DTMF 的检测与识别

2018010811 罗一夫 生医 81

一、 课程设计内容

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 个文件所对应的真实数字。

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

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。

二、 程序设计思路与流程图

首先，我先对一些共通的算法和流程进行简略的说明。

无论是 FFT 算法，亦或 Goertzel 算法，均先需要对音频文件进行 MATLAB 处理，使其转换为显示时域幅度的文本文件。本部分程序文件为 wav_to_txt.m，输出文件为“txt 附件 1”和“txt 附件 2”的的 txt 文件，这也是唯一使用 MATLAB 的地方。代码主要如下：

```
close all;clc;clear;

%% Data1
dirpath = './附件1/';
writepath = './txt附件1/';
d = dir( dirpath );
for i = 1 : length( d )
    if( ~isequal( d( i ).name, '.' ) &~isequal( d( i ).name, '..') )
        datpath = [dirpath d( i ).name];
        [y, Fs] = audioread(datpath);
        f = fopen([writepath d( i ).name(1:8) '.txt'],'w');
        fprintf(f, '%f\n', y);
        fclose(f);
    end
end

%% Data2
datpath = './附件2/data.wav';
[y, Fs] = audioread(datpath);
f = fopen('./txt附件2/data.txt','w');
fprintf(f, '%f\n', y);
fclose(f);
```

在转换为 txt 文件后，我们在 C++中还需要对文件进行读入操作，也就是对 X【n】序列进行赋值。值得注意的是，这里还存在着一个补零的问题。为了解决这一问题，以第一 FFT 算法为例，我设置了一个 audio 数组，先读取文件对 audio 数组赋值，而之后的 X【n】序列则是在补零后根据 audio 赋值即可。

对 audio 数组的赋值属于基本文件操作，代码如下图。

```
std::ifstream fstr(fileName.c_str());
std::string lineStr;
std::stringstream sstr;
std::string seg_path;
double data;
memset(&audio, 0, sizeof(audio));

while (std::getline(fstr, lineStr))
{
    audio[j] = atof(lineStr.c_str());
    sstr >> seg_path >> data;
    sstr.clear();
    num++;
    j++;
}
```

而补零的操作也很简单，就是一个找到最短 L 的过程，代码如下图。

```
// 补零
int addzero = 0;
for (j = 1; j < 20; j++)
{
    if (pow(2, j) < num)
    {
        continue;
    }
    else if (pow(2, j) == num)
    {
        break;
    }
    else
    {
        addzero = pow(2, j) - num;
        break;
    }
}
L = num + addzero;
```

最后，无论是 FFT 算法，亦或 Goertzel 算法，均需要根据频谱信息来识别数字，这里的频谱信息主要为频谱的峰值。为此，首先需要寻找频谱的峰值位置，然后根据峰值信息来判断数字。为此，我写了一个 find_s 函数，作用是根据频谱峰值 a，b 来识别数字。为了方便，我并没有在文件清单中单列出这个函数，而是让这个函数作为全局函数插入到每一问的代码中。

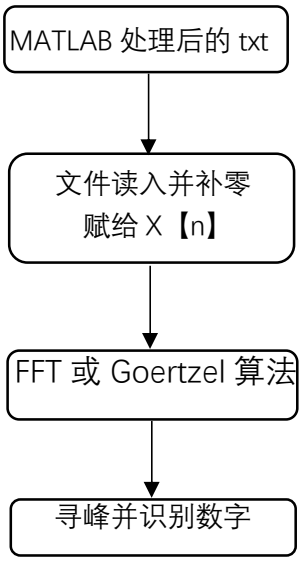
首先给出 find_s 代码如下：

```
char find_s(double a, double b)
{
    double freqs1[] = { 697,770,852,941 };
    double freqs2[] = { 1209,1336,1477,1633 };
    char symbol1[] = { '1','2','3','A','4','5','6','B','7','8','9','C','*','0','#','D' };

    if (a > b)
    {
        int temp = a;
        a = b;
        b = temp;
    }

    int x = 100;
    int y = 100;
    for (int i = 0; i < 4; i++)
    {
        if (abs(freqs1[i] - a) <= 10)
        {
            x = i;
            break;
        }
    }
    for (int i = 0; i < 4; i++)
    {
        if (abs(freqs2[i] - b) <= 10)
        {
            y = i;
            break;
        }
    }

    if (x == 100 || y == 100)
    {
        char error_char = 'x';
        return error_char;
    }

    char symbol = symbol1[x * 4 + y];
    return symbol;
}
```

接下来，还是以第一问为例，给出我的寻峰并调用 find_s 识别数字的代码：

```
//寻峰，识别
for (j = 0; j < L / 2; j++)
{
    am = pow(output[j].real(), 2) + pow(output[j].imag(), 2);
    if (am > max2)
    {
        if (am > max1)
        {
            max2 = max1;
            pos2 = pos1;
            max1 = am;
            pos1 = j;
        }
        else
        {
            max2 = am;
            pos2 = j;
        }
    }
}
double step = double(fs) / (L - 1);
double* x_axis = new double[L];
for (j = 0; j < L; j++)
{
    x_axis[j] = j * step;
}
output_symbol = find_s(x_axis[pos1], x_axis[pos2]);
```

