## 基于OpenMV的智能人脸识别追踪

## 

## 作者:董润（DuckQucik）

## 

## 

## 人脸识别——热门的计算机技术研究领域

## Email:m18834164553@163.com

目录

[**1.前言** 4](#_Toc529810981)

[**2.环境配置** 4](#_Toc529810982)

[**3.原理简述** 4](#_Toc529810983)

[**3.1.机器视觉与计算机视觉** 4](#_Toc529810984)

[**3.2.OpenMV Cam** 5](#_Toc529810985)

[**3.3.特征脸算法** 10](#_Toc529810986)

[**4.项目各功能描述** 15](#_Toc529810987)

[**4.1.实现人脸识别** 15](#_Toc529810988)

[**4.2.实现人脸识别效验追踪** 16](#_Toc529810989)

[**4.3.实现带舵机的人脸识别效验追踪** 18](#_Toc529810990)

[**5.源码** 18](#_Toc529810991)

[**5.1.人脸识别追踪** 19](#_Toc529810992)

[**5.2.人脸识别效验追踪** 19](#_Toc529810993)

[**5.3.带舵机的人脸识别效验追踪** 20](#_Toc529810994)

[**6.结语及展望** 22](#_Toc529810995)

[**7.参考资料** 22](#_Toc529810996)

[**7.1.英文注解：** 22](#_Toc529810997)

[**7.2.参考文献：** 23](#_Toc529810998)

**1. 前言**

本次讲课题目为**“基于OpenMV的智能人脸识别追踪”**，引入“机器视觉”、“计算机视觉”概念，简要介绍OpenMV和特征脸算法，并在此基础上实现人脸识别效验追踪等等功能。

