

二零二零冬季教程 POWERSENSOR

零基础入门篇

图像处理基础篇

典型案例应用篇

进阶功能改装篇

人工智能加速篇

兼容拓展模块篇



Powersensor 入门教程

Tutorial for Powersensor

作者：xiaobo & 师

组织：Powersensor Developer

时间：2020 年 12 月

版本：V1.5



创新、便捷、高效、愉悦

更新说明

主要内容

2020/12/31 更新：Powersensor Tutorial 1.5，**配套固件：img_2020.12**

- ① 教程重新组织，目前分为入门篇、图像处理基础篇、图像应用篇、典型应用案例篇、进阶功能改装篇、人工智能加速篇、兼容拓展模块篇共 6 个部分。同时新增独立的《API 文档》方便给高级开发用户查阅；
- ② 增加了 Opencv 的 contribute 模块支持；
- ③ 增加了 dlib 图像处理库的支持；
- ④ 【进阶功能改装篇】，增加了 TcpUdp 上位机的支持，可以在 offline 模式下通过网络传输图像；
- ⑤ 【典型应用案例篇】，增加了 kcf 目标跟踪案例；
- ⑥ 【典型应用案例篇】，增加了人眼检测案例；
- ⑦ 【人工智能加速篇】，增加了 5 花分类的案例；
- ⑧ 【人工智能加速篇】，增加了石头剪刀步的案例；
- ⑨ 【人工智能加速篇】，增加了口罩佩戴识别的案例；
- ⑩ 【兼容拓展篇】，增加了 3 个信号转换、隔离模块的教程，分别是 IO 电平转换与 232 电平转换、双舵机云台驱动板、双直流电机驱动板；
- ⑪ 【兼容拓展篇】，增加了 pwm 波拓展芯片（Pca9685）的支持；
- ⑫ 【兼容拓展篇】，增加了 tof 距离测量芯片（vl53lx）的支持；
- ⑬ 【兼容拓展篇】，增加了红外温度阵列测量芯片（Mlx90621）的支持；
- ⑭ 【兼容拓展篇】，增加了热成像传感器（Lepton 3.5）的支持；
- ⑮ 【兼容拓展篇】，增加了 2.8 寸 TFT 液晶屏（ili9341）的支持；
- ⑯ 修复了一些错误；

2020/05/28 更新：Powersensor Tutorial 1.4.1 beta，**配套固件：img_2020.05Beta**

- ① 增加了 DPU 支持（2020.05 及以上固件支持）；
- ② 增加了 usb 转网口支持，RTL8152 模组（2020.05 及以上固件支持）；
- ③ 增加了深度学习训练、部署和加速的案例（minist 数字识别，tensorflow 转 fpga 模型）（2020.05 及以上固件支持）；
- ④ 增加了通用无线键盘的支持；
- ⑤ 修复了一些错误；

2019/12/25 更新：Powersensor Tutorial 1.3，**配套固件：img_2019.12**

- ① 增加了背板的 GPIO 接口的教程（2019.12 及以上固件支持）；

-
- ② 增加了相机白平衡设置的教程 (2019.12 及以上固件支持);
 - ③ 增加了相机弱光模式 (2019.12 及以上固件支持);
 - ④ 增加了相机的曝光控制 (手动模式/自动恒亮度模式) (2019.12 及以上固件支持);
 - ⑤ 增加了开机连接局域网无线路由器的教程 (2019.12 及以上固件支持);
 - ⑥ 增加了传感器板 IMU 读取的教程 (需要使用带 IMU 的传感器板, 2019.12 及以上固件支持);
 - ⑦ 增加了多边形检测的教程 (Demo3);
 - ⑧ 增加了相机标定的教程 (Demo4);
 - ⑨ 增加了通过 apriltag 进行位姿测量的教程 (Demo6);
 - ⑩ 开放了硬件说明, 可以进行 fpga 开发;
 - ⑪ 修复了一些错误;

2019.10.10 更新: Powersensor Tutorial 1.2 更新, 配套固件: [img_2019.10](#)

- ① 增加了背板的 i2c、spi 接口的教程 (2019.10 及以上固件支持);
- ② 增加了 PWM 波输出的教程 (2019.10 及以上固件支持);
- ③ 增加了固件烧写的教程;
- ④ 增加了 demo-小球颜色识别;
- ⑤ 修复了一些错误;

常见问题

1. 有没有屏幕?

考虑许多的用户的建议, 我们这次更新添加了 2.8 寸液晶屏的支持。

2. 与单片机如何通信?

TTL 串口, 什么是 TTL 串口? 有这个疑问的小伙伴应该不关心这个功能。。

3. 需要安装什么开发软件?

需要浏览器, 推荐使用 google chrome, (这是给它打广告么?)。然后如果你需要使用串口的话, 还需要安装串口调试助手和驱动。这些软件在 sd 卡里都有的噢!

4. 学习 python 需要多久?

python 是一种非常平易近人的语言, 是为人类设计的语言 (不像 C, 设计主要考虑机器的心情), 非计非电的人也可以轻松使用。如果你只会 c 语言, 那么大概半个月基本掌握; 如果你会 java, 那么大概一周就够了; 如果你会 C++, 转 python 一小时够了。

5. 一会儿能连上, 一会儿连不上, 网络搜不到等问题

检查供电, 网络模块启动需要 800ma 电流, 建议使用我们配套的电源或者华为的电源供电。连不上 ps 首先检查通信模块, usb 转网口闪红灯, wifi 模块闪蓝灯。

6. offline.py 使用 dpu 失败? 或其他在线正常, 离线失败的情况:

需要把配套的文件, 像模型文件等复制到 jupyter 的 ‘/’ 目录下, 这个目录实际是

linux 下的 ‘/home/debian’

7. 运行了一会儿 powersensor 发现变卡了？

减少正在运行的 notebook 的数量，正在运行的 notebook（就是.ipynb 文件）在 jupyter 的文件管理器里面会显示成绿色，把用不到的 notebook shutdown 掉可以减少资源消耗。另外，带电脑与 powersensor 离得太远，超过 5 米后，WiFi 通信的速度也会下降。

8. offline.py 不能运行？

offline.py 不支持中文注释，不要使用原生的 printf，不要显示图片（也看不到），另外可以在 notebook 里使用

9. sd 卡插到电脑出现不能识别的盘？

这个是正常的，powersensor 的 sd 卡有两个盘，一个 boot 是可以打开的，包含教程、常用软件等内容，另一个是 ext4 格式的磁盘，linux 才能看到，windeos 无法识别，会显示报错。

10. 目前有哪些 FPGA 硬件加速？

目前主要是对输入的图像进行白平衡、去噪以及 dpu 深度学习加速。很快会有一些新的硬件加速功能被加进来。如一些特征算子，像 Sobel 之类的。什么是硬件加速？powersensor 使用的平台含一个强大的 FPGA，硬件加速是在使用 FPGA 读取传感器数据的同时对数据流进行处理，所以基本获取到图像，也就同时获取到相应的特征信息（会提供函数来读取）。

11. 能否进行 fpga 开发？

可以!!! 经过漫长时间的整理，这一次的教程提供了完善的芯片接口信息，arm 核、内存的配置信息，传感器板信息等。基于这些资料，用户可以方便自行开发所需要的 fpga 功能。当然对于硬件开发，我们不提供技术支持，因为这并不是我们主推方向，只是方便有基础有经验的小伙伴玩耍。

目录

第一部分 入门篇 - Quick Start	1
1 Quick start 1 - 开机测试及整体介绍	2
1.1 第一次开机	2
1.2 软件测试	4
1.3 教程介绍	7
2 Quick start 2 - 最基本的功能	9
2.1 启动管理	9
2.2 Jupyter 常见操作	10
2.3 Powersensor 的常用编程模板	12
2.4 IMU 读取实验	16
3 Quick start 3 - SD 卡固件烧写说明	19
3.1 烧写步骤	19
4 Quick start 4 - 通用通信接口	21
4.1 Powersensor 的常用接口定义	21
4.2 SPI 通信	22
4.3 I2C 通信	24
4.4 串口通信	26
4.5 PWM 输出	28
4.6 普通 IO	31
第二部分 图像处理基础篇 - Image Basic	34
5 Tutorial 1 - 在图像上做标记	35
5.1 图片读取	35
5.2 在图片上做标记	36
5.3 小结一下	40
6 Tutorial 2 - 图像预处理（一）常见变换	41
6.1 缩放/裁剪/平移	41
6.2 旋转/扭曲/镜像	44
6.3 色彩空间变换	46
6.4 小结一下	49

7 Tutorial 3 - 图像预处理（二）平滑滤波	50
7.1 均值滤波	50
7.2 高斯滤波	52
7.3 中值滤波	54
7.4 双边滤波	56
7.5 小结一下	58
8 Tutorial 4 - 图像预处理（三）-形态学处理	59
8.1 腐蚀膨胀	59
8.2 开/闭运算	62
8.3 高帽黑帽	65
8.4 小结一下	66
9 Tutorial 5 - 图像的特征（一）-直方图	67
9.1 直方图	67
9.2 直方图均衡	70
9.3 小结一下	72
10 Tutorial 6 - 图像的特征（二）梯度特征	73
10.1 Sobel 算子	73
10.2 Laplacian 算子	75
10.3 Roberts 算子	78
10.4 Canny 算子	80
10.5 小结一下	82
11 Tutorial 7 - 图像的特征（三）频域特征	83
11.1 离散傅里叶变换	83
11.2 频域高通滤波	86
11.3 频域低通滤波	88
11.4 小结一下	90
12 Tutorial 8 - 图像的轮廓提取	91
12.1 固定阈值二值化	91
12.2 自适应二值化	95
12.3 小结一下	97
13 Tutorial 9 - 图像的形状检测	98
13.1 直线检测	98
13.2 圆检测	102
13.3 小结一下	104

14 Tutorial 10 - 动态图形处理基础	105
14.1 帧检测	105
第三部分 典型应用案例篇 - Demo	108
15 Demo1 - 人脸检测	109
15.1 人脸检测	109
15.2 猫脸检测	111
16 Demo2 - 彩色小球检测	112
16.1 原理讲解	112
16.2 参考例程	112
16.3 结果展示	114
16.4 小结一下	114
17 Demo3 - 多边形检测	115
17.1 原理讲解	115
17.2 参考例程	115
17.3 结果展示	117
18 Demo4 - 相机标定	118
18.1 原理讲解	118
18.2 标定原理	121
18.3 参考例程	121
18.4 结果展示	124
19 Demo5 - 二维码识别	125
19.1 普通二维码识别	125
19.2 Apriltag 识别	127
20 Demo6 - 二维码位姿测量	130
20.1 检测原理	130
21 Demo 7 - kcf 目标跟踪	133
21.1 检测原理	133
22 Demo 8 - Dlib 人眼检测	136
22.1 检测原理	136

第四部分 进阶功能改装篇 - Advance	140
23 Advance 1 - 相机参数设置	141
23.1 明亮模式/弱光模式	141
23.2 自动/手动曝光模式设置	143
23.3 固定白平衡/自动白平衡	148
24 Advance 2 - FPGA 底层定制	150
24.1 SSH 登录	150
24.2 FPGA 开发	150
25 Advance 3 - TcpUdp 通信	154
25.1 介绍	154
25.2 实验过程	155
26 Advance 4 - 高级网络连接	159
26.1 连接无线路由	159
26.2 USB 转网口设备的使用	161
第五部分 人工智能加速篇 - AI	163
27 AI 1.1 - 深度学习模型训练	164
27.1 写在前面	164
27.2 下载链接	164
27.3 背景知识	164
27.4 深度学习案例的开发流程	165
28 AI 1.2 - 深度学习模型转换	177
28.1 介绍	177
28.2 编译过程	177
29 AI 1.3 - 深度学习模型部署	182
29.1 介绍	182
29.2 主要过程	182
30 AI 2 - 石头剪刀布 - 彩色数据预处理	188
30.1 介绍	188
30.2 PC 训练模型	188
30.3 模型的训练和保存	193
30.4 edge 调用	195
30.5 本章小结	201

31 AI 3 - 五花分类-从现有模型迁移训练	202
31.1 介绍	202
31.2 PC 训练模型	202
31.3 DNNDK 编译	210
31.4 EDGE 调用	210
32 AI 6 - 人脸口罩识别 - 综合案例	216
32.1 人脸口罩识别方案	216
32.2 原理说明	216
32.3 需要准备的东西	216
32.4 操作教程	216
第六部分 兼容拓展模块篇 - Extend	222
33 Extend 1 - IO_ 串口隔离板	223
34 Extend 2 - 电机驱动板	225
35 Extend 3 - 双舵机云台板	227
35.1 PWM 原理简述	227
35.2 舵机云台简介	227
35.3 舵机云台控制	231
36 Extend 4 - 16 路 PWM 驱动板	234
37 Extend 5 - TOF 测距板 - VL531X	235
37.1 tof 模块介绍	235
37.2 函数介绍	236
37.3 使用教程	236
38 Extend 6 - 红外温度测量阵列 - MLX90621	239
38.1 红外温度测量阵列模块介绍	239
38.2 函数介绍	240
38.3 使用教程	240
39 Extend 7 - 热成像仪 - Lepton 3.5	243
39.1 模块介绍	243
39.2 函数介绍	243
39.3 使用教程	244

40 Extend 8 - 2.8 寸液晶屏 - ili9341	247
40.1 原理介绍	247
40.2 函数介绍	248
40.3 使用教程	248

第一部分

入门篇 - Quick Start

第一章 Quick start 1 - 开机测试及整体介绍

内容提要

- 硬件组装
- 教程介绍
- 开机测试
- 问题答疑

PowerSensor 是我们正在打造的一款高性能、轻巧快捷、方便易用传感器算法开发平台，具有以下特点：

- 可以把他看作一个小巧的传感器，它的体积只有 7mmx5mmx2.5mm
- 也可以把它当成一个高性能的处理器，搭建 zynq70x0 系列 fpga，主频高达 767M，内存 1GB
- 它非常平易近人，只需要一台有无线网卡的电脑和浏览器就可以对它进行编程操作
- 它是站在巨人的肩膀上，预装了 opencv、zar、numpy 等常用库，只需要几行代码，新手即可实现各种炫酷功能
- 它具有极好的实时性和低延时，摄像头数据在 fpga 解析后通过 dma 直接存入 DDR 内存，比 usb 摄像头省去了许多中间环节
- 功耗低，炎炎夏日测到的峰值电流 0.8A，整体功耗不到 4W，使用配备的电池可以跑 3 个小时以上。
- 支持可配置的异步串口通信，TTL3.3 电平，与目前市面上大部分的单片机兼容

1.1 第一次开机

1.1.1 硬件检查

打开包装至少应该包含以下模块

1. usb 线，电源（5V， 2A）
2. sd 卡（32G）
3. wifi 模块
4. 主板
5. 摄像头板
6. 6 寸测试照片 10 张
7. molex 数据线 2 条（一条是两端 molex，一条是一端 molex 一端杜邦线）
8. 两排拓展排针
9. tf 卡读卡器
10. usb 转串口模块（cp2102）

本次开机实验需要其中 1-6 的物品。

1.1.2 硬件安装

1. powersensor 的常用硬件接口如图1.1所示：



图 1.1: 传感器接口示意图

2. 把 tf 卡插在主板上的 tf 卡座上
3. 把 wifi 模块插在主板上的 usb 插座上
4. 把传感器板安装在主板上（镜头出厂是对好焦的，请不要动）



图 1.2: 安装传感器

5. 把主板上的开关拨到 jupyter 那一边



图 1.3: 模式开关

6. 使用电源线给主板供电，然后 powersensor 就会启动

正常启动时，传感器板的彩灯会亮白色，启动完成时彩灯会变成蓝色，关于彩灯的说明：

表 1.1: LED 颜色的含义

白色	蓝色	绿色	红色
正在启动	Jupyter 模式	Offline 模式	故障



图 1.4: 正常启动处于 jupyter 模式下会亮蓝灯

1.1.3 wifi 连接及 jupyter 登入

- 打开电脑 wifi，连接那个以 powersensor-xxx 模样的 wifi，初始密码为 12345678。

笔记 Wifi 连接时如有询问是否允许设备发现，请点否，可以提高 wifi 传输速度。

- 正常连接 wifi 后，打开浏览器（建议使用 chrome），在浏览器里面键入

```

1 jupyter.powersensor.cn
2 # 如果域名不能正常解析的话请使用ip地址登入:
3 192.168.8.8

```

正常情况会需要输入密码，默认密码为 123。然后就可以进入这个 powersensor 的编程环境

1.2 软件测试

1.2.1 测试 1，jupyter 模式下

- 打开 jupyter 页面里的 Powersensor_quickStart.ipynb
- 按照文件中的提示，选中第一个程序框，并点击工具栏里的运行按钮
- 以同样的方法运行第二个程序框，可以看到实时刷新图像，刷新图像是非常消耗资源的，一般测试的时候需要显示，实际运行时要把图像刷新关掉，只输出最后处理的结果。



图 1.5: Powersensor 的 wif 连接结

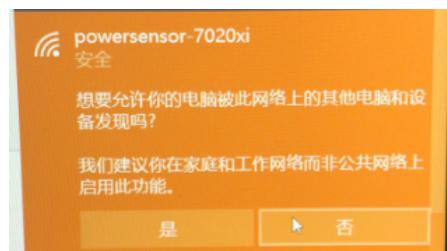


图 1.6: 关闭网络发现可以提升速度

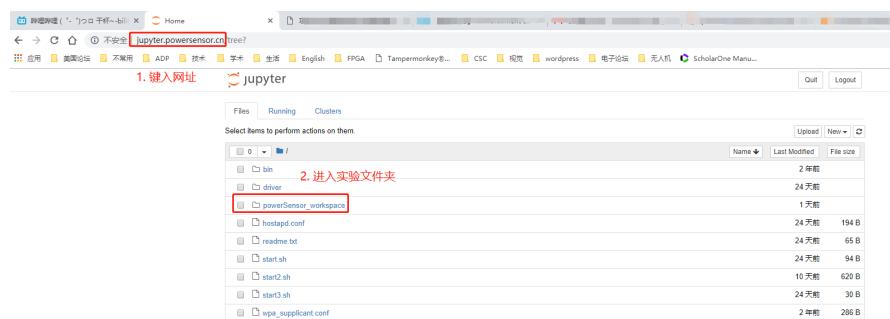


图 1.7: Jupyter 的主界面



Files Running Clusters Nbextensions

Select items to perform actions on them.

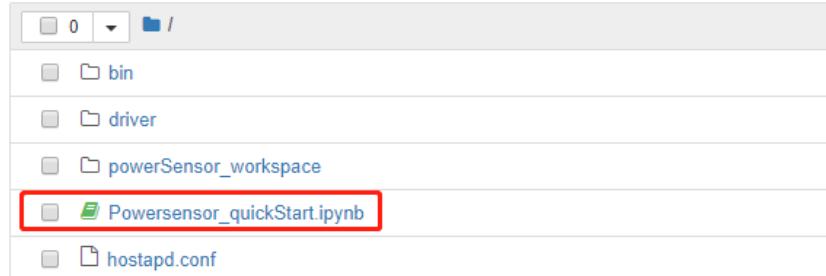


图 1.8: 第一次开机测试视频



图 1.9: 运行第一个程序

```
In [4]: for i in range(100):
    start = time.time() # 记录开始时间
    clear_output(wait=True) # 清除图片，在同一位置显示，不使用会打印多张图片
    imgMat = cam.readImgOrI() # 读入图像

    # 将小图放大为200x240尺寸
    origin = cv2.resize(imgMat, (320,240))

    ps.CommonFunction.show_im_jupyter(imgMat) # 打印用于差分的两张图片
    end = time.time() # 记录结束时间
    print(end - start)
    time.sleep(0.1)
```

图 1.10: 测试 1 结果

1.2.2 测试 2, jupyter 模式下

在运行完测试 1 后（其实测试 1 的前 2 步就够了），运行测试 2 下从程序框，这是一个 atrilTag 的检测实验：

1. 运行 Powersensor_quickStart.ipynb 中测试 2 下面的程序框
2. 拿出测试照片中的二维码的那一张，把 tag36h11 上方的 atriltag 放在画面中，成功识别的话，画面的下方会打印这个标记的横纵坐标：

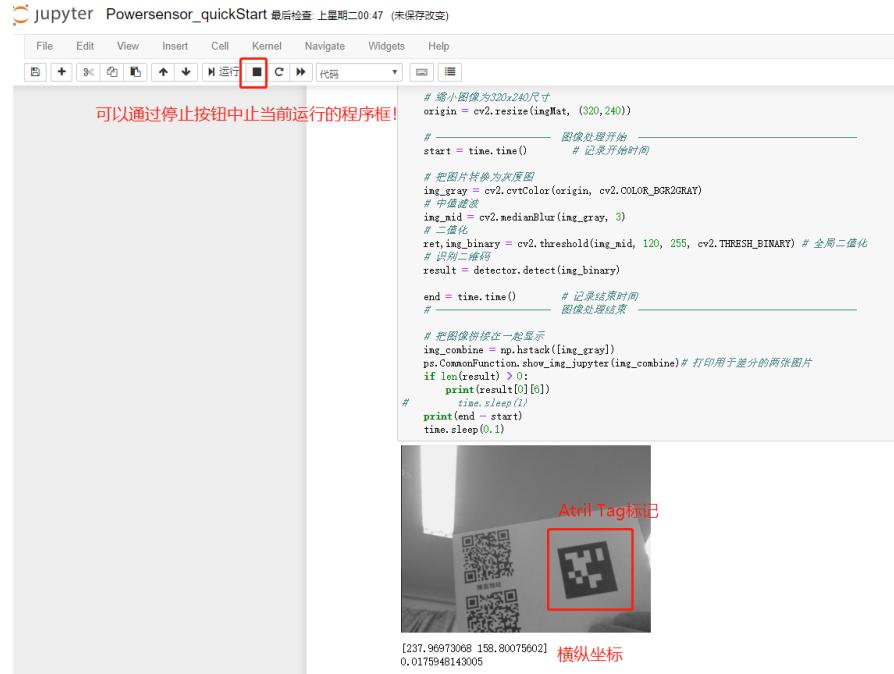


图 1.11: 测试 2 结果

3. 可以通过工具栏上的停止按钮强行停止这个运行框，同一个时刻只能由一个程序框正在运行。

1.3 教程介绍

验证完设备是否正常就可以开始愉快的学习了，powersensor 提供非常丰富的教程：《开发指导》（即本文档）、《API 手册》、视频教程（在 bili 上）。本教程是信息最丰富的教程，本教程分为 6 个篇章：

1. 入门篇，主要介绍如何连接启动 powersensor、如何刷最新的固件、通信接口等；
2. 图像处理基础篇，基础图像处理的原理和例程，适合零图像基础的用户；
3. 典型应用案例篇，以案例的形式进一步讲解具体的图像处理方法；
4. 进阶功能改装篇，适合高级用户，介绍 Powersensor 的网络接口，介绍 linux 系统操作的方法，提供 fpga 二次开发需要的信息；
5. 人工智能加速篇，介绍如何将 tensorflow 训练的模型部署到 powersensor 上运行，并使用 fpga 加速；
6. 兼容拓展模块篇，介绍 Powersensor 支持的拓展模块，如热成像仪、舵机板等；

为了兼顾教程质量与更新速度，本教程将提供三个版本：

1.3.1 博客版本教程

发布在京联博客的 powersensor 板块上，地址是

<http://www.jingliankeji.cn/powersensor/>

这个教程注重 powersensor 和图像处理的工作原理，更新最快，带的大部分程序可以直接运行。不过发布后不再修改，因此部分程序可能由于固件更新无法使用。

1.3.2 Ipython 版本教程

发布在 powersensor 的 github 上，地址是

https://github.com/powersensor-cn/powerSensor_workspace

出厂的 tf 卡里会带由一个较新的版本，在 powerSensor_workspace 目录下

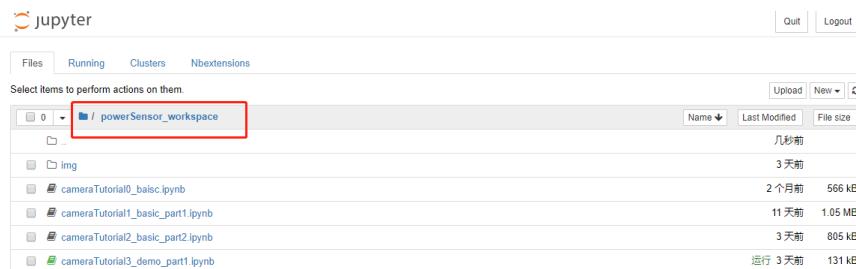


图 1.12: Ipython 教程的位置

这个教程更注重程序的实用性，更新较快，所有程序可以正确运行，发布后有 bug 的程序会优先更新。不过这个版本考虑实效性，只提供有限的原理介绍和注释。



笔记 Github 访问缓慢的小伙伴可以去 gitee 上下载，也会随之更新：

https://gitee.com/xiaobolin/powerSensor_workspace

1.3.3 Pdf 版本的教程

即本教程，以 pdf 文件的形式整理上述的两种教程，完成后发布在京联科技的论坛上，地址是：

<http://bbs.jingliankeji.com/forum.php?mod=forumdisplay&fid=57>

这个教程会同时兼顾原理解释和代码可行性，一个季度更新一个版本，系统性好，基本所有程序可以正确运行，发布后有 bug 的程序会在下一个版本更新。

V1.0 版 pdf 教程中的例程安排如下：

Tutorial 部分的 1-5 讲的例程在 /powerSensor_workspace/cameraTutorial_basic_part1.ipynb 中；

Tutorial 部分的 6-10 讲的例程在 /powerSensor_workspace/cameraTutorial_basic_part2.ipynb 中；

Demo 部分的例程在 /powerSensor_workspace/cameraTutorial_demo_part1.ipynb 中；

注 wifi 名称/密码设置，开机模式选择会在教程 cameraTutorial0_baisc 中进行介绍。

第二章 Quick start 2 - 最基本的功能

内容提要

- | | |
|-------------------------------------|---------------------------------|
| <input type="checkbox"/> 启动管理 | <input type="checkbox"/> Led 控制 |
| <input type="checkbox"/> Jupyter 基础 | <input type="checkbox"/> 视频保存 |
| <input type="checkbox"/> 串口读写 | |

这个教程提供了一些相对基础的 Powersensor 的功能，比如开机模式的选择、WIFI 密码修改、LED 灯的控制、串口通讯的控制、Jupyter 的使用等。

本实验的准备需要：

1. 需要安装浏览器，推荐 Chrome。（包装包在 SD 卡中）
2. PC（或平板）需要有无线网卡，用于连接 Powersensor 的无线网
3. Powersensor 传感器
4. PC 预装串口调试工具，推荐串口猎人（包装包在 SD 卡中）

2.1 启动管理

2.1.1 启动模式

Powersensor 有两种启动模式，Jupyter 模式和 Offline，它们有以下区别：

- Jupyter 模式（蓝灯）
 - 会启动 Jupyter Server 和 WIFI AP，这是日常开发的模式。在通过 WIFI 与 Powersensor 链接后，用户可以通过浏览器对 Powersensor 进行编程、调试。缺点在于这个模式的程序不会开机自动运行，还有这个模式的运行效率会低一点。
 - 在 Jupyter 模式中进行图像显示是非常消耗资源的，而且实际运行的时候一般不需要显示图片，所以估计计算速度的时候可以屏蔽掉图像显示的时间消耗。
- offline 模式（绿灯）
 - offline 模式不会启动 jupyter，而是开机直接运行 powersensor_workspace 目录下的 offline.py 文件，这是实际使用模式，运行性能比 jupyter 模式高（大概高 20%）。
 - 这个模式的编程需要提前在 jupyter 模式下把代码复制粘贴到 offline.py 文件中。
 - 这个模式一般使用串口把图像处理的结果（如中线位置、识别类型）输出给其他设备（如小车、无人机）使用。

笔记

1. 在 offline 模式下不要使用 print 函数，这个函数会把数据不断写入 sd 卡，严重降低效率。

2. 在 offline 模式下不能使用中文注释，会导致报错。
3. offline 模式可以在 Jupyter 下进行调试，在单元格里使用%run offline.py 可以运行 python 文件。
4. 在 offline 模式下不要使用图片显示函数，既不能显示图片，又十分浪费资源。
5. 模式的切换需要重启（重新上电），切换方法是上电前拨动主板的开关到指定位置即可。

2.1.2 Wifi 名称/密码修改

wifi 名称和密码的修改需要在 Jupyter 模式下进行，登入后打开主目录下的 hostapd.conf 文件



图 2.1: WIFI 配置文件

打开后可以看到以下内容，除了 WIFI 名称和 WIFI 密码外，其他设置请不要修改。

```

1 interface=wlan0
2 driver=n180211
3 ssid=powersensor-7020xi    # 这个是 WiFi 名称
4 channel=1
5 hw_mode=g
6 ignore_broadcast_ssid=0
7 auth_algs=1
8 wpa=2
9 wpa_passphrase=12345678    # 这个是 Wifi 密码
10 wpa_key_mgmt=WPA-PSK
11 wpa_pairwise=TKIP
12 rsn_pairwise=CCMP

```

修改后请断电 10s 才启动，让空间里的磁场休息一会儿。重启后需要用新的名称和密码来连接 Powersensor。

 **笔记** 请不要忘记设置的密码，后果很严重，准备刷机吧。

2.2 Jupyter 常见操作

jupyter 日常运行的程序文件是 ipython 文件，可以在首页下点击 new 按钮可以在当前目录下新建文件：



图 2.2: 新建 Ipython 文件

2.2.1 程序运行控制

新建或者打开已有的 ipython 文件后可以看到以下页面，首先我们来看工具栏的功能：



图 2.3: Jupyter 工具栏按钮

1. 运行按钮

选中一个单元格后，点运行按钮可以运行这个单元格，单元格运行时，左边会出现 *，等它变成数字时代表运行结束。

2. 停止按钮

按停止按钮可以手动停止正在运行的单元格，由于 python 的结构比较复杂，停止单元格不是立即完成的，等左边的 * 变成数字代表已经成功停止了。

3. 重启内核按钮

任何时候都可以点击重启内核按钮重启内核，这个操作会删除内存中所有已经定义的变量函数，重启后库也要重新引用。这个操作一般在感觉 Jupyter 死机或者运行严重报错的时候使用。任何时候都可以点击重启内核按钮重启内核，这个操作会删除内存中所有已经定义的变量函数，重启后库也要重新引用。这个操作一般在感觉 Jupyter 死机或者运行严重报错的时候使用。

4. 保存按钮

通过这个按钮可以保存当前的 Jupyter 文件，包括里面的代码和实验结果。

笔记 同一个时间只能有一个单元格处于运行状态，要等一个单元运行结束才能运行下一个单元格。

2.2.2 单元格间操作

Ipython 文件是由一个个单元格组成，每个单元格可以看作一段代码，单元格有很多种模式，常用的有两种 markdown 和 code。markdown 一般用来写注释内容，它是富文本

类型，可以插入图片公式等。code 模式就是拿来写 python 代码的。模式可以通过工具栏上右侧的下拉下单来选择（代码/markdown）。

Ipython 的单元格常用的有以下操作，下面列表序号与图2.4中的序号是一致的。

1. 复制粘贴单元格常用操作
2. 拆分单元格把光标定在一个单元格的某处，点击此按钮即可把这个单元格拆成两个
3. 合并单元格把当前单元格与上方或者下方单元格合并
4. 移动单元格向上或向下合并单元格



图 2.4: Jupyter 单元格操作

2.3 Powersenoser 的常用编程模板

本章教程所用的程序在 powerSensor_workspace 目录下的 cameraTutorial0_baisc.ipynb 中。

2.3.1 初始化

Python 语言的优势就在于大量已有的非常方便的库，Powersensor 官方也把传感器本身的操作封装到一个库（Powersensor）里，在运行其他程序前需要进行初始化。

1. import 需要的包

```

1 # numpy 是用于矩阵等科学计算的常用库
2 import numpy as np
3 # opencv 库
4 import cv2
5 # 画图库，相当于 matlab 的 plot 系列函数
6 import matplotlib.pyplot as pyplot
7 # 用于控制 jupyter 输出单元格的函数
8 from IPython.display import clear_output
9 # 时间相关的库
10 import time
11 # PowerSensor 库，包含这款传感器的常用传感器
12 import PowerSensor as ps
13

```

在执行其他语句前需要包含所需的库，由于 opencv 库比较大，所以要等大概 10s 左右等它运行完（左边的标识变成数字）。

2. 初始化 Powersensor 对象

```

1 # 这个对象用于操作powersensor摄像头
2 cam1 = ps.ImageSensor()
3 # 这个对象用于操作powersensor串口
4 s1 = ps.UsartPort()
5

```

以上两个是基本的 Powersensor 对象，初始化一次即可，通过这些对象可以方便地操作 Powersensor 的外设。

 **笔记** 初始化代码只需要运行一次就会被 ipython 记住，但是硬件重启以及点击 ipython 工具栏上的重启按钮后需要重新初始化。另外不同 ipython 文件的初始化是独立的。

2.3.2 摄像头读取与显示实验

这个实验主要是让用户熟悉如何读取 Powersensor 上摄像头的数据，在运行完上面的初始化后才可以读取摄像头的数据（在 Jupyter 运行期间初始化一起即可），把下面的代码复制到一个空白的 Jupyter 单元格中运行：

```

1 while(True):
2     start = time.time()
3     # 读取图像
4     imgMat = cam1.read_img_ori()
5     # 显示最终的图像
6     clear_output(wait=True)
7     ps.CommonFunction.show_img_jupyter(imgMat)
8     # 计算消耗时间
9     end = time.time()
10    print(end - start)

```

 **笔记** 这个代码的运行需要手动停止，即通过上面的停止按钮来停止。

这段代码用于读取摄像头数据，并把图像显示在 Jupyter 的输出单元格中：

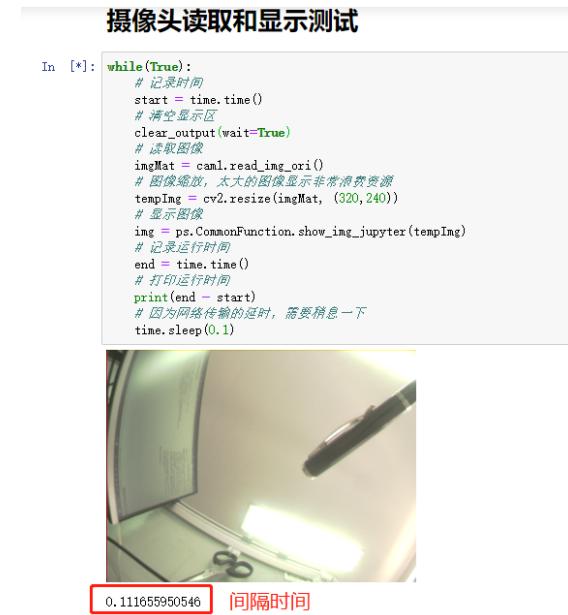


图 2.5: Powersensor 读取图像

2.3.3 OpenCV 轮廓提取实验

本实验在摄像头读取实验的基础上，调用 opencv 的函数对图像进行处理，并将结果展示出来。这个实验只是熟悉以下 opencv 的操作流程，详细的过程在后面的教程里会解释。

将下列代码复制粘贴到 Jupyter 的一个空白单元格后运行即可

```
1 while (True):
2     start = time.time()
3     imgMat = cam1.read_img_ori()
4     # 转换为灰度图
5     gray = cv2.cvtColor(imgMat, cv2.COLOR_BGR2GRAY)
6     # 二值化图片
7     ret, binary = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)
8     # 寻找图像轮廓
9     im2, contours, hierarchy = cv2.findContours(binary, cv2.RETR_TREE, cv2.
10 CHAIN_APPROX_SIMPLE)
11     img3 = cv2.drawContours(gray, contours, -1, ( 0, 128, 128 ), 3)
12     # 显示最终的图像
13     clear_output(wait=True)
14     ps.CommonFunction.show_img_jupyter(img3)
15     # 计算消耗时间
16     end = time.time()
17     print(end - start)
```

图像读取部分与上一个实验相同，不过这里要调用 opencv 函数进行图像处理，完成轮廓提取的工作：

注意：这个代码的运行需要手动停止，即通过上面的停止按钮来停止。



图 2.6: Powersensor-opencv 测试

其中黑色的框就是 cv 函数提取的轮廓。

2.3.4 摄像头视频保存实验

实验时在线预览视频是非常消耗计算资源和传输带宽的，为了分析实验数据，可以把视频数据先保存起来，实验过后再线下分析。

1. 键入所需代码并运行

```

1 # 指定编码方式
2 fourcc = cv2.VideoWriter_fourcc(*'MJPG')
3 # 设置保存的文件名，图像的格式
4 out1 = cv2.VideoWriter('output3.avi',fourcc, 20.0, (320,240))
5
6     # 保存的帧数
7 for i in range(200):
8     start = time.time()
9     #     clear_output(wait=True)
10    imgMat = cam1.read_img_ori()
11    # 缩放图像，以减小保存的文件体积
12    tempImg = cv2.resize(imgMat, (320,240))
13    out1.write(tempImg)
14    #     ps.CommonFunction.show_img_jupyter(tempImg)
15    # 显示图像
16    #     display(img)
17    end = time.time()
18    print(i, end - start)
19    time.sleep(0.1)
20 out1.release()
```

2. 程序在运行的过程中左边会显示一个星号

视频保存测试

```

In [*]: source = cv2.VideoCapture('output3.avi')
          out1 = cv2.VideoWriter('output1.avi',source, 20.0, (320,240))

# while(True):
#     for i in range(200):
#         start = time.time()
#         # clear_output(wait=True)
#         imgMat = cam.read_img ori()
#         r = imgMat[:, :, 0]
#         tempImg = cv2.resize(imgMat, (320,240))
#         out1.write(tempImg)
#         # ps.CommonFunction.show_img_jupyter(tempImg)
#         # display(img)
#         end = time.time()
#         print(end - start)
#         time.sleep(0.1)
#     out1.release()

```

保存的帧数

```

(0, 0.04307818412780762)
(1, 0.019251108169559664)
(2, 0.019633054733276367)
(3, 0.019633054733276367)
(4, 0.019633054733276367)
(5, 0.019633054733276367)
(6, 0.02035589671256826)
(7, 0.0196404941485957)
(8, 0.019744157791137695)
(9, 0.019496917724009375)
(10, 0.01956009884807129)
(11, 0.019057035446166992)
(12, 0.020489116409301758)

```

图 2.7: Powersensor-视频录取

3. 程序运行完成后，会在当前目录下生成 output3.avi 文件，下载到本地即可方便查看了



图 2.8: Powersensor 视频文件

2.3.5 LED 灯的闪烁控制实验

摄像头板上有一个彩色的 LED 灯，这个灯的颜色是由系统控制的，用于指示系统的状态，状态的解释见开机测试教程。而这个灯是闪烁是用户可以控制的。

led 灯的闪烁控制非常简单通过调用 `ps.CommonFunction.led_spark(flag)` 即可
注 flag 是标志，0 代表常亮；1 代表单闪；2 代表双闪；

在运行相关 import 后，调用 LED 闪烁控制代码如下：

```
ps.CommonFunction.led_spark(2)
```

2.4 IMU 读取实验

新型号的传感器板上配置了一个型号为 MPU9250 的 IMU，底层驱动已经做好，用户可以通过简单的函数读取 IMU 的数据。

2.4.1 原理讲解

欧拉角是一种常见的描述三维旋转的方式，IMU 返回的数据会先经过简单的处理，最终给用户使用的是三个欧拉角：

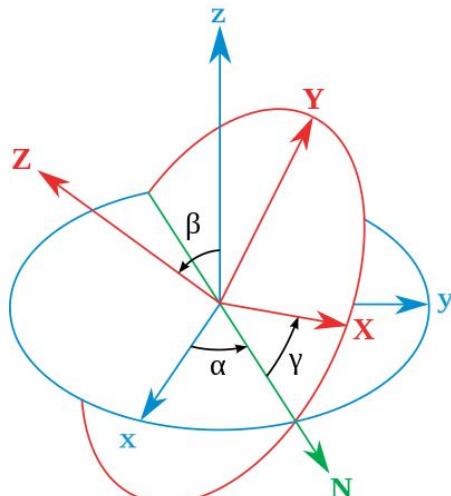


图 2.9: 欧拉角

三个欧拉角中的滚转角（Roll），俯仰角（Pitch）是通过加速度计和陀螺仪融合滤波的结果，相对稳定。偏航角单独由陀螺仪积分得到，存在飘逸现象，不能长时间使用。

2.4.2 函数介绍

```

1 # 类名: SensorConfig
2 # 与相机配置的类是同一个
3 #
4 # 构造函数: SensorConfig(), 无参数
5 #
6 #
7 # 读取函数: read_imu_att(),
8 # -return, 返回一个三维数组, 分别滚转、俯仰、偏航三个方向的角度值。
9 #

```

2.4.3 参考例程

为了方便展示效果，这里使用了 jupyter 的 ipywidgts 的滑块的来表示度数：

```

1 # 交互式控件用的用库
2 from ipywidgets import widgets
3 from IPython.display import display
4 sensor = ps.SensorConfig()
5
6 # 初始化
7 slider_roll = widgets.FloatSlider(
8     value=0,

```

```

9  min=-180, # max exponent of base
10 max=180, # min exponent of base
11 step=0.1, # exponent step
12 description='Roll'
13 )
14 slider_roll = widgets.FloatSlider(
15 value=0,
16 min=-180, # max exponent of base
17 max=180, # min exponent of base
18 step=0.1, # exponent step
19 description='Pitch'
20 )
21
22 # 显示
23 display(slider_roll)
24 display(slider_pitch)
25 for i in range(200):
26     # 读取 IMU 数据
27     res = sensor.read_imu_att()
28     # 更新滑块
29     slider_roll.value = res[0]
30     slider_pitch.value = res[1]
31     time.sleep(0.1)

```

运行效果如图所示

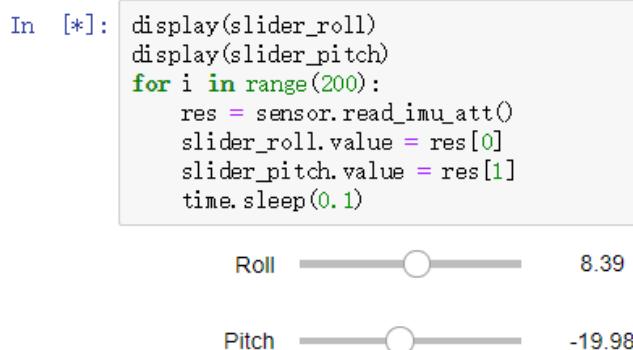


图 2.10: IMU 读取与控件展示效果

第三章 Quick start 3 - SD 卡固件烧写说明

内容提要

固件烧写

SD 卡

固件烧写是为 tf 卡 (micro SD 卡) 烧写新的固件，用于固件版本的更新。

 **笔记** 固件烧写会删除原有的所有数据，所以在烧写固件前请备份好自己的程序文件。

本实验的准备需要：

1. 固件一般放在百度网盘上，下载地址会随相应版本的教程在论坛上发布

<http://bbs.jingliankeji.com/forum.php?mod=forumdisplay&fid=57>

固件下载到的是一个 rar 压缩文件，需要使用 winrar 等解压工具，解压完成后会得到一个类似 psImg_xxxyy.img 的镜像，其中 xx 代表年份，yy 代表月份。

2. 烧录工具 Win32DiskImager，这个工具在原 tf 卡的 software 目录下有，或者去其官网下载：

<https://sourceforge.net/projects/win32diskimager/>

3. tf 卡读卡器，在盒子里有

3.1 烧写步骤

1. 为了防止误操作损坏您的 U 盘，请将其他 U 盘、移动硬盘等 USB 存储设备拔出。然后将 TF 卡装在读卡器里，插入电脑。会弹出无法读取、需要格式化等警告，点取消即可。

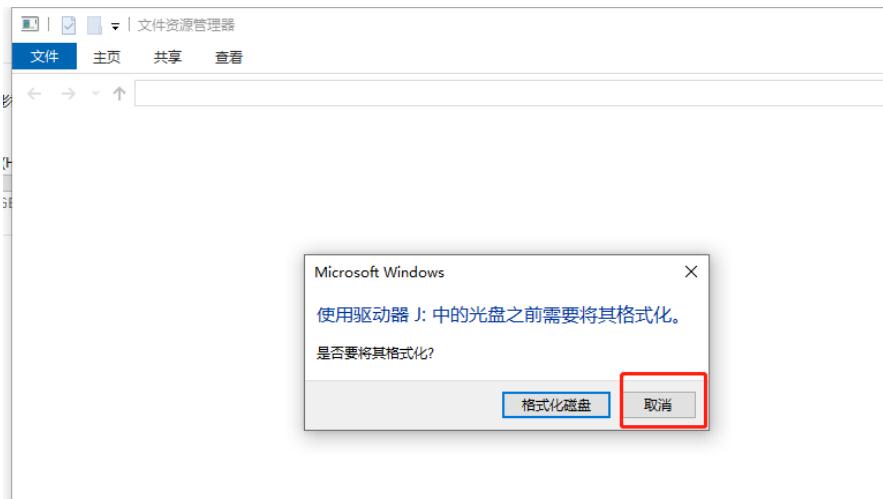


图 3.1：插入读卡器

2. 等 U 盘启动稳定后（大概 10 秒），启动 Win32DiskImager，选择镜像文件（即解压得到的 img 文件），选择设备（即 u 盘盘符，tf 卡对应有两个，选前面那个即可）。



图 3.2: 烧写镜像

3. 点击写入，等候大概 5 分钟即可完成。写入成功后会产生两个磁盘，一个是 Boot，一个 windows 下无法识别。Boot 那个磁盘里会有最新的教程，和常用软件。

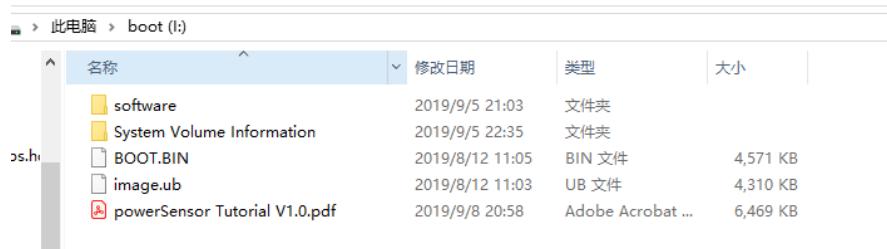


图 3.3: Boot 目录

到此固件烧写就完成了。

第四章 Quick start 4 - 通用通信接口

内容提要

- SPI 接口
- I2C 接口
- 串口
- PWM 输出

这个教程介绍了 Powersensor 的通信接口、背板接口的使用方法。

本实验的准备需要：

1. Powersensor 传感器
2. 背板需焊上排针（盒子里有排针）
3. usb 转串口模块（盒子里有）
4. I2C 无法回环测试，需要一个外设，Mpu6050 模块（自行采购，或其他 I2C 外设）。
5. 示波器，用于测量 PWM 生成的波形
6. 万用表用于验证普通 IO 的输出

4.1 Powersensor 的常用接口定义

4.1.1 背板排针接口定义

Powersensor 有 PWM 输出，SPI 接口，I2C 接口等通信方式，位置图如下：

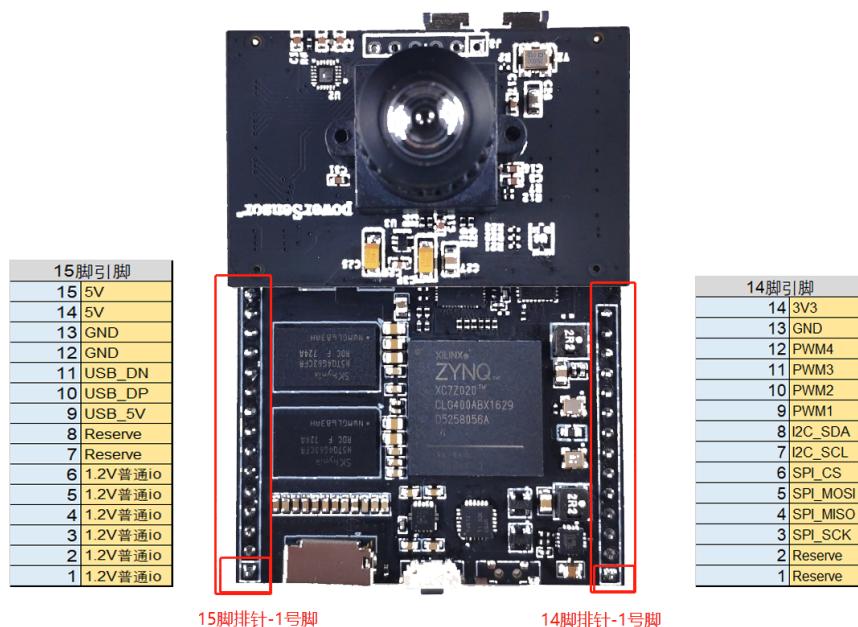


图 4.1：背板管脚定义



笔记 本章介绍的通信接口的电平都是 3.3V 的 TTL

4.1.2 串口接口定义

Powersenosr 支持串口输出，接口定义如图：

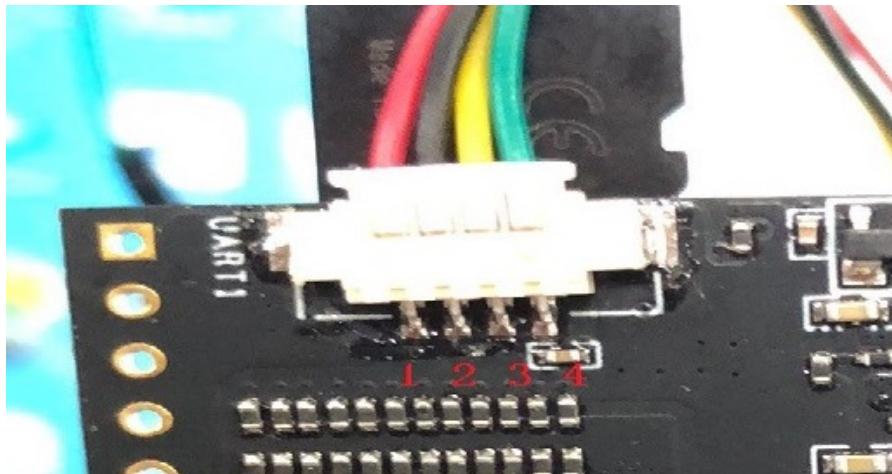


图 4.2: 串口管脚定义

从左往右 1, 2, 3, 4 依次为：

表 4.1: 默认串口参数

管脚号:	1	2	3	4
功能:	TXD0	RXD0	GND	3V3

4.2 SPI 通信

4.2.1 原理介绍

SPI(Serial peripheral interface) 即串行外围设备接口，SPI 通讯需要使用 4 条线：3 条总线和 1 条片选：

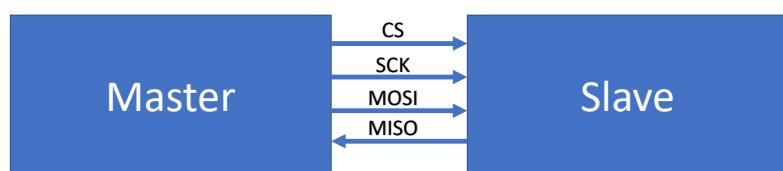


图 4.3: SPI 主从

SPI 还是遵循主从模式，主机提供 SCK、MOSI、CS（低电平有效），SPI 协议支持多个从机，但 Powersenosr 只提供了一个 CS 脚，所以只能接一个外设。

1. CS(Chip Select): 片选信号线，用于选中 SPI 从设备。每个从设备独立拥有这条 CS 信号线，占据主机的一个引脚。设备的其他总线是并联到 SPI 主机的，即无论多少

个从设备，都共同使用这 3 条总线。当从设备上的引脚被置拉低时表明该从设备被主机选中。

2. SCK(Serial Clock): 时钟信号线，通讯数据同步用。时钟信号由通讯主机产生，它决定了 SPI 的通讯速率。
3. MOSI(Master Ouput Slave Input): 主机 (数据) 输出/从设备 (数据) 输入引脚，即这条信号线上传输从主机到从机的数据。
4. MISO(Master Input Slave Ouput): 主机 (数据) 输入/从设备 (数据) 输出引脚，即这条信号线上传输从机从到主机的数据主从机通过两条信号线来传输数据。

SPI 协议根据时钟信号的时钟极性 (CPOL) 和时钟相位 (CPHA) 可以分为四种情况：

时钟极有两种选择，0 代表空闲状态 SCK 为低，1 代表空闲状态 sck 为高，时钟相位也有两种选择，0 代表第一个沿采样，1 代表第二个沿采样。两两结合，所以总共有四种情况。

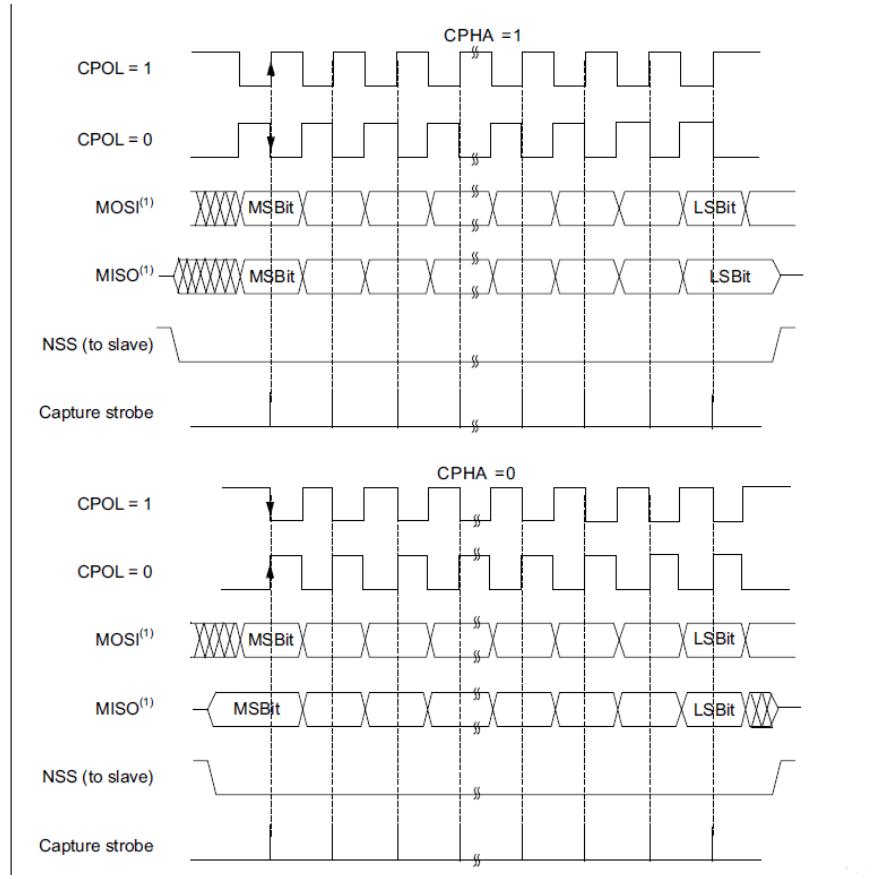


图 4.4: CPOL 和 CPHA 对 sck 的影响

注 Powersensor 目前支持的数据结构都是 byte 类型的（即 8 位的整形），数据是 MSB 的，就是最高位先发。

4.2.2 函数介绍

```

1 # 类名: SpiPort
2 #
3 # 构造函数: SpiPort(), 无参数
4 #
5 # 配置函数: set_clock(speed, cpol, cpha),
6 # -speed, 必须, 通信速率1Hz-10MHz
7 # -cpol, 必须, 时钟极性, 见原理介绍
8 # -cpha, 必须, 时钟相位, 见原理介绍
9 #
10 # 读写函数: swap_data(to_send),
11 # -to_send, 需要发送的数据, 必须是list类型, 数据类型为uint8
12 # -return, 返回值是接受到的数据, spi是双工的, 发送的同时会收到数据

```

4.2.3 参考例程

```

1 import PowerSensor as ps
2 # 初始化i2c, 运行一次即可
3 spi = ps.SpiPort()
4
5 spi.set_clock(1000000, 0, 0)
6
7 # 读写数据, 回环测试: 请将mosi和miso短接, 这样发送数据的时候就可以收到数据
8 # 读写的数据类型为list, dtype为uint8
9 to_send = [1, 3, 5, 7]
10 print(spi.swap_data(to_send))

```

单机实验时需要进行回环测试，把 SPI 接口的 MOSI 和 MISO 用杜邦线短接，运行程序即可看到接受到的发送的数据。

4.3 I2C 通信

4.3.1 原理介绍

I2C 是一种两线的总线结构通信协议，分主从机，一般一个总线上有一个主机和多个从机，写数据的主要过程如下：

1. 主机发出开始信号
2. 主机接着发出一字节的从机地址信息，包含 7 位从机地址，即最低位的读写信号（1 为读、0 为写）
3. 从机发出应答信号
4. 主机开始发送信号，每发完一字节后，从机发出应答信号给主机
5. 主机发出停止信号

时序图如下：

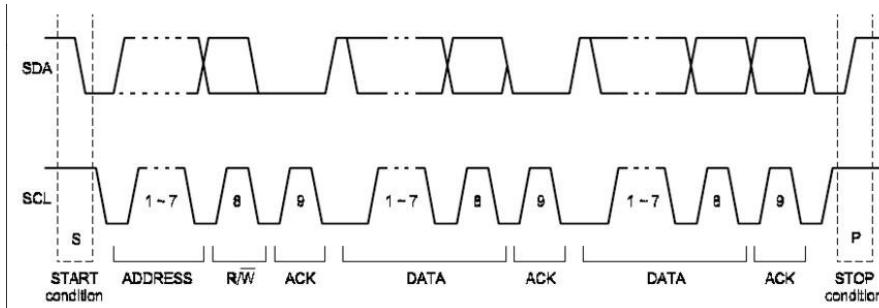


图 4.5: I2C 时序

读的过程与写数据的过程类似，不过在发完从机地址后，接的是重启动信号加地址+1（读信号），然后主机释放总线，由从机控制数据线，主机负责应答。

I2C 通信的详细过程可以上网搜索，不过现在这些过程一般都硬件实现了，Powersensor 只需要通过简单的函数调用即可完成 I2C 通信。

 **笔记** Powersensor 支持的从机地址是 7 位的。考虑到兼容性，Powersensor 的 I2C 管脚没有焊上拉电阻，因此在使用时需要根据实际情况外接合适的上拉电阻。

4.3.2 函数介绍

```

1 # 类名: I2cPort
2 #
3 # 构造函数: I2cPort(), 无参数
4 #
5 # 配置函数: set_slave_addr(slave_addr),
6 # -slave_addr, 必须, 设置从机地址
7 #
8 # 单字节写函数: write_byte(reg_addr, value),
9 # -reg_addr, 需要写入的地址 (寄存器地址, 不是从机地址)
10 # -value, 需要写入的值
11 #
12 # 单字节读函数: write_byte(reg_addr),
13 # -reg_addr, 需要读的地址 (寄存器地址, 不是从机地址)
14 # -retuan, uint8 类型, 读到的数据
15 #
16 # 块写函数: write_block(reg_addr, data),
17 # -reg_addr, 需要写入的地址 (寄存器地址, 不是从机地址)
18 # -data, 需要写入的数组, list 类型, 数据类型位 uint8
19 #
20 # 块写函数: read_block(reg_addr, data, num),
21 # -reg_addr, 需要读取的地址 (寄存器地址, 不是从机地址)
22 # -num, 需要读取的数据的数量
23 # -return, 返回的读取的数据

