Zhaoyuan Fang

EE 10114

DSP Project

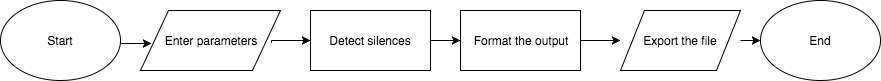
Professor Fuja

**Silence Modifier Technical Manual**

**Introduction**

This software works as a silence modifier for an audio file containing music or speech separated by multiple silences. Using some DSP Techniques, MATLAB functions, and self-devised algorithms, the software is able to achieve silence detection, replacement, and removal. This technical manual will first summarize the logic flow of the software, and various important components will then be developed in separate sections. In the appendix, the MATLAB code listing will be attached.

**Logic Flow**



The basic logic flow is shown as above, and the overall structure follows a user-friendly pattern. In each step, different algorithms, sub-structures and skills are used to ensure the efficiency and correctness of the program. Dividing the program into smaller parts makes it easier for understanding, debugging and specified improvements. The technical details will be rendered in the following section.

**Design Analysis**

Enter parameters

For a typical input, the program employs the while loop structure, compared to the simple one line input structure. The use of while loop leads to a user-friendly function: the user’s mistake of input will be detected and he or she will be given a chance to re-enter as long as the input is not in the correct form. More importantly, another utility of using the while loop input is that the program won’t proceed unless the input is correct. Thus, the part of the program following the input is guaranteed to work appropriately.

Besides the input technique, a remarkable component of the program is the feature of reading the audio file automatically. The program saves all the audio suffixes in a cell array *“fileSuffix”* beforehand. Once the user enters the name of the file, the program loops through the cell array, matches the file name and the suffix by checking the existence of the file in each loop. In this way, the user does not have to remember the exact suffix for a file.

Detect silences—DSP Techniques

The overarching idea of detecting silences is: first, search through all tiny pieces (each with length *Tstep*) in the audio and check whether that satisfy the magnitude threshold of silence; second, when a continuous interval (consisted of several tiny pieces) comes to an end, check the interval and save the front and rear indices of if it satisfies the length requirement of a silence; third, in order to make the program clearer, a *“delete”* vector with length of the audio is used to mark whether each frame is kept or deleted.

When the program loops through the audio file to detect the silences, in each loop, it calls the fft function (Fast Fourier Transform) over a certain length (*Tstep*). The use of fft is important here because it helps to generate the actual amplitude of a certain piece of audio, and it makes it easier to manipulate using the magnitude function. The fft returns an array of complex numbers saved in the *“spec”* vector, and the program plugs the first half of *spec* into the mag() function to get the magnitudes of this piece of audio. By squaring those magnitudes and adding them up, the software derives a fair parameter to evaluate the loudness of this tiny piece in the audio. In the program, this parameter (denoted by *“eit”*) is used not to determine whether that piece of sound is silence. If it is silence, delete that piece and add the piece into the calculation of “safe noise” for later use; else, the piece is kept in the output, by setting the corresponding value in the *“delete”* vector.

In this section of detecting noises, digital signal processing skills is combined with self-devised algorithms. However, this part does not really delete the “silences”. Instead, it keeps all the needed information in arrays and variables such as *“eitTot”* (total silence energy), *“eitCount”* (number of silences), *“front\_array”* *& “rear\_array”* (array consisting head and tail marks of replacement), etc. By separating detection and deletion/replacement, the program has more clarity and tidiness.

Format the output

Based on the number of qualified sound intervals (zero or nonzero) found and the length of replacement sound (zero or nonzero), the program automatically makes the choice of whether it plays the sound and whether it insert additional noises. Note that many previously modified parameters come into real use in this section of the program. The way the program formats its output employs the idea of copying. Since it is complicated and confusing to manipulate the original sound file directly, the program copies the only useful pieces from the original audio file, and selectively adds new pieces generated in the program to the output file.