**2. 环境配置**

操作系统：联想(Lenovo)笔记本电脑(i7-7700HQ) Windows 10 64位 家庭中文版

语言：python3

库：OpenMV库（包括sensor、image、pyb、pid）、micropython库

软件：OpenMV IDE用于OpenMV Cam的代码编写、编译、调节、图像绘制等等功能

硬件：OpenMV Cam M7 - OV7725 、焦距为2.8mm的摄像头、 3D打印舵机 、 3.7V 1000mAh锂电池 、 USB数据线

**3. 原理简述**

**3.1. 机器视觉与计算机视觉**

**3.1.1. 机器视觉（Machine Vision, MV）**

**机器视觉**是指用摄影机和计算机等代替人眼对目标进行识别、跟踪和测量。

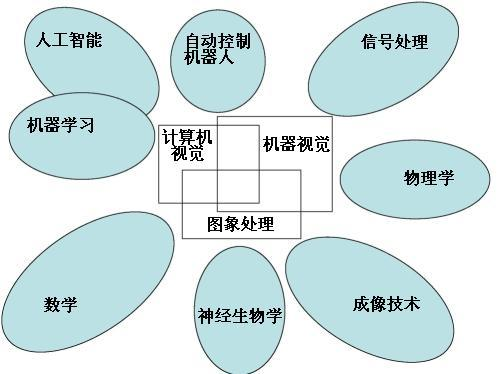
**3.1.2. 计算机视觉(Computer Vision, CV）**

**计算机视觉**是利用计算机和其辅助设备来模拟人的视觉功能，实现对客观世界的三维场景的感知、识别和理解。

**3.1.3. MV与CV的区别**

**计算机视觉**的研究对象主要是**映射到单幅或多幅图像上的三维场景**，例如三维场景的重建。

**机器视觉**主要是指**工业领域的视觉研究**，例如自主机器人的视觉，用于检测和测量的视觉。

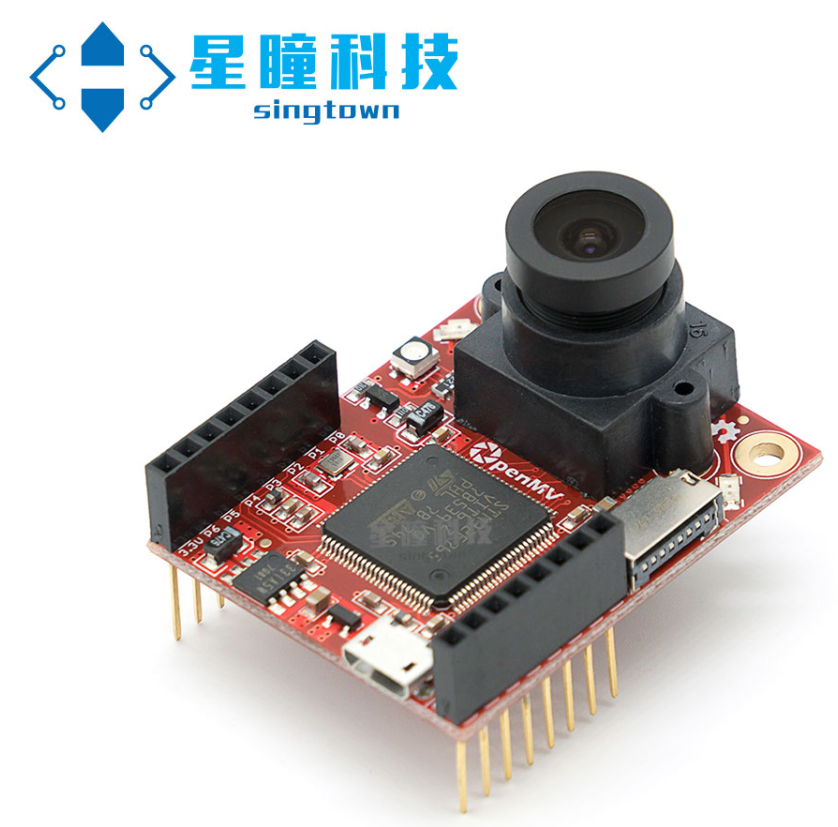


***总的来说，机器视觉偏向硬件，计算机视觉偏向软件。***

**3.2. OpenMV Cam**

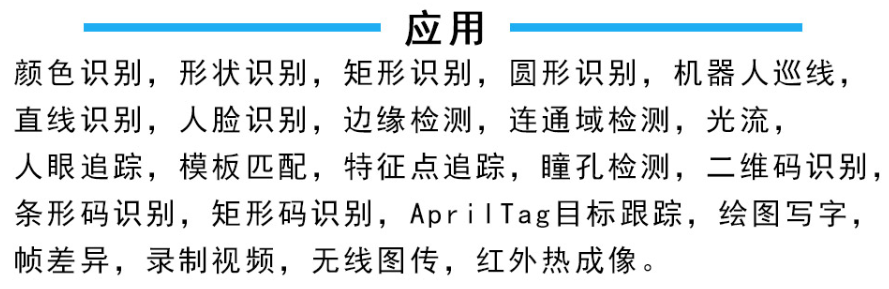
**3.2.1. OpenMV Cam简介**

**OpenMV Cam**是一款低价，可扩展，支持**Python**的**机器视觉模块**，这个模块的目标是成为**“机器视觉世界的Arduino”**。 OpenMV搭载**MicroPython解释器**，这允许你在嵌入式上使用**Python**来编程。

