_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

IMPLEMENTACIÓN DE UN SISTEMA EN TIEMPO REAL DE ECUALIZACIÓN DE SALAS SOBRE CPU.

Alberto Hinojosa Sancho

Tutor: María de Diego Antón

Cotutor: Miguel Ferrer

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2015-16

Valencia, 21 de noviembre de 2015

Resumen

En los sistemas de reproducción de sonido en el espacio resulta crítico compensar el entorno acústico para eliminar el efecto de la reverberación de la sala.

En el caso de los sistemas multicanal o MIMO (Multiple-Input Multiple-Output) en los que el número de altavoces para la reproducción es significativo, el uso de algortitmos adaptativos para compensar los efectos de la sala o ecualizar simplifica el proceso de ecualización al resolver de forma iterativa la obtención de los filtros compensadores multicanal. Así pues, debido a la gran complejidad y al número de filtro que necesitan estos sistemas, debe establecerse un compromiso entre las prestaciones finales que proporcionen una adecuada ecualización sin incrementar la complejidad computacional de los algoritmos.

Tomando como base el clásico algoritmo LMS, en este trabajo se implementa un sistema de compensación de salas adaptativo multicanal que emplea una versión del LMS normalizada, en el dominio de la frecuencia y funcionando en bloques de muestras particionados (PFBNLMS). Este algoritmo se ha desarrollado en C sobre una plataforma hardware basada en la CPU de un ordenador que comunica con la tarjeta de sonido MOTU haciendo uso de los drivers ASIO (Audio Stream Input Output). El sistema se ha evaluado en una sala acústica situada en el laboratorio del grupo GTAC del iTEAM de la UPV.

Resum

Els sistemes multicanal de control d'àudio tenen com a objectiu exercir algún tipus de control sobre les senyals sonores existents en determinants punts de l'espai. La compensació auditiva en sales per ecualització s'aplica en sistemes de reproducció d'àudio per a crear un entorn acústic determinat en sistemes 3D o aplicacions en què diferents sons es desitgen separar en l'espai sonor.

L'us d'algoritmes adaptatius en ecualització multicanal és determinant per a compensar els efectes en les sales. A causa de la gran compleixitat y el nombre de filtres compensadors necessaris en aquestos sistemes MIMO (multiple-input multiple-output), ha d'establir-se un compromís per a obtindre una adequada ecualització sense incrementar la complexitat del algoritme adaptatiu.

En aquest treball s'implementa un sistema de compensació de sales adaptatiu multicanal basat en la versió de l'algoritme LMS (Least Mean Square) normalitzat, dividit en blocs y en la freqüència (FBNLMS) d'aplicació en temps real programat en llenguatge C sobre una plataforma hardware basada en la CPU d'un ordinador que comunica amb una targeta de so MOTU fent ús dels drivers ASIO (Audio Stream Input Output).

Abstract

The multichannel control audio systems have as a goal to exert some type of control on the existent audible signals at certain points of the listening space. Listening room compensation finds application in audio reproduction systems that create given acoustic environment such 3D audio system or audio applications in that different sounds are supply to separate listeners in the same listening space.

The use of adaptive algorithms in multichannel equalization has become essential to compensate room effects of real sound reproduction systems. Due to the high complexity and number of

compensation filters that involve the multiple-input multiple-output (MIMO) systems, a compromise has to be taken to provide good equalization without increasing the complexity of the adaptive algorithms.

In this work it is implemented a multichannel adaptive equalization system based in a frequency block version of the normalize Least Mean Square (FBNLMS) algorithm in real time programmed in language C on a CPU platform that communicates with a audio card MOTU using the ASIO (Audio Stream Input Output) drivers.

Índice general

Capítulo 1. Introducción al Trabajo Fin de Grado.

1.1 Motivación.

Existen multitud de situaciones tanto en audio como en ámbitos del procesado de señales en comunicaciones en que se requiere modificar o compensar la respuesta de un filtro. Para ello se emplean filtros ecualizadores. De entre los diferentes tipos de ecualizadores, los algoritmos adaptativos se muestran como una solución sencilla en cuanto a su implementación y su uso. Son numerosas las aplicaciones de sonido, aparte de la ecualización de salas, en las que se ha usado de forma exitosa los algoritmos adaptativos, sirvan de ejemplo los sistemas de control activos de ruido. Por otro lado, son numerosas las aportaciones teóricas y desarrollos matemáticos de técnicas adaptativas basadas en algoritmos de la familia del algoritmo Least Mean Square (LMS). Son escasos los ejemplos de ecualización en un sistema en tiempo real y la mayoría de sistemas descritos en la literatura corresponden a simulaciones software.

1.2 Objetivos del TFG.

Este Trabajo Fin de Grado pretende implementar un sistema multicanal de compensación de salas en tiempo real aplicando un algoritmo adaptativo basado en el LMS. Este objetivo requiere una identificación previa de los canales acústicos a emplear en el sistema. Como plataforma hardware se empleará una CPU haciendo servir los drivers ASIO para la comunicación entre los equipos y el sistema.

1.3 Metodología.

El desarrollo del Trabajo Fin de Grado se ha estructurado en seis fases. Inicialmente familiarizarse con los conceptos que se hacen referencia en el proyecto, esto es un estudio del estado del arte acerca de la ecualización adaptativa en tiempo como en el dominio de la frecuencia. En segundo lugar diseñar las diferentes versiones de algoritmos adaptativos en lenguaje Matlab para simular y configurar un sistema de identificación. A continuación implementar un algoritmo ecualizador adaptativo de filtrado-x también en Matlab rediseñando lo ya desarrollado, concretamente para la versión PFBLMS. El siguiente paso, programar el algoritmo PFBLMS de filtrado-x en lenguaje C para después crear un prototipo sobre CPU en tiempo real. Tras todo ello, testear la aplicación en la sala acústica del GTAC.

Capítulo 2. Sistemas de ecualización

2.1 Introducción a tipos de filtros en ecualización.

Un problema histórico en el campo de las telecomunicaciones ha sido lograr recuperar de forma fiable la información que ha sido enviada desde algún transmisor. El filtrado de señales permite seleccionar o variar a su salida ciertas componentes relacionadas con la amplitud y la fase.

El procesamiento de la señal mediante un filtro pasivo puede eliminar o reducir la parte no deseada de la señal como puede ser el ruido o degradaciones por perturbaciones. También compensar la pérdida de potencia con filtros activos. Los filtros se clasifican en dos familias, filtros únicamente dependientes de las muestras de entrada denominados de Respuesta Finita al Impulso (FIR) y aquellos que además dependen de las muestras a la salida llamados de Respuesta Infinita al Impulso (IIR).

Con el paso del tiempo los filtros digitales como DSPs, FPGAs o microprocesadores han ido desplazando en ciertas aplicaciones a los filtros analógicos por su bajo coste y mejores prestaciones. Los DSPs se caracterizan por su alta capacidad de procesamiento ya que son capaces de multiplicar, sumar y guardar el resultado en un ciclo de reloj. Otra ventaja de los filtros digitales es la facilidad con la que se puede calcular el espectro de frecuencias. Sin embargo requiere de conversores A/D con altas frecuencias de muestreo.

Los filtros ecualizadores con el objetivo de restablecer las características de las señales pueden diseñarse con estructura fija, pero su sencillez repercute en la calidad de ecualización. Como alternativa se presenta en este trabajo algunos esquemas de filtrado adaptativo LMS, siendo estos capaces de responder ante cambios en el escenario en que se aplica.

2.2 Filtros adaptativos.

El procesado adaptativo de señales digitales es el estudio de algoritmos y técnicas capaces de variar de manera afín a las propiedades estadísticamente cambiantes de las señales reales. Estas técnicas se han aplicado exitosamente en multitud de áreas como son la identificación de la respuesta de un sistema, ecualización de canales en comunicaciones *beam forming* o conformación de haz y control dinámico de sistemas.

El esquema clásico de un sistema adaptativo es el que se muestra en la figura 2.1.

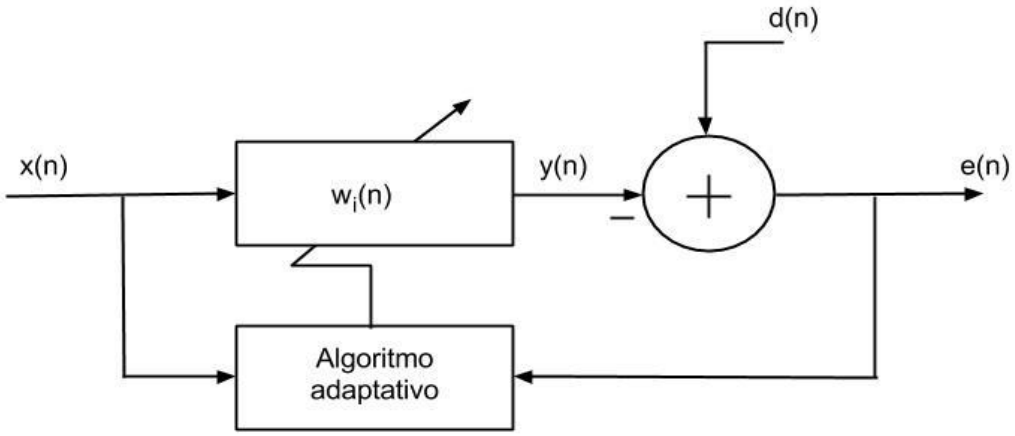


Figura 2. 1: Sistema adaptativo

Entre la nomenclatura que interviene en dicho sistema encontramos:

- d(n): señal deseada u objetivo. En aplicaciones de identificación de función de transferencia o canal corresponde a la señal respuesta del sistema.
- x(n): señal de referencia ó de entrada al filtro adaptativo.
- $w_i(n)$: filtro transversal de N coeficientes actualizados cada iteración. i: 0,1,2, …, N-1.
- y(n): señal generada a la salida del filtro adaptativo.
- e(n): señal de error obtenida como diferencia entre la señal deseada y la generada.

Los filtros adaptativos se basan en filtros digitales estructurados preferiblemente con respuestas tipo FIR (Finite Impulse Response) por su estabilidad al depender solamente de las entradas y no de valores pasados de la salida frente a filtros IIR (Infinite Impulse Response), junto a un algoritmo que ajusta los coeficientes del filtro. El algoritmo adaptativo varía la respuesta del filtro para minimizar la función error, *e(n)*, entre la señal deseada, *d(n)*, y la salida del filtro adaptativo, *y(n)*.

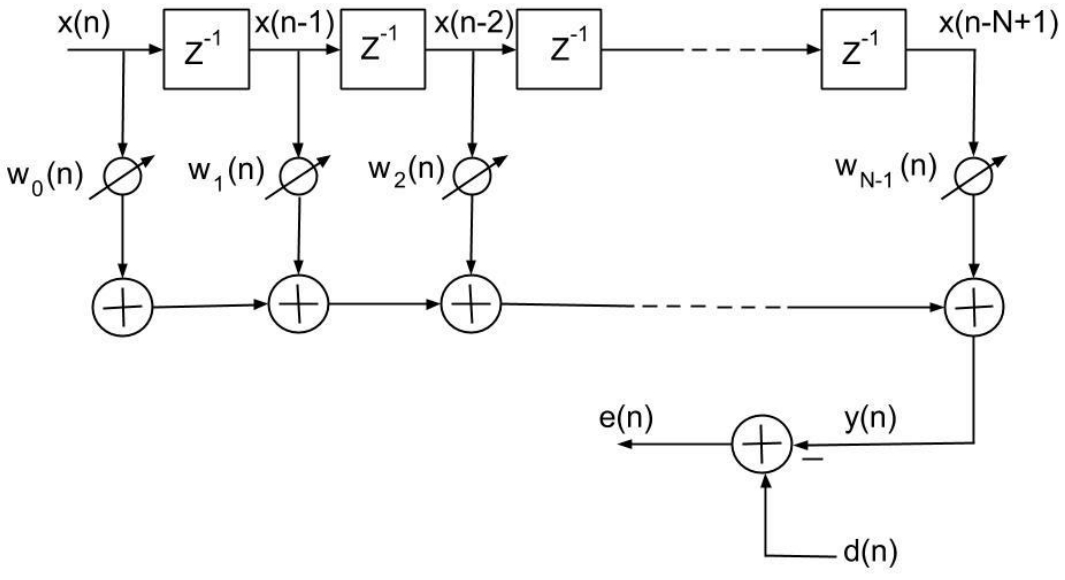


Figura 2. 2: Filtro adaptativo transversal de N etapas

La salida de un filtro transversal de N componentes como el de la Figura 2. 2 viene dada por la expresión (2. 1):

(2. 1)

$$y(n) = \sum_{i=0}^{N-1} w_i(n) \cdot x(n-i) = \boldsymbol{w}^T(n) \cdot \boldsymbol{x}(n)$$

Los pesos del filtro $w_0(n)\ w_1(n)\ \ldots\ w_{N-1}(n)$son seleccionados en cada iteración *n* para que el error diferencia (2. 2) se minimice en algún sentido.

(2. 2)

$$e(n) = d(n) - y(n) = d(n) - \boldsymbol{w}^T(n) \cdot \boldsymbol{x}(n)$$

Expresado matricialmente, la entrada del filtro es un vector columna de N muestras

(2. 3)

$$\boldsymbol{x}(n) = [x(n)\ x(n-1)\ \ldots\ x(n-N+1)]^T$$

Y los coeficientes del filtro en el instante *n*

(2. 4)

$$\boldsymbol{w}(n) = [w_0(n)\ w_1(n)\ \ldots\ w_{N-1}(n)]^T$$

Esto significa que el filtro adaptativo varía temporalmente su respuesta siguiendo continuamente una adaptación a los cambios estadísticos de las señales en un sistema desconocido.

De entre las familias existentes de algoritmos adaptativos aplicaremos un análisis teórico del LMS (Least Mean Square) que minimiza el error medio cuadrático o MSE (Mean Square Error) en entornos aleatorios no estacionarios. En procesos estacionarios el algoritmo ofrece una solución idéntica a la caracterizada por un filtro Wiener.

2.2.1 Algoritmo LMS.

El algoritmo LMS es ampliamente usado en aplicaciones de filtrado adaptativo debido a su simplicidad computacional al no requerir funciones de correlación ni inversión de la matriz de autocorrelación. Presenta alta robustez a la estadística de las señales y además su comportamiento es sencillo. El algoritmo LMS se ha citado y estudiado en números trabajos de investigación y con el tiempo se han propuesto algunas modificaciones.

Como su propio nombre indica y a diferencia de los filtros fijos, las características de los filtros adaptativos cambian con el tiempo puesto que pretenden modelar o adaptarse iterativamente a la respuesta de un entorno variable.

El algoritmo LMS fue desarrollado por Bernard Windrow y Ted Hoff in 1960. Emplea el método de optimización por gradiente estocástico para ajustar los coeficientes del filtro y encontrar el mínimo del cuadrado de la señal error.

2.2.1.1 Principios básicos de funcionamiento

El algoritmo LMS básico es una implementación estocástica frente al algoritmo por gradiente descendiente o *Steepest Descent*, desconociendo por tanto los parámetros estadísticos de las señales, que procura minimizar la función error de coste a través del error cuadrático instantáneo como aproximación del error cuadrático medio:

(2. 5)

$$\xi(n) = E\{\, e^2(n) \,\} \approx\ e^2(n)$$

donde el operador E{·} denota el valor medio.

La ecuación del algoritmo LMS para la actualización periódica de los N coeficientes del filtro viene expresada en términos matriciales por:

(2. 6)

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) -\ \mu\nabla e^2(n)$$

siendo μ el factor de convergencia o paso de adaptación del algoritmo y ∇ el operador gradiente definido como vector columna:

(2. 7)

$$\nabla = \left[\frac{\partial}{\partial w_0}\ \ \frac{\partial}{\partial w_1}\ \ \cdots \frac{\partial}{\partial w_{N-1}}\right]^T$$

De esta forma podemos escribir el i-ésimo elemento del vector gradiente $\nabla e^2(n)$:

(2. 8)

$$\frac{\partial e^2(n)}{\partial w_i(n)} = 2 \cdot e(n) \cdot \frac{\partial e(n)}{\partial w_i(n)}$$

Sustituyendo (2. 2) en la última derivada a la parte derecha de la igualdad (2. 8) y siendo *d(n)* independiente de $w_i(n)$, obtenemos:

(2. 9)

$$\frac{\partial e^2(n)}{\partial w_i(n)} = -2 \cdot e(n) \cdot \frac{\partial y(n)}{\partial w_i(n)}$$

Sustituyendo *y(n)* de la ecuación (2. 1)

(2. 10)

$$\frac{\partial e^2(n)}{\partial w_i(n)} = -2 \cdot e(n) \cdot \frac{\partial (w_i(n) \cdot x(n-i))}{\partial w_i(n)} = -2 \cdot e(n) \cdot x(n-i)$$

Aplicando (2. 7) y (2. 3) sobre (2. 10) el gradiente de la función de coste queda:

(2. 11)

$$\nabla e^2(n) = = -2 \cdot \mathrm{e(n)} \cdot \mathbf{x}(\mathrm{n})$$

Finalmente, añadiendo el resultado (2. 11) en (2. 6) tenemos la ecuación de ajuste para los N coeficientes del filtro:

(2. 12)

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) +\ 2\mu \cdot \mathrm{e(n)} \cdot \mathbf{x}(\mathrm{n})$$

Podemos comprobar la sencillez del método viendo que los pesos del vector $\boldsymbol{w}(n+1)$, en el instante n, sólo dependen de tres entradas:

- El vector actual $\mathbf{x}(\mathrm{n})$ a la entrada del filtro y anteriores $\mathbf{x}(\mathrm{n}-1), \ldots, \mathbf{x}(1)$
- La muestra actual $\mathrm{d(n)}$ deseada y anteriores $\mathrm{d(n}-1), \ldots, \mathrm{d(1)}$
- El valor de $\boldsymbol{w}(n)$ actual tras modificaciones desde el vector inicial de pesos $\boldsymbol{w}(0)$, generalmente inicializado a cero.

Estas tres entradas son independientes entre sí para el cálculo de $\boldsymbol{w}(n+1)$ y del mismo modo también $\boldsymbol{x}(n+1)\ y\ d(n+1)$serán independientes de $\boldsymbol{w}(n+1)\ y\ e(n+1)$ en la siguiente iteración.

El procedimiento práctica para trabajar con el algoritmo LMS en un sistema adaptativo recursivo tras obtener la nueva muestra *x(n)* y la muestra objetivo *d(n)*:

1. Actualizar el vector de entrada $\boldsymbol{x}(n)$
2. Calcular la salida $y(n) = \boldsymbol{w}^T(n) \cdot \boldsymbol{x}(n)$
3. Estimar la señal $e(n) = d(n) - y(n)$
4. Actualizar el filtro $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + 2\mu \cdot \mathrm{e(n)} \cdot \mathbf{x(n)}$

Este algoritmo necesita únicamente 2N+1 productos por iteración, N para calcular la señal filtrada *y(n)*, una para obtener $2\mu \cdot \mathrm{e(n)}$ y otras N últimas para realizar el producto interno en (2. 12).

El factor μ es un parámetro constante menor a la unidad clave en el control del ajuste de los N coeficientes del filtro estableciéndose comúnmente un límite según la potencia y número de coeficientes:

(2. 13)

$$0 < \mu < \frac{1}{NP_x}$$

2.2.1.2 LMS normalizado

Como hemos visto, el algoritmo LMS depende directamente de la potencia de entrada.

El algoritmo LMS normalizado (NLMS) es una implementación especial del algoritmo LMS que tiene en cuenta la variación de los niveles de la señal a la entrada del filtro y emplea un factor de convergencia μ normalizado que resulta más estable amortiguando variaciones bruscas y con un buen ritmo de convergencia.

Partiendo de la ecuación principal recursiva del algoritmo LMS en (2. 12) pero considerando μ variable en el tiempo se presenta:

(2. 14)

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + 2\mu(\mathrm{n}) \cdot \mathrm{e(n)} \cdot \mathbf{x(n)}$$

Seleccionamos μ(n) para que el error a posteriori se minimice en magnitud:

(2. 15)

$$e^+(n) = d(n) - \boldsymbol{w}(n+1)^{\boldsymbol{T}} \cdot \mathbf{x(n)}$$

Sustituyendo (2. 14) en (2. 15) y considerando la definición del error *e(n)* en (2. 2):

$$e^+(n) = d(n) - \boldsymbol{x}(n)^{\boldsymbol{T}} \cdot \left(\boldsymbol{w}(n) + 2\mu(\mathrm{n})\mathrm{e(n)}\mathbf{x(n)}\right) = \mathrm{e(n)} \cdot \left(1 - 2\mu(\mathrm{n})\boldsymbol{x}(n)^{\boldsymbol{T}}\mathbf{x(n)}\right)$$

Minimizando a cero el coste del error a posteriori $\left(e^+(n)\right)^2$ con respecto a μ(n) forzando $e^+(n)$ a cero:

(2. 16)

$$1 - 2\mu(n)\boldsymbol{x}(n)^{\boldsymbol{T}}\boldsymbol{x}(n) = 0 \;\rightarrow\; \mu(n) = \frac{1}{2\boldsymbol{x}(n)^{\boldsymbol{T}}\boldsymbol{x}(n)}$$

Sustituyendo (2. 16) en (2. 14) obtenemos:

(2. 17)

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{1}{\boldsymbol{x}(n)^{\boldsymbol{T}}\boldsymbol{x}(n)} \cdot e(n) \cdot \boldsymbol{x}(n)$$

Luego, según (2. 16) el parámetro μ(n) será inversamente proporcional a la energía instantánea de las muestras de la señal a la entrada del filtro, manteniendo una fuerte dependencia con éstas al estar vinculadas en la misma iteración. Se da la conveniencia de seleccionar el factor de convergencia inversamente proporcional a la potencia en la entrada al filtro. Para evitar

divisiones entre cero se emplea una constante ψ pequeña y otra constante $\tilde{\mu}$ para controlar el paso de convergencia y el desajuste, quedando el algoritmo NLMS recursivo:

(2. 18)

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \frac{\tilde{\mu}}{\boldsymbol{x}(n)^{\boldsymbol{T}}\boldsymbol{x}(n) + \psi} \cdot e(n) \cdot \boldsymbol{x}(n)$$

2.2.2 Implementación por bloques.

Hay ciertas aplicaciones que requieren de un filtro adaptativo cuyo tamaño o longitud excede algunos cientos o miles de coeficientes haciendo que el sencillo algoritmo LMS se vuelva computacionalmente costoso de implementar. Por otro lado, en aplicaciones de sistemas en tiempo real en los que se utilizan tarjetas de sonido, los algoritmos que funcionan muestra a muestra como es el LMS descrito, resultan imposibles de implementar. Estos dispositivos trabajan con bloques de muestras y por tanto se necesitan algoritmos adaptativos que procesen un bloque completo de muestras.

Teniendo en cuenta todo lo anterior se propone el uso del algoritmo LMS por bloques. La implementación por bloques almacena tanto las muestras de entrada como las deseadas en vectores procesándolos para obtener un bloque de muestras a la salida. Ello implica conversiones serie-paralelo (S/P) y viceversa para generar el dato de salida como se muestra en la Figura 2.3.

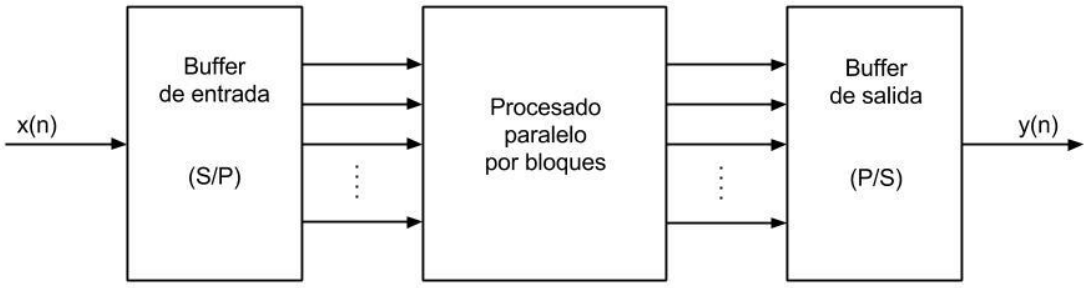


Figura 2. 3: Esquema sistema procesado por bloques

2.2.2.1 Algoritmo BLMS

Partiendo de la ecuación recursiva (2. 12) del LMS y las definiciones de la señal de entrada al filtro (2. 3) y los coeficientes del filtro (2. 4), el algoritmo LMS por bloques (BLMS) trabaja actualizando los pesos del filtro tras acumular B muestras de entrada en bloques cada iteración. El vector a la entrada del filtro, como en (2. 3), es:

(2. 19)

$$\boldsymbol{x}_n = [x(n)\ \ x(n-1)\ \ \ldots\ x(n-N+1)]^T$$

Definiendo *k* como el índice de iteración o de cada bloque, se puede expresar matricialmente ordenando cada entrada del bloque *k* en filas la matriz $\boldsymbol{X}_k\ BxN$:

(2. 20)

$$\boldsymbol{X}_k = [\boldsymbol{x}_{kB}\ \ \boldsymbol{x}_{kB+1}\ \ \ldots\ \boldsymbol{x}_{kB+B-1}]^T$$

Y el resto de vectores:

(2. 21)

$$\boldsymbol{d}_k = [d(kB)\ \ d(kB+1)\ \ \ldots\ d(kB+B-1)]^T$$

(2. 22)

$$\boldsymbol{y}_k = [y(kB)\ \ y(kB+1)\ \ \ldots\ y(kB+B-1)]^T = \boldsymbol{X}_k \cdot \boldsymbol{w}_k$$

(2. 23)

$$\boldsymbol{e}_k = [e(kB)\ \ e(kB+1)\ \ \ldots\ e(kB+B-1)]^T$$

Denotando en cada iteración el bloque de la estimación del error:

(2. 24)

$$\boldsymbol{e}_k = \boldsymbol{d}_k - \boldsymbol{y}_k = \boldsymbol{d_k} - \ \boldsymbol{X}_k \cdot \boldsymbol{w}_k$$

El método del gradiente utilizado en (2. 11) se calcula en cada iteración para el bloque *k*, quedando la ecuación de adaptación:

(2. 25)

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \ 2\mu_B \cdot \frac{\sum_{i=0}^{B-1} e(kB+i) \cdot \boldsymbol{x_{k+i}}}{B} = \ \boldsymbol{w}(k) + \ 2\frac{\mu_B}{B} \cdot \boldsymbol{X}_k^T \cdot \boldsymbol{e}_k$$

Donde μ_B es el factor de convergencia que se relaciona con el del LMS (μ) del siguiente modo:

(2. 26)

$$\mu = \frac{\mu_B}{B}$$

Este algoritmo requiere *(2·N+1)·B* productos por cada bloque de *B* muestras. *(N+1)·B* en la ecuación de adaptación y *N*·B en el filtrado, que es lo mismo que ejecutar con *B* muestras las *2N+1* multiplicaciones del LMS.

Por último indicar que la normalización del factor de adaptación μ será distinta cada bloque de entrada y para cada uno de los vectores de entrada al filtro de la matriz $\boldsymbol{X}_k$:

(2. 27)

$$\boldsymbol{P}_{x_k} = diag\left(\boldsymbol{X}_k \cdot \boldsymbol{X}_k{}^{\boldsymbol{T}}\right) = \ [P_{x_k}{}^0, P_{x_k}{}^1, \cdots, P_{x_k}{}^{B-1}]^T$$

(2. 28)

$$\boldsymbol{\mu}_k = \frac{\mu}{\boldsymbol{P}_{x_k}} = [\mu_k^0, \mu_k^1, \cdots, \mu_k^{B-1}]$$

Quedando:

(2. 29)

$$\boldsymbol{w}(k+1) = \ \boldsymbol{w}(k) + \ 2 \cdot \boldsymbol{X}_k^T \cdot (\boldsymbol{\mu}_k{}^T \circ \boldsymbol{e}_k)$$

Donde ∘ denota multiplicación elemento a elemento.

2.2.2.2 Algoritmo FBLMS

Existe una implementación especial del algoritmo LMS por bloques que aporta rapidez computacional explotando las características del sistema en el dominio frecuencial denominado algoritmo *fast* BLMS (FBLMS) o LMS por bloques en la frecuencia. Así la convolución lineal en el tiempo puede llevarse a cabo más eficientemente mediante el procesado digital en la frecuencia con las propiedades de la Transformada de Fourier Discreta (DFT).

Definimos el vector columna $\widetilde{\boldsymbol{x}}_k$de longitud *N'=N+B-1*, siendo N el número de coeficientes del filtro y B el tamaño de bloque:

(2. 30)

$$\widetilde{\boldsymbol{x}}_k = [x(kB-N+1), x(kB-N+2), \cdots, x(kB+B-1)]^T$$

Almacena B muestras nuevas cada iteración. Y $\widetilde{\boldsymbol{w}}_k$de N' elementos:

(2. 31)

$$\widetilde{\boldsymbol{w}}_k = \begin{bmatrix} \boldsymbol{w}_k \\ \boldsymbol{0} \end{bmatrix}$$

Donde $\boldsymbol{w}_k = [w_0(k)\ w_1(k)\ \ldots\ w_{N-1}(k)]^T$ son las componentes del filtro y **0** hace referencia a un vector columna de *B-1* ceros. El índice *k* indica como la actualización de los pesos varía bloque a bloque.

A la salida real del filtro tendremos, siendo matemáticamente el resultado de la convolución lineal (2. 1), considerando en la siguiente ecuación a $\widetilde{\boldsymbol{x}}_k$ periódica para recorrerla desde el final:

(2. 32)

$$\boldsymbol{y}_k^c = \sum_{m=0}^{m=N'-1} \widetilde{w}_m(k) \cdot \widetilde{\boldsymbol{x}}_k * \delta[n-m] = \boldsymbol{X}_k^c \cdot \widetilde{\boldsymbol{w}}_k$$

La convolución circular de $\widetilde{\boldsymbol{w}}_k$ y $\widetilde{\boldsymbol{x}}_k$ se muestra matricialmente en (2. 33).

(2. 33)

$$\begin{bmatrix} * \\ * \\ \vdots \\ * \\ y(kB) \\ y(kB+1) \\ \vdots \\ y(kB+B-1) \end{bmatrix} = \begin{bmatrix} x(kB-N+1) & x(kB+B-1) & x(kB+B-2) & \cdots & x(kB-N+2) \\ x(kB-N+2) & x(kB-N+1) & x(kB+B-1) & \cdots & x(kB-N+3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x(kB-1) & x(kB-2) & x(kB-3) & \ddots & x(kB) \\ x(kB) & x(kB-1) & x(kB-2) & \cdots & x(kB+1) \\ x(kB+1) & x(kB) & x(kB-1) & \cdots & x(kB+2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x(kB+B-1) & x(kB+B-2) & x(kB+B-3) & \cdots & x(kB-N+1) \end{bmatrix} \begin{bmatrix} w_0(k) \\ w_1(k) \\ \vdots \\ w_{N-2}(k) \\ w_{N-1}(k) \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Se observa que sólo las *B* últimas muestras resultado de la convolución lineal coinciden con los elementos correspondientes a (2. 22), mientras que los elementos representados por el asterisco (*) no tienen ninguna utilidad. Por ello suele elegirse un valor $N' = 2 * B$ [1]

Así la convolución lineal generará el resultado (2. 22) de tal modo:

(2. 34)

$$\widetilde{\boldsymbol{y}}_k = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{y}_k \end{bmatrix} = \boldsymbol{P}_{\boldsymbol{0},\boldsymbol{B}} \cdot \boldsymbol{y}_k^c = \boldsymbol{P}_{\boldsymbol{0},\boldsymbol{B}} \cdot \boldsymbol{X}_k^c \cdot \widetilde{\boldsymbol{w}}_k$$

Donde $\boldsymbol{P}_{\boldsymbol{0},\boldsymbol{B}}$ es una matriz *N'xN'* que enventana el resultado útil, $\boldsymbol{y}_k$, a través de una matriz de identidad $\boldsymbol{I}_{\boldsymbol{B}}$ *BxB* que multiplica por cero los elementos denotados por asteriscos (*) en (2. 33). Se intuye fácilmente que $\boldsymbol{X}_k^c$ es la matriz circular de $\widetilde{\boldsymbol{x}}_k$.

La convolución circular anterior puede obtenerse más eficientemente transformando ambos vectores a su equivalente dominio frecuencial, multiplicando elemento por elemento y deshaciendo la transformación de nuevo al dominio temporal utilizando la Transformada Rápida de Fourier (FFT) y su inversa (IFFT). Suele elegirse un tamaño de bloque *B* igual al tamaño del filtro *N* por comodidad para generar la matriz circular, al ser útiles sólo los últimos *B* y para que coincida directamente con la transformada de Fourier del vector $\widetilde{\boldsymbol{x}}_k$.

