山东理工大学

shandong university of technology

大 作 业 报 告

《机器学习综合实训》

学　　院： 计算机科学与技术学院

专　　业： 人工智能

班　　级： 智能 2302

姓　　名： 陈佳乐

学　　号： 23110570044

二〇二六 年 一 月

得分	
教师签字	

AI 机器视觉智控系统

一、实验目的

掌握人脸识别的基本原理，了解 GPIO 的基本概念和使用方法，掌握中断机制和串口通信，能够使用 python 语言实现人脸识别和对下位机的控制，能够使用 C\C++实现对 STM32 的控制和对上位机的信息交流。具体包括：

1. 通过 opencv 库中自带的人脸识别模型 haarcascade_frontalface_alt2. xml 进行人脸识别，并实现对相应信息进行标注.

2. 利用 python 实现对下位机传来的字节数据的接收和处理，使下位机传过来的数据能够发挥其真正的作用；此外还要实现向下位机发送数据，使得下位机能够对上位机相应的功能作出反应

3. 利用 C\C++实现对串口的通信，是的相应元器件能够对相应的功能做出相应的反应，并实现反馈；此外还要实现对上位机的通信，确保各项指令能够施行

二、实验内容

围绕 “上位机人脸检测 + 下位机硬件控制” 展开，构建一套完整的智控系统。系统分为上位机（Python）和下位机（STM32）两大模块，通过串口通信实现数据交互，最终达成 “人脸检测显示、硬件状态监控、指令双向传输” 的核心功能。

1. 上位机模块设计

上位机承担人脸检测、数据解析与指令下发功能，按功能拆分为 3 个子模块：

（1）人脸检测与显示模块：调用 OpenCV 的 haarcascade_frontalface_alt2. xml 模型，从摄像头实时采集画面，检测人脸并绘制矩形框；同时在画面中叠加显示下位机上报的硬件状态（锁开关状态、环境温度），实现 “视觉 + 数据” 的一体化展示。

（2）串口通信与数据解析模块：通过 serial 库初始化串口（匹配下位机波特率 115200），接收下位机发送的二进制数据；利用 struct 库按 iif（int-int-float）格式解析数据，提取抓拍标记、锁状态、温度值；同时提供 send_cmd 函数，向下位机发送控制指令（如 OpenLock、OpenBuzzer）。

（3）异常处理与资源管理模块：实现串口、摄像头的异常捕获（如串口打开失败、摄像头未连接），避免程序崩溃；程序退出时自动释放摄像头、关闭串口，确保硬件资源正常回收。

2. 下位机模块设计

下位机负责硬件驱动、状态采集与指令执行，按功能拆分为 4 个子模块：

（1）外设驱动模块：初始化 GPIO（控制锁、蜂鸣器、LED）、定时器 TIM16（用于定时任务）、USART1（串口通信）、DHT11 温湿度传感器；其中 GPIO 配置为推挽输出（锁、蜂

鸣器）或上拉输入（锁状态检测），确保硬件正常驱动。

（2）串口中断与指令处理模块：实现 USART1 空闲中断回调函数，接收上位机指令并执行对应动作（如接收 OpenLock 则控制 GPIOB_PIN7 置 1 开锁，接收 OpenBuzzer 则控制 GPIOB_PIN4 置 1 触发蜂鸣器）；同时提供二进制数据打包函数，将抓拍标记、锁状态、温度值按 iif 格式打包后发送至上位机。

（3）状态采集与业务逻辑模块：通过 DHT11 传感器定时读取环境温度，通过 GPIO 读取锁状态（检测 GPIOB_PIN8 电平）；主循环中定时更新系统状态，并触发串口数据上报；同时实现硬件联动逻辑（如温度超限触发蜂鸣器、锁状态变化反馈至上位机）。

（4）中断管理模块：配置外部中断（用于抓拍指令触发）、定时器中断（预留定时任务），通过中断机制提升指令响应速度，避免主循环阻塞导致的功能延迟。

3. 系统交互逻辑

上位机与下位机通过串口建立双向通信，核心交互流程为：

（1）下位机每 3 秒采集一次温度、锁状态，按结构体格式通过 USART1 发送至上位机；

（2）上位机解析数据后，在人脸检测画面中更新锁状态和温度值（显示在人脸框上方）；

（3）上位机下发控制指令（如 CloseLock），下位机中断回调函数接收指令后，控制对应 GPIO 引脚电平，执行关锁动作，并将新的锁状态通过后续数据上报反馈至上位机；

