Dedication

بسم الله الرحمن الرحيم

الحمد لله ربّ العالمين و الصلاة و السّلام على خاتم الانبياء و المرسلين. بعد الانتهاء من هذا العمل لا بدّ من شكر كل من ساهم في انجاحه. و لذلك نبدء بشكر و حمد الاحقّ بالحمد، فالحمد لله ربّ العالمين، الذّي هو وحده المعين، و الذّي اعاننا على تأديته و اكماله على الوجه الانسب.

ثمّ كما قال الحبيب المصطفى، صلّى الله عليه و سلّم : من لا يشكر النّاس لا يشكر الله. فلا بدّ من شكر كلّ من السيدة صوفية بن جبارة و السيد فتحي التّليلي على توجيهاتهما القّيمة التّي اعاننا الله بها على اتمام هذا العمل. و الحمد لله اوّلا و اخيرا.

محمد على الجمعاوي و محمد أمين الحزّامي

2012

Contents

List of Tables

List of Figures

Introduction

This work has been done as a project of secondary year of communication engineering studies in the highest school of communication of Tunisia.

In this report we study ***the effect of GSM codec on the quality of pathological voice*** with the aim of determining the correction functions to the modification introduced by the GSM codec on human pathological voice's parameters in order to retreive the original ones. Thus, the voice parameters accuray will be increased which will allow the implementation of the solution provided in a real time embedded application.

The approach adapted for determining the correction functions is explained further in depth within the first and second chapters. The first chapter dresses an overview of the whole study context. At first, the problematic behind this study is explained. Second, the adopted methodology to resolve this problem will be explained.

For further understanding of the study, the second and third parts of the first chapter will give the fundamentals needed to carry on within this study: the second part will give some of the fundamentals of the human voice, how it is produced and what causes its pathology. Then the third part will give a quick overview about the GSM codec in order to understand the alteration origins. This first chapter will be closed by an overview of the voice databases and the toolset used within this work.

The second chapter will study further in depth the problematic behind this work. At the very begining, an explanation of the voice parameters is given. Then, the second part explains the process of GSM codec simulation and the results will be shown at the third part. Last but not least, a study of the performance of the determined correction functions will be issued.

The implementation of the results of the study in a embedded system application will be the subject of the third chapter. The first part of this chapter explains the conception of the application and the adopted methodology of development. It also gives an overview of the implementation of the solution on the Visual DSP ++ development environment as a simulation of ADSP-BF533 processor. Then , a study of the performance of the implementation is given in the second part. Third part of this chapter will try to optimize the implementation based on the results of the performance analysis.

Chapter 1

Problematic & fundamentals

This chapter gives an overview to the study. First, it presents the problematic behind this work and dresses the resolution approach that has been adopted. Second, an overview of the human voice and GSM codec fundamentals is given in order to enhance the understanding of the inners of this study. Last but not least, the voice database and the toolset used in this study are presented.

1.1 Problematic & Resolution Approach

In the new era of revolutionnary communication techologies several studies have been launched, with the aim of getting more usability of the progress that has been made in communication technologies to simplify more and more the human life. This study is one of the attempts of bringing the usability of the communication technologies advances to a closer manner to the human life. In this work we try to make an integration of the GSM technology to the medical care field.

Figure 1.1: The use of GSM network for voice diagnostics

The system we try to make, aims at allowing a voice therapist of studying the progress of his patient's situation via GSM network. In fact, a voice therapist can determine whether a subject's voice is pathologic or not by the study of its voice parameters such as jitter and shimmer[1].

[1]refer to section 2.1 to understand voice parameters

Figure 1.1 illustrates the use of GSM network in voice diganostic. Using the developed system, a patient will no longer need personnally to his therapist. He would use the GSM network to allow the doctor to diagnostic his voice. First, The subject would call his doctor. The doctor would get a sample of the subject's voice while pronouncing the sustained voice "aaa". From this point, the developed system will issue a number of procedures, including voice parameters calculus and correction, to dispaly at the end a description of voice parameters close to the original values. Thus permitting a more accurate voice diagnostic.

1.1.1 Problematic

As explained above, the system to be implemented will use a subject's voice resulting of a transmission over the GSM network. It, then, calculates the voice parameters that a therapist will use later to determine whether a subject's voice is pathological or not. This process is described in figure 1.1.

The problem encountred in this system is that the accuray of the voice parameters calculated from the voice resulting of the transmission over the GSM network is altered. This alteration is due to the accumulation of several errors within the GSM transmission chain. Thus, the analysis a therapist would make, based on the results of the described system, will not be accurate, which may cause faulty recommendation to be given to the subject in question. The system that this work studies aims at correcting the alteration of the GSM codec. In fact, to resolve this issue, the present work adopted a methodology that permits to invert the effect of the modification introduced by the GSM transmission mechanism to the studied subject's voice parameters.

In order to invert the effect of the modification introduced by the GSM codec on the voice parameters, we need to determine the modification functions that modelise the GSM codec influence. Then, this modification can be inverted by simply inverting the modification function. The diagram in figure 1.2 shows a model of the resolution approach that has been adopted in this work.

Figure 1.2: A block diagram of the resolution methodology

- ***in*** **:** the pathological voice audio file of the patient being talking over the GSM network
- ***TO GSM*** **:** Transmission of pathological voice over the GSM network
- ***A in*** **:** Altered pathological voice after transmission over GSM network
- ***VPC*** **:** Voice Parameters Calculus based on the A in
- ***EVP*** **:** Errenous Voice Parameters
- ***GSM EI*** **:** GSM Error Inversion
- ***CVP*** **:** Corrected Voice Parameters

Before starting the resolution process, some of the fundamentals related to the human voice and the GSM codec will be presented so that the rest of this work can be understood.

1.2 Human Voice

In order to understand speech codec, to analyse the features of the GSM modification to the human voice and to implement the solution, it is important to have some knowledge on the basic features of human speech. So how the human voice is produced ? What causes its pathology ? The next two parts will try to respond to these questions.

1.2.1 Voice Production

When a person starts speaking, an aeordynamic process collaborates with muscular forces to produce voice. In fact, by the use of muscular force, air is moved from the lungs through the vocal tract which extends from the glottis to the mouth, including three different cavities : pharyngeal cavity, nasal cavity and mouth cavity[1]. What happens during speech production process is depicted in figure 1.3.

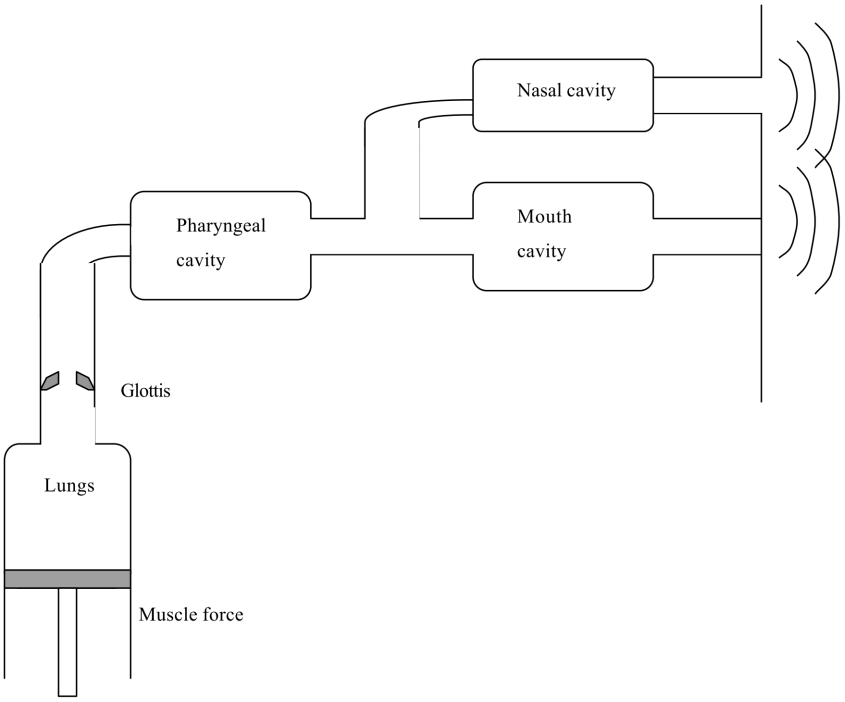


Figure 1.3: A block diagram of the human speech production

Sound is produced when the glottis, which is an opening in the vocal cords, vibrates. This interrupts the flow of air and creates a sequence of impulses, which have some basic frequency called the pitch. With males this frequency is typically between 80-160 Hz and with females it is between 180-320 Hz. A typical speech signal is shown in the figure 1.4 which illustrates a pseudo-periodic nature. This is due to the pitch of the sound.

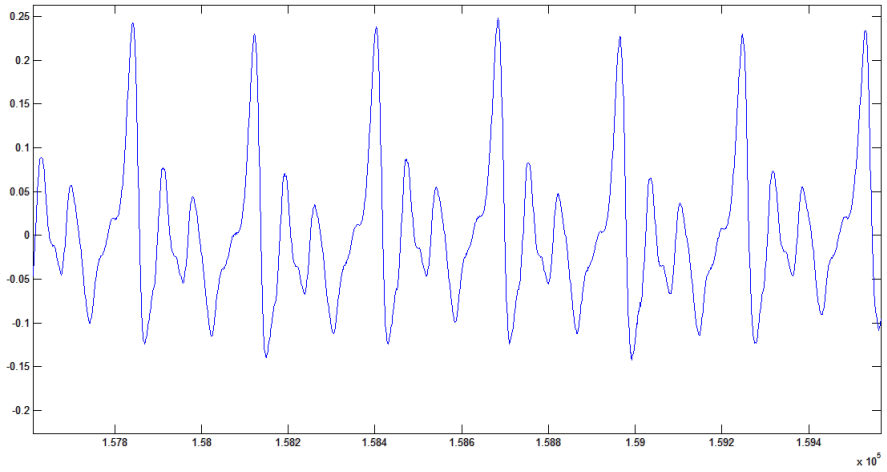


Figure 1.4: Example of a femle speech in time-domain: vocal 'aaa'

The spectrum of the sound is formed in the vocal tract. The strongest of the frequency components in the speech are called formants. The frequencies in the spectrum of the sound are controlled by varying the shape of the tract, for example by moving the tongue. For further understanding of the voice production mechanism, see *the anatomy of the human vocal tract* in figure 1.5.

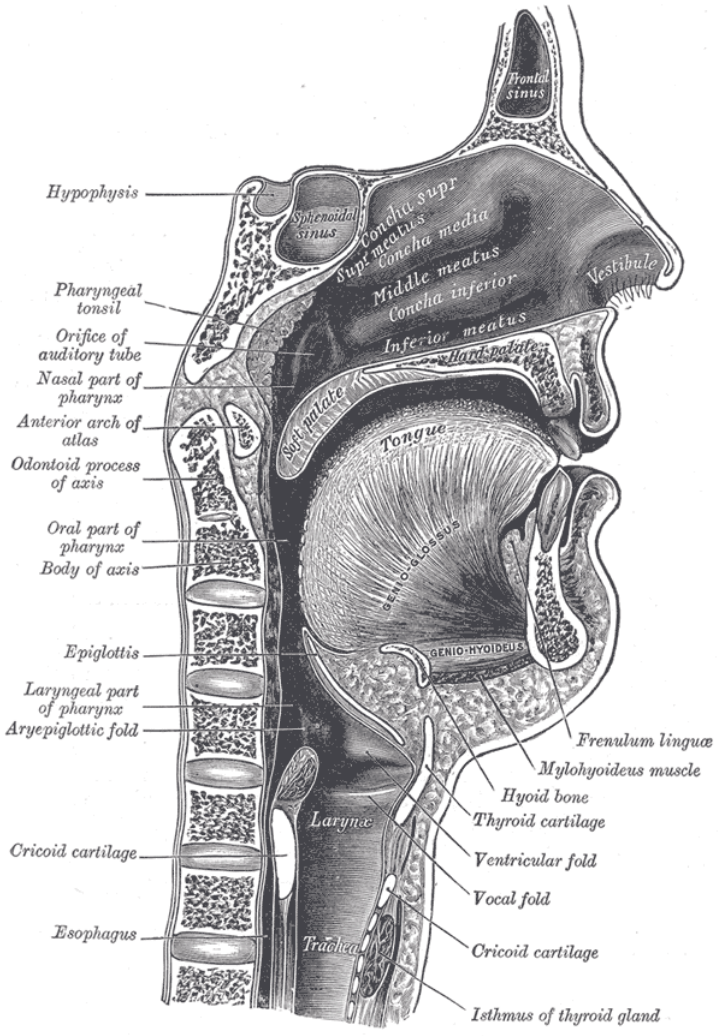
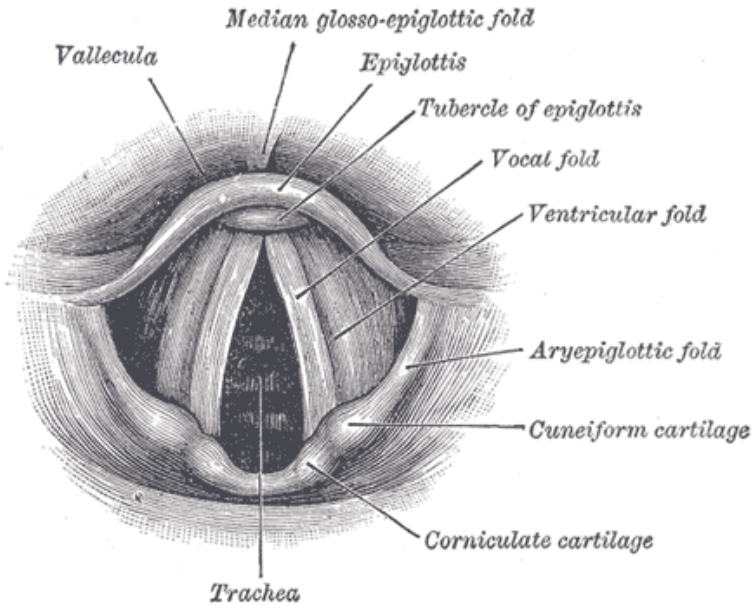


Figure 1.5: Anatomy of the human vocal tract[2]

1.2.2 Voice Pathology

As explained in the previous part, voice production starts with the vibration of the vocal folds which can be more or less streched to acheive higher or lower pitch tones. In normal conditions and in spite of this pitch variation ability, phonation is considered stabilized and regular.

