



有线外设无线化应用 项目设计

本项目通过蓝牙传输,对无线化的摄像头做了优化与提升,利用人脸识别技术,实现外设无线化具体应用与拓展延伸。

3.1 功能及总体设计

本项目外部设备与主机的连接方式由原本的有线连接变为无线连接,解决有线外设连接主机和距离限制的问题。

要实现上述功能需将作品分成四部分进行设计,即有线外设、信号处理与控制部分、信号传输中介和主机。有线外设是已有的普通外部设备,包括有线键盘、有线音箱和有线摄像头;信号处理与控制部分是 Arduino 开发板;信号传输中介是蓝牙模块或路由器;主机是计算机或者手机。其中,信号传输中介是本项目的枢纽所在,它实现了主机与有线外设之间信号的无线传输。

1. 整体框架

整体框架如图 3-1 所示,有线键盘无线化结构框架如图 3-2 所示,有线音箱无线化结构框架如图 3-3 所示,人脸跟踪无线化摄像头结构框架如图 3-4 所示。

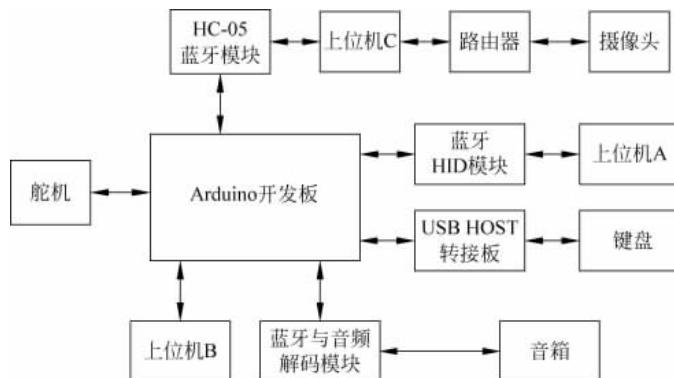


图 3-1 整体框架

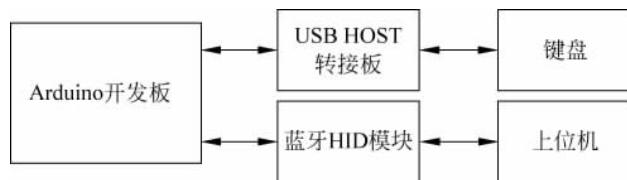


图 3-2 有线键盘无线化结构框架



图 3-3 有线音箱无线化结构框架

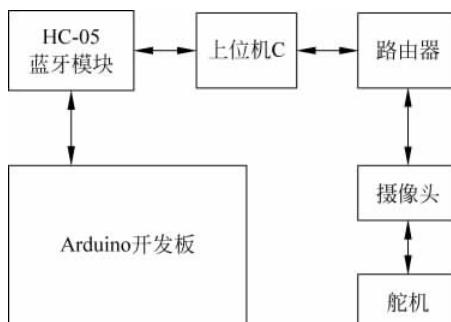


图 3-4 人脸跟踪无线化摄像头结构框架

2. 系统流程

有线键盘无线化系统流程如图 3-5 所示, 有线音箱无线化系统流程如图 3-6 所示, 人脸跟踪无线化摄像头系统流程如图 3-7 所示。

3. 总电路

总电路如图 3-8 所示, 引脚连线如表 3-1 所示。

表 3-1 引脚连线

元件及引脚名	Arduino 开发板引脚	
HID 蓝牙键盘模块	RX	16
	TX	17
HC-05 蓝牙模块	RX	18
	TX	19

续表

元件及引脚名		Arduino 开发板引脚
蓝牙音频解码模块	TX	0
	RX	1
舵机 1	控制引脚	8
舵机 2	控制引脚	9
红外模块	控制引脚	A0

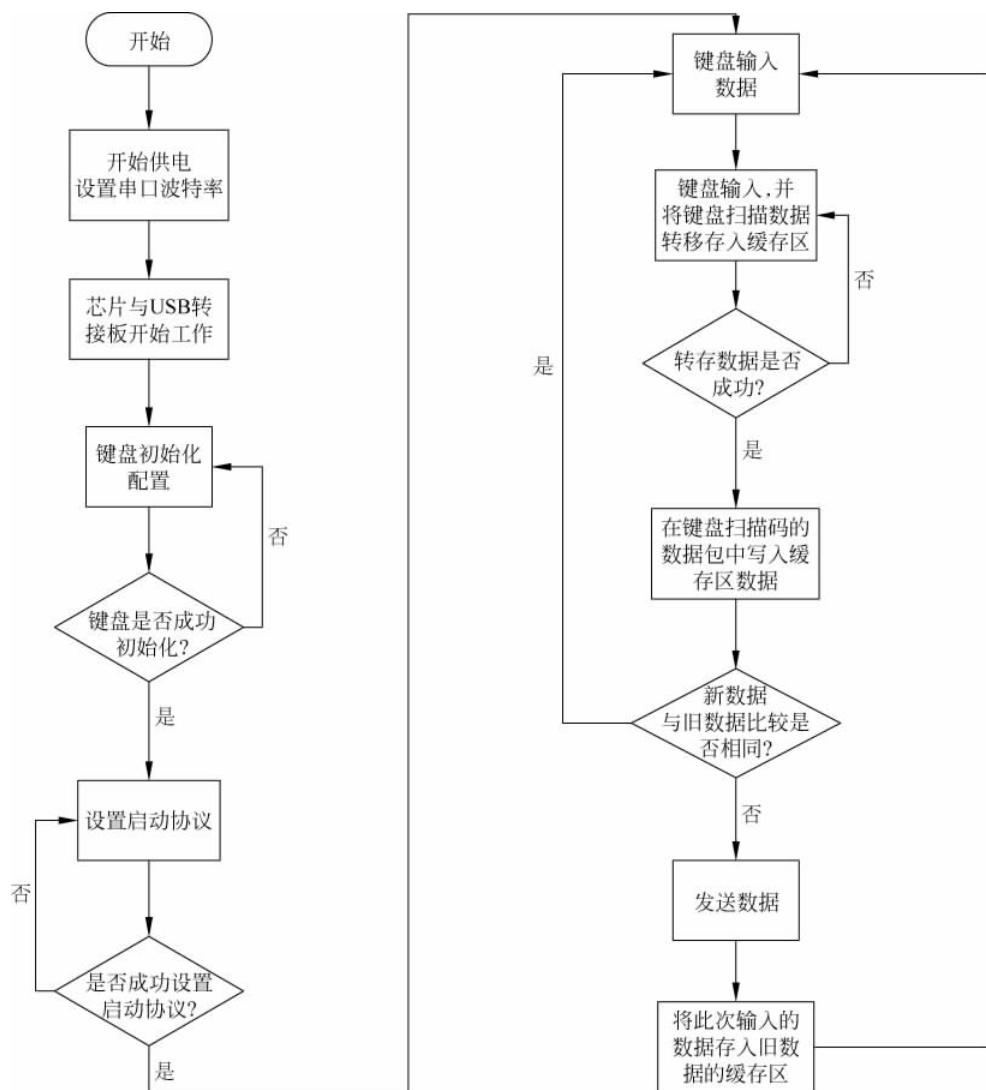


图 3-5 有线键盘无线化系统流程

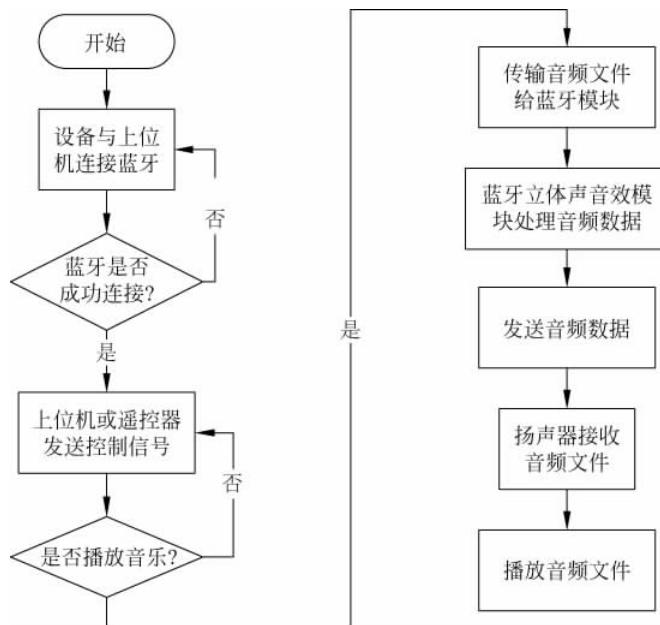


图 3-6 有线音箱无线化系统流程

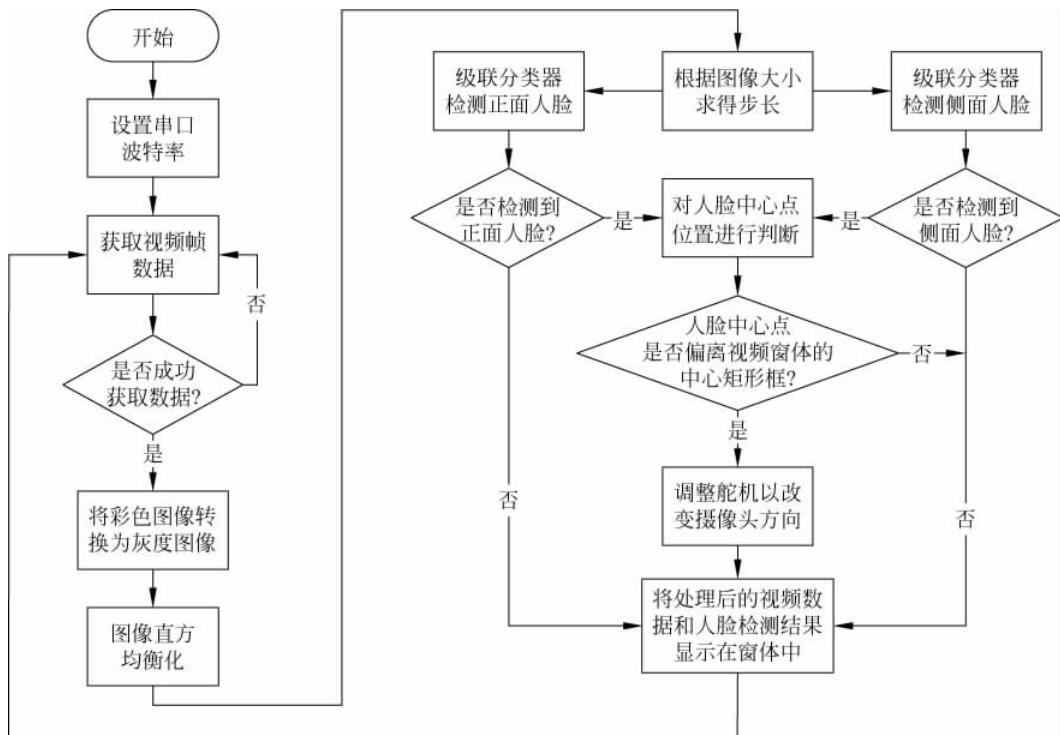


图 3-7 人脸跟踪无线化摄像头系统流程

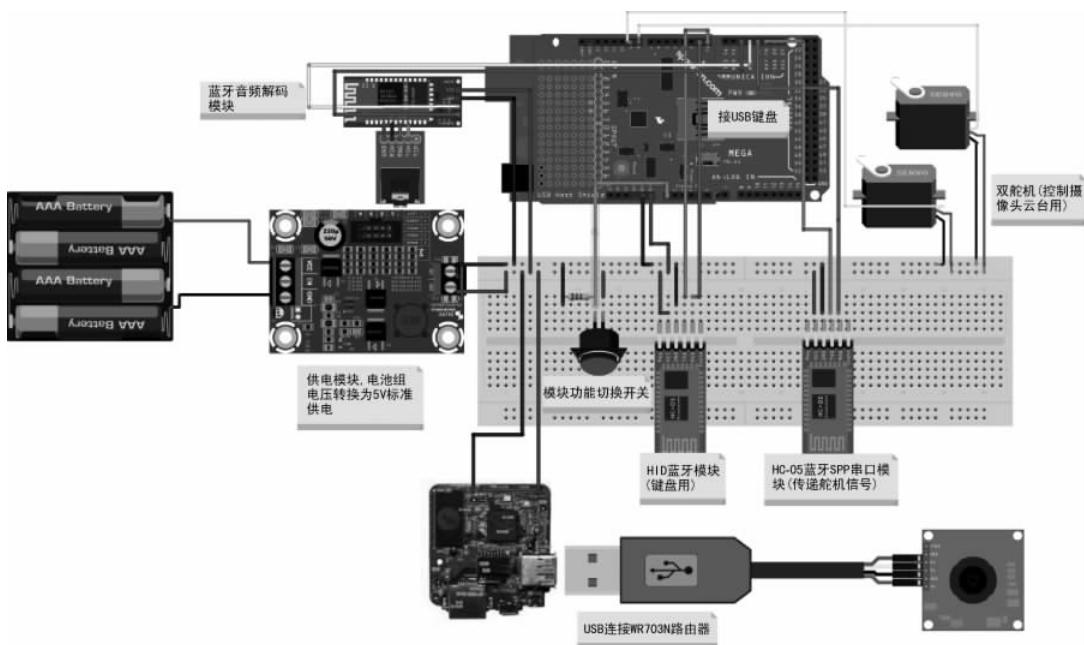


图 3-8 总电路

3.2 模块介绍

本项目主要包括有线键盘无线化模块、有线音箱无线化模块和人脸跟踪无线化摄像头模块，下面分别给出各模块的功能介绍及相关代码。

3.2.1 有线键盘无线化

本节包括有线键盘无线化模块的工作原理、功能介绍及相关代码。

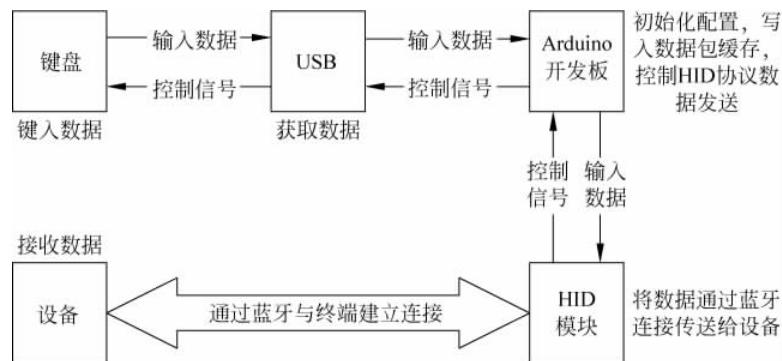
1. 工作原理及功能介绍

1) 工作原理

如图 3-9 所示,通过底层的硬件采集,将键盘的扫描码进行预处理打包,以三个字节封装一个键盘扫描码。由于扫描码分为接通扫描码和断开扫描码,因此,在对键盘数据包预处理的过程中,需要给它们定义不同的识别符,以便在 PC 中分离数据并进行处理。存放扫描码的缓冲区定义为 keybuffer,对于每个 keybuffer 由三字节构成。若 $\text{keybuffer}[0]=0$,表示采集的是断开扫描码;若 $\text{keybuffer}[0]=0\text{xff}$,表示采集的是接通扫描码。

