文章编号：2095-2163(2021)02-0195-07 中图分类号：TP399 文献标志码：B

# 基于 TFLite 实现个性化灯光控制系统

盛雪丰
（苏州信息职业技术学院，江苏 苏州 215200）

**摘 要：** 本文主要实现语音输入“开”、“关”信号来完成开发板上 LED 灯的开启和关闭，通过分别采集 100 次“开”、“关”、“环境噪音”等声音信号，并将采集的声音以 CSV 格式的数据集进行训练，再将训练得到的模型转换为 TFLite 格式，同时将该模型运行到开发板进行声音的推理，进而控制开发板上 LED 灯的开启和关闭操作。

**关键词：** 人工智能；语音识别；PCM 信号；快速傅里叶变换

## Realization of personalized lighting control system based on TFLite

SHENG Xuefeng
（Suzhou College of Information Technology, Suzhou Jiangsu 215200, China）

**【Abstract】** This paper mainly realizes the voice input "on" and "off" signals to complete the opening and closing of LED lights on the development board. Through the acquisition of 100 times of "on", "off", "environmental noise" and other sound signals, the collected sound is trained in the data set of CSV format, and the training model is transformed into TFLite format. Based on the aboved, the model is loaded to the development board for sound reasoning. It is demonstrated that the LED lamp on the development board can be turned on and off.

**【Key words】** artificial intelligence; speech recognition; PCM signal; fast Fourier transform

## 0 引 言

当今世界信息产业发展中物联网的崛起，无疑是继计算机、互联网后信息化时代的又一重大变革。基于物联网的各种创新应用将成为新一轮的创业热点话题，而这些创新领域中一个重要特点就是物联网与人工智能的深度融合。可以期待，这些融合必将广泛应用于智慧城市、工业物联网、智能家居、农业物联网和各种可穿戴设备等领域，有着巨大的发展潜力和可观的应用前景。

在人工智能走入人们生活场景前，物联网应用已被广泛运用于智能家居场景，比如智能照明，就是一种非常直观的物联网家居体验，通过手机应用控制灯光的开关，类似的应用还有家庭安防、空调温度调节等。直到 2017 年 7 月，阿里巴巴推出的智能音箱产品-天猫精灵，其中内置了阿里研发的语音助手 AliGenie，伴随着和用户的持续互动，让 AliGenie 这个语音大脑不断进化成长，从而实现更多技能，这也可以视作人工智能技术第一次真正意义上走进人们的生活场景。随着阿里智能以及一些第三方应用的加入，天猫精灵逐渐能控制多达数十种品类的智能家居产品，正式开启了物联网与人工智能结合的道路。随后的几年时间里，小米、百度、京东等知名厂商也纷纷推出了自己的智能音箱产品，但无论是哪家厂商推出的智能音箱产品，都只是识别了用户语音输入的内容，再传送到云端进行数据匹配，最终返回相应的控制指令来操控物联网设备，无法达到个性化的语音控制效果。

其实每个人发出的声音都各不相同，这是因为人发出的声音是由音色决定的。一个人的声音不会是单调的 100 Hz，可能是由 100 Hz（10 分贝）、200 Hz（20 分贝）、50 Hz（5 分贝）等频率的一种组合，而这种组合就被称为音色。本次研究旨在通过采集不同人的声音，分别进行模型训练，从而达到个性化的语音控制效果，并将这种应用和物联网进行结合，例如声控门锁、声控灯光等智能家居产品，在达到智能化的同时也更好地保护了用户的隐私安全。

为了实现个性化的语音控制，研究中先要采集家庭中各个成员的音频数据，然后对采集的音频数据进行预处理，接着采用人工智能框架搭建合适的模型进行训练，最终将模型搭载到相应的物联网设备进行推理。

**基金项目：** 2019 年江苏高校“青蓝工程”培养对象（苏教师〔2019〕3 号）；2020 年苏州高职高专院校第二批产学研合作示范基地项目“信息通信技术产学研合作示范基地”（2020-5）。

**作者简介：** 盛雪丰（1981-），男，副教授，主要研究方向：人工智能、移动应用开发、物联网。

**收稿日期：** 2020-11-18

## 1 音频数据的采集

声音是连续的声波信息,也可称为模拟音频信号。声音有 2 个重要指标,分别是:振幅(声音的强弱)、频率(音调的高低)。而计算机内部使用 0 或者 1 离散地表示数据,则被称为数字音频信号。纯粹的模拟信号无法完全不失真地用数据来描述,必需经过定制化处理,才能得到用于描述现象的理想数据。这些处理中至少需要用到采样和量化这两个过程,从而将外界的模拟信号转化为数字信号。综上可知,模拟信号转化为数字信号的设计流程则如图 1 所示。