Any transformation of the vocal fold's tissue can cause an irregular, non-periodic vibration which will change the shape of the glottal source signal from one period to the next[3]. The same problem can occur in amplitude. If, for instance, the vocal folds are too stiff, they will need a higher sub-glottal pressure to vibrate. The glottal cycle can thus be irregularly distributed also in amplitude.



[5]

Figure 1.6: Vocal folds[2]

Not less important is the existence of high frequency noise, especially during the closed phase of the glottal cycle, originated by a partial closure of the vocal folds, which will cause air leakage through the glottis, providing a turbulence effect. All these phenomena affect the glottal source signal, only to the sound pressure radiated at the lips.

Vocal folds pathology can be classified in two categories: functional pathology or organic pathology. The functional pathologies category includes malfunctional organs without the presence of any organic disease what so ever. Whereas, the organic pathology categorie includes modifications and diseases that may occur to the vocal folds system[4].

1.2.3 Organic Pathologies

Organic voice pathologies are due to modification anywhere in the vocal tract, especially alteration that affects the vocal folds system. This category includes many voice pathology types. Table1.1 resumes some specific examples related to organic voice pathologies :

Voice Pathology	Description
Severe infectious voice pathology	It is caused by a viral or bacterial infection and may also be caused by an allergic effect. This pathology is often caused by some irritant predisoposing factors (tabacco, alcohol , dust, vapors, voice mishandling, moisture, cold inflections rhinosinusiennes).
Spasmodic dysphonia	It is part of dystnias, if the laryngeal musculature is affected. This pathology is related to a disorder in the acetylcholine release of the basal ganglia.
Exceptional pathologies	These class includes exceptional infectious diseases like syphilis, laryngeal tuberculosis, AIDS, lupus or larngeal exceptional tumors : osteoma of the vocal cord, chondroma, chondrosarcoma.

Table 1.1: Organic Voice Pathologies

1.2.4 Functional Pathologies

Functional pathologies does not occur due to any alteration in the vocal tract system tissue. It does occur without any modification what so ever, to the tissue. This voice pathology categorie can be divided in two classes : vocal folds without lesion and vocal folds with lesion. Table 1.2 resumes a description of functional pathologies :

Voice Pathology	Description
Vocal folds without lesion	Speech ability alteration or dysphonia affect one or more voice parameters. A singer voice alteration is called dysodie. Vocal folds are normal, still there is a malfunctioning in the ”vocal gesture” and a vibrational disorder.
Vocal folds with lesion	This pathology occurs as a result of a violent laryngeal effort, which may be even brief!. This is the case of sports fan screaming in a stadium, the case of loud cry after an attack .

Table 1.2: Functional Voice Pathologies

Figure 1.7 illustrates some of these exceptional voice pathologies . These examples are taken from the Vocal Pathology Image Library of the Greater Baltimore Medical Center[2].

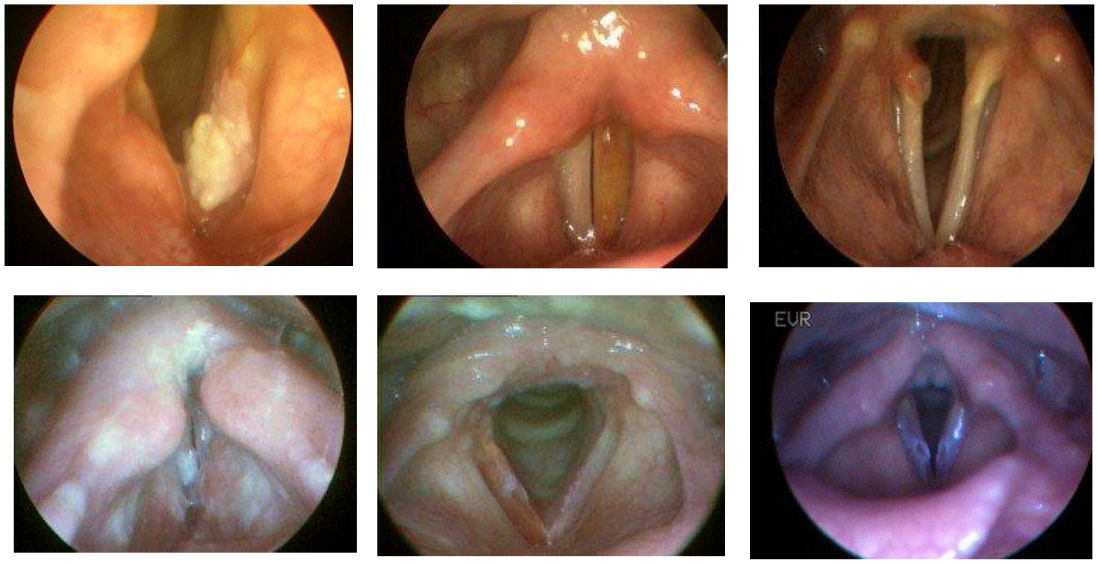


Figure 1.7: Voice pathology, in order, Benign Tumor, Hemorrhage, Intubation Granuloma, Fungal Infection, Contact Ulcers

1.3 GSM Codec

The conversion of analogue speech waveform into digital form is done by sampling the original anlog signal then followed by speech coding. A major benefit of using speech coding is the possibility to compress the signal, that is, to reduce the bit rate of the digital speech. In other words, speech codec represents speech with as few bits as possible, while maintaining a speech quality that is acceptable[1].

In the subsequent sections, the inners of the GSM codec are first explained in order to gain some understanding of the basic principles in speech coding. After that, the origins behind the error that a GSM codec based systems introduces are further analysed in order to get a clearer view of the problem.

1.3.1 Inner Functionnality

The GSM system uses Linear Predicitve Coding with Regular Pulse Excitation (LPC-RPE codec). It is a full rate speech codec and operates at 13 kbits/s. As a comparaison, the public telephone networks use speech coding with bit rate of 64 Kbit/s. Despite this there is no significant difference in the speech quality.

The voice signal is divided by the GSM encoder into blocks of 20 ms. Each speechblock contains 260 bits as depicted in the diagram of figure 1.8. This is reasonable since 260 bits/20 ms = 13Kbits/s. The encoder has three major parts:

[2]the source website of these images : `http://www.gbmc.org/home_voicecenter.cfm?id=1563`

- linear prediction analysis (short-term prediction)
- long-term prediction
- excitation analysis

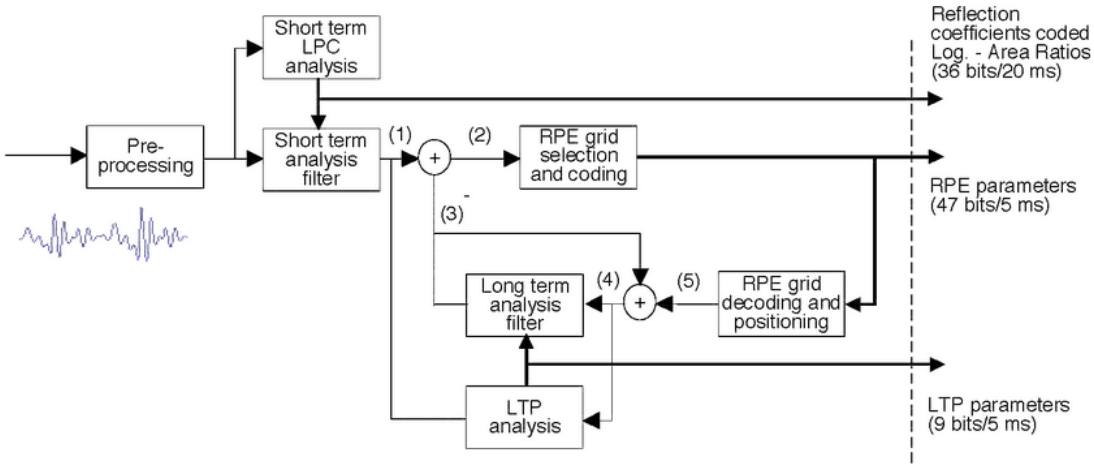


Figure 1.8: a diagram presentation of the GSM Full-Rate LPC-RPE codec[1]

Linear prediction uses transfer function of order 8. Each coefficient is encoded into 8 bits which gives 36 bits for this part. The long-term predictor decomposes each frame into 4 subframes. Then, it estimates pitch and gain for each subframe of duration 5ms intervals.

The long-term predictor estimates pitch and gain four times at 5ms intervals. Each estimate provides a lag coefficient and gain coefficient of 7 bits and 2 bits, respectively. Together these four estimates require $4(7+2)bits = 36bits$. The gain factor in the predicted speech sample ensures that the synthesised speech has the same energy level as the original speech signal[1].

The remaining 188 bits are derived from the regular pulse excitation analysis. After both short and long term filtering the residual signal, that is the difference between the predicted signal and the actual signal, is quantized for each subframe.

As for any other codec, the GSM codec introduces a certain destorsion between original speech and the one reconstructed after coding/decoding. The amount of error is not crutial for the conversation to be understood from both parts using the GSM network. But, for the purpose of this study, its very crutial to reduce the amount of error introduced on the pathological voice parameters, as any alteration to the parameters values could result in faulty diagnositc.

1.4 Voice Database & Toolset

In order to determine the modification introduced by the GSM codec to the pathological voice features, a series of experiments have been realized for a number of pathological voices. The simulation of the GSM codec influence has been realized within the Linux OS. All of these tools and others, that have been used in the theorital study of the problem are presented in the following subsections.

1.4.1 Sustained vowels database

All the simulation experiments described in this report have been performed with two sustained voice databases. The first database is a collection of 57 sample of sustained vowels voice 'aaa' which is composed of 24 sample of pathological voices, the remaining are healthy voices. This database is an extraction from pathological and healthy sustained "aaa" database made by a Spanish research group.

The second database contains a larger number of samples, few are sustained voices. From the second database it has been extracted around 10 samples to enlarge the previous extraction and get a more accurate result. This database is part of a larger database taken from a CD associated of D.G . Childers entitled **Speech Processing and Synthesis Toolboxes** from the JHON WILLEY & SONS, INC publications Edition 2000.

The types of pathologies coverd in these two databases include functional and organic pathologies.

- ***functional pathologies*:** such as mechanical agression, chemical irretation and psychogenic dyphonia.
- ***organic pathologies*:** such as chronic inflammatory laryngitis, tumor of larynx, dysphonia and immobility.

1.4.2 Toolset

A collection of Open Sourced free of charge and Commercial tools have been used in order to realize the experiments described in this work. First, the host environment for GSM codec simulation will be presented. Second, the use of the Praat Software for acoustic analysis will be explained. Finally, the Matlab numerical environment will be presented.

Linux-Ubuntu 11.04 & GSM 06.10

In order to determine the GSM codec influence on the voice parameters, the Ubuntu[7] Linux Open Sourced Operating System has been used as the host for the GSM codec simulation application to run. In fact, the GSM 06.10 lossy speech compression library[8] is an installable package from the Linux Teminal. By simply typing the command line ***$sudo apt-get install libgsm-tools*** , the application that simulates the GSM codec is installed. After the installation two commands are added to the Linux Operating System :

- ***toast*** : this is the major linux command-line used in this study for the purpose of simulating the GSM codec. In fact, the command-line ***toast -p audio_file.au***[3] will generate as output a new file with the gsm extension ***audio_file.au.gsm***. This new file is a gsm coded audio file from which we can extract another au file via the command-line ***toast -pd audio_file.au***. The restituted au file is not the

[3]the -p command switch tells the command-line to conserve the original file and write a new one otherwise it will be erased.

same as the original file. That is due to the errors introduced by the GSM codec.
- ***untoast*** : it is similar to the *toast -d* command, so this command hasn't accually been used in this study, its alternative *toast -d* has been used instead for the purpose of decoding the gsm file.

For instance, the Voice databases that have been used for this study are in **WAV** format. When working with GSM codec's toast command-line, this audio file format is not supported. In order to carry the experiments, the WAV format audio files needed to be converted to another file format supported by the simualtion tool. This file format is, as you may noticed above, the **AU** file format. The AU file format stands for ***Sun Microsystems Audio file format***. To convert the whole database to the specified file format, two solution has been determined.

First, by the use of the sox linux command-line, which is an audio manipulation command-line the WAV audio files have been converted to AU format. For convenience purposes, a Matlab script that do the same task has been conceived.

The amount of audio data used in this study is relatively large, the process of gsm codec has been automated using the Linux Shell Programming scripting language. Following is illustrated the shell code that has been used for automating the simulation.

