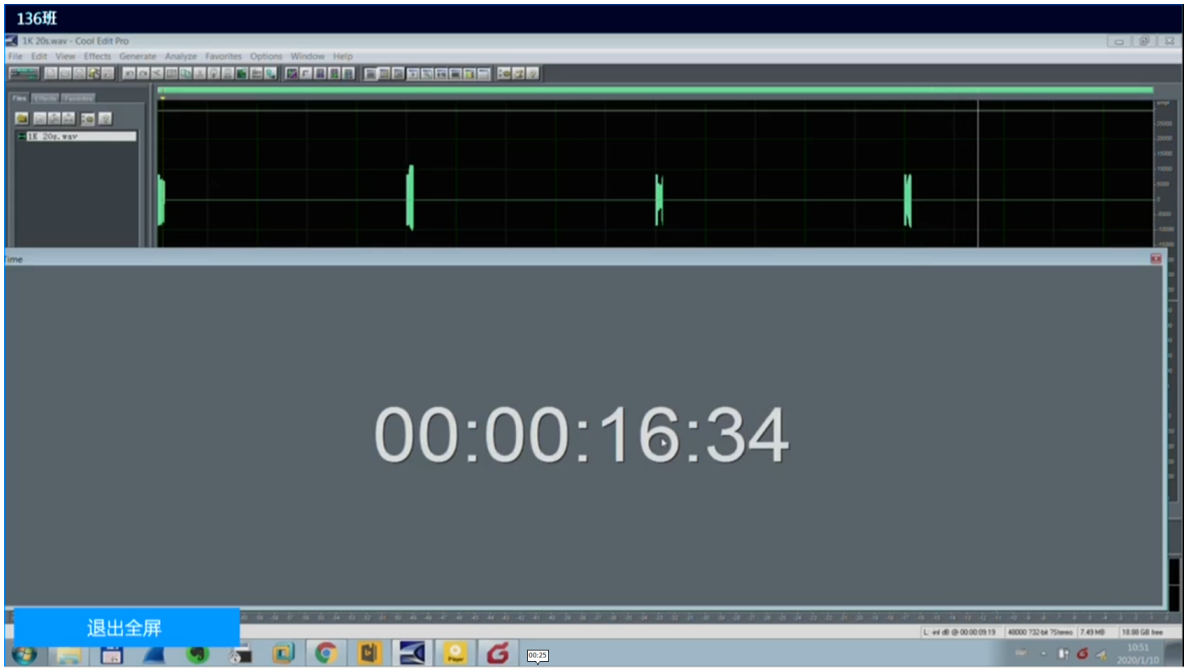
音视频同步检测的方法

背景

对于在本地录制后传送到网络中的视频，可能会因硬件原因或网络不稳定等原因导致音画不同步。如果要使视频同步，那么就必须先算出音频和视频之间的延时。

视频录制

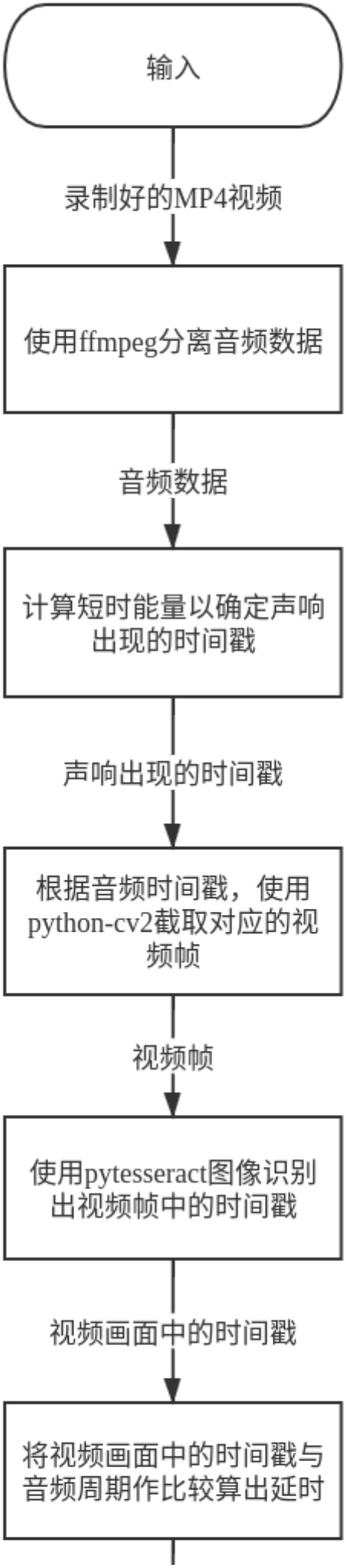
在录制的时候，选取一个周期性发声的音频数据(比如，每5s发出一个声响)。对于视频画面端，留出一部分区域来显示音频的播放时间。音频播放时间可通过软件解析出来(COOL EDIT)。



