UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

T E S I S

PLUMA MAGNETOELÁSTICA PARA RECONOCIMIENTO DE FIRMA AUTÓGRAFA

QUE PARA OBTENER EL TÍTULO DE INGENIERO ELÉCTRICO ELECTRÓNICO PRESENTAN:

MANUEL RAMÍREZ ÁLVAREZ

POLO FRANCISCO PADILLA MONROY

DIRECTOR DE TESIS:

DR. JOSÉ ISRAEL BETANCOURT REYES

MÉXICO D. F. **2004**

Con gratitud, admiración y respeto, dedico este trabajo:

A mi mamá María Santos. Por que de ella aprendí a luchar en la vida, por que me alentó a seguir adelante y a esforzarme para alcanzar mis objetivos, por que me enseñó el valor del trabajo y la constancia, por el cariño y apoyo que me ha brindado, por que una de sus principales satisfacciones ha sido verme convertido en un profesionista y por que nunca la podría defraudar.

A mi hermana Dulce. Por su apoyo incondicional, por la confianza que en mí ha depositado, por sus palabras de motivación presentes al enfrentar cada nuevo reto y por que sin su ayuda esto no hubiera sido posible.

A mi hermano Leonardo. Por que además de ser mi hermano es mi amigo y por que me ha transmitido la alegría e inquietud de vivir y disfrutar las grandes experiencias que se presentan en la vida.

A Alicia. Por ayudarme a ser más feliz cada día, por estar junto a mí en todo momento, por su amor y amistad incondicional, y por brindarme siempre palabras de aliento y esperanza que me dan fuerza para seguir adelante.

A Polo. Por ser de esa clase de personas que todo lo comprenden y dan lo mejor de sí, por ser un gran compañero de equipo y por su valiosa amistad.

A todos mis compañeros y amigos de la carrera. Por todos esos momentos que compartimos juntos y por todas aquellas inolvidables experiencias.

A todos los que hicieron posible este trabajo, en especial a nuestro asesor y sinodales, que compartieron con nosotros tiempo y esfuerzo.

A todas aquellas personas que de alguna u otra manera contribuyeron en la realización de este trabajo, pues no hubieron ayudas pequeñas y todas fueron bien recibidas.

Manuel

Con todo el cariño amor y respeto que se merecen, dedico este trabajo a mis padres, Esteban Francisco Padilla Pérez y Bertha Monroy Alonso, por haberme apoyado de manera incondicional todo este tiempo. Agradezco a mi madre, las palabras de aliento que siempre ha tenido para mí, a mi padre el haber dedicado su vida a su familia, a ambos agradezco tener la dicha de vivir.

Con gran respeto y admiración dedico este trabajo a mi hermano Ignacio Padilla Monroy, él ha sido parte fundamental de este logro, por tal motivo lo comparto con él, en espera de que él pronto comparta el suyo conmigo.

Con todo el amor del mudo dedico este trabajo a Teresita, pues ella ha sido la luz que ilumina mi sendero, el motivo que tengo para luchar, seguir adelante y nunca darme por vencido. Porque ella apareció en mi vida y ahora es parte de mi destino.

En especial dedico este trabajo a Manuel, por haberme permitido realizar este sueño con él, por que de él aprendí mucho y hay mucho más por aprender.

Para todos los profesores, amigos y compañeros de la facultad, que hicieron mi estancia más placentera. Dedicado a todas las personas que de algún modo contribuyeron en la elaboración de este trabajo.

Polo

ÍNDICE

CAPÍTULO 4
ANÁLISIS, DISEÑO Y DESARROLLO DEL SISTEMA DE RECONOCIMIENTO DE FIRMA AUTÓGRAFA

CAPÍTULO 5
RESULTADOS Y CONCLUSIONES

APÉNDICE A
PROGRAMAS

APÉNDICE B
TRABAJOS Y RECONOCIMIENTOS

ANTECEDENTES HISTÓRICOS

Las características de un objeto, que son inspeccionadas con el fin de identificarlo, se denominan patrones. El reconocimiento de patrones como desarrollo tecnológico ha tenido un gran impulso desde los años 50´s debido a la aparición de las computadoras digitales y a su creciente capacidad para procesar información, lo que ha permitido programar y poner a prueba algoritmos de reconocimiento que por su elevado coste computacional no habían podido ser implementados previamente [1]. Algunos de los primeros esfuerzos en el campo del reconocimiento de patrones fueron tentativas de programar computadoras para tomar decisiones automáticas y desarrollar hardware especializado en leer patrones tales como caracteres alfanuméricos [1]. Un ejemplo de estos desarrollos incipientes fue el Perceptrón de Rossenblatt, basado en el algoritmo Perceptrón, que describe la forma de almacenar la información en el cerebro [1].

Aunado al desarrollo del reconocimiento de patrones está el avance en el estudio de la inteligencia artificial, pues utiliza el reconocimiento de patrones como herramienta para lograr sus objetivos. El primer reporte sobre la inteligencia artificial fue realizado en 1943 por W. McCulloch y W. Pitts [1], quienes propusieron un modelo construido por neuronas artificiales. Posteriormente, D. Hebb en 1949 demostró una sencilla regla de actualización para modificar las conexiones entre neuronas de manera que ocurriera el aprendizaje [2]. A principios de la década de los 50’s, C. Shannon y A. Turing, escribieron programas de ajedrez para computadoras convencionales, mientras que, M. Minsky y D. Edmonds, construyeron la primera computadora de red neuronal [2]. Estos esfuerzos iniciales en la resolución de problemas se basaban en un mecanismo de búsqueda de propósito general en el que se entrelazaban pasos de razonamiento elementales para encontrar de esta forma, soluciones completas. El programa DENDRAL, construido por E. Feigenbaum, B. Buchanan y J. Lederberg en 1969 [2], constituye un ejemplo típico de este enfoque.

La necesidad de aplicaciones prácticas impulsó el aumento en la demanda de esquemas de representación del conocimiento, para lo cual, se desarrollaron varios lenguajes de representación como PROLOG [2], que era una PROgramación LÓGica diseñada para la manipulación de símbolos [3]. En 1981 los japoneses anunciaron el proyecto de la “quinta generación”, un plan de 10 años para construir computadoras inteligentes en las que corriera PROLOG [2]. Desde entonces, han sido muchas y muy variadas las aplicaciones que se le han

dado al reconocimiento de patrones. La primera máquina lectora de Kurzweil (KRM, 1976) por ejemplo, consistía en una máquina exploradora de imágenes [4]. El reconocimiento auditivo ha recibido también mucha atención, ya que los sonidos son fundamentales para la interacción con el mundo.

En 1985, Kurzweil introdujo el primer dispositivo comercial de reconocimiento de voz con un vocabulario de 1000 palabras, y en 1987, la versión mejorada de 10000 palabras [4]. Una de las grandes aplicaciones comerciales del reconocimiento de patrones se encuentra en las fábricas, donde se emplean sistemas para la inspección, el ensamble y los procesos de control automático. Estos sistemas son cámaras de estado sólido con electrónica especializada que codifica numéricamente las imágenes en movimiento para el cálculo intensivo, en fase temprana del procesamiento [4]. Por su parte, los sistemas militares cuentan con una mayor aplicación de la inteligencia artificial, enfocada por ejemplo, en la capacidad para buscar y reconocer terrenos de altitudes muy bajas, aptitud crucial para los misiles cruise [4].

Otra área emergente e importante en su aplicación es la medicina, ya que el diagnóstico médico se puede apoyar en el reconocimiento de patrones relevantes a partir de los síntomas, resultados de pruebas y otros exámenes. Una aplicación médica particularmente promisoria es el análisis de imágenes de células sanguíneas para la detección de anomalías [4]. Una vez que las computadoras hayan dominado los requisitos de las tareas de reconocimiento de patrones, habrá potencialmente una mayor transformación de las pruebas médicas y el diagnóstico. Técnicas similares son utilizadas en dispositivos de seguridad capaces de leer rápidamente el patrón de la huella dactilar de una persona para confrontarla mediante técnicas de reconocimiento de patrones, contra imágenes previamente almacenadas [4]. Algunas aplicaciones más del reconocimiento de patrones en la inteligencia artificial son: sistemas traductores de lenguaje, controladores de tráfico aéreo, sistemas supervisados, robótica y exploración espacial [5]. El reconocimiento de patrones también ha influido notablemente en el diseño de modernos sistemas computarizados de información en áreas del conocimiento tan diversas como: astronomía, geología, biología, medicina, reconocimiento de caracteres e imágenes, análisis de voz y análisis de firma autógrafa, entre otras muchas aplicaciones [6].

INTRODUCCIÓN

El actual desarrollo científico y tecnológico de la Ciencia de Materiales ha permitido la obtención de materiales funcionales que hagan las veces de transductores mecánico–magnéticos. Junto al avance de la electrónica de estado sólido, que ha permitido diseñar e implementar dispositivos capaces de trabajar a grandes velocidades de procesamiento, han posibilitado la implementación y desarrollo de dispositivos detectores cada vez más pequeños, versátiles y eficientes. En particular, el desarrollo de nuevos materiales magnéticos en forma de pequeños alambres amorfos con propiedades magnéticas y elásticas únicas ha posibilitado su uso como detectores finos de esfuerzos, tanto de tensión como de compresión [7]. Esta característica del material lo convierte en el transductor apropiado para la transformación de esfuerzos mecánicos en señales eléctricas de intensidad y duración variables, que puedan ser digitalizadas, almacenadas y analizadas con la ayuda de una computadora digital.

El objetivo de este trabajo de tesis es diseñar e implementar un dispositivo capaz de reconocer una firma autógrafa mediante un transductor mecánico–eléctrico, en forma de “pluma magneto-elástica”, que convierta los esfuerzos mecánicos de tensión y compresión generados por la mano al efectuar una firma autógrafa, en señales eléctricas.

El sistema de reconocimiento de firma autógrafa realiza la autentificación de una firma mediante la aplicación de técnicas y algoritmos de reconocimiento de patrones. Las etapas diseñadas e implementadas para el desarrollo del sistema son: generación de la señal eléctrica, digitalización, preprocesamiento, entrenamiento y reconocimiento.

La etapa inicial para realizar el reconocimiento de la firma autógrafa, consiste en la obtención de una señal eléctrica que sea proporcional a la rúbrica del firmante. Para esto se utiliza un transductor mecánico–eléctrico, con el cual es posible obtener señales eléctricas que reflejen características importantes de una firma. Debido a que la magnitud de dicha señal es relativamente pequeña, es necesario implementar una etapa de amplificación para reducir errores de cuantización al momento de muestrear la señal.

Para realizar un análisis más completo de la señal analógica obtenida, es conveniente digitalizarla. Una vez que efectuada la digitalización se deben aplicar técnicas y algoritmos que permitan el proceso de identificación. La etapa de digitalización de una señal, realiza la conversión analógica-digital mediante la programación de la tarjeta de sonido de una computadora personal, con lo que se transforma a ésta en una herramienta para digitalizar, almacenar y estudiar este tipo de señales.

El preprocesamiento consiste en la aplicación de un conjunto de procesos que estandaricen a las señales digitalizadas, debido a que éstas se producen bajo diferentes condiciones: firmado rápido, lento, firme, débil, etc. En esta etapa, las señales digitalizadas y almacenadas son sometidas a procesos de filtrado, recorte por los extremos y normalización tanto en magnitud como en longitud. Del conjunto de señales estandarizadas se obtienen los patrones característicos que servirán para su posterior autentificación.

El filtrado permite eliminar el ruido eléctrico que ha sido agregado a la señal durante el proceso de su obtención, amplificación y digitalización. Para el recorte de la señal se aplican técnicas para localizar en la señal digitalizada el inicio y fin de la información de la firma. En la normalización en magnitud, se escala cada firma de modo que el máximo valor digitalizado ocupe el mayor valor permitido. Finalmente, la normalización en longitud consiste en representar a todas las firmas mediante un mismo número de muestras.

Una vez que las diferentes señales han sido digitalizadas y preprocesadas prosigue el entrenamiento del sistema, en el que se realiza la extracción de parámetros mediante la aplicación de técnicas de reconocimiento de patrones. En esta etapa se emplean distintos algoritmos de segmentación y ventaneo para la obtención de los patrones de las firmas, entre los que se encuentran: el análisis de predicción lineal, el análisis de aproximación polinomial y la envolvente promediadora. Finalmente, se realiza el agrupamiento de los patrones de diversas firmas del mismo autor para obtener las características más relevantes de aquellas en conjunto. De este modo culmina el entrenamiento del sistema de reconocimiento de firma autógrafa.

En la etapa de reconocimiento, la firma autógrafa que se requiere verificar se digitaliza y se almacena en un archivo para preprocesarla, segmentarla, ventanearla, extraerle sus patrones y agruparlos. Por último, los resultados

obtenidos en el agrupamiento se comparan con los obtenidos en la etapa de entrenamiento. Para realizar esta comparación se utiliza una función distancia como medida de similitud, y dependiendo de su valor, se valida o rechaza la firma bajo prueba.

Esta tesis se desarrolla en cinco capítulos. En el capítulo 1, se incluyen algunos conceptos básicos del magnetismo junto con una descripción general de los alambres magneto–elásticos. En el capítulo 2, se estudian diferentes conceptos que tienen relación directa con la implementación del sistema de reconocimiento de firma autógrafa, tales como el acondicionamiento de las señales bajo estudio, el procesamiento digital de señales y temas relacionados con el reconocimiento de patrones. En el capítulo 3, se realiza una breve descripción de la tarjeta de sonido de una computadora, mostrando algunas de sus características y detallando algunos pormenores sobre su programación. En el capítulo 4, se muestran detalladamente las etapas que conforman al sistema desarrollado para el reconocimiento, haciendo mención de los conceptos aplicados y su justificación. En el capítulo 5, se presentan los resultados obtenidos al someter a prueba al sistema de reconocimiento de firma autógrafa, junto con las conclusiones obtenidas al realizar este trabajo. Por último, en el apéndice A se muestran los programas realizados en lenguaje C y en el apéndice B algunos documentos y reconocimientos relacionados con el desarrollo del presente trabajo de tesis.

CAPÍTULO 1
ALAMBRES FERROMAGNÉTICOS AMORFOS

En este capítulo se mencionan brevemente conceptos básicos de materiales magnéticos y se introduce al lector al conocimiento de los alambres amorfos magneto–elásticos, con el objetivo de fundamentar su aplicación en el desarrollo del presente trabajo. A continuación, se describen las características más importantes de los alambres magneto–elásticos y se mencionan algunos datos importantes sobre ellos.

1.1 Propiedades magnéticas de la materia

Toda la materia está compuesta fundamentalmente de átomos y cada átomo está formado por electrones en movimiento alrededor del núcleo. El movimiento de cada electrón genera una corriente atómica. En un material se tienen entonces dos tipos de corrientes: una corriente macroscópica que circula por el material y transporta carga neta, y las ya mencionadas corrientes atómicas que son corrientes que circulan sin proporcionar transporte de carga. Ambas corrientes producen campos magnéticos [8].

Cada corriente atómica puede describirse como un dipolo magnético con dos polos magnéticos iguales pero opuestos en signo, convencionalmente, un polo norte y un polo sur. Sumando vectorialmente todos los dipolos magnéticos de un material y dividiendo entre su volumen se obtiene el momento magnético por unidad de volumen o magnetización (**M**).

Cuando a un material se le aplica un campo magnético externo (**H**), se puede definir la susceptibilidad magnética (χ_m) como sigue:

$$\chi_m = \frac{M}{H} \qquad \text{.....(1.1)}$$

Esta χ_m permite clasificar a los materiales en tres grandes grupos: diamagnéticos, paramagnéticos y ferromagnéticos. Los materiales diamagnéticos se caracterizan por tener una χ_m negativa y muy pequeña. Esto se debe a que al aplicar un campo magnético externo, los momentos magnéticos varían oponiéndose al campo, es decir, se debilita la acción del campo aplicado. Este comportamiento es el

resultado de la ley de Lenz operando a nivel atómico. En los materiales paramagnéticos, la χ_m es positiva pero muy pequeña, pues muchos de los momentos magnéticos tratan de alinearse en la dirección del campo pero la agitación térmica lo impide. En los materiales ferromagnéticos, la χ_m es positiva y muy grande, por lo que se observan grandes variaciones de la magnetización al aplicar un campo magnético externo pequeño. Estos últimos materiales son ampliamente utilizados para aplicaciones tecnológicas debido a la facilidad con que producen variaciones de magnetización.

1.2 Alambres ferromagnéticos

Debido a la enorme variedad de las aplicaciones que tienen los materiales ferromagnéticos, éstos se pueden obtener en formas igualmente variadas: desde grandes bloques de aleaciones metálicas de Fe-Si para núcleos de transformadores, hasta imanes minúsculos de Al-Ni-Co ó Fe-Nd-B para bocinas o motores de paso de dimensiones reducidas. Recientemente [7] se han preparado materiales ferromagnéticos en forma de alambres metálicos amorfos muy delgados, del orden de 120 micras de diámetro (Fig. 1.1) con propiedades magnéticas excepcionales que los habilitan como sensores muy versátiles. Estas propiedades magnéticas son básicamente la biestabilidad y la magnetostricción.

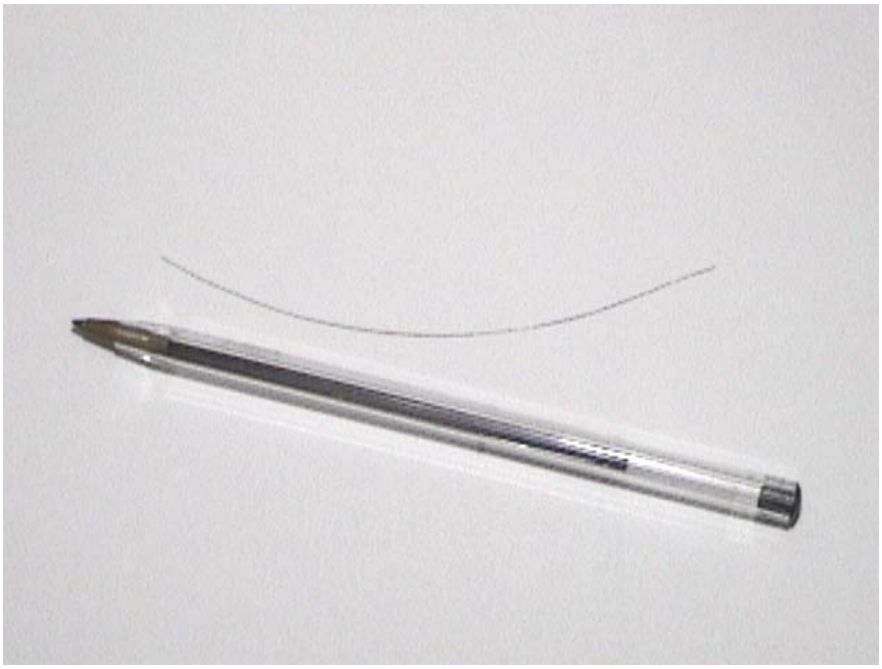

Fig. 1.1. Alambre ferromagnético junto a una pluma.

La biestabilidad consiste en un cambio súbito de la orientación de la magnetización del alambre cuando se le aplica un campo crítico (**Hc**). Por su parte, la magnetostricción (λ_s) consiste en la capacidad del alambre para cambiar la orientación de su magnetización cuando se somete a esfuerzos de tensión–compresión que modifican su longitud. Ambas propiedades son resultado de la peculiar estructura de dominios magnéticos que caracterizan a estos alambres: un núcleo central con magnetización orientada axialmente y una capa que lo rodea, cuya magnetización se orienta en forma radial como se muestra en la Fig.1.2.

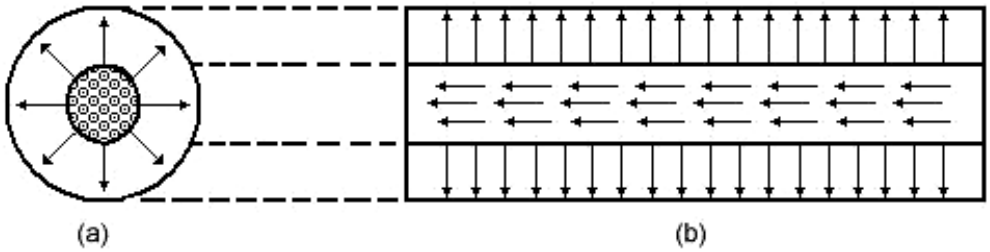

Figura 1.2. Estructura interna de los dominios magnéticos de los alambres ferromagnéticos: sección transversal (a) y sección longitudinal (b).

