**信号与信息处理综合实验**

**实验报告**

**学院： 电子信息工程学院**

**姓名： 高成鑫**

**学号： 17211270**

**小组成员： 李洺慧**

**指导教师： 陈新**

**北京交通大学**

**2020年 5月 18日**

目录

[1 总体描述： 3](#_Toc40733162)

[1.1 Android下的程序框架 3](#_Toc40733163)

[1.2 Android平台下一维声音信号的采集与处理实验 4](#_Toc40733164)

[2 设计内容 5](#_Toc40733165)

[2.1 FFT算法 5](#_Toc40733166)

[2.2滤波算法 6](#_Toc40733167)

[3.参考文献： 7](#_Toc40733168)

[4.总结与体会： 7](#_Toc40733169)

[5.附录代码 8](#_Toc40733170)

[5-1 FFT算法 8](#_Toc40733171)

[5-2 滤波算法（频域滤波，低通滤波） 9](#_Toc40733172)

多平台下一维声音信号的采集与处理实验

# 1 总体描述：

本次实验任务是首先是回顾信号与系统、数字信号处理相关的理论知识与分析方法，包括信号的频谱概念、FFT快速傅里叶变换、滤波器设计等；其次才是一维声音信号分析与处理，如在Matlab平台下、Visual Studio平台下，Adroid平台下的实现，我们是两个人合作完成，我主要完成的是Android平台下的信号分析与处理。

本次的实验过程大概的流程就是，录音、声音处理（频域分析）、滤波设计，本次实验的用时跨度较长，由于是第一次接触Android平台下的程序开发，对Android的认识与了解同样占据了本次实验的很大部分时间，除本身实验外，对Android平台的学习与实验的过程也将在报告中被提到。

## 1.1 Android下的程序框架

本次实验中，我们是使用Android Studio（版本3.6.3，以下简称AS）、安卓手机（Android 8.1系统）。在一个新的Project（使用Empty Activity模板）中，下图是本次实验中，我们主要进行改动的部分

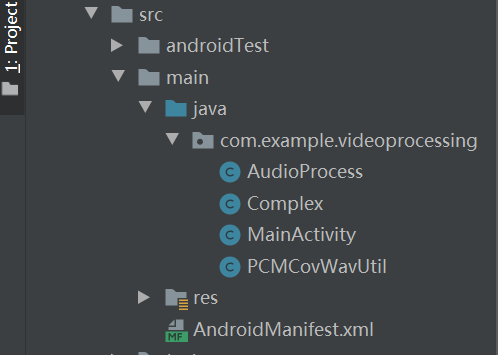
在图中文件夹里，AndroidManifest.xml是整个工程文件的配置文件，其中有对应用程序活动的注册等信息，本次实验需要的录音、读取/写入SD卡权限也在这里写入。

图 1 新的Project工程

Android程序设计讲究逻辑与视图分离，我们用java目录和res目录来将其分开。

java目录下是本次实验的代码文件，其中MainActivity是主活动（在上面的xml文件里注册并声明），其中定义了打开程序的点击按钮对应的函数，Complex是一个辅助类，用于表示复数运算的加减乘除过程，之后在计算FFT时候会用得到。PCMCovWavUtil是一个将PCM文件转化为WAV文件的类，实际上就是给PCM文件加上了44字节的头信息，这里不再详细描述。AudioProcessing类是本次实验的主要的类，因为里面定义了FFT算法、画出频谱图形、滤波算法等函数，这些函数在MainActivity中被实例化后调用，以完成了本次实验。

res文件夹下是本次的资源文件，程序中使用的各种图片、布局、字符串等都在这儿，本次主要修改了两个地方，一个是在drawable子文件夹下新增了button\_bg.xml，是按钮的背景文件，还有就是layout子文件夹下的activity\_main.xml文件，这是我们一打开应用图标看到的界面的定义，在里面添加了不同的button按钮，以及一个文本显示区（用于显示程序状态），一个图像/视频播放区（用于画频谱图象），最终程序的界面打开如下图。



