İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

NOISE RECOGNITION BY THE AUDIO FINGERPRINT METHOD IN AUTOMOTIVE APPLICATIONS

M.Sc. Thesis by
Emin ERENSOY

Department : Mechanical Engineering

Programme : Automotive

Thesis Supervisor: Prof. Dr. Ahmet GÜNEY

JUNE 2009

İSTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

NOISE RECOGNITION BY THE AUDIO FINGERPRINT METHOD IN AUTOMOTIVE APPLICATIONS

M.Sc. Thesis by
Emin ERENSOY
(503071706)

Date of submission : 29 April 2009
Date of defence examination: 05 July 2009

Supervisor (Chairman) : Prof. Dr. Ahmet GUNEY (ITU)
Members of the Examining Committee : Prof. Dr. İrfan YAVAŞLIOL (YTU)
Assis. Prof. Dr. Özgen AKALIN (ITU)

JUNE 2009

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

ARAÇLARDA SES İMZASI METODUYLA GÜRÜLTÜ TANIMLAMA

YÜKSEK LİSANS TEZİ

Emin ERENSOY
(503071706)

Tezin Enstitüye Verildiği Tarih : 29 Nisan 2009
Tezin Savunulduğu Tarih : 05 Haziran 2009

Tez Danışmanı : Prof. Dr. Ahmet GÜNEY (İTÜ)
Diğer Jüri Üyeleri : Prof. Dr. İrfan YAVAŞLIOL (YTÜ)
Yrd. Doç. Dr. Özgen AKALIN (İTÜ)

HAZİRAN 2009

FOREWORD

I would like to express my deep appreciation and thanks for my advisor, Prof. Dr. Ahmet GÜNEY, for giving me the opportunity to work on this project and for his support, guidance and suggestions. It is a pleasure and honor being his student.

I would like to thank to co-workers in ROTAM and Res. As. Hasan KÖRÜK from Vibration and Acoustics Laboratory in Mechanical Engineering Department.

I also would like to special thank to my family who have given me eddless support all my life long.

Many thanks are also due to my dearest friends, Oğuz BOLATA, Ömer ÖZÇAM, Hande BABAESKİ, for their invaluable supports.

May 2009

Emin ERENSOY
Mechanical Engineer

TABLE OF CONTENTS

Page

ABBREVIATIONS

CPB **:** Constant Percentage Bandwidth
DFT **:** Discrete Fourier Transform
FFT **:** Fast Fourier Transform
FT **:** Fourier Transform

LIST OF FIGURES

Page

NOISE RECOGNITION BY THE AUDIO FINGERPRINT METHOD IN AUTOMOTIVE APPLICATIONS

SUMMARY

Although audio fingerprinting method is not well known in noise recognition in automotive applications, it is shown that audio fingerprinting method could be used for noise recognition in automotive applications by this study.

There is need of educated people to solve the noise complaining of the drivers and to educate people for these purposes is a long period and highly cost subject in automotive applications. By this study to recognise the noise is more independent from the knowledge of the human beings.

Although simple data recorders are enough to collect the data, for better measurements, more sophisticated measurement equipments, which are shortly presented in this study, are needed.

To extract the audio fingerprint, the recorded signal is windowed by hanning function. This is because of leakage effect. After that the signal's Fourier Transform (FT) is taken to compare the signals in a better way. Following part of the creation of fingerprint is that FT of the signals is divided in to the bands and takes the power spectrum. As a final of the creation of the fingerprint, the values of the band are written in hexadecimal numbers.

The reference fingerprints which are stored in the database and the test fingerprint which is extracted from the unknown data are compared to match each other.

Algorithm are coded with "C+" and "C #.NET", which source codes are, included in appendixes, for Windows platform computers with installed ".NET Framework3.5" package.

Algorithms could reduce the cost of recognition of noise in automotive applications after selling.

ARAÇLARDA SES İMZASI METODUYLA GÜRÜLTÜ TANIMLAMA

ÖZET

Ses imzası metodu; gürültü tayini konusunda daha önce kullanılan bir yöntem olmamasına rağmen, bu çalışmayla binek araçlarda olabilecek gürültünün tayininde kullanılabiliceği ortaya konulmuştur.

Binek araçlarda, kullanıcalardan gelen gürültü şikeyetlerinin çözümü için eleman yetiştirilmesi hem uzun hem de maliyetli bir uygulamadır. Bu çalışmayla, şikayet nedenlerinin tayini (kimliklendirilmesi) konusunda insan faktörünün en aza indirilmesini sağlayacak bir çalışma yapılmıştır.

Basit ses kayıt cihazları ile kullanılabilir sinyal toplanabilmesine rağmen daha sofistike ölçüm aletlerini ve tekniklerinin kullanılması yararlı olucaktır. Bu çalışmada da bu ölçüm cihazları ve teknikleri kısaca anlatılmıştır.

Ses imzasının oluşturulması için öncelikle Hanning fonksiyonun yardımıyla "windowing" yapılıcaktır. Bu işlem sinyalde ortaya çıkabilecek "leakage" etkisinin ortadan kaldırılması için yapılır. Daha iyi bir karşılaştırma yapılması için Fourier Transform'u (FT) alınacaktır. Ses imzasını oluşturmada devam bölümü ise FT sı alınmış verinin bantlara bölünerek güç spektrumunun alınmadır. Ses imzası oluşturmada son bölüm bant değerlerini hexadesimal sayı olarak yazılması ve ses imzasının oluşturulmasıdır.

Veri tabanında saklanan referans ses imzaları ile testlerde alınmış olan bilinmeyen ses imzasının karşılaştırmaları ile eşlemeler yapılmaktadır.

Algoritmalar, "C+" ve C#.NET" ile yazılmış olup eklerde bulunmaktadır. Bu programların kullanımı için ".NET Framework3.5" güncellemesinin bilgisayarda yapılmış olması gerekmektedir.

Algoritmaların yardımıyla satış sonrası ortaya çıkan taşıtlardaki gürültünün tanımlama maliyetlerini düşürülmesinde etkin rol oynayabilir.

1. INTRODUCTION

1.1 Purpose of the Thesis

There are lots of methods to find out the failures and breakdowns in the automotive applications. One of these methods is noise recognition. By the way that in this method, the reasons of the noise are the reason of the failure or breakdown in the automotive applications.

These failures or breakdowns are generally marked by the customers of the vehicles such as trim noises, air condition failures etc. The customers notice what they complaining about while driving.

In such a condition, to solve failure or break down of the vehicle, high amount of money and time could be spent. If wrong diagnoses are seen, these amounts could be become higher.

During the manufacturing of the vehicle, to identify the possible failure or break down can provide to gain the time and reduce the cost. In some cases, it can be done by recognising the noise.

As a main objective of this study is the noise recognition by the audio fingerprint method. Although the audio fingerprint method is not well-known method for automotive applications, by this study the new approach for noise recognition will be gained.

This method “audio fingerprint” helps to identify the failure or breakdown of the vehicle by the noise recognition in both cases which are before and after selling the vehicle. By the way that, the cost and time, which is spent for the vehicle, can be minimized.

1.2 Background

Audio fingerprinting is not well known method in automotive sector. It is best known for its ability to link unlabeled audio to corresponding metadata (e.g. artist and song name), regardless of the audio format. Although there are more applications to audio fingerprinting, the most popular ones are:

- Content-based integrity verification
- Broadcast Monitoring,
- Connected Audio.

Audio fingerprinting systems extract a perceptual digest of a piece of audio content, i.e. the fingerprint and store it in a database. When presented with unlabeled audio, its fingerprint is calculated and matched against those stored in the database. Using fingerprints and matching algorithms, distorted versions of a recording can be identified as the same audio content [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

Most of the studies, which are about the audio fingerprint method, have the aim to identify unknown audio files. These studies are trying to improve the scripts or algorithms of building the database and efficient fingerprint search strategy [3, 4, 5, 8, 9, 10, 11, 12, 13, 14].

Generally patents for this purpose are based on fingerprinting of the audio data and search in its database too [15, 16, 17, 18, 19, 20].

As a summary audio fingerprint method contains two main parts. One of them is to create a database in which all information of audio files contains. And the second part is the matching in which unknown audio file is searched in the database to find the similar one. As it is guessed unless there a similar audio file in the database it is not possible to designated the audio file by this method.

1.3 Hypothesis

Audio fingerprint of a noise is hold its audio contents which are defined so when complained about noise by the driver of the vehicle, its fingerprint is calculated and matched against those stored in the database. Using fingerprints and matching algorithms, noise of a recording can be identified as if seen in Figure 1.1.

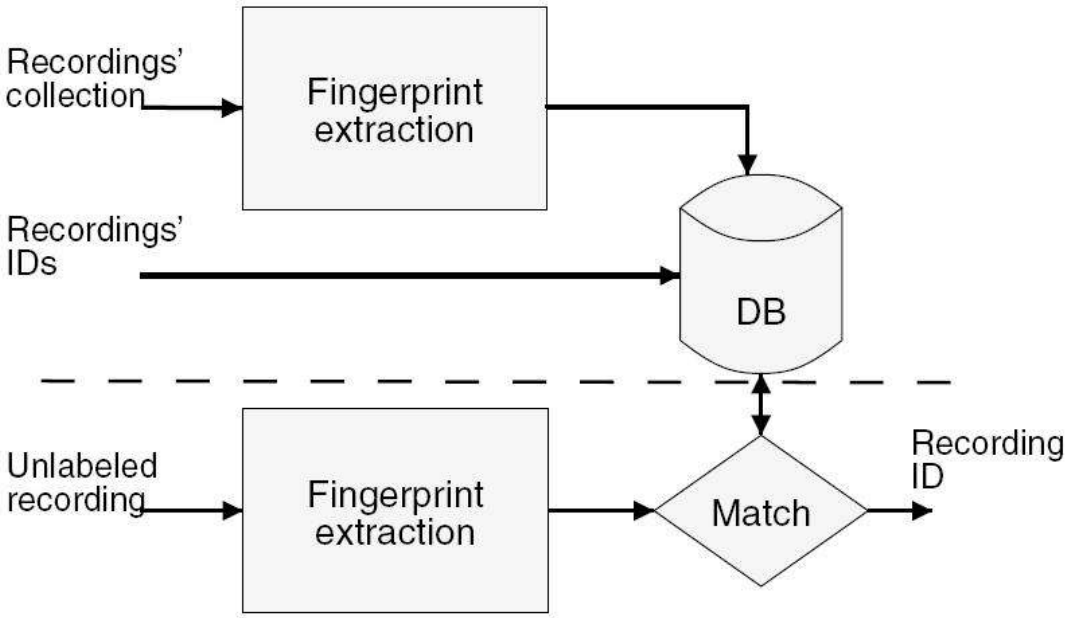

Figure 1.1 : Identification of The Audio Fingerprint [5].

