# 摘要

目前说话人识别研究主要是基于专业录音设备和处理机，来提取用户的语音中的特征并训练模型和识别确认处理。主要适用于大型安全领域。整套解决方案的可移动性不强，且不利于大规模普及和部署。因此本课题旨在研究开发出一个面向智能手机的说话人识别系统，在智能手机上获取用户语音，并在手机上提取特征并训练模型和识别确认处理。从而实现说话人识别方案的大规模普及和快速部署。本课题开发的说话人识别系统主要面向Android平台手机用户。

关键字：智能手机，说话人识别，特征提取，训练模型，Android

# Abstract

Speaker verificationresearch at present is mainly based on theprofessional recording equipment and processor, to extract the features in the user's voice and training model and processing verification.It is mainly suitable for large areas of security.The mobility of the whole solution is not strong, and not conducive to large-scale popularization and deployment.Therefore, the purpose of this subject is to develop a speaker verificationsystem for the smartphone,get the user voice and extract the features and training model and processingverification in smart phone. So it can realize the mass popularity and rapid deployment of the speaker verificationsystem.The speaker verificationsystem developed in this project is for the Android platform mobile phone users.

Keywords：Smart phone,speaker verification, feature extraction, training model, Android

目录

[1 背景 1](#_Toc417998998)

[1.1 课题背景 1](#_Toc417998999)

[1.2 开发背景 1](#_Toc417999000)

[1.3 研究目的和意义 2](#_Toc417999001)

[1.4 论文组织结构 2](#_Toc417999002)

[2 文献综述 3](#_Toc417999003)

[2.1 识别系统结构 3](#_Toc417999006)

[2.2 HTK工具箱 3](#_Toc417999007)

[2.3 Android 3](#_Toc417999008)

[2.4 SQLite数据库以及ORM框架 4](#_Toc417999009)

[2.5 Apache-commons-io包 5](#_Toc417999010)

[3 系统需求分析 6](#_Toc417999011)

[3.1 系统概述 6](#_Toc417999015)

[3.2 功能描述 6](#_Toc417999016)

[4 详细设计 7](#_Toc417999017)

[4.1 总体设计 7](#_Toc417999022)

[4.2 体系结构 7](#_Toc417999023)

[4.3 开发平台及开发工具 7](#_Toc417999024)

[4.4 各个模块功能设计 7](#_Toc417999025)

[5 用户功能和实现 8](#_Toc417999026)

[5.1 用户登陆 8](#_Toc417999032)

[5.2 修改密码 8](#_Toc417999033)

[5.3 解锁界面 8](#_Toc417999034)

[5.4 用户管理 8](#_Toc417999035)

[5.5 用户训练 8](#_Toc417999036)

[6 实验和测试 9](#_Toc417999037)

[6.1 实验遇到的问题和解决方法 9](#_Toc417999044)

[7 展望与总结 10](#_Toc417999045)

[7.1 总结 10](#_Toc417999053)

[7.2 展望 10](#_Toc417999054)

[参考文献 11](#_Toc417999055)

[致谢 12](#_Toc417999056)

## 背景

### 课题背景

说话人识别最初被广泛应用是在司法领域，用于帮助对嫌疑人的查证或判定罪犯，经过不断发展后进入安保和军事领域，用于一些机密场所出入控制，机要设备的使用控制和战场监听等等。

出此之外，说话人识别还有着广阔的市场应用前景。例如在通信和互联网尤其是新兴的移动互联网领域，说话人识别技术可以应用于语音拨号，电话银行，信息服务，安全控制，账户登录，E-mail，即时通信等。

说话人识别是一种生物认证技术。通过语音信号中的波形变化反映说话人生理和应为上的特征，并根据特征识别说话人。这些特征涉及到说话人的年龄，性别，感情，种族等等。

说话人识别的大致过程是首先录入说话人的语音样本，提取其中的语音特征并保存以待应用。在识别时将待测试的语音的特征与保存的语音特征做比较，从而确定说话人身份。

### 开发背景

#### 开发平台的选择

本课题开发的说话人识别系统是面向Android平台手机用户。Android是一种基于Linux的开源的自由的操作系统，主要使用于智能手机，平板电脑，相机，游戏机，电视机等移动设备。最初由由Andy Rubin开发，后在 2005年8月被Google收购，随后Google以Apache开源许可证和GPL的授权方式，发布了Android的源代码。

从Android刚刚发布时的1.0版本到目前的5.0版本，Android逐渐从一个崭新的系统发展为最成熟的移动端操作系统。

Android作为市场份额最大的手机端操作系统，得益于其的平台开放性和开发开放性。Google允许任何手机终端厂商加入Android平台联盟，也正因为众多的手机厂商加盟，使得Android平台拥有了众多的开发者，开发的Android应用更加丰富，用户数越来越多。

Google提供给Android一个相当自由的开发环境，有着不停更新的Development Toolkit，和较少约束的系统API，使得开发者可以自由地进行开发。

