

题 目 **基于树莓派的IoT语音控制和认证系统设计与实现**

**计算机科学与工程**院（系）**计算机科学与技术**专业

|  |  |
| --- | --- |
| 学　　号 | **09013322** |
| 学生姓名 | **张洋铭** |
| 指导教师 | **凌　振** |
| 起止日期 | **2017年3月1日—2017年5月22日** |
| 设计地点 | **东南大学九龙湖校区计算机楼** |

论文独创性声明

本人郑重声明：此处所呈交的本科毕业论文《基于树莓派的IoT语音控制和认证系统设计与实现》，是本人在导师指导下进行的研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或者撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明。

作者签名：

摘 要

如今，物联网（Internet of Things, IoT）技术得到了长足的发展，并且逐渐从工业、军事等领域走向千家万户，走进普通人的生活中。随着IoT技术的普及和广泛应用，其所带来的安全性问题愈发引起人们重视，一个突出的安全性问题是对于IoT设备的访问控制。为了加强IoT设备的安全性，本论文引入IoT语音网关，通过语音识别和认证技术实现对IoT设备的访问控制。语音识别和认证，作为人工智能、生物识别等领域的热门技术，得到了越来越广泛的关注，中国学术界和产业界在语音技术领域也做出了卓越的贡献。

本论文设计并实现了一个基于树莓派（Raspberry Pi）的IoT语音控制和认证系统。文中首先进行需求分析并选定软硬件平台、介绍系统的总体架构和运行流程，随后分别论述语音识别、语音认证和MQTT通信三个功能模块的理论、技术，以及基于CMU Sphinx、ALIZE、mosquito等工具集的开发，并对语音识别、语音认证和MQTT通信三个功能模块进行测试和分析。此外，文中也说明了作为IoT终端设备的ESP8266无线网络模块的结构、特点、软硬件开发方式和在嵌入式系统上实现MQTT应用层协议的方法。最后，将各个子系统集成部署在树莓派平台上测试运行，运行结果表明该系统可用。

**关键词**：物联网，语音控制，语音认证，树莓派

**Design and Implementation of a Raspberry Pi based IoT Voice Control and Authentication System**

**ABSTRACT**

We have witnessed the rapid growth and widely application in industry, military and home of the technology of Internet of Things (IoT). With fast and widely spreading of applications on IoT, safety issues on IoT are becoming increasingly serious. One of these issues is access control for devices linked to IoT networks. In order to enhance the safety of IoT devices, we introduced IoT speech gateway in this paper. As parts of technologies on artificial intelligence and biometrics, voice recognition and voice identification has attracted more and more attention of scholars and industry circle. Chinese academics and industry have made a significant contribution on this field.

In this paper, we designed and implemented an IoT voice-control and voice-identification system based on a single board computer which is called Raspberry Pi. First of all, we made a requirement analysis and determined the platform of software and hardware, and introduced the architecture and operating procedure of this system. Then, we demonstrated theories and technologies of voice recognition, voice identification and MQTT communication, and re-development based on open-source libraries and toolkits such as CMU Sphinx, ALIZE, mosquitto. Tests on these functional modules are accomplished in order to assemble them later. The results of the tests are also analyzed. Moreover, we demonstrated the structure, features, methods of software and hardware developing, and the route to implement the MQTT application-layer protocol on embedded systems of the ESP8266 wireless network module as IoT terminal device. Finally, we integrated the subsystems and configured the completed system installed on Raspberry Pi platform in order to run overall testing. The test result validates its feasibility and effectiveness.

**KEY WORDS**: IoT, voice control, voice identification, Raspberry Pi

**目　　录**

论文独创性声明 I

摘要 II

ABSTRACT III

[第一章 绪论 1](#_Toc483235853)

[1.1 选题背景 1](#_Toc483235854)

[1.2 研究现状 1](#_Toc483235855)

[1.2.1 IoT硬件设备 1](#_Toc483235856)

[1.2.2 语音控制 2](#_Toc483235857)

[1.2.3 IoT网络通信 2](#_Toc483235858)

[1.3 研究目标和研究内容 3](#_Toc483235859)

[1.4 论文组织结构 3](#_Toc483235860)

[第二章 系统总体设计 5](#_Toc483235861)

[2.1 需求分析与平台选择 5](#_Toc483235862)

[2.1.1 树莓派单板计算机 5](#_Toc483235863)

[2.1.2 ESP8266无线网络通信模块 7](#_Toc483235864)

[2.2 架构设计 8](#_Toc483235865)

[2.2.1 IoT网络拓扑 8](#_Toc483235866)

[2.2.2 软件系统架构 9](#_Toc483235867)

[2.3 流程设计 10](#_Toc483235868)

[2.4 IoT终端设备的软硬件设计和实现 12](#_Toc483235869)

[2.4.1 传感器读取 12](#_Toc483235870)

[2.4.2 GPIO操作 13](#_Toc483235871)

[第三章 语音识别 14](#_Toc483235872)

[3.1 Sphinx语音识别引擎介绍 14](#_Toc483235873)

[3.2 语音识别的基本理论和算法 14](#_Toc483235874)

[3.2.1 语音编码与特征提取 14](#_Toc483235875)

[3.2.2隐马尔可夫模型（HMM） 16](#_Toc483235876)

[3.3 Sphinx安装部署 17](#_Toc483235877)

[3.3.1 依赖软件和PocketSphinx安装 17](#_Toc483235878)

[3.3.2 语言模型生成 18](#_Toc483235879)

[3.3.3 执行语音识别 19](#_Toc483235880)

[3.4 本章小结 20](#_Toc483235881)

[第四章 语音认证 21](#_Toc483235882)

[4.1 ALIZE工具集介绍与部署 21](#_Toc483235883)

[4.2 语音认证的基本理论和算法流程 22](#_Toc483235884)

[4.2.1 基于GMM-UBM的语音认证技术 22](#_Toc483235885)

[4.2.2 分类判定规则 23](#_Toc483235886)

[4.3 GMM-UBM模型训练与结果决策 23](#_Toc483235887)

[4.3.1 MFCC参数提取 23](#_Toc483235888)

[4.3.2 静音去除与特征参数标准化 24](#_Toc483235889)

[4.3.3 UBM-GMM训练和判定分数标准化 24](#_Toc483235890)

[4.4 本章小结 25](#_Toc483235891)

[第五章 MQTT协议及其实现 26](#_Toc483235892)

[5.1 MQTT协议介绍 26](#_Toc483235893)

[5.2 MQTT协议的实现 27](#_Toc483235894)

[5.2.1 树莓派MQTT服务器的实现 27](#_Toc483235895)

[5.2.2 ESP8266的MQTT客户端实现 28](#_Toc483235896)

[5.3 MQTT消息传输测试 30](#_Toc483235897)

[5.4 本章小结 31](#_Toc483235898)

[第六章 系统实现与测试 32](#_Toc483235899)

[6.1 流程控制程序 32](#_Toc483235900)

[6.2 功能测试 33](#_Toc483235901)

[6.3 语音识别和认证性能测试 36](#_Toc483235902)

[第七章 总结与展望 39](#_Toc483235903)

[7.1 论文总结 39](#_Toc483235904)

[7.2 工作展望 39](#_Toc483235905)

参考文献 IV

致谢 V

# 第一章 绪论

## 1.1 选题背景

随着网络通信技术、计算机技术和传感控制技术的发展，将原本不具有计算能力或者计算能力很弱的设备接入互联网，实现“万物互联”，已经成为可能。万物互联形成的全新网络形态，称为物联网（Internet of Things, IoT）。IoT将众多传感器、伺服器和计算机集成在同一个网络中，使得网络拥有了与现实世界进行交互的能力。IoT通过大量的、时刻在线的传感器从物理世界收集大量信息，通过互联网上强大的计算能力对现实世界进行分析，再通过伺服器对物理世界进行操作，真正将互联网上的信息流下沉到真实世界中，实现全方位的自动控制，极大地节约了人类的精力。

目前，IoT除了在工业、军事等领域得到广泛应用之外，家居自动化也是IoT发挥其能力的一个重要领域。针对家居自动化的需求特点，IoT设备必须支持足够友好的人机交互方式，而语音交互是最为友好的人机交互方式之一，也是未来大有前途的发展方向。自动语音识别（Auto Speech Recognition, ASR）技术可以将人类语音转换成机器可识读的文字符号，利用该技术可以实现语音交互。

强大的IoT汇聚着大量关于现实世界的信息，这些信息中往往蕴含着大量个人隐私，即便对网络中传输的数据流量进行加密，也无法完全避免隐私的泄露[1]。因此，对于IoT设备的访问控制，是IoT系统实现中必须考虑的一个重大安全性问题。声纹识别（Voiceprint Recognition, VPR）技术是生物识别技术的一种，该技术可以和语音识别技术一起，实现对IoT设备的人机交互和访问控制。

语音控制和语音认证二者的结合，既可以提高IoT人机交互效率，同时也可以保证IoT应用的安全性，解决了IoT应用领域的一个实际问题。本文将IoT语音网关引入IoT网络，通过环境搭建、系统部署、参数训练与调试，利用语音识别和认证技术，控制IoT网关通过MQTT协议对IoT终端设备进行读取和控制。

## 1.2 研究现状

### 1.2.1 IoT硬件设备

如今，学术界和产业界对IoT相关技术都非常感兴趣，相关的研究工作方兴未艾，同时也吸引了大量创客和电子爱好者的兴趣。世界知名CPU厂商Intel顺应潮流，推出了Galileo、Edison、Curie、Joule等一系列面向IoT的开发套件，中兴、Apple、Amazon、小米等企业也在积极布局IoT时代的产品体系，并且有成熟的产品在售。基于树莓派、Arduino、Intel Galileo、Intel Edison、Beagle Bone、NanoPi等嵌入式单板机平台的IoT相关开发，也已经有了不少研究成果。

作为一款流行的嵌入式单板机，树莓派强大的性能、庞大的开发者群体和完善的软硬件生态环境使得它颇受欢迎。文献[2][3][4]集中介绍和论述了树莓派平台的参数、结构、性能、软硬件平台、基本使用方法和进阶使用方法等内容，根据这些文献的介绍可知，尽管性能仍然有限，但树莓派完全能够胜任比较复杂的信号处理和控制工作。文献[5]便论述了一款基于树莓派的智能家居系统的设计。文献[6]则论述了在树莓派平台上实现智能家居系统的更多技术。作者在该文献中，介绍了ZigBee等IoT领域的无线通信经典解决方案，并且在树莓派上搭建了Web服务器，充分发挥了树莓派作为IoT 网关的作用。

### 1.2.2 语音控制

就系统整体而言，对于IoT设备的语音控制，市场上已经有若干款比较成熟的产品，例如Amazon Echo[7]，Apple HomeKit[8]等。这些产品往往与厂商产品线上的其他产品紧密集成，用户体验良好。尽管可以通过厂商提供的开发者接口进行二次开发，但是基本功能被厂商提供的API严格限制，并且不同厂商之间的同类产品不能兼容。在安全性和隐私保障方面，由于这些产品高度依赖服务器端（即“云端”）的存储和计算能力，因此存在隐私泄露、IoT设备遭攻击的风险，而这种风险是用户端完全不可控的。树莓派作为一款流行的、高度可扩展的软硬件平台，吸引了大量开发者对其进行开发。Tanay Pant等人[9]则已经在树莓派上实现了可用的智能家居助手。

就语音识别子系统而言，语音识别技术无论是在理论上还是在产品上都已经非常成熟。语音MFCC特征参数是最为可靠的特征参数之一，在语音识别和语音认证两个环节都需要用到MFCC参数。在语音识别领域，基于HMM模型的语音识别算法是一款常用且有效的方法[10]。为降低重复开发程度，目前有众多成熟可靠的语音识别工具可以选择。国外有CMU Sphinx[11]、ALIZE[12]等语音识别工具，国内有科大讯飞[13]、百度语音识别[14]等。

就语音认证子系统而言，语音认证（即说话人识别）的核心技术是声纹识别技术。声纹识别在刑事侦查、隐私安全保护等领域发挥重要作用。基于高斯混合模型（GMM）的声纹识别技术已经非常成熟[15]，文献[16]论述了基于MFCC参数和SVM（支持向量机）的说话人识别算法，文献[17]则将小波包分析引入说话人识别算法。基于i-vector方法的说话人识别算法[22]也得到了广泛应用。目前已经有开源的说话人识别工具库ALIZE[18] ，支持GMM和i-vector两种方法。