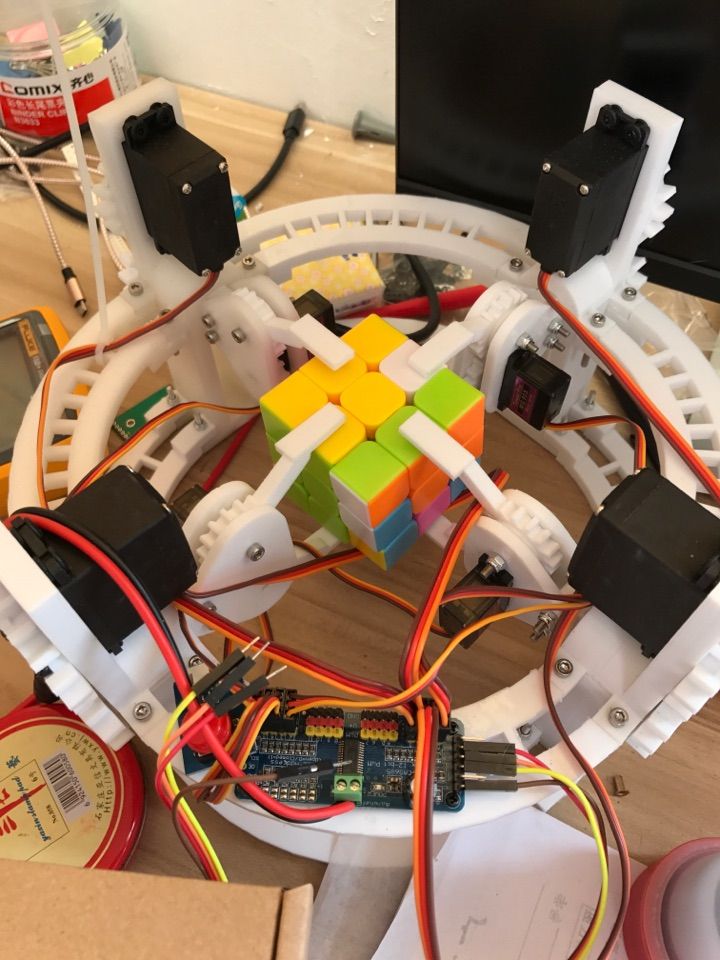


**3.2.2. OpenMV Cam应用**

**常用的有颜色识别、形状识别、二维码识别、人脸识别、边缘检测、特征点追踪等等。**



**解魔方机器人：**



**3.2.3. OpenMV Cam的优点与不足**

优点：

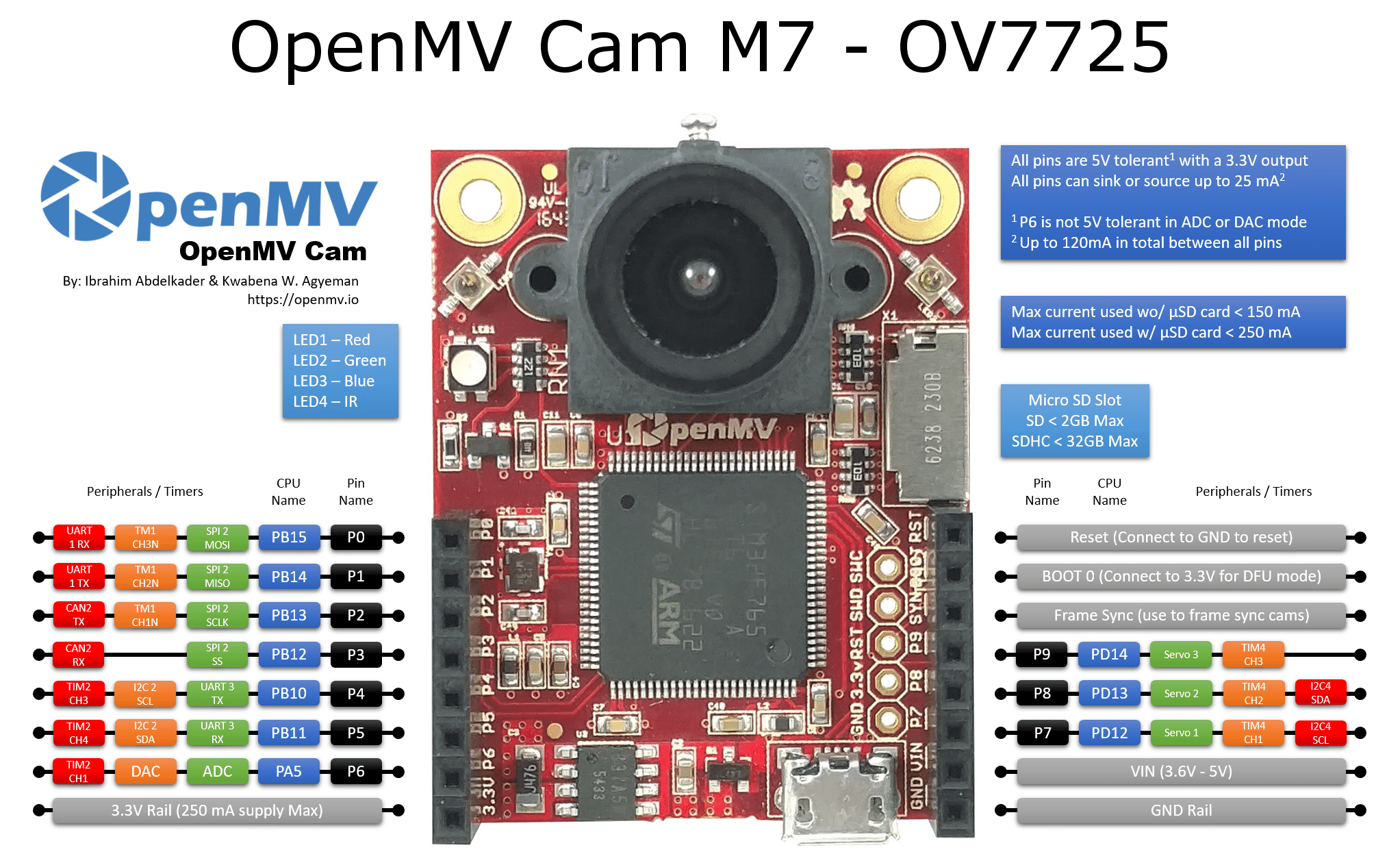
1. 支持Python语法，即Micropython库，即可用python写实现功能的代码。
2. 板子自带解释器且官方自带IDE，编写代码、调试方便。
3. 可以完全控制OpenMV，包括IO引脚。