选择Android平台开发，主要是看重它丰富的硬件资源的开发上的开放性。

#### 认证方式的选择

根据识别系统对待识别语音内容的不同，又可以分为文本相关和文本无关两种方式。

文本相关的说话人识别方式要求说话人发音的关键词或关键句子与训练文本相同，且在识别时也要按照相同内容发音。

文本无关的说话人识别方式在训练和识别时都不对说话人的的发音内容做要求，其识别对象是任意的语音信号。

由于文本无关的说话人识别方式的无法控制的特性，而文本相关的识别系统的鲁棒性较强，本课题所研究开发的系统采用文本相关的说话人识别方式。

### 研究目的和意义

通过本课题的面向Android平台手机用户的说话人识别系统，可以对用户进行生物身份识别确认以替代原本的手机上的数字密码验证，避免了数字密码遭非法用户窃取的风险。

除此之外本课题还使得生物身份识别系统更简单易用，在安保、司法等以外领域的地方部署，可以极大地促进说话人识别技术的发展。

最后，目前移动互联网行业发展迅速，但是安全性还是原地踏步，本系统还可以与Android平台上的其他应用进行绑定，以极大地增强一些移动互联网应用的安全性，例如手机支付钱包等功能。

### 论文组织结构

本文组织如下，第一章，背景，主要介绍课题背景，开发平台和认证方法的选取，课题的目的和意义等；第二章，文献综述，主要介绍系统开发所需要基本知识；第三章，系统的需求分析；第四章，系统的详细设计；第五章，系统的主要功能及实现；第六章，实验中所遇到的问题及解决方法；第七章，对未来的展望和当前系统的总结。

## 文献综述



### 识别系统结构

说话人识别系统主要分为三个模块，分别是特征提取模块，模型训练模块，模型识别模块。

#### 特征提取模块

特征提取模块首先对要处理的语音信号进行预加重、分帧、加窗等处理。预处理后得到大量的数据量。提取特征这一过程就是为了去除原来语音中的冗余信息，减少数据量。

当频率在1000HZ以下时，人们的耳朵的感知能力与频率成线性关系；但当频率在1000HZ以上时，人们的耳朵的感知能力与频率不再构成线性关系，而更偏向于对数关系，这就使得人们的耳朵相比于对比高频信号来说对低频信号更敏感，故采用Mel 频率倒谱系数(即 MFCC)。

Mel 频率倒谱系数提取过程如图2.2所示：

1. 对语音信号s(n)进行预加重，分帧，加窗等处理，得到每个帧的时域信号x(n)。
2. 将上述对时域信号x(n)后补0，直至形成长度为N的序列(N一般为256)，然后进行快速傅立叶变换(FFT)后得到线性频谱x(k)：

|  |  |
| --- | --- |
|  | ——公式2-1 |

1. 将上一步的线性频谱x(k)通过Mel频谱滤波器，并取每个三角形的滤波器频率带宽内的所有信号幅度加权和作为某个带通滤波器的输出，从而得到Mel频谱，并通过对数能量的处理，得到对数频谱s(m)：

|  |  |
| --- | --- |
|  | ——公式2-2 |

为各个带通滤波器的传递函数。

1. 将上一步的对数频谱s(m)经过离散余弦变换(DCT)得到倒谱频率，即可得到Mel频率倒谱系数c(n)：

|  |  |
| --- | --- |
|  | ——公式2-3 |

图2.2 Mel 频率倒谱系数提取过程

#### 模型训练和识别模块

在特征被提取出来后，需要用模型为说话人建模，并根据特征以标记其是属于哪一个说话人。一般常用的建模模型有参数模型、非参数模型、人工神经网模型和支持向量机模型。

典型的参数模型包括高斯混合模型(GMM)，隐马尔科夫模型(HMM)，也是在语音识别领域，最常用到的两个模型。高斯混合模型常用于文本无关的语音识别，而隐马尔科夫模型常用于文本相关的语音识别。

对于一个时间长度为T的语音序列X=()，它的GMM似然函数可以写作：

|  |  |
| --- | --- |
|  | ——公式2-2 |
|  | ——公式2-2 |

识别时运用贝叶斯定理，在N个未知话者的模型中，得到的最大似然概率的模型对应的说话人即为识别结果。

|  |  |
| --- | --- |
|  | ——公式2-2 |

### 隐马尔科夫模型(HMM)

隐马尔科夫模型（Hidden Markov Model，HMM）是一个统计模型，用来描述一个含有隐含未知参数的状态跳转过程。其要点在于从可观察的参数中找出这一系列状态跳转过程中的隐含参数。然后利用这些参数来做进一步的分析，例如语音识别。

在正常的马尔科夫模型中，状态参数对于观察者来说是完全直接可见的。而在隐马尔科夫模型中，状态参数并不是直接可见的，但模型中受状态影响的某些参数变量则是可见的。

例如在某地的某人在雨天喜欢宅在家中，在晴天时喜欢出去踢球，那么当你得知这个人在某一天出去踢球(或宅在家中)，就可以推断出这个人所在的地方的天气是晴天(或雨天)。这个人告诉了你的他的活动，也就是一个受天气影响的参数变量，而我们对天气并不是可知的，这样一个系统就是隐马尔科夫型模型。

通过给定测试序列(测试用户的语音特征序列) 和模型参数(事先训练好的语音特征序列集合)，再利用Viterbi算法寻找某种意义上最优的状态序列(测试用户的身份)。这就是在文本相关语音识别领域使用隐马尔科夫模型的过程。

### HTK工具箱

#### HTK工具箱概述

HTK工具箱是一个用于构建和操作隐马尔科夫模型的便携式工具箱。主要用于语音识别的研究。

HTK工具箱的多数功能已被建立成库模块。这些模块确保了每个工具对外部的接口使用完全相同的方法，并提供了通用功能的中心资源。图2.1说明了HTK工具箱的架构和输入输出接口。