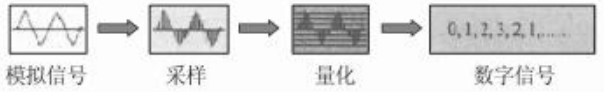


**图 1 模拟信号转化为数字信号**

**Fig. 1 Conversion of analog signal to digital signal**

在转换的过程中,需要考虑 3 个重要的技术指标,也就是:采样频率,量化位数,通道数。其中,采样频率是指每秒钟取得声音样本的次数。采样频率越高,声音的质量就越好,但同时所占用的资源也就越多。量化就是将采样样本幅度加以量化,也是用来衡量声音波动变化的一个参数。例如,一个 2 bit 量化的过程即如图 2 所示。2 bit 的模数转换采样意味着只能取值 4 次,通过 2 bit 去量化世界上所有的声音,那么最终只能得到 4 种声音,与实际的模拟量声音效果有比较大的出入,会带来很明显的噪声,也可将其称为量化噪声。

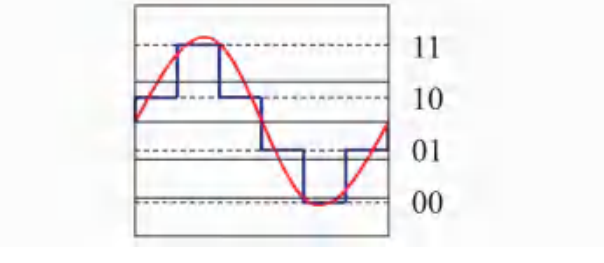


**图 2 采用 2 bit 量化图**

**Fig. 2 2 bit quantization chart was used**

通道数,即采集声音的通道数目。常有单声道和立体声之分,单声道的声音只能使用一个喇叭发声(有的也处理成 2 个喇叭输出同一个声道的声音),立体声可以使 2 个喇叭都发声(一般左右声道有分工),对空间效果的感受也将更为丰富,当然还有更多的通道数。

本次研究使用的是 TinyML 开发板,主要考虑到该开发板上自带一个全向麦克风设备,当然也可以选择其他类型的开发板,然后外接音频采样设备。同时在初始化方法中完成采样频率、量化位数以及采样通道等参数的设置。设置代码具体如下。

```
void i2s_init(){
const i2s_config_t i2s_config = {
. sample_rate = 8 000,
// 设置采样频率为 8 000 Hz
. bits_per_sample = i2s_bits_per_sample_t(16),
// 设置量化位数为 16 bit
. channel_format = I2S_CHANNEL_FMT_
ONLY_LEFT,
//设置采样通道数为单通道
}
}
```

在本次实验中,采样频率设置为 8 000,主要是考虑到开发板的存储空间有限,并且 8 000 Hz 常用于电话音频的采样,已经能够提供清晰的音频信息。量化位数设置为 16 bit,可以将振幅划分为 65 536 个等级,这已经是 CD 的标准。采样通道,这里设置为单通道。

当成功设置了声音采集参数后,在 *loop* 函数中使用 *i2s_pop_sample* 函数对声音进行持续的采样。音频数据的串口输出如图 3 所示,此时研究中编写的代码见如下。

```
void loop() {
int16_t sample = 0;
// 保存当前采样点的具体值
int bytes = i2s_pop_sample( I2S_PORT , (void
*)&sample , portMAX_DELAY );
if(bytes > 0){
Serial. println(sample);
}
}
```

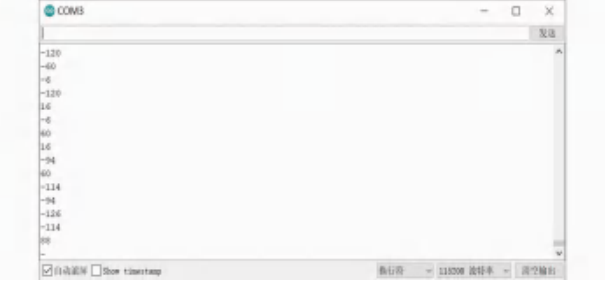


**图 3 音频数据的串口输出**

**Fig. 3 Serial output of audio data**

通过串口输出可以发现,模拟信号经过采样、量化、编码转换成了标准数字音频数据,并被称为

PCM 音频数据。通过 Arduino 工具菜单中自带的串口绘图器，来查看用户采集到的 PCM 数据，以图形化的方式进行展示。PCM 音频数据的图形化输出结如图 4 所示。