Export the file

The last part of the program is relatively short, but similar to the other parts, details are taken into account. Similar to the first part of the program with the *audioread()* function, this section employs the *audiowrite()* function, with the path defined in the program so that the user knows which folder to look for.

**Post-mortem Analysis**

After finishing the project, there are several take-away points worth mentioning:

* When writing the program, first plan the overarching structure and the functionalities of each section, in order to have a clear roadmap and to make it easier when debugging.
* Reading through your own codes and running the program over and over again are very helpful for improvements, since they discover flaws for you.
* Representative variable names are useful for both your and others’ understanding of the program, and they are a great tool to guide someone through the codes.
* Functions like *audiowrite()* and ways to use it might not be mentioned in class, but there are many resources online and they could be a great help for the program.

**Appendix**

MATLAB Code listing:

clear

clc

rng('shuffle')

%% read file and set parameters

%save all the audio suffix types MATLAB supports

fileSuffix={'.wav','.ogg','.flac','.au','.aiff','.aif','.aifc','.mp3','.m4a','.mp4'};

openName=input('Please enter the name of the audio file: ','s');    %input filename

path='/Users/zhaoyuanfang/Desktop/EE/DSP/';                         %specify the file path

foundfile=false;                                                    %file not found yet

%prompt the user to enter filename, until it is a legal one

while ~foundfile

    %loop through the suffix folder

    for it=1:length(fileSuffix)

        %update the temporary (test) file name

        tempname=[path,openName,char(fileSuffix(it))];

        if exist(tempname, 'file')      %check if this file exists

            fullname=tempname;          %if exists, set the fullname

            foundfile=true;             %the file has been found

        end

    end

    if ~foundfile

        disp('File not found!');        %display error message

        openName=input('Please choose another file: ','s'); %prompt for another input

    end

end

[x,fs]=audioread(fullname); %read the audio file, fs---sampling frequency

fN=fs/2;                    %Nyquist frequency

audioLen=length(x);         %record the length

playOri=input('Play the original file? Y/N: ','s');

while ~((playOri=='Y') || (playOri=='N'))

    playOri=disp('Please enter Y for yes or N for no: ','s');

end

if playOri=='Y'

    playChoice=true;

else

    playChoice=false;

end

if playChoice

    sound(x,fs);    %playback the original audio

end

%prompt the user to enter parameters, using representative variable names

%the while loops guarantee the inputs are usable for the program

%minimum length of silence

minLen=input('Lower bound: ');

while minLen<0

    minLen=input('Please enter a positive number');

end

%maximum length of silence

maxLen=input('Higher bound: ');

while maxLen<minLen

    disp('Higher bound must be larger than lower bound.');

    maxLen=input('Higher bound: ');

end

%maximum amplitude to be a silence

thresh=input('Silence threshold: ');

%replacement length

repSec=input('Replace these silences with silences of length: ');

while repSec<0

    repSec=input('Please enter a positive number');

end

%choice of replacement: safe noise/complete silence

safeinput=input('Replace silences with safe noise? Y/N: ','s');

while ~((safeinput=='Y') || (safeinput=='N'))

    safeinput=input('Please enter Y for yes or N for no: ','s');

end

%modify the inputs in the way the program wants

if safeinput=='Y'

    safenoise=true;

else

    safenoise=false;

end

repLen=repSec\*fs;   %length of silences to insert

Tstep=minLen\*fs/10; %set the step of the loop later

if mod(Tstep,2)==1  %make sure that Tstep is even, so that Tstep/2 can be used directly later

    Tstep=Tstep-1;

end

%% detect the silences

isSilence=true;             %flag for the silence

delete=zeros(1,audioLen);   %"bool" array

Nsi=0;                      %number of silences accumulated

front=1;                    %front array mark

rear=1;                     %rear array mark

Nright=0;

front\_array=[];             %the starts of "delete" zones

rear\_array=[];              %the ends of "delete" zones

eitCount=0;                 %the total number of noises

eitTot=0;                   %the total "energy" of noises

%loop through the audio

for it=1:Tstep:audioLen-Tstep

    spec=fft(x(it:it+Tstep-1)); %use fft

    mag=abs(spec);

    eit=sum(mag(1:Tstep/2).^2);

