# 人脸识别智能门锁

## 设计方案：

### 硬件部分：

嵌入式板卡：树莓派（鉴于树莓派的社区较为完善，能够找到足够多的项目作为案例进行参考，同时也能找到大量成熟的驱动与环境的配置；同时树莓派拥有足够的性能以支持程序的运行）

摄像头：CSI摄像头（CSI是一种串行摄像头接口，通常用于嵌入式系统和移动设备）

整体结构如图一所示：



图 1 硬件结构

### 软件部分：

要完成人脸识别功能，软件部分主要分为四个步骤：

1. 人脸采集

人脸采集调用摄像头即可

1. 数据预处理
2. 人脸检测
3. 人脸识别

这里我们使用python的opencv，opencv提供了一些关于人脸检测和人脸识别的功能；

代码流程主要分为三个部分：

1. 初始化；图2详述了初始化流程
2. 人脸录入功能；图3详述了人脸录入功能
3. 人脸识别功能；图4详述了人脸识别功能



图 2 初始化流程



图 3 人脸录入功能



图 4 人脸识别功能

## 代码实现：

import cv2

import numpy as np

import os

import shutil

import threading

import tkinter as tk

from PIL import Image, ImageTk

# 首先读取config文件，第一行代表当前已经储存的人名个数，接下来每一行是（id，name）标签和对应的人名

id\_dict = {} # 字典里存的是id——name键值对

Total\_face\_num = 0 # 已经被识别有用户名的人脸个数，默认设为0

def init(): # 将config文件内的信息读入到字典中

f = open('config.txt')

global Total\_face\_num

Total\_face\_num = len(f.readlines())

for i in range(int(Total\_face\_num)):

line = f.readline()

id\_name = line.split(' ')

id\_dict[int(id\_name[0])] = id\_name[1]

f.close()

init()

# 加载OpenCV人脸检测分类器Haar

print("load face cascade\n")

face\_cascade = cv2.CascadeClassifier("./haarcascade\_frontalface\_default.xml")

# 准备好识别方法LBPH方法

print("load lbph\n")

recognizer = cv2.face.LBPHFaceRecognizer\_create()

# 打开标号为0的摄像头

camera = cv2.VideoCapture(0) # 摄像头

success, img = camera.read() # 从摄像头读取照片

W\_size = 0.1 \* camera.get(3)

H\_size = 0.1 \* camera.get(4)

system\_state\_lock = 0 # 标志系统状态的量 0表示无子线程在运行 1表示正在刷脸 2表示正在录入新面孔。

# 相当于mutex锁，用于线程同步

def Get\_new\_face():

print("正在从摄像头录入新人脸信息 \n")

# 存在目录data就清空，不存在就创建，确保最后存在空的data目录

filepath = "data"

if not os.path.exists(filepath):

os.mkdir(filepath)

else:

shutil.rmtree(filepath)

os.mkdir(filepath)

sample\_num = 0 # 已经获得的样本数

while True: # 从摄像头读取图片

#debug

print("get img...")

global success

global img # 因为要显示在可视化的控件内，所以要用全局的

success, img = camera.read()

# 转为灰度图片

if success is True:

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

else:

break

# 检测人脸，将每一帧摄像头记录的数据带入OpenCv中，让Classifier判断人脸

# 其中gray为要检测的灰度图像，1.3为每次图像尺寸减小的比例，5为minNeighbors

face\_detector = face\_cascade

faces = face\_detector.detectMultiScale(gray, 1.3, 5)

# 框选人脸，for循环保证一个能检测的实时动态视频流

for (x, y, w, h) in faces:

# xy为左上角的坐标,w为宽，h为高，用rectangle为人脸标记画框

cv2.rectangle(img, (x, y), (x + w, y + w), (255, 0, 0))

# 样本数加1

sample\_num += 1

# 保存图像，把灰度图片看成二维数组来检测人脸区域，这里是保存在data缓冲文件夹内

T = Total\_face\_num

cv2.imwrite("./data/User." + str(T) + '.' + str(sample\_num) + '.jpg', gray[y:y + h, x:x + w])

pictur\_num = 30 # 表示摄像头拍摄取样的数量,越多效果越好，但获取以及训练的越慢

cv2.waitKey(1)

if sample\_num > pictur\_num:

break

else: # 控制台内输出进度条

l = int(sample\_num / pictur\_num \* 50)

r = int((pictur\_num - sample\_num) / pictur\_num \* 50)

print("\r" + "%{:.1f}".format(sample\_num / pictur\_num \* 100) + "=" \* l + "->" + "\_" \* r, end="")

var.set("%{:.1f}".format(sample\_num / pictur\_num \* 100)) # 控件可视化进度信息

# tk.Tk().update()

window.update() # 刷新控件以实时显示进度

def Train\_new\_face():

print("\n正在训练")

