2023光电赛设计方案（赛题2）

目录

[2023光电赛设计方案（赛题2） 1](#_Toc132748196)

[视觉 1](#_Toc132748197)

[藏宝图识别 1](#_Toc132748198)

[目标 1](#_Toc132748199)

[配置需求 1](#_Toc132748200)

[实现流程 2](#_Toc132748201)

[代码说明 8](#_Toc132748202)

[常规部署（仅参考） 9](#_Toc132748203)

[定制部署方案1 10](#_Toc132748204)

[图像检测识别 11](#_Toc132748205)

[方案1 11](#_Toc132748206)

[路径规划 17](#_Toc132748207)

[目标 17](#_Toc132748208)

[方案 17](#_Toc132748209)

[演示 17](#_Toc132748210)

[纯算法 17](#_Toc132748211)

[可视化交互 18](#_Toc132748212)

[车辆运动 19](#_Toc132748213)

[运动策略 19](#_Toc132748214)

# 视觉

## 藏宝图识别

### 目标

将拍摄获得的藏宝图进行处理，提取宝藏位置信息

### 配置需求

平台：window,linux

软件环境：python3 +opencv

硬件环境：树莓派、jetson系列、香橙派、nanopi、荔枝派、地平线旭日X3派等等

### 实现流程

0原始图像

图片包含 图示

描述已自动生成

1提取图像边缘信息

图片包含 图形用户界面

描述已自动生成

2从边缘中提取轮廓

文本, 白板

描述已自动生成

3筛选轮廓大小

QR 代码

描述已自动生成

4筛选四边形

图示

描述已自动生成

5筛选正方形

图片包含 白板

描述已自动生成

6筛选定位点

图片包含 图示

描述已自动生成

7图像校正并识别

图示

描述已自动生成

8输出结果

图片包含 图示

描述已自动生成

9动态调参（可选）

图片包含 表格

描述已自动生成

### 代码说明

1参数: change\_parameter

屏幕上有字

描述已自动生成默认为False，如果需要动态调参、中间过程可视化改为True

2识别部分

函数get\_maze\_map\_pose()，返回两个值，第一个是宝藏坐标（10x10坐标）第二个是处理后的图像。传入参数图像frame，display是否展示中间过程

文本

描述已自动生成

### 常规部署（仅参考）

1linux系统下安装python环境

检查是否已有python环境

终端输入python3

出现>>>则说明有python环境

2安装pip包管理工具

sudo apt update

sudo apt install python3-pip

sudo apt install python-pip

pip3 install --upgrade pip

pip install --upgrade pip

3安装opecv

pip3 install opencv-python

pip3 install opencv-contrib-python

4开启终端测试

python3

import cv2

cv2.\_\_version\_\_

如果正常输出版本号则成功

5终端运行detect7.py

python3 detect7.py

出现识别图像

注意：如果没有屏幕，仅在终端运行，需要在代码中关闭图像输出功能，否则会报错

### 定制部署方案1

思路：如果选择stm32，esp32等不支持linux系统的主控，我们的代码不能直接运行在上面。可以选择将图像识别外挂到另一台设备上。通过串口通讯将图像识别结果传回主控。外挂设备可以是自己的电脑、或者其他可以运行图像识别代码的设备上。

以宝藏识别挂载到电脑上为例：

1提供图像识别串口发送代码（以下为部分展示）

def send\_maze(maze\_list):

    txt=str(maze\_list)

    ser.write(txt.encode())

    print("串口发送成功",txt)

2电脑端配置参数，端口号、波特率等

# 打开串口，设置波特率为9600，超时时间为1秒

ser = serial.Serial('COM5', 9600, timeout=1)

3电脑端运行程序开始串口发送

4接收端串口数据解析方案

提供一个在openmv上接收串口信号的代码，数据存储在maze\_location中

uart = UART(3, 9600)

maze\_location=[]

    while True:

        # 读取串口数据

        data = uart.read()

        if data is not None:

            try:

                # 解析数据，转换成列表

                maze\_location=eval("["+data.decode().split("][")[1]+"]")

                if len(maze\_location)==8:

                    print("地图数据接收成功",maze\_location)

                    break

            except:

                pass

        time.sleep(2)