### 1.2.3 IoT网络通信

在IoT网络通信方面，已经有不少适合于IoT场景的轻量级通信协议。在无线通信领域，Wi-Fi和ZigBee是两种常用的物理层手段。文献[6]利用ZigBee无线通信模块实现了IoT终端设备之间的无线通信。ZigBee无线通信具有自组网能力，并且功耗较小，但其成本较高，通信速率不如Wi-Fi，并且容易受干扰。基于Wi-Fi的无线通信则非常灵活，兼容现有无线网络。应用层协议方面，主要有非常常见的HTTP协议和其他适合于IoT领域的轻量级通信协议。HTTP协议一般用于具有Web服务器的场合，如文献[6]实现的系统。Amaran等人的文章[19]中对CoAP、MQTT、MQTT-SN等若干种常见的轻量级通信协议做了比较。MQTT协议非常适合IoT场景下的通信，文献[20][21]论述了MQTT在IoT领域和即时通讯领域的应用。在家庭IoT领域，为避免布线的复杂性，除了利用无线通信实现IoT设备之间的互联互通外，文献[6]介绍了X-10电力线载波通信技术。该技术作为有线通信技术的一种，其利用50Hz或60Hz的市电作为载波，再以120kHz的脉冲对信号作调制，使数据信号在设备间经电力线传输，但这种通信方式存在数据速率低下、安全性差等缺点。

## 1.3 研究目标和研究内容

通过对该领域其他工作的分析和比较，本文确定了以下几项研究目标：

1.在树莓派平台上实现语音识别和说话人识别引擎，并通过反复训练、调参，优化语音识别和语音认证的性能和效果；

2.分别在树莓派平台和ESP8266模块上实现MQTT应用层协议，并实现IoT网关与IoT终端设备的通信；

3. 基于ESP8266模块进行IoT终端设备的软硬件综合开发和调试，包括GPIO控制和传感器读取两大方面。这两方面是对外设端口的各种操作的代表，具有通用性和代表意义，若系统可以实现GPIO控制和传感器读取，意味着系统在理论上可以进行绝大多数端口操作。

4. 熟悉基于树莓派硬件平台和Debian Linux操作系统的应用程序开发，通过Python脚本语言，将语音识别、语音认证、MQTT通信等功能流程串联起来，并集成到树莓派平台，并完善系统的运行流程和交互体验。

最终实现的系统具备以下功能：

1. 可通过特定说话人的语音控制ESP8266模块上的GPIO、读取连接在ESP8266模块上的传感器数据，若说话人不符，则语音验证不通过，不执行动作；

2. 系统在硬件上提供一个录音按钮和一个状态指示灯，按住录音按钮可以录音，松手后开始语音认证和识别，若认证通过，则根据语音识别的结果执行相应动作。

本文的研究内容有以下三个方面：

1. 语音识别、说话人识别的模型训练、结果判决方法；

2. 深入研究MQTT协议，并实现MQTT的客户端/服务端通信；

3. 在树莓派平台上进行开发的一般方法和步骤。

## 1.4 论文组织结构

本论文正文分为七章，分别论述以下内容：

第一章为绪论，介绍本文的选题背景以及相关技术的研究现状，明确设计目标，并介绍文章组织结构。

第二章阐述系统总体设计。本章从需求分析入手，引入开发所需的树莓派平台和ESP8266模块，并从系统全局的视角说明了系统的架构和流程设计。此外，本章还详细阐述了IoT终端设备侧ESP8266模块的GPIO控制和传感器读取功能的实现。

第三章阐述语音识别功能。本章首先介绍Sphinx语音识别引擎以及语音识别的基本理论和算法，然后说明在Linux系统中部署Sphinx的方法。

第四章阐述语音认证功能。本章首先介绍ALIZE说话人识别工具集及其所使用的基于GMM-UBM的说话人识别算法。随后说明利用训练样本对GMM-UBM模型进行训练，并利用该模型对待判定样本进行分类决策的方法。在此基础上，探索了判定性能调优的方法。

第五章论述MQTT协议的实现。本章首先深入分析MQTT协议，然后在ESP8266的传输层接口上实现了MQTT应用层协议。

第六章阐述系统的实现和测试。本章首先论述了系统主流程的Python实现方式，实现各个子系统的集成，随后对系统整体的功能和性能进行测试分析。

第七章为总结与展望，总结了文中工作的可取经验和尚存不足，并指出针对不足之处的改进方案。随后对本文涉及的各技术领域的未来发展进行展望。

# 第二章 系统总体设计

## 2.1 需求分析与平台选择

IoT正在以前所未有的面貌改变着我们的生活。目前，市面上的IoT设备不少，但是几乎都存在着这样几个问题：

一、交互和部署成本仍然偏高：很多IoT设备是通过手机应用程序（下文简称“App”）和Web浏览器进行操作的，这本身就是与物理世界的脱离，没有充分调动用户的感官和肢体。例如Belkin WeMo智能插座，只能通过手机App和浏览器的方式进行开关，如果用户就在家中，可能不如亲自去拔插插头来得方便。

二、功能单一，性能孱弱，性价比偏低：IoT设备往往功能单一，因此在性价比方面，要么失之强大，要么失之实惠。例如，Apple WATCH是一款非常出众的IoT终端设备，但是其相对高昂的售价和相对简单的功能，使得它并没有那么普及。

三、通信协议和后台平台不统一，兼容性差：Apple、Amazon、小米、Google等行业巨头都推出了各自的IoT终端产品和服务，但是，这些设备尽管使用通用的底层通信协议、电子技术，但是在上层的应用层面上，仍然不能实现有效整合。例如，Apple推出的HomeKit兼容设备，就只能由HomeKit平台的iPhone等终端设备进行操作。

四、安全性不够高。IoT网络上存储着大量隐私信息，如果被轻易攻破，则隐私安全完全无法保障。

针对上述第一点问题，本课题选择语音交互作为主要交互方式，并辅以物理按键、Web浏览器等备用交互方式，也为高级用户和开发者预留了API和控制台。针对上述第二点问题，我们选用Raspberry Pi（下文简称“树莓派”）作为IoT网关平台。目前最新的树莓派3代B型具备很高的性能，性价比极高，扩展性极强，是实现本课题预定目标的理想平台。针对上述第三点问题，开发中尽可能采用开源的、通用的、简单易维护、易二次开发的库、工具和接口等，既可以保证成品的兼容性，也可以保证可扩展性，供高级用户进行进一步的开发。针对上述第四点问题，为与语音控制相结合，本课题选择语音认证方式控制用户对网络的访问。

### 2.1.1 树莓派单板计算机

本文中，语音相关的应用属于计算密集型应用，因此需要一个性价比高、能耗比高、性能足够强大的计算机平台来运行语音识别和语音认证算法，并控制系统运行流程。体积小、性能强、性价比高、扩展性强的嵌入式单板计算机非常适合IoT领域的要求。在嵌入式Linux单板机领域，有众多成熟可靠、生态健全的开发板产品可以选择。Raspberry Pi（树莓派，图2-1）是一款由英国树莓派基金会出品的单板微型计算机系统，其最新版本是2016年发布的树莓派3代B型。树莓派3代B型配置如下：CPU为主频1.2 GHz的64位ARM Cortex-A53核心，采用ARMv8-A (64/32-bit)指令集，主存1GB（与GPU共享），具有以太网、USB2.0、802.11n无线Wi-Fi网络适配器、Bluetooth、HDMI、IIC、SPI、IIS以及17个GPIO（General Purpose Input/output，通用输入输出端口）等接口资源。树莓派可以运行Linux操作系统，甚至最新的Windows IoT操作系统。在本文中，使用的是树莓派官方提供的，基于Debian Linux的Raspbian系统。

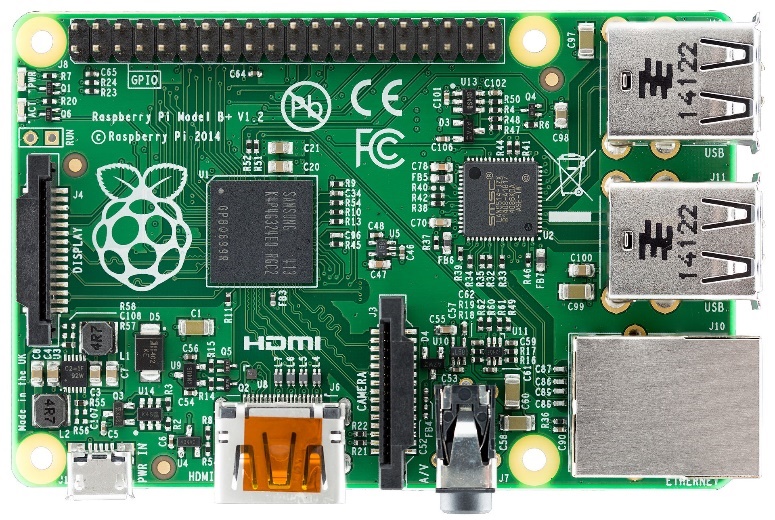


图2-1 树莓派单板计算机[36]

由于树莓派性能强大，且具有网络接口，因此，选用树莓派作为IoT网关，处理重量级的语音处理任务和通信任务，对所有IoT设备进行集中控制。另外，树莓派提供了丰富的GPIO资源（图2-2），我们将利用这些GPIO资源，丰富IoT设备与用户的交互体验。

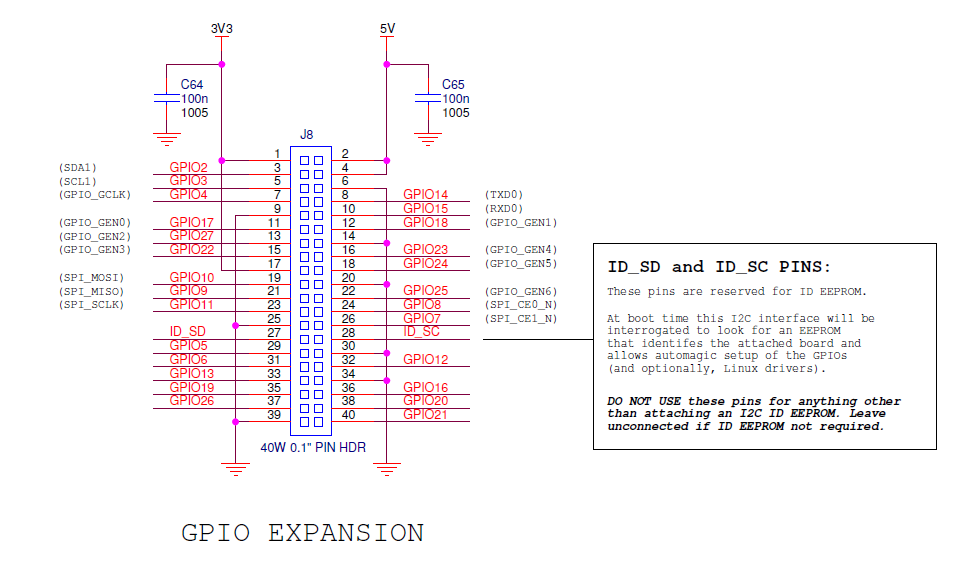


图2-2 树莓派提供的部分IO资源[37]

### 2.1.2 ESP8266无线网络通信模块

ESP8266是上海乐鑫出品的一款低功耗、高性价比SoC（System on Chip，片上系统）。ESP8266在单个芯片内集成了32位MCU和802.11射频电路模块，将嵌入式微控制器和Wi-Fi网络通信的功能集成在同一块芯片上，极大地方便了开发人员。

市面上可以买到的一般都是基于ESP8266的Wi-Fi模块（图2-3），型号为ESP12。该模块除含有ESP8266芯片之外，还有板载天线、板载LED指示灯、8Mb（即1MB）SPI Flash和若干IO口。

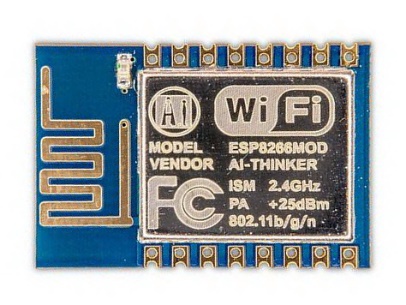


图2-3 ESP8266无线模块

模块的IO端口配置如图2-4所示。从图中可见，除GPIO外，模块还提供了与GPIO复用的UART串口、SPI接口、IIC接口、模拟输入等端口。丰富的端口外设为我们的功能开发提供了极大的发挥空间。

