DTMF 信号的检测与识别

邓博文

2018 年 12 月 9 日

1 定义按键与频率对

key.h

```
#ifndef KEY_H_
#define KEY_H_

#define SAMPLE_RATE 8000
typedef struct
{
        unsigned int row_freq;
        unsigned int col_freq;
} freq_pair;

static char key_name[] = { '1', '2', '3', 'A', '4', '5', '6', 'B','7', '8',
                                        '9', 'C', '*', '0', '#', 'D' };

static freq_pair key_freq[] = {{697, 1209}, {697, 1336}, {697, 1477}, {697, 1633},
                               {770, 1209}, {770, 1336}, {770, 1477}, {770, 1633},
                               {852, 1209}, {852, 1336}, {852, 1477}, {852, 1633},
                               {941, 1209}, {941, 1336}, {941, 1477}, {941, 1633}};

#endif
```

为了后续程序处理方便，将已经确定的（行频率，列频率）以及对应的按键名称按照图中所示定义好。

2 FFT 进行 DTMF 信号分析

此部分整体流程图见图1。

图 1: FFT 部分的流程图

这一部分使用第一次课程设计中编写的 FFT 程序，因为详细的思路已在上次实验报告中进行了说明，故不再赘述。这里只说明不同的部分。

2.1 读入 wav 文件

通过上网查阅相关资料，了解到 wav 文件中从第 40 个字节开始 4 个字节是采样的总数据量，单位为字节，又因为每一个采样点的数据量为 2 字节，由此可以算出总的采样点的数目。此外，还对输入数据进行了归一化操作，使得其值位于 [-1, 1]。为了套用上一次课程设计中的 FFT 算法，如果总的采样点数不是 2 的整数次幂，则需要进行补零。与上一次课程设计中相同，将读入操作放在 FFT 类的构造函数中完成。此部分代码见图2。

```
FFT(std::string filename)
{
    std::ifstream in(filename, std::ios::binary);
    if (!in.is_open())
        error("Cannot open data file!");
    else
    {
        this->filename = filename;
        unsigned bytes, dots;
        in.seekg(40);
        in.read(reinterpret_cast<char *> (&bytes), sizeof(unsigned int));
        dots = bytes / sizeof(short);
        // padding
        N = 1 << static_cast<unsigned> (ceil(log2(1.0 * dots)));
        if (N == 0 || (N & (N - 1)))
            error("N is not a power of two!");
        x.resize(N);
        for (unsigned int i = 0; i < N; ++i)
        {
            // reverse order for DIT-FFT
            unsigned int t = getReverse(i);
            short data;
            in.read(reinterpret_cast<char *> (&data), sizeof(short));
            // normalize
            x[t] = (1.0 * data - SHRT_MIN) / (SHRT_MAX - SHRT_MIN) - 0.5;
        }
    }
    in.close();
}
```

图 2: FFT 算法读入 wav 文件

2.2 输出结果

```
FFT(std::string filename)
{
    std::ifstream in(filename, std::ios::binary);
    if (!in.is_open())
        error("Cannot open data file!");
    else
    {
        this->filename = filename;
        unsigned bytes, dots;
        in.seekg(40);
        in.read(reinterpret_cast<char *> (&bytes), sizeof(unsigned int));
        dots = bytes / sizeof(short);
        // padding
        N = 1 << static_cast<unsigned> (ceil(log2(1.0 * dots)));
        if (N == 0 || (N & (N - 1)))
            error("N is not a power of two!");
        x.resize(N);
        for (unsigned int i = 0; i < N; ++i)
        {
            // reverse order for DIT-FFT
            unsigned int t = getReverse(i);
            short data;
            in.read(reinterpret_cast<char *> (&data), sizeof(short));
            // normalize
            x[t] = (1.0 * data - SHRT_MIN) / (SHRT_MAX - SHRT_MIN) - 0.5;
        }
    }
    in.close();
}
```