```

4.3.3 参考例程

I2C 无法回环测试，需要外接一个设备，这里以 MPU6050 举例，示例程序如下

```

1 import PowerSensor as ps
2 # 初始化 i2c，运行一次即可
3 i2c = ps.I2cPort()
4
5 # 配置从机地址，注意，这里使用的是7位的从机地址，
6 # mpu6050 文档上写的从机地址是 0xD0，这个是8位地址，需要右移一位变成 0x68
7 i2c.set_slave_addr(0x68)
8
9 # 这里读取 mpu6050 的 ID 寄存器，这个寄存器是只读的，值为 104
10 imu_id = i2c.read_byte(117)
11 print(imu_id)

```

4.4 串口通信

异步串口接口相关的函数封装在 `UsartPort()` 类中。

4.4.1 原理介绍

串口是 powersensor 把图像处理的结果（跟踪物体的位置，速度，二维码读取结果等）交付给单片机/pc 等设备的重要方式。

串口接口的接线顺序见本章的第一节 4.1.2，串口接线时注意 TX 接 RX，RX 接 TX。

Powersenosr 的串口是一种单机对单机的双线全双工异步通信方式，没有时钟线，双方通过波特率校准，因此需要较高的时钟精度，默认的串口参数是

表 4.2: 默认串口参数

项	项	项	值
波特率:	115200	奇偶校验:	无校验
数据长度:	8	停止位:	1

串口接口的接线顺序见本章的第一节，串口接线时注意 tx 接 rx，rx 接 tx。

4.4.2 函数介绍

```

1 # 类名: UsartPort
2 #
3 # 构造函数: UsartPort(), 无参数
4 #
5 # 波特率设置函数: set_baudrate(baudrate),
6 # -baudrate, 必须, 波特率
7 #

```

```

8 # 字符串打印函数: u_print(str),
9 # -str, 需要打印的字符串
10 #
11 # 字节数组打印函数: u_send_bytes(arr),
12 # -arr, 需要输出的字节数组
13 #
14 # 按字节数读取函数: ul.read(num),
15 # -num, 需要读取的字节数
16 #
17 # 读取缓存区所有字节函数: ul.read_all(),
18 # -无参数

```

4.4.3 参考例程

- 字符串输出实验，直接用 print 输出 ASCII 字符串，适合与 pc 通信

```

1 # 设置波特率，只需要设置一次，默认为115200
2 s1 = ps.UartPort()
3 s1.set_baudrate(115200)
4 for i in range(2):
5     s1.u_print('hello world!\n')
6

```

实验结果：



图 4.6: Powersensor-串口测试 2

笔记 注意串口调试助手要设置正确的参数。

- 字节数组输出实验，输出可以自定义数据包结构的字节数组，适合与单片机通信

```

1 # 设置波特率，只需要设置一次，默认为115200

```

```

2 s1.set_baudrate(9600)
3 for i in range(2):
4     s1.u_send_bytes([1, 3, 5, 7, 9])
5

```

结果如下所示

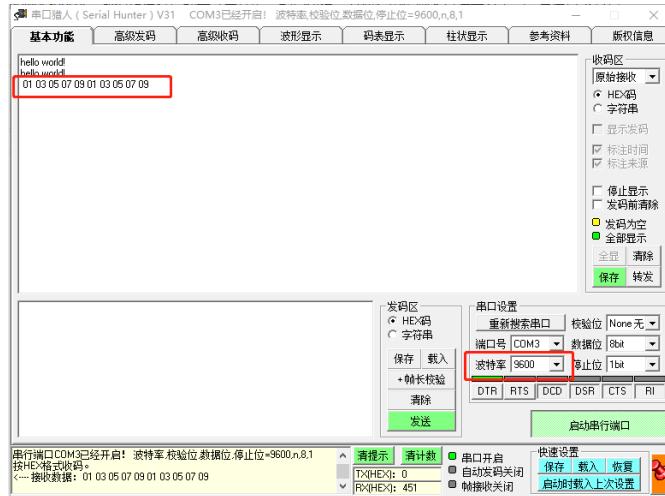


图 4.7: Powersensor-串口测试

这里只是举例，输出的是 1, 3, 5, 7, 9 数组，正常与对单片机通信时应该有完整的包头包尾校验字，方便单片机捡包和校验，如

```

1 pack = np.array([0xa5, 0x08, 数据0, 数据1, 数据2, 数据3, 数据4, 数据5,
      校验)
2

```

注 其中 0xa5 是包头，0x08 是包的长度，校验可以是 5 个数据的求和的后 8 位。

4.5 PWM 输出

4.5.1 原理介绍

PWM 脉冲宽度调制技术，是一系列可以对脉冲的宽度进行调制的方波，来等效地获得所需要波形（含形状和幅值），在电机控制、led 亮度等场合有广泛的应用。

PWM 波有几个重要的概念：

1. 时钟源 (设频率为 f_s)，PWM 一般是一种频率固定，高电平时间可以调节的矩形波，时钟源决定 PWM 的精度、最高频率。Powersensor 的 PWM 时钟源是 100Mhz
2. 预分频 (设为 n_f)，由于时钟源的频率较高，为了计算方便以及得到低频的信号，会使用预分频器对时钟源进行分频。Powersensor 的预分频器是 16 位的。
3. 计数器，计数器的单位是时钟源经过预分频后的时钟信号的周期，Powersensor 使用的计数器是 32 位的。
4. 周期/频率 (设周期为 n_p)，周期是指生成的 PWM 的一个高电平和一个低电平的总时长，单位是计数器计数的个数，频率是周期的倒数。

5. 高电平时间/占空比，高电平时间是指 PWM 一个周期内信号的高电平时间，单位是计数器计数的个数，占空比是高电平时间与周期的比值。

PWM 波的频率与时钟源的关系为：

$$f_{pwm} = \frac{f_s}{(n_f + 1)(n_p + 1)}$$

PWM 产生的时序图如下所示：

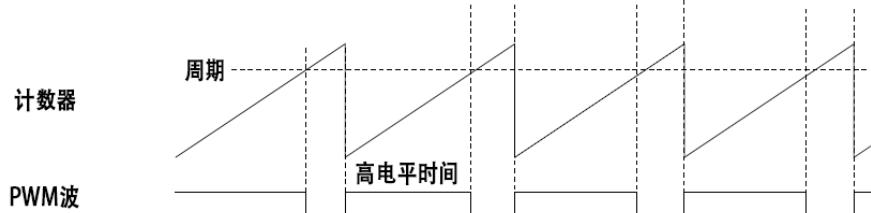


图 4.8: PWM 时序

在电机控制的半桥电路中，一般需要两个极性相反的 PWM 来控制两个 mos 管的开关，为了防止瞬间出现的上下两个 mos 同时开启导致的电源短路问题，一般要设置死区，死区的单位也是计数器计数的个数，死区就是高电平前的延时时间（为了等待另一路 mos 管彻底关闭）。Powersensor 提供了两种 PWM 模式，普通模式和互补模式，普通的模式是 4 路周期固定，脉冲宽度可以调节的 PWM 波，互补模式是两路（也是 4 个信号）宽度可调节的 PWM 信号，每路含两个极性相反的 PWM 信号，同时带死区控制。带死区控制的波形如下所示：

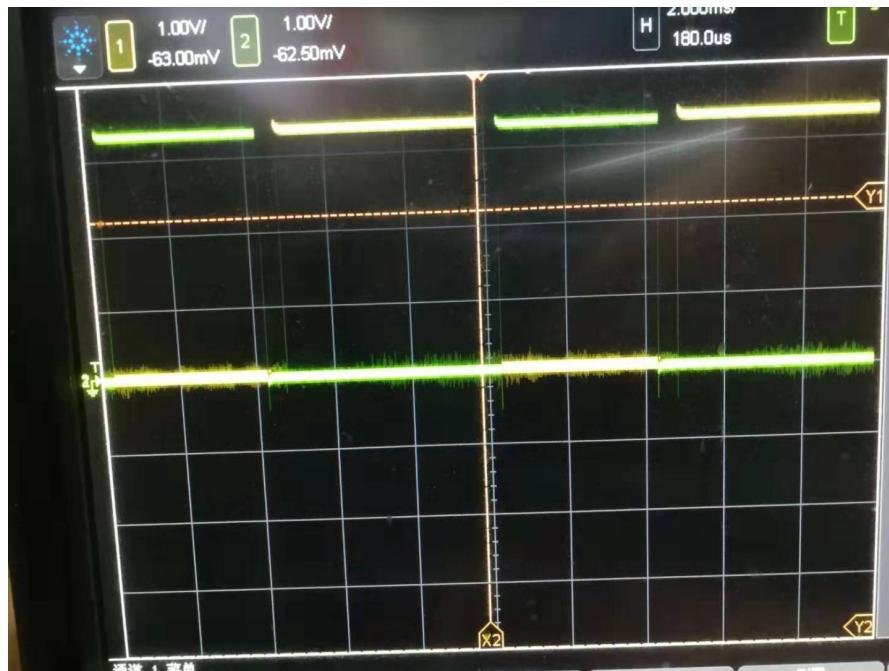


图 4.9: 互补 PWM

4.5.2 函数介绍

```
# 类名: PwmActuator
```

```

2 # _____
3 # 构造函数：PwmActuator()，无参数
4 #
5 # 属性：
6 # -preDividor, 预分频，16位整形
7 # -period, 周期，32位整型
8 # -mode, 模式，0关闭输出，1普通模式，2互补输出模式
9 # -preDeadZoneee, 死区时间，互补模式下起作用
10 # -pwm0,pwm1,pwm2,pwm3, 高电平时间，整形，最大值位周期，互补模式时只有pwm0和
11 #     pwm2起作用
12 #
13 # 配置函数：pwm_setup(enable),
14 # -enable, 必须，设置为True时，上述配置的属性才会起作用，否则输出低电平
15 #
16 # 通道高电平时间设置：pwm_set_width(chanel, width),
17 # -chanel, 通道选择，0~3
18 # -width, 高电平时间，整形，最大为周期

```

4.5.3 参考例程

```

1 import PowerSensor as ps
2 # 初始化pwm对象，运行一次即可
3 pwmx = ps.PwmActuator()
4
5 # 普通模式
6 #
7 # 预分频，实际分频数=设定值+1
8 pwmx.preDividor = 10 - 1
9 # 死区，实际的周期=设定值+1
10 pwmx.period = 1000 - 1
11 # 高电平时间，最大为周期的设定值，即999
12 pwmx.pwm0 = 200
13 pwmx.pwm1 = 400
14 pwmx.pwm2 = 600
15 pwmx.pwm3 = 800
16 # 模式设置，0：低电平；1：普通模式；2：互补输出模式；
17 pwmx.mode = 1
18 # 写入设置，写入这句后，上述配置的内容才起效
19 pwmx.pwm_setup(True)
20
21 # 互补模式
22 #
23 # 互补模式下的pwm输出
24 # 预分频，实际分频数=设定值+1
25 pwmx.preDividor = 10 - 1
26 # 周期，实际的周期=设定值+1

```

```

27 pwmx.period = 1000 - 1
28 # 死区，即互补的两个信号的高电平间的间隔，实际值为设定值+1
29 pwmx.preDeadZonee = 50 - 1
30 # 互补滤波通道1，高电平时间，最大为周期的设定值，即999
31 pwmx.pwm0 = 200
32 # 互补滤波通道2，高电平时间，最大为周期的设定值，即999
33 pwmx.pwm2 = 600
34 ## 此模式下，pwm1和pwm3不起作用
35 # 模式设置，0：低电平；1：普通模式；2：互补输出模式；
36 pwmx.mode = 1
37 # 写入设置，写入这句后，上述配置的内容才起效
38 pwmx.pwm_setup(True)
39 # 预分频，实际分频数=设定值+1
40 pwmx.preDividior = 10 - 1
41
42 # 动态改变高电平时间
43 #
44 pwmx.pwm_set_width(0, 300)

```

4.6 普通 IO

4.6.1 原理介绍

对应的硬件是排针引脚上的 6 个 1.2V 的普通 IO。普通 IO 有输出和输入两种模式，输出模式下可以输出高电平或者低电平，输入模式下可以读取外界输入的电平。

 **笔记** 这几个 Gpio 是 1.2V 的，不要输入过高电平，若需要其他电平请设置电平转换电路。
一个简单的参考电路如下：

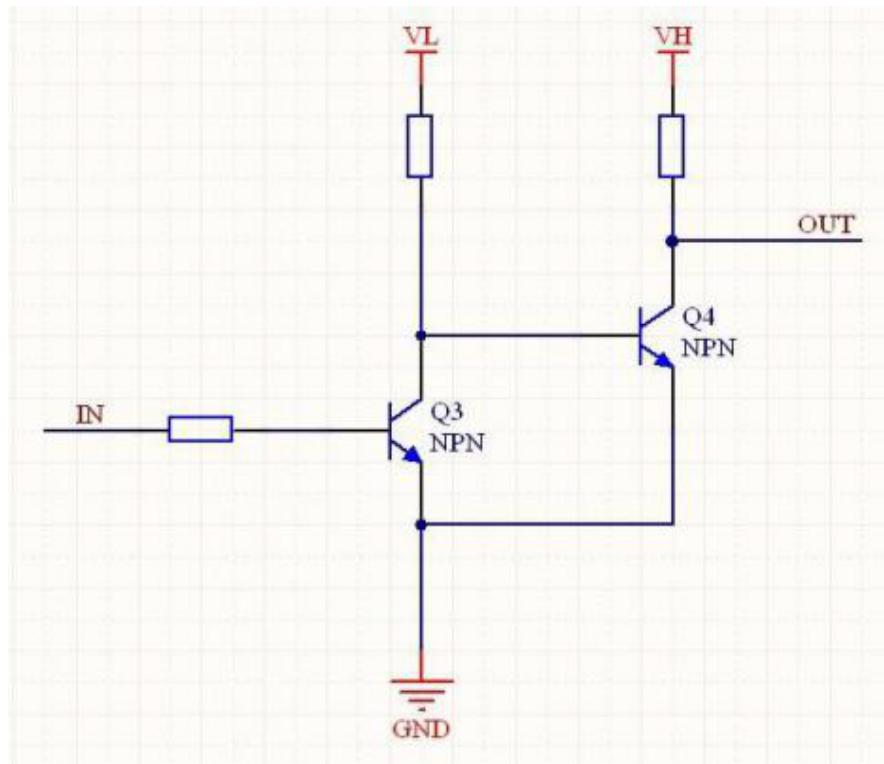


图 4.10: 输出电平转换电路

4.6.2 函数介绍

```

1 # 类名: GpioPort
2 #
3 # 构造函数: GpioPort(), 无参数
4 #
5 # 配置函数: set_mode(pos, mode),
6 # -pos, 必须, 管脚号, 从0开始, 到5结束
7 # -mode, 必须, 可以是.PortPara.Gpio_Mode_Out 和 PortPara.Gpio_Mode_In 中的一个,
8 # 即输出模式或输入模式
9 #
10 # 配置函数: set_value(pos, value),
11 # 仅对输出模式的管脚有效
12 # -pos, 必须, 管脚号, 从0开始, 到5结束
13 # -value, 必须, 可以是.PortPara.Gpio_Value_High 和 PortPara.Gpio_Value_Low 中的
14 # 一个, 即输出高电平或低电平
15 #
16 # 电平读取函数: read_value(pos),
17 # 仅对输入模式的管脚有效
18 # -pos, 必须, 管脚号, 从0开始, 到5结束
19 # -return, 返回值, 高电平返回1, 低电平返回0
20 #

```

4.6.3 参考例程

```
1 # 初始化控制对象
2 gpiox = ps.GpioPort()
3
4 # 设置输出
5 gpiox.set_mode(1, ps.PortPara.Gpio_Mode_Out)
6 gpiox.set_mode(3, ps.PortPara.Gpio_Mode_Out)
7 gpiox.set_mode(4, ps.PortPara.Gpio_Mode_Out)
8 gpiox.set_mode(5, ps.PortPara.Gpio_Mode_Out)
9
10 gpiox.set_value(1, ps.PortPara.Gpio_Value_High)
11 gpiox.set_value(3, ps.PortPara.Gpio_Value_Low)
12 gpiox.set_value(4, ps.PortPara.Gpio_Value_Low)
13 gpiox.set_value(5, ps.PortPara.Gpio_Value_High)
14
15 # 设置输入
16 gpiox.set_mode(0, ps.PortPara.Gpio_Mode_In)
17 gpiox.set_mode(2, ps.PortPara.Gpio_Mode_In)
18 # 设置电平
19 for i in range(15):
20     x1 = gpiox.read_value(0)
21     x2 = gpiox.read_value(2)
22     print(x1, x2)
23     time.sleep(1)
```

输出模式的管脚可以通过万用表测量来验证程序，注意第一个管脚是 0 号脚。

输入模式可以这样验证：输入程序是进行一秒一次的轮询管脚读取，可以使用金属短路管脚，0-1，或者 1-2 管脚，此时读书会变成 1（因为上一段程序把 1 号脚设成输出，输出高电平）。

第二部分

图像处理基础篇 - Image Basic

第五章 Tutorial 1 - 在图像上做标记

内容提要

□ 图片读取

□ 在图片上做标记

本章实验需要的配件:

- Powersensor 传感器

5.1 图片读取

在图像处理过程中，有时我们需要用一些本地的静态的图片做测试，使用 OpenCV 来读取本地的图片是一个必备的技能。

5.1.1 函数介绍

OpenCV 的图片读取函数是 cv2.imread(), 原型如下:

```
1 cv2.imread(filename[, flags]) → retval
2 # -filename, 必须, 是图片的路径+文件名;
3 # -flags, 可选, 读取方式, 可以是CV_LOAD_IMAGE_GRAYSCALE,
4     CV_LOAD_IMAGE_ANYDEPTH, CV_LOAD_IMAGE_COLOR 中的一个;
5 # 其中CV_LOAD_IMAGE_GRAYSCALE是灰度图(2维), CV_LOAD_IMAGE_COLOR的是彩色图
6     (3维), CV_LOAD_IMAGE_COLOR 用来读高清图
7 # -retval, 返回值是读取到的图像, 是一个数组, 可以使用numpy的函数来操作它
```

cv2.imread() 支持的图片格式有:

- Windows 位图 - *.bmp, *.dib (always supported)
- JPEG 文件 - *.jpeg, *.jpg, *.jpe
- JPEG 2000 文件 - *.jp2
- Portable Network Graphics - *.png
- web 图像 - *.webp
- Portable image format - *.pbm, *.pgm, *.ppm
- Sun rasters - *.sr, *.ras
- TIFF files - *.tiff, *.tif

注 cv2.imread() 根据图片的拓展名来选择读取的方式。

5.1.2 参考例程

读取本地图像的示例代码如下:

```
1 img = cv2.imread("./img/flower.jpg")
2 img_w = img.shape[1]
3 img_h = img.shape[0]
```

```

4 # 缩放图片
5 smallImg = cv2.resize(img, (img_w / 2, img_h / 2))
6 ps.CommonFunction.show_img_jupyter(smallImg)
7 b = smallImg[:, :, 0]
8 g = smallImg[:, :, 1]
9 r = smallImg[:, :, 2]
10 ps.CommonFunction.show_img_jupyter(b)
11 ps.CommonFunction.show_img_jupyter(g)
12 ps.CommonFunction.show_img_jupyter(r)

```

这里假设 ipython 程序是在 powerSensor_workspace 目录下，上传的图片是这个当前目录的 img 目录下。

Jupyter 上传图片的方法是切换到首页，然后按图5.1所示操作，上传的文件会放在 Jupyter 的当前目录：



图 5.1: Jupyter 上传图片

5.1.3 结果展示

结果如图5.4所示：



图 5.2: 显示本地图片结果

5.2 在图片上做标记

在图像处理的调试过程中，经常需要框出特定区域，或者在图片上标注类型等信息，这就是在图片上做标记。OpenCV 提供了大量的绘图函数，比较常用的有画线、矩形、圆圈、椭圆、多边形、文本等，以下将进行逐个介绍。

5.2.1 函数介绍

1. 画线

```

1 cv2.line(img, pt1, pt2, color[, thickness[, lineType[, shift]]]) ->
2     img
3     # -img, 必须, 原图像
4     # -pt1/2, 必须, 第一/二个点的坐标
5     # -color, 可选, 颜色
6     # -thickness, 可选, 线的粗细 (运行浮点, 但是会被取整)
7     # -lineType, 可选, 线的类型, 可以是LINE_8 (默认), LINE_4, LINE_AA
8     # -shift, 可选, Number of fractional bits in the point coordinates
9     # img, 返回值, 返回值直接作用于原图

```

2. 画矩形

```

1 cv2.rectangle(img, pt1, pt2, color[, thickness[, lineType[, shift]]]) -> img
2     # -img, 必须, 原图像
3     # -pt1/2, 必须, 第一点的坐标, 对角点的坐标
4     # -color, 可选, 颜色
5     # -thickness, 可选, 线的粗细, 负数会导致填充
6     # -lineType, 可选, 与line的解释相同
7     # -shift, 可选, Number of fractional bits in the point coordinates
8     # img, 返回值, 返回值直接作用于原图
9

```

3. 画圆圈

```

1 cv2.circle(img, center, radius, color[, thickness[, lineType[, shift]]]) -> img
2     # -img, 必须, 原图像
3     # -center, 必须, 中心点的坐标
4     # -radius, 必须, 半径大小
5     # -color, 可选, 颜色
6     # -thickness, 可选, 与矩形的解释相同
7     # -lineType, 可选, 与line的解释一样
8     # -shift, 可选, Number of fractional bits in the point coordinates
9     # img, 返回值, 返回值直接作用于原图
10

```

4. 画椭圆。椭圆的绘图原理见注解。

```

1 cv2.ellipse(img, center, axes, angle, startAngle, endAngle, color
2     [, thickness[, lineType[, shift]]]) -> img
3     # -img, 必须, 原图像
4     # -center, 必须, 中心点的坐标
5     # -axes, 必须, 长轴和短轴的半径

```

```

5      # -angle, 必须, 绕中心旋转的角度
6      # -startAngle, 必须, 可以画圆弧, 弧线开始的角度
7      # -endAngle, 必须, 可以画圆弧, 弧线结束的角度
8      # -color, 可选, 颜色
9      # -thickness, 可选, 与矩形的解释相同
10     # -lineType, 可选, 与line的解释一样
11     # -shift, 可选, Number of fractional bits in the point coordinates
12     # img, 返回值, 返回值直接作用于原图
13

```

5. 画多边形

```

1      cv2.polyline(img, pts, isClosed, color[, thickness[, lineType[, shift]]]) → img
2          # -img, 必须, 原图像
3          # -pts, 必须, 沿途各点的坐标
4          # -isClosed, 必须, 多边形十分封闭, True会自动把最后一个点和第一个
5              点连起来
6          # -color, 可选, 颜色
7          # -thickness, 可选, 与矩形的解释相同
8          # -lineType, 可选, 与line的解释一样
9          # -shift, 可选, Number of fractional bits in the point coordinates
10         # img, 返回值, 返回值直接作用于原图
11
12
13

```

6. 添加文本

```

1      cv2.putText(img, text, org, fontFace, fontScale, color[, thickness
2          [, lineType[, bottomLeftOrigin]]]) → None
3          # -img, 必须, 原图像
4          # -text, 必须, 想添加的文本, 必须string类型,
5          # -org, 必须, 文本左下角的坐标
6          # -fontFace, 必须, 字体类型, 可以是FONT_HERSHEY_SIMPLEX,
7              FONT_HERSHEY_PLAIN, FONT_HERSHEY_DUPLEX, FONT_HERSHEY_COMPLEX,
8              FONT_HERSHEY_TRIPLEX, FONT_HERSHEY_COMPLEX_SMALL,
9              FONT_HERSHEY_SCRIPT_SIMPLEX, or FONT_HERSHEY_SCRIPT_COMPLEX, 中的一个
10             # -fontScale, 必须, 字体大小
11             # -color, 可选, 颜色
12             # -thickness, 可选, 与矩形的解释相同
13             # -lineType, 可选, 与line的解释一样
14             # -shift, 可选, Number of fractional bits in the point coordinates
15
16             # -bottomLeftOrigin, 可选, 原点位置, True, 起点在左下角, False, 起
17                 点在左上角
18             # img, 返回值, 返回值直接作用于原图
19
20
21
22
23

```

注 LINE_8, LINE_4, LINE_AA, 分别是 4/8 连通线、抗锯齿线, 详情见[连接方式介绍](#)

注 椭圆绘图的原理:

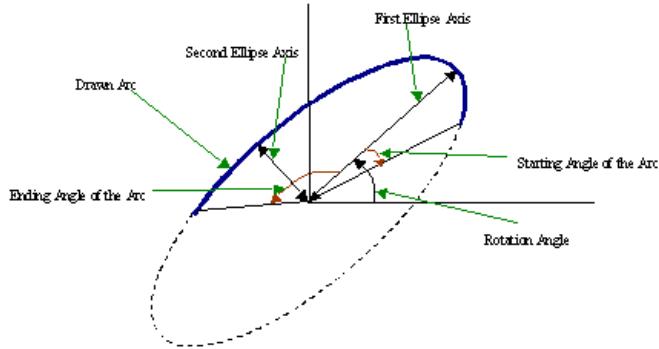


图 5.3: 显示本地图片结果

5.2.2 参考例程

首先使用 Powersensor 读取实时的图像，然后在当前图像上做标记：

```

1 for i in range(100):
2     start = time.time()          # 记录开始时间
3     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
4     imgMat = cam1.read_img_ori()  # 读入图像
5
6     # 缩小图像为320x240尺寸
7     tempImg = cv2.resize(imgMat, (320,240))
8
9     # 1. 在tempImg图像上划线，从(10,10)到(200,200)坐标，绿色，3个像素宽度
10    cv2.line(tempImg, (10,10), (200,200), (0,255,0), 3)
11
12    # 2. 在tempImg图像上画矩形，(10,10)为左上顶点，(30,40)为右下顶点
13    cv2.rectangle(tempImg, (200,200), (300,210), (0,0,255), 1)
14
15    # 3. 画圆，参数依次为圆心、半径、颜色BGR, 填充-1或线宽像素n
16    cv2.circle(tempImg, (100,200), 30, (0,0,213), -1)
17
18    # 4. 画椭圆_需要输入中心点位置，长轴和短轴的长度，
19    cv2.ellipse(tempImg, (160,120), (100,50), 0, 0, 360, (20,213,79), 2) # 椭圆沿逆时
        针选择角度，椭圆沿顺时针方向起始角度和结束角度
20
21    # 5. 绘制多边形
22    pts=np.array([[10,3],[60,3],[48,19],[98,19]],np.int32) # 数据类型必须是
        int32
23    pts=pts.reshape((-1,1,2))
24    # 这里 reshape 的第一个参数为-1，表明这一维的长度是根据后面的维度的计算出来
        的。
25    # 如果第三个参数是 False，我们得到的多边形是不闭合的（首尾不相连）。
26    cv2.polyline(tempImg,[pts],True,(0,0,255),1) # 图像，点集，是否闭合，颜
        色，线条粗细
27

```

```
28 # 6. 添加文字，参数：绘制的文字，位置，字型，字体大小，文字颜色，线型
29 font=cv2.FONT_HERSHEY_SIMPLEX
30 cv2.putText(tempImg, 'OpenCV', (110, 60), font, 1, (255, 255, 255), 2)
31
32 end = time.time()
33 ps.CommonFunction.show_img_jupyter(tempImg)
34 print(end - start)
35 time.sleep(0.1)
```

5.2.3 结果展示

实际运行时效果如下图所示：

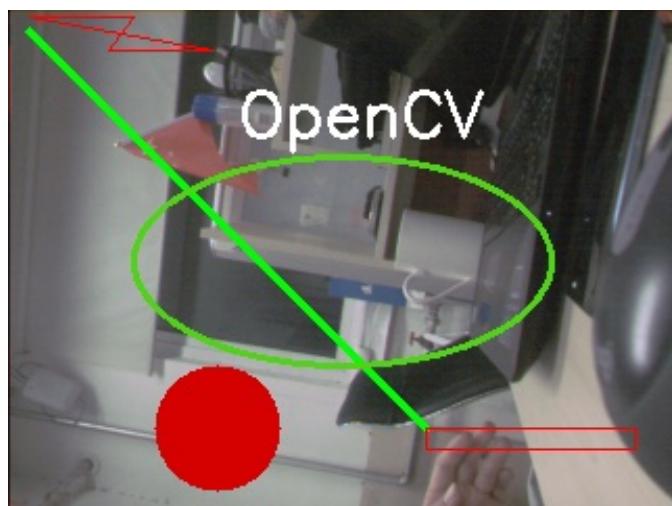


图 5.4: 做标记实验效果

5.3 小结一下

结论 图片读取、在图像上做标记都是非常基础简单的功能，但是使用又非常经常，可以作为字典在需要的时候翻阅。

第六章 Tutorial 2 - 图像预处理（一）常见变换

内容提要

- 平动变换
- 旋转变换
- 色彩空间

本章实验需要的配件：

- Powersensor 传感器

进行传统的图像处理前经常需要对图像进行一些预处理，预处理的目的是增强图像特征、减小计算量等，图像变换是一种常见的图像预处理手段。常用的图像变换有比如像缩放/裁剪/平移之类的平动变换，以及像旋转/扭曲/镜像等的旋转变换。

图像的这两种变换都可以借助仿射变换（Affine Transformation 或 Affine Map）来实现，仿射变换是一种二维坐标 (x, y) 到二维坐标 (u, v) 的线性变换，其数学表达式形式如下：

$$\begin{cases} u = a_1x + b_1y + c_1 \\ v = a_2x + b_2y + c_2 \end{cases}$$

公式中的参数与 OpenCV 使用的仿射变换的矩阵对应的是

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix}$$

注 这里的 (x, y) 是原图像的坐标， (u, v) 是新图像的坐标。仿射变换的实际操作是把原图像上的每个位置的像素点映射到新图像的特定位置上。

6.1 缩放/裁剪/平移

6.1.1 原理讲解

1. 缩放对应的仿射变换

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix}$$

若 $s_x > 1$ 表示沿水平方向放大， $0 < s_x < 1$ 表示沿水平方向缩小；

若 $s_y > 1$ 表示沿垂直方向放大， $0 < s_y < 1$ 表示沿垂直方向缩小；

若 $s_y == s_x$ 表示等比例缩放。

2. 裁剪

裁剪一般不用仿射变换，直接通过数组选择相应的像素即可，如：

$$I_{new} = I_{old}[s_y : e_y, s_x : e_x, :]$$

其中 s_x, e_x 是水平方向起止像素位置, s_y, e_y 是水平方向起止像素位置。

3. 平移对应的仿射变换

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & m_x \\ 0 & 1 & m_y \end{bmatrix}$$

其中 m_x, m_y 表示的沿水平和垂直方向平移的距离。

6.1.2 函数介绍

1. 缩放

由于缩放是非常常用的功能, OpenCV 提供了 cv2.resize() 函数, 可以更方便地操作缩放, 原型如下:

```

1 cv2.resize(img, dsize[, fx[, fy[, interpolation]]])) -> retval
2 # -img, 必须, 原图像;
3 # -dsize, 必须, 目标尺寸, 如果给0则按照fx,fy参数操作;
4 # -fx,fy, 可选, 按比例缩放, 当dsize为0时其作用
5 # -interpolation, 可选, 插值方式, 可以是INTER_NEAREST, INTER_LINEAR(默认),
6 #           INTER_AREA, INTER_CUBIC,
7 # -retval, 返回值是修改后的图像

```

不同插值的区别详情见[不同插值对比](#);

2. 裁剪

裁剪使用数组选择即可实现

3. 平移

平移需要借助 OpenCV 提供的仿射变换函数来实现:

```

1 cv2.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue
    ]]]]) -> dst
2 # -src, 必须, 原图像;
3 # -M, 必须, 操作矩阵即前面原理讲解的矩阵;
4 # -dsize, 可选, 目标尺寸, 仿射变换函数同时提供缩放功能;
5 # -dst, 可选, 输出的图像
6 # -flags, 可选, combination of interpolation methods (see resize())
#           and the optional flag WARP_INVERSE_MAP that means that M is the
#           inverse transformation ( \texttt{dst} \rightarrow \texttt{src} )
7 # -borderMode, 可选, 边缘像素拓展的方式
8 # -borderValue, 可选, 边缘像素拓展的值, 默认为0
9

```

6.1.3 参考例程

缩放/裁剪/平移的示例代码如下：

```

1 img = None
2 for i in range(100):
3     start = time.time()          # 记录开始时间
4     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为320x240尺寸
9 origin = cv2.resize(imgMat, (320, 240))
10 print("原图像：")
11 ps.CommonFunction.show_img_jupyter(origin)
12
13 # 按比例缩放
14 img_scale = cv2.resize(origin, (0, 0), fx=0.5, fy=0.5, interpolation=cv2.
    INTER_NEAREST)
15 print("比例缩放：")
16 ps.CommonFunction.show_img_jupyter(img_scale)
17
18 # 裁剪，第一个变量是行，第二个变量是列
19 img_crop = origin[20:140, 20:200]
20 print("裁剪：")
21 ps.CommonFunction.show_img_jupyter(img_crop)
22
23 # 平移
24 # 平移可以通过仿射变换来实现
25 matrix = np.float32([[1, 0, 100], [0, 1, 100]])
26 rows, cols, _ = origin.shape
27 print("平移：")
28 img_mv = cv2.warpAffine(origin, matrix, (cols, rows))
29 ps.CommonFunction.show_img_jupyter(img_mv)
30
31 end = time.time()
32 print(end - start)
33 time.sleep(0.1)

```

这份代码先读取传感器的图像，经过处理后输出，由于这个例程输出的多个图片的尺寸不一致，实验时可能会有闪烁感。

6.1.4 结果展示

结果如图所示：



图 6.1: 缩放/裁剪/平移实验结果

6.2 旋转/扭曲/镜像

6.2.1 原理讲解

1. 旋转对应的仿射变换

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \end{bmatrix}$$

其中 α 是旋转的角度。

2. 扭曲

这里的扭曲指的是 shear 变换

x 方向的 shear 变换:

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} = \begin{bmatrix} 1 & \tan \phi & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

其中 ϕ 是错切的角度。

y 方向的 shear 变换:

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \tan \phi & 1 & 0 \end{bmatrix}$$

其中 ϕ 是错切的角度。

3. 镜像

镜像对应的仿射变换

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \end{bmatrix}$$

若 $(f_x == -1) \&\& (f_y == 1)$, 则图像对 Y 轴镜像;

若 $(f_x == 1) \&\& (f_y == -1)$, 则图像对 X 轴镜像;

若 $(f_x == -1) \&\& (f_y == -1)$, 则图像对对角线镜像;

6.2.2 函数介绍

1. 旋转

旋转需要通过仿射变换来实现，仿射变换的说明见上一节。另外，OpenCV 提供一个通过角度求仿射变换矩阵的函数：

```

1 cv2.getRotationMatrix2D(center, angle, scale) → retval
2 # -center, 必须, 旋转的中心点;
3 # -angle, 必须, 旋转的角度;
4 # -scale, 必须, 缩放倍数, 这个函数同时提供缩放功能;
5 # -retval, 返回值是修改后的图像
6

```

2. 扭曲

通过构造仿射变换矩阵和仿射变换函数实现。

3. 镜像（翻转）

OpenCV 提供了专门的镜像函数：

```

1 cv2.flip(src, flipCode[, dst]) → dst
2 # -src, 必须, 原图像;
3 # -flipCode, 可选, 翻转的轴 0->x 轴, 1->y 轴, -1->对角
4 # -dst, 可选, 输出的图像
5

```

6.2.3 参考例程

缩放/裁剪/平移的示例代码如下：

```

1 img = None
2 for i in range(100):
3     start = time.time()           # 记录开始时间
4     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
5
6     imgMat = cam1.read_img_ori()    # 读入图像
7
8     # 缩小图像为 320x240 尺寸
9     origin = cv2.resize(imgMat, (320, 240))
10    rows, cols, _ = origin.shape
11
12
13    # 第一个参数是旋转中心，第二个参数是旋转角度，第三个参数是缩放比例
14    matrix = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 1)
15    img_rot = cv2.warpAffine(origin, matrix, (cols, rows))
16
17    # 翻转，第二个参数是方向，1是水平，0是垂直，-1是都翻
18    img_flip = cv2.flip(origin, 1)
19

```

```

20 # 扭曲, x轴的剪切shear变换, 角度45°
21 theta = 45 * np.pi / 180
22 M_shear = np.array([
23 [1, np.tan(theta), 0],
24 [0, 1, 0]
25 ], dtype=np.float32)
26 img_sheared = cv2.warpAffine(origin, M_shear, (cols, rows))
27
28 # 把图像拼接在一起显示
29 img_combine = np.hstack([img_rot, img_flip, img_sheared])
30
31 ps.CommonFunction.show_img_jupyter(img_combine)
32
33 end = time.time()
34 print(end - start)
35 time.sleep(0.1)

```

未来减小输出图像时的闪烁感，这个代码在最后把经过各种处理后的图片拼接在一起显示。

6.2.4 结果展示

结果如图所示：

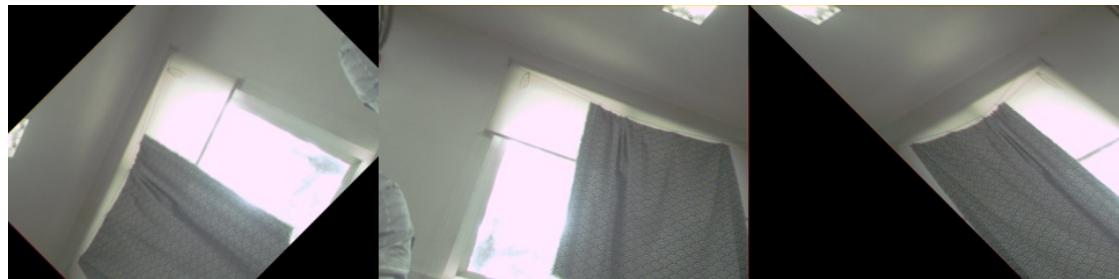


图 6.2：旋转/镜像/扭曲实验结果

从左到右分别是旋转-镜像-扭曲的实验结果。

6.3 色彩空间变换

6.3.1 原理讲解

常见的色彩空间有 RGB, HSV, Gray 等，RGB 比较贴近传感器采集的原始数据，而 HSV 比较贴近视觉感受的颜色，gray 是灰度。RGB 是红绿蓝，HSV 由色相、饱和度、明度组成。OpenCV 提供丰富的函数来进行各种通道的颜色转换，本摄像头采集的原始图像数据是 RGB 的。

在图像处理的时候，有时候在一个色彩空间不明显的特征可能在另一个色彩空间效果非常好，因此掌握色彩空间变换是图像处理的必备技能之一。

1. RGB

RGB 是从颜色发光的原理来设计定的, 通俗点说它的颜色混合方式就好像有红、绿、蓝三盏灯, 当它们的光相互叠合的时候, 色彩相混, 而亮度却等于两者亮度之总和, 越混合亮度越高, 即加法混合。

红、绿、蓝三个颜色通道每种色各分为 256 阶亮度, 在 0 时“灯”最弱——是关掉的, 而在 255 时“灯”最亮。当三色灰度数值相同时, 产生不同灰度值的灰色调, 即三色灰度都为 0 时, 是最暗的黑色调; 三色灰度都为 255 时, 是最亮的白色调。

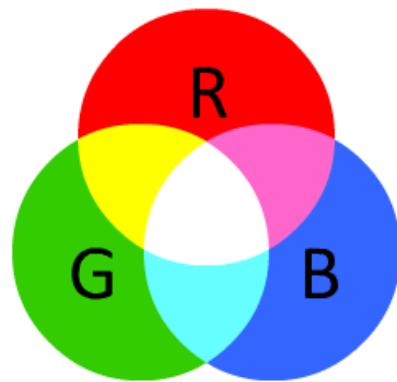


图 6.3: RGB 颜色混合

2. HSV

HSV 又称 HSB, 表示一种颜色模式: 在 HSB 模式中, H(hues) 表示色相, S(saturation) 表示饱和度, B (brightness) 表示亮度 HSB 模式对应的媒介是人眼。

色相 (H,hue): 在 0-360° 的标准色轮上, 色相是按位置度量的。在通常的使用中, 色相是由颜色名称标识的, 比如红、绿或橙色。黑色和白色无色相。饱和度 (S,saturation): 表示色彩的纯度, 为 0 时为灰色。白、黑和其他灰色色彩都没有饱和度的。在最大饱和度时, 每一色相具有最纯的色光。取值范围 0~100%。亮度 (B,brightness 或 V,value): 是色彩的明亮度。为 0 时即为黑色。最大亮度是色彩最鲜明的状态。取值范围 0~100

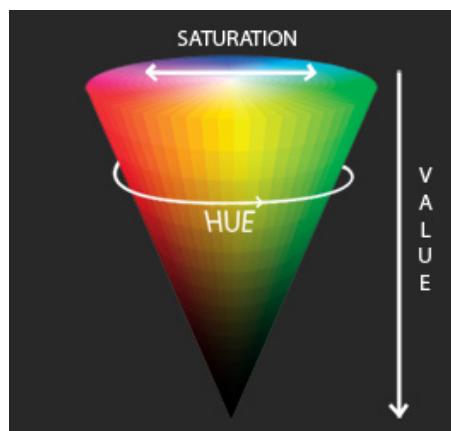


图 6.4: HSV 颜色混合

6.3.2 函数介绍

OpenCV 提供 cv2.cvtColor() 函数来实现图像色彩空间的转换:

```
1 cv2.cvtColor(src, code[, dst[, dstCn]]) -> dst
2 # -src, 必须, 原图像;
3 # -code, 可选, 转换的标志, 可以是CV_BGR2GRAY、CV_BGR2HSV等中的一个;
4 # -dst, 可选, 输出的图像;
5 # -dstCn, 可选, 输出图像的通道数, 默认与原图相同;
```

 **笔记** OpenCV 支持大量色彩空间的转换, 不过 powersensor 传进来的图像是 bgr 格式的。

关于其他色彩空间如 GRAY、YCrCb、HLS 等等请自行查阅相关资料。

6.3.3 参考例程

色彩空间变换的示例代码如下:

```
1 img = None
2 for i in range(100):
3     start = time.time()           # 记录开始时间
4     clear_output(wait=True)       # 清除图片, 在同一位置显示, 不使用会打印多张图片
5
6     imgMat = cam1.read_img_ori()    # 读入图像
7
8     # 缩小图像为 160x120 尺寸,
9     origin = cv2.resize(imgMat, (160,120))
10    rows,cols,_ = origin.shape
11
12    img_r = origin[:, :, 0]
13    img_g = origin[:, :, 1]
14    img_b = origin[:, :, 2]
15
16    # bgr2hsv
17    img_hsv=cv2.cvtColor(origin,cv2.COLOR_BGR2HSV) #HSV 空间
18    img_h = img_hsv[:, :, 0]
19    img_s = img_hsv[:, :, 1]
20    img_v = img_hsv[:, :, 2]
21
22    img_gray=cv2.cvtColor(origin,cv2.COLOR_BGR2GRAY) #HSV 空间
23
24    # 把图像拼接在一起显示
25    # print("第一行: 灰度, r,g,b; 第二行: 灰度,h,s,v")
26    img_line1 = np.hstack([img_gray, img_r, img_g, img_b])
27    img_line2 = np.hstack([img_gray, img_h, img_s, img_v])
28    img_combine = np.vstack([img_line1, img_line2])
29
30    ps.CommonFunction.show_img_jupyter(img_combine)
31
```

```
32 end = time.time()  
33 print(end - start)  
34 time.sleep(0.1)
```

用来减小输出图像时的闪烁感，这个代码在最后把经过各种处理后的图片拼接在一起显示。

6.3.4 结果展示

结果如图所示：

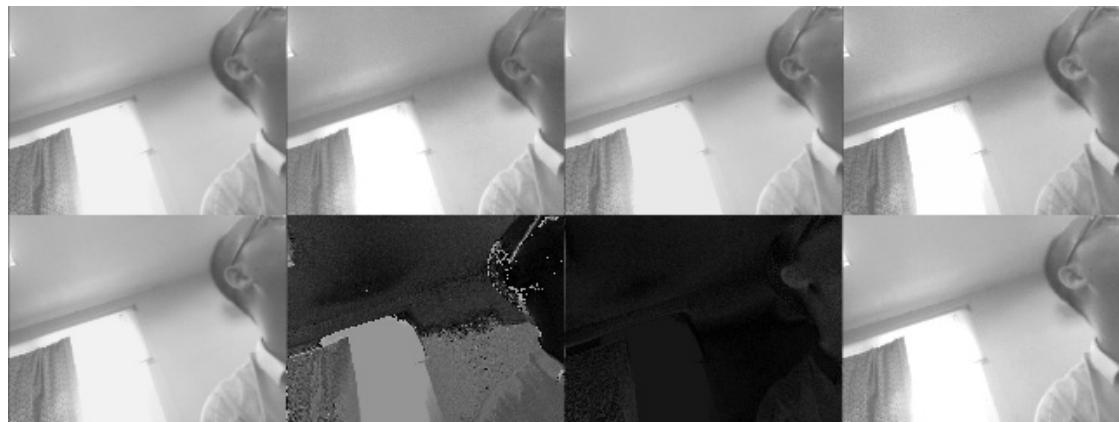


图 6.5：色彩空间变换实验结果

第一行从左到右分别是灰度-R 通道-G 通道-B 通道，第二行从左到右分别是灰度-H 通道-S 通道-V 通道。

6.4 小结一下

结论 本章教程主要介绍一些常见的图像变换，重点是掌握仿射变换和理解不同的色彩空间。

第七章 Tutorial 3 - 图像预处理（二）平滑滤波

内容提要

- 均值滤波
- 中值滤波
- 高斯滤波
- 双边滤波

本章实验需要的配件：

- Powersensor 传感器
- 椒盐噪声图片（F-35 那张）



图 7.1: 椒盐噪声测试图

图像的平滑处理主要用于减小图像上的噪点或者失真，也经常被称为模糊处理。常见的平滑滤波函数分线性和非线性两类，线性的有方框滤波、均值滤波、高斯滤波等，非线性滤波有中值滤波、双边滤波等。

从理论上讲，方框滤波、均值滤波、高斯滤波对高斯白噪声滤除效果较好，但是会带来模糊，中值滤波对椒盐噪声滤除效果较好。实际情况由于真正的白噪声很难存在，所以均值滤波的效果大打折扣。

7.1 均值滤波

7.1.1 原理讲解

这是最简单的均值滤波器，可以去除均匀噪声和高斯噪声，但会对图像造成一定程度的模糊，它的公式如下：

$$f(u, v) = \frac{1}{mn} \sum_{(u,v) \in S_{uv}} g(x, y)$$

其中 S_{uv} 是中心在 (u, v) 处，大小为 (m, n) 的滤波器窗口， $g(x, y)$ 是原始图像， $f(u, v)$ 是滤波后得到的图像。均值滤波就是让当前点（一般是中心点）的像素值为当前窗口区域的均值。

一个 3×3 的均值滤波对应的矩阵如下图所示：

$$f(u, v) = g(x, y) * \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

方框滤波与均值滤波非常相似，就是它的归一化参数是可调的，不一定是 mn 。

7.1.2 函数介绍

OpenCV 提供了方便快捷的平均滤波的函数 cv2.blur():

```

1 cv2.blur(src, ksize[, dst[, anchor[, borderType]]]) → dst
2 # -src, 必须, 原图像;
3 # -ksize, 可选, 卷积核的大小, 如 (5, 5);
4 # -dst, 可选, 图像输出;
5 # -anchor, 可选, 锚点, 默认 (-1, -1) 是核中心;
6 # -borderType, 可选, 边缘的补充模式, 默认是 BORDER_DEFAULT;
```

7.1.3 参考例程

均值滤波的示例代码如下：

```

1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为 320x240 尺寸
9 origin = cv2.resize(imgMat, (320, 240))
10 rows, cols, _ = origin.shape
11
12 start = time.time()          # 记录开始时间
13
14 # 第一个参数是原图像, 第二个参数是窗口的大小
15 img_mean1 = cv2.blur(origin, (5, 5))
16 img_mean2 = cv2.blur(origin, (10, 10))
17
18 end = time.time()          # 记录开始时间
19
20 #     # 把图像拼接在一起显示
21 img_combine = np.hstack([origin, img_mean1, img_mean2])
22
23 ps.CommonFunction.show_img_jupyter(img_combine)
24
25
26 print(end - start)
```

```
27 time.sleep(0.1)
```

这个代码测试了两个参数不同均值滤波器。

7.1.4 结果展示

结果如图所示：



图 7.2：均值滤波实验结果

从左到右分别是原图、 5×5 的均值滤波、 10×10 的均值滤波。可以看出，滤波器窗口越大，对噪声的滤除效果越好，但是图像也越模糊。

7.2 高斯滤波

7.2.1 原理讲解

高斯滤波将卷积核换成一组符合高斯分布的数值放在模板里，这时模板中的数值将会中间的数值最大，往两边走越来越小，构造一个小的高斯包，如下图所示。这样可以减少原始图像信息的丢失。

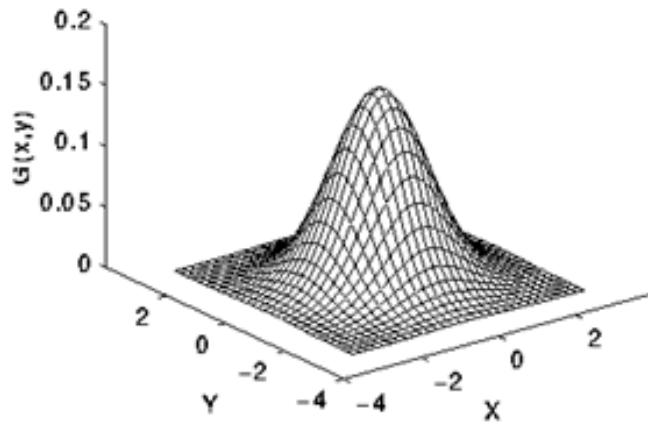


图 7.3：高斯滤波权重变换

高斯滤波的作用原理与均值滤波相似，不同的地方在于它对与滤波窗口内不同位置点所乘的权重是不同的，距离中心近的点权重大，远的点权重小。所以高斯滤波器在保证滤波效果的同时减小了图像的模糊程度。权重的公式如下：

$$h(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

其中 (x, y) 为点的坐标, σ 为待设定的标准差。一个 3×3 的高斯核如下所示:

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

7.2.2 函数介绍

OpenCV 提供的高斯滤波的函数为: cv2.GaussianBlur()。使用这个函数时我们需要指定的是高斯核的大小 (奇数), 以及沿的标准差的大小 (如果给 0, 函数会自己计算)。

```
1 cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) -> dst
2 # -src, 必须, 原图像;
3 # -ksize, 必须, 卷积核的大小, 如 (5, 5);
4 # -sigmaX, 可选, x 方向的标准差;
5 # -dst, 可选, 输出的图像;
6 # -sigmaY, 可选, y 方向的标准差;
7 # -borderType, 可选, 边缘的补充模式, 默认是 BORDER_DEFAULT;
```

7.2.3 参考例程

高斯滤波的示例代码如下:

```
1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为 320x240 尺寸
9 origin = cv2.resize(imgMat, (320, 240))
10 rows, cols, _ = origin.shape
11
12 start = time.time()          # 记录开始时间
13
14 # 第一个参数是原图像, 第二个参数是窗口的大小
15 img_gauss1 = cv2.GaussianBlur(origin, (5, 5), 0)
16 img_gauss2 = cv2.GaussianBlur(origin, (9, 9), 0)
17
18 end = time.time()          # 记录开始时间
19
20 # 把图像拼接在一起显示
21 img_combine = np.hstack([origin, img_gauss1, img_gauss2])
22
23 ps.CommonFunction.show_img_jupyter(img_combine)
```

```

24
25 print(end - start)
26 time.sleep(0.1)

```

这个代码测试了两个参数不同高斯滤波器。

7.2.4 结果展示

结果如图所示：



图 7.4：高斯滤波实验结果

从左到右分别是原图、 5×5 的高斯滤波、 9×9 的高斯滤波。可以看出，相比于均值滤波，高斯滤波考虑了全局信息，在达到与均值滤波差不多的去噪效果时依然有较好的边缘特性，但是高斯滤波要消耗更多的计算资源。

7.3 中值滤波

7.3.1 原理讲解

中值滤波是一种非线性滤波方式，是基于排序统计理论的一种能有效抑制噪声的非线性信号处理技术，它的基本原理是把数字图像或数字序列中一点的值用该点的一个邻域中各点值的中值代替，让周围的像素值接近的值，从而消除孤立的噪声点。中值滤波一般使用模板的方法实现，对模板内的像素按照像素值的大小进行排序，生成单调上升（或下降）的二维数据序列，并使用下面的公式进行输出：

$$g(u, v) = \text{med}\{f(x - m, y - n), (m, nW)\}$$

其中 $g(x, y)$ 是原图像， $f(u, v)$ 是新图像。

中值滤波模板就是用卷积框中像素的中值代替中心值，达到去噪声的目的。这个模板一般用于去除椒盐噪声。前面的滤波器都是用计算得到的一个新值来取代中心像素的值，而中值滤波是用中心像素周围（也可以使他本身）的值来取代他，卷积核的大小也是个奇数。

7.3.2 函数介绍

OpenCV 提供的中值滤波的函数为：cv2.medianBlur()。

```

1 cv2.medianBlur(src, ksize[, dst]) → dst
2 # -src, 必须, 原图像;
3 # -ksize, 可选, 卷积核的大小, 必须是奇数且大于1, 如(5, 5);
4 # -dst, 可选, 输出的图像;

```

7.3.3 参考例程

中值滤波的示例代码如下：

```

1 img = None
2 for i in range(100):
3
4     clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印多张图片
5
6     imgMat = cam1.read_img_ori()      # 读入图像
7
8     # 缩小图像为320x240尺寸
9     origin = cv2.resize(imgMat, (320, 240))
10    rows, cols, _ = origin.shape
11
12    start = time.time()          # 记录开始时间
13
14    # 第一个参数是原图像, 第二个参数是窗口的大小
15    img_mid1 = cv2.medianBlur(origin, 5)
16    img_mid2 = cv2.medianBlur(origin, 9)
17
18    end = time.time()          # 记录开始时间
19
20    # 把图像拼接在一起显示
21    img_combine = np.hstack([origin, img_mid1, img_mid2])
22
23    ps.CommonFunction.show_img_jupyter(img_combine)
24
25    print(end - start)
26    time.sleep(0.1)

```

这个代码测试了两个参数不同中值滤波器。

7.3.4 结果展示

结果如图所示：



图 7.5: 中值滤波实验结果

从左到右分别是原图， 5×5 的中值滤波， 9×9 的中值滤波。可以看出，相比于线性滤波，中值滤波对于椒盐噪声的照片滤除效果非常好，但是中值滤波需要消耗比高斯滤波更多的计算资源。

7.4 双边滤波

7.4.1 原理讲解

双边滤波（Bilateral filter）是一种非线性的滤波方法，是结合图像的空间邻近度和像素值相似度的一种折中处理，同时考虑空域信息和灰度相似性，达到保边去噪的目的。具有简单、非迭代、局部的特点。

该算法结合空间信息和亮度相似性对图像进行滤波处理，在平滑滤波的同时能大量保留图像的边缘和细节特征。公式如下：

$$f(u, v) = \frac{\sum_{(x,y) \in S} g(x, y) \omega(x, y)}{\sum_{(x,y) \in S} \omega(x, y)}$$

其中 $g(x, y)$ 是原图像， $f(u, v)$ 是新图像， S 是滤波器窗口， $\omega(x, y)$ 是滤波器权值矩阵，定义如下：

$$\begin{aligned}\omega(x, y) &= \varphi(x, y) \phi(x, y) \\ \varphi(x, y) &= e^{-\frac{\|y-x\|^2}{2\sigma_s^2}} \\ \phi(x, y) &= e^{-\frac{(g(x)-g(y))^2}{2\sigma_c^2}}\end{aligned}$$

其中 $\varphi(x, y)$ 为空域滤波权值，采用高斯核的形式，距离中心越近，权值越大； $\phi(x, y)$ 为值域滤波权值，采用高斯核的形式，与中心的像素越相似，权值越大； σ_s 和 σ_c 分别是这两个权值的标准差。

因此，在灰度变化平缓区域，值域滤波系数接近 1，此时空域滤波起主要作用，双边滤波器退化为传统的高斯低通滤波器，对图像进行平滑操作。而在图像变化剧烈的部分（即图像边缘），像素间差异较大，值域滤波起主要作用，因而能保持边缘信息。示意图如下：

