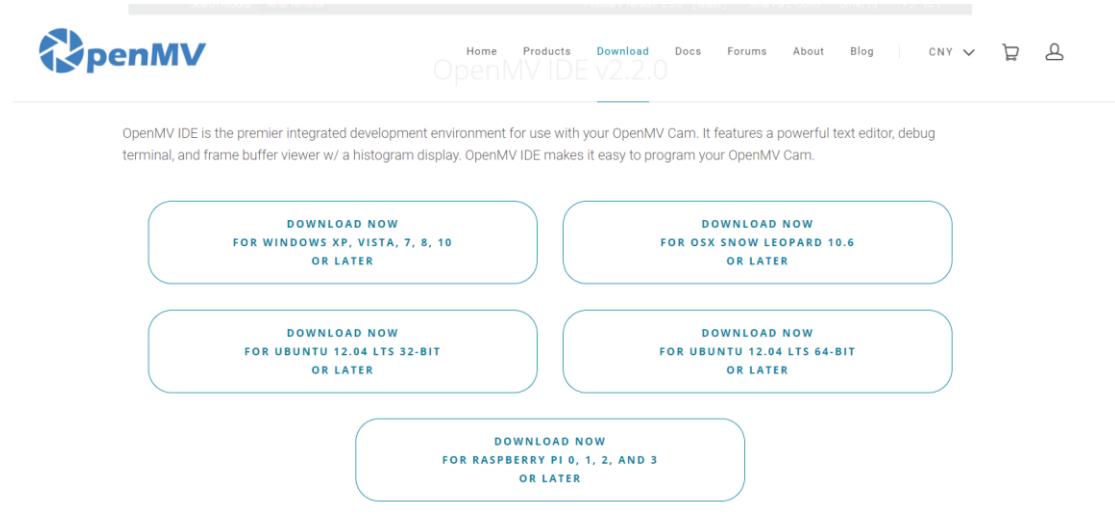


图 3-1 OpenMV IDE 下载界面

安装完成后打开软件，如下图所示，至此安装完成。

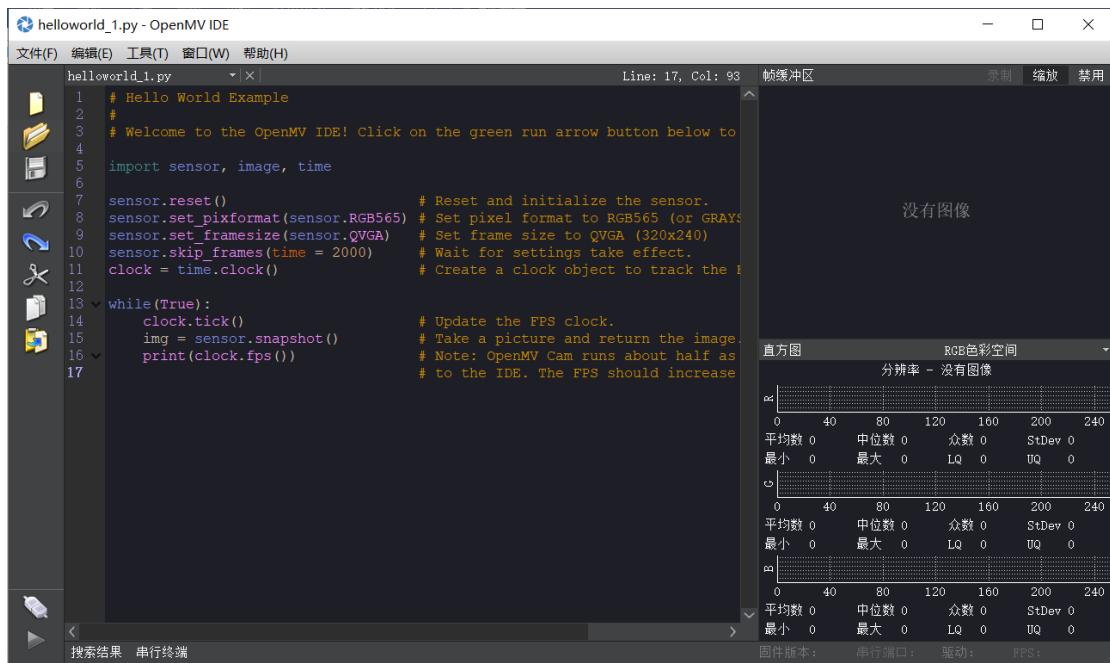


图 3-2 OpenMV IDE

### 3.1.2 开发套件使用

#### 3.1.2.1 驱动安装

我们在上一节安装完 OpenMV IDE 后，驱动也一起安装了。我们将 pyAI-OpenMV4 开发板通过 MicroUSB 数据线连接到电脑：

我们将 MicroPython 开发板通过 MicroUSB 数据线连接到电脑：

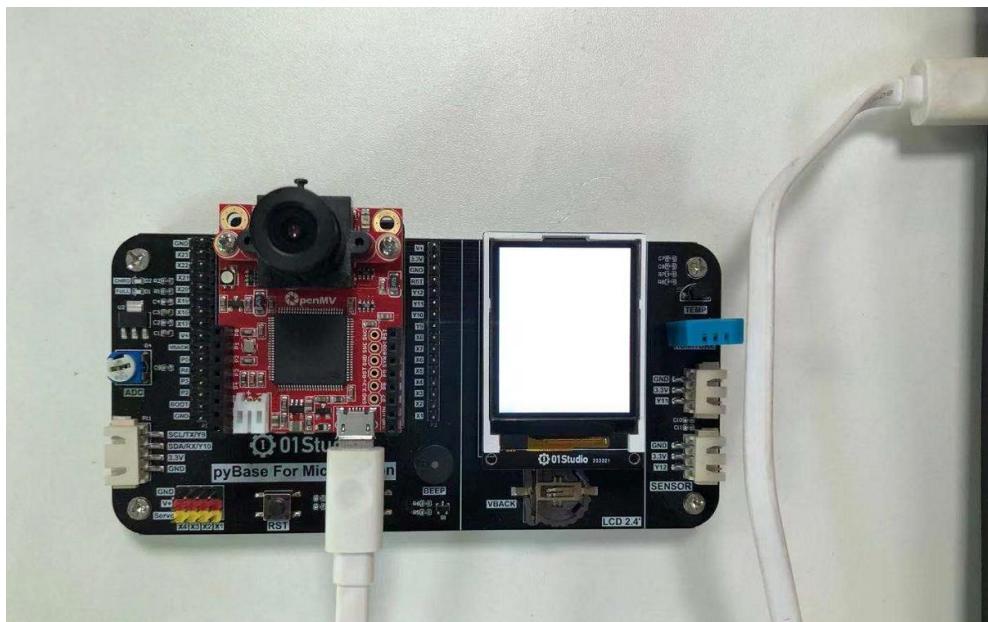


图 3-3 通过 MicroUSB 线连接到电脑

连接电脑后，会发现出现一个 U 盘，点击打开，看到以下文件。

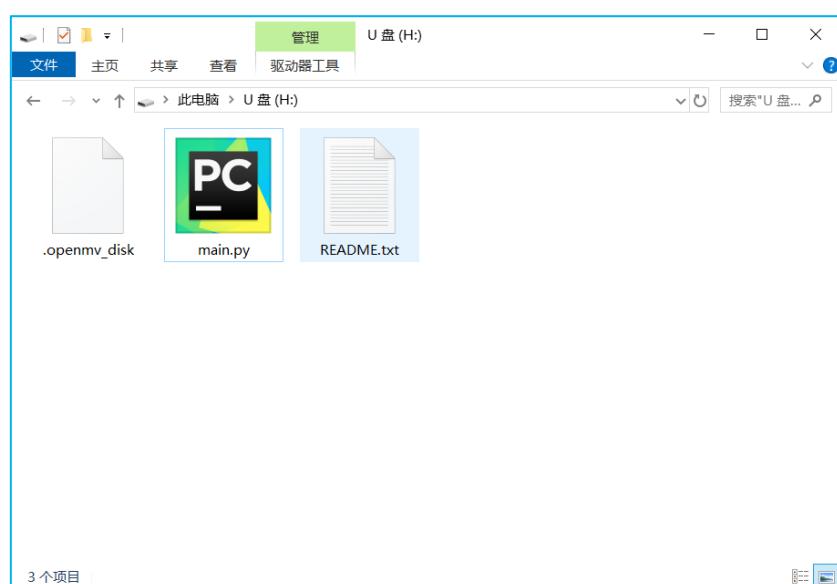


图 3-4 OpenMV4 的系统文件

下面是对主要文件的简介：

1. **main.py**: 主函数代码文件，上电后首先执行；
2. **README.txt**: 说明文件；

另外系统还自动安装一个 USB 转串口的驱动，用于调试，鼠标右键点击“我的电脑”—属性—设备管理器：能找到 OpenMV4 虚拟串口新设备：

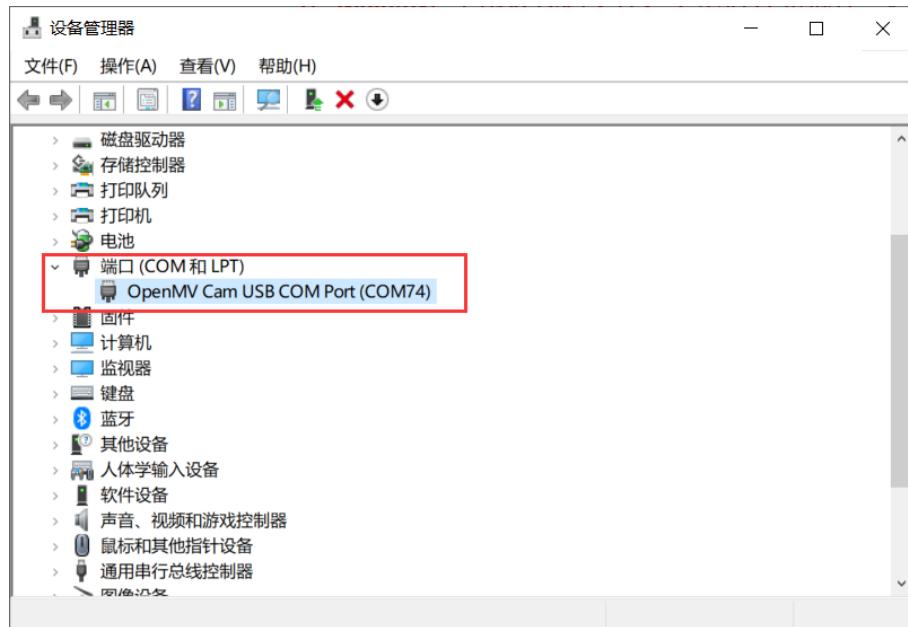


图 3-5 OpenMV 串口驱动

### 注意事项：

部分用户的 Win7 系统由于使用的是 Ghost 版系统，Ghost 系统精简了很多系统文件，有可能导致驱动无法安装。我们不建议开发者安装 Ghost 的系统。这时候建议用户采取以下措施。

- (1) 使用 360 的系统重装功能，实测重装后驱动可以安装；
- (2) 升级成 Win10 系统。

我们更推荐方法 (2)，因为微软已经宣布停止对 Win7 进行更新，另外就是我们的教程都会基于 Win10 系统。

### 3.1.2.2 例程测试

我们使用 OpenMV IDE 来进行我们的第一个实验，借此来熟悉开发环境。将 pyAI-OpenMV4 开发板通过 MicroUSB 线连接到电脑。打开 OpenMV IDE，点击左下角连接按钮：

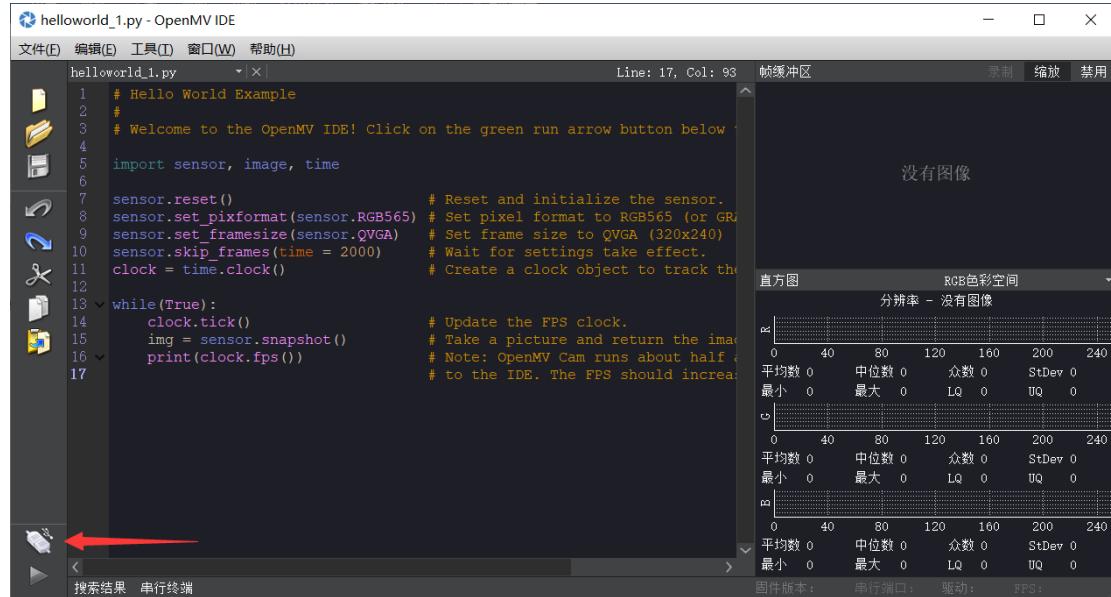


图 3-6 点击连接按钮

连接成功后，运行按钮变成绿色。

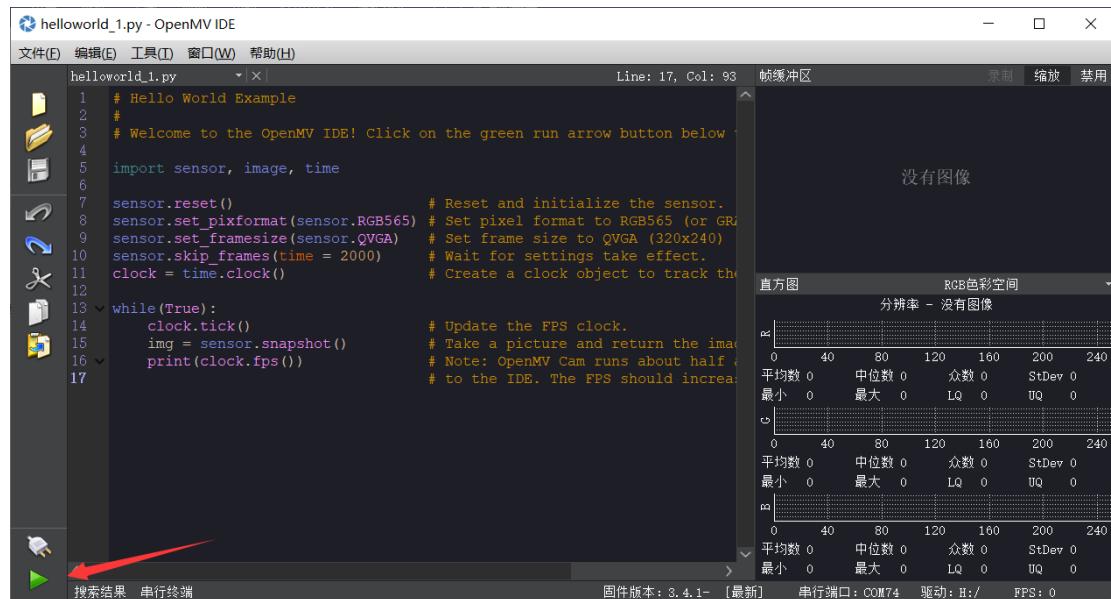


图 3-7 连接成功

当前的文件是 IDE 自带的 `hello_world.py` 程序文件，我们点击绿色按键“运行”按钮（记得打开摄像头盖），可以看到程序在运行，右上角出现当前摄像头实时采集图像和色彩直方图，说明实验成功：

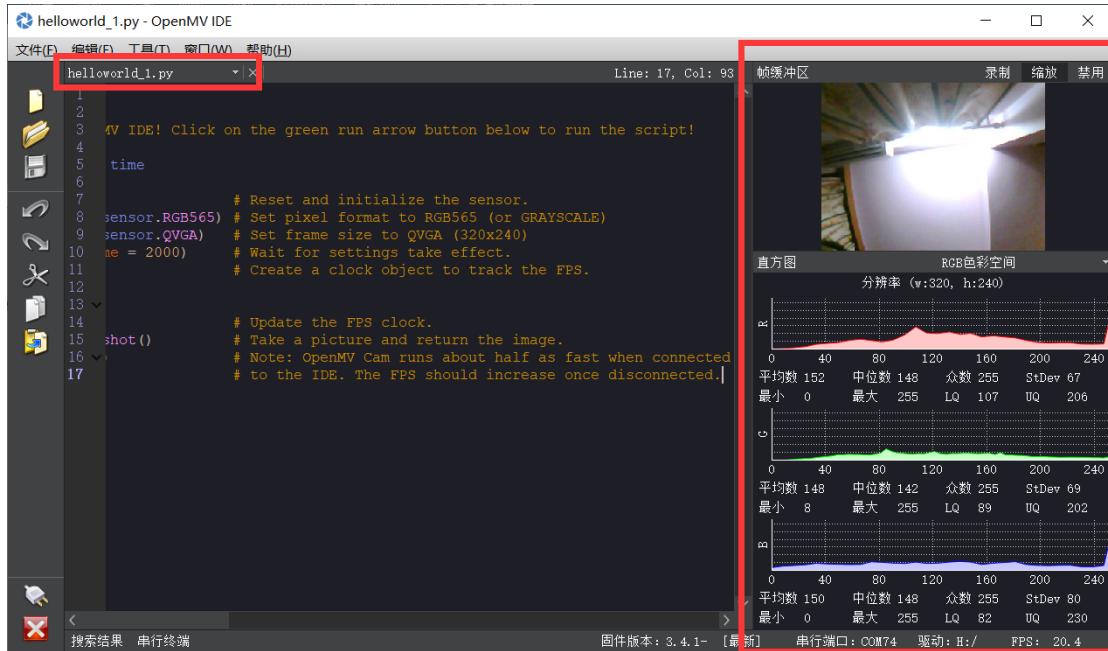


图 3-8 第一个实验

以上测试是基于 IDE 的运行功能，当然你也可以直接将 `py` 文件保存到 U 盘文件系统中，这样程序不再依赖 IDE 就可以直接运行。具体方法如下：

将代码放在 `main.py` 文件中（`main.py` 为上电自动运行的文件）。然后将 `main.py` 复制到设备的文件系统中，设备红色 LED 灯亮表示文件正在重写，需要等待红色灯灭掉，然后按下复位按键。程序即重新运行。

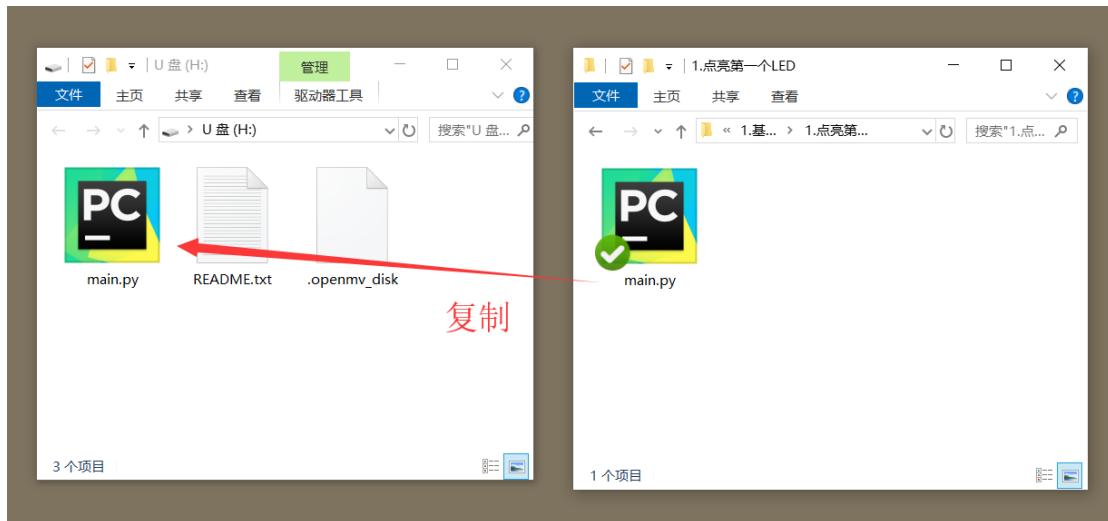


图 3-9 复制文件到 OpenMV4 文件系统

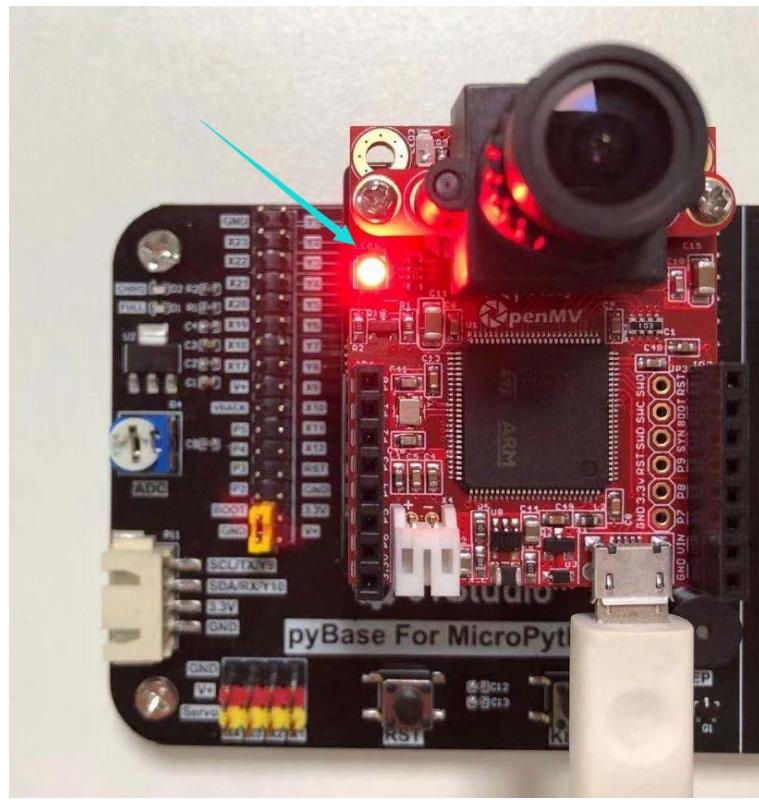


图 3-10 等待红灯熄灭再操作

OpenMV IDE 集成了串口调试窗口，点击 IDE 左下角“串口终端”，即可观察到串口打印的信息。

```

helloworld_1.py - OpenMV IDE
文件(F) 编辑(E) 工具(T) 窗口(W) 帮助(H)
helloworld_1.py Line: 17, Col: 93 帧缓冲区 录制 缩放 禁用
1
2
3 # IDE! Click on the green run arrow button below to run the script!
4
5 time
6
7     # Reset and initialize the sensor.
8 sensor.RGB565 # Set pixel format to RGB565 (or GRayscale)
9 sensor.QVGA # Set frame size to QVGA (320x240)
10 ne = 2000 # Wait for settings take effect.
11 # Create a clock object to track the FPS.
12
13     # Update the FPS clock.
14 shot() # Take a picture and return the image.
15 # Note: OpenMV Cam runs about half as fast when connected
16 # to the IDE. The FPS should increase once disconnected.
17

< 串行终端 >
50.82932
50.63291

Traceback (most recent call last):
  File "<stdin>", line 15, in <module>
Exception: IDE interrupt
MicroPython v1.9.4-4553-gb4eccdfe3 on 2019-05-02; OPENMV4 with STM32H743
Type "help()" for more information.
>>>

固件版本: 3.4.1- [最新] 串行端口: COM74 驱动: H:/ FPS: 0

```

图 3-11 串口终端调试

### 3.1.2.3 固件更新

当 OpenMV4 上的固件意外丢失或者我们希望升级到较新版本固件时候，就需重新烧录固件。OpenMV4 上面是一个 STM32 单片机，所以这个操作相当于给 STM32 单片机重新烧录固件。

(先将 OpenMV4 与电脑断开连接，务必先断开)，然后打开 OpenMV IDE，点击左下角连接按钮：

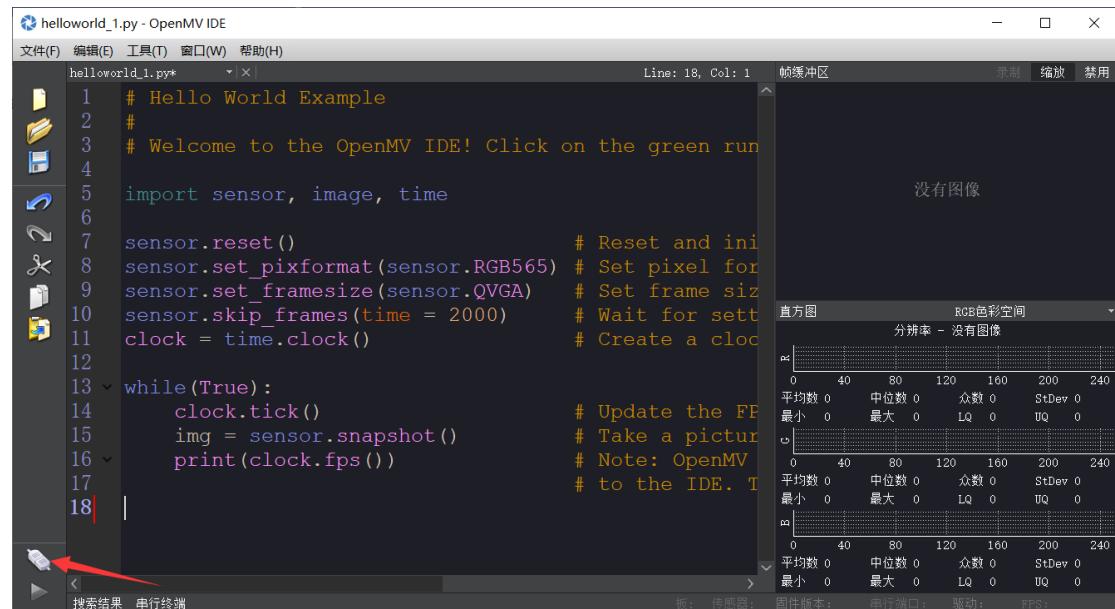


图 3-12

在弹出对话框“没有找到 OpenMV Cams!”，直接点击 OK：



图 3-13

询问 OpenMV Cam 是否变砖了，点击 Yes:

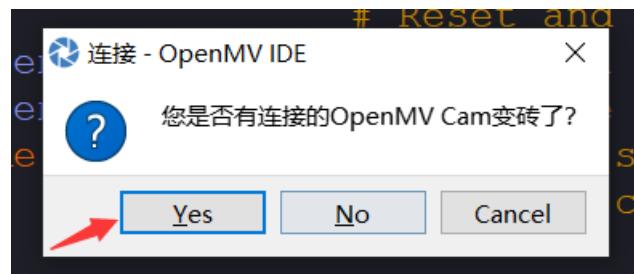


图 3-14

选择开发板类型，OpenMV4 就是下图的 H7 (如果你使用 openmv plus，则选择 plus 主板类型):



图 3-15

询问是否擦除文件系统，点击 Yes:

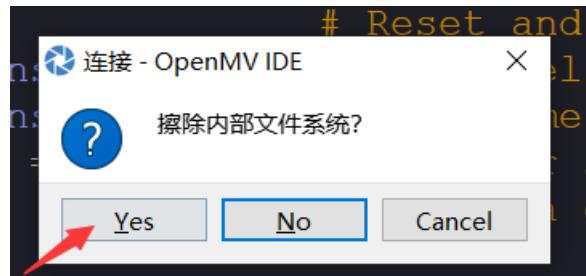


图 3-16

出现以下界面:

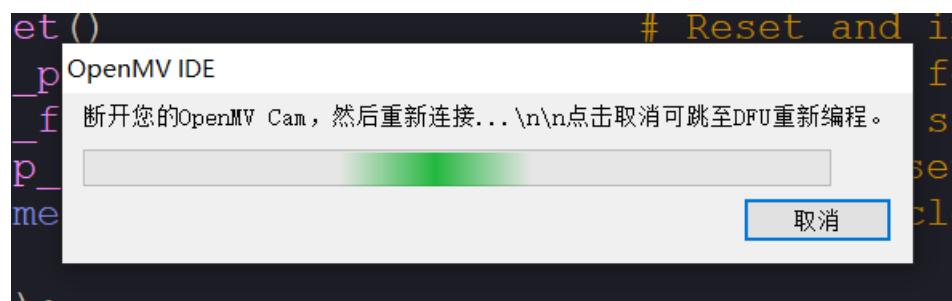


图 3-17

此时才连接 OpenMV 开发板，出现下面界面表示开始擦除：



图 3-18

擦除后开始烧写固件：



图 3-19

烧写完成后出现下面界面，然后模块进入自检，等开发板蓝色灯闪亮再点击 OK 即可：

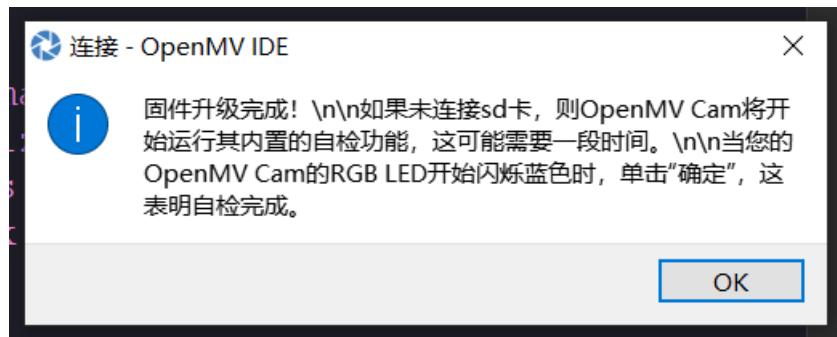


图 3-20

自检完成后会弹出 U 盘盘符：



图 3-21

至此，重烧固件完成。

### 3.1.2.4 正版激活

(01Studio 官方 openmv 系列产品出厂均已带官方正版 KEY)

OpenMV4 的官方团队为 OpenMV4 打造了完善的开发环境 OpenMV IDE，但并不是无偿的。需要购买官方激活码，俗称（KEY）来对设备进行激活，如果你在其它渠道或自制的 OpenMV4 设备尚未激活，那么使用官方 OpenMV4 IDE 将在每次连接设备时候弹出 3 次提示激活，影响用户体验。

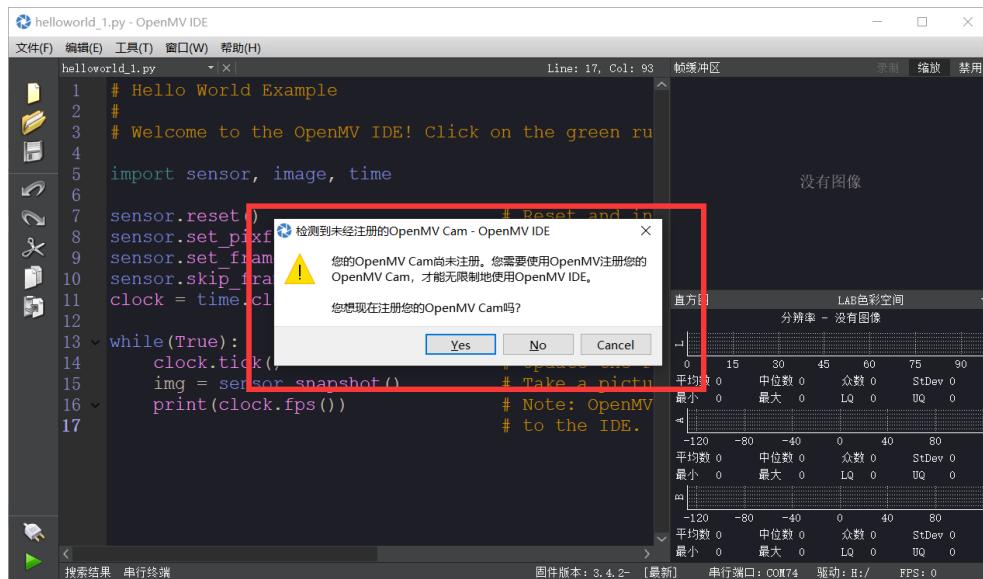


图 3-22 提示激活

这时候你需要到 OpenMV 官网或者联系 01Studio 购买 KEY，KEY 为一串 25 个字符的编码，一旦激活后跟当前 OpenMV 设备唯一绑定，IDE 即可正常使用。

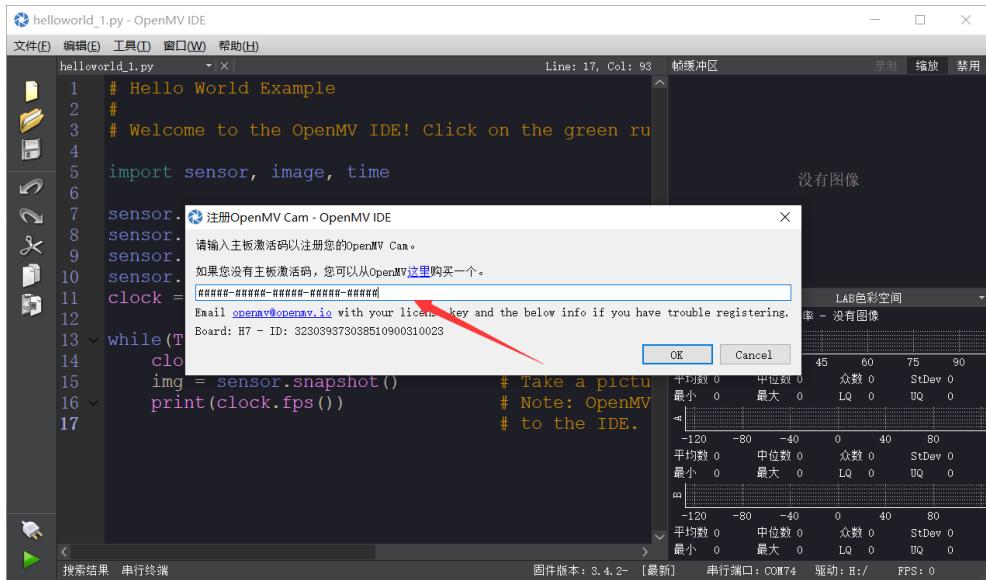


图 3-23 激活 OpenMV 设备

尽管 OpenMV 是开源项目，官方在开发环境和升级更新花了不少精力，因此我们 01Studio 出品的产品均为已激活产品以表示对其团队支持，用户如果是在市面上其它渠道购买 OpenMV 设备而没激活的建议购买 KEY 激活以支持官方。

## 3.2 基于 Mac OS

### 3.2.1 安装开发软件 OpenMV IDE

这个跟 Windows 环境安装方法完全一样。具体请参考：[3.2.1 安装开发软件 OpenMV IDE](#) 章节内容，这里不再重复。

下载地址：<https://openmv.io/pages/download>，下载界面如下图。用户可以根据自己系统选择合适的版本下载安装！

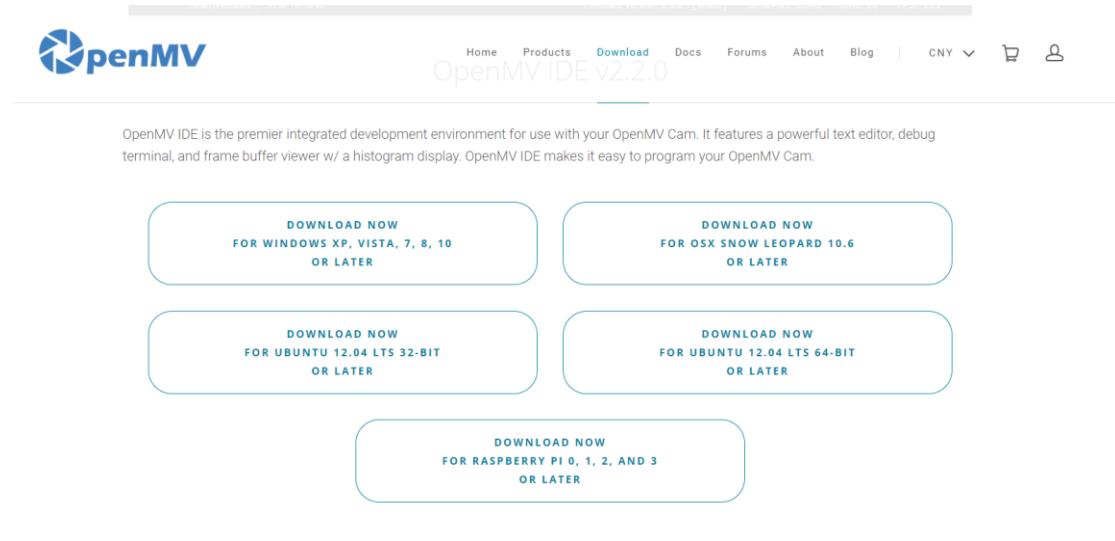


图 3-24 OpenMV IDE 下载界面

### 3.3 基于 Linux（树莓派）

OpenMV IDE 提供 Linux 和树莓派版本，下载压缩包看 Readme 文件按步骤安装即可，使用方法请参考：[3.2.1 安装开发软件 OpenMV IDE](#) 章节内容，这里不再重复。

下载地址：<https://openmv.io/pages/download>，下载界面如下图。用户可以根据自己系统选择合适的版本下载安装！

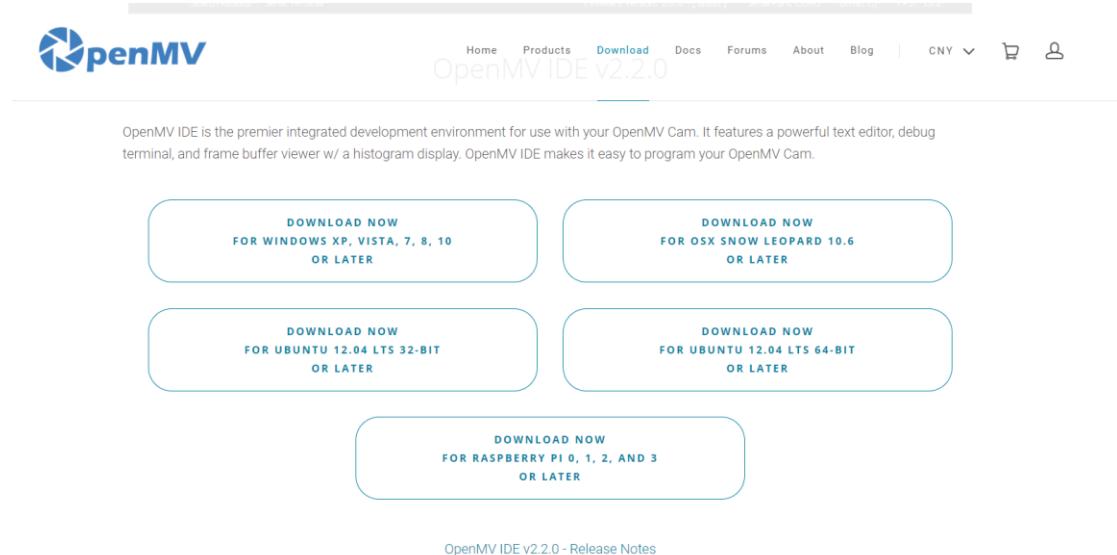


图 3-25 OpenMV IDE 下载界面

## 第4章 基础实验

MicroPython 更强调的是针对应用的学习，强大的底层库函数让我们可以直接关心功能的实现，也就是说我们只要理解和熟练相关的函数用法，就可以很好的玩转 MicroPython。它让我们可以做到不关心硬件和底层原理（当然有兴趣和能力的小伙伴可以深入研究）而直接跑起硬件。

基础实验是针对一些简单的实验学习讲解，可以让我们更快和更直接感受到 MicroPython 针对嵌入式开发的强大之处，我后面的学习和开发夯实基础。

请先看实验讲解格式预览，每一节我们都会以以下形式讲解，图文并茂，力求达到快速理解和学习的作用：

- (1) **前言**: 简单介绍这个实验;
- (2) **实验平台**: 实验所用到的开发板、配件和接线说明;
- (3) **实验目的**: 本实验要实现的功能;
- (4) **实验讲解**: 对函数、代码、编程方法以及实验的详细讲解;
- (5) **实验结果**: 记录程序下载到开发板上的图片示例;
- (6) **总结**: 对实验进行总结。

## 4.1 点亮第一个 LED

- **前言:**

相信大部分人开始学习嵌入式单片机编程都会从点亮 LED 开始，基于 OpenMV4 平台的 MicroPython 的学习也不例外，通过点亮第一个 LED 能让你对编译环境和程序架构有一定的认识，为以后的学习和更大型的程序打下基础，增加信心。

- **实验平台:**

pyAI-OpenMV4。

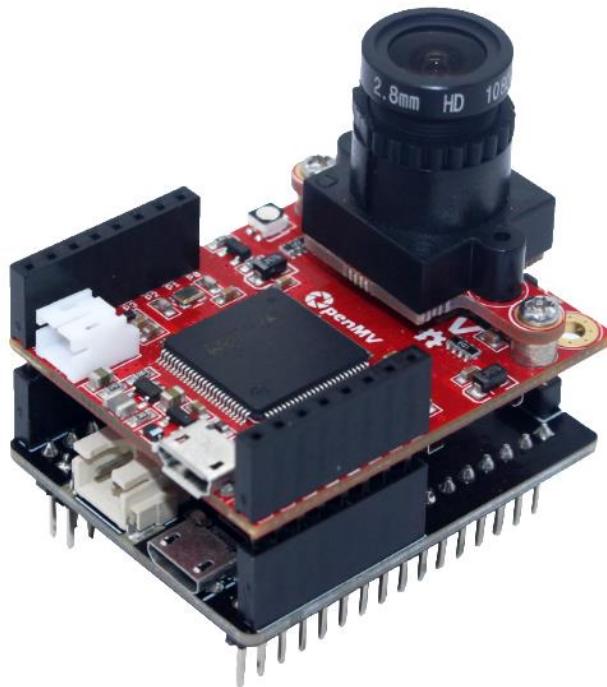


图 4-1 pyAI-OpenMV4

- **实验目的:**

学习 LED 的点亮，点亮 LED3（蓝灯）。

- **实验讲解:**

pyAI-OpenMV4 上总共有 4 个 LED，分别是 LED1(红色)、LED2(绿色)、LED3(蓝色)和 LED4(红外拍照灯)；我们可以直接通过 MicroPython 对象编程来控制。控制 LED 使用到 LED 对象。LED 的构造函数和使用方法如下表所示：

构造函数
<code>pyb.LED(id)</code>
LED 对象在 <code>pyb</code> 模块下。
使用方法
<code>LED.on()</code>
点亮 LED，输出低电平。
<code>LED.off()</code>
关闭 LED，输出高电平。
<code>LED.toggle()</code>
LED 亮灭状态翻转，输出电平翻转

表 4-1 LED 对象

从上表可以看到，LED 通过对象编程让使用变得非常简单。我们先来看看 OpenMV4 的 LED 的布局和接线图。

RGB 为三色一体 LED，在下图黄色框内。IR 红外 LED 有 2 个在摄像头的两旁，下图蓝色框内。

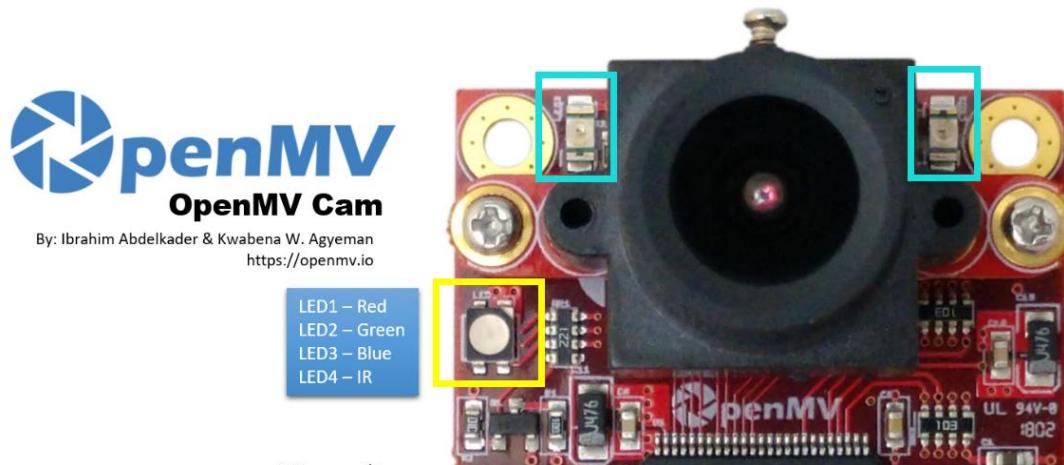


图 4-2 LED 布局

从下面原理图可以看到，LED 的正极都是接在 VCC (3.3V)，因此当 IO 为低电平点亮。

# IR-RGB LEDs

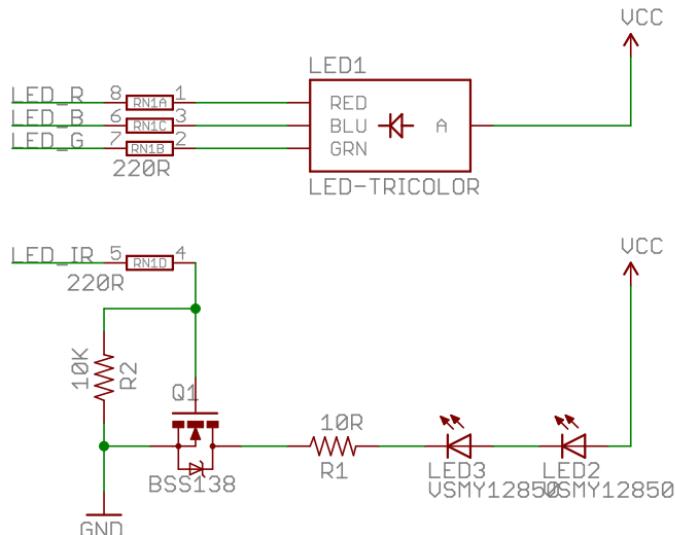


图 4-3 LED 原理图

上表对 MicroPython 的 LED 对象做了详细的说明，`pyb` 是大模块，`LED` 是 `pyb` 下面的其中一个小模块，在 `python` 编程里有两种方式引用相关模块：

方式 1 是：`import pyb`，然后通过 `pyb.LED` 来操作；

方式 2 是：`from pyb import LED`，意思是直接从 `pyb` 中引入 `LED` 模块，然后直接通过 `LED` 来操作。显然方式 2 会显得更直观和方便，本实验也是使用方式 2 来编程。代码编写流程如下：

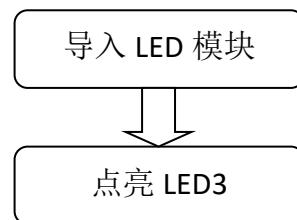


图 4-4 代码编写流程

参考代码：

实验名称：点亮 LED(3) 蓝灯

版本：v1.0

日期：2019.9

作者：01Studio

实验目的：学习 led 点亮。

...

```
from pyb import LED #导入 LED 模块  
LED(3).on()          #点亮 LED3， 蓝灯
```

### ● 实验结果：

将以上代码文件 main.py 拷贝到 OpenMV4 文件系统，等待红色灯从亮到灭（烧写完成）。按下开发板复位键重新启动。可以看到 LED 蓝灯被点亮。

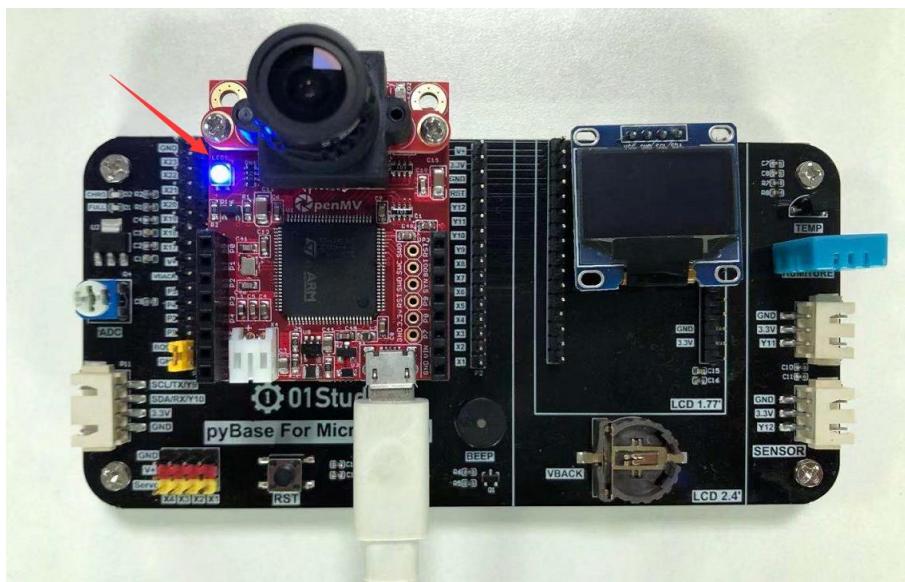


图 4-5 点亮 LED3 (蓝灯)

注意：上述代码无法直接通过 OpenMV ID 来运行，因为里面没有 `while True()` 死循环。

- **总结：**

从第一个实验我们可以看到，使用 MicroPython 来开发关键是要学会构造函数和其使用方法，便可完成对相关对象的操作，在强大的模块函数支持下，实验只用了简单的两行代码便实现了点亮 LED 灯。

## 4.2 流水灯

- **前言:**

通过上一节点亮 LED 灯的学习，我们已经对 OpenMV4 使用 micropython 的编程有了初步的了解，这一节来做一个功能稍微复杂一点的实验，流水灯。流水灯也叫跑马灯，也就是让几个 LED 来回亮灭，达到好像流水的效果。也是单片机开发学习的典型例子。

- **实验平台:**

pyAI-OpenMV4。

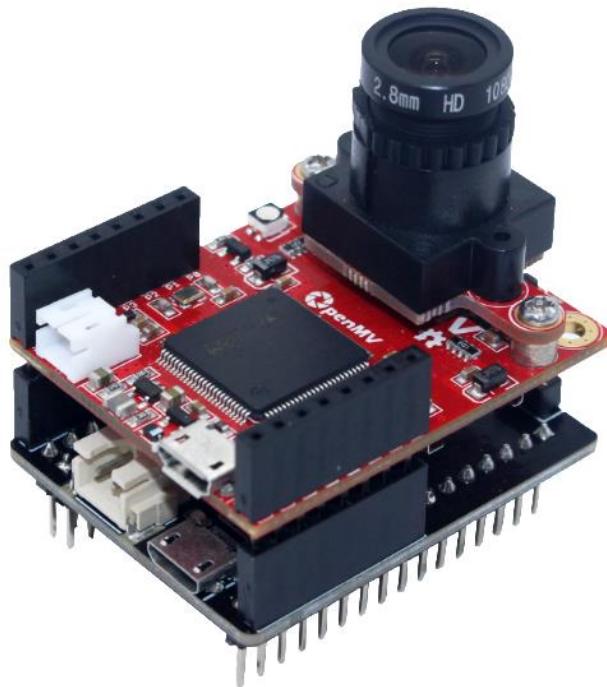


图 4-6 pyAI-OpenMV4

- **实验目的:**

流水灯。让 LED2-LED3 循环亮灭，达到像流水一样的效果。

- **实验讲解:**

OpenMV4 上总共有 4 个 LED，分别是 LED1(红色)、LED2(绿色)、LED3(蓝色)、LED4(IR 红外灯)；控制 LED 使用到 LED 对象。上一章节我们已经学习过 LED 点亮，这里要实现固定时间来亮灭，需要用到 utime 模块中的延时的函数。具体如下：

构造函数
<code>utime()</code>
时间模块，直接使用。
使用方法
<code>utime.sleep_ms(ms)</code>
毫秒级延时。 <code>ms</code> : 延时毫秒数。
<code>utime.sleep_us(us)</code>
微秒级延时。 <code>us</code> : 延时微秒数。
*更多用法请阅读 <code>openmv</code> 官方文档: 文档链接: <a href="http://docs.openmv.io/library/utime.html">http://docs.openmv.io/library/utime.html</a>

表 4-2 `utime` 时间对象

知道了延时函数的使用方法后，我们可以简单的梳理一下流程，首先导入 `LED` 和 `utime` 模块，程序开始先让 `LED2-LED3` 灭掉，开启循环，依次点亮每个 `LED`，延时 1 秒，关闭 `LED`。流程如下：

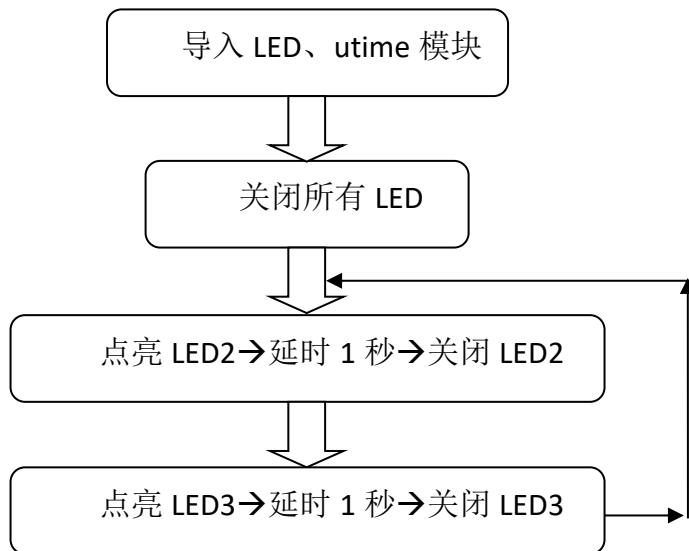


图 4-7 代码编写流程

以下是本实验参考程序代码：

```
...
实验名称：流水灯
版本：v1.0
日期：2019.9
作者：01Studio
...

from pyb import LED #从pyb导入LED模块
import utime

#关闭全部LED
LED(2).off()
LED(3).off()

#while True表示一直循环
while True:
    #LED2亮1秒
    LED(2).on()
    utime.sleep(1)
    LED(2).off()

    #LED3亮1秒
    LED(3).on()
    utime.sleep(1)
    LED(3).off()
```

上述代码没错是完整的按照编程思路来编写，但可以见到有很多格式相似的地方，这显得代码非常冗余。我们可以通过 `for` 函数来编写程序，由于是对 LED2、

LED3 的操作，因此我们可以用 `for i in range(2,4):` 语句来修改。参考代码如下：

参考代码：

```
...
实验名称：流水灯
版本：v2.0
日期：2019.9
作者：01Studio
...

from pyb import LED #从pyb导入LED模块
import utime

#相当于for i in [2, 3], LED(i).off()执行2次，分别是LED 2, 3
for i in range(2,4):
    LED(i).off()

while True:
    #使用for循环
    for i in range(2,4):
        LED(i).on()
        utime.sleep(1) #延时1秒
        LED(i).off()
```

### ● 实验结果：

将上述代码对应的 `main.py` 复制到文件系统，或者直接在 IDE 中运行，可以看到 `LED2` 和 `LED3` 也就是绿灯和蓝灯交替闪烁，像流水的效果。

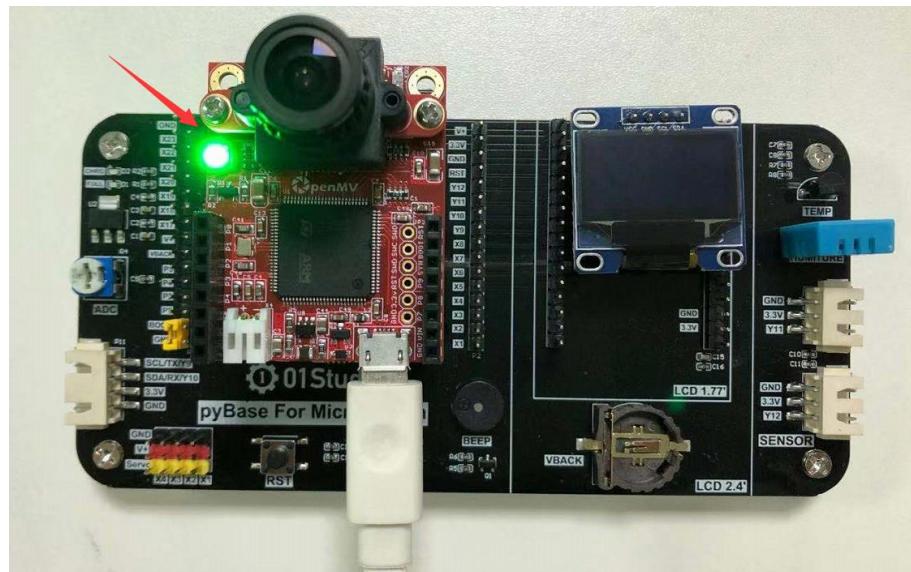


图 4-8

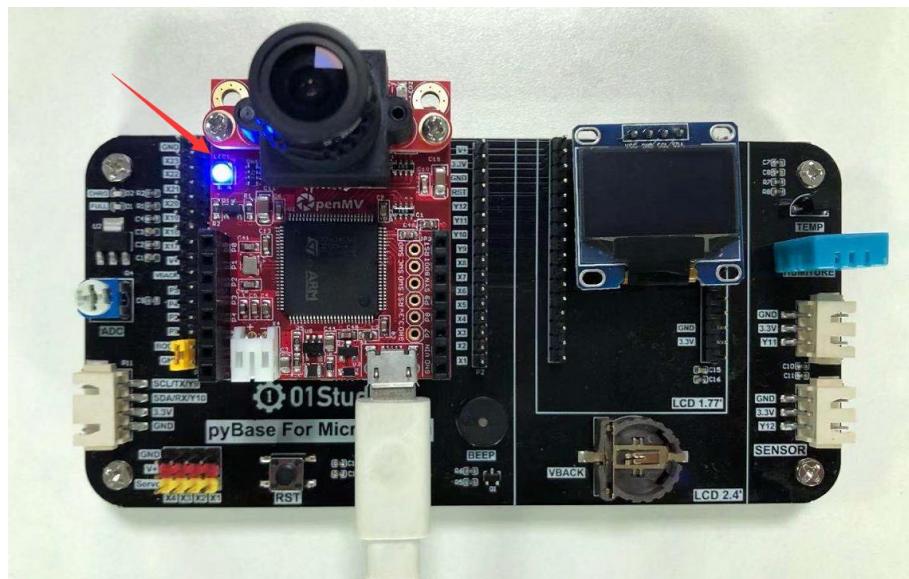


图 4-9

### ● 总结：

本节实验相对于第一节相比增加了延时函数的应用，可见 MicroPython 编程功能增加就是各类函数的叠加，当开发复杂功能时候，因为底层函数已经封装好，所以面向应用的开发使用起来非常方便。除此之外我们也体验了 Python 代码的简洁和高效，使得程序更好的阅读。

### 4.3 GPIO（按键）

- **前言：**

按键是最简单也最常见的输入设备，很多产品都离不开按键，包括早期的 iphone，今天我们就来学习一下如何使用 MicroPython 来编写按键程序。有了按键输入功能，我们就可以做很多好玩的东西了。

- **实验平台：**

pyAI-OpenMV4 开发板。



图 4-10 pyAI-OpenMV4

- **实验目的：**

学会使用微处理器的 GPIO 的输入/输出，使用 GPIO 方式来控制 LED 和按键。

- **实验讲解：**

OpenMV4 上引出了 P0-P9 共 10 个 GPIO（General Purpose Input/Output，通用输入输出口），基本上每个 GPIO 口都可以配置成特定的 GPIO 方式来进行应用。我们先来了解一下 GPIO（Pin 对象）的构造函数和使用方法：

## 构造函数

`pyb.Pin(id, mode, pull_mode)`

Pin 对象在 `pyb` 模块下。

【`id`】引脚号，如“P0”，“P1”；

【`mode`】输入输出模式选择

`Pin.IN` 输入模式，

`Pin.OUT_PP` 输出带推挽；

【`pull_mode`】上下拉电阻配置

`Pin.PULL_UP` 上拉电阻，

`Pin.PULL_DOWN` 下拉电阻，

`Pin.PULL_NONE` 没上下拉电阻；

例：`KEY = Pin('P9', Pin.IN, Pin.PULL_UP)` #将按键"P9"配置为输入方式

## 使用方法

`Pin.high()`

引脚输出高电平。

`Pin.low()`

引脚输出低电平。

`Pin.value()`

获取当前引脚的输入电平，返回 0（低电平）或者 1（高电平）。仅在引脚配置为输入模式有效。

表 4-3 Pin 对象 (GPIO)

我们来看看 pyAI-OpenMV4 的原理图，看看按键连接的引脚为 P9。

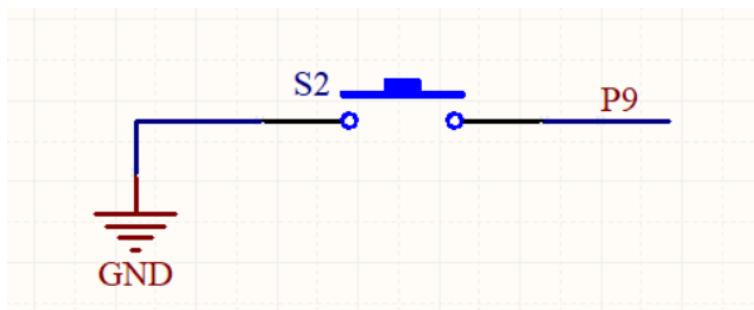


图 4-11 按键接线图

GPIO 对象使用非常简单，我们将按键即“P9”引脚配置成输入，实现当检测到按键被按下时候点亮 LED (2)，松开时关闭 LED (2)。代码编写流程如下：

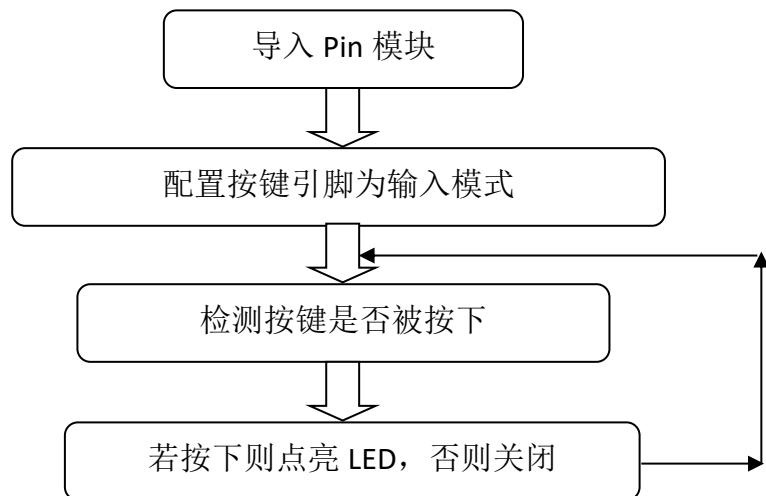


图 4-12 代码编写流程

参考代码：

```
...
实验名称: GPIO (按键)
版本: v1.0
日期: 2019.9
作者: 01Studio
社区: www.01studio.org
...

from pyb import Pin,LED

#将 KEY 按键"P9"配置为输入方式
KEY = Pin('P9', Pin.IN, Pin.PULL_UP)
```

```
while True:  
  
    if KEY.value()==0: #按键被按下接地  
        LED(3).on()      #点亮 LED (3) 蓝灯  
  
    else:  
        LED(3).off()     #关闭 LED (3) 蓝灯
```