# cv2.destroyAllWindows()

path = 'data'

# 初始化识别的方法

recog = cv2.face.LBPHFaceRecognizer\_create()

# 调用函数并将数据喂给识别器训练

faces, ids = get\_images\_and\_labels(path)

print('本次用于训练的识别码为:') # 调试信息

print(ids) # 输出识别码

# 训练模型 #将输入的所有图片转成四维数组

recog.train(faces, np.array(ids))

# 保存模型

yml = str(Total\_face\_num) + ".yml"

rec\_f = open(yml, "w+")

rec\_f.close()

recog.save(yml)

# recog.save('aaa.yml')

# 创建一个函数，用于从数据集文件夹中获取训练图片,并获取id

# 注意图片的命名格式为User.id.sampleNum

def get\_images\_and\_labels(path):

image\_paths = [os.path.join(path, f) for f in os.listdir(path)]

# 新建连个list用于存放

face\_samples = []

ids = []

# 遍历图片路径，导入图片和id添加到list中

for image\_path in image\_paths:

# 通过图片路径将其转换为灰度图片

img = Image.open(image\_path).convert('L')

# 将图片转化为数组

img\_np = np.array(img, 'uint8')

if os.path.split(image\_path)[-1].split(".")[-1] != 'jpg':

continue

# 为了获取id，将图片和路径分裂并获取

id = int(os.path.split(image\_path)[-1].split(".")[1])

# 调用熟悉的人脸分类器

detector = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

faces = detector.detectMultiScale(img\_np)

# 将获取的图片和id添加到list中

for (x, y, w, h) in faces:

face\_samples.append(img\_np[y:y + h, x:x + w])

ids.append(id)

return face\_samples, ids

def write\_config():

print("新人脸训练结束")

f = open('config.txt', "a")

T = Total\_face\_num

f.write(str(T) + " User" + str(T) + " \n")

f.close()

id\_dict[T] = "User" + str(T)

# 这里修改文件的方式是先读入内存，然后修改内存中的数据，最后写回文件

f = open('config.txt', 'r+')

flist = f.readlines()

flist[0] = str(int(flist[0]) + 1) + " \n"

f.close()

f = open('config.txt', 'w+')

f.writelines(flist)

f.close()

def scan\_face():

# 使用之前训练好的模型

for i in range(Total\_face\_num): # 每个识别器都要用

i += 1

yml = str(i) + ".yml"

print("\n本次:" + yml) # 调试信息

recognizer.read(yml)

ave\_poss = 0

for times in range(10): # 每个识别器扫描十遍

times += 1

cur\_poss = 0

global success

global img

global system\_state\_lock

while system\_state\_lock == 2: # 如果正在录入新面孔就阻塞

print("\r刷脸被录入面容阻塞", end="")

pass

success, img = camera.read()

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# 识别人脸

faces = face\_cascade.detectMultiScale(

gray,

scaleFactor=1.2,

minNeighbors=5,

minSize=(int(W\_size), int(H\_size))

)

# 进行校验

for (x, y, w, h) in faces:

# global system\_state\_lock

while system\_state\_lock == 2: # 如果正在录入新面孔就阻塞

print("\r刷脸被录入面容阻塞", end="")

pass

# 这里调用Cv2中的rectangle函数 在人脸周围画一个矩形

cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

# 调用分类器的预测函数，接收返回值标签和置信度

idnum, confidence = recognizer.predict(gray[y:y + h, x:x + w])

conf = confidence

# 计算出一个检验结果

if confidence < 100: # 可以识别出已经训练的对象——直接输出姓名在屏幕上

if idnum in id\_dict:

user\_name = id\_dict[idnum]

else:

# print("无法识别的ID:{}\t".format(idnum), end="")

user\_name = "Untagged user:" + str(idnum)

confidence = "{0}%", format(round(100 - confidence))

else: # 无法识别此对象，那么就开始训练

user\_name = "unknown"