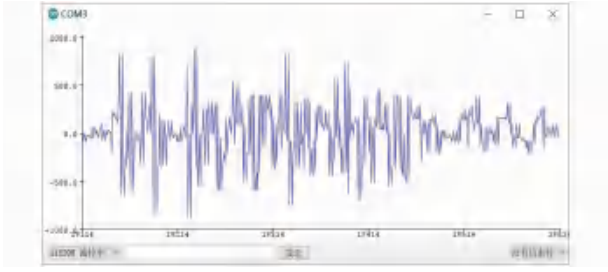

图 4　PCM 音频数据的图形化输出

Fig. 4　Graphic output of PCM audio data

由于开发板的内存空间有限，无法预留很大的缓冲区去暂存采集的 PCM 数据。实践也发现，串口输出释放内存的速度远不如采样消耗内存来得快，于是本次研究将不再采用串口输出的模式，而是改用 websocket 的方式将数据传到浏览器端进行 PCM 信号的收集，以确保缓冲区内数据不会溢出。

因此需要将开发板连接路由器，使之和 PC 端的浏览器处于同一个局域网内，可以在开发板上嵌入一个 WiFi 模块，同时配置路由器采用 DHCP 动态路由的方式，稍后在程序代码中配置相应的 ssid 和 password，让开发板连接路由器。

当开发板与路由器连接成功后，会在串口监视器中返回一个 IP 地址，该 IP 地址代表路由器分配给开发板的动态 IP 地址，得到的开发板的 IP 地址界面如图 5 所示。此时，开发板和 PC 就处于同一个局域网了。

图 5　获取开发板的 IP 地址

Fig. 5　Get the IP address of the development board

至此，将通过浏览器访问这个 IP 地址，就可以实时输出开发板中采集的 PCM 音频数据。但是实践发现，随着用户音频数据的采集，浏览器会出现卡死的现象，为此将不会直接在浏览器页面输出 PCM 数据，而是编写 python 程序将采集到的 PCM 音频数据存入文本文件，具体如图 6 所示。

图 6　采集的 PCM 数据存入文本文件

Fig. 6　The collected PCM data is stored in a text file

## 2　音频数据的处理

对于前述采集的 PCM 音频数据，如果用一幅图像来表示，可以观察到这样一幅图像，如图 7 所示。其中，横坐标表示时间，纵坐标表示此刻的声波能量大小。

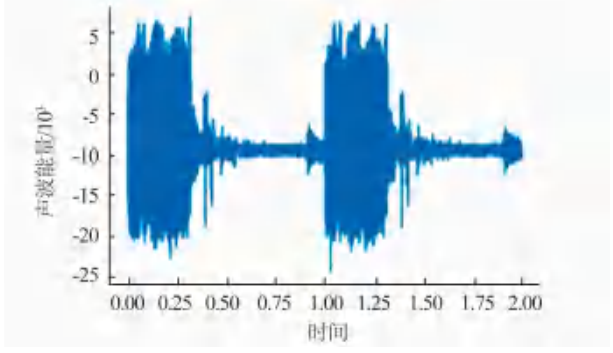


图 7　PCM 音频数据的图像表示

Fig. 7　Image representation of PCM audio data

假设在图 7 中图像的任意一个时刻，用刀进行切割，会发现此刻的声音其实是由很多不同频率的声音构成的，而每个频率上响度值也各不相同，人的耳朵所听到的其实是该时刻所有声音频率和响度的总合。本次研究期望实现的个性化语音控制，其实就是利用每个人的声音在频率和响度上都是不一样的原理。

PCM 音频数据的时域和频域表示如图 8 所示。一个 PCM 采样点的数字是无法表示当前时刻声音的频率和响度的，也很难直接体现出其特征，但是如果变换到频域之后，就很容易得出其特征了。那么如何将 PCM 图转化为频谱图？这里可以使用离散傅里叶变换的快速算法（FFT）。假设采样频率为 $Fs$，信号频率 $F$，采样点数为 $N$。在 FFT 后的结果就是 $N$ 个点的复数，每一个点对应一个频率，其模值就是该频率下的幅度特性。第一个点表示直流分量（即 0 Hz），而最后一个点 $N$ 的再下一个点（实际上