* HShell控制用户的输入输出以及与操作系统的交互。
* HMem控制所有的内存管理。
* HMath提供数学支持。
* HSigP完成语音分析所需的信号量处理操作。
* HLabel提供标签文件的接口。
* HLM用于语言模型文件。
* HNet用于网络和网格。
* HDict用于字典。
* HVQ用于矢量量化码本。
* HModel用于隐马尔科夫模型（HMM）的定义。
* HWave完成所有在波形阶段的语音的输入输出。
* HParm完成语音的参数化阶段。
* HWave和HLabel支持多文件格式并允许数据从其他系统输入。
* HAudio支持音频的直接输入。
* HGraf提供简单的交互式图形。
* HUtil提供大量用于计算HMM的通用程序。
* HTrain和HFB支持多种HTK训练工具。
* HAdapt提供多种适应工具的支持。
* HRec包含了识别过程中使用的主要函数。



图2.1 HTK工具箱的架构和输入输出接口

#### HTK工具箱使用流程

HTK工具箱的使用有四个主要的阶段：数据准备，训练，测试，分析。(图2.2)

* 数据准备

对于语音信号，可以使用HCopy参数化波形，得到MFCC特征文件。

* 训练

由HInit读入所有的引导训练数据，并迭代计算一组初始参数值。通过HInit计算的初始参数值由HRest进一步重估，直至收敛。

* 测试

HVite使用令牌传递算法来执行基于维特比的语音识别。HVite把输入看作是一个网络，该网络描述了所容许的单词序列，并定义了每个单词如何发声的字典和一组HMM。它将单词网络转换成音素网络，之后在每个音素实例附上合适的HMM定义，以实现运转。接着就可在一系列存储的语音文件上或者直接的音频输入上进行识别。

* 分析

基于HMM的识别系统的性能评估由HResults完成。



图2.2 HTK工具箱使用的四个阶段

### Android

Android开发有4个层次(如图2.3所示)，自下而上分别是Linux内核层开发，类库层开发，应用框架层开发，应用层开发。

第一层Linux内核层开发由C实现，主要负责管理Android硬件的驱动。

第二层类库层开发依然由C实现，主要负责Android上一些常用类库和组件的开发，例如数据库SQLite，浏览器内核WebKit，常用的C函数库libc，运行应用的虚拟机Dalvik。

第三层是Android系统应用框架，由Java实现，包含了Activity Manager界面管理，Notification Manager通知管理，Resource Manager资源管理等等，实现了对Linux内核层和类库层的封装，让Android不过度依赖与Linux内核，达到内核无关的特性，让Android框架开发和Android应用开发能在不考虑Linux内核和驱动下顺利完成，以吸引更多的开发者。

第四层是Android应用程序的开发，由Java实现，可以利用Android提供的API接口实现各种功能的应用，例如信息服务，拍照，录音，定位等等，以及各种界面控件完成用户交互，例如文本框EditText，按钮Button等等。

图2.3 Android开发层次图

Android应用程序由4大组件构成，分别是Activity、Service、Broadcast Receiver、Content Provider。Activity是Android应用与用户交互的界面，可以显示一些控件也可以监听并处理用户的事件做出响应。Broadcast Receiver用来对外部事件进行过滤，从而只获得自己感兴趣的内容。Service 是没有用户界面的程序，主要用来开发如监控类等后台程序。Content Provider用于给应用提供各种各样的数据，例如SQLite数据库，SharedPreferences配置文件。

本课题的开发的说话人识别系统主要在第二层和第四层完成。先在第二层中移植好HTK语音工具箱类库，再在第四层中开发一个应用调用这个类库，完成语音特征提取，训练建模，测试识别等任务。

### SQLite数据库以及ORM框架

SQLite是Android应用中最常用的数据库，不同于MySql、MS SQL Server等客户端服务器数据库，SQLite是一个单文件数据库。使用C构建，并集成在Android系统的第二层——类库层中，每个Android应用都可以获得一个专属的SQLite数据库。Android应用在开发时可以使用SQLiteOpenHelper类中的方法，将应用的Context作为参数传入，得到这个应用的SQLiteDatabase，之后就可以通过execSQL方法将SQL语句传入并执行。

然而使用SQLiteOpenHelper提供的方法操作数据库并不符合面向对象的软件开发思想。因此本课题在操作数据库的过程中使用了ORM对象关系映射(Object Relational Mapping)框架，目前ORM框架可谓汗牛充栋，但为了能扎实掌握这一门技术以及更好地适应于本课题的在该方面的需求，在借鉴了一些ORM框架后，改编了其中之一xUtils，使其成为更小更适合本课题的xutil\_orm框架。通过使用ORM框架，不仅可以省去撰写SQL语句的困扰，还可以更好地对一些异常和事务进行处理。

### Apache-commons-io包

本课题中涉及到不少文件的操作，例如WAV语音文件的新建、复制与删除，MFCC特征文件的新建、复制与删除，HMM模型文件的新建、复制与删除，以及文件路径的获取和传递。为了使得本系统有更好的健壮性，以及遵守软件工程中“不重复造轮子”的思想，本系统中使用了Apache-commons-io包。通过其中的FileUtils类中的不同的方法可以清晰简单地对文件、文件夹进行新建、修改、删除、复制、获取路径等操作。使用Apache-commons-io类库可以提升开发效率，并且更好地处理一些文件异常，增强应用的健壮性。