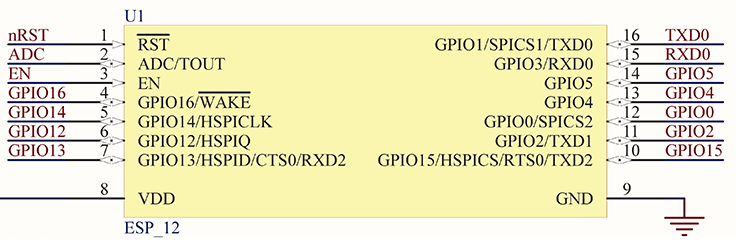


图2-4 ESP8266模块提供的接口资源[38]

为方便软件开发和硬件搭建，我们选用了NodeMCU开发平台。该平台内置CH340串口芯片，可通过USB直接下载程序，并且将ESP模块的邮票孔端口全部以排针形式引出，方便硬件搭建。NodeMCU的外观和引脚配置如图2-5所示。

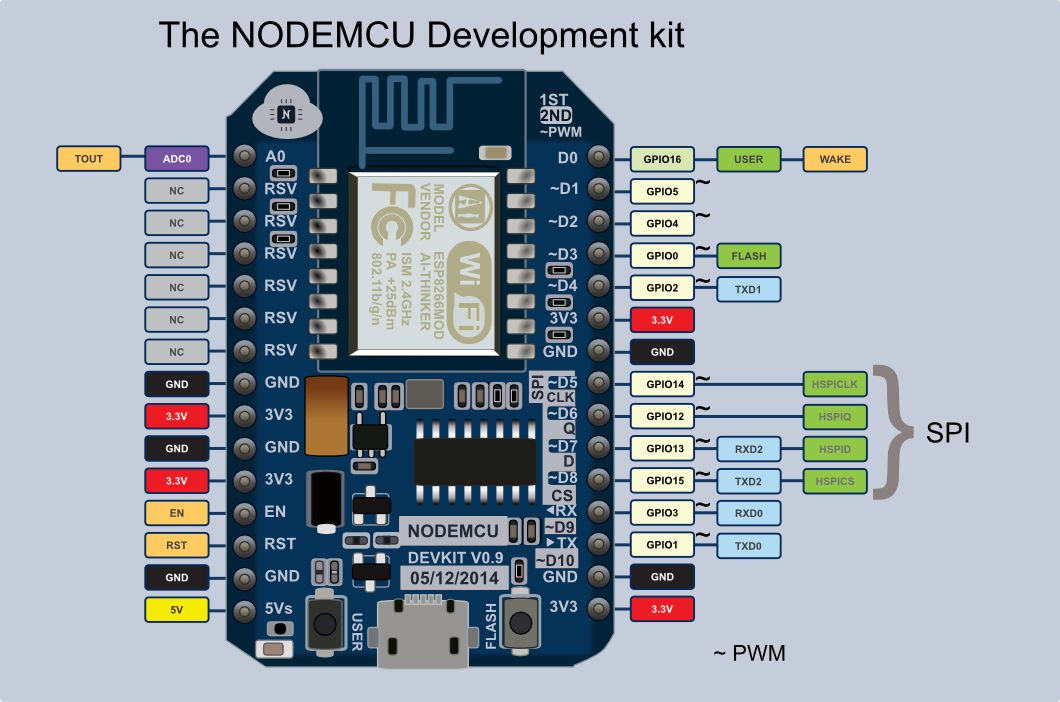


图2-5 NodeMCU开发板的端口资源[38]

## 2.2 架构设计

本节将分别从硬件网络拓扑和树莓派软件架构两方面说明系统的总体结构。

### 2.2.1 IoT网络拓扑

若干IoT设备组成一个IoT局域网。每个IoT设备都可以通过有线或无线的方式接入局域网，并将自身获取到的环境信息上传到网络，或者受网络中其他设备的控制。其中，有一台IoT设备性能较高，负责控制所有其他IoT设备，并与Internet进行通信，该设备称为“IoT网关”，其他受控设备称为“IoT终端”。

如图2-6所示，本系统包含一台IoT网关、一台连接到WAN的路由器，以及若干IoT终端设备。IoT网关和IoT终端设备均连接到作为局域网网关的路由器，并通过该路由器建立对Internet的连接。IoT网关和IoT终端设备可通过802.3以太网或者802.11 Wi-Fi方式连接到路由器，形成一个局域网。局域网的DHCP服务器由路由器担任，并为IoT网关设置静态IP地址。

局域网内部各主机的IP地址设定如下：

- 路由器（默认网关）：192.168.0.1/24

- IoT网关（树莓派）：192.168.0.3/24

- 其他IoT终端设备：192.168.0.4-192.168.0.199/24（DHCP自动分配）

路由器

IoT网关

（树莓派）

IoT终端设备

（ESP8266）

Internet

图2-6 IoT局域网拓扑

在逻辑上，IoT网关与各个IoT终端设备形成虚拟局域网，IoT网关作为IoT虚拟局域网的网关，通过实际网络的IP地址与各个IoT终端设备进行双向通信。所有的网络通信依赖于实际的局域网络，逻辑上的通信是以IoT网关作为交换机，集中控制局域网内所有IoT终端设备，不允许IoT终端设备之间的直接通讯。做出这种设计主要是出于安全性考虑：集中安全压力于IoT网关一身，通过语音认证实现对所有IoT终端节点的访问控制，防止各个节点之间相互牵连。

经过对比和论证，我们选择树莓派作为IoT网关设备，选择ESP8266模块作为IoT终端设备。树莓派和ESP8266均具备丰富的对外接口，这意味着，树莓派可以作为IoT局域网的核心设备——IoT网关。而ESP8266所具有的低功耗、高性价比、丰富的接口资源和内置完整无线网络协议栈，使其具备作为IoT终端设备的能力。

### 2.2.2 软件系统架构

**A) 树莓派软件架构**

树莓派运行基于Linux的Raspbian系统，因此同绝大多数嵌入式Linux系统类似，在系统层次上可划分为应用层和内核层。应用层程序包括语音认证、语音识别、MQTT应用层程序，以及将这三者串联起来形成完整控制流程的控制程序。（图2-7）

Linux用户态程序

语音识别程序

中间件-脚本语言

MQTT协议和外设控制程序

网络协议栈、硬件驱动程序

语音输入

MQTT报文

图2-7 树莓派的软件系统层次

**B) ESP8266软件架构**

与树莓派至少8GB的外部存储空间相比，ESP8266模块只有1MB的Flash存储空间。因此，ESP8266使用了官方定制的小型RTOS，该RTOS提供了访问GPIO、各种总线接口、文件系统的API，甚至集成了完整的TCP/IP协议栈（因为ESP8266硬件上集成有Wi-Fi功能）。我们编写的MQTT应用层协议、GPIO控制、传感器控制等功能程序，实际上都是调用RTOS提供的API（图2-8）。程序开发完毕后，经编译、与RTOS底层代码进行链接，形成一个完整的固件程序文件。为了向ESP8266模块中下载程序，首先按住图2-5中的FLASH按钮，然后将模块的USB线插入PC机的USB端口；随后在Arduino IDE中，选择模块对应的USB-UART端口，点击“下载”按钮，即可将程序下载到ESP8266模块中。下载完成后，模块即自动开始运行新下载的程序。

ESP8266固件

MQTT应用层程序，以及GPIO控制和传感器控制代码

RTOS提供的硬件接口API和TCP/IP协议栈

图2-8 ESP8266的软件架构

## 2.3 流程设计

系统的工作流程如图2-9所示。

用户GMM-UBM模型训练

**系统启动**，授权状态置0

录音键是否被按下？否则等待

执行arecord命令开始录音（44100Hz）

录音键是否被松开？否则等待

结束arecord进程并保存录音

对语音进行说话人匹配度评分

评分是否高于阈值？否则转移

根据识别结果执行相应的动作

（读取传感器，控制GPIO等）

当前授权状态==0？否则跳转

执行arecord命令开始录音（16000Hz）

录音键是否被松开？否则等待

结束arecord进程并保存录音

识别语音内容，提取关键词

录音键是否被按下？否则等待，超时计数器自增。若超时，授权状态置0并跳转

授权状态置1，并初始化超时计数器

图2-9 系统流程

ESP8266连接有温度气压传感器，每五秒向MQTT代理服务器（树莓派）发送一个主题为“sensor”的消息，内容是实时气温和气压。树莓派同时作为sensor主题的订阅者，接收ESP8266发来的气温气压消息并保存到文件。若语音识别结果需要读取传感器数据时，则从文件中读取最新一条消息，反馈给用户，即说话人。若语音识别结果需要控制GPIO时，则树莓派作为MQTT发布者发布一条主题为“gpio”的消息。ESP8266收到主题为gpio的消息推送后，即根据消息内容修改GPIO的状态。

流程控制程序的具体实现将在第六章中进行论述。

## 2.4 IoT终端设备的软硬件设计和实现

IoT网络中设置有大量的传感器和伺服器，带有这些传感器和伺服器的终端设备可能被放置在微波炉、电冰箱、空调、洗衣机、自动窗帘、热水机、甚至是水杯[23]、鞋帽等。因此，IoT终端设备必须是小型化、低功耗、高度集成的。ESP8266模块只有邮票大小，采用3.3V电源电压，Wi-Fi连续在线电流200mA左右，待机功率低至1mA，功耗较低，适合IoT应用。并且，ESP8266自带TCP协议栈，这意味着可以通过编程实现MQTT协议。MQTT协议在ESP8266上的实现将在第五章中进行详细说明。

为提高调试效率，我们选用NodeMCU开发板和Arduino IDE进行开发。

### 2.4.1 传感器读取

IIC总线接口协议，全称Inter-Integrated Circuit，又称I2C协议，是飞利浦公司开发的双总线接口协议，使用广泛。IIC协议包含SCL（Serial Clock，串行时钟线）和SDA（Serial Data，串行数据线）两条漏极开路总线，分别传输时钟和数据。图2-10是参考文档[24]提供的IIC总线配置图。由图可知，在IIC总线网络中，存在主从设备之分。主从设备均可作为发送端或者接收端。

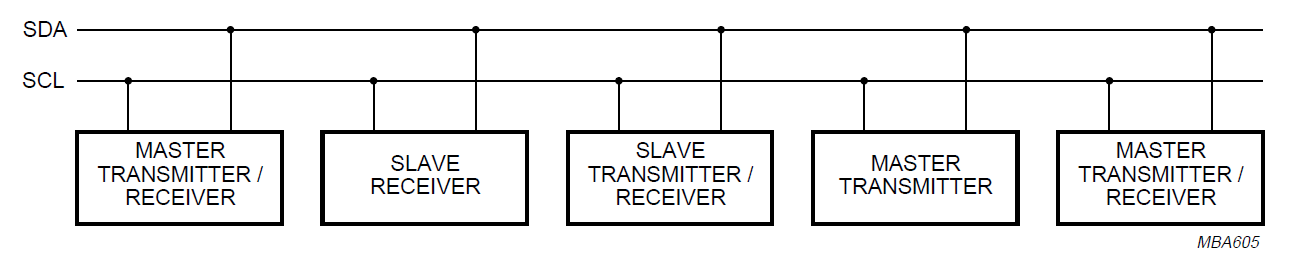


图2-10 IIC总线配置[24]

IIC总线应用极为广泛，许多传感器都采用IIC作为接口标准。本系统中，我们选用采用IIC接口的博世BMP280温度气压传感器[25]。该传感器分辨率高、功耗低，非常适合IoT应用场景。

由于NodeMCU已经在电路板上提供了将SCL、SDA两条IIC信号线拉高到高电平的上拉电阻，因此可直接将ESP8266与BMP280连接，如图2-11。

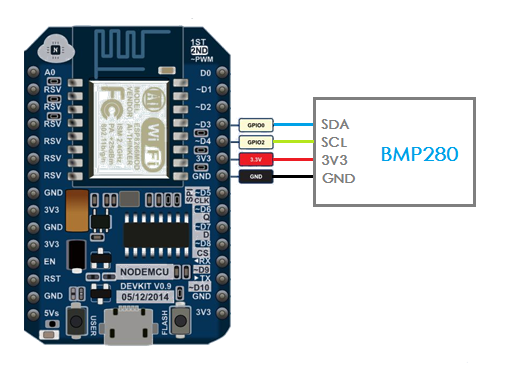


图2-11 BMP280与ESP8266的连接方式

值得强调的是，如果按照图2-11的接线方法，GPIO0接SDA，GPIO2接SCL，需要在代码中对接线方式作出说明：

|  |
| --- |
| Wire.begin(0,2); // GPIO0作为SDA，GPIO2作为SCL |

根据参考文档[25]的规定，按照规定的时序对BMP280内部寄存器进行读写，即可读取温度和气压的测量结果。读取温度和气压的函数被封装为BMP280器件类的两个成员函数：

|  |
| --- |
| float BMP280::readTempC();  float BMP280::readFloatPressure(); |

供ESP8266编程时调用。

### 2.4.2 GPIO操作

Arduino IDE已经将ESP8266的GPIO操作封装为两个C++函数：

|  |
| --- |
| digitalWrite(int pin, int value);  digitalRead(int pin); |

分别对应端口置位和端口读取。

控制GPIO的目的之一，在于驱动继电器以操作强电设备，例如控制窗帘升降电机的升降和启停，控制饮水机发热元件的通断，等等。为了直观显示结果，同时避免调试时操作强电的风险，我们通过GPIO控制NodeMCU开发板左下角的LED灯来显示GPIO状态。LED对应的GPIO编号由宏BUILTIN\_LED指定。