GSM encoding script

GSM encoding script
```
#!/bin/bash
for i in *.au
do
toast -p $i
done
```

GSM decoding script

GSM encoding script
```
#!/bin/bash
for i in *.gsm
do
toast -pd $i
done
```

Praat Software for Acoustic Analysis

The study of the voice parameters has been done using the Praat software for actoustic analysis[9]. As mentioned above, it is an open sourced software that can be gotten freely from the official website. Praat has been used in this study in order to determine the voice parameters of the studied subject. A typical use of Praat is shown in the figure 1.9.

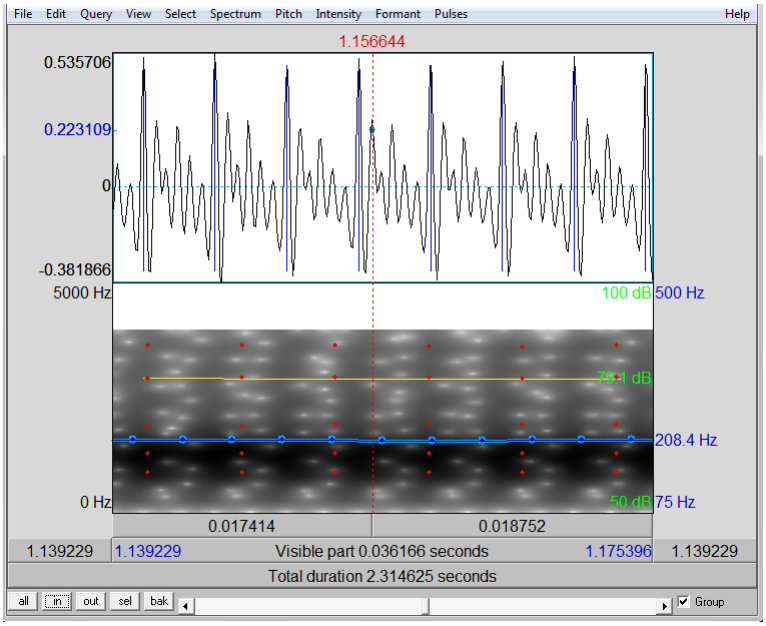


Figure 1.9: Illustration of the use of Praat in Voice Analysis

By choosing a selection from the studied voice within the Praat interface, then going to ***voice report*** Praat give a list of 28 voice parameters that have been determined via algrithms, such as pulse detection algorithm, jitter and shimmer estimation algorithms. A typical voice report is shown in the following illustration.

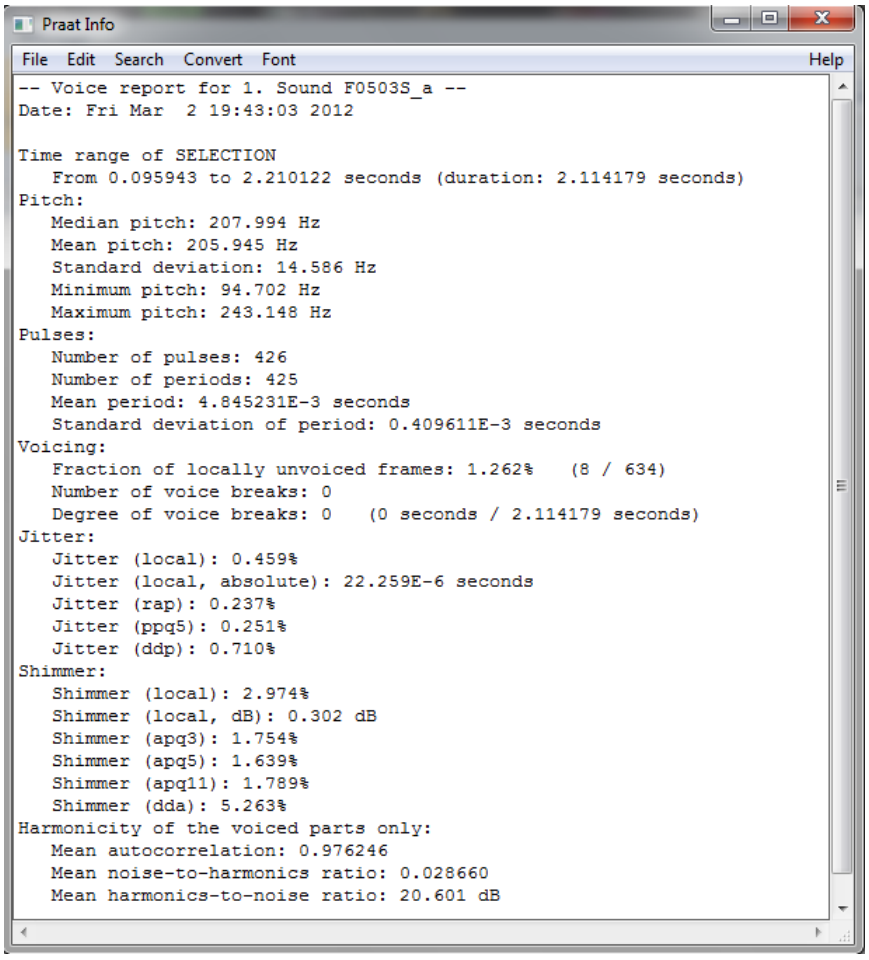


Figure 1.10: Illustration of Praat's Voice Report

Matlab

The well known numerical computing environment Matlab has been largely used to fullfill the present work. In fact, Matlab has been used to extract from the simulation results the modification functions and the correction functions. Also, it has been used in the design and implementation of the embedded solution.

In some cases the Praat software for acoustic analysis has not been accurate. In fact, the implemented algorithm for pulse detection in Praat is not well fit for pathological voice analysis. For instance, in working with normal voices, the Praat software has been used, otherwise, for matters of accuracy, another more accurate algorithm for pulse detection have been computed and used for pathological voice analysis. However, the same approach that Praat uses for determining voice parameters has been conserved.

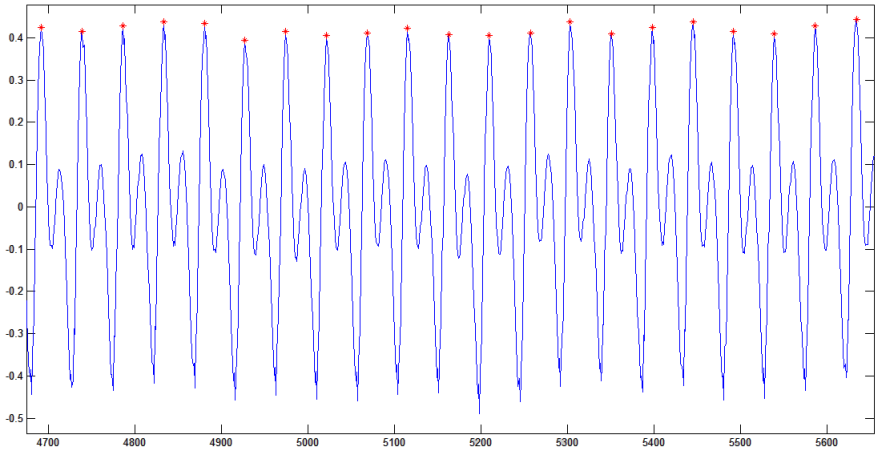


Figure 1.11: Pulse Detection Algorithm in Matlab

Algorithms that make the inner functionnality of the embedded application like the ***Pulse Detection Algorithm*** or ***Voice Parameters Calculus Algorithm*** have been, first, designed and simulated in Matlab, then implemented within Visual DSP ++ environment for embedded application development.

1.5 Conclusion

In this chapter the purpose and the problematic behind this study has been presented. Also, a board outline of the scientific, technical and medical fundamentals that helps to understand the developed system, like human voice production and the GSM codec functionality, have been dressed. Afterward, a description of the toolset used within this study has been given. Now we are ready to start the description of the realized experiments and present the found results.

Chapter 2

Effect of GSM codec on voice parameters

This chapter resumes the major functionality of the developed system. It, first, starts with a description of the voice parameters that are mostly used in pathological voice diagnosis. Second comes an explanation of the modification functions determination process which is concluded with a summary of the voice parameters's modification functions. Then an explanation of the correction functions determination approach is given. Finally, the last part studies the performance of the determined correction functions.

2.1 Voice parameters

The major criteria used in this study to allow a voice therapist to determine whether a subject's voice is pathological or not are voice parameters such as Jitter and Shimmer. For instance, Jitter and Shimmer are mesures of the cycle-to-cycle variations of fundamental frequency and amplitude, respectively, which have largely used on the description of pathological voice quality. If the jitter(relative) value of subject's voice is greater then 1.04%[1] or if the shimmer value of a subject's voice is greater then 3.810%, the subject's voice is considered to be pathological. Thus, the therapist would give his recommendation for the therapy. The next part explains the analysis of these features based on the formulas used by the Open Sourced Praat acoustic analysis software.

2.1.1 Jitter measurements

- *jitter (absolute)* is the cycle-to-cyle variation of fundamental frequency, i.e. The average absolute difference between consecutive periods expressed as :

$$jitter(absolute) = \frac{1}{N-1}\sum_{i=1}^{n}|T_i - T_{i+1}| \tag{2.1}$$

  where T_i are the extracted periods and N is the number of periods on the speech sequence.

[1] see praat documentation

- *jitter (relative)* is the average absolute difference between consecutive periods, divided by the average period. It is expressed as a percentage:

$$jitter(relative) = \frac{\frac{1}{N-1}\sum_{i=1}^{n}|T_i - T_{i+1}|}{\frac{1}{N}\sum_{i=1}^{N}T_i} \tag{2.2}$$

- *jitter (rap)* is defined as the Relative Average Perturbation, the average of it and its two neighbours, divided by the average period.

$$jitter(rap) = \frac{\frac{1}{N-1}\sum_{i=1}^{N}|T_i - \frac{T_{i+1}+T_i+T_{i-1}}{3}|}{\frac{1}{N}\sum_{i=1}^{N}T_i} \tag{2.3}$$

- *jitter(ppq5)* is the five-point Period Perturbation Quotient, computed as the average absolute difference between a period and the average of it and its four closest neighbours , divided by the average period.

$$jitter(ppq5) = \frac{\frac{1}{N-1}\sum_{i=1}^{N}|T_i - \frac{\sum_{k=i-2}^{i+2}T_k}{5}|}{\frac{1}{N}\sum_{i=1}^{N}T_i} \tag{2.4}$$

- *jitter(ppq55)* is the 55-point Period Perturbation Qutient, computed as the avarage asbolute difference between a period and the average of it and its four closed neighbours, divided by the average period.

$$jitter(ppq55) = \frac{\frac{1}{N-1}\sum_{i=1}^{N}|T_i - \frac{\sum_{k=i-27}^{i+27}T_k}{55}|}{\frac{1}{N}\sum_{i=1}^{N}T_i} \tag{2.5}$$

2.1.2 shimmer measurements

- *Shimmer(dB)* is expressed as the variability of the peak-to-peak amplitude in decibels, i.e. The average absolute base-10 logarithm of the difference between the amplitudes of consecutive periods, multiplied by 20:

$$Shimmer(dB) = \frac{1}{N-1}\sum_{i=1}^{N-1}|20\log(A_{i+1}/A_i)| \tag{2.6}$$

  where A_i are the extracted peak-to-peak amplitude data and N is the number of extracted fundamental frequency periods.

- *Shimmer(relative)* is defined as the average absolute difference between the amplitudes of consecutive periods, divided by the average amplitude, expressed as a percentage:

$$Shimmer(relative) = \frac{\frac{1}{N-1}\sum_{i=1}^{n}|A_i - A_{i+1}|}{\frac{1}{N}\sum_{i=1}^{N}A_i} \tag{2.7}$$

- *Shimmer(apq3)* is the three-point Amplitude Perturbation Quotient, the average absolute difference between the amplitude of a period and the average of the amplitudes of its neighbours, divided by the average amplitude.

$$Shimmer(apq3) = \frac{\frac{1}{N-1}\sum_{i=1}^{N}|A_i - \frac{A_{i+1}+A_i+A_{i-1}}{3}|}{\frac{1}{N}\sum_{i=1}^{N}A_i} \tag{2.8}$$

- *Shimmer(apq5)*is defined as the five-point Amplitude Perturbation Quotient, the average absolute difference between the amplitude of a period and the average of the amplitudes of it and its four closest neighbours, divided by the average amplitude.

$$Shimmer(apq5) = \frac{\frac{1}{N-1}\sum_{i=1}^{N}|A_i - \frac{\sum_{k=i-2}^{i+2}A_k}{5}|}{\frac{1}{N}\sum_{i=1}^{N}A_i} \tag{2.9}$$

- *shimmer(apq55)* is expressed as the 55-point Amplitude Perturbation Quotient, the average absolute difference between the amplitude of a period and the average of the amplitudes of it and its closest neighbours, divided by the average amplitude.

$$Shimmer(apq55) = \frac{\frac{1}{N-1}\sum_{i=1}^{N}|A_i - \frac{\sum_{k=i-27}^{i+27}A_k}{55}|}{\frac{1}{N}\sum_{i=1}^{N}A_i} \tag{2.10}$$

2.1.3 Other voice parameters

Other voice paramters has been used in this study to enhance the accuracy of the therapist analysis of the voice pathology such as the standard deviation of pitch and the pitch itself. In the table 2.1 a list of these vocie parameters is dressed.

