SentryMV 使用手册

V1.1

1. SentryMV 简介

SentryMV 模式基于嘉楠科技 CanMV 开源固件而开发，该模式采用 MicroPython 编程语言对硬件进行编程和交互。MicroPython 是业内十分知名的低资源需求 Python 脚本解释器，直接运行于嵌入式处理器内，仅需通过串口向硬件设备传输 Python 脚本即可实现对其控制和逻辑处理，如读取摄像头，运行算法，输出数据等，无需固件编译，简单易用。

CanMV IDE 是嘉楠科技提供的一款可视化的开发环境软件，用于编写 MicroPython 程序，还可以进行上传代码，打印调试日志等操作，软件内置大量的开源示例程序可供用户学习和使用。SentryMV 模式同样可以通过该软件进行开发，并运行里面的开源程序。

2. SentryMV 固件下载与升级

Sentry2 视觉传感器使用 SentryMV 模式需要烧录 sentry2mv 的固件，该固件做了相应的硬件适配，对模型算法也做了函数优化处理，调用更简单易懂。

2.1 固件下载

目前 SentryMV 固件属于内测阶段，可通过瞳芯智能网盘下载点击此处进入网盘

或打开网盘地址，提取码：1022：

https://pan.baidu.com/s/1Ur39pkhnL8yznRqGbX2tkA?pwd=1022

网盘位置“***资料下载/视觉传感器/Sentry2/Sentry2MV 固件***”，下载相应的 sentry2mv 固件，分为消费版和企业版固件

企业版：vs_sentry2mv_k210_vX_X_X_XXXXXXXX_enterprise_e.kfpkg

消费版：vs_sentry2mv_k210_vX_X_X_XXXXXXXX_consumer_e.kfpkg

提示：上电后查看 Sentry2 屏幕运行界面右下方的字母标识，显示字母“E”或“S”则为企业版，字母“C”则为消费版

注意：使用 sentry2mv 固件后，Sentry2 的标准固件功能将无法使用，如果要切换回标准模式，需要重新烧录标准版的固件。

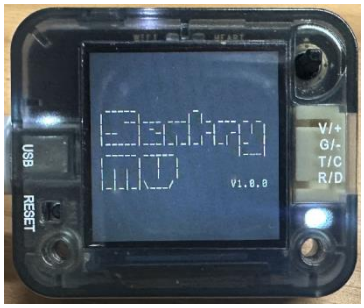
2.2 固件升级

请参考固件升级教程：

https://tosee.readthedocs.io/zh/latest/Sentry2/Upgrade/index.html

2.3 固件运行

新固件上电后会显示如下的开机画面：



3. CanMV IDE 软件下载与安装

3.1. 软件下载

CanMV IDE 版本为 V2.9.2，可以直接从 Github 上下载：

https://github.com/kendryte/canmv_ide/releases/download/v2.9.2-2/canmv-ide-windows-2.9.2-v2.9.2-2-0-g132467a.exe

或从网盘中下载：

网盘位置“***资料下载/视觉传感器/Sentry2/Sentry2MV 固件/CanMV IDE***”

3.2. 软件安装

打开安装包后根据提示进行安装，如遇问题，可以参考官方在线文档：

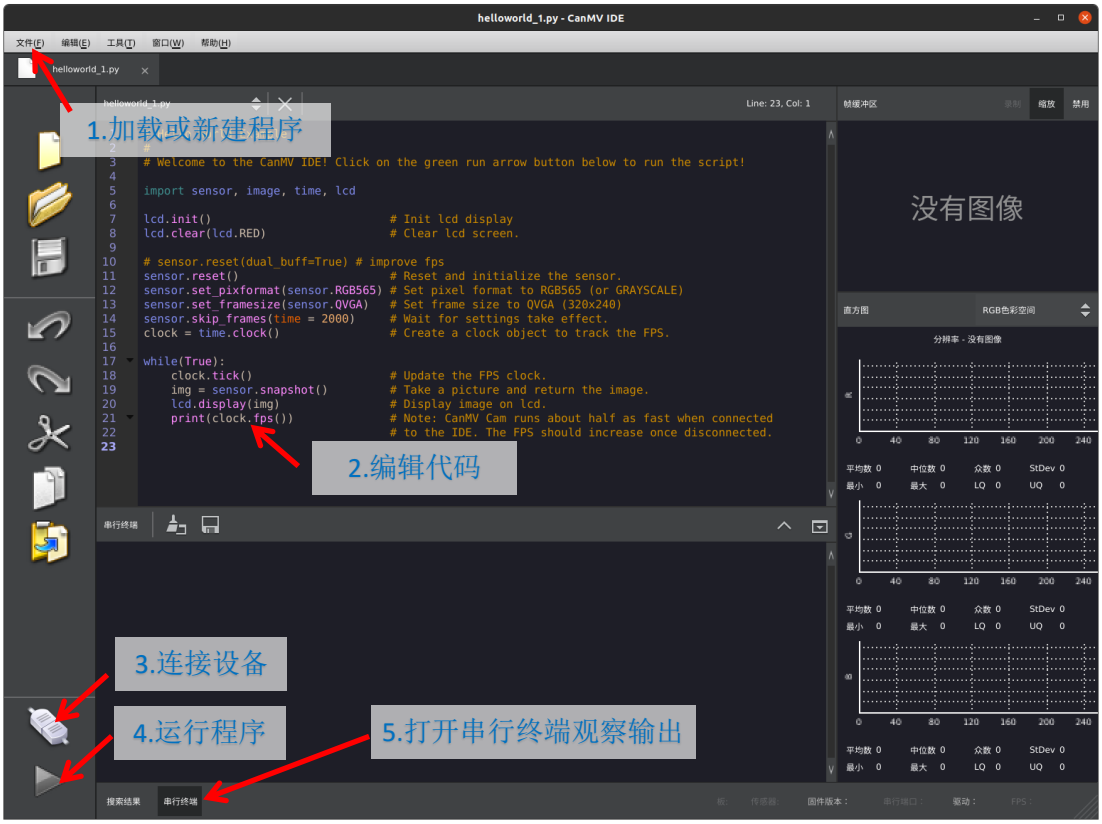
https://developer.canaan-creative.com/canmv/main/canmv/get-start/quick-start.html#canmv-ide