# 第三章 语音识别

本系统可以识别一般的英语词汇，并且可以输出词库中已存在的词汇。系统指定的关键词如表3-1所示。

表3-1 语音识别关键词及对应的测试动作

|  |  |
| --- | --- |
| 关键词 | 动作 |
| HELLO | 点亮LED（拉低ESP8266的GPIO） |
| OPEN | 点亮LED（拉低ESP8266的GPIO） |
| TEMPERATURE | 读取BMP280温度气压传感器测量的温度 |
| PRESSURE | 读取BMP280温度气压传感器测量的气压 |
| CLOSE | 熄灭LED（置高ESP8266的GPIO） |
| GOODBYE | 熄灭LED（置高ESP8266的GPIO） |

## 3.1 Sphinx语音识别引擎介绍

Sphinx[11]是卡内基·梅隆大学的开源语音识别引擎，最初版本由李开复实现。Sphinx运用隐马尔可夫模型以及n-gram模型等算法，可以对连续的、非特定说话人的语音进行识别。

## 3.2 语音识别的基本理论和算法

语音识别，又叫自动语音识别（Automatic Speech Recognition，ASR），目的是将人类语音转化成有意义的字符串。在Sphinx语音识别引擎中，其ASR流程主要分为三步：MFCC（Mel-scale Frequency Cepstral Coefficients，Mel倒谱系数）特征提取、HMM（Hidden Markov Model，隐马尔可夫模型）语音模型训练和基于Viterbi（维特比）动态规划算法的特征匹配。

### 3.2.1 语音编码与特征提取

语音是人类通过发声器官发出的，具有某种语义、遵循某种语法规律的可以传达信息的声音。为了使用计算机处理语音信号，首先要对语音信号进行预处理，将其转化为计算机可以识别的时域离散的数字信号。语音信号的预处理过程主要有采样、量化、编码三个主要步骤（图3-1）。Sphinx所用的音频信号参数如下：

- 采样率44100Hz，根据奈奎斯特采样定理，该采样率可保留22050Hz的音频带宽，接近人耳听觉极限，足够语音识别之用。

- 采样位深16位，16位的位深可提供

dB

的动态范围，这个动态范围对于语音处理来讲已经足够了。

- 采用脉冲编码调制（PCM）方式，该编码方式也为微软WAVE格式所采用，Sphinx接受WAVE格式的数字音频。

经过处理，语音信号被转化成PCM（脉冲编码调制）编码格式（也可以是SPHERE等其他格式）的语音数据文件。Sphinx语音识别引擎可以处理PCM编码的语音文件。

采样

44.100kHz

量化

16bit Signed

编码

脉冲编码调制

图3-1 语音信号编码

语音信号是一种典型的非平稳随机信号，这意味着，语音信号是“随机信号中的随机信号”，在整个时间轴上都不遵循某种特定的随机分布。如果对语音信号在全时域范围内进行特征提取，得到的结果是没有意义的。由于语音信号的分布特性随时间的变动并不频繁[26]，因此可以假定，在一个短的时间窗口内，语音信号是平稳的。为提取短时间窗口内的语音信号特征，首先需要对语音信号进行分帧加窗（图3-2）。对于语音信号，窗宽一般为20-30毫秒，相邻窗口间有10到15毫秒的重叠。为避免单纯截断信号带来的频谱能量泄露问题，需要使用窗口函数对信号进行加窗。常用的窗口函数有Hamming（汉明）窗、Hanning（汉宁）窗等。Sphinx可以自由选择需要使用的窗口函数。

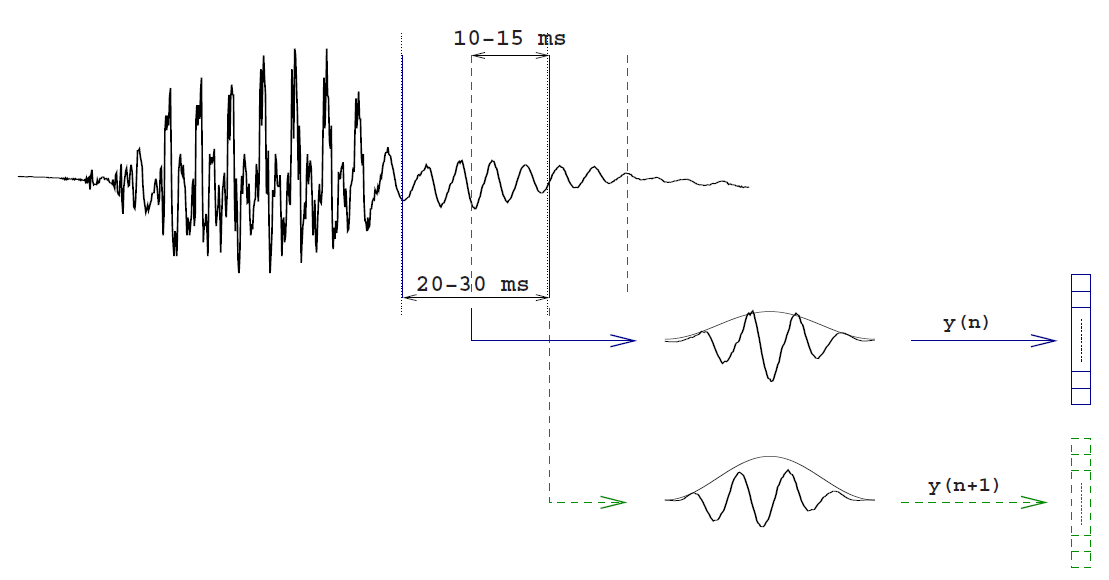


图3-2 语音信号分帧与加窗示意图[35]

经加窗处理得到的信号帧随后经一阶高通滤波器

处理，以提升高频部分。该过程称为“预加重”。经分帧、加窗和预加重的语音信号帧可以视为短时平稳信号，可用于语音参数提取。

一般而言，信号的时间域对特征提取的帮助不大，因此需要对预加重后的语音信号进行FFT（快速傅里叶变换），以获得其频谱。

由于人耳本身可以视为滤波器组，因此语音信号经过人耳会产生失真。为了模拟人耳的滤波器特性，因此首先对语音帧频率刻度进行Mel弯曲，以适应人耳的听觉特性:

随后在Mel弯曲的频率轴上采用由若干个三角滤波器组成的Mel滤波器组，对语音信号帧进行滤波，得到若干个输出。随后，对输出结果进行DCT（离散余弦变换），取变换结果向量的前n个分量（Sphinx是39个），即为每帧的Mel倒谱系数（Mel-scale Frequency Cepstral Coefficients，MFCC）。

### 3.2.2隐马尔可夫模型（HMM）

马尔可夫过程（Markov Process）是一种离散随机过程。一阶马尔可夫过程相当于一个随机转移的自动机，它包含三个要素：状态集、初始状态向量和状态转移矩阵。该过程满足两个约束条件：

- 状态转移矩阵是时不变的，即状态间转换的概率是不变的；

- *t*时刻的状态仅与(*t*-1)时刻的状态相关。

如果系统有*n*个可能的状态，那么可以用一个*n*×*n*大小的矩阵描述任何两个状态之间转换的概率，这个矩阵就称为状态转移矩阵。

下面举例说明马尔可夫过程在语音识别领域的应用。

假设一句话只有两个字的长度，每个字都只可能是“你”“好”两个字之一。现在希望利用马尔可夫模型来解决这样一个问题：已知第一个字是“你”，那么下一个字最有可能是什么？

根据问题描述，可以知道，该随机过程的状态集是{你，好}，初始状态向量是(你：1，好：0)，为了解决问题，需要得到状态转移矩阵。为了得到状态转移矩阵，使用大量的语料去估计这个矩阵（该过程称为模型的训练），暂时不考虑具体的估计方法，根据自然语言的经验，我们可以得到如图3-3的状态转移模型。

0.01

0.10

0.99

0.90

图3-3 仅有“你”“好”两个字的语言的状态转移自动机

对应的状态转移矩阵为（表3-2）

|  |  |  |
| --- | --- | --- |
|  | 第二个字：你 | 第二个字：好 |
| 第一个字：你 | 0.01 | 0.99 |
| 第一个字：好 | 0.10 | 0.90 |

表3-2 仅有“你”“好”两个字的语言的状态转移矩阵

有了这些信息，就有了完整的马尔可夫模型。现在可计算第二个字分别取“你”和“好”的概率：

*P*(第二个字是“你”)

=*P*(第一个字是“你”)×*P*(从“你”到“你”)+*P*(第一个字是“好”)×*P*(从“好”到“你”)

=1×0.01+0×0.10

=0.01

*P*(第二个字是“好”)

=*P*(第一个字是“你”)×*P*(从“你”到“好”)+*P*(第一个字是“好”)×*P*(从“好”到“好”)

=1×0.99+0×0.90

=0.99

由于*P*(第二个字是“你”)<< *P*(第二个字是“好”)，因此可以认为，第一个字是“你”的情况下，整句话极有可能是“你好”。

然而，在语音识别流程中，我们直接获得的是语音片段的MFCC参数向量，而MFCC参数向量与语音的内容之间的关系则尚未知晓。如果说MFCC参数向量是观察结果，那么MFCC参数向量所对应的词汇就是“隐藏”状态。例如，一段语音提取出了“ni”的发音特征，但是“ni”只是“你”“拟”“逆”“泥”等众多可能的隐藏状态的表象。隐藏状态与观察结果之间的转换概率同样可以构成一个矩阵，这个矩阵称为对应概率矩阵。

观测序列*O*、隐状态集合*S*、初始隐状态向量*V*、隐状态转移矩阵*A*和对应概率矩阵*B*，由这五个元素组成的五元组即可描述一个HMM。

在语音识别技术中，语音特征参数组成的集合即为观测序列，所有可能的词汇组成的词汇集即为隐状态集合。对于一个特定的语料库，可以通过前向后向算法对初始隐状态向量*V*、隐状态转移矩阵*A*和对应概率矩阵*B*进行估计，*V*、*A*、*B*三者称为HMM的模型参数，通过前向后向算法对HMM模型参数进行估计的过程，称为“学习”。

得到HMM参数之后，即可通过观测序列*O*对隐状态序列的概率做出预报，这个过程称为HMM的解码。Sphinx使用了应用最为广泛的Viterbi算法，该算法是一种动态规划算法，可以在遍历HMM状态网络后得到最优的隐状态序列，即语音识别结果。

## 3.3 Sphinx安装部署

### 3.3.1 依赖软件和PocketSphinx安装

在树莓派Raspbian操作系统中，首先执行以下命令，更新系统：

|  |
| --- |
| sudo apt-get update  sudo apt-get upgrade |

随后，执行以下命令，安装PocketSphinx的依赖软件：

|  |
| --- |
| sudo apt-get install bison  sudo apt-get install libasound2-dev  sudo apt-get install swig  sudo apt-get install python-dev  sudo apt-get install mplayer |

执行下列命令，从sourceforge取得PocketSphinx基础库的源代码：

|  |
| --- |
| wget http://sourceforge.net/projects/cmusphinx/files/sphinxbase/5prealpha/sphinxbase-5prealpha.tar.gz |

执行下列命令，解压源代码，并编译、安装：

|  |
| --- |
| tar -zxvf ./sphinxbase-5prealpha.tar.gz  cd ./sphinxbase-5prealpha  ./configure --enable-fixed  make clean all  make check  sudo make install |