The noises, which are designated before, are used to create a database and then when these defined noises are faced, with the help of audio fingerprint method they are indentified

1.4 Motivation

Audio fingerprint method brings a new approach for noise recognition in automotive applications because it is not well known method for identification of the noise in automotive sector. By this study it may give new opportunity for noise recognition in such applications.

Using the audio fingeprint method helps experiences to be stored in database so that it can take dependence of the human being away. This methot also reduces the cost and the time, which are spent fot noise recognition before and after selling to consumer.

1.5 Application Areas of Audio Fingerprint Method

Audio fingerprint method has a lot of application scenario. Here most well known ones which are tried to put in the picture.

1.5.1 Broadcast Monitoring

This application scenario - broadcast monitoring - is possibly the most well known function for audio fingerprinting method. Audio fingerprint method refers to the automatic playlist generation of Radio, TV or web shows. It is because of purposes of royalty collection, program verification, advertisement verification and people metering [11].

Meta-datas are generally incoherent, unfinished or incorrect. If the audio fingerprint database include correct meta-datas, audio fingerprinting method can make possible to know that whether or not audio data is in the library consistents and it is an easy organization based on album, artist, etc. [11].

Content distributors might want to know whether or not they have the rights to broadcast the content to consumers. Audio fingerprinting mehtod is able to assist to recognize unlabeled audio in TV and Radio channels repositories [5].

1.5.2 Conneted Audio

Connected audio is an application scenario which is generally for consumer applications where audio data are connected to any information somehow. Using a mobile phone to identify a song is one of most well known examples for this application scenario [11].

1.5.3 Integrity Verification Systems

As an application scenario, the integrity of audio recordings should be recognized before the signal can be used in fact, for example audio recording, which has not been changed or deformed, should be reassured [5].

1.6 Properties of Audio Fingerprint Method

Accuracy: The number of correct, missed or wrong identifications [1, 5].

Robustness: The capacity to accurately identify an audio data instead of noise, distortion or etc. [1, 5, 11].

Granularity (Fingerprint Size): Ability to identify from pieces which are a few seconds long [1, 5, 11].

Security: Weakness of the answer for cracking or tampering [1, 5].

Versatility: Ability to identify audio regardless of the audio format. [1, 5].

Seach Speed and Scalability: Performance with very large databases of titles or a large number of concurrent identifications [1, 5, 11].

2. MEASUREMENTS

To create a database, there is a need of proper measurement. Although simple data recorders are enough to collect the data and create a database, for better measurements more sophisticated measurement equipments are needed. For this need shortly these measurement‘s equipments and techniques are discussed here.

2.1 Measurement Equipments

2.1.1 Microphones and Preamplifiers

The acoustical pressure variations switch into electrical signals, which are amplified in a preamplifier, by the condenser microphone.

The preamplifier should always be connected very close to the microphone because its main purpose is to convert the very high impedance of the microphone into a low output impedance permitting use of long cables and connection to instruments with relatively low input impedance and noisy signals may distribute the real signal.

The low impedance makes sure that very little picks up of external electrical noise and this is especially significant while using long cables due to possibility of noise disturbing [21].

A Brüel&Kjaer type 4189 microphone, which could be used during the sound pressure measurements, is seen in Figure 2.1.

Figure 2.1 : The Microphone [22]

2.1.1.1 Microphones Types of Microphones

Microphones are divided into 3 types according to their response in the sound field.

Types of microphone are:

- Free field
- Pressure
- Random incidence

Figure 2.2 shows these types of microphone.

Free field microphones have uniform frequency response for the sound pressure that existed before the microphone was introduced into the sound field. It is of importance to note that any microphone will disturb the sound field, but the free field microphone is designed to compensate for its own disturbing presence.

The pressure microphone is designed to have a uniform frequency response to the actual sound level present. When the pressure microphone is used for measurement in a free sound field, it should be oriented at a 90° angle to the direction of the sound propagation, so that the sound grazes the front of the microphone [21].

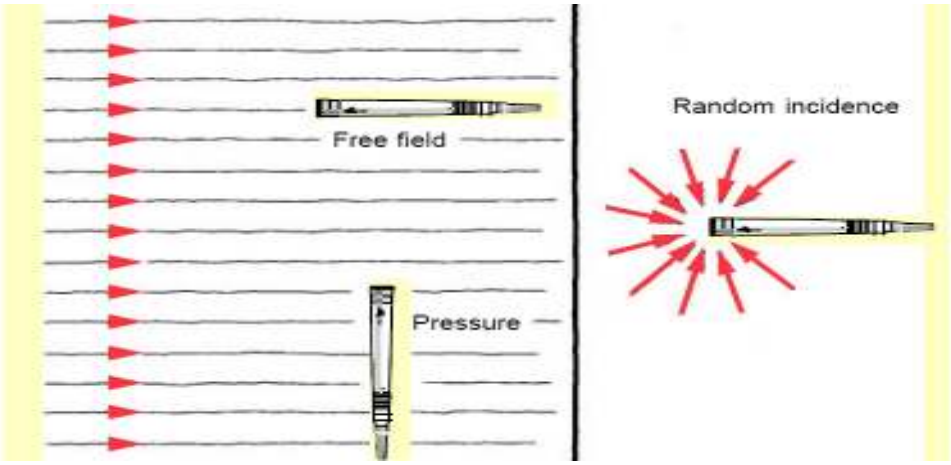

Figure 2.2 : Types of Microphones [21]

The random incidence microphone is designed to respond uniformly to signals arriving simultaneously from all angles. When used in a free field it should be oriented at an angle of 70° – 80° to the direction of propagation [21].

2.1.2 Analysis System

2.1.2.1 Front-ends

Front-ends are multichannel data acquisition units for real-time measurements on channels. Figure 2.3 shows a 5 Input Channels front-end which can be used almost all PULSE application for:

- Time data acquisition
- General noise and vibration measurements
- Basic and advance acoustics
- Structural Analysis
- Machine Diagnostics
- Electro acoustic testing

Figure 2.3 : 5 Input Channels Front-End [22]

2.1.2.2 Analyzers

With user-definable measurement solutions, all basic requirements, including data acquisition, measurement, analysis, calibration, post-processing and reporting are convenient and manageable.

FFT and CPB analyzers provides general noise and vibration testing using real-time, multichannel analysis as well as general research and development, noise and vibration analysis using several analyzers and multiple frequency spans simultaneously.

Display of functions in a range of graph types including:

- Waterfall,
- Waterfall (step),
- Colour contour,
- Bar, Line,
- Curve,
- Curve (step),
- Overlay,
- Overlay (all),
- Multi-value [21].

Recorder analyzers supply the possibility to save all data by the way that all manipulation could be done over the signal.

2.2 Measurement Technique

During experiment all conditions should be defined and satisfied by person who does the experiment.

Sound pressure measurements are taken from position which will always be the same. By the way that the noise is simulated for a specific position. This means that the measurement will be fixed.

Experiment conditions should be fixed too. The vehicle speed, the road where the vehicle goes, etc. are the experiment conditions which have to be decided before the database is created.

3. THEORY OF AUDIO FINGERPRINT METHOD

Audio fingerprint can be seen as a short summary of the signal. That's why the fingerprint maps the data with the limited number of bits instead of large number of bits.

Audio fingerprint method has two fundamental processes which are the fingerprint extraction and the matching algorithm as seen in Figure 3.1.

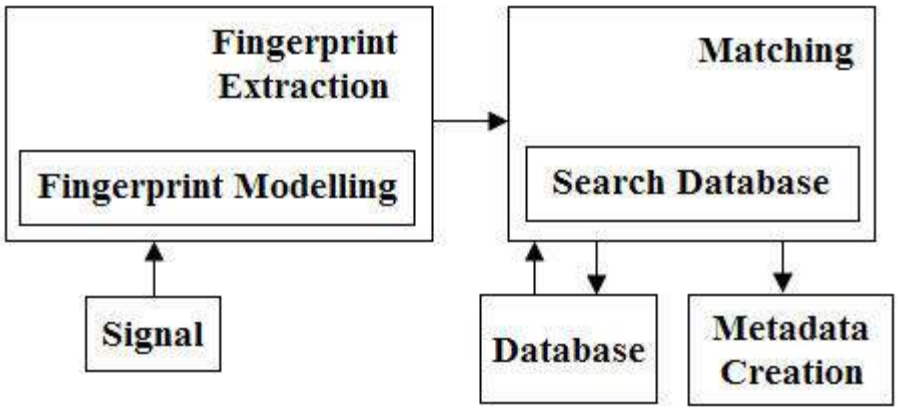

Figure 3.1 : Framework of the Audio Fingerprint Method

3.1 Fingerprint Extraction

Fingerprint extraction algorithms are generally based on the following approaches. First the audio signal is divided into parts and for every part a set of features is computed. If possible the features are selected by means of their invariance. Features that have been planned are well known audio features such as FFT, Mel frequency cepstral coefficients, spectral flatness, sharpness, linear predictive coding coefficients and etc [11].

The fingerprint needs to include:

- Discrimination power,
- Invariance to distortions,
- Compactness,
- Computational simplicity [1].

In this study the following approach, which is used for fingerprint extraction, is seen in Figure 3.2.

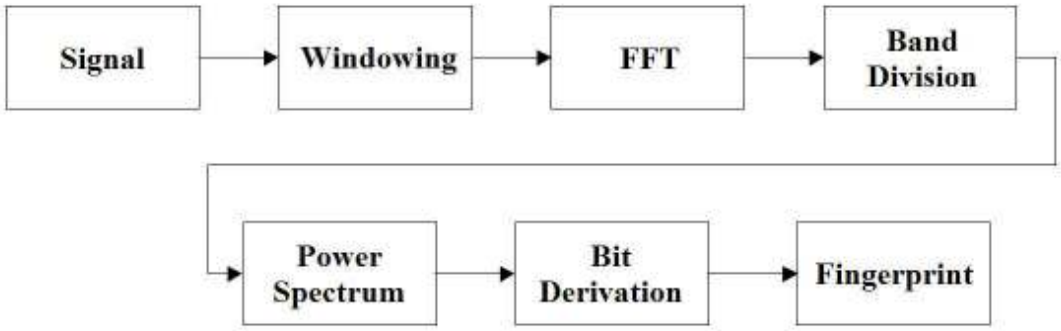

Figure 3.2 : Overview of the Audio Fingerprint Extraction

3.1.1 Windowing

In many situations, the most practical solution to the leakage problem involves the use of windowing and there are a range of different windows for different classes of problems. Windowing involves the imposition of a prescribed profile on the time signal, prior to performing the FT and the profiles or "windows" are generally depicted as a time function w(t). The analysed signal is

$$x'(t) = x(t) * w(t) \quad \textbf{(3.1)}$$

And the hanning function is

$$w(t) = 0,5 * (1 - \cos(2\pi n / N - 1)) \quad \textbf{(3.2)}$$

This produced the “improved spectrum”. The Hanning, Cosine Taper, windows are typically used for continues signals, such as are produced by steady periodic or random vibration, while the exponential window is used for transient vibration applications, where much of the important information is concentrated in the initial part of the time record and thus, would be suppressed by either of the above choices [23].