    %check whether it is silence and update the information

    if (eit>thresh)

        isSilence=false;

        %if there is no silence before, it's safe to update front mark

        if Nsi==0

            front=it+Tstep;

        end

    else

        eitCount=eitCount+1;

        eitTot=eitTot+sum(x(it:it+Tstep-1));

        Nsi=Nsi+1;          %extend the length of silence

        rear=it+Tstep-1;    %update rear mark

        isSilence=true;     %is still silence

    end

    %if the length of this silence meets all the requirements

    if (~isSilence) && ((Nsi\*Tstep>=minLen\*fs) && (Nsi\*Tstep<=maxLen\*fs))

        Nsi=0;                %this silence is over and processed, reset Nsi.

        delete(front:rear)=1; %mark these "qualified" silences as true in delete vector

        Nright=Nright+1;      %update the number of "qualified" silences

        if Nright==1          %if this is the first qualified period, no possiblity of repetition

            front\_array(Nright)=front;

            rear\_array(Nright)=rear;

        else

            if front~=front\_array(Nright-1) %make sure that the front\_array marks are unique

                front\_array(Nright)=front;

                rear\_array(Nright)=rear;

            else

                Nright=Nright-1;            %if not unique, go back

            end

        end

        front=it+Tstep;       %finished processing, update front mark

    end

    if (~isSilence) %after the condition above, now is safe to update front mark

        Nsi=0;

        front=it+Tstep;

    end

end

%% format the result & play the soun

index=1;                            %index of the output vector

Nprocessed=1;                       %number of qualified silences processed

repfinished=false;                  %whether the replacement has been finished

safeMin=0;                          %mininum of safe noise

safeMax=eitTot/(eitCount\*Tstep/2);  %maximum of safe noise

if Nright==0

    disp(['No silences found!']);

else

    %if not replacing the silences, just copy the sounds without silence

    if repLen==0                                %loop & copy the sounds without silence

        for it=1:audioLen

            if ~delete(it)                      %if this frame is kept, copy it

                out(index)=x(it);

                index=index+1;

            end

        end

    else

        for it=1:audioLen                       %loop & copy the sounds without silence & replace the silences

            if ~repfinished                     %only insert when not finished

                if it==front\_array(Nprocessed)  %if previous silence mark

                    if safenoise                %check if safe noise is needed

                        out(index:index+repLen-1)=(safeMax-safeMin).\*rand(repLen,1)+safeMin; %generate random safe noise

                    else

                        out(index:index+repLen-1)=0;

                    end

                    index=index+repLen;

                    Nprocessed=Nprocessed+1;    %proceed to the next

                end

                if Nprocessed>Nright             %check if finished

                    repfinished=true;

                end

            end

            if ~delete(it)                      %if this frame is kept, copy it

                out(index)=x(it);

                index=index+1;

            end

        end

    end

    pause();

    sound(out,fs);

end                            %playback the modified audio

%% export the modified audio file

%check if the user wants to save the file

dosavestr=input('Do you want to save the audio? Y/N: ','s');

while ~((dosavestr=='Y') || (dosavestr=='N'))

    dosavestr=input('Please enter Y for yes or N for no: ','s');

end

if dosavestr=='Y'

    dosave=true;

else

    dosave=false;

end

%if the user wants to save, ask for the file and export it.

if dosave

    filepath='/Users/zhaoyuanfang/Desktop/EE/DSP/';

    filename=input('Please enter the filename: ','s');

    filename=[filepath,filename,'.wav'];

    audiowrite(filename,out,fs);                      %write the file

    disp(['File saved as ',filename,', thank you!']); %report the information

%if the user does not want to save, give a thank message.

else

    disp('File not saved, thank you!');

end