为什么要选择周期性发声的音频&&为什么这样可以计算延时？

为了方便音视频延时的检测。音频每个周期发出一个声响，理论上来说，如果没有延时的话，那么视频画面中的时间戳应该正好是这个周期的整数倍。比如，如果音频每5s发出一个声响，在视频播放的15s出现了一个声响，此时视频画面中对应的时间戳是15.32s。理论上视频画面中的时间戳应该是15s，这样我们就知道延时是0.32s。

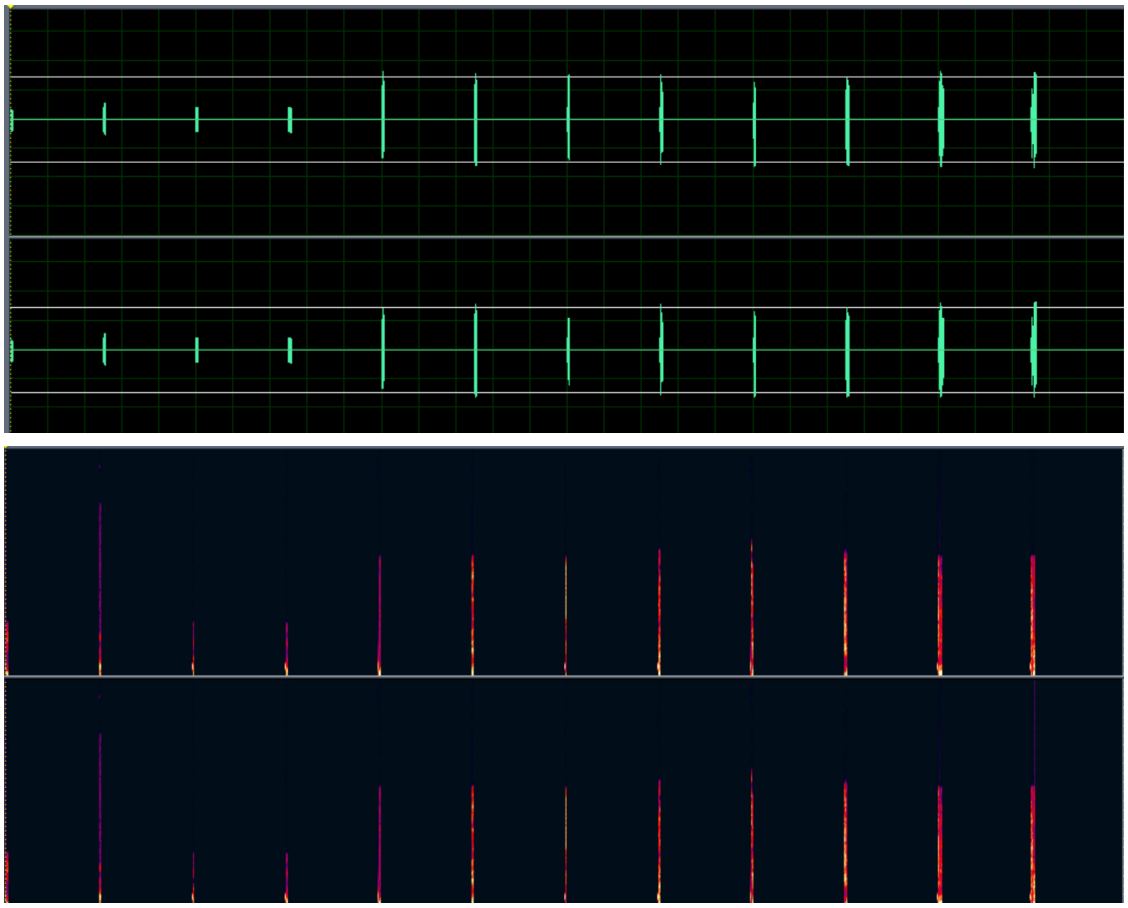
检测步骤


输入
录制好的MP4视频
使用ffmpeg分离音频数据
音频数据
计算短时能量以确定声响出现的时间戳
声响出现的时间戳
根据音频时间戳，使用python-cv2截取对应的视频帧
视频帧
使用pytesseract图像识别出视频帧中的时间戳
视频画面中的时间戳
将视频画面中的时间戳与音频周期作比较算出延时

1. 使用ffmpeg分离音频并转换成未压缩的音频数据

   因为音频短时能量必须用原始的，未压缩过的音频数据来计算，而mp4中的音频(aac)是经过压缩的，所以在提取出音频数据后，还要将其转换成pcm格式的数据。这个转换过程网上已经有很多blog了，就不赘述了。

