# Selección de parametrizaciones para Reconocimiento de Habla Local/Distribuido en dispositivos PDAs y móviles

Ana Isabel García Moral, Carmen Peláez Moreno, Fernando Díaz de María
Departamento de Teoría de la Señal y Comunicaciones
EPS, Avenida de la Universidad 30, 28912 Leganés, Madrid
Telf.: 916 248 805, Fax: 916 248 749
E-mail: aisabel@tsc.uc3m.es

**Resumen**

*En este documento se pretende establecer una comparativa entre varios Front-Ends implementados en aritmética de punto fijo. Nuestra investigación tratará de determinar cuál de ellos es el más adecuado para su instalación en dispositivos móviles de baja capacidad, tales como PDAs o teléfonos móviles, para su utilización en reconocimiento de habla, ya sea distribuido o local. La comparativa entre las distintas opciones de parametrización estudiadas se hará en términos de coste computacional, memoria, tiempo y prestaciones, utilizando para ello dos bases de datos diferentes que incluyen entorno limpio y ruidoso. Así, nuestros resultados experimentales nos permitirán concluir que la mejor solución consiste en utilizar los parámetros FF (Frequency Features) definidos por Nadeu et al. [8, 9], o, siempre que el coste computacional sea aceptable (para el dispositivo considerado), el Análisis LPC-Cepstrum con mejores prestaciones en ambientes ruidosos.*

## 1. Introducción

Hoy en día, los dispositivos móviles, especialmente los teléfonos, forman parte de nuestra vida diaria. Su tamaño se ha ido reduciendo al mismo tiempo que sus capacidades computacionales y de memoria se han ido incrementando, lo que ha convertido a estos terminales en una plataforma atractiva para el Reconocimiento Automático de Habla (ASR: Automated Speech Recognition). Así, por un lado, al tratarse de aparatos muy pequeños, las tradicionales interfaces de usuario, como por ejemplo el teclado, se han hecho más difíciles de utilizar, abriendo camino a otras más naturales y simples, como las interfaces vocales; y por otro, si no fuera por este incremento de recursos (computacionales y memoria), sería imposible incluir en tales dispositivos un procesado tan costoso como el necesario para ASR.

Si consideramos ahora las dos principales etapas en que se puede dividir un sistema típico de reconocimiento de habla: extracción de características (front-end) y clasificación-reconocimiento (back-end), encontramos tres arquitecturas distintas dependiendo de dónde se localice cada una de las etapas: Reconocimiento de Habla Remoto, Distribuido (DSR: Distributed Speech Recognition) y Embebido (ESR: Embedded Speech Recognition). En la primera de las arquitecturas, tanto el front-end como el back-end se localizan en un servidor remoto que, en general, no tiene las mismas restricciones de memoria y capacidad de cómputo que los dispositivos móviles, y por lo tanto no será objeto de este trabajo. Sin embargo, en las otras dos arquitecturas, al menos una de las etapas del reconocimiento tiene lugar en el dispositivo móvil. Así, en el DSR, el dispositivo local alberga sólo el módulo de parametrización (incluyendo un codificador para adaptar los vectores de características al canal de transmisión), mientras que en el ESR ambos módulos están en el dispositivo móvil. Dado que este documento trata sobre el estudio de distintos front-ends, nuestros resultados serán interesantes en estos dos últimos escenarios, que pueden seleccionarse según las capacidades del dispositivo a utilizar y la complejidad de la aplicación.

Además, no debemos olvidar que la mayoría de estos terminales portátiles usan una CPU en punto fijo de 16 bits con un sólo chip de memoria, y, consecuentemente, el front-end deberá implementarse usando algoritmos en punto fijo que presentan un menor consumo de recursos, aunque añaden complejidad en términos de ruido de cuantificación, overflows, disminución del rango dinámico y, sobre todo, un esfuerzo extra en lo que a programación se refiere.

A la hora de implementar el front-end en punto fijo dentro de un dispositivo móvil tenemos que resaltar dos aspectos importantes: 1) la optimización de los requisitos computacionales y de memoria, lo que no permitirá extender el vocabulario y afrontar tareas más complejas como el reconocimiento de habla continua; 2) la construcción de un sistema robusto, ya que los dispositivos móviles pueden utilizarse en casi cualquier lugar.

Una vez considerados los trabajos previos existentes en este contexto [1, 2, 3, 4], hemos visto necesario realizar un estudio comparativo de varias parametrizaciones, implementadas en punto fijo y probadas en diversas condiciones de ruido. El propósito será localizar una parametrización con menor carga computacional y menor consumo en memoria pero, al mismo tiempo, que sea lo

suficientemente robusta dentro de un ambiente ruidoso.

Este documento se organiza como sigue: la Sección 2 hace un breve resumen del estado del arte, en la Sección 3 pasaremos a describir las distintas parametrizaciones consideradas y su implementación en punto fijo, los experimentos realizados y resultados obtenidos se presentarán en la Sección 4, y finalmente, en la Sección 5 se hará una breve discusión sobre los resultados y en la Sección 6 se mostrarán las conclusiones y posibles líneas futuras.

## 2. Estado del arte

En este apartado se va a realizar un breve análisis del estado del arte en los sistemas ASR embebidos y, más concretamente, de las técnicas de parametrización utilizadas por ellos.

En primer lugar, puesto que hay muchos tipos de dispositivos móviles, vamos a distinguir entre dos de ellos: PDAs y teléfonos móviles. En el primer caso, debido a su mayor capacidad de memoria y computacional, ya existen soluciones bastante completas, aunque todavía queda abierto el camino de investigación en la mejora de la robustez en ambientes ruidosos y del trabajo con grandes vocabularios. En cambio, en el caso de los teléfonos móviles queda mucho por hacer, ya que siguen dando prioridad a la telefonía, y aunque sus capacidades de memoria y computacionales se han acercado a las de las PDAs, todavía no se han implementado soluciones del tipo de las ya existentes para éstas.

