Universidad de Alcalá

ENRIQUE ALEXANDRE CORTIZO
Dpto. de Teoría de la Señal y Comunicaciones
Campus Universitario s/n
28805 Alcalá de Henares (Madrid)
Telf: +34 91 885 67 27
Fax: +34 91 885 66 99
enrique.alexandre@gmail.com

D. ENRIQUE ALEXANDRE CORTIZO, Profesor Titular de Universidad del Área de Conocimiento de Teoría de la Señal y Comunicaciones de la Universidad de Alcalá,

CERTIFICA

Que la tesis **"Signal Processing Algorithms for Digital Hearing Aids"**, presentada por Dña. Lorena Álvarez Pérez, realizada en el Departamento de Teoría de la Señal y Comunicaciones bajo mi dirección, reúne méritos suficientes para optar al grado de Doctor, por lo que puede procederse a su depósito y defensa.

Alcalá de Henares, 19 de diciembre de 2011.

Fdo. Dr. D. Enrique Alexandre Cortizo

Universidad de Alcalá

Dpto. de Teoría de la Señal y Comunicaciones
Campus Universitario s/n
28805 Alcalá de Henares (Madrid)
Telf: +34 91 885 88 99
Fax: +34 91 885 66 99

Dña. Lorena Álvarez Pérez ha realizado en el Departamento de Teoría de la Señal y Comunicaciones y bajo la dirección del Doctor D. Enrique Alexandre Cortizo, la tesis doctoral titulada **"Signal Processing Algorithms for Digital Hearing Aids"**, cumpliéndose todos los requisitos para la tramitación que conduce a su posterior lectura.

Alcalá de Henares, 19 de diciembre de 2011.

EL DIRECTOR DEL DEPARTAMENTO

Fdo. Dr. D. Saturnino Maldonado Bascón

Universidad de Alcalá

Escuela Politécnica Superior

Departamento de Teoría de la Señal y Comunicaciones

Ph. D. Thesis

Signal Processing Algorithms for Digital Hearing Aids

Author:
Lorena Álvarez Pérez

Supervisor:
Enrique Alexandre Cortizo, Ph. D.

December, 2011

Abstract

Hearing loss is a problem commonly suffered by middle-aged and elderly people, as a consequence of the result of the natural process of aging and deterioration of the human auditory system. Hearing loss severely affects the speech communication and disqualify most hearing-impaired people from holding a normal life. Although the vast majority of hearing loss cases (90 %) could be corrected by using hearing aids, however, only a scarce 20 % of hearing-impaired people who could be benefited from hearing aids purchase one, and even more, about 25 % of people who purchase a hearing aid do not wear it in their everyday life, despite the substantial financial outlay involved in purchasing and maintaining a hearing aid.

This irregular use of hearing aids arises from the existence of a critical, scientific and technological problem that, to date, has not been solved effectively and comfortably: the automatic adaptation of the hearing aid to the changing acoustic environment that surrounds its user. It has been shown that most hearing aid users generally prefer to have a variety of amplification programs adapted to different acoustic conditions. There are two approaches aiming to comply with it. On the one hand, the "manual" approach, in which the user has to identify the acoustic situation and choose the adequate amplification program by using a switch on the hearing instrument or some kind of remote control, has been found to be very uncomfortable and frequently exceeds the abilities of many hearing aid users, especially the abilities of the eldest. The second approach, probably more comfortable for the user, requires to include an automatic program selection within the hearing aid. This latter approach is deemed very useful by most hearing aid users, even if its performance is not completely perfect, because it would help them in improving the speech intelligibility, holding a normal life, and increasing the comfort level.

Although the necessity of the aforementioned sound classification system seems to be clear, its implementation is, on the contrary, a very difficult matter. Despite the impressive advances in microelectronics, the development of an automatic sound classification system in a digital hearing aid is a challenging goal because of the inherent limitations of the Digital Signal Processor (DSP) the hearing aid is based on. The underlying reason is that most digital hearing aids have very strong constraints in terms of computational capacity, memory and battery, which seriously limit the implementation of advanced algorithms in them.

With this in mind, this thesis focuses on the design and implementation of a prototype for a digital hearing aid able to automatically classify the acoustic environments hearing aid users daily face on and select the amplification program that is best adapted to such environment ("self-adaptation"), aiming at enhancing the speech intelligibility perceived by the user.

The thesis can be divided into three major parts. The first one is related to the design of an automatic sound classification system that allows to properly discriminate the

input sound signal into speech, music and noise (the acoustic environments considered in this thesis). Note that not only this part involve the selection of the best suited feature set, but also the selection of the most appropriate classifying algorithm and the optimization of its parameters for its further implementation in the DSP. The second part deals with the design of an approach that aims at enhancing speech in hearing aids, not only in terms of speech intelligibility but also in terms of speech quality. Finally, the third part, probably the most important from the practical point of view, describes in detail the way both the automatic sound classification system and the speech enhancement approach are implemented in the DSP used to carry out the experiments.

The main contributions of this thesis are listed below:

- The design of a set consisting of low-complexity features. The key advantage regarding this feature set consists in that the number of instructions demanded by the DSP for its computation is extremely low.
- A feature-selection approach for sound classification in hearing aids through restricted search driven by genetic algorithms.
- A combined growing-pruning method for multilayer perceptrons (MLPs) that aims at finding the most appropriate number of hidden neurons in MLPs for a particular classification task.
- An algorithm for automatically selecting, among some piecewise linear approximations, the "approximated" activation function best suited for each of the hidden and output neurons comprising a multilayer perceptron.
- The design of a gain function aiming at speech enhancement in hearing aids, not only in terms of speech quality but also in terms of speech intelligibility. This gain function is created by using a gaussian mixture model fueled by a genetic algorithm.
- An approach that aims at "simplifying" the implementation of the compressor-expander algorithm (the very core of a hearing aid) in the DSP. The practical implementation of this approach consists in storing in the data-memory of the DSP, a table containing "tabulated" values of the gain to be applied as a function of both the input signal level (dB SPL) and the frequency band.

The final, global conclusion is that we have implemented a prototype for a digital hearing aid that automatically classifies the acoustic environment surrounding its user and selects the most appropriate amplification program for such environment, aiming at enhancing the sound quality perceived by the user. The battery life of this hearing aid is 140 hours (or equivalently, approximately 6 days), which has been found to be very similar to that of hearing aids in the market, and what is of key importance, there is still about 30 % of the DSP resources available for implementing other algorithms, such as, for instance, those involved in sound source separation or acoustic feedback reduction.

Resumen

La pérdida de audición es un problema que generalmente sufren las personas de mediana edad y las más mayores, como consecuencia del envejecimiento natural y deterioro del sistema auditivo humano. Esta pérdida de audición afecta de manera significativa a la comunicación e imposibilita a la mayoría de las personas con discapacidad auditiva a llevar una vida normal. Aunque la inmensa mayoría de los casos de pérdida de audición (90 %) podrían tratarse mediante la utilización de audífonos, sin embargo, pocas personas que podrían beneficiarse de su uso (concretamente un 20 %) compra uno, e incluso, muchos de los que han comprado su audífono terminan por no utilizarlo, a pesar del fuerte desembolso económico que puede suponer la adquisición y mantenimiento de un audífono.

Las razones para este aparentemente extraño comportamiento surge de la existencia de un problema fundamental, científico y tecnológico que, hasta día de hoy, no ha sido resuelto de manera eficiente y cómoda: la adaptación automática del audífono dependiendo del entorno sonoro en el que se encuentra el usuario. Se ha demostrado que la mayoría de las personas que hacen uso de audífonos prefieren tener una variedad de programas de amplificación adaptados a diferentes entornos sonoros, incluso aunque su funcionamiento no sea del todo perfecto. Hay dos posibles formas de satisfacer tal necesidad. Por un lado, existe el modo "manual", en el que el usuario tiene que identificar el entorno sonoro en el que se encuentra, y elegir él mismo el programa de amplificación más apropiado para ese entorno, a través de un conmutador situado en el propio audífono o de alguna clase de control remoto. Este modo es bastante incómodo y en la mayoría de los casos excede las habilidades de muchos de los usuarios de audífonos, especialmente las de los más mayores. El segundo modo, probablemente más cómodo para el usuario, consiste en que el propio audífono clasifique el entorno en el que se encuentra el usuario y él mismo seleccione el programa de amplificación más apropiado para ese entorno. Este último modo permite a los usuarios de los audífonos mejorar la inteligibilidad de voz percibida, llevar una vida normal e incrementar su nivel de comodidad.

Aunque parece evidente la necesidad de incluir un sistema de clasificación de sonidos en un audífono, su implementación es, por el contrario, algo difícil de conseguir. A pesar de los importantes avances en microelectrónica que ha habido en los últimos años, el desarrollo de un sistema de clasificación automática de sonidos en un audífono digital constituye todo un desafío debido a las limitaciones del procesador digital de señal (también conocido como DSP, de sus siglas en inglés, *Digital Signal Processor*) en el que se basa cualquier audífono digital. Nótese que los audífonos digitales tienes importantes restricciones en términos de capacidad computacional, memoria y batería, que limitan la implementación de algoritmos complejos en ellos.

Esta tesis se centra en el diseño e implementación de un prototipo de audífono digital capaz de clasificar automáticamente el entorno sonoro en el que se encuentra la persona que lleva el audífono y seleccionar el programa de amplificación mejor adaptado

a ese entorno, con el objetivo de mejorar la inteligibilidad de voz percebida por el usuario.

Esta tesis se puede dividir en tres partes. La primera de ellas está relacionada con el diseño de un sistema de clasificación automática de sonidos que permita clasificar correctamente la señal sonora de entrada entre voz, música y ruido (los entornos sonoros considerados en esta tesis). Nótese que esta parte no sólo implica la selección del conjunto de características más apropiado, sino también la selección del algoritmo de clasificación más adecuado y la optimización de sus parámetros para su futura implementación en el DSP. La segunda parte incluye el diseño de una función de ganancia que tiene como objetivo mejorar la percepción de la voz, no sólo en términos de inteligibilidad, sino también en términos de calidad. Finalmente, la última parte, quizás la más importante desde el punto de vista práctico, describe en detalle el modo en que, tanto el sistema de clasificación automática de sonidos y la función de mejora de la voz, son implementados en el DSP usado en esta tesis para llevar a cabo los experimentos.

A continuación se describen las principales contribuciones de esta tesis:

- El diseño de un conjunto de características de baja complejidad. La principal ventaja de este conjunto de características consiste en que el número de instrucciones requeridas por el DSP para su cálculo es muy bajo.
- Un método de selección de características para la clasificación de sonidos en audífonos digitales haciendo uso de algoritmos genéticos que restriguen el espacio de búsqueda.
- Un método que combina algoritmos de crecimiento y poda para perceptrones multicapa (también conocidos como MLPs, de sus siglas en inglés, *Multilayer Perceptrons*), que permite determinar el número idóneo de neuronas ocultas en MLPs para cada problema de clasificación.
- Un algoritmo que permite seleccionar automáticamente, entre varias aproximaciones lineales por tramos de la función de activación original (función *logarítmica sigmoidal*, en nuestro caso), la aproximación más apropiada para cada una de las neuronas de la capa oculta y de salida que forman un perceptrón multicapa.
- El diseño de una función de ganancia para audífonos que mejore tanto la calidad, así como la inteligibilidad de la voz percebida por los usuarios. Esta función de ganancia se genera usando un modelo de mezcla de gausianas en el que sus parámetros se estiman por medio de algoritmos genéticos.
- Un método que trata de "simplificar" la implementación del algoritmo de compresión-expansión (el núcleo principal de un audífono) en el DSP. La implementación práctica de este método consiste en almacenar en la memoria de datos del DSP, una matriz que contenga valores "tabulados" de la ganancia a aplicar en función del nivel de señal de entrada y de la banda de frequencia.

La conclusión final, global de esta tesis es que se ha desarrollado un prototipo de audífono digital que automáticamente clasifica el entorno sonoro en el que se encuentra su usuario y selecciona el programa de amplificación más apropiado para ese entorno, con el objetivo de aumentar tanto la calidad como la inteligibilidad de voz percibida por el usuario. La duración de la batería de este audífono es 140 horas (o equivalentemente, aproximadamente 6 días), valor muy similar a la duración de la batería de los audífonos disponibles en el mercado actualmente, y lo que es más importante aún, queda aproximadamente el 30 % de los recursos del DSP libre para la implementación de otros algoritmos, como, por ejemplo, aquellos que llevan a cabo separación de fuentes sonoras o la reducción de la realimentacióon acústica.

A todos los que habéis
escrito conmigo este libro

Agradecimientos

Hace unos años entré en contacto con la investigación, con la suerte de hacerlo de manos de alguien que disfruta trabajando. Desde entonces, en su entusiasmo y su laboriosidad, he tenido un referente magnífico para realizar mi trabajo con ilusión. Si la función de un director de tesis es guiar y aconsejar, no puede haber mejor forma de hacerlo que animando a curiosear y desarrollar las ideas propias con libertad. Muchas gracias, Enrique.

Muchas gracias a Roberto. Gracias por ofrecerme valiosas opiniones y consejos que me han hecho mejorar, aprender y avanzar por buen camino. Si esta tesis tuviera un segundo director, estoy segura que ése serías tú. Gracias a José Carlos por todos esos cafés y esas charlas que tanto me han animado. Gracias a Lucas por "enseñarme" a escribir como sólo Shakespeare sabía hacer. Y gracias también a Raúl y a Manuel, por sus ánimos y su apoyo.

Gracias a Cruza y a Leti, porque una gran parte de esta tesis ha sido también trabajo suyo y no hubiera podido llegado a ser lo que es sin sus aportaciones.

I would like to thank Professor Gianpaolo Evangelista for advising and guiding me through my research at Linköping University. My time at Norrköping was made enjoyable in large part due to people I met there: Lisa, Yoyo, Marc, Mahziar, Sara, Hiu and Kiruthika, among others. I am deeply grateful to them for kindly offering their help whenever possible. Thank you for making me feel like being at home.

A la Comunidad de Madrid, por la concesión de una contrato de investigación para realizar esta tesis.

No es menor la suerte que he tenido al compartir el día a día con personas maravillosas. Me gustaría agradecer a todos los compañeros del fondo S31 que me han animado, apoyado en mi trabajo, y que en definitiva han hecho más alegre el trayecto recorrido. Especialmente, quiero dar las gracias a Cosme y David. Con ellos he compartido mis alegrías y mis malos ratos de esta tesis, porque aunque ahora todo parezca maravilloso, este trabajo trae más de un dolor de cabeza. No tengo palabras para agradeceros lo bien que siempre os habéis portado conmigo. Voy a echar mucho de menos aquellos momentos en el Índalo y en el Hemisferio, y siempre recordaré con un sonrisa aquel "dulce noviembre". Cosme, muchas veces el camino es difícil, pero cuando terminas, la satisfacción y la alegría harán que te olvides de los malos momentos, mucho ánimo! Gracias también a Ramón, que aunque sin trabajar en nuestro fondo, ha sido un compañero más. Gracias por todos esos cafés que hemos compartido y por alegrarme el día a día con esas caritas sonrientes :).

Hay otros cafés que, aunque a veces "virtuales", me han rescatado del naufragio. Los de esas amigas que a pesar de la distancia siempre me hacen llegar su afecto. Gracias Eva, Cris y Cris, sin vosotras, que me comprendéis como nadie, nada habría sido lo mismo. Gracias también a Laura, Cris, Bea, Raquel y María Jesús, por comprender la importancia que esta tesis tenía para mí y por ello condicionar muchos fines de semana.

A Maite, por ser de esa clase de personas que dan lo mejor de sí mismas sin esperar nada a cambio, sabe escuchar y brindar ayuda cuando es necesario, y sobre todo, sabe ganarse el cariño, admiración y respeto de todo el que la conoce. Gracias también a Angie, ha estado ahí cuando más la he necesitado, y siempre le voy a estaré agradecida por ello.

A Leo, por preocuparse por mí y preguntarme cómo llevaba la tesis. Siempre le decía que ya me quedaba poco, pero hoy ya le puedo decir que por fin la he terminado!

A Raquel, Susi y Rocío, amigas desde pequeñas, las cuales siempre, y de manera incondicional, han estado a mi lado cuantas veces he necesitado su ayuda.

Gracias a Diego, Esther, Leti y Jesús, por todos los momentos, por vuestro apoyo, por vuestras sonrisas, vuestras confidencias y todo vuestro cariño. Gracias por haberme brindado vuestra amistad.

No me puedo olvidar de María Jesús y Nieves, con las que tantos ratos a última hora del día he compartido.

Y vaya mi mayor agradecimiento a mi familia. En especial, a mis padres, Basilio y Maribel, porque este trabajo es el resultado de lo que me habéis enseñado en la vida. Gracias por confiar siempre en mí y darme la oportunidad de culminar esta etapa. Gracias a mi hermana Raquel, por su encanto especial y por sus ánimos. Gracias por aguantar estoicamente mis malos ratos. A mis abuelos, porque sé que no hay nadie que se alegre tanto como ellos de que hoy termine este trabajo, y por supuesto, a mis tíos y a mis primos, por estar pendientes de mí a lo largo de todo este tiempo, brindándome su apoyo incondicional.

Gracias a todos aquellos que no están aquí, pero que me ayudaron a que este gran esfuerzo se volviera realidad.

Lorena

Contents

List of Figures

List of Tables

List of Symbols

g	Number of gaussian components in the gaussian mixture model
C	Number of output neurons in a multilayer perceptron
$\mathcal{C}$	Computational cost
$\mathbf{F}$	Feature vector
$\mathbf{F}_{\text{practical}}$	Feature vector programmed in the digital signal processor
$\mathcal{G}$	Gain matrix
$\mathcal{L}_{\text{average}}$	Load average
L	Number of input neurons in a multilayer perceptron
M	Number of hidden neurons in a multilayer perceptron
M_1	Number that represents the lower limit of speech frequency band
M_2	Number that represents the upper limit of speech frequency band
N_{B}	Number of frequency bands available in the digital signal processor
N_{L}	Number of input signal levels
N_{P}	Number of patterns available in a design set
N_{frames}	Number of frames into which the sound signal is segmented
$\mathcal{P}$	Training or design set
$P_{\text{avg}_i}(k)$	Average power of input frame i-th at frequency band k-th
$P_{\text{ins}_i}(k)$	Instantaneous power of input frame i-th at frequency band k-th
$\mathcal{T}$	Test set
$\mathcal{V}$	Validation set

Glossary

A

ADC	Analog-to-Digital Converter.
AFE	Acoustic Feedback Equivalent.
AI	Articulation Index.

B

BSD	Bark Spectral Distortion.
BTE	Behind The Ear.

C

CIC	Completely In-the-Canal.
CPU	Central Processing Unit.

D

DAC	Digital-to-Analog Converter.
DCT	Discrete Cosine Transform.
DFT	Discrete Fourier Transform.
DSP	Digital Signal Processor.

E

EPROM	Erasable Programmable Read-Only Memory.

F

FET	Field Effect Transistor.
FFT	Fast Fourier Transform.

G

GA	Genetic Algorithm.
GMM	Gaussian Mixture Model.

I

IC	Integrated Circuit.
IDFT	Inverse Discrete Fourier Transform.
IEEE	Institute of Electrical and Electronics Engineers.
IIR	Infinite Impulse Response.
ITC	In-The-Canal.
ITE	In The Ear.
ITU-R	ITU Radiocommunications Sector.
ITU-T	ITU Telecommunication Standardization Sector.

M

MAC	Multiplier/ACcumulator.
MFCC	Mel Frequency Cepstral Coefficients.
MIPS	Millions Instructions Per Second.
MLP	Multi-Layer Perceptron.
MNB	Measuring Normalizing Blocks.
MOS	Mean Opinion Score.
MSB	Most Significant Bit.
MSE	Mean Square Error.

N

NN	Neural Network.

P

PAMS	Perceptual Analysis Measurement System.
PAQM	Perceptual Audio Quality Measure.
PEAQ	Perceptual Evaluation of Audio Quality.
PESQ	Perceptual Evaluation of Speech Quality.
PSQM	Perceptual Speech Quality Measure.

R

RAM	Random-Access Memory.
RBF	Radial Basis Function.
RMSE	Root Mean Square Error.
ROM	Read Only Memory.

S

SII	Speech Intelligibility Index.
SNR	Signal to Noise Ratio.
SPL	Sound Pressure Level.
STE	Short Time Energy.
STFT	Short Time Fourier Transform.

U

USA	United States of America.

W

WOLA	Weighted Overlapp-Add.

Chapter 1

Introduction and scope

1.1 Background

According to the latest scientific statistics, hearing loss has become a global major healthcare concern. Nowadays, almost 600 million people worldwide, which amounts to approximately 10 % of the total population, suffer from hearing loss, and, regrettably, this number of deaf and hard of hearing people is increasing at an alarming rate not only because of the aging of the world's population, and consequently, the deterioration of the auditory system (in particular, most of elderly people suffer usually from specifically pernicious high-frequency hearing loss) [Gordon-Salant, 2005], but also because of the growing exposure to excessive noise in their quotidian life, such as, for instance, in the work environment (construction site, factory, etc.) or when they listen to loud music (MP3 players, iPod, concerts, etc.). Just in this respect, it is worth mentioning that it has been estimated that the total number of people suffering from hearing loss of more than 25 dB (the definition of hearing impairment recognized by the World Health Organization) will exceed 700 million worldwide by 2015, and this statistics also suggests that more than 900 million people will suffer from hearing loss of more than 25 dB in 2025 [Hear-it, 2011]. Figure 1.1 aims at illustrating this situation by showing the estimation of the number of people worldwide suffering from hearing loss of more than 25 dB from years 1995 to 2025.

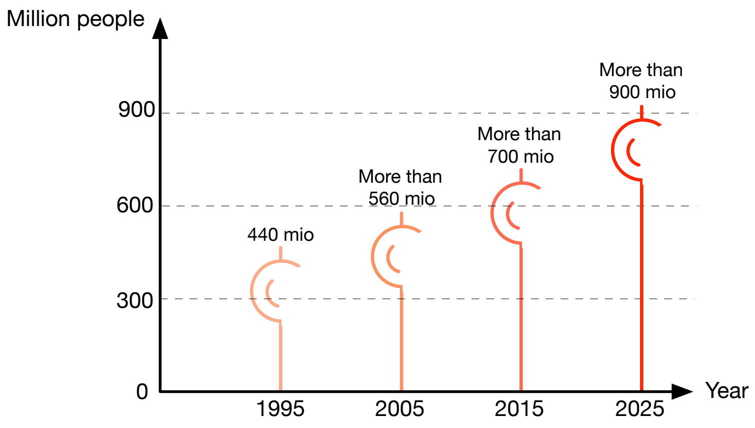


Figure 1.1: Estimation of the number of people worldwide suffering from hearing loss of more than 25 dB (the definition of hearing impairment recognized by the World Health Organization) from years 1995 to 2025 [Hear-it, 2011].

As clearly depicted in Figure 1.2, hearing loss is a problem that commonly suffer middle-aged and elderly people. In this sense, it is very illustrative to point out that about 17 % of American adults, or in others words, 36 million people, report some degree of hearing loss, there being a strong relationship between age and reported hearing loss: the National Institute of Health estimates that 8 % of American population between the ages of 18 and 44, 19 % of adults 45 to 64 years old, 30 % of adults 65 to 74 years old and 40 % over the age of 75 have significant hearing loss [NIH, 2008].

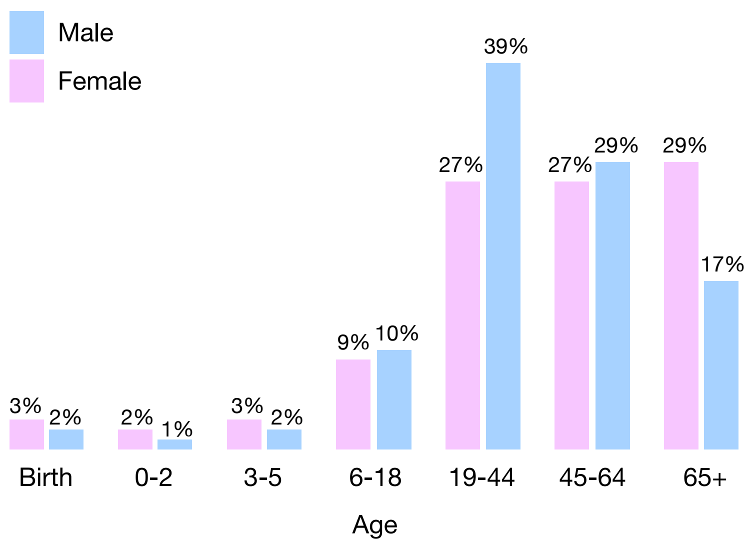


Figure 1.2: An illustrative representation of the percentage distribution, by sex and age, depicting the age at which hearing loss begins. This representation is based on the subjects' report of any permanent hearing loss [NHIS, 2002].

In Europe, for instance, about 71 million adults, aged between 18 and 80, have hearing loss greater than 25 dB. It is interesting to note that, only in European Union, the number of people suffering from hearing loss is more than 55 million, being the number of hearing-impaired people in Spain about 5.5 million of those. For comparative purposes, Table 1.1 shows the hearing loss statistics for some specific countries in Europe [Kochin, 2009; Shield, 2006a].

Country	**Million people**
Germany	10.2
France	7.6
United Kingdom	7.5
Italy	7.2
Spain	5.5
Poland	4.7
The Netherlands	2

Table 1.1: Number of people suffering from hearing loss for different European countries [Kochin, 2009; Shield, 2006a].

The question arising here is: *How much does hearing loss affect hearing-impaired people in their everyday life*? Addressing this question demands to mention that hearing loss severely affects the speech communication and disqualifies most of hearing-impaired people from holding a normal life, or in other words, hearing loss degrades considerably the quality of life of hearing-impaired people. In order to better understand what this assertion means, it is worth mentioning that hearing loss might result in psychological, physical and social disorders. Most of deaf and hard of hearing people, whose hearing loss has not been correctly treated, usually find it difficult to participate in social activities, even within their own family. According to [RNID, 1999], it has been reported that 71 % of hearing-impaired people feel isolated because of their hearing loss, 39 % avoid meeting new people and that 91 % have difficulties coping in public places.

For illustrative purposes, we summarize in the following paragraphs some common social problems affecting hearing-impaired people with untreated hearing loss:

- Isolation and withdrawal
- Inattentiveness
- Bluffing
- Distraction or lack of concentration
- Problems at work that many times involve ceasing working or retiring
- Problems participating in social life (and consequently, a reduced social activity)
- Communication problems with his wife or her husband, friends and relatives

The aforementioned problems become in serious negative psychological effects, such as, for instance:

- Shame, guilt and anger
- Embarrassment
- Problems concentrating
- Sadness or depression
- Worry and frustration
- Anxiety and suspiciousness
- Insecurity
- Self-criticism and low self-esteem or self-confidence

Additionally, from an economic point of view, untreated hearing loss costs Europe 213.000 million euros per year [Shield, 2006b]. In this sense, mild hearing loss costs 2.200 euros per individual each year, moderate hearing loss costs 6.600 euros annually per person, whereas a severe or profound hearing loss costs about 11.000 euros per person every year[1]. Note that, these figures do not include lost income and lost tax revenues due to unemployment or early retirement because of hearing loss.

[1]In the aim of facilitating the reader's comprehension, although will be explained in a detailed way in Chapter 2, it is worth mentioning that a person's hearing loss is generally categorized as being mild, moderate, severe or profound, depending upon how well a person can hear the intensities or frequencies most strongly associated with speech.

The good news is that 90 % of hearing loss cases could be mitigated by using some kind of hearing aid[1], whereas 10 % of them would require medical or surgical intervention. The bad news is that only about 5 % of people who could be benefited from hearing aids, wear it in their daily life.

These previous paragraphs probably make the reader not directly involved in hearing loss problems wonder, among others, the following two questions:

- If the vast majority of hearing problems (90 %) could be corrected with hearing aids, why do only a scarce 20 % of hearing-impaired people who could be benefited from digital hearing aids purchase one?
- Why do not 25 % of people who purchase a hearing aid wear it in their everyday life, despite the substantial financial outlay involved in purchasing and maintaining a hearing aid[2]?

This astonishing irregular use of hearing aids arises from a variety of problems, which may include not only sanitary arguments, but also social, economic and technological reasons. In [Kochkin, 2000], it was reported that some of the reasons were: poor benefit, background noise, negative side effects, price, sound quality and volume adjustments. In this sense, the most common complaints are listed below [Killion, 1997; Kochkin, 2000]:

- *"It does not work well in background noise"*
- *"I can't adjust the hearing aids constantly to every noise"*
- *"Volume is OK, but I can't distinguish words"*
- *"Hearing aids amplify other sounds so much that I actually feel pain"*

The abovementioned complaints are closely related to the following fact: hearing aid users face a variety of different acoustic environments in their quotidian life, such as, for instance, conversation in a quiet environment or speech in a noisy crowded cafe, noise, music, etc., as clearly represented in Figure 1.3; and although they expect authentic, significant benefits in each of the mentioned situations, regrettably, this is not the case.

To illustrate this, we can imagine, for the sake of simplicity, the following situation considered as being representative enough of the aforementioned problem: the person wearing the hearing aid goes from a quiet place to a noisy one, such as, for instance, a crowded cafe, where there are many talks, the TV sound and background noise like, for example, the coffee maker. Not only will the hearing aid amplify the speech but also the other sounds and thus, upon entering the cafe, the background noise will be extremely unpleasant for the hearing aid user. Although will be explained in detail in Chapter 2, we can say in advance that the problem becomes more accentuated because, due to the dynamic range compression performed by the hearing aid, understanding speech with background noise is much more difficult for hearing-impaired people than for healthy listeners [Moore, 1989].

In order to overcome this, some modern current digital hearing aids, especially the most sophisticated and high-end devices[3] allow their users to face a variety of different acoustic environments, or in other words, these devices have a variety of amplification programs fitted to

[1]Put it very simple, a hearing aid is basically an electronic device for sound amplification.

[2]Hearing aids are powered by batteries, which are not exactly cheap. Increasing the battery life consumed by hearing aids would be good news for their users.

[3]We say sophisticated devices in the sense that these instruments usually provide higher fidelity of sound, greater overall amplification, directional sound detection, dynamic compression and frequency-specific amplification.

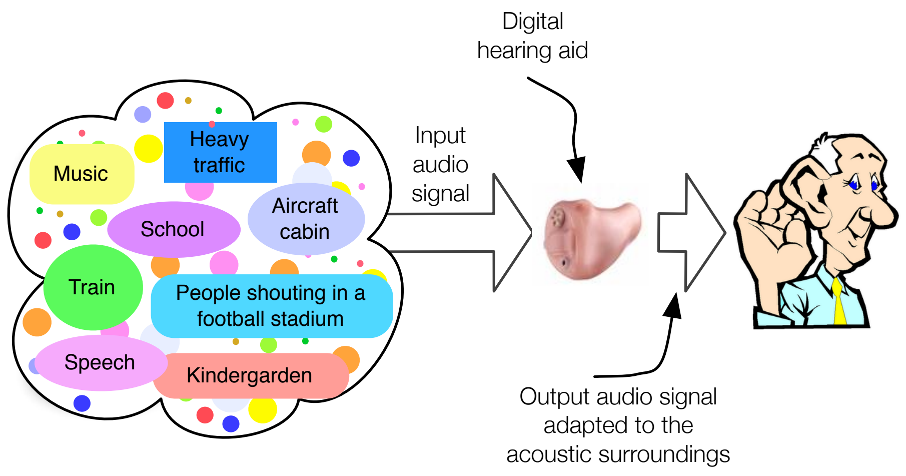


Figure 1.3: Artistic representation of the variety of sounds a hearing aid user faces daily. Please note that, apart from the speech signal (the general target of hearing aid users), a number of different sound sources has been represented, such as, for instance, traffic, school, people shouting in a football stadium, etc.

different acoustic situations, which can be manually selected by the user. The main drawback in these instruments is that, sometimes, they are uncomfortable and difficult to handle, especially for elderly people. In addition, these devices are very expensive (generally, their price is higher than 3.000 euros), making its acquisition and maintenance difficult for a particular social group, retired people (the main group of hearing aid users) who, unfortunately, have lower purchasing power. Many times, this scenario forces elderly people to purchase a low-cost device, which is worthless in many situations, like, for example, those in which an elderly grandfather or grandmother is with their grandchildren, or in a social gathering, in which the hearing aid only produces very unpleasant whistles, irritating howls and/or other amplified noises.

Therefore, we can say that there is a still crucial, scientific and technological problem in hearing aids that has not been solved efficiently and comfortably at low-cost: the automatic adaptation to the changing acoustic environment.

1.2 Motivation

Some studies have shown that hearing aid users generally prefer to have a number of amplification programs fitted to different listening conditions, even if its performance is not always perfect [Büchler, 2002; Keidser, 1996].

In the effort of satisfying such necessity, there are two basic approaches in digital hearing aids. As mentioned in the previous section, some modern digital multi-memory hearing aids in the market allow their users to face a variety of different acoustic environments, such as, for instance, conversation in quiet or conversation in the presence of background noise, music, etc. The main drawback in this approach is that the hearing aid user has to recognize the acoustic environment he or she is in and then choose the amplification program that best fits this situation by using a switch on the hearing instrument or some kind of remote control. Even for normal hearing people, it is not always clear which program should be selected to ensure the best performance and bearing in mind that elderly people are the main group of hearing aid users, it cannot be expected that they will always handle this task properly.

A second, feasible approach that could be deemed as very useful by most of hearing aid users is that in which the hearing aid *itself* classifies the acoustic environment that surrounds its user [Alexandre et al., 2006a, 2008c; Kates, 2008; Nordqvist, 2004; Nordqvist and Leijon, 2004] and automatically selects the amplification program that is best adapted to such surrounding [Alexandre et al., 2006a, 2008c]. This is referred to as “self-adaptation” and could significantly improve the usability of hearing aids, increasing *presumably* the speech intelligibility perceived by its user, and, what is of key importance, allowing him or her to hold a normal life. In fact, within the more general and highly relevant topic of sound classification in hearing aids [Kates, 2008; Nordqvist, 2004; Nordqvist and Leijon, 2004], self-adaptation is currently deemed as very appreciated by hearing aid users, especially by the elderly because, as mentioned, the “manual” approach very often exceeds their abilities [Büchler, 2002; Büchler et al., 2005].

Although the necessity of the aforementioned sound classification system seems to be clear, its implementation is, on the contrary, a very difficult matter. Despite the impressive advances in microelectronics, the development of an automatic sound classification system in a digital hearing aid is a challenging goal because of the inherent limitations of the digital signal processor (DSP) the hearing aid is based on. The underlying reason is that most digital hearing aids have very strong constraints in terms of computational capacity, memory and battery, which seriously limit the implementation of advanced algorithms in them. For the sake of clarity and for making this thesis self-contained, we have summarized in Appendix A the restricted resources of the particular DSP used by our platform to carry out the experiments. This restriction in number of operations per second enforces us throughout this entire thesis to put special emphasis on *signal processing techniques and algorithms tailored for properly classifying while using a number of operations as small as possible to bring down consumption and extend battery life.*

With these motivations in mind, we can proceed further in explaining the main *objective* of this thesis.

1.3 Objective

According to the concepts stated in the two previous sections and after the analysis of the state of the art about this topic, the following objective is set as principal in this thesis:

> *To design and implement a prototype for a digital hearing aid able to automatically classify the acoustic environments its user faces daily and select the “amplification program” that is best adapted to such environment (“self-adaptation”).*

As previously mentioned, hearing aid users face a variety of different acoustic environments in their everyday life and, although at a first glance, the reader could expect that we have implemented an amplification scheme for each one of those situations, this is not the case. This is basically due to the fact of programming on a digital hearing aid an amplification program for each of the acoustic listenings hearing aid users face on in their daily life would exceed the scarce DSP computational resources and thus, from a practical point of view, it has been found to be *unfeasible.*

In this respect, it is worth mentioning that we have considered in this thesis three different acoustic environments: speech, music and noise. We restrict the classification process to these three particular acoustic environments (and *not* to more environments) because these situations have been found enough by most hearing aid users aiming at holding a normal life. For the particular case of the listening environment consisting in speech, not only have we aimed at improving the speech intelligibility perceived by hearing-impaired people, but also the speech quality.

Some particular aims emerge from this main objective, which are presented as follows:

- To design a sound classification system that allows to properly classify the input sound signal into speech, music or noise, taking into account the aforementioned severe constraints digital hearing aids suffer from. Note that not only does this aim involve the selection of the best suited feature set, but also the selection of the most appropriate classifying algorithm and the optimization of its parameters.

- To design an approach that aims at speech enhancement in hearing aids, not only in terms of speech intelligibility but also in terms of speech quality.

- To clearly quantify the effects of finite-precision limitations on the performance of the automatic sound classification system prior to its practical implementation in the DSP.

- To implement the overall automatic sound classification system in the DSP used by our platform to carry out the experiments.

1.4 Organization

The rest of this thesis describes the research that has been undertaken to carry out the above-mentioned aims. In this respect, the remaining chapters are organized and presented as follows:

- Chapter 2: this chapter illustrates the way the sound is perceived the by the human auditory system, which will assist us in introducing the types and degrees of hearing loss. It is also shown here a brief review of hearing aids history, starting from large, horn-shaped devices to lightweight and almost invisible current assistive medical devices, putting special emphasis on the common models currently available in the market.

- Chapter 3: in the effort of facilitating the reader's comprehension of the remainder of this thesis, this chapter describes the blocks involved in the implementation of an automatic sound classification system for hearing aids, which basically include: 1) the data processing stage, 2) the feature extraction stage and 3) the classifying algorithm *itself*. For the sake of clarity, the basic concept underlying the data processing stage, along with the cornerstone ideas describing the feature extraction block and the classifying algorithm are also described in this chapter. With this in mind, this chapter presents a summary of a set of well-known spectral sound signal-describing features, proposed in the literature, for the purpose of sound classification in hearing aids. With regard to the classifying algorithms, this chapter describes a variety of widely-used classifiers that exhibit good performance for the purpose of sound classification in hearing aids.

- Chapter 4: aiming at reducing the computational cost associated to the programming in the DSP of the set of features described in Chapter 3, a set of low-complexity features is proposed in this chapter. These novel features are based on a "variation" of some of the features explored in the aforementioned chapter. Since the fact of programming in the DSP all the features explored in this thesis has been found to be unfeasible from a practical implementation point of view, this chapter also proposes a novel feature-selection approach for sound classification in hearing aids through restricted search driven by genetic algorithms, or in other words, the goal of this approach consists in finding a feature subset that assists the overall classification system in reducing its computational cost without degrading its ability to classify sounds into speech, music and noise.

- Chapter 5: this chapter explores the feasibility of the classifying algorithms described in Chapter 4 for the speech/music/noise classification task in hearing aids. In this respect, a number of experiments are carried out aiming at evaluating the performance of these classifiers. The aim of this experimental work is to determine the classifying algorithm that best performs for the aforementioned classification task.

- Chapter 6: this chapter intends to clearly quantify the effects of finite precision limitations on the overall classification performance of the automatic sound classification system designed in this thesis, with special emphasis on the effects of finite word length for the sound signal-describing features and the parameters of the classifying algorithm.

- Chapter 7: this chapter proposes a novel approach aiming at speech enhancement in hearing aids, not only in terms of speech quality but also in terms of speech intelligibility, regardless the user's hearing loss. To what extent the proposed approach enhances the speech quality and speech intelligibility perceived by the user is measured by means of two standardized objective measures.

- Chapter 8: one important question that may arise is whether the overall classification system could be implemented in real-time on an in-the-market, average performance hearing aid. Elucidating this is the objective of this chapter. In particular, this chapter clearly illustrates the way each functional block, involved in the implementation of an automatic sound classifier embedded in a self-adapting hearing aid, is programmed in the DSP used by our platform to carry out the experiments.

- Chapter 9: finally, the conclusions obtained throughout this work are shown, including some guidelines for research lines opened by this thesis. A list of published work derived from this thesis is also illustrated.

Additionally, five appendices have been added that include supplementary information without disrupting the development of the main argument. These appendices are as follows:

- Appendix A: this appendix describes the digital signal processor (DSP) used by our platform to carry out the experiments.

- Appendix B: this appendix describes, in a detailed way, the hearing aid simulator we will make use of in order to evaluate the performance of the different hearing-aid processing algorithms proposed in this thesis, prior to their implementation in the DSP.

- Appendix C: this appendix describes the two databases used to carry out the experiments in this thesis. These two databases are as follows:

  1. A "sound database" containing 7299 audio files, including speech-in-quiet, speech-in-noise, speech-in-music, vocal music, instrumental music and noise files, each of 2.5 seconds length.
  2. A "patient database", which collects data from 18 *real* hearing aid users, with different types of hearing loss, including mild, moderate, severe and profound hearing loss.

- Appendix D: since a number of experiments carried out in this thesis make use of a genetic algorithm (GA), this appendix aims at introducing the main concepts related to this successful technique.

- Appendix E: this last appendix aims at helping the reader understand the assembly code programmed in this thesis for the computation of the feature vector that feeds the classifying algorithm.

Chapter 2

Hearing aids

2.1 Introduction

A hearing aid is basically an electronic device that intends for partially overcoming auditory deficits and is commonly employed to compensate the particular hearing loss any person suffers from. The main objective of a hearing aid is to fit the dynamic range of speech and some common quotidian sounds, such as, for instance, household noises (dishwasher, water dripping or noise from a vacuum cleaner), or environmental noises (aircrafts, trains, road vehicles, etc.), into the *reduced* dynamic range of hearing-impaired people, as will be clearly shown later on. In the effort of achieving a better understanding of these instruments and their functionality, the first section of this chapter will explain in detail how sound is perceived by the human auditory system and, which exactly the problems encountered by hearing-impaired people are. Note that the location of the disease within the peripheral auditory system[1] will assist us in introducing the types and the degrees of hearing loss.

As the reader may intuitively imagine, current hearing aids have changed considerably since the first large, horn-shaped devices from more than two hundred years ago. Apart from the fact that they have become much smaller and practically invisible to the casual observer[2], there is a still *noticeable* change concerning to current hearing aids. This change is due to the introduction of digital signal processing into hearing aids in 1996 [Edwards, 2007], which allowed signal processing algorithms could be programmed in hearing aids in the aim of increasing the speech intelligibility perceived by their users. Note that speech signals in hearing aids can be also processed in the analog domain but digital domain offers high speed, better accuracy, greater flexibility and increased storing capabilities. For illustrative purposes, and facilitating the reader's comprehension, the second section of this chapter will show a brief review of hearing aids history, putting special emphasis on the main benefits of current digital hearing aids.

Since current hearing aids may vary in size, power and circuitry, the last section of this chapter will help us explain the main models of hearing aids currently available in the market. As will be shown throughout this last section, roughly speaking, the smaller the hearing aid is, the better benefits it exhibits (not only in terms of being smaller in size but also in terms of better sound quality perceived by the user), but at the expense of generally being more expensive.

With this in mind, this chapter is structured in four sections. Section 2.2 shows the way sound is perceived by the human auditory system, and, which exactly the problems encountered

[1]The human auditory system is composed of two parts: 1) the peripheral auditory system and 2) the central nervous system.

[2]The demand of making them smaller and less conspicuous has been a constant driving force behind technological progress. Sometimes, these size reductions have been achieved at the expense of their performance, but over time, performance has been improved despite, and sometimes because of, the size reduction [Dillon, 2001].

by hearing-impaired people are. This will assist us in introducing the types and the degrees of hearing loss. Section 2.3 shows a brief review of hearing aids history, starting from large, horn-shaped devices to lightweight and almost invisible to the casual observer devices available in the market currently. Finally, Section 2.4 describes the main models of hearing aids available currently in the market.

2.2 Hearing loss

Hearing loss is defined as the reduced ability to hear sound signals, and can happen in one ear or in both ears. Prior to describing the types of hearing loss, it is interesting to have a brief look at the way sound is perceived by the human auditory system, since problems in any part of this system may lead to loss of hearing.

2.2.1 Description of human auditory system

The human auditory system is the responsible for transmitting sound from outside the ear to the auditory nerve, which sends the information as an electrical impulse to the brain, where it is processed and interpreted as sound. Although, strictly speaking, the human auditory system consists of both the auditory periphery system, or in other words, the human ear, and the central nervous system comprising infinitely many neurons, we will focus here on the peripheral auditory system, since most of hearing loss cases are due to problems in this part of the human auditory system.

The peripheral auditory system is divided anatomically into three parts, as clearly depicted in Figure 2.1. These parts are as follows:

- The *outer ear* consists of both the visible part of the ear, named "pinna", and the external auditory canal. What we call "sound signals" or simply "sounds", are actually just sound waves, which are transmitted by the air. These sound waves are collected in the outer ear and guided through the air-filled external auditory canal to the eardrum. The eardrum is a flexible, circular membrane that vibrates when sound waves strike it.

- The *middle ear* contains, apart from the aforementioned eardrum, three tiny bones: malleus, incus and stapes, often referred to as the hammer, the anvil and the stirrup, respectively. They are collectively known as the ossicles. The ossicles also vibrate in response to movements of the eardrum and in doing so, amplify and guide the sound to the inner ear by means of the oval window.

- The *inner ear* basically consists of the cochlea, which is similar in shape to a snail shell. It contains several membranous sections filled with fluids. When the ossicles conduct the sound to the oval window, the fluid begins to move, stimulating the tiny hearing nerve cells (also called "hair cells") inside the cochlea. These hair cells in turn send electrical impulses by means of the auditory nerve to the brain, where they will be finally interpreted as sound.

Perhaps the reader may wonder what the relationship between the types of hearing loss and some problems in any of the mentioned parts of the periphery human ear is. Elucidating this is just the objective of the following subsection.

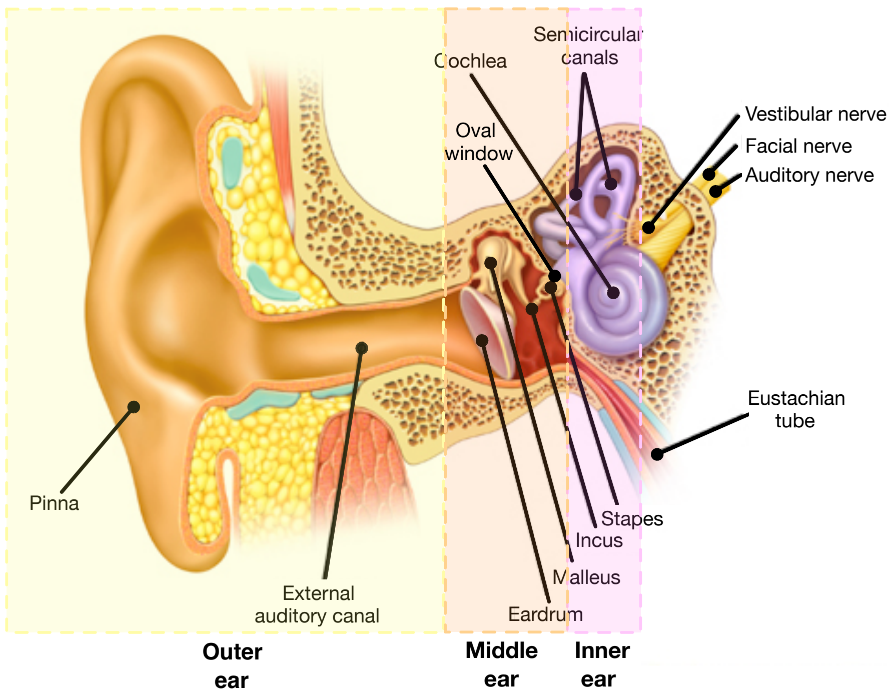


Figure 2.1: A detailed aspect of the anatomy of the peripheral human auditory system. As shown, it basically consists of three parts: the outer ear, the middle ear and the inner ear.

2.2.2 Hearing loss classification

There are three different types of hearing loss according to the "location" of the disease within the peripheral auditory system:

- *Conductive* hearing loss is the most common type of hearing loss and happens when sound is not guided *efficiently* through either the external auditory canal or the middle ear. This type of hearing loss can be originated from an abnormality of development, such as, for instance, the absence or incomplete formation of a part of the outer or middle ear system, or caused by a disease within the outer or the middle ear like, for example, severe o continuous otitis media.

  Conductive hearing loss usually involves a reduction in the sound level or ability to hear faint sounds. Fortunately, this type of hearing loss can often be medically or surgically corrected.

- *Sensorineural* hearing loss happens when there is a damage either to the inner ear or the nerve pathways from the inner ear to the brain. It can be caused by either an abnormality of development or a disease affecting the cochlea or auditory nerve.

  Not only does sensorineural hearing loss involve a reduction in sound level or ability to hear faint sounds, but it also affects speech understanding or ability to hear clearly. Regrettably, this type of hearing loss cannot be treated by current medical or surgical techniques and consequently, it is a *permanent* hearing loss.

- *Mixed* hearing loss is caused, intuitively, by a combination of both sensorineural and conductive factors. This type of hearing loss usually happens when a person firstly has sensorineural hearing loss and then develops conductive hearing loss (to understand this, it is convenient to imagine a person who has already sensorineural loss and gets a middle ear infection; these two types of loss combined result in a greater hearing loss), or other times, may be the result of the outer and inner ear malformation, which causes both types of hearing loss.

  In general, the conductive part of a mixed loss may be treated by medical or surgical techniques, whereas the sensorineural part is usually permanent and cannot be medically treatable.

2.2.3 Degrees of hearing loss

In general, the degree of hearing loss can be classified into different categories. In this respect, it is very illustrative to notice how a person may hear some speech sounds but find soft sounds difficult to hear, whereas other person, for instance, may hear only loud sounds.

The degree of hearing loss is commonly based on the average hearing loss measured at a particular octave-band, ranging the level of loss from slight to profound. Although there is no a standard classification system that is universally used, a very well-known system is the one recognized by the American Speech-Language-Hearing Association [GAO, 2011]. According to this classification system, hearing loss, measured in decibels (dB HL), is defined as the average hearing loss at frequencies 250, 500, 1000, 2000 and 4000 Hz.

For illustrative purposes, this system is represented in Table 2.1, showing also to what extent each degree of hearing loss affects speech communication and whether the use of hearing aids could be beneficial or not to compensate that degree of acoustic loss. As depicted, for hearing impaired people suffering from *slight* to *mild* hearing loss, a hearing aid is needed in some specific situations, or at least on a frequent basis. For *moderate* hearing loss cases, a hearing aid is needed for all communications, whereas for *severe* hearing loss cases, the use of a hearing aid may be combined with speech-reading (lip-reading) or sign language. Regrettably, for *profound* hearing loss cases, the use of hearing aids is of little benefit. For this latter degree of hearing loss, cochlear implants may be considered.

2.2.4 Problems faced by hearing-impaired people

We put forward in the introductory chapter that hearing-impaired people face a variety of different problems in their everyday life when they do not make use of hearing aids. Deepening a little more in these problems and in an effort of better understanding the main goals of hearing aids, we summarize in the following paragraphs the main difficulties that hard of hearing people face in their everyday life [Dillon, 2001].

- **Decreased audibility**
  As stated beforehand, whereas a person with slight or mild hearing loss will hear some speech sounds but find soft sounds difficult to be heard, a person with moderate hearing loss will probably hear only *few* sounds, and the one suffering from severe hearing loss will not hear any speech sounds unless they are shouted.

  In this respect, softer phonemes, basically consonants, may be difficult to be heard. In the aim of facilitating the reader's comprehension, let us imagine, for instance, the sequence of sounds *i e a ar* that might have been originated as "pick the black harp" and could

Degree of hearing loss	Hearing loss (dB HL)	Effect on speech communication	Use of hearing aid
Normal	−10-15	no hearing impairment	–
Slight	16-25	speech understanding not affected	–
Mild	26-40	speech understanding reduced, especially in noisy environments	may be helpful
Moderate	41-55	conversation noticeably difficult	very beneficial
Moderately severe	56-70	speakers must raise their voice to be heard	essential
Severe	71-90	conversational speech cannot be heard	of less benefit
Profound	91 and above	deaf	of little benefit; cochlear implants may be considered

Table 2.1: Classification system for the degree of hearing loss. According to the American Speech-Language-Hearing Association, hearing loss, measured in decibels (dB HL), is defined as the average hearing loss at frequencies 250, 500, 1000, 2000 and 4000 Hz. It is also represented to what extent each degree of hearing loss affects speech communication and whether the use of hearing aids could be beneficial or not to compensate that degree of hearing loss.

have been heard as “kick the cat hard”. For hard of hearing people, essential parts of some phonemes are not perceptible and they recognize sounds by noting which frequencies contain the most energy. In general, the high-frequency components of speech are weaker than the low-frequency components and consequently, hearing-impaired people tend to miss high-frequency information. Figure 2.2 aims at illustrating this situation by showing the so-called “speech-banana” that depicts how vowels and consonants (in English language) spoken with normal loudness are experienced at a distance of one meter from the speaker. It seems clear to notice that vowels have a higher intensity level than consonants and are hence easier to be heard. The picture also includes some drawings depicting typical quotidian sounds, such as, for instance, a bird chirping, a dog barking, a baby crying, a telephone ringing, fall leaves or jet planes. Just in this respect, it is very clear to note that jet planes are loud, high frequency sounds while wind rustling leaves is a much lower sound.

With these considerations in mind, hearing aids intend for providing more gain at frequencies where speech has the weakest components (usually high frequencies). Or in other words, hearing aids provide different amount of gain in different frequency regions.

- **Reduced dynamic range**
  Although not clear at first glance, it is interesting to note that it is not always appropriate to amplify soft sounds by the amount needed to make them perceptible. The reason is as follows. The “dynamic range” of the ear is defined as the level difference between the

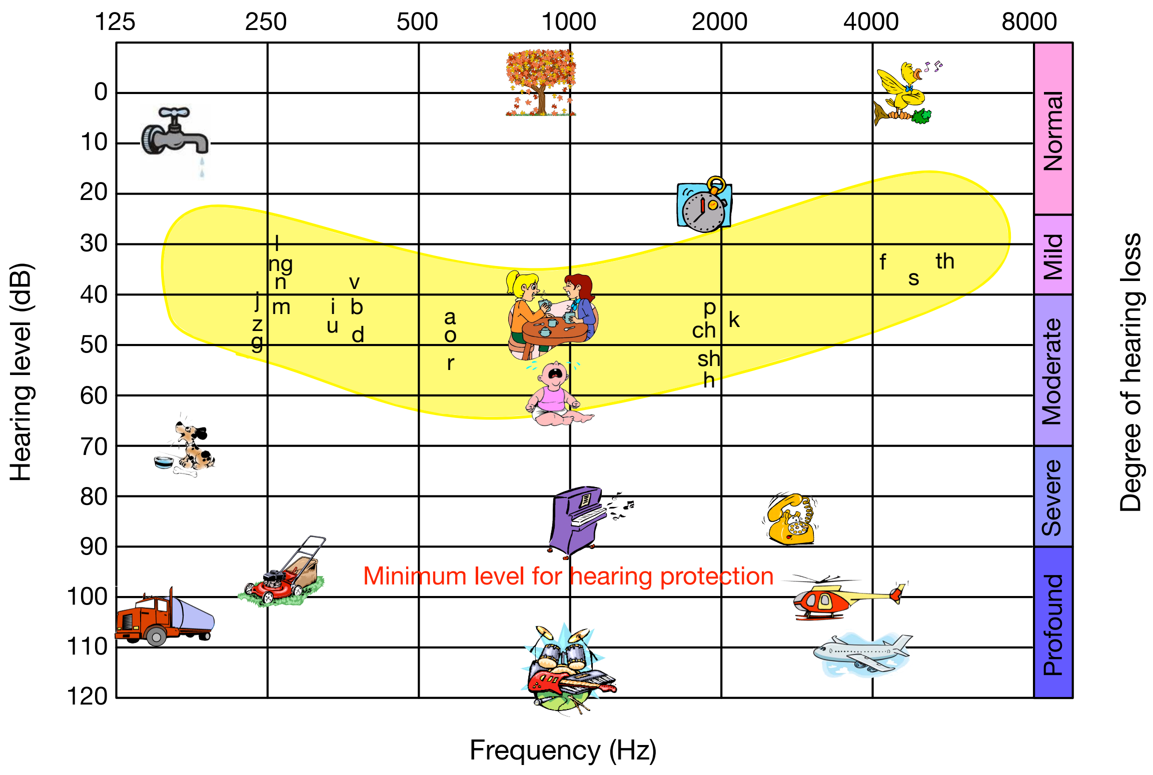
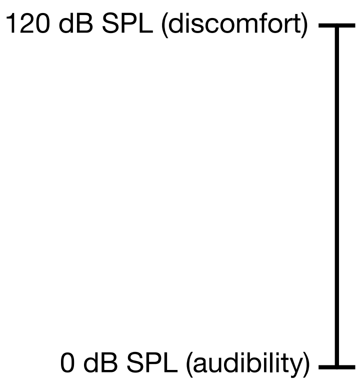


Figure 2.2: The "shaded" area designates the so-called "speech-banana" that essentially depicts the most important area for the perception of the most of English vowels and consonants as a function of both the hearing level (dB HL) and frequency (Hz). For illustrative purposes, this picture also includes some drawings depicting typical sounds, such as, for instance, a bird chirping, a dog barking, a baby crying or a telephone ringing.

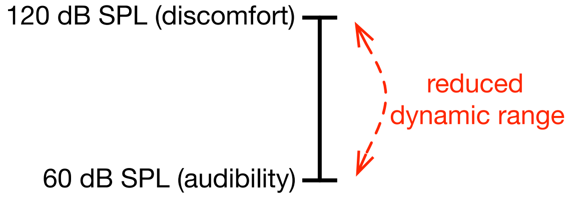
audibility threshold and the discomfort threshold (sometimes, also called "threshold of pain"). Since for hearing-impaired people, the audibility threshold is increased much more than that for normal hearing people, their dynamic range is thus more reduced than for healthy listeners. This can be clearly noticed in Figure 2.3. Figure 2.3(a) illustrates the dynamic range for a normal hearing person, whereas Figure 2.3(b) depicts the dynamic range for a hearing-impaired person.

Since the different sounds hearing-impaired people face daily must be fit within the mentioned restricted dynamic range, hearing aids must thus amplify weak sounds more than intense sounds, as clearly represented in Figure 2.4. Although will be explained in detail in Section 2.3.5, we can say in advance that this is the main purpose of the compression-expansion algorithm implemented in hearing aids, which aims at compensating for the reduced dynamic range in the impaired ear not only by applying a frequency-dependent gain but also a signal level-dependent gain.

- **Reduced frequency resolution**
  Hearing-impaired people have great difficulty for distinguishing between a variety of sounds of different frequencies simultaneously. To illustrate this situation, let us imagine in the frequency-domain sound signal, a speech and a noise component that have different fre-

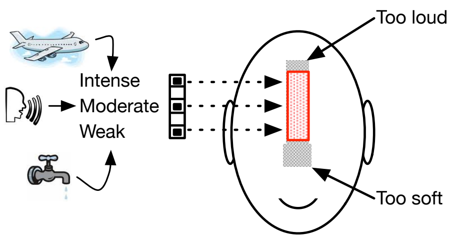


(a) Dynamic range for a normal hearing person.

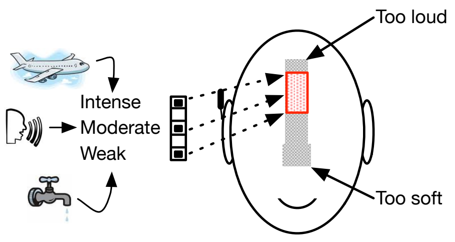


(b) Dynamic range for a hearing-impaired person.

Figure 2.3: In (a), it is illustrated the dynamic range for a normal hearing person, whereas in (b), it is shown the dynamic range for a hearing-impaired person, which is much more reduced than that for a healthy listener.



(a) Amplified sounds for a healthy listener.



(b) Amplified sounds for a hearing-impaired person.

Figure 2.4: Illustrative picture depicting the way in which some sounds should be amplified for: (a) a healthy listener and (b) a hearing-impaired person. Specifically, it is shown the spacing of different sound levels before their amplification (the left end of the dashed lines) and after their amplification (the right end of the dashed lines) for both a normal and a hearing-impaired person.

quencies which are close enough. In this scenario, the cochlea will have a single broad region of activity rather than two finely tuned separate regions and the brain will be probably unable to discern the signal from the noise.

Since frequency resolution gradually decreases as the degree of hearing loss increases, hearing-impaired people usually experience a decreased frequency resolution [Dillon, 2001]. This decreased frequency resolution is partially due to 1) the loss of ability of hair cells to increased sensitivity of the cochlea for tuning frequencies (or in other words, frequencies at which the affected part of the cochlea is tuned) and 2) their need to listen at elevated level[1].

Some solutions, which we summarize below, have been proposed in the literature aiming at hearing aids overcoming these problems:

[1]Even for healthy listeners, their frequency resolution is poorer at high intensity input signal levels than at low input signal levels.

  - The use of noise reduction schemes.
  - The use of directional microphones to focus on the desired sounds and eliminate those coming from undesired directions.
  - Variation of the gain with the frequency aiming at low-frequency components of speech or noise *do not* mask high-frequency components.

- **Decreased time resolution**
  In general, weaker sounds may often be masked by intense sounds that immediately precede or follow them, which involves that the speech intelligibility perceived by hearing-impaired people is decreased.

  Since the ability to hear weak sounds during short time slots gradually decreases as the degree of hearing loss increases, hearing-impaired people usually experience decreased time resolution. In the effort of compensating this decreased temporal resolution, hearing aids perform the so-called "fast-acting compression", where the gain is rapidly increased or decreased during weak or intense sounds, respectively. The main drawback of this method is that undesired background noise is also amplified and thus, made more audible.

All these abovementioned problems combined together can cause significant reduction in the speech intelligibility perceived by hearing-impaired people. In this respect and as it was advanced in the introductory chapter, it has been estimated that a person who suffers from moderate or severe hearing loss requires, respectively, a SNR of about 5-10 dB higher than that required by a normal hearing listener in an effort of achieving the same level of speech understanding [den Bogaert et al., 2009].

Therefore, it seems to be apparent from the previous paragraphs the necessity for hearing-impaired people to make use of hearing aids in order to overcome this kind of problems.

2.3 A brief history of hearing aids

As succinctly stated in the Introduction, hearing aids have experienced major changes as time goes by. Just in this respect, their history may be divided into five periods or eras, as shown below:

1. Acoustic era
2. Carbon era
3. Vacuum tube era
4. Analog era
5. Digital era

Figure 2.5 will assist us in explaining when each of the mentioned eras have began in the history and whether hearing aids designed in these eras were wearable or not, because, as will be shown later on, not all hearing aids designed in the history have been wearable.

A more detailed description of this variety of hearing aids can be found in [Bennion, 1994; Bentler and Duve, 2000; Dillon, 2001] and the excellent online resources in [Bauman, 2011].

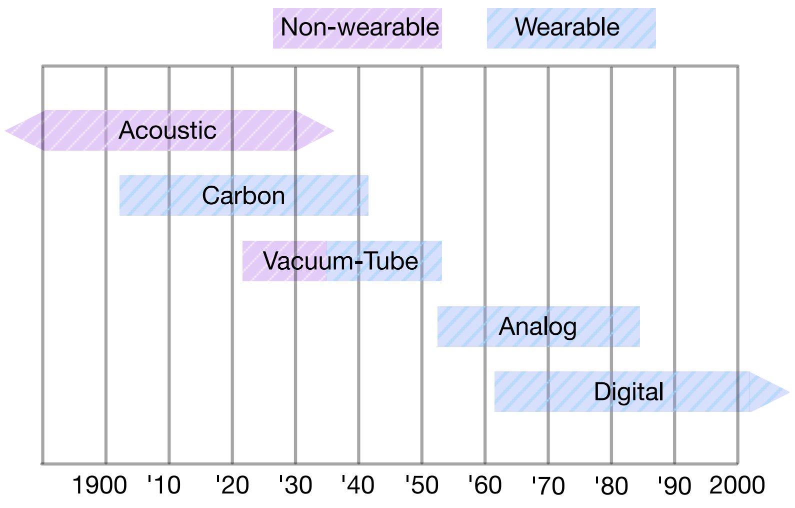


Figure 2.5: Picture showing when principal hearing aid eras have began in the history. It is also represented whether hearing aids designed in these eras were wearable or not.

2.3.1 The acoustic era

The acoustic era began the first time someone cupped a hand behind an ear. Collecting sound from an area larger than the ear's area may produce about 5-10 dB of gain at middle and high frequencies.

In the middle of the 17th century, more "sophisticated" acoustic aids appeared. They were formed by anything with a shape like, for example, a *trumpet* (depicted in Figure 2.6(a)), *horn* or *funnel*. The basic idea underlying these instruments was to have a large open end to collect as much sound as possible. The energy was transferred to the ear by means of a gradual reduction in area along the length of the trumpet or funnel[1]. It is very convenient to note that these instruments had to be both wide and long.

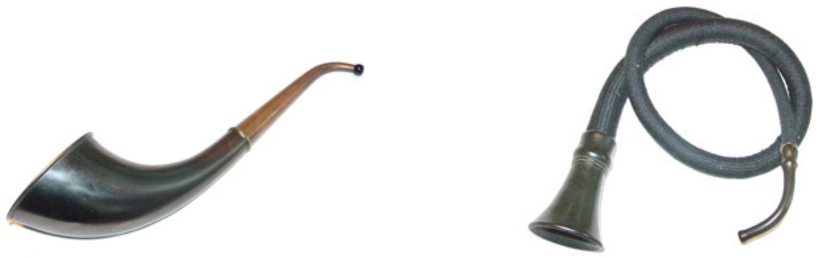
Deepening a little more in these instruments, it is worth noting that if the open end of these acoustic hearing aids was moved closer to the talker, it collected more intense sound as well as a greater area of sound. In order to do this, the speaking tube, shown in Figure 2.6(b), was designed. It basically consisted of a horn-shaped end attached to a long tube that terminated in an earpiece. If the talker spoke directly into the horn end, the SNR at the input to the device was much better (about 15-30 dB higher) than that obtained when the listener received naturally.

2.3.2 The carbon era

A carbon hearing aid, in its simplest form, consisted of a carbon microphone, a battery of $3-6$ V and a magnetic receiver, all components connected in series. Although the way this instrument worked is out of the scope of this thesis, it is worth mentioning that the level at the output of the receiver was about 20-30 dB greater than the level at the input to the microphone.

Aiming at achieving greater gain, the "carbon amplifier" was invented. If one microphone

[1]If the area decreases too quickly, most of the sound just reflects back out again instead of traveling into the ear.



(a) The ear trumpet (b) The speaking tube

Figure 2.6: Picture illustrating two "hearing aids" from the acoustic era. In (a), it is represented the so-called ear trumpet, and in (b), it is illustrated the speaking or conversation tube.

and receiver pair could increase the sound level about 20-30 dB, in the same line of reasoning, a second pair (but with only a single diaphragm in common) could increase it much more.

In this respect, the first carbon amplifier hearing aid (a large table model) appeared in 1899, and the first practical commercially wearable model, shown in Figure 2.7, appeared shortly after in 1902. It used a carbon microphone and a battery, which was as large as a desk radio. Carbon hearing aids continued to be used through to the 1940s, being beneficial only for people who had mild or moderate hearing loss.

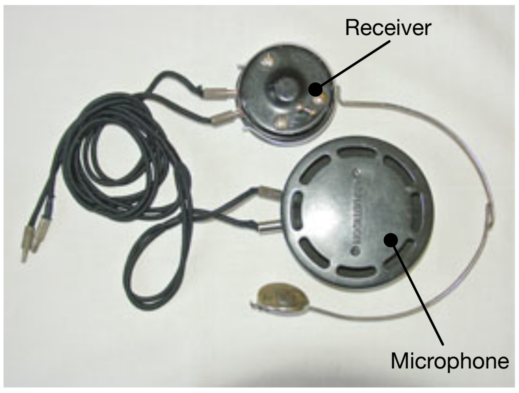


Figure 2.7: The first practical commercially wearable carbon hearing aid, called the Akouphone or Acousticon, appeared in USA in 1902. The battery, which was as large as a desk radio, is not shown in the figure.

The key point to highlight in this era is that the idea of amplifying different frequencies by different gain amounts emerged. This was basically achieved by selecting different combinations of microphones, receivers and amplifiers.

2.3.3 The vacuum tube era

The vacuum tube electronic amplifier was invented in 1907 and applied to hearing aids in 1920. By combining several vacuum tubes, one after the other, the resulting system was a very powerful amplifier (70 dB gain and 130 dB SPL output), increasing the range of hearing loss that could be treated.

The main drawback of vacuum tube aid devices was their large size. Although, thanks to military requirements, the size of these devices was significantly reduced over time, however, two batteries were needed to make them work. Vacuum tube hearing aids became practical during the 1930s, but until 1944, their batteries were so large that they had to be housed separately from the microphone and amplifier. In 1944, both vacuum tube and battery technology had advanced sufficiently to make possible a one-piece hearing aid. Batteries, microphone and amplifier were combined into a single body-worn electrical hearing aid, which connected to a receiver placed in the ear by means of a wire. For the sake of clarity, Figure 2.8 shows a relatively late vacuum tube hearing aid with the receiver, amplifier and its two separate batteries.

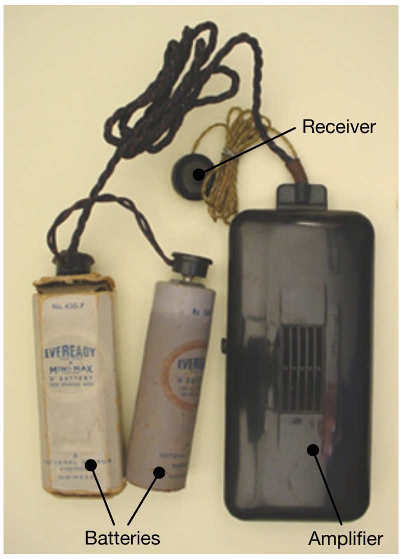


Figure 2.8: A relatively late vacuum tube hearing aid. The microphone, amplifier and the two batteries were combined into a single body-worn electrical hearing aid, which connected to a receiver placed in the ear by means of a wire.

It is worth mentioning that important concepts concerning to current hearing aids such as, for instance, earmold venting, magnetic microphones, piezoelectric microphones or compression amplification were already devised during this era. This early origin of compression amplification is surprising; compression seems to have rapidly become largely forgotten until the 1980s, but then became the dominant type of advanced amplification schemes in the late 1990s.

2.3.4 The analog era

With the introduction of the transistor in the market in 1952, there was a such significant reduction in terms of battery size, that all devices designed in the middle 1950s used transistors rather than valves. Not only did this reduction in battery size and the small size of transistors (when compared to valves) allow that all elements composing a hearing aid could be moved up to the head but it also extended the battery life, the most important constraint for designing advanced algorithms in hearing aids.

A simplified model of an analog hearing aid is shown in Figure 2.9. As clearly illustrated, it is basically composed of:

- A microphone to convert the input sound into an electrical signal.
- An amplifier to intensify the electrical signal in the aim of compensating the acoustic loss the user suffers from. Please note that the core of this amplifier essentially consists of one or more stages of transistors and resistors.
- A tiny loudspeaker to convert the amplified signal back to sound.
- A small battery to supply electric power to the electronic devices that compose the hearing instrument.

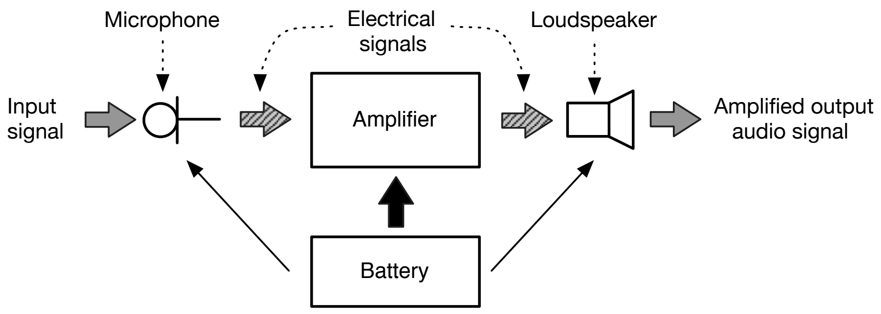


Figure 2.9: A simplified diagram of the structure of a typical analog hearing aid. As illustrated, it basically consists of a microphone, an amplifier, a loudspeaker and a small battery.

The idea of packaging the microphone, amplifier and batteries into one piece with a single wire to the receiver had some advantages such as, for instance, clothing did not create noise as a consequence of clothes rubbing against the microphone, the body did not have such adverse effects on the tonal balance of sound coming from different direction, large wires were no longer required or true binaural hearing aids were physically possible.

The first analog hearing aids worn completely in the head were the so-called barrettes (which had an external receiver like a body aid) or eyeglass devices, represented in Figure 2.10(a) and 2.10(b), respectively.

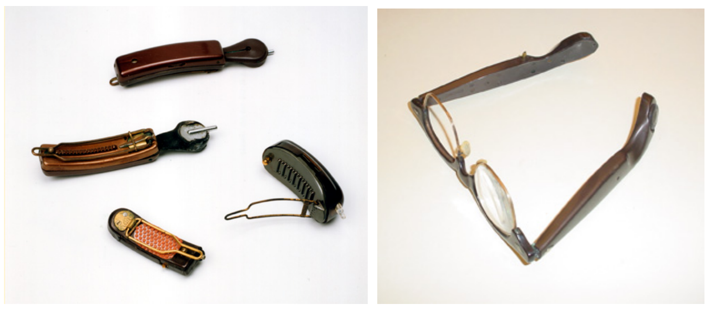


Figure 2.10: First analog hearing aids worn completely in the head. In (a), it is shown several barrette hearing aids, which had an external receiver like a body aid, and in (b), it is depicted an eyeglass aid, in which all the components were included into the device.

The continuous reduction in the size of the components allowed that they could soon all be moved behind the ear, either as part of the eyeglass bow, as a self-contained curved package that attached to a sawn-off standard eyeglass bow, or finally as the so-called behind-the-ear (BTE) hearing aid. With further decreases in the size of components, in-the-ear (ITE) devices started to appear in the mid and late 1950s.

However, it was not until the the introduction in the market of the integrated circuit (IC) in 1964 and a new type of transistor, the FET, when hearing aids became much smaller. By the early 1980s, ITE aids had become small enough for most of components to fit within the ear canal portion of the device, what involved the design of a new hearing aid model, the so-called ITC device. With further improvements in electronics, the entire hearing aid could finally be located inside the ear canal by the early 1990s. The CIC had arrived and at last the hearing aid was practically invisible to the casual observer. Because of their greatest relevance in current hearing aids, these four types of hearing aids, that is, BTE, ITE, ITC and CIC models, will be explained in-depth in Section 2.4.

For many years, conventional analog hearing aids have been the only option available for hearing-impaired people. However, these instruments have very serious associated drawbacks, which, for the sake of clarity, we summarize below:

1. Analog hearing aids provide the same gain regardless the frequency, however, hearing loss may not be the same for the different frequencies. In this respect, a hearing aid capable to provide a variable gain with respect to frequency would be deemed as very appreciated by most of hearing aid users.

2. Since the advantage of programmability is not included in analog circuits, adjusting the hearing aid to compensate the particular hearing loss the user suffers from becomes a challenging goal because it would involve designing a unique circuit for each particular user. This would be excessively costly.

3. The fact of amplifying all input sound signals, regardless the frequency and the level input signal, leads to increase the high-intensity level signals to extremely high values, which results in very unpleasant and amplified sounds for the user. This illustrates the necessity for hearing aids can also adjust its gain according to the level of the input sound signal.

2.3.5 The digital era

The introduction of the digital signal processor (DSP) in the market, along with the important abovementioned drawbacks associated to analog hearing aids, led to the development of digital hearing aids, which usually offered their users greater flexibility to compensate for their hearing loss, and, in many situations, provided a more natural sound quality than conventional analog hearing aids.

Figure 2.11 will assist us in introducing the structure of a typical digital hearing aid. As clearly depicted, it is composed of:

- A microphone to convert the input sound into an electrical signal.

- A analog-to-digital converter (ADC), to transform the analog electrical signal into a digital signal.

- A digital signal processor (DSP), the very core of a digital hearing aid, that intends for processing the digital signal aiming at compensating the particular hearing loss the user suffers from.

- A digital-to-analog converter (DAC) to back the amplified digital signal into an analog signal.
- A tiny loudspeaker to convert the analog signal back to sound.
- A small battery to supply electric power to the electronic devices that compose the instrument.

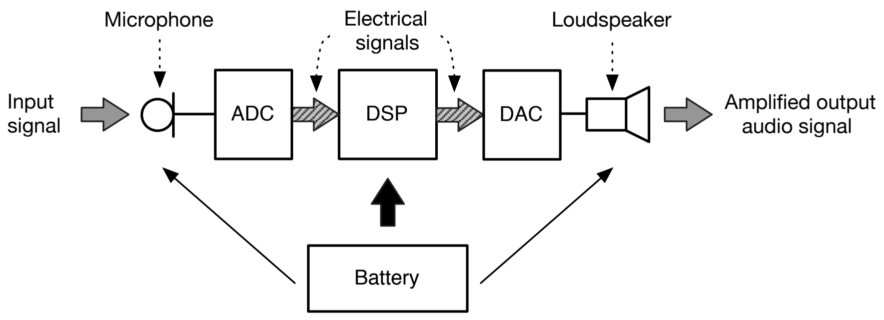


Figure 2.11: A simplified diagram of the structure of a typical digital hearing aid. As shown, it is composed of a microphone, an analog-to-digital converter (ADC), a digital signal processor (DSP) that it is the very core of a digital hearing aid, a digital-to-analog converter (DAC), a loudspeaker and a small battery.

Digital hearing aids exhibit two key advantages when compared to analog hearing aids. The first one is that the input sound signal is "separated" into a number of frequency bands aiming at some frequency bands, such as, for instance, the high-frequency bands or the bands that contain the speech information being more amplified than other bands, like, for example, low-frequency bands or bands that contain a high level of noise. It is interesting to note that this technique solves one of the problems associated with analog hearing aids: providing the same gain value regardless the frequency. The second benefit is based on the dynamic range compression that plays the key role of estimating a desirable gain to map the wide range of an input sound signal (e.g. speech) into the reduced dynamic range of a hearing impaired listener. Basically, this strategy is an automatic gain control, in which the gain is automatically adjusted based on the intensity level of the input signal. In this scenario, frames with a high intensity level (loud sounds) are amplified less compared to frames with a low intensity level (soft sounds), since a comfortable listening level for loud sounds makes soft sounds inaudible. In this respect, we are made sure that loud sounds are not becoming uncomfortably loud because, apart from improving speech intelligibility, this strategy is also designed to avoid discomfort, distortion, and damage. With this in mind, the gain function in each frequency band may be based on a curvilinear or piecewise linear function, such as, for instance, the 3-piece linear approximation illustrated in Figure 2.12 [Hersh and Johnson, 2003]. In this example at hand, it seems clear to notice that low input level sound signals are expanded and high input level signals are compressed in the impaired dynamic range. The two advantages combined together is the so-called "multichannel compression", or also known "multiband compression", that intends for applying a different separate compression function in each frequency band, since the subject's dynamic range often differs in different frequency bands.

Additionally, the most sophisticated digital hearing aids may also provide some interesting

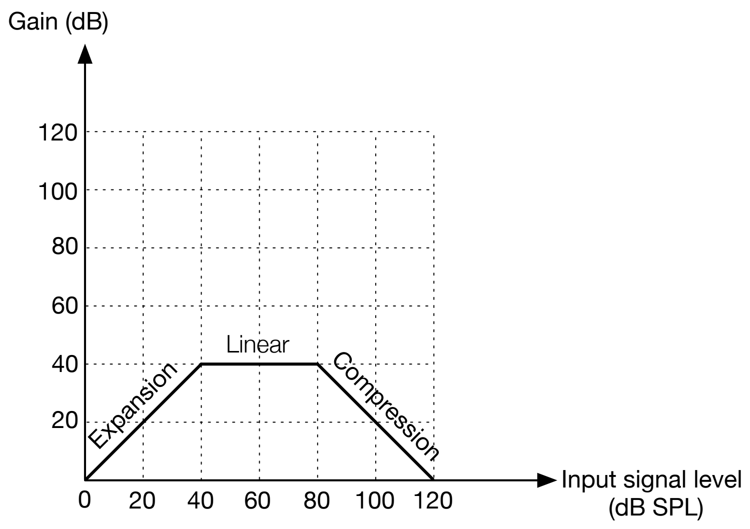


Figure 2.12: An illustrative representation of a typical gain function, for each frequency band, in digital hearing aids. It is based on a 3-piece linear approximation in which low input level sound signals are expanded ("expansion" area), whereas high input level signals are compressed in the impaired dynamic range ("compression" area).

advantages that for making this chapter self-contained, we summarize in the following paragraphs.