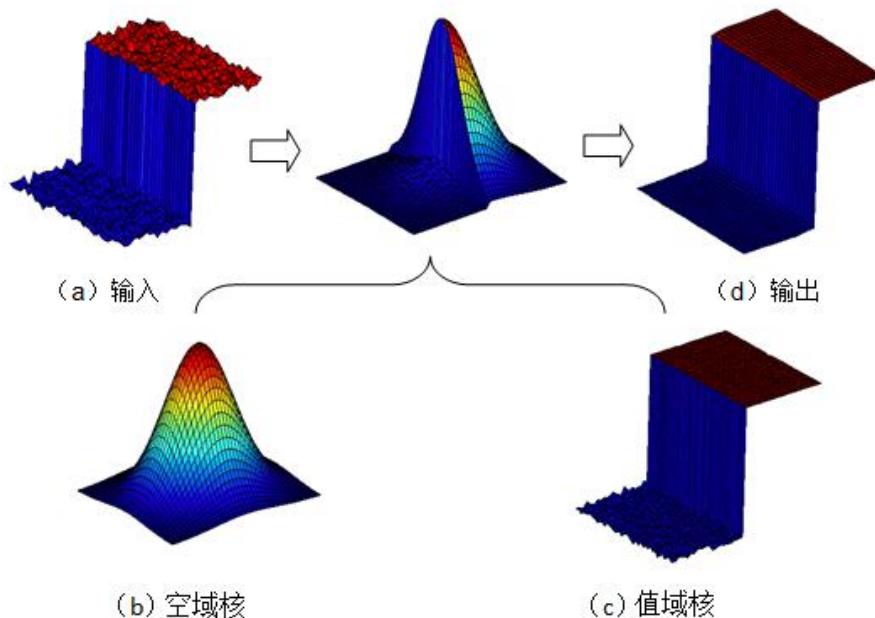


图 7.6: 双边滤波示意图

7.4.2 函数介绍

OpenCV 提供的中值滤波的函数为: cv2.medianBlur()。

```

1 cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]]) ->
2     dst
3 # -src, 必须, 原图像;
4 # -d, 必须, 滤波器直径, 如果给的参数非正, 函数会自动计算;
5 # -sigmaColor, 必须, 值域滤波器的标准差, 越大滤波效果越强;
6 # -sigmaSpace, 可选, 空域滤波的标准差, 越大滤波效果越强;
7 # -dst, 可选, 输出的图像;

```

注 标准差: 简单的说, 可以两个 σ 设置成一样。当 $\sigma < 10$, 滤波器效果非常微弱; 当 $\sigma > 150$, 滤波器效果非常强, 整个画面看起来会比较卡通。

7.4.3 参考例程

双边滤波的示例代码如下:

```

1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为 320x240 尺寸
9 origin = cv2.resize(imgMat, (320, 240))
10 rows, cols, _ = origin.shape

```

```

11
12 start = time.time()          # 记录开始时间
13
14 # 第一个参数是原图像，第二个参数是窗口的大小
15 img_dual1 = cv2.bilateralFilter(origin, 7, 120, 120)
16 img_dual2 = cv2.bilateralFilter(origin, 9, 120, 120)
17
18 end = time.time()           # 记录开始时间
19
20 # 把图像拼接在一起显示
21 img_combine = np.hstack([origin, img_dual1, img_dual2])
22
23 ps.CommonFunction.show_img_jupyter(img_combine)
24
25 print(end - start)
26 time.sleep(0.1)

```

这个代码测试了两个直径不同的双边滤波器，双边滤波器的计算量受直径影响非常大，9以上的都会比较慢。

7.4.4 结果展示

结果如图所示：



图 7.7：双边滤波实验结果

从左到右分别是原图，直径为 7 的双边滤波，直径为 9 的双边滤波。可以看出，双边滤波的效果还是不错的，就是要消耗大量的计算资源。

7.5 小结一下

结论 本章教程主要介绍一些常见的图像平滑滤波的算法，不同的算法有不同的适应环境。opencv 提供了大量的滤波器函数，实际应用时使用这些函数会比自己编写效率更高、效果更好，开发周期也短。但是，理解滤波器的原理对于滤波器参数的设置有重要意义，因此还是要掌握每种滤波器的基本原理。

第八章 Tutorial 4 - 图像预处理（三）- 形态学处理

内容提要

- 腐蚀膨胀
- 高帽黑帽
- 开闭操作

本章实验需要的配件：

- Powersensor 传感器
- 形态学测试图片（有字母 J 那一张）



图 8.1: 形态学测试图

图像的平滑处理是在灰度图的层面对图像进行去初步的去噪，这个过程之后有时需要对图像进行二值化操作，二值化后的图像更容易让‘程序’发现隐含在图像中的特征。二值化的图像也经常需要进行预处理，去除一些噪点，建立更好的连通区，这个称为形态学预处理。

形态学操作的主要功能有：

- 消除噪声
- 分割 (isolate) 独立的图像元素，以及连接 (join) 相邻的元素。
- 寻找图像中的明显的极大值区域或极小值区域。

8.1 腐蚀膨胀

8.1.1 原理讲解

膨胀或者腐蚀操作就是将图像（或图像的一部分区域，我们称之为 A）与核（我们称之为 B）进行卷积。核可以是任何的形状和大小，它拥有一个单独定义出来的参考点，我们称其为锚点（anchorpoint）。多数情况下，核是一个小的中间带有参考点和实心正方形或者圆盘，其实，我们可以把核视为模板或者掩码。

膨胀就是求局部最大值的操作，核 B 与图形卷积，即计算核 B 覆盖的区域（体现局部）的像素点的最大值，并把这个最大值赋值给参考点指定的像素。这样就会使图像中的高亮区域逐渐增长，一个 4 联通的膨胀示意图如下：



图 8.2: 腐蚀示意图

腐蚀是与膨胀相反的操作，它是把局部的最小值赋值给指定的像素，指定像素的位置是由模板决定的。一个四联通的腐蚀示意图如下：



图 8.3: 膨胀示意图



笔记 一般来说，腐蚀膨胀这个操作是针对白色的区域而言，就是对灰度值为 255 的点而言的。因此，对于黑色的区域，效果刚好是相反的。

8.1.2 函数介绍

1. 腐蚀

```

1 cv2.erode(img,kernel,iterations) → dst
2 # -src, 必须, 原图像;
3 # -kernel, 必须, 锚点矩阵, 可以用numpy矩阵来构建;
4 # -iterations, 必须, 迭代次数, 就是滤波多少遍;
5 # -dst, 可选, 图像输出;
6

```

2. 膨胀

```

1 cv2.dilate(img,kernel,iterations) → dst
2 # -src, 必须, 原图像;
3 # -kernel, 必须, 锚点矩阵, 可以用numpy矩阵来构建;
4 # -iterations, 必须, 迭代次数, 就是滤波多少遍;
5 # -dst, 可选, 图像输出;
6

```

8.1.3 参考例程

腐蚀膨胀的示例代码如下，这份代码先使用二值化函数对采集到的图像进行二值化，然后再进行腐蚀膨胀操作。由于图片上的 J 是黑色的，与腐蚀膨胀的对象是白色区域，因此二值化后需要进行取反操作。

```

1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为320x240尺寸
9 origin = cv2.resize(imgMat, (320,240))
10 rows,cols,_ = origin.shape
11
12 start = time.time()          # 记录开始时间
13
14
15 # 把图片转换成灰度图
16 img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
17 # 二值化灰度后的图片
18 _, img_binary = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY)
19 # 因为图片上的字是黑色的，而腐蚀、膨胀是针对白色区域的，因此需要进行翻转
20 img_binary = 255 - img_binary
21
22 # 腐蚀
23 kernel = np.ones((3, 3), np.uint8)
24 img_eroде = cv2.erode(img_binary, kernel, iterations=6)
25
26 # 膨胀
27 kernel = np.ones((3, 3), np.uint8)
28 img_dilate = cv2.dilate(img_binary, kernel, iterations=6)
29
30
31 end = time.time()          # 记录开始时间
32
33 #     # 把图像拼接在一起显示
34 img_combine = np.hstack([img_gray, img_eroде, img_dilate])
35
36 ps.CommonFunction.show_img_jupyter(img_combine)
37
38
39 print(end - start)
40 time.sleep(0.1)

```

进行实验时，一般是固定摄像头，然后调整照片的角度和位置，腐蚀膨胀用的是第一行的图片。

8.1.4 结果展示

结果如图所示：



图 8.4: 腐蚀膨胀实验结果

从左到右分别是原图，腐蚀后图片，膨胀后的图片。可以看出，腐蚀可以去掉轨迹上的毛线，膨胀可以把断开的几节接在一起。

8.2 开/闭运算

腐蚀和膨胀运算虽然可以去除噪点以及连接散块，但是会影响原图像的性状，把它们两个结合在一起可以解决这个问题，这就是开/闭运算。

8.2.1 原理讲解

开闭运算的定义如下：

- **开运算：**先腐蚀后膨胀，用于移除由图像噪音形成的斑点，就是能够排除小团块物体(假设物体较背景明亮)，示意图如下：

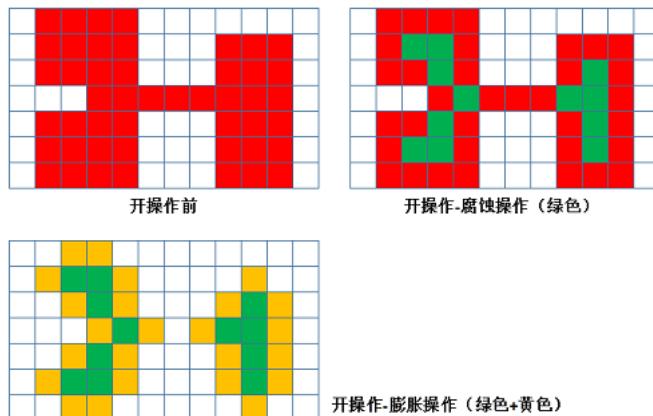


图 8.5: 开运算示意图

- **闭运算：**先膨胀后腐蚀，用来连接被误分为许多小块的对象。

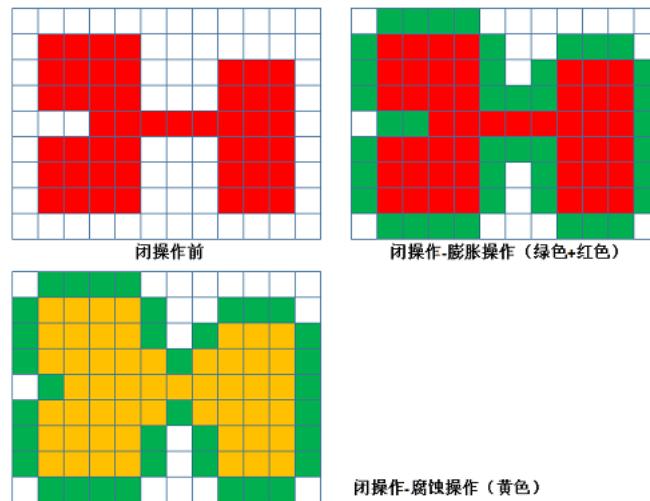


图 8.6: 闭运算示意图

8.2.2 函数介绍

OpenCV 使用 morphologyEx() 来完成开闭运算。

```

1 cv2.morphologyEx(img, flag, kernel) → dst
2 # -src, 必须, 原图像;
3 # -flag, 必须, MORPH_OPEN 或 MORPH_CLOSE 或 MORPH_TOPHAT 或 MORPH_BLACKHAT 中的一个, 分别代表开运算、闭运算、高帽运算、黑帽运算;
4 # -kernel, 必须, 锚点矩阵, 可以用 numpy 矩阵来构建;
5 # -dst, 可选, 图像输出;

```

8.2.3 参考例程

开闭运算的示例代码如下:

```

1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为 320x240 尺寸
9 origin = cv2.resize(imgMat, (320, 240))
10 rows, cols, _ = origin.shape
11
12 start = time.time()          # 记录开始时间
13
14
15 # 把图片转换成灰度图
16 img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
17 # 二值化灰度后的图片
18 _, img_binary = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY)

```

```

19 # 因为图片上的字是黑色的，而腐蚀、膨胀是针对白色区域的，因此需要进行翻转
20 img_binary = img_binary
21
22 # 开运算
23 kernel = np.ones((3, 3), np.uint8)
24 img_open = cv2.morphologyEx(img_binary, cv2.MORPH_OPEN, kernel)
25 #     img_erode = cv2.erode(img_binary, kernel, iterations=6)
26
27 # 闭运算
28 kernel = np.ones((5, 5), np.uint8)
29 img_close = cv2.morphologyEx(img_binary, cv2.MORPH_CLOSE, kernel)
30 #     img_dilate = cv2.dilate(img_binary, kernel, iterations=6)
31
32
33 end = time.time()           # 记录开始时间
34
35 #     # 把图像拼接在一起显示
36 img_combine = np.hstack([img_gray, img_open, img_close])
37
38 ps.CommonFunction.show_img_jupyter(img_combine)
39
40
41 print(end - start)
42 time.sleep(0.1)

```

进行实验时，一般是固定摄像头，然后调整照片的角度和位置，通过移动视角查看不同图片经过开闭运算后的效果。

8.2.4 结果展示

结果如图所示：



图 8.7：开闭运算实验结果

从左到右分别是原图，开运算后图片，闭运算后的图片。可以看出，开闭运算并不会影响原来线条的粗细，又能够滤除噪点。

8.3 高帽黑帽

高帽黑帽是用来查看开闭运算到底滤除了哪些噪点。

8.3.1 原理讲解

高帽黑帽的定义如下：

- 高帽（或礼帽）运算：原始图像减去图像开运算的结果；
- 黑帽运算：图像闭运算减去原始图像的结果。

8.3.2 函数介绍

OpenCV 使用 morphologyEx() 来完成开闭运算，详情见章节8.2.2。

8.3.3 参考例程

高帽黑帽的示例代码如下：

```

1 img = None
2 for i in range(100):
3
4     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
5
6     imgMat = cam1.read_img_ori()      # 读入图像
7
8     # 缩小图像为320x240尺寸
9     origin = cv2.resize(imgMat, (320, 240))
10    rows, cols, _ = origin.shape
11
12    # ----- 图像处理开始
13
14    start = time.time()          # 记录开始时间
15
16    # 把图片转换成灰度图
17    img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
18    # 二值化灰度后的图片
19    _, img_binary = cv2.threshold(img_gray, 128, 255, cv2.THRESH_BINARY)
20    # 因为图片上的字是黑色的，而腐蚀、膨胀是针对白色区域的，因此需要进行翻转
21    img_binary = cv2.bitwise_not(img_binary)
22
23    # 高帽运算
24    kernel = np.ones((3, 3), np.uint8)
25    img_tophat = cv2.morphologyEx(img_binary, cv2.MORPH_TOPHAT, kernel)
26    #     img_eroode = cv2.erode(img_binary, kernel, iterations=6)
27
28    # 黑帽运算
29    kernel = np.ones((5, 5), np.uint8)

```

```

29 img_blackhat = cv2.morphologyEx(img_binary, cv2.MORPH_BLACKHAT, kernel)
30 #     img_dilate = cv2.dilate(img_binary, kernel, iterations=6)
31
32
33 end = time.time()          # 记录结束时间
34 # ----- 图像处理结束
35
36 #     # 把图像拼接在一起显示
37 img_combine = np.hstack([img_gray, img_tophat, img_blackhat])
38
39 ps.CommonFunction.show_img_jupyter(img_combine)
40
41
42 print(end - start)
43 time.sleep(0.1)

```

这个实验的过程与开闭运算实验过程类似。

8.3.4 结果展示

结果如图所示：



图 8.8：高帽黑帽运算实验结果

从左到右分别是原图、高帽运算后图片、黑帽运算后的图片。

8.4 小结一下

结论 本章教程主要介绍几种常见的图像形态学运算，进行形态学运算前一般需要进行二值化，一个好的二值化可以提高算法的鲁棒性。腐蚀运算作用于去除噪点，膨胀运算用于连接断点，开闭运算与上述两者对应，但是不影响原线条的大小，因此实际使用的一般是开闭运算。高帽黑帽运算可以用于查看开闭运算产生了哪些影响，有助于图像参数的调试。

第九章 Tutorial 5 - 图像的特征（一）-直方图

内容提要

直方图

直方图平衡

本章实验需要的配件：

- Powersensor 传感器
- 直方图测试图片（有小女孩那张）



图 9.1: 直方图测试图

9.1 直方图

9.1.1 原理讲解

直方图表征的是图像的统计信息，直方图的横轴是灰度值（0-255），纵轴是这张图片上对应灰度值的数量的统计。直方图在计算机视觉领域通常用来反映图片的亮度，色彩，强度在整个图片里面的分布情况。

所以，直方图反映的是一张图片不同灰度（颜色）含量的大小。就灰度图而言，如果直方图曲线右边高左边低，那么这张图片就偏亮，左边高右边低，那么图片看起来就比较暗。

一张向日葵的照片及其直方图展示如下：

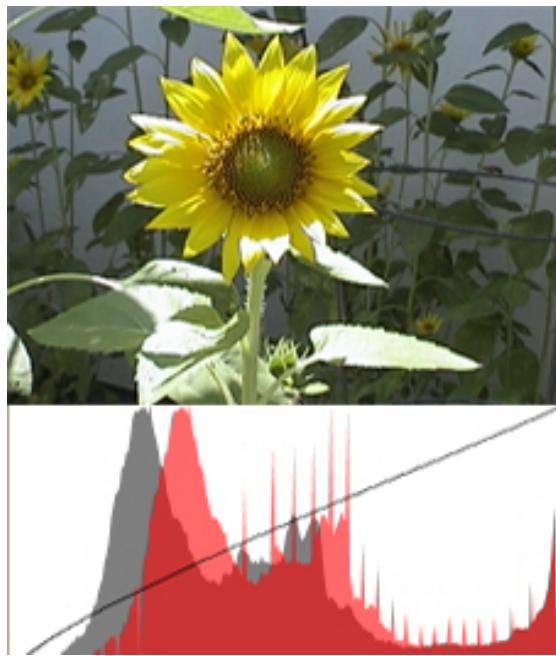


图 9.2: 向日葵和它的直方图

9.1.2 函数介绍

```

1 cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])
   → hist
2 # -images, 必须, 原图;
3 # -channels, 必须, 通道选择, 见注;
4 # -mask, 必须, 掩盖区, 指示要统计哪些区域, 一般给None (要求图片的像素精度是8
   位才可以给None) ;
5 # -histSize, 必须, 每个通道的灰度值要统计成多少级, 简单的说最后会生成多少个
   bins, 习惯给255;
6 # -ranges, 可选, 统计的灰度值的范围, 一般是0-255;
7 # -accumulate, 可选, 是否累计统计, 默认即可;
8 # -hist, 输出, 直方图参数;

```

注 OpenCV 支持多通道的直方图绘制，不过一般为了简便操作，每次绘制一个通道的直方图，然后通过传入的图像来选择颜色通道。

注 OpenCV 直方图相关的函数介绍的官方文档连接：[直方图函数](#)

9.1.3 参考例程

官方给出了计算直方图参数的函数，但是没有给具体绘制直方图的函数，因此，我们需要先封装一个输入为图片，输出为直方图的函数 `get_hist_img()`。

大概的流程是先使用官方的函数得到直方图参数，然后使用 `numpy` 对其归一化，最后使用 OpenCV 的多边形绘制函数绘制直方图（见 tutorial 1），详细代码如下：

```

1 # 根据灰度图像生成直方图的函数
2 def get_hist_img(img_single, color):
3     img_h = np.zeros((240, 320)) # 创建用于绘制直方图的全0图像

```

```

4 originHist = cv2.calcHist([img_single], [0], None, [256], [0,256]) # 计算直方图
    参数
5 cv2.normalize(originHist, originHist, 0, 240*0.9, cv2.NORM_MINMAX) # 对数据进
    行归一化，方便绘图
6 hist = np.int32(np.around(originHist)) # 取整
7 bins = np.arange(256).reshape(256,1) # 直方图中各bin的顶点位置
8 pts = np.column_stack((bins, hist))
9 cv2.polylines(img_h, [pts], False, color)
10 img_h = np.fliplr(img_h)
11 return img_h

```

然后我们就可以调用上面定义好的函数来绘制直方图。

为了清晰地表示明暗图片的直方图的差异，本实验在读入图像后，利用以前介绍的图片裁剪（见 tutorial 2）获取图像的左右两部分，然后分别对这两部分进行直方图统计。

```

1 img = None
2 for i in range(100):
3
4 clear_output(wait=True) # 清除图片，在同一位置显示，不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori() # 读入图像
7
8 # 缩小图像为320x240尺寸
9 origin = cv2.resize(imgMat, (320,240))
10 rows,cols,_ = origin.shape
11
12 # ----- 图像处理开始
13
14 start = time.time() # 记录开始时间
15
16 # 把图片转换成灰度图
17 img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
18
19 # 把图片分成左右两个部分
20 img1 = img_gray[:, 0:160]
21 img2 = img_gray[:, 160:]
22
23 # 调用自定义函数绘制直方图
24 img_h1 = get_hist_img(img1, (255, 255, 255))
25 img_h2 = get_hist_img(img2, (128, 128, 128))
26
27 end = time.time() # 记录结束时间
28 # ----- 图像处理结束
29
30 # 把图像拼接在一起显示
31 img_combine = np.hstack([img_gray, img_h1, img_h2])

```

```

31 ps.CommonFunction.show_img_jupyter(img_combine)
32
33
34 print(end - start)
35 time.sleep(0.1)

```

我们提供的小女孩图片，左边部分是过曝的，右边部分是光照不足的，因此这个实验进行时要求图片的中线刚好处于摄像头的中线位置才可以看到比较好的效果。

9.1.4 结果展示

结果如图所示：

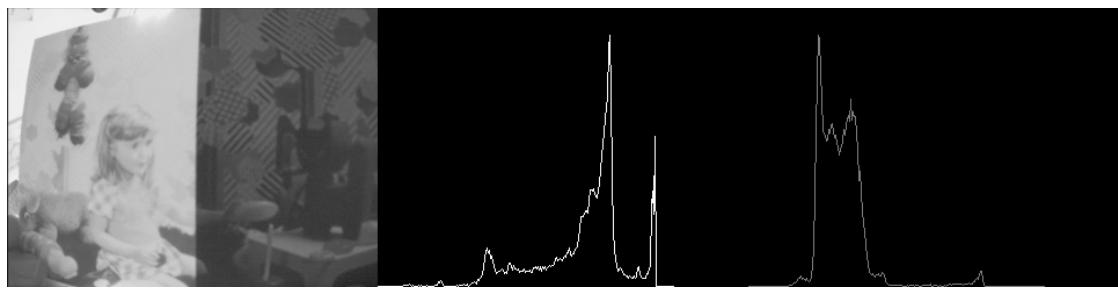


图 9.3：直方图显示实验结果

从左到右分别是原图、左半图的直方图统计、右半图的直方图统计。可以看出做左半图高灰度的分量多，右半图低灰度的分量多。

9.2 直方图均衡

9.2.1 原理讲解

一般来说，如果一个画面里各种灰度的含量是差不多的，那这个画面看起来细节会比较丰富，简单的说就是曝光准确。对于直方图不均衡的图片，比如左边高右边低的情况，可以通过直方图均衡来修补曝光不准确的问题。

通过亮度和增益的控制可以改善图像的显示，那么我们怎么自动选择它们的最佳取值呢？一种方法是寻找图像中最亮和最暗的像素值，将他们映射到纯白和纯黑。另一种方法是寻找图像中像素值的平均值作为中间灰度值，然后扩展范围以达到尽量充满可显示的值。

直方图均衡用一张图片来形象的表示就是：

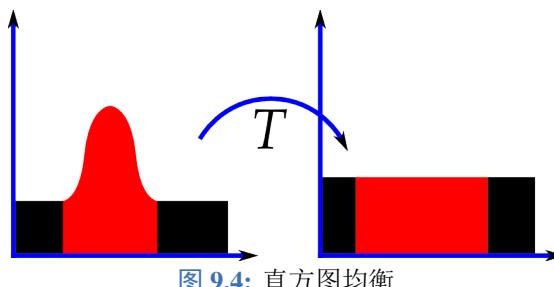


图 9.4：直方图均衡

9.2.2 函数介绍

OpenCV 提供的直方图均衡函数是 cv2.equalizeHist():

```
1 cv2.equalizeHist(src[, dst]) → dst
2 # -src, 必须, 原图;
3 # -dst, 可选, 均衡化后的图片;
```

9.2.3 参考例程

直方图均衡化的参考例程如下：

为了清晰地表示明暗图片的直方图的差异，这个实验在读入图像后，利用以前介绍的图片裁剪（见 tutorial 2）获取图像的左右两部分，然后分别对这两部分进行直方图均衡化。

```
1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为320x240尺寸
9 origin = cv2.resize(imgMat, (320, 240))
10 rows, cols, _ = origin.shape
11
12 # ----- 图像处理开始
13
14 start = time.time()          # 记录开始时间
15
16 # 把图片转换成灰度图
17 img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
18
19 # 把图片分成左右两个部分
20 img1 = img_gray[:, 0:160]
21 img2 = img_gray[:, 160:]
22
23 equ1 = cv2.equalizeHist(img1)
24 equ2 = cv2.equalizeHist(img2)
25
26 end = time.time()          # 记录结束时间
27 # ----- 图像处理结束
28
29 # 把图像拼接在一起显示
30 img_combine = np.hstack([img_gray, equ1, equ2])
ps.CommonFunction.show_img_jupyter(img_combine)
```

```
31  
32  
33 print(end - start)  
34 time.sleep(0.1)
```

看我们给的小女孩图片，左边部分是过曝的，右边部分是光照不足的，因此这个实验进行时要求图片的中线刚好处于摄像头的中线位置才能看到比较好的效果。

9.2.4 结果展示

结果如图所示：



图 9.5：直方图均衡化实验结果

从左到右分别是原图，左半图的直方图统计，右半图的直方图统计。可以看出经过均衡化后的图片的细节明显增多。

9.3 小结一下

结论 直方图是检查照片曝光是否准确的重要标准，但是如果画面中纯色（如纯白色）的面积过大，这个判断可能失效，因此要根据实际情况进行调整。直方图均衡化是弥补曝光不准确的一种手段，但是也会带来噪声，因此需要与一些平滑滤波算法配合使用。

第十章 Tutorial 6 - 图像的特征（二）梯度特征

内容提要

- Sobel 算子
- Roberts 算子
- Laplacian 算子
- Canny 算子

本章实验需要的配件：

- Powersensor 传感器
- 边缘测试图片（有钟表和塔那张）



图 10.1: 边缘测试图

图像处理的一个关键就是处理画面中各个物体的边缘，从数学的角度上说，边缘附近的像素点的灰度值会有明显的变化，而梯度特征真是这些变化的一个很好的描述。由于图像的梯度特征比较复杂，需要考虑噪声等问题，有许多效果较好的梯度算子被人们总结出来，常见的有 Sobel 算子、Laplace 算子、Robert 算子、Canny 算子等。

10.1 Sobel 算子

10.1.1 原理讲解

Sobel 算子是一个小且是整数的滤波器对整张影像在水平及垂直方向上做卷积，因此它所需的运算资源相对较少，另一方面，对于影像中的频率变化较高的地方，它所得的梯度之近似值也比较粗糙。

Sobel 算子认为，邻域的像素对当前像素产生的影响不是等价的，所以距离不同的像素具有不同的权值，对算子结果产生的影响也不同。一般来说，距离越远，产生的影响越小。

Sobel 算子的操作是用一个设计好的卷积核和原图像进行卷积操作，这个卷积操作的实质就是在求上下（左右）的梯度，因此它有 X 方向的算子和 Y 方向的算子两种， 3×3 的卷积核如下所示：

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \text{ and } G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

其中 G_x 是 X 方向的 Sobel 卷积核, G_y 是 Y 方向的 Sobel 卷积核。

10.1.2 函数介绍

```

1 cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType
   ]]]]]) → dst
2 # -src, 必须, 原图;
3 # -ddepth, 必须, 输出图像的像素精度, 可以是CV_8U、CV_16U、CV_16S、CV_32F、
   CV_64F 中的一个;
4 # -dx,dy, 必须, dx是在x方向上求导的阶数, 0为不求导, dy同理, 是y方向上的, 比如
   要在x方向求导就给1,0;
5 # -dst, 可选, 输出图像;
6 # -ksize, 可选, 卷积核大小, 必须是1,3,5,7中的一个;
7 # -scale, 可选, 详情见官方文档, 默认不设置;
8 # -delta, 可选, 可选的偏移值, 会被直接加到结果上;
9 # -borderType, 可选, 边界拓展方式(see borderInterpolate for details);

```



笔记 Powersensor 的摄像头输入的图像是 8 位无符号的, 但是经过梯度算子后会越界, 因此这里要使用 16 位有符号位的数据, 即 CV_16S。

10.1.3 参考例程

Sobel 算子的参考例程如下:

```

1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为320x240尺寸
9 origin = cv2.resize(imgMat, (320,240))
10 rows,cols,_ = origin.shape
11
12 # ----- 图像处理开始
13
14
15 # 把图片转换成灰度图
16 img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
17 img_sobelx = cv2.Sobel(img_gray, cv2.CV_16S, 1, 0, ksize=3) # x 方向, ksize是
   卷积核的大小, 换成16位是因为算子过程可以使原图像的值溢出。

```

```

18 img_sobely = cv2.Sobel(img_gray, cv2.CV_16S, 0, 1, ksize=3) # y 方向
19
20 end = time.time()          # 记录结束时间
21 # ----- 图像处理结束
22
23 #     # 把图像拼接在一起显示
24 img_combine = np.hstack([img_gray, img_sobelx, img_sobely])
25
26 ps.CommonFunction.show_img_jupyter(img_combine)
27
28
29 #     print(end - start)
30 time.sleep(0.1)

```

这个例程测试了 X 和 Y 两个方向的 Sobel 算子。

10.1.4 结果展示

结果如图所示：

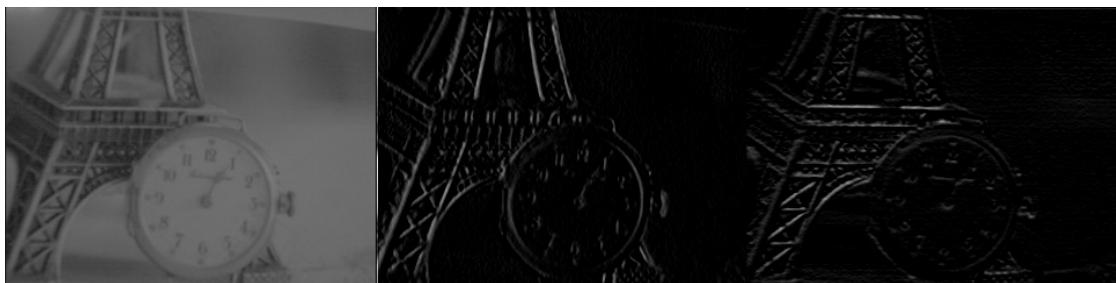


图 10.2: Sobel 算子实验结果

从左到右分别是原图，X 方向的 Sobel 算子求出的边缘，Y 方向的 Sobel 算子求出的边缘。可以看出可以发现 X 方向的 Sobel 对垂直方向的纹理更敏感（如塔的轮廓），而 Y 方向的 Sobel 对水平方向的纹理更敏感（如钟上的英文）。

10.2 Laplacian 算子

10.2.1 原理讲解

Laplace 算子实现的方法是先用 Sobel 算子计算二阶 x 和 y 导数，再求和。因此 Laplacian 算子对横竖两个方向的纹理都很敏感。

Laplace 算子是最简单的各向同性微分算子，具有旋转不变性。一个二维图像函数的拉普拉斯变换是各向同性的二阶导数，公式为：

$$\Delta = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

如果在图像中一个较暗的区域中出现了一个亮点，Laplace 算子就会使这个亮点变得更亮。因为图像中的边缘就是那些灰度发生跳变的区域，所以 Laplace 算子在边缘检测中很有用。一般增强技术对于陡峭的边缘和缓慢变化的边缘很难确定其边缘线的位置。但此算子却可用二次微分正峰和负峰之间的过零点来确定，对孤立点或端点更为敏感，因此特别适用于以突出图像中的孤立点、孤立线或线端点为目的的场合。同梯度算子一样，Laplace 算子也会增强图像中的噪声，有时用 Laplace 算子进行边缘检测时，可将图像先进行平滑处理。Laplace 算子的几何解释如下图所示：

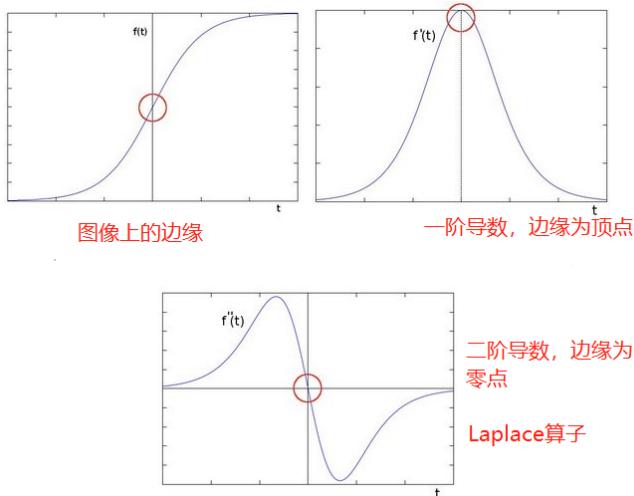


图 10.3: Laplace 算子几何原理

一个 3×3 的 Laplace 算子如下所示：

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

10.2.2 函数介绍

```

1 cv2.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]]) →
2     dst
3 # -src, 必须, 原图;
4 # -ddepth, 必须, 输出图像的像素精度, 可以是CV_8U、CV_16U、CV_16S、CV_32F、
5     CV_64F 中的一个;
6 # -dst, 可选, 输出图像;
7 # -ksize, 可选, 卷积核大小, 必须是正奇数;
8 # -scale, 可选, 详情见官方文档, 默认不设置;
9 # -delta, 可选, 可选的偏移值, 会被直接加到结果上;
10 # -borderType, 可选, 边界拓展方式(see borderInterpolate for details);

```

10.2.3 参考例程

Laplacian 算子的参考例程如下：

```

1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片，在同一位显示，不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为320x240尺寸
9 origin = cv2.resize(imgMat, (320,240))
10 rows,cols,_ = origin.shape
11
12 # ----- 图像处理开始
13
14 start = time.time()          # 记录开始时间
15
16 # 把图片转换成灰度图
17 img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
18 img_lap = cv2.Laplacian(img_gray, cv2.CV_16S, ksize=3) # ksize是卷积核的大小，换成16位是因为算子过程可以使原图像的值溢出。
19
20 end = time.time()          # 记录结束时间
21 # ----- 图像处理结束
22
23 #     # 把图像拼接在一起显示
24 img_combine = np.hstack([img_gray, img_lap])
25 ps.CommonFunction.show_img_jupyter(img_combine)
26
27
28 #     print(end - start)
29 time.sleep(0.1)

```

10.2.4 结果展示

结果如图所示：

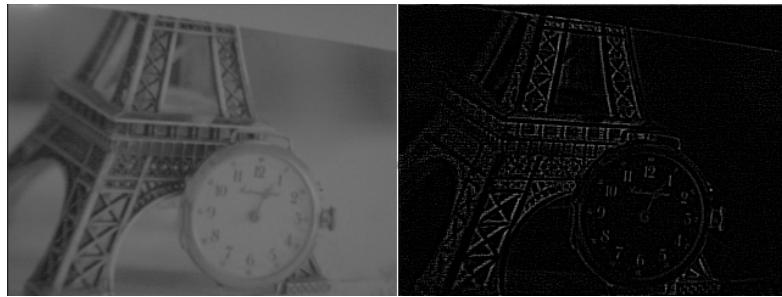


图 10.4: Laplace 算子实验结果

从左到右分别是原图，Laplace 算子求出的边缘。可以看出可以发现 Laplace 算子对水平和垂直方向的边缘都有很好检测效果。

10.3 Roberts 算子

10.3.1 原理讲解

Roberts 边缘算子（交叉梯度算子）是一个 2×2 的模板，这个算子在检测边缘效果非常好，但是它对噪声相当敏感，适用于边缘明显且噪声较少的图像分割。

Roberts 算子对边缘检测的作用是提供边缘候选点，相比于其他 3×3 算子，它可以给出相对较细的边缘。

常见的 Roberts 算子有以下两种：

$$G_x = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \text{ and } G_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

其中 G_x 是 X 方向的 Roberts 算子， G_y 是 Y 方向的 Roberts 算子。

10.3.2 函数介绍

Roberts 算子 opencv 没有提供直接的函数，不过可以用它提供的卷积滤波函数 cv2.filter2D() 自行构建。

```

1 cv2.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]]) ->
2     dst
3 # -src, 必须, 原图;
4 # -ddepth, 必须, 输出图像的像素精度, 可以是CV_8U、CV_16U、CV_16S、CV_32F、
5 # CV_64F 中的一个;
6 # -kernel, 可选, 卷积核, 一般是一个numpy的方阵, 数据类型为单精度浮点;
7 # -dst, 可选, 输出图像;
8 # -anchor, 可选, 卷积核的锚点, 详情见官方解释;
9 # -delta, 可选, 可选的偏移值, 会被直接加到结果上;
10 # -borderType, 可选, 边界拓展方式(see borderInterpolate for details);

```

10.3.3 参考例程

Roberts 算子的参考例程如下：

```

1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片，在同一位显示，不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为320x240尺寸
9 origin = cv2.resize(imgMat, (320,240))
10 rows,cols,_ = origin.shape
11
12 # ----- 图像处理开始
13
14 start = time.time()          # 记录开始时间
15
16 # 把图片转换成灰度图
17 img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
18 kernelx = np.array([[-1,0],[0,1]], dtype=int)
19 kernely = np.array([[0,-1],[1,0]], dtype=int)
20 img_robertsX = cv2.filter2D(img_gray, cv2.CV_16S, kernelx)
21 img_robertsY = cv2.filter2D(img_gray, cv2.CV_16S, kernely)
22
23 end = time.time()          # 记录结束时间
24 # ----- 图像处理结束
25
26 # 把图像拼接在一起显示
27 img_combine = np.hstack([img_gray, img_robertsX, img_robertsY])
28 ps.CommonFunction.show_img_jupyter(img_combine)
29
30
31 #     print(end - start)
32 time.sleep(0.1)

```

10.3.4 结果展示

结果如图所示：



图 10.5: Roberts 算子实验结果

从左到右分别是原图, X 轴的 Roberts 算子求出的边缘, Y 轴的 Roberts 算子求出的边缘。可以看出可以发现 Roberts 算子检测出来的边缘比较细, 不容易看出来。

10.4 Canny 算子

10.4.1 原理讲解

Canny 算子是 Sobel 算子的拓展版, 它目标是找到一个最优的边缘检测算法, 最优边缘检测的含义是:

- 好的检测 - 算法能够尽可能多地标识出图像中的实际边缘。
- 好的定位 - 标识出的边缘要与实际图像中的实际边缘尽可能接近。
- 最小响应 - 图像中的边缘只能标识一次, 并且可能存在的图像噪声不应标识为边缘。

为了实现这些功能 Canny 算子主要进行了以下步骤:

1. 噪音去除

由于边缘检测很容易受到噪音影响, 所以 Canny 算子使用一个高斯滤波器去除噪音。Canny 算子常用的是一个 5 维的高斯核, 如下所示:

$$K = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2. 计算图像梯度

对平滑后的图像使用 Sobel 计算水平方向和竖直方向的一阶导数 (图像梯度) (G_x 和 G_y)。根据得到的这两幅梯度图找到边界的梯度和方向, 公式见 Sobel 算子部分。

3. 滞后阈值

较高的亮度梯度比较有可能是边缘, 但是没有一个确切的值来限定多大的亮度梯度是边缘多大又不是, 所以 Canny 使用了滞后阈值。Canny 算子有两个阀值参数: minVal 和 maxVal 。当图像的灰度梯度高于 maxVal 时被认为是真正的边界, 那些低于 minVal 的边界会被抛弃。如果介于两者之间的话, 就要看这个点是否与某个被确定为真正边界点相连, 如果是, 就认为它也是边界点, 如果不是就抛弃。滞后阈值的

几何原理所示：

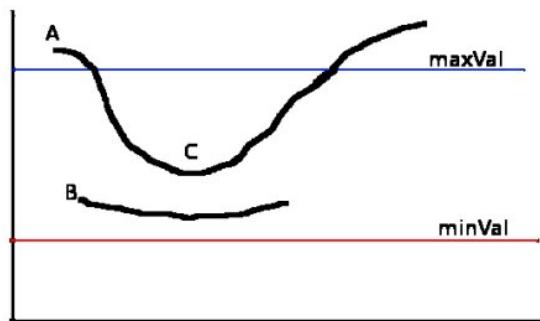


图 10.6：滞后阈值的几何解释

其中 A 点大于高阈值所以是边缘，C 在两个阈值中间且与 A 联通，所以 C 也是边缘，而 B 则不是边缘。

10.4.2 函数介绍

OpenCV 提供了强大的 Canny() 算子函数：

```

1 cv2.Canny(src, lowThreshold, highThreshold, apertureSize) → dst
2 # -src, 必须, 原图;
3 # -lowThreshold, 必须, 低阈值, 解释见原理部分;
4 # -highThreshold, 必须, 高阈值, 解释见原理部分;
5 # -apertureSize, 可选, 使用的Sobel算子的核的大小;
6 # -dst, 可选, 输出图像;

```

10.4.3 参考例程

Canny 算子的参考例程如下：

```

1 img = None
2 for i in range(100):
3
4 clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印多张图片
5
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为 320x240 尺寸
9 origin = cv2.resize(imgMat, (320, 240))
10 rows, cols, _ = origin.shape
11
12 # ----- 图像处理开始
13
14
13 start = time.time()      # 记录开始时间
14
15 # 把图片转换成灰度图
16 img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)

```

```

17 img_canny = cv2.Canny(img_gray, 30, 60, apertureSize=3) # apertureSize 是
   sobel 算子的卷积核大小
18
19 end = time.time()          # 记录结束时间
20 # _____ 图像处理结束
21
22 #      # 把图像拼接在一起显示
23 img_combine = np.hstack([img_gray, img_canny])
24
25 ps.CommonFunction.show_img_jupyter(img_combine)
26
27
28 #      print(end - start)

```

10.4.4 结果展示

结果如图所示：



图 10.7: Canny 算子实验结果

从左到右分别是原图、Canny 算子求得的边缘。可以看出相比于上述的几种边缘提取方法，Canny 算子的效果应该是最好的，就是计算量比较大。

10.5 小结一下

结论 本章主要介绍了几种常用的边缘滤波方法，边缘特征从数学上说就是梯度特征，掌握这些滤波算子对图像处理水平提高有着重要的意义。

第十一章 Tutorial 7 - 图像的特征（三）频域特征

内容提要

- 离散傅里叶变换
- 高通滤波
- 低通滤波

本章实验需要的配件：

- Powersensor 传感器
- 边缘测试图片（有船的那一张）



图 11.1: 频域测试图

有时候在空间域分析图像特征比较艰难的时候，可以试试把图像转换到频域进行分析。

11.1 离散傅里叶变换

11.1.1 原理讲解

离散傅里叶（DFT）变换是一种把空间域图像变化到频域图像的一种数学变换方法，它的公式是：

$$F(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) e^{-i(ux+vy)} dx dy$$

这个公式可以看起来像是在 X 和 Y 方向上分别做一维傅里叶变换，然后再把他们的乘积叠加。

在频率域中，高频分量表示图像中灰度变换比较快的地方，比如物体的边缘就是灰度的突然变化，所以物体边缘就是高频分量。而物体内部比较平坦的区域，灰度基本没有变化，对应的就是低频分量。

考虑到离散傅里叶变换的原理比较抽象难懂，这里只介绍几个它的基本特性：

1. 频域图的周期性

DFT 变换后频域图会沿着 X 和 Y 方向无限拓展，如图所示：

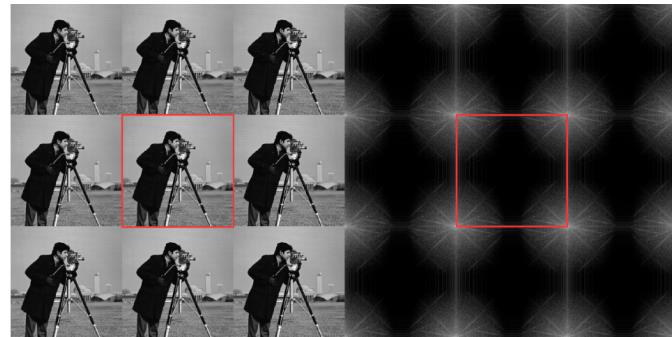


图 11.2: 频域图的周期性

2. 频域图的中心化

由于 DFT 变换时用的指数项是 $e^{i\pi} = -1$, 所以在得到原始频域图后往往需要乘以 $(-1)^{x+y}$ 来达到频域图居中的效果, 如图所示:

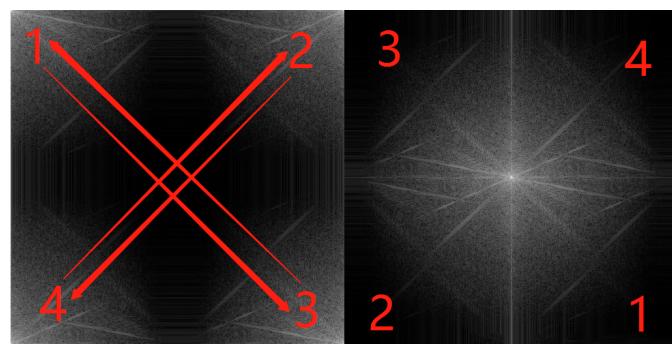


图 11.3: 频域图的居中

3. 频域图的解读

- 频域图的每个位置表征一种频率, 中心是低频, 四周是高频。
- 频域图用灰度值表示每种频率的含量, 越白的地方含量越高。
- Y 轴上的白点表示纵向交错的频率 (比如纵向分布的条纹), X 轴上的点表示横向交错的频率, 斜线的点也同理。

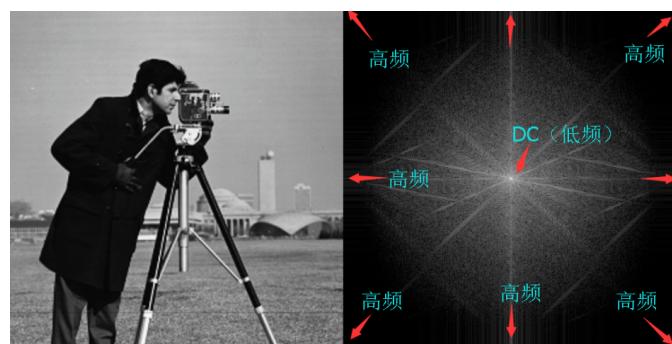


图 11.4: 频域图的解读

4. 频域图的旋转和平移

频域图具有平移不变性, 但是旋转会产生相位的偏移:

5. DFT 变换是可逆的操作。

11.1.2 函数介绍

DFT 变换主要使用 numpy 的函数来实现，这个是 2D 的 DFT 变换函数：

```

1 numpy.fft.fft2(a, s=None, axes=(-2, -1), norm=None)
2 # -a, 必须, 原图;
3 # -s, 可选, 输出的 shape;
4 # -axes 可选, 通道选择, 默认是None, 会对所有通道操作;
5 # -norm, 可选, 归一化的方式, 详情见官方文档;
```

下面这个是 FFT 中心化的函数，即原理讲解里面的乘-1：

```

1 numpy.fft.fftshift(x, axes=None)
2 # -x, 必须, 原图;
3 # -axes 可选, 通道选择, 默认是None, 会对所有通道操作;
4 # -norm, 可选, 归一化的方式, 详情见官方文档;
```

11.1.3 参考例程

这个例程是先对图像进行傅里叶变换，然后中心化，再反中心化，再傅里叶变换，变回原图。详细代码如下：

```

1 for i in range(20):
2
3     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
4     imgMat = cam1.read_img_ori()      # 读入图像
5     # 缩小图像为 320x240 尺寸
6     origin = cv2.resize(imgMat, (320, 240))
7
8     # ----- 图像处理开始
9
10
11    f = np.fft.fft2(origin)      # 傅里叶变换
12    fImg = abs(f)
13
14    fshift = np.fft.fftshift(f)    # 移动
15    ishift = np.fft.ifftshift(fshift) # 移动回去
16    iimg = np.fft.ifft2(ishift)   # 逆变换
17    iimg = np.abs(iimg)      # 取绝对值变为实数
18
19    end = time.time()          # 记录结束时间
20    # ----- 图像处理结束
21
22
23    # 把图像拼接在一起显示
24    img_combine = np.hstack([origin, iimg])
25    ps.CommonFunction.show_img_jupyter(img_combine) # 打印用于差分的两张图片
```

```

25
26 print(end - start)
27 time.sleep(0.1)

```

11.1.4 结果展示

结果如图所示：



图 11.5：傅里叶变换实验

从左到右分别是原图，然后是两次傅里叶变换后还原的图像。

11.2 频域高通滤波

11.2.1 原理讲解

高通滤波就是滤除低频的成分，保留高频的成分。从上一节的原理讲解中，我们知道经过 DEF 变换及中心化后，中间的部分是低频的，周围是高频的。因此高通滤波，只需把低频的这些地方变成黑（0），然后再反变换回去即可。

11.2.2 函数介绍

简单的高通滤波不需要专门的函数，用以前介绍过的图像裁剪的方法取出频域图中中心部分的像素值赋值零即可。

11.2.3 参考例程

频域高通滤波的示例代码如下：

```

1 for i in range(100):
2
3     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
4     imgMat = cam1.read_img_ori()    # 读入图像
5
6     # 缩小图像为320x240尺寸

```

```

7 origin = cv2.resize(imgMat, (320,240))
8 origin = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
9
10 # ----- 图像处理开始 -----
11
12 start = time.time()          # 记录开始时间
13
14 f = np.fft.fft2(origin)      # 傅里叶变换
15 fshift = np.fft.fftshift(f)   # 低频点移动到中心
16
17 # 构造掩模，去掉高频，低通滤波器
18 rows,cols = origin.shape
19 crow,ccol = int(rows/2),int(cols/2) # 中心
20 mask = np.zeros((rows,cols),np.uint8)
21 mask[crow-30:crow+30,ccol-30:ccol+30] = 1
22 fshift = fshift*mask
23
24 # ifft
25 ishift = np.fft.ifftshift(fshift) # 移动回去
26 iimg = np.fft.ifft2(ishift)    # 逆变换
27 iimg = np.abs(iimg)           # 取绝对值变为实数
28
29 end = time.time()            # 记录结束时间
30 # ----- 图像处理结束 -----
31
32 # 把图像拼接在一起显示
33 img_combine = np.hstack([origin, iimg])
34 ps.CommonFunction.show_img_jupyter(img_combine) # 打印用于差分的两张图片
35
36 print(end - start)
37 time.sleep(0.1)

```

11.2.4 结果展示

结果如图所示：

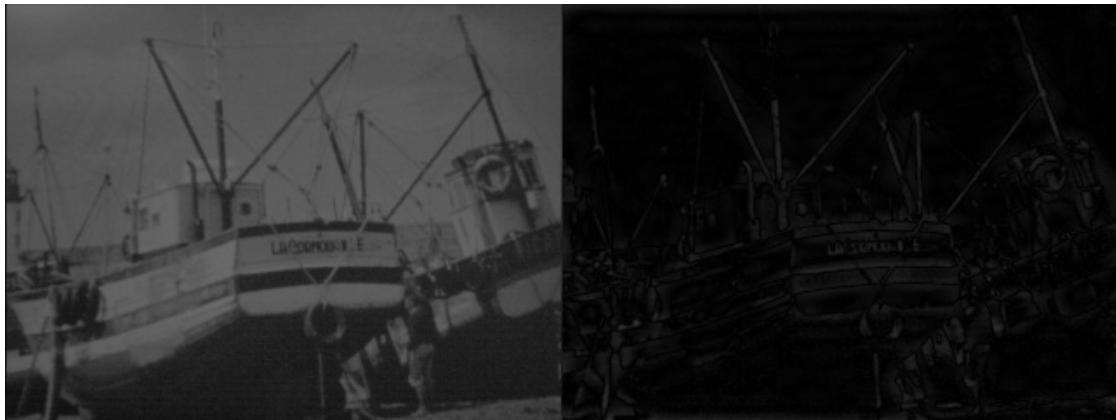


图 11.6: 频域高通滤波实验

从左到右分别是原图，然后是高通滤波后还原的图像。可以看出，高通滤波后，图像上只剩下变化比较剧烈的边缘部分了。

11.3 频域低通滤波

11.3.1 原理讲解

低通滤波和高通滤波类似，不过实现时候可以通过增强低频部分（中心化的频域图的中间部分）来实现。

11.3.2 函数介绍

简单的低通滤波不需要专门的函数，用以前介绍过的图像裁剪的方法取出频域图中心部分的像素值赋值一即可。

11.3.3 参考例程

频域低通滤波的示例代码如下：

```

1 for i in range(100):
2
3     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
4     imgMat = cam1.read_img_ori()    # 读入图像
5
6     # 缩小图像为320x240尺寸
7     origin = cv2.resize(imgMat, (320, 240))
8     origin = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
9
10    # ----- 图像处理开始 -----
11
12    start = time.time()          # 记录开始时间
13
14    # fft

```

```

14 f = np.fft.fft2(origin)          # 傅里叶变换
15 fshift = np.fft.fftshift(f)      # 低频点移动到中心
16
17 # 构造掩模，去掉高频，低通滤波器
18 rows,cols = origin.shape
19 crow,ccol = int(rows/2),int(cols/2) # 中心
20 mask = np.zeros((rows,cols),np.uint8)
21 mask[crow-30:crow+30,ccol-30:ccol+30] = 1
22 fshift = fshift*mask
23
24 # ifft
25 ishift = np.fft.ifftshift(fshift) # 移动回去
26 iimg = np.fft.ifft2(ishift)     # 逆变换
27 iimg = np.abs(iimg)            # 取绝对值变为实数
28
29 end = time.time()             # 记录结束时间
30 # -----
31 # 图像处理结束
32
33 # 把图像拼接在一起显示
34 img_combine = np.hstack([origin, iimg])
35 ps.CommonFunction.show_img_jupyter(img_combine) # 打印用于差分的两张图片
36
37 print(end - start)
38 time.sleep(0.1)

```

11.3.4 结果展示

结果如图所示：

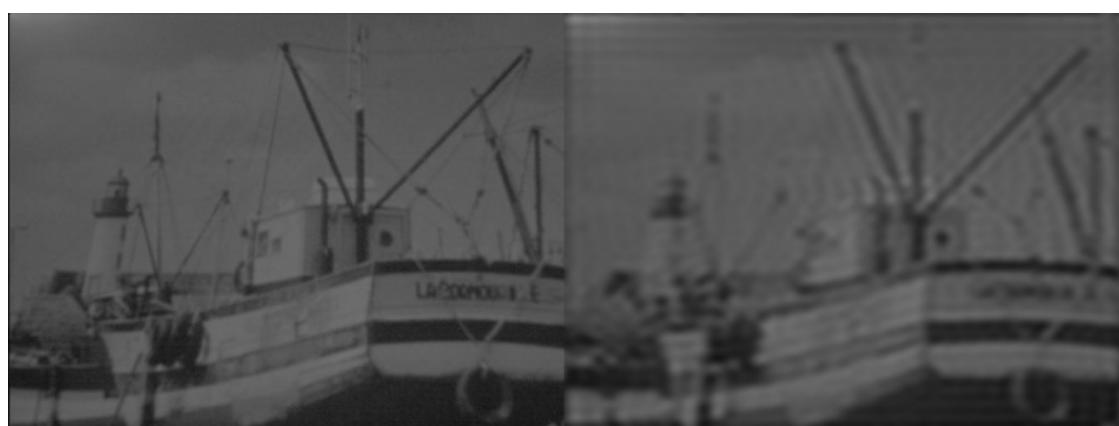


图 11.7：频域低通滤波实验

从左到右分别是原图、低通滤波后还原的图像。可以看出，低通滤波后，图像的边缘变的模糊了。

11.4 小结一下

结论 本章主要介绍了离散傅里叶变换及其简单的频率高低通滤波，频域是一个很神奇的东西，它很抽象，不好懂，但是有时候用的好可以起到意想不到的效果。

第十二章 Tutorial 8 - 图像的轮廓提取

内容提要

阈值二值化

自适应二值化

本章实验需要的配件:

- Powersensor 传感器
- 轮廓提取测试图片 (有三角形圆心等很多形状的那一张)

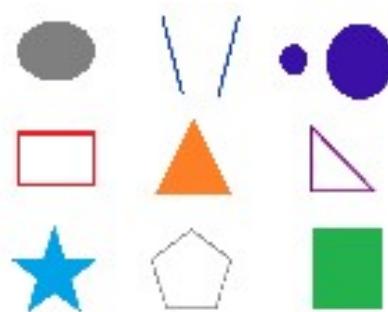


图 12.1: 轮廓提取测试图

什么是轮廓? 简单地说, 轮廓是一个封闭的曲线, 这条曲线把拥有相似灰度值的边缘连起来, 比如一个苹果的边缘就构成一个轮廓。轮廓是一个非常有用的工具, 尤其对物体检测和分类等应用有重要意义。

使用轮廓提取有以下注意点:

- 为了提高轮廓提取的精度和鲁棒性, 轮廓提取时应该使用二值化后的图像。
- 轮廓提取得到的是一组轮廓曲线, 要注意遍历。
- 轮廓提取一般假设在黑色的背景里寻找白色的轮廓, 所以需要根据实际情况对图像进行反相操作。

由于 OpenCV 提供了非常强大的 `findContours()` 函数来完成轮廓提取, 一般直接使用即可, 具体的原理可以查看这篇论文: [Topological structural analysis of digitized binary images by border following](#)

而我们日常进行图像轮廓处理的一个重要工作就是如何把图像进行合理的二值化, 接下来介绍常用的二值化方法。

12.1 固定阈值二值化

12.1.1 原理讲解

全局阈值就是一幅图像包括目标物体、背景还有噪声, 要想从多值的数字图像中直接提取出目标物体; 常用的方法就是设定一个阈值 T , 用 T 将图像的数据分成两部分: 大

