第二次课程设计报告

无 52 肖善誉 2015011009

一、程序设计思路分析

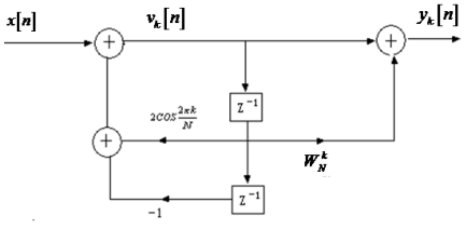
本次实验中，我们主要通过编写程序，来对特定频率的双音多频 DTMF 信号进行检测。运用了如下两种算法实现：

1.FFT：利用 FFT 算法对整个时域信号做离散傅里叶变换，得到整个频谱的信息。在已知 8 个不同的双音多频编码频率的情况下，我们可以提取这些特定频点的幅值（或功率密度大小），以此得到对应的按键。

2.Goertzel：利用相位因子的周期性，可将 DFT 定义表示成如下形式：

$$\mathrm{X[k]} = \sum_{n=0}^{N-1} x[n]W_N^{nk} = \sum_{n=0}^{N-1} x[n]W_N^{(n-N)k} = x[n] * W_N^{-kn}$$

因此，我们可以设计一冲击响应为W_N^{-kn}的滤波器，信号$x[n]$通过该滤波器后在$\mathrm{n} = \mathrm{N}$时刻的值就是特定频点的离散傅里叶系数。将该滤波器转化为差分方程形式：



$$\begin{cases} y_k[n] = v_k[n] - W_N^k v_k[n-1] \\ v_k[n] = x[n] + 2\cos(\omega_k)\, v_k[n-1] - v_k[n-2] \end{cases}$$

故该滤波器计算对应的 c++函数如下：

```
//v: the hidden states of filter, length 2. v[0], v[1] represent vk[n - 1], vk[n - 2]
//every time the func is called, v is updated
//a1, b1: cos(omega_k), Wk
void Goertzel_filter(double &x, double *v, complex<double> &y, double &a1,
complex<double> &b1) {
    double vn = 0.0;

    vn = x + 2.0 * a1 * v[0] - v[1];
    y = vn - b1 * v[0];

    //update state
    v[1] = v[0];
    v[0] = vn;
```

```
    return;
}
```

可以看出，在该函数实现中，每个采样点n处，该滤波器进行**2+1=3次实数加法**，**2+2=4次实数乘法**（计算$v_k[n]$的差分方程只涉及实数运算，实数+复数为1次实数加法，实数*复数为2次实数乘法）。

整个Goertzel算法函数如下：

```
void Goertzel(int N, int k, double *x, complex<double> &Xk) {
    //compute filter coeff
    double a1 = cos(2.0 * PI * k / N);
    complex<double> b1(cos(-2.0 * PI * k / N), sin(-2.0 * PI * k / N));

    double v[2] = { 0.0, 0.0 };
    complex<double> y;

    for (int i = 0; i < N; i++) {
        Goertzel_filter(x[i], v, y, a1, b1);
    }

    double xN = 0.0;
    Goertzel_filter(xN, v, y, a1, b1);

    Xk = y;
}
```

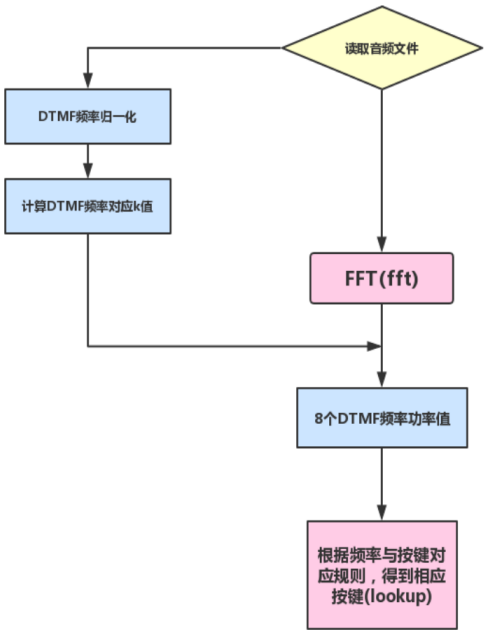
计算一个频点的频率值X[k]需要$(\mathbf{N}+\mathbf{1})$次滤波器操作。