- **Acoustic feedback cancellation**
  As clearly illustrated in Figure 2.13, acoustic feedback appears when part of the conveniently amplified output signal produced by the hearing aid returns through the external auditory canal, and enters again this device, being thus anew amplified [Dillon, 2001; Hellgren et al., 1999; Kates, 2008; Spriet, 2004; Spriet et al., 2007, 2008]. Sometimes, feedback may cause the hearing aid to become unstable, producing very unpleasant and irritating howls. Preventing the hearing aid from such instability enforces designers to limit the maximum gain that can be used to compensate the user's acoustic loss.

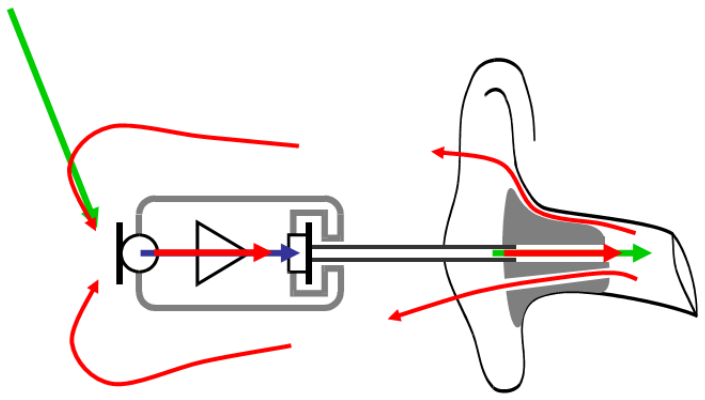


Figure 2.13: Acoustic feedback in hearing aids appears when part of the conveniently amplified output signal produced by the device returns through the external auditory canal, and enters again this instrument, being thus anew amplified.

  In the effort of canceling acoustic feedback in digital hearing aids, there are two widely-used approaches that we list in the following paragraphs.

  1. To selectively attenuate the frequency components for which feedback happens. This approach is effective in avoiding feedback, but the drawback here consists in the fact that it is equivalent to a narrowband hearing aid gain reduction.
  2. To model the feedback path with an internal filter in parallel to the feedback path, subtracting thus the feedback signal. It is interesting to note that, by making use of this approach, the hearing aid gain is not affected.

- **Environmental classification**
  As advanced in the introductory chapter, hearing aid users face a variety of different acoustic environments in their daily life, and, although they expect authentic, significant benefit in each of these situations, this is not the case. The reason is as follows. Analog hearing aids are designed and programmed for only one listening environment what, in turns, means that the gain to apply is the same regardless the acoustic situation the user is in.

  In this respect, advanced digital hearing aids have a variety (although reduced) of amplification schemes or programs adapted to different listening conditions what, in turns, means that the amplification program to be applied when the user is embedded in an acoustical environment consisting, for instance, in a quiet situation (i.e., a conference) will be different than that to be used when the user is embedded in a noisy place, such as, for instance, a traffic jam.

- **Directional microphones**

  As stated beforehand, one of the main problems for hearing-impaired people is the reduction of speech intelligibility when they are immersed in noisy environments, which is mainly caused by the loss of temporal and spectral resolution in the auditory processing of the impaired ear.

  In the effort of improving speech intelligibility in many noisy environments, directional microphones are used in both digital hearing aids [Hamacher et al., 2005], which basically allows to locate the sound source of interest in the horizontal plane, by means of *spatial cues* estimated from both signals entering the two ears. This is referred to as "binaural hearing aids".

  The basic idea underlying to the use of binaural hearing aids is to locate and enhance the sound source of interest (the speech signal for most of hearing aid users), and attenuate other sound sources, such as, for instance, noises or interferences. This leads to improve speech intelligibility in noisy environments when compared to conventional, analog hearing aids, as succinctly illustrated in Figure 2.14.

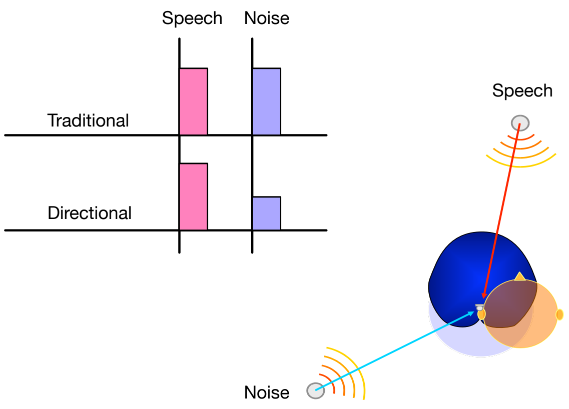


Figure 2.14: Picture depicting the purpose of binaural hearing aids. They allow to locate the sound source of interest by employing both signals entering the two ears. The basic idea underlying this system is to focus on the sound source of interest (the speech signal for most of hearing aid users), and attenuate other sound sources, such as, for instance, noises or interferences, coming from other undesired directions.

  Note that this approach requires the necessity to wear two hearing aids. Many hearing-impaired people, specially elderly people, have acoustic loss in both ears, and consequently they need two devices. Often, when hearing aids are worn at both ears, these devices usually operate independently. However, in binaural hearing aids, both devices should work *collaboratively* at two ears ("ipsilateral" and "contralateral" ear). This obviously requires a communication link between both hearing devices. The simplest solution would be to connect them by using a wire. However, most users do not like this approach because of the non-aesthetic aspect of the wire linking both hearing aids from one ear to the other. This enforces to use a *wireless* link between both devices, what unavoidably increases the power consumption and, consequently, reduces the battery life, one of the most important limiting factors for implementing complex signal processing in digital hearing aids. Roughly speaking, the current technology demands as much power to communicate both hearing aids as that required for the signal processing on a mono-aural device [Kates, 2008]. Thus, only most sophisticated, high-cost digital hearing aids make use of directional microphones in order to improve speech intelligibility in noisy environments.

Having a look at the abovementioned advantages, it seems to be clear that digital hearing aids would improve significantly the quality of life of many hearing-impaired people. Nevertheless, currently, high costs of these advanced devices force many hearing-impaired people to choose cheaper, analog hearing aids instead, or medium-cost digital hearing aids, which regrettably do not perform the aforementioned functionalities.

Regrettably, an extremely point to note here is that most of digital hearing aids in the market suffer from important constraints in terms of computational capability, memory and battery, that make difficult the implementation of an automatic sound classification system in the DSP hearing aids are based on. For the sake of clarity and for making this thesis self-contained, we have summarized in Appendix A the restrictions imposed by the DSP used by our platform to carry out the experiments.

2.4 Hearing aids models

Although all current hearing aids essentially consist of the same basic components, they may come in different models, which basically differ on their size and their location on the hearing aid user's ear. Roughly speaking, the degree and type of acoustic loss the user suffers from determine the model and the way of placement the device on the ear.

Obviously, due to the differences in physical appearance of these four models of hearing aids, they have rather different properties regarding the maximum gain, dynamic range, sound quality, minimum feedback channel attenuation and so on. These current hearing aid models are briefly described in the following paragraphs:

- **Behind-the-ear** (**BTE**):
  This type of hearing aid has its central processing unit placed behind the pinna of the user. The microphone and control buttons are placed on this unit and the loudspeaker is connected to the external auditory canal by means of a sound tube.

  Among all hearing aid models currently available in the market, BTE devices fit the widest range of hearing loss, from mild to profound hearing loss, and generally offer the greatest possibilities for hearing aid adjustment.

- **In-the-ear** (**ITE**):
  This model of tailored hearing aid is much smaller than BTE hearing aid and is placed inside the deepest impression of the pinna, the concha. Note that we say tailored in the sense that the earmold is fitted to each particular hearing aid user's ear.

  Among fitted hearing aids models, this style lends itself best to venting, although the benefit is minimized if large vents are used, which leads to increase the acoustic feedback.

- **In-the-canal** (**ITC**):
  ITC hearing aid is inserted into the external auditory canal of the user. Due to its small physical size, it is more discrete than ITE hearing aid.

  Even though it has less gain and output than ITE device, the microphone position and slightly deeper insertion of the device provide a smaller residual volume, what, in turns, means that it requires approximately 5 dB less gain to fit a comparable acoustic loss [Sandlin, 1990].

ITC models are highly recommended for those people who suffer from mild to moderate acoustic loss.

- **Completely-in-the-canal** (**CIC**):
  This device is, among the hearing aid models explained in this section, the most inconspicuous one since it is inserted deep into the external auditory canal of the user.

  The main advantages of CIC devices are summarized below:

  – To reduce occlusion effect.
  – To reduce feedback.
  – To improve sound localization.
  – To require less gain than other hearing aid models, because the distance between the end of the hearing aid and the eardrum is minimal.
  – To eliminate wind noise.
  – To provide greater high-frequency gain.

For the sake of clarity, we summarize in Table 2.2 the most typical features of the abovementioned hearing aid models (in particular, BTE, ITC and CIC models) [Hersh and Johnson, 2003]. Being more precise, we illustrate the frequency range (Hz), the maximum gain (dB), and the maximum output (dB), the battery life and the feature telecoil. Note that telecoil in the table simply refers a feature available on many current hearing aids. Basically, a telecoil is a tiny coil of wire around a core that will induce an electric current in the coil when it is in the presence of a changing magnetic field. The basic idea underlying the fact of using a telecoil, within a hearing aid, is that the telecoil is used as the input sound source instead of (or in addition to) the microphone and thus the hearing aid user can hear a magnetic signal which represents sound.

Hearing aid model	**Frequency range (Hz)**	**Maximum gain (dB)**	**Maximum output (dB)**	**Battery life (hours)**	**Telecoil**
CIC	4 to 5 octaves	45 ± 10	100	100	No
ITC	starting in the	45 ± 10	100	100	Few
BTE	region 100 to 500	65 ± 10	140	$200 - 500$	Most

Table 2.2: Summary of most typical features for different current hearing aid models [Hersh and Johnson, 2003].

Finally, for properly completing this section, these hearing aids models are depicted in Figure 2.15. In this figure, from top to bottom row the types are: BTE, ITE, ITC and CIC models. Please note that the pictures are courtesy of Widex, Sweden.

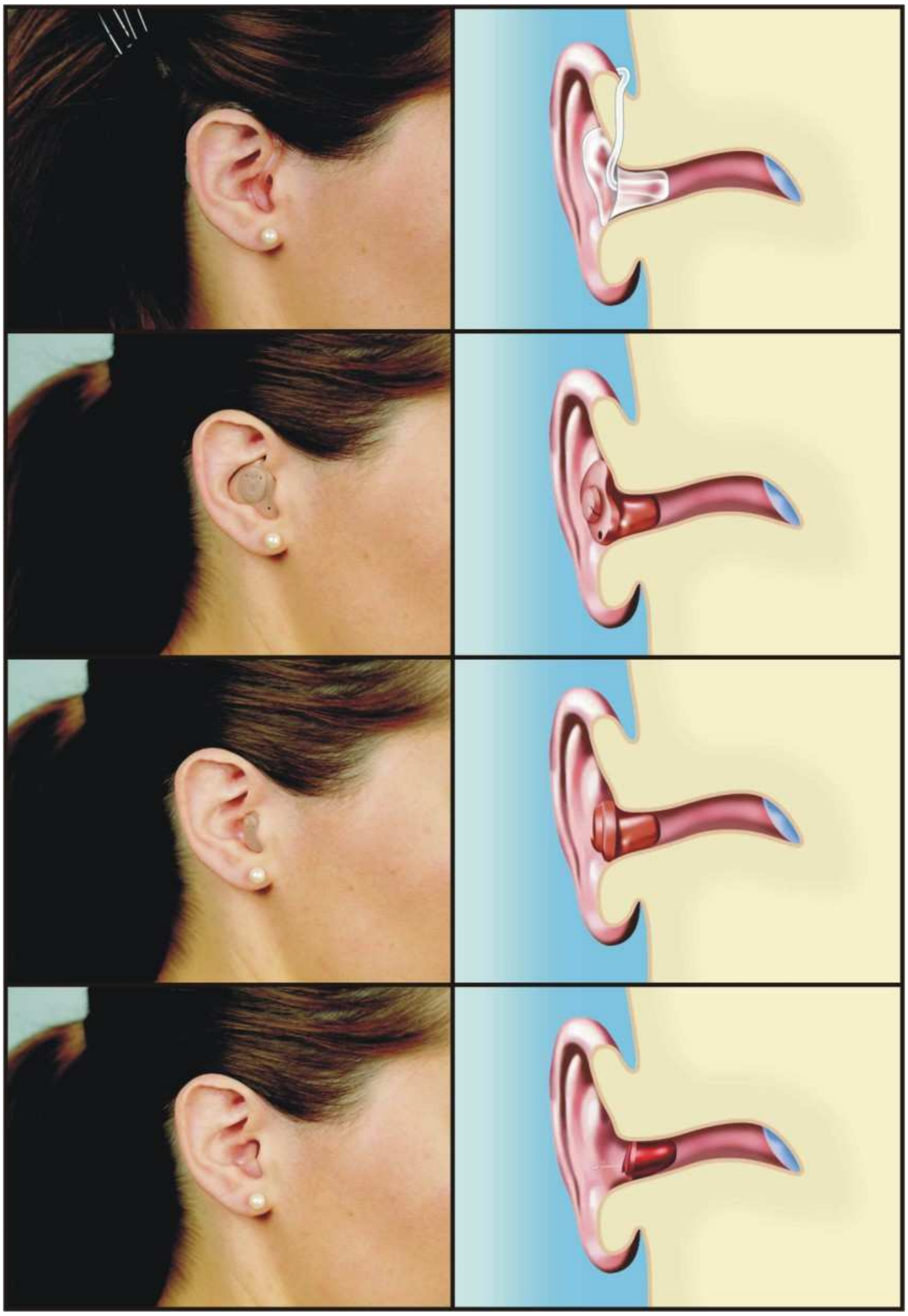


Figure 2.15: Picture illustrating the four most common hearing aid models available in the market currently. From top to bottom row these models are: BTE, ITE, ITC and CIC models. The pictures are courtesy of Widex, Sweden.

Chapter 3

Overview of automatic sound classification systems

3.1 Introduction

As mentioned in the introductory chapter, the objective of this thesis is to design and implement a prototype for a digital hearing aid that can be automatically fitted according to the preferences of its user for various listening conditions. Just in this respect, this chapter describes in a detailed way the basic blocks composing any automatic sound classification system.

Figure 3.1 depicts the structure of an automatic sound classification system. The basic idea underlying to this system is that the classifier categorizes the input sound signal into one of the output audio classes considered, by means of a set of sound signal-describing features extracted from the input sound.

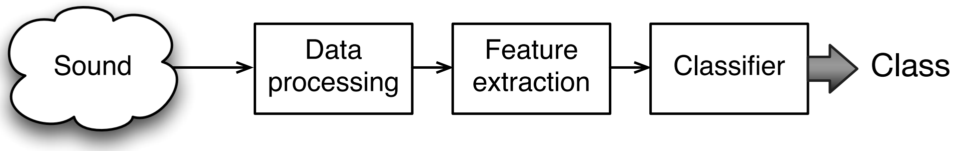


Figure 3.1: Picture illustrating an automatic sound classification system. It classifies the input sound signal into one of the output audio classes considered, by means of a set of sound signal-describing features extracted from the input sound.

Deepening a little more in the particular application at hand, the purpose of implementing an automatic sound classification system in a DSP-based hearing aid is that, according to the classifier decision (that is, the output audio class returned by the classifier), the most appropriate "amplification program" to such class, or within the context of the application at hand, acoustic environment (that is, speech, music or noise), is automatically selected, as clearly depicted in Figure 3.2. To understand the importance of this purpose, it is convenient to imagine the two following situations that we have considered as being representative enough of the importance of implementing this automatic classification system in a hearing aid. Let us suppose that the user is in an acoustical environment consisting in a speech-in-quiet situation, such as, for instance, a conference, in which the user aims to clearly understand what the lecturer is saying. In this scenario, the hearing aid will decide that it is worth amplifying the signal (the hearing aid will recognize the acoustic environment as "speech" environment and will automatically select the "speech amplification program") and thus, the user will not lose the information contained in the speech fragment. Let us now imagine that the user is embedded in a noisy place, such as, for instance, a traffic jam. In this situation, the hearing will decide that it is not worth amplifying such a signal, and thus will reduce the gain, which will also result in saving battery life, the most important constraint for designing advanced algorithms in hearing aids. With these two situations in mind, it seems to be clear that the implementation of this automatic sound classification system is important, since not only does it allow to improve speech intelligibility, but also the sound quality perceived by its user.

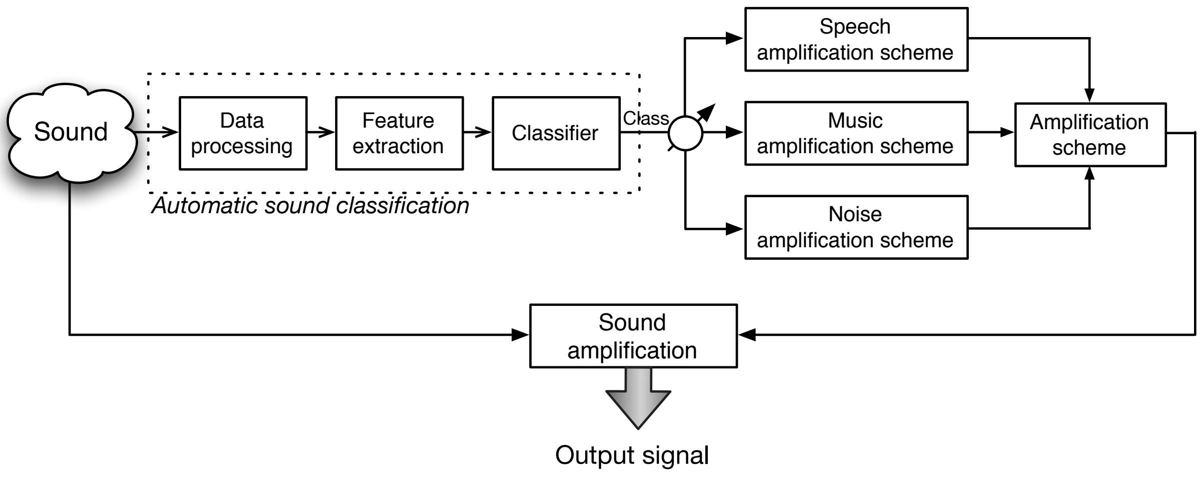

Sound
Data processing
Feature extraction
Classifier
Class
Automatic sound classification
Speech amplification scheme
Music amplification scheme
Noise amplification scheme
Amplification scheme
Sound amplification
Output signal

Figure 3.2: An illustrative representation of the way an automatic sound classification system, implemented in a DSP-based hearing aid, works. According to the classifier decision (that is, the output audio class returned by the classifier), the most appropriate "amplification program" to such class, or within the context of the application at hand, acoustic environment (that is, speech, music or noise), is automatically selected.

It is important to highlight that sound classification *does not* modify the hearing aid output directly, but it rather provides a control signal to select the best amplification program for each of the acoustic environments considered within the context of each classification task.

From a practical implementation point of view, Figure 3.3 will assist us in showing the two main functional blocks that should be implemented in the DSP our hearing aid is based on. The first one, labelled "Functional block A", would correspond to the set of signal processing stages aiming to compensate the *particular* hearing loss the user suffers from. The second one, labelled "Functional block B", would be related to the set of signal processing steps aiming at classifying the input sound signal and, by means of a control signal, selecting the most appropriate amplification program for user's comfort, whose overall implementation is the main focus of this thesis. As previously mentioned, this classifying system consists conceptually of three basic parts:

- A pre-processing functional block that aims at carrying out a sequence of processes to be applied to all the frames in which the input signal has been segmented.
- A feature extraction process that calculates a number of features arranged in the feature vector.

- A three-classes classifier that categorizes input sound signals into three classes (speech, music and noise).

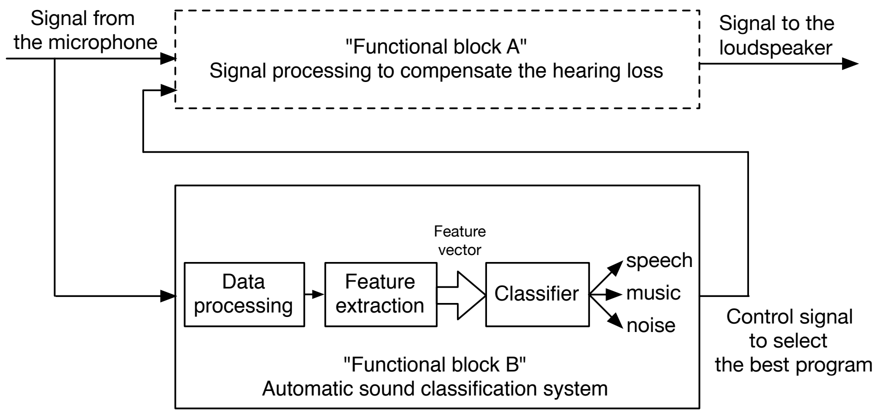

Signal from the microphone
"Functional block A"
Signal processing to compensate the hearing loss
Signal to the loudspeaker
Feature vector
Data processing
Feature extraction
Classifier
speech
music
noise
Control signal to select the best program
"Functional block B"
Automatic sound classification system

Figure 3.3: Conceptual representation of the two main functional blocks to be implemented in the DSP our hearing aid is based on. The block labelled “Functional block A” groups all the signal processing operations leading to compensate the *particular* hearing loss the user suffers from. The other block, labelled “Functional block B”, is related to the set of signal processing steps aiming at classifying the sound and, by means of a control signal, selecting the most appropriate amplification program for user’s comfort.

The key points underlying these three blocks will be explained in a very detailed way in the sections that follow.

With these concepts in mind, this chapter is structured in four sections. Section 3.2 describes the purpose of the data processing block, with a detailed description of the steps commonly carried out in this block when classifying environmental sounds in digital hearing aids. Section 3.3 shows, in an illustrative way, the basic idea underlying to the crucial issue of selecting features when discriminating among different acoustic environments. As will be better understood later on, the classifier performance is severely dependent on the features selected in the sense that, the most appropriate features for a particular classification task, like, for example, distinguishing between music and speech, may not be the most well-suited for another different one, such as, for instance, separating speech from noise. Finally, Section 3.4 depicts the purpose of the classifier *itself*, putting special emphasis on both the kind of classifiers proposed in the literature and the way these classifiers are trained, or in other words, the way these classifiers are able to learn from an appropriate set of training patterns, and properly classify other patterns that have never been found before. This section also explains in-depth the study-case classifiers: the mean square error (MSE) linear classifier (Subsection 3.4.2), the k-NN algorithm (Subsection 3.4.3) and two particular kinds of neural networks: multilayer perceptrons (MLPs) (Subsection 3.4.4.2) and radial basis function (RBF) networks (Subsection 3.4.4.3).

3.2 Data processing

In the effort of classifying the input sound signal, it is necessary to carry out a number of signal processing steps that, for the sake of clarity, we summarize in the following paragraphs. Prior to describing this sequence of processes, it is worth pointing out that the input audio signal to be classified, labelled $X(t)$ within our framework, is the "digitized" signal obtained after the analog-to-digital conversion performed by the ADC integrated in the DSP.

With this in mind, the signal $X(t)$, which will be assumed as a *stochastic process*, is segmented into consecutive frames, as clearly shown below:

$$X_i(t_k) \qquad i = 1, \ldots, N_{\text{frames}} \tag{3.1}$$

where $i = 1, \ldots, N_{\text{frames}}$ is the index over the N_{frames} composing a time slot and the index k simply labels the one over the p samples of each frame. Please note that the classifier will return a decision (speech, music and noise, in our case) for each time slot. For illustrative purposes, and facilitating the reader's comprehension, Figure 3.4 aims at depicting in a very illustrative way our framework.

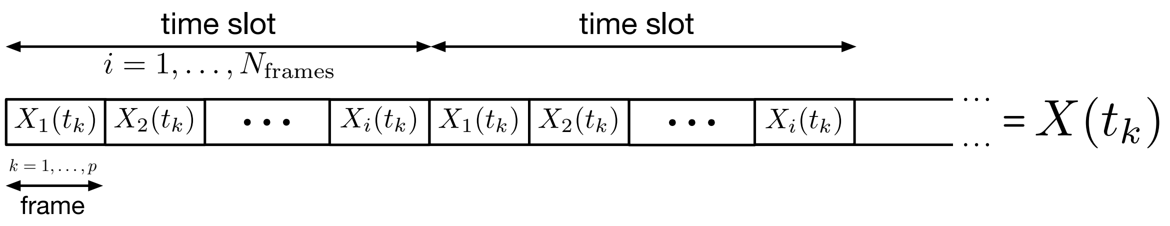


Figure 3.4: The "digitized" input sound signal, labelled $X(t)$, is segmented into frames, composing N_{frames} consecutive frames a time slot. The classifier will return a decision (speech, music or noise, in our case) for each time slot. In this picture, k labels the index over the p samples of each frame.

Since each frame $X_i(t_k)$ ($X_i(t)$, henceforth) is a windowed stochastic process, any of its samples, $X_i(t_k)$, is thus a random variable that, for the sake of simplicity, we label it X_{ik}.

Thus, for each audio frame, $X_i(t)$, the following random vector is obtained:

$$\mathbf{X_i} = [X_{i1}, \ldots, X_{ip}]\,. \tag{3.2}$$

With this initial data, and bearing in mind the DSP used by our platform to carry out the experiments (see Appendix A for further details), the WOLA filterbank coprocessor computes the Discrete Fourier Transform (DFT) of this initial data, leading to the matrix that we label $\overline{\chi_i} = [\chi_i(1), \chi_i(2), \ldots, \chi_i(N_{\text{B}})]$, with N_{B} the number of frequency bands available in the DSP. This is just the initial information the system uses to calculate the complete set of the sound signal-features describing any frame $X_i(t)$.

3.3 Feature extraction

3.3.1 Introduction

A reliable and robust feature extraction process is an important step in any automatic sound classification system. As sketched in Figure 3.5, this stage plays the key role of processing the input audio signal, after carrying out the sequence of operations described in the data processing block, aiming at extracting some kind of relevant, essential information that characterizes it. Thus, each input audio signal is represented by a number of sound signal-describing features that, for computational reasons, are grouped forming a vector, commonly called *feature vector*,

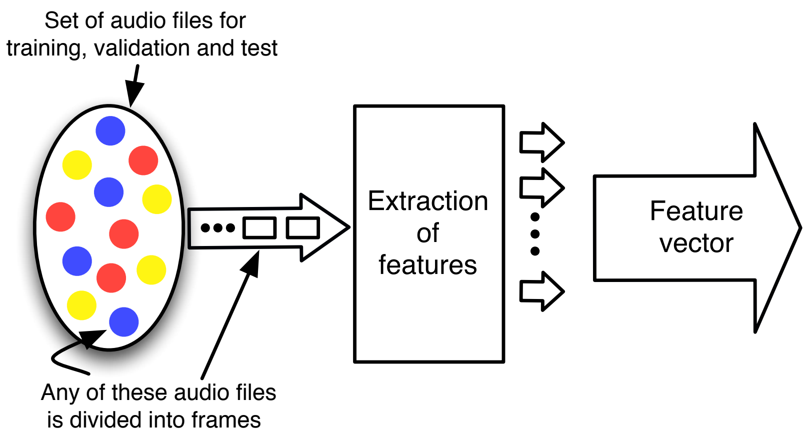

Set of audio files for training, validation and test
Extraction of features
Feature vector
Any of these audio files is divided into frames

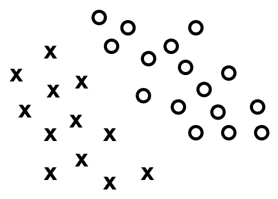
Figure 3.5: Picture illustrating the way the feature extraction process works. Basically, after a sequence of mathematical operations performed in the data processing stage, the feature extraction block generates a number of sound signal-describing features that, for computational reasons, are grouped forming a vector, commonly called *feature vector*, labelled **F** within our framework.

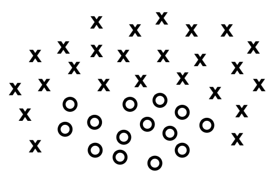
labelled **F** within our framework. Note that this feature vector should be as much discriminative as possible among the audio classes to distinguish (speech, music and noise, in our case).

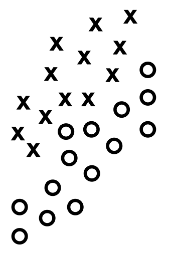
In general, we can say that a "proper" feature is the one that contains the kind of valuable audio signal information that allows the classifier to properly distinguish among the classes considered. If this is not the case, some features may degrade the classifier performance because of two fundamental reasons: 1) if the features do not contain all the essential, adequate information, the classifier may not perform properly, and 2) if the features contain excessive, irrelevant information, the system will also fail; in this sense, it is sometimes said that such unrelated features behave as noise [Duda et al., 2001].

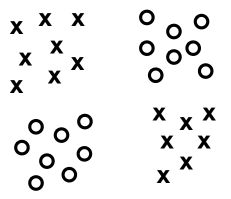
In the effort of better understanding why some features may improve or degrade the classifier performance, we have included Figure 3.6, which basically assists us in showing different groups of features for the same two-classes classification task. As clearly illustrated, the linear separable features, shown in Figure 3.6(a), are the most desirable from the practical point of view of the classifier, because using a simple linear discriminant is good enough to separate both classes. The nonlinear separable, correlated and multimodal features, illustrated, respectively, in Figures 3.6(b), 3.6(c) and 3.6(d), are generally manageable, but features with poor separability, depicted in Figure 3.6(e), are practically impossible to deal with from the classifier point of view.

Many signal features have been proposed in the literature for classifying environmental sounds in hearing aids [Büchler et al., 2005]. In particular, there is a number of available, general-purpose spectral features that could potentially exhibit different behavior for speech, music and noise, and, thus may assist the classifier in properly discriminating the input signal into one of the mentioned classes. Some experiments [Alexandre et al., 2008a; Carey et al., 1999; Li et al., 2001; Lu et al., 2002; Tzanetakis and Cook, 2002] suggest that the features that we describe in the subsection that immediately follows provide a high discriminating capability for the problem of speech, music and noise classification in hearing aids.


(a) Features with linear separability.


(b) Features with nonlinear separability.


(c) Highly correlated features.


(d) Multimodal features.

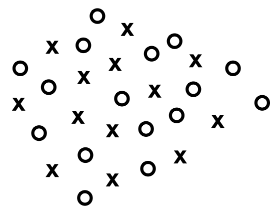

(e) Features with poor separability.

Figure 3.6: In the effort of better understanding why some features may improve or degrade the classifier performance, this picture aims at showing different groups of features for the same two-classes classification task. The linear separable features, shown in 3.6(a), are the most desirable from the classifier point of view. The nonlinear separable, correlated and multimodal features, illustrated, respectively, in 3.6(b), 3.6(c) and 3.6(d), are generally manageable, but features with poor separability, depicted in 3.6(e), are practically impossible to deal with from the classifier point of view.

3.3.2 Classical spectral features

As previously mentioned, there is a number of interesting spectral features that exhibit different behavior for speech, music and noise, and thus, may assist the classifier in appropriately classifying the input sound signal. Note that we say spectral features in the sense that they are basically computed from the spectrum magnitude corresponding to each sound frame, which is deemed as very beneficial from our practical implementation point of view, since the WOLA coprocessor provides at its output the spectrum magnitude of such sound frame. As stated beforehand, it provides the k-th frequency bin of the spectrum of any sound frame $X_i(t)$. For the sake of clarity, we labelled this matrix $\overline{\chi_i} = [\chi_i(1), \chi_i(2), \ldots, \chi_i(N_\mathrm{B})]$, where $k = 1, \ldots, N_\mathrm{B}$ is the index over the N_B frequency bands available in the DSP ($N_\mathrm{B} = 64$, in our case).

With this in mind, these spectral features for each analysis sound frame $X_i(t)$, which take as initial information the aforementioned matrix $\overline{\chi_i}$, have been found to be as follows.

- **Spectral centroid**
  As defined in [Scheirer and Slaney, 1997], the spectral centroid of a frame $X_i(t)$ is a measure of the center of gravity of the spectral power distribution, and thus, it outlines if the spectrum contains a majority of high or low frequencies. From a mathematical point of view, the spectral centroid of a frame $X_i(t)$, that we label it SC_i, can be calculated by

means of the expression shown below:

$$\mathrm{SC}_i = \frac{\sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)| \cdot k}{\sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|}, \tag{3.3}$$

being N_B the number of frequency bands available in the DSP used to carry out the experiments.

This feature has been widely-used for sound classification in hearing aids. At this respect, it is worth mentioning that not only has it been used to discriminate among speech, music and noise [Li et al., 2001], but also to classify music into different genres [Tzanetakis and Cook, 2002].

- **Voice2white**
This parameter, proposed in [Guaus and Batlle, 2004], is a measure of the energy inside the typical speech band (300-3600 Hz) with respect to the whole energy of frame $X_i(t)$. We label this feature V2W$_i$ and, from a mathematical point of view, it can be computed by means of the following expression:

$$\mathrm{V2W}_i = \frac{\sum_{k=M_1}^{M_2} |\chi_i(k)|}{\sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|} \tag{3.4}$$

with M_1 and M_2 being the indexes that limit the speech band (300-3600 Hz), that is, in terms of number of frequency band, $M_1 = 2$ and $M_2 = 32$.

This feature has been widely used in our work to discriminate between speech and non-speech signals [Alexandre et al., 2006b] and also to distinguish among speech, music and noise [Alexandre et al., 2008d]. Note that we say non-speech signals in the sense that they contain any other signal different from speech, including not only environmental noise but also music, both instrumental and vocal.

- **Spectral flux**
The spectral flux of a frame $X_i(t)$, labelled SF$_i$, is defined as the average variation value of spectrum magnitude between two adjacent frames [Lu et al., 2002], as clearly defined as follows:

$$\mathrm{SF}_i = \sum_{k=1}^{N_\mathrm{B}} (|\chi_i(k)| - |\chi_{i-1}(k)|)^2. \tag{3.5}$$

With this in mind, it is very clear to note that this feature is closely related to the amount of spectral local changes. The practical point to note regarding this feature is that SF values for speech sounds are generally higher and vary rapidly, whereas SF values for music vary slower and SF values for noisy signals barely vary. This is the reason why this feature has been used in the literature for distinguishing between speech and music [Lu et al., 2002; Tzanetakis and Cook, 2002] or even for discriminating between music and environmental sounds.

- **Spectral roll-off**
  This parameter, that we label it SR_i, is defined as the frequency bin of the spectrum below which a PR % of the magnitude distribution of the spectrum is concentrated [Tzanetakis and Cook, 2002]. Its mathematical definition is depicted in the following expression:

$$\sum_{k=1}^{R_\mathrm{t}} |\chi_i(k)| = \mathrm{PR} \cdot \sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)| \tag{3.6}$$

  where a typical value for PR is 85 %. This feature "gives an idea" of the shape of the spectrum, or in other words, it shows how high in the spectrum a certain part of the energy lies.

  This feature is very adequate for properly classifying between speech and music, since speech signals tend to have lower spectral roll-off values than music signals. In general, music signals contain higher frequencies from instruments such as, for instance, flutes, distorted guitars or hi-hats, and thus, these sound signals tend to have higher values of this feature. It is worth mentioning that SR feature has been already used to discriminate among speech, music and noise [Li et al., 2001].

- **Short time energy**
  Although short time energy is a "simple" feature, it is, however, widely-used in sound classification systems [Li et al., 2001; Lu et al., 2002]. We label it STE_i and, mathematically, it is basically defined as the mean energy of the signal within each analysis frame $X_i(t)$:

$$\mathrm{STE}_i = \frac{1}{N_\mathrm{B}} \sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2. \tag{3.7}$$

  In general, speech signals consist of words mixed with silence what involves the variation of STE values for this kind of signals is generally higher than that obtained for music signals. This is the main reason why this feature is commonly used for discriminating between speech and music.

- **Spectral flatness measure**
  Spectral flatness measure (SFM) is a parameter that describes the flatness properties of the spectrum magnitude of a sound frame. From a mathematical point of view, the SFM value for the frame $X_i(t)$, that we label it SFM_i, can be computed as the ratio of the geometric mean to the arithmetic mean of the spectrum magnitude [Izmirli, 1999], as stated in the following expression:

$$\mathrm{SFM}_i = \frac{\left[\prod_{k=1}^{N_\mathrm{B}} |\chi_i(k)|\right]^{1/N_\mathrm{B}}}{\frac{1}{N_\mathrm{B}} \cdot \sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|}. \tag{3.8}$$

  Sometimes, this feature is also called "tonality coefficient", and is widely-used to quantify *how much* tonal a sound signal is. Note that we say "tonal" in the sense that a high value of SFM indicates that the spectrum magnitude of the sound signal has a comparable amount of power in all frequency bands, what, in turns, means that the signal is "similar" to a noisy signal, whereas a low value of SFM indicates that, the spectral power is concentrated in a relatively small number of frequency bands, what involves the presence of tonal components in the sound signal, or equivalently, the signal can be considered as a speech sound.

It is interesting to note that this measure is one of the many features used in the MPEG-7 standard, in which it is named "Audio Spectral Flatness" [Martínez, 2004].

- **Spectral crest factor**
  Similar to the SFM feature, the spectral crest factor (SCF) is also a measure for quantifying the tonality of a sound signal. Instead of computing the geometric mean of the spectrum magnitude for the numerator, the maximum of the spectral energy is calculated, and consequently, this feature is obtained by calculating the ratio between the maximum and the mean value of the spectral energy [Hosseinzadeh and Krishnan, 2008]. Put it in a more mathematical way, it can be expressed as follows:

$$\mathrm{SCF}_i = \frac{\max(|\chi_i(k)|^2)}{\frac{1}{N_\mathrm{B}} \sum_{k=1}^{N_\mathrm{B}} (|\chi_i(k)|^2)}. \tag{3.9}$$

  In general, SCF values for musical instruments are higher than those obtained for speech signals.

- **Spectral bandwidth**
  Spectral bandwidth (SB) is a parameter closely related to the spectral centroid. It can be computed by using the following expression [Andersson, 2004]:

$$\mathrm{SB}_i^2 = \frac{\sum_{k=1}^{N_\mathrm{B}} (k - \mathrm{SC}_i)^2 \cdot |\chi_i(k)|^2}{\sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2}, \tag{3.10}$$

  where SC_i labels the spectral centroid at frame $X_i(t)$. Please note that prior to computing this feature it is necessary to calculate the value of the spectral centroid for each analysis frame.

  Spectral bandwidth is a measure that indicates if the spectral power is concentrated around the centroid or if it is dispersed over the spectrum. Music generally consists of a broad mixture of frequencies, whereas speech consists of a limited range of frequencies. This makes spectral bandwidth an appropriate feature for discriminating between speech and music. This feature can be also appropriate for distinguishing among different musical genres since, for instance, rock music has higher spectral power spread than flute instruments.

- **Renyi entropy**
  The Renyi entropy of a sound frame $X_i(t)$, labelled RE_i, is a measure of its spectral distribution. From a mathematical point of view, it can be computed by using the following expression [Hosseinzadeh and Krishnan, 2008]:

$$\mathrm{RE}_i = \frac{1}{1-\alpha} \cdot \log_2 \left(\sum_{k=1}^{N_\mathrm{B}} \left| \frac{\chi_i(k)}{\sum_{k=1}^{N_\mathrm{B}} \chi_i(k)} \right|^\alpha \right). \tag{3.11}$$

Here, the normalized energy of the band can be seen as a probability distribution for calculating entropy. Experimentally, α has been found to be $\alpha = 3$ [Aviyente and Williams, 2001; Flandrin et al., 1994].

This feature is very adequate for discriminating between speech and noise, since it can *detect* the degree of randomness in the signal in the sense that a "structured" sound corresponds to a sound signal and has a lower entropy than that compared to a "non-structured" sound, which corresponds to noise and has a larger entropy value.

- **Shannon entropy**
  In the same line of reasoning as that in the previous feature, this parameter that we label it SE_i, is a measure that quantifies the randomness of each analysis frame $X_i(t)$. From a mathematical point of view, it is computed by means of the expression shown below [Ramalingam and Krishnan, 2006]:

$$\text{SE}_i = \sum_{k=1}^{N_\text{B}} |\chi_i(k)| \cdot \log_2 |\chi_i(k)|. \tag{3.12}$$

  This feature is deemed useful for distinguishing between speech and noise.

- **Mel-frequency cepstral coefficients**
  Psychophysical studies have shown that human perception of the frequency contents of sounds for speech signals does not follow a linear scale [Moore, 1997]. Just in this respect, the basic idea underlying to Mel-frequency cepstral coefficients (MFCCs) is to "imitate" the behavior of the human ear. In this sense, for each tone with an actual frequency f, measured in Hz, a subjective pitch is measured on the "Mel-frequency scale", having the Mel-frequency scale a linear frequency spacing below 1 kHz and a logarithmic spacing above 1 kHz, as clearly shown in Figure 3.7.

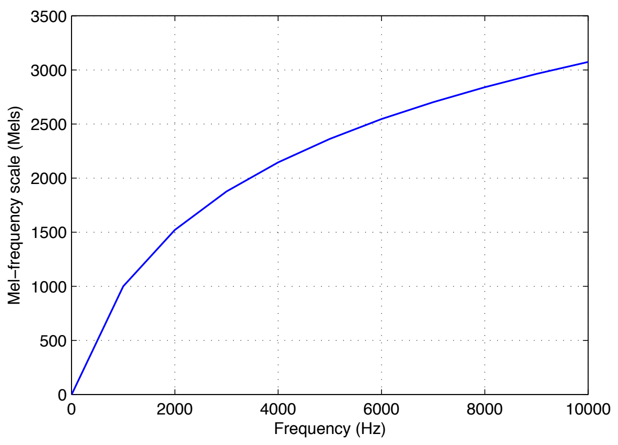


Figure 3.7: Picture illustrating the Mel-frequency scale. As shown, it is a linear frequency spacing below 1 kHz and a logarithmic spacing above 1 kHz.

As a reference point, the pitch of a 1 kHz frequency tone, 40 dB above the perceptual hearing threshold, is defined as 1000 Mels. Therefore, we can use the following expression to approximate the Mels for a particular frequency f in Hz,

$$\text{Mel}(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right). \tag{3.13}$$

In the effort of simulating the subjective spectrum, we have made use of a filter bank, spaced uniformly on the Mel scale, as depicted in Figure 3.8. This filter bank is applied to the spectrum of the speech signal to obtain a Mel-spectrum. The Mel-spectrum is log compressed and, when transformed back to time domain by means of DCT, provides us the so-called MFCCs. Therefore, MFCCs, that we label them c_n (n denoting the index over the n_c MFCCs), can be calculated by means of:

$$c_n = \sum_{d=1}^{D} (\log S_d) \cdot cos\left[n\left(d - \frac{1}{2}\right)\frac{\pi}{D}\right] \qquad n = 1, 2, \ldots, n_c \tag{3.14}$$

being S_d the Mel-power spectrum coefficients (where d is the index over the D Mel-spaced filters, that is, $d = 1, 2, \ldots, D$).

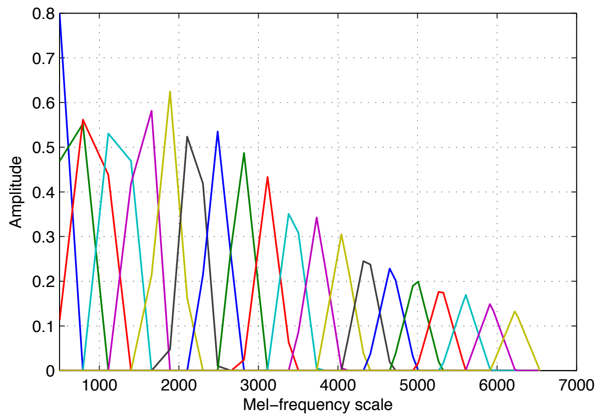


Figure 3.8: An illustrative representation of Mel-frequency filter bank with $D = 20$ filters. In this picture, the vertical axis represents the weights of the filter coefficients.

We represent MFCCs, calculated by means of the Expression 3.14, as an MFCC column feature vector, or in other words,

$$\mathbf{c} = [c_1\ c_2\ c_3\ \ldots\ c_{n_c}]^T. \tag{3.15}$$

Aiming at providing a good representation of the local spectral properties of the signal, we have computed 20 MFCCs, although it has been demonstrated that for classification tasks, it is good enough to take into account only the first 5 coefficients [Tzanetakis and Cook, 2002].

- **Delta Mel-frequency cepstral coefficients**
  These features are basically motivated by the non-stationarity of speech signals, since it has been observed that the short-time spectrum of speech signals varies much more rapidly than the short-time spectrum of noisy signals [Carey et al., 1999; Chou and Gu, 2001; Takeuchi et al., 2001].

The great difference between the rate of change of the spectrum magnitude for speech and noisy signals is probably one of the many cues that human ear can use to ignore the relatively stationary noisy signals and focus on the rapidly-changing spectrum of speech signals [Kumar et al., 2011].

Mathematically, for an analysis frame $X_i(t)$, delta mel-frequency cepstral coefficients, that we label ΔMFCCs, are calculated by means of the expression stated below:

$$\Delta\text{MFCC}_i(k) = \text{MFCC}_i(k-2) - \text{MFCC}_i(k+2), \tag{3.16}$$

where $k = 1, \ldots, N_\text{B}$ is the index over the N_B frequency bands available in the DSP.

In the same line of reasoning as that in MFCCs, we have computed 20 ΔMFCCs.

An important point to note regarding Expression 3.16 is that since there *does not* exist neither MFCC(−1) nor MFCC(0) for a particular sound frame, ΔMFCC(1) is simply equal to MFCC(3) and ΔMFCC(2) is equal to MFCC(4). In the same line of reasoning, since there *does not* exit neither MFCC(21) nor MFCC(22), ΔMFCC(19) is simply equal to MFCC(17) and ΔMFCC(20) is equal to MFCC(18).

Finally, for completing this section, it is worth mentioning that we label this set of classical, spectral features $\mathcal{S}_\mathcal{C}$, as clearly shown below:

$$\mathcal{S}_\text{C} = \{\text{SC}, \text{V2W}, \text{SF}, \text{STE}, 20\text{MFCCs}, 20\Delta\text{MFCCs}, \text{SR}, \text{SB}, \text{SCF}, \text{SFM}, \text{RE}, \text{SE}\},$$

subscript C meaning "classical".

3.4 Classifying algorithm

3.4.1 Introduction

After the feature extraction process, a decision on the class to which the input sound signal belongs must be made, based on the features extracted from the sound signal. This process is performed by the classifying algorithm. The aforementioned feature vector, that is, **F**, forms the input to the classifier, and the output of the classifier is the assignment of the input sound signal to one of the audio output classes considered (speech, music and noise, in our case), as clearly shown in Figure 3.2.

Roughly speaking, classifiers can be classified into *discriminative* and *generative* [Rubinstein and Hastie, 1997]. The first ones determine the boundaries between classes without modeling the probabilities directly. In order to better understand what this assertion means, it is worth having a look at Figure 3.9, which represents a two-dimensional *feature space*. In this picture, for every point in the *feature space*, a corresponding class is defined by mapping the *feature space* to the *decision space*. The borders between classes (the dashed lines in the *feature space*) are found by performing some kind of "training" process, which is accomplished by an adequate sound database. Once the borders are fixed by means of a set of design patterns, the performance of the classifier is tested with the patterns available in a *test* phase, which are completely different from those patterns available in the *design* phase. This type of classification algorithms include, among other classifiers, linear discriminants, Artificial Neural Networks or Support Vector Machines.

On the one hand, training this kind of algorithms can be a difficult task because it involves considering all audio classes at the same time and often demands the use of iterative algorithms which are not guaranteed to converge to a good result. On the other hand, discriminative

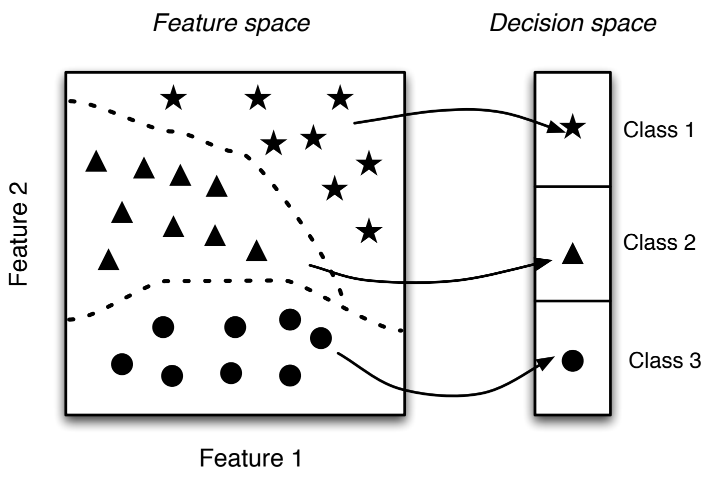


Figure 3.9: Representation of a two-dimensional *feature space*. In this figure, for every point in the *feature space*, a corresponding class is defined by mapping the *feature space* to the *decision space*. The borders between classes (the dashed lines in the *feature space*) are found by performing some kind of "training" process, which is accomplished by an adequate sound database.

classifiers can be very adequate when there is enough sufficiently varied design patterns to train the classifier, and can work even in those situations where the underlying probability distribution for the feature set is unknown *a priori*.

The second type of classifiers, that is, generative classifiers, model the probability density function for the feature set arising from each audio class. In this case, classification is performed by determining the probability of each class produced by the feature sets, and selecting the class that has the highest probability. This type of classification algorithms include, among other classifiers, Hidden Markov Models or Bayes classifiers. The key point to note regarding this type of classifiers is that they are relatively easy to train because the probability density function for the feature set can be determined separately for each class. The serious associated drawback is, however, that these classifiers demand to know the probability distributions for the feature set *a priory*. In our particular case at hand, since we *do not* know this initial information, the different experiments carried out in this thesis have been done using discriminative classifiers.

Bearing in mind that we have made use of discriminative classifiers, perhaps the reader may wonder what *training the classifier* exactly means. Aiming at answering this question demands to mention that training the classifier basically refers to the process in which, by learning from appropriate patterns available in a *design* phase, the parameters of the classifier are adjusted to exhibit the *best* performance, or equivalently, within the application at hand, to properly classify other patterns (sound signals, in our case) that have never been found before. In other words, the purpose of the training process is to give the most accurate performance in the real world, where patterns not included in the *design* set can be expected to happen. The ability to deal with these "unfound" patterns is called "generalization", and the goal when training the classifier is thus to produce accurate generalized results.

Roughly speaking, there are two types of algorithms to train classifiers: *supervised* and *unsupervised* algorithms. In the first ones, the classifier is presented with a set of "training vectors" (feature vectors, in our case) and its associated class (that is, the desired class), and, a classifier output that differs from the desired one generates an error that is used to modify the classifier parameters. In the second ones, the classifier organizes the training vectors into groups that occur "naturally", without reference to the assigned classes. The classifier looks for training vectors that tend to form related clusters. It can be specified the number of clusters at the beginning of the unsupervised learning, but the labels are not known *a priory* (that is, we do not know the desired classes). Classificatory experiments that rely on unsupervised learning are often used for exploratory data analysis, in which the main question is: *how many classes are there*?, rather than *what is the best way to assign any input sound to the classes that have already been identified*? Intuitively, it seems to be clear that, in the batches of classification experiments carried out in this thesis, the classifiers have been trained in a supervised way, because we are interested in properly assign to an input sound to the correct output, rather than in knowing how many classes there are.

An extremely important point to note here is that the training process of the classifier is always performed *off-line* on a desktop computer, and *never* on the hearing aid itself. Only once the classifier has been properly trained, validated and tested in the laboratory, this classifier will be uploaded onto the hearing aid.

With these concepts in mind, we describe in the following subsections the classifiers used in this thesis to carry out the experiments.

3.4.2 Mean square error (MSE) linear classifier

One of the classifiers explored in this thesis is the mean square error (MSE) linear classifier because of its simplicity and good results [Alexandre et al., 2008d].

Although the Bayes classifier that minimizes the mean square error, shown in Expression 3.17, is *optimum*,

$$\widehat{Y}_{opt} = \int_{-\infty}^{+\infty} Y \cdot f(Y|\mathbf{F}) \cdot dY = E[Y|\mathbf{F}] \tag{3.17}$$

the problem is that it requires to know *a priory* the probability density function $f(Y|\mathbf{F})$. As stated beforehand, this is a difficult task in the problem at hand. A feasible, although *sub-optimal* solution to overcome this drawback is to use the MSE linear classifier [Srinath et al., 1995]. In this approach, the classifier is forced to be a linear combination of the available data, whose weighting coefficients will be those that minimize its mean square error computed over the patterns available in a *design* phase.

Being more explicit, if Y is the *unknown* class of a given audio file, let say, for instance, the j-th file, and whose class we would like to recognize, the output class determined by the MSE linear classifier, $\widehat{Y}_j$, can be computed by means of,

$$\widehat{Y}_j = w_0 + \sum_{n=1}^{L} w_n \cdot F_{n,j} \tag{3.18}$$

where:

- w_n labels any of the *weights* in such a linear combination,
- $F_{n,j}$ represents the value of th n-th element in the classifying feature vector $\mathbf{F}$ applied to the j-th sound file, and finally,
- L is the number of features, that is, $dim(\mathbf{F}) = \mathbf{L}$.

Note in Expression 3.18 that the sound $X(t)$ is *not* actually the signal that directly enters the classifier: the signal that *do* enter the classifier is the feature vector, labelled $\mathbf{F}$, that contains the kind of *adequate* information that assists it in accurately classifying the sound signal $X(t)$.

Taking into account the nature of the patterns available in the *design* phase, it is convenient to describe this by using matrix notation. In this regard, if N_{P} represents the number of input sound patterns available in such a *design* phase, the matrix of input design patterns can be described as follows:

$$\mathbf{Q} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ F_{11} & F_{12} & F_{13} & \cdots & F_{1\mathrm{N_P}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ F_{\mathrm{L}1} & F_{\mathrm{L}2} & F_{\mathrm{L}3} & \cdots & F_{\mathrm{LN_P}} \end{pmatrix} \tag{3.19}$$

L being, as mentioned before, the total number of features used to characterize the patterns, and N_{P} the number of input patterns available in the *design* phase.

Note in Expression 3.19 that the matrix elements in the first row are equal to *unity* in order to implement the independent terms of the linear combinations. With this in mind, the corresponding *matrix of weights* can be defined as shown below:

$$\mathbf{W} = \begin{pmatrix} w_{01} & w_{11} & w_{21} & \cdots & w_{\mathrm{L}1} \\ w_{02} & w_{12} & w_{22} & \cdots & w_{\mathrm{L}2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{0\mathrm{C}} & w_{1\mathrm{C}} & w_{2\mathrm{C}} & \cdots & w_{\mathrm{LC}} \end{pmatrix} \tag{3.20}$$

where C represents the number of classes to classify ($C = 3$, in our case).

Thus, the output of the linear combinations for the input patterns can be expressed as stated below:

$$\mathbf{Y} = \mathbf{W} \cdot \mathbf{Q} \tag{3.21}$$

Y being a matrix with C rows and N_{P} columns.

In the supervised learning approach, if $\mathbf{T}$ represents the *target matrix* containing the class of each input pattern, that is, the desired class, the error matrix is thus defined as follows:

$$\mathbf{Y} - \mathbf{T} = [e_o \quad e_1 \quad \ldots \quad e_{N_{\mathrm{P}}}] \,.$$

If we define the mean square error (MSE) being

$$\mathrm{MSE} = \frac{1}{N_{\mathrm{P}}} \sum_{n=1}^{N_{\mathrm{P}}} e_n^2, \tag{3.22}$$

the coefficients w_n in Expression 3.18 are just those that *minimize* the MSE computed over the patterns available in the design phase:

$$\frac{\partial \mathrm{MSE}}{\partial w_n} = 0, \qquad n = 1, \ldots, L \tag{3.23}$$

what leads to the usually named "Wiener-Hopf equations" [Asmussen, 1998].

Thus, using Expression 3.23, the values of the *matrix of weights* can be determined as follows:

$$\mathbf{W} = \mathbf{T} \cdot \mathbf{Q}^{\mathbf{T}} \cdot \left(\mathbf{Q} \cdot \mathbf{Q}^{\mathbf{T}}\right). \tag{3.24}$$

3.4.3 k-nearest neighbor (k-NN) classifier

The k-nearest neighbor classifier, commonly known as the "k-NN classifier", is a distance-based classifier [Duda and Hart, 1973].

Put it very simple, to obtain the class corresponding to a new vector $\mathbf{p}$, the algorithm has simply to look for the k nearest vectors, called "neighbors" within this algorithm, to the mentioned vector $\mathbf{p}$, and weigh, usually using a majority rule, their class numbers they belong to. Please note that, within the application at hand, $\mathbf{p}$ is simply the classifying feature vector that we have labelled $\mathbf{F}$.

For the sake of clarity, and in the aim of expressing this idea in a more formal way, let us consider a set of feature vectors $\{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_{N_\mathrm{P}}\}$ available in a *design* phase, with $\mathbf{F}_i \in \mathbb{R}^n$ organized into C different classes Y_i and N_P being, as previously mentioned, the number of patterns available in the *design* phase.

Let $\mathbb{R}^n(\mathbf{F}) = \mathbf{F}' : ||\mathbf{F} - \mathbf{F}'|| \leqslant r^2$ be a volume centered in the vector $\mathbf{F}$ that encompasses k patterns of the N_P available patterns in the design phase. With this in mind, the k-nearest neighbor classification rule is defined as:

$$q(\mathbf{F}) = arg_{max} v(\mathbf{F}, y) \tag{3.25}$$

where $v(\mathbf{F}, y)$ is the number of feature vectors $\mathbf{F}_i$ with hidden state $Y_i = Y$, which lie in the volume $\mathbf{F}_i \in \mathbb{R}^n(\mathbf{F})$.

Although it is possible to use different distance metrics [Cost and Salzberg, 1993; Everitt, 1974; Fukunaga and Beauregard, 1984; Salton and McGill, 1983; Wilson and Martinez, 1997], most implementations employ euclidean distance, and consequently, the volumes are hyperspheres around the vector $\mathbf{F}$.

For illustrative purposes, we will make use of Figure 3.10, which will assist us in better understanding the way this classifying algorithm works. In the particular two-dimensional space represented in this figure, the green circle represents a pattern to be classified into either "class 1" (blue plus sign-class $\rightarrow +$) or "class 2" (red cross-class $\rightarrow$ x). It is important to mention that this pattern to be classified *does not* belong to the set of patterns available in the *design* phase. With the k-NN scheme in mind, in order to assign the class, the algorithm looks for the k nearest neighbors to that pattern, and, by using the majority rule, weighs their class number they belong to. In this respect, it is worth noting that the performance of this classifier is severely dependent on the number of k patterns considered. In order to better understand what this assertion means, it is convenient to imagine the two following situations, which have been considered to be representative enough: if k is set to be 3, the pattern is assigned to "class 2" because there are 2 patterns belonging to "class 2" (red cross-class $\rightarrow$ x) and only 1 pattern belonging to "class 1" (blue plus sign-class $\rightarrow +$) inside the inner circle. On the contrary, if k is set to be 5, the pattern is assigned to "class 1" because there are 3 patterns belonging to "class 1" and only 2 patterns belonging to "class 2", inside the dashed outer circle.

Thus, the question arising here is *how to determine the most appropriate value of* k? Intuitively, the value of k in the k-NN classifier is an user-specific parameter. In many articles, it is automatically selected in order to minimize the error probability over the patterns available in a *validation* phase. In this thesis, different k-NN classifiers with values of k ranging from 1

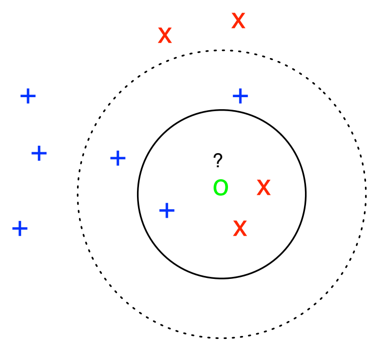


Figure 3.10: Two-dimensional picture depicting, in a very illustrative way, the way the k-NN classifier works. The green circle represents a pattern to be classified into either "class 1" (blue plus sign-class → +) or "class 2" (red cross-class → x). If k is chosen to be 3, the pattern is then assigned to the "class 2" because there are 2 patterns belonging to the "class 2" and only 1 pattern belonging to the "class 1", inside the inner circle. On the contrary, if k is chosen to be 5, the pattern is assigned to the "class 1" because there are 3 patterns belonging to the "class 1" and only 2 patterns belonging to the "class 2", inside the dashed outer circle.

to 20 have been implemented, and the value of k that achieves the best percentage of correct classification computed over the patterns available in the *validation* phase has been ultimately selected.

3.4.4 Artificial neural networks

3.4.4.1 Introduction

The human brain is the most complex, nonlinear, and parallel *information processing system* produced by the nature. We can say that it is a living proof that not only is the *fault tolerant parallel processing* physically possible but also fast and robust. It has the ability of organizing its main constituents, called *neurons*, in order that certain computations are carried out many times faster than the *fastest* digital computer. It has a specific structure and the capacity for constructing its own rules through experience. This experience is constructed over the years, with the most dramatic development of the human brain in the first years, producing millions of synapses[1] per second.

Artificial neural networks, commonly called simply "neural networks" (NNs), emerge as an approach to model the way in which brain performs some particular tasks, which can be usually difficult to carry out using traditional programming methods, such as, for instance, classification tasks using pattern recognition, prediction, etc.

Although will be explained in a very clear way later on, we can formally define in advance *artificial neural networks* as follows [Hetch-Nielsen, 1990]:

An artificial neural network is a parallel, distributed information processing structure consisting of processing units (which can possess a local memory and can carry out localized information processing operations) interconnected via unidirectional signal channels called connections. Each processing unit has a single output connection that branches into as many collateral connections as desired; each one carries the same signal - ***the processing unit output signal****. The processing unit output signal can be of any mathematical type desired. The information processing that*

[1]A synapse is a connection between two neurons through which "information" (or being more precise, nerve impulses) flows from one neuron to another.

goes on within each processing unit can be defined arbitrarily with the restriction that it must be completely local; that is, it must depend only on the current values of the input signals arriving at the processing element via impinging connections and on values stored in the processing unit's local memory.

It is worth mentioning that, we have explored in this thesis the feasibility of two kinds of tailored neural networks for driving the particular sort of sound classification in digital hearing aids: multilayer perceptrons (MLPs) and radial basis function (RBF) networks, which will be explained in-depth in Subsections 3.4.4.2 and 3.4.4.3, respectively.

3.4.4.2 Multilayer perceptrons (MLPs)

Architecture

The basic architecture of a multilayer perceptron consists of three layers of neurons (input, hidden and output layers) in which each neuron in the hidden and output layer is interconnected with all the neurons in the previous layer by links with *adjustable weights* [Bishop, 1995; Duda et al., 2001], as clearly represented in Figure 3.11. This type of neural networks is commonly known as "feedforward neural networks" and is probably the most popular and widely-used network in many practical implementations. It is worth mentioning that multilayer perceptrons may consist of more than one hidden layer, but it has been shown that a single hidden layer is sufficient enough to approximate any function to arbitrary accuracy, given a sufficient and finite number of neurons [Cybenko, 1989].

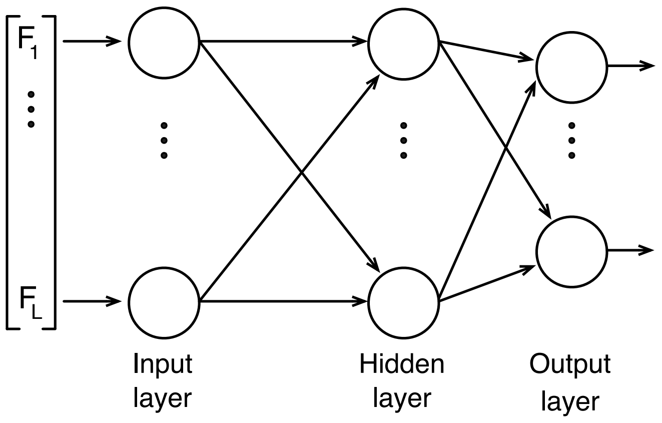


Figure 3.11: Multilayer perceptron (MLP) consisting of three layers of neurons: input, hidden and output layers. Each of the hidden and outputs neurons is interconnected with all the neurons in the previous layer by links with *adjustable weights*. In this picture, L represents the number of features selected for feeding the classifying algorithm, or in other words, the dimension of the feature vector that we have labelled $\mathbf{F}$.

Having a look at Figure 3.11, one can notice that each neuron in the input layer holds a value, holding thus the input layer the *input vector*, or being more precise, within the particular application at hand, the *feature vector* (labelled $\mathbf{F}$), being thus $dim(\mathbf{F}) = L$. As mentioned before, each of these neurons in the input layer is connected to every neuron in the next layer, called the *hidden layer*, and consequently, each connection has an associated weight that, for the sake of clarity, we have labelled w_{lm}, where $l = 1, \ldots, L$ is the index over the L input neurons, while $m = 1, \ldots, M$ labels the one over the M hidden neurons.

In the same line of reasoning, each neuron in the hidden layer is connected to every neuron in the output layer and each connection has an associated weight that we have labelled w_{mc} where, intuitively, $m = 1, \ldots, M$ is the index over the M hidden neurons and $c = 1, \ldots, C$ labels the one over the C output neurons. Note that, within the application at hand, C represents the number of output audio classes to distinguish by the classifier, that is, $C = 3$.

Deepening a little more in the way a *particular* neuron in the network works, Figure 3.12 illustrates the m-th hidden neuron in a multilayer perceptron.

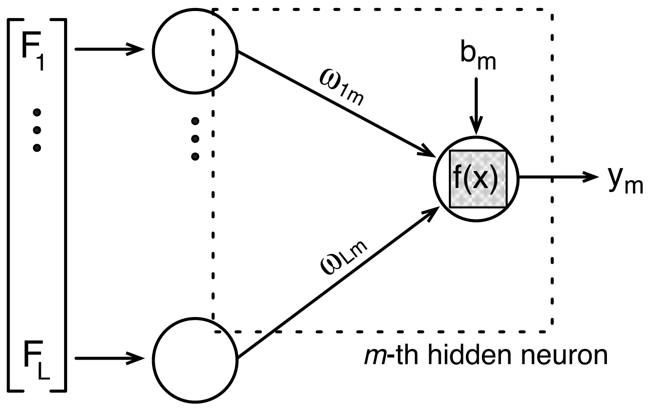


Figure 3.12: Schematic representation of the m-th hidden neuron in a multilayer perceptron. In this picture, the vector $\mathbf{F} = [F_1, F_2, \ldots, F_\mathrm{L}]$ labels the L-dimensional feature vector that feeds the classifier, the weight vector $\mathbf{w} = [w_{1\mathrm{m}}, \ldots, w_{\mathrm{L}m}]$ represents the weights associated with the connections between the input neurons and the study-case m-th hidden neuron, b_m is the bias that can be interpreted as a constant value, and finally, $f(x)$ labels the transfer function used for the m-th hidden neuron.

Put it in a more mathematical way, the output neuron value in this picture, that is, y_m can be computed by means of the expression that follows:

$$y_m = f\left(\sum_{l=1}^{L} F_l \cdot w_{lm} + b_m\right) \tag{3.26}$$

where:

- $\mathbf{F} = [F_1, F_2, \ldots, F_\mathrm{L}]$ labels the L-dimensional feature vector that feeds the classifier,
- w_lm labels the weight associated with the connection between the l-th input neuron and the study-case m-th hidden neuron,
- b_m represents the *bias node*, which can be interpreted as a constant value, and finally,
- $f(\cdot)$ is the transfer function used for the study-case m-th hidden neuron.

Perhaps the reader may wonder what the "transfer function" exactly means. Put it very simple, the transfer function in a neuron defines the output of that neuron as a function of the input value to it. The mathematical expression and the range of output values of the most

Function	**Mathematical expression**	**Range**
Identity	$y = x$	$(-\infty, +\infty)$
Step	$y = sign(x)$	$[-1, 1]$
	$y = u(x)$	$[0, 1]$
Piecewise linear	$y = \begin{cases} -1, & x \leqslant -l \\ x, & -l < x < l \\ 1, & x \geqslant l \end{cases}$	$[-1, 1]$
Logarithmic sigmoid	$y = \frac{1}{1+e^{-x}}$	$(0, 1)$
Hyperbolic tangent	$y = \frac{e^{2x}-1}{e^{2x}+1}$	$(-1, 1)$
Gaussian	y=Ae^{-Bx^2}	$(0, 1]$

Table 3.1: Mathematical expression and range of output values of the most common transfer functions used for hidden and output neurons in multilayer perceptrons.

common transfer functions used for hidden and output neurons in multilayer perceptrons are represented in Table 3.1. In this thesis, the transfer function that we have considered as the most appropriate one for the hidden and output neurons is the logarithmic sigmoid, or in other words, the so-called "logsig". As shown in the table, the output of this function is limited to the range $(0, 1)$, which is very beneficial from the point of view of interpreting the network outputs as Bayesian posterior probabilities. Please note that the input neurons *do not* perform any operation on the input values, and consequently, we *do not* specify any transfer function for these neurons.

Finally, for completing this description, it is worth having a look at Figure 3.13. This figure represents the *particular* multilayer perceptron considered in this thesis: it consists of three layers of neurons, which aim at classifying the input sound signal into the three classes of interest: speech, music and noise. L represents the number of features selected in a general case, that is, the dimension of the feature vector $\mathbf{F}$. As stated beforehand, this figure also illustrates the transfer functions considered as the most appropriate: the logarithmic sigmoid for hidden and output neurons. In this MLP architecture, the number of input neurons corresponds to that of the features used to characterize the sound, the number of output neurons is related to the three classes we are interested in, and finally, the number of hidden neurons depends on the adjustment of the complexity of the network [Duda et al., 2001]. If too many hidden neurons are used, the capability to generalize will be poor; on the contrary, if too few hidden neurons are considered, the training data cannot be learned satisfactorily. In this respect and as will be shown throughout this chapter, selecting the appropriate number of hidden neurons becomes a fundamental issue when designing multilayer perceptrons.

MLP training

Roughly speaking, "network training" is related to the way in which the weights of each neuron composing the network are adjusted. To explain this, it is convenient to consider an *untrained* network, or in other words, a network in which the weights have been randomly initialized. The feature vectors corresponding to the patterns available in a *design* phase are presented to the input layer of this network, and the output networks are determined at the output layer. The

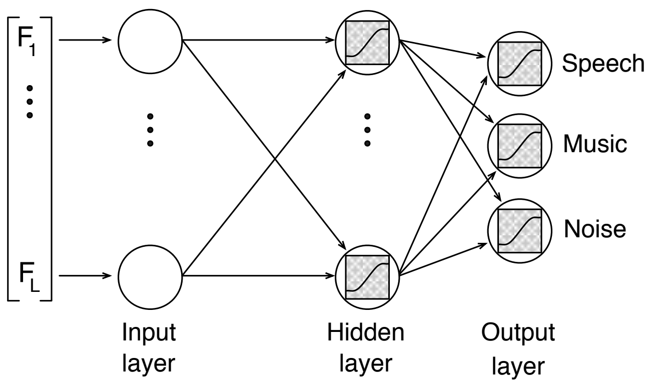


Figure 3.13: An illustrative representation of the *particular* multilayer perceptron (MLP) considered in this thesis to carry out the experiments. It consists of three layers of neurons, which aim at classifying the input sound signal into the three classes of interest: speech, music and noise. L represents the number of features selected for the classifying algorithm, that is, the dimension of the feature vector $\mathbf{F}$. The figure also illustrates the transfer functions considered as the most appropriate ones for the problem at hand: the logarithmic sigmoid, or in other words, the so-called "logsig", for hidden and output neurons.

difference between desired or correct outputs and network outputs is known as "network error". Typically, this error is the so-called mean square error (MSE) between desired and network outputs. With this in mind, the purpose of the training process is just to minimize that error by means of modifying the weights in the network, being thus the weights "adjusted" at the end of this process.

A variety of algorithms has been proposed in the literature aiming at training multilayer perceptrons. They include the gradient descent, Gauss-Newton, Levenberg-Marquardt or Levenberg-Marquardt with the use of Bayesian regularization techniques. The following paragraphs just focus on describing these training techniques.

- **Gradient descent**
  Gradient descent is one of the simplest MLPs training algorithms [Bishop, 1995]. The algorithm usually starts initializing the weight vector, labelled $\mathbf{w}^{(0)}$, with random values. Then, this weight vector is updated iteratively, moving a small amount in the direction of the greatest rate of decrease of the error. $\mathbf{w}^{(i)}$ refers to the weight vector at epoch number i. Denoting the error function for a given weight vector $\mathbf{w}^{(i)}$ by $E(\mathbf{w}^{(i)})$, the weight vector for the epoch $i+1$ is thus given by:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \nabla E(\mathbf{w}^{(i)}) = \mathbf{w}^{(i)} - \eta \mathbf{J}^{(i)}, \tag{3.27}$$

  where $\mathbf{J}^{(i)}$ represents the Jacobian of the error function evaluated in $\mathbf{w}^{(i)}$, given by:

$$\mathbf{J}^{(i)} = \begin{pmatrix} \frac{\partial E(\mathbf{w}^{(i)},\mathbf{x}_1)}{\partial w_1} & \cdots & \frac{\partial E(\mathbf{w}^{(i)},\mathbf{x}_1)}{\partial w_P} \\ \vdots & \ddots & \vdots \\ \frac{\partial E(\mathbf{w}^{(i)},\mathbf{x}_N)}{\partial w_1} & \cdots & \frac{\partial E(\mathbf{w}^{(i)},\mathbf{x}_N)}{\partial w_P} \end{pmatrix}. \tag{3.28}$$

  Note that the gradient is re-evaluated at each step. Turning again our attention to Expression 3.27, the parameter η is the so-called "learning rate", and is of great importance for the convergence of the algorithm. If η is too small, the learning process will be slow, and it may take a long time to train the MLP. On the contrary, if η is too high, the learning process will be much faster, but the algorithm may not converge, and the system could become unstable.

- **Gauss-Newton**
  The Newton method can be seen as an "evolution" of the gradient descent algorithm in the sense that information from the second derivative is considered. The expression for updating the weight vector is given by:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{H}^{(i)^{-1}} \nabla E(\mathbf{w}^{(i)}) \tag{3.29}$$

  where $\mathbf{H}^{(i)}$ represents the Hessian matrix of the error function $E(\mathbf{w})$ evaluated in $\mathbf{w}^{(i)}$, given by:

$$\mathbf{H}^{(i)} = \begin{pmatrix} \frac{\partial^2 E(\mathbf{w}^{(i)},\mathbf{x}_1)}{\partial w_1^2} & \cdots & \frac{\partial^2 E(\mathbf{w}^{(i)},\mathbf{x}_1)}{\partial w_1 \partial w_P} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\mathbf{w}^{(i)},\mathbf{x}_N)}{\partial w_P \partial w_1} & \cdots & \frac{\partial^2 E(\mathbf{w}^{(i)},\mathbf{x}_N)}{\partial w_P^2} \end{pmatrix}. \tag{3.30}$$

  This method converges in only one iteration when the error surface is quadratic, and is in general, much more efficient than the gradient descent method for two main reasons: 1) there is no need to adapt any constants, and 2) because the direction in which the weights vary is more efficient than for the gradient descent.
  The main problem of the Newton method is that it demands to calculate the Hessian matrix and its inverse, which may require large training time. Just in this respect, the Gauss-Newton method is a simplification for the case when the error function is a sum of square errors. In this case, the Hessian matrix can be approximated by a function of the Jacobian ($\mathbf{J}^T\mathbf{J}$), and the gradient of the error function can be approached by a function of the Jacobian and the error function ($\mathbf{J}^T\mathbf{E}$). Thus, according to the Gauss-Newton method, the expression for the calculation of the network weights is given by:

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{i} - \left(\mathbf{J}^{(i)^T}\mathbf{J}^{(i)} + \alpha\mathbf{I}\right)^{-1} \mathbf{J}^{(i)^T}\mathbf{E}^{(i)}, \tag{3.31}$$

  where the parameter α is chosen to ensure that the matrix $\mathbf{J}^{(i)^T}\mathbf{J}^{(i)} + \alpha\mathbf{I}$ is positively defined. That is, the parameter α compensates the most negative eigenvalue in case the matrix is not positively defined. Note that if $\alpha = 0$, Expression 3.31 leads to the one of the Newton method (Expression 3.29), whereas if α is very high the method behaves like the gradient descent with a small learning rate.

- **Levenberg-Marquardt**
  In the Levenberg-Marquardt algorithm [Hagan and Menhaj, 1994], the parameter α varies depending on the value of the error function, being smaller when the error function decreases and higher when it increases.

If the Levenberg-Marquardt algorithm is selected in order to train a multilayer perceptron, the key point of adjusting the size and complexity of the MLP arises: *how to obtain good generalization capability with a "small" MLP size*? This objective basically motivates the use of regularization techniques. Just in this respect, it has been shown that Bayesian regularization techniques exhibit good performance. The next subsection is devoted to clearly explain this issue.

- **Levenberg-Marquardt with Bayesian regularization**
  The technique of Bayesian regularization is based on the minimization of the following error function:

$$E(\mathbf{w}) = \alpha \mathbf{w}^T \mathbf{w} + \beta E_D \tag{3.32}$$

where E_D labels the sum of square errors. If $\alpha \ll \beta$, then the training algorithm will make the errors smaller. If $\alpha \gg \beta$, the training algorithm will emphasize weight size reduction. The Bayesian regularization algorithm aims to find the optimal values of α and β by means of maximizing the joint probability function of α and β assuming a uniform density for them [Foresee and Hagan, 1997]. This leads to the following expressions for α and β:

$$\alpha^{(i+1)} = \frac{P - \frac{2\alpha^{(i)}}{tr\left(\mathbf{H}^{(i+1)}\right)}}{2\mathbf{w}^{(i+1)^T}\mathbf{w}^{(i+1)}} = \frac{\gamma}{2\mathbf{w}^{(i+1)^T}\mathbf{w}^{(i+1)}} \tag{3.33}$$

$$\beta^{(i+1)} = \frac{N_\mathrm{P} - P + \frac{2\alpha^{(i)}}{tr\left(\mathbf{H}^{(i+1)}\right)}}{2E_D^{(i+1)}} = \frac{N_\mathrm{P} - \gamma}{2E_D^{(i+1)}} \tag{3.34}$$

being:

- γ the *effective* number of parameters,
- P the total number of parameters in the network, and finally,
- N_P the total number of available patterns in the design phase.

Note that the main advantage of using regularization techniques is that the generalization capabilities of the classifier are improved and it is possible to obtain better results with smaller networks, since the regularization algorithm *itself* prunes those neurons that are not strictly necessary.

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \left(\mathbf{J}^{(i)^{(T)}}\mathbf{J}^{(i)}\right)^{-1} \mathbf{J}^{(i)^{(T)}}\mathbf{E}^{(i)}. \tag{3.35}$$

For the reasons stated above, all MLPs experiments in this thesis will be done using the Levenberg-Marquardt algorithm with Bayesian regularization techniques [Álvarez et al., 2007].

Backpropagation of error

Common to all abovementioned training algorithms, the computation of the Jacobian or the Hessian matrix is involved, which demands to calculate the derivative of the error function with respect to each weight in the network. This can be simply noted by having a look at Expressions 3.28 and 3.30. Regarding this, it is worth mentioning that the computation of the derivative of the error function with respect to the output neurons weights (that is, weights associated to the connections between hidden and output neurons) is a relatively easy task since the outputs of these neurons depend directly on the weights, but the calculation of the derivative of the error function with respect to the hidden neurons weights (or in other words, weights associated to the connections between input and hidden neurons) is not an easy task because we do not have any desired output values for these hidden neurons, and consequently, we are *not* able to compute the error between the desired and hidden neuron outputs.

The specialized literature contains the backpropagation algorithm aiming at solving this type of problem. This algorithm is quite popular as a learning strategy for multilayer perceptrons because it is conceptually simple, computationally efficient and exhibits good performance [Rumelhart et al., 1986; Werbos, 1974].

The basic purpose of backpropagation algorithm is to minimize iteratively the error function (labeled E) with respect to the network weights, such that, the change in the network weights, labeled $\triangle w$, complies with the following condition at each epoch in the training process:

$$\triangle w \propto -\frac{\partial E}{\partial w}. \tag{3.36}$$

As previously mentioned, the training algorithms require, during the training process, the computation of the partial derivative of error function with respect to each weight in the network. This computation is simple for output neurons weights, but, nevertheless for weights associated to connections between input and hidden neurons is not so simple. Thanks to the backpropagation error approach, instead of computing the partial derivative of error function with respect to each weight in the hidden layer, we reformulate it in the following way by using the chain rule:

$$\frac{\partial E}{\partial w_{lm}} = \frac{\partial E}{\partial f_{\mathrm{TF}_m}} \frac{\partial f_{\mathrm{TF}_m}}{\partial w_{lm}} \tag{3.37}$$

where:

- E is the error function,
- w_{lm} is the weight associated to the connection between l-th input neuron and m-th hidden neuron, and finally,
- f_{TF_m} is the transfer function of m-th neuron in the hidden layer.