图 2 Android平台下的用户界面

## 1.2 Android平台下一维声音信号的采集与处理实验

在mainActivity.java中，我们定义了按钮的不同事件，对定义等不再描述，以下是和本次实验有关的函数。

StartRecord():点击开始录音按钮对应的函数，录音过程中记录了音频文件的PCM数据，并实时画出了频谱图。（使用audioProcess类中的画图、fft等函数）

PlayRecord(): 播放录音文件

PlayRecord2()：播放处理后的录音文件

deleFiles():删除录音文件、处理后的文件

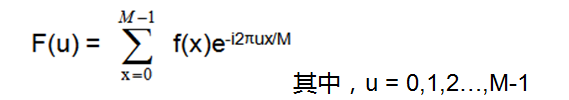
AnalysisRecord():将pcm转为wav文件（没什么用处，写了舍不得删）、对保存的pcm文件进行滤波，将结果保存在新的AfterProcessing.pcm中。

# 2 设计内容

根据上述描述，我们知道，在本次实验中所写的大部分代码，实际上是对布局、事件管理等其他的内容，其中主要的处理算法如下：

## 2.1 FFT算法

一维离散傅里叶变换的公式为：



如果直接基于该定义进行编程实现，则算法时间复杂度为O(N2)。

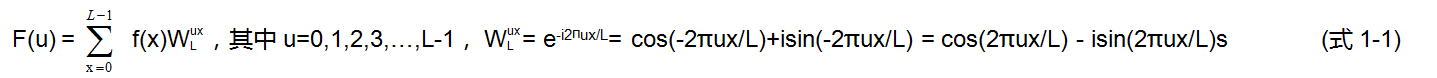
当一维信号长度达到几十万个信号时，当前主流4G主频CPU完成一次傅里叶变换需要约几十到几百秒的时间，这样的效率显然是让人无法接受的。

为了解决傅里叶变换的计算效率问题，行业专家们提出了蝶形算法，极大地提升了傅里叶变换的运算效率。

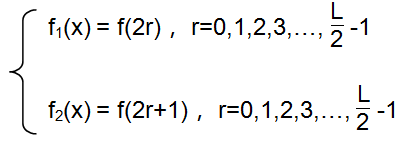
在蝶形算法中，较为流行的是基于时间抽取的基-2快速傅里叶变换算法（以下简称为基-2FFT算法）。

基-2FFT算法要求原始信号长度L=2N，N为正整数。也就是说，信号长度必须为2的整数次方，如4、8、16、32、64、512、1024。这是由蝶形算法的二进分解性质决定的。

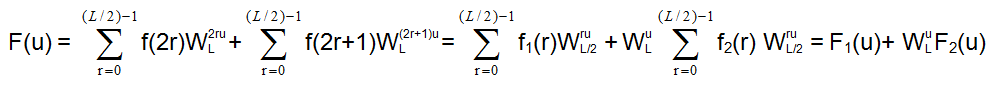
设原始信号序列为f(x)，长度为L（L=2N，为2的整数次方），x=0,1,2,3,…,L-1，则傅里叶变换可表达为：