3.3. 使用简介

打开 CanMV IDE 后的可见到如下所示的界面

详细功能介绍可以参考官方在线文档：

https://developer.canaan-creative.com/canmv/main/canmv/index.html



4. SentryMV 使用指导

4.1. SD 卡导入模型

SentryMV 运行神经网络类型的算法，首先需要将算法模型按照固定的路径存储在 TF 卡中。

（1）准备一张 micro SD 卡 (即 TF 卡)，嘉楠 K210 芯片只能加载 SPI 信号的 SD 卡，所以部分 SD 卡无法被识别，具体情况可查看嘉楠科技对 SD 卡的解释与说明：

https://developer.canaan-creative.com/canmv/main/canmv/faq.html#tf-micro-sd

（2） 将 SD 卡格式化为 FAT32 格式，格式化会删除 SD 卡内所有的数据文件，请务必提前做好备份

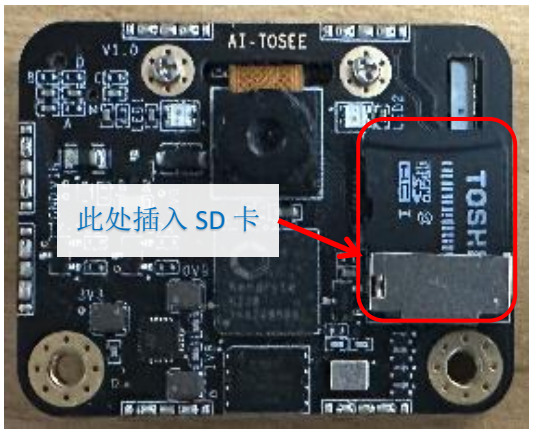
（3）从网盘中下载算法模型： “***资料下载/视觉传感器/Sentry2/Sentry2MV 固件/算法模型文件***”

（4）将算法模型解压后，连同“KPU”文件夹一起导入 TF 卡中，不要修改路径，算法模型保存的路径形式为：

/KPU/yolo_face_detect/face_detect_320x240.kmodel

（5）拆开 Sentry2 的外壳，并将 TF 插入卡座中，TF 卡的金手指朝下。

提示：即将推出可外接 SD 卡的 Sentry2 外壳，可免去拆除外壳的步骤

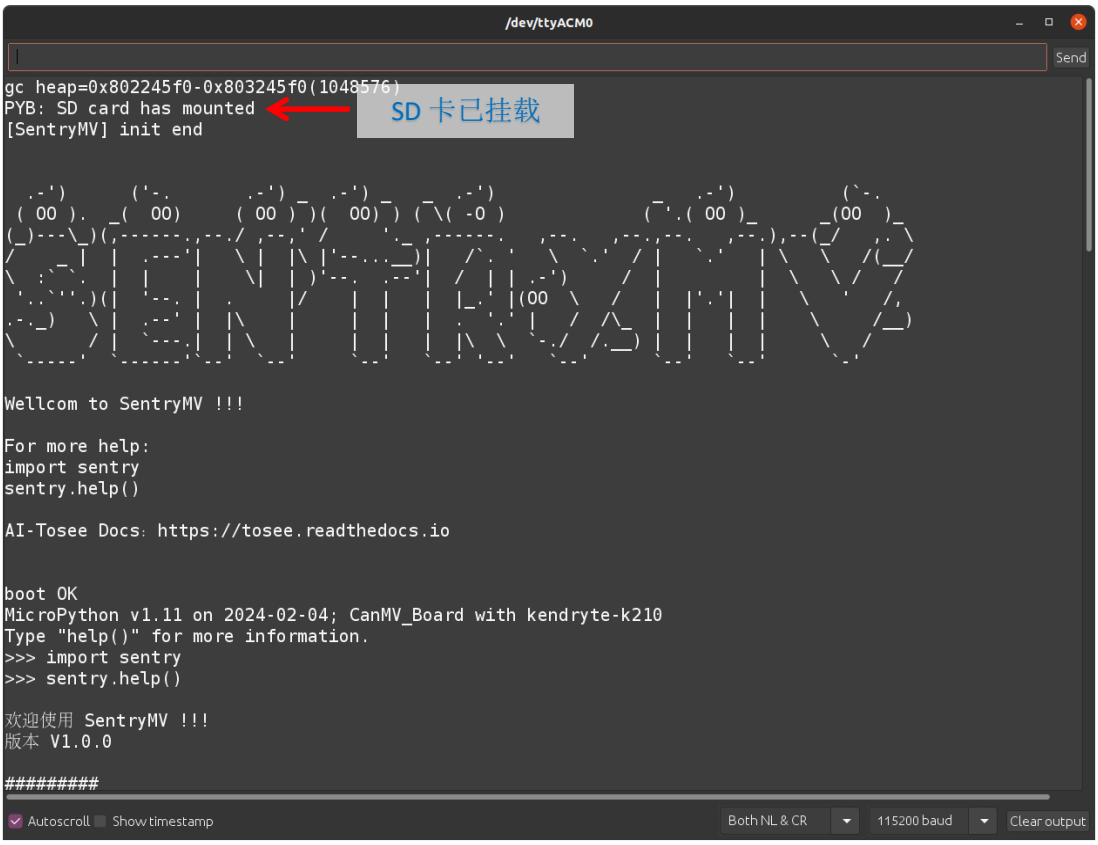


4.2. 通过串口助手查看信息

（1） 将 Sentry2 通过 USB-C 连接至电脑并打开串口助手（需要支持中文字符，比如 Arduino 中自带的串口调试器），设置波特率为 115200，数据位 8,停止位 1,无校验。