## 系统需求分析



### 系统概述

经过简单的需求分析，本课题开发的说话人识别系统主要有一下几个模块(图3.1)：



图3.1功能模块

### 模块功能描述

#### 登录模块

系统的登录界面，要求用户输入密码，防止非法用户登录本系统进行操作。

#### 主界面模块

系统的主界面，是其他功能模块的入口所在。

#### 训练模块

系统的训练界面，通过AudioRecord类实现对用户的语音的记录，并利用HTKTools中的HCopy进行特征提取处理得到MFCC模型，再利用HTKTools中的HInit和HRest依次进行建模训练得到迭代两次过后的HMM模型。

#### 测试模块

系统的测试界面，通过AudioRecord类实现对用户的语音的记录，并利用HTKTools中的HCopy进行特征提取处理得到MFCC模型，再利用HTKTools中的HVite对MFCC模型和原本的HMM模型结合打分，得到测试结果。

#### 锁屏界面模块

通过Android系统的API对Android手机进行锁定，通过AudioRecord类实现对用户的语音的记录，以进行特征提取和解码，当通过系统确认时锁定解除。

#### 用户管理模块

系统的用户管理界面，是用户训练界面的上一级入口。在该界面上可以添加删除说话人或是查看用户详细信息。

## 详细设计



### 总体设计

Android上应用开发普遍遵循MVC设计模式，MVC的全称是Model模型，View视图和Controller控制器。在Android应用上的体现就是Entity实体例如文件、用户，XML界面布局和Activity界面模块。通过MVC设计模式，可以将数据，业务逻辑和界面解耦，将这三部分分别都集中部署，在定制数据结构、改进业务逻辑或修改界面时可以不需要重新编写全部代码。使用MVC设计模式可以提高代码质量，提升开发效率。

### 体系结构

#### 模型层Model

模型层主要包含有用户、KeyValue参数、文件等结构。分别用于表示用户信息（包含编号、姓名、训练状态、训练时间、测试时间等信息），系统参数（包含系统密码、状态参数等信息），和训练后生成的相关文件。

用户和KeyValue参数储存于数据库中，并通过AccessObject封装后访问，可以进行添加删除修改用户，或修改系统参数等操作。

#### 视图层View

视图层包含有初始注册界面、登陆界面、主界面、用户管理界面、训练界面、测试界面、修改密码界面。

初始注册界面用以在安装系统过后初始化登陆密码。

登陆界面用以输入并校验登陆密码，防止非法用户登陆。

主界面包含有其他界面的入口，是整个系统的中心。

用户管理界面用以添加删除说话人账户，是训练界面的入口。

训练界面包含有录音功能，帮助用户录音，以进行特征提取和模型训练。

测试界面包含有录音功能，帮助用户录音，以确认该录音是否来自用户声明的说话人。

修改密码界面用于帮助用户修改密码。

#### 控制器层Controller

控制器层包含有登陆模块、主界面模块、用户管理模块、训练模块、测试模块。

登陆模块用以控制初始界面和登陆界面的用户交互。在安装后第一次登陆时获取界面上用户输入的密码并加密存储到数据库中。之后使用时再将获取界面上用户输入的密码加密后与数据库中的数据校验。若正确则跳转到主界面，不正确则继续输入直至密码正确。

主界面模块上含有不同界面的入口。当用户选取主界面上不同的控件时，则根据用户需要执行相应的操作。

用户管理模块用以控制用户管理界面的用户交互。根据用户的响应执行添加删除说话人，或进入该说话人训练模块的操作，同时通过AccessObject将添加或删除的操作保存到数据库。

训练模块用以控制训练界面的用户交互。在用户确认训练后开始录音，并对录音进行特征提取，再将特征进行建模并储存。

测试模块用以控制测试界面的用户交互。在用户声明了一个说话人后开始录音，并对录音进行特征提取，再将特征进行建模，最后再将建模的结果与之前训练的说话人对比，确认是否是合法的说话人。

### 数据结构设计

本系统数据结构主要包含三个部分，分别是用户信息、系统参数信息、文件。其中用户信息和系统参数信息存放于数据库中，文件存放于手机存储卡中。

USER用户信息表设计视图

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 允许null | 备注 |
| Id | Int | 不允许 | 编号，作为主键可自增长 |
| NAME | String | 不允许 | 姓名 |
| IS\_TRAINED | Boolean | 允许 | 训练状态 |
| TRAIN\_TIME | Date | 允许 | 训练时间 |
| LAST\_VERIFY\_TIME | Date | 允许 | 最后一次识别时间 |
| QUESTION | String | 允许 | 问题提示 |

KEYVALUE系统参数表设计视图

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 数据类型 | 允许null | 备注 |
| KEY | String | 允许 | 参数名，作为主键 |
| VALUE | String | 不允许 |  |