将f(x)按x的奇偶性分成两组：



则式1-1变为：



其中，F1(u)和F2(u)分别为f1(x)和f2(x)的(L/2)点的一维离散傅里叶变换。

FFT算法用Java代码写下来附录代码5-1，对声音信号的实时分析见下图。

图 3 FFT算法对播放的2kHZ的正弦波声音信号的频域分析

## 2.2滤波算法

滤波算法包括频域的滤波分析，空间域的滤波等，从大的范围可分为以下三类

（一）克服大脉冲干扰的数字滤波法：

克服由仪器外部环境偶然因素引起的突变性扰动或仪器内部不稳定引起误码等造成的尖脉冲干扰，是仪器数据处理的第一步。通常采用简单的非线性滤波法。

（二）抑制小幅度高频噪声的平均滤波法

小幅度高频电子噪声：电子器件热噪声、A/D量化噪声等。通常采用具有低通特性的线性滤波器：算数平均滤波法、加权平均滤波法、滑动加权平均滤波法一阶滞后滤波法等。

（三）复合滤波法

在实际应用中，有时既要消除大幅度的脉冲干扰，有要做到数据平滑。因此常把前面介绍的两种以上的方法结合起来使用，形成复合滤波。去极值平均滤波算法：先用中值滤波算法滤除采样值中的脉冲性干扰，然后把剩余的各采样值进行平均滤波。连续采样N次，剔除其最大值和最小值，再求余下N－2个采样的平均值。显然，这种方法既能抑制随机干扰，又能滤除明显的脉冲干扰。

由于我们已经得到了频谱的信息，我们可以在频域进行滤波滤去高频的噪声，在进行IFFT得到处理后的数据（IFFT也可以根据FFT得到），频域滤波的函数放在附录代码5-2中，由于没有画处理后的频谱图，这里没有贴出，但根据实际使用中可以得到，高频部分的噪声被滤去，但同时，对于声音信号中本来就存在的高频声音也有所破坏。

# 3.参考文献：

1.音频PCM知识整理<https://blog.csdn.net/pds574834424/article/details/78174097>

2.安卓参考：PCM数据的采集和播放<https://blog.csdn.net/cyq7on/article/details/79691769>

3.Android音频处理——通过AudioRecord去保存PCM文件进行录制，播放，停止，删除功能<https://blog.csdn.net/qq_26787115/article/details/53078951>

4. C++实现一维快速傅里叶变换(FFT)<https://blog.csdn.net/shadown1ght/article/details/77628389>

5. c++算法学习之常见的6种滤波算法<https://www.cnblogs.com/variance/p/6994923.html>

6.Android AS下的NDK开发-Java与C混合编程(以硬件串口读写操作为例) <https://blog.csdn.net/u012902367/article/details/84797093>

7.Java DSP <https://github.com/sintrb/JavaDsp>

8. javaFFT算法 <https://blog.csdn.net/lwz45698752/article/details/81540855?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522158963284319726867815812%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=158963284319726867815812&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~blog~first_rank_v2~rank_v25-2-81540855.nonecase&utm_term=Java%E5%AE%9E%E7%8E%B0FFT>

9.音频格式简介和PCM转换成WAV <https://blog.csdn.net/u010126792/article/details/86493494?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-6.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromBaidu-6.nonecase>

10 任衍敏,侯锟,王大东.基于Android的声音频谱分析器研究与设计[J].佳木斯大学学报(自然科学版),2018,36(01):55-57. [http://kns.cnki.net/KXReader/Detail?TIMESTAMP=637253294611336250&DBCODE=CJFQ&TABLEName=CJFDLAST2018&FileName=JMDB201801015&RESULT=1&SIGN=zIOXEK27iMYIkm7Vk6fFOgNnX4o%3d#](http://kns.cnki.net/KXReader/Detail?TIMESTAMP=637253294611336250&DBCODE=CJFQ&TABLEName=CJFDLAST2018&FileName=JMDB201801015&RESULT=1&SIGN=zIOXEK27iMYIkm7Vk6fFOgNnX4o%3d)

11.基于Android的实时音频频谱仪[日期：2012-01-17] 来源：Linux社区  作者：sunshine\_okey https://www.linuxidc.com/Linux/2012-01/51770p2.htm

12.Android中SurfaceView使用详解<https://blog.csdn.net/android_cmos/article/details/68955134?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase>

# 4.总结与体会：

本次实验从原理上并不是很难得内容，所用到的知识都是已经学过的，而且我们在matlab环境里已经使用的比较熟练了，但这次实验还是比较难的，因为本次实验来说，我们花费了大量的时间在开发板的熟悉以及安卓平台的认识上，老师也是考虑到我们的陌生，第一次实验给的时间尤其多，对我们来说是一个好的消息，最后一路坐下来，虽然结果也不是很理想，但至少从中了解了很多以前没注意的，首先是信号处理方面的知识，例如时频域对应关系，数字频率与模拟频率，其次是编程能力等，或者说是阅读别人代码的能力，自己查找资料的能力，总的来说，是一场挺不错的体验。