La magnetostricción de estos alambres permite utilizarlos como transductores de esfuerzos tensión–compresión a señales eléctricas, ya que dichos esfuerzos provocan un cambio de flujo magnético que puede detectarse como voltaje inducido en las terminales de una bobina de recolección.

Las composiciones químicas de interés que caracterizan a los alambres ferromagnéticos tienen la fórmula general $(Fe,Co,Ni)_XSi_YB_Z$, donde el contenido del metal de transición está comprendido entre x=70 y x=80%, mientras que la concentración de metaloides oscila entre el y=z=10% y y=z=20%. Se pueden añadir otros elementos para mejorar algunas propiedades físicas en particular, como el Cr, que aumenta la resistencia a la corrosión [7]. La tabla 1.1 muestra algunas propiedades físicas para alambres ferromagnéticos de diferente composición química. En el presente trabajo se usó el alambre $Fe_{77.5}Si_{10}B_{12.5}$, que posee la magnetostricción más elevada y por lo tanto, la mejor capacidad para actuar como transductor mecánico–magnético.

Composición	$\rho(g/cm^3)$	$E(GN/m^2)$	$v_L(km/s)$	Ms(T)	$10^6\lambda_s$
$Fe_{77.5}Si_{10}B_{12.5}$	7.21	164	4.77	1.6	35
$Co_{72.5}Si_{12.5}B_{15}$	7.74	174	4.73	0.64	-5.6
$(Fe_{0.06}Co_{0.94})_{72.5}Si_{12.5}B_{15}$	7.70	173	4.74	08	-0.08

Tabla 1.1. Propiedades físicas para alambres ferromagnéticos de diferente composición química.

Donde:

P es la densidad del material.
E es el módulo de Young.
$\mathbf{v_L}$ es la velocidad del sonido en el material.
$\mathbf{M_s}$ es la magnetización.
λ_s es la magnetostricción.

Se han propuesto diferentes aplicaciones de los alambres con propiedades magnetostrictivas, tales como el desarrollo de micromotores cuyo funcionamiento se basa en un fenómeno descubierto recientemente, el cual consiste en una rotación espontánea del alambre cuando se le aplica un campo magnético de alta frecuencia.

Otra aplicación que ha sido propuesta es la “pluma magneto-elástica” para identificación de firmas autógrafas [7], la cual basa su funcionamiento en la magnetostricción de los alambres amorfos. En el presente trabajo se retoma y modifica esta idea, para diseñar e implementar un dispositivo capaz de reconocer una firma autógrafa.

CAPÍTULO 2
BASES TEÓRICAS PARA EL DESARROLLO DEL SISTEMA

En este capítulo se realiza una descripción general de la teoría empleada para el desarrollo del sistema de reconocimiento de firma autógrafa. Se tratan conceptos básicos del amplificador operacional y del procesamiento digital de señales, y se describen técnicas y algoritmos para el reconocimiento de patrones.

2.1 Amplificador operacional

Los amplificadores operacionales tienen una estructura interna compleja pero no es necesario conocer mucho sobre ésta para poder utilizarlos. Algunas de las características del amplificador operacional ideal, mostrado en la Fig. 2.1, son: ganancia infinita, respuesta en frecuencia infinita, impedancia de entrada infinita, impedancia de salida de cero y terminales de entrada que no toman corriente de señal ni de polarización [9].

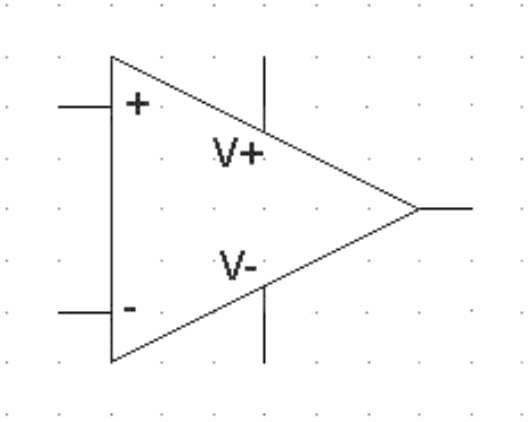

Fig. 2.1. Amplificador operacional ideal.

El circuito de la Fig. 2.2 se denomina seguidor de voltaje. El voltaje de entrada se aplica directamente en la entrada no inversora. Debido a que el voltaje entre las terminales inversora y no inversora puede considerarse cero, el voltaje de salida es igual al voltaje de entrada [9]. El seguidor de voltaje se utiliza debido a que su resistencia de entrada es alta, por lo que requiere de una corriente despreciable de la fuente de señal para su operación. Si se debe amplificar una fuente de señal y no se desea tomar corriente de ella, se debe incluir una etapa intermedia por medio de un seguidor de voltaje [9].

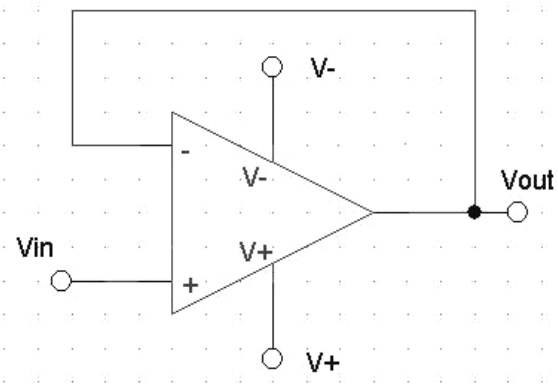

Fig. 2.2. Circuito seguidor de voltaje con amplificador operacional.

En la Fig. 2.3 se muestra el diagrama de un amplificador inversor, es decir, el voltaje de salida (**Vo**) es opuesto al voltaje de entrada (**Vin**), en lo que se refiere a la polarización. La resistencia de entrada del amplificador es $\mathbf{R_2}$, cuyo valor puede ser grande o pequeño. El voltaje de salida se calcula mediante la Ec. 2.1.

$$Vo = -\frac{R_1}{R_2} Vin \quad \text{.....(2.1)}$$

Se puede observar que la magnitud de la señal de salida depende de la relación $\mathbf{R_1/R_2}$, de ahí que se pueda hacer un amplificador de ganancia variable al variar el valor de $\mathbf{R_1}$. Hay que tener cuidado al escoger el valor de $\mathbf{R_2}$, ya que un valor pequeño puede significar una carga excesiva para la fuente de señal, por este motivo se debe tomar un valor grande de $\mathbf{R_2}$ o colocar un acoplamiento intermedio entre la fuente de señal y el amplificador mediante un seguidor de voltaje [9].

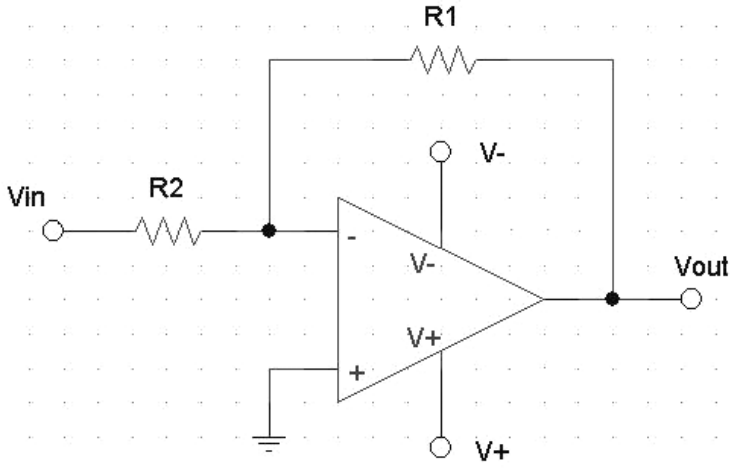

Fig. 2.3. Amplificador inversor.

2.2 Segmentación

El objetivo de procesar una señal es obtener una conveniente o más útil representación de la información contenida en ella [10]. La precisión en la representación es determinada por la información de la señal que se desea preservar o destacar. Por ejemplo, el propósito del procesamiento digital de una señal puede ser el determinar qué partes de ella contienen información útil [10]. En la Fig. 2.4 se muestra un ejemplo de una señal proporcional a la firma de una persona, obtenida con el sistema desarrollado en el presente trabajo. En dicha figura se puede observar que las propiedades de la señal cambian con el paso del tiempo. Debido a esto, nacen los métodos de procesamiento en tiempo corto, en los cuales, segmentos cortos de la señal son separados y procesados individualmente. El resultado de procesar uno de esos segmentos puede arrojar un número o un conjunto de números. Tal proceso genera una nueva secuencia de valores que dependen del tiempo y además pueden servir como una representación de la señal [10]. Al proceso de analizar a una señal por segmentos se le llama segmentación.

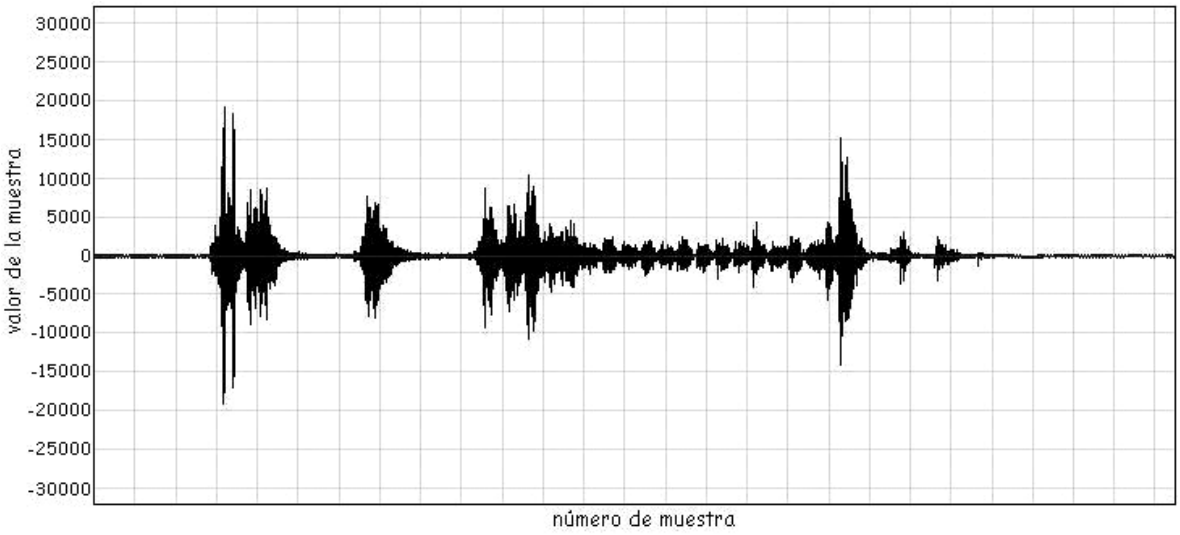

Fig. 2.4. Señal variante en el tiempo.

Los procesos en tiempo corto (**Qn**) pueden ser matemáticamente representados con la Ec. 2.2, en donde la señal discreta **x**(**n**) está sujeta a cierta transformación (**T**), mientras que **n** representa al tiempo discreto como variable independiente. La secuencia resultante es multiplicada por una ventana **w**(**n**) posicionada en un tiempo correspondiente a la muestra **n**. La Ec. 2.2 corresponde a una convolución

discreta de la ventana **w**(**n**) con la secuencia **T**[**x**(**n**)], por lo que **Qn** se puede interpretar como la salida de un sistema con respuesta al impulso **h**(**n**) = **w**(**n**) [10].

$$Qn = \sum_{m=-\infty}^{\infty} T[x(m)]\, w(n-m) \quad \text{.....(2.2)}$$

2.3 Tasa de cruces por cero de una señal

En el estudio de señales discretas se dice que un cruce por cero ocurre si muestras consecutivas tienen diferente signo. La tasa de cruces por cero es una técnica empleada para determinar las componentes de frecuencia con mayor peso contenidas en una señal [10]. Esta técnica puede servir para determinar si un segmento de la señal contiene esencialmente información o únicamente ruido. Se define la tasa de cruces por cero (**Zn**) por medio de: [10].

$$Zn = \sum_{m=-\infty}^{\infty} \left| signo[x(m)] - signo[x(m-1)] \right| w(n-m) \quad \text{.....(2.3)}$$

Donde:

$$signo[x(n)] = \begin{cases} 1 & x(n) \geq 0 \\ -1 & x(n) < 0 \end{cases} \qquad w(n) = \begin{cases} 1/(N-1) & 0 \leq n \leq N-1 \\ 0 & otro\ caso \end{cases}$$

2.4 Magnitud promedio de una señal

En la Fig. 2.4, se observa que la amplitud de la señal cambia considerablemente con el tiempo. En particular, la magnitud de las muestras contenidas en segmentos que contienen información es mucho mayor que las de las muestras contenidas en segmentos carentes de información. La energía en tiempo corto de una señal (**En**), dada por la Ec. 2.4, proporciona una forma conveniente de representar los efectos ocasionados por la variación de la amplitud de una señal [10].

$$En = \sum_{m=-\infty}^{\infty} x^2(m) w(n-m) \quad \text{.....(2.4)}$$

La dificultad de utilizar la Ec. 2.4 se debe a que es muy sensible a valores grandes y resulta inadecuada para ser programada en una computadora. Un camino sencillo para obtener un parámetro que sea similar a la energía en tiempo corto es el de estimar la magnitud promedio en tiempo corto (**Mn**) dada por la Ec. 2.5 [10], donde la suma de los valores absolutos de la señal es similar a la suma de sus cuadrados.

$$Mn = \sum_{m=-\infty}^{\infty} |x(m)| w(n-m) \quad \text{.....(2.5)}$$

Los efectos de la ventana **w**(**n**) en la representación de la magnitud en tiempo corto se pueden determinar analizando a la propia ventana. La Ec. 2.5 es el resultado de filtrar al valor absoluto de **x**(**m**) con un sistema cuya respuesta al impulso (**h**(**n**)) es **w**(**n**). En un caso particular **w**(**n**) puede ser una ventana cuadrada cuya respuesta en frecuencia **H**(ω) (Fig. 2.5) representa a un filtro pasa bajas (Ec. 2.6), donde ω representa la frecuencia angular, **T** es el periodo de muestreo de la señal, **N** es el número de elementos de la ventana y **j** es la base de los números complejos [10].

$$H(\omega) = \frac{sen(\omega TN/2)}{sen(\omega T/2)} e^{-j\omega T(N-1)/2} \quad \text{.....(2.6)}$$

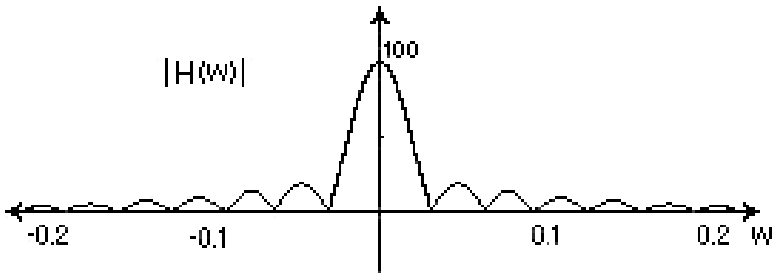

Fig. 2.5. Magnitud de la respuesta en frecuencia
de la Ec. 2.6, para T=2 y N=100.

La primera ocasión en que **H**(ω) toma el valor cero ocurre a una frecuencia $\mathbf{f_0}$=**fs**/**N**, donde **fs** = 1/**T** es la frecuencia de muestreo de la señal. Normalmente $\mathbf{f_0}$ es la frecuencia de corte del filtro digital pasa bajas y depende de **N**, es decir, del número de elementos de **h**(**n**), o sea, del tamaño de la ventana **w**(**n**). Al calcular la magnitud promedio en tiempo corto se está filtrando a la señal, es decir, **Mn**

resulta afectada por aquellas señales que contienen información ya que éstas son de baja frecuencia. Por otro lado, señales de alta frecuencia (como el ruido) son atenuadas y no afectan significativamente al valor de **Mn**. Por esto, **Mn** es un buen parámetro para determinar cuáles segmentos de la señal contienen información y cuáles no [10]. La frecuencia de corte $\mathbf{f_0}$ depende del tamaño de la ventana. Si **N** es muy pequeña, se tendrá una frecuencia de corte alta y **Mn** estará afectada tanto por señales de ruido como de información. Si **N** es muy grande, la frecuencia de corte será baja, por lo que **Mn** cambiará muy poco y no dará una buena representación del cambio de las propiedades de la señal que contiene información [10]. Una forma de determinar el tamaño y tipo de la ventana es determinando las características del filtro digital pasa bajas, de modo que **Mn** sea afectada esencialmente por señales que contengan información.

2.5 Filtros digitales FIR

Un filtro, en términos generales, es un sistema con un conjunto de entradas y un conjunto de salidas. El sistema contiene una forma de procesamiento que genera las salidas a partir de las entradas. Para que se justifique la existencia del filtro la salida o salidas deben ser de algún modo, más útiles que la o las entradas. Existen dos fuertes motivaciones para efectuar operaciones de filtrado: una es la de mejorar la calidad de las señales de entrada y la otra es el procesamiento o extracción de información a partir de las entradas [11].

Los filtros digitales están basados en la realización de operaciones sencillas de suma y multiplicación. Algunas características de los filtros digitales son:

- Pueden ser implementados con programas de aplicación sobre computadoras digitales de propósito general, por lo que resulta sencillo construirlos y probarlos.

- Su operación está basada en la realización de operaciones aritméticas de suma y multiplicación, por lo que son extremadamente estables, es decir, su comportamiento no se modifica ni con el tiempo ni con la temperatura [11].

Los filtros digitales se pueden englobar en dos grupos: filtros realimentados y filtros no realimentados. Un filtro no realimentado genera su salida a partir de la suma de la entrada presente y **m** entradas que le preceden, ponderadas por un conjunto de **m**+**1** constantes denominadas coeficientes del filtro, siendo estos los

responsables de su forma de operar [11]. La forma general de un filtro no realimentado se muestra en la Ec. 2.7, donde los **a**(**i**) son los coeficientes de ponderación de la señal **x**(**n**) para obtener la salida **y**(**n**). Si **x**(**n**) es un impulso, la respuesta impulsiva del filtro no realimentado equivaldrá a los coeficientes del filtro [11]. La relación anterior hace que la respuesta al impulso sea una forma natural para la descripción de los filtros no realimentados.

$$y(n) = \sum_{i=0}^{m} a(i)x(n-i) \quad \text{.....(2.7)}$$

A los filtros no realimentados se les denomina filtros de respuesta impulsiva finita o filtros FIR (por sus siglas en inglés Finite Impulse Response) [11]. Los filtros digitales FIR presentan las siguientes características: no requieren de salidas anteriores para el cálculo de la salida actual, son de fase lineal, su comportamiento no se modifica con el paso del tiempo o los cambios de temperatura y requieren de un alto orden para ser implementados [11].

La respuesta en frecuencia de un filtro FIR (Ec. 2.8) queda determinada totalmente por los coeficientes del filtro. La respuesta en frecuencia de un filtro digital es una función periódica, con periodo 2 , de manera que basta trabajar sobre un intervalo de este tamaño.

$$H(\omega) = \sum_{i=-m}^{m} h(i)e^{-j\omega i} \quad \text{.....(2.8)}$$

Una forma de diseñar filtros FIR requiere la especificación de la respuesta en frecuencia deseada para determinar los coeficientes del filtro. La Ec. 2.9 sirve para calcular los coeficientes del filtro en términos de la respuesta en frecuencia.

$$h(n) = \frac{1}{2\pi}\int_{-\omega_0}^{\omega_0} H(\omega)e^{j\omega n}d\omega \quad \text{.....(2.9)}$$

La Ec. 2.10 sirve para calcular los coeficientes del filtro cuya respuesta en frecuencia se muestra en la Fig. 2.6 y que corresponde a un filtro digital pasa bajas, donde **N** es el número de coeficientes calculados y **k** es la ganancia del filtro.