Desarrollando en el dominio frecuencial y definiendo la FFT del vector que contiene los pesos del filtro:

(2. 35)

$$\boldsymbol{w}_k^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\{\widetilde{\boldsymbol{w}}_k\}$$

Y de los bloques almacenados en el buffer de entrada:

(2. 36)

$$\boldsymbol{\chi}_k^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\{\boldsymbol{X}_k^c\}\boldsymbol{\mathcal{F}}^{-1} = \boldsymbol{\mathcal{F}}\{\widetilde{\boldsymbol{x}}_k\}$$ [1]

[1] Para el tamaño del bloque igual a la longitud del filtro, N=B. Se asume que el vector de ceros **0** también es de B elementos y no B-1. Así para una matriz cuadrada circular $\boldsymbol{A}^{\boldsymbol{c}}$ de tamaño *MxM*, su matriz frecuencial $\boldsymbol{A}_k^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\{\boldsymbol{A}^{\boldsymbol{c}}\}\boldsymbol{\mathcal{F}}^{-1} = diag[\boldsymbol{a}^{\mathcal{F}}]$ es diagonal caracterizada por la primera columna $\boldsymbol{a}$ de $\boldsymbol{A}^{\boldsymbol{c}}$.

Obtenemos que la salida es el producto elemento a elemento en la frecuencia de las muestras de entrada al filtro por sus coeficientes seguido de la Transformada Rápida de Fourier Inversa (IFFT) enventanando finalmente el resultado como en (2. 34):

(2. 37)

$$\widetilde{\boldsymbol{y}}_k = \boldsymbol{P}_{\boldsymbol{0},\boldsymbol{B}} \cdot \boldsymbol{\mathcal{F}}^{-1}(\,\boldsymbol{\mathcal{F}}\{\boldsymbol{X}_k^c\}\boldsymbol{\mathcal{F}}^{-1}) \cdot \boldsymbol{\mathcal{F}}\{\widetilde{\boldsymbol{w}}_k\} = \boldsymbol{P}_{\boldsymbol{0},\boldsymbol{B}} \cdot \boldsymbol{\mathcal{F}}^{-1}\{\boldsymbol{\chi}_k^{\mathcal{F}} \cdot \boldsymbol{w}_k^{\mathcal{F}}\}$$

El vector de las muestras deseadas se define como:

(2. 38)

$$\widetilde{\boldsymbol{d}}_{\boldsymbol{k}} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{d}_{\boldsymbol{k}} \end{bmatrix}$$

Donde $\boldsymbol{d_k}$ se define en (2. 21) y $\boldsymbol{0}$ un vector de N-1 ceros. Del mismo modo el vector error extendido contendrá $\boldsymbol{e}_k$ (2. 23):

(2. 39)

$$\tilde{\boldsymbol{e}}_k = \widetilde{\boldsymbol{d}}_k - \widetilde{\boldsymbol{y}}_k = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{e}_k \end{bmatrix}$$

Reemplazando $\boldsymbol{w}_k$y $\boldsymbol{e}_k$ en la ecuación del algoritmo BLMS (2. 25) por sus homólogos extendidos (2. 31) y (2. 39) la fórmula de adaptación se modifica en esta versión FBLMS como sigue:

(2. 40)

$$\widetilde{\boldsymbol{w}}_{k+1} = \widetilde{\boldsymbol{w}}_k + 2\mu \boldsymbol{P}_{\boldsymbol{N},\boldsymbol{0}} \cdot \boldsymbol{X}_k^{c\,T} \cdot \tilde{\boldsymbol{e}}_k$$

Donde $\boldsymbol{X}_k^c$ es la matriz circular del bloque *k* a la entrada, $\mu = \frac{\mu_B}{B}$ y $\boldsymbol{P}_{\boldsymbol{N},\boldsymbol{0}}$ una matriz de enventanado *N'xN'* cuya finalidad es asegurar que los B-1 elementos inferiores del vector de pesos extendido se mantengan a cero cada iteración.

(2. 41)

$$\boldsymbol{P}_{\boldsymbol{N},\boldsymbol{0}} = \begin{bmatrix} I_N & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}$$

Transformando al dominio de la frecuencia todos los parámetros involucrados en (2. 40) con las transformadas de Fourier definidas en (2. 35) y (2. 36):

(2. 42)

$$\boldsymbol{w}_{k+1}^{\mathcal{F}} = \boldsymbol{w}_k^{\mathcal{F}} + 2\mu \cdot (\,\boldsymbol{\mathcal{F}}\{\boldsymbol{P}_{\boldsymbol{N},\boldsymbol{0}}\}\boldsymbol{\mathcal{F}}^{-1}) \cdot (\,\boldsymbol{\mathcal{F}}\{\boldsymbol{X}_k^{c\,T}\}\boldsymbol{\mathcal{F}}^{-1}) \cdot \boldsymbol{\mathcal{F}}\{\tilde{\boldsymbol{e}}_k\} = \boldsymbol{w}_k^{\mathcal{F}} + 2\mu \cdot \mathcal{P}_{N,0} \cdot \boldsymbol{\chi}_k^{\mathcal{F}^*} \cdot \boldsymbol{e}_k^{\mathcal{F}}$$

Donde $\boldsymbol{e}_k^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\{\tilde{\boldsymbol{e}}_k\}$ y $\boldsymbol{\chi}_k^{\mathcal{F}^*}$ el conjugado de $\boldsymbol{\chi}_k^{\mathcal{F}}$.

En resumen, el procedimiento para trabajar con el algoritmo FBLMS en un sistema adaptativo recursivo tras obtener las *B* nuevas muestras *x(n)* y las muestras objetivo *d(n)* es:

1. Actualizar el vector de entrada $\widetilde{\boldsymbol{x}}_k$y generar su $\boldsymbol{\chi}_k^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\{\boldsymbol{X}_k^c\}\boldsymbol{\mathcal{F}}^{-1}$
2. Calcular la salida $\widetilde{\boldsymbol{y}}_k = \boldsymbol{P}_{\boldsymbol{0},\boldsymbol{B}} \cdot \boldsymbol{\mathcal{F}}^{-1}\{\boldsymbol{\chi}_k^{\mathcal{F}} \cdot \boldsymbol{w}_k^{\mathcal{F}}\}$
3. Estimar la señal $\tilde{\boldsymbol{e}}_k = \widetilde{\boldsymbol{d}}_k - \widetilde{\boldsymbol{y}}_k$ y generar su $\boldsymbol{e}_k^{\mathcal{F}}$
4. Actualizar el filtro $\boldsymbol{w}_{k+1}^{\mathcal{F}} = \boldsymbol{w}_k^{\mathcal{F}} + 2\mu \cdot \mathcal{P}_{N,0} \cdot \boldsymbol{\chi}_k^{\mathcal{F}^*} \cdot \boldsymbol{e}_k^{\mathcal{F}}$

El proceso tiene 3 pasos claros: filtrado, estimación del error y actualización de los coeficientes con el matiz de que se trabaja en el dominio de la frecuencia como se puede ver en el siguiente diagrama de bloques:

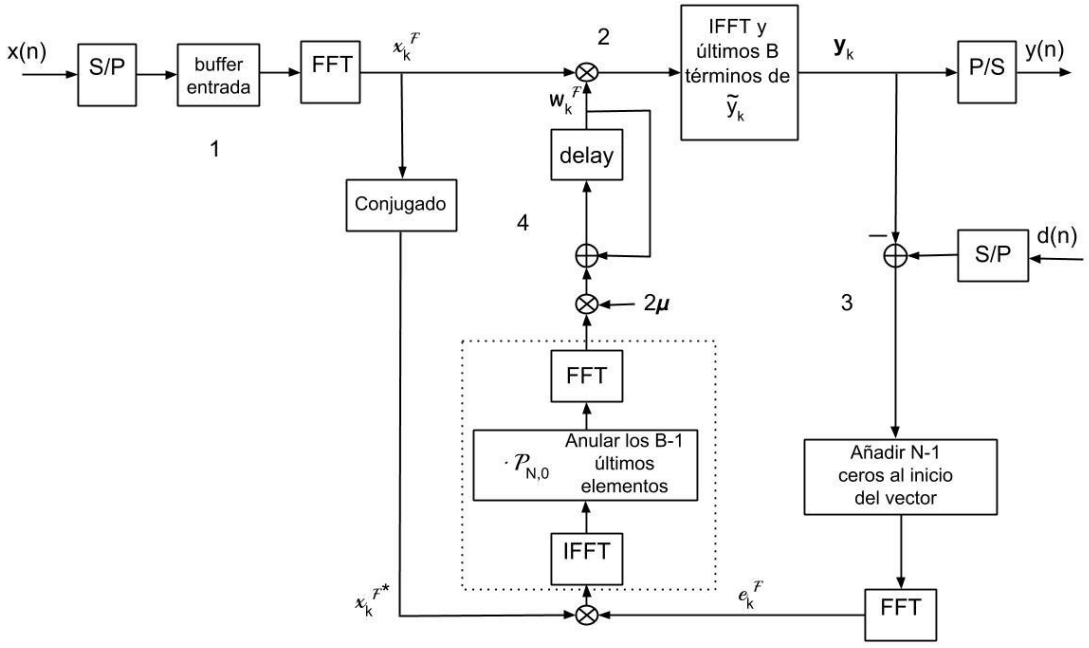


Figura 2. 4: Esquema FBLMS

A la entrada las muestras son almacenadas en el buffer de entrada cuya salida es (ident) $\widetilde{\boldsymbol{x}}_k$ con B muestras nuevas y N-1 muestras de bloques anteriores, método conocido como *overlap-save*. El vector $\widetilde{\boldsymbol{x}}_k$ es transformado al dominio frecuencial multiplicándose muestra a muestra con el vector de coeficientes $\boldsymbol{w}_k^{\mathcal{F}}$. El resultado se reconvierte al dominio temporal utilizando la IFFT. Las últimas B muestras son el resultado correspondiente al bloque actual de salida $\boldsymbol{y}_k$, utilizado para estimar el error $\boldsymbol{e}_k$ con $\boldsymbol{d}_k$. Se añaden N-1 ceros para extender el vector y obtener $\tilde{\boldsymbol{e}}_k$ (2. 39) al cual se le aplica una FFT para calcular la actualización, junto al complejo conjugado de las muestras a la entrada en la frecuencia $\boldsymbol{\chi}_k^{\mathcal{F}^*}$, de los coeficientes según la ecuación (2. 42), multiplicando previamente por $\mathcal{P}_{N,0}$para asegurar que los B-1 últimos coeficientes en el tiempo son cero (2. 31).

Este algoritmo tiene un rendimiento similar al BLMS en cuanto a velocidad de convergencia, sin embargo es más eficiente al necesitar menos operaciones trabajando en la frecuencia Además el desajuste del error en la versión normalizada disminuye según la constante de paso inicial μ_0 que se utilice y la relación entre número de coeficientes N y tamaño de bloque B:

(2. 43)

$$\mathcal{M}_{FBNLMS} \approx \mu_0 \frac{N}{N'}$$

La normalización en potencia del factor μ_k se realiza para cada bloque *k* de entrada queda de manera simplificada:

(2. 44)

$$\bar{P}_{x_k} = \sum_{i=0}^{B-1} \boldsymbol{P}_{x_k}$$

(2. 45)

$$\mu_k = \frac{\mu}{\bar{P}_{x_k}}$$

Así la ecuación de adaptación en forma vectorial a partir de (2. 42), siendo $\boldsymbol{\mathcal{F}}\{\widetilde{\boldsymbol{x}}_k\} = \boldsymbol{x}_k^{\mathcal{F}}$, queda:

(2. 46)

$$\boldsymbol{w}_{k+1}^{\mathcal{F}} = \boldsymbol{w}_k^{\mathcal{F}} + 2\mu_k \cdot \boldsymbol{x}_k^{\mathcal{F}^*} \circ \boldsymbol{e}_k^{\mathcal{F}}$$

Se debe reemplazar o añadir los ceros adecuadamente según (2. 31), (2. 34) y (2. 39) para ser rigurosos con el cálculo del paso. Destacar que la primera mitad de los vectores frecuenciales contiene toda la información necesaria al ser vectores complejos simétricos conjugados.

Por último señalar que esta versión es buena para valores de bloque B más pequeños que el número de pesos del filtro N aumentando el coste computacional conforme N≈B. Esto genera retardos de tiempo por procesado que en algunas aplicaciones muy restrictivas como en sistemas en tiempo real puede ser intolerable.

2.2.2.3 Algoritmo PFBLMS

El algoritmo LMS por bloques funcionando en la frecuencia y particionado (PFBLMS) surge como alternativa al algoritmo FBLMS para filtros de longitud N mayores al tamaño de bloque B bajo la idea de dividir la convolución lineal en un procesado matricial basado en suma de convoluciones más pequeñas al particionar el filtro en P tramas de tamaño M.

Asumiendo que el vector $\boldsymbol{w}_k$ es de N coeficientes, siendo P y M números naturales, se divide en P secuencias $\boldsymbol{w}_{k,l}$ de M componentes para *l=0,1…, P-1*. Si en cada iteración existe B nuevas muestras la salida combinando las P convoluciones es:

(2. 47)

$$\boldsymbol{y}_k = \left[y_k^0, y_k^1, \dots, y_k^{B-1}\right]^T = \sum_{l=0}^{P-1} \boldsymbol{y}_{k,l} \ , \ \ para \ \ y_k^j = \sum_{l=0}^{P-1} y_{k,l}^j$$

Donde cada subtrama *l* viene dada por la suma de M contribuciones:

(2. 48)

$$\boldsymbol{y}_{k,l} = \boldsymbol{X}_{k,l} \cdot \boldsymbol{w}_{k,l} = [y_{k,l}^0, y_{k,l}^1, \dots, y_{k,l}^{B-1}]^T$$

$$para \quad y_{k,l}^j = \sum_{i=0}^{M-1} w_{i+lM}(k) \cdot x(kB - lM - i + j) = \boldsymbol{w}_{k,l}{}^T \cdot \boldsymbol{x}_{k,l}^j$$

Definiendo $\boldsymbol{X}_{k,l} = [\boldsymbol{x}_{k,l}^0, \boldsymbol{x}_{k,l}^1, \dots, \boldsymbol{x}_{k,l}^{B-1}]^{\boldsymbol{T}}$ y para *j=0,1,…B-1*:

(2. 49)

$$\boldsymbol{x}_{k,l}^j = [x(kB - lM + j) \ \ x(kB - lM - 1 + j) \ \ \dots \ x(kB - lM - M + 1 + j)]^T$$

De manera gráfica:

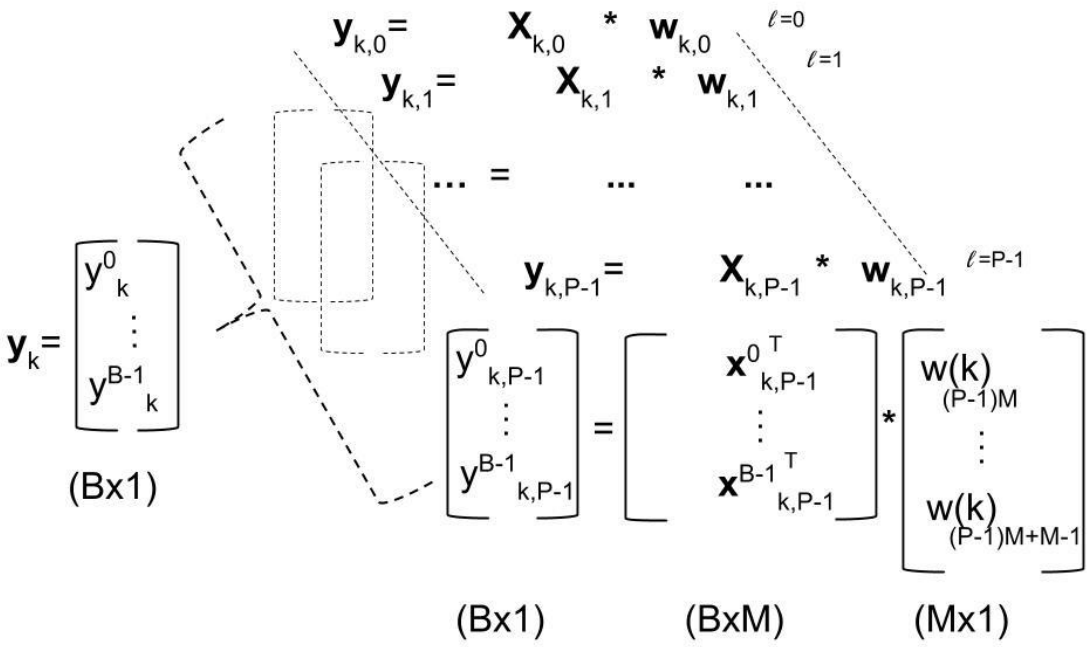


Figura 2. 5: Convolución circular particionada

Si $M=p\cdot B$, para p un número entero, tendríamos como vector de entrada:

$$\boldsymbol{x}_{k,l} = [x\big((k-lp)B\big)\ x\big((k-lp)+1\big)\ \dots\ x((k-lp)B+M-1)]^T \quad \textbf{(2. 50)}$$

Su transformada en frecuencia vendrá dada, como se vio anteriormente, por un vector de entrada extendido de tamaño $M+B-1= B\cdot(p+1)-1$:

$$\widetilde{\boldsymbol{x}}_{k,l} = [x\big((k-lp)B-M\big)\ x\big((k-lp)B-M+1\big)\ \dots\ x((k-lp)B+B-1)]^T \quad \textbf{(2. 51)}$$

Denotamos la FFT,

$$\boldsymbol{x}^{\mathcal{F}}_{k,l} = \boldsymbol{\mathcal{F}}\{\widetilde{\boldsymbol{x}}_{k,l}\} = \boldsymbol{x}^{\mathcal{F}}_{k-lp,0} \quad \textbf{(2. 52)}$$

Se puede obtener a partir de la primera partición l=0 y considerando el bloque k-lp. El i-ésimo elemento de $\boldsymbol{x}^{\mathcal{F}}_{k,l}$:

$$x_{i}{}^{\mathcal{F}}_{k,l} = \sum_{m=0}^{M+B-1} x\big((k-lp)B-M+m\big)\cdot e^{-j\frac{2\pi im}{M+B}} = x_{i}{}^{\mathcal{F}}_{k+lp,0} \quad \textbf{(2. 53)}$$

De manera similar a (2. 31) y (2. 34) y manteniendo la coherencia con (2. 47) y (2. 48):

$$\boldsymbol{w}^{\mathcal{F}}_{k,l} = \boldsymbol{\mathcal{F}}[w_{lM}(k), w_{lM+1}(k)\ \dots\ w_{lM+M-1}(k), \overbrace{0, 0\ \dots, 0}^{B}]^T \quad \textbf{(2. 54)}$$

$$\boldsymbol{y}_{k,l} = \boldsymbol{\mathcal{F}}^{-1}\big(\boldsymbol{w}^{\mathcal{F}}_{k,l} \circ \boldsymbol{x}^{\mathcal{F}}_{k-lp,0}\big)\ ^{2} \quad \textbf{(2. 55)}$$

Con estos resultados la estimación del error es idéntica a (2. 39):

[2] $\boldsymbol{y}_{k,l}$ corresponde a los últimos B elementos de la IFFT al ser el resultado útil según se vio en (2. 34)

(2. 56)

$$\boldsymbol{e}_k^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\{\tilde{\boldsymbol{e}}_k\} = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{e}_k \end{bmatrix} \quad , \qquad para \quad \boldsymbol{e}_k = \boldsymbol{d}_k - \boldsymbol{y}_k$$

Siendo $\boldsymbol{d}_k = \left[d_k^0, d_k^1, \dots, d_k^{B-1}\right]^T = [d(kB)\ \ x(kB+1)\ \dots\ x(kB+B-1)]^T$

Usando todo lo anterior la ecuación de adaptación del algoritmo PFBLMS queda:

(2. 57)

$$\boldsymbol{w}_{k+1,l}^{\mathcal{F}} = \boldsymbol{w}_{k,l}^{\mathcal{F}} + 2\boldsymbol{\mu}_k \cdot \boldsymbol{x}_{k-pl,0}^{\mathcal{F}\ *} \cdot \boldsymbol{e}_k^{\mathcal{F}}$$

Finalmente, para ser rigurosos con el comportamiento de los coeficientes frecuenciales en la siguiente iteración se debe asegurar que se emplean los M elementos del filtro y el vector correspondiente de ceros **0**:

(2. 58)

$$\boldsymbol{\phi}_{k+1,l}^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\left\{\begin{bmatrix} \boldsymbol{\mathcal{F}^{-1}}\left\{\boldsymbol{x}_{k-pl,0}^{\mathcal{F}\ *} \cdot \boldsymbol{e}_k^{\mathcal{F}}\right\}^{3} \\ \mathbf{0} \end{bmatrix}\right\}$$

Donde se ha utilizado μ_k para normalizar el paso de actualización tal y como se vio en **¡Error! o se encuentra el origen de la referencia.**. Puede sustituirse por μ constante. Destacar que a partir del vector a la entrada $\widetilde{\boldsymbol{x}}_{k,0}$, el cálculo para cada partición *l* en frecuencia puede realizarse utilizando *p* retardos de B muestras del vector de entrada $\boldsymbol{x}_{k,0}^{\mathcal{F}}$ como se ha indicado en (2. 52).

En resumen el algoritmo precisa de los siguientes pasos:

1. Actualizar el nuevo bloque de B muestras en $\widetilde{\boldsymbol{x}}_{k,l}$ y generar $\boldsymbol{x}_{k,0}^{\mathcal{F}}$
2. Obtener los últimos B elementos de la salida $\boldsymbol{y}_k = \sum_{l=0}^{P-1} \boldsymbol{\mathcal{F}^{-1}}\left(\boldsymbol{w}_{k,l}^{\mathcal{F}} \circ \boldsymbol{x}_{k-lp,0}^{\mathcal{F}}\right)$
3. Estimar el error $\boldsymbol{e}_k = \boldsymbol{d}_k - \boldsymbol{y}_k$ y transformarlo a la frecuencia $\boldsymbol{e}_k^{\mathcal{F}}$
4. Restringir el paso $\boldsymbol{\phi}_{k+1,l}^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\left\{\begin{bmatrix} \boldsymbol{\mathcal{F}^{-1}}\left\{\left(\boldsymbol{x}_{k-pl,0}^{\mathcal{F}\ *} \cdot \boldsymbol{e}_k^{\mathcal{F}}\right)\right\} \\ \mathbf{0} \end{bmatrix}\right\}$ seleccionando los M primeros elementos de la IFFT y el vector **0** de B ceros.
5. Actualizar el filtro en frecuencia $\boldsymbol{w}_{k+1,l}^{\mathcal{F}} = \boldsymbol{w}_{k,l}^{\mathcal{F}} + 2\boldsymbol{\mu}_k \cdot \boldsymbol{\phi}_{k+1,l}^{\mathcal{F}}$

Esquemáticamente, marcado en azul caminos por vectores en el dominio de la frecuencia, el diagrama de bloques de la actualización:

[3] Se emplean las M primeras componentes de la Transformada Inversa de Fourier tal y como se precisa en (2. 54). Este paso puede eliminarse en procesos más laxos.

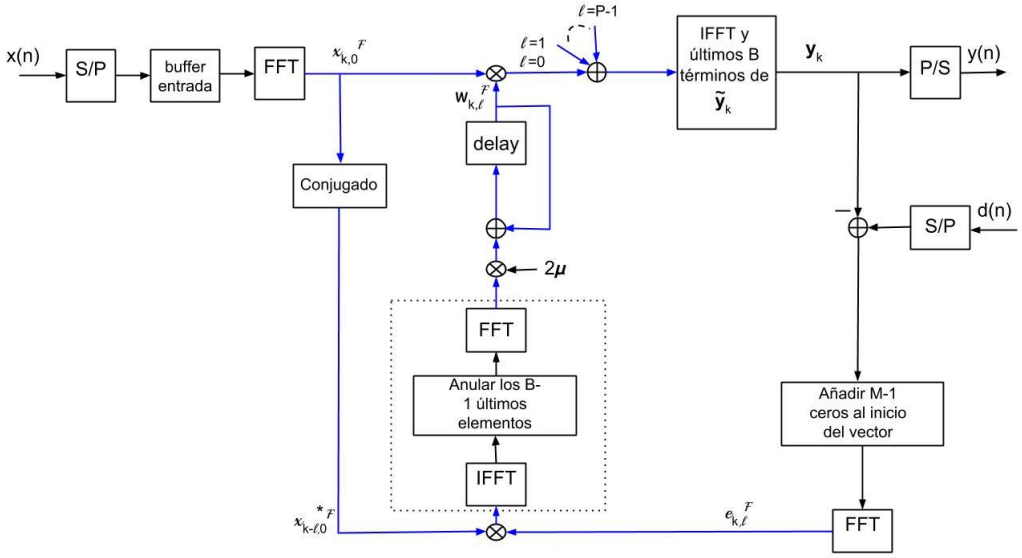


Figura 2. 6: Esquema PFBLMS por partición l

Con este algoritmo se logra implementar un procesado paralelo de P filtros transversales retardados entre sí M muestras, esto *p* retardos de B muestras entre filtros:

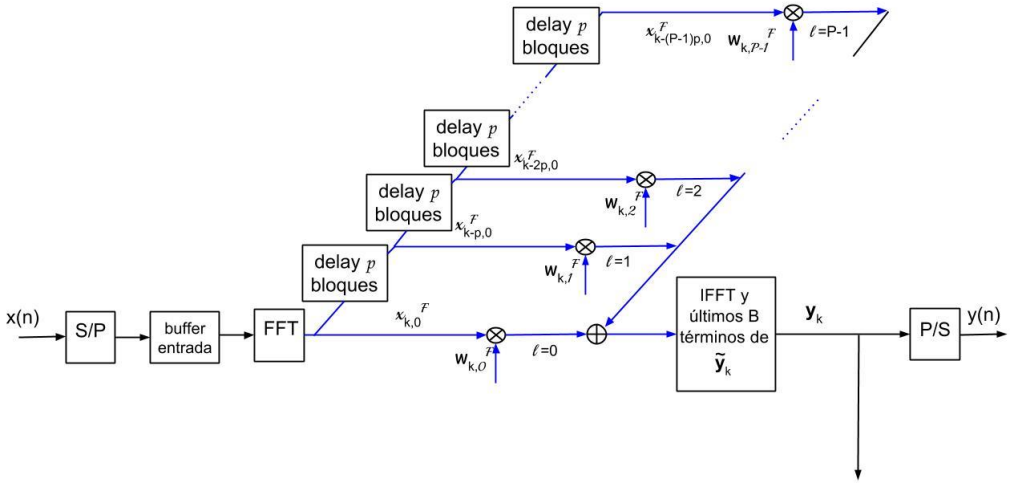


Figura 2. 7: Contribuciones de las l particiones en el esquema PFBLMS

Es común tomar valores en que el tamaño del bloque y las particiones del filtro son iguales, esto es *M=B*, luego *p*=1 facilitando las transformadas en frecuencia con vector de tamaño *2·B*.

Por último hacer referencia al desajuste del error con esta implementación. Como se puede comprobar en la ecuación inferior el desajuste empeora un factor P con respecto al algoritmo FBLMS:

(2. 59)

$$\mathcal{M}_{PFBLMS} = P \cdot \mathcal{M}_{FBLMS}$$

E igualmente en el caso de normalizar por la potencia:

(2. 60)

$$\mathcal{M}_{FBNLMS} = \mu_0 \mathrm{P} \frac{B}{M+B}$$

Mejora con respecto al no normalizado con el precio de pagar en mayor coste computacional.

2.3 Filtrado-x

2.3.1 Algoritmo FxLMS

En determinadas ocasiones el filtrado adaptativo necesita modificarse ya que la señal de error a la que accede el filtro está filtrada. Esto obliga a emplear una estimación de ese filtro para filtrar la señal de referencia del algoritmo y asegurar que éste converja. En las aplicaciones de control activo de ruido y ecualización resulta imprescindible considerar la estructura de filtrado-x en los algoritmos adaptativos. El algoritmo LMS con esta estructura se denomina FxLMS.

2.3.1.1 Estructuras de filtrado para ecualización.

Si se desea caracterizar o identificar la respuesta $(\hat{h})$ de un sistema (h) se emplea un esquema paralelo del LMS como se muestra a continuación [8]:

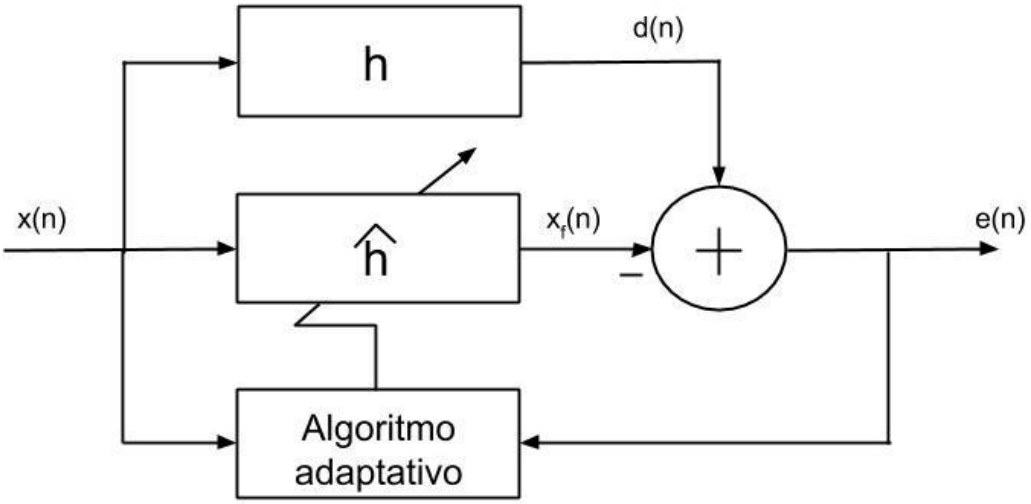


Figura 2. 8: Esquema clásico de identificación adaptativa

Otra utilidad puede ser conocer o hallar la respuesta inversa del sistema en cuestión para su ecualización, diseñando una estructura en serie como el de la figura de abajo:

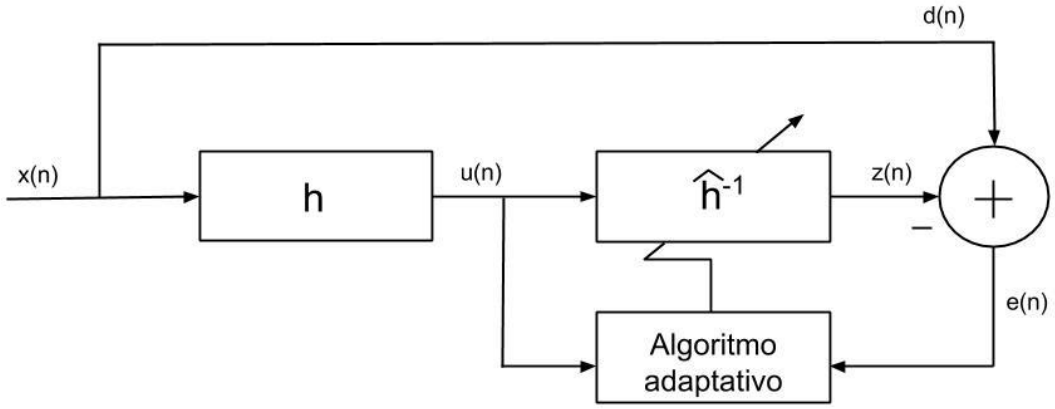


Figura 2. 9: Esquema clásico de ecualización adaptativa en recepción

Sin embargo, el objetivo de la aplicación acústica es hacerle llegar a un oyente la señal ecualizada y no la salida del canal acústico. Además la respuesta inversa obtenida puede ser

modificada por elementos que influyen en todo el conjunto como puede ser en un sistema acústico los equipos electro-acústicos tales como altavoces y micrófonos. Para solventar éstos problemas se tiende a un sistema en paralelo en el que se hace uso de una respuesta estimada del sistema analizado ($\hat{h}$) con una estructura tal que así:

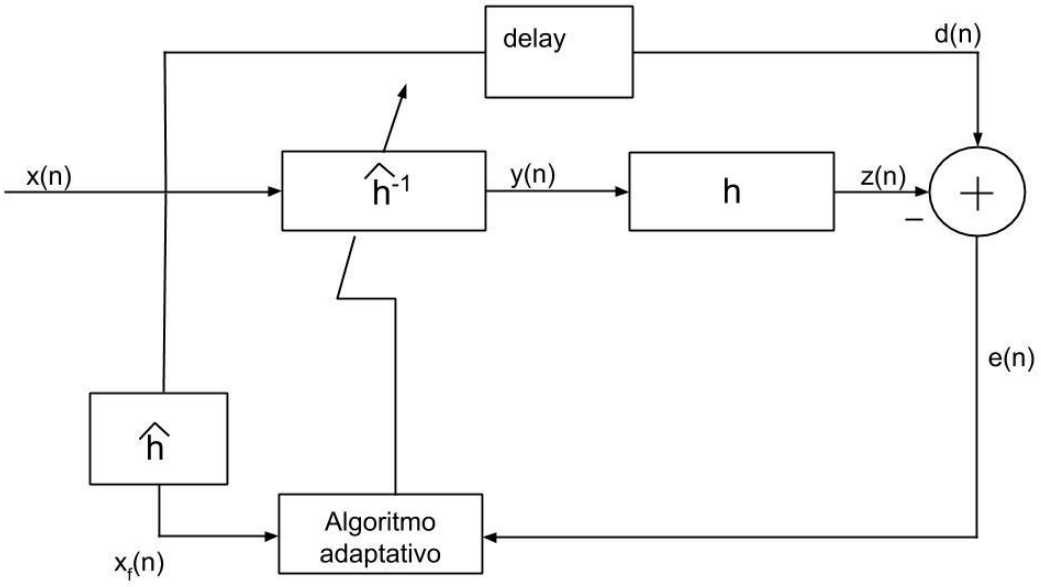


Figura 2. 10: Sistema adaptativo de ecualización en transmisión

Como se observa esta estructura se emplea un retardo de *delay* muestras para sincronizar las entradas al algoritmo adaptativo con su salida ya que de lo contrario la actualización vendría dada por un vector de error ***e*** estimado incorrecto. El retardo del procesado por filtrado-x se compensa en la entrada de la fuente.