于 T 的像素群和小于 T 的像素群。这是研究灰度变换的最特殊的方法，称为图像的二值化（Binarization）。

固定阈值二值化根据二值化过程不同可以分成以下几种类型：

1. 原图，实线为待二值化的像素点的灰度值，虚线为阈值

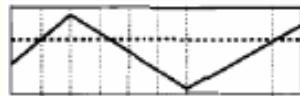


图 12.2：阈值化前的原图

2. THRESH_BINARY，大于阈值部分设置为 255，小于部分设置为 0，是最常用的二值化



图 12.3：THRESH_BINARY

3. THRESH_BINARY_INV，与 THRESH_BINARY 相反



图 12.4：THRESH_BINARY

4. THRESH_TRUNC，大于阈值部分设置为阈值，小于部分保持不变

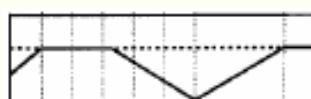


图 12.5：THRESH_TRUNC

5. THRESH_TOZERO，大于阈值部分保持不变，小于部分设置为 0

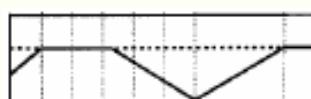


图 12.6：THRESH_TOZERO

6. THRESH_TOZERO_INV，大于阈值部分设置为 0，小于部分保持不变

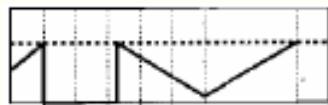


图 12.7: THRESH_BINARY

12.1.2 函数介绍

1. 轮廓寻找函数

```

1 cv2.findContours(image, mode, method[, contours[, hierarchy[, offset
   ]]]) -> contours, hierarchy
2 # -image, 必须, 原图, 要求是8bit的单通道图片;
3 # -mode, 必须, 搜索方式, RETR_EXTERNAL, RETR_LIST, RETR_CCOMP,
   RETR_TREE 中的一个;
4 #      --RETR_EXTERNAL, 只搜索最外侧的轮廓;
5 #      --RETR_LIST, 搜索所有轮廓, 而且不建立轮廓间的层次关系;
6 #      --RETR_CCOMP, 搜索所有轮廓, 并建立一个二层的关系, 顶层包括所
   有的外轮廓, 第二层包含一些内部的孔的边界;
7 #      --RETR_TREE, 搜索所有轮廓, 并简历一个树状的层次结构;
8 # -method, 可选, 轮廓近似方式, CHAIN_APPROX_NONE,
   CV_CHAIN_APPROX_SIMPLE,CV_CHAIN_APPROX_TC89_L1,
   CV_CHAIN_APPROX_TC89_KCOS 等中的一个;
9 #      --CHAIN_APPROX_NONE, 无近似, 保留所有轮廓的点;
10 #      --CV_CHAIN_APPROX_SIMPLE, 比较水平、垂直、对角的各个线段, 只
    留下端点;
11 #      --CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS, 见官方
    注解;
12 # -hierarchy, 可选, 输出, 图像的拓扑结构信息;
13 # -offset, 可选, 位置偏置, 会被直接加到输出上, 当分析的图片是原来一张
    大图的一部分时比较有用;
14 # -contours, 可选, 输出, 找到的轮廓;
15

```



笔记 mode 中, RETR_CCOMP 与 RETR_TREE 的区别在于, 前者是一个两层的结构, 如果第二层的内部又包含一些轮廓, 那些轮廓会被当成新的顶层, 而后者不会, 会把所有的内层轮廓当成内层轮廓。

2. 阈值二值化函数

```

1 cv2.threshold(src, thresh, maxval, type[, dst]) -> retval, dst
2 # -src, 必须, 原图;
3 # -thresh, 必须, 阈值;
4 # -maxval, 必须, 最大值, 影响type中的最大值;
5 # -type, 可选, 类型, 见原理讲解部分, 是THRESH_BINARY,
   THRESH_BINARY_INV, THRESH_TRUNC, THRESH_TOZERO, THRESH_TOZERO_INV中
   的一个;
6 # -dst, 可选, 输出的二值化图片;
7

```

12.1.3 参考例程

固定阈值二值化及轮廓提取的参考例程如下：

```

1 for i in range(100):
2
3     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
4     imgMat = cam1.read_img_ori()    # 读入图像
5
6     # 缩小图像为320x240尺寸
7     origin = cv2.resize(imgMat, (320,240))
8
9     # ----- 图像处理开始 -----
10
10    start = time.time()          # 记录开始时间
11
12    img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY) # 转成灰度图
13    ret, img_binary = cv2.threshold(img_gray, 120, 255, cv2.THRESH_BINARY_INV) # 全局二值化
14    thresh, contours, hierarchy = cv2.findContours(img_binary, cv2.RETR_EXTERNAL,
15                                                    cv2.CHAIN_APPROX_SIMPLE) # 查找轮廓
15    cv2.drawContours(img_gray, contours, -1, 0, 3) # 把轮廓画在灰度图上
16
17
18    end = time.time()          # 记录结束时间
19    # ----- 图像处理结束 -----
20
20
21    # 把图像拼接在一起显示
22    img_combine = np.hstack([img_gray, img_binary])
23    ps.CommonFunction.show_img_jupyter(img_combine) # 打印用于差分的两张图片
24
25    print(end - start)
26    time.sleep(0.1)

```

12.1.4 结果展示

结果如图所示：



图 12.8：固定阈值二值化及轮廓提取

从左到右分别是原图和轮廓，二值化的图片。

12.2 自适应二值化

12.2.1 原理讲解

局部阈值就是当同一幅图像上的不同部分的具有不同亮度时。这种情况下我们需要采用自适应阈值。此时的阈值是根据图像上的每一个小区域计算与其对应的阈值。因此在同一幅图像上的不同区域采用的是不同的阈值，从而使我们能在亮度不同的情况下得到更好的结果。

一个理想的自适应阈值算法应该能够对光照不均匀的图像产生类似上述固定阈值算法对光照均匀图像产生的效果一样好。为了补偿或多或少的照明，每个像素的亮度需要正则化，之后才能决定某个像素是黑色还是白色。问题是如何决定每个点的背景亮度。

自适应阈值算法基本的细想就是遍历图像，计算一个移动的平均阈值。如果某个像素明显的低于这个平均值，则设置为黑色，否则设置为白色。

opencv 的自适应二值化在计算输出的时候需要指定阈值类型，THRESH_BINARY 或 THRESH_BINARY_INV：

1. THRESH_BINARY

$$dst(x, y) = \begin{cases} maxValue & if src(x, y) > T(x, y) \\ 0 & otherwise \end{cases}$$

2. THRESH_BINARY

$$dst(x, y) = \begin{cases} 0 & if src(x, y) > T(x, y) \\ maxValue & otherwise \end{cases}$$

12.2.2 函数介绍

```

1 cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize
   , C[, dst]) → dst
2 # -src, 必须, 原图;
3 # -maxval, 必须, 最大值, 影响thresholdType 中的最大值;
4 # -adaptiveMethod, 必须, 自适应方法, ADAPTIVE_THRESH_MEAN_C,
   ADAPTIVE_THRESH_GAUSSIAN_C 中的一个, 前者是采用窗口内像素的均值作为阈值,
   后者是考虑窗口内像素的高斯加权的均值位置作为均值;
5 # -thresholdType, 必须, 类型, 见原理讲解部分, 是THRESH_BINARY,
   THRESH_BINARY_INV 中的一个;
6 # -blockSize, 必须, 自适应窗口大小, 必须是奇数, 3, 5, 7...
7 # -C, 可选, 常数偏置, 直接被加到阈值上, 一般是正数, 也可以是0或负数, 正数可
   以较少噪声以及不明显的边缘;
8 # -dst, 可选, 输出的二值化图片;

```

12.2.3 参考例程

自适应阈值二值化及轮廓提取的参考例程如下：

```

1 for i in range(100):
2
3     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
4     imgMat = cam1.read_img_ori()      # 读入图像
5
6     # 缩小图像为320x240尺寸
7     origin = cv2.resize(imgMat, (320,240))
8
9     # ----- 图像处理开始 -----
10
10    start = time.time()          # 记录开始时间
11
12    img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY) # 转成灰度图
13    img_binary = cv2.adaptiveThreshold(img_gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
14                                       cv2.THRESH_BINARY, 3, 2) # 全局二值化，
14    thresh, contours, hierarchy = cv2.findContours(img_binary, cv2.RETR_EXTERNAL,
15                                                   cv2.CHAIN_APPROX_SIMPLE) # 查找轮廓
15    cv2.drawContours(img_gray, contours, -1, 0, 3) # 把轮廓画在灰度图上
16
17
18    end = time.time()          # 记录结束时间
19    # ----- 图像处理结束 -----
20
20
21    # 把图像拼接在一起显示
22    img_combine = np.hstack([img_gray, img_binary])
23    ps.CommonFunction.show_img_jupyter(img_combine) # 打印用于差分的两张图片
24
25    print(end - start)
26    time.sleep(0.1)

```

12.2.4 结果展示

结果如图所示：

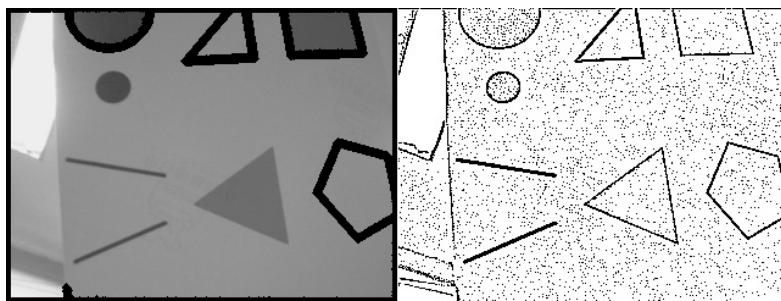


图 12.9：固定阈值二值化及轮廓提取

从左到右分别是原图和轮廓，二值化的图片。

12.3 小结一下

结论 本章主要介绍了轮廓提取和二值化的方法，对于光照均匀不变的情况固定阈值二值化的性能极佳，是最好的选择，但是对于光照变化明显、明暗相间的场景下，自适应二值化的稳定性更好。

第十三章 Tutorial 9 - 图像的形状检测

内容提要

直线检测

圆检测

本章实验需要的配件:

- Powersensor 传感器
- 轮廓提取测试图片 (有三角形圆心等很多形状的那一张)

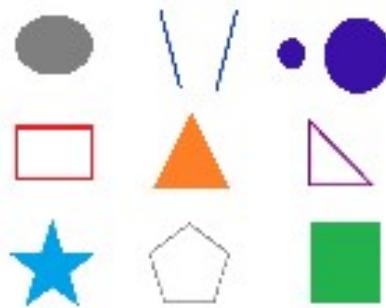


图 13.1: 轮廓提取测试图

形状检测一般在轮廓提取或者边缘运算之后进行。形状检测的方法很多，霍夫变换、模板匹配等，本章主要介绍霍夫变换用于直线和圆的检测。

Hough 变换（霍夫变换）主要用来识别已知的几何形状，最常见的比如直线、线段、圆形、椭圆、矩形等。霍夫变换检测的是原图的二值化后的图片。霍夫变换的原理是根据参数空间的统计规律进行参数估计。具体说来就是，将直角坐标系中的图形 (x, y) 变换到参数空间 (k_1, \dots, k_n) ，对直角坐标系中的每一个像素点，计算它在参数空间里的所有可能的参数向量。处理完所有像素点后，把出现次数（频率）最多的（一个或几个）参数向量的坐标作为参数代入直角坐标方程，即检测到的图形方程。

13.1 直线检测

13.1.1 原理讲解

在 2 维图像中，一条直线在直角坐标系中可以由斜率和截距两个变量表示。而在极坐标系中可以由极径 ρ 和极角 θ 表示，如图所示：

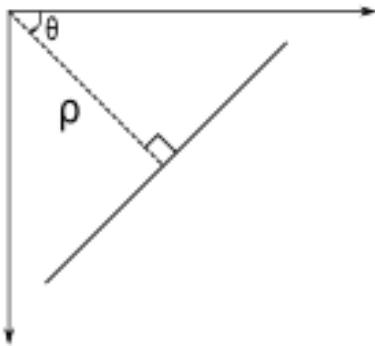


图 13.2: 极坐标系下的直线

基于霍夫变换的直线检测原理如下所示：

1. 极坐标系下的直线可以表示为：

$$\rho = x \cos \theta + y \sin \theta \quad (13.1)$$

其中 (x, y) 是其在直角坐标系下的座标。

2. 霍夫空间：

假设 $(x, y) = (x_1, y_1)$, 把 (ρ, θ) 看成变量, 那么式 13.1 可以表示极坐标下过 (x_1, y_1) 的所有直线。

设置 θ 为横轴, ρ 为纵轴, 那么由 (ρ, θ) 就构成一个 2 维的霍夫空间。

3. 直线的交点：

从式 13.1 还可以看出, 在霍夫空间中, 直线的 (ρ, θ) 曲线应该是一个正弦曲线。这条曲线代表了经过这个点的所有直线, 这条曲线上的每一个点都确定着一条直线。我们称原来二值化过那张的图像为原图, 霍夫空间里的图为统计图。那么原图里面的每一个亮点 (假设二值化后背景为黑色), 就对应统计图里的一条正弦线。

如果原图中的 n 个点在同一条直线上, 那么它们对应到霍夫空间里的曲线应该会有一个共同的交点。如下图所示:

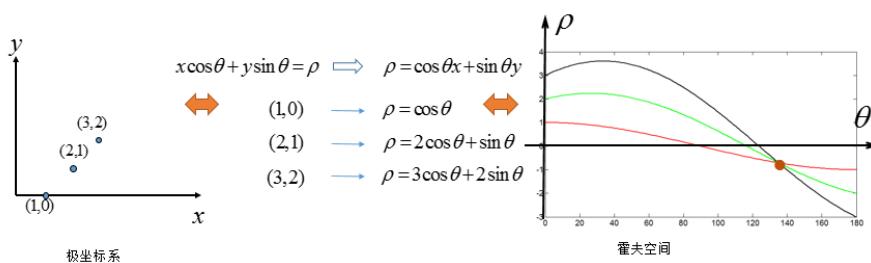


图 13.3: 霍夫变换原理

4. 统计

接着我们通过统计这些交点, 就可以得出哪些点构成了一条直线。这里涉及到 3 个重要的参数。

首先离多远可以算过同一个交点, 简单的说就是分辨率, 角度的分辨率表征的是两个 θ 差多少可以算同一个 θ , 比如要分辨相差 1 度的, 角度的分辨率应该设置成

$\pi/180$ (因为 opencv 使用弧度); 同理距离的分辨率, 就是 ρ 差多少可以算做同一个 ρ , 单位是像素。

第三个参数是阈值，表征的是霍夫空间里多少条曲线交到同一个点后，那个点才可以算成一条直线。

13.1.2 函数介绍

OpenCV 提供了霍夫变换的函数 cv2.HoughLines(), 及其改进版 cv2.HoughLinesP().

1. 直线检测函数: cv2.HoughLines().

```
1 cv2.HoughLines(image, rho, theta, threshold[, lines[, srn[, stn]]]) →  
2     lines  
3  
4 # -dst, 必须, 原图, 要求二值化后的图片;  
5 # -rho, 必须, \rho 的分辨率, 单位是像素, 原理中有介绍;  
6 # -theta, 必须, \theta 的分辨率, 单位是弧度, 原理中有介绍;  
7 # -threshold, 可选, 阈值, 一个直线上至少需要包含多少个霍夫空间的交点,  
8     原理中有介绍;  
9  
10    # -srn and stn, Default parameters to zero. Check OpenCV reference for  
11        more info.  
12  
13    # -lines, 可选, 输出的直线, 注意这个函数输出的是极坐标形式;
```

2 带概率筛选的直线检测函数:

```
1 v2.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[,  
maxLineGap]]]) → lines  
2 # -dst, 必须, 原图, 要求二值化后的图片;  
3 # -rho, 必须, \rho 的分辨率, 单位是像素, 原理中有介绍;  
4 # -theta, 必须, \theta 的分辨率, 单位是弧度, 原理中有介绍;  
5 # -threshold, 可选, 阈值, 一个直线上至少需要包含多少个霍夫空间的交点,  
原理中有介绍;  
6 # -minLineLength, 可选, 最小线长, 这个直线至少要包含多少个像素;  
7 # -maxLineGap, 可选, 最大线距, 被当作在同一条直线的两个点的最大间距;  
8 # -lines, 可选, 输出的直线, 注意这个函数输出的是直角坐标形式;
```

2

笔记 HoughLines() 输出的是极坐标形式的直线，HoughLinesP() 输出的是直角坐标形式的直线

13.1.3 参考例程

本例程使用带概率筛选的直线检测函数来实现。

```
1 for i in range(100):
2
3 clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
4 imgMat = cam1.read_img_ori()      # 读入图像
5
```

```

6 # 缩小图像为320x240尺寸
7 origin = cv2.resize(imgMat, (320,240))
8
9 # ----- 图像处理开始 -----
10 start = time.time()      # 记录开始时间
11
12 img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
13 img_canny = cv2.Canny(img_gray, 30, 60, apertureSize=3)  # apertureSize是
    sobel算子的卷积核大小
14
15 lines = cv2.HoughLinesP(img_canny, 1, np.pi/180, 40, minLineLength=40,
    maxLineGap=4)
16 if lines is None:
17     pass
18 else:
19     for i in range(len(lines)):
20         for x1,y1,x2,y2 in lines[i]:
21             cv2.line(img_gray, (x1,y1), (x2,y2), (0), 2)
22
23
24 end = time.time()      # 记录结束时间
25 # ----- 图像处理结束 -----
26
27 # 把图像拼接在一起显示
28 img_combine = np.hstack([img_gray, img_canny])
29 ps.CommonFunction.show_img_jupyter(img_combine) # 打印用于差分的两张图片
30
31 print(end - start)
32 time.sleep(0.1)

```

13.1.4 结果展示

结果如图所示：

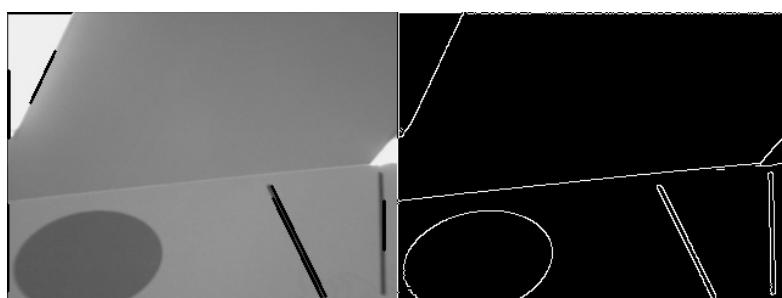


图 13.4: 直线检测

从左到右分别是原图和轮廓，二值化的图片。

13.2 圆检测

13.2.1 原理讲解

与直线检测的原理类似，圆检测也是通过霍夫变换来实现，不过与直角检测有以下几点差异。

1. 搜索目标差异

对一个原图上的一个白点，直线检测的过程是寻找所有经过这个点的直线的集合，并用一个方程表示，而圆检测是寻找所有经过这个点的圆的圆心的位置的集合（如果半径已知），这个式子可以表示成：

$$\begin{aligned}x &= x_1 + r \cos \theta \\y &= y_1 + r \sin \theta\end{aligned}\quad (13.2)$$

其中 (x_1, y_1) 就是原图上的点。在半径已知的情况下，可能经过点 (x_1, y_1) 的圆的圆心组成的轨迹其实也是一个圆，由上次也可以得出类似结论。

2. 霍夫空间的差异

由式13.2可得，在圆检测里，原图上的点转换得到霍夫空间有三个变量 (x, y, r) ，比直线检测多了一个。半径未知时，可能经过点 (x_1, y_1) 的圆的圆心组成的轨迹是一个圆锥，是三维立体的结构。

3. 交点的差异

直线检测统计的是 n 个正弦曲线的交点的个数，半径已知的圆检测统计的是 n 个圆的交点的个数，半径未知的圆检测统计的是 n 个圆锥的交点的个数。直观的图示如下：

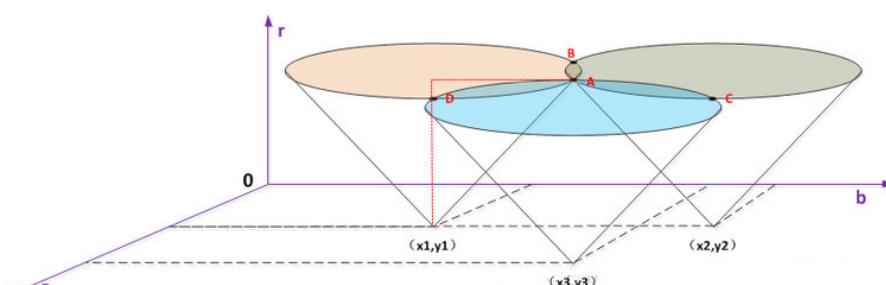


图 13.5: 霍夫圆检测原理

13.2.2 函数介绍

OpenCV 提供了检测圆的霍夫变换的函数 cv2.HoughCircles()。

```
1 cv2.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[,  
    minRadius[, maxRadius]]]]]) → circles  
2 # -image, 必须, 原图, 要求二值化后的图片;  
3 # -method, 必须, 检测方法, 目前仅支持CV_HOUGH_GRADIENT, 基于2HT实现;  
4 # -dp, 必须, 检测的缩放比例参数, 如果为1, 就是检测原图, 为2就是缩小一半检测,  
    影响计算效率;
```

```

5 # -minDist, 可选, 最小圆心距离。太小会导致一个圆被错误地检测成多个相邻的圆,
6   太大会导致一些圆检测不出来;
7 # -param1, 可选, 高阈值, 传递给canny()检测器的高阈值, 见注;
8 # -param2, 可选, 低阈值, 传递给canny()检测器的低阈值;
9 # -minRadius, 可选, 检测的最小圆半径
10 # -maxRadius, 可选, 检测的最大圆半径
11 # -circles, 可选, 输出的圆;

```

注 Opencv 实际使用的圆检测方法和原理介绍里类似, 但是更复杂一些, 详情见Yuen90

注 关于 Canny 检测器的高/低阈值, 详情见“图像的特征 2-梯度梯度特征”的"Canny 算子" 部分。

13.2.3 参考例程

本例程使用霍夫变换的圆检测函数来实现:

```

1 for i in range(100):
2
3     clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印多张图片
4     imgMat = cam1.read_img_ori()      # 读入图像
5
6     # 缩小图像为320x240尺寸
7     origin = cv2.resize(imgMat, (320,240))
8
9     # ----- 图像处理开始
10
11
12     start = time.time()          # 记录开始时间
13
14     img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
15     img_canny = cv2.Canny(img_gray, 30, 50, apertureSize=3)    # apertureSize是
16           sobel算子的卷积核大小
17
18     circles = cv2.HoughCircles(img_gray, cv2.HOUGH_GRADIENT, 1, 40, param1=50,
19         param2=35, minRadius=0, maxRadius= 80)
20     if circles is None:
21         pass
22     else:
23         circles = np.uint16(np.around(circles))
24         for i in circles[0,:]:
25             # draw the outer circle
26             cv2.circle(img_gray,(i[0],i[1]),i[2],(0,255,0),2)
27             # draw the center of the circle
28             cv2.circle(img_gray,(i[0],i[1]),2,(0,0,255),3)
29
30
31     end = time.time()            # 记录结束时间
32     # ----- 图像处理结束

```

```

29
30 # 把图像拼接在一起显示
31 img_combine = np.hstack([img_gray, img_canny])
32 ps.CommonFunction.show_img_jupyter(img_combine) # 打印用于差分的两张图片
33
34 print(end - start)
35 time.sleep(0.1)

```

13.2.4 结果展示

结果如图所示：

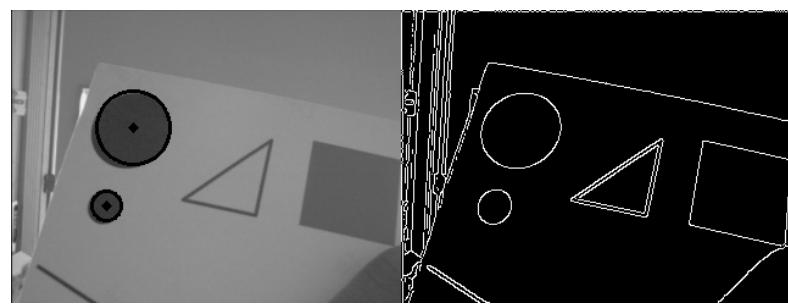


图 13.6：圆检测

从左到右分别是原图和轮廓，二值化的图片。

13.3 小结一下

结论 本章主要介绍了基于霍夫变换的直线检测和圆检测方法，从数学的角度来说，直线和圆都可以用参数来表示，直线和圆的检测就是这些参数的求取，而霍夫变换的核心就是把所求的参数变化化，然后就可以通过优化理论来求解最优的（或者满足需求的）参数。霍夫变换还可以检测许多其他的形状，有兴趣的用户可以上网搜索。

第十四章 Tutorial 10 - 动态图形处理基础

内容提要

□ 帧差分

本章实验需要的配件:

- Powersensor 传感器

14.1 帧检测

14.1.1 原理讲解

帧间差分法是通过比较两帧之间的像素差来确定运动的物体，是一种常用的运动检测方法。如果物体是运动的，那么在视频中两帧之间其所在位置的像素是不一样的，我们只需要求出两帧之间的像素差，并设定一个阈值，就能轻易地检测出视频中运动的物体。

帧间差分法是一种通过对视频图像序列的连续两帧图像做差分运算获取运动目标轮廓的方法。当监控场景中出现异常目标运动时，相邻两帧图像之间会出现较为明显的差别，两帧相减，求得图像对应位置像素值差的绝对值，判断其是否大于某一阈值，进而分析视频或图像序列的物体运动特性。其数学公式描述如下：

$$D(x, y) = \begin{cases} 1 & \text{if } |I(t) - I(t-1)| > T \\ 0 & \text{otherwise} \end{cases} \quad (14.1)$$

其中 $D(x, y)$ 为连续两帧图像之间的差分图像， $I(t)$ 和 $I(t-1)$ 分别为 t 和 $t-1$ 时刻的图像， T 为差分图像二值化时选取的阈值， $D(x, y) = 1$ 表示前景， $D(x, y) = 0$ 表示背景。

其具体步骤为：

1. 读取视频
2. 选取两帧
3. 灰度化
4. 滤波
5. 帧间做差
6. 二值化
7. 形态学操作

14.1.2 函数介绍

```
1 cv2.absdiff(src1,src2)) → retval
2 # -src1, 必须, 背景图片, 灰度图像;
3 # -src2, 必须, 前景图片, 灰度图像;
```

4 # -retval, 函数返回值为差分图像。

14.1.3 参考例程

```

1 def absdiff_demo(backgroundImg, currentImg, Threshold):
2     gray_image_1 = cv2.cvtColor(backgroundImg, cv2.COLOR_BGR2GRAY)    # 灰度化
3     gray_image_1 = cv2.GaussianBlur(gray_image_1, (3, 3), 0)          # 高斯滤波
4     gray_image_2 = cv2.cvtColor(currentImg, cv2.COLOR_BGR2GRAY)
5     gray_image_2 = cv2.GaussianBlur(gray_image_2, (3, 3), 0)
6     d_frame = cv2.absdiff(gray_image_1, gray_image_2)
7     ret, d_frame = cv2.threshold(d_frame, Threshold, 255, cv2.THRESH_BINARY)
8     return d_frame
9
10 Threshold = 12 # sThre 表示像素阈值
11 i = 0
12
13 # currentImg = np.ones((320,240),dtype=np.uint8)#random.random()方法后面不能
14      加数据类型
15
16 for i in range(100):
17     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
18     imgMat = cam1.read_img_ori()      # 读入图像
19     # 缩小图像为320x240尺寸
20     origin = cv2.resize(imgMat, (320,240))
21     img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)    # 灰度化
22
23     # ----- 图像处理开始
24
24     start = time.time()          # 记录开始时间
25
26     # 第一帧不处理
27     if(i==0):
28         currentImg = origin
29         backgroundImg = currentImg
30         diffImg = absdiff_demo(backgroundImg, currentImg, Threshold)
31     else:
32         backgroundImg = currentImg
33         currentImg = origin
34         diffImg = absdiff_demo(backgroundImg, currentImg, Threshold)
35
36     end = time.time()          # 记录结束时间
37     # ----- 图像处理结束
38
39     # 把图像拼接在一起显示

```

```
40 img_combine = np.hstack([img_gray, diffImg])
41 ps.CommonFunction.show_img_jupyter(img_combine) # 打印用于差分的两张图片
42 #     ps.CommonFunction.show_img_jupyter()      # 打印差分图像
43
44 print(end - start)
45 time.sleep(0.1)
```

14.1.4 结果展示

结果如图所示：



图 14.1：直线检测

测试方法：执行程序后保持相机和背景不动，在摄像头前摇摆手可以看到检测到的手的边缘。

第三部分

典型应用案例篇 - Demo

第十五章 Demo1 - 人脸检测

内容提要

人脸检测

猫脸检测

本章实验需要的配件：

- Powersensor 传感器
- 人脸检测测试图（有女孩和猫的那一张）



图 15.1：人脸检测测试图

15.1 人脸检测

15.1.1 介绍

本章节使用 Haar 实现人脸检测，首先训练分类器查找人的正脸，侧脸，人体轮廓，包括猫脸等，当然如果你想自己训练各种识别目标的模型并生成 xml 文件，之后用于检测也是可以的。如果更换了 xml 文件，只需要将训练文件上传到目录，之后更改程序中 classifier 的名称即可。本示例中我已经将人脸、猫脸、人眼检测的文件放到了当前目录下，直接更改相应名称即可使用。如果您想用别人训练好的文件，可以百度 google 搜索一下对应的，基本的人脸训练 xml 文件可以在我给出的博客链接中下载。

Harr 模板参考：<https://github.com/opencv/opencv/tree/master/data/haarcascades>

15.1.2 参考例程

```
1 classifier=cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")    #确保  
2     此xml文件与该py文件在一个文件夹下，否则将这里改为绝对路径，此xml文件可在D  
3     :\My Documents\Downloads\opencv\sources\data\haarcascades下找到。  
4  
5     for i in range(10):  
6         start = time.time()          #记录开始时间  
7         clear_output(wait=True)      #清除图片，在同一位置显示，不使用会打印多张图  
8         片
```

```

6     imgMat = cam1.read_img_ori()    # 读入图像
7
8     tempImg = cv2.resize(imgMat, (320,240))    # 缩小图像为320x240尺寸
9
10    image=cv2.cvtColor(tempImg,cv2.COLOR_BGR2GRAY)
11    cv2.equalizeHist(image)
12    divisor=8
13    h=320
14    w=240
15    minSize=(w/divisor,h/divisor)
16    faceRects=classifier.detectMultiScale(image,1.2,2,cv2.CASCADE_SCALE_IMAGE
17 ,minSize)
18
19    if len(faceRects)>0:
20        for faceRect in faceRects:
21            x,y,w,h=faceRect
22            cv2.circle(tempImg,(x+w/2,y+h/2),min(w/2,h/2),(255,0,0))
23            #           cv2.circle(frame,(x+w/4,y+h/4),min(w/8,h/8),(255,0,0))
24            #           cv2.circle(frame,(x+3*w/4,y+h/4),min(w/8,h/8),(255,0,0))
25            #           cv2.rectangle(frame,(x+3*w/8,y+3*h/4),(x+5*w/8,y+7*h/8)
26 , (255,0,0))
27
28    end = time.time()
29    # 把图像拼接在一起显示
30    ps.CommonFunction.show_img_jupyter(tempImg) # 打印用于差分的两张图片
31    print(end - start)
32    time.sleep(0.1)

```

15.1.3 结果展示

结果如图所示：



图 15.2: 二维码识别结果

从图中可以看出人脸被检测（圈）出来了。

15.2 猫脸检测

15.2.1 介绍

与人脸检测方法类似，用户可以自行操作。

第十六章 Demo2 - 彩色小球检测

内容提要

- 模糊滤波
- 霍夫圆检测
- 颜色识别

本章实验需要的配件：

- Powersensor 传感器
- 各种颜色小球

16.1 原理讲解

1. 霍夫圆变换：

基本原理和上个教程中提到的霍夫线变换类似，只是点对应的二维极径极角空间被三维的圆心点 x, y 还有半径 r 空间取代。原理：从平面坐标圆上的点到极坐标转换的三个参数 $C(x_0, y_0, r)$ 其中 x_0, y_0 是圆心， r 取一固定值时扫描 360 度， x, y 跟着变化，若多个边缘点对应的三维空间曲线交于一点，则他们在共同圆上，在圆心处有累积最大值，也可以用同样的阈值的方法来判断一个圆是否被检测到。

2. 相关 API - HoughCircles

因为霍夫圆检测对噪声比较敏感，所以首先要对图像做滤波（比如椒盐噪声用中值滤波，其他的也可以用高斯模糊）。基于效率考虑，Opencv 中实现的霍夫变换圆检测是基于图像梯度（霍夫梯度法，也叫 2-1 霍夫变换 (2HT)）的实现，分为两步（已封装到 HoughCircles）：(1) Canny 检测边缘，发现可能的圆心。圆心一定是在圆上的每个点的模向量上，这些圆上点模向量的交点就是圆心，霍夫梯度法的第一步就是找到这些圆心，这样三维的累加平面就又转化为二维累加平面。(2) 基于第一步的基础上从候选圆心开始计算最佳半径大小。第二步根据所有候选中心的边缘非 0 像素对其的支持程度来确定半径。

16.2 参考例程

```
1 # 小球颜色识别bgr
2 def colour2(img_b,img_g,img_r) :
3     if ((img_b>=60 and img_b<=130 ) and (img_g>=60 and img_g<=130) and (
4         img_r>=140 and img_r<=220) ):
5         cv2.putText(origin,'red', (i[0]-i[2],i[1]-i[2]), cv2.FONT_HERSHEY_PLAIN
, 2.0, (0, 0, 255), 2)
6     elif ((img_b>=70 and img_b<=140) and (img_g>=120 and img_g<=190) and (
7         img_r>=170 and img_r<=240) ):
```

```

6     cv2.putText(origin,'yellow',(i[0]-i[2],i[1]-i[2]), cv2.
7 FONT_HERSHEY_PLAIN, 2.0, (0, 0, 255), 2)
8 elif ((img_b>=70 and img_b<=140) and (img_g>=150 and img_g<=230) and (
9 img_r>=100 and img_r<=170) ):
10    cv2.putText(origin,'green', (i[0]-i[2],i[1]-i[2]), cv2.
11 FONT_HERSHEY_PLAIN, 2.0, (0, 0, 255), 2)
12 elif ((img_b>=50 and img_b<=120) and (img_g>=60 and img_g<=120) and (
13 img_r>=70 and img_r<=120) ):
14    cv2.putText(origin,'brown', (i[0]-i[2],i[1]-i[2]), cv2.
15 FONT_HERSHEY_PLAIN, 2.0, (0, 0, 255), 2)

16
17 while(True):
18     clear_output(wait=True)
19     imgMat = cam1.read_img_ori()
20     origin = cv2.resize(imgMat, (320,240))
21     start = time.time()
22     # 转换为灰度图
23     img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
24     # medianBlur 平滑（模糊）处理
25     img_gray = cv2.medianBlur(img_gray, 7)
26     # 圆检测
27     circles = cv2.HoughCircles(img_gray, cv2.HOUGH_GRADIENT, 1, 40, param1
28 =50,param2=35, minRadius=0, maxRadius= 300)
29     if circles is None:
30         pass
31     else:
32         circles = np.uint16(np.around(circles))
33         for i in circles[0,:]:
34             # 勾画正方形，origin图像、i[2]*2是边长
35             cv2.rectangle(origin,(i[0]-i[2],i[1]-i[2]),(i[0]+i[2],i[1]+i[2])
36             ,(255,0,0), 2)
37             # 取球心一小块区域
38             roi = origin[i[0]:i[1] , i[0]:i[1]+1]
39             # 分离bgr通道
40             img_b = np.uint16(np.mean(roi[:, :, 0]))
41             img_g = np.uint16(np.mean(roi[:, :, 1]))
42             img_r = np.uint16(np.mean(roi[:, :, 2]))
43             # 判断小球颜色
44             colour2(img_b,img_g,img_r)

# 计算消耗时间
end = time.time()
ps.CommonFunction.show_img_jupyter(origin)
print(end - start)
#time.sleep(0.1)

```

16.3 结果展示

结果如图所示：

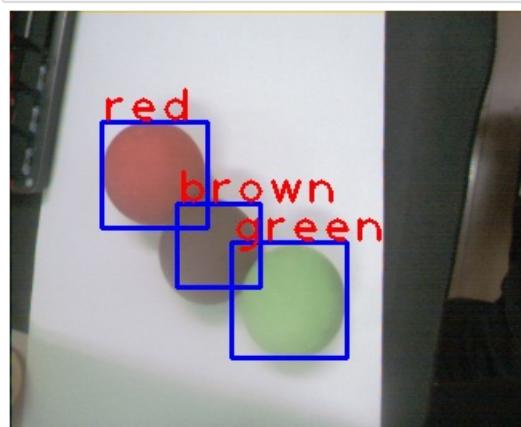


图 16.1：小球识别结果

框出小球并识别到三个小球颜色

16.4 小结一下

本 Demo 还提供了 offline 的版本，结果通过串口输出。程序及视频见 Powersensor 的 Github：

<https://github.com/powersensor-cn/PowersensorDemo/tree/master/demo3-ballDetect>

结论 本文主要介绍了基于霍夫圆检测方法以及如何框出小球并识别颜色，霍夫变换的原理就是利用图像全局特征将边缘像素连接起来组成区域封闭边界，它将图像空间转换到参数空间，在参数空间对点进行描述，达到检测图像边缘的目的。识别小球颜色主要难点在调整 RGB 参数方面，利用 TakeColor 取色器选取参数范围，设置正确后便可以准确判断出小球颜色。

第十七章 Demo3 - 多边形检测

内容提要

❑ 模糊滤波

❑ 多边形拟合

本章实验需要的配件：

- Powersensor 传感器
- 多边形图例

17.1 原理讲解

使用 opencv 中多边形拟合函数，用多边形拟合连通域的轮廓。通过检测拟合后轮廓的角度数量判断多边形形状。

17.2 参考例程

```
1  for i in range(250):
2      clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
3
4      imgMat = cam1.read_img_ori()      # 读入图像
5
6      # 缩小图像为320x240尺寸
7      origin = cv2.resize(imgMat, (320,240))
8
9      # ----- 图像处理开始
10
11     start = time.time()      # 记录开始时间
12
13     # 把图片转换为灰度图
14     img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
15     # 中值滤波
16     img_mid = cv2.medianBlur(img_gray, 3)
17     # 二值化
18     ret,img_binary = cv2.threshold(img_mid, 125, 255, cv2.THRESH_BINARY_INV) # 固定值二值化，二值化阈值125
19     #img_binary = cv2.adaptiveThreshold(img_mid, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 3, 2) #自适应二值化
20     #搜索连通域
21     thresh,contours,hierachy = cv2.findContours(img_binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

```

22     for cnt in range(len(contours)):
23
24         #拟合度参数设置，系数可修改
25         epsilon = 0.05*cv2.arcLength(contours[cnt],True)
26         #多边形拟合
27         approx = cv2.approxPolyDP(contours[cnt],epsilon,True)
28
29         #角点检测
30         corners = len(approx)
31         #计算面积，用于滤去面积过大或过小的连通域
32         area = cv2.contourArea(contours[cnt])
33
34
35
36         if area > 30: #面积范围可随环境变化修改
37
38             #利用图形矩计算连通域质心
39             mm = cv2.moments(contours[cnt])
40             cx = int(mm['m10'] / mm['m00'])
41             cy = int(mm['m01'] / mm['m00'])
42
43             #绘制质心
44             #cv2.circle(img_gray, (cx, cy), 3, 0, -1)
45
46             #检测角点
47             if corners == 3:
48                 cv2.drawContours(img_gray,contours[cnt],-1,0,3)
49                 cv2.putText(img_gray, "triangle ", (cx-30, cy), cv2.
50 FONT_HERSHEY_PLAIN, 1.2, 255, 1)
51             elif corners == 4:
52                 cv2.drawContours(img_gray,contours[cnt],-1,0,3)
53                 cv2.putText(img_gray, "square ", (cx-30, cy), cv2.
54 FONT_HERSHEY_PLAIN, 1.2, 255, 1)
55             elif corners == 5:
56                 cv2.drawContours(img_gray,contours[cnt],-1,0,3)
57                 cv2.putText(img_gray, "pentagon ", (cx-30, cy), cv2.
58 FONT_HERSHEY_PLAIN, 1.2, 255, 1)
59             elif corners == 6:
60                 cv2.drawContours(img_gray,contours[cnt],-1,0,3)
61                 cv2.putText(img_gray, "hexagon ", (cx-30, cy), cv2.
62 FONT_HERSHEY_PLAIN, 1.2, 255, 1)
63             # ----- 图像处理结束
64
65             #计算消耗时间
66             end = time.time()
67             img_combine = np.hstack([img_gray,img_binary])
68             ps.CommonFunction.show_img_jupyter(img_combine)

```

```
64     print(end - start)
```

17.3 结果展示

结果如图所示：



图 17.1：三角形识别结果

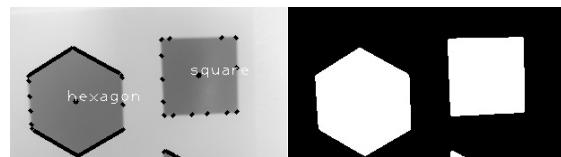


图 17.2：多边形识别结果

框出图形并判断到图形的形状

第十八章 Demo4 - 相机标定

内容提要

相机成像原理

标定原理

本章实验需要的配件:

- Powersensor 传感器
- 标定板

注 标定板需要自行购买, 一般是棋盘格:

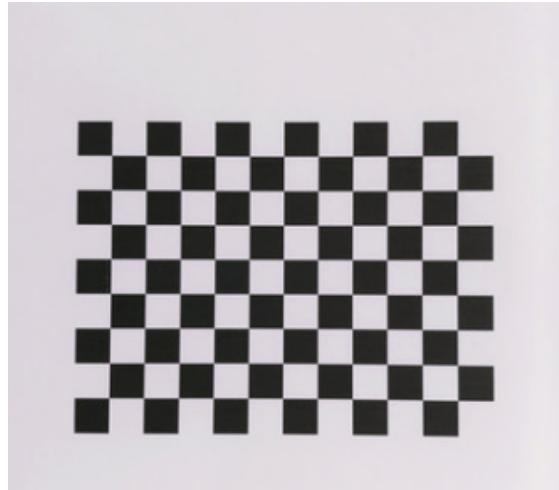


图 18.1: 标定板

18.1 原理讲解

18.1.1 标定目的

1. 建立摄像机成像模型:

标定相机, 可以求解相机的内参和外参, 获取摄像头在三维世界中的位置、姿态, 以及焦距等物理信息。

2. 纠正畸变

透镜的制造工艺和物理结构, 决定了透镜会带来一定程度的成像畸变问题。通过标定相机, 获取摄像头的畸变参数并进行相应矫正, 可以使得成像后的图像与现实世界的景象保持一致。

18.1.2 坐标系定义

首先先定义四个坐标系。

1. 世界坐标系 (World reference frame): 现实世界的三维坐标系, 标记为 w , 单位: m

2. 相机坐标系 (Camera reference frame): 以相机光心为原点建立的坐标系, 标记为 c , 单位: m
3. 图像坐标系 (Image reference frame): 以成像平面中心为原点建立的坐标系, 标记为 i , 单位: m
4. 像素坐标系 (Pixel reference frame): 以图像左上角第一个像素为原点建立的像素级坐标系, 标记为 c , 单位: 像素

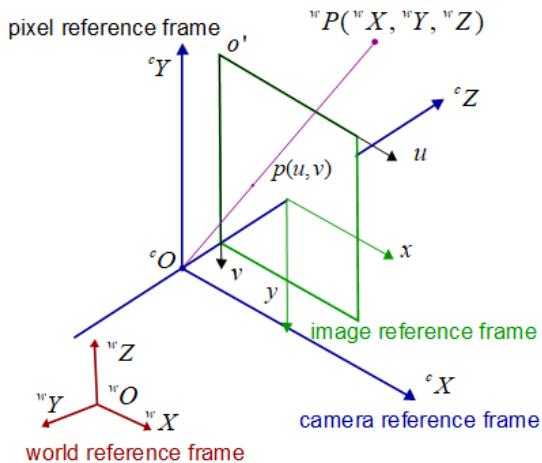


图 18.2: 标坐标系定义

将刚体的坐标从世界坐标系转换到相机坐标系, 可以通过平移和旋转的方式, 其变换矩阵则是由平移矩阵和旋转矩阵组合而成的其次坐标矩阵。

$${}^c X = \begin{bmatrix} {}^c x \\ {}^c y \\ {}^c z \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ O_{3 \times 1} & 1 \end{bmatrix} \begin{bmatrix} {}^w x \\ {}^w y \\ {}^w z \\ 1 \end{bmatrix} = R_{w>c} {}^w X$$

其中

$$R_{w>c} = \begin{bmatrix} R & T \\ O & 1 \end{bmatrix}$$

即相机的外参矩阵。

18.1.3 小孔模型

世界坐标系中的景象到图像坐标系的仿射变换, 可以近似用小孔模型拟合。

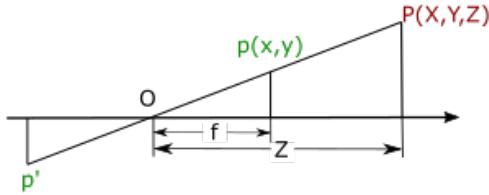


图 18.3: 小孔成像模型

由图可得世界坐标系和图像坐标系的变换关系为

$${}^i x = {}^w x f / {}^w z, {}^i y = {}^w y f / {}^w z$$

写成其次坐标的形式则为

$$\begin{bmatrix} {}^i x \\ {}^i y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^w x \\ {}^w y \\ {}^w z \end{bmatrix} = R_{c>i} \begin{bmatrix} {}^w x \\ {}^w y \\ {}^w z \end{bmatrix}$$

18.1.4 像素坐标系的描述

由于像素坐标系的单位尺度、原点与图像坐标系皆不重合，定义像素坐标系的原点在图像坐标系下的坐标为 (u_0, v_0) ，每个像素的长度为 d_x, d_y ，则像素坐标与图像坐标的关系为

$$u = u_0 + {}^i x / d_x, v = v_0 + {}^i y / d_y$$

即

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1/d_x & 0 & u_0 \\ 0 & 1/d_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^i x \\ {}^i y \\ {}^i z \end{bmatrix} = R_{i>p} \begin{bmatrix} {}^i x \\ {}^i y \\ {}^i z \end{bmatrix}$$

所以，如果不考虑相机畸变，从世界坐标系到像素坐标系的变换矩阵为

$${}^p X = R_{i>p} R_{c>i} R_{w>c} {}^w X = R_{w>p} {}^w X$$

18.1.5 畸变

在实际中，由于制作工艺的原因，经过透镜的图像会产生一定的形变，形成镜像畸变和切向畸变。使用 OpenCV 的标定技术，则可以矫正畸变，以获得更准确的信息。通常图像畸变都通过以下方式矫正。径向畸变：

$$x_{correct} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), y_{correct} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

切向畸变：

$$x_{correct} = x + [2p_1xy + p_2(r^2 + 2x^2)], y_{correct} = y + [2p_1xy + p_2(r^2 + 2y^2)]$$

对参数 k_1, k_2, k_3, p_1, p_2 求解，则可得到图像的矫正方程。

18.2 标定原理

使用已知的棋盘格，利用棋盘格角点对相机进行标定。在不考虑畸变的情况下，取棋盘格上的 4 个角点，构建 $8k$ 个约束方程，用以求解相机 $4+6k$ 个参数，当 $8k > 4+6k$ ，即 $k \geq 2$ 时，可求解无畸变情况下相机的内参和外参。

在实际情况下，一般会采用至少 10 张 7×8 或更大的棋盘图像，采用最小二乘的方法求得最优解。在标定过程中，只在移动棋盘在不同图像中足够大以从视场图像中得到更加丰富的信息。

18.3 参考例程

18.3.1 获取棋盘格图像

程序功能：获取 20 张棋盘格图像，并保存在路径“./Images/”下，命名为 img01.jpg, img02.jpg ……标定一般选取 10 张以上图片即可，数量可视情况自行选取。

```

1 #帧数计数器
2 frameCnt = 0
3
4 frameNum = 200;
5 for i in range(frameNum):
6     clear_output(wait=True)      #清除图片，在同一位置显示，不使用会打印多张图片
7     start = time.time()
8
9     imgMat = cam1.read_img_ori()
10    #缩放图像
11    tempImg = cv2.resize(imgMat, (960, 720))
12    displayMat = cv2.resize(imgMat, (320, 240))
13    #提取灰度图像
14    grayMat = cv2.cvtColor(imgMat, cv2.COLOR_BGR2GRAY)
15    #固定值二值化
16    ret,img_binary = cv2.threshold(grayMat, 100, 255, cv2.THRESH_BINARY_INV)
17
18
19    if frameCnt < 10:
20        path = "./Images/img0"+ str(frameCnt)+".jpg"
21    else:
22        path = "./Images/img"+ str(frameCnt)+".jpg"
23

```

```

24 # 存储图片间隔：每隔十帧
25 if i%10 == 9:
26
27     # 计数器自增
28     frameCnt += 1
29
30     # 存储图像
31     cv2.imwrite(path,grayMat, [int(cv2.IMWRITE_JPEG_QUALITY), 100])
32     print("存储第"+str(frameCnt)+"张图片")
33
34 ps.CommonFunction.show_img_jupyter(displayMat)
35
36 end = time.time()
37 time.sleep(0.1)
38

```

18.3.2 标定

程序功能：读取已获取的 20 张棋盘格图像，标定并获取相机参数。

首先需要根据自己选取的标定板设置自定义参数。

```

1 # 定义标定板参数，本次实验采用12x9个棋盘格的标定板，每个方格边长20mm
2 # 标定板列数
3 tagCol = 12-1
4 # 标定板行数
5 tagRow = 9-1
6 # 标定板单元格尺寸
7 tagLen = 20
8

```

标定例程如下

```

1 # 设置寻找亚像素角点的参数，采用的停止准则 是最大循环次数30 和最大误差容限0
2 .001
3 criteria = (cv2.TERM_CRITERIA_MAX_ITER | cv2.TERM_CRITERIA_EPS, 30, 0.001)
4
5 # 定义标定板参数，本次实验采用12x9个棋盘格的标定板，每个方格边长20mm
6 tagCol = 12-1
7 tagRow = 9-1
8 tagLen = 20
9
10 # 获取标定板角点的位置
11 objp = np.zeros((tagCol*tagRow,3), np.float32)
12 # 将世界坐标系建在标定板上，所有点的z坐标全部为0，所以只需要赋值x和y
13 objp[:, :2] = np.mgrid[0:tagCol, 0:tagRow].T.reshape(-1, 2)
14
15 obj_points = []      # 存储3D点
16 img_points = []      # 存储2D点

```

```

16 # 获取图像路径
17 imgnum = glob.glob('./Images/*.jpg')
18 frameCnt = 0
19 print("加载图片中...")
20
21 for fname in imgnum:
22     imgMat = cv2.imread(fname)
23
24     frameCnt = len(img_points)+1
25     displayStr = "加载第"+ str(frameCnt)+"张图"
26     print (displayStr)
27
28     # 检测棋盘格角点
29     grayMat = cv2.cvtColor(imgMat, cv2.COLOR_BGR2GRAY)
30     ret, corners = cv2.findChessboardCorners(grayMat, (tagCol,tagRow), None)
31
32     if ret:
33         obj_points.append(objp*tagLen)
34         # 在原角点的基础上寻找亚像素角点
35         corners2 = cv2.cornerSubPix(grayMat, corners, (5,5), (-1,-1), criteria)
36
37     if corners2.all() == None:
38         img_points.append(corners)
39     else:
40         img_points.append(corners2)
41
42     # 绘制棋盘角点
43     cv2.drawChessboardCorners(imgMat, (tagCol,tagRow), corners,ret)
44     #ps.CommonFunction.show_img_jupyter(imgMat)
45     else:
46         print(fname,"not found")
47         time.sleep(0.1)
48
49     print("相机标定中...")
50     ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(obj_points, img_points
51     ,(960,720), None, None)
52     print("标定完成! ")
53
54     print("ret",ret)
55     print("mtx:",mtx)
56     print("dist:",dist)
57     print("rvecs:",rvecs)
58     print("tvecs:",tvecs)

```

18.4 结果展示

本次使用的标定图片如下图所示：



图 18.4: 标定使用图片

最终得到的参数矩阵为：

```
('ret': 0.12179968304423555)
('mtx': array([[573.40849765, 0.           , 398.29907511],
               [0.           , 573.6509063 , 234.51605829],
               [0.           , 0.           , 1.           ]]))
('dist': [[-0.0165949054, 0.448949565, -0.00095217672, 0.0010348
9317, -1.70621062]])
```

图 18.5: 标定结果

第十九章 Demo5 - 二维码识别

内容提要

二维码

qrCode

本章实验需要的配件：

- Powersensor 传感器
- 二维码测试图片（有二维码的那一张）

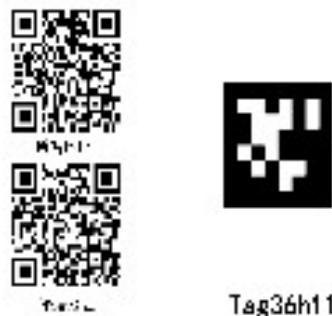


图 19.1: 二维码测试图

19.1 普通二维码识别

19.1.1 介绍

二维码是日常生活中非常常见的，二维码识别是一种很有用、很高效的通信和认证的手段。

本教程主要使用 zbar 库识别通用的二维码。很多网站都有提供二维码生成功能，本教程实验的二维码是在<https://cli.im/>上生成的。

进行二维码识别主要需要以下步骤：

1. 把图片转换成灰度图
2. 对图片进行滤波和二值化
3. 识别二维码
4. 输出结果

19.1.2 参考例程

1. 使用 zbar 库识别二维码非常容易，首先 import 以前那些常用的库、初始化摄像头对象后，还需要 import zbar 的库。

```
1 # 引用二维码库
2 import zbar
3 from pyzbar import pyzbar
```

2. 调用 zbar 库识别二维码

```

1  for i in range(100):

2

3      clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印
4      多张图片
5
6      imgMat = cam1.read_img_ori()      # 读入图像
7
8
9      # ----- 图像处理开始
10
11
12      start = time.time()          # 记录开始时间
13
14
15      # 把图片转换为灰度图
16      img_gray = cv2.cvtColor(origin, cv2.COLOR_BGR2GRAY)
17      # 中值滤波
18      img_mid = cv2.medianBlur(img_gray, 3)
19      # 二值化
20      ret,img_binary = cv2.threshold(img_mid, 120, 255, cv2.THRESH_BINARY)
21      # 全局二值化
22
23      # 识别二维码
24      result = pyzbar.decode(img_binary)
25      # 把结果标在图片上
26      if len(result) == 0:
27          pass
28      else:
29          font=cv2.FONT_HERSHEY_SIMPLEX
30          text_x = result[0][2][0]
31          text_y = result[0][2][1]
32          text_wid = result[0][2][2]
33          cv2.putText(img_gray, result[0][0], (text_x - text_wid / 2 , text_y),
34          , font, 0.5, 0, 1)

35
36      end = time.time()          # 记录结束时间
37      # ----- 图像处理结束
38
39
40      # 把图像拼接在一起显示
41      img_combine = np.hstack([img_gray, img_mid, img_binary])
42      ps.CommonFunction.show_img_jupyter(img_combine) # 打印用于差分的两张
43      图片
44
45      print(result)
46      print(end - start)
47
48      time.sleep(0.1)

```

39

19.1.3 结果展示

结果如图所示：



图 19.2: 二维码识别结果

从左到右分别是原图，滤波后的图像，二值化后的图像。下面是识别的结果，包含内含的网址、尺寸、座标等。

19.2 Apriltag 识别

19.2.1 介绍

Apriltag 是一类特殊的二维码，普通的二维码读取工具并不能读取它（如手机扫一扫）。

Apriltag 标记是并不是设计来传递信息的，它是设计来给机器人定位的，所以它包含的内容很少，但是又大量的校验，形状也较为简单，很容易识别。

Apriltag 标记的识别结果如下所示：

```

1 Family: tag36h11
2 ID: 0
3 Hamming error: 0
4 Goodness: 0.0
5 Decision margin: 124.666664124
6 Homography: [[ 7.02583484e-01 -2.35382052e-01  4.84189739e+00]
7   [-6.82695217e-02  4.98301018e-01  2.90763050e+00]
8   [-2.24977475e-04 -1.27799591e-03  2.34032006e-02]]
9 Center: [206.89039396 124.24072034]
10 Corners: [[175.64704895  99.47730255]
11   [236.33509827  95.72452545]
12   [242.42208862  152.40307617]
13   [174.67114258  155.44396973]]
```

其中 Family 是标记的种类；Homography 是单应矩阵，表征的是从 Apriltag 标记的平面到相机平面的转换映射；Center 是 Apriltag 标记在画面里的中心位置；Corners 是这个 Apriltag 标记四个角的位置。

注 Corners 是有顺序的，也就是可以通过这个来确定 Apriltag 自转的角度。

19.2.2 参考例程

- 和普通的二维码识别一样，需要 import Apriltag 专用的库。

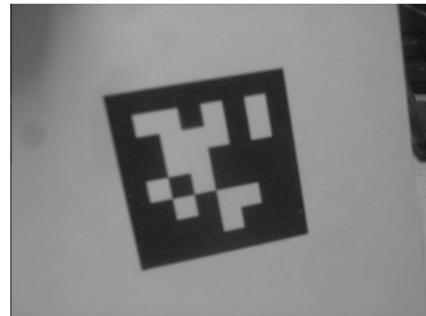
```
1 import apriltag
2
```

- 调用 apriltag 库识别 apriltag 标记

```
1 detector = apriltag.Detector()
2 for i in range(200):
3     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印
4         多张图片
5     imgMat = cam1.read_img_ori()      # 读入图像
6
7     # ----- 图像处理开始 -----
8     start = time.time()          # 记录开始时间
9
10    # 把图片转换为灰度图
11    img_gray = cv2.cvtColor(imgMat, cv2.COLOR_BGR2GRAY)
12    # 中值滤波
13    img_mid = cv2.medianBlur(img_gray, 3)
14    # 二值化
15    ret,img_binary = cv2.threshold(img_mid, 90, 255, cv2.THRESH_BINARY)
16    # 全局二值化
17    #img_binary = cv2.adaptiveThreshold(img_mid,255,1,1,11,20) #自适应二
18    值化
19    # 识别二维码
20    detections = detector.detect(img_binary)
21
22    #二维码识别和测量结果
23    for det in detections:
24        #计算二维码位姿信息,tag_size是apriltag的物理尺寸，边长4厘米
25        pose_mtx, init_error, final_error = detector.detection_pose(det,
26        camera_para, tag_size=4)
27        x = pose_mtx[0][3]
28        y = pose_mtx[1][3]
29        z = pose_mtx[2][3]
30
31        end = time.time()          # 记录结束时间
32        # ----- 图像处理结束 -----
33
34        # 显示图片
35        ps.CommonFunction.show_img_jupyter(img_binary) # 打印用于差分的两张图
36        片
37        # 打印计算时间
38        print(end - start, x,y,z)
39
40        time.sleep(0.1)
```

19.2.3 结果展示

结果如图所示：



[154.44482476 127.03884661]

Tim:17

图 19.3: AprilTag 识别结果

从上到下分别是原图、中心座标、消耗时间。

第二十章 Demo6 - 二维码位姿测量

内容提要

- 二维码
- 单应性矩阵
- qrCode
- 位姿检测

本章实验需要的配件：

- Powersensor 传感器
- 二维码测试图片（有二维码的那一张）

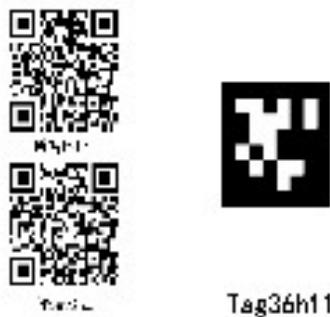


图 20.1: 二维码测试图

20.1 检测原理

使用相机，对 AprilTag 的位置和姿态进行估计。由于二维码的图形特征和尺寸是给定的，根据相机成像原理（Demo4 中已详细解释）多点匹配原理，将图像中的点映射到世界坐标系下，从而获得二维码单应性矩阵，实现二维码的位置和姿态求解。

AprilTag 检测和定位常常用在机器人（多机器人）系统中对特定机器人的识别和相对位姿测量。

20.1.1 参考例程

1. 根据标定的结果载入相机参数