With this expression in mind, the derivative of error function with respect to the weight w_{lm} can be expressed as the product of the derivative of error function with respect to the transfer function of the neuron and the derivative of transfer function with respect to the weight. The derivative of error function with respect to the transfer function can be computed taking into account that such activation is contributing to network outputs, and thus, to the error.

3.4.4.3 Radial basis function (RBF) networks

Multilayer perceptrons have been widely-used as nonlinear approaches in the aim of solving certain problems that otherwise would be unfeasible [Agarwal, 1997]. Regrettably, from a practical implementation point of view, MLPs may be a time-consuming task, especially because of their associated training process, which may demand a large processor time. This is basically the reason why there has been a considerable emerging interest in a particular kind of neural network, the so-called radial basis function (RBF) network, in an effort of reaching similar performance, but, with the added advantage of avoiding some time-consuming calculations [Chen et al., 1990].

Roughly speaking, RBF networks can be employed in any sort of model (linear and nonlinear) and any sort of network (single-layer or multi-layer). However, they have been traditionally associated with multilayer networks (in particular, networks consisting of three layers of neurons), as will be shown in the subsection that immediately follows [Broomhead and Lowe, 1998].

RBF network architecture

The basic architecture of a radial basis function network consists of three layers of neurons (input, hidden and output layers), in which each hidden neuron implements a radial basis activation function and, unlike what happens with MLPs, the output layer performs a simple weighted sum with a linear output.

Perhaps the reader may wonder what a radial basis activation function is. Put it very simple, a radial basis function is a multidimensional function that describes the distance (typically, the euclidean distance) between an input vector and a pre-defined center vector. Although there are different types of radial basis functions, the gaussian function is the most commonly used. For the sake of clarity, Figure 3.14 shows the architecture of an RBF network, in which the hidden neurons perform a gaussian function with different variances values.

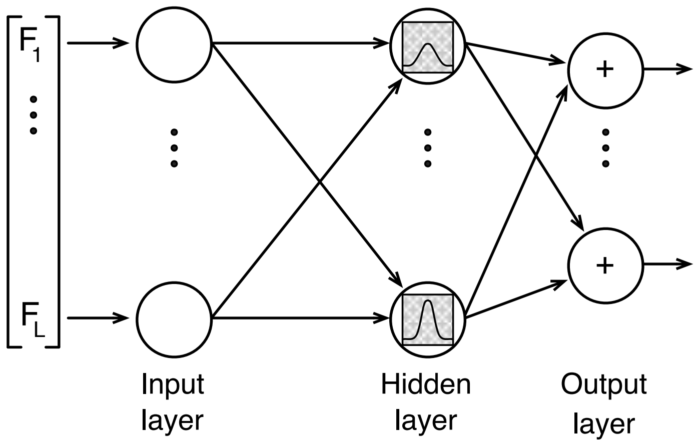


Figure 3.14: Radial basis function (RBF) network consisting of three layers: input, hidden and output layers. Each hidden neuron implements a radial-basis activation function, being the gaussian function the most commonly used. The output layer performs a simple weighted sum with a linear output. In this picture, L represents the number of features selected for the classifying algorithm, that is, the dimension of the feature vector labelled $\mathbf{F}$.

In the effort of expressing these concepts in a more formal way, it is worth noting that the euclidean distance between an input vector, or being more precise, the feature vector $\mathbf{F}$ in our particular problem at hand and a pre-defined center vector (labeled $\mathbf{z}$) can be mathematically written as:

$$I(\mathbf{F}) = |\mathbf{F} - \mathbf{z}|_{\Sigma}^{2} = (\mathbf{F} - \mathbf{z})^T \Sigma^{-1} (\mathbf{F} - \mathbf{z}) \tag{3.38}$$

where:

- $\mathbf{F} = [F_1, F_2, \ldots, F_L]$ labels the L-dimensional feature vector,
- $\mathbf{z} = [z_1, z_2, \ldots, z_L]$ is the L-dimensional pre-defined center vector of the radial-basis function for the hidden neuron, and finally,
- $\mathbf{\Sigma}$ is the covariance matrix, which controls the "smoothness" and the orientation of the radial-basis function for each hidden neuron.

Note that Expression 3.38 is particular for each hidden neuron, or in other words, both the pre-defined center vector $\mathbf{z}$ and the covariance matrix $\mathbf{\Sigma}$ are different for each hidden neuron.

Considering that the radial basis function is the gaussian function, the expression of the radial-basis function can be determined as follows:

$$g(I(\mathbf{F})) = \exp\left(\frac{-I(\mathbf{F})}{2}\right) = \exp\left(-\frac{(\mathbf{F}-\mathbf{z})^T\Sigma^{-1}(\mathbf{F}-\mathbf{z})}{2}\right). \quad (3.39)$$

In the above expression, the argument of the function is a *distance* and is always positive, which leads to evaluate the non-linear function only for positive values of the argument.

For the sake of clarity, Figure 3.15 illustrates, in a mathematical way, an RBF network in which the radial basis activation function implemented in each hidden neuron is a gaussian function.

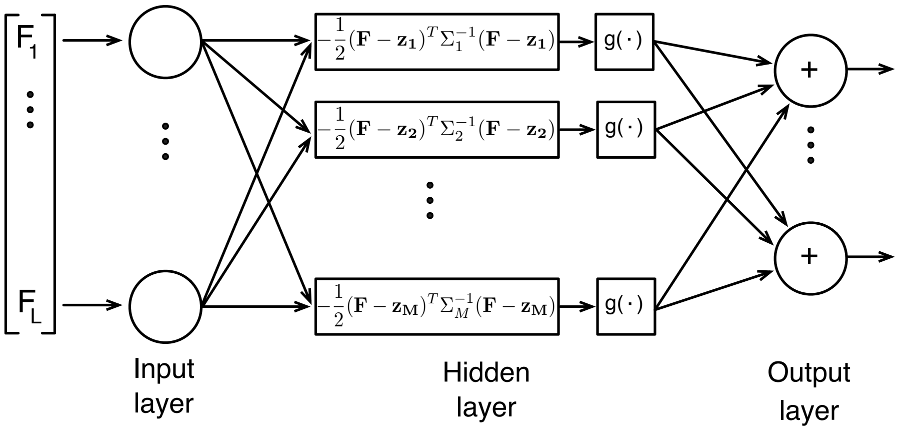

F_1
F_L
$-\frac{1}{2}(\mathbf{F}-\mathbf{z_1})^T\Sigma_1^{-1}(\mathbf{F}-\mathbf{z_1})$
$-\frac{1}{2}(\mathbf{F}-\mathbf{z_2})^T\Sigma_2^{-1}(\mathbf{F}-\mathbf{z_2})$
$-\frac{1}{2}(\mathbf{F}-\mathbf{z_M})^T\Sigma_M^{-1}(\mathbf{F}-\mathbf{z_M})$
g(·)
g(·)
g(·)
+
+
Input layer
Hidden layer
Output layer

Figure 3.15: "Mathematical" representation of a radial basis function (RBF) network in which the radial-basis activation function implemented in each hidden neuron is a gaussian function.

With this in mind, if Y is the *unknown* class of a given audio file, let say, for instance, the j-th file, and whose class we would like to recognize, the output class determined by the c-th output neuron in the RBF network, $\widehat{Y}_c$, can be computed by means of:

$$\widehat{Y}_c = \sum_{m=1}^{M} w_{mc} \cdot g(|\mathbf{F}-\mathbf{z}_m|^2_{\Sigma_m}) + b_c. \quad (3.40)$$

Completing this section demands to mention that the covariance matrix Σ can be set to the identity matrix (or a scalar multiple), to a diagonal matrix with different diagonal elements or to a non-diagonal matrix. In the particular problem at hand, we have set Σ to a diagonal matrix with different diagonal elements h_{lm}, which basically involves that the radial-basis functions are oriented following the axis of coordinates. With this in mind, Expression 3.39, evaluated for diagonal Σ_i matrices, can be rewritten as:

$$g_m = \exp\left(-\frac{1}{2h_{lm}}\sum_{l=1}^{L}(x_l - z_{lm})^2\right). \quad (3.41)$$

RBF network training

Not only does the *training* of RBF networks involve computing the pre-defined center vectors and the covariance matrices but also estimating the weights that connect the hidden and the output layers. For this purpose, nonlinear optimization algorithms, such as, for instance, the gradient descent, are the most appropriate approaches for training RBF networks [Poggio and Girosi, 1990]. However, most nonlinear algorithms suffer from problems of demanding a large processor training time and the possibility of being trapped in local minima. Just in this respect, RBF neural networks are usually trained by using a combination of supervised and non-supervised training techniques, which are basically applied in two steps, as follows:

1. The RBF centers are determined empirically by applying a non-supervised clustering technique, such as, for instance, the k-means clustering algorithm [Moody and Darken, 1989] or vector quantization [Kohonen, 2001], to form templates of the input. In this thesis, we have made use of the k-means clustering algorithm.

2. The elements of the covariance matrix are varied and those that minimize the mean square error computed over the patterns available in a *validation* phase are selected.

3. The weights are computed by minimizing the mean square error between desired and network outputs.

Finally, it is worth mentioning that the above steps describe the way RBF networks have been trained in this thesis.

Chapter 4

Proposed feature extraction and selection algorithms

4.1 Introduction

As mentioned in Chapter 3, within any automatic sound classification system, the feature extraction task plays the key role of processing the input sound signal in order to extract some kind of valuable information that helps the classifying algorithm properly discriminate the input signal among the classes considered (speech, music or noise, in our case). The great difficulty here relies on the fact that due to the *complexity* of human audio perception, extracting representative features is a crucial task because no feature has been designed so far that allows the classifier to discriminate accurately among the different classes. Or in other words, the features best suited for a particular classification task, such as, for instance, discriminating between speech and music may not be the same as those best fitted to a different classification task, like, for example, separating speech from traffic noise.

In the aforementioned chapter, we listed a set of available, general-purpose spectral features that could potentially exhibit different behavior for speech, music and noise, and thus, may assist the classifier in properly discriminating the input sound signal into one of the mentioned classes. Bearing in mind the severe design restrictions imposed by the DSP hearing aids are based on, we have focused in this chapter on a procedure to improve sound classification in hearing aids by designing a set of "low-complexity" features based on a variation of some of the features listed in Chapter 3. The key point that exhibits the greatest relevance here is that these new features can be programmed in the DSP by using a lower number of operations per seconds, or equivalently, the computational cost associated to their implementation is significantly reduced, with the added advantage of not degrading the classification performance. For the sake of clarity, this set of low-complexity features is described in a detailed way in Subsection 4.2.1.

Taking into account that the features listed in Chapter 3 (Subsection 3.3.2) and those included in Section 4.2.1 result in a number of features larger than that we can program in a DSP-based hearing aid, we have placed special emphasis here on designing an feature-selection approach that aims at finding a subset $\mathcal{H} \subseteq \mathcal{F}$ (being $\mathcal{F}$ the set of all features listed in Subsection 3.3.2 and Subsection 4.2.1), consisting of a "reduced" number of satisfactory sound-describing features that, by feeding the classifier, assists it in properly distinguishing among the different acoustic environments considered, while saving DSP resources. In the effort of doing this, we have explored the benefits of using a genetic algorithm with constrained maximum computational cost (or, equivalently, number of instructions per second), which will be described in a detailed way in Section 4.3.

With these ideas in mind, this chapter has been structured in four sections. Section 4.2 mathematically defines the new low-complexity features proposed in this chapter. Section 4.3 shows in a very illustrative way the feature-selection approach we propose in this chapter. For the sake of clarity, a detailed description of the computational cost (in number of clock cycles), demanded by the DSP used by our platform to carry out the experiments, to compute the features considered in the different experiments is also outlined in this section. Finally, Section 4.4 concludes the chapter with a summary of the key points.

4.2 Low-complexity features

In the effort of reducing the computational cost demanded by the DSP to compute some of the features described in Subsection 3.3.2 (page 36), we have designed in this thesis a set of five new, "low-complexity" features inspired by some of the mentioned features, aiming at achieving comparable or even better results in terms of classification performance than those obtained with the original ones, but, this is the crucial point, with the added advantage of extremely reducing the computational cost demanded by the DSP for their computation.

4.2.1 Mathematical expression of the low-complexity features

For this goal to be reached, we have based the design of these features on the following original ones: spectral centroid (SC), voice2white (V2W) and spectral flux (SF). Although other spectral features, such as, for instance, the Mel-frequency cepstral coefficients could be used, we have deliberately restricted the study to the variation of the mentioned features because they are well-known in hearing aid applications, and the computational cost demanded to calculate them in the DSP used to carry out the experiments is relatively low [Cuadra et al., 2010].

The key point that exhibits the greatest relevance in these features basically consists in the fact of *avoiding* some complex mathematical operations, or in other words, operations that require high computational cost for being programmed in the DSP, such as, for instance, the computation of the magnitude of the spectrum of a given sound frame $X_i(t)$ at a particular k-th frequency band (or, mathematically written as, $|\chi_i(k)|$), or the division operation.

With these ideas in mind, the low-complexity features, for each analysis sound frame $X_i(t)$, have been found to be as follows:

- New features based on spectral centroid:

$$\widehat{\mathrm{SC}}_i = \frac{\sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2 \cdot k}{\sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2} \tag{4.1}$$

$$\widetilde{\mathrm{SC}}_i = \sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2 \cdot k \tag{4.2}$$

- New features based on voice2white:

$$\widehat{\mathrm{V2W}}_i = \frac{\sum_{k=M_1}^{M_2} |\chi_i(k)|^2}{\sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2} \tag{4.3}$$

$$\widetilde{\widehat{\mathrm{V2W}}}_i = \sum_{k=M_1}^{M_2} |\chi_i(k)|^2 \tag{4.4}$$

- New feature based on spectral flux:

$$\widehat{\mathrm{SF}}_i = \sum_{k=1}^{N_{\mathrm{B}}} (|\chi_i(k)|^2 - |\chi_{i-1}(k)|^2)^2. \tag{4.5}$$

We label this set of new, low-complexity features $\mathcal{S}_{\mathrm{LC}}$, as depicted below:

$$\mathcal{S}_{\mathrm{LC}} = \{\widehat{\mathrm{SC}}, \widetilde{\mathrm{SC}}, \widehat{\mathrm{V2W}}, \widetilde{\widehat{\mathrm{V2W}}}, \widehat{\mathrm{SF}}\},$$

subscript LC meaning "low-complexity".

4.2.2 Mathematical characterization of the feature extraction process

In the effort of computing the aforementioned low-complexity features and features listed in Subsection 3.3.2, it is necessary to carry out a number of signal processing steps that for the sake of clarity we summarize below.

Let us assume that $\mathcal{F}$ is the set that contains all the available features as follows:

$$\mathcal{F} = [f_1, \ldots, f_{N_{\mathrm{F}}}] \tag{4.6}$$

N_{F} being the number of features.

Any feature f_v can be assumed, in the most general case, as a complex function of p real variables,

$$f_v : \mathbb{C}^p \to \mathbb{R}. \tag{4.7}$$

In our particular case, all the considered features are real.

Since frame $X_i(t)$ has been shown to be a random-variable vector in Expression 3.2 (page 34), then any feature $f_v \in \mathcal{F}$ applied on it, $f_v(\mathbf{X_i})$, is thus a function of p random variables, $f_v(\mathbf{X}_{i1}, \ldots, \mathbf{X}_{ip})$, and, consequently, a random variable. In order to simplify the notation, the random variable $f_v(\mathbf{X}_{i1}, \ldots, \mathbf{X}_{ip})$ will be labelled f_{vi}. Finally, to complete the characterization of the input audio signal, the aforementioned sequence of processes has to be applied onto all the N_{frames} frames into which the input audio signal has been segmented. Please note that N_{frames} consecutive frames compose a time slot, and the classifying algorithm returns a decision per time slot.

Let us imagine now that we are interested in how properly feature $f_v \in \mathcal{F}$ describes the input signal. One of the results that provides the previously described sequence of processes is the random data vector

$$[f_{v1}, \ldots, f_{vN_{\mathrm{frames}}}] \equiv \mathbf{F_v}. \tag{4.8}$$

The elements of this vector are the results obtained when feature f_v is applied to each of the N_{frames} frames into which the input audio signal has been segmented to be processed. The random vector $\mathbf{F_v}$ can be characterized, for instance, by estimating its mean value:

$$\widehat{E}[\mathbf{F_v}] \equiv \mu = \frac{1}{N_{\mathrm{frames}}} \sum_{i=1}^{N_{\mathrm{frames}}} f_{v_i} \tag{4.9}$$

where N_{frames} represents the number of values that the *given* random variable $\mathbf{F_v}$ has. The vector $\mathbf{F_v}$ can be also characterized by estimating its variance:

$$\widehat{\sigma}^2[\mathbf{F_v}] = \frac{1}{N_{\text{frames}}} \sum_{i=1}^{N_{\text{frames}}} (f_{v_i} - \mu)^2 \tag{4.10}$$

its standard deviation:

$$\widehat{\sigma}[\mathbf{F_v}] = \sqrt{\frac{1}{N_{\text{frames}}} \sum_{i=1}^{N_{\text{frames}}} (f_{v_i} - \mu)^2} \tag{4.11}$$

its range:

$$R[\mathbf{F_v}] = \max\,[\mathbf{F_v}] - \min\,[\mathbf{F_v}] \tag{4.12}$$

the most significative bit (MSB) of the mean value:

$$N[\mathbf{F_v}] = \log_2\{\hat{E}[\mathbf{F_v}]\} \tag{4.13}$$

and the MSB of the variance:

$$W[\mathbf{F_v}] = \log_2\{\hat{\sigma}^2[\mathbf{F_v}]\}. \tag{4.14}$$

It is important to note that the previously described statistical characterization can be done for all the available features $f_v \in \mathcal{F}$, so that the feature extraction algorithm generates:

$$\begin{aligned}\mathbf{F} = [&\widehat{E}[\mathbf{F}_1], \widehat{\sigma}^2[\mathbf{F}_1], \widehat{\sigma}[\mathbf{F}_1], R[\mathbf{F}_1], N[\mathbf{F}_1], W[\mathbf{F}_1] \ldots \\ &\ldots \widehat{E}[\mathbf{F}_{N_\text{F}}], \widehat{\sigma}^2[\mathbf{F}_{N_\text{F}}], \widehat{\sigma}[\mathbf{F}_{N_\text{F}}], R[\mathbf{F}_{N_\text{F}}], N[\mathbf{F}_{N_\text{F}}], W[\mathbf{F}_{N_\text{F}}]]^T\end{aligned} \tag{4.15}$$

a random vector whose dimension is $dim(\mathbf{F}) = 6 \cdot N_F = L$.

This is just the signal-describing vector that should feed the classifier. For the sake of clarity, it is formally written as

$$\mathbf{F} = [F_1, \ldots, F_L]^T. \tag{4.16}$$

4.2.3 Preliminary classification results

For illustrative purposes, we have included this brief section that aims at succinctly exhibiting the performance of the aforementioned low-complexity features for the problem of discriminating among speech, music and noise in hearing aids. To achieve this, we have made use of the sound database described in Section C.1 in Appendix C (page 211) (we refer the reader to the mentioned appendix for further details) and the mean square error (MSE) linear classifier, explained in a detailed way in Section 3.4.2 in Chapter 3 (page 44). The results we illustrate below correspond to the test set.

In the batches of experiments we have put into practice, we have considered three different feature sets, which are defined as follows:

- **Set 1** $= \{\text{SC}, \text{V2W}, \text{SF}, \text{STE}\}$.
- **Set 2** $= \{\widehat{\text{SC}}, \widehat{\text{V2W}}, \widehat{\text{SF}}, \text{STE}\}$.

- **Set 3** $= \{\widehat{\mathrm{SC}}, \widetilde{\mathrm{SC}}, \widehat{\mathrm{V2W}}, \widetilde{\mathrm{V2W}}, \widehat{\mathrm{SF}}, \mathrm{STE}\}$.

As shown, features included in “Set 1” label the original ones, along with STE feature because this latter feature demands a relative low number of operations for its programming in the DSP, whereas “Set 2” includes the low-complexity features “more similar” to the original ones, along with also STE feature, and finally, “Set 3” contains *all* the low-complexity features proposed in this thesis, along with STE feature. For the sake of simplicity, we have only computed the mean and variance of the features included in each feature set.

Table 4.1 depicts the percentage of correct classification, P_{CC} (%), reached by the three mentioned feature sets for the speech/music/noise classification task. Please note that it is also shown the number of features included in each set.

	Number of features	P_{CC} (%)
Set 1	8	66.1
Set 2	8	65.5
Set 3	12	67.5

Table 4.1: Mean percentage of correct classification, P_{CC} (%), reached by three different feature sets, which include low-complexity features proposed in this thesis, for distinguishing among speech, music and noise. Note that it is also shown the number of features included in each feature set.

Thanks to this table, we can observe that the new, low-complexity features proposed here achieve similar results than those obtained with the original ones, along with considerably lower computational cost. The way to note this simply consists in comparing the percentage of correct classification, P_{CC} (%), listed in row “Set 1” (that is, that achieved by using the original features), which has been found to be 66.1 %, with that corresponding to “Set 2” (or in other words, that achieved by using the low-complexity features proposed in this thesis), which has been found to be 65.5 %, and the number of features used in both sets. It is worth mentioning that “Set 3” provides a better result than the others do, assisting the classifier in achieving $P_{\mathrm{CC}} = 67.5\,\%$ at the expense of making use of a greater number of features

Finally, completing this brief section demands to mention that these results are very promising in the sense that the low-complexity features included in set $\mathcal{S}_{\mathrm{LC}}$, proposed in this thesis, exhibit very good results, with the added bonus of saving a great amount of computational resources available in the DSP as will be shown later on. Regrettably, the problem here is that implementing in the DSP *all* the features described in this section, along with the classical, spectral features described in Subsection 3.3.2, has been found to be *unfeasible* from a practical implementation point of view, because it exceeds the scarce computational resources available in the DSP. It seems to be clear that it is necessary to reduce the number of features to be programmed in the DSP. Elucidating this is just the objective of the section that immediately follows.

4.3 Fundamentals of feature selection

4.3.1 An intuitive approach

As advanced in the introductory chapter and explained in a more detailed way in Chapter 3, the problem of the “self-adapting hearing aid” is that its implementation is very difficult because of the inherent limitations of the DSP the hearing aid is based on. This *enforces* us to put special emphasis on signal processing techniques and algorithms tailored for properly classifying while using the minimum number of operations. Bearing in mind that the calculations involved in the feature extraction process may be the system's most time-consuming task, programming in the DSP all the features listed in Subsections 3.3.2 and 4.2.1 have been found to be *unfeasible* from a practical implementation point of view.

Indeed such limitations motivate the purpose of this section. Figure 4.1 will assist us in introducing such objective. As shown, the purpose basically consists in searching inside the set of all available features, labelled within our stated framework $\mathcal{F}$, a subset $\mathcal{H} \subseteq \mathcal{F}$, composed of a "reduced" number of satisfactory sound signal-describing features that, by feeding a three-classes classifier, assists it in properly distinguishing among the different acoustic environments (speech, music and noise, in our case), while *presumably* saving DSP resources.

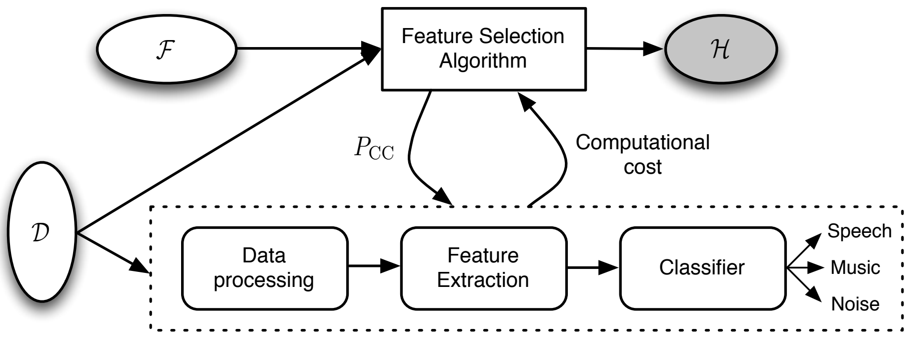


Figure 4.1: Conceptual description of the way the proposed feature-selection algorithm works. It selects features (from the feature set labelled $\mathcal{F}$) and calculates the percentage of correct classification (P_{CC}) and the computational cost demanded by the DSP to calculate the selected features. By minimizing the computational cost and maximizing P_{CC}, the algorithm creates the optimized low-cardinality subset $\mathcal{H} \subseteq \mathcal{F}$.

There are two latent reasons to work towards the objective of selecting sound signal-describing features in sound classification for digital hearing aids:

- The first, more evident reason, is that a lessened number of features to be computed on the DSP will save some of its limited resources.

- The second cause is related to the feature concept itself. As mentioned in Chapter 3, a "proper" feature must contain the sort of audio signal information that allows the classifier to properly distinguish among the different classes considered in each particular classification task. If this is not the case, some features may degrade the classifier performance, basically for two reasons. On the one hand, if the features do not contain all the essential information, the classifier may not perform properly. But on the other, if the selected features contain excessive, irrelevant information, the system will also fail; in this sense, it is said that such irrelevant features behave as noise.

Once we have explained the purpose of this section, the immediate point arising here is how to put it into practice. The method to reach the aforementioned goal is immediately related to: 1) the feature selection problem exhibits an NP-complete behavior [Weston et al., 2000] (NP stands for nondeterministic polynomial time and means that a computer -as we know them today- cannot solve the problem in polynomial time), and 2) the large number of local minima the classifying function has. If the number of available features is large, an exhaustive search through all the combinations of features is not computationally feasible. Solving the stated problem requires to make use of global search and optimization techniques able to deal with solution spaces that have constrained parameters, and a large of local relative extreme. In this respect, a variety of algorithms have been proposed in the literature for solving this type of problem. They include genetic algorithms (GAs) [Goldberg, 1989; Haupt and Haupt, 2004], mutual information techniques [Torkkola, 2002; Torkkola and Campbell, 2000], sequential forward search, sequential backward search [Bishop, 1995], principal/independent component analysis [Hyvaarinen et al., 2001; Jolliffe, 1986], self-organizing maps [Kohonen, 2001], and so on. Genetic algorithms, described in-depth in Appendix D, are particularly effective in finding the global optimum, or at least a local optimum that is good enough, of a multiple local problem [Goldberg, 1989], such as, for instance, the one at hand. Thus, GAs will be explored in this section in the effort of reaching the mentioned goal.

Perhaps the reader may wonder why we make use of GAs for solving this problem, because it may be inconsistent that, although we are looking for ways to save DSP resources, we use GAs, which can eventually suffer from high computational load. In this respect, it is worth mentioning that this does not matter because these algorithms will be run *off-line* in the aim of selecting the most appropriate features. The selected feature subset, $\mathcal{H}$, which has low computational cost, will be the one that is finally implemented in the DSP.

4.3.2 How should the feature selection algorithm work?

Keeping in mind the data structure described in Subsection 4.2.2, the purpose of this section can be "reformulated" in the following way. Given $\mathcal{F}$, the set of L available, general-purpose features, the algorithm should find a subset $\mathcal{H} \subseteq \mathcal{F}$ that maximizes the percentage of correct classification, P_{CC}, and minimizes the computational cost, constrained by the following conditions:

- $card(\mathcal{H}) \leqslant card(\mathcal{F})$, $card()$ denoting the cardinality of the considered set. Note that $card(\mathcal{H})$ and the elements belonging to subset $\mathcal{H}$ are unknown *a priori*. This statement is equivalent to reducing the dimension of the feature vector, or in other words, to finding a vector $\mathbf{H}$, which contains N sound signal-describing features and whose dimension $dim(\mathbf{H}) = N \leqslant dim(\mathbf{F}) = L$,
- the classifier accuracy is maximized (or, equivalently, the percentage of correct classification is maximized), and finally,
- the computational cost required to compute the feature vector $\mathbf{H}$ and classify the input audio signal are minimized.

Put it in a more schematic way, we can say that, for any candidate feature subset $\mathcal{H}$, the algorithm has to complete the following sequence of operations:

- feeding the complete system with audio input files from the adequate database $\mathcal{D}$ as illustrated in Figure 4.1,
- calculating the feature vector $\mathbf{H}$ for the complete file,
- classifying the file, and finally,
- calculating 1) the mean square error between the classifier outputs $(\widehat{O})$, and the correct outputs (O), labelled MSE$(\widehat{O}, O)$, 2) the percentage of correct classification (P_{CC}) and 3) the computational cost demanded by the DSP to compute the feature vector $\mathbf{H}$.

This sequence of operations can be considered a wrapper method [Kohavi and John, 1997] since, in order to select features, it uses, as a selection criterion, the output of the learning machine *itself*. The main advantage of this approach is that, if properly designed, it guarantees that in each step of the algorithm, the performance of the selected subset is better than the previous one. Just in this respect, GAs are easy to implement and provide very good results in terms of the selected features and the overall performance of the classifier.

4.3.3 Practical implementation

To put into practice this global design, it is worth mentioning that, although for the particular problem at hand, one may expect that $\mathcal{F} = \mathcal{S}_{\mathrm{C}} \cup \mathcal{S}_{\mathrm{LC}}$, in an effort of saving computational resources, we have reduced the number of signal-describing features involved in the feature set $\mathcal{F}$. In this sense, regarding the spectral features proposed in the literature (that is, those listed in Subsection 3.3.2), it has been found to be *unfeasible*, from a practical implementation point of view, the fact of programming in the DSP *all* these features. In order to find the classical, spectral features more appropriate for carrying out the batches of experiments outlined in this section, we have defined two different feature sets that are defined as follows:

- **Set 4** $= \{\mathrm{SC}, \mathrm{V2W}, \mathrm{SF}, \mathrm{STE}, [1, \ldots, 20]\,\mathrm{MFCCs}, [1, \ldots, 20]\,\Delta\mathrm{MFCCs}\}$.
- **Set 5** $= \{\mathrm{SC}, \mathrm{V2W}, \mathrm{SF}, \mathrm{STE}, [1, \ldots, 20]\,\mathrm{MFCCs}, [1, \ldots, 20]\,\Delta\mathrm{MFCCs}, \mathrm{SR}, \mathrm{SB}, \mathrm{SCF}, \mathrm{SFM}, \mathrm{RE}, \mathrm{SE}\}$.

Perhaps the reader may wonder why we have selected these two particular feature sets. The reason is as follows. On the one hand, the set of features labelled "Set 4" includes the features contained in "Set 1", along with $[1, \ldots, 20]$ MFCCs and $[1, \ldots, 20]$ ΔMFCCs. Both MFCCs and ΔMFCCs features are also well-known in sound classification and, as will be shown throughout this chapter, they can be computed in the DSP with relatively low computational complexity. On the other hand, "Set 5" includes *all* the classical, spectral features considered in this work in the effort of evaluating the performance of the whole classical, spectral feature set $\mathcal{S}_{\mathrm{C}}$.

Aiming at evaluating the performance of these two feature sets, we have illustrated in Table 4.2 the percentage of correct classification reached by the two sets for the speech/music/noise classification task. To achieve this, we have made use of the sound database describe in Section C.1 in Appendix C (page 211) and the mean square error (MSE) linear classifier. As in Table 4.1, it is also shown the number of features included in each set.

	Number of features	P_{CC} (%)
Set 4	88	86.6
Set 5	100	89.4

Table 4.2: Mean percentage of correct classification, P_{CC} (%), reached by two different feature sets, consisting of classical, spectral features proposed in the literature, for distinguishing among speech, music and noise. Note that it is also shown the number of features included in each feature set.

Having a brief look at the above table, one can notice that "Set 5" provides a better result than the other does, assisting the classifier in achieving $P_{\mathrm{CC}} = 89.4\,\%$. However, it is worth noting that the result reached by "Set 4", being close to 86.6 %, is quite similar than that obtained by "Set 3", with the added advantage of using a lower number of features, or equivalently, requiring lower computational cost for its implementation in the DSP.

With this in mind, it seems to be clear that:

$$\mathcal{F} = \mathcal{S}_{\mathrm{C}} \cup \text{Set } 4$$

that is,

$$\mathcal{F} = \text{Set } 3 \cup \text{Set } 4$$

or, equivalently,

$$\mathcal{F} = \{\mathrm{SC}, \mathrm{V2W}, \mathrm{SF}, \mathrm{STE}, 20\mathrm{MFCCs}, 20\Delta\mathrm{MFCCs}, \widehat{\mathrm{SC}}, \widetilde{\mathrm{SC}}, \widehat{\mathrm{V2W}}, \widetilde{\mathrm{V2W}}, \widehat{\mathrm{SF}}\}.$$

With regard to the statistical characterization of the features included in $\mathcal{F}$, it is worth noting that the mean, variance, standard deviation, range and MSB of both mean and variance of the listed features are estimated, which basically leads to compute the resulting feature vector as follows:

$$\begin{aligned}
\mathbf{F} = [&\widehat{E}[\mathbf{SC}], \widehat{\sigma}^2[\mathbf{SC}], \widehat{\sigma}[\mathbf{SC}], R[\mathbf{SC}], N[\mathbf{SC}], W[\mathbf{SC}],\\
&\widehat{E}[\mathbf{V2W}], \widehat{\sigma}^2[\mathbf{V2W}], \widehat{\sigma}[\mathbf{V2W}], R[\mathbf{V2W}], N[\mathbf{V2W}], W[\mathbf{V2W}],\\
&\ldots\\
&\widehat{\sigma}^2[\mathbf{\Delta MFCC}], \widehat{\sigma}[\mathbf{\Delta MFCC}], R[\mathbf{\Delta MFCC}], W[\mathbf{\Delta MFCC}],\\
&\ldots\\
&\widehat{E}[\widetilde{\mathbf{V2W}}], \widehat{\sigma}^2[\widetilde{\mathbf{V2W}}], \widehat{\sigma}[\widetilde{\mathbf{V2W}}], R[\widetilde{\mathbf{V2W}}], N[\widetilde{\mathbf{V2W}}], W[\widetilde{\mathbf{V2W}}],\\
&\widehat{E}[\widehat{\mathbf{SF}}], \hat{\sigma}^2[\widehat{\mathbf{SF}}], \widehat{\sigma}[\widehat{\mathbf{SF}}], R[\widehat{\mathbf{SF}}], N[\widehat{\mathbf{SF}}], W[\widehat{\mathbf{SF}}]]^T
\end{aligned} \tag{4.17}$$

being thus $dim(\mathbf{F}) = L = 254$, or in other words, the total number of features available in $\mathcal{F}$ is 254. Just in this respect, it is important to note that as the mean as MSB of mean of ΔMFCCs are not computed because they result in zero.

Having a look at the previously described sequence of operations in the proposed algorithm, it seems clear that it is necessary to calculate the computational cost demanded by the DSP to compute the features available in the feature $\mathcal{F}$. Just in this respect, the following subsections aim to help the reader understand the computational cost demanded by the DSP, used by our platform to carry out the experiments, to program the features included in the feature set $\mathcal{F}$.

4.3.3.1 Computational cost of classical and low-complexity features

We deduce here the computational cost (in number of clock cycles) required to compute the feature set $\{\mathrm{SC}, \mathrm{V2W}, \mathrm{SF}, \mathrm{STE}, \widehat{\mathrm{SC}}, \widetilde{\mathrm{SC}}, \widehat{\mathrm{V2W}}, \widetilde{\mathrm{V2W}}, \widehat{\mathrm{SF}}\}$. For the sake of clarity, we have labelled this feature set $\mathcal{S}_1$. Therefore, the feature set $\mathcal{S}_1$ can be written formally as follows:

$$\mathcal{S}_1 = \{\mathrm{SC}, \mathrm{V2W}, \mathrm{SF}, \mathrm{STE}, \widehat{\mathrm{SC}}, \widetilde{\mathrm{SC}}, \widehat{\mathrm{V2W}}, \widetilde{\mathrm{V2W}}, \widehat{\mathrm{SF}}\}.$$

Aiming at making this section stand by itself, we have defined seven "building blocks" that we have labelled $\mathrm{BB}_i(b)$, where $b = 1, \ldots, B$ is the index over the B building blocks (in our study-case, as will be shown later on, $B = 7$), while $i = 1, \ldots, N_{\text{frames}}$ labels the one over the N_{frames} frames into which any time slot is segmented. The main purpose of these building blocks is just to provide a valuable mathematical insight that helps us analytically define the features included in the set $\mathcal{S}_1$ as a function of the mentioned building blocks, as clearly explained in the paragraphs that follow. With this in mind, these building blocks are defined as depicted in Table 4.3.

Building block	**Definition**
$\mathrm{BB}_i(1)$	$\sum_{k=1}^{N_\mathrm{B}} \lvert\chi_i(k)\rvert \cdot k$
$\mathrm{BB}_i(2)$	$\sum_{k=1}^{N_\mathrm{B}} \lvert\chi_i(k)\rvert^2 \cdot k$
$\mathrm{BB}_i(3)$	$\sum_{k=1}^{N_\mathrm{B}} \lvert\chi_i(k)\rvert$
$\mathrm{BB}_i(4)$	$\sum_{k=1}^{N_\mathrm{B}} \lvert\chi_i(k)\rvert^2$
$\mathrm{BB}_i(5)$	$\sum_{k=M_1}^{M_2} \lvert\chi_i(k)\rvert$
$\mathrm{BB}_i(6)$	$\sum_{k=M_1}^{M_2} \lvert\chi_i(k)\rvert^2$
$\mathrm{BB}_i(7)$	$\sum_{k=1}^{N_\mathrm{B}} \lvert\chi_i(k)\rvert \cdot \lvert\chi_{i-1}(k)\rvert$

Table 4.3: Mathematical expression of seven "building blocks" designed aiming at providing a valuable mathematical tool that help us analytically define the features included in the feature set $\mathcal{S}_1 = \{\mathrm{SC}, \mathrm{V2W}, \mathrm{SF}, \mathrm{STE}, \widehat{\mathrm{SC}}, \widetilde{\mathrm{SC}}, \widehat{\mathrm{V2W}}, \widetilde{\mathrm{V2W}}, \widehat{\mathrm{SF}}\}$ as a function of the mentioned building blocks. We have labelled these building blocks $\mathrm{BB}_i(b)$, being b-index the index over the B building blocks (in our study-case, $B = 7$), while $i = 1, \ldots, N_\mathrm{frames}$ labels the one over the N_frames frames into which any time slot is segmented.

For the sake of clarity and in the aim of facilitating the reader's comprehension, we have preferred to clearly explain the way such involved features in the set $\mathcal{S}_1$ can be expressed as a function of these building blocks. In particular, we have illustrated the way that, for instance, the SC and $\widetilde{\mathrm{SC}}$ features calculated for a given sound frame $X_i(t)$ are expressed. For this purpose, having a look at Table 4.3 and Expressions 3.3 and 4.1, it can be clearly noticed that SC_i (note that SC_i simply labels the SC feature for the frame i-th) makes use of $\mathrm{BB}_i(1)$ and $\mathrm{BB}_i(3)$, whereas $\widetilde{\mathrm{SC}}_i$ (in the same line of reasoning, $\widetilde{\mathrm{SC}}_i$ designates the $\widetilde{\mathrm{SC}}$ feature for the frame i-th) only uses $\mathrm{BB}_i(2)$. By means of this example, it is very illustrative to note that, at first glance, the computational cost demanded by the DSP to calculate the $\widetilde{\mathrm{SC}}$ feature for a given input sound frame is much lower than that demanded to compute the original one, that is, the SC feature for such frame.

In the effort of showing the way the remaining features are expressed as a function of the building blocks, we have included Table 4.4 in which, for each of the features available in the set $\mathcal{S}_1$, we have placed an X in each building block they make use of. Please note that i simply labels the index over the N_frames frames into which any time slot is segmented.

With this in mind, we can proceed further in explaining the computational cost (in number of clock cycles) demanded by the DSP to program the computation of the mentioned building blocks. For this purpose, since the building blocks are basically computed by using addition, addition-multiplication or square root operations, we have partially determined the computational cost, for each building block, as the sum of number of clock cycles required to carry out the aforementioned operations. In this respect, it is worth mentioning that the addition operation as well as the addition-multiplication operation require only 1 clock cycle in the DSP used to carry out the experiments, whereas the root square operation needs for its implementation 11 clock cycles. For example, since BB(1) needs 64 addition-multiplication operations, and 64

Features	**Building blocks**						
	$BB_i(1)$	$BB_i(2)$	$BB_i(3)$	$BB_i(4)$	$BB_i(5)$	$BB_i(6)$	$BB_i(7)$
SC_i	X		X				
$\widehat{SC}_i$		X		X			
$\widetilde{SC}_i$		X					
$V2W_i$			X		X		
$\widehat{V2W}_i$				X		X	
$\widetilde{V2W}_i$						X	
SF_i				X			X
$\widehat{SF}_i$				X			
STE_i				X			

Table 4.4: Table illustrating for each of the features arranged in the set $\mathcal{S}_1$, the building blocks they make use of. For this purpose, we have placed an X on each building block they use. Note that $i = 1, \ldots, N_{\text{frames}}$ is the index over the N_{frames} frames into which any time slot is segmented.

square roots operations, the total number of clock cycles needed would be 768. In the aim of making easier the reader's comprehension, we have succinctly represented the total number of clock cycles required by each building block in Table 4.5.

Building block	**Operations**			**Total clock cycles**
	Addition (1 cycle)	**Multiplication** (1 cycle)	**Square root** (11 cycles)	
$BB_i(1)$	0	64	64	768
$BB_i(2)$	0	64	0	64
$BB_i(3)$	64	0	64	768
$BB_i(4)$	64	0	0	64
$BB_i(5)$	34	0	34	408
$BB_i(6)$	34	0	0	34
$BB_i(7)$	0	64	64	768

Table 4.5: Total number of clock cycles demanded by the DSP, used to carry out the experiments, to implement each building block. Since the building blocks are calculated by means of addition, addition-multiplication or square root operations, we have partially determined the computational cost as the sum of clock cycles needed to carry out the aforementioned operations, which require 1 clock-cycle, 1 clock-cycle and 11 clock-cycles, respectively.

In this respect, it is very illustrative to note that the building blocks that require for its computation to calculate the magnitude of the spectrum and consequently to carry out a root square operation at each analysis frame, such as, for instance, building blocks labelled BB(1), BB(3) or BB(7), are the blocks that demand significantly higher computational cost.

Prior to depicting the total number of clock cycles demanded by the DSP to program the features included in the set $\mathcal{S}_1$, it is indispensable to have a brief look at the number of clock cycles needed to carry out "additional operations", such as, for instance, to perform a division operation between two building blocks or to store and to recover data from registers (by means of `push` and `pop` instructions), to call a subroutine (by using `call` instruction) or to return from a subroutine (by making use of `ret` subroutine). We have clearly summarized in Table 4.6, for each building block, these additional number of clock cycles. Please note that the division operation requires a total of 15 clock cycles.

Feature	**Additional operations**			**Additional clock cycles**	**Total clock cycles**
	Addition (1 cycle)	**Multiplication** (1 cycle)	**Division** (15 cycles)		
SC_i	1	1	1	21	38
$\widehat{\mathrm{SC}_i}$	1	1	1	21	38
$\widetilde{\mathrm{SC}_i}$	1	1	0	11	13
$\mathrm{V2W}_i$	1	1	1	19	36
$\widehat{\mathrm{V2W}_i}$	1	1	1	19	36
$\widetilde{\mathrm{V2W}_i}$	1	1	0	9	11
SF_i	2	2	0	8	12
$\widehat{\mathrm{SF}_i}$	2	1	0	8	11
STE_i	1	1	0	8	10

Table 4.6: Summary of the number of "additional" clock cycles required for calculating the features, arranged in the set $\mathcal{S}_1$, in the DSP used to carry out the experiments. We say additional in the sense that these clock-cycles are due to additional operations, such as, for instance, to compute the division operation between two building blocks or to store and to recover data from registers, to call a subroutine or to return from a subroutine.

Now, with this in mind, we have the background to define the computational cost demanded by the DSP to program any of the features included in the set $\mathcal{S}_1$. Let us imagine that we are interested in calculating the total computational cost associated to the programming of the SC feature for the analysis frame i-th (that is, SC_i). For this purpose, we have a look at Table 4.4 that basically assists us in knowing what building blocks the feature SC_i makes use of. Regarding this, it is very clear to note that this feature makes use of both building blocks labelled $\mathrm{BB}_i(1)$ and $\mathrm{BB}_i(3)$. Then, thanks to Tables 4.5 and 4.6, we can conclude that the computational cost required to implement the SC_i feature in the DSP has been found to be:

$$\begin{aligned}\mathcal{C}(\mathrm{SC}) &= \mathcal{C}(\mathrm{BB}(1)) + \mathcal{C}(\mathrm{BB}(3)) + \mathcal{C}(\text{Additional cycles}) \\ &= 768 + 64 + 38 \\ &= 870 \text{ clock cycles.}\end{aligned} \tag{4.18}$$

Perhaps the reader may wonder why we have written that the computational cost of the building block labelled BB(3) is 64 clock cycles instead of 768 clock cycles as depicted in Table 4.5. In this respect, it is worth mentioning that the computation of the magnitude of $\chi_i(k)$, or in other words, $|\chi_i(k)|$, needs to be calculated for *both* building blocks. Intuitively, instead of computing $|\chi_i(k)|$ twice, it is only once calculated and then the value is stored in the data memory available in the DSP. For that reason, we have written that the computational cost for programming the building block labelled BB(3) is only 64 clock cycles instead of 768 clock cycles as represented in the Table 4.5.

Finally, we complete this section by summarizing in Table 4.7 the total computational cost (in number of clock cycles) of each of the features available in the set $\mathcal{S}_1$. As shown, the computational cost required to implement in the DSP the low-complexity features proposed in this chapter is significantly lower than that needed to program the "original" features, or in other words, the SC, V2W or SF features. Bearing in mind the classification results obtained in Subsection 4.2.3, we can conclude that the low-complexity features proposed in this chapter are very appropriate for the classification problem at hand, not only in terms of computational complexity but also in terms of classification performance.

Feature	**Computational cost (clock cycles)**
SC_i	870
$\widehat{\mathrm{SC}}_i$	166
$\widetilde{\mathrm{SC}}_i$	77
$\mathrm{V2W}_i$	838
$\widehat{\mathrm{V2W}}_i$	134
$\widetilde{\mathrm{V2W}}_i$	45
SF_i	844
$\widehat{\mathrm{SF}}_i$	75
STE_i	74

Table 4.7: Total computational cost (in number of clock cycles) demanded by the DSP, used to carry out the experiments, to implement each of the features arranged in the set $\mathcal{S}_1$. As clearly illustrated, the new, low-complexity features proposed in this thesis (that is, $\widehat{\mathrm{SC}}$, $\widetilde{\mathrm{SC}}$, $\widehat{\mathrm{V2W}}$, $\widetilde{\mathrm{V2W}}$ and $\widehat{\mathrm{SF}}$) demand a lower number of clock cycles than those demanded by the original ones (or equivalently, SC, V2W and SF).

4.3.3.2 Computational cost of MFCCs and ΔMFCCs

Roughly speaking, the computational cost demanded to program in the DSP the computation of MFCCs and ΔMFCCs features depends on the number of filter coefficients needed in the filter bank to obtain each MFCC. In this respect, we can proceed in explaining the computational cost (in number of clock cycles) required to compute each MFCC as follows:

$$\mathcal{C}(\mathrm{MFCC}_n) = \mathcal{C}(\text{Filter bank}) + \mathcal{C}(\text{Logarithm}) + 1 \tag{4.19}$$

where:

- $\mathcal{C}$(Filter bank) represents the computational cost (in number of clock cycles) needed to obtain, at the output filter bank, the n-th MFCC, or as we have labelled it MFCC_n (n-index ranging from 1 to 20, in our case),
- $\mathcal{C}$(Logarithm) designates the computational cost required to perform the logarithm operation in the DSP, being $\mathcal{C}(\text{Logarithm}) = 13$ clock cycles in the DSP used by our platform to carry out the experiments (see [Dspfactory, 2002a,b] for further details), and finally,
- 1 simply *labels* an additional clock cycle due to an accumulation operation.

Table 4.8 will assist us in illustrating the total computational cost required to compute each MFCC in the DSP. Note that the higher the number of calculated MFCCs is, the higher the computational cost required is. Intuitively, this is due to the fact that as the number of MFCC to be computed is higher, the number of filter coefficients to be calculated increases.

On the other hand, the computational cost required to compute in the DSP each ΔMFCC is intimately related to the computational cost needed to calculate the MFCCs they depend on. To better explain this, the expression shown below illustrates the computational cost (in number of clock cycles) demanded to calculate each ΔMFCC in the DSP used by our platform to carry out the experiments:

$$\mathcal{C}(\Delta\mathrm{MFCC}_n) = \mathcal{C}(\mathrm{MFCC}_{n-2}) + \mathcal{C}(\mathrm{MFCC}_{n+2}) + 1 \tag{4.20}$$

Coefficient	**Computational cost (clock cycles)**			**Total**
	Filter bank	**Logarithm**	**Extra operation**	**clock cycles**
$MFCC_1$	1	13	1	15
$MFCC_2$	2	13	1	16
$MFCC_3$	2	13	1	16
$MFCC_4$	2	13	1	16
$MFCC_5$	2	13	1	16
$MFCC_6$	3	13	1	17
$MFCC_7$	4	13	1	18
$MFCC_8$	3	13	1	17
$MFCC_9$	3	13	1	17
$MFCC_{10}$	5	13	1	19
$MFCC_{11}$	6	13	1	20
$MFCC_{12}$	6	13	1	20
$MFCC_{13}$	6	13	1	20
$MFCC_{14}$	7	13	1	21
$MFCC_{15}$	8	13	1	22
$MFCC_{16}$	9	13	1	23
$MFCC_{17}$	10	13	1	24
$MFCC_{18}$	12	13	1	26
$MFCC_{19}$	14	13	1	28
$MFCC_{20}$	14	13	1	28

Table 4.8: Total computational cost (in number of clock cycles) demanded to compute each MFCC in the DSP used to carry out the experiments. Note that the higher the number of calculated MFCC is, the higher the computational cost required is. Intuitively, this is due to the fact that as the number of MFCC to be computed is higher, the number of filter coefficients to be calculated increases.

where:

- $\mathcal{C}(\text{MFCC}_{n-2})$ intuitively represents the computational cost needed to compute in the DSP the MFCC_{n-2}, and in the same line of reasoning,
- $\mathcal{C}(\text{MFCC}_{n+2})$ represents the computational cost needed to obtain in the DSP the MFCC_{n+2}, and finally,
- 1 simply *labels* an additional clock cycle due to an accumulation operation.

With this scenario in mind, Table 4.9 summarizes the total number of clock cycles needed to compute each ΔMFCC in the DSP.

As illustrated, the higher the number of calculated ΔMFCC is, the higher the computational cost required is, except from that corresponding to the one for ΔMFFC_{19} or ΔMFFC_{20}. To explain this, it is worth mentioning that for determining the computational cost demanded to program in the DSP, such as, for instance, ΔMFCC_1 or ΔMFCC_2, we would need to compute the computational cost to implement MFCC_{-1} or MFCC_0, respectively. Since these MFCCs *do not* exist, we assume that,

$$\mathcal{C}(\text{MFCC}_{-1}) = \mathcal{C}(\text{MFCC}_0) = 0. \tag{4.21}$$

In the same line of reasoning, we can assume that the computational costs for computing MFCC_{21} or MFCC_{22} required to implement ΔMFCC_{19} or ΔMFCC_{20}, respectively, have been

Coefficient	Computational cost (clock cycles)			Total clock cycles
	$\mathbf{MFCC}_{n-2}$	$\mathbf{MFCC}_{n+2}$	**Extra operation**	
ΔMFFC$_1$	0	16	1	17
ΔMFFC$_2$	0	16	1	17
ΔMFFC$_3$	15	16	1	32
ΔMFFC$_4$	16	17	1	34
ΔMFFC$_5$	16	18	1	35
ΔMFFC$_6$	16	17	1	34
ΔMFFC$_7$	16	17	1	34
ΔMFFC$_8$	17	19	1	37
ΔMFFC$_9$	18	20	1	39
ΔMFFC$_{10}$	17	20	1	38
ΔMFFC$_{11}$	17	20	1	38
ΔMFFC$_{12}$	19	29	1	41
ΔMFFC$_{13}$	20	22	1	43
ΔMFFC$_{14}$	20	23	1	44
ΔMFFC$_{15}$	20	24	1	45
ΔMFFC$_{16}$	21	26	1	48
ΔMFFC$_{17}$	22	28	1	49
ΔMFFC$_{18}$	23	28	1	52
ΔMFFC$_{19}$	24	0	1	25
ΔMFFC$_{20}$	26	0	1	27

Table 4.9: Total computational cost (in number of clock cycles) required to compute each ΔMFCC in the DSP used to carry out the experiments. In general, the higher the number of calculated ΔMFCC is, the higher the computational cost required is.

found to be:

$$\mathcal{C}(\text{MFCC}_{21}) = \mathcal{C}(\text{MFCC}_{22}) = 0. \tag{4.22}$$

Another important point to note is the fact of considering the scenario in which we are interested in determining the computational cost to implement a feature vector **F** which contains, for instance, ΔMFCC$_6$ and ΔMFCC$_{10}$, that is, $\mathbf{F} = [\Delta\text{MFCC}_6, \Delta\text{MFCC}_{10}]$. With this in mind, the computational cost of MFCCs to be considered would be: $\mathcal{C}(\text{MFCC}_4)$, $\mathcal{C}(\text{MFCC}_8)$ for computing ΔMFCC$_6$, and, $\mathcal{C}(\text{MFCC}_8)$, $\mathcal{C}(\text{MFCC}_{12})$ for calculating ΔMFCC$_{10}$. At this respect, it is important to note that in the aim of estimating the computational cost of the mentioned feature vector, the number of clock cycles needed to obtain MFCC$_8$ should be taking into account only once.

For completing this section, it is worth mentioning that the computational cost for programming in the DSP the computation of MFCCs and ΔMFCCs features is relatively low, when compared to that needed to implement the SC, V2W or SF features. Basically, this will involve that in experiments in which the feature-selection approach is constrained to a relatively low number of clock cycles (or, equivalently, computational cost), the *trial* feature vector **H** will probably consist only of MFCCs and ΔMFCCs features.

4.3.3.3 Computational cost of statistical operators

Finally, for properly completing this description, we illustrate in Table 4.10 the computational cost (in clock-cycles) demanded by the DSP to estimate the statistical operators considered in the batches of experiments. In the effort of facilitating the reader's comprehension, it is worth mentioning that the statistical operators considered are the mean, variance, standard deviation, range and MSBs of both mean and variance (see Subsection 4.2.2 for further details).

Statistical operator	**Computational cost (clock cycles)**
Mean	9
Variance	23
Standard deviation	23
Range	7
Most significative bit (MSB)	3

Table 4.10: Summary of the computational cost (in number of clock cycles) required by the DSP, used to carry out the experiments, to program the computation of the statistical operators considered in the different experiments (mean, variance, standard deviation, range and MSBs of both mean and variance).

Having a look at the aforementioned table, it is worth mentioning that, since computing the variance of a vector requires to calculate previously the mean of the mentioned vector, the number of clock cycles illustrated in the table for both measures, the variance or standard deviation, include thus the number of clock cycles required to compute also the mean of the vector, that is, 9 clock cycles. For illustrative purposes, this basically means that the total computational cost for computing in the DSP, for instance, the variance of SC feature for the analysis sound frame has been found to be 893 clock cycles, as summarized below:

$$\begin{aligned} \mathcal{C}(\widehat{\sigma}^2[\mathrm{SC}_i]) &= \mathcal{C}(\mathrm{SC}_i) + \mathcal{C}(\widehat{\sigma}^2[\mathrm{SC}_i]) \\ &= 870 + 23 \\ &= 893 \text{ clock cycles.} \end{aligned} \tag{4.23}$$

However, the total computational load for computing in the DSP both the mean and variance of SC feature for the analysis frame has been found to be also 893 clock cycles. This is clearly summarized below:

$$\begin{aligned} \mathcal{C}(\widehat{E}[\mathrm{SC}_i] \,\&\, \widehat{\sigma}^2[\mathrm{SC}_i]) &= \mathcal{C}(\mathrm{SC}_i) + \mathcal{C}(\hat{E}[\mathrm{SC}_i]) + \mathcal{C}(\hat{\sigma}^2[\mathrm{SC}_i]) \\ &= 870 + 9 + 14 \\ &= 893 \text{ clock cycles.} \end{aligned} \tag{4.24}$$

4.3.4 Experimental work and results

Prior to the description of the batches of experiments carried out in this section in an effort of evaluating the performance of the feature-selection approach when selecting features with a constrained maximum number of clock cycles (or, equivalently, computational cost), and the discussion of the corresponding results (Subsection 4.3.4.2), it is worth having a brief look at the sound database and classifier used for the experiments, along with a summary of the main design parameters the GA, in which the feature-selection approach is based on, makes use of (Subsection 4.3.4.1).

4.3.4.1 Experimental setup

In order to carry out the batches of experiments, we have made use of the sound database described in Section C.1 in Appendix C (page 211). Regarding the classifier, in the same line of reasoning as that in the sequence of experiments carried out in Subsection 4.2.3, we have made use of the mean square error (MSE) linear classifier because of its simplicity and good results.

Concerning to the feature set $\mathcal{F}$, into which our feature-selection approach searches for the best sound-describing features, it is worth mentioning that we have not used *all* the features included in both $\mathcal{S}_{\mathrm{C}}$ and $\mathcal{S}_{\mathrm{LC}}$ feature sets because it would take a large time for our approach to properly converge in the experiments. As mentioned, according the classification results shown in Table 4.1 and 4.2, we have made use of the features included in both "Set 3" and "Set 4", that is, $\mathcal{F} = \text{Set } 3 \cup \text{Set } 4$, as a balance between the number of features used and a good discrimination among speech, music and noise. With this in mind, we list below the features that we have extracted:

- Mean, variance, standard deviation, range and MSB of both mean and variance of: SC, $\widehat{\mathrm{SC}}$, $\widetilde{\mathrm{SC}}$, V2W, $\widehat{\mathrm{V2W}}$, $\widetilde{\mathrm{V2W}}$, SF, $\widehat{\mathrm{SF}}$ and STE.

- Mean, variance, standard deviation, range and MSB of both mean and variance of $[1, \ldots, 20]$ MFCCs.

- Variance, standard deviation and MSB of variance of $[1, \ldots, 20]$ of ΔMFCCs.

Intuitively, the final 254-feature vector $\mathbf{F}$ (note that 254 here labels the dimension of the feature vector, that is, $dim(\mathbf{F}) = 254$) is created by calculating the abovementioned features from the input sound signal. It is interesting to point out that some of these features show a high correlation between them which assist us in exploring the efficiency of the feature-selection approach proposed in this thesis. As will be shown in the numerical results obtained throughout this section (Subsection 4.3.4.2), our approach is robust enough to manage this problem.

Figure 4.1 assisted us in introducing the feature-selection approach proposed in this thesis. As shown in that figure, the basic idea underlying this approach is to compute the percentage of correct classification (P_{CC}) achieved by using each candidate feature vector $\mathbf{H}$ and the computational cost ($\mathcal{C}$) demanded by the DSP to calculate the mentioned feature vector $\mathbf{H}$. The higher the percentage of correct classification is, the best suited the feature vector is. The GA looks for, by maximizing the P_{CC}, the best suited feature vector $\mathbf{H}$ for the particular problem at hand. Or in other words, for selecting the best feature set $\mathcal{H}$, we have utilized a single GA that computes those features v_q that maximize the percentage of correct classification (the *objective function* here) for the available validation-patterns in the classification process with a constrained maximum number of clock cycles (or, equivalently, computational cost). For the sake of clarity when discussing the results achieved with the different approaches explored in this section, we have labelled this approach "constrained GA-based approach". With this in mind, any *individual* is thus a binary vector with the structure $I \equiv [v_1, v_2, v_3, \ldots, v_{254}]$, whose elements v_q encode whether the q-th feature in the candidate vector $\mathbf{H}$ is used or not. Intuitively, the allele $v_q = 1$ codifies that the feature H_q has been selected, however, $v_q = 0$ indicates that this has not happened. The goal of the GA is to find the *best individual* $I_{\mathrm{best}} \equiv [v_{\mathrm{best}1}, v_{\mathrm{best}2}, v_{\mathrm{best}3}, \ldots, v_{\mathrm{best}254}]$ that is "best fitted", or in other words, the one that maximizes the percentage of correct classification (computed over the patterns available in the validation phase), while minimizes the computational cost demanded by the DSP for its implementation. The complete GA operates as listed in Appendix D.

For comparative purposes, we have also considered the approach in which the feature-selection approach is *not* constrained to a maximum number of clock cycles. We have labelled this last approach "unconstrained GA-based approach".

We complete this description by summarizing in Table 4.11 the main design parameters of the GA, in which the feature-selection approach is based on. These values have been found to be large enough to allow the algorithm to properly converge in our experiments.

Parameters	**Value**
Initial population (p_0)	1000
Crossover probability (p_c)	0.8
Mutation probability (p_m)	0.1
Max. no. of generations	50
Max. no. of iterations in which P_{CC} remains unchanged	20

Table 4.11: Summary of the design parameters the GA makes use of for automatically selecting the "best suited" feature vector with a constrained maximum number of clock cycles (or, equivalently, computational cost).

Completing this description of the batches of experiments demands to mention that the experiments have been repeated 20 times. The best of these realizations in terms of percentage of correct classification computed over the patterns available in a *validation* phase has been selected. The results we have illustrated in the following subsection correspond to the *test* set.

4.3.4.2 Numerical results

Figure 4.2 shows the percentage of correct classification, $P_{\mathrm{CC}}\,(\%)$, for distinguishing among speech, music and noise as a function of the maximum study-case number of clock cycles (or, equivalently, computational cost).

The single, filled dot shows the percentage of correct classification achieved by using the "unconstrained GA-based approach", or in other words, when the feature-selection approach is *not* constrained to a maximum number of clock cycles. The continuous line shows the results achieved by the "constrained GA-based approach" we explore in this chapter. For comparative purposes, the dashed line represents the percentage of correct classification reached when there is *no feature selection*, or equivalently, when *all* features available in both "Set 3" and "Set 4" feature sets are used to feed the MSE linear classifier. Regarding the results illustrated in Figure 4.2, we would like to emphasize that:

1. The dashed line (which represents the *no selection feature* situation) is constant since it *does not* depend on the computational cost. Or in other words, it represents the percentage of correct classification when *all* features available in both "Set 3" and "Set 4' feature sets are used to feed the classifier.

2. The "unconstrained GA-based approach" selects, among the 254 available features, a subset $\mathcal{H}$, which for the best realization results in containing $dim(\mathbf{H}) = 109$ features. Making use of these features, the classifier achieves $P_{\mathrm{CC}} \approx 91\,\%$.

3. To reach $P_{\mathrm{CC}} \approx 88.5\,\%$, the "constrained GA-based approach" requires only 50 features. In other words, the restricted GA algorithm selects a feature set $\mathcal{H}$ containing a reduced number of features ($dim(\mathbf{H}) = 50$), which allows the classifier to reach quite similar results

than those obtained with the feature vector obtained by means of the “unconstrained GA-based approach”, with the added bonus of decreasing the computational cost demanded by the DSP to implement the feature vector. Thus, the fact of using more features *does not* cause a substantial improvement of percentage of correct classification, though it would require larger amount of computational power.

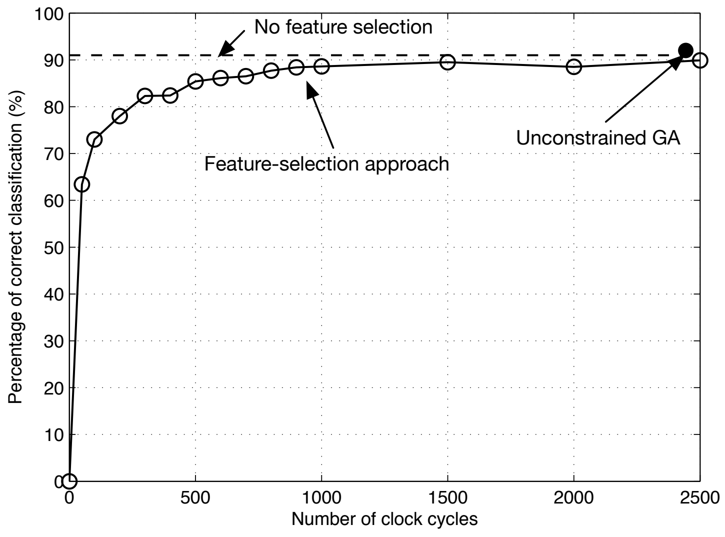


Figure 4.2: Mean percentage of correct classification, P_{CC} (%), for distinguishing among speech, music and noise as a function of the maximum study-case number of clock cycles (or, equivalently, computational cost), attained by the feature-selection approach proposed in this thesis and that obtained by means of the “unconstrained GA-based approach” (or in other words, when the approach is *not* constrained to a maximum number of clock cycles). The dashed line corresponds to the result obtained when there is *no feature selection.*

For the sake of clarity, the number of selected features (according to the percentage of correct classification computed over the patterns available in the validation phase), for each maximum study-case computational cost (in number of clock cycles), has been listed in Table 4.12. Having a look at this table, it seems to be clear that, the higher the computational cost becomes, the higher of the number of selected features is.

As succinctly mentioned, for the study-case $\mathcal{C}_{\max} = 900$ clock cycles, the dimension of the feature vector has been found to be $dim(\mathbf{H}) = 50$. By making use of this feature vector, the MSE linear classifier achieves $P_{\mathrm{CC}} \approx 88.5\,\%$. Note that this numerical result is close to the percentage of correct classification reached by using *all* the available features. For illustrative purposes, Table 4.13 summarizes the set of features selected by our feature-selection approach for this particular study-case in which $\mathcal{C}_{\max} = 900$ clock cycles. Please note that it is only shown those features in which any of its statistical measures is selected.

Regarding this table, it is worth mentioning that the low-complexity features, based on spectral centroid have been selected, the same for MFCCs or ΔMFCCs. Intuitively, the features not used are those that are most complex from a practical implementation point of view, such as, for instance, spectral centroid, voice2white or spectral flux. This is because selecting these features generally exceeds the maximum computational cost the algorithm is constrained to. It

Max. computational cost (clock cycles)	**Number of features**	**P_{CC} (%)**	**Computational cost (clock cycles)**
50	6	63.4	42
100	7	73.0	96
200	16	78.0	198
300	17	82.3	300
400	20	82.4	396
500	26	85.4	499
600	33	86.1	600
700	35	86.5	691
800	42	87.7	794
900	50	88.4	876
1000	51	88.6	998
1500	87	89.5	1351
2000	56	88.5	1984
2500	103	89.9	2457
3000	119	90.8	2443

Table 4.12: For each maximum study-case computational cost, this table depicts the mean percentage of correct classification achieved for distinguishing among speech, music and noise, and the number of selected features. Note that it is also shown the computational cost finally needed to implement the selected feature vector **H** for each study-case (column labelled "Computational cost").

is interesting to note that the low-complexity features based on voice2white or spectral flux have not been chosen.

For the sake of clarity, Figure 4.3 illustrates the mean percentage of use (%) of the features arranged in the set $\mathcal{F}$, for the different maximum study-case computational cost scenarios considered in this section. These results correspond to the average of 20 iterations. The features, illustrated in the picture, are ordered as follows (from the left side to the right one): SC, $\widehat{\text{SC}}$, $\widetilde{\text{SC}}$, V2W, $\widehat{\text{V2W}}$, $\widetilde{\text{V2W}}$, SF, $\widehat{\text{SF}}$, STE, $[1, \ldots, 20]$ MFCCs and $[1, \ldots, 20]$ ΔMFCCs.

As clearly shown, for scenarios in which the maximum computational cost is relatively low, MFCCs are usually selected, in particular MFCC_1, MFCC_2, MFCC_3, MFCC_4 and MFCC_{17}, whereas, the less-used features are those which require high computational cost for its implementation in the DSP, such as, for instance, spectral centroid, voice2white or spectral flux. As the maximum computational cost becomes higher, the selection process favors those features that provide a higher discrimination capability at the expense of requiring higher computational cost. In this respect, for situations in which the maximum computational cost ranges from 500 to 1000 clock cycles, the five new, low-complexity features proposed in this thesis appear to be very adequate for the problem at hand because they are selected in most of the repetitions. The key point in these new features is that they require lower computational cost than the original ones and, for that reason, they are selected instead of spectral centroid, voice2white or spectral flux. Finally, for the scenario in which the maximum computational cost is equal or greater than 2000 clock cycles, the features selected are spectral centroid, voice2white, spectral flux, the five new, low-complexity features, most of MFCCs and ΔMFCC_1, ΔMFCC_4, ΔMFCC_{10}, ΔMFCC_{15} and ΔMFCC_{20}.

Feature	**Statistic**					
	mean	var	std	range	MSB mean	MSB var
$\widehat{SC}$						X
$\widetilde{SC}$						X
MFCC$_1$					X	
MFCC$_2$		X	X			
MFCC$_3$	X	X				X
MFCC$_4$			X			
MFCC$_6$		X				X
MFCC$_7$						X
MFCC$_{10}$			X	X	X	
MFCC$_{11}$				X		
MFCC$_{12}$			X			
MFCC$_{13}$					X	
MFCC$_{14}$	X					X
MFCC$_{15}$	X					X
MFCC$_{16}$		X	X	X		X
MFCC$_{17}$		X			X	
MFCC$_{19}$	X				X	X
ΔMFCC$_2$	-		X		-	
ΔMFCC$_3$	-	X			-	
ΔMFCC$_4$	-		X		-	
ΔMFCC$_7$	-		X		-	
ΔMFCC$_9$	-	X			-	X
ΔMFCC$_{10}$	-				-	X
ΔMFCC$_{12}$	-		X		-	
ΔMFCC$_{13}$	-				-	X
ΔMFCC$_{14}$	-	X	X		-	X
ΔMFCC$_{15}$	-				-	X
ΔMFCC$_{16}$	-		X		-	
ΔMFCC$_{17}$	-	X	X		-	
ΔMFCC$_{19}$	-	X			-	
ΔMFCC$_{20}$	-		X		-	X

Table 4.13: Summary of the statistical measures selected by the feature-selection approach, for each feature arranged in the set $\mathcal{F}$, for the particular case of maximum computational cost equal to 900 clock cycles. Please note that it is only shown those features in which any of its statistical measures has been selected.

4.4 Conclusions

Since DSP-based hearing aids have very strong constraints in terms of computational capacity, memory and battery and taking into account that the feature extraction process can be the most time-consuming task in the classification system, the reduction of the number of features used to automatically classify the sound signals in hearing aid applications becomes a challenging topic. In the effort of reducing the computational cost associated with this process, we have explored:

- The design of a set of new, low-complexity features based on a variation of some spectral features widely-used in the literature (more specifically, spectral centroid, voice2white and spectral flux). These low-complexity features aims at achieving similar or even better classification results than those obtained with original ones the added bonus of demanding

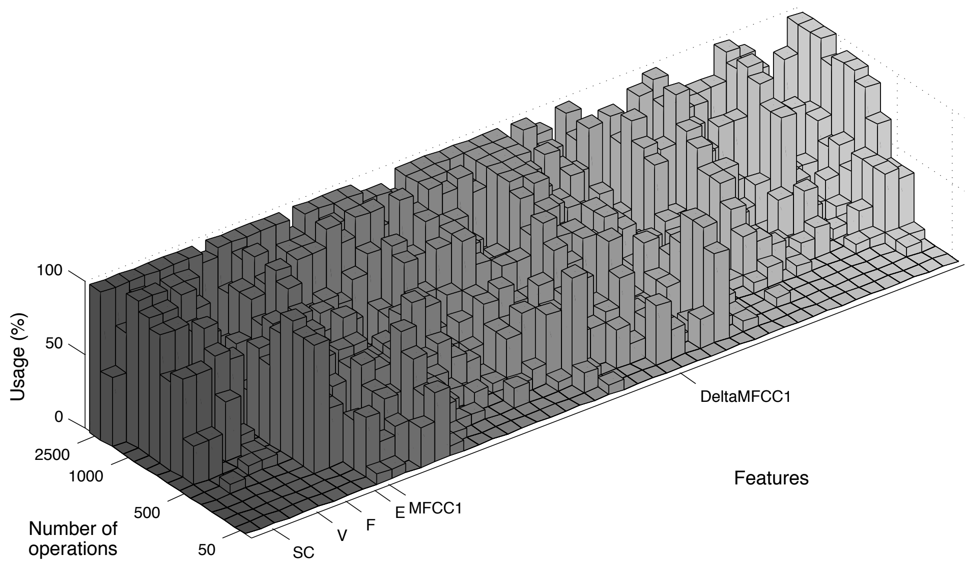


Figure 4.3: Illustrative histogram showing the mean percentage of use (%) of the features arranged in the set $\mathcal{F}$, for the different maximum study-case computational cost scenarios considered. The features, illustrated in the picture, are ordered as follows (from the left side to the right one): SC, $\widehat{\text{SC}}$, $\widetilde{\text{SC}}$, V2W, $\widehat{\text{V2W}}$, $\widetilde{\text{V2W}}$, SF, $\widehat{\text{SF}}$, STE, $[1, \dots, 20]$ MFCCs and $[1, \dots, 20]$ ΔMFCCs.

lower computational cost than that required by the original ones.

- The benefits of a feature-selection approach, based on a genetic algorithm, that aims at selecting the "best suited" feature vector for a particular classification task. "Best suited" here means those feature vector trials that make the classifier work as accurately as possible, along with demanding lower computational cost for their programming in the DSP.

Regarding the low-complexity features, it is important to note that they assist the classifier in discriminating the input sound signal with the same accuracy than that reached when using the original features. This can be noticed simply by noting that when using the original feature set **Set 1** $= \{\text{SC}, \text{V2W}, \text{SF}, \text{STE}\}$, the classifier achieves $P_{\text{CC}} = 66.1\,\%$, whereas when using the low-complexity feature set **Set 2** $= \{\widehat{\text{SC}}, \widehat{\text{V2W}}, \widehat{\text{SF}}, \text{STE}\}$, the classifier reaches $P_{\text{CC}} = 65.4\,\%$ and finally, when using the complete set of low-complexity features, or equivalently, **Set 3** $= \{\widehat{\text{SC}}, \widetilde{\text{SC}}, \widehat{\text{V2W}}, \widetilde{\text{V2W}}, \widehat{\text{SF}}, \text{STE}\}$, the classifier achieves $P_{\text{CC}} = 67.5\,\%$. These results are very promising in the sense that the low-complexity features, proposed in this thesis, exhibit very good performance, with the added bonus of saving a great amount of DSP computational resources.

With respect to the feature-selection approach proposed here, it basically searches into a complete feature set, labelled $\mathcal{F}$, that within our stated framework contains a total of 254 features, and selects a subset $\mathcal{H} \subseteq \mathcal{F}$ constrained by the following conditions: 1) $card(\mathcal{H}) < card(\mathcal{F})$, $card()$ denoting the cardinality of the considered set, 2) the classifier accuracy is maximized, or in other words, maximizes the percentage of correct classification and 3) the computational cost required to compute the feature vector $\mathbf{H}$ is minimized. The experimental work carried out in this thesis show that, implementing all the features included in the set $\mathcal{F}$, assists the classifier in reaching $P_{\text{CC}} = 91.1\,\%$. The "unconstrained GA-based approach", or in other words, when the feature-selection approach is *not* constrained to a maximum number of clock cycles (or, equivalently, computational cost) selects a subset $\mathcal{H}$, which for the best realization results in containing 109 features. By making use of these features, the classifier achieves $P_{\text{CC}} \approx 91\,\%$. Finally, the feature-selection approach proposed here, with only 50 features, assists the classifier in reaching $P_{\text{CC}} \approx 88.5\,\%$. This basically means that using more features does not cause a substantial improvement of percentage of correct classification, though it would require larger amount of computational power.

The final, global conclusion is that the feature-selection approach proposed in this chapter will assist us in selecting the best suited feature set for being programmed in the hearing aid. As will be shown in the next chapters, this feature set will basically consist of most of "low-complexity" features proposed here because of their good classification performance, along with the fact of demanding a lower number of clock cycles for their implementation in the DSP used by our platform to carry out the experiments.

Chapter 5

Implementation and optimization of classifying algorithms

5.1 Introduction

As advanced in Chapter 3, the cornerstone idea describing the classifier in any automatic sound classification system is to discriminate the input sound signal, into one of the output audio classes considered (speech, music and noise, in our case), by means of a set of sound signal-describing features extracted from the input sound.

Many different classifiers have been proposed in the literature and several of them have been applied to the problem of classifying sounds [Kates, 2008]. Bearing in mind that the implementation of signal processing techniques in digital hearing aids is severely constrained by the inherent limitations of the digital signal processor hearing aids are based on, the drawback regarding these classifiers consists in the fact that programming any of these complex algorithms in DSP-based hearing aids is a challenge goal, because it requires to make use of the scarce computational resources of the DSP. Thus, *this forces us to design classification algorithms so that they require a number of operations per seconds as small as possible to reduce power consumption and thus extend the battery life.*

With this in mind, we have evaluated in this thesis a variety of well-known classifiers that exhibit good performance in sound classification in hearing aids. These classifiers have been described in-depth in Chapter 3 and include:

- The mean square error (MSE) linear classifier.
- The k-nearest neighbor (k-NN) classifier.
- Two particular kinds of neural networks:
  - Multilayer perceptrons (MLPs).
  - Radial basis function (RBF) networks.

In the effort of achieving good classification results, we have explored in this thesis three different classification approaches, which are descried below:

- A simple one-layer system in which each study-case three-classes classifier is evaluated separately.

- A divide-and-conquer strategy that leads to a classification system composed of two layers that make use of more specialized binary classifiers. The input sound signal is discriminated by the first layer into either signals containing speech (in a quiet environment or in the presence of music/noise) or signals containing only music or noise and these latter signals are ulteriorly classified more specifically in a second layer depending on whether the hearing aid user is in a music environment or in a noisy one.

- A separation system based on the combination of multiple classifiers. Instead of relying on the decision made by a single classifier, within this classification system, the decision is made by combining, using a standard method, like, for example, the majority rule, the decision of multiple classifiers.

As will be pointed out in the results obtained in this chapter, the approach that reaches the best results, in terms of percentage of correct classification, is the one that makes use of a *single* classifier. To be more precise, an RBF network with a large number of hidden neurons is the classification approach that best performs for the problem at hand. However, from a practical implementation point of view, this number of hidden neurons is extremely large. Just in this respect, and having look at the results obtained, it is interesting to note that MLPs exhibit slightly worse classification performance than that reached by RBF networks, but, this is the crucial point, they demand a much lower number of hidden neurons, which involves that the computational cost is lessened. This is basically the *motivation* that has compelled us to choose an MLP as the classifier to be implemented in the DSP used in our platform to carry out the experiments.

In the effort of helping the classification system achieve accurate enough results by using an MLP as classifier, and aiming at reducing the number of instructions per seconds demanded by the DSP for its implementation, we have also focused on exploring 1) the use of growing and pruning algorithms aiming at determining the most appropriate number of hidden neurons in MLPs and 2) the effects of simplifying the original logarithmic sigmoid activation function used in hidden and outputs neurons by using some kind of piecewise linear approximation.

With this in mind, this chapter is composed of four sections. Section 5.2 illustrates the classification performance of the study-case classifying algorithms when using the abovementioned three classification approaches. In particular, Subsection 5.2.2 describes the database and protocol used for the batches of experiments and Subsection 5.2.3 shows the results obtained with the different approaches. Section 5.3 describes the batches of experiments carried out aiming at reducing the computational cost associated to the implementation of MLPs in the DSP. Being more precise, Subsection 5.3.2 aims at determining the appropriate number of hidden neurons in MLPs for a particular classification task, whereas Subsection 5.3.3 intends for simplifying the activation function used in the hidden and output neurons of the MLP. Finally, Section 5.4 presents the conclusions obtained throughout this chapter.

5.2 Comparative analysis

5.2.1 Introduction

As mentioned in the Introduction, the study-case classifiers are the mean square error (MSE) linear classifier, the k-nearest neighbor (k-NN) classifier and two particular kinds of neural networks: multilayer perceptrons (MLPs) and radial basis function (RBF) networks. This section aims at evaluating the performance of each study-case classifying algorithm when discriminating among speech, music and noise in hearing aids. Apart from evaluating these classifiers separately, we have explored two more classification approaches consisting in the combination of the mentioned classifiers in an effort of improving the classification performance.

Prior to the description of the batches of experiments carried out and the discussion of the corresponding results (Subsection 5.2.3), it is worth having a brief look at the sound database and features used for the experiments, along with the normalization strategy that can be used for normalizing the feature vector in the aim of improving the percentage of correct classification (Subsection 5.2.2).