To avoid leakage effect, we have to use appropriate windowing. In this case hanning windowing function is applied which is shown in Figure 3.3.

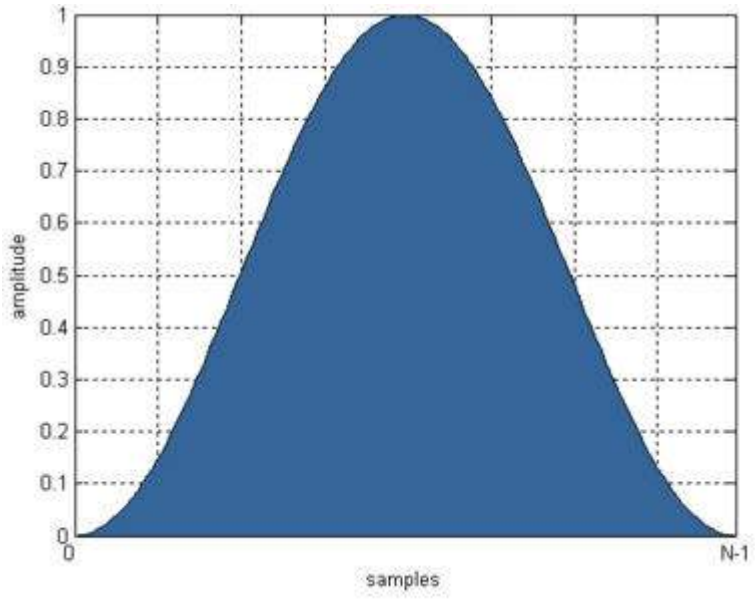

Figure 3.3 : Hanning Window Function [24]

3.1.2 Fast Fourier Transform

If the signal is periodic in time domain, the signal could be written in sum of many harmonics by the help of following functions as if it is seen Figure 3.4[23].

$$x(t) = a_0 + 2\sum_{k=1}^{\infty}(a_k \cos(\frac{2\pi kt}{T}) + b_k \sin(\frac{2\pi kt}{T})) \quad \textbf{(3.3)}$$

$$a_k = \frac{1}{T}\int_0^T x(t)\cos(\frac{2\pi kt}{T})dt, k \geq 0 \quad \textbf{(3.4)}$$

$$b_k = \frac{1}{T}\int_0^T x(t)\sin(\frac{2\pi kt}{T})dt, k \geq 1 \quad \textbf{(3.5)}$$

If the signal is not periodic in time, the signal is attached several times continuously. It means that the signal is assumed as if it is periodic in time.

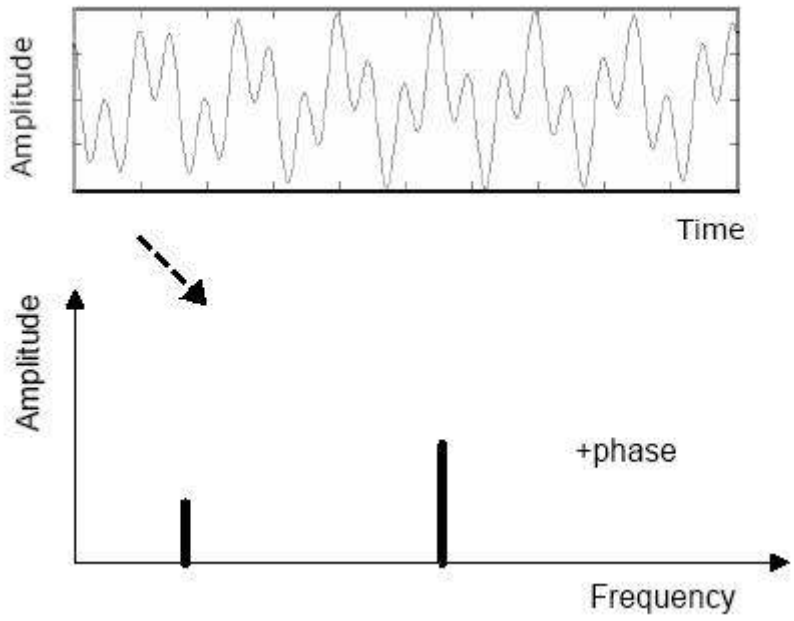

Figure 3.4 : Overview of Fourier Transform [23]

DFT is one of the specific forms of FT. DFT requires an input function that is discrete and whose non-zero values have a finite duration. A FFT is an efficient algorithm to compute the DFT. In other word practical implementation of DFT is FFT.

$$x_k = \frac{1}{N}\sum_{r=1}^{N-1} x_r e^{-i(2\pi kr/N)} \qquad \textbf{(3.6)}$$

3.1.2.1 Band Division

As seen in the Figure 3.5, for the aim of the band division, After the FFT spectrum is built, it is divided in to 1/3 Octave bands.

3.1.2.2 Power Spectrum

Power spectrum is half of the square of the amplitude as seen in the Figure 3.6.

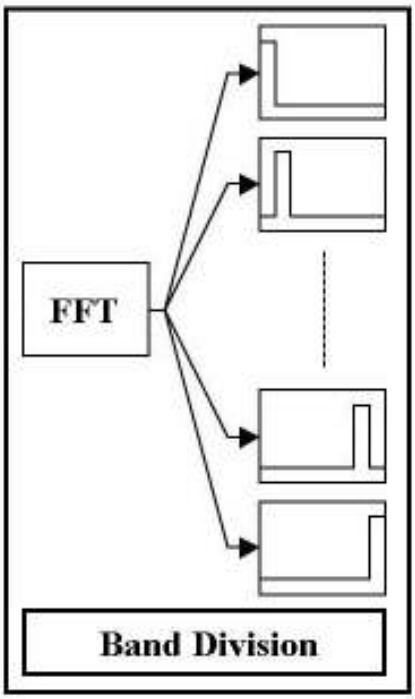

Figure 3.5 : Overview of the Band Division

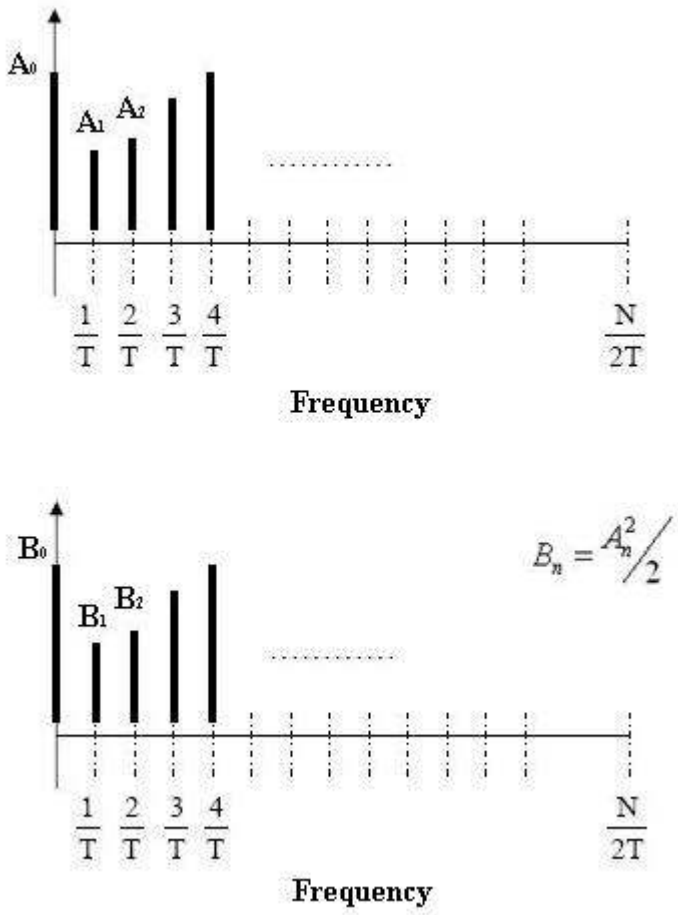

Figure 3.6 : Amplitude and Power Spectrum of the Frequencies

3.1.3 Bit Derivation

Every power spectrum values of the frequencies, after the band division and power spectrum, are given digit numbers and letters. By means of that their values are captured as seen in the Figure 3.7.

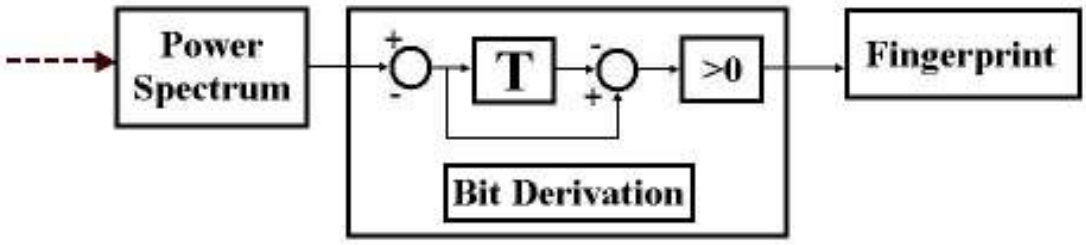

Figure 3.7 : Overview of the Bit Derivation

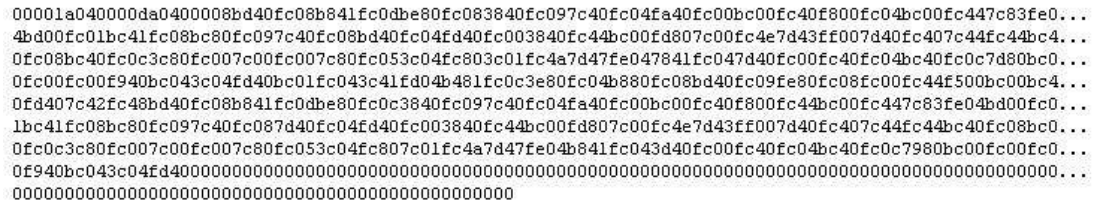

Figure 3.8 : Overview of the Audio Fingerprint Example

Audio fingerprint extracting algorithms, which are in Appendix A.1., [25, 26, 27, 28] gives values are numbers which are hexadecimal counting base for every each octave bands as seen in Figure 3.8.

3.2 Methodology of Searching and Matching

The most similar fingerprint needs to be found in the database so finding extracted fingerprints in a fingerprint database is an important task.

Identification framework follows a guide to a matching theory: both the reference the fingerprints which are stored in the database and the test the fingerprint which is extracted from the unknown data. These data are in the same format and are compared according to FFT [1].

3.2.1 String Matching

The string matching is to figure out all the positions in a given text Between two strings, the minimum number of inclusions, cuttings, and replacements are needed to convert one string to the other It is suitable to inform that is a substring matches the pattern [29].In Figure 3.9 a basic string matching scenario is seen.