### ● 实验结果：

运行代码，可以看到当按键被按下时候，LED(3)蓝灯点亮，松开时熄灭。按键可以按下 pyAI-OpenMV4 顶部的按键或者 pyBase 开发底板上的 KEY 按键。

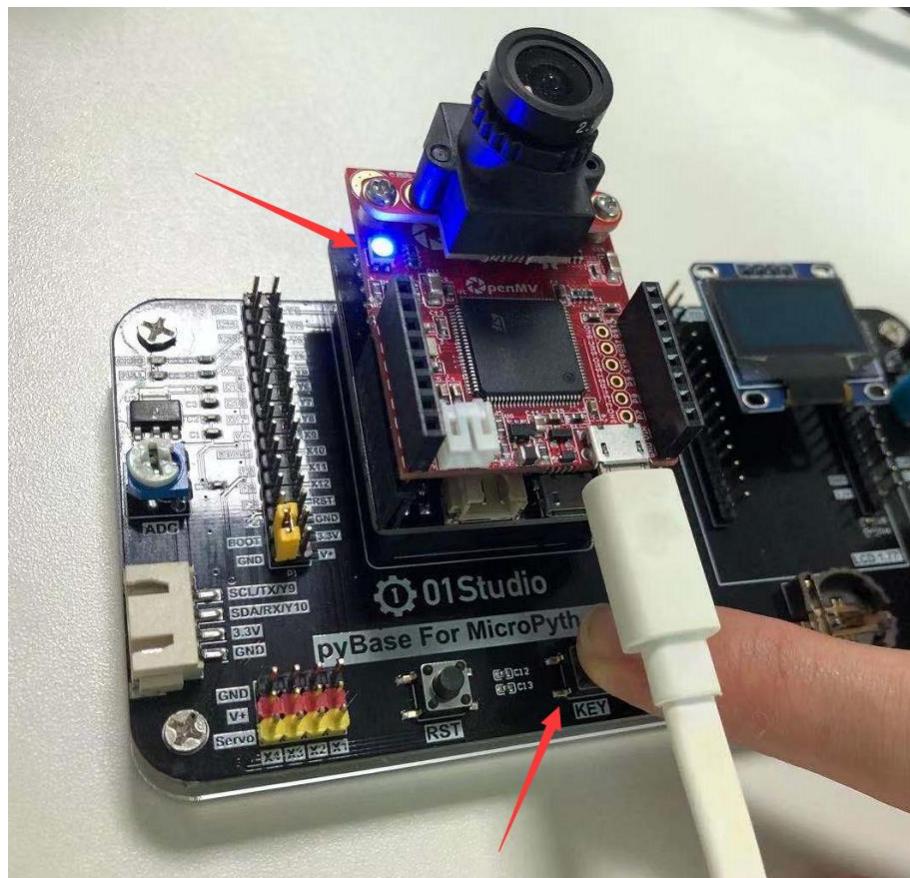


图 4-13 GPIO 按键实验

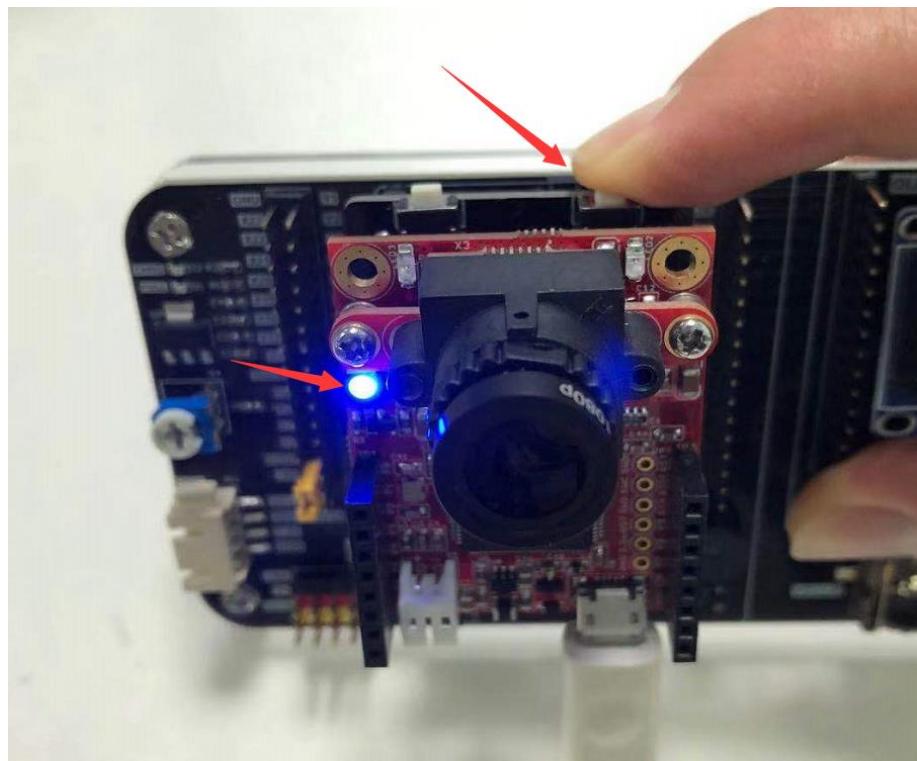


图 4-14 上方的按键

● 总结：

GPIO 是非常通用的实验和功能，学会了 GPIO，就可以把开发板所有的引脚为自己所用，灵活性很强。

## 4.4 外部中断

- **前言：**

前面我们在做普通的 GPIO 时候，虽然能实现 IO 口输入输出功能，但代码是一直在检测 IO 输入口的变化，因此效率不高，特别是在一些特定的场合，比如某个按键，可能 1 天才按下一次去执行相关功能，这样我们就浪费大量时间来实时检测按键的情况。

为了解决这样的问题，我们引入外部中断概念，顾名思义，就是当按键被按下(产生中断)时，我们才去执行相关功能。中断的应用非常普遍，而 OpenMV4 上基本每个 IO 口都具备中断功能。

- **实验平台：**

pyAI-OpenMV4。

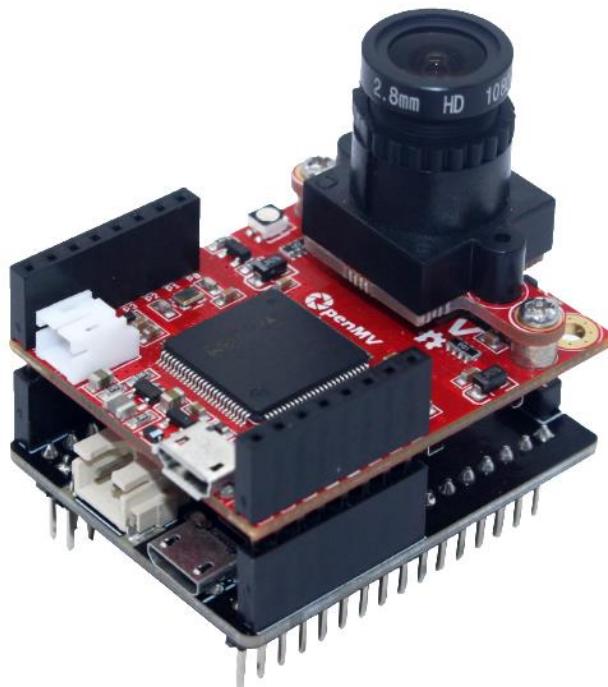


图 4-15 pyAI-OpenMV4

- **实验目的：**

利用中断方式来检查按键 KEY 状态，被按键被按下(产生外部中断)后使 LED (2) 蓝灯的亮灭状态反转。

### ● 实验讲解：

从上一节 GPIO 实验我们知道按键是连接到 P9 引脚，我们先来看看外部中断的用法。

构造函数
<code>pyb.ExtInt(pin,mode,pull_mode,callback)</code>
外部中断 ExtInt 对象在 pyb 模块下。
【pin】引脚号，如“P0”，“P1”；
【mode】中断触发方式
<code>ExtInt.IRQ_RISING</code> : 上升沿触发
<code>ExtInt.IRQ_FALLING</code> : 下降沿触发
<code>ExtInt.IRQ_RISING_FALLING</code> : 上升或下降沿触发
【pull_mode】上下拉电阻设置
<code>Pin.PULL_NONE</code> : 无上下拉电阻；
<code>Pin.PULL_UP</code> : 启用上拉电阻；
<code>Pin.PULL_DOWN</code> : 启用下拉电阻；
【callback】中断后执行的回调函数。

表 4-4 外部中断对象

我们先来了解一下上升沿和下降沿的概念，由于按键 KEY 引脚是通过按键接到 GND，也就是我们所说的低电平“0”，所以当按键被按下再松开时，引脚先获得下降沿，再获得上升沿，如下图所示：

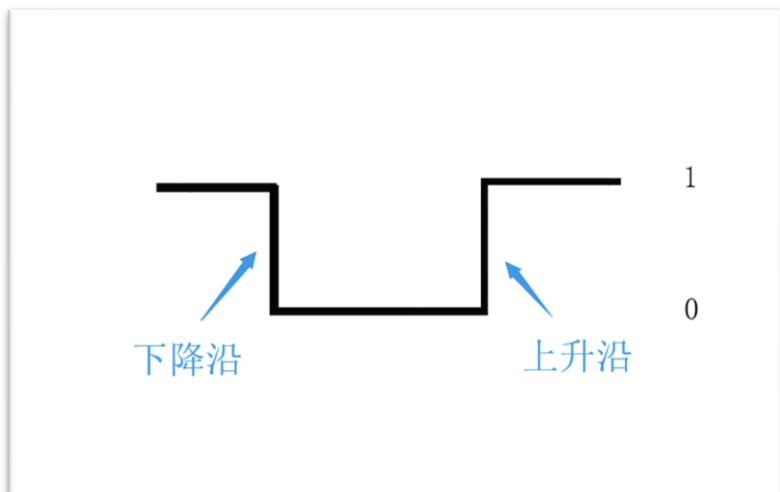


图 4-16 上升沿和下降沿

由此可见，我们可以选择下降沿方式触发外部中断，也就是当按键被按下的时候立即产生中断。

编程思路中断跟 GPIO 按键章节类似，在初始化中断后，当系统检测到外部中断时候，执行 LED 状态反转的代码即可。

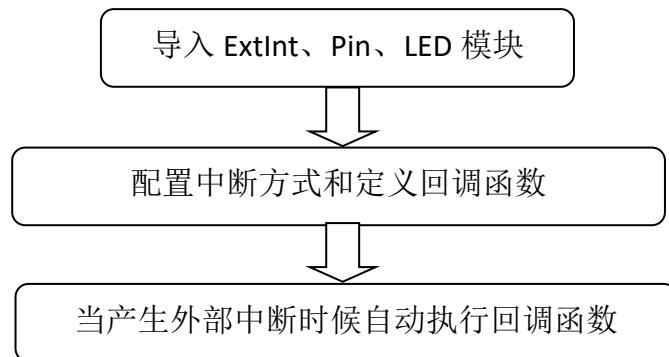


图 4-17 代码编写流程

参考代码：

```
...
实验名称: 外部中断
版本: v1.0
日期: 2019.9
作者: 01Studio
社区: www.01studio.org
...
from pyb import Pin,ExtInt,LED

callback = lambda e: LED(3).toggle()
#下降沿触发，打开上拉电阻
ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, callback)
```

- 实验结果：

烧录程序，可以看到每次按下按键时候，LED3 蓝灯的亮灭状态翻转。按键可以使用 pyAI-OpenMV4 顶部的按键或者 pyBase 开发底板上的 KEY 按键。（注：中断无法通过 IDE 运行来执行。）

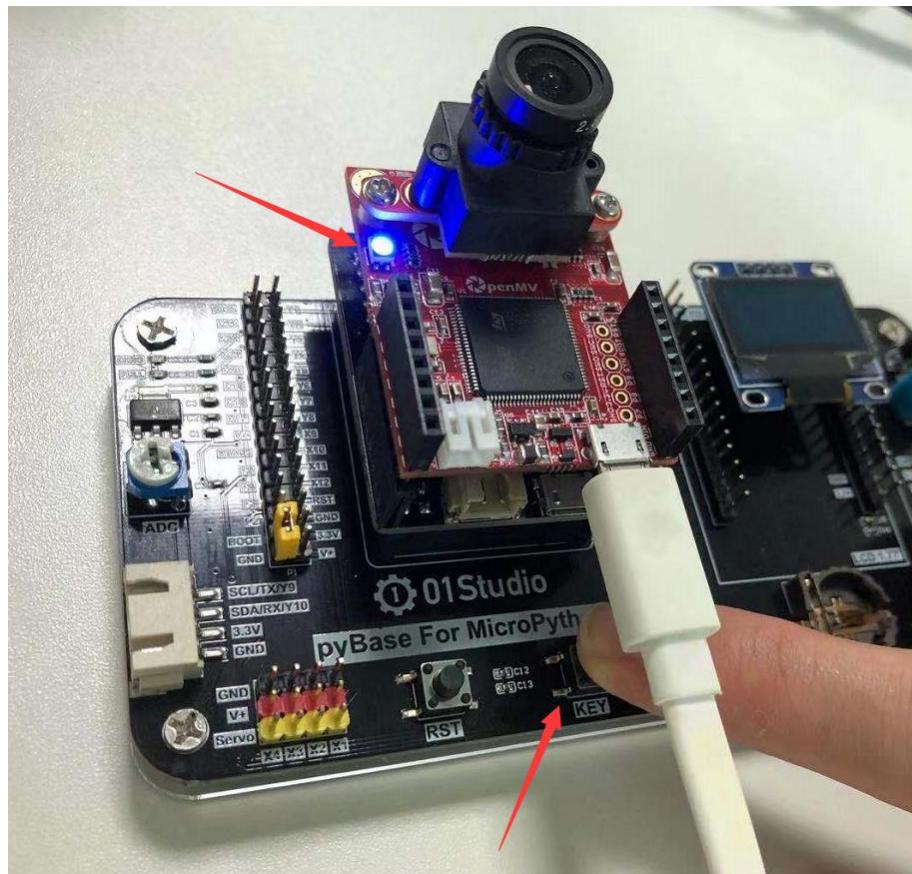


图 4-18 GPIO 按键实验

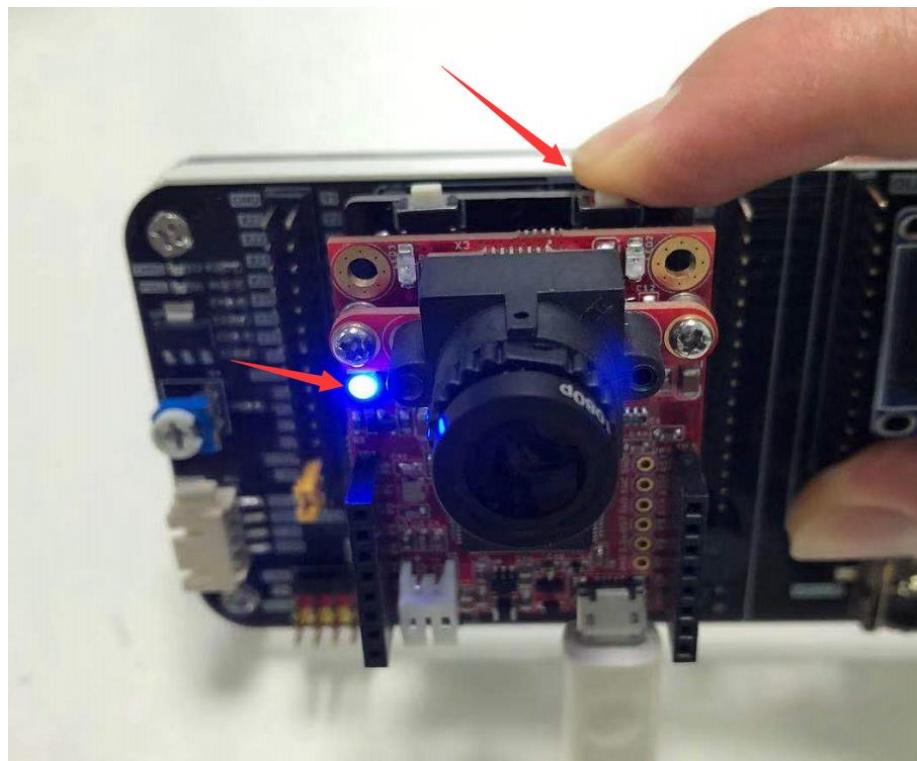


图 4-19 上方的按键

● 总结：

从参考代码来看，只是用了 3 行代码就实现了实验功能，而且相对于使用 `while True` 实时检测函数来看，代码的效率大大增强。外部中断的应用非常广，除了普通的按键输入和电平检测外，很大一部分输入设备，比如传感器也是通过外部中断方式来实时检测，从而提供 CPU 的利用率。

## 4.5 定时器

- **前言:**

定时器，顾名思义就是用来计时的，我们常常会设定计时或闹钟，然后时间到了就告诉我们要做什么了。单片机也是这样，通过定时器可以完成各种预设好的任务。

- **实验平台:**

pyAI-OpenMV4。

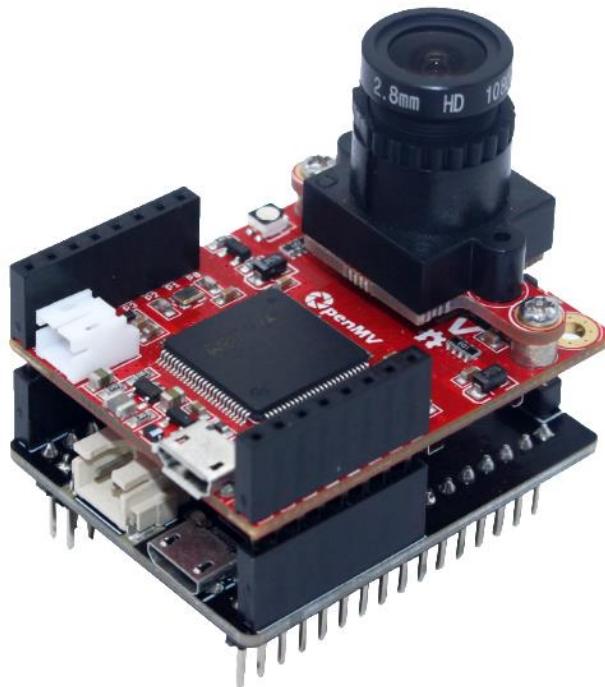


图 4-20 pyAI-OpenMV4

- **实验目的:**

通过定时器让 LED(3)蓝灯周期性每秒闪烁 1 次。

- **实验讲解:**

定时器模块 Timer 在 `pyb` 模块中。对象说明如下：

构造函数
<code>pyb.Timer(id,freq,.....)</code>
定时器对象 <code>Timer</code> 对象在 <code>pyb</code> 模块下。

【id】定时器编号，1-14;
【freq】定时器中断频率
使用方法
Timer.callback(fun)
定义执行的回调函数。
Timer.deinit()
终止定时器。
*更多用法请阅读 openmv 官方文档： 文档链接： <a href="http://docs.openmv.io/library/pyb.Timer.html">http://docs.openmv.io/library/pyb.Timer.html</a>

表 4-5 定时器对象

定时器到了预设指定时间后，也会产生中断，因此跟外部中断的编程方式类似，代码编程流程图如下：

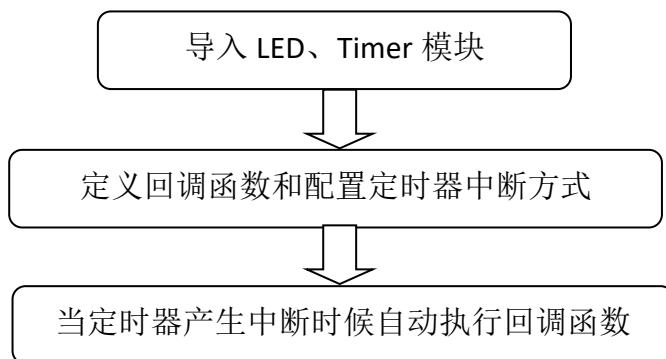


图 4-21 代码编写流程

参考代码如下：

```

...
实验名称: 定时器 (蓝灯周期性闪烁)
版本: v1.0
日期: 2019.9
作者: 01Studio
...
  
```

```
import pyb

tim = pyb.Timer(4,freq=1)      # 使用定时器 4 创建定时器对象,频率 1Hz

#定时器中断回调函数, 执行 LED (3) 蓝灯状态反转

tim.callback(lambda t:pyb.LED(3).toggle())
```

### ● 实验结果:

烧录程序，可以看到 LED(2)蓝灯每隔 1 秒闪烁 1 次。（注：中断无法通过 IDE 运行来执行。）

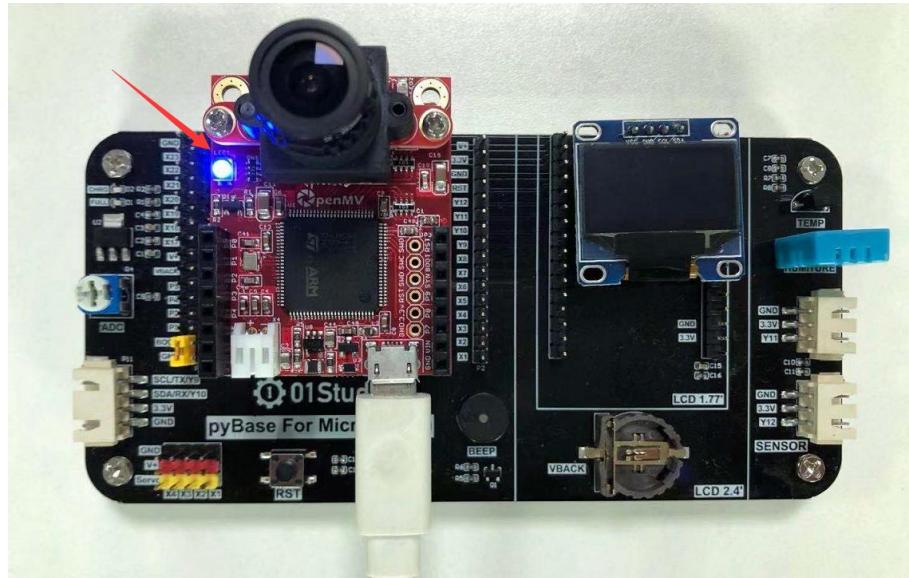


图 4-22

### ● 总结:

本节实验介绍了定时器的使用方式，有用户可能会认为使用延时延时也可以实现这个功能，但相比于延时函数，定时器的好处就是不占用过多的 CPU 资源。

有兴趣的用户也可以多定义几个定时器对象 `tim2,tim3`，通过不同的参数配置实现多任务的操作。

## 4.6 I2C 总线（OLED 显示屏）

### ● 前言：

在上一节学习了按键输入设备后，我们这一节先来学习输出设备 OLED 显示屏，其实之前的 LED 灯也算是输出设备，因为它们确切地告诉了我们硬件的状态。只是相对于只有亮灭的 LED 而言，显示屏可以显示更多的信息，体验更好。

本章节的 OLED 显示屏学习，实际上是在使用 I2C 的总线接口，pyAI-OpenMV4 是通过 I2C 总线与 OLED 显示屏通讯的。这章稍微复杂一点，我们把它放在了前面来学习，是因为学会了显示屏的使用，那么在后面的实验中可玩性就更强了。

### ● 实验平台：

pyAI-OpenMV4 开发套件。

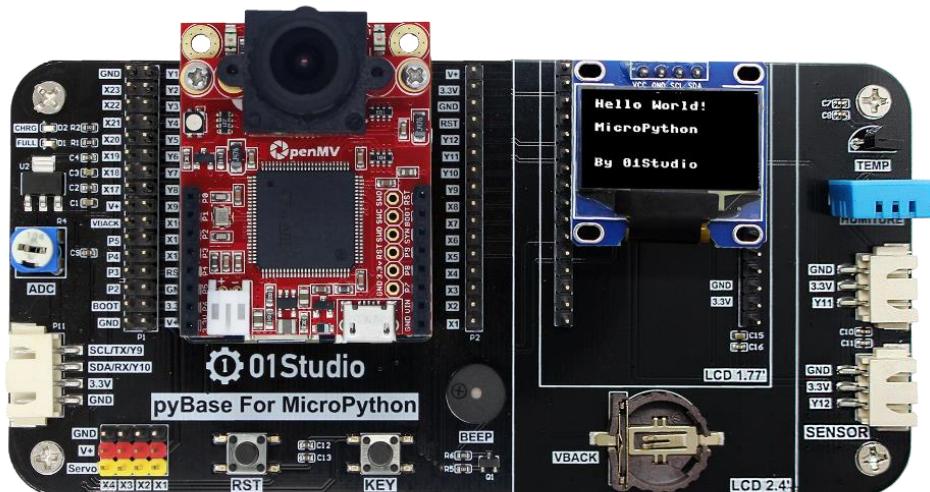


图 4-23 pyAI-OpenMV4 开发套件

### ● 实验目的：

学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。

### ● 实验讲解：

#### 什么是 I2C？

I2C 是用于设备之间通信的双线协议，在物理层面，它由 2 条线组成：SCL 和 SDA，分别是时钟线和数据线。也就是说不通设备间通过这两根线就可以进行通信。

## 什么是 OLED 显示屏？

OLED 的特性是自己发光，不像 TFT LCD 需要背光，因此可视度和亮度均高，其次是电压需求低且省电效率高，加上反应快、重量轻、厚度薄，构造简单，成本低等特点。简单来说跟传统液晶的区别就是里面像素的材料是由一个个发光二极管组成，因为密度不高导致像素分辨率低，所以早期一般用作户外 LED 广告牌。随着技术的成熟，使得集成度越来越高。小屏也可以制作出较高的分辨率。



图 4-24 I2C 接口的 OLED

在了解完 I2C 和 OLED 显示屏后，我们先来看看开发板的原理图，也就是 MicroPython 上的 OLED 接口是如何连线的。下图是 pyBase 开发底板的原理图。

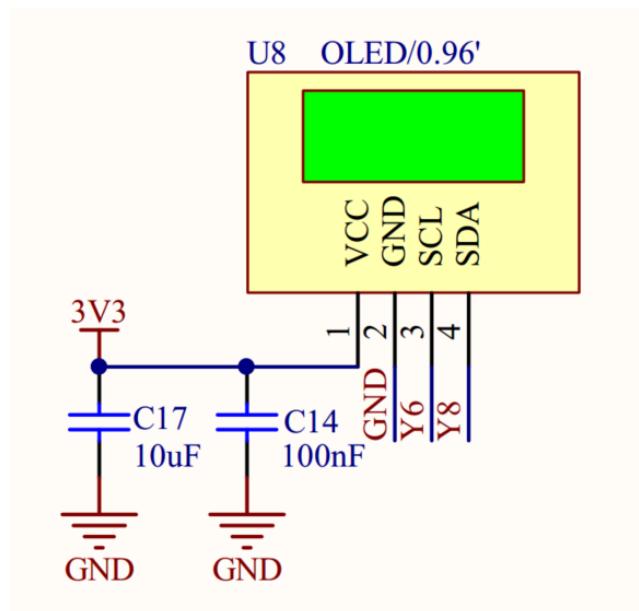


图 4-25 OLED 接口原理图

我们再来看看 pyAI-OpenMV4 转接板的原理图接口部分。

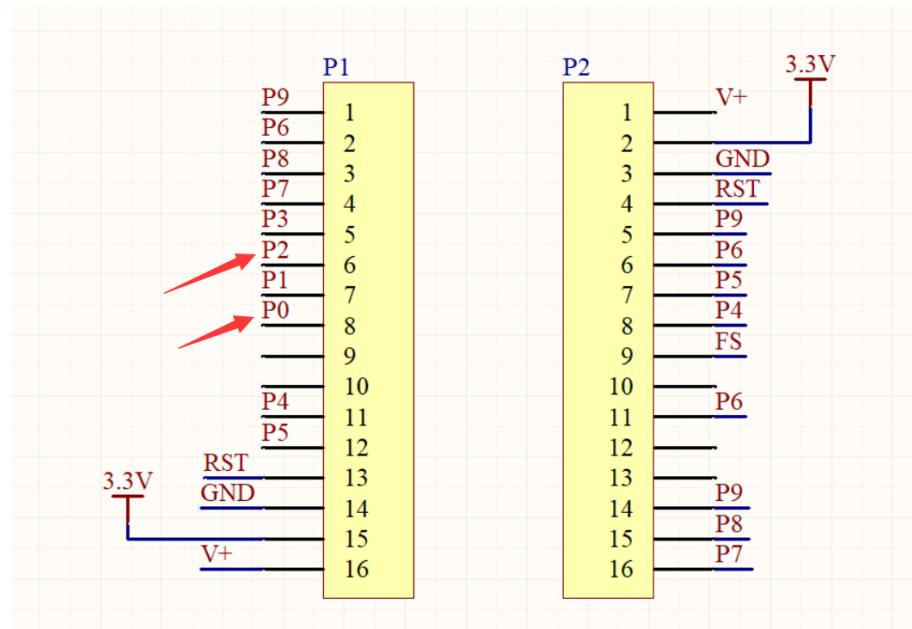


表 4-6 pyAI-OpenMV4 转接板接口图

结合以上可以得知 pyBase 底板连接到 OLED 的对应关系是 Y6→SCL 和 Y8→SDA。对应 OpenMV4 的关系是：P2→Y6→SCL，P0→Y8→SDA。尽管没有连接到自带的 I2C 接口，我们也可以通过软件模拟 I2C 方式来实现 OLED 的数据通信和使用。本例程将使用 MicroPython 的 Machine 模块来定义 Pin 口和 I2C 初始化。具体如下：

构造函数
<pre>i2c = machine.I2C(scl,sda,freq)</pre>
构建 I2C 对象。 【scl】时钟引脚； 【sda】数据引脚； 【freq】通信频率，即速度。
使用方法
<pre>i2c.scan()</pre>
扫描 I2C 总线的设备。返回地址，如：0x3c；

<code>i2c.readfrom(addr, nbytes)</code>
从指定地址读数据。 <code>addr</code> :指定设备地址; <code>nbytes</code> :读取字节数;
<code>i2c.write(buf)</code>
写数据。 <code>buf</code> :数据内容;
*其它更多用法请阅读官方文档: 文档链接: <a href="http://docs.openmv.io/library/pyb.I2C.html">http://docs.openmv.io/library/pyb.I2C.html</a>

表 4-7 I2C 对象

定义好 I2C 后，还需要驱动一下 OLED。这里我们已经写好了 OLED 的库函数，在 `ssd1306x.py` 文件里面。开发者只需要将改 `py` 文件拷贝到 OpenMV4 文件系统里面，然后在 `main.py` 里面调用函数即可。人生苦短，我们学会调用函数即可，也就是注重顶层的应用，想深入的小伙伴也可以自行研究 `ssd1306.py` 文件代码。OLED 显示屏对象介绍如下：

构造函数
<code>oled = SSD1306_I2C(width, height, i2c, addr)</code>
构 OLED 显示屏对象。
【width】 屏幕宽像素；
【height】 屏幕高像素；
【i2c】 定义好的 I2C 对象；
【addr】 显示屏设备地址。
使用方法
<code>oled.text(string,x,y)</code>
将 <code>string</code> 字符写在指定为位置。 <code>string</code> : 字符; <code>x</code> :横坐标; <code>y</code> :纵坐标。
<code>oled.show()</code>
执行显示。
<code>oled.fill(RGB)</code>
清屏。 <code>RGB</code> : 0 表示黑色, 1 表示白色。

表 4-8 OLED 对象

学习了 I2C、OLED 对象用法后我们通过编程流程图来理顺一下思路：

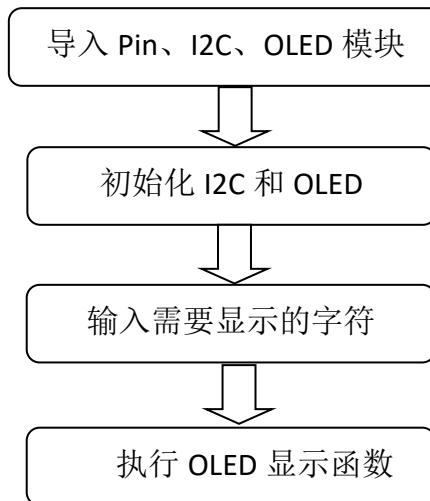


图 4-26 代码编写流程

参考代码如下：

实验名称: OLED 显示屏 (I2C 总线)  
版本: v1.0  
日期: 2019.9  
作者: 01Studio  
...  
  
from machine import I2C,Pin #从 machine 模块导入 I2C、Pin 子模块  
from ssd1306x import SSD1306\_I2C #从 ssd1306 模块中导入 SSD1306\_I2C 子模块  
  
#I2C 初始化: sda--> P0, scl --> P2, 频率 8MHz  
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"), freq=80000 )  
#OLED 显示屏初始化: 128\*64 分辨率,OLED 的 I2C 地址是 0x3c  
oled = SSD1306\_I2C(128, 64, i2c, addr=0x3c)  
  
oled.text("Hello World!", 0, 0) #写入第 1 行内容  
oled.text("MicroPython", 0, 20) #写入第 2 行内容

```
oled.text("By 01Studio", 0, 50)      #写入第3行内容  
  
oled.show()    #OLED 执行显示
```

上述代码中 OLED 的 I2C 地址是 0x3C,不同厂家的产品地址可能预设不一样，具体参考厂家的说明书。或者也可以通过 I2C.scan()来获取设备地址。

另外记得将我们提供的示例代码中的 ssd1306x.py 驱动文件拷贝到 OpenMV4 的文件系统下，跟 main.py 保持同一个路径。

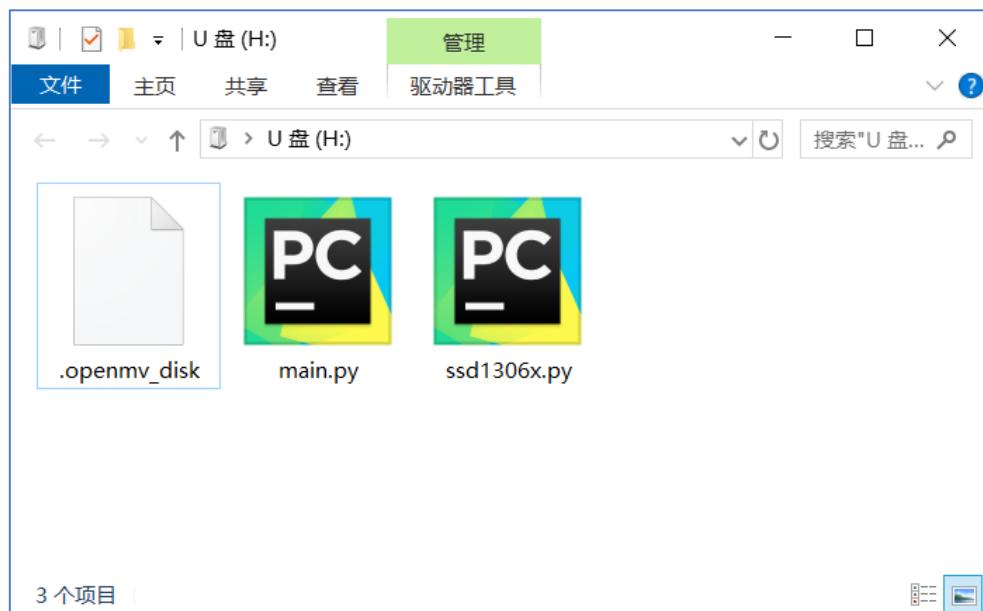


图 4-27 将 main.py、ssd1306x.py 拷贝到文件系统

### ● 实验结果：

运行代码，可以看到实验现象如下：

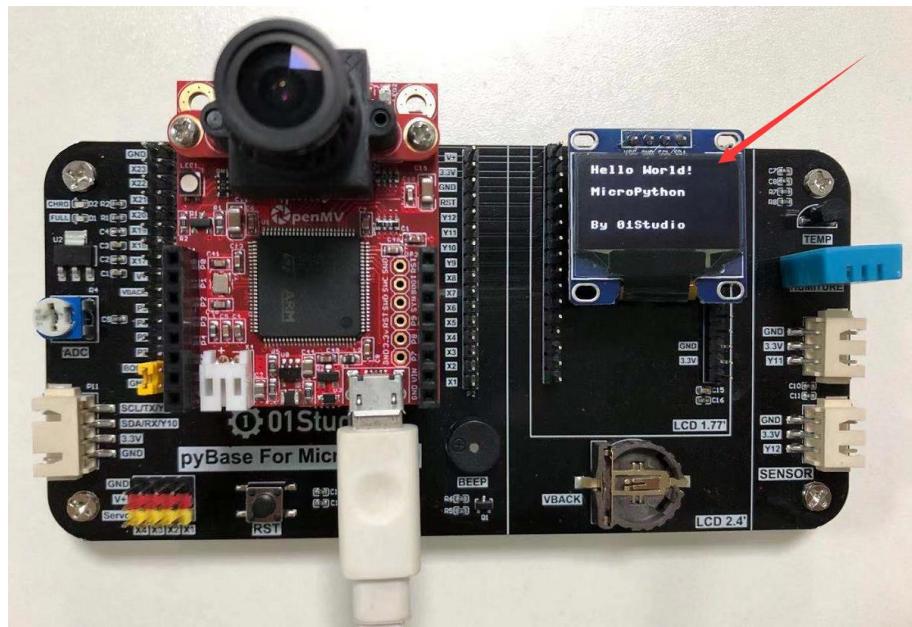


图 4-28 实验结果

### ● 总结：

这一节我们学会了驱动 OLED 显示屏，换着以往如果从使用单片机从 0 开发的话你需要了解 I2C 总线原理，了解 OLED 显示屏的使用手册，编程 I2C 代码，有经验的嵌入式工程师搞不好也要弄个几天。现在基本半个小时解决问题。当然前提是别人已经给你搭好桥了，有了强大的底层驱动代码支持，我们只做好应用就好。

这一节学习的意义不仅在完成实验。在学习完 OLED 显示屏实验后，接下来我们的实验都可以使用这个 OLED 来跟用户交互了，这大大提高了实验的可观性。

## 4.7 RTC 实时时钟

- 前言：

时钟可以说我们日常最常用的东西了，手表、电脑、手机等等无时无刻不显示当前的时间。



图 4-29 时钟

可以说每一个电子爱好者心中都希望拥有属于自己制作的一个电子时钟，接下来我们就用 MicroPython 开发板来制作一个属于自己的电子时钟。

- 实验平台：

pyAI-OpenMV4 开发套件。

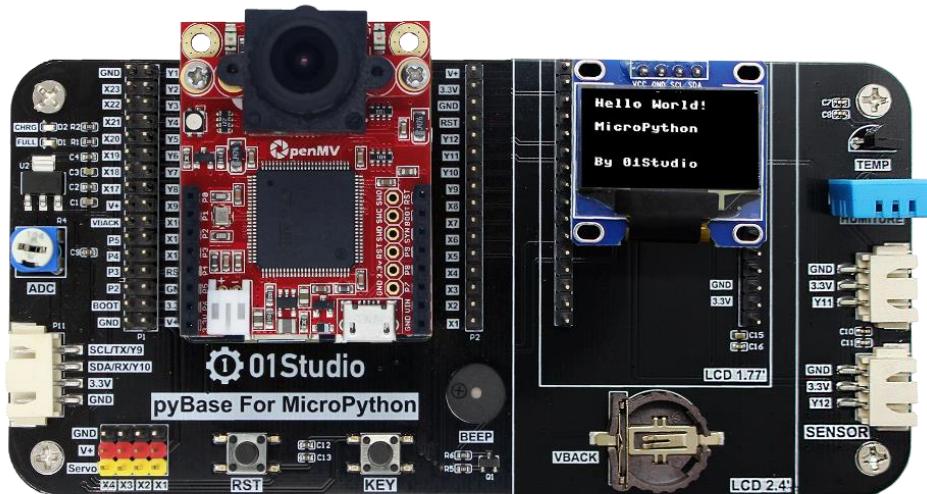


图 4-30 pyAI-OpenMV4 开发套件

### ● 实验目的:

学习 RTC 编程和制作电子时钟，使用 OLED 显示。

### ● 实验讲解:

实验的原理是读取 RTC 数据，然后通过 OLED 显示。毫无疑问，强大的 pyBoard 已经集成了内置时钟函数模块。具体如下：

构造函数
<code>pyb.RTC()</code>
RTC()是内置的实时时钟函数模块。
使用方法
<code>RTC().datetime((2019,4,1,1,0,0,0,0))</code>
设置日期时间： 顺序分别是年，月，日，星期，时，分，秒，次级秒。 星期：1-7 表示周一至周日； 次级秒：255-0 倒着计数。
<code>RTC().datetime()</code>
获取当前日期时间数据；

表 4-9 RCT 实时时钟对象

从上表可以看到 RTC () 的使用方法，我们需要做的就是先设定时间，然后再获取当前芯片里的时间，通过 OLED 显示屏显示，如此循环。在循环里，如果一直获取日期时间数据会造成资源浪费，所以可以每隔第一段时间获取一次数据，又由于肉眼需要看到至少每秒刷新一次即可，这里每隔 300ms 获取一次数据，所以具体流程如下：

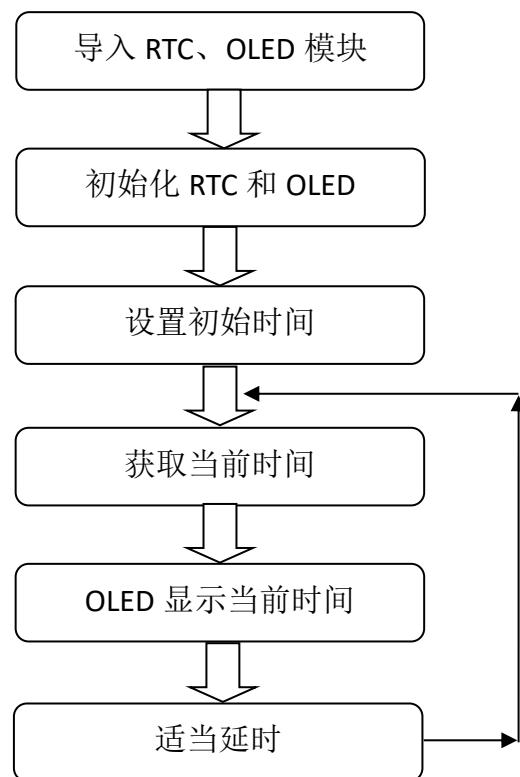


图 4-31 代码编写流程

参考代码如下：

```
...  
实验名称: RTC 实时时钟  
版本: v1.0  
日期: 2019.9  
作者: 01Studio  
...  
  
# 导入相关模块  
from pyb import RTC  
from machine import Pin, I2C  
from ssd1306x import SSD1306_I2C  
import utime
```

```

# 定义星期和时间（时分秒）显示字符列表

week = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun']

time = ['', '', '']


# 初始化所有相关对象

i2c = I2C(sda=Pin("P0"), scl=Pin("P2"), freq=80000) #频率 8MHz

oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

rtc = RTC()


# 首次上电配置时间，按顺序分别是：年，月，日，星期，时，分，秒，次秒级；这里做
#了一个简单的判断，检查到当前年份不对就修改当前时间，用户可以根据自己实际情况来
# 修改。

if rtc.datetime()[0] != 2019:
    rtc.datetime((2019, 4, 1, 1, 0, 0, 0, 0))

while True:

    datetime = rtc.datetime() # 获取当前时间

    oled.fill(0) # 清屏显示黑色背景

    oled.text('01Studio', 0, 0) # 首行显示 01Studio

    oled.text('RTC Clock', 0, 15) # 次行显示实验名称


    # 显示日期，字符串可以直接用“+”来连接

    oled.text(str(datetime[0]) + '-' + str(datetime[1]) + '-' +
              str(datetime[2]) + ' ' + week[(datetime[3] - 1)], 0, 40)


    # 显示时间需要判断时、分、秒的值否小于 10，如果小于 10，则在显示前面补“0”以达
    # 到较佳的显示效果

    for i in range(4, 7):

```

```

if datetime[i] < 10:
    time[i - 4] = "0"
else:
    time[i - 4] = ""

# 显示时间
oled.text(time[0] + str(datetime[4]) + ':' + time[1] +
str(datetime[5]) + ':' + time[2] + str(datetime[6]), 0, 55)

oled.show()

utime.sleep_ms(300) #延时 300ms

```

● 实验结果：

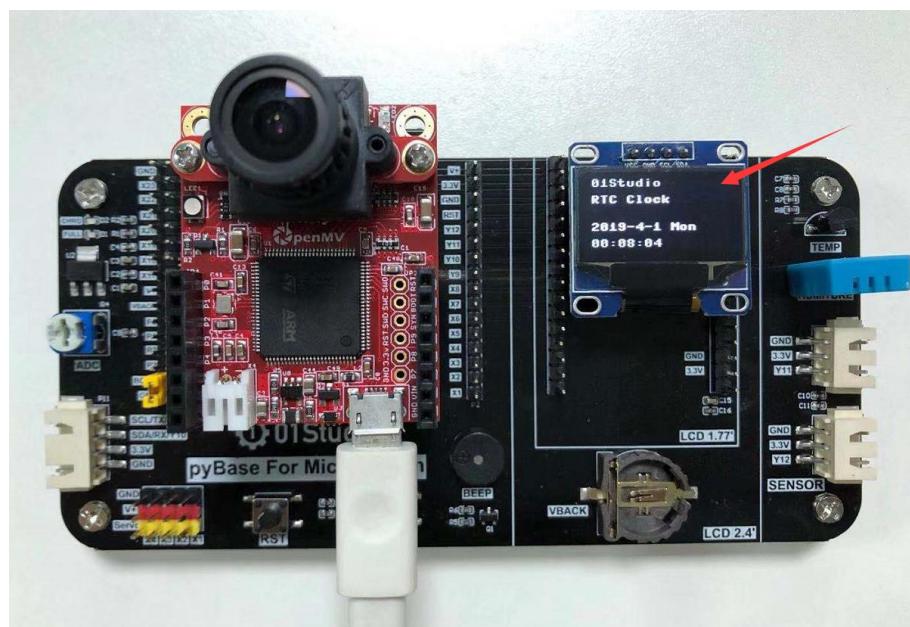


图 4-32 实验结果

由于 OpenMV4 没有引出后备电池引脚，所以不支持掉电保存。因此 pybase 上面的纽扣电池是不起作用的。

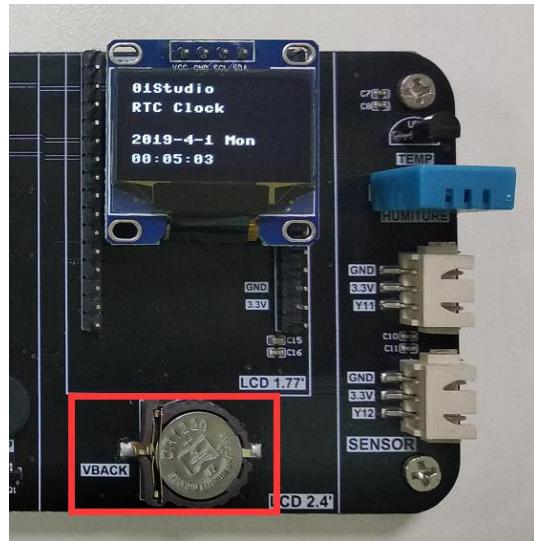


图 4-33 后备电池 (VBACK)

● 总结：

RTC 实时时钟的可玩性很强，我们还可以根据自己的风格来设定数字显示位置，以及加上一些属于自己的字符标识。打造自己的电子时钟。

## 4.8 ADC

### ● 前言：

ADC(analog to digital conversion) 模拟数字转换。意思就是将模拟信号转化成数字信号，由于单片机只能识别二级制数字，所以外界模拟信号常常会通过 ADC 转换成其可以识别的数字信息。常见的应用就是将变化的电压转成数字信号。

### ● 实验平台：

pyAI-OpenMV4 开发套件。

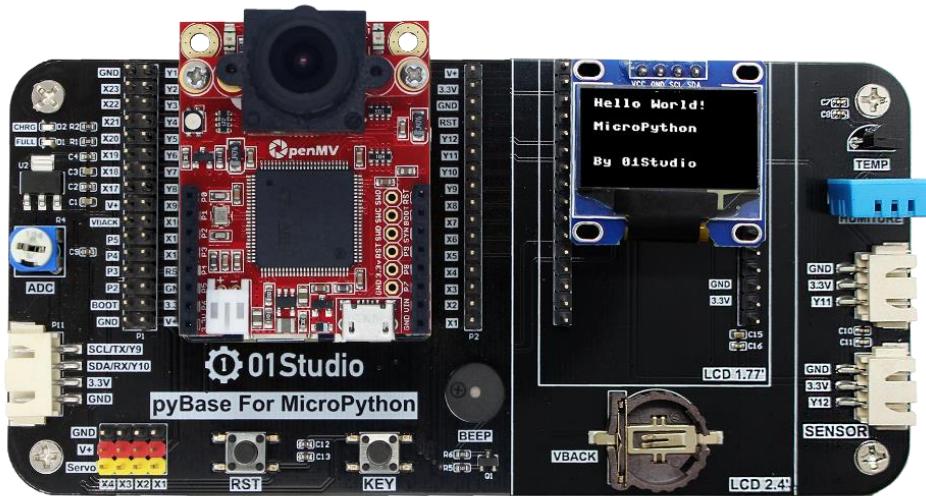


图 4-34 pyAI-OpenMV4 开发套件

### ● 实验目的：

通过编程调用 MicroPython 的内置 ADC 函数，实现测量 0-3.3V 电压，并显示到 OLED 屏幕上。

### ● 实验讲解：

pyBase 开发底板的的 X7 引脚连接到了电位器，通过电位器的调节可以使得 X7 引脚上的电压变化范围实现从 0-3.3V。

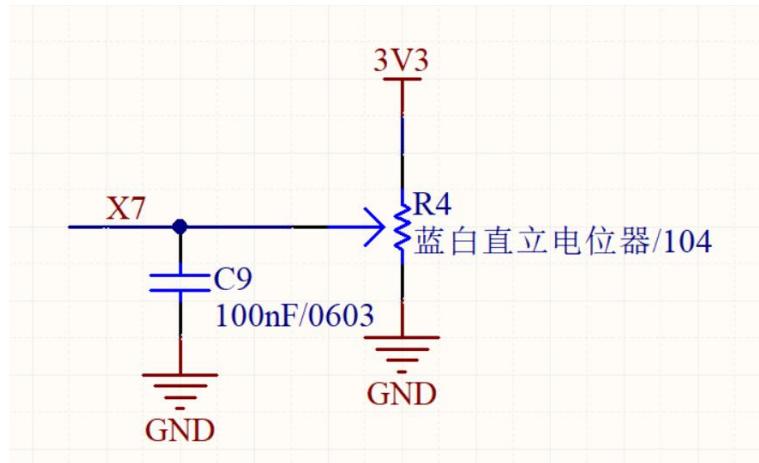


图 4-35 电位器原理图

我们再来看看 pyAI-OpenMV4 的引脚情况，其对用的 X7 是悬空，但 P6 是接在 pyBase 的 X6 上。

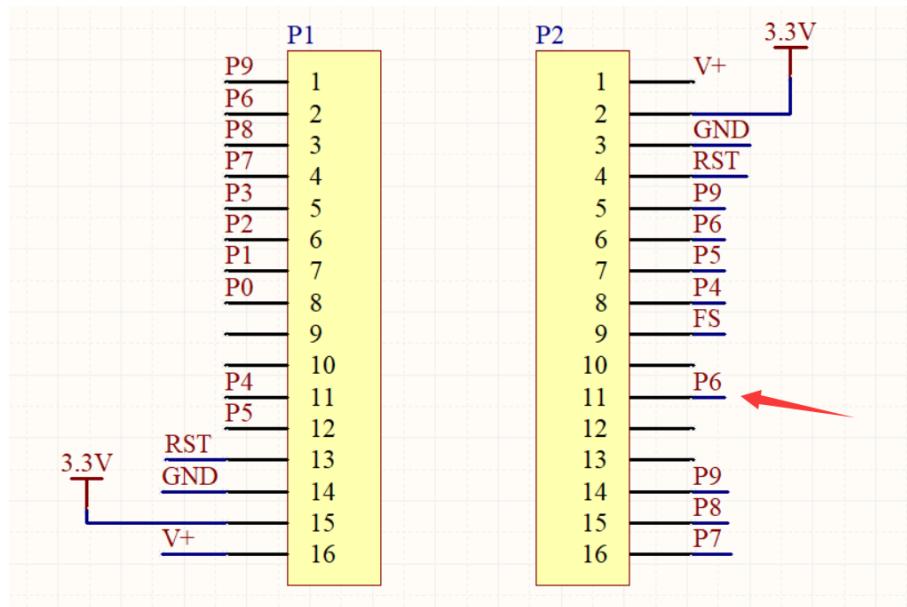


图 4-36 P6 连接到 X6

因此我们可以用一个跳线帽或者跳线将 pyBase 开发底板上的 X6 跟 X7 相连接，从而使用 P6 引脚编程实现 AD 数据采集。

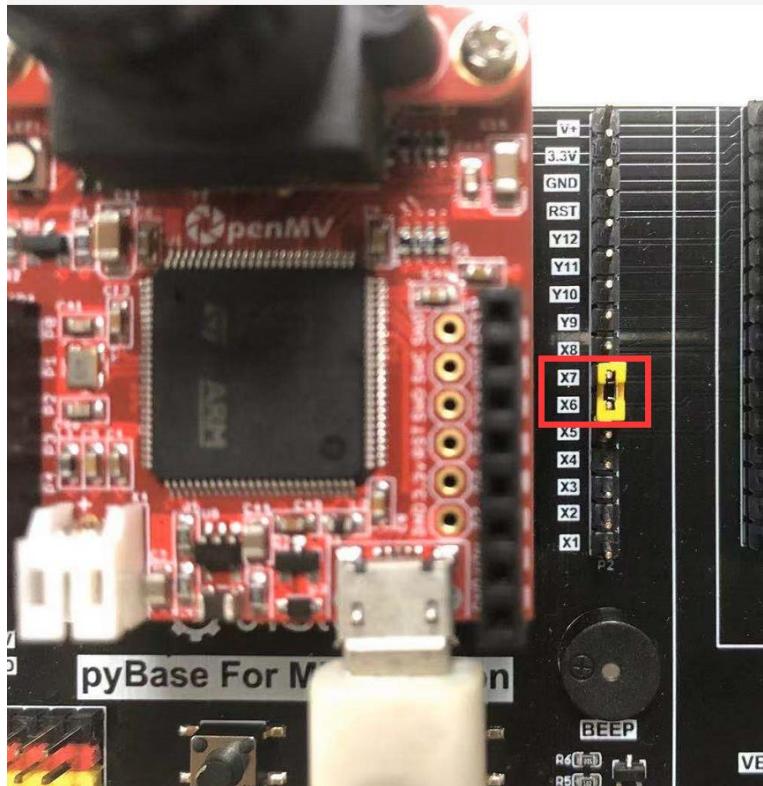


图 4-37 使用跳线帽将 X6 和 X7 短接

在了解了接线后，我们来看看内置 ADC 模块的函数使用方法。

构造函数
<code>pyb.ADC(pin)</code>
ADC 对象在 pyb 模块下。pin: ADC 输入引脚
使用方法
<code>ADC.read()</code>
读取 AD 值，返回 0-4095 (0v-3.3V)，自带 ADC 的测量精度是 12 位， $2^{12}=4096$

表 4-10 ADC 对象

你没看错，就这么简单。两句函数就可以获得 AD 数值，让我们来理顺一下编程逻辑。先导入相关模块，然后初始化模块。在循环中不断读取 ADC 的值，转化成电压值后在 OLED 上面显示，每隔 1 秒读取一次，具体如下：

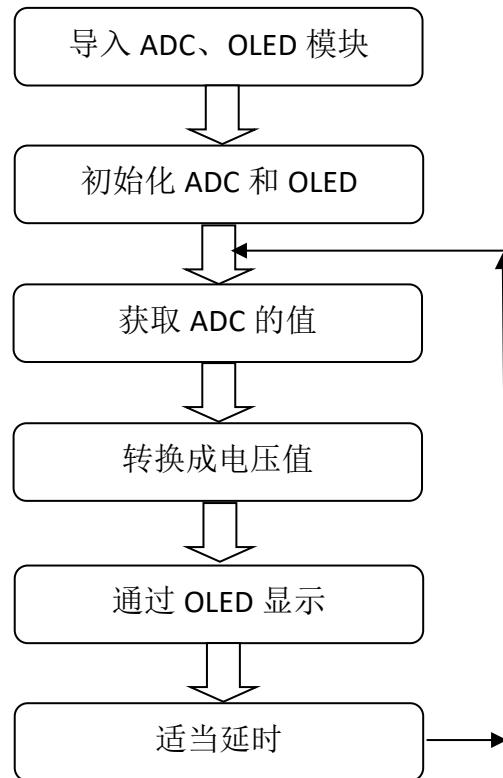


图 4-38 代码编写流程

实验参考代码：

```

...
实验名称: ADC-电压测量
版本: v1.0
日期: 2019.9
作者: 01Studio
说明: 通过对 ADC 数据采集, 转化成电压在显示屏上显示。ADC 精度 12 位, 电压 0-
3.3V。
...
#导入相关模块
import pyb,utime
from machine import Pin,I2C
from ssd1306x import SSD1306_I2C

```

```
#初始化相关模块
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"), freq=80000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
adc = pyb.ADC('P6') #Pin='P6'

while True:

    oled.fill(0) # 清屏显示黑色背景
    oled.text('01Studio', 0, 0) # 首行显示 01Studio
    oled.text('ADC', 0, 15)      # 次行显示实验名称

    #获取 ADC 数值
    oled.text(str(adc.read()),0,40)
    oled.text('(4095)',40,40)

    #计算电压值，获得的数据 0-4095 相当于 0-3.3V，('%.2f'%) 表示保留 2 位小数
    oled.text(str('%.2f'%(adc.read()/4095*3.3)),0,55)
    oled.text('V',40,55)

    oled.show()
    utime.sleep(1) #延时 1 秒
```

## ● 实验结果：

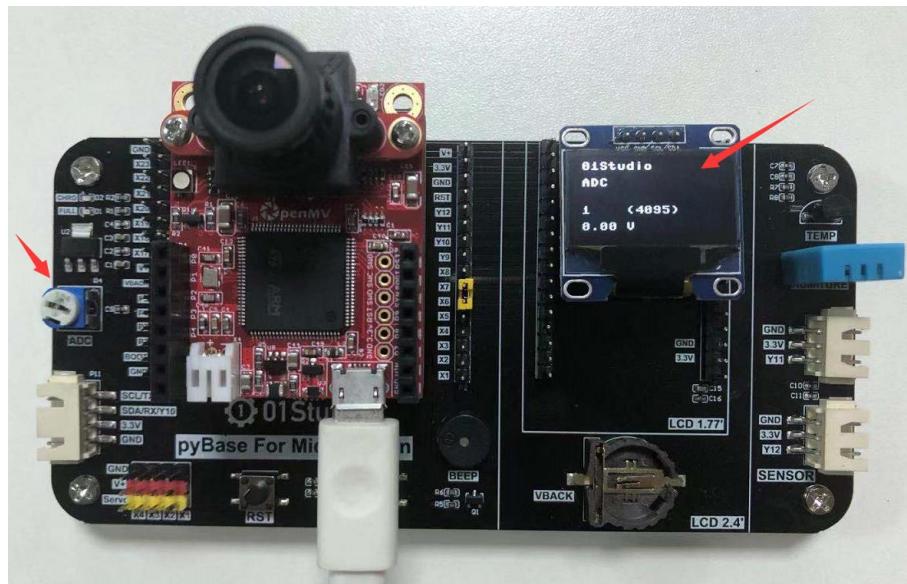


图 4-39 电位器顺时钟拧到尽头是 0V

调节电位器，发现电压测量值不断变化。

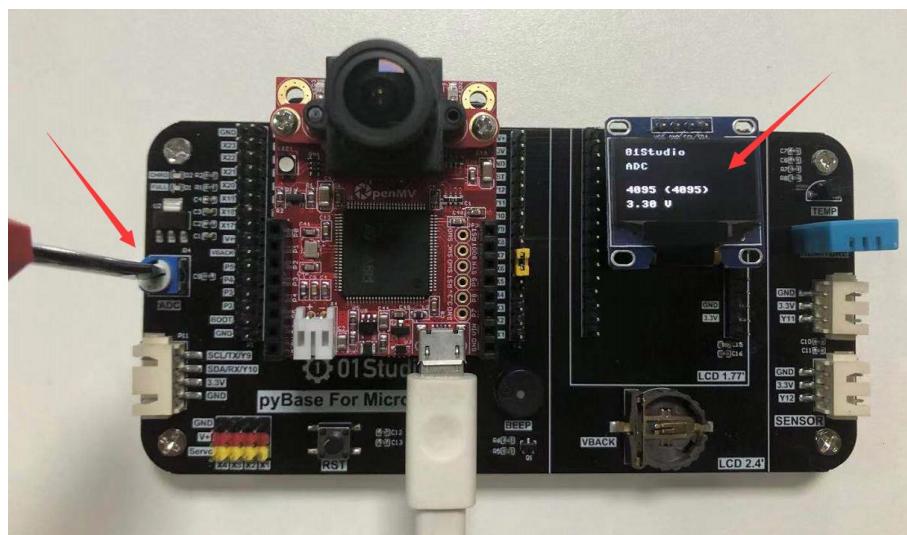


图 4-40 调节电位器来改变输入电压

### ● 总结：

这一节我们学习了 ADC 的应用。主要用于电压的检测、模拟信号输入（传感器）检测等应用。

## 4.9 DAC

### ● 前言：

DAC (Digital-to-Analog Converter) 数字模拟转换器。我们前面做的 ADC 实验，是将模拟信号转化为了数字信号。这一节做的实验刚好是反转，我们将特定的数字信号通过 DAC 输出。输出的形式多种，如特定电压、信号波形（正弦波、方波、三角波等），有条件的用户可以用示波器来观看输出。本章节利用开发板上的无源蜂鸣器，通过 DAC 输出不同的频率来改变其音色。