El parametrizador es uno de los módulos más importantes debido a la gran influencia que tiene en el resto de etapas, principalmente en la clasificación/decodificación. Además, es responsable en gran medida de la robustez del sistema, tan importante en los ambientes en los que se van a utilizar los dispositivos portátiles [3].

Dentro de la bibliografía consultada se encontraron comparativas de distintas parametrizaciones [4] donde se estudian: coeficientes LPC (Linear Predictive Coding) extraídos utilizando un codificador GSM estándar, LPCC (Linear Predictive Cepstral Coefficients), MFCC (Mel-Frequency Cepstral Coefficients) y PLP (Perceptual Linear Predictive), concluyendo que las dos últimas proporcionan una mayor robustez.

Otro punto a considerar es el esfuerzo dedicado a reducir la carga computacional y, por tanto, la tasa de información que se transfiere al módulo clasificador. Así, algunos autores sugieren extraer características cada 15ms en vez de cada 10ms [1] mostrando que las prestaciones en cuanto a WER (Word Error Rate) no empeoran considerablemente. En [5] se propone el uso de LDA (Linear Discriminant Analysis) para reducir la carga computacional. Y otros autores optan por reducir el número de componentes del vector de características argumentando que la WER no aumenta significativamente [4].

También encontramos referencias que sugerían eliminar tramas consideradas no-voz [1, 2, 3] para reducir los errores de inserción así como la carga computacional.

## 3. Descripción de Parametrizaciones. Implementación en punto fijo.

Un sistema típico ASR se puede dividir en dos etapas principales: la extracción de características del Front-End y el reconocimiento, o decodificado, también llamado Back-End.

El objetivo de la extracción de características es convertir la forma de onda de la voz en un conjunto de características útiles para el reconocimiento. Estos parámetros deben ser invariantes con respecto al hablante y perceptualmente significativos. En esta sección compararemos varias parametrizaciones implementadas en punto fijo, estudiando también su comportamiento en ambientes ruidosos.

### 3.1. Parametrizaciones consideradas

Clasificaremos las parametrizaciones estudiadas en dos grandes grupos: basadas en FFT y basadas en LPC. Dentro del primer grupo consideraremos los MFCCs y las FFs (Frequency Features) definidas por Nadeu et al. [8, 9], y dentro del segundo los LPCCs y los PseudoCepstrum, además de otra variante de los FFs (esta vez derivados del análisis LPC).

Para obtener los convencionales Mel-Cepstrums (MFCCs), seguimos el algoritmo de extracción de características descrito en el DSR front-end estándar de ETSI [8], implementado en este caso la versión completa en punto fijo. El diagrama de bloques general es el que se muestra en la Fig. 1.

Los FF son parámetros espectrales obtenidos mediante el filtrado en frecuencia (FF: Frequency Filtering) de las log-energías en banda (logFBEs), y para extraerlos hemos seguido los pasos indicados en [6] y [7].

La técnica de filtrado en frecuencia es una transformación lineal que es capaz de casi decorrelar las logFBEs convolucionándolas con la respuesta al impulso de un filtro FIR. En esta técnica, la secuencia de logFBEs de una trama dada, p.e. $S_1, S_2, \ldots, S_k, \ldots, S_Q$, donde Q es el número de bandas, se convoluciona con la respuesta al impulso de un filtro FIR de 1$^{er}$ o 2º orden (1), para obtener una nueva secuencia de parámetros filtrados, que serán nuestros parámetros FF.

$$H_1(z) = 1 - z^{-1}$$
$$H_2(z) = z - z^{-1} \quad (1)$$

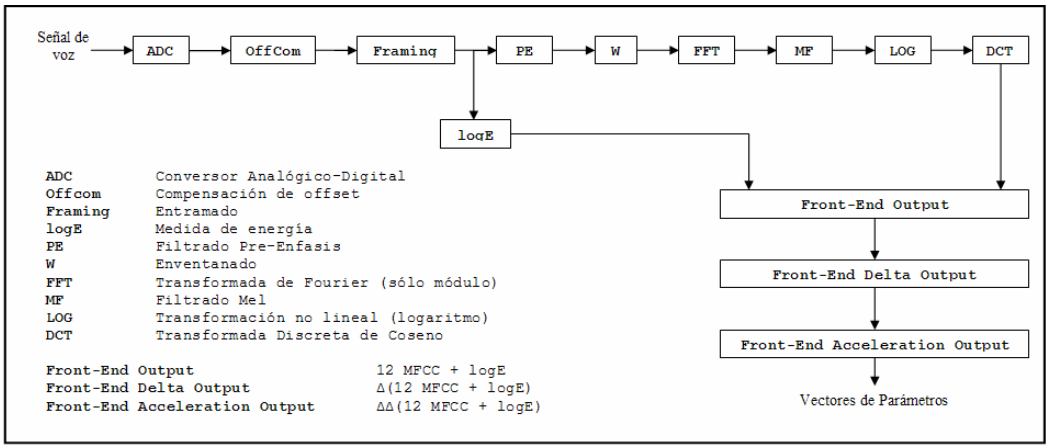


Figura 1: *Front-End MFCC.*

A la hora de aplicar estos filtros a la secuencia de logFBEs tenemos varias opciones para evitar problemas en el/los extremo/s de dicha secuencia: considerar la secuencia logFBE extendida con ceros o calcular una o dos bandas más (dependiendo del filtro). Después de realizar varios experimentos con el fin de determinar la mejor opción tanto de filtro como de extensión (ceros o más bandas) optamos por trabajar con el filtro FIR de primer orden ($H_1(z)$) y extender la secuencia logFBE con una banda más (Q+1). Así, nuestros parámetros FF serán:

$$(F_1, F_2, \ldots, F_Q) = (S_2 - S_1, S_3 - S_2, \ldots, S_{Q+1} - S_Q) \qquad (2)$$

Con respecto a su implementación en punto fijo, la hemos realizado siguiendo los mismos pasos que para la implementación de los MFCCs (Fig. 1), sustituyendo el último bloque DCT por el filtro FIR de primer orden.