```
1 # 载入相机参数
2 camera_para_mtx = [[573.40849765,    0.          , 398.29907511],
3 [  0.          , 573.6509063 , 234.51605829],
4 [  0.          ,    0.          ,   1.          ]]
5 camera_para = (573.40849765,573.6509063,398.29907511,234.51605829)
6
```

2. 测量识别和测量程序如下：

```
1 import apriltag
```

```
2 detector = apriltag.Detector()
3 for i in range(200):
4     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印
5     多张图片
6
7     imgMat = cam1.read_img_ori()           # 读入图像
8
9
10    # ----- 图像处理开始 -----
11    start = time.time()                  # 记录开始时间
12
13    # 把图片转换为灰度图
14    img_gray = cv2.cvtColor(imgMat, cv2.COLOR_BGR2GRAY)
15    # 中值滤波
16    img_mid = cv2.medianBlur(img_gray, 3)
17    # 二值化
18    ret,img_binary = cv2.threshold(img_mid, 90, 255, cv2.THRESH_BINARY)
19    # 全局二值化
20    #img_binary = cv2.adaptiveThreshold(img_mid,255,1,1,11,20) #自适应二
21    值化
22
23    # 识别二维码
24    detections = detector.detect(img_binary)
25
26
27    #二维码识别和测量结果
28    for det in detections:
29        #计算二维码位姿信息,tag_size是apriltag的物理尺寸，边长4厘米
30        pose_mtx, init_error, final_error = detector.detection_pose(det,
31        camera_para, tag_size=4)
32
33        x = pose_mtx[0][3]
34        y = pose_mtx[1][3]
35        z = pose_mtx[2][3]
36
37
38        end = time.time()                  # 记录结束时间
39        # ----- 图像处理结束 -----
40
41
42        # 显示图片
43        ps.CommonFunction.show_img_jupyter(img_binary) # 打印用于差分的两张图
44        片
45
46        # 打印计算时间
47        print(end - start, x,y,z)
48
49
50        time.sleep(0.1)
```

20.1.2 结果展示

结果如图所示：

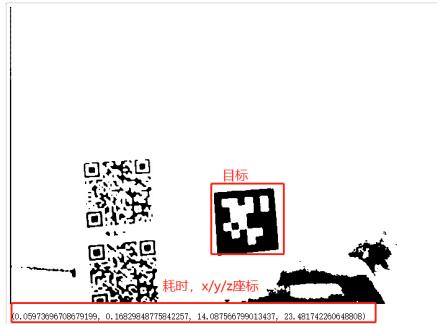


图 20.2: 测试结果

第二十一章 Demo 7 - kcf 目标跟踪

内容提要

□ kcf

□ 目标跟踪

本章实验需要的配件：

- Powersensor 传感器

21.1 检测原理

视觉目标跟踪是计算机视觉中的一个重要研究方向，有着广泛的应用，如：视频监控，人机交互，无人驾驶等。kcf（Kernelized Correlation Filters）是一种基于相关滤波的跟踪算法，大致原理是：

1. 初始化：在指定的位置附近采样（生成大量窗口），训练一个回归器，这个回归器能计算一个小窗口里目标中心的偏离程度。
2. 更新目标中心：在第 I 帧图像中的原目标中心（第 I-1 帧求得的）附近采样（生成大量窗口），求解使上一步回归器偏移度最小的坐标，以此坐标为新的目标中心。
3. 更新回归器：在 I 帧图像中的新的目标中心附近采样（生成大量窗口），修正训练回归器。

从上述原理可以出看，kcf 是一种递归的目标跟踪求解算法，要求解算的两帧之间的目标图像有一定的重合度，算法的跟踪性能与帧率、以及目标运动的快慢关系巨大。采样过程如下图所示：

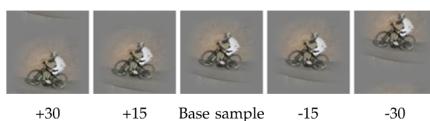


图 21.1: kcf 采样的采样窗口

上述只是基本原理，具体的 kcf 算法过程复杂很多，采用了大量如样本循环移位代替采样窗口、Hog 特征、核函数映射等技术来提升性能和加快计算效率。



笔记 Kcf 的论文下载地址：<https://ieeexplore.ieee.org/abstract/document/6870486>

21.1.1 参考例程

1. 包含库和实例化相机对象

```
1 import cv2
2 import numpy as np
3 import cv2
4 import matplotlib.pyplot as pyplot
5 from IPython.display import clear_output
```

```

6 import time
7 import PowerSensor as ps
8
9 cam1 = ps.ImageSensor()
10

```

2. 跟踪算法调用：在启动时把要跟踪的目标静止地放到中间的红色方框里，等方框变成蓝色的时候代表跟踪算法开始启动：

```

1 state = 0
2 start_cnt = 0
3 gROI = (130, 80, 60, 80)
4 gROI_raw = (260, 160, 120, 160)
5 flag_img = []
6 font=cv2.FONT_HERSHEY_SIMPLEX
7 tracker = cv2.TrackerKCF_create()
8 for i in range(100):
9     start = time.time()
10    if state == 0:
11        if start_cnt > 30:
12            state = 1
13            flag_img = imgMat
14            tracker.init(flag_img, gROI_raw)
15        else:
16            start_cnt = start_cnt + 1
17
18    clear_output(wait=True)
19    imgMat = cam1.read_img_ori()
20    tempImg = cv2.resize(imgMat, (320,240))
21    if state == 0:
22        x, y, w, h = gROI
23        res_img = tempImg.copy()
24        cv2.rectangle(res_img, pt1=(x, y), pt2=(x + w, y + h), color=(0,
0, 255), thickness=2)
25    else:
26        # 处理图像用的 640*480
27        status, coord = tracker.update(imgMat)
28        if status:
29            # 考虑到处理图像的像素和显示用的像素不一样，这里画框的时候要除2
30            p1 = (int(coord[0] / 2), int(coord[1]) / 2)
31            p2 = (int(coord[0] / 2 + coord[2] / 2), int(coord[1] / 2 + coord
[3] / 2))
32            res_img = tempImg
33            cv2.rectangle(res_img, p1, p2, (255, 0, 0), 2, 1)
34        else:
35            tracker.init(flag_img, gROI_raw)
36            res_img = tempImg
37            cv2.putText(res_img,'False',(20,20), font, 1,(255,255,255),2)

```

```
38 ps.CommonFunction.show_img_jupyter(res_img)
39 end = time.time()
40 pr = end - start
41 #     print(end - start)
42 # if the period is too short, needs to increase for display
43 if pr < 0.1:
44     time.sleep(0.1 - pr)
45
```

21.1.2 结果展示

结果如图所示：

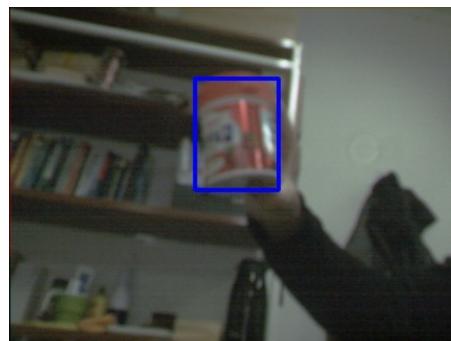


图 21.2: 测试结果

第二十二章 Demo 8 - Dlib 人眼检测

内容提要

dlib

人眼检测

本章实验需要的配件:

- Powersensor 传感器

22.1 检测原理

Dlib 是一个包含机器学习算法的 C++ 开源工具包。Dlib 可以帮助您创建很多复杂的机器学习方面的软件来帮助解决实际问题。目前 Dlib 已经被广泛的用在行业和学术领域，包括机器人，嵌入式设备，移动电话和大型高性能计算环境。

Dlib 是开源的、免费的；官网和 git 地址：

注意 22.1

- 官网

<http://dlib.net/>

- github

<https://github.com/davisking/dlib>



dlib 库有一个用于人脸 68 个关键点检测的 dat 模型库，[shape_predictor_68_face_landmarks.dat](#)，使用这个模型库可以很方便地进行人脸检测，并进行简单的应用。相比于 opencv 的人脸检测，这个库不但可以圈出人脸，还可以提取出人脸的特征：

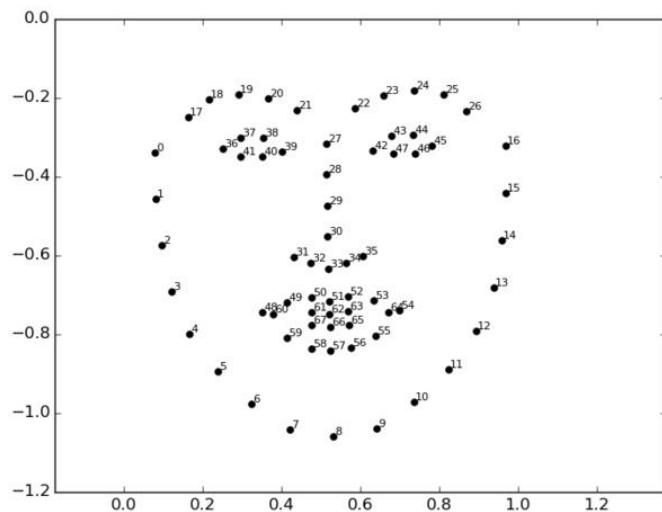


图 22.1: dlib 的人脸特征模型

本章教程主要结合 opencv 库和 dlib 库，利用提取到的人眼特征分析人的疲劳程度。原理如图：

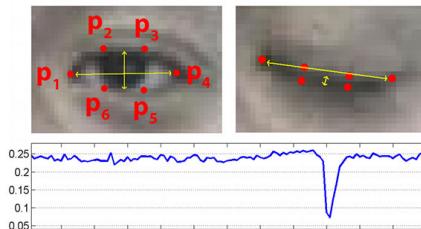


图 22.2: 疲劳度检测原理

计算公式如下：

$$EAR = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|} \quad (22.1)$$

22.1.1 参考例程

1. 包含库

```

1 import dlib
2 import cv2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from IPython.display import clear_output
6 import time
7 import PowerSensor as ps
8 import imutils
9 from imutils import face_utils
10

```

2. 初始化重要参数，加载人脸模型库

```

1 thresh = 0.25
2 frame_check = 6
3 detect = dlib.get_frontal_face_detector()
4 predict = dlib.shape_predictor("/home/debian/
    shape_predictor_68_face_landmarks.dat")
5 (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
6 (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
7

```

3. 定义欧拉距离函数、眼角比函数和检测函数

```

1 # 求两点间的欧拉距离函数
2 def dis_eucl(p1, p2):
3     return np.sqrt(np.square(p1[0] - p2[0]) + np.square(p1[1] - p2[1]))
4
5 # 眼角比检测函数（就是上面的EAR公式对应的函数）
6 def eye_aspect_ratio(eye):
7     A = dis_eucl(eye[1], eye[5])
8     B = dis_eucl(eye[2], eye[4])
9     C = dis_eucl(eye[0], eye[3])

```

```

10     ear = (A + B) / (2.0 * C)
11     return ear
12
13 # 检测函数，第一个参数是待检测的图片，第二个参数是否画出人眼
14 def check_eye(img, draw_res=False):
15     # 首先把彩图转成灰度图
16     img_temp = img.copy()
17     gray = cv2.cvtColor(img_temp, cv2.COLOR_BGR2GRAY)
18     # 检测灰度图中的人脸信息
19     subjects = detect(gray, 0)
20     # 逐个人脸检测人眼
21     for subject in subjects:
22         # 在人脸上套人脸模型
23         shape = predict(gray, subject)
24         # 转换成numpy，便于处理
25         shape = face_utils.shape_to_np(shape)
26         # 获取两眼的特征点坐标
27         leftEye = shape[lStart:lEnd]
28         rightEye = shape[rStart:rEnd]
29         # 计算眼角比函数
30         leftEAR = eye_aspect_ratio(leftEye)
31         rightEAR = eye_aspect_ratio(rightEye)
32         # 如果画图的标志为True，就在原图上画出人眼的位置，并表示EAR参数
33         if draw_res:
34             leftEyeHull = cv2.convexHull(leftEye)
35             rightEyeHull = cv2.convexHull(rightEye)
36             cv2.drawContours(img_temp, [leftEyeHull], -1, (0, 255, 0), 1)
37             cv2.drawContours(img_temp, [rightEyeHull], -1, (0, 255, 0), 1)
38             res_str = 'l:' + str(float('%.2f' %leftEAR)) + " r:" + str(float('%.2f' %rightEAR))
39             cv2.putText(img_temp, res_str, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
40                         0.7, (0, 0, 255), 2)
41         # 如果检测到人脸，返回结果
42         if len(subjects) > 0:
43             return leftEAR, rightEAR, img_temp
44         # 否则返回False
45     else:
46         return False

```

4. 图片测试：

我们在 img 文件夹里准备了一些测试用的图片

```

1 img = cv2.imread('img/1.jpg')
2 score1, score2, img_res = check_eye(img, True)
3 ps.CommonFunction.show_img_jupyter(img_res)# 打印用于差分的两张图片
4 img = cv2.imread('img/2.jpg')
5 score1, score2, img_res = check_eye(img, True)

```

```
6 ps.CommonFunction.show_img_jupyter(img_res) # 打印用于差分的两张图片  
7
```

22.1.2 结果展示

结果如图所示：



图 22.3：测试结果

第四部分

进阶功能改装篇 - Advance

第二十三章 Advance 1 - 相机参数设置

内容提要

- 自动/手动曝光
- 明亮/弱光模式
- 自动/固定白平衡

这个教程介绍了 Powersensor 的采集参数设置方法，适用于 MT9v034 版的 sensor 板。由于相机的感光元件无法像人眼那样具备强大的自适应能力，因此需要在不同情景下设置不同的采集参数，以满足任务需求。

本实验的准备需要：

1. Powersensor 传感器

23.1 明亮模式/弱光模式

23.1.1 原理简介

相机的正常采集需要有足够的光线（光强）进入传感器，在明亮环境下很容易得到高画质的画面，在黑暗的环境，Powersensor 提供了高感光度的方案来采集图像，虽然会牺牲画面的清晰度，但能够有效地在弱光模式下得到图像信息。

23.1.2 函数说明

```
1 # 类名： SensorConfig  
2 # _____  
3 # 构造函数： SensorConfig()，无参数  
4 # _____  
5 # 明亮 / 弱光模式设置函数： set_light_mode(mode)，  
6 # -mode，必须，模式，可以是PowerSensor.SensorPara.Light_Dark（弱光模式）和  
    PowerSensor.SensorPara.Light_Normal（明亮模式）种的一个
```

23.1.3 参考例程

1. 弱光模式

```
1 config.set_exposure(mode=ps.SensorPara.EXPOSURE_MODE_AUTO, desired_lumin  
                      =48, max_gain=64)  
2 config.set_light_mode(ps.SensorPara.Light_Dark)  
3 for i in range(50):  
4     start = time.time()          # 记录开始时间  
5     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多  
        张图片  
6     imgMat = cam1.read_img_ori()    # 读入图像
```

```

7
8     # 缩小图像为320x240尺寸
9     origin = cv2.resize(imgMat, (320,240))
10
11    ps.CommonFunction.show_img_jupyter(imgMat) # 打印用于差分的两张图片
12    end = time.time()           # 记录结束时间
13    print(end - start)
14    time.sleep(0.1)

```

2. 明亮模式

```

1 config.set_exposure(mode=ps.SensorPara.EXPOSURE_MODE_AUTO, desired_lumin=48,
2                         max_gain=64)
3 config.set_light_mode(ps.SensorPara.Light_Normal)
4 for i in range(50):
5     start = time.time()           # 记录开始时间
6     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多张图片
7     imgMat = cam1.read_img_ori()   # 读入图像
8
9     # 缩小图像为320x240尺寸
10    origin = cv2.resize(imgMat, (320,240))
11
12    ps.CommonFunction.show_img_jupyter(imgMat) # 打印用于差分的两张图片
13    end = time.time()           # 记录结束时间
14    print(end - start)
15    time.sleep(0.1)

```

23.1.4 效果展示

1. 明亮模式

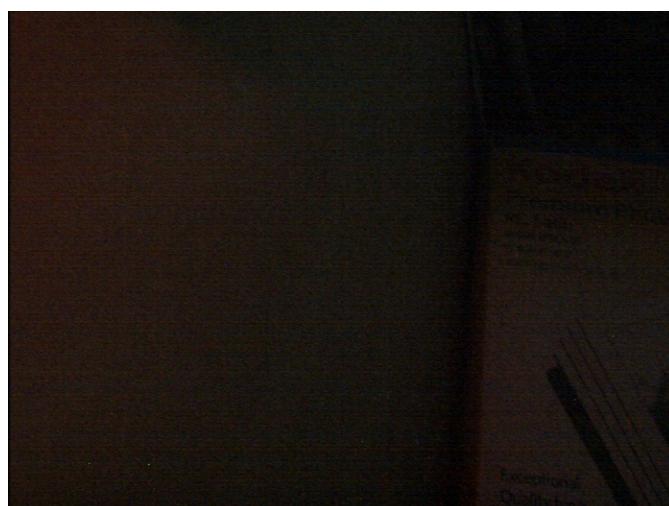


图 23.1: 黑暗环境下明亮模式采集效果

2. 弱光模式

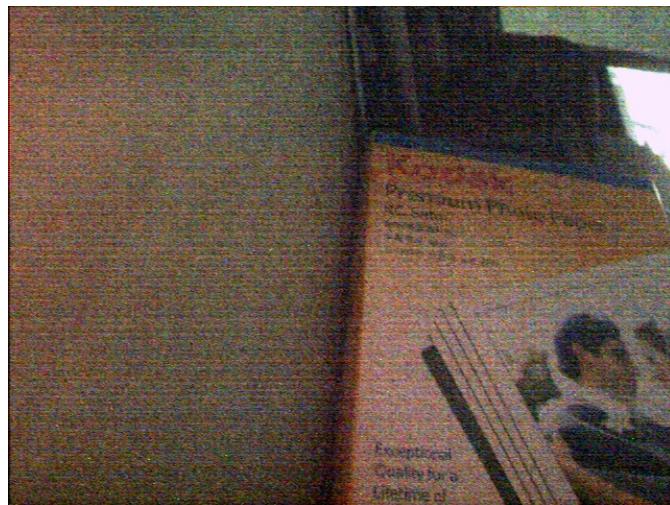


图 23.2: 黑暗环境下弱光模式采集效果

23.2 自动/手动曝光模式设置

23.2.1 原理简介

相机曝光是一个非常复杂的过程，曝光准确是使画面的亮度处于用户需求的状态。影响曝光的最重要的因素有三个，环境亮度、曝光时间、曝光增益，采集到的图像的亮度与环境亮度、曝光时间、曝光增益均正相关，因此在明亮的环境下要试用小的曝光时间和小的曝光增益，黑暗的环境则反之。

虽然加大曝光时间和曝光增益可以提高画面亮度，但是各有缺点，曝光时间大了会导致画面变糊，即拖尾的效果，曝光增益大了会增加画面的噪声。所以用户要根据情况调整参数设置，拍快速运动的物体用高增益，拍静止高画质的东西用低增益。

Powersensor 支持自动曝光和手动曝光两种模式。自动曝光主要设置的参数有期望的亮度（流明数）和最大增益，即相机会根据设置的亮度和允许的最大增益自动调节曝光时间和曝光增益；复杂的多干扰的环境下，需要使用手动曝光模式，手动曝光需要设置的主要参数有曝光时间和曝光增益，即相机直接使用用户设置的参数来曝光成像。

23.2.2 函数说明

```

1 # 类名: SensorConfig
2 #
3 # 构造函数: SensorConfig(), 无参数
4 #
5 # 自动/手动曝光设置函数: set_exposure(mode, desired_lumin, max_gain, period,
   gain),
6 # -mode, 必须, 模式, 可以是PowerSensor.SensorPara.EXPOSURE_MODE_MANUAL (手动
   曝光) 和 PowerSensor.SensorPara.EXPOSURE_MODE_AUTO (自动曝光, 默认) 种的一
   个;

```

```

7 # -desired_lumin, 自动曝光需要, 期望的亮度, 值范围是(10-64), 一般地明亮模式
8     下推荐为56左右, 弱光模式下推荐为15左右;
9 # -max_gain, 自动曝光需要, 最大增益, 值范围是(16-63), 按需要设置;
10 # -period, 手动曝光需要, 曝光时间, 值范围是(10-30000), 按需要设置;
11 # -gain, 手动曝光需要, 曝光增益, 值范围是(16-63), 按需要设置;

```

23.2.3 参考例程

1. 自动曝光模式

```

1   # 自动曝光, 亮度=58, 最大增益64
2 config.set_exposure(mode=ps.SensorPara.EXPOSURE_MODE_AUTO,
3     desired_lumin=58, max_gain=64)
4 for i in range(30):
5     start = time.time()          # 记录开始时间
6     clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印
7     多张图片
8     imgMat = cam1.read_img_ori()    # 读入图像
9
10
11    # 缩小图像为320x240尺寸
12    origin = cv2.resize(imgMat, (320, 240))
13
14
15    ps.CommonFunction.show_img_jupyter(imgMat) # 打印用于差分的两张图片
16    end = time.time()          # 记录结束时间
17    print('0-58-64', end - start)
18    time.sleep(0.1)
19
20
21    # 自动曝光, 亮度=32, 最大增益64
22 config.set_exposure(mode=ps.SensorPara.EXPOSURE_MODE_AUTO,
23     desired_lumin=32, max_gain=64)
24 for i in range(30):
25     start = time.time()          # 记录开始时间
26     clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印
27     多张图片
28     imgMat = cam1.read_img_ori()    # 读入图像
29
30
31    # 缩小图像为320x240尺寸
32    origin = cv2.resize(imgMat, (320, 240))
33
34
35    ps.CommonFunction.show_img_jupyter(imgMat) # 打印用于差分的两张图片
36    end = time.time()          # 记录结束时间
37    print('0-32-64', end - start)
38    time.sleep(0.1)
39
40
41    # 自动曝光, 亮度=32, 最大增益32
42 config.set_exposure(mode=ps.SensorPara.EXPOSURE_MODE_AUTO,
43     desired_lumin=32, max_gain=32)

```

```

33 for i in range(30):
34     start = time.time()          # 记录开始时间
35     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印
36     多张图片
37     imgMat = cam1.read_img_ori()    # 读入图像
38
39     # 缩小图像为320x240尺寸
40     origin = cv2.resize(imgMat, (320,240))
41
42     ps.CommonFunction.show_img_jupyter(origin) # 打印用于差分的两张图片
43     end = time.time()          # 记录结束时间
44     print('0-32-32', end - start)
45     time.sleep(0.1)

```

2. 手动曝光模式

```

1 # 手动曝光，时间500个单位，增益32
2 config.set_exposure(mode=ps.SensorPara.EXPOSURE_MODE_MANUAL, period
=500, gain=32)
3 for i in range(30):
4     start = time.time()          # 记录开始时间
5     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印
6     多张图片
7     imgMat = cam1.read_img_ori()    # 读入图像
8
9     # 缩小图像为320x240尺寸
10    origin = cv2.resize(imgMat, (320,240))
11
12    ps.CommonFunction.show_img_jupyter(origin) # 打印用于差分的两张图片
13    end = time.time()          # 记录结束时间
14    print('1-500-32', end - start)
15    time.sleep(0.1)
16
17 # 手动曝光，时间2000个单位，增益32
18 config.set_exposure(mode=ps.SensorPara.EXPOSURE_MODE_MANUAL, period
=2000, gain=32)
19 for i in range(30):
20     start = time.time()          # 记录开始时间
21     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印
22     多张图片
23     imgMat = cam1.read_img_ori()    # 读入图像
24
25     # 缩小图像为320x240尺寸
26     origin = cv2.resize(imgMat, (320,240))
27
28     ps.CommonFunction.show_img_jupyter(origin) # 打印用于差分的两张图片
29     end = time.time()          # 记录结束时间

```

```
28     print('1-2000-32', end = start)
29     time.sleep(0.1)
```

23.2.4 效果展示

1. 自动曝光模式

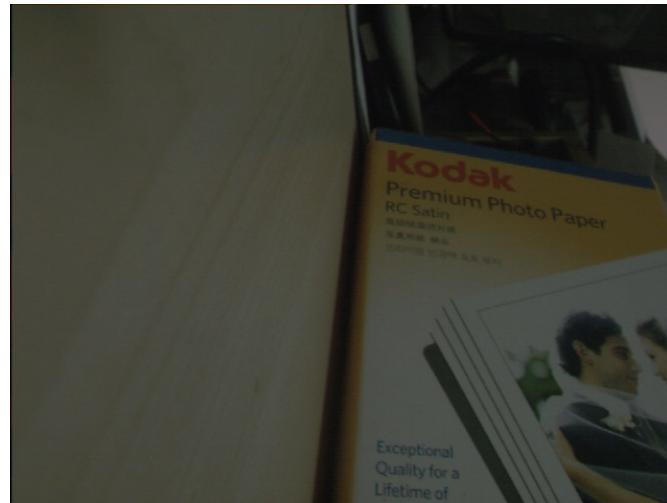


图 23.3: 自动曝光-亮度 58-最大增益 16

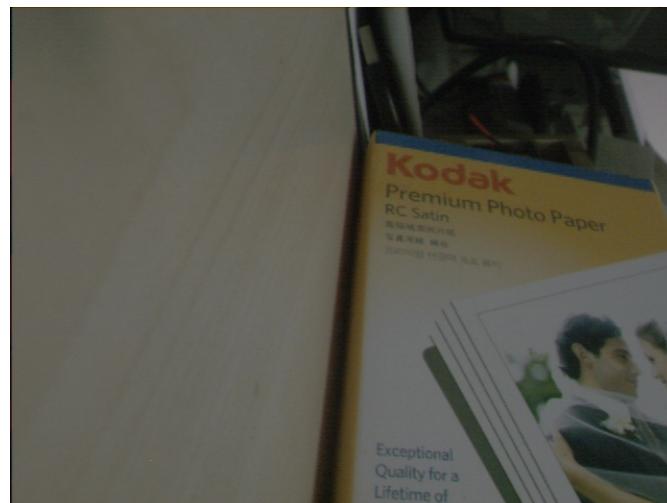


图 23.4: 自动曝光-亮度 58-最大增益 32

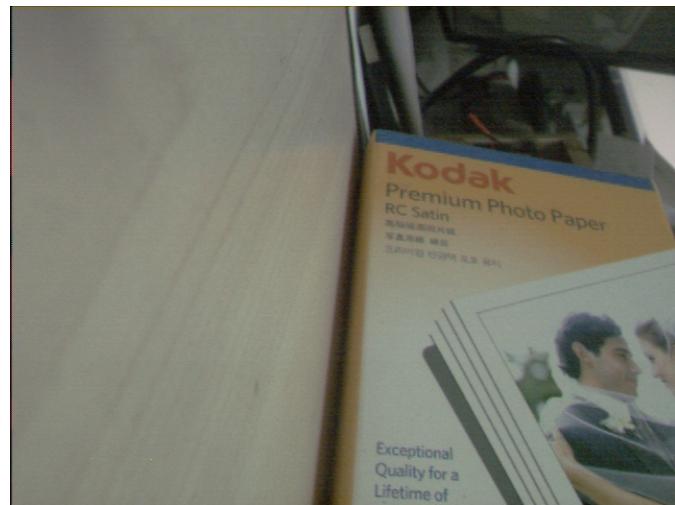


图 23.5: 自动曝光-亮度 58-最大增益 64

可以看出，随着最大增益的增大，画面的噪声也逐渐变大。

2. 手动曝光模式

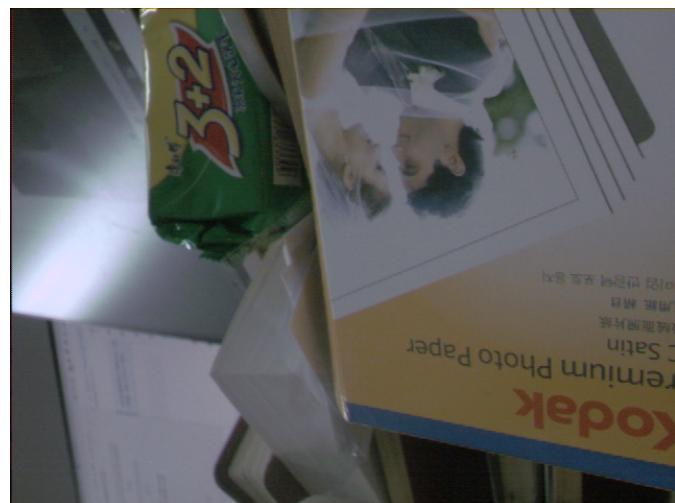


图 23.6: 手动曝光-曝光时间 500-增益 32



图 23.7: 手动曝光-曝光时间 2000-增益 32

手动曝光可以精确控制画面的效果。

23.3 固定白平衡/自动白平衡

23.3.1 原理简介

白平衡的作用是让画面看起来更白，是某些场合非常有用。颜色识别建议使用固定白平衡。固定白平衡即使用出厂标定的固定的白平衡参数，自动白平衡使用实时估计的白平衡参数。

23.3.2 函数说明

```

1 # 类名: SensorConfig
2 #
3 # 构造函数: SensorConfig(), 无参数
4 #
5 # 白平衡设置函数: set_wb_mode(mode),
6 # -mode, 必须, 模式, 可以是PowerSensor.SensorPara.AWB_Fix(固定白平衡) 和
    PowerSensor.SensorPara.AWB_GrayWold(自动白平衡) 中的一个

```

23.3.3 参考例程

1. 固定白平衡

```

1 # 普通白平衡
2 config.set_wb_mode(ps.SensorPara.AWB_Fix)
3 for i in range(30):
4     start = time.time()          # 记录开始时间
5     clear_output(wait=True)      # 清除图片, 在同一位置显示, 不使用会打印
多张图片
6 imgMat = cam1.read_img_ori()      # 读入图像
7
8 # 缩小图像为320x240尺寸
9 origin = cv2.resize(imgMat, (320, 240))
10
11 ps.CommonFunction.show_img_jupyter(imgMat) # 打印用于差分的两张图片
12 end = time.time()          # 记录结束时间
13 print('1-2000-32', end - start)
14 time.sleep(0.1)
15

```

2. 自动白平衡

```

1 # 自动白平衡
2 config.set_wb_mode(ps.SensorPara.AWB_GrayWold)
3 for i in range(30):
4     start = time.time()          # 记录开始时间

```

```

5      clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印
6      多张图片
7
8      imgMat = cam1.read_img_ori()          # 读入图像
9
10
11     # 缩小图像为320x240尺寸
12     origin = cv2.resize(imgMat, (320,240))
13
14     ps.CommonFunction.show_img_jupyter(imgMat) # 打印用于差分的两张图片
15     end = time.time()           # 记录结束时间
16     print('1-2000-32', end - start)
17     time.sleep(0.1)

```

23.3.4 效果展示

1. 固定白平衡



图 23.8：固定白平衡的采集效果

2. 自动白平衡



图 23.9：自动白平衡的采集效果

第二十四章 Advance 2 - FPGA 底层定制

内容提要

SSH 登录

FPGA 开发

新手请跳过这节教程。这节教程介绍的功能适合有经验的、准备进一步开发 powersensor 的开发者，更多的是提供 powersensor 本身设计的一些细节，方便用户 diy。所以对于使用本节教程出现问题，我们并没有义务提供技术支持。

这节教程的每个小节前会提示所需的知识和技能。

24.1 SSH 登录

24.1.1 需要的技能

- linux 基础指令

24.1.2 介绍

Powersensor 采用的是 linux 操作系统，在系统启动后可以通过 ssh 连接系统。

无论是 offline 模式还是 jupyter 模式，powersensor 的 ip 都是 192.168.8.8。

登录的账户是 debian，密码是 123。

登录后，用户就可以对 powersensor 的许多细节进行修改。

出了问题的最佳方式是重新刷机，问我们很可能也不知道，linux 水太深，容易淹死。

24.2 FPGA 开发

24.2.1 ARM 配置

- zynq 型号： xc7z020clg400-2 (active)
- 内存： MT41J256M16 RE-125
- Cpu 频率： 767MHz

24.2.2 常用接口

如果需要使用 USB、SD 卡、串口、按键等，请参照下图管脚定义：

功能	NET
USB	MIO28~39
SD卡	MIO40~45
u0-tx	MIO10
u0-rx	MIO11
u1-tx	MIO12
u1-rx	MIO13
btn0	MIO14
btn1	MIO15
PL_clk (50M)	U18

图 24.1: 常用接口管脚表

如果需要传感器板 (MT9V034)，请按如下管脚定义：

引脚	FPGA管脚	NET
led0-blue	U12	IO_L2N_T0_34
led1	T11	IO_L1P_T0_34
led2	T14	IO_L5N_T0_34
cm0-rst	M20	IO_L14P_T2_SRCC_34
cm0-scl	P19	IO_L13N_T2_MRCC_34
cm0-sda	P20	IO_L14N_T2_SRCC_34
cm0-vsync	R19	IO_0_34
cm0-href	T19	IO_25_34
cm0-pclk	M18	IO_L13P_T2_MRCC_34
cm0-d0	V16	IO_L18N_T2_34
cm0-d1	V15	IO_L10N_T1_34
cm0-d2	Y14	IO_L8N_T1_34
cm0-d3	Y18	IO_L17P_T2_34
cm0-d4	V16	IO_L18P_T2_34
cm0-d5	T16	IO_L9P_T1_DQS_34
cm0-d6	U17	IO_L9N_T1_DQS_34
cm0-d7	U20	IO_L15N_T2_DQS_34
cm0-d8	U19	IO_L12N_T1_MRCC_34
cm0-d9	T20	IO_L15P_T2_DQS_34

图 24.2: 传感器板管脚表

24.2.3 PL 接口

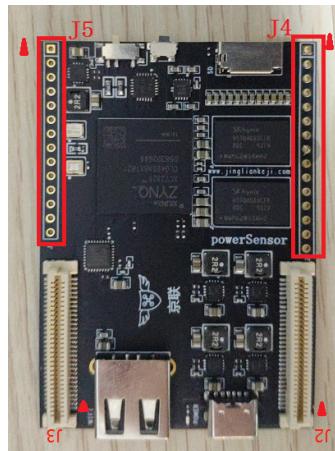


图 24.3: PL 接口图

1. 前面板接口：

如果需要自行定制传感器板，主板前面板的接口定义如下：

注 J2 为左侧接插件，J3 为右侧接插件。

其中 J2 的引脚定义（每个接插件有两排引脚）如下：

J2左			J2右		
NET	FPGA管脚	引脚号	引脚号	FPGA管脚	NET
IO_L4N_TO_34	V13	1	2	U14	IO_L11P_T1_SRCC_34
IO_L5P_T1_34	V14	3	4	V13	IO_L3N_TO_DQS_34
IO_L10P_T1_34	V15	5	6	U15	IO_L11N_T1_SRCC_34
IO_L7P_T1_34	Y16	7	8	Y14	IO_L8N_T1_34
IO_L7N_T1_34	Y17	9	10	W15	IO_L10N_T1_34
IO_L17N_T2_34	Y19	11	12	W16	IO_L18N_T2_34
IO_L16N_T2_34	Y20	13	14	Y18	IO_L17P_T2_34
IO_L16P_T2_34	Y20	15	16	Y16	IO_L18P_T2_34
JTAG_TCK	F9	17	18	T16	IO_L9P_T1_DQS_34
JTAG_TDI	G6	19	20	U17	IO_L9N_T1_DQS_34
JTAG_TDO	F6	21	22	U20	IO_L15N_T2_DQS_34
JTAG_TMS	J6	23	24	U19	IO_L12N_T1_MRCC_34
IO_L6P_TO_34	P14	25	26	T20	IO_L15P_T2_DQS_34
IO_L1N_TO_34	T10	27	28	T19	IO_25_34
IO_L6N_TO_WREF_34	R14	29	30	R19	IO_0_34
IO_L2P_TO_34	T12	31	32	P20	IO_L14N_T2_SRCC_34
IO_L1N_TO_34	T10	33	34	P19	IO_L13N_T2_MRCC_34
IO_L5P_TO_34	T14	35	36	N20	IO_L14P_T2_SRCC_34
IO_L2N_TO_34	U12	37	38	N18	IO_L13P_T2_MRCC_34
IO_L5N_TO_34	T15	39	40		
IO_L4P_TO_34	V12	41	42		
		43	44		
		45	46	3.3V	
		47	48	3.3V	
		49	50	3.3V	
		51	52	3.3V	
5V	53	54	GND		
5V	55	56	GND		
5V	57	58	GND		
5V	59	60	GND		

图 24.4: 左侧 PL 接口

其中 J3 的引脚定义（每个接插件有两排引脚）如下：

J3左			J3右		
NET	FPGA管脚	引脚号	引脚号	FPGA管脚	NET
		1	2	G20	IO_L18N_T2_AD13N_35
		3	4	G19	IO_L18P_T2_AD13P_35
GND		5	6	GND	
		7	8	H20	IO_L17N_T2_AD5N_35
		9	10	J20	IO_L17P_T2_AD5P_35
GND		11	12	GND	
		13	14	H18	IO_L14N_T2_AD4N_SRCC_35
		15	16	J18	IO_L14P_T2_AD4P_SRCC_35
GND		17	18	GND	
		19	20	K18	IO_L12N_T1_MRCC_35
		21	22	K17	IO_L12P_T1_MRCC_35
GND		23	24	GND	
		25	26	L16	IO_L11P_T1_SRCC_35
		27	28	L17	IO_L11N_T1_SRCC_35
GND		29	30	GND	
		31	32	L20	IO_L9N_T1_DQS_AD3N_35
		33	34	L19	IO_L9P_T1_DQS_AD3P_35
GND		35	36	GND	
		37	38	M20	IO_L7N_T1_AD2N_35
		39	40	M19	IO_L7P_T1_AD2P_35
GND		41	42	GND	
		43	44	M18	IO_L8N_T1_AD10N_35
		45	46	M17	IO_L8P_T1_AD10P_35
GND		47	48	GND	
GND		49	50	GND	
GND		51	52	GND	
GND		53	54	GND	
GND		55	56	GND	
5V		57	58	5V	
5V		59	60	5V	

图 24.5: 右侧 PL 接口

2. 背板接口：

powersensor 配置了一个 2.54 间距的排针接口：

注 J4 为左侧接插件，J5 为右侧接插件。

其中 J4 的引脚定义（每个接插件有两排引脚）如下：

J4		
引脚	FPGA管脚	NET
1	N16	IO_L21N_T3_DQS_AD14N_35
2	N15	IO_L21P_T3_DQS_AD14P_35
3	L15	IO_L22N_T3_AD7N_35
4	L14	IO_L22P_T3_AD7P_35
5	J14	IO_L20N_T3_AD6N_35
6	K14	IO_L20P_T3_AD6P_35
7	E8	PS_MIO13_500
8	D9	PS_MIO12_500
9		
10		
11		
12	GND	
13	GND	
14	5V	
15	5V	

图 24.6: 左侧排针接口

其中 J5 的引脚定义（每个接插件有两排引脚）如下：

J5		
引脚	FPGA管脚	NET
1	P15	IO_L24P_T3_34
2	R16	IO_L19P_T2_34
3	P16	IO_L24N_T3_34
4	V17	IO_L21P_T3_DQS_34
5	W18	IO_L22P_T3_34
6	V18	IO_L21N_T3_DQS_34
7	W19	IO_L22N_T3_34
8	T17	IO_L20P_T3_34
9	N17	IO_L23P_T3_34
10	P18	IO_L23N_T3_34
11	R18	IO_L20N_T3_34
12	R17	IO_L19N_T3_VREF_34
13	GND	
14	3V3	

图 24.7: 右侧排针接口

24.2.4 机械尺寸

机械尺寸详情见<http://bbs.jingliankeji.com/forum.php?mod=viewthread&tid=10&extra=page%3D1>

第二十五章 Advance 3 -TcpUdp 通信

内容提要

- TcpUdp
- 上位机
- 参数设置
- 图像传输

25.1 介绍

这个教程主要解决 powersensor 部署远程图像显示和调参的问题。主要通过 tcp、udp 和远程上位机通信，完成图像和数据的传输。主要有以下特点：

这个通信有两个部分，我们称 powersensor 为节点，称 pc 为主机。所以这种通信方式是一个主机，多个节点的。

关键概念：

- 这个通信的通道是以太网，可以是有线的网络，也可以是无线 wifi。
- TCP 通信用于传输参数，此时，节点为服务器，主机为客户端；UDP 通信用于传输图像，此时，节点为客户端，主机为服务器。
- 启动过程，首先，主机通过节点的 IP 连接节点的 TCP 服务器，同时节点会获取主机的 IP 地址；然后，节点会连接主机的 UDP 服务器，在主机发出显示图像的指令后，在主机上显示图像。
- 在主机发出写入参数指令后，参数会被传输到节点，节点更新参数的同时，会将参数写入文件，下次启动服务器时自动读取此文件作为初始参数。
- 这个程序采用多线程实现，与用户代码不会冲突。
- 传输的参数主要分 3 类，一个是布尔值参数，共 8 个，与上位机的勾选框对应。一个是整形参数，与上位机的整形参数区对应。一个是浮点参数，与上位机的浮点参数区对应。

注意 25.1

- 通信协议、powersenser 端的代码是开源的，windoes 上位机只是我们提供的一种方式，免费使用，用户可以开发自己的主机端。
- 只有 1.4 及以后的固件可以使用这个功能。
- UDP 传输图像一样是消耗计算资源的。



25.2 实验过程

25.2.1 准备工作

1. Powersensor 端：在 powersensor_workspace0.quick_starttcpudp_img 目录下应该存在 TCP 数传和 UDP 图传案例.ipynb 和 bk.jpg。没有的话需要用户自行上传和创建。
2. 主机端：请下载安装 Powersensor_tcpUdp 上位机 v1.0.0.msi，下载地址：

链接: <https://pan.baidu.com/s/1JxLUMGoxo4PvUets-89Nlw>

提取码: 1cfj

复制这段内容后打开百度网盘手机App，操作更方便哦

25.2.2 主要过程

1. 在 powersensor 上，首先包含需要的包

```

1 import numpy as np
2 import cv2
3 import time
4 import numpy as np
5 import struct
6 import socket
7 import TcpUdpHelper
8 import PowerSensor as ps
9

```

2. 在 powersensor 上，构建自定义的数据类，这个应该与上位机适配，如果是用户自己开发的上位机，此处应该修改通信协议

```

1 class PowersensorPara_ex(TcpUdpHelper.PowersensorPara):
2     def __init__(self, debug=False):
3         super(PowersensorPara_ex, self).__init__()
4         self.paraSwitch = [False, False, False, False, False, False, False,
5                           False]
6         self.paraInt = np.array(np.zeros(2), dtype=np.int32)
7         self.paraFloat = np.array(np.zeros(8), dtype=np.float32)
8
8     def unpackBuffer(self, data_buf):
9         for i in range(8):
10             # print(data_buf[4 + i])
11             if data_buf[4 + i] > 0:
12                 self.paraSwitch[i] = True
13             else:
14                 self.paraSwitch[i] = False
15             temp = struct.unpack('ii', data_buf[12:20])
16             self.paraInt = np.array(temp, np.int32)
17             temp = struct.unpack('ffffffff', data_buf[20:52])

```

```

18     self.paraFloat = np.array(temp, np.float32)
19
20     def packParas(self):
21         buffer = struct.pack("i" * 8, self.paraInt[0], self.paraInt
22 [1], self.paraFloat[0], self.paraFloat[1],
23         self.paraFloat[2], self.paraFloat[3], self.paraFloat[4], self.
24         paraFloat[5],
25         self.paraFloat[6], self.paraFloat[7])
26         cmd_buf = bytearray(52)
27         cmd_buf[0] = 0xfa
28         cmd_buf[1] = 0xf5
29         cmd_buf[2] = 0
30         cmd_buf[3] = 0x02
31         for i in range(8):
32             if self.paraSwitch[i]:
33                 cmd_buf[i + 4] = 1
34             else:
35                 cmd_buf[i + 4] = 0
36         for i in range(len(buffer)):
37             cmd_buf[i + 12] = buffer[i]
38
39     return cmd_buf
40
41

```

3. 在 powersensor 上，新建服务器，这个端口号需要和上位机保持一致。如果出现端口已经被占用，则需要修改，使用其他的端口。

```

1 x = PowersensorPara_ex()
2 # 端口号必须和上位机定义的一致
3 server = TcpUdpHelper.PowersensorServer(port=10775, para=x)
4

```

4. 在 Powersensor 上，测试原始数据：

```

1 # 打印原始数据
2 print('开关型数据:' + str(x.paraSwitch))
3 print('整型数据:' + str(x.paraInt))
4 print('浮点型数据:' + str(x.paraFloat))
5
6 # 在上位机上修改数据后，点写入参数，然后打印查看参数的变化
7 print('开关型数据:' + str(x.paraSwitch))
8 print('整型数据:' + str(x.paraInt))
9 print('浮点型数据:' + str(x.paraFloat))
10

```

1.4 测试

```
In [12]: # 打印原始数据
print('开关型数据:' + str(x paraSwitch))
print('整形数据:' + str(x paraInt))
print('浮点型数据:' + str(x paraFloat))

开关型数据:[True, False, False, True, True, False, True, False]
整形数据:[10 10]
浮点型数据:[ 0.1 -0.2  0.3 -0.4  0.5 -0.6 -0.7 -0.8]

In [13]: # 在上位机上修改数据后, 点写入参数, 然后打印查看参数的变化
print('开关型数据:' + str(x paraSwitch))
print('整形数据:' + str(x paraInt))
print('浮点型数据:' + str(x paraFloat))

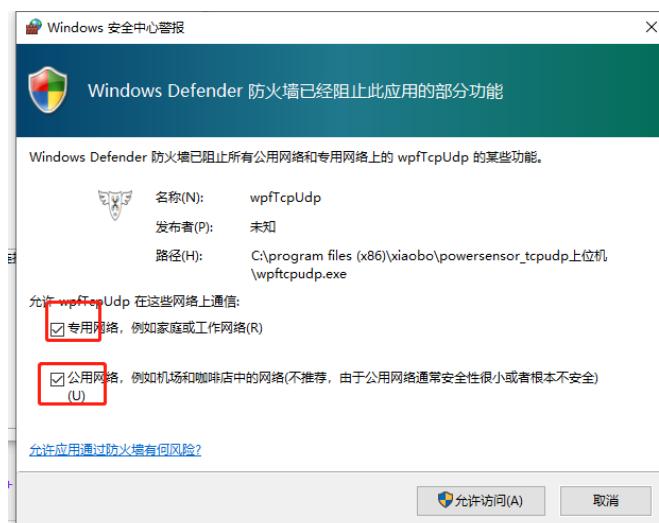
开关型数据:[True, False, False, True, True, False, True, False]
整形数据:[34 10]
浮点型数据:[ 0.1 -0.2  0.3 -0.4  0.5 -0.6 -0.7 -0.8]
```

5. 在 powersensor 上, 启动服务器

```
1 x = server.startServer()
2
```

到此就可以使用上位机连接 powersensor 的 tcp 服务器。

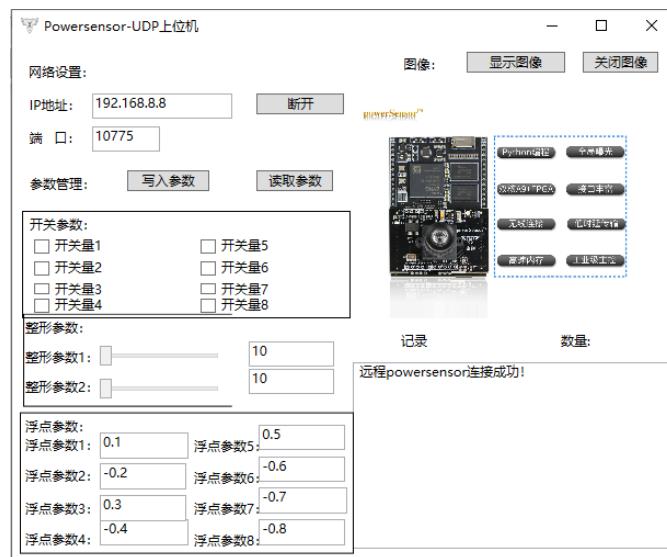
6. 在 PC 上, 启动 Powersensor_tcpUpd 上位机, 填入 powersensor 的 ip 地址和端口号。如果采用 jupyter 模式, powersensor 的 ip 就是 192.168.8.8; 如果采用 offline 模式或者使用用户无线路由, 请通过路由器后台查找 powersensor 的 ip:



7. 在 PC 上, 点击读取参数按钮, 即可读取 Powersensor 现在的变量, 读取到的参数应该与 powersensor 测试得到的数据一致。
8. 在 PC 上, 修改一个参数后, 点击写入参数按钮, 然后在 powersensor 上测试, 应该可以读取到参数的变化。

* UDP 图像显示

1. 在上位机成功连接到 Powersensor 后, 点击显示图像按钮, 即可在上位机上看到 powersensor 写入到变量 x.img 的数据。默认是一张 powersensor 的图片。



2. 可以通过动态地给 x.img 赋值, 修改 UDP 传输的图像



3. 在上位机上, 点击关闭图像按钮, powersensor 的 UDP 客户端会停止传输图像, 这有利于节省计算资源。

25.2.3 offline 模式

资料包里提供了一个 offline 模式的参考模板, 用户可以按自己的需求自行修改。

第二十六章 Advance 4 - 高级网络连接

内容提要

连接无线路由

网线使用

Powersensor 主要通过以太网进行编程、交互，目前支持 wifi 无线网络和 DJ45 百兆网口。wifi 和 DJ45 分布通过 usb-wifi 模块和 usb 转网口的方式实现。

目前测试过的支持的 wifi 芯片有：

- RT3070
- RT5370
- RTL8188cu

Powersensor 的 wifi 无线连接在 jupyter 和 online 的普通模式下，是 ap 状态，会主动发射局域网络供用户连接，这个大家应该很了解了，所以这里针对无线网络主要介绍如何连接用户路由器的部分。

针对 USB 转网口的部分，相对复杂一些，会在本章的第二部分进行介绍。

26.1 连接无线路由

26.1.1 需要的技能

- linux 基础指令
- 路由器配置相关知识

26.1.2 介绍

Powersensor 的 jupyter 模式除了通常的状态，还有一种特殊的调试状态，进入调试状态的方法是：

在上电时，用手按住主板上的按键，一直等到白灯变成红灯（通常的状态是会变成蓝灯）。此时，Powersensor 会按 sd 卡（boot 盘）下的 wpa_supplicant.conf 的配置连接指定的路由器。

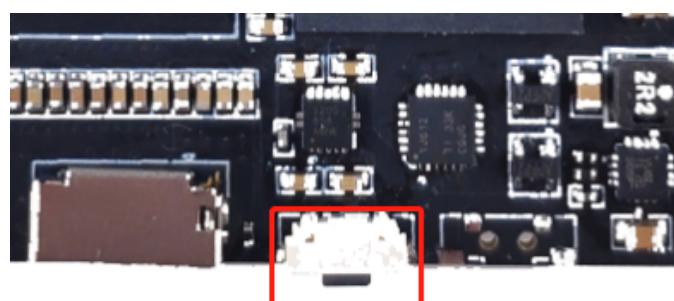


图 26.1: 连接无线路由的按键

```

1 ctrl_interface=/var/run/wpa_supplicant
2 ctrl_interface_group=0
3 network={
4 # 这里放你的路由器名称
5 ssid="tplink"
6 # 这个是原密码，这行是没用的
7 #psk="12345678"
8 # 这里放加密后的密码
9 psk=c3e47aa2a2c3b969ea9af4a3af850c66b392279722612e30e133842e36513ef3
10 # 下面是wpa2加密方式的配置（通常的方式），如果你的路由器加密方式特殊请另外配置
11 proto=RSN
12 key_mgmt=WPA-PSK
13 pairwise=CCMP TKIP
14 group=CCMP TKIP
15 }

```

那么如何根据明文密码（如 12345678）求密文？下面的网址可以提供密文计算功能：

<https://www.wireshark.org/tools/wpa-psk.html>

效果如下所示：

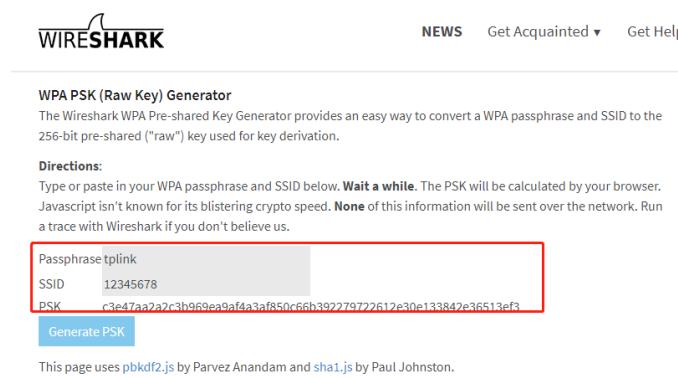


图 26.2: 获取加密后的密文

那么此时如何访问 Powersensor 呢？因为 IP 是路由器动态分配的，所以 IP 不再是固定的 192.168.8.8。

有两种办法来登录 Powersensor：

- 通过主机名访问

当确保只有一个 Powersensor 连接路由器的时候，可以通过主机名访问。

<http://powersensor-7020>



图 26.3: 使用主机名登录 Powersensor

注 推荐使用 Chrome 浏览器或者 firefox 浏览器，ie/edge 浏览器被 360 劫持后是无法登录的。

2. 通过路由器客户端查询 IP。

登录你的路由器，从路由器的 DHCP 服务器-> 客户端列表中可以找到 powersensor 对应的 ip。然后通过查询到的 ip 来登录。



图 26.4: 无线路由客户端列表

26.2 USB 转网口设备的使用

1.4 以后的固件版本支持 usb 转网口设备



使用方法与 usbwifi 模块类似，将此模块的 usb 与 ps 相连，同时使用一根网线与用户电脑或者路由器相连。

1. 使用路由器时，用户电脑与 powersensor 必须处于同一个路由器下。
2. 使用路由器时，路由器应该开启 dhcp 功能
3. 如果使用 usb-hub 同时连接用户 wifi 模块和 usb 转网口模块，默认优先使用 wifi 模块

主要有 3 种模式

模式	连接对象	ip地址分配
jupyter直连	PC	powersensor
jupyter连路由器	路由	路由器
offline	路由	路由器

1. jupyter 直连

这个与 wifi 模块最为相似，将此模块的 usb 与 ps 相连，同时使用一根网线与用户电脑相连，然后给 powersensor 上电，等待蓝灯亮起，即可在浏览器中启动 jupyter。

2. jupyter 连路由器

这个与 wifi 模块连接用户自定义路由器相似，将此模块的 usb 与 ps 相连，同时使用一根网线与路由器相连，路由器需要开启 dhcp 功能。此时 powersensor 的 ip 由路由器分配，通过查询得到的 ip 登录 powersensor，用户需要通过路由器的后台或者以下命令登录：

`http://wersensor-7020/`

3. offline

这个模式与 jupyter 连路由器模式比较相似，唯一的区别是不启动 jupyter，而是执行 offline.py 文件。

第五部分

人工智能加速篇 - AI

第二十七章 AI 1.1 - 深度学习模型训练

内容提要

- | | |
|-------------------------------|----------------------------------|
| <input type="checkbox"/> 写在前面 | <input type="checkbox"/> 软件安装与启动 |
| <input type="checkbox"/> 背景知识 | <input type="checkbox"/> 模型训练与保存 |

27.1 写在前面

- 最好懂一点机器学习
- 案例的整个流程(包括训练)建议先在虚拟机里跑通测试,再尝试外部显卡训练的模型

27.2 下载链接

注意 27.1

考虑到软件有可能及时更新,我们把虚拟机及本章各个案例的下载链接统一整理在网站论坛的指定页面上,请大家移步下载:

<http://bbs.jingliankeji.com/forum.php?mod=viewthread&tid=16&extra=>



27.3 背景知识

1. 人工神经网络

人工神经网络(Artificial Neural Network,即ANN),是20世纪80年代以来人工智能领域兴起的研究热点。它从信息处理角度对人脑神经元网络进行抽象,建立某种简单模型,按不同的连接方式组成不同的网络。在工程与学术界也常直接简称为神经网络或类神经网络。神经网络是一种运算模型,由大量的节点(或称神经元)之间相互联接构成。每个节点代表一种特定的输出函数,称为激励函数(activation function)。每两个节点间的连接都代表一个对于通过该连接信号的加权值,称之为权重,这相当于人工神经网络的记忆。网络的输出则依网络的连接方式,权重值和激励函数的不同而不同。而网络自身通常都是对自然界某种算法或者函数的逼近,也可能是一种逻辑策略的表达。

2. 深度学习

深度学习是机器学习中一种基于对数据进行表征学习的算法。目前常说的深度学习技术主要指利用深层(5层或以上)的神经网络对一个模型(比如手写数字识别)进行逼近的技术。深度学习可以看成一个万能逼近器,只要提供足够多的样本和标签,通过训练,深度学习就可以代替人类,完成生活中的分类、跟踪等任务。

3. 深度学习的模型、结构、权值

我们常说的深度学习的模型主要包含神经网络的结构和权值两个部分，为什么要把这两个参数分开呢？因为一般网络结构具有较好的泛化能力，就是说识别水果和识别数字可以使用同一个网络结构，但是权值往往具有针对性，同一个结构的神经网络通过载入不同的权值来实现不同的识别任务。下文所说的模型编译指：当网络训练好要对模型进行部署时，把网络权值固化到网络结构里，并编译成 powersensor 能够运行的机器码。

4. tensorflow

tensorflow 是 google 开发的一个机器学习库，支持 python 等多种编程语言。虽然深度学习本身的结构复杂，很难从零开始编程实现，但是通过 tensorflow，哪怕一个中学生也可以通过几十行代码完成简单的深度学习网络的实现和训练。

5. dpu 和 dnndk

DPU 是 xilinx 推出的一个适用于 FPGA 的用于深度学习加速的 IP 核，为了方便用户将自己训练的模型部署到 DPU 中，xilinx 推出了 dnndk 来完成模型编译（将用户模型编译成 DPU 能运行的模型）。可以说，DPU 是硬件 IP 核，而 dnndk 是软件管理的函数库。由于 dnndk 只能在 linux 平台下运行，为了方便大家，我们准备了一个按照好 dnndk 的 vmware 虚拟机，这样用户就可以在 windows 下通过虚拟机完成模型编译。

6. Anaconda

Anaconda 是一个免费开源 [5] 的 Python 和 R 语言的发行版本，用于计算科学（数据科学、机器学习、大数据处理和预测分析），Anaconda 致力于简化包管理和部署。Anaconda 的包使用软件包管理系统 Conda[6] 进行管理。（来自 wiki 百科）。简单的说，Anaconda 是一个 python 的包提供平台，软件仓库，同时配有 conda 管理工具。

7. conda

conda 是一个开源跨平台的包管理和环境管理系统，常用于 python 虚拟环境的搭建与管理。conda 允许用户从不同的源下载和升级软件包。conda 使得用户可以同时安装多个 python 虚拟环境，通过激活的方式，在不同的任务中使用不同的虚拟环境。

27.4 深度学习案例的开发流程

在了解了以上的基础知识后，如果要把一个深度学习模型部署到 powersensor 上运行需要进行以下流程。

首先打开虚拟机，在虚拟机中运行我们编写好的脚本 `star.sh` (这个脚本会帮助你打开 jupyter)

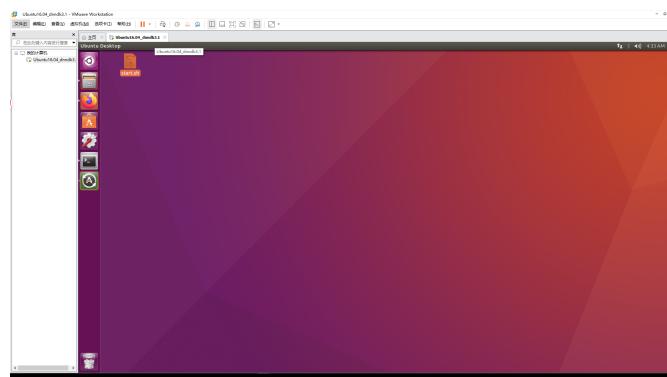
然后使用 tensorflow（本文使用的是 tensorflow2.0）完成深度学习模型的设计和训练

再然后，dnndk 环节，在虚拟机下完成模型权值固化，并使用 dnndk 将 tensorflow 模型编译成 dpu 能够运行的模型（elf 模型）。

最后，edge 环节，在 powersensor 下编写调度程序，完成最终的深度学习任务。
本章主要介绍前两步，网络的训练和编译。

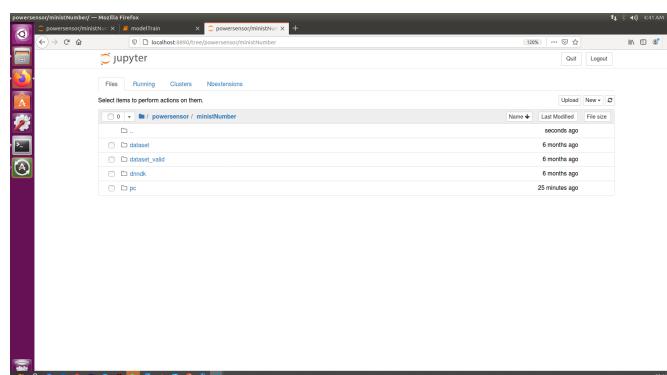
27.4.1 虚拟机的安装与启动

- 安装 vmware 并打开我们准备好的虚拟机，下载链接见27.2
- 启动虚拟机，并运行 star.sh 脚本（虚拟机密码是 123）



27.4.2 模型训练

1. 在虚拟机中打开 star.sh 脚本，打开 powerSensor 文件夹，里边有我为你们下载好的 ministNumber 数据集和会用到的一些文件。可以看到以下目录结构



其中，dataset 中存放的是用于训练和测试的数据集，dataset_valid 存放的是用于验证的数据集。pc 存放的是 tensorflow 相关的训练文件，用于训练模型。dnndk 存放的是编译所需的配置文件。edge 存放的是在 powersensor 运行的调度文件。

2. 进入 pc 文件夹，打开 modelTrain.ipynb，按照记事本的提示逐个运行。
3. 首先是包含头文件和重要的参数初始化，如果没有 GPU，请把 GPU 下面的 4 行注释掉（注意虚拟机不能使用 GPU）