（4）当下位机外部中断触发（如抓拍按键），更新抓拍标记并上报，上位机检测到抓拍标记后，保存当前画面至本地目录。

三、实验步骤

1. 下位机开发与外设配置

先对锁、蜂鸣器、LED 引脚及逆行配置，然后配合时钟进行设置；在串口调试助手里面配置好串口的比特率等信息，同时在代码中配置好相应指令。下面是下位机的核心代码：

```
#include "main.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
#include "dht11.h"
uint8_t humiH;
uint8_t humiL;
uint8_t tempH;
uint8_t tempL;
float temp,humi;
```

```
struct cmd_t
{
    int cap;
    int lock_state;
    float temp;
};
struct cmd_t cmd;
void SystemClock_Config(void);
__weak void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin)
{
  cmd.cap = 1;
}
char buf[32]={0};
int lockstate = 0;
int a=0;
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart,uint16_t
Size){
    HAL_TIM_Base_Stop_IT(&htim16);
    if(buf[0]=='N'){
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, 0);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, 0);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, 0);
    }
    else if(buf[0]=='F'){
        a=1;
    }
    else if(buf[0]=='A'){
        HAL_TIM_Base_Start_IT(&htim16);
    }
    else if(strcmp(buf,"OpenLock",8)==0)
    {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, 1);
    }
    else if(strcmp(buf,"CloseLock",9)==0)
    {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, 0);
    }
    else if(strcmp(buf,"OpenBuzzer",10)==0)
    {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, 1);
    }
    else if(strcmp(buf,"CloseBuzzer",11)==0)
    {
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, 0);
    }
```

```
    memset(buf,0,sizeof(buf));
    HAL_UARTEx_ReceiveToIdle_IT(&huart1,buf,32);
}
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  delay_init(16);
  MX_GPIO_Init();
  MX_TIM16_Init();
  MX_USART1_UART_Init();
  HAL_TIM_Base_Start_IT(&htim16);
  HAL_UARTEx_ReceiveToIdle_IT(&huart1,buf,32);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  cmd.cap = 0;
  while (1)
  {
      DHT11_Read_Data(&humiH,&humiL,&tempH,&tempL);
      cmd.temp = tempH + tempL * 0.1;
      if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_8)==1){
          cmd.lock_state = 1;
          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, 0);
      }
      else{
          cmd.lock_state = 0;
      }
      if(a==1)
      {
          HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_1);
          HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_2);
          HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_0);
      }

      HAL_UART_Transmit(&huart1,(uint8_t*)&cmd, sizeof(struct cmd_t), 200);
      cmd.cap = 0;
      HAL_Delay(1000);
  `
```