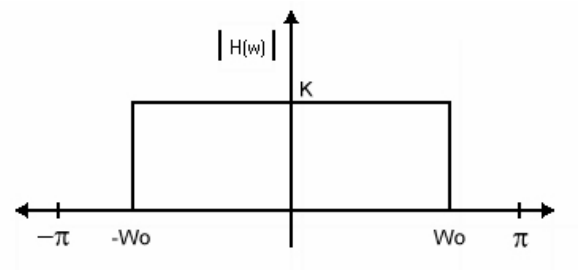

Fig. 2.6. Respuesta ideal en frecuencia de un filtro pasa bajas.

$$h(n) = \frac{k}{\pi(n-N)} sen(\omega_0 (n-N)) w(n) \quad \text{.....(2.10)}$$

La Ec. 2.11 muestra la relación entre la frecuencia digital, frecuencia analógica y la frecuencia de muestreo.

$$f_d = \frac{f_a}{f_s} \quad \text{.....(2.11)}$$

Partiendo de la Ec. 2.9, se puede obtener la expresión para un filtro digital pasa banda (Ec. 2.12) donde ω_a y ω_b son las frecuencias de corte alta y baja respectivamente del filtro pasa banda.

$$h(n) = \frac{k}{\pi n}\left(sen(\omega_a n) - sen(\omega_b n)\right) \quad \text{.....(2.12)}$$

2.6 Ventaneo

Truncar una serie infinita de términos de una señal se puede realizar con la ayuda de una función ventana, es decir, se puede pensar que los elementos de la señal son “vistos” a través de una ranura. Solamente los elementos que son “vistos” son utilizados para el análisis que se desee realizar. El truncamiento por una función ventana rectangular es una operación abrupta, dependiendo de la naturaleza de la señal truncada. Cuando se hace el truncamiento de los coeficientes de un filtro digital por medio de una función ventana, se realiza una multiplicación en el dominio del tiempo, lo que equivale a una convolución en el dominio de la frecuencia. La respuesta en frecuencia del filtro es convolucionada con la respuesta en frecuencia de la ventana. Para evitar que la respuesta en frecuencia del filtro cambie significativamente por el truncamiento, se utiliza una ventana cuya

respuesta en frecuencia tenga la mayor parte de su energía cerca de la frecuencia cero [12]. Se busca que la respuesta en frecuencia de la función ventana se asemeje lo más que se pueda a un impulso, pues al convolucionarlo con una señal, el resultado es la misma señal [12]. Algunas ventanas que mejoran la respuesta en frecuencia del filtro con coeficientes truncados se muestran a continuación:

- Ventana rectangular.

$$w(n) = \begin{cases} 1 & para \quad 0 \le n \le N-1 \\ 0 & otro\ caso \end{cases}$$

- Ventana de Bartlett.

$$w(n) = \begin{cases} \left(\frac{2}{N-1}\right)n & 0 \le n \le \frac{N-1}{2} \\ 2-\left(\frac{2}{N-1}\right)n & \frac{N-1}{2} < n \le N-1 \\ 0 & otro\,caso \end{cases}$$

- Ventana de Hamming.

$$w(n) = \begin{cases} 0.54 - 0.46\cos\left(2n\pi/N-1\right) & 0 \le n \le N-1 \\ 0 & otro\,caso \end{cases}$$

- Ventana de Hanning.

$$w(n) = \begin{cases} 0.5 - 0.5\cos\left(2n\pi/N-1\right) & 0 \le n \le N-1 \\ 0 & otro\,caso \end{cases}$$

- Ventana de Blackman.

$$w(n) = \begin{cases} 0.42 + 0.08\cos\left(4n\pi/N-1\right) - 0.5\cos\left(2n\pi/N-1\right) & 0 \le n \le N-1 \\ 0 & otro\,caso \end{cases}$$

En la tabla 2.1 se indican algunas características de su respectiva respuesta en frecuencia [12].

Tipo de ventana	Transición del lóbulo principal (N=tam vent)	Atenuación del lóbulo principal [dB]
Rectangular	4π/N	-13
Bartlett	8π/N	-27
Hanning	8π/N	-32
Hamming	8π/N	-43
Blackman	12π/N	-58

Tabla 2.1. Bandas de transición y atenuación de diferentes ventanas.

2.7 Análisis de predicción lineal

El análisis de predicción lineal LPC (por sus siglas en inglés Linear Predictive Coding) es una de las técnicas más utilizadas para analizar señales [10]. Este análisis permite representar una señal de forma precisa y eficiente mediante pocos parámetros obtenidos con cálculos sencillos. A esto se le conoce como extracción de parámetros, la cual consiste en obtener una serie de características de una señal tales que permitan identificarla con relativa facilidad [10].

La idea básica detrás de la técnica LPC consiste en que el valor de una muestra (**S**[**n**]) de la señal puede ser aproximado a partir de una combinación lineal del valor de sus muestras anteriores (**S´**[**n**]), ponderados por coeficientes de peso ($\mathbf{a_i}$). Este hecho se puede expresar de la forma mostrada en la Ec. 2.13, donde **p** es el orden del predictor [10].

$$S'[n] = a_1 S[n-1] + a_2 S[n-2] + ... + a_p S[n-p] \quad \text{.....(2.13)}$$

El problema es determinar el valor de los coeficientes de modo que **S**'[**n**] sea lo más parecida a **S**[**n**]. Para determinar el valor de estos coeficientes, se debe hallar una forma de medir el error que producen los mismos al estimar el valor de un conjunto de muestras. Una forma de medirlo es sumando todas las diferencias entre los valores estimados y los reales, pero ésta no es una buena forma de medir el error pues las diferencias pueden tener signos opuestos y anularse entre sí. Otra forma de medir el error es sumando el valor absoluto de las diferencias entre las muestras estimadas y sus valores reales. De este modo, al final se

obtiene un número que muestra una idea acerca de la calidad de la predicción, pero el uso del valor absoluto dificulta la obtención de los coeficientes LPC. Por lo anterior, se determinó que la mejor forma de medir el error es sumando el cuadrado de las diferencias entre las muestras estimadas y sus valores reales. De este modo, se trabaja con una expresión que indica el valor del error de estimación, y como es una función cuadrática, es fácil trabajar con ella para determinar el valor de los coeficientes que minimicen el error. Además, por su naturaleza cuadrática, dichos valores corresponden a un error mínimo y no a un máximo [10].

Dadas **N** muestras de una señal, es necesario calcular o estimar los valores de los coeficientes, de modo que hagan una buena estimación de la señal en estudio. Definido el error (**e**[**n**]), al hacer una estimación para una muestra **S**[**n**] por medio de la Ec. 2.14, la suma del error cuadrático medio (**E**) a lo largo de una ventana finita de longitud **N**, estará dada por la Ec. 2.15.

$$e[n] = S[n] - \sum_{j=1}^{p} a_j S[n-j] \quad \text{.....(2.14)}$$

$$E = \sum_{n=0}^{N-1} e^2[n] = \sum_{n=0}^{N-1} \left(S[n] - \sum_{j=1}^{p} a_j S[n-j]\right)^2 \quad \text{.....(2.15)}$$

Derivando a la Ec. 2.15 e igualando a cero, se obtiene un conjunto de **p** ecuaciones con **p** incógnitas, siendo estas últimas iguales a los coeficientes LPC. A continuación se muestra el procedimiento para determinar el valor de los coeficientes que minimizan el error cuadrático de la estimación [10].

$$\frac{\partial E}{\partial a_i} = 0 \qquad \Rightarrow$$

$$\Rightarrow \frac{\partial}{\partial a_i}\left(\sum_{n=0}^{N-1} \left(S[n] - \sum_{j=1}^{p} a_j S[n-j]\right)^2\right) = 0$$

Desarrollando la derivada, se tiene:

$$\Rightarrow \sum_{n=0}^{N-1} \frac{\partial}{\partial a_i}\left(S^2[n] - 2S[n]\sum_{j=1}^{p} a_j S[n-j] + \left(\sum_{j=1}^{p} a_j S[n-j]\right)^2 \right) =$$

$$= \sum_{n=0}^{N-1}\left(\frac{\partial}{\partial a_i} S^2[n] - \frac{\partial}{\partial a_i} 2S[n]\sum_{j=1}^{p} a_j S[n-j] + \frac{\partial}{\partial a_i}\left(\sum_{j=1}^{p} a_j S[n-j]\right)^2 \right) =$$

$$= \sum_{n=0}^{N-1}\left(0 - 2S[n]\frac{\partial}{\partial a_i}\sum_{j=1}^{p} a_j S[n-j] + 2\left(\sum_{j=1}^{p} a_j S[n-j]\right)\frac{\partial}{\partial a_i}\sum_{j=1}^{p} a_j S[n-j] \right) =$$

$$= \sum_{n=0}^{N-1}\left(2\sum_{j=1}^{p} a_j S[n-j]S[n-i] - 2S[n]S[n-i] \right) =$$

$$= 2\sum_{j=1}^{p} a_j \sum_{n=0}^{N-1} S[n-j]S[n-i] - 2\sum_{n=0}^{N-1} S[n]S[n-i] = 0$$

Finalmente se llega a:

$$\sum_{n=0}^{N-1} S[n-i]S[n] = \sum_{j=1}^{p} a_j \sum_{n=0}^{N-1} S[n-i]S[n-j]$$

De aquí, se define la matriz de covarianza **Φ** con elementos $\Phi_{i,j}$ como:

$$\phi_{i,j} = \sum_{n=0}^{N-1} S[n-i]S[n-j] \quad \text{.....(2.16)}$$

para 1≤i≤p ; 0≤j≤p

Por lo tanto, se puede escribir el resultado en forma reducida como:

$$\phi_{i,0} = \sum_{i=1}^{p} \phi_{i,j} \bullet a_j$$

En la Ec. 2.16 se puede observar que al calcular el elemento $\Phi_{p,p}$, se realizan productos con valores que no están dentro de las **N** muestras bajo estudio, pues se necesitan las muestras **S**(-1), **S**(-2), …, **S**(-**p**). Por tanto, para evaluar adecuadamente la matriz de covarianza se requiere usar valores de **S**(**n**) comprendidos entre -**p** ≤ **n** ≤ **N-1**. Tomando en cuenta esta consideración, se puede determinar el valor de los coeficientes por medio de la matriz de covarianza. Este procedimiento para determinar los valores de los coeficientes LPC se llame método de la covarianza [10].

2.8 Aproximación polinomial

La aproximación polinomial, más que una técnica de análisis de señales, es un método numérico para aproximar señales de naturaleza desconocida por medio de un polinomio de grado **p**, es decir, deben calcularse **p**+**1** coeficientes. El polinomio debe ser calculado de modo que el error de aproximación sea el menor posible. En general, el polinomio de grado **p** se puede expresar matemáticamente por la Ec. 2.17, donde **S**(**n**) es el polinomio aproximado y los $\mathbf{a_i}$ sus coeficientes.

$$S(n) = a_0 n^0 + a_1 n^1 + a_2 n^2 + ... + a_p n^p \quad \text{.....(2.17)}$$

Al igual que en el análisis de predicción lineal, se puede definir el error de aproximación por medio de la Ec. 2.18 y el error total de aproximación sobre **N** muestras por medio de la Ec. 2.19.

$$e[n] = S[n] - \sum_{j=0}^{p} a_j n^j \quad \text{.....(2.18)}$$

$$E = \sum_{n=0}^{N-1} e^2[n] = \sum_{n=0}^{N-1} (S[n] - \sum_{j=0}^{p} a_j n^j)^2 \quad \text{..... (2.19)}$$

Siguiendo un procedimiento similar al utilizado para determinar el valor de los coeficientes de predicción lineal, se puede obtener la expresión para determinar los coeficientes de aproximación polinomial (Ec. 2.20). De este modo se obtiene la Ec. 2.21, que genera a los elementos de la matriz y la Ec. 2.22, que genera al vector de términos independientes.

$$\sum_{n=0}^{N-1} S[n]n^i = \sum_{j=0}^{p} a_j \sum_{n=0}^{N-1} n^{(i+j)} \quad \text{.....(2.20)}$$

$$\phi_{i,j} = \sum_{n=0}^{N-1} n^{(i+j)} \quad \text{.....(2.21)}$$

$$b_i = \sum_{n=0}^{N-1} S[n]n^i \quad \text{.....(2.22)}$$

2.9 Agrupamiento

Un patrón es una característica relevante o representativa de un objeto y es útil para distinguirlo frente a otros. Si un patrón es representado por un vector, entonces se dice que un patrón **x** es similar a un patrón **y** si la distancia entre ellos es pequeña [6]. A la técnica de asociar a un conjunto de patrones que presentan un comportamiento similar entre sí se le llama agrupamiento. Cada conjunto de patrones agrupados se llama clase y cada una de ellas tiene un prototipo o valor característico llamado centroide [6].

Determinar los prototipos o centroides de cada clase es esencial en lo que se refiere al diseño de clasificadores, los cuales están basados en el cálculo de distancias mínimas. Dado un conjunto de patrones, se debe determinar una manera de agruparlos entre sí en diferentes clases [6]. El resultado de un proceso de agrupamiento es usualmente aplicado en el reconocimiento de patrones con dos objetivos: el de obtener características de la estructura geométrica de los patrones agrupados y el de diseñar funciones de decisión para futuros clasificadores en un problema de reconocimiento de patrones [6].

2.10 Distancias y medidas de similitud

Existen diferentes medidas de similitud para comparar patrones (**a**,**b**,**c**). Entre ellas se tienen la distancia euclidiana y la distancia de error cuadrático medio, cada una con sus características propias.

Para que una función pueda ser definida como una función distancia (**D**) se deben de cumplir las siguientes propiedades:

$$D(a,b) = D(b,a)$$
$$D(a,c) \leq D(a,b) + D(b,c)$$
$$D(a,a) = 0$$

- La distancia euclidiana es la medida comúnmente usada para comparar dos patrones y se define por medio de la siguiente ecuación:

$$D(a,b) = |a-b| \quad \text{..... (2.23)}$$

Donde: $|a-b| = \sqrt{(a-b)(a-b)^T}$

- La distancia de error cuadrático medio se define con la Ec. 2.24.

$$D(a,b) = \frac{1}{n}(a-b)(a-b)^T \quad \text{.....(2.24)}$$

Donde **n**, es la dimensión de los patrones y **T** indica que es un vector transpuesto. Esta medida es comúnmente utilizada debido a su fácil programación.

2.11 Algoritmo de C – medias

Dado un conjunto de patrones **X** = {$\mathbf{x_1}$, $\mathbf{x_2}$, $\mathbf{x_3}$,…,$\mathbf{x_m}$}, se asume que existen **C** grupos de patrones o clases, cuyos prototipos o centroides son inicialmente aproximados por **Z** = {$\mathbf{Z_1}$, $\mathbf{Z_2}$, $\mathbf{Z_3}$, …, $\mathbf{Z_C}$}. El proceso de buscar el valor final de los prototipos de cada clase es iterativo. En cada iteración, todos los patrones son clasificados y los prototipos son ajustados utilizando un esquema de minimización que asocia a cada patrón con otro u otros más cercanos, creando así los grupos de patrones y ajustando de una iteración tras otra al prototipo de cada grupo [6].

Significado de los símbolos utilizados en el algoritmo:

n – Dimensión de los patrones y de los centroides de las clases.
m – Número de patrones a agrupar.
c – Número de clases en que se desea agrupar a los patrones.
X = {$\mathbf{x_i}$} - Patrones de dimensión **n**, para **i**=1 hasta **m**.

N – Número máximo de iteraciones permitidas.
Z = {$\mathbf{Z_{j(k)}}$} – Prototipo de dimensión **n** de la clase **j** en la iteración **k**, para **j**=1 hasta **c**, para **k**=1 y máximo **N**.
A = {$\mathbf{a_j}$} – Número de patrones contenidos en la clase **j**, para **j**=1 hasta **c**.
L = {**li**} – Clase a la cual pertenece el patrón **i**, para **i**=1 hasta **m**.

Teniendo patrones de dimensión **n** y especificando el número de clases en las que se desean agrupar, además de especificar un valor inicial para el centroide de cada clase, se define el algoritmo de **C** medias como sigue:

Paso 1. En la primera iteración, se inicializa arbitrariamente el valor de los prototipos de cada clase. Generalmente se asigna a los **c** primeros patrones como prototipos iniciales.

$$Z_{j(1)} = x_j \qquad para \qquad j = 1 \qquad hasta \qquad c$$

Paso 2. Clasificar a cada $\mathbf{x_i}$, para **i** = 1 hasta **m**, en las clases cuyos prototipos son $\mathbf{Z_{j(k)}}$, para **j** = 1 hasta **c**, usando la siguiente condición:

En la iteración **k** el patrón $\mathbf{x_i}$, para **i**=1 hasta **m**, se asocia a la clase **j**, cuyo prototipo es $\mathbf{Z_{j(k)}}$, y se incrementa $\mathbf{a_j}$, si se cumple que:

$$D(x_i, Z_{j(k)}) \leq D(x_i, Z_{r(k)}) \qquad para \qquad j, r = 1 \qquad hasta \qquad c \qquad y \qquad r \neq j$$

Paso 3. Con los resultados obtenidos en el paso 2, calcular los nuevos prototipos de cada clase empleando la siguiente expresión para **j**=1 hasta **c**.

$$Z_{j(k+1)} = \frac{1}{a_j} \sum_{i=1}^{m} x_i \qquad tal\ que \qquad l_i = j$$

Paso 4. Si $\mathbf{Z_{j(k+1)}}$=$\mathbf{Z_{j(k)}}$, para **j**=1 hasta **c**, el algoritmo termina y los prototipos de las clases quedan definidos por **Z** = {$\mathbf{Z_j}$}, para **j**=1 hasta **c**. Si por lo menos un $\mathbf{Z_{j(k+1)}}$≠$\mathbf{Z_{j(k)}}$, para **j**=1 hasta **c**, se regresa al paso 2, haciendo la asignación $\mathbf{Z_{j(k)}}$=$\mathbf{Z_{j(k+1)}}$. Si **k** > **N** , el algoritmo termina y se dice que no converge dentro del límite especificado de iteraciones.

El comportamiento del algoritmo de **C** medias es determinado por el número de clases en que se desee agrupar a los patrones, por la asignación inicial del prototipo de cada clase, por el orden en que las muestras son tomadas y por la propia naturaleza de los datos [1]. Escoger un número adecuado de clases y el valor inicial de sus prototipos son condiciones suficientes para la convergencia del algoritmo. Cuando algún patrón se queda oscilando entre una clase y otra, de una iteración a otra, basta con cambiar el número de clases para hacer que el algoritmo converja [6].

CAPÍTULO 3
DESCRIPCIÓN Y PROGRAMACIÓN DE LA TARJETA DE SONIDO SB 16

En este capítulo se describen los pormenores de la programación de la tarjeta de sonido de la PC para ser utilizada como tarjeta de adquisición de datos. Se muestran las bondades de la tarjeta de sonido SoundBlaster 16, así como los registros y comandos de control utilizados.

3.1 Estándar SoundBlaster

Debido a la muy amplia variedad de tarjetas de sonido que hay en el mercado, existe un estándar que regula su programación para que ésta sea independiente del fabricante. Este estándar tiene el nombre de SoundBlaster y fue establecido por Creative Labs [13].

3.2 Muestreo

En el ámbito de las tarjetas de sonido se le llama muestreo a la técnica mediante la cual una señal analógica es digitalizada por medio del convertidor analógico–digital incorporado en la tarjeta de sonido de la PC. Así se pueden almacenar señales digitales en disco duro para después reproducirlas o analizarlas sin pérdida de información, salvo la que hubo en el proceso de muestreo [13]. La frecuencia de muestreo y la resolución de las muestras de la conversión analógica–digital son factores que determinan la calidad del muestreo. Lo ideal sería muestrear a espacios de tiempo infinitamente pequeños, pero esto requeriría una cantidad de memoria demasiado grande para el almacenamiento de la información. El término de resolución se refiere al número de bits utilizados para representar el valor de una muestra. En las primeras tarjetas de sonido se tenía una resolución de 8 bits, que ofrecía como máximo 256 niveles de cuantización. La tarjeta de sonido SoundBlaster 16 tiene una resolución de 16 bits, lo cual ofrece un mayor número de niveles de cuantización [13].