keybuffer[1]定义为键盘扫描码。keybuffer[2]定义为0,作为扫描码数据包结束的标志在对扫描码转换为系统扫描码后,按照HID规范进行数据封装,软件的具体实现通过调用l2cap.c、sdp.c来进行。如上所述,对于键盘还有字节1、3、4、6被设置为对应的特殊标识值,字节5对应到当前的修饰键(Shift\Ctrl等),字节7~12是实际的数据传输位,可以处理

6个按键同时按下而不产生冲突。这样就形成标准 HID 格式的数据包。数据包通过提供的 API 函数进行无线发送，在 PC 接收端进行相同的处理过程，调用相应的协议封装函数进行数据解析，还原系统扫描码。



HID 蓝牙键盘模块需要使用 AT 命令进行参数设置方可使用。模块支持 HID 协议与具有蓝牙功能的 HOST 主机(包括 Android 和 iOS 设备)通信,或使用 SPP 透传方式进行原始的数据或命令传输。

引脚连接如图 3-10 所示,UART_TXD 以及 RXD 负责串口通信、VCC 以及 GND 供电。PIO 0~11 是可编程输入/输出口,通过设置引脚电平实现清除记忆、改变工作状态、进入 AT 工作模式、休眠控制等高级功能。一般初始化配置需要进入模块的 AT 模式,AT+RESET 复位模块,AT+BAUD 设置波特率,AT+DEVTYPEN 设置服务类型,AT+PSWD 设置配对码(默认 0000),最后连接单片机通电使用。

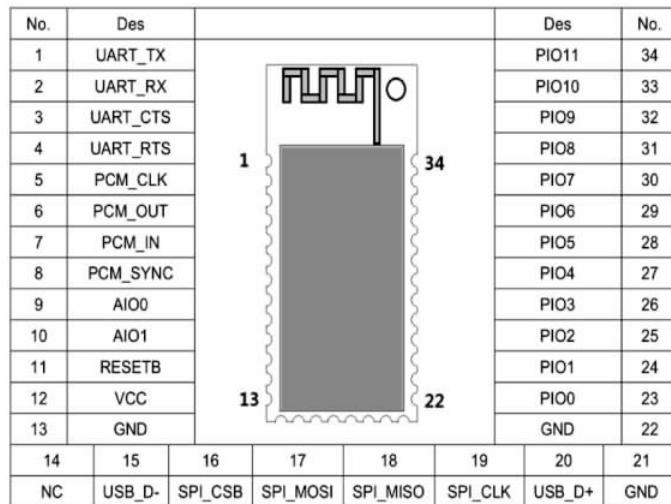


图 3-10 蓝牙 HID 模块引脚

2) 功能介绍

除了基本的字母、数字输入之外,还对键盘的修饰键、功能键做了处理,使输入的组合键实现用户期望的功能和效果。

2. 相关代码

此代码是用 Arduino USB Shield 获取按键和处理输出。包括键盘初始化配置与获取键盘按键之后,根据大小写切换状态以及修饰键状态向 Arduino 开发板及蓝牙输出数据。

```
# include <Spi.h>
# include "Max3421e.h"
# include "Usb.h"

//从键盘配置描述符中获取键盘数据的预定义
#define KBD_ADDR 1
#define KBD_EP 1
#define KBD_IF 0
#define EP_MAXPKTSIZE 8
#define EP_POLL 0x0a

//对键盘修饰键数据的定义(除了数字和字母)
#define BANG (0x1E)
#define AT (0x1F)
#define POUND (0x20)
#define DOLLAR (0x21)
#define PERCENT (0x22)
#define CAP (0x23)
#define AND (0x24)
#define STAR (0x25)
#define OPENBKT (0x26)
#define CLOSEBKT (0x27)
#define RETURN (0x28)
#define ESCAPE (0x29)
#define BACKSPACE (0x2A)
#define TAB (0x2B)