2. 上位机的开发与功能实现

调用 opencv 库中自带的人脸识别模型 haarcascade_frontalface_alt2. xml 进行人脸识别，并绘制矩形框并叠加状态信息；此外进行串口初始化，解析下位机发送的数据，同时提供指令下发函数；整合人脸识别与串口通信，实现多任务同时工作。上位机代码如下：

```
import cv2
import time
import serial
import struct
import threading
CMD_LOCK_STATE = 0;
CMD_TEMP_VAL = 1;
CMD_CAP_CTL = 2;
lockstate = "closing"
UART_NAME = 'COM15'
ser = serial.Serial(UART_NAME,115200,timeout=1)
cap = cv2.VideoCapture(0)
flag = 0
class Cmd:
    def __init__(self, cap_ctl, temp_val=0.00, lock_state=0):
        self.lock_state = lock_state
        self.temp_val = temp_val
        self.cap_ctl = cap_ctl
    def from_bytes(data):
        cap_ctl, lock_state, temp_val = struct.unpack('<iif', data)
        return Cmd(cap_ctl, round(temp_val,2), lock_state)
def send_cmd(data):
    if ser.is_open:
        send_data = data.encode('utf-8')
        ser.write(send_data)
def serial_rec():
    if ser.in_waiting > 0:
        d = ser.read(struct.calcsize('iif'))
        cmd = Cmd.from_bytes(d)
        print(cmd.cap_ctl, cmd.lock_state,cmd.temp_val)
        threading.Thread(target=video(cmd), daemon=True).start()
        video(cmd)
def video(cmd):
    ret,src = cap.read()
    gray=cv2.cvtColor(src,cv2.COLOR_BGR2GRAY)
    mode = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_alt2.xml')
    faces = mode.detectMultiScale(src, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))
    if len(faces) > 0:
        if cmd.lock_state != 1:
            send_cmd("OpenLock")
    else:
        global flag
```

```
        print("flag = ",flag)
                flag += 1
                if flag == 1:
                    send_cmd("F")
                elif flag > 5:
                    send_cmd("OpenBuzzer")
                    print("已经超过 5 次未检测到人脸，蜂鸣器已响")
                    time.sleep(0.1)
                    send_cmd("CloseBuzzer")
            if cmd.cap_ctl == 1:
                lock = threading.Lock()
                with lock:
                    last_src = src.copy()
                    if last_src is not None:
                        timestamp = time.strftime("%Y%m%d_%H%M%S")
                        filename = f"capture_{timestamp}.jpg"
                        cv2.imwrite(filename, last_src,
[cv2.IMWRITE_JPEG_QUALITY, 90])
            global lockstate
            if cmd.lock_state:
                lockstate = "opening"
            else:
                lockstate = "closing"
            cv2.putText(src, "The state of lock is "+lockstate+".", (10,30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
            for (x, y, w, h) in faces:
                cv2.rectangle(src, (x, y), (x + w, y + h), (255, 0, 0), 2)
                cv2.putText(src, "Temp:" + str(cmd.temp_val), (x,y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.7,
                                (0, 255, 0), 2)
            cv2.imshow("input",src)
            cv2.waitKey(1)
        def start_time():
            while True:
                serial_rec()
                time.sleep(0.02)
        def main():
            start_time()
        if __name__=="__main__":
            main()
```

四、实验结果分析

开启上位机，运行代码，能够识别到人脸，也能够正确接收并解析下位机传过来的数据（按键状态、锁的状态、温度），但由于时间差，打开摄像头前就已经接收到下位机传的信息，所以接收信息的时间要早于摄像头识别到人脸的时间，当识别到人脸之后还有一个信号传输时间，