（2） 打开 Sentry2 对应的串口，此时可以看到有开机信息输出。如果没有自动打印开机日志，可以手动按一下 Sentry2 的 Reset 按键

输出日志中如果有“SD card has mounted”字样（下图所示），表示 SD 卡可以被识别



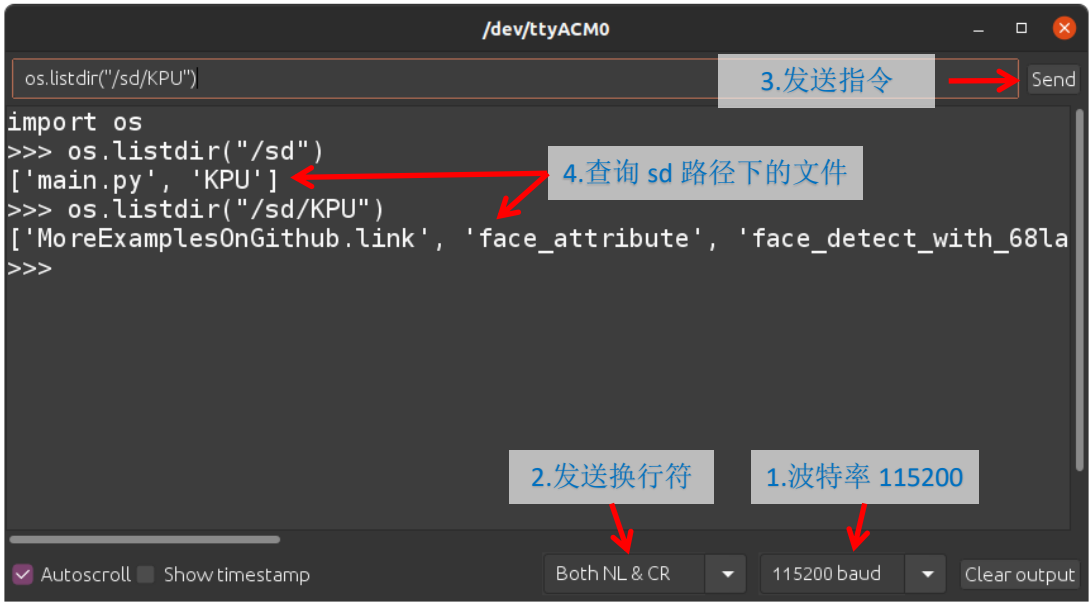
（3） 在串口助手中分别发送以下指令查看 sd 卡数据，发送指令时需要发送换行符号

import os

os.listdir（“/sd”）

os.listdir（“/sd/KPU”）

看到‘KPU’等字样则表明 sd 卡的数据是可以被读写的



4.3. 运行官方示例程序

CanMV IDE 内置丰富的示例程序，包括对硬件接口的操作、对图像的处理、寻找边缘特征、绘制图像、输出文字、识别形状、识别条码、识别颜色、识别人脸等，可以查看在线用户手册进行学习和使用：

https://developer.canaan-creative.com/canmv/main/canmv/demo/index.html

4.4. sentry.py 函数库

SentryMV 固件内置了 sentry.py 函数库，提供了几个常用的函数，并对部分 KPU 神经网络类型的算法采用了统一的函数接口，相比于自带的示例程序而言，代码变的更简洁明了，易于理解，从而可以更好的专注于应用程序的开发

sentry.py 支持的算法列表：

人脸属性检测 - FaceAttributeDetect
人脸 68 关键点检测 - FaceDetectWith68landMark
人脸口罩检测 - FaceMaskDetect
头部检测 - HeadDetect
身体检测 - BodyDetect
车牌识别 - LicenseplateRecognize
手写数字识别 - MnistNumber
常见 20 类物体检测 - Voc20ObjectDetect
人脸检测 - YoloFaceDetect
手部检测 – YoloHandDetect
自训练检测模型 – MyDetector
自训练分类模型 – MyClassifier

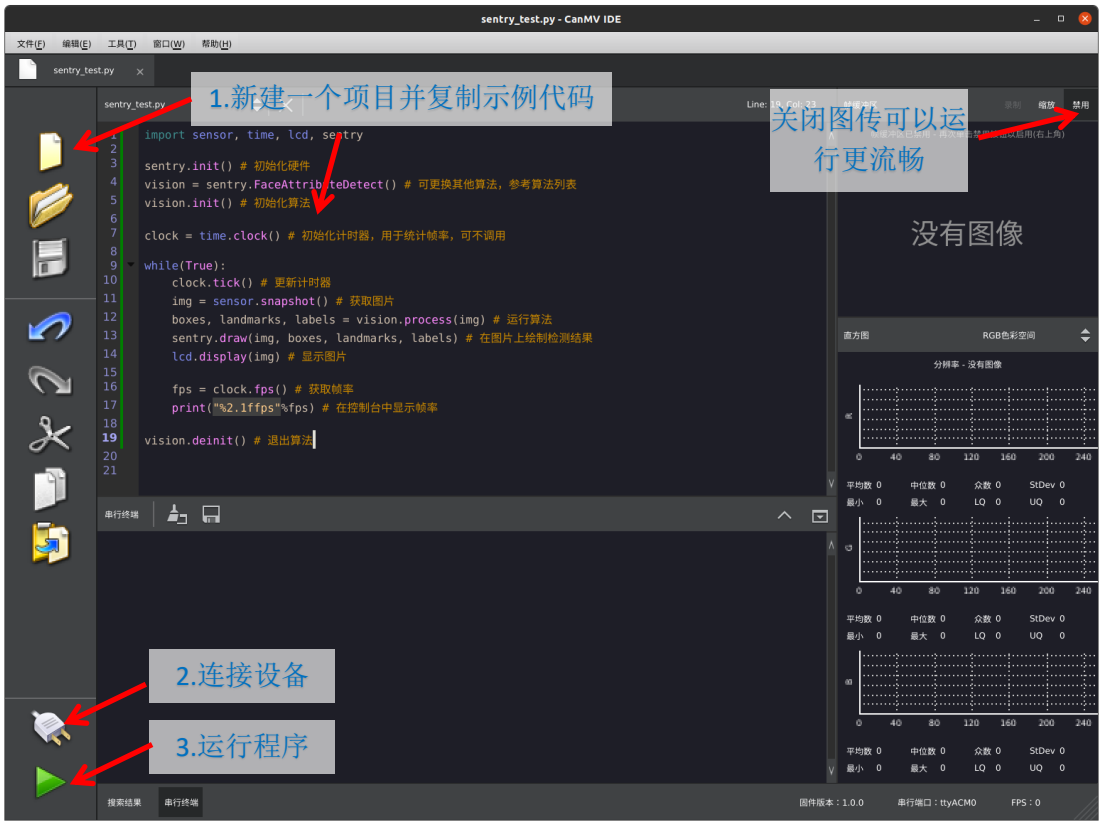
通过串口助手发送下面的指令查看 sentry.py 的帮助文档：

```
import sentry
sentry.help()
```