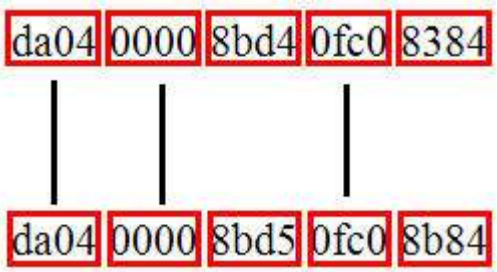

Figure 3.9 : An Configuration Between Two Strings

3.2.2 Algorithms of the Search and Matching of the Audio Fingerprints

As if seen in Figure 3.9., fingerprint divided into peaces which have 4 length values which are amplitudes of every octave bands. These values are in hexadecimal counting base so that they transform in ten counting base.

For every fingerprint, the values of the fingerprint in octave bands compare individually by taking away each other. The smaller difference of the numbers means the more similarity of the fingerprints each other.

The overview of the result of the searh and matching algorithms is seen in Figure 3.10.

Figure 3.10 : An Example Overview of the Result of the Searh and Matching

Searching algorithms of the audio fingerprint are in Appendix A.2 [27, 28].

3.3 Requirements and introduction of the Algorithms

Algorithms of the audio fingerprint extraction are written by “C+” and algorithms of the search and matching of the audio fingerprints are written by “C#.NET” These algorithms are consistent by “.NET Framework3.5”.

For the right results of the comparing, the data should be collected in same circumstances.

For using the programs, “.NET Framework3.5” should be set up.

For extraction of the audio fingerprint, the data formats are needed to be “wav”.

For better solutions the data length should be longer than 10 seconds.

3.4 An Usage of the Algorithms

There are two program which have been developed. One is named as "AudioFingerPrinting.exe" This program exract the audio fingerprint of the noise record, which is formatted as "wav (audio file format standard for storing an audio bitstream on PCs)"

To use the "AudioFingerPrinting.exe", noise data is dragged over it and then it gives the audio fingerprint of the data.

The second one is named as "SearchAndMatch.exe", which is used to compare and match the audio fingerprints which are the reference fingerprints stored in the database and the test fingerprint extracted from the unknown data

To use the "SearchAndMatch.exe", the audio fingerprint is dragged over the it. Then it asked in which database it will search as see in Figure 3.11

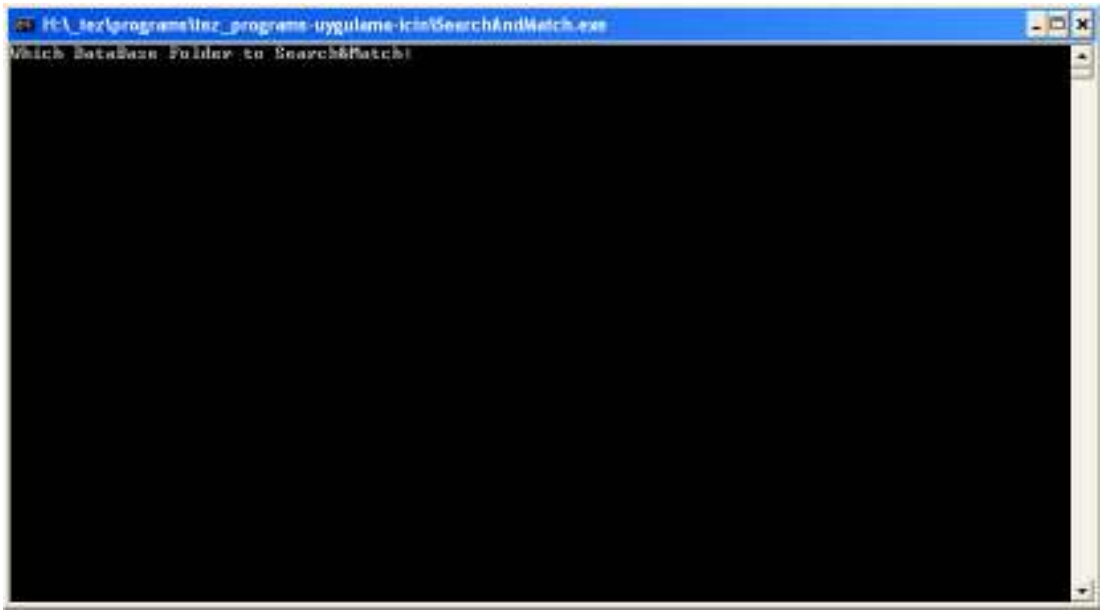

Figure 3.11 : An Example Overview "SearhAndMatching.exe"

After the database name is given the program, it give the result file which is named as "Results-Of-Search-And-Matching.txt" as seen Figure 3.12 and Figure 3.10

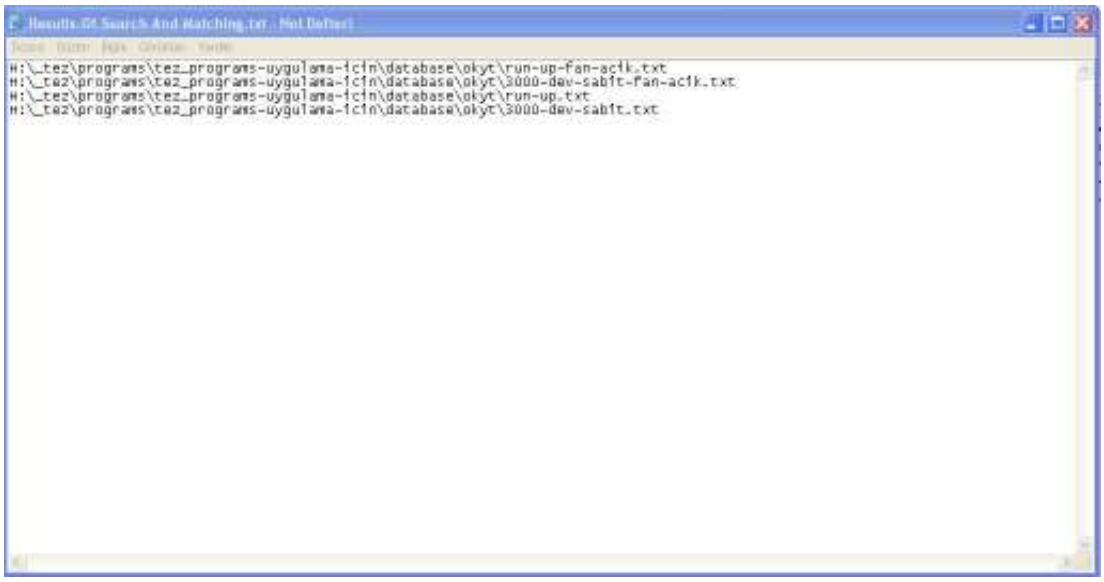

Figure 3.12 : An Example Overview “Results-Of-Search-And-Matching.txt”

4. CONCLUSION

4.1 Conclusion

Audio fingerprint method is used for only sound recognitions up to now. By the study it is shown that it can be used for noise recognitions in the automotive applications.

Easy to use algorithms of the audio fingerprint extraction, searching and matching are written by the way that anyone can use this programs while noise recognitions and needs of educated people, who has great experience for noise recognition by themselves, could be lessen in the companies of vehicle manufacturing.

Both times, which is used to recognise the noise reasons, and the cost of this work and to educate people, could be lessen.

Although the experiments and data collected could not be done. The study can be a background of usage of the audio fingerprint in noise recognitions.

4.2 Future Works

This work begin with require company against the difficulty to recognize the noise in the vehicle after and before the vehicle is sold.

To confirm the algorithms, there is a need of real industrial applications have to be done. In another words, to confirm the programs some data are needed to collected in known circumstances by the way that it can be certainly know even if the programs are worked or not. If there is need, adjusting can be done.

In some cases, in which the noise is transient, the algorithm have to be tested. Such as while the vehicle door is closing.

After confirming the algorithms, the graphical user interface will be developed as if it answers industrial needs.

REFERENCES

[1] **Cano, R., Batlle, E., Kalker, T. and Haitsma, J.,** 2002: A Review of Algorithms for Audio Fingerprinting.

[2] **Batlle, E., Cano, R., Kalker, T. and Haitsma, J.,** 2005: A Review of Audio Fingerprinting, Journal of VLSI Signal Processing, Springer Science, Netherlands.

[3] **Brun, L., Leboss´e, J., and Pailles, J.,** 2007: A Robust Audio Fingerprint's Based Identification Method, Springer-Verlag Berlin Heidelberg.

[4] **Hellmuth, O., Allamanche, E., Herre, J., Kastner, T., Cremer, M. and Hirsch, W.,** 2001 : Advanced Audio Identication Using MPEG-7 Content Description, Audio Engineering Society, New York.

[5] **Gomes, L., G´omez, E., Bonnet M., Batlle, E., Cano, R.,** 2005: Audio Fingerprinting: Concepts And Applications, Springer-Verlag Berlin Heidelberg.

[6] **Park, M., Kim, H., Shin, D., and Yang, S.,** 2005: Audio Fingerprinting Scheme by Temporal Filtering for Audio Identification Immune to Channel-Distortion

[7] **Ibarrola, A, C., and Ch´avez, E.,** 2007: Robust Audio-Fingerprinting With Spectra Entropy Signatures

[8**]** **Masip, J., Batlle, E., Guaus, E.,** Automatic Song Identification in Noisy Broadcast Audio

[9**]** **Clara, M., Serrão, C.,** 2007 : Describing Acoustic Fingerprint Technology Integration for Audio Monitoring Systems, Springer.

[10**]** **Christopher J. C., Burges, J. C., Platt and Soumya J.,** Distortion Discriminant Analysis for Audio Fingerprinting, IEEE TRANSACTIONS ON SPEECH AND AUDIO PROCESSING, VOL. XX.

[11] **Haitsma, J., and Kalker, T.,** 2002: A Highly Robust Audio Fingerprinting System.

[12] **Kimura, A., Kashino, K., Kurozumi, T., and Murase, H.,** 2007:A quick search method for audio signals based on a piecewise linear representation of feature trajectories.

[13**] Kırbız, S., Yaslan, Y., Günsel, B.,** 2005 : Robust Audio Watermark Decoding by Nonlinear Classification, ITU.

[14] **Baluja, S., Covell, M.,** 2008 : Waveprint: Efficient wavelet-based audio fingerprinting, Elsevier

[15**] US20020152078A1**, 2002 : Voiceprint identification system, United States Patent Application Publication, US.

[16**] US6384308B1**, 2004 : Method and apparatus for identifying media content presented on a media playing device, United States Patent Application Publication, US.

[17**] US6968337B2**, 2005 : Method and apparatus for identifying an unknown work, United States Patent Application Publication, US.

[18**] US5918223**, 1999 : Method and article of manufacture for content-based analysis, storage, retrieval and segmentation of audio information, United States Patent Application Publication, US.