图 3: 输出音频文件对应的数字

输出时只需要比较信号进行 FFT 变换后，在各个按键对应的（行频率，列频率）处的幅度值之和即可，取最大值时就是该音频文件实际对应的数字。这部分代码见图3。

3 Goertzel 进行 DTMF 信号分析

此部分对应的流程图见图4。

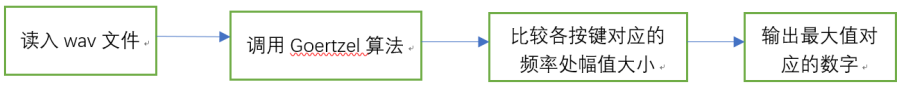

图 4: Goertzel 部分的流程图

3.1 读入 wav 文件

此部分代码与 FFT 相应部分几乎相同，只是为了后面使用 Goertzel 算法进行长音频的识别，在构造函数中添加了一个标志变量 need_pad，表明是否需要对原始数据进行填充。此部分代码对应图5。

```
DTMF(std::string filename, bool need_pad)
{
    std::ifstream in(filename, std::ios::binary);
    if (!in.is_open())
        error("Cannot open data file!");
    else
    {
        this->filename = filename;
        unsigned bytes, dots;
        in.seekg(40);
        in.read(reinterpret_cast<char *> (&bytes), sizeof(unsigned int));
        dots = bytes / sizeof(short);
        // padding according to the flag
        N = (need_pad)? 1 << static_cast<unsigned> (ceil(log2(1.0 * dots))): dots;
        x.resize(N);
        for (unsigned int i = 0; i < dots; ++i)
        {
            short data;
            in.read(reinterpret_cast<char *> (&data), sizeof(short));
            // normalize
            x[i] = (1.0 * data - SHRT_MIN) / (SHRT_MAX - SHRT_MIN) - 0.5;
        }
    }
}
```

图 5: Goertzel 算法读入 wav 文件

3.2 Goertzel 算法的实现

```
double Goertzel(unsigned int start, unsigned int k, unsigned int N)
{
    // cosine factor
    double w = cos(2 * M_PI * k / N);
    double v[2];
    v[0] = x[start + 1] + 2.0 * w * x[start + 0];
    v[1] = x[start + 2] + 2.0 * w * v[0];
    unsigned int i = 2;

    for (; i < N; ++i)
        v[i % 2] = x[start + i + 1] + 2.0 * w * v[(i - 1) % 2] - v[(i - 2) % 2];
    return sqrt((v[i % 2] - v[(i - 1) % 2] * w) *
                (v[i % 2] - v[(i - 1) % 2] * w) +
                (v[(i - 1) % 2] * sin(2 * M_PI * k / N)) *
                (v[(i - 1) % 2] * sin(2 * M_PI * k / N)));
}
```

图 6: Goertzel 算法的实现

实现 Goertzel 算法只需要根据实验说明中给出的递推式逐项计算即可，比较直接，无需特殊说明。开始时初始化序列 v[n] 的前两项，最后返回对应频点处的幅度值。为了方便后续进行长序列识别，函数的接口中三个参数依次为序列 x[n] 开始处的偏移量，频率 k，音频长度 N。此部分代码见图6。算法流程图见图7。

图 7: Goertzel 算法的流程图

4 长音频的识别

此部分的流程图见图8。

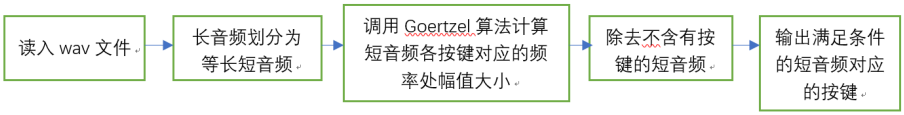

图 8: 长音频部分的流程图

4.1 识别含有按键音的片段

在长音频的识别中，因为其中包含有一串 DTMF 信号，所以一个重要的问题是判断各个 DTMF 信号的开始与结束。这里使用了一种比较简单的方法，将长音频分隔为等长的短音频段，如果某一短音频段中包含有按键，那么它在 16 个按键中的某一个按键对应的频率处的幅度值必然足够大，如果不包含有按键，那么它在 16 个按键中任意一个对应的频率处的幅度值都会比较小。短音频的长度（通过人耳听长音频文件，可以发现共包含有 15 个按键音），以及阈值的大小都是通过实验得出的，实现的算法最终可以识别出 13 个按键音。此部分代码见图9。

```
// Goertzel for each interval
for (unsigned int k = 0; k < N / length; ++k)
{
    unsigned int max_index = 0;
    double max = 0.0;
    // try Goertzel for every frequency pair of a certain key
    for (unsigned int i = 0; i < 16; ++i)
    {
        unsigned int f1 = key_freq[i].row_freq * length / SAMPLE_RATE;
        unsigned int f2 = key_freq[i].col_freq * length / SAMPLE_RATE;
        double amp = Goertzel(k * length, f1, length) +
                     Goertzel(k * length, f2, length);
        if (amp > max)
        {
            max = amp;
            max_index = i;
        }
    }

    // interval contains a key must have an amplitude big enough
    // at certain frequecny pair corresponding to some key
    if (max > 10.0)
        keys[k] = key_name[max_index];
```

