实验报告

计76　张翔　 2017011568

2019 年 12 月 8 日

1　双音频按键识别

1.1　算法

这里实现了两种识别DTMF频谱的方法，分别为通过FFT变换后寻找谱峰、通过Goertzel算法直接找到频率峰值的方法，代码在`fft_dtmf.m`与`goertzel.m`中，通过`main.m`运行。

在FFT方法中，先取得音频FFT变换后的序列，然后从按键对应频率区间内寻找峰值，并将其匹配到最靠近的按键频率上；在Goertzel方法中，按键对应频率的能量直接被取得，能量最大的频率所对应的按键即最终结果。

1.2　实验结果

1.2.1　单按键合成音频测试

这部分的数据是合成的0-9按键音频，没有明显的噪声，结果如下：

按键	长度	FFT耗时(ms)	Goertzel耗时(ms)
0	4687	2.299000	0.814000
1	4049	1.188000	0.375000
2	3731	0.891000	0.239000
3	5254	0.691000	0.258000
4	5326	0.775000	0.362000
5	4523	0.494000	0.143000
6	5613	0.492000	0.156000
7	5071	0.458000	0.155000
8	4918	0.450000	0.181000
9	4658	0.466000	0.132000

表 1: 单按键测试结果（两种算法检测结果均与按键一致）

从表(1)中可以看出，两种算法在无噪声的情况下精度一致，所有情况下Goertzel算法速度均比FFT快。

1.2.2　连续合成音频测试

这部分同上，只不过换成了连续合成音频，其中按键共10次被按下，结果为

```
Another synthesized sample:
Result: 4074152685
```

两种算法输出均正确（Result相应位只有在两种算法输出相同时才不为”?”）

1.2.3 实际音频测试

这里使用几年前记者采访360老总周鸿祎的音频，截取了其中含有拨号音的部分，源音频噪声较大，这里使用了一些处理技巧，没有对音频降噪而直接输出结果：

```
Zhou Hongwei's phone number[FFT] is 370119110938
Original seq: ?2?11132?4?3333333333?3?1?1?77778?1?2?10000?20?1?1?1111111?...
Zhou Hongwei's phone number[Goertzel] is 13701191098
Original seq: 362211132?41?4?3333333333?32111?77778219482247?8?2?22?10000...
Zhou Hongwei's phone number[Consensus] is 13701191098
Original seq: 72111132?12?6244?33333333336331?147777831?77?2?8?3?035?34?61...
```

这里的处理方法是取一个合适的段长度（这里取$t = 1000/44100\,\mathrm{s}$），将音频的每一段作为一个单音，输入到对应函数中做按键识别，如果得到数字结果，就将该结果append到序列中，得不到结果时则append一个'?'。

得到临时序列（Original Seq）后，可以设定合适的阈值，当数字连续出现频率超过阈值后，认为该数字是一个合法按键，作为结果输出。这里的Consensus方法是使用两种算法的输出值，若相等，认为识别成功，将结果append到序列中。

可以看到，在噪声较大的情况下，Goertzel的效果好于FFT，它正确地输出了手机号13701191098，因为FFT算法的实现中是寻找频率区间中的峰值，容易受噪声影响；Goertzel算法直接取按键对应频率的能量，在录音频率准确时能正常工作。

2 卷积计算方法的性能比较

2.1 算法与代码

四种卷积方法在`conv_naive.m`，`conv_fft.m`，`conv_overlap_add.m`，`conv_overlap_save.m`中，通过文件名即可看出相应算法类别。`main.m`是程序主入口。

2.2 实验结果

给定的待卷积序列为X和Y，其中Y固定长度，X长度变化范围是[1000, 30000]。这里取Y长度为1000和10两种情况进行测试，后者对应序列长度相差过大的情况（$len(Y) < \log_2 len(X)$）