En cuanto a la extracción de los parámetros LPCC, utilizamos los coeficientes del filtro LP obtenidos siguiendo la recomendación UIT-T G.729 [9] (ya expresada en punto fijo), continuamos calculando el espectro LP (LPS), el filtrado Mel (MF), la transformación logaritmo (Log) y finalmente, la DCT, tal y como se ilustra en la Fig. 2.

De nuevo podemos obtener los parámetros FF partiendo esta vez de los parámetros LPC y no de los MFCC. En este caso el proceso vuelve a ser el mismo que para la obtención de los LPCC sustituyendo el último bloque DCT por el filtro FIR de primer orden.

Para calcular los parámetros Pseudocepstrum partimos de los parámetros LSF (LSPs normalizados a 0-0.5) obtenidos siguiendo los pasos de la recomendación G.729 [9] y posteriormente, aplicando el filtrado Mel y la transformación Pseudocepstrum tal y como se describe en [12, 13]. Así, partimos de los parámetros LSF y los transformamos mediante el filtrado Mel, bien el tradicional (3) o el descrito en [11] por Choi, Kim y Lee (4). A continuación, realizamos la transformación a Pseudocepstrum [10] descrita en (5). El proceso total se describe con detalle en la Fig. 3.

$$Mel(f(Hz)) = 2595 \cdot \log 10\left(1 + \frac{f(Hz)}{700}\right) \qquad (3)$$

$$Mel_{CKL}(\omega_i) = 2\pi\left[\omega_i + \frac{1}{\pi}\arctan\left(\frac{0.45\sin(2\pi\omega_i)}{1 - 0.45\cos(2\pi\omega_i)}\right)\right] \qquad (4)$$

donde $i = 1, \ldots, 10$

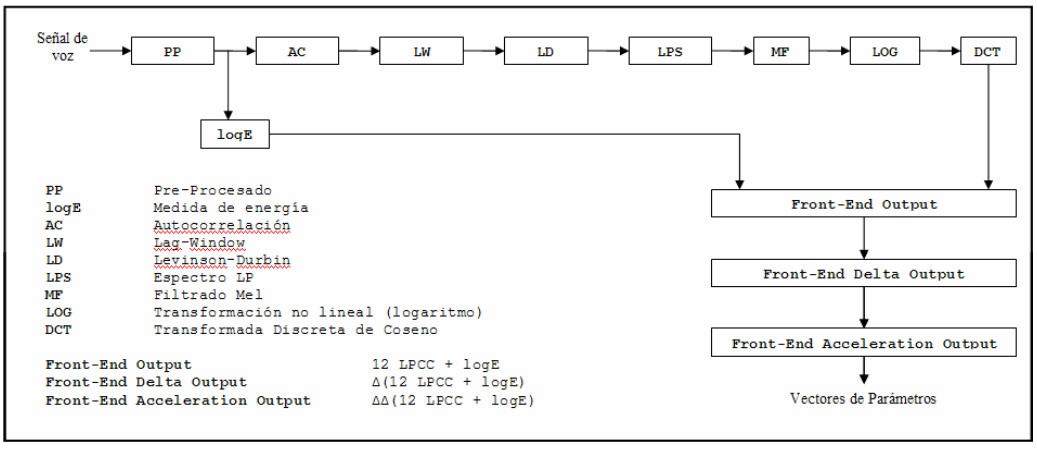


Figura 2: *Front-End LPCC.*

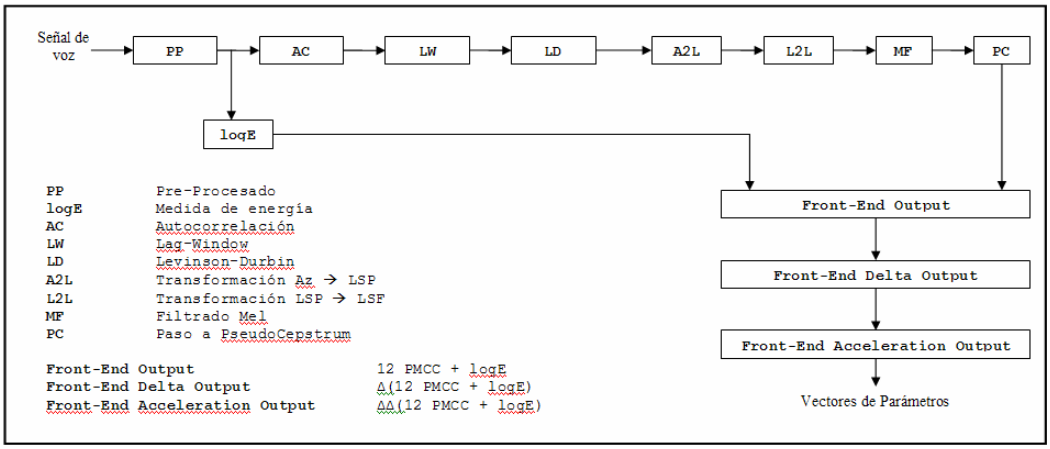


Figura 3: *Front-End PseudoCepstrum.*

$$pseudomel_i = \sum_{j=1}^{M} \frac{1}{k} \cos\left(k\omega_j\right) \quad \text{donde M} = 10 \text{ e } i = 1,\ldots,12 \qquad (5)$$

Para todas las parametrizaciones obtenemos 12 coeficientes, la log-energía, los coeficientes delta y, para la base de datos Aurora (ver 4.1), también los coeficientes de aceleración.