```

1 import cv2
2 import numpy as np
3 import os
4 import tensorflow as tf
5 from tensorflow import keras

```

```

6      import random
7      import time
8      import matplotlib.pyplot as plt
9      from matplotlib.font_manager import FontProperties
10
11      # 有GPU才能打印
12      # gpus = tf.config.experimental.list_physical_devices(device_type
13      # ='GPU')
14      # cpus = tf.config.experimental.list_physical_devices(device_type
15      # ='CPU')
16      # print(gpus, cpus)
17      # for gpu in gpus:
18      #     tf.config.experimental.set_memory_growth(gpu, True)
19
20      # 让Matplotlib正确显示中文
21      import matplotlib as mpl
22      mpl.style.use('seaborn')
23      # mpl.rcParams['font.sans-serif']=['SimHei']      # 用黑体显示中文
24      # mpl.rcParams['axes.unicode_minus']=False        # 正常显示负号
25      font = FontProperties(fname="/home/powersensor/Documents/simhei.
26      ttf")
27      # 训练用的图像尺寸
28      img_size_net = 28
29      # 训练的batch大小
30      batch_size = 32
31      # 数据库路径
32      dataset_path = '../dataset/minist-number/'
33      # 存放过程和结果的路径
34      run_path = './run/'
35      if not os.path.exists(run_path):
36          os.mkdir(run_path)
37      wordlist = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
38
39      # # 存放转换后的tf数据集的路径
40      # dataset_tf_path = run_path + 'flowersTf.tfrecords'
41      # dataset_nums = 4300

```

训练用的图像尺寸指输入给神经网络的图像的尺寸，决定模型的入口大小，这一讲用的数据集是预处理好的，因此原图像尺寸与训练用的图像尺寸是一致的，但在其他问题里面可能不一样，需要进行预处理。神经网络的训练一次往往使用多个样本，训练的 batch 大小描述的就是这个大小，一般去 2 的幂次。数据库的路径，请确认在 dataset\minist-number 下有 4 个训练数据。run 路径会临时生成，存放的是训练完的结果（需要拷贝到虚拟机里）。wordlist 是样本的标签，比如数字识别就是 0-9，水果识别就是苹果、梨。。。

4. 数据集读取与测试，在训练前，我们需要加载数据集，并把数据集分成训练集和测试集，训练集用于训练神经网络，测试集用于测试训练效果（不参与训练）。这里会随机打印几张样本，用来验证读取是否正确。

```

1      """
2          data_path: 数据集本地保存路径
3          kind: 数据集类别 (训练集: kind='train', 测试集: kind='t10k')
4      """
5
6      def load_mnist(data_path, kind='train'):
7          import os
8          import gzip
9          import struct
10         import numpy as np
11
12         '''Load MNIST data from `path`'''
13         labels_path = os.path.join(data_path, '%s-labels-idx1-ubyte.gz'%kind)
14
15         images_path = os.path.join(data_path, '%s-images-idx3-ubyte.gz'%kind)
16
17         print(images_path, labels_path)
18         with gzip.open(labels_path, 'rb') as lbpath:
19             data_file = lbpath.read()
20             _, nums = struct.unpack_from('>II', data_file, 0) # 取前2个整数, 返回一个元组
21             labels = np.frombuffer(data_file, dtype=np.uint8, offset=8)
22             print("{} labels:{}".format(kind, nums))
23
24         with gzip.open(images_path, 'rb') as imgpath:
25             data_file = imgpath.read()
26             _, nums, width, height = struct.unpack_from('>IIII', data_file, 0) # 取前4个整数, 返回一个元组
27             images = np.frombuffer(data_file,
28                 dtype=np.uint8, offset=16).reshape(len(labels), width, height)
29             print("{} datas shape:({},{},{})".format(kind, nums, width, height))
30
31         return images, labels
32
33     # 1. 读取数据集
34     (train_images, train_labels) = load_mnist(dataset_path, 'train')
35     (test_images, test_labels) = load_mnist(dataset_path, 't10k')
36
37     # 2. 图像预处理
38     train_images = train_images.reshape((-1, 28, 28, 1)) / 255.
39     test_images = test_images.reshape((-1, 28, 28, 1)) / 255.

```

```

39     # 3. 统计各种训练集中各种样本的数量
40     print('各个样本的数量：')
41     l = []
42     for x in train_labels:
43         l.append(wordlist[x])
44     plt.hist(l, rwidth=0.5)
45     plt.show()
46
47     # 4. 随机打印8个测试图像
48     fig, ax = plt.subplots(5, 2)
49     fig.set_size_inches(15, 15)
50     for i in range(5):
51         for j in range(2):
52             l = random.randint(0, len(test_labels))
53             ax[i, j].imshow(test_images[l, :, :, 0])
54             ax[i, j].set_title(wordlist[test_labels[l]])
55             ax[i, j].grid(False)
56     plt.tight_layout()
57

```

5. 网络结构设计，这里设计的是一个较为简单的6层神经网络，包含2个卷积层（用于提取特征），2个池化层（用于抽象特征），2个全连接层，全连接层用于特征的合并和类别的预测。因为我们有0-9共10个类别需要预测，所以输出层的神经元数量是10。compile是将设计的神经网络实现处理，这里我们使用了Adam作为优化器，这种优化器具有动量的特性，能够跃出一些常见的极小值点。lr是学习率。

```

1     model = keras.Sequential([
2         #keras.layers.Flatten(input_shape=(128, 128, 3)),
3         keras.layers.Conv2D(32, (3,3), padding="same", activation=tf.nn.
4             relu, input_shape=(img_size_net, img_size_net, 1), name='x_input'),
5             keras.layers.MaxPooling2D(pool_size=(2,2)),
6             keras.layers.Conv2D(64, (3,3), padding="same", activation=tf.nn.
7                 relu),
8                 keras.layers.MaxPooling2D(pool_size=(2,2)),
9                 keras.layers.Flatten(),
10                keras.layers.Dense(128, activation=tf.nn.relu),
11                keras.layers.Dropout(0.5),
12                # 最后一个层决定输出类别的数量
13                keras.layers.Dense(10, activation=tf.nn.softmax, name='y_out')
14            ])
15            model.compile(optimizer=keras.optimizers.Adam(lr=0.001),
16            loss='sparse_categorical_crossentropy',
17            metrics=['accuracy'])
18            model.summary()

```

6. 神经网络训练，对于这种简单的数据集，第一次训练的效果会非常好，一般地训练

集精度会随着训练进行逐渐提高，而测试集会呈现一个先上升再下降的趋势（过拟合）。

```

1      import datetime
2      log_dir = "." + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
3      tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=
4          log_dir, histogram_freq=1)
5      tick_start = time.time()
6      history = model.fit(train_images, train_labels, batch_size=
7          batch_size, epochs=5, validation_data=(test_images, test_labels),
8          callbacks=[tensorboard_callback])
9      tick_end = time.time()
10     print("Training completed. Elapsed ", str(tick_end - tick_start))

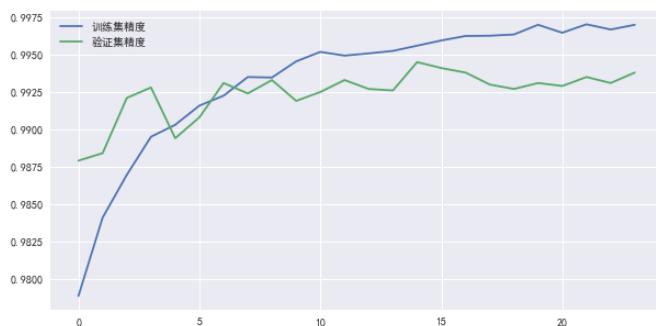
```

7. 训练完成，我们可以打印训练过程的精度变化

```

1      import matplotlib.pyplot as plt
2      fig, ax = plt.subplots(1, 1)
3      fig.set_size_inches(10, 5)
4      ax.plot(history.history['acc'][1:])
5      ax.plot(history.history['val_acc'][1:])
6      title = ['训练集精度'.decode('utf8'), '验证集精度'.decode('utf8')]
7
8      ax.legend(title, loc='upper left', prop=font)
9      plt.show()

```



27.4.3 模型保存

使用以下 python 语句将模型结构与模型权值分开保存，会在 run 目录下生成 `model_weight.h5` 和 `model_config.json`，保存的参数在 dnndk 的编译中会使用到这两个文件。分开保存是为了更好地适应 tensorflow 的版本变化。

```

1  model_test.save_weights("model_weight.h5")
2  json_config = model_test.to_json()
3  with open('model_config.json', 'w') as json_file:
4      json_file.write(json_config)

```

检查 ragged-tensor

使用记事本打开 modelconfig.json，如果在输入里出现"ragged": false, 字眼的字段，请手动把它删掉，因为 dnndk 里的 tensorflow 没有这个参数。没有就不用处理。

★ 模型验证

在训练好模型后，我们一般需要通过一些验证程序来测试保存的模型。

1. 加载模型，由于我们把结构和权值分开存储，所以我们要先解码模型再加载权值。

```

1 # 加载训练好的模型
2
3 with open(run_path + 'model_config.json') as json_file:
4     json_config = json_file.read()
5     model_test = tf.keras.models.model_from_json(json_config)
6     # Load weights
7     model_test.load_weights(run_path + 'model_weight.h5')
8     # model_test = tf.keras.models.load_model(run_path + "model.h5")
9     #model_test.compile(optimizer=keras.optimizers.Adam(lr=0.001),
10     # loss='sparse_categorical_crossentropy',
11     # metrics=['accuracy'])
12     model_test.summary()
13

```

```

Model: "sequential"
-----  

Layer (type)          Output Shape       Param #
-----  

x_input (Conv2D)      (None, 28, 28, 32)   320  

max_pooling2d (MaxPooling2D) (None, 14, 14, 32) 0  

conv2d (Conv2D)        (None, 14, 14, 64)   18496  

max_pooling2d_1 (MaxPooling2D) (None, 7, 7, 64) 0  

flatten (Flatten)      (None, 3136)        0  

dense (Dense)          (None, 128)         401536  

dropout (Dropout)      (None, 128)         0  

y_out (Dense)          (None, 10)          1290  

-----  

Total params: 421,642  

Trainable params: 421,642  

Non-trainable params: 0

```

2. 加载验证集，因为这个案例中测试集没有参与超参数的选择，所以直接使用了测试集当验证集。实际案例的训练集、测试集、验证集应该是独立的。此外，dnndk 在进行量化和在 powersensor 上测试的时候也需要使用验证集，所以这里将验证集放在一个独立的文件夹 dataset_valid 里。

```

1 """
2 data_path: 数据集本地保存路径
3 kind: 数据集类别 (训练集: kind='train', 测试集: kind='t10k')
4 """
5 def load_mnist(data_path, kind='train'):
6     import os
7     import gzip

```

```

8     import struct
9     import numpy as np
10
11    '''Load MNIST data from `path`'''
12    labels_path = os.path.join(data_path, '%s-labels-idx1-ubyte.gz'% kind)
13
14    images_path = os.path.join(data_path, '%s-images-idx3-ubyte.gz'% kind)
15
16    print(images_path, labels_path)
17    with gzip.open(labels_path, 'rb') as lbpath:
18        data_file = lbpath.read()
19        _, nums = struct.unpack_from('>II', data_file, 0) # 取前2个整数, 返回一个元组
20        labels = np.frombuffer(data_file, dtype=np.uint8, offset=8)
21        print("{} labels:{}".format(kind, nums))
22
23    with gzip.open(images_path, 'rb') as imgpath:
24        data_file = imgpath.read()
25        _, nums, width, height = struct.unpack_from('>IIII', data_file, 0) # 取前4个整数, 返回一个元组
26        images = np.frombuffer(data_file,
27                               dtype=np.uint8, offset=16).reshape(len(labels), width, height)
28        print("{} datas shape:({},{},{})".format(kind, nums, width, height))
29    return images, labels
30
31 dataset_valid_path = '../dataset_valid//minist-number/'
32 # 1. 加载数据集
33 (validSet_images, validSet_labels) = load_mnist(dataset_valid_path,
34 't10k')
35
36 # 2. 图像预处理
37 validSet_images = validSet_images.reshape((-1, 28, 28, 1)) / 255.

```

3. 使用验证集评价模型精度

```

1 # 使用tensorflow的函数评估精度
2 res = model_test.evaluate(validSet_images, validSet_labels)
3

```

4. 使用随机样本直观地观测预测结果:

```

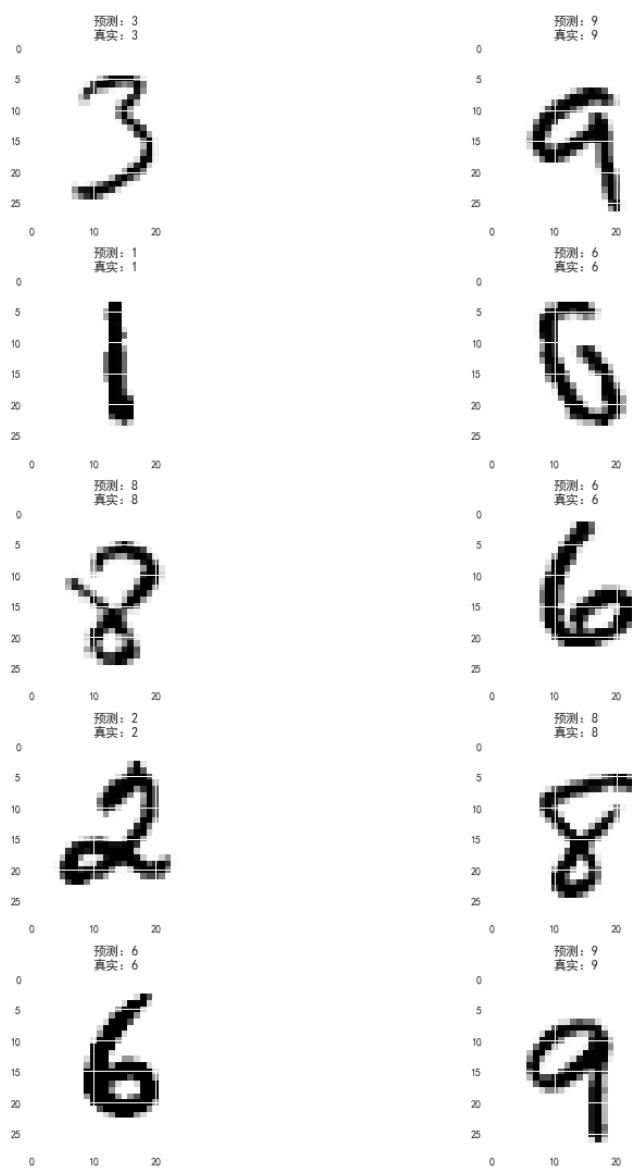
1 fig, ax = plt.subplots(5, 2)
2 fig.set_size_inches(15,15)
3 for i in range(5):
4     for j in range(2):

```

```

5         l = random.randint(0, len(validSet_labels))
6         pdt_label_fre = model_test.predict(validSet_images[l:l+1])
7         pdt_label = np.argmax(pdt_label_fre, axis=1)
8         ax[i, j].imshow(validSet_images[l, :, :, 0])
9         title = "预测: " + wordlist[pdt_label[0]] + "\n" + "真实: " +
wordlist[validSet_labels[l]]
10        title = title.decode('utf8')
11        ax[i, j].set_title(title, fontproperties=font)
12        l += 1
13
14        plt.tight_layout()

```



5. 模型转换

```

1      # 将模型和权值文件复制到dnndk文件夹中
2      !cp run/model_config.json run/model_weight.h5 ../dnndk
3      !chmod +x ../dnndk/1_vitisAI_keras2frozon.sh

```

```

4      ! chmod +x ../dnndk/2_vitisAI_tf_quantize.sh
5      ! chmod +x ../dnndk/3_vitisAI_tf_compile.sh
6      # 把权值固化到模型中
7      !cd '../dnndk';pwd;../dnndk/1_vitisAI_keras2frozen.sh
8      # 量化
9      !cd '../dnndk';pwd;../dnndk/2_vitisAI_tf_quantize.sh
10     # 编译
11    !cd '../dnndk';pwd;../dnndk/3_vitisAI_tf_compile.sh
12

```

若出现如下错误提示，是因为虚拟机内存不够导致的，重启虚拟机即可（注意重启虚拟机后只运行最后一段编译程序即可，之前训练好的模型和权值已经保存）

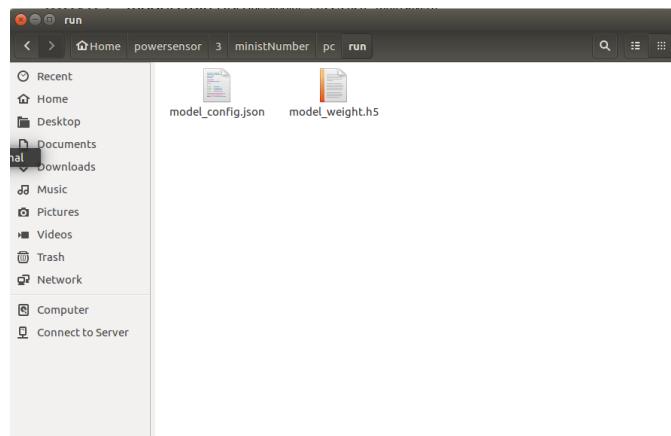
```

-----[OS error]----- Traceback (most recent call last)
<ipython-input-20-a6d87892171b> in <module>()
      1 get_ipython().system(u'cp run/model_config.json run/model_weight.h5 ./dnndk')
      2
      3 # Instead, we store the exit code in user_ns.
      4 self.user_ns['exit_code'] = system(self.var_expand(cmd, depth=1))
      5
      6 def system_raw(self, cmd):
-----[OS error]----- /home/powersensor/.local/lib/python2.7/site-packages/IPython/core/interactiveshell.py in system_piped(self, cmd)
 2209         # a non-None value would trigger func=sys_displayhook calls.
 2210         # Instead, we store the exit code in user_ns.
 2211         self.user_ns['exit_code'] = system(self.var_expand(cmd, depth=1))
 2212
 2213     def system_raw(self, cmd):
-----[OS error]----- /home/powersensor/.local/lib/python2.7/site-packages/IPython/utils/_process_posix.py in system(self, cmd)
 2214         child = pexpect.spawn(self.sh, args=['-c', cmd]) # Pexpect-U
 2215         else:
 2216             child = pexpect.spawn(self.sh, args=['-c', cmd]) # Vanilla Pexpect
 2217             flush = sys.stdout.flush
 2218             while True:
 2219
-----[OS error]----- /home/powersensor/.local/lib/python2.7/site-packages/pexpect/pty_spawn.py in __init__(self, command, args, timeout, maxread, searchwindowsize, logfile, cwd, env, ignore_sighup, echo, preexec_fn, encoding, codec_errors, dimensions, use_poll)
 2220         self.name = '<pexpect factory incomplete>'
 2221
 2222     else:
 2223         self._spawn(command, args, preexec_fn, dimensions)
 2224
 2225     self.use_poll = use_poll
 2226
 2227     self.pid = self.ptyproc.pid
-----[OS error]----- /home/powersensor/.local/lib/python2.7/site-packages/pexpect/pty_spawn.py in _spawn(self, command, args, preexec_fn, dimensions)
 2228         self.ptyproc = self._spawnpty(self.args, env=self.env,
 2229                                     cwd=self.cwd, **kwargs)
 2230
 2231     self.pid = self.ptyproc.pid
-----[OS error]----- /home/powersensor/.local/lib/python2.7/site-packages/ptyprocess/ptyprocess.py in spawn(cls, args, cwd, env, echo, preexec_fn, dimensions)
 2232         if use_native_pty_fork:
 2233             pid = os.fork()
 2234             if pid == CHILD:
 2235                 # Use internal fork_pty, for Solaris
 2236             else:
 2237                 # Use native pty_fork
 2238
-----[OS error]----- /usr/lib/python2.7/pty.py in fork()
 2239             master_fd, slave_fd = openpty()
 2240             pid = os.fork()
 2241             if pid == CHILD:
 2242                 # Establish a new session.
 2243
-----[OS error]----- [Errno 12] Cannot allocate memory

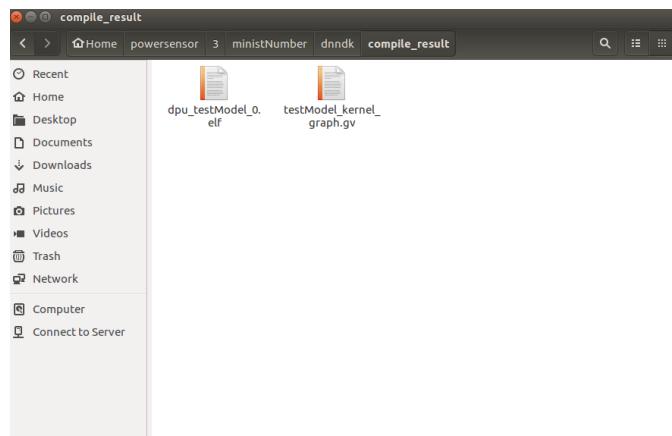
```

编译完成后会在 dnndk/compile_result 文件夹目录下生成 dpu_testModel_0.elf 文件

- 到这里训练和编译的任务就结束了。这个阶段的主要任务是根据实际问题设计合适的深度网络结构，通过训练得到满足需求的网络权值，并对模型进行编译。训练的结果在 pc 文件夹的 run 目录里：



编译生成的文件在 dnndk/compile_result 目录里：



27.4.4 加速模型训练过程 *

笔记 如果你擅长深度学习，并且能够在 PC 端独立地安装 anaconda 和配置环境，下面的信息可能对你有用，没有基础的用户建议跳过。

1. 安装 anaconda，下载地址（去我们的百度网盘下载）：

<https://www.anaconda.com/products/individual>

2. 启动 anaconda prompt，这个是终端，大部分包的安装等操作会在终端进行
3. 新建一个虚拟的 Anaconda 环境。命令过程需要输入 y 确认，这里都把这个步骤略过。

```

1 # 新建一个虚拟环境，名为ps
2 conda create -n ps
3 # 激活ps环境，每次启动prompt都需要激活，激活后左侧的标识会变成ps
4 (base) C:\Users\xxx>conda activate ps
5
6 (ps) C:\Users\xxx>
7

```

4. 逐条输入以下指令完成 python 软件包的安装

熟悉的用户可以通过 anaconda 换源来提升在国内的下载速度，教程见

<https://mirror.tuna.tsinghua.edu.cn/help/anaconda/>

```

1 conda install opencv
2 conda install matplotlib
3 conda install jupyter
4
5 # 没有显卡的安装这个tf
6 conda install tensorflow
7 # 有显卡的安装这个tf
8 conda install tensorflow-gpu
9

```

* 启动 jupyter（每次启动需要进行一次）

```
1 # 首先激活环境  
2 conda activate ps  
3 # 启动jupyter，路径填powersensor_flowerclassification所在盘的盘符（如e:  
4 )  
5 jupyter-notebook --notebook-dir 路径
```

需要注意的是，虽然在PC端神经网络的训练会比较快，但是后续的模型编译仍然要在虚拟机中完成。

第二十八章 AI 1.2 - 深度学习模型转换

内容提要

□ 介绍

□ 编译过程

28.1 介绍

本章主要介绍的 PowersensorAI 部署的第二个环节，通过 xilinx 的 DNNDK，将 tensorflow 训练好的模型进行固化、编译等操作。最终生成 powersensor 可以运行的 elf 文件。这个环节称为 dnndk 环节。**本章内容主要针对使用 PC 进行神经网络训练的同学，使用虚拟机进行训练的同学在最后一步已经完成模型编译，可以跳过这一章。**

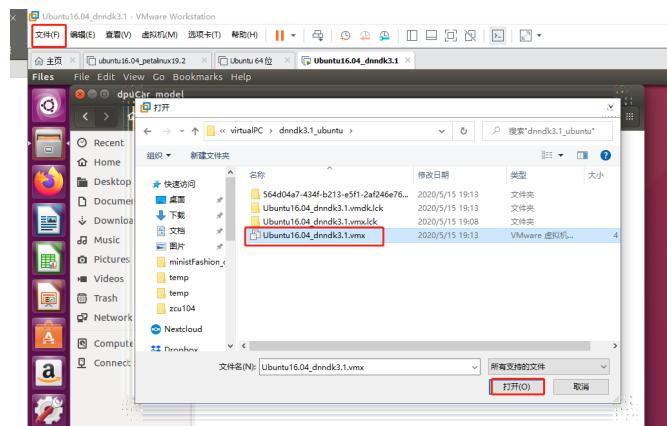
注意 28.1

事实上从上节的例程中可以看到，使用虚拟机可以直接完成模型训练、保存和编译过程，因此使用虚拟机训练的同学可直接跳过本章，不影响后续内容的学习，当然如果对模型转换的具体细节比较感兴趣，也可以按照本节介绍方法进行模型转换。

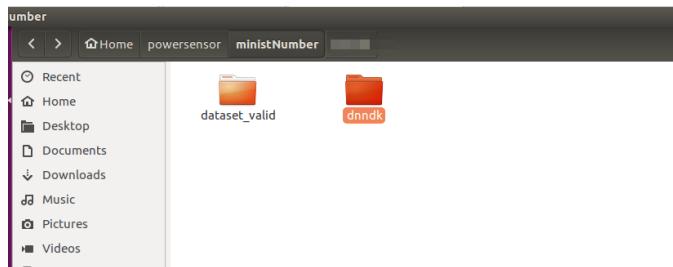


28.2 编译过程

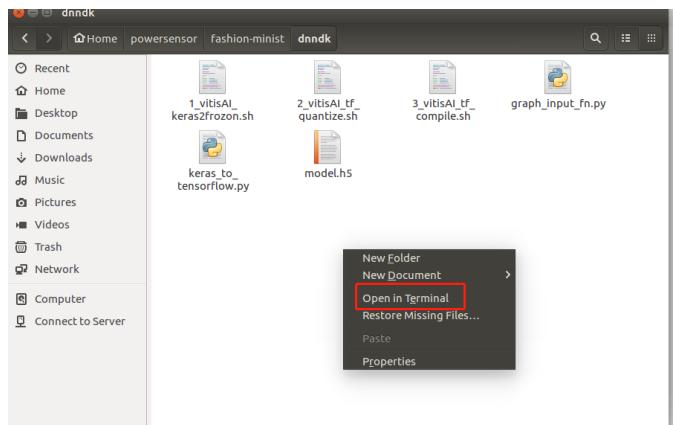
1. 打开虚拟机，登录密码是 123



2. 在虚拟机的/home/powersensor 目录下新建 ministNumber 文件夹（随教程发布的案例会已经存在），并把案例目录下的 dnndk 文件夹和 dataset_valid 文件夹复制到新建的文件夹下面。如果要使用自己新训练的模型，需要把自己的模型（在案例目录/pc/run 下面的 2 个模型文件）替换掉我们准备好的文件。



3. 在虚拟机里的 dnndk 文件夹里打开终端:



4. 第一步,模型固定,把tensorflow的权值固定到结构中,指令是./1_vitisAI_keras2frozen.sh

```
xiaobo@ubuntu:~/powersensor/fashion-minist/dnndk# ./1_vitisAI_keras2frozen
.sh
*****
Convert keras h5 model to frozon model begin
Vitis AI 1.1
*****
I0515 05:22:53.318743 139635748165376 keras_to_tensorflow.py:146]
    Converted output node names are: [u'dense_out/Softmax']
INFO:tensorflow:Froze 8 variables.
I0515 05:22:53.326545 139635748165376 tf_logging.py:115] Froze 8 variables
.
INFO:tensorflow:Converted 8 variables to const ops.
I0515 05:22:53.329962 139635748165376 tf_logging.py:115] Converted 8
variables to const ops.
I0515 05:22:53.332588 139635748165376 keras_to_tensorflow.py:178] Saved
the freezed graph at frozon_result/model.pb
*****
Convert completed
*****
```

frozon 完模型, 工具会把网络节点的名称打印出来。一般地, 第一个是输入节点, 最后一个是输出节点。

```

INFO:tensorflow:Converted 8 variables to const ops.
I0526 05:25:52.753216 139928559638272 tf_logging.py:115] Converted 8 variables to const
I0526 05:25:52.819758 139928559638272 keras_to_tensorflow.py:183] Saved the freezed graph.
WARNING:tensorflow:From printNodes.py:9: __init__ (from tensorflow.python.platform.gfile)
Instructions for updating:
use tf.gfile.FastGlob.
(u'input/input', '\n')
(u'x_input/kernel', '\n')
(u'x_input/bias', '\n')
(u'x_input/Conv2D/ReadVariableOp', '\n')
(u'x_input/Conv2D', '\n')
(u'x_input/BiasAdd/ReadVariableOp', '\n')
(u'_input/BiasAdd', '\n')
(u'_input/Relu', '\n')
(u'max_pooling2d/MaxPool', '\n')
(u'conv2d/kernel', '\n')
(u'conv2d/bias', '\n')
(u'conv2d/Conv2D/ReadVariableOp', '\n')
(u'conv2d/Conv2D', '\n')
(u'conv2d/BiasAdd/ReadVariableOp', '\n')
(u'conv2d/BiasAdd', '\n')
(u'conv2d/Relu', '\n')
(u'conv2d/Relu_1/MaxPool', '\n')
(u'flatten/shape', '\n')
(u'flatten/strided_slice/stack', '\n')
(u'flatten/strided_slice/stack_1', '\n')
(u'flatten/strided_slice/stack_2', '\n')
(u'flatten/strided_slice', '\n')
(u'flatten/Reshape/shape/1', '\n')
(u'flatten/Reshape/shape', '\n')
(u'flatten/Reshape', '\n')
(u'dense/kernel', '\n')
(u'dense/bias', '\n')
(u'dense/MathMul/ReadVariableOp', '\n')
(u'dense/MathMul', '\n')
(u'dense/BiasAdd/ReadVariableOp', '\n')
(u'dense/BiasAdd', '\n')
(u'dense/Relu', '\n')
(u'dropout/Identity', '\n')
(u'y_out/kernel', '\n')
(u'y_out/bias', '\n')
(u'y_out/MathMul/ReadVariableOp', '\n')
(u'y_out/MathMul', '\n')
(u'y_out/BiasAdd/ReadVariableOp', '\n')
(u'y_out/BiasAdd', '\n')
(u'y_out/softmax', '\n')
#####
Convert completed
#####

```

5. 第二步，量化，注意把 2_vitisAI_tf_quantize.sh 中的 input_nodes、input_shapes、output_nodes 改成与第一步打印的节点名称一致。

```

#!/bin/bash

## 
# @Author: xiaobo
# @Email: 729527658@qq.com
# @Date: 2020-04-20
# @Description: quantize frozen tf model
# @Dependence: tensorflow 1.13, Vitis-AI Release 1.1
## 

# activate DECENT_Q Python3.6 virtual environment
#conda activate decent_q

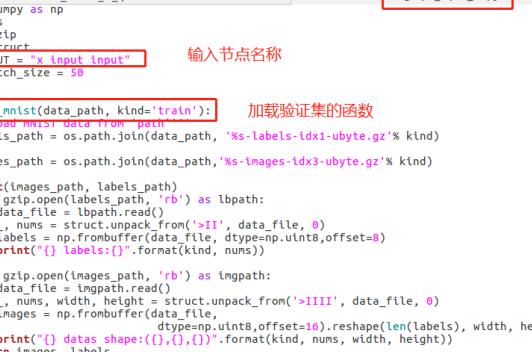
# generate calibraion images and list file
#python generate_images.py

# remove existing files
rm -rf ./quantize_results

# run quantization
echo #####
echo "Quantize begin"
echo "Vitis AI 1.1"
echo #####
decent_q quantize \
--input_frozen_graph ./frozen_result/model.pb \
--input_nodes x_input_input \
--input_shapes 7,28,28,1 \
--output_nodes y_out_Softmax \
--method 1 \
--input_fn graph_input_fn.calib_input \
--gpu 0 \
--calib_iter 100 \

```

然后根据读取验证集的方法修改 graph_input_fn.py，这个文件的作用是为 dnndk 的量化函数提供样本输入。



```
graph_input_fn.py (-/powersensor/ministNumber/dnnck) -gedit

Open [x] graph_input_fn.py (-/powersensor/ministNumber/dnnck) -gedit

import numpy as np
import os
import gzip
import struct
CONV_INPUT = "x {input input}"
calib_batch_size = 50

def load_mnist(data_path, kind='train'):
    labels_path = os.path.join(data_path, '%s-labels-idx1-ubyte.gz' % kind)
    images_path = os.path.join(data_path, '%s-images-idx3-ubyte.gz' % kind)

    print(images_path, labels_path)
    with gzip.open(labels_path, 'rb') as lbpath:
        data_file = lbpath.read()
        _, nums = struct.unpack_from('>II', data_file, 0)
        labels = np.frombuffer(data_file, dtype=np.uint8, offset=8)
        print("  {} labels:{}".format(kind, nums))

    with gzip.open(images_path, 'rb') as imgpath:
        data_file = imgpath.read()
        _, nums, width, height = struct.unpack_from('>IIII', data_file, 0)
        images = np.frombuffer(data_file,
                               dtype=np.uint8, offset=16).reshape(len(labels), width, height)
        print("  {} data shape:({},{},{})".format(kind, nums, width, height))
    return images, labels

dataset_path = './dataset_valid/minist-number/'
(test_images, test_labels) = load_mnist(dataset_path, 't10k')
test_images = test_images.reshape((-1, 28, 28, 1)) / 255.

输入节点名称          加载验证集的函数          验证集路径
```

然后键入./2_vitisAI_tf_quantize.sh 开始量化模型。

```
1 xiaobo@ubuntu:~/powersensor/fashion-minist/dnndk# ./2
2 _vitisAI_tf_quantize.sh
3 ****
4 Quantize begin
5 Vitis AI 1.1
6 ****
7 ('../dataset/fashion-mnist/t10k-images-idx3-ubyte.gz', '../dataset/
8   fashion-mnist/t10k-labels-idx1-ubyte.gz')
9 t10k labels:10000
10 t10k datas shape:(10000,28,28)
11 INFO: Checking Float Graph...
12 INFO: Float Graph Check Done.
13 INFO: Calibrating for 100 iterations...
14 100% |#
15 # ##### #
16 INFO: Calibration Done.
17 INFO: Generating Deploy Model...
18 INFO: Deploy Model Generated.
19 **** Quantization Summary ****
20 INFO: Output:
21 quantize_eval_model: ./quantize_results/quantize_eval_model.pb
22 deploy_model: ./quantize_results/deploy_model.pb
23 ****
24 QUANTIZATION COMPLETED
25 ****
```

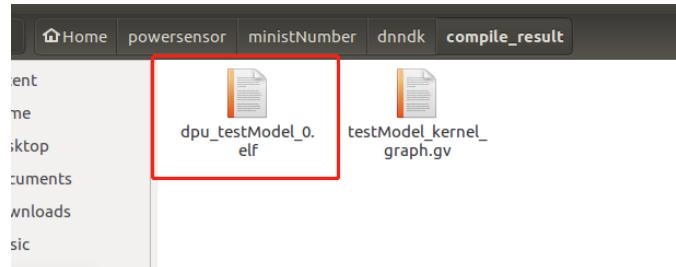
6. 第三步，编译模型

```
./3_vitisAI_tf_compile.sh
```

```
xiaobo@ubuntu:~/powersensor/ministNumber/dnndk$ ./3_vitisAI_tf_compile.sh
#####
COMPILE WITH DNNC begin
#####
[DNNC][Warning] layer [y_out_Softmax] (type: Softmax) is not supported in DPU, deploy it to CPU instead.
[DNNC][Warning] Fail to convert gv file to jpg because 'dot' is not installed in current system.
#####
DNNC Kernel topology "testModel_kernel_graph.jpg" for network "testModel"
DNNC kernel list info for network "testModel"
    Kernel ID : Name
        0 : testModel_0
        1 : testModel_1
    Kernel Name : testModel_0
    -----
    Kernel Type : DPUKernel
    Code Size : 6.53KB
    Param Size : 0.49MB
    Workload MACs : 8.48MOPS
    IO Memory Space : 0.01MB
    Mean Value : 0, 0, 0,
    Node Count : 4
    Tensor Count : 5
    Input Node(s)(H*W*c)
        x_input_Conv2D(0) : 28*28*1
    Output Node(s)(H*W*c)
        y_out_MatMul(0) : 1*1*10
    -----
    Kernel Name : testModel_1
    -----
    Kernel Type : CPUKernel
    Input Node(s)(H*W*c)
        y_out_Softmax : 1*1*10
    Output Node(s)(H*W*c)
        y_out_Softmax : 1*1*10
#####
COMPILATION COMPLETED
#####
```

记住这两个节点的名称和模型的名称，在 Powersensor 上的 edge 阶段就使用这两个名称来引用节点。

编译完成后会在当前目录的 compile_result 目录下生成 dpu_testModel_0.elf 文件，这个就是最终要在 DPU 上运行的二进制库，把它从虚拟机里拷贝出来备用：



第二十九章 AI 1.3 - 深度学习模型部署

内容提要

□ 介绍

□ 主要过程

29.1 介绍

本章主要介绍 edge 环节，即如何通过调度程序看，将编译好 elf 模型部署到 powersensor 上面运行。

本章需要使用新的 powersensor 镜像（2020.5 月以后发布的镜像会带有 DPU），下载地址：

链接：https://pan.baidu.com/s/18CFK2aXonxuFF6-L_ittdw

提取码：8ve4

本章的例程与上一节的相同，请移步

Powersensor AI 教程 1.1-数字识别-tf 模型训练下载。

29.2 主要过程

- 进入 powersensor 的 jupyter 文件管理页面，在/powersensor_workspace/powersensor_ai 下面新建 ministNumber 目录（随教程发布的案例已经准备好文件了）。在新建的 ministNumber 目录下新建 dataset_valid 和 edge 文件夹



- 通过 jupyter 的上传功能，把案例目录下的 edge 文件夹下的两个文件上传到 powersensor 的 edge 目录；验证集 dataset_valid 下的文件也同理上传到相应目录（可能需要新建 minist-number 文件夹）。
 - 如果要使用自己新训练的模型，可以把 edge 下的 elf 换成虚拟机里的 compileResult 下的 elf 文件。
 - 注意虚拟机里的文件不能直接上传（找不到），要先拷贝到自己的电脑里才能上传。
- 打开 powersensor 的 edge 下的 powersensor_ministNumer.ipynb 文件，按照 notebook 里面的指导逐个运行程序。

4. 首先也是加载头文件和重要的参数，其中 DPU 网络参数应该与 DPU 的编译结果输出保持一致，否则会导致 DPU 崩溃。

```

1   from dnndk import n2cube
2   import numpy as np
3   from numpy import float32
4   import os
5   import cv2
6   import matplotlib.pyplot as plt
7   import random
8   import time
9   import matplotlib as mpl
10  from matplotlib import font_manager
11  import PowerSensor as ps
12
13  mpl.rcParams['axes.unicode_minus']=False      # 正常显示负号
14
15  # 训练用的网络尺寸
16  img_size_net = 28
17
18  # 词汇表
19  wordlist = ['1', '1', '2', '3', '4', '5', '6', '7', '8', '9']
20
21  # DPU 网络参数
22  ELF_NAME = "dpu_testModel_0.elf"
23  CONV_INPUT_NODE = "x_input_Conv2D"
24  CONV_OUTPUT_NODE = "y_out_MatMul"
25

```

5. 测试集读取与测试，这个程序应该与 pc 环节的加载验证集一致。加载完会随机打印 10 个样本：

```

1  """
2  data_path: 数据集本地保存路径
3  kind: 数据集类别 (训练集: kind='train', 测试集: kind='t10k')
4  """
5
6  def load_mnist(data_path, kind='train'):
7      import os
8      import gzip
9      import struct
10     import numpy as np
11
12     """Load MNIST data from `path`"""
13     labels_path = os.path.join(data_path, '%s-labels-idx1-ubyte.gz' %
14                               kind)
15
16     images_path = os.path.join(data_path, '%s-images-idx3-ubyte.gz' %
17                               kind)

```

```

15
16     print(images_path, labels_path)
17     with gzip.open(labels_path, 'rb') as lbpath:
18         data_file = lbpath.read()
19         _, nums = struct.unpack_from('>II', data_file, 0) # 取前2个整
数, 返回一个元组
20         labels = np.frombuffer(data_file, dtype=np.uint8, offset=8)
21         print("{} labels:{}".format(kind, nums))
22
23     with gzip.open(images_path, 'rb') as imgpath:
24         data_file = imgpath.read()
25         _, nums, width, height = struct.unpack_from('>IIII', data_file,
0) # 取前4个整数, 返回一个元组
26         images = np.frombuffer(data_file,
27             dtype=np.uint8, offset=16).reshape(len(labels), width, height)
28         print("{} datas shape:({},{},{})".format(kind, nums, width,
height))
29     return images, labels
30
31 dataset_valid_path = '../dataset_valid//minist-number/'
32 # 1. 加载数据集
33 (validSet_images, validSet_labels) = load_mnist(dataset_valid_path,
't10k')
34
35 # 2. 图像预处理
36 validSet_images = validSet_images.reshape((-1, 28, 28, 1)) / 255.
37 validSet_images = np.array(validSet_images, dtype='float32')
38
39 # 3. 随机打印8个测试图像
40 fig, ax = plt.subplots(5, 2)
41 fig.set_size_inches(15, 15)
42 for i in range(5):
43     for j in range(2):
44         l = random.randint(0, len(validSet_labels))
45         ax[i, j].imshow(validSet_images[l, :, :, 0])
46         title = wordlist[validSet_labels[l]]
47         title_utf8 = title.decode('utf8')
48         ax[i, j].set_title(title_utf8)
49 plt.tight_layout()
50

```

6. 加载 DPU 模型，powersensor 在 xilinx 的 dnndk 的函数库上做了一个简单的封装，对于一些简单的分类任务已经够用，对于复杂的检测任务请使用 xilinx 的 dpu 的原生的 dnndk 函数。这里的 ELF_NAME 是 elf 文件的文件名，后面两个参数分布是网络的输入节点和输出节点，需要和 dnndk 的编译输出保持一致。

```
1 dpu1 = ps.DpuHelper()
```

```

2 dpu1.load_kernel(ELF_NAME, input_node_name=CONV_INPUT_NODE,
3 output_node_name=CONV_OUTPUT_NODE)

```

7. 测试验证集的精度：注意 dpu1.predit_softmax() 输入是一个 3 维的图像，比如灰度图的最后一个维度虽然是 1，但不能缺少。

```

1 tick_start = time.time()
2 test_num = len(validSet_lables)
3 right_eg_cnt = 0
4 for i in range(test_num):
5     img1_scale = validSet_images[i]
6     softmax = dpu1.predit_softmax(img1_scale)
7     pdt= np.argmax(softmax, axis=0)
8     if pdt == validSet_lables[i]:
9         right_eg_cnt += 1
10    tick_end = time.time()
11    print('精度: ' + str((right_eg_cnt*1.) / test_num))
12    print('测试 ' + str(test_num) + ' 个样本。耗时 ' + str(tick_end - tick_start) + ' 秒! ')
13

```

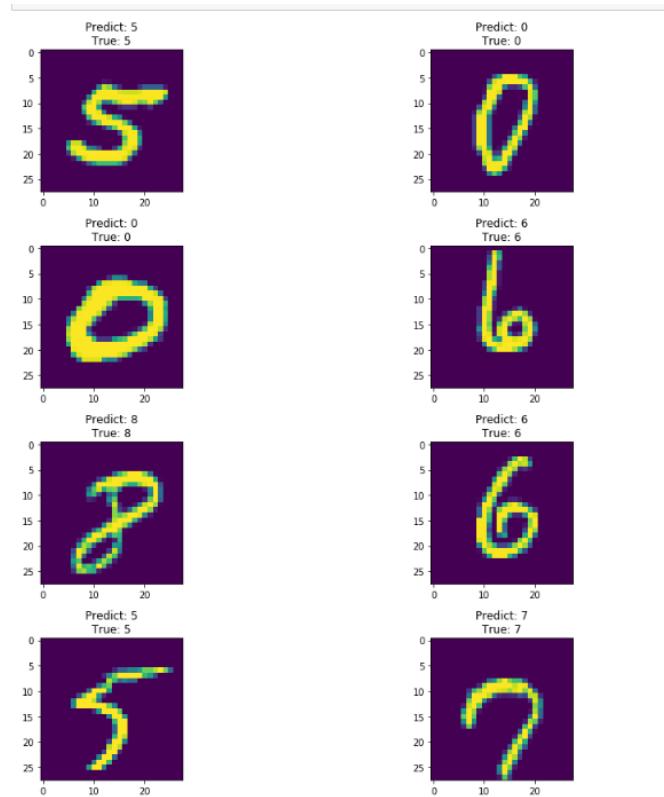
精度: 0.9933
测试 10000 个样本。耗时 35.3518271446秒!

8. 随机测试一些样本并打印结果：

```

1 fig, ax = plt.subplots(5, 2)
2 fig.set_size_inches(15,15)
3 for i in range(5):
4     for j in range(2):
5         l = random.randint(0, len(validSet_lables))
6         img1_scale = validSet_images[l]
7         softmax = dpu1.predit_softmax(img1_scale)
8         pdt= np.argmax(softmax, axis=0)
9         ax[i, j].imshow(validSet_images[l, :, :, 0])
10        # title = "预测: " + str(wordlist[pdt]) + "\n" + "真实:
11        # " + str(wordlist[test_labels[l]])
12        title = "Predict: " + wordlist[pdt] + "\n" + "True: " + wordlist
13        [validSet_lables[l]]
14        title_utf8 = title.decode('utf8')
15        ax[i, j].set_title(title_utf8)
16 plt.tight_layout()

```



9. 使用 power Sensor 上的摄像头进行实际拍摄，并识别拍摄图片中的数字

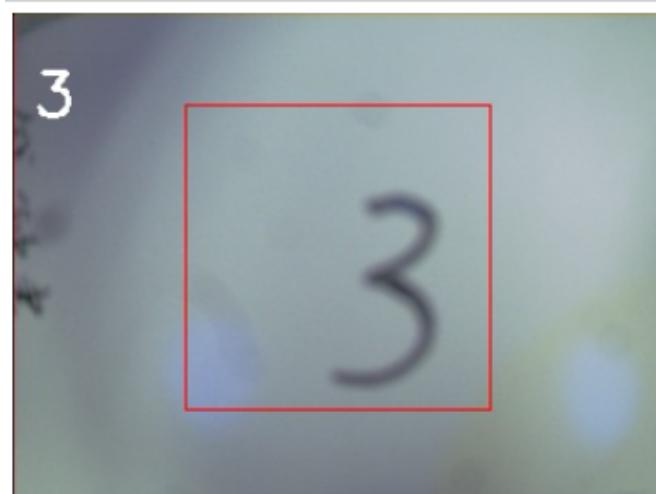
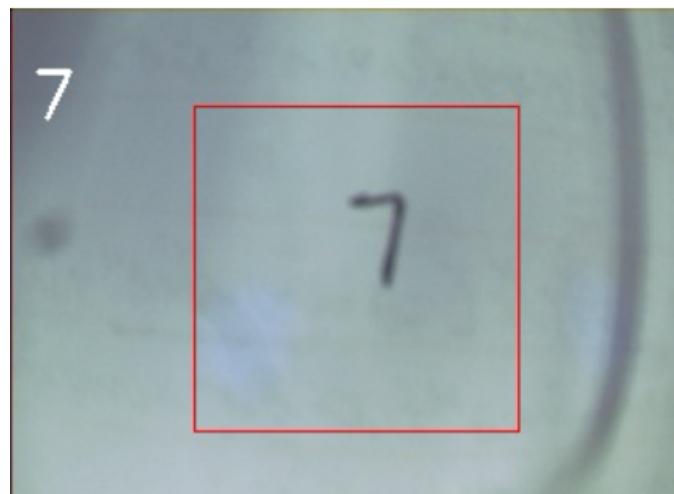
```

1  cam1 = ps.ImageSensor()
2  for i in range(200):
3      # 记录时间
4      start = time.time()
5      # 清空显示区
6      clear_output(wait=True)
7      # 读取图像
8      imgMat = cam1.read_img_ori()
9      imgShow = cv2.resize(imgMat, (320,240))
10     #     imgMat_cali = grey_world2(imgMat)
11     # 图像缩放，太大的图像显示非常浪费资源
12     imgCut = imgMat[90:390, 250:550, :]
13     tempImg = cv2.resize(imgCut, (28,28))
14     grayImg = cv2.cvtColor(tempImg, cv2.COLOR_BGR2GRAY)
15     ret,img_binary = cv2.threshold(grayImg, 100, 255, cv2.
16         THRESH_BINARY_INV) # 全局二值化
17     grayImg = img_binary[:, :, np.newaxis]
18     img_scale = grayImg / 255.
19     img_scale = np.array(img_scale, dtype=np.float32)
20     softmax = dpu1.predict_softmax(img_scale)
21     pdt= np.argmax(softmax, axis=0)
#         cv2.putText(imgShow,wordlist[pdt], (10,10), fontHeight=30,
color=(255,255,255), thickness=-1, line_type=cv2.LINE_4,
bottomLeftOrigin=False)

```

```
22     font=cv2.FONT_HERSHEY_SIMPLEX
23     cv2.putText(imgShow,wordlist[pdt],(10,50), font, 1,(255,255,255)
24 ,2)
25     cv2.rectangle(imgShow,(85,45),(235,195),(0,0,255),1)
26     #      tempImg = imgMat
27     # 显示图像
28     img = ps.CommonFunction.show_img_jupyter(imgShow)
29     img = ps.CommonFunction.show_img_jupyter(grayImg)
30     #      img = ps.CommonFunction.show_img_jupyter(grayImg)
31     # 记录运行时间
32     end = time.time()
33     # 打印运行时间
34     #      print(pdt)
35     #      print(end - start)
36     # 因为网络传输的延时，需要稍息一下
37     time.sleep(0.1)
```

实验结果：



第三十章 AI 2 - 石头剪刀布 - 彩色数据预处理

内容提要

视频教程

图文教程

 **笔记** 案例资料包的下载地址请见[27.2。](#)

30.1 介绍

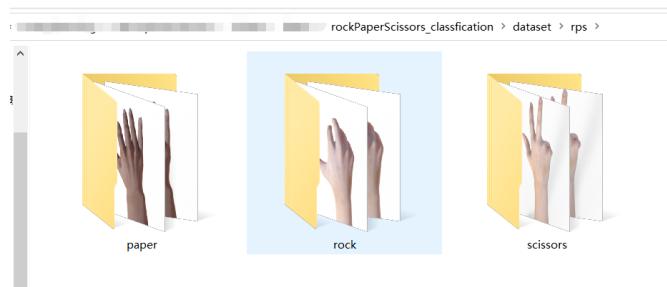
上一期的 minist 案例是为了让大家熟悉一下 powersensor ai 的整个流程，使用的数据集是 google 提供好的 minist 数据集，是灰度的、尺寸统一的。然而，实际的识别任务往往需要从不同的来源获取不同尺寸的原始图像，且一般是彩色。为了解决这个问题，本章主要介绍如何对收集的数据进行预处理，以及生成适合 tensorflow 训练的训练集，以及如何调用 powersensor 获取彩色的图片并进行识别。

这次的案例是一个大家熟知的小游戏，“石头剪刀布。。”，这个布要拉长音。

30.2 PC 训练模型

* 数据集准备与预处理

本节使用数据集是 google 的石头剪刀布数据集，已经包含在我们的百度网盘下载包里。下载解压完在 dataset 目录下，可以看到：



除了 minist 这种封装好的数据集，更普遍的机器学习数据集（如本节的数据集）一般是以原图像文件分文件夹存放的形式提供，这样有两个优势，一个是便于压缩和分享，避免传输大体积文件；另一个是用户较容易添加自己的数据集，以及便于使用网络上下载的不同来源数据集的合并。但是，它也存在问题，深度网络训练时使用的数据集输入往往是尺寸一致并经过标准化、归一化操作的。因此，我们在获取完数据集后，需要对数据集进行预处理，主要解决以下问题

原图像的存在分辨率的差异

图像的标签和样本没有有机绑定（往往是一个标签的图片放一个文件夹）

图像没有随机排列，不利于训练

为了解决上述问题，我们使用 tensorflow 的 tf recorder 来完成数据集的准备和预处理工作。

- 首先，包含一些重要的头文件和定义参数：

```

1 import cv2
2 import numpy as np
3 import os
4 import tensorflow as tf
5 from tensorflow import keras
6 import random
7 import time
8 import matplotlib.pyplot as plt
9 from matplotlib.font_manager import FontProperties
10
11 # 有GPU才能打印
12 # gpus = tf.config.experimental.list_physical_devices(device_type='GPU')
13 # cpus = tf.config.experimental.list_physical_devices(device_type='CPU')
14 # print(gpus, cpus)
15 # for gpu in gpus:
16 #     tf.config.experimental.set_memory_growth(gpu, True)
17
18 # 让Matplotlib正确显示中文
19 import matplotlib as mpl
20 mpl.style.use('seaborn')
21 # mpl.rcParams['font.sans-serif']=['SimHei']      # 用黑体显示中文
22 mpl.rcParams['axes.unicode_minus']=False          # 正常显示负号
23 font = FontProperties(fname="/home/powersensor/Documents/simhei.ttf")
24
25 tf.enable_eager_execution()
26
27 # 训练用的图像尺寸
28 img_size_net = 128
29 # 训练的batch大小
30 batch_size = 32
31 # 数据库路径
32 dataset_path = '../dataset/'
33 # 存放过程和结果的路径
34 run_path = './run/'
35 if not os.path.exists(run_path):
36     os.mkdir(run_path)
37 wordlist = ['布', '石头', '剪刀']
38 sorts_pathes = ['paper', 'rock', 'scissors']
39
40 # 存放转换后的tf数据集的路径
dataset_tf_path_train = run_path + 'datasetTfTrain.tfrecords'
```