#define SPACE (0x2C)
#define HYPHEN (0x2D)
#define EQUAL (0x2E)
#define SQBKTOPEN (0x2F)
#define SQBKTCLOSE (0x30)
#define BACKSLASH (0x31)
#define SEMICOLON (0x33)
#define INVCOMMA (0x34)
#define TILDE (0x35)
#define COMMA (0x36)
#define PERIOD (0x37)
#define FRONTSLASH (0x38)
#define DELETE (0x4c)
#define SHIFT 0x22
```

```

#define CTRL 0x11
#define ALT 0x44
#define GUI 0x88
#define CAPSLOCK (0x39)
#define NUMLOCK (0x53)
#define SCROLLLOCK (0x47)
#define bmNUMLOCK 0x01
#define bmCAPSLOCK 0x02
#define bmSCROLLLOCK 0x04
EP_RECORD ep_record[2];
char buf[8] = { 0 };
char old_buf[8] = { 0 };
//某些功能键状态信息的定义与初始赋值
bool numLock = false;
bool capsLock = false;
bool scrollLock = false;
bool line = false;
//对设备的定义
MAX3421E Max;
USB Usb;
void setup() { //前期准备工作
    Serial.begin( 9600 ); //设置串口波特率
    Serial.println("Start"); //显示启动
    Max.powerOn(); //开始供电
    delay( 200 );
}
void loop() {
    Max.Task(); //芯片开始工作
    Usb.Task(); //USB 开始工作
    if( Usb.getUsbTaskState() == USB_STATE_CONFIGURING ) { //获取键盘工作状态
        kbd_init(); //键盘的初始化配置
        Usb.setUsbTaskState( USB_STATE_RUNNING ); //设置 USB 工作状态
    }
    if( Usb.getUsbTaskState() == USB_STATE_RUNNING ) {
        kbd_poll(); //获取键盘数据
    }
}
//键盘初始化配置函数
void kbd_init( void )
{
    byte rcode = 0; //定义返回码 return code(作为多个函数的返回值)
//初始化设置端点记录结构的数据
    ep_record[0] = *( Usb.getDevTableEntry( 0,0 ) );
    ep_record[1].MaxPktSize = EP_MAXPKTSIZE;
    ep_record[1].Interval = EP_POLL;
}