这个点是不存在的,这里是假设第 $N+1$ 个点)表示采样频率 $Fs$,采样频率被平均分成 $N$ 等份,每个点的频率依次增加,例如某点 $n$ 所表示的频率值为:$Fn=(n-1)*Fs/N$。采样频率越高,可以测量的频谱范围越大,因此这里将采样频率从 8 000 Hz 提高到 44 100 Hz。

同时,考虑到 FFT 结果的对称性,通常只使用前半部分的结果,即小于采样频率一半的结果,最多只能测到 22 050 Hz。本次研究中选取了 8 个特征频率{"125 Hz", "250 Hz ", "500 Hz ", "1 KHz ", "2 KHz ", "4 KHz ", "8 KHz ", "16 KHz "},以此为基础来分析在这些频率上的响度情况。

通过在 PC 端的浏览器中输入开发板的 IP 地址进行访问,可以看到 PCM 信号已经成功转化为频谱图了。PCM 音频数据的频谱图表示如图 9 所示。

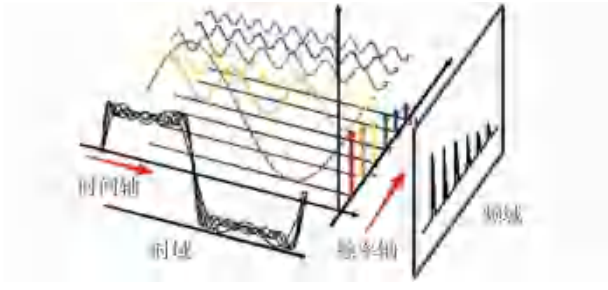


**图 8 PCM 音频数据的时域和频域表示**

**Fig. 8 Time domain and frequency domain representation of PCM audio data**

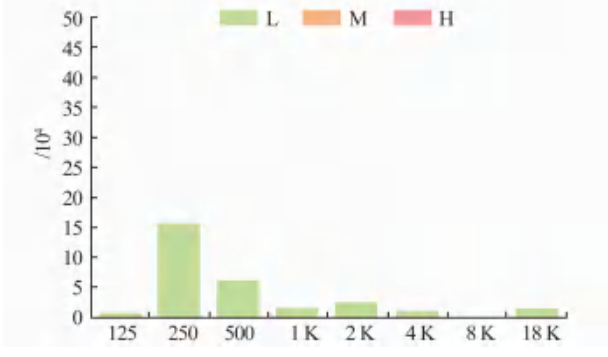


**图 9 PCM 音频数据的频谱图表示**

**Fig. 9 Spectrum representation of PCM audio data**

经过多次实验发现,某些频率上的振幅比较明显,而其他频率上的振幅相对较小,可以认为声音的特性是由这些变化明显的频率所掌控,而其他频率影响较小,一旦明显变化的频率振幅出现问题,将直接影响到目标声音的预测。把预测的好坏由少数频率来控制,会存在高风险,故而需要对其进行标准化处理,使得频谱图中不同的特征频率具有相同的尺度(将特征的值控制在某个范围内),这样目标声音就可以由多个相同尺度的特征频率共同控制,也便于后续训练过程中加速权重参数的收敛。本次研究将振幅情况映射到 0 ~ 128 的区间。

## 3 音频数据集的收集

综合前文所述,已经大致了解如何来收集音频数据,以及如何区分不同音频。但是在模型训练时,需要用到大量连续的音频数据样本,所以还需要完成音频数据的收集。

本次研究拟通过用户语音输入的“开”、“关”来控制 LED 的开启和关闭。因此这里主要来收集用户“开”和“关”的音频数据,统计每次音频采集所获取到的数据条数。实验发现,“开”和“关”这样一个字大约会产生 20 ~ 30 条左右的数据。综合考虑实际测量结果后,文中将使用 30 条频谱数据来描述一个语音指令。