执行下列命令，从sourceforge取得PocketSphinx本体的源代码：

|  |
| --- |
| wget http://sourceforge.net/projects/cmusphinx/files/pocketsphinx/5prealpha/pocketsphinx-5prealpha.tar.gz |

执行下列命令，解压源代码，并编译、安装：

|  |
| --- |
| tar -zxvf pocketsphinx-5prealpha.tar.gz  cd ./pocketsphinx-5prealpha  ./configure  make clean all  make check  sudo make install |

至此，PocketSphinx安装完毕。

### 3.3.2 语言模型生成

在PocketSphinx中，已经内置了英语常用词汇的HMM模型参数，但n-gram语言模型则需要用户上传语料库到http://www.speech.cs.cmu.edu/tools/lmtool-new.html，由服务器生成语言模型文件。

我们准备的语料如下：

|  |
| --- |
| HELLO  OPEN  TEMPERATURE  PRESSURE  CLOSE  GOODBYE |

将这段文本保存并上传，点击“COMPILE KNOWLEDGE BASE”，以生成语言模型。随后，服务器会生成语言模型文件。对于单词识别应用来说，暂时只需要8404.dic和8404.lm两个文件即可。这两个文件即为Sphinx所需要的语言模型文件。

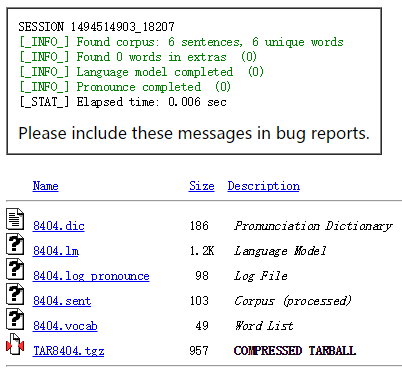


图3-4 生成语言模型文件

### 3.3.3 执行语音识别

首先切换工作目录到用户主目录~，执行以下命令，以设置当前终端窗口中PocketSphinx所需的共享库的路径：

|  |
| --- |
| export LD\_LIBRARY\_PATH=/usr/local/lib  export PKG\_CONFIG\_PATH=/usr/local/lib/pkgconfig |

执行以下命令进行录音:

|  |
| --- |
| arecord -D plughw:0,0 --channels=1 --format=S16\_LE --rate=16000 ~/test.wav |

用来测试的语音内容是“Hello, Temperature, pressure, Goodbye.”。

参数分别指定了输入设备、声道数（1），采样格式（16位有符号数小端模式），采样率（16kHz）以及文件路径。按Ctrl+C结束录音。

随后执行以下命令：

|  |
| --- |
| pocketsphinx\_continuous -hmm /usr/local/share/pocketsphinx/model/en-us/en-us -lm ~/8404.lm -dict ~/8404.dic -samprate 16000/8000/48000 -infile ~/test.wav |

其中，-hmm参数指定了保存在本地的英语HMM模型参数，-lm参数指定了3.3.2中得到的语言模型，-dict参数指定了3.3.2中得到的词库文件，-samprate参数指定语音文件采样率，-infile则指定了输入文件。

由于输出信息过多，我们使用Python脚本提取出识别结果。执行结果如图3-5所示。



图3-5 语音识别结果

## 3.4 本章小结

本章简要介绍了Sphinx语音识别引擎，并详细说明了基于MFCC参数提取和HMM语音识别算法的ASR技术。在此基础上，给出了树莓派Raspbian系统上安装、配置PocketSphinx离线语音识别引擎的操作性步骤，并对其进行了功能测试。测试结果表明，PocketSphinx可以根据自定义的语料库，准确地识别非特定说话人的语音内容。

# 第四章 语音认证

## 4.1 ALIZE工具集介绍与部署

ALIZE是法国Avignon大学开发的一套开源语音处理工具包。该工具包提供一系列底层函数和高级工具，可以完成语音识别、说话人识别等与语音相关的任务，甚至可以直接运用其底层函数完成诸如FFT、DCT等最基本的信号处理工作，避免开发者的重复劳动。

ALIZE由底层函数库和顶层工具集构成（图4-1）。其中，LIA\_SpkSeg用于说话人分类、LIA\_SpkDet用于说话人识别认证、LIA\_Utils则提供了通用的工具集，例如GMM模型训练等。LIA\_SpkTools是上述工具集的API集合。

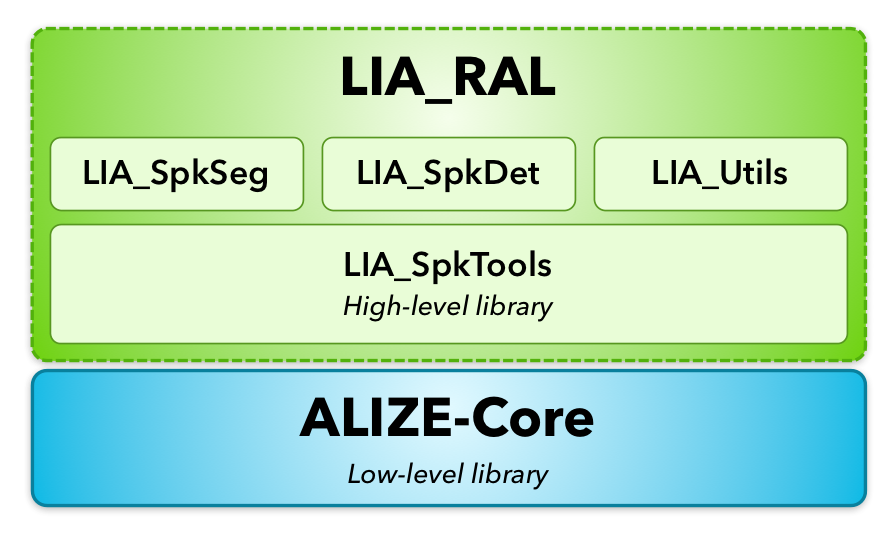


图4-1 ALIZE语音处理工具包的层次结构

ALIZE不能提取语音特征，需要使用其他的特征提取工具。ALIZE支持SPro和HTK两种语音特征提取工具，本系统中选用SPro。

SPro和ALIZE、LIA\_RAL的安装均采用编译安装方式。根据说明文档的指示，我们在树莓派上依次编译并安装SPro、ALIZE和LIA\_RAL，获得语音处理工具如表4-1所示。

表4-1 说话人识别工具集合

|  |  |  |
| --- | --- | --- |
| 工具 | 所属工具集 | 用途说明 |
| sfbcep | SPro | 该工具读取WAVE格式的语音音频文件，并输出每一帧的MFCC特征参数向量到文件。 |
| EnergyDetector | LIA\_SpkDet | 通过能量检测的方式检测静音。 |
| NormFeat | LIA\_SpkDet | 将特征向量按高斯分布标准化。 |
| TrainWorld | LIA\_SpkDet | 利用EM算法，训练得到UBM。 |
| TrainTarget | LIA\_SpkDet | 利用MAP算法，在UBM的基础上训练得到目标说话人的GMM。 |
| ComputeTest | LIA\_SpkDet | 特征相关度测试。 |
| ComputeNorm | LIA\_SpkDet | 对ComputeTest得出的分数进行标准化。 |
| Scoring | LIA\_Utils | 对ComputeNorm打出的分数进行阈值判定。 |

## 4.2 语音认证的基本理论和算法流程

### 4.2.1 基于GMM-UBM的语音认证技术

如图4-2所示，说话人确认的目的是根据已有的说话人语音特征模型，与待判定说话人的语音特征进行对比，进而得到待判定说话人与已有的说话人是否匹配。

待判定语音

目标说话人的语音

待判定语音的MFCC特征

目标说话人的MFCC特征

目标说话人的

语音特征模型

相似度计算

图4-2 说话人确认的一般流程

匹配度计算

GMM-UBM模型训练

EM

MAP

待判定语音

目标说话人的语音

待判定语音的MFCC特征

目标说话人的MFCC特征

目标说话人的

GMM模型

相似度计算

非目标说话人的语音

非目标说话人的MFCC特征

UBM模型

图4-3 使用GMM-UBM方法的说话人确认算法

语音文件采用WAVE文件格式，该格式可直接用于SPro进行MFCC特征提取。关于MFCC语音特征参数，已经在第三章介绍过，因此下文重点介绍GMM-UBM语音模型。图4-3描述了采用GMM-UBM模型的说话人确认的流程。

同语音识别相似，说话人确认的本质是模式匹配的过程。对于说话人特征的建模方法，主要有模板匹配、概率统计模型、人工神经网络等方法。上文介绍的HMM属于概率统计模型方法。由于HMM为双随机过程，因此对于语音内容识别效果较好，但对于说话人特征的描述则易受噪声影响[27]。

GMM（Gaussian Mixture Model，高斯混合模型）是若干个高斯分布的叠加，它们的线性组合可以模拟说话人矢量特征的连续概率分布，即语音的说话人特性。由于任何概率分布都可以由高斯分布的线性叠加进行描述，因此GMM可以描述一段连续语音的说话人特征[27]。

在具备说话人语音样本的条件下，通过对模型进行最大期望（Expectation Maximization，EM）迭代，可以推知满足已知样本的GMM参数，这个过程就是对GMM参数的训练，使用的算法称为EM算法。

在实际的说话人确认场合下，由于受到环境噪声、乃至说话人本身身体状态（例如感冒等）的影响，单纯使用目标说话人的GMM进行匹配的效果并不理想。为解决该问题，引入了UBM（Universal Background Model，通用背景模型）。UBM本质上也是GMM，但训练GMM的样本不仅包含目标说话人的语音，也包含其他说话人（imposter）的语音以及话筒、环境噪音的因素。UBM仅需使用EM算法，采用大量训练样本训练一次即可。

得到UBM后，即可通过MAP（Maximum A Posteriori，最大后验估计）算法，利用已有的目标说话人特征数据，对UBM进行自适应，得到效果更好的目标说话人GMM。

得到目标说话人GMM之后，即可将待判定说话人的特征参数代入GMM进行匹配，得到得分后，依据具体问题的情况确定的判定准则进行通过或者不通过的判定。在本系统中，我们采用下述方法进行通过判定。

### 4.2.2 分类判定规则

据4.2.1节的讨论，待测语音与目标说话人的匹配度是以得分形式给出。因此，为得到可靠的判定结果，需要选择恰当的分类门限值。

实际运行中，使用同一目标说话人的不同内容的语音训练得到的GMM，分别对待测语音进行打分，二者取算术平均，作为匹配度打分。

确认通过原则如下：

- 计算两个匹配得分，计算算术平均；

- 若算术平均高于21.0，则认证通过，否则不通过。

该规则基于我们的统计结果得到，详细论述将在第六章中展开。

## 4.3 GMM-UBM模型训练与结果决策

由于树莓派性能有限，且GMM模型训练仅需要一次，因此在Ubuntu 12.04操作系统的PC机上先训练出UBM模型和目标说话人GMM模型。

### 4.3.1 MFCC参数提取

这一步主要是应用了SPro提供的sfbcep工具，对所有的训练用语音和待测试语音做特征提取。语音文件列表放在工作目录下的data/data.lst文件中，编写bash脚本，遍历所有语音文件并进行特征提取，脚本如下：

|  |
| --- |
| for i in `cat data/data.lst`;do  sfbcep -l 20 -d 10 -w Hamming -p 16 -e -D -k 0 -i 300 -u 3400 -F wave data/wave/$i.wav data/prm/$i.tmp.prm  done |

各参数的意义如下：

-l 20 窗宽20ms

-d 10 窗位移10ms

-w Hamming 选用汉明窗函数对语音帧加窗

-p 16 输出16个倒谱系数

-e 采用对数能量

-D 输出一阶微分特征参数

-k 0 预加重系数0

-i 300 频率下限0.3kHz

-u 3400 频率上限3.4kHz

-F wave 输入语音编码格式为WAVE PCM

输出的特征向量以prm文件形式保存，供后续步骤使用。

### 4.3.2 静音去除与特征参数标准化

静音去除的目的是为了去掉特征参数中反映静音帧的部分，避免对模型训练等后续步骤造成干扰。为去除静音，首先利用NormFeat工具对特征参数按标准高斯分布（即正态分布）N(0,1)进行标准化，然后利用EnergyDetector工具检测静音帧，并对语音帧做标记。

经过静音检测的特征参数，需要再次按N(0,1)分布进行标准化，供UBM和目标GMM训练。这一步骤也是利用了NormFeat工具。这个过程写成bash脚本如下：

|  |
| --- |
| NormFeat --config cfg/NormFeat\_energy\_SPro.cfg  EnergyDetector --config cfg/EnergyDetector\_SPro.cfg  NormFeat --config cfg/NormFeat\_SPro.cfg |