4.5. 运行人脸属性检测算法示例程序

通过该程序可以了解 sentry.py 算法调用的标准流程。主要分为硬件初始化、算法初始化、运行算法、绘制结果、屏幕显示几个步骤

人脸属性检测算法-*FaceAttributeDetect* 可以检测到人脸方位、大小，人脸 5 个关键点位（眼睛、鼻子、嘴角），以及性别，是否张嘴，是否微笑，是否戴眼镜的属性信息。操作步骤如下：



（1） 在 CanMV IDE 中新建一个项目，并复制以下代码。也可以从 sentry.help()的帮助信息中复制代码：

```
import sensor, time, lcd, sentry
sentry.init() # 初始化硬件
vision = sentry.FaceAttributeDetect() # 可更换其他算法，参考算法列表
vision.init() # 初始化算法
clock = time.clock() # 初始化计时器，用于统计帧率，可不调用
while(True):
    clock.tick() # 更新计时器
    img = sensor.snapshot() # 获取图片
    boxes, landmarks, labels = vision.process(img) # 运行算法
    sentry.draw(img, boxes, landmarks, labels) # 在图片上绘制检测结果
    lcd.display(img) # 显示图片
    fps = clock.fps() # 获取帧率
    print("%2.1ffps"%fps) # 在控制台中显示帧率
vision.deinit() # 退出算法
```

（2） 连接 Sentry2 设备

（3） 点击运行程序按钮

（4） 检测人脸并在 Sentry2 的屏幕上观察检测结果

提示：必要时可以关闭掉 CanMV IDE 中右上角的”图传功能”以获得更流畅的体验

（5） 当要停止程序运行时，需要点击 CanMV IDE 左下角结束按钮

注意：程序停止后会释放资源，这样才能继续运行下一个程序，否则会出现内存不足等错误提示。

注意：如果没有释放资源，则需要断开连接并重启设备后再运行新的代码

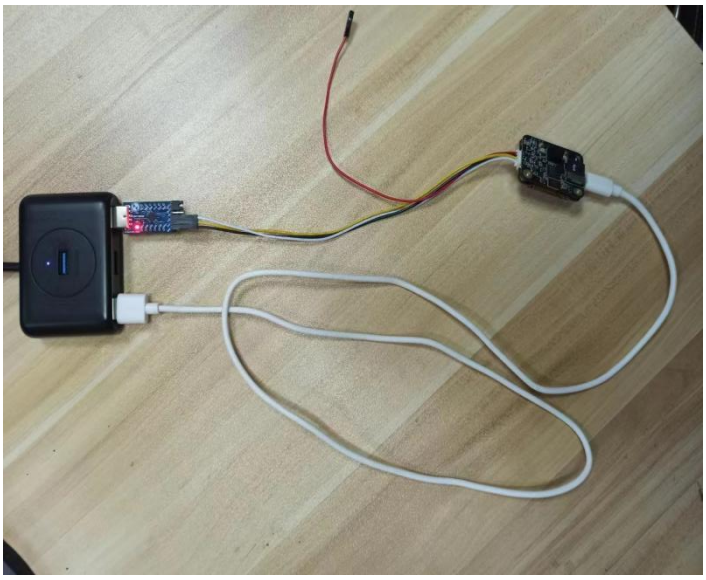
4.6. 运行车牌识别算法程序

该程序用于介绍如何调用 Sentry2 的 PH2.0 端口，并将识别结果进行输出

车牌识别算法-*LicenseplateRecognize* 可以进行离线车牌识别，其中车牌的省份用拼音标注

（1） 该例程可以将识别到的车牌信息通过 Sentry2 的 PH2.0 数据口对外传输，传输方式为 UART 模式，115200 波特率。

测试的电路如下图所示，Sentry2 的 USB-C 口作为程序的调试端口，Sentry2 的 PH2.0 端口作为数据输出端口，通过一块 USB 转串口模块连接至电脑，电脑端通过串口调试助手读取数据，用户也可以直接将 PH2.0 端口连接主控板



注意：当使用 USB-C 口供电时，请断开 PH2.0 数据口的供电，图中红色线，避免电流倒灌损坏器件。

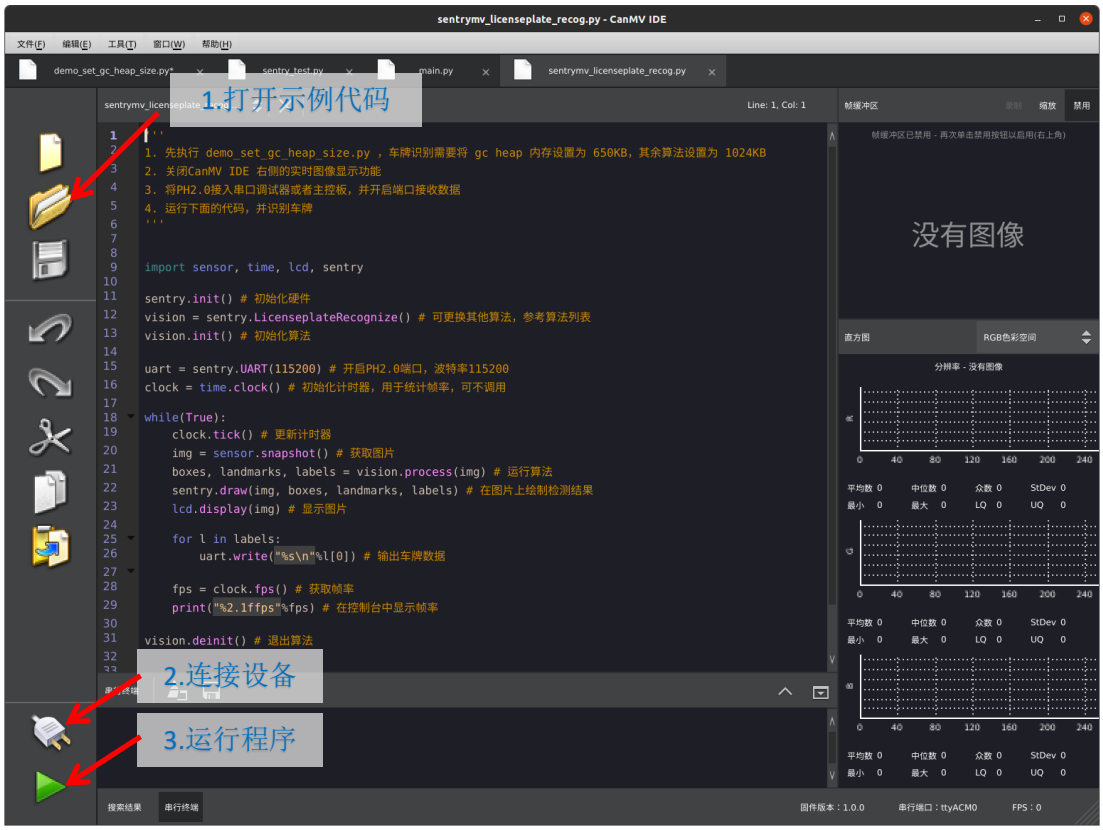
（2） 更改内部 gc heap 内存大小，因为该示例程序模型文件比较大，需要将 gc heap 内存设为 650KB 的大小才能正常运行。使用下面的代码进行设置，或在网盘中找到 ***demo_set_gc_heap_size.py*** 文件：

```
import machine
import maix

gc_mem_size = 650*1024

if maix.utils.gc_heap_size() != gc_mem_size:
    maix.utils.gc_heap_size(gc_mem_size)
    machine.reset()