所以当识别到人脸之后打印的锁的状态并不是立马变化成 1（也就是打开状态），而是要等待几毫秒之后才显示（如图 1）。

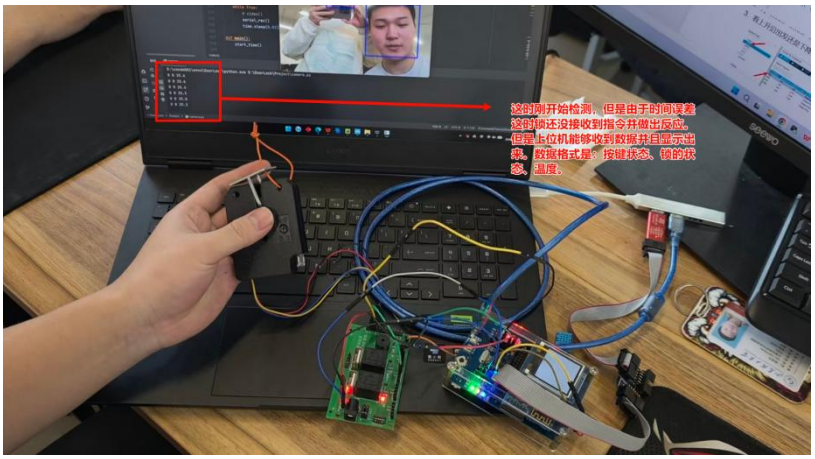

图 1

当识别到人脸之后，上位机发送 OpenLock 指令到下位机，随后锁打开，并且上位机输出的锁的状态置 1（如图 2）。

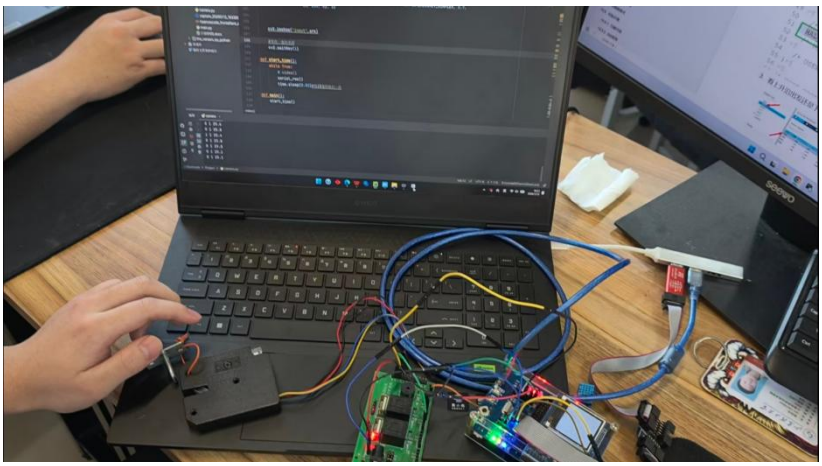

图 2

当按下开发板上的按键之后，按键状态位置 1，并且以图片形式保存当前摄像头状态（如图 3）。

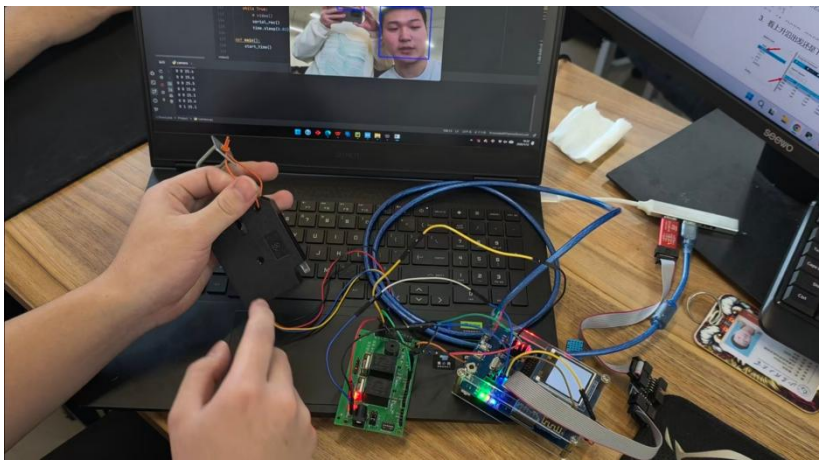

图 3

随后用手掌遮住摄像头模拟识别不到人的情况，当识别一次不成功时，灯闪烁（但是由于图

片的局限性，没有直观显示出 LED 灯的闪烁，在后面的视频当中可以直观看出）；当识别 5 次及以上不成功时，蜂鸣器报警（如图 4）

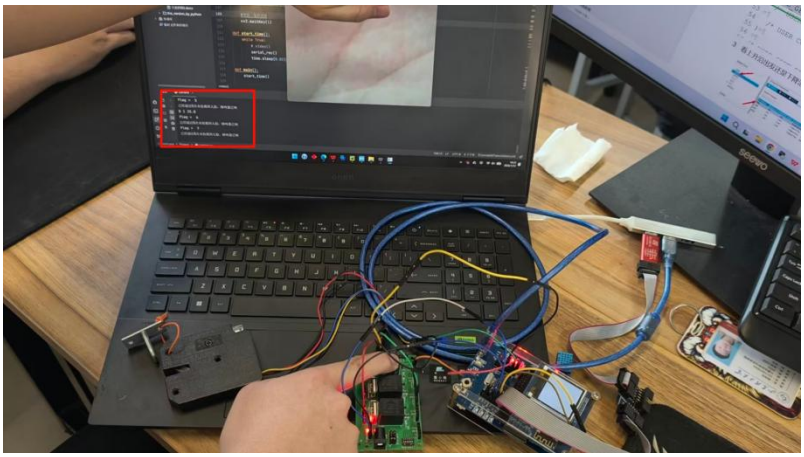

图 4