接收端需要配置端口号，波特率与电脑端一致

每隔2s读取一次串口接收的数据，以”][”为分割符号，分割字符串，选取分割列表中任意一个元素即可得到坐标

## 图像检测识别

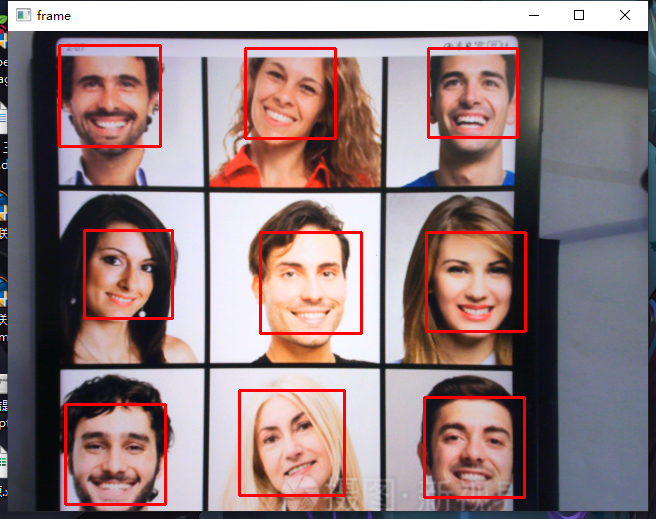
### 方案1

#### 1思路：

opencv级联分类器

针对没有配置机器学习环境的设备，我们使用opencv中的级联分类器来识别图像中的宝藏，速度快可以实现实时监测。

这是一个使用人脸监测模型的目标监测demo，能准确识别目标在图像中的位置。



使用方法非常简单，只需要准备一个分类器模型文件cascade.xml即可，不过这个模型文件需要自己训练。步骤比较繁琐。为了简化步骤。这里提供一个自动训练的脚本。tarin.py

#### 2如何训练模型

注意！每个模型只能监测一种物体

1首先准备一个文件夹任意命名，在这个文件夹下创建两个子文件夹分别为pos和neg

图形用户界面, 文本, 应用程序

描述已自动生成

#### 2待识别图片

将待识别的图像储存在pos中，注意图像要只包含被识别目标，不能有多余的物体

图形用户界面, 应用程序

描述已自动生成

#### 3背景图片

将背景图像储存在neg中，注意一定不要包含被识别物体。

这个背景图片是任意的，不做演示

注意：识别桌子上的苹果，pos文件夹中只能放苹果，需要裁剪。neg中放桌子的图片。图片格式不限。

#### 4配置训练脚本

打开train.py修改

opencv\_createsamples="D:\\Final\\2023guangdian\\opencv\_train\\opencv341\_bin\\opencv\_createsamples.exe"

opencv\_traincascade="D:\\Final\\2023guangdian\\opencv\_train\\opencv341\_bin\\opencv\_traincascade.exe"