3.3 Procesador digital de señal (DSP)

Para realizar el muestreo de una señal analógica es necesario programar el procesador digital de señal DSP (por sus siglas en inglés Digital Signal Processor) de la tarjeta SB. Cada familia de tarjetas de sonido ha incorporado procesadores DSP más potentes, es por esto que las tarjetas de sonido se pueden diferenciar a

través de la versión del DSP con que cuentan. La consulta sobre el número de la versión del DSP constituye la única forma confiable para determinar las características del DSP que contiene la tarjeta utilizada, con la cual se determinan las bondades que ofrece. Los principales criterios para diferenciar las distintas versiones del DSP son la resolución y frecuencia de muestreo.

3.4 Modos de transferencia del DSP

El DSP soporta varios modos de muestreo que pueden ser utilizados según el problema que se quiera resolver. Algunos de esos modos utilizan el controlador de acceso directo a memoria DMA (por sus siglas en inglés Direct Memory Access) para llevar acabo la transferencia de datos entre la memoria principal de la computadora y el DSP. Otro atiende el DSP por el método de poleo. La gran ventaja de los modos DMA radica en la ejecución automática, en un segundo plano, de la transferencia de los datos entre el controlador DMA y el DSP, mientras que el procesador puede ocuparse de otras tareas [13].

a) Modo directo: En el modo directo se opera al DSP por poleo, es decir el software transfiere individualmente los bytes muestreados. La ventaja de este modo es su relativa sencillez, pero las tasas de muestreo alcanzadas no son muy altas.

b) Modo ciclo sencillo DMA: En este modo, la transferencia de datos entre el DSP y la memoria principal es efectuada a través del controlador DMA, el cual tiene que programarse junto con el DSP para cada bloque de muestras por transferir. La longitud del bloque no puede ser mayor a 64KB para un canal DMA de 8 bits y 128 KB para un canal DMA de 16 bits. El fin de una transferencia lo indica el DSP mediante la ejecución de una interrupción. Este modo es recomendable cuando se tiene que transferir un solo bloque de datos.

c) Modo auto inicialización DMA: En este modo de transferencia, el controlador DMA se inicializa automáticamente con la dirección y tamaño de bloque originales, inmediatamente después de transferir un bloque. Este modo abre la posibilidad de transferir bloques de casi cualquier tamaño. En este modo se pueden alcanzar elevadas tasas de muestreo. La técnica utilizada para programar el DSP en este modo se llama “double buffering” y está implícita en la forma en que se efectúa la transferencia del bloque de

datos. Para detener la transferencia de datos se puede ejecutar el modo de transferencia ciclo sencillo DMA o se envía al DSP un comando creado especialmente para detener el modo de auto inicialización DMA. De ambas formas, la transferencia de datos no termina hasta que no se haya transferido el último bloque de datos.

3.5 Interacción entre el controlador DMA y el DSP

En los diferentes modos de transferencia de datos, el controlador DMA y el DSP trabajan juntos. Cuando el DSP toma una señal analógica de una fuente de entrada como el Line-In, la convierte en un conjunto de muestras y las transfiere a un buffer ubicado en la memoria principal por medio del controlador DMA. Debido a las limitaciones del controlador DMA, la dirección lógica del buffer tiene que estar por debajo de 1MB. La iteración entre el controlador DMA y el DSP es controlada por medio del software de muestreo. Si deben realizarse transferencias en el modo auto inicialización DMA el software primero debe programar el controlador DMA, especificando la dirección inicial del buffer, la longitud de la transferencia y el correspondiente canal del controlador DMA con el modo de operación. Con esto se concluye su programación y las demás configuraciones se hacen en el DSP. Como el controlador DMA asume el control de la memoria para el DSP, no es necesario configurar ninguna dirección de memoria en éste, en su lugar, el DSP espera la frecuencia de muestreo y el tamaño del bloque a transferir. Una vez configurados estos dos parámetros, el software de muestreo le envía al DSP un comando especial que le permite empezar con la grabación de las muestras. Para que el software de muestreo pueda saber cuándo concluye la transferencia, el DSP ejecuta una interrupción después de procesar la última muestra [13].

3.6 Técnica del “Double Buffering”

En el caso del Double Buffering (Fig. 3.1) se trata de convertir a la PC en un sistema eficiente de adquisición de datos por medio de la tarjeta de sonido. Primero se ubica en la memoria principal un buffer de transferencia que sea capaz de contener los datos muestreados por el DSP. Antes de iniciar el muestreo se programa al controlador DMA con la dirección y longitud del bloque y se cambia al modo auto inicialización DMA. La técnica del Double Buffering consiste en que el DSP no se programa con la longitud completa del bloque, sino con la mitad de su longitud. Cuando se inicie el muestreo, el DSP colocará las muestras en la primera

mitad del buffer a través del controlador DMA y seguidamente ejecutará una interrupción, la cual es aprovechada por el software para almacenar las muestras en archivo. Además, dentro del manejador de interrupciones se le indicará al DSP que coloque nuevamente datos muestreados, otra vez de la mitad de la longitud del buffer. Se escribirá ahora en la segunda mitad del buffer debido a que el controlador DMA no ha sido reprogramado y ha confirmado la primera mitad de la longitud de la transferencia que se le configuró. Mientras que el DSP escribe en segundo plano la segunda mitad del buffer, con ayuda del controlador DMA, la primera mitad del buffer es leída por el software de muestreo y almacenada en disco duro. Del mismo modo, cuando el DSP ha completado la segunda mitad del buffer, el controlador DMA salta de nuevo al inicio del buffer (gracias al modo auto inicialización DMA) para comenzar nuevamente con el llenado de la primera mitad mientras que la segunda es almacenada en archivo.

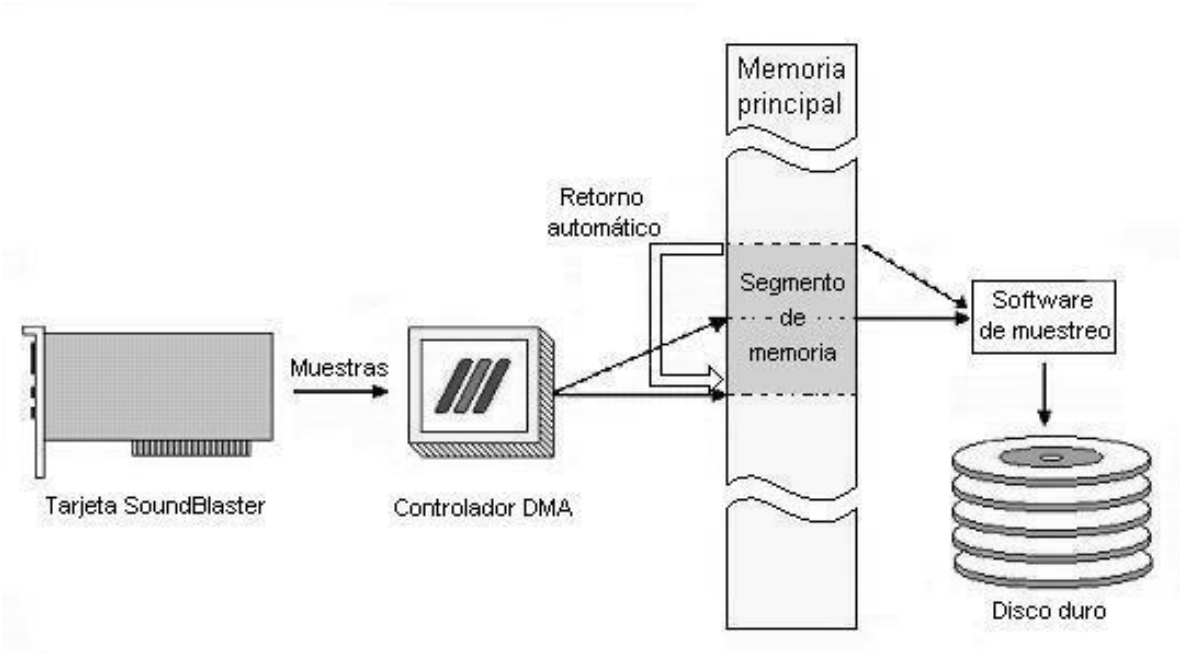

Fig. 3.1. La técnica del “Double Buffering”.

El factor para utilizar con éxito esta técnica radica en el hecho de tener un sistema lo suficientemente rápido para guardar los datos de la mitad del buffer que no está siendo procesada por el DSP [13].

3.7 Estructura de los datos del muestreo

El DSP coloca los datos muestreados en memoria siguiendo un esquema definido. La secuencia exacta de las distintas muestras depende de si el muestreo es de 8

o 16 bits y de uno o dos canales. En vinculación con los distintos comandos para leer y reproducir muestras, el DSP soporta muestras con y sin signo. Las muestras de 8 bits se manejan sin signo, mientras que las de 16 bits soportan ambos formatos. En un muestreo a 8 bits, las muestras presentan valores desde 00h hasta FFh, y en uno a 16 bits sin signo desde 0000h hasta FFFFh. En el muestreo a 16 bits con signo, son posibles tanto números positivos como negativos, por lo que el valor de una muestra es representado en un intervalo que va desde –32768 a 32767, o bien, de 8000h a 7FFFh. En la Fig. 3.2 se muestra la estructura de los datos muestreados en memoria.

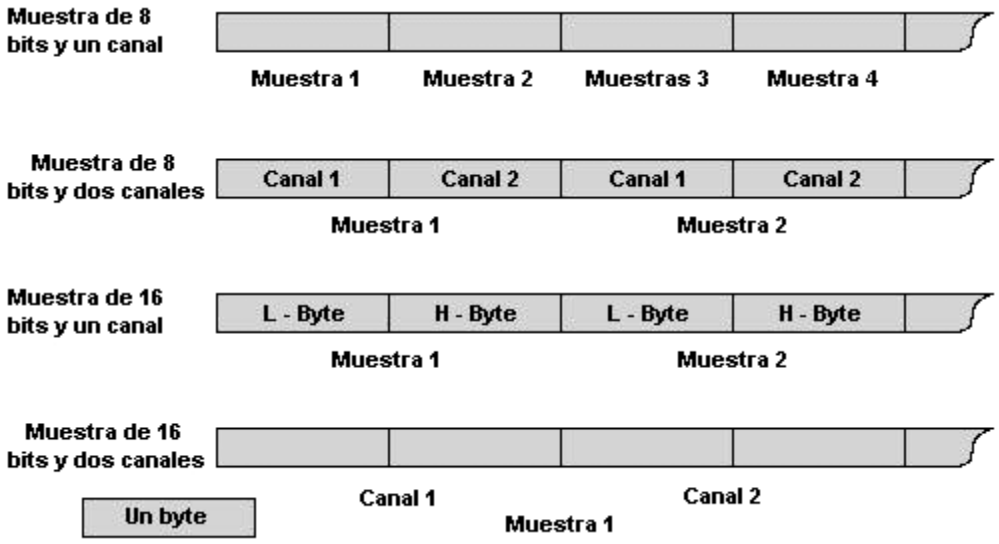

Fig. 3.2. Estructura de los datos muestreados en memoria.

3.8 Registros de acceso al DSP

El control del DSP se realiza por medio de cinco registros, los que se describen en la tabla 3.1. Sus direcciones se presentan de forma relativa a la dirección base de la tarjeta SoundBlaster.

Puerto = DirBase +	Nombre	Modo	Descripción
06h	Reset	Escritura	Reinicia el DSP
0Ah	Read Data	Lectura	Lee datos del DSP

0Ch	Write Command / Data	Escritura	Escribe datos y comandos en el DSP
0Ch	Write Buffer-Status	Lectura	Indica si el DSP está listo para recibir datos o comandos
0Eh	Read Buffer-Status	Lectura	Indica si los datos del DSP pueden ser leídos

Tabla 3.1. Registros de acceso al DSP.

3.9 Manejador de interrupciones

Un papel importante para la manipulación de interrupciones del DSP lo juega el Read Buffer Status Register. Si después del procesamiento de un bloque de muestras, el DSP ejecuta una interrupción, el manejador de interrupciones, instalado por un programa, debe mostrarle al DSP que una interrupción ha sido interceptada por el software. Para el DSP de la tarjeta SoundBlaster 16 es necesario determinar la fuente de la interrupción, pues ésta puede provenir de una transferencia de muestras de 8 bits o de 16 bits. El manejador de interrupciones tiene que leer primero el registro 82h del mezclador de la tarjeta SB, donde se encuentra la información correspondiente a la fuente de la interrupción. El acceso a los registros internos del mezclador de la SB se hace por medio de un puerto de datos y uno de direcciones. Por el puerto de direcciones se tiene que especificar el número del registro deseado del mezclador antes de leer o escribir en él. En la tabla 3.2, se muestra la dirección de los puertos de datos y de direcciones del mezclador en relación a la dirección base de la tarjeta SB [13].

Puerto de direcciones del mezclador	**=**	**DirBase + 04h**
Puerto de datos del mezclador	**=**	**DirBase + 05h**

Tabla 3.2. Dirección de los puertos de datos y de direcciones.

Especificando primero el valor 82h en el puerto de direcciones del mezclador y leyendo a continuación el puerto de datos se obtiene el contenido del Interrup Status Register de la tarjeta SB. Si al leer este registro se verifica que la interrupción fue realizada debido a una transferencia de 8 bits, entonces se tiene que leer el Read Buffer Status Register, pero si se trata de una transferencia de 16 bits, tiene que ser válido el acceso de lectura del registro DirBase + 0Fh. Sólo de

esta forma el DSP recibe la confirmación de que la interrupción ha sido recibida por el manejador de interrupciones [13]. Como la interrupción de la SB es por hardware, antes de regresar al programa interrumpido se debe enviar la confirmación al controlador de interrupciones para indicarle que ésta ha sido atendida.

3.10 Comandos del DSP

El DSP dispone de alrededor de 40 comandos diferentes, de los cuales la mayoría ya existen desde la versión 1.0. Casi la mitad de ellos sirven para la lectura y reproducción de las muestras y los restantes complementan esos comandos y aportan funciones auxiliares como activar y desactivar la vinculación entre los elementos de la tarjeta de sonido [13]. La tabla 3.3 brinda una visión general sobre los comandos disponibles del DSP de una tarjeta SoundBlaster 16 para un muestreo a 16 bits.

Código	Descripción
42h	Configura la frecuencia de muestreo de entrada
48h	Configura la longitud del bloque para una transferencia en modo auto inicialización DMA
A0h	Configura la entrada a un solo canal
Bxh	Muestreo a 16 bits DMA Entrada / Salida
D5h	Detiene transferencia DMA de muestras de 16 bits
D9h	Termina transferencia de muestras de 16 bits en modo auto inicialización DMA
E1h	Obtiene la versión del DSP

Tabla 3.3. Comandos del DSP de la tarjeta SB 16 para muestreo a 16 bits.

3.11 Muestreo a 16 bits en modo auto inicialización DMA

Los procesos de lectura y reproducción de muestras han sido simplificados en la versión 4.xx del DSP con el comando Bxh, que permite realizar muestreos a 16 bits. El código de comando se forma mediante la especificación de la dirección de la transferencia, el modo DMA deseado y la utilización del buffer FIFO, siguiendo la estructura mostrada en la Fig. 3.3a. En el modo de operación debe especificarse si el muestreo será de uno o dos canales y si las muestras son representadas con o sin signo (Fig. 3.3b). Finalmente debe especificarse el número de muestras –1, enviando primero la parte baja y luego la parte alta. Antes

de la llamada al comando debe configurarse la frecuencia de muestreo y programarse el controlador DMA con la dirección inicial y la longitud del bloque. Además, debe configurarse previamente en el DSP la longitud del bloque a través del comando 48h.

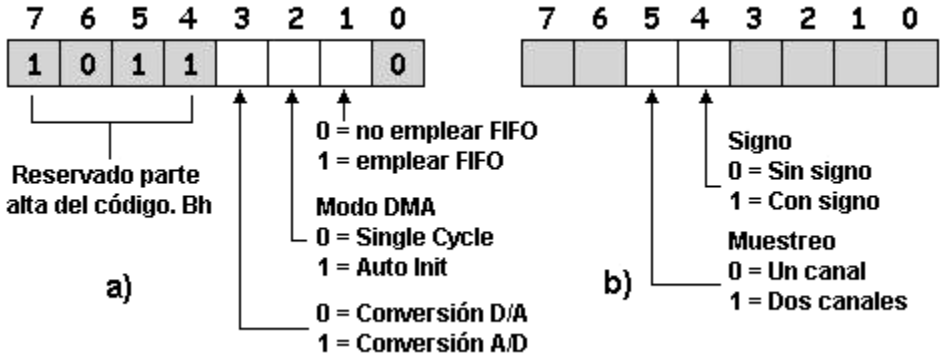

Fig. 3.3. Estructura del código de comando (a)
y estructura del modo de operación (b).

3.12 El mezclador

El mezclador tiene la tarea de controlar las distintas entradas y salidas de la tarjeta de sonido, de definir la vinculación de las distintas fuentes y habilitar las señales de entrada para el muestreo [13]. Se accede a los registros del mezclador a través de un puerto para datos y uno para direcciones. A través del puerto de direcciones, se debe especificar el registro deseado del mezclador antes de leer o escribir su contenido. La dirección de los puertos de datos y de direcciones del mezclador, en relación a la dirección base de la tarjeta SB, fue mostrada anteriormente en la tabla 3.2. Después de enviar el número de registro deseado al puerto de direcciones del mezclador, será necesario realizar un acceso de lectura al puerto de datos si el registro especificado debe ser leído o un acceso de escritura si debe colocarse un nuevo contenido. La tabla 3.4 proporciona una descripción general de los registros del mezclador de la tarjeta SoundBlaster 16.

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
00h	Reinicio del mezclador							
04h	Voice Volumen Izquierdo				Voice Volumen Derecho			
0Ah						Mic Volumen		
22h	Master Volumen Izquierdo				Master Volumen Derecho			
26h	MIDI Volumen Izquierdo				MIDI Volumen Derecho			
28h	CD Volumen Izquierdo				CD Volumen Derecho			
2Eh	Line In Volumen Izquierdo				Line In Volumen Derecho			
30h	Master Volumen Izquierdo							
31h	Master Volumen Derecho							
32h	Voice Volumen Izquierdo							
33h	Voice Volumen Derecho							
34h	MIDI Volumen Izquierdo							
35h	MIDI Volumen Derecho							
36h	CD Volumen Izquierdo							
37h	CD Volumen Derecho							
38h	Line In Volumen Izquierdo							
39h	Line In Volumen Derecho							
3Ah	Mic Volumen							
3Bh	PC Altavoz Volumen							
3Ch	Salida			L-In Izq	L-In Der	CD Izq	CD Der	Mic
3Dh	Ent Izq	MIDI Izq	MIDI Der	L-In Izq	L-In Der	CD Izq	CD Der	Mic
3Eh	Ent Der	MIDI Izq	MIDI Der	L-In Izq	L-In Der	CD Izq	CD Der	Mic
3Fh	Amp entrada Izq							
40h	Amp entrada Der							
41h	Amp salida Izq							
42h	Amp salida Der							
43h								Amp Mic
44h	Agudos Izq							
45h	Agudos Der							
46h	Bajos Izq							
47h	Bajos Der							

Tabla 3.4. Registros del mezclador de la tarjeta SB 16.

CAPÍTULO 4
ANÁLISIS, DISEÑO Y DESARROLLO DEL SISTEMA DE RECONOCIMIENTO DE FIRMA AUTÓGRAFA

En el presente capítulo se muestran detalladamente las etapas del sistema de reconocimiento de firma autógrafa: generación de la señal eléctrica, digitalización, preprocesamiento, entrenamiento y reconocimiento. Para cada una de ellas se mencionan las técnicas aplicadas, se justifica la utilización de éstas y se muestran algunos avances preliminares.

4.1 Descripción general del sistema

El sistema de reconocimiento de firma autógrafa tiene como etapa inicial la obtención de la señal eléctrica por medio del transductor mecánico–eléctrico. Los cambios de magnetización del transductor magneto-elástico, producidos al momento de hacer una firma, son detectados por medio de una bobina colectora de dimensiones reducidas. La señal inducida en las terminales de la bobina es relativamente pequeña y requiere ser amplificada, de este modo se reducen los errores de cuantización producidos durante el muestreo.

La segunda etapa consiste en digitalizar la señal por medio del sistema de adquisición de datos basado en la programación de la tarjeta de sonido SoundBlaster 16. En esta etapa se programa en la SB 16 la frecuencia de muestreo, resolución de las muestras y el modo de muestreo a utilizar para la transferencia de los datos.

