数字信号处理第二次课程设计报告

DTMF 信号的检测与识别

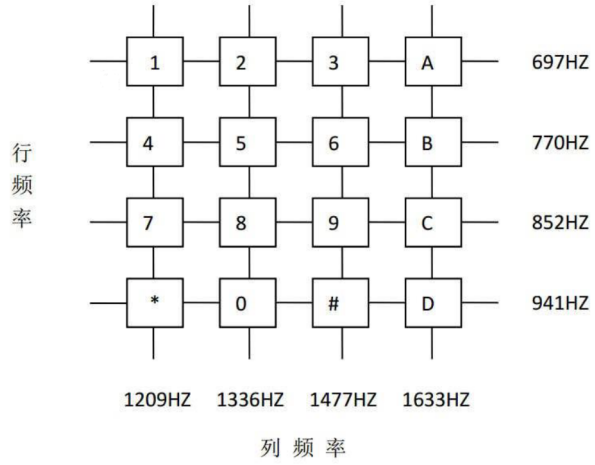
BobAnkh

2020 年 12 月 19 日

1 算法原理和设计思路

1.1 DTMF 信号产生

DTMF 信号可直接利用行频和列频两组正弦信号的叠加来产生。行频和列频分别包含 4 个频率，每个按键对应着一个行频和列频的频率组合，因此 DTMF 信号共有 16 个编码。DTMF 信号的具体频率配置见下图：



根据如下差分公式和框图设计数字正弦振荡器，进行正弦信号的产生，进而编写 DTMF 信号产生程序——根据读取到的键值查表得到对应的双频，从而产生对应的 DTMF 信号。

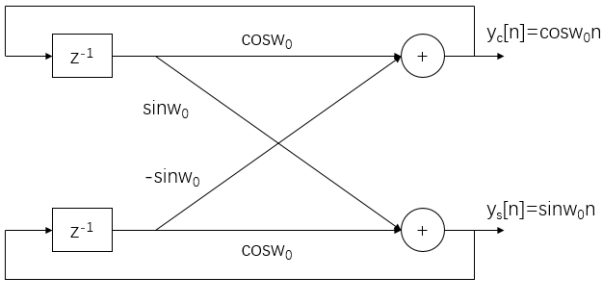
差分方程：

$$\begin{cases} y_c[n] = (cos\omega_0)y_c[n-1] - (sin\omega_0)y_s[n-1] \\ y_s[n] = (sin\omega_0)y_c[n-1] + (cos\omega_0)y_s[n-1] \end{cases}$$

起始条件：

$$\begin{cases} y_c[-1] = cos\omega_0 \\ y_s[-1] = -sin\omega_0 \end{cases}$$

框图：



1.2 DTMF 信号检测与识别

1.2.1 采用 FFT 方法来进行检测识别

利用 FFT 直接计算输入信号的 DFT，通过对信号整个频域信息的分析来检测 DTMF 的存在并识别相应的按键。

具体实现中，根据时域信号确定合适的 FFT 点数后采用 FFT 得到频谱，在可能的频率范围内，使用 Matlab 自带的 `findpeaks` 函数寻找峰值，找出两个最大的峰值并确定其对应的频率，根据频率查表来判别对应的按键值，部分参数的选取也是实际尝试之后得到。对于频率查表也采用了 20Hz 的误差容限。