[19**] US7363278B2**, 2008 : Copyright detection and protection system and method, United States Patent Application Publication, US.

[20**] US4833713,** 1989 : Voice recognition system, United States Patent Application Publication, US.

[21] **Girgin, Z.,** 2006 : Vehicle Booming Noise Investigation, ITU

[22] **Url-1** <*http://www.bkhome.com*>, accessed at 19.06.2008.

[23] **Şanlıturk, K. Y.,** 2008 : Data Acquisition and Signal Processing Lecture Notes

[24] **Url-2** < *http://www.wikipedia.org/>,* accessed at 02.03.2009.

[25] **Url-6** <*http://wiki.musicbrainz.org/AudioFingerprint*>, accessed at 09.02.2009.

[26] **Url-3** < *http://code.google.com/p/musicip-libofa/downloads/list*>, accessed at 02.02.2009.

[27] **Url-4** < *http://www.computerhope.com/*>, accessed at 05.02.2009.

[28] **Url-5** < *http://www.java2s.com/*>, accessed at 09.02.2009.

[29] **Cole, R. and Hariharan, R.,** 2002 : Approximate String Matching: A Simpler Faster Algorithm, Society for Industrial and Applied Mathematics, Siam J. Comput, Vol. 31, No. 6, pp. 1761–1782

APPENDICES

APPENDIX A.1 : Extracting algorithms of audio fingerprint[25, 26, 27, 28]

APPENDIX A.1

Cmmn.c

```
#include "cmmn.h"
const int bitlen(int n)
{
   int res = 0;
   while(n) {
      n = n >> 1;
      res++;
   }
   return res;
}
#if defined(SLOWROUND) || defined(WIN64)
const int round(const float x) {
   assert(x >= INT_MIN-0.5);
   assert(x <= INT_MAX+0.5);
   if (x >= 0) {
      return (int)(x+0.5);
   }
   return (int)(x-0.5);
}
#else
const int round(const float x)
{
   int a;
   __asm {
      fld x
      fistp a
   }
   return (a);
}
#endif
```

Cmmn.h

```
#ifndef CMMN_H
#define CMMN_H
#include "fd.h"
#if (_MSC_VER >= 1400)
   #pragma warning(disable : 4996)
#endif
#define FALSE       0
#define TRUE        1
#define PI          3.14159265358979323846f
#define EPSILON     1e-15
#define FRAME_LEN               8192
#define SPEC_LEN       (FRAME_LEN / 2)
#define MAX_BARK                 17
#define SSIZE       (8000 * 100)
#define IN_LEN        2048
#define FPVERSION        0
#define FPSIZE         424
struct t_fingerprint
{
   short version;
   int length;
   short avg_fit;
   short avg_dom;
   unsigned char r[348];
   unsigned char dom[66];
};
struct t_fd
{
   float window[SPEC_LEN];
   int line_to_cb[SPEC_LEN];
   int cb_start[MAX_BARK];
   int cb_size[MAX_BARK];
   int max_sfb;
   int channels;
   int samplerate;
   float *samples;
   float *sbuffer;
   int soundfound;
   float RS_ratio;
   void *RS_h;
   int outpos;
   struct t_fingerprint fp;
};
const int bitlen(int n);
int const round(const float x);
#endif
```

Fd.c

```
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <limits.h>
#include "cmmn.h"
#include "fd.h"
#include "io.h"
#include "spectrum.h"
#include "RS.h"
FDAPI struct t_fd* fp_init(int samplerate, int channels)
{
   t_fd *res = (t_fd*)malloc(sizeof(t_fd));
   if (res == NULL) {
      return NULL;
   }
   memset(&(res->fp), 0, sizeof(struct t_fingerprint));
   res->fp.version = FPVERSION;
   res->channels = channels;
   res->samplerate = samplerate;
   res->soundfound = 0;
   res->outpos = 0;
   init_sine_window(res);
   init_scales(res);
   res->samples = (float *)calloc(SSIZE, sizeof(float));
   res->sbuffer = (float *)malloc(sizeof(float) * IN_LEN);
   if (res->samples == NULL || res->sbuffer == NULL) {
      return NULL;
   }
res->RS_ratio = 8000.0f / (float)res->samplerate;res->RS_h = RS_open(FALSE, res-
>RS_ratio, res->RS_ratio);
   if (res->RS_h == NULL) {
      return NULL;
   }
   return res;
}
FDAPI int fp_feed_float(t_fd * fid, float *data, int len)
{
   int pos;
   int c;
   float accum;
   int inpos;
   int res_out;
   int in_used;
   if (len % fid->channels != 0) {
      return -1;
   }
   len = len / fid->channels;
```

```
    if (!fid->soundfound) {
        pos = 0;
        while (!fid->soundfound && pos < len) {
            for (c = 0; c < fid->channels; c++) {
                if (fabs(data[(pos * fid->channels) + c]) >= (1.0f/32768.0f - EPSILON)) {
                    fid->soundfound = TRUE;
                }
            }
            if (!fid->soundfound) {
                pos++;
            }
        }
       if (pos >= len && !fid->soundfound) {
            return TRUE;
        }
        len  = len - pos;
        data = data + (pos * fid->channels);
    }
   if (fid->outpos >= SSIZE) {
        return FALSE;
    }
    do {
        for (pos = 0; pos < len && pos < IN_LEN; pos++) {
            accum = 0;
            for (c = 0; c < fid->channels; c++) {
                accum += data[(pos * fid->channels) + c];
            }
            accum /= (float)fid->channels;
            fid->sbuffer[pos] = accum;
        }
        inpos = 0;

        do {
            res_out = RS_process(fid->RS_h, fid->RS_ratio,
                                 &(fid->sbuffer[inpos]), min(IN_LEN - inpos, len), FALSE,
                                 &in_used,
                                 &(fid->samples[fid->outpos]), SSIZE - fid->outpos);
            fid->outpos += res_out;
            inpos       += in_used;
        } while (in_used < min(IN_LEN, len) && fid->outpos < SSIZE);

        if (fid->outpos >= SSIZE) {
            return FALSE;
        }
        len  = len - in_used;
        data = data + (in_used * fid->channels);
    } while (len > 0);
    return TRUE;
}
FDAPI  int fp_feed_short(t_fd *fid, short *data, int len){
```

```
    int res;
    int i;
    float *buff;
    buff = (float*)malloc(sizeof(float) * len * fid->channels);
    if (buff == NULL) {
        return FALSE;
    }
    for (i = 0; i < len * fid->channels; i++) {
        buff[i] = data[i] / 32767.0f;
    }
    res = fp_feed_float(fid, buff, len);
    free(buff);
    return res;
}
FDAPI int fp_getversion(t_fd *fi)
{
    return FPVERSION;
}
FDAPI int fp_getsize(t_fd *fi)
{
    return FPSIZE;
}
FDAPI int fp_calculate(t_fd *fi, int songlen, unsigned char* buff)
{
    if (songlen < 1000 || !fi->soundfound || fi->outpos < (8000 * 10)) {
        return -1;
    }
    fi->fp.length = songlen;
    get_params(fi);
    memcpy(buff, &(fi->fp.version), sizeof(short));
    buff += sizeof(short);
    memcpy(buff, &(fi->fp.length), sizeof(int));
    buff += sizeof(int);
    memcpy(buff, &(fi->fp.avg_fit), sizeof(short));
    buff += sizeof(short);
    memcpy(buff, &(fi->fp.avg_dom), sizeof(short));
    buff += sizeof(short);
    memcpy(buff, &(fi->fp.r), sizeof(unsigned char) * 348);
    buff += sizeof(unsigned char) * 348;
    memcpy(buff, &(fi->fp.dom), sizeof(unsigned char) * 66);
    buff += sizeof(unsigned char) * 66;
    return 0;
}
FDAPI void fp_free(t_fd * fid)
{
    RS_close(fid->RS_h);
    free(fid->sbuffer);
    free(fid->samples);
    free(fid);
}
```

Fd.h

```
#ifndef FD_H
#define FD_H
#if defined(__cplusplus)
extern "C" {
#endif
#if defined(_WIN32) && defined(DYNAMIC)
#ifdef LIBFD_EXPORTS
#define FDAPI _declspec(dllexport)
#else
#define FDAPI _declspec(dllimport)
#endif
#else
#define FDAPI
#endif
typedef struct t_fd t_fd;
FDAPI t_fd * fp_init(int samplerate, int channels);
FDAPI void fp_free(t_fd * fid);
FDAPI int fp_feed_short(t_fd * fi, short *data, int size);
FDAPI int fp_feed_float(t_fd * fi, float *data, int size);
FDAPI int fp_getsize(t_fd *fi);
FDAPI int fp_getversion(t_fd *fi);
FDAPI int fp_calculate(t_fd *fi, int songlen, unsigned char* buff);
#if defined(__cplusplus)
}
#endif
#endif
```

Harmonic.c

```
#include <stdlib.h>
#include <stdio.h>
#include "cmmn.h"
#include "spectrum.h"
#include "Harmonic.h"
static const int quantize_harmonic(const float dom)
{
   int i;
   static float quantbord[63] = {
       15.63f,      40.04f,      44.92f,      48.83f,
       50.78f,      52.73f,      54.69f,      57.62f,
       59.57f,      62.50f,      64.45f,      66.41f,
       69.34f,      71.29f,      73.24f,      74.22f,
       77.15f,      79.10f,      82.03f,      83.01f,
       83.98f,      86.91f,      87.89f,      91.80f,
       93.75f,      97.66f,      98.63f,      99.61f,
      103.52f,     105.47f,     109.38f,     110.35f,
      111.33f,     116.21f,     119.14f,     123.05f,
      125.00f,     130.86f,     134.77f,     140.63f,
      146.48f,     151.37f,     159.18f,     165.04f,
      171.88f,     182.62f,     195.31f,     201.17f,
      218.75f,     228.52f,     247.07f,     262.70f,
      291.02f,     310.61f,     333.98f,     372.07f,
      414.06f,     461.91f,     523.44f,     592.77f,
      699.22f,     868.16f,     1176.76
   };
   for (i = 0; i < 63; i++) {
      if (dom < quantbord[i]) {
         return i;
      }
   }
   return 63;
}
void get_dominant_harmonic(const t_complex *data, int *idom){
   int i;
   int maxid = 0;
   float pwr;
   float maxpwr = 0.0f;
   float dom;
   for (i = 0; i < SPEC_LEN; i++) {
      pwr = data[i].re * data[i].re + data[i].im * data[i].im;
      if (pwr > maxpwr) {
         maxpwr = pwr;
         maxid = i;
      }
   }
   dom = 4000.0f * ((float)maxid / SPEC_LEN);
   *idom = quantize_harmonic(dom);
}
```

Harmonic.h

```
#ifndef HARMONIC_H
#define HARMONIC_H
#include "fft.h"
void get_dominant_harmonic(const t_complex *data, int *idom);
#endif
```