La tercera etapa del sistema es denominada preprocesamiento. En dicha etapa, las señales muestreadas y almacenadas en archivo son sometidas a un procesamiento digital que permita lograr una mejor extracción de sus características. Este procesamiento filtra el ruido agregado a la señal en las etapas previas, posteriormente obtiene la envolvente de la señal filtrada, la cual sirve para localizar el inicio y fin de las muestras que contienen información concerniente a la firma, después realiza el recorte de ambas señales (filtrada y envolvente) además de normalizarlas en amplitud. Aquella que proviene de la señal filtrada se utiliza para el cálculo de los coeficientes LPC. Finalmente se normaliza en longitud a la envolvente para el cálculo de los coeficientes CAP y EP.
La cuarta etapa se denomina entrenamiento, en ésta se extraen las características o patrones de un conjunto de muestras de la firma de una persona y se

determinan los intervalos de certidumbre para la autentificación de una firma posterior. Inicialmente se realiza la extracción de patrones de las firmas: coeficientes de predicción lineal (LPC), coeficientes de aproximación polinomial (CAP) y la envolvente promediadora (EP). La extracción de patrones se realiza analizando a las firmas por segmentos pequeños. Para el caso LPC y CAP, los patrones obtenidos se agrupan en clases y se calculan sus centroides. Estos centroides son utilizados como referencia para validar una firma posterior. Para determinar los intervalos de certidumbre se calculan las distancias entre los centroides obtenidos de las firmas en conjunto y los de cada firma. De este modo, se toma la distancia promedio y su desviación estándar con objeto de determinar el intervalo de certidumbre necesario para validar o rechazar una firma bajo prueba. Este criterio se aplica en cada uno de los métodos utilizados para identificar una señal: LPC, CAP y envolvente promediadora.

En la quinta y última etapa del sistema se llamada reconocimiento. En ésta se valida o rechaza a una firma nueva. Para lograr dicho objetivo, la etapa de reconocimiento incluye los procesos de generación de la señal eléctrica, digitalización y preprocesamiento. También utiliza el proceso de entrenamiento, aplicado únicamente a la firma que requiere ser autentificada. En el reconocimiento se obtienenen la envolvente y centroides de una firma y se comparan contra los obtenidos de las firmas en conjunto durante la etapa de entrenamiento. Dependiendo del valor obtenido por dicha comparación y tomando como referencia el intervalo de certidumbre establecido en la etapa de entrenamiento, se valida o rechaza la firma bajo prueba.

4.2 Obtención y amplificación de las señales producidas por el transductor magneto–elástico

Basándose en las propiedades magnetostrictivas del alambre magneto-elástico [7], se puede obtener una señal eléctrica proporcional a la rúbrica de una persona. Para obtener dicha señal se utiliza el dispositivo denominado “pluma magneto–elástica” (Fig. 4.1). En este dispositivo, las vibraciones producidas por fricción al plasmar una firma sobre una superficie son transferidas al alambre por medio del sistema masa-resorte. De esta manera se producen los cambios de tensión–compresión que originan las variaciones de magnetización del alambre magneto–elástico, presentándose así un voltaje inducido en las terminales de la bobina colectora.

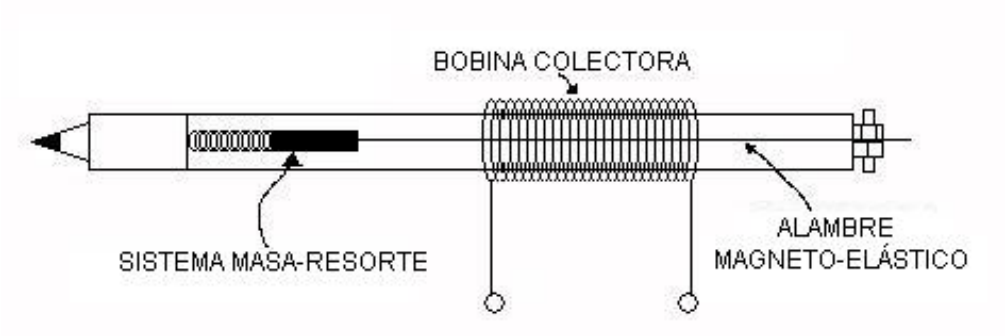

Fig. 4.1. Pluma magneto–elástica.

Las señales eléctricas que se obtienen son relativamente pequeñas, del orden de 40mV, por tal motivo es necesario amplificarlas para obtenerlas con un nivel de voltaje que permita procesarlas de manera eficiente. En la Fig. 4.2 se muestra el diagrama del circuito utilizado para amplificar las señales. Éste consta de un amplificador inversor con ganancia variable y un seguidor de voltaje como acoplamiento entre la fuente de señal y el amplificador. En la Fig. 4.3 se muestra un ejemplo de la señal eléctrica proveniente de una firma autógrafa obtenida por medio de la pluma magneto-elástica, después de haber sido amplificada y digitalizada.

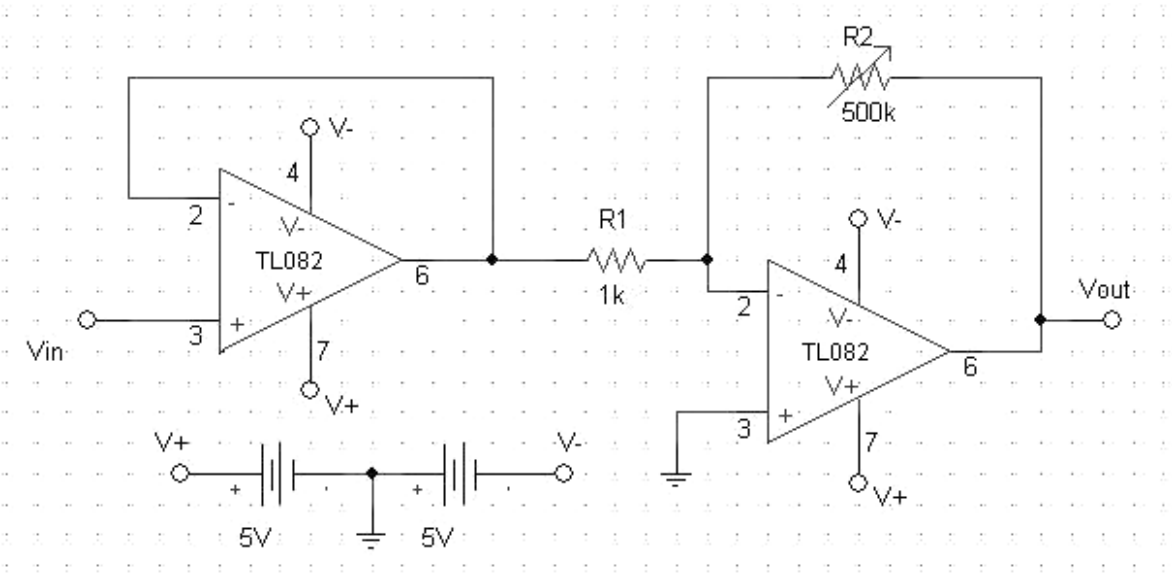

Fig. 4.2. Circuito amplificador de señal.

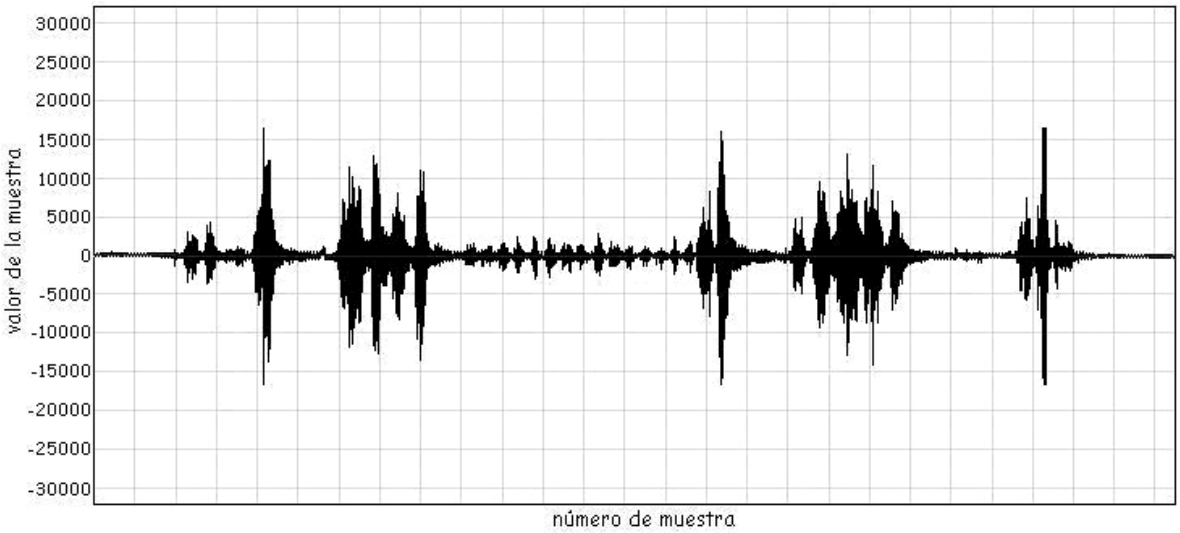

Fig. 4.3. Señal de una firma autógrafa, obtenida con la pluma magneto-elástica después de la etapa de amplificación y digitalización.

Como se puede pensar intuitivamente, si la señal generada por la pluma magneto–elástica contiene información relativa al movimiento natural de la mano, ésta deberá ser de baja frecuencia. En la Fig. 4.4 se muestra la imagen completa y seccionada de una firma autógrafa.

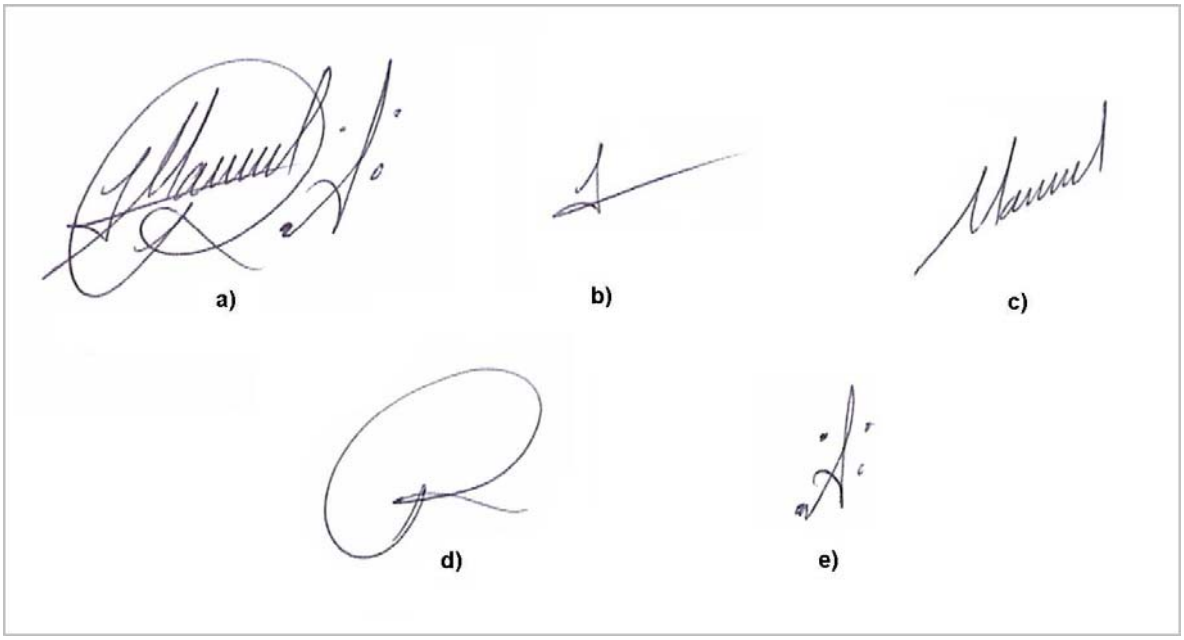

Fig. 4.4. Imagen de una firma autógrafa: a) completa; b),…,e) trazos.

En la Fig. 4.5 se muestra el perfil de la señal eléctrica generada por la firma anterior, éste se encuentra dividido en 4 zonas diferentes que representan a los

trazos que conforman a la firma. La señal de baja frecuencia generada por el movimiento de la mano se ve reflejada en la envolvente de la señal (Fig. 4.6), la cual se obtiene rectificando a ésta y filtrándola con un filtro digital pasa bajas. La señal mostrada en la Fig. 4.5 es el resultado de modular en amplitud a la señal de alta frecuencia, producida esencialmente por la frecuencia natural del alambre asociada a la del sistema masa-resorte, con la señal de baja frecuencia proveniente del movimiento natural de la mano representada en la envolvente.

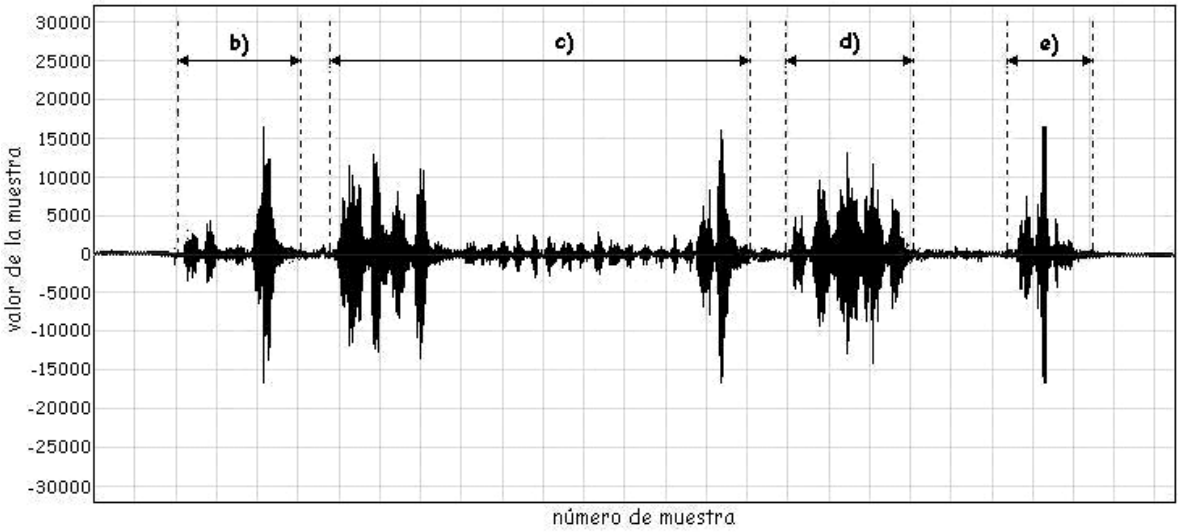

Fig. 4.5. Oscilograma que muestra las secciones de la señal eléctrica generada por la firma de la Fig. 4.4.

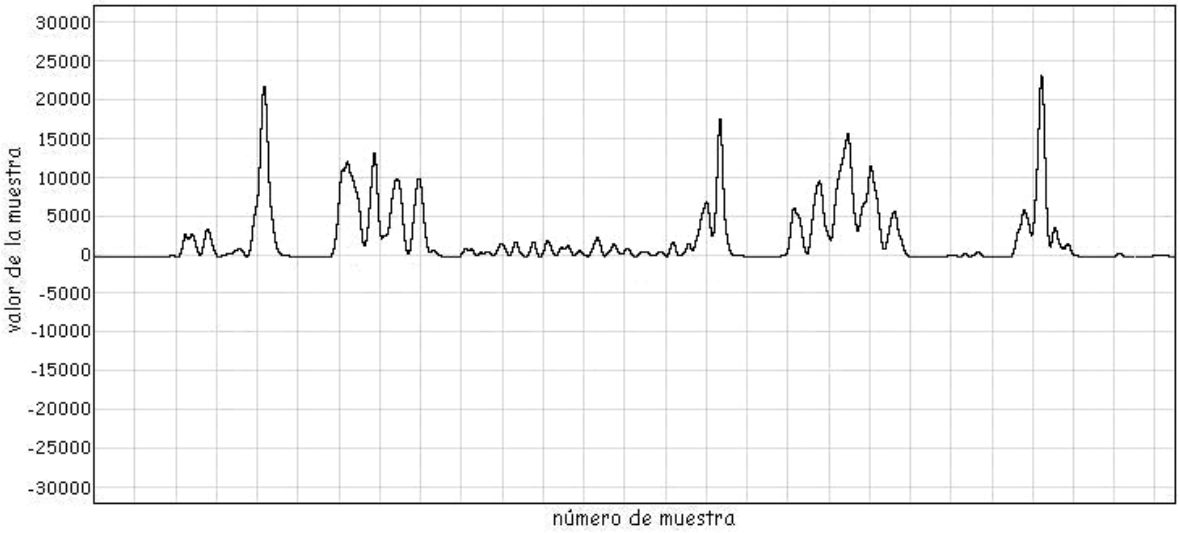

Fig. 4.6. Envolvente de la señal de la Fig. 4.5 que representa a los movimientos naturales de la mano.

En la Fig. 4.7a se muestra otro ejemplo de la imagen de una firma autógrafa, en las figuras 4.7b y 4.7c, respectivamente se muestran los oscilogramas de la señal modulada en amplitud y la envolvente.

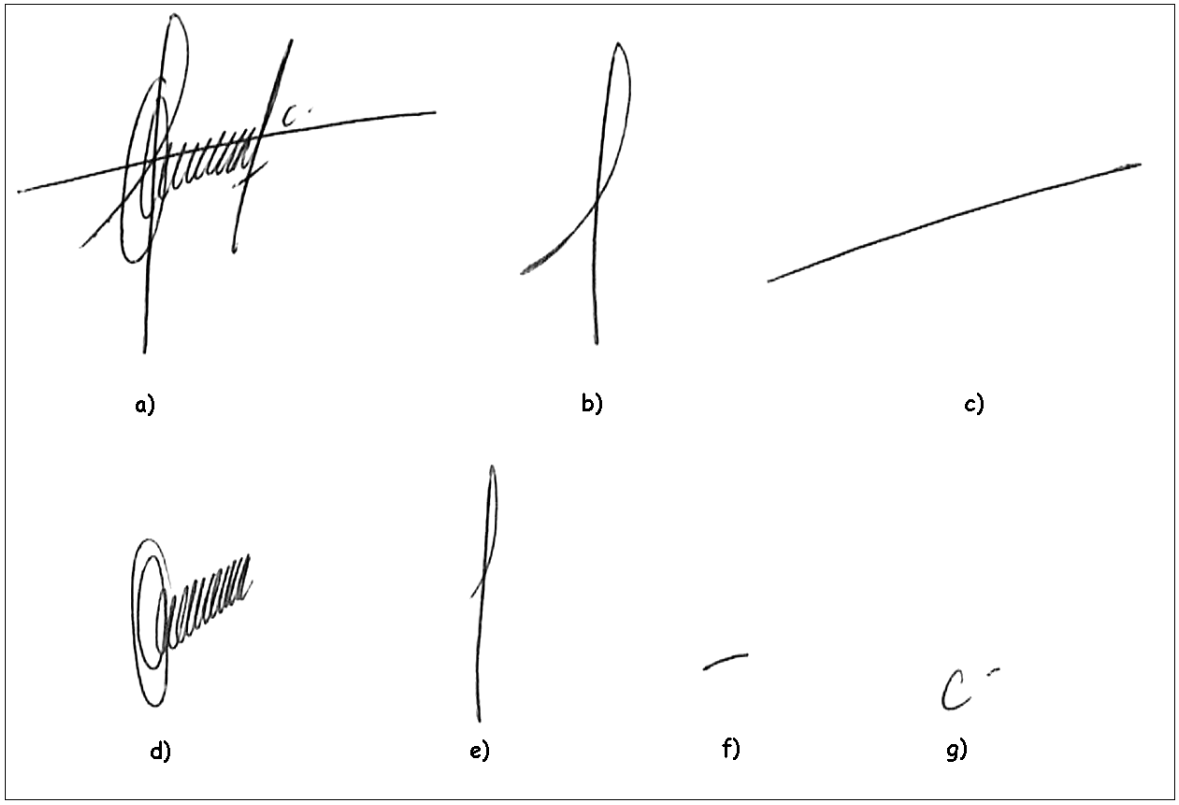

Fig. 4.7a. Imagen de una firma autógrafa: a) completa; b),…,g) trazos.

