ECE 397 report
An Zhao
Prof. Beauchamp
May 4th, 2016

Squeak elimination

Introduction

Squeak is a short, high-pitched sound. Although used in some contemporary works, squeak sound could be a defection in many music recording files, which is considered a mistake from performers. This project aimed to eliminate the sound of reed squeak in a wav format sound file. The project is implemented by C and the approach to delete the squeak will be in the following discussions.

Theoretical Discussion

Woodwind reed resonance can be adjusted by clarinet players from 2khz – 3khz (Thompson). I got my test sample from a saxophone player. The spectrum shows that the squeak frequency is about 2khz, which is the reed resonance. The goal is to detect, or extract samples with the fundamental frequency of 2khz in frequency domain, and eliminate these samples from the wav file in time domain.

In order to analyze components in the frequency domain while still keep time detail of the signal, short time fourier transform is applied here. The size of window is critical here. The larger the window size is, the better frequency resolution is achieved; while the smaller the window size is, the better time resolution is achieved. Window size also limits the lowest frequency component can be analyzed. The lowest fundamental frequency for my test file is manually recognized as 220 Hz.

The window size, or the time duration of signal to be fourier transformed each time is 3 times the period of the lowest frequency. The length is window size is:

$$L = 3 * \frac{1}{f_{lowest}} * f_s$$

where fs is the samplerate.

This length is then zero padded to the closest 2's power number, so that it can go into the fft algorithm.

$$L_2 = 2^{ceil(\log_2 L)}$$

The frequency related to each bin is

$$f(n) = \frac{n}{N} * f_s$$

To extract components of squeak frequency, spectral energy density is calculated in the certain range. The spectral energy density is defined as

$$E_s(f) = |X(f)|^2$$

where X(f) represents the Fourier transform of x(t).

Programming Implementation

fs = 48000hz; WINDOW_LEN = 656;

1. To read in the sound file, I used a library called libsndfile, by which I can read the input sound file into an array. The array size would be the number of samples in my input file.
2. Once the input is in an array, I apply stft on it. As aforementioned, the window length calculated according to my test file is 655. This length of signal is zero padded to 1024. For fft, I used a library called kissFFT. The fft function returns output with two 1-D arrays, one for real value and one for imaginary value.
3. The total number of windows is

$$stftnumber = ceil(frames / WINDOW\_LEN * 2 - 1)$$

*2 here refers to my hop size, which is half window. I stored my fft magnitude for all the windows into a 2D array. Each row corresponds to the window. Each column corresponds to the certain frequency bin.

4. I set the frequency range I want to extract to be 1900-2200hz, where the bins corresponding to this range would be:

downlimit = ceil(1900 * windowzeropadsize / samplerate);

uplimit = ceil(2200 * windowzeropadsize / samplerate);

I summed up the spectral energy in this range for every window and store the result in another array.

$$E(f) = \sum_{downlimit}^{uplimit} X(f)^2$$

The threshold is the average of the summation spectral energy for each window in this range.

5. If the value of the summation of a window is above average, the samples in the time domain corresponds to that window will be deleted.

The relationship between the number of window and the number of sample is: samples[i*WINDOW_LEN/2+k], where i is the number of window, and k is the number of sample in the window. For example, if window 47 has spectral energy above average, then samples[47*656/2] to samples[47*656/2 +655] will be deleted. In order to shorten the file in time domain, the samples to be deleted are given 987654321.0f as a label, because 0 won't work.

6. The number of samples to be deleted will be counted by this label. Output size array would be the original length – deleted length. The remaining samples will be rearranged into the output file.