### ● 实验平台：

pyAI-OpenMV4 开发套件。

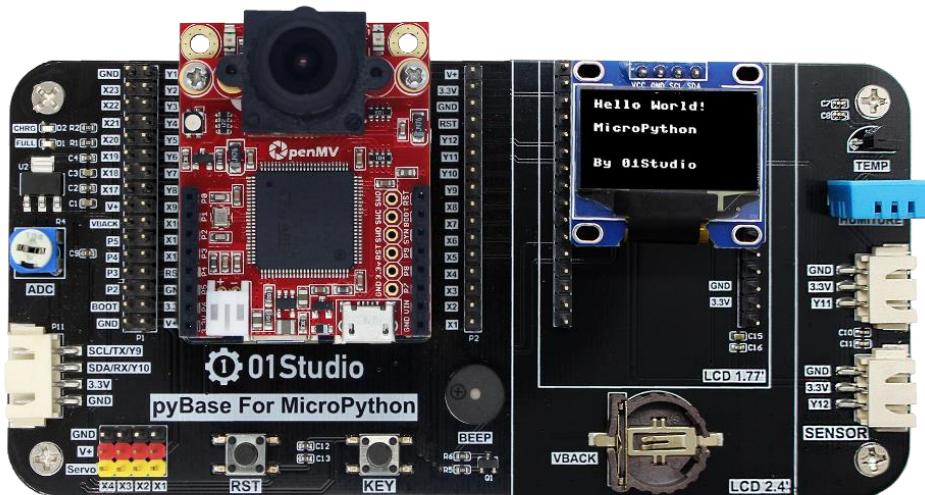


图 4-41 pyAI-OpenMV4 开发套件

### ● 实验目的：

通过 DAC 输出不同频率的方波来驱动蜂鸣器。

### ● 实验讲解：

蜂鸣器分有源蜂鸣器和无源蜂鸣器，有源蜂鸣器的使用方式非常简单，只需要接上电源，蜂鸣器就发声，断开电源就停止发声。而本实验用到的无源蜂鸣器，是需要给定指定的频率，才能发声的，而且可以通过改变频率来改变蜂鸣器的发声音色，以此来判定 OpenMV4 的 DAC 输出频率是在变化的。

从下图可以看到，pyBase 开发底板的 X5 引脚连接了无源蜂鸣器。如下图所示：

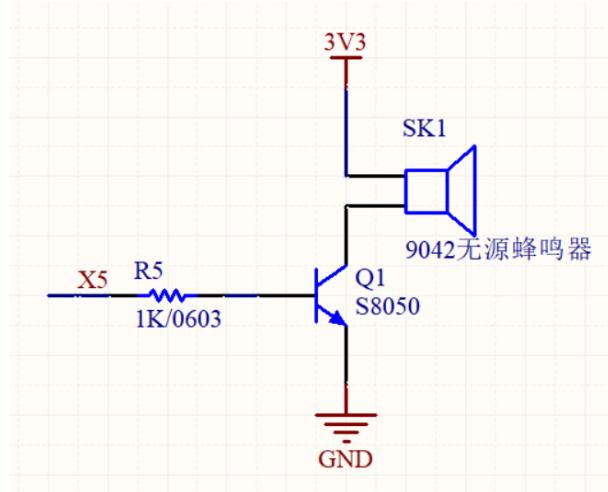


图 4-42 蜂鸣器原理图

而 pyAI-OpenMV4 的 DAC 引脚 P6 则是连接到 pyBase 开发底板的 X6 引脚，因此我们可以用跳线帽或者跳线来连接 pyBase 的 X5 和 X6 引脚。相当于将无源蜂鸣器接到 OpenMV4 的 P6 引脚。

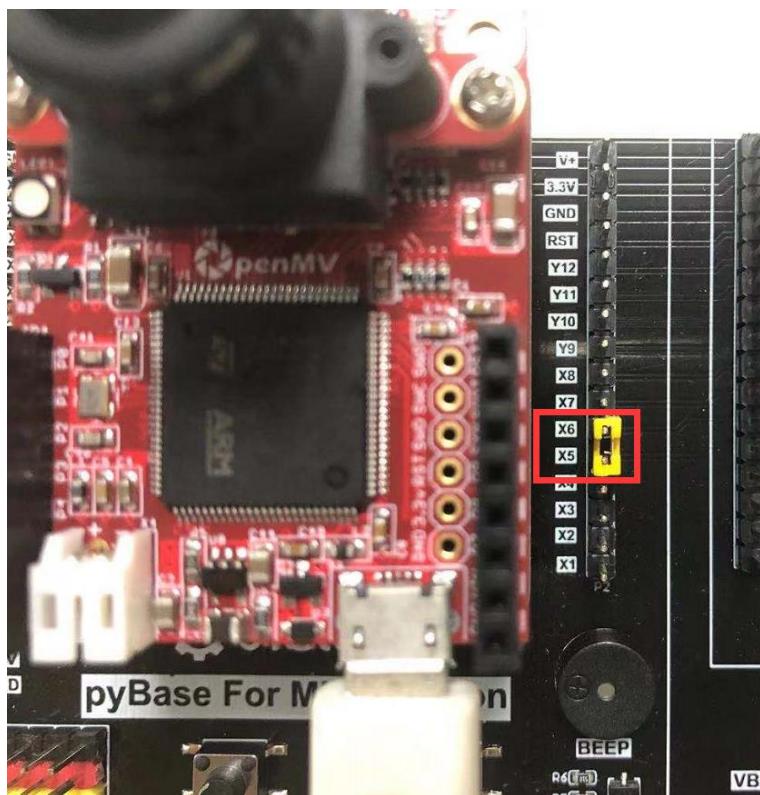


图 4-43 使用跳线帽将 X6 和 X5 短接

在了解了接线后，我们来看看 DAC 模块的函数使用方法。

构造函数
<code>pyb.DAC(port,bits=8)</code>
DAC 对象在 pyb 模块下。 【port】: 1, 对应 “P6” ; 【bits】: 8 或者 12
使用方法
<code>DAC.noise(freq)</code>
产生特定频率的噪声信号
<code>DAC.triangle(freq)</code>
产生特定频率三角波信号
<code>DAC.write(value)</code>
输出特定电压(0-VCC)。 【value】范围: 0~ $2^{\text{bits}} - 1$ (例: 8bits 那么范围就是 0-255)
<code>DAC.write_timed(data,freq,mode)</code>
输出特定信号。 【data】字节数组; 【freq】频率输出; 【mode】模式: <code>DAC.NORMAL</code> : 单次模式; <code>DAC.CIRCULAR</code> : 循环模式。

表 4-11 DAC 对象

无源蜂鸣器我们可以用特定频率的方波来驱动，方波的原理很简单，就是一定频率的高低电平转换，使用 `DAC.write(0)` 和 `DAC.write(255)` 交替输出即可，可以通过延时函数来控制输出变换的次数。实际过程中我们没必要用低效率的延时函数，因为看到 DAC 模块里面已经定义好了特定格式输出的函数 `DAC.write_timed(data,freq,mode)`，将 mode 设定成循环模式，就可以循环不断地输出指定频率的信号了。

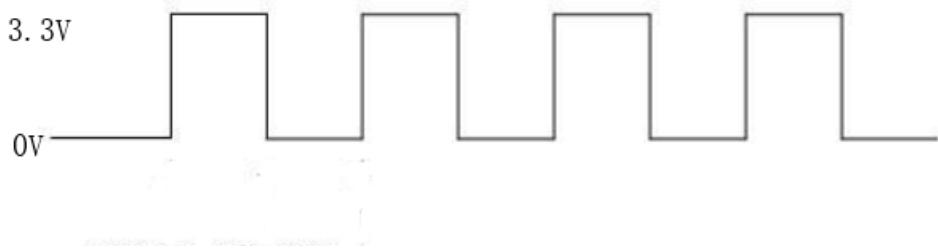


图 4-44 方波信号

结合我们前面的按键实验，我们可以通过 KEY 按键按下来切换输出频率，并在 OLED 上直观体验。具体流程如下：

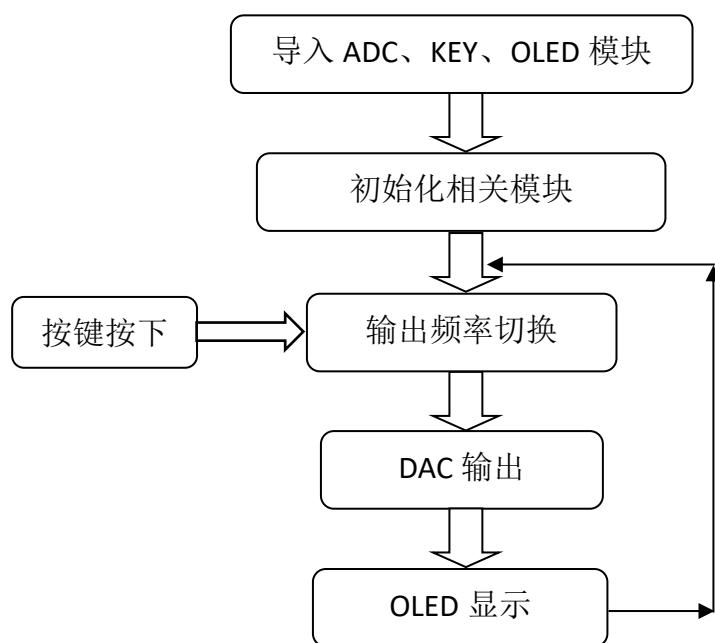


图 4-45 代码编程流程

实验参考代码如下：

```

...
实验名称: DAC-蜂鸣器
版本: v1.0
日期: 2019.9

```

作者: 01Studio

说明: 通过 KEY 按键让 DAC 输出不同频率的方波来驱动蜂鸣器。

'''

#导入相关模块

```
from pyb import DAC,ExtInt  
from machine import Pin,I2C  
from ssd1306x import SSD1306_I2C
```

#初始化相关模块

```
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"), freq=80000)  
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
```

```
dac = DAC(Pin("P6"))      #定义 DAC 对象名字为 dac, 输出引脚为 P6
```

#定义 4 组频率值: 1Hz、200Hz、1000Hz、5000Hz

```
freq=[1,200,1000,5000]
```

# 定义 8 位精度下方波的值。0、255 分别对应输出 0V、3.3V。需要定义成字节数组。

```
buf = bytearray(2)
```

```
buf[0]=0
```

```
buf[1]=255
```

key\_node = 0 #按键标志位

i = 0 #用于选择频率数组

```
#####
```

# 按键和其回调函数

```
#####
```

```
def key(ext):
```

```

global key_node
key_node = 1

#下降沿触发，打开上拉电阻
ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, key)

#####
# OLED 初始显示
#####
oled.fill(0) # 清屏显示黑色背景
oled.text('01Studio', 0, 0) # 首行显示 01Studio
oled.text('DAC-Beep', 0, 15) # 次行显示实验名称
oled.text('Pls Press USER', 0, 40) # 显示当前频率
oled.show()

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 4:
            i = 0
        key_node = 0 #清空按键标志位

        #DAC 输出指定频率
        dac.write_timed(buf, freq[i]*len(buf), mode=DAC.CIRCULAR)

    #显示当前频率
    oled.fill(0) # 清屏显示黑色背景
    oled.text('01Studio', 0, 0) # 首行显示 01Studio
    oled.text('DAC-Beep', 0, 15) # 次行显示实验名称

```

```
oled.text(str(freq[i]) + 'Hz', 0, 40) # 显示当前频率  
oled.show()
```

以上代码中 `DAC` 的函数使用非常简单，但为了让实验能更直观，加入按键和 `OLED` 模块后需要更多的代码来支持，但编程逻辑不变。以上代码有几个关键的地方需要补充说明一下：

- (1) 由于底层独特的操作方式，使得输入的数据 `buf` 必须是字节数组，所以需要加入 `bytearray()` 来定义。
- (2) `Key_node` 是全局变量，因此在 `key()` 函数里面用该变量必须添加 `global key_node` 代码，否则会在函数里面新建一个样的变量。
- (3) 变量 `i` 的作用是为了让 `DAC` 选择指定的频率数组 `freq` 输出

### ● 实验结果：

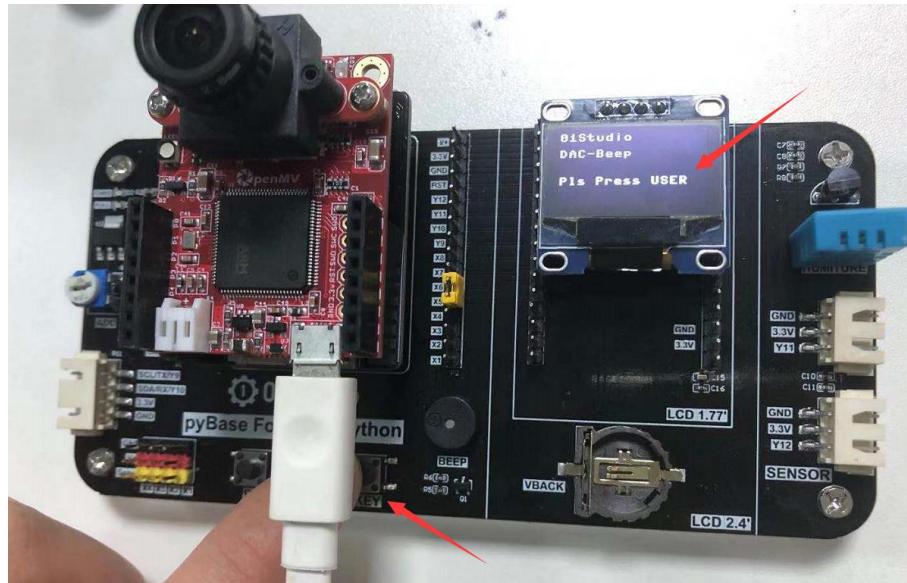


图 4-46 按下按键 KEY 开始

每次按下后发现 `OLED` 上的频率示意值发生变化，蜂鸣器的叫声也跟着变化。

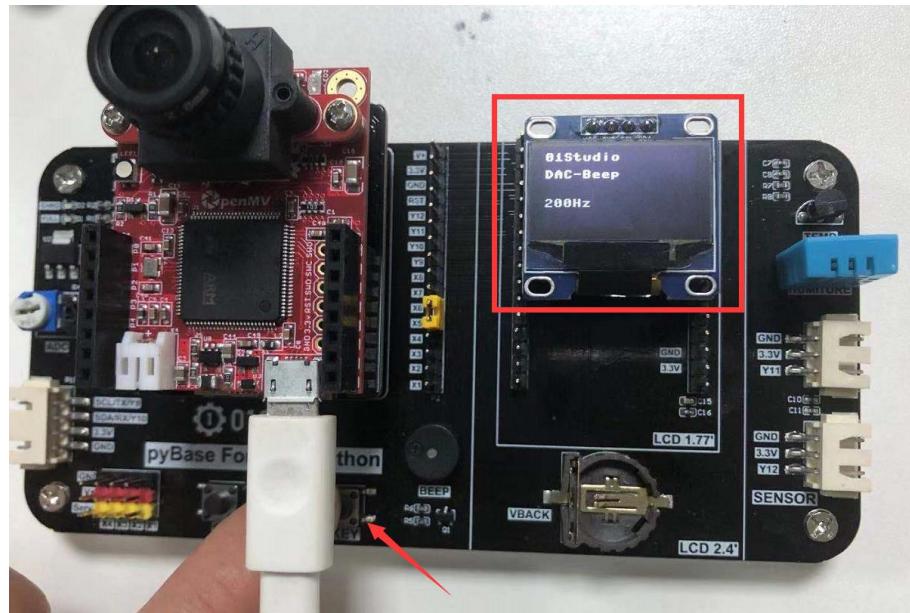


图 4-47 通过按键切换输出频率

有条件的朋友可以使用示波器连接 X5，观察信号波形的变化：

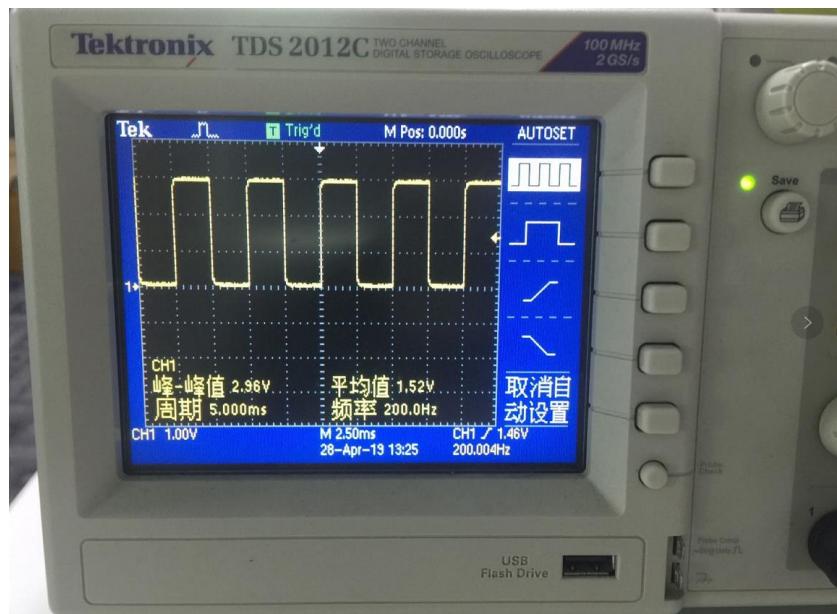


图 4-48 示波器显示

### ● 总结：

到了这一节，我们发现实验对象函数使用方法非常简单，而随着要实现功能的复杂化让编程的代码数量变多，逻辑也略显复杂。这是好事，让我们可以将更多精力放在应用上，做出更多好玩的创意。而不需要过多的关注复杂的底层代码开发。

## 第5章 机器视觉

### 5.1 摄像头应用

- **前言:**

从前面开发环境建立章节我们测试了 IDE 自带的 hello world 例程，看到了 IDE 成功显示摄像头实时采集的图像以及颜色直方图的显示。今天我们就通过示例代码来看看 OpenMV4 的摄像头是如何使用的。

- **实验平台:**

pyAI-OpenMV4。



图 5-1 pyAI-OpenMV4

- **实验目的:**

学习官方自带的 hello world 例程，理解 OpenMV4 摄像头基本编程和配置原理。

- **实验讲解:**

我们打开 IDE 中的 hello world 示例代码，其位置在文件---示例---01-Basics---helloworld.py。

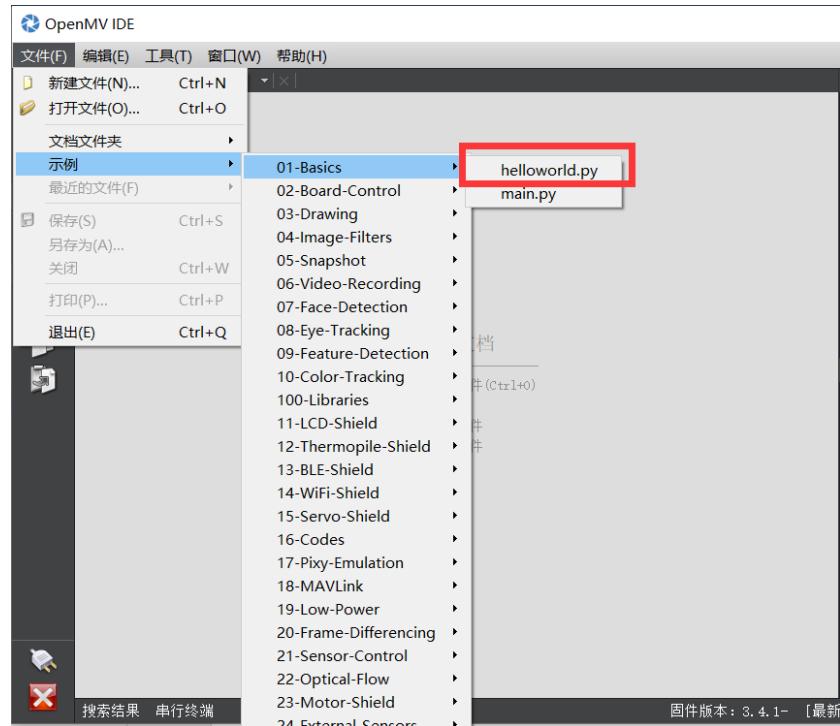


图 5-2 helloworld 例程位置

打开后发现编辑框出现了相关代码，我们可以先直接跑一下代码看看实验现象，连接 pyAI-OpenMV4，点击运行，可以发右图上方出现了摄像头实时采集的图像。

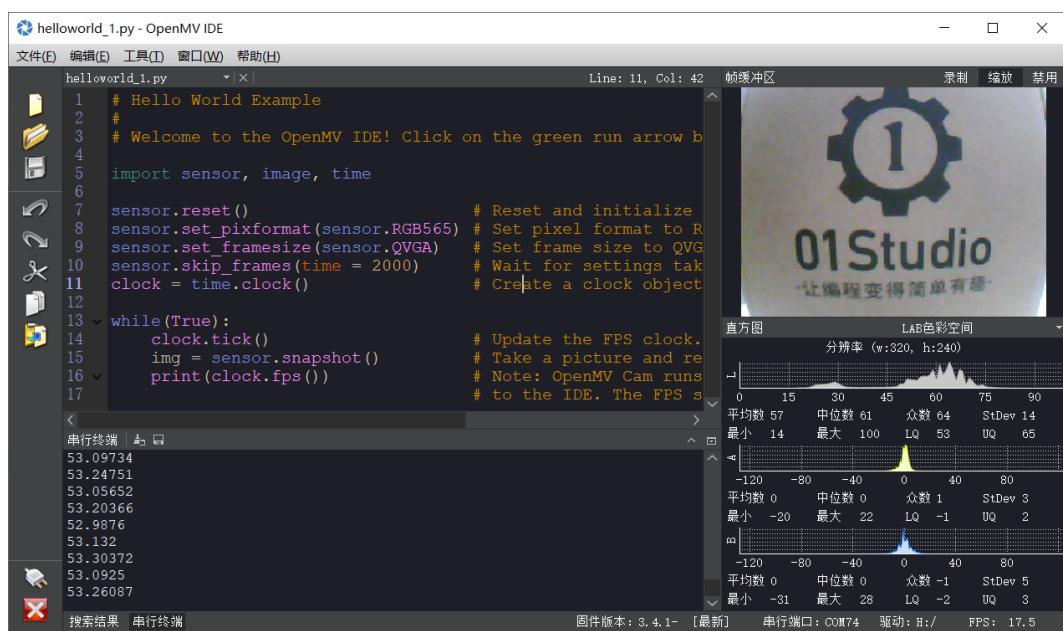


图 5-3

OpenMV 已经将所有的摄像头功能封装到 `sensor` 模块中，用户可以通过调用轻松使用。这也是使用 MicroPython 编程的魅力所在。

构造函数
<code>sensor</code>
摄像头对象，通过 <code>import</code> 直接调用
使用方法
<code>sensor.reset()</code>
初始化摄像头
<code>sensor.set_pixformat(<i>pixformat</i>)</code>
设置像素格式。 <code>pixformat</code> 有 3 个参数。 <code>sensor.GRAYSCAL</code> : 灰度图像，每像素 8 位（1 字节），处理速度快； <code>sensor.RGB565</code> : 每像素为 16 位（2 字节），5 位用于红色，6 位用于绿色，5 位用于蓝色，处理速度比灰度图像要慢。 <code>sensor.BAYER</code> : 占空间小，仅捕捉图像用，不可做图像处理。
<code>sensor.set_framesize(<i>framesize</i>)</code>
设置每帧大小（即图像尺寸）。常用的 <code>framesize</code> 参数有下面这些： <code>sensor.QQVGA: 160*120;</code> <code>sensor.QVGA: 320*240;</code> <code>sensor.VGA: 640*480;</code> <code>sensor.LCD: 128x160</code> (LCD 模块用) <code>sensor.QQVGA2: 128x160</code> (LCD 模块用) (更多参数请看表格末尾官方文档链接。)
<code>sensor.skip_frames([<i>n</i>, <i>time</i>])</code>
摄像头配置后跳过 <i>n</i> 帧或者等待时间 <i>time</i> 让其变稳定。 <i>n</i> : 跳过帧数； <i>time</i> : 等待时间，单位 ms。 (如果 <i>n</i> 和 <i>time</i> 均没指定，则默认跳过 300 毫秒的帧。)
<code>sensor.snapshot()</code>
使用相机拍摄一张照片，并返回 <code>image</code> 对象。
*其它更多用法请阅读 <code>openmv</code> 官方文档： 文档链接： <a href="http://docs.openmv.io/library/omv.sensor.html">http://docs.openmv.io/library/omv.sensor.html</a>

表 5-1 sensor 摄像头对象

我们再来看看本例程用于计算 FPS（每秒帧数）的 CLOCK 对象模块。

构造函数
<code>clock=time.clock()</code>
创建一个时钟。
使用方法
<code>clock.tick()</code>
开始追踪运行时间。
<code>clock.fps ()</code>
停止追踪运行时间，并返回当前 FPS（每秒帧数）。
在调用该函数前始终首先调用 <code>tick</code> 。

图 5-4 Clock 对象

我们来看看 helloworld 代码的编写流程图：

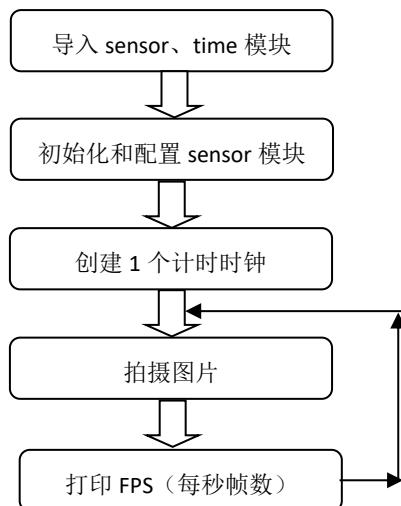


图 5-5 helloworld 代码编写流程

这个实验运行的就是编辑框里面的 helloworld 代码：

```
# Hello World 例程
```

```
#
```

```

#翻译和注释: 01Studio

#导入相关模块

import sensor, image, time, lcd

lcd.init(freq=15000000)    # LCD 初始化
sensor.reset()              # 复位和初始化摄像头设备, 执行 sensor.run(0)停止

sensor.set_pixformat(sensor.RGB565) # 设置像素格式为彩色 RGB565 (或灰色
GRAYSCALE)

sensor.set_framesize(sensor.QVGA)   # 设置帧大小为 QVGA (320x240) 默认 LCD
大小

sensor.skip_frames(time = 2000)      # 等待设置生效
clock = time.clock()               # 创建一个时钟来追踪 FPS (每秒拍摄帧数) .

while(True):

    clock.tick()                  # 更新 FPS 时钟.
    img = sensor.snapshot()        # 拍摄图像并返回 image.
    lcd.display(img)              # 在 LCD 上显示
    print(clock.fps())            # 注意: 当 MaixPy 连接到 IDE 时候, 运行速
度减半,
                                         # 因此当脱机运行时, FPS 会提升。

```

### ● 实验结果:

点击运行, 可以看到在右边显示摄像头实时拍摄情况, 下方则显示 RGC 颜色直方图。

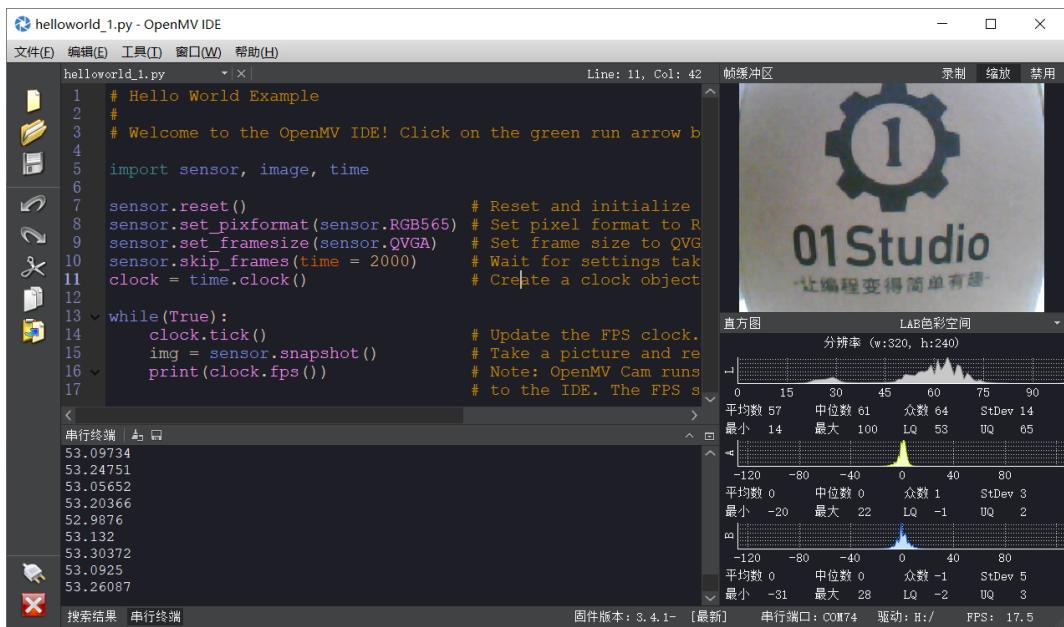


图 5-6 实验现象

点击左下角串口终端，可以看到软件弹出串口打印串口，实时显示当前的 FPS(每秒帧数)值为 50 多帧。

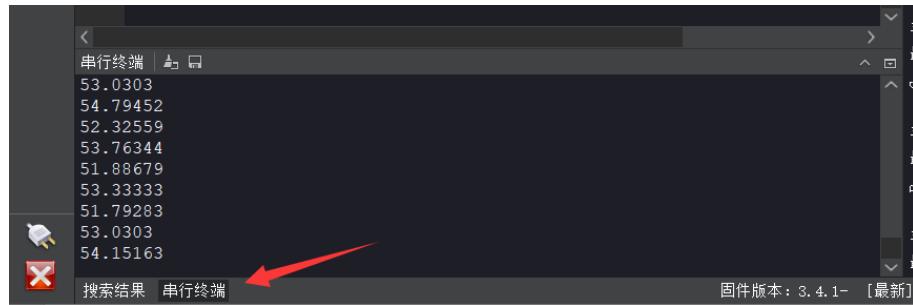


图 5-7 串口终端显示 FPS

### ● 总结:

通过本实验，我们了解了摄像头 sensor 模块以及时间 time 模块的原理和应用，可以看到 OpenMV 将摄像头功能封装成 sensor 模块，用户不必关注底层代码编可以轻松使用。

## 5.2 画图

- **前言:**

通过摄像头采集到照片后，我们会进行一些处理，而这时候往往需要一些图形来指示，比如在图片某个位置标记箭头、人脸识别后用矩形框提示等。本节就是学习在图形上画图的使用功能。

- **实验平台:**

pyAI-OpenMV4。



图 5-8 pyAI-OpenMV4

- **实验目的:**

在拍摄的图片上画各种图形。

- **实验讲解:**

在 OpenMV IDE 的示例程序里面有各类画图的示例程序，大家可以自行阅读。

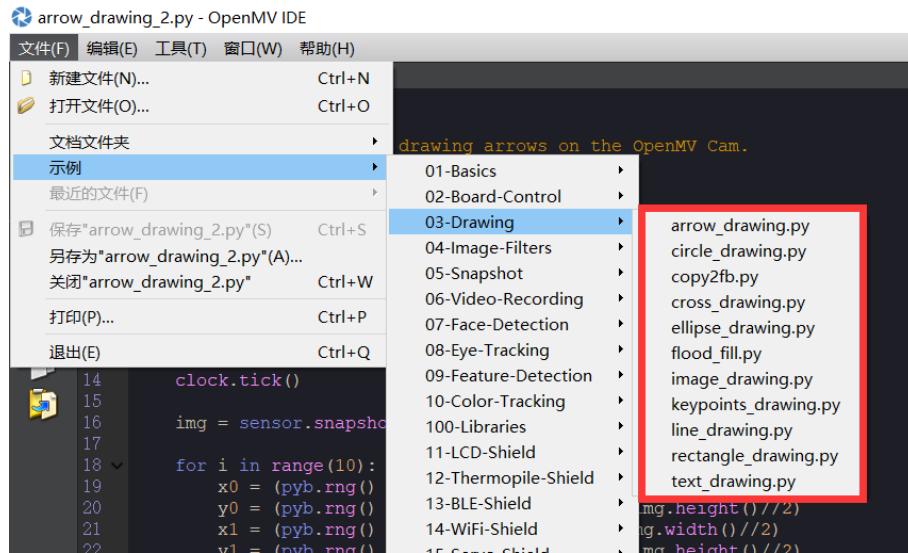


图 5-9 画图示例程序

上一节我们学习了摄像头 `sensor` 模块应用，通过摄像头实时采集到的是图片 `image`，没错，本节实验就是建立在非常重要的 `image` 模块上面。`openmv` 已经将图片处理（包含画图）封装成各类模块，我们只需要熟悉其构造函数和使用方法即可，具体如下：

构造函数
<code>img=sensor.snapshot()</code> 或 <code>img=image.Image(path[, copy_to_fb=False])</code>
创建图像，通过拍摄或者读取文件路径获取。  <code>copy_to_fb=True</code> : 可以加载大图片； <code>copy_to_fb=False</code> : 不可以加载大图片。  示例： <code>img = image.Image("/01Studio.bmp", copy_to_fb=True)</code> , 表示加载根目录下的 <code>01Studio.bmp</code> 图片。
使用方法
<code>image.draw_line(x0, y0, x1, y1[, color[, thickness=1]])</code>  画线段。 <code>(x0,y0)</code> :起始坐标； <code>(x1,y1)</code> :终点坐标； <code>color</code> :颜色，如 <code>(255,0,0)</code> 表示红色； <code>thickness</code> : 粗细。
<code>image.draw_rectangle(x, y, w, h[, color[, thickness=1[, fill=False]]])</code>  画矩形。 <code>(x,y)</code> :起始坐标； <code>w</code> :宽度； <code>h</code> :长度； <code>color</code> : 颜色； <code>thickness</code> : 边框粗

细; <b>fill</b> :是否填充。
<code>image.draw_circle(x, y, radius[, color[, thickness=1[, fill=False]]])</code>
画圆。 <b>(x,y)</b> :圆心; <b>radius</b> :半径; <b>color</b> : 颜色; <b>thickness</b> :线条粗细; <b>fill</b> : 是否填充。
<code>image.draw_arrow(x0, y0, x1, y1[, color[, size,[thickness=1]]])</code>
画箭头。 <b>(x0,y0)</b> :起始坐标; <b>(x1,y1)</b> :终点坐标; <b>color</b> :颜色; <b>size</b> :箭头位置大小。 <b>thickness</b> : 线粗细。
<code>image.draw_cross(x, y[, color[, size=5[, thickness=1]]])</code>
画十字交叉。 <b>(x,y)</b> : 交叉坐标; <b>color</b> :颜色; <b>size</b> :尺寸; <b>thickness</b> : 线粗细。
<code>image.draw_string(x, y, text[, color[, scale=1[,mono_space=True...]]])</code>
写字符。 <b>(x,y)</b> : 起始坐标; <b>text</b> :字符内容; <b>color</b> : 颜色; <b>scale</b> : 字体大小; <b>mono_space</b> :强制间距。
*其它更多用法请阅读 OpenMV 官方文档: 文档链接: <a href="http://docs.openmv.io/library/omv.image.html">http://docs.openmv.io/library/omv.image.html</a>

表 5-2 image 对象画图功能

熟悉了 image 对象的画图功能后, 我们尝试在摄像头采集到的画面依次画出线段、矩形、圆形、箭头、十字交叉和字符。具体编程思路如下:

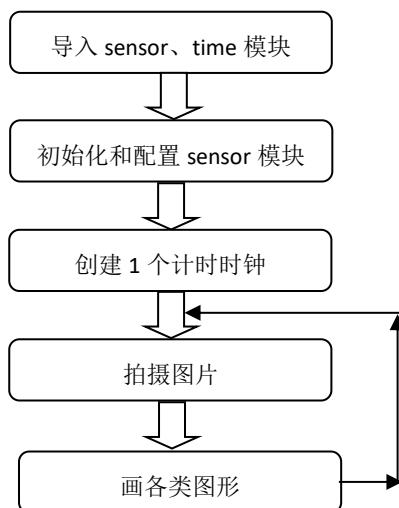


图 5-10 代码编写流程

参考代码：

```
# 在这里写上你的代码 :-)
...
实验名称：画各种图形和写字符
版本：v1.0
日期：2019.10
作者：01Studio
...
import sensor, image, time, pyb

sensor.reset()
sensor.set_pixformat(sensor.RGB565) # or GRayscale...
sensor.set_framesize(sensor.QVGA) # or QQVGA...
sensor.skip_frames(time = 2000)
clock = time.clock()

while(True):

    clock.tick()

    img = sensor.snapshot()

    # 画线段：从 x0, y0 到 x1, y1 坐标的线段，颜色红色，线宽度 2。
    img.draw_line(20, 20, 100, 20, color = (255, 0, 0), thickness = 2)

    # 画矩形：绿色不填充。
    img.draw_rectangle(150, 20, 100, 30, color = (0, 255, 0),
                      thickness = 2, fill = False)

    # 画圆：蓝色不填充。
    img.draw_circle(200, 50, 50, color = (0, 0, 255), thickness = 2, fill = False)
```

```
img.draw_circle(60, 120, 30, color = (0, 0, 255), thickness = 2,  
                fill = False)  
  
#画箭头：白色。  
img.draw_arrow(150, 120, 250, 120, color = (255, 255, 255), size =  
               20, thickness = 2)  
  
#画十字交叉。  
img.draw_cross(60, 200, color = (255, 255, 255), size = 20,  
               thickness = 2)  
  
#写字符。  
img.draw_string(150, 200, "Hello 01Studio!", color = (255, 255,  
           255), scale = 2, mono_space = False)  
  
print(clock.fps())
```

### ● 实验结果：

在 OpenMV IDE 中打开例程文件，或者直接复制到新建的文件中，点击运行。可以看到在图像缓冲区中画上了各种图形。

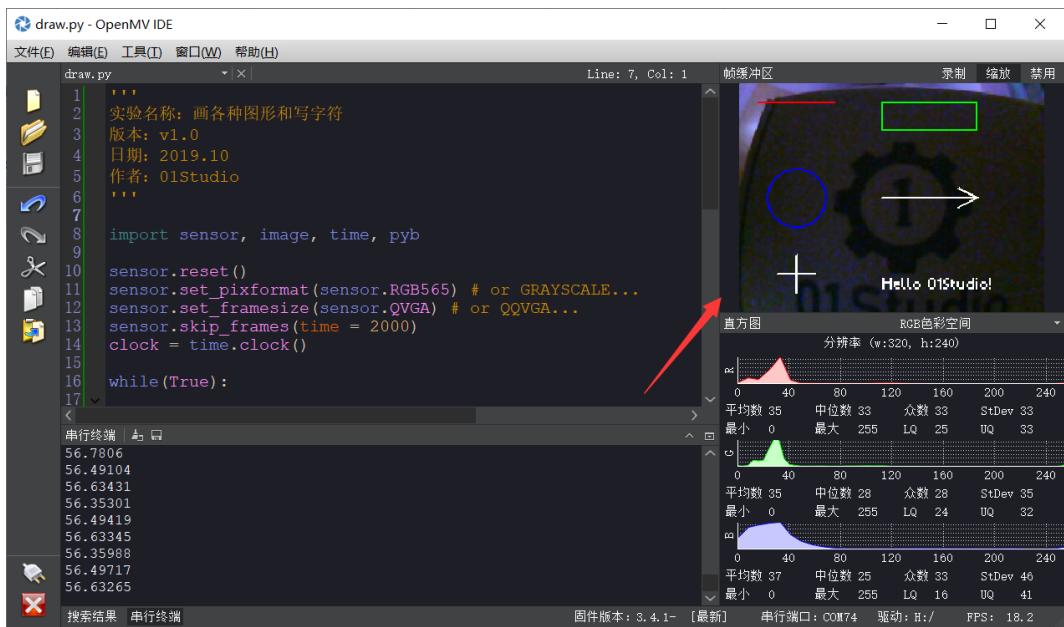


图 5-11 画图形

### ● 总结:

画图形是很基础的功能,但在以后的实验中特别是指示识别内容时候会经常用到。

## 5.3 特征检测

从本节起，可以说是正式进入视觉检测的学习。

特征检测，要做的就是对图像进行特定的特征检测，如本节包含的边缘检测、各种形状识别、特征点识别、线性回归等都是非常有代表性的实验。特征检测的全部官方例程位于 **Feature-Detection** 例程下，本节挑选重点实验进行讲解。

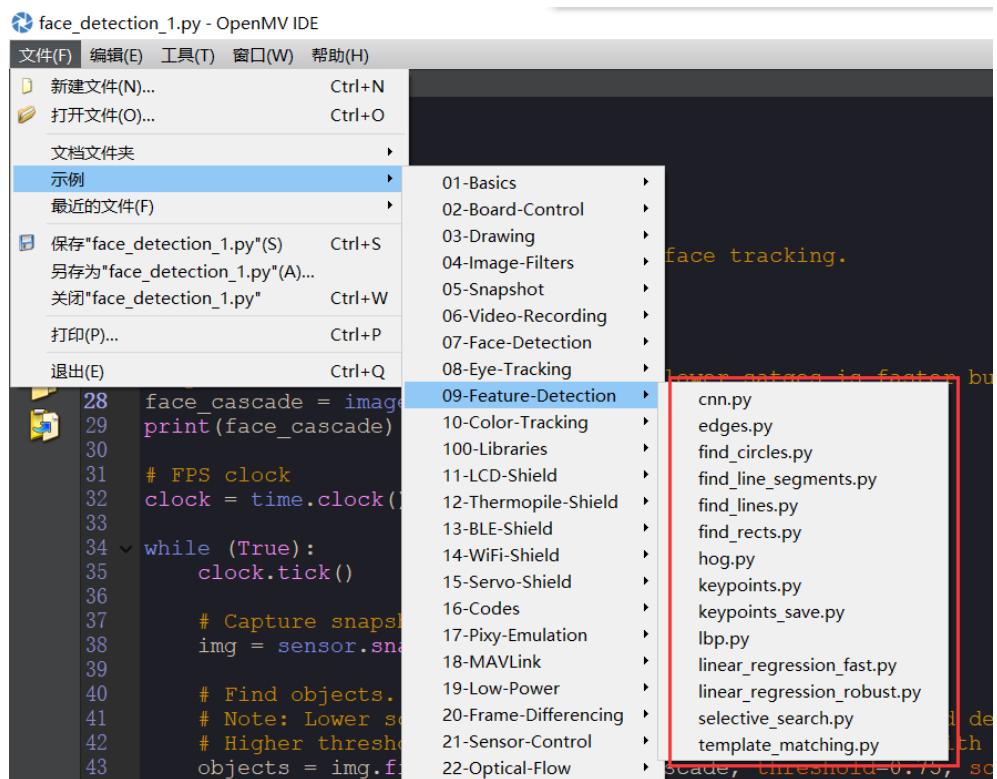


图 5-12 特征检测官方例程位置

### 5.3.1 边缘检测

- **前言:**

生活中每个物体都有一个边缘。简单来说就是轮廓，本节学习的是使用 MicroPython 结合 OpenMV4 自带的库来做图像轮廓检测。

- **实验平台:**

pyAI-OpenMV4。



图 5-13 pyAI-OpenMV4

- **实验目的:**

识别摄像头采集的图像轮廓并显示。

- **实验讲解:**

OpenMV4 集成了边缘识别 `find_edges` 函数，位于 `image` 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下边缘处理函数相关说明，具体如下：

## 构造函数

```
image.find_edges(edge_type[, threshold])
```

边缘检测，将图像变为黑白，边缘保留白色像素。

【edge\_type】处理方式：

`image.EDGE_SIMPLE` - 简单的阈值高通滤波算法

`image.EDGE_CANNY` - Canny 边缘检测算法

【threshold】包含高、低阈值的二元组，默认是（100,200），仅支持灰度图像。

## 使用方法

直接调用该函数。

表 5-3 边缘处理函数

由此可见边缘处理的方法非常简单，我们结合前面摄像头的应用，整理一下编程思路如下：

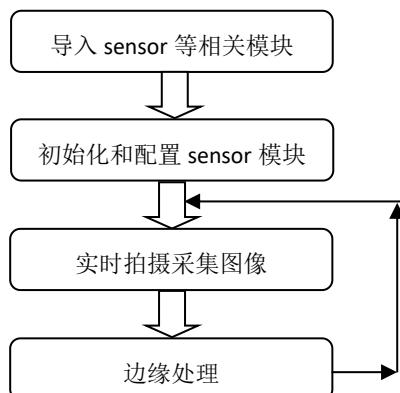


图 5-14 代码编写流程

官方参考代码路径【示例→Feature-Detection→edges.py】，具体如下：

```
# 使用 Canny 算法做边缘检测:  
#  
# 这个例子演示了 Canny 边缘检测器.  
#
```

```
#翻译: 01Studio

import sensor, image, time

#初始化摄像头
sensor.reset() # 初始化摄像头模块.
sensor.set_pixformat(sensor.GRAYSCALE) # 或者使用 sensor.RGB565 彩色
sensor.set_framesize(sensor.QQVGA) # 或者使用 sensor.QVGA (or others)
sensor.skip_frames(time = 2000) #延时让摄像头文稳定.
sensor.set_gainceiling(8) #设置增益, 这是官方推荐的参数

clock = time.clock() # Tracks FPS.

while(True):

    clock.tick() # 用于计算 FPS (每秒帧数) .

    img = sensor.snapshot() # 拍摄并返回图像.

    #使用 Canny 边缘检测器
    img.find_edges(image.EDGE_CANNY, threshold=(50, 80))

    # 也可以使用简单快速边缘检测, 效果一般, 配置如下
    #img.find_edges(image.EDGE_SIMPLE, threshold=(100, 255))

    print(clock.fps()) #显示 FPS (每秒帧数)
```

- **实验结果：**

使用 IDE 运行上面的代码，可以看到实验现象如下：



图 5-15 边缘检测实验：原图（左）和实验图片（右）

- **总结：**

本节做了特征检测的边缘检测实验，可以看到使用 MicroPython 开发是如此的简单美妙。本节仅仅是一个开始，接下来我们将学习更多的机器视觉应用。

### 5.3.2 圆形识别

- **前言:**

本节学习的是对图像中的圆形进行识别。

- **实验平台:**

pyAI-OpenMV4。



图 5-16 pyAI-OpenMV4

- **实验目的:**

识别图像中的圆形并用画圆功能指示出来。

- **实验讲解:**

OpenMV4 集成了圆形识别 `find_circles` 函数，位于 `image` 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下圆形识别函数相关说明，具体如下：

构造函数
<code>image.find_circles([roi[, x_stride=2[, y_stride=1[, threshold=2000[, x_margin=10[, y_margin=10[, r_margin=10[, r_min=2[, r_max[, r_step=2]]]]]]]]])</code>

```
image.find_circles([roi[, x_stride=2[, y_stride=1[, threshold=2000[, x_margin=10[, y_margin=10[, r_margin=10[, r_min=2[, r_max[, r_step=2]]]]]]]]])
```

找圆函数。返回一个 `image.circle` 圆形对象，该圆形对象有 4 个值：`x, y`（圆心），`r`（半径）和 `magnitude`（量级）；量级越大说明识别到的圆可信度越高。

【roi】识别区域 (`x,y,w,h`)，未指定则默认整张图片；

【threshold】阈值。返回大于或等于 `threshold` 的圆，调整识别可信度；

【x\_stride】【y\_stride】霍夫变换时跳过 `x, y` 像素的量；

【x\_margin】【y\_margin】【r\_margin】控制所检测圆的合并；

【r\_min】【r\_max】控制识别圆形的半径范围；

【r\_step】控制识别步骤

### 使用方法

直接调用该函数。（大部分参数使用默认即可，不支持压缩图像和 `bayer` 图像）

表 5-4 圆形识别函数

圆形检测的编程思路如下：

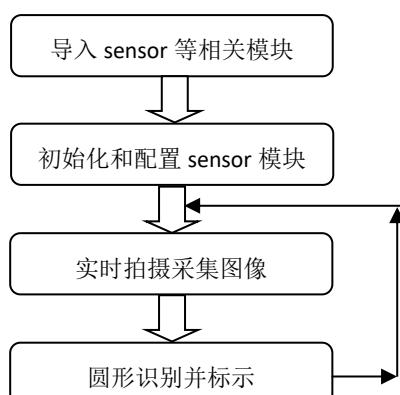


图 5-17 代码编写流程

官方参考代码路径【示例 → Feature-Detection → `find_circles.py`】，具体如下：

```
# 找圆形例程  
#  
# 这个例程使用 Hough 算法（霍夫变换）  
# 参考链接: https://en.wikipedia.org/wiki/Circle\_Hough\_Transform  
#  
# 需要注意的是该例程只适用于识别在图像内部完整的圆形，超出图像或者识别区域
```

```
# (roi) 的无效。  
#  
#翻译: 01Studio  
  
import sensor, image, time  
  
#摄像头模块初始化  
sensor.reset()  
sensor.set_pixformat(sensor.RGB565) # 使用 grayscale 灰度图像处理速度会更快  
sensor.set_framesize(sensor.QQVGA)  
sensor.skip_frames(time = 2000)  
clock = time.clock()  
  
while(True):  
  
    clock.tick()  
  
    #lens_corr 为了去除畸变, 1.8 是默认参数, 可以根据自己实际情况调整  
    img = sensor.snapshot().lens_corr(1.8)  
  
    # 圆形类有 4 个参数值: 圆心(x, y), r (半径)和 magnitude (量级);  
    # 量级越大说明识别到的圆可信度越高。  
  
    # `threshold` 参数控制找到圆的数量, 数值的提升会降低识别圆形的总数。  
    # `x_margin`, `y_margin`, and `r_margin` 控制检测到接近圆的合并调节。  
    # r_min, r_max, and r_step 用于指定测试圆的半径范围。  
  
    for c in img.find_circles(threshold = 2000, x_margin = 10, y_margin  
        = 10, r_margin = 10, r_min = 2, r_max = 100, r_step = 2):
```

```
#画红色圆做指示  
img.draw_circle(c.x(), c.y(), c.r(), color = (255, 0, 0))  
print(c) #打印圆形的信息  
  
print("FPS %f" % clock.fps()) #打印 FPS (每秒采集帧数)
```

### ● 实验结果：

使用 IDE 运行以上代码，可以看到实验现象如下：



图 5-18 圆形识别实验：原图（左）和实验图片（右）

在串行终端可以看到代码 `print(c)` 通过串口打印出来圆形的信息。

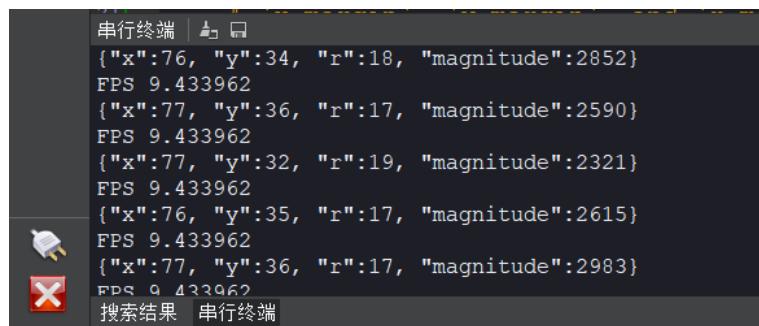


图 5-19 串口终端

### ● 总结：

本节学习了圆形识别，可以使用该例程来识别圆形图案或者球类物体。

### 5.3.3 线段识别

- **前言:**

本节学习的是对图像中的线段进行识别。

- **实验平台:**

pyAI-OpenMV4。



图 5-20 pyAI-OpenMV4

- **实验目的:**

识别图像中的线段并用画线段功能指示出来。

- **实验讲解:**

OpenMV4 集成了线段识别 `find_line_segments` 函数，位于 `image` 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下线段识别函数相关说明，具体如下：

构造函数
<code>image.find_line_segments([roi[, merge_distance=0[, max_theta_difference=15]]])</code>
线段识别函数。返回一个 <code>image.line</code> 线段对象列表。

【roi】识别区域 (x,y,w,h)，未指定则默认整张图片；

【merge\_distance】两条线段间可以相互分开而不被合并的最大像素；

【max\_theta\_difference】将少于这个角度值的线段合并。

### 使用方法

直接调用该函数。(大部分参数使用默认即可，不支持压缩图像和 bayer 图像)

表 5-5 线段识别函数

线段检测的编程思路如下：

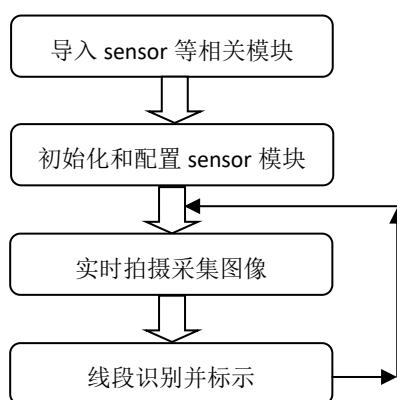


图 5-21 代码编写流程

官方参考代码路径【示例→Feature-Detection→find\_line\_segments.py】，具体如下：

```
# 线段识别例程
#
# 该例程演示了如果在图像中找出线段，每个被找出的线段都会返回一个对象，包括线端
# 旋转的线对象。

# 使用 find_line_segments() 可以找特定的线段（速度比较慢）。
# 使用 find_line_segments() 找任意线段（速度比较快）。
#
# 翻译：01Studio
```

```
enable_lens_corr = False # 打开以获得更直的线段

import sensor, image, time

#摄像头初始化
sensor.reset()

sensor.set_pixformat(sensor.RGB565) # grayscale 灰度图像处理速度更快
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)

clock = time.clock()

# 所有线段对象都可以通过 `x1()`，`y1()`，`x2()`，and `y2()` 方法获取其端
# 点，当然也可以使用`line()` 获取上面 4 个值以用于 `draw_line()`画线.

while(True):

    clock.tick()

    img = sensor.snapshot()

    if enable_lens_corr: img.lens_corr(1.8) # for 2.8mm lens...

    # `merge_distance` 控制相近的线段是否合并。 数值 0 (默认值)表示不合并。数值
    #为 1 时候表示相近 1 像素的线段被合并。因此你可以通过改变这个参数来控制检测到线
    #段的数量。

    # `max_theta_diff` 控制相差一定角度的线段合并， 默认是 15 度， 表示 15 度内的线
    # 段都会合并

    for l in img.find_line_segments(merge_distance = 0,
                                    max_theta_diff = 5):
```

```
img.drawLine(l.line(), color = (255, 0, 0))

print(l)

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

- **实验结果：**

使用 IDE 运行以上代码，可以看到实验现象如下：

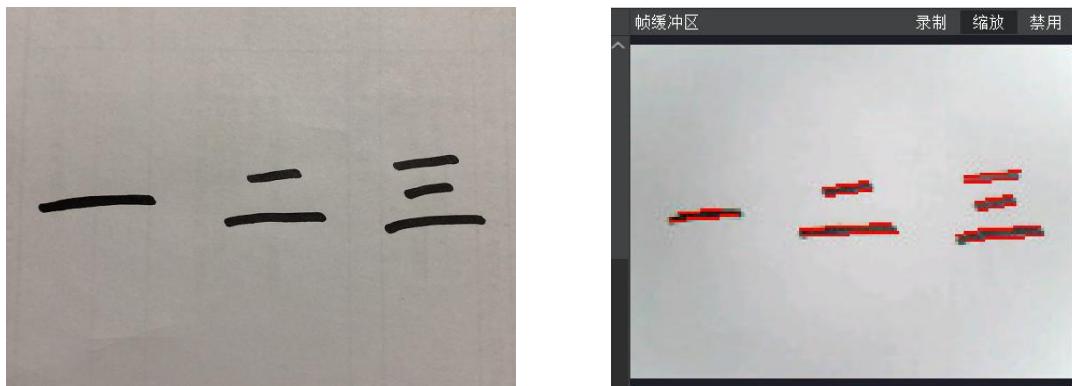


图 5-22 线段识别实验：原图（左）和实验图片（右）

- **总结：**

本节学习了线段识别，可以使用该例程来识别线段图案或者相关物体。

### 5.3.4 直线识别

- **前言:**

本节学习的是对图像中的直线进行识别。

- **实验平台:**

pyAI-OpenMV4。

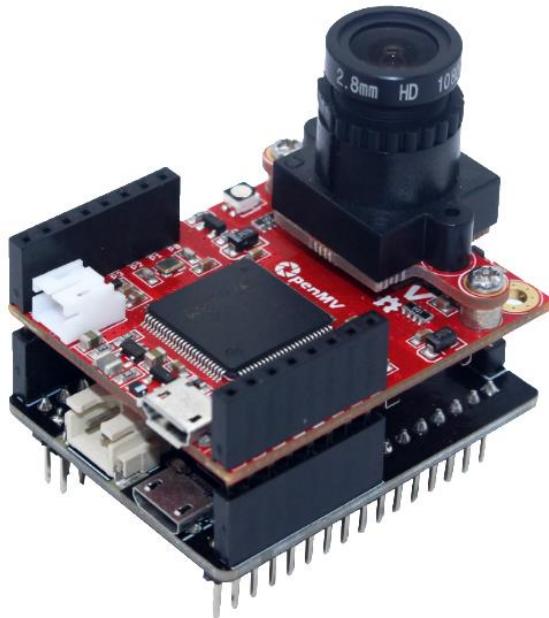


图 5-23 pyAI-OpenMV4

- **实验目的:**

识别图像中的直线并用画直线功能指示出来。

- **实验讲解:**

OpenMV4 集成了直线识别 `find_lines` 函数，位于 `image` 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下识别函数相关说明，具体如下：

构造函数
<code>image.find_lines([roi[, x_stride=2[, y_stride=1[, threshold=1000[, theta_margin=25[, rho_margin=25]]]]]])</code>
直线识别函数。返回一个 <code>image.line</code> 直线对象；

**【roi】** 识别区域 ( $x, y, w, h$ )，未指定则默认整张图片；

**【threshold】** 阈值。返回大于或等于 threshold 的直线，调整识别可信度；

**【x\_stride】【y\_stride】** 霍夫变换时跳过  $x, y$  像素的量；

**【theta\_margin】【rho\_margin】** 控制所检测直线的合并规则；

### 使用方法

直接调用该函数。(大部分参数使用默认即可，不支持压缩图像和 bayer 图像)

表 5-6 直线识别函数

圆形检测的编程思路如下：

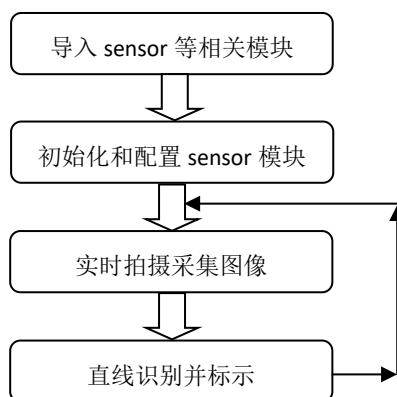


图 5-24 代码编写流程

官方参考代码路径【示例→Feature-Detection→find\_lines.py】，具体如下：

```
# 直线识别例程

#
# 该例程演示了如何在图像中找出直线，每个被找出的直线都会返回一个对象，包括直线
# 的线路定位。
#
# 直线识别使用了霍夫转换算法：
#
# 参考链接: http://en.wikipedia.org/wiki/Hough\_transform
# 你可以从上面链接更深入了解 `theta` 和 `rho` 参数的概念。
#
# 使用 find_lines() 找直线（速度比较慢）。
#
# 使用 find_line_segments() 找线段（速度比较快）。
```

```
#  
#翻译: 01Studio  
  
enable_lens_corr = False # 打开以获得更直的线段  
  
import sensor, image, time  
  
sensor.reset()  
sensor.set_pixformat(sensor.RGB565) # grayscale is faster  
sensor.set_framesize(sensor.QQVGA)  
sensor.skip_frames(time = 2000)  
clock = time.clock()  
  
# 所有直线都可以通过 `theta()` 函数来获取跟水平线的夹角。  
# 你也可以通过滤波算法来平滑角度。  
  
min_degree = 0  
max_degree = 179  
  
# 所有直线对象都可以通过 `x1()`，`y1()`，`x2()`，and `y2()` 方法获取其端  
#点，当然也可以使用`line()` 获取上面 4 个值以用于 `draw_line()`画线。  
  
while(True):  
  
    clock.tick()  
    img = sensor.snapshot()  
    if enable_lens_corr: img.lens_corr(1.8) # for 2.8mm lens...  
  