El algoritmo adaptativo funcionaría idénticamente a lo comentado en apartados anteriores. Por ejemplo, para la versión de ecualización FxNMLS [5] y siguiendo los bloques de la Figura 2. 10 denotamos la salida del filtro ecualizador *y(n)* como:

(2. 61)

$$y(n) = \widehat{\boldsymbol{h}}^{-1}(n-1)^T \cdot \boldsymbol{x}_N(n)$$

Donde $\widehat{\boldsymbol{h}}^{-1}$ es el filtro inverso a $\boldsymbol{h}$ de N coeficientes y $\boldsymbol{x}_N$ incluye las N últimas muestras de la señal a la entrada *x(n)*. La señal *y(n)* se emite por el altavoz atravesando el canal $\boldsymbol{h}$. El micrófono captura la señal *m(n)* con la que junto a la deseada *d(n)* se estima el error.

(2. 62)

$$m(n) = \boldsymbol{h} * y(n)$$

(2. 63)

$$e(n) = d(n) - m(n)$$

Se ha comentado la aparición de retardos en el proceso, al que se deben añadir otros retardos por efectos eléctricos, mecánicos y de los equipos como se detalla más adelante en los resultados de la implementación en tiempo real, hace que la señal deseada sea la de entrada retardada τ muestras:

(2. 64)

$$d(n) = x(n-\tau)$$

Por último resta la actualización del filtro $\widehat{\boldsymbol{h}}^{-1}$:

(2. 65)

$$\widehat{\boldsymbol{h}}^{-1}(n) = \widehat{\boldsymbol{h}}^{-1}(n-1) + \bar{\mu} \cdot e(n) \cdot \boldsymbol{x}_f(n)$$

Donde $\boldsymbol{x_f}(n)$ contiene las últimas N muestras de la señal de entrada $x(n)$ filtrada por $\widehat{\boldsymbol{h}}$. El factor de convergencia es:

(2. 66)

$$\bar{\mu} = \frac{\mu}{\psi + \left\|\boldsymbol{x_f}\right\|^2}$$

2.3.2 Filtrado-x por bloques en el dominio de la frecuencia (FxFBLMS)

En sistemas complejos y en tiempo real no resulta práctico basar el algoritmo en un sistema que procesa muestra a muestra y más aún si se quiere aprovechar la ventaja de implementar el prototipo en el dominio de la frecuencia. Por ello deben usarse técnicas de procesado por bloques como algunos como FBLMS o su versión modificada PFBLMS, que se analizaron en el punto anterior.

2.3.2.1 Estructura FxFBLMS

Manteniendo la idea del FxLMS y del algoritmo FBLMS e incorporando los pasos relativos al procesado por bloques en la frecuencia como es los conversores Serie-Paralelo (S/P) y Paralelo-Serie (P/S), los buffers de almacenamiento, los vectores extendidos y las transformadas de Fourier (FFT) e inversas (IFFT), el esquema de la Figura 2. 4 queda modificado de la siguiente manera [9]:

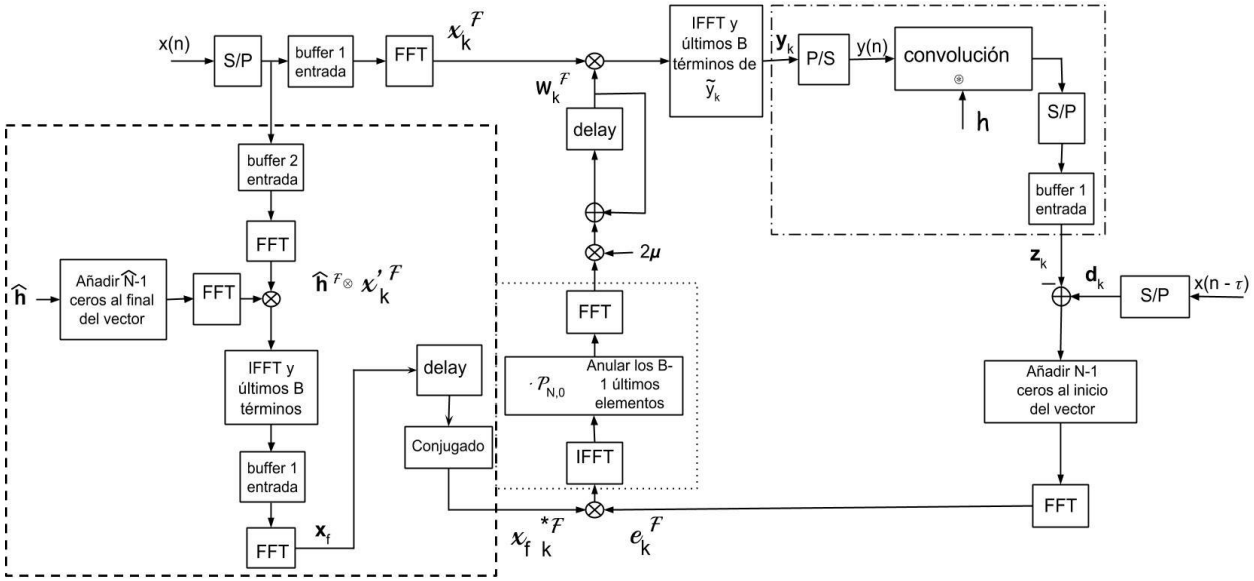


Figura 2. 11: Esquema FxFBLMS

Se comprueba fácilmente que únicamente se han incorporado dos etapas nuevas (destacadas por rectángulos a rayas) respecto al esquema FBLMS de la Figura. La modificación de la izquierda de la Figura 2. 11 corresponde a considerar el canal estimado $\widehat{\boldsymbol{h}}$ de $\widehat{N}$ elementos mientras que el filtro adaptado $\boldsymbol{w}_k$, para la ecualización del canal $\boldsymbol{h}$, es de N coeficientes. Esto implica realizar la salida convolucional en la frecuencia entre filtros y señales de entrada con buffers de tamaño adecuado en cada uno. Por esto se identifica el buffer_2 de $\widehat{N} + B - 1$ y buffer_1 de $N + B - 1$

como tamaño de las muestras a la entrada, tanto antiguas como las B nuevas. La segunda ventana destacada a la derecha es la respuesta del sistema bajo estudio.

La matemática implícita se mantiene con la expuesta en el punto **¡Error! No se encuentra el rigen de la referencia.** con la introducción de éstas dos nuevas etapas comentadas. Para el bloque *k*-ésimo, $\widetilde{\boldsymbol{x}}_k$ tiene como entrada el buffer_1 y $\widetilde{\boldsymbol{x}}'_k$ entra al buffer_2. $\widehat{\boldsymbol{h}}^{\mathcal{F}}$ hace referencia a la FFT del canal estimado $\widehat{\boldsymbol{h}}$.

En general el algoritmo precisa de los siguientes pasos:

1. Actualizar el nuevo bloque de B muestras en $\widetilde{\boldsymbol{x}}_k$ y $\widetilde{\boldsymbol{x}}'_k$. Generar $\boldsymbol{x}_k^{\mathcal{F}}$ y $\boldsymbol{x}'^{\mathcal{F}}_k$
2. Calcular la salida por el canal estimado $\boldsymbol{x}_{f\,k}$ y generar $\boldsymbol{x}_{f\,k}^{\mathcal{F}}$
3. Obtener los últimos B elementos de la salida $\boldsymbol{y}_k = \boldsymbol{\mathcal{F}^{-1}}\left(\boldsymbol{w}_k^{\mathcal{F}} \circ \boldsymbol{x}_k^{\mathcal{F}}\right)$
4. Recibir $\boldsymbol{m}_k$ como la respuesta $\boldsymbol{y}_k$ con el sistema $\boldsymbol{h}$
5. Estimar el error $\boldsymbol{e}_k = \boldsymbol{d}_k - \boldsymbol{m}_k$ y transformarlo a la frecuencia $\boldsymbol{e}_k^{\mathcal{F}}$
6. Restringir el paso $\boldsymbol{\phi}_{k+1}^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\left\{\begin{bmatrix}\boldsymbol{\mathcal{F}^{-1}}\{\boldsymbol{w}'^{\mathcal{F}}_{k+1}\} \\ \mathbf{0}\end{bmatrix}\right\}$, seleccionando los N primeros elementos de la IFFT de $\boldsymbol{w}'^{\mathcal{F}}_{k+1} = \boldsymbol{x}_k^{\mathcal{F}^*} \cdot \boldsymbol{e}_k^{\mathcal{F}}$ y el vector de ceros **0** de B elementos.
7. Actualizar el filtro en frecuencia $\boldsymbol{w}_{k+1}^{\mathcal{F}} = \boldsymbol{w}_k^{\mathcal{F}} + 2\boldsymbol{\mu}_k \cdot \boldsymbol{\phi}_{k+1}^{\mathcal{F}}$

El caso del FxPFBLMS, funciona como se explicó en el punto 2.2.2.3 sobre el mismo esquema de la Figura 2. 11 aunque replicado paralelamente con retardos entre filtros particionados y considerando las contribuciones de todos ellos para la actualización. La etapa del filtrado con la función de transferencia estimada se resolvería con su particionado. Se desarrolla a continuación de forma genérica este algoritmo ecualizador.

2.3.2.2 FxPFBLMS multicanal

En un sistema de ecualización adaptativa de varias entradas y salidas (MIMO) intervienen en términos acústico la emisión de I fuentes de sonido por *J* altavoces y recibida por *K* altavoces como se muestra en la Figura 2. 12:

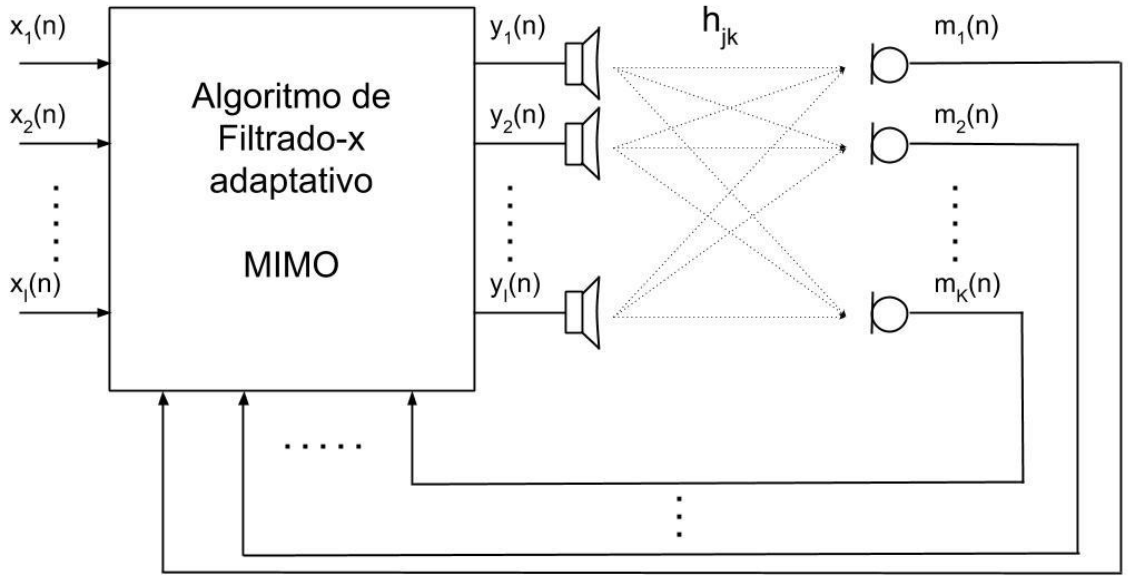


Figura 2. 12: Sistema MIMO adaptativo

Esto implica considerar en el proceso de ecualización, para cada fuente *i*, *J·K* diferentes canales $\boldsymbol{h}_{jk}$con su correspondiente estimación $\widehat{\boldsymbol{h}}_{jk}$ de $\widehat{N}$ elementos, existiendo además K salidas $\boldsymbol{m}_{k_n}$ del sistema global, por tanto K vectores de estimación de error $\boldsymbol{e}_{k_n}$, y J·I filtros transversales $\boldsymbol{w}_{ij_n}$ de N coeficientes como se detalla en la imagen contigua. Denotar que la iteración temporal de cada nuevo bloque de muestras a la entrada se identifica ahora por el subíndice *n* en lugar de *k* debido a la notación multicanal introducida [6] [7].

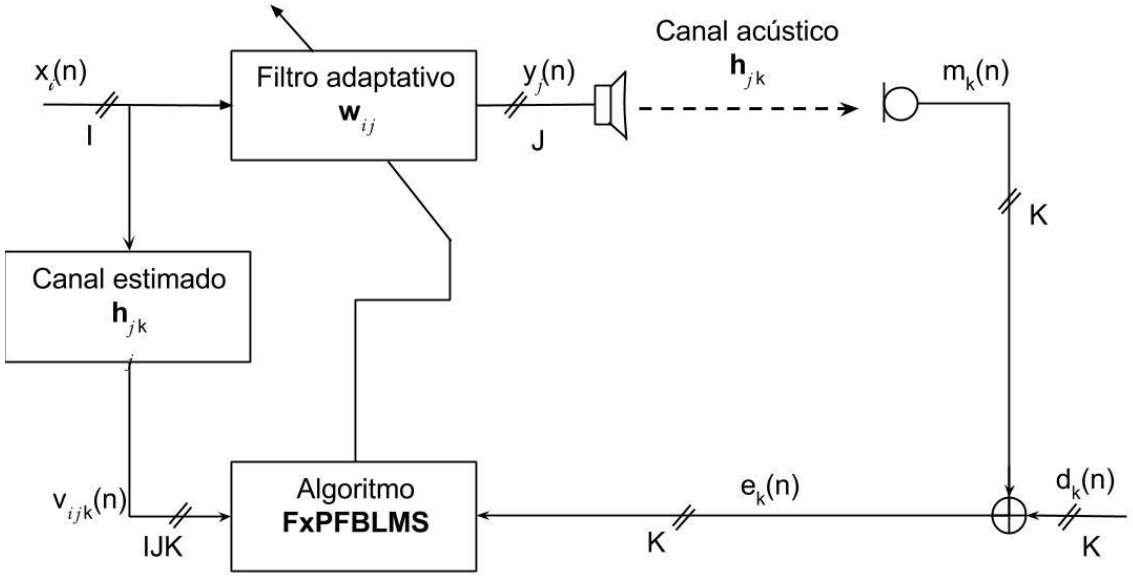


Figura 2. 13: Esquema conceptual de ecualización adaptativa MIMO

La composición de los filtros será de $F = N/M$ particiones, de M elementos para cada filtro adaptativo transversal o ecualizador, y de $P = \widehat{N}/M$ para los estimados siendo *F*, *P* y *M* enteros. Continuando con la suposición de que *M=B·q* la salida del filtro adaptativo se calcula como sigue:

$$\boldsymbol{y}_{j_n} = \sum_{i=0}^{I-1} \sum_{f=0}^{F-1} \boldsymbol{\mathcal{F}^{-1}} \left(\boldsymbol{w}_{ij,f_n}^{\mathcal{F}} \circ \boldsymbol{x}_{i\,n-fq,0\,n}^{\mathcal{F}} \right)$$

Para:

$$\boldsymbol{x}_{i\,n-fq,0\,n}^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\{[x_i(nB - Mf - M), x_i(nB - Mf - M + 1), \dots, x_i(nB - Mf - M + B - 1)]\}$$

La transformada de Fourier de cada bloque *f* particionado a la entrada del filtro.

También se utilizará en la etapa del filtro estimado h_{jk}, siendo cada partición *p*:

$$\boldsymbol{x}_{i\,n-pq,0\,n}^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\{[x_i(nB - Mp - M), x_i(nB - Mp - M + 1), \dots, x_i(nB - Mp - M + B - 1)]\}$$

Generalizando las dos entradas se puede sustituir tanto *f* como *p* por *l* quedando el mismo bloque de señal de entrada en función del número de particiones:

$$\boldsymbol{x}_{i\,n-lq,0\,n}^{\mathcal{F}} = \boldsymbol{\mathcal{F}}\{[x_i(B(n - lq) - M), x_i(B(n - lq) - M + 1), \dots, x_i(B(n - lq) - M + B - 1)]\}$$

En resumen el algoritmo precisa de los siguientes pasos:

1. Actualizar el nuevo bloque de B muestras en $\widetilde{\boldsymbol{x}}_{\iota\,n,0}$ y generar $\boldsymbol{x}_{i\,n-lq,0}^{\mathcal{F}}$

2. Calcular la salida por el canal estimado $\boldsymbol{v}_{ijk_n} = \sum_{p=0}^{P-1} \boldsymbol{\mathcal{F}^{-1}} \left(\widehat{\boldsymbol{h}}_{jk,p\,n}^{\mathcal{F}} \circ \boldsymbol{x}_{i\,n-pq,0\,n}^{\mathcal{F}} \right)$ y generar adecuadamente los JK vectores particionados en F tramas de M elementos $\boldsymbol{V}_{ijk,f}^{\mathcal{F}}$

3. Obtener los últimos B elementos de la salida para cada altavoz sumando las contribuciones de las I fuentes $\boldsymbol{y}_{j_n} = \sum_{i=0}^{I-1} \sum_{f=0}^{F-1} \boldsymbol{\mathcal{F}^{-1}} \left(\boldsymbol{w}_{ij,f_n}^{\mathcal{F}} \circ \boldsymbol{x}_{i\,n-fq,0\,n}^{\mathcal{F}} \right)$

4. Recibir $\boldsymbol{m}_{k_n}$ como la suma de las contribuciones $\boldsymbol{y}_{j_n}$ con el sistema $\boldsymbol{h}_{jk}$ sobre cada altavoz, esto es $\boldsymbol{m}_{k_n} = \sum_{j=0}^{J-1} \boldsymbol{h}_{jk} * \boldsymbol{y}_{j_n}$

5. Estimar el error $\boldsymbol{e}_{k_n} = \boldsymbol{d}_{k_n} - \boldsymbol{m}_{k_n}$ y transformarlo a la frecuencia $\boldsymbol{e}_{k_n}^{\mathcal{F}} = \boldsymbol{\mathcal{F}} \left\{ \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{e}_{k_n} \end{bmatrix} \right\}$ extendiendo las B muestras de $\boldsymbol{e}_{k_n}$ con un vector **0** de M ceros.

6. Restringir el paso de actualización del filtro calculando $\boldsymbol{\phi}_{ijk,f_n}^{\mathcal{F}} = \boldsymbol{\mathcal{F}} \left\{ \begin{bmatrix} \boldsymbol{\mathcal{F}^{-1}} \{ \widehat{\boldsymbol{\mu}}_{ijk,f}^{\mathcal{F}} \} \\ \boldsymbol{0} \end{bmatrix} \right\}$

   Para ello seleccionar los primeros M coeficientes de $\boldsymbol{\mathcal{F}^{-1}} \{ \widehat{\boldsymbol{\mu}}_{ijk,f}^{\mathcal{F}} \} = \boldsymbol{\mathcal{F}^{-1}} \{ \boldsymbol{V}_{ijk,f}^{\mathcal{F}\,*} \cdot \boldsymbol{e}_k^{\mathcal{F}} \}$ y extendiendo el cálculo de la FFT con un vector **0** de B ceros.

7. Actualizar el filtro en frecuencia $\boldsymbol{w}_{ij,f_{n+1}}^{\mathcal{F}} = \boldsymbol{w}_{ij,f_n}^{\mathcal{F}} + 2\boldsymbol{\mu}_k \cdot \sum_{k=0}^{K-1} \boldsymbol{\phi}_{ijk,f_n}^{\mathcal{F}}$.

El esquema de funcionamiento por partición se muestra en el siguiente diagrama de bloques:

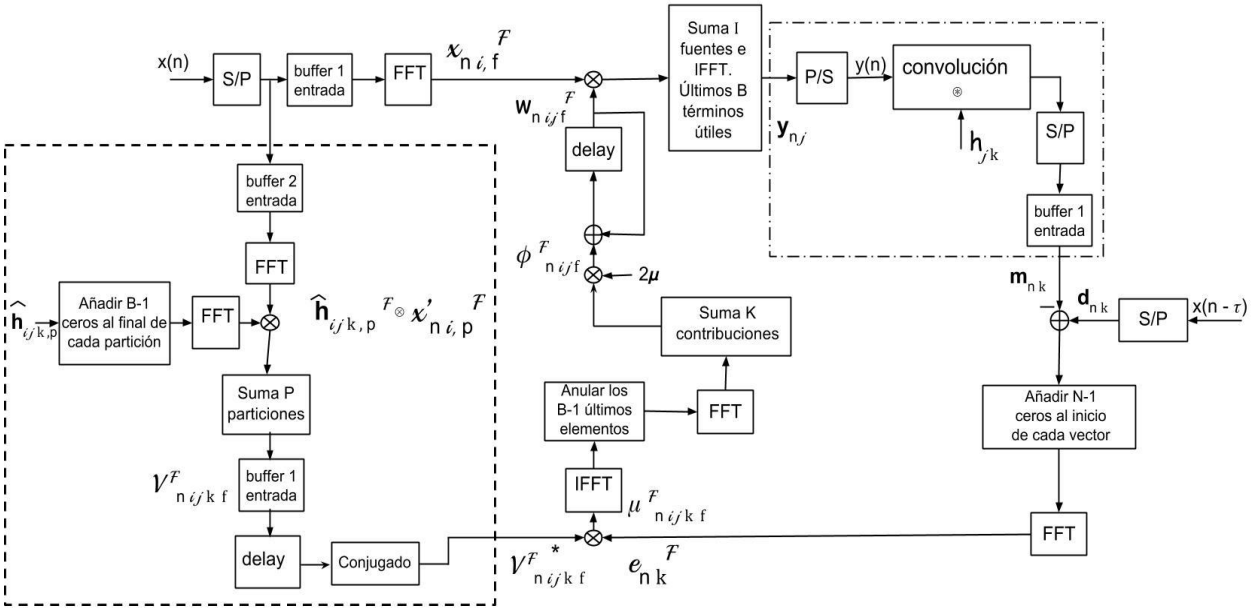


Figura 2. 14: Esquema FxPFBLMS MIMO

Los buffers almacenan las entradas anteriores desplazando las columnas con cada iteración en matrices de (M+B-1)xL para *L={P o F}* según se trate del buffer_2 ó del buffer_1 respectivamente.

Capítulo 3. Análisis de algoritmos adaptativos en Matlab.

3.1 Diseño y análisis.

Para desarrollar la aplicación del sistema en tiempo real sobre CPU es imprescindible analizar previamente el comportamiento del código en un entorno de simulación de datos como Matlab permitiendo esto diseñar la estructura del programa base a implementar.

3.1.1 Identificación de canal

Detallada la estructura clásica de los sistemas adaptativos de identificación de canal en el apartado 2.3.1.1 figura 2.8 se aprecia que principalmente se necesita para simular los resultados una respuesta impulsional que caracterice el canal a identificar *h* ó la señal objetivo *d* a su salida y una señal de entrada *x* que consistirá en una secuencia de ruido blanco gaussiano, de valor promedio μ_x cero y varianza σ^2 igual a la potencia de *x* por el hecho que $\sigma^2 = E[x^2] - \mu_x{}^2$, de tal modo se hace uso de una entrada estocástica, esto es no periódica y sin correlación. Además la densidad espectral de potencia de ésta señal aleatoria cubre todo el espacio de frecuencias uniformemente con valor σ^2. Para generarla se recurre a la función interna de Matlab *randn* como se especifica en las funciones adjuntas.

Recordar la importancia del factor de convergencia μ por su implicación en la divergencia o convergencia del proceso así como su dependencia entre velocidad de convergencia y desajuste de adaptación.

Indicar que en las versiones del algoritmo LMS por bloques se considerará que en cada bloque de entrada existan tantas muestras nuevas como el tamaño del filtro al que atacan. En otras palabras, para un filtro de N coeficientes, el bloque de entrada será de *B=N* muestras. Ídem con la modificación por particiones, haciendo que cada filtro particionado sea de *M=B* elementos.

3.1.1.1 LMS y NLMS

Los resultados de la simulación de los algoritmos en Matlab que a continuación se presentan se han obtenido a partir de un filtro desconocido de 32 elementos como canal acústico evaluado en un entorno tanto ideal como afectado por un ruido interferente sobre la señal deseada.

La convergencia del algoritmo se determina calculando en cada iteración la raíz de la suma de los la diferencia al cuadrado entre las componentes del filtro real *h* y el adaptado *w*. Se obtiene la siguiente evolución del ajuste para diferentes valores del factor de adaptación μ:

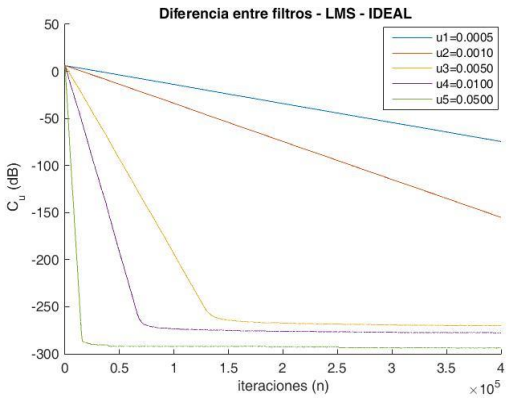


Figura 3. 1: Convergencia ideal LMS

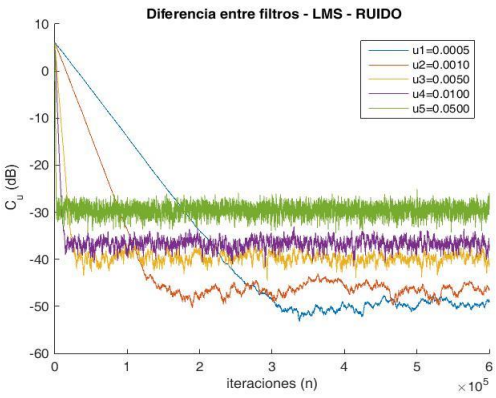


Figura 3. 2: Convergencia real LMS

Se aprecia que en el algoritmo converge idealmente muy rápido para valores de μ altos sin embargo si existe ruido los resultados de convergencia se estancan en peores niveles cuanto mayor es μ empeorando la identificación. Al contrario, cuanto menor es el factor de convergencia, requiere más iteraciones para alcanzar o mejorar la adaptación del filtro siendo además más estable.

El factor de convergencia del algoritmo μ determinará la velocidad de convergencia del sistema adaptativo siendo a su vez directamente proporcional con el desajuste o remanente del error respecto al caso ideal. Debe existir un compromiso entre tiempo que dedica el algoritmo y el nivel de convergencia al que puede llegar.

La velocidad de convergencia se puede apreciar claramente en el avance de las primeras muestras del error estimado

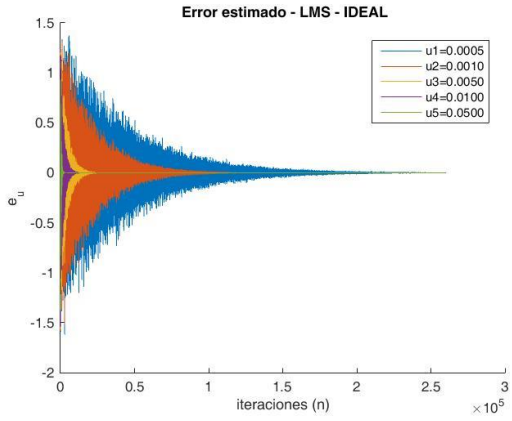


Figura 3. 3: Error ideal LMS

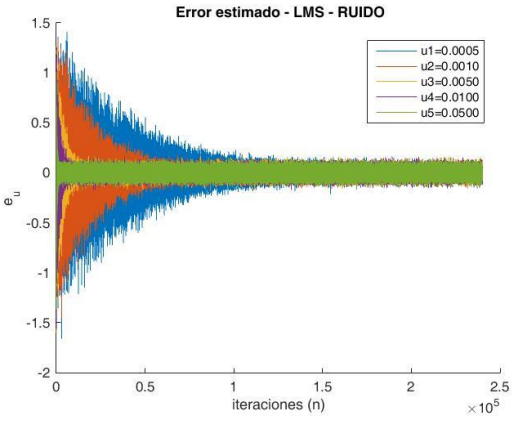


Figura 3. 4: Error real LMS

En condiciones ideales logra converger a un error estimado nulo más rápido cuanto más alto es μ si bien en presencia de ruido se alcanza un límite según la intensidad del ruido puesto que se trata de algoritmos lineales, esto es que existe una dependencia lineal entre la entrada y la salida.