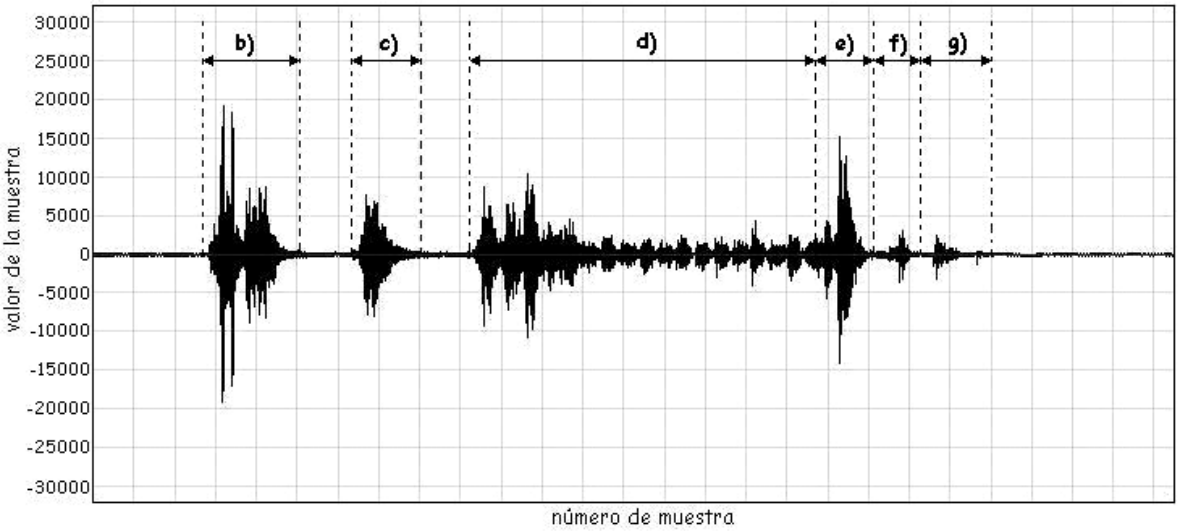

Fig. 4.7b. Oscilograma que muestra las secciones de la señal eléctrica generada por la firma de la Fig. 4.7a.

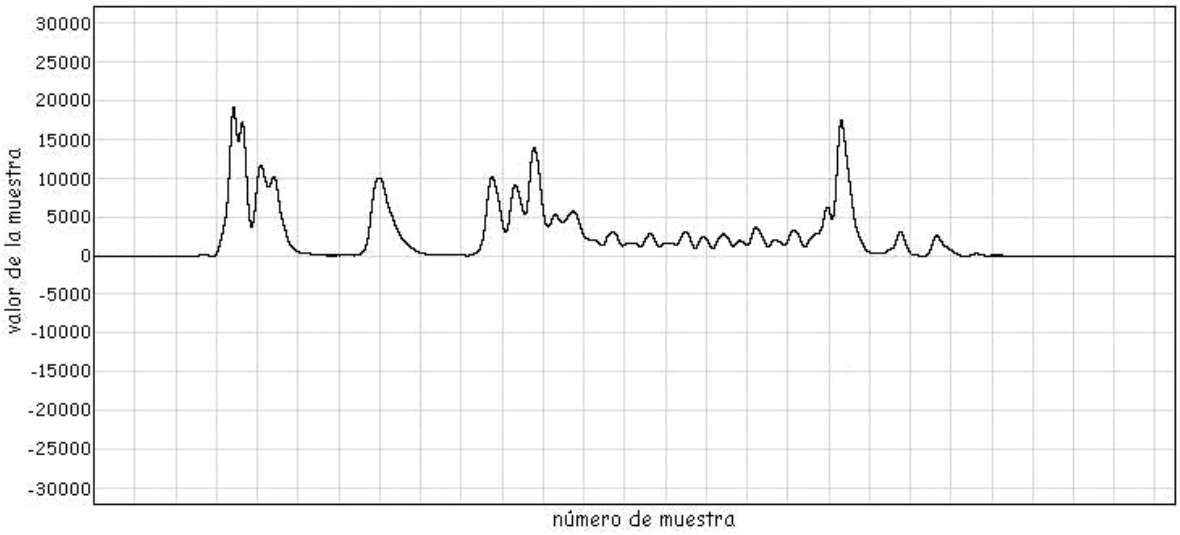

Fig. 4.7c. Envolvente de la señal de la Fig. 4.7b que representa los movimientos naturales de la mano.

Como puede observarse en los dos ejemplos anteriores, cada trazo de una firma autógrafa tiene asociado un segmento de señal eléctrica, cuyo comportamiento representa los movimientos que realiza el firmante al plasmar su rúbrica, además cada trazo tiene un orden específico y secuencial de aparición, el cual puede apreciarse en el oscilograma correspondiente. Este orden no cambia para distintas repeticiones de la firma de una misma persona y puede servir para discernir entre una firma original y una falsa.

Para resaltar que las señales generadas por la pluma magneto-elástica son el resultado de una modulación en amplitud, en la Fig. 4.8 se muestra el espectrograma de las señales que genera. Este gráfico muestra que el mayor contenido en frecuencia de las señales de interés se localiza en un rango de 800 a 1000Hz y la frecuencia cuya componente tiene mayor peso se ubica aproximadamente a 900Hz. También se puede observar que las señales presentan ruido agregado, principalmente con una componente de 60Hz.

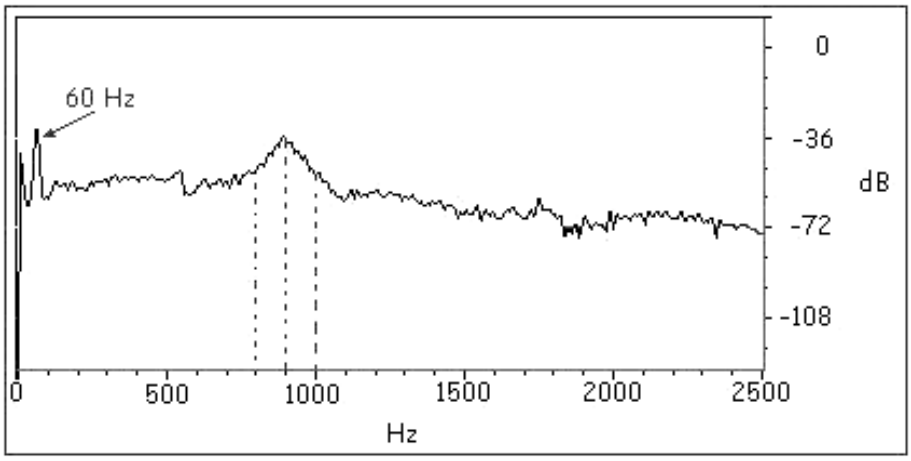

Fig. 4.8. Espectrograma de las señales generadas por la pluma magneto–elástica.

De esta forma se completa la etapa de generación de la señal eléctrica proporcional a la rúbrica de una persona. Es necesario diseñar un sistema de adquisición de datos para muestrear y almacenar las señales, con el objeto de analizarlas y procesarlas digitalmente a través de una computadora personal. El ruido agregado a la señal eléctrica es un factor no deseado y se puede reducir por medio de un filtro analógico aplicado antes del muestreo de la señal. Este paso se omite pues una vez que la señal es digitalizada y almacenada en archivo, el ruido es reducido mediante un filtro digital en la etapa de preprocesamiento. De aquí en adelante se referirá a la señal contenida en archivo mediante el término “firma”, salvo que se especifique otra cosa.

4.3 Sistema de adquisición de datos basado en la programación de la tarjeta de sonido SoundBlaster 16

Para analizar las señales producidas por la pluma magneto–elástica, es necesario disponer de un sistema de adquisición de datos que pueda ser controlado de acuerdo a las necesidades del proyecto. La tarjeta de sonido de una computadora personal ofrece una buena solución. A continuación se mencionan los requisitos para el muestreo de las señales a través del sistema de adquisición de datos.

Inicialmente se verifica que la tarjeta de sonido sea SB 16, para ésto la versión del DSP debe ser 4.xx. Se configura al mezclador para capturar señales por el “Line In”. Posteriormente se especifica la frecuencia de muestreo con una resolución de 16 bits con signo y un solo canal. Se configura el modo de muestreo como auto inicialización DMA para transferir los datos muestreados hacia un bloque de

memoria RAM y posteriormente a disco duro. Una vez concluido el muestreo de la señal, se finaliza el modo de transferencia y se salvan las muestras en archivo.

4.4 Preprocesamiento de las señales bajo estudio

Se le llama preprocesamiento a la etapa en la cual las señales adquiridas a través de la tarjeta de sonido y almacenadas en archivo, son procesadas antes de la extracción de sus parámetros. El preprocesamiento consiste en dar un tratamiento digital a las firmas y adecuarlas para la obtención de sus características más representativas. En la Fig. 4.9, se muestra el diagrama de bloques de esta etapa.

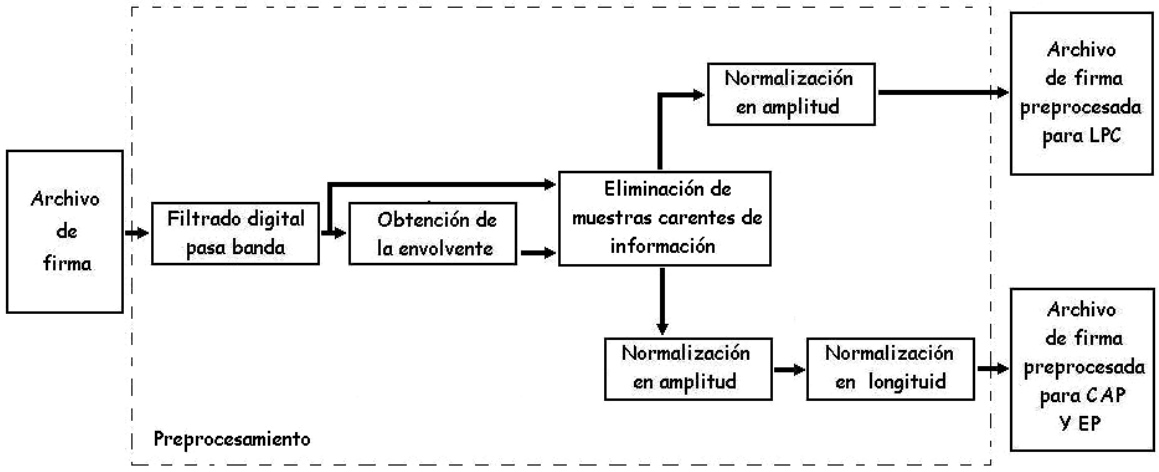

Fig. 4.9. Diagrama de bloques de la etapa de preprocesamiento.

El preprocesamiento tiene como entrada una firma almacenada en archivo y produce dos archivos de salida, uno para el cálculo de los LPC y otro para el de los CAP y EP. El módulo de filtrado digital pasa banda sirve para eliminar ruido de alta y baja frecuencia presente en las firmas. Se implementó un filtro digital FIR pasa banda de 800 a 1000Hz, con ganancia unitaria, de 501 coeficientes y ventana de Blackman. Los resultados de filtrar a las señales pueden verse en la Fig. 4.10, donde se muestran los oscilogramas y espectrogramas de una señal antes y después de haber sido filtrada. Antes de filtrar, se presenta una componente de 60Hz junto con otras de diferentes frecuencias, y después de filtrar, se observa que éstas han disminuido considerablemente. Además, se puede ver que los oscilogramas presentan un comportamiento muy similar debido a que la mayor parte de la banda de información se ha conservado.

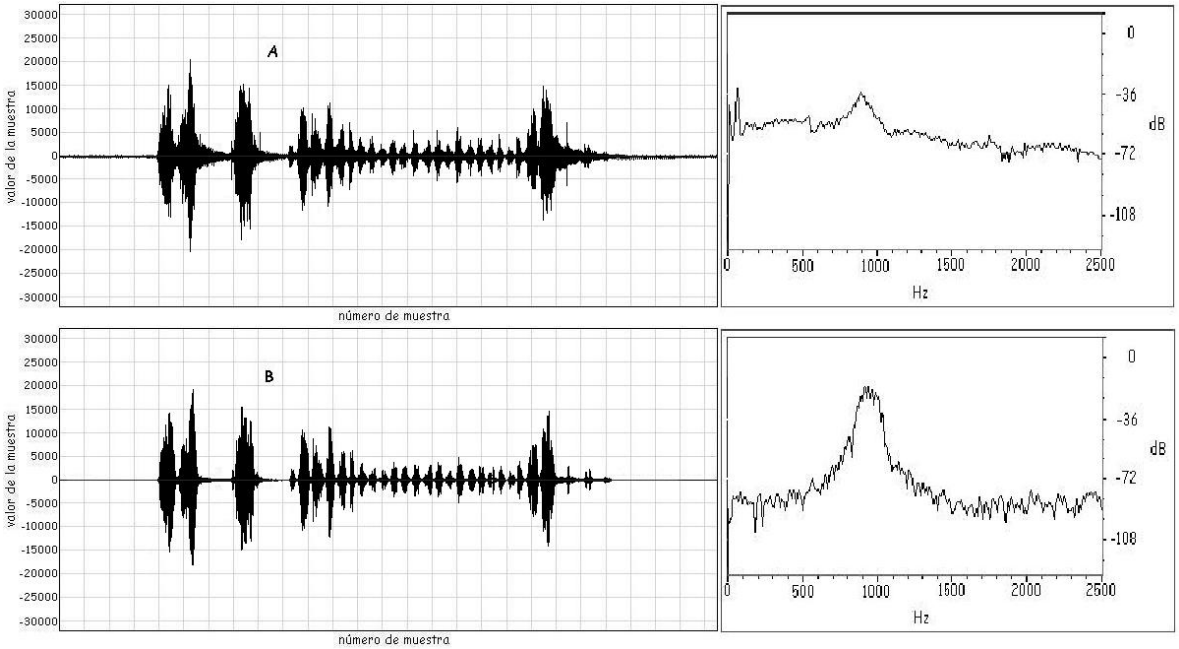

Fig. 4.10. Oscilograma y espectro de una firma: A) antes de ser filtrada
B) después de haber sido filtrada.

Otra justificación para filtrar la señal es eliminar la componente de directa presente en la firma, lo que permite aplicar con mayor precisión la técnica de magnitud promedio en tiempo corto para identificar el inicio y fin de las muestras que contienen información relativa a la firma.

En el módulo de obtención de la envolvente, la señal es rectificada de modo que las muestras presenten únicamente valores positivos (Fig.4.11) y posteriormente es procesada por un filtro pasa bajas para así obtener su envolvente. En la Fig.4.12 se observan los resultados de filtrar una señal rectificada con un filtro digital pasa bajas de 501 coeficientes y ganancia unitaria para una frecuencia de corte de 100Hz y una de 10Hz.

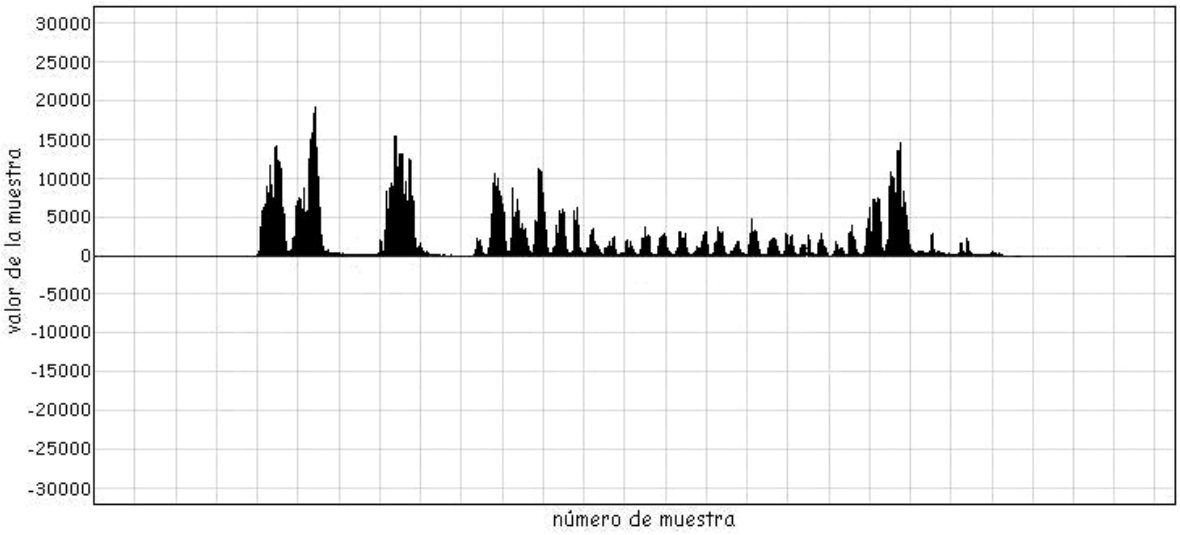

Fig. 4.11. Señal rectificada.

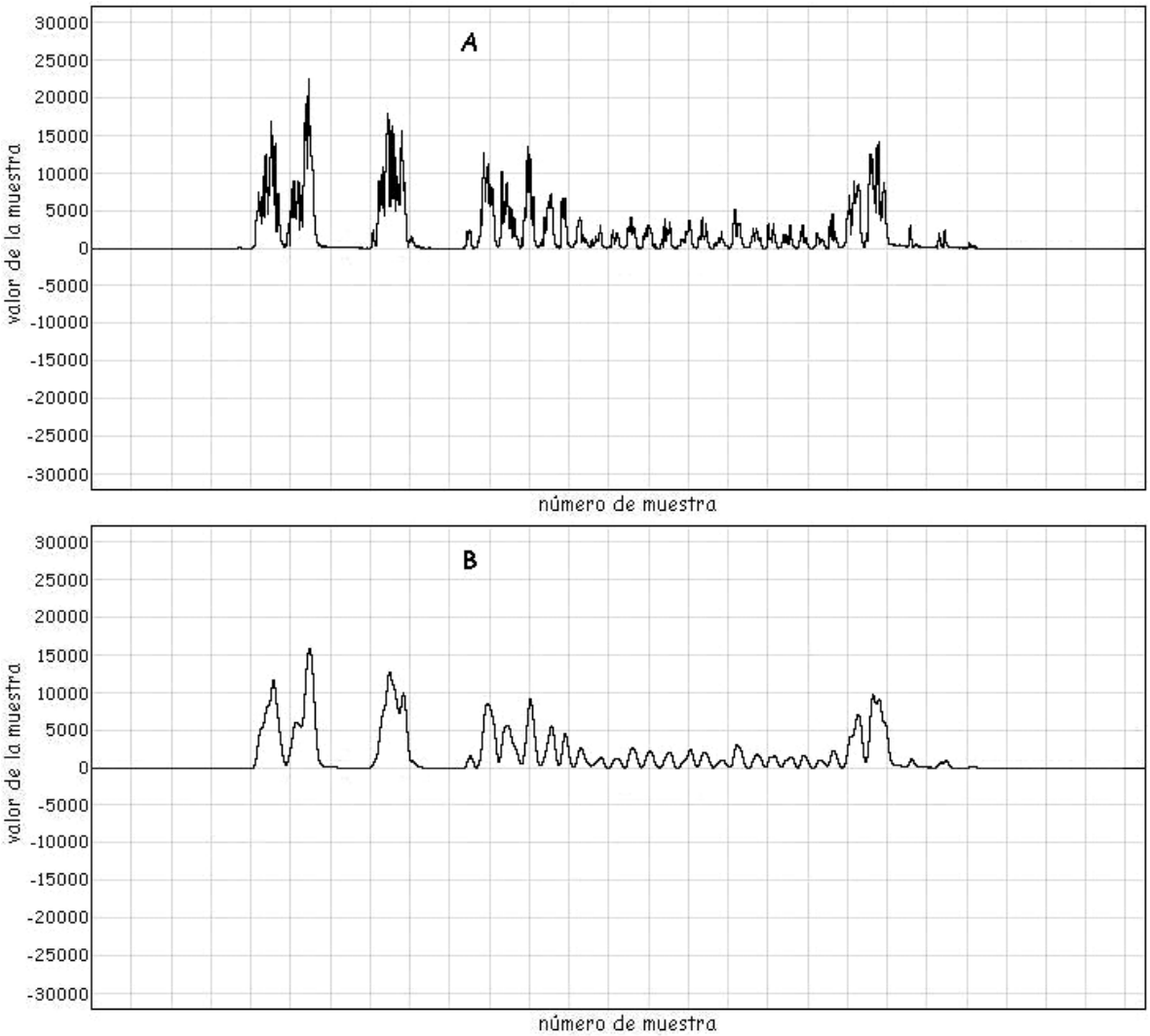

Fig. 4.12. Señal filtrada: A) a 100Hz, B) a 10Hz.