    # `threshold` 参数控制找到直线的数量，数值的提升会降低识别直线的总数。
```

```

# `theta_margin` and `rho_margin` 控制直线是否合并。若少于预设值则合并

for l in img.find_lines(threshold = 1000, theta_margin = 25,
                        rho_margin = 25):

    # 直线角度在 0 至 180 度范围内

    if (min_degree <= l.theta()) and (l.theta() <= max_degree):
        img.draw_line(l.line(), color = (255, 0, 0))
        print(l)

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

# 关于 rho 出现负数:

# [theta+0:-rho] 元组与 [theta+180:+rho] 相同.

```

### ● 实验结果:

使用 IDE 运行以上代码，可以看到实验现象如下：

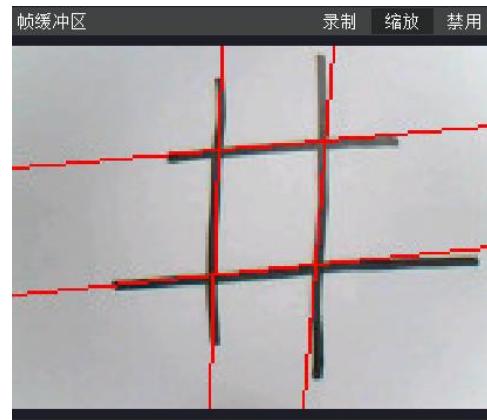
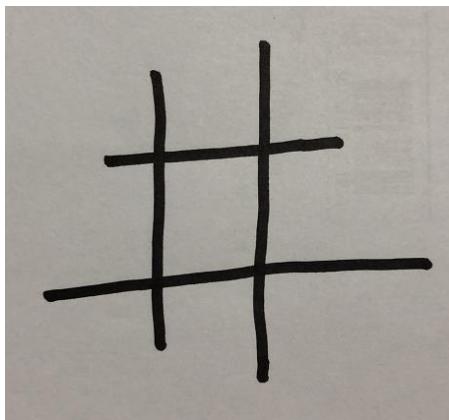


图 5-25 直线识别实验：原图（左）和实验图片（右）

### ● 总结:

本节学习了直线识别，可以使用该例程来识别直线图案或者相关物体。

### 5.3.5 矩形识别

- **前言:**

本节学习的是对图像中的矩形进行识别。

- **实验平台:**

pyAI-OpenMV4。



图 5-26 pyAI-OpenMV4

- **实验目的:**

识别图像中的矩形并用画图功能指示出来。

- **实验讲解:**

OpenMV4 集成了矩形识别 `find_rects` 函数，位于 `image` 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下识别函数相关说明，具体如下：

构造函数
<code>image.find_rects([roi=Auto, threshold=10000])</code>
矩形识别函数。返回一个 <code>image.rect</code> 矩形对象列表；
【 <code>roi</code> 】识别区域 ( <code>x,y,w,h</code> )，未指定则默认整张图片；
【 <code>threshold</code> 】阈值。返回大于或等于 <code>threshold</code> 的矩形，调整识别可信度；

## 使用方法

直接调用该函数。(大部分参数使用默认即可, 不支持压缩图像和 bayer 图像)

表 5-7 矩形识别函数

圆形检测的编程思路如下:

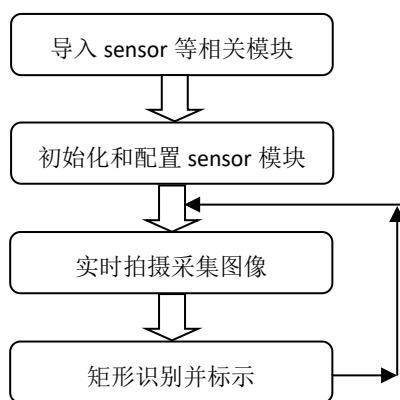


图 5-27 代码编写流程

官方参考代码路径【示例→Feature-Detection→find\_rects.py】，具体如下：

```
# 矩形识别例程

#
#这个例子演示了如何使用 quad 阈值在图像中找到矩形
#四元阈值检测算法检测矩形使用鲁棒的方式，是远远好于霍夫
#转换的方法。例如，它仍然可以检测矩形，即使镜头
#扭曲会使这些矩形看起来弯曲。所以圆角矩形是没有问题的！
#(鉴于此，代码也将检测小半径圆)...
#
#翻译：01Studio

import sensor, image, time

sensor.reset()
# grayscale is faster (160x120 max on OpenMV-M7)
```

```
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

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

    # `threshold` 需要设置一个比价大的值来过滤掉噪声。
    #这样在图像中检测到边缘亮度较低的矩形。矩形
    #边缘量级越大，对比越强...

    for r in img.find_rects(threshold = 10000):
        img.draw_rectangle(r.rect(), color = (255, 0, 0)) #画矩形显示
        for p in r.corners(): #四角画小圆形
            img.draw_circle(p[0], p[1], 5, color = (0, 255, 0))
        print(r)

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

- 实验结果：

使用 IDE 运行以上代码，可以看到实验现象如下：

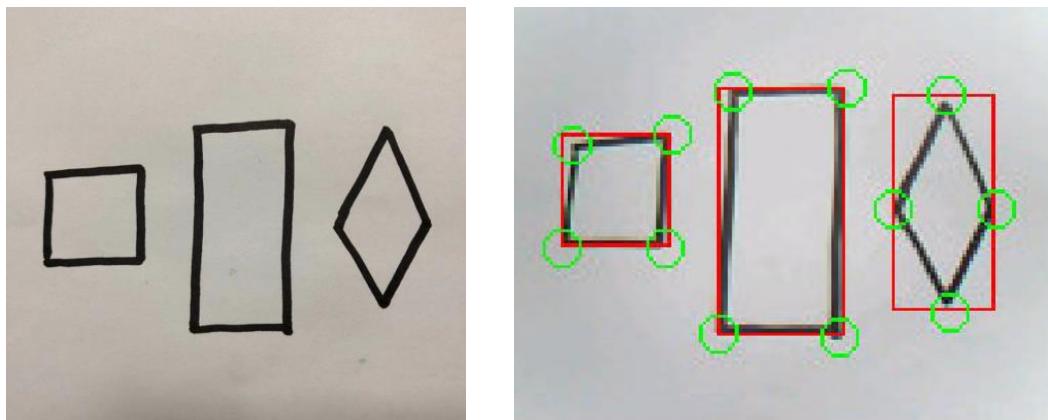


图 5-28 矩形识别实验：原图（左）和实验图片（右）

- 总结：

本节学习了矩形识别，可以使用该例程来识别矩形图案或者相关物体。

### 5.3.6 特征点识别

- **前言:**

在本章节中我们前面学习了几个常见的特征形状识别，都是检测跟预设图案特征来对比，而本节会有点区别，本节是先对采集到的图案或物体进行学习识别，然后当摄像头重新捕抓到跟学习对象一样的图片，就标识出来。

也就是说有一个先学习对象物体，再重新识别的过程，理论上这个方式能识别任何物体。

- **实验平台:**

pyAI-OpenMV4。



图 5-29 pyAI-OpenMV4

- **实验目的:**

编程实现学习和记录图像的特征点，当识别到相关特征点后用方框指示出来。

- **实验讲解:**

OpenMV4 集成了特征点识别 `find_keypoints` 函数，位于 `image` 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下识别函数相关说明，具体如下：

构造函数
<code>image.find_keypoints([roi[, threshold=20[, normalized=False[, scale_factor=1.5[, max_keypoints=100[, corner_detector=image.CORNER_A GAST]]]]]))</code>

特征点识别函数。返回一个 `image.rect` 矩形对象列表；

【`roi`】识别区域  $(x,y,w,h)$ ，未指定则默认整张图片；

【`threshold`】控制提取的数量的数字（取值 0-255）。对于默认的 AGAST 角点检测器，该值应在 20 左右。对于 FAST 角点检测器，该值约为 60-80。阈值越低，您提取的角点越多；

【`normalized`】布尔值。若为 `True`，在多分辨率下关闭提取键点。若您不关心处理扩展问题，且希望算法运行更快，就将之设置为 `True`。

【`scale_factor`】是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间。

【`max_keypoints`】是一个键点对象所能容纳的键点最大数量。若键点对象过大可能会导致内存问题，需要降低该值。

\*更多参数说明请阅读 OpenMV 官方文档：

文档链接：<http://docs.openmv.io/library/omv.image.html>

使用方法
直接调用该函数。（仅支持灰度图像）

表 5-8 特征点识别函数

`image` 同时包含了特征点对比函数 `match_descriptor`，用于比较 2 个特征点的相似程度，也就是判断是否一样，说明如下：

构造函数
<code>image.match_descriptor(descriptor0, descriptor1[, threshold=70[, fi lter_outliers=False]])</code>

特征点对比函数。返回 `kptmatch` 对象；

`【descriptor0】， 【descriptor1】`: 要对比的 2 个特征点；

`【threshold】`: 控制匹配程度。

### 使用方法

直接调用该函数。

表 5-9 特征点对比函数

从上表可以知道 `image.match_descriptor()` 函数返回的是 `kptmatch` 对象，对象说明如下：

构造函数
<code>image.kptmatch()</code>
特征点对象。由 <code>image.match_descriptor()</code> 函数返回
使用方法
<code>kptmatch.rect()</code>
返回特征点边界框。是一个矩形元组 ( <code>x, y, w, h</code> )
<code>kptmatch.cx()</code>
返回特征点中心 <code>x</code> 坐标位置 ( <code>int</code> )。也可以通过索引 [0] 取得这个值
<code>kptmatch.cy()</code>
返回特征点中心 <code>y</code> 坐标位置 ( <code>int</code> )。也可以通过索引 [1] 取得这个值
*更多参数说明请阅读 OpenMV 官方文档： <a href="http://docs.openmv.io/library/omv.image.html#class-kptmatch-keypoint-object">http://docs.openmv.io/library/omv.image.html#class-kptmatch-keypoint-object</a>

表 5-10 特征点对象

可以看到尽管特征识别相对于之前的实验变得复杂，但是由于 MicroPython 面向对象编程的优势，使用起来并没有复杂很多。依然是直接使用库。编程思路如下：

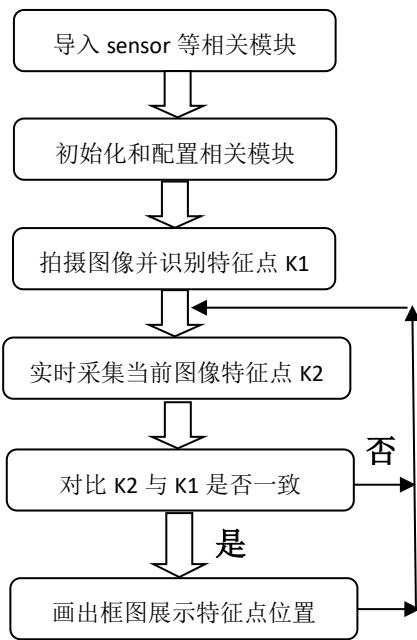


图 5-30 代码编写流程

官方参考代码路径【示例→Feature-Detection→find\_keypoints.py】，具体如下：

```

#通过特征点识别对象例程

#
#给摄像头展示一个对象，然后运行程序，程序将提取一组特征点。
#提取完成后将自动跟踪设备对象。如果需要重新识别请重新运行程序。
#注意：可以在文档中查看 find_keypoints 和 match_keypoints 两个调优参数。
#
#翻译：01Studio


import sensor, time, image

# 重启摄像头
sensor.reset()

# 摄像头初始化配置
sensor.set_contrast(3)

```

```
sensor.set_gainceiling(16)
sensor.set_framesize(sensor.VGA)
sensor.set_windowing((320, 240))
sensor.set_pixformat(sensor.GRAYSCALE)

sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False, value=100)

def draw_keypoints(img, kpts):
    if kpts:
        print(kpts)
        img.draw_keypoints(kpts)
        img = sensor.snapshot()
        time.sleep(1000)

kpts1 = None
# 提示：也可以使用下面语句从文件导入特征点。
#kpts1 = image.load_descriptor("/desc.orb")
#img = sensor.snapshot()
#draw_keypoints(img, kpts1)

clock = time.clock()

while (True):

    clock.tick()
    img = sensor.snapshot()
    if (kpts1 == None):
        # 提示：默认情况下 find_keypoints 会返回多种尺寸的特征点，识别灵活。
        kpts1 = img.find_keypoints(max_keypoints=150, threshold=10,
```

```
        scale_factor=1.2)

    draw_keypoints(img, kpts1)

else:

# 提示：当提取关键字以匹配第一个描述符时，我们使用 normalization =True

#方式来提取。关键点只来自第一个刻度，它将匹配第一个描述符中的一个刻度。

kpts2 = img.find_keypoints(max_keypoints=150, threshold=10,
                           normalized=True)

if (kpts2):

    match = image.match_descriptor(kpts1, kpts2, threshold=85)

    if (match.count()>10):

        # >10 判断为特征点一致。用户可以自行调整

        # 画圆和交叉用于展示特征点.

        img.draw_rectangle(match.rect())

        img.draw_cross(match.cx(), match.cy(), size=10)

    print(kpts2, "matched:%d dt:%d"%(match.count(),
                                    match.theta()))

    # 提示：如果你想绘制观点的，请取消下一行语句的注释

    #img.draw_keypoints(kpts2, size=KEYPOINTS_SIZE, matched=True)

# Draw FPS

img.draw_string(0, 0, "FPS:%.2f"%(clock.fps()))
```

- **实验结果：**

将摄像头正对要识别的对象，在 IDE 中运行代码，可以看到 OpenMV4 很快就识别了特征点 K1 并在 IDE 上显示。

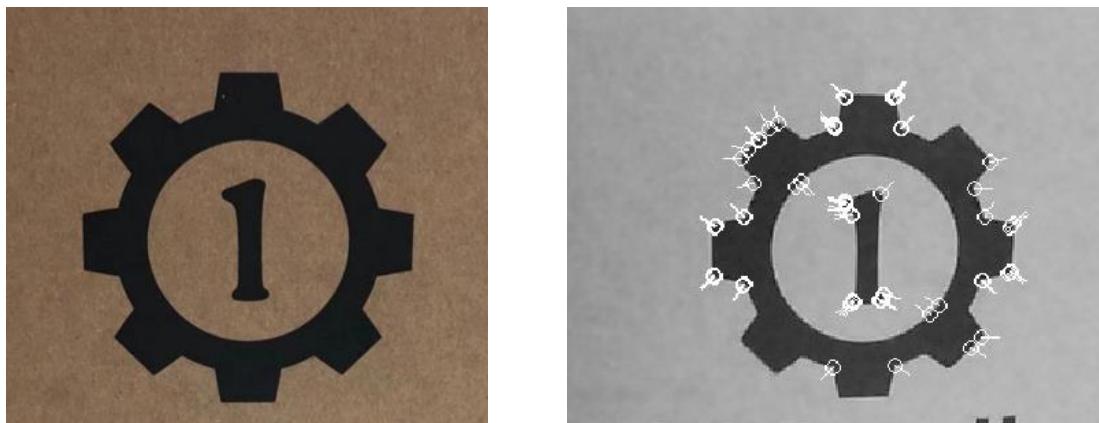


图 5-31 原图 (左) 和特征点学 K1 学习 (右)

学习完成后，我们将摄像头朝向不同的物体，可以看到当摄像头拍摄到我们学习的特征点 K1 时候，会通过矩形框和十字画图方式展示出来。

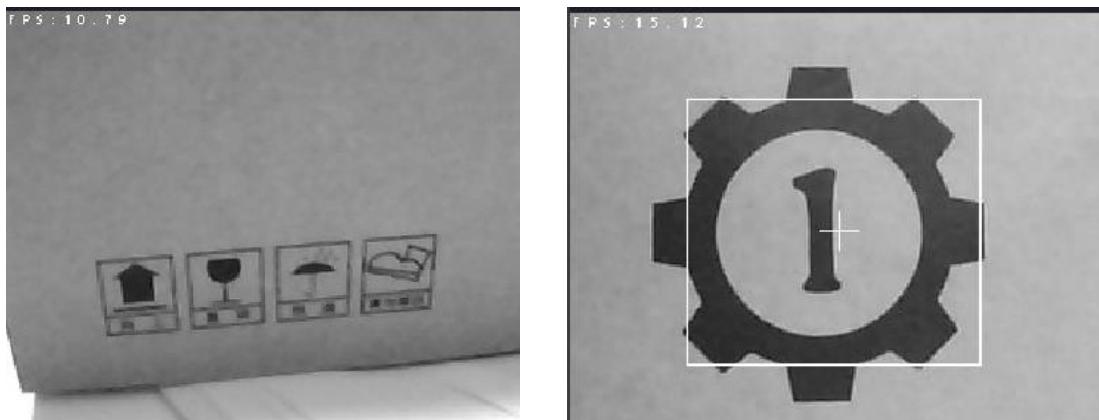


图 5-32 无效识别 (左) 与有效识别 (右)

- **总结：**

通过使用特征点学习函数和特征点对比函数，我们可以非常快速实现特征点的识别。

### 5.3.7 快速线性回归（巡线）

- **前言：**

快速线性回归的用途非常广泛，如比赛经常用到的小车、机器人巡线，可以通过线性回归的方式判断虚线和实线的轨迹。从而做出判断和响应。本节我们就来学习一下 OpenMV4 线性回归的编程方法。

- **实验平台：**

pyAI-OpenMV4。



图 5-33 pyAI-OpenMV4

- **实验目的：**

编程实现快速线性回归。

- **实验讲解：**

OpenMV4 集成了快速线性回归 `get_regression` 函数，位于 `image` 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下函数相关说明，具体如下：

构造函数
<code>image.get_regression(thresholds[, invert=False[, roi[, x_stride=2[, y_stride=1[, area_threshold=10[, pixels_threshold=10[, robust=False]]]]]]])</code>
线性回归计算。对图像所有阈值像素进行线性回归计算，通过最小二乘法进行，通常速度较快，但不能处理任何异常值；
【thresholds】必须是元组列表。 <code>(lo, hi)</code> 定义你想追踪的颜色范围。对于灰度图像，每个元组需要包含两个值：最小灰度值和最大灰度值。
*更多参数说明请阅读 OpenMV 官方文档： 文档链接： <a href="http://docs.openmv.io/library/omv.image.html">http://docs.openmv.io/library/omv.image.html</a>
使用方法
直接调用该函数。

表 5-11 线性回归函数

为了提高处理效果，我们可以先将图像变成二值（黑白）图像。方法如下：

构造函数
<code>image.binary(thresholds[, invert=False[, zero=False[, mask=None[, to_bitmap=False[, copy=False]]]]])</code>
图像二值（黑白）化。
【thresholds】必须是元组列表。 <code>(lo, hi)</code> 定义你想追踪的颜色范围。对于灰度图像，每个元组需要包含两个值：最小灰度值和最大灰度值。
例： <code>thresholds=(0,100)</code> ，则该函数表示将 <code>(0,100)</code> 灰度值范围变成白色。
*更多参数说明请阅读 OpenMV 官方文档： 文档链接： <a href="http://docs.openmv.io/library/omv.image.html">http://docs.openmv.io/library/omv.image.html</a>
使用方法
直接调用该函数。

表 5-12 图像二值化函数

编程思路如下：

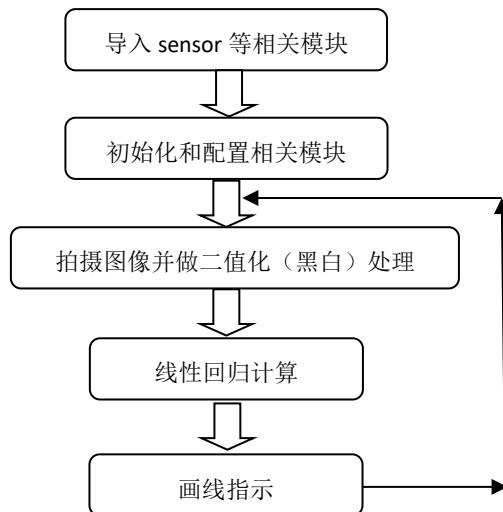


图 5-34 代码编写流程图

官方参考代码路径【示例→Feature-Detection→linear\_regression\_fast.py】，具体如下：

```
# 快速线性回归例程
#
#本例程展示了如何在 OpenMV Cam 上使用 get_regression()方法
#得到 ROI 的线性回归。使用此方法可以轻松构建
#一种机器人，它可以跟踪所有方向相同的线，在适当的线上使用 find_blobs()，以获得更好的过滤选项和控制。
#
#这方法被称为快速线性回归，因为我们使用了最小二乘
#方法来拟合直线。但是，这种方法并非适用于任何图像
#有很多异常点会破坏线的识别。
#
#翻译：01Studio

THRESHOLD = (0, 100) # 黑白图像的灰度阈值
```

```

BINARY_VISIBLE = True # 使用二值化图像你可以看到什么是线性回归。
# 这可能降低 FPS (每秒帧数) .

import sensor, image, time

sensor.reset()
sensor.set_pixformat(sensor.GRAYSCALE)
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

while(True):

    clock.tick()

    #image.binary([THRESHOLD])将灰度值在 THRESHOLD 范围变成了白色
    img = sensor.snapshot().binary([THRESHOLD]) if BINARY_VISIBLE else
                                                    sensor.snapshot()

    # 返回一个类似 find_lines() 和 find_line_segments() 的对象。
    # 有以下函数使用方法: x1(), y1(), x2(), y2(), length(),
    # theta() (rotation in degrees), rho(), and magnitude().
    #
    # magnitude() 代表线性回归的指令，其值为(0, INF]。
    # 0 表示一个圆，INF 数值越大，表示线性拟合的效果越好。

    line = img.get_regression([(255,255) if BINARY_VISIBLE else
                                THRESHOLD])

    if (line): img.draw_line(line.line(), color = 127)

```

```
print("FPS %f, mag = %s" % (clock.fps(), str(line.magnitude()) if  
                                (line) else "N/A"))  
  
# 有关 rho 负值说明：  
# [theta+0:-rho] 元组和 [theta+180:+rho]是一样的.
```

### ● 实验结果：

运行代码，我们分别来测试一下实线和虚线线性回归效果，结果如下：

#### 虚线测试：

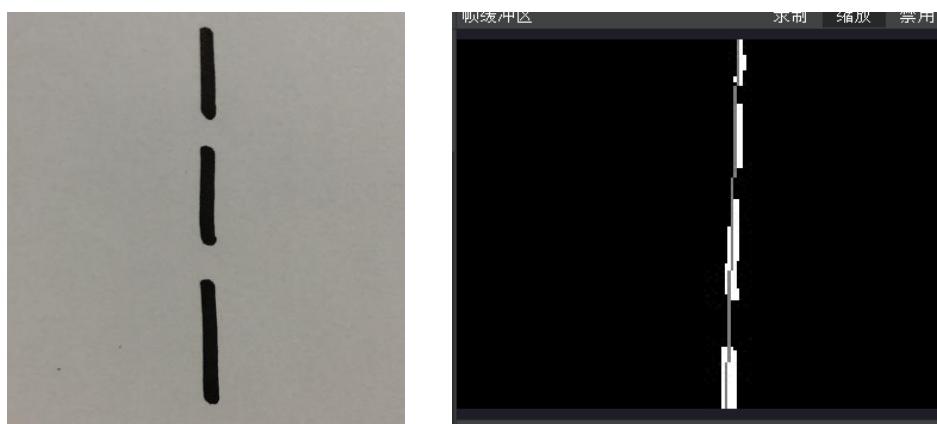


图 5-35 虚线直线测试原图（左）和实验图片（右）



图 5-36 虚线曲线测试原图（左）和实验图片（右）

### 实线测试:

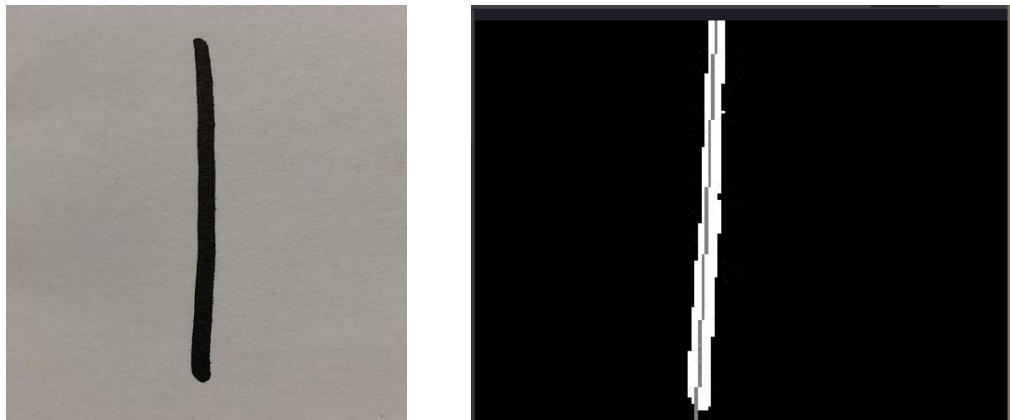


图 5-37 实线直线测试原图 (左) 和实验图片 (右)

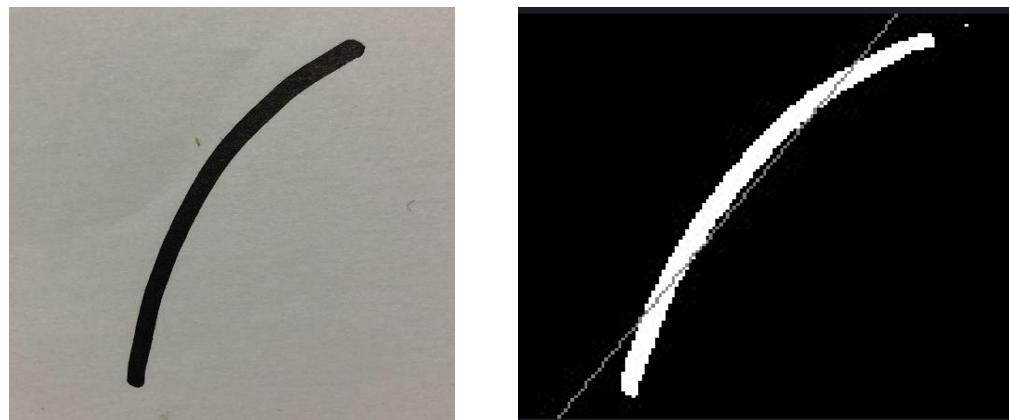


图 5-38 实线曲线测试原图 (左) 和实验图片 (右)

### ● 总结:

可以看到通过 MicroPython 编程实现的线性回归实验非常容易，效果也是非常不错。

## 5.4 颜色追踪

颜色追踪，要做的就是对图像指定的颜色进行检测识别，如本节包含的RGB565 彩色自动追踪、机器人巡线、颜色识别等都是非常有代表性的实验。特征检测的全部官方例程位于 Color-Tracking 例程下，本节挑选重点实验进行讲解。

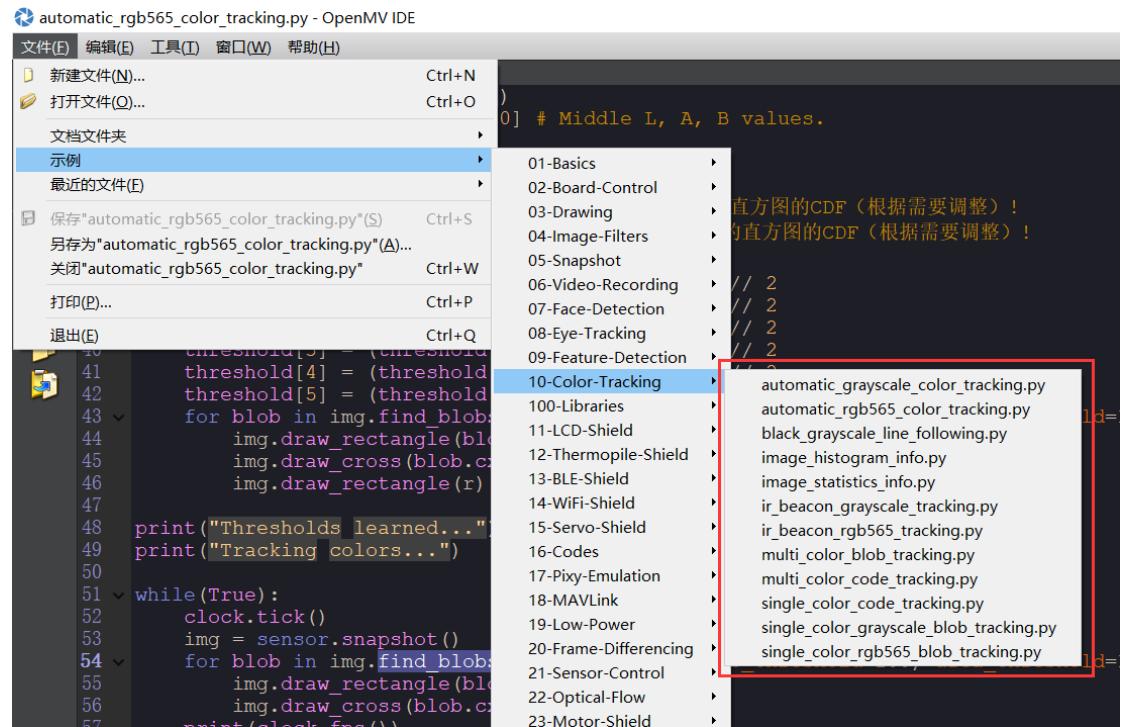


图 5-39 特征检测官方例程位置

### 5.4.1 RGB565 彩色自动追踪

- **前言:**

本节讲述颜色识别，我们会让 OpenMV4 先学习一种摄像头上的颜色，然后再找出新拍摄图像中所有相同颜色的色块。这可以说让摄像头从色盲到开始认识颜色的过程，非常有趣。

- **实验平台:**

pyAI-OpenMV4。

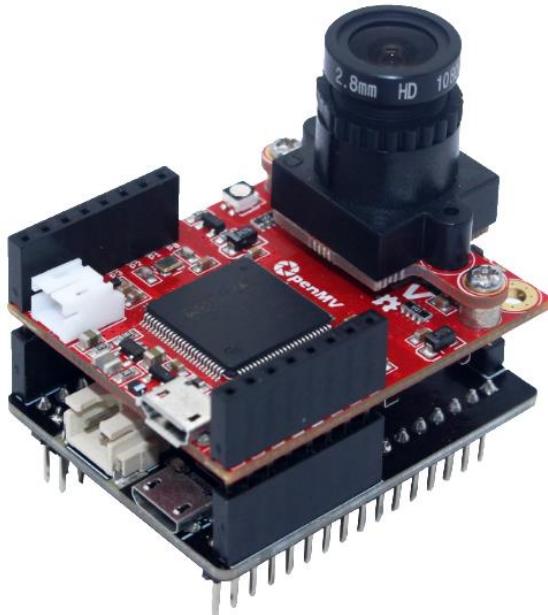


图 5-40 pyAI-OpenMV4

- **实验目的:**

通过编程实现 OpenMV4 识别图像中学习到的颜色色块。

- **实验讲解:**

OpenMV4 集成了 RGB565 颜色块识别 `find_blobs` 函数，主要是基于 LAB 颜色模型（有兴趣的用户可以自行查阅相关模型资料）。其位于 `image` 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下本实验相关对象和函数说明，具体如下：

## 构造函数

```
image.find_blobs(thresholds[, invert=False[, roi[, x_stride=2[, y_
stride=1[, area_threshold=10[, pixels_threshold=10[, merge=False[,_
margin=0[, threshold_cb=None[, merge_cb=None]]]]]]]]])
```

查找图像中指定的色块。返回 `image.blob` 对象列表；

**【thresholds】** 必须是元组列表。 `[(lo, hi), (lo, hi), ..., (lo, hi)]` 定义你想追踪的颜色范围。 对于灰度图像，每个元组需要包含两个值 - 最小灰度值和最大灰度值。 仅考虑落在这些阈值之间的像素区域。 对于 RGB565 图像，每个元组需要有六个值(`l_lo, l_hi, a_lo, a_hi, b_lo, b_hi`) - 分别是 LAB L, A 和 B 通道的最小值和最大值。

**【area\_threshold】** 若色块的边界框区域小于此参数值，则会被过滤掉；

**【pixels\_threshold】** 若色块的像素数量小于此参数值，则会被过滤掉；

**【merge】** 若为 `True`, 则合并所有没有被过滤的色块；

**【margin】** 调整合并色块的边缘。

\*更多参数说明请阅读 OpenMV 官方文档：

文档链接：<http://docs.openmv.io/library/omv.image.html>

## 使用方法

以上函数返回 `image.blob`。

`blob.rect()`

返回一个矩形元组 `(x, y, w, h)`，如色块边界。

`blob.cx()`

返回色块(`int`)的中心 x 位置。

`blob.cy()`

返回色块(`int`)的中心 y 位置。

\*更多使用说明请阅读 OpenMV 官方文档：

文档链接：<http://docs.openmv.io/library/omv.image.html#class-blob-blob-object>

表 5-13 `find_blobs` 函数说明

了解了找色块函数应用方法后，我们可以理清一下编程思路，上电后先让OpenMV4学习摄像头当前图像指定小区域的颜色，学习完成后，进入循环检测，当检测到新的图像中色块颜色跟所学习色块颜色一致时，用矩形和十字交叉表示出来，从而实现颜色追踪。代码编写流程如下：

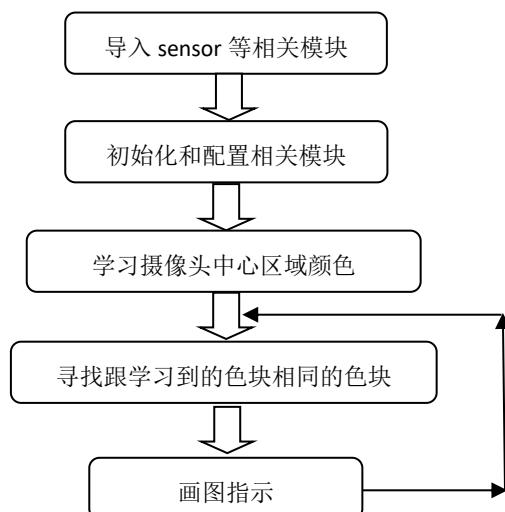


图 5-41 代码编写流程图

官方参考代码路径【示例→Color-Tracking→  
automatic\_rgb565\_color\_tracking.py】，具体如下：

```
# RGB565 彩色自动追踪

#
# 本例程展示了单个颜色追踪。
#
#翻译: 01Studio

import sensor, image, time
print("Letting auto algorithms run. Don't put anything in front of the
camera!")
```

```

#摄像头初始化

sensor.reset()

sensor.set_pixformat(sensor.RGB565)

sensor.set_framesize(sensor.QVGA)

sensor.skip_frames(time = 2000)

sensor.set_auto_gain(False) # 颜色追踪必须关闭摄像头的自动增益

sensor.set_auto_whitebal(False) # 颜色追踪必须关闭自动白平衡

clock = time.clock()

#捕获摄像头中心区域的颜色阈值.50*50 中心区域。

r = [(320//2)-(50//2), (240//2)-(50//2), 50, 50]

#将需要识别的颜色放在摄像头中心

print("Auto algorithms done. Hold the object you want to track in front
      of the camera in the box.")

print("MAKE SURE THE COLOR OF THE OBJECT YOU WANT TO TRACK IS FULLY
      ENCLOSED BY THE BOX!")

for i in range(60):

    img = sensor.snapshot()

    img.draw_rectangle(r)

print("Learning thresholds...")

threshold = [50, 50, 0, 0, 0, 0] # Middle L, A, B values.

for i in range(60):

    img = sensor.snapshot()

    hist = img.get_histogram(roi=r)

    # 获取 1% 范围的直方图的 CDF (根据需要调整) !

    lo = hist.get_percentile(0.01)

    # 获取 99% 范围的直方图的 CDF (根据需要调整) !

```

```

hi = hist.get_percentile(0.99)

# 百分位数平均值.

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

for blob in img.find_blobs([threshold], pixels_threshold=100,
                           area_threshold=100, merge=True, margin=10):
    img.draw_rectangle(blob.rect())
    img.draw_cross(blob.cx(), blob.cy())
    img.draw_rectangle(r)

print("Thresholds learned...")
print("Tracking colors...")

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

#跟学习到的色块 LAB 模型[Threshold]做对比

for blob in img.find_blobs([threshold], pixels_threshold=100,
                           area_threshold=100, merge=True, margin=10):
    img.draw_rectangle(blob.rect())
    img.draw_cross(blob.cx(), blob.cy())
    print(clock.fps())

```

- **实验结果：**

运行代码，当摄像头中心出现小矩形区域时候，将区域对准要学习的颜色色块，这里学习黄色：



图 5-42 学习黄色区域

注意学习过程需要保存摄像头不动，直到今日检测判断模式，串口出现打印 FPS（每秒帧数），如下图。

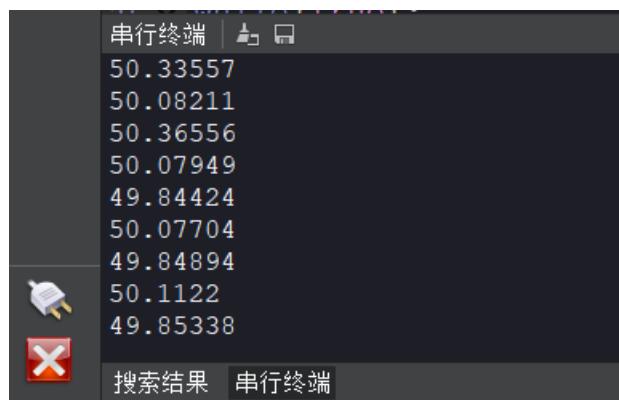


图 5-43

学习完成后，我们将摄像头拍照相关颜色物体，可以看到检测到学习的色块（黄色）。具体如下：



图 5-44 色块追踪 原图 (左) 和实验图片 (右)

● 总结：

本节学习了通过 MicroPython 编程在 OpenMV4 上实现色块追踪。本实验主要是基于 LAB 颜色模型来判断。有兴趣的小伙伴可以自行查阅 LAB 模块相关资料，再结合打印 `threshold` 查看其元组数据来深入学习。

## 5.4.2 机器人巡线（实线巡线）

- **前言：**

之前我们做过巡线实验，是利用线性回归方法测量的。本节我们将学习一个新方法，那就是直接根据摄像头采集到的图像直线与中心偏离的位置计算出偏离角度，这个方法让选线变得更容易。如果你的 OpenMV4 连接到机器人（或相关设备），那么机器人（设备）可以直接根据计算角度偏离结果做出相应调整。

- **实验平台：**

pyAI-OpenMV4。



图 5-45 pyAI-OpenMV4

- **实验目的：**

编程实现 OpenMV4 计算画面中黑线的偏离角度。

- **实验讲解：**

本实验对画面是有一定要求的，也就是摄像头采集图像一定要出现唯一一条连续的直线。程序通过对画面切割成三部分，计算每个部分黑色线的中心点 X 坐标，然后采用加权平均算法估算出直线的偏离位置。通常情况下越靠近底部的地方离摄像头越近，顶部表示远方线段。因此底部的图形权重高。以下下是示意图

讲解：

假设摄像头当前画面的像素是 120（宽） $\times$ 160（高），左上角坐标为 (0,0)，然后当前出现直线坐标为 (80,120) 至 (160,0) 偏右的直线。上中下三个部分的权重分别为 0.1、0.2、0.7（底部图像靠近机器人，权重大，权重总和不一定为 1），我们来计算一下其中心值。

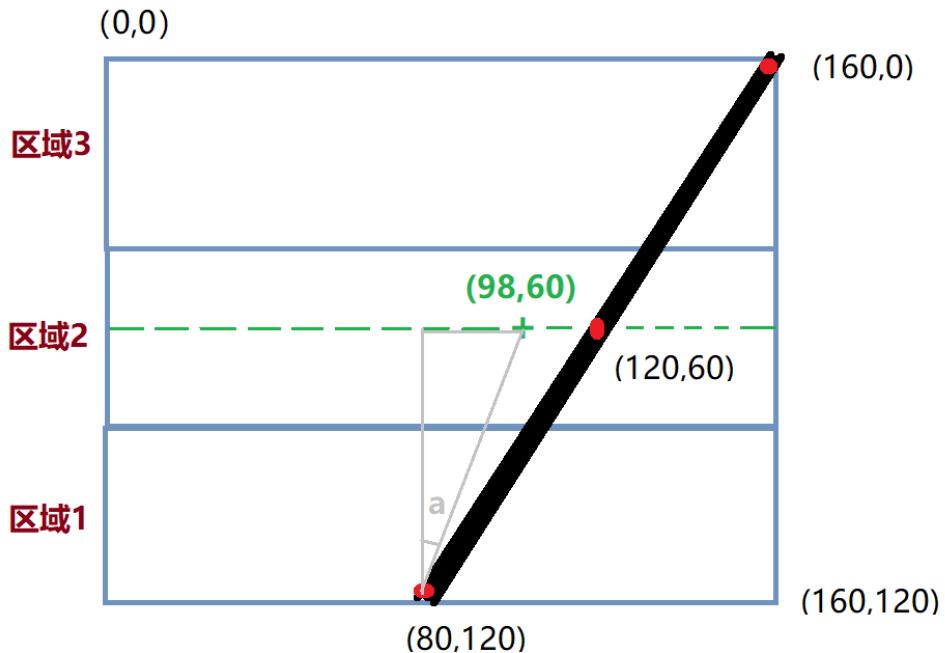


图 5-46 直线偏离计算示意图

我们默认了 Y 轴的中点坐标就是 60，X 坐标加权平均值计算如下：

$$X' = (80 \cdot 0.7 + 120 \cdot 0.3 + 160 \cdot 0.1) / (0.7 + 0.3 + 0.1) = 98$$

那么直线偏离坐标可以认为是 (98,60)，图中绿色“+”位置。那么利用反正切函数可以求出偏离角度： $a = \text{atan}((98-80)/60)=16.7^\circ$ ，机器人相当于实线的位置往左偏了，所以加一个负号，即  $-16.7^\circ$ ；偏离角度就是这么计算出来的。

本实验主要采用 `find_blobs` 函数编程，这个函数在上一节已经有讲述，具体使用方法如下表：

构造函数
<code>image.find_blobs(thresholds[], invert=False[], roi[], x_stride=2[], y_stride=1[], area_threshold=10[], pixels_threshold=10[], merge=False[], margin=0[], threshold_cb=None[], merge_cb=None]]]]]]])</code>

查找图像中指定的色块。返回 `image.blob` 对象列表；

**【thresholds】** 必须是元组列表。 `[(lo, hi), (lo, hi), ..., (lo, hi)]` 定义你想追踪的颜色范围。 对于灰度图像，每个元组需要包含两个值 - 最小灰度值和最大灰度值。 仅考虑落在这些阈值之间的像素区域。 对于 RGB565 图像，每个元组需要有六个值(`l_lo, l_hi, a_lo, a_hi, b_lo, b_hi`) - 分别是 LAB L, A 和 B 通道的最小值和最大值。

**【area\_threshold】** 若色块的边界框区域小于此参数值，则会被过滤掉；

**【pixels\_threshold】** 若色块的像素数量小于此参数值，则会被过滤掉；

**【merge】** 若为 `True`,则合并所有没有被过滤的色块；

**【margin】** 调整合并色块的边缘。

\*更多参数说明请阅读 OpenMV 官方文档：

文档链接：<http://docs.openmv.io/library/omv.image.html>

## 使用方法

以上函数返回 `image.blob`。

**blob.rect()**

返回一个矩形元组 `(x, y, w, h)` ,如色块边界。

**blob.cx()**

返回色块(`int`)的中心 x 位置。

**blob.cy()**

返回色块(`int`)的中心 y 位置。

\*更多使用说明请阅读 OpenMV 官方文档：

文档链接：<http://docs.openmv.io/library/omv.image.html#class-blob-blob-object>

表 5-14

编程思路如下：

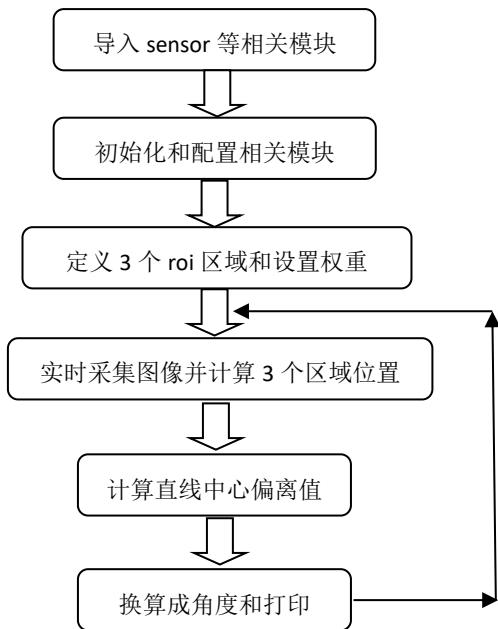


图 5-47 代码编写流程图

官方参考代码路径【示例→Color-Tracking→

`black_grayscale_line_following.py`】, 具体如下:

```

# 黑色灰度线巡线跟踪示例
#
#做一个跟随机器人的机器人需要很多的努力。这个示例脚本
#演示了如何做机器视觉部分的线跟随机器人。你
#可以使用该脚本的输出来驱动一个差分驱动机器人
#跟着一条线走。这个脚本只生成一个表示的旋转值（偏离角度）
#你的机器人向左或向右。
#
# 为了让本示例正常工作，你应该将摄像头对准一条直线（实线）
#并将摄像头调整到水平面 45 度位置。请保证画面内只有 1 条直线。
#
#翻译：01Studio

import sensor, image, time, math
  
```

```

# 追踪黑线。使用 [(128, 255)] 追踪白线。
GRAYSCALE_THRESHOLD = [(0, 64)]


# 下面是一个 roi 【区域】元组列表。每个 roi 用 (x, y, w, h) 表示的矩形。
# 本例程的采样图像为 160*120，列表把 roi 把图像分成 3 个矩形，越靠近的摄像头
# 视野（通常为图像下方）的矩形权重越大。

ROIS = [ # [ROI, weight]
    (0, 100, 160, 20, 0.7), # 可以根据不同机器人情况进行调整。
    (0, 50, 160, 20, 0.3),
    (0, 0, 160, 20, 0.1)
]

# 计算以上 3 个矩形的权值【weight】的和，和不需要一定为 1.

weight_sum = 0

for r in ROIS: weight_sum += r[4] # r[4] 为矩形权重值.


# 摄像头配置

sensor.reset() # 初始化.

sensor.set_pixformat(sensor.GRAYSCALE) # 使用灰度图像.

sensor.set_framesize(sensor.QQVGA) # 使用 QQVGA 分辨率 (160*120) .

sensor.skip_frames(time = 2000) # 等待摄像头稳定.

sensor.set_auto_gain(False) # 必须关闭自动增益
sensor.set_auto_whitebal(False) # 必须关闭白平衡
clock = time.clock() # Tracks FPS.

while(True):
    clock.tick() # Track elapsed milliseconds between snapshots().
    img = sensor.snapshot() # 实时拍照.

```

```

centroid_sum = 0

for r in ROIS:

    blobs = img.find_blobs(GRAYSCALE_THRESHOLD, roi=r[0:4],
                           merge=True) # r[0:4] 是上面定义的 roi 元组.

    if blobs:

        # Find the blob with the most pixels.
        largest_blob = max(blobs, key=lambda b: b.pixels())

        # Draw a rect around the blob.
        img.draw_rectangle(largest_blob.rect())
        img.draw_cross(largest_blob.cx(),
                      largest_blob.cy())

        # r[4] 是每个 roi 的权重值.
        centroid_sum += largest_blob.cx() * r[4]

    center_pos = (centroid_sum / weight_sum) # 确定直线的中心.

    # 将直线中心位置转换成角度，便于机器人处理.
    deflection_angle = 0

    # 使用反正切函数计算直线中心偏离角度。可以自行画图理解
    # 权重 X 坐标落在图像左半部分记作正偏，落在右边部分记为负偏，
    # 所以计算结果加负号。
    deflection_angle = -math.atan((center_pos-80)/60)

    # 将偏离值转换成偏离角度.

    deflection_angle = math.degrees(deflection_angle)

```

```

# 计算偏离角度后可以控制机器人进行调整.

print("Turn Angle: %f" % deflection_angle)

print(clock.fps()) # 连接计算机后 FPS 速度减半。

```

### ● 实验结果：

运行程序，分别观察摄像头采集到没偏移、左偏和右偏各个直线的实验结果。

#### 机器人无偏移：

可以看到偏移角度接近  $0^\circ$ ；

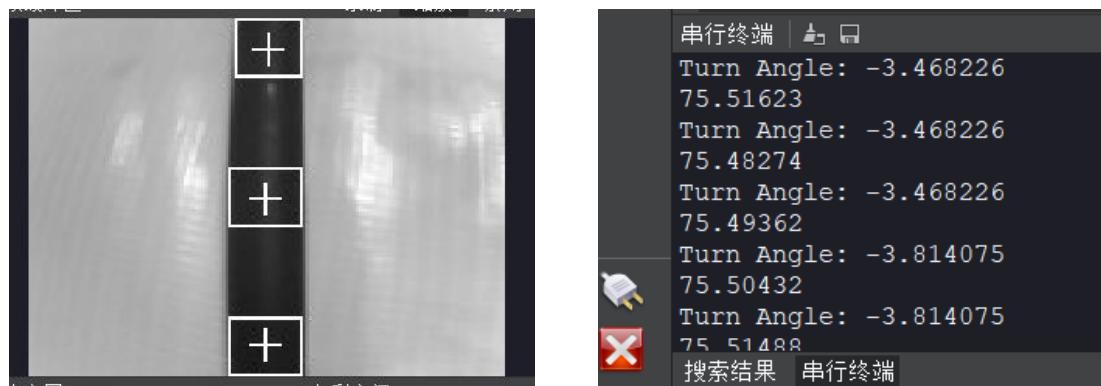


图 5-48 实验图片（左）和偏移角度（右）

#### 机器人左偏：

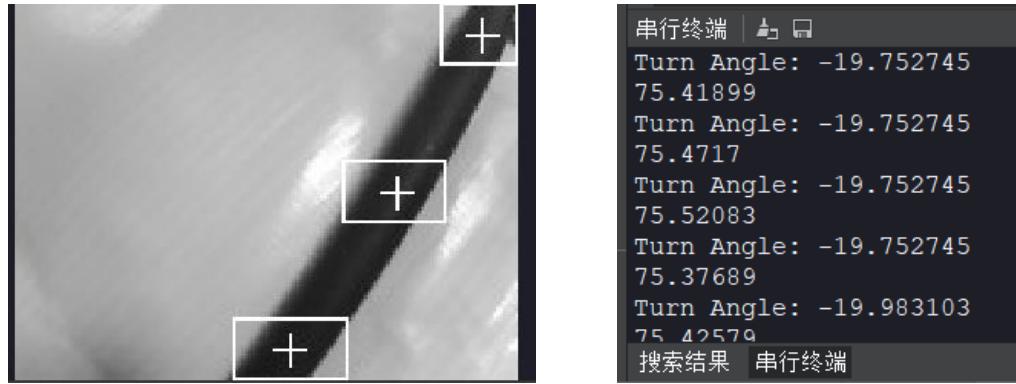


图 5-49 实验图片（左）和偏移角度（右）

### 机器人右偏:

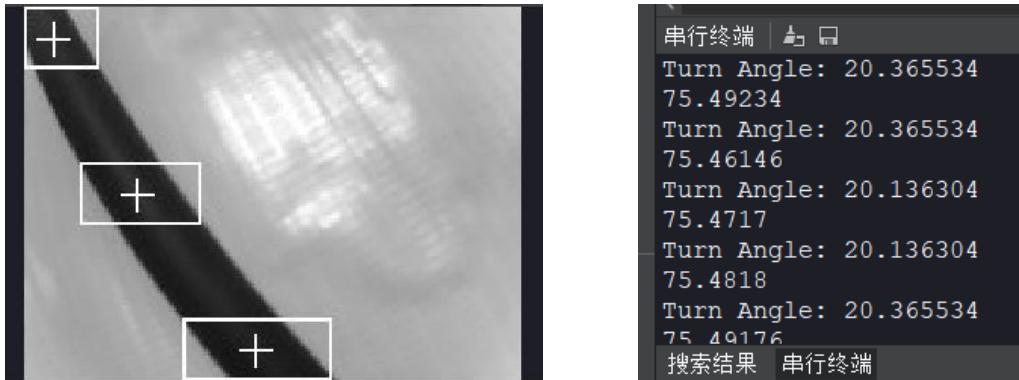


图 5-50 实验图片 (左) 和偏移角度 (右)

### ● 总结:

从本节可以看到，OpenMV4 算法应用简单便捷，我们通过库可以轻松的实现图像特定的采集方式和计算，以便将精力都放在数学模型上。

### 5.4.3 单个颜色识别

- **前言:**

我们前面学习了颜色自动追踪，这一节的区别就是指定 OpenMV4 去识别系统预先设定的颜色，如红、绿、蓝。这样当摄像头打开后就能自动识别了。

- **实验平台:**

pyAI-OpenMV4 开发板。



图 5-51 pyAI-OpenMV4

- **实验目的:**

通过编程实现 OpenMV4 识别程序预先设定的颜色色块，分别是红、绿、蓝三种颜色。

- **实验讲解:**

前面已经学习过 OpenMV4 集成了颜色块识别 `find_blobs` 函数，主要还是基于 LAB 颜色模型（有兴趣的用户可以自行查阅相关模型资料）。其位于 `image` 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下本实验相关对象和函数说明，具体如下：

## 构造函数

```
image.find_blobs(thresholds[, invert=False[, roi[, x_stride=2[, y_
stride=1[, area_threshold=10[, pixels_threshold=10[, merge=False[,_
margin=0[, threshold_cb=None[, merge_cb=None]]]]]]]]])
```

查找图像中指定的色块。返回 `image.blob` 对象列表；

**【thresholds】** 必须是元组列表。 `[(lo, hi), (lo, hi), ..., (lo, hi)]` 定义你想追踪的颜色范围。 对于灰度图像，每个元组需要包含两个值 - 最小灰度值和最大灰度值。 仅考虑落在这些阈值之间的像素区域。 对于 RGB565 图像，每个元组需要有六个值(`l_lo, l_hi, a_lo, a_hi, b_lo, b_hi`) - 分别是 LAB L, A 和 B 通道的最小值和最大值。

**【area\_threshold】** 若色块的边界框区域小于此参数值，则会被过滤掉；

**【pixels\_threshold】** 若色块的像素数量小于此参数值，则会被过滤掉；

**【merge】** 若为 `True`,则合并所有没有被过滤的色块；

**【margin】** 调整合并色块的边缘。

\*更多参数说明请阅读 OpenMV 官方文档：

文档链接：<http://docs.openmv.io/library/omv.image.html>

## 使用方法

以上函数返回 `image.blob`。

**blob.rect()**

返回一个矩形元组 `(x,y,w,h)` ,如色块边界。

**blob.cx()**

返回色块(`int`)的中心 x 位置。

**blob.cy()**

返回色块(`int`)的中心 y 位置。

**blob.elongation()**

用于块形状识别。返回介于 `0-1` 的值， `0` 接近圆、`>0.5` 可以看成矩形、`1` 表示线；

**blob.min\_corners()**

始终返回一个矩形边界 4 个 `(x,y)` 元组。常用不规则形状补偿成矩形框展示；

<code>blob.major_axis_line()</code>
返回一个线元组(x1,y1,x2,y2)。可以看做 <code>blob.min_corners()</code> 的最长边;
<code>blob.minor_axis_line()</code>
返回一个线元组(x1,y1,x2,y2)。可以看做 <code>blob.min_corners()</code> 的最短边;
*更多使用说明请阅读 OpenMV 官方文档: 文档链接: <a href="http://docs.openmv.io/library/omv.image.html#class-blob-blob-object">http://docs.openmv.io/library/omv.image.html#class-blob-blob-object</a>

表 5-15 find\_blobs 函数说明

了解了找色块函数应用方法后，我们可以理清一下编程思路，代码编写流程如下：

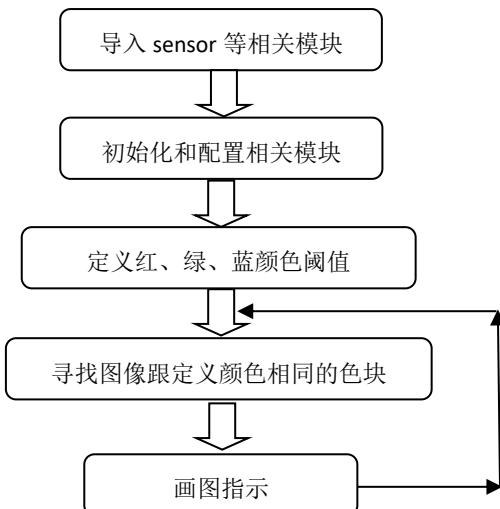


图 5-52 代码编写流程图

官方参考代码路径【示例→Color-Tracking→

`single_color_rgb565_blog_tracking.py`】，具体如下：

```

# 单个颜色块识别例程
#
# 这个例程使用 OpenMV 来识别单种颜色。
#
#翻译: 01Studio

import sensor, image, time, math
  
```

```
threshold_index = 0 # 0 红色, 1 绿色, 2 蓝色

# 颜色识别阈值 (L Min, L Max, A Min, A Max, B Min, B Max) LAB 模型
# 下面的阈值元组是用来识别 红、绿、蓝三种颜色, 当然你也可以调整让识别变得更好。
thresholds = [(30, 100, 15, 127, 15, 127), # 红色阈值
               (30, 100, -64, -8, -32, 32), # 绿色阈值
               (0, 30, 0, 64, -128, 0)] # 蓝色阈值

#摄像头初始化
sensor.reset()

sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)

sensor.set_auto_gain(False) # 摄像头必须关闭自动增益
sensor.set_auto_whitebal(False) # 摄像头必须关闭白平衡
clock = time.clock()

# "pixel_threshold" 和 "area_threshold" 参数用于控制返回确定色块的数量。
# "merge=True" 用于合并找到的色块。

while(True):

    clock.tick()

    img = sensor.snapshot()

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

        # blob.elongation() > 0.5 来判断颜色块不是圆形的时候。
```

```

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))

# 下面这些画图展示一直会出现.

img.draw_rectangle(blob.rect())

img.draw_cross(blob.cx(), blob.cy())

# 颜色块旋转角度只有 0-180 度.

img.draw_keypoints([(blob.cx(), blob.cy(),
                     int(math.degrees(blob.rotation())))], size=20)

print(int(math.degrees(blob.rotation())))

print(clock.fps())

```

### ● 实验结果：

我们可以将代码中的 `threshold_index` 变量修改成 0、1、2 然后运行，这分别代表识别红色、绿色和蓝色。

```
threshold_index = 0 # 0 红色, 1 绿色, 2 蓝色
```

红色识别：

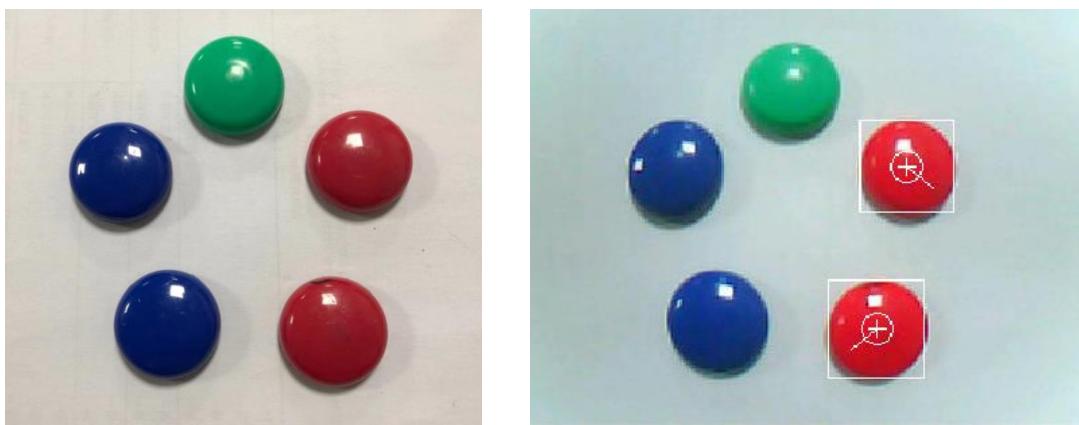


图 5-53 红色识别原图 (左) 和实验图片 (右)

### 绿色识别:

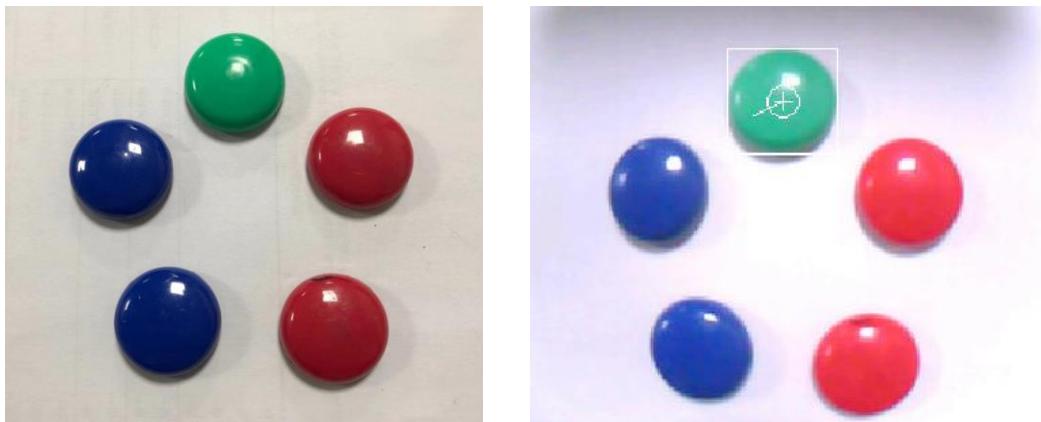


图 5-54 绿色识别原图 (左) 和实验图片 (右)

### 蓝色识别:

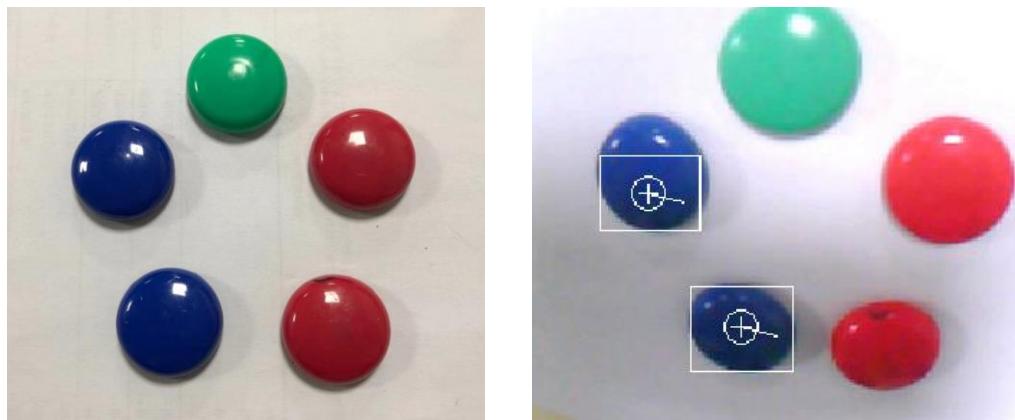


图 5-55 蓝色识别原图 (左) 和实验图片 (右)

### ● 总结:

本节学习了通过 MicroPython 编程在 OpenMV4 上实现单种颜色识别。本实验主要是基于 LAB 颜色模型来判断。有兴趣的小伙伴可以自行查阅 LAB 模块相关资料，再结合打印 threshold 查看其元组数据来深入学习。

还有就是可以找多种不规则形状的颜色物体对比学习。

## 5.5 人脸检测

人脸检测类实验主要是人脸检测和人脸特征识别，这是在安防、支付等领域非常流行的应用。全部官方例程位于 Face-Detection 例程下，本节挑选重点实验进行讲解。

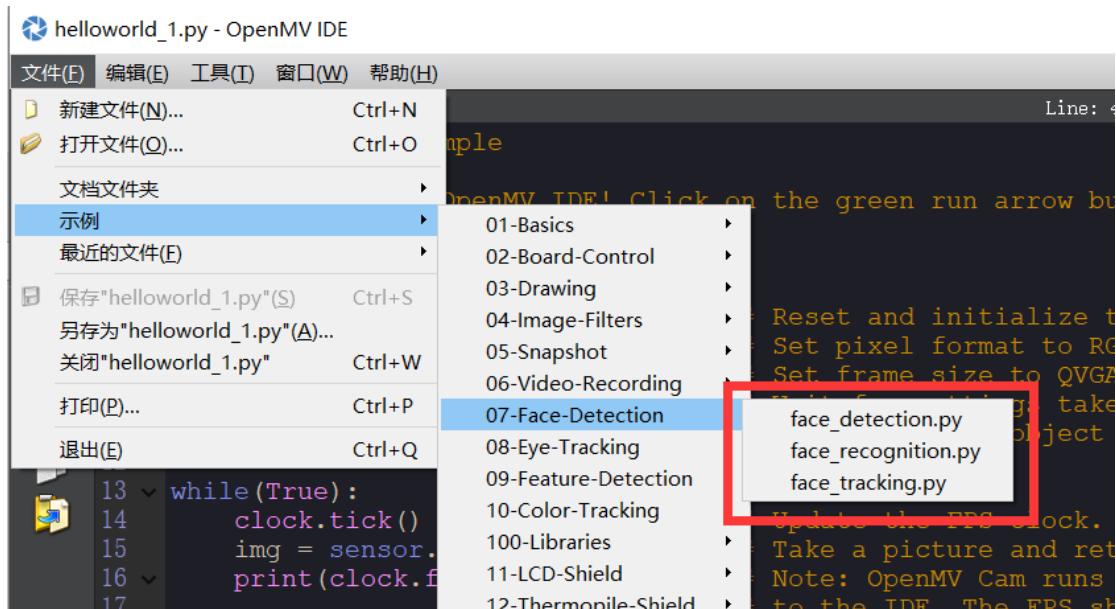


图 5-56 人脸检测官方例程位置

### 5.5.1 人脸检测

- **前言:**

人脸检测，简单来说就是告诉你将一幅图片中哪些是人脸。今天我们来学习一下如何通过 MicroPython 编程快速实现人脸识别。

- **实验平台:**

pyAI-OpenMV4。



图 5-57 pyAI-OpenMV4

- **实验目的:**

将摄像头拍摄到的画面中的人脸用矩形框表示出来。

- **实验讲解:**

人脸识别本质是特征识别，OpenMV4 已经集成了非常多的特征库和算法库，人生苦短，通过 MicroPython 编程我们可以快速实现人脸识别应用。其使用的是 image 模块下的 find\_features() 特征寻找函数，说明如下：