这两个文件夹下。

文件地址需要双斜杠！这两个文件在opencv\_341文件夹内







修改第一个参数，改为图片为文件夹地址。

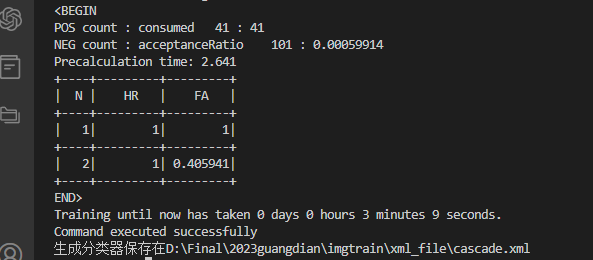
#### 5 运行文件开始训练

文本

描述已自动生成

文本

描述已自动生成



训练完成后会在目标文件夹下生成一个xml\_file文件夹，其中cascade.xml就是我们需要的模型文件

表格

描述已自动生成

应用程序, 表格

描述已自动生成

#### 6使用模型进行预测

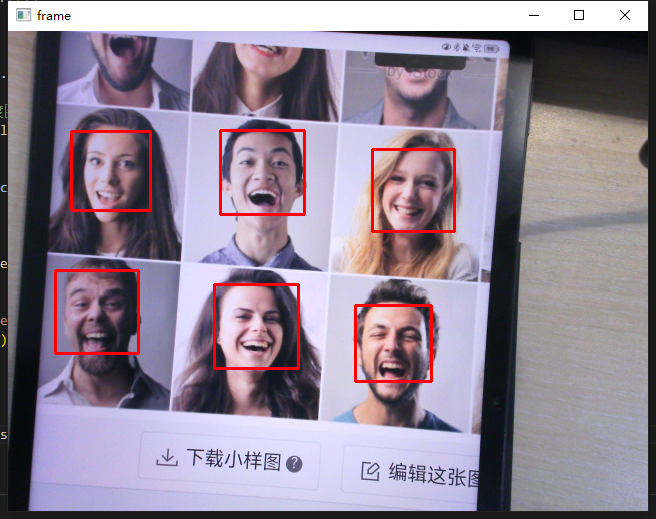
由于不同的摄像头拍摄的宝藏情况不同，训练的模型文件每个人都不一样。所以暂将官方提供的人脸模型作为我们训练好的模型使用作为演示，文件名为haarcascade\_frontalface\_default.xml，已经放在文件夹中了。

打开demo.py文件。将刚才训练好的模型xml文件地址填入该位置。

文本

描述已自动生成

运行该demo文件，即可显示效果



这是我训练的手柄识别模型，由于图片太少，效果比较不稳定，与官方的人脸监测模型效果有差距，但是图片多了效果应该就好了



# 路径规划

## 目标

设计一条经过所有目标点的最短路径

## 方案

预先计算每个点到其他点的最短路径长度，并使用A\*算法来计算点与点之间的距离，设计一条最短路径。由于中间点个数有限，可以进行全部计算得到最优解。具体实现上，使用全排列的方法枚举中间点的顺序，计算经过这些点的最短路径。最后返回最短路径长度和路径本身。该代码的时间复杂度为O(n!)，但实际中可以使用预先计算的路径长度字典来加速计算。

## 演示

### 纯算法

打开path\_plan.py

文本

描述已自动生成

最后一行 start起始点、end目标点、mids中间点，map1是地图矩阵，不要动。

电脑屏幕的照片

低可信度描述已自动生成

输出路径点

### 可视化交互

以下是实现效果：

提供可交互式的使用demo，Visualization\_of\_path\_planning.py，运行，左键选点，右键清空，不要超过10个点，否则会计算量过大卡住。

QR 代码

描述已自动生成

左键选择路径点，右键清空

演示视频[2023光电赛路径规划\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1Ds4y1P7rr/?vd_source=e35ce87e928881f97c8ab5e89ddcd0bf" \l "reply160844348384)，输出路径点

背景图案

描述已自动生成

# 车辆运动

## 运动策略

路径规划完毕后，计算出每个多岔路口需要执行的动作即可

只需要关注地图中的多叉路口，计算目标路径中遇到多叉路口需要执行的动作。其他位置巡线行驶即可。算法集成在main中了。

需要自己完成底层控制，参考代码注释自行完善代码。

    for i in range(len(turn\_points)-1):

        action=turn\_direction(now\_dir, turn\_points[i], turn\_points[i+1])

        now\_dir=turn\_points[i+1][0]-turn\_points[i][0],turn\_points[i+1][1]-turn\_points[i][1]

        if turn\_points[i] in cross\_points:# or action=="Reverse direction":

            print(i,turn\_points[i],action)

            action\_list.append(action)

    action\_list=[]#遇到多叉路口执行动作的列表"right" "left" "straight" "Reverse direction"

    print("开始寻线行驶")

    move\_with\_one\_line()

    #stm32识别到多叉口

    #先把小车头朝迷宫放置。

    for action in action\_list:

        print("到达岔路口选择动作",action)

        car\_move(action)#多叉口路口选择方向

        color=move\_with\_one\_line()#寻线行驶返回条件有两个 1到达新的岔路口 2识别到宝藏

        if color==0:#没有识别到宝藏

            continue

        elif color==1:#己方宝藏

            pass

            attactk()#碰撞动作

            car\_move("Reverse direction")#执行掉头动作

        else:#对方宝藏

            car\_move("Reverse direction")#执行掉头动作

            pass

    #到达终点

    print("到达终点结束运行")

该方案算法方便移植，简单可靠性强

最终能到达终点

文本

描述已自动生成

给出了到达岔路口位置需要执行的动作