2. 音频短时能量计算



   短时能量指的是较短时间内的语音能量。这个较短时间通常指的是一帧。对于音频来说，实际上没有一个标准来规定一帧是多长时间，不过一般都选取20毫秒为一帧。在这一帧时间内，音频数据包含了成百上千个采样(sample)，通过求20毫秒内sample的平方和，我们可以算出一帧视频所蕴含的能量。

$$short\_time\_energy = \sum sample\_i^2$$

   对于转换后的音频数据，它在程序中以数组的形式保存。通过ffmpeg的解析，我们可以知道一个sample是多少个字节(一般是2个字节)和采样率等数据。这样就可以知道这个音频数据中总采样数

$$total\_sample = data.size() \div sample\_width$$

$$total\_frame = audio\_duration \div frame\_size$$

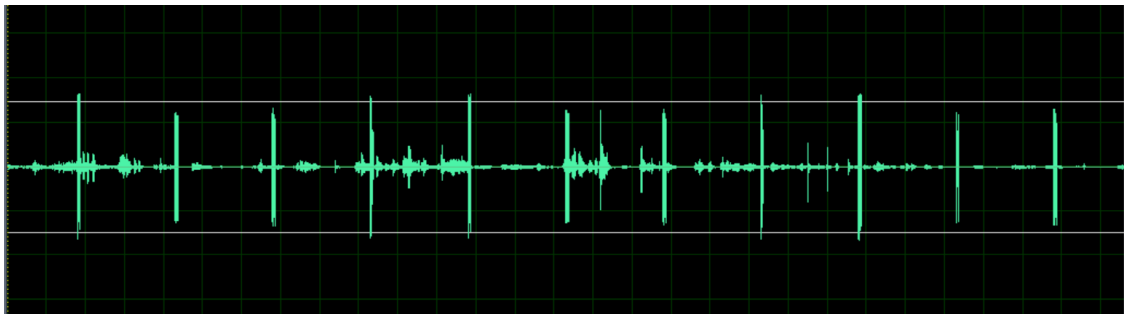
$$sample\_per\_frame = total\_sample\_num \div total\_frame$$