Es interesante mostrar gráficamente la adaptación final del filtro con respecto al analizado en el caso con ruido, comprobándose la consistencia del algoritmo:

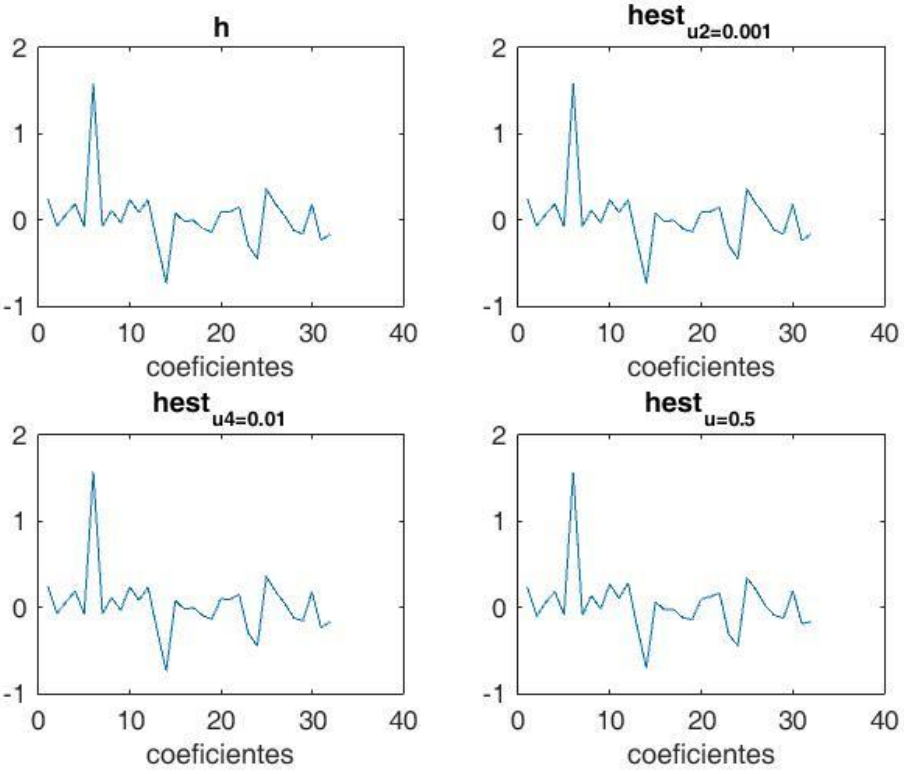


Figura 3. 5: Filtros con la identificación del canal *h*

Por último ejemplificar el uso de un factor de convergencia inestable ($\mu = 1.6 > 1$) causante de que el algoritmo diverja la adaptación de los coeficientes del filtro. El filtro resultante alcanza valores muy elevados:

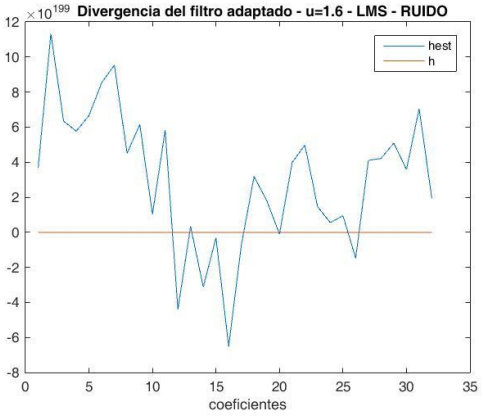


Figura 3. 6: Filtro inestable

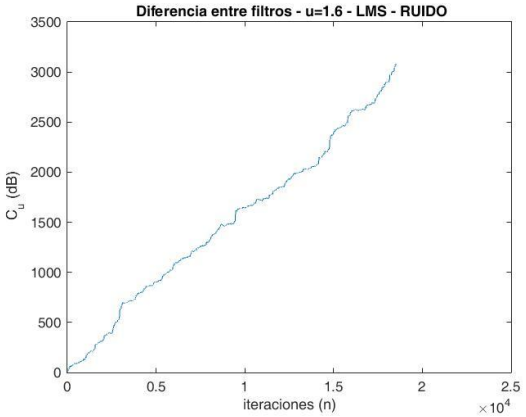


Figura 3. 7: Divergencia para μ alto

Siguiendo la teoría estudiada los valores de μ que aseguran la convergencia vendrán dados por la inversa de la potencia y el número de coeficientes, recordando que el hecho de aportar convergencia no implicaba necesariamente estabilidad:

(3. 1)

$$0 < \mu_{estable} < \frac{1}{3NP_x} < \mu_{convergente} < \frac{1}{NP_x}$$

En este caso tendríamos:

$$0 < \mu_{estable} < 0.224 < \mu_{convergente} < 0.674$$

La ventaja de normalizar el algoritmo, además de independizar el algoritmo de la potencia de la señal de entrada, es su mejora en estabilidad. Sin embargo requiere de más capacidad computacional para realizar la normalización y tarda más en converger como se puede ver en la comparativa de la convergencia entre NLMS y LMS:

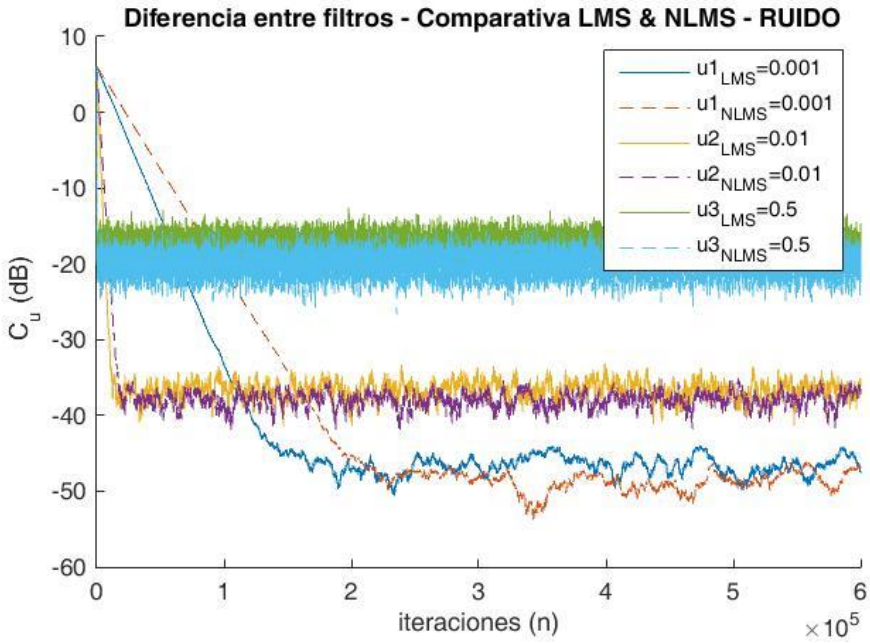


Figura 3. 8: Comparativa LMS, NLMS y μ

Puede comprobarse la estabilidad para valores altos de μ. Así, siguiendo el ejemplo anterior, para μ=1.6 obtenemos que el algoritmo NLMS no diverge mientras que el LMS si:

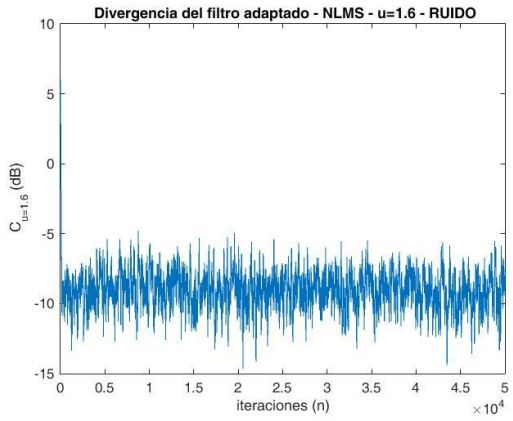


Figura 3. 9: Convergencia del NLMS

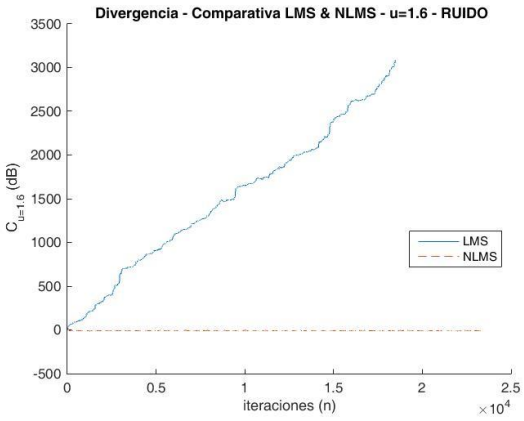


Figura 3. 10: Robustez a la divergencia

3.1.1.2 BLMS y FBLMS

El algoritmo LMS tradicional trabaja muestra a muestra, sin embargo el caso práctico que se está tratando requiere trabajar con bloques de señal por requerimientos de procesado sobre CPU, empleando buffers a la salida y entrada de los equipos acústicos.

Como se ha venido comentando el algoritmo BLMS tiene una versión en la frecuencia que mejora la eficiencia reduciendo el coste computacional con la Transformada Rápida de Fourier (FFT) utilizando un bloque con tantas muestras nuevas como elementos tenga el filtro transversal, esto es *M=B*.

A continuación se muestra la convergencia del BLMS para diferentes tamaños de bloque B y $\mu = 0.01$

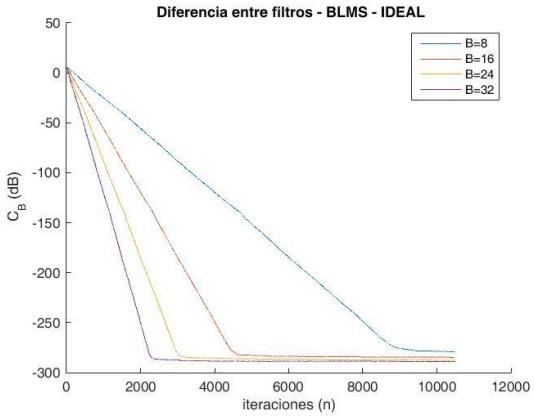


Figura 3. 11: Efecto tamaño de bloque B

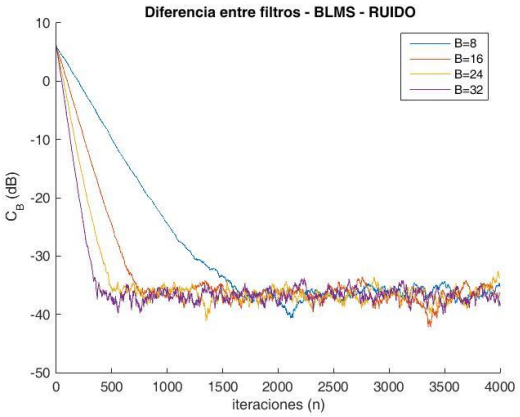


Figura 3. 12: Tamaño bloque con ruido

Se extrae que cuanto mayor es B más rápido converge, es decir que logra adaptar el filtro más velozmente que el LMS, que en este caso sería considerar B=1. En la gráfica del BLMS con ruido comprobamos que sea el valor que tome B el nivel de diferencia entre filtro real y transversal se estanca en función del ruido existente, aunque para valores de B bajos se producen menos perturbaciones en la adaptación, dicho de otro modo es más estable.

Por tanto es preferible hacer uso de un tamaño de bloque B máximo, esto es igual al número de coeficientes del filtro, para disminuir el número de iteraciones necesarias. La repercusión afecta sobre el coste y tiempo computacional hasta alcanzar la convergencia. En caso de necesitar mayor precisión se puede recurrir a disminuir el factor μ.

Una comparativa global de las 4 modificaciones muestra que la versión en frecuencia es la que mejor responde en velocidad de convergencia superando el comportamiento del algoritmo BLMS. La versión normalizada del FBLMS se muestra como más estable durante las iteraciones y destaca como la que mayor nivel de parecido logra.

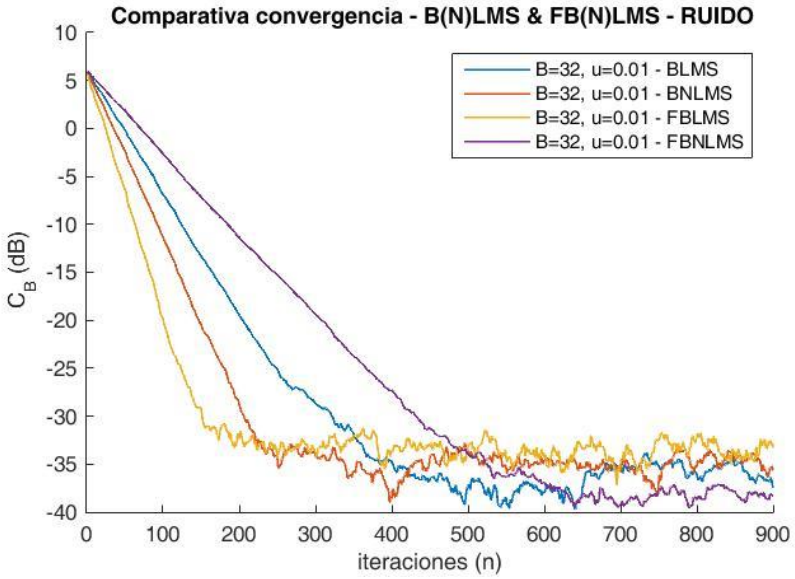


Figura 3. 13: Comparativa BLMS y FBLMS

3.1.1.3 PFBLMS

Una limitación del algoritmo en la frecuencia por bloques (FBLMS) es el tamaño de la respuesta del sistema capaz de identificar ligado al tamaño de bloque que la CPU puede emplear. La tarjeta de sonido MOTU empleada limitan el tamaño del buffer máximo a 2048 muestras. Si se desea obtener una respuesta mayor a este límite el algoritmo FBLMS no es útil. Por esta razón se recurre al particionado del filtro en divisiones de B elementos. Además este algoritmo se puede emplear paralelamente dado que cada partición se adapta y trabaja independientemente en frecuencia.

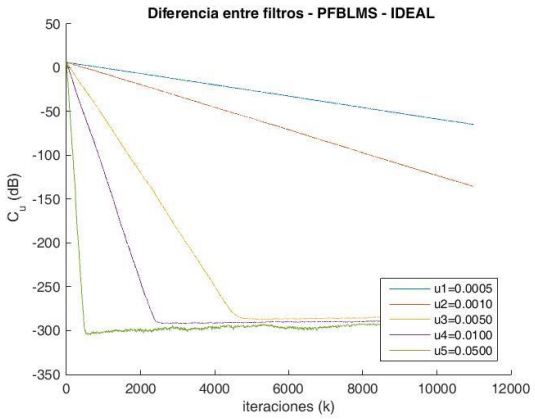


Figura 3. 14: Convergencia ideal para diferentes μ

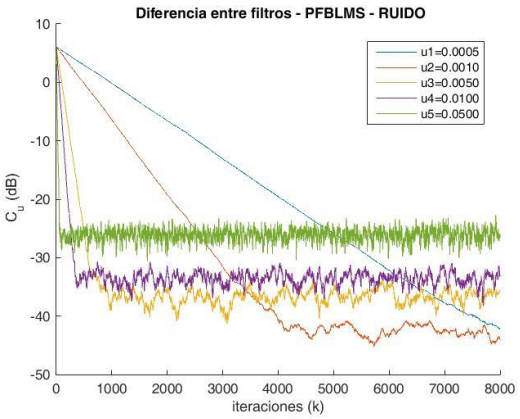


Figura 3. 15: Convergencia con ruido para diferentes μ

El algoritmo se comporta a grandes rasgos como los ya comentados. Conforme aumenta el factor de convergencia la adaptación empeora al aumentar la desviación del ajuste cada iteración repercutiendo en la estabilidad y nivel de similitud entre filtros. En caso de aumentar el tamaño del bloque B el algoritmo converge antes como hemos visto pero tiene más perturbaciones haciendo que la potencia del error sea mayor que para tamaños de bloque pequeños.

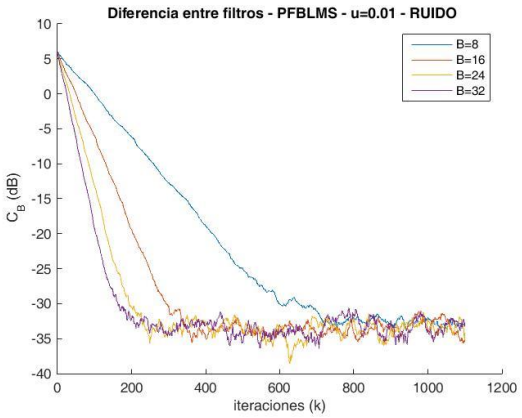


Figura 3. 16: Comparativa de B en convergencia idealmente.

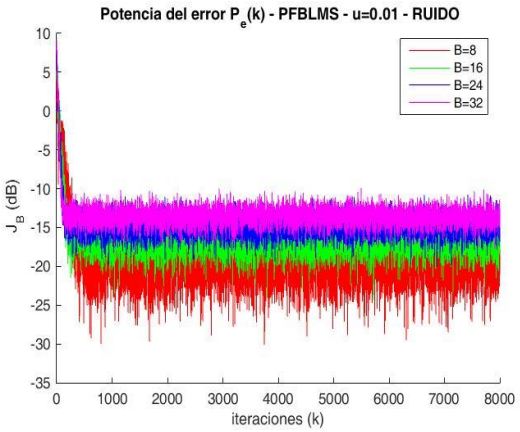


Figura 3. 17: Comparativa de B en convergencia con ruido

En cuando al FBLMS normalizado se aprecia en la gráfica siguiente que para valores de B grandes no existe gran diferencia en el nivel de convergencia alcanzado con la ventaja de que además fluctúa en menor medida:

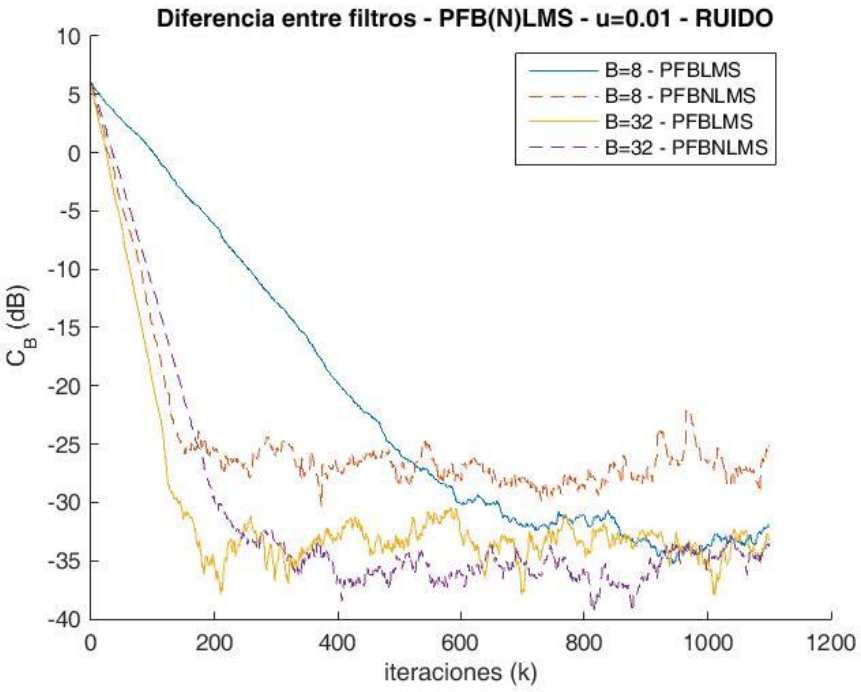


Figura 3. 18: Comparativa PFBLMS normalizado

Con el algoritmo PFBLMS se consigue reducir el tamaño del bloque con respecto al caso FBLMS, que sería igual al número de coeficientes del filtro transversal, además de ser el más rápido en converger en su versión normalizada como se extrae de la Figura 3.20.

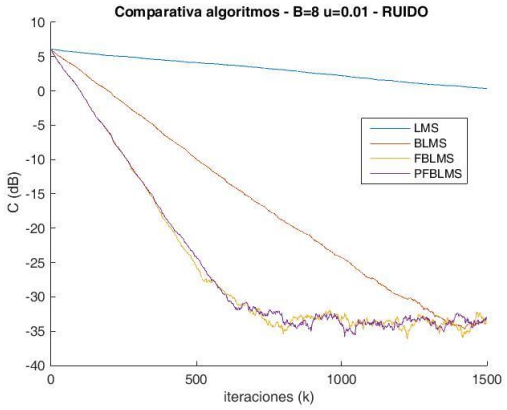


Figura 3. 19 Comparativa de la convergencia para algoritmos LMS

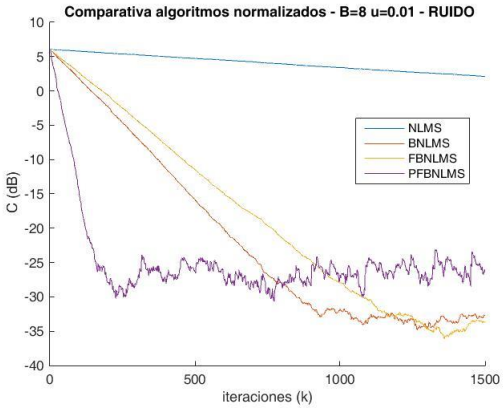


Figura 3.20 Comparativa de la convergencia para algoritmos LMS normalizados

De este modo se ha logrado obtener un algoritmo veloz en adaptación y cuyo nivel de parecido o nivel de identificación puede mejorarse ajustando el factor de convergencia.

En cuanto a la implementación en tiempo real se debe tener muy en cuenta los retardos del sistema para que el algoritmo converja adecuadamente. En particular para identificación existirá una diferencia de B muestras entre la señal de entrada y la señal deseada causada por los buffers de la tarjeta de sonido o la comunicación ASIO. La sincronía se logra retrasando la entrada una iteración, por ejemplo actualizando las B nuevas muestras al final, tras acabar el algoritmo.

Como consecuencia al retardo existente entre la entrada del micrófono y el buffer ocurre que la identificación resultante del algoritmo se desplaza B muestras. Esta característica se debe tener en cuenta si se quiere hacer uso de los resultados. Se ejemplifica a continuación con *B=16*:

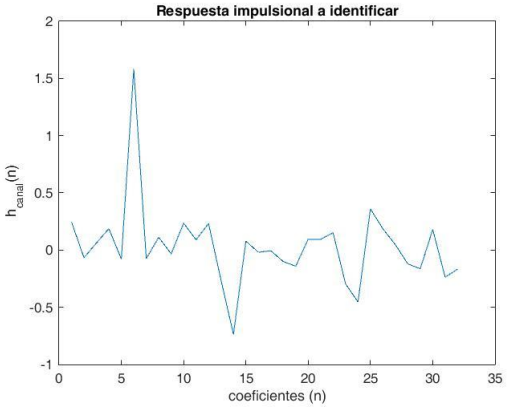


Figura 3. 21: Canal a identificar.

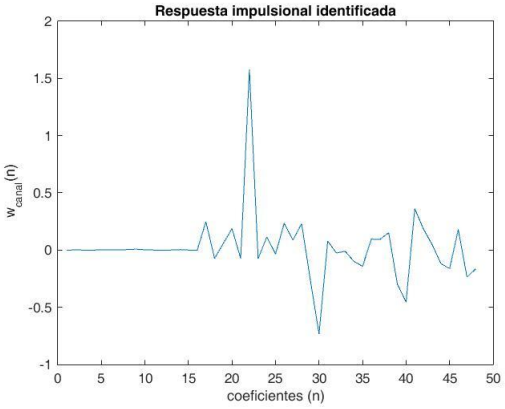


Figura 3. 22: Resultado de la identificación con retardo por simulación tiempo real.

3.1.2 Ecualización de canal.

Partiendo de la estructura del filtrado-x en transmisión detallada en la Figura 2. 10 del apartado 2.3.1.1. y centrando el estudio de la simulación en la versión del algoritmo PFBNLMS, puesto que es el que finalmente se decide utilizar por su ventaja computacional y rápida convergencia, se observa que la implementación en Matlab requiere como entradas la respuesta impulsional que caracteriza el canal acústico ó del sistema a ecualizar *h*, la estimación previa realizada en identificación h_{est} y la señal de entrada que estimule la adaptación que, al igual que se ha utilizado en identificación, se trata de una señal de ruido blanco gaussiano generada con *randn* cuyas propiedades, ya comentadas, son conocidas y permite garantizar una ecualiación del filtro genérica por ser banda ancha.

El tamaño de las particiones del filtro *M* serán del mismo número de elementos que el de los bloques de entrada *B* como se ha utilizado anteriormente. Puesto que los posibles valores de B que la aplicación en tiempo real permite son potencias de 2 desde 16 a 2048, se ceñirán los tamaños de los filtros *h* y h_{est} también a potencias de 2 para evitar particiones residuales que en todo caso se completarían con ceros.

De nuevo señalar la importancia del factor de convergencia μ por su implicación en la divergencia o convergencia del proceso así como su dependencia entre velocidad de convergencia y desajuste de adaptación.

El nivel de ecualización del algoritmo o convergencia se determina en cada iteración comparando la convolución resultante entre las respuesta de transferencia del filtro ecualizador y del canal o sistema a ecualizar. La convolución ideal daría como resultado una delta unidad desplazada cierto retardo. Calculando la diferencia o distancia entre la convolución ideal y la generada cada iteración puede evaluarse la convergencia del algoritmo [5]. Siendo $\|\cdot\|_2$ la l^2-norm (norma2):

(3. 2)

$$D_k = 20 \cdot log_{10}(\ \|\boldsymbol{w}_k * \boldsymbol{h} - \ \delta(n-\tau)\|_2 / \ \|\ \delta(n-\tau)\|_2 \ )$$

3.1.2.1 FxPFBNLMS monocanal

Comprobaremos el comportamiento del algoritmo filtrado-x particionado por bloques en la frecuencia con una respuesta identificada real de la sala en la que se testearán las pruebas en tiempo real. La respuesta *h* se puede ver en la Figura 3. 23:

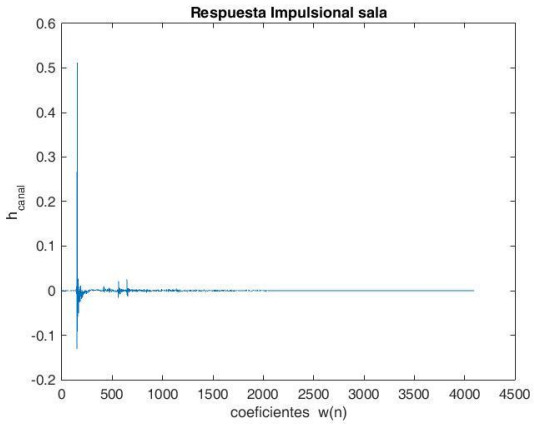


Figura 3. 23: Respuesta impulsional real

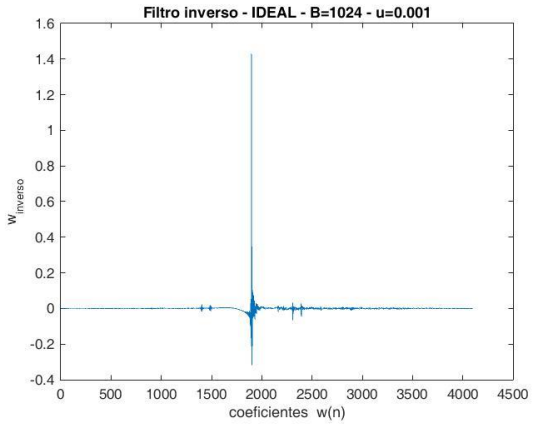


Figura 3. 24: Filtro inverso

Muestra baja reverberación, con la primera componente dominante y efectos secundarios multicamino poco importantes. A la derecha, Figura 3. 24, el filtro inverso correspondiente obtenido.

En condiciones ideales la respuesta estimada *hest* sería idéntica a la que se pretende ecualizar, *h*. En este caso, con los siguientes valores, el algoritmo responde de la siguiente manera:

Utilizar tamaños de bloque altos implica pocas iteraciones para alcanzar el mismo nivel que emplear bloques más pequeños y con más iteraciones. La convolución del filtro ecualizador final y la respuesta impulsional *h* consiste en una delta centrada y perfecta.

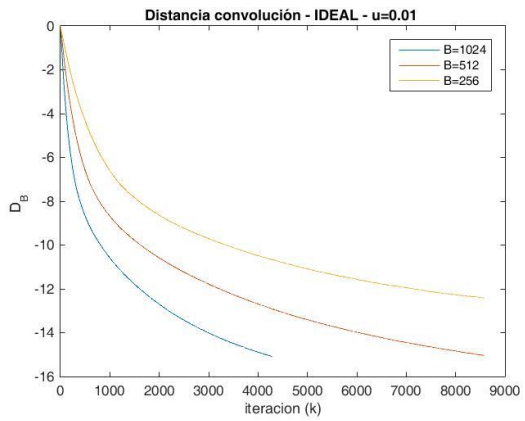


Figura 3. 25: Distancia con diferentes B

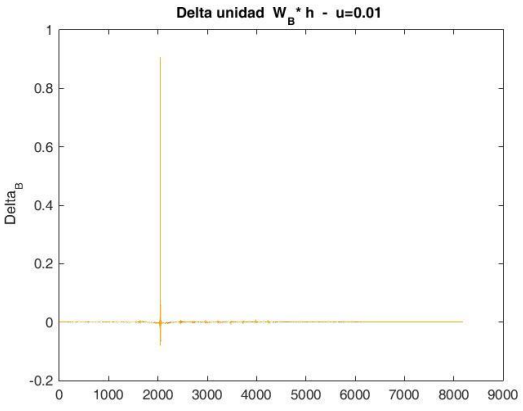


Figura 3. 26: Delta por convolución

El retardo *delay* permite desplazar la respuesta del filtro, centrándola para aprovechar el mayor número de coeficientes. La delta resultado de la convolución entre el filtro ecualizador de *N* coeficientes y la respuesta del canal *h* de *Nh* elementos estará desplazada también tantas muestras como el valor de *delay*. Este retardo pospone la actualización del filtro ecualizador puesto que la salida por éste es nula y por tanto el bloque error *e_B* está sincronizado con el bloque de la señal deseada *d_B*.

A partir de la presencia de la señal de entrada tras cierto retardo (*delay*) la señal error deja de ser nula y actualiza el filtro con la salida del canal estimado V_k de forma que a la iteración siguiente se obtiene datos a la salida del canal acústico.

La evolución del error se puede comprobar calculando la potencia o coste por bloque en cada iteración, como se muestra en la grafica de la Figura 3. 27, u observando la evolución que tiene el error con respecto a la señal deseada. Se pueden obviar las primeras *delay* muestras de estas dos señales porque son nulas.A partir del fin del retardo *delay*, el primer bloque del error distinto de cero será el primer bloque de la señal *x*. Las siguientes muestras del error serán la resta entre la salida combinada del ecualizador y del canal con la deseada, una diferencia que irá disminuyendo con el tiempo.

Se puede comprobar que tal relación entre potencias entre el error y la señal deseada permite hacer una idea de la convergencia:

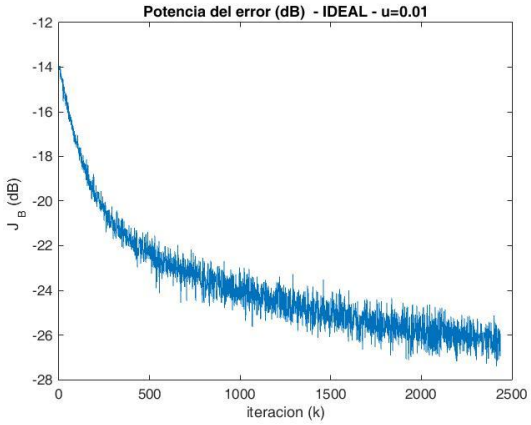


Figura 3. 27: Función de coste

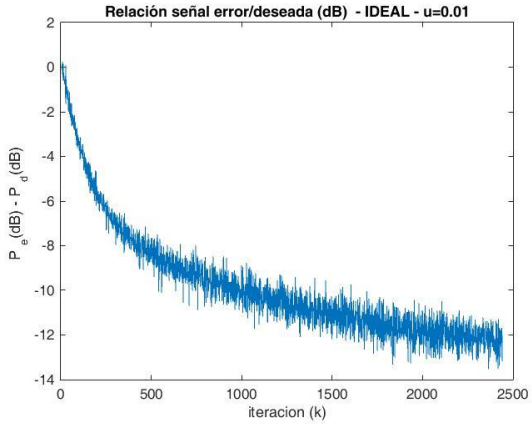


Figura 3. 28: Potencia del error respecto a la deseada

Por último se analizará el comportamiento en presencia de ruido tanto en la adquisición de la señal salida por el micrófono como en la respuesta *h*:

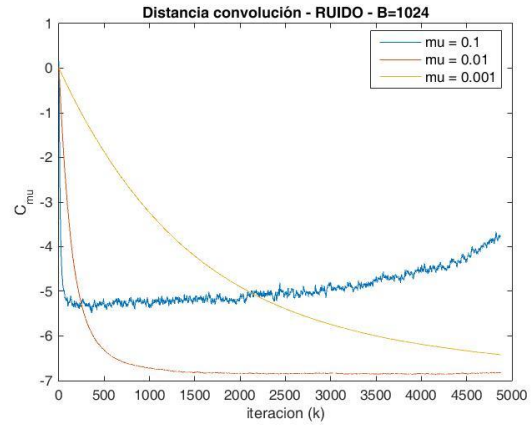


Figura 3. 29: Convergencia diferentes µ

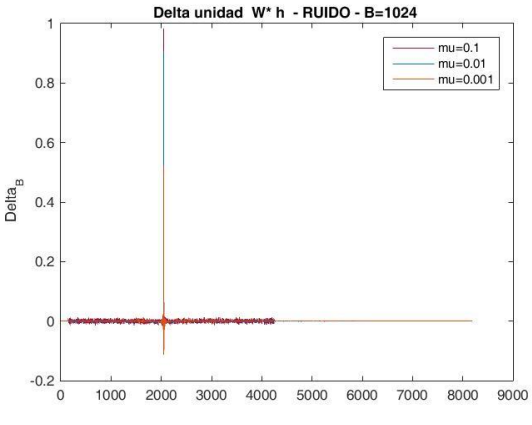


Figura 3. 30: Delta convolucionada

Se observa que la convergencia del algoritmo se vuelve inestable para factores µ en torno a 0.1, luego se emplearán por precaución valores centesimales. Por otro lado, en la delta generada se ve que con µ pequeñas se logra reducir la influencia del ruido sobre el canal útil para identificaciones estimadas en entornos ruidos o de menor calidad. Sin embargo no se alcanzan niveles de convergencia similares al caso ideal, todo lo contrario, son bastante peores.

El mismo resultado encontramos en cuanto a potencia del error. Se ve limitado por efecto de la interferencia.

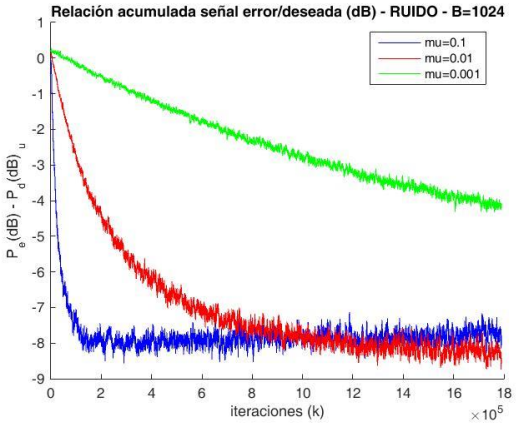


Figura 3. 31: Reducción del error

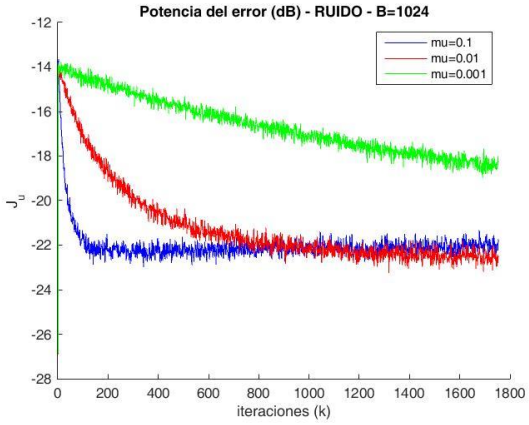


Figura 3. 32: Función de coste

Como conclusión, en el testeo de la aplicación en tiempo real se tendrá que tener en cuenta que cuanto mayor sea el valor del bloque, limitado a 2048 muestras, más rápida será la identificación y ecualización. En cuanto al factor de convergencia μ parece recomendable un valor en torno a [0.01 – 0.001], a expensas de comprobar el comportamiento con otros valores.

Para ejemplificar la capacidad de adaptación del algoritmo se muestra a continuación la convergencia durante el proceso de ecualización del canal *h1* y del canal *h2* distinto inicializando los coeficientes del segundo filtro con los obtenidos en la primera ecualización.