在我们已知8个双音多频频率的前提下，我们只需要计算特定8个k值处的X[k]，然后可以知道整个信号由哪两个单频信号组成，以此得到对应的按键。

二、程序流程图

（1）下载附件包中第一小题的 10 个长度不一的音频文件，利用第一次课程设计中编写的 FFT 程序对这 10 个文件中的 DTMF 信号进行频谱分析，最后给出 10 文件所对应的真实数字。

主要流程如下（括号中为对应的 c++函数名）：



先将 8 个频率 697.0, 770.0, 852.0, 941.0, 1209.0, 1336.0, 1477.0, 1633.0Hz 归一化到$[0,2\pi]$。然后计算出这 8 个频率点在$X[k]$中对应的k。

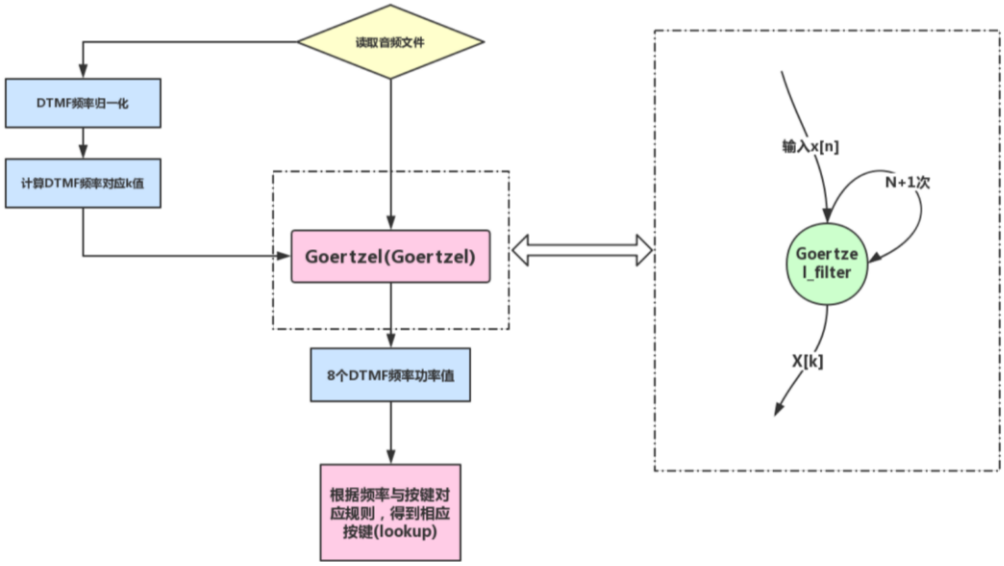
由于每次计算所得的k一般不是正整数（频率离散采样点很难完全在指定的频率上采样），因此每次索引到该k值上下两个整数值处的频率值，并比较它们的幅值大小，将幅值更大的那个值作为实际幅度值。

计算得到 8 个频点的幅度值后，分别找到行频率和列频率中幅度值最大的那个。根据频率查表(lookup 函数)得对应的按键。

具体请见 FFT_anaysis 函数。

（2）编写 Goertzel 算法的 C/C++语言程序，完成（1）中的要求。

主要流程如下（括号中为对应的 c++函数名）：



同样先将 8 个频率 697.0, 770.0, 852.0, 941.0, 1209.0, 1336.0, 1477.0, 1633.0Hz 归一化到$[0,2\pi]$。然后计算出这 8 个频率点在$X[k]$中对应的k，并每次计算该k值上下两