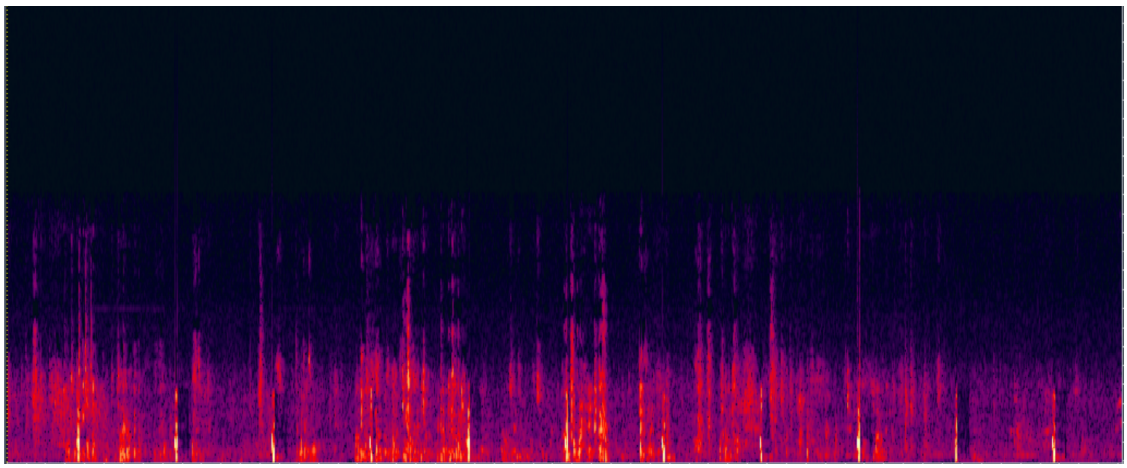
遍历这个数组，计算出每帧的短时能量，如果能量值不为0，则记录下能量出现的时间戳。理论上来说，音频是每个周期发出一个声响，所以只有在周期整数倍的时刻能量值才大于0，其他时刻能量值都是等于0的。伪代码如下所示:

```
time_offset = 0;
energy = 0;

for (i = 0; i < sample_num; ++i) {
    energy = samples[i] * samples[i];

    // 一帧
    if (i % sample_per_frame == 0) {
        if (energy > 0) {
            record time offset;
        }

        time_offset += frame_size;
        energy = 0;
    }
}
```

计算短时能量遇到了什么问题？

因为录制并非在一个完全无噪声的环境下进行，所以会存在噪音对短时能量的计算造成干扰。本来短时能量为0的时间点这时候会出现能量。





解决方案

虽然无法要求录制在一个完全安静的环境下进行，但是可以要求在一个相对安静的环境下进行。这时候即使存在噪声，噪声的能量值也不会超过实际音频数据的能量值。所以可以设置一个过滤值，小于该过滤值的能量都认为是噪声产生的。

3. 使用python-cv2截取对应的视频帧

   通过音频数据的时间戳可以计算出对应的视频帧，通过cv2截取就可以了。

```
video_frame_num = audio_time_stample * video_frame_rate
cv2.set(cv2.CAP_PROP_POS_FRAMES, video_frame_num)
cv2.read()
```