不足：

1. 内置RAM小，不足以同时处理多张图片。
2. OV7725感光元件最多处理640×480 RGB565彩色图像。
3. 价格对于学生来说偏贵。

**3.2.4. OpenMV Cam M7 - OV7725**

**OpenMV3 Cam M7**是目前**OpenMV系列**性能最优秀的版本。



**参数表：**

|  |  |
| --- | --- |
| **Tables** | **OpenMV3(M7)** |
|  |  |
| Pin | 10 |
| ADC/DAC | 1 |
| SPI | 1 |
| I2C | 2 |
| UART | 1 |
| Servo | 3 |
| CAN bus | 1 |
| IC | STM32F765 |
| RAM | 512KB |
| Flash | 2MB |
| Frequency | 216MHz |
|  |  |

**3.3. 特征脸算法**

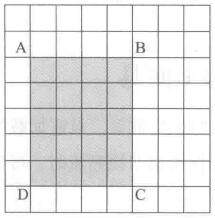
特征脸算法是主流的人脸识别技术，也是具有最好性能的识别方法之一。

**3.3.1. Haar-like特征**

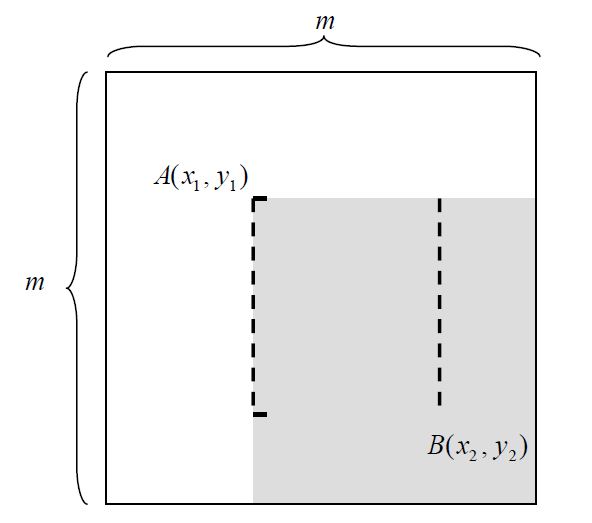
**哈尔特征** 是用于物体识别的一种数字图像特征。**是第一种即时的人脸检测运算**。