### 3.2. Implementación en punto fijo.

La representación en punto fijo de enteros implica (virtualmente) colocar el punto que separa la parte entera de la decimal en algún lugar dentro de la representación binaria del número. Por ejemplo, si tenemos un número entero normal (16 bits de longitud) como el mostrado en la Fig. 4 (a), éste puede servir para representar un número entre -32768 y 32767. Ahora pretendemos que pase a ser un número entero de 4 bits (más el bit de signo) con una componente racional de 11 bits, obteniendo así un número racional representado en aritmética de punto fijo como se muestra en la Fig. 4 (b). Este entero representará números desde -16.0 hasta aproximadamente +15.999512, con una precisión de $2^{-11}$. En este caso, la longitud de palabra (WL: Word Length) es el número total de bits para una variable representada en punto fijo. El IWL ( Integer Word Length) es el número de bits a la izquierda del hipotético punto binario, mientras que el número de bits a la derecha del mismo es la denominada FWL (Fraccional Word Length). Puesto que un mismo número puede tener diferente valor dependiendo del rango, se puede asignar un único IWL a cada variable. El rango R y el paso de cuantificación Q dependen del IWL como sigue:

$$\begin{aligned} Q &= 2^{-FWL} = 2^{WL-IWL-1} \\ -2^{IWL} &\le R \le 2^{IWL} - 2^{-FWL} \end{aligned} \qquad (6)$$

Así, un dato en punto fijo con resolución $Q_n$ tiene n bits a la derecha del (hipotético) punto decimal. La conversión de una variable X en punto flotante a otra en punto fijo con resolución $Q_n$ y viceversa requiere las operaciones descritas en (7) y (8), respectivamente.

$$\text{Float2Int}\left(X, Q_n\right) = Round\left[X * \left(2^{Q_n} - 1\right)\right] \qquad (7)$$

$$Int2Float\left(X, Q_n\right) = \left(float\right) X * 2^{-Q_n} \qquad (8)$$

Hay que hacer notar que la operación potencia se puede implementar mediante simples desplazamientos de bit, p.e., $2^{Qn} = 1 << Q_n$.

Por otro lado, los números en punto fijo pueden sumarse y restarse usando las operaciones normales con enteros + y -. La multiplicación y división ya no son tan sencillas debido a que hay que incluir desplazamientos de bits. Además hay que tener en cuenta para todas las operaciones los problemas de desbordamiento y redondeo que siempre pueden aparecer.

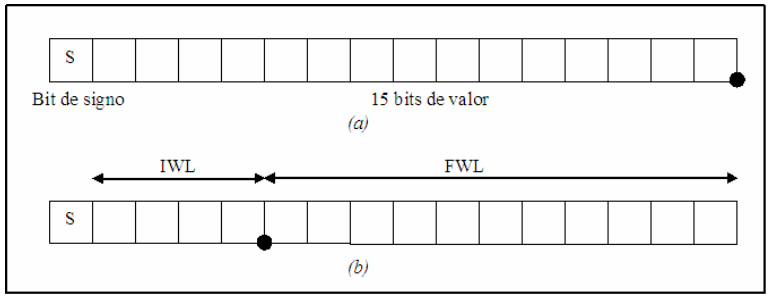


Figura 4: *(a) representación de un número entero en aritmética entera; (b) representación de un número racional en aritmética entera.*

Inicialmente se implementaron las operaciones aritméticas básicas mencionadas en el párrafo anterior y algunas más pero finalmente se optó por utilizar las implementadas en el estándar de la ITU-T G.729 [9], ya que resultaban muy fáciles de incluir en nuestro código.

Sin embargo, los algoritmos de procesado Front-end de una señal contienen mucha más matemática que otras aplicaciones, pero sin hacer un uso intensivo de la memoria. Los pasos de conversión hacia algoritmos en punto fijo se pueden agrupar en dos etapas: optimización algorítmica y optimación de la arquitectura. La primera de ellas consiste en reemplazar funciones como el logaritmo natural, la raíz cuadrada o la Transformada de Fourier (FFT) por aproximaciones de esos algoritmos en punto fijo [14, 15]. En la segunda se utilizarán algunos recursos como la representación en enteros de 16 bits, el uso de tablas precalculadas (LUT: Look.Up Tables) en algunas funciones como el filtrado Mel, la inclusión del control de overflow, y la búsqueda de la mejor resolución $Q_n$ en cada paso del proceso, de forma que éste sea más rápido a la vez que se consumen menos recursos.

## 4. Experimentos y Resultados.

### 4.1. Bases de datos y experimentos de referencia.

Hemos utilizado dos bases de datos para llevar a cabo los experimentos. La primera de ellas es una base de datos grabada con una PDA usada para hacer los experimentos. Es una base de datos de palabras aisladas en castellano, consistente en 6300 grabaciones de 50 locutores y con un máximo de tres repeticiones por locutor. Fue grabada en un ambiente no ruidoso con una frecuencia de muestreo de 8 Khz. La segunda base de datos utilizada fue la Aurora database, un subconjunto de la base de datos SDC (SpeechDat-Car). Es una base de datos en castellano de dígitos conectados, con supresión de continua, grabada a 16 Khz. y submuestreada a 8 Khz. Contiene 4914 realizaciones y más de 160 locutores. Las grabaciones fueron hechas en un coche, unas utilizando un micrófono cercano al hablante, y otras en el mismo entorno pero con un manos-libres. Los ficheros grabados se clasifican en tres categorías según las condiciones de ruido:

- **quiet:** con el motor de coche parado. Representan el 16.12% del total de grabaciones.
- **low noisy:** tráfico de ciudad y a baja velocidad por una carretera en mal estado. Representan el 49.28% del total.
- **high noisy**: gran velocidad en una carretera en buenas condiciones. Representan el 34.6% del total.

Para la BD PDA, el conjunto de realizaciones se dividió en 9 grupos, de manera que en cada experimento se realizaba entrenamiento con 8 grupos y reconocimiento con el noveno, usándose validación cruzada para obtener resultados más fiables.