```

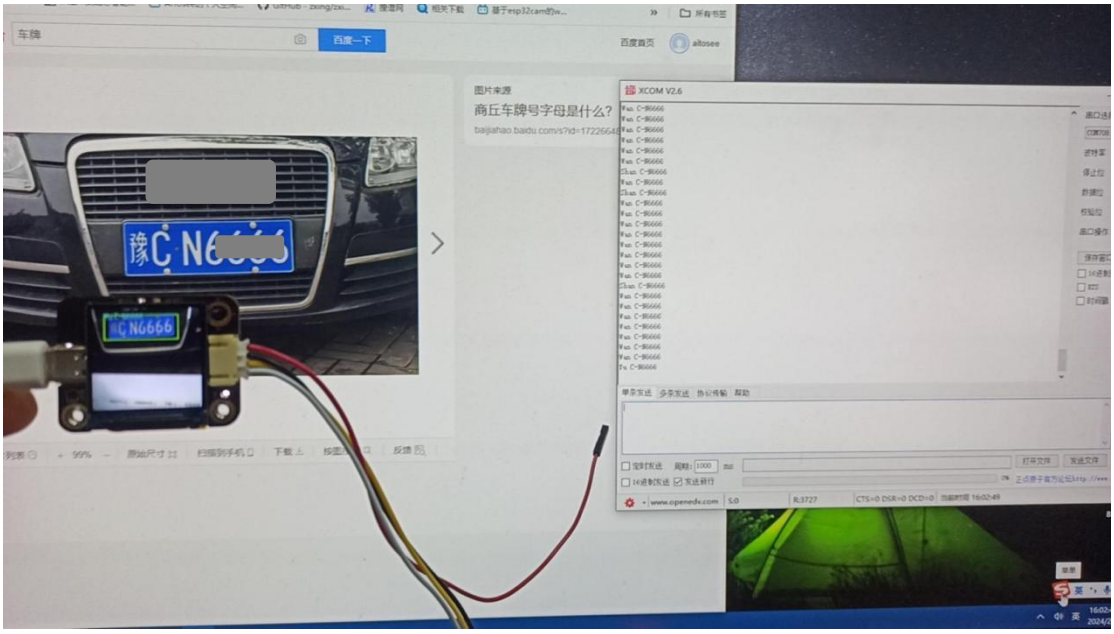
注意：运行其他算法时，需要将 gc heap 设置为 1024KB 的大小

（3） 加载示例程序

在网盘中找到示例程序并在 CanMV IDE 中打开：“***资料下载/视觉传感器/Sentry2/Sentry2MV 固件/应用例程***”



（4） 将 Sentry2 对准车牌并观察输出结果

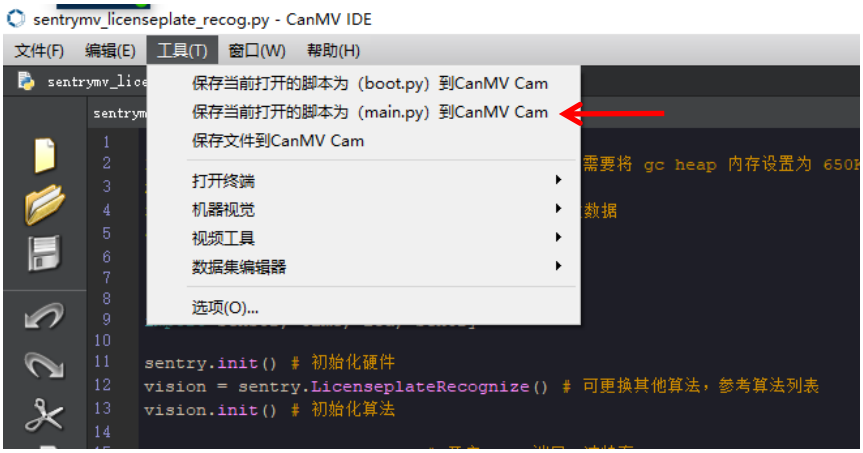


4.7. 算法开机自启动

SentryMV 开机后会加载并自动运行 main.py 脚本，用户可以将自己的程序存放在这个 main.py 中从而实现算法开机自启动，有 2 个方法可以实现：

方法 1（推荐）：将所需要运行的.py 文件重命名为 main.py，并将其拷贝至 SD 卡的根目录中实现开机自启动

方法 2：在 CanMV IDE 的“工具菜单”中，将当前打开的.py 文件直接烧录到 Sentry2 的 flash 中，如下图箭头所示：



注意：如果通过方法 2 向 flash 中写入的 main.py 代码有问题且无法再通过方法 2 进行改写，则可以将修改该后的.py 脚本并命名为“cover.main.py”存放到 SD 卡中，SentryMV 启动后会将该脚本替换到 flash 中，启动正常后从 SD 卡中删除该文件即可

注意：请不要覆盖掉 flash 中的 boot.py 脚本

5. 用户模型训练

SentryMV 支持导入用户自训练的模型文件，本章节将通过第三方 K210 模型训练软件 Mx-yolo 来讲解如何训练自己的算法模型。该软件集成度高，使用简单方便，支持“目标检测”和“图像分类”两种模型的训练，两者的训练和使用流程略有差异，如下所示：