# 5.附录代码

## 5-1 FFT算法

//快速傅里叶变换  
public void fft(Complex[] xin,int N)  
{  
 int f,m,N2,nm,i,k,j,L;//L:运算级数  
 float p;  
 int e2,le,B,ip;  
 Complex w = new Complex();  
 Complex t = new Complex();  
 N2 = N / 2;//每一级中蝶形的个数,同时也代表m位二进制数最高位的十进制权值  
 f = N;//f是为了求流程的级数而设立的  
 for(m = 1; (f = f / 2) != 1; m++); //得到流程图的共几级  
 nm = N - 2;  
 j = N2;  
 /\*\*\*\*\*\*倒序运算——雷德算法\*\*\*\*\*\*/  
 for(i = 1; i <= nm; i++)  
 {  
 if(i < j)//防止重复交换  
 {  
 t = xin[j];  
 xin[j] = xin[i];  
 xin[i] = t;  
 }  
 k = N2;  
 while(j >= k)  
 {  
 j = j - k;  
 k = k / 2;  
 }  
 j = j + k;  
 }  
 /\*\*\*\*\*\*蝶形图计算部分\*\*\*\*\*\*/  
 for(L=1; L<=m; L++) //从第1级到第m级  
 {  
 e2 = (int) Math.pow(2, L);  
 //e2=(int)2.pow(L);  
 le=e2+1;  
 B=e2/2;  
 for(j=0;j<B;j++) //j从0到2^(L-1)-1  
 {  
 p=2\*pi/e2;  
 w.real = Math.cos(p \* j);  
 //w.real=Math.cos((double)p\*j); //系数W  
 w.image = Math.sin(p\*j) \* -1;  
 //w.imag = -sin(p\*j);  
 for(i=j;i<N;i=i+e2) //计算具有相同系数的数据  
 {  
 ip=i+B; //对应蝶形的数据间隔为2^(L-1)  
 t=xin[ip].cc(w);  
 xin[ip] = xin[i].cut(t);  
 xin[i] = xin[i].sum(t);  
 }  
 }  
 }  
}

## 5-2 滤波算法（频域滤波，低通滤波）

/\*\*

\* 频域滤波

\* @param originalPoints 原始数据

\* @param samplingInterval 采样间隔, 单位是秒(s)

\* @param lowpass 低通

\* @return 滤波之后的数据

\*/

public static List<SamplingPoint> lowpassFromFrequencyDomain(List<SamplingPoint> originalPoints, double samplingInterval, int lowpass){

int dataLen = originalPoints.size();

double[] signal = new double[dataLen];

for(int i = 0; i < originalPoints.size(); i++){

signal[i] = originalPoints.get(i).getRealY();

}

double Fs = 1.0 / samplingInterval;//采样频率

double frequencyResolution = Fs / dataLen;//频率分辨率

//进行傅里叶变换,使用的是commons-math包里面的快速傅里叶算法

FastFourierTransformer fft = new FastFourierTransformer(DftNormalization.STANDARD);

Complex[] complexArr = fft.transform(signal, TransformType.FORWARD);//正向傅里叶变换

//进行滤波,注意条件

if(lowpass < Fs / 2.0){

for(int i = 0; i < complexArr.length; i++){

//对称

if(i \* frequencyResolution > lowpass && i \* frequencyResolution < (Fs - lowpass)){

complexArr[i] = new Complex(0, 0);

}

}

}

//反傅里叶变换

Complex[] timeDomainArr = fft.transform(complexArr, TransformType.INVERSE);

List<SamplingPoint> points = new ArrayList<>();

for(int i = 0; i < timeDomainArr.length; i++){

//只用获取实部，不用获取虚部(虚部理论上应该是0),实部的数据就是时域曲线

SamplingPoint point = new SamplingPoint(i, (float) timeDomainArr[i].getReal());

points.add(point);

}

return points;

}