Para la BD Aurora, se desarrollaron los tres experimentos base descritos en [16]. Éstos, hacen uso de diferentes mezclas de muestras de entrenamiento para recrear varios ambientes, y reciben el nombre de experimento Well-Matched (WM), Médium-MisMatch (MM) y High-Mismatch (HM), respectivamente.
Los parámetros fueron extraídos con los distintos parametrizadores descritos en la sección 3, utilizando para todos ellos una ventana de Hamming de 25ms con desplazamientos de 10ms. Se usaron HMMs (Hidden Markov Models) de tres estados, con 3 mezclas de gaussianas en el caso de la BD PDA y de 6 mezclas en la BD Aurora.

### 4.2. Resultados

Pasamos en este apartado a describir los resultados obtenidos en las simulaciones realizadas. Procederemos, al igual que hicimos en la descripción de los parámetros, separando los parámetrizadores en dos grupos: basados en FFT y basados en LPC. Para cada uno de los grupos haremos un estudio individual (por tipo de parametrización) y comparativo (entre las parametrizaciones del mismo grupo) para, finalmente, mostrar una comparativa entre las mejores opciones de cada grupo.

#### 4.2.1. Parametrizaciones basadas en FFT

Una vez implementada la versión en punto flotante de la parametrización MFCC se hicieron unos experimentos de referencia con las dos bases de datos de trabajo, para tener unos resultados iniciales con los que comparar y poder determinar si el paso a aritmética de punto fijo se había hecho correctamente.

Uno de los puntos más importantes dentro de la implementación en punto fijo fue la determinación de la resolución $Q_n$ en cada una de las etapas del proceso. Una vez fijada ésta para cada una de las etapas intermedias, se hicieron varios experimentos para elegir la $Q_n$ más adecuada para los datos de salida. Así, aunque los resultados no variaban demasiado en media, se llegó a la conclusión de que $Q_7$ era la mejor opción, como puede apreciarse en la Fig. 5 que muestra una gráfica con los resultados de WAR (Word Accuracy Rate) para varios valores de $Q_n$ y para cada una de las bases de datos de trabajo.

En la Tabla 1 se muestra una comparativa entre las implementaciones en punto fijo y flotante de los MFCCs, tomando como elementos de comparación WAR y el consumo en tiempo y memoria.

Tabla 1: *Comparativa MFCC*

| Aritmética | | | Punto Flotante | Punto Fijo ($Q_7$) |
|---|---|---|---|---|
| WAR (%) | BD Aurora | WM | 88.89 | 90.04 |
| | | MM | 77.13 | 80.10 |
| | | HM | 69.77 | 67.34 |
| | | Media | 78.60 | 79.16 |
| | BD PDA | Media | 96.81 | 96.08 |
| Consumo medio en Memoria (KB) | | | 887 | 843 |
| Consumo medio en Tiempo[1] | | | 0.6323 | 1 |

Vemos que, para el caso de BD Aurora, la tasa de acierto de palabra es prácticamente la misma en punto fijo y flotante, incluso ligeramente mayor en punto fijo para algunos casos. Esto se debe a que los experimentos con esta base de datos permiten realizar inserciones y el número de estas puede variar al cambiar de aritmética por los efectos de redondeo. Igualmente, para el caso de BD PDA los resultados en ambas aritméticas son muy similares.

En lo referente al consumo en memoria, vemos que es inferior en el caso de punto fijo, aunque la diferencia tampoco es importante. En cuanto al consumo en tiempo, vemos como la versión en punto flotante es casi el doble de rápida que la versión en punto fijo. Esto no debería ser así ya que se supone que se han realizado optimizaciones suficientes como para acelerar el proceso, pero suponemos que el problema está en que, por ejemplo, en la versión en punto fijo ha aumentado el número de accesos a ficheros (para leer los datos de las numerosas LUTs que utiliza).

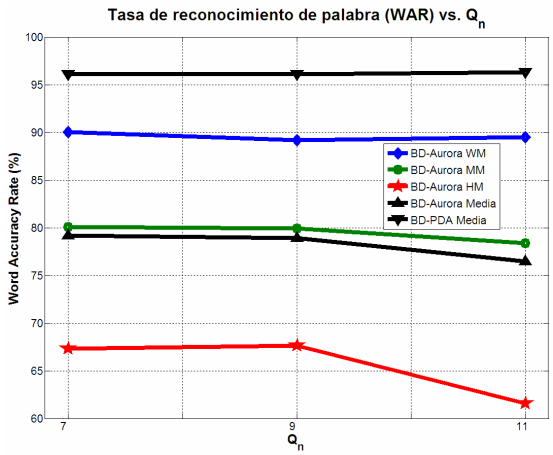


Figura 5: *WAR para distintos valores de $Q_n$ (MFCC Punto Fijo)*

[1] Las medidas de tiempo no se dan e valor absoluto sino que se toma como referencia la parametrización más lenta, a la que se da valor 1, calculándose los tiempos relativos a ésta para el resto de parametrizaciones.

La implementación de los parámetros $FF_{fft}$[2] fue sencilla debido a su semejanza con la parametrización anterior (MFCC) (ya que simplemente requería sustituir el bloque final DCT por un filtro FIR muy fácil de implementar, incluso en aritmética entera).

Antes de hacer pruebas en punto fijo preferimos hacer simulaciones en punto flotante para determinar con exactitud cuál de los dos filtros indicados en (1) era el más adecuado, además de elegir el modo en que debíamos extender la secuencia de logFBEs para el cálculo de los parámetros $FF_{fft}$, como se indicó en el apartado 3.1. También se aprovechó para extraer vectores de características de distintos tamaños y poder así determinar el número de coeficientes necesarios para obtener resultados similares a los conseguidos con los MFCCs. Estas pruebas se realizaron sólo para la base de datos Aurora y los resultados se muestran en la Fig. 6. Observando esta gráfica podemos ver que la curva más estable corresponde al uso de un filtro FIR de primer orden y de una extensión del número de bandas.