5.2.2 Description of the experimental work

In order to carry out the batches of experiments, we have made use of the sound database described in Section C.1 in Appendix C (page 211). Regarding the features that feed the study-case classifying algorithms, we list below the particular classical, spectral features that we have extracted:

- Spectral centroid
- Voice2white
- Spectral flux
- Short time energy
- 5 Mel-frequency cepstral coefficients

or, equivalently, the feature set that we have extracted is defined as follows:

$$\mathcal{F} = \{\mathrm{SC}, \mathrm{V2W}, \mathrm{SF}, \mathrm{STE}, \mathrm{MFCC}_1, \mathrm{MFCC}_2, \mathrm{MFCC}_3, \mathrm{MFCC}_4, \mathrm{MFCC}_5\}. \tag{5.1}$$

Perhaps the reader may wonder why we have selected the above listed features and why we have not considered all the features described in Subsections 3.3.2 and 4.2.1 aiming at evaluating the performance of the study-case classifying algorithms. Addressing this question requires to mention that considering all the features listed in the mentioned subsections would require large training time. In the effort of reducing this computation time and clearly exploring the efficiency of the classifying algorithms implemented in this thesis, we have selected a subset of them that provide a high discrimination capability for the classification problem at hand, along with medium computational cost, as shown in Subsection 4.3.4.2 (page 76). Please note that the priority of this chapter is not to propose these features as the best ones for the problems considered in this chapter, but to efficiently evaluate the classifying algorithms proposed in this thesis.

As stated in Chapter 3, the features are computed every 20 ms in our platform, and, at the end, the classifier makes a decision every 2.5 seconds. In order to make this decision, and for the sake of simplicity, we have completed the *statistical* characterization of the random vector that contains the above listed features computed every 20 ms by estimating its mean value and variance. With this in mind, the dimension of the resulting feature vector, labelled $\mathbf{F}$, that feeds the classifying algorithm, is found to be $dim(\mathbf{F}) = L = 2 \times 9 = 18$.

In the effort of removing "biases" associated with different scaled features and preserving good numerical behavior, we have used the so-called "zero-mean and unit-variance normalization" strategy. It basically aims at normalizing the individual elements of the extracted feature vectors in such a way that the resulting (normalized) vectors are better suited for sound classification. "Better suited" here means that the normalized feature vectors make the classifier work better than with non-normalized feature vectors. The particular way this feature normalization technique works is easier to understand if we make use of the conceptual representation we have illustrated in Figure 5.1.

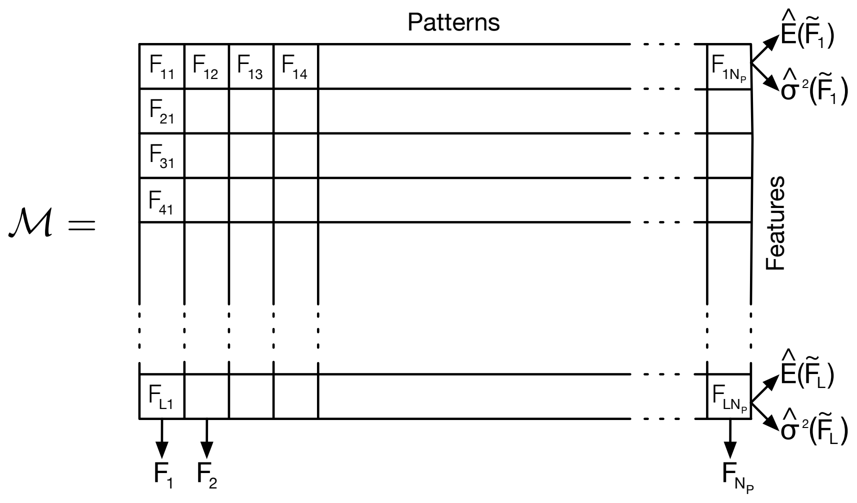


Figure 5.1: Conceptual representation of the way the "zero-mean and unit-variance normalization" strategy works. It aims at normalizing the individual elements of the extracted feature vectors in such a way that the resulting (normalized) feature vectors are better suited for sound classification.

In this figure, the matrix $\mathcal{M}$ contains the feature vectors extracted from the N_{P} patterns available in a *design* phase, labelled $[\mathbf{F}_1, \ldots, \mathbf{F}_{N_{\mathrm{P}}}]$. Please note that: *a*) the number of rows in matrix $\mathcal{M}$ is $dim(\mathbf{F}) = L$ and *b*) the number of columns in matrix $\mathcal{M}$ is $Card(\mathcal{P}) = N_{\mathrm{P}}$. Thus $\mathcal{M}$ is a $L \times N_{\mathrm{P}}$ matrix. Mathematically, the zero-mean and unit-variance normalization technique consists in:

$$F^*_{ln} = \frac{F_{ln} - \widehat{E}(\widetilde{\mathbf{F}}_{\mathbf{l}})}{\widehat{\sigma}^2(\widetilde{\mathbf{F}}_{\mathbf{l}})} \qquad n = 1, 2, \ldots, N_{\mathrm{P}} \qquad l = 1, 2, \ldots, L \tag{5.2}$$

where:

- F^*_{ln} is the "normalized" element of the feature vector,
- F_{ln} is the "non-normalized" element of the feature vector,
- $\widehat{E}(\widetilde{\mathbf{F}}_{\mathbf{l}})$ is the mean of the feature $\widetilde{\mathbf{F}}_{\mathbf{l}}$, being $l = 1, \ldots L$ the index over the L features, and finally,
- $\widehat{\sigma}^2(\widetilde{\mathbf{F}}_{\mathbf{l}})$ is the variance of the feature $\widetilde{\mathbf{F}}_{\mathbf{l}}$.

Please note in Expression 5.2 that the vector $\widetilde{\mathbf{F}}$ is note *actually* the feature vector ($\mathbf{F} = [F_1, \ldots, F_L]$) extracted when applying the data process described in Section 4.2.2 in Chapter 4 (page 61) on each of the available patterns in the *design* phase: the elements composing the l-th row in matrix $\mathcal{M}$ are arranged in the vector $\widetilde{\mathbf{F}}_{\mathbf{l}}$.

Obviously, the resulting vector $\widetilde{\mathbf{F}}^*_l \equiv [F^*_{l1}, \ldots, F^*_{lN_{\mathrm{P}}}]$, where $l = 1, \ldots, L$ is the index over the L extracted features, ends in having zero mean ($\widehat{E}(\widetilde{\mathbf{F}}^*_{\mathbf{l}}) = 0$) and standard deviation equal to one

$(\widehat{\sigma}^2(\widetilde{\mathbf{F}}^*_{\mathbf{l}}) = 1)$. It is important to highlight that the values of $\widehat{E}(\widetilde{F}_l)$ and $\widehat{\sigma}^2(\widetilde{F}_l)$ computed for the patterns available in the *design* phase are also used to normalize the patterns available in both *validation* and *test* phases. Note that the patterns available in the *design* phase are unknown *a priori*.

With these considerations in mind and for the sake of generality, we have explored here three different separation systems, these being:

- A simple one-layer three-classes system in which the input sound signal is classified into speech, music and noise (Subsection 5.2.3.1). In this subsection, we have carried out experiments with both normalized and non-normalized feature vectors. As will be shown, the fact of normalizing the feature vector has been found to provide very good results in terms of percentage of correct classification than those reached when the feature vector is not normalized. These results have led us to normalize the feature vectors for the remaining two separation systems explored in this chapter.

- In the effort of working more efficiently, the second approach consists in dividing the classifying system into two layers that make use of two more specialized binary classifiers (Subsection 5.2.3.2). With this scheme in mind, the input sound signal is discriminated by the first layer into either signals containing speech (in a quiet environment or in the presence of music/noise) and signals containing music (vocal and instrumental) or noise and these latter signals are ulteriorly classified more specifically in a second layer depending on whether the user is in a music environment ("music" class) or in a noisy ambient ("noise" class).

- The last separation system is based on the combination of multiple classifiers in the aim of exploring whether this approach works better than a single classifier (Subsection 5.2.3.3). Within this framework, the global decision is made by combining, using a standard method, like, for example, the majority rule, the individual decisions of R classifiers.

Completing this description of the batches of experiments demands to mention that all the MLPs experiments have been carried out using the MATLAB's Neural Network Toolbox [Demuth and Beale, 1993], and they have been trained using the Levenberg-Marquardt algorithm with Bayesian regularization techniques. The layer's weights and biases are initialized according to the Nguyen-Widrow initialization algorithm [Nguyen and Widrow, 1990]. Any of neural networks experiments have been independently trained 20 times and the best realization in terms of percentage of correct classification computed over the patterns available in a *validation* phase has been selected. The results we have illustrated in the next subsection correspond to the *test* set.

5.2.3 Numerical results

5.2.3.1 One-layer structure

The first batch of experiments has explored the performance of the study-case classifying algorithms in discriminating sounds among the three mentioned classes: speech, music, and noise, as clearly represented in Figure 5.2. It is important to note that, within this framework, not only does the class labelled "speech" refer to speech-in-quiet files, but also to speech-in-noise and speech-in-music files.

In this batch of experiments we have put into practice, it is worth mentioning three comments:

- k-NN experiments have explored a value of k ranging from 1 to 20.

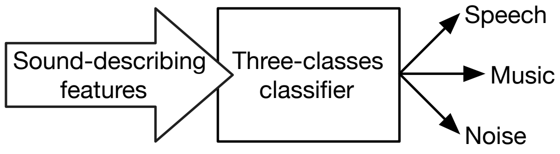


Figure 5.2: Illustrative representation of a one-layer classifying system containing three classes: speech, music and noise. Sound-describing features should assist the system in classifying input sound signals into any of the three illustrated classes. Please note that not only does the class labelled "speech" refer to speech-in-quiet files, but also to speech-in-noise and speech-in-music files.

- MLPs experiments have explored a number of hidden neurons ranging from $M = 1$ to $M = 40$. A superior number of hidden neurons has been found to be unfeasible because of the greater associated computational cost.
- RBF networks experiments have explored a number of hidden neurons that ranges from $M = \frac{1}{15} \cdot N_{\mathrm{P}}$ to $M = \frac{10}{15} \cdot N_{\mathrm{P}}$, in steps of $\frac{1}{15} \cdot N_{\mathrm{P}}$, N_{P} being the number of patterns available in the *design* phase [Gil-Pita, 2006].

Table 5.1 shows the results in terms of mean percentage of correct classification, $P_{\mathrm{CC}}\,(\%)$, achieved, respectively, by the corresponding study-case classifying algorithms. Note that it is also shown the results reached by both "normalized" and "non-normalized" feature vectors. As previously mentioned, in this design process, the "adequate" value of k in k-NN experiments or the number of hidden neurons (M) in MLPs and RBF networks experiments is precisely the one corresponding to that which exhibits the best realization in terms of percentage of correct classification computed over the patterns belonging to the *validation* set, respectively.

Normalized feature vector		**Non-normalized feature vector**	
Classifier	$P_{\mathrm{CC}}\,(\%)$	**Classifier**	$P_{\mathrm{CC}}\,(\%)$
MSE	77.1	MSE	77.1
4-NN	81.0	13-NN	75.5
MLP 23	86.4	MLP 13	85.9
RBF network 388	87.2	RBF network 1164	81.0

Table 5.1: Mean percentage of correct classification, $P_{\mathrm{CC}}\,(\%)$, for the speech/music/noise classification task obtained by the study-case classifying algorithms: mean square error (MSE) linear classifier, k-nearest neighbor (k-NN) algorithm, and two particular kinds of neural networks: a multilayer perceptron (MLP) and a radial basis function (RBF) network. k-NN simply means that k neighbors are used to assign the class to a new sound signal and MLP M or RBF network M simply means that the multilayer perceptron or radial basis function network contains M hidden neurons.

A detailed observation of Table 5.1 leads to the following key conclusions:

- The fact of normalizing the feature vector makes any of the study-case classifying algorithms achieve equal or better results than those obtained when the feature vector is *not* normalized.

- The use of neural networks make the classifying system achieve better classification results than those obtained with the other algorithms. In particular, the best result is achieved by using an RBF network since it assists the system in reaching $P_{\text{CC}} = 87.2\,\%$ (with $M = 388$ hidden neurons).

- From a practical implementation point of view, it is important to highlight that a similar result than that achieved by the sound classifier based on an RBF network can be also achieved by a classifier based on an MLP. To be more precise, an MLP with $M = 23$ hidden neurons (*reference* MLP 23, henceforth) reaches $P_{\text{CC}} = 86.4\,\%$. This result, slightly lower than the previous one, exhibits the added bonus of demanding a lower number of hidden neurons and thus involves lower computational cost.

5.2.3.2 Two-layer structure

In the first batch of experiments we have explored the performance of using a *single* classifier to distinguish among three classes: speech, music and noise, but this approach has some disadvantages for the particular application at hand. To better understand this, it is convenient to imagine the two following situations that we have considered as being representative enough of the problem at hand:

- Situation 1: The hearing aid user is in an acoustical environment consisting in speech in a quiet situation, such as, for instance, at a conference. If the system erroneously classifies the speech-in-quiet signal as noise, then the hearing aid will likely decide that it is not worth amplifying such a signal, and thus will reduce the gain. The consequence is that the hearing-impaired person will loose all the information contained in the speech fragment. On the other hand, if the systems classifies the sound as music, the signal will be amplified but the speech intelligibility perceived by the user will be degraded because, instead of amplifying mid and high frequencies as desired with speech-in-quiet signals, the hearing aid will strongly amplify the low-frequency signal components. These situations degrade the usability of hearing aids.

- Situation 2: The hearing aid user is now embedded in a noisy place like, for example, a traffic jam or a crowded cafe. If the system wrongly classifies the sound as speech, then the hearing aid will decide that it is worth amplifying the signal. The immediate consequence is that the user hears a sudden, unpleasant and irritating amplified noise. This degrades the comfort of hearing aids.

Since the two abovementioned scenarios clearly exhibit the *crucial* task of discriminating between signals containing speech (in a quiet environment or in the presence of music/noise) and signals containing only music (vocal or instrumental) or only noise in hearing aids, the question arising here is: *Is a two-layer structure composed of two binary classifiers more successful than using a single classifier for the speech/music/noise classification task?*

With this in mind, we have explored in this second batch of experiments the performance of an approach based on a divide-and-conquer strategy that leads to a two-layer structure comprising two more specialized binary classifiers [Alexandre et al., 2006a,b]. Figure 5.3 will assist us in better explaining this approach. As depicted, the first classifier aims to distinguish between speech signals (in a quiet environment or in the presence of music/noise) and signals containing only noise or music (vocal or instrumental), while the second one would ulteriorly classify more specifically the latter signals depending on whether the hearing aid user is in a music environment or in a noisy ambient. Please note that each one of the two classifiers may be based on a different algorithm, and both will be separately trained.

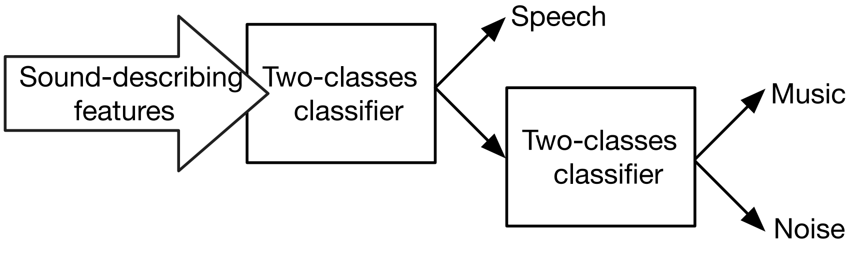


Figure 5.3: Illustrative representation of a two-layer structure comprising two binary classifiers for discriminating among three classes: speech, music and noise. Following a divide-and-conquer strategy, a preliminary classification is made to distinguish between signals containing speech (in a quiet environment or in the presence of music/noise) and signals containing only music (vocal or non-vocal) or only noise. The second layer classifies more specifically the latter signals depending on whether the hearing aid user is in a music environment or in a noisy ambient. Note that each one of the two classifiers may be based on a different algorithm, and both will be separately trained.

Making use of this structure, Table 5.2 shows the results in terms of percentage of correct classification, P_{CC} (%), achieved, respectively, by the global approach when the corresponding study-case classifying algorithms are used in each layer. Please note that, according to the results obtained in the first batch of experiments, these experiments have been carried out by using normalized feature vectors.

First layer		**Second layer**		**Total**
Classifier	P_{CC} (%)	**Classifier**	P_{CC} (%)	P_{CC} (%)
MSE	87.4	MSE	84.0	78.0
8-NN	89.0	4-NN	86.7	80.3
MLP 8	91.8	MLP 20	89.5	85.4
RBF network 388	92.1	RBF network 388	86.9	84.0

Table 5.2: Mean percentage of correct classification, P_{CC} (%), for the speech/non-speech classification task (column labelled "First layer") and for the music/noise classification task (column labelled "Second layer"), obtained by using the study-case classifying algorithms: mean square error (MSE) linear classifier, k-nearest neighbor (k-NN) algorithm, and two particular kinds of neural networks: a multilayer perceptron (MLP) and a radial basis function (RBF) network. It is also shown the total mean percentage of correct classification reached by the global approach that make use of two-layers of two binary classifiers (column labelled "Total P_{CC} (%)"). k-NN simply means that k neighbors are used to assign the class to a new sound signal and MLP M or RBF network M simply means that the multilayer perceptron or radial basis function network contains M hidden neurons.

As illustrated, the best result in terms of global percentage of correct classification is reached by making use of an MLP classifier in each layer, with a global percentage of correct classification equal to $P_{\text{CC}} = 85.4\,\%$. To be more precise, the first layer, which centers on classifying the input signal into either speech or non-speech, by using an MLP 8 ($M = 8$ hidden neurons) returns $P_{\text{CC}} = 91.8\,\%$, whereas the second layer, which classifies non-speech signals between music or noise, making use of an MLP 20 ($M = 20$ hidden neurons) returns $P_{\text{CC}} = 89.5\,\%$.

Thanks to Table 5.2 we can now answer the stated question at the very beginning of this subsection: *Is a two-layer structure composed of two more specialized binary classifiers more successful than a single classifier for distinguishing among speech, music and noise?* Regarding this table and the results obtained in the previous subsection, we can conclude that the use of a two-layer binary classifiers does not improve the performance of a neural network-based sound classifier. Note also that the computational complexity associated to the implementation of a classification system consisting of two classifiers would be higher than that demanded to implement a single classifier.

5.2.3.3 Combination of multiple classifiers

The last batch of experiments has explored the feasibility of using an approach based on the combination of multiple classifiers aiming at exploring whether this classification approach works better than a *single* classifier for the particular application at hand.

The cornerstone concept describing this approach is not relying on the decision made by a single classifier. Instead, multiple classifiers are used to make a global decision by combining their individual decisions [Kittler et al., 1998]. To better explain this, it is convenient to have a look at Figure 5.4, which illustrates the approach proposed in this subsection. As shown, the decision of R classifiers is combined by using a standard method, such as, for instance, the *majority rule* [Franke and Mandler, 1992; Kimura and Shridhar, 1991]. Put it very simple, this method counts the votes for each of the considered classes (speech, music and noise, in our case) over the R classifier outputs and selects the majority class. For the sake of simplicity, we have assumed an *odd* number of classifiers comprising the approach because, otherwise, we would have to implement a rejection rule for an equal number of votes, which is beyond the scope of this thesis.

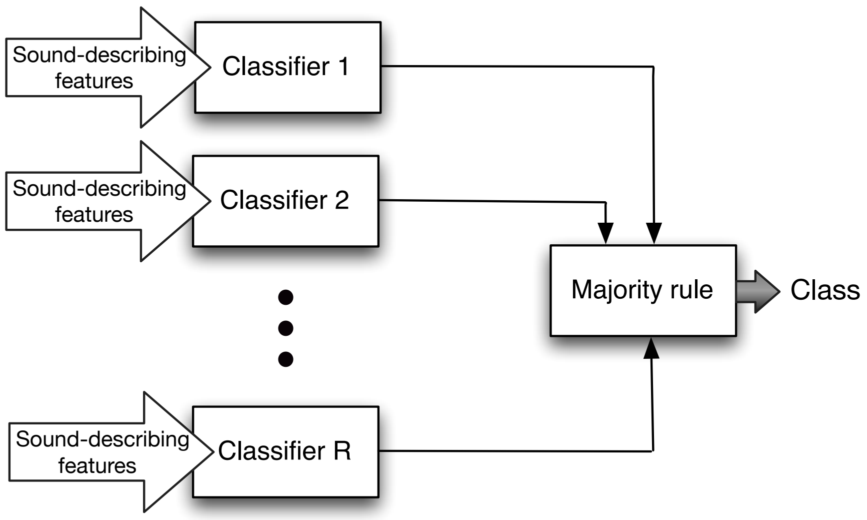

Sound-describing features
Classifier 1
Sound-describing features
Classifier 2
Sound-describing features
Classifier R
Majority rule
Class

Figure 5.4: Illustrative representation of a classification system based on the combination of multiple classifiers. In this approach, the global decision ("Class" labelled in the picture) is made by taking into account the decisions of R classifiers by using the majority rule method. This method counts the votes for each of the considered classes (speech, music and noise, in our case) over the R classifier outputs and selects the majority class.

In the effort of carrying out the experiments, since the number of study-case classifiers evaluated in this chapter is even, being these as follows: 1) MSE linear classifier, 2) k-NN algorithm, 3) an MLP-based classifier and finally, 4) an RBF network-based classifier, it is convenient to discard one of them. For this purpose and regarding the numerical results obtained the two previous subsections, we have excluded the MSE linear classifier because it exhibits the worst performance when compared to the other classifiers. With this in mind, three classifiers ($R = 3$) have been considered in the combination approach. Note that, according to the results obtained in the first batch of experiments, these experiments have been also carried out by using normalized feature vectors.

Making use of this taxonomy and using the classifiers obtained in Subsection 5.2.3.1 (see Table 5.1), the classification system, represented in Figure 5.5, achieves a global mean percentage of correct classification equal to $P_{\mathrm{CC}} = 84.4\,\%$.

Comparing this result with those achieved by using the study-case classifiers separately (see Table 5.1), we can conclude that this approach also performs worse than using a single neural network-based classifier for the particular application at hand. Note that the computational complexity associated to the implementation of this approach would be also higher than that demanded to implement a single classifier.

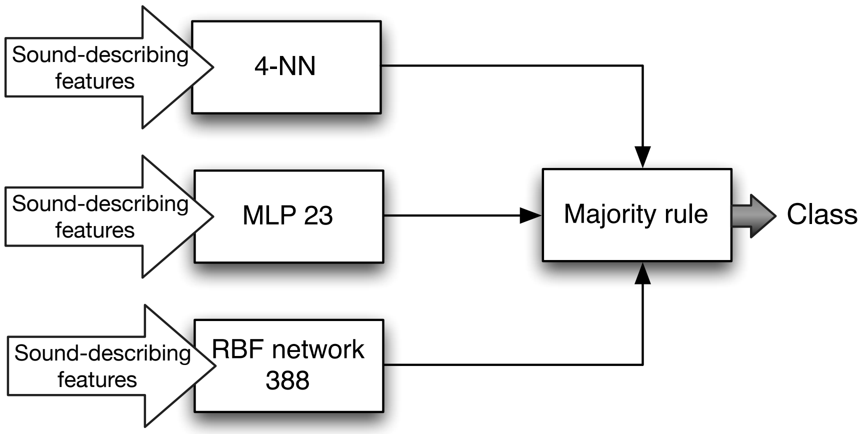

Sound-describing features
4-NN
Sound-describing features
MLP 23
Sound-describing features
RBF network 388
Majority rule
Class

Figure 5.5: Representation of the classification system based on the combination of three particular classifying algorithms ($R = 3$ classifiers), being these individual classifiers those that best perform separately.

5.2.4 Conclusions

Thanks to the results obtained in this section, we can conclude that a single classifier based on a multilayer perceptron is the classifying algorithm that best performs for the problem at hand, as a *balance* between classification performance and computational cost. This is the key reason that has compelled us to choose an MLP-based sound classifying system as the classifier to be implemented in the DSP used to carry out the experiments.

Regrettably, one argument against the feasibility of multilayer perceptrons for being used in DSP-based hearing aids consists in its, *a priori*, high computational complexity, which, among other topics, is related to the network size. However, it is worth exploring its implementation because, as pointed out in Table 5.1, MLPs are able to achieve very good results in terms of percentage of correct classification when compared to other popular algorithms.

We have included the next section in the effort of elucidating how to reduce the computational cost associated to the implementation of an MLP-based sound classifier in the DSP used by our platform to carry out the experiments.

5.3 Optimization of multilayer perceptron parameters

5.3.1 Introduction

Roughly speaking, the complexity of multilayer perceptrons depends on the number of weights that need to be adapted, and consequently, on the number of neurons which compose the neural network. Being more precise, in an MLP architecture, the number of input neurons corresponds to that of the features used to characterize the sound, or in other words, to the dimension of the feature vector, the number of output neurons is related to the number of classes we are interested in, and finally, the number of hidden neurons depends on the adjustment of the complexity of the network [Duda et al., 2001]. If too many hidden neurons are used, the capability to generalize will be poor; on the contrary, if too few hidden neurons are considered, the training data cannot be learned satisfactorily. In this respect and as will be shown throughout this section, selecting the appropriate number of hidden neurons becomes a fundamental issue when designing neural networks.

From this, it can be observed that, on the one hand, one way to decrease the complexity in MLPs is to reduce the number of input features (that is, the dimensionality of the feature vector inputting the network), and consequently, the number of input neurons. Note that this is already achieved by means of the feature-selection method proposed in Section 4.3 in Chapter 4 (page 63), which aims to select, among a set of sound signal-describing features, a subset of them with lower cardinality that assists the classifier in properly discriminating sounds. Note again the key importance of the feature-selection approach proposed in this thesis. On the other hand, another way to reduce the computational complexity would be to decrease the number of neurons in the hidden layer. As the reader might intuitively imagine, this approach is closely related to the selection of the appropriate number of neurons in the hidden layer. This is the approach that we have considered in this thesis in a double effort of *a*) selecting the appropriate number of hidden neurons in MLPs, and *b*) reducing their *presumably* greater associated computational cost.

The question arising here is: *how to obtain good generalization capability with a "small" MLP size*? In this respect, a simple feasible approach to reduce the number of hidden neurons, without degrading the overall network performance, is to make use of growing and pruning algorithms [Mathia, 2004; Pearce, 2001]. Growing algorithms progressively construct the MLP by adding hidden neurons and retraining the network until a minimum of the validation error is achieved, whereas pruning algorithms start with an extremely large network and progressively remove unnecessary weights (or equivalently, neurons) according to their contribution to the overall performance of the network. The following subsection will assist us in elucidating the growing and pruning algorithms proposed in this thesis, along with the results obtained when making use of them.

5.3.2 Appropriate number of hidden neurons

This subsection aims at determining the appropriate number of hidden neurons in a multilayer perceptron. To achieve this, we have made use of growing and pruning algorithms, along with the combination of both algorithms. In particular, Subsection 5.3.2.1 depicts the growing algorithm proposed in this thesis, along with the results obtained in two different scenarios in which this growing algorithm has been used. Subsection 5.3.2.2 describes in detail the three pruning algorithms proposed in this thesis, inspired by a pruning algorithm proposed in the literature. This subsection also illustrates the results obtained when using these pruning algorithms and they are compared to those obtained when using a pruning algorithm proposed in the literature. The last subsection, Subsection 5.3.2.3, shows the results obtained when combining both approaches, that is, growing and pruning algorithms.

5.3.2.1 Growing algorithm

We have implemented a growing algorithm inspired by the one proposed in [Mathia, 2004]. Aiming at clearly explaining how this algorithm works, it is convenient to have a look at Figure 5.6.

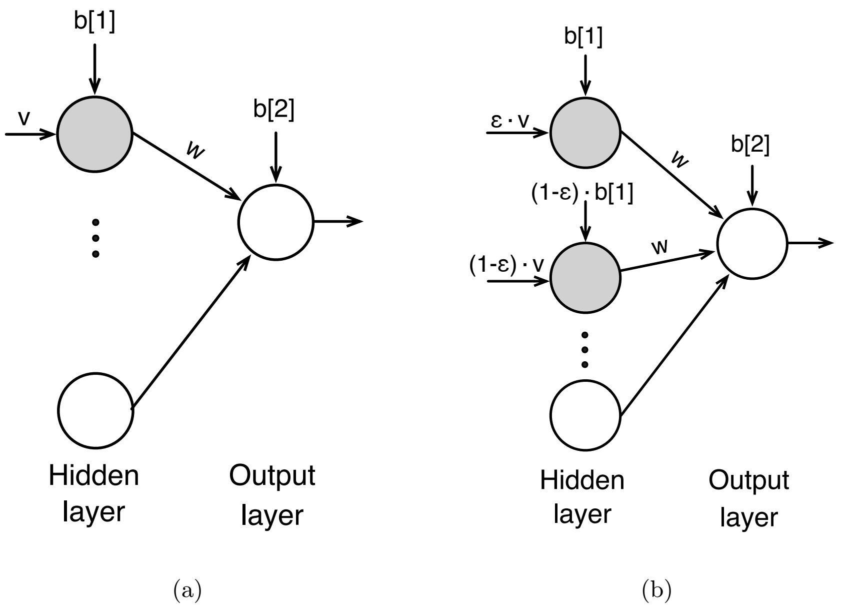


Figure 5.6: A schematic representation of the growing algorithm proposed in this thesis, which aims at determining the appropriate number of hidden neurons in MLPs, or in other words, the minimum complexity MLP that best performs for a given problem. In 5.6(a), the shaded neuron is considered as the “parent” neuron and its bias ($b[1]$) and input weights (v) are divided to obtain the bias and input weight of the new neuron, as shown in Figure 5.6(b). The output weights of the parent neuron (w) are copied to the new neuron.

When a new hidden neuron must be added, the following process is carried out:

1. The neuron with the highest weight at its output is selected. This will be considered as the “parent” neuron. Let it be the shaded one in Figure 5.6(a).

2. The bias ($b[1]$) and input weights (v) of the parent neuron are divided to obtain the bias and input weights of the new neuron, as represented in Figure 5.6(b). A factor of ξ, with $0 < \xi < 1$ is used.

3. The output weights of the parent neuron (w) are copied to the new neuron.

While the algorithm proposed in [Mathia, 2004] uses a fixed value of ξ, in this thesis we have decided to modify this algorithm by selecting, for each iteration, a random value of ξ. Our experiments show that with this modification, the results obtained by the growing algorithm are better in terms of percentage of correct classification.

Aiming at studying the performance of this growing algorithm in the aim of constructing an MLP, we have considered two different scenarios in the experiments. In the first scenario, the growing algorithm basically consists in:

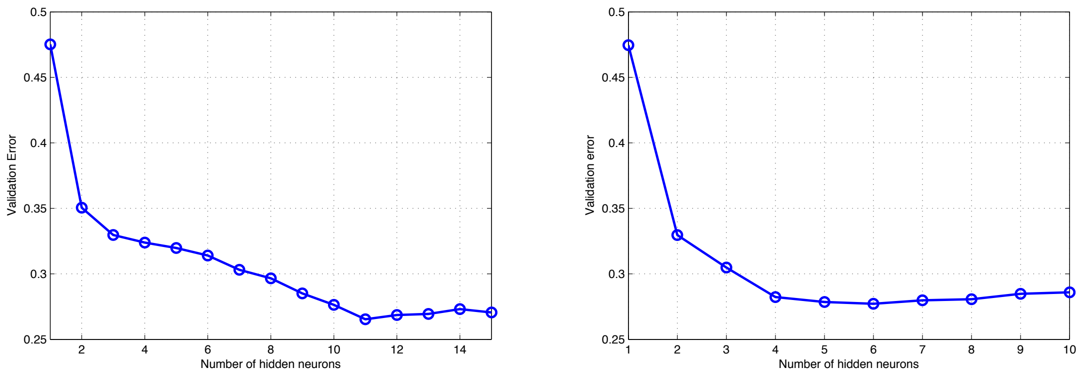
1. Starting with a small multilayer perceptron. In particular, the size of this initial MLP is one neuron in the hidden layer ($M = 1$).
2. Training this MLP by using the Levenberg-Marquardt algorithm with Bayesian regularization techniques.
3. Adding a new hidden neuron according to the growing criterion previously explained and retraining the MLP.
4. Calculating the MSE between the output of the network for the patterns available in a *validation* phase and their desired output.
5. This approach iterates steps $3 - 4$ until the lowest MSE value is reached for the available validation-patterns.

This experiment has been repeated 20 times and the best realization in terms of MSE value computed over the patterns available in the *validation* phase has been selected.

In a second scenario, the growing approach consists in:

1. Creating 20 initial MLPs containing one layer of one hidden neuron ($M = 1$).
2. Training each of the 20 MLPs by using the Levenberg-Marquardt algorithm with Bayesian regularization techniques.
3. Once trained, the MLP that exhibits the lowest MSE value computed over the patterns available in the *validation* phase is selected.
4. Adding a new hidden neuron to the selected MLP according to the growing criterion explained in this section and retraining the MLP.
5. Repeating this training process 20 times and the MLP that, once trained, exhibits the lowest MSE value computed over the available validation-patterns is selected.
6. This approach iterates steps $4 - 5$ until the lowest MSE value is reached for the available training-patterns.

Figure 5.7 shows the behavior of these two abovementioned growing approaches. In particular, it is shown the MSE value for the patterns available in the validation set as a function of the number of the hidden neurons reached by the two growing scenarios proposed in this thesis. In the first scenario (Figure 5.7(a)), the growing algorithm "estimates" that the minimum-complexity MLP that best performs for the problem at hand contains one layer of $M = 11$ hidden neurons, allowing the classifier to achieve $P_{\text{CC}} = 85.2\,\%$. In the second scenario, the growing approach "decides" that the minimum-complexity MLP that best performs contains one layer of $M = 6$ hidden neurons. Making use of this structure, the classifier achieves $P_{\text{CC}} \approx 85\,\%$. These results are slightly lower than that reached by the *reference* MLP 23, which allowed the classifier to achieve $P_{\text{CC}} = 86.4\,\%$ (see Table 5.1), but, this is the key point, when using the growing algorithm proposed in this thesis, especially making use of the second scenario, the number of hidden neurons has dramatically decreased when compared to the *reference* MLP 23 what, in turns, means that the computational complexity demanded to implement the MLP constructed by the growing algorithm in the DSP is lower.



(a) (b)

Figure 5.7: MSE value computed over the patterns available in the validation set, as a function of the number of hidden neurons, reached by two different growing scenarios. In the first scenario (Figure 5.7(a)), the approach "estimates" that the MLP that best performs for the problem at hand contains one layer of $M = 11$ hidden neurons, allowing the classifier to achieve $P_{\mathrm{CC}} = 85.2\,\%$. In the second scenario (Figure 5.7(b)), the growing approach "decides" that the MLP that best performs contains one layer of $M = 6$ hidden neurons, allowing the classifier to achieve $P_{\mathrm{CC}} \approx 85\,\%$.

5.3.2.2 Pruning algorithms

We have designed three different pruning algorithms based on the one proposed in [Mathia, 2004] ("Mathia algorithm", henceforth). The key difference between our algorithms and Mathia algorithm is basically the *stopping criterion* for the pruning process. In the effort of facilitating the reader's comprehension, it is worth mentioning that the stopping criterion assists us in knowing when the pruning algorithm must be finished, or in other words, in finding the minimum-complexity MLP that best performs for a given problem.

Whereas in Mathia algorithm, the stopping criterion only depends on the MSE value between the output of the network for the available training-patterns and their desired outputs, we have proposed here a new criterion for our pruning algorithms that not only does it depend on that MSE value between the outputs of the network and the desired outputs, but also on two more parameters: 1) the number of hidden neurons (M) and 2) a factor α, with $0 < \alpha < 1$. As will be shown later on, our experiments show that by using this new criterion, the results achieved are better than those obtained when using the stopping criterion proposed in [Mathia, 2004].

Once we have mentioned the key difference between our pruning algorithms and "Mathia algorithm", the question arising here is: *what is the difference between the three pruning algorithms proposed in this thesis*? The answer to this question is related to the way in which the *least* significant hidden neuron is selected to be pruned. In this respect, two different approaches, based on a set of statistical strategies and techniques, have been proposed in this thesis to determine which neurons are the least significant. With these approaches in mind, it is very intuitive to note that two of our three pruning algorithms make use of these approaches, along with the stopping criterion proposed in this thesis. Finally, the remaining pruning algorithm uses the same approach as that proposed in [Mathia, 2004] to determine the least significant neuron, but with the difference that it employs the stopping criterion proposed in this thesis.

In the effort of better understanding these algorithms, they will be explained in a detailed way in the paragraphs that immediately follows.

Mathia algorithm

This algorithm is proposed in [Mathia, 2004] and has been used in this thesis for the purpose of comparing the performance of our pruning algorithms. It basically operates in the following way:

1. Creating an initial multilayer perceptron, containing one layer with a large number of hidden neurons.
2. Training this initial MLP by using the Levenberg-Marquardt algorithm with Bayesian regularization techniques.
3. Once trained, the hidden neuron which its sum of the output weights is the *smallest* one is selected to be pruned and thus its weights are eliminated.
4. The rest of weights are retrained by using the Levenberg-Marquardt algorithm with Bayesian regularization.
5. Calculating the MSE value between the outputs of the network and the correct outputs for the available training-patterns.
6. This process iterates steps $3 - 5$ until the lowest MSE value is reached for the available validation-patterns.

Pruning algorithm #1

The cornerstone idea describing this algorithm is similar to the one depicting the Mathia algorithm, being the key difference the stopping criterion used. As in Mathia algorithm, the *least significant* hidden neuron is the one which its sum of the output weights is the smallest one. This neuron is thus pruned and the pruning process is repeated while maintaining a continuous decreasing function. For this purpose, we have implemented in this thesis a novel "cost function" that we have labelled it C, which adopts the following formulation:

$$C = E_{\text{training}} + \alpha \cdot M \tag{5.3}$$

where:

- $E_{training}$ labels the MSE value between the classifier output and the correct output for the patterns available in a *design* phase,
- α, which varies from 0 to 1, is a parameter that needs to be optimized for each particular problem, and finally,
- M labels the number of hidden neurons.

With this in mind, pruning algorithm #1 works as follows:

1. Creating an initial multilayer perceptron, containing one layer with a large number of hidden neurons.

2. Training this initial MLP by using the Levenberg-Marquardt algorithm with Bayesian regularization techniques.
3. Once trained, the hidden neuron which its sum of the output weights is the *smallest* one is selected to be pruned and thus its weights are eliminated.
4. The rest of weights are retrained by using the Levenberg-Marquardt algorithm with Bayesian regularization.
5. The cost function is evaluated and the value obtained is stored.
6. This process iterates steps $3-5$ until the lowest cost function value is reached for the available validation-patterns.

Pruning algorithm #2

The key concept underlying this algorithm consists in calculating the average statistical measure of the outputs of hidden neurons in the aim of estimating what neurons are contributing *less* to the overall network performance, or equivalently, to find the least significant hidden neurons. To be more precise, the following sequence of operations is carried out:

1. Creating an initial multilayer perceptron, containing one layer with a large number of hidden neurons.
2. Training this initial MLP by using the Levenberg-Marquardt algorithm with Bayesian regularization techniques.
3. Once trained, the "average" value of the outputs of each hidden neuron is calculated.
4. In turn, for each hidden neuron, its output weights are set to zero and its corresponding average value of its outputs (previously computed) is added to the output bias.
5. Calculating the MSE value between the outputs of the network and the correct outputs for the available training-patterns.
6. Repeating from steps $3-5$ until *all* the hidden neurons are investigated.
7. Pruning the hidden neuron which obtains the highest MSE value.
8. The rest of weights are retrained by using the Levenberg-Marquardt algorithm with Bayesian regularization.
9. The cost function is evaluated and the value obtained is stored.
10. This process iterates steps $3-9$ until the lowest cost function value is reached for the available validation-patterns.

Pruning algorithm #3

In the same line of reasoning as that in the previous method, the estimation of the least significant hidden neuron here is also based on the calculation of a statistical measure. In particular, this algorithm consists in calculating the standard deviation value of each hidden neuron in order to determine the least significant hidden neuron. This algorithm operates as follows:

Pruning algorithm	**Initial MLP size**	**Final MLP size**	P_{CC} (%)
Mathia algorithm	18-30-3	18-26-3	85.7
Algorithm #1	18-30-3	18-5-3	84.2
Algorithm #2	18-30-3	18-6-3	85.2
Algorithm #3	18-30-3	18-7-3	84.6

Table 5.3: Mean percentage of correct classification, P_{CC} (%), for the speech/music/noise classification task obtained by applying the three pruning algorithms proposed in this thesis to a trained MLP 30 (M=30 hidden neurons). For comparative purposes, it is also shown the results achieved, for the same purpose, by the Mathia algorithm proposed in the literature.

1. Creating an initial multilayer perceptron, containing one layer with a large number of hidden neurons.
2. Training this initial MLP by using the Levenberg-Marquardt algorithm with Bayesian regularization techniques.
3. Once trained, the “standard deviation” of the outputs of each hidden neuron is calculated.
4. The hidden neuron which achieves the lowest standard deviation value is pruned.
5. The rest of weights are retrained by using the Levenberg-Marquardt algorithm with Bayesian regularization.
6. The cost function is evaluated and the value obtained is stored.
7. This process iterates steps $3 - 6$ until the lowest cost function value is reached for the available validation-patterns.

We have explored the feasibility of using any of these pruning algorithms to automatically construct the minimum-complexity MLP that best performs for the problem at hand. For this purpose, we have initialized 20 different MLPs containing one layer of thirty hidden neurons ($M = 30$). These MLPs have been trained by using the Levenberg-Marquardt algorithm with Bayesian regularization techniques and the best realization in terms of MSE value for the available validation-patterns has been selected as the MLP to be pruned. With this in mind, each of the three pruning algorithms proposed in this thesis has been applied to this trained MLP 30. Any of the experiments by using each of the three pruning algorithms has been repeated 20 times and the best realization in terms of MSE value for the available validation-patterns has been selected. Note that, after a number of experiments, the appropriate value of α has been found to be $\alpha = 0.01$.

Table 5.3 illustrates the results obtained for each of the three pruning algorithms. For comparative purposes, it is also shown the results obtained by using the Mathia algorithm. From this comparison, it is worth mentioning three comments:

- The three pruning algorithms proposed in this thesis appear to be robust enough since all the experiments result in finding smaller MLPs without degrading the potential of the classifying algorithm to properly classify among speech, music and noise.

- Mathia algorithm results in finding an MLP that reaches a percentage of correct classification slightly better than those achieved by any of our pruning algorithms, at the expense of demanding significantly higher computational complexity, since the MLP achieved by means of Mathia algorithm contains a higher number of hidden neurons.

- Pruning algorithm #2 has been chosen as the pruning algorithm that best performs for the problem at hand, as a *balance* between computational complexity and performance. Making use of this algorithm, the classifier achieves $P_{\mathrm{CC}} = 85.2\,\%$, with $M = 6$ hidden neurons. Note that this algorithm provides very similar results than those obtained by the growing algorithm, especially in the second growing scenario, in which the appropriate number of hidden neurons was found to be $M = 6$, assisting the classifier in achieving $P_{\mathrm{CC}} \approx 85\,\%$.

Completing this brief section demands to mention that, as happened with the growing algorithm, by using pruning algorithm #2, the result obtained is slightly lower that that reached by the *reference* MLP 23, which allowed the classifier to achieve $P_{\mathrm{CC}} = 86.4\,\%$ (see Table 5.1), but, this is the key point, the number of hidden neurons has dramatically decreased when compared to the *reference* MLP what, in turns, means that the computational complexity demanded to implement the MLP constructed by the pruning algorithm in the DSP is significantly lower.

5.3.2.3 Combination of growing and pruning algorithms

One question that could arise when using growing and pruning algorithms is whether these both methods could be combined in some way that, the global approach resulted in finding a smaller MLP (in terms of network size) that enhanced the classification performance than that achieved by using the algorithms *separately*, with the added bonus of demanding a lower number of hidden neurons. From a practical point of view, making use of the growing approach and pruning algorithm #2 proposed in this thesis, the global design would add hidden neurons to an initial small MLP until the lowest MSE value is reached for the available validation-patterns. Then, the MLP size would be reduced by successively pruning the least significant hidden neurons while maintaining a continuous decreasing function.

To put into practice this global design, we have initialized 20 MLPs containing one hidden layer with one neuron ($M = 1$ hidden neuron). These MLPs have been trained by using the Levenberg-Marquardt algorithm with Bayesian regularization techniques and the best realization in terms of MSE value for the available validation-patterns has been selected. With this in mind, the abovementioned global approach has been applied to this MLP, in which 13 hidden neurons are added in the growing stage, whereas in the pruning stage, it is determined that 7 hidden neurons are good enough for the problem at hand. Making use of this MLP, the classifier achieves $P_{\mathrm{CC}} = 84.6\,\%$. This result clearly shows that the approach of combining both growing and pruning algorithms *does not* provide better results that those obtained by using the growing or pruning algorithms separately.

The performance of this approach can not be illustrated in a figure because the stopping criterion for both growing and pruning algorithms is different. Note that for the growing algorithm, the stopping criterion depends on the MSE value for the available validation-patterns, whereas the stopping criterion for the pruning algorithms is based on the cost function proposed in this thesis that not only does it depend on the MSE value for the available validation-patterns, but also on a parameter α and the number of hidden neurons (M).

5.3.3 Simplification of the activation function

A topic that plays a key role in reducing the computational cost associated to the implementation of an MLP in the DSP is the feasibility of simplifying the activation function used in the MLP.

As mentioned in Subsection 3.4.4.2 (page 48), the activation function used in the hidden and output neurons is the logarithmic sigmoid, or commonly known as “logsig”, which can be calculated by using the following expression (see Table 3.1):

$$f(x) = \frac{1}{1+e^{-x}}\,. \tag{5.4}$$

From the expression above, it is straightforward to note that, from a practical implementation point of view, implementing this function in the DSP is not an easy task, since an exponential and a division need to be computed. Being more precise, the DSP used by our platform to carry out the experiments demands 25 assembler instructions to implement this activation function. In some cases (basically when the number of neurons is high), this number of instructions is too long. This motivates the need for exploring simplifications of this activation function that could provide similar results in terms of percentage of correct classification than those reached by the original function, but, with the additional bonus of demanding lower computational load.

With this in mind, the next subsection has particularly centered on exploring the effects of the simplification of this activation function. In this sense, to what extent an approximation, labelled, for instance, $\widetilde{f}$, is adequate enough is a balance between how well it “fits” the logsig function, labelled f, and the number of instructions the DSP requires to compute $\widetilde{f}$.

5.3.3.1 Piecewise linear approximations

Aiming at finding a suitable enough approximation and taking into account that a typical DSP is able to implement a saturation using one clock cycle, we have evaluated the feasibility of fitting the original activation function f by using two different approximations, which are based on a piecewise linear approximation. In general, the way an approximation, $\widetilde{f}(x, \{\phi_1, \ldots, \phi_{\mathrm{P}}\})$, fits f will depend on a set of design parameters, labelled $\{\phi_1, \ldots, \phi_{\mathrm{P}}\}$, whose optimum values have to be computed by minimizing some kind of error function. In this thesis, we have decided to minimize the root mean square error (RMSE) between the logsig function and the approximation, for input values uniformly distributed from -10 to $+10$, as stated in the following expression:

$$\mathrm{RMSE}(f, \widetilde{f}) = +\sqrt{\widehat{E}\left\{\left(f(x) - \widetilde{f}(x)\right)\right\}}. \tag{5.5}$$

With this in mind, we can proceed further with the approximations proposed in this thesis. In this respect, the first practical implementation for approximating $f(x)$ is based on a 5-piece linear approximation. This approximation, which has been labelled (f_{5PLA}), exhibits the following expression:

$$f_{\mathrm{5PLA}}(x) = \begin{cases} 0, & x < -3.948 \\ 0.0595\,x + 0.2348, & -3.948 \leqslant x < -1.628 \\ 0.2224\,x + 0.5, & -1.628 \leqslant x < 1.628 \\ 0.0595\,x + 0.7652, & 1.628 \leqslant x < 3.948 \\ 1, & x \geqslant 3.948 \end{cases} \tag{5.6}$$

where subscript “5PLA” stands for “5-piece linear approximation”, and the values shown in the expression are the corresponding design parameters, whose optimum values are the ones that optimize the RMSE(f,$f_{5\mathrm{PLA}}$). Note that these values that make the RMSE(f,$f_{5\mathrm{PLA}}$) be minimum ($6 \cdot 10^{-5}$) have been calculated by using "bio-inspired algorithms", or in other words, algorithms that are inspired by nature [Amor, 2008; Huerta et al., 2008].

The practical point to note here regarding this approximation is that the DSP requires, at least, the following 5 assembler instructions for its programming:

1. comparing x with the first threshold value,
2. comparing x with the second threshold value,
3. comparing x with the third threshold value,
4. copying the result of the multiplication into the accumulator,
5. a saturation operation.

Please note that the instructions required to copy x and the constant values into some input registers of the MAC unit have not been considered.

In the same line of reasoning, the second practical implementation, proposed in this thesis, for approximating $f(x)$ is based on a 3-piece linear approximation. This approximation, which has been labelled ($f_{3\mathrm{PLA}}$), exhibits the following expression:

$$f_{3\mathrm{PLA}}(x) = \begin{cases} 0, & x < -2.592 \\ 0.1929\,x + 0.5, & -2.592 \leqslant x < 2.592 \\ 1, & x \geqslant 2.592 \end{cases} \tag{5.7}$$

where subscript "3PLA" stands for "3-piece linear approximation", and the values shown in the expression are the corresponding design parameters, whose optimum values are the ones that make the RMSE(f,$f_{3\mathrm{PLA}}$) be minimum ($5 \cdot 10^{-4}$). Note that these values have been also calculated by using bio-inspired algorithms [Amor, 2008; Huerta et al., 2008].

The practical point to note here regarding this approximation is that the computational load is lessened with respect to that demanded by the 5-piece linear approximation, as it demands the following 4 assembler instructions:

1. comparing x with the first threshold value,
2. comparing x with the second threshold value,
3. copying the result of the multiplication into the accumulator,
4. a saturation operation.

Please note that the instructions required to copy x and the constant values into some input registers of the MAC unit have not been considered.

In order to simplify the calculation of these piecewise approximations, or in other words, to reduce even more the number of instructions demanded by the DSP for their programming, we have tested a third approach, based on the so-called *hard-limit* function, which could also be seen as a 2-piece linear approximation. This approximation, which has been labelled ($f_{2\mathrm{PLA}}$), adopts the following formulation:

$$f_{2\mathrm{PLA}}(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geqslant 0 \end{cases}. \tag{5.8}$$

The additional bonus regarding this approximation consists in that the number of instructions is *drastically* reduced to only 1 assembler instruction.

Completing this description of the approximated activation functions demands to mention that we have explored here the strategy of including a *null* activation function, that, intuitively exhibits the expression stated below:

$$f_{1\text{PLA}}(x) = 0 \qquad \forall \;\; x. \tag{5.9}$$

For the sake of clarity when discussing the results achieved with the different approximations explored in this chapter, this null activation function has been named ($f_{1\text{PLA}}$). We mention this approximation here for reasons that will be better understood later on, and that we postpone to the following paragraphs when using the approach for selecting the most appropriate approximated activation function.

For illustrative purposes, we complete this description of the approximated activation functions by having a look at Figure 5.8. It represents the 2 piecewise linear approximations proposed in this thesis: the 5-piece linear approximation ($f_{5\text{PLA}}$) shown in Figure 5.8(a), and the 3-piece linear approximation ($f_{3\text{PLA}}$) shown in Figure 5.8(b). In Figure 5.8(c), it is depicted the so-called "hard-limit" function ($f_{2\text{PLA}}$) also considered in this thesis, and finally, Figure 5.8(d) depicts the *null* activation function ($f_{1\text{PLA}}$).

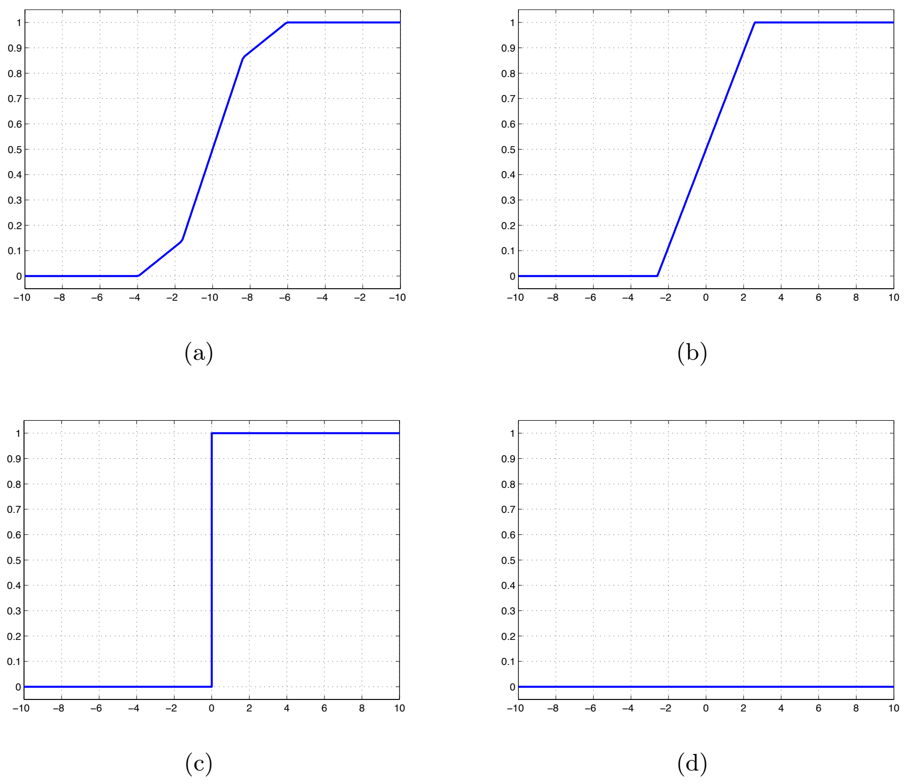


Figure 5.8: Representation of the "approximated" activations functions, proposed in this thesis, for the logarithmic sigmoid (commonly named "logsig") activation function: the 5-piece linear approximation ($f_{5\text{PLA}}$) in 5.8(a) and the 3-piece linear approximation ($f_{3\text{PLA}}$) in 5.8(b). In 5.8(c), it is shown the so-called "hard-limit" function ($f_{2\text{PLA}}$) also considered in this thesis and finally, in 5.8(d) it is depicted the *null* activation function ($f_{1\text{PLA}}$).

In the effort of exploring the effects of simplifying the logsig activation function in the performance of an MLP, we have considered the following scenario: the *reference* MLP 23, which allowed the classifier to achieve $P_{\mathrm{CC}} = 86.4\,\%$ (see Table 5.1), has been evaluated by using, for each of the hidden and output neurons, the approximated activation functions illustrated in Figure 5.8. The purpose of this set of experiments consists in quantitatively evaluating how well each piecewise approximation proposed in this thesis "fits" the original *logsig* activation function. Please note that for this first batch of experiments, the null activation function, depicted in Figure 5.8(d), has not been considered because it would clearly result in $P_{\mathrm{CC}} = 0\,\%$.

Table 5.4 will assist us in explaining how well each approximation proposed in this thesis "fits" the original logsig activation function. It represents the mean percentage of correct classification, P_{CC} (%), reached by the *reference* MLP 23 when it makes use of the different approximated functions considered in this thesis, labelled f_{5PLA} (5-piece linear approximation), f_{3PLA} (3-piece linear approximation) and f_{2PLA} (2-piece linear approximation), along with the number of assembler demanded by the DSP to implement each approximated MLP. For comparative purposes, it is also shown the results corresponding to those reached by the "exact" MLP, or in other words, the MLP 23 that employs of the original logsig activation function (*reference* MLP 23).

Activation function	**Assembler instructions**	P_{CC} **(%)**
f_{logsig}	1231	86.4
f_{5PLA}	711	86.3
f_{3PLA}	685	86.8
f_{2PLA}	607	76.0

Table 5.4: Mean percentage of correct classification, P_{CC} (%), and number of assembler instructions required for implementing in the DSP an MLP 23 ($M = 23$ hidden neurons) that makes use of the different "approximated" activation functions proposed in this thesis, which are as follows: 5-piece linear approximation (f_{5PLA}), 3-piece linear approximation (f_{3PLA}) and 2-piece linear approximation (f_{2PLA}). It is also shown the results reached by the MLP 23 that employs the original logarithmic sigmoid activation function (row labelled "f_{logsig}").

A detailed observation of Table 5.4 leads to the following conclusions:

- The 2-piece linear approximation, labelled f_{2PLA}, is the piecewise approximation that makes the MLP demand the lowest number of assembler instructions for its implementation, but at the expense of working worse than the MLP that makes use of the original logsig activation function. This is clearly noted by comparing the mean percentage of correct classification, P_{CC} (%), and the number of assembler instructions listed on row "f_{logsig}" (in which the activation function has *not* yet been approximated) with those corresponding to row "f_{2PLA}".

- The 5-piece linear approximation, labelled f_{5PLA}, makes the MLP achieve very similar results to those obtained when the MLP uses the original logsig activation function, but exhibiting clear advantages over the number of assembler instructions required. The way to note this consists in comparing the mean percentage of correct classification, P_{CC} (%), and the number of assembler instructions listed on row "f_{logsig}" with those corresponding to row "f_{5PLA}".

- The use of the 3-piece linear approximation, labelled f_{3PLA}, leads to an average relative increase in the percentage of correct classification of approximately 0.4 %, which does not cause substantial improvement of P_{CC} (%), but, taking into account that the considered approximations for the activation function and the number of neurons composing the network are closely related to the *number of assembler instructions* needed to implement the MLP in the DSP, it is very relevant to notice that the 3-piece linear approximation exhibits slightly better performance to that of the original activation function, with the additional bonus of *requiring about half the number of assembler instructions.* This can be noted by having a look at the mean percentage of correct classification and assembler instructions listed on row "f_{logsig}" with those corresponding to column "f_{3PLA}".

The first key conclusion that can be inferred from the performance of the approximated activation functions in Table 5.4 is that, the use of the 3-piece linear approximation is a suitable way to approach the original *logsig* activation function because it exhibits very similar results to those achieved by the MLP that makes use of the *logsig* function, with the additional bonus of demanding lower computational cost for its implementation.

5.3.3.2 Activation function selection algorithm

In the aim of selecting the most appropriate approximated function for each of the hidden and output neurons, among the abovementioned approximations, we have designed a method that aims at automatically selecting the best suited approximation. To better explain this method, we will make use of Figure 5.9, which will help us introduce this approach. As illustrated, the RMSE block calculates the root mean square error between the MLP outputs ($\hat{o}$) and the desired or correct outputs (o). The lower the error is, the best suited is the approximated function selected for each of the hidden and output neurons in the MLP. Since the network outputs produced by the MLP depend on the activation function in each neuron, the GA looks for, by minimizing the function MSE, the best suited approximated activation function for each of the hidden and outputs neurons in the MLP. For the sake of clarity when discussing the results achieved with the different approaches explored in this thesis, we have labelled this approach "unconstrained GA-based activation function selection approach".

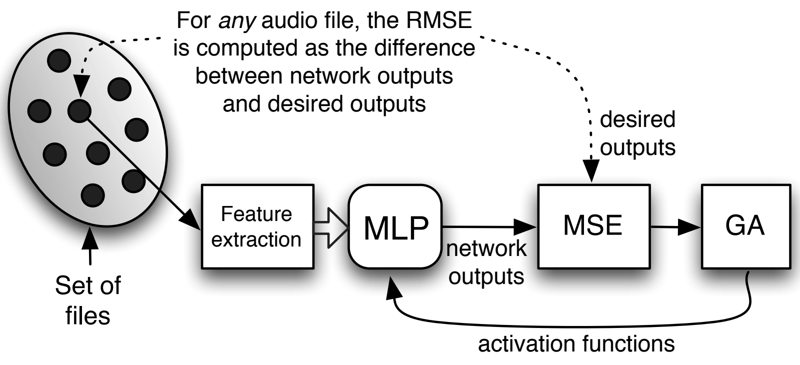

For *any* audio file, the RMSE is computed as the difference between network outputs and desired outputs
desired outputs
Feature extraction
MLP
network outputs
MSE
GA
Set of files
activation functions

Figure 5.9: Simplified block diagram illustrating the way the proposed "unconstrained GA-based activation function selection approach" works. The MSE block computes the mean square error between MLP outputs and desired outputs. Since the MLP outputs depend on the activation functions in the MLP, the genetic algorithm (GA), by minimizing the function MSE, searches for the best activation functions for each neuron composing the MLP.

In the effort of exploring the feasibility of using this method, we have carried out a second batch of experiments. The basic idea underlying this second batch of experiments is to utilize a single GA to compute those approximated activation functions v_q that minimize the function MSE between the MLP outputs ($\widehat{o}$) and the desired outputs (o) (the *objective function* here) for the available validation-patterns in the classification process. With this in mind, any *individual* is thus a coefficient vector with the structure $I \equiv [v_1, v_2, v_3, \ldots, v_{26}]$, where these coefficients may take different integer values, ranging from 0 to 3, corresponding to the following approximated activation functions:

- $0 \rightarrow$ 1-piece linear approximation (f_{1PLA}), or equivalently, the *null* activation function,
- $1 \rightarrow$ 2-piece linear approximation (f_{2PLA}), or in other words, the so-called *hard-limit* function,
- $2 \rightarrow$ 3-piece linear approximation (f_{3PLA}), and,
- $3 \rightarrow$ 5-piece linear approximation (f_{3PLA}).

Please note that the 23 first elements in the aforementioned vector represent the activation function used for each of the hidden neurons, whereas the 3 last elements in the vector label the activation functions employed for the 3 output neurons. The goal of the GA is to find the *best individual* $I_{\mathrm{best}} \equiv [v_{\mathrm{best1}}, v_{\mathrm{best2}}, v_{\mathrm{best3}}, \ldots, v_{\mathrm{best26}}]$ that is "best fitted", or in other words, the one that minimizes the fitness function MSE between MLP outputs and desired outputs (computed over the patterns available in the validation phase). The complete GA operates as listed in Appendix D.

We complete this description by summarizing the main design parameters of the GA in Table 5.5. These values have been found to be large enough to allow the algorithm to properly converge in our experiments.

Parameters	**Value**
Initial population (p_0)	200
Crossover probability (p_c)	0.8
Mutation probability (p_m)	0.1
Max. no. of generations	50
Max. no. of iterations in which the MSE remains unchanged	20

Table 5.5: Summary of the design parameters the GA makes use of for automatically selecting the "best suited" approximated activation functions for each of the hidden and outputs neurons composing the MLP.

Table 5.6 will assist us in explaining the most important results achieved by using the "unconstrained GA-based activation function selection approach". The table represents the mean percentage of correct classification, P_{CC} (%), and the number of assembler instructions required to implement the "exact", *reference* MLP 23, or in other words, the MLP that makes use of the *logsig* activation function and the approximated MLP obtained when applying the above-mentioned approach to the exact MLP 23, row labelled "Approximated MLP" in the table. For comparative purposes, it is also represented the results obtained by the MLP that makes use of the 3-piece linear approximation, row labelled "f_{3PLA} MLP" in the table (see Table 5.4, for further details). As clearly shown, the "unconstrained GA-based activation function selection approach" results in obtaining an approximated MLP that not only does it work better than both "exact" MLP and f_{3PLA} MLP in terms of percentage of correct classification (the approximated

MLP allows our classifier to achieve $P_{\mathrm{CC}} = 87.5\,\%$, but also it requires lower computational cost than any of them. Note that the 1-piece linear approximation, or equivalently, the *null* activation function has been selected for three hidden neurons in the approximated MLP.

	Hidden neurons	**Assembler instructions**	P_{CC} (%)
Exact MLP	23	1231	86.4
f_{3PLA} MLP	23	685	86.8
Approximated MLP	20	659	87.5

Table 5.6: Mean percentage of correct classification, P_{CC} (%), and number of assembler instructions demanded to implement in the DSP the approximated MLP obtained when applying the "unconstrained GA-based activation function selection approach" to an exact MLP. For comparative purposes, it is also shown the results obtained when the exact MLP makes use of the 3-piece linear approximation instead of using the original *logsig* activation function (row labelled "f_{3PLA} MLP").

In the same line of reasoning as that of "unconstrained GA-based activation function approach", the last set of experiments put into practice here explore the feasibility of using an approach, also based on a genetic algorithm, for automatically selecting the "best suited" approximated activation functions for each of the hidden and output neurons in the MLP, when we *restrict the number of assembler instructions allowed.* Bearing in mind that we are constrained to the hardware requirements of the DSP the hearing aid is based on (low clock rates in order to minimize power consumption), we can *realistically* assume that the number of assembler instructions for implementing the activation function that the MLP uses is limited. We have labelled this last approach "constrained GA-based activation function selection approach".

Figure 5.10 shows the mean percentage of correct classification, P_{CC} (%), as a function of the *maximum* number of assembler instructions allowed. There are some issues in the results shown in Figure 5.10 that we would like to stress:

- Intuitively, as the maximum number of assembler instructions allowed increases, the better percentage of correct classification is achieved. In this respect, it is important to highlight that the best result in terms of percentage of correct classification is not achieved for the maximum number of assembler instructions. As shown in the mentioned figure, the best result occurs for 800 assembler instructions (the classifier achieves $P_{\mathrm{CC}} \approx 87\,\%$).

- When the maximum number of assembler instructions is greater than approximately 550, the results in terms of percentage of correct classification *do not* vary significantly. This basically means that using the most complex approximated activation functions for each of the hidden and output neurons *does not* cause a substantial improvement of percentage of correct classification, although it would require larger amount of computational load.

Although this approach does not result in obtaining an approximated MLP that works better than that achieved by the "unconstrained GA-based activation function selection approach", its implementation of the crucial importance because but it allows us to obtain an approximated MLP when the number of assembler instructions is limited.

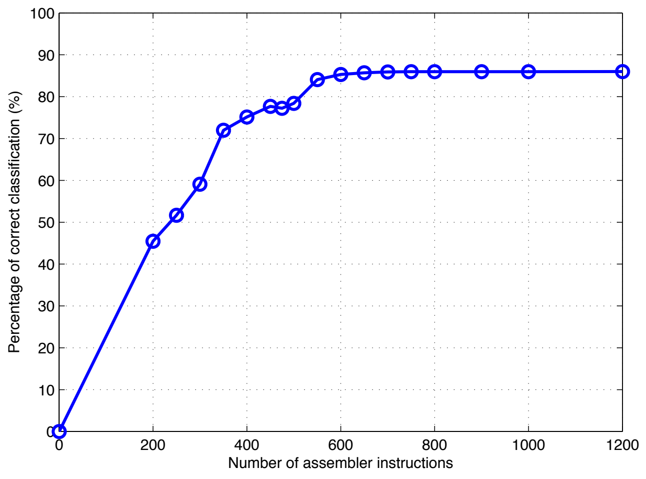


Figure 5.10: Mean percentage of correct classification reached by the "constrained GA-based activation function selection approach", as a function of the number of assembler instructions allowed.

5.3.4 Conclusions

As a conclusion, we can say that, on the one hand, the use of piecewise linear approximations makes the MLPs achieve very similar results than those obtained when using the original *logsig* activation function. In particular, the use of the 3-piece linear approximation leads to an average relative increase in the classification performance when compared to that of the original function. On the other hand, the use of an approach, based on a genetic algorithm, for automatically selecting the "best suited" approximated activation function for each of the hidden and output neurons comprising the MLP, leads to prune "unnecessary neurons", and consequently, to reduce the computational complexity, while maintaining the classification performance.

5.4 Conclusions

This chapter has been motivated by the fact that implementing a classification system in digital hearing aids is strongly constrained by the hardware requirements of the DSP the hearing aid is based on. In this respect, the objective of this chapter has been to evaluate the performance of different sound classifiers, very well-known in hearing aid applications, for the speech/music/noise classification task. These study-case classifiers are as follows: the mean square error (MSE) linear classifier, the k-nearest neighbor (k-NN) algorithm and two particular kinds of neural networks: a multilayer perceptron (MLP) and a radial basis function (RBF) network.

Regarding the classifier performance, it is worth mentioning that it must be a *delicate* balance between keeping the percentage of correct classification within high values (in order to not disturb the user's comfort) and achieving this by using a classifier with low computational complexity. Within this framework, the chapter has particularly centered on exploring the following:

- The performance of the study-case classifiers in discriminating the input sound signal among speech, music and noise. Or in other words, we have explored the feasibility of using each study-case classifying algorithm for discriminating among the aforementioned classes in hearing aids.

- The performance of an approach, based on a divide-and-conquer strategy, which makes use of two more specialized binary classifiers. With this in mind, the input sound signal is discriminated by the first layer into either signals containing speech (in a quiet environment or in the presence of music/noise) and signals containing only noise or music and the latter signals are ulteriorly classified more specifically in a second layer depending on whether the user is in a music environment or in a noisy environment.

- The performance of a separation system based on the combination of multiple classifiers. Within this classification system, the decision is made by combining, using a standard method, like, for example, the majority rule, the decision of multiple classifiers.

According to the results obtained in Section 5.2, aiming at reducing the computational cost demanded for the DSP to implement an MLP-based classifier, we have additionally explored:

- In a double effort of a) selecting the most appropriate number of hidden neurons in MLPs, and b) reducing their *presumably* greater associated computational cost, the feasibility of using growing and pruning algorithms for automatically determining the adequate number of hidden neurons in MLPs, or in other words, the minimum-complexity MLP that best performs for a particular classification task.

- The effects of simplifying the logarithmic sigmoid activation function, used in the hidden and output neurons, in the performance of an MLP. In this respect, we have proposed in this thesis two approaches based on a piecewise linear approximation (a 5-piece linear approximation and a 3-piece linear approximation) for approximating the original activation function. In addition, we have considered the so-called *hard limit* function, which can be also seen as a 2-piece linear approximation, and a *null* activation function that, intuitively, results in zero output values for any input value. In the aim of automatically selecting the "best suited" approximated activation function, among the approximations proposed in this thesis, for each of the hidden and output neurons comprising the MLP, we have designed a method based on an algorithm genetic, which also allows us to prune unnecessary neurons in the MLP.

With this in mind, the different batches of experiments lead to the following conclusions:

- When using a *single* classifier for distinguishing among speech, music and noise, the use of neural networks make the classifying system achieve the best classification results. In particular, the best result is achieved by using an RBF network since it assists the system in reaching $P_{\mathrm{CC}} = 87.2\,\%$ (with $M = 388$ hidden neurons). However, from a practical implementation point of view, it is worth mentioning that by using an MLP 23 ($M = 23$ hidden neurons), the results achieved are very similar than those reached by the RBF network (making use of an MLP 23, the classification system achieves $P_{\mathrm{CC}} = 86.4\,\%$), but, this is the crucial point, with the added advantage of *extremely* reducing the computational cost in the DSP. Note that the number of hidden neurons of the MLP is much lower than that needed for the RBF network.

- The use of a classification system, based on a divide-and-conquer strategy, which makes use of two more specialized binary classifiers, does not improve the percentage of correct classification than that reached by using a single MLP-based classifier. To illustrate this, it is worth mentioning that the best result is achieved by using an MLP 8 in the first layer (speech/non-speech classification task) and an MLP 20 (music/noise classification task) in the second one, being the global percentage of correct classification $P_{\mathrm{CC}} = 85.4\,\%$.

- The strategy of combining multiple classifiers in order to make the global classification decision has been proven to work worse than when using a single classifier. To check this, it is worth mentioning that the combination of three classifiers: a 4-NN ($k = 4$ neighbors), an MLP 23 ($M = 23$ hidden neurons) and an RBF network 388 ($M = 388$ hidden neurons) assist the classification system in achieving $P_{\mathrm{CC}} = 84.4\,\%$.

- The use of growing and pruning algorithms has been proven to achieve reduced MLPs without degrading the classification performance. For the particular application at hand, they result in obtaining an MLP 6 ($M = 6$ hidden neurons) that allows the classification system to achieve $P_{\mathrm{CC}} = 85.2\,\%$. Note that the percentage of correct classification is slightly lower than that reached by the MLP 23, but, this is the key advantage, the number of hidden neurons is lower, and consequently, the computational cost demanded for the DSP for its implementation is also lessened.

- The piecewise linear approximation makes the MLPs achieve very similar results than those obtained when using the original *logsig* funtion. It is worth mentioning that the use of the 3-piece linear approximation leads to an average relative increase in the percentage of correct classification when compared to that of the original function.

- The use of an approach, based on a genetic algorithm, for automatically selecting the “best suited” approximated activation function for each of the hidden and output neurons comprising the MLP, leads to prune “unnecessary neurons” while maintaining the classification performance. In particular, the experimental work shows that the aforementioned MLP 23 can be approximated by an MLP 20 that performs better for the problem at hand (making use of the MLP 20 assists the classification system in achieving $P_{\mathrm{CC}} = 87.5\ \%$) with the added bonus of demanding lower computational cost for its implementation.

The final, global conclusion is that we have chosen, among a number of explored classifiers, a particular kind of neural network as the classifier to be implemented in the DSP the hearing aid is based on. To be more precise, we have chosen a multilayer perceptron (MLP). The variety of approaches explored in this chapter will help us design the MLP-based classifier that will be finally implemented in the DSP.

Chapter 6

Analysis of the effects of finite precision

6.1 Introduction

Most of the algorithms to be programmed in DSPs are commonly designed using "no-quantization", double floating-point precision, which usually offers both a wide range and high precision for each numerical computation. However, for practical hardware implementations, the "final model" barely makes use of "no-quantization", double floating-point precision formats, and consequently, these algorithms need to be "redesigned" for being implemented in some quantized finite-precision format [Fang et al., 2003]. Typically, these final finite-precision formats can be either fixed-point or reduced-precision floating-point (or also known as "lightweight" format [Fang et al., 2002]) formats.

Just in this respect, the most widely-used format employed for programming the algorithms in the DSP is the fixed-point format, in which the integer and fractional parts of a number are represented by using a determined number of bits. The scientific reason for which fixed-point format is the most widely-used one basically relies on the fact that it exhibits three key advantages [Fang et al., 2003]. The first one is that they behave mostly like integers, except for the need to round the fractional parts of the numbers after each mathematical operation. The second benefit is that it is easy to transform one fixed-point format into another format, which uses different number of bits to represent the integer and fractional parts, by means of shifting and zero-padding strategies. As a consequence of this second benefit, the third advantage is that it is possible to assign each number a *unique*, minimal number of bits, and thus minimize the computational complexity.

Turning again our attention to finite-precision formats and within the application at hand, it is worth mentioning that most of actual DSPs for hearing aids make use of a 16-*bit word-length Harvard Architecture*, and only modern hearing instruments have larger internal bit range for number presentation (22-24 bits). In some cases, the use of larger numerical representations is reserved for the filterbank analysis and synthesis stages, or to the Multiplier/ACcumulator that multiplies 16-bit registers, and stores the result in a 40-bit accumulator. In this thesis, since the DSP used to carry out the experiments (see Appendix A for further details) makes use of a 16-bit fixed-point format, we have focused on the case in which we have 16 bits to represent numbers.

Regarding this, the question arising here is: *How are the number of bits used to represent the integer and the fractional part of a number "efficiently" assigned?* Since the number of bits used to represent the integer part and the fractional portion of a number have strong influence on the final performance of the algorithms implemented in the hearing aid, an improper selection of these values can lead to saturations or lack of precision in the operations of the DSP. In this way, adjusting the adequate number of bits used to represent the integer and the fractional part of a number becomes a fundamental topic whose main issues we explore in the following sections.

With these ideas in mind, the chapter focuses on: 1) clearly quantifying the effects of the finite-precision limitations on the performance of an automatic sound classification system for hearing aids, with special emphasis on the effects of finite word length for the sound signal-describing features and weights of an MLP-based classifier, and 2) selecting, among the available 16-bit fixed-point format quantization schemes, the most adequate configuration for the hearing aid performance. The purpose of our study is to demonstrate that a 16-bit numerical representation configured in a proper way can produce good results in the implementation of an MLP-based sound classifier. Please note that we say an MLP-based sound classifier because, as shown in Chapter 5, an MLP-based classifier has been chosen as the classifying algorithm for our practical implementation since it provides very good results when compared to those obtained with other popular classification algorithms.

Finally, completing this section demands to mention that this chapter has been structured as follows. Section 6.2 defines the problem of quantizing the sound signal-features and weights of the MLP for sound classification in hearing aids. Section 6.3 describes the database and the protocol used for the experiments (Subsection 6.3.1) and shows the results obtained (Subsection 6.3.2), along with a discussion in Section 6.4.

6.2 The quantization problem

As mentioned in the Introduction, there is a topic that plays a key role in the performance of the MLP-based sound classification system when it is implemented in a DSP-based hearing aid, and that constitutes the core of this chapter: the effects of finite-precision for the sound signal-describing features and weights of the MLP in the classification system performance.

As stated beforehand, the DSP used by our platform to carry out the experiments makes use of a 16-bit fixed-point format. In this respect, fixed-point numbers are usually represented by using the so-called "Q number format" [Dspfactory, 2002a,b]. Within the application at hand, we have used the notation $Qx.y$, where:

- Q labels the signed fixed-point number is in the "Q format notation",
- x symbolizes the number of bits used to represent the 2's complement of the integer part of the number, and finally,
- y designates the number of bits used to represent the 2's complement of the fractional proportion of such number.

As an illustrative example, using a numerical representation of 16 bits, we could decide to use the quantization scheme $Q16.0$, which is used for representing 16-bit 2's complement integers and 0 bits to represent the 2's complement of the fractional proportion. Similarly, we could use $Q8.8$ quantization, what, in turns, means that 8 bits are used to represent the 2's complement of the integer part of the number, and 8 bits are used to represent the 2's complement of the fractional proportion, or $Q4.12$, which basically assigns 4 bits to the integer part, and 12 bits to the fractional part and so forth.

Apart from this question to be answered later on, there is also a crucial problem related to the small number of bits available to represent the integer and the fractional parts of numbers: the *limited precision*. Although not clear at first glance, it is worth noting that a low number of bits for the integer part may cause the register to saturate, while a low number of bits in the fractional proportion may cause a loss of precision in the number representation. The appropriate number of bits for both representing the integer part and the fractional proportion will be determined in the section that immediately follows.

It is important to highlight that in those modern DSPs that use larger numerical representations the quantization problem is minimized, since there are several configurations that yield very good results.

6.3 Experimental work and results

Prior to the description of the experiments carried out in a double effort of 1) exploring the effects of quantifying both the signal-describing features and the weights of an MLP-based classifier (by using the so-called $Qx.y$ format) and 2) selecting the most adequate 16-bit quantization scheme for hearing aid performance and the discussion of the corresponding results (Section 6.3.2), it is worth having a look at the sound database and features used for the experiments (Section 6.3.1).

6.3.1 Experimental setup

In order to carry out the experiments, we have made use of the sound database described in Section C.1 in Appendix C (page 211).

In the same line of reasoning as that of Chapter 5, the features that we have selected to feed an MLP-based sound classifier are the mean and variance of the following ones:

- Spectral centroid
- Voice2white
- Spectral Flux
- Short time energy
- 5 Mel-frequency cepstral coefficients

or, equivalently, the feature set that we have extracted is defined as follows:

$$\mathcal{F} = \{\mathrm{SC}, \mathrm{V2W}, \mathrm{SF}, \mathrm{STE}, \mathrm{MFCC}_1, \mathrm{MFCC}_2, \mathrm{MFCC}_3, \mathrm{MFCC}_4, \mathrm{MFCC}_5\}. \quad (6.1)$$

In the batches of experiments we have put into practice, the experiments have been repeated 20 times and the best realization in terms of percentage of correct classification computed over the patterns available in a *validation* phase has been selected. The results that we have illustrated below correspond to the *test* set.

6.3.2 Numerical results

In the effort of studying the effects of finite precision, we have considered two different scenarios in the experiments, as clearly shown below:

1. The classifiers were configured for returning a decision every 2.5 seconds, or in other words, for time slots of 2.5 seconds. The aim of this study is to determine the effects of the limited precision over the classifiers (MLPs, in our case) for applications like, for example, automatic program switching, in which a large time scale is used.