```

```

ep_record[1].sndToggle = bmSNDEGO;
ep_record[1].rcvToggle = bmRCVTOGO;
Usb.setDevTableEntry( 1, ep_record );           //将键盘端点数据写入设备表
//设置配置参数
rcode = Usb.setConf( KBD_ADDR, 0, 1 );
if( rcode ) {                                //异常检测与处理,当 rcode 为 0 时表示设置成功
Serial.print("Error attempting to configure keyboard. Return code :");
//显示异常
Serial.println( rcode, HEX );
while(1);
}
rcode = Usb.setProto( KBD_ADDR, 0, 0, 0 );      //设置启动协议
if( rcode ) {                                //异常检测与处理,当 rcode 为 0 时表示设置成功
Serial.print("Error attempting to configure boot protocol. Return code :");
//显示异常
Serial.println( rcode, HEX );                  //以十六进制换行输出返回码
while( 1 );
}
delay(2000);
Serial.println("Keyboard initialized");        //显示键盘已成功初始化
}
//获取键盘数据以及输出结果的函数
void kbd_poll( void )
{
char i;
boolean samemark = true;
static char leds = 0;
byte rcode = 0;                                //重新设置返回码为 0
//将键盘的扫描数据转移存入缓存区
rcode = Usb.inTransfer( KBD_ADDR, KBD_EP, 8, buf );
if( rcode!= 0 ) {                            //若返回码为 1,表示转存数据成功
{
return;
}
//若数据中有某些功能键,则进行以下操作
for( i = 2; i < 8; i++ )
{
if( buf[ i ] == 0 ) {
break;
}
if( buf_compare( buf[ i ] ) == false ) {        //如果新数据与旧数据不同
switch( buf[ i ] ) {                         //若缓存数据为以下几项中的某一种
case CAPSLOCK:                           //若缓存或者键盘输入的数据为 CAPSLOCK
capsLock = !capsLock;                    //使 capsLock 的值翻转
leds = ( capsLock )?leds| bmCAPSLOCK : leds &= ~bmCAPSLOCK;
//将 LED 的倒数第一个数值置为 1(开启)或 0(关闭)
break;
}
}
}
}

```

```

case NUMLOCK:                                //若缓存或者键盘输入的数据为 NUMLOCK
numLock = !numLock;                          //使 numLock 的值翻转
leds = ( numLock )?leds| = bmNUMLOCK : leds &= ~bmNUMLOCK;
break;
case SCROLLLOCK:                            //若缓存或者键盘输入的数据为 SCROLLLOCK
scrollLock = !scrollLock;                  //使 scrollLock 的值翻转
leds = ( scrollLock )?leds| = bmSCROLLLOCK : leds &= ~bmSCROLLLOCK;
//在键盘扫描码的数据包中写入数据
Serial.write(0x0c);                         //字节 1
Serial.write(0x00);                         //字节 2
Serial.write(0xA1);                         //字节 3
Serial.write(0x01);                         //字节 4
Serial.write(00);                           //字节 5
Serial.write(0x00);                         //字节 6
Serial.write(0x1e);                         //字节 7
Serial.write(0);                            //字节 8
Serial.write(0);                            //字节 9
Serial.write(0);                            //字节 10
Serial.write(0);                           //字节 11
Serial.write(0);                           //字节 12
delay(500);
Serial.write(0x0c);                         //字节 1
Serial.write(0x00);                         //字节 2
Serial.write(0xA1);                         //字节 3
Serial.write(0x00);                         //字节 4
Serial.write(0);                           //字节 5
Serial.write(0x00);                         //字节 6
Serial.write(0);                           //字节 7
Serial.write(0);                           //字节 8
Serial.write(0);                           //字节 9
Serial.write(0);                           //字节 10
Serial.write(0);                           //字节 11
Serial.write(0);                           //字节 12
break;
case DELETE:                                 //为 0
line = false;
break;
case RETURN:                                //值翻转
line = !line;
break;
} //USB 通过 getReport() 和 setReport() 发送数据
rcode = Usb.setReport( KBD_ADDR, 0, 1, KBD_IF, 0x02, 0, &leds );
if( rcode ) {
Serial.print("Set report error: ");          //异常提示
Serial.println( rcode, HEX );                //以十六进制换行输出返回码
//if( rcode ...
//if( buf_compare( buf[ i ] ) == false ...

```

```

} //for( i = 2...
i = 0;
while (i < 8)
{
if (old_buf[i] != buf[i]) { i = 12; }
i++;
}
if (i == 13) {
for (i = 0; i < 8; i++) {
Serial.print(buf[i],HEX);
Serial.print(']');
}
Serial.println('');
//将此次键盘输入的数据包传输出去
Serial.write(0x0c); //字节 1
Serial.write(0x00); //字节 2
Serial.write(0xA1); //字节 3
Serial.write(0x01); //字节 4
Serial.write(buf[1]); //字节 5
Serial.write(0x00); //字节 6
Serial.write(buf[2]); //字节 7
Serial.write(buf[3]); //字节 8
Serial.write(buf[4]); //字节 9
Serial.write(buf[5]); //字节 10
Serial.write(buf[6]); //字节 11
Serial.write(buf[7]); //字节 12
}
for( i = 2; i < 8; i++)
{
old_buf[i] = buf[i];
}
}

//将新数据与原数据进行比较的函数
bool buf_compare( byte data )
{
char i;
for( i = 2; i < 8; i++) {
if( old_buf[i] == data ) { //将缓存数据逐个比较
return( true ); //若二者完全相同则返回 true
}
}
return( false ); //若二者有任何的不同则返回 false
}

```

3.2.2 有线音箱无线化

本节包括有线音箱无线化的功能介绍及相关代码。

1. 功能介绍

本部分使用 Arduino 开发板、蓝牙立体音效模块和红外遥控模块,将原本与手机或者计算机必须有线连接的音箱以及扬声器,实现与上位机的无线连接,变成蓝牙音箱。上位机中的音频数据与控制信号通过蓝牙连接传送到立体音效模块,再通过线路传送到音箱或者扬声器播放。

除了音频数据的无线传输,还对控制信号的解析实现了音频播放的控制功能,并且有两种无线连接控制方式:一种是通过手机或者计算机直接控制;另一种是通过红外遥控器控制。对音频播放的控制包括开始播放、暂停播放、停止播放、下一首、上一首、增大音量、降低音量等操作。