En la Tabla 2 se muestra una comparativa entre la versión en punto fijo y flotante de los parámetros $FF_{fft}$. Vemos que para esta parametrización también tenemos resultados muy similares en ambas aritméticas, con algunas mejoras llamativas en la versión en punto fijo y la BD Aurora, posiblemente por los mismos motivos comentados para la parametrización MFCC.

En lo referente al consumo en tiempo y memoria no se indican nuevos datos porque la variación con respecto a la parametrización MFCC es mínima y por tanto los consumos son muy similares. Se hicieron pruebas para distintas resoluciones y los resultados no variaban demasiado por lo que optamos por utilizar $Q_7$ al igual que en la parametrización anterior.

Tabla 2: *Comparativa $FF_{fft}$*

| Aritmética | | | Punto Flotante | Punto Fijo ($Q_7$) |
|---|---|---|---|---|
| WAR (%) | BD Aurora | WM | 87.95 | 90.74 |
| | | MM | 72.24 | 80.87 |
| | | HM | 69.23 | 68.84 |
| | | Media | 76.47 | 80.15 |
| | BD PDA | Media | 96.17 | 95.98 |

[2] Los llamamos así para diferenciarlos de los parámetros FF obtenidos a partir de los LPCs, a los que llamaremos $FF_{lpc}$.

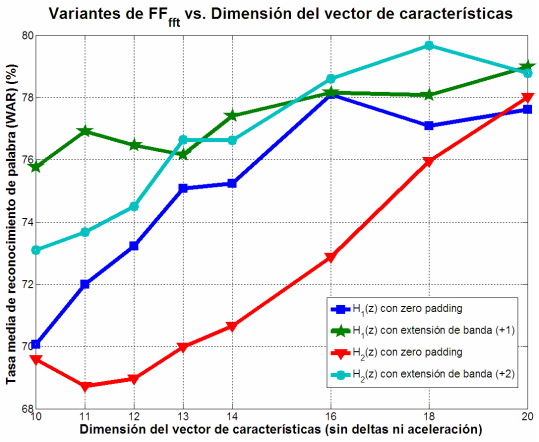


Figura 6: *WAR Media para distintas variantes de $FF_{fft}$ y distinto tamaño de vector de características.*

### 4.2.2. Parametrizaciones basadas en LPC

Para la parametrización Pseudocepstrum tuvimos que decidir cuál de las dos implementaciones del filtrado Mel, (3) o (4), íbamos a utilizar. Para ello, realizamos experimentos en punto flotante y fijo para ambos y, en todos los casos, obtuvimos que el filtrado recomendado por Choi, Kim y Lee [11] daba mejores resultados. En la Tabla 3 se muestran las comparativas en WAR para ambas aritméticas y filtrados.

Al igual que para la parametrización MFCC, se hicieron pruebas para determinar el valor de $Q_n$ más adecuado para la parametrización Pseudocepstrum, y en este caso se llegó a la conclusión de que $Q_9$ era la mejor opción, como puede apreciarse en la Fig. 7 que muestra una gráfica con los resultados de WAR (Word Accuracy Rate) para varios valores de $Q_n$ y para cada una de las bases de datos de trabajo.

En la Tabla 4 se muestra una comparativa entre las implementaciones en punto fijo y flotante de los PseudoCepstrum, tomando como elementos de comparación la WAR y el consumo en tiempo y memoria.

Tabla 3: *Comparativa PseudoCepstrum para distinto tipo de filtrado Mel.*

| | WAR Media (%) | | | |
|---|---|---|---|---|
| | BD Aurora | | BD PDA | |
| | Punto Flotante | Punto Fijo $Q_9$ | Punto Flotante | Punto Fijo $Q_9$ |
| **Mel** | 66.53 | 69.18 | 96.17 | 96.11 |
| **$Mel_{cld}$** | 70.75 | 74.40 | 96.46 | 96.03 |

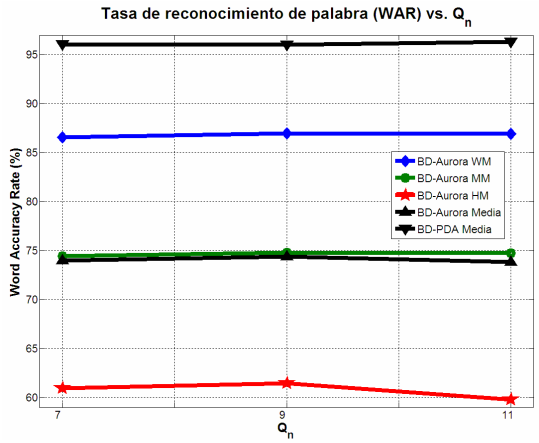


Figura 7: *WAR para distintos valores de $Q_n$ (PseudoCepstrum punto fijo)*

Vemos que, para el caso de BD Aurora, la WAR es incluso mayor en punto fijo que en punto flotante, debido de nuevo al efecto de redondeo y a la variación en el número de inserciones que éste provoca. Igualmente, para el caso de BD PDA los resultados en ambas aritméticas son muy similares.

En lo referente al consumo en memoria, vemos que es inferior en el caso de punto fijo, aunque la diferencia tampoco es importante. En cuanto al consumo en tiempo, vemos como la diferencia entre ambas versiones es mínima, siendo ligeramente más rápida de nuevo la versión en punto flotante.

Para implementar la parametrización LPC utilizamos como base el código proporcionado en [9], lo que nos simplificó mucho el trabajo puesto que ya venía desarrollado en aritmética entera, y simplemente hubo que añadir los cuatro últimos bloques del procesado, algunos de los cuales son comunes a las otras parametrizaciones. El principal problema que volvimos a encontrar aquí fue la optimización del valor de $Q_n$ a la salida, pero de nuevo lo fijamos a $Q_7$ ya que los resultados obtenidos en comparación con los de punto flotante eran bastante satisfactorios, como se puede ver en la Tabla 5.