文件结构树形图

│ all.mmf // HVite解码时所需的所有hmm模型

│ config //存放HTK的配置信息

│ dict.txt //依据当前训练的用户生成的词典文件

│ gram.txt //语法格式文件

│ hmmlist.txt // HVite解码时所需的所有hmm模型的名称的列表

│ net.slf //HParse解析后生成的网格文件

│ reco.mlf //HVite解码识别结果

│ trainlist.txt //需要训练hmm模型的特征文件的名称的列表

│ wavlist.txt //需要生成特征文件的wav音频文件的名称的列表

│

├─hmm0 //存放执行HInit后的hmm模型结果的文件夹

│ hmm\_id4

│ hmm\_id5

│

├─hmm1 //存放第一次执行HRest后的hmm模型结果的文件夹

│ hmm\_id4

│ hmm\_id5

│

├─hmm2 //存放第二次执行HRest后的hmm模型结果的文件夹

│ hmm\_id4

│ hmm\_id5

│

├─lab //需要训练hmm模型的标签文件的文件夹

│ ├─id4

│ │ id4\_1.lab

│ │ id4\_2.lab

│ │ id4\_3.lab

│ │

│ └─id7

│ id7\_1.lab

│ id7\_2.lab

│

├─mfcc //存放提取了的MFCC特征文件

│ id4-540968289.mfc

│ id4-541404317.mfc

│

├─proto //存放最初始的hmm模型文件，以供训练使用

│ hmm\_id4

│ hmm\_id5

│

├─testwav //存放测试时录制的声音文件

│ id4-540968289.wav

│ id4-541404317.wav

│

└─trainwav //存放训练时时录制的声音文件

id4\_1.wav

id4\_2.wav

id4\_3.wav

### 开发平台及开发工具

开发平台：Android（系统运行平台）。

开发工具：Eclipse with Android Development Tool（集成开发环境）、Android SDK（Android应用编译工具）、Android NDK（交叉编译工具）、Microsoft Visual Studio 2013（x86环境下编译工具）。

Eclipse 是一个开放源代码的、基于Java的集成开发环境（IDE）。在刚面世时，主要是用来进行Java相关的开发，但是之后可以通过安装不同的插件，支持不同的计算机开发语言，比如C++和Python等。这使得Eclipse成为了一个框架平台，众多的插件使得Eclipse不同于MS Visual Studio等IDE，具有较强的灵活性。

本系统的开发使用到了Eclipse集成开发环境，以及其中的Android Development Tools插件。Android Development Tools为Eclipse扩展了功能，使得开发者能够较为轻松地建立Android项目，以及绘制应用程序界面。除此之外，还能调用Android SDK和Android NDK完成编译和调试。

Android SDK是Android软件开发工具包。包含开发者用于开发应用的各种开发工具。例如emulator模拟器、adb系统调试工具、ddms Dalvik虚拟机调试工具、ant编译工具等等。

Android NDK是Android Native Development Kit即原生开发工具包。包含了基于gcc的不同手机平台的toolchains交叉编译工具集。利用Android NDK可以将用C编写的HTK工具箱编译运行在Android应用中使用。

## 功能和实现



### 数据模型

* 用户信息的POJO类，允许获取或设置用户信息。

实现代码：

User.java

public class User {

private int id;

private String name;

private Boolean isTrained;

private Date trainTime;

private Date lastVerifyTime;

private String tips

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public Date getTrainTime() {

return trainTime;

}

public Date getLastVerifyTime() {

return lastVerifyTime;

}

……

}

* KeyValue的POJO类，允许获取或设置系统参数，例如系统密码、是否第一次使用等信息。

实现代码：

KeyValue.java

public class KeyValue {

private String key;

private String value;

public String getKey() {

return key;

}

public void setKey(String key) {

this.key = key;

}

public String getValue() {

return value;

}

public void setValue(String value) {

this.value = value;

}

}

* User的AccessObject类，使用singleton单例模式以节约内存。用来操作数据库中的用户表，包含有获得用户表、获得已训练用户表、添加用户、删除用户、训练用户、校验用户等操作。

实现代码：

UserAccessObject.java

public class UserAccessObject {

Context context;

DbUtils db;

static UserAccessObject singleton=null;

public static UserAccessObject getInstance(Context context) {

……

}

private UserAccessObject (Context context) {

……

}

public List<User> getUserList() {

……

}

public List<User> getTrainedUserList() {

……

}

public boolean addUser(User user) {

……

}

public boolean deleteUser(User user) {

……

}

public void trainUser(int id) {

……

}

public void verifyUser(int id){

……

}

}

* KeyValue的AccessAccessObject类，使用singleton单例模式以节约内存。用来操作数据库中的KeyValue表。包含有设置密码、检查是否第一次登陆、校验密码、获取密码等操作。

实现代码：

KeyValueAccessObject.java

public class KeyValueAccessObject {

DbUtils db;

static KeyValueAccessObject singleton=null;

public static KeyValueAccessObject getInstance(Context context) {

……

}

private KeyValueAccessObject (Context context){

……

}

public boolean validate(String password) {

……

}

public String getOldPassword() {

……

}

public boolean isFirstLogin() {

……

}

public boolean setPassword( String password) {

……

}

}

* 各类文件的AccessObject类。使用singleton单例模式以节约内存。用来进行各种文件操作，包括获得目录应用根目录、获得存放模型目录、获得存放mfcc特征目录、创建lab文件、创建dict文件、创建netslf文件、创建模型集合文件等操作。

实现代码：

FileAccessObject.java

public class FileAccessObject {

String appRoot;

String hmm0Path;

String hmm1Path;

String hmm2Path;

String protoPath;

String mfccPath;

String labPath;

String labUserPath;

String trainWavPath;

String testWavPath;

String resultFile;

Context context;

static FileService singleton=null;

public static FileAccessObject getInstance(Context context) {

……

}

private FileAccessObject (Context context) {

……

}

public String getAppRoot() {

……

}

public String getHmm0Path() {

……

}

public String getHmm1Path() {

……

}

public String getHmm2Path() {

……

}

public String getProtoPath() {

return protoPath;