5.1. 软件安装

进入网盘并下载安装程序 Mx-yolo-setup_V4.0.exe，双击运行并根据提示进行安装

网盘位置：“***资料下载/视觉传感器/Sentry2/Sentry2MV 固件/算法模型自训练***”

点击网盘下载 Mx-yolo-setup_V4.0.exe

注意：安装路径中不能包含中文字符

注意：该软件需要 3G 的硬盘空间，请确保所选择的安装盘符有足够的剩余空间

安装好后双击打开 Mx-yolo.exe 即可运行该软件

5.2. 图片数据采集

图片数据集是模型训练的基础，图片采集的数量和质量将会对模型效果起到至关重要的作用。要训练一个较好的算法模型通常需要一万张甚至数十万张以上的图片数据，涵盖多种不同环境和角度的图片数据。对于教学而言，单个物体 100 张左右的图片数据一样可以满足训练需求，建议不少于 300 张有效图片。

图片数据可以来自于网上的开源数据集，对于特殊物体则需要自行拍摄。Mx-yolo 自带“图片采集”功能，可以通过调用电脑上的摄像头进行图片采集。

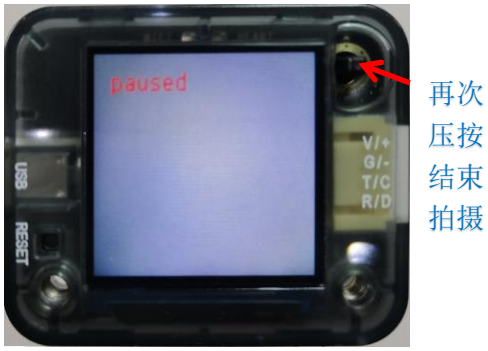
但为了取得更好的识别效果，我们推荐使用 sentry2 的相机进行拍摄，这样得到的原始数据与硬件最为吻合。可以从网盘上下载拍摄脚本 sentrymv_auto_capture_picture.py，该脚本可以实现连续拍摄图片并保存至 SD 卡中

网盘地址：“***资料下载/视觉传感器/Sentry2/Sentry2MV 固件/应用例程/拍照并保存至 TF 卡例程***”

通过 CanMV IDE 加载该文件并运行，程序运行后左上角会显示“paused”字样，垂直压按摇杆可以启动拍照功能，将每隔 200ms 拍摄一张图片，再次压按摇杆将会停止拍照。

为适配 Mx-yolo，拍摄图片将保存为 224x224 分辨率的图片，文件名按照 image_xxx.jpg 的格式保存在 SD 卡的“image”文件夹中。文件名中的 xxx 为图片编号，从 0 开始顺序增加，拍摄完的数据请及时拷贝到电脑端，避免丢失或被覆盖

拍摄中



停止拍摄

为了取得较好的识别效果，应当在实际的使用场景内或者接近的场景内采集数据，降低由背景与光照差异引起的不利影响

5.3. 图片数据整理

首先需要对所采集的图片进行筛选，删除以下几类图片：

•无关图片

•目标物体模糊

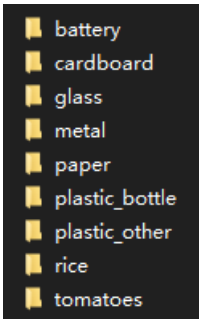
•目标物体不完整

•目标物体过小

遍历结束后，如果认为某个或者某些角度下的图片数量不足，请补采集数据

对于“目标检测”算法，所有图片存放于同一个 images 文件夹内即可

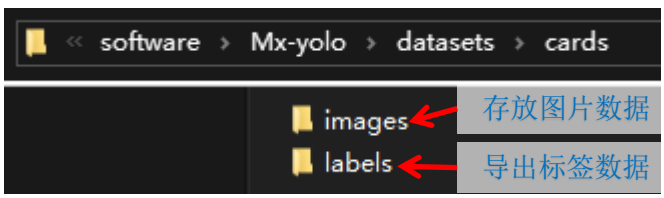
对于“图像分类”算法，所拍摄图片需要按类别分别存放于不同的文件夹内，例如垃圾分类的图片数据：



5.4. 图片数据标注

训练“目标检测”模型之前需要对每一张图片进行标注处理，标注是指在原始图片中标记出目标物体并分配一个分类标签，最后生成一系列的标注文档用于训练。

在工作目录下新建两个文件夹：images 和 labels。将所有训练图片数据拷贝至 images 文件夹内



在 Mx-yolo 的界面中选择“工具集”标签页，点击“打开图片标注助手”按钮，将会打开图片标注工具：

按图片中所示步骤进行图片数据标注：

注意：加载的图片和标签路径中不能包含中文字符

5.5. 目标检测模型训练

训练“目标检测”模型步骤如下：

(1) 在 Mx-yolo 上方标签栏中选择“目标检测”页面；

(2) 选择图片文件夹地址，不可包含中文字符

(3) 选择标签文件夹地址，不可包含中文字符

(4) 配置训练参数：软件提供了 4 个参数可以配置。其中“训练次数”越大，模型训练所需的时间也就越长，模型效果也会有所改善且越趋于稳定。“Alpha”值决定了输出模型的大小，当硬件内存不足导致加载模型失败时，可以减小该值，但识别效果可能会有所降低，一般取 0.75 或 0.5 即可。“Batch Size”和“数据增强”保持默认值即可，一般不建议改变

(5) 点击“开始训练”

(6) 查看训练过程数据和日志，如果训练出现问题，需要点击右上角“停止训练”按钮，然后再重新“开始训练”。一般常见的问题为训练图片与标签数据不对应，或者图片尺寸有问题等。