命令行的config参数指定了工具所需的各种参数，包括输入输出文件的位置等。静音去除的输出是标签文件，参数标准化输出的是扩展名为.norm.prm的特征文件。

### 4.3.3 UBM-GMM训练和判定分数标准化

对所有训练样本提取参数后，利用它们训练一次UBM。训练UBM的工具是TrainWorld，包括输入输出的参数储存在cfg文件中。训练得到的UBM模型存储在./gmm/world.gmm文件中。命令行如下：

|  |
| --- |
| TrainWorld --config cfg/TrainWorld.cfg |

随后针对目标说话人的语音，利用TrainTarget工具对UBM做自适应，得到目标GMM，命令行如下：

|  |
| --- |
| TrainTarget --config cfg/TrainTarget.cfg |

随后计算初步的判定分数，使用ComputeTest工具：

|  |
| --- |
| ComputeTest --config cfg/ComputeTest\_GMM.cfg |

为了提高评分的稳定性，需要使用其它非目标说话人（imposter）的语音特征去代入已有的目标说话人GMM，通过得到的得分对判定分数进行标准化，这个过程使用了ComputeTest和ComputeNorm工具，imposter语音来自TIMIT语音语料库。最终，使用Scoring工具，即可得到最终判定结果。

|  |
| --- |
| ComputeTest --config cfg/ComputeTestZNorm.cfg > log/ComputeTest\_znorm.log  ComputeNorm --config cfg/ComputeNorm\_znorm.cfg > log/ComputeNorm\_znorm.log  Scoring --config cfg/Scoring.cfg > log/Scoring.log |

输出结果decision.res格式如下。第五列代表被匹配的GMM说话人（spk01），第六列代表待匹配的语音样本。事实上，所用的这些语音样本都是出自spk01（即作者本人），因此判定结果分数都比较高。

|  |
| --- |
| 1side u 1side m spk01 01-01 t 57.7978  1side u 1side m spk01 01-02 t 8.33473  1side u 1side m spk01 01-03 t 10.7941  1side u 1side m spk01 test1 t 12.9316  1side u 1side m spk01 test2 t 20.1072  1side u 1side m spk01 test3 t 18.936  1side u 1side m spk01 test4 t 22.9485  1side u 1side m spk01 test5 t 29.0714  1side u 1side m spk01 test6 t 14.1478 |

## 4.4 本章小结

本章从介绍ALIZE语音识别工具包开始，说明了基于GMM-UBM的说话人确认算法。鉴于传统的GMM在说话人识别领域存在不稳定性，因此引入了能够描述话筒和环境噪音因素的UBM对目标说话人的GMM进行自适应调谐。为了使得到的评分结果更加稳定，流程中引入了非目标说话人作为负样本，并依据非目标说话人的语音特征对评分结果进行T-标准化，进而通过确认规则，得到最终的说话人确认结果。最后，依据此算法的流程，描述了运用ALIZE工具集对语音特征模型的训练和结果决策流程。实验证明，该算法可在可接受的时间范围内，给出较好的说话人确认结果。

# 第五章 MQTT协议及其实现

## 5.1 MQTT协议介绍

在IoT网络中，存在大量功耗敏感、计算能力较弱的终端设备，这些设备是IoT网络的神经末梢，承担着收集数据、传输数据的任务。但是，传统的HTTP等应用层协议是为PC、移动智能终端等功耗和计算能力都比较强大的设备设计的，并不适合IoT终端设备之间的通信。MQTT等轻量级的应用层协议从一开始就是为IoT设计的，该协议将运用在IoT设备间的通信，提供一种更加适合IoT应用场景的应用层协议解决方案。

MQTT（Message Queue Telemetry Transport，消息队列遥测传输）协议由IBM开发，是一款轻量级的、基于代理的、发布/订阅式消息传输协议。这种特征使得它特别适用于资源有限的应用环境，例如高通信成本、低通信带宽、有限的存储和计算能力等等。MQTT协议的具体特性有：

- 采用发布/订阅的消息传输模式；

- 基于TCP/IP；

- 提供三种QoS选项；

- 通信开销非常小，固定头部仅需2字节；

- 发生连接中断时，可通过Last Will 和 Testament 机制通知有关各方。

下文描述的是MQTT3.1.1版本。

MQTT协议同样是客户端-服务器通信，但作为发布/订阅式的通信协议，通信参与方有发布者（Publisher）、订阅者（Subscriber）和代理（Broker）三方。

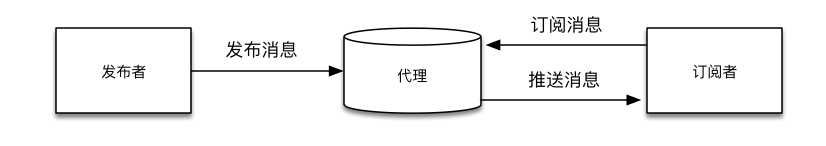


图5-1 MQTT协议中的角色

MQTT客户端可扮演Publisher或者Subscriber的角色，相应地，作为Publisher，客户端可以向Broker发布带有某个主题（Topic）的消息，作为Subscriber的客户端则可以向Broker订阅或者退订某个Topic的消息。而MQTT服务器作为Broker，则可以接收Subscriber的订阅或者退订请求，或者接收Publisher发布的带有某个Topic的消息，再向订阅该主题的Subscriber推送消息。

MQTT报文可分为报头（Header）和负载（Payload）两部分。协议头分为固定报头（Fixed header）和可变报头（Variable header）两部分。

固定报头由2字节组成，其字段划分由表5-1给出。第一字节的7-4位是报文类型字段，表5-2给出了全部报文类型。需要说明的是，由于只有7-4位是报文类型字段，所以表5-2每种类型的编号都左移4位。第一字节的3-0位是报文标识位，其中PUBLISH型报文的标识位字段分配已在表5-1中体现。第二字节用来保存可变报头和负载的总长度。

表5-1 MQTT固定报头（PUBLISH型报文）

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **位** | **7** | **6** | **5** | **4** | **3** | | **2** | **1** | **0** |
| 字节1 | 报文类型 | | | | | DUP标志位 | QoS等级 | | RETAIN位 |
| 字节2 | 报文剩余长度 | | | | | | | | |

表5-2 MQTT报文类型

|  |
| --- |
| #define MQTTCONNECT 1 << 4 // Client连接请求  #define MQTTCONNACK 2 << 4 // Server连接应答  #define MQTTPUBLISH 3 << 4 // 发布消息  #define MQTTPUBACK 4 << 4 // 发布应答  #define MQTTPUBREC 5 << 4 // 发布收到  #define MQTTPUBREL 6 << 4 // 发布释放  #define MQTTPUBCOMP 7 << 4 // 发布完成  #define MQTTSUBSCRIBE 8 << 4 // 订阅请求  #define MQTTSUBACK 9 << 4 // 订阅应答  #define MQTTUNSUBSCRIBE 10 << 4 // 退订请求  #define MQTTUNSUBACK 11 << 4 // 退订应答  #define MQTTPINGREQ 12 << 4 // PING请求  #define MQTTPINGRESP 13 << 4 // PING响应  #define MQTTDISCONNECT 14 << 4 // Client连接断开  #define MQTTReserved 15 << 4 // 保留 |

可变报头和负载根据报文类型的不同，内容也不同。具体内容在参考文献[28]中已有详细说明，也是后续在ESP8266上实现MQTT的依据。

## 5.2 MQTT协议的实现

5.1节中简要描述了MQTT的内容。由于ESP8266提供了完整的TCP/IP协议栈，因此可以在ESP8266平台上实现MQTT协议。在本系统中，IoT网关和IoT终端之间的通信，也是基于TCP/IP的，由于MQTT协议对IoT应用领域的优势，我们选择MQTT作为应用层协议。运行在树莓派上的IoT网关，既是MQTT Broker，也是Subscriber和Publisher，作为IoT终端的ESP8266也同时是Subscriber和Publisher。这样即可实现网关与终端之间的双向通信。

### 5.2.1 树莓派MQTT服务器的实现

Mosquitto是一款开源的MQTT服务器软件。编译、安装后，执行

|  |
| --- |
| mosquitto |

即可启动树莓派上的MQTT Broker。

然后执行以下命令，订阅主题为“sensor”的MQTT消息。此时树莓派也作为MQTT Subscriber。命令中使用了输出流重定向，每当Subscriber接收到Broker推送来的消息，都会将消息输出到esplog.log文件中，供主流程适时读取。

|  |
| --- |
| mosquitto\_sub -v -t sensor > /home/pi/esplog.log |

### 5.2.2 ESP8266的MQTT客户端实现

按照自顶向下的设计思路，首先确定程序框架。依据2.3中描述的工作流程，ESP8266应该既作为Subscriber，又作为Publisher。作为Publisher，ESP8266应当以5秒的间隔，读取BMP280传感器，并向Broker发布主题为sensor、内容为气温气压数据的MQTT报文。同时，作为Subscriber，ESP8266应当以中断方式接收Broker发来的消息推送，并根据报文内容对GPIO进行操作。

Arduino IDE已经将ESP8266的TCP/IP客户端相关的操作封装成WiFiClient类，该类为Arduino抽象类Client的派生类，其提供的部分接口如下：

|  |
| --- |
| class Client: public Stream {  public:  virtual int connect(const char \*host, uint16\_t port) =0;  virtual size\_t write(const uint8\_t \*buf, size\_t size) =0;  virtual int available() = 0;  virtual int read(uint8\_t \*buf, size\_t size) = 0;  virtual int peek() = 0;  virtual void flush() = 0;  virtual void stop() = 0;  virtual uint8\_t connected() = 0;  /\*...\*/  }; |

WiFiClient也实现了这些标准接口。利用这些接口，即可按照MQTT协议[28]的规定，构造MQTT报文，并调用WiFiClient提供的接口，发送TCP报文。

Arduino IDE已经提供了封装好的WiFiClient及其实现，在此基础上，通过MqttClient类来实现MQTT应用层协议。MqttClient提供如下接口：

|  |
| --- |
| #define MQTT\_CALLBACK void (\*callback)(char\*, uint8\_t\*, unsigned int)  class MqttClient {  private:  Client\* tcp\_client;  public:  MqttClient(Client& client);  MqttClient& setServer(uint8\_t \* ip, uint16\_t port);  MqttClient& setCallback(MQTT\_CALLBACK);  MqttClient& setClient(Client& client);  boolean connect(const char\* id, const char\* user, const char\* pass, const char\* willTopic, uint8\_t willQos, boolean willRetain, const char\* willMessage);  boolean connected();  void disconnect();  boolean publish(const char\* topic, const char\* payload);  boolean subscribe(const char\* topic, uint8\_t qos);  boolean unsubscribe(const char\* topic);  boolean loop();  }; |

代码省略了部分私有成员和私有函数。每个接口的功能通过函数名就可以一目了然。其中，loop()用来监听推送，并发送PING报文（心跳报文，用于保持唤醒），而setCallback()则是注册回调函数。私有成员tcp\_client由构造函数指定，该Client提供了传输层协议的接口，供其它成员函数使用。

接口函数中，connect、publish、subscribe和unsubscribe均需要向对端服务器发送MQTT报文。每个函数都是先按照MQTT协议[28]的规定，依据传入的实际参数构造报文，保存到字符buffer中，然后通过Client tcp\_client提供的传输层接口，将buffer中的MQTT报文发送出去。这个过程非常清楚地展示了传输层和应用层之间的层次关系。

据此，可写出程序主体的Arduino C++代码（摘要）：

|  |
| --- |
| const char\* mqtt\_server = "192.168.0.3";  WiFiClient espClient;  MqttClient client(espClient);  char msg[50];  void callback(char\* topic, byte\* payload, unsigned int length) {  if (payload == "ON") { // pseudo code  digitalWrite(BUILTIN\_LED, LOW);  }  else if (payload == "OFF"){  digitalWrite(BUILTIN\_LED, HIGH);  }  else {return;}  }  void setup() {  pinMode(BUILTIN\_LED, OUTPUT);  setup\_wifi();  client.setServer(mqtt\_server, 1883);  client.setCallback(callback);  }  void loop() {  if (!client.connected()) {  reconnect();  client.subscribe("gpio");  }  client.loop();  long now = millis();  if (now - lastMsg > 2000) {  lastMsg = now;  BMP280\_ReadTempBaro(msg);//Pseudo code  client.publish("sensor", msg);  }  } |