哈尔特征最主要的优势是**它的计算非常快速**。使用一个称为**积分图**的结构，任意尺寸的哈尔特征可以在常数时间内进行计算。

**积分图**是类似动态规划的方法，主要的思想是将图像从起点开始到各个点所形成的矩形区域像素之存在数组中，当要计算某个区域的像素和时可以直接从数组中索引，不需要重新计算这个区域的像素和，从而加快了计算。

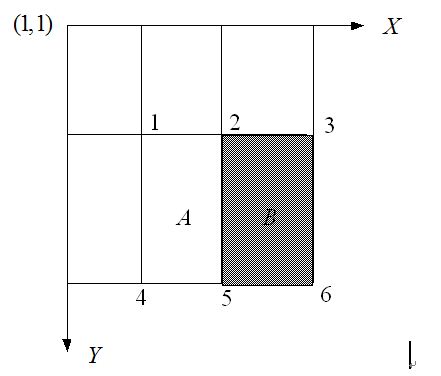


**如图所示的一个m\*m大小的子窗口，可以计算在这么大的子窗口内存在多少个矩形特征。**



**矩形范围(x2-x1,y2-y1)**由左上角点A与右下角点B确定，即两个(x,y)不同的点确定了一个矩形。

**如何计算特征值？**



哈尔特征使用**检测窗口中指定位置的相邻矩形，计算每一个矩形的像素和并取其差值**。**其差值即为特征值**。然后用这些差值来对图像的子区域进行分类。

1. 先计算像素值，即计算点6与(1,1)构成的矩形的像素值为 D6(x6-1,y6-1) ，其它的点分别为 D1、D2等等。

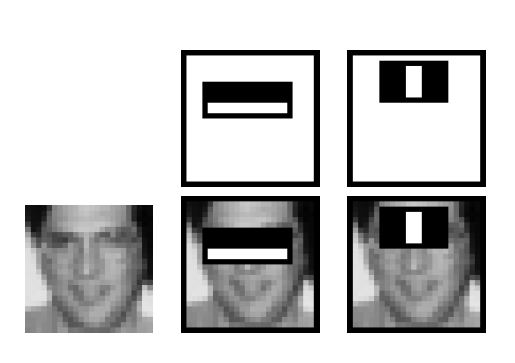
此**矩形B的像素值** 即为**矩形范围B的像素之和Sum\_B = D6 + D2 - D3 - D5**

同理得**矩形A的像素值为Sum\_A = D5 + D1 - D2 - D4**

1. B的特征值即为 **Sum\_A - Sum\_B**

**所以，矩形特征的特征值只与特征矩形端点的积分图有关，而与图像的坐标无关。**

下图中两个矩形特征，表示出人脸的某些特征。

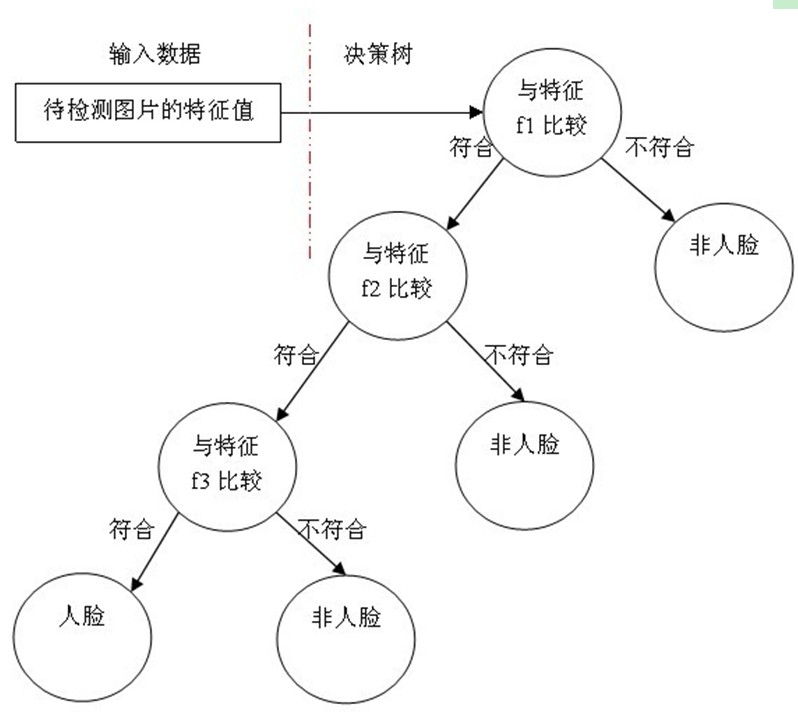


比如中间一幅表示**眼睛区域的颜色比脸颊区域的颜色深**，右边一幅表示**鼻梁两侧比鼻梁的颜色要深**。同样，其他目标，如眼睛等，也可以用一些矩形特征来表示。

**使用特征比单纯地使用像素点具有很大的优越性，并且速度更快**。

**3.3.2. 决策树**

决策树是一个**树结构**（可以是二叉树或非二叉树）。其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。