Tabla 4: *Comparativa PseudoCepstrum*

| Aritmética | | | Punto Flotante | Punto Fijo ($Q_9$) |
|---|---|---|---|---|
| WAR (%) | BD Aurora | WM | 86.13 | 86.98 |
| | | MM | 67.29 | 74.77 |
| | | HM | 58.83 | 61.44 |
| | | Media | 70.75 | 74.40 |
| | BD PDA | Media | 96.46 | 96.03 |
| Consumo medio en Memoria (KB) | | | 847 | 818 |
| Consumo medio en Tiempo | | | 0.9835 | 1 |

Tabla 5: *Comparativa LPCC*

| Aritmética | | | Punto Flotante | Punto Fijo ($Q_9$) |
|---|---|---|---|---|
| WAR (%) | BD Aurora | WM | 85.72 | 88.60 |
| | | MM | 69.71 | 77.64 |
| | | HM | 71.85 | 73.50 |
| | | Media | 75.76 | 79.91 |
| | BD PDA | Media | 95.44 | 96.08 |
| Consumo medio en Memoria (KB) | | | 879 | 854 |
| Consumo medio en Tiempo | | | 0.5814 | 1 |

Al igual que ocurría con las anteriores parametrizaciones, los resultados en punto fijo y flotante son similares, el consumo en memoria es prácticamente el mismo y en tiempo sigue siendo más rápida la versión en punto flotante.

En cuanto a la implementación de los parámetros $FF_{lpc}$, al igual que para el caso de los $FF_{fft}$, parece sencilla debido a su semejanza con la parametrización anterior (LPCC), ya que simplemente requiere sustituir el bloque final DCT por un filtro FIR muy fácil de implementar incluso en aritmética entera.

Las pruebas para selección del filtro y el tipo de extensión de la secuencia de logFBEs que se hicieron para el caso de $FF_{fft}$ se volvieron a repetir para esta parametrización, obteniéndose las mismas conclusiones. En la Fig. 8 se pueden ver los resultados de estas simulaciones.

En la Tabla 6 se muestra una comparativa entre la versión en punto fijo y flotante de los parámetros $FF_{lpc}$. Como se puede comprobar la diferencia es considerable, suponemos que debido a un error en la elección de la $Q_n$ de salida, ya que la implementación además de sencilla (al partir de los LPCC) es idéntica en estructura a la hecha en punto flotante y de la que obtuvimos buenos resultados. En principio se hicieron pruebas para $Q_7$ además de las mostradas para $Q_9$ pero resultaron tasas de reconocimiento mucho más bajas. En este punto no se dedicó más esfuerzo puesto se comprobó que los resultados ya en punto flotante no eran mejores a los obtenidos con los $FF_{fft}$ ni en WAR ni mucho menos en consumo de tiempo.

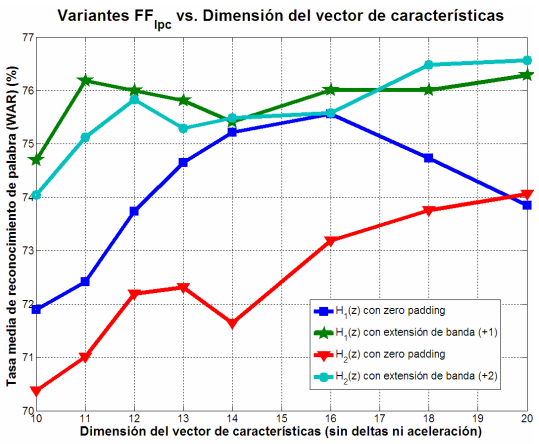


Figura 8: *WAR Media para distintas variantes de $FF_{lpc}$ y distinto tamaño de vector de características.*

## 5. Discusión

Analizando los resultados obtenidos para ambos grupos de parametrizaciones, podríamos concluir que las basadas en FFT tienen un comportamiento muy similar entre ellas y ni siquiera podemos decir que consuman mucho menos tiempo los $FF_{fft}$ pese a presentar una etapa final más simple que los MFCCs, puesto que en punto fijo esta simplicidad también se produce en la etapa DCT de los MFCCs al sustituir esta función por una LUT.

Entre los parámetros de grupo basado en LPC, encontramos más diferencias. Así, mientras que en media los resultados son mejores en los LPCCs, el procesado para obtener estos parámetros requiere casi el doble de tiempo que para obtener los PseudoCepstrum.

En la Tabla 7, y Fig. 9 mostramos la comparativa entre todas las parametrizaciones estudiadas. Así, podemos comprobar que en media $FF_{fft}$ y LPCC son las que presentan mejores resultados, siendo la primera más eficiente en cuanto a tiempo de ejecución. Para un ambiente ruidoso (HM – BD Aurora), parece mejor opción LPCC, aunque si nos fijamos en un entorno intermedio (MM – BD Aurora) los $FF_{fft}$ se comportan incluso mejor que éstos.

Tabla 6: *Comparativa $FF_{lpc}$*

| Aritmética | | | Punto Flotante | Punto Fijo ($Q_7$) |
|---|---|---|---|---|
| WAR (%) | BD Aurora | WM | 86.80 | 67.24 |
| | | MM | 72.71 | 55.91 |
| | | HM | 68.48 | 39.37 |
| | | Media | 76.00 | 54.17 |
| | BD PDA | Media | 96.08 | - |

Tabla 7: *Comparativa entre parametrizaciones en punto fijo.*

| | WAR (%) | | | | | Consumo medio | |
|---|---|---|---|---|---|---|---|
| | BD Aurora | | | | BD PDA | Memoria (KB) | Tiempo relativo |
| | WM | MM | HM | Media | Media | | |
| **MFCC** | 90,04 | 80,10 | 67,34 | 79,16 | 96,08 | 843 | 0,8494 |
| **$FF_{fft}$** | 90,74 | 80,87 | 68,84 | 80,15 | 95,98 | 843 | 0,8072 |
| **PseudoCepstrum** | 86,98 | 74,77 | 61,44 | 74,40 | 96,03 | 818 | 0,5697 |
| **LPCC** | 88,60 | 77,64 | 73,50 | 79,91 | 96,08 | 854 | 1 |

Con respecto al consumo en tiempo, el parametrizador más rápido es el PseudoCepstrum que no presenta malos resultados en cuanto a WAR para entornos poco ruidosos, pero que pierde muchas prestaciones al pasar a entornos no limpios.