2.2.1 $len(Y) = 1000$

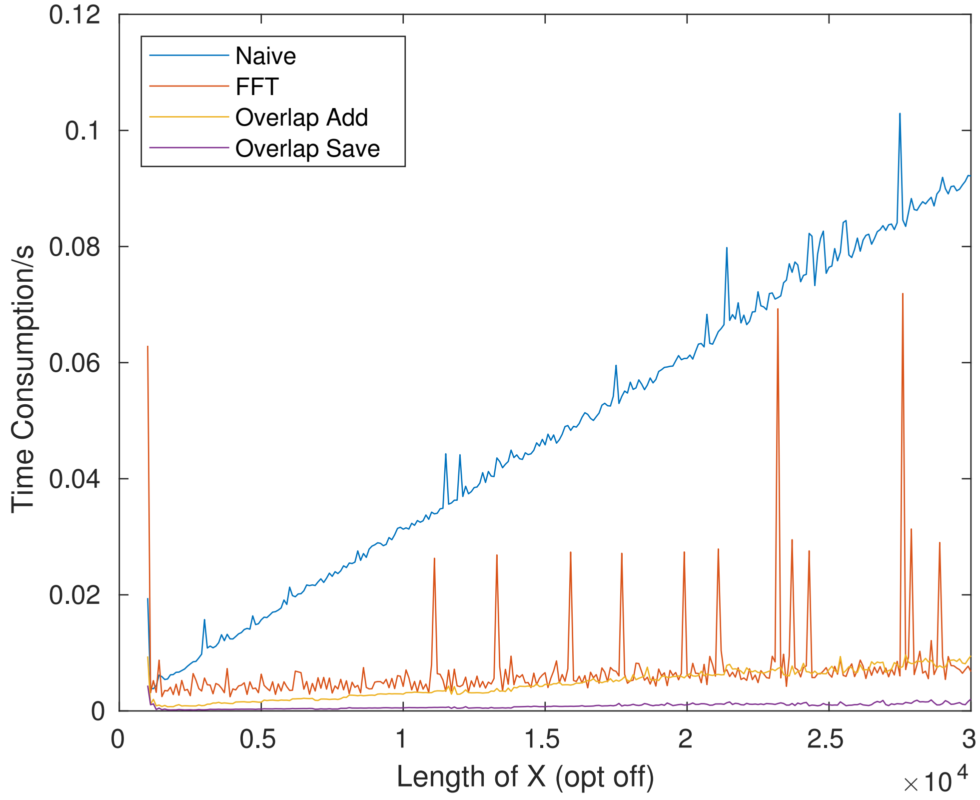


图 1: $len(Y) = 1000$且2^n优化关闭时的结果

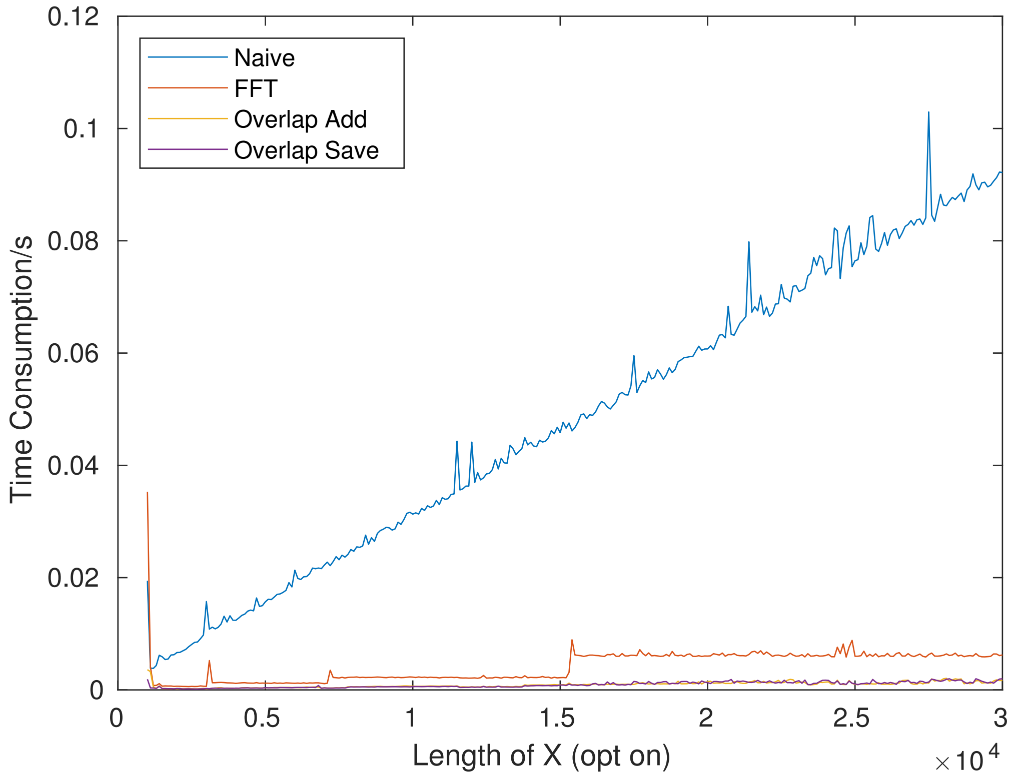


图 2: $len(Y) = 1000$且2^n优化开启时的结果

2.2.2 $len(Y) = 10$

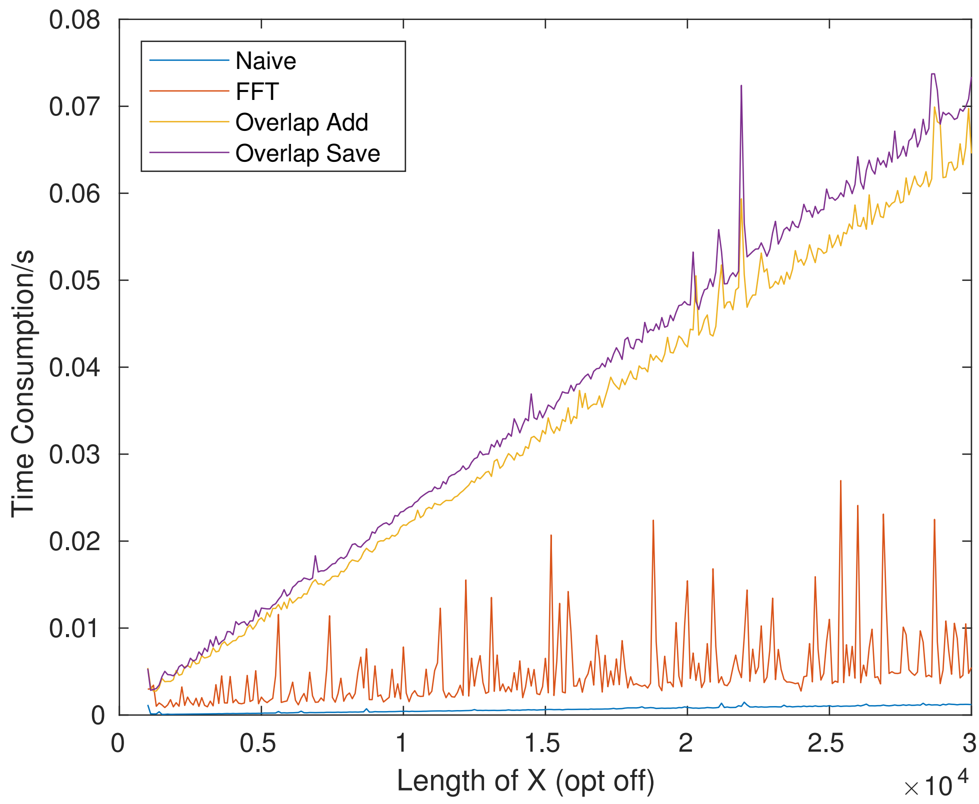


图 3: $len(Y) = 10$且2^n优化关闭时的结果

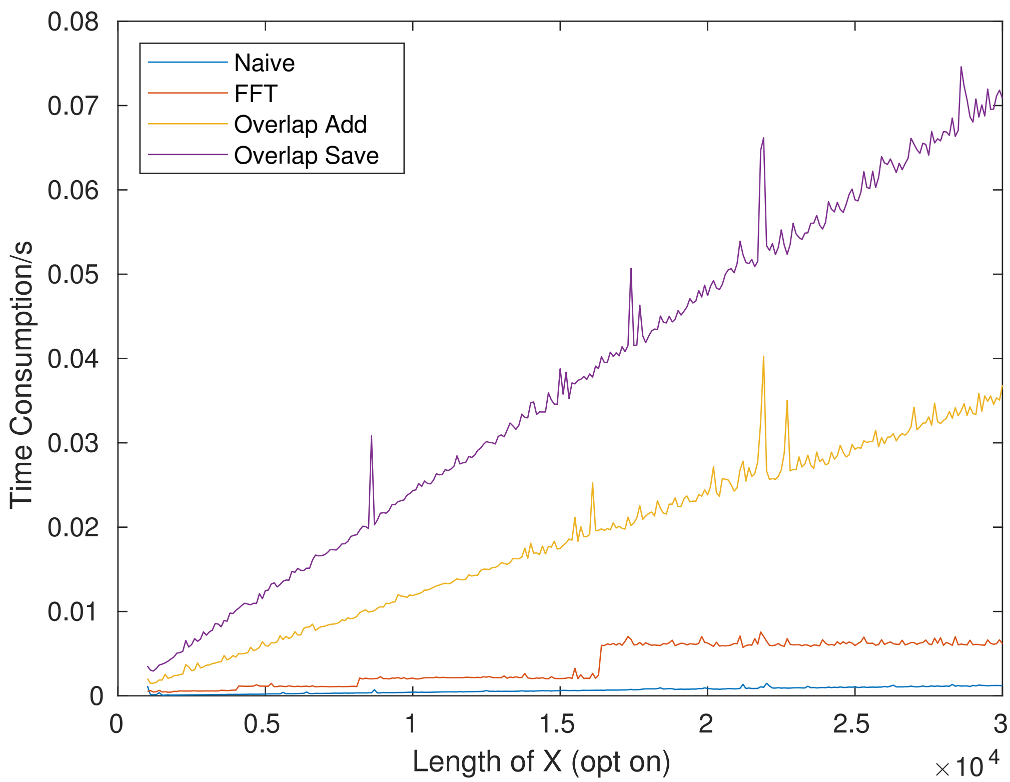


图 4: $len(Y) = 10$且2^n优化开启时的结果