	No quant.				Quant. scheme					
Classifier	**Double**	Q1.15	Q2.14	Q3.13	Q4.12	**Q5.11**	Q6.10	Q7.9	Q8.8	Q9.7
MLP 1	**73.4**	31.2	32.1	70.3	73.6	**73.4**	73.2	72.8	63.7	50.1
MLP 2	**77.2**	19.7	50.3	74.3	77.0	**77.1**	77.0	76.8	72.5	49.8
MLP 3	**79.1**	20.4	49.3	78.3	79.1	**79.1**	79.1	79.0	71.9	42.7
MLP 4	**81.6**	18.6	47.2	76.8	81.3	**81.5**	81.4	81.0	71.7	40.9
MLP 5	**82.8**	20.8	47.8	77.2	82.4	**82.6**	82.4	82.5	71.0	46.8
MLP 6	**82.9**	24.9	53.1	79.4	82.1	**82.8**	82.6	82.1	71.3	45.9
MLP 7	**83.2**	25.0	50.6	79.9	83.1	**83.2**	83.1	82.8	70.7	44.7
MLP 8	**83.3**	26.2	55.4	80.2	83.1	**83.2**	83.1	82.9	71.3	42.5
MLP 9	**83.8**	26.8	53.9	81.2	83.7	**83.7**	83.6	82.5	71.1	43.8
MLP 10	**84.1**	25.9	53.8	81.9	83.9	**84.0**	83.8	83.0	71.0	44.2
MLP 15	**84.7**	26.2	54.4	82.0	84.6	**84.7**	84.5	84.0	71.4	47.9
MLP 20	**85.8**	36.9	58.3	81.7	85.6	**85.8**	84.2	84.3	70.9	45.8
MLP 25	**86.3**	28.5	57.2	81.8	86.2	**86.3**	85.9	85.1	71.5	47.9
MLP 30	**85.9**	36.9	54.8	79.9	85.6	**85.8**	85.6	84.9	71.7	46.8
MLP 35	**85.7**	37.3	56.9	79.5	85.6	**85.7**	85.5	76.4	71.5	45.8
MLP 40	**84.9**	36.8	58.8	79.4	84.7	**84.8**	84.5	76.6	71.6	47.4

Table 6.1: Mean percentage of correct classification of different MLPs, returning a decision with time slots of 2.5 seconds, achieved by using 9 quantization schemes: $Qx.y$ represents the quantization scheme with x bits for the integer part, and y bits for the fractional one. MLP M means a multilayer perceptron with M neurons in the hidden layer. The column labelled "Double" corresponds to the percentage of correct classification when no-quantization, double floating-point precision has been used. Columns in bold aims at helping the reader focus on the most relevant result: $Q5.11$ provides very similar results to those of double precision.

2. The classifiers were configured for making a decision with time slots of 20 milliseconds. In this case, the objective is to study the effects of the limited precision in a classification scenario in which a small time scale is required, such as, for instance, in noise reduction or sound source separation applications.

It is important to highlight that in a real-time classification system the classification evidence can be accumulated across the time for achieving higher percentages of correct classification. This fact makes necessary a study of the tradeoff between the selected time scale, the integration of the decision for consecutive time slots, the performance of the final system and the required computational load. This analysis is out of the scope of this chapter, since our aim is not to propose a particular classification system, that must be tuned for the considered hearing aid application, but to illustrate a set of tools and strategies that can be used for determining the way an MLP can be efficiently implemented in real-time for sound environment classification tasks with *limited* computational capabilities.

With these considerations in mind, we proceed further in explaining the results obtained in this chapter. In this respect, Tables 6.1 and 6.2 show the percentage of correct classification for a variety of multilayer perceptrons with different numbers of hidden neurons. In this table, MLP M labels that the corresponding MLP contains M neurons in the hidden layer. These batches of experiments have explored a number of hidden neurons ranging from 1 to 40. Aiming at clearly understanding the effect of different quantization schemes, we have also listed the percentage of correct classification computed with no-quantization, double floating-point precision formats. These have been labelled in Tables 6.1 and 6.2 by using the header "double".

Tables 6.1 and 6.2 supply some important pieces of useful information that we would like to stress:

- The quantization formats with a low number of bits for representing the *integer part*, such as, for instance, $Q2.14$, finally lead to decrease the percentage of correct classification when

	No quant.	Quant. scheme								
Classifier	**Double**	Q1.15	Q2.14	Q3.13	Q4.12	**Q5.11**	Q6.10	Q7.9	Q8.8	Q9.7
MLP 1	**53.5**	46.6	48.2	53.0	53.4	**53.4**	53.1	53.0	49.2	30.0
MLP 2	**62.6**	48.3	55.8	61.6	62.4	**62.6**	62.3	57.5	43.5	29.8
MLP 3	**63.9**	45.8	46.3	53.3	59.8	**63.6**	62.8	53.7	43.7	39.0
MLP 4	**65.3**	40.1	39.4	48.9	59.6	**65.1**	63.9	53.8	39.8	28.7
MLP 5	**67.2**	40.3	44.2	50.9	62.3	**66.9**	62.6	48.8	37.8	25.6
MLP 6	**66.9**	35.9	39.8	44.9	58.8	**65.9**	62.7	56.8	40.4	28.3
MLP 7	**68.3**	37.2	40.9	45.9	61.1	**67.7**	63.6	51.9	39.8	27.2
MLP 8	**69.4**	36.3	38.2	42.4	63.2	**68.8**	64.5	55.2	39.7	29.4
MLP 9	**69.8**	35.8	40.1	42.8	63.8	**68.7**	66.6	53.4	39.9	30.1
MLP 10	**70.1**	33.8	38.9	41.0	62.3	**68.2**	67.1	54.3	40.3	29.6
MLP 15	**70.2**	31.7	34.4	37.8	68.6	**70.0**	69.3	58.6	42.0	27.4
MLP 20	**70.4**	31.5	33.1	35.7	60.9	**70.3**	69.7	59.8	43.2	29.3
MLP 25	**70.5**	31.4	33.0	36.8	60.2	**69.9**	69.8	59.7	40.8	27.0
MLP 30	**70.9**	31.0	33.2	38.1	58.3	**69.8**	69.5	62.1	41.5	29.7
MLP 35	**70.7**	30.8	34.1	37.8	57.4	**69.5**	69.0	63.2	45.8	30.1
MLP 40	**70.4**	32.7	35.8	41.0	61.8	**70.3**	70.2	63.5	44.2	28.9

Table 6.2: Mean percentage of correct classification of different MLPs, returning a decision with time slots of 20 milliseconds, achieved by using 9 quantization schemes: $Qx.y$ represents the quantization scheme with x bits for the integer part, and y for the fractional one. MLP M means a multilayer perceptron with M neurons in the hidden layer. The column labelled "Double" corresponds to the percentage of correct classification when no-quantization, double floating point precision has been used. Columns in bold aims at helping the reader focus on the most relevant result: Q5.11 provides very similar results to those of double precision.

compared to those computed with double-precision. This decrease is caused by saturations of the features and weights of the MLP-based classifier.

- On the other hand, the use of a low number of bits for the *fractional portion* also causes a decrease in the percentage of correct classification, basically arising from the loss of precision in the numerical representation.

These facts illustrate the need for a tradeoff between integer and fractional bits. For the sake of clarity, Figure 6.1 shows the average relative increase in the error percentage with respect to the use of double-precision, as a function of the number of bits of the fractional portion. Computing this relative increase requires the use of those results obtained when using all the MLPs listed in Table 6.1 and 6.2, and the average estimated as depicted in the expression shown below:

$$I = 100 \cdot \widehat{E}\left\{\frac{P_{Qi.j} - P_{\text{double}}}{P_{\text{double}}}\right\} (\%) \tag{6.2}$$

where $\widehat{E}\{\cdot\}$ represents the mean value of the error percentages involving all the number of hidden neurons considered. Having a look at this figure, one can see that the lowest relative increase is achieved by the $Q5.11$ quantization scheme, for both time slot configurations (files of 2.5 seconds and files of 20 ms). This is basically the reason by which we have chosen the $Q5.11$ quantization format for evaluating the effects of quantizing the overall sound classification system prior to its practical implementation in the DSP. Please note also that for this quantization scheme format, the lowest relative increase is achieved for files of 2.5 seconds.

Finally, for properly completing this section and making use of the $Q5.11$ quantization scheme, we have quantified the signal-features and the weights of the MLP 20 obtained in Chapter 5, which assisted the classification system in achieving $P_{\text{CC}} = 87.5\,\%$ (see Table 5.6). After

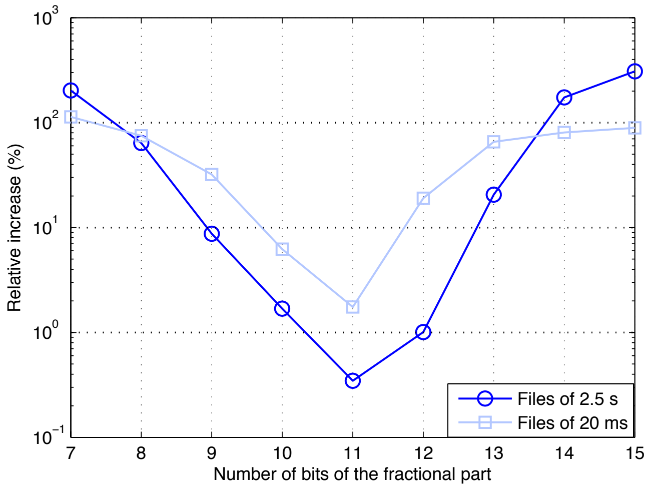


Figure 6.1: Average relative increase (%) in the percentage of error, as a function of the number of bits of the fractional part, for the MLP-based classifiers explored in this chapter. Note that it is shown for both time slot configurations considered: files of 2.5 seconds and files of 20 ms.

quantifying the features and the weights, the percentage of correct classification has been found to be $P_{\mathrm{CC}} = 87.3\,\%$. These results clearly prove that the quantized MLP-based classifier achieves perceptually the same efficiency as that reached by "exact" MLPs (without quantization), or in other words, the performance of the classification system, once it is implemented in the DSP, is not *appreciably* degraded.

6.4 Conclusions

This chapter has been motivated by the fact that the implementation of signal processing algorithms in hearing aids is strongly constrained by the finite precision used for representing the numbers to implement the algorithms in the DSP the hearing is based on. Within this framework, this chapter has particularly centered on exploring the following:

- Studying the effects of the quantization format, $Qx.y$, used for representing both the sound signal-describing features and the weights of the MLP.
- Selecting, among the available 16-bit quantization schemes, the most adequate one for hearing aid performance.

The different batches of experiments, for both 2.5 seconds and 20 milliseconds time slots configurations, lead to the following conclusions.

- The use of time slots of 2.5 seconds has been found to provide better results than those reached by using shorter time slots (20 milliseconds).
- The $Q5.11$ quantization scheme exhibits very similar results to those obtained when no-quantization, double-precision is used, mainly for the case of files of 2.5 seconds.

The final, global conclusion is that the results clearly prove that the quantized MLP-based classifier achieves perceptually the same efficiency as that reached by "exact" MLPs (that is, without these approximations), or in other words, the performance of the classification system, once it is implemented in the DSP, is not *appreciably* degraded.

Chapter 7

Speech enhancement algorithms

7.1 Introduction

As advanced in the introductory chapter, hearing aid users usually face a variety of different acoustic environments in their daily life, such as, for instance, speech, music or noisy environments. It seems to be clear that, among these acoustic listenings, the "noisy environment" is the most difficult one to have a conversation for hearing aid users because of the surrounding background noise.

In the effort of achieving a good communication in noisy environments, the relevant information (the *speech source*) should be extracted from the background noise, which is a significant problem for hearing aid users[1]. Furthermore, it is important to note that the listening environment for hearing aid users can change rapidly, like, for example, from being outdoor (the *noise source* may come from a car passing by) or in a car (the *noise source* may come from engine noise) to entering an office (e. g. fan noise, telephone ringing), a restaurant (e. g. people talking), home (e. g. television, radio, household appliances), church or a concert hall. These effects can seriously degrade the speech intelligibility perceived by hearing aid users[2], especially for those users which require a higher SNR. This enforces engineers to design algorithms that allow hearing aids to compensate for all these effects.

Many algorithms have been proposed in the literature aiming at speech enhancement in hearing aids. Regrettably, the drawback in these approaches consists in the fact that their practical implementation in the DSP usually exceeds its scarce computational resources. To better understand this, it is worth mentioning that only a reduced number of these algorithms have been effectively implemented and integrated in hearing aids, being these hearing aids the most sophisticated (and consequently, the most expensive) devices currently available in the market [Eneman et al., 2008].

With these ideas in mind, we propose in this chapter a different approach aiming at speech enhancement in hearing aids. As will be shown throughout this chapter, this approach basically consists in automatically generating -by making use of perceptual concepts [Zwicker and Fastl, 1999], and a supervised learning process driven by a genetic algorithm- a gain function $\mathfrak{G}$ that aims at enhancing speech quality. Such perceptual concepts take advantage of the particular way the human auditory system works. As it will be illustrated in Section 7.5, the proposed algorithm

[1]As mentioned in Chapter 2, hearing aid users have great difficulty to understand speech when they are immerse in background noise. Many times, the users can "hear" but *not understand* the speech signal. They typically require a SNR of about 5-10 dB higher than that required by normal hearing listeners to achieve the same level of speech understanding [den Bogaert et al., 2009].

[2]Improving the speech intelligibility is one of the most desirable improvements sought by hearing aid users [Kochin, 2002]

creates the gain function $\mathcal{G}$ by using a gaussian mixture model driven by a genetic algorithm. This genetic algorithm computes the optimized parameters this model depends on in order to enhance the speech quality the user perceives. To what extent this is enhanced is measured by the algorithm *itself* by using a scheme based on an *objective* measure. Section 7.2 illustrates the reason why a scheme based on an objective measure has been chosen. In this respect, it is important to highlight that, in many applications, the main purpose of speech enhancement algorithms is to improve speech quality while preserving, at the very last, speech intelligibility. A few number of algorithms have been evaluated by using intelligibility tests [Arehart et al., 2003; Boll, 1979; Lim, 1978; Tsoukalas et al., 1997], and in these studies, only *one* speech enhancement algorithm was evaluated in different noisy environments [Hu and Loizou, 2007]. In an effort of properly validating the approach proposed in this thesis, not only in terms of speech quality, but also in terms of speech intelligibility, we have made use of two different standardized objective measures: the perceptual evaluation of speech quality (PESQ) and the speech intelligibility index (SII), which, as their names suggest, aim to evaluate speech quality and speech intelligibility, respectively. These both standardized measures are described in a detailed way in Section 7.3 and 7.4, respectively. Intuitively, the higher these scores are, the better the speech "perceived" by the user is. Experimental work, included in Section 7.6, suggests the feasibility of the mentioned approach to enhance speech quality and speech intelligibility in hearing aids. Finally, this chapter is completed with a summary and discussion in Section 7.7.

7.2 Measuring the speech quality

Roughly speaking, speech quality measures can be accomplished by using either *subjective* or *objective* algorithms. We say that a measure is considered as objective if it is "computable", or in other words, if a numerical value is returned after some mathematical calculations, in contrast with a subjective measure, which is always based on human perception tests [Halka and Heute, 1992].

Subjective measures usually involve a large number of subjects (typically among 20 and 50 subjects), listening to a live or recorded conversation and assigning a rating to it. This rating can be either a single overall quality quantifying a particular characteristic, like, for example, clarity or listening effort, or a particular distortion, such as, for instance, clipping or hum. Since subjective measures demand to make use of people, these measuring approaches are often very accurate and appropriate aiming to evaluate speech quality. In this respect, the standardized mean-opinion-score, commonly named "MOS", is one of such valuable metrics involved subjective measures [ITU-T, 2001a]. Although the MOS is not the only subjective measure, it is, however, one of the most widely-used and recognized.

In order to calculate the MOS value, a large number of subjects listen to a set of *test* speech sentences, and assign to each sentence an overall quality according to a N-point quality scale (being a typical value $N = 5$). The results of these individual ratings are then averaged to obtain the MOS value, which ultimately ranges from 1 (bad) to 5 (excellent). Concerning this, it is worth mentioning that there are two standardized 5-point quality scales widely-used in the literature [Jayant and Noll, 1990]: 1) a scale for evaluating the signal quality (intuitively, the higher the MOS score is, the better the signal quality is), and 2) a signal impairment scale that measures the quality signal with respect to the audibility of the distortion in the signal. For illustrative purposes, these both 5-point (also called, five-grade) quality scales, along with their associated numerical scores are outlined in Table 7.1.

Clearly, a metric, such as MOS, which makes use of subjects, can be a good measure of the speech quality perceived by the user. However, subjective measures have important *drawbacks*

MOS	Quality	Impairment
5	Excellent	Imperceptible
4	Good	(Just) Perceptible but not annoying
3	Fair	(Perceptible and) Slightly annoying
2	Poor	Annoying (but not Objectionable)
1	Bad	Very annoying (Objectionable)

Table 7.1: Two widely-used standardized 5-point (also called, five-grade) quality scales. One of the scales aims at evaluating the signal quality (column labelled "Quality") and the other scale intends for measuring the quality signal with respect to the audibility of the distortion (column labelled "Impairment"). For both scales, it is shown their associated MOS values, which range from 1 to 5.

that make them difficult to be used in a practical implementation. In particular, these measuring approaches can be both excessively time-consuming and expensive tasks, in the sense that, some researchers or organizations might not have the resources to carry out these particular tests involving large numbers of subjects (as in our case). Furthermore, such metrics cannot be used in any sort of real-time or online application. Please note that this latter drawback is not actually important from the point of view of our approach because, as will be shown throughout this chapter, the optimized parameters of the model the gain is based on are computed *off-line*. Only when these values have been computed on the laboratory, they are stored in the data-memory available in the DSP.

These drawbacks, among other reasons, have led to the development of objective metrics. Ideally, these measures should be as highly correlated as possible to the standard MOS value, or to any another subjective quality measure. In other words, these measures should be good predictors of average subjective preferences.

Several different objective prediction methods for measuring speech quality when transmitting packets over networks, or in the design of algorithms, such as, for instance, those involved in low bit-rate speech coding, have been proposed in the literature [Beerends, 1994; Hansen and Kollmeier, 2000; ITU-R, 1998; ITU-T, 1996a, 2000, 1997, 1996b; Rix and Hollier, 2000; Thiede et al., 2000; Voran, 1999a,b]. Such methods basically predict perceived speech quality based typically on a computation of distortion between the original transmitted signal (also named "clean") and the received signal (sometimes named "noisy").

Among these several objective measures, we have focused here on a model, proposed by the International Telecommunications Union, for the perceptual evaluation of speech quality (PESQ) [ITU-T, 2001b]. It achieves an extremely good performance for a very wide range of applications, such as, for instance, speech quality assessment in speech coding algorithms or in the end-to-end testing of networks. This sort of applications might make the reader wonder why we have explored this model to evaluate the performance of the speech enhancement approach for hearing aids proposed in this chapter. The reason is as follows. Although PESQ measure was not initially designed for the evaluation of speech enhancement algorithms in hearing instruments, it has been recently found to provide a significant good indication of its performance and is commonly used by engineers for evaluating speech quality in these devices [Mancuso et al., 2006; Rohdenburg et al., 2006].

Deepening a little more in this latter measure, it is worth noting that PESQ measure predicts speech quality but not speech intelligibility, which may lead to wonder the following question: *a speech sound that shows excellent speech-quality, is it also perfectly intelligible?* Although not clear at first at glance, it is worth noting that excellent speech-quality *does not* always involve perfect speech-intelligibility [Preminger and van Tasell, 1995]. Aiming at overcoming this, we have also made use here of another standardized measure that intends for measuring speech intelligibility. This measure is the so-called speech intelligibility index (SII) [ANSI, 1997]. It is commonly used to predict speech intelligibility in real-world environments with stationary background noise[1]. The key point that exhibits the greatest relevance here, when contrasted with the PESQ model, is that this measure takes into account the subject's acoustic loss and thus, it can be used for predicting speech intelligibility for both normal-hearing and hearing-impaired subjects.

In the effort of properly evaluating the speech enhancement approach for hearing aids proposed in this chapter, we have made use of the two aforementioned objective measures, since it has been proven that they both show good performance in predicting speech quality and speech intelligibility for different speech quality test databases. These both objective measures will be described in a detailed way in the two sections that immediately follow.

7.3 Perceptual evaluation of speech quality

7.3.1 Introduction

In the early 1990s, several new perceptual models for measuring the quality of speech in speech coding algorithms were proposed. We say *perceptual* in the sense that they apply a psychoacoustic model, or in other words, a mathematical model based on psychoacoustic measurements of the masked threshold[2]. In 1996, the perceptual speech quality measure (PSQM) [Beerends, 1994], was adopted as ITU-T recommendation P.861 [ITU-T, 1996b]. The purpose of this algorithm consisted in quantitatively evaluating the performance of speech coding algorithms. Two years later, an alternative system based on measuring normalizing blocks (MNB) [Voran, 1999a], was added as an appendix to recommendation P.861. Another model, the perceptual audio quality measure (PAQM) [Beerends, 1992], was combined with several different audio models to produce a new algorithm known as perceptual evaluation of audio quality (PEAQ), which became an ITU-R recommendation BS.1387 in 1999[3] [ITU-R, 1998; Thiede et al., 2000]. Some extensions to the bark spectral distortion (BSD) model [Wang et al., 1992] led to the development of the perceptual analysis measurement system (PAMS) [Hollier et al., 1992, 1994; Rix et al., 1999; Rix and Hollier, 2000]. This was the first model in the literature to focus on end-to-end quality speech assessment of networks, including the effects of filtering and variable delay [Rix et al., 1999; Rix and Hollier, 2000].

These effects, along with certain types of coding distortion, packet loss and background noise, were found to cause earlier models, such as, for instance, BSD, PSQM and MNB, to provide inaccurate scores [Rix et al., 1999, 2000; Rix and Hollier, 2000]. A competition was therefore held by ITU-T study group 12 to select a new model with good performance across a wide range of speech coding algorithms and network conditions. In this respect, the two algorithms with the highest performance in this competition, PAMS and PSQM99 (an updated and extended version of PSQM), were combined to produce a new model known as perceptual evaluation of speech quality (PESQ), which became an ITU-R recommendation P.861 [Beerends et al., 2002; ITU-T, 2001b; Rix et al., 2002].

Although will be explained in a detailed way later on, we can say in advance that PESQ algorithm predicts subjective MOS scores by comparing the input signal (called the "reference" signal in the standard) to a "system under test", and the signal that has been transmitted

[1]Stationary noise is defined as noise that does not contain large or rapid changes in its spectrum over time, such as, for instance, the sound of a fan or engine noise, thermal noise, etc.

[2]Auditory masking is a well-known psychoacoustic phenomenon in which a weak signal becomes inaudible in the presence of a stronger masker signal [Moore, 2003].

[3]It was originally released in 1998 and last updated in 2001.

through that system under test (called the "degraded" signal in the standard), and penalizes the final score based on measures of distortion. The PESQ measure has a high correlation with subjective listening tests for a large number of testing conditions [ITU-T, 2001b], which makes it a measure widely-used for evaluating speech quality.

7.3.2 Description of PESQ model

Figure 7.1 shows, in an illustrative way, the diagram block of PESQ algorithm. As clearly depicted, PESQ algorithm requires both a "reference" signal and a "degraded" signal for comparison and a "system under test". Please note that, within the particular application at hand, the system under test is the hearing aid *itself*, and the reference and degraded signals are the original input signal to the hearing aid and that input signal after being processed by the hearing aid, respectively.

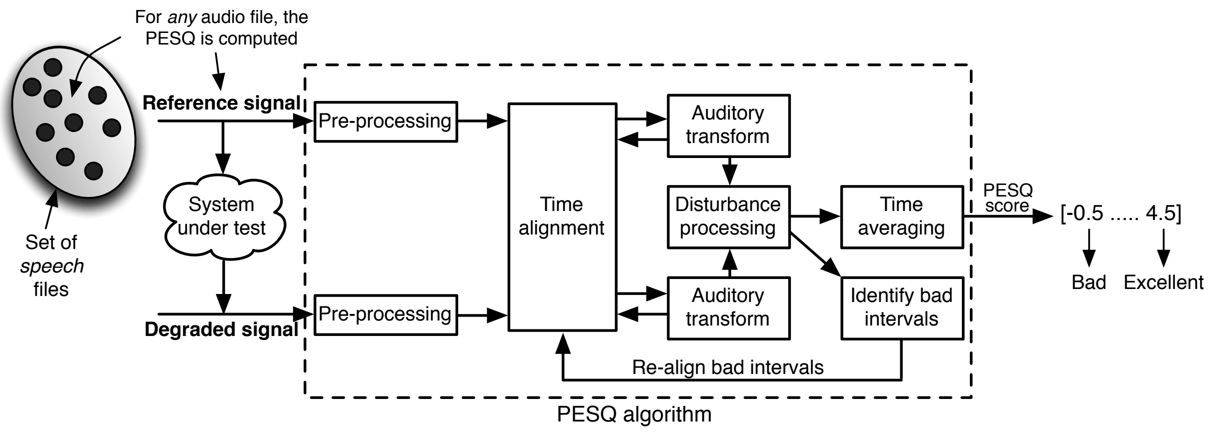


Figure 7.1: Illustrative diagram block of PESQ algorithm. For *any* input speech file, the PESQ block computes, after some mathematical handling, a score ranging from -0.5 (bad, *completely no understanding*) to 4.5 (excellent, *perfect understanding*). This score basically depends on the "system under test", the reference and degraded signals, and the psychoacoustic model applied. Note that, within the particular application at hand, the system under test is the hearing aid *itself*, and the reference and degraded signals are the original input signal to the hearing aid and that input signal after being processed by the hearing aid, respectively.

PESQ algorithm basically consists of different functional blocks, which are briefly summarized in the following paragraphs:

- Pre-processing: The reference and degraded signals are first level-equalized to a standard listening level, and then filtered by using a filter with response similar to a standard telephone handset. Although this filter is band limited to about 3 kHz, however, its use is very appropriate since all the intelligibility of the speech signal is contained within this band.

- Time alignment: The signals are aligned in time in order to correct time delays, which basically allows us to compare corresponding parts between the reference and degraded signals.

- Auditory transform: In the effort of comparing the reference and degraded signals, taking into account of how a listener would have heard them, each sound signal is passed through an auditory transform that imitates certain key properties of human ear[1]. This assists us in obtaining a time-frequency representation of the perceived loudness of the signal (sometimes known as the “sensation surface”).

- Calculation of distortion measures: The difference between the sensation surfaces for the reference and degraded signals is known as the “error surface”, which basically shows any audible differences introduced by the system under test. The error surface is analyzed by a process that takes account of the effect that small distortions in a signal are inaudible in the presence of strong signals (auditory masking).

  From the positive and negative errors, two disturbance parameters are calculated. They are calculated as non-linear averages over specific areas of the error surface. These two disturbance parameters are:

  - The absolute (symmetric) disturbance, or in other words, a measure of absolute audible error.
  - The additive (asymmetric) disturbance, that is, a measure of audible errors that are significantly louder than the reference.

  This analysis allows us to obtain two error parameters that summarize the amount of each type of audible error.

- Realignment of bad intervals: In certain cases, the time alignment may fail to correctly identify a delay change, resulting in large errors for each section with incorrect delay. These are identified by labeling bad frames (which have a symmetric disturbance of more than 45) and joining together bad sections in which bad frames are separated by less than 5 good frames. Each bad section is then realigned, and the disturbance is recalculated, estimating thus a new delay. The auditory transform of the degraded signal is recalculated and the disturbance estimated again. For any frame, if the realignment results obtained lead to a lower disturbance value, the new value is used.

- Mapping to MOS scale: The asymmetric and symmetric distortion measures are mapped in order to produce the prediction of MOS score that, within this context, varies from -0.5 (bad, or equivalently *completely no understanding*) to 4.5 (excellent, or in other words, *perfect understanding*), although for many cases it will be a MOS-like score ranging from 1 (bad) to 4.5 (excellent), as listed in Table 7.2. Intuitively, the higher the PESQ score is, the better the speech “perceived by the human auditory system” is.

7.4 Speech intelligibility index

7.4.1 Introduction

The speech intelligibility index (SII) [ANSI, 1997] is a standardized objective measure commonly used to predict speech intelligibility in stationary noisy environments, like, for example, when speech goes through noisy telecommunication channels or is processed by electronic devices, such as, for instance, hearing aids.

[1]Among other sequence of processes, both signals are divided into frames and the Short-Time-Fourier-Transform (STFT) of any frame, which makes use of a window length of 512 samples and an overlap factor of 0.5, is computed

PESQ score	**Linguistic equivalent**	**Quality degradation**
4.5	Excellent	None
4	Good	
3.5	Good/Fair	Moderate
3	Fair	
2.5	Fair/Poor	Severe
2	Poor	
1	Bad	

Table 7.2: Illustrative representation of PESQ quality scale. Although PESQ score originally ranges from −0.5 (bad, or equivalently *completely no understanding*) to 4.5 (excellent, or in other words, *perfect understanding*), it is a very common situation to *interpret* PESQ scores as MOS-like scores, ranging from 1 (bad) to 4.5 (excellent), as depicted in this table. Note that it is also shown *both* the linguistic equivalent and the quality degradation scales.

It was created as a major revision of the articulation index (AI) [French and Steinberg, 1947]. Basically, the articulation index is a measure that quantifies the relationship between the proportion of the average speech spectrum that remains perceptible in the presence of filtering, background noise or low-level speech. It was originally developed to assist engineers in the design of telephone communication systems but it has been proven recently to be a valuable tool for hearing aids fitting (say, for instance, the design of the best "amplification program" in order to maximize speech intelligibility in different acoustic situations). Like AI and PESQ measures, SII is also a measure highly correlated with the intelligibility of speech under a variety of adverse listening conditions [Azman, 2010].

Being more precise, SII can be interpreted as the total number of speech cues to reach the listener (the hearing aid user, in our application at hand). As depicted in the intelligibility rating scale shown in Figure 7.2, the value of SII is always a number between 0 (bad, or equivalently *completely no understanding*) and 1 (excellent, or in other words, *perfect understanding*). According to [ANSI, 1997], good communication systems have a SII value of 0.75 or above, while poor communication systems have a SII value below 0.45.

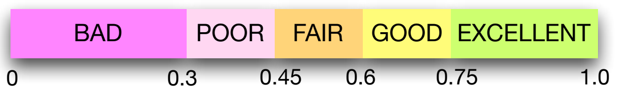


Figure 7.2: Speech intelligibility index (SII) rating scale. SII score is expressed by a number ranging between 0 (bad, or equivalently, *completely no understanding*) and 1 (excellent, or in other words, *perfect understanding*).

As mentioned in the Introduction, the computation of the SII requires, apart from the subject's acoustic loss, that the spectra corresponding to the speech and noisy signals are available separately. The cornerstone idea describing how this method works relies on the fact that the intelligibility of speech depends on the proportion of spectral information that is perceptible to the listener. In this sense, the spectrum is divided into bands (that is, frequency bands) and the average of the SNRs in each frequency band is estimated. The SNRs are then weighted by band-importance functions, or in other words, a numerical value that characterizes the relative significance of each frequency band to the speech intelligibility. Aiming at clearly understanding why *not all* the frequency bands *do* contribute in the same proportion to the speech intelligibility, we have included Figure 7.3 that represents an illustrative example of the "importance" of any particular frequency [ANSI, 1997]. As shown, the contribution of central frequencies is higher than those of lower or higher frequencies.

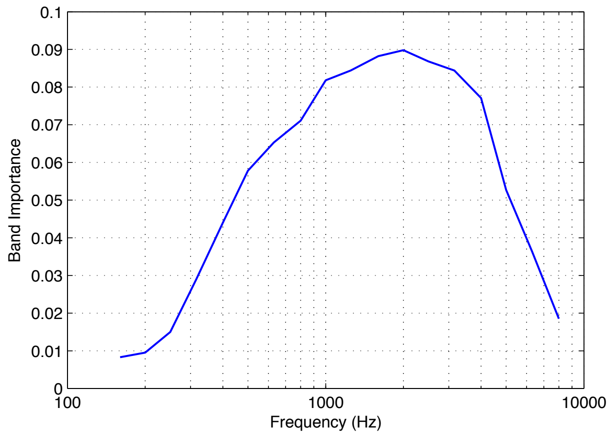


Figure 7.3: "Importance" of the different frequencies aiming to calculate the speech intelligibility index. As represented, the contributions of central frequencies is higher than those of lower or higher frequencies.

According to the number of frequency bands selected to compute the SII, the methods are classified as follows (in descending order of accuracy):

1. Critical bands (21 bands)
2. One-third octave bands (18 bands)
3. Equally-contributing critical bands (17 bands)
4. Octave bands (6 bands)

The basic fundamentals of the four methods are essentially the same. The computations basically differ in the number and the bandwidth of the frequency bands selected. Intuitively, the higher the number of frequency bands is, the more accurate the method is. In this respect, it is very illustrative to note that the "critical bands" method is the most precise of the four listed. Although this is the most accurate method, however, the method we have chosen here to carry out the experiments is the "one-third octave bands" (from 160 Hz to 8 kHz with a bandwidth of 150 Hz), since this approach usually corresponds with speech analysis experiments [Muthukumarasamy, 2009].

The sequence of operations involved in calculating the SII score using this method are analyzed in the following subsection. We have grouped them under the heading "description of SII". For this calculation, it is supposed that, 1) the subject faces the speech source, and 2) the speech source is assumed to be omnidirectional.

7.4.2 Description of SII

In this section, a review of the most important aspects for the calculation of the SII score are described, although a more detailed description of the SII can be found in [ANSI, 1997].

With the "one-third octave bands" method in mind, the SII score is computed based on three parameters that we will describe later on. Such parameters, as will be shown, include some factors we are familiar, like, for example, the particular hearing loss the subject suffers from. Once we have these necessary parameters, the sequence of operations that will help us analytically compute the SII score is explained in the paragraphs that follow.

- The first descriptive parameter is called *speech spectrum level* (E') (dB) and represents, for each frequency band labelled with the k-index, the spectrum level of the speech signal at the centre of subject's head. In the problem at hand, this parameter will be determined as:

$$E_k^{'} = E_k + G_k \tag{7.1}$$

  where:

  - E_k is the speech spectrum level at frequency band k, and
  - G_k is the insertion gain for the frequency band k. In the particular case at hand, this insertion gain designates the gain that the hearing aid *exhibits* to compensate the particular subject's acoustic loss.

- The second necessary parameter to estimate the SII score is the *noise spectrum level* (N_k') that, essentially, measures the spectrum power level of noise in each frequency band. Since SII score has been used to create an amplification scheme, or in other words, a gain function ($\mathcal{G}$) when the input signal to the hearing aid is classified as a "speech-in-quiet" signal, or in other words, speech without presence of background noise, this parameter has been assumed to be 0 dB SPL for all frequency bands in our experiments.

- The third parameter is the *equivalent hearing threshold* (T_k'). Basically, this parameter is the *amount* that a tone must be increased aiming at a hearing-impaired subject can "hear" sounds as a normal-hearing person does, or in other words, this parameter represents *the hearing loss level* (dB) listed Table C.2 in Appendix C (page 213), for the study-case patients used in our experimental work.

Now, with all these parameters we have defined, we have the background to define the SII score as follows:

1. Computing the *equivalent masking spectrum level* ($Z_k^{'}$) (dB)
   This parameter can be seen as the sound pressure level in the spectrum that appropriately accounts for the masking of speech produced by the equivalent noise. It comprises masking from within-band, out-of-band (spread of masking) and masking of one speech frequency by another (known as "self-speech masking"). In the effort of estimating this parameter when the "one-third octave bands" method is chosen, the following parameters need to be computed for each frequency band k:

   - Self-speech masking spectrum level, V_k (dB): This parameter computes the masking of higher speech frequency bands by lower speech frequency bands in cases of severe low-pass or band-pass filtering. It can be calculated by using the following expression:

$$V_k = E_k - 24 \tag{7.2}$$

     being E_k the equivalent speech spectrum level (dB) for the frequency band k.

- Then, the value of B_k (dB) is determined as the larger of $N_k^{'}$ and V_k. We shall mathematically define this latter statement as:

$$B_k = \begin{cases} N_k^{'}, & \text{if } N_k^{'} > V_k \\ V_k, & \text{if } V_k > N_k^{'} \end{cases} . \tag{7.3}$$

- The next step is to determine C_k (dB) which is the slope per octave (doubling of frequency) of the upward spread of masking in dB/octave for each frequency band. In our particular case, this parameter can be computed by using the expression given below:

$$C_k = -80 + 0.6 \cdot [B_k + 10 \cdot \log F_k - 6.353] \tag{7.4}$$

  where:

  - B_k labels the parameter previously explained, and
  - F_k represents the nominal midband frequency of the "one-third octave bands" method, for each particular frequency band k.

  With this in mind, the corresponding *equivalent masking spectrum level* ($Z_k^{'}$), for central and higher frequency bands can now be written as:

$$Z_k = 10 \cdot \log \{10^{0.1 N_k^{'}} + \sum_{b=0}^{k-1} 10^{0.1[B_b + 3.32 \cdot C_b \cdot \log(0.89 \cdot F_k / F_b)]}\} \tag{7.5}$$

  where:

  - $N_k^{'}$ represents the equivalent noise spectrum level,
  - B_b is the same as B_k,
  - F_k is the nominal midband frequency of the "one-third octave bands" method, and finally,
  - F_b is the nominal midband frequency for frequency band k.

  Note that for lower frequency bands, we assume that $Z_k = B_k$.

2. Estimating the *equivalent internal noise spectrum level* ($X_k^{'}$) (dB)
   In this thesis, this parameter adopts the following formulation:

$$X_k^{'} = X_k + T_k^{'} \tag{7.6}$$

   where:

   - X_k is the reference internal noise spectrum level listed in Table 3 in [ANSI, 1997], and
   - $T_k^{'}$ is the estimated equivalent hearing threshold level or, in other words, the particular hearing loss the subject suffers from.

3. Determining the *equivalent disturbance level* (D_k) (dB)
   This parameter is determined as the larger of Z_k and $X_k^{'}$. We shall mathematically define this statement as:

$$D_k = \begin{cases} Z_k, & \text{if } Z_k > X_k^{'} \\ X_k^{'}, & \text{if } X_k^{'} > Z_k \end{cases} .$$

4. Computing the *speech level distortion factor* (L_k) (dB)
   This parameter accounts for the decrease in the intelligibility of speech at high input signal levels. It ranges from 0 (high levels) to 1 (there is no distortion due to presentation level). It can be computed by using the expression shown below:

$$L_k = 1 - \frac{E_k^{'} - U_k - 10}{160} \tag{7.7}$$

   where:

   - $E_k^{'}$ is the *equivalent speech spectrum level* defined in Expression 7.1, and
   - U_k is the standard speech spectrum level at normal vocal effort listed in Table 3 in ANSI [1997].

5. Computing the *band audibility function* (A_k) (dB)
   This important parameter is the effective portion of the speech dynamic range within the band that contributes to the speech intelligibility under conditions less than optimal. In the effort of computing this value, a temporary variable K_k needs to be calculated previously. This variable represents the total effective signal to noise ratio and is limited to the range of 0 and 1. The way this variable can be computed is depicted in the following expression:

$$K_k = 1 - \frac{E_k^{'} - D_k + 15}{30} \tag{7.8}$$

   where

   - $E_k^{'}$ represents the *equivalent speech spectrum level* whose expression for the problem at hand we deduced in Expression 7.1, and
   - D_k is the *equivalent disturbance level.*

   Therefore, the *band audibility* is computed by using the expression shown below:

$$A_k = L_k \cdot K_k \tag{7.9}$$

   where:

   - L_k is the *speech level distortion* factor computed from Expression 7.7, and
   - K_k is the constant computed in Expression 7.8.

6. Speech intelligibility index (SII)
   With this scenario in mind, we can proceed in explaining the way the SII score can be computed, as clearly shown below:

$$\text{SII} = \sum_{k=1}^{N_\text{B}} I_k \cdot A_k \tag{7.10}$$

   where:

- I_k is the band-importance function. This function depends on the sort of speech signal and the proficiency of talkers and listeners to understand speech (a range of band-importance functions is given in Table B.2 of the standard [ANSI, 1997]), and
- A_k is the *band audibility function* that we have previously deduced.

For illustrative purposes, and facilitating the reader's comprehension, Figure 7.4 aims at illustrating the conceptual representation of a general SII computation system. The practical point to highlight here regarding this figure is that the system requires, for estimating the SII score, the input speech signal, the subject's hearing threshold and the background noise level. It is important to note that, within the particular application at hand, since the goal will be to design an "amplification scheme" for speech-in-quiet signals (or in other words, speech without presence of background noise or with a high SNR), the background noise will be assumed to be 0 dB SPL.

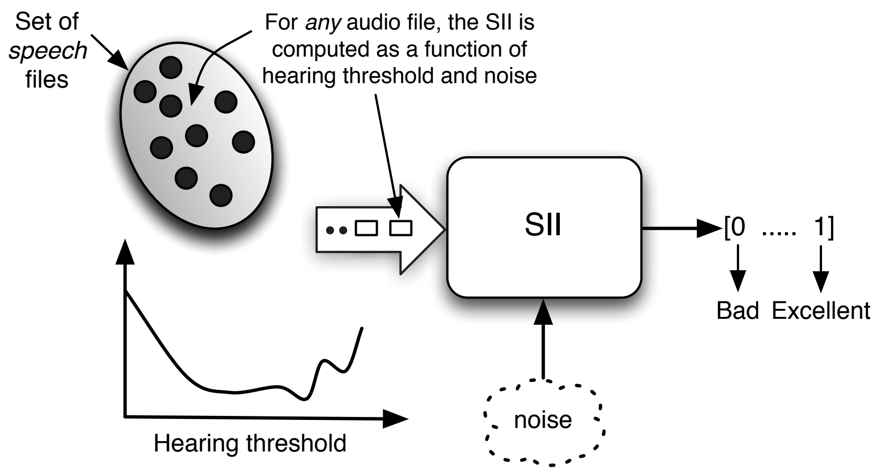

Set of *speech* files
For *any* audio file, the SII is computed as a function of hearing threshold and noise
SII
[0 ..... 1]
Bad
Excellent
Hearing threshold
noise

Figure 7.4: Conceptual representation of the way the SII score is computed. For *any* speech file, the SII block computes, after some mathematical handling, a score between 0 (bad) and 1 (excellent) as a function of the patient's hearing threshold and the background noise. Within the application at hand, since the goal is to design an "amplification program" for speech-in-quiet files (with no background noise or with a high signal-to-noise ratio), the background noise is assumed to be 0 dB SPL.

7.5 Blind modeling of the gain function by using gaussian mixtures

As mentioned in Chapter 2, a digital hearing aid basically consists of a bank of sound compressors, one for each frequency band, $k = 1, \dots, N_B$, N_B being the number of frequency bands available in the DSP ($N_B = 64$ bands, in the particular case at hand). We can formally express the gain matrix $\mathcal{G}$, containing all the parameters required to completely compute the gain in any band for any input signal level, as shown below:

$$\mathcal{G} = f(\mathbf{X}, k) \tag{7.11}$$

where:

- $\mathbf{X}$ is the level (dB) of the input signal, and
- $k = 1, \dots, N_{\mathrm{B}}$ is the index over the N_{B} frequency components of the STFT of the input signal.

Parameters involved in matrix $\widehat{\mathcal{G}}$ (matrix $\widehat{\mathcal{G}}$ representing the optimized parameters), have to be optimized in the effort of enhancing: 1) the speech quality (PESQ measure), and 2) the speech intelligibility (SII measure) perceived by hearing-impaired people. This optimization process can be mathematically written as:

$$\widehat{\mathcal{G}} = \max_{\mathcal{G}} \text{PESQ} \tag{7.12}$$

where "PESQ" represents the PESQ measure, or also,

$$\widehat{\mathcal{G}} = \max_{\mathcal{G}} \text{SII} \tag{7.13}$$

where "SII" represents the SII measure.

Computing the gain, as a function of the input signal level ($\mathbf{X}$) and the frequency (k), can require intensive computational cost. To illustrate this statement, we can imagine, for the sake of simplicity, a simple scenario with a conventional hearing aid, in which for any input level and any frequency, the gain can be modeled with a 3-piece linear approximation (we refer the reader to Subsection 2.3.5 in page 23 for further details). This demands to compute 5 parameters to define the curve for each frequency band. If $N_{\mathrm{B}} = 64$ bands are considered, the total number of parameters included in matrix $\mathcal{G}$ would be $65 \times 5 = 320$.

In the effort of reducing this number, and in the aim of "not contaminating" the gain design with non-perceptual concepts, we propose to model the complete gain $\mathcal{G}$ (and not its separate pieces), as a smooth function in a "blind" method that we describe later on. Prior to this, it is convenient to remark we say the method is "blind" in the sense we *do not* introduce any initial information but that related to "perceptual concepts" (just those that can be numerically quantified by the PESQ or SII measures). And this is just the reason why we have mentioned that the method aims to "not contaminate" the gain with non-perceptual impositions. An elegant way of modeling $\mathcal{G}$ is by making use of the gaussian mixture model described in the following subsection.

7.5.1 Gaussian mixture model

With the aforementioned concepts in mind, the initial, blind structure of the gain $\mathcal{G}$ can be formally approached by using a gaussian mixture model (GMM) [Roberts et al., 1998], which basically consists in a properly weighted combination of gaussian components, and exhibits the following expression:

$$\mathcal{G} \approx p(\mathbf{x}) = \sum_{ind=1}^{g} p_{ind} \cdot \mathcal{N}(\mathbf{m}_{ind}, \mathbf{C}_{ind}) \tag{7.14}$$

where:

- $p(\mathbf{x})$ is the probability density function of the gaussian mixture,
- $\mathbf{x} = [x_1, x_2, \dots, x_d]$ is a d-dimensional data vector,
- g represents the number of gaussian components,

- $\mathcal{N}$ is a (normalized) multivariate normal or gaussian distribution (with mean value $\mathbf{m_{ind}}$ and covariance matrix $\mathbf{C_{ind}}$) whose density function can be expressed as:

$$\mathcal{N} = \frac{1}{(2\pi)^{(d/2)}|\mathbf{C}_{ind}|^{1/2}} \exp\Big\{ -\frac{1}{2}(\mathbf{x}-\mathbf{m}_{ind})^T\mathbf{C}_{ind}^{-1}(\mathbf{x}-\mathbf{m}_{ind})\Big\}. \tag{7.15}$$

The weights p_{ind} in Expression 7.14 can be interpreted as the *a priori* probability that an observation of $\mathbf{x}$ comes from the source governed by the g-th gaussian distribution. Thus, the following conditions must be fulfilled:

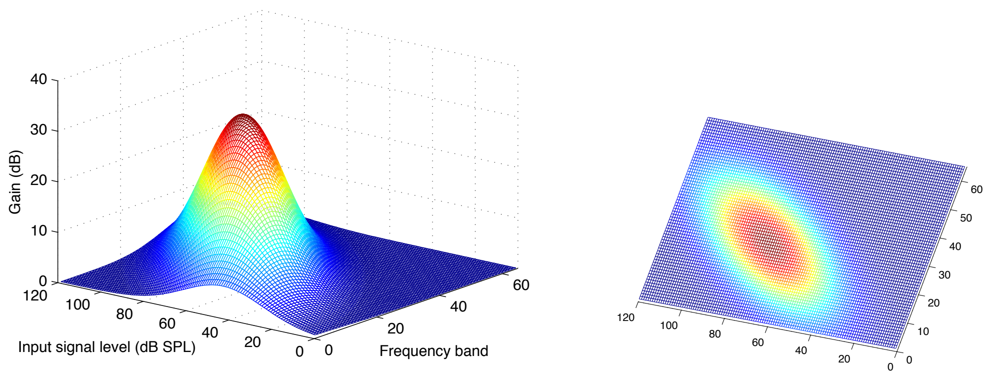
$$0 \leqslant p_{ind} \leqslant 1, \qquad \forall\ ind = 1, \ldots, g \tag{7.16}$$

and

$$\sum_{ind=1}^{g} p_{ind} = 1. \tag{7.17}$$

With this scenario in mind, the complete gaussian mixture model is parametrized by the mean vectors, covariance matrices and mixture weights from all components.

In the effort of facilitating the reader's comprehension, we have included Figure 7.5, which basically represents an illustrative example of an "ideal" gain matrix ($\mathcal{G}$) obtained with the approach proposed in this chapter. As shown, this gain function would amplify soft-medium speech sounds (60-85 dB SPL) and would reduce the gain for not speech sounds. This latter can be better noticed in Figure 7.17(b), in which the maximum of the gaussian mixture is approximately located at speech frequencies (frequency bands from 15 to 35) and thus, the gain for *not* speech sounds would be approximately 0 dB.



(a) (b)

Figure 7.5: An illustrative example of an *ideal* gain matrix ($\mathcal{G}$) obtained with the speech enhancement approach proposed in this chapter. As shown, this gain function would amplify soft-medium speech sounds (60-85 dB SPL) and reduce the gain for not speech sounds.

Prior to focus on other important details, it is convenient to emphasize that, apart from the fact we *do not introduce non-perceptual information*, a second advantage of this approach is that the number of parameters to be optimized is appreciably reduced since now that number *does not* depend on the number of frequency bands in the hearing aid.

An important issue in mixture modeling is the selection of the number of gaussian distributions that compose the GMM, or in other words, the value of g. This number of components is usually unknown in real-world problems. A mixture with too many components will overfit the data, whereas a mixture with too few components will be too simple to approximate underlying distributions. Many algorithms have been proposed in the literature for the selection of the proper number of components. Some of these approaches add a penalty term in the objective function that can potentially penalize more complex models. A detailed review of different criteria used to penalize complex models is given in [McLachlan and Peel, 2000]. In the batches of experiments carried out in this chapter, a number of gaussian components higher than $g > 3$ has been found to *not appreciably increase* either the speech quality or speech intelligibility perceived by the user, and thus, for the sake of simplicity and aiming at using the lowest number of computational resources, $g = 3$ has been considered as the maximum number of gaussian components in $\mathfrak{G}$.

The rest of parameters in Expression 7.14 will be computed by making use of a genetic algorithm (see Appendix D for further details).

7.5.2 The approach at work

Figure 7.6 will assist us in more clearly introducing the approach proposed in this thesis. As shown, *any* speech signal available in a training phase, S_{C}, subscript "C" meaning "clean", is processed by the hearing aid, and for each amplified output signal produced by the hearing aid, S_{O}, subscript "O" meaning "output", a score (based on an objective measure) is computed.

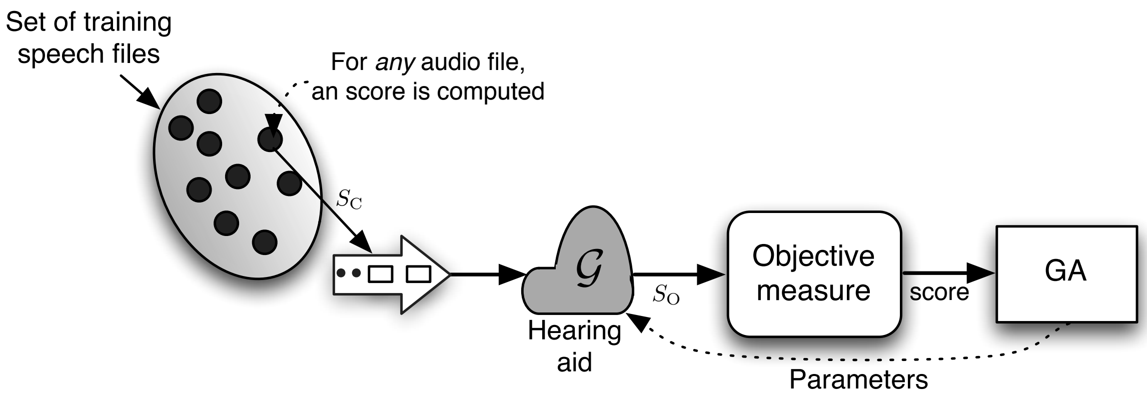


Figure 7.6: Simplified block diagram illustrating the way the proposed speech enhancement approach works. Any speech file available in the training set (S_{C}) is processed by the hearing aid, and for each amplified output signal produced by the hearing aid (S_{O}), a score (based on an objective measure) is computed. Since the signal S_{O} depends on the gain matrix ($\mathfrak{G}$), the goal of the genetic algorithm (GA) is to find the best candidate solution $\mathfrak{G}$ that is "best fitted", or in other words, the one that maximizes the objective measure (that is, the fitness function).

This amplified output signal produced by the hearing, S_{O}, depends on the amplifying matrix of the hearing aid. This gain matrix ($\mathfrak{G}$) is created by using a gaussian mixture model driven by a genetic algorithm. The goal of the GA is thus to find the best candidate solution $\mathfrak{G}$ in Expression 7.14 that is "best fitted", or in other words, the one that maximizes the objective measure (computed over the signal S_{O}).

As pointed out, in an effort of not only improving the speech quality, but also the speech intelligibility perceived by the user, two different objective methods have been considered here to compute the score. These objective methods are based on: 1) the PESQ measure and 2) the

SII measure, and consequently, they intend for quantitatively measuring the speech quality and the speech intelligibility, respectively.

Depending on the method chosen to compute the score, the approach, shown in Figure 7.6, will be slightly different. For illustrative purposes, and facilitating the reader's comprehension, we have preferred to explain such approaches in the ordered way that follows.

1) Figure 7.7 aims at illustrating the approach when it makes use of PESQ measure. As depicted, this approach basically compares the amplified output signal produced by the hearing aid, S_O, with a reference, high quality speech signal, S_C. It is important to highlight that, conceptually, the output signal S_O is basically the reference signal S_C after 1) being corrupted with different kinds of noise (N), and 2) after being modified by the hearing aid. Using both signals, the PESQ block generates a *score* ranging from -0.5 (bad) to 4.5 (excellent). The higher the score is, the better the speech "perceived by the human auditory system" is. Please note that, in this case, the fitness function (PESQ measure) is computed over both signals S_C and S_O.

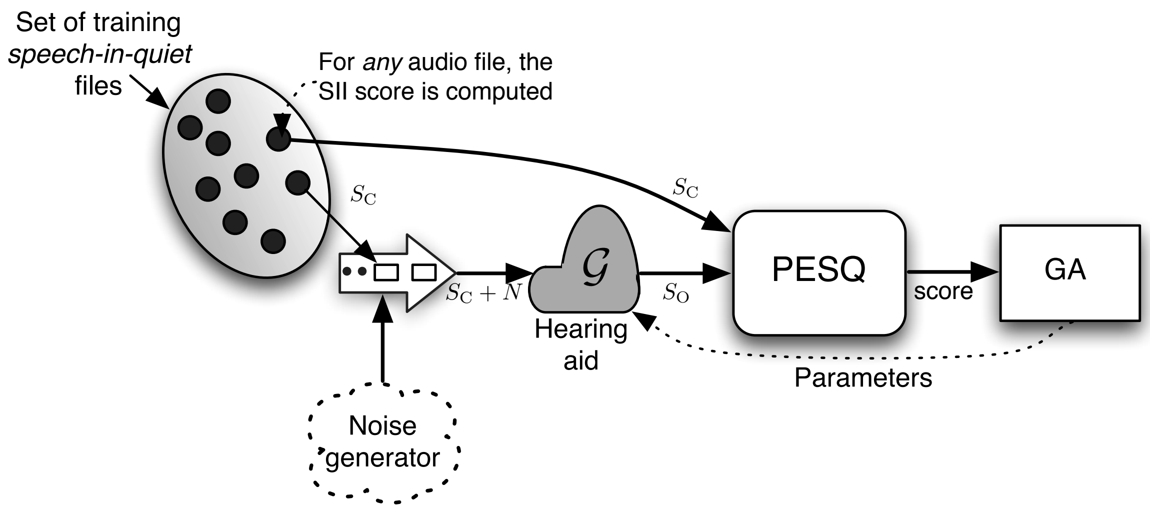

Set of training *speech-in-quiet* files
For *any* audio file, the SII score is computed
S_C
S_C
$S_C + N$
$\mathcal{G}$
Hearing aid
S_O
PESQ
score
GA
Parameters
Noise generator

Figure 7.7: Block diagram illustrating the way the speech enhancement approach works when it makes use of the PESQ measure. The PESQ block compares the amplified output signal produced by the hearing aid, S_O, with a reference, high quality speech signal, S_C. Conceptually, the output signal S_O is basically the reference signal S_C after 1) being corrupted with different kinds of noise (N), and 2) after being modified by the hearing aid. Using both signals, the PESQ block generates a "score" ranging from -0.5 (bad) to 4.5 (excellent). Since S_O (and thus the score) depends on the gaussian mixture gain $\mathfrak{G}$, the goal of the GA is thus to find the best candidate solution $\mathfrak{G}$, or in other words, the one that maximizes the PESQ measure (computed over signals S_C and S_O).

2) Figure assists us in introducing the approach when it makes use of SII measure. As illustrated, the approach generates a score as a function of: 1) the input speech signal level (in the particular problem at hand, a speech-in-quiet signal), 2) the background noise level, 3) the threshold of hearing, and 4) the amplifying function (or gain function) of the hearing aid. In order to compute this score, since the subject is embedded in a scenario without background noise, we have assumed the background noise to be 0 dB SPL, as depicted in the figure. Please note that the output signal S_O is basically the input signal (S_C) after having been modified by the hearing aid.

Bearing in mind that one of the parameters for the SII computation is the threshold of

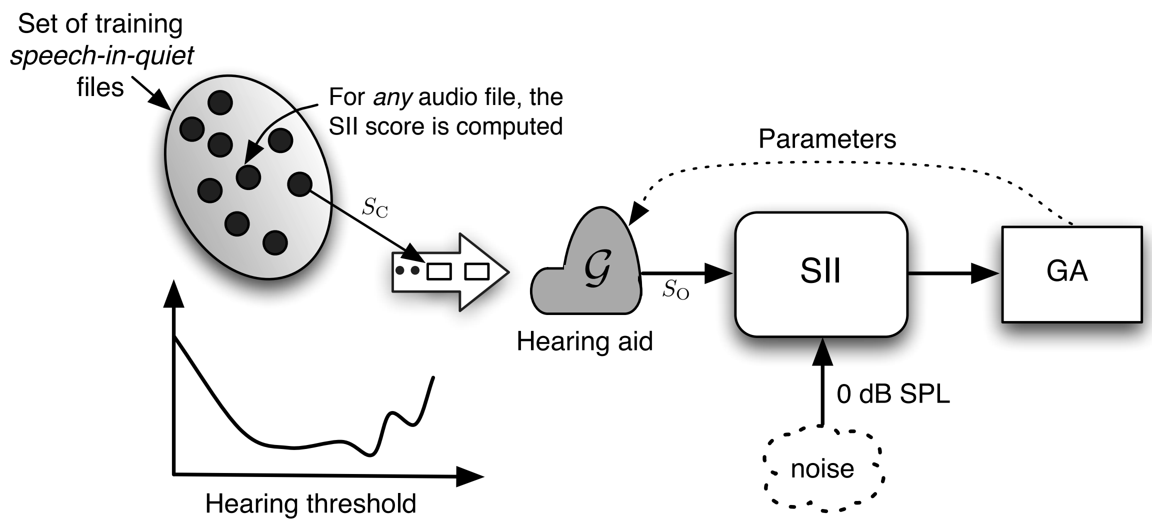

Set of training *speech-in-quiet* files
For *any* audio file, the SII score is computed
S_C
$\mathcal{G}$
S_O
Hearing aid
Parameters
SII
GA
0 dB SPL
noise
Hearing threshold

Figure 7.8: Block diagram illustrating the way the speech enhancement approach works when it makes use of the SII measure. The SII block computes a score as a function of: 1) the input speech-in-quiet signal level, 2) the background noise level, 3) the threshold of hearing and 4) the amplifying function (gain) of the hearing aid. The output signal (S_O) is basically the input signal after having been modified by the hearing aid. Since S_O (and thus the score) depends on the gaussian mixture gain $\mathcal{G}$, the goal of the GA is thus to find the best candidate solution $\mathcal{G}$, or in other words, the one that maximizes the SII measure (computed over the signal S_O).

hearing and in the effort of making the more *realistic* approach, we have carried out a series of experiments with different thresholds of hearing. To be more precise, we have considered a subset of 4 different *real* patients, with mild to profound hearing loss, which have been found to be sufficiently representative for the problem at hand.

These patients, in ascending order of acoustic loss, are as follows:

- Patient 6 (P_6): a *mild* hearing loss subject, with an average hearing threshold level of 31 dB.
- Patient 2 (P_2): a *moderate* hearing loss subject, with an average hearing threshold level of 56 dB.
- Patient 9 (P_9): a *severe* hearing loss subject, with an average hearing threshold level of 72 dB.
- Patient 8 (P_8): a *profound* hearing loss subject, with an average hearing threshold level greater than 80 dB.

The main characteristics of these patients are listed in Table C.2 in Appendix C (page 213).

7.6 Experimental work and results

Prior to the description of the experiments carried out in the effort of automatically generating, by making use of perceptual concepts, and a supervised learning algorithm driven by a genetic algorithm, a gain matrix ($\mathcal{G}$) that aims at enhancing, not only speech quality, but also speech intelligibility perceived by the hearing aid user (Subsection 7.6.2), it is worth having a look at the sound database used for the experiments (Subsection 7.6.1).

7.6.1 Experimental setup

In order to carry out the experiments, we have made use of different sound databases. These databases will be briefly described below.

- **Database 1**. This database contains a total of 30 speech-in-quiet files, randomly selected from the database described in Section C.1 in Appendix C (page 211). These 30 clean speech sentences have been degraded by additive white gaussian noise at SNRs 0 dB, 5 dB, 10 dB and 15 dB. These values of SNRs have been taken from [Loizou, 2007]. The sentences have been sampled at 8 kHz, and the number of bits per sample is 16. The database includes utterances by American English male and female speakers. For properly training and testing the learning process, it is necessary for the database to be divided into two different sets. For this reason, 7 files (25 %) have been picked randomly as training data, and the remaining, 23 (75 %) files have been used for the purpose of testing. This division has been made ensuring that the relative proportion of files of each category is preserved for each set.

- **Database 2**. It is composed of 30 IEEE clean speech sentences, extracted from the noisy speech corpus (NOIZEUS) available in [Loizou, 2007]. These clean speech sentences have been degraded by additive white gaussian noise at different SNRs, ranging from 0 dB to 15 dB in steps of 5 dB. These sentences were recorded in a sound-proof booth using Tucker Davis Technologies recording equipment. The sentences were produced by three males and three females (five sentences per speaker). This IEEE database has been used because it contains *phonetically balanced* sentences with relatively low word-context predictability. The aforementioned sentences have been selected from the IEEE database so as they include all phonemes in the American English language. The sentences were sampled at 8 kHz and the number of bits per sample is 16.

- **Database 3**. This database is composed of the same 30 IEEE clean speech sentences used to design the previous database ("database 2"), but in this case, these clean speech sentences have been corrupted by eight different *real-world* noises at SNRs 0 dB, 5 dB, 10 dB and 15 dB. The sampling frequency is 8 kHz with 16 bits per sample. The real-world noises were taken from the AURORA database [Hirsch and Pearce, 2000], and include suburban train noise, multi-talker babble, car, exhibition hall, restaurant, street, airport, and train-station noise. This database is compiled by Loizou [Loizou, 2007].

- **Database 4**. This database contains a total of 20 speech-in-quiet files (10 files by American English male speakers and 10 files by American English female speakers), randomly selected from the database described in Section C.1 in Appendix C (page 211). For properly training and testing, this database has been divided into two subsets, "training" and "test" subsets. These sets contain 10 files (50 %) for training and 10 files (50 %) for testing. This division has been done randomly, ensuring the relative proportion of files of each category is preserved for each set.

Regarding the parameters involved in the gain matrix ($\mathcal{G}$), it is worth mentioning that we have utilized a single GA that computes those parameters v_q, the gain matrix depends on, that maximize the PESQ or SII measure (the *objective function* here) for the patterns available in a design phase. With this in mind, any *individual* is thus a real-number vector with the structure $I \equiv [v_1, v_2, v_3, \ldots, v_6]$. Figure 7.9 aims at illustrating what each element v_q in the aforementioned vector represents.

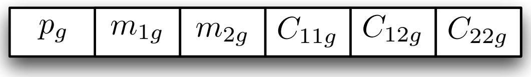


Figure 7.9: Schematic representation of an *individudal* that encodes a trial solution of the parameters the gain matrix $\mathfrak{G}$ depends on. For the sake of clarity, it shows an individual that encodes a solution in the easiest situation to represent in which $g = 1$ (that is, only 1 gaussian has been used for the GMM). In this picture, p_k represents the probability density function of the GMM, m_{1g} and m_{2g} designate the mean elements, and finally, C_{11g}, C_{12g} and C_{22g} label the covariance elements.

Having a look at the figure, it seems clear to notice that v_1 designates the probability density function of the gaussian mixture (that is, p_g), v_2 and v_3 represent the mean elements (that is, m_{1g} and m_{2g}, respectively) and finally, v_4, v_5 and v_6 label the covariance elements, (that is, C_{11g}, C_{12g} and C_{22g}, respectively). With this in mind, the goal of the GA is to find the *best individual* $I_{\text{best}} \equiv [v_{\text{best}1}, v_{\text{best}2}, v_{\text{best}3}, \ldots, v_{\text{best}6}]$ that is "best fitted", or in other words, the one that maximizes the fitness score. Please note that the individual shown in the picture encodes a solution in the easiest situation to represent in which $g = 1$ (that is, only 1 gaussian has been used for the GMM). When the mixture consists of a greater number of components, such as, for instance, $g = 2$ gaussian components, the dimension of the individual would be 12 (the first six parameters would define one gaussian component and the following six parameters would define the second gaussian component) and so on. The complete GA operates as clearly explained in Appendix D.

We have summarized in Table 7.3 the main design parameters the GA makes use of. These values have been found to be reaching an adequate balance between training time and good results in the batches of experiments carried out.

Parameters	**Value**
Initial population (p_0)	$100 \times g$
Crossover probability (p_c)	0.8
Mutation probability (p_m)	0.1
Max. no. of generations	50
Max. no. of iterations in which the score remains unchanged	20

Table 7.3: Summary of the design parameters the GA makes use of for automatically generating a gain function ($\mathfrak{G}$) that aims at not only enhancing the speech quality perceived by the hearing aid user, but also the speech intelligibility. In this table, g designates the number of gaussian components used in the mixture.

Regarding the fitness function, it is worth noting that it will be different depending on the objective measure chosen. On the one hand, when the approach makes use of the PESQ measure, since the PESQ score ranges from -0.5 to 4.5, the fitness function has been found to be:

$$f_{\text{fitness}} = 1 - \frac{\text{PESQ\_score} + 0.5}{5}. \tag{7.18}$$

On the other hand, when the approach makes use of the SII measure, taking into account that SII score ranges from -0 to 1, the fitness function has been found to be:

$$f_{\text{fitness}} = 1 - \text{SII\_score}. \tag{7.19}$$

Completing this description of the batches of experiments demands to mention that the experiments have been repeated 10 times. The best of these realizations in terms of fitness score computed over the patterns available in a design phase has been selected. Perhaps the reader may wonder why we have repeated this batch of experiments only 10 times and not more times. In this respect, it is important to point out that the variance of the results obtained by 10 runs of the genetic algorithm is kept below 0.013 and 0.001 for the case in which the approach makes use of the PESQ measure and SII measure, respectively. Therefore, aiming at saving computational resources, it is not worthwhile to run the experiments a large number of times. The results we have illustrated below correspond to the *test* set.

7.6.2 Numerical results

In the effort of studying the efficiency achieved by the approach proposed here in the sense of improving not only speech quality but also speech intelligibility in hearing aids, we have considered two different scenarios depending on the objective measure used.

In the first set of experiments, we have considered the particular case in which the approach intends for improving speech quality, and consequently, it makes use of the PESQ measure (see for further details Figure 7.7). To carry out the experiments, we have made use of "database 1". The speech-in-quiet files included in this database have been "contaminated" with additive white gaussian noise at SNR $= 0$ dB. We have chosen this value of SNR because this case represents the situation in which: 1) the energy of the speech signal and that corresponding to noise exhibits the same level, and 2) white noise is uniformly distributed in all frequencies, and consequently, is especially hard to be detected and removed.

Figures 7.10, 7.11 and 7.12, represent, the mean value and the standard deviation of the computed PESQ score as a function of SNR, when the number of gaussian components in the obtained gain matrix ($\mathfrak{G}$) has been found to be $g = 1$, 2, and 3, respectively.

The solid lines in this sequence of figures correspond to the mean value and the standard deviation of the PESQ scores computed over the patterns available in a test phase at different SNRs. Note that these results correspond to the scenario in which the hearing aid implements the gain matrix created by the aforementioned algorithm. For comparative purposes, the dashed, monotonously increasing lines in the figures correspond to the situation in which the hearing aid does not implement the aforementioned gain matrix and the gain function programmed in it is based on a standard 3-piecewise linear approximation for each frequency band.

Figures 7.10, 7.11 and 7.12 provide the following valuable information:

1. The computed, perceptual-based gain matrix ($\mathfrak{G}$), makes the hearing aid reach better results in terms of speech quality, regardless the number of gaussian components (g) used.

2. Regardless the SNR, the results obtained with $g = 3$ gaussian components are always better than those when using only $g = 1$ or 2 gaussian components.

3. These figures clearly illustrate to what extent the proposed method enhances the speech quality, especially in the complicated scenario in which the level of noise equals the level of speech (that is, SNR$= 0$ dB).

In the effort of clearly showing to what extent the proposed method achieves to enhance speech, not only in terms of speech quality, but also in terms of speech intelligibility, Table 7.4 lists the mean speech quality improvement (%) and the mean speech intelligibility improvement (%), as a function of the number of gaussian components used (g). It seems clear to note that the best result is reached with $g = 3$ components, leading to speech quality improvement equal

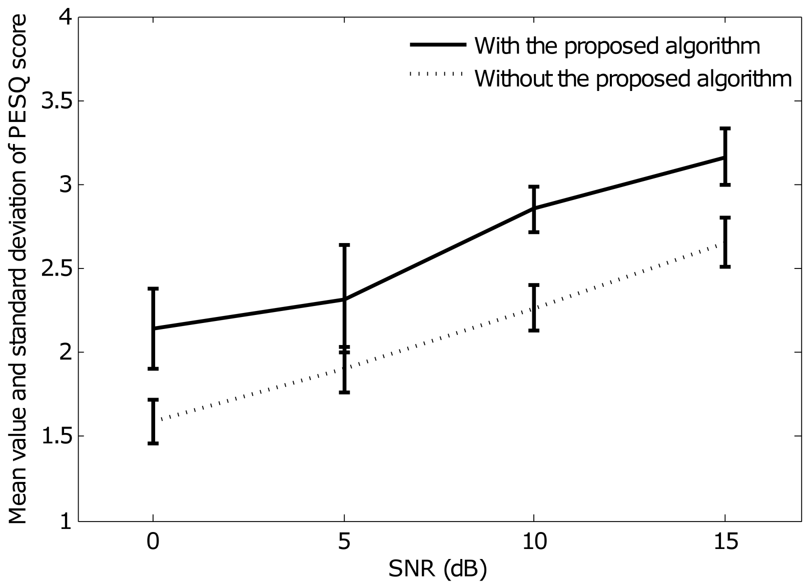


Figure 7.10: Mean value and standard deviation of the PESQ score computed as a function of the SNR (dB) for $g = 1$ gaussian component in the mixture. These results have been computed over the patterns available in a test phase, after having trained the algorithm by using files at SNR $= 0$ dB.

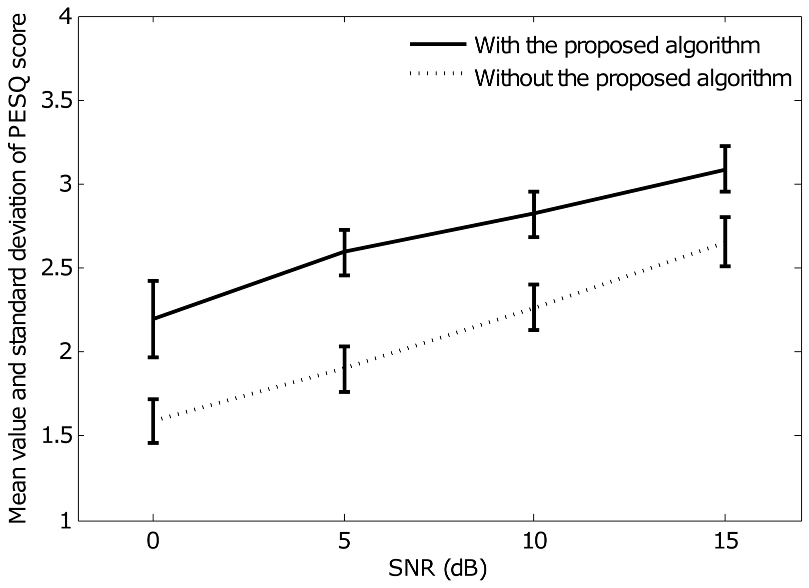


Figure 7.11: Mean value and standard deviation of the PESQ score computed as a function of the SNR(dB) for $g = 2$ gaussian components in the mixture. These results have been computed over the patterns available in a test phase, after having trained the algorithm by using files at SNR $= 0$ dB.

to 14.5 % and speech intelligibility improvement equal to 10.9 %. Even for g =1 ($\mathfrak{G}$ consists of a single gaussian component), speech quality is enhanced in 10.5 % and speech intelligibility is enhanced 5.7 %. The results listed in this table correspond to the scenario in which the algorithm has been trained by using speech-in-noise files at SNR $= 0$ dB.

In order to explore the influence of other SNRs, we have carried out a similar batch of

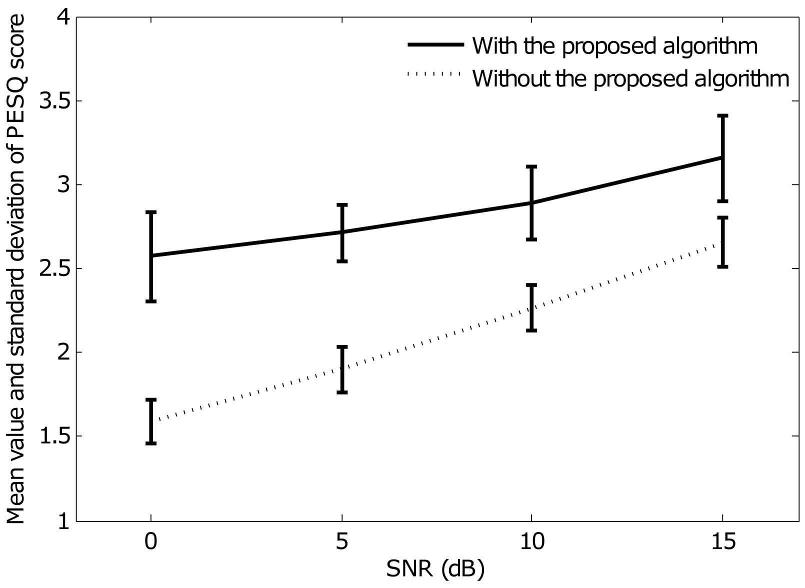


Figure 7.12: Mean value and standard deviation of the PESQ score computed as a function of the SNR(dB) for $g = 3$ gaussian components in the mixture. These results have been computed over the patterns available in a test phase, after having trained the algorithm by using files at SNR $= 0$ dB.

Gaussian components (g)	**Improvement (%)**	
	Speech quality	**Speech intelligibility**
1	10.3	5.7
2	11.4	8.9
3	14.5	11.9

Table 7.4: Mean speech quality improvement (%) and mean speech intelligibility improvement (%), as a function of the number of gaussian components used (g). These results correspond to the situation in which the algorithm has been trained by using speech-in-noise files at SNR $= 0$ dB.

experiments in which the training process is carried out with sets of files containing speech-in-noise at SNR $= 5$ dB. The system has been ulteriorly tested with sets of files at different SNRs. Table 7.5 shows the mean speech quality improvement (%) and the mean speech intelligibility improvement (%) for different SNRs when using $g = 1$, 2 or 3 gaussian components. Note that, as in the former batch of experiments, the best result is achieved when $g = 3$ gaussian components are used in the mixture model, with speech quality improvement of 15.1 % and speech intelligibility improvement of 12.3 %. The speech quality enhancement exceeds 10 % even with $g = 1$ gaussian.

Gaussian components	**Improvement (%)**	
	Speech quality	**Speech intelligibility**
1	11.0	6.8
2	14.7	9.6
3	15.1	12.2

Table 7.5: Mean speech quality improvement (%) and mean speech intelligibility improvement (%), as a function of the number of gaussian components used (g). These results correspond to the situation in which the algorithm has been trained by using speech-in-noise files at SNR $= 5$ dB.

Finally, and for comparative purposes in order to complete this first set of experiments, and to evaluate the influence of the database used, we have tested the results obtained with "database 1" using now "database 2" and "database 3". The results shown in the following two figures correspond to those obtained when the system was trained with speech-in-noise files at SNR $= 5$ dB and $g = 3$, since this gaussian mixture has been found to be the scheme that provides the best result in terms of speech enhancement for the patterns available in the *test* phase. Figure 7.13 represents the mean value of the PESQ scores obtained when using "database 2". The mean speech improvement quality has been found to be 15.0 %. This is a very significant enhancement. On the other hand, Figure 7.14 illustrates the mean value of the PESQ scores obtained when using "database 3", which includes speech-in-real-noises. In this case, the mean speech quality enhancement has been found to be about 8.0 %.

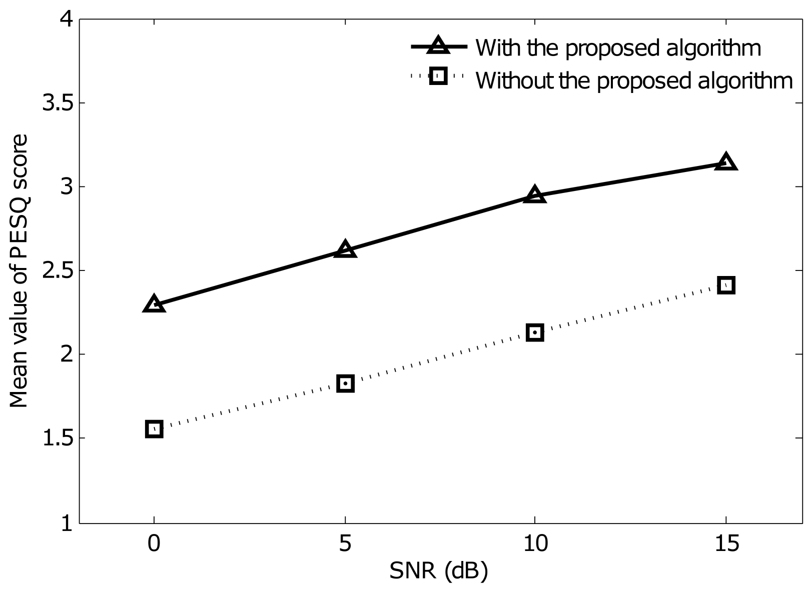


Figure 7.13: Mean value of the PESQ score computed as a function of SNR (dB). These results correspond to those obtained by using "database 2" (speech degraded with gaussian noise). The algorithm has been trained with files containing speech-in-noise at SNR $= 5$ dB. The mixture has been found to have $g = 3$ gaussian components.

In the second batch of experiments, we have considered the particular case in which the approach attempts for improving speech intelligibility, and consequently, it makes use of the SII measure (see for further details Figure 7.8). To carry out this batch of experiments, we have made use of "database 4" discussed in Subsection 7.6.1.

Figure 7.15 illustrates, for the subset of the 4 *real* subjects under study (or in other words, subjects that suffer from mild, moderate, severe and profound hearing loss), the mean speech intelligibility curves, calculated over the speech files available in a *test* phase, as a function of the input signal level (dB SPL), ranging from 0 to 120 dB SPL. For the sake of simplicity and using the lowest number of computational resources, the gaussian mixture gain ($\mathcal{G}$) makes use of $g = 1$ component.

The green lines in this sequence of figures correspond to the mean value and the standard deviation of the SII scores computed in the scenario in which the subject wears a hearing aid that implements the gain matrix obtained by using the algorithm proposed in this chapter. For

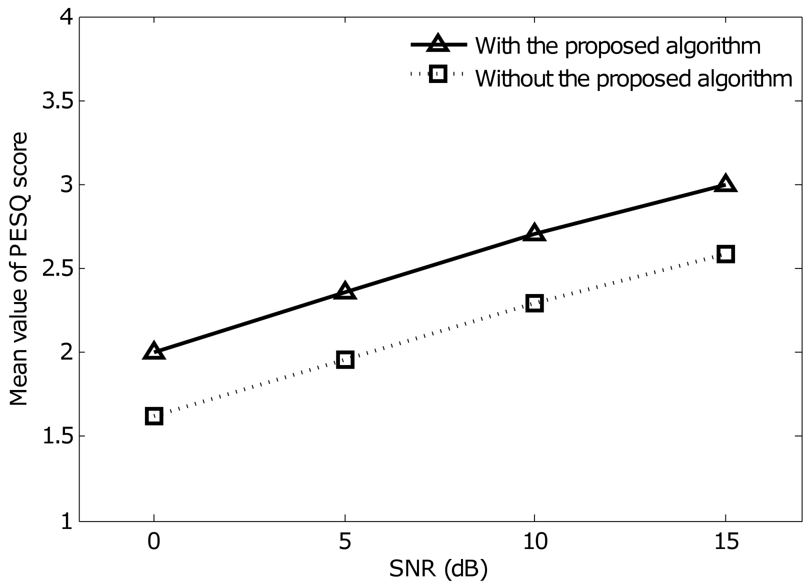


Figure 7.14: Mean value of the PESQ score computed as a function of SNR (dB). These results correspond to those obtained by using "database 3" (speech degraded with real noises). The algorithm has been trained with files containing speech-in-noise at SNR $= 5$ dB. The mixture has been found to have $g = 3$ components.

comparative purposes, the blue lines in the same sequence of figures correspond to the situation in which the subject *does not* wear any type of hearing aid. Error bars here denote ± 1 standard deviation.

