信号第九次作业实验报告

翁家翌 2016011446

2018.11

1 双音频按键识别

1.1 算法与代码

我实现了以下两种方法检测按键音频：

`fft_test.m` 使用普通 FFT 算法求出频域信号，分别找出低频和高频的两个峰值并查表，作为预测结果；

`goertzel.m` 使用 Goertzel 算法求出在八个频率中最有可能的两个频率，作为预测结果；

运行“`run.m`”即可查看结果。

此外，我还写了一个生成模拟音频的程序“`gen_dtmf.m`”，能够输入一串数字输出对应的 DTMF 音频。

1.2 实验结果

1.2.1 iPhone 单个按键

这部分的数据是从网络上下载下来的，为“`dtmf-{0-9}.wav`”，实验结果如下：

按键	长度 (帧)	FFT 结果	Goertzel 结果	FFT 耗时	Goertzel 耗时
0	4687	0	0.001241	0	0.001608
1	4049	1	0.000533	1	0.000475
2	3731	2	0.000660	2	0.000427
3	5254	3	0.000776	3	0.000506
4	5326	4	0.000819	4	0.000507
5	4523	5	0.000591	5	0.000291
6	5613	6	0.000510	6	0.000263
7	5071	7	0.000513	7	0.000232
8	4918	8	0.000481	8	0.000230
9	4658	9	0.000507	9	0.000224

表 1: Experiment Result of Default Setting

从表1中可以看出，Goertzel 算法在大部分情况下速度快于原始的 FFT，并且准确度一致。

1.2.2 合成音频测试

根据脚本“gen_dtmf.m”生成的音频进行测试，每 5000 帧检测一次数字，结果如下：

```
FFT: 时间已过 0.014279 秒。
FFT gives 11188877555007776663311199988
Goertzel: 时间已过 0.006554 秒。
Goertzel gives 11188877555007776663311199988
```

可以看出，在合成音频测试中，Goertzel 算法要快于原始 FFT，并且识别准确度一致。

1.2.3 实际音频测试

网络上有一段优酷记者打电话给 360 老总周鸿炜的视频，里面有手机号拨打的声音。我把它下载下来并用 ffmpeg 简单处理了一下变成一段约 4s 的音频文件。每隔 3500 帧检测一次。考虑到实际音频噪声比较大，在 FFT 识别的时候加入了一条规则：“如果频域能量大于某个阈值，则认为这是个电话按键音”。测试结果如下：

```
FFT: 时间已过 0.085034 秒。
FFT gives 233211123333333331777782700000002#31111211*31111111299999999077
121111112*20000322013999991#888805
Goertzel: 时间已过 0.010734 秒。
Goertzel gives 2225222223271111A32*15A64*6A33333333616353B467777837271BA#9
B22A22132#C*0000D*0020B271213*53111121444402*551111111A11B*571528*3538C32
AA99999999C937874A9B071233403CAA111111111*211#A52*8000030502*DA999999*993
08888DCABB895338
```