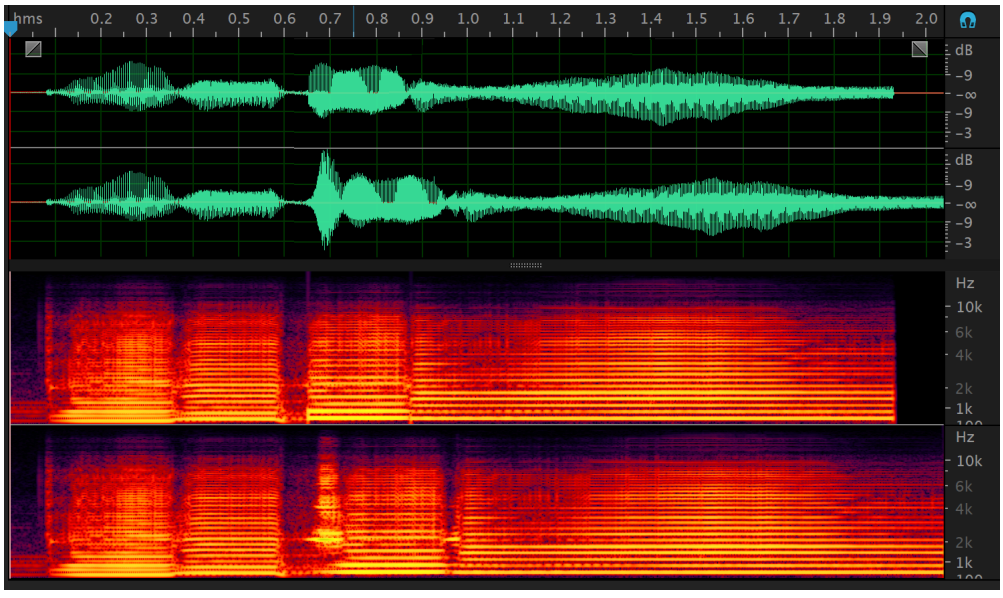
How to use

To run the program, do

Make

./main

Test results



Code

```
#include   <stdio.h>
#include   <sndfile.h>
#include   <math.h>
#include   <stdlib.h>
#include   "kiss_fft.h"
#define    BUFFER_LEN  1024
#define    WINDOW_LEN  655 // fundamental frequency is 220hz,
//the window size needs to be two cycles of the period of fund freq
//3/220*48000(samplerate) = 655 is the number of samples need to be fft each time

int fft2(float a[], float b[], int m, float dx, int inverse);

kiss_fft_cpx* copycpx(float *mat, int nframe)
{
   int j; // counting variable for fft
   kiss_fft_cpx * mat2;
   mat2 = (kiss_fft_cpx*)calloc(sizeof(kiss_fft_cpx), nframe);
   kiss_fft_scalar zero = 0;
   for (j=0; j<nframe; j++)
   {
      mat2[j].r = mat[j];
      mat2[j].i = zero;
   }
   return mat2;
}
int main (void)
{   /* This is a buffer of double precision floating point values
   ** which will hold our data while we process it.
   */
   static float data [BUFFER_LEN] ;

   /* A SNDFILE is very much like a FILE in the Standard C library. The
   ** sf_open function return an SNDFILE* pointer when they sucessfully
   ** open the specified file.
   */
   SNDFILE      *infile, *outfile ;

   /* A pointer to an SF_INFO stutct is passed to sf_open.
   ** On read, the library fills this struct with information about the file.
   ** On write, the struct must be filled in before calling sf_open.
   */
   SF_INFO     sfinfo, sfinfo2;
   int         readcount ;
   const char  *infilename = "saxphone_sample.wav" ;
   const char  *outfilename = "output.wav" ;
```

```
    /* Here's where we open the input file. We pass sf_open the file name and
    ** a pointer to an SF_INFO struct.
    ** On successful open, sf_open returns a SNDFILE* pointer which is used
    ** for all subsequent operations on that file.
    ** If an error occurs during sf_open, the function returns a NULL pointer.
    */

    int i; // counting variable

    if ( !(infile = sf_open (infilename, SFM_READ, &sfinfo)) )
    {
       /* Open failed so print an error message. */
       printf ("Not able to open input file %s.\n", infilename) ;
       /* Print the error message from libsndfile. */
       puts (sf_strerror (NULL)) ;
       return  1 ;
    } ;

    memcpy(&sfinfo2,& sfinfo, sizeof(sfinfo)); // formatting memory for output file

    /* Open the output file. */
    if (! (outfile = sf_open (outfilename, SFM_WRITE, &sfinfo2)))
    {
       printf ("Not able to open output file %s.\n", outfilename) ;
       puts (sf_strerror (NULL)) ;
       return  1 ;
    } ;

    /* While there are.frames in the input file, read them, process
    ** them and write them to the output file.
    */
    const int windowzeropadsize = 1024; //calculate size of zeropad length to power of 2
    int frames = sfinfo.frames;
    int samplerate = sfinfo.samplerate;
    printf ("%d %d\n", frames, samplerate);
    float * samples = (float*) calloc(frames, sizeof(float));
    float * windowzeropad = (float *) calloc(windowzeropadsize, sizeof(float));//memory for