Regres.h

```
#ifndef REGRES_H
#define REGRES_H
void do_linear_Regres(float *dbspec, int len, float *r);
#endif
```

Regres.c

```
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "cmmn.h"
void do_linear_Regres(float *dbspec, int len, float *r)
{
   int i;
   float ssx;
   float ssy;
   float sxy;
   float ssxx;
   float ssyy;
   float ssxy;
   float avx;
   float avy;
   float rsq;
   ssx = 0.0f;
   ssy = 0.0f;
   sxy = 0.0f;
   avx = 0.0f;
   avy = 0.0f;
   for (i = 0; i < len; i++) {
      avx += i;
      avy += dbspec[i];
      ssx  += i * i;
      ssy  += dbspec[i] * dbspec[i];
      sxy  += i * dbspec[i];
   }
   avx /= (float)len;
   avy /= (float)len;
   ssxx = ssx - (float)len * avx * avx;
   ssyy = ssy - (float)len * avy * avy;
   ssxy = sxy - (float)len * avx * avy;
   if (ssyy <= 0.0f + EPSILON) {
      *r = 1.0f;
   }
   rsq = (ssxy * ssxy) / (ssxx * ssyy);
   *r = (float)sqrt(sqrt(rsq));
}
```

fft.c

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <math.h>
#include "fft.h"
#include "cmmn.h"
t_fft_data* fft_init(const int fftsize)
{
   const int tabsize = fftsize;
   int i, j;
   int powlen, logb_n;
   float e, theta;
   t_fft_data *tb;
   tb = (t_fft_data*)malloc(sizeof(t_fft_data));
   tb->twiddle_tab  = (t_twiddle*)malloc(sizeof(t_twiddle) * tabsize);
   e = (2.0f * PI) / (float)fftsize;
   for (i = 0; i < tabsize; i++) {
      theta = e * (float) i;
      tb->twiddle_tab[i].cos_t = (float)cos(theta);
      tb->twiddle_tab[i].sin_t = (float)sin(theta);
      tb->twiddle_tab[i].cos3_t = (float)cos(theta * 3.0f);
      tb->twiddle_tab[i].sin3_t = (float)sin(theta * 3.0f);
   }
    powlen = bitlen(fftsize - 1);
   logb_n = powlen >> 1;
   if ((powlen & 1) == 1) {
      logb_n++;
   }
   tb->seed_tab = (unsigned*)malloc(sizeof(unsigned) * (1 << logb_n));
   tb->seed_tab[0] = 0;
   tb->seed_tab[1] = 1;
   for (i = 2; i <= logb_n; i++) {
      for (j = 0; j < (1 << (i-1)); j++) {
         tb->seed_tab[j] <<= 1;
         tb->seed_tab[j + (1 << (i-1))] = tb->seed_tab[j] + 1;
      }
   }
   tb->size = fftsize;
   tb->work = (t_complex*)malloc(sizeof(t_complex) * fftsize);
   return tb;
}
void fft_free(t_fft_data *tb)
{
   free(tb->twiddle_tab);
   free(tb->seed_tab);
   free(tb->work);
   tb->twiddle_tab = NULL;
   tb->seed_tab = NULL;
   tb->work = NULL;
```

```
    tb->size = 0;
    free(tb);
}
static void fast_reorder(const t_fft_data *tb, t_complex *x)
{
    int i, j, n, firstj, offset, powlen;
    unsigned groupn;
    t_complex tmp;
    powlen = bitlen(tb->size - 1);
    n = 1 << (powlen >> 1);
    for (offset = 1; offset < n; offset++) {
        firstj = n * tb->seed_tab[offset];
        i = offset;
        j = firstj;
        tmp = x[i];
        x[i] = x[j];
        x[j] = tmp;
        for (groupn = 1; groupn < tb->seed_tab[offset]; groupn++) {
            i = i + n;
            j = firstj + tb->seed_tab[groupn];
            tmp = x[i];
            x[i] = x[j];
            x[j] = tmp;
        }
    }
}
static void radix_2_step(const t_fft_data *tb, t_complex *x)
{
    const int size = tb->size;
    int i0, i1, i_s, i_i;
    float R1, R2;
    i_s = 0;
    i_i = 4;
    do {
        for (i0 = i_s; i0 < size; i0 += i_i) {
            i1 = i0 + 1;
            R1 = x[i0].re;
            R2 = x[i0].im;
            x[i0].re = R1 + x[i1].re;
            x[i0].im = R2 + x[i1].im;
            x[i1].re = R1 - x[i1].re;
            x[i1].im = R2 - x[i1].im;
        }
        i_s = (i_i << 1) - 2;
        i_i <<= 2;
    } while (i_s < size);
}
static void L_step(const t_fft_data *tb, t_complex *x)
{
    const int size = tb->size;
```

```
const int halfsize = size >> 1;
int N2, N4;
int k, j;
int i0, i1, i2, i3;
int i_s, i_i;
int trig, trigexp;
float CC1, SS1, CC3, SS3, R1, S1, S2, R2, R3;
N2 = 4;
N4 = 1;
trigexp = halfsize >> 1;
for (k = 2; k < halfsize; k <<= 1) {
   i_s = 0;
   i_i = N2 << 1;
   do {
      for (i0 = i_s; i0 < size; i0 += i_i) {
         i1 = i0 +   N4;
         i2 = i0 + 2*N4;
         i3 = i1 + 2*N4;
         R3 = x[i2].re + x[i3].re;
         R2 = x[i2].re - x[i3].re;
         R1 = x[i2].im + x[i3].im;
        S2 = x[i2].im - x[i3].im;
         x[i2].re = x[i0].re - R3;
         x[i2].im = x[i0].im - R1;
         x[i3].re = x[i1].re - S2;
         x[i3].im = x[i1].im + R2;
         x[i0].re += R3;
         x[i0].im += R1;
         x[i1].re += S2;
         x[i1].im -= R2;
      }
      i_s = (i_i << 1) - N2;
      i_i <<= 2;
   } while (i_s < size);
   trig = trigexp;
   for (j = 1; j < N4; j++) {
      i_s = j;
      i_i = N2 << 1;
      CC1 = tb->twiddle_tab[trig].cos_t;
      SS1 = tb->twiddle_tab[trig].sin_t;
      CC3 = tb->twiddle_tab[trig].cos3_t;
      SS3 = tb->twiddle_tab[trig].sin3_t;
      trig += trigexp;
      do {
         for (i0 = i_s; i0 < size; i0 += i_i) {
            i1 = i0 +   N4;
            i2 = i0 + 2*N4;
            i3 = i1 + 2*N4;
            R1 = x[i2].re * CC1 + x[i2].im * SS1;
            S1 = x[i2].im * CC1 - x[i2].re * SS1;
```

```
            R2 = x[i3].re * CC3 + x[i3].im * SS3;
            S2 = x[i3].im * CC3 - x[i3].re * SS3;
            R3 = R1 + R2;
            R2 = R1 - R2;
            R1 = S1 + S2;
            S2 = S1 - S2;
            x[i2].re = x[i0].re - R3;
            x[i2].im = x[i0].im - R1;
            x[i3].re = x[i1].re - S2;
            x[i3].im = x[i1].im + R2;
            x[i0].re += R3;
            x[i0].im += R1;
            x[i1].re += S2;
            x[i1].im -= R2;
         }
         i_s = (i_i << 1) - N2 + j;
         i_i <<= 2;
      } while (i_s < size);
   }
   N2 <<= 1;
   N4   = N2 >> 2;
   trigexp >>= 1;
}
i0 =         0;
i1 =        N4;
i2 =      2*N4;
i3 = i1 + 2*N4;
while (i0 < N4) {
   CC1 = tb->twiddle_tab[i0].cos_t;
   SS1 = tb->twiddle_tab[i0].sin_t;
   CC3 = tb->twiddle_tab[i0].cos3_t;
   SS3 = tb->twiddle_tab[i0].sin3_t;
   R1 = x[i2].re * CC1 + x[i2].im * SS1;
   S1 = x[i2].im * CC1 - x[i2].re * SS1;
   R2 = x[i3].re * CC3 + x[i3].im * SS3;
   S2 = x[i3].im * CC3 - x[i3].re * SS3;
   R3 = R1 + R2;
   R2 = R1 - R2;
   R1 = S1 + S2;
   S2 = S1 - S2;
   x[i2].re = x[i0].re - R3;
   x[i2].im = x[i0].im - R1;
   x[i3].re = x[i1].re - S2;
   x[i3].im = x[i1].im + R2;
   x[i0].re += R3;
   x[i0].im += R1;
   x[i1].re += S2;
   x[i1].im -= R2;
   i0++;
   i1++;
```

```
        i2++;
        i3++;
    }
}
static void fft_proc_split(const t_fft_data *tb, t_complex *x)
{
    radix_2_step(tb, x);
    L_step(tb, x);
}
void fft(const t_fft_data *tb, t_complex *x)
{
    assert(tb != NULL);
    fast_reorder(tb, x);
    fft_proc_split(tb, x);
}
```

fft.h

```
#ifndef FFT_DEFINED
#define FFT_DEFINED
typedef struct
{
   float re;
   float im;
} t_complex;
typedef struct
{
   float cos_t;
   float sin_t;
   float cos3_t;
   float sin3_t;
} t_twiddle;
typedef struct
{
   int size;
   t_twiddle *twiddle_tab;
   unsigned *seed_tab;
   t_complex *work;
}
t_fft_data;
t_fft_data* fft_init(const int fftsize);
void fft_free(t_fft_data *tb);
void fft(const t_fft_data *tb, t_complex *x);
extern t_fft_data *xtb;
#endif
```

Spectrum.h

```
#ifndef SFM_H
#define SFM_H
#include "cmmn.h"
void get_params(t_fd *fi);
void init_sine_window(t_fd *fi);
void init_scales(t_fd *fi);
#endif
```

Spectrum.c

```
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "cmmn.h"
#include "fft.h"
#include "spectrum.h"
#include "Harmonic.h"
#include "Regres.h"
static const float toBARK(const float f)
{
   float z;
   z = ((26.81f * f) / (1960.0f + f)) - 0.53f;
   if (z < 2.0f) {
      z = z + 0.15f * (2.0f - z);
   } else if (z > 20.1f) {
      z = z + 0.22f * (z - 20.1f);
   }
   return z;
}
void init_sine_window(t_fd *fi)
{
   int i;
   for (i = 0; i < SPEC_LEN; i++) {
      fi->window[i] = (float)sqrt(0.5 - 0.5*cos(2*PI*(float)i/(FRAME_LEN)));
   }
}
static void windowize(float *window, float *smp)
{
   int i;
   for(i = 0; i < SPEC_LEN; i++) {
      smp[i] *= window[i];
   }
   for(i = SPEC_LEN; i < FRAME_LEN; i++) {
      smp[i] *= window[FRAME_LEN - i - 1];
   }
}
void init_scales(t_fd *fi)
{
   int i;
   float f;
   int lastcb;
   int cbsize;
   int cb;
   fi->cb_start[0] = 0;
   cbsize = 0;
   lastcb = 0;
   for (i = 0; i < SPEC_LEN; i++) {
```