以上就是本次作业几问的通用部分，总结一下就是下图的流程。

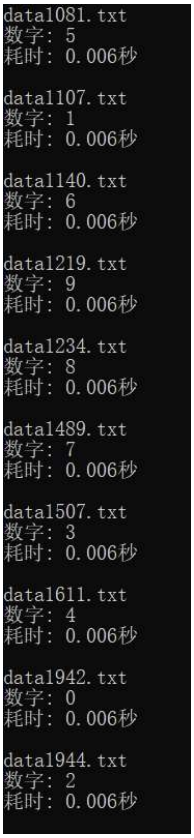


再分析完以上通用部分后，下面我们逐问进行算法的细节分析。

1、 利用 FFT 分析 DTMF 信号

因为上一次作业已经写好了 FFT 算法，因此这一问比较简单，只需要把上一次的 FFT 算法（我这里选用的是 DIT）和我上面讲的几个点（文件读入、数字识别）结合起来就可以了，因此不再赘述。值得注意的是，在上一次作业编写 FFT 时我没有注意到 C++本身就提供 complex 库，因此还自己编写了一个复数类 Complex。为了程序的顺承性和简便性，在这一问中我既在头文件中包含了 C++的标准 complex 库，也在全局定义里使用了上次编写的复数类 Complex。另外，在程序运行时，也需要将

本问对应的代码文件为 DTMF-FFT.cpp，程序运行结果截图如下：



可见几个数字依次为：5、1、6、9、8、7、3、4、0、2

2、 Goertzel 算法

本部分唯一的区别就在于将上一问中使用的 FFT 算法变为了 Goertzel 算法。在这里，我们主要利用的公式是

$$X[k] = v_k[n-1] - W_N^k v_k[n-2]$$

$$v_k[n] = x[n] + 2\cos(w_k)\, v_k[n-1] - v_k[n-2]$$

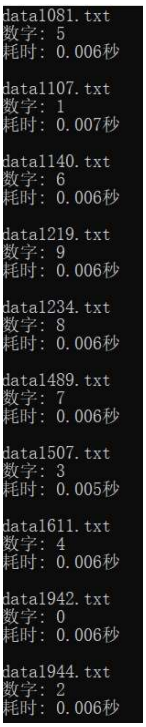
$$v_k[-2] = v_k[-1] = 0, v_k[0] = x_k[0]$$

$$k = N * \frac{f}{fs}, fs = 8000Hz$$

特别的，由于我们已经知道了目标频率（即程序中的 fre 数组那 8 个频率），因此我们可以由此推出 k，再根据递推关系式推出 v_k，最后推出 X_k。下面附上我编写的 Goertzel 函数。

```cpp
double* Goertzel(complex<double> input[], int L)
{
    complex<double> unit;
    complex<double>* WN = new complex<double>[L];
    complex<double>* v = new complex<double>[L];
    complex<double> target[8];
    double* am = new double[8];
    double wk;
    int i, j, k;

    unit.real(cos(2 * PI / L));
    unit.imag(-sin(2 * PI / L));
    WN[0] = 1;
    for (i = 1; i < L; i++)
        WN[i] = WN[i - 1] * unit;

    for (i = 0; i < 8; i++)
    {
        k = L * fre[i] / fs;
        wk = 2 * PI * fre[i] / fs;

        v[0] = input[0];
        v[1] = input[1] + 2 * cos(wk) * input[0];
        for (j = 2; j < L; j++)
            v[j] = (input[j] + (2 * cos(wk) * v[j - 1])) - (v[j - 2]);
        target[i] = (v[L - 1] - ((WN[k] * v[L - 2])));
    }

    for (i = 0; i < 8; i++)
        am[i] = pow(target[i].real(), 2) + pow(target[i].imag(), 2);

    return am;
}
```

其余解答部分基本与上一问一致，本问对应的代码文件为 DTMF-G.cpp，程序运行后截图如下：



可见 Goertzel 算法给出的结果与 FFT 一样。

3、 处理长音频文件

本问我采用的是 Goertzel 算法，基本思路与上一问相同，只不过是事先通过 MATLAB 找到音频的间断点，从而分别识别每段的数字。分段赋值的代码如下：

```
begin = stops[i * 2] / seconds * len;
end = stops[i * 2 + 1] / seconds * len;
num = end - begin + 1;
double* audio1 = new double[num];

for (j = 0; j < num; j++)
    audio1[j] = audio[j + begin];
```

本问对应的代码文件为 longaudio.cpp，程序运行截图如下：

其中 x 表明无法识别。

三、 时间复杂度分析

在上一次作业中已经分析过 FFT 的时间复杂度为 $O(N\log_2 N)$，而对于 Goertzel 算法，由于只需要关心 8 个频点，因此 16N 次实乘，16N 次实加，8 次复乘，8 次复加，因此为 O(N)。

可见 Goertzel 算法在时间上是优于 FFT 的。