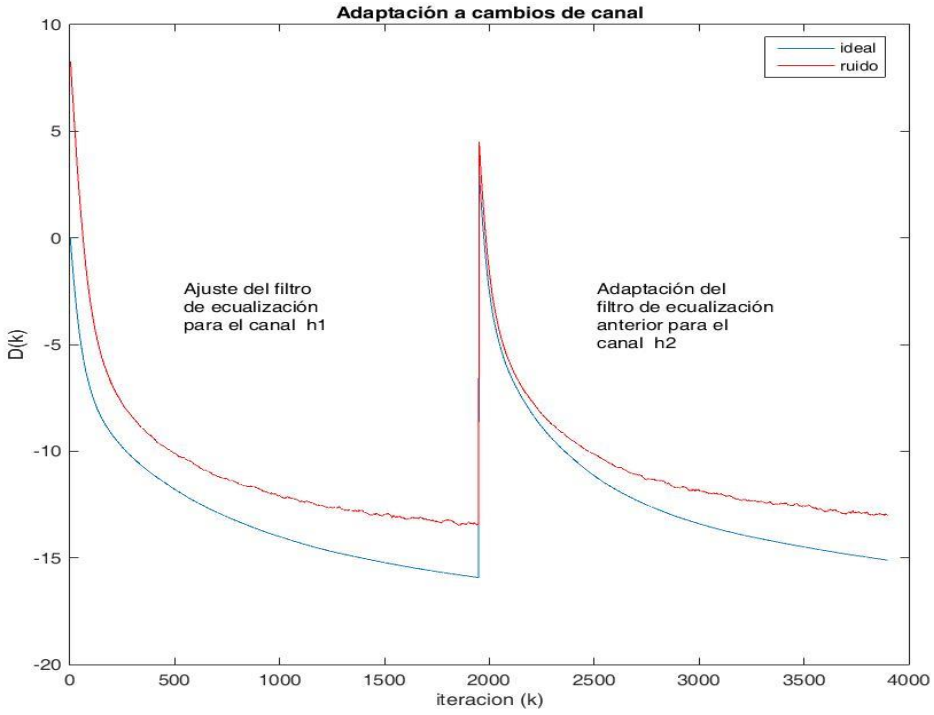


Figura 3. 33 Adaptación a cambios de canal

En cuanto a la simulación del comportamiento en tiempo real se debe tener en cuenta que el canal estimado no es idéntico al canal acústico puesto que se genera con un retado inicial de B

muestras como se ha visto en identificación. Además se deben tener en cuenta los tiempos de sincronía con los buffers en la emisión-recepción con los drivers ASIO.

El retardo temporal entre la entrada y salida de la tarjeta es del tamaño del bloque empleado, B muestras, debido a que en el instante *k*-1 se emite y el micrófono lo captura pero la adquisición de los datos se emplea en el instante siguiente *k*. Se compensa retardando la señal deseada una iteración, permaneciendo el *delay*, y así calcular sin problemas la estimación del error. La salida por el canal acústico estimado ya incorpora el retardo por ASIO-MOTU del proceso de identificación.

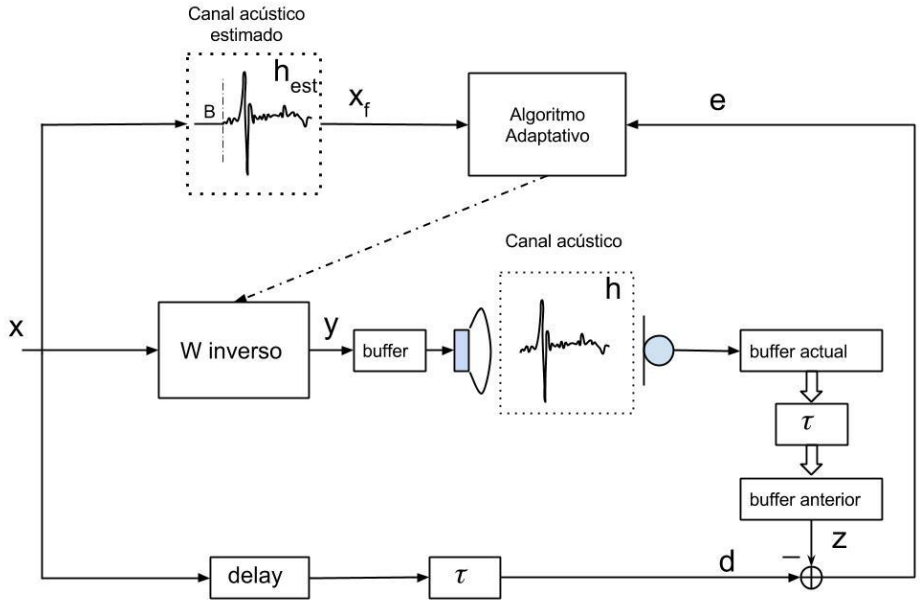


Figura 3. 34: Esquema retardos en tiempo real

Estas consideraciones suponen que se debe emplear el mismo tamaño de bloque B tanto en la ecualización como en identificación. De lo contrario los bloques de datos en las iteraciones no estarán sincronizados adecuadamente. Por otro lado, del mismo modo que viene ocurriendo, el resultado del filtro inverso estará desplazado *delay* muestras.

Empleando como canal el de la Figura 3. 23 y como respuesta estimada la identificación con el retardo por buffer se obtiene el filtro inverso de la Figura 3. 35, semejante al mostrado al inicio.

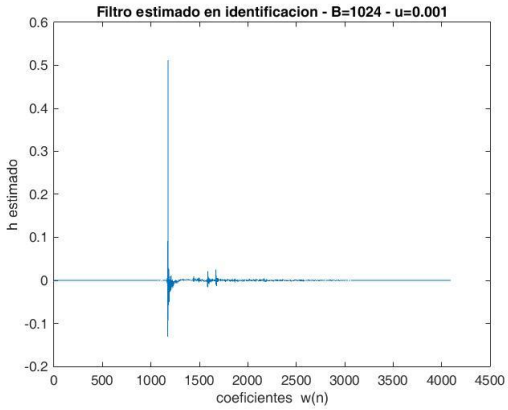


Figura 3. 35 Filtro estimado simulación tiempo real

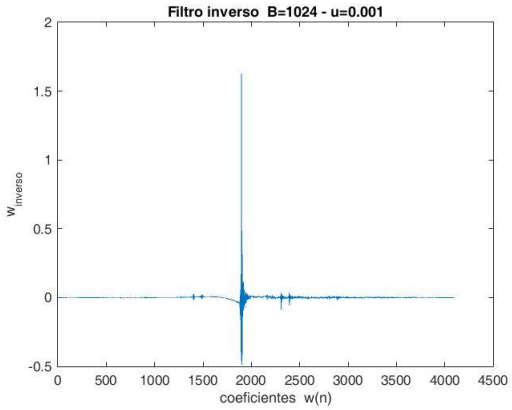


Figura 3. 36 Filtro inverso simulación tiempo real

3.1.2.2 FxPFBNLMS multicanal

Siguiendo el desarrollo teórico matemático estudiado en el capítulo 2 apartado **¡Error! No se ncuentra el origen de la referencia.** y con la estructura aplicada en el código de Matlab del algoritmo de ecualización monocanal se ha elaborado la función 'FxPFBNLMS_MIMO'.

El algoritmo admite *I* fuentes de entrada, agrupadas en columnas, *K* receptores y *J* emisores. Para estos valores existirán tantos canales como *J·K*.

Las entradas *h* y *hest* que corresponden a las respuestas de los canales acústicos y de las estimaciones de canal respectivamente, estas agrupadas en columnas. Debe respetarse un orden en la agrupación caracterizado por la relación entre el número del emisor (altavoz) y del receptor (micrófono). La columna *Col*, tanto de la matriz *h* como *hest*, deberá almacenar el canal entre el altavoz *j* y el micrófono *k*, para *j* y *k* números naturales $\mathbb{N} = \{1, 2, 3, 4,...\}$, del siguiente modo:

(3. 3)

$$Col = j + (k-1) * J \quad \rightarrow \quad h(:, Col) = h_{jk}$$

Esta disposición almacena consecutivamente para cada micrófono las *J* respuestas entre cada altavoz.

Cada fuente de sonido será ecualizada por un filtro independiente en cada altavoz *j* sumándose las *I* contribuciones para su emisión. Por tanto, en recepción, la señal deseada será la suma de las *I* fuentes. En cualquier caso se hará un análisis empleando una única fuente de entrada.

Con los siguientes canales reales *hMIMO* se identifican las respuestas estimadas *hestMIMO*:

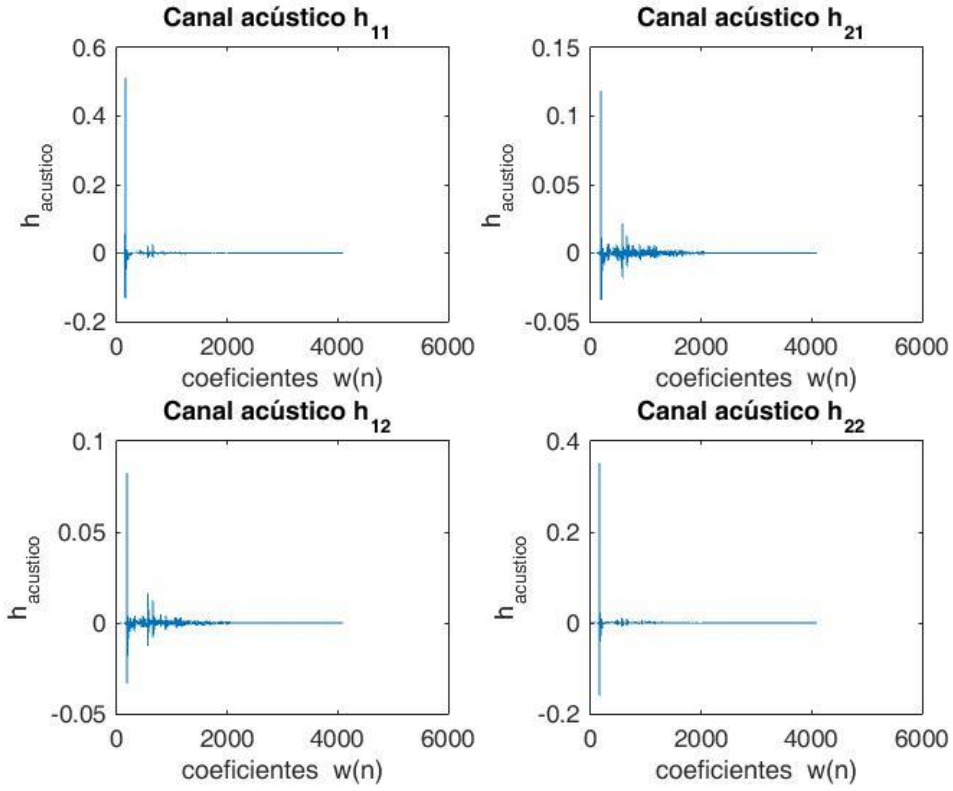


Figura 3.37: Canales acústicos para el análisis del algoritmo FxFPBNMS multicanal.

para, a continuación, obtener los siguientes resultados de convergencia sobre los puntos del espacio sonoro donde se encuentran situados los micrófonos o en el caso práctico posibles oyentes:

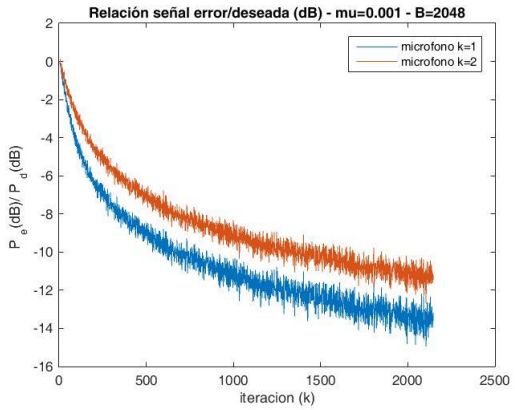


Figura 3.38 Evolución del error en MIMO

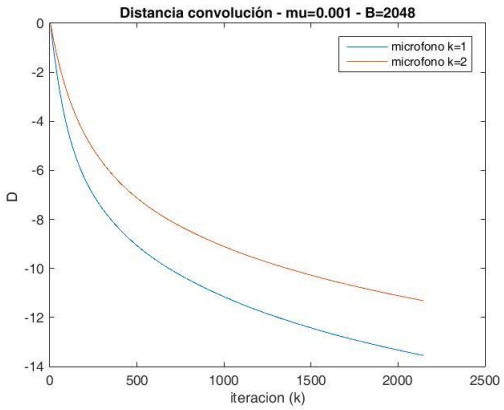


Figura 3.39: Distancia ecualización MIMO

Cada filtro inverso previo a la emisión por cada altavoz *j*:

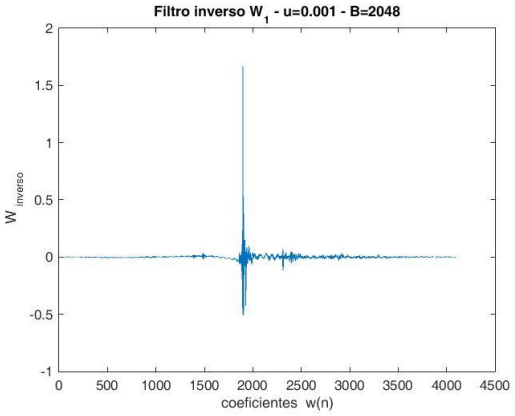


Figura 3.40 Filtro inverso j=1

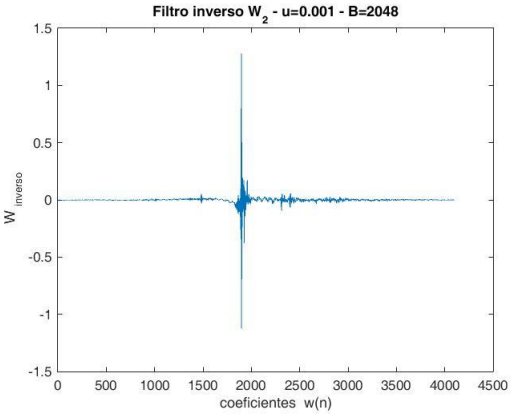


Figura 3.41: Filtro inverso j=2

El resultado ecualizado en la posición del micrófono o de un oyente se logra al sumar las salidas por cada altavoz, dicho de otro modo cada emisión por separado no está fielmente ecualiza. Se ejemplifica este caso a continuación, mostrándose la convolución de los filtros previos a cada altavoz con los canales acústicos correspondientes y la suma de todos ellos.

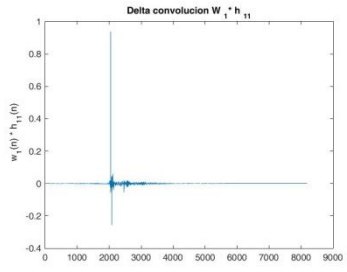


Figura 3.42 convolución W_1*h_{11}

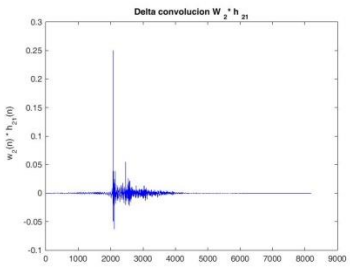


Figura 3.43 convolución W_2*h_{21}

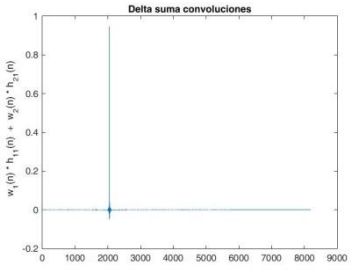


Figura 3.44 Suma convoluciones

Capítulo 4. Implementación en tiempo real

4.1 Programación en C

Un sistema en tiempo real responde a sucesos externos *in situ* dentro de un tiempo especificado para controlar inmediatamente el evento.

El desarrollo del programa se implementa en lenguaje de programación C, lenguaje de Alto Nivel mundialmente extendido que dispone de una sintaxis más natural e intuitiva a diferencia de lenguaje ensamblador. Al ser un lenguaje multiplataforma puede emplearse como software en PC y sistemas embebidos como microcontroladores.

Tras el estudio y análisis de los diferentes algoritmos adaptativos se selecciona como prototipo base para la aplicación de ecualización el correspondiente a la versión PFBLMS normalizada tanto para identificación como para ecualización, con la estructura del filtrado-x, ya que aporta una rápida convergencia y un eficiente procesado de computación explotando las propiedades que aporta trabajar en el dominio frecuencial. En contraposición sufrirá un mayor desajuste que se procura paliar seleccionando un adecuado valor para el factor de convergencia y normalizando el algoritmo.

El desarrollo en programación de la aplicación fue diseñar la estructura del código en C e incorporar sobre éste prediseño las funciones y sentencias relativas a los drivers ASIO para la comunicación con los equipos.

4.1.1 Prototipo de funciones generales

Los prototipos de funciones auxiliares utilizadas en cada programa se presentan agrupados al inicio de éste. Por ejemplo, en el programa de identificación:

```
void FFT(ComplexV dato, int i_fft);
void conjugar( ComplexV a);
double SignalPower( ComplexV v);
int readmewav(int iorden);
int entradaTeclado(char entrada_teclado[40]);
int valorDeB();
int valorDeP(int B);
int valorDeF(int B, int Nh);
void mostrarAllComplejo(ComplexV V);
void mostrarComplejo(Complex V);
Complex ProductoComplejo(Complex a, Complex b);
void exportarDatosTXT( ComplexV w);
void pausa();
```

La mayoría son intuitivas y fácilmente comprensibles visualizando el código. Sin embargo, y aprovechando que se usan indistintamente en identificación como en ecualización, se detallará a

continuación algunas más complicadas como el método para aplicar la Transformada Rápida de Fourier (FFT) y la adquisición de datos con la entrada de un archivo sonoro entre otros.

La función empleada para realizar la FFT e IFFT de tamaño L, para L=2^N muestras, es el método recursivo Danielson-Lanczos [10] [11], conocido desde 1982. Se basa en reescribir la FFT como suma de dos FFT de tamaño L/2, una con los elementos pares y la otra los impares de la siguiente manera:

$$ {}^{L}X_k = \sum_{j=0}^{L-1} e^{-\frac{2i\pi \cdot kj}{L}} \cdot x_j = \sum_{j=0}^{L/2-1} e^{-\frac{2i\pi \cdot k(2j)}{L}} \cdot x_{2j} + \sum_{j=0}^{L/2-1} e^{-\frac{2i\pi \cdot k(2j+1)}{L}} \cdot x_{2j+1} $$

$$ = \sum_{j=0}^{L/2-1} e^{-\frac{2i\pi \cdot kj}{L/2}} \cdot x_{2j} + W_L^k \sum_{j=0}^{L/2-1} e^{-\frac{2i\pi \cdot kj}{L/2}} \cdot x_{2j+1} = {}^{L/2}X_k^e + W_L^k \cdot {}^{L/2}X_k^o $$

Donde $W_L^k = e^{-\frac{2i\pi \cdot k}{L}}$ para $k = 0, \dots, L$.

Este procedimiento se puede aplicar recursivamente dividiendo las FFT de L/2 muestras en otras dos cada una de L/4 y sucesivamente hasta $\log_2 L = N$ trasformadas de Fourier. Cada trasformada puede identificarse binariamente considerando la partición par ó *even* como *e=0* y la partición impar ó *odd* como *o=1* quedando combinaciones de un solo elemento:

$$ {}^{L/L}X_k^{oooee\dots eo} = x_m $$

Se demuestra que la posición de cada muestra de la señal de entrada queda en orden opuesto o del revés a la actual en valor binario. Visto con un ejemplo, como el efecto del reflejo en un espejo:

$$ \begin{aligned} 0 &\rightarrow 000 \rightarrow \ 000 \rightarrow 0 \\ 1 &\rightarrow 001 \rightarrow \ 100 \rightarrow 4 \\ 2 &\rightarrow 010 \rightarrow \ 010 \rightarrow 2 \\ 3 &\rightarrow 011 \rightarrow \ 110 \rightarrow 6 \\ 4 &\rightarrow 100 \rightarrow \ 001 \rightarrow 1 \\ 5 &\rightarrow 101 \rightarrow \ 101 \rightarrow 5 \\ 6 &\rightarrow 110 \rightarrow \ 011 \rightarrow 3 \\ 7 &\rightarrow 111 \rightarrow \ 111 \rightarrow 7 \end{aligned} $$

Por lo tanto lo primero es reordenar los datos cambiando su posición de acuerdo a un orden según el revés de la posición binaria que ocupan empleando el doble de muestras complejas, *n*.

La segunda mitad del código es la realización propia de la FFT calculando la parte real e imaginaria de cada producto complejo por la exponencial compleja W_L^k correspondiente considerando que $e^{-\frac{2i\pi}{mmax}} = e^{i\cdot\theta} = wr + i \cdot wi = \cos(\theta) + i \cdot \sin(\theta)$.

De relaciones trigonométricas se tiene que $\mathrm{wr} = \cos(\theta) = 1 - 2 * \sin^2(\theta/2)$. El algoritmo comienza por parejas de elementos aumentando recursivamente el conjunto de transformadas por iteración hasta alcanzar *n.*

Para esta función se definen las constantes *fft=1* e *ifft=-1* puesto que según el signo de la variable *i_fft* la función Danielson-Lanczos ejecuta la transformada en frecuencia o la inversa. También se define la constante *PI* con el valor 3.14159265358979323846. Todo ello al inicio del programa.

La función de lectura del archivo wav, *readmewav*, descarta los 44 primeros bytes de la cabecera del formato utilizando el resto del archivo sonoro. Cada muestra sonora corresponde a 2Bytes que se almacena como bloques de 32bits añadiendo ceros delante. El valor decimal tipo *double* se obtiene dividiendo por el valor máximo tipo *int* de 32bits, 2^{31}. Se crean dos punteros

dobles globales de tipo *unsigned char* y *double* donde almacenar las señales leidas llamados *filewav32* y *señal* respectivamente.

Los coeficientes del filtro adaptado que se obtienen al finalizar el algoritmo se exportan en formato *.txt* con la función *exportarDatosTXT* especificando el nombre con que se desea salvar. El primer valor es el número de coeficientes o muestras del vector y el resto los valores reales del vector complejo intercalados por espacios. Esto es así porque se ha aprovechado la misma estructura de datos tanto para vectores en la frecuencia como en el dominio temporal siendo nula la parte imaginaria de estos segundos.

Para importar la estimación del filtro identificado en ecualización se usa la función *importarDatosTXT* que hace uso de otro puntero doble global llamado *h_t* en el que se almacenan los valores reales guardados en el archivo de texto. Se duplica el número de muestras de este vector con respecto al indicativo inicial del archivo por si se fuese a utilizar con vectores complejos, caso en el que se almacenaría la parte real e imaginaria por separado. Completado este puntero doble *h_t[pos]* se ordenan los valores correspondientes a la parte real en la variable tipo vector complejo *wH*, como se muestra:

```
for(cont1=0; cont1<wH.NumElemComp; cont1++){
	wH.c[cont1].r = (double) h_t[wHpos][cont1];
	wH.c[cont1].i = 0;
}
free(h_t[wHpos]);
```

Liberando de la memoria el puntero *h_t*.

Funciones relativas a los números complejos se tiene *conjugar* que cambia el signo de la parte imaginaria, *ProductoComplejo* calcula la parte real e imaginaria resultante de la multiplicación entre dos números complejos, *mostrarAllComplejo* saca por pantalla el vector de números complejos imprimiendo uno a uno con *mostrarComplejo*. Estos dos últimos se han utilizado para seguir el correcto funcionamiento en el proceso de programación. El cálculo de la potencia se resuelve con la función *SignalPower* sumando el cuadrado de las muestras.

Otras funciones como *valorDeP*, *valorDeF* y *valorDeB* son simples listas de opciones tipo *switch-case* que restringen las posibilidades para un correcto funcionamiento. La función *pausa*, utilizada a modo de *breakpoint*, permite pausar la ejecución hasta que se presiona *enter*, limpiando la pantalla.

Se añaden unas opciones de interrupción del programa por teclado de modo que presionando la tecla '*s*' se detenga la ejecución como si de una pausa se tratará reactivándolo fácilmente. Para un cierre definitivo de la ejecución se ha dispuesto la letra '*x*' de forma que se aborta el procesado y se puede exportar los datos hasta el momento calculados.

4.1.2 Drivers ASIO

Para compatibilizar el uso con los drivers de entrada y salida de audio ASIO se necesita incorporar al código las cabeceras principales a todas sus librerías, en concreto *asiosys.h*, *asio.h* y *asiodrivers.h* con las que inicializar la comunicación entre la tarjeta de sonido y el equipo de procesado. Por otro lado se hace servir la librería *time.h*. para cronometrar el tiempo de ejecución del programa en tiempo real.

La inclusión de ASIO en el código implica añadir la estructura *DriverInfo*, con alias *asioDriverInfo*, referida a los datos internos que se completará con las características especificadas a lo largo del proceso como número de canales de salida y de entrada, tamaño del buffer, frecuencia de muestreo, etc. También agregar las funciones necesarias para la inicialización y sincronización de la comunicación:

```
extern AsioDrivers* asioDrivers;
bool loadAsioDriver(char *name);
```

```
// internal prototypes (required for the Metrowerks CodeWarrior compiler)
int main(int argc, char* argv[]);
long init_asio_static_data (DriverInfo *asioDriverInfo);
ASIOError create_asio_buffers (DriverInfo *asioDriverInfo);
unsigned long get_sys_reference_time();

// callback prototypes
void bufferSwitch(long index, ASIOBool processNow);
ASIOTime    *bufferSwitchTimeInfo(ASIOTime    *timeInfo,long    index,ASIOBool
processNow);
void sampleRateChanged(ASIOSampleRate sRate);
long asioMessages(long selector,long value,void* message,double* opt);
```

De todas ellas la encargada del hilo de transmisión en tiempo real es *AsioTime* bufferSwitchTimeinfo* que recoge el valor seleccionado en el menú propio de ASIO en el mimebro *preferredSize* de la estructura interna, es decir *asioDriverInfo.preferredSize*, y donde se copia en cada puntero de memoria los datos como sigue:

```
// captura microfono
memcpy((int*)(pM[0]+cont_g*glbuffSize),(int*)asioDriverInfo.bufferInfos[0].bu
ffers[index], glbuffSize*sizeof(int));

//emision altavoz
memcpy((int*)asioDriverInfo.bufferInfos[nAltavoces[0]].buffers[index],(int
*)pAltavozBuff[0]+cont_g*glbuffSize,glbuffSize*sizeof(int));
```

Siendo *pM* y *pAltavozBuff* un puntero doble que, al ser monocanal, apuntan a un puntero simple cada uno, *pM[0]* como datos grabados por el micrófono y *pAltavozBuff[0]* como datos de archivo sonoro respectivamente.

Por otro lado ASIO tiene un buffer por cada canal de entrada o de salida. El identificado del buffer de cada micrófono o altavoz se realiza previamente en el menú propio de ASIO. Estableciendo las 4 primeras entradas [0, 1,2 y 3] para micrófonos y el resto, hasta el máximo de 96 conexiones, para altavoces [4,5,…,95], por lo que está limitado el número de buffers de entrada desde micrófonos. Este detalle es significativo porque se debe indicar la conexión al buffer adecuadamente. Por esto el buffer correspondiente a *pM* es *bufferInfos[0]* al utilizar el primer micrófono y el buffer correspondiente a *pAltavozBuffer* es *bufferInfos[4]* puesto que se emplea el primer altavoz.

En el desarrollo de los programas se explicará con más detalle la manera en que se realiza la comunicación con los equipos acústicos. En identificación se puede obtener toda la señal de salida del canal acústico mientras que en ecualización se procesa bloque a bloque.

Para activar los drivers de ASIO se debe ejecutar las instrucciones de inicialización o carga para configurar los datos internos. Esto se realiza dentro del programa principal antes de iniciar el algoritmo de adaptación. Cuando finaliza el procesado del algoritmo se debe desconectar la comunicación entre los equipos cerrando los drivers de ASIO. La introducción de estas fases en el código es superficial al algoritmo, lo cual supone pocas modificaciones. La que la estructura del programa en tiempo real tiene este aspecto:

INICIALIZACIÓN DRIVERS ASIO

```
Int main(){

// load the driver, setup all the necessary internal data structures
if (loadAsioDriver (ASIO_DRIVER_NAME)){
if (ASIOInit (&asioDriverInfo.driverInfo) == ASE_OK){
printf ("asioVersion:   %d\n"
"driverVersion: %d\n"
```

```
"Name:          %s\n"
"ErrorMessage:  %s\n",
asioDriverInfo.driverInfo.asioVersion,
asioDriverInfo.driverInfo.driverVersion,
asioDriverInfo.driverInfo.name, asioDriverInfo.driverInfo.errorMessage);
if (init_asio_static_data (&asioDriverInfo) == 0){
// set up the asioCallback structure and create the ASIO data buffer
asioCallbacks.bufferSwitch = &bufferSwitch;
asioCallbacks.sampleRateDidChange = &sampleRateChanged;
asioCallbacks.asioMessage = &asioMessages;
asioCallbacks.bufferSwitchTimeInfo = &bufferSwitchTimeInfo;
if (create_asio_buffers (&asioDriverInfo) == ASE_OK){
if (ASIOStart() == ASE_OK){
```

INICIO DEL ALGORITMO

...

ALGORITMO CON FLAGS

...

FIN DEL ALGORITMO – DESCONEXIÓN DRIVERS ASIO

```
ASIOStop();
} // if ASIO start
ASIODisposeBuffers();
printf("disposed\n");}}
ASIOExit();
printf("exit\n");   }
asioDrivers->removeCurrentDriver();
printf("removed driver\n");}

return 0;}
```

Mientras se ejecuta el algoritmo se activan unos flags que permiten el salto entre el procesado matemático de adaptación y la emisión/recepción de la señal en la función *AsioTime* bufferSwitchTimeinfo*.

En definitiva sigue el siguiente diagrama de estadas [12]:

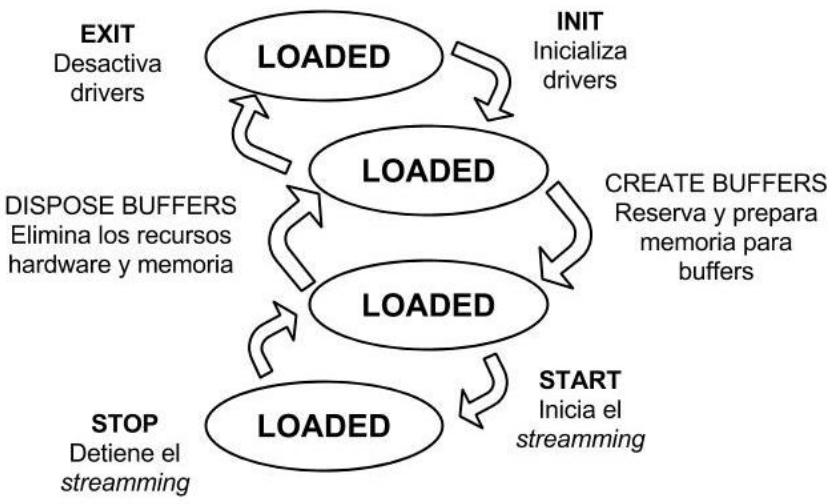


Figura 4. 1 Diagrama de estados básicos ASIO

Un aspecto importante en los programas con ASIO, en tiempo real, es que el valor del bloque B se ha de cambiar manualmente igualando la variable *B* al valor seleccionado en el menú o consola de la tarjeta MOTU.

4.1.3 Identificación

El diseño previo en C de la aplicación en tiempo real capaz de identificar el canal se llevó a cabo con el código del algoritmo en el programa principal de C y prototipos de funciones auxiliares anteriormente vistas, que se encuentran desarrolladas al finalizar el *main*.

Se incluyen en el programa las librerías internas *math.h* para expresiones matemáticas, *stdio.h* para entrada y salida de texto, finalmente las librerías *string.h* y *stdlib.h* para utilizar instrucciones relativas a la memoria tales como *memcpy*, *memset*, *calloc* …

4.1.3.1 Programa principal.

El programa principal se inicia con el almacenamiento en memoria del archivo sonoro en formato *.wav* de la señal AWGN empleada como entrada y, en el caso del diseño previo también la señal que capturaría el micrófono como resultado a la respuesta impulsional a identificar. Este proceso se realiza con la función denominada *readmewav* guardando los bytes en el puntero doble *filewav32* y en formato *double* en el puntero doble *senyal*. Se especifica para señal un punter a través de un indicativo siendo el de señal de entrada *xpos=0* y *ypos=1*.*MaxMues* guarda el número de muestras a utilizar de las señales.

```
unsigned char** filewav32=(unsigned char**) malloc(sizeof(unsigned char **));
double** senyal = (double **) malloc (sizeof(double **));

printf("Senyal entrada X:\n");
gMaxMues_x=readmewav(xpos);
printf("Senyal deseada Y:\n");
gMaxMues_y=readmewav(ypos);
ite= (gMaxMues_y>gMaxMues_x)?gMaxMues_x:gMaxMues_y;
```

Continúa con la inicialización de las variables principales solicitando el valor deseado para el tamaño de bloque *B* a utilizar y la longitud del filtro transversal a adaptar *N* con respecto al valor entero de particiones *P* permitido. Por último presenta por pantalla los valores globales.

```
B=valorDeB();
P=valorDeP(B);
nBlock=(ite)/B;
ite=B*nBlock;
N=P*B;
printf("B= %d , P = %d  ->  N = %d\n", B,P,N);
printf("length_x= %d , nBlock = %d\n",gMaxMues_x,nBlock);
printf("length_y= %d , ite= %d\n",gMaxMues_y, ite);
```

Se declaran todos los vectores y matrices (punteros) necesarios para llevar a cabo el algoritmo. Puesto que se trabaja en el dominio de la frecuencia existirán números complejos. Por facilidad se ha decidido crear una estructura, con el alias Complex, con dos elementos de tipo *double* que harán de parte real e imaginaria. A su vez, generalizando para todos los bloques de muestras utilizadas, se crea una estructura vectorial denominada *ComplexV* que representa los vectores complejos, indicando número de elementos y el puntero complejo.

```
typedef struct COMPLEX { double r; double i; }  Complex;
typedef struct COMPLEXV { Complex* c; int NumElemComp;} ComplexV;
```

Con esto se crean las variables vectoriales inicializando su reserva de memoria en cero con la instrucción *calloc*.