## 构造函数

```
image.find_features(cascade[, threshold=0.5[, scale=1.5[, roi]]])
```

搜索和 Haar Cascade 匹配的所有区域的图像，并返回一个关于这些特征的边界框矩形元组(x, y, w, h)的列表。若未发现任何特征，则返回一个空白列表。

**【cascade】** Haar Cascade 对象

**【threshold】** 是浮点数 (0.0-1.0)，其中较小的值在提高检测速率同时增加误报率。相反，较高的值会降低检测速率，同时降低误报率。;

**【scale】** 是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间；

**【roi】** 指定识别区域的矩形元组(x, y, w, h)。如果未指定，roi 即整个图像的图像矩形。

## 使用方法

直接调用该函数。建议使用灰度图像。

表 5-16 寻找人脸特征函数

从上表可以看到人脸检测的模型和算法已经在 OpenMV4 里面内置，我们直接调用即可，具体编程思路如下：

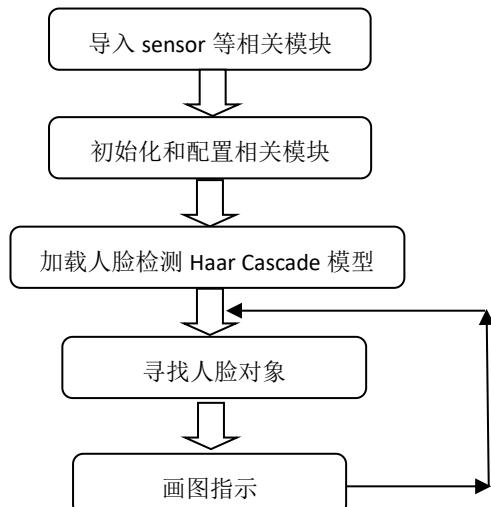


图 5-58 代码编写流程图

官方参考代码路径【示例→Face-Detection→ face\_detection.py】，具体如下：

```
# 人脸检测例程

#
# 这个例子展示了 OpenMV 内置的人脸检测功能。

#
# 人脸检测是通过在图像上使用 Haar Cascade 特征检测器来实现的。

# 一个 Haar Cascade 是一系列简单区域的对比检查。

# 人脸识别有 25 个阶段，每个阶段有几百次检测。Haar Cascades 运行很快因为是逐个
# 阶段递进检测的。此外，OpenMV Cam 使用一种称为积分图像的数据结构来在恒定时间
# 内快速执行每个区域的对比度检查(特征检测需要用灰度图像的原因是因为图像积分需
# 要更多空间)。

#
#翻译和注释: 01Studio

import sensor, time, image

# Reset sensor
sensor.reset()

# Sensor settings
sensor.set_contrast(1) #设置相机图像对比度为 1
sensor.set_gainceiling(16) #设置相机图像增益上限为 16
# HQVGA 和 GRayscale 是人脸检测最佳配置。
sensor.set_framesize(sensor.HQVGA)
sensor.set_pixformat(sensor.GRAYSCALE)

# 加载 Haar Cascade 模型
# 默认使用 25 个步骤，减少步骤会加快速度但会影响识别成功率。
face_cascade = image.HaarCascade("frontalface", stages=25)
print(face_cascade)
```

```
# FPS clock  
clock = time.clock()  
  
while (True):  
    clock.tick()  
  
    # Capture snapshot  
    img = sensor.snapshot()  
  
    # 找人脸对象。  
    # threshold 和 scale_factor 两个参数控制着识别的效率和准确性。  
    objects = img.find_features(face_cascade, threshold=0.75,  
                                scale_factor=1.25)  
  
    # 画图展示对象（将人脸用矩形展示出来）  
    for r in objects:  
        img.draw_rectangle(r)  
  
    # Print FPS.  
    # 注意：连接电脑会降低 FPS。  
    print(clock.fps())
```

### ● 实验结果：

运行代码，将摄像头正对自己，可以看到将自己的脸检测出来。我们用图片演示一下单个人脸和多个人脸检测的场景。

单个人脸检测:

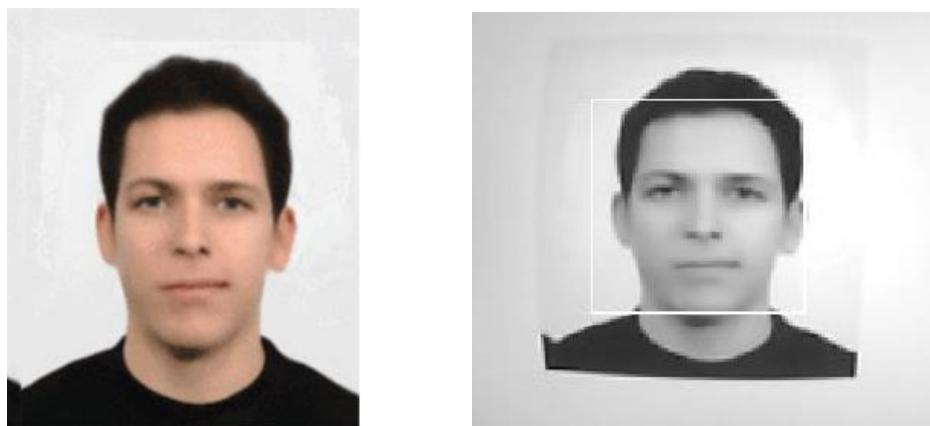


图 5-59 单个人脸检测 原图 (左) 和实验图片 (右)

多个人脸识别:



图 5-60 多个人脸检测原图

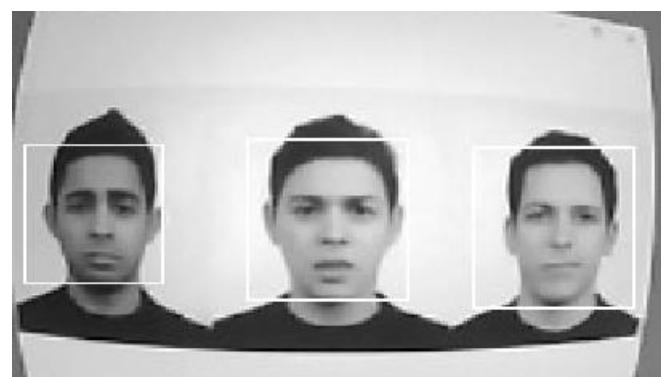


图 5-61 多个人脸检测实验图片

- **总结：**

本节学习了人脸检测，通过 OpenMV4 内置的模型和算法，结合 MicroPython 编程我们轻松完成了实验。而且本实验支持单个和多个人脸同时检测，是机器视觉中非常有代表性的实验。

## 5.5.2 人脸追踪

- **前言:**

人脸追踪，是通过短时间的人脸特征学习，再重新识别的过程。这节内容是基于上一节人脸检测基础上完成的。今天我们来学习一下如何通过 MicroPython 编程快速实现人脸追踪。

- **实验平台:**

pyAI-OpenMV4。



图 5-62 pyAI-OpenMV4

- **实验目的:**

追踪出学习记录下来的人脸。

- **实验讲解:**

本节是人脸检测和特征点识别的相结合，实验用到的函数和对象之前都有介绍过，这里再次列出来让大家重温一下：

## 构造函数

`image.find_features(cascade[, threshold=0.5[, scale=1.5[, roi]]])`

搜索和 Haar Cascade 匹配的所有区域的图像，并返回一个关于这些特征的边界框矩形元组(x, y, w, h)的列表。若未发现任何特征，则返回一个空白列表。

**【cascade】** Haar Cascade 对象

**【threshold】** 是浮点数 (0.0-1.0)，其中较小的值在提高检测速率同时增加误报率。相反，较高的值会降低检测速率，同时降低误报率。;

**【scale】** 是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间；

**【roi】** 指定识别区域的矩形元组(x, y, w, h)。如果未指定，**roi** 即整个图像的图像矩形。

## 使用方法

直接调用该函数。建议使用灰度图像。

表 5-17 寻找人脸特征函数

## 构造函数

`image.find_keypoints([roi[, threshold=20[, normalized=False[, scale_factor=1.5[, max_keypoints=100[, corner_detector=image.CORNER_FAST]]]]]])`

特征点识别函数。返回一个 `image.rect` 矩形对象列表；

**【roi】** 识别区域 (x,y,w,h)，未指定则默认整张图片；

**【threshold】** 控制提取的数量的数字（取值 0-255）。对于默认的 AGAST 角点检测器，该值应在 20 左右。对于 FAST 角点检测器，该值约为 60-80。阈值越低，您提取的角点越多；

**【normalized】** 布尔值。若为 True，在多分辨率下关闭提取键点。若您不关心处理扩展问题，且希望算法运行更快，就将之设置为 True。

**【scale\_factor】** 是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但

其图像匹配相应较差。理想值介于 1.35-1.5 之间。

【max\_keypoints】是一个键点对象所能容纳的键点最大数量。若键点对象过大可能会导致内存问题，需要降低该值。

\*更多参数说明请阅读 OpenMV 官方文档：

文档链接：<http://docs.openmv.io/library/omv.image.html>

### 使用方法

直接调用该函数。（仅支持灰度图像）

表 5-18 特征点识别函数

image 同时包含了特征点对比函数 match\_descriptor，用于比较 2 个特征点的相似程度，也就是判断是否一样，说明如下：

### 构造函数

```
image.match_descriptor(descriptor0, descriptor1[, threshold=70[, filter_outliers=False]])
```

特征点对比函数。返回 kptmatch 对象；

【descriptor0】，【descriptor1】：要对比的 2 个特征点；

【threshold】：控制匹配程度。

### 使用方法

直接调用该函数。

表 5-19 特征点对比函数

从上表可以知道 image.match\_descriptor() 函数返回的是 kptmatch 对象，对象说明如下：

### 构造函数

```
image.kptmatch()
```

特征点对象。由 image.match\_descriptor() 函数返回

使用方法	
kptmatch.rect()	返回特征点边界框。是一个矩形元组 ( $x, y, w, h$ )
kptmatch.cx()	返回特征点中心 $x$ 坐标位置 (int)。也可以通过索引 [0] 取得这个值
kptmatch.cy()	返回特征点中心 $y$ 坐标位置 (int)。也可以通过索引 [1] 取得这个值
*更多参数说明请阅读 OpenMV 官方文档：	
	<a href="http://docs.openmv.io/library/omv.image.html#class-kptmatch-keypoint-object">http://docs.openmv.io/library/omv.image.html#class-kptmatch-keypoint-object</a>

表 5-20 特征点对象

以上函数和对象都是我们之前使用过的，对于人脸追踪实验，具体编程思路如下：

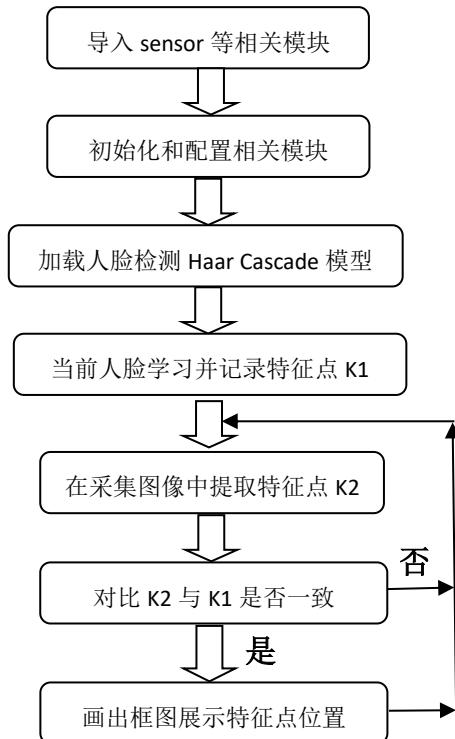


图 5-63 代码编写流程图

官方参考代码路径【示例→Face-Detection→face\_tracking.py】，具体如下：

```
# 人脸追踪例程

#
# 这个例程展示了如何使用关键特征来追踪一个已经使用 Haar Cascade 检测出来的人脸。
# 程序第一阶段先使用 Haar Cascade 找出人脸.然后使用关键特征来学习，最后不停的找这个人脸。
# 关键特征点可以用来追踪任何栋。
#
#翻译: 01Studio

import sensor, time, image

# Reset sensor
sensor.reset()
sensor.set_contrast(3)
sensor.set_gainceiling(16)
sensor.set_framesize(sensor.VGA)
#在 VGA (640*480) 下开个小窗口，相当于数码缩放。
sensor.set_windowing((320, 240))
sensor.set_pixformat(sensor.GRAYSCALE)

# 延时以便摄像头稳定工作
sensor.skip_frames(time = 2000)

# 加载 Haar Cascade 模型
# 默认使用 25 个步骤，减少步骤会加快速度但会影响识别成功率。
face_cascade = image.HaarCascade("frontalface", stages=25)
print(face_cascade)
```

```

# 特征 kpts1

kpts1 = None


# 找到人脸！

while (kpts1 == None):

    img = sensor.snapshot()

    img.draw_string(0, 0, "Looking for a face...")

    # Find faces

    objects = img.find_features(face_cascade, threshold=0.5, scale=1.25)

    if objects:

        # 将 ROI (x,y,w,h) 往各个方向扩展 31 像素

        face = (objects[0][0]-31, objects[0][1]-31, objects[0][2]+31*2,
                 objects[0][3]+31*2)

        # 使用扩展后的 ROI 区域（人脸）学习关键点

        kpts1 = img.find_keypoints(threshold=10, scale_factor=1.1,
                                   max_keypoints=100, roi=face)

        # 用矩形框展示人脸

        img.draw_rectangle(objects[0])



# 打印关键点

print(kpts1)

img.draw_keypoints(kpts1, size=24)

img = sensor.snapshot()

time.sleep(2000) #暂停以便观察特征


# FPS clock

clock = time.clock()


while (True):

    clock.tick()

```

```

img = sensor.snapshot()

# 从图像中提取关键点

kpts2 = img.find_keypoints(threshold=10, scale_factor=1.1,
                           max_keypoints=100, normalized=True)

if (kpts2):

    # 跟关键点 kpts1 匹配

    c=image.match_descriptor(kpts1, kpts2, threshold=85)

    match = c[6] # C[6] 为 matches 值，这个值越大表示匹配程度越高。

    if (match>5): #设置当大于 5 的时候为匹配成功，并画图标示。打印信息。

        img.draw_rectangle(c[2:6])

        img.draw_cross(c[0], c[1], size=10)

        print(kpts2, "matched:%d dt:%d"%(match, c[7]))

    # Draw FPS

    img.draw_string(0, 0, "FPS: %.2f"%(clock.fps()))

```

### ● 实验结果：

运行代码，将摄像头正对自己学习，学习完成后移动摄像头，当再次拍到自己的时候会识别出来。

我们用图片演示一下人脸追踪实验。先学习人脸特征：

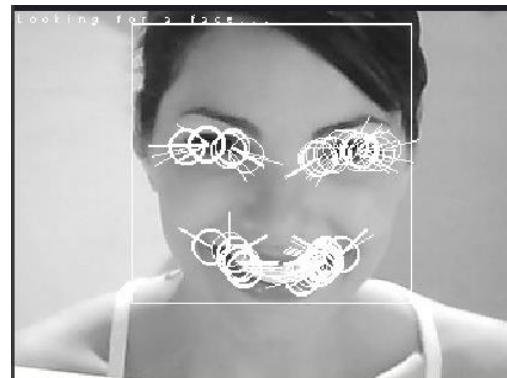
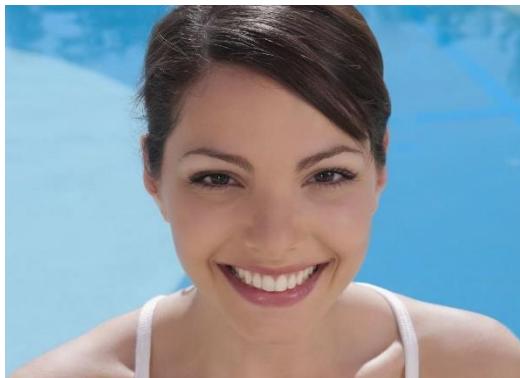


图 5-64 人脸学习图片 原图（左）和特征提取图（右）

学习后保存程序运行，拍摄一下干扰图片，发现没有成功匹配：（有时候由于光线问题错误匹配，但可信度 **match** 值也不高。）



图 5-65 干扰图片（左）和无法匹配（右）

再次拍摄原来的学习图片，发现轻松成功匹配。

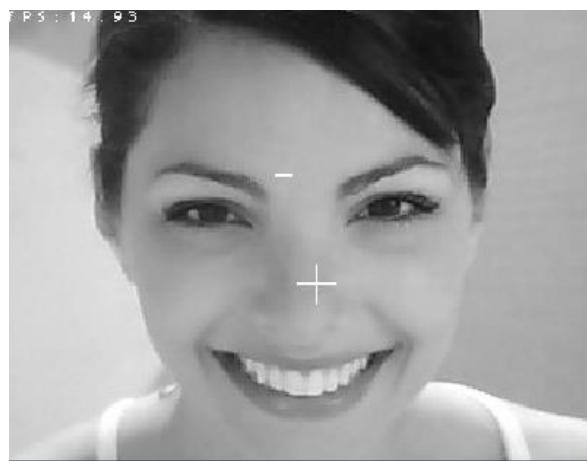


图 5-66 匹配成功

观察串口，发现匹配程度也非常高。**matched** 值大于 10。

```
串行终端 本地连接
{"size":29, "threshold":10, "normalized":1} matched:10 dt:0
 {"size":29, "threshold":10, "normalized":1} matched:14 dt:0
 {"size":25, "threshold":10, "normalized":1} matched:12 dt:0
 {"size":29, "threshold":10, "normalized":1} matched:11 dt:0
 {"size":34, "threshold":10, "normalized":1} matched:12 dt:0
 {"size":30, "threshold":10, "normalized":1} matched:12 dt:0
 {"size":28, "threshold":10, "normalized":1} matched:9 dt:0
 {"size":30, "threshold":10, "normalized":1} matched:13 dt:0
 {"size":29, "threshold":10, "normalized":1} matched:10 dt:0
```

图 5-67

● **总结:**

本节学习了人脸追踪，本质是人脸检测和特征识别的相结合，因此我们可以加以扩展，指定区域学习相关物体特征，然后就可以追踪任何物体了。

## 5.6 眼球追踪

眼球追踪类实验主要是眼球检测和瞳孔特征识别追踪，这是在安防、支付等领域非常流行的应用。全部官方例程位于 Eye-Tracking 例程下，本节挑选重点实验进行讲解。

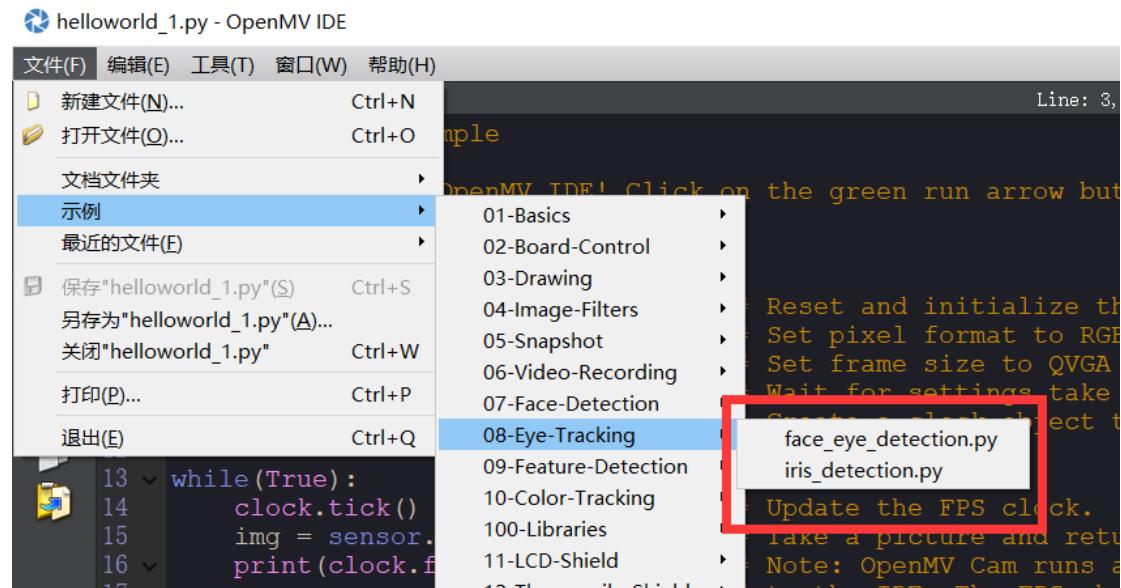


图 5-68 眼球官方例程位置

### 5.6.1 眼球检测

- **前言:**

前面我们学习过人脸检测，这一节我们来学习更细节的检测，那就是眼球的检测。

- **实验平台:**

pyAI-OpenMV4。



图 5-69 pyAI-OpenMV4

- **实验目的:**

将画面中人脸的眼球用检测出来并用矩形框表示。

- **实验讲解:**

眼球检测是建立在人脸检测的基础上的，我们可以先识别人脸，进而再识别人脸中的人眼，提高识别的效率和准确率。跟人脸特征模型一样，OpenMV4 也已经集成了眼球模型以及其处理函数。通过 MicroPython 编程我们可以快速实现人眼检测应用。本实验使用到的函数和对象说明如下：

## 构造函数

```
image.find_features(cascade[, threshold=0.5[, scale=1.5[, roi]]])
```

搜索和 Haar Cascade 匹配的所有区域的图像，并返回一个关于这些特征的边界框矩形元组(x, y, w, h)的列表。若未发现任何特征，则返回一个空白列表。

**【cascade】** Haar Cascade 对象模型。可以是人脸或者眼球。

**【threshold】** 是浮点数 (0.0-1.0)，其中较小的值在提高检测速率同时增加误报率。相反，较高的值会降低检测速率，同时降低误报率。;

**【scale】** 是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间；

**【roi】** 指定识别区域的矩形元组(x, y, w, h)。如果未指定，roi 即整个图像的图像矩形。

## 使用方法

直接调用该函数。建议使用灰度图像。

表 5-21 寻找人脸和眼球特征函数

从上表可以看到人眼检测的模型和算法已经在 OpenMV4 里面内置，我们直接调用即可，具体编程思路如下：

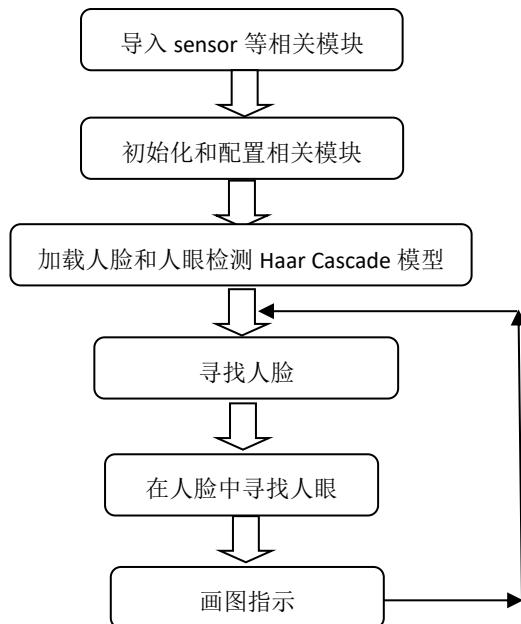


图 5-70 代码编写流程图

官方参考代码路径【示例→Eye-Tracking→face\_eye\_detection.py】，具体如下：

```
# 人眼检测例程

#
# 这个程序是先利用人脸检测确定人的脸位置，然后在检测眼球位置。
# 如果你想检测瞳孔，请参考 iris_detection 例程.

#
#翻译和注释: 01Studio

import sensor, time, image

# Reset sensor
sensor.reset()

# Sensor settings
sensor.set_contrast(1)
sensor.set_gainceiling(16)
sensor.set_framesize(sensor.QVGA)
sensor.set_pixformat(sensor.GRAYSCALE)

# 加载 Haar Cascade 模型
# 默认使用全部 stages，提高识别的准确率。

face_cascade = image.HaarCascade("frontalface", stages=25) #定义人脸模型
eyes_cascade = image.HaarCascade("eye", stages=24) #定义眼球模型
print(face_cascade, eyes_cascade)

# FPS clock
clock = time.clock()

while (True):
    clock.tick()
```

```
# Capture snapshot

img = sensor.snapshot()

# 人脸检测

objects = img.find_features(face_cascade, threshold=0.5,
                             scale_factor=1.5)

# 画出人脸方框

for face in objects:

    img.draw_rectangle(face)

    # 在人脸中寻找眼球.

    # 提示: 使用一个较大的 threshold 值 (识别更多) 和 较小的
    # scale 值 (寻找较小区域的对象)

    eyes = img.find_features(eyes_cascade, threshold=0.5,
                             scale_factor=1.2, roi=face)

    for e in eyes:

        img.draw_rectangle(e)

# Print FPS.

# Note: Actual FPS is higher, streaming the FB makes it slower.

print(clock.fps())
```

### ● 实验结果:

运行代码，将摄像头正对自己脸，可以看到将自己眼睛检测出来。注意光线不能太亮。我们用图片演示一下人眼检测实验。

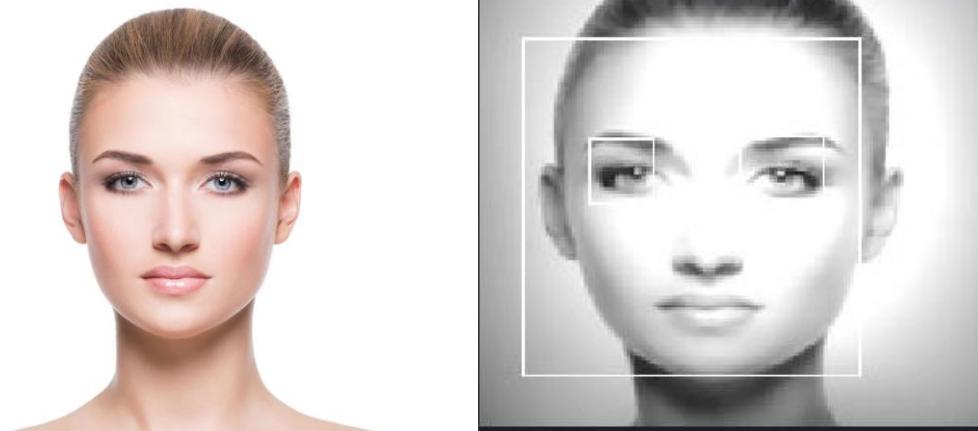


图 5-71 人眼检测 原图 (左) 和实验图片 (右)

● 总结：

通过本节人眼检测实验学习我们发现人眼和人脸检测原理非常相近，只是选择了不用的模型来判断。本质都是特征识别。

## 5.6.2 瞳孔追踪

- 前言：

瞳孔，是动物或人眼睛内虹膜中心的小圆孔，为光线进入眼睛的通道。虹膜上瞳孔括约肌的收缩可以使瞳孔缩小，瞳孔开大肌的收缩使瞳孔散大，瞳孔的开大与缩小控制进入瞳孔的光量。

本节学习瞳孔的追踪。



图 5-72 瞳孔

- 实验平台：

pyAI-OpenMV4。



图 5-73 pyAI-OpenMV4

- 实验目的：

对人眼中的瞳孔进行追踪并画图标示。

### ● 实验讲解：

瞳孔追踪是建立在眼球检测的基础上的，我们识别眼球后，再利用 OpenMV4 瞳孔追踪函数 `find_eyes` 函数来追踪瞳孔即可。通过 MicroPython 编程我们可以快速实现瞳孔追踪的应用。本实验使用到的函数和对象说明如下：

构造函数
<code>image.find_features(cascade[, threshold=0.5[, scale=1.5[, roi]]])</code>
搜索和 Haar Cascade 匹配的所有区域的图像，并返回一个关于这些特征的边界框矩形元组(x, y, w, h)的列表。若未发现任何特征，则返回一个空白列表。
<b>【cascade】</b> Haar Cascade 对象模型。可以是人脸或者眼球。
<b>【threshold】</b> 是浮点数 (0.0-1.0)，其中较小的值在提高检测速率同时增加误报率。相反，较高的值会降低检测速率，同时降低误报率。;
<b>【scale】</b> 是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间；
<b>【roi】</b> 指定识别区域的矩形元组(x, y, w, h)。如果未指定，roi 即整个图像的图像矩形。
使用方法
直接调用该函数。建议使用灰度图像。

表 5-22 寻找人脸和眼球特征函数

构造函数
<code>image.find_eye(roi)</code>
寻找瞳孔。返回中心坐标 (x,y)。
<b>【roi】</b> 指定识别区域的矩形元组(x, y, w, h)。如果未指定，roi 即整个图像的图像矩形。
使用方法
直接调用该函数。建议使用灰度图像。

表 5-23 寻找瞳孔函数

从上表可以看到人眼和瞳孔检测的模型和算法已经在 OpenMV4 里面内置，我们直接调用即可，具体编程思路如下：

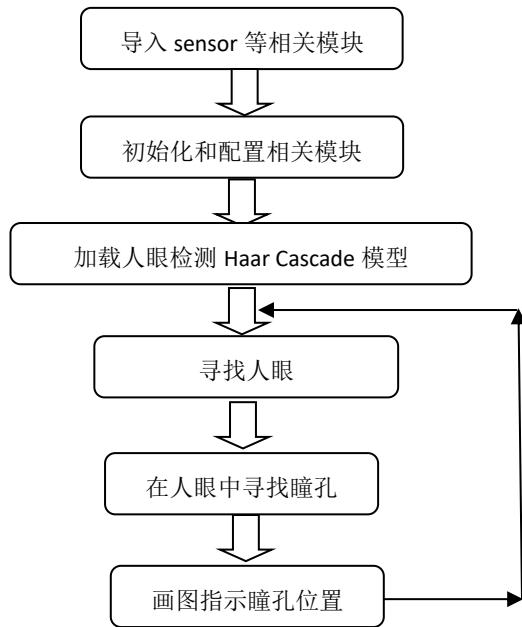


图 5-74 代码编写流程图

官方参考代码路径【示例→Eye-Tracking→iris\_detection.py】，具体如下：

```
# 瞳孔追踪示例

#
#这个例子展示了如何找到眼睛的注视(瞳孔检测)
#图像中的眼睛。此脚本使用 find_eyes 函数来确定
#应该包含瞳孔的感兴趣区域的中心点。它基本上是这样做的
#找到眼睛最黑区域的中心，也就是瞳孔中心。
#
#注意：这个脚本不是先检测人脸，而是用长焦镜头。所以请将摄像头直接对准眼睛。
#
#翻译和注释：01Studio

import sensor, time, image
```

```
# Reset sensor
sensor.reset()

# Sensor settings
sensor.set_contrast(3)
sensor.set_gainceiling(16)

# Set resolution to VGA.
sensor.set_framesize(sensor.VGA)

# 裁剪图像到 200x100, 这提供了更多的细节和更少的数据要处理
sensor.set_windowing((220, 190, 200, 100))

sensor.set_pixformat(sensor.GRAYSCALE)

# 加载 Haar Cascade 模型
# 使用默认参数步骤
eyes_cascade = image.HaarCascade("eye", stages=24)
print(eyes_cascade)

# FPS clock
clock = time.clock()

while (True):
    clock.tick()

    # Capture snapshot
    img = sensor.snapshot()

    # 找眼睛！
    # 提示：使用一个较大的 threshold 值（识别更多）和 较小的
    # scale 值（寻找较小区域的对象）
```

```
eyes = img.find_features(eyes_cascade, threshold=0.5,  
                        scale_factor=1.5)  
  
# 找瞳孔  
  
for e in eyes:  
  
    iris = img.find_iris(e)  
    print(iris)  
  
    img.draw_rectangle(e)  
  
    img.draw_cross(iris[0], iris[1])#瞳孔中心坐标 x,y  
  
# Print FPS.  
  
# Note: Actual FPS is higher, streaming the FB makes it slower.  
print(clock.fps())
```

### ● 实验结果：

运行代码，将摄像头正对自己眼睛，可以看到 OpenMV4 可以将瞳孔位置检测出来，注意光线不能太亮以及眼镜的影响。我们用图片演示一下人眼检测实验。



图 5-75 原图

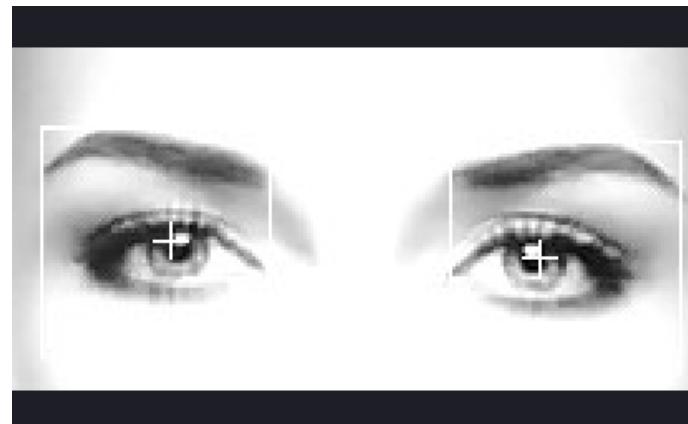


图 5-76 实验图片

● 总结：

有了前面的基础，我们发现瞳孔追踪实验非常容易就完成了。我们转动眼球，OpenMV4 依然能捕捉到瞳孔位置的变化，这适用于需要对眼球瞳孔实时追踪的应用场景。

## 5.7 图片拍摄

拍摄可以说是最基本的功能了，本节内容有三部分，分别是普通拍摄、人脸追踪拍摄和移动物体抓拍。是我们摄像头安防监控常用的功能。图片拍摄官方例程在 Snapshot 中。

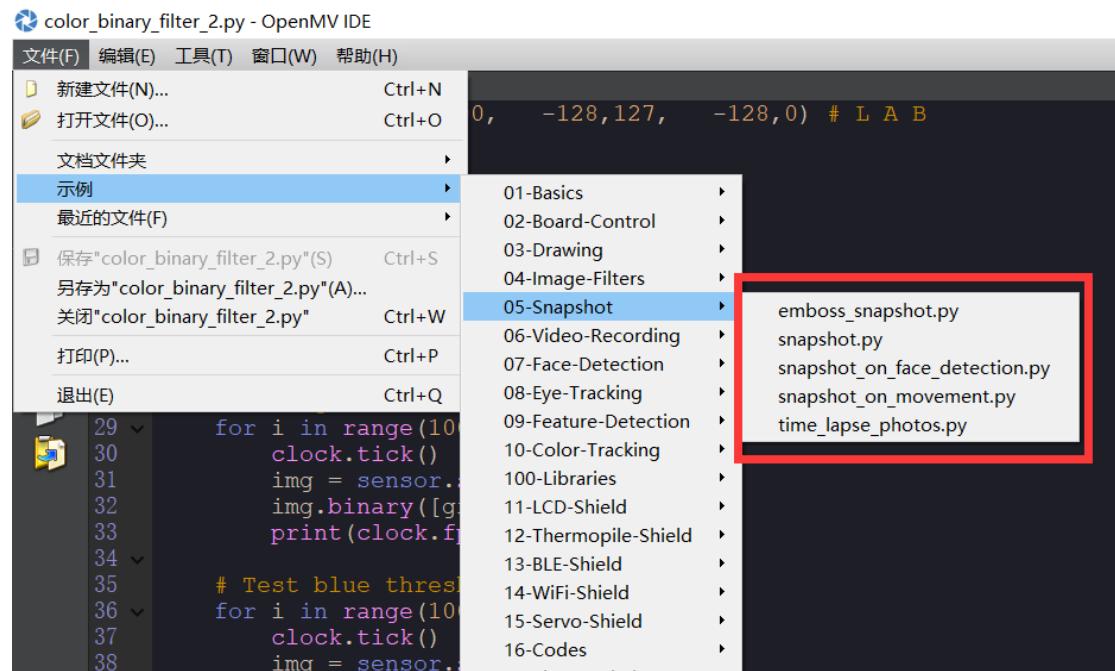


图 5-77 图片拍摄官方例程

### 5.7.1 普通拍摄

- **前言:**

普通拍摄是最基础的功能，OpenMV4 图像算法处理的前提就是实时连续采集多帧图片，然后进行分析。这个功能可以将 OpenMV4 变成一个随身微型照相机。

- **实验平台:**

pyAI-OpenMV4。另外需要配备 SD 卡进行图片储存。



图 5-78 pyAI-OpenMV4

- **实验目的:**

通过编程实现拍照并保存。

- **实验讲解:**

我们在前面摄像头应用章节已经学习过拍摄是使用 `image=sensor.snapshot()` 函数模块，那么我们只需要学会将图片保存即可。保存也是可以直接使用 `image` 下的 `save` 模块，具体如下：

构造函数
<code>img=sensor.snapshot()</code>
通过拍摄创建图像 <code>img</code>

## 使用方法

`image.save(path[, roi[, quality=50]])`

保存图片。

`path`: 保存路径;

`roi`: 指定保存区域(`x, y, w, h`), 默认全图保存;

`quality`: 仅针对 JPEG 格式的质量控制, 有效值为 0-100。

表 5-24 摄像头对象 snapshot

掌握了拍照和保存功能, 我们就可以编程实现了, 官方例程编程代码流程图如下:

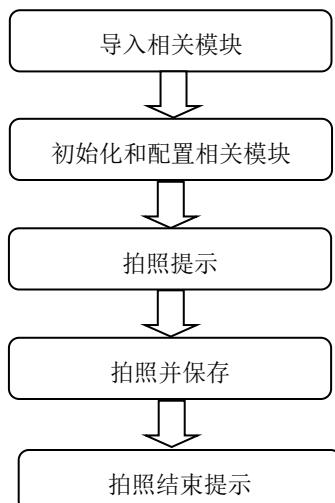


图 5-79 代码编写流程

官方参考代码路径【示例→Snapshot→snapshot.py】，具体如下：

```
# 普通拍照例程  
# 提示：你需要插入 SD 卡来运行这个例程。  
# 你可以使用你的 OpenMV 设备保存图片。
```

```
# 导入相关模块, pyb 用于 LED 控制。
```

```
import sensor, image, pyb
```

```
RED_LED_PIN = 1
BLUE_LED_PIN = 3

#摄像头相关初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.QVGA) # or sensor.QQVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.

#红灯亮提示拍照开始
pyb.LED(RED_LED_PIN).on()
sensor.skip_frames(time = 2000) # 给 2 秒时间用户准备.
pyb.LED(RED_LED_PIN).off()

#蓝灯亮提示正在拍照
pyb.LED(BLUE_LED_PIN).on()
print("You're on camera!")

# 拍摄并保存相关文件，也可以用"example.bmp"或其它文件方式。
sensor.snapshot().save("example.jpg")

pyb.LED(BLUE_LED_PIN).off() #提示拍照完成
print("Done! Reset the camera to see the saved image.")
```

### ● 实验结果：

打开拍照例程或者将上面代码复制到 OpenMV IDE，点击运行。可以看到串口提示完成了一次完整的拍照过程。



图 5-80 实验图片

按下开发板复位键，可以看到 SD 卡的文件系统出现了刚刚拍照保存的 example.jpg 文件。



图 5-81

### ● 总结：

可以看到 OpenMV 通过 micropython 编程可以非常容易使用起来。本节结合基础例程的按键可以打造按键照相机。有兴趣的用户可以自行编程实现。

## 5.7.2 人脸检测拍摄

- **前言:**

上一节普通拍摄是最基础的功能，本节我们将结合前面人脸检测实验，以便实现当摄像头检测到人脸时，拍照记录下来。

- **实验平台:**

pyAI-OpenMV4 开发板。另外需要配备 SD 卡进行图片储存。



图 5-82 pyAI-OpenMV4

- **实验目的:**

通过编程实现人脸检测并拍摄图片记录下来。

- **实验讲解:**

本节可以说是人脸检测和普通拍摄的相结合，人脸检测请参考 [5.5.1 人脸检测](#) 章节内容，这里不再重复。

结合上一节普通拍摄内容，我们的例程编程代码流程图如下：

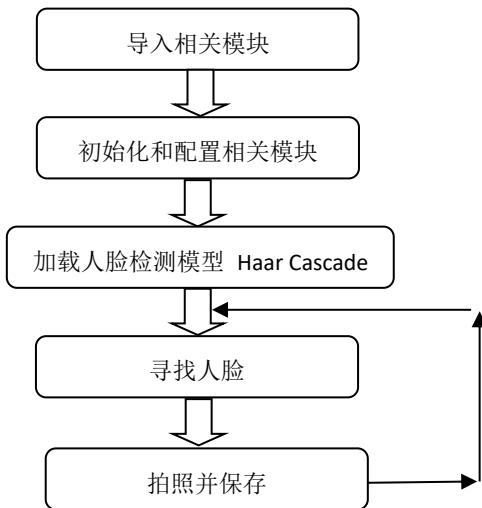


图 5-83 代码编写流程

官方参考代码路径【示例→Snapshot→snapshot\_on\_face\_detection.py】，具体如下：

```

# 人脸检测拍摄影示例

#
# 提示：本实验需要 SD 卡。
#
# 这个例子演示了如何在你的 OpenMV 摄像头上使用人脸跟踪然后拍照
#
#翻译和注释：01Studio


import sensor, image, pyb

RED_LED_PIN = 1
BLUE_LED_PIN = 3

#配置摄像头参数
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.GRAYSCALE)

```

```
sensor.set_framesize(sensor.QVGA) # or sensor.QQVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.

# 加载人脸检测 HaarCascade 模型. 默认使用 stages=25 以提高识别正确率
face_cascade = image.HaarCascade("frontalface", stages=25)

while(True):

    #红灯亮提示开始识别
    pyb.LED(RED_LED_PIN).on()
    print("About to start detecting faces...")
    sensor.skip_frames(time = 2000) # 给时间用户准备.

    #人脸检测中。
    pyb.LED(RED_LED_PIN).off()
    print("Now detecting faces!")
    pyb.LED(BLUE_LED_PIN).on()

    diff = 10 # 成功识别 10 次后确认为识别到人脸.

    while(diff):
        img = sensor.snapshot()

        # Threshold 和 scale 两个参数控制着识别质量和效率。具体看文档说明。
        faces = img.find_features(face_cascade, threshold=0.5,
                                  scale_factor=1.5)

        if faces:
            diff -= 1
            for r in faces:
                img.draw_rectangle(r)
```

```
pyb.LED(BLUE_LED_PIN).off()  
print("Face detected! Saving image...")  
sensor.snapshot().save("snapshot-%d.jpg" % pyb.rng()) # 保存照片.
```

### ● 实验结果：

运行代码，当摄像头捕捉到人脸时就会拍照并保存到 SD 卡：

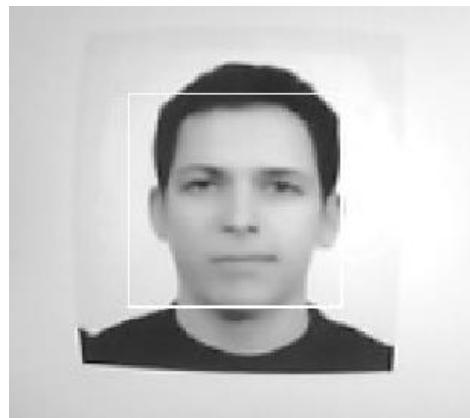


图 5-84 扑捉到图片后开始拍摄

按下开发板 RST 复位键，可以看到 SD 卡的文件系统出现了刚刚拍照保存的图片文件。

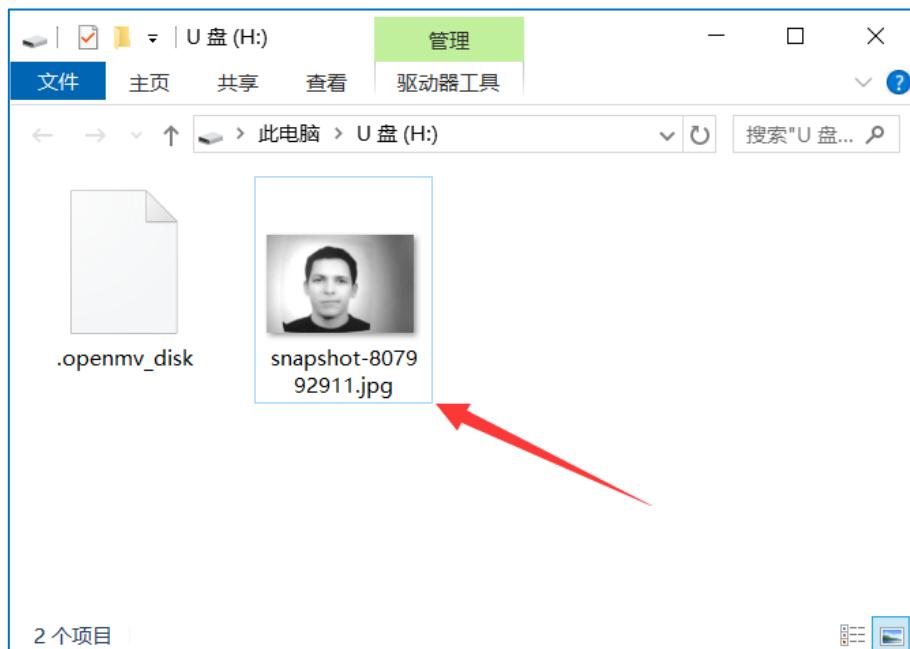


图 5-85 图片储存在 SD 里

● 总结：

本节通过对人脸检测和拍照的整合，实验了人脸检测追踪拍摄，这适用于一些场合的安防应用。

### 5.7.3 移动物体抓拍

- **前言:**

移动物体抓拍，是指当摄像头判断到画面有变化时候（这可能是陌生人闯入或者动物活动）拍摄图片记录。这在安防摄像头上市非常常见的功能。今天我们就编程来实现这个功能。

- **实验平台:**

pyAI-OpenMV4 开发板。另外需要配备 SD 卡进行图片储存。



图 5-86 pyAI-OpenMV4

- **实验目的:**

检测移动物体（画面变化）并拍照保存。

- **实验讲解**

检测移动物体并拍照保存，简单来说就是判断画面的变化，换言之就是判断当前图片和最初记录的图片相比较，看有无变化。一旦发生变化，就判定为画面有东西移动，然后拍照保存。

对比函数 `difference` 在 `image` 下，具体说明如下：

构造函数
<code>image.difference(<i>image</i>[, <i>mask=None</i>])</code>
图像对比函数。将两个图像彼此按像素取绝对值。对于每个颜色通道而言，将每个像素替换为 <code>ABS(this.pixel-image.pixel)</code> 并返回至当前 <code>image</code> 对象。
【 <code>image</code> 】: 需要对比的图像
例：两张图像如果完全一样，则每个像素返回 0，返回的是全黑的图像。
使用方法
直接调用该函数。

表 5-25 图像对比函数

通过上面对比函数后 `image` 的信息会改变，可以检测 `image` 状态来判断图片是否相同。

构造函数
<code>stats=image.statistics()</code>
统计数据对象。 灰度统计数据有一个通道，使用非 <code>l_*</code> 、 <code>a_*</code> 或 <code>b_*</code> 方法； RGB565 百分比值有三个通道。使用 <code>l_*</code> 、 <code>a_*</code> 和 <code>b_*</code> 方法。
使用方法
<code>stats.mean()</code>
返回灰度均值（0-255）整型。
<code>stats.median()</code>
返回灰度中值（0-255）整型。
<code>stats.mode()</code>
返回灰度众值（0-255）整型。
<code>stats.min()</code>
返回灰度最小值（0-255）整型。
<code>stats.max()</code>
返回灰度最大值（0-255）整型。

以上说明均以灰度为例。

\*更多使用说明请阅读 OpenMV 官方文档：

<https://docs.singtown.com/micropython/zh/latest/openmvcam/library/omv.image.html#class-statistics>

表 5-26 数据统计对象

实验原理就是先拍下当前画面并保存，然后摄像头不停的采集实时画面，与保存的图片对比，有差异的话即执行拍摄功能，代码编写流程如下：

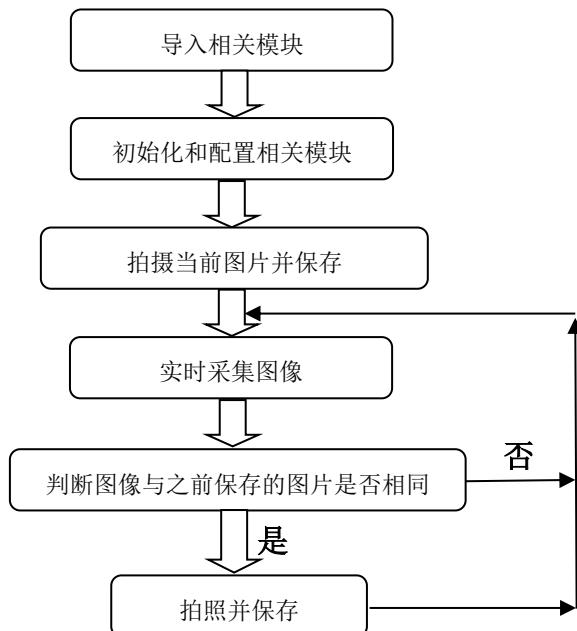


图 5-87 代码编写流程图

官方参考代码路径【示例→Snapshot→snapshot\_on\_movement.py】，具体如下：

```
# 移动物体抓拍示例
#
# 提示：本实验需要插入 SD 卡。
#
```

```
# 本示例是通过帧差异来检测移动物体。检测到移动物体后拍摄图片并保存。  
#  
#翻译和注释: 01Studio  
  
import sensor, image, pyb, os  
  
RED_LED_PIN = 1  
BLUE_LED_PIN = 3  
  
#摄像头初始化  
sensor.reset() # Initialize the camera sensor.  
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE  
sensor.set_framesize(sensor.QVGA) # or sensor.QQVGA (or others)  
sensor.skip_frames(time = 2000) # Let new settings take affect.  
sensor.set_auto_whitebal(False) # Turn off white balance.  
  
if not "temp" in os.listdir(): os.mkdir("temp") # 建立 temp 文件夹  
  
while(True):  
  
    pyb.LED(RED_LED_PIN).on()  
    print("About to save background image...")  
    sensor.skip_frames(time = 2000) # 给一定时间用户做测试准备。  
  
    pyb.LED(RED_LED_PIN).off()  
    sensor.snapshot().save("temp/bg.bmp")  
    print("Saved background image - Now detecting motion!")  
    pyb.LED(BLUE_LED_PIN).on()  
  
    diff = 10 # 连续测试 10 帧，减少误判。
```

```
while(diff):  
    img = sensor.snapshot()  
    #和开始拍摄图片对比，不变时界面全黑色，返回图像对象。  
    img.difference("temp/bg.bmp")  
  
    stats = img.statistics()  
    # 下面 Stats[5] 返回 RGB5656 LAB 中 L 的最大值(0-255) (int)，大于 20  
    # 判定为图片变化，即物体移动。  
    if (stats[5] > 20):  
        diff -= 1  
  
    pyb.LED(BLUE_LED_PIN).off()  
    print("Movement detected! Saving image...")  
    sensor.snapshot().save("temp/snapshot-%d.jpg" % pyb.rng())  
    # 保存图片。
```

### ● 实验结果：

运行代码，可以看到过一会 IDE 的帧缓冲区是黑色。这是因为图像没变化，对比函数处理好图像的值变成 0（黑色）。

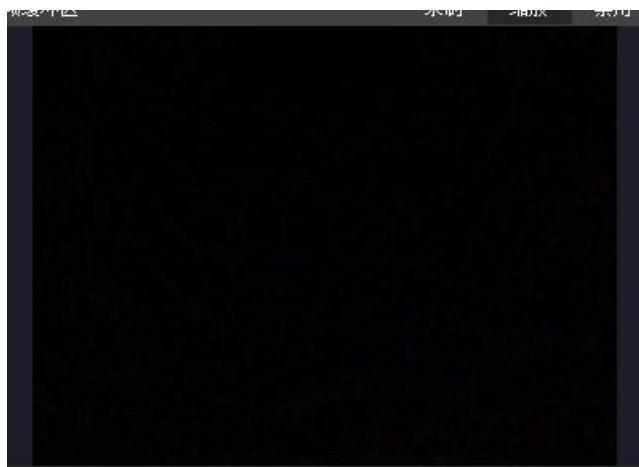


图 5-88 图像界面没变化时候显示情况

我们将手指伸到摄像头前，可以看到摄像头检测到变化并提示拍照保存。



图 5-89 让画面发生变化

按下开发板复位，打开 U 盘（SD 卡），看到 temp 文件夹下保存里面相关照片文件。bg.bmp 为上电首次拍摄图片，snapshot-xxx.jpg 为移动界面抓拍图片。

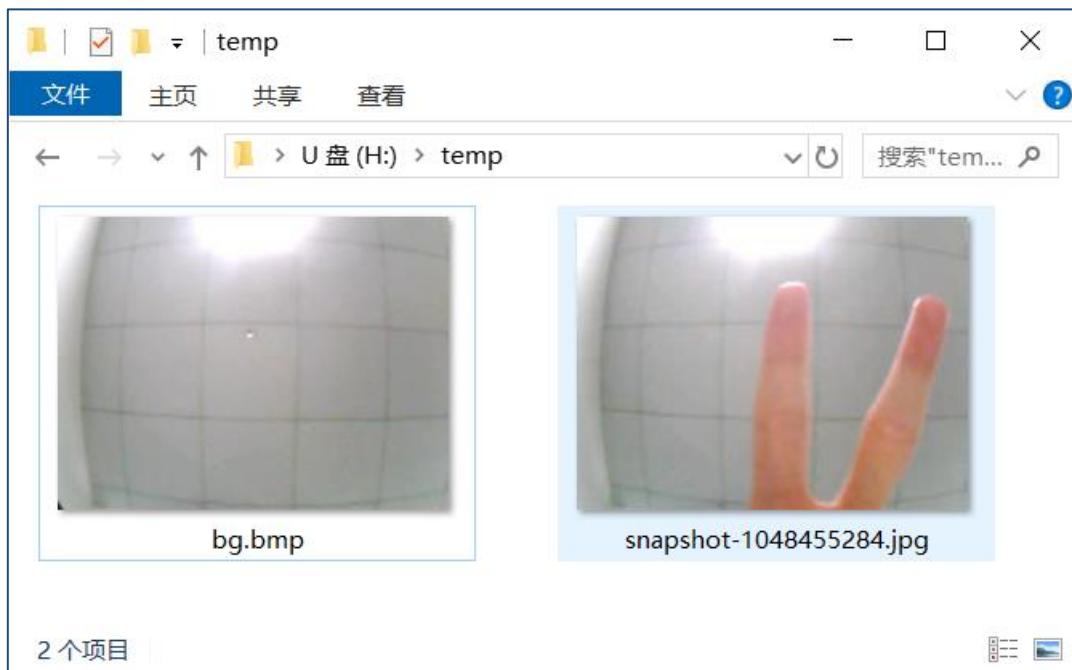


图 5-90 U 盘（SD 卡）保存的文件

### ● 总结：

本节通过图像对比函数，轻松实现图像界面变化识别。从而实现简易的移动物体抓拍功能。

## 5.8 视频录制

视频录制的学习方式跟图片拍摄有点类似，本节内容有三部分，分别是普通视频拍摄、人脸追踪视频拍摄和移动物体视频抓拍。也是我们摄像头安防监控常用的功能。图片拍摄官方例程在 Video-Recording 中。

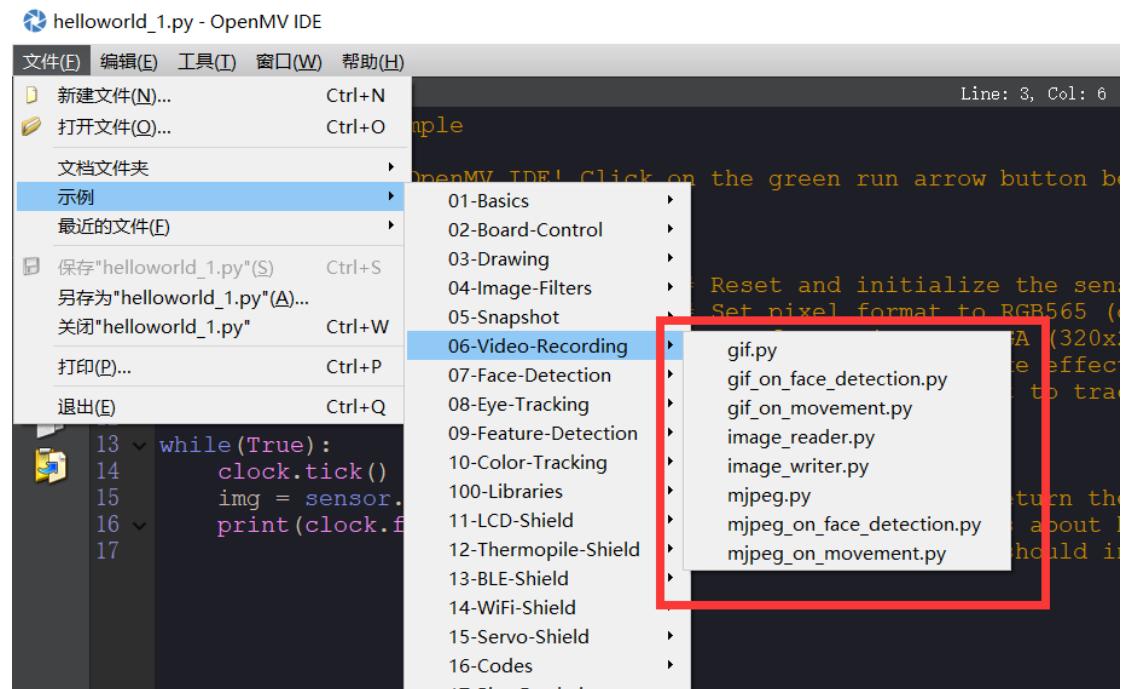


图 5-91 视频拍摄官方例程

### 5.8.1 普通 GIF 视频录制

- **前言:**

GIF 是非常流行的动图文件，我们平时在网上看到的表情包或短视频很多都是使用 GIP 格式。今天我们就来学习一下使用 OpenMV4 来录制视频。

- **实验平台:**

pyAI-OpenMV4。另外需要配备 SD 卡进行储存。



图 5-92 pyAI-OpenMV4

- **实验目的:**

录制 GIF 视频。

- **实验讲解:**

OpenMV4 集成了视频采集函数，也是使用拍摄功能，原理是将一张张拍摄的图片添加一起组合成动图。我们来看看 GIF 的对象：

构造函数
------

gif.Gif(filename, width=Auto, height=Auto, color=Auto, loop=True)
---

创建一个 Gif 对象。

【filename】保存的路径和文字；

【width】Auto 为图像传感器水平分辨率（除非显式覆盖）；

**【height】** Auto 为图像传感器垂直分辨率（除非显式覆盖）；

**【color】** Auto 为图像传感器颜色模式（除非显式覆盖）；

**【loop】** True 表示自动循环播放

## 使用方法

**gif.width()**

返回 gif 对象的宽度（水平分辨率）。

**gif.height()**

返回 gif 对象的高度（垂直分辨率）。

**gif.add\_frame(image, delay=10)**

将一张图像添加到 gif 记录中。图像的宽度、高度和颜色模式必须与 gif 构造函数中使用的宽度、高度和颜色模式相同。

**【image】** 添加的图像

**【delay】** 是前一帧结束后等待播放此帧的毫秒数， 默认是 10。

\*更多使用说明请阅读 OpenMV 官方文档：

<https://docs.singtown.com/micropython/zh/latest/openmvcam/library/omv.gif.html#id3>

表 5-27 GIF 对象

学习了 gif 的相关用法后，我们整理思路，代码编写流程如下：

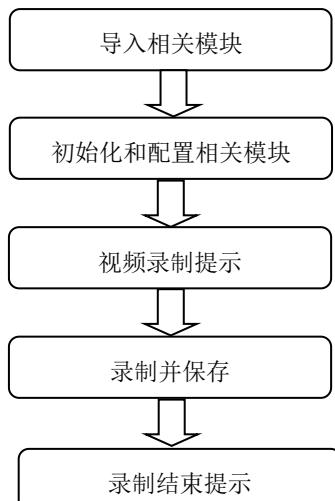


图 5-93 代码编写流程

官方参考代码路径【示例→Video Recording→gif.py】，具体如下：

```
# GIF 视频录制示例

#
# 提示：你需要插入 SD 卡来运行此程序.

#
# 你可以使用 OpenMV 来录制 gif 文件. 可以选择性录制 RGB565 或灰度图像.

# 使用像 GIMP 这样的照片编辑软件来压缩和优化 Gif, 然后再上传到 web.

#
#翻译和注释: 01Studio

import sensor, image, time, gif, pyb

RED_LED_PIN = 1
BLUE_LED_PIN = 3

#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.QQVGA) # or sensor.QVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.
clock = time.clock() # Tracks FPS.

pyb.LED(RED_LED_PIN).on()
sensor.skip_frames(time = 2000) # Give the user time to get ready.

pyb.LED(RED_LED_PIN).off()
pyb.LED(BLUE_LED_PIN).on()

g = gif.Gif("example.gif", loop=True)
```

```
print("You're on camera!")

for i in range(100):
    clock.tick()
    # clock.avg() 返回每帧平均运行时间,单位 ms;
    # gif 的延时是厘秒级别 (百分之一秒)
    g.add_frame(sensor.snapshot(), delay=int(clock.avg()/10))
    print(clock.fps())

g.close()

pyb.LED(BLUE_LED_PIN).off()

print("Done! Reset the camera to see the saved recording.")
```