This figure provides the following valuable information:

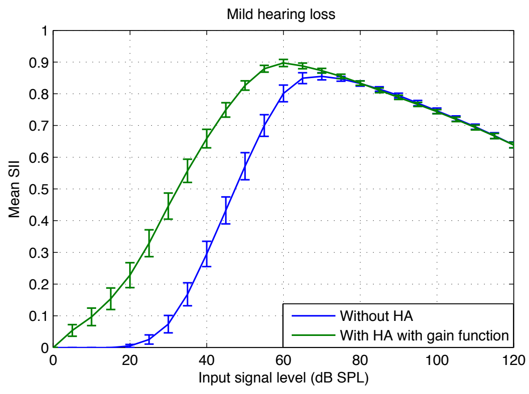
1. Obviously, the computed, perceptual-based gain matrix ($\mathcal{G}$) always makes the hearing aid reach better results in terms of speech intelligibility, regardless the degree of subject's acoustic loss.

2. In particular, it is important to highlight that, for the subjects that suffer from moderate and severe hearing loss, the hearing aid that implements the gain matrix obtained by using the algorithm proposed here, achieves a significant improvement in the speech intelligibility perceived by the user of 60 %, with respect to the situation in which these subjects *do not* wear any type of hearing aid.

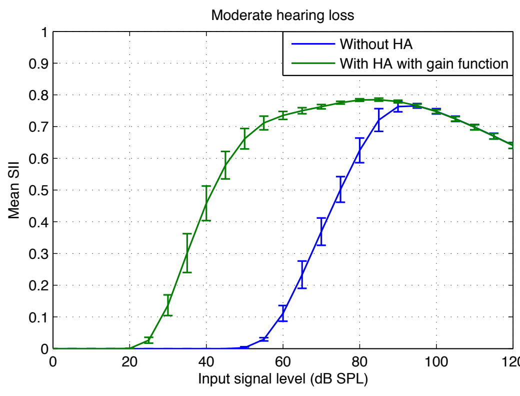
Figure 7.16 aims at showing in an comprehensive way the results we have obtained in the effort of validating the approach in terms of speech intelligibility. We have chosen a representation that intends for clearly illustrating such results of the mean and standard deviation of the SII scores attained by the hearing aid designed with the gain matrix proposed here and the hearing aid designed with a typical compression system based on a standard 3-piecewise linear approximation for each frequency band (labelled "conventional HA" in the picture).

The key points to note here in Figure 7.16 are:

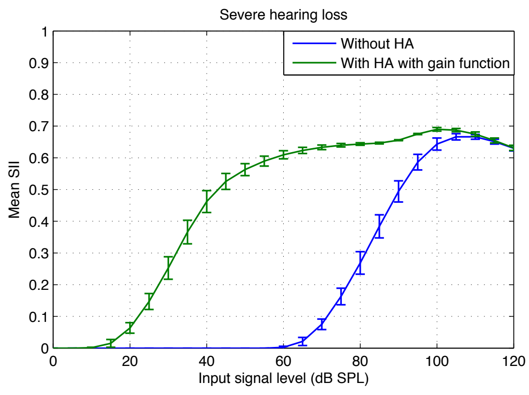
1. The conventional hearing aid performs worst than the hearing aid that implements the gain matrix obtained with the novel strategy proposed here: the conventional hearing aid



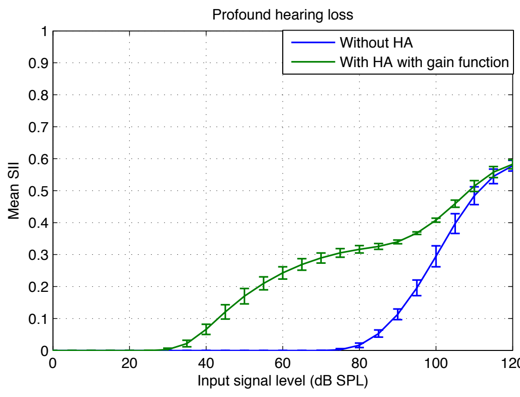
(a)



(b)



(c)



(d)

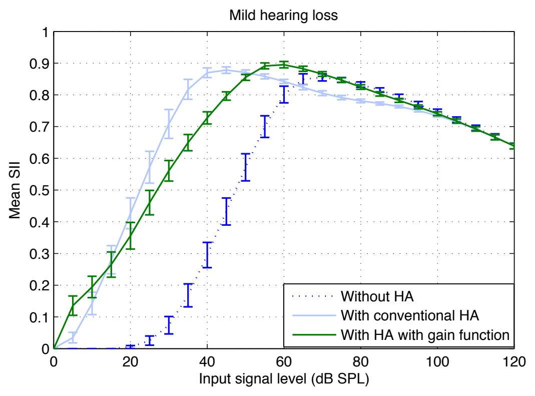
Figure 7.15: Mean speech intelligibility index (SII) (green line), computed over the available test-patterns, as a function of the input signal level (dB SPL) for the subset of 4 *real* study-case subjects: mild hearing loss subject, moderate hearing loss subject, severe hearing loss subject, and finally, a profound hearing loss subject. It is also shown the mean SII score for the situation in which these patients do not wear any type of hearing aid (blue line). Error bars here denote ± 1 standard deviation.

generally exhibits lower intelligibility curves than those achieved by the hearing aid that implements the gain matrix created by using the aforementioned algorithm.

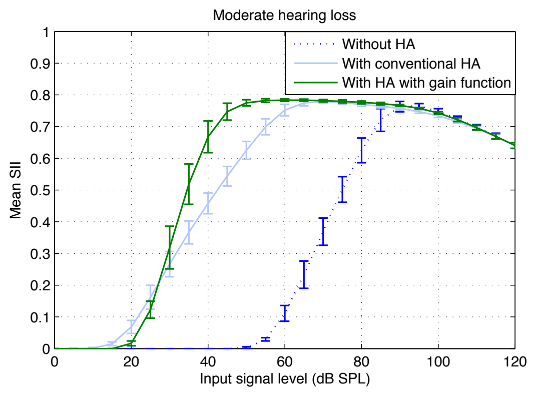
2. For the case of mild hearing loss, the conventional hearing aid behaves slightly better than the hearing aid designed with the novel strategy for low input signal levels (about $0 - 50$ dB SPL). However, as far as conversational speech is concerned, normal face-to-face communication is in the range of 53 dB SPL (*soft* speech conversation) to 80 dB SPL (raised speech conversation)[Firzst, 2010]. In this range, the hearing designed with the novel approach performs better, being 58 dB SPL (normal speech conversation) the *optimum* level for greatest speech intelligibility.

3. Although the novel approach assists the hearing aid in increasing the mean SII score, it sometimes achieves this at the expense of increasing the standard deviation when compared to that of the conventional hearing aid.

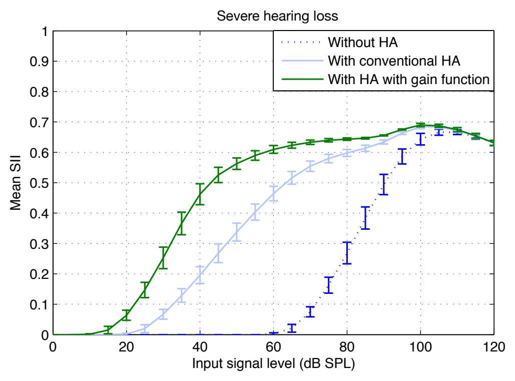
For illustrative purposes, Figure 7.17 aims at representing the gain matrix obtained in the experiments (with $g = 3$ gaussian components) for the subject that suffers from severe hearing



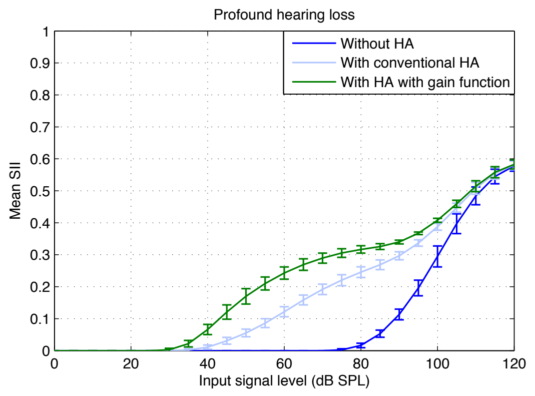
(a)



(b)



(c)



(d)

Figure 7.16: Mean speech intelligibility index (SII) score (green line), computed over the patterns available in a *test* phase, as a function of the input signal level (dB SPL) for the subset of the 4 *real* study-case subjects: mild hearing loss subject, moderate hearing loss subject, severe hearing loss subject, and finally, a profound hearing loss subject. It is also shown the mean SII score, computed over the the same test files, when these subjects *do not* wear any type of hearing aid (dark blue line) and the mean SII score obtained when these subjects wear a conventional hearing aid (light blue green). Error bars here denote ± 1 standard deviation.

loss. As depicted, this gain matrix basically aims at enhancing medium speech sounds ($\approx$ 80 dB SPL) for mid-high frequencies. This makes sense since this particular subject suffers from high frequency hearing loss. See row "P_9" in Table C.2 in Appendix C (page 212) for further details.

Finally, we complete this second batch of experiments by summarizing in Table 7.6, the most important numerical results considered as representative enough of the feasibility of our approach. This table represents the mean speech intelligibility improvement (%) and the mean speech quality improvement (%) achieved by the hearing aid that implements the gain matrix obtained by using the approach proposed here with respect to the situation in which the gain provided by the hearing aid is based on a standard 3-piecewise linear approximation for each frequency band, for the four study-case subjects and making use of g = 1, 2 and 3 gaussian components.

Having a look at the mentioned table, it is important to note that the results show that for the subjects that suffer from mild and moderate hearing loss, the speech intelligibility improvement, for conversational speech, could be considered to be very close. However, for the subject that

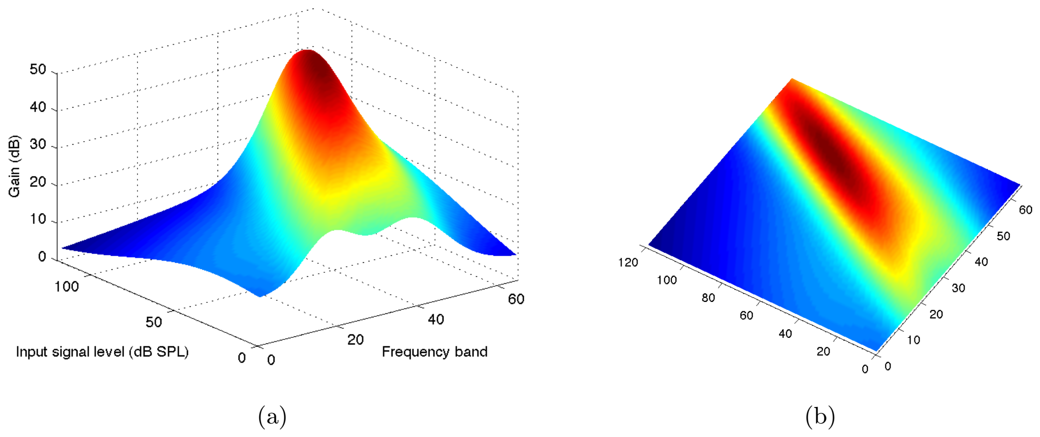


Figure 7.17: An illustrative representation of the gain matrix (with $g = 3$ gaussian components) obtained with the algorithm proposed in this thesis for the subject that suffers from severe hearing loss. As depicted, this gain matrix basically aims at enhancing medium speech sounds ($\approx$ 80 dB SPL) for mid-high frequencies. This makes sense since this particular subject suffers from high frequency hearing loss.

Subject's hearing loss	**Improvement (%)**					
	Speech intelligibility			**Speech quality**		
	$g = 1$	$g = 2$	$g = 3$	$g = 1$	$g = 2$	$g = 3$
Mild	5.1	5.8	5.9	3.4	3.6	3.6
Moderate	3.7	3.7	-1.8	-1.9	-1.4	-2.1
Severe	14.7	14.7	21.0	10.2	11.1	13.4
Profound	19.6	19.6	21.4	10.4	11.0	14.9

Table 7.6: Mean speech intelligibility improvement (%) and mean speech quality improvement (%), for the subset of four study-case subjects, achieved by the hearing aid that implements the gain matrix obtained by using the approach proposed here with respect to the situation in which the gain provided by the hearing aid is based on a standard 3-piecewise linear approximation for each frequency band. In this table, g labels the number of gaussian components that compose the mixture.

suffers from severe hearing loss, the speech intelligibility is improved up to 21 % by making use of the hearing aid designed with our gain matrix (with $g = 3$ gaussian components), with respect to the case in which a conventional hearing aid is used, leading to a global improvement of about 60 % with respect to the situation in which the subject *does not* wear any kind of hearing aid (note that this last figure is not shown in the table). For the subject that suffers from profound hearing loss, it can be observed that better results are also reached by the hearing aid that implements the gain matrix obtained by using the algorithm proposed here, with an improvement of 21.4 % with respect to the conventional hearing aid (also with $g = 3$ gaussian components).

An extremely important point to note regarding this table is that the speech quality perceived by the hearing-impaired subjects under study is not always increased. This can be clearly noticed by having a look at the speech quality perceived by the subject that suffers from moderate hearing loss. Regardless the number of gaussian component used (g), the speech quality is always degraded when compared to that perceived by the subject when he or she makes use of a hearing in which the gain is based on a typical 3-piecewise linear approximation for each frequency band.

This assists us in elucidating the following key conclusion: when the algorithm designed in this chapter makes use of a fitness function based on the PESQ measure (Expression 7.18), which intends for quantitatively measuring the speech quality, not only the speech quality perceived by the subject is improved but also the speech intelligibility when compared to the situation in which the subject makes use of a conventional hearing that implements a gain function based on 3-piecewise linear approximation for each frequency band. However, when the approach makes use of the of a fitness function based on the SII measure (Expression 7.19), which intends for quantitatively measuring the speech intelligibility, the speech intelligibility is always increased but the speech quality is not always enhanced. This leads to design the approach, illustrated in Figure 7.6, by using the PESQ measure as objective measure, what makes it more speech enhancement-efficient.

7.7 Conclusions

This chapter focuses on a *novel approach* aiming at speech enhancement in hearing aids. It basically consists in creating -by exploiting perceptual concepts, and a supervised learning process driven by a genetic algorithm- a gain matrix (labelled $\mathfrak{G}$) that enhances not only the speech quality, but also the speech intelligibility perceived by the user. To what extent this is enhanced is measured by the algorithm itself by using a scheme based on an objective measure. In this respect, it is important to highlight that the main purpose of many algorithms of speech enhancement proposed in the literature is to improve speech quality while preserving, at the very last, speech intelligibility. In this sense, only a few number of algorithms have been evaluated by using intelligibility tests, and in those studies, only a single speech enhancement algorithm was evaluated in different noisy environments. Aiming at properly validating the speech enhancement approach proposed in this chapter, not only in terms of speech quality, but also in terms of speech intelligibility, we have made use of two different standardized objective measures: the perceptual evaluation of speech quality (PESQ) and the speech intelligibility index (SII), which aim to evaluate speech quality and speech intelligibility, respectively.

The proposed algorithm creates the enhanced gain matrix, $\mathfrak{G}$, by using a gaussian mixture model driven by a genetic algorithm, and *does not* use any initial information but that related to “perceptual concepts”, or in other words, just those that can be iteratively quantified by the PESQ measure (that is, the fitness function is based on the PESQ measure) or SII measure (or equivalently, the fitness function depends on the SII measure). This gain matrix under construction (and, in turns, the PESQ or SII scores) depends on a number of parameters. In this respect, the GA computes the “optimized” parameters that maximize these scores and consequently, the speech “perceived by the human auditory system”.

The results show that the computed, perceptual-based gain matrix ($\mathfrak{G}$) makes the hearing aid reach better results in terms of speech quality and speech intelligibility perceived by the user (regardless the number of gaussian components -g- used in the mixture and the subject’s hearing loss) than those obtained when the subject makes use of a hearing aid in which the gain is based on a 3-piecewise linear approximation for each frequency band. Regarding the speech quality, it is worth mentioning that it is is improved up to 14.5 % and the speech intelligibility is enhanced 11.9 % (using $g = 3$ gaussian components in the mixture) in the complicated scenario in which the algorithms is trained by using a database that contains speech-in-noise files at SNR= 0 dB. Concerning the speech intelligibility, it is worth pointing out that, for instance, for a subject that suffers from severe hearing loss, the speech intelligibility is boost up to 21.0 % (by using $g = 3$ gaussian components in the mixture). A key point to note is that when the approach makes use of the fitness function based on the PESQ measure, not only the speech quality perceived by the user is improved but also the speech quality. However, when the approach makes use of the fitness function based on the SII measure, the speech intelligibility is always increased regardless the subject's hearing loss, but the speech intelligibility is not always enhanced. This leads to design the approach designed in this thesis by using the PESQ measure as objective measure, what makes it more speech enhancement-efficient.

These results point out to a new field of research in the jointly use of perceptual and artificial intelligent concepts applied on speech enhancement in hearing aids. Besides the obvious improvement in the quality and intelligibility of speech perceived by the user, the key point, very appreciated from the scientific and technological perspective, consists in its lower computational cost. This is because, once the computed, perceptual-based gain matrix $\mathfrak{G}$ is programmed in the DSP, the hearing aid saves a great amount of computational resources since it is no longer necessary to run any other noise reduction algorithm. This is of crucial importance because of the inherent limitations of the DSP the hearing aid is based on. Note that the hearing aid has to work at a very low clock frequency in order to minimize power consumption and thus maximize the battery life.

Chapter 8

Hardware implementation

8.1 Introduction

As mentioned in the introductory chapter, the *purpose* of this thesis is to design a *prototype* for a digital hearing aid, which aims to automatically recognize the acoustic environment its user is in and choose the amplification program that best fits such environment. The practical implementation of this type of hearing aid demands, at least, the programming of the following three functional blocks in the DSP:

1. A multiband compressor-expander algorithm: put it very simple, this block, the very core of a hearing aid, amplifies the input audio signal to a level that allows a hearing impaired person to hear the sounds that otherwise he or she *would not able to* hear.

2. A feature extraction stage: this block plays the key role of processing the input audio signal, after a sequence of operations carried out in the data processing block (for the sake of simplicity, this latter block is also included in the implementation of the feature extraction block *itself*), aiming at extracting some kind of relevant, essential information that characterizes the audio signal for further classification.

3. A classifying algorithm: this classifier, based on the features extracted from the input audio signal in the previous stage, makes a decision on the audio class (speech, music and noise, in our case) to which the input signal belongs to.

The previous chapters discussed the overall classification system that best performed for distinguishing among speech, music and noise in hearing aids. In this respect, it is worth mentioning that the features arranged in the vector $\mathbf{F}_{\text{practical}}$, which will be finally implemented in the DSP, have been experimentally found to be the mean and variance of the features listed below:

- The two novel, low-complexity spectral centroid-based features, which we have labelled $\widehat{\text{SC}}$ and $\widetilde{\text{SC}}$.

- The two novel, low-complexity voice2white-based features, which we have labelled $\widehat{\text{V2W}}$ and $\widetilde{\text{V2W}}$.

- The two novel, low-complexity spectral flux-based features, which we have labelled $\widehat{\text{SF}}$.

- The short-time-energy feature, labelled STE.

With this in mind, the sound signal-describing vector that feeds the classifier has been found to be:

$$\begin{aligned}\mathbf{F}_{\text{practical}} = \{&\widehat{E}[\widehat{\text{SC}}], \widehat{\sigma}^2[\widehat{\text{SC}}], \widehat{E}[\widetilde{\text{SC}}], \widehat{\sigma}^2[\widetilde{\text{SC}}], \widehat{E}[\widehat{\text{V2W}}], \widehat{\sigma}^2[\widehat{\text{V2W}}],\\ &\widehat{E}[\widetilde{\text{V2W}}], \widehat{\sigma}^2[\widetilde{\text{V2W}}], \widehat{E}[\widehat{\text{SF}}], \widehat{\sigma}^2[\widehat{\text{SF}}], \widehat{E}[\text{STE}], \widehat{\sigma}^2[\text{STE}]\}.^T\end{aligned} \tag{8.1}$$

Note that the dimension of the feature vector is thus $dim(\mathbf{F}_{\text{practical}}) = 12$.

Among the classifiers evaluated in this thesis, we have chosen a multilayer perceptron (a particular kind of neural network), which had to be properly designed for being constrained to the hardware requirements of the DSP used by our platform to carry out the experiments. The key reason that compelled us to choose the MLP approach is that, as shown in the results obtained in Chapter 5, multilayer perceptrons are able to learn from appropriate patterns available in a *design* phase, and properly classify other patterns that have never been found before. This ultimately leads to very good results, as a balance between percentage of correct classification and computational cost, when compared to those from other popular algorithms evaluated in this thesis, such as, for instance, the mean square error (MSE) linear classifier, the k-nearest neighbor algorithm, or even, a radial-basis function (RBF) network. As mentioned in Chapter 5, an important issue when designing an MLP-based classifier is the selection of the number of hidden neurons (M). In this sense, and thanks to the growing and pruning algorithms proposed in this thesis, the appropriate number of hidden neurons, when the abovementioned 12-feature vector ($\mathbf{F}_{\text{practical}}$) feeds the MLP, has been found to be 20 (that is, $M = 20$). Note that the number of input neurons (L) represents the number of features selected for feeding the classifying algorithm, or in other words, the dimension of the feature vector $\mathbf{F}_{\text{practical}}$ (that is, $L = 12$) and the number of output neurons (C) is related to the three classes we are interested in (or equivalently, $C = 3$).

With these ideas in mind, the structure of this chapter is as follows. Section 8.2 depicts the *particular* way the compressor-expander approach proposed in this thesis is implemented in the DSP, showing also the computation time (in clock cycles) and CPU-load average (in %) required for its implementation. Section 8.3 clearly explains the way the feature vector $\mathbf{F}_{\text{practical}}$ is programmed in the DSP. It is also shown the computation time (in clock cycles) and CPU-load average (in %) demanded for its computation. Section 8.4 describes the way the multilayer perceptron obtained in the experiments is implemented in DSP, along with the computation time (in clock cycles) and CPU-load average (in %) needed for its practical implementation. Section 8.5 depicts the consumption of the functional hearing aid implemented in this thesis, one of the *crucial* aspects when designing these devices. Finally, Section 8.6 discusses the key conclusions obtained in the implementation of the prototype for the hearing aid.

8.2 Implementation of the multiband compressor-expander

8.2.1 Introduction

Instead of programming in our DSP a multiband compressor-expander algorithm strictly speaking, we have proposed in this thesis, taking advantage of the fact that the WOLA filterbank provides at its output the time/frequency decomposition of any input sound frame (with $N_{\text{B}} = 64$ frequency bands), a *novel* approach consisting in a gain matrix $\mathfrak{G}$ (sometimes named as "gain table"), which assists us in estimating the gain value to apply to the input audio signal, as clearly explained in Section 7.5 in Chapter 7 (page 130). Deepening a little more in this design strategy, Figure 8.1 illustrates the conceptual representation of the mentioned gain matrix. As shown, this table, stored in memory available in the DSP, basically includes the "tabulated" gain values to apply as a function of *both* the input signal level (dB SPL) and the number of frequency band. Note that, since the number of acoustic environments to recognize in this thesis are three (speech, music and noise), this obviously requires to design three different gain matrixes, one for each audio classes considered.

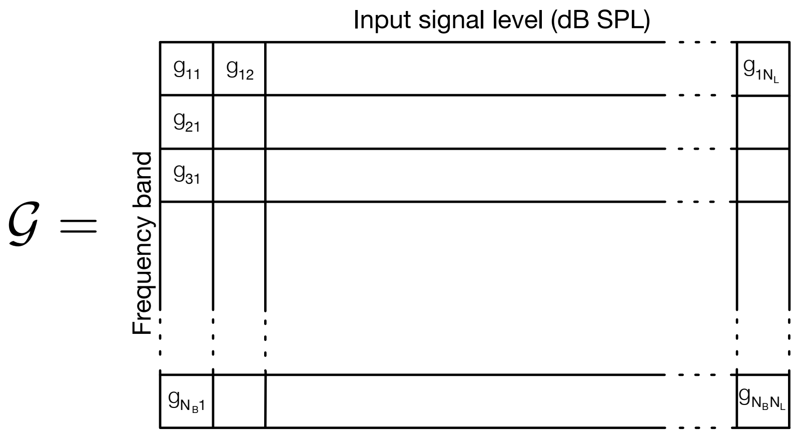


Figure 8.1: Picture illustrating the gain matrix $\mathcal{G}$ used in this thesis to estimate the gain value to apply to the input audio signal from a hardware implementation point of view. It consists of a table stored in data-memory available in the DSP, which basically includes the "tabulated" gain values to apply as a function of *both* the input signal level (dB SPL) and the frequency band.

The motivation for exploring this strategy is to significantly reduce the number of instructions per second demanded to implement the typical compressor-expander algorithm based on a standard 3-piecewise linear approximation for each frequency band. Once we have explained the compression-expansion strategy proposed in this thesis, the question emerging here is: *what is the most adequate way of implementing these gain matrixes in the DSP?* At a first glance, the most feasible approach, which significantly could reduce the computational cost required for implementing any gain matrix, is as follows: 1) computing the gain values to apply *offline* (in the laboratory); 2) tabulating these gain values for any input signal level (dB SPL) and any frequency band; and 3) storing the tabulated gain values in the memory available in the DSP. We propose this approach instead of, for instance, computing the gain values directly in the DSP which involves high computational cost. In other words, note that the key point that exhibits the greatest relevance in this first approach resides on the fact that the gain values *are computed offline*. Only when these values have been computed in the laboratory, they are stored in the memory available in the DSP. The main advantage of this approach is that it requires much lower computational cost at the expense of increasing the amount of memory used. However, as will be shown throughout this chapter, this stronger use of data-memory is perfectly feasible and *does not* exceed the restrictions imposed by the DSP used by our platform to carry out the experiments.

Thus, using this approach, the reader may wonder the subsequent questions: *how many different input signal-levels (N_L) and frequency bands (N_B) should be considered*? And consequently, *how many different values of gain should we consider*? As stated beforehand, the WOLA filter bank provides at its output 64 frequency bands (and consequently, $N_B = 64$), and since the 16-bit ADC and DAC converters, included in the DSP used by our platform, exhibit a dynamic range of 96.3 dB, we have set the dynamic margin ranging from 1 to 96 dB for the input signal level, in steps of 1 dB, which has been found in our experiments as having enough precision (that is, $N_\mathrm{L} = 96$). Within this framework, the total number of tabulated values for each gain matrix to be stored in the data-memory has been found to be $N_\mathrm{L} \times N_\mathrm{B} = 96 \times 64 = 6144$.

With these considerations in mind and in an effort of putting this compression-expansion strategy in a more in-depth way, we can proceed further in describing the sequence of operations carried out in the DSP aiming to obtain the gain value.

1. Computation of the average power of the input frame i-th at each frequency band k-th (labelled $P_{\mathrm{avg}_i}(k)$), as will be explained in Subsection 8.2.2. Please note that, within our framework, $k = 1, \ldots, N_\mathrm{B}$ is the index over the N_B frequency components of the WOLA filter bank, while $i = 1, \ldots, N_\mathrm{frames}$ labels the one over the N_frames frames into which any sound signal is segmented.

2. Searching the gain value, in the corresponding gain matrix, as a function of the input signal level (dB SPL) and the number of frequency band. This task will be described in Subsection 8.2.3.

3. Applying the gain values to the corresponding input frame at each frequency band, as will be shown in Subsection 8.2.4.

Prior to describing the way we have programmed this complete functional block, it is indispensable to have a look at the design restrictions imposed by the DSP used by our platform to carry out the experiments. As mentioned, digital hearing aids habitually suffer from inherent design limitations that make difficult the practical implementation of the aforementioned gain matrixes in the DSP. Among these limitations, memory and computational cost (this latter is naturally related to the number of assembler instructions demanded to implement the gain matrixes) still remain certainly a big problem. To better understand this, these limitations are explained in-depth in the paragraphs that immediately follow.

- **Computational cost limitation**:
  As stated beforehand, for any analysis frame, we need to obtain a gain value for each of the $N_\mathrm{B} = 64$ frequency bands. Within our framework, since the input block size is $R = 64$ samples, the interrupt `IO_BLOCK_FULL` is activated when 64 *new* samples of the input audio signal are arranged in the input buffer[1]. With this scenario in mind and taking into account that the sampling frequency is 16 kHz, the maximum number of interrupts per second to attend has been found to be:

$$\frac{16000 \text{ samples/s}}{64 \text{ samples/interrupts}} = 250 \text{ interrupts/s}.$$

  Since the clock frequency of the DSP used by our platform to carry out the experiments is 1.92 MHz, the number of clock cycles per interrupt is:

$$\frac{1.92 \cdot 10^6 \text{ cycles/s}}{250 \text{ interrupts/s}} = 7680 \text{ cycles/interrupt}.$$

[1]Note that, within the application at hand, the analyze window length is $L = 256$ samples, but the input block size is $R = 64$ samples, what, in turns, means that these 64 samples and the latter 192 samples from the previous frame comprising the current input frame (see for further details Appendix B).

Bearing in mind that we need to carry out this operation for each of the N_{B} frequency bands available in the DSP ($N_{\mathrm{B}} = 64$, in our case), the number of clock cycles per frequency band adopts the following expression:

$$\frac{7680 \text{ cycles/interrupt}}{64 \text{ bands/interrupt}} = 120 \text{ cycles/band.}$$

This figure illustrates that the *maximum* number of clock cycles available in the DSP to obtain the gain value, for each of the frequency bands obtained at the output of the WOLA filter bank for the analysis frame, has been found to be 120. Taking into account that most of instructions demand, at least, 1 clock cycle for their implementation, the fact of including a new instruction in the design involves an additional CPU-load of approximately 0.83 %, which clearly shows the severe computational cost limitations the complete system suffers from.

- **Memory limitation**:
  As previously mentioned, within our framework, the number of tabulated gain values comprising each gain matrix has been found to be $64 \times 96 = 6144$, which involves that 6144 values need to be stored in the memory available in the DSP. Bearing in mind that we need to store a gain matrix for each of the acoustic environments considered (speech, music and noise, in our case), the total number of tabulated values to be stored in memory, within the particular application at hand, has been found to be $3 \times 6144 = 18432$. Since the program-memory capacity of the DSP is 12 kwords, and the data-memory capacity is 8 kwords, we are also strongly constrained to the restrictions imposed by the DSP in terms of memory capacity for the practical implementation of the aforementioned gain matrixes.

8.2.2 Average power computation

Although not clear at first glance, it is worth noting that the task of estimating the power of an input signal (or being more precise, of an input frame) is relatively easy and simple within our framework, by taking into account that the WOLA coprocessor provides at its output the spectrum of such input signal (or such input frame).

We shall mathematically define the computation of the power of a signal $X(t)$ as depicted below:

$$P(k) = |\chi(k)|^2 \tag{8.2}$$

where $\chi(k)$ labels the k-th frequency band of the spectrum of signal $X(t)$, being $k = 1, \ldots, N_{\mathrm{B}}$ the index over the N_{B} frequency bands ($N_{\mathrm{B}} = 64$ bands, in our case). Bearing in mind that the input signal is segmented into N_{frames} frames within our framework, or in other words, $X_i(t)$, $i = 1, \ldots, N_{\mathrm{frames}}$, N_{frames} being the number of frames into which the signal is divided, we shall rewrite Expression 8.2 as shown below:

$$P_i(k) = |\chi_i(k)|^2 \tag{8.3}$$

where $i = 1, \ldots, N_{\mathrm{frames}}$ labels the index over the N_{frames} frames into which the input signal $X(t)$ is segmented.

It is important to note that the computation of the power of a frame as shown in Expression 8.3 *does not* designate the *average power* of such frame. The key point to understand this is that the power computed as shown in the mentioned expression labels the power corresponding to the “short” time slot between two input frames. This could be better understood if the reader remembers that, within our framework, each analysis frame length is $L = 256$ samples, being $R = 64$ samples of the input signal and the remaining 192 samples of the previous frame. With this in mind and considering that the sampling frequency of our DSP is 16 kHz, the time slot between two adjacent frames can be computed as follows:

$$\frac{64 \text{ samples/frame}}{16000 \text{ samples/s}} = 4 \text{ ms/frame.}$$

The computation of the power, according to Expression 8.3, labels thus a power that is computed every 4 ms within the application at hand. This is basically the reason why we refer to such power as the “instantaneous power” of the signal (and not as the “average power” of the signal). For the sake of clarity, we label it “P_{ins}”.

With this in mind, the reader, perhaps, may wonder the way the average power of a signal is exactly computed. Answering this question demands to mention that the computation of the average power, that we have labelled P_{avg}, demands to calculate some instantaneous powers. To be more precise, the total number of instantaneous powers needed to compute the average power will depend on the length of the time slot in which we would like to calculate such average power. Obviously, the higher the time slot length is, the higher computational load becomes. In the effort of reducing such computational load, instead of calculating the average power strictly speaking, we have computed an “estimation” of the average power of a frame i-th by means of the following expression:

$$P_{\text{avg}_i}(k) = \begin{cases} \beta_r \cdot (P_{\text{avg}_{i-1}}(k) - P_{\text{ins}_i}(k)) + P_{\text{avg}_{i-1}}(k) & \text{if } P_{\text{ins}_i}(k) < P_{\text{avg}_{i-1}}(k) \\ \beta_a \cdot (P_{\text{ins}_i}(k) - P_{\text{avg}_{i-1}}(k)) + P_{\text{avg}_{i-1}}(k) & \text{if } P_{\text{ins}_i}(k) \geqslant P_{\text{avg}_{i-1}}(k) \end{cases}$$

where:

- $P_{\text{avg}_{i-1}}(k)$ represents the k-th frequency band of the average power for the previous frame $i-1$-th,
- $P_{\text{ins}_i}(k)$ represents the k-th frequency band of the instantaneous power for the analysis frame i-th,
- $P_{\text{ins}_{i-1}}(k)$ represents the k-th frequency band of the instantaneous power for the previous frame $i-1$-th,
- β_a labels the “attack compressor time”, and finally,
- β_r depicts the “release compressor time”.

Both β_a and β_r constants must be a *negative power of two*, such as, for instance, 0.5, 0.25, 0.125 and so on. Aiming at clearly understanding this latter restriction, it is worth mentioning that a multiplication by a number that is an exact power of two can be made simply by a shift operation in the DSP, which significantly reduces the computational load. For instance, multiplying a number by 2 is equivalent to shift the number left by one bit. Similarly, multiplying a number by 0.5 is equivalent to shift the number right by one bit.

Figure 8.2 will assist us in illustrating the main concepts involved in the sequence of operations carried out for properly computing the average power. The average power is represented by a 32-bit fixed-point format and its 16 most significative bits are stored in data-memory for further computation of the feature vector as will shown later on. Please note that this sequence of operations is carried out for each of the frames into which the input signal is segmented.

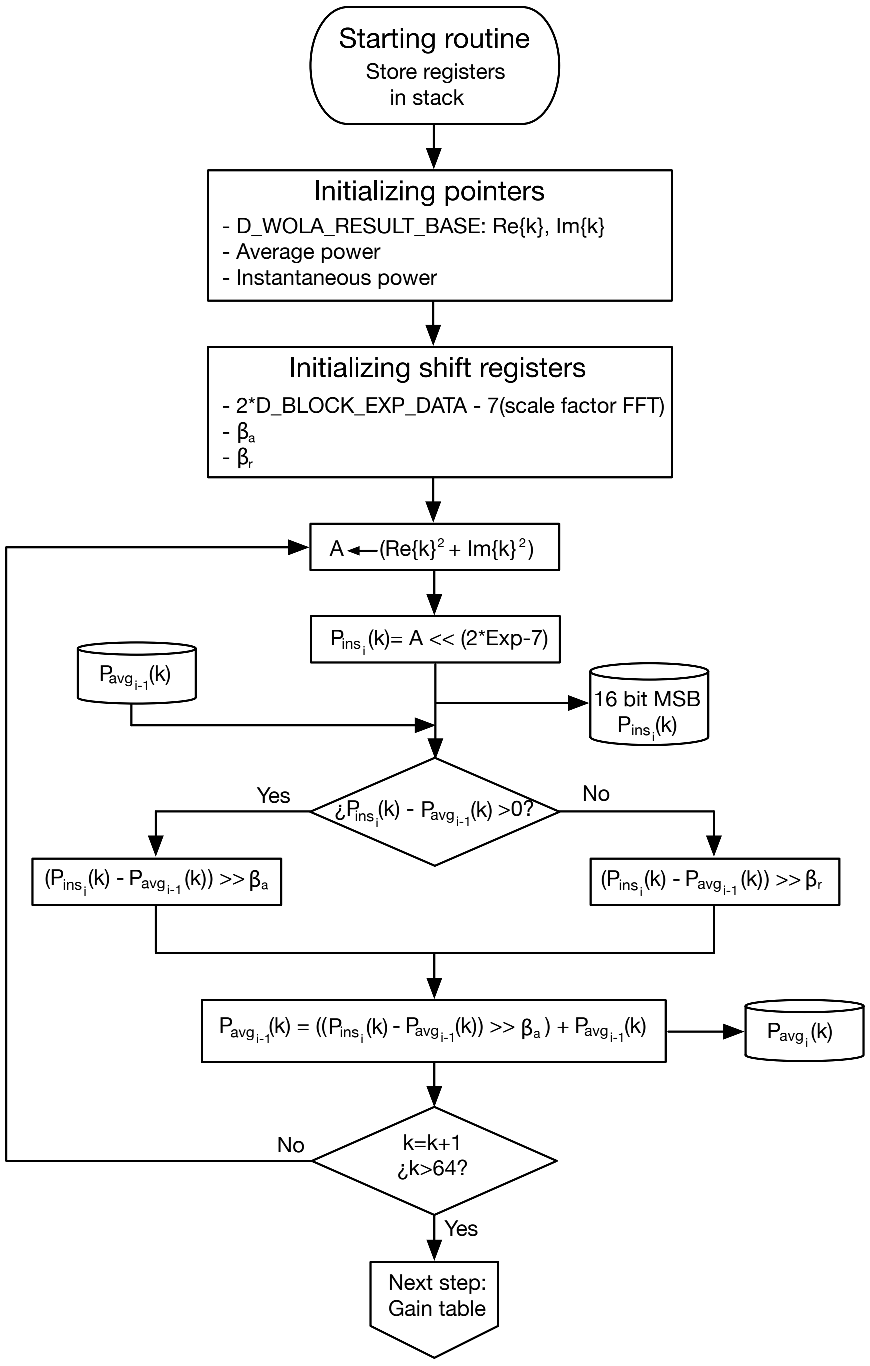

Starting routine
Store registers in stack
Initializing pointers
- D_WOLA_RESULT_BASE: Re{k}, Im{k}
- Average power
- Instantaneous power
Initializing shift registers
- 2*D_BLOCK_EXP_DATA - 7(scale factor FFT)
- βa
- βr
A ← (Re{k}² + Im{k}²)
P_ins_i(k)= A << (2*Exp-7)
P_avg_i-1(k)
16 bit MSB P_ins_i(k)
¿P_ins_i(k) - P_avg_i-1(k) >0?
Yes
No
(P_ins_i(k) - P_avg_i-1(k)) >> βa
(P_ins_i(k) - P_avg_i-1(k)) >> βr
P_avg_i-1(k) = ((P_ins_i(k) - P_avg_i-1(k)) >> βa) + P_avg_i-1(k)
P_avg_i(k)
k=k+1 ¿k>64?
No
Yes
Next step: Gain table

Figure 8.2: An illustrative flowchart representing the main concepts involved in the sequence of operations carried out for properly computing the average power (P_{avg}) in the DSP. In this picture, $k = 1, \ldots, N_{\mathrm{B}}$ labels the index over the N_{B} frequency bands, whereas $i = 1 \ldots, N_{\mathrm{frames}}$ depicts the one over the N_{frames} frames into which any sound signal is segmented within our framework. Note that this sequence of operations is carried out for each of the frames into which the input signal is segmented.

8.2.3 Gain-search in the gain matrix

Undoubtedly, this second step represents the very core of the compression-expansion approach proposed in this thesis and thus, we have put special emphasis in its implementation. To better explain the scheme we have programmed, let us imagine that we are interested in obtaining the gain value to apply for the following scenario:

- The acoustic environment the hearing aid user is in is classified, among the listening conditions considered in this thesis (that is, speech, music and noise), as a "music" environment.
- The study-case frequency is 125 Hz, which involves $k = 1$.
- The average power in that frequency band, that is, $P_{\text{avg}}(1)$, is found to be: $P_{\text{avg}}(1)$ =0×16EA = 0001 0110 1110 1010 = 0.179016.

Please note that we have not included the i-th index in the average power expression, because this study is carried out only for one input frame.

Once we have defined the scenario, we can proceed further with the sequence of operations involved in the search of the gain value in the corresponding gain matrix, which works as follows:

1. **Selection of the gain matrix as a function of the acoustic environment.** As mentioned, there is a gain matrix programmed in the DSP for each of the three audio classes to classify in this thesis. Depending on the decision returned by the classifier, the base address of the corresponding gain matrix is loaded in the pointer `Gain_pointer`. In the particular case at hand, since the acoustic environment the user is in has been classified as "music", the base address of the music gain matrix is loaded in the mentioned pointer, as clearly illustrated in Figure 8.3.

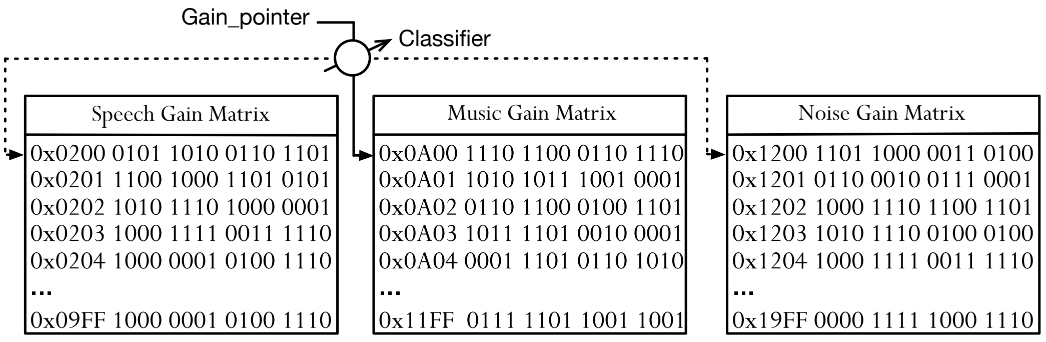

Gain_pointer
Classifier
Speech Gain Matrix
0x0200 0101 1010 0110 1101
0x0201 1100 1000 1101 0101
0x0202 1010 1110 1000 0001
0x0203 1000 1111 0011 1110
0x0204 1000 0001 0100 1110
...
0x09FF 1000 0001 0100 1110
Music Gain Matrix
0x0A00 1110 1100 0110 1110
0x0A01 1010 1011 1001 0001
0x0A02 0110 1100 0100 1101
0x0A03 1011 1101 0010 0001
0x0A04 0001 1101 0110 1010
...
0x11FF 0111 1101 1001 1001
Noise Gain Matrix
0x1200 1101 1000 0011 0100
0x1201 0110 0010 0111 0001
0x1202 1000 1110 1100 1101
0x1203 1010 1110 0100 0100
0x1204 1000 1111 0011 1110
...
0x19FF 0000 1111 1000 1110

Figure 8.3: Depending on the decision returned by the classifier, the base address of the corresponding gain matrix is loaded in the pointer `Gain_pointer`. For the particular example at hand, since the acoustic environment the hearing aid user is in has been classified as "music", the base address of the music gain matrix is loaded in the mentioned pointer.

2. **Gain pointer location in the gain matrix as a function of the number of frequency band.** Since each gain matrix consists of 64 "blocks", one for each of the available frequency bands in the DSP, the next step is to locate the aforementioned pointer (that is, `Gain_pointer`) in the block corresponding to the case-study frequency band. In the example at hand, since

the study-case frequency is 125 Hz (or equivalently, the number of frequency band is 1), the pointer is located at the block corresponding to such frequency band, as clearly shown in Figure 8.4. Please note that each block in the picture consists of 32 words of 16-bit each.

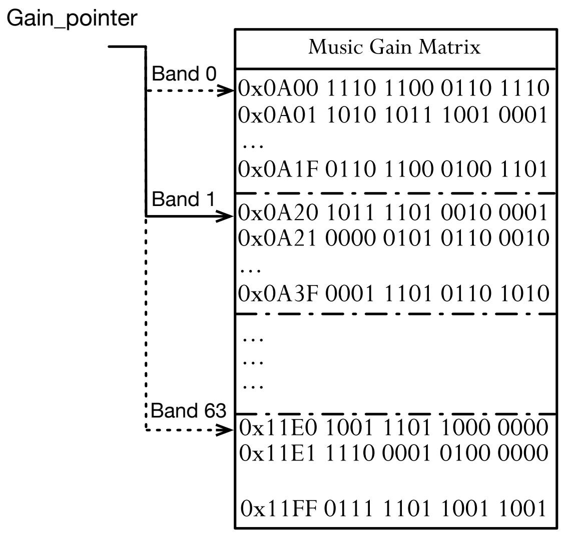

Gain_pointer
Music Gain Matrix
Band 0
0x0A00 1110 1100 0110 1110
0x0A01 1010 1011 1001 0001
...
0x0A1F 0110 1100 0100 1101
Band 1
0x0A20 1011 1101 0010 0001
0x0A21 0000 0101 0110 0010
...
0x0A3F 0001 1101 0110 1010
...
Band 63
0x11E0 1001 1101 1000 0000
0x11E1 1110 0001 0100 0000
0x11FF 0111 1101 1001 1001

Figure 8.4: Each gain matrix consists of 64 “blocks”, one for each of the available frequency bands in the DSP. The pointer `Gain_pointer` is located in the block corresponding to the analysis frequency band. For the particular example at hand, since the study-case frequency is 125 Hz (or equivalently, the number of frequency band is 1), the pointer is located at the block corresponding to such frequency band (named as “Band 1” in the picture).

3. **Conversion of average power (a fixed-point number) into a floating-point number.**
Once we have identified the corresponding block in the gain matrix, the next step consists in locating the gain pointer (`Gain_pointer`) in the exact row in the block. To achieve this, it is convenient for the average power (a fixed-point number) to convert it into a floating-point number, what involves computing the base (we label it B) and the exponent (we label it E) of it by making use of the two expressions shown below, respectively:

$$E[P_{\text{avg}}(k)] = \lfloor log_2(P_{\text{avg}}(k)) \rfloor \tag{8.4}$$

$$B[P_{\text{avg}}(k)] = P_{\text{avg}} \cdot 2^{-E(P_{\text{avg}}(k))}. \tag{8.5}$$

For the particular example at hand, since $P_{\text{avg}}(1) \equiv 0{\times}16\text{EA}$, the corresponding exponent (E) and base (B) are computed as follows:

$$E[P_{\text{avg}}(1)] = \lfloor log_2(P_{\text{avg}}(1)) \rfloor = -2 \tag{8.6}$$

$$\begin{aligned} B[P_{\text{avg}}(1)] = P_{\text{avg}} \cdot 2^{-E(P_{avg_i}(1))} &\equiv 0 \times 5\text{BA8} \\ &\equiv 0101101110101000 \\ &\equiv 0.716064. \end{aligned} \tag{8.7}$$

The negative value of the exponent, in the case at hand, $-E[P_{\text{avg}}(1)] = +2$, is used to locate the pointer `Gain_pointer` in the exact row in the block labelled "Band 1". With this in mind, since the negative value of the exponent is +2, the gain pointer is located in the third row. Perhaps the reader may wonder why in the third row and why not in the second one. Please note that the first row correspond to a negative value of the exponent equal to 0, the second row to a value of 1 and so on.

4. **Obtaining the exponent of the gain**
   The gain value, stored in the DSP memory, is a floating-point number and thus, it consists of a base and an exponent. As mentioned beforehand, the words that compose each block consist of 16 bits, in which the 7 most significant bits (or more precisely, `<15:9>`), codify the value of the exponent of the gain value.

   For the particular example at hand, having a look at the third row in the block corresponding to the frequency band 1, it seems clear to note that the corresponding word is 0000 0111 1001 1100, and thus, the 7 most significative bits have been found to be 0000 011 (base 2) or equivalently, 3 (base 10), which means that the exponent of the gain value to apply is 3. In the effort of better understanding this latter sequence of operations, we have included Figure 8.5, which clearly shows the way the exponent of the gain value to be applied is obtained.

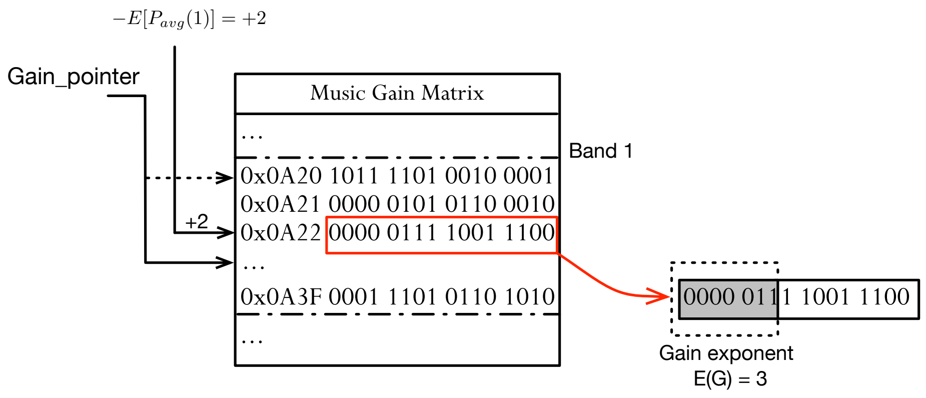


Figure 8.5: Picture depicting the way the exponent of the gain value to be applied is obtained for the example at hand. The 7 most significant bits (or more precisely, `<15:9>`) of the words in each block codify the exponent of the gain value to apply. For the particular example at hand, the corresponding word is 0000 0111 1001 1100, and thus, the 7 most significative bits have been found to be 0000 011 (base 2) or equivalently, 3 (base 10), which means that the exponent of the gain value to apply is 3.

5. **Obtaining the base of the gain**
   As previously mentioned, the 7 most significant bits of each word are used to codify the exponent of the gain value. Perhaps, the reader may wonder if the remaining 9 bits of each word, that is, the 9 lowest significant bits are used to obtain the value of the base of the gain value to apply. The answer to this question is that the reader is not *exactly* right. This will be better understood later on.

   For the purpose of obtaining the base of the gain value, the 9 least significant bits are firstly arranged into 3 groups of 3 bits each, as shown in Figure 8.6. The question arising here is: *which 3-bit group should we use to obtain the base of the gain to apply?*

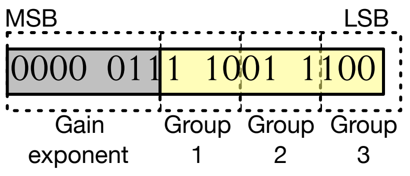


Figure 8.6: The 9 lowest significant bits of each word are arranged into 3 different groups (3 bits per group) in the aim of further obtaining the base of the gain value to be applied.

The group of bits to be used depends on the value of the base of the average power for the each particular frequency band ($B[P_{avg}(1)]$, in our case). Aiming at identifying which 3-bit group we should use, we have defined three different intervals, labelled I1, I2 and I3, according to the values of thresholds TH1 (0×50D7) and TH2 (0×65B9) , as described below:

$$\begin{aligned} \text{I1} &\equiv [0000, 50\text{D}7] \\ \text{I2} &\equiv [50\text{D}8, 65\text{B}9] \\ \text{I3} &\equiv [65\text{BA}, 7\text{FFF}]. \end{aligned}$$

Intuitively, if the numerical value of the base of average power value belongs to the interval I1, the 3-bit group to be used is “group 1” (or more precisely, bits `<2:0>`of the word). Similarly, if it belongs to the interval I2, the 3-bit group to be used is “group 2” (or more precisely, bits `<5:3>`of the word) and finally, if it belongs to the interval I3, the 3-bit group to be used is the “group 3” (or more precisely, bits `<8:6>`of the word). For illustrative purposes, we have included Figure 8.7 that clearly represents this scenario.

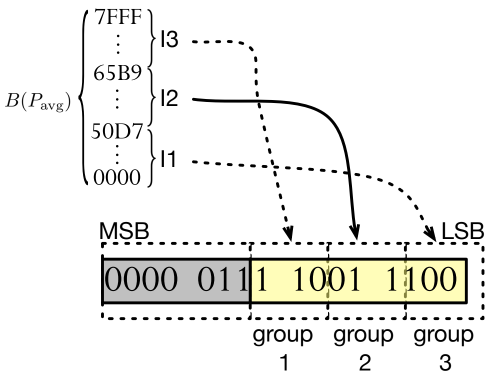


Figure 8.7: An illustrative representation of the way of selecting the corresponding 3-bit group as a function of the numerical value of the base of average power value ($B(P_{avg})$). If this value belongs to the interval I1, the 3-bit group to be used is “group 1”. Similarly, if it belongs to the interval I2, the 3-bit group to be used is “group 2” and finally, if it belongs to the interval I3, the 3-bit group to be used is the “group 3”.

In the study-case at hand, the numerical value of the base of the average power value has been found to be $0 \times 5BA8$ as depicted in Expression 8.7. This value belongs to interval I2 and thus, the 3-bit group to be used is "group 2" (or in other words, bits `<5:3>`).

Each 3-bit group codifies a numerical value ranging from 0 to 7 (base 10). This number *does not* represent the base of the gain value, it *does* represent an index that we use to search in a block of 8 words the value of the base of the gain. Aiming at better understanding the way it works, we have included Figure 8.8 that shows for the particular example at hand, the value of the base of the gain to be applied. As illustrated, the 3-bit group to be used consists of the bits 011 (base 2) or equivalent 3 (base 10). This data is used to search in the block of 8 words, the corresponding value of the base of the gain to be applied. In particular, this numerical value indicates exactly the row in the block that codifies the value of the base of the gain. As shown in this figure, this value has been found to be 0×3A1F.

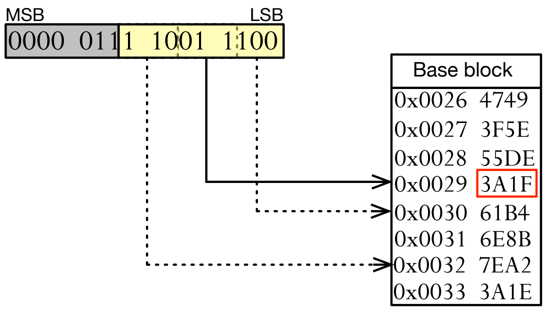


Figure 8.8: The corresponding 3-bit group to be used codifies a numerical value ranging from 0 to 7 (base 10). This number is used to search in a block consisting of 8 words the corresponding value of the base of the gain value. In particular, this numerical value indicates exactly the row in the block that codifies the value of the base of the gain.

With this in mind, we can finally conclude that the gain value for the analysis frame at the frequency band 1 (or in other words, $k = 1$) has been found to be:

$$G(1) = \begin{cases} E[G(1)] \equiv 0 \times 0003 \\ B[G(1)] \equiv 0 \times 3AF1 \end{cases} .$$

Please note that this sequence of operations should be carried out for the remaining 63 frequency bands of the analysis frame.

8.2.3.1 Practical implementation

As stated beforehand, this is probably the most complex task involving the compression-expansion approach. Prior to illustrating the flowchart of its implementation in the DSP, it is important to mention that:

- Apart from obtaining the numerical values of *both* the base and the exponent of gain values for the $N_B = 64$ frequency bands available in the DSP, this routine also computes, among all the gain exponent values obtained, the *maximum* value of them and stores this numerical value in the variable `Max_exp`. For the sake of clarity, we have labelled it "the block exponent" (E_{block}), and as will be clearly explained in the subsection that follows, it will be used in the next routine. Note that the numerical value of the block exponent is stored in the variable `D_GAIN_EXP_DATA`.

- Despite there is a conditional jump instruction and consequently, there are two alternative paths of the program flow, the total loop length is the same in both cases. This is achieved by means of using some waiting loops such as, for instance, the instruction `NOP`, which has *no* effect in the code.

Finally, for properly completing this subsection, Figure 8.9 will assist us in showing the main concepts involved in the sequence of operations carried out for properly searching the gain value in the corresponding gain matrix.

8.2.4 Base adjustment

As advanced in the previous subsection, for each analysis input frame, we have computed the maximum value of the 64 gain exponent values obtained (one for each frequency band available in the DSP), that we have named as "the block exponent" (E_{block}). The reason is as follows. The WOLA filterbank demands that the 64 exponents of the gain values are exactly the same and just in this respect, the value of the block exponent is used as the *common exponent* for the 64 gain values.

The fact of using the block exponent as the common exponent for the 64 gain exponents demands, in an effort ensuring gain values consistency, to shift right a determined number of bits in those gain base values whose exponents differ from the block exponent value. For illustrative purposes, let us imagine a scenario in which, for the sake of simplicity, the total number of frequency bands available in the DSP are only $N_{\text{B}} = 2$, or equivalently, $k = 0$ and 1. With this in mind, let us suppose that the numerical value of the gain base and gain exponent for the frequency band 0 are, for instance, 0×45E2 and −1, respectively, as depicted below:

$$\begin{aligned} B(G(0)) &\equiv 0100010111100010 \equiv 0 \times 45\text{E}2 \\ E(G(0)) &\equiv -1. \end{aligned}$$

Now, let us suppose that the numerical value of the gain base and the gain exponent for the frequency band 1 are, for example, 0×701F and +4, respectively, as shown as follows:

$$\begin{aligned} B(G(1)) &\equiv 0111000000011111 \equiv 0 \times 701\text{F} \\ E(G(1)) &\equiv 4. \end{aligned}$$

In this particular example at hand, the maximum gain exponent is +4, and consequently, the exponent block is set to be $E_{\text{block}} = +4$. Since this is the *common* gain exponent for both gain values, we should shift the gain base value corresponding to the frequency band 0 (0×45E2) right by five bits, as clearly depicted below:

$$B(G(0)) \equiv 0000001000101111 \equiv 0 \times 022\text{F}.$$

With these considerations in mind, the final gain base values for the frequency band 0 and 1 have been found to be:

$$\begin{aligned} B(G(0)) &\equiv 0 \times 022\text{F} \\ B(G(1)) &\equiv 0 \times 701\text{F} \end{aligned}$$

being their gain exponent value as shown in the next page:

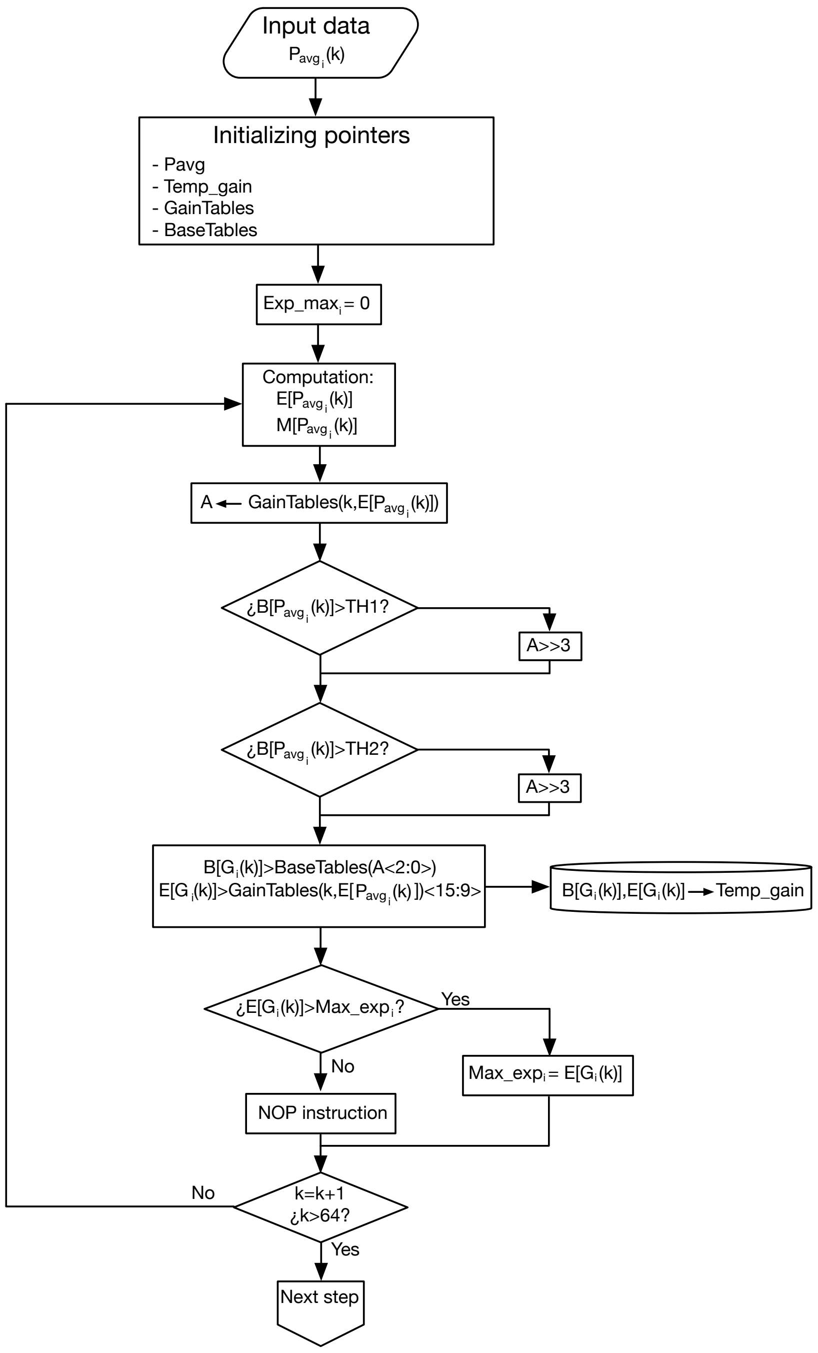


Figure 8.9: An illustrative flowchart representing the main concepts involved in the sequence of operations carried out for properly searching the gain value in the corresponding gain matrix. In this picture, $k = 1, \ldots, N_{\mathrm{B}}$ labels the index over the N_{B} frequency bands, whereas $i = 1, \ldots, N_{\mathrm{frames}}$ depicts the one over the N_{frames} frames into which any sound signal is segmented. Note that this sequence of operations is carried out for each of the frames into which the input signal is segmented.

$$E(G(0)) \equiv +4$$
$$E(G(1)) \equiv +4.$$

Finally, for properly completing this section, Figure 8.10 will assist us in showing the main concepts involved in the sequence of operations carried out for properly implementing the base adjustment process in the DSP.

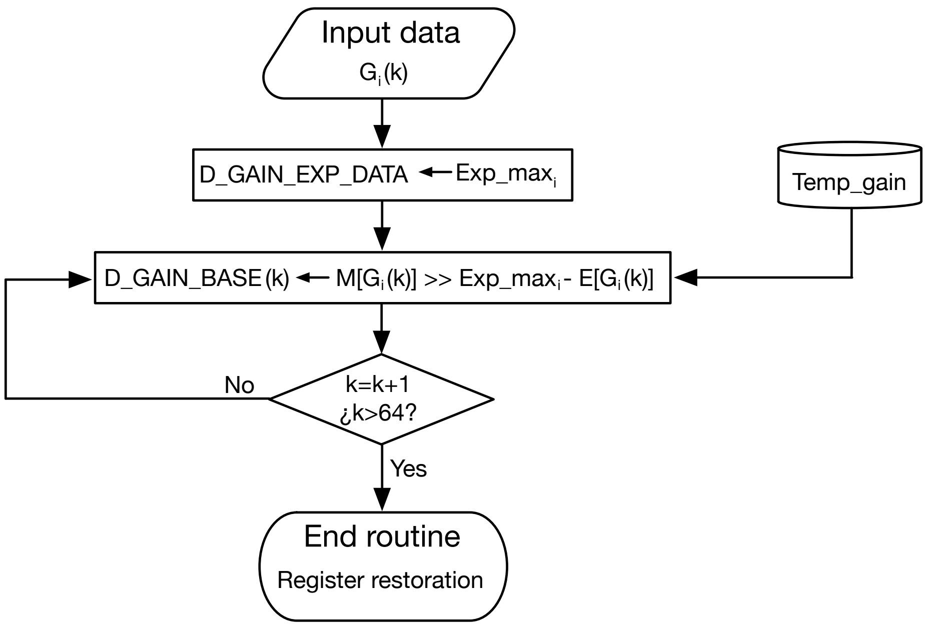

Input data
G_i(k)
D_GAIN_EXP_DATA ← Exp_max_i
Temp_gain
D_GAIN_BASE(k) ← M[G_i(k)] >> Exp_max_i - E[G_i(k)]
No
k=k+1
¿k>64?
Yes
End routine
Register restoration

Figure 8.10: An illustrative flowchart representing the main concepts involved in the sequence of operations carried out for properly implementing the base adjustment process in the DSP. In this picture, $k = 1, \ldots, N_{\mathrm{B}}$ labels the index over the N_{B} frequencies, whereas $i = 1, \ldots, N_{\mathrm{frames}}$ depicts the one over the N_{frames} frames into which any sound signal is segmented.

8.2.5 Computational cost

Once we have explained the way the complete compression-expansion approach works, the question arising here is: *What is the total computational cost required for implementing this approach in the DSP used by our platform to carry out the experiments?* Aiming at answering this question, we have included Table 8.1 that depicts, in a very detailed way, the computational cost (in clock cycles) and the CPU-load average (in %) demanded by our DSP to implement each of the aforementioned blocks, that is, 1) the "average power computation" block, 2) the "gain-search in the gain matrix" block and 3) the "base adjustment" block.

Understanding Table 8.1 demands to mention the following four comments:

- Apart from representing the computational cost required for implementing the three aforementioned blocks, this table also depicts the computational cost of the assembler instructions involved in the implementation of the compression-expansion approach. These instructions are as follows: `Push` and `Pop` assembler instructions, which aim to store and to recover data from the registers, `Call` assembler instruction that allows us to call a subroutine and finally, Ret assembler instruction, which basically assists us in returning from a subroutine or interrupting a service routine (see for further details [Dspfactory, 2002a,b]).
- The column labelled "Fixed" designates the fixed cost (in clock cycles), or in other words, the computational cost associated to each block or instruction that *does not* depend on

Block or instruction	**Computational cost (clock cycles)**			**CPU-load average**
	Fixed	Variable	Total	(%)
`Call`	2	0	2	0.026
`Push`	21	0	21	0.273
Average power computation	15	17	1103	14.36
Gain-search in the gain matrix	18	35	2258	29.40
Base adjustment	12	8	524	6.822
`Pop`	21	0	21	0.273
`Ret`	2	0	2	0.026
Total	91	60	**3931**	**51.18**

Table 8.1: Computational cost (in clock cycles) and CPU-load average (in %) demanded by our DSP to program the three main functional blocks, along with the assembler instructions, involved in the implementation of the compression-expansion approach proposed in this thesis. Note that the total computational cost (that we label it C_{total}) is computed as a function of both the "fixed" cost (that we label it C_{fixed}) and the "variable" cost (that we label it C_{variable}) and adopts the following formulation: $C_{\text{total}} = C_{\text{fixed}} + C_{\text{variable}} \cdot N_{\text{B}}$, being N_{B} the number of frequency bands available in the DSP ($N_{\text{B}} = 64$, in our case).

the number of frequency bands (N_{B}). With this in mind, this computational cost basically corresponds to that involved in the initialization of address registers or counter variables. For the sake of clarity, we label it C_{fixed}.

- The column labelled "Variable" labels the variable cost (in clock cycles), or equivalently, the computational cost demanded by our DSP to process the sequence of operations carried out for each of the N_{B} frequency bands separately. Please note that the number of clock cycles listed in this column correspond to those required for only one frequency band. For the sake of clarity, we label it C_{variable}.

- The column labelled "Total" summarizes the total computational cost (in clock cycles) required by our DSP to implement each block or instruction. It is computed as a function of both the fixed cost and the variable cost, and adopts the following formulation:

$$C_{\text{total}} = C_{\text{fixed}} + C_{\text{variable}} \cdot N_{\text{B}} \tag{8.8}$$

  where:

  – C_{total} is the total computational cost

  – C_{fixed} represents the fixed computational cost

  – C_{variable} labels the variable computational cost

  – N_{B} is the number of frequency bands available in the DSP ($N_{\text{B}} = 64$, in our case).

Thanks to Table 8.1, we can now answer the question that we have asked at the very beginning of this subsection: *what is the computational cost required for implementing the compressor-expander approach in the DSP?* Having a look at the mentioned table, it seems clear to note that the *total* computational cost has been found to be 3931 clock cycles. With this figure in mind, the reader may wonder: *is it a large number of clock cycles?* Aiming at answering this question, we have also computed the CPU-load average (in %), that we label it $\mathcal{L}_{\text{average}}$, by means of the expression stated below:

$$\mathcal{L}_{\text{average}} = \frac{C_{\text{U}}}{C_{\text{T}}} \cdot 100 \tag{8.9}$$

where:

- C_{U} is the number of clock cycles ($C_{\text{U}} = 3931$ cycles, in our case), and
- C_{T} labels the total number of clock cycles.

In this work, the total number of clock cycles (C_{T}) adopts the following formulation:

$$C_{\text{T}} = \frac{f_{\text{CPU}}}{f_{\text{ps}}} \tag{8.10}$$

where:

- f_{CPU} represents the clock frequency of the DSP ($f_{\text{CPU}} = 1.92$ MHz, in our case) and,
- f_{ps} labels the number of sound frames analyzed per second, that is, 16000 samples/ 64 samples per frame $= 250\ s^{-1}$.

  With these considerations in mind, the total CPU-load average has been found to be:

$$\mathcal{L}_{\text{average}} = \frac{3931\,[\text{cycles}] \cdot 250\,[1/\text{s}]}{1.92 \cdot 10^6\,[\text{cycle/s}]} \cdot 100 = 51.18\,\%. \tag{8.11}$$

Note that this the CPU-load average, that is, $\mathcal{L}_{\text{average}} \approx 51.2\,\%$ represents the total computation time required by the DSP to implement the system involved in compensating the particular acoustic loss a hearing aid user suffers from by using the approach proposed in this thesis. As final remark, we can say that there is still left about 50 % of the DSP resources available for implementing other algorithms, such as, for instance, those involved in self-adaptation, as will be explained below.

8.3 Implementing the feature extraction

8.3.1 Introduction

In the aim of selecting the features that properly characterize any input sound for a further classification, Chapter 4 assisted us in discussing those features, with a constrained *maximum* number of assembler instructions, best suited for being finally programmed in the DSP. Note that "best suited" means here those features, with a constrained maximum number of assembler instructions, that make the classifier work as accurately as possible. As mentioned in the Introduction, this set, that we label it $\mathcal{S}_{\text{F}}$, consists of the mean and variance of the following features:

$$\mathcal{S}_{\text{F}} = \{\widehat{\text{SC}}, \widetilde{\text{SC}}, \widehat{\text{V2W}}, \widetilde{\text{V2W}}, \widehat{\text{SF}}, \text{STE}\}.$$

With this in mind, the number of features arranged in the feature vector to be finally programmed in the DSP, labelled $\mathbf{F}_{\text{practical}}$, has been found to be $dim(\mathbf{F}_{\text{practical}}) = 12$. For illustrative purposes, this practical feature vector is shown below:

$$\begin{aligned}\mathbf{F}_{\text{practical}} = \{&\widehat{E}[\widehat{\text{SC}}], \widehat{\sigma}^2[\widehat{\text{SC}}], \widehat{E}[\widetilde{\text{SC}}], \widehat{\sigma}^2[\widehat{\text{SC}}], \widehat{E}[\widehat{\text{V2W}}], \widehat{\sigma}^2[\widehat{\text{V2W}}] \ldots \\ &\ldots \widehat{E}[\widetilde{\text{V2W}}], \widehat{\sigma}^2[\widehat{\text{V2W}}], \widehat{E}[\widehat{\text{SF}}], \widehat{\sigma}^2[\widehat{\text{SF}}], \widehat{E}[\widetilde{\text{STE}}], \widehat{\sigma}^2[\widehat{\text{STE}}]\}.\end{aligned}$$

In the effort of efficiently programming these features in the DSP, we have established a set of strategies and techniques that consists in the use of three "building blocks", included in Table 4.3 (page 67). In particular, the three building blocks we have made use of are the following ones: BB(1), BB(4) and BB(6). They help us analytically define the features included in the set $\mathcal{S}_{\text{F}}$ as a function of them, as clearly explained in the aforementioned chapter. Although will be explained later on, we can say in advance that the basic reason underlying the programming of these building blocks is that, the features included in the set $\mathcal{S}_{\text{F}}$ can be easily calculated by means of the aforementioned building blocks (which *do not* demand high computational cost) and some mathematical operations, such as, for instance, addition or multiplication operations, which are both efficiently implemented in the DSP and thus, it can reduce the computational cost demanded by the DSP for programming the feature vector $\mathbf{F}_{\text{practical}}$.

To better illustrate this, we have included the following subsections that aim to clearly explain the way the features included in the set $\mathcal{S}_{\text{F}}$ are programmed in the DSP used by our platform to carry out the experiments.

8.3.2 Obtaining the building blocks

Although not clear at first glance, it is worth noting that, for each analysis frame i-th, these building blocks, which are calculated for any of the N_{B} frequency bands available in the DSP, basically depend on the instantaneous power of such frame (that is, P_{ins_i}). The importance of this dependence could be better understood if the reader remembers that the WOLA filterbank provides, at its output, both the real and imaginary parts of the spectrum of an input frame, and thus, according to Expression 8.3, computing the instantaneous power is a relatively easy task, as shown below:

$$P_{\text{ins}_i}(k) = \text{Re}\{\chi_i(k)\}^2 + \text{Im}\{\chi_i(k)\}^2 = |\chi_i(k)|^2 \tag{8.12}$$

where:

- $P_{\text{ins}_i}(k)$ labels the k-th frequency band of the instantaneous power at frame i-th,
- $\chi_i(k)$ represents the k-th frequency band of the spectrum at frame i-th, and finally,
- Re{ } and Im{ } depict the real and imaginary parts of the number contained in brackets, respectively.

Deepening a little more in the programming of these building blocks in the DSP, it is worth mentioning three comments:

- They are computed every 64 new samples (note that the input block size is $R = 64$ samples).
- They are represented using signed floating-point precision (16-bit base and 16-bit exponent) in the aim of avoiding saturations or lack of precision in the operations of the DSP.
- In concordance with the programming language used in the DSP, we have labelled these building blocks "BB_XXX$_i$", where "BB" simply labels building block, and "XXX" basically refers to name of the feature. As an illustrative example, "BB_STE$_{i-1}$" would represent the short-time-energy (STE) feature for the frame $i-1$-th.

In the effort of simplifying *even more* the calculation of these building blocks in the DSP, or in other words, reducing the number of assembler instructions, we have mathematically "modified" the mentioned building blocks. To better understand this, we illustrate below the *particular* expression of the building blocks, for each analysis frame i-th and each frequency band k-th, finally implemented in the DSP.

Prior to stating these expressions, it is worth mentioning that we assume that,

- $P_{\mathrm{ins}_i}(k)$ labels the k-th frequency band of the instantaneous power at frame i-th,
- $\chi_i(k)$ represents the k-th frequency band of the spectrum at frame i-th,
- N_{B} represents the number of frequency bands ($N_{\mathrm{B}} = 64$, in our case), and finally,
- M_1 and M_2 designate the indexes that limit the speech band (300-3600 Hz), or equivalently, from a practical implementation point of view, $M_1 = 2$ and $M_2 = 32$.

With these assumptions in mind, we can proceed further with the expressions:

- BB_SSC$_i$: this building block is based on the feature $\widetilde{\mathrm{SC}_i}$ described in Expression 4.2 (page 60), but the difference here is that, for avoiding register saturations, we have divided the expression of the feature $\widetilde{\mathrm{SC}_i}$ into the value $2 \cdot N_{\mathrm{B}}$. With this in mind, the building block BB_SSC$_i$ can be computed as follows:

$$\mathrm{BB\_SSC}_i = \sum_{k=0}^{N_{\mathrm{B}}-1} \frac{k}{2 \cdot N_{\mathrm{B}}} \cdot |\chi_i(k)|^2 = \sum_{k=0}^{N_{\mathrm{B}}-1} \frac{k}{2 \cdot N_{\mathrm{B}}} P_{\mathrm{ins}_i}(k). \tag{8.13}$$

- BB_V2W$_i$: this block building is exactly the same as the $\widetilde{\mathrm{V2W}_i}$ feature described in Expression 4.4 (page 61) and thus, it adopts the following formulation:

$$\mathrm{BB\_V2W}_i = \sum_{k=M_1}^{M_2} |\chi_i(k)|^2 = \sum_{k=M_1}^{M_2} P_{\mathrm{ins}_i}(k). \tag{8.14}$$

- BB_STE$_i$: this last building block is exactly the same as the STE feature described in Expression 3.7 (page 38). In the aim of helping the reader remember the expression, we show its formulation below:

$$\mathrm{BB\_STE}_i = \sum_{k=0}^{N_{\mathrm{B}}-1} |\chi_i(k)|^2 = \sum_{k=0}^{N_{\mathrm{B}}-1} P_{\mathrm{ins}_i}(k). \tag{8.15}$$

8.3.3 Obtaining the features included in the set $\mathcal{S}_{\mathcal{F}}$

Similarly, the features included in the set $\mathcal{S}_{\mathrm{F}}$ that, for the sake of generality and clarity we have named "compound features", are computed every 64 new samples (note that the input block size is $R = 64$ samples). They are also represented using signed floating-point precision (16-bit base and 16-bit exponent) in the aim of avoiding saturations or lack of precision in the operations of the DSP.

In the same line of reasoning as that in the building blocks, we have labelled the features included in the set $\mathcal{S}_{\mathrm{F}}$ "CF_XXX$_i$", where "CF" simply labels compound feature, and "XXX" basically refers to the name of the feature. As an illustrative example, "CF_SSF$_i$" would represent the spectral flux feature for the frame i-th.

Although we explained in Chapter 4 the way these features can be computed as a function of the building blocks, in an effort of making this chapter stands itself, we express below these compound features as a function of the building blocks by using the same labels as those used in their practical implementation in the DSP. For the expressions shown below, we assume that,

- $\chi_i(k)$ represents the k-th frequency band of the spectrum at frame i-th,
- N_B is the number of frequency bands available in the DSP, that is, $N_\mathrm{B} = 64$ in our case, and finally,
- M_1 and M_2 label the indexes that limit the speech band (300-3600 Hz), or equivalently, from a practical implementation point of view, $M_1 = 2$ and $M_2 = 32$.

Once mentioned these assumptions, we can proceed further with the expressions.

- CF_NSC$_i$: this feature is based on the feature $\widehat{\mathrm{SC}_i}$ described in Expression 4.1 (page 60) and can be written as:

$$\mathrm{CF\_NSC}_i = \frac{\sum_{k=1}^{N_\mathrm{B}} k \cdot |\chi_i(k)|^2}{\sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2} \approx \frac{\mathrm{BB\_SSC}_i}{\mathrm{BB\_STE}_i}. \tag{8.16}$$

  Please note that instead of using the equality sign (=), we have used the approximately symbol (≈). This could be better understood if the reader remembers that the expression of BB_SSC$_i$ *does not* exactly represent the same expression as that shown in the numerator of CF_NSC$_i$, because we have included a division operation in BB_SSC$_i$ in the aim of avoiding some register saturations.