El algoritmo iterará hasta alcanzar el máximo de bloques de *B* muestras a generar por la señal de entrada, limitada temporalmente. Comienza con el primer bloque de la señal deseada (*d_B*) y de la de entrada (*inB*) almacenado en la parte real de la estructura vectorial compleja.

El procedimiento del algoritmo es superponer cada nuevo bloque a la entrada del filtro anterior (*in2B*) desplazando B elementos desde la posición inferior hacia arriba utilizando la instrucción *memcpy* Después se incorpora a la matriz de entrada (*inT)* de *2BxP* elementos. Se realiza la Transformada Rápida de Fourier (FFT) sobre esta matriz obteniendo la entrada a cada partición del filtro en la frecuencia (*IN*).

Esta entrada convolucionada por el filtro adaptado (*w*) en la frecuencia es el producto elemento a elemento entre *IN* y *W* mediante la función *ProductoComplejo* creada para tal propósito. La suma de las componentes frecuenciales de cada salida completa la respuesta del filtro a la entrada en el dominio frecuencial. La agrupación de las componentes en frecuencia de la salida se realiza muestra a muestra sobre la variable *Yaux* de la que, una vez transformada al dominio temporal mediante una IFFT, se seleccionan las últimas *B* muestras, obteniendo la salida temporal *outB* a comparar con la deseada *d_B*, lo cual da el error estimado *e_B*.

En la parte de ajuste se pasa el bloque error *e_B* al dominio frecuencial incorporando previamente B ceros en la parte superior del vector *E* y el bloque error en la parte inferior:

```
memcpy(E.c, zeroB.c, (B)*sizeof(Complex));
memcpy(E.c+B, e_B.c, (B)*sizeof(Complex));
FFT(E, fft);
```

El vector *E* es replicado *P* veces en la matriz *Etotal* calculándose a continuación la correlación entre el error y la entrada conjugada (*IN**) elemento a elemento, guardando el resultado en la variable *aux2*. Se calcula la IFFT de cada correlación para anular las *B* últimas muestras.

```
for(cont2=0; cont2<IN.NumElemComp; cont2+=2*B){
memcpy(ax2B.c, aux2.c+cont2, (2*B)*sizeof(Complex));
FFT(ax2B, ifft);
memcpy(aux2.c+cont2,ax2B.c, (B)*sizeof(Complex));
memcpy(aux2.c+B+cont2, zeroB.c, (B)*sizeof(Complex));
}
```

El resultado se vuelve a almacenar en *aux2*, del cual se calcula de nuevo la IFFT y se guarda la actualización a sumar al filtro adaptado en la variable *paso*.

Se normaliza el factor de convergencia μ con la potencia del bloque de entrada actual obteniendo *mu2* y finalmente se actualiza el filtro:

```
for(cont1=0; cont1<W.NumElemComp; cont1++){
W.c[cont1].r+=2*mu2*paso.c[cont1].r;
W.c[cont1].i+=2*mu2*paso.c[cont1].i;
}
```

Tras finalizar todos los bloques los coeficientes del filtro en la frecuencia se transforman al dominio temporal salvando los B primeros valores de cada partición.

Se verifica el correcto funcionamiento utilizando dos señales previamente preparadas en Matlab, una la señal de entrada, el ruido blanco, y otra la respuesta impulsional a identificar. Se exporta el resultado obteniendo el archivo *hest.txt* con este resultado:

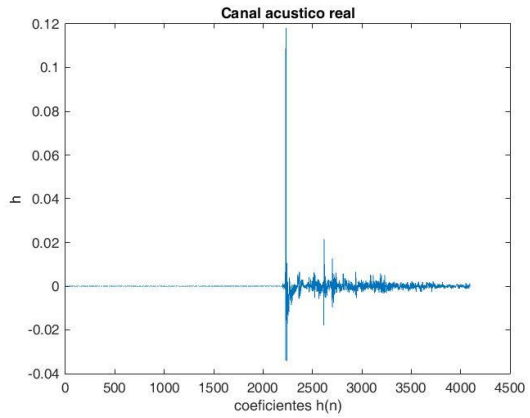


Figura 4. 2: Canal acústico utilizado

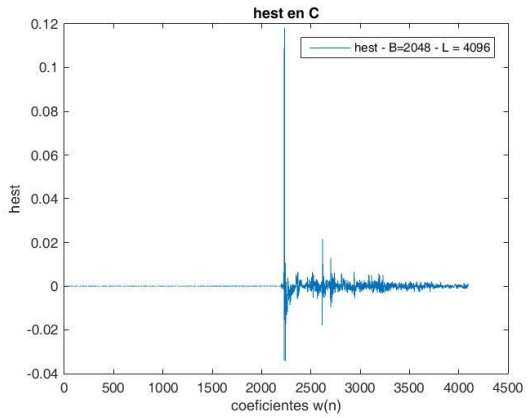


Figura 4. 3: Filtro mimético obtenido

4.1.3.2 Programa en tiempo real con ASIO

Diseñado en C el esquema o estructura del programa se procede a añadir la interacción en tiempo real.

Se incluyen la estructura y funciones necesarias para utilizar los drivers de ASIO comentado todo ello en el punto 4.1.2. Creados los punteros dobles globales se definen número de altavoces y micrófonos a utilizar. Al ser monocanal será un canal de entrada y otro de salida. Por otro lado se especifica el índice *04* del buffer del altavoz en la variable *nAltavoces[0]*.

El flag *bufferFIN* inicializado a uno permitirá la ejecución del algoritmo tras finalizar la comunicación a través de la tarjeta de sonido. Es decir que tras emitirse y recibirse las *ite* muestras útiles del archivo sonoro el programa salta a la ejecución del algoritmo.

```
long glbuffSize = asioDriverInfo.preferredSize;

if (whileFIN){
bufferFin=0;

// captura microfono
memcpy(pM[0],(int*)asioDriverInfo.bufferInfos[0].buffers[index],
glbuffSize*sizeof(int));

//emision altavoz
memcpy((int*)asioDriverInfo.bufferInfos[4].buffers[index],pAltavozBuff[0],glb
uffSize*sizeof(int));

if( ((cont_g+1)*glbuffSize) >= ite ){
       asioDriverInfo.stopped = true;
       ASIOStop();
       } cont_g++; ASIOOutputReady();
whileFIN=1; //Activamos el flag
```

El contador *cont_g* inicializado a cero contabiliza las iteraciones desde la propia función. El tamaño de *glbuffSize* muestras utilizado tanto en el micrófono como en el altavoz corresponde a las B muestras. Tras finalizar activa el flag *BufferFin=1* que da paso a la evaluación del algoritmo con los datos recogidos. Cuando el siguiente bloque supera el máximo de muestras *ite* quiere decir que ya no existen futuras muestras a emitir, por lo que se detienen los drivers ASIO.

```
while(contNBlock <= nBlock){
if (bufferFin){
```

```
        bufferFin=0

… ALGORITMO …

        whileFIN=1;

        } // end if
} // end While
ASIOStop();
```

Deteniendo y cerrando al finalizar la configuración ASIO.

Se exporta los datos obtenidos del filtro transversal *w*, las muestras del error estimado *error* para comprobar su evolución y la salida del canal acústico capturado por el micrófono *micro*. El tiempo total empleado en el procesado *time_procesado* y el tiempo de iteración máximo *time_procesado_max*.

4.1.3.3 Multicanal

Mantiene la misma estructura que el monocanal solo que replicando la ejecución sobre los canales acústicos con la emisión de la señal sonora por el altavoz. El proceso consiste en emitir por cada uno de los *j* altavoces altavoz consecutivamente tras acabar con las muestras del ruido blanco. Se graba por los *K* micrófonos cada iteración. Para realizarlo de esta manera se inicializa una variable *j* a cero de forma que tras el vaciado completo del puntero conteendor del estímulo sonoro (ruido blanco) pase a emitir por el siguiente altavoz. Cundo finaliza de obtener la señal del micrófono por todos los canales mediante ASIO alterna la activación del flag *whileFIN/bufferFIN* dando paso a la etapa de cálculo.

4.1.4 Ecualización monocanal

De manera similar a la explicación del programa encargado de identificar la respuesta del canal se desarrolla a continuación el código empleado para la ecualización haciendo servir igualmente de las cabeceras mentadas en identificación.

4.1.4.1 Programa principal

La ecualización parte del filtro estimado salvado en un archivo de texto *.txt* de *Nh* coeficientes que limita el tamaño del bloque *B* puesto que $B \leq Nh$ determinando el número de particiones P del filtro estimado. El tamaño del filtro ecualizador debe ser $N \geq Nh$ y múltiplo de *B* para tener *F* particiones exactas. El factor de convergencia *mu* se cambia manualmente en el programa. Por último el retardo o *delay* que se ha de añadir a la señal de entrada para obtener la señal deseada tras la ecualización y el canal acústico. El valor analizado en Matlab es *N/2+Nh/2*.

Con la inicialización de variables, punteros, constantes y la transformación de la *hest* al dominio frecuencial particionando en bloques de *B* coeficientes el filtro comienza el algoritmo *filtrado-x* de ecualización diseñado en Matlab.

Calcula la salida de la señal de entrada por el filtro ecualizador a emitir por el altavoz y como le afectaría la respuesta impulsional real del canal a la emisión *in3* obteniendo la salida del sistema global *ystB*. Se le suma en la recepción del micrófono una posible interferencia a activar descomentando manualmente algunas líneas. A continuación se estima el error entre la señal en el micrófono y la deseada. El error será nulo hasta avanzar todo el retardo previo a la señal de entrada, momento en que el error ya distinto de cero actualizará el filtro. Hasta entonces la recepción en el micro será nula o solo la interferencia porque el filtro transversal está inicializado a cero.

Con la entrada del nuevo bloque se actualizan los vectores y matrices de entrada. Como la entrada al filtro estimado debe retrasarse un bloque de muestras para que el proceso converja se utiliza las variables con el sufijo *2* para indicar el retardo con respecto a las que contienen el sufijo *1*, como es el caso de *in2* e *in1* y las siguientes variables a las que modifican (*in1m, in2m, IN1m, IN2m …*)

Por último se pasa a la fase de actualización replicando el error en la frecuencia y calculando la salida *V* por el filtro estimado que se incorpora en la matriz *XPmF* de tamaño *BxF* para incluir el efecto que produce el canal acústico real. Con todo ello se calcula el paso de actualización con la precaución de incluir únicamente las B primeras muestras temporales en el ajuste al filtro ecualizador en la frecuencia.

Finalmente se actualiza el filtro normalizando el factor de convergencia con la potencia de la señal que atraviese el canal estimado, que siguiendo el diagrama de bloques de la Figura 2. 9 sería la señal a la entrada del bloque encargado de ejecutar el algoritmo adaptativo.

4.1.4.2 Programa en tiempo real con ASIO

La diferencia con respecto al de identificación radica en que la actualización del filtro se debe realizar cada iteración de bloque, mientras que anteriormente se han obtenido los todos los bloques de datos previamente al procesado. Esto requiere ir saltando entre la función de ASIO *AsioTime* bufferSwitchTimeinfo* y el hilo de ejecución del algoritmo. Se emplean dos flags para este propósito denominados *bufferFin* que activa la ejecución del algoritmo tras finalizar la comunicación ASIO entre buffers del sistema y el flag *whileFin* que permite actualizar los buffers de ASIO tras la actualización del filtro ecualizador. Cuando se realiza una llamada prohibida a una de estas dos funciones la condición contraria (*else*) al flag se activa desviando la ejecución.

- Función *AsioTime* bufferSwitchTimeinfo*:

```
if (whileFin){
//printf("---DENTRO PARA TX/RX POR MOTU \n");
whileFin=0;

if (((cont_g)*glbuffSize) >= ite){
   asioDriverInfo.stopped = true;
   ASIOStop();
   printf("\nLIMITE MUESTRAS ENTRADA EN BUFFER \n");
}else
processedSamples += glbuffSize;

// captura microfono
memcpy((int*)(pM[0]),(int*)asioDriverInfo.bufferInfos[0].buffers[index],
glbuffSize*sizeof(int));

//emision altavoz
memcpy((int*)asioDriverInfo.bufferInfos[4].buffers[index],(int*)
pAltavozBuff[0],glbuffSize*sizeof(int));

cont_g++;
}else{ //printf("QUIERO ENVIAR Y NO HE ACABADO PROCESADO \n");
   }

if (asioDriverInfo.postOutput)
   ASIOOutputReady();

//printf("------------FIN INTERRUMPO\n");
```

```
bufferFin=1;
```

El procedimiento es desactivar el flag una vez se accede a la interrupción y comprobar si el nuevo bloque supera el máximo de muestras útiles de la señal de entrada, deteniendo los drivers ASIO en tal caso. Si no se transmiten a la tarjeta de sonido y se activa el flag para saltar a la ejecución del algoritmo.

- Función algoritmo adaptativo:

```
while (contNBlock<nBlock){    //printf(" -->> DENTRO WHILE\n");
   if (bufferFin){                  //printf(" -->> PROCESADO\n");
       bufferFin=0;
          … ALGORITMO …

          contNBlock++;
          whileFin=1;          // Interrupción en bufferSwitchTimeinfo

    }                          //if bufferFin
   else{               //printf(" -->>     N O   PROCESADO\n");
   whileFin=1;
   for (int cc=0; cc<200000;cc++){
      kkk=cc+1;} kkk=0;}
}                              //while
```

Como en el caso anterior, tras iniciar la ejecución del nuevo bloque, se desactiva el flag del proceso permitiendo el siguiente salto de interrupción al finalizar el algoritmo. Si se intenta entrar con el flag desactivado se ejecuta un *for* como pausa temporal. Esto solo puede ocurrir cuando los tiempos del programa entre ejecución del algoritmo y la transferencia de datos es inadecuada.

Los punteros con los datos son ahora de *B* muestras, para *i=0,1,2,..,B-1*:

```
pM[0]=(int*)calloc(B,sizeof(int));
ymicro.c[i].r = (double) pM[0][i]/scale;

pAltavozBuff[0] = (int*)calloc(B*As,sizeof(int));
pAltavozBuff[0][i] = (int) scale*yspB.c[i].r;
```

La señal deseada se obtiene retardando el puntero *senyal[xpos]*, con la señal AWGN de entrada, tantas muestras como el valor *delay* sobre el puntero *senyal[xR]* tipo *double*, siendo *xR=1* y *xpos=0*.

```
senyal[xR] = (double *) calloc (ite, sizeof(double));
memcpy(senyal[xR]+delay, senyal[xpos], (ite-delay)*sizeof(double));
```

El valor del retardo *delay* se selecciona para que la convolución entre el filtro ecualizador y el canal acústico generen una delta retardada. Como se analizó con Matlab el valor adecuado es $delay = N/2$. El retardo que permite sincronizar la señal deseada con la recibida y el error puede sumarse a *delay* o implementarse por variables internas a modo de registros.

El algoritmo programado es idéntico al descrito en el apartado anterior y al desarrollado en Matlab. Tras su finalización se exportan los resultados y los punteros con datos interesantes de analizar como las muestras del error y la señal grabada por el micrófono. Tras liberar la memoria de los punteros utilizados se cierra la comunicación y los drivers de ASIO.

4.1.5 Ecualización multicanal

4.1.5.1 Programa principal

Comienza solicitando el número de parámetros a emplear comprobando que sean compatibles puesto que está limitado el número de conexiones según la configuración de la tarjeta de sonido. Se pide la cantidad de entradas *I*, micrófonos *K* y altavoces *J* con los que solicita las fuentes de entrada y los canales estimados correspondientes a los caminos *'jk'*. A continuación se deben introducir los *J·K* canales estimados indicados de mismo tamaño así como las fuentes a emplear. También tamaño de bloque y número de coeficientes del filtro inverso.

Con todo ello pasa a la reserva de memoria inicializando los punteros simples y dobles de tipo *ComplexV* y los punteros-buffers encargados de la transferencia de memoria con la tarjeta en la función de ASIO encargada de ello.

Internamente calcula la señal deseada en cada micrófono como la suma de todas las fuentes en cada micrófono. Puede añadirse un offset de retardo distinto sobre el *delay* si los canales fueran muy distintos.

Tras esto reserva espacio a todos los arrays de datos a emplear como punteros simples o dobles según el caso. Esta reserva requiere de un gran espacio de memoria aumentando en función del número de canales a emplear.

El funcionamiento sigue basándose en las dos etapas claramente diferencias entre ejecución del algoritmo con el *flag bufferFIN* activado y *whileFIN* desactivado y el traspaso de información mediante ASIO con el valor de los *flags* opuestos.

El algoritmo procede almacenando los datos recibidos por la tarjeta sobre punteros identificados por el índice *k*, que hace referencia a cada uno de los micrófonos. Efectúa sobre todas las *k* el error diferencia entre los datos capturados y la señal objetivo relativas a *k*, pasándolo a frecuencia a continuación.

```
for(k=0;k<K;k++){
	for (cont1=0; cont1<B; cont1++){
		e_B[k].c[cont1].r = d_B[k].c[cont1].r - ymicro[k].c[cont1].r ;
	}
}
```

Después calcula el conjunto de señales introducidas como fuentes sobre cada uno de los canales estimados importados al programa con el fin de reproducir la salida que generarían los canales acústicos, guardándola en el puntero `XPmF[i][j+J*k]`.Éste valor es la entrada al algoritmo de ecualización junto al error. Se calcula la correlación entre los *j* bloques de *XPmF* y el error del micrófono *k* al que apuntan acumulando el resultado y finalmente se pone a emplea el método restrictivo anulando con ceros la parte baja de los bloques en tiempo encargados de la actualización.

```
for(i=0;i<I;i++){
	for(k=0;k<K;k++){
		for(j=0;j<J;j++){

// PRODUCTO  IN2m*WH
for(cont1=0; cont1<2*B*P; cont1++){
VmP[i][j+k*J].c[cont1] = ProductoComplejo( IN2m[i].c[cont1],
WH[i][j+k*J].c[cont1]);
		}
	}
}
```

```
for(k=0;k<K;k++){
	for(j=0;j<J;j++){
// SUM(VmP,2) = V
memcpy(V[i][j+k*J].c, VmP[i][j+k*J].c, (2*B)*sizeof(Complex));
if(P>1){
	for(cont2=1; cont2<P; cont2++){
	memcpy(ax2B.c, VmP[i][j+k*J].c+cont2*2*B, (2*B)*sizeof(Complex));
		for(int cont1=0; cont1<2*B; cont1++){
			V[i][j+k*J].c[cont1].r+=ax2B.c[cont1].r;
			V[i][j+k*J].c[cont1].i+=ax2B.c[cont1].i;
				}
			}
		}
	}
}

for(k=0;k<K;k++){
	for(j=0;j<J;j++){
		// Almacenar en matriz XPmF
memcpy(XPmF[i][j+J*k].c+2*B, XPmF[i][j+J*k].c, (2*B*(F-1))*sizeof(Complex));
// Paso los P-1 columnas a la derecha
memcpy(XPmF[i][j+J*k].c, V[i][j+J*k].c, 2*B*sizeof(Complex) );
		}
	}
}
```

Con el cálculo de la μ normalizada se reajustan los coeficientes en frecuencia del filtro para procesar la siguiente salida hacia el altavoz.

```
for(i=0;i<I;i++){
	for(j=0;j<J;j++){
// ACTUALIZACION W
for(cont1=0; cont1<2*B*F; cont1++){
	WF[i][j].c[cont1].r+=2*mu2*PHI[i][j].c[cont1].r;
	WF[i][j].c[cont1].i+=2*mu2*PHI[i][j].c[cont1].i;
		}
	}
}
```

Se adquieren los bloques para la próxima iteración de la señal deseada, bloque de entrada y actualización de matrices de entrada en frecuencia. Finalmente se activa el *flag* correspondiente al intercambio entre *buffers* mediante ASIO.

En la función de ASIO se adquieren las B muestras del bloque de entrada al micrófono almacenado en la tarjeta y se pasan los datos a emitir mediante bucles, puntero de memoria a *buffer*.

Con la finalización de la ecualización y el cese del procesado se termina el programa calculando tiempos de procesados del algoritmo y tiempo total empleado, entre otros, se transforman al dominio temporal los coeficientes del filtro estimado y se exportan, así como otros resultados de interés, a saber señal grabada, señal error, etc. Finalmente se librea memoria y se cierran los drivers ASIO.

Capítulo 5. Mediciones en sala de pruebas

5.1 Entorno para el testeo de la aplicación

5.1.1 Sala y equipos empleados

El sistema de ecualización en tiempo real se ha montado en el laboratorio de pruebas sonoras del iTEAM ubicado en la segunda planta del edificio 8G de la Ciudad Politécnica de la Innovación.

Los altavoces son modelo Event PS6 (Project Studio 6 Biamped System) presenta un rango de frecuencias entre 45Hz – 20kHz, ±3dB, Ref 500Hz. Se emplea la conexión XLR balanceada con una impedancia de 40Ω.

Los micrófonos son de la serie Earthworks QTC (Quiet Time Coherent) con un patrón polar onmidireccional con una respuesta en frecuencia comprendida entre 4Hz a 40kHz ±1dB. Se ha empleado un preamplificador para alimentar la entrada *phantom* de 48V del micrófono.

La CPU es un Intel Xeon X5675(3.07GHz) con 136GB de RAM y la tarjeta de sonido es una MOTU 24I/O con una tasa de muestreo máxima de 96kHz y resolución máxima de 24bits. La comunicación a través de los buffers se realiza con los drivers ASIO (Audio Stream Input/Output).

Los parámetros utilizados han sido una frecuencia de muestreo (f_S) de 44.1kHz y el tamaño de bloque *B* seleccionado en el menú desplegable de MOTU entre 16 y 2048. El valor de *B* describe el número de muestras temporales transferidas por iteración determinando con ello la latencia del algoritmo.

Las entradas o conexiones para micrófonos desde la 1 hasta la 4. Salidas para altavoz activas las 24.

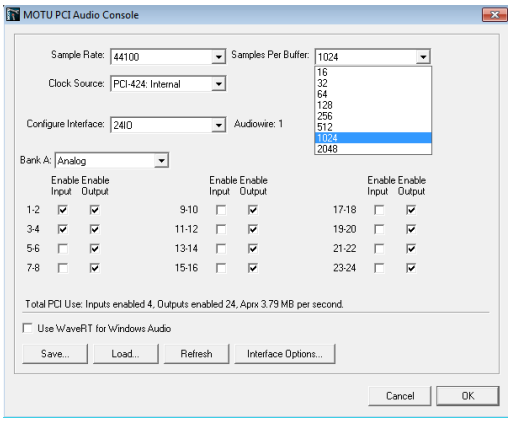


Figura 5. 1: Consola MOTU para fijar número entradas/salidas, frecuencia f_s y tamaño B

5.1.2 Aspectos de implementación en tiempo real

En capítulos anteriores se ha mentado el retardo existente en la comunicación entre buffers. Exactamente la adquisición de datos en tiempo real se transfiere por la conexión PCI entre la tarjeta de sonido y la CPU donde se ejecuta el algoritmo. El llenado del buffer de salida para su posterior emisión por el altavoz se realiza al mismo tiempo que vacio del buffer de entrada del micrófono que es enviado al procesado en la CPU. Esta metodología se repite hasta finalizar el programa.

Para trabajar en tiempo real debe satisfacerse una relación temporal entre la ejecución del algoritmo (t_{proc}) y el tiempo de respuesta dedicado a adquirir los datos de entrada sobre los buffers con valor $t_{buff} = B/f_s$. Para que todas las acciones ocurran dentro del pazo especificado el tiempo de procesado debe ser menor al tiempo de adquisición del nuevo bloque, esto es:

(5. 1)

$$t_{proc} < t_{buff} \quad , \qquad t_{buff} = B/f_s$$

5.2 Resultados experimentales

5.2.1 Mediciones en identificación.

La prueba se ha realizado con un altavoz apuntando directamente al micrófono situado enfrente a una distancia de 27 cm. La identificación estimada del canal intermedio entre los dispositivos sonoros se presenta a continuación. Ésta ha sido obtenida para 4096 muestras empleando diferentes tamaños de buffer *B*.

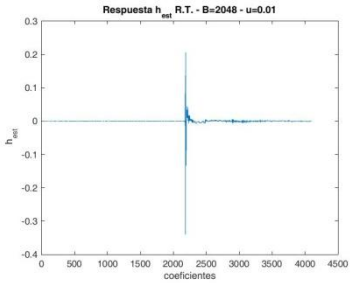


Figura 5. 2: Respuesta B=2048

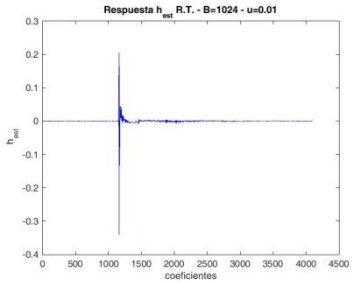


Figura 5. 3: Respuesta B=1024

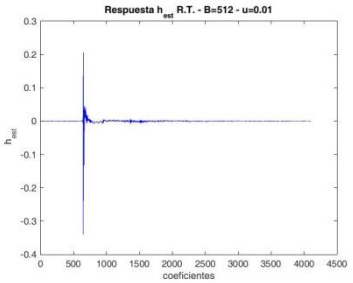


Figura 5. 4: Respuesta B=512

Se comprueba que a la estimación le precede un gap de B muestras nulas. Este desplazamiento o retardo inicial de un bloque en la respuesta se debe al retraso que se produce en el volcado entre la memoria del buffer de la tarjeta de sonido MOTU y el puntero que almacena los datos para que sean procesados Esto obliga a utilizar un tamaño de filtro $N \geq 2B$. Eliminando éste retardo sigue apreciándose una cantidad de muestras, de valor prácticamente cero, hasta alcanzar la componente principal o el impulso directo. En este caso existen unas 138 muestras.

La causa de éste otro retardo viene dada por el desplazamiento físico de la señal sonora. A una velocidad estándar de 340m/s, se extrae rápidamente la relación entre muestras de retardo y distancia entre receptor-transmisor según la frecuencia de muestreo seleccionada:

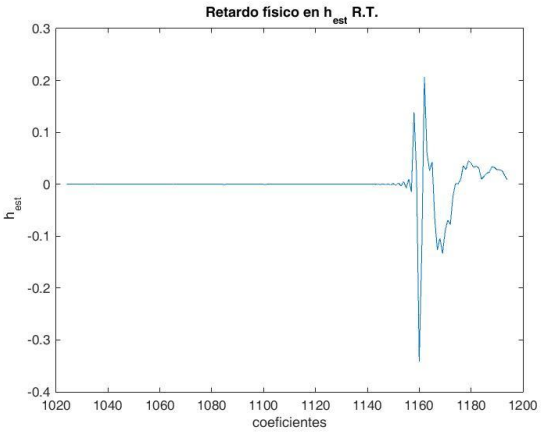


Figura 5. 5: Retardo físico y electrónico.

(5. 2)

$$\frac{f_s}{v_{sonido}} = \frac{44100\ (^{muestras}/_{s})}{340\ (^{m}/_{s})} \approx 1.29706\ (\frac{muestras}{cm})$$

Por tanto la distancia produce un retardo de 35 muestras. Las 103 muestras restantes corresponden a *2.33ms* de latencia. Los drivers ASIO hacen de nexo entre el muestreo en recepción y el procesado del algoritmo, produciendo esto la única latencia electrónica existente por el intercambio de memoria entre la CPU y la tarjeta de sonido así como la conexión con los equipos acústicos. La planificación de procesos del sistema operativo Windows también restringe el tamaño de buffer máximo con el que se puede trabajar puesto que para *B* menores a 256 la emisión suena intermitente, esto es la señal se oirá entrecortada. Para suplir este problema se debe ejecutar el proceso del programa en alta prioridad ganando margen desde las 256 hasta el tamaño de bloque de 64 muestras y se recomienda deshabilitar los recursos y programas que no se vayan a utilizar.

Se han considerado estudiar dos casos diferenciados por su particularidad, el primero es el caso en que la proximidad entre altavoz y micrófono sea cercana logrando canales en los que destaca la componente principal del camino directo y que contienen pocas componentes multicamino relevantes. El caso opuesto es alejar ambos transductores de forma que el canal contenga más contribuciones reflexivas y el camino directo predomine en menor medida.

Comenzando por el caso cercano, se presentan los resultados para diferentes tamaños de bloque y factores de convergencia aplicados para generar una identificación del canal sobre un filtro de 2048 muestras más el bloque inicial nulo.

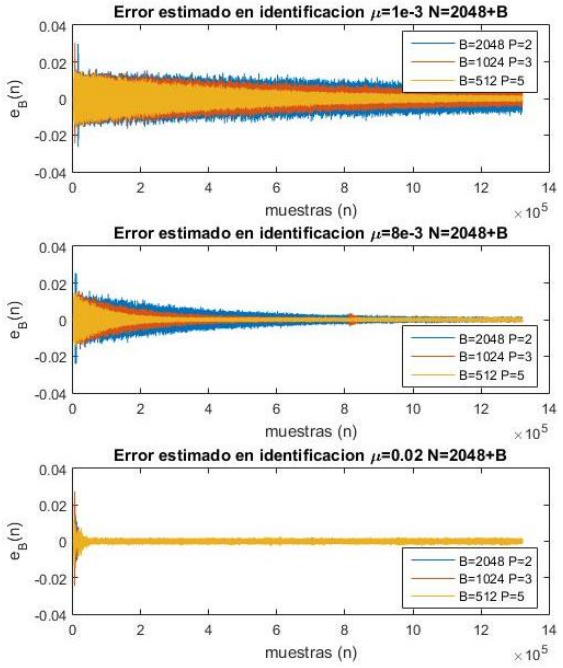


Figura 5. 6: Estimación del error para diferentes tamaños de bloque B y μ

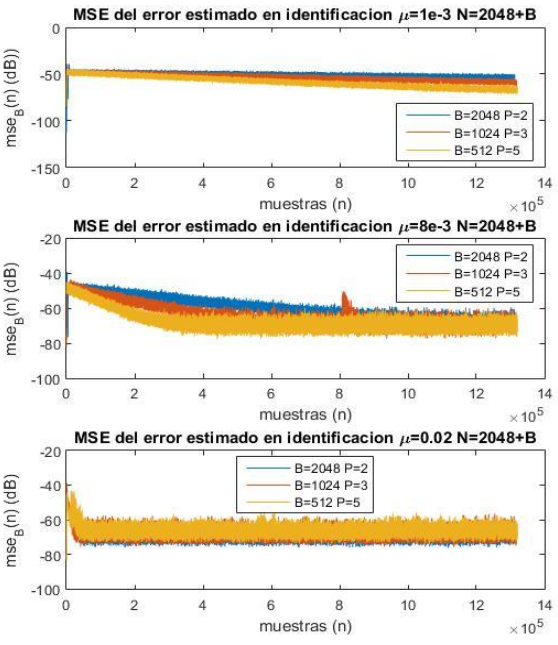


Figura 5. 7: Estimación del error en potencia (mse acumulada) para diferentes B

Observamos que el funcionamiento del algoritmo LMS es correcto puesto que el error tiende a reducirse. Sea cual sea el tamaño del bloque prácticamente todos tienden al mismo residuo final una vez convergen. Para el uso de μ elevadas converge más rápido. Los resultados reflejar que emplear tamaños de bloque pequeños acelera la convergencia. Debido a que estas medidas se han realizado manteniendo fijo el tamaño del filtro útil de 2048 muestras, esto es sin tener en

cuenta el bloque inicial nulo, cada estimación se ha realizado con diferente número de particiones aportando más capacidad de adaptación a unos respecto a otros.