编写了如下的 `find_key_fft` 函数来进行检测，接受时域信号序列和采样率，返回按键值 key，相关代码如下：

```
function key = find_key_fft(xn, fs)
%find_key_fft: 使用fft来寻找键值
%   xn: 待测音频序列
%   fs: 采样率
    keys = [
    '1', '2', '3', 'A';
    '4', '5', '6', 'B';
    '7', '8', '9', 'C';
    '*', '0', '#', 'D'];
    len_x = length(xn);
    N = 2 ^ nextpow2(len_x);
    X = fft(xn, N);
    X_abs = abs(X(1 : round(2000 / fs * N)));
    [~, locs] = findpeaks(X_abs, 'MinPeakHeight', max(X_abs) * 0.01, '
        MinPeakDistance', floor(200 / fs * N), 'SortStr','descend','
        MinPeakProminence',10);
    if length(locs) < 2
        key = 'X';
    else
        freq1 = (min(locs(1:2)) - 1) / N * fs;
        freq2 = (max(locs(1:2)) - 1) / N * fs;
        if freq1 > 677 && freq1 < 717
            row = 1;
        elseif freq1 > 750 && freq1 < 790
            row = 2;
        elseif freq1 > 832 && freq1 < 872
```

```
            row = 3;
        elseif freq1 > 921 && freq1 < 961
            row = 4;
        else
            row = -1;
        end

        if freq2 > 1189 && freq2 < 1229
            col = 1;
        elseif freq2 > 1316 && freq2 < 1356
            col = 2;
        elseif freq2 > 1457 && freq2 < 1497
            col = 3;
        elseif freq2 > 1613 && freq2 < 1653
            col = 4;
        else
            col = -1;
        end
        if row == -1 || col == -1
            key = 'X';
        else
            key = keys(row, col);
        end
    end
end
```

1.2.2 采用 Goertzel 算法来进行检测识别

考虑到检测 DTMF 信号时只关心其 8 个已知频率的行频/列频信息，故可以有选择地计算特定频率点处的频域信息，以大大降低计算复杂度。Goertzel 算法正是利用相位因子的周期性，将 DFT 计算表示为线性滤波形式，实现了有选择地计算特定频点处频域信息的效果。基于一定的数学推导，可以得到如下的迭代公式：

$$v_k[n] = 2cos(\omega_k)v_k[n-1] - v_k[n-2] + x[n]$$
$$y_k[n] = v_k[n] - W_N^k v_k[n-1]$$

具体实现中，按照迭代公式进行运算，取前四位最大值对应的频点为行频，后四位最大值对应的频点为列频，同样查表来判别对应的按键值。

编写了如下的 `find_key_goertzel` 函数来进行检测，接受时域信号序列和采样率，返回按键值 key，相关代码如下：

```
function key = find_key_goertzel(xn, fs)
%find_key_goertzel: 使用goertzel来寻找键值
%   xn: 待测音频序列
%   fs: 采样率
    freq_list = [697, 770, 852, 941, 1209, 1336, 1477, 1633];
    keys = [
    '1', '2', '3', 'A';
    '4', '5', '6', 'B';
```

```
        '7', '8', '9', 'C';
        '*', '0', '#', 'D'];
    N = length(xn);
    omega_list = 2 * pi * round(freq_list / fs * N) / N;
    cos_wk = cos(omega_list);

    v0 = zeros(1, 8);
    v1 = zeros(1, 8);
    v = zeros(1, 8);
    for a = 1 : N
        v0 = v1;
        v1 = v;
        v = 2 * cos_wk .* v1 - v0 + xn(a);
    end
    XK = v - v1 .* exp(-1j * omega_list);
    [~, row] = max(abs(XK(1:4)));
    [~, col] = max(abs(XK(5:8)));
    key = keys(row, col);
end
```

1.2.3 长音频文件识别

在本问中，主要需要实现的是分段的算法。具体的算法思路是：从左到右扫描整段时域音频信号，如果发现目前的值大于某一阈值，则认为这是某一段 DTMF 信号的起点，此时继续向右扫描；如果发现某一点及其后连续 100 个点均小于阈值，则标记此时的位置为这一段 DTMF 信号的终点。保留起点和终点的位置后，重复这样的扫描直至音频信号的末尾。相关代码封装为了一个函数，具体如下：

```
function [start, finish] = segment(xn)
% segment: 切分出DTMF信号，返回起点和终点的位置数组
%   xn: 时域信号
threshold = 0.1 * max(xn);
win_len = 100;
frame = 1;

xn_len = length(xn);
start = [];
finish = [];
silence = true;

while frame <= xn_len
    if xn(frame) > threshold && silence
        start = [start, frame];
        silence = false;
    end
    if sum(xn(frame:min(xn_len, frame + win_len)) > threshold) == 0 && ~
        silence
        finish = [finish, frame];
        silence = true;
```