在音频解码与处理方面,采用了 XS3868 音频模块,它有比较好的音频数据处理功能,可以保证音乐播放时良好的音质与用户体验。而且,对于各式各样的音箱或者耳机,产品也都具有广泛的适用性。

2. 相关代码

```
# include
int RECV_PIN = 14; //AO
IRrecv irrecv(RECV_PIN);
decode_results results;
unsigned long BlinkTime;
void setup(){
Serial.begin(115200);
irrecv.enableIRIn(); //启动音箱
}
void loop() {
if (Serial.available() > 0)Serial.write(Serial.read());
if (BlinkTime == 0)BlinkTime = millis();
if(millis() - BlinkTime > 10000){
Serial.println("AT#CB"); //取消配对模式
BlinkTime = 0;
}
if (irrecv.decode(&results))
{
irrecv.resume();
//接收下一个命令
if (results.value!= REPEAT){
Serial.println(results.value);
if(results.value == 0xF129)Serial.println("AT#VU"); //增加音量
else if(results.value == 0xF1A9)Serial.println("AT#VD"); //降低音量
else if(results.value == 0xF171)Serial.println("AT#MA"); //播放或者暂停
else if(results.value == 0xF1B1)Serial.println("AT#MC"); //停止播放音乐
else if(results.value == 0xF149)Serial.println("AT#MD"); //播放下一首音乐
else if(results.value == 0xF1C9)Serial.println("AT#ME"); //播放上一首音乐
}
}
}
```

```

//else if(results.value == 3993014240)Serial.println("AT# CB"); //取消配对状态
//else if(results.value == 3993014224)Serial.println("AT# CA"); //进入配对状态
//else if(results.value == 3993014256)Serial.println("AT# MI"); //连接声源
//else if(results.value == 3993014216)Serial.println("AT# CC"); //连接到手机
delay(300); }
}
}

```

3.2.3 人脸跟踪无线化摄像头

本节包括有线摄像头转人脸跟踪无线化摄像头的功能介绍及相关代码。

1. 功能介绍

1) 有线音箱无线化模块

有线音箱无线化模块主要包括 Arduino 开发板、WR703N 路由器、HC-05 蓝牙 SPP 模块、舵机和摄像头。摄像头拍摄视频，通过 WiFi 将视频流传到上位机，并在上位机的屏幕中显示。上位机对传入的视频流数据进行处理和识别（此过程利用了 Adaboost 分类器、OpenCV 框架和机器学习），检测出视频中的人脸并进行标识，以使用户看到识别的效果。

在实现人脸检测方面，使用的是 Adaboost 级联分类器，其中正脸识别使用的是 Haar 特征的级联分类器，侧脸识别使用的是采取 LBP 特征的级联分类器。我们的人脸识别级联分类器基于 OpenCV 的框架，经过了大量数据集的训练，对无遮挡人脸检测的稳定性较高。而且，正脸识别与侧脸识别两种方式并行，使得视频中的人脸进行平移或者头部进行较大角度的偏转时仍然可以被检测到。

当视频中的人脸移动时，偏离于窗体的中心人脸，人脸新状态将被实时检测。此时，计算机处理后会识别到人脸的新状态，并得出如何调整摄像头角度和方向，从而使摄像头对准目标人脸，下一个状态的视频中，目标人脸将处于视频窗体的中心位置。这个过程的周期是极其短暂的，其延迟时间将忽略不计，因此，摄像头可以完全实时地自动跟踪目标人脸。另外，我们后期又对舵机控制进行了改进，增强了摄像头转向时的连续性，保证了视频的流畅度与稳定性。

在这个模块中，除了视频流传输到上位机的无线化之外，上位机对舵机控制信号的传输也是无线化的，这个环节采用了相对快速低能耗的蓝牙技术。

2) Android 手机 APP

经过计算机处理与识别视频流数据后，上位机可以产生相应的控制信号传送给舵机，达到自动控制的效果。除了自动控制舵机的方式外，还可以在手机端 APP 控制舵机的转动。用户在手机端 APP 按键，将用户的指令转换成控制信号，并通过蓝牙传送给舵机，这样摄像头在舵机的物理支撑下就可以实现根据用户的意愿转动角度的效果。

APP 端除了按键控制之外，还可以接收摄像头通过 WiFi 传入的视频流数据，并展示在手机屏幕上，用户既可以通过 PC，也可以在手机 APP 端观看视频，同时对摄像头角度进行

控制。Android 手机端 APP 界面如图 3-11 所示。



图 3-11 Android 手机端 APP 界面

2. 相关代码

1) 图像处理与人脸识别代码

```
import cv2
import numpy as np
import serial
//设置串口波特率
ser = serial.Serial("/dev/cu.HC-05-SPPDev", 9600, timeout = 0.5)
cv2.namedWindow("CV")
//从路由器视频流端口获取视频数据
cap = cv2.VideoCapture("http://192.168.8.1:8083/?action=stream")
//读取视频数据
success, frame = cap.read()
# width = int(cv2.cvGetCaptureProperty(cap, cv2.CV_CAP_PROP_FRAME_WIDTH))
# height = int(cv2.cvGetCaptureProperty(cap, cv2.CV_CAP_PROP_FRAME_HEIGHT))
//引入 Haar 与 LBP 两种级联分类器
classifier = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")
classifier1 = cv2.CascadeClassifier("lbpcascade_profileface.xml")
while success:
//转存视频数据到 frame 中
```

```

success, frame = cap.read()
# cv2.flip(frame, frame, -1)
size = frame.shape[ :2]
image = np.zeros(size, dtype = np.float16)
//视频帧灰度转化
image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
//视频帧直方均衡化
cv2.equalizeHist(image, image)
//设置步长
divisor = 8
hi, wi = size
//设置扫描框的大小
minSize = (wi//divisor, hi//divisor)
//检测正面人脸
faceRects = classifier.detectMultiScale(image, 1.2, 2, cv2.CASCADE_SCALE_IMAGE, minSize)
//检测侧面人脸
faceRects1 = classifier1.detectMultiScale(image, 1.2, 2, cv2.CASCADE_SCALE_IMAGE, minSize)
//判断标识正面人脸的矩形框中心是否在视频窗体中心
//若不在则对舵机发送相应的控制信号
if len(faceRects)> 0:
    for faceRect in faceRects:
        x, y, w, h = faceRect
        cv2.circle(frame, (x + w//2, y + h//2), min(w//2, h//2), (255, 0, 0))
        if((y + h//2)> hi * 2/3):
            ser.write("DJ_Xia\r\n")
        if((y + h//2)< hi * 1/3):
            ser.write("DJ_Shang\r\n")
        if((x + w//2)< wi * 1/3):
            ser.write("DJ_Zuo\r\n")
        if((x + w//2)> wi * 2/3):
            ser.write("DJ_You\r\n")
//在视频中用圆形标识出人脸
# cv2.circle(frame, int((x + w/4, y + h/4)), min(w//8, h//8), (255, 0, 0))
# cv2.circle(frame, int((x + 3 * w/4, y + h/4)), int(min(w/8, h/8)), (255, 0, 0))
# cv2.rectangle(frame, (x + 3 * w/4, y + 3 * h/4), (x + 5 * w/8, y + 7 * h/8), (255, 0, 0))
//判断标识侧面人脸的矩形框中心是否在视频窗体中心
//若不在则对舵机发送相应的控制信号
if len(faceRects1)> 0:
    for faceRect1 in faceRects1:
        x, y, w, h = faceRect1
        cv2.circle(frame, (x + w//2, y + h//2), min(w//2, h//2), (255, 0, 0))
        if((y + h//2)> hi * 0.6):
            ser.write("DJ_Xia\r\n")
        if((y + h//2)< hi * 0.4):
            ser.write("DJ_Shang\r\n")
        if((x + w//2)< wi * 0.4):
            ser.write("DJ_Zuo\r\n")