Si comparamos en función de las particiones del filtro encontramos que se confirma el hecho comentado en la Figura 5. 8 Evaluación del error en función de las particiones para B=2048:

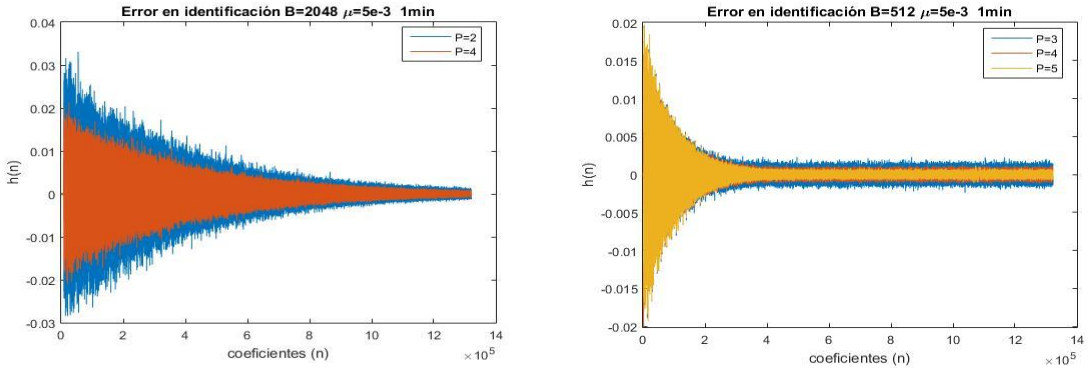


Figura 5. 8 Evaluación del error en función de las particiones para B=2048

El factor de convergencia μ juega un papel relevante tanto en la velocidad como en la fidelidad del resultado. En la Figura 5.9 y Figura 5.10 se aprecia que el residuo de la potencia del error a las que se tiende es irrisoria.

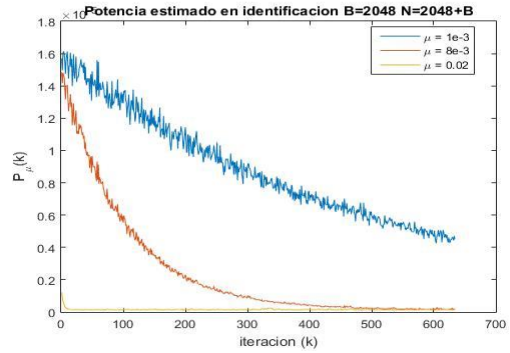


Figura 5.9: Estimación del error en potencia para diferentes μ y B=2048

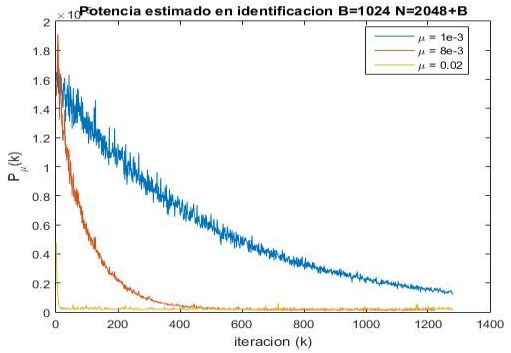


Figura 5.10: Estimación del error en potencia para diferentes μ y B=1024

Si vemos la evolución de la adaptación del filtro con respecto al último obtenido comprobamos cómo para tamaños de bloque pequeños el número de iteraciones es mayor y la bondad de la adaptación en función de μ.

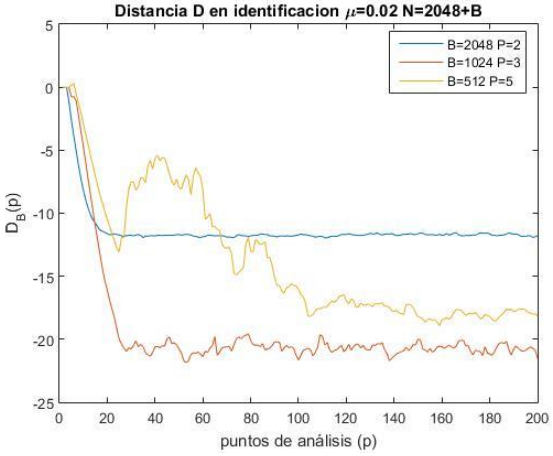


Figura 5. 11 Distancia filtros en identificación μ=0.02

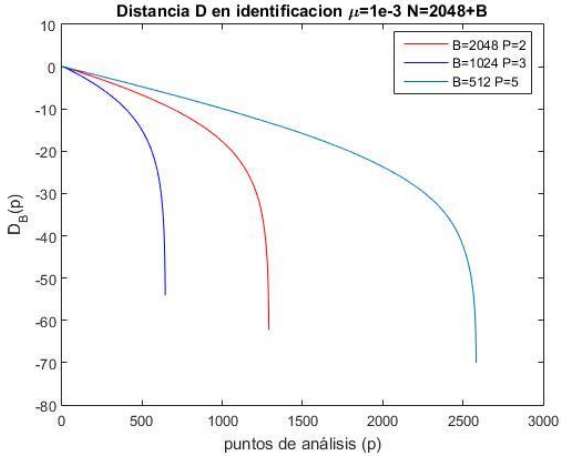


Figura 5. 12 Distancia filtros en identificación μ=1e-3

Según el factor de convergencia y número de particiones se comprueba que la evolución adaptativa varía en mayor o menor medida. Para μ altas la variación del filtro a lo largo de las iteraciones es inicialmente rápida y después no cambia. Con factores de convergencia bajos la adaptación es lenta en todo el proceso pero evoluciona con más precisión.

Para el caso lejano los resultados son similares. Con el fin de mejorar la identificación es preferible emplear un tamaño de bloque o tamaño de filtro suficientemente grande para cubrir la adaptación con las muestras significativas necesarias con el fin de reducir el error diferencia del algoritmo, como se ve en la Figura 5. 14.

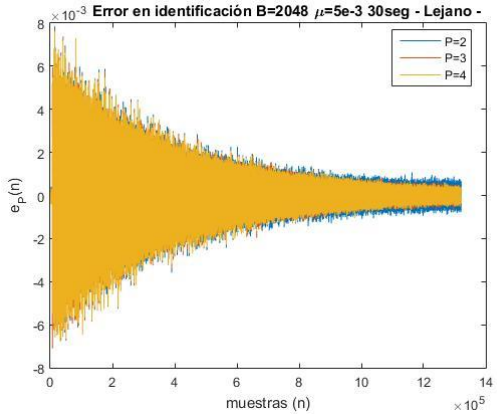


Figura 5. 13 Comparativa del error en caso lejano para B=2048 y diferente número de particiones

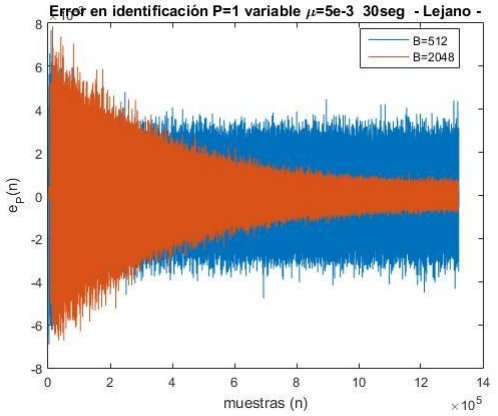


Figura 5. 14 Comparativa del error en caso lejano entre tamaños de bloque para el filtro mínimo

En la Figura 5. 13 vemos de nuevo que, para las mismas circunstancias, cuanto mayor es el número de particiones el error se reduce antes, aunque al ser más complejo el caso lejano la convergencia requiere de más muestras. Aún con ello con 30 segundos de ejecución se logran identificaciones suficientemente bien caracterizadas:

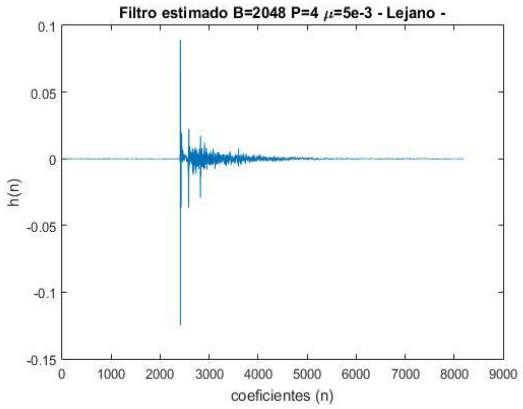


Figura 5. 15 Identificación caso lejano

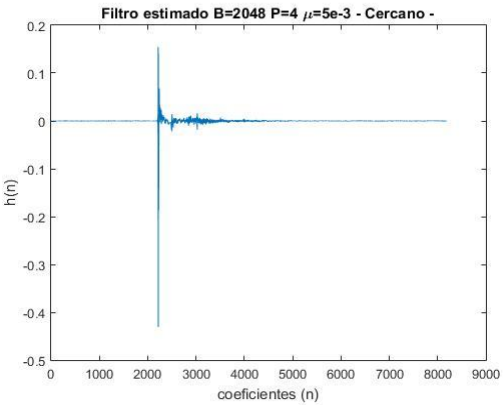


Figura 5. 16 Identificación caso cercano

La respuesta impulsional del canal del caso lejano contiene más reverberación o efectos multicamino puesto que mientras el camino directo se atenúa las contribuciones secundarias se amplían aproximándose temporalmente a la recepción del primer impulso en comparación con el caso cercano de la Figura 5. 16 Identificación caso cercano.

En cuanto al tiempo del procesado del algoritmo se han obtenido los siguientes rendimientos para diferentes tamaños de bloque y la longitud del filtro directamente relacionada con el número de particiones *P* en el procesado:

	P Límite		P=4		P=3		P=2		
Tamaño del bloque B	*P*	t_{proc} *(ms)*	t_{total} *(s)*	t_{proc} *(ms)*	t_{total} *(s)*	t_{proc} *(ms)*	t_{total} *(s)*	t_{proc} *(ms)*	t_{buff} *(ms)*
2048	12	43.05	10.489	16.262	7.466	11.575	5.319	8.247	46.439
1024	14	25.19	9.556	7.402	7.653	5.928	5.094	3.946	23.220
512	14	12.749	9.693	3.753	7.560	2.927	4.830	1.870	11.610
256	14	6.68	9.531	1.845	6.963	1.348	5.182	1.003	5.805

Tabla 5. 1: tiempos de procesado del algoritmo de identificación para diferentes tamaños de bloque

Dado que el mínimo tamaño de filtro en identificación es de *2·B* muestras por el retardo inicial de B coeficientes nulos inherente del sistema, el caso más sencillo será para dos particiones. Se observa que conforme aumenta el tamaño de bloque empleado el tiempo de procesado también aumenta prácticamente de forma proporcional, con lo que el tiempo que el algoritmo emplea globalmente en el procesado es muy parecido.

Para garantizar un correcto funcionamiento se debe asegurar que el tiempo de procesado en cada iteración (t_{proc}) sea menor que el tiempo que necesita la tarjeta para vaciar y rellenar sus buffers. A partir del número de particiones indicado en la columna $P_{Límite}$ de la tabla se llega a apreciar auditivamente la interrupción del sonido emitido por el altavoz.

Por tanto se logra mayor eficiencia seleccionando el tamaño de bloque más grande y un número de particiones suficientemente necesario para que el filtro se adapte más fácilmente y no incluya efectos de ruido en sus extremos. Un mayor tamaño del filtro permite converger antes como se ha visto en la Figura 5. 7. En cuanto al factor de convergencia el sistema de identificación no se ve gravemente afectado por el uso de μ altas.

5.2.2 Mediciones en ecualización.

El algoritmo de ecualización permite obtener la señal deseada en recepción siendo esta la señal de entrada original desplazada por un retardo como respuesta a la combinación de los filtros o canales del sistema en su conjunto. El retado permite caracterizar adecuadamente el filtro ecualizador. Debido al empleo de la tarjeta de sonido como elemento intermediario entre la algorítmica de la CPU y los transductores electroacústicos existen retardos o tiempos del sistema añadidos a las señales empleadas en el algoritmo a tener en cuenta como son los efectos acumulados en la identificación del canal como es el primer bloque de B muestras nulo y el momento en que es procesada la señal recibida por el micrófono.

Esto hace que el funcionamiento del algoritmo de ecualización sea muy sensible a la sincronización entre las señales. Al emplear una señal de ruido blanco, es imprescindible que los bloques procesados y encargados de actualizar el filtro inverso estén perfectamente correlados, puesto que la actualización parte de la diferencia entre bloques. De lo contrario el error estimado no disminuirá y el resultado divergirá con total seguridad.

Por tanto, debido a los retardos inherentes de los equipos acústicos y el espacio que recorre la señal, como ya se ha explicado, y la propia estructura que sigue el código programado se hace necesario aplicar cuantas modificaciones sean necesarias en las diferentes etapas o pasos de procesado en la ejecución del algoritmo para solventar posibles inconvenientes.

El objetivo es garantizar que, por un lado, la señal deseada *d* y la obtenida en el buffer del micrófono *z* estén correladas respecto a *x* así como, el error resultante *e*, también lo esté con la salida del canal estimado x_f siguiendo el esquema de la Figura 3. 34.

Matemáticamente, el flujo de datos debe satisfacer:

$$d(n) = x(n - delay - \tau_2) \quad \textbf{(5. 3)}$$

$$z(n) = x(n) * w_{inv_{ideal}}(n - delay) * h(n) * \delta_{MOTU+fisico}(n - \tau_2) \quad \textbf{(5. 4)}$$

$$x_f(n) = x(n) * h_{est_{ideal}}(n - \tau_2) * \delta(n - delay) \quad \textbf{(5. 5)}$$

Siendo *delay* igual a *N/2* para desplazar el filtro inverso y centrarlo de tal modo que actualice sus coeficientes no-causales más las muestras τ_2 necesarias para sincronizar la señal deseada con la señal recibida por el micrófono. τ_2 comprenderá el retardo de B muestras de la h_{est}, el retardo tanto físico como eléctrico y se deberá tener en cuenta que el algoritmo arranca sin muestras. La señal *z(n)* sufre todos los retardos entre el canal acústico y del filtro ecualizador al ser resultado de ambos. La señal x_f sufre por definición, al emplearse directamente el canal estimado con el bloque nulo inicial, un retardo τ_2, el retardo de la h_{est} , por lo que habrá que forzar un retardo de *delay* muestras empleando la señal en instantes posteriores. A la señal deseada habrá que forzar que tras τ_2 muestras y los tiempos de arranque comience el *delay.*

En la primera implementación del código, que seguía la estructura desarrollada en Matlab, se obtuvieron resultados incorrectos y defectuosos debido a la incorrelación entre las señales. Aunque el filtro inverso modelaba una respuesta aparentemente similar a la espera por simulación, no alcanzaba una delta unidad quedándose en torno a la mitad. Y se hacía imposible evaluar el sistema en el tiempo puesto que el error diferencia entre la señal deseada y la entrada del micrófono en vez de disminuir, aumentaba, a causa de que la seña recibida estaba siendo amplificada. Se muestran en las siguientes figuras el filtro inverso con su respectiva delta por convolución con el canal estimado:

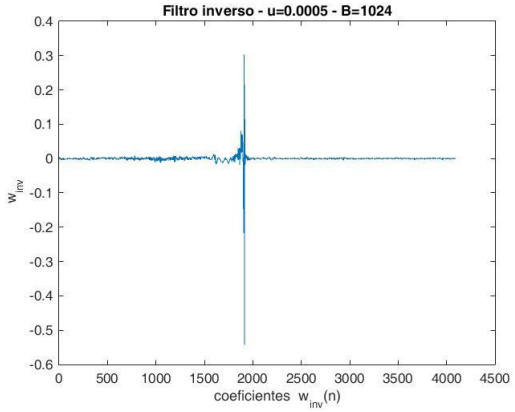


Figura 5. 17: Filtro inverso

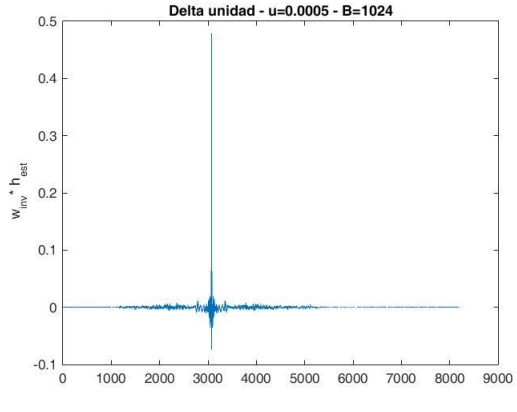


Figura 5. 18: Delta resultante por convolución.

Utilizando el mismo valor de B tanto para la identificación como para la ecualización con el fin de que el procesado sea compatible, en todos los casos se obtiene la misma respuesta. En la Figura 5. 17 se aprecia que el filtro es un no-causal desplazado un *delay* de *2·B* muestras menos el retardo eléctrico y físico (~138 muestras). La delta es el resultado de la convolución del filtro con la identificación estimada de canal. Se visualiza que está ubicado en torno a 3000 debido al retardo por el *delay* del filtro inverso y las *B* muestras de la estimación del canal (h_{est}). El retardo eléctrico y físico se compensado en el ecualizador.

Sin embargo, la disposición del código no lograba la sincronía necesaria entre bloques para reducir el error. Una posible salida generada del algoritmo era:

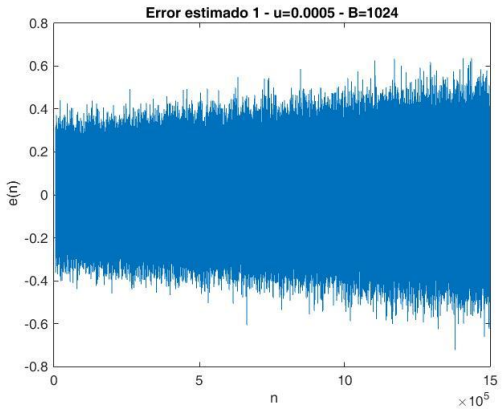


Figura 5. 19: Error estimado tipo 1

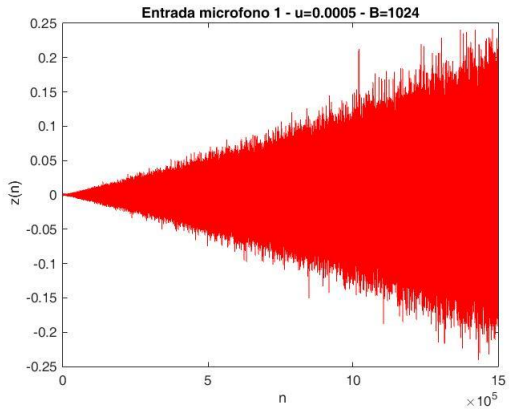


Figura 5. 20: Señal obtenida en micrófono

En la que se aprecia claramente la señal a la entrada del micrófono, teóricamente la deseada, incrementándose a lo largo del proceso al no estar adecuadamente dispuestas las etapas del procesado y relacionar las señales en éste incorrectamente. Esto dio paso a una sucesión de pruebas y modificaciones en las que se analizaban las señales exportadas del programa y se calculaba la correlación entre el error, la señal del micrófono recibida, la respuesta a la salida del filtro del canal estimado y tanto la señal deseada como la propia señal de entrada haciendo uso la función *xcorr* de Matlab, a partir de lo cual se reajustaba de nuevo el código.

La relación temporal entre las señales finalmente ha sido para un tamaño de bloque B=2048, filtro estimado de 3 particiones en total y un filtro ecualizador de 2 particiones:

Parámetro 1	Parámetro 2	Retardo (muestras)		Parámetro 1	Parámetro 2	Retardo (muestras)
Señal entrada	Señal deseada	3B		Señal entrada	Señal recibida	4B
Señal entrada (filtrada por h)	xf1	2B		Señal entrada (filtrada por h)	Señal recibida	2B+1736
Señal deseada	Señal error	B		Señal entrada (filtrada por h)	Señal deseada	B+1736
Señal entrada	Señal entrada (filtrada por w)	N/2 - 312		Señal entrada	Señal entrada (filtrada por h)	B+312

Tabla 5. 2 Relación temporal entre señales

Adecuando la estructura del programa para su mejor funcionamiento y tras solucionar los problemas en la implementación inicial, se pasa a continuación a analizar los resultados finalmente obtenidos.

Atendiendo al caso en el que los equipos de sonido están próximos[4] encontramos en primer lugar, tal y como cabía esperar, que el error del sistema disminuye. Lo comprobamos para dos tamaños de bloque distintos, 2048 y 1024:

[4] Distancias: Centro del altavoz al micrófono=0.2175m; Tweeter del altavoz al micrófono=0.27m; Micrófono respecto del suelo=1.26m; Centro del altavoz respecto del suelo=1.285; Base del altavoz respecto del suelo=1.148m; Micrófono situado de frente al altavoz.

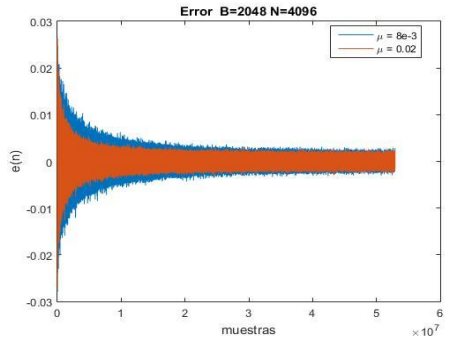


Figura 5. 21 Error para B=2048

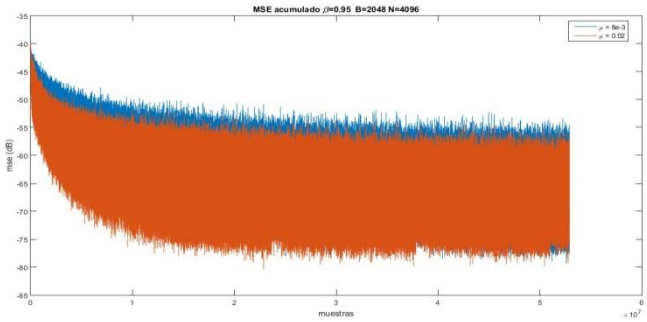


Figura 5. 22 Mse acumulado para B=2048

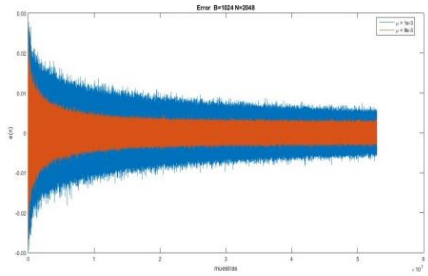


Figura 5. 23 Error para B=1024

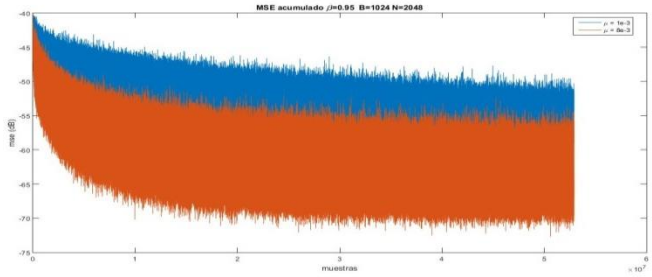


Figura 5. 24 Mse acumulado para B=1024

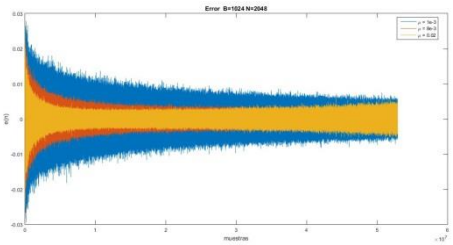


Figura 5. 25 Error divergente para B=1024

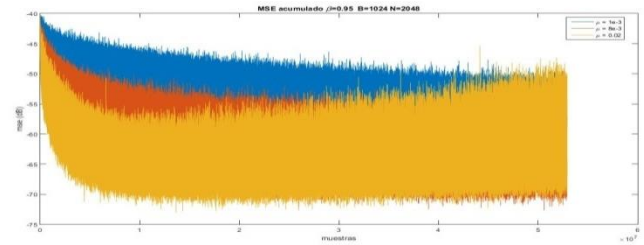


Figura 5. 26 Mse acumulado para B=1024 caso divergencia

Observando el error *mse* acumulado[5] respecto un factor *β=0.95* es posible distinguir la potencia de error a la que tiende el algoritmo; Con *B=2048* el error acaba rondando los -68dB en la Figura 5. 22, presentando una varianza $\sigma_{\mu=8\cdot10^{-3}}$ de $3.8 \cdot 10^{-7}$frente a $\sigma_{\mu=0.02}$ igual a $2.86 \cdot 10^{-7}$ considerando el último quinto de muestras. Con la mitad del bloque, *B=1024* y $\mu = 8 \cdot 10^{-3}$ presenta una varianza de $5.5 \cdot 10^{-7}$ rondando los -64dB de potencia *mse* acumulada. Con ajustes pequeños como $\mu = 1 \cdot 10^{-3}$ el *mse* alcanza los -58dB, puesto que tarda más en converger.

El empleo de pasos de convergencia elevados hace que el sistema alcance rápidamente el límite del error. En cambio el uso de valores de μ pequeños hace que la convergencia requiera más tiempo aportando como ventaja que la oscilación por varianza de ruido sea menor. El tamaño del bloque influye en cuanto al número de muestras procesadas por iteraciones.

En las Figura 5. 25 y Figura 5. 26 se observa que el error comienza reduciéndose antes que los casos precedentes pero, sin embargo, el valor del paso elevado (μ=0.02) hace divergir el sistema incrementando el error, efecto que se ve más claramente comprobando la evolución de *mse.*

Luego el error que el sistema tiene que procesar se mantiene estable para factores de convergencia bajos, mientras que si el paso es elevado el error acaba divergiendo. Vemos a continuación la distancia según (3. 2). A partir de más de 5000 respuestas del filtro ecualizador equiespaciadas a lo largo de la duración del programa se compara la convolución con el canal estimado frente a una delta ideal en la Figura 5. 27 Distancia en ecualización en función de B y μ.

[5] $mse(n) = \beta \cdot mse(n-1) + (1-\beta) \cdot e(n)^2$

Destaca principalmente la progresión para μ altas. Tras alcanzar un máximo de ecualización la medición de la distancia se perturba y toma un carácter creciente frente a los factores μ de bajo valor.

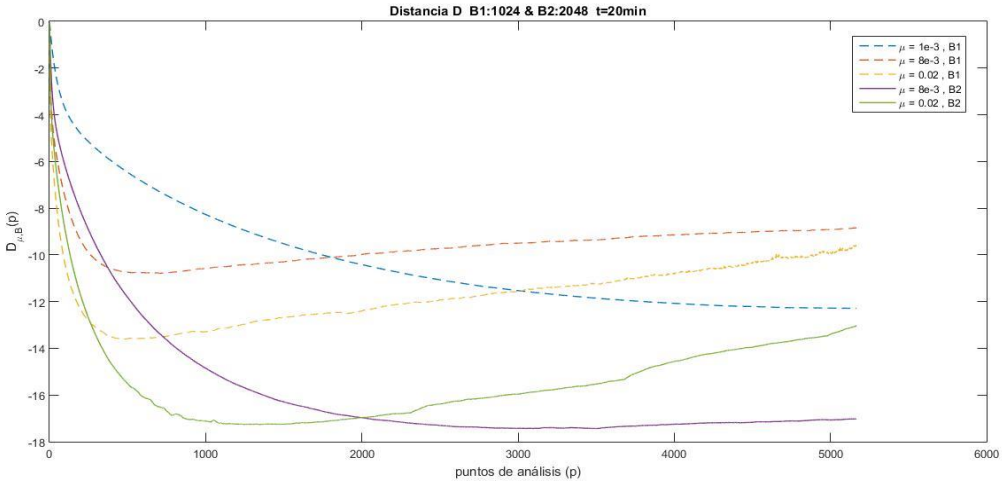


Figura 5. 27 Distancia en ecualización en función de B y μ

Pasamos a analizar cómo influye el tamaño del filtro empleado. Se estudiará para dos tamaños de bloque con diferentes mues, concretamente *B1=2048* y $\mu_{B1} = 0.02$ así como *B2=1024* y $\mu_{B2} = 8 \cdot 10^{-3}$. A la derecha se muestran los 3 filtros ecualizadores resultantes para *B1*, cada uno con diferente número de coeficientes, con *N=B1*, *N=2·B1* y *N=4·B1*.

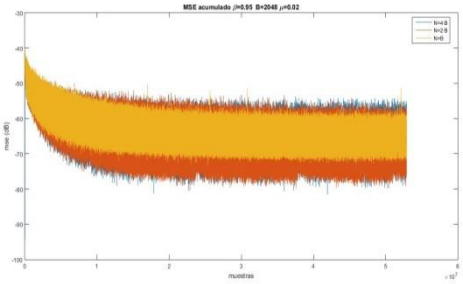


Figura 5. 28 Mse diferentes tamaños de filtro B=2048 y μ=0.02

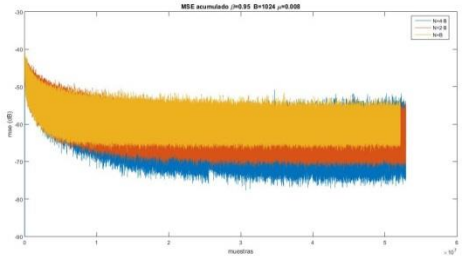


Figura 5. 30 Mse diferentes tamaños de filtro B=1024 y μ=8e-3

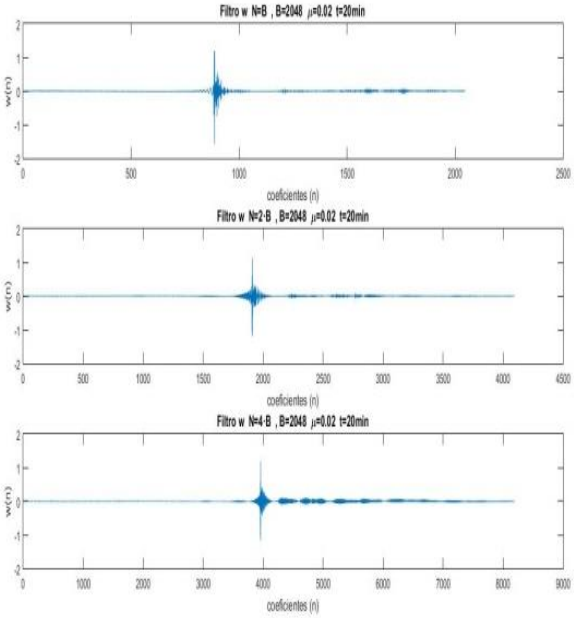


Figura 5. 29 Filtros del tamaño analizado

Es preferible reducir el tamaño del filtro para evitar acumular efectos ruidosos en los extremos de las componentes principales.

Se centra el análisis en dos tamaños de bloque con diferentes mues, concretamente *B1=2048* evaluado para $\mu_{B1} = 0.02$ y *B2=1024* con $\mu_{B2} = 8 \cdot 10^{-3}$. A la derecha, Figura 5. 29 Filtros del tamaño analizado, se muestran los 3 filtros ecualizadores resultantes para *B1*, cada uno con diferente número de coeficientes.

Para tamaños del filtro grandes el error contiene más varianza oscilando entre un margen mayor respecto al filtro de tamaño *B*. Esto se relaciona con el ruido de los coeficientes del filtro. El algoritmo basa el proceso de ajuste adaptativo en el error. El error está limitado siendo imposible que el filtro alcance los valores óptimos ya que los coeficientes oscilan con la energía del ruido. Cuando mayor sea el filtro más ruido acumula entre sus coeficientes. Esto explica que el cálculo de la distancia se desvirtúe ya que la suma de todas las componentes ruidosas altera el resultado.

El empleo de tamaños de bloque pequeños para filtros grandes hace que aumente el desajuste del filtro, puesto que divergen antes. Se refleja este comportamiento en la gráfica siguiente, en la que conforme aumenta el tamaño del filtro la respuesta diverge más tempranamente:

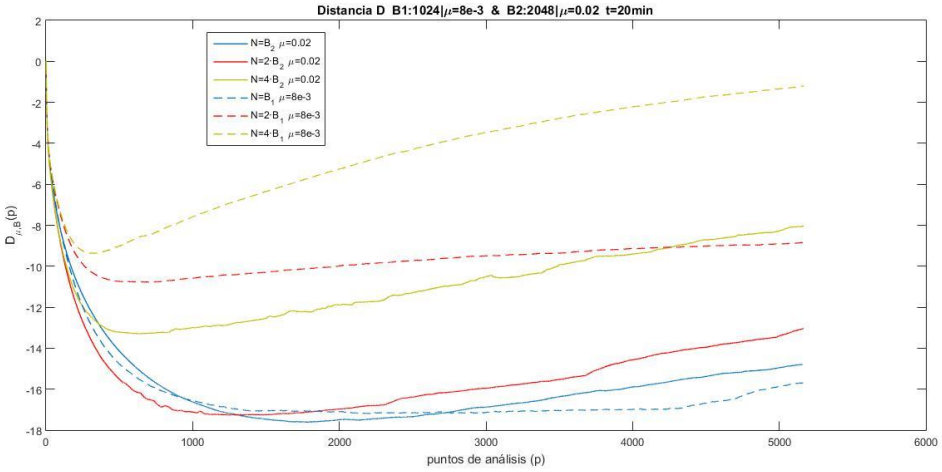


Figura 5. 31 Distancia en función del tamaño del filtro para varios B y μ

Otro método para comprobar el efecto de los coeficientes afectados por el ruido de varianza es eliminar del filtro los términos que están añadiendo la interferencia o perturbación y dejar aquellos que compensan el canal acústico.