}

public String getMfccPath() {

……

}

public String getLabPath(String userid) {

……

}

public String getTrainWavPath() {

……

}

public String getTestWavPath() {

……

}

public String getConfigFilePath() {

……

}

public String getDictFilePath() {

……

}

public String getSlfFilePath() {

……

}

public String getResultFilePath() {

……

}

public String createLab(String userid) {

……

}

public String createWavList(String wavPath, String userid, boolean isTrain) {

……

}

public String createProto(String userid) {

……

}

public String createTrainList(String userid) {

……

}

public void deleteUser(String userid) {

……

}

public String createGram(List<User> userList) {

……

}

public String createDict(List<User> userList) {

……

}

public String createHmmListFile(List<User> userList) {

……

}

public String getHviteE() {

……

}

public String createAllMmf(List<User> userList) {

……

}

public String parseRecoMlf() {

……

}

}

### HTK封装实现

* 要在Android平台上使用HTK工具箱，需要对HTK工具箱进行编译和封装，本系统使用到了HTK工具箱中的HCopy、HInit、HParse、HRest等。

1. Android NDK的编译描述文件，此处以HCopy为例，不详细赘述。

实现代码：

Android.mk

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_MODULE := libHTKCore

LOCAL\_SRC\_FILES := esignal.c esig\_asc.c esig\_edr.c esig\_nat.c HAdapt.c HArc.c HAudio.c HDict.c HExactMPE.c HFB.c HFBLat.c HGraf.null.c HLabel.c HLat.c HLM.c HMap.c HMath.c HMem.c HModel.c HNet.c HParm.c HRec.c HShell.c HSigP.c HTrain.c HUtil.c HVQ.c HWave.c strarr.c Tool\_HCopy.c Tool\_HInit.c Tool\_HParse.c Tool\_HRest.c Tool\_HVite.c

include $(BUILD\_STATIC\_LIBRARY)

include $(CLEAR\_VARS)

LOCAL\_MODULE := HCopy

LOCAL\_SRC\_FILES := com\_flo\_util\_HCopyFunc.c

LOCAL\_WHOLE\_STATIC\_LIBRARIES := libHTKCore

include $(BUILD\_SHARED\_LIBRARY)

……

1. Java中利用Java Native Interface接口调用编译好的C动态共享库，此处以调用HTK中的HCopy为例

实现代码：

HCopyFunc.java

public class HCopyFunc {

static {

System.loadLibrary("HCopy");

}

public static void exec(String configFilePath, String wavlistPath) {

HCopy(configFilePath, wavlistPath);

}

private native static void HCopy(String configFilePath, String wavlistPath);

}

HTK中HCopy的JNIEXPORT封装。

实现代码：

com\_flo\_util\_HCopyFunc.c

JNIEXPORT void JNICALL Java\_com\_flo\_util\_HCopyFunc\_HCopy

(JNIEnv \* env, jclass j, jstring jconfigFilePath,jstring jwavlistPath) {

char\* configFilePath = (char\*) (\*env)->GetStringUTFChars(env,jconfigFilePath, 0);

char\* wavlistPath = (char\*) (\*env)->GetStringUTFChars(env,jwavlistPath, 0);

int argc=5;

char \*argv[5];

argv[0]="HCopy";

argv[1]="-C";

argv[2]=configFilePath;

argv[3]="-S";

argv[4]=wavlistPath;

main\_HCopy(argc, argv);

}

### 用户登陆

* 为了防止非法用户登陆本系统修改数据，本系统使用密码来管理使用权限。在第一次使用时需要设置一次密码，并储存到数据库中。若不是第一次登陆则要求输入密码并校验是否正确。

实现代码：

LoginActivity.java

if (loginAccessObject.isFirstLogin()) {

……

editText\_NewPassword = (EditText) view1

.findViewById(R.id.editText\_NewPassword);

editText\_ConfirmPassword = (EditText) view1

.findViewById(R.id.editText\_ConfirmPassword);

button\_Register.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

String password1 = editText\_NewPassword.getText()

.toString();

String password2 = editText\_ConfirmPassword.getText()

.toString();

if (password2.equals(password1)) {

if (password1.equals("")) {

ToastUtil.show(getApplicationContext(),

R.string.password\_blank);

} else {

loginAccessObject.setPassword(password1);

ToastUtil.show(getApplicationContext(),

R.string.register\_success);

Intent intent = new Intent(getApplicationContext(),

MainActivity.class);

startActivity(intent);

}

} else {

ToastUtil.show(getApplicationContext(),

R.string.passwords\_diff);

}

}

});

} else {

……

editText\_Password = (EditText) view2

.findViewById(R.id.editText\_Password);

button\_Submit.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

password1 = editText\_Password.getText().toString();

if (loginAccessObject.validateUser(password1)) {

Intent intent = new Intent(getApplicationContext(),

MainActivity.class);

startActivity(intent);

} else {

ToastUtil.show(getApplicationContext(),

R.string.password\_error);

}

}

});

}

### 主界面

* 主界面上主要用来响应用户的操作，当用户点击不同的控件时在、跳转到响应的模块。

实现代码：

MainActivity.java

此处以跳转到用户管理界面为例：

button\_UserManage.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

Intent intent = new Intent(getApplicationContext(),

UserActivity.class);

startActivityForResult(intent, 100);

}

});