在前文分析基础上,每一条频谱数据又分别采集了 8 个特征频率上的响度值,因此,一个控制指令所包含的数据量为 30 * 8 个值。

但是需要面对开发板重复收集很多次同一个控制指令的数据,比方说“开”字的 100 次的数据。假如在认真喊话的时候数错或者喊错,为了保证训练数据的准确性,就又得重新开始收集。本次仿真时通过在开发板上外接一个 LED 灯,再控制 LED 的颜色来表示采集和结束采集的情况。到达规定次数时,红灯亮起;超过规定次数,蓝灯亮起,表示收集完毕。研究中编写的程序代码见如下。

```
int data_count = 0;
// 当前已经采集的声音次数
int  n = 100;
// 需要采集声音的次数
void Check_Start( ){
if( bsum  >=  Threshold_HIGH&&record_count
==- 1){
data_count ++;
if( data_count == n){
pixels. setPixelColor(0,pixels. Color(255,0,
0));// 红灯
} else if( data_count > n){
pixels. setPixelColor(0,pixels. Color(0,0,
255)); // 蓝灯
}
record_count = 0;
}
```

}

喊完 100 次的时候，红灯亮起，此后将串口监视器中的数据复制到 open. txt 文件，并将该文件名重命名为 open. csv，将其打开则会发现在这张表格中已经存在 3 000 条记录，而且每条记录都是由 8 个频率数据组成。

采用同样的方式，再收集 100 次“关”控制指令的数据集，并保存为 close. csv 文件。除了收集“开”和“关”数据集外，还要收集环境噪音。因为最终要分类的其实是环境噪音、“开”和“关”这三种声音。收集环境噪音十分简单，只要将高阈值调到 0 就可以了，等到红灯亮起，再将这些数据保存为 silence. csv 文件。

## 4　搭建模型完成训练

到目前为止，已经收集了 3 种声音，分别为“开”、“关”和“环境噪音”，每种声音各采集了 100 次的数据集。但将这些数据集提交训练前，还需要进行预处理，主要涉及如下操作：

(1)将采集的每个声音以 30 条数据进行切割，生成一个二维矩阵。

(2)将声音数据进行标准化操作，使每个数据的范围都控制在[0,1]之间。

(3)给声音数据打上数字标签，最终生成一维标签矩阵。

在模型训练时，可以选择本地服务器，也可以选择采用各种云平台。而本次实验采用的是浙江城市学院创建的黑胡桃实验室云平台进行模型的训练。

在本次实验中，决定采用 keras 框架搭建模型进行训练，使用的是 Sequential 模型。在确定好每一层的输入数据尺寸、输出数据尺寸以及激活函数后，使用 *keras. layers. Dense* 定义模型的每一层计算。相应的程序代码参见如下。

```
model. add( keras. layers. Dense(32,input_
shape = (FEATURE_NUM * SAMPLES_PER_
VOICE,), activation = 'relu'))
model. add( keras. layers. Dense(16,activation =
'relu'))
model. add( keras. layers. Dense(3, activation =
'softmax'))
```

分析可知，本实验是一个多分类的问题，因此在输出层使用的激活函数是 *softmax* 函数。最终输出只有“开”、“关”和“环境噪音”三种声音，所以最后一层的输出节点尺寸定义为 3。定义好模型结构之后还需要定义优化器、损失函数、性能评估函数等参数来编译模型。程序代码具体如下。

```
adam = keras. optimizers. Adam(0. 000 01)
model. compile( loss = 'sparse_categorical_
crossentropy',optimizer = adam,metrics = ['sparse_
categorical_accuracy'])
```

在上述的参数设置中，则将学习速率设定为 0. 000 01。一般学习速率越低，训练的时长越长，但是学习效果越好。可执行代码的内容见如下。

```
history = model. fit(xTrain, yTrain, batch_
size = 1, validation_data = (xTest, yTest), epochs =
1 000, verbose = 1)
```

设置 *epoch* 为 1 000，也就是经过 1 000 步后结束训练。当然这个步数需要根据实际情况进行调整。本实验中由于数据量比较小，不会占用太多的处理器内存，*batch_size* 设置为 1，这些数据会一次性加载到处理器中进行训练。*verbose* = 1 表示在训练过程中会有进度条记录输出。

为了最终能将训练好的模型运用到开发板上，在训练结束后，需要使用模型转换器将模型转换为 TFLite 格式，再将转换的结果从内存中捞出保存到磁盘中。与其相关联的代码可写为如下形式。

```
converter = tf. lite. TFLiteConverter. from_keras_
model(model)
tflite_model = converter. convert( )
// 保存 TFLite 格式的模型到磁盘
open("model", "wb"). write(tflite_model)
```

但是，这个模型还不能直接输送到开发板上去运行，还需要对其进行一定的转换，转换为能被 C 语言加载的静态数组形式。可使用 linu 命令 xxd-i 来将二进制文件内容存储到 C 代码静态数组内，最终生成 model. h 格式的文件。