**4. 项目各功能描述**

**4.1. 实现人脸识别**

因为OpenMV Cam自有特定的库，比如sensor感光元件库、image机器视觉库等等，所以能很方便的调用来实现所需功能，但记住OpenMV Cam因为内存容量问题不支持numpy等等较大的库。

import sensor, image

首先，初始化感光元件，这里为了匹配了摄像头的参数而对感光元件进行必要的设置，比如对比度、相机帧大小、像素模式（灰度图、RGB565彩色图）等等。

sensor.reset()

sensor.set\_contrast(1) #对比度

sensor.set\_gainceiling(16) #图像增益上限

sensor.set\_framesize(sensor.HQVGA) #相机帧大小（分辨率），HQVGA即240x160像素

sensor.set\_pixformat(sensor.GRAYSCALE) #像素模式

其次，加载haar人脸识别算法。

face\_cascade = image.HaarCascade("frontalface", stages=25) #加载人脸模型

最后，拍照，通过照片进行特征分析，如果分析到人脸则在IDE里的帧缓冲图像区画出匹配的方框。

while (True):

img = sensor.snapshot() #拍照

objects = img.find\_features(face\_cascade, threshold=0.75, scale\_factor=1.25) #寻找特征点，匹配人脸

for r in objects:

img.draw\_rectangle(r) #画方框

例图：



**4.2. 实现人脸识别效验追踪**

因为“世界上没有一模一样的树叶”，故每个人的人脸也是不一样的。

人脸效验需要的是在图片中搜寻特征关键点，与匹配人物的特征点进行对比匹配，如果相同的特征点数量超过一定数目，则可以认定是同一个人脸。

我们在人脸识别追踪的代码上进行修改补充。

首先，使用的库不变，设置阈值，并改变相机帧大小。

Threshold = 0.5 #数值越大匹配速度越快但错误率上升

sensor.set\_framesize(sensor.VGA) #设置相机的帧大小

#VGA为640x480像素

sensor.set\_windowing((320,240)) #选取中间区域，防止内存爆满。

设置完后需要稳定摄像头。

sensor.skip\_frames(time = 2000) #跳过2000毫秒的帧

然后，进行匹配人物特征点的获取。

Face1 = None

while(Face1 == None):

img = sensor.snapshot()

objects = img.find\_features(FaceCascade,threshold = Threshold,scale = 1.25)

#寻找与Haar Cascade匹配的图像区域。scale越大，比例因子运行更快，但图像匹配较差。此函数返回列表(x,y,w,h)

if objects:

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

#使用检测区域大小作为ROI提取关键点

Face1 = img.find\_keypoints(threshold = 10,scale\_factor = 1.1,max\_keypoints = 100,roi = face)

#从ROI元组face(x,y,w,h)中提取ORB键点。max\_keypoints为一个键点对象的最大键点数量

最后，进行匹配。如果匹配特征点大于5则匹配成功。

while(True):

img = sensor.snapshot()

Face2 = img.find\_keypoints(threshold = 10, scale\_factor = 1.1, max\_keypoints = 100, normalized = True)

#normalized为布尔值,若为True,在多分辨率下关闭提取键点。设为False则算法运行更快。

if Face2:

ImgMatch = image.match\_descriptor(Face1,Face2,threshold = 85)

#返回Kptmatch特征点对象

ImgMatchNum = ImgMatch[6]

if ImgMatchNum > 5:

img.draw\_rectangle(ImgMatch[2:6])

img.draw\_cross(ImgMatch[0],ImgMatch[1],size = 10)

print(Face2,"matched:dt:%d"%(ImgMatch[7]))

**4.3. 实现带舵机的人脸识别效验追踪**

增加了舵机，便可真正实现追踪效果。 主要是增加了舵机模块。

import pyb,pid #板级功能,舵机库

PanServo = pyb.Servo(1) #舵机 IO口接P7

TiltServo = pyb.Servo(2) #P8

设置舵机灵敏度等参数

PanPid = pid.PID(p=0.11, i=0, imax=90) #p为反应速度，i为稳差

TiltPid = pid.PID(p=0.11, i=0, imax=90)