Al inicio y al final de las señales muestreadas mediante el sistema de adquisición de datos, se presentan muestras que no contienen información concerniente a las firmas (Fig. 4.13). Para lograr que los parámetros extraídos representen esencialmente a las firmas, es necesario eliminar las muestras de los extremos que no tienen información. Ésta es la función del bloque de eliminación de muestras carentes de información. Este bloque calcula la magnitud promedio total de la envolvente y utiliza un porcentaje de su valor como umbral de decisión, determina en qué muestras se supera dicho umbral, con lo que se determina el inicio y fin de la firma. Por último recorta la señal filtrada con pasa banda y la envolvente. En el presente trabajo se tomó como umbral un valor igual al 10% de la magnitud promedio total. El resultado de aplicar esta técnica sobre una firma se muestra en la Fig. 4.14.

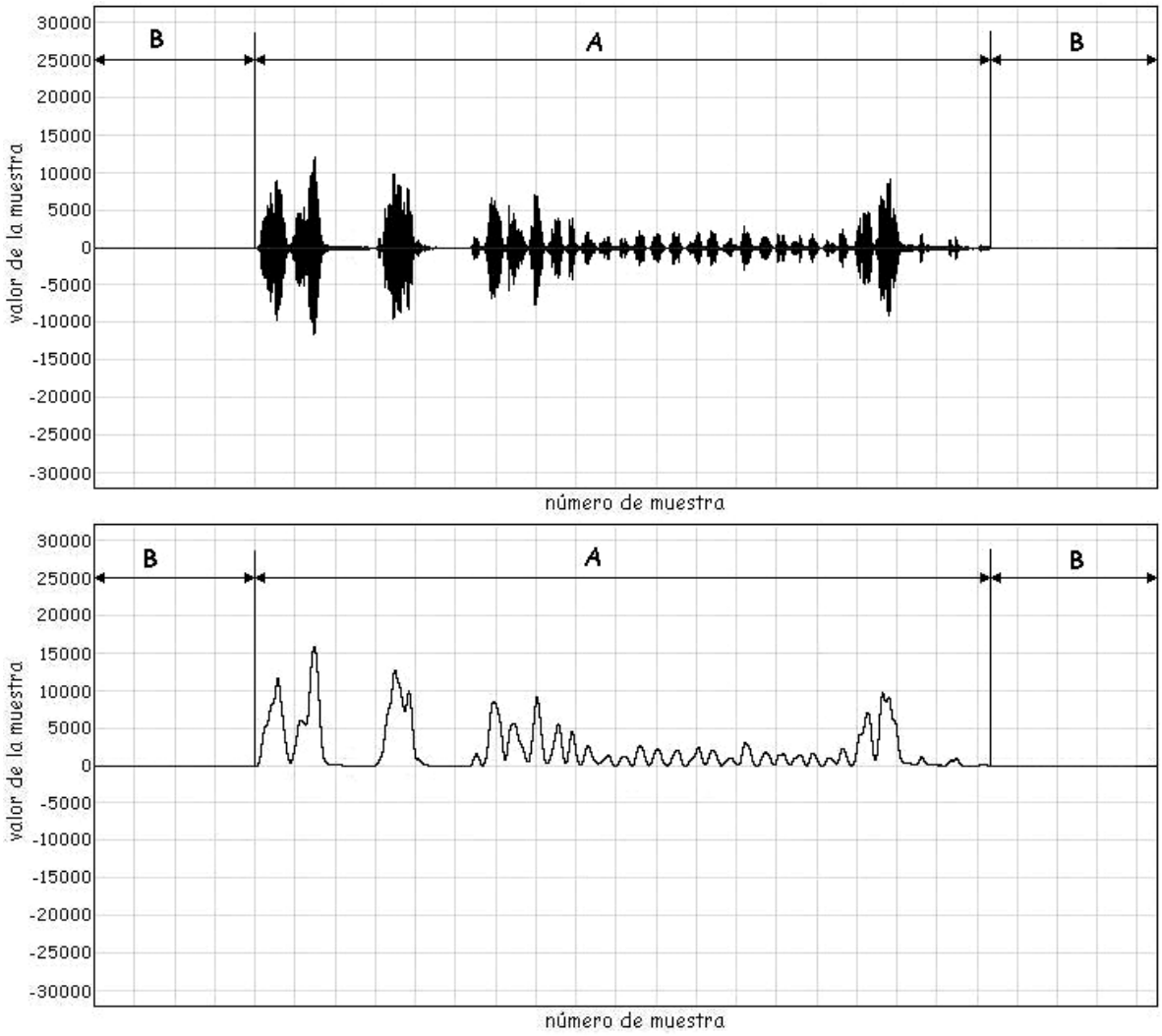

Fig. 4.13. A) muestras que contienen información de la firma,
B) muestras carentes de información.

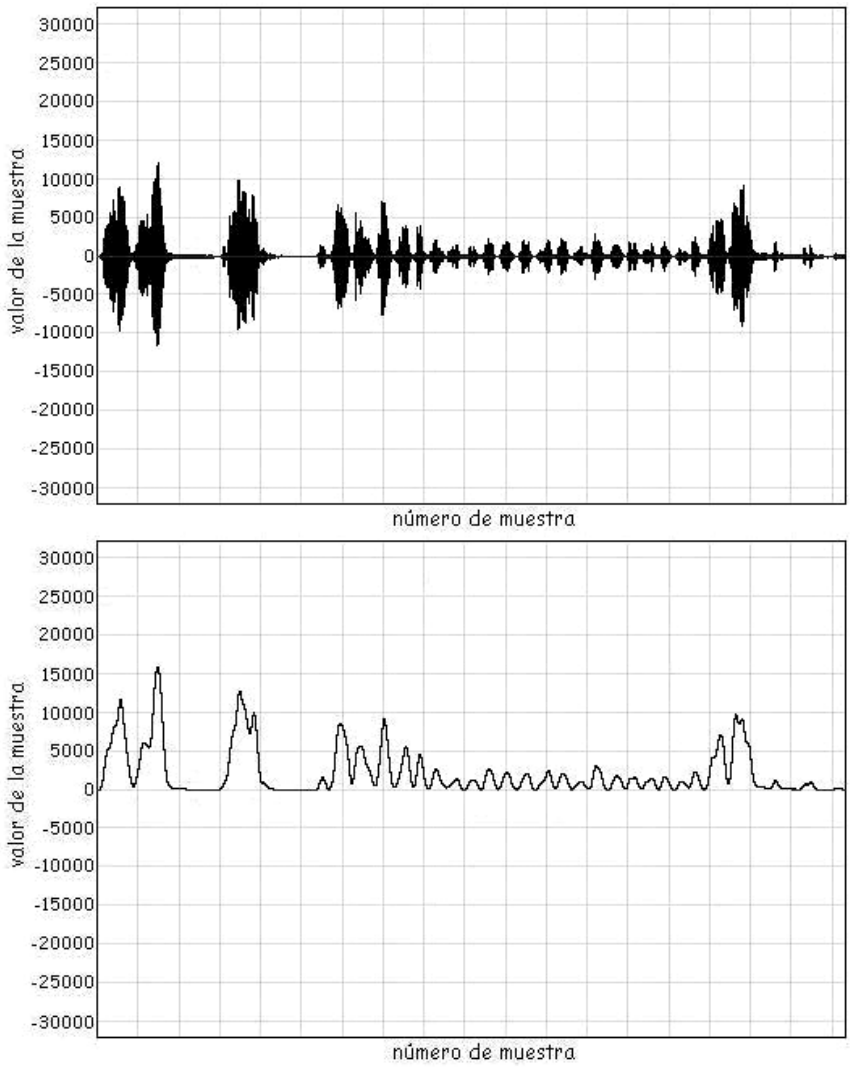

Fig. 4.14. Señales que esencialmente contienen muestras con información concerniente a la firma.

El módulo de normalización en amplitud escala a la señal de modo que su máximo valor sea representado con el mayor valor permitido por el sistema (Fig. 4.15). El objetivo de este módulo es hacer que las señales presenten un comportamiento similar en lo que se refiere a la magnitud, ya que ésta depende de la “fuerza” con la que el firmante realiza su rúbrica y es de carácter variable.

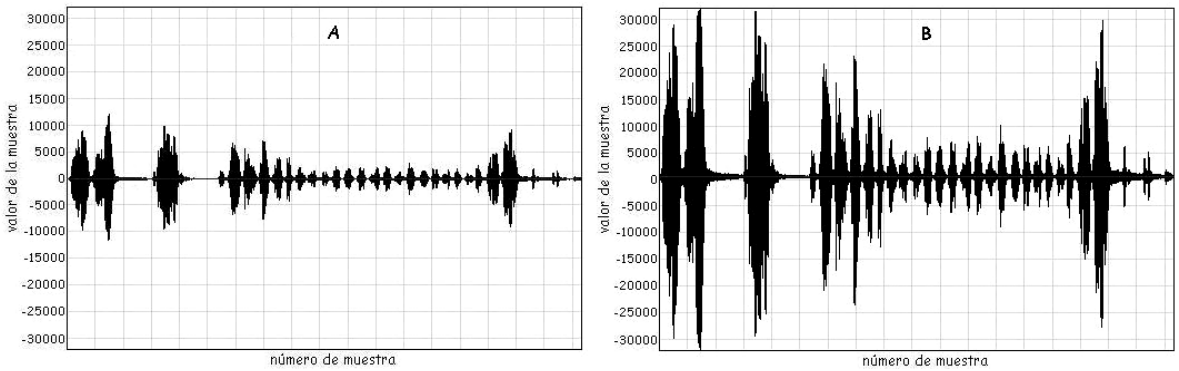

Fig. 4.15. A) oscilograma original, B) oscilograma normalizado en amplitud.

El módulo de normalización en longitud realiza una expansión de las envolventes para que todas tengan el mismo número de muestras totales (Fig. 4.16).

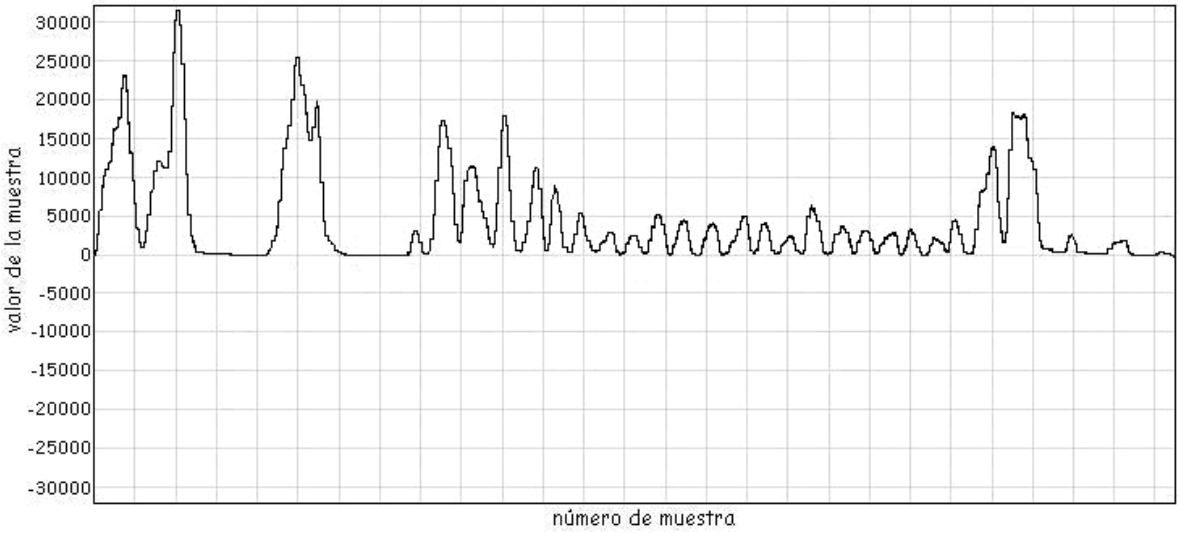

Fig. 4.16. Normalización en longitud.

De esta forma concluye la etapa de preprocesamiento. Como se puede observar, el tipo de preprocesamiento al cual son sometidas las firmas depende de los parámetros que se les desee extraer. Al final de esta etapa se tienen dos archivos: uno con la firma filtrada por un filtro pasa banda, recortada y normalizada en amplitud, que será utilizado para el cálculo de los LPC y otro con la firma filtrada por un filtro pasa banda, rectificada, filtrada por un filtro pasa bajas, recortada y normalizada tanto en amplitud como en longitud, que será utilizado para el cálculo de coeficientes CAP y EP. Ambos tipos de archivos son utilizados posteriormente en la etapa de entrenamiento.

4.5 Entrenamiento del sistema

El entrenamiento del sistema consiste en realizar la extracción de características o patrones de un conjunto de firmas preprocesadas. En este trabajo se emplean tres métodos diferentes para la obtención de dichos patrones: coeficientes de predicción lineal, coeficientes de aproximación polinomial y la envolvente promediadora. En la Fig. 4.17 se muestra el diagrama de bloques de la etapa de extracción de patrones.

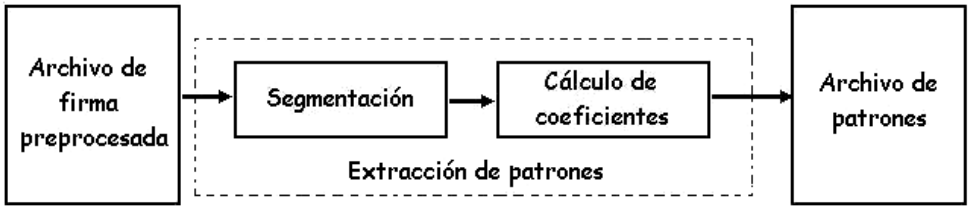

Fig. 4.17. Diagrama de bloques de la extracción de patrones.

Para calcular los coeficientes LPC de una firma preprocesada se establece un número fijo de patrones a determinar por firma, por lo que el tamaño de la segmentación es variable y depende únicamente del número de muestras de la firma bajo estudio. El número de coeficientes LPC calculados por segmento determina la dimensión de los patrones LPC y se especifica dependiendo de la calidad deseada en la predicción.

Los archivos de las firmas preprocesadas para el análisis LPC son utilizados para el cálculo de los coeficientes de predicción lineal. A través estos coeficientes se busca representar los cambios de frecuencia que presentan las señales, razón por la cual éstas no son expandidas en longitud en la etapa de preprocesamiento, ya que esto implicaría un cambio significativo de sus características en frecuencia, como se ilustra en la Fig. 4.18.

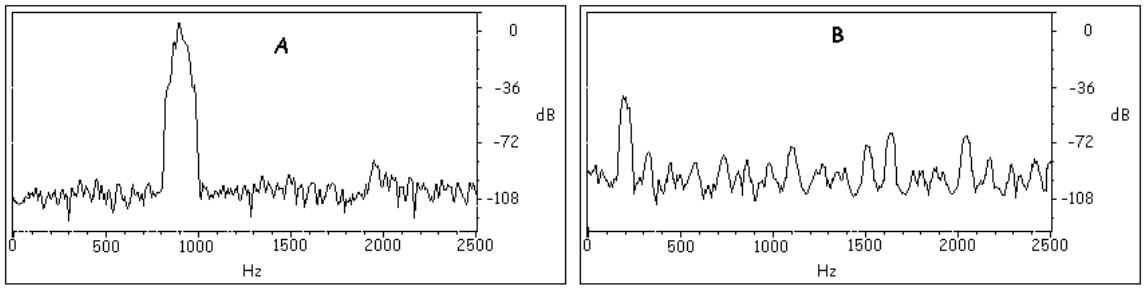

Fig. 4.18. Espectro de una firma: A) sin expandir, B) expandida.

A través de los coeficientes CAP, se busca representar por segmentos a la envolvente de la firma mediante un conjunto de polinomios que se aproximen lo mejor posible a la forma de la curva que describe. Como en este caso se trata de representar a la señal atendiendo el cambio de magnitud, es factible realizar la normalización en longitud. Como puede verse en la Fig. 4.19, la envolvente presenta diferentes comportamientos a lo largo de la firma. Al tomar segmentos muy pequeños de la señal, los coeficientes CAP aproximan a una línea recta, por lo que éstos únicamente presentan valores para la ordenada al origen y la pendiente de la recta. Para lograr que la mayoría de los coeficientes CAP tomen valores distintos de cero, es necesario que éstos aproximen a un segmento de la señal que no se comporte como una recta. En la Fig. 4.19 se puede observar que al tomar segmentos de un tamaño suficientemente grande, se presentan curvas cuyos comportamientos son muy diferentes entre sí.

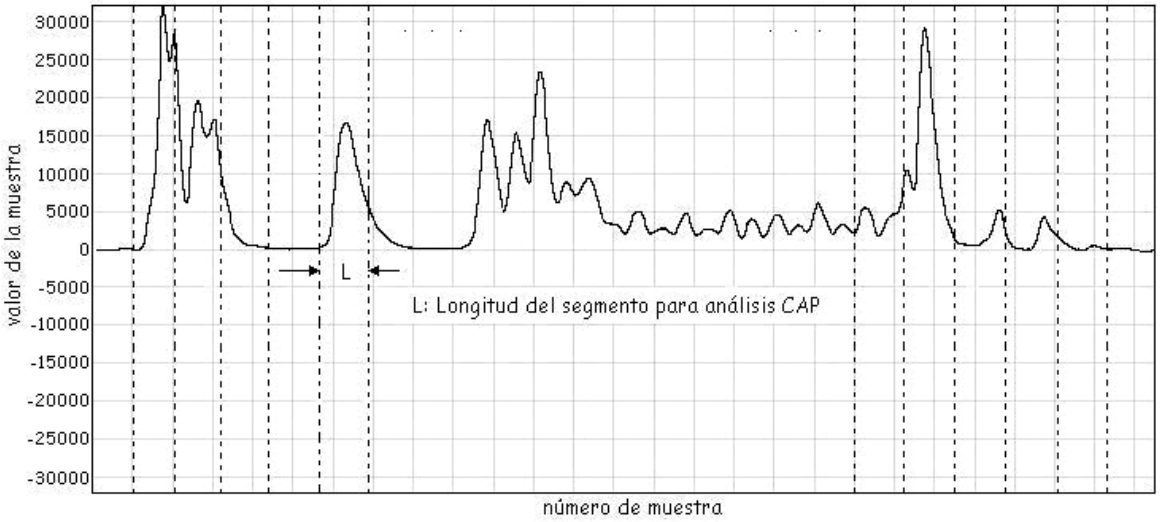

Fig. 4.19. Segmentación de la envolvente para calcular los coeficientes CAP.

El método de la envolvente promediadora consiste en obtener un promedio del contorno de todas las firmas. Para esto, las muestras de posiciones correspondientes entre sí, de los archivos preprocesados para CAP y EP, son sumadas y divididas entre el número total de firmas. Posteriormente, la envolvente resultante es almacenada en disco duro y es utilizada como referencia en la etapa de reconocimiento. De esta manera se obtiene un método muy sencillo para la autentificación de firma autógrafa, basado en la simple comparación entre la envolvente de una firma nueva y la envolvente promedio.

Una vez que se han obtenido los archivos de patrones de cada una de las firmas, éstos son concatenados en un solo archivo para agruparlos mediante el algoritmo de **C** medias, con lo cual se obtiene un conjunto reducido de centroides que representa al total de los patrones (Fig. 4.20). Este procedimiento se aplica por separado a los patrones CAP y a los LPC, pero esto no quiere decir que el algoritmo sea diferente para cada caso. El número de clases en que son agrupados los patrones depende de la naturaleza de los mismos.

Fig. 4.20. Diagrama de bloques del agrupamiento.