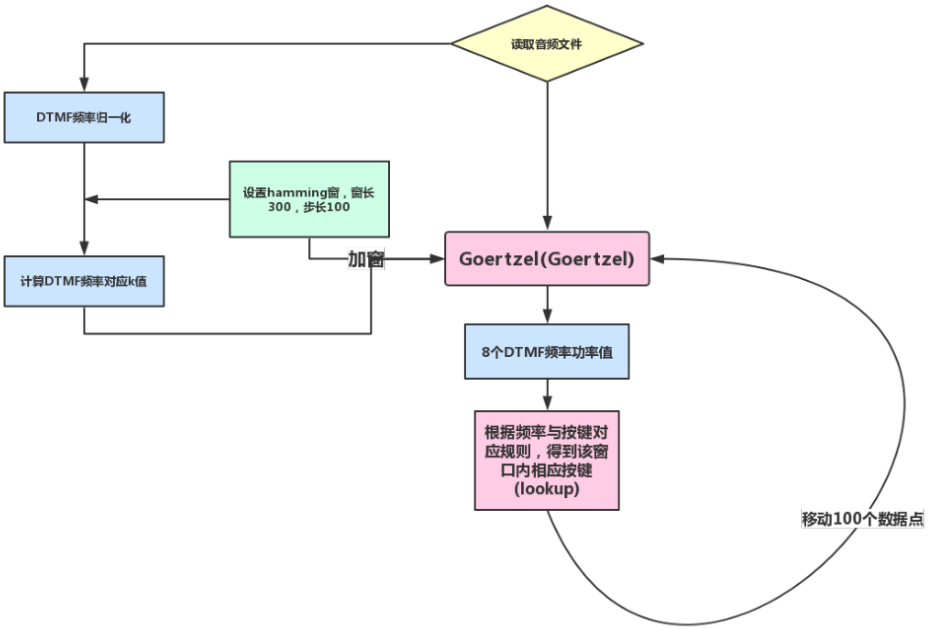
个整数值处的频率值，并比较它们的幅值大小，将幅值更大的那个值作为实际幅度值。

其余与上 FFT 流程相同。

具体请见 Goertzel_anaysis 函数。

（3）下载附件包中第二小题的一个长音频文件，文件中包含了一串 DTMF 信号，每个双音多频信号之间的时间间隔不一，对本串 DTMF 信号进行识别。

主要流程如下（括号中为对应的 c++函数名）：



由于每个双音多频信号之间的时间间隔不一，因此采用类似短时傅里叶变换（STFT）中的方法，在时域上加窗，分析每个窗口内的频率特征。

DTMF 的 8 个频率之间的间隔在73Hz~156Hz之间，因此选取窗长为 300，此时频率最小分辨率为$\frac{8000}{300}=26.7Hz$，频率分辨力足够。同时选取每次滑动的步长为 100，选取窗的类型为 hamming 窗。

噪声判断：

因为每段 DTMF 音频之间存在空隙，因此需要设置一定的条件来判断窗口内的信号是否是有有效信号。以行频率为例：选出 4 个频率中最大的幅度值，并求其他 3 个频率处的平均值。如果**最大的幅度值 > 5 倍平均值，则认为该窗口内的信号为有效信号**。如果认为不是有效信号，则输出字符 '_'。

具体请见 long_wav_anaysis 函数

三、不同算法结果与复杂度比较

（1）（2）FFT 与 Goertzel 算法结果如下：

```
FFT, data1081.wav: 5
Goertzel, data1081.wav: 5
FFT, data1107.wav: 1
Goertzel, data1107.wav: 1
FFT, data1140.wav: 6
Goertzel, data1140.wav: 6
FFT, data1219.wav: 9
Goertzel, data1219.wav: 9
FFT, data1234.wav: 8
Goertzel, data1234.wav: 8
FFT, data1489.wav: 7
Goertzel, data1489.wav: 7
FFT, data1507.wav: 3
Goertzel, data1507.wav: 3
FFT, data1611.wav: 4
Goertzel, data1611.wav: 4
FFT, data1942.wav: 0
Goertzel, data1942.wav: 0
FFT, data1944.wav: 2
Goertzel, data1944.wav: 2
```