Voice Parameter	**mathematic description**
Mean Pitch	$\overline{F} = \frac{1}{N}\sum_{i=1}^{N} Fi$
Minimum pitch	$\min(F_i)$
Maximum pitch	$max(F_i)$
Standard deviation of pitch	$\frac{1}{N-1}\sum_{i=1}^{N}\|F_i - \overline{F}\|$
Amplitude average	$\frac{1}{N}\sum_{i=1}^{N} Ai$

Table 2.1: Other voice parameters modelisation

2.2 Analysis of GSM codec modification

In order to determine the voice parameters's correction functions the modification introduced by the GSM codec need to be modilized. Thus, the mechanism of the GSM codec has been simulated. The next part explains the simulation methodology and the following part resumes the results of the simulation. Then, the process of modification functions determination will be studied further in depth later in this section.

2.2.1 Simulation steps

The simulation of the GSM codec has been done thanks to the GSM 6.0 lossy compression library. The encoding step of the codec is simulated via the ***toast*** command-line and the decoding step is simulated via the command-line ***toast -pd audio_file.au***. Figure 2.1 shows a block diagram of the simulation of the GSM codec.

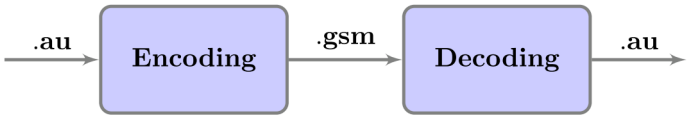


Figure 2.1: a block diagram of the GSM codec Simulation

For a more detailed explaination of the GSM codec simulation process refer to subsection 1.4.2 on page 16.

2.2.2 Modification functions analysis

After simulating the GSM codec, the next subsection illustrates the process of modification functions determination.

To analyse the GSM codec influence to the pathological voice, Matlab has been used. For instance, say that the studied voice parameter is the Mean pitch. The process of analysis is as follows: a vector called **mean_pitch_org** has been declared where the values of mean pitch, before GSM codec, for the whole database's voice sequences are stored. After simulating the GSM codec on the same mentioned database, the values of mean pitch, after GSM codec, for the whole database are stored within a new vector called **mean_pitch_mod**.

Now, two vectors containing the mean pitch values for the database before and after GSM codec are prepared. As illustrated in figure 2.2, using Matlab's **plot** utility the mean pitch evolution before and after the GSM codec can be seen. In blue is the mean pitch evolution before GSM codec and in red is the evolution of mean pitch after GSM codec.

The following code extraction shows the methology of Matlab usage in the modification function analysis.

Modification function analysis

```
figure;
Mean_pitch_org = [206.732350 269.740407 .....
188.846251 209.762003 ];
a = Mean_pitch_org;
subplot(311);
plot(a);
title('Mean_pitch_org');
hold on
figure;
Mean_pitch_org = [210.587135 268.060037 .....
188.853496 209.787623 ];
b = Mean_pitch_org;
plot(b,'r');
```

From the two graphical descriptions of the evolution of mean pitch before and after simulation, we see clearly an alteration to the original voice parameters values due to the GSM codec. Thus, we need to determine a modelisation to this modification in order to invert its influence.

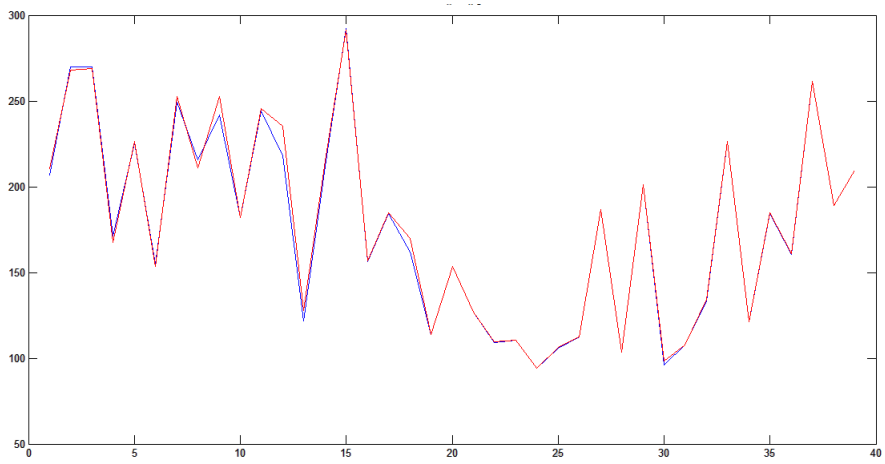


Figure 2.2: Mean pitch before and after GSM codec

The above described approach of modification function analysis has been adopted to all the studied voice parameters.

2.2.3 Modification functions determination

To determine a modelisation of the modification functions, the plot Matlab's tool has been used. In fact, using the function ***plot(a,b,"*")*** gives a dotted description of the modification function as illustrated in figure 2.3. The vector "a" contains the voice parameter's values bofore GSM codec and "b" contains the voice parameters values after the GSM codec. To further understand the process of modification functions determination the process of mean pitch modification function determination is given below.

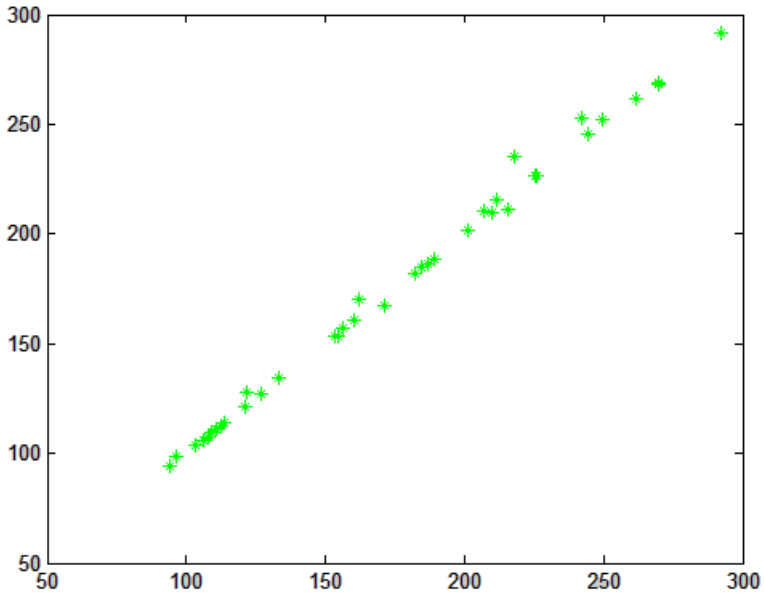


Figure 2.3: Mean pitch before and after GSM codec

Mean pitch modification function determination

Here is a sample of the approach adopted in this work to determine the modification functions. After putting the modification values in a vector. Matlab's function ***plot(a,b,"*")*** gives a dotted overview of the modification. Using the Matlab's tool ***Basic filtering***, then manually choosing the best fit ***interpolation function*** and finally checking the box ***show equation***, a mathematic modelisation of the modification function is given for this specific voice parameter. The following figure illutrates the described process.

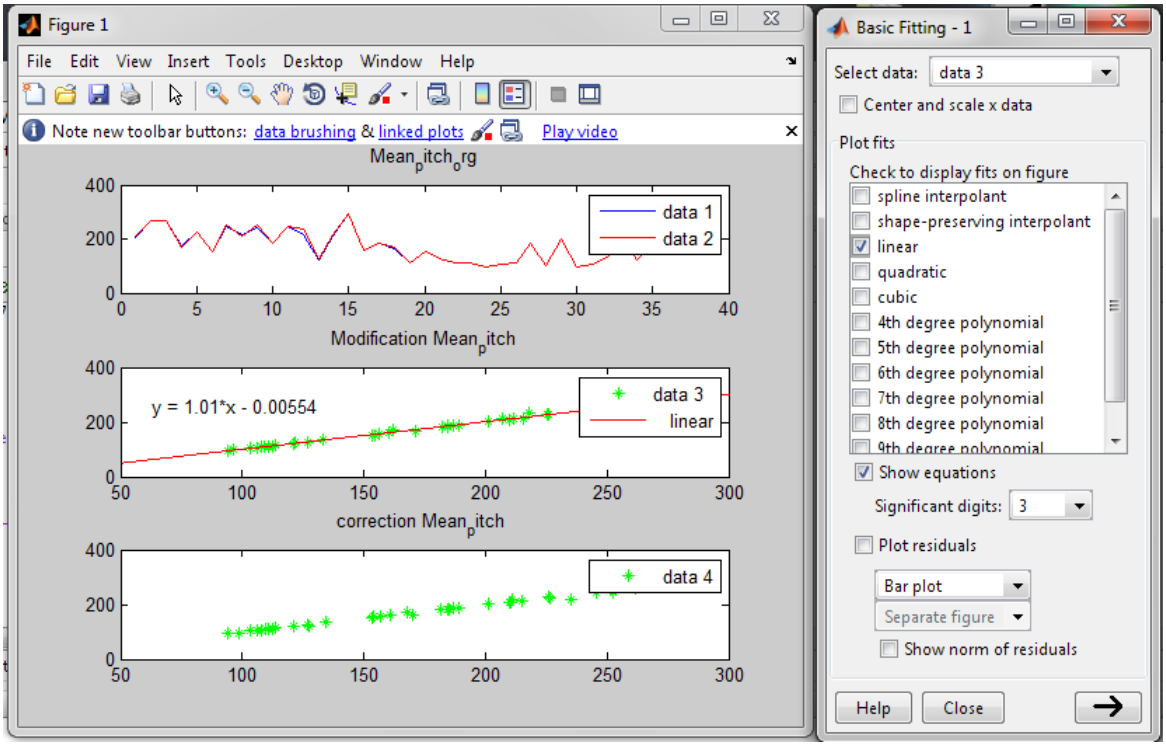


Figure 2.4: Mean Pitch Modification function determinaiton illustration

For a larger list of voice parameter determination illustrations refer to Appendix A. Subsection 2.2.4 gives a list of the determined modification functions.

2.2.4 Modification functions overview

Voice Parameter	modification function
Jitter absolute	$1.05x + 0.867$
Jitter relative	$\emptyset$
Jitter rap	$1.06x + 0.36$
Jitter ppq5	$0.994x + 0.475$
Jitter ppq55	$0.928x + 0.54$

Table 2.2: Jitter Voice parameters modification functions modelisation

Voice Parameter	modification function
Mean Pitch	$1.01x + 0.00554$
Minimum pitch	$0.925x + 8.84$
Maximum pitch	$0.87x + 56.6$
Standard deviation of pitch	$0.904x + 170$

Table 2.3: Pitch Voice parameters modification functions modelisation

Voice Parameter	modification function
Amplitude average	$0.96x + 0.00148$
Shimmer relative	$0.959x + 0.00525$
Shimmer dB	$\emptyset$
Shimmer apq3	$0.944x + 1.74$
Shimmer apq5	$0.955x + 1.88$
shimmer apq55	$x + 1.85$

Table 2.4: Shimmer parameters modification functions modelisation

2.3 Correction of GSM codec modification

After simulating the GSM codec mechanism and finding the modification functions, these functions should be inverted to inverse the influence of the GSM codec. The inversion has been done in two manners. First, the vectors of the modification functions has been inverted, giving the correction functions. This approach was not sufficiently accurate. Thus, another method has been proceeded. Using Matlab's function ***finverse(f(x))***, a more accurate result could be extracted from the modification functions.

Mean Pitch correction function determination

Here is a sample of the approach adopted in this work to determine the correction functions. After collecting the modification functions two method can help determining the correction function. The one is obtained via the use of the modification values in the already gotten vector : Matlab's function ***plot(b,a,"*")*** gives a dotted overview of the correction function. Then via the use of Matlab's tool ***Basic filtering***, choosing manually the inetrpolation function and checking the box ***show equation*** a mathematic modelisation of the correction functions is given. The following figure illutrates the discribed process.

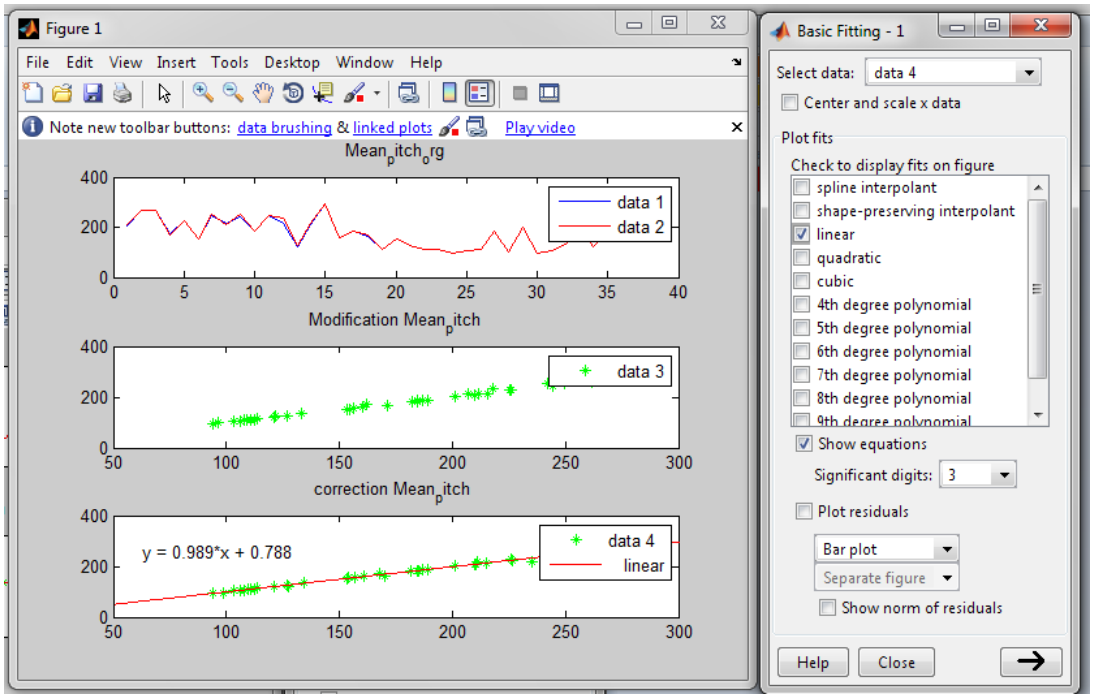


Figure 2.5: Mean Pitch correction function determinaiton illustration

The second method uses the Matlab's function ***finverse(f(x))*** to invert the modification functions that have already been determined. Following is an illustration of this method.

finverse(f(x)) method

GSM encoding script

```
syms x ;
f = 0.99*x+0.79 ;
finverse(f) ;
```

First, the finverse function works only on symbolic expressions, thus the x parameter shown in the above illustration has been declared as symbol with the Matlab's function **syms**. Then applying the finverse function to the already determined modification function gives symbolic expression of the correction function.