- CF_NSC$_i$: this feature is exactly the same feature as $\widetilde{\mathrm{SC}_i}$ feature depicted in Expression 4.2 (page 60), and thus is defined as follows:

$$\mathrm{CF\_SSC}_i = \sum_{k=1}^{N_\mathrm{B}} k \cdot |\chi_i(k)|^2 \approx \mathrm{BB\_SSC}_i. \tag{8.17}$$

- CF_NVW$_i$: this feature is exactly the same feature as $\widehat{\mathrm{V2W}_i}$ feature, described in Expression 4.3 (page 60), and thus is given by:

$$\mathrm{CF\_NVW}_i = \frac{\sum_{M_1}^{M_2} |\chi_i(k)|^2}{\sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2} = \frac{\mathrm{BB\_V2W}_i}{\mathrm{BB\_STE}_i}. \tag{8.18}$$

- CF_SVW$_i$: this feature adopts the same formulation as the $\widetilde{\mathrm{V2W}_i}$ feature described in Expression 4.4 (page 61), and thus can be determined as follows:

$$\mathrm{CF\_SVW}_i = \sum_{k=M_1}^{M_2} |\chi_i(k)|^2 = \mathrm{BB\_V2W}_i. \tag{8.19}$$

- CF_SSF$_i$: this feature is exactly the same feature as the $\widehat{\mathrm{SF}}_i$ feature depicted in Expression 4.5 (page 61), and thus can be computed by means of:

$$\mathrm{CF\_SSF}_i = \sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2 - |\chi_{i-1}(k)|^2 = \mathrm{BB\_STE}_i - \mathrm{BB\_STE}_{i-1}. \tag{8.20}$$

- CF_STE$_i$: this last feature labels the feature STE$_i$ described in Expression 3.7 (page 38) and adopts the following formulation:

$$\mathrm{CF\_STE}_i = \sum_{k=1}^{N_\mathrm{B}} |\chi_i(k)|^2 = \mathrm{BB\_STE}. \tag{8.21}$$

8.3.4 Statistical characterization

As stated in Chapter 3, each of the available features $f_k \in \mathcal{S}_\mathcal{F}$ is applied to each of the frames into which the input audio signal has been segmented to be processed. We complete the *statistical* characterization of the random vector $\mathbf{F_k}$ by estimating its mean value, $\widehat{E}[\mathbf{F_k}]$, and its variance, $\widehat{\sigma}^2[\mathbf{F_k}]$. Finally, this statistical characterization must be done for all the available features $f_k \in \mathcal{S}_\mathcal{F}$. The feature extraction algorithms ends in generating a feature vector that is just the signal-describing vector that finally feeds the classifier. Note that this feature vector is the one mentioned in the Introduction that, for the sake of clarity, we have labelled $\mathbf{F}_{\mathrm{practical}}$.

With this in mind, perhaps the reader may wonder: *how often are the mean and variance operations carried out*? or equivalently, *how often is the classification process carried out*? The answer to this question is intuitively related to the time interval to make and return a decision by the classifier, that is, the length of the time slot. Just in this respect, the batches of experiments carried out in this thesis have been done with files of 2.5 seconds and 20 milliseconds, that is, the time slot for the classifier to make and return a decision has been configured to be 2.5 seconds and 20 milliseconds. According to the results obtained in Chapter 6, the fact of using files of 2.5 seconds was found to exhibit very good results when compared to those obtained for the case of files of 20 milliseconds.

Assuming that the length of files is 2.5 seconds, and consequently, the time slot is 2.5 seconds, a key point to note here is that the number of frames (N_{frames}) per slot time (or equivalently, file) to be processed by the DSP *should* be a number power of two, that is, $N_{\mathrm{frames}} = 1, 2, 4, 8, 16, 32, 64\ldots$ in an effort of significantly reducing the computational cost associated to the calculation of the mean and variance operations in the DSP. In this sense, the subsequent questions arising are: *how many frames are included in files of* 2.5 *seconds*? and consequently, *is that number of frames a power of two*?

Answering this question demands to mention that, for files of 2.5 seconds, the number of samples per file or time slot is given by:

$$2.5 \text{ s/time slot} \times 16000 \text{ samples/s} = 40000 \text{ samples/time slot},$$

and thus, the number of frames per time slot is:

$$N_{\mathrm{frames}} = \frac{40000 \text{ samples/time slot}}{64 \text{ samples/frame}} = 625 \text{ frames/time slot}.$$

By using the abovementioned number of frames per time slot shown above ($N_{\mathrm{frames}} = 625$) and thanks to estimating the logarithm base-2 of that number of frames, we can now answer the question previously stated: *is the number of frames included in files of* 2.5 *seconds a power of two*? Since $log_2(625) = 9.2877$ is not an integer number, we can say that $N_{\text{frames}} = 625$ is not a power of two.

In the effort of estimating the number of frames per time slot for the practical implementation of the classifier in the DSP, we have rounded 9.2877 towards the nearest lower integer, that is, $\lfloor 9.2877 \rfloor = 9$, and consequently, the number of frames per time slot has been found to be:

$$2^9 = 512 \text{ frames/time slot}$$

what, in turns, means that the number of samples per time slot is:

$$512 \text{ frames/time slot} \times 64 \text{ samples/frame} = 32768 \text{ samples/time slot}$$

and consequently, the length of the *practical* time slot is:

$$\frac{32768 \text{ frames/time slot}}{16000 \text{ samples/s}} = 2.048 \text{ s/time slot.} \tag{8.22}$$

Please note that, although in our practical implementation the classifier returns a makes and returns a decision every 2.048 seconds, or equivalently, each 512 frames, it is worth mentioning that this number of frames per time slot can be easily modifiable in the assembler code that we have implemented by any other number provided this number is a power of two.

In this respect, perhaps the reader may wonder whether the fact of using files of 2.048 seconds, instead of files of 2.5 seconds, degrades the percentage of correct classification. In this sense, it has been shown that the fact of using files of 2.048 seconds *does not* degrade the quality of the system to distinguish among speech, music and noise when compared to that obtained when using files of 2.5 seconds [Fernández-Cruza, 2009].

Turning again our attention to the implementation of the abovementioned statistical operators, it is worth mentioning that *all* the operations are carried out by using a 32-bit signed floating-point precision, and the 16 bits most significative will be those that finally feed the classifier. In concordance with the programming language for the DSP, we have labelled these statistical measures FM_YYY_XXX, where "FM" simply labels feature measure, and XXX is the feature in which the statistical operator is applied, and finally, YYY represents the statistical operator applied, that is, AVG for the mean or VAR for the variance. With these considerations in mind, we shall mathematically define these both statistical measures as shown below:

- Average operator:

$$\text{FM\_AVG\_XXX} = \frac{1}{2^{N_{\text{frames}}}} \sum_{i=0}^{2^{N_{\text{frames}}}-1} \text{CF\_XXX}_i \tag{8.23}$$

- Variance operator:

$$\begin{aligned} \text{FM\_VAR\_XXX} &= \frac{1}{2^{N_{\text{frames}}}} \sum_{i=0}^{2^{N_{\text{frames}}}-1} (\text{CF\_XXX}_i - \text{FM\_AVG\_XXX})^2 \\ &= \left(\frac{1}{2^{N_{\text{frames}}}} \sum_{i=0}^{2^{N_{\text{frames}}}-1} \text{CF\_XXX}_i \right) - \text{FM\_AVG\_XXX}_i^2 \end{aligned} \tag{8.24}$$

8.3.5 Practical implementation

We shall explain here the main concepts involved in the feature extraction approach implemented in the DSP in an effort of clearly understanding the assembler source code designed. For further details, we refer the reader to Appendix E.

8.3.5.1 Preprocessor code

Despite the fact of using a feature set consisting of a *determined* number of features, it is convenient to design a source code that is *adaptable* to the compiler options aiming at simplifying future implementations, in which, for instance, we would like to make use of a new spectral feature or just a different feature set.

The fact of designing a configurable assembler code for the feature extraction task involves programming a source code that includes many preprocessor directives. Aiming at having a more structured code, we have created a file including only the definitions of constants and preprocessor variables (we label it `const_features.inc`). The objective of these constants or labels is just that the code automatically generates *itself* in order to include *only* the part of assembler code strictly necessary to execute the routines.

Additionally, as in *all* critical code sections implemented in this thesis (we say critical in the sense of computational cost), we have defined two preprocessor variables as clearly depicted below:

- `FEATURES_CYCLES`: this variable contains the number of clock cycles required to carry out a time-critical routine.
- `FINAL_CYCLES`: this variable contains the number of clock cycles required to carry out a non time-critical routine.

8.3.5.2 Routines

The feature extraction task is carried out in the DSP by means of the following two routines:

- a priority routine, that we label it `Features`, and
- a non-priority routine, that we label it `Final_features`.

Intuitively, we say a *priority* or *non-priority* routine in the sense whether it is a *time-critical* or *non time-critical* routine, respectively. The basic idea underlying the implementation of these two routines is that, although most of computations need to be carried out for *all* the frames comprising the time slot, there are few calculations that need to be carried out *only* for a low number of frames. In this respect, we have implemented a non-priority routine (labelled `Final_features`) that carries out these latter calculations and thus, it reduces the computational complexity associated to the priority routine (labelled `Features`). Despite its *presumably* high use of data-memory storage, it is worthwhile including this non-priority routine because it reduces significantly the computation load associated to the time-critical sections. As will be shown later on, this use of data-memory storage *does not* exceed the restrictions imposed by the DSP used by our platform to carry out the experiments.

We describe in a more detailed way both routines in the paragraphs that immediately follow.

- **Priority routine**
  This priority routine, that we label it `Features`, is the very *core* of the feature extraction

task implemented in the DSP. It is run at each analysis frame update, and since the input parameters to this routine are the instantaneous powers (P_{ins}) computed at each of the N_{B} frequency bands available in the DSP, this routine *must* be finished before the instantaneous powers corresponding to the following frame, that is, the frame $i+1$-th, are computed and thus, these new values replace the instantaneous powers computed at frame i-th, before this routine had finished its execution. Note that the assembler code included in this routine is severely dependent on the labels defined in the abovementioned file `const_features.inc`, in the sense that only the necessary building blocks demanded to calculate the features included in the feature set $\mathcal{S}_{\mathcal{F}}$ will be computed.

This routine basically consists of three different steps clearly distinguished, as illustrated below:

1. Computation of the building blocks defined in Subsection 8.3.2.
2. Computation of the mean and variance of the feature vector.
3. Since the classifier returns a decision every 512 frames, depending on:
   - If the analysis frame is the 512-th frame, we can proceed further with:
     * copying the feature measures obtained in the statistical measure block (variables FM_AVG_XXX and FM_VAR_XXX), which will be the input parameters to the non-priority routine,
     * reseting the variables FM_AVG_XXX and FM_VAR_XXX for being used for the next time slot, and finally,
     * calling to the non-priority routine (whenever there are no interrupts to be handled).
   - If the current frame is *not* the 512-th frame, we execute waiting loops aiming at the function length being the same. Note that the hearing aid is a *hard* real-time system, in which one of the key parameters is the execution time. Therefore, guaranteeing that all the functions are the same length, we will make sure that the priority routines are *always* executed.

- **Non-priority routine**
  This non-priority routine, labelled `Final_features`, basically runs a sequence of instructions aiming to compute some calculations demanded for the priority routine in order to compute the feature vector that feeds the classifier. Aiming at clearly understanding the difference between both routines, we can mention that, whereas the `Features` routine computes the average of mean-square value of the random feature vector, `Final_features` just subtracts the average mean-square value from the average square feature vector for the further computation of the feature vector variance.

  An extremely difference to note here is that this routine is executed every 512 analysis frame updates, whereas the priority routine is executed at each analysis frame update.

8.3.5.3 Computing the statistical measures

At a first glance, implementing in the DSP the assembler code demanded to compute the statistical measures of a vector, that is, the average and variance, could be approached from the two points of view that follow.

The first approach consists in storing in data-memory *all* the features computed for each frame composing the time slot and computing the statistical measures. The main advantage of this approach is that requires *lower computational cost*. The very serious associated drawback here is, however, that it is very difficult its implementation because it requires an *intensive use of data-memory storage*. In particular, this approach would require:

$$512 \text{ frames} \times 6 \text{ features/frames} \times 2 \text{ words/features} = 6144 \text{ words}. \qquad (8.25)$$

The second, feasible approach, which significantly reduces the use of data-memory storage for computing the statistical measures, consists in making use of macros that are run *just after* extracting the feature for the corresponding frame in the slot time. Or in other words, note that the key point in this second approach resides on the fact that not *all* features need to be stored in data-memory. Therefore, the main advantage of this second approach is that it requires a much lower use of data-memory storage at the expense of increasing the computational cost. However, as will be shown in the paragraphs below, this requirement of computation time is perfectly feasible and does not exceed the restrictions imposed by the DSP. These are basically the reasons why we have chosen this approach as a case-study illustrating the feasibility of its implementation.

8.3.5.4 Implementation

Aiming at clearly understanding the implementation of feature extraction process in the DSP, Figure 8.11 represents an illustrative flowchart of it. It is represented the flowchart corresponding to the priority routine (`Features` routine). As shown, there are three different blocks clearly distinguished. Within the application at hand, the first block computes the three building blocks (labelled `BB_x`, where x is the number of building block computed, ranging from 1 to 3), the second one is the block that not only does it compute the features included in $\mathbb{S}_{\mathrm{F}}$ (`CF_y`, where y is the number of feature computed, ranging from 1 to 6) but also its statistical measures, that is, the average and the variance, and finally, the third block is the one that stores in data-memory the feature statistical measures if the current analysis frame is the 512-th frame composing the time slot, or otherwise, it runs waiting loops in the aim of all time-critical routines being the same length.

8.3.6 Computational cost

In the effort of reducing the computational cost demanded by the DSP to implement the feature extraction block, it is worth mentioning three comments:

- The sequence of operations demanded to call and return from a subroutine requires 4 clock cycles (see for further details [Dspfactory, 2002a,b]), which would involve high computational cost because it is very common the programming of subroutines to implement short codes. In this respect, instead of using this sequence of operations to call and return from a subroutine, we have made use of *macros* for the same purpose. The main advantage of using macros is that it requires much lower computational cost at the expense of increasing the amount of data-memory usage. However, this use of data-memory is perfectly feasible and *does not* exceed the restrictions imposed by the DSP.

- The building blocks and the statistical measures calculated are stored in the data-memory available in the DSP in a sequential way, aiming at reducing the number of *redundant* memory accesses. Although this sequential access way increases the average memory access time, it strongly reduces the computational cost by means of using post-increase and post-decrease instructions [Dspfactory, 2002c].

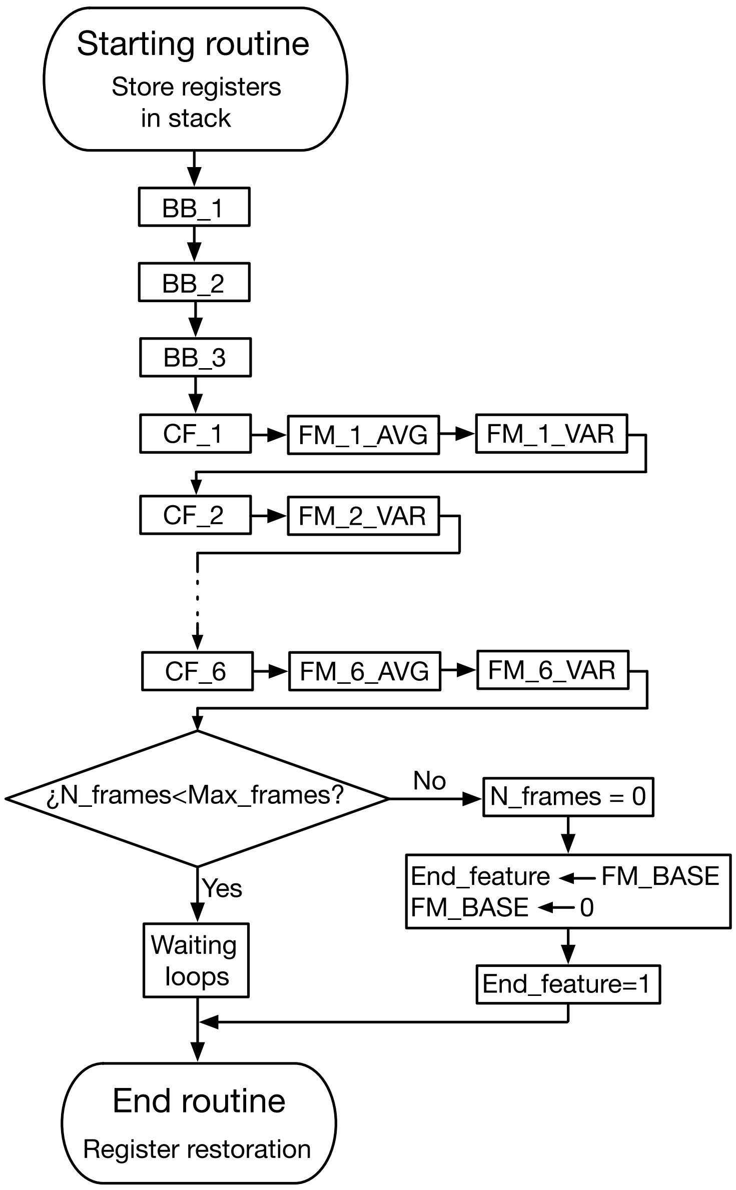


Figure 8.11: An illustrative flowchart representing the way the feature extraction task is programmed in the DSP. It is represented the flowchart corresponding to the priority routine, that we label it `Features`. As shown, there are three different blocks clearly distinguished. The first one, is the block that computes the three simple features (labelled BB_x , where x designates the number of the building block computed, ranging from 1 to 3), the second one is the block that computes the statistical measures, that is, the average and variance of the features (CF_y , where y is the number of novel feature computed, ranging from 1 to 6) used for further classification, and finally, the third block is the one that stores in data-memory the feature statistical measures if the current analysis frame is the 512-th frame comprising the time slot, or otherwise, it runs waiting loops aiming at all time-critical routines are the same length.

- The division operation cannot be carried out directly in the DSP because, it requires, at least, 1) a function to compute the inverse of a number and 2) a function to carry out the multiplication among two numbers. Aiming at clearly understanding this, let us imagine

Building block	**Computational cost (clock cycles)**	**CPU-load average (%)**	**Priority**
BB_SSC	78	1.02	✓
BB_STE	76	0.98	✓
BB_V2W	45	0.59	✓
Total	**199**	**2.59**	✓

Table 8.2: Computational cost (in clock cycles) and CPU-load average (in %) demanded by the DSP, used in the experiments, to program the three building blocks considered in the final implementation, that is, BB_SSC, BB_STE and BB_V2W. It is also shown whether their execution is priority or not, or in other words, if the data involved in their computation are volatile or not. Please note that the symbol ✓ designates that the building block execution is priority, whereas the symbol ✗ labels that the building block execution is not priority.

Compound feature	**Computational cost clock cycles**	**CPU-load average (%)**	**Priority**
CF_NSC	17	0.221	✓
CF_SSC	7	0.091	✓
CF_NVW	17	0.221	✓
CF_SVW	7	0.091	✓
CF_SF	16	0.208	✓
CF_STE	7	0.091	✓
Total	**71**	**0.924**	✓

Table 8.3: Computational cost (in clock cycles) and CPU-load average (in %) demanded by the DSP, used in the experiments, to program the six features considered in the final implementation, that is, CF_NSC, CF_SSC, CF_NVW, CF_SVW, CF_SF and CF_STE. It is also shown whether their execution is priority or not, or in other words, if the data involved in their computation are volatile or not. Please note that the symbol ✓ designates that the execution is priority, whereas the symbol ✗ labels that the execution is not priority.

that we would like to calculate x_1/x_2. Firstly, the inverse of x_2, or in other words, x_2^{-1} is computed, and after that, the multiplication operation between x_1 and x_2^{-1} is carried out, that is, $x_1 \cdot x_2^{-1}$ With this in mind, the DSP *firmware* includes a function that inverts a number, but it requires 13 clock cycles. In the effort of reducing this number of clock cycles, we have *tabulated* the inverse function so that the number of clock cycles is *drastically* reduced to only 5 assembler instructions, and, what is of key importance, without degrading the results.

With these considerations in mind, we can proceed further in explaining the computational cost demanded by the DSP to implement the feature extraction block. On the one hand, Table 8.2 and 8.3 show the computation time (in clock cycles) and the CPU-load average (in %) required to program in the DSP the three building blocks involved in the final implementation (that is, BB_SSC, BB_STE and BB_V2W) and the features included in the set $\mathcal{S}_\mathrm{F}$ (that is, CF_NSC, CF_SSC, CF_NVW, CF_SVW, CF_SF and CF_STE), respectively. For both the building blocks and the features, it is also shown whether their execution is priority or not, or in other words, if the data involved in their computation are volatile or not. Note that the priority code is run in the routine `Features`, whereas the non-priority code is run in the routine `Final_features`. In the aforementioned table, the symbol ✓ designates that the building block execution is priority, whereas the symbol ✗ labels that the building block execution is not priority

On the other hand, Table 8.4 shows the computational cost (in clock cycles) and the CPU-load average (in %) required to compute in the DSP the abovementioned statistical operators,

Statistical measure	**Computational cost (clock cycles)**	**CPU-load average (%)**	**Priority**
`AVERAGE`	9	0.117	✓
`VARIANCE`	12	0.156	✓
`VAR_FINAL`	11	0.0002	✗

Table 8.4: Computational cost (in clock cycles) and CPU-load average (in %) demanded by the DSP, used in the experiments, to compute the statistical operators considered in the experiments, that is, the mean and variance measures. It is also shown whether their execution is priority or not, or in other words, if the data involved in their computation are volatile or not. Please note that the symbol ✓ designates that the execution is priority, whereas the symbol ✗ labels that the execution is not priority.

Block or instruction	**Computational cost (clock cycles)**	**CPU-load average (%)**
`Call`	2	0.06
`Push`	13	0.17
Register preparation	13	0.17
Building blocks	199	2.59
Features	71	0.92
Statistical measures	126	1.64
Return a decision	108	1.41
`Pop`	13	0.17
`Ret`	2	0.06
Total	**547**	**7.12**

Table 8.5: Total computational cost (in clock cycles) and CPU-load average (in %) demanded by the DSP, used in the experiments, to implement the priority routine labelled `Features`, which assists us in computing the feature vector that finally feeds the classifying algorithm. Note that this routine is a time-critical routine.

that is, the mean and variance measures. In the same line of reasoning as that in the building blocks as in the compound features, it is also shown whether their execution is priority or not.

Having a look at the mentioned tables, it is quite remarkable how low the computational cost demanded by the DSP to program the feature extraction task is, especially when compared to that required, for instance, to implement the approach that assigns the gain value to be applied to the input signal.

It is important to note that the figures illustrated *do not* represent the *total* computation time and CPU-load average demanded to compute the feature vector in the DSP. It is also necessary to take into account the number of clock cycles related to run some stack instructions, such as, for instance, `push`, `pop`, `call` or `ret` instructions, which aim to store and to recover data from the registers, to call a subroutine and to return from a subroutine or to interrupt a service routine (see for further details [Dspfactory, 2002a,b]), respectively. Table 8.5 and 8.6 will assist us in addressing this stated issue. They show the total computational cost (in clock cycles) and the CPU-load average (in %) of both priority and non-priority routines, which assist us in computing the feature vector that finally feeds the classifying algorithm.

Thanks to these two tables, we can now say that about 7.13 % represents the total CPU-load average required by the DSP to program the blocks involved in the feature extraction task. Having a look at Table 8.6, one can notice that the computational cost required by the DSP to compute the non-priority routine is *extremely* low. In this respect, perhaps the reader may wonder whether it is worthwhile including this routine or not. The answer to this question is that

Block or instruction	Cost (cycles)	CPU-load average (%)
`Call`	2	0.00005
`Push`	12	0.00030
`VAR_FINAL`	66	0.00168
Storage	76	0.00193
Other instructions	13	0.00033
`Pop`	12	0.00030
`Ret`	2	0.00005
Total	**183**	**0.00465**

Table 8.6: Total computational cost (in clock cycles) and CPU-load average (in %) demanded by the DSP, used in the experiments, to implement the non-priority routine labelled `Final_features`, which assists the priority routine (labelled `Features`) in carrying out some calculations that are not strictly necessary to be computed in the priority routine. Note that this routine is a non time-critical routine.

the fact of *not* implementing this routine would require that the priority routine carried out the sequence of operations included in the non-priority routine, which would involve an additional CPU-load average associated to the priority routine of about 2.4 %.

8.4 Implementing a multilayer perceptron-based classifier

8.4.1 Introduction

As mentioned in Chapter 5, a multilayer perceptron, which is a particular kind of neural network, has been found to exhibit very good results in terms of higher percentage of correct classification when compared to those from other popular algorithms evaluated in this thesis, such as, for instance, the mean square error (MSE) linear classifier, the k-nearest neighbor algorithm, or even, a radial-basis function (RBF) network. One question that may arise regarding multilayer perceptrons is whether an MLP-based classifier can be implemented in real-time on an in-the-market, average-performance device. Elucidating this is just the objective of this section.

Although not clear at first glance, it is worth mentioning that despite its *presumably* high computational cost, its implementation has been proven to be the functional block, among the three main blocks involved in the design of the hearing aid (that is, the compression-expansion approach, the feature extraction task and the classifier *itself*), that demands the lowest computational cost, as will be shown throughout this section. Although will be explained in detail later on, we can say in advance that this is basically due to two main reasons: 1) this task is run every 2.048 seconds, whereas other tasks, such as, for instance, the compression-expansion approach is run for each analysis frame update (or in other words, each 4 ms) and 2) most of the operations needed to run this task are addition-multiplication operations, which are optimized in any DSP.

8.4.2 Creating a multilayer-perceptron

In the effort of implementing an MLP-based classifier in the DSP used by our platform to carry out the experiments, we have designed a *versatile* source code. We say “versatile” in the sense that the fact of adding new neurons to the hidden layer or programming a different activation function for any of the hidden or output neurons is not a complex task. To achieve this, we have programmed a module that includes a *macro* (that we label it `LAYER`), which intends for automatically generating the necessary assembler code required to implement a layer of neurons with an arbitrary number of neurons and any activation function in each neuron. Intuitively, this code will depend on some input user-specific parameters, such as, for instance, the number of neurons or the number of inputs to each neuron.

With this in mind, we shall define this macro as follows:

```
#macro LAYER (Name,n_inputs,n_neurons,Input_pointer_X, ...
      ... , Weight_pointer_Y,Output_pointer_X,ACTIVATION_FUNCTION_MACRO)
```

As clearly shown, this macro demands 7 input arguments, which will be explained in a detailed way in the paragraphs that immediately follow.

- `Name`: this first input parameter is the layer's name. This parameter assist us in naming the labels and compiler variables created for each layer of neurons, which basically allows us to avoid any confusion with the labels and variables corresponding to any other different layer. By using this input parameter, the names of compiler variables adopt the following nomenclature:

  ```
  Variable1_##Name.
  ```

  For illustrative purposes, let us imagine that we make use of the macro in the following way: `CAPA(First,...)`. In this case, the aforementioned variable would adopt the nomenclature `Variable1_First` (see for further details [Archelon, 1994]).

- `n_inputs`: this second parameter labels the number of inputs to each neuron comprising the layer. At this respect, it is worth mentioning that the bias is also taken into account as one input. Please note that this number of inputs is the same for each neuron comprising the layer.

- `n_neurons`: this third parameter intuitively labels the number of neurons composing the layer.

- `Input_data_pointer`: this pointer points at the input data, which are stored in the X memory available in the DSP. Note that the data are 16-bit signed floating-point precision numbers.

- `Weight_data_pointer`: this pointer points at the layer's weights, which are stored in the Y memory available in the DSP. It is worth mentioning that *all* the weights corresponding to a neuron are located in a contiguous way, as shown below:

  ```
  Weight_Input1_Neuron1
  Weight_Input2_Neuron1
  ...
  Weight_InputN_Neuron1
  Weight_Input1_Neuron2
  Weight_Input2_Neuron2
  ...
  Weight_InputN_Neuron2
  Weight_Input1_NeuronM
  Weight_Input2_NeuronM
  ...
  Weight_InputN_NeuronM
  ```

in an effort of reducing the number of *redundant* memory accesses.

- `Output_data_pointer`: this pointer intuitively points at the network outputs, which will be stored in the X memory available in the DSP.

- `MACRO_ACTIVATION_FUNCTION`: this last input parameter designates the activation function used in each neuron comprising the layer.

Figure 8.12 will assist us in better explaining the sequence of operations carried out in the aforementioned macro labelled `LAYER`.

Regarding the activation function of the hidden and output neurons, we have implemented the logarithmic sigmoid function (see for further details Section 3.4.4.2). To achieve this, we have made use of a table stored in the data-memory available in the DSP, which includes 2^N "tabulated" 16-bit values, being N a configurable parameter. Note that, intuitively, the implementation of this table requires storing 2^N memory words. The computational cost demanded by the DSP for implementing this function has been found to be 13 clock cycles.

8.4.3 Running a multilayer-perceptron

Once we have explained in detail the macro `LAYER` that automatically generates the assembler code strictly necessary for creating a layer of neurons in a multilayer perceptron, we can proceed further in explaining the sequence of operations, included in the routine `Execution_MLP`, that we have performed to run the multilayer perceptron.

1. Saving the registers used by the macro `LAYER`, and those used by the routine `MLP_Execution`, for their running.

2. Running the macro `LAYER` to generate the necessary assembler code to create a layer of 20 hidden neurons. Note that the final MLP implemented in the device consists of only one hidden layer of 20 neurons and this is the reason why we create only one hidden layer.

3. Running the macro `LAYER` to generate the assembler code necessary to create a layer of 3 output neurons.

4. Computing the output of the MLP to make a decision. This basically assists us in selecting the correct gain matrix according to the classifier decision. Note that, since there are 3 output neurons, the MLP will return 3 output values. In this sense, the variable `P_Table_Gain_X_L` will point at the gain matrix corresponding to the greater output neuron value. To better explain this, let us imagine that the values returned by the MLP are as follows $[0.3454, 0.6543, 0.7534]$, since the greater value corresponds to the third output neuron, the variable `P_Tabla_Gan_X_L` will point at the "noise gain matrix". On the other hand, if the values returned by the MLP are: $[0.9123, 0.2019, 0.3221]$, the variable `P_Table_Gain_X_L` will point at the "speech gain matrix". Finally, if the values returned by the MLP are: $[0.1498, 0.8765, 0.2312]$, the variable `P_Tabla_Gan_X_L` will intuitively point at the "music gain matrix".

5. Reseting the bit used to call the routine `Execution_MLP` from the main loop.

6. Restoring the value of registers previously saved.

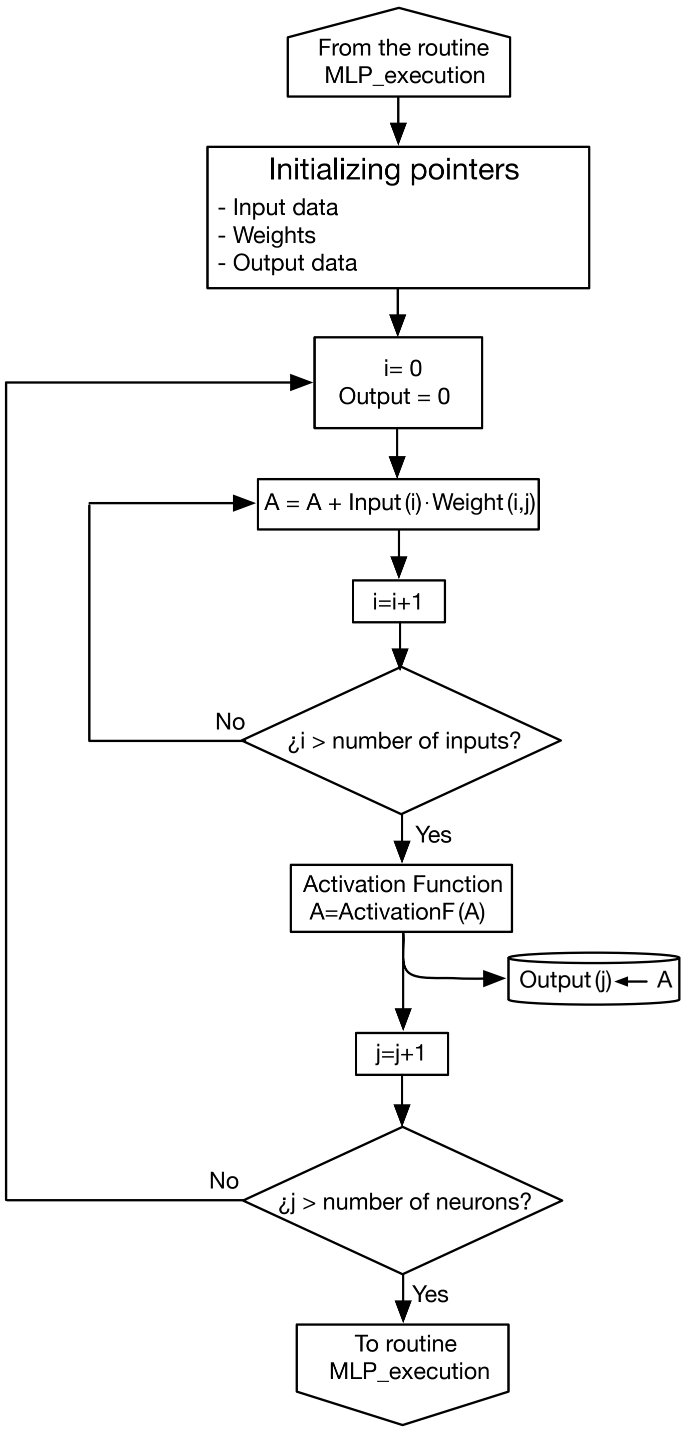


Figure 8.12: An illustrative flowchart that assist us in explaining the sequence of operations carried out in the macro labelled `LAYER`, which basically intends for creating a layer of neurons in a multilayer perceptron.

As stated beforehand, this routine is called from the main loop and is run with the enabled interrupts. This makes this routine a *low-priority* task and thus, it will be run when there are no other critical routines waiting to be run.

For properly completing this section, we have included Figure 8.12 that assist us in explaining the sequence of operations carried out in the aforementioned macro routine `MLP_EXECUTION`.

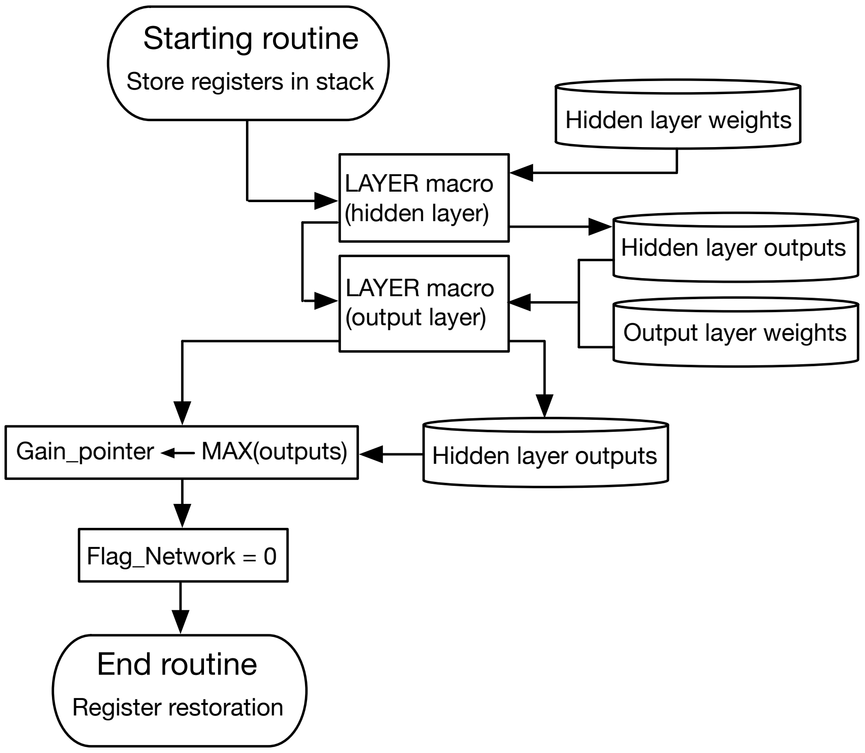


Figure 8.13: An illustrative flowchart that assist us in explaining the sequence of operations carried out in the macro labelled `MLP_EXECUTION`, which basically intends for running a multilayer perceptron.

8.4.4 Computational cost of multilayer perceptron (MLP)

Turning again our attention to the important constrains that digital hearing aids suffer from, it is very convenient to evaluate the computational complexity, or in other words, the number of assembler operations per second required by the DSP, used by our platform to carry out the experiments, to implement an MLP-based classifier. In this respect, it is very important to point out that, only the computational load needed by the DSP to implement the MLP once it has been *trained*, has been considered. The reason is as follows. The computational complexity associated with the MLP training is *not* a critical parameter in real-time applications because this *network learning* is carried out *offline* making use of the all available resources in the laboratory. Only when these network parameters have been computed in the laboratory, they are stored in the data-memory available in the DSP.

Bearing in mind that the final MLP-based classifier implemented in this thesis consists of a *single* hidden layer, we can realistically assume that the MLP consists of L input neurons, M neurons in the hidden layer and C output neurons, one for each audio class to classify (note that, within the context of the application at hand, speech, music and noise have been considered as the audio classes to distinguish, what, in turns, means that $C = 3$).

With these assumptions in mind, Table 8.7 will assist us in explaining the computational cost (in clock cycles) required by the DSP to implement an MLP- based classifier whose design parameters have been computed *offline* [Fernández-Cruza, 2009]. We have chosen a representation that intends for clearly illustrating such computation. In this respect, this table shows the computational cost demanded to implement a layer comprising an arbitrary number of neurons as a function of both 1) the number of input parameters to each neuron (I) and 2) the total number of neurons comprising the layer (N).

Input parameters (I)	**Computational cost (clock cycles)**
2	$9+(6+\mathcal{C}_{\mathrm{AF}})\cdot\mathrm{N}$
3	$11+(7+\mathcal{C}_{\mathrm{AF}})\cdot\mathrm{N}$
4	$11+(8+\mathcal{C}_{\mathrm{AF}})\cdot\mathrm{N}$
5	$11+(9+\mathcal{C}_{\mathrm{AF}})\cdot\mathrm{N}$
$\mathrm{I}(I \geqslant 6)$	$11+(5+\mathrm{I}+\mathcal{C}_{\mathrm{AF}})\cdot\mathrm{N}$

Table 8.7: Computational cost (in clock cycles) demanded by the DSP, used in the experiments, to implement a layer comprising an arbitrary number of neurons (N). In this table, I represents the number of input parameters to each neuron in the layer considered, or equivalently, for the hidden layer, $I = L + 1$, labeling 1 the bias input, and for the output layer, $I = M + 1$. $\mathcal{C}_{\mathrm{AF}}$ represents the computational cost, that is, the number of clock cycles required to implement the activation function. In the particular application at hand, the activation function used in this thesis is the logarithmic sigmoid function, which demands 13 clock cycles for its implementation.

Understanding Table 8.7 requires, at least, three comments:

- I represents the number of *input parameters* to the each neuron comprising the layer. This basically means that if the layer under study is the hidden layer, the number of input parameters to each single neuron can be computed as $I = L + 1$, being L the dimension of the feature vector and 1 designating the bias weight. Similarly, if the layer under study is the output layer, then $I = M + 1$, being M, in this case, the number of hidden neurons and 1 the bias weight. It is also worth mentioning that those neurons corresponding to the input layer have not been considered in the computation since these neurons label just the initial *feature vector* that feeds the classifier and thus no operations are carried out in this layer.

- $\mathcal{C}_{\mathrm{AF}}$ represents the computational cost required to implement the activation function. In our particular case, the activation function of the MLP is the logarithmic sigmoid function, which demands 13 clock cycles for its implementation [Fernández-Cruza, 2009].

- N represents the number of neurons comprising the case-study layer, what, in turns, means that N is set to be $N = M$ for the hidden layer or is set to be $N = C$ for the output layer.

Now, with all these expressions we have described, we can proceed to define the formulation for computing the total computational cost. For this purpose, we also have to take into account the number of clock cycles required to compute the MLP, or equivalently, to make a decision. This final decision is taken by comparing the outputs of the neural network and looking for the greatest values. The number of clock cycles needed to carry out this process can be mathematically written as $7 \cdot C$, being C the number of output neurons [Fernández-Cruza, 2009].

With this in mind, the number of clock cycles required by, a MLP-based classifier based on a multilayer perceptron with a single hidden layer, to produce one output, can now be written as:

$$\mathcal{C}_{\text{MLP}} = \mathcal{C}_{\text{hidden layer}} + \mathcal{C}_{\text{output layer}} + 7 \cdot C \tag{8.26}$$

- **Practical case-study**
  For illustrative purposes, we complete this section by computing the computational cost of the *particular* MLP-based classifier implemented in the DSP. Let us consider an MLP-classifier with $L = 12$ inputs, $M = 20$ hidden neurons and $C = 3$ output neurons, interconnected by links with adjustable *weights*, whose values have been optimized *offline*. Then, *what is the computational cost required by the DSP, used in the experiments, to implement this MLP-based classifier?* As stated in Expression 8.26, the total computational cost can be computed as:

  $$\mathcal{C}_{\text{MLP}} = \mathcal{C}_{\text{hidden layer}} + \mathcal{C}_{\text{output layer}} + 7 \cdot C$$

  where:

  - the computational cost of the hidden layer ($\mathcal{C}_{\text{hidden layer}}$) is:

  $$\begin{aligned}\mathcal{C}_{\text{hidden layer}} &= 11 + (5 + I + \mathcal{C}_{\text{AF}}) \cdot W \\ &= 11 + (5 + 13 + 13) \cdot 20 \\ &= 631 \text{ clock cycles}\end{aligned}$$

  - the computational cost of the output layer ($\mathcal{C}_{\text{output layer}}$) is:

  $$\begin{aligned}\mathcal{C}_{\text{output layer}} &= 11 + (5 + I + \mathcal{C}_{\text{AF}}) \cdot C \\ &= 11 + (5 + 21 + 13) \cdot 3 \\ &= 128 \text{ clock cycles}\end{aligned}$$

  - and finally, the computational cost required to evaluate the greatest neuron is:

  $$\begin{aligned}\mathcal{L} &= 7 \cdot C \\ &= 7 \cdot 3 \\ &= 21 \text{ clock cycles}\end{aligned}$$

  So the total computation cost is:

  $$\mathcal{L} = 631 + 128 + 21 = 780 \text{ clock cycles} \tag{8.27}$$

  or equivalently, it requires a CPU-load average of $\approx 0.02\%$.

8.5 Hearing aid consumption

One of the *crucial* aspects when designing and implementing a real-time hearing aid is the battery life. Although hearing aid users expect that batteries have a longer "working life", battery life may range anywhere from 4 to 7 days, depending on the hearing aid type, as clearly shown in Table 2.2 (page 29).

This probably makes the reader wonders *what is the battery life of the hearing aid designed in this thesis?* Aiming at answering this question, we show here the relationship between the hearing aid current consumption (in mA) and the CPU-load average (in %). With these two numerical values, we can proceed further in calculating the battery life (in hours). Obviously, the lower the CPU-load average is, the lower the hearing aid consumption is and consequently, the higher the battery life is.

In order to calculate the CPU-load average, it is necessary for the assembler code to include an additionally instruction that attempts to count how many times the main loop is run. If we know the main loop length and we count the times that it is run, we can proceed further in calculating the time the processor *is not* running (or in other words, the time the processor *is not* active) and consequently, the CPU-load average. Please note that we send the processor into the "sleep mode" until an interrupt occurs. By using this latter and the *battery monitor* function available in our DSP, we can calculate the hearing aid consumption and consequently, the battery life of our hearing aid.

In the batches of experiments carried out in this thesis to calculate the battery life of our hearing aid, we have considered four different situations as follows:

1. Situation 1: the gain approach and the feature extraction block *are not* programmed in the DSP, that is, *only* the WOLA filterbank, the input/output processor and the core with minimum load are active. With this idea in mind, this basic hearing aid would be a *linear* system, in which we could program the gain for each frequency band available in the DSP. Please note that, within this particular situation, the hearing aid would work as an audio equalizer.

2. Situation 2: the gain approach is implemented but the feature extraction block *is not* implemented in the DSP. With this scenario in mind, the hearing aid performs the compression of input sound signals, but it cannot automatically recognize the acoustic listening the person wearing the device is in, and consequently, it cannot select the amplification program that best fits for such environment.

3. Situation 3: *both* the gain approach and feature extraction functional block are implemented in the DSP. Note that this scenario is just the purpose of this thesis, that is, the situation in which *not only* does the hearing aid perform the compression of input sound signals *but also* recognizes the acoustic listening the person wearing the hearing aid is in and selects the amplification program that best fits to that situation.

4. Situation 4: the processor is not sent into "sleep mode" in the main loop. This situation illustrates the hearing aid consumption when the processor is always running, or in other words, the processor is always *active*. To achieve this, we simply comment the line in the assembler code that sends the processor into "sleep mode", or equivalently, into a low-consumption current mode, until some event occurs.

Once we have summarized the set up of the experimental work, Table 8.8 shows the CPU-load average (in %), the current consumption hearing aid (in mA) and the battery life (in hours) obtained for the four aforementioned situations.

Please note that the CPU-load average 71 % represents the total computation load required by the DSP to implement "situation 3", which involves programming the gain approach, the feature extraction and classification functional blocks, the very *core* of this thesis. The hearing battery life for this scenario has been found to be 140 hours, or in other words, approximately 6

Situation	**CPU-load average (%)**	**Current consumption [mA]**	**Battery life (hours)**
Situation 1	1	0.5	182
Situation 2	57	0.63	143
Situation 3	71	0.65	140
Situation 4	100	0.71	128

Table 8.8: Numerical values of CPU-load average (%), current-consumption (mA) and battery life (hours) corresponding to the digital hearing aid implemented in this thesis for four different scenarios considered in the experiments. “Situation 1” labels the scenario in which both the gain approach and the feature extraction functional blocks are *not* implemented in the DSP. “Situation 2” represents the case in which the gain approach is implemented but the feature extraction block is *not* implemented in the DSP. “Situation 3” illustrates the scenario in which *both* the compressor-expander and feature extraction functional blocks are implemented in the DSP, and finally, “situation 4” depicts the case in which the processor is always active (it is never sent to “sleep mode”).

days, being a reasonably value when compared with that corresponding to current in-the-market hearing aids. Even for “situation 4” (the worst case in terms of CPU-load average) the hearing aid battery life has been found to be 128 hours, or equivalently, 5 and a half days.

Furthermore, the results shown in the table “agree” with the manufacturer’s data. Clearly, this can be inferred by comparing the nominal power consumption of integrated circuit, when the core is always active, provided by the manufacturer and whose value has been found to be 0.47 mA, with that corresponding to that illustrated in the table for situation 4, which has been found to be 0.71 mA. Although not clear at first glance, it is worth mentioning that results provided by the manufactures are obtained for a clock frequency of the DSP of 1.28 MHz, whereas the results depicted in the aforementioned table are obtained for a clock frequency of the DSP of 1.92 MHz. Therefore, the current consumption of the hearing aid provided by the manufacturer for a clock frequency of the DSP of 1.92 MHz is given by:

$$0.47\,\text{mA} \cdot \frac{1.92\text{ MHz}}{1.28\text{ MHz}} = 0.705\text{ mA} \approx 0.71\text{ mA} \tag{8.28}$$

that leads our results make sense.

Finally, we can deduce a *practical* expression for the relationship between the current consumption of our hearing aid and the CPU-load average, as illustrated below:

$$\text{Consumption} = 0.21 \cdot \frac{\text{CPU-load average}\,(\%)}{100} \quad (\text{mA}). \tag{8.29}$$

8.6 Conclusions

This chapter has been motivated by the fact that implementing an automatic sound classification system in DSP-based hearing aids is strongly limited in terms of computational capacity, memory and battery. In this respect, we have shown here that the hardware implementation of the three main blocks involved in the design of a digital hearing that automatically recognizes the acoustic environment its user is in and selects the amplification program that best fits that situation (that is, the compression-expansion algorithm, the feature extraction task and the classifying algorithm) has been proven to be *feasible*. The experimental work carried out in this thesis leads to the following conclusions:

- We have proposed here a *novel* compression-expansion approach that significantly reduces the CPU-load average required for its implementation. In particular, this approach demands a CPU-load average of about 52 %, which involves there is still left about 50 % of the DSP resources available for implementing *both* the feature extraction and classification functional blocks. The only drawback here consists in the fact of the large amount of memory required. However, this stronger use of data-memory is perfectly feasible and does not exceed the restrictions imposed by the DSP.

- The number of clock cycles required for implementing in the DSP the feature extraction functional block has been found to be 734, which represents a CPU-load average of about 7.13 %.

- Despite the *presumably* high computational cost associated to implement a multilayer perceptron in the DSP, it has been found that its implementation is the functional block that requires the *lowest* CPU-load average. In particular, the CPU-load average demanded for programming the particular MLP-based classifier considered in the hardware implementation has been found to be about 0.02 %.

- The idea of "tabulating mathematical functions", such as, for instance, the function that computes the inverse of a number, has been proven to reduce drastically the computational cost without degrading the results. As an illustrative example, we can mention that the function that computes the inverse of a number, provided by the DSP *firmware*, requires 13 clock cycles, whereas the "tabulated" inverse function that we have designed requires 5 clock cycles.

- The battery life of the real-time digital hearing aid implemented in this thesis has been found to be 140 hours, or in other words, approximately 6 days. This is a very reasonable value when compared with that corresponding to current in-the-market digital hearing aids.

The final, global conclusion is that the real-time implementation of a digital hearing aid that automatically recognizes the acoustic environment its user is in and selects the amplification program that best fits that situation is perfectly feasible, and what is of key importance, there is still left about 30 % of the DSP resources available for implementing other algorithms such as, for instance, those involved in sound source localization or acoustic feedback reduction.

Chapter 9

Conclusions

9.1 Introduction

The overall aim of this thesis was to design and implement a *prototype* for a digital hearing aid, which aimed at automatically classifying the acoustic environments that surrounded its user into three classes: *speech*, *music* and *noise*, and selected the "amplification program" best adapted to such environments ("self-adaptation"), in an effort of improving, not only the speech intelligibility, but also the sound quality perceived by the hearing aid user. These three acoustic environments (speech, music and noise) were found to be the most important ones for the majority of hearing-impaired people in their quotidian life.

The *motivation* of this thesis basically arose from the fact that the sound classifier for self-adaptation was deemed as very appreciated by most of hearing aid users, specially by those eldest, because the "manual" approach (in which the user had to identify the acoustic listenings that surrounded him or her and then choose the most appropriate program) was very uncomfortable, and very often exceeded the abilities of many hearing aid users. This illustrated the necessity for hearing instruments to automatically classify the acoustic environment the user was in. Intuitively, this type of hearing aid could help the user *improve speech intelligibility, increasing his or her comfort level*, and *allowing him or her to lead a normal life*. This social demand was what compelled us to develop in this thesis an automatic sound classifier embedded in a self-adapting hearing aid that aimed at classifying sounds into the three mentioned classes: speech, music and noise.

With this idea in mind, the current chapter summarizes the findings of this thesis, "revisiting" the research objectives shown in the introductory chapter. First, Section 9.2 describes the contents of this study, outlining the main conclusions that were obtained from each chapter. The main contributions of this thesis in sound classification in real-time digital hearing aids are listed in Section 9.3. Recommendations for future lines of research, in the sense of how to progress with this research study, are discussed in Section 9.4. Additionally, the final section includes a list of published works occurred during the course of candidature of the degree.

9.2 Summary

As advanced in the introductory chapter, the problem of implementing an automatic sound classifier in a digital hearing aid was that the DSP it is based on has generally very considerable constraints in terms of computational capability, memory and especially battery life[1]. Addition-

[1]Note that the hearing aid has to work at low clock rates in order to minimize the power consumption and thus, maximize the battery life.

ally, these restrictions become stronger because a considerable part of the DSP computational capabilities is already being used for running the algorithms aiming to compensate the particular hearing loss the user suffers from. Therefore, the implementation of any automatic sound classifier is strongly constrained to the use of the remaining resources of the DSP. This restriction in number of operations per second enforced us, throughout this entire thesis, to put special emphasis on designing *signal processing techniques and algorithms tailored for properly classifying sounds in hearing aids while using a reduced number of assembler instructions.*

In the effort of facilitating the reader's comprehension of this thesis, Chapter 3 described the blocks involved in a typical automatic sound classification system for hearing aids, which basically include the data processing stage, the feature extraction block and, finally, the classifying algorithm *itself*. For the sake of clarity, the basic concept underlying the data processing stage, along with the cornerstone ideas describing the feature extraction block and the classifying algorithm were described in-depth in the mentioned chapter. With this in mind, this chapter presented a summary of a set of classical, spectral features proposed in the literature that have been widely-used for the purpose of sound classification in hearing aids. Regarding the classifying algorithms, this chapter also described in a clear way a variety of widely-used classifiers that exhibit good performance in sound classification in hearing aids. These classifiers include the mean square error (MSE) linear classifier, the k-nearest neighbor (k-NN) classifier and two particular kinds of neural networks: multilayer perceptrons (MLPs) and radial basis function (RBF) networks.

As detailed in Chapter 4, some of the features proposed in the literature, in particular, spectral centroid, voice2white, spectral flux and short-time-energy, were used in this thesis aiming to design a set of novel, low-complexity features, based on the *variation* of the aforementioned ones for a given input frame. Although other spectral features (such as, for instance, Mel-frequency cepstral coefficients features) could have been used, we deliberately restricted the study to the variation of the aforementioned ones because they are well-known in hearing aid applications, and they can be computed on them with low computational cost. Turning again our attention to the set of novel, low-complexity features, it is worth mentioning that the key point that exhibited the greatest relevance of those features, when contrasted with the original ones, was that some complex mathematical operations (or in other words, operations that demanded high computational cost), such as, for instance, the division or the square root operations, *were not* needed for their calculation, what, in turns, meant that the computational cost associated to their computation in the DSP was drastically reduced. Regarding their classification performance, the preliminary classification results obtained in this chapter clearly proved that these novel features achieved similar results, in terms of percentage of correct classification, than those obtained with the original ones.
Alternatively, in an double effort of both 1) reducing the number of features to be programmed in the DSP, and 2) selecting the proper features that contained the kind of audio signal information that allowed the classifier to properly distinguish among the mentioned three classes, Chapter 4 also described a feature-selection approach proposed in this thesis for searching the best feature vector $\mathbf{H}$ that helped the classification system perform a proper classification (or equivalently, maximize the percentage of correct classification) with a constrained *maximum* number of assembler instructions demanded by the DSP to compute the vector $\mathbf{H}$. The experiments carried out in this chapter illustrated that the features chosen by our approach, for scenarios in which the *maximum* computational cost allowed was relatively low, were the mean and variance of the novel features proposed here, along with the mean and variance of short-time-energy feature.

Concerning the classifying algorithm *itself*, we chose, among a variety of classifiers explored in Chapter 3, a multilayer perceptron (a particular kind of neural network), commonly known as MLP, which was *properly* designed for being constrained to the hardware requirements of the DSP used by our platform to carry out the experiments. The key reason that compelled us to choose the MLP approach is that multilayer perceptrons are able to learn from an appropriate set of training patterns, and properly classify other patterns that have never been found before. This ultimately leads to very good results, as a balance between percentage of correct classification and computational cost, when compared to those from other popular algorithms evaluated in this thesis, such as, for instance, the MSE linear classifier, the k-NN algorithm, or even, an RBF network.

Nevertheless, one argument against the feasibility of multilayer perceptrons for being used in DSP-based hearing aids consisted in its, *a priori*, high computational complexity, which among other topics, was intimately related to the network size. To be more precise, this complexity depended on the number of weights that needed to be adapted, and consequently, on the number of neurons that composed the multilayer perceptron, that is, the number of input neurons (L), the number of hidden neurons (M) and the number of output neurons (C). In the particular case at hand, the number of input and output neurons seemed to be clear: the input neurons designated the dimensionality of the feature vector inputting the network, which had been experimentally found to be $dim(\mathbf{F}) = 12$, and the number of output neurons was the number of acoustic environments to classify, being in our case, speech, music and noise (and thus, $C = 3$). With these considerations in mind, it seemed evident that the only way to reduce the computational cost demanded to implement the MLP in the DSP consisted in determining the *proper* number of neurons in the hidden layer. In this respect, Chapter 5 depicted a simple approach that made use of both growing and pruning algorithms proposed in this thesis, and aimed to estimate the appropriate number of hidden neurons in a multilayer perceptron. The batches of experiments carried out in this chapter illustrated that the suitable number of hidden neurons, when the abovementioned 12-feature vector fed the MLP-based classifier, had been found to be $M = 20$.

Another important point to note here was that a topic that played a key role in reducing the computational cost associated to the implementation of a multilayer perceptron in the DSP was the "simplification" of the activity function used in each of the hidden and output neurons. To better understand this, it is worth mentioning that the activation function used in our experimental work was the logarithmic sigmoid function, commonly known as "logsig" which, in order to be implemented in the DSP required a proper approximation (since it demanded 25 assembler instructions for its practical implementation in the DSP) and that was the reason why we also explored in this chapter some "approximations" for this activation function. In particular, we proposed two approximations that were based on a piecewise linear approximation, these being: 1) a 5-piece linear approximation and 2) a 3-piece linear approximation. Additionally, we explored the benefits of using two more approximations: 3) the so-called *hard-limit* function that could be seen as a "2-piece linear approximation" and 4) a *null* function, which helped us prune unnecessary hidden neurons. The practical point to note here was that the "5-piece linear approximation" and the "3-piece linear approximation" required for its implementation 5 and 4 assembler instructions, respectively, whereas the *hard-limit* function demanded only 1 instruction and the *null* function intuitively needed 0 instructions. Note that the additional bonus regarding these approximations consisted in that the number of instructions had been drastically reduced when compared to that of the original activation function. In the effort of automatically selecting the most appropriate approximation (among the proposed ones) for each of the hidden and output neurons composing the MLP, we proposed a method that aimed at automatically selecting the approximation "best suited" to each of the mentioned neurons. The feasibility of the method was proven by comparing the results obtained with the "approximated" MLP with those obtained with the "exact" MLP. In this respect, we can say that the "3-piece linear approximation" was the approximation that led to the lowest average decrease in the percentage of correct classification, and thus, this was the approximation selected for the practical implementation in the DSP of the activation function of each of the hidden and output neurons. As previously mentioned, this method also helped us *prune* unnecessary hidden neurons in the MLP, what, in turns, meant that the computational cost required for implementing such network was also reduced since it was not necessary for the MLP to be implemented all its hidden neurons and their associated weights.

Generally, many algorithms that are designed for being further programmed in DSPs are firstly designed using no-quantization, double floating-point precision, which usually offers both large dynamic range and high precision for each numerical computation. However, from the practical implementation point of view, the "final model" barely makes use of no-quantization, double floating-point precision formats, and consequently, these algorithms need to be "redesigned" for being implemented in some quantized finite-precision format.
In this respect, Chapter 6 clearly quantified the effects of the finite-precision limitations on the performance of the automatic sound classification system for the hearing aid. In particular, this chapter detailed the effects of finite-precision word length for the sound signal-describing features and the weights of the MLP. Bearing in mind that the DSP used by our platform to carry out the experiments makes use of a 16-bit-word-length Harvard Architecture, we used a 16-bit fixed-point format in the batches of experiments carried out in this thesis. With this format in mind, the numbers are usually represented by using the so-called $Qx.y$ format, x being the number of bits used to represent the 2's complement of the integer part of the number, and y representing the number of bits used to represent the 2's complement of the fractional part of such number.
The results obtained in this chapter confirmed that the lowest relative increase in the error percentage was achieved by using the Q5.11 quantization scheme. Making use of this quantization scheme, we quantified the aforementioned 12-feature vector and the weights of the MLP-based sound classifier, and the results obtained clearly proved that the quantized MLP-based classifier achieved perceptually the same efficiency as that reached by the "exact" MLP (without quantization), or in other words, the performance of the classification system that would be further programmed in the DSP had not *appreciably* degraded.
This chapter also illustrated the influence of using different time slots for returning a classification decision by the classification system. It was shown that the fact of employing relatively large time slots, such as, for instance, files of 2.5 seconds, exhibited very good results when compared with those obtained for shorter time slots, such as, for instance, files of 20 milliseconds.

Chapter 7 presented a *novel* approach aiming at speech enhancement perceived by hearing aids users, not only in terms of speech quality but also in terms of speech intelligibility. As shown in this chapter, the approach basically consisted in automatically generating -by making use of perceptual concepts, and a supervised learning process driven by a genetic algorithm- a gain matrix that aimed at enhancing speech perceived by hearing aid users. To achieve this, we made use of two different standardized objective measures: the perceptual evaluation of speech quality (PESQ) and the speech intelligibility index (SII), which aim to evaluate speech quality and speech intelligibility, respectively. To be more precise, the proposed algorithm created the enhanced gain matrix by using a gaussian mixture model driven by a genetic algorithm, and did not use any initial information but that related to "perceptual concepts", or in other words, just those that could be iteratively quantified by the PESQ measure (that is, the fitness function was based on the PESQ measure) or SII measure (or equivalently, the fitness function depended on the SII measure). This gain matrix under construction (and, in turns, the PESQ or SII scores) depended on a number of parameters. In this respect, the GA computed the "optimized" parameters that maximized these scores and consequently, the speech "perceived by the human auditory system". The results obtained in this chapter depicted that the computed, perceptual-based gain matrix made the hearing aid reach better results in terms of speech quality and speech intelligibility perceived by the user (regardless the number of gaussian components used in the mixture and the user's hearing loss) than those measured when the user made use of a hearing aid in which the gain is based on a piecewise linear approximation for each frequency band. A key point to note here was that when the approach made use of the fitness function based on the PESQ measure, not only the speech quality perceived by the user was improved but also the speech intelligibility. However, when the approach made use of the fitness function based on the SII measure, the speech intelligibility was always increased regardless the user's hearing loss, but the speech quality was not always enhanced. This led to design the approach by basing the fitness function of the genetic algorithm on the PESQ measure, what made it more speech enhancement-efficient.

Finally, Chapter 8 depicted the feasibility of implementing the MLP-based sound classification system in the DSP used by our platform to carry out the experiments. The implementation of this classifier basically involved implementing the following three *basic* functional blocks: the compression-expansion algorithm, the feature extraction block and the MLP-based classifier. In this respect, it is important to highlight that the practical implementation of the compression-expansion algorithm *did not* involve implementing the typical compression system, which is based on a the aforementioned piecewise linear approximation for each frequency band. In the effort of reducing the computational cost demanded for its implementation, we made use of a completely different approach, consisting of a table stored in data-memory available in the DSP, which contained "tabulated" values of gain to be applied as a function of *both* the input signal level (dB SPL) and the frequency band. The key point to note here was that this approach required about 51.2 % of the DSP resources for its implementation. Concerning the feature extraction block and the classifying algorithm *itself*, it is worth mentioning that the computational cost demanded for their implementation was found to be 7.13 % and 0.022 %, respectively. Regarding the battery life, the most important constraint for designing advanced algorithms in hearing aids, it is worth mentioning that it was found to be 140 hours, or equivalently, approximately 6 days. This was a very reasonable value when compared with those corresponding to other medium-cost hearing aids in the market. These figures clearly illustrated the feasibility of designing a prototype for a digital hearing for automatically classifying acoustic environments, and what is of key importance, there was still about 30 % of the DSP resources available for implementing other algorithms, such as, for instance, those involved in sound source separation or acoustic feedback reduction.

9.3 Contribution to knowledge

This section illustrates how this research work has contributed to the design of a prototype for a digital hearing aid that aims to automatically classify the acoustic environments that surround its user and select the amplification program that is best adapted to such environment. This basically involves the implementation of a compression-expansion algorithm, a feature extraction block and the classifying algorithm *itself* in the DSP used by our platform to carry out the experiments. The contributions are listed as follows:

- **A set of novel, low-complexity features** inspired by four spectral features (spectral centroid, voice2white, spectral flux and short-time energy), which are widely-used for sound classification in hearing aids. The key point of the greatest relevance to note here is that the

computational cost associated to their computation in the DSP is drastically reduced when compared to that of the original ones, and what is of key importance, **feeding the complete classification system with these low-complexity features does not degrade the classification performance**.

- A **feature-selection** approach for **sound classification in hearing aids** through **restricted search driven by genetic algorithms**. We presented in Chapter 4 an approach that illustrates the benefits of using a genetic algorithm that restricts the space search for the mentioned feature selection. To be more precise, given $\mathcal{F}$ the set of available, general-purpose features, the goal consists in finding a subset $\mathcal{H} \subseteq \mathcal{F}$ that maximizes the percentage of correct classification with a constrained maximum number of instructions required to program the feature vector $\mathbf{H}$ in the DSP. This may permit these selected features to be programmed in the DSP that the hearing aid is based on, and to **make efficient use of the limited computational capabilities**.

- A **combined growing-pruning method for multilayer perceptrons training**. The explored method allows us to design a reduced multilayer perceptron without degrading the ability of the system to classify among the three acoustic listenings considered in this thesis. The **experimental comparison** between the results obtained with a methodology proposed in the literature for the same purpose and those achieved with the method proposed in this thesis **shows that our method made the multilayer perceptron reach better results in terms of classification performance** with the added bonus of **reducing the computational cost** on the DSP the hearing aid is based on. This method was presented in Chapter 5.

- A **piecewise linear approximation for the logarithmic sigmoid activation function**. In an effort of reducing the computational cost associated to the programming of the activation function of each of the hidden and output neurons composing a multilayer perceptron, Chapter 4 illustrated that the use of a piecewise linear approximation as activation function made the multilayer perceptron **achieve very similar results than those obtained when using the original activation function, with the additional advantage of reducing the computational cost**. In particular, the "3-piece linear approximation" is the approximation that leads to the lowest relative decrease in the percentage of correct classification.

- An **algorithm for automatically selecting**, among the approximations proposed in this thesis, **the "approximated" activation function best suited for each of the hidden and output neurons comprising a multilayer perceptron** was proposed in Chapter 5. The feasibility of this algorithm was proven by comparing the classification results obtained with the MLP that makes use of the original logarithmic sigmoid function with those obtained with the "approximated" MLP (or in other words, the MLP that makes use of the approximated activation functions). Additionally, **this algorithm also helps us "prune" unnecessary hidden neurons in the multilayer perceptron**, what, in turns, means that the **computational cost** required for implementing such MLP **is reduced even more** since it is not necessary for the MLP to be implemented all its hidden neurons and their associated weights.

- An **approach aiming at speech enhancement in hearing aids**, not only **in terms of speech quality** but also in terms of **speech intelligibility**. As illustrated in Chapter 7, this approach basically consists in automatically generating -by making use of perceptual concepts, and a supervised learning process driven by a genetic algorithm- a gain matrix that aims at enhancing speech perceived by hearing aid users. The results obtained in this chapter clearly depict that the computed, perceptual-based **gain matrix makes the hearing aid reach better results** in terms of speech quality and speech intelligibility perceived by the user (regardless the user's hearing loss) **than those measured when the user makes use of a hearing aid in which the gain is based on a 3-piecewise linear approximation for each frequency band**.

- An **approach that aims at "simplifying" the implementation of the compression-expansion algorithm** (the very core of a hearing aid) **in the DSP**. This approach has been proven to be **very advantageous** in the sense that it **demands lower computational cost** for its implementation than that demanded to program **the typical compression-expansion algorithm, which is based on a piecewise linear approximation for each frequency band**. The practical implementation of our approach consists in storing in the data-memory of the DSP a table containing "tabulated" values of the gain to be applied as a function of *both* the input signal level (dB SPL) and the frequency band.

- The last contribution, but not the *least* important, is that, the **practical implementation of an automatic sound classification in a digital hearing aid** has been found to be **feasible**. The total computational load demanded for implementing such automatic sound classification system has been found to be approximately 70 %, and thus, there is still 30 % of the DSP resources available for implementing other algorithms. Regarding the **battery life**, it has been found to be 140 hours, or equivalently, **approximately 6 days**.

Additionally, it has been shown in this thesis that:

- The fact of **quantizing the sound signal-describing features** that feed the classifying algorithm and the **parameters of the classifier does not cause a degradation in the performance of the classification system**. Or in other words, the "quantized" complete classification system (or equivalently, the classification system to be programmed in the DSP) achieves perceptually the same efficiency as that reached by the "exact" classification system (that is, without quantization).

- The fact of using relatively **large time slots for the classifier returns a decision** such as, for instance, files of 2.5 seconds, **exhibits very good results when compared with those obtained for shorter time slots**, like, for example, files of 20 milliseconds.

9.4 Future work

Following the investigations described in this thesis, the main lines of research that remain open, roughly speaking, are listed below:

- Throughout this thesis, we *have not* taken into account the "acoustic feedback problem" in the hearing aid performance. Acoustic feedback appears when part of the conveniently amplified output signal produced by the device returns through the auditory canal, and enters again this device, being thus anew amplified. Sometimes, feedback may cause the hearing aid to become unstable, producing irritating howls. Therefore, an interesting line of research would be to explore whether it is worthwhile or not to include the effects of this distortion (caused by the feedback problem) in the training process of a multilayer perceptron-based classifier.

- Deepening a little more in this line of research, the questions arising here could be as follows: *what is the effect of feedback reduction algorithms on the classification process?* Additionally, these algorithms may increase the overall gain significantly before feedback effect is occurred. *Would the classifier implemented in the device have to select different feature sets based on the presence or absence of feedback?* Obviously, addressing these complex questions in detail would require a new line of study.

- This thesis *has not* focused on binaural speech localization in hearing aids. Binaural source localization is the problem of determining the location of a sound source of interest by using both signals entering the two ears. In particular, it would be interesting to focus on the binaural localization of the *speech* signal (the preferred one for most of the users that wear two hearing aids) and by means of "spatial filtering", suppressing the background noise and enhancing the *speech* source. The main drawback here is that, the automatic binaural source localization of the interesting *speech* source is a difficult problem, not only because of the complexity of the problem in *itself* (background noise and the wealth of sound sources mixing simultaneously), but also because of some *critical* design restrictions that limit the design of advanced algorithms in DSP-based hearing aids.

- Since the main problem of implementing complex algorithms in DSP-based hearing aids is that they have very important constraints in terms of computational capability and memory, it would be very convenient to design low-complexity acoustic feedback reduction and speech source localization algorithms, in order to be further implemented in the DSP.

9.5 List of publications

The following paragraphs present a list of published work produced during the course of candidature for the degree. None of these publications have previously formed a part of another thesis. These publications are listed below:

- **International journals**

  - L. Álvarez, L. Cuadra, E. Alexandre, R. Gil-Pita and M. Rosa-Zurera, "Speech enhancement in hearing aids via gaussian mixture perceptual-based gain driven by genetic algorithms", *Artificial Intelligence in Medicine*, Elsevier. Accepted with minor changes.
  - R. Gil-Pita, L. Cuadra, E. Alexandre, D. Ayllón. L. Alvarez and M. Rosa-Zurera, "Enhancing the energy efficiency of wireless-communicated binaural hearing aids for speech separation driven by soft-computing algorithms", *Applied Soft Computing*, 2011,
    doi: 10.1016/j.asoc.2011.03.022.
  - L. Cuadra, E. Alexandre, R. Gil-Pita, R. Vicen-Bueno and L. Álvarez, "Influence of acoustic feedback on the learning strategies of neural network-based sound classifiers in digital hearing aids", *EURASIP Journal on Advances in Signal Processing*, vol. 2009, Article ID 465189, 10 pages, 2009. doi:10.1155/2009/465189.

- E. Alexandre, L. Cuadra, L. Álvarez, M. Rosa-Zurera and F. López-Ferreras, “Two-layer automatic sound classification system for conversation enhancement in hearing aids”, *Integrated Computer Aided Engineering*, vol. 15, no. 1, 2008.
- E. Alexandre, L. Cuadra, L. Álvarez, M. Rosa-Zurera and F. López-Ferreras, “Automatic sound classification for improving speech intelligibility in hearing aids using a layered structure”. Lecture Notes in Computer Science, 2006.

• **International book chapters**

- E. Alexandre-Cortizo, L. Álvarez-Pérez, L. Cuadra-Rodríguez, M. Rosa-Zurera and F. López-Ferreras, "Automatic sound classification algorithm for hearing aids using a hierarchical taxonomy", in *New Trends in Audio and Video*, A. Dobrucki, A. Petrovsky and W. Skarbek, Eds., vol. 1, Politechnika Bialostocka, 2006.

• **International conferences**

- L. Álvarez, C. Llerena, E. Alexandre, R. Gil-Pita and M. Rosa-Zurera, “Selection of approximated activation function in neural network-based sound classifiers for digital hearing aids”, in *AES 130th Convention*, London (United Kingdom), 2011.
- R. Ahmed, R. Gil-Pita, D. Ayllón and L. Álvarez, “Speech source separation using a multi-pitch harmonic product spectrum based algorithm”, in *AES 130th Convention*, London (United Kingdom), 2011.
- L. Álvarez, L. Vaquero, E. Alexandre, L. Cuadra and R. Gil-Pita, “Evaluation of speech intelligibility in digital hearing aids”, in *AES 128th Convention*, London (United Kingdom), 2010.
- L. Álvarez, E. Alexandre and M. Rosa-Zurera, “Pruning algorithms for multilayer perceptrons tailored for speech/non-speech classification in digital hearing aids”, in *AES 126th Convention*, Munich (Germany), 2009.
- E. Alexandre, L. Álvarez-Pérez, R. Gil-Pita, R. Vicen-Bueno and L. Cuadra, “On the design of automatic sound classification systems for digital hearing aids”, in *AES 126th Convention*, Munich (Germany), 2009.
- L. Álvarez, E. Alexandre, R. Vicen, L. Cuadra and M. Rosa, “A constructive algorithm for multilayer perceptrons for speech/non-speech classification in hearing aids”, in *124th AES Convention*, Amsterdam (Netherlands), 2008.
- E. Huerta, E. Alexandre, R. Gil-Pita, L. Álvarez and J. Amor, “Analysis of the effects of finite precision in sound classifiers for digital hearing aids”, in *124th AES Convention*, Amsterdam (Netherlands), 2008.
- J. Amor, E. Alexandre, R. Gil-Pita, L. Álvarez and E. Huerta, “Music-inspired harmonic search algorithm applied to feature selection for sound classification in hearing aids”, in *124th AES Convention*, Amsterdam (Netherlands), 2008
- L. Cuadra, E. Alexandre, L. Álvarez and M. Rosa-Zurera, “Reducing the computational cost for sound classification in hearing aids by selecting features via genetic algorithms with restricted search”, in *IEEE International Conference on Audio, Language and Image Processing (ICALIP)*, Shanghai (China), 2008.

- E. Alexandre, R. Gil-Pita, L. Cuadra, L. Álvarez and M. Rosa-Zurera, “Speech-/music/noise classification in hearing aids using a two-layer classification system with MSE linear discriminants”, in *16th European Signal Processing Conference (EUSIPCO)*, Lausanne (Switzerland), 2008.
- L. Álvarez, E. Alexandre, L. Cuadra and M. Rosa-Zurera, “On the training of multilayer perceptrons for speech/non-speech classification in hearing aids”, in *122nd AES Convention*, Viena (Austria), 2007.
- E. Alexandre, L. Cuadra, L. Álvarez and M. Utrilla, “Exploring the feasibility of a two-layer NN-based sound classifier for hearing aids”, in *15th European Signal Processing Conference (EUSIPCO)*, Poznań (Poland), 2007.
- E. Alexandre, L. Cuadra, L. Álvarez and M. Rosa-Zurera, “NN-based automatic sound classifier for digital hearing aids”, in *IEEE International Symposium on Intelligent Signal Processing (WISP)*, Alcalá de Henares (Spain), 2007.
- R. Vicen-Bueno, R. Gil-Pita, M. Utrilla-Manso and L. Álvarez-Pérez, “A hearing aid simulator to test adaptative signal processing algorithms”, in *IEEE International Symposium on Intelligent Signal Processing (WISP)*, Alcalá de Henares (Spain), 2007.

Appendix A

Digital signal processor

The digital signal processor (DSP) used in this thesis to carry out the experiments is Toccata Plus [Dspfactory, 2002a,b], currently manufactured by ON Semiconductor. This small DSP is one of the most widely-used in hearing aid applications.

This DSP is basically composed of two coprocessors operating concurrently, as schematically illustrated in Figure A.1.

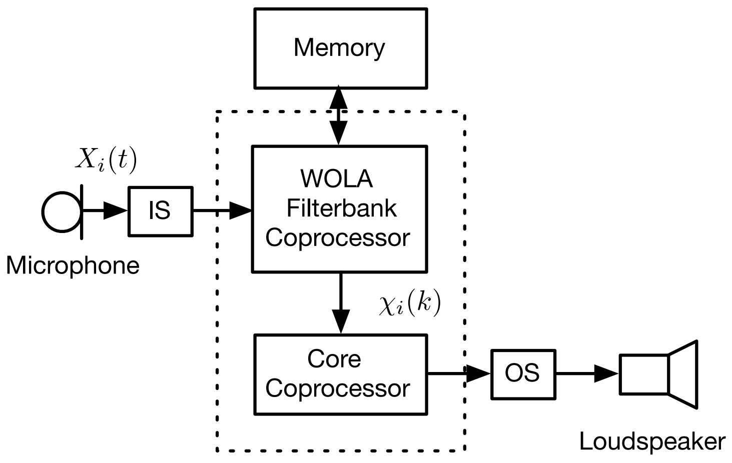


Figure A.1: Simplified architecture diagram of the DSP used by our platform to carry out the experiments. For the sake of clarity, it only shows the two coprocessors that operate concurrently. The WOLA filter bank coprocessor computes $|\chi_i(k)|^2$, $\chi_i(k)$ being the k-th frequency bin of the spectrum of a given sound frame $X_i(t)$, while the core coprocessor deals with the remaining signal processing tasks. In this picture, IS and OS represent, respectively, the input and output stages.

The first one is the Weighted Overlapp-Add (WOLA) filter bank coprocessor, which performs the time/frequency decomposition of a given sound frame $X_i(t)$ with N_{B} frequency bands. Or in other words, it provides the matrix $\overline{\chi_i} = [\chi_i(1), \chi_i(2), \ldots, \chi_i(N_{\mathrm{B}})]$, N_{B} being the number of frequency bands available in the DSP ($N_{\mathrm{B}} = 64$, in our case). Please note that $\chi_i(k)$ simply labels the k-th frequency bin of the spectrum of a sound frame $X_i(t)$, where $k = 1, \ldots, N_{\mathrm{B}}$ is the index over the N_{B} frequency bands. The second coprocessor is the core coprocessor dealing with the remaining tasks, such as, for instance, compensating the hearing loss, classifying the acoustic environment [Alexandre et al., 2006a, 2008b], noise reduction schemes, and so on. In that picture, IS and OS represent, respectively, the input and output stages.

The 16-bit, programmable, fixed-point Toccata Plus DSP - which additionally integrates not only the two aforementioned coprocessors, but also two 16-bit ADCs, two 16-bit DACs, having both a dynamic range of 96.3 dB, the RAM, ROM and EPROM memories, and some input/output ports - has to work at a very low clock frequency (1.92 MHz, in our case) to minimize power consumption ($\sim 600\ \mu A$ at 1.4 V) and thus maximize battery life. Note that the power consumption must be low enough to ensure that neither the size of the battery pack nor the frequency of the battery changes will annoy the user.

Roughly speaking, the computational power available in this DSP does not exceed 3 MIPS, with only 32 Kbytes of internal memory. It must be considered that only the WOLA filter bank requires about 33 % of the computation time of the DSP. With this in mind, around 2 MIPS being thus free for implementing other algorithms.

Appendix B

Hearing aid simulator

B.1 Introduction

In the effort of evaluating the performance of the different hearing-aid processing algorithms proposed in this thesis, prior to their circuit design and their development in the DSP used by our platform to carry out the experiments, we have developed a "digital hearing aid simulator" [Vicen-Bueno et al., 2007]. Although will be explained in detail later on, we can say in advance that this hearing aid simulator basically works as a filter that takes a sound file and generates hearing-aid affected sounds like those the person wearing the hearing aid finally hears in his/her daily life.

Roughly speaking, the design of this digital hearing aid simulator involves several important aspects, such as, for instance, the selection of the signal processing techniques to be applied, the selection of the models that simulate the acoustic feedback or the noise reduction techniques. Aiming at better understanding the way it works, we describe in a detailed way throughout this appendix the functional blocks that compose the hearing aid model that we have designed in this thesis.

Completing this brief section demands to mention that this hearing aid simulator, which offers the benefit of a great flexibility in controlling some important parameters like, for example, hearing thresholds, compression range or attack/release times, has been proven to be a valuable simulation tool in creating realistic sounds as the ones that enter into the real patient's ear after suffering effects from the hearing aid [Gil-Pita et al., 2009].

B.2 Digital hearing aid model

The digital hearing aid model used for the design of the hearing aid simulator implemented in this thesis is schematically depicted in Figure B.1. Although not clear at first glance, it is worth mentioning that it consists of three main functional blocks, which are listed below:

- The acoustic feedback equivalent (AFE) path.
- The hearing aid core, whose main component is the Weighted Overlapp-Add (WOLA) filter bank.
- The gain evaluation stage, which includes both audio level compression and noise reduction.