# print("检测到陌生人脸\n")

# cv2.destroyAllWindows()

# global Total\_face\_num

# Total\_face\_num += 1

# Get\_new\_face() # 采集新人脸

# Train\_new\_face() # 训练采集到的新人脸

# write\_config() # 修改配置文件

# recognizer.read('aaa.yml') # 读取新识别器

# 加载一个字体用于输出识别对象的信息

font = cv2.FONT\_HERSHEY\_SIMPLEX

# 输出检验结果以及用户名

cv2.putText(img, str(user\_name), (x + 5, y - 5), font, 1, (0, 0, 255), 1)

cv2.putText(img, str(confidence), (x + 5, y + h - 5), font, 1, (0, 0, 0), 1)

# 展示结果

# cv2.imshow('camera', img)

print("conf=" + str(conf), end="\t")

if 15 > conf > 0:

cur\_poss = 1 # 表示可以识别

elif 60 > conf > 35:

cur\_poss = 1 # 表示可以识别

else:

cur\_poss = 0 # 表示不可以识别

k = cv2.waitKey(1)

if k == 27:

# cam.release() # 释放资源

cv2.destroyAllWindows()

break

ave\_poss += cur\_poss

if ave\_poss >= 5: # 有一半以上识别说明可行则返回

return i

return 0 # 全部过一遍还没识别出说明无法识别

def f\_scan\_face\_thread():

# 使用之前训练好的模型

# recognizer.read('aaa.yml')

var.set('刷脸')

ans = scan\_face()

if ans == 0:

print("最终结果：无法识别")

var.set("最终结果：无法识别")

else:

ans\_name = "最终结果：" + str(ans) + id\_dict[ans]

print(ans\_name)

var.set(ans\_name)

global system\_state\_lock

print("锁被释放0")

system\_state\_lock = 0 # 修改system\_state\_lock,释放资源

def f\_scan\_face():

global system\_state\_lock

print("\n当前锁的值为：" + str(system\_state\_lock))

if system\_state\_lock == 1:

print("阻塞，因为正在刷脸")

return 0

elif system\_state\_lock == 2: # 如果正在录入新面孔就阻塞

print("\n刷脸被录入面容阻塞\n"

"")

return 0

system\_state\_lock = 1

p = threading.Thread(target=f\_scan\_face\_thread)

p.setDaemon(True) # 把线程P设置为守护线程 若主线程退出 P也跟着退出

p.start()

def f\_rec\_face\_thread():

var.set('录入')

cv2.destroyAllWindows()

global Total\_face\_num

Total\_face\_num += 1

Get\_new\_face() # 采集新人脸

print("采集完毕，开始训练")

global system\_state\_lock # 采集完就可以解开锁

print("锁被释放0")

system\_state\_lock = 0

Train\_new\_face() # 训练采集到的新人脸

write\_config() # 修改配置文件

def f\_rec\_face():

global system\_state\_lock

print("当前锁的值为：" + str(system\_state\_lock))

if system\_state\_lock == 2:

print("阻塞，因为正在录入面容")

return 0

else:

system\_state\_lock = 2 # 修改system\_state\_lock

print("改为2", end="")

print("当前锁的值为：" + str(system\_state\_lock))

p = threading.Thread(target=f\_rec\_face\_thread)

p.setDaemon(True) # 把线程P设置为守护线程 若主线程退出 P也跟着退出

p.start()

# tk.Tk().update()

def f\_exit(): # 退出按钮

exit()

window = tk.Tk()

window.title('Cheney\' Face\_rec 3.0') # 窗口标题

window.geometry('1000x500') # 这里的乘是小x

# 在图形界面上设定标签，类似于一个提示窗口的作用

var = tk.StringVar()

l = tk.Label(window, textvariable=var, bg='green', fg='white', font=('Arial', 12), width=50, height=4)

# 说明： bg为背景，fg为字体颜色，font为字体，width为长，height为高，这里的长和高是字符的长和高，比如height=2,就是标签有2个字符这么高

l.pack() # 放置l控件

# 在窗口界面设置放置Button按键并绑定处理函数

button\_a = tk.Button(window, text='开始刷脸', font=('Arial', 12), width=10, height=2, command=f\_scan\_face)

button\_a.place(x=800, y=120)

button\_b = tk.Button(window, text='录入人脸', font=('Arial', 12), width=10, height=2, command=f\_rec\_face)

button\_b.place(x=800, y=220)

button\_b = tk.Button(window, text='退出', font=('Arial', 12), width=10, height=2, command=f\_exit)