虽然 Goertzel 跑的比 FFT 快，但是识别精度惨不忍睹，FFT 至少还能看出号码是什么。搜了一下“13701191098”发现的确是他的号码。

2 卷积计算方法的性能比较

2.1 算法与代码

我实现了以下四种方法计算卷积：

conv_origin.m 使用公式法求解卷积；

conv_circle.m 使用 FFT 求解卷积；

overlap_save.m 使用 Overlap-Save 方法求解卷积；

overlap_add.m 使用 Overlap-Add 方法求解卷积；

运行“run.m”即可查看结果。在测试中，X 序列长度从 1000 递增至 20000，Y 序列长度维持 1000 不变。

2.2 实验结果

从图1可以看出，公式法求解卷积速度最慢，圆卷积 FFT 次之，两个 overlap 方法不相上下，因为它们最适合这种情况（一个很长一个很短）下求卷积。

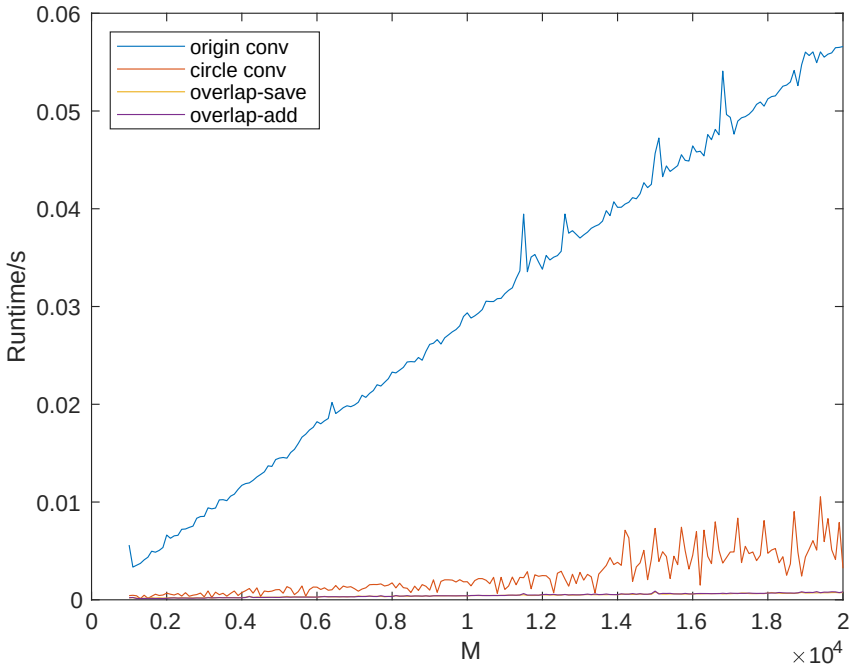


图 1: 卷积计算方法的性能比较

3 语音信号的频分复用

3.1 算法与代码

我在“FDM.m”中实现了针对给定的 3 个音频信号的编码与解码功能，具体而言：

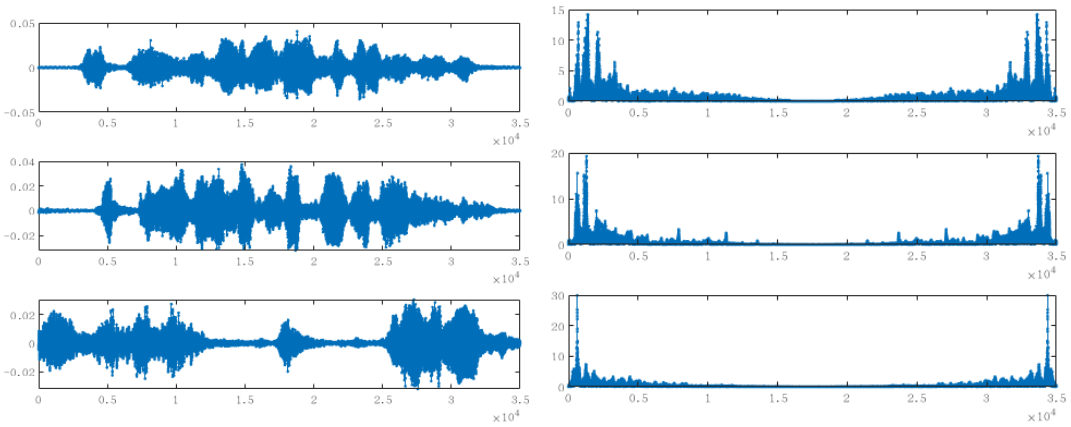
1. 将 3 个音频裁剪成相同长度；

2. 经过 FFT 之后去掉低频部分，保留低频与高频信号并拼接在一起；

3. 从编码的音频 FFT 变换之后提取每个音频的高频部分并使用 IFFT 变换为原始音频；

3.2 实验结果

图2、图3、图4分别显示了编码前、编码和解码之后的频域与时域信号。对比图2(a)和4(a)可以看出该方法几乎没有损失主要信息。实际人耳听的效果也很好。(那些只放了频域的图的同学，时域的效果简直不忍直视……)



(a) (b)

图 2: 原始音频信号，左边是时域，右边是频域

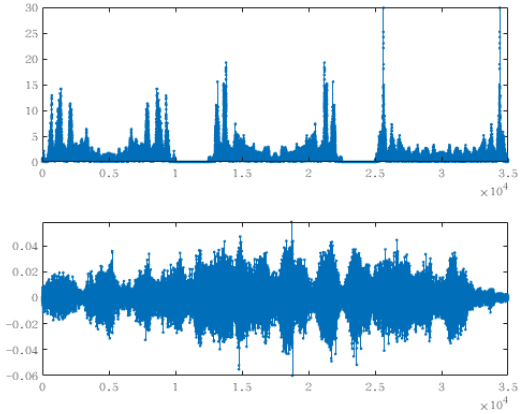
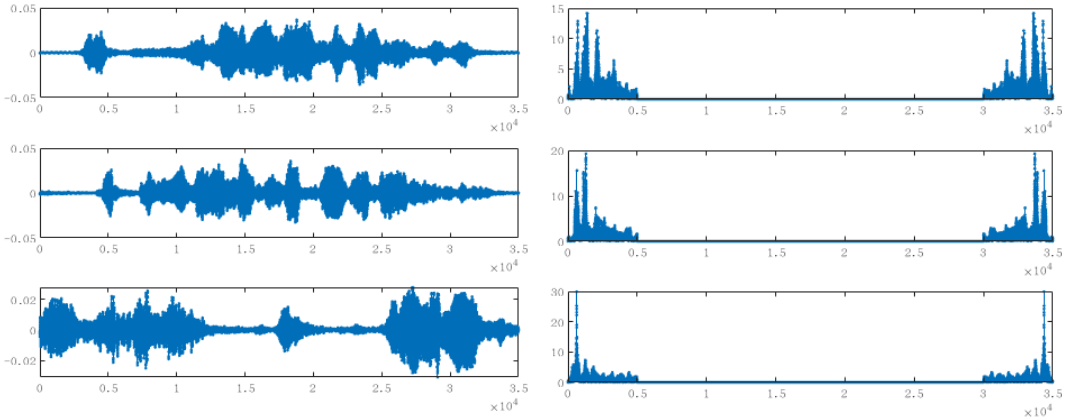


图 3: 编码音频信号，上边是频域，下边是时域



(a) (b)

图 4: 解码音频信号，左边是时域，右边是频域