These three functional blocks will be explained in depth throughout this section because of its relevance in the hearing aid design. Prior to the description of these functional blocks, it is worth

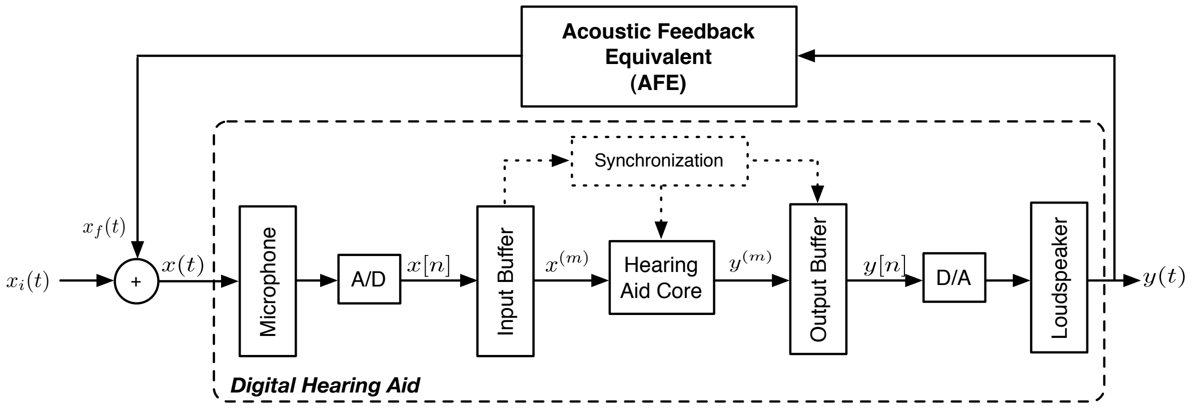


Figure B.1: Schematic diagram of the digital hearing aid model used for the purpose of designing the "digital hearing aid simulator". Any input sound $x_i(t)$ is filtered by the system that generates a hearing-aid affected sound $y(t)$ similar to that the person wearing the hearing aid finally hears in his/her daily life.

having a look again at Figure B.1 because one can notice that, apart from the aforementioned blocks, this hearing aid also consists of:

- A microphone to convert the input sound into an electric signal.
- A 16-bit analog-to-digital converter (ADC) that transforms the analog electrical signal into a digital signal.
- A 16-bit digital-to-analog converter (DAC) that backs the amplified digital signal into an analog signal.
- A small loudspeaker to convert the electrical signal back to sound.

Both ADC and DAC have a quantification noise level of 96.3 dB SPL.

Concerning the buffers shown in the figure, the "input buffer" accumulates R samples (a block of samples, $\mathbf{x}^{(m)}$) to pass to the "hearing aid core" or equivalently, to the WOLA filter bank, whereas the "output buffer" takes the R hearing aid core output samples ($\mathbf{y}^{(m)}$) to pass the samples sequentially to the DAC at the sampling rate f_s (f_s = 16 kHz, in our case). This block of samples at the WOLA output is delayed one block because the hearing aid core *does not* give the output block since another input block is available at the hearing aid input. This means that the time for a sound at the hearing aid input to be presented again at its input due to the feedback effect (labelled t_AFE) is mathematically defined as follows:

$$t_\mathrm{AFE} = \frac{R}{f_\mathrm{s}} + T \tag{B.1}$$

where:

- R is the number of samples,
- f_s labels the sampling frequency, and finally,

- T represents the propagation time of the sound from the loudspeaker to the microphone (measured at the sound velocity in the air), which is slightly higher than the time between two samples.

The core of this hearing aids is the WOLA filter bank because it establishes the way of working. This filter bank is explained in depth in the section that immediately follows.

B.2.1 Hearing aid core: the WOLA filter bank

As previously mentioned, the hearing aid core is the WOLA filter bank (sometimes named "DFT-filter bank"), which is a complex-modulation filter bank that has been successfully used in many hearing aid many applications [Alexandre et al., 2008a; Schneider et al., 1999; Vicen-Bueno et al., 2009].

As schematically represented in Figure B.2, this filter bank is divided into three blocks: 1) an analysis stage, 2) a gain stage (where the gain to apply is computed considering, among other parameters, the patient's acoustic loss and the audio level compression and noise reduction parameters), and 3) the synthesis stage. These three stages will be explained in a detailed way in the subsections that follow.

It is important to note that the WOLA filter bank has been implemented using a configurable arithmetic types like, for example, fixed-point, floating-point and block-floating-point with a variable number of bits for both the data and the operations. This configurable arithmetic is applied in the different parts of the hearing aid simulator with its own properties (fixed or floating-point and number of bits) according to necessities of each part. This fact allows us to test different hardware architectures for design purposes.

B.2.1.1 The analysis stage

In this first stage, the sequence of input R samples ($\mathbf{x}^{(m)}$)) is multiplied by a L samples length Hamming window in order to provide better frequency resolution. Bearing in mind that $L = n \cdot R$ (with $n = 1, 2, \ldots, N$), the reader may wonder how it is possible to multiply the input block, which contains R samples, by a L samples length Hamming window in case $L \neq R$. Answering this question demands to mention that not only is the R samples block considered as the input samples to the WOLA filter bank but also the $L - R$ previous samples, or in other words, $(L - R)/R$ previous blocks are considered. This is clearly represented in the "WOLA filter bank analysis stage" shown in Figure B.2.

Regarding the WOLA filterbank [Schneider et al., 1999], it is based on DFTs of N frequency samples of the "windowed input", where *even* stacking is selected. This is basically the reason why in Figure B.2, the DFT and its inverse transform, the IDFT are done with $N/2$ points, which leads to reduce the computational cost in the implementation of both transforms. This DFT is usually implemented using algorithms of fast computation, such as, for instance, the Fast Fourier Transform (FFT).

Finally, it is important to note that an oversampling factor $OS = N/R$ is considered, which is necessary to adjust the filter bank bands without aliasing and improving resolution.

B.2.1.2 The gain stage

Although will be explained in a more detailed way in Section B.2.3, we can say that the main purpose of this second stage is to apply the gain to each of the $N/2$ frequency bands in order to compensate the particular patient's acoustic losses. As stated beforehand, this value of gain is

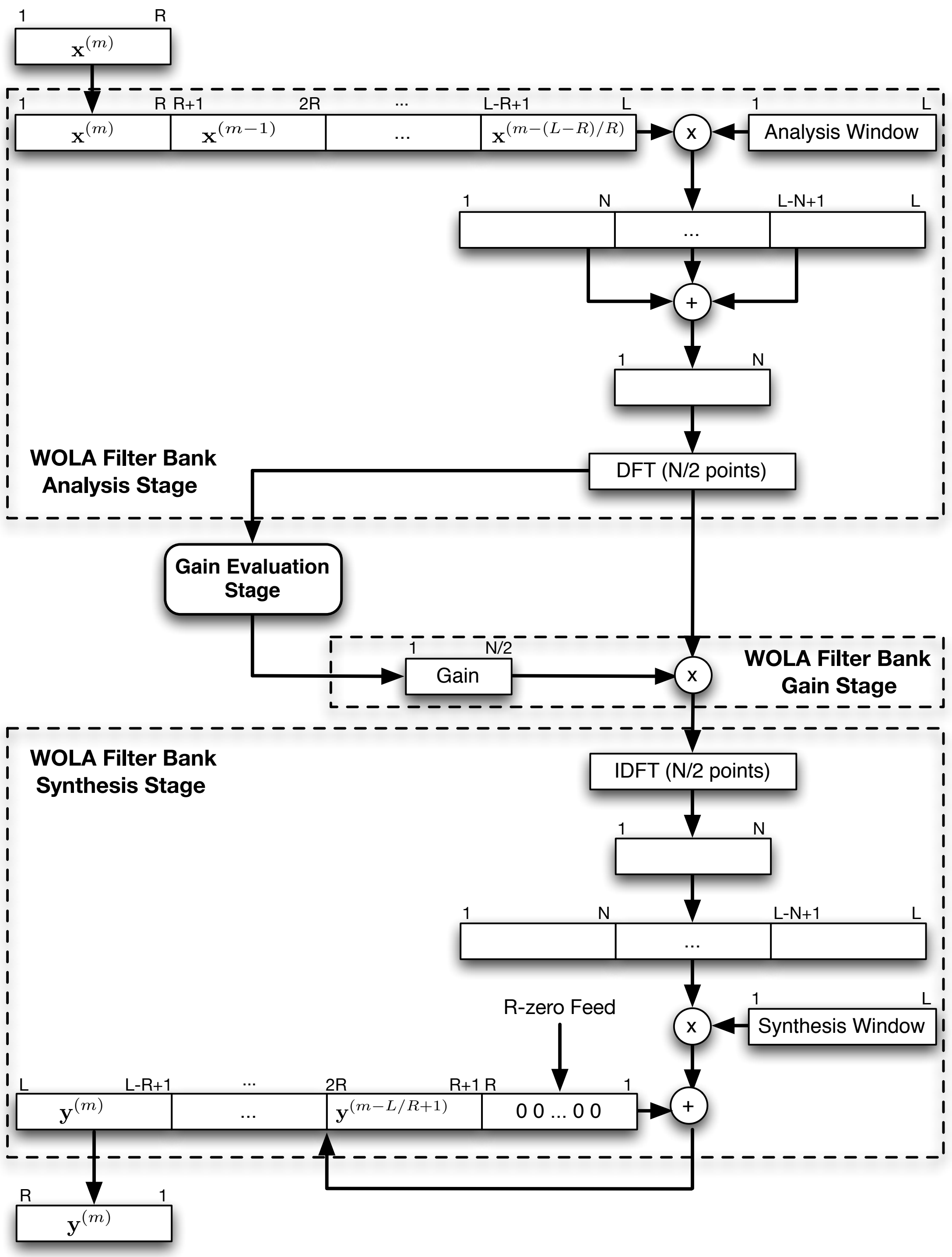


Figure B.2: Conceptual description of the hearing aid core, which is basically composed of the WOLA filter bank. This filter bank consists of three blocks: 1) an analysis stage, 2) a gain stage (where the gain to apply is computed considering, among other parameters, the patient's acoustic loss and the audio level compression and noise reduction parameters), and 3) the synthesis stage.

computed taking into account, among others parameters, the patient's acoustic losses, the audio level compression and noise reduction parameters.

B.2.1.3 The synthesis stage

In this third stage, a decimation factor (DF) determines how much the analysis window is decimated to generate the synthesis window. In our model, we have considered $DF = 1$, which involves both windows have the same length. It is important to note that this stage is the inverse stage of the analysis one, and as depicted in the "WOLA filter bank synthesis stage" represented in Figure B.2, its corresponding inverse operations are carried out.

According to the WOLA filter bank configuration, the filter bank group delay (GD) can be obtained by means of the following expression:

$$GD = \left(\frac{L}{2} + \frac{L}{2 \cdot DF + R}\right) \cdot \frac{1}{f_s}. \tag{B.2}$$

B.2.1.4 Practical implementation: WOLA filter bank parameters

In order to carry out the batches of experiments by using the hearing aid simulator, the parameters that we list below have been selected for the WOLA filter bank:

- Sampling frequency: $f_{\mathrm{s}} = 16$ kHz.
- Input block size: $R = 64$ samples.
- Analysis/synthesis window length: $L = 256$ samples.
- FFT size: $N/2 = 64$ points.
- Oversampling factor: $OS = N/R = 2$.
- Decimation factor: $DF = 1$.

With this sampling frequency in mind, it is important to note that $N_{\mathrm{B}} = 64$ frequency bands (spaced 125 Hz) in the range of $0 - 8$ kHz are obtained.

Concerning the architecture, the simulator have been configured for a 16-bit word length Architecture with a Multiplier/ACcumulator that multiplies 16-bit registers and stores the result in a 40-bit accumulator.

B.2.2 AFE paths in hearing aids

As stated beforehand, this block represents the second key functional block in the hearing aid simulator design. In particular, we have considered here a new single path, the AFE path, whose effect is equivalent to the combination of all the feedback factors in a hearing aid.

A variety of empirical studies have found that the acoustic feedback path frequency responses of the different categories of hearing aids have common properties [Hellgren et al., 1999]. In this respect, the frequency response of the AFE path magnitude is approximately "flat" from low frequency to 2 kHz and presents a peak with low quality factor around 4 kHz and a dip around 7 kHz. The phase response is practically linear all over the frequency range, except at high frequencies where the response suffers from a slight phase change.

With this in mind, the AFE path of a variety of hearing aid styles can be modeled by its transfer function in the Laplace domain using the following expression [Hellgren et al., 1999]:

$$H_{AFE}(s) = Ke^{-sT_\mathrm{p}} \cdot \frac{(s-z_1)(s-z_2)(s-z_2^*)}{(s-p_1)(s-p_1^*)(s-p_2)(s-p_2^*)} \tag{B.3}$$

where:

- K is the gain factor used to control the magnitude,
- T_p is the propagation time from the loudspeaker to the microphone,
- z_1 and z_2 are the positions of the zeros of the transfer function in the Laplace domain, and finally,
- p_1 and p_2 are the position of its poles.

The zero (z_2^*) and the poles (p_1^*, p_2^*) denote the complex conjugate of the zero (z_2) and the poles (p_1, p_2), respectively. The single zero (z_1) and a complex conjugate poles pair (p_1, p_1^*) are used to control the slope up to 4 kHz. The second complex conjugate poles pair (p_2, p_2^*) is used to obtain a peak at middle frequency. Thus, they should be placed closer to the imaginary axis in order to provide a higher quality factor to this peak. Finally, the complex conjugate zeros pair (z_2, z_2^*) is used to give the dip at high frequencies.

The question arising here is: *which are the optimal zeros and poles positions?* The answer to this question is as follows. The process of optimizing the zeros and poles positions of the AFE transfer function is divided into several steps, which are described in detail in [Hellgren et al., 1999]. We have briefly described these steps here in the effort of making this section stand itself. First, the AFE path frequency response is normalized in magnitude, and after that, an optimization algorithm based on an equation error method searches the best model of the frequency response. Then, the used equation error method is divided into two steps. In the first step, a variant of the algorithm proposed in [Levi, 1959] is applied. Whereas, the second step is based on a damped Gauss-Newton method for iterative search [Jr. and Schnabel, 1983]. Note that the first algorithm is applied in the first iteration of the algorithm and the second algorithm is applied during the remaining iterations until a maximum of iterations is reached or the minimum of error function is found.

B.2.2.1 Practical implementation: AFE paths

For illustrative purposes, the digital hearing aids models selected in this study to carry out the batches of experiments are the ITC, the ITE, the BTEA2 and BTEB models [Hellgren et al., 1999]. According to Expression B.3 and the frequency responses of the hearing aids for the considered models, the hearing aid AFEs can be computed by means of the suitable establishment of the poles (p_1, p_2 and their conjugates) and zeros (z_1, z_2 and their conjugates), the gain factor (K) and the propagation time from the loudspeaker to the microphone (T_p). For these kinds of models, it is worth mentioning these two comments:

- The gain factors are computed according to the losses of the returning path.
- The propagation time of the sound, from the loudspeaker to the microphone, at the sound velocity in the air is equivalent to the time to cover the average distance of 17.6 mm at 340 m/s, which results in 51.76 μs.

Model	**Zeros**		**Poles**	
	z_1	z_2	p_1	p_2
ITC	- 800000	-60000+26400j	-5000+22600j	-500+37700j
ITE	-1500000	-40000+31400j	-4500+25000j	-900+44000j
BTEA2	-150000	-65000+39000j	-7500+25000j	-1500+44000j
BTEB	-300000	-40000+29000j	4500+25000j	-900+44000j

Table B.1: Appropriate zero-pole configuration of the AFEs filters of a variety of hearing aid models that include the ITC, the ITE, the BTEA2 and BTEB models.

With this in mind, the appropriate value of the zeros and the poles of the considered hearing aid models are shown in Table B.1.

Finally, for properly completing this section, Figure B.3 illustrates the magnitude response of the AFE filters for the four considered hearing aids. As depicted in this figure, the AFE magnitude of BTE model is slower, which involves that the maximum gain without feedback is higher.

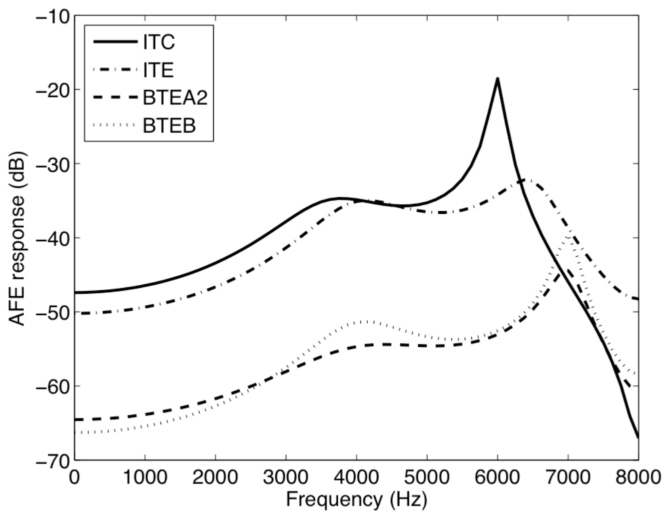


Figure B.3: Representation of the AFE path-magnitude frequency response for the hearing aid models considered: the ITC, the ITE, the BTEA2 and the BTEB.

B.2.3 Gain evaluation stage: audio level compression and noise reduction

The gain evaluation stage is based on a multi-level and multi-frequency compressor-expander algorithm, in which the gain value to apply by the hearing aid, also named "insertion gain", basically depends on the average input sound level (dB) in each frequency band.

Perhaps the reader may wonder how this gain value is exactly computed. Answering this question demands to mention that this gain value is based on some gain curves, which are *properly* designed in an effort of fulfilling the requirements specified by the FIG6 prescription method [Dillon, 2001]. This method basically specifies the amount of gain required to normalize the loudness for different levels of the input sound signal. In other words, for any study-case patient, whose hearing losses are H_k (dB HL) at the k-th frequency band, the aforementioned method sets that the required insertion gain at this frequency (labelled IG_k) for the following three input audio levels: 40 dB SPL (labelled $IG40_k$), 65 dB SPL (labelled $IG65_k$) and 95 dB SPL (labelled $IG95_k$) as shown below:

$$IG40_k = \begin{cases} 0 & \text{if } H_k \leq 20 \\ H_k - 20 & \text{if } 20 < H_k \leq 60 \\ 0.5H_k + 10 & \text{if } 60 < H_k \end{cases} \tag{B.4}$$

$$IG65_k = \begin{cases} 0 & \text{if } H_k \leq 20 \\ 0.6(H_k - 20) & \text{if } 20 < H_k \leq 60 \\ 0.8H_k - 23 & \text{if } 60 < H_k \end{cases} \tag{B.5}$$

$$IG95_k = \begin{cases} 0 & \text{if } H_k \leq 40 \\ 0.1(H_k - 40)^{1.4} & \text{if } 40 < H_k \end{cases}. \tag{B.6}$$

For comparative purposes, Figure B.4 shows the insertion gains for the three aforementioned input signal levels as a function of the hearing losses. Note that these gain curves are obtained for a *particular* k-th frequency band. As clearly shown, the insertion gain to apply by the hearing aid varies depending on the input signal level in the sense that low sounds are less amplified than quiet sounds. This makes definitely the use of audio level compression techniques very appropriate for hearing aids applications because "hearing aid output saturation" is controlled, preventing thus possible hearing damage.

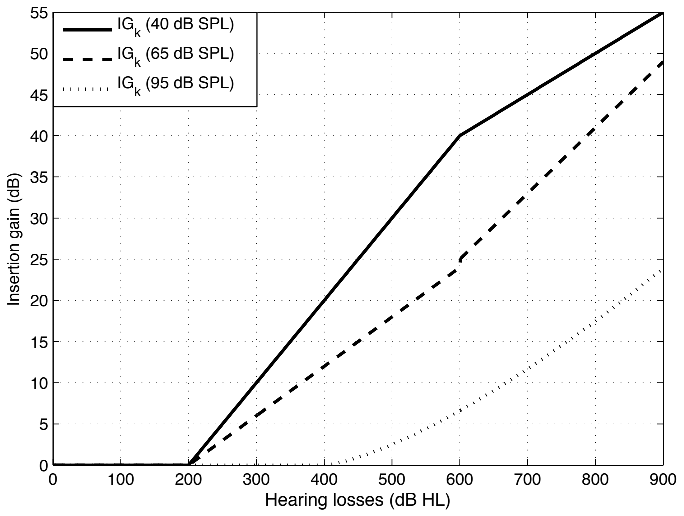


Figure B.4: Illustrative representation of the insertion gain curves for the following three input audio levels: 40 dB SPL (labelled $IG40_k$), 65 dB SPL (labelled $IG65_k$) and 95 dB SPL (labelled $IG95_k$) as a function of the hearing losses at the k-th frequency band. These gain curves are obtained according to FIG6 prescription method [Dillon, 2001].

Once $IG40_k$, $IG65_k$ and $IG95_k$ values are obtained (please note that these values are obtained at the particular k-th frequency band), it is necessary to establish a set of strategies and techniques for efficiently designing the gain curves that achieve the desired insertion gains. In this respect, an input-to-gain mapping model, based on a standard 3-piece linear approximation as the one represented in Figure B.5, is used here to design the gain curve at each k-th frequency band.

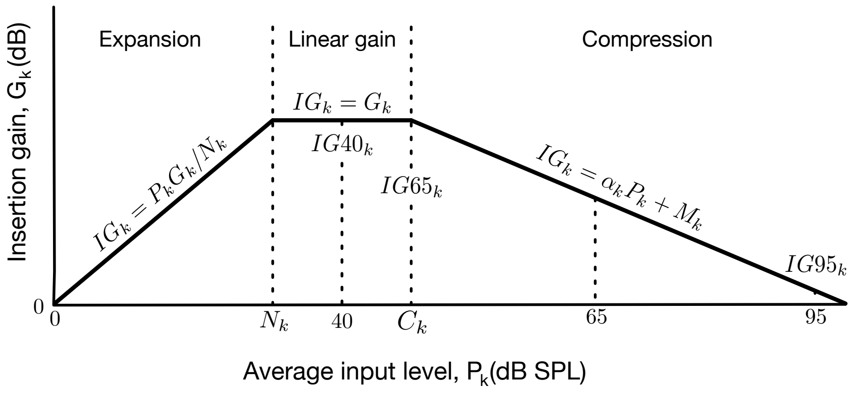


Figure B.5: Representation of the input-to-gain mapping model, based on a standard 3-piece linear approximation, used here to design the gain curves at each k-th frequency band.

Having a look at this figure, one can notice that there are three different input regions clearly distinguished:

- The *expansion region* that deals with low average input sound levels, where the insertion gain to apply starts with 0 dB for an input level of 0 dB SPL, and increases linearly until the input level reaches the configurable level N_k (dB SPL). In this case, N_k is selected in the sense that the gain applied to the majority of quiet background noises, such as, for instance, fan noise, is not too high aiming at avoiding the loss of quality in the interesting speech source or audio signal.

- The *linear gain region* that deals with medium average input sound levels, where the insertion gain to apply, regardless the input level within this region, is always a constant value. This insertion gain is applied to average input sound levels between N_k (named "noise reduction threshold") and C_k (named "compression threshold").

- The *compression region* deals with high average input levels, where the insertion gain applied decreases as the input sound level increases. This is the key "working region" of the curve because it allows to yield different gain values for input levels between 65 dB SPL and 95 dB SPL, although it is not strictly restricted to this range.

With this compression level and noise reduction schemes in mind, the insertion gain applied at k-th frequency band, that is, IG_k, is calculated as a function of the corresponding average input level, labelled P_k (dB SPL), as shown below:

$$IG_k = \begin{cases} P_k G_k / N_k & \text{if } P_k < N_k \\ G_k & \text{if } C_k \geq P_k > N_k \\ \alpha_k P_k + M_i & \text{if } P_k > C_k \end{cases} \tag{B.7}$$

where:

- G_k labels the maximum gain applied by the hearing aid, and

- α_k and M_k are the parameters that define the *compression region.*

The question arising here is: *how are α_k and M_k parameters determined?* These parameters are computed by using the insertion gains given by $IG65_k$ and $IG95_k$, as shown in the expressions that immediately follow:

$$\alpha_k = \frac{IG95_k - IG65_k}{30} \tag{B.8}$$

$$M_k = \frac{19 \cdot IG65_k - 13 \cdot IG95_k}{6}. \tag{B.9}$$

With respect to the remaining parameters, that is, G_k, C_k and N_k, it is worth mentioning three last comments:

1. The maximum gain value (G_k) has been found to be $G_k = IG40_k$, or in other words, it has been found to be the gain value in the *linear gain region.*

2. Bearing in mind the continuity of the curves, the expression of the compression threshold C_k is given by:
$$C_k = \frac{IG40_k - M_k}{\alpha_k}. \tag{B.10}$$

3. The upper average input level of the *expansion region*, or equivalently, the lowest average input level of the *linear gain region* is adjusted according to the input noise level that the hearing aid has to reduce or minimize. Please note that this level must be always lower than C_k that is controlled by the insertion gain constraints ($IG40_k$, $IG65_k$ and $IG95_k$), which basically depend on the particular hearing losses the patient suffers from.

In the effort of avoiding "fast changes" in the gain applied by the hearing aid to the input signal, the average input level (P_k) is evaluated by time averaging of the instantaneous input signal level (X_k). To achieve this, an IIR-based average estimation of the input level is used taking into account the instantaneous input levels of different frames. Being thus $P_k^{(m-1)}$ the estimated input signal level obtained in the $(m-1)$-th frame, the actual input level $P_k(m)$ is obtained by means of the expression shown below:

$$P_k^{(m)} = \begin{cases} (1-\beta_a)P_k^{(m-1)} + \beta_a X_k & \text{if } X_k \geqslant P_k^{(m-1)} \\ (1-\beta_r)P_k^{(m-1)} + \beta_r X_k & \text{if } X_k < P_k^{(m-1)} \end{cases} \tag{B.11}$$

where:

- X_k is the input level signal at k-th frequency,
- $P_k(m)$ represents the k-th frequency bin of the average power at frame m-th,
- $P_k^{(m-1)}$ is the k-th frequency bin of the instantaneous power at frame $m-1$-th,
- β_a labels the attack compressor time, and finally,
- β_r depicts the release compressor time.

Perhaps the reader may wonder what β_a and β_r exactly mean. In this respect, it is worth noting that a compressor amplifier is characterized by its temporal dynamic characteristics. Roughly speaking, most of compressor amplifiers require a short period of time to react to an increase in input signal level. This time is known as "attack time" and we label it β_a[1]. A similar scenario happens when the input signal decreases in level. The "release time" is the time need for the compressor to react to a decrease in the input level[2]. We label it β_r. Note please that the stability of this IIR-based average estimation is fulfilled when the attack and release parameters are always greater than 0 and lower than 2, that is, $0 < \beta_a < 2$ and $0 < \beta_r < 2$.

It is important to highlight that the insertion gain is applied in the hearing aid core and thus we must take into account the responses of both the microphone and the loudspeaker. In this respect, the model of microphone implemented in this simulation tool is characterized at each frequency, f_k, where $k = 1, \ldots, N/2$ is the index over the $N/2$ frequency components, by its both frequency responses for linear and saturation conditions, labelled M_k^{lin} and M_k^{sat}, respectively. Saturation conditions here involve that the sound pressure level, at the input of the hearing aid, is the maximum input level considered in our studies. Therefore, the input signal to the hearing aid core, labeled X_k^{core}, being X_k the input audio level, is given by:

$$X_k^{core} = \begin{cases} X_k + M_k^{lin} & \text{if } X_k < M_k^{sat} \\ M_k^{sat} + M_k^{lin} & \text{if } X_k \geqslant M_k^{sat} \end{cases} . \tag{B.12}$$

In the same line of reasoning, the loudspeaker is also characterized by its frequency responses for linear and saturation conditions at each k-th frequency band, labelled L_k^{lin} L_k^{sat}, respectively. In this case, the linear condition establishes the relationship between the level of the signal at the output of the hearing aid core, labelked Y_k^{core}, and the sound pressure level at the output of the hearing aid (Y_k) at the k-th frequency band. On the other hand, the saturation condition sets its maximum output audio level that can be achieved and its corresponding hearing aid core output level, which can be determined by means of the following expression:

$$Y_k = \begin{cases} Y_k^{core} + L_k^{lin} & \text{if } Y_k^{core} + L_k^{lin} < L_k^{sat} \\ L_k^{sat} & \text{if } Y_k^{core} + L_k^{lin} \geqslant L_k^{sat} \end{cases} . \tag{B.13}$$

Bearing in mind that 1) the usual work region of both, the loudspeaker and microphone, is the *linear region*, and 2) the insertion gain relates the input audio level and the output level in the hearing aid in the sense that $IG_k = Y_k - X_k$, it is possible to relate the insertion gain that must be achieved by the hearing aid, that is, IG_k (depicted in Expression B.14), to the insertion gain that must introduce the hearing aid core, labelled $IG_k^{(core)}$, by means of the expression shown below:

$$IG_k = Y_i^{core} + L_k^{lin} - X_k^{core} + M_k^{lin} = IG_k^{core} + L_k^{lin} + M_k^{Lin}. \tag{B.14}$$

In this case, IG_k^{core} considers the effects of the microphone and loudspeaker, as shown below:

$$IG_k^{core} = IG_k - L_k^{lin} - M_k^{lin} \tag{B.15}$$

[1]Specifically, it is defined as the time needed for the output of the compressor to come within 3 dB of its steady state level after abrupt increase in input signal level.

[2]Specifically, it is defined as the time needed for the output of the compressor to come within 3 dB of its new steady state level after an abrupt decrease in input signal level.

Appendix C

Databases

This appendix aims at describing the two databases used in this thesis to carry out the batches of experiments. These databases include the database of sounds and the database of *real* patients, which have been clearly described in Sections C.1 and C.2, respectively.

C.1 Database of sounds

An appropriate database of sounds is needed to evaluate a classification system. Since there are diverse types of listening environments the real patients inevitably face in their everyday life, the database has to be carefully designed. In this respect, we have created a database of sounds, labelled $\mathcal{D}$, which contains a total of 7299 audio files, including speech-in-quiet, speech-in-noise, speech-in-music, vocal music, instrumental music and noise, each of 2.5 seconds length. It is worth mentioning that, within the context of the application at hand, the class silence has not been considered because it has been found to be easy to distinguish and consequently, it would not make sense to include audio files consisting basically of quiet only.

The database has been manually labelled, leading to a total of 807 files of speech-in-quiet, 1615 of speech-in-noise, 2440 of music (including both vocal and instrumental music) and 2437 of noise. Each sound file represents exactly one of the aforementioned sound classes, or in other words, class changes *do not* occur within the file. Please note that, speech-in-music files have been also categorized as speech-in-noise sources in the experiments, since emphasis is placed here on improving speech intelligibility when the patient is immersed in background noise and, in this respect, in an effort of reducing data to be stored in RAM memory, the "amplification program" fitted to these both different listening conditions can be considered as the same amplification program.

Turning again our attention to audio files included in the database, it is convenient to mention that these files are monophonic, have been sampled with a sampling frequency of 16 kHz, and have been coded with 16 bits per sample. Both speech and music files were provided by D. Ellis, and were recorded by Scheirer and Slaney [Scheirer and Slaney, 1997]. This database [Scheirer, 2006] has already been used in a number of different works [Scheirer and Slaney, 1997; Williams and Ellis, 1999]. The designers made a strong attempt to collect a data set which represented as much of the breadth of available input signals as possible as follows:

- Speech was recorded by digitally sampling FM radio stations, using a variety of stations content styles and levels, and was recorded from a uniformly distributed set of male and female speakers. The sound files exhibit different input levels, with a range of 30 dB between the lowest and the highest, which allows us to test the robustness of the systems explored against different sound input levels.

- Music sounds include samples of jazz, pop, country, salsa, reggae, classical, various non-Western styles, various sorts of rock, and new age music, both with and without vowels.

Concerning noise files, it is worth mentioning that noise samples contain sounds from the following diverse environments: aircraft, bus, cafe, car, kindergarten, living room, nature, school, shop, sports, traffic, train, and train station. These sources of noise have been artificially mixed with those of speech files (with varying grades of reverberation) at different signal-to-noise ratios (SNRs) ranging from 0 to 10 dB to create speech-in-noise files. In a number of experiments, these values have been found to be representative enough of the following fact related to perceptual criteria: lower SNRs could be treated by the hearing aid as noise, and higher SNRs could be considered as clean speech.

In the same line of reasoning, the sources of instrumental music have been artificially mixed with those of speech files (with varying grades of reverberation) at different SNRs ranging from 0 to 10 dB to obtain speech-in-music files.

For properly training, validation, and testing, the sound database has to be partitioned into three different subsets. We formally write this as $\mathcal{D} = \mathcal{P} \cup \mathcal{V} \cup \mathcal{T}$, where $\mathcal{P}$, $\mathcal{V}$ and $\mathcal{T}$, represent, respectively, the "training", "validation", and "test" subsets. These sets contain 2905 files ($\approx 39.8\,\%$) for training, 985 files ($\approx 13.5\,\%$) for validation, and 3409 ($\approx 46.7\,\%$) files for testing. This partitioning has been made such that each set covers entire sound classes. Since there is no clear rule in how to make this choice [Büchler, 2002], the division of the complete database has been randomly, ensuring that the relative proportion of files of each category is preserved for each set.

For the sake of clarity, Table C.1 summarizes the number of sound files for each class belonging to "train", "validation" and "test" sets.

Class	**Train**	**Validation**	**Test**	**Total**
Speech-in-quiet	299	105	403	807
Speech-in-noise	342	113	345	800
Speech-in-music	327	105	383	815
Music	969	331	1140	2440
Noise	968	331	1138	2437
Total	**2905**	**985**	**3409**	**7299**

Table C.1: Summary of the sound database created in this thesis to carry out the experiments. It is shown the number of sound files for each class belonging to "training", "validation" and "test" sets.

Completing this section requires to mention that each sound file belonging to $\mathcal{D}$ has been properly modified by the "hearing aid simulator" modeled in [Vicen-Bueno et al., 2007] and described in-depth in Appendix B, by using data from "real" hearing-impaired people.

C.2 Database of patients

In order to carry out "realistic" simulations on the performance of the hearing aid, or in other words, aiming at creating sounds similar to those hearing aids users are used to hear in their everyday life, we have considered in the experiments some important hearing aid users' parameters. These parameters rely on several factors that basically depend upon: 1) the specific hearing loss any patient suffers from and 2) the type of hearing aid. In the effort of considering the influence of these factors on the sound learning, we have studied and modeled the hearing loss of 18 well-selected "real" different patients whose main characteristics have been listed in Table C.2.

		Hearing loss (dB) as a function of frequency (Hz)								
P_i	**HA**	**125**	**250**	**500**	**1k**	**2k**	**3k**	**4k**	**6k**	**8k**
P_1	ITE	35	35	45	45	55	55	55	70	85
P_2	ITE	55	55	55	60	55	60	55	60	65
P_3	ITE	75	75	45	55	50	60	75	67	60
P_4	ITC	35	35	40	40	60	60	65	63	60
P_5	ITC	25	25	25	40	40	55	60	62	65
P_6	ITC	35	35	35	30	15	35	40	60	70
P_7	ITE	30	30	30	45	55	45	60	63	65
P_8	BTE	80	80	85	95	90	75	85	85	85
P_9	BTE	75	75	70	75	65	60	75	78	80
P_{10}	BTE	70	70	70	90	80	80	95	95	95
P_{11}	ITC	30	30	25	40	50	55	50	65	65
P_{12}	ITC	13	13	12	30	70	78	76	75	75
P_{13}	ITE	60	60	45	35	40	30	40	50	65
P_{14}	ITE	50	50	45	35	45	55	70	80	95
P_{15}	ITE	55	55	60	60	65	75	75	75	75
P_{16}	ITE	60	60	35	50	80	80	80	80	80
P_{17}	ITC	40	40	45	55	50	50	50	50	50
P_{18}	ITC	35	35	35	35	60	65	65	65	65

Table C.2: Fundamental parameters for the 18 real patients that have been found to be sufficiently representative. P_i represents a "representative" patient in group i. The second column, labeled "HA", lists the type of hearing aid (HA), ITE, ITC, and BTE meaning, respectively, "in the ear", "in the canal" and "behind the ear". Those columns labeled "125 … 8k" list the loss-level (dB) in any of these bands (from 125 Hz to 8 kHz).

We say "real" in the sense that the data illustrated in this table have been extracted from *real* people who participated in this study. For the sake of generality and making our analysis independent of particular details, we have named the number of patients throughout this thesis as N_{patients} ($N_{\text{patients}} = 18$, in this case).

Note that when we say "patient" we refer indeed to the *pair* formed by the patient and his/her hearing aid (HA). In this respect, the two first columns in Table C.2 reflect this fact: for any patient P_i in column "1" we have detailed in column "2" the model of his/her hearing aid: in the ear (ITE), in the canal (ITC) and behind the ear (BTE).

For each P-HA$_i$ (P$_i$, henceforth) in Table C.2, we have focused on one important parameter on the application at hand. This parameter is related to the specific hearing loss level (dB) the patient suffers from. Such hearing loss (dB) has been measured at the nine frequencies typically used to measure audiograms in Spain (125 Hz, 250 Hz, 500 Hz, 1 kHz, 2 kHz, 3 kHz, 4 kHz, 6 kHz and 8 kHz). Having a look at this table, one can notice that the hearing loss-level is a function that depends on the frequency band, and models this very common fact: some patients may suffer strong losses in some frequency bands, although they heard well in the remaining ones.

Finally, it is worth mentioning one last comment regarding the design of the experiments described in this thesis. It is important to notice that the different hearing loss cases are only measured for a reduced number of frequency values. In order to determine the hearing loss for an arbitrary frequency distinct from the standard values, linear interpolation (in dB) must be applied.

Appendix D

Genetic algorithms

D.1 Introduction

A genetic algorithm (GA) is an optimization and search technique which, inspired by the principles of genetics and natural selection [Haupt and Haupt, 2004], exhibits useful properties for solving certain problems that otherwise would be intractable. Among these properties, it is worth mentioning that GAs deal with a large number of variables, provide a global solution for multi-local extrema problems, optimize functions with continuous or discrete variables, optimize variables with extremely complex cost surfaces and *do not* require derivate information. These benefits basically arise from the fact that GAs are inspired by the way nature finds solutions to extremely complex problems such as, for instance, the survival-of-the-fittest individual in a changing ecosystem.

The immediate questions arising here are: *What is the analogy to our problems*? or *How to apply these concepts to our problems?* In the problems at hand, we are looking for the best candidate solution that is "best fitted", or in other words, the one that *minimizes the fitness function* (sometimes also called *objective function*). Any possible candidate vector is termed "trial solution" (also named "individual" or simply "solution"). In this search, a group (or "population", using genetic terminology) of trial solutions are evaluated with the aim of minimizing this fitness function. How well the trial solution solves the problem at hand is the so-called "fitness" of the individual.

For this particular terminology to be understood, it is important to have a brief look at the following two biological items: 1) genetic information codifies the external characteristics and abilities of living organisms, and 2) evolution is the result of the interaction between the random creation of novel genetic information and the evaluation and selection of those novel individuals that are best adapted to the changing environment. With these two ideas in mind, we can proceed further: prior to designing the optimization algorithm *itself* (Section D.1.2), it is indispensable for our data to be represented or encoded into a more suitable form (Section D.1.1) that allows GAs to work more efficiently.

D.1.1 Codifying the trial solution

In the biological analogy, the genetic information that encodes and causes the external characteristics of a living organism (or individual) is called "genotype". Any particular characteristic produced by a piece of this genetic information is encoded by a "gene", the "chromosome" being the set of these genes. Each gene is located at a particular position on the chromosome and can have different values, called "allele".

This strategy can be considered as transforming the real search space into another in which working is much easier. From a mathematical viewpoint, if $\mathbb{F}$ is the set containing all the trial solutions, and $\mathbb{G}$ is the set of chromosomes that codifies them, this representation is equivalent of defining a bijection: $\epsilon : \mathbb{F} \rightarrow \mathbb{G}$, in the sense that any trial solution is represented by an unique chromosome so that $\mathbf{C} = \epsilon(\mathbf{F})$. The importance of this bijection will be better understood in the section that immediately follows.

Completing this section demands to mention that since data in the chromosome to be optimized are usually real numbers or bits, a proper way to represent this chromosome is a real-number vector or a binary string, respectively.

D.1.2 Applying genetic mechanisms

The way a GA works is inspired by Evolution. This is the result of the interaction between the *random* creation of novel genetic information and the *evaluation* and *selection* of those novel individuals that are best adapted to the changing environment. In nature, the random creation of novel genetic information may lead to the ability to survive. The better an individual is suited to an environment, the higher the probability of survival is. This is the so-called "survival of the fittest", and the longer the individual's life is, the higher its chances of having descendants are. In this procreation process, the parent chromosomes are combined ("recombination") to provide a novel chromosome. Sporadically, and because of unavoidable errors in copying genetic information or external factors (for instance, radiation), *mutations* (random variations) occur. The consequence is the creation of a generation of living beings with some novel characteristic which are slightly different from those of their progenitors. If the new attribute makes the offspring, or in other words, the novel trial solutions, better suited to the varying environment, the probabilities of survival and having descendants also increase. Part of offspring could inherit the modified genes and the corresponding external characteristic. In this way, the population of individuals evolves and, after a number of generations, the described process results in the creation of individuals better adapted to the environment and in the extinction of those worst suited.

In our problems, the creation of novel chromosomes that encode *novel trial solutions* for a number of consecutive generations has been carried out as described in the paragraphs below.

1. Creating the initial population of chromosomes, $P_{\text{chr,ini}} = \{\mathrm{C}_1, \ldots, \mathrm{C}_{\mathrm{p}_0}\}$.
   The number p_0 of chromosomes in the initial population is a crucial issue for GA performance [Goldberg, 1989; Haupt and Haupt, 2004]. On the one hand, a large population could cause more genetic diversity (and thus, a higher search space) and, consequently, suffer from slower convergence. On the other hand, with a very small population, only a reduced part of the search space is explored, thus increasing the risk of prematurely converging to a local extreme. In the batches of experiments carried out in this work, the initial population size p_0 has been found to be a trade-off between computational complexity and performance.

2. To select a subpopulation of chromosomes that encode the "best suited" candidate solutions.
   As stated beforehand, here "best suited" means those trial solutions that minimize the fitness function. Note that the fitness function, which is different for each particular problem, is the only connection between the real problem to be solved and its genetic representation. This means that it is necessary for a chromosome to be selected to be previously decoded using ϵ^{-1}, the inverse of the coding bijection ϵ.

In a more detailed way, the ulterior selection process is composed of the following steps: 2.1) Estimate the fitness function of every individual composing the population p_0. 2.2) Update, if necessary, the minimum value of fitness function and store the corresponding trial solution. 2.3) Select, using a stochastic uniform function, the mates and a small amount of *elite* individuals to survive the next generation. To select the mates, a line is laid out in which each parent corresponds to a section of the line of length proportional to its scaled value. The algorithm moves along the line in steps of equal size. At each step, the algorithm allocates a parent from the section it lands on. The first step is a uniform random number less than the step size. Note that not all of the worst individuals in the current generation are replaced. This replacement strategy, called "steady state approach", prevents the algorithm from prematurely converging to a local minima.

3. Recombining the parent chromosomes by making use of a "crossover" operator.
   In the effort of exploring other solutions in the search space, variations are introduced into the intermediate population. Crossover is a genetic operator that combines two pairs of chromosomes ("parents") picked at random from the current population to produce a new chromosome ("offspring"). The objective is that the new chromosome be better than both of the parents since it takes the best characteristics form each of the parents.

   In particular, the crossover operator we have implemented is a stochastic function that randomly generates a binary vector and selects the genes where the vector is "1" from the first parent, and the genes where the vector is "0" from the second parent, combining these genes to form the offspring. The probability that crossover operator is applied to each individual is called *crossover probability*, labelled p_c ($p_\mathrm{c} < 1$) and is a parameter to be optimized. After a number of experiments, it has been found to be $p_c = 0.8$. It is important to note that *not* all individuals are selected for crossover.

4. To mutate or randomly change the offspring chromosome.
   The main purpose of a mutate operator is to maintain diversity within the population and inhibit premature convergence to local minima. Similarly, not all the offspring chromosomes are mutated: the probability that the mutation operator is applied is called *mutation probability*, labelled p_m ($p_\mathrm{m} < 1$). Empirically, it has been observed that a value of $p_m = 0.1$ gives good results.
   With this in mind, a mutation operator, consisting in changing an "allele" with a probability of $p_m = 0.1$ (uniform distribution), has been applied.

Finally, the algorithm iterates steps 2-4 until a maximum number of generations ($N_\mathrm{generations}$) is reached or if the lowest fitness function value remains unchanged for a given number of iterations ($N_\mathrm{iterations}$). Both $N_\mathrm{generations}$ and $N_\mathrm{iterations}$ values must be found to be large enough to allow the algorithm to properly converge in the experiments.

Appendix E

Implementing new features

This appendix aims at helping the reader understand the assembly source code programmed in this thesis that intends for computing in the DSP, used by our platform to carry out the experiments, the feature vector that properly characterizes the input sound signal for a further classification.

E.1 Implementation of a new building block

In the effort of programming the computation of a new building block, labelled, for instance, BB_XXX, the following sequence of operations need to be carried out:

1. Modifying the file `def_features.inc`:

   - In the assembly code corresponding to the programming of the features composing the feature vector (or in other words, the features included in the set $\mathcal{S}_{\mathrm{F}}$), if we have implemented a feature, such as, for instance, the feature CF_YYY, that needs for its implementation the new building block to be programmed, that is, the building block BB_XXX, it is necessary to include the following assembly code in the file `def_features.inc`, aiming at establishing a relationship between the feature CF_YYY and the new building block BB_XXX.

     ```
     #ifdef  CF_YYY
        #ifndef  BB_ENE
            #define  BB_ENE
        #endif

        / / Assembly code to include / /
        #ifndef BB_XXX
         #define BB_XXX
        #endif
        / / / / / / / / / / / / / / / /

        #define CF_YYY_P  CF_CONT
        #set CF_CONT  (CF_CONT+1)
     #endif
     ```

     By means of this sequence of instructions, when we compile the assembly code of the feature CF_YYY, the assembly code *strictly* necessary to compute the building block BB_XXX is automatically generated.

- In the assembly code corresponding to the programming of the building blocks, we have defined 1) a preprocessor variable, labelled `BB_CONT`, that intends for counting the number of building blocks that are compiled, and 2) a variable per each building block, labelled `BB_XXX_P`, which basically points at the location of the corresponding building block in the data-memory available in the DSP. Note that the location in the memory of the new building block to be programmed will be determined as follows: `BB_INITIAL+2*BB_XXX_P`. With this in mind, the fact of programming the new building block involves including the corresponding assembly code:

```
. . .
  #ifdef  BB_SSC
  #set BB_SSC_P  BB_CONT
  #set BB_CONT (BB_CONT+1)
#endif

/ / Assembly code to include / /
#ifdef BB_XXX
  #set BB_XXX_P  BB_CONT
  #set BB_CONT  (BB_CONT+1)
#endif
/ / / / / / / / / / / / / / / /
. . .
```

2. Modifying the file `features.inc`:
   Apart from the mentioned assembly code to be included in the file `features.inc`, it is also necessary to introduce the sequence of instructions that allow us to compute this new building block BB_XXX in the file `features.inc`, and, since the time for running this routine will be increased, it is also necessary to include the preprocessor code needed to estimate the new computational cost. Please note that we *do not* show here the assembly code to be included, because this code will obviously depend on the new building block to be programmed.

   Furthermore, it is worth mentioning that we have put special emphasis on the order we compute a building block in this thesis. In this sense, the register `R0` always points at the location in the memory available in the DSP in which the sequence of instructions implemented to compute a building block are stored and consequently, thanks to using post-increase instructions, it is not necessary to initialize the mentioned register for each new instruction, which significantly reduces the computational cost.

E.2 Implementation of a new feature

Let us imagine that we would like to program a new feature labelled, for instance, CF_ZZZ. To achieve this, the following sequence of instructions need to be performed:

1. Modifying the file `def_features.inc`:

   - In the assembly code corresponding to the computation of the statistical measures, that is, the average and variance measures, it is necessary to include a sequence of preprocessor instructions in the file `def_features.inc`, that aims to calculate both the average and variance measures of the new feature to program, that is, CF_ZZZ. The assembly code to be included for this purpose is as follows:

```
/ / Assembly code to include / /
#ifdef FM_AVG_ZZZ
  #ifndef CF_ZZZ
    #define CF_ZZZ
  #endif
  #set FM_AVG_ZZZ_P FM_CONT
  #set FM_CONT (FM_ZZZ+1)
#endif

#ifdef FM_VAR_ZZZ
  #ifndef CF_ZZZ
    #define CF_ZZZ
  #endif
  #set FM_VAR_ZZZ_P FM_CONT
  #set FM_CONT (FM_ZZZ+1)
#endif
/ / / / / / / / / / / / / / / /
```

Please note that the variable `FM_CONT` simply indicates the location in the data-memory available in the DSP in which the statistical measures of the features are located.

- In the assembly code corresponding to the programming of the features that feed the classifier (or in other words, the features included in the set $\mathcal{S}_{\mathcal{F}}$), it is necessary to make sure that the preprocessor constants corresponding to the building blocks necessary for the computation of the new feature are defined. For illustrative purposes, let us imagine that the new feature to be programmed depends on both the BB_SSC and BB_STE building blocks. This scenario would involve including the following assembly code:

```
/ / Assembly code to include / /
#ifdef CF_ZZZ
  #ifndef BB_SSC
    #define BB_SSC
  #endif
  #ifndef BB_STE
    #define BB_STE
  #endif
  #set CF_ZZZ_P CF_CONT
  #set CF_CONT (CF_ZZZ+1)
#endif
/ / / / / / / / / / / / / / / / /
```

2. Modifying the file `features.inc`:

- In the routine `Features`, we include the assembly code that allows us to compute the new feature to be programmed, that is, CF_ZZZ, as a function of the building blocks that it depends on. After that, the macro "statistical characterization" is called.
- In the routine `Final_features`, we include the assembly code required to call the necessary macros. Since in the assembly code designed for this thesis, only the macro `Final_features` is implemented, the assembly code to be included is as follows:

```
/ / Assembly code to include / /
#ifdef FM_VAR_ZZZ
```

```
        VAR_FINAL(Final_features_X+(2*FM_AVG_ZZZ_P),Final_features_X+(2*FM_VAR_ZZZ_P))
        #set TOTAL_CYCLES (TOTAL_CYCLES+COST_VAR_FINAL)
      / / / / / / / / / / / / / / / /
```

3. Modifying the file `gen_constants.inc`:
   This file contains all the compiler options and thus, it is necessary to include in it the compiler options that define the computation of the statistical measures of the new feature to be programmed. This is as follows:

```
#define FM_AVG_NSC
#define FM_AVG_SSC
#define FM_AVG_NVW
#define FM_AVG_SVW
#define FM_AVG_STE
#define FM_AVG_SSF
#define FM_AVG_ZZZ  / / Assembly code to include / /

#define FM_VAR_NSC
#define FM_VAR_SSC
#define FM_VAR_NVW
#define FM_VAR_SVW
#define FM_VAR_STE
#define FM_VAR_SSF
#define FM_VAR_ZZZ  / / Assembly code to include / /
```

E.3 Implementation of a new statistical measure

Let us now imagine that we would like to program a new statistical measure labelled, for instance, FM_AAA. The sequence of operations required for this purpose is as follows:

1. Modifying the file `def_carac.inc`:
   In the assembly code corresponding to the computation of the statistical measures, it is necessary to include the assembly code that allows us to calculate the new statistical measure for each of the features programmed in the DSP. For the sake of clarity, we show below the assembly code to be included to calculate this new statistical measure of the feature labelled CF_ZZZ.

```
/ / Feature CF_ZZZ / /
#ifdef FM_AVG_ZZZ
  #ifndef CF_ZZZ
    #define CF_ZZZ
  #endif
  #set FM_AVG_ZZZ_P FM_CONT
  #set FM_CONT (FM_CONT+1)
#endif

#ifdef FM_VAR_ZZZ
  #ifndef CF_ZZZ
    #define CF_ZZZ
  #endif
  #set FM_VAR_ZZZ_P FM_CONT
  #set FM_CONT (FM_CONT+1)
#endif

/ / Assembly code to include / /
#ifdef FM_AAA_ZZZ
  #ifndef CF_ZZZ
    #define CF_ZZZ
  #endif
  #set FM_AAA_ZZZ_P FM_CONT
  #set FM_CONT (FM_CONT+1)
#endif
/ / / / / / / / / / / / / / / /
```

2. Modifying the file `features.inc`:

   - Creating the corresponding macros: note that the computation of the statistical measures considered in this thesis (average and variance) is carried out by means of two different macros: 1) a macro that is carried out each analysis frame update and 2) a macro that is run when there are 512 frames. In this sense, it is possible to obtain the new statistical measure by using only one macro.
   - In the routine `Features`, for each feature programmed in the DSP, it is necessary to run the macro that computes the new statistical measure, after running the sequence of instructions that computes each feature. To understand this, we depict below the assembly code to be included, which intends for computing the new statistical measure for the feature labelled CF_NVW.

```
#ifdef CF_NVW
  LDI R1, BB_INITIAL_X+(2*BB_V2W_P)+1
  LDI R2, BB_INITIAL_X+(2*BB_STE_P)+1
  LD AH, (R1-)
  SUB A, (R2-)
  ADSI A, 1
  LD EXP, AH
  LD AH, (R2)
  SHFT 16-1-1-BITS_INV_TABLE
  ROUND_UP
  ADD A, INV_TABLE_POINTER_Y_L,Y
  LD R5, AH
  MSET (R5), (R1)
  MUL A
  LD (R6+), AH
  LD (R6-), AL
  #set  FEATURES_CYCLES (FEATURES_CYCLE+16)
  #ifdef FM_AVG_NVW
     AVERAGE
     #set FEATURES_CYCLES (FEATURES_CYCLE+AVERAGE_COST)
  #endif
  #ifdef FM_AVG_NVW
     VARIANCE
     #set FEATURES_CYCLES (FEATURES_CYCLES+VARIANCE_COST)
  #endif
```

```
/ / Assembly code to include / /
#ifdef FM_AAA_NVW
   AAA
   #set FEATURES_CYCLES (FEATURES_CYCLES+COST_AAA)
  #endif
/ / / / / / / / / / / / / / / /
#endif
```

- In the routine `Final_features`, if the new statistical measure to be programmed requires a last operation after updating *all* the features comprising a time slot, it is necessary to run the macro that computes the aforementioned last operation for each of the features in the routine `Final_features`. To better clearly explain this, let us imagine that we would like to compute the new statistical measure of the feature labelled CF_SSF, which requires this last operation. The assembly code to be included is as follows:

```
     / / Assembly code to include / /
    #ifdef FM_AAA_SSF
         AAA_FINAL(Parameter1, Parameter2)
  #set CICLOS_FINAL (CICLOS_FINAL+COST_AAA_FINAL)
    #endif
   / / / / / / / / / / / / / / / /
#endif
```

3. Modifying the file `const_gen.inc`:
   Aiming at the new statistical measure can be included in the feature vector that finally feeds the classifier, it is necessary to include the following instructions in the file `const_gen.inc`.

```
#define FM_AVG_NSC
#define FM_AVG_SSC
#define FM_AVG_NVW
#define FM_AVG_SVW
#define FM_AVG_STE
#define FM_AVG_SSF

#define FM_VAR_NSC
#define FM_VAR_SSC
#define FM_VAR_NVW
#define FM_VAR_SVW
#define FM_VAR_STE
#define FM_VAR_SSF

/ / Assembly code to include / /
#define FM_AAA_NSC
#define FM_AAA_SSC
#define FM_AAA_NVW
#define FM_AAA_SVW
#define FM_AAA_STE
#define FM_AAA_SSF
/ / / / / / / / / / / / / / / /
```

Bibliography

M. Agarwal. A systematic classification of neural-network-based control. In *Journal of Guidance, Control and Dynamics*, pages 75–93, April 1997.

E. Alexandre, Lorena Alvarez, Lucas Cuadra, Manuel Rosa, and Francisco Lopez. Automatic sound classification algorithm for hearing aids using a hierarchical taxonomy. In Andrzej Dobrucki, Alexander Petrovsky, and Wladyslaw Skarbek, editors, *New Trends in Audio and Video*, volume 1. Politechnika Bialostocka, 2006a.

E. Alexandre, L. Cuadra, L. Alvarez, M Rosa, and Francisco Lopez. Automatic sound classification for improving speech intelligibility in hearing aids using a layered structure. In *Lecture Notes in Computer Science*. Springer Verlag, 2006b.

E. Alexandre, L. Cuadra, L. Álvarez, M. Rosa-Zurera, and F. López-Ferreras. Two-layer automatic sound classification system for conversation enhancement in hearing aids. *Integrated Computer-Aided Engineering*, 15(1):85–94, 2008a.

E. Alexandre, L. Cuadra, M. Rosa, and F. López. Automatic sound classification algorithm for hearing aids using a hierarchical taxonomy. In B. Prasad and S. M. Prassana, editors, *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks*, pages 145–167. Springer Verlag, 2008b.

E. Alexandre, L. Cuadra, M. Rosa-Zurera, and F. López-Ferreras. Speech/non-speech classification in hearing aids driven by tailored neural networks. In *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks*, volume 83/2008, pages 145–167. Springer Verlag, 2008c.

E. Alexandre, R. Gil-Pita, L. Cuadra, L. Álvarez, and M. Rosa-Zurera. Speech/muisc/noise classification in hearing aids using a two-layer classification system with mse linear discriminants. In *16th European Signal Processing Conference (EUSIPCO)*, Lausanne, Switzerland, 2008d.

Lorena Álvarez, Enrique Alexandre, Lucas Cuadra, and Manuel Rosa-Zurera. On the training of multilayer perceptrons for speech/non-speech classification in hearing aids. In *Audio Engineering Society Convention 122, Preprint #7136*, Viena, Austria, 5 2007. URL http://www.aes.org/e-lib/browse.cfm?elib=14121.

Javier Amor. Aplicación de algoritmos heurísticos bioinspirados a la selección automática de características. M. S. Degree, University of Alcalá, December 2008.

Tobias Andersson. Audio classification and content description. Master's thesis, Lulea University of Technology, Lulea, Sweden, 2004.

ANSI. Methods for calculation of the speech intelligibility index. Technical report, American National Standards Institute, 1997.

Archelon. *MCPP Macro Preprocessor Version 2.10.* Archelon Inc., 1994.

Kathryn Hoberg Arehart, John H. L. Hansen, Stephen Gallant, and Laura Kalstein. Evaluation of an auditory masked threshold noise suppression algorithm in normal-hearing and hearing-impaired listeners. *Speech Communication*, 40(4):575–592, 2003.

S. Asmussen. A probabilistic look at the wiener-hopf equationt. *SIAM Review*, 40(12): 189–201, June 1998.

S. Aviyente and W. J. Williams. Information bounds for random signals in time-frequency plane. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 6, pages 3549–3552, May 2001.

Amanda S. Azman. Estimating the performance of sound restoration hearing protectors using the speech intelligibility index. In *Proceedings of the International Mechanical Engineering Congress Exposition (IMECE 2010)*, Vancouver, British Columbia, Canada, 2010. ASME 2010.

Neil Bauman, 2011. URL http://www.hearingaidmuseum.com/. The Hearing Aid Museum.

J. G. Beerends, A. P. Hekstra, A. W. Rix, and M. P. Hollier. Perceptual Evaluation of Speech Quality (PESQ) The New ITU Standard for end-to-end Speech Quality Assessment Part II–Psychoacoustic Model. *J. Audio Eng. Soc.*, 50(10):765–778, October 2002.

Jan A. Beerends, John G.; Stemerdink. A perceptual audio quality measure based on a psychoacoustic sound representation. *J. Audio Eng. Soc*, 40(12):963–978, 1992. URL http://www.aes.org/e-lib/browse.cfm?elib=7019.

Jan A. Beerends, John G.; Stemerdink. A perceptual speech-quality measure based on a psychoacoustic sound representation. *J. Audio Eng. Soc*, 42(3):115–123, 1994. URL http://www.aes.org/e-lib/browse.cfm?elib=6957.

E. Bennion. *Antique Hearing Aids.* London: Venier Press, 1994.

R. A. Bentler and M. Duve. Comparison of hearing aids over the 20th century. *Ear and Hearing*, 21(6):625–639, December 2000.

C.M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press. Oxford, 1995.

S. F. Boll. Suppression of acoustic noise in speech using spectral subtraction. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(2):113–120, April 1979.

D. S. Broomhead and D. Lowe. Multivariate functional interpolation and adaptive networks. *Complex Sytems*, 2:321–355, 1998.

M.C. Büchler. *Algorithms for sound classification in hearing instruments.* PhD thesis, Swiss Federal Institute of Technology, Zurich, 2002.

M.C. Büchler, S. Allegro, S. Launer, and N. Dillier. Sound classification in hearing aids inspired by auditory scene analysis. *EURASIP Journal on Applied Signal Processing*, 2005(18):2991–3002, 2005.

M. J. Carey, E. S. Parris, and H. Lloyd-Thomas. A comparison of features for speech, music discrimination. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, March 1999.

S. Chen, S.A. Billings, C. Couan, and P. M. Grant. Practical identification of narmax models using radial basis function. In *Int. J. Control*, volume 52, pages 1327–1350, 1990.

W. Chou and L. Gu. Robust singing detection in speech/music discriminator design. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2001.

S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. In *Machine Learning*, volume 10, pages 57–78, 1993.

L. Cuadra, R. Gil-Pita, E. Alexandre, and M. Rosa-Zurera. Joint design of gaussianized spectrum-based features and least-square linear classifier for automatic acoustic environment classification in hearing aids. *Signal Processing*, 90(8):2628 – 2632, 2010. ISSN 0165-1684. doi: DOI:10.1016/j.sigpro.2010.02.024. Special Section on Processing and Analysis of High-Dimensional Masses of Image and Signal Data.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematical Control Signal Systems*, 2:303–314, 1989.

Howard Demuth and Mark Beale. Neural network toolbox for use with matlab, 1993.

T. V. den Bogaert, S. Doclo, J. Wouters, and M. Moonen. Speech enhancement with multichannel wiener filter techniques in multimicrophone binaural hearing aids. *The Journal of the Acoustical Society of America*, 125(1):360–371, 2009.

H. Dillon. *Hearing Aids*. Boomerang Press; Thieme, 2001. ISBN 1588900525.

Dspfactory. *Toccata Plus EDK 2.3.1 Getting Started Guide*, 2002a.

Dspfactory. *Toccata Plus Evaluation and Development Board Manual*. Dspfactory Ltd., 2002b.

Dspfactory. *Toccata Plus Rcore Manual*, 2002c.

Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. New York Wiley, 1973.

Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2001.

Brent Edwards. The future of hearing aid technology. *Trends Amplification*, 11(1):31–46, 2007. URL http://tia.sagepub.com/content/11/1/31.

Koen Eneman, Heleen Luts, Jan Wouters, Michael Büchler, Norbert Dillier, Wouter Dreschler, Matthias Froehlich, Giso Grimm, Volker Hohmann, Rolph Houben, Arne Leijon, Anthony Lombard, Dirk Mauler, Marc Moonen, Henning Puder, Michael Schulte, Ann Spriet, and Matthias Vormann. Evaluation of signal enhancement algorithms for hearing instruments. In *Proceedings of the 16th European Signal Processing Conference (EUSIPCO'08)*, 2008.

B. Everitt. *Cluster Analysis*. New York: Halsted Press, 1974.

Claire F. Fang, Rob A. Rutenbar, and Tsuhan Chen. Fast, accurate static analysis for fixed-point finite-precision effects in dsp designs. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, ICCAD '03, pages 275–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 1-58113-762-1. doi: http://dx.doi.org/10.1109/ICCAD.2003.68. URL http://dx.doi.org/10.1109/ICCAD.2003.68.

F. Fang, Tsuhan Chen, and Rob A. Rutenbar. Lightweight floating-point arithmetic: Case study of inverse discrete cosine transform. In *EURASIP Journal on Signal Processing, Special Issue on Applied Implementation of DSP and Communication Systems*, volume 9, pages 879–892, September 2002.

J. Fernández-Cruza. Diseño e implementación de un audífono digital con capacidad de adaptación al entorno sonoro. M. S. Degree, University of Alcalá, October 2009.

Jill B. Firzst. *Effect of unilateral hearing loss on the speech-evoked auditory brainstem response in the presence of noise.* PhD thesis, Washington University School of Medicine, 2010.

P. Flandrin, R.G. Baraniuk, and O. Michael. Time-frequency complexity and information. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 3, pages 329–332, April 1994.

F. Dan Foresee and Martin T. Hagan. Gauss-newton approximation to bayesian learning. *Proceedings of the 1997 International Joint Conference on Neural Networks*, pages 1930–1935, 1997.

J. Franke and E. Mandler. A comparison of two approaches for combining the votes of cooperating classifiers. In *Proceedings of the 11th International Conference on Pattern Recognition*, volume 2, pages 611–614, 1992.

N R French and J C Steinberg. Factors governing the intelligibility of speech sounds. *Journal of the Acoustical Society of America*, 19(1):90–119, 1947.

J. Fukunaga and R. Beauregard. An optimal global nearest neighbor metric. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 3:314–338, 1984.

GAO. Deaf and hard of hearing children: Federal support for developing language and literacy. Technical report, United States Government Accountability Office, May 2011.

R. Gil-Pita, E. Alexandre, L. Cuadra, R. Vicen-Bueno, and M. Rosa-Zurera. Analysis of the effects of finite precision in neural network-based sound classifiers for digital hearing aids. In *EURASIP Journal on Advances in Signal Processing*, volume 2009, pages 1–10, 2009.

Roberto Gil-Pita. *Sistemas de Clasifiación de Blancos Radar mediante Métodos Estadísticos y de Inteligencia Artificial.* PhD thesis, University of Alcalá, November 2006.

D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley publishing, Reading, MA, 1989.

Sandra Gordon-Salant. Hearing loss and aging: new research findings and clinical implications. *Journal Of Rehabilitation Research And Development*, 42(4 Suppl 2):9–24, 2005. URL http://www.rehab.research.va.gov/jour/05/42/4suppl2/gordon-salant.html.

Enric Guaus and Eloi Batlle. A non-linear rhythm-based style classification for broadcast speech-music discrimination. In *AES 116th Convention*, 2004.

M.T. Hagan and M.B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, 1994.

Ulrich Halka and Ulrich Heute. A new approach to objective quality-measures based on attribute-matching. *Speech Communication*, 11(1):15–30, 1992.

V. Hamacher, J. Chalupper, J. Eggers, E. Fischer, U. Kornagel, H. Puder, and U. Rass. Signal processing in high-end hearing aids: State of art, challengues, and future trends. *EURASIP Journal on Applied Signal Processing*, 18:2915–2929, 2005.

M. Hansen and B. Kollmeier. Objective modeling of speech quality with a psychoacoustically validated auditory model. *Journal Audio Engineering Society*, 48:395–408, May 2000.

Randy L. Haupt and Sue Ellen Haupt. *Practical genetic algorithms, 2nd Edition*. John Wiley & Sons, Inc., New Jersey, 2004.

Hear-it, 2011. URL http://www.hear-it.org/page.dsp?area=134. Hear-it.

J. Hellgren, T. Lunner, and S. Arlinger. System identiïňĄcation of feedback in hearing aids. *The Journal of the Acoustical Society of America*, 105(6):3481–3496, June 1999.

M.A. Hersh and M.A. Johnson. *Assistive Technology for the Hearing Impaired, Deaf and Deafblind*. Springer, London, UK, 2003. URL http://eprints.gla.ac.uk/33030/.

R. Hetch-Nielsen. *Neurocomputing*. Addison-Wesley, New York, 1990.

H. Hirsch and D. Pearce. The aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions. In *ISCA ITRW ASR2000*, pages 181–188, 2000.

M. P. Hollier, D. R. Guard, and Malcolm J. Hawksford. Characterization of communications systems using a speech-like test stimulus. In *Audio Engineering Society Convention 93*, 10 1992. URL http://www.aes.org/e-lib/browse.cfm?elib=6738.

M. P. Hollier, M. O. Hawksford, and D. R. Guard. Error activity and error entropy as a measure of psychoacoustic significance in the perceptual domain. *IEEE Proc. Vision, Image and Signal Processing*, 141(3):203–208, 1994.

Danoush Hosseinzadeh and Sridhar Krishnan. On the use of complementary spectral features for speaker recognition. In *EURASIP Journal on Advances in Signal Processing*, volume 2008, pages 1–10, 2008.

Yi Hu and Philipos C. Loizou. A comparative intelligibility study of single-microphone noise reduction algorithms. *The Journal of the Acoustical Society of America*, 122(3): 1777–1786, 2007. doi: 10.1121/1.2766778.

Ester Huerta, Enrique Alexandre, Roberto Gil-Pita, Lorena Álvarez, , and Javier Amor. Analysis of the effects of finite precision in sound classifiers for digital hearing aids. In *AES 124th Convention, Preprint #7420*, Amsterdam, The Netherlands, May 2008.

A. Hyvaarinen, J. Karhunen, and A. Oja. *Independent Component Analysis*. Wiley, New York, 2001.

ITU-R. Method for objective measurements of perceived audio quality. Recommendation BS.1387, International Telecommunications Union, Geneva, 1998.

ITU-T. Review of validation tests for objective speech quality measures. Document COM 12-74, International Telecommunications Union, Geneva, 1996a.

ITU-T. Report of the question 13/12 rapporteur's meeting, solothurn, switzerland. Document COM 12-117, International Telecommunications Union, 2000.

ITU-T. Telecommunication objective objective speech quality asessment. Document COM 12-34, International Telecommunications Union, 1997.

ITU-T. Mean opinion score (MOS) terminology. Recommendation P.800.1, International Telecommunications Union, Geneva, 2001a.

ITU-T. Objective quality measurement of telephone-band (300–3400 Hz) speech codecs. Recommendation P-861, International Telecommunications Union, Geneva, 1996b.

ITU-T. Perceptual evaluation of speech quality (pesq), an objective method for end-to-end speech quality assessment of narrowband telephone networks and speech codecs. Recommendation P-862, International Telecommunications Union, Geneva, 2001b.

Ozgur Izmirli. Using spectral flatness based feature for audio segmentation and retrieval. Technical report, Center for Arts and Technology, Department of Mathematics and Computer Science, Connectucut College, 1999.

N. S. Jayant and Peter Noll. *Digital Coding of Waveforms - Principles and Applications to Speech and Video*, page 688. Kai Fa Book Company, Taipei, Taiwan, 1990.

I.T. Jolliffe. *Principal Component Analysis*. Wiley, New York, 1986.

J.E. Dennis Jr. and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

J. M. Kates. *Digital Hearing Aids*. Plural Publishing Inc., 2008.

G. Keidser. Selecting different amplification for different listening conditions. *J. of the American Academy of Audiology*, 7:92–104, 1996.

Mead C. Killion. Snr loss: I can hear what people say, but i can't understand them. *The Hearing Review*, 4(12):8–14, 1997. URL http://www.etymotic.com/pdf/erl-0037-1997.pdf.

F. Kimura and M. Shridhar. Handwritten numerical recognition based on multiple algorithms. *Pattern Recognition*, 24(10):969–983, 1991.

Josef Kittler, Ieee Computer Society, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, March 1998.

S. Kochin. Marketrak vi: Consumers rate improvements sought in hearing instruments. *The Hearing Review*, 2002.

S. Kochin. 25-year trends in the hearing health market. *The Hearing Review*, 11(16):11–31, 2009. URL http://www.betterhearing.org/pdfs/M8_hearing_loss_trends_2009.pdf.

S. Kochkin. Why my hearing aids are in the drawer: The consumer's perspective. *The Hearing Journal*, 2(53):34–42, February 2000. URL http://www.betterhearing.org/hia/publications/MR41.PDF.

R. Kohavi and G.H. John. Wrappers for features subset selection. *Int. Journal Digit. Libr.*, 1:108–121, 1997.

T. Kohonen. *Self-organizing maps.* Springer Series in Information Sciences, New York, 3rd extended edition, 2001.

K. Kumar, K. Chanwoo, and R. Stern. Deltra-spectral cepstral coefficients for robust speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2011.

E. C. Levi. Complex-curve fitting. In *IRE Transactions on Automatic Control*, volume AC-4, pages 37–44, 1959.

Dongge Li, Ishwar K. Sethi, Nevenka Dimitrova, and Tom McGee. Classification of general audio data for content-based retrieval. *Pattern recognition letters*, pages 533–544, 2001.

J. S. Lim. Evaluation of a correlation subtraction method for enhancing speech degraded by additive white noise. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26:471–472, 1978.

Philipos C. Loizou. *Speech Enhancement: Theory and Practice.* CRC PRESS LLC, 2007.

Lie Lu, Hong Jiang Zhang, and Hao Jiang. Content analysis for audio classification and segmentation. *IEEE Transactions on speech and audio processing*, 10(7):504–516, October 2002.

A. Mancuso, L. Picinale, and G. Vercellesi. Evaluation of the perceived quality of hearing aids, 2006. DSP Application Day.

J. M. Martínez. Mpeg-7 overview (version 10). *MPEG Document ISOIEC JTC1SC29WG11*, 6828, 2004. URL http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.

Karl Mathia. A variable structure learning algorithm for multilayer perceptrons. *Intelligent Engineering Systems Through Artificial Neural Networks*, 14:93–100, 2004.

G. McLachlan and D. Peel. *Finite Mixture Models.* Wiley, New York, 2000.

J. Moody and C. J. Darken. Fast learning in network of locally-tuned processing units. *Neural Computing*, 1:281–294, 1989.

B.C. J. Moore. Information extraction and perceptual grouping in the auditory system. In *Human and Machine Perception: Information Fusion*, pages 1–12. Plenum, New York, 1997.

Brian C. J. Moore. *An Introduction to the Psychology of Hearing, Fifth Edition.* Academic Press, April 2003. ISBN 0125056281.

Brian C.J. Moore. *An introduction to the psychology of hearing.* Academic Press, third edition, 1989.

Arulkumaran Muthukumarasamy. Impact of microphone positional errors on speech intelligibility. Master's thesis, University of Kentucky, 2009.

D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural network by choosing initial values of the adaptative weigths. In *Proceedings of the IEEE First International Joint Conference on Neural Networks*, volume 3, pages 21–26, 1990.

NHIS, 2002. URL http://www.nidcd.nih.gov/health/statistics/begins.htm. National Health Interview Survey.

NIH, 2008. URL http://www.noisehelp.com/degrees-of-hearing-loss.html. National Institute of Health, News & Events.

P. Nordqvist. *Sound Classification in Hearing Instruments*. PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2004.

P. Nordqvist and A. Leijon. An efficient robust sound classification algorithm for hearing aids. *J. Acoustic Soc. Am.*, 115(6):3033–3041, 2004.

Billie Yevonne Pearce. Implementation of a variable structure neural network. Master's thesis, Auburn University, 2001.

Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=58326.

Jill. E. Preminger and Dianne J. van Tasell. Quantifying the relation between speech quality and speech intelligibility. *Journal of Speech and Hearing Research*, 38(3):714–725, June 1995.

Arunan Ramalingam and Sridhar Krishnan. Gaussian mixture modeling of short-time fourier transform features for audio fingerprinting. *IEEE Transactions on Information Forensics and Security*, 1(4):457–463, December 2006.

A. W Rix, M. P. Hollier, A. P. Hekstra, and J. G. Beerends. Perceptual Evaluation of Speech Quality (PESQ) The New ITU Standard for end-to-end Speech Quality Assessment Part II–Time-Delay Compensation. *J. Audio Eng. Soc.*, 50(10):755–764, October 2002.

Antony Rix, Richard Reynolds, and Mike Hollier. Perceptual measurement of end-to-end speech quality over audio and packet-based networks. In *Audio Engineering Society Convention 106*, 5 1999. URL http://www.aes.org/e-lib/browse.cfm?elib=8307.

Antony W. Rix, Michael P. Hollier, John G. Beerends, and Andries P. Hekstra. Pesq-the new itu standard for end-to-end speech quality assessment. In *Audio Engineering Society Convention 109*, 9 2000. URL http://www.aes.org/e-lib/browse.cfm?elib=9078.

A.W. Rix and M.P. Hollier. The perceptual analysis measurement system for robust end-to-end speech quality assessment. *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 3:1515–1518, 2000. doi: http://doi.ieeecomputersociety.org/10.1109/ICASSP.2000.861935.

RNID. Breaking the sound barrier. Technical report, Royal National Institute for Deaf People, 1999.

S. J. Roberts, D. Husmeier, L. Rezek, and W. Penny. Bayesian approaches to gaussian mixture modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1133–1142, 1998. ISSN 0162-8828.

T. Rohdenburg, V. Hohmann, R. Huber, and B. Kollmeier. Quality aspects of digital hearing aid algorithms, 2006. ITG–Fachtagung.

Y. D. Rubinstein and T. Hastie. Discriminative vs. informative learning. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, volume 3, pages 49–53. AAAI Press, 1997.

D. Rumelhart, G. Hinton, and R. Williams. Learning representation by back propagating errors. *Nature*, 323:533–536, 1986.

G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* New York: McGraw-Hill, 1983.

Robert E. Sandlin. *Handbook of hearing aid amplification.* College-Hill Press, Boston, 1990. ISBN 0316770272 (v.2).

E. Scheirer. http://labrosa.ee.columbia.edu/sounds/musp/scheislan.html, 2006. The Music-Speech Corpus.

Eric Scheirer and Malcom Slaney. Construction and evaluation of a robust multifeature speech/music discriminator. In *ICASSP*, 1997.

Todd Schneider, Robert Brennan, Peter Balsiger, and Re Heubi. An ultra low-power programmable dsp system for hearing aids and other audio applications. In *Proc. ICSPAT 1999*, pages 1–4, Orlando, Florida, USA, November 1999.

Bridget Shield, 2006a. URL http://spanish.hear-it.org/page.dsp?page=5770. Hear-it.

Bridget Shield. Evaluation of the social and economic costs of hearing impairment. Technical report, Hear-it, October 2006b.

A. Spriet. *Adaptative filtering techniques for noise reduction and acoustic feedback cancellation in hearing aids.* PhD thesis, Katholieke Univ Leuven, Leuven, 2004.

A. Spriet, G. Rombouts, M. Moonen, and J. Wouters. Adaptative feedback cancellation in hearing aids. *Journal of the Franklin Institute*, 343:545–573, 2007.

A. Spriet, K. Eneman, M. Moonen, and J. Wouters. Objective measures for real-time evaluation of adaptive feedback cancellation algorithms in hearing aids. In *16th European Signal Processing Conference (EUSIPCO)*, Lausanne, Switzerland, 2008.

M. D. Srinath, P. K. Rajasekaran, and R. Viswanathan. *Introduction to statistical signal processing with applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995. ISBN 0-13-125295-X.

S. Takeuchi, M. Yamashita, T. Uchida, and M. Sugiyama. Optimization of voice/music detection in sound data. *Proceedings of the CRAC*, September 2001.

T. Thiede, W. C. Treurniet, R. Bitto, C. Schmidmer, T. Sporer, J. G. Beerends, C. Colomes, M. K., G. Stoll, K. Brandenburg, and B. Feiten. PEAQ - The ITU standard for objective measurement of perceived audio quality. *Journal Audio Engineering Society*, 48(1/2):3–29, January/February 2000.

K. Torkkola. On feature extraction by mutual information maximization. In *IEEE ICASSP*, pages 821–824, 2002.

K. Torkkola and W.M. Campbell. Mutual information in learning feature transformations. In *17th Int. Conf. Mach. Learning*, pages 1015–1022, 2000.

D. E. Tsoukalas, J. N. Mourjopoulos, and G. Kokkinakis. Speech enhancement based on audible noise suppression. *IEEE Transactions on Speech Audio Processing*, 5:479–514, 1997.

George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302, July 2002.

Raúl Vicen-Bueno, Roberto Gil-Pita, Manuel Utrilla-Manso, and Lorena Álvarez-Pérez. A hearing aid simulator to test adaptive signal processing algorithms. *Proceedings of the IEEE International Symposium on Intelligent Signal Processing (WISP)*, pages 619–624, October 2007.

Raul Vicen-Bueno, Almudena Martinez-Leira, Roberto Gil-Pita, and Manuel Rosa-Zurera. Modified lms-based feedback-reduction subsystems in digital hearing aids based on wola filter bank. *IEEE Transactions on Instrumentation and Measurement*, 58(9): 3177–3190, 2009.

S. Voran. Objective estimation of perceived speech quality – Part I: Development of measuring normalizing block technique. *IEEE Transactions on Speech and Audio Processing*, 7(4):371–382, July 1999a.

S. Voran. Objective estimation of perceived speech quality – Part II: Evaluation of measuring normalizing block technique. *IEEE Transactions on Speech and Audio Processing*, 7:383–390, July 1999b.

Shihua Wang, Andrew Sekey, and Allen Gersho. An objective measure for predicting subjective quality of speech coders. *IEEE Journal on Selected Areas in Communications*, 10(5):819–829, 1992.

P. J. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.

J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for svms. In *NIPS*, pages 668–674, 2000.

Gethin Williams and Daniel P. W. Ellis. Speech/music discrimination based on posterior probability features. In *Proceedings of the 6th European Conference on Speech Communication and Technology (EUROSPEECH)*, Budapest, Hungary, September 1999.

Randall D. Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.9623.

Eberhard Zwicker and Hugo Fastl. *Psychoacoustics: Facts and Models (Springer Series in Information Sciences) (v. 22)*. Springer, 2nd updated ed. edition, April 1999. ISBN 3540650636.