```

41 dataset_tf_path_test = run_path + 'datasetTfTest.tfrecords'
42
43 # # 存放转换后的tf数据集的路径
44 # dataset_tf_path = run_path + 'flowersTf.tfrecords'
45 dataset_nums = 2520
46 testSet_nums = 372
47

```

2. 使用 tf-recorder 生成数据集

```

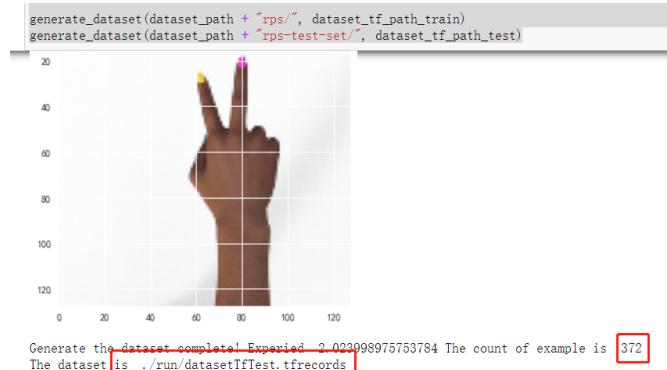
1 def generate_dataset(raw_data_path, dataset_path):
2     tick_begin = time.time()
3     img_cnt = int(0)
4     label_cnt = int(0)
5     with tf.io.TFRecordWriter(dataset_path) as writer:
6         for sort_path in sorts_pathes:
7             exp_list = os.listdir(raw_data_path + sort_path)
8             for img_name in exp_list:
9                 img_path = raw_data_path + sort_path + "/" + img_name
10                img = cv2.imread(img_path)
11                img_scale = cv2.resize(img, (img_size_net, img_size_net),
12                                       interpolation = cv2.INTER_CUBIC)
13                if not img is None:
14                    feature = {
15                        'img1':tf.train.Feature(bytes_list=tf.train.BytesList(
16                            value=[img_scale.tostring()])),
17                        'label':tf.train.Feature(int64_list=tf.train.Int64List(
18                            value=[label_cnt]))
19                        #                     'label':tf.train.Feature(int64_list=tf.
20                        train.Int64List(value=[label_cnt]))
21                    }
22                    example = tf.train.Example(features=tf.train.Features(
23                        feature=feature))
24                    writer.write(example.SerializeToString())
25                    # 每隔50张打印一张图片
26                    if img_cnt % 100 == 0:
27                        print('The ', str(img_cnt), ' image')
28                        plt.imshow(cv2.cvtColor(img_scale, cv2.COLOR_BGR2RGB))
29                        plt.show()
30                        img_cnt += 1
31                    label_cnt = label_cnt + 1
32                writer.close()
33                tick_end = time.time()
34                print('Generate the dataset complete! Experied ', str(tick_end -
35 tick_begin), 'The count of example is ', str(img_cnt))
36                print('The dataset is ', dataset_path)
37
38 generate_dataset(dataset_path + "rps/", dataset_tf_path_train)

```

```

33     generate_dataset(dataset_path + "rps-test-set/",
34                         dataset_tf_path_test)

```



生成完会打印数据集的数量、数据集的名称。

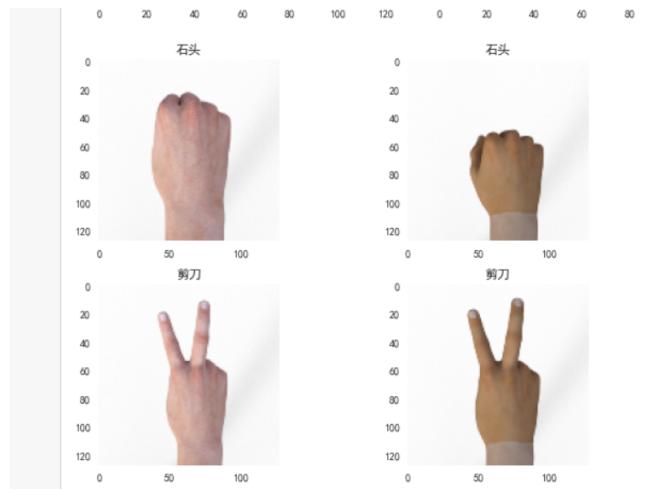
- 读取和测试数据集，生成数据集只需要运行一次就会自动保存到相应的目录。读取数据集在每次重启 jupyter 的时候都需要加载一次。

```

1 def read_and_decode(example_proto):
2     """
3         从TFrecord格式文件中读取数据
4
5     """
6     image_feature_description = {
7         'img1':tf.io.FixedLenFeature([],tf.string),
8         'label':tf.io.FixedLenFeature([1], tf.int64),
9     }
10    feature_dict = tf.io.parse_single_example(example_proto,
11                                              image_feature_description)
12    img1 = tf.io.decode_raw(feature_dict['img1'], tf.uint8)
13    label = feature_dict['label']
14    return img1, label
15
16    dataset_train = tf.data.TFRecordDataset(dataset_tf_path_train)
17    dataset_train = dataset_train.map(read_and_decode)
18    dataset_test = tf.data.TFRecordDataset(dataset_tf_path_test)
19    dataset_test = dataset_test.map(read_and_decode)
20
21
22    # 2. 随机打印8个训练集测试图像
23    dataset = dataset_train.shuffle(buffer_size=dataset_nums)
24    dataSet = [x1 for x1 in dataset.take(10)]
25    dataSet_img = np.array([x1[0].numpy() for x1 in dataSet])
26    dataSet_img = dataSet_img.reshape((-1,img_size_net,img_size_net, 3))
27    / ((np.float32)(255.))
28    dataSet_label = np.array([x1[1].numpy()[0] for x1 in dataSet])
29
30    fig, ax = plt.subplots(3, 2)

```

```
27     fig.set_size_inches(9,15)
28     l = 0
29     for i in range(3):
30         for j in range(2):
31             ax[i, j].imshow(cv2.cvtColor(dataSet_img[1], cv2.COLOR_BGR2RGB))
32             #         ax[i, j].set_title(wordlist[dataSet_label[1]])
33             title = wordlist[dataSet_label[1]]
34             title_utf8 = title.decode('utf8')
35             ax[i, j].set_title(title_utf8, fontproperties=font)
36             ax[i, j].grid(False)
37             l += 1
38     plt.tight_layout()
39
40 # 2. 随机打印8个测试集测试图像
41 dataset = dataset_test.shuffle(buffer_size=testSet_nums)
42 dataSet = [x1 for x1 in dataset.take(10)]
43 dataSet_img = np.array([x1[0].numpy() for x1 in dataSet])
44 dataSet_img = dataSet_img.reshape((-1,img_size_net,img_size_net, 3))
45 / ((np.float32)(255.))
46 dataSet_label = np.array([x1[1].numpy()[0] for x1 in dataSet])
47
48 fig, ax = plt.subplots(2, 2)
49 fig.set_size_inches(9,6)
50 l = 0
51 for i in range(2):
52     for j in range(2):
53         ax[i, j].imshow(cv2.cvtColor(dataSet_img[l], cv2.COLOR_BGR2RGB))
54         #         ax[i, j].set_title(wordlist[dataSet_label[l]],
55         fontproperties=font)
56         title = wordlist[dataSet_label[l]]
57         title_utf8 = title.decode('utf8')
58         ax[i, j].set_title(title_utf8, fontproperties=font)
59         ax[i, j].grid(False)
60         l += 1
61 plt.tight_layout()
```



30.3 模型的训练和保存

1. 训练集和测试集的打乱和重排，打乱和重排数据集可以大大减小训练不稳定的风险。

```
1 num_train_eg = dataset_nums
2 num_test_eg = testSet_nums
3 # 1. 打乱数据集
4 dataset_train = dataset_train.shuffle(buffer_size=num_train_eg)
5 # 3. 分离训练集和测试集
6 trainSet = [x1 for x1 in dataset_train.take(dataset_nums)]
7 trainSet_img = np.array([x1[0].numpy() for x1 in trainSet])
8 trainSet_img = trainSet_img.reshape((-1, img_size_net, img_size_net,
9 3)) / ((np.float32)(255.))
10 trainSet_label = np.array([x1[1].numpy()[0] for x1 in trainSet])
11
12 testSet = [x1 for x1 in dataset_test.take(num_test_eg)]
13 testSet_img = np.array([x1[0].numpy() for x1 in testSet])
14 testSet_img = testSet_img.reshape((-1, img_size_net, img_size_net, 3))
15 / ((np.float32)(255.))
16 testSet_label = np.array([x1[1].numpy()[0] for x1 in testSet])
```

2. 深度学习网络模型设计，这次的模型比 minist 的稍微复杂一丢丢哈，也就 10 层：

```
1 model = keras.Sequential([
2     #keras.layers.Flatten(input_shape=(128, 128, 3)),
3     keras.layers.Conv2D(32, (3,3), padding="same", input_shape=(
4         img_size_net, img_size_net, 3), name='x_input', activation=tf.nn.
5         relu),
6     #    keras.layers.BatchNormalization(),
7     #    keras.layers.Activation('relu'),
8     keras.layers.MaxPooling2D(pool_size=(2,2)),
```

```

7     keras.layers.Conv2D(64, (3,3), padding="same", activation=tf.nn.
8         relu),
9     keras.layers.MaxPooling2D(pool_size=(2,2)),
10    keras.layers.Conv2D(128, (3,3), padding="same", activation=tf.nn.
11        relu),
12    keras.layers.MaxPooling2D(pool_size=(2,2)),
13    keras.layers.Conv2D(128, (3,3), padding="same", activation=tf.nn.
14        relu),
15    keras.layers.MaxPooling2D(pool_size=(2,2)),
16    keras.layers.Flatten(),
17    keras.layers.Dense(30, activation=tf.nn.relu),
18    keras.layers.Dropout(0.1),
19    # 最后一个层决定输出类别的数量
20    keras.layers.Dense(3, activation=tf.nn.softmax, name='y_out')
21)
22 model.compile(optimizer=tf.train.RMSPropOptimizer(0.001),
23                 loss='sparse_categorical_crossentropy',
24                 metrics=['accuracy'])
25 model.summary()

```

3. 神经网络训练，先用较大的学习率：

```

1 tick_start = time.time()
2 history = model.fit(trainSet_img, trainSet_label, batch_size=
3 batch_size, epochs=15, validation_data=(testSet_img, testSet_label))
4 tick_end = time.time()
5 print("Tring completed. Experied ", str(tick_end - tick_start))

```



4. 减小学习率，再训练一会儿

```

1 from tensorflow.keras.optimizers import SGD
2 model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss=tf.keras.
3     losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
4 tick_start = time.time()
5 history = model.fit(trainSet_img, trainSet_label, batch_size=
6     batch_size, epochs=30, validation_data=(testSet_img, testSet_label))
7 tick_end = time.time()

```

```

6   print("Tring completed. Experied ", str(tick_end - tick_start))
7

```

别看精度没有提高，只要误差下降就能提升性能。

5. 测试训练效果

```

1 # 使用tensorflow的函数评估精度
2 res = model.evaluate(testSet_img, testSet_label)
3 print('test set: ', res)
4 res = model.evaluate(trainSet_img, trainSet_label)
5 print('train set: ', res)
6
7 =====
8 # 372/372 [=====] - 0s 445us/sample - # loss:
9     0.4589 - accuracy: 0.9355
10    # test set: [0.45888313828211436, 0.9354839]
11    # 2520/2520 [=====] - 1s 398us/sample - #
12     loss: 4.8881e-06 - accuracy: 1.0000
13    # train set: [4.888053711369814e-06, 1.0]
14
15

```

6. 保存模型

```

1 model.save_weights(run_path + "model_weight.h5")
2 json_config = model.to_json()
3 with open(run_path + 'model_config.json', 'w') as json_file:
4     json_file.write(json_config)
5

```

7. 模型转换

```

1 # 将模型和权值文件复制到dnndk文件夹中
2 !cp run/model_config.json run/model_weight.h5 ..dnndk
3 ! chmod +x ..dnndk/1_vitisAI_keras2frozon.sh
4 ! chmod +x ..dnndk/2_vitisAI_tf_quantize.sh
5 ! chmod +x ..dnndk/3_vitisAI_tf_compile.sh
6 # 把权值固化到模型中
7 !cd '../dnndk';pwd;../dnndk/1_vitisAI_keras2frozon.sh
8 # 量化
9 !cd '../dnndk';pwd;../dnndk/2_vitisAI_tf_quantize.sh
10 # 编译
11 !cd '../dnndk';pwd;../dnndk/3_vitisAI_tf_compile.sh
12

```

30.4 edge 调用

- 进入 powersensor 的 jupyter 文件管理页面，在/powersensor_workspace/powersensor_ai 下面新建 ministNumber 目录（随教程发布的案例已经准备好文件了）。在新建的

ministNumber 目录下新建 dataset_valid 和 edge 文件夹



2. 通过 jupyter 的上传功能，把案例目录下的 edge 文件夹下的两个文件上传到 powersensor 的 edge 目录；验证集 dataset_valid 下的文件也同理上传到相应目录（可能需要新建 paper、rock、scissors 文件夹）。
 - ★如果要使用自己新训练的模型，可以把 edge 下的 elf 换成虚拟机里的 compileResult 下的 elf 文件。
 - ★注意虚拟机里的文件不能直接上传（找不到），要先拷贝到自己的电脑里才能上传。
3. 打开 powersensor 的 edge 下的 powersensor_ministNumer.ipynb 文件，按照 notebook 里面的指导逐个运行程序。
4. 首先也是加载头文件和重要的参数，其中 DPU 网络参数应该与 DPU 的编译结果输出保持一致，否则会导致 DPU 崩溃。

```

1  from dnndk import n2cube
2  import numpy as np
3  from numpy import float32
4  import os
5  import cv2
6  import matplotlib.pyplot as plt
7  import random
8  import time
9  import matplotlib as mpl
10 from matplotlib import font_manager
11 import PowerSensor as ps
12 from IPython.display import clear_output
13
14 mpl.rcParams['axes.unicode_minus']=False      # 正常显示负号
15 font = font_manager.FontProperties(fname="/usr/share/fonts/truetype/
droid/DroidSansFallbackFull.ttf")
16
17 wordlist = ['布', '石头', '剪刀']
18
19 # DPU 网络 参数
20 # KERNEL_CONV="testModel"
21 ELF_NAME = "dpu_testModel_0.elf"
22 CONV_INPUT_NODE = "x_input_Conv2D"
23 CONV_OUTPUT_NODE = "y_out_MatMul"
24

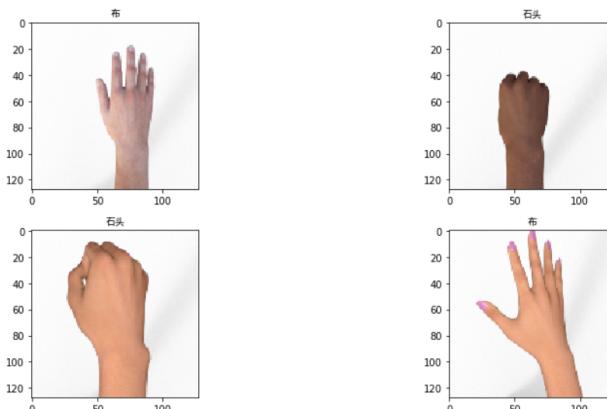
```

5. 测试集读取与测试

```

1 dataset_path = '../dataset_valid/'
2 (validSet_images, validSet_labels) = load_valid_data(dataset_path)
3
4 # 2. 图像预处理
5 # test_images = test_images.reshape((-1,28,28,1)) / 255.
6 validSet_images = np.array(validSet_images, dtype='float32')
7
8 # 3. 随机打印8个测试图像
9 fig, ax = plt.subplots(5, 2)
10 fig.set_size_inches(15,15)
11 for i in range(5):
12     for j in range(2):
13         l = random.randint(0, len(validSet_labels))
14         ax[i, j].imshow(cv2.cvtColor(validSet_images[l], cv2.COLOR_BGR2RGB))
15         title = wordlist[validSet_labels[l]]
16         title_utf8 = title.decode('utf8')
17         ax[i, j].set_title(title_utf8, fontproperties=font)
18 plt.tight_layout()

```



6. 加载 DPU

```

1 dpu1 = ps.DpuHelper()
2 dpu1.load_kernel(ELF_NAME, input_node_name=CONV_INPUT_NODE,
3                  output_node_name=CONV_OUTPUT_NODE)

```

7. 测试验证集

```

1 tick_start = time.time()
2 test_num = len(validSet_labels)
3 right_eg_cnt = 0
4 for i in range(test_num):
5     img1_scale = validSet_images[i]
6     softmax = dpu1.predict_softmax(img1_scale)
7     pdt= np.argmax(softmax, axis=0)

```

```

8     if pdt == validSet_labels[i]:
9         right_eg_cnt += 1
10    tick_end = time.time()
11    print('精度：' + str((right_eg_cnt*1.) / test_num))
12    print('测试 ' + str(test_num) + ' 个样本。耗时 ' + str(tick_end -
13      tick_start) + ' 秒！')
14
15    ######
16    # 精度：1.0
17    # 测试 30 个样本。耗时 5.25601506233 秒！

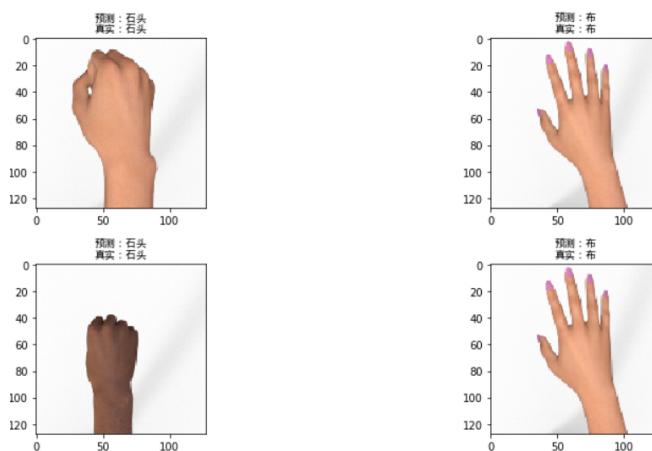
```

8. 随机样本测试

```

1 fig, ax = plt.subplots(5, 2)
2 fig.set_size_inches(15,15)
3 for i in range(5):
4     for j in range(2):
5         l = random.randint(0, len(validSet_labels)-1)
6         img1_scale = validSet_images[l]
7         softmax = dpu1.predit_softmax(img1_scale)
8         pdt= np.argmax(softmax, axis=0)
9         ax[i, j].imshow(cv2.cvtColor(validSet_images[l], cv2.COLOR_BGR2RGB))
10        # title = "预测：" + str(wordlist[pdt]) + "\n" + "真实：" +
11        # str(wordlist[test_labels[l]])
11        title = "预测：" + wordlist[pdt] + "\n" + "真实：" + wordlist[
12          validSet_labels[l]]
12        title_utf8 = title.decode('utf8')
13        ax[i, j].set_title(title_utf8, fontproperties=font)
14 plt.tight_layout()

```



9. 加载相机对象

```

1 # 这个对象用于操作摄像头
2 cam1 = ps.ImageSensor()

```

```

3 wordlist = ['p', 'r', 's']
4 # cv_font = cv2.freetype.createFreeType2()
5 # cv_font.loadFontData(fontFileName='/usr/share/fonts/truetype/droid/
DroidSansFallbackFull.ttf', id=0)

```

10. 相机读取图像并识别测试

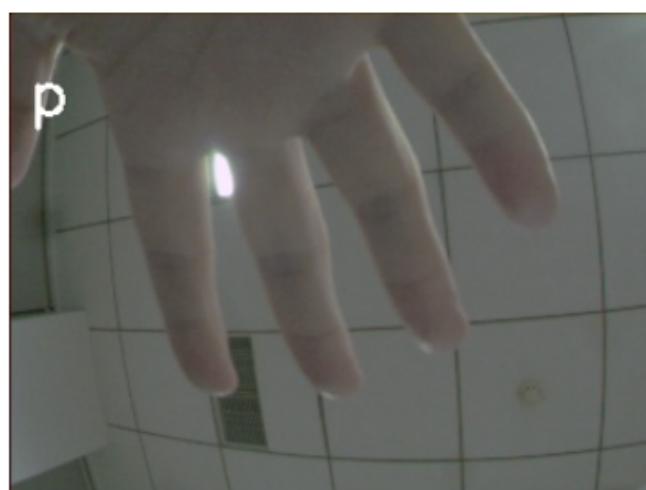
```

1 for i in range(100):
2     # 记录时间
3     start = time.time()
4     # 清空显示区
5     clear_output(wait=True)
6     # 读取图像
7     imgMat = cam1.read_img_ori()
8     imgShow = cv2.resize(imgMat, (320,240))
9     #     imgMat_cali = grey_world2(imgMat)
10    # 图像缩放，太大的图像显示非常浪费资源
11    tempImg = cv2.resize(imgMat, (128,128))
12    img_scale = tempImg / 255.
13    img_scale = np.array(img_scale, dtype=np.float32)
14    softmax = dpu1.predit_softmax(img_scale)
15    pdt= np.argmax(softmax, axis=0)
16    #     title = wordlist[pdt]
17    #     title_utf8 = title.decode('utf8')
18    font=cv2.FONT_HERSHEY_SIMPLEX
19    cv2.putText(imgShow,wordlist[pdt],(10,50), font, 1,(255,255,255),2)
20    #     cv_font.putText(imgShow,wordlist[pdt], (10,10), fontHeight=30,
color=(0,0,0), thickness=-1, line_type=cv2.LINE_4, bottomLeftOrigin=False)
21    #     font=cv2.FONT_HERSHEY_SIMPLEX
22    #     tempImg = imgMat
23    # 显示图像
24    img = ps.CommonFunction.show_img_jupyter(imgShow)
25    # 记录运行时间
26    end = time.time()
27    # 打印运行时间
28    print(end - start)
29    # 因为网络传播的延时，需要稍息一下
30    time.sleep(0.1)

```



0.26219201088



0.262503147125



0.26589012146

30.5 本章小结

本章主要通过石头剪刀布的例子，让大家了解如何处理比较通用的数据集，以及进一步熟悉 dpu 的操作流程。

第三十一章 AI 3 - 五花分类-从现有模型迁移训练

内容提要

- 图文教程
- DNNDK 编译
- PC 训练模型
- EDGE 调用

31.1 介绍

 **笔记** 案例资料包的下载地址请见[27.2](#)。

对于一些简单的分类问题，如 PowerSensor AI 1 中的 ministFashion，随手设计一个简单的深度网络就可以得到较好的分类效果。然而，这样的手法对于复杂一些的数据集很难保证效果。对于复杂的分类问题，使用已发布的精心设计的神经网络作为基础网络是更明智的选择。同时，从头开始训练深度的神经网络是一个繁重的工作，使用别人在 imagenet 上预训练好的权值来初始化我们的网络可以起到事半功倍的效果。本节我们将以五花分类问题（虽然这个问题不见得多复杂）为例，给大家介绍如何把 tensorflow2.0 官方训练好的 InceptionV3，通过简单的几步操作，迁移到我们自己的数据集上使用，并在 powersensor 上验证运行。

经过上一讲的学习，我们已经知道 DPU 的应用开发流程分为 PC 模型训练、DNNDK 模型编译、EDGE 模型部署，三个步骤，本章剩下的内容将逐个介绍。

31.2 PC 训练模型

1. PC 训练模型

首先我们需要包含一些重要的头文件和定义重要的参数

```
1 import cv2
2 import numpy as np
3 import os
4 import tensorflow as tf
5 from tensorflow import keras
6 import random
7 import time
8 import matplotlib.pyplot as plt
9 # gpus = tf.config.experimental.list_physical_devices(device_type='GPU')
10 # cpus = tf.config.experimental.list_physical_devices(device_type='CPU')
11 # print(gpus, cpus)
12 # for gpu in gpus:
13 #     tf.config.experimental.set_memory_growth(gpu, True)
```

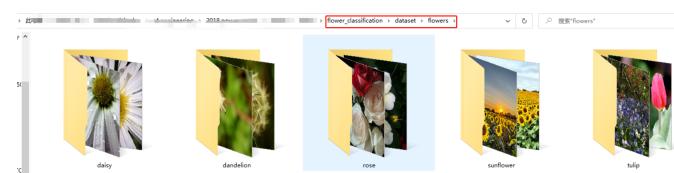
```

14 # 让 Matplotlib 正确显示中文
15 import matplotlib as mpl
16 mpl.style.use('seaborn')
17 mpl.rcParams['font.sans-serif']=['SimHei']      # 用黑体显示中文
18 mpl.rcParams['axes.unicode_minus']=False        # 正常显示负号
19
20 # 训练用的图像尺寸
21 img_size_net = 128
22 # 训练的batch大小
23 batch_size = 32
24 # 数据库路径
25 dataset_path = '../dataset/'
26 # 各个花的路径
27 flower_pathes = ['flowers/daisy', 'flowers/dandelion', 'flowers/rose',
28 ', 'flowers/sunflower', 'flowers/tulip']
29 wordlist = ['雏菊', '蒲公英', '玫瑰', '向日葵', '郁金香']
30 # 存放过程和结构的路径
31 run_path = './run/'
32 if not os.path.exists(run_path):
33     os.mkdir(run_path)
34 # 存放转换后的tf数据集的路径
35 dataset_tf_path = run_path + 'flowersTf.tfrecords'
36 dataset_nums = 4300

```

* 数据集准备与预处理

本节使用数据集是五花（玫瑰、向日葵、蒲公英、郁金香、雏菊）数据集，可以到我们的百度网盘数据集文件夹里下载到。下载完解压到 flower_classification 的 dataset 目录下，如下图所示：



在数据集预处理上，我们使用了上一章“石头剪刀布”识别的方案，先把原图像处理后转换为 tf-recorder，然后再使用 tfrecoeder 完成样本训练。

```

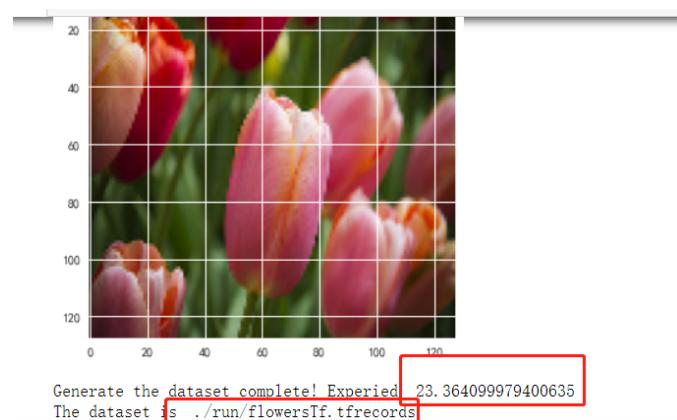
1 tick_begin = time.time()
2 img_cnt = int(0)
3 label_cnt = int(0)
4 with tf.io.TFRecordWriter(dataset_tf_path) as writer:
5     for sort_path in flower_pathes:
6         flower_list = os.listdir(dataset_path + sort_path)
7         for img_name in flower_list:
8             img_path = dataset_path + sort_path + "/" + img_name
9             img = cv2.imread(img_path)

```

```

10         img_scale = cv2.resize(img,(img_size_net, img_size_net),
11                         interpolation = cv2.INTER_CUBIC)
12         if not img is None:
13             feature = {
14                 'img1':tf.train.Feature(bytes_list=tf.train.BytesList(
15                     value=[img_scale.tostring()])),
16                 'label':tf.train.Feature(int64_list=tf.train.Int64List(
17                     value=[label_cnt]))
18             }
19             example = tf.train.Example(features=tf.train.Features(
20                 feature=feature))
21             writer.write(example.SerializeToString())
22             # 每隔50张打印一张图片
23             if img_cnt % 100 == 0:
24                 print('The ', str(img_cnt), ' image')
25                 plt.imshow(cv2.cvtColor(img_scale, cv2.COLOR_BGR2RGB))
26                 plt.show()
27                 img_cnt += 1
28             label_cnt = label_cnt + 1
29             writer.close()
30             tick_end = time.time()
31             print('Generate the dataset complete! Experied ', str(tick_end -
32             tick_begin))
33             print('The dataset is ', dataset_tf_path)
34

```



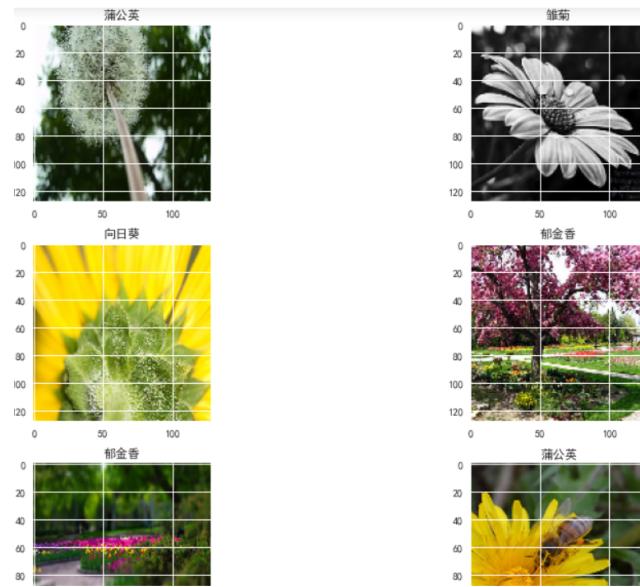
读取和测试数据集，生成数据集只需要运行一次就会自动保存到相应的目录。读取数据集在每次重启 jupyter 的时候都需要加载一次。

```

1 def read_and_decode(example_proto):
2     ...
3     从TFrecord格式文件中读取数据
4

```

```
5
6     image_feature_description = {
7         'img1':tf.io.FixedLenFeature([],tf.string),
8         'label':tf.io.FixedLenFeature([1], tf.int64),
9     }
10    feature_dict = tf.io.parse_single_example(example_proto,
11        image_feature_description)
12    img1 = tf.io.decode_raw(feature_dict['img1'], tf.uint8)
13    label = feature_dict['label']
14    return img1, label
15
16 # 1. 读取数据集
17 dataset = tf.data.TFRecordDataset(dataset_tf_path)
18 dataset = dataset.map(read_and_decode)
19
20 # 2. 随机打印8个测试图像
21 dataset = dataset.shuffle(buffer_size=dataset_nums)
22 dataSet = np.array([x1 for x1 in dataset.take(10)])
23 dataSet_img = np.array([x1[0].numpy() for x1 in dataSet])
24 dataSet_img = dataSet_img.reshape((-1,img_size_net,img_size_net, 3))
25 / ((np.float32)(255.))
26 dataSet_label = np.array([x1[1].numpy()[0] for x1 in dataSet])
27 fig, ax = plt.subplots(5, 2)
28 fig.set_size_inches(15,15)
29 l = 0
30 for i in range(5):
31     for j in range(2):
32         ax[i, j].imshow(cv2.cvtColor(dataSet_img[l], cv2.
33 COLOR_BGR2RGB))
34         title = wordlist[dataSet_label[l]]
35         title_utf8 = title.decode('utf8')
36         ax[i, j].set_title(title_utf8, fontproperties=font)
37         ax[i, j].grid(False)
38         l += 1
39 plt.tight_layout()
```



31.2.1 网络设计与预训练

1. 训练集和测试集的打乱和重排，打乱和重排数据集可以大大减小训练不稳定的风

险。

```

1 # 1. 打乱数据集
2 dataset = dataset.shuffle(buffer_size=dataset_nums)
3
4 # 2. 抓取数据集
5 dataSet = np.array([x1 for x1 in dataset.take(dataset_nums)])
6 dataSet_img = np.array([x1[0].numpy() for x1 in dataSet])
7 dataSet_img = dataSet_img.reshape((-1, img_size_net, img_size_net, 3))
8 / ((np.float32) (255.))
9 dataSet_label = np.array([x1[1].numpy()[0] for x1 in dataSet])
10
11 # 3. 分离训练集和测试集
12 trainSet_num = int(0.75 * dataset_nums)
13 trainSet_img = dataSet_img[0 : trainSet_num, :, :, :]
14 testSet_img = dataSet_img[trainSet_num : , :, :, :]
15 trainSet_label = dataSet_label[0 : trainSet_num]
16 testSet_label = dataSet_label[trainSet_num : ]
17
18 # 3. 统计各种训练集中各种样本的数量
19 print('数据集中各个样本的数量：')
20 l = []
21 for x in dataSet_label:
22     l.append(wordlist[x])
23 plt.hist(l, rwidth=0.5)
24 plt.show()

```

2. 网络设计，对于 minist 那种简单的数据集，我们可以随手搭一个自己的网络就可以有不俗的表现，这对于复杂一些的数据集很难保证效果。对于复杂的分类问题，使

用已发布的精心设计的神经网络作为基础网络是更明智的选择。同时，从头开始训练深度的神经网络是一个繁重的工作，使用别人在 imagenet 上预训练好的权值来初始化我们的网络可以起到事半功倍的效果。

```

1  input_tensor = tf.keras.layers.Input(shape=(img_size_net,
2      img_size_net,3), name="x_input")
3  base_model = tf.keras.applications.InceptionV3(weights='imagenet',
4          include_top=False,
5          input_tensor=input_tensor,
6          input_shape=(img_size_net,img_size_net,3))
7
8  base_model.summary()      # 打印网络结构

```

activation_85 (Activation)	(None, 2, 2, 320)	0	batch_normalization_85[0][0]
mixed9_1 (Concatenate)	(None, 2, 2, 768)	0	activation_87[0][0] activation_88[0][0]
concatenate_1 (Concatenate)	(None, 2, 2, 768)	0	activation_91[0][0] activation_92[0][0]
activation_93 (Activation)	(None, 2, 2, 192)	0	batch_normalization_93[0][0]
mixed10 (Concatenate)	(None, 2, 2, 2048)	0	activation_85[0][0] mixed9_1[0][0] concatenate_1[0][0] activation_93[0][0]
<hr/>			
=====			
Total params: 21,802,784			
Trainable params: 21,768,352			
Non-trainable params: 34,432			
<hr/>			

可以看出这个网络的层数比我们自己设计的多多了。

- 一般来说，为了更高的效率，我们不需要整个网络，我们从输入开始，截取到输出为 6×10^6 的最后一个激活层。接上我们自己的全连接层和输出层，生成我们自己的网络：

```

1  base_model = tf.keras.Model(inputs=input_tensor, outputs=base_model.
2      get_layer('activation_74').output)
3
4  # 添加全局平均池化层
5  x = base_model.output
6  x = tf.keras.layers.GlobalAveragePooling2D()(x)
7
8  # 添加一个全连接层
9  x = tf.keras.layers.Dense(64, activation='relu')(x)
10 x = tf.keras.layers.Dropout(0.5)(x)
11
12 # 添加一个分类器，假设我们有200个类
13 predictions = tf.keras.layers.Dense(5, activation='softmax', name='
14 y_out')(x)
15
16 # 构建我们需要训练的完整模型
17 model = tf.keras.Model(inputs=input_tensor, outputs=predictions)
18
19 # 首先，我们只训练顶部的几层（随机初始化的层）
20 # 锁住所有 InceptionV3 的卷积层

```

```

19   for layer in model.layers[:195]:
20     layer.trainable = False
21   for layer in model.layers[195:]:
22     layer.trainable = True
23
24   # 编译模型（一定要在锁层以后操作）
25   model.compile(optimizer='rmsprop', loss=tf.keras.losses.
26   SparseCategoricalCrossentropy(), metrics=['accuracy'])
26   model.summary()      # 打印网络结构
27

```

4. 训练，首先用较大的学习率训练：

```

1 history = model.fit(trainSet_img, trainSet_label,
2                      batch_size=batch_size,
3                      epochs=20,
4                      validation_data=(testSet_img, testSet_label)
5
6

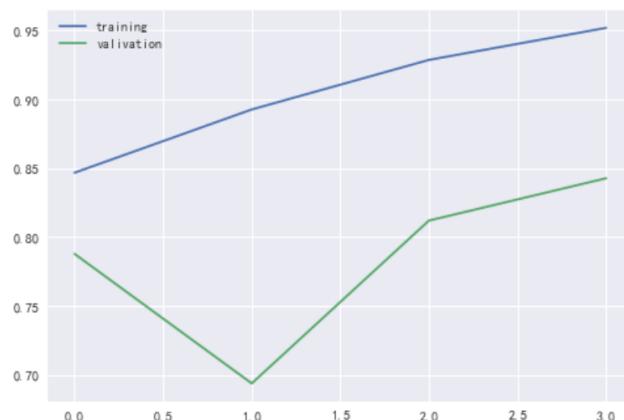
```

5. 查看训练精度的变化

```

1 import matplotlib.pyplot as plt
2 plt.plot(history.history['accuracy'][1:])
3 plt.plot(history.history['val_accuracy'][1:])
4 plt.legend(['training', 'validation'], loc='upper left')
5 plt.show()
6

```



6. 网络微调

固定后端的网络，调整顶层的几个模块的权值

```

1 # 让我们看看每一层的名字和层号，看看我们应该锁多少层呢：
2 for i, layer in enumerate(base_model.layers):
3   print(i, layer.name)
4   # 我们选择训练最上面的两个 Inception block

```

```

5      # 也就是说锁住前面135层，然后放开之后的层。
6      for layer in model.layers[:135]:
7          layer.trainable = False
8      for layer in model.layers[135:]:
9          layer.trainable = True
10     # 我们需要重新编译模型，才能使上面的修改生效
11     # 让我们设置一个很低的学习率，使用 SGD 来微调
12     from tensorflow.keras.optimizers import SGD
13     model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss=tf.
14         keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
15

```

7. 训练

```

1         history = model.fit(trainSet_img, trainSet_label,
2                             batch_size=batch_size,
3                             epochs=5,
4                             validation_data=(testSet_img, testSet_label)
5                         )
6

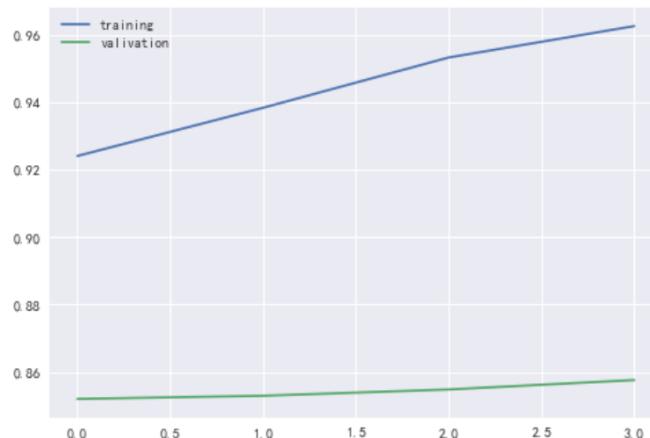
```

8. 查看训练精度的变化

```

1         import matplotlib.pyplot as plt
2         plt.plot(history.history['accuracy'][1:])
3         plt.plot(history.history['val_accuracy'][1:])
4         plt.legend(['training', 'validation'], loc='upper left')
5         plt.show()
6

```



9. 保存模型

```

1         model.save_weights(run_path + "model_weight.h5")
2         json_config = model.to_json()
3         with open(run_path + 'model_config.json', 'w') as json_file:
4             json_file.write(json_config)

```

10. 模型验证部分详情见源代码

11. 使用记事本查看生成的 model_config.json，如果在输入里出现"ragged": false, 字眼的字段，请手动把它删掉，因为 dnndk 里的 tensorflow 没有这个参数。没有就不用处理。

```
{"class_name": "Model", "config": {"name": "model", "layers": [{"class_name": "Input", "config": {"name": "input_1", "inbound_nodes": []}, "name": "input_1"}, {"name": "Conv1", "config": {"padding": "valid", "data_format": "channels_last", "name": "conv1", "trainable": false, "dtype": "float32", "filters": 32, "kernel_size": [3, 3], "dilation_rate": [1, 1], "activation": "linear", "use_bias": false, "kernel_initializer": "Zeros", "config": {}}, "name": "conv1"}]}
```

31.3 DNNDK 编译

同 minisFashiont 的例子，我们将在 DNNDK 里将模型编译成 DPU 能够运行的模型。

1. 启动虚拟机

2. 在虚拟机的/home/xiaobo/powersensor 目录下新建 flowerFive 文件夹，并把案例目录下的 dnndk 文件夹和 dataset_valid 文件夹复制到新建的文件夹下面。如果要使用自己新训练的模型，需要把自己的模型（在案例目录/pc/run 下面的 2 个模型文件）替换掉我们准备好的文件。

3. 进入 dnndk 目录，右击启动 shell，使用下面指令固化模型

```
1 ./1_vitisAI_keras2frozon.sh  
2
```

4. 第二步，量化，注意把 2_vitisAI_tf_quantize.sh 中的 input_nodes、input_shapes、output_nodes 改成与第一步打印的节点名称一致。

```
1 ./2_vitisAI_tf_quantize.sh  
2
```

5. 第三步，编译模型

```
1 ./3_vitisAI_tf_compile.sh  
2
```

编译的结果在 compile_result 文件夹里，里面的.elf 文件就是编译好的模型。

31.4 EDGE 调用

1. 进入 powersensor 的 jupyter 文件管理页面，在/powersensor_workspace/powersensor_ai 下面新建 flowerFive 目录（随教程发布的案例已经准备好文件了）。在新建的 flowerFive 目录下新建 dataset_valid 和 edge 文件夹



2. 通过 jupyter 的上传功能，把案例目录下的 edge 文件夹下的两个文件上传到 powersensor 的 edge 目录；验证集 dataset_valid 下的文件也同理上传到相应目录。
 - ★ 如果要使用自己新训练的模型，可以把 edge 下的 elf 换成虚拟机里的 compileResult 下的 elf 文件。
 - ★ 注意虚拟机里的文件不能直接上传（找不到），要先拷贝到自己的电脑里才能上传。
3. 打开 powersensor 的 edge 下的 powersensor_ministNumer.ipynb 文件，按照 notebook 里面的指导逐个运行程序。
4. 首先也是加载头文件和重要的参数，其中 DPU 网络参数应该与 DPU 的编译结果输出保持一致，否则会导致 DPU 崩溃。

```

1   from dnndk import n2cube
2   import numpy as np
3   from numpy import float32
4   import os
5   import cv2
6   import matplotlib.pyplot as plt
7   import random
8   import time
9   import matplotlib as mpl
10  from matplotlib import font_manager
11  import PowerSensor as ps
12  from IPython.display import clear_output
13
14  mpl.rcParams['axes.unicode_minus']=False      # 正常显示负号
15  font = font_manager.FontProperties(fname="/usr/share/fonts/truetype/
droid/DroidSansFallbackFull.ttf")
16
17  # 训练用的图像尺寸
18  img_size_net = 128
19  # 训练的batch大小
20  batch_size = 32
21
22  wordlist = ['雏菊', '蒲公英', '玫瑰', '向日葵', '郁金香']
23
24  # DPU 网络参数
25  # KERNEL_CONV="testModel"
26  ELF_NAME = "dpu_testModel_0.elf"
27  CONV_INPUT_NODE = "x_input_Conv2D"

```

```

28 CONV_OUTPUT_NODE = "y_out_MatMul"
29

```

5. 读取测试集

```

1 dataset_path = '../dataset_valid/'
2 (validSet_images, validSet_labels) = load_valid_data(dataset_path)
3
4 # 2. 图像预处理
5 # test_images = test_images.reshape((-1,28,28,1)) / 255.
6 validSet_images = np.array(validSet_images, dtype='float32')
7
8 # 3. 随机打印8个测试图像
9 fig, ax = plt.subplots(5, 2)
10 fig.set_size_inches(15,15)
11 for i in range(5):
12     for j in range(2):
13         l = random.randint(0, len(validSet_labels))
14         ax[i, j].imshow(cv2.cvtColor(validSet_images[l], cv2.
15 COLOR_BGR2RGB))
16         title = wordlist[validSet_labels[l]]
17         title_utf8 = title.decode('utf8')
18         ax[i, j].set_title(title_utf8, fontproperties=font)
19 plt.tight_layout()

```

6. 加载 dpu

```

1 dpu1 = ps.DpuHelper()
2 dpu1.load_kernel(ELF_NAME, input_node_name=CONV_INPUT_NODE,
3 output_node_name=CONV_OUTPUT_NODE)

```

7. 测试集精度测试

```

1 tick_start = time.time()
2 test_num = len(validSet_labels)
3 right_eg_cnt = 0
4 for i in range(test_num):
5     img1_scale = validSet_images[i]
6     softmax = dpu1.predict_softmax(img1_scale)
7     pdt= np.argmax(softmax, axis=0)
8     if pdt == validSet_labels[i]:
9         right_eg_cnt += 1
10    tick_end = time.time()
11    print('精度: ' + str((right_eg_cnt*1.) / test_num))
12    print('测试 ' + str(test_num) + ' 个样本。耗时 ' + str(tick_end -
13 tick_start) + ' 秒!')
14 #####

```

```

14 # 精度: 1.0
15 # 测试 50 个样本。耗时 9.1545650959 秒!
16

```

8. 随机样本测试

```

1 fig, ax = plt.subplots(5, 2)
2 fig.set_size_inches(15,15)
3 for i in range(5):
4     for j in range(2):
5         l = random.randint(0, len(validSet_labels)-1)
6         img1_scale = validSet_images[l]
7         softmax = dpu1.predict_softmax(img1_scale)
8         pdt= np.argmax(softmax, axis=0)
9         ax[i, j].imshow(cv2.cvtColor(validSet_images[l], cv2.
10 COLOR_BGR2RGB))
11         #           title = "预测: " + str(wordlist[pdt]) + "\n" + "真实:
12         " + str(wordlist[test_labels[l]])
13         title = "预测: " + wordlist[pdt] + "\n" + "真实: " + wordlist[
14         validSet_labels[l]]
15         title_utf8 = title.decode('utf8')
16         ax[i, j].set_title(title_utf8, fontproperties=font)
17 plt.tight_layout()
18

```



9. 拍照测试

```

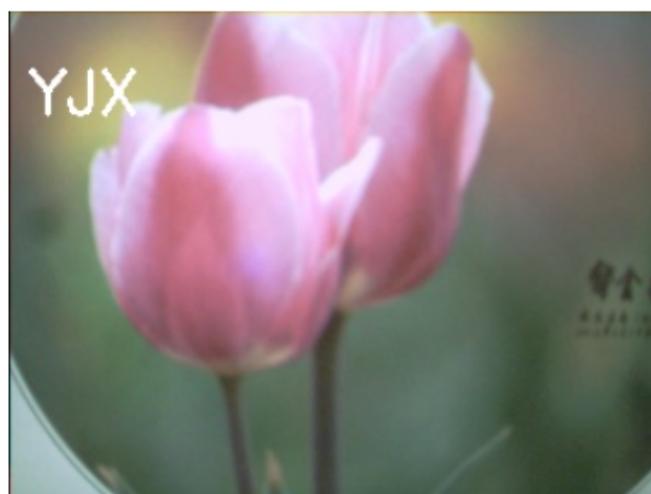
1 # 这个对象用于操作摄像头
2 cam1 = ps.ImageSensor()
3 wordlist = ['CJ', 'PGY', 'MG', 'XR', 'YJX']
4 #cv_font = cv2.freetype.createFreeType2()
5 #cv_font.loadFontData(fontFileName='/usr/share/fonts/truetype/droid/
6 DroidSansFallbackFull.ttf', id=0)
7 # 指定编码方式
8 fourcc = cv2.VideoWriter_fourcc(*'MJPG')

```

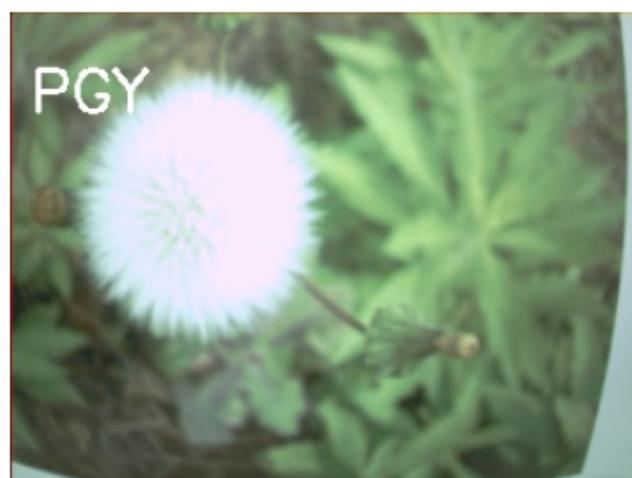
```
1     out1 = cv2.VideoWriter('output3.avi',fourcc, 1, (320,240))
2     for i in range(50):
3         # 记录时间
4         start = time.time()
5         # 清空显示区
6         clear_output(wait=True)
7         # 读取图像
8         imgMat = cam1.read_img_ori()
9         imgShow = cv2.resize(imgMat, (320,240))
10        #     imgMat_cali = grey_world2(imgMat)
11        # 图像缩放，太大的图像显示非常浪费资源
12        tempImg = cv2.resize(imgMat, (128,128))
13        img_scale = tempImg / 255.
14        img_scale = np.array(img_scale, dtype=np.float32)
15        softmax = dpu1.predit_softmax(img_scale)
16        pdt= np.argmax(softmax, axis=0)
17        font=cv2.FONT_HERSHEY_SIMPLEX
18        cv2.putText(imgShow,wordlist[pdt],(10,50), font, 1,(255,255,255)
19,2)
20        #     cv_font.putText(imgShow,wordlist[pdt], (10,10), fontHeight
21        #=30, color=(255,255,255), thickness=-1, line_type=cv2.LINE_4,
22        # bottomLeftOrigin=False)
23        #     tempImg = imgMat
24        # 显示图像
25        out1.write(imgShow)
26        img = ps.CommonFunction.show_img_jupyter(imgShow)
27        # 记录运行时间
28        end = time.time()
29        # 打印运行时间
30        print(end - start)
31        # 因为网络传输的延时，需要稍息一下
32        time.sleep(0.02)
33        out1.release()
```



0.300905942917



0.297462940216



0.296260118484

第三十二章 AI6 - 人脸口罩识别 - 综合案例

内容提要

- | | |
|-----------------------------------|----------------------------------|
| <input type="checkbox"/> 人脸口罩识别方案 | <input type="checkbox"/> 需要准备的东西 |
| <input type="checkbox"/> 原理说明 | <input type="checkbox"/> 操作教程 |

32.1 人脸口罩识别方案

32.2 原理说明

身份识别功能和口罩是否佩戴规范

这两个问题可以合二为一，我们提供的思路是将其转换为 4 类的分类问题

- (1) 未佩戴口罩或者口罩不规范；
- (2) 佩戴规范的同学甲；
- (3) 佩戴规范的同学乙；
- (4) 佩戴规范的同学丙。

收集一定量的图片数据，然后通过深度学习训练得到分类模型，再转换为 powersensor 可以运行的模型，最后再部署在 ps 上即可。

32.3 需要准备的东西



笔记 案例资料包的下载地址请见 27.2。

32.4 操作教程

32.4.1 视频教程

1. 数据收集 <https://www.bilibili.com/video/BV18a4y1L72M>
2. 训练与部署 <https://www.bilibili.com/video/BV1wy4y1r76m>

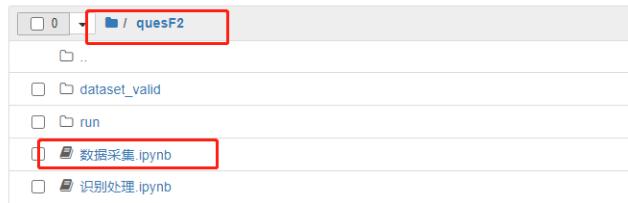
32.4.2 数据收集

由于我们是采用深度学习的方式实现分类，因此需要准备训练用的数据集：

- (1) 数据集要按文件夹存放，即同学甲，同学乙，同学丙，未带口罩 4 类存放在 4 个不同的文件夹里。
- (2) 数据集收集要保证不同光照，不同场景，不同角度的数据集，但是人脸要完整不能缺失。
- (3) 数据集可以用手机或者 powersensor 来录制。

* 使用 powersensor 录制数据集的方法如下：

1. 把 powersensor 上电，通过浏览器进入 quesF2，打开数据采集.ipynb



2. 逐个运行单元格，重要代码解释：

1 删掉原来的数据和新建数据集文件夹

```
In [1]: !rm -rf ./run/
!mkdir ./run
!mkdir ./run/aa/
!mkdir ./run/ab/
!mkdir ./run/ac/
!mkdir ./run/an/
```

这个是用来删掉上次录制的数据

2 数据收集

```
In [2]: cam = ps.ImageSensor()
```

```
In [3]: sort = "a"
# 这是保存的文件夹路径，每种数据录制前需要修改
sort = "aa" # 例如：./run/aa/
name_start = "aa"
for i in range(100):
    # 记录时间
    start = time.time()
    # 读取相机
    clear_output(wait=True)
    # 读取图像
    img = cam.readImg()
    imgShow = cv2.resize(img, (320, 240))
    cv2.imwrite("./run/" + sort + "/" + name_start + str(cnt) + ".jpg", imgShow)
    cnt += 1
    # 显示图像
    imgShow = CaoaoFunction.showImg_Jupyter(imgShow)
    # 记录结束时间
    end = time.time()
    # 打印时长
    print(i, end - start)
    # 因为摄像头的延时，需要稍等一下
    time.sleep(0)
```

3. 最后一格代码是把保存的图片文件夹打包成一个压缩包，

```
1 !tar -cvf dataset.tar.gz ./run/
2
```

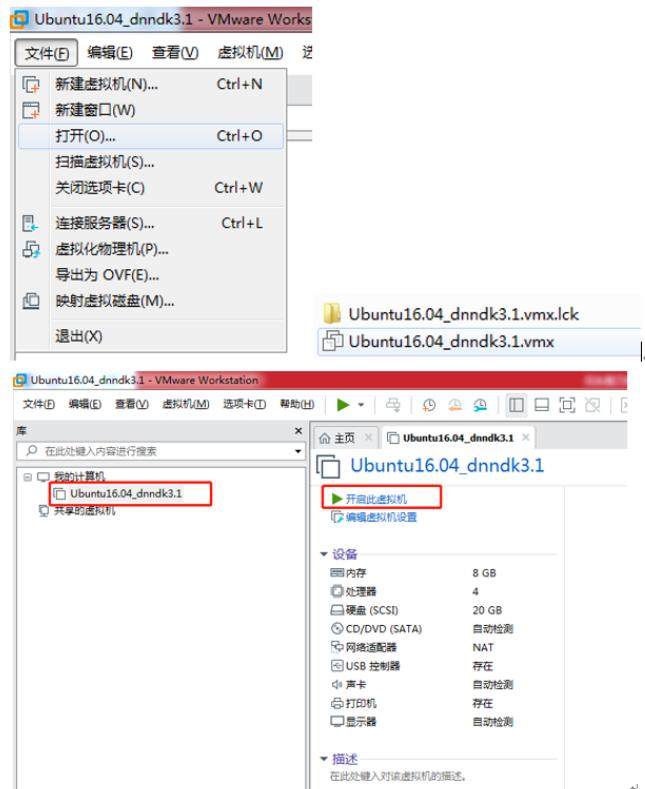
数据集的优化

数据集应该经过人工挑选，把半边脸，没有脸等不合格的数据剔除，方便训练。

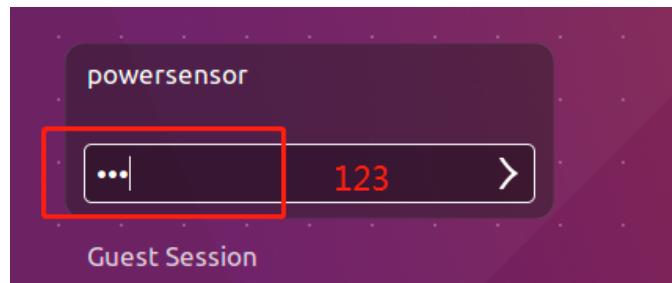
采集过程中手请不要碰 powersensor 的线的位置，不然可能短路网口造成卡顿

32.4.3 模型设计、训练与转换

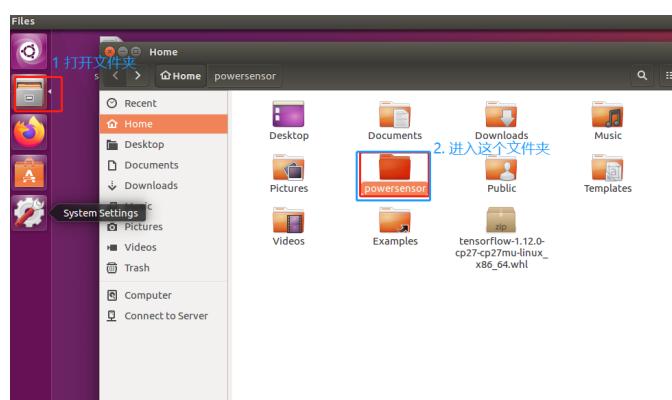
1. 打开 vmware，启动虚拟机文件（第一次使用后，在左边有看到）



2. ubuntu 系统的密码是 123



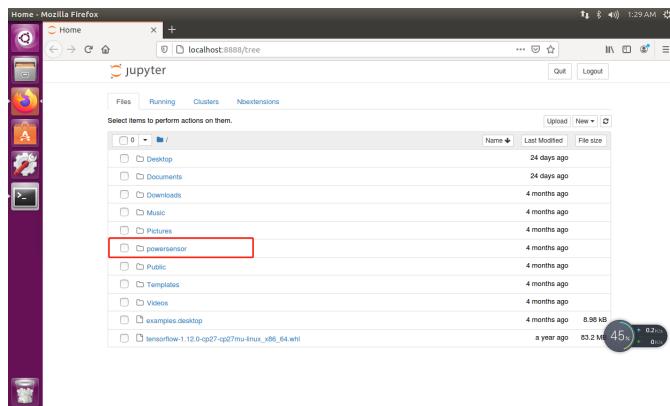
3. 打开文件管理器,把我们的资料包复制粘贴到 Ubuntu 的/home/powersensor/powersensor 目录下



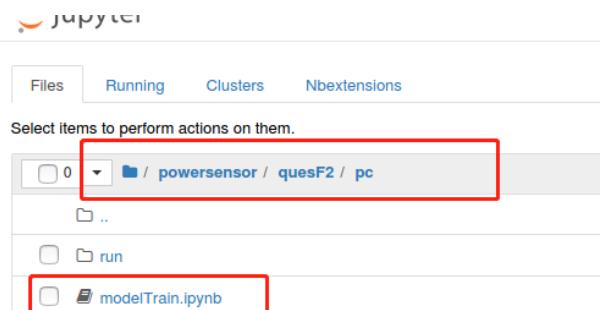
4. 训练前, 需要把用户自己的数据集拷贝到里面的 dataset 和 dataset_valid 文件夹下

注意这里面是分 4 个文件夹存放，ma 存放同学甲的带口罩的图片，mb 存放同学乙的带口罩的图片，mc 存放同学丙的带口罩的图片，um 存放所有不带口罩的图片。

- 双击 ubuntu 桌面上 “start.sh”，会启动浏览器，打开 powersensor->rps->pc，



- 打开 “modelTrain.ipynb”



- 下面的训练模型过程是在 Jupyter 中进行。一个一个点就可以了。。。

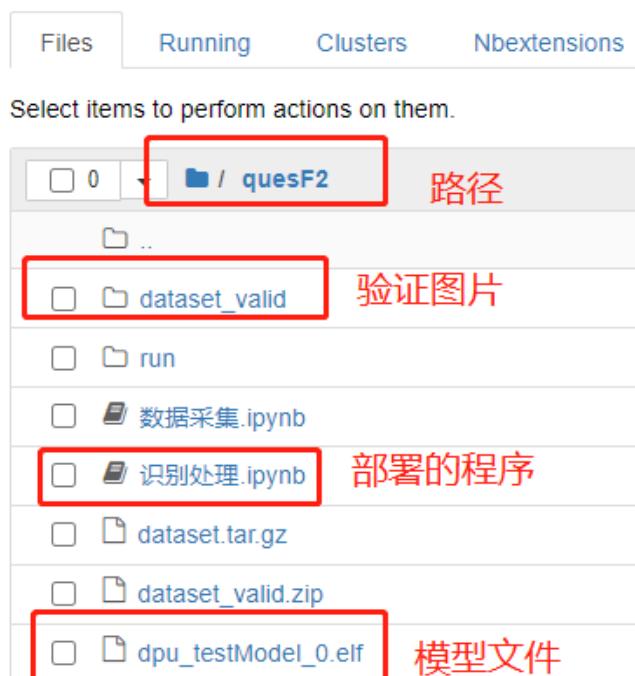
- 包含头文件：主要执行包含一些相关头文件库文件等。
- 一些重要的训练参数：主要对路径等参数进行说明。
- 生成 tf 适用的数据集：原始采集的图片格式、尺寸大小不同，这个文件可以将不同的图片转换为统一尺寸统一格式。
- 数据集读取测试：测试一下上一步生成的数据集对不对（理论上可以不执行，但是为了验证正确性可以读一下）。
- 训练集和测试集准备：c 步骤生成的图片全部放在一起，在实际中要拿出一部分做训练，另外剩下的图片做验证，这样可以保证数据集和测试集不重合，更能说明检测新图片的效果。
- 深度学习网络模型设计：设计深度学习的网络模型，包括卷积层、池化层、全连接层和输出层。
- 神经网络训练：用 f 步骤设计的 tf 网络进行训练。（根据电脑性能不同训练时间不同，比如我的电脑训练了 10 分钟左右）
- 打印训练过程的精度的变化：这一步是为了直观的感受训练过程，可以看到随着训练的进行精度越来越高。
- 测试：用测试集标签验证一下模型预测的效果和精度。

- j. 保存模型：验证效果还行的话就可以保存模型了，这样下次直接调用训练结果就可以啦！
8. 下面是验证过程，在 jupyter 中进行：
- 加载预训练的模型：比如上次训练好的模型，就可以不再次训练直接调用哦
 - 加载验证数据集：加载需要验证的数据图片集。
 - 使用训练的网络预测测试样本：嗯，就是就 a 的模型验证 b 的数据集。
9. 下面是模型转换过程，在 jupyter 中进行：
- 模型转换：将 j 步骤模型保存为 Powersensor 的可以使用的模型，主要是将训练的浮点模型转换为定点模型，主要作用是加速。
 - 下面的文件是训练的模型文件，可以先拷贝到我们电脑的桌面上。



32.4.4 在 powersensor 上部署模型

1. 给 ps 上电，打开 192.168.8.8/，进入 quesF2 文件夹：
2. 把你们的 dataset_valid 和刚才训练生产的模型文件上传到响应目录下：



dataset_valid 要打包成 dataset_valid.zip 文件，然后上传一次即可：

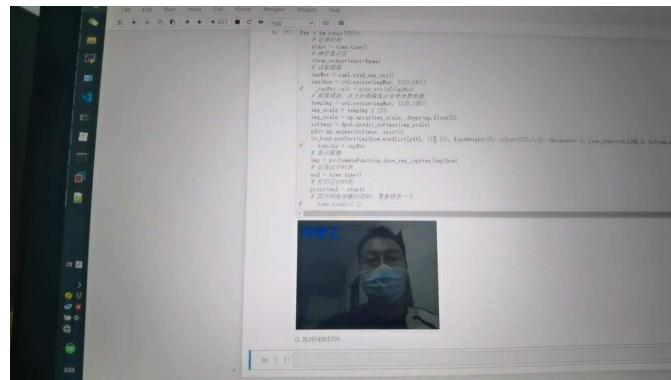
3. 打开识别处理.ipynb，里面的第二格可以把 dataset_valid.zip 解压出来

1 解压验证集

In [3]: `!unzip dataset_valid.zip`

Archive: dataset_valid.zip

4. 接着逐个运行 jupyter 的单元格，即可看到识别效果



第六部分

兼容拓展模块篇 - Extend

第三十三章 Extend 1 - IO_ 串口隔离板

本程序需要配套京联科技的 IO 扩展板，本扩展板带 4 路光耦隔离输入、2 路光耦隔离输出、1 个标准 RS232 接口。其中 IO 功能和 PLC 电路兼容。

其中主要代码及注释如下：

首先一定要初始化 IO 对象!!