En definitiva y a la vista de los resultados, la mejor opción en media es la de los $FF_{fft}$ puesto que no es la más lenta y es la que mantiene una respuesta más estable independientemente de ambiente considerado.

## 6. Conclusiones y líneas futuras

En este trabajo se han implementado un conjunto de Front-Ends en punto fijo con el fin de determinar la mejor solución para un sistema ASR Local/Distribuido. Todos ellos han sido probados en distintas condiciones y para dos bases de datos diferentes.

Así, hemos notado que, aunque hay cierta variación dependiendo del ambiente de test, las características más estables en casi todos ellos fueron los $FF_{fft}$ (y los MFCC por extensión ya que los primeros se extraer siguiendo un procedimiento muy similar a los segundos). En cambio, los LPCC son una solución también recomendable si el ambiente de trabajo es ruidoso y siempre que el sistema tenga los recursos computacionales necesarios, ya que es la opción más costosa.

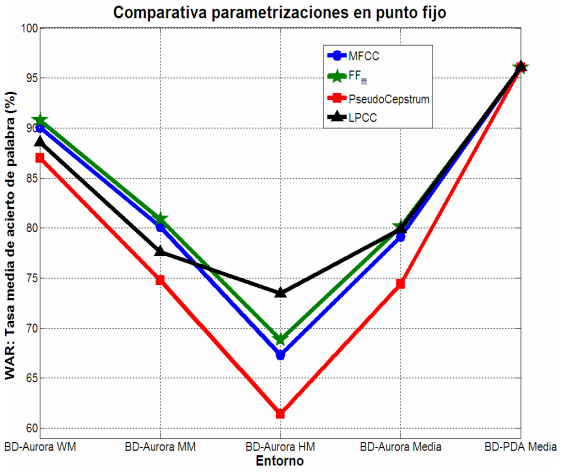


Figura 9: *Comparativa WAR entre todas las parametrizaciones consideradas.*

Como primera línea futura ya en desarrollo, podemos considerar la ampliación este estudio utilizando técnicas de reducción del número de parámetros como LDA o PCA con el fin de comprobar el efecto que produciría este proceso en las parametrizaciones consideradas. Además se pueden estudiar técnicas de optimización de las implementaciones en punto fijo, sobre todo de los LPCC, para intentar hacerlas más rápidas.

## 7. Referencias


[1] Deligne, S., Dharanipragada, S., Gopinath, R., Maison, B., Olsen, P. and Printz, H., "A Robust High Accuracy Speech Recognition System for Mobile Applications", *IEEE Trans. on Speech and Audio Processing*, Vol. 10, No. 8, pp. 551-561, Nov. 2002.

[2] Varga, I., Aalburg, S., Andrassy, B., Astrov, S., Bauer, J.G. , Beaugeant, C., Geibler, C. and Höge, H. , "ASR in Mobiles Phones – An Industrial Approach", *IEEE Trans. on Speech and Audio Processing*, Vol. 10, No. 8, pp. 562-569, Nov. 2002.

[3] Bi, N. Garudadri, H., Chang, C., DeJaco, A., Qi, Y., Malayath, N., y Huang, W. "A robust speech recognition system embedded in CDMA cellular phone chipsets" ICASSP, IV: 3804-3807, 2002.

[4] Lévy, C., Linarès, G:, Nocera P., Bonastre, J.-F., "Reducing Computacional and Memory Cost for Cellular Phone Embedded Speech Recognition System", in *Proceedings of ICASSP*, vol. V, pp. 309-312, 2004.

[5] Astrov, S., Bauer, J. G., y Stan, S. "High performance speaker and vocabulary independent ASR technology for mobile phones." ICASSP, II:281-284, 2003.

[6] Nadeu, C., Mario, J.B., Hernando, J., y NOgueiras, A. "Frequency and time filtering of filter-bank energies for HMM speech recognition." ICSLP 96, 1996.

[7] Pujol, P., Pol, S., Hagen, A., Boulard, H., y Nadeu, C. “Comparison and combination of RASTA-PLP and FF features in a hybrid HMM/MLP speech recognition system.” ICSLP, 10:562-569, 2002.

[8] ETSI Technical Committee. Speech processing, transmission and quality aspects; distributed speech recognition; front-end feature extraction algorithm. ETSI ES 201 108, 2004.

[9] Recomendación UIT-T G.729. 1996.

[10] Peláez-Moreno, C., Gallardo-Antolín, A., Díaz de María, F. “A comparison of Front-Ends for Bitstream-Based ASR over IP”. Elsevier Science, 2004.

[11] Kim, H.K., Choi, S.H., Lee, H.S. “On Approximating Line Spectral Frequencies to LPC Cepstral Coefficients.” IEEE Transactions on speech and audio processing, Vol. 8, 2000.

[12] Piromsopa, K., Aportewan, C., y Chongstivatana, P. “An FPGA implementation of a fixed-point square root operation.” Inter. Symposium on Communications and Information Technology, pages 587-589, 2001.

[13] Crenshaw, J.W. “Math toolkit for real-time programming. CMP Books, 2000.

[14] AU/271/00. Spanish SDC-Aurora database for ETSI STQ Aurora WI008 advanced DSR front-end evaluation: Description and baseline results. UPC, 2000.

[15] Gallardo-Antolín, A., Díaz de María, F., Martín-Iglesias, D., Pereiro-Estevan, Y., y García-Moral, A.I. “Design of a voice-enabled interface to check the IBEX-35 price from a PDA by GPRS. INTERSPEECH 2005, 2005.