button\_b.place(x=800, y=320)

panel = tk.Label(window, width=500, height=350) # 摄像头模块大小

panel.place(x=10, y=100) # 摄像头模块的位置

window.config(cursor="arrow")

def video\_loop(): # 用于在label内动态展示摄像头内容（摄像头嵌入控件）

global success

global img

success, img = camera.read() # 从摄像头读取照片

if success:

cv2.waitKey(1)

cv2image = cv2.cvtColor(img, cv2.COLOR\_BGR2RGBA) # 转换颜色从BGR到RGBA

current\_image = Image.fromarray(cv2image) # 将图像转换成Image对象

imgtk = ImageTk.PhotoImage(image=current\_image)

panel.imgtk = imgtk

panel.config(image=imgtk)

window.after(1, video\_loop)

video\_loop()

# 窗口循环，用于显示

window.mainloop()

## 执行结果：

这里使用了笔记本电脑作为代码执行的测试平台；

图5为采集到的人脸素材，素材由本人亲自提供；

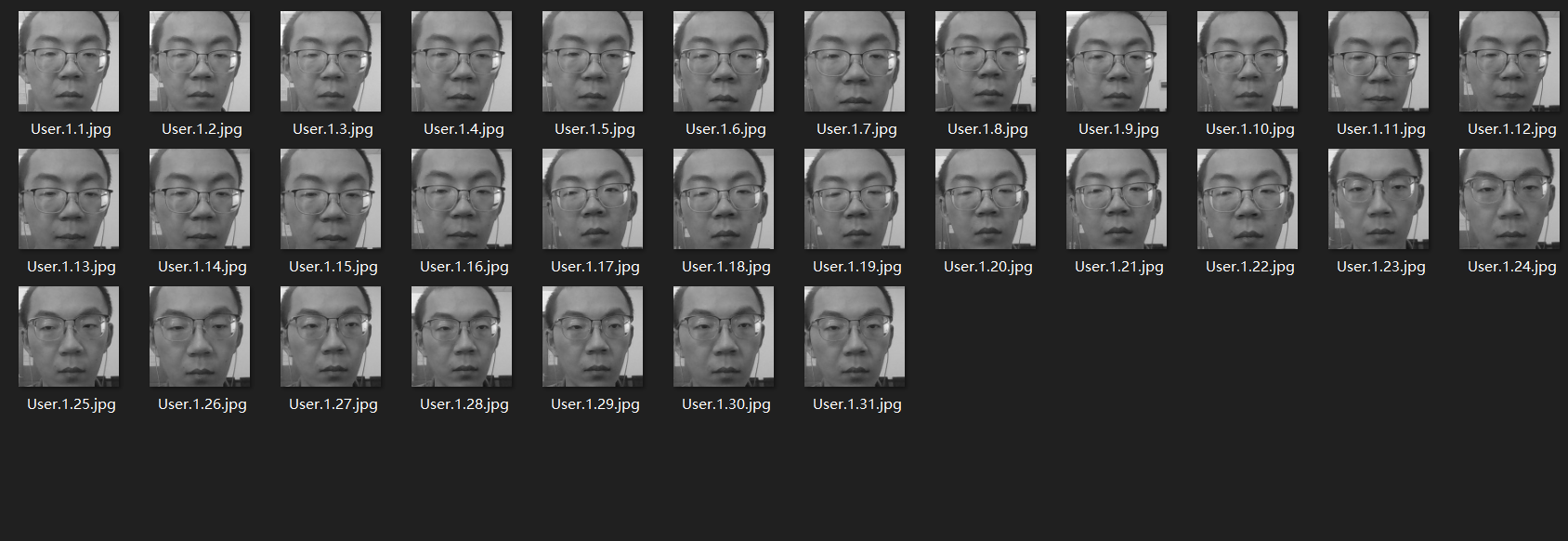


图 5 人脸素材

图6为执行结果：成功识别到人脸，id为User1



图 6 识别结果

## 避免欺诈

人脸识别的伪造攻击，一般由已被录入系统的用户的图片，视频，来欺骗摄像头；由于可能存在视频欺诈，所以单纯的检测用户是否为动态用户无法完全避免欺诈；

我认为比较合理的防欺诈手段为，使用激光摄像头，捕获空间数据，激光摄像头可以获得距离数据，如此可以避免设备被平面数据所欺骗；必须是立体的人脸才能通过检测；