其中，回调函数callback在收到推送后执行，作用是根据报文内容修改GPIO；BMP280\_ReadTempBaro()是通过IIC总线读取BMP280的接口函数，其技术细节已在2.4.1中说明。

## 5.3 MQTT消息传输测试

按照第二章的内容搭建局域网，使用Arduino IDE将MQTT客户端代码烧写入ESP8266，随后上电。按照5.2.1的说明，在树莓派上运行MQTT Broker和Subscriber，订阅主题为sensor的消息。由于ESP8266每隔5秒发布一次消息，因此，树莓派每5秒收到一条推送消息（图5-2）。

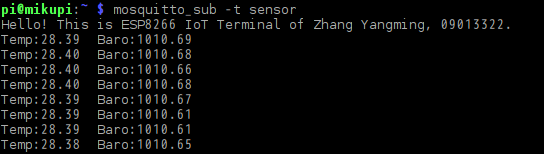


图5-2 Subscriber接收到的消息推送

在树莓派的终端中执行如下命令，向Broker发布主题为gpio、负载内容为ON的消息：

|  |
| --- |
| mosquitto\_pub -t gpio -m ON |

由于ESP8266已订阅该主题的消息，因此可以看到NodeMCU开发板上的内建LED被点亮（图5-3）。再次执行

|  |
| --- |
| mosquitto\_pub -t gpio -m OFF |

可见LED熄灭。说明MQTT通信可用。



图5-3 内建LED被点亮

## 5.4 本章小结

本章介绍了MQTT协议的特点、机制、报文格式和通信行为，并且在树莓派和ESP8266平台上分别实现了MQTT服务器和客户端。依据协议规范，并结合系统的功能要求，在ESP8266模块上使用C++实现了基于TCP传输层接口的MQTT应用层协议接口，并完成了IoT终端的全部功能实现。最终通过实际编程、组网、运行，验证了MQTT通信的可用性。

# 第六章 系统实现与测试

## 6.1 流程控制程序

本文2.3节描述了系统的总体流程，该流程涉及多个程序。处理这种问题，一般选用脚本语言充当“胶水语言”。Python是一门强大、应用广泛的脚本语言，也是Linux各个发行版内置的脚本语言。Python提供了一套功能齐全而强大的库，对于树莓派，Python提供了控制GPIO的库。运用此库，即可读取按键状态，并点亮LED。

硬件上，我们设置了一个按钮按键和两个不同颜色的LED（两个不同颜色的LED被封装在同一个管子里），作为状态指示。按钮按键的一端连接在树莓派的GPIO18上，另一端接地；LED的正极则分别连接到树莓派的GPIO24和GPIO25上，负极相互连接，共同接地。由于按钮按下相当于GPIO18接地，电位为0，因此GPIO18不允许浮空，需要接上拉电阻。另外，由于树莓派的驱动能力有限，而LED导通电流高达几十mA，因此必须在两个LED上分别串联限流电阻，以保护树莓派。（图6-1）

树莓派的主控芯片内部已经提供了上拉电阻，因此只需要软件设置GPIO18为上拉即可；另外，为节省器件数量，我们使用了集成有限流电阻的LED模块，方便了硬件的搭建。

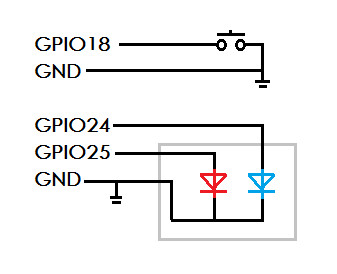


图6-1 按键和指示灯与树莓派的连接方式

软件上，首先需要设置GPIO状态：

|  |
| --- |
| button = 18  lights = [24, 25]  #GPIO设置  GPIO.setwarnings(False)  GPIO.cleanup()  GPIO.setmode(GPIO.BCM)  GPIO.setup(button, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)  GPIO.setup(lights, GPIO.OUT)  GPIO.output(lights, GPIO.LOW) |

为获取按键状态，我们采取软件轮询的方式，反复读取GPIO18的值。这部分的简要Python代码描述如下：

|  |
| --- |
| def start():  last = GPIO.input(button)  flag = 0  while True:  val = GPIO.input(button)  #若检测到按键边沿，则开始识别  if val != last:  last = val  if val == 0:  #第一次按下：开始录音  if flag == 0:  flag = 1  #录音  #第二次按下：结束录音  elif flag == 1:  flag = 0  #结束录音并开始语音识别 |

作为胶水语言的Python，需要使用commands.getoutput()、popen()等方法执行外部命令，并输出其结果，例如：

|  |
| --- |
| cmdline = "arecord -D plughw:0,0 --channels=1 --format=S16\_LE --rate=16000 /home/pi/srtemp/temp.wav > /dev/null"  subprocess.Popen(cmdline, shell=True) |

由此，便编写出控制整个流程的Python脚本程序，可直接在命令行环境执行。

## 6.2 功能测试

按照前文的说明搭建完整的系统，并开机运行。如图6-2所示，为开发、调试方便，我们将一台PC机接入局域网内，用来通过SSH连接到树莓派、执行命令行。

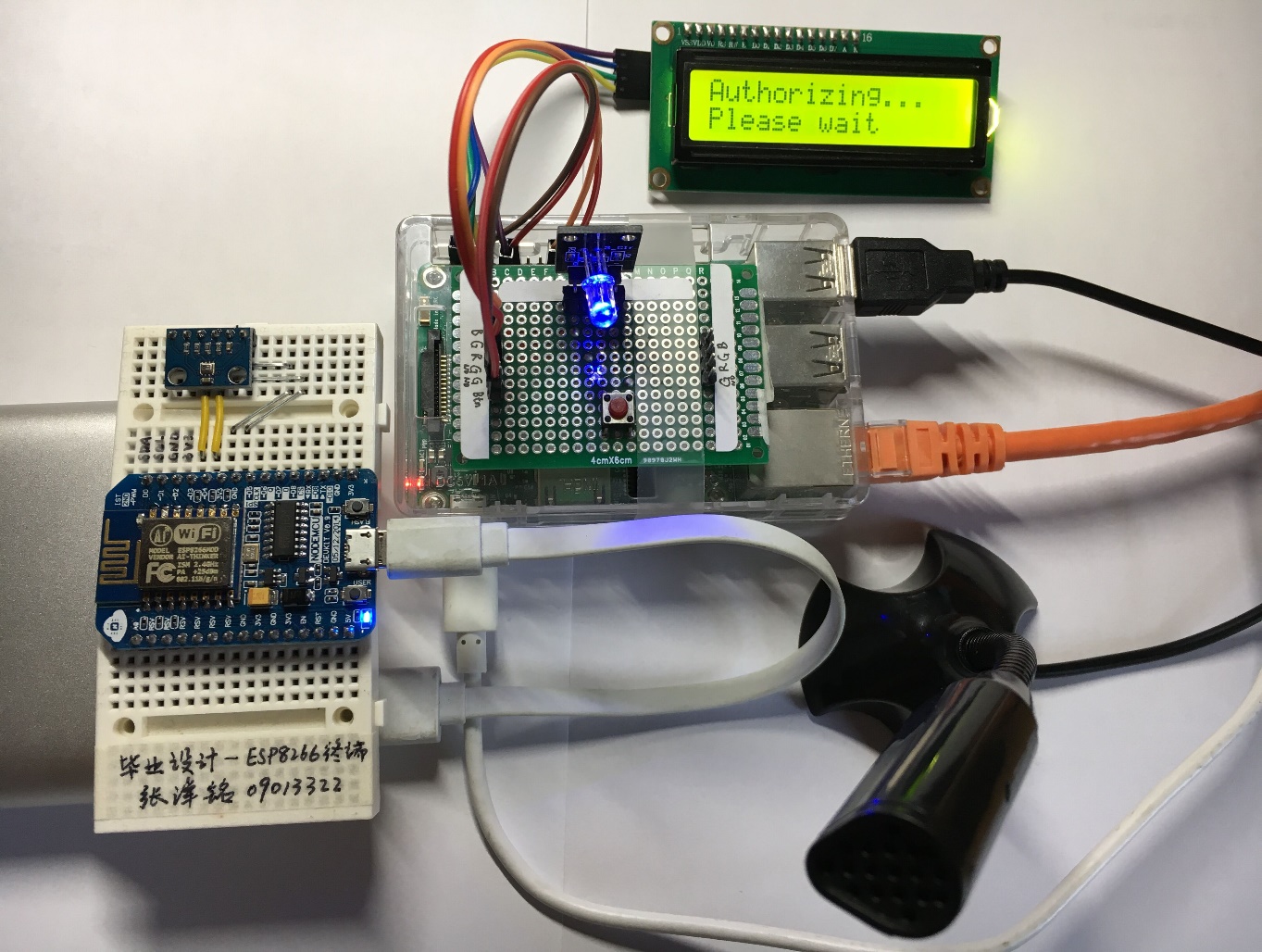


图6-2 完整系统

将树莓派电源接通，由于采用静态IP，在PC机上可通过192.168.0.3访问树莓派。测试时，我们通过PuTTY终端登录树莓派，通过命令行对其进行控制。为了正常使用Sphinx，需要在当前终端设置共享库目录。（图6-3）

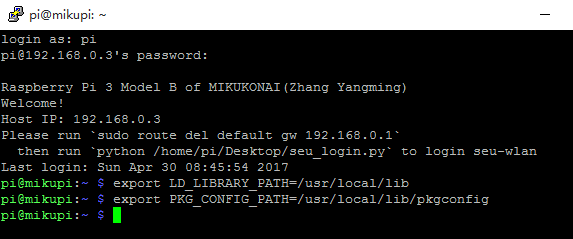


图6-3 从PC登录树莓派，并设置共享库目录

在运行主流程之前，先开启树莓派上的MQTT Broker。随后，ESP8266模块开机，自动连接无线网络并连接到树莓派Broker，终端上会显示如下信息，如图6-4。

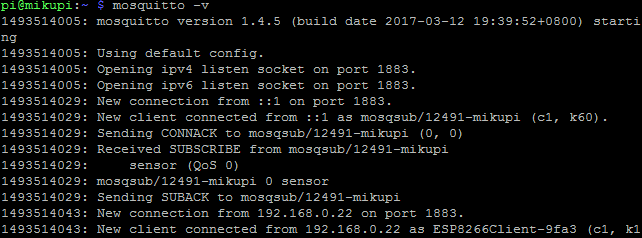


图6-4 ESP8266连接到Broker

运行树莓派MQTT Subscriber，打开/home/pi/esplog.log文件，可以看到间隔5秒的气温气压信息。

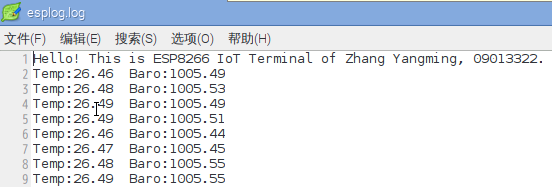


图6-5 ESP8266推送到树莓派的气温气压消息

此时，开始运行主流程，首先进行语音认证。按住按钮，口述认证语音，说完松手。若认证通过，即可开始语音控制，如图6-5。

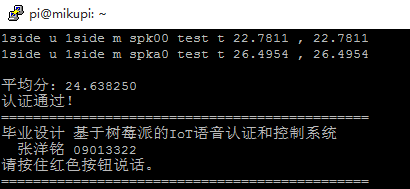


图6-5认证通过的效果

若说话人认证不通过，或者一段时间内没有执行任何语音控制，则需要重新认证，如图6-6。

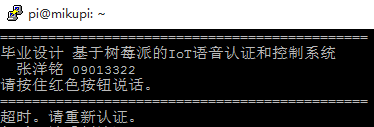


图6-6 语音认证不通过

认证通过后，即可进行语音识别。例如使用语音“Temperature”进行测试，测试结果如图6-7。



图6-7 语音识别结果

结果说明，系统整体是可用的。

## 6.3 语音识别和认证性能测试

为测试语音识别性能，我们进行了数百次测试。在系统所用的含有6个英文单词的封闭的词汇集的条件下，识别正确率可达到100%，完全满足需要。下文重点论述对语音认证功能的性能测试。

根据第四章的论述，基于GMM-UBM的说话人确认算法不仅需要目标说话人的语音作为训练数据，也需要一定量的imposter语音作为校正最终得分的依据。目标说话人设定为本文作者，录制了时长约40min的语音数据；imposter语音则选用TIMIT语音库中40个男性语音片段和40个女性语音片段。UBM由目标说话人和TIMIT语音共同训练得到。由于UBM和GMM训练速度耗时较长，因此我们在Ubuntu 12.04的PC机环境下训练UBM和GMM，并将其复制到树莓派。