可以看到：利用 FFT 与 Goertzel 算法的结果一致。但两种方法的算法复杂度不同。

FFT 需要$\frac{M}{2}log_2(M)$次复数乘法，即 $2\mathrm{M}\cdot log_2(M)$)次实数乘法，$\mathrm{M}log_2(M)$次复数加法，其中M为N向上补全至 2^m 的值。

而由程序设计思路中分析可知，Goertzel 算法只需要$16(\mathrm{N}+1)\cdot 4$ 次实数乘法，$16(\mathrm{N}+1)\cdot 3$ 次实数加法。

相比较于直接 FFT 而言，在序列长度 N 比较大时，Goertzel 算法的运算次数较少。

文件名	行频率	列频率	对应字符
data1081.wav	770	1336	5
data1107.wav	697	1209	1
data1140.wav	770	1477	6
data1219.wav	852	1477	9
data1234.wav	852	1336	8
data1489.wav	852	1209	7
data1507.wav	697	1477	3
data1611.wav	770	1209	4
data1942.wav	941	1336	0
data1944.wav	697	1336	2

（3）利用 Goertzel 算法对长 wav 文件分析结果如下：

```
long wav anaysis:
_____2222222222222222_____0000000000000000_____
___555555555555555_____88888888888888_____9999999999999_____
___1111111111111111_____11111111111111_____3333333333333333_____
_____22222222222222222_____00000000000000_____
_____444444444444444_____6666666666666666_____44444444444444_____
_____99999999999999_____
```

故实际的按键序列为 20589113204649。按键之间的信号被认为是噪声信号，以字符‘_’显示。

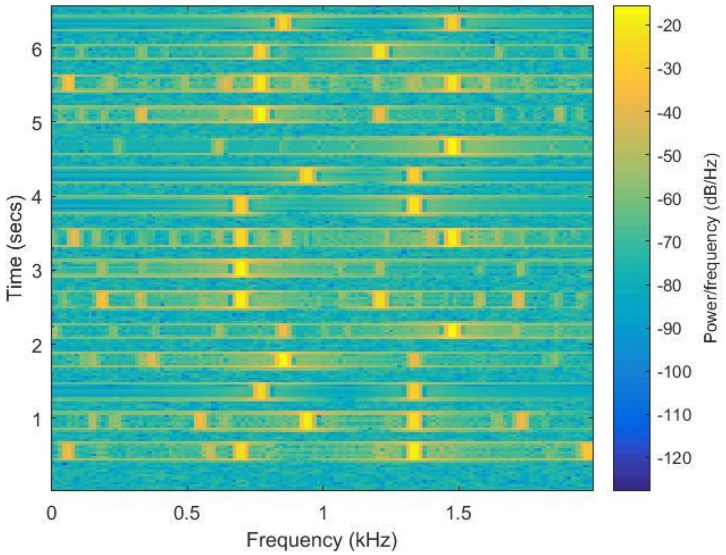
注意到整个音频中，倒数第五段信号的频率似乎有问题，似乎是一个单频信号，找不到对应的按键。长得有点像“3”，但对比整个按键序列的两个“3”的时频分析结果不完全一致，对比如下：

倒数第 5 段信号： 

倒数第 8 段信号： 

倒数第 5 段信号, 697HZ 频段有些偏移，因此被程序认为是噪声信号。

利用 matlab 时频分析得到的时频图作为比较，结果如下：



根据时频图中的结果，查表可得，按键序列确实为 2058911320(3)4649。因此利用 Goertzel 算法得到的结果正确。