Correction functions

Voice Parameter	correction function
Mean Pitch	$11/2000 + x$
Minimum pitch	$-880/93 + 100/93x$
Maximum pitch	$-5660/87 + 100/87x$
Standard deviation of pitch	$-21250/113 + 125/113x$

Table 2.5: Pitch Voice parameters correction functions modelisation

Voice Parameter	correction function
Jitter absolute	$-289/350 + 20/21x$
Jitter relative	$\emptyset$
Jitter rap	$-16/53 + 50/53x$
Jitter ppq5	$-475/994 + 500/497x$
Jitter ppq55	$-17/29 + 125/116x$
Jitter ddp	$-216/239 + 250/239x$

Table 2.6: Jitter Voice parameters correction functions modelisation

Voice Parameter	correction function
Amplitude average	$-37/24000 + 25/24x$
Shimmer relative	$-3/548 + 1000/959x$
Shimmer dB	$\emptyset$
Shimmer apq3	$-435/236 + 125/118x$
Shimmer apq5	$-376/191 + 200/191x$
shimmer apq55	$-37/20 + x$

Table 2.7: Shimmer parameters correction functions modelisation

2.4 Performance Analysis

The study of the performance of the correction functions have shown that the determined correction function gives accurate results for some of the voice parameters such as Mean Pitch and does not give accurate correction function description for other voice parameters such as standard deviation. Figure 3.14 shows an illustration of the Mean pitch correction function.

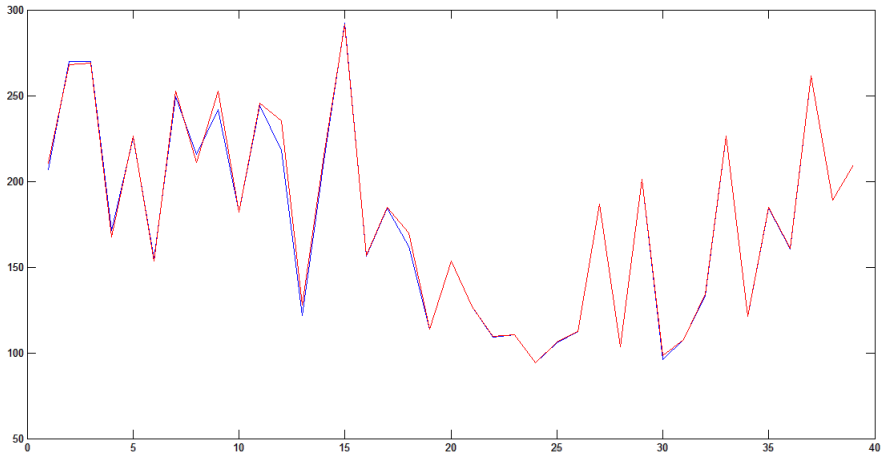


Figure 2.6: Comparing Mean pitch before GSM codec and after correction

- ***in blue*** **:** the Mean pitch evolution before the GSM codec application.
- ***in red*** **:** the Mean pitch evolution after both steps of GSM codec and correction function application

correction function	**value interval**	**mean margin of error**
Mean pitch	[0 .. 400]	∓2.5852
Minimum pitch	[0 .. 400]	∓9.3291]
Maximum pitch	[0 .. 1000]	∓71.0053
Standard deviation of pitch	[0 .. 7000]	∓355.9659
Jitter absolute	[0 .. 50]	∓1.3930
Jitter rap	[0 .. 20]	∓0.4518
Jitter ppq5	[0 .. 20]	∓0.4718
Jitter ppq55	[0 .. 10]	∓0.5906
Jitter ddp	[0 .. 40]	∓0.8343
Shimmer relative	[0 .. 0.1]	∓0.0045
Shimmer apq3	[0 .. 25]	∓0.731
Shimmer apq5	[0 .. 25]	∓0.6624
shimmer apq55	[0 ..40]	∓1.0197

Table 2.8: Jitter Voice parameters correction functions modelisation

As described above, the accuracy of the results is not granted. This may be due to the limitation of the number of samples used within the simulation. It may also be the result of some of experimental errors or the tools used within this study. For further enhancement of these results, the first step to be taken is to work on a larger database. Also, while choosing the interpolation functions to describe the modification function, linear interpolation may not be the fit description. Thus, the interpolation function for determining modification function need to be choosed based on the calculus of the variance of that specific interpolation.

2.5 Conclusion

In this chapter, the simulation steps and results has been described all together with the process of modification and correction function determination. Now, that the correction functions expressions are revealed we can carry on to the embedded application implementation. The study of the performance, of the correction functions showed accurate results for some voice parameters and less accurate results for others, thus further corrections need to be applied in order to get more accurate results.

Chapter 3

Embedded Solution Implementation

After studying the theoritecal background of the problem in chapter 1 and presenting the adopted resolution approach and the obtained results in chapter 2, chapter 3 will present the implementation of the determined solution as an embedded application. First, the design of the application and the methodology of implementation will be presented. Second an overview of the development environment, **Visual DSP ++**, and the targeted hardaware, **Blackfin BF533 processor**, will be given. Then a presentation of the taken implementation steps is presented. Last, but not least, is a study of the performance of the implemented embedded solution and the taken optimization steps.

3.1 Design & Methodology

As for any embedded application, the interaction with the environment of implementation of this solution shoud be taken on consideration. For the present studied application, it will be implemented on an embedded audio mircoship based on a ADSP-BF533 processor which will be presented in the next part of this chapter. For instance, the application will take as input an audio file that will be loaded to the embedded microship external memory. The targeted hardware's memory is of 148 kb size. Thus, it cannot support the amount of data to be treated later in the several algorithms of the application. So, to resolve this problem and for the purpose of simulating the real world implementation environment, an external memory has been used.

In the external memory, an audio file, in **.DAT** file format, has been loaded to the Visual DSP ++ Development environment in a virtual external memory described by an **.LDF**[1] file.

After getting the data to the memory, the application starts by determining the instants of pulses via the implemented **Pulse Detection Algorithm**. These instants of pulses are the major assets for calculating the human voice parameters from the loaded audio file. when voice parameters are ready, the implemented correction functions will be applied to them to correct any distorsion caused by the GSM codec. As an output of the application, the corrected voice parameters will be presented to the therapist

[1]Linker Description file

and may then be used for the pupose of voice diagnosis. In Figure 3.1 a block diagram discription of the several parts of the implementation is given.

Figure 3.1: Block Diagram of the implementation conception

- ***voice*** : the input pathological sustained vowel 'aaa'
- ***PDA*** : Pulse Detection Algorithm
- ***PI*** : Pulse Instants
- ***VDP*** : Voice Parameters Determination
- ***EVP*** : Errenous Voice Parameters
- ***VPC*** : Voice Parameter Correction
- ***CVP*** : Corrected Voice Parameters

3.2 Development environment

Before turning the studied solution to a real world embedded application, it should first get through several steps of developing and testing. To do so, the Visual DSP ++ development envrionment has been used which is presented in subsection 3.2.1. The targeted hardware for the embedded application is a Blackfin BF533 processor and it will also be presented in subsection 3.2.2.

3.2.1 Visual DSP ++

The following illustration presents the Visual DSP++ development environment in action.

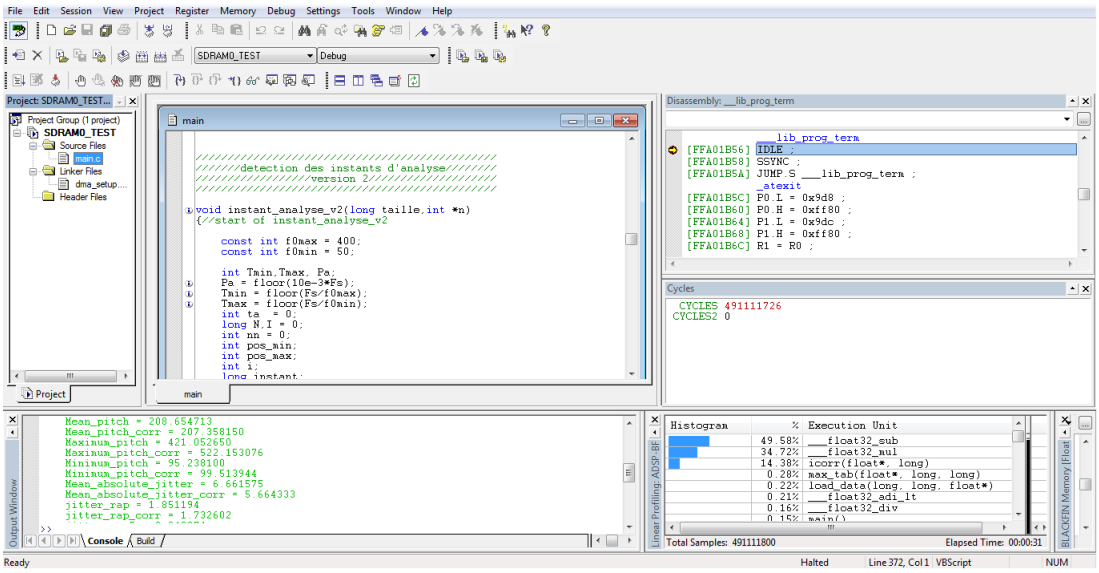


Figure 3.2: Visual DSP ++ development environment

Visual DSP ++ is an integrated software development environment and debugging environment, **IDDE** for ADI processors that consists of an IDE, debugger, C/C++ compiler, assembler, linker, and simulator[2].

3.2.2 Blackfin BF533 Core architecture

The Blackfin processor core contains two 16-bit multipliers, two 40-bit accumulators, two 40-bit arithmetic logic units (ALUs), four 8-bit video ALUs, and a 40-bit shifter, shown in following figure. The process 8-, 16- or 32-bit data from the register file[10]. see the analog devices offical website and sheet

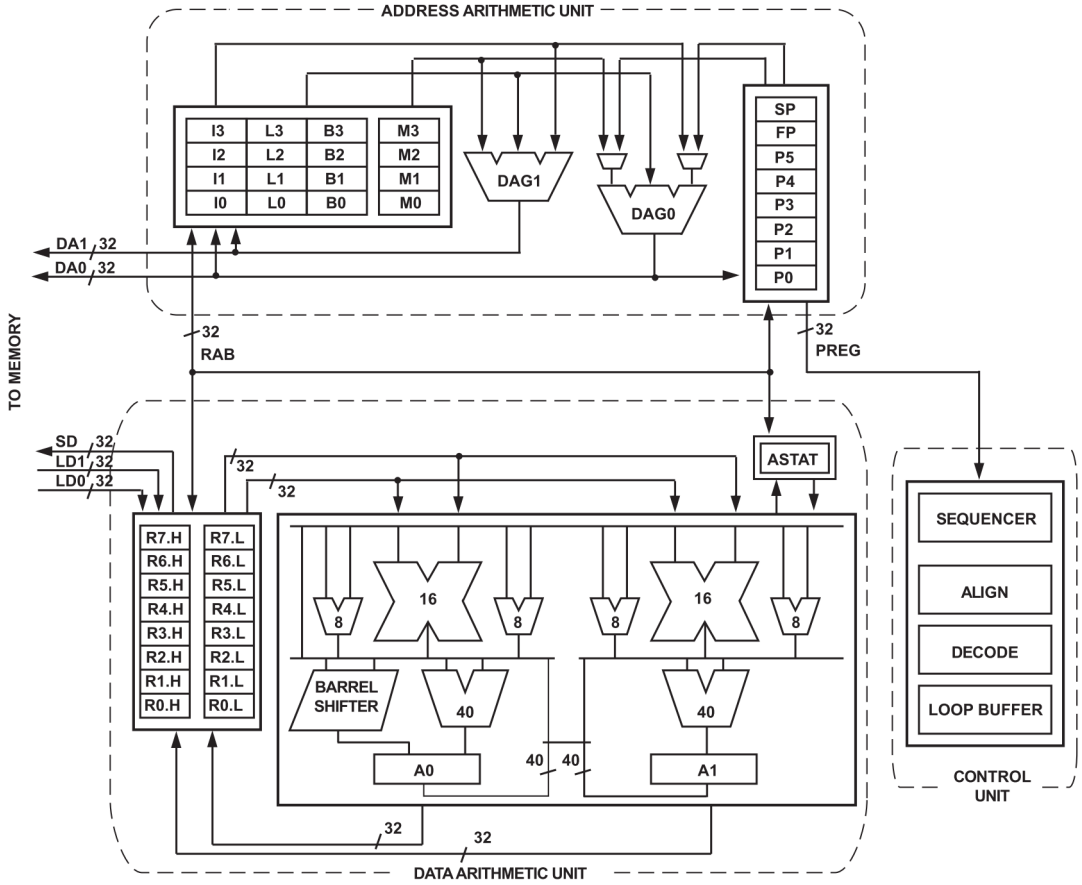


Figure 3.3: Blackfin Processor Core Architecture[11]

The computer register file contains eight 32-bit registers. When performing compute operations on 16-bit operand data, the register file operates as 16 independent 16-bit registers. All operands for compute operations come from the multiported register file and instruction constant fields.