因为在舵机上接上摄像头后图像在IDE调试时是反着的，故需调节。

sensor.set\_vflip(True) #图像垂直方向翻转

最后当人脸匹配成功时进行舵机运动。注意的是因为图像垂直翻转过，则舵机垂直运动与水平运动函数不一样。

PanError = ImgMatch.cx() - img.width() / 2

TiltError = img.height() / 2 - ImgMatch.cy()

PanOutput = PanPid.get\_pid(PanError,1)

TiltOutput = TiltPid.get\_pid(TiltError,1)

PanServo.angle(PanServo.angle() + PanOutput)

TiltServo.angle(TiltServo.angle() - TiltOutput)

**5. 源码**

**5.1. 人脸识别追踪**

import sensor, image

sensor.reset()

sensor.set\_contrast(1)

sensor.set\_gainceiling(16)

sensor.set\_framesize(sensor.HQVGA)

sensor.set\_pixformat(sensor.GRAYSCALE)

face\_cascade = image.HaarCascade("frontalface", stages=25)

while (True):

img = sensor.snapshot()

objects = img.find\_features(face\_cascade, threshold=0.75, scale\_factor=1.25)

for r in objects:

img.draw\_rectangle(r)

**5.2. 人脸识别效验追踪**

import sensor,image #传感器(摄像头),机器视觉库

Threshold = 0.5 #阈值,数值越大匹配速度越快但错误率上升

sensor.reset()

sensor.set\_contrast(1) #设置对比度

sensor.set\_gainceiling(16) #设置相机图像增益上限

sensor.set\_framesize(sensor.VGA) #设置相机的帧大小

#VGA为640x480像素

sensor.set\_windowing((320,240)) #选取中间区域，防止内存爆满

sensor.set\_pixformat(sensor.GRAYSCALE) #设置相机的像素模式

#GRAYSCALE为灰度图,8位比特每像素

sensor.skip\_frames(time = 2000) #跳过2000毫秒的帧

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

#加载haar人脸模型,stages为Haar Cascade中的阶段数。此函数返回Haar Cascade对象

Face1 = None

while(Face1 == None):

img = sensor.snapshot()

objects = img.find\_features(FaceCascade,threshold = Threshold,scale = 1.25)

#寻找与Haar Cascade匹配的图像区域。scale越大，比例因子运行更快，但图像匹配较差。此函数返回列表(x,y,w,h)

if objects:

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

#使用检测区域大小作为ROI提取关键点

Face1 = img.find\_keypoints(threshold = 10,scale\_factor = 1.1,max\_keypoints = 100,roi = face)

#从ROI元组face(x,y,w,h)中提取ORB键点。max\_keypoints为一个键点对象的最大键点数量

img.draw\_rectangle(objects[0])

img.draw\_keypoints(Face1,size = 24)

img = sensor.snapshot()

while(True):

img = sensor.snapshot()

Face2 = img.find\_keypoints(threshold = 10, scale\_factor = 1.1, max\_keypoints = 100, normalized = True)

#normalized为布尔值,若为True,在多分辨率下关闭提取键点。设为False则算法运行更快。

if Face2:

ImgMatch = image.match\_descriptor(Face1,Face2,threshold = 85)

#返回Kptmatch特征点对象

ImgMatchNum = ImgMatch[6]

if ImgMatchNum > 5:

img.draw\_rectangle(ImgMatch[2:6])

img.draw\_cross(ImgMatch[0],ImgMatch[1],size = 10)

print(Face2,"matched:dt:%d"%(ImgMatch[7]))