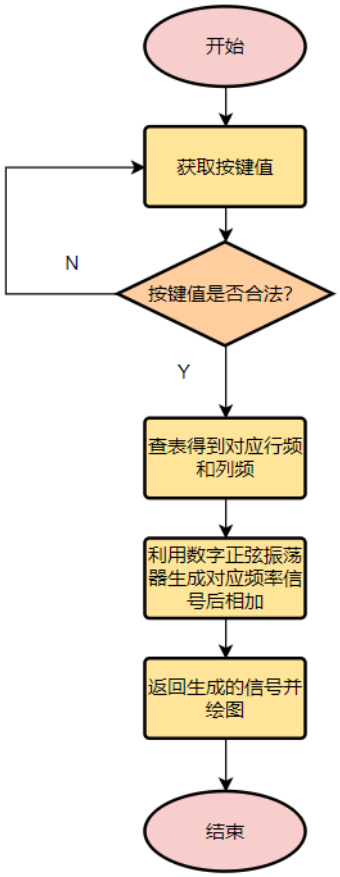
```
        end
        frame = frame + 1;
    end
    if length(start) - length(finish) == 1
        finish = [finish, xn_len];
    end

    end
```

在有了分段结果之后，将各分段 DTMF 信号分别取出送入到 fft 或者 goertzel 方法中得到按键值，从而可获得长音频中的全部 DTMF 信号对应的按键值串。