```

```

        if((x+w//2)>wi*0.6):
            ser.write("DJ_You\r\n")
//在上位机中显示出视频与人脸检测结果
cv2.imshow("CV",frame)
key = cv2.waitKey(10)
c = chr(key&255)
if c in ['q','Q',chr(27)]:
    break
//释放视频窗体
cv2.destroyAllWindows()
ser.close()

```

2) 舵机控制摄像头转动代码

```

#include <Servo.h>
#include <Wire.h>
Servo servoX;
Servo servoY;
byte serialIn = 0;
byte commandAvailable = false;
String strReceived = "";
//两个舵机的居中角度
byte servoXCenterPoint = 94;
byte servoYCenterPoint = 88 ;
//两个舵机的最大角度
byte servoXmax = 170;
byte servoYmax = 130;
//两个舵机的最小角度
byte servoXmini = 10;
byte servoYmini = 10;
//两个舵机的当前角度,用于回传,不用再读取计算,节约资源
byte servoXPoint = 0;
byte servoYPoint = 0;
//舵机每转动一次的角度递增值
byte servoStep = 4;
//声明部分结束,注意,很多用 byte 类型,如果大于 255 用 int 类型
void setup()
{
    servoX.attach(10);                                //1号舵机信号引脚
    servoY.attach(11);                                //2号舵机信号引脚
    servo_test();                                     //舵机测试
    Serial.begin(9600);
}
void loop()
{
    getSerialLine();
    if (commandAvailable) {

```

```
processCommand(strReceived);
strReceived = "";
commandAvailable = false;
}
}
void getSerialLine()
{
    //使用\r字符作为两条命令间隔符,拼接收到的字符
    while (serialIn != '\r')
    {
        if (!(Serial.available() > 0))
        {
            return;
        }
        serialIn = Serial.read();
        if (serialIn != '\r') {
            //对于某些语言可能无法单独输出\r忽略后续的\n字符
            if (serialIn != '\n'){
                char a = char(serialIn);
                strReceived += a;
            }
        }
    }
    //读取到分隔符,重新启动拼接
    if (serialIn == '\r') {
        commandAvailable = true;
        serialIn = 0;
    }
}
//命令处理过程,接收并处理命令,使舵机做出相应动作
//所有的命令识别都在这里完成
void processCommand(String input)
{
    String command = getValue(input, ' ', 0);
    byte iscommand = true;
    int val;
    if (command == "DJ_CS")
    {
        servo_test();
        //Serial.println("ok");
    }
    else if (command == "DJ_Shang")
    {
        servo_up();
        //Serial.println("ok");
    }
    else if (command == "DJ_Xia")
```

```

{
    servo_down();
    //Serial.println("ok");
}
else if (command == "DJ_Zuo")
{
    servo_left();
}
else if (command == "DJ_You")
{
    servo_right();
}
else if (command == "DJ_Zhong")
{
    servo_center();
}
else if (command == "DJ_CZ_JD")           //垂直旋转
{
    val = getValue(input, ' ', 1).toInt();
}
else if (command == "DJ_SP_JD")           //水平旋转
{
    val = getValue(input, ' ', 1).toInt();
}
else
{
    iscommand = false;
}
//收到的是不是已经定义的命令,如果不是则不回送状态,以免浪费带宽
if (iscommand){
    SendMessage("cmd:" + input);
    SendStatus();
}
}

//命令参数获取方法,支持一个命令多个参数,采用空格符作为分隔符号,注意是半角空格符,不是制表符也不是全角空格
String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[ ] = {0, -1 };
    int maxIndex = data.length() - 1;
    for (int i = 0; i <= maxIndex && found <= index; i++){
        if (data.charAt(i) == separator|| i == maxIndex){
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex)?i + 1:i;
        }
    }
}

```

```
        }
        return found> index?data.substring(strIndex[0], strIndex[1]) : "";
    }
//舵机动作部分
//舵机测试函数
void servo_test(void) {
    int nowcornerY = servoY.read();
    int nowcornerX = servoX.read();
    servo_Vertical(servoymini);
    delay(500);
    servo_Vertical(servoymax);
    delay(500);
    servo_Vertical(servoyCenterPoint);
    delay(500);
    servo_Horizontal(servoxmini);
    delay(500);
    servo_Horizontal(servoxmax);
    delay(500);
    servo_Horizontal(servoxCenterPoint);
    delay(500);
    servo_center();
}
//舵机向右转动
void servo_right(void)
{
    int servotemp = servoX.read();
    servotemp -= servoStep;
    servo_Horizontal(servotemp);
}
//舵机向左转动
void servo_left(void)
{
    int servotemp = servoX.read();
    servotemp += servoStep;
    servo_Horizontal(servotemp);
}
//舵机向下转动
void servo_down(void)
{
    int servotemp = servoY.read();
    servotemp += servoStep;
    servo_Vertical(servotemp);
}
//舵机向上转动
void servo_up(void)
{
    int servotemp = servoY.read();
```

```
servotemp -= servoStep;
servo_Veritical(servotemp);
}

//舵机回到正中央
void servo_center(void)
{
    servo_Veritical(servoyCenterPoint);
    servo_Horizontal(servoxCenterPoint);
}

//舵机在垂直方向转动
void servo_Veritical(int corner)
{
    int cornerY = servoy.read();
    if (cornerY > corner) {
        for (int i = cornerY; i > corner; i = i - servoStep) {
            servoy.write(i);
            servoyPoint = i;
            delay(50);
        }
    }
    else {
        for (int i = cornerY; i < corner; i = i + servoStep) {
            servoy.write(i);
            servoyPoint = i;
            delay(50);
        }
    }
    servoy.write(corner);
    servoyPoint = corner;
}

//舵机在水平方向转动
void servo_Horizontal(int corner)
{
    int i = 0;
    byte cornerX = servox.read();
    if (cornerX > corner) {
        for (i = cornerX; i > corner; i = i - servoStep) {
            servox.write(i);
            servoXPoint = i;
            delay(50);
        }
    }
    else {
        for (i = cornerX; i < corner; i = i + servoStep) {
            servox.write(i);
            servoXPoint = i;
            delay(50);
        }
    }
}
```

```

        }
    }
    servoX.write(corner);
    servoXPoint = corner;
}
//拼接字符串并向串口发送当前舵机运动状态
void SendStatus(){
    String out = "";
    out += servoXPoint;
    out += ",";
    out += servoYPoint;
    out += ",";
    SendMessage(out);
}
//串口消息发送
void SendMessage(String data){
    Serial.println(data);
}