对比图(1)(2)，当序列长度相差不是非常大时（$\log_2 len(X) < len(Y) < len(X)$），公式法最慢，FFT和两种Overlap方法较快，当未开启2^n优化时，Overlap Add性能仅略好于FFT，和Overlap Save相比有较大差距，且此时FFT性能有明显波动；当开启2^n优化时，两种Overlap方法性能几乎一致，均明显好于FFT，且此时FFT性能较为稳定。这里的2^n优化指取2^n作为FFT长度，从而有利于CPU对运算以及访存的优化。

当序列长度相差很大时（$len(Y) < \log_2 len(X)$）时，从图(3)(4)可以看出，此时两种Overlap方法以及FFT性能均不如公式法，开启2^n优化后，Overlap Add性能得到提高，但仍远慢于公式法。

这些结论可以用大O记号重写：当序列长度数量级相差特别大（$len(X) = O(2^{len(Y)})$）时，直接使用公式法较快，而Overlap Save/Add方法在序列长度有一定差异但不是很大（$len(X) = O(len(Y))$）时具有较好的性能。开启2^n优化对于提高性能也有一定的帮助。

3 语音信号的频分复用

3.1 算法流程

1. 输入三段长度相同、采样率（记为f_s）相同的音频信号；
2. 将信号做FFT变换（点数为$f_s \cdot L$，即信号的采样点个数，L为信号的时间长度）；
3. 将2段信号频谱平移至高频部分（移动$+f_s$，$+2f_s$），与剩下一段信号的频谱混合，装载入频率为$3f_s$的信道（这里用采样率为$3f_s$的音频文件模拟），IFFT后得到与原长度一致，采样率为$3f_s$的音频（编码后的音频）；
4. 将编码后的音频做FFT变换后得到频谱，原来三段音频的频谱分别在$0, +f_s, +2f_s$处，取出后做$f_s \cdot L$点IFFT即可恢复原音频。