### 用户管理

* 用户管理界面显示有全部用户的信息，并允许添加删除用户，或进入某一用户的训练界面。

实现代码：

UserActivity.java

绑定数据到ListView控件上：

private void bindView() {

listView = (ListView) findViewById(R.id.listView);

userMapList = list2Map(userAccessObject.getUserList());

adapter = new SimpleAdapter(this, userMapList, R.layout.item\_user,

new String[] { "textView\_Username", "textView\_TrainState",

"textView\_TrainTime", "textView\_TestTime" }, new int[] {

R.id.textView\_Username, R.id.textView\_TrainState,

R.id.textView\_TrainTime, R.id.textView\_TestTime });

listView.setAdapter(adapter);

}

绑定ListView单击事件：

private void bindListener() {

listView.setOnItemClickListener(new OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,

long arg3) {

button\_Train.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

Intent intent = new Intent(getApplicationContext(),

TrainActivity.class);

intent.putExtra("USERNAME", map

.get("textView\_Username").toString());

intent.putExtra("USERID", map.get("USERID").toString());

dialog.cancel();

startActivityForResult(intent, 100);

}

});

button\_Delete.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

……

userAccessObject.deleteUser(user);

}

});

}

绑定右上角添加按钮单击事件：

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.add\_user:

……

button\_Register.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

……

User user = new User();

user.setName(username);

user.setIsTrained(false);

userAccessObject.addUser(user);

……

}

});

return true;

case android.R.id.home:

finish();

break;

}

return super.onOptionsItemSelected(item);

}

}

### 修改密码

* 系统允许用户多次修改密码，以提高安全性。

实现代码：

MainActivity.java

button\_ChangePassword.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

……

button\_Register.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

if (loginAccessObject.validateUser(editText\_OldPassword

.getText().toString())) {

String password1 = editText\_NewPassword.getText()

.toString();

String password2 = editText\_ConfirmPassword

.getText().toString();

if (password2.equals(password1)) {

if (password1.equals("")) {

ToastUtil.show(getApplicationContext(),

R.string.password\_blank);

} else {

loginAccessObject.setPassword(password1);

ToastUtil.show(getApplicationContext(),

R.string.password\_change\_success);

dialog.cancel();

}

} else {

ToastUtil.show(getApplicationContext(),

R.string.passwords\_diff);

}

} else {

ToastUtil.show(getApplicationContext(),

R.string.password\_error);

}

}

});

……

dialog.show();

}

});

### 说话人训练

* 说话人训练时，首先将进行三次相同的录音以去除噪声。录音后得到3个wav音频文件，再将这3个wav音频文件通过HCopy进行批量特征提取，得到3个MFCC特征文件。最后在生成lab标签文件，并对这3个MFCC特征文件进行建模训练。

实现代码：

TrainActivity.java

开始录音并在3秒之后停止录音：

protected void startRecord(final int n) {

audioRecordFunc = AudioRecordFunc.getInstance();

int result = audioRecordFunc.startRecordAndFile(wavPath, wavString,

rawString);

……

new Handler().postDelayed(new Runnable() {

public void run() {

alertDialog.cancel();

stopRecord(n);

}

}, 3000);

}

停止录音进行特征提取并开始准备训练：

protected void stopRecord(int n) {

audioRecordFunc.stopRecordAndFile();

switch (n) {

case 1:

……

case 2:

……

case 3:

button\_Record1.setEnabled(false);

button\_Record2.setEnabled(false);

button\_Record3.setEnabled(false);

NativeHTK.createMFCC(fileAccessObject, wavPath, userid,true);

final Context mContext=this;

new Handler().postDelayed(new Runnable() {

public void run() {

NativeHTK.train(fileService, userid);

userAccessObject.trainUser(Integer.valueOf(userid.substring(2)));

ToastUtil.show(mContext, R.string.train\_end);

}

}, 1000);

break;

default:

break;

}

}

生成相关文件并开始训练。

实现代码：

NativeHTK.java

MFCC特征提取：

public static void createMFCC(FileAccessObject fileAccessObject, String wavPath,

String userid, boolean isTrain) {

String wavlist = fileAccessObject.createWavList(wavPath, userid, isTrain);

HCopyFunc.exec(fileAccessObject.getConfigFilePath(), wavlist);

}

执行HTK工具箱中的HInit、HRest开始训练：

public static void train(FileAccessObject fileAccessObject, String userid) {

String labUserPath = fileAccessObject.createLab(userid);

String protoFile = fileAccessObject.createProto(userid);

String trainlist = fileAccessObject.createTrainList(userid);

HInitFunc.exec(trainlist, fileService.getHmm0Path(), protoFile, userid,

labUserPath);

HRestFunc.exec(trainlist, fileService.getHmm1Path(),

fileService.getHmm0Path() + "/hmm\_" + userid, userid,

labUserPath);

HRest2Func.exec(trainlist, fileService.getHmm2Path(),

fileService.getHmm1Path() + "/hmm\_" + userid, userid,

labUserPath);

}

### 说话人测试

* 说话人测试时，先声明身份，再进行录音，录音后得到wav音频文件，其通过HCopy进行批量特征提取，得到MFCC特征文件。最后利用HTK工具箱中的HVite进行识别，若结果与声明的相同则测试通过，否则测试不通过。

实现代码：

AuthActivity.java

绑定已训练的说话人列表：

private void bindView() {

……

userMapList = list2Map(userAccessObject.getTrainedUserList());

if (userMapList.size() < 3) {

isSoundMode=false;