Each MAC can perform a 16-by 16-bit multiply per cylce, with accumulation to a 40-bit result. Signed and unsigned formats, rounding, and saturation are supported.

The ALUs perform a traditional set of arithemtic and logical operations on 16-bit or 32-bit data. Many special instructions are included to accelerate various signal

[2]refer to analog devices official website `http://www.analog.com/en/index.html`

processing tasks. These include bit operations such as field extract and population count, modulo 2^{32} multiply, divide primitives, saturation and rounding, and sign/exponent detection. The set of video instructions include byte alignment and packing operations, 16-bit and 8-bit adds with clipping, 8-bit average operations, and 8-bit subtract/absolute value/accumulate (SAA) operations. Also provided are the compare/select and vector search instructions. For some instructions, two 16-bit ALU operations can be performed simultaneously on register pairs (a 16-bit high half and 16-bit low half of a comute register). By also using the second ALU, quad 16-bit operations are possible.

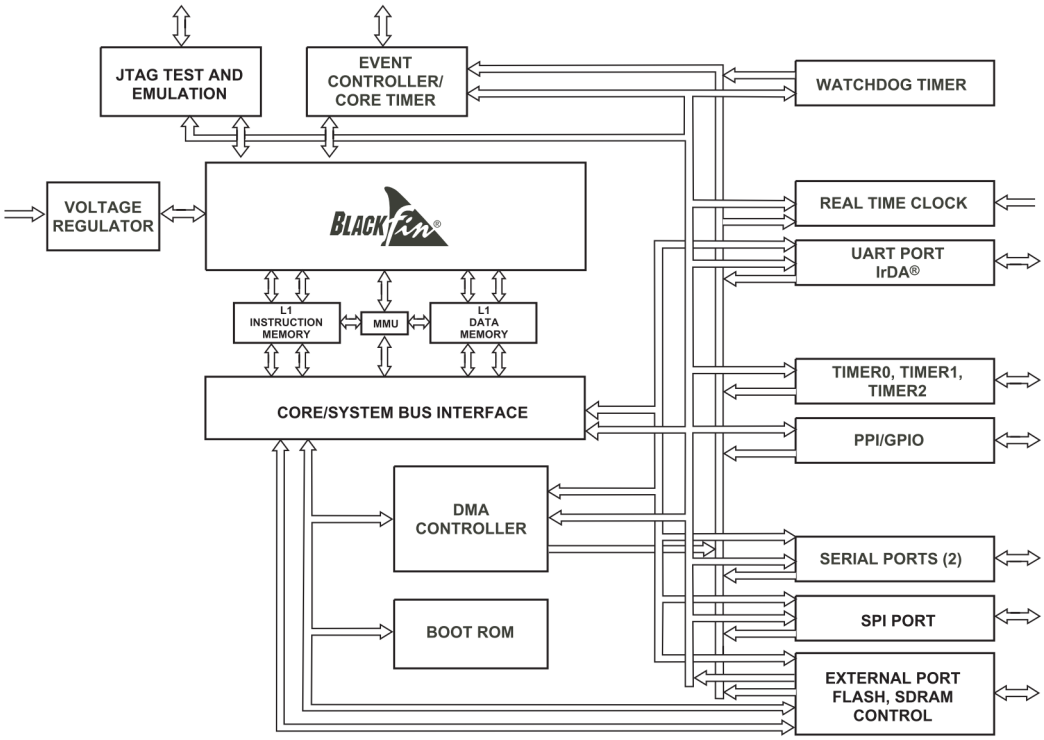


Figure 3.4: Processor Block Diagram[12]

Figure 3.4 shows how external peripherals such as Watchdog timers, event controllers or real time clocks are connected to the core Blackfin processor via several high bandwidth buses. This figure illustrates a possible real world implementation environment architecture for the studied solution.

3.3 Implementation

The purpose of this section is to give a board outline of the implementation process. Subsection 3.3.1 on the following page describes the use of external SDRAM[3] for convenience purposes. Subsection 3.3.2 describes the Pulse Detection Algorithm. Subsection 3.3.3 presents the Voice Parameters Calculus based on the pulse instants already determined in the previous step. Subsection 3.3.4 on page 38 explains the correction function application to the altered voice parameters.

[3]Synchronous Dynamic Random Access Memory

3.3.1 SDRAM usage

For the purpose of simulating a real world environment, an audio **.DAT** file had to be loaded to the Visual DSP++ development environement. The limitation of size, which is 148 kb, of the memory-on-chip on the ADSP-BF533 processor has been a constraint for the data to be loaded. In fact, the amount of data to be treated in the application varies between around 190kb to more than 800kb. Thus, an external memory has been used. In order to the external memory to be considered in the final code compilation an LDF file has been used. LDF file, Linder Description File, gives a description of the external memory associated to this application. this has been done by including the file ***dma_setup.ldf*** to the project. The following code extraction presents how the audio file has been loaded.

load data computation

Load data computation and SDRAM usage

segment("sdram0") float s[10000]={
#include "C:$\backslash Users\backslash user\backslash Desktop\backslash test\_sdram0\_p2a\backslash audio\backslash F0503SMOD.DAT$"
};

Now a table named "s" of type float contains 10000 audio voice samples with 8000 sampling rate ready to be used in the next algorithms.

3.3.2 Pulse Detection Algorithm

The first step in the computation of the embedded application is the determination of the **Pulses Instants**. For instance, pulses instants describe the rank of pitch pulse within the table **s**. A pulse instant is the position of the maximum of voice signal in one pitch period. As for their determination, the function declared as ***void instant_analyse_v2(long taille,int *n)*** has been computed. This function takes as input the length of the voice data extraction "s". As for the data itself in the table "s", it has been declared as global so it does not need to be load as input to the function. The determination of pulses instants is made in two phases: first an approximation of the pulse instant is made via the calculus of the autocorrelation of a specifically rendered voice extraction. The maximum of autocorrelation function gives an approximation of the voice pulse instant within the extraction in question. Further corrections to the pulse instant position in this extraction are made in phase two : the algorithm takes an extraction of the voice signal around the approximated pulse instant and recompute the maximum of the signal, thus, giving a more accurate pulse instant.

The autocorrelation has been computed as a seperate function called within the pulse detection algorithm.

$$autocorrelation(k) = \sum_{i=k}^{N-k-1} x(i)x(k-i) \tag{3.1}$$

Figure 3.10 gives a block diagram description of the processes that are being computed within the pulse detection algorithm.

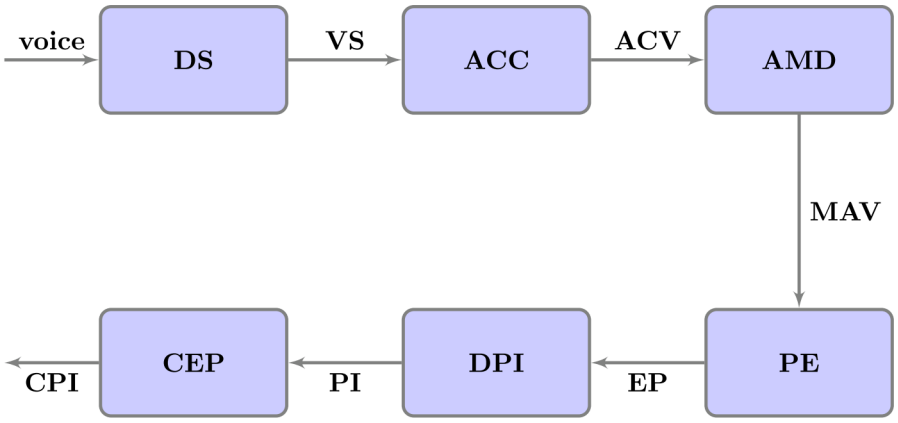


Figure 3.5: Block diagram of the Pulse Detection Algorithm

- ***voice*** : input voice signal
- ***DS*** : Data Segementation : extract from the s a small piece of data
- ***SV*** : segmented voice
- ***ACC*** : Auto-Correlation Calculus
- ***ACV*** : Auto-Correlation Value
- ***AMD*** : Auto-Correlation Maximum Detection
- ***MAV*** : Maximum Auto-Correlation Value
- ***PE*** : Period Estimation
- ***EP*** : Estimated Period
- ***DPI*** : Detection of Pulse Instant
- ***PI*** : Pulse Instant
- ***CEP*** : Correction of Estimated Pulse instant
- ***CPI*** : Corrected Pulse Instant

Figure 3.6 illustrates the results of the Pulse Detection Algorithm. The instant of pulses are marked with an asterisk. These pulses instants are later used to determine the periods between two pulses for jitter parameters calculus. They are also used to determine the values of amplitude maximums for shimmer parameters calculus.

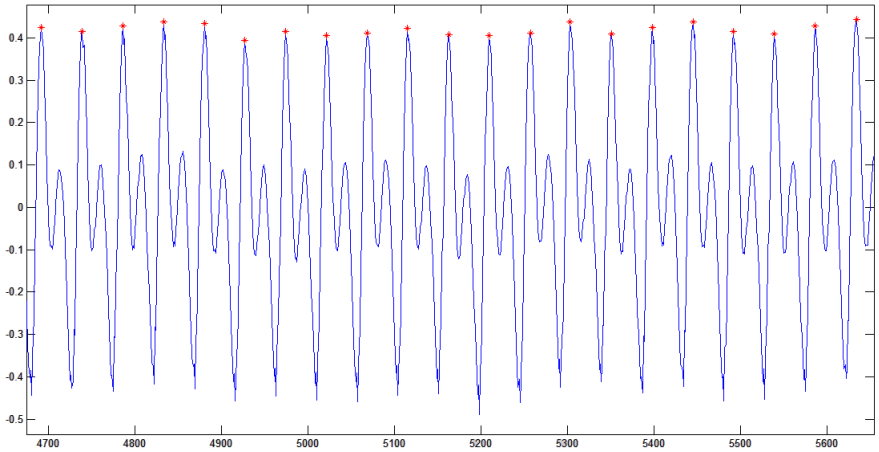


Figure 3.6: Pulse Detection Algorithm results

The flow chart in figure 3.7 gives a description of the inner functionnality of the

implemented Pulse Detection Algorithm.

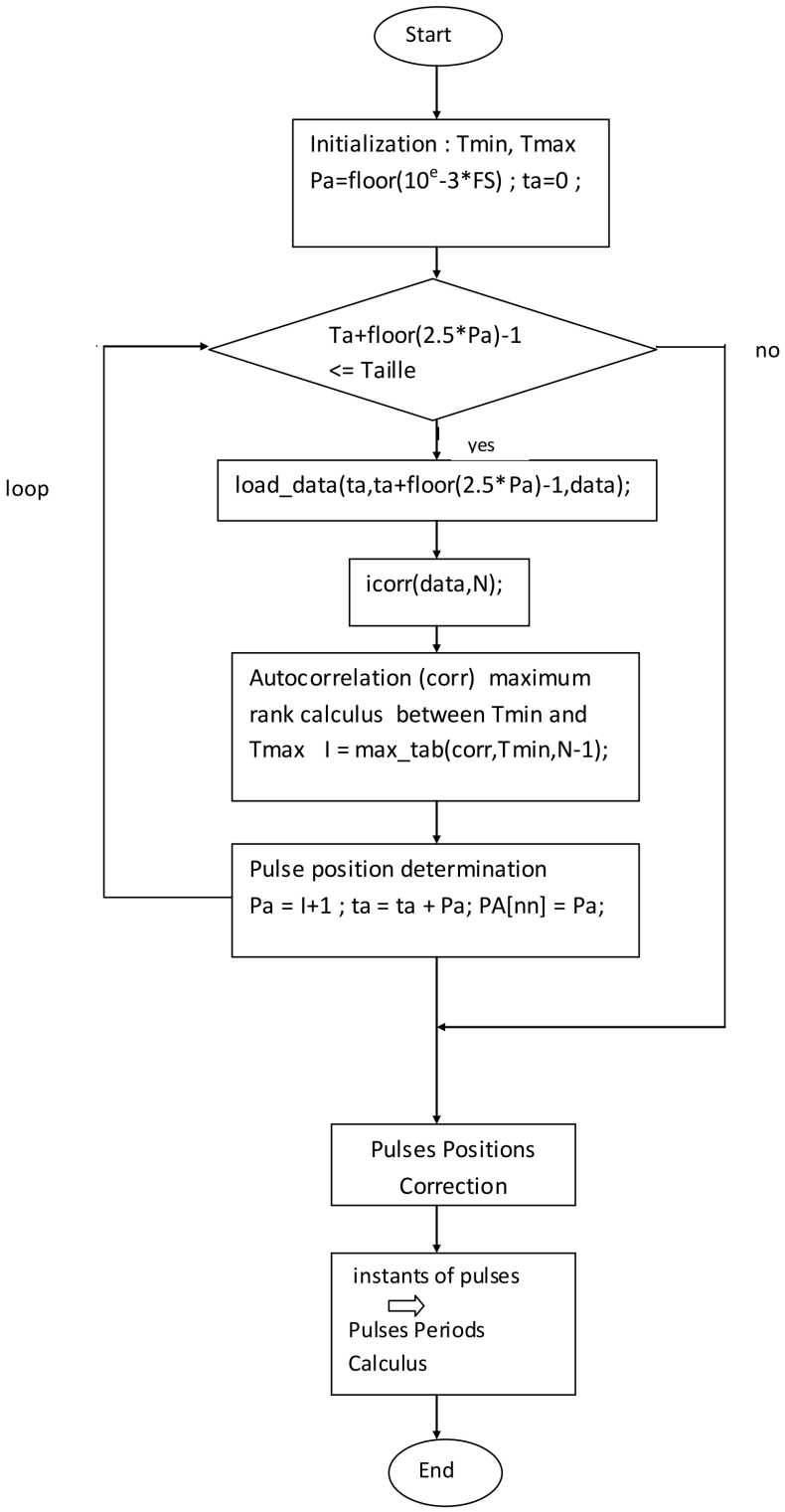


Figure 3.7: Flow chart of the pulse detection algorithm

3.3.3 Voice Parameters Calculus

The pulses instants are the major assets to be used to calculate voice parameters. In fact, using the instants of pulses, the maximum signal amplitude values can be determined thus permitting the determination of shimmer parameters. Also, the jitter

values can be calculated after determining the pitch values. The pitch values are described as follows :

$$pitch(i) = \frac{Fs}{pulse(i+1) - pulse(i)} \tag{3.2}$$

where Fs is the sample rate of the treated signal, pulse(i) is the i^{th} pulse instant and pitch is i^{th} pitch value.