3.2 实验结果

一个常见的错误做法如下

```
res(1     :fs ) = f_audio1(1:fs)
res(fs+1  :2fs) = f_audio2(1:fs)
res(2fs+1:3fs) = f_audio3(1:fs)
```

这样得到的信号IFFT后是无法存储为声音文件（只有实部）的，如果强行去掉虚部，`res(1:fs)`与`res(2fs+`
发生混叠，从而使第一段信号与第三段信号无法被正确提取。

当信号为实信号，长度N为偶数时，DFT得到的结果关于中点共轭对称，即

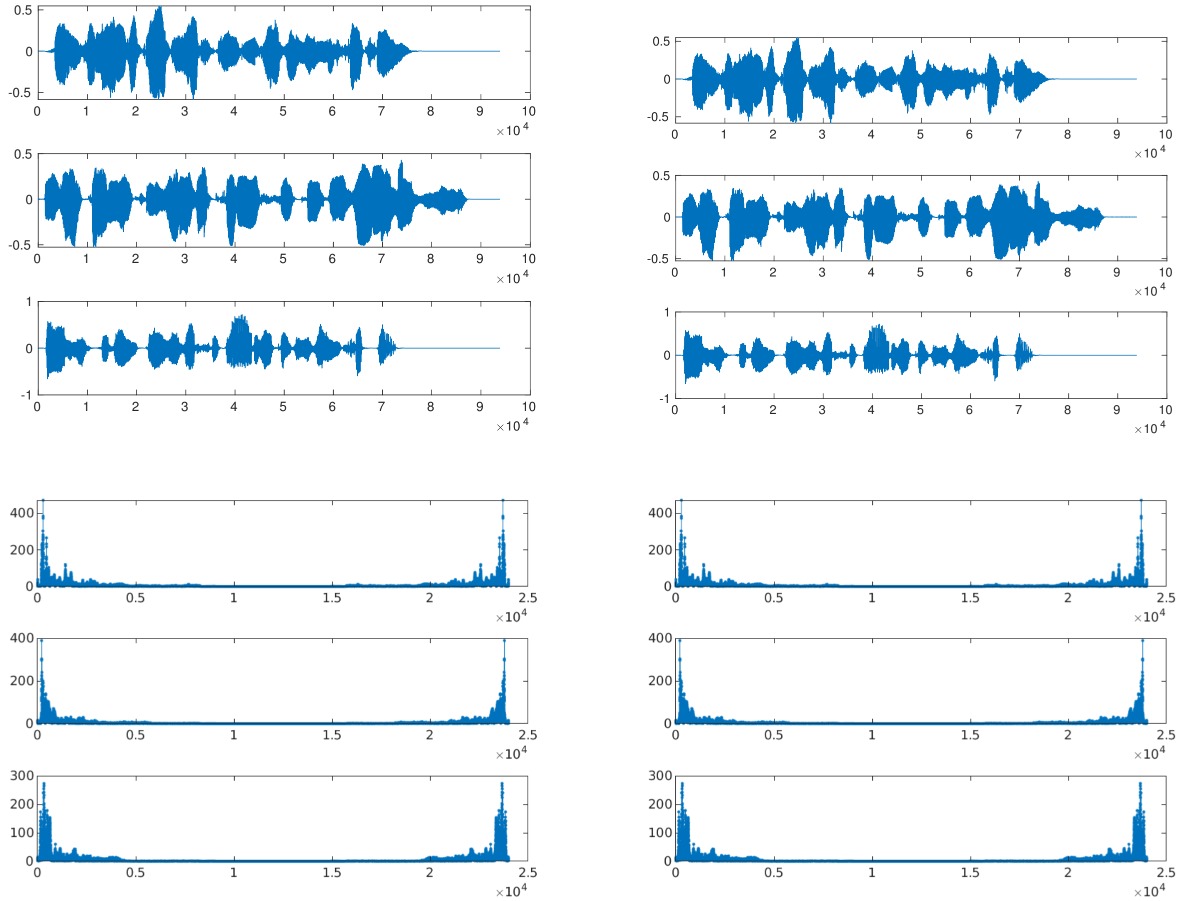
$$X(k) = X^*(N-k) \quad k \in [0, N], k \in \mathbb{Z} \tag{1}$$