}

adapter = new AuthListViewAdapter(this, userMapList,

R.layout.item\_auth, new String[] { "textView\_UserName",

"imageButton\_UnLock", "userId" }, new int[] {

R.id.textView\_UserName, R.id.imageButton\_UnLock });

listView\_User.setAdapter(adapter);

}

AuthListViewAdapter.java

声明说话人身份及开始录音按钮绑定单击事件：

class ButtonListener implements OnClickListener {

……

@Override

public void onClick(View v) {

……

startRecord(userId);

}

}

开始录音并在3秒之后停止：

protected void startRecord(final String userId) {

……

audioRecordFunc = AudioRecordFunc.getInstance();

int result = audioRecordFunc.startRecordAndFile(wavPath, wavString,

rawString);

……

new Handler().postDelayed(new Runnable() {

public void run() {

stopRecord(userId);

alertDialog.cancel();

}

}, 3000);

}

停止录音进行特征提取并准备识别：

protected void stopRecord(String userId) {

audioRecordFunc.stopRecordAndFile();

NativeHTK.createMFCC(userAccessObject, wavPath, userId, false);

NativeHTK.test(fileAccessObject, userAccessObject, userId);

verify(userId);

}

验证录音结果是否符合声明，是则通过测试，否则报出提示：

protected void verify(String userId) {

boolean result=userId.equalsIgnoreCase(fileAccessObject.parseRecoMlf());

if(result){

int id=Integer.parseInt(userId.substring(2));

userAccessObject.verifyUser(id);

ToastUtil.show(mContext, R.string.unlock\_success);

AuthActivity authActivity=(AuthActivity)mContext;

authActivity.unLock();

authActivity.finish();

}else{

ToastUtil.show(mContext, R.string.unlock\_failure);

}

}

实现代码：

NativeHTK.java

利用HTK中的HParse生成词典网格文件，再用HVite进行识别测试：

public static void test(FileAccessObject fileAccessObject, UserAccessObject userAccessObject,

String userId) throws IOException, InterruptedException {

List<User> userList = userAccessObject.getTrainedUserList();

String gramFile = fileAccessObject.createGram(userList);

HParseFunc.exec(gramFile, fileService.getSlfFilePath());

String dictFile = fileAccessObject.createDict(userList);

String allMmfFile = fileAccessObject.createAllMmf(userList);

String netSlfFile = fileAccessObject.getSlfFilePath();

String hmmListFile = fileAccessObject.createHmmListFile(userList);

String resultFile = fileAccessObject.getResultFilePath();

String mfcFile = fileAccessObject.getMfccPath() + "/" + userId + ".mfc";

String hViteE = fileAccessObject.getHviteE();

HViteFunc.exec(hViteE, allMmfFile, resultFile, netSlfFile, dictFile,

hmmListFile, mfcFile);

}

## 实验和测试



### 实验遇到的问题和解决方法

在进行系统的需求分析和代码编写以及测试的过程中遇到了一些问题，经过查询资料、仔细分析和不断的调试，问题也得到了解决，所遇到的主要的问题如下。

#### HTK工具箱使用问题

HTK工具箱分为两个部分，分别是HTKLib和HTKTool。HTKLib是HTK工具箱的核心类库，包含HShell命令处理、HDcit词典、HUtil通用计算程序等。而HTKTool中的每个HTK工具是分别需要完整的HTKLib。在一开始编译Andorid平台上的HTK动态库时，并没有意识到这一点，而是将所有的HTKTool共享使用一个HTKLib，因此在实验时，当执行了一个HTKTool工具后在次执行时会出现内存溢出或占用的情况，经过仔细分析后，将所有的HTKTool各自生成独立使用的HTKLib，或是将HTKTool编译成可在Android平台上执行的二进制文件。

#### 识别率问题

在系统需求分析时，所有的说话人的语音文本被设计成是相同的，这样就导致了在系统初步编码完毕之后发现有识别率较低的问题，无法准确地识别出说话人。之后在老师的指导下，将不同的说话人的语音文本被设计成不同的，较大地提高了识别率，同时内置了一个噪声模型作为噪声说话人，使得系统的鲁棒性更强。

## 展望与总结



### 总结

本课题下的系统是面向Android平台手机用户的说话人识别系统的，可以丰富Android平台的安全性，解决传统的文本密码和图案认证的不安全性。

系统开发之后，之前在需求分析时定下的目标基本完成。

系统的界面美观大方，功能详实有效。用户管理模块界面简约清晰、功能完整，训练和测试模块界面操作便捷、提示丰富、功能有效、识别率较高。

作者自己在这次开发过程中也有不少体会和收获。技术方面，掌握了Android开发的基本知识，例如Activity、Service、Broadcast Receiver、Content Provider四大组件的生命周期和使用，以及组件之间的数据交互和数据存储。还掌握了Android NDK编译工具的使用，例如Makefile的编写和JNI接口的封装。识别环节上还了解了说话人识别的主要流程及所涉及到的算法，例如HTK工具箱的使用等等。

### 展望

## 参考文献

[1] 李兴华，王月清. Java Web开发实战经典基础篇[M]. 北京：清华大学出版社，2010. 90~324.

[2] 赵俊峰，姜宁，焦学理，梁立新. Java Web应用开发案例教程[M]. 北京：清华大学出版社，2012. 265~282.

[3] 孙卫琴. Tomcat与Java Web开发技术详解[M]. 北京：电子工业出版社，2012. 3~400.

## 致谢