```
        f = ((i * 8000.0f) + SPEC_LEN) / (FRAME_LEN);
        cb = (int)toBARK(f);
        if (cb > MAX_BARK - 1) {
            cb = MAX_BARK - 1;
        }
        fi->line_to_cb[i] = cb;
        if (cb != lastcb) {
            fi->cb_start[lastcb + 1] = i;
            fi->cb_size[lastcb] = cbsize;
            lastcb++;
            cbsize = 0;
        }
        cbsize++;
    }
    fi->cb_size[lastcb] = cbsize;
    fi->max_sfb = lastcb + 1;
}
static void get_dbpower(t_complex *work, float *dbpower)
{
    int i;
    float power;

    for (i = 0; i < SPEC_LEN; i++) {
        power = (work[i].re * work[i].re) + (work[i].im * work[i].im);

        if (power <= EPSILON) {
            dbpower[i] = 0.0f;
        } else {
            dbpower[i] = (float)log(power) * 4.34294480f;
        }
    }
}
static int quantize_r(const float r, const int band)
{
    const static float q1[MAX_BARK] = {
        0.8116f,
        0.4273f,    0.4233f,    0.3827f,    0.3783f,
        0.3848f,    0.3726f,    0.3669f,    0.3500f,
        0.3440f,    0.3281f,    0.3256f,    0.3148f,
        0.3091f,    0.3031f,    0.3340f,    0.5660f
    };

    const static float q2[MAX_BARK] = {
        0.8528f,
        0.5300f,    0.5267f,    0.4800f,    0.4765f,
        0.4853f,    0.4711f,    0.4635f,    0.4430f,
        0.4356f,    0.4147f,    0.4117f,    0.3993f,
        0.3892f,    0.3825f,    0.4151f,    0.6249f
    };
    const static float q3[MAX_BARK] = {
```

```
        0.8824f,
        0.6216f,    0.6220f,    0.5754f,    0.5736f,
        0.5838f,    0.5699f,    0.5595f,    0.5374f,
        0.5281f,    0.5032f,    0.4983f,    0.4850f,
        0.4674f,    0.4612f,    0.4929f,    0.6746f
    };
    if (r < q1[band]) {
        return 0;
    }
    if (r < q2[band]) {
        return 1;
    }
    if (r < q3[band]) {
        return 2;
    }

    return 3;
}
void get_params(t_fd *fi)
{
    t_fft_data *fft_data;
    int i, j;
    int frames;
    int ansize;
    float r[MAX_BARK];
    int qr[MAX_BARK];
    float dbpower[SPEC_LEN];
    int counts[4];
    int doms[88];
    int domidx;
    int idom;
    int total_dom;
    float avg_dom;
    float avg_qr;
          fft_data = fft_init(FRAME_LEN);
    ansize = (8000 * 90);
    frames = ansize / FRAME_LEN;
    counts[0] = 0;
    counts[1] = 0;
    counts[2] = 0;
    counts[3] = 0;
    total_dom = 0;
    memset(doms, 0, sizeof(int) * 88);
    for (i = 0; i < frames; i++) {
        windowize(fi->window, &(fi->samples[i * FRAME_LEN]));
        for (j = 0; j < FRAME_LEN; j++) {
            fft_data->work[j].re = fi->samples[i * FRAME_LEN + j];
            fft_data->work[j].im = 0.0f;
        }
        fft(fft_data, fft_data->work);
```

```
      get_dbpower(fft_data->work, dbpower);
      for (j = 1; j < fi->max_sfb; j++) {
         do_linear_Regres(&dbpower[fi->cb_start[j]], fi->cb_size[j], &r[j]);
         qr[j] = quantize_r(r[j], j);
      }
      get_dominant_harmonic(fft_data->work, &idom);
      total_dom += idom;
      for (j = 1; j < fi->max_sfb; j++) {
         counts[qr[j]]++;
      }
      fi->fp.r[(i*4)]   = (qr[1] << 6)  | (qr[2] << 4)  | (qr[3] << 2)  | qr[4];
      fi->fp.r[(i*4)+1]  = (qr[5] << 6)  | (qr[6] << 4)  | (qr[7] << 2)  | qr[8];
      fi->fp.r[(i*4)+2]  = (qr[9] << 6)  | (qr[10] << 4) | (qr[11] << 2) | qr[12];
      fi->fp.r[(i*4)+3]  = (qr[13] << 6) | (qr[14] << 4) | (qr[15] << 2) | qr[16];
      doms[i]   = idom;
   }
   domidx = 0;
   for (i = 0; i < 87; i += 4) {
      fi->fp.dom[domidx++] = (doms[i] << 2)          | (doms[i+1] >> 4);
      fi->fp.dom[domidx++] = (doms[i+1] & 0xF) << 4   | (doms[i+2] >> 2);
      fi->fp.dom[domidx++] = ((doms[i+2] & 0x3) << 6) | (doms[i+3]);
   }
   avg_dom = (float)total_dom / (float)frames;
   avg_qr  = ((1.0f * counts[1]) + (2.0f * counts[2]) + (3.0f * counts[3]))
           / ((float)frames*(float)(fi->max_sfb-1));
   fi->fp.avg_dom = round(avg_dom *  100.0f);
   fi->fp.avg_fit = round(avg_qr  * 1000.0f);
   fft_free(fft_data);
}
```

Filter.c

```
#include "RS_defs.h"
#include "Filter.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#define IzeroEPSILON 1E-21
tatic float Izero(float x)
{
  float sum, u, halfx, temp;
  int n;
  sum = u = n = 1;
  halfx = x/2.0f;
  do {
    temp = halfx/(float)n;
    n += 1;
    temp *= temp;
    u *= temp;
    sum += u;
  } while (u >= IzeroEPSILON*sum);
  return(sum);
}
void LpFilter(float c[], int N, float frq, float Beta, int Num)
{
  float IBeta, temp, inm1;
  int i;
  c[0] = 2.0f*frq;
  for (i=1; i<N; i++) {
    temp = PI*(float)i/(float)Num;
    c[i] = sin(2.0f*temp*frq)/temp;
  }
   IBeta = 1.0f/Izero(Beta);
  inm1 = 1.0f/((float)(N-1));
  for (i=1; i<N; i++) {
    temp = (float)i * inm1;
    c[i] *= Izero(Beta*sqrt(1.0f-temp*temp)) * IBeta;
  }
}
float FilterUp(float Imp[],
               float ImpD[],
               UWORD Nwing,
               BOOL Interp,
               float *Xp,
               float Ph,
               int Inc)
{
  float *Hp, *Hdp = NULL, *End;
  float a = 0;
  float v, t;
```

```
  Ph *= Npc;
  v = 0.0;
  Hp = &Imp[(int)Ph];
  End = &Imp[Nwing];
  if (Interp) {
    Hdp = &ImpD[(int)Ph];
    a = Ph - floor(Ph);
  }
  if (Inc == 1)
  {
    End--;
    if (Ph == 0)
    {
      Hp += Npc;
      Hdp += Npc;
    }
  }
  if (Interp)
    while (Hp < End) {
      t = *Hp;
      t += (*Hdp)*a;
      Hdp += Npc;
      t *= *Xp;
      v += t;
      Hp += Npc;
      Xp += Inc;
    }
  else
    while (Hp < End) {
      t = *Hp;
      t *= *Xp;
      v += t;
      Hp += Npc;
      Xp += Inc;
    }
  return v;
}
float FilterUD(float Imp[],
               float ImpD[],
               UWORD Nwing,
               BOOL Interp,
               float *Xp,
               float Ph,
               int Inc,
               float dhb)
{
  float a;
  float *Hp, *Hdp, *End;
  float v, t;
```

```
  float Ho;
  v = 0.0;
  Ho = Ph*dhb;
  End = &Imp[Nwing];
  if (Inc == 1)
  {
    End--;
    if (Ph == 0)
      Ho += dhb;
  }
  if (Interp)
    while ((Hp = &Imp[(int)Ho]) < End) {
      t = *Hp;
      Hdp = &ImpD[(int)Ho];
      a = Ho - floor(Ho);
      t += (*Hdp)*a;
      t *= *Xp;
      v += t;
      Ho += dhb;
      Xp += Inc;
    }
  else
    while ((Hp = &Imp[(int)Ho]) < End) {
      t = *Hp;
      t *= *Xp;
      v += t;
      Ho += dhb;
      Xp += Inc;
    }
  return v;
}
```

Filter.h

```
#include "RS_defs.h"
float FilterUp(float Imp[], float ImpD[], UWORD Nwing, BOOL Interp,
        float *Xp, float Ph, int Inc);
float FilterUD(float Imp[], float ImpD[], UWORD Nwing, BOOL Interp,
        float *Xp, float Ph, int Inc, float dhb);
void LpFilter(float c[], int N, float frq, float Beta, int Num);
```

RS.c

```
#include "RS.h"
#include "RS_defs.h"
#include "Filter.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
typedef struct {
  float  *Imp;
  float  *ImpD;
  float   LpScl;
  UWORD   Nmult;
  UWORD   Nwing;
  float  minFactor;
  float  maxFactor;
  UWORD   XSize;
  float  *X;
  UWORD   Xp;
  UWORD   Xread;
  UWORD   Xoff;
  UWORD   YSize;
  float  *Y;
  UWORD   Yp;
  float  Time;
} rsdata;
void *RS_open(int highQuality, float minFactor, float maxFactor){
  float *Imp64;
  float Rolloff, Beta;
  rsdata *hp;
  UWORD   Xoff_min, Xoff_max;
  int i;
    if (minFactor <= 0.0 || maxFactor <= 0.0 || maxFactor < minFactor) {
     #if DEBUG
     fprintf(stderr,
           "libRS: "
           "minFactor and maxFactor must be positive real numbers,\n"
           "and maxFactor should be larger than minFactor.\n");
     #endif
     return 0;
  }
  hp = (rsdata *)malloc(sizeof(rsdata));
  hp->minFactor = minFactor;
  hp->maxFactor = maxFactor;
  if (highQuality)
    hp->Nmult = 35;
  else
    hp->Nmult = 11;
  hp->LpScl = 1.0f;
  hp->Nwing = Npc*(hp->Nmult-1)/2;
```