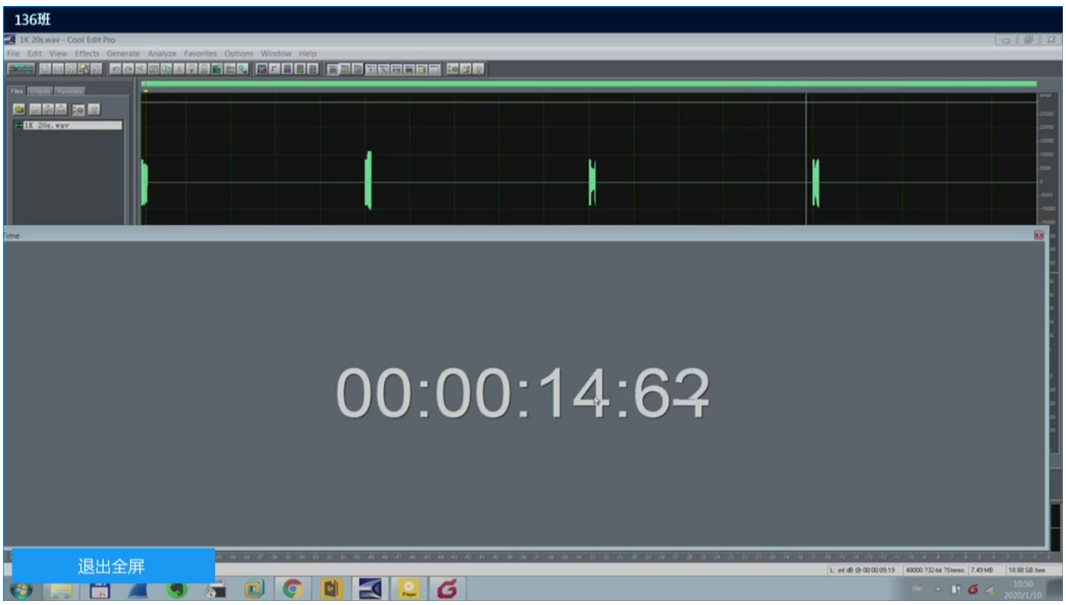
4. 使用pytesseract识别出视频帧中的时间戳

   pytesseract是一个光学字符识别库，可用于提取图片中的文字。光学字符识别是指对文本资料的图像文件进行分析识别处理，获取文字及版面信息的过程。

   通过调用pytesseract api即可提取视频画面中的时间戳

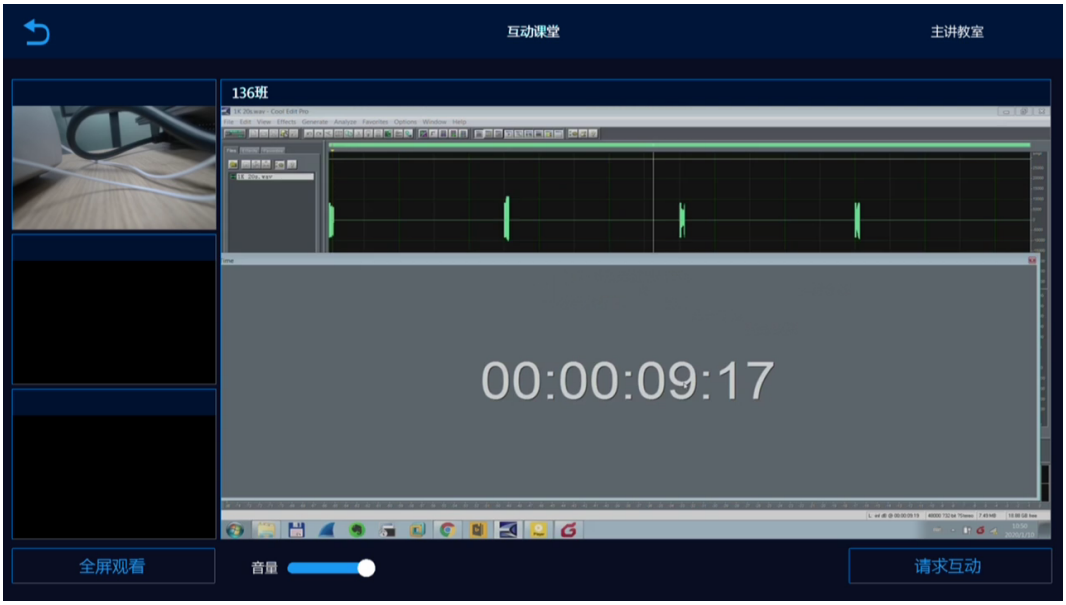
提取视频画面时间戳遇到的问题---图像无法识别

1. 花屏



   出现该问题的原因是因为计算机采取逐行扫描的方式来刷新屏幕，对于该帧，屏幕下半部分还没刷新。

2. 复杂画面



该画面中的特征信息太多了，而且时间戳的特征不够明显，如果只使用pytesseract默认的训练集，是无法识别出时间戳的。识别出来的结果可能是个乱码的字符串。

解决方案

对于第一个问题，我们可以每次截取前后三帧。计算这三帧的时间间隔，如果时间间隔在某个合理的范围内，则认为没有出现花屏。比如说时间间隔在80毫秒以内。如果间隔1过大，那么取后面两帧作为结果。如果间隔2过大，则取前两帧作为结果。如果三帧都有问题，则放弃计算该时间戳的误差，并打log。