图 9: 识别含有按键音的片段

4.2 结果的输出

```
bool begin = false;
for (unsigned i = 1; i < N / length; ++i)
{
    // prior interval doesn't contain a key, and current does
    if (begin == false && !keys[i - 1] && keys[i])
        begin = true;

    // interval does contain a key only when the key
    // can be detected in the current interval and its next one
    if (begin && keys[i] == keys[i + 1])
    {
        std::cout << "Key detected:" << keys[i] << std::endl;
    }
    begin = false;
}
```

图 10: 长音频序列识别结果的输出

在前面步骤的基础上，输出时要进行判断。一个短音频确实包含一个按键当且仅当满足下列条件：当前短音频段之前的短音频段不包含按键音，并

且下一短音频段中包含同样的按键音。这样的判决方法可以有效地减少误报。此部分代码见图10。

5 主程序与测试批处理文件

5.1 主程序

主程序部分从命令行读入参数，其中，最后一个命令行参数应为长音频序列的文件名，其余应为前两问的 10 个短音频序列的文件名。这部分代码见图11。

```
int main(int argc, char **argv)
{
    // question(1)
    std::cout << "FFT:" << std::endl;
    for (int i = 1; i < argc - 1; ++i)
    {
        FFT f(argv[i]);
        f.dit();
        f.output();
    }

    // question(2)
    std::cout << "Goertzel:" << std::endl;
    for (int i = 1; i < argc - 1; ++i)
    {
        DTMF d(argv[i], true);
        d.output();
    }

    // question(3)
    DTMF d(argv[argc - 1], false);
    d.detect();

    return 0;
}
```

图 11: 主程序

5.2 测试批处理文件

为了便于对结果进行测试，编写了批处理程序，其中已经提供了主程序所需的所有命令行参数。注意，需要所有音频文件与主程序在同一文件夹下。批处理文件如下。

test.bat

```
@echo off
call main.exe data1081.wav data1107.wav data1140.wav data1219.wav data1234.wav
 ↪  data1489.wav data1507.wav data1611.wav data1942.wav data1944.wav data.wav
pause
```

得到运行结果见图12。

```
FFT:
data1081.wav: The actual number of this file is 5
data1107.wav: The actual number of this file is 1
data1140.wav: The actual number of this file is 6
data1219.wav: The actual number of this file is 9
data1234.wav: The actual number of this file is 8
data1489.wav: The actual number of this file is 7
data1507.wav: The actual number of this file is 3
data1611.wav: The actual number of this file is 4
data1942.wav: The actual number of this file is 0
data1944.wav: The actual number of this file is 2
Goertzel:
data1081.wav: The actual number of this file is 5
data1107.wav: The actual number of this file is 1
data1140.wav: The actual number of this file is 6
data1219.wav: The actual number of this file is 9
data1234.wav: The actual number of this file is 8
data1489.wav: The actual number of this file is 7
data1507.wav: The actual number of this file is 3
data1611.wav: The actual number of this file is 4
data1942.wav: The actual number of this file is 0
data1944.wav: The actual number of this file is 2
Use Goertzel to detect data.wav:
Key detected:2
Key detected:0
Key detected:5
Key detected:8
Key detected:9
Key detected:1
Key detected:3
Key detected:2
Key detected:0
Key detected:4
Key detected:6
Key detected:4
Key detected:9
请按任意键继续. . .
```

图 12: 运行结果

5.3 Makefile

为了更好地管理代码，书写了一个比较简单的 Makefile 文件。内容如下：

Makefile

```
main.exe: fft.h dtmf.cpp
	g++ -Wall -Og -o main.exe dtmf.cpp
clean:
	del *.exe
```

6 复杂度比较

6.1 FFT 算法

设序列长度为 N，FFT 算法中乘法操作的时间复杂度为 $O(N \log N)$，寻找幅度最大值需要常数时间复杂度，总的时间复杂度为 $O(N \log N)$。

6.2 Goertzel 算法

设序列长度为 N,Goertzel 算法中计算一个频率所需的乘法操作的时间复杂度为 $O(N)$。

6.3 比较

当 N 比较大时，Goertzel 算法的时间复杂度性能优于 FFT 算法。

7 总结

在这次实验中，我亲自动手实现了 Goertzel 算法，并将它和 FFT 算法运用于 DTMF 信号的检测与识别中。这个过程加深了自己对于相关知识的理解，更重要的是，让自己从应用的角度看到了这些经典算法的价值，收获很大。