```

3) Android APP 代码

```

package com.bluetoothtest;
import java.io.BufferedReader;
import java.io.BufferedInputStream;
import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.UUID;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.os.Vibrator;
import android.app.Activity;
import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.view.Menu;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.TextView;

```

```

import android.widget.Toast;
//蓝牙接收线程类
class bluetoothMsgThread extends Thread {
    private DataInputStream mmInStream; //in 数据流
    private Handler msgHandler; //Handler
    public bluetoothMsgThread(DataInputStream mmInStream, Handler msgHandler) {
        //构造函数,获得 mmInStream 和 msgHandler 对象
        this.mmInStream = mmInStream;
        this.msgHandler = msgHandler;
    }
    public void run() {
        byte[ ] InBuffer = new byte[15]; //创建缓冲区
        while (!Thread.interrupted()) {
            try {
                mmInStream.readFully(InBuffer, 0, 15); //读取蓝牙数据流
                Message msg = new Message(); //定义一个消息,并填充数据
                msg.what = 0x1234;
                msg.obj = InBuffer;
                msg.arg1 = InBuffer.length;
                msgHandler.sendMessage(msg); //通过 Handler 发送消息
            }catch(IOException e) {
                e.printStackTrace();
            }
        }
    }
}
public class MainActivity extends Activity {
    BluetoothAdapter mBluetoothAdapter; //蓝牙适配器
    BluetoothDevice device; //蓝牙设备
    BluetoothSocket clientSocket; //Socket 通信
    BluetoothServerSocket btserver; //蓝牙设备地址
    String address; //out 数据流
    OutputStream mmOutStream; //in 数据流
    DataInputStream mmInStream;
    bluetoothMsgThread blue_tooth_msg_thread;
    UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B37FB");
    //蓝牙连接用 UUID 标识
    Vibrator vib; //手机系统振动对象
    //显示从蓝牙设备接收到的数据
    public void show_result(byte[ ] buffer, int count)
    {
        StringBuffer msg = new StringBuffer(); //创建缓冲区
        StringBuffer msg2 = new StringBuffer();
        TextView tvInfo = (TextView)findViewById(R.id.textViewReceiveInfo);
        //创建文本显示对象
        tvInfo.setText("");
        //清空对象内容
    }
}

```

```

        for (int i = 0; i < count; i++) {
            //循环加入数据,十六进制格式
            msg.append(String.format("0x % x ", buffer[i]));
            msg2.append(String.format(" % c", buffer[i]));
        }
        msg.append("\r\n");
        msg.append(msg2);
        tvInfo.setText(msg); //显示到界面上
    }

    //设置按钮的状态,根据参数设置一批按钮的状态
    public void set_btn_status(boolean status)
    {
        Button ledonBtn = (Button)findViewById(R.id.ledonBtn);
        ledonBtn.setEnabled(status);
        Button ledoffBtn = (Button)findViewById(R.id.ledoffBtn);
        ledoffBtn.setEnabled(status);
        Button jdqonBtn = (Button)findViewById(R.id.jdqonBtn);
        jdqonBtn.setEnabled(status);
        Button jdqoffBtn = (Button)findViewById(R.id.jdqoffBtn);
        jdqoffBtn.setEnabled(status);
    }

    protected void onDestroy() {
        super.onDestroy();
        try {
            if (mmOutStream!= null)
                mmOutStream.close(); //关闭 out 数据流
            if (mmInStream!= null)
                mmInStream.close(); //关闭 in 数据流
            if (clientSocket!= null)
                clientSocket.close(); //关闭 Socket
            blue_tooth_msg_thread.interrupt();
            Toast.makeText(getApplicationContext(), "蓝牙应用程序退出", Toast.LENGTH_LONG).show(); //提示信息
        }catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        set_btn_status(false);
        //蓝牙设备未连接,设置一些按钮不能操作
        vib = (Vibrator) getSystemService(Service.VIBRATOR_SERVICE);
        //获取手机振动对象
        Button searchDeviceBtn = (Button)findViewById(R.id.searchDeviceBtn);
        //创建搜索按键对象并监听 click 事件
    }
}

```

```
searchDeviceBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        //获取蓝牙适配器
        if (mBluetoothAdapter == null) {
            //手机无蓝牙功能,提示并退出
            Toast.makeText(getApplicationContext(), "bluetooth is no available",
Toast.LENGTH_LONG).show();
            finish();
            return;
        }
        mBluetoothAdapter.enable();
        //打开手机蓝牙功能
        if (!mBluetoothAdapter.isEnabled()) {
            //手机未打开蓝牙功能,提示并退出
            Toast.makeText(getApplicationContext(), " bluetooth function is no
available",Toast.LENGTH_LONG).show();
            finish();
            return;
        }
        Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
        //获取已经配对的蓝牙设备列表
        if (pairedDevices.size() < 1) {
            //无配对蓝牙设备,则退出
            Toast.makeText(getApplicationContext(), "没有找到已经配对的蓝牙设备,
请配对后再操作",Toast.LENGTH_LONG).show();
            finish();
            return;
        }
        Spinner spinner = (Spinner)findViewById(R.id.spinner1);
        //获取下拉框控件对象
        List<String> list = new ArrayList<String>();
        //创建列表,用于保存蓝牙设备地址
        for (BluetoothDevice device:pairedDevices) {
//myArrayAdapter.add(device.getName() + " " + device.getAddress());
//list.add(device.getName() + " " + device.getAddress());
            list.add(device.getAddress());
            //将蓝牙地址进入到列表
        }                                              //创建数组适配器
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(getApplicationContext(),
android.R.layout.simple_spinner_item,list);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        //设置下拉显示方式
        spinner.setAdapter(adapter);
        //将适配器中数据给下拉框对象
        Button connectBtn = (Button)findViewById(R.id.connectBtn);
```

```

        //创建连接按钮对象
        connectBtn.setEnabled(true);
        //允许连接对象按钮操作
    }
}
);
Button connectBtn = (Button) findViewById(R.id.connectBtn);
//创建连接按钮对象,设置监听器
connectBtn.setEnabled(false);
//不允许连接对象按钮操作
connectBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        Spinner spinner = (Spinner) findViewById(R.id.spinner1);
        //获取下拉框对象
        address = spinner.getSelectedItem().toString();
        //从下拉框中选择项目,并获得它的地址
        try {
            device = mBluetoothAdapter.getRemoteDevice(address);
            //根据蓝牙设备的地址连接单片机蓝牙设备
            clientSocket = device.createRfcommSocketToServiceRecord(uuid);
            //根据 uuid 创建 socket
            clientSocket.connect();
            //手机 socket 连接远端蓝牙设备
            mmOutputStream = clientSocket.getOutputStream();
            //从 socket 获得数据流对象,实现读写操作
            mmInputStream = new DataInputStream(new BufferedInputStream(clientSocket.getInputStream()));
            Toast.makeText(getApplicationContext(), "蓝牙设备连接成功", Toast.LENGTH_SHORT).show();
            vib.vibrate(100);
            //手机振动,时长 100ms
            set_btn_status(true);
            //允许按钮操作,定义多线程对象,并执行线程,用于接收蓝牙数据
            blue_tooth_msg_thread = new bluetoothMsgThread(mmInputStream,
            bluetoothMessageHandle);
            blue_tooth_msg_thread.start();
        } catch (Exception e) {
            set_btn_status(false); //不允许按钮操作
            Toast.makeText(getApplicationContext(), "蓝牙设备连接失败!", Toast.LENGTH_SHORT).show();
            e.printStackTrace();
        }
    }
});