**5.3. 带舵机的人脸识别效验追踪**

import sensor,pyb,image,pid #传感器(摄像头),板级功能,机器视觉,舵机库

PanServo = pyb.Servo(1) #舵机 IO口接P7

TiltServo = pyb.Servo(2) #P8

Threshold = 0.5 #阈值,数值越大匹配速度越快但错误率上升

PanPid = pid.PID(p=0.11, i=0, imax=90)#在线调试使用这个PID

TiltPid = pid.PID(p=0.11, i=0, imax=90)#在线调试使用这个PID

sensor.reset()

sensor.set\_contrast(1) #设置对比度

sensor.set\_gainceiling(16) #设置相机图像增益上限

sensor.set\_framesize(sensor.QVGA) #设置相机的帧大小

#VGA为640x480像素

#QQVAG为320×240像素

#QQVAG为160×120像素

#sensor.set\_windowing((320,240)) #选取中间区域，防止内存爆满

sensor.set\_pixformat(sensor.GRAYSCALE) #设置相机的像素模式

#GRAYSCALE为灰度图,8位比特每像素

sensor.set\_vflip(True) #图像垂直方向翻转

sensor.skip\_frames(time = 2000) #跳过2000毫秒的帧

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

#加载haar人脸模型,stages为Haar Cascade中的阶段数。此函数返回Haar Cascade对象

Face1 = None

while(Face1 == None):

img = sensor.snapshot()

objects = img.find\_features(FaceCascade,threshold = Threshold,scale = 1.25)

#寻找与Haar Cascade匹配的图像区域。scale越大，比例因子运行更快，但图像匹配较差。此函数返回列表(x,y,w,h)

if objects:

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

#使用检测区域大小作为ROI提取关键点

Face1 = img.find\_keypoints(threshold = 10,scale\_factor = 1.1,max\_keypoints = 100,roi = face)

#从ROI元组face(x,y,w,h)中提取ORB键点。max\_keypoints为一个键点对象的最大键点数量

while(True):

img = sensor.snapshot()

Face2 = img.find\_keypoints(threshold = 10, scale\_factor = 1.1, max\_keypoints = 100, normalized = True)

#normalized为布尔值,若为True,在多分辨率下关闭提取键点。设为False则算法运行更快。

if Face2:

ImgMatch = image.match\_descriptor(Face1,Face2,threshold = 85)

#返回Kptmatch特征点对象

ImgMatchNum = ImgMatch[6]

if ImgMatchNum > 5:

img.draw\_rectangle((ImgMatch.x(),ImgMatch.y(),ImgMatch.w(),ImgMatch.h()))

img.draw\_cross((int(ImgMatch.cx()),int(ImgMatch.cy())),size = 10)

PanError = ImgMatch.cx() - img.width() / 2

TiltError = img.height() / 2 - ImgMatch.cy()

PanOutput = PanPid.get\_pid(PanError,1)

TiltOutput = TiltPid.get\_pid(TiltError,1)

PanServo.angle(PanServo.angle() + PanOutput)

TiltServo.angle(TiltServo.angle() - TiltOutput)

print(ImgMatch.cx(),ImgMatch.cy())

**6. 结语及展望**

**希望自己能在硬件的路上一直走下去。**

**7. 参考资料**

**7.1. 英文注解：**

1. Machine Vision（MV）：机器视觉
2. Computer Vision（CV）：计算机视觉
3. Cam（Camera）：摄像头
4. OV7725：感光元件型号
5. Pin：I/O引脚（Input / Output）
6. ADC（Analog-to-digital converter）：模拟数字转换器
7. DAC（Digital to analog converter）：数字模拟转换器
8. SPI（Serial Peripheral Interface Bus）：串行外设接口
9. I2C（Inter-Integrated Circuit）：集成电路总线
10. UART（Universal Asynchronous Receiver/Transmitter）：异步串行通信口
11. Servo：这里指的是舵机
12. CAN bus（Controller Area Network）：控制器局域网
13. IC（Integrated Circuit）：集成电路，这里指微芯片，即STM32F765VI ARM CortexM7处理器
14. RAM（Random Access Memory）：随机存取存储器，也叫主存
15. Flash（Flash Memory）：快闪存储器，也叫闪存
16. Frequency：频率

**7.2. 参考文献：**

1. OpenMV Cam官方手册
2. Micropython 中文教程 V2.0
3. 哈尔特征 - 维基百科
4. OV7725 Color CMOS VGA (640x480) CAMERACHIPTM SensorOmniision®with OmniPixel2TM Technology

因参考文献过多，此处不再列举。