对于第二个问题，有两种解决办法。一个是开发者针对这种图片训练出训练集。另一个是在录制的时候，确保时间戳区域占据画面的1/2大小。第一种方法的缺点是比较耗时，因为训练数据的时候要对数据做标注。第二种方法的缺点是严格限制了录屏的方式。

5. 将视频画面中的时间戳与音频周期做计算，算出延时

   先将时间戳对音频周期做取模运算。

   对于延时，有两种情况。音频快于视频或者视频快于音频。我这里判断的方法比较粗暴，如果延时小于1/2音频周期，则认为视频快于音频。否则认为音频快于视频。

```
delay = time_stample % audio_time_interval

if delay <= (audio_time_interval / 2)
    delay
else
    delay = delay - audio_time_interval
```

举个例子

```
音频周期      :    5000ms
时间戳1       :    10200ms
时间戳2       :    14000ms
```

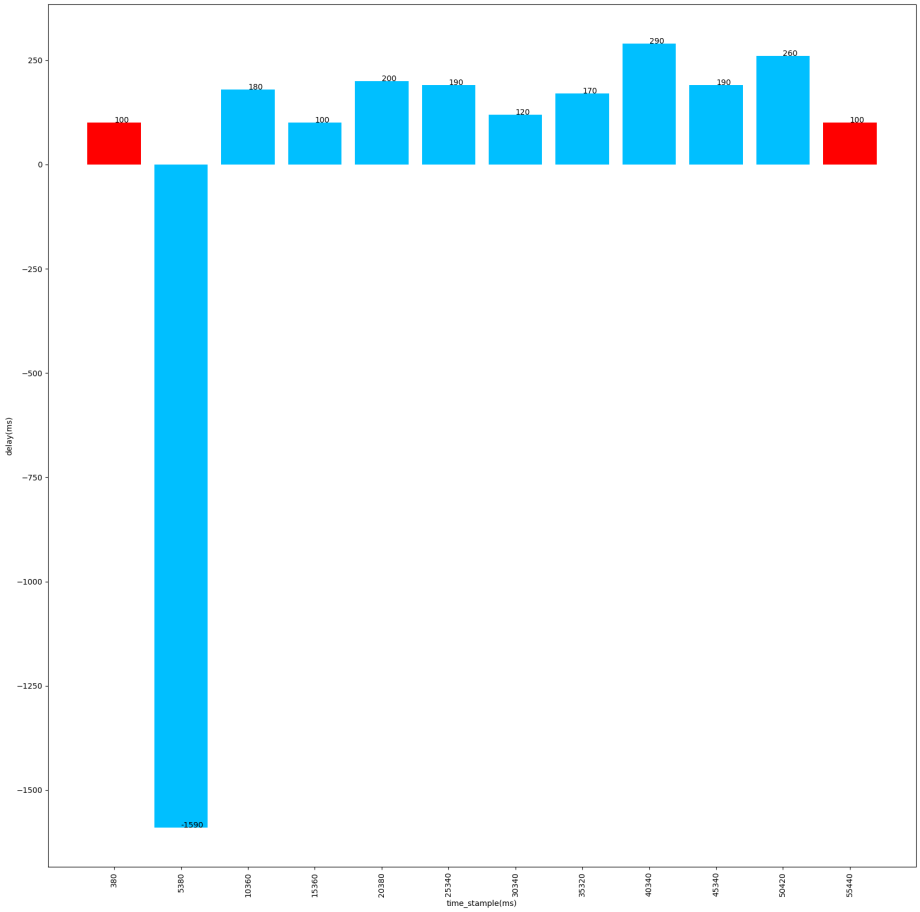
那么，

对于delay1=200ms < 2500ms，视频快于音频200ms

对于delay2=4000ms > 2500ms，音频快于视频1000ms

6. 画图

   使用matplotlib api即可



关于性能以及优化

对于一个4.4Mb的mp4文件，程序运行了7s。

程序主要慢在图像识别这部分，一张图像要识别0.3秒。改进方法是增加训练集，加快识别的速度。另外计算短时能量，图像识别也可以用多线程来做。程序短时能量计算部分使用c++写的，程序其余部分是python写的。如果全部用c++写程序也许能再快些。