zeropadded signal
    float * fftrout = (float *) calloc(windowzeropadsize, sizeof(float));//realpart output for fft
    float * fftiout = (float *) calloc(windowzeropadsize, sizeof(float)); // imaginary output for fft2
    int loop = 0;
```

```
while ((readcount = sf_read_float(infile, data, BUFFER_LEN)))
  {
     for (i=0; i<readcount; i++)
     {
        samples[loop*BUFFER_LEN+i] = data[i]; // read all the signal into the zeropad memory
     }
     loop++;
  } ;

  //zeropad
  float fftmag;
  int windownum = 0;
  int stftnumber = ceil(frames/WINDOW_LEN*2 -1);
  kiss_fft_cfg cfg = kiss_fft_alloc(windowzeropadsize, 0, 0, 0);

  //assign memory for 2D array to store fft magnitude
  int r = stftnumber;
  int c = windowzeropadsize;
  int j;
  float *window[r];

  for (i=0;i<r;i++)
     window[i] = (float *) malloc(c * sizeof(float));

  for (windownum=0; windownum <stftnumber; windownum ++)
  {
     //take 656 samples, zeropad each to 1024
     for(i=0; i< WINDOW_LEN; i++)
        windowzeropad[i] = samples[windownum * WINDOW_LEN/2 + i];
     //fft
     kiss_fft_cpx out_cpx[windowzeropadsize], *cpx_buf;
     cpx_buf = copycpx(windowzeropad, windowzeropadsize);
     kiss_fft(cfg,cpx_buf,out_cpx);
     //store fft magnitude into 2D array
     for (i = 0; i < 1024; i++)
     {
        fftmag = sqrt((out_cpx[i].r * out_cpx[i].r)+(out_cpx[i].i * out_cpx[i].i));
        window[windownum][i]=fftmag;
        //printf("%f\n",fftmag);
     }
  }
```

```
 float * freqsum = (float *)calloc(stftnumber, sizeof(float));
   int k;//counting variable
   float sum = 0;// accumulation of fftmag in frequency range
   int downlimit;
   downlimit = ceil(1900 * windowzeropadsize / samplerate);
   int uplimit;
   uplimit = ceil(2200 * windowzeropadsize / samplerate);
   float avg = 0.0;
   //printf ("%d %d\n", downlimit, uplimit);
   for (i = 0; i< stftnumber; i++)
   {
      sum = 0;
      for (j= downlimit; j<uplimit+1; j++)
      {
         sum = sum + window[i][j]*window[i][j];
      }
      freqsum[i] = sum;
      printf("%d %f\n",i,freqsum[i]);

      avg = avg + sum;
   }
   avg = avg / stftnumber; //calculating average for threshold value
   printf("%f\n",avg );
   for (i=0; i<stftnumber; i++)
   {
      if (freqsum[i] > avg)
      {
         for(k=0; k<WINDOW_LEN; k++)
            samples[i*WINDOW_LEN/2+k] = 987654321.0f;// number for detecting deleted
items
      }
   }
   int count = 0; // count the number of samples to be deleted
   for (i = 0; i<frames; i++)
   {
      if (samples[i] == 987654321.0f)//label of deleted sample
         count ++;
   }
   printf("%d\n", count);
```

```
  // rearrange the output file size and put samples in
    int outsize;
    int t = 0;
    outsize = frames - count;
    float * outputs = (float * )calloc (outsize, sizeof (float));
    for (i=0; i<frames; i++)
    {
       if (samples[i] != 987654321.0f)
           outputs[t++] = samples[i];
    }


    sf_write_float (outfile, outputs, outsize);

    /* Close input and output files. */
    sf_close (infile) ;
    sf_close (outfile) ;

    return 0 ;
}
```