## 5　模型推理

在前文的模型搭建和模型训练研发的基础上，此时的开发板已经实现智能化，能够通过用户的语音输入来区分“开”、“关”、“环境噪音”等不同声音。

修改原来的程序，将原本通过串口输出的数据直接输入到模型中去进行推理，使用 *tflInputTensor* 将一组 30 条、每条 8 种响度值的数据输入到模型中。需要注意的是，这些数据在输入前也需要经过与模型训练时一样的标准化操作，这里也是将各种频率的响度值除以 128，使每个数据的范围也都控

制在[0,1]之间。

一次声音数据输入完毕后,就使用 *Invoke* 进行推理。并且在完成推理后,将推理结果进行输出,使用 *tflOutputTensor* 来取出这 3 个声音的准确率。程序代码内容详见如下。

```
TfLiteStatus invokeStatus = tflInterpreter -> Invoke();
for (int i = 0; i < NUM_VOICES; i ++) {
Serial.print(VOICES[i]);
Serial.print(": ");
Serial.println(tflOutputTensor -> data.f[i], 6);
}
```

使用 TFLite 模型进行模型推理如图 10 所示。图 10 中,分别展示了每次用户语音输入后,判定为“开”、“关”、“环境噪音”的可能性,最终以百分比的数值进行显示。百分比越高,表明此次用户输入的音频数据为对应的类别可能性就越高。实验证明,环境噪音对最终识别的准确性有一定的影响,在上述第四次实验中,随着环境噪音的增强,识别的结果受到了一定的影响。


**图 10 使用 TFLite 模型进行模型推理**

**Fig. 10 Using TFLite model for model reasoning**

## 6 实现个性化的灯光控制

至此,研究中将使用 LED 灯来真实反映语音识别的结果。首先删除前面收集数据集时的灯光效果,并在推理结束后对推理结果进行判断,从而控制 LED 的开关。程序代码可依次展开如下。

```
float silence = tflOutputTensor -> data.f[0];
float open = tflOutputTensor -> data.f[1];
float close = tflOutputTensor -> data.f[2];
if(silence == 1){
}else{
if(open > close&&open > 0.6){
pixels.clear(); // 调颜色
pixels.setPixelColor(0,pixels.Color(255,255,255)); // 调颜色
}else if(close > open&&close > 0.6){
pixels.clear(); // 调颜色
pixels.setPixelColor(0,pixels.Color(0,0,0)); // 调颜色
}
}
```

因考虑到环境噪音对语音识别结果的影响,在上述的程序中,并没有采用 100% 的概率来进行判定,而是采用判定用户输入的音频数据达到 60% 以上的概率为“开”或者“关”的时候,则执行相应的操作。实验证明,当测试者对着开发板喊出“开”的指令后,LED 灯就会自动开启,再对着开发板喊出“关”的指令后,LED 灯会立刻关闭。

## 7 实验测试及思考

到目前为止,本实验已经可以通过语音输入“开”、“关”的指令来控制 LED 的开启和关闭操作,而且通过前面的分析,每个人发出的声音的频率不同,以及在每个频率上的响度不同等特点,再进行模型训练等操作后,可以达到个性化的灯光控制效果。

实验中还发现,识别的准确率并不能达到 100%,可能跟多种因素有关。例如,声音采集时所处的环境和最后推理时的环境不一致,从而造成了环境噪音的不一致,影响了识别的准确率。在采集 100 次“开”、“关”的数据集时,特别是在采集初期,因为人还没有喊话,而开发板却已经开始采集数据了,这时候其实采集到的仅仅是环境噪音的数据,却被用户认定为“开”或者“关”指令的数据,也在一定程度上影响了识别的准确率。

另外,还会发现一个问题,这套程序只能采集并识别话音起始的那一个字,而人们平时说话很显然并不是使用单个字的发音进行沟通交流的。一句话可以由多个字所构成,机器认识的那个字,并不一定是这句话的第一个字,就比如芝麻开门这句话,“开”是位于中间位置的,那么对于这种情况的处理方式,也是亟待不断探索和尝试的问题。

## 8 结束语

未来势必是万物在线—智联网的时代,曾有专家说过,物联网的本质首先必须是智联网,只有在连起来之后,才能实现智能化。智能音箱的出现,用语

(下转第 206 页)