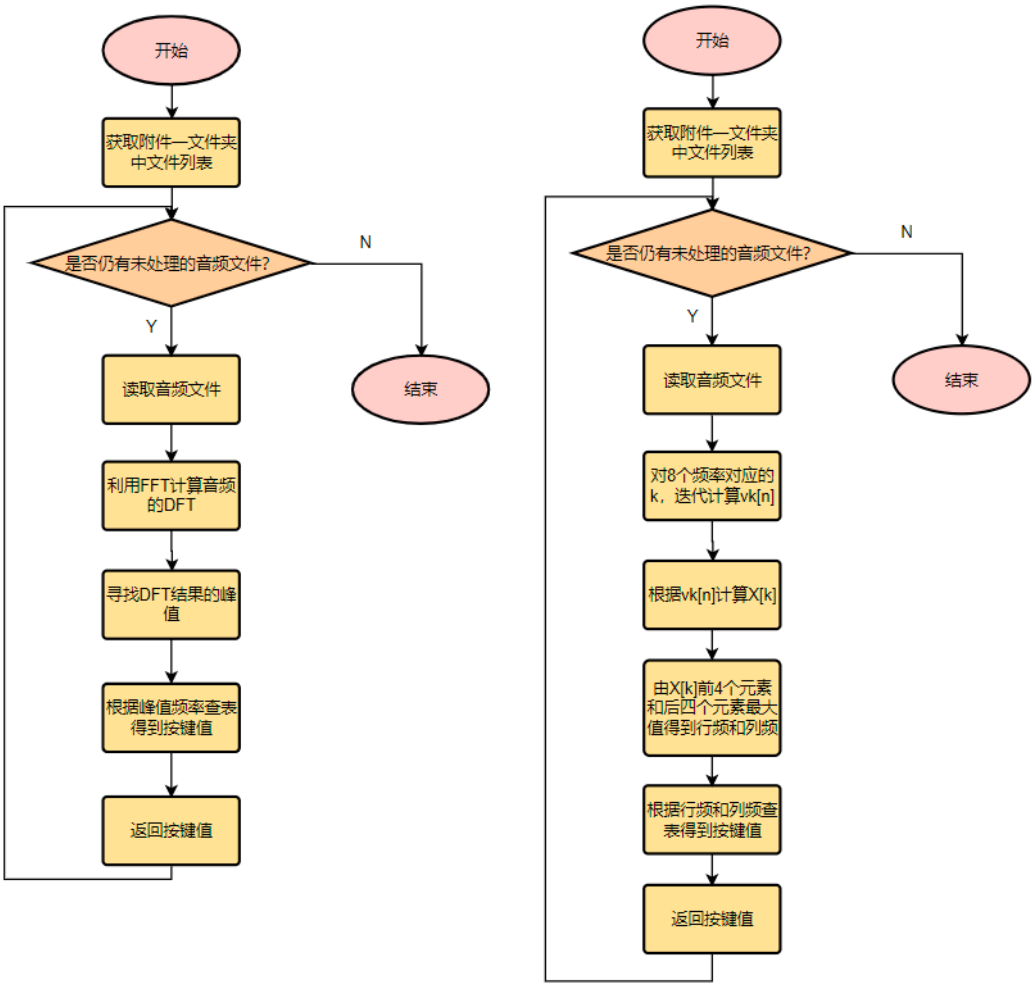
2 程序流程图

2.1 DTMF 信号产生

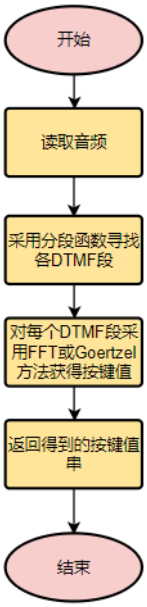


2.2 DTMF 信号检测识别

左侧为 FFT 方法，右侧为 Goertzel 算法


开始
获取附件一文件夹中文件列表
是否仍有未处理的音频文件?
N
Y
读取音频文件
结束
利用FFT计算音频的DFT
寻找DFT结果的峰值
根据峰值频率查表得到按键值
返回按键值
开始
获取附件一文件夹中文件列表
是否仍有未处理的音频文件?
N
Y
读取音频文件
结束
对8个频率对应的k，迭代计算vk[n]
根据vk[n]计算X[k]
由X[k]前4个元素和后四个元素最大值得到行频和列频
根据行频和列频查表得到按键值
返回按键值