```
  Rolloff = 0.90f;
  Beta = 6;
  Imp64 = (float *)malloc(hp->Nwing * sizeof(float));
  LpFilter(Imp64, hp->Nwing, 0.5f*Rolloff, Beta, Npc);
  hp->Imp = (float *)malloc(hp->Nwing * sizeof(float));
  hp->ImpD = (float *)malloc(hp->Nwing * sizeof(float));
  for(i=0; i<hp->Nwing; i++)
    hp->Imp[i] = Imp64[i];
  for (i=0; i<hp->Nwing-1; i++)
    hp->ImpD[i] = hp->Imp[i+1] - hp->Imp[i];
  hp->ImpD[hp->Nwing-1] = - hp->Imp[hp->Nwing-1];
  free(Imp64);
  Xoff_min = ((hp->Nmult+1)/2.0f) * MAX(1.0f, 1.0f/minFactor) + 10;
  Xoff_max = ((hp->Nmult+1)/2.0f) * MAX(1.0f, 1.0f/maxFactor) + 10;
  hp->Xoff = MAX(Xoff_min, Xoff_max);
  hp->XSize = MAX(2*hp->Xoff+10, 4096);
  hp->X = (float *)malloc((hp->XSize + hp->Xoff) * sizeof(float));
  hp->Xp = hp->Xoff;
  hp->Xread = hp->Xoff;
  for(i=0; i<hp->Xoff; i++)
    hp->X[i]=0;
  hp->YSize = (int)(((float)hp->XSize)*maxFactor+2.0);
  hp->Y = (float *)malloc(hp->YSize * sizeof(float));
  hp->Yp = 0;
  hp->Time = (float)hp->Xoff;
  return (void *)hp;
}
int RS_get_filter_width(void   *handle){
  rsdata *hp = (rsdata *)handle;
  return hp->Xoff;
}
int RS_process(void   *handle,
               float  factor,
               float  *inBuffer,
               int    inBufferLen,
               int    lastFlag,
               int    *inBufferUsed,
               float  *outBuffer,
               int    outBufferLen){
  rsdata *hp = (rsdata *)handle;
  float  *Imp = hp->Imp;
  float  *ImpD = hp->ImpD;
  float  LpScl = hp->LpScl;
  UWORD  Nwing = hp->Nwing;
  BOOL interpFilt = FALSE;
  int outSampleCount;
  UWORD Nout, Ncreep, Nreuse;
  int Nx;
  int i, len;
  #if DEBUG
```

```
fprintf(stderr, "RS_process: in=%d, out=%d lastFlag=%d\n",
       inBufferLen, outBufferLen, lastFlag);
#endif
*inBufferUsed = 0;
outSampleCount = 0;
if (factor < hp->minFactor || factor > hp->maxFactor) {
  #if DEBUG
  fprintf(stderr,
        "libRS: factor %f is not between "
        "minFactor=%f and maxFactor=%f",
        factor, hp->minFactor, hp->maxFactor);
  #endif
  return -1;
}
if (hp->Yp && (outBufferLen-outSampleCount)>0) {
  len = MIN(outBufferLen-outSampleCount, hp->Yp);
  for(i=0; i<len; i++)
    outBuffer[outSampleCount+i] = hp->Y[i];
  outSampleCount += len;
  for(i=0; i<hp->Yp-len; i++)
    hp->Y[i] = hp->Y[i+len];
  hp->Yp -= len;
}
 if (hp->Yp)
  return outSampleCount;
if (factor < 1)
  LpScl = LpScl*factor;
for(;;) {
  #ifdef DEBUG
  printf("XSize: %d Xoff: %d Xread: %d Xp: %d lastFlag: %d\n",
       hp->XSize, hp->Xoff, hp->Xread, hp->Xp, lastFlag);
  #endif
  len = hp->XSize - hp->Xread;
  if (len >= (inBufferLen - (*inBufferUsed)))
    len = (inBufferLen - (*inBufferUsed));
  for(i=0; i<len; i++)
    hp->X[hp->Xread + i] = inBuffer[(*inBufferUsed) + i];
  *inBufferUsed += len;
  hp->Xread += len;
  if (lastFlag && (*inBufferUsed == inBufferLen)) {
    Nx = hp->Xread - hp->Xoff;
    for(i=0; i<hp->Xoff; i++)
      hp->X[hp->Xread + i] = 0;
  }
  else
    Nx = hp->Xread - 2 * hp->Xoff;
  #ifdef DEBUG
  fprintf(stderr, "new len=%d Nx=%d\n", len, Nx);
  #endif
  if (Nx <= 0)
```

```
        break;
      if (factor >= 1) {
        Nout = SrcUp(hp->X, hp->Y, factor, &hp->Time, Nx,
                  Nwing, LpScl, Imp, ImpD, interpFilt);}
      else {
        Nout = SrcUD(hp->X, hp->Y, factor, &hp->Time, Nx,
                  Nwing, LpScl, Imp, ImpD, interpFilt);
      }
      #ifdef DEBUG
      printf("Nout: %d\n", Nout);
      #endif
      hp->Time -= Nx;
      hp->Xp += Nx;
      Ncreep = (int)(hp->Time) - hp->Xoff;
      if (Ncreep) {
        hp->Time -= Ncreep;
        hp->Xp += Ncreep;      }
      Nreuse = hp->Xread - (hp->Xp - hp->Xoff);
      for (i=0; i<Nreuse; i++)
        hp->X[i] = hp->X[i + (hp->Xp - hp->Xoff)];
      #ifdef DEBUG
      printf("New Xread=%d\n", Nreuse);
      #endif
      hp->Xread = Nreuse;
      hp->Xp = hp->Xoff;
      if (Nout > hp->YSize) {
        #ifdef DEBUG
        printf("Nout: %d YSize: %d\n", Nout, hp->YSize);
        #endif
        return -1;}
      hp->Yp = Nout;
      if (hp->Yp && (outBufferLen-outSampleCount)>0) {
        len = MIN(outBufferLen-outSampleCount, hp->Yp);
        for(i=0; i<len; i++)
          outBuffer[outSampleCount+i] = hp->Y[i];
        outSampleCount += len;
        for(i=0; i<hp->Yp-len; i++)
          hp->Y[i] = hp->Y[i+len];
        hp->Yp -= len;}
      if (hp->Yp)
        break;}
    return outSampleCount;}
  void RS_close(void *handle){
    rsdata *hp = (rsdata *)handle;
    free(hp->X);
    free(hp->Y);
    free(hp->Imp);
    free(hp->ImpD);
    free(hp);
  }
```

RS.h

```
#ifndef LIBRS_INCLUDED
#define LIBRS_INCLUDED
#ifdef __cplusplus
extern "C" {
#endif
void *RS_open(int      highQuality,
              float   minFactor,
              float   maxFactor);
int RS_get_filter_width(void *handle);
int RS_process(void   *handle,
               float  factor,
               float  *inBuffer,
               int    inBufferLen,
               int    lastFlag,
               int    *inBufferUsed,
               float  *outBuffer,
               int    outBufferLen);
void RS_close(void *handle);
#ifdef __cplusplus
}
#endif
#endif
```

Configwin.h

```
/* Run configure to generate config.h automatically on any
   system supported by GNU autoconf.  For all other systems,
   use this file as a template to create config.h
*/
#undef HAVE_INTTYPES_H
```

APPENDIX A.2 : Searching algorithms of the audio fingerprint [27, 28]

APPENDIX A.2

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
namespace SearchAndMatch
{
   class Program
   {
      public struct results
      {
         public int diff;
         public string filename;
      }
      public static results[] Results = new results[1024];
      public static int[] fftNumbers = new int[212];
      static void Main(string[] args)
      {
         File.Delete("Results-Of-Search-And-Matching.txt");
         foreach (string path in args)
         {
            if (File.Exists(path))
            {
               ReadFile(path);
            }
         }
         for ( ; ; )
         {
            try
            {
               RunSearchAndMatch();
               SortResults(Results);
               for (int i = 0; i < Results.Length; i++)
               {
                  if (Results[i].filename != null)
                  {
                     Console.WriteLine("{0}", Results[i].filename);
                     AddResultToFile(Results[i].filename);
                  }
               }
               System.Diagnostics.Process.Start("notepad.exe", "Results-Of-Search-
And-Matching.txt");
               break;
            }
            catch
            {
               Console.WriteLine("File not found or invalid file type !");
            }
         }
```

```
        }
    public static void RunSearchAndMatch()
    {
        Console.Write("Which DataBase Folder to Search&Match: ");
        string Path = string.Empty;
        Path = Console.ReadLine();
ProcessDirectory(System.IO.Path.GetDirectoryName(System.Reflection.Assembly.
GetExecutingAssembly().Location) + "\\database\\" + Path);
    }
    public static void ProcessDirectory(string targetDirectory)
    {
        int x = 0;
        string[] fileEntries = Directory.GetFiles(targetDirectory);
        foreach (string fileName in fileEntries)
        {
            SearchAndMatch(fileName, x);
            x++;
        }
       }
    public static void ReadFile(string fileName)
    {
        string str = string.Empty;
        int index = 0;
        FileStream fsFile = new FileStream(fileName, FileMode.Open,
FileAccess.Read);
        StreamReader srReader = new StreamReader(fsFile);
        str = srReader.ReadLine();
        for (int i = 0; i < str.Length; i += 4)
        {
            fftNumbers[index++] = Convert.ToInt32(str.Substring(i, 4), 16);
            }
        srReader.Close();
        fsFile.Close();
    }
    public static void SearchAndMatch(string fileName, int x)
    {
        string str = string.Empty;
        int index = 0;
        FileStream fsFile = new FileStream(fileName, FileMode.Open,
FileAccess.Read);
        StreamReader srReader = new StreamReader(fsFile);
        str = srReader.ReadLine();
        for (int i = 0; i < str.Length; i += 4)
        {
            Results[x].filename = fileName;
            Results[x].diff += Math.Abs(fftNumbers[index++] -
Convert.ToInt32(str.Substring(i, 4), 16));
            }
        srReader.Close();
        fsFile.Close();
```

```
        }
        public static void SortResults(results[] result)
        {
            for (int i = 0; i < result.Length - 1; i++)
            {
                for (int j = i + 1; j < result.Length; j++)
                {
                    if (result[j].diff < result[i].diff)
                    {
                        results temp = new results();
                        temp.diff = result[i].diff;
                        temp.filename = result[i].filename;
                        result[i].diff = result[j].diff;
                        result[i].filename = result[j].filename;
                        result[j].diff = temp.diff;
                        result[j].filename = temp.filename;
                    }
                }
            }
        }
        public static void AddResultToFile(string rslt)
        {
            try
            {
                FileStream fsResultFile = new
FileStream(System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExe
cutingAssembly().Location) + "\\Results-Of-Search-And-Matching.txt",
FileMode.Append, FileAccess.Write);
                StreamWriter swWrite = new StreamWriter(fsResultFile);
                swWrite.WriteLine(rslt);
                swWrite.Flush();
                swWrite.Close();
                fsResultFile.Close();
            }
            catch { }
        }
    }
  }
}
```

CURRICULUM VITA

Candidate's full name:	***Emin ERENSOY***
Place and date of birth:	**Istanbul – May 15th 1984**
Permanent Address:	**Kozyatağı – Istanbul/TURKEY**
Universities and Colleges attended:	**Istanbul Technical University – Mechanical Engineering**