Based on the pitch and maximum signal amplitude values, and using the mathematic description of voice parameters presented in section 2.1 on page 19 several functions has been implemented to calculate voice parameters. The following illustration presents the implemented function to calculate the Mean Absolute Shimmer voice parameter. The same logic has been adopted to compute all the other voice parameters. As mentioned in Subsection 2.1.2 on page 20 the shimmer(relative) voice parameter is decribed as follows

$$Shimmer(relative) = \frac{\frac{1}{N-1}\sum_{i=1}^{n}|A_i - A_{i+1}|}{\frac{1}{N}\sum_{i=1}^{N}A_i} \tag{3.3}$$

To calculate the value of shimmer relative the consecutive signal amplitude maximum values subtracted then accumulated to a variable called som. Dividing the som to the total number of pulse instants gives mean absolute shimmer value. Then dividing the mean absolute shimmer to average of signal amplitude maximum values gives the shimmer(relative). Figure 3.9 a block diagram of the shimmer (relative) function.

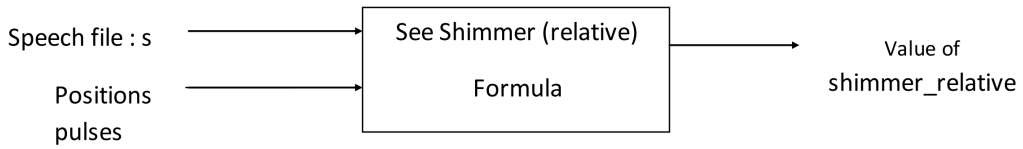


Figure 3.8: shimmer(relative) block diagram

The same implementation approach has been adopted to all of the others voice parameters.

3.3.4 Correction functions implementation

After being determined in Section 2.3, the correction function could be implemented with the same logic as for voice parameters modification functions described above. For each voice parameter correction function a new function has been declared. The function gets as input the altered voice parameter value and apply the implemented correction function for that specific voice parameter then returns as ouput the value of the corrected voice parameter. For example, to implement the Mean Pitch voice parameter a block diagram description is giving below, in figure 3.9.

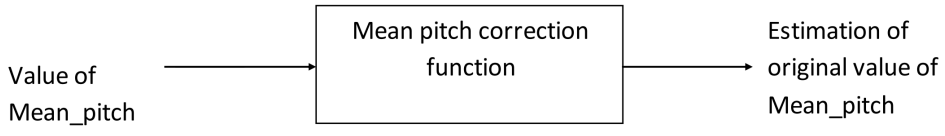


Figure 3.9: mean pitch correction block diagram

The same implementation approach has been adopted to all of the others voice parameters.

3.4 Performance Analysis

In order to be implemented as a real time application, the studied solution must be optimized to overcome any possible bottleneck in the embedded application. The Visual DSP ++ development environment has a set of ready to use tools that helps to study the performance of the implemented solution. In this work several methods has been used to analyse the perfomance of the application.

CPU usage frequency

Using the **Linear Profiling** tool built into the Visual DSP ++ development environment, a description of the CPU usage frequency can be taken. As illustrated in figure 3.10 the functions that take the most of the CPU frequency usage are consecutively **_float32_sub** with 49.09%,**_float32_mul** with 34.44% and the **icorr(float *, long) with 14.26%** function which determines the autocorrelation for the input voice signal. The extensive use of the standard DSP functions **_float32_sub** and **_float32_mul** is due to the autocorrelation function. In fact, the autocorrelation function is a set of multiplications and accumulations of 32 bit floats. Thus considering these two functions within the CPU usage frequency of the autocorrelation function, the total icorr use of the CPU raises to more then 90% . Obviously, the icorr function is a bottleneck for the embedded application and it needs further optimizations.

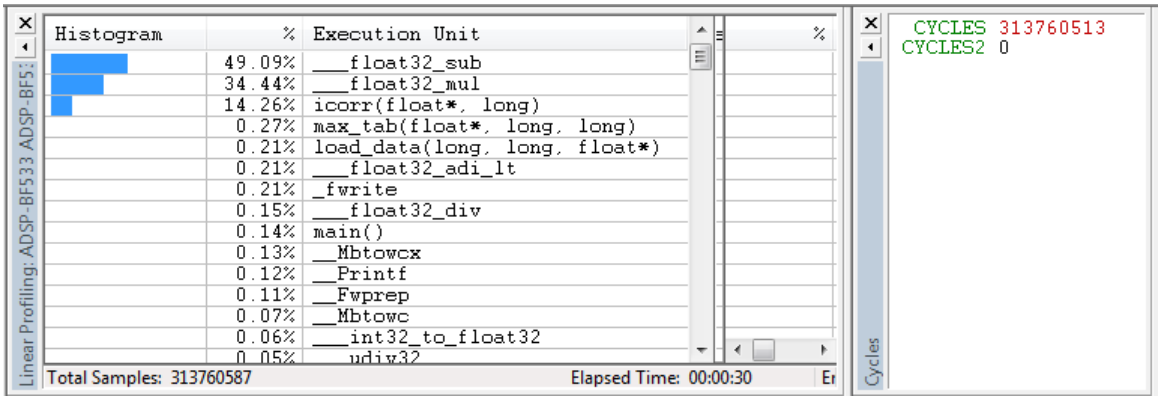


Figure 3.10: Performance Analysis in Visual DSP ++ : C version

Number of computational cycles

Another tool can be used to study the performance of the application, it is the CYCLES register. The cycle count registers of the Blackfin architecture are called CYCLES and CYCLES2 registers. These registers are 32-bit registers. The CYCLES register is incremented at every processor cycle; when it wraps back to zero the CYCLES2 register is incremented. The total number of cycles can be obtained within the linear profiling window, it is resumed in the Total Samples label, that can be found at the bottom of the linear profiling window.

For the original embedded application coded with C programming language and compiled with the viual DSP compiler the number of CYCLES is 313760513. Also a detailed description of the number of cycles used in each function within the program can be obtained by right clicking and choosing **View Sampe Count**. The table 3.1 resumes the repartition of CYCLES to the several parts of the algorithm.

C function	**number of Cycles**
float32_sub	153943943
float32_mul	107982004
icorr(float *, long)	44757643
max_tab(float *, long, long)	858288
load_data(long, long, float*)	669062
float32_adi_lt	649464
float32_div	472479
main()	424953

Table 3.1: CYCLES repartition

Other parameters

Other parameters such as the elapsed time or number of samples can be used to further observe the performance of the implemented system. For instance elapsed time gives in seconds the time that the targeted harware will take to run the algorithm. Thus, it is a crutial parameter to determine wether the studied application can be implemented in a real time system or not. For the studied application the elapsed time for the C version is up to 30 seconds.

3.5 Performance Optimization

From the previous performance study, a bottleneck has been found , it is obviously the autocorrelation function. Thus further optimization has been applied to this function to reduce its time of computation and its CPU frequency usage. The following parts describe the steps taken to optimize the performance of the embedded application.

3.5.1 DSP assembly implementation

The autocorrelation function has been recomputed using DSP assembly. The flow chart in figure 3.11 illustrates the DSP assembly implementation of autocorrelation function.

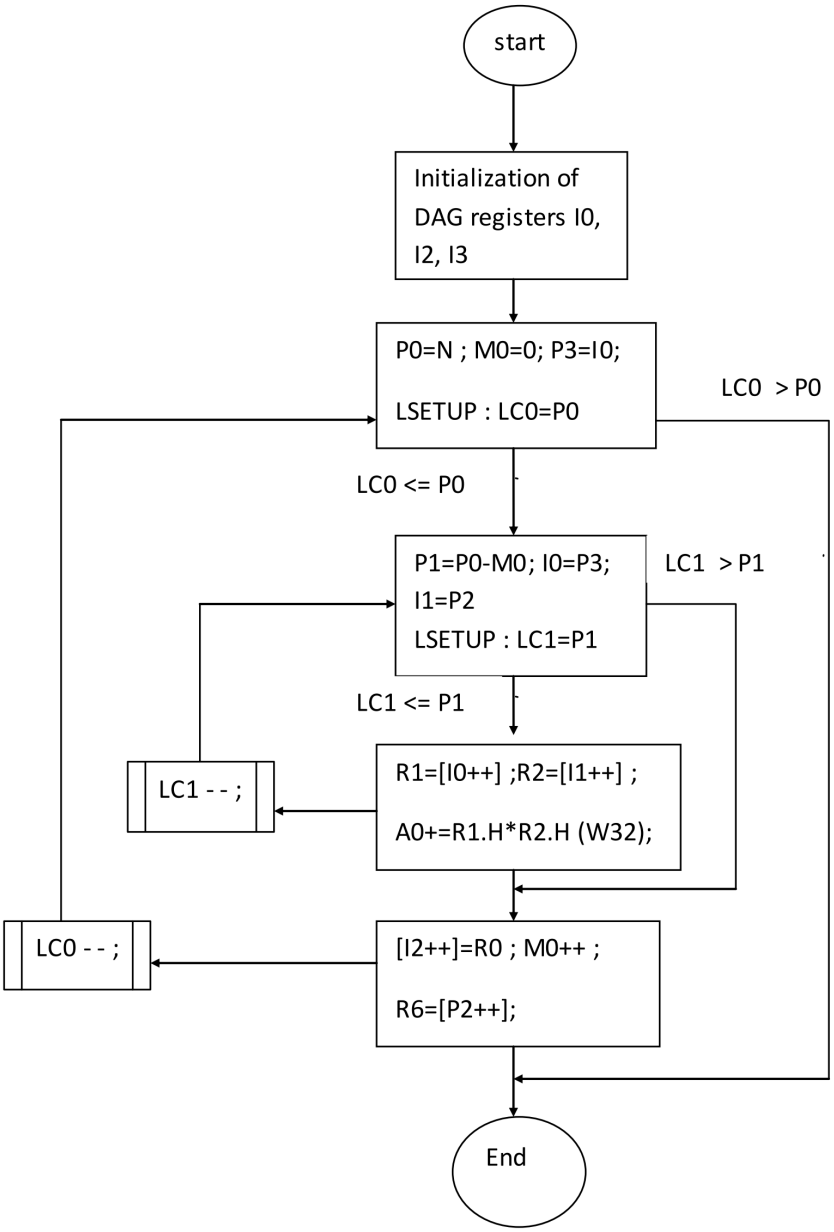


Figure 3.11: DSP assembly autocorreation flow chart

For instance, as described in section 3.2.2 the DSP code processor does not support 32-bit float multiplication, thus the float values will be converted to fractional values that can be implemented into DSP assembly. The rendering of data from 32-bit float to fractional and back to 32-bit float has been done using standard DSP C libraries.

3.5.2 performance optimization analysis

After computing the autocorrelation function in the ADSP-BF533 assembly, a significant execution time reduction has been noticed. In fact, the elapsed time parameter for the new version of the embedded application shows that the execution time is less then 3 seconds. Considering the execution time of the original C program of 30 seconds, a reduction to the tenth of execution time has been acheived.

As for the number of cycles, the linear profiling tool shows that the total number of cycles of updated version of the application is 12906164 , thus a reduction to more then 24 times the original cycles count value has been made.

$$\frac{313760513}{12906164} = 24.31$$

The figure 3.12 gives a description of the CPU time usage and the total samples repartition after using the DSP assembly.