(7) 训练结束后会弹出“训练完成”对话框

(8) 可以在右侧上方查看“训练记录”，查看训练效果是否满足需求

5.6. 图像分类模型训练

训练“图像分类”模型步骤与“目标检测”操作步骤基本一致，只是不需要加载“标签文件夹地址”，可以参考图片中的步骤进行操作：

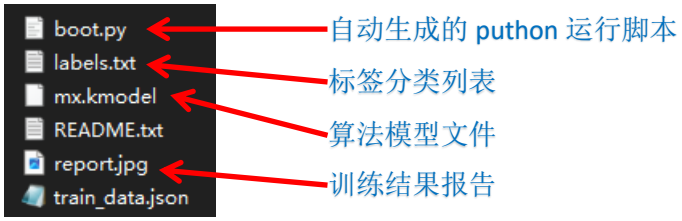
注意：加载的图片路径中不能包含中文字符

5.7. 查看模型文件

“目标检测”模型训练结束后，所生成的模型文件会存放于：
Mx-yolo/out/yolo_xxxx-xx-xx_xx-xx-xx/result_root_dir/detector_result

“图像分类”模型训练结束后，所生成的模型文件会存放于：
Mx-yolo/out/classifier_xxxx-xx-xx_xx-xx-xx/result_root_dir/detector_result

文件夹里包含以下文件：



5.8. 调用自训练模型

用户自训练模型可以在 CanMV 软件中，参考上章节的 boot.py 脚本来运行算法，但一般来说需要做一些代码修改该和算法后处理操作才能稳定运行起来，适合有经验的工程师。

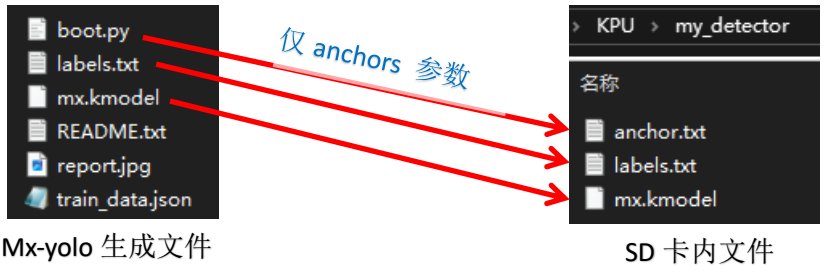
SentryMV 为了方便用户调用自训练的算法模型，在 sentry.py 中提供了两个通用算法类，只需将算法模型相关的文件拷贝至 SD 卡中即可方便的运行算法：

目标检测 - MyDetector 算法类

图像分类 - MyClassifier 算法类

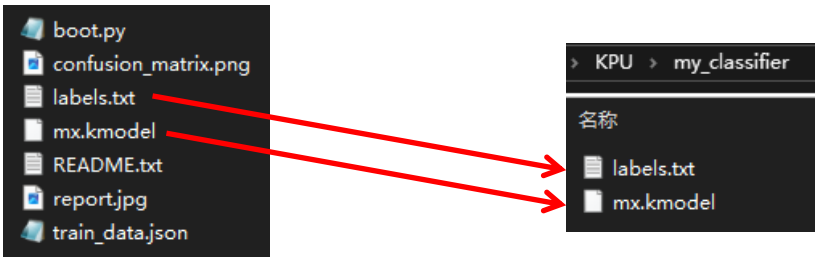
算法调用步骤：

（1） 将模型及其相关文件存入 SD 卡中：调用“目标检测”算法需要准备 3 个文件 mx.kmodel，labels.txt 和 anchor.txt，存放于 SD 卡的“/sd/KPU/my_detector”地址。“图像分类”只需要 mx.kmodel，labels.txt 两个文件，存放于 SD 卡的“/sd/KPU/my_classifier”地址。其中 mx.kmodel 和 labels.txt 为 5.7 章节中模型训练后生成的文件，直接复制和替换掉原有文件即可。



Mx-yolo 生成文件　　SD 卡内文件

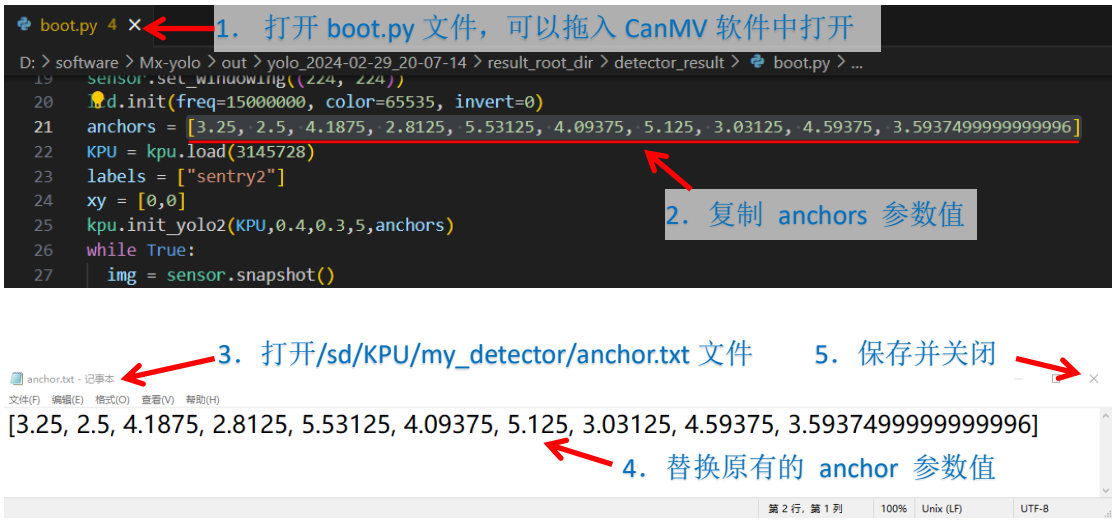
目标检测算法所需文件



Mx-yolo 生成文件　　SD 卡内文件

图像分类算法所需文件

（2） “目标检测”算法所需的 anchor.txt 为算法初始化时的参数，可以从 5.7 章节的 boot.py 中获取到，可以按下图所示步骤操作：



（3） 打开 CanMV，修改 4.5 章节人脸模型调用示例中的算法类型，并运行算法即可

目标检测：vision = sentry.MyDetector()

图像分类：vision = sentry.MyClassifier()