### ● 实验结果：

运行代码，将摄像头对准要拍摄的物体，点击运行。

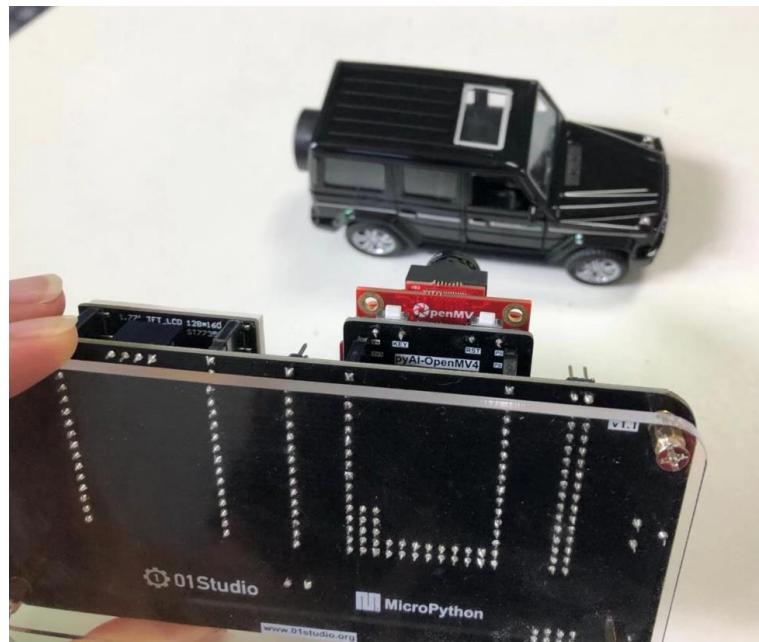


图 5-94 对准拍摄物体



图 5-95 拍摄效果

拍摄完成后，按下开发板复位键，可以看到 U 盘（SD 卡）上出现了刚刚拍摄的 gif 文件，文件名为：“example.gif”。点击打开就可以看到录制的视频。



图 5-96 gif 视频文件

### ● 总结：

本节学习了视频录制，结合基础例程的按键可以打造按键录像机。有兴趣的用户可以自行编程实现。

## 5.8.2 人脸检测 GIF 视频录制

- **前言：**

上一节普通拍摄是最基础的功能，本节我们将结合前面人脸检测实验，以便实现当摄像头检测到人脸时，拍摄视频记录下来。

- **实验平台：**

pyAI-OpenMV4。另外需要配备 SD 卡进行图片储存。



图 5-97 pyAI-OpenMV4

- **实验目的：**

通过编程实现人脸检测并拍摄视频记录下来。

- **实验讲解：**

本节可以说是人脸检测和普通 GIF 视频拍摄的相结合，人脸检测请参考 [5.5.1 人脸检测](#) 章节内容，这里不再重复。

结合上一节普通 GIF 视频拍摄内容，我们的例程编程代码流程图如下：

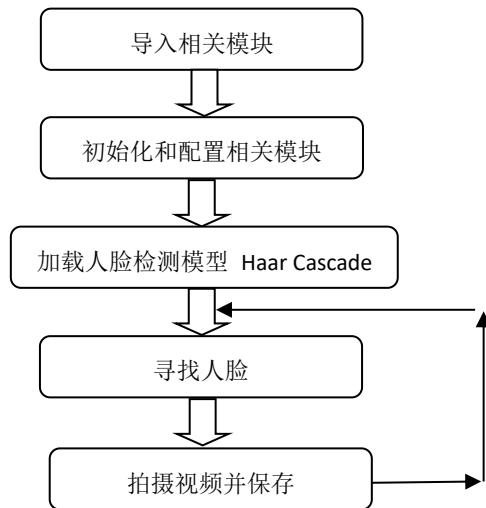


图 5-98 代码编写流程

官方参考代码路径【示例→Video Recording→`gif_on_face_detection.py`】，具体如下：

```

# 人脸检测 GIF 视频录制例程

#
# 提示：此程序需要使用 SD 卡。
#
# 你可以使用 OpenMV 来录制 gif 文件。可以选择性录制 RGB565 或灰度图像。
# 使用像 GIMP 这样的照片编辑软件来压缩和优化 Gif，然后再上传到 web。
#
# 这个例程是实现当检测到人脸时候自动录制 GIF 视频
#
# 翻译和注释：01Studio

import sensor, image, time, gif, pyb

RED_LED_PIN = 1
BLUE_LED_PIN = 3

```

```
#摄像头初始化

sensor.reset() # Initialize the camera sensor.

sensor.set_pixformat(sensor.GRAYSCALE) # or sensor.

sensor.set_framesize(sensor.QQVGA) # or sensor.HQVGA (or others)

sensor.skip_frames(time = 2000) # Let new settings take affect.

# 人脸检测是通过在图像上使用 Haar Cascade 特征检测器来实现的。

# 一个 Haar Cascade 是一系列简单区域的对比检查. 人脸识别有 25 个阶段，每个阶段#
# 有几百次检测。Haar Cascades 运行很快因为是逐个阶段递进检测的。

# 此外，OpenMV Cam 使用一种称为积分图像的数据结构来在恒定时间内快速执行每个区
# 域的对比度检查(特征检测需要用灰度图像的原因是因为图像积分需要更多空间)。

face_cascade = image.HaarCascade("frontalface", stages=25)

while(True):

    pyb.LED(RED_LED_PIN).on()

    print("About to start detecting faces...")

    sensor.skip_frames(time = 2000) # Give the user time to get ready.

    pyb.LED(RED_LED_PIN).off()

    print("Now detecting faces!")

    pyb.LED(BLUE_LED_PIN).on()

    diff = 10 # 重复检测 10 帧，确认人脸.

    while(diff):

        img = sensor.snapshot()

        # threshold 和 scale_factor 两个参数控制着识别的效率和准确性.

        faces = img.find_features(face_cascade, threshold=0.5,
scale_factor=1.5)
```

```
if faces:

    diff -= 1

    for r in faces:

        img.draw_rectangle(r)

# 创建 GIF 对象, pyb.rng()随机返回一个 30 位的二进制数值,
# 最终文件名转换成 10 进制。

g = gif.Gif("example-%d.gif" % pyb.rng(), loop=True)

clock = time.clock() # Tracks FPS.

print("You're on camera!")

for i in range(100):

    clock.tick()

    # clock.avg() 返回每帧平均运行时间, 单位 ms ;

    # gif 的延时是厘秒级别 (百分之一秒)

    g.add_frame(sensor.snapshot(), delay=int(clock.avg()/10))

    print(clock.fps())



g.close()

pyb.LED(BLUE_LED_PIN).off()

print("Restarting...")
```

### ● 实验结果:

在 IDE 中运行代码, 将摄像头对准人脸, 检测到后开始拍摄 GIF 视频:

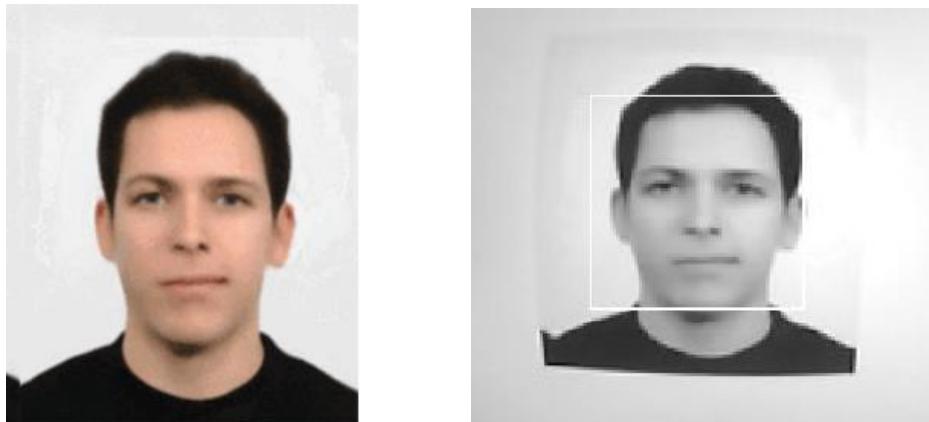


图 5-99 单个人脸检测 原图 (左) 和实验图片 (右)

拍摄完成后，按下开发板复位键，可以看到 U 盘 (SD 卡) 上出现了刚刚拍摄的 gif 文件，文件名为：“example-xxx.gif”。点击打开就可以看到刚刚录制的视频。



图 5-100 GIF 视频文件

### ● 总结：

本节通过对人脸检测和拍摄视频的整合，实验了人脸检测追踪拍摄视频，这适用于一些场合的安防应用。

### 5.8.3 移动物体 GIF 视频录制

- **前言：**

移动物体视频录制，是指当摄像头判断到画面有变化时候（这可能是陌生人闯入或者动物活动）拍摄一段小视频记录。这在安防摄像头上市非常常见的功能。今天我们就编程来实现这个功能。

- **实验平台：**

pyAI-OpenMV4。另外需要配备 SD 卡进行 GIF 视频文件储存。



图 5-101 pyAI-OpenMV4

- **实验目的：**

检测移动物体（画面变化）并拍摄视频保存。

- **实验讲解**

检测移动物体并拍摄视频保存，简单来说就是判断画面的变化，换言之就是判断当前图片和最初记录的图片相比较，看有无变化。一旦发生变化，就判定为画面有东西移动，然后拍摄视频保存。

本节内容可以说是前面学习过的移动物体抓拍和 GIF 视频拍摄内容相结合。移动问题判断可以在 [5.7.3 移动物体抓拍](#) 章节内容找到，这里不在重复。

结合通过 GIF 视频拍摄内容，我们的例程编程代码流程图如下：

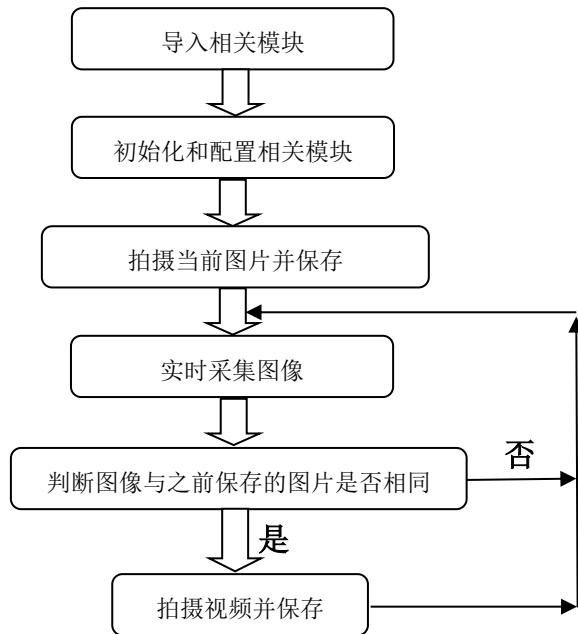


图 5-102 代码编写流程图

官方参考代码路径【示例→Video Recording→`gif_on_movement.py`】，具体如下：

```
# 移动物体 GIF 视频录制例程
#
# 提示：你需要插入 SD 卡来运行此程序。
#
# 你可以使用 OpenMV 来录制 gif 文件。可以选择性录制 RGB565 或灰度图像。
# 使用像 GIMP 这样的照片编辑软件来压缩和优化 Gif，然后再上传到 web。
#
# 本示例是通过帧差异来检测移动物体。检测到移动物体后拍摄视频并保存。
#
# 翻译和注释：01Studio

import sensor, image, time, gif, pyb, os
```

```

RED_LED_PIN = 1
BLUE_LED_PIN = 3

#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.QQVGA) # or sensor.QVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.
sensor.set_auto_whitebal(False) # Turn off white balance.

if not "temp" in os.listdir(): os.mkdir("temp") # 新建一个 temp 文件夹存放
视频

while(True):

    pyb.LED(RED_LED_PIN).on()
    print("About to save background image...")
    sensor.skip_frames(time = 2000) # Give the user time to get ready.

    pyb.LED(RED_LED_PIN).off()
    sensor.snapshot().save("temp/bg.bmp")
    print("Saved background image - Now detecting motion!")
    pyb.LED(BLUE_LED_PIN).on()

    diff = 10 # 连续测试 10 帧，减少误判。
    while(diff):
        img = sensor.snapshot()
        img.difference("temp/bg.bmp")
        stats = img.statistics()

```

```
#下面 Stats[5] 返回 RGB5656 LAB 中 L 的最大值(0-255) (int), 大于 20 判定为图片变化, 即物体移动.

if (stats[5] > 20):
    diff -= 1

g = gif.Gif("example-%d.gif" % pyb.rng(), loop=True)

clock = time.clock() # Tracks FPS.
print("You're on camera!")

for i in range(100):
    clock.tick()
    # clock.avg() returns the milliseconds between frames - gif
delay is in
    g.add_frame(sensor.snapshot(), delay=int(clock.avg()/10)) #
centiseconds.

    print(clock.fps())

g.close()

pyb.LED(BLUE_LED_PIN).off()
print("Restarting...")
```

### ● 实验结果:

运行代码，可以看到过一会 IDE 的帧缓冲区是黑色。这是因为图像没变化，对比函数处理好图像的值变成 0 (黑色)。

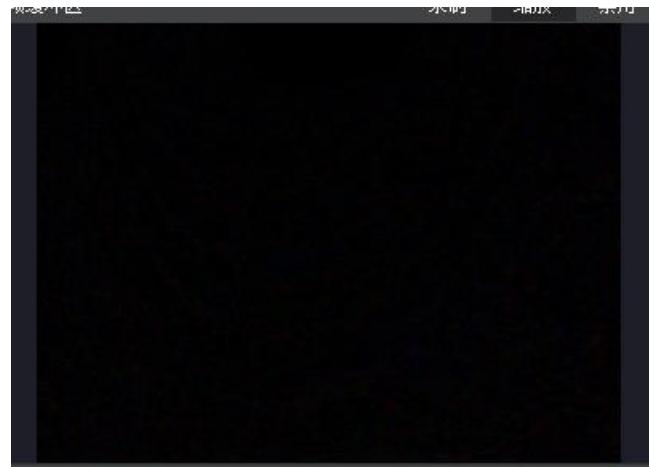


图 5-103 图像界面没变化时候显示情况

我们将手指伸到摄像头前，可以看到摄像头检测到变化并提示拍摄视频。



图 5-104 让画面发生变化

按下开发板复位，打开 U 盘 (SD 卡)，看到 temp 文件夹下保存的 bg.bmp 为上电首次拍摄图片。

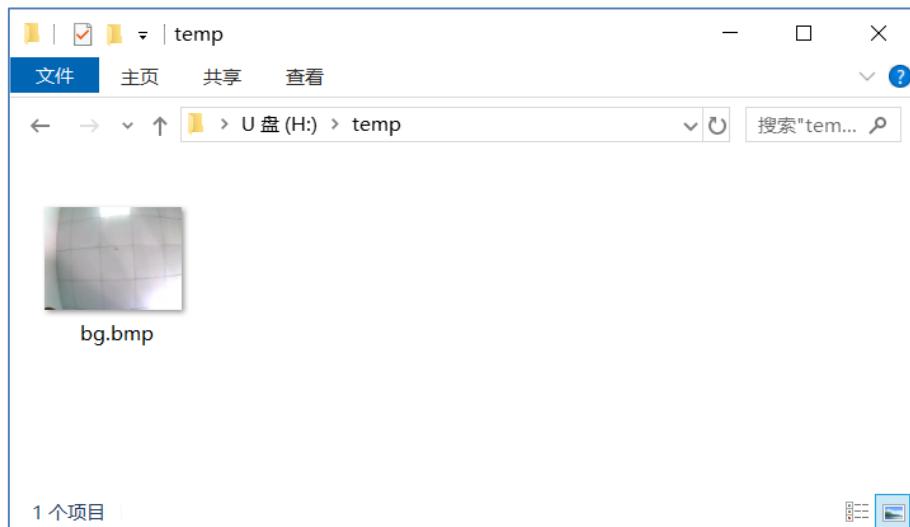


图 5-105 U 盘 (SD 卡) temp 目录保存的图片文件

根目录里面保存相关 GIF 视频文件。

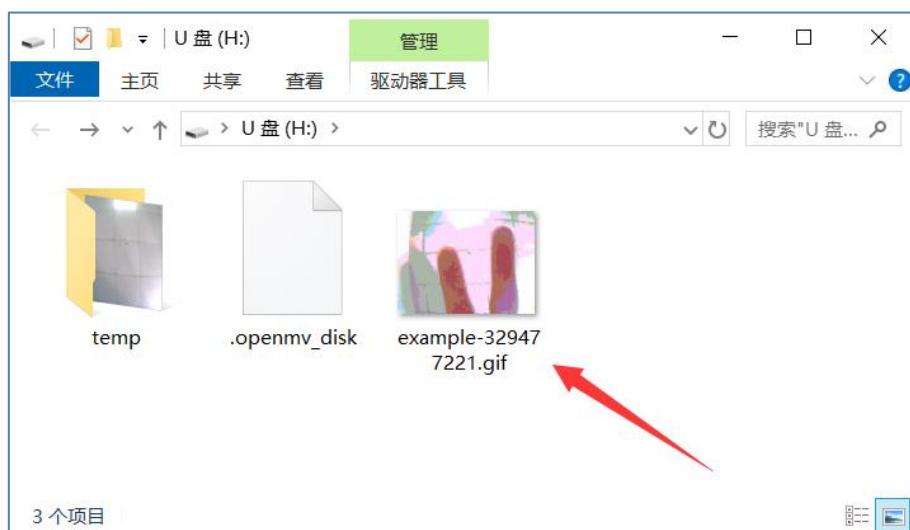


图 5-106 GIF 视频文件在 U 盘根目录下

### ● 总结：

本节通过图像对比函数，轻松实现图像界面变化识别。从而实现简易的移动物体抓拍视频功能。

## 5.9 图像滤波

图片滤波处理可以说是整个机器视觉应用的基础，当我们通过摄像头取得拍摄图片时候，通过特定方式的滤波可以让图片去掉无相关信息，让后期图像处理变得更容易。日常生活中的拍照美颜，以及前面的机器视觉实验大多都运用了图像滤波算法。

OpenMV ID 集成了非常多的图像滤波例程，在 `image-filters` 中。我们挑选常用的例程来学习，以便了解其使用方法。

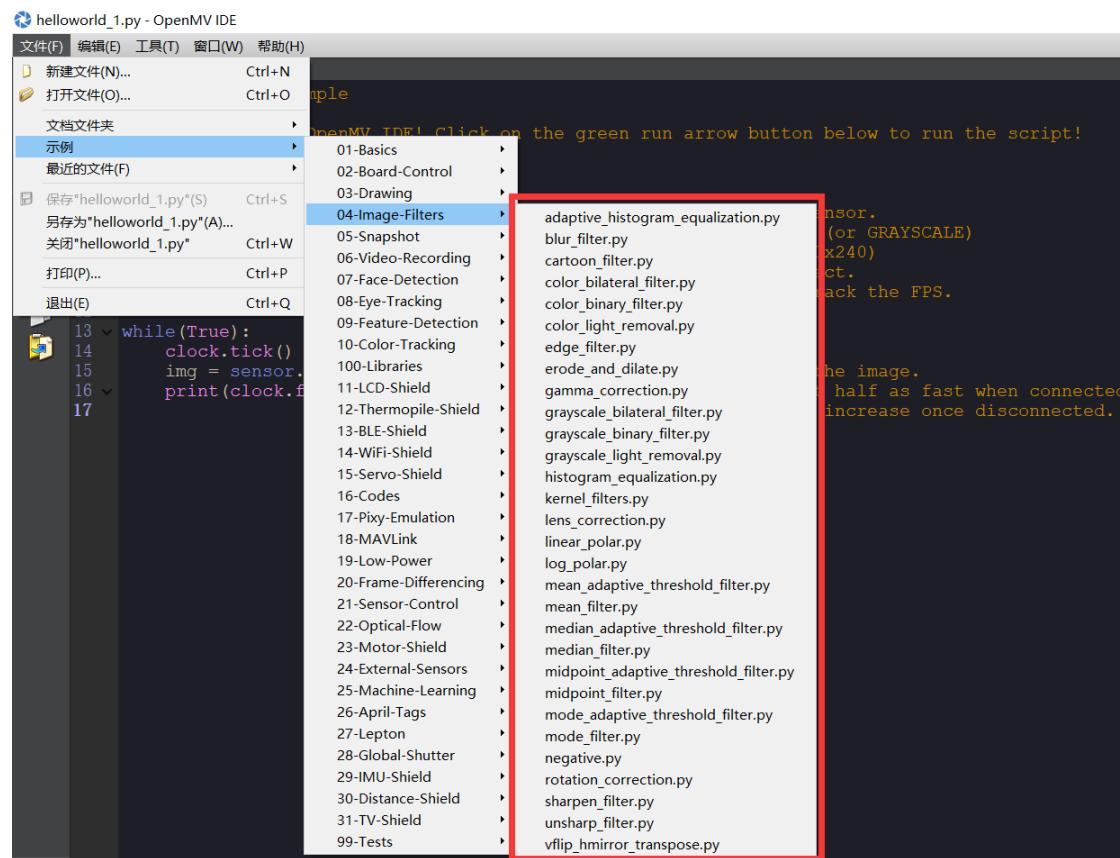


图 5-107 图像滤波相关例程

### 5.9.1 颜色二值化滤波

- **前言:**

颜色二值化是指将彩色图像黑白图像，而凸显出图像中某种颜色，比如我们想突出一幅图像中的红色，我们可以通过二值化将图像中红色部分变成白色，然后其余部分变成黑色。变化后的图像文件将变得非常小，以提高后期图片处理的速度。

- **实验平台:**

pyAI-OpenMV4。另外需要配备 SD 卡进行储存。



图 5-108 pyAI-OpenMV4

- **实验目的:**

将图像中特定颜色的物体进行二值化滤波。

- **实验讲解:**

图像二值化我们在特征检测章节学习过，这里重温一下其对象说明：

构造函数
<code>image.binary(thresholds[, invert=False[, zero=False[, mask=None[, to_bitmap=False[, copy=False]]]]])</code>

```
image.binary(thresholds[, invert=False[, zero=False[, mask=None[,  
to_bitmap=False[, copy=False]]]]])
```

图像二值（黑白）化。

【thresholds】必须是元组列表。 (lo, hi) 定义你想追踪的颜色范围。对于灰度图像，每个元组需要包含两个值：最小灰度值和最大灰度值。

例： thresholds=(0,100) ，则该函数表示将(0,100)灰度值范围变成白色。

【invert】如果为 True，则 0 1（黑 白）反转；默认 False 不反转。

\*更多参数说明请阅读 OpenMV 官方文档：

文档链接：<http://docs.openmv.io/library/omv.image.html>

## 使用方法

直接调用该函数。

表 5-28 图像二值化应用

二值化的函数非常简单，我们定义好阈值即可，代码编写流程如下：

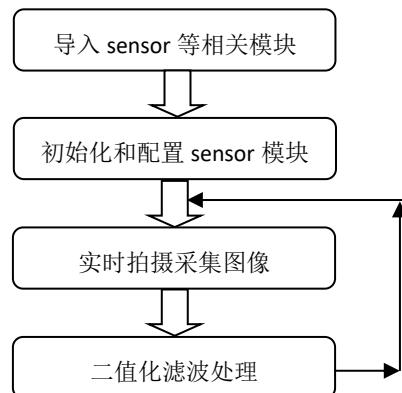


图 5-109 代码编写流程图

官方参考代码路径【示例→Image Filters→Color\_Binary\_Filter.py】，具体如下：

为了方便测试，我们将蓝色和绿色的二值化用'''注释掉，所以以下代码仅进行红色和非红色两个情况的二值化滤波。

```
# 颜色二值化滤波示例
#
# 这个程序展示了图像二值化滤波。你可以自定义任何阈值来分割不同颜色的图像。
#
```

```
#翻译和注释: 01Studio

import sensor, image, time

sensor.reset()
sensor.set_framesize(sensor.QVGA)
sensor.set_pixformat(sensor.RGB565)
sensor.skip_frames(time = 2000)
clock = time.clock()

# 可以使用 工具-> 机器视觉 -> 阈值编辑器 来调整阈值.

red_threshold = (0,100,    0,127,    0,127) # L A B
green_threshold = (0,100,   -128,0,    0,127) # L A B
blue_threshold = (0,100,   -128,127,   -128,0) # L A B

while(True):

    # Test red threshold
    for i in range(100):
        clock.tick()
        img = sensor.snapshot()
        img.binary([red_threshold])
        print(clock.fps())

    ...

    # Test green threshold
    for i in range(100):
        clock.tick()
        img = sensor.snapshot()
        img.binary([green_threshold])
```

```
print(clock.fps())


# Test blue threshold

for i in range(100):

    clock.tick()

    img = sensor.snapshot()

    img.binary([blue_threshold])

    print(clock.fps())

    ...

# Test not red threshold

for i in range(100):

    clock.tick()

    img = sensor.snapshot()

    img.binary([red_threshold], invert = 1)

    print(clock.fps())

    ...

# Test not green threshold

for i in range(100):

    clock.tick()

    img = sensor.snapshot()

    img.binary([green_threshold], invert = 1)

    print(clock.fps())


# Test not blue threshold

for i in range(100):

    clock.tick()

    img = sensor.snapshot()

    img.binary([blue_threshold], invert = 1)

    print(clock.fps())

    ...
```

### ● 实验结果：

运行代码，我们将红色物体放在摄像头前，可以看到程序对图像进行红色和非红色二值化滤波（不断循环）。

对红色进行二值化滤波，红色物体会用白色显示：



图 5-110 红色二值化滤波 原图 (左) 和实验图片 (右)

对红色进行二值化滤波，红色物体会用黑色显示（跟上面红色二值化滤波刚好相反）：

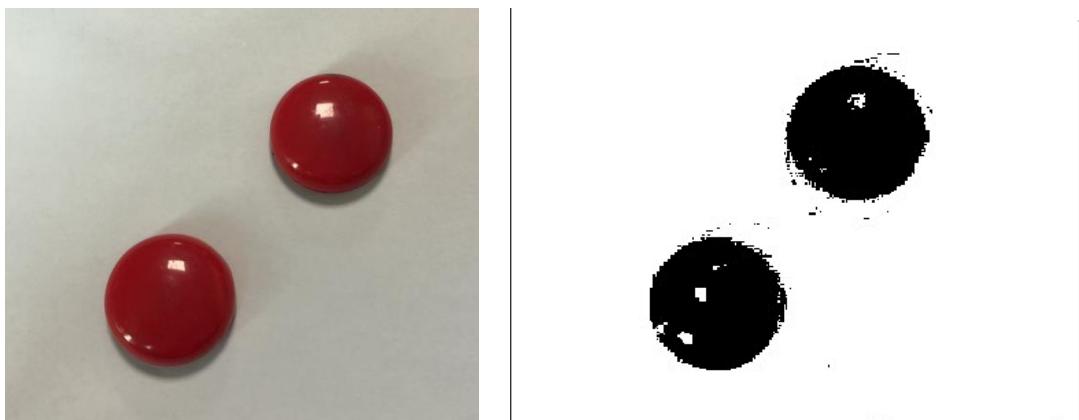


图 5-111 非红色二值化滤波 原图 (左) 和实验图片 (右)

### ● 总结：

本节学习了图像二值化滤波处理，这是应用非常广泛的滤波示例，特别适用于指定颜色追踪和处理。因为滤波后的图像变得更小，特征更明显，方便处理。

## 5.9.2 锐化滤波

- **前言:**

图像锐化(*image sharpening*)是补偿图像的轮廓，增强图像的边缘及灰度跳变的部分，使图像变得清晰，分为空间域处理和频域处理两类。图像锐化是为了突出图像上地物的边缘、轮廓，或某些线性目标要素的特征。这种滤波方法提高了地物边缘与周围像元之间的反差，因此也被称为边缘增强。

简答来说锐化就是为了让图像变得更清晰。

- **实验平台:**

pyAI-OpenMV4 开发板。



图 5-112 pyAI-OpenMV4

- **实验目的:**

编程实现将拍摄到的图像进行锐化处理。

- **实验讲解:**

本节实验是通过使用拉普拉斯滤波器来锐化图像。当然和以往一样，我们不需要再重复造轮子，OpenMV4 已经集成了 `laplacian` 函数算法。我们学会对象构建和使用即可，具体如下：

## 构造函数

```
image.laplacian(size[, sharpen=False[, mul[, add=0[, threshold=False[, offset=0[, invert=False[, mask=None]]]]]])
```

通过边缘检测拉普拉斯核来对图像进行卷积

【size】 内核大小。1（3X3 内核），2（5X5 内核）或更高值；

【sharpen】 如果为 True，则为锐化图像。

\*更多参数说明请阅读 OpenMV 官方文档：

<https://docs.singtown.com/micropython/zh/latest/openmvcam/library/omv.image.html>

## 使用方法

直接调用该函数。

表 5-29 拉普拉斯函数

从上表可以看到，锐化使用方法只需要配置拉普拉斯函数终端 size 和 sharpen 即可，代码编写流程如下：

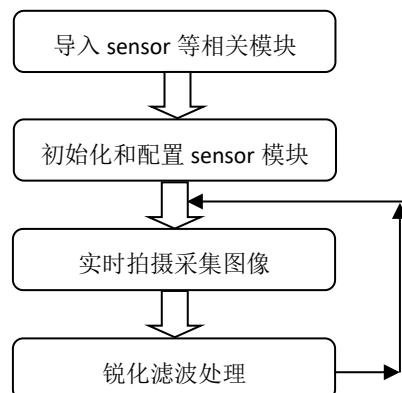


图 5-113 代码编写流程

官方参考代码路径【示例→Image Filters→sharpen\_filter.py】，具体如下：

```
# 锐化滤波示例  
#  
# 这个例子展示了如何使用拉普拉斯滤波器来锐化图像。
```

```
#  
#翻译和注释:01Studio  
  
import sensor, image, time  
  
#摄像头初始化  
sensor.reset() # Initialize the camera sensor.  
sensor.set_pixformat(sensor.GRAYSCALE) # or sensor.RGB565  
sensor.set_framesize(sensor.QQVGA) # or sensor.QVGA (or others)  
sensor.skip_frames(time = 2000) # Let new settings take affect.  
clock = time.clock() # Tracks FPS.  
  
while(True):  
    clock.tick() # Track elapsed milliseconds between snapshots().  
    img = sensor.snapshot() # Take a picture and return the image.  
  
    # 运行拉普拉斯内核。  
    img.laplacian(1, sharpen=True)  
  
    print(clock.fps()) #打印 FPS
```

### ● 实验结果:

运行程序，我们可将代码 `img.laplacian(1, sharpen=True)` 注释，表示不进行锐化，与锐化进行比较。可见锐化后的图片更清晰。所得结果如下：

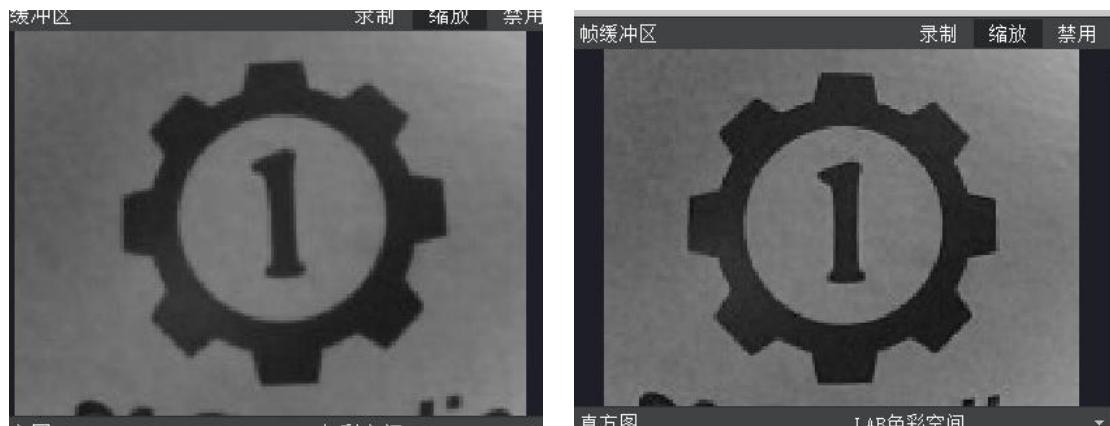


图 5-114 锐化前（左）与锐化后（右）比较

● 总结：

本节学习了图像滤波中的锐化，主要实现的功能是让拍摄的图片变得更清晰。这是我们在 Photoshop 中经常使用的功能。

### 5.9.3 图像翻转

- **前言:**

图像旋转和镜像是经常用到的，比如某些场合我们需要将摄像头硬件反过来安装，如装在天花板上的监控，那么我们未来正确地显示图像就会使用图像旋转或者镜像功能。学会了这个功能我们足以应付摄像头各种不同安装时的应用场景。

- **实验平台:**

pyAI-OpenMV4。



图 5-115 pyAI-OpenMV4

- **实验目的:**

编程实现图像的各种旋转和镜像。

- **实验讲解:**

不难想象，一张图片有 0、90、180、270 四个角度一共有 4 种情况，加上镜像那么一共有八种显示情况，OpenMV4 就是通过垂直反转、水平镜像、转置 3 个参数 ( $2^3=8$ ) 来确定图像的 8 种显示方式。使用的是 `image` 下的 `replace` 函数，具体如下：

## 构造函数

```
image.replace(image[, hmirror=False[, vflip=False[, transpose=False[, mask=None]]]])
```

图像翻转功能。

【image】要操作的图像对象；

【hmirror】如果为 True，则水平镜像替换图像。

【vflip】如果为 True，则垂直翻转替换图像。

【transpose】如果为 True，则沿对角线翻转图像。

## 使用方法

直接调用该函数。

例：

vflip=False, hmirror=False, transpose=False -> 旋转 0°

vflip=True, hmirror=False, transpose=True -> 旋转 90°

vflip=True, hmirror=True, transpose=False -> 旋转 180°

vflip=False, hmirror=True, transpose=True -> 旋转 270°

表 5-30 图像翻转函数

可见图像的各种翻转就是通过配置函数参数实现，代码编写流程如下：

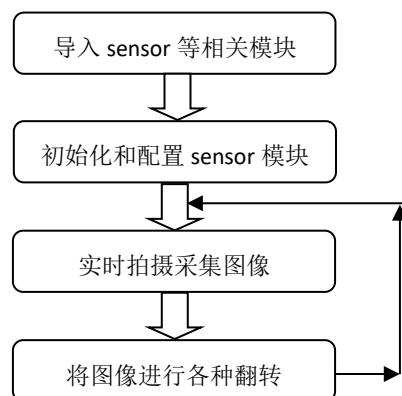


图 5-116 代码编写流程

官方参考代码路径【示例→Image Filters→vflip\_hmirror\_transpose.py】，具体如下：

```
# 垂直反转 - 水平镜像 - 转置

#
# 这个示例展示了如何将图像实现 垂直反转 - 水平镜像 - 转置
# 具体参数如下：3 个参数一共有 8 中情况，以下罗列了 4 种无镜像情况
#
# vflip=False, hmirror=False, transpose=False -> 0 degree rotation
# vflip=True,  hmirror=False, transpose=True  -> 90 degree rotation
# vflip=True,  hmirror=True,  transpose=False -> 180 degree rotation
# vflip=False, hmirror=True,  transpose=True   -> 270 degree rotation
#
#翻译和注释：01Studio

import sensor, image, time, pyb

#摄像头初始化
sensor.reset()

sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

mills = pyb.millis()
counter = 0

while(True):
    clock.tick()

    img = sensor.snapshot().replace(vflip=(counter//2)%2,
                                    hmirror=(counter//4)%2,
```

```
        transpose=(counter//8)%2)

# 也可以手动修改参数观察图像变化情况。

#img = sensor.snapshot().replace(vflip=False,
#                                  hmirror=False,
#                                  transpose=False)

if (pyb.millis() > (mills + 1000)):

    mills = pyb.millis()

    counter += 1

print(clock.fps())
```

上面代码中通过 `counter` 变量和运算符`//`和`%`的结合，巧妙地实现了 8 种参数组合循环。我们也可以手动修改参数观察图像变化情况。

- **实验结果：**

运行代码，可以看到图像在 8 种显示方式中不断循环。





图 5-117 实验部分图片展示

● **总结：**

本节学习了图像的各种旋转和镜像，有了这个技能，无论我们的 OpenMV4 如何安装，都可以轻松实现画面的调整。

## 第6章 拓展模块

由于 OpenMV 的集成度非常高，因此为了完全更多的外扩功能，就需要通过拓展模块来实现。如 LCD 显示屏、舵机控制、WiFi 模块、摄像头镜头等。本章将对 OpenMV4 的常用拓展模块进行讲解。

## 6.1 LCD 显示屏

- **前言：**

我们前面的实验实时观察图像都是使用 OpenMV IDE 的实时缓冲区，有些时候我们需要离线观察，这时候可以使用 LCD，可以实现本地显示。这大大方便调试和应用。

- **实验平台：**

pyAI-OpenMV4 开发套件 + 01Studio 1.77（128x160） LCD 显示屏。

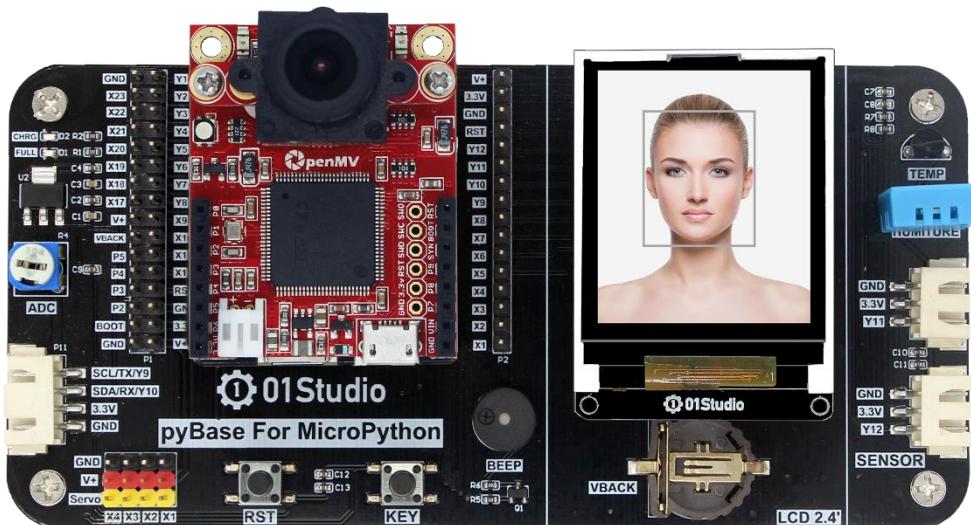


图 6-1 pyAI-OpenMV4 开发套件

- **实验目的：**

使用 LCD 实时显示摄像头采集的图像。

- **实验讲解：**

OpenMV IDE 集成了 LCD 驱动模块，目前只支持 1 款，分辨率必须为(128\*160)，SPI 接口的 LCD 模块。其对象说明如下：

构造函数
<code>import lcd</code>
导入 lcd 模块

使用方法
<code>lcd.init(type=1)</code>
初始化 lcd 模块。 【type】表明 lcd 扩展板的类型； 0: None 1: lcd (128*160) *目前仅支持 1 中类型的 LCD
<code>lcd.deinit()</code>
反初始化 lcd 模块，释放 GPIO 引脚。
<code>lcd.display(image, roi=Auto)</code>
在 lcd 上显示图片。 【image】显示的图片； 【roi】显示区域(x,y,w,h)。  *更多关于 LCD 使用方法说明，请阅读官方文档： <a href="http://docs.openmv.io/library/omv.lcd.html">http://docs.openmv.io/library/omv.lcd.html</a>

表 6-1 LCD 对象

结合前面学习过的摄像头的应用，本实验的代码编写流程如下：

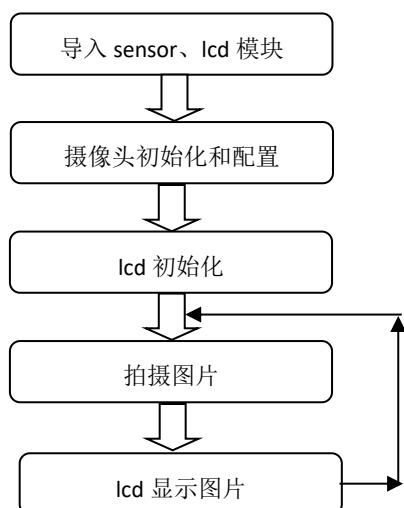


图 6-2 代码编写流程图

官方参考代码路径【示例→LCD-Shield→lcd.py】，具体如下：

```
# LCD 示例程序

#
# 提示：这个实验需要外接一个 LCD 模块.

#
# 本示例实现了 LCD 实时显示摄像头帧缓冲画面.

#
#翻译和注释: 01Studio

import sensor, image, lcd

#摄像头初始化
sensor.reset() # 初始化摄像头.
sensor.set_pixformat(sensor.RGB565) # 或者 sensor.GRAYSCALE
sensor.set_framesize(sensor.QQVGA2) # LCD 的分辨率为 128x160 .

#LCD 初始化
lcd.init()

while(True):
    lcd.display(sensor.snapshot()) # 拍照和显示图像.
```

### ● 实验结果：

在 IDE 中运行代码或者将代码文件修改成 main.py 拷贝到 OpenMV4 里面，可以看到 LCD 实时显示采集图像。

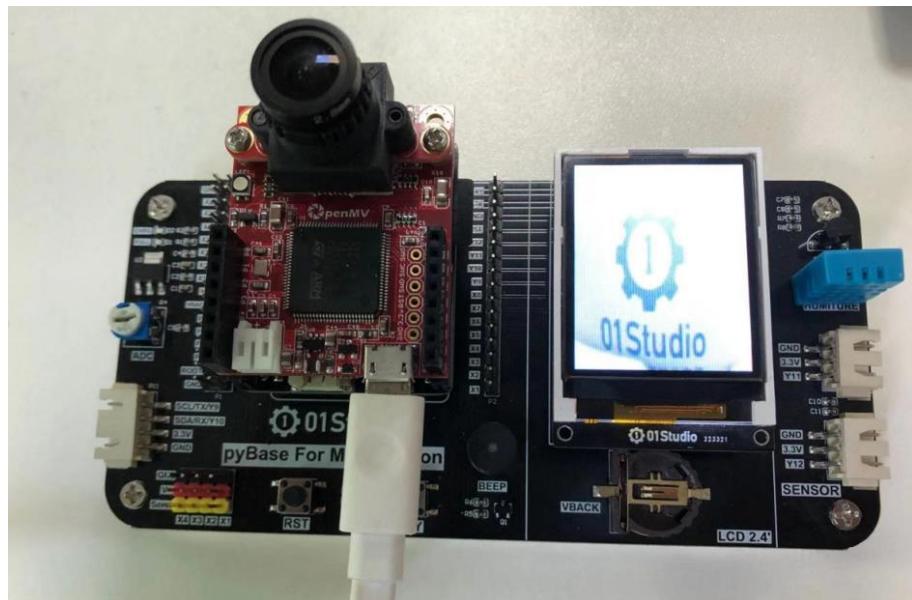


图 6-3 LCD 实时显示

你也可以直接将 LCD 显示屏插在 pyAI-OpenMV4 的背面来进行拍摄。



图 6-4 直接用 pyAI-OpenMV4 接 LCD 模块拍摄

### ● 总结：

本节学习了 LCD 的使用，也是做了最基础的例子。其实 LCD 的用途非常广泛，我们前面学习的 OpenMV4 的大部分例程只要修改一下图像分辨率参数，全部都可以用这个 LCD 来实时显示。比如巡线实验，可以通过 LCD 实时显示线路轨迹。有兴趣的用户可以自行修改测试。

## 6.2 舵机

### ● 前言：

舵机又叫伺服电机，是一个可以旋转特定角度的电机，可转动角度通常是 $90^\circ$ 、 $180^\circ$  和  $360^\circ$  ( $360^\circ$  可以连续旋转)。我们看到的机器人身上就有非常多的舵机，它们抬手或者摇头的动作往往是通过舵机完成，因此机器人身上的舵机越多，意味着动作越灵活。

### ● 实验平台：

pyAI-OpenMV4 开发套件和舵机。需要接上锂电池。

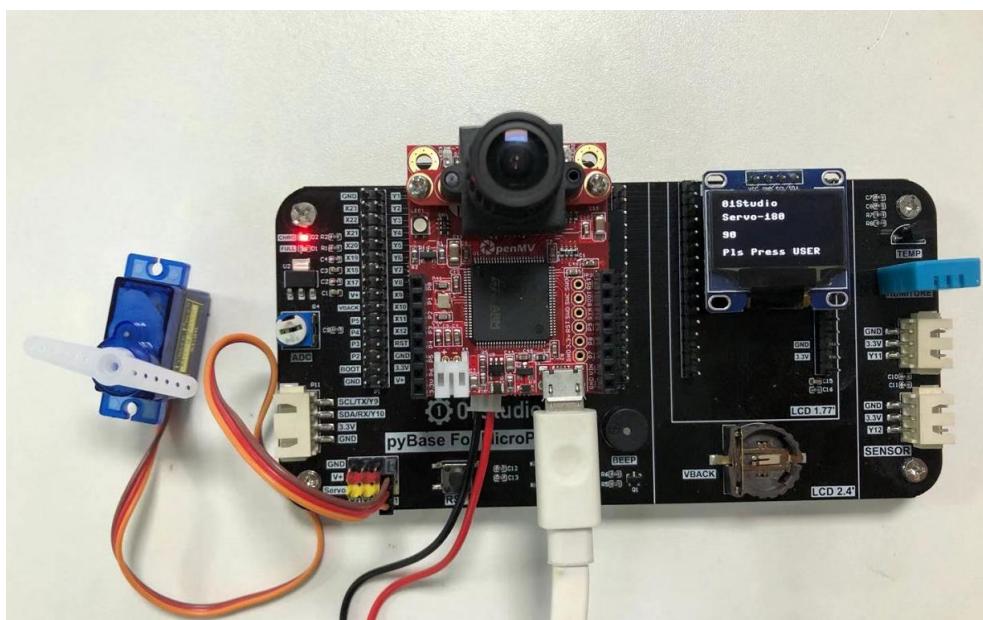


图 6-5 舵机接线

通常情况下：黑色表示 GND，红色表示 VCC，橙色表示信号线。

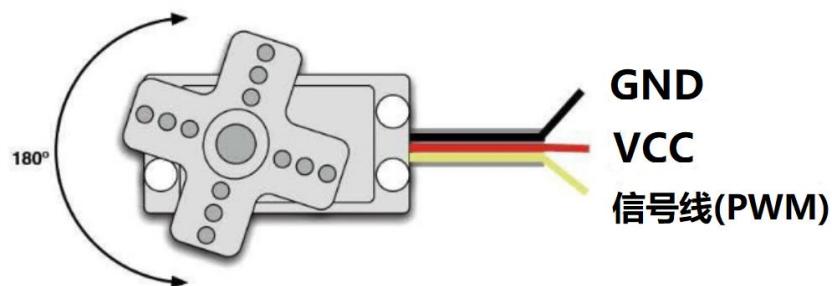


图 6-6 舵机示意图

### ● 实验目的:

通过编程实现对舵机的控制。

### ● 实验讲解:

伺服电机对象通过 3 线（信号，电源，地）控制，pyAI-OpenMV4 核心板有 3 个引脚可以用于直接控制舵机，分别连接到 pyBase 底板的 X1,X2,X3 引脚。对应关系是 P7→Servo1→X1, P8→Servo2→X2, P9→Servo3→X3。

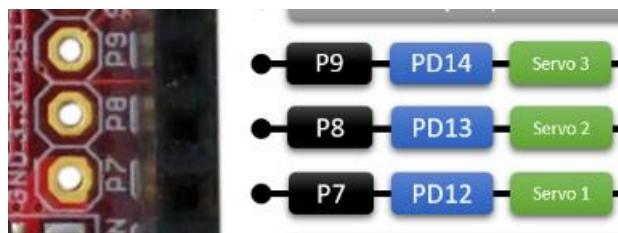


图 6-7 pyAI-OpenMV4 舵机控制引脚

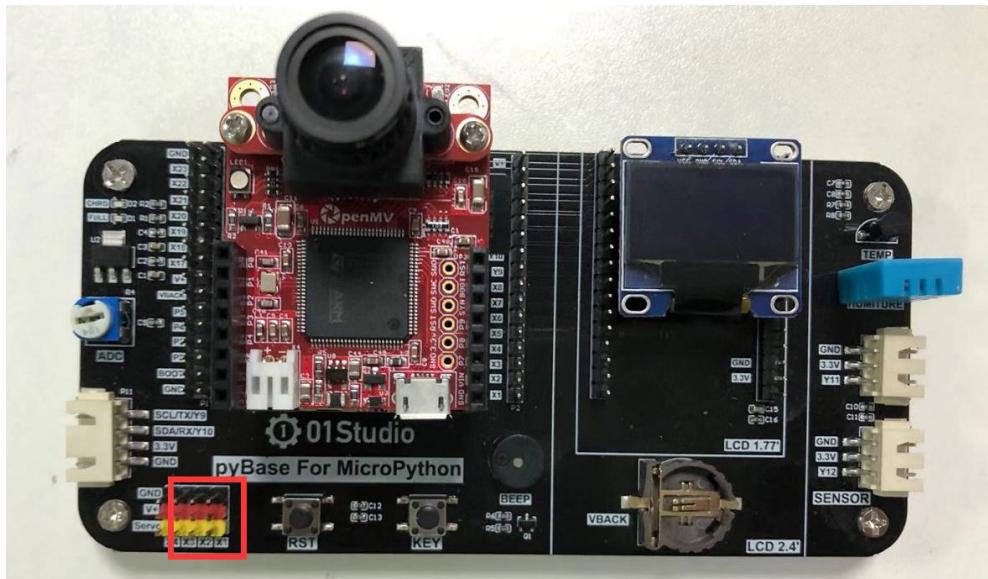


图 6-8 pyBase 对应的 3 路舵机接口

180° 舵机的控制一般需要一个 20ms 左右的时基脉冲，该脉冲的高电平部分一般为 0.5ms-2.5ms 范围内的角度控制脉冲部分，总间隔为 2ms。以 180 度角度伺服为例，在 MicroPython 编程对应的控制关系是从 -90° 至 90°，示例图如下：

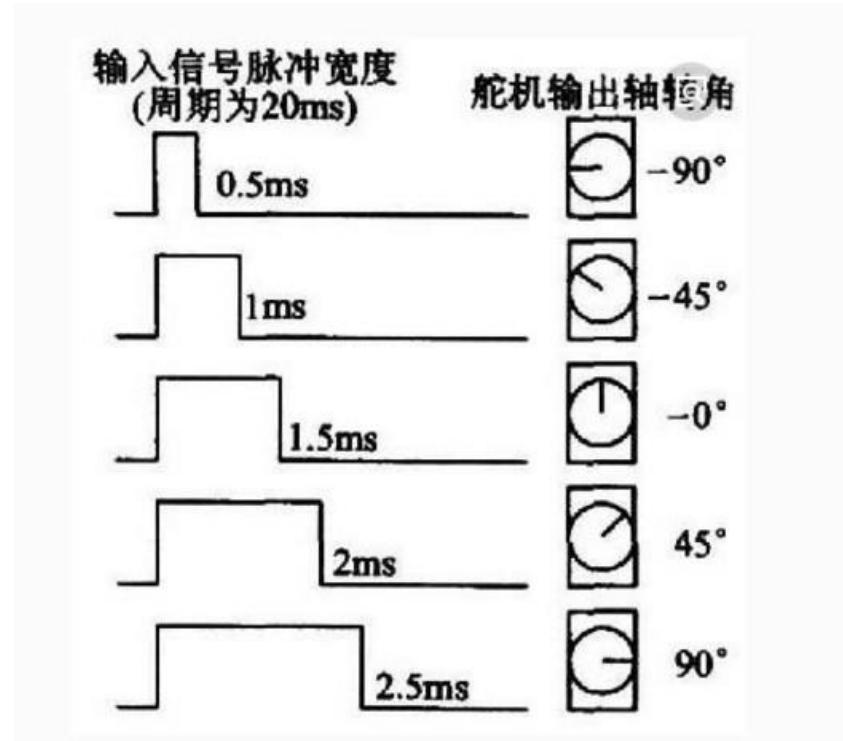


图 6-9 PWM 与角度关系

而对于 360° 连续旋转舵机，上面的脉冲表则对应从正向最大速度旋转到反向最大速度旋转的过程。

OpenMV 自带了舵机 (Servo) 的对象模块，我们可以直接调用，下面是 Servo 对象说明。

构造函数
<code>s=pyb.Servo(id)</code>
构建舵机对象。id 是 1-3，对应引脚 P7-P9
使用方法
<code>s.angle(angle,time=0)</code>
角度控制。angle:角度[-90 至 90]; time:旋转到指定角度的时间，单位 ms， 默认 0 表示最快
<code>s.speed(speed,time=0)</code>
速度控制（针对 360 度连续旋转舵机）。speed:速度[-100 至 100]，time: 开始启动到指定速度的时间，单位 ms， 默认 0 表示以最快

表 6-2 舵机对象

可以看到 MicroPython 让舵机控制变得非常简单，代码编程流程图如下：

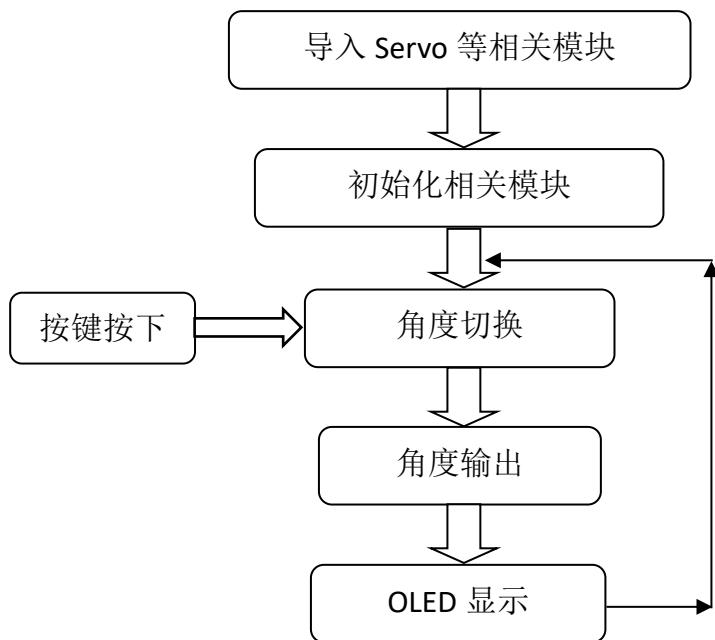


图 6-10 编程流程图

参考代码如下：

```
...  
实验名称: 舵机(Servo)-180°  
版本: v1.0  
日期: 2019.9  
作者: 01Studio  
说明: 通过 USER 按键让让舵机旋转不同角度。  
...
```

```
#导入相关模块  
from pyb import Servo,ExtInt  
from machine import Pin,I2C  
from ssd1306x import SSD1306_I2C
```

```

#初始化相关模块

i2c = I2C(sda=Pin("P0"), scl=Pin("P2"), freq=80000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

s1 = Servo(1)      #构建舵机对象 s1, 输出引脚为 X1

#定义 5 组角度: -90、-45、0、45、90
angle=[ -90,-45,0,45,90]

key_node = 0 #按键标志位
i = 0          #用于选择角度

#####
# 按键和其回调函数
#####

def key(ext):
    global key_node
    key_node = 1

ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, key) #下降沿触发, 打开上拉电阻

#####
# OLED 初始显示
#####

oled.fill(0) # 清屏显示黑色背景
oled.text('01Studio', 0, 0) # 首行显示 01Studio
oled.text('Servo-180', 0, 15) # 次行显示实验名称
oled.text(str(angle[i]), 0, 35) # 显示当前角度

```

```

oled.text('Pls Press USER', 0, 55) #提示按按键
oled.show()

#X1 指定角度,启动时 i=0, 默认-90°
s1.angle(angle[i])

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 5:
            i = 0
        key_node = 0 #清空按键标志位

    #X1 指定角度
    s1.angle(angle[i])

    #显示当前频率
    oled.fill(0) # 清屏显示黑色背景
    oled.text('01Studio', 0, 0) # 首行显示 01Studio
    oled.text('Servo-180', 0, 15) # 次行显示实验名称
    oled.text(str(angle[i]), 0, 35) # 显示当前角度
    oled.text('Pls Press USER', 0, 55) #提示按按键
    oled.show()

```

### ● 实验结果：

将实验代码拷贝到 OpenMV4。包括主函数代码和 OLED 驱动代码。



图 6-11

由于 OpenMV4 没有电源 V+输出引脚，因此需要让 pyAI-OpenMV4 接上锂电池，从而给 pyBase 的 V+供电。当然你可以将程序拷贝到 OpenMV4 后，将 USB 口插到 pyAI-OpenMV4 转接板。

将 180° 舵机插到 X1 三线接口，通过按键按下实现舵机的不同角度，同时在 OLED 上显示角度值。

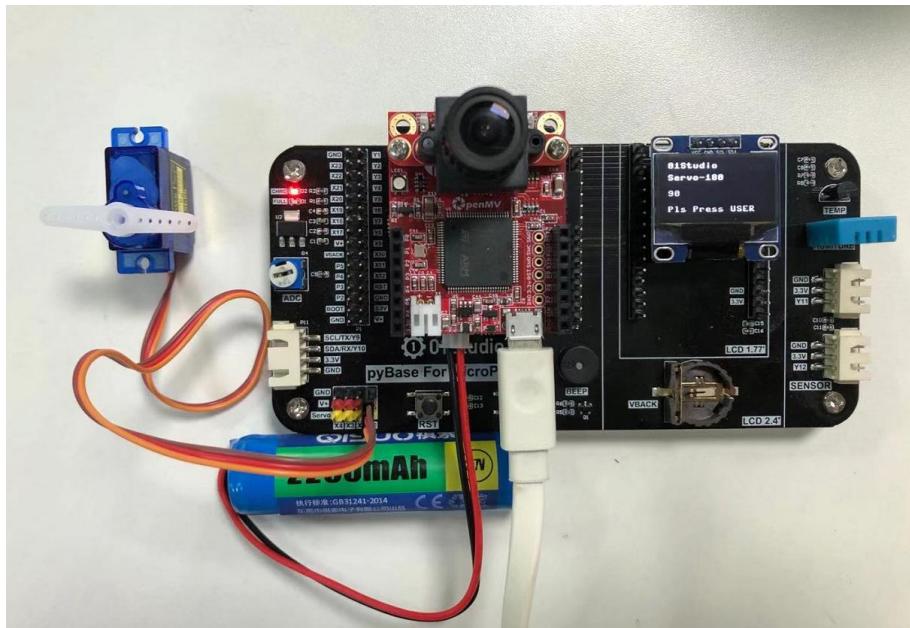


图 6-12 舵机角度 90°

### ● 实验拓展：

我们刚刚实现了 180° 舵机的角度控制，现在来做一下 360° 连续旋转舵机

的实验， $360^\circ$  连续旋转舵机可以实现直流减速电机功能，用在小车或者航模上。

实验的基础条件和方式不变，我们只需要将角度参数【-90 至 90】改成速度【-100 至 100】即可。修改后的程序代码如下：

```
...
实验名称: 舵机(Servo)-360°连续旋转
版本: v1.0
日期: 2019.9
作者: 01Studio
说明: 通过 USER 按键让舵机实现不同速度旋转。
...

#导入相关模块
from pyb import Servo,ExtInt
from machine import Pin,I2C
from ssd1306x import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"), freq=80000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

s1 = Servo(1)      #构建舵机对象 s1, 输出引脚为 X1

#定义 5 组速度值
speed=[-100,-50,0,50,100]

key_node = 0  #按键标志位
i = 0          #用于选择角度

#####
```

```

# 按键和其回调函数

#####
def key(ext):
    global key_node
    key_node = 1

#下降沿触发，打开上拉电阻
ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, key)

#####
# OLED 初始显示

#####
oled.fill(0) # 清屏显示黑色背景
oled.text('01Studio', 0, 0) # 首行显示 01Studio
oled.text('Servo-360', 0, 15) # 次行显示实验名称
oled.text(str(speed[i]), 0, 35) # 显示当前速度值
oled.text('Pls Press USER', 0, 55) #提示按按键
oled.show()

#X1 指定速度，启动时 i=0， 默认 -100
s1.speed(speed[i])

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 5:
            i = 0
            key_node = 0 #清空按键标志位

```

```

#X1 指定角度

s1.speed(speed[i])

#显示当前频率

oled.fill(0) # 清屏显示黑色背景

oled.text('01Studio', 0, 0) # 首行显示 01Studio

oled.text('Servo-360', 0, 15) # 次行显示实验名称

oled.text(str(speed[i]), 0, 35) # 显示当前速度值

oled.text('Pls Press USER', 0, 55) #提示按键

oled.show()

```

实验结果如下：

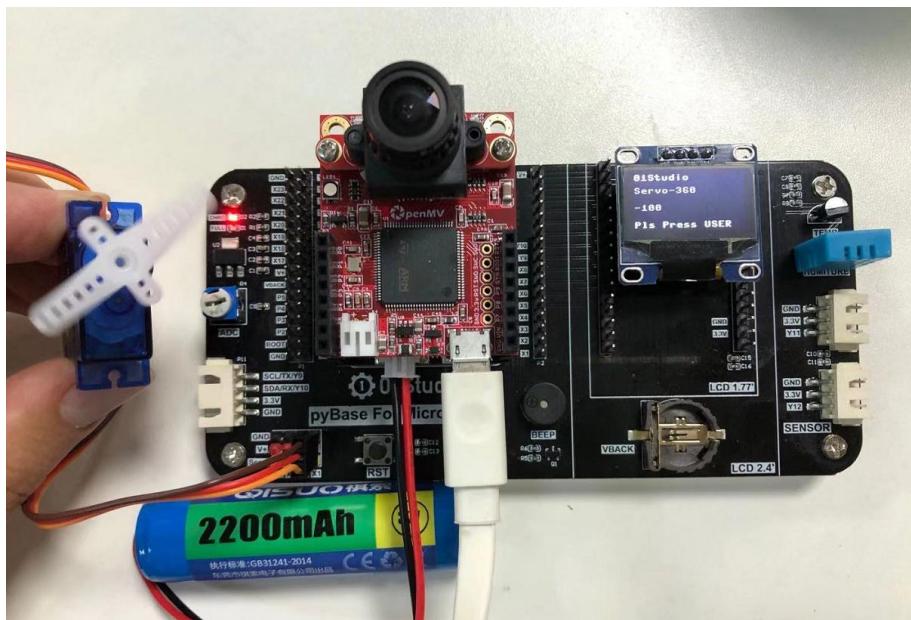


图 6-13 360°连续旋转舵机

### ● 总结：

通过本节我们学会了使用不同类型的舵机，通过多路电机的组合使用，可以实现模型、航模、小车、机器人的实验。

## 6.3 多路舵机模块

- **前言：**

在上一节我们已经学习了舵机的应用。今天我们来学习一下如何同时控制多路舵机。

- **实验平台：**

pyAI-OpenMV4 开发套件、OpenMV 舵机模块和舵机。需要接上锂电池。

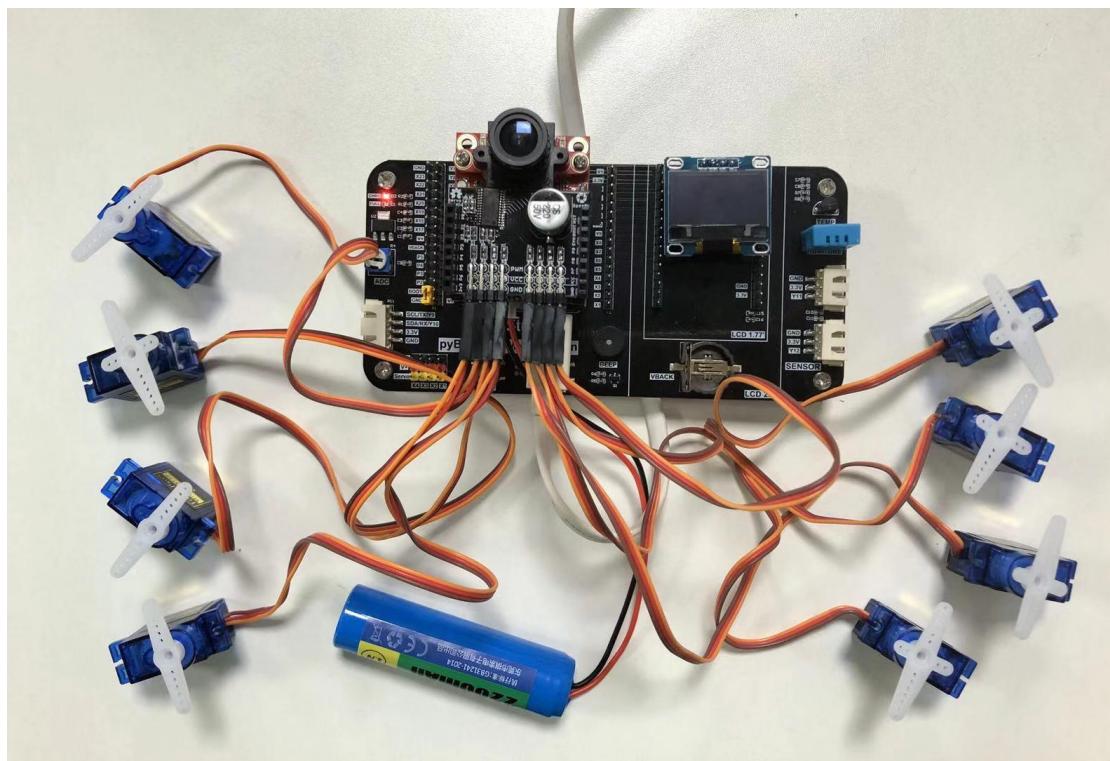


图 6-14 8 路舵机接线

- **实验目的：**

通过编程实现对 8 路舵机的控制。

- **实验讲解：**

我们这上一节已经学习过舵机的原理了，这里不再重复。OpenMV 舵机拓展模块是使用了 PCA9685 芯片，通过 I2C 跟 OpenMV 通讯，使得 OpenMV 只需要一个 I2C 接口就可以同时控制多路舵机。舵机模块功能参数说明如下：