2.3 长音频检测

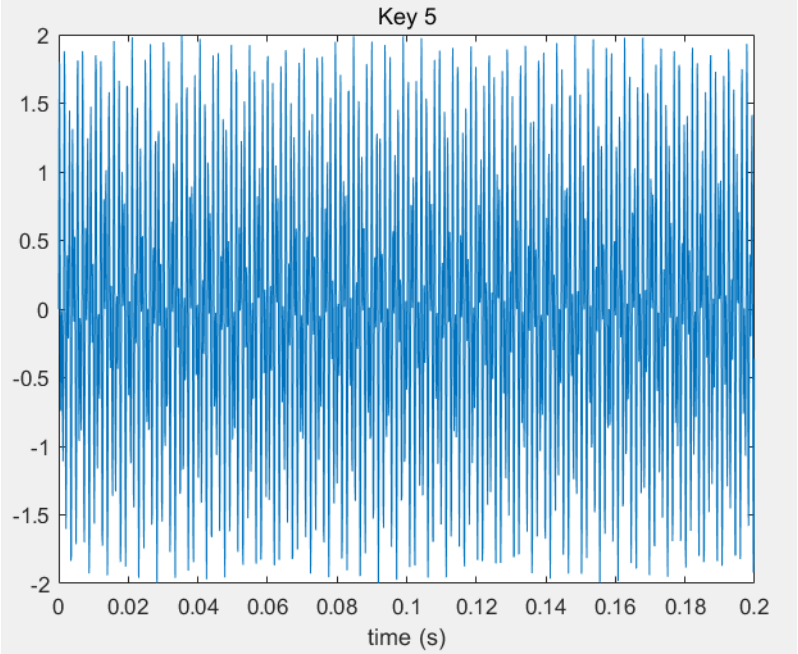

开始
读取音频
采用分段函数寻找各DTMF段
对每个DTMF段采用FFT或Goertzel方法获得按键值
返回得到的按键值串
结束

3 结果与分析

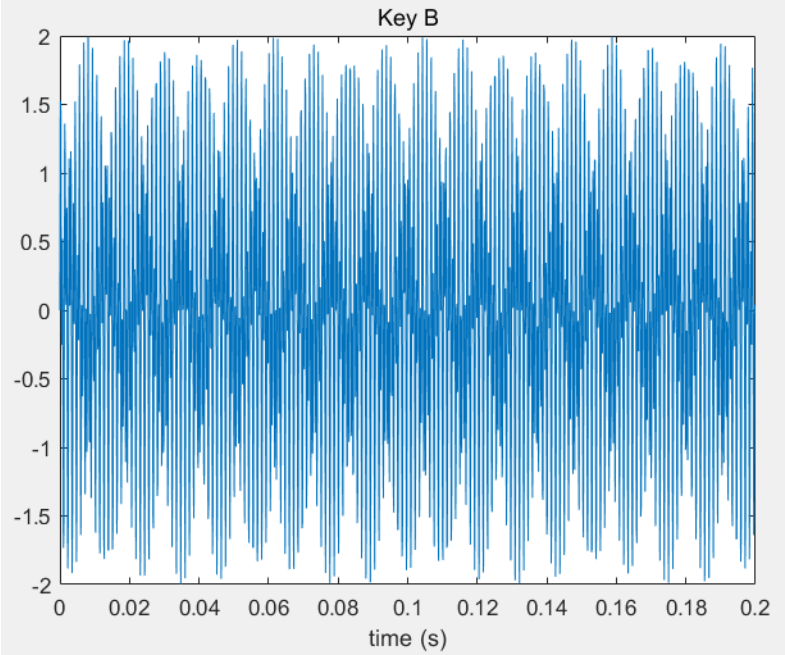
3.1 DTMF 信号产生

采用时长 t=0.2s，分别键入不同的按键，得到的结果图像如下：

按键值为 5 时，输出 DTMF 信号为：



按键值为 B 时，输出 DTMF 信号为：



3.2 DTMF 信号检测识别

3.2.1 FFT 方法

```
>> dtfm2_1
Use FFT to detect:
File Name: data1081.wav  Key: 5
File Name: data1107.wav  Key: 1
File Name: data1140.wav  Key: 6
File Name: data1219.wav  Key: 9
File Name: data1234.wav  Key: 8
File Name: data1489.wav  Key: 7
File Name: data1507.wav  Key: 3
File Name: data1611.wav  Key: 4
File Name: data1942.wav  Key: 0
File Name: data1944.wav  Key: 2
>>
```