6. 硬件内置函数

本章节对 sentry.py 的内置函数做一些简介，并给出一些示例代码

6.1. 硬件相关函数

函数名	简介
help()	打印帮助文档
UART(baudrate=115200)	初始化通讯端口为 UART 模式，可修改波特率，其余参数为：数据位 8,停止位 1,校验无
I2C(freq=100000)	初始化 PH 通讯端口为 I2C 模式，可修改该时钟频率
UpKey(irq_handle=None)	初始化摇杆按键上，可添加中断函数
DownKey(irq_handle=None)	初始化摇杆按键下，可添加中断函数
LeftKey(irq_handle=None)	初始化摇杆按键左，可添加中断函数
RightKey(irq_handle=None)	初始化摇杆按键右，可添加中断函数
EnterKey(irq_handle=None)	初始化摇杆按键中，可添加中断函数
Led()	初始化 RGB LED，基于 ws2812 驱动，总共有 2 颗
LedHeart()	初始化心跳指示灯，基于 GPIO 控制
init(sensor_pixformat=sensor.RGB565, sensor_framesize=sensor.QVGA, sensor_skip_frames=2000)	初始化摄像头和屏幕等硬件，默认采用 320x240 的 QVGA 分辨率，如果不使用内置函数，则无需加载，直接用 CanMV 示例中的 sensor 调用方法即可
draw(img, boxes, landmarks=None, labels=None, color_boxes=(0, 255, 0), color_landmarks=(0, 0, 255), color_labels=(0, 255, 255), label_scale=2, fullscreen=True)	在图片 img 上绘制检测结果，包括识别框 boxes，关键点 landmarks，标签名称 labels color_xxx：可以修改颜色 label_scale：可以缩放字体 fullscreen：是配合 sentry.display 函数使用的，详见该函数的介绍
display(img, top_string = "", bot_string = "", fullscreen = True)	sentry.py 内置的屏幕显示函数，可以显示一些文字和图片缩放，但于某些算法运行时会遇到内存不足的问题，请使用 CanMV 自带的 lcd.display 函数 top_string：屏幕上方添加待显示的字符串 bot_string：屏幕下方添加待显示的字符串， fullscreen：决定图片是否全屏显示，为 True 时，将全屏显示图片，但只会从 320x240 的图片中截取中间 240x240 的区域进行显示，帧率高，内存少。为 False 时，会将 320x240 的图片压缩至 240x180 进行显示，可以显示完整的摄像头画面

6.2. 算法相关函数

函数名	简介
init(...)	算法初始化，加载模型和参数，参数全部采用默认值，需要将所有的模型存放在 SD 卡中的指定位置，否则加载不到
boxes,landmarks,labels = process(img)	运行算法并返回结果，boxes（物体中心点坐标、大小框，分类标签 id 值，检测得分），landmarks（关键点或中心点），labels（分类标签字符串）
deinit()	退出算法，释放资源

6.3. 自训练检测算法初始化函数

函数名	简介
init(self, model_path="/sd/KPU/my_detector/mx.kmodel", label_path="/sd/KPU/my_detector/labels.txt", anchor_path="/sd/KPU/my_detector/anchor.txt", img_w=224, img_h=224, net_w=224, net_h=224, layer_w=7, layer_h=7, threshold=0.7, nms_value=0.3, classes=1)	算法初始化有默认参数，必要时可以修改其参数 仅适合专业工程师使用

6.4. 自训练分类算法初始化函数

函数名	简介
init(self, model_path="/sd/KPU/my_classifier/mx.kmodel", label_path="/sd/KPU/my_classifier/labels.txt", net_w=224, net_h=224, threshold=0.7)	算法初始化有默认参数，必要时可以修改其参数 仅适合专业工程师使用

6.5. 硬件调用示例程序

```
import  sentry, time
# 右按键中断函数
def irq_right_key(pin_num):
    led.set_led(0,(10,0,0)) # 设置 Led0 的颜色，R=10,G=0,B=0，即红色
    led.set_led(1,(0,0,0)) # 设置 Led1 的颜色，R=0,G=0,B=0，即关闭
    led.display() # 显示颜色
# 左按键中断函数
def irq_left_key(pin_num):
    led.set_led(0,(0,0,0))
    led.set_led(1,(0,10,0))
    led.display()

uart = sentry.UART(115200) # 初始化 PH2.0 的串口
led = sentry.Led() # 初始化 2 颗 RGB LED
right_key = sentry.RightKey(irq_right_key) # 初始化右按键
left_key = sentry.LeftKey(irq_left_key) # 初始化左按键
heart = sentry.LedHeart() # 初始化心跳指示灯

while(True):
    uart.write("hello world") # 串口打印
    heart.value(1) # 心跳灯亮起
    time.sleep_ms(300) # 延时 300ms
    heart.value(0) # 心跳灯关闭
    time.sleep_ms(200)
```

6.6. 算法结果输出示例

```
import sensor, time, lcd, sentry

sentry.init() # 初始化硬件
vision = sentry.FaceAttributeDetect() # 可更换其他算法，参考算法列表
vision.init() # 初始化算法
uart = sentry.UART(115200) # 初始化通讯端口为 UART 模式

while(True):
    img = sensor.snapshot() # 获取图片
    boxes, landmarks, labels = vision.process(img) # 运行算法
    sentry.draw(img, boxes, landmarks, labels) # 在图片上绘制检测结果
    lcd.display(img) # 显示图片

    # 如果要在终端中显示数据，可以将以下 uart.write 替换为 print
    # boxes 数据结构：
    # [[x1,y1,w1,h1,l1,s1],[x2,y2,w2,h2,l2,s2],...]
    for b in boxes:
        uart.write('x=%d'%b[0]) # 输出中心点水平坐标 x
        uart.write('y=%d'%b[1]) # 输出中心点垂直坐标 y
        uart.write('w=%d'%b[2]) # 输出物体宽度 w
        uart.write('h=%d'%b[3]) # 输出物体高度 h
        uart.write('l=%d'%b[4]) # 输出物体分类标签号 l
        uart.write('s=%f'%b[5]) # 输出物体检测得分 s，小数值

    # landmarks 数据结构：
    # [[(x11,y11),(x12,y12),...],[(x21,y21),(x22,y22),...]],...]
    for lm in landmarks:
        for m in lm:  # 遍历所有关键点
            uart.write('lmx=%d'%m[0]) # 输出关键点坐标 x
            uart.write('lmy=%d'%m[1]) # 输出关键点坐标 y

    # labels 数据结构：
    # [['label11','label12',...],['label21','label22',...],...]
    for lb in labels:
        for l in lb:  # 遍历所有标签
            uart.write('label=%s'%l)  # 输出标签字符串

vision.deinit() # 退出算法
```