功能参数	
主控芯片	PCA9685
控制方式	I2C 总线
输出接口	8 路舵机 (PWM)
其他功能	支持级联，最多支持 16 路输出

表 6-3 舵机模块功能参数

OpenMV 集成了基于 PCA9685 舵机驱动的对象模块，我们可以直接调用，下面是对象相关说明。

构造函数
<code>servo = Servos(i2c, address=0x40, freq=50, min_us=500, max_us=2500, degrees=180)</code>
构建舵机模块对象。 <b>【i2c】</b> 控制的 I2C 接口， 默认接口是 I2C2,即 <code>sda=Pin('P5')</code> , <code>scl=Pin('P4')</code> ; <b>【address】</b> I2C 地址， 默认是 0x40 （拓展板背部有焊点，焊接后地址为 0x60，方便同时控制 16 路舵机） <b>【freq】</b> 根据舵机配置，我们使用舵机周期是 20ms，即频率 50Hz; <b>【min_us】</b> 根据舵机配置， 占空比最低 0.5ms,即 500us <b>【max_us】</b> 根据舵机配置， 占空比最低 2.5ms,即 2500us <b>【degrees】</b> 舵机初始化角度。
使用方法
<code>servo.position(i, 0)</code> 舵机控制。 <b>【i】</b> 0-7,分别对应舵机 S0-S7。总共 8 路； <b>【0】</b> 角度。0-180。

表 6-4 舵机模块对象

可以看到通过 I2C 可以轻松控制多路电机，代码编程流程图如下：

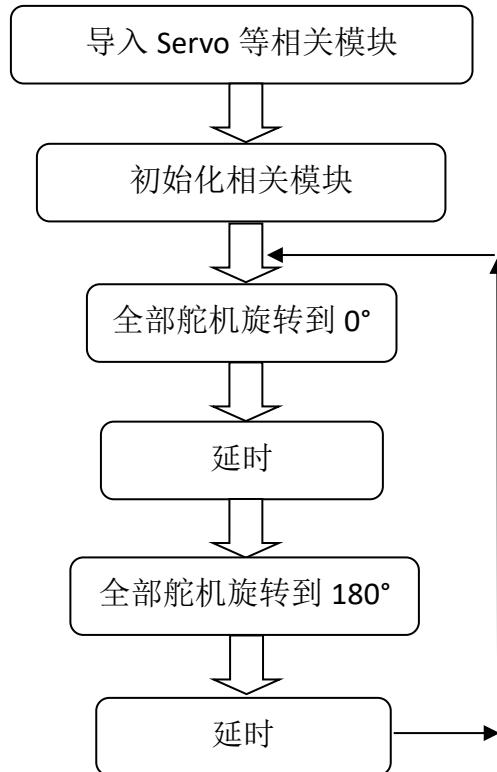


图 6-15 编程流程图

官方参考代码路径【示例→Servo-Shield→main.py】，注意舵机初始化参数需要修改成我们使用的舵机参数。具体如下：

```

# 舵机模块例程.

#
# 本实验请遵循以下要求：
#
# 1. 将舵机连接到任意 PWM 输出口.
# 2. 需要连接 3.7V 锂电池 (或 5V 电源).
# 3. 将 pca9685.py 和 servo.py 文件拷贝到 OpenMV 的文件系统，然后复位.
# 4. 在 IDE 里运行本实验代码.

#
#翻译和注释: 01Studio


import time
from servo import Servos
from machine import I2C, Pin
  
```

```

#初始化 I2C

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


#初始化舵机模块

servo = Servos(i2c, address=0x40, freq=50, min_us=500, max_us=2500,
degrees=180)

while True:

    for i in range(0, 8):

        servo.position(i, 0) #8 路舵机旋转到 0°

        time.sleep(500)

    for i in range(0, 8):

        servo.position(i, 180)#8 路舵机旋转到 180°

        time.sleep(500)

```

### ● 实验结果：

接上锂电池。将 pca9685.py 和 servo.py 文件拷贝到 OpenMV 的文件系统，然后复位。在 IDE 运行以上代码，可以看到 8 路舵机依次旋转到 0 度和 180 度。

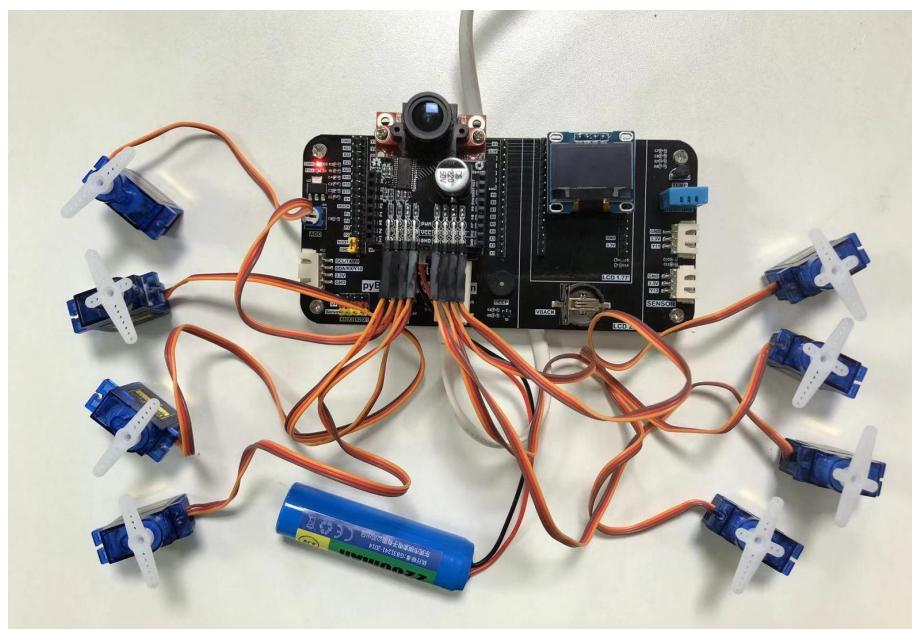


图 6-16

- **实验拓展:**

在 8 路舵机模块的背面有一个焊接点，将这个焊接上可以改变其 I2C 地址，从而实现 2 个 8 路舵机模块叠加，控制 16 路舵机。焊接焊点后的模块 I2C 地址为 0x60，有兴趣的用户可以自行测试。

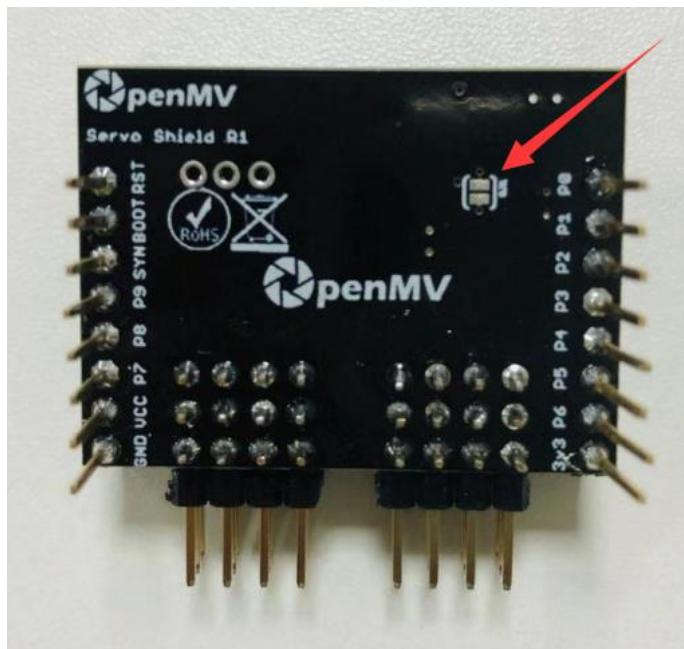


图 6-17 8 路舵机模块焊点

- **总结:**

本节学习了多路舵机的编程应用，可以见到通过简单的 I2C 实现了多路舵机控制，本模块也适用于其它平台的 MicroPython 开发板。

## 6.4 WiFi 模块

OpenMV 可以通过 WiFi 扩展模块连接无线路由器，从而实现 Socket、MQTT、图传等相关无线和物联网应用。模块的接口和功能参数介绍如下：



图 6-18 OpenMV WiFi 模块 PinOut 图

功能参数	
WiFi 主控	WINC1500
控制方式	SPI 接口
通信速率	IEEE 802.11b (up to 11Mbps) IEEE 802.11g (up to 48Mbps) IEEE 802.11n (up to 48Mbps)
加密类型	IEEE 802.11 WEP IEEE 802.11 WPA IEEE 802.11 WPA2
功耗	待机 80mA(3.3V) 工作 100mA(3.3V)
适用平台	pyAI-OpenMV4

表 6-5 OpenMV WiFi 模块功能参数

### 6.4.1 连接无线路由器

- **前言:**

WIFI 是物联网中非常重要的角色，现在基本上家家户户都有 WIFI 网络了，通过 WIFI 接入到互联网，成了智能家居产品普遍的选择。而要想上网，首先需要连接上无线路由器。这一节我们就来学习如何通过 WiFi 扩展模块让 OpenMV 通过 MicroPython 编程实现连上路由器。

- **实验平台:**

pyAI-OpenMV4 和 OpenMV WiFi 模块。(如果使用 pyBase 需要将 OLED 拔掉。  
避免 IO 冲突)

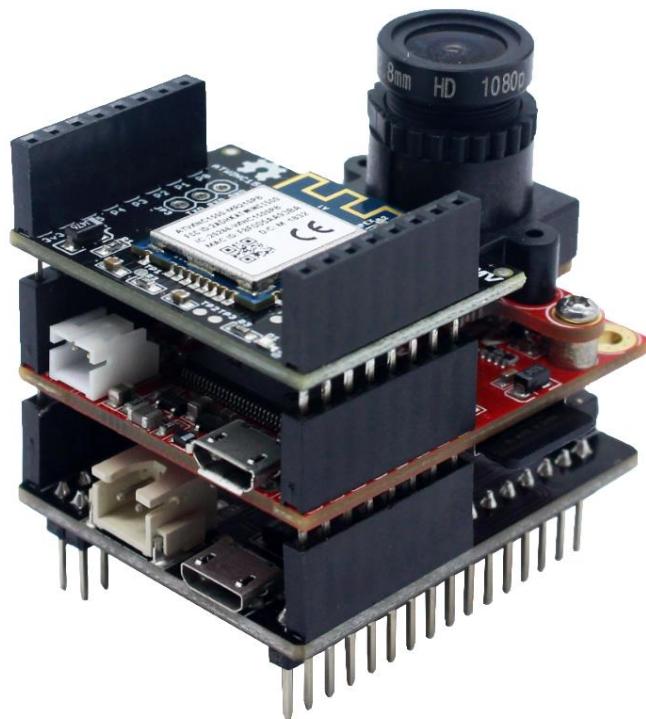


图 6-19 pyAI-OpenMV4

- **实验目的:**

编程实现连接路由器，将 IP 地址等相关信息通过串口终端打印(只支持 2.4G 网络)。

- **实验讲解:**

连接路由器上网是我们每天都做的事情，日常生活中我们只需要知道路由器

的账号和密码，就能使用电脑或者手机连接到无线路由器，然后上网冲浪。

OpenMV 已经集成了 network 模块，开发者使用内置的 network 模块函数可以非常方便地连接上路由器。

我们先来看看 network 基于 WiFi (WLAN 模块) 的构造函数和使用方法。

构造函数	
wlan = network.WINC()	构建 WIFI 模块连接对象。WINC 表示模块型号。
使用方法	
wlan.scan ()	扫描允许访问的 SSID。
wlan.isconnected()	检查设备是否已经连接上。返回 True: 已连接; False: 未连接。
wlan.connected(ssid,passwork,security)	WIFI 连接。 【ssid】账号; 【passwork】密码; 【security】加密方式。
wlan.ifconfig([ip,subnet,gateway,dns])	设备信息配置。ip: IP 地址; subnet: 子网掩码; gateway: 网关地址; dns: DNS 信息。(如果参数为空，则返回当前连接信息。)
wlan.disconnect()	断开连接。
*更多参数说明请阅读 OpenMV 官方文档: 文档链接: <a href="http://docs.openmv.io/library/network.WINC.html">http://docs.openmv.io/library/network.WINC.html</a>	

表 6-6 WINC 对象

从上表可以看到 MicroPython 通过模块封装，让 WIFI 联网变得非常简单。模块包含热点 AP 模块和客户端 STA 模式，热点 AP 是指电脑端直接连接 OpenMV 发出的热点实现连接，但这样你的电脑就不能上网了，因此我们一般情况下都是

使用 STA 模式。也就是电脑和设备同时连接到相同网段的路由器上。

代码编写流程如下：

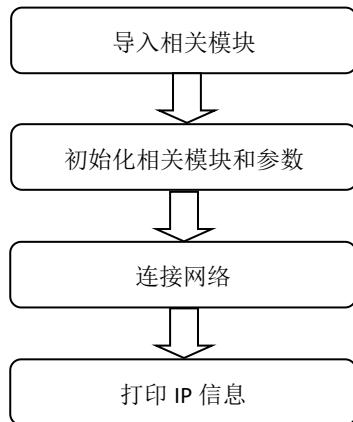


图 6-20 代码编写流程图

实验参考代码如下：

```
# WiFi 无线连接实验

#
# OpenMV 通过 WiFi 拓展模块连接无线网络
#
# 翻译和注释：01Studio


import network

SSID='01Studio' # WiFi 账号
KEY='88888888' # WiFi 密码


# 打印提示
print("Trying to connect... (may take a while)...")

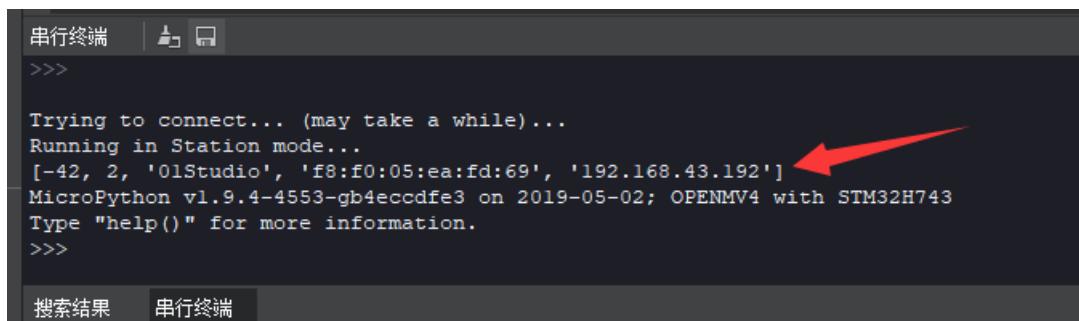

#构建 WiFi 对象
wlan = network.WINC()
```

```
#连接网络  
wlan.connect(SSID, key=KEY, security=wlan.WPA_PSK)  
  
#打印 IP 相关信息  
print(wlan.ifconfig())
```

上面代码在 `main.py` 文件中，我们也可以新建一个 `WIFI.py` 文件，然后通过模块引用方式来供 `main.py` 调用，以提高程序的灵活性。

### ● 实验结果：

运行程序，可以观察到连接成功后串口终端打印 IP 等信息。



```
串行终端 |    
>>>  
Trying to connect... (may take a while)...  
Running in Station mode...  
[-42, 2, '01Studio', 'f8:f0:05:ea:fd:69', '192.168.43.192']   
MicroPython v1.9.4-4553-gb4eccdfe3 on 2019-05-02; OPENMV4 with STM32H743  
Type "help()" for more information.  
>>>  
搜索结果 串行终端
```

图 6-21

### ● 总结：

本节是 WiFi 应用的基础，成功连接到无线路由器的实验后，后面就可以做 socket 和 MQTT 等相关网络通信的应用了。

## 6.4.2 Socket 通信

- 前言：

上一节我们学习了如何通过 MicroPython 编程实现 pyAI-OpenMV4 结合 WiFi 拓展模块连接到无线路由器。这一节我们则来学习一下 Socket 通信实验。Socket 几乎是整个互联网通信的基础。

- 实验平台：

pyAI-OpenMV4 和 OpenMV WiFi 模块。(如果使用 pyBase 需要将 OLED 拔掉。  
避免 IO 冲突)

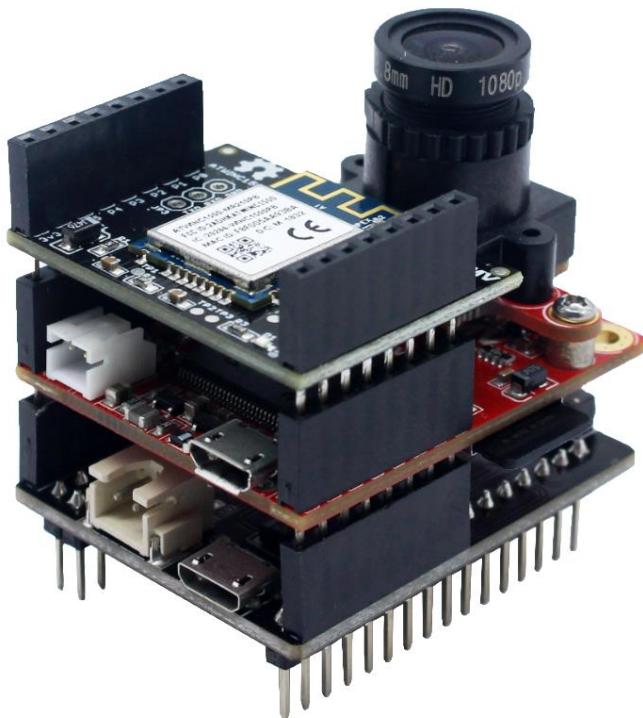


图 6-22 pyAI-OpenMV4 开发套件

- 实验目的：

通过 Socket 编程实现 pyAI-OpenMV4 与电脑服务器助手建立连接，相互收发数据。

- 实验讲解：

Socket 我们听得非常多了，但由于网络工程是一门系统工程，涉及的知识非

常广，概念也很多，任何一个知识点都能找出一堆厚厚的的书，因此我们经常会混淆。在这里，我们尝试以最容易理解的方式来讲述 Socket，如果需要全面了解，可以自行查阅相关资料学习。

我们先来看看网络层级模型图，这是构成网络通信的基础：

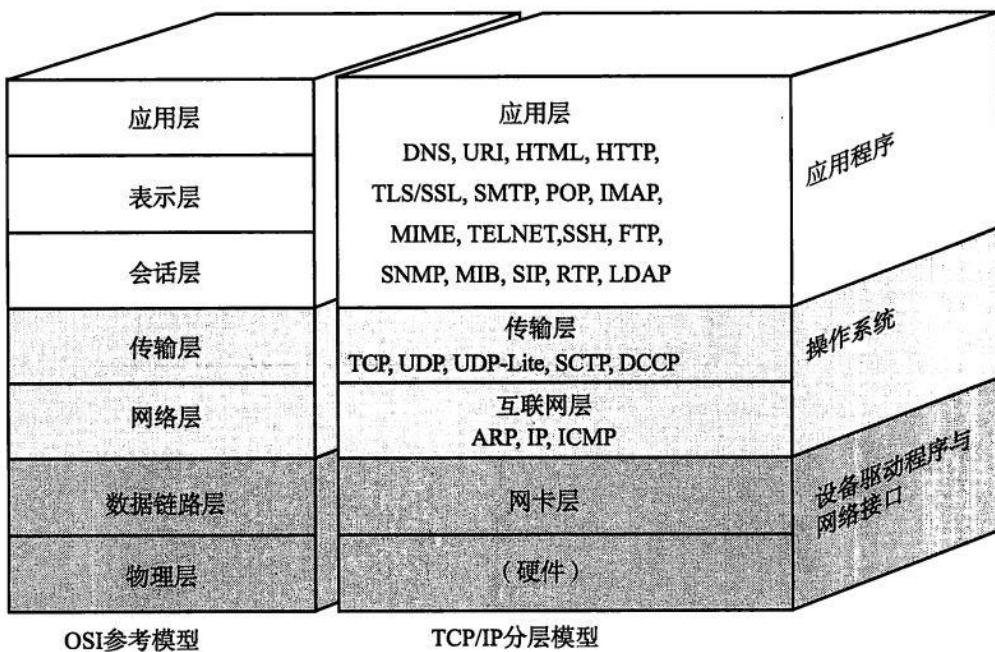


图 6-23 网络层级模型

我们看看 TCP/IP 模型的传输层和应用层，传输层比较熟悉的概念是 TCP 和 UDP，UDP 协议基本就没有对 IP 层的数据进行任何的处理了。而 TCP 协议还加入了更加复杂的传输控制，比如滑动的数据发送窗口（Slice Window），以及接收确认和重发机制，以达到数据的可靠传送。应用层中网页常用的则是 HTTP。那么我们先来解析一下这 TCP 和 HTTP 两者的关系。

我们知道网络通信是最基础是依赖于 IP 和端口的，HTTP 一般情况下默认使用端口 80。举个简单的例子：我们逛淘宝，浏览器会向淘宝网的网址（本质是 IP）和端口发起请求，而淘宝网收到请求后响应，向我们手机返回相关网页数据信息，实现了网页交互的过程。而这里就会引出一个多人连接的问题，很多人访问淘宝网，实际上接收到网页信息后就断开连接，否则淘宝网的服务器是无法支撑这么多人长时间的连接的，哪怕能支持，也非常占资源。

也就是应用层的 HTTP 通过传输层进行数据通信时，TCP 会遇到同时为多个应

用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字(Socket)接口。应用层可以和传输层通过 Socket 接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

简单来说，Socket 抽象层介于传输层和应用层之间，跟 TCP/IP 并没有必然的联系。Socket 编程接口在设计的时候，就希望也能适应其他的网络协议。

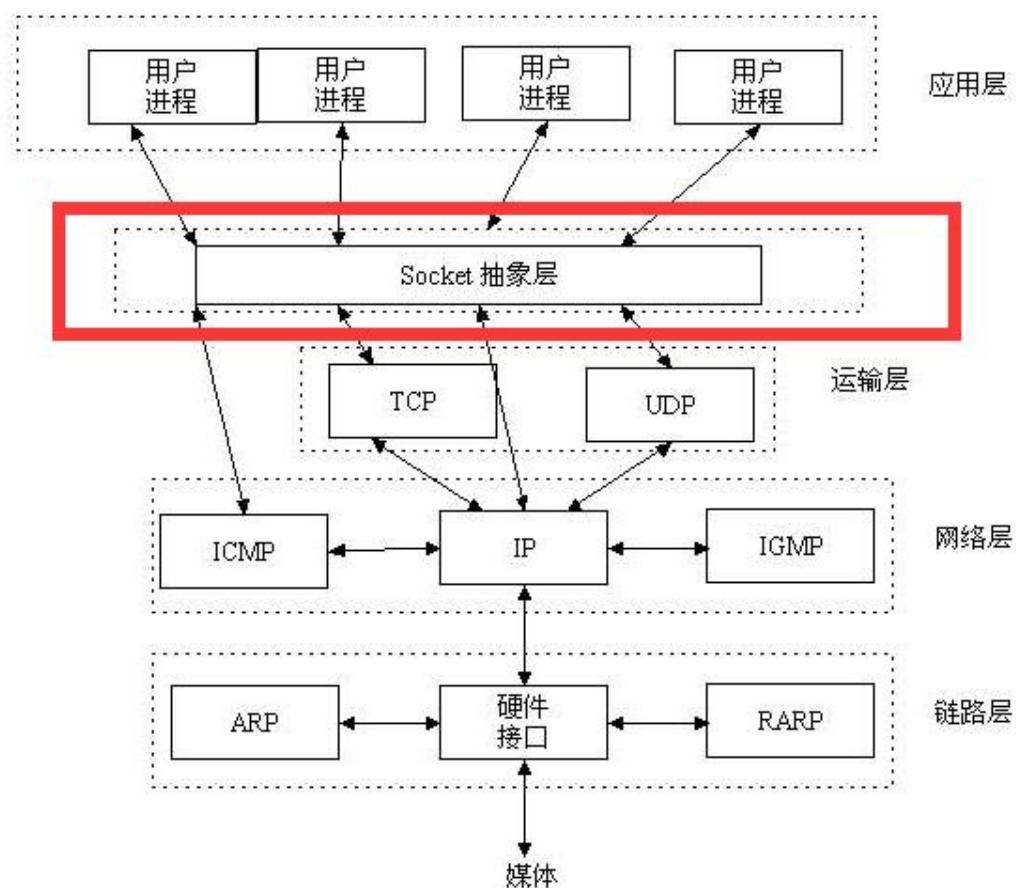


图 6-24 Socket 抽象层

套接字（socket）是通信的基石，是支持 TCP/IP 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：连接使用的协议（通常是 TCP 或 UDP），本地主机的 IP 地址，本地进程的协议端口，远地主机的 IP 地址，远地进程的协议端口。

所以，socket 的出现只是可以更方便的使用 TCP/IP 协议栈而已，简单理解就是其对 TCP/IP 进行了抽象，形成了几个最基本的函数接口。比如 create, listen,

accept, connect, read 和 write 等等。以下是通讯流程:

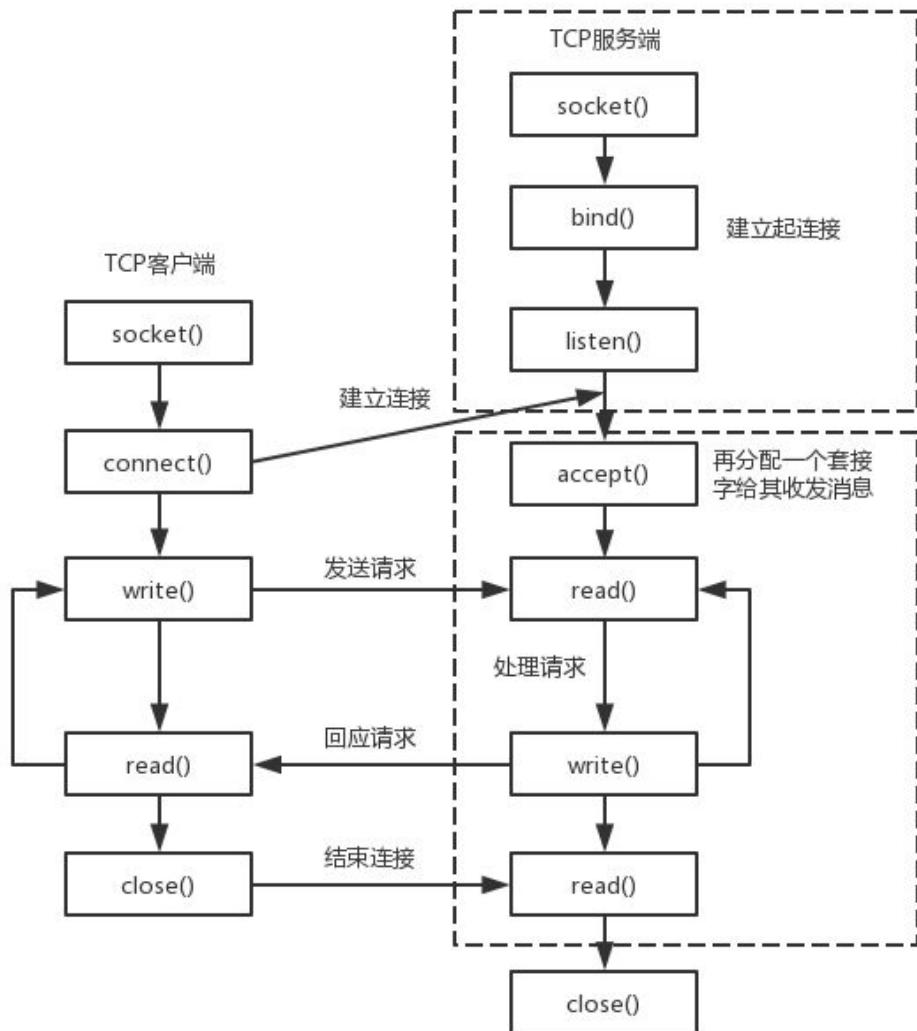


图 6-25 Socket 通信过程

从上图可以看到，建了 Socket 通信需要一个服务器端和一个客户端，以本实验为例，pyAI-OpenMV4 作为客户端，电脑使用网络调试助手作为服务器端，双方使用 TCP 协议传输。对于客户端，则需要知道电脑端的 IP 和端口即可建立连接。（端口可以自定义，范围在 0~65535，注意不占用常用的 80 等端口即可。）

以上的内容，简单来说就是如果用户面向应用来说，那么 OpenMV 只需要知道通讯协议是 TCP 或 UDP、服务器的 IP 和端口号这 3 个信息，即可向服务器发

起连接和发送信息。就这么简单。

MicroPython 已经封装好相关模块 `usocket`, 跟传统的 `socket` 大部分兼容，两者均可使用，本实验使用 `usocket`，对象如下介绍：

构造函数
<code>s=usocket.socket(af=AF_INET, type=SOCK_STREAM,proto=IPPROTO_TCP)</code>
构建 <code>usocket</code> 对象。 <code>af: AF_INET→IPV4, AF_INET6 → IPV6;</code> <code>type: SOCK_STREAM→TCP, SOCK_DGRAM→UDP;</code> <code>proto: IPPROTO_TCP→TCP 协议, IPPROTO_UDP→UDP 协议。</code> (如果要构建 TCP 连接，可以使用默认参数配置，即不输入任何参数。)
使用方法
<code>addr=usocket.getaddrinfo('www.01studio.org', 80)[0][-1]</code>
获取 Socket 通信格式地址。返回： ('47.91.208.161', 80)
<code>s.connect(address)</code>
创建连接。 <code>address: 地址格式为 IP+端口。例： ('192.168.1.115', 10000)</code>
<code>s.send(bytes)</code>
发送。 <code>bytes: 发送内容格式为字节</code>
<code>s.recv(bufsize)</code>
接收数据。 <code>bufsize: 单次最大接收字节个数。</code>
<code>s.bind(address)</code>
绑定，用于服务器角色
<code>s.listen([backlog])</code>
监听，用于服务器角色。 <code>backlog: 允许连接个数，必须大于 0。</code>
<code>s.accept()</code>
接受连接，用于服务器角色。
*其它更多用法请阅读 MicroPython 文档：(搜索: <code>usocket</code> ) 文档链接： <a href="http://docs.openmv.io/library/usocket.html">http://docs.openmv.io/library/usocket.html</a>

表 6-7 Socket 对象

本实验中 pyAI-OpenMV4 属于客户端，因此只用到客户端的函数即可。实验代码编写流程如下：

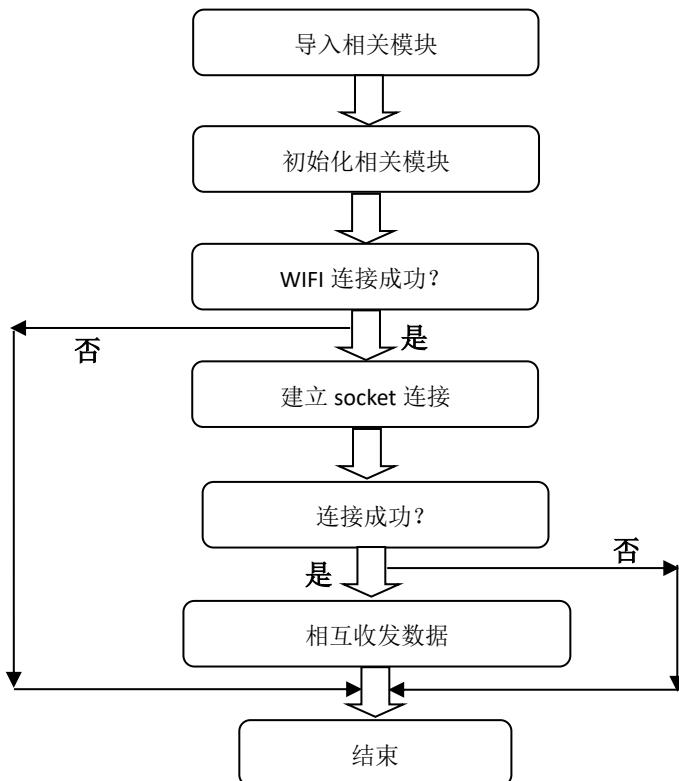


图 6-26 代码编写流程图

实验参考代码：

```
# TCP 客户端（Socket 通信）实验

#
# 通过 WiFi 模块编程实现 OpenMV 的 Socket 通信，数据收发。
#
#作者：01Studio

import network, usocket,pyb

# WiFi 信息
SSID='WeBee_office_2.4G' # Network SSID
KEY='webee0123456789' # Network key
```

```
#socket 数据接收中断标志位
socket_node = 0

# Init wlan module and connect to network
print("Trying to connect... (may take a while)...")

wlan = network.WINC()
wlan.connect(SSID, key=KEY, security=wlan.WPA_PSK)

# We should have a valid IP now via DHCP
print(wlan.ifconfig())

#创建 socket 连接，连接成功后发送“Hello 01Studio! ”给服务器。
client=usocket.socket()
addr=('192.168.1.117',10000) #服务器 IP 和端口
client.connect(addr)
client.send('Hello 01Studio!')

#开启定时器，周期 100ms,重复执行 socket 通信接收任务
def fun(tim):
    global socket_node
    socket_node = 1
    pyb.LED(3).toggle()

tim = pyb.Timer(4,freq=10)
tim.callback(fun)

while True:
```

```

if socket_node:

    text=client.recv(128) #单次最多接收 128 字节

    if text == '':
        pass

    else: #打印接收到的信息为字节，可以通过 decode('utf-8')转成字符串
        print(text)

        client.send('I got:' +text.decode('utf-8'))


socket_node=0

```

WIFI 连接代码在上一节已经讲解，这里不再重复，程序在连接成功后建了 Socket 连接，连接成功发送 ‘Hello 01Studio!’ 信息到服务器。另外 OpenMV 定时器设定了每 100ms 处理从服务器接收到的数据。将接收到数据通过串口打印和重新发送给服务器。

### ● 实验结果：

先在电脑端打开网络调试助手并建立服务器，软件在 零一科技 (01Studio) MicroPython 开发套件配套资料\_latest\01-开发工具\01-Windows\网络调试助手 下的 NetAssist.exe ，直接双击打开即可！

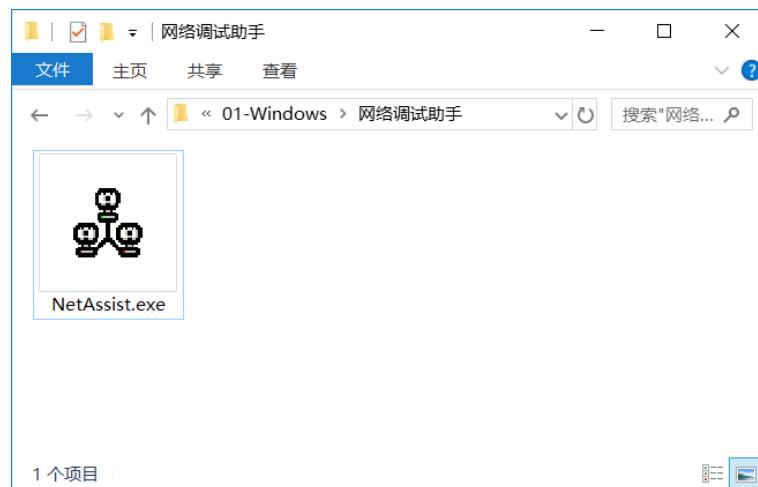


图 6-27 网络调试助手

以下是新建服务器的方法，打开网络调试助手后在左上角协议类型选择 TCP Server；中间的本地 IP 地址是自动识别的，不要修改，这个就是服务器的 IP 地址。然后端口写 10000（0-65535 都可以。），点击连接，成功后红点亮。如下图：



图 6-28 TCP 服务器配置

这时候服务器已经在监听状态！用户需要根据自己的实际情况自己输入 WIFI 信息和服务器 IP 地址+端口。即修改上面的代码以下部分内容。（服务器 IP 和端口可以在网络调试助手找到。）

```
# WiFi 信息  
SSID='01Studio' # Network SSID  
KEY='88888888' # Network key
```

```
addr=('192.168.1.115',10000) #服务器 IP 和端口
```

下载程序，开发板成功连接 WIFI 后，发起了 socket 连接，连接成功可以可以看到网络调试助手收到了开发板发来的信息。在下方列表多了一个连接对象，点击选中：

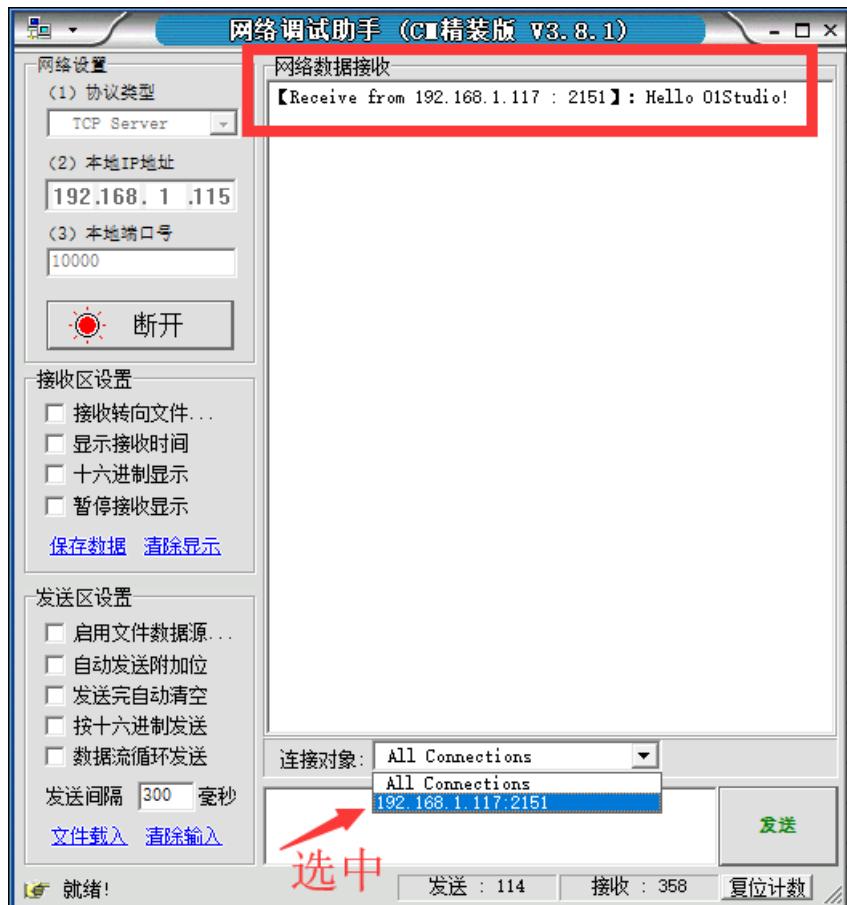


图 6-29 选中连接对象

选中后我们在发送框输入信息“Hi”，点击发送，可以看到开发板的 REPL 打印出来信息 Hi。为字节数据。另外由于程序将收到的信息发回给服务器，所以在网络调试助手中也接收到开发板返回的信息：I got:Hi。

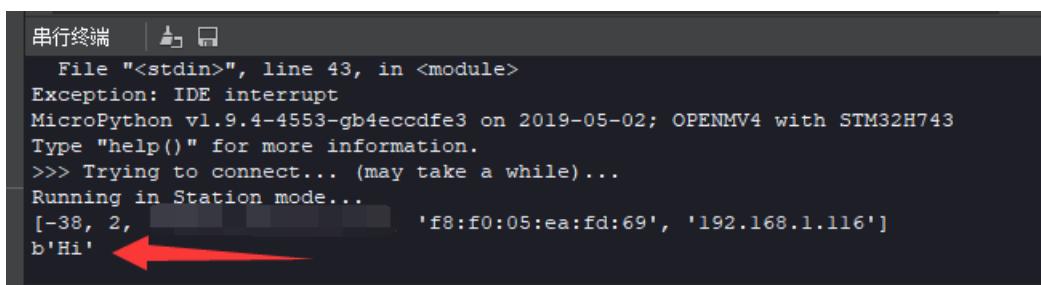


图 6-30

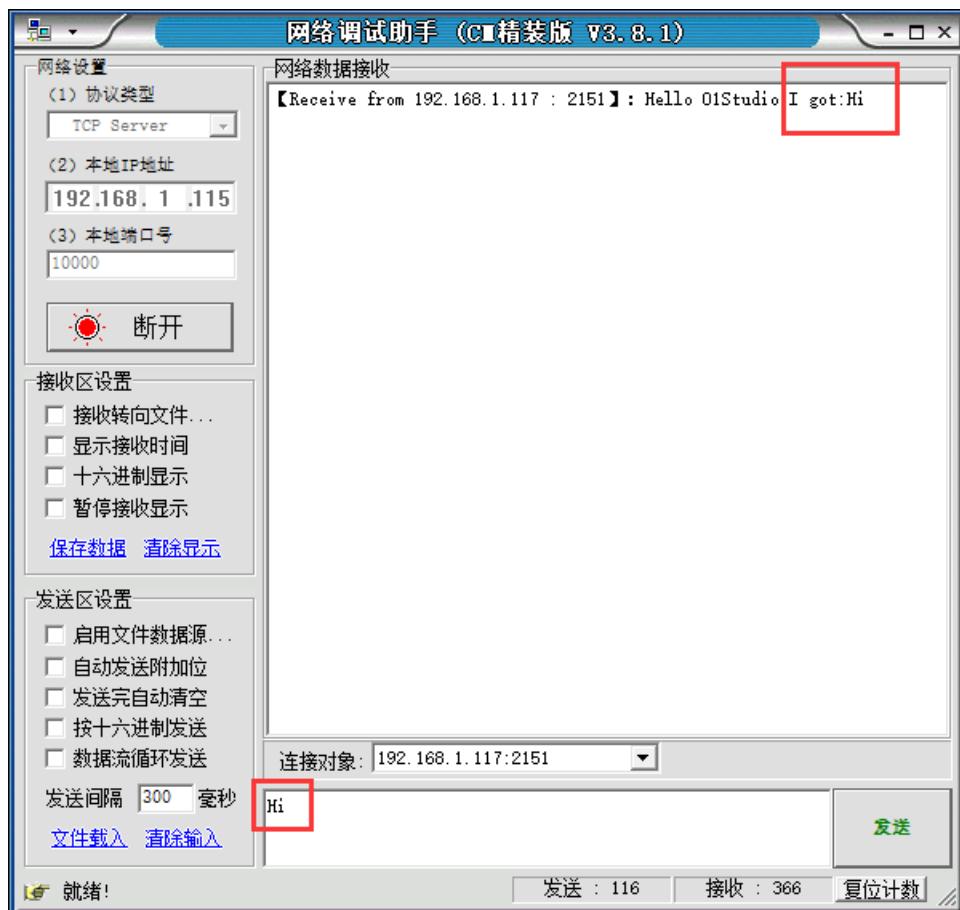


图 6-31 服务器发送数据

### ● 总结：

通过本节学习，我们了解了 `socket` 通信原理以及使用 MicroPython 进行 `socket` 编程并且通信的实验。得益于优秀的封装，让我们可以直接面向 `socket` 对象编程就可以快速实现 `socket` 通信，从而开发更多的网络应用，例如将前面采集到的传感器数据发送到服务器。

### 6.4.3 MQTT 通信

- 前言：

上一节，我们学习了 Socket 通信，当服务器和客户端建立起连接时，就可以相互通信了。在互联网应用大多使用 WebSocket 接口来传输数据。而在物联网应用中，常常出现这样的情况：海量的传感器，需要时刻保持在线，传输数据量非常低，有着大量用户使用。如果仍然使用 socket 作为通信，那么服务器的压力和通讯框架的设计随着数量的上升将变得异常复杂！

那么有无一个框架协议来解决这个问题呢，答案是有的。那就是 MQTT(消息队列遥测传输)。

- 实验平台：

pyAI-OpenMV4 和 OpenMV WiFi 模块。(如果使用 pyBase 需要将 OLED 拔掉。避免 IO 冲突)

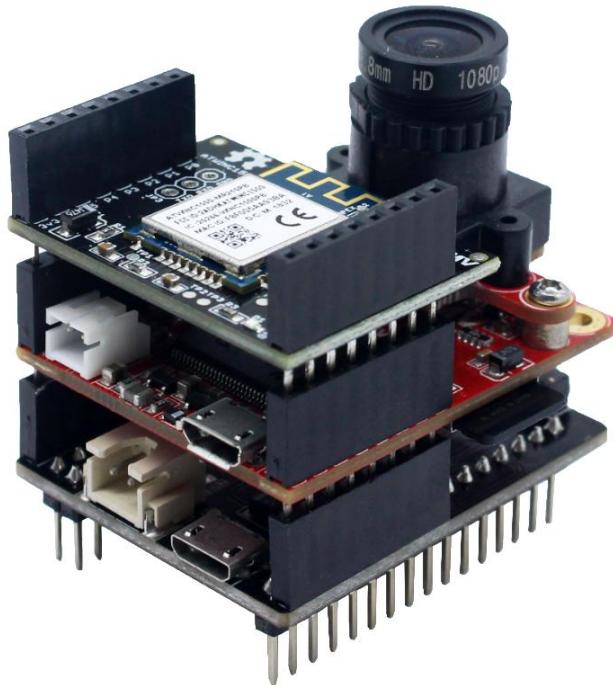


图 6-32 pyAI-OpenMV4

- 实验目的：

通过编程让 pyAI-OpenMV4 实现 MQTT 协议信息的发布和订阅（接收）。

### ● 实验讲解：

MQTT 是 IBM 于 1999 年提出的，和 HTTP 一样属于应用层，它工作在 TCP/IP 协议族上，通常还会调用 socket 接口。是一个基于客户端-服务器的消息发布/订阅传输协议。其特点是协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。

总结下来 MQTT 有如下特性/优势：

- 异步消息协议
- 面向长连接
- 双向数据传输
- 协议轻量级
- 被动数据获取

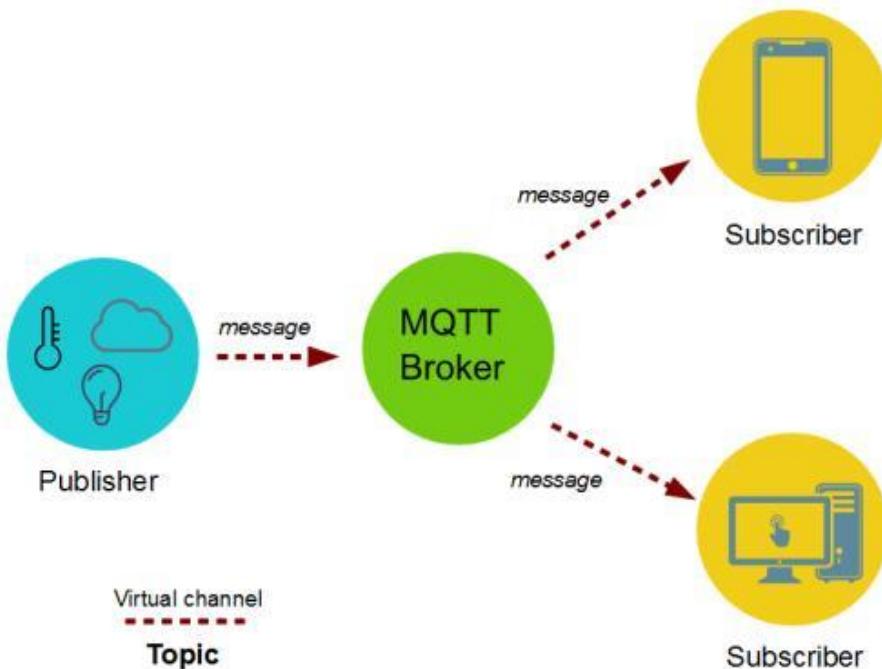


图 6-33 MQTT 通信流程

从上图可以看到，MQTT 通信的角色有两个，分别是服务器和客户端。服务器只负责中转数据，不做存储；客户端可以是信息发送者或订阅者，也可以同时是两者。具体如下图：

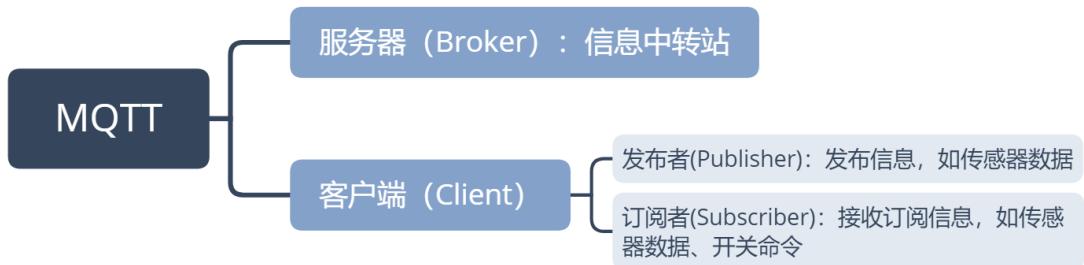


图 6-34 MQTT 角色说明

确定了角色后是如何传输数据呢？下表示 MQTT 最基本的数据帧格式，例如温度传感器发布主题“Temperature”编号,消息是“25”（表示温度）。那么所有订阅了这个主题编号的客户端（手机应用）就会收到相关信息，从而实现通信。如下表所示：

MQTT 数据帧格式	
Topic ID (主题编号)	Message (消息)
Temperature	25

图 6-35 MQTT 数据格式

由于特殊的发布/订阅机制，服务器不需要存储数据（当然也可以在服务器的设备上建立一个客户端来订阅保存信息），因此非常适合海量设备的传输。

人生苦短，而 MicroPython 已经封装好了 MQTT 客户端的库文件。让我们的应用变得简单美妙。OpenMV 的 MQTT 模块文件为例程文件夹里面的 `mqtt.py` 文件。使用方法如下：

构造函数
<code>client= mqtt.MQTTClient(client_id, server, port)</code>
构建 MQTT 客户端对象。
<code>client_id</code> : 客户端 ID，具有唯一性；
<code>server</code> : 服务器地址，可以是 IP 或者网址；
<code>port</code> : 服务器端口。（默认是 1883，服务器通常采用的端口，也可以自定义。）
使用方法
<code>client.connect()</code>

连接到服务器。
<code>client.publish(TOPIC,message)</code>
发布。TOPIC: 主题编号; message: 信息内容, 例: 'Hello 01Studio!'
<code>client.subscribe(TOPIC)</code>
订阅。TOPIC: 主题编号。
<code>client.set_callback(callback)</code>
设置回调函数。callback: 订阅后如果接收到信息, 就执行相名称的回调函数。
<code>client.check_msg()</code>
检查订阅信息。如收到信息就执行设置过的回调函数 callback。

表 6-8 MQTT 客户端对象

由于客户端分为发布者和订阅者角色, 因此为了方便大家更好理解, 本实验分开两个案例来编程, 分别为发布者和订阅者。再结合 MQTT 网络调试助手来测试。代表编写流程图如下:

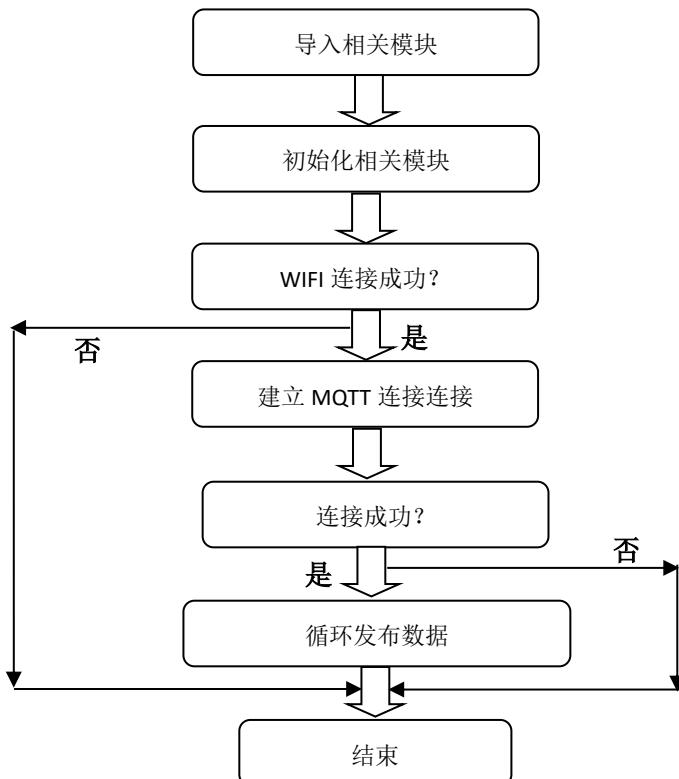


图 6-36 发布者代码编写流程

导入相关模块

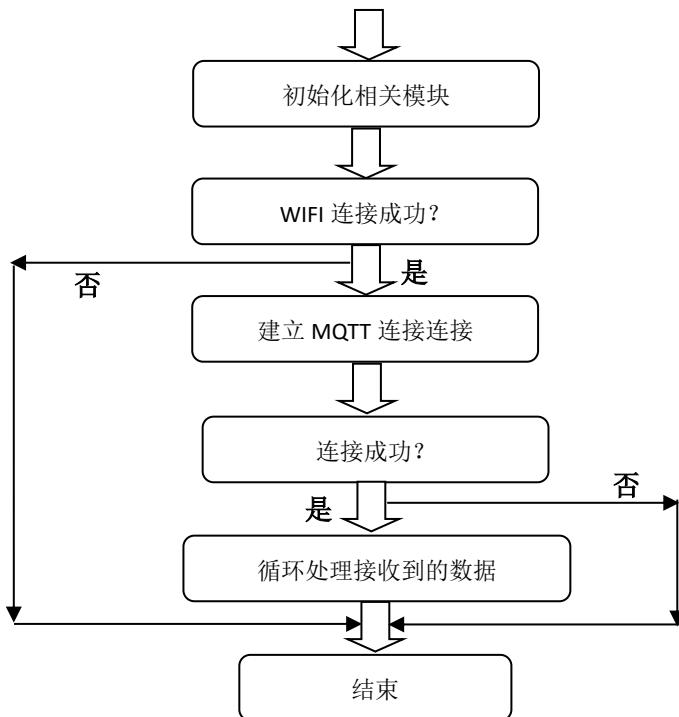


图 6-37 订阅者代码编写流程

发布者（publish）参考代码：

```

# MQTT 发布者（publish）例程.

#
#需要复制 mqtt.py 文件到 OpenMV 文件系统.
#
#翻译和注释: 01Studio

import time, network
from mqtt import MQTTClient

SSID='WeBee_office_2.4G' # Network SSID
KEY='webee0123456789' # Network key

#
# Init wlan module and connect to network
print("Trying to connect... (may take a while)...")