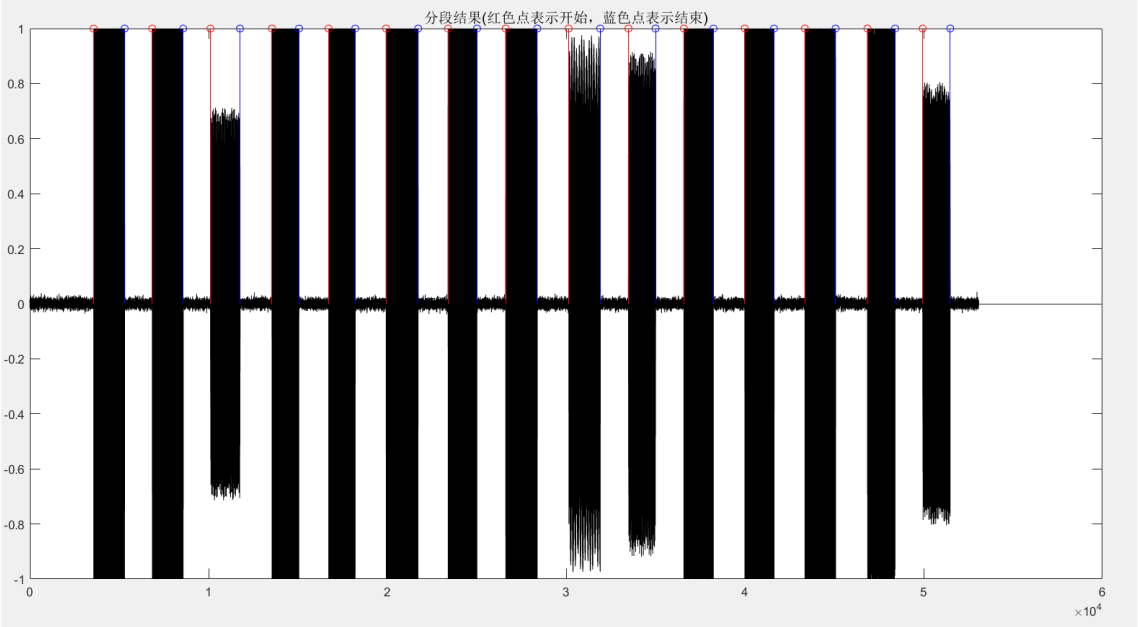
3.2.2 Goertzel 算法

```
>> dtfm2_2
Use Goertzel to detect:
File Name: data1081.wav  Key: 5
File Name: data1107.wav  Key: 1
File Name: data1140.wav  Key: 6
File Name: data1219.wav  Key: 9
File Name: data1234.wav  Key: 8
File Name: data1489.wav  Key: 7
File Name: data1507.wav  Key: 3
File Name: data1611.wav  Key: 4
File Name: data1942.wav  Key: 0
File Name: data1944.wav  Key: 2
>>
```

可以看到两种方法得到的结果是一致的。

3.3 长音频检测

下图为分段函数探测各 DTMF 段的结果 (其中，红色表示 DTMF 段的开始，蓝色表示 DTMF 段的结束):



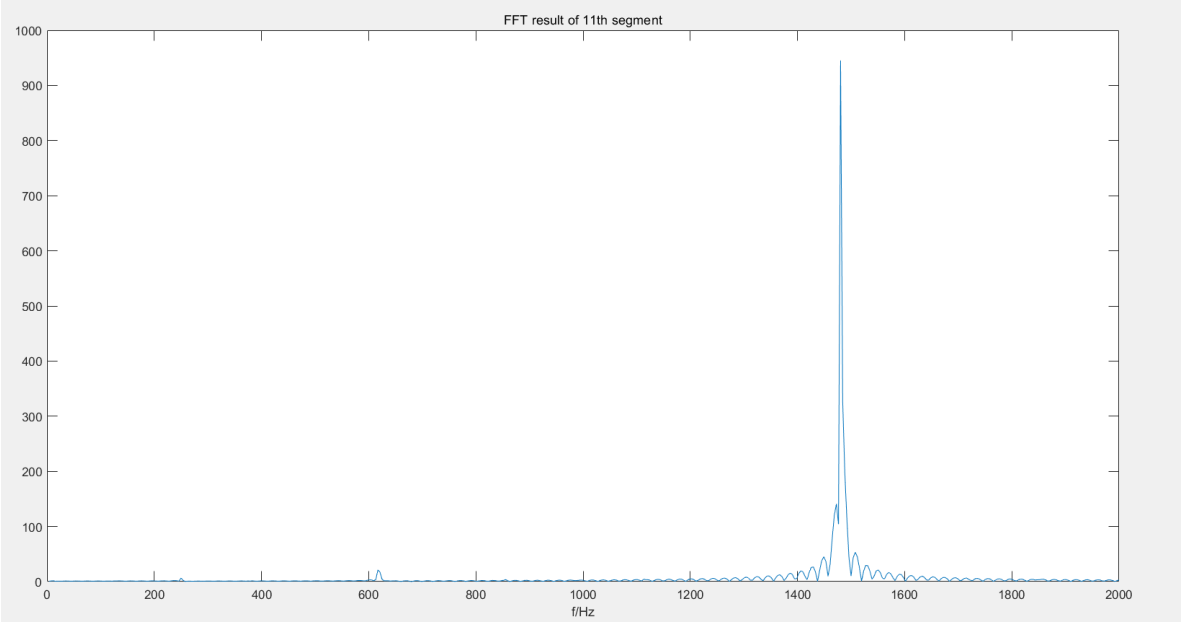
下图为采用两种方法对长音频各 DTMF 段检测结果所得的按键值串（X 表示错误）：

```
>> dtfm2_3
FFT result:       2058911320X4649
Goertzel result: 205891132094649
>>
```