按照第四章描述的方法，评分时，将待评分语音特征参数分别代入目标说话人GMM中，结合80个imposter提供的分数校正信息，计算出匹配度评分。为衡量系统性能，我们进行了多次测试，以统计系统的决策准确性。

我们进行了20次实验，其中10次是目标说话人测试（记为pos测试），10次是非目标说话人测试（记为neg测试）。我们统计了各次实验的匹配分数。

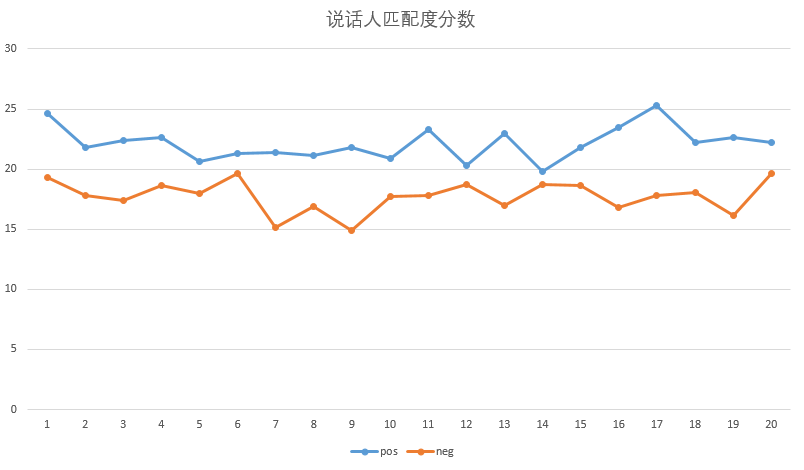


图6-8 匹配度统计

根据图6-8，pos测试中，系统给出的分数普遍高于neg测试，这与我们的预期是一致的。

任何语音认证系统都会涉及到决策准确性的评价。为评价一个说话人确认系统，记“说话人是目标说话人”为P，记“说话人不是目标说话人”为N，记“系统判断说话人是目标说话人”为p，记“系统判断说话人不是目标说话人”为n，则有以下四个参数值得关注：

True Positive：*TP* = *P*(p|P)

True Negative：*TN* = *P*(n|N)

False Positive：*FP* = *P*(p|N)

False Negative：*FN* = *P*(n|P)

对于同一待测样本，系统必定给出p或者n，非此即彼，即有以下关系存在：

*TP*+*FN*=1

*TN*+*FP*=1

显然，一个好的语音认证系统应当具备较高的*TP*、*TN*和较低的*FP*（假正，误报率）、*FN*（假负，漏报率）。但是，在实际系统中，*FP*和*FN*是此消彼长的关系，不可能同时做到较低水平[27]。因此，为确定系统的判决门限，可借助DET（Detection Error Tradeoff，检测误差权衡）曲线[34]寻找合适的门限值，如图6-9。

*FN*(漏报)

*FP*(误报)

错误率

门限值

*a b c*

图6-9 DET曲线

较高的*FP*意味着有较大概率将原本不是目标说话人的语音判决为目标说话人，而这对于语音认证来说是不可容忍的错误。但较高的*FN*意味着将目标说话人的语音判决为非目标说话人，尽管也属于错误，但这种错误的代价显然要小得多。图6-9中，把*FP*=*FN*的判决门限称为EER（等错误率）点，为首要地降低*FP*，并兼顾*FN*水平，应当选择EER点偏右的点，即图6-9中的c点附近，作为判决门限值。因此，在图6-8中，根据统计结果，我们选择21.0分作为门限值。该门限值可保证较低的*FP*和相对不高的*FN*，实现性能的最优化。

# 第七章 总结与展望

## 7.1 论文总结

本文论述了在树莓派平台上构建IoT语音认证和控制系统的理论、方法和流程。系统综合运用多种软硬件技术和语音处理算法，实现了语音认证、语音控制、IoT局域网内的MQTT通信、以及IoT终端的读写等功能。

本文描述的系统旨在解决第一章中提出的四个问题。本系统采用语音交互，是一种对用户非常友好的交互方式。本系统采用性价比高、易于开发的树莓派和ESP8266平台，功能可扩展性强。本系统中IoT设备之间通信的应用层协议采用在IoT领域应用广泛的MQTT协议，方便与其他类型的IoT设备兼容和对接。本系统引入了生物特征认证方式之一的语音认证，提高了整个IoT网络的安全性。

经过具体的硬件搭建、环境配置、工具部署和软件编程，最终实现了一个实际可用的完整系统，从而验证了文中提出的技术路线和实现方案。

## 7.2 工作展望

本文实现的系统尽管完整可用，但作为一个原理验证原型，距离实用化尚有一段距离。在未来的设计研究工作中，可以从以下几个方面来完善本系统的设计：

1. 进一步提高语音认证、语音识别的精度和速度。语音识别领域并非只有HMM这一种模型，说话人识别也不只有GMM-UBM一种模型。未来的工作中，可以对比多种语音模型，通过比较研究，选择最适合家用环境的语音算法。

2. 进一步完善语音交互方式。在提高语音识别、语音认证的精度和速度的基础上，可以取消本系统中设计的硬件录音按钮，将语音识别设置成“时刻在线”的。为实现该功能，需要对系统运行流程和具体算法做进一步设计，以有效区分语音命令和日常讲话的区别，避免误动作。

3. 选择更加强大、性价比更高的软硬件平台。例如麦克风阵列等硬件，可以显著提高语音处理的效果和性能。

4. 综合包括说话人识别在内的多种生物识别手段，进一步提高系统的访问控制能力，以进一步提高安全性

参考文献

[1] Noah Apthorpe, Dillon Reisman, Nick Feamster. A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic. New York, NY: Workshop on Data and Algorithmic Transparency (DAT). November 2016.

[2] Warren W. Gay. Raspberry Pi Hardware Reference. Apress. 2014.

[3] Warren W. Gay. Raspberry Pi System Software Reference. Apress. 2014.

[4] Eben Upton.树莓派用户指南（第三版）.人民邮电出版社.2016.

[5] 殷贤华, 刘明缘, 王宁. 基于Raspberry Pi的智能家居系统设计. 现代电子技术. 2016, 23: 161-164.

[6] Steven Goodwin. Smart Home Automation with Linux and Raspberry Pi. Second edition. [Berkeley, CA] : Apress. 2013.

[7] Amazon Echo. Wikipedia. https://en.wikipedia.org/wiki/Amazon\_Echo

[8] HomeKit. Apple Developer Website. https://developer.apple.com/homekit/

[9] Tanay Pant. Building a Virtual Assistant for Raspberry Pi. Apress, 2016.

[10] Rabiner L R. A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition[J]. Proceedings of the IEEE, 1990, 77(2):267-296.

[11] Lee, Kai Fu, and R. Reddy. Automatic Speech Recognition: The Development of the Sphinx Recognition System. Kluwer Academic Publishers, 1989.

[12] ALIZE website. http://alize.univ-avignon.fr

[13] 科大讯飞语音识别引擎. http://www.iflytek.com/audioengine/list\_3.html

[14] 百度语音识别. http://yuyin.baidu.com/asr

[15] Reynolds D A, Rose R C. Robust text-independent speaker identification using Gaussian mixture speaker models[J]. IEEE Transactions on Speech & Audio Processing, 1995, 3(1):72-83.

[16] 金银燕, 于凤芹, 何艳. 基于时频分布与MFCC的说话人识别. 计算机系统应用, 2012, 04: 189-192.

[17] 张晶, 范明, 冯文全, 董金明. 基于MFCC参数的说话人特征提取算法的改进. 电声技术, 2009, 9: 61-64.

[18] Bonastre J.F., Wils F., Meignier S.. ALIZE, A FREE TOOLKIT FOR SPEAKER RECOGNITION. Avignon, France: University of Avignon (LIA) Interspeech'05.

[19] Muhammad Harith Amaran, Nazmin Arif Mohd Noh, Mohd Saufy Rohmad, etc.. Comparison of Lightweight Communication Protocols in Robotic Applications. In 2015 IEEE International Symposium on Robotics and Intelligent Sensors (IEEE IRIS2015), Procedia Computer Science, 2015, 76:400-405.

[20] Hyun Cheon Hwang, JiSu Park, Jin Gon Shon. Design and Implementation of a Reliable Message Transmission System Based on MQTT Protocol in IoT Wireless Personal Communications. Wireless Pers Commun, December 2016, Volume 91, Issue 4, pp 1765–1777.

[21] 马跃, 孙翱, 贾军营, 孙建伟, 于碧辉, 杨雪华. MQTT协议在移动互联网即时通信中的应用. 计算机系统应用, 2016, 03: 170-176.

[22] Bousquet P.M., Matrouf D., Bonastre J.F.. Intersession compensation and scoring methods in the I-vectors space for speaker recognition. Avignon, France: University of Avignon (LIA) Interspeech'11.

[23] 陈士凯, 程晨, 臧海波. intel Edison智能硬件开发指南:基于Yocto Project[M]. 北京:人民邮电出版社, 2015.

[24] PCF8591 Product specification. Philips Electronics N.V., 1998.

[25] BMP280 Data Sheet. Bosch Sensortec GmbH, 2015.

[26] Zhen B, Wu X, Liu Z, et al. On the importance of components of the MFCC in speech and speaker recognition[C]. International Conference on Spoken Language Processing, Icslp 2000 / INTERSPEECH 2000, Beijing, China, October. DBLP, 2000:487-490.

[27] 李欢. 基于GMM-UBM模型的说话人识别系统[D]. 呼和浩特：内蒙古大学, 2016.

[28] Protocol Specifications of MQTT. OASIS Open. 2014.

[29] Tiwari V. MFCC and its applications in speaker recognition[J]. International Journal on Emerging Technologies Issn, 2010.

[30] Thaker T. ESP8266 based implementation of wireless sensor network with Linux based web-server[C]. Colossal Data Analysis and NETWORKING. IEEE, 2016.

[31] Reynolds D A, Quatieri T F, Dunn R B. Speaker Verification Using Adapted Gaussian Mixture Models[J]. Digital Signal Processing. 2000, 10(1-3):19-41.

[32] 姜妮, 张宇, 赵志军. 基于MQTT物联网消息推送系统[J]. 网络新媒体技术, 2014, 3(6):62-64.

[33] 曾霞霞, 徐戈, 吴征远. 基于MFCC特征组合参数的说话人识别研究[J]. 集美大学学报(自然版), 2016, 21(4):317-320.

[34] Martin A F, Doddington G R, Kamm T, et al. The DET curve in assessment of decision task performance[C]. Rhodes, Greece: European Conference on Speech Communication and Technology, Eurospeech 1997, September. 1997:1895--1898.

[35] Guillaume Gravier. SPro Documentation. 2010.

[36] Raspberry Pi. Wikipedia. https://en.wikipedia.org/wiki/raspberry\_pi

[37] Raspberry Pi Schematics R1.0.

[38] NodeMCU on GitHub. https://github.com/nodemcu/nodemcu-devkit-v1.0

**致　　谢**

百载文枢江左，东南辈出英豪。在东南大学求学的四年间，我学到了扎实的专业知识和技能，领略了各位老师的风采和学识，结识了志同道合的同学和朋友，收获了美好的回忆。四年级的毕业设计是本科阶段最后一项任务，也是对四年所学的综合考验。值此毕业设计完成之际，我想对在毕业设计期间关心和帮助我的各位老师、亲人和好友致以由衷的谢意。

在这里，首先要感谢毕设指导教师凌振老师的悉心指导。凌老师是一位富有创意和热情的老师，选择了时代和技术前沿的物联网相关的毕设课题。在毕业设计进行期间，凌老师的悉心指导和帮助使我受益匪浅。从报告论文的字斟句酌到研究方法的有力指导，从疑难问题的精妙点拨到领域前沿的介绍和引导，都使我受益匪浅，不仅为我的毕业设计提供了有力的指导帮助，更让我领略了凌老师的丰富学识和人格魅力。

我还要感谢我的父母家人，离家求学的我离不开父母的支持和关心。

感谢东南大学四年来对我的培养，“止于至善”的理念已经深入我的内心。

感谢所有参考文献和资料的作者，感谢各位先贤和前辈为物联网技术的发展做出的杰出贡献。

再次感谢凌振老师的对我的耐心指导和帮助。