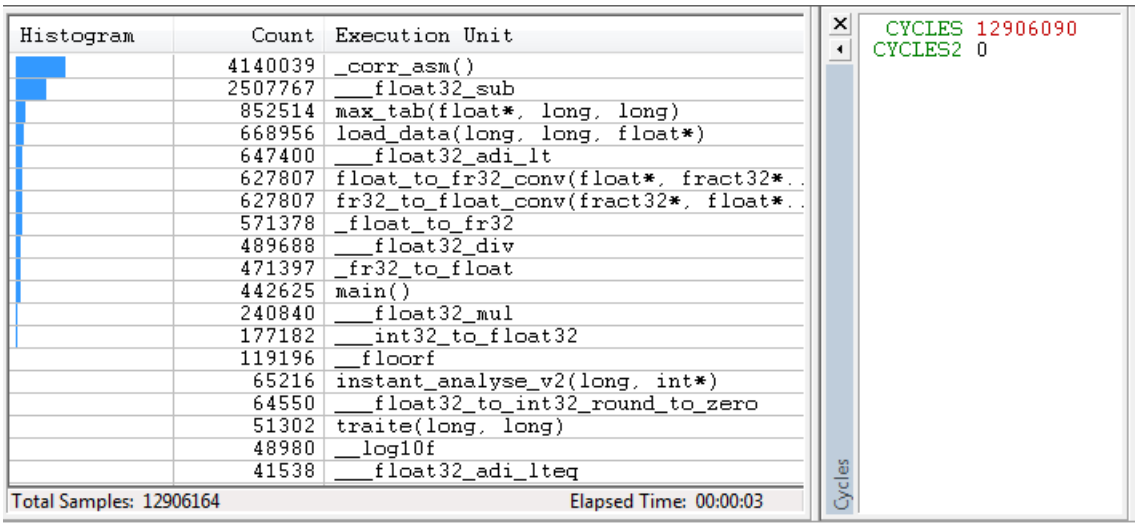


Figure 3.12: Performance Analysis in Visual DSP ++ : assembly version

As illustrated in the below figure the number of cycles for the application to run has been significantly reduced thus optimizing the performance of the embedded solution. The table 3.2 describes the new cycles count repartition:

C function	number of Cycles
_corr_asm( )	4140039
float32_sub	2507767
float32_mul	240840
max_tab(float *, long, long)	852514
load_data(long, long, float*)	668956
float32_adi_lt	647400
float32_div	489688
main()	442625

Table 3.2: CYCLES repartition with assmebly usage

3.6 Conclusion

In this chapter, the conception of implementation has been presented where we have seen the several block implemented so that the embedded application fullfill its tasks. Then, a board outline of the development toolset and targeted hardware have been described. After presenting the architecure of the embedded application implementation within the ADSP-BF533 processor an overview of the implementation steps and of the computational details have been given. Finally, the analysis of application performance and the taken optimization steps have been explained.

Conclusion

In this work, the use of voice parameters for pathology detection has been opened to a larger scale. The measurement of the voice features over a GSM network will allow for more mobility and even for the implementation of this solution to a more complex system, like bringing the voice pathology detection mechanisms to a certain level of mobility by implementing them over the GSM network via the work described in this report. For instance, the accuracy of the results is questionnable because the implemented algorithms for jitter and shimmer estimation varies from one acoustic analysis software to another, and for the Praat software for acoustic analysis, the pulse detection algorithm is not well fitted to pathological voice analysis, this issue has been studied by Mirieria Farrus, Javier Hernando, Pascual Ejarque from the unversity of PolitÃĺcnica de Catalunya, Barcelona, Spain in their paper titled ***Jitter and Shimmer Measurments for Speaker Recognition*** and by Darcio G. Silva, Luis C. Oliveira, and Mario Andrea from the University of Lisbon, Portugal in these research article titled ***Jitter Estimation Algorithms for Detection of Pathological Voices***. Therefore, a more robust solution would implement several estimation algorithms for voice parameters and provide the therapist with the flexibility of choosing the best fit parameter values. This approach may be a guide-line to more revisions to the adopted resolution methodology and to the implmentation of a more complete solution.

This study does not pretend getting to the best solution to the problem nor it does the assumption of making the best implemenation of such a solution, but it hass been one of the major concerns of the authors to make it clear and steady to allow for further revisions and updates to the current version and to be a guide-line to a more robust and general study to this problem.

Appendix A

Illustration of voice paramters modification functions determination

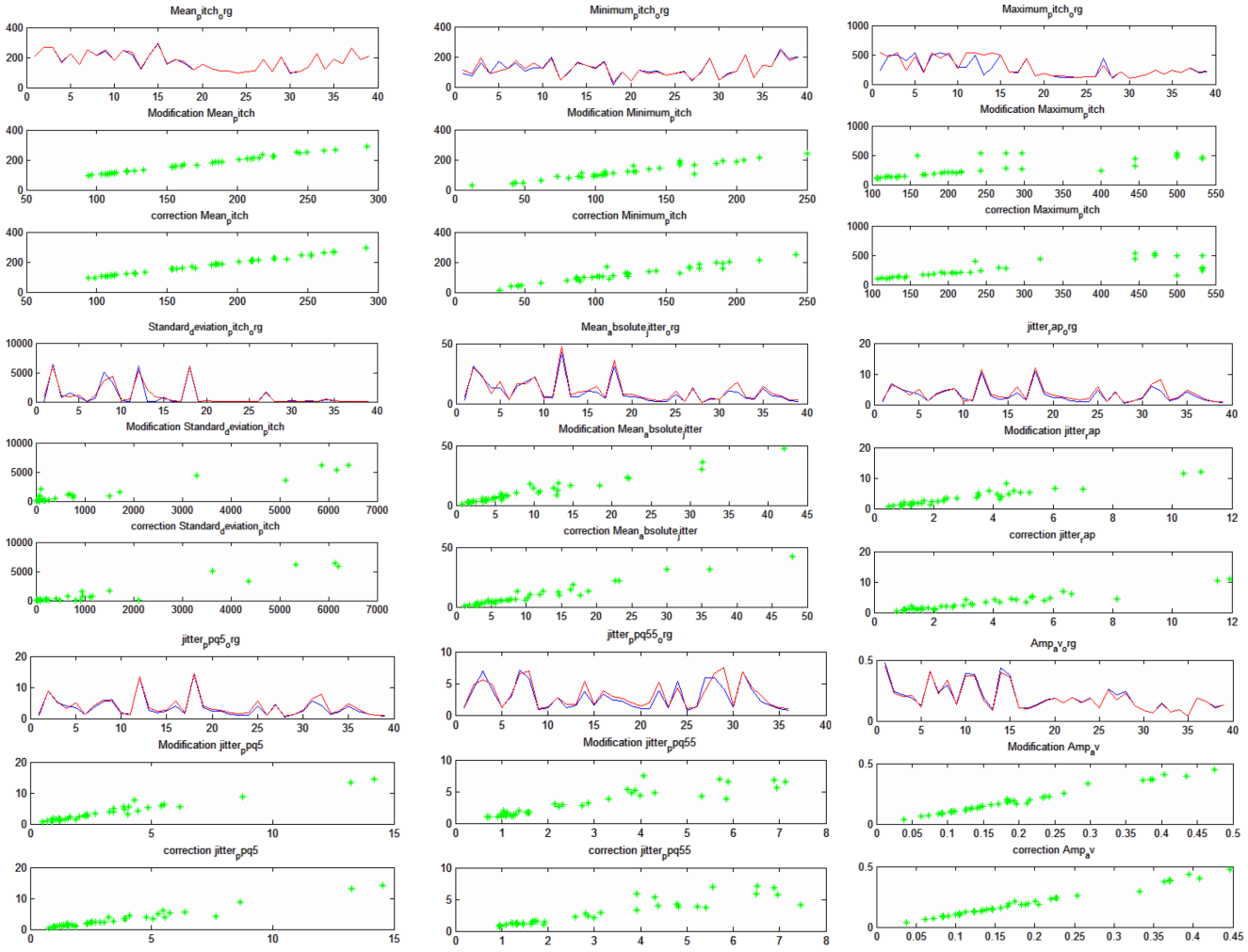


Figure 3.13: Modification function determination : Appendix A

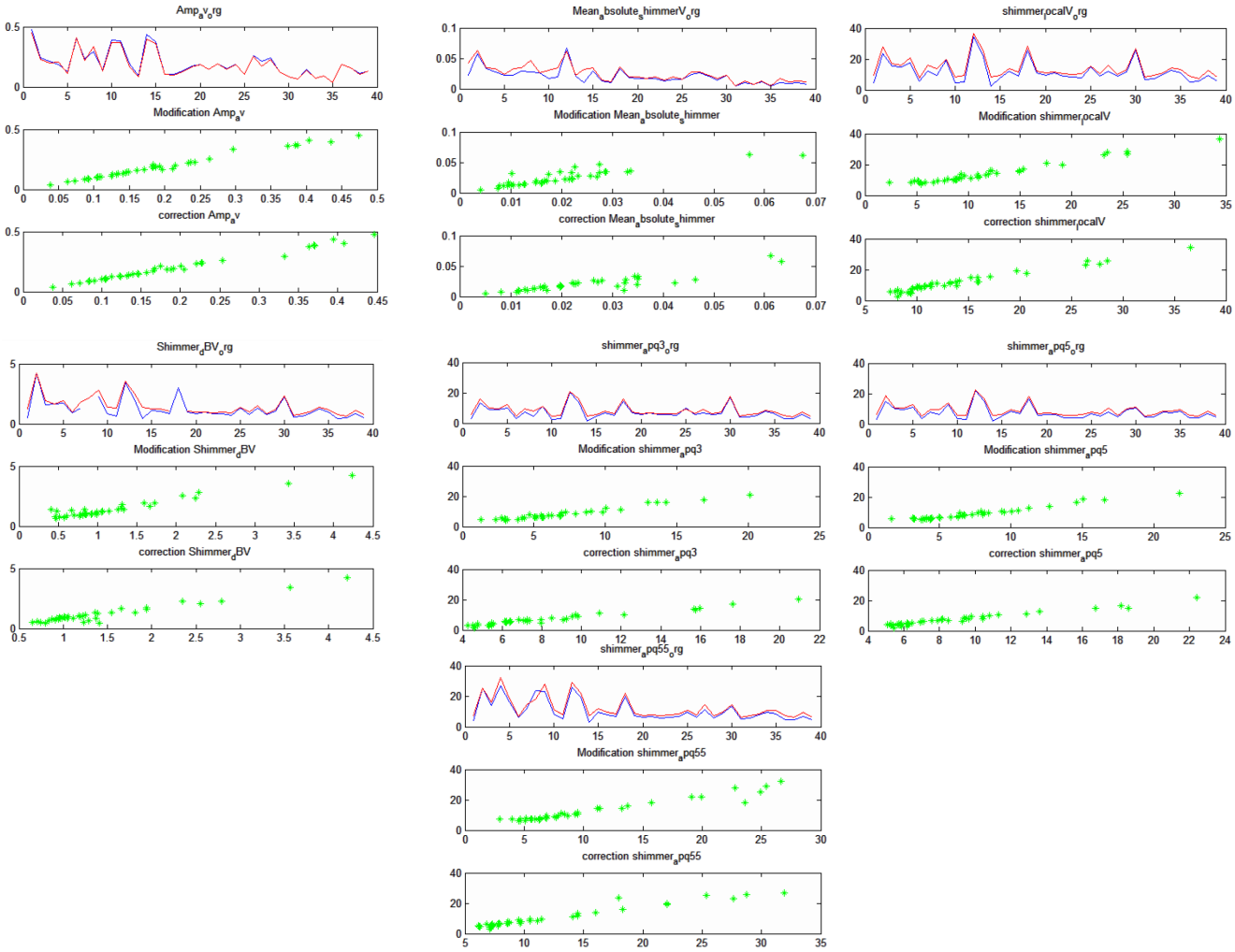


Figure 3.14: Modification function determination : Appendix A (others)

Bibliography

[1] Kristo Lehtonen, *Digital Signal Processing and Filtering : GSM Codec*

[2] Sagittalmouth , *Source : http: // upload. wikimedia. org/ wikipedia/ commons/ 2/ 20/ Sagittalmouth. png*

[3] D. Wong, M. R. Ito, N. B. Cox, and I. R. Titze, *Observation of perturbations in a lumped-element model of the vocal folds with application to some pathological cases The Journal of the Acoustical Society of America, vol. 89, no. 1, pp. 383âĂŞ394, 1991.*

[4] adapted from, *StÃŀphane de CORBIERE, Elisabeth FRESNEL, LA VOIX : LA CORDE VOCAL ET SA PATHOLOGI*

[5] Gray956 , *source : http://upload.wikimedia.org/wikipedia/commons/5/50/Gray956.png*

[6] Vocal Pathology Image Library *source http://www.gbmc.org/home_voicecenter.cfm?id=1563*

[7] Ubuntu Linux OS : *downloadable for free from http://www.ubuntu.com/*

[8] GSM 06.10 lossy speech compression, *downloadable from http://www.quut.com/gsm/*

[9] the praat, *software for actoustic analysis website http://www.praat.org* http://upload.wikimedia.org/wikipedia/commons/2/20/Sagittalmouth.png

[10] Processor Core Architecture, *Blackfin Processor Programming Reference , Analog Devices, Inc.*

[11] Blackfin Processor Core architecture, *BlackfinÂő Processor Programming Reference(Includes All ADSP-BF5xx Blackfin Processors)*

[12] Processor Block Diagram, *ADSP-BF533 BlackfinÂő Processor Hardware Reference (Includes ADSP-BF531 and ADSP-BF532 Blackfin Processors)*