因此实际移动频谱时，正确操作为

```
res(1       :fs/2 ) = f_audio1(1     :fs/2)
res(2fs     :5fs/2) = f_audio1(fs/2 :fs   )
res(fs/2+1 :2fs   ) = f_audio2(1     :fs/2)
res(5fs/2+1:3fs   ) = f_audio2(fs/2 :fs   )
res(fs+1    :2fs   ) = f_audio3(1     :fs   )
```

此时编码后的信号频谱也是关于中点共轭对称的，IFFT后得到的信号也是实信号。

下面是实验结果



(a) 原信号　　(b) 解码后信号

图 5: 原信号与解码后信号对比（上为时域，下为频域）

可以看出解码后信号与原信号完全一致（原信号的所有信息均被完整保留），听感上也没有差别。

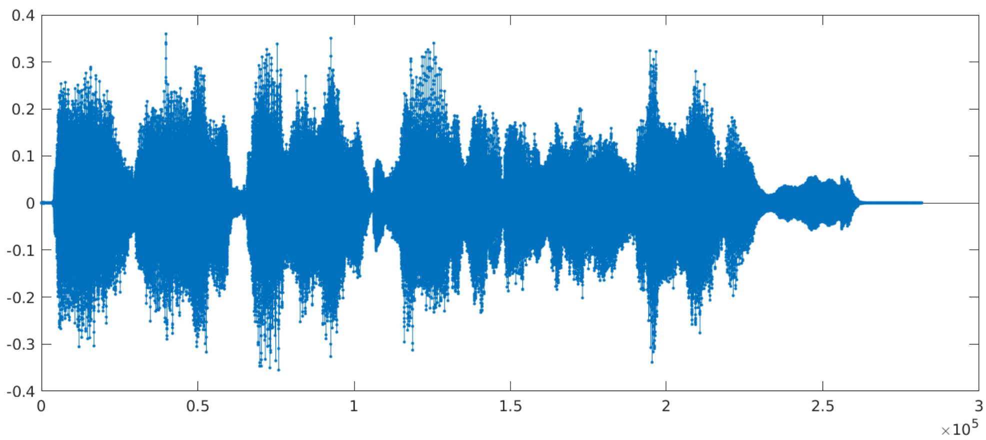


图 6: 编码后信号时域

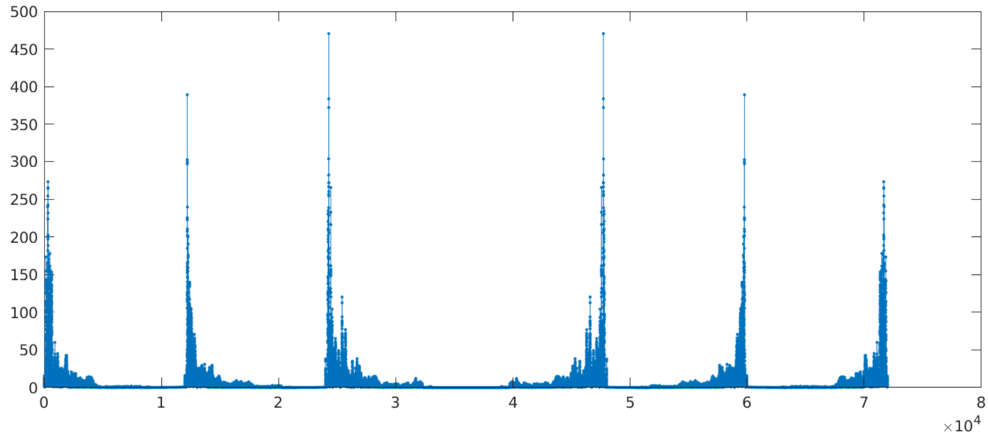


图 7: 编码后信号频域

编码后的信号听起来类似于第一段信号+比较尖锐的高频部分，因为它的基频来自第一段信号。