可以看到，除去倒数第 5 处外，两种方法得到的结果是一致的。

在 FFT 方法的检测中，将此处识别为错误，经过我进一步的检查，我发现，此处的音频可能本身就存在错误。

将这一段音频信号做 FFT 的结果取出来，如下图所示：



可以看到，只在 1477Hz 前后处，有一个单峰，其他频率点基本上都很小，610Hz 处也有一个非常小的峰，但是此处并非有效频率，由此可以认为这段音频信号包含不正确的频率组合。

3.4 两种方法识别性能对比分析

主要从计算复杂度和识别鲁棒性这两个角度来进行对比分析。

3.4.1 计算复杂度

1. 基于 FFT 的识别算法

   对于一个长度为 N 的片段（忽略补零），采用基 2-FFT 算法的时间复杂度为 $O(N\log N)$，具体而言需要 $2N\log_2 N$ 次实乘和 $3N\log_2 N$ 次实加。

   执行完 DFT 后需要寻找峰值，该操作的时间复杂度为 $O(N)$。

   故而总体的时间复杂度为 $O(N\log N)$。

2. 基于 Goertzel 的识别算法

   对于一个长度为 N 的片段，由于只关心 8 个频点，时间复杂度为 $O(N)$，而此后只需在 8 个频点中去寻找低频和高频区的最大值，该部分为常数时间复杂度，故而总体的时间复杂度为 $O(N)$。

由此可见，在计算时间复杂度上，基于 Goertzel 的识别算法是要明显优于基于 FFT 的识别算法的。并且实际在 Matlab 中运行并计时也可以看到基于 Goertzel 的识别算法是要明显快于基于 FFT 的识别算法的。

3.4.2 鲁棒性

根据之前的长音频检测过程和结果，我们可以看到，对于其中的一个错误的 DTMF 段，FFT 轻松的发现了错误存在，而 Goertzel 则由于算法自身设计的原因，依然给出了一个结果。由此可见，基于 FFT 的识别方法虽然计算复杂度较高，但是其可以更好的应对各种错误情况。

4 总结

本次课程设计中，我除了进一步熟练使用 FFT 这一工具进行信号分析之外，还学习了 Goertzel 这一种全新的算法。这种算法可以说是利用了更多的条件约束来达到更低的时间复杂度。对于长音频信号的分析中比较核心的分割问题，我也尝试了好几种方法，最后根据长音频信号的特点选择了现在所使用的方法，最后实现效果很好。这一次课程设计过程中，我除了巩固了已经学习的知识之外，也更加深切地感悟到了 DSP 理论应用于实际时的强大威力，可以说是收获满满。

5 文件清单

文件名称	说明
dtfm2_1.m	采用 FFT 进行 10 个文件检测的主程序
dtfm2_2.m	采用 Goertzel 进行 10 个文件检测的主程序
dtfm2_3.m	采用两种方法检测识别长音频文件的主程序
dtmf1.m	产生 DTMF 信号的的主程序
find_key_fft.m	根据音频信号采用 FFT 检测对应按键值的函数
find_key_goertzel.m	根据音频信号采用 Goertzel 检测对应按键值的函数
generate_dtmf.m	产生 DTMF 信号的函数

备注：需将存放音频数据的附件 *1* 和附件 *2* 这两个文件夹放置在此根目录中方可运行