```

```

wlan = network.WINC()

wlan.connect(SSID, key=KEY, security=wlan.WPA_PSK)

# We should have a valid IP now via DHCP
print(wlan.ifconfig())


SERVER = 'mqtt.p2hp.com'
PORT = 1883
CLIENT_ID = '01Studio-OpenMV' # 客户端 ID
TOPIC = 'OpenMV/test' # TOPIC 名称


client = MQTTClient(CLIENT_ID, SERVER, PORT)
client.connect()


while (True):
    client.publish(TOPIC, "Hello 01Studio!") #发布消息
    time.sleep_ms(1000)

```

订阅者（subscribe）参考代码：

```

# MQTT 订阅者（subscribe）例程.

#
#需要复制 mqtt.py 文件到 OpenMV 文件系统.
#
#翻译和注释: 01Studio


import time, network
from mqtt import MQTTClient

SSID='WeBee_office_2.4G' # Network SSID

```

```

KEY='webee0123456789' # Network key

# Init wlan module and connect to network
print("Trying to connect... (may take a while)...")

wlan = network.WINC()
wlan.connect(SSID, key=KEY, security=wlan.WPA_PSK)

# We should have a valid IP now via DHCP
print(wlan.ifconfig())

#设置 MQTT 回调函数,有信息时候执行
def MQTT_callback(topic, msg):
    print('topic: {}'.format(topic))
    print('msg: {}'.format(msg))

SERVER = 'mqtt.p2hp.com'
PORT = 1883
CLIENT_ID = '01Studio-OpenMV' # 客户端 ID
TOPIC = 'OpenMV/test' # TOPIC 名称

client = MQTTClient(CLIENT_ID, SERVER, PORT)
client.set_callback(MQTT_callback) #配置回调函数
client.connect()
client.subscribe(TOPIC) #订阅主题

while (True):
    client.check_msg() #检测是否收到信息, 收到则执行打印。
    time.sleep_ms(300)

```

从以上代码可以看到发布者和订阅者的编程方式相近，另外本实验需要一个 MQTT 服务器（Broker），这里使用的是跟我们将用到的 MQTT 在线网络助手同一个服务器和端口。

```
SERVER = 'mqtt.p2hp.com'  
PORT = 1883
```

### ● 实验结果：

我们将 O1Studio 示例程序中的 mqtt.py 文件拷贝到文件系统，然后在 OpenMV IDE 分别运行上述代码。

OpenMV IDE 可能有个 BUG，就是运行时候发现 mqtt.py 文件跟文件系统不一致，弹出提示。我们要选 No 忽略它。

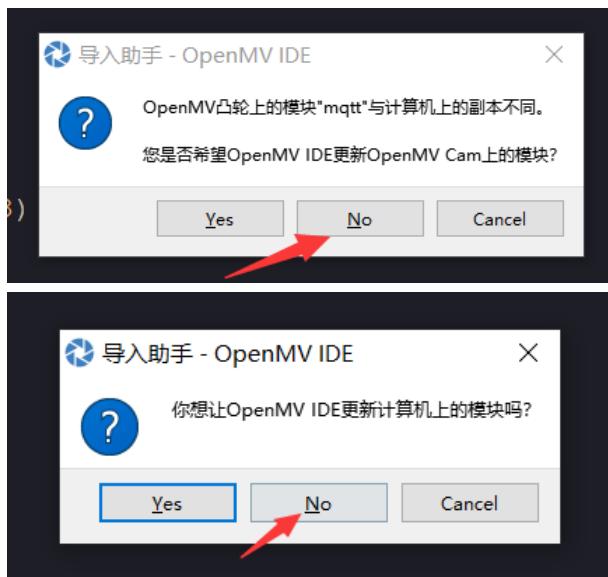


图 6-38 选择 No 忽略

为了方便测试，我们可以使用 MQTT 网络助手进行调试。这里推荐一个在线 MQTT 网络调试助手：<http://mqtt.p2hp.com/websocket/>

打开上面网址，即可看到 MQTT 在线调试助手。可以配置基本信息，这里默认即可，点击连接。

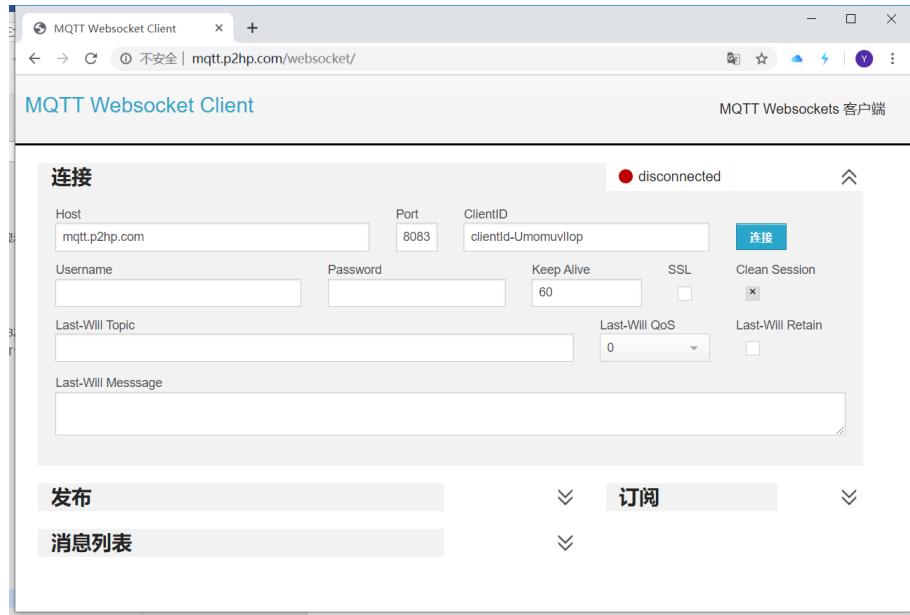


图 6-39 MQTT 助手

连接成功可以看到显示 Connected。

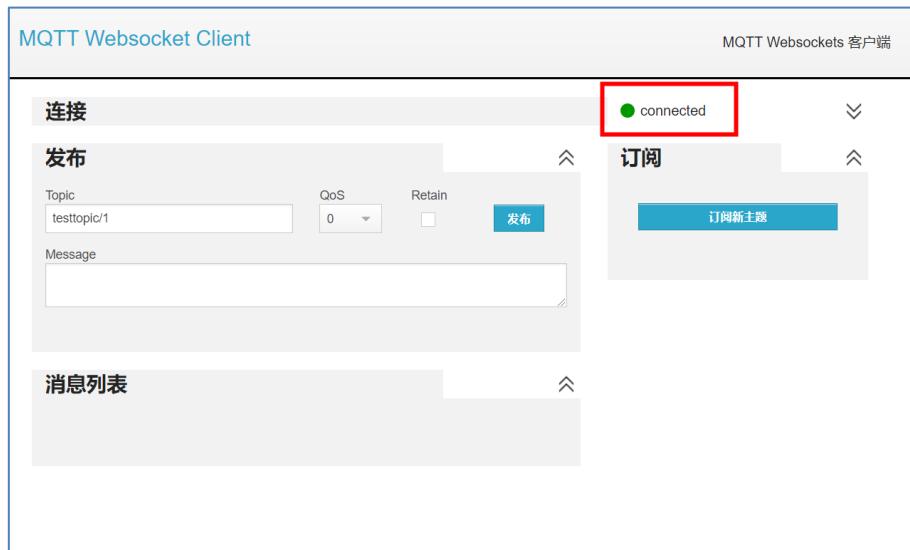


图 6-40 MQTT 助手连接成功

### 测试“发布者”代码：

我们先测试“发布者”代码。将“发布者”代码下载到开发板，然后在 MQTT 助手中订阅主题修改为：'/public/01Studio/1'（跟代码发布的主题一致），QOS 选择 0 即可。然后点击订阅主题。

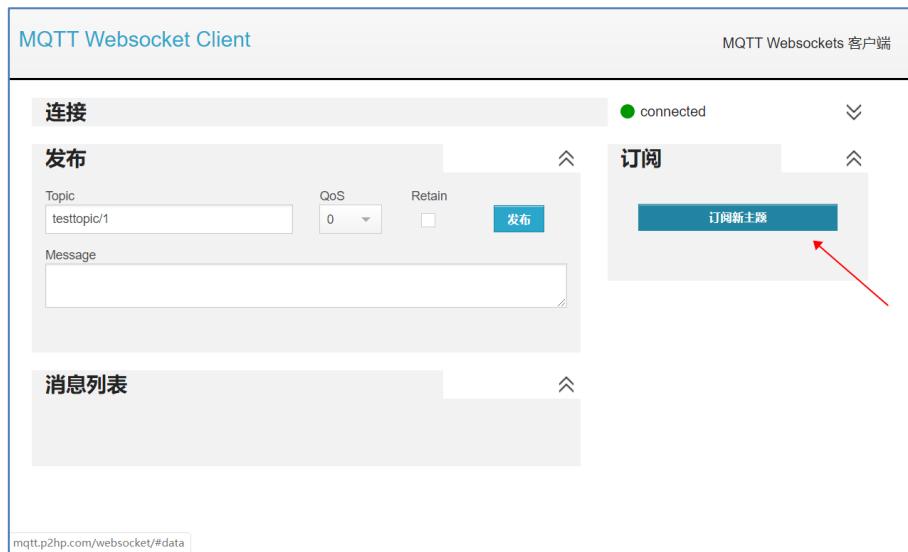


图 6-41 订阅主题

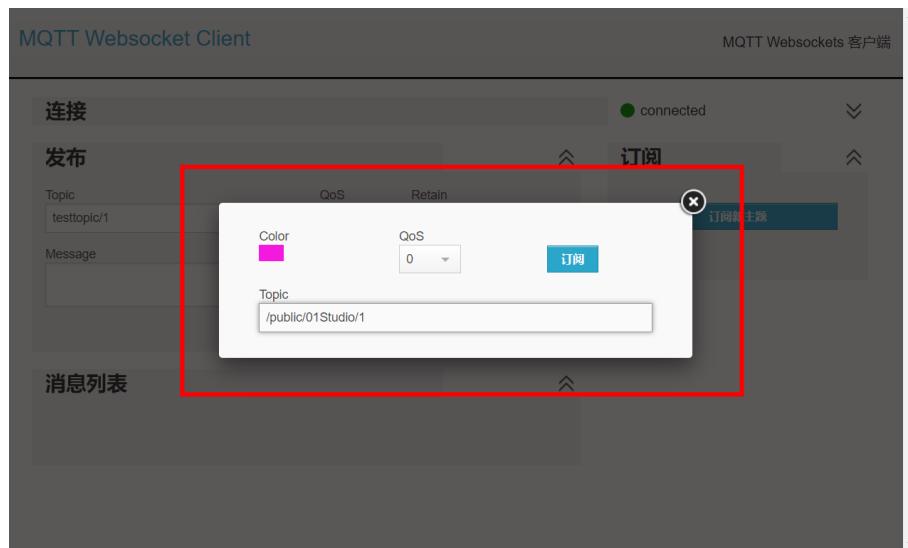


图 6-42

订阅后可以看到接收框收到来自开发板发布的信息。

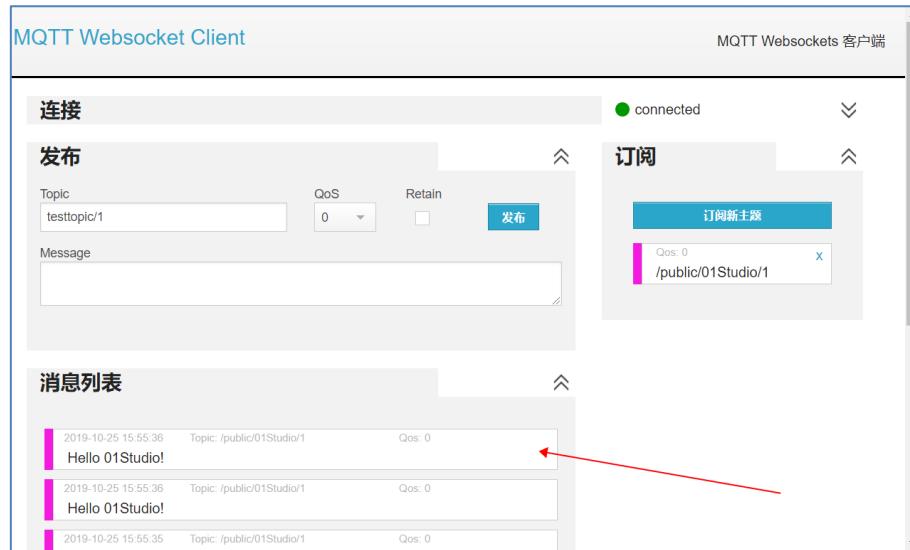


图 6-43 MQTT 助手收到开发板发布的信息

### 测试“订阅者”代码：

“订阅者”代码测试方法跟“发布者”相反。将“订阅者”代码下载到开发板，然后在电脑 MQTT 助手中发布主题修改为：'/public/01Studio/1'（跟代码发布的主题一致。）在下方空白框输入“Hello 01Studio!”。

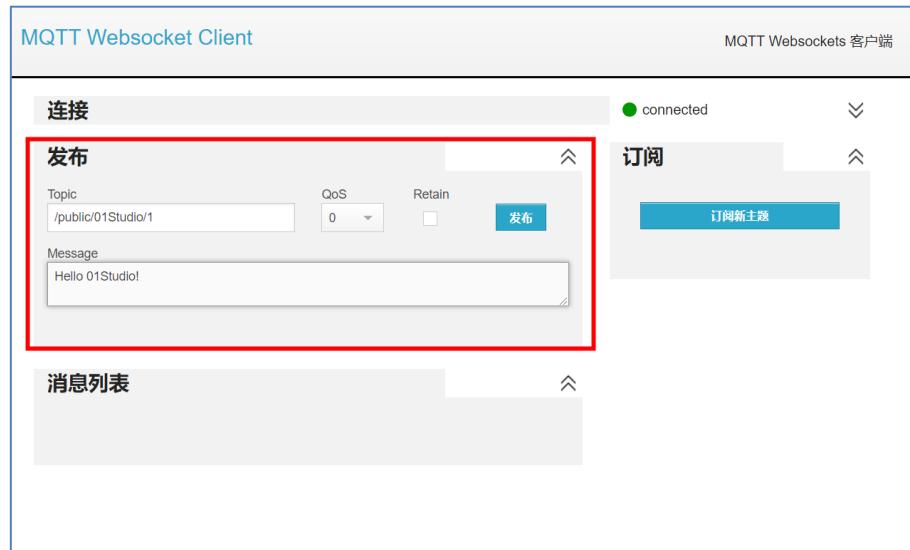
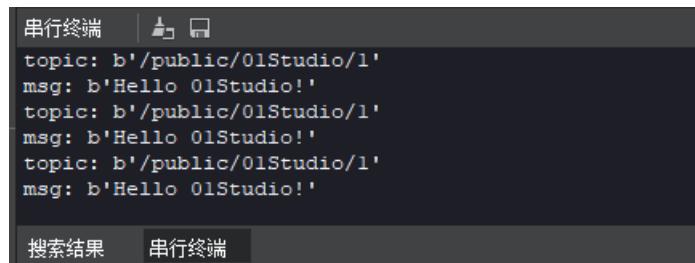


图 6-44 输入即将发布的主题和信息

将“订阅者”代码下载到开发板，成功连接后回到 MQTT 助手点击【发布】

发布信息，可以在开发板的 REPL 看到接收到的订阅信息“Hello 01Studio!”被打印出来了（数据格式为字节数据）。



```
串行终端 | □ □
topic: b'/public/01Studio/1'
msg: b'Hello 01Studio!'
topic: b'/public/01Studio/1'
msg: b'Hello 01Studio!'
topic: b'/public/01Studio/1'
msg: b'Hello 01Studio!'
```

图 6-45 开发板接收到相关信息并打印

当使用 main.py 脱机运行时候，IDE 的串口终端和 putty 并不能打印信息。可以使用串口助手观察。波特率为 115200。

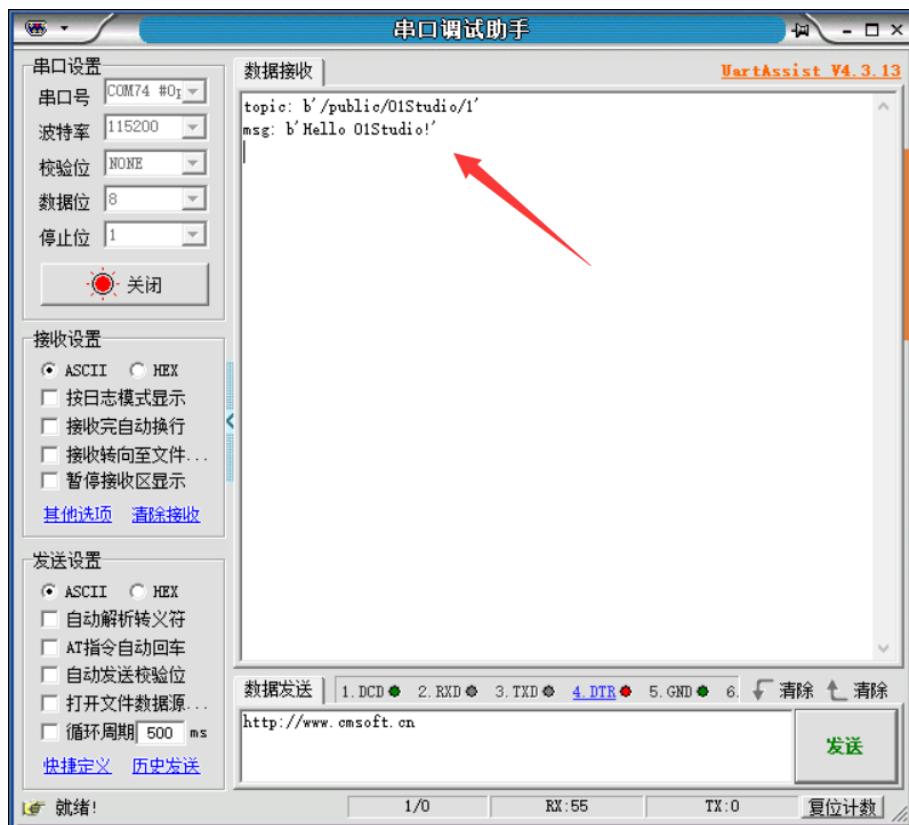


图 6-46 串口调试助手

当然你也可以在同一个 MQTT 在线助手下测试发布和订阅功能来做一些测试，只需要将主题设置一致即可，如下图所示：

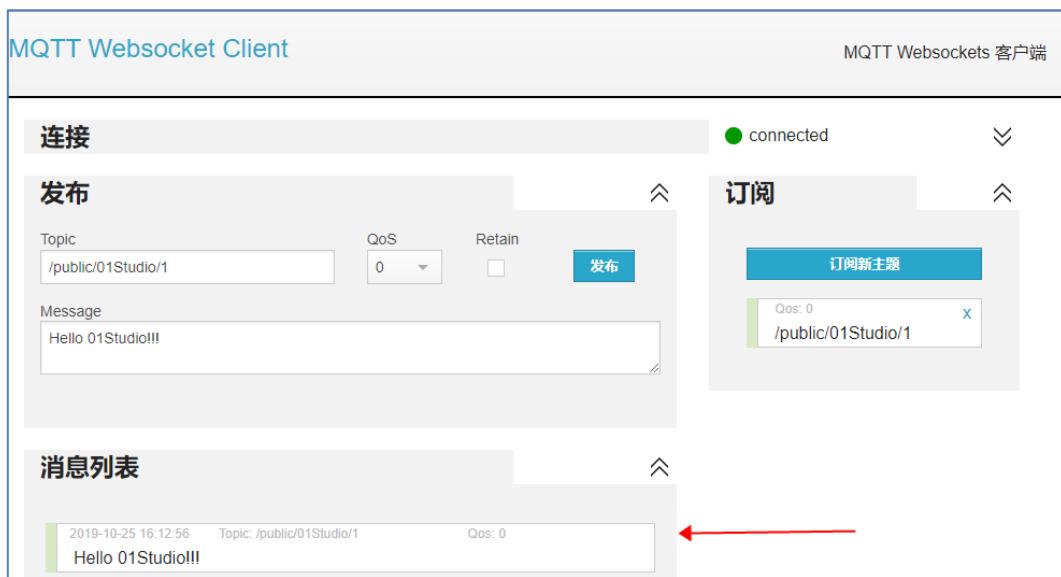


图 6-47 客户端同时发布和订阅信息

### ● 总结：

通过本节我们了解了 MQTT 通信原理以及成功实现通信。目前市面上大部分物联网云平台支持 MQTT，原理大同小异。大家可以基于不同平台协议来开发，实现自己的物联网设备远程连接。

#### 6.4.4 无线视频传输

- 前言：

pyAI-OpenMV4 配合 WiFi 模块能实现浏览器视频传输，本节只讲述实验方法，关于原理和代码用户可以自行研究。

- 实验平台：

pyAI-OpenMV4 和 OpenMV WiFi 模块。(如果使用 pyBase 需要将 OLED 拔掉。避免 IO 冲突)。另外最好配上锂电池，保证开发板供电的稳定性。

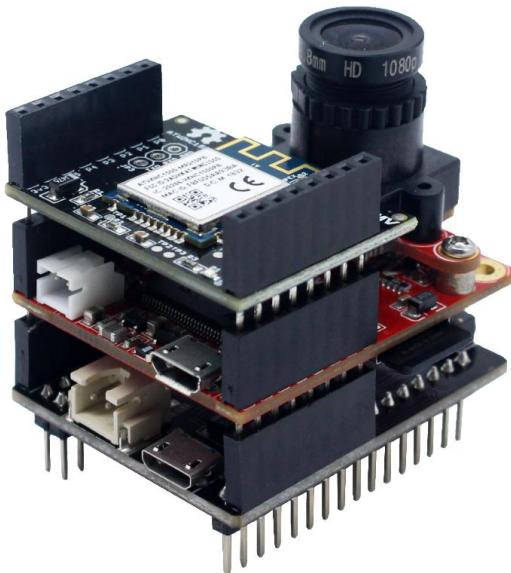


图 6-48 pyAI-OpenMV4

- 实验说明：

我们打开官方例程【示例→WiFi-Shield→mjpeg\_streamer.py】，具体如下：

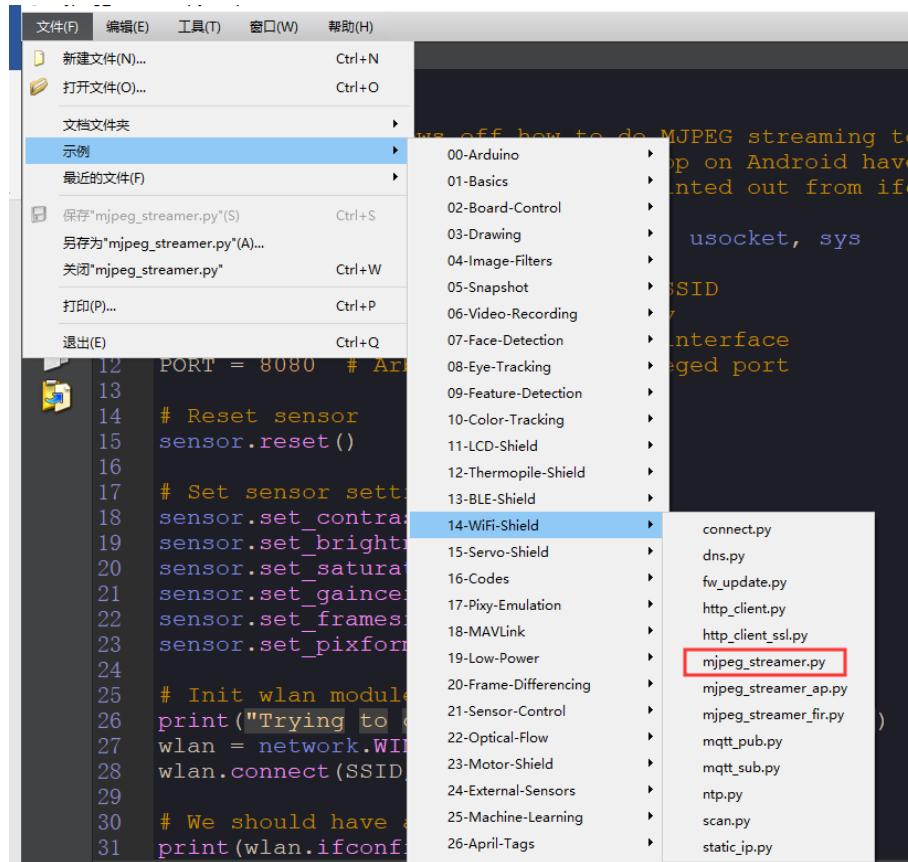


图 6-49 视频传输例程

我们只需要修改自己的无线网络,端口默认 8080 即可:

```
SSID='01Studio' # Network SSID
KEY='88888888' # Network key
HOST = ''      # Use first available interface
PORT = 8080 # Arbitrary non-privileged port
```

运行程序，可以看到串口中断打印出当前 OpenMV 设备的连接信息：

```
MicroPython v1.11-omv OpenMV v3.5.0 2019-11-04; OPENMV4-STM32H743
Type "help()" for more information.
>>> Trying to connect... (may take a while)...
Running in Station mode...
('192.168.1.117', '255.255.255.0', '192.168.1.1', '192.168.1.1') ↗
```

图 6-50 OpenMV4 WiFi 连接信息

可以看到 IP 是 192.168.1.117，我们在同一网络的电脑或者手机浏览器下打开 **192.168.1.117:8080** 这个地址，可以看到连接成功后出现图像。（建议使用谷歌浏览器。）

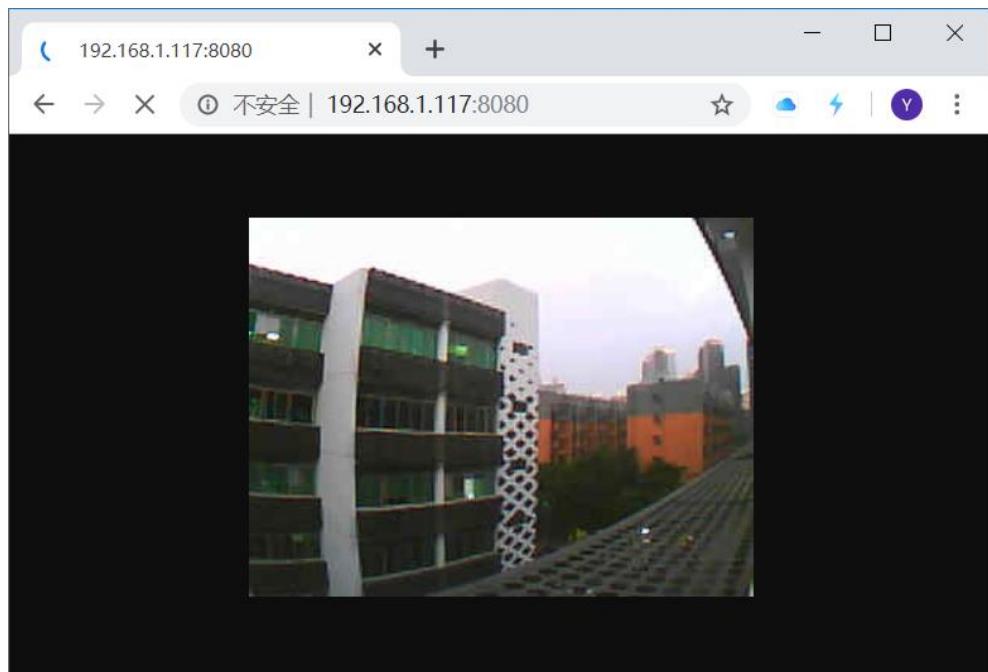


图 6-51 PC 浏览器

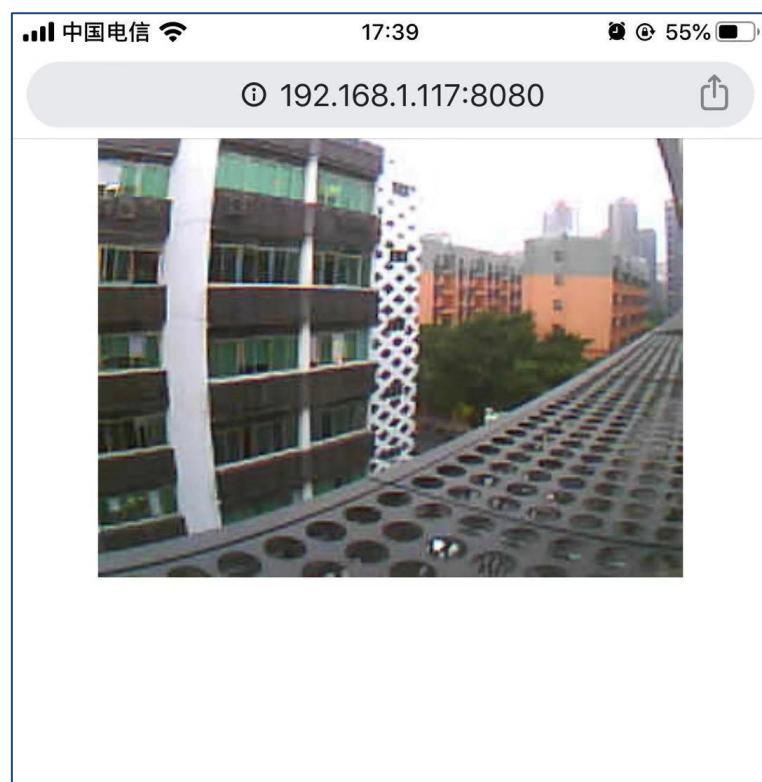


图 6-52 手机浏览器

## 6.5 摄像头模块

OpenMV 更换摄像头模块非常简单。可以将整个摄像头模块拆下更换。默认的摄像头模块是 OV7725 卷帘快门。

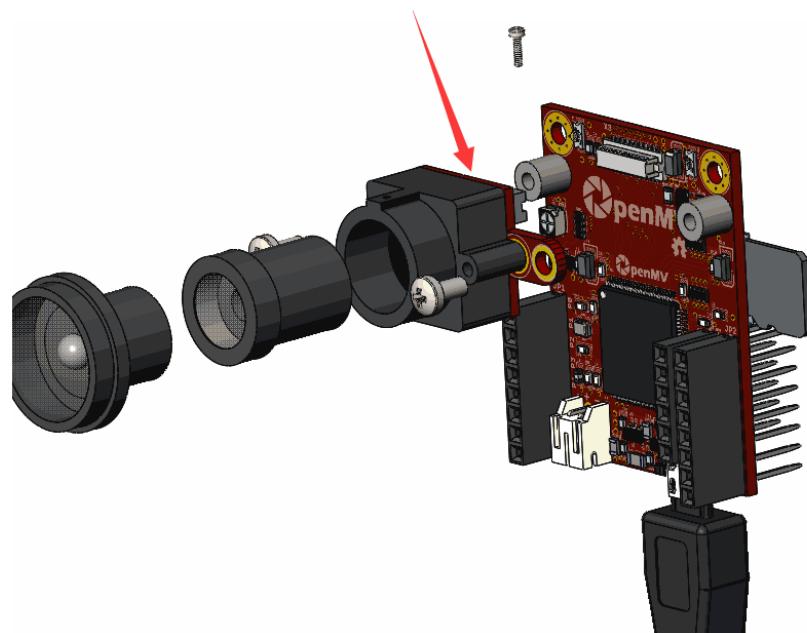


图 6-53

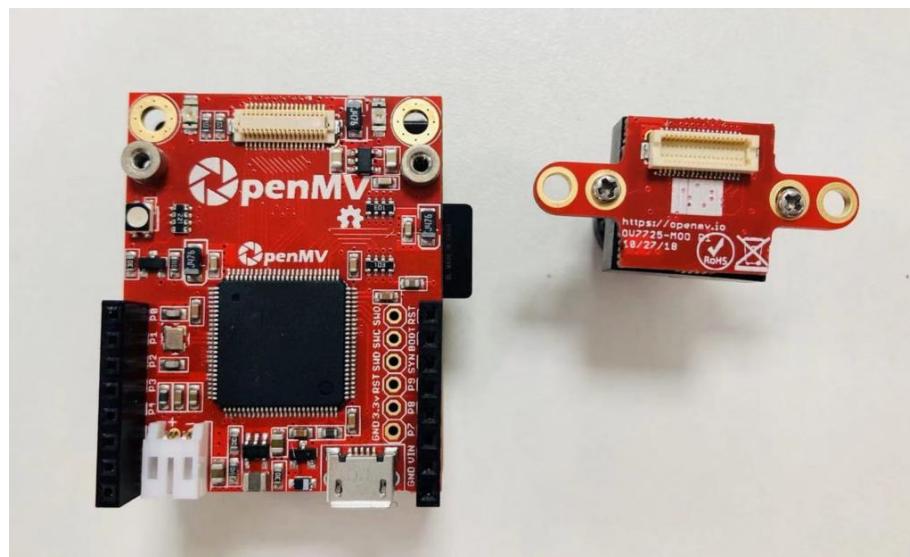


图 6-54 更换摄像头

### 6.5.1 MT9V034 全局快门

- 前言：

MT9V034 全局快门适用于高速运动状态拍摄，支持工业高速运动的应用。每秒帧率相对于 OV7725 卷帘快门大大提升，仅支持灰度图像拍摄。

- 实验平台：

pyAI-OpenMV4 开发板+MT9V034 全局快门模块。



图 6-55 pyAI-OpenMV4

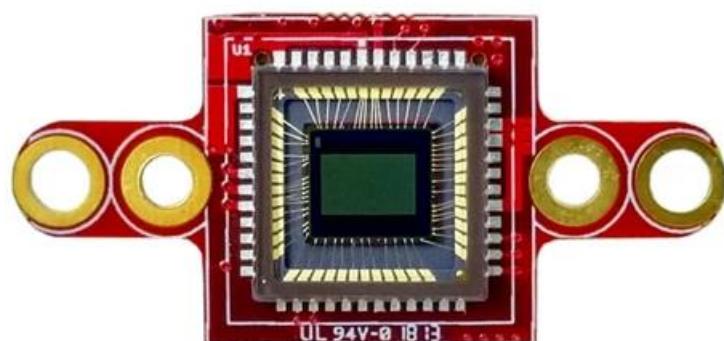


图 6-56 MT9V034 全局快门模组

- **实验目的:**

测试 MT9V034 全局快门在不同图像尺寸下的 FPS (每秒帧率) 变化。

- **实验讲解:**

全局快门模块功能参数如下表:

功能参数	
型号	MT9V034
分辨率	752*480 像素
像素间距	6um
重量	10g
工作温度	-30℃至 70℃
使用场景	高速移动物体拍摄

表 6-9 功能参数

我们先来看看卷帘快门和全局快门的曝光方式和原理:

**卷帘快门曝光方式:**





图 6-57 卷帘快门曝光

全局快门曝光方式：



图 6-58 全局快门曝光

从上图可以知道，卷帘快门的曝光方式决定了它在拍摄高速运动物体会让图像出现变形，而全局快门很好解决了这个问题。以下是 MT 摄像头模组官方材料说明，分别是肉眼、卷帘快门和全局快门的拍摄效果。

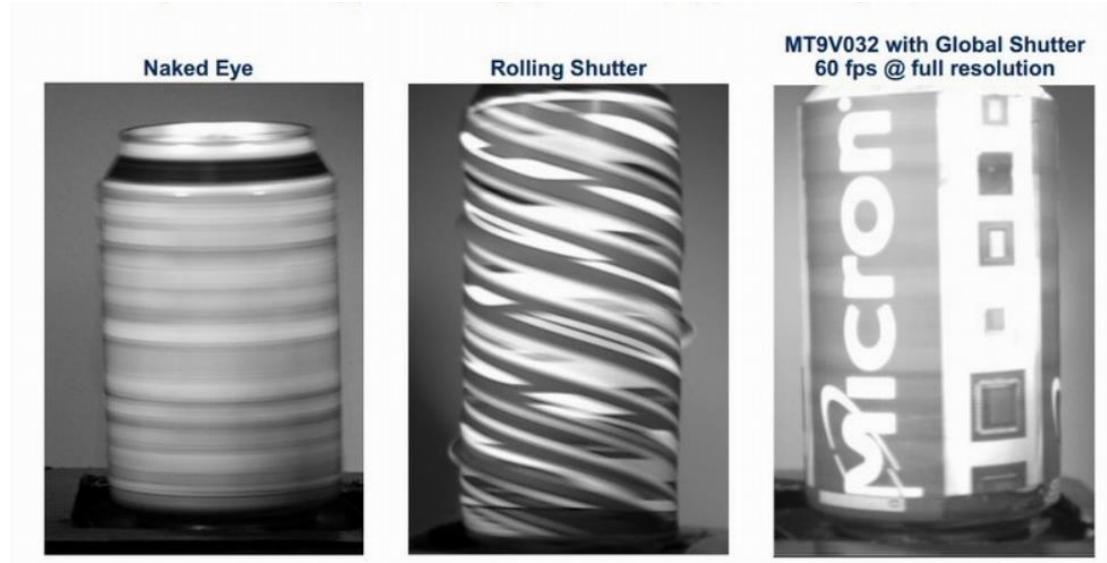


图 6-59 MT9V034 官方效果图 (肉眼、卷帘快门、全局快门)

OpenMV 官方提供了全局快门高速 FPS (每秒帧率) 的测试例程，参考代码路径【示例→Global Shutter→ high\_fps.py】，具体如下：

```
# 高 FPS(每秒帧率)例程
#
#这个例程展示了全局快门的高帧率。你可以通过改变图像像素和曝光时间来调整帧率。
#
# 图像大小从 VGA 到 QVGA，FPS 提升 2 倍，从 QVGA 到 QQVGA，FPS 再提升 2 倍。
#
# 除此之前你也可以降低曝光时间来提供 FPS，但这样拍摄出来的图像会非常暗。因此你
# 需要额外补充光源。
#
#翻译和注释：01Studio

import sensor, image, time
```

```

#摄像头初始化

sensor.reset()                      # Reset and initialize the sensor.

sensor.set_pixformat(sensor.GRAYSCALE) # Set pixel format to GRayscale

sensor.set_framesize(sensor.QQVGA)    # 可以测试 VGA/QVGA/QQVG 对比 FPS 效果

sensor.skip_frames(time = 2000)        # Wait for settings take effect.

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

sensor.set_auto_exposure(True, exposure_us=5000)

# 曝光时间，使用小的值提高速度。

while(True):

    clock.tick()                      # Update the FPS clock.

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

    print(clock.fps())              # 使用 IDE 运行会降低 FPS。

```

### ● 实验结果：

我们将以下代码的图像尺寸分别改成 VGA(640X480)、QVGA(320\*240)和 QQVGA(160X120)。在串口中观察 FPS 变化。

```
sensor.set_framesize(sensor.QQVGA) # 可以测试 VGA/QVGA/QQVG 对比 FPS 效果
```

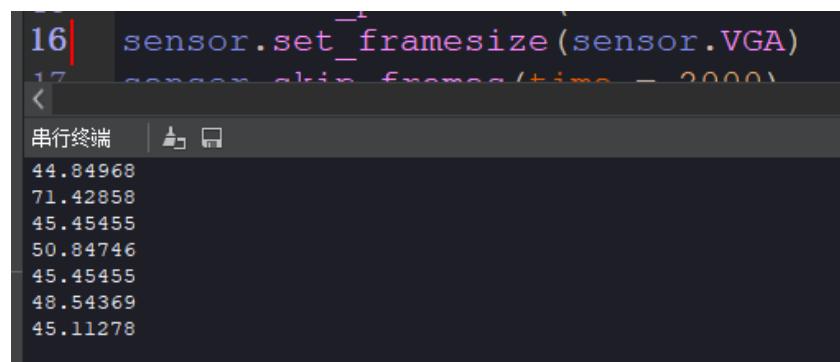
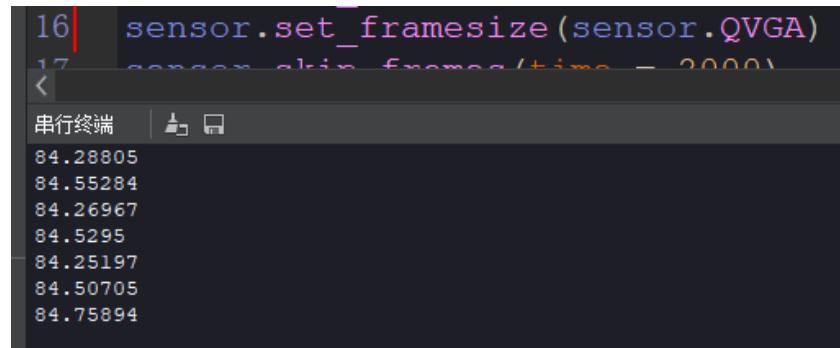
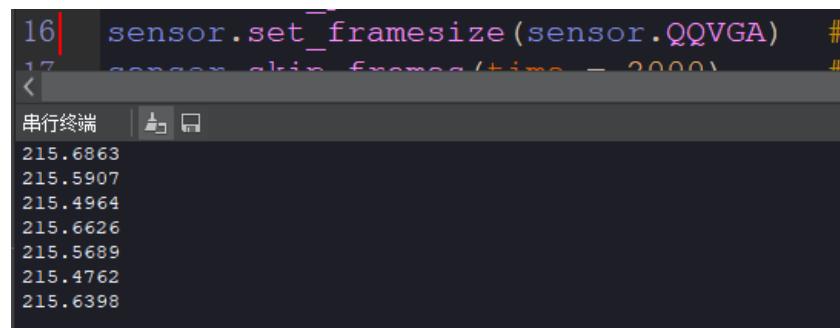


图 6-60 VGA 下的 FPS



```
16| sensor.set_framesize(sensor.QVGA) #  
17| sensor.shutter_time = 2000  
<  
串行终端 |    
84.28805  
84.55284  
84.26967  
84.5295  
84.25197  
84.50705  
84.75894
```

图 6-61 QVGA 下的 FPS



```
16| sensor.set_framesize(sensor.QQVGA) #  
17| sensor.shutter_time = 2000  
<  
串行终端 |    
215.6863  
215.5907  
215.4964  
215.6626  
215.5689  
215.4762  
215.6398
```

图 6-62 QQVGA 下的 FPS

### ● 实验总结：

通过本实验，可以看到全局快门在不同图像尺寸下的 **FPS**(每秒帧率)变化，图像越小，FPS 越高。全局快门在用于曝光时间短的情景下，可以增加拍摄物的光线（补光）以获取更好的拍摄效果。

## 6.5.2 FLIR Lepton 红外热成像

- 前言：

FLIR Lepton 红外热成像摄像头模块适用于识别人和动物、夜晚视觉、检测温度热量以及工业检测故障等应用，属于军工级产品。本节只讲述测试方法，具体代码内容用户可以自行深入研究。

- 实验平台：

pyAI-OpenMV4 开发板+FLIR Lepton 3.5 红外热成像模块。

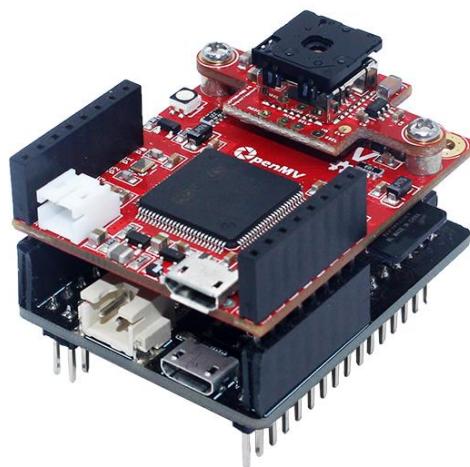


图 6-63 pyAI-OpenMV4

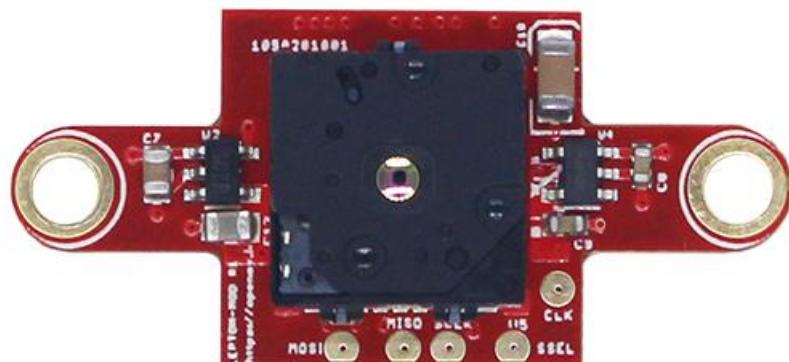


图 6-64 FLIR Lepton 3.5(160\*120) 红外热成像模组

- 实验目的:

测试 FLIR Lepton 3.5(160\*120) 红外热成像模组。

- 实验讲解:

热成像的工作原理是什么?

通俗地讲热像仪就是将物体发出的不可见红外能量转变为可见的热图像。热图像的上面的不同颜色代表被测物体的不同温度。通过查看热图像，可以观察到被测目标的整体温度分布状况，研究目标的发热情况，从而进行下一步工作的判断。现代热像仪的工作原理是使用光电设备来检测和测量辐射，并在辐射与表面温度之间建立相互联系。所有高于绝对零度 (-273°C) 的物体都会发出红外辐射。热像仪利用红外探测器和光学成像物镜接受被测目标的红外辐射能量分布图形反映到红外探测器的光敏元件上，从而获得红外热像图，这种热像图与物体表面的热分布场相对应。

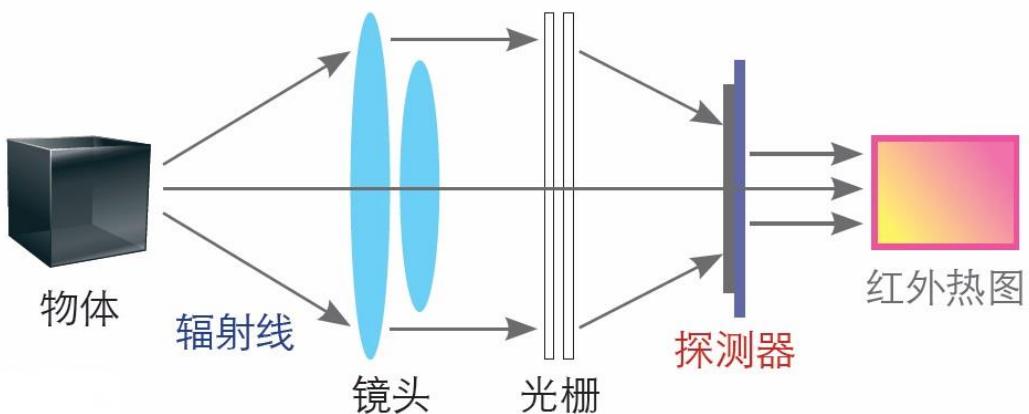


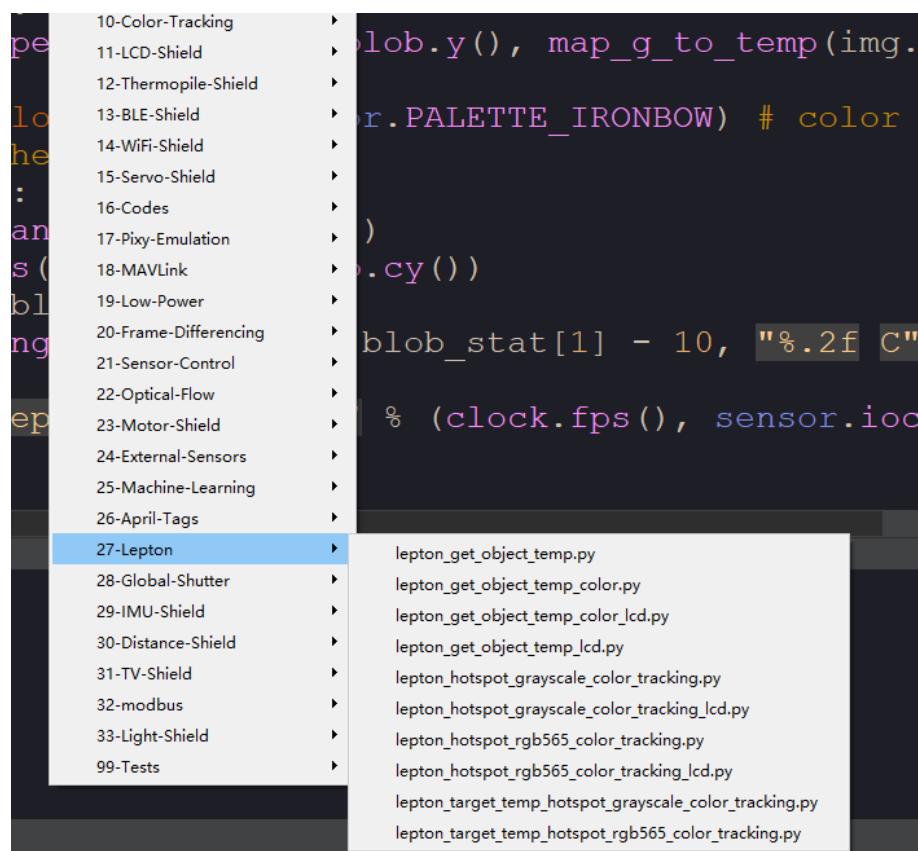
图 6-65 热成像工作原理

我们再来看看 Flir lepton 3.5 模块功能参数:

功能参数	
型号	FLIR Lepton 3.5
分辨率	160*120 像素
像素间距	12um
波长范围	8um – 14um
视场角	对角线 71°， 水平 57°
其它功能	温度检测
适用场景	识别任何动物、夜晚视觉、检查温度热量及工业检测故障等。

表 6-10 功能参数

OpenMV 官方提供了非常全面的 Lepton 红外热成像模块测试例程，位于【示例—Lepton】下，带 lcd 的例程表示可以同时在 1.77 寸 LCD 液晶屏显示。



```

pe          10-Color-Tracking
lo          11-LCD-Shield
he          12-Thermopile-Shield
:
an          13-BLE-Shield
bl          14-WiFi-Shield
ng          15-Servo-Shield
ep          16-Codes
          17-Pixy-Emulation
          18-MAVLink
          19-Low-Power
          20-Frame-Differencing
          21-Sensor-Control
          22-Optical-Flow
          23-Motor-Shield
          24-External-Sensors
          25-Machine-Learning
          26-April-Tags
          27-Lepton
          28-Global-Shutter
          29-IMU-Shield
          30-Distance-Shield
          31-TV-Shield
          32-modbus
          33-Light-Shield
          99-Tests
          lepton_get_object_temp.py
          lepton_get_object_temp_color.py
          lepton_get_object_temp_color_lcd.py
          lepton_get_object_temp_lcd.py
          lepton_hotspot_grayscale_color_tracking.py
          lepton_hotspot_grayscale_color_tracking_lcd.py
          lepton_hotspot_rgb565_color_tracking.py
          lepton_hotspot_rgb565_color_tracking_lcd.py
          lepton_target_temp_hotspot_grayscale_color_tracking.py
          lepton_target_temp_hotspot_rgb565_color_tracking.py

```

图 6-66 Lepton 测试例程

- 实验结果：

Lepton 红外热成像具备了测温和成像功能。以下是部分实验测试图片。

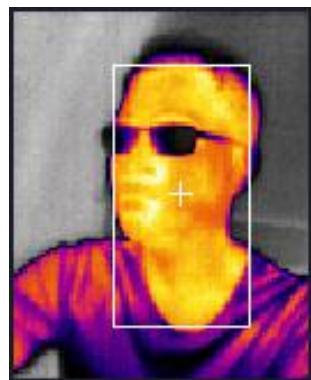


图 6-67

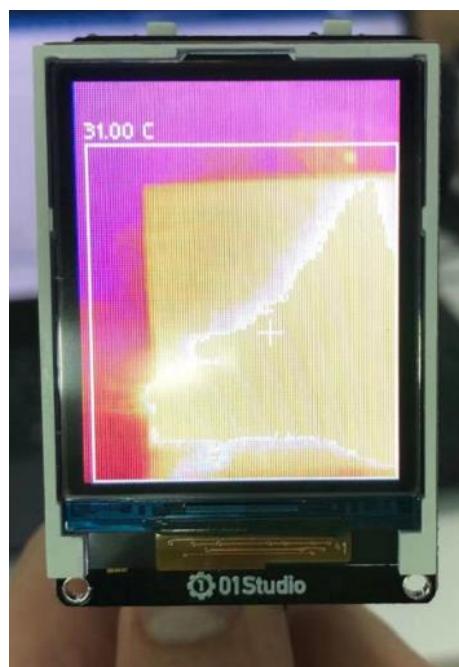


图 6-68

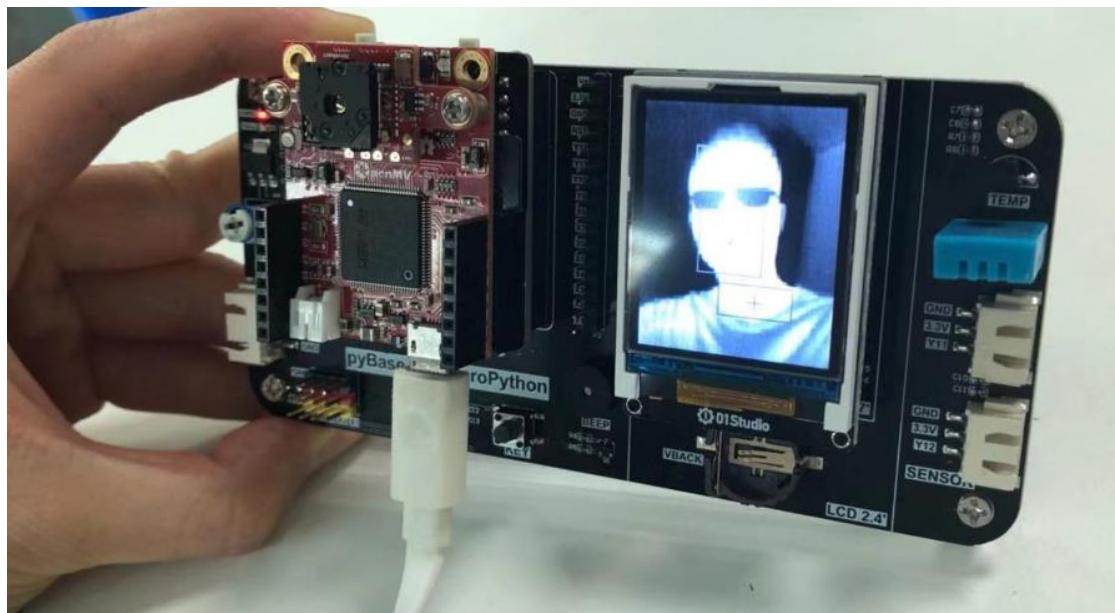


图 6-69

● 实验总结：

通过本实验，可以看到 OpenMV 让 Lepton 红外成像变得非常简单有趣，你可以结合 pyAI-OpenMV4 打造属于自己的红外热成像仪器。

## 6.6 摄像头镜头

OpenMV4 摄像头镜头更换非常简单，只需要拧下原来镜头，将新的镜头拧上并调好焦距即可。

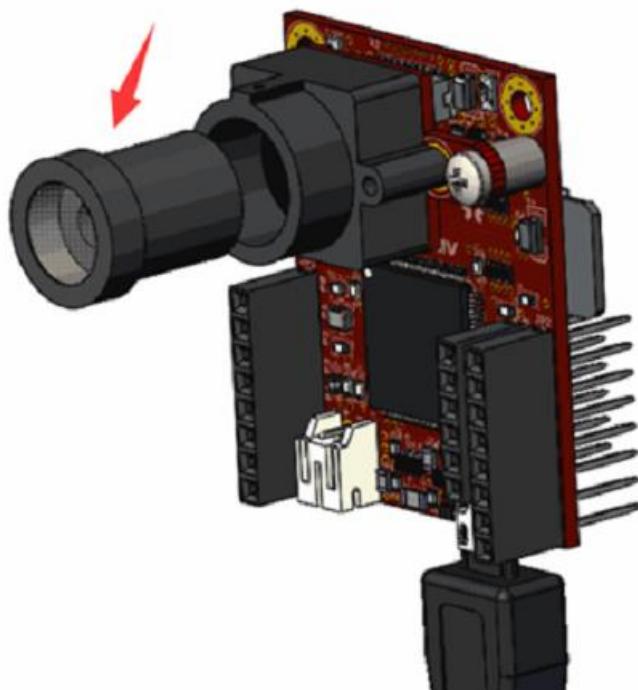


图 6-70

### 6.6.1 广角镜头

广角镜头是指可以让摄像头拍摄的范围变大，我们直接将广角镜头替换到OpenMV4上，通过旋转调好焦距即可使用。



图 6-71 广角镜头

功能参数	
焦距	1.7mm
像素	5mp (500 万)
视场角	1/2.5": 185°
接口	M12*0.5 mm
功能	广角拍摄
其它	含滤光片

表 6-11

以下是标准镜头和广角镜头拍摄对比。



图 6-72 标准镜头



图 6-73 广角镜头

## 6.6.2 长焦镜头

长焦镜头可以理解成是望远镜，是指可以让摄像头拍摄远景，我们直接将长焦镜头替换到 OpenMV4 上，通过旋转调好焦距即可使用。



图 6-74 长焦镜头

功能参数	
焦距	2.5mm
像素	300 万
视场角	D*H*V (1/3''): 13*11*8
畸变率	1/3'': -0.55%
功能	长焦（远景）拍摄
其它	含滤光片

表 6-12

以下是标准镜头和长焦镜头拍摄对比。



图 6-75 标准镜头



图 6-76 长焦镜头

### 6.6.3 无畸变镜头

标准镜头下的图片会出现一定的畸变（图像变形），而使用无畸变镜头可以解决这一问题，使拍摄出来的图片形状发生变形。我们直接将无畸变镜头替换到OpenMV4上，通过旋转调好焦距即可使用。



图 6-77 无畸变镜头

功能参数	
焦距	3.6mm
像素	500 万
视场角	D*H*V (1/2.5")： 96*81.8*65.8
畸变率	1/2.5"： <0.7% (无畸变)
光圈	F2.8
功能	无畸变拍摄
其它	含滤光片

表 6-13

以下是标准镜头和无畸变镜头拍摄对比。明细看到无畸变摄像头拍摄出来的图像两侧毫无变形。



图 6-78 标准镜头

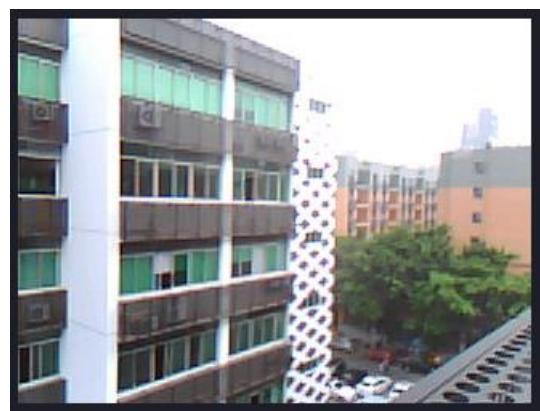


图 6-79 广角镜头

#### 6.6.4 手动调焦镜头

手动调焦镜头可以让摄像头通过手动调节来获取最佳拍摄效果。我们直接将手动调焦镜头替换到 OpenMV4 上，通过旋转调好焦距即可使用。



图 6-80 手动调焦镜头

功能参数	
焦距	2.8mm – 12mm (4 倍变焦)
像素	300 万
光圈类型	固定
功能	手动调焦拍摄
其它	含滤光片

表 6-14

以下是标准镜头和手动调焦镜头拍摄对比。下面对比图片看到差别不大，但手动调焦的优势是针对不同距离的拍摄都可以任意调焦到最清晰的位置，特别是近景。支持 2.8-12mm。



图 6-81 标准镜头



图 6-82 手动调节镜头

## 第7章 项目应用

### 7.1 照相机

- **前言：**

pyAI-OpenMV4 的外观非常酷，拥有按键功能，因此我们完全可以用来打造一台属于自己的照相机（尽管当前像素有点低）。现在无论数码相机还是手机都是带屏幕，主要是能实时显示方便拍摄，再配合锂电池，你就可以带上自制的照相机出去浪了！

- **实验平台：**

pyAI-OpenMV4 或 pyAI-OpenMV4 开发套件，需要使用 SD 卡和 LCD 显示屏。

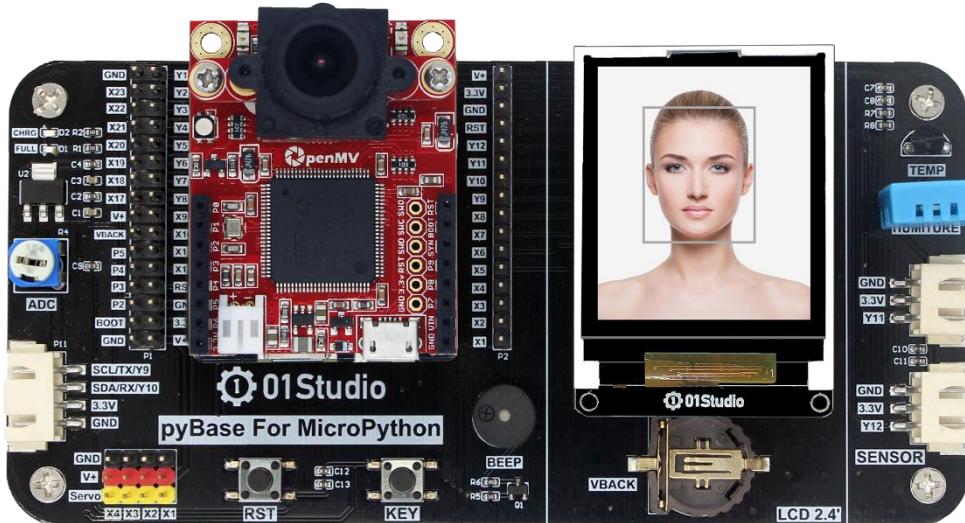


图 7-1 pyAI-OpenMV4 开发套件

- **实验目的：**

打造一台照相机，可以通过按键拍摄和 LCD 实时显示。

- **实验讲解：**

本项目主要是按键应用和拍照的相结合，这些内容可以在前面的实验找到，这里不再重复。

外部中断按键实验：请参阅 [4.4 外部中断](#) 章节内容；

拍摄照片实验：请参阅 [5.7.1 普通拍摄](#) 章节内容。

拍照后我们应该让图片停留一段时间，让用户观察照片的拍摄情况，然后再进行继续拍摄。代码编写流程如下：

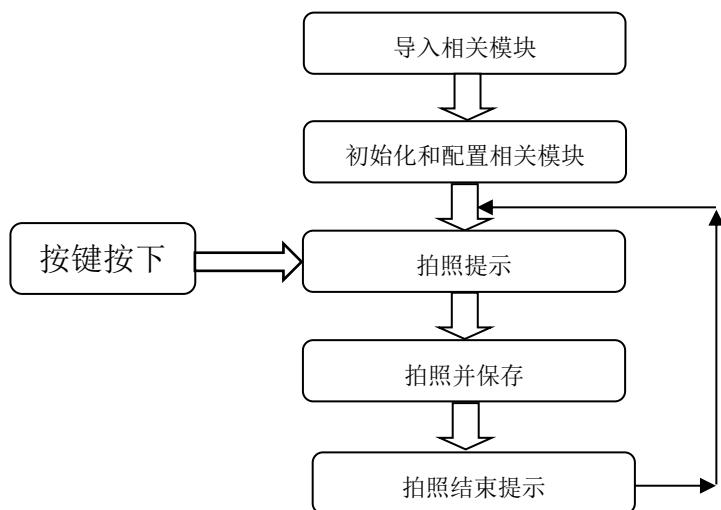


图 7-2 代码编写流程

参考代码如下：

```
...  
实验名称: 照相机  
版本: v1.0  
日期: 2019.10  
作者: 01Studio 【www.01Studio.org】  
说明: 通过按键拍照并在 LCD 上显示 (本实验需要 SD 卡)。  
...  
  
import sensor, image, pyb, lcd  
from pyb import ExtInt,Pin,RTC
```

```
RED_LED_PIN = 1
BLUE_LED_PIN = 3

#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.LCD) # or sensor.QQVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.

#LCD 初始化
lcd.init()

#时间初始化，用于给图片命名
rtc=RTC()

key_node = 0 #按键标志位

#####
# 按键和其回调函数
#####

def key(ext):
    global key_node
    key_node = 1

ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, key) #下降沿触发，打开上拉电阻

while True:

    lcd.display(sensor.snapshot()) # LCD 实时显示
```

```
if key_node==1: #按键被按下
    key_node = 0 #清空按键标志位

    #红灯亮提示用户看镜头
    pyb.LED(RED_LED_PIN).on()
    sensor.skip_frames(time = 2000)

    #红灯灭， 蓝灯亮提示开始拍照
    pyb.LED(RED_LED_PIN).off()
    pyb.LED(BLUE_LED_PIN).on()

    print("You're on camera!")

    #拍照并保存， 保存文件用时间来命名。
    lcd.display(sensor.snapshot().save(str(rtc.datetime())+".jpg"))

    pyb.LED(BLUE_LED_PIN).off()
    print("Done! Reset the camera to see the saved image.")

    #延时 3 秒， 观看拍摄图片
    pyb.delay(3000)
```

### ● 实验结果：

pyAI-OpenMV4 插入 SD 卡、接上 LCD 和锂电池。然后将以上代码编写好的 main.py 文件复制到 OpenMV4 文件系统中，按下复位即可运行。

然后对准要拍摄的东西，按下上方的 KEY 按键，即可拍摄！



图 7-3 拍摄图片

当然你也可以结合 pyBase 底板来自拍：

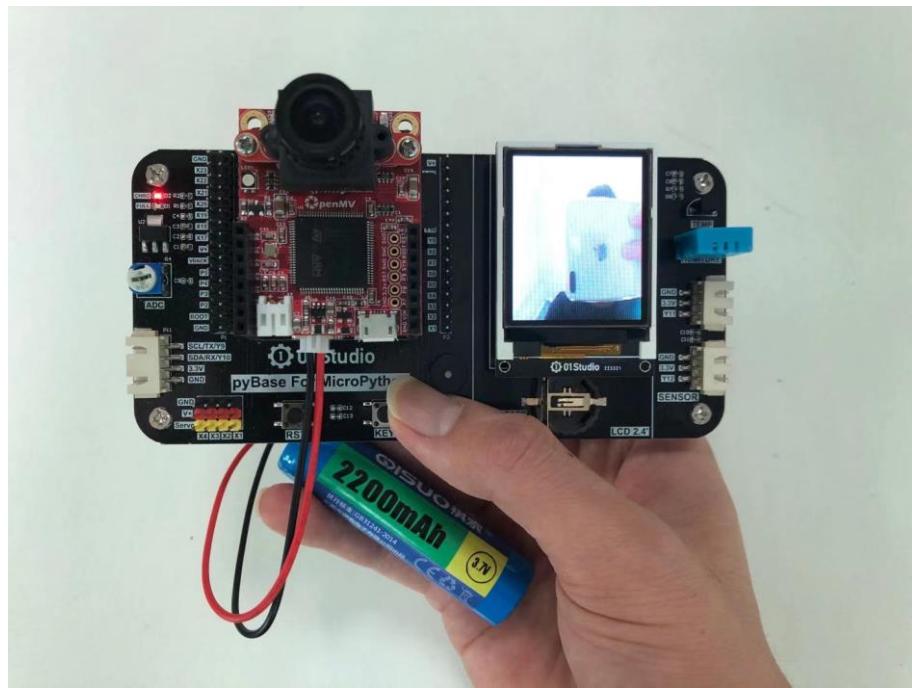


图 7-4 结合开发底板自拍

通过 Usb 线连接到 OpenMV4，按下复位，可以看到被保存的图片文件。(RTC 时间没做修改)

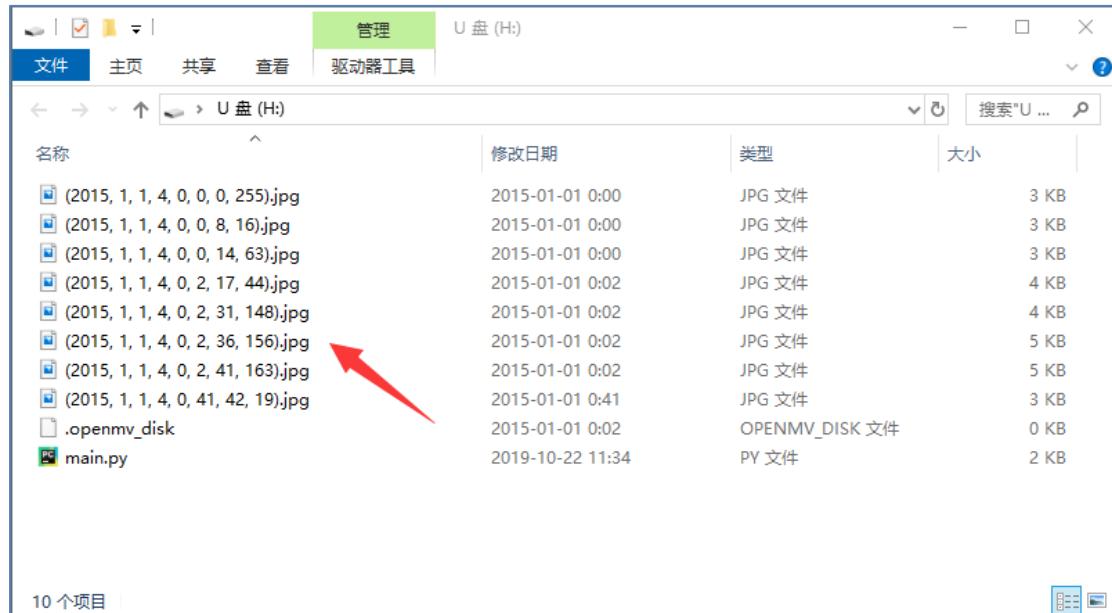


图 7-5 保存的图片文件

### ● 总结：

本节学习了一个简单的项目应用，照相机。可以看到项目是前面实验的一个综合，只要把基础知识掌握牢固了，就可以轻松使用 pyAI-OpenMV4 实现更多好玩有趣的项目。同时 OpenMV 也是不断升级的产品，相信不久就能用上更高清的摄像头。

# MicroBit 从0到1

用方块开始你的编程之旅

01Studio团队 编著



01Studio  
-让编程变得简单有趣-

# MicroPython 从0到1

用python做嵌入式编程

(基于pyBoard STM32F405平台)

01Studio团队 编著

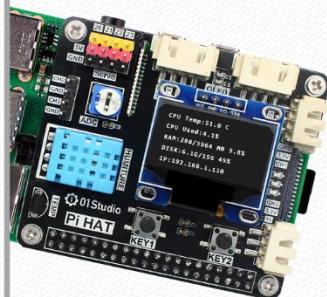


01Studio  
-让编程变得简单有趣-

# 树莓派从0到1

(基于树莓派4B平台)

01Studio团队 编著



01Studio  
-让编程变得简单有趣-

# 基于pyBoard STM32F405平台

MicroPython  
从0到1  
用python做嵌入式编程  
(基于pyBoard STM32F405平台)  
01Studio团队 编著



关注公众号，免费下载