```
1 # 初始化控制对象
2 gpiox = ps.GpioPort()
3
```

IO 输入测试程序!!

```
1 # 测试隔离板IO输入
2 # 设置输入模式
3 gpiox.set_mode(0, ps.PortPara.Gpio_Mode_In)
4 gpiox.set_mode(1, ps.PortPara.Gpio_Mode_In)
5 gpiox.set_mode(2, ps.PortPara.Gpio_Mode_In)
6 gpiox.set_mode(3, ps.PortPara.Gpio_Mode_In)
7 # 打印输入给IO隔离板的电平高低
8 for i in range(10):
9     x1 = gpiox.read_value(0)
10    x2 = gpiox.read_value(1)
11    x3 = gpiox.read_value(2)
12    x4 = gpiox.read_value(3)
13    print(x1, x2, x3, x4)
14    time.sleep(1)
15
```

IO 输出测试程序!!

```
1 # 测试隔离板IO输出
2 # 设置输出模式
3 gpiox.set_mode(4, ps.PortPara.Gpio_Mode_Out)
4 gpiox.set_mode(5, ps.PortPara.Gpio_Mode_Out)
5 # 输出通断1s切换一次
6 for i in range(50):
7     gpiox.set_value(4, ps.PortPara.Gpio_Value_Low)
8     gpiox.set_value(5, ps.PortPara.Gpio_Value_Low)
9     time.sleep(1)
10    gpiox.set_value(4, ps.PortPara.Gpio_Value_High)
11    gpiox.set_value(5, ps.PortPara.Gpio_Value_High)
12    time.sleep(1)
13
```

使用串口一定要初始化串口

```
1 # 初始化串口，运行一次即可
```

```
2 s1 = ps.UartPort()
```

RS232 发送数据程序

```
1 # 串口发送
2 s1.set_baudrate(9600)
3 for i in range(2):
4     s1.u_send_bytes([1, 3, 5, 7, 9])
```

第三十四章 Extend 2 - 电机驱动板

使用京联科技的电机扩展板，可以直接驱动两路小型直流电机。具体的使用案例就是双轮小车。

```
1 # 初始化控制对象
2 pwx = ps.PwmActuator()
3 gpx = ps.GpioPort()
4 # IO为输出模式
5 gpx.set_mode(1, ps.PortPara.Gpio_Mode_Out)
6 gpx.set_mode(2, ps.PortPara.Gpio_Mode_Out)
7 gpx.set_mode(3, ps.PortPara.Gpio_Mode_Out)
8 gpx.set_mode(4, ps.PortPara.Gpio_Mode_Out)
9 # 普通模式下的pwm输出
10 # 预分频，实际分频数=设定值+1
11 pwx.preDividor = 10 - 1
12 # 高电平时间，最大为周期的设定值，即999
13 pwx.period = 1000 - 1
14 # 模式设置，1：普通模式；
15 pwx.mode = 1
16
17 # 电机转动
18 # MotorNum:1左电机 2右电机
19 # Direction: 电机方向：1前进 2后退
20 # Speed: 电机速度200~999
21 def MotorMove(MotorNum,Direction,Speed):
22     if MotorNum==1 and Direction==1:
23         gpx.set_value(1, ps.PortPara.Gpio_Value_High)
24         gpx.set_value(2, ps.PortPara.Gpio_Value_Low)
25     elif MotorNum==1 and Direction==2:
26         gpx.set_value(1, ps.PortPara.Gpio_Value_Low)
27         gpx.set_value(2, ps.PortPara.Gpio_Value_High)
28     elif MotorNum==2 and Direction==1:
29         gpx.set_value(3, ps.PortPara.Gpio_Value_High)
30         gpx.set_value(4, ps.PortPara.Gpio_Value_Low)
31     elif MotorNum==2 and Direction==2:
32         gpx.set_value(3, ps.PortPara.Gpio_Value_Low)
33         gpx.set_value(4, ps.PortPara.Gpio_Value_High)
34 # 设置转速
35 if MotorNum==1:
36     pwx.pwm3 = Speed
37     pwx.pwm_setup(True)
38 elif MotorNum==2:
39     pwx.pwm2 = Speed
40     pwx.pwm_setup(True)
```

```
42 # 刹车
43 def MotorBrake(MotorNum):
44     if MotorNum==1:
45         gpiox.set_value(1, ps.PortPara.Gpio_Value_High)
46         gpiox.set_value(2, ps.PortPara.Gpio_Value_High)
47     elif MotorNum==2:
48         gpiox.set_value(3, ps.PortPara.Gpio_Value_High)
49         gpiox.set_value(4, ps.PortPara.Gpio_Value_High)
50
51 # 自由停车
52 def MotorFreeParking(MotorNum):
53     if MotorNum==1:
54         gpiox.set_value(1, ps.PortPara.Gpio_Value_Low)
55         gpiox.set_value(2, ps.PortPara.Gpio_Value_Low)
56     elif MotorNum==2:
57         gpiox.set_value(3, ps.PortPara.Gpio_Value_Low)
58         gpiox.set_value(4, ps.PortPara.Gpio_Value_Low)
```

具体小车使用程序如下！

```
1 # 初始化控制对象
2 pwmx = ps.PwmActuator()
3 gpiox = ps.GpioPort()
4
5 # 正向跑5ms，反向跑5s
6 MotorMove(1,1,800)
7 MotorMove(2,1,800)
8 time.sleep(1)
9 MotorMove(1,2,800)
10 MotorMove(2,2,800)
11 time.sleep(1)
12 # 1电机刹车，2电机自由停车
13 MotorBrake(1)
14 MotorFreeParking(2)
```

第三十五章 Extend 3 - 双舵机云台板

内容提要

- PWM 原理简述
- 舵机云台控制
- 舵机云台简介

35.1 PWM 原理简述

PWM 的详细原理可参考第一部分 4.7 节 PWM 输出

下面仅针对本节内容对 PWM 做简要介绍

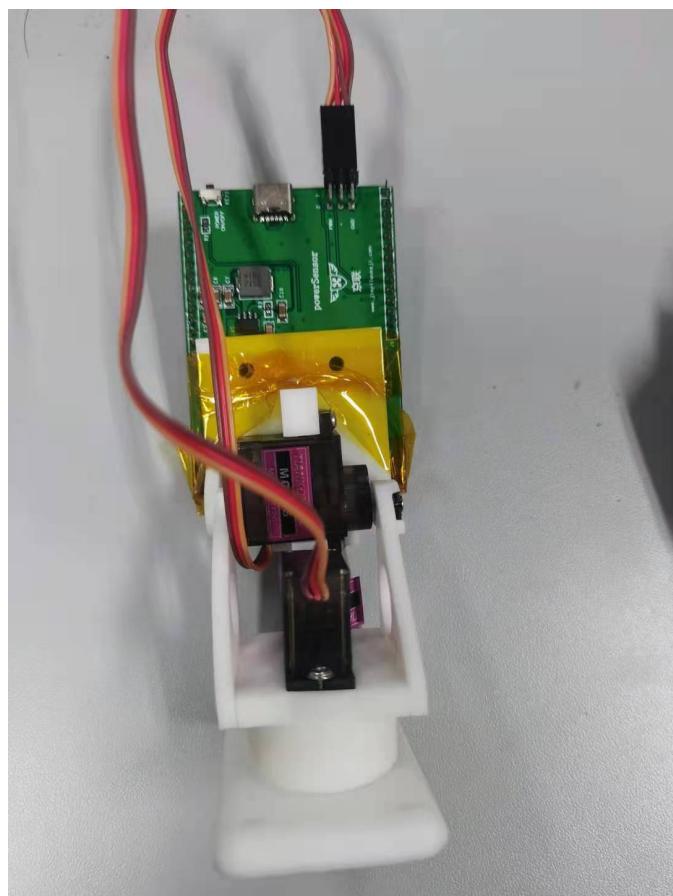
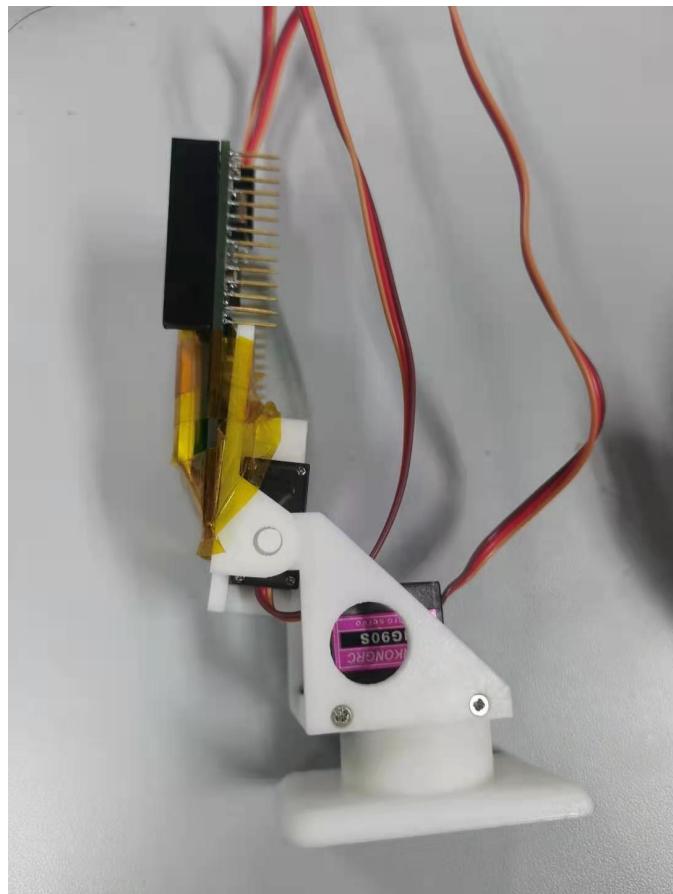
1. 时钟源频率 $f_s = 100MHz$
2. 由于时钟源频率较高，为了方便得到低频的 PWM 信号需要进行预分频，预分频频率记为 n_f
3. 利用预分频之后的信号配合同步器就可以控制 PWM 波的脉冲重复频率 f_p 和占空比
4. PWM 波的脉冲重复频率和时钟源频率的关系为：

$$f_p = \frac{f_s}{(n_f + 1)(n_p + 1)}$$

35.2 舵机云台简介

35.2.1 舵机云台的组成

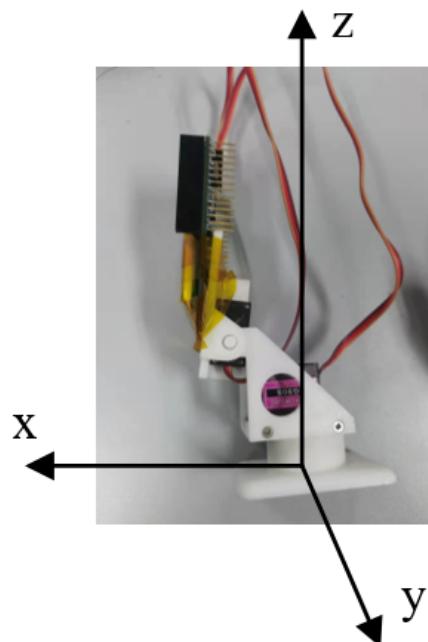
舵机云台如下图所示，主要由两个舵机和相应的控制模块组成



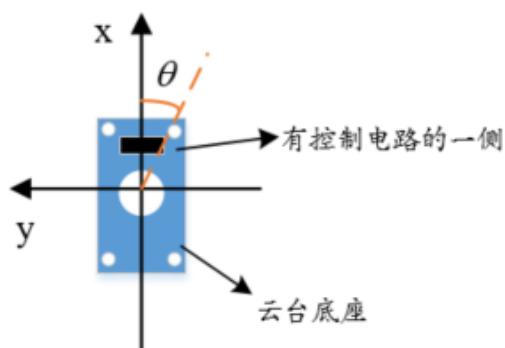
两个舵机可分别控制云台进行水平和垂直方向上的运动。

35.2.2 舵机云台坐标系的建立

为了更加准确地描述云台的运动，我们将云台放到如下坐标系中。云台出于下图状态时，我们把它叫做初始状态。

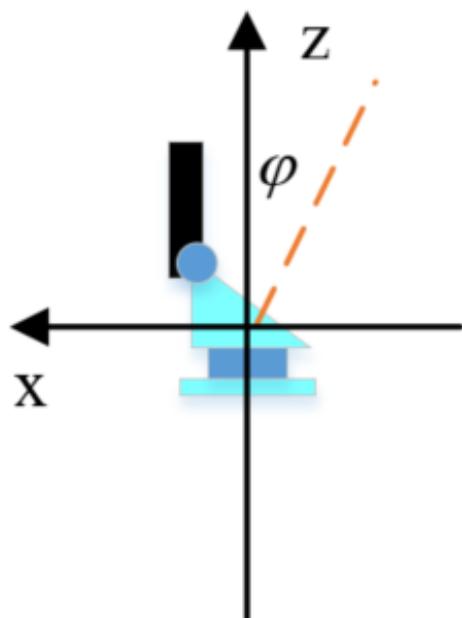


从 z 轴正方向往下看（俯视）,xoy 平面如下图所示。其中 x 轴正方向为云台底座有控制电路和摄像头的方向。云台与 x 轴正方向的夹角定义为水平方位角，记作 θ 。云台处于如上图所示状态时可以看出与 x 轴正方向的夹角为 0 度。云台顺时针旋转时方位角为正（下图所示 θ 方向），反之为负。经过测试，该云台水平方位角的转动范围为 $[-113.64^\circ, 75.76^\circ]$ 。



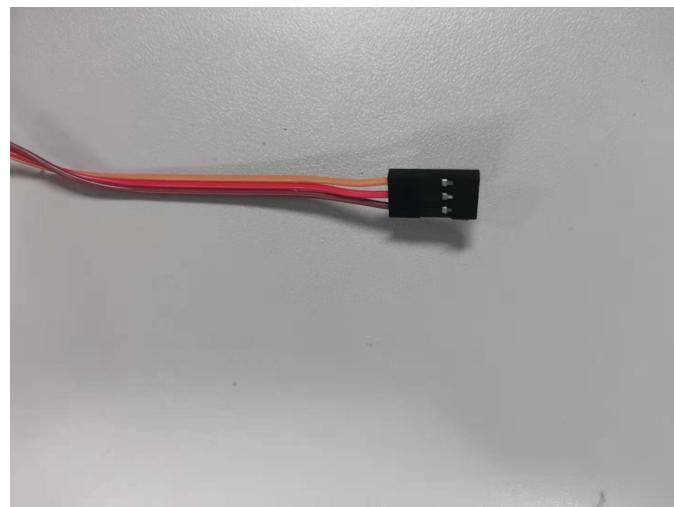
从 y 轴正方向看（侧视）, xoz 平面上如图所示。云台与 z 轴正方向的夹角定义为俯仰角，记作 ϕ 。云台处于初始状态时可以看出与 z 轴正方向的夹角为 0 度。同样定义云

台顺时针旋转时俯仰角为正值（下图所示 ϕ 方向），反之为负。经过测试，该云台水平方位角的转动范围为 [-28.41°, 56.82°]。（俯仰角范围主要受云台机械结构的限制）

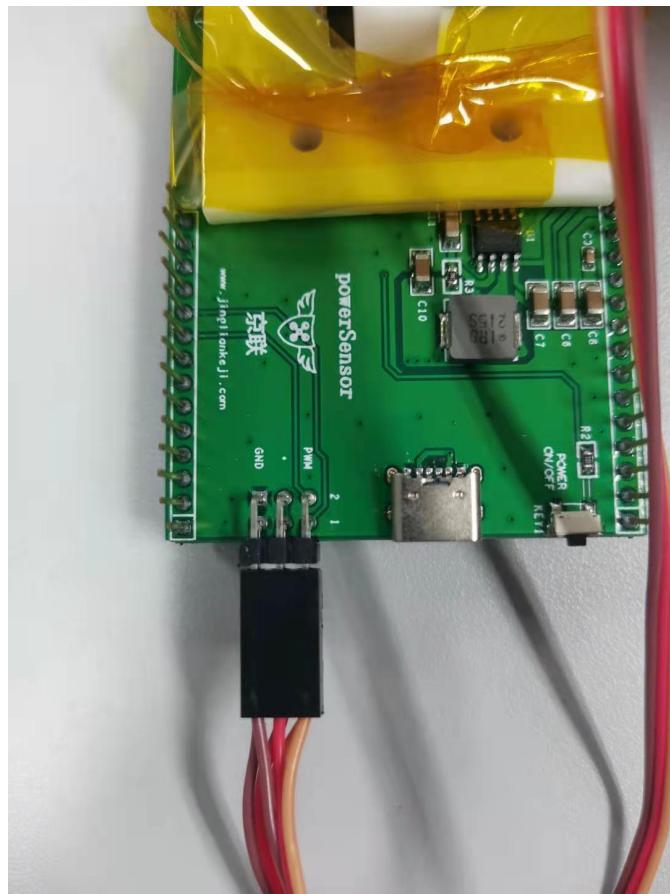


35.2.3 舵机与控制电路板的连接

舵机由一块控制电路板进行控制。舵机与控制电路由三根线连接，如下图所示。



其中，棕色的线为接地线，黄色的线为传输 PWM 信号的线，中间红色的线是为舵机供电的 VCC。舵机需要由这三根线和控制电路进行连接，控制电路板上对应的接口如下图所示。



可以看到，控制电路板上有两组相同的接口分别记作 1 接口和 2 接口。将两个舵机分别与两个接口进行连接，注意连接时要将 PWM 信号线和 GND 与电路板上的 PWM 和 GND 对应，不要插反了!!!!

 **笔记** 在本节中我们将控制云台俯仰的舵机（与电路板相连）与控制电路板的接口 1 相连，控制云台水平方向的舵机（与云台底座相连）与控制电路板的接口 2 相连。

35.3 舵机云台控制

舵机旋转的角度由 PWM 信号的脉宽控制。简单的说，一个固定脉宽的 PWM 波对应舵机旋转的一个角度。因此，控制舵机的旋转实际上就相当于控制 PWM 波的脉宽。首先我们需要包含一些重要的头文件和定义重要的参数

```

1 import PowerSensor as ps
2 import time
3 from IPython.display import clear_output
4 import numpy as np

```

获取 PWM 模块的对象

```
1 pwmx = ps.PwmActuator()
```

进行预分频，并确定 PWM 波的频率（设为 100Hz）。

```

1 pwmx.preDividor = 100 - 1 # 1M hz
2 pwmx.period = 10000 - 1 # 100 hz

```

pwmx.pwm_set_width(par1,par2) 为控制通道和 PWM 脉宽的函数。其中参数 par1 控制电路板的通道，par1=0 时对应电路板上的接口 2，par1=1 时对应电路板上的接口 1。参数 par2 则控制 PWM 信号的脉宽（实际上控制的是计时器计时的个数），由于 PWM 信号的脉宽通常在 0.5ms 到 2.5ms 之间，而经过预分频后的频率（PWM 波的时钟频率）为 1MHz，因此产生 0.5ms 的脉宽需要 500 个时钟周期（分频后的），因此产生 2.5ms 的脉宽需要 2500 个时钟周期，因此参数 par2 的取值范围为 [500,2500]。

由上述原理，我们可以使云台旋转至某一固定方位角（注意，控制电路板上的接口 2 对应 par1=0）

```

1 # 方位角设置 0 通道
2 azimuth=0;# 输入方位角，单位为度 范围为 [-113.64,75.76]
3 accuracy=1/0.0947;# 精度，90/950
4 a=int(-azimuth*accuracy)+1300 # 1300 对应 0 度
5 if a>2500:
6     a=2500
7 if a<500:
8     a=500
9 pwmx.pwm_set_width(0, a)

```

也可以使云台旋转至某一固定俯仰角（注意，控制电路板上的接口 2 对应 par1=1）

```

1 # 俯仰角设置 1 通道
2 elevation=30;# 输入俯仰角，单位为度 范围为 [-28,41.56.82]
3 accuracy=1/0.0947;# 精度，90/950 和之前一样
4 e=int(-elevation*accuracy)+1100 # 1100 对应 0 度
5 if e>1400:
6     e=1400
7 if e<500:
8     e=500
9 pwmx.pwm_set_width(1, e)

```

接着，让云台将所有的角度都转一下

```

1 # 遍历最大范围
2 num=5
3 a1=[]
4 e1=[]
5 for i in range(num):
6     a1.append(-113.64+i*(75.76+113.64)/(num-1))
7     e1.append(-28.41+i*(28.41+56.82)/(num-1))
8 for i in range(num):
9     # 方位角设置 0 通道
10    azimuth=a1[i];# 输入方位角，单位为度 范围为 [-113.64,75.76]
11    accuracy=1/0.0947;# 精度，90/950

```

```

12     a=int(-azimuth*accuracy)+1300 #1300对应0度
13     if a>2500:
14         a=2500
15     if a<500:
16         a=500
17     pwmx.pwm_set_width(0, a)
18     #俯仰角设置1通道
19     elevation=e1[i];#输入俯仰角，单位为度 范围为[-28,41.56.82]
20     accuracy=1/0.0947;#精度，90/950和之前一样
21     e=int(-elevation*accuracy)+1100 #1100对应0度
22     if e>1400:
23         e=1400
24     if e<500:
25         e=500
26     pwmx.pwm_set_width(1, e)
27     time.sleep(1)

```

最后，我们要回到初始状态

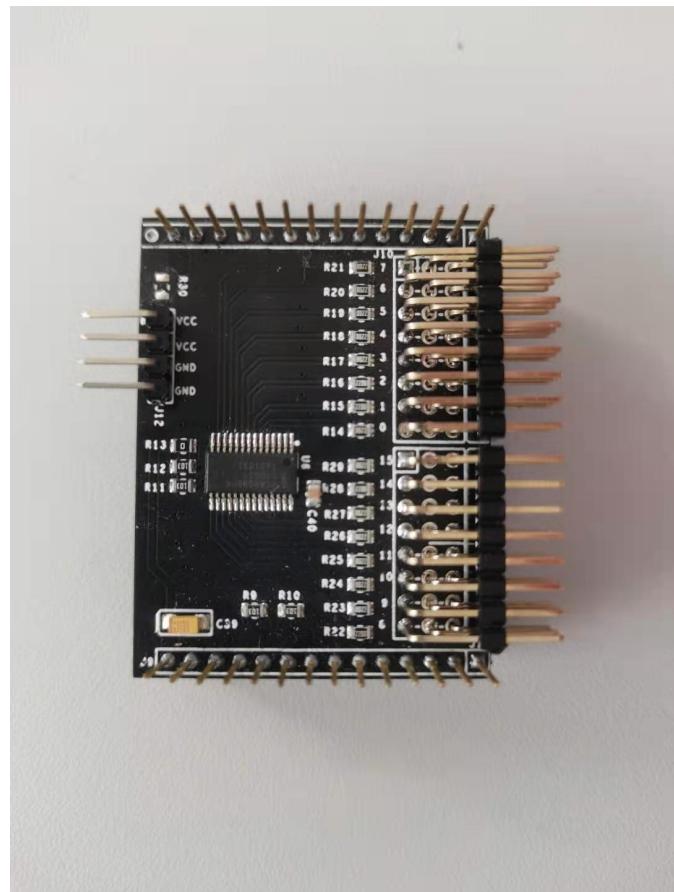
```

1     #方位角设置0通道
2     azimuth=0;#输入方位角，单位为度 范围为[-113.64,75.76]
3     accuracy=1/0.0947;#精度，90/950
4     a=int(-azimuth*accuracy)+1300 #1300对应0度
5     if a>2500:
6         a=2500
7     if a<500:
8         a=500
9     pwmx.pwm_set_width(0, a)
10    #俯仰角设置1通道
11    elevation=0;#输入俯仰角，单位为度 范围为[-28,41.56.82]
12    accuracy=1/0.0947;#精度，90/950和之前一样
13    e=int(-elevation*accuracy)+1100 #1100对应0度
14    if e>1400:
15        e=1400
16    if e<500:
17        e=500
18    pwmx.pwm_set_width(1, e)

```

第三十六章 Extend 4 - 16 路 PWM 驱动板

该扩展板可同时输出 16 路 PWM 信号，控制 16 个舵机同时工作，模块如下图所示：



首先包含一些头文件

```
1 import PowerSensor as ps  
2 import Pca9685
```

初始化

```
1 pwm16 = Pca9685.PCA9685(0x60)  
2 pwm16.setPWMFreq(50) # 频率
```

设置脉冲宽度

```
1 for i in range(16):  
2     pwm16.setServoPulse(i, i * 50 + 1000) # PWM frequency is 50HZ, the period is  
20000us
```

第三十七章 Extend 5 - TOF 测距板 - VL53L1X

内容提要

- tof 模块介绍
- 使用教程
- 函数介绍

37.1 tof 模块介绍

tof 的全称是 time of fly, 即通过测量光的飞行时间来测量距离, 具有抗干扰、精度高等多种优势。本文选的 tof 传感器为 st 公司开发的 vl53l1x, 配套的模块如下图所示:



主要特性有：

- 940nm 的激光传感器
- 最远可以测量 4m
- 更新频率为 50Hz
- 通信协议 I2C
- 精度大概 0.3cm 左右

37.2 函数介绍

```

1 # Vl531x_helper
2 # - sensor = Vl531x_helper(i2c)
3 # --- 构造函数，输入为预配置的i2c
4 # - sensor.vl5311x_setDistanceMode(dis)
5 # --- 配置函数，设置最大测量距离，距离越长需要的探测时间越长；参数为模式，可以是'long', 'medium', 'short'中的一个
6 # - sensor.vl5311x_setMeasurementTimingBudget(tim)
7 # --- 配置函数，设置最大测量时间；参数为时间，单位微秒，long模式建议33000
8 # - sensor.sensor_start(tim)
9 # --- 测量函数，启动测量；参数为采样周期，单位毫秒，原则上要大于最大测量时间。
10 # - sensor.sensor_new_data()
11 # --- 测量函数，判断是否收到新数据，返回值为布尔型
12 # - sensor.sensor_read_mm()
13 # --- 测量函数，读到的距离值，单位为毫米
14 # - sensor.sensor_stop()
15 # --- 测量函数，停止测量，进入休眠模式

```

37.3 使用教程

1. 初始化

```

1 i2c1 = ps.I2cPort()
2 sensor = Vl531x_helper.vl531x_helper(i2c1)
3
4 # 1. 传感器初始化
5 sensor.sensor_init()
6 # 2. 设置参数
7 # 2.1 最大距离参数，可以是'long', 'medium', 'short'中的一个
8 sensor.vl5311x_setDistanceMode('short')
9 # 2.2 设置最大测量时间，单位微秒，medium模式建议10000
10 sensor.vl5311x_setMeasurementTimingBudget(33000)
11
12

```

2. 启动测量

```

1 sensor.sensor_start(50)
2

```

3. 周期性数据读取

```

1 for i in range(500):
2     clear_output(wait=True)
3     over_time = 0

```

```

4     while(not sensor.sensor_new_data()):
5         over_time = over_time + 1
6         time.sleep(0.001)
7         if over_time > 200:
8             print('over tiem')
9             break
10        print(sensor.sensor_read_mm())
11        time.sleep(0.1)
12

```

结果：

1766

4. 滚动条显示测量结果

```

1     # 交互式控件用的用库
2     from ipywidgets import widgets
3     from IPython.display import display
4     slider_dis = widgets.FloatSlider(
5         value=0,
6         min=-0, # max exponent of base
7         max=600, # min exponent of base
8         step=0.1, # exponent step
9         description='Distance'
10    )
11
12    display(slider_dis)
13    for i in range(500):
14        res = sensor.sensor_read_mm()
15        slider_dis.value = res
16        time.sleep(0.1)
17

```

结果：

1

Distance —○— 600.00

5. 停止测量

```
1 sensor.sensor_stop()
```

第三十八章 Extend 6 - 红外温度测量阵列 -

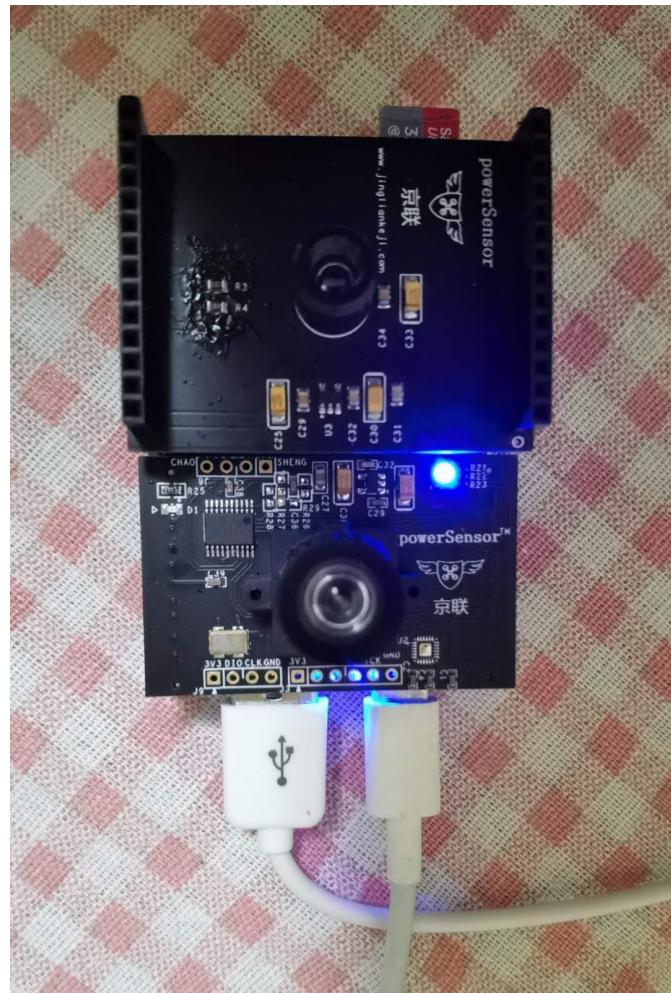
MLX90621

内容提要

- 红外温度测量阵列模块介绍
- 使用教程
- 函数介绍

38.1 红外温度测量阵列模块介绍

红外温度测量阵列，顾名思义，是一种通过测量红外辐射来测量（看到）物体的温度，本教程的传感器型号是 mlx90621，是一款 16×4 大小的红外测量阵列。我们的红外测量模块如下图所示：



这个模块的主要参数如下：
工作温度：−40° 到 85°
测量温度：−20° 到 300°

更新频率：最高 100Hz 左右
 通信协议 I2C
 内置的 eeprom 保存有出厂标定的校正数据
 噪声范围：0.20K rms

测量温度比工作温度高的原因是非接触，通过看的方式测量

38.2 函数介绍

```

1 # 模块名称: Mlx90621
2 ## - sensor = Mlx90621.mlx90621_helper(i2c)
3 ##### 这个函数是实例化函数（构造函数），需要传递需要使用的i2c
4 ## - sensor.sensor_init()
5 ##### 这个函数是传感器的初始化和配置函数
6 ## - sensor.sensor_read()
7 ##### 这个函数可以读取到校正的温度数据；返回值有4项：(1) 传感器本身温度，(2) 温度
     阵列(16x4)，(3) 最小温度，(4) 最大温度
8 ## - sensor.sensor_get_img(block_size=15)
9 ##### 可以函数可以读取到一个图像，这个图像是温度数据经过伪色处理的rgb图像，
     block_size是填充的色块大小

```

38.3 使用教程

1. 包含库文件

```

1 import PowerSensor as ps
2 import numpy as np
3 from smbus2 import SMBus, i2c_msg
4 import matplotlib.pyplot as plt
5 from IPython.display import clear_output
6 import cv2
7 import imp
8 import time

9
10 # 除了常用的模块，还需要import 90621模块
11 import Mlx90621 as Mlx90621
12

```

2. 初始化

```

1 # 1. 初始化i2c，即准备用于操作红外测温传感器的i2c
2 i2c = ps.I2cPort()
3 # 2. 实例化红外测温传感器的对象
4 sensor = Mlx90621.mlx90621_helper(i2c)
5 # 3. 初始化红外测温传感器
6 sensor.sensor_init()

```

```

7     # 4。 初始化摄像头
8     cam1 = ps.ImageSensor()
9

```

3. 读取温度数据

```

1     x1, x2, x3, x4 = sensor.sensor_read()
2     print("传感器本身温度是: " + str(x1))
3     print("测量(看到)温度阵列是: " + str(x2))
4     print("测量温度最小值是: " + str(x3))
5     print("测量温度最大值是: " + str(x4))
6

```

结果如下：

```

传感器本身温度是: 36.70853158892781
测量(看到)温度阵列是: [19.52076333 18.82496852 18.36947801 14.90737531 20.5469089
9 20.30326203
20.82600882 19.10676991 21.89554307 24.45890088 22.05777012 18.29867623
21.99345029 23.2531097 22.43435587 20.80230739 23.08293985 23.95048024
24.629493 21.30446969 22.83718048 24.01769583 23.80581959 21.38887569
23.31887703 24.08156868 24.46276111 20.65753858 23.61202077 25.17866136
24.88065447 21.07941283 23.08087638 25.26136767 23.04286191 21.22100491
23.01638702 24.14304023 24.88733398 20.77798658 23.0822955 24.57686247
24.44874565 20.90521233 22.77655629 23.66630873 24.84073306 20.10154866
22.21445321 22.99874535 24.4021742 21.2156125 21.78063383 23.03481002
22.86192686 20.66129623 20.70940526 21.49623116 22.57157765 18.80870933
19.54534457 21.46301157 20.71153091 17.1935571 ]
测量温度最小值是: 14.907375312774263
测量温度最大值是: 25.26136766669589

```

4. 读取伪彩色温度图像

```

1     for i in range(1000):
2         start = time.time()
3         clear_output(True)
4         x_img = sensor.sensor_get_img(block_size=15)
5         # 由于传感器测量的温度范围是-22 到 300， 所以要进行一定程度的放大才
6         # 方便观察
7         ps.CommonFunction.show_img_jupyter(x_img * 5)
8         #     time.sleep(0.1)
9         end = time.time()

```

结果如下：



5. 将伪彩色热力图与光学图像拼合

```

1     for i in range(50):
2         start = time.time()
3         clear_output(wait=True)
4         imgMat = cam1.read_img_ori()

```

```
5      tempImg = cv2.resize(imgMat, (320,240))
6
7      # 显示图像
8      #     display(img)
9      end = time.time()
10     #     print(end - start)
11     t_mon = sensor.sensor_read()[3]
12     x_img = sensor.sensor_get_img(block_size=15)
13     image_black = np.zeros_like(tempImg)
14     x_img_rot = np.rot90(x_img)
15     x_img_rot = np.rot90(x_img_rot)
16     x_img_rot = np.rot90(x_img_rot)
17     image_black[:, 130:190, :] = x_img_rot
18     img_add = cv2.addWeighted(tempImg, 0.8, image_black, 5, 0)
19
20     font=cv2.FONT_HERSHEY_SIMPLEX
21     cv2.putText(img_add,str(t_mon + 10),(110,60), font, 1,(255,255,255)
22 ,2)
23     img = ps.CommonFunction.show_img_jupyter(img_add)
24     time.sleep(0.1)
```

结果如下：



第三十九章 Extend 7 - 热成像仪 - Lepton 3.5

内容提要

- 模块介绍
- 使用教程
- 函数介绍

39.1 模块介绍

Lepton 3.5 和上一章节介绍的 mlx90621 十分相似，也是一种通过测量红外辐射来测量（看到）物体的温度，但是它的分辨率大很多，有 160*120。我们的热成像仪模块如下图所示：



这个模块的主要参数如下：

1. 工作温度：-40° 到 85°
2. 测量温度：-10° 到 400°
3. 更新频率：最高 7Hz（我们提供的例程大概 3 帧）
4. 通信协议：I2C 配置，spi 传输数据

 **笔记** 测量温度比工作温度高的原因是非接触，通过看的方式测量

39.2 函数介绍

```
1 # 类名：Lepton_helper  
2 #  
3 # 构造函数：Lepton_helper
```

```

4 # - i2c, Powersensor 的 i2c 对象
5 # - spi, Powersensor 的 spi 对象
6 #
7 # 指令函数: command(moduleId, commandId, comond),
8 # - moduleId
9 # - commandId
10 # - comond
11 # - 返回值, 无
12 #
13 # 寄存器读取函数: read_reg(cmd),
14 # - cmd, 必须, 指令
15 # - 返回值, 指令处理结果
16 #
17 # 指令结果读取函数: read_data(),
18 # - 无参数
19 # - 返回值, 指令执行结束后的结果
20 #
21 # 以上3个函数提供给专业用户使用
22 #
23 # 读取热成像图像数据: sensor_read_frame()
24 # - 无参数
25 # - 返回值, 二维数组, 整形, 14位ad采样结果;

```

39.3 使用教程

1. 包含库文件

```

1 from smbus2 import i2c_msg
2 import PowerSensor as ps
3 import numpy as np
4 import time
5 import matplotlib.pyplot as plt
6 import cv2
7 from IPython.display import clear_output
8 # 重点要包含LeptonHelper模块
9 import LeptonHelper
10

```

2. 初始化

```

1 # 1. 初始化i2c, 即准备用于操作红外测温传感器的i2c
2 i2c = ps.I2cPort()
3 # 2. 实例化红外测温传感器的对象
4 lepton = LeptonHelper.Lepton_helper(i2c=i2c, spi=spi)
5 # 4. 初始化摄像头
6 cam1 = ps.ImageSensor()
7

```

3. i2c 配置，这个接口留给阅读 lepton 技术文档的专业用户使用

```

1  lepton.command(LeptonHelper.SYS, 0x28 >> 2 , LeptonHelper.GET)
2  res_temp = lepton.read_data()
3  res = []
4  res_len = len(res_temp)
5  for i in range(res_len):
6      res.append(ord(res_temp[i]))
7  print(res)
8

```

结果如下：

2 i2c配置，留给专业用户的接口

```

[4]: 1  lepton.command(LeptonHelper.SYS, 0x28 >> 2 , LeptonHelper.GET)
2  res_temp = lepton.read_data()
3  res = []
4  res_len = len(res_temp)
5  for i in range(res_len):
6      res.append(ord(res_temp[i]))
7  print(res)
8
[191, 221, 232, 255, 169, 249, 219, 95, 17, 120, 217, 235, 218, 246, 235, 182, 159, 205, 151, 209, 143, 240, 237, 43, 68, 98, 248, 59, 2
80, 231, 245, 174]

```

图 39.1: I2C 配置效果图

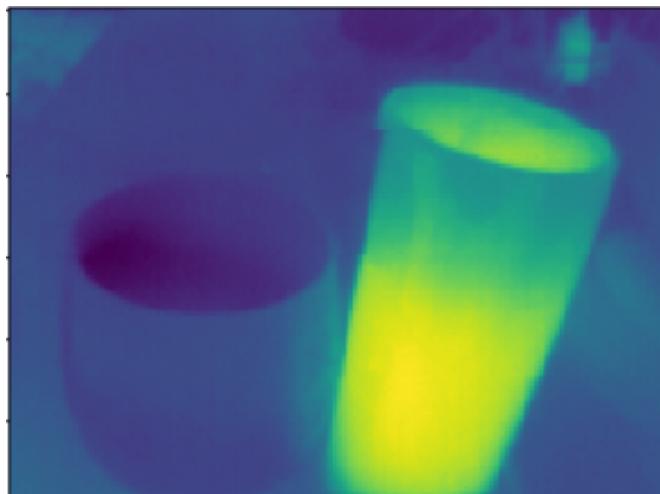
4. 读取温度图像

```

1  img = lepton.sensor_read_frame()
2  plt.imshow(img)
3

```

结果如下：



5. 对比伪彩色图片和光学图片

```

1  img = lepton.sensor_read_frame()
2  plt.imshow(img)
3  plt.show()
4  img2 = cam1.read_img_ori()
5  ps.CommonFunction.show_img_jupyter(img2)
6

```

结果如下：

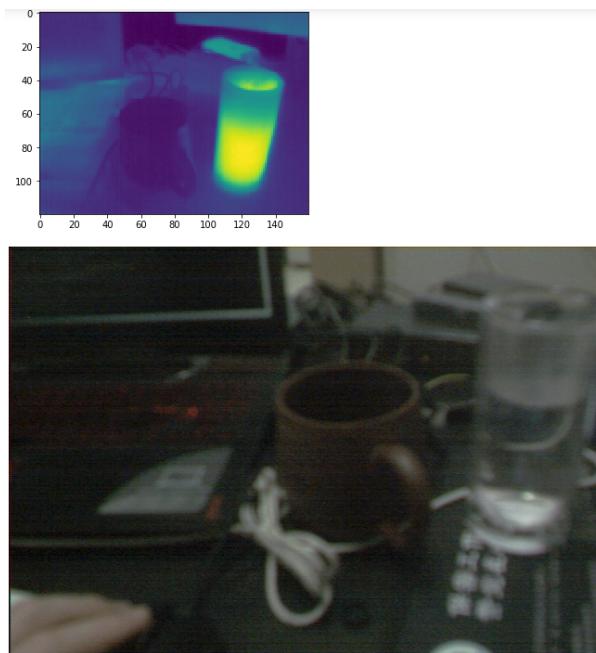


图 39.2：左边是冷杯子，右边是有热水的杯子

第四十章 Extend 8 - 2.8 寸液晶屏 - ili9341

内容提要

❑ 液晶屏

❑ spi

40.1 原理介绍

我们提供的液晶屏模块是 2.8 寸的彩色屏幕，这块拓展板提供 TFT 显示功能，分辨率是 320×240 ，通信协议是 spi。spi 协议在接口那一章已经介绍过了，主要由一根片选线、一根时钟线和一根数据线组成。我们使用的 spi 协议的通信速度是 20M，tft 总共由 320×240 共 76800 个点，每个点采用 rgb565 的格式传输颜色信息，即 16bit，因此，只传输数据的理论最大速度是 $20M \div 76800 \div 16$ ，大概 16 帧。实际上考虑片选线的建立时间、指令字消耗、包间间隔损耗，全分辨率的刷新大概 8 帧左右。

💡 笔记 2021 年以前购买的 powersensor 要修改后才能使用液晶屏，详情见链接：<http://bbs.jingliankeji.com/forum.php?mod=viewthread&tid=17&extra=page%3D1>

我们的液晶屏模块如下图所示：

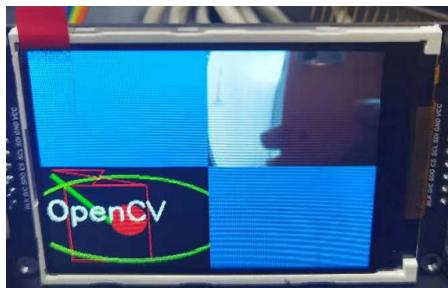


图 40.1: TFT 显示屏

这个模块的主要参数如下：

1. 尺寸： $(50 \times 80 \times 3.6)mm$
2. 电源和功率： $5v \times 100ma$
3. 更新频率：最高 100Hz 左右
4. 可视范围： 270°

我们提供的接口函数的坐标定义如图40.2所示，就是说，横屏，以左上角为原点，向右和向下为正方向作图。画图的原理是，指定一个起点坐标，然后沿正方向（向右和向下），画一个符合要求的矩形区域的图像。这里的符合要求是，计算得到的起点和终点坐标必须在屏幕以内 ($0 \leq x \leq 320 \& \& 0 \leq y \leq 240$)。输入的图像可以是彩色的 rgb 图像，即 opencv 的 mat 类型，或者单色的黑白图像，这里的黑白图像只允许 0 和 255 两种灰度值。

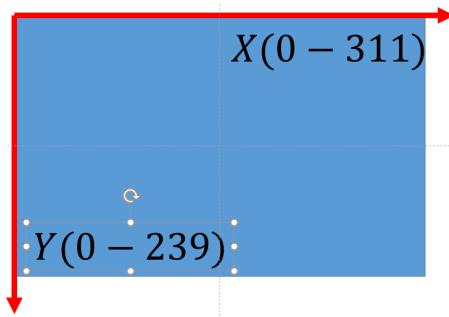


图 40.2: TFT 显示屏的坐标系

注 使用 120x160 的分辨率进行小区域的刷屏可以有效提高帧率。

40.2 函数介绍

```

1 # 类名: Ili9341_helper
2 #
3 # 构造函数: Ili9341_helper(spi, dc)
4 # - spi, Powersensor 的 spi 对象
5 # - dc, Powersensor 的 gpio 对象, 用于控制 IO 管脚
6 #
7 # 绘图函数: LCD_Draw(sx, sy, img_data, color=0xff00),
8 # 这个函数在指定的起点开始, 填充一个小图片
9 # - sx, 整形, 起点的 x 坐标
10 # - sy, 整形, 起点的 y 坐标
11 # - img_data, 整形的 2 维, 3 维数组, opencv 的 mat 对象; 如果是 3 维图像, 为彩色图
    像; 如果是 2 维图像, 会绘制出 color 指定颜色的单色图像, 所以请保证 img_data 为
    二值化后的图像。
12 # - color, 整形的 16 位, 当 img_data 为二维图像时起作用, 为单色图像的颜色
13 # - 返回值, 无
14 #
15 # 清屏函数: LCD_Clear(color),
16 # - color, 清屏的颜色, 16 位无符号整形, rgb565 格式。在 Lcd_helper 模块中提供了几
    种参考色, WHITE, BLACK, BLUE, BRED, GRED, GBLUE, RED, MAGENTA, GREEN,
    CYAN, YELLOW, BROWN, BRRED, GRAY;
17 # - 返回值, 无

```

40.3 使用教程

1. 包含库文件

```

1 import PowerSensor as ps
2 import time
3 import cv2
4 import numpy as np
5 from IPython.display import clear_output

```

```

6
7     # 除了常用的模块，还需要Lcd_helper模块
8     import Mlx90621 as Mlx90621
9

```

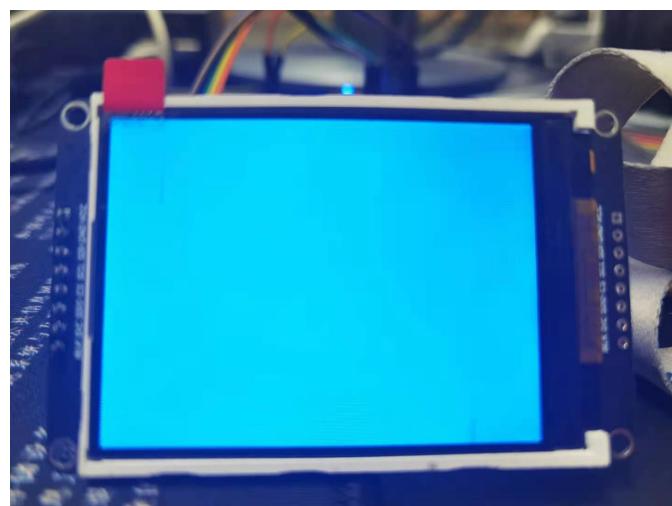
2. 接口、液晶屏对象定义，模块初始化

```

1      # 1. 初始化 IO
2      gpio = ps.GpioPort()
3      gpio.set_mode(0, ps.PortPara.Gpio_Mode_Out)
4      gpio.set_value(0, ps.PortPara.Gpio_Value_Low)
5      gpio.set_mode(1, ps.PortPara.Gpio_Mode_Out)
6      gpio.set_value(1, ps.PortPara.Gpio_Value_High)
7      # 2. 初始化 spi,
8      spi = ps.SpiPort()
9      spi.set_clock(100000000, 1, 1)
10     # 2. 实例化LCD的对象
11     lcd = Lcd_helper.Ili9341_helper(spi=spi, dc=gpio)
12     # 3. 初始化lcd设备
13     lcd.device_init()
14     # 4. 初始化摄像头
15     cam1 = ps.ImageSensor()
16

```

正常初始化后，屏幕会从白屏变成蓝屏：



3. 图片显示测试

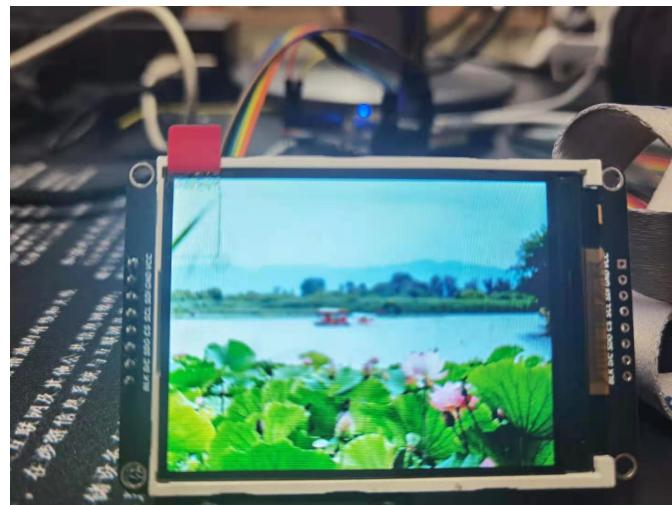
我们在目录下准备了两张图片，用户可以用它来测试显示是否正常：

```

1     img = cv2.imread('/home/debian/bk.jpg')
2     img = cv2.imread('./img/flower.jpg')
3     img = cv2.resize(img, (320,240))
4     lcd.LCD_Draw(0, 0, img)
5

```

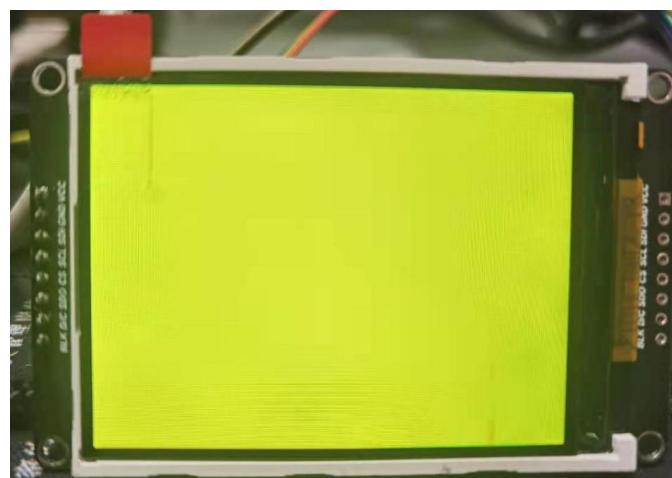
结果如下：



4. 使用 clear 函数可以清屏到指定的颜色：

```
1 lcd.LCD_Clear(Lcd_helper.YELLOW)
2
```

结果如下：



5. 实时显示摄像头拍摄到的图片：

```
1 for i in range(100):
2     clear_output(wait=True)      # 清除图片，在同一位置显示，不使用会打印多
3         张图片
4     t1 = time.time()
5     img = cam1.read_img_ori()
6     img_scale = cv2.resize(img, (160, 120))
7     lcd.LCD_Draw(160, 0, img_scale)
8     #     lcd.display(img_scale)
9     t2 = time.time()
10    time.sleep(0.05)
11    print('tim', t2 - t1)
```

结果如下：



6. 借助 opencv 强大的画图功能，我们可以在液晶屏上的指定区域画出各种花样的图案：

```

1 # 注意numpy定义数组的时候是，先列再行；Opencv里面一般是先宽度再高度；恰好相反
2 canvas = np.zeros([120, 160, 3])
3 tempImg = canvas
4 # 1. 在tempImg图像上划线，从(10,10)到(200,200)坐标，绿色，3个像素宽度
5 cv2.line(tempImg, (10,10), (70,70), (0,255,0),3)
6
7 # 2. 在tempImg图像上画矩形，(10,10)为左上顶点，(30,40)为右下顶点
8 cv2.rectangle(tempImg, (20,20), (100,110), (0,0,255),1)
9
10 # 3. 画圆，参数依次为圆心、半径、颜色BGR，填充-1或线宽像素n
11 cv2.circle(tempImg, (80,60),18,(0,0,213),-1)
12
13 # 4. 画椭圆_需要输入中心点位置，长轴和短轴的长度，
14 cv2.ellipse(tempImg, (80,60), (100,50),0,0,360, (20,213,79),2) # 椭圆沿逆时针选择角度，椭圆沿顺时针方向起始角度和结束角度
15
16 # 5. 绘制多边形
17 pts=np.array([[10,3],[60,3],[48,19],[98,19]],np.int32) # 数据类型必须是int32
18 pts=pts.reshape((-1,1,2))
19 # 这里 reshape 的第一个参数为-1，表明这一维的长度是根据后面的维度的计算出来的。
20 # 如果第三个参数是 False，我们得到的多边形是不闭合的（首尾不相连）。
21 cv2.polylines(tempImg,[pts],True,(0,0,255),1) # 图像，点集，是否闭合，颜色，线条粗细
22
23 # 6. 添加文字，参数：绘制的文字，位置，字型，字体大小，文字颜色，线型
24 font=cv2.FONT_HERSHEY_SIMPLEX
25 cv2.putText(tempImg, 'OpenCV', (0,60), font, 1,(255,255,255),2)

```

```
26  
27 lcd.LCD_Draw(0, 120, tempImg, debug=True)
```

结果如下：