```

```
Button ledonBtn = (Button) findViewById(R. id. ledonBtn);
//创建向上按钮对象,设置监听器
ledonBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        byte[ ] InBuffer = new byte[64];
        //输入缓存
        byte buffer[ ] = "DJ_Shang\r\n";
        //字符串缓冲区写入将要发送的命令,向上转
        try {
            mmOutStream.write(buffer);
        //数据流发送数组,发送给单片机蓝牙设备
        //mmInStream.readFully(InBuffer, 0, 8);
        //读取外部蓝牙设备发送回来的数据
        //show_result(InBuffer,8);
        //显示到界面上
        vib. vibrate(100);
        //手机振动,时长 100ms
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
});
Button ledoffBtn = (Button) findViewById(R. id. ledoffBtn);
//创建向下按钮对象,设置监听器
ledoffBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        byte[ ] InBuffer = new byte[64];
        //输入缓存
        byte buffer[ ] = "DJ_Xia\r\n";
        //字符串缓冲区写入将要发送的命令,向下转
        try {
            mmOutStream.write(buffer);
        //数据流发送数组,发送给单片机蓝牙设备
        //mmInStream.readFully(InBuffer, 0, 8);
        //读取外部蓝牙设备发送回来的数据
        //show_result(InBuffer,8);
        //显示到界面上
        vib. vibrate(100); //手机振动,时长 100ms
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
});
```

```

Button jdqonBtn = (Button) findViewById(R. id. jdqonBtn);
//创建左转按钮对象,设置监听器
jdqonBtn. setOnTouchListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        byte[ ] InBuffer = new byte[64];
        //输入缓存
        byte buffer[ ] = "DJ_Zuo\r\n";
        //字符串缓冲区写入将要发送的命令,向左转
        try {
            mmOutStream.write(buffer);
        //数据流发送数组,发送给单片机蓝牙设备
        //mmInStream. readFully(InBuffer, 0, 8);
        //读取外部蓝牙设备发送回来的数据
        //show_result(InBuffer,8);
        //显示到界面上
        vib. vibrate(100);
        //手机振动,时长 100ms
        }catch (Exception e) {
            e. printStackTrace();
        }
    }
));
Button jdqoff Btn = (Button) findViewById(R. id. jdqoffBtn);
//创建右转按钮对象,设置监听器
jdqoffBtn. setOnTouchListener(new View.OnClickListener() {
    @Override
    public void onClick(View arg0) {
        byte[ ] InBuffer = new byte[64];
        byte buffer[ ] = "DJ_You\r\n";
        //输入缓存
        //字符串缓冲区写入将要发
        //送的命令,向右转
        try {
            mmOutStream.write(buffer);
        //数据流发送数组,发送给单片机蓝牙设备
        //mmInStream. readFully(InBuffer, 0, 8);
        //读取外部蓝牙设备发送回来的数据
        //show_result(InBuffer,8);
        //显示到界面上
            vib. vibrate(100);
        //手机振动,时长 100ms
        }catch (Exception e) {
            e. printStackTrace();
        }
    }
});

```

```
});  
  
Button jdqoffBtn = (Button) findViewById(R.id.jdqoffBtn);  
//创建居中复位按钮对象,设置监听器  
jdqoffBtn.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View arg0) {  
        byte[] InBuffer = new byte[64];  
        byte buffer[] = "DJ_Zhong\r\n";  
        //字符串缓冲区写入将要发送的命令,复原到初始位置  
        try {  
            mmOutStream.write(buffer);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
});  
Handler bluetoothMessageHandle = new Handler() {  
    public void handleMessage(Message msg) {  
        if (msg.what == 0x1234) {  
            //如果消息是 0x1234,则是从线程中传输的数据  
            show_result((byte[])msg.obj, msg.arg1);  
            //将缓冲区的数据显示到 UI  
        }  
    }  
};  
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    getMenuInflater().inflate(R.menu.main, menu);  
    return true;  
}  
}
```

3.3 产品展示

有线键盘无线化模块如图 3-12 所示,有线音箱无线化模块如图 3-13 所示,人脸跟踪无线化摄像头模块如图 3-14 所示,产品内部如图 3-15 所示,产品外观如图 3-16 所示。



图 3-12 有线键盘无线化模块

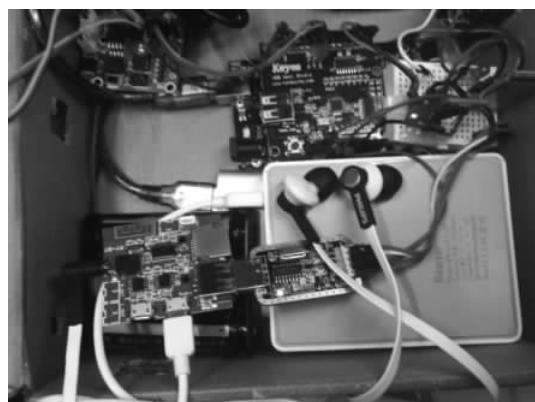


图 3-13 有线音箱无线化模块



图 3-14 人脸跟踪无线化摄像头模块



图 3-15 产品内部



图 3-16 产品外观

3.4 元件清单

完成本项目所用到的元件及数量如表 3-2 所示。

表 3-2 元件清单

元件/测试仪表	数 量	元件/测试仪表	数 量
蓝牙 HID 模块	1 个	开关	1 个
USB HOUST 转接板	1 个	云台底座	1 个
导线	若干	面包板	1 个
Arduino MEGA 2560 开发板	1 个	摄像头	1 个
XS3868 蓝牙立体声音频模块(主控 OVC3860)	1 个	LM2596S 电源稳压模块	1 个
3.5mm 音频母座接口	1 个	3 节串联 18650 电池盒	1 个
红外 IR Receiver	1 个	WR703N 路由器	1 个
红外遥控发射器	1 个	舵机	2 个
HC-05 蓝牙模块	1 个	3.7V 18650 锂电池	3 个