Partiendo de un filtro de 4096 valores, obtenido para $\mu_{B\_1024} = 8 \cdot 10^{-3}$, con alto contenido ruidoso en sus coeficientes laterales y en un estado en que el valor de distancia D generado está tendiendo a los 0dB como se muestran en las figuras Figura 5.32 y Figura 5. 35, si se eliminan los primeros 1730 así como los 1940 últimos valores con un tránsito exponencial[6] , obteniendo el filtro de la Figura 5.33 se mantienen los coeficientes principales de la ecualización puesto que en la evolución de los coeficientes se ha realizado la misma operación de truncado, ver en la **¡Error! No se encuentra el origen de la referencia.**, y se observa que no ha variado en gran edida. La diferencia se presenta en la **¡Error! No se encuentra el origen de la referencia.**. Con ello se obtiene como conclusión que el filtro original contenía la suficiente información para mantener su nivel ecualización en -9dB visto en la Figura 5. 35 pero el ruido en los coeficientes laterales distorsiona el resultado.

[6] w_truncado= [(w(1:1730).*($e^{(-(1730-1)/2:1/2:0)'}$));w(1730+1:1940-1);w(1940:end).*($e^{0:-1/2:-(\text{length}(w)-1940)/2)'}$)];

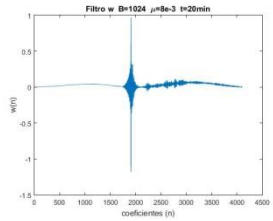


Figura 5.32 Filtro original

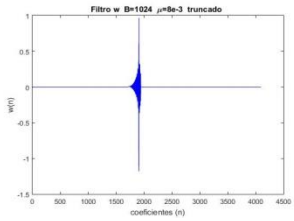


Figura 5.33 Filtro truncado

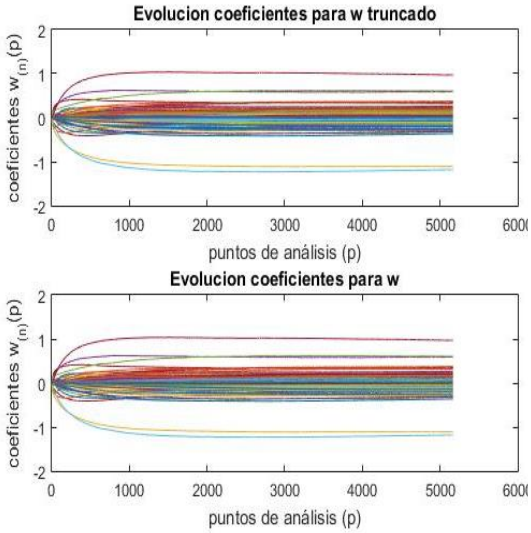


Figura 5. 34 Evolución temporal coeficientes

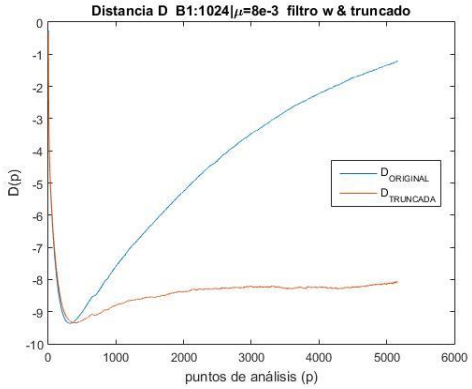


Figura 5. 35 Distancia filtro truncado y original

También es interesante comprobar como la delta unidad que genera el filtro ruidoso existe aunque parezca que el cálculo de la distancia sea incorrecto:

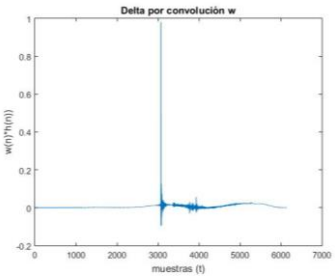


Figura 5. 36 Convolución del filtro original

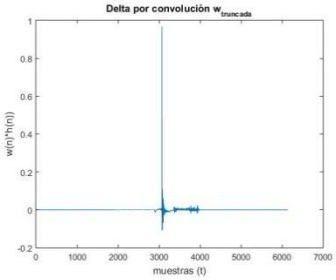


Figura 5. 37 Convolución del filtro truncado

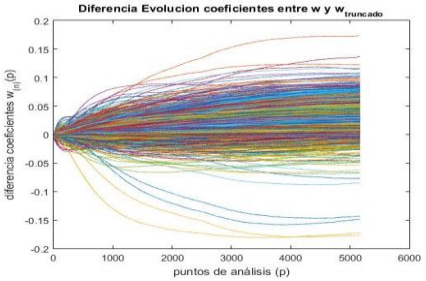


Figura 5. 38 Diferencia entre evoluciones del filtro

Recortando el filtro, es decir eliminando el ruido de los laterales, es posible evitar la oscilación que toma la respuesta así como mejorar el comportamiento del ecualizador. Puede comprobarse la cantidad de elementos con interferentes acumulados en el filtro inicial que aportan una gran energía en global.

El error entre la señal deseada y la recibida en el micrófono se mantiene constante a lo largo del proceso:

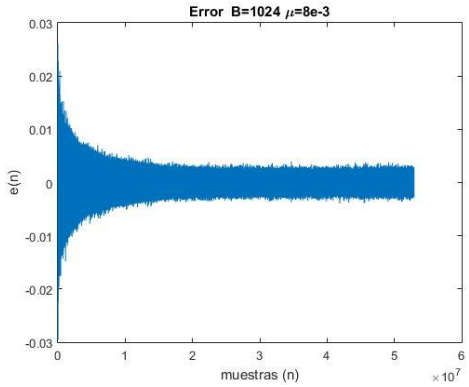


Figura 5. 39 Error en la obtención del filtro original

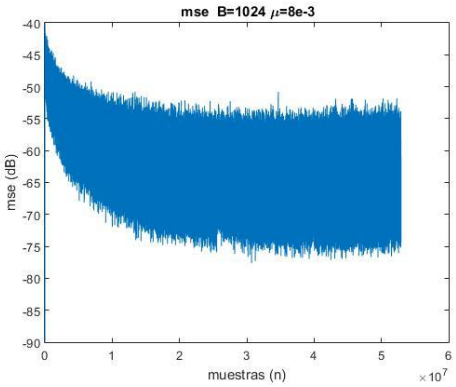


Figura 5. 40 Mse del error del filtro original

Aunque con el deterioro del filtro la respuesta emitida se amplificaría y finalmente el sistema divergiría puesto que el error tendería a aumentar.

El efecto más limitante es el ruido del filtro. Podrían aplicarse técnicas para reducirlo como enventanar la actualización del filtro con el fin de destacar las componentes importantes o emplear una normalización vectorial actuando sobre cada coeficiente con un valor de normalización en función de su potencia.

Recapitulando sobre los aspectos a tener en cuenta en el sistema encontramos como parámetros principales:

- Tamaño del bloque (B): Está comprendido como potencia de 2 entre 32 hasta 2048. Cuanto mayor sea más rápido convergerá el algoritmo realizando menor número de iteraciones. El filtro estimado empleado debe iniciar sus coeficientes nulos por el mismo número que el tamaño de bloque a utilizar, en cambio los sucesivos pueden diferir.
- Factor de convergencia (μ): Condiciona la velocidad de convergencia así como la varianza del error resultante. Cuanto menor sea más fina es la actualización de los coeficientes logrando mejores resultados y evitando que el ruido afecte drásticamente sobre los coeficientes del filtro, sin embargo más muestran se deben emplear para obtener un buen resultado.
- Tamaño del filtro (N): Cuanto mayor sea más rangos de libertad tendrá el filtro para adaptarse si bien en exceso producirá que los coeficientes laterales incrementen su ruido de varianza y la respuesta del filtro se vea distorsionada.
- Tiempo de convergencia: Dependiendo de la aplicación el tiempo ecualizando puede ser variable. Si únicamente se desea caracterizar los filtros para su uso posterior se puede emplear convergencias rápidas procurando no extender en exceso el proceso con el fin de evitar un incremento del ruido que deteriore el filtro. De lo contrario se pueden usar factores de convergencia bajos y preferiblemente tamaños de filtro reducidos para evitar efectos ruidosos indeseados.

A pesar de las limitaciones, el sistema funciona correctamente y muestra de ello es que el error entre señal recibida y deseada disminuye, la convolución del filtro ecualizador y el canal estimado generan una delta próxima a la unidad. Se puede comprobar la capacidad de regenerar una señal acústica filtrando un sonido por los datos obtenidos para *B=2048* y *μ=8e-3*:

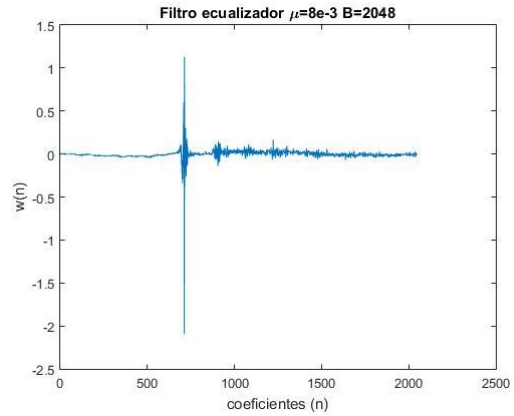


Figura 5. 41 Filtro ecualizador

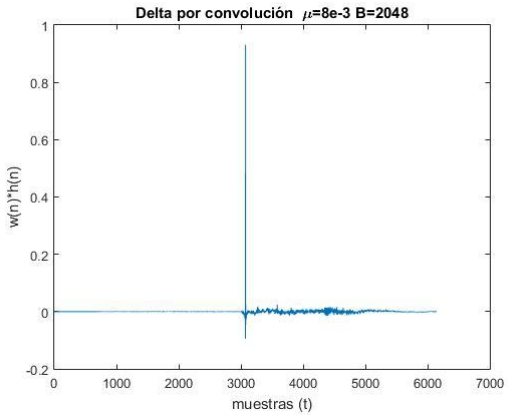


Figura 5. 42 Convolución respuesta sistema

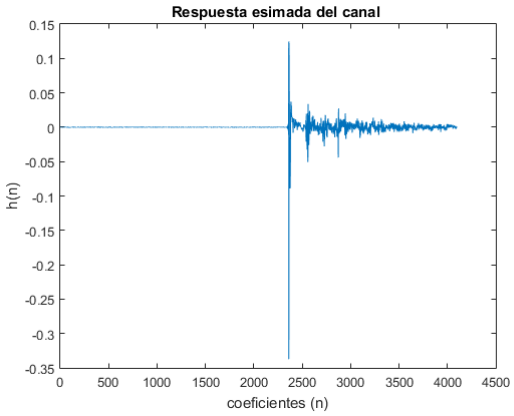


Figura 5. 43 Canal estimado

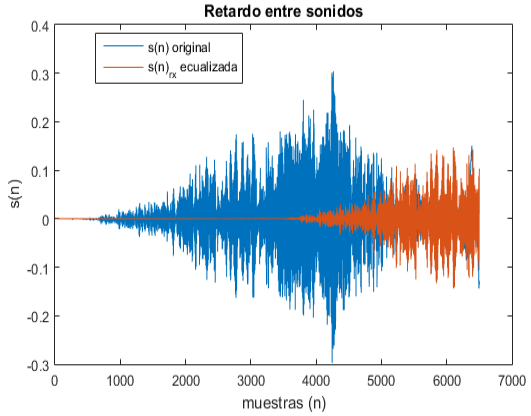


Figura 5. 44 Retardo señal recibida y original

Se utiliza un filtro de corta longitud para evitar el efecto del ruido acumulado. La delta que genera es bastante buena pese la varianza que acumula en el extremo derecho. El sistema genera un retardo desde que comienza hasta que se recibe la señal, en este caso de 3073 muestras, 662 del ecualizador, 2048 del bloque nulo del canal estimado que en un caso real no se tendría en cuenta y 361 de latencia.

En la Figura 5. 45 se muestra la evolución de la forma de un sonido al ser tratado por los filtros. En ella se aprecia que el canal modificaría algunos aspectos temporales de la señal y prefiltrando el audio por ecualizador logra asemejar la envolvente del sonido.

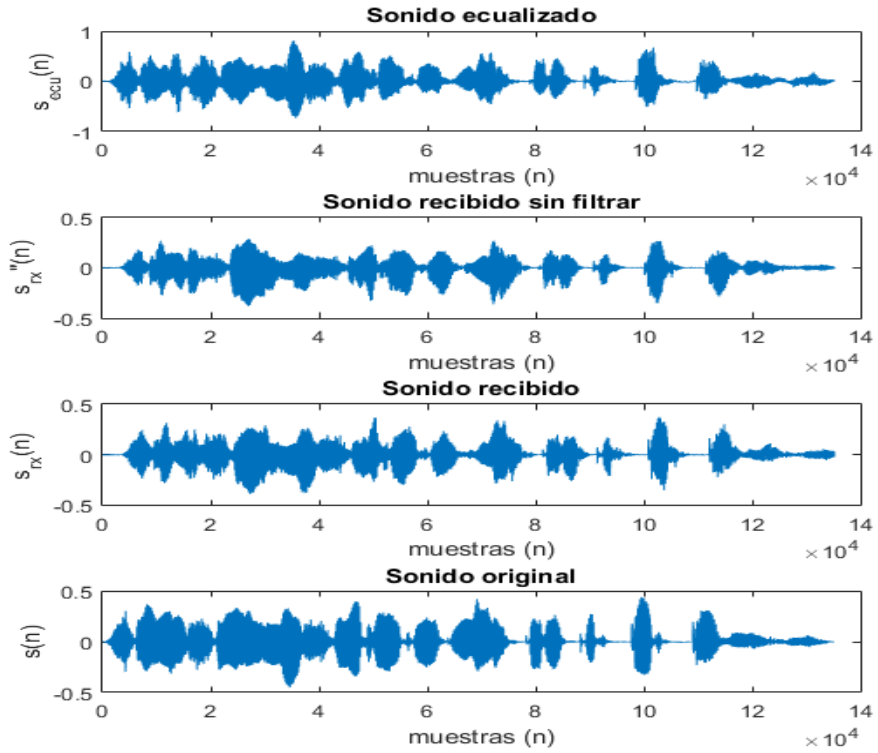
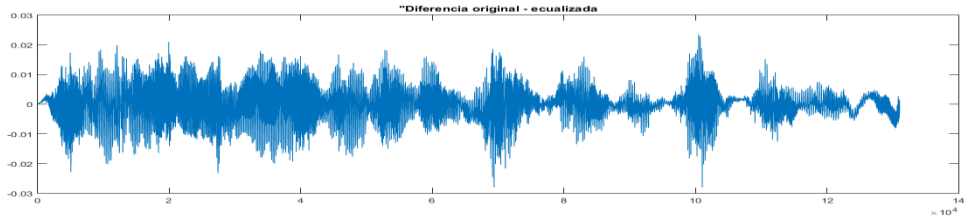


Figura 5. 45Ejemplo sonido ecualizado y filtrado

Siendo la diferencia entre la original y la recibida de 0.02 respecto a los casi 0.5:



Por otro lado se muestra a continuación la capacidad del algoritmo de adaptarse a cambios. Cada tramo corresponde a 1 minuto y medio. Comenzando la ecualización con el canal estimado y para una μ de 1e-3 el ecualizador se reajusta tendiendo a una distancia D de -10dB

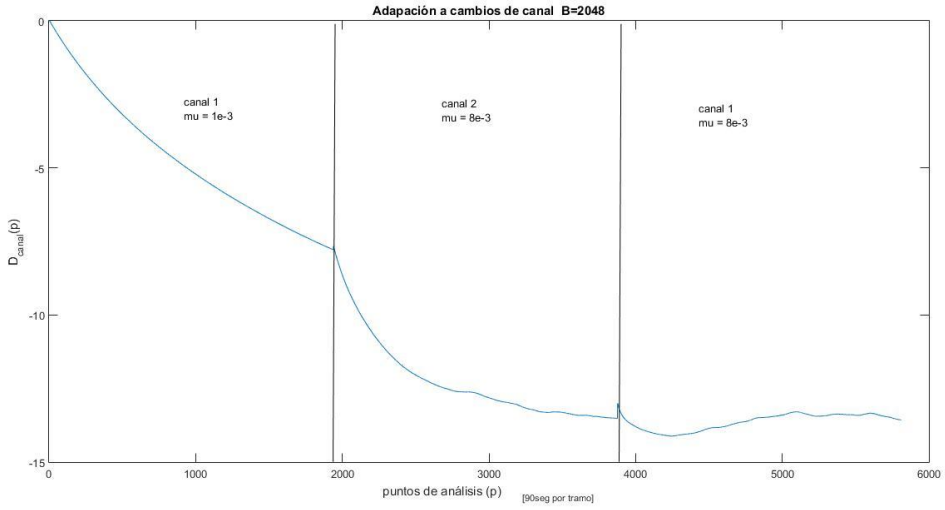


Figura 5. 46 Adaptación del filtro ecualizador frente a cambios de entorno

Con los coeficientes obtenidos se modifica el canal y aplicando la misma identificación inicial se ejecuta el algoritmo para una μ algo más alta de 8e-3. Con esta nueva configuración el filtro vuelve a configurar sus coeficientes logrando una distancia de en torno -13dB. Finalmente regresando a la posición inicial vemos que el filtro intenta mantener el nivel. En la Figura 5. 47 se observa el ajuste final en cada tramo. Concretamente en el último comienzan a surgir componentes ruidosas lo que puede dar lugar a equivoco en una estimación de diferencia entre la convolución y la delta unidad ideal.

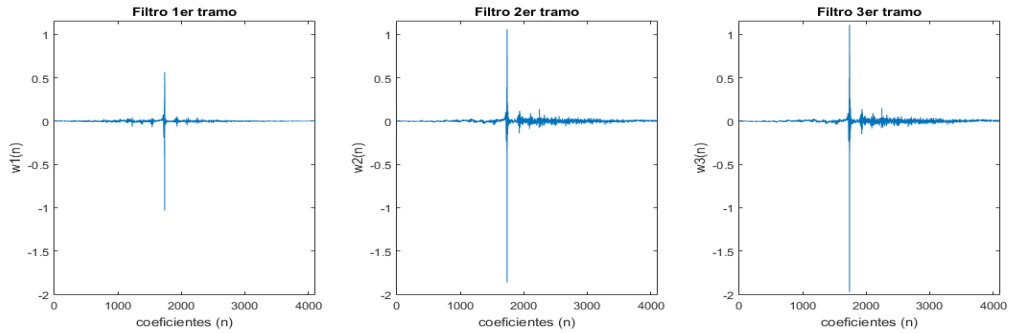


Figura 5. 47 Filtros finales obtenidos en cada tramo

Se muestra otro ejemplo de adopción con tramos de 3minutos:

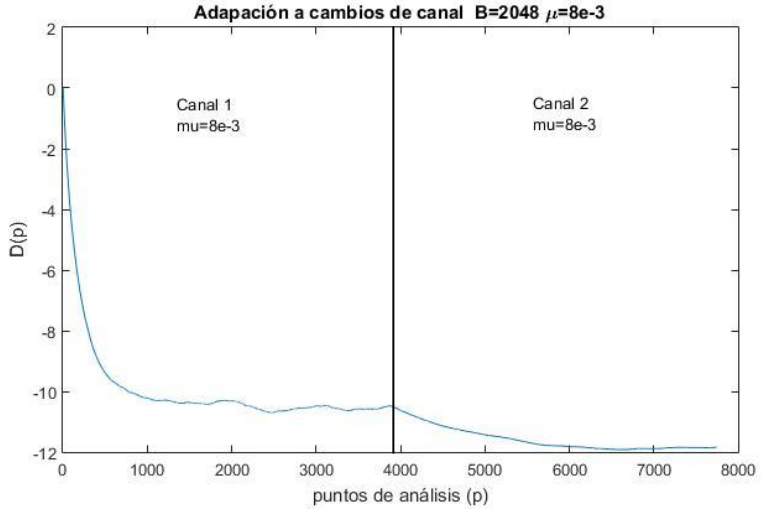


Figura 5. 48 Adaptación del algoritmo a cambios de entorno

A continuación veremos el comportamiento a distancias más largas de 1 metro, concretamente a una distancia entre el altavoz y el micrófono de 1.5m:

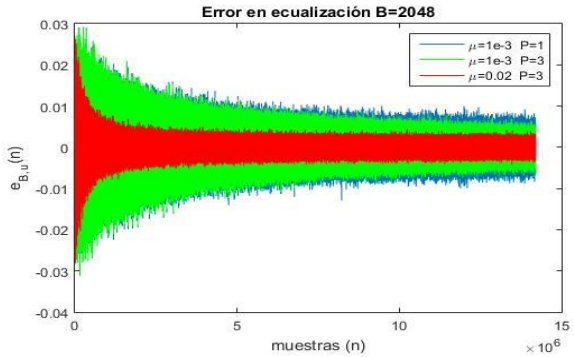


Figura 5. 49 Error diferentes parámetros caso lejano

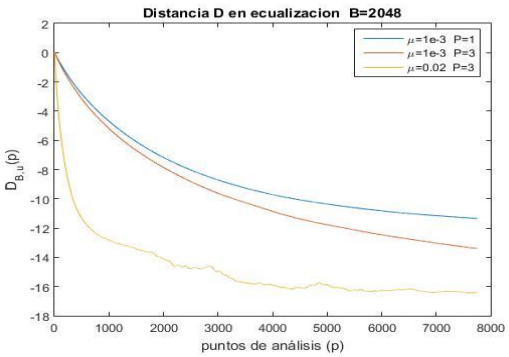


Figura 5. 50 Distancias caso lejos para 6min.

Exactamente igual que como el algoritmo se comporta para distancias cortas. El error disminuye más rápidamente para factores de convergencias mayores a costa de incrementar la varianza de ruido en la adaptación (Figura 5. 50 Distancias caso lejos para 6min.Figura 5. 50). Conforme se emplean más coeficientes también mejora la rapidez del sistema. La ecualización genera un filtro de este tipo:

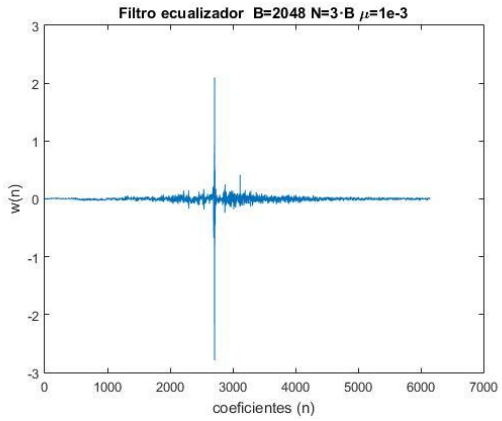


Figura 5. 51 Filtro inverso

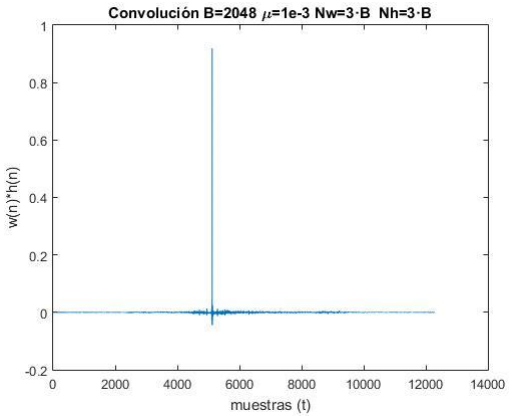
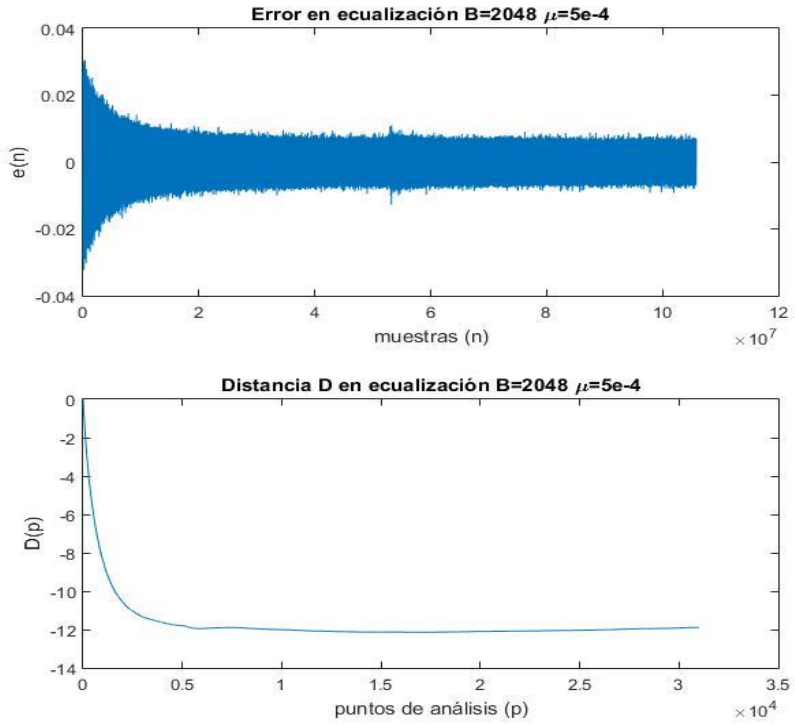


Figura 5. 52 Convolución resultante

La respuesta global del sistema acaba siendo bastante limpia.

El sistema tiene un buen comportamiento para tiempos prolongados. Se muestra la ecualización con mínimo tamaño de filtro para minimizar efectos de ruido sobre los coeficientes laterales para un canal lejano. La prueba es de 40minutos empleando *B=2048* y μ=5e-4 :



El error no disminuye demasiado debido al reducido número de coeficientes llegando a una diferencia respecto a la delta ideal de -12dB. La ventaja de usar μ bajas es lograr mantener la estabilidad del filtro.

En cuanto a la importancia entre los tiempos de procesado y tiempos de buffer comentado en 5.1.2 la siguiente tabla permite extraer la limitación existente en el programa. Se ha realizado manteniendo el mismo número de particiones en el cálculo de la identificación estimada de canal, es decir el tamaño del filtro estimado de 2B + B nulos iniciales:

Tamaño buffer (B)		$t_{buff}\ (ms)$	$t_{proc}\ (ms)$	$t_{buff} + t_{proc}\ (ms)$	*Ratio Muestras/seg.*
2048	F=1	46.439	9.117	55.556	36.864e3
	F=2		12.951	59.39	34.484e3
	F=3		17.018	63.457	32.273e3
	F_{max}=12		48.342		
1024	F=1	23.220	4.228	27.448	37.306e3
	F=2		5.904	29.124	35.160e3
	F=3		7.545	30.765	33.284e3
	F_{max}=12		23.647		
512	F=1	11.610	2.051	13.661	37.478e3
	F=2		2.895	14.505	35.298e3
	F=3		3.592	15.202	33.679e3
	F_{max}=12		11.803		
256	F=2	5.805	1.403	7.08	36.158e3
	F=4		2.161	7.966	32.136e3
	F=8		3.569	9.374	27.309e3
	F_{max}=12		6.446		

Tabla 5. 3: Latencia por procesado más tarjeta de audio y ratio para diferentes tamaños de bloque

Considerando la latencia del programa como el tiempo desde que el procesado comienza hasta que se obtiene una respuesta de salida, hablaríamos del t_{proc} con el que se logra un *throughput* o ratio en cuanto al número de muestras de entrada procesadas por segundo.

Sin embargo, el tiempo necesario para que comience a actualizarse el filtro es de 4 iteraciones, es decir la primera iteración de arranque y las 3 sucesivas siguientes hasta que el bloque de la señal deseada contiene un valor y añade los primeros valores a la trama del error. En este sentido se tendría una latencia de $t_{proc} + 3t_{buff}$ para comenzar a actualizar el filtro, con una cadencia de t_{buff}.

En cuanto a la restricción de tiempos se ha medido experimentalmente que el límite de particiones a partir de las cuales el sistema deja de responder adecuadamente a las especificaciones temporales es a partir de un tamaño de filtro *12·B* puesto que el tiempo medio dedicado en el procesado supera el máximo a partir del cual se obtiene el siguiente bloque complejo, de forma que tanto la emisión como la captura son susceptibles de sufrir cortes intermitentes.

5.2.3 Mediciones multicanal.

Tanto el programa de identificación como ecualización multicanal radica en proceso iterativo entre dos etapas separadas en las que primero se obtienen los datos y a continuación se procesan.

Los programas de identificación y ecualización multicanal se encuentran diseñados en C y con la compatibilidad para su funcionamiento en tiempo real con ASIO a falta únicamente de depurar algunos pasos y verificar el correcto comportamiento entre las señales y su paso por los canales. El hecho de no haber profundizado en su desarrollo final ha sido el preferir focalizar el tiempo en solventar y mejorar la respuesta del sistema de un solo canal.

5.2.4 Conclusiones.

Tras un estudio teórico de las versiones de la familia LMS, se han aplicado éstos conocimientos para el desarrollo de un software programado sobre CPU de ámbito práctico capaz de ecualizar señales sonoras a través de canales acústicos en tiempo real.

Se ha comprobado que el sistema de identificación es eficiente logrando converger a unos resultados estables en cuestión de segundos con errores residuales muy pequeños incluso para factores de convergencia altos. La identificación acumula un desplazamiento inicial de tantas muestras como el tamaño del bloque empleado producido por la vinculación con la tarjeta de sonido y el programa en tiempo real de la CPU. Los parámetros destacados son el tamaño del bloque, el tamaño del filtro y el factor de convergencia. Cuando mayor es el tamaño del bloque la convergencia es más veloz así como aumentar el factor de convergencia. En cuanto al tamaño del filtro lo ideal es utilizar los coeficientes necesarios para recoger las características principales del canal y no sobrecargar de cálculos la aplicación de ecualización.

La compensación del canal se lleva a cabo con una identificación previa. El tamaño del filtro ecualizador puede diferir del estimado. El algoritmo disminuye el error iterativamente manteniéndolo estable. La respuesta por convolución entre filtros genera una delta próxima a la unidad. Se ha comprobado el efecto degradante del ruido acumulado en los coeficientes del ecualizador. Cuanto mayor es el tamaño del filtro ecualizador más interferencia se introduce a la señal a emitir puesto que a la salida se suman los ruidos de varianza en potencia de los N coeficiente. Esto es así porque las colas del filtro idealmente tendrían que ser nulos y en la realidad acumulan una dispersión significativa que deteriora el resultado. Se ha comprobado cómo eliminando las colas el parámetro de distancia mejoraba respecto al original. El uso de factores de convergencia bajos permite ralentiza el deterioro.

La aplicación responde exitosamente a cambios de entorno durante la actualización del filtro, logrando reajustar los coeficientes para optimizar su salida según el criterio de reducir el error con respecto a la señal objetivo.

Las condiciones de tiempo real se satisfacen para un conjunto amplio de combinaciones entre tamaños de bloque y número de particiones, incluso suficiente para estimar y ecualizar sin problemas casi cualquier canal. Pero es preferible emplear el tamaño de bloque más grande que permita unos tiempos de procesado holgados.

En general se recomienda usar tamaños de bloque a partir de 512 ya que tamaños más pequeños hacen que los filtros reduzcan su tamaño significativamente o necesitan de muchas particiones.

Capítulo 6. BIBLIOGRAFÍA.

6.1 Recursos consultados:

[1] B. Widrow and S.D. Stearns. *Adaptive Signal Processing*. Prentice Hall, 1985.

[2] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, 4th ed., 2002.

[3] B. Farhang-Boroujeny. Adaptive Filters: Theory and applications. National University of Singapore. Ed. Wiley & Sons. 1998 (Chapters 6-8).

[4] B. Farhang-Boroujeny. Adaptive Filters: Theory and applications. National University of Singapore. Ed. Wiley & Sons. 1998. Chapters 6, pp.154-156.

[5] L. Fuster, M. de Diego, M. Ferrer, A. González. *A Biased multichannel adaptive algorithm for room equalization*. Proceedings of the EUSIPCO 2012.

[6] J. Lorente, A. Gonzalez, M. Ferrer, M. de Diego. *GPU based implementation of multichannel adaptive room equalization*. Proceedings of the ICASSP 2014

[7] J. Lorente, M. Ferrer, José A. Belloch, Gema Piñero, M. de Diego , A. Gonzalez, Antonio M. Vidal. *Real-time adaptive algorithms using a Graphics Processing Unit*.

[8] Moran Moskovitch and David Fishelzo. *FilteredxLMS Algorithm DSP Implementation on BLACKFin BF533 EZ-KIT Lite*. Thursday, June 29, 2006.

[9] Ronald M. Aarts, Alexander W.M. Mathijssen, Piet C.W. Sommen, John Garas. *Efficient Block Frequency Domain Filtered-x applied to Phantom Sound Source Generation*.

[10] "The Danielson Lanczos Algorithm"
http://beige.ucs.indiana.edu/B673/node14.html

[11] " Danielson-Lanczos Lemma"
http://mathworld.wolfram.com/Danielson-LanczosLemma.html

[12] Steinberg Media Technologies GmbH. "*Steinberg Audio Streaming Input Output Specification. Development Kit ASIO 2.2*". Chapter II. Implementation Guide.