De esta manera se hace la extracción y agrupamiento de los patrones de las firmas. Cada método requiere definir un intervalo de certidumbre para aceptar o rechazar una firma bajo prueba. La medida utilizada para comparar los centroides LPC y CAP de dos firmas es la distancia euclidiana; mientras que la utilizada para comparar por el método de la envolvente promediadora es la distancia de error cuadrático medio. Se determina la distancia que existe entre los centroides del conjunto de firmas y los centroides de cada firma individual. La certidumbre en la distancia, para cada método, está en función de la distancia promedio y la desviación estándar.

Al concluir la etapa de entrenamiento se genera un archivo de centroides LPC, uno de centroides CAP y otro con la envolvente promedio, que contienen las características más representativas de las firmas utilizadas en esta etapa. Además, se establecen también los intervalos de certidumbre requeridos para validar una firma nueva en la etapa de reconocimiento.

4.6 Etapa de reconocimiento

En la Fig. 4.21 se muestra el diagrama de bloques de la etapa de reconocimiento. Se puede observar que la señal bajo prueba se encuentra almacenada en archivo, lo que implica que la señal ha sido muestreada con anterioridad. Posteriormente se efectúa el preprocesamiento de la señal y se extraen sus patrones LPC y CAP,

agrupando éstos para obtener sus centroides. Finalmente, se comparan estos resultados contra los obtenidos en la etapa de entrenamiento, de acuerdo a lo establecido por los métodos propuestos. Dependiendo del valor de la distancia resultante y del intervalo de certidumbre establecido en el entrenamiento, se valida o rechaza la firma bajo prueba.

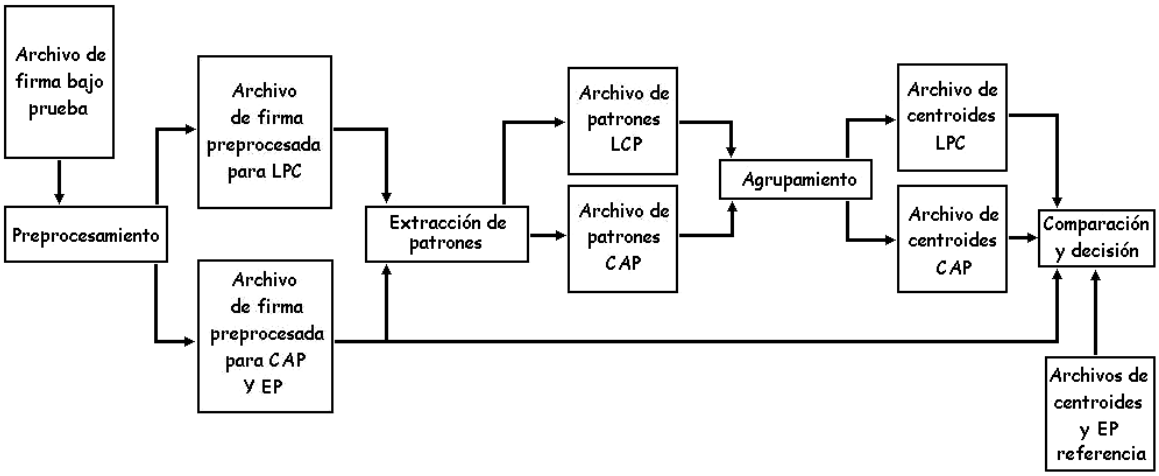

Fig. 4.21. Diagrama de bloques de la etapa de reconocimiento.

El primer método de validación toma como referencia a los centroides LPC obtenidos de las firmas en conjunto en la etapa de entrenamiento y los compara contra los centroides LPC de la señal bajo prueba mediante el cálculo de la distancia euclidiana. Para que el 95% de las firmas autenticas sean validadas, es condición que la distancia obtenida sea menor al promedio más dos veces la desviación estándar y mayor al promedio menos dos veces la desviación estándar [14].

El segundo método toma como referencia a los centroides CAP de las firmas en conjunto y los compara contra los centroides CAP de la señal bajo prueba mediante el cálculo de la distancia euclidiana. De manera semejante al método anterior, el 95% de las firmas autenticas son validadas si la distancia obtenida es menor al promedio más dos veces la desviación estándar y mayor al promedio menos dos veces la desviación estándar.

EL tercer método toma como referencia a la envolvente promedio de las firmas y la compara contra la envolvente de la señal bajo prueba. Para esto, calcula la distancia de error cuadrático medio. Las firmas autenticas serán validadas el 95% de las veces si la distancia obtenida es menor al promedio más dos veces la

desviación estándar y mayor del promedio menos dos veces la desviación estándar.

De esta forma se tienen tres métodos para validar una firma bajo prueba, de los cuales dos han sido propuestos de manera novedosa en este trabajo (CAP y EP). En el capítulo siguiente se muestran los resultados obtenidos al someter a prueba al sistema de reconocimiento de firma autógrafa.

CAPÍTULO 5
RESULTADOS Y CONCLUSIONES

En este capítulo, se muestran los resultados obtenidos al someter a prueba el sistema de reconocimiento de firma autógrafa. El sistema fue entrenando para autentificar la firma de dos personas por separado, para esto se utilizaron los tres métodos propuestos ya mencionados. Finalmente se muestran las conclusiones obtenidas al realizar el presente trabajo de tesis.

5.1 Resultados

Envolvente promediadora: Inicialmente se utilizó el método de la envolvente promediadora para autentificar la firma de dos personas por separado, se utilizaron 40 firmas para realizar el entrenamiento de cada una de ellas.

A partir de las distancias obtenidas, al comparar la envolvente promedio con cada una de las 40 envolventes que la generaron, por cada firmante se obtuvo la distancia promedio y su desviación estándar con objeto de determinar el intervalo de certidumbre para la autentificación de una firma nueva. En la tabla 5.1 se muestran los estadísticos obtenidos.

Persona	Distancia promedio	Desviación estándar
A	4376.536021	749.766490
B	4567.170178	695.325658

Tabla 5.1. Estadísticos EP, de las distancias obtenidas al comparar el prototipo total con el de cada una de las firmas que lo generaron.

Para que el 95% del conjunto de firmas auténticas sean validadas, se toma como certidumbre, un intervalo entre la distancia media menos dos veces la desviación estándar y la distancia media más dos veces la desviación estándar. En la tabla 5.2, se muestran los intervalos de certidumbre obtenidos en la etapa de entrenamiento de ambas personas.

Persona	Intervalo de certudumbre
A	2877.003041 < D < 5876.069001
B	3176.518862 < D < 5957.821494

Tabla5.2. Intervalos de certidumbre utilizados para autentificar una firma por el método de la envolvente promediadora.

Al someter a prueba un conjunto de 10 firmas auténticas se obtienen los resultados mostrados en la tabla 5.3.

Firma	Persana A		Persona B	
	Distancia	Validación	Distancia	Validación
1	5294.529695	Sí	4582.931846	Sí
2	4158.606832	Sí	4056.143376	Sí
3	3824.438953	Sí	3367.821857	Sí
4	4968.192827	Sí	4572.996382	Sí
5	5257.775420	Sí	5314.786973	Sí
6	4850.274577	Sí	4470.568001	Sí
7	4564.674650	Sí	4241.244707	Sí
8	3722.122135	Sí	4133.248537	Sí
9	4554.998252	Sí	4177.627699	Sí
10	3426.843438	Sí	3815.482167	Sí

Tabla 5.3. Resultados obtenidos al validar 10 firmas auténticas por el método de la envolvente promediadora.

En la tabla 5.4 se muestran los resultados obtenidos al someter a prueba un conjunto de 10 firmas falsas.

Firma	Persana A		Persona B	
	Distancia	Validación	Distancia	Validación
1	8286.936734	No	7305.607800	No
2	7518.729611	No	7671.288226	No
3	6512.194977	No	6839.901301	No
4	6479.177217	No	6710.082223	No
5	6454.500084	No	6403.420826	No
6	6866.452619	No	6566.881679	No
7	7320.920968	No	6687.593401	No
8	7646.362132	No	6601.806812	No
9	6914.928638	No	6456.758655	No
10	6780.865609	No	6710.082223	No

Tabla 5.4. Resultados obtenidos al autentificar 10 firmas falsas por el método de la envolvente promediadora.

Coeficientes CAP: Después se utilizó el método de identificación basado en los coeficientes CAP. Para el entrenamiento se utilizaron 40 firmas por persona y de cada una se calcularon 52 polinomios de grado 7.

A partir de las distancias obtenidas, al comparar los prototipos totales CAP con los de cada una de las 40 firmas que los generaron, por cada firmante se obtuvo la distancia promedio y su desviación estándar con objeto de determinar el intervalo de certidumbre para la autentificación de una firma nueva. En la tabla 5.5 se muestran los estadísticos obtenidos.

Persona	Distancia promedio	Desviación estándar
A	2686.640770	430.968272
B	2982.452440	505.711652

Tabla 5.5. Estadísticos CAP, de las distancias obtenidas al comparar el prototipo total con el de cada una de las firmas que lo generaron.

Para que el 95% del conjunto de firmas auténticas sean validadas, se toma como certidumbre, un intervalo entre la distancia media menos dos veces la desviación estándar y la distancia media más dos veces la desviación estándar. En la tabla 5.6, se muestran los intervalos de certidumbre obtenidos en la etapa de entrenamiento de ambas personas.

Persona	Intervalo de certudumbre
A	1824.704226 < D < 3548.577314
B	1971.029136 < D < 3993.875744

Tabla5.6. Intervalos de certidumbre utilizados para autentificar una firma por el método CAP.

Al someter a prueba un conjunto de 10 firmas auténticas se obtienen los resultados mostrados en la tabla 5.7.

Firma	Persana A		Persona B	
	Distancia	Validación	Distancia	Validación
1	2862.418659	Sí	2311.605579	Sí
2	2762.304661	Sí	3325.691430	Sí
3	3210.260815	Sí	2603.786098	Sí
4	2810.271720	Sí	2921.229964	Sí
5	2630.502473	Sí	3597.910401	Sí

6	2331.497122	Sí	2729.143309	Sí
7	3098.440239	Sí	2422.829686	Sí
8	3240.996657	Sí	3049.673039	Sí
9	2482.092717	Sí	2797.091154	Sí
10	3461.284805	Sí	2612.609388	Sí

Tabla 5.7. Resultados obtenidos al validar 10 firmas auténticas por el método de coeficientes de aproximación polinomial.

En la tabla 5.8 se muestran los resultados obtenidos al someter a prueba un conjunto de 10 firmas falsas.

Firma	**Persana A**		**Persona B**	
	Distancia	**Validación**	**Distancia**	**Validación**
1	5100.667771	No	4412.416213	No
2	3804.444516	No	5075.609988	No
3	4087.091040	No	4543.104574	No
4	4539.381372	No	4221.239922	No
5	4618.571116	No	4737.064840	No
6	4340.189428	No	4415.557213	No
7	4771.205820	No	4141.357930	No
8	5021.068952	No	4288.198627	No
9	4733.899466	No	4544.635588	No
10	3937.086972	No	4221.239922	No

Tabla 5.8. Resultados obtenidos al autentificar 10 firmas falsas por el método de coeficientes de aproximación polinomial.

Coeficientes LPC: El último método propuesto para autentificar una firma autógrafa, está basado en el cálculo de los coeficientes de predicción lineal. Para el entrenamiento se utilizaron 40 firmas por persona. Se obtuvieron 280 patrones LPC de dimensión 10 para cada firma.

A partir de las distancias obtenidas, al comparar los prototipos totales LPC con los de cada una de las 40 firmas que lo generaron, por cada firmante se obtuvo la distancia promedio y desviación estándar con objeto de determinar el intervalo de certidumbre para autentificar una firma nueva. En la tabla 5.9 se muestran los estadísticos obtenidos.

Persona	Distancia promedio	Desviación estándar
A	1.238950	0.184761
B	1.343459	0.265174

Tabla 5.9. Estadísticos LPC, de las distancias obtenidas al comparar el prototipo total con el de cada una de las firmas que lo generaron.

Para que el 95% del conjunto de firmas auténticas sean validadas, se toma como certidumbre un intervalo entre la distancia media menos dos veces la desviación estándar y la distancia media más dos veces la desviación estándar. En la tabla 5.10, se muestran los intervalos de certidumbre obtenidos en la etapa de entrenamiento de ambas personas.

Persona	Intervalo de certudumbre
A	0.869428 < D < 1.608472
B	0.813111 < D < 1.873807

Tabla5.10. Intervalos de certidumbre utilizados para autentificar una firma por el método LPC.

Al someter a prueba un conjunto de 10 firmas auténticas se obtienen los resultados mostrados en la tabla 5.11.

Firma	Persana A		Persona B	
	Distancia	Validación	Distancia	Validación
1	1.327282	Sí	1.399624	Sí
2	1.761148	No	1.206776	Sí
3	1.790497	No	1.238658	Sí
4	1.022190	Sí	1.083232	Sí
5	1.276796	Sí	1.307250	Sí
6	1.211983	Sí	1.399574	Sí
7	1.101571	Sí	1.517618	Sí
8	1.020549	Sí	1.495272	Sí
9	1.192100	Sí	1.248007	Sí
10	1.252313	Sí	1.583969	Sí

Tabla 5.11. Resultados obtenidos al validar 10 firmas auténticas por el método de coeficientes de predicción lineal.

En la tabla 5.12 se muestran los resultados obtenidos al someter a prueba un conjunto de 10 firmas falsas.

Firma	Persana A		Persona B	
	Distancia	Validación	Distancia	Validación
1	1.867695	No	1.970983	No
2	2.039706	No	1.987540	No
3	1.826567	No	2.029617	No
4	1.804512	No	1.968369	No
5	1.929965	No	2.103424	No
6	1.992353	No	2.075571	No
7	1.781131	No	1.971398	No
8	2.027397	No	2.040670	No
9	2.092619	No	1.913288	No
10	1.967854	No	1.983691	No

Tabla 5.12. Resultados obtenidos al autentificar 10 firmas falsas por el método de coeficientes de predicción lineal.

5.2 Conclusiones

En el presente trabajo se ha mostrado un novedoso dispositivo capaz de autentificar una firma autógrafa, a lo largo de su desarrollo se presentaron diversos problemas que paulatinamente tuvieron una solución aceptable, desde el diseño y construcción de la pluma hasta la elección y aplicación de las técnicas de procesamiento digital de señales y reconocimiento de patrones.

Basándose en los resultados obtenidos al aplicar los tres métodos propuestos para el reconocimiento de firma autógrafa, se puede observar que los métodos de la envolvente promediadora y CAP ofrecen los mejores resultados, mientras que el LPC presenta algunas fallas en la validación de firmas auténticas. Cabe mencionar que ninguno de los tres métodos valida firmas falsas, lo que hace seguro al sistema como medio de autentificación.

El método CAP, novedoso por su aplicación y utilizado por primera vez en este campo, ofrece mejores resultados que el de envolvente promediadora pues rechaza las firmas falsificadas con mayor énfasis. Para lograr esto, fue necesario determinar el grado y número de polinomios adecuados para las etapas de entrenamiento y reconocimiento del sistema.

Con el método LPC no se obtienen tan buenos resultados en la validación de firmas autenticas debido a la poca variación en frecuencia de las señales, además los coeficientes LPC están normalizados y no dependen de la magnitud de la señal. El método CAP es el mejor debido a los coeficientes de aproximación polinomial, que modelan los cambios de magnitud presentes en las señales provenientes de las firmas. El método de la envolvente promediadora, aunque simple en concepto, ofrece buenos resultados, pues contempla el comportamiento general de las envolventes de las firmas.

Este proyecto puede tener varias aplicaciones, tan variadas como los lugares donde se requiera tener certeza de la autenticidad de una firma autógrafa; en trámites bancarios o transacciones comerciales, por citar dos ejemplos. Este sistema está pensado para las personas físicas que requieran un medio adicional para demostrar que realmente son ellas las que están dando fe de algún trámite, con lo cual se puede evitar algún robo o fraude planeado por terceras personas que traten de imitar sus firmas. Otra aplicación puede ser el analizar el cambio que presenta la firma de una persona cuando ésta ha realizado una gran cantidad de rúbricas de manera consecutiva, tal como le sucede a personas que requieran autorizar diversas diligencias, con esto se puede ayudar a diseñar plumas ergonómicas que disminuyan la tensión muscular en la mano de la persona.

Como trabajo a futuro se pueden realizar mejoras al sistema, como el obtener una señal que represente esencialmente los movimiento de la mano, para lo cual se puede utilizar otro tipo de transductor. También se puede hacer más versátil eliminando el cable que transmite la señal eléctrica desde la pluma hasta el dispositivo amplificador, para esto se puede utilizar un pequeño dispositivo transmisor de radiofrecuencia colocado en el interior de la pluma. Además se puede mejorar la etapa de amplificación implementando un control automático de ganancia o eliminando el ruido por medios analógicos, aunque el digital ofrece buenos resultados. También se pueden buscar o proponer nuevos algoritmos de reconocimiento de patrones que ofrezcan mejores resultados, asociando medidas de distancia especialmente diseñadas para comparar patrones provenientes del tipo de señales analizadas en este trabajo. Como mejora final, se puede extender al sistema para que trabaje con la base de datos de un conjunto de personas para identificar de quién es la firma que requiere ser autentificada.

APÉNDICE A

En este apéndice se muestran los programas realizados en lenguaje C que fueron utilizados por el sistema de reconocimiento de firma autógrafa, éstos se presentan en forma secuencial de acuerdo al orden en que fueron descritos en el capítulo 4. Además se muestran los programas que sirvieron como herramientas para el desarrollo del sistema.

Listado de los programas realizados en lenguaje C

wave.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Función principal
void main(void){
      puts("\nPara empezar a grabar, pulse una tecla.");
      getch();
      //Graba el wav
      system("wavrec f00.wav 8000 64000");
      //Grafica la señal
      system("graph f00.wav");
      }
```

wavrec.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <wav.h>

//Definiciones
#define  _BLOCK      16000UL
#define  _LINEINV    0xFF
#define  _LINEINC    29<<3
#define  _AMPLIS     3<<6

//Declarativa de funciones de la librería wav.h
void create_file(unsigned char *,unsigned long,unsigned long);
size_t write_file(const void *,size_t);
void save_file(void);

//Variables globales
unsigned char aligned_physical;
unsigned char *aligned=NULL,*sbmem=NULL;
unsigned char flag;

//Escribe en el DSP
void dsp_write(unsigned char c){
      //Lee el bit 7 del Write Buffer Status
      while(inp(0x022C)&0x80);//Se espera
      //Escribe en el Write Command/Data Port
      outp(0x022C,c);
      }
```

```
//Lee del DSP
unsigned char dsp_read(void){
      //Lee el bit 7 del Read Buffer Status Register
      while(!(inp(0x022E)&0x80));//Se espera
      //Lee el Read Data Register
      return(inp(0x022A));
      }

//Resetea el DSP
void dsp_reset(void){
      int i;
      //Envía 1 al Reset Port
      outp(0x0226,0x01);
      //Envía 0 al Reset Port
      outp(0x0226,0x00);
      //Verifica que ponga 0xAA en el Read Data Register
      //Intenta 100 veces y si responde antes se sale
      for(i=0;i<100;i++){
            //Bit 7 de Read Buffer Status
            if(inp(0x022E)&0x80){
                  //Lee el Read Data Register
                  if(inp(0x022A)==0xAA)break;
                  }
            }
      //En caso de error se sale
      if(i==100){
            printf("SoundBlaster no encontrada en 0x0220\n");
            exit(1);
            }
      }

//Lee las interrupciones del DSP del Interrupt Status Register
unsigned char dsp_interrupt(void){
      unsigned char x;
      //Puerto de direcciones del mezclador
      outp(0x0224,0x82);
      //Puerto de datos del mezclador
      x=inp(0x0225)&0x03;
      //Si el DSP ejecuta una interrupción se confirma recepción
      if(x==0x01)inp(0x022E);//Disparo por transferencia de 8 bits
      if(x==0x02)inp(0x022F);//Disparo por transferencia de 16 bits
      return x;
      }

//Obtiene la versión del DSP
unsigned char *dsp_version(unsigned char *cad){
      unsigned char mayor,menor;
      //Versión del DSP
      dsp_write(0xE1);
      mayor=dsp_read();//Principal
      menor=dsp_read();//Complementarios
      sprintf(cad,"%u.%u",mayor,menor);
      return cad;
      }

//Configura la constante de tiempo para la tasa o frecuencia de muestreo
void set_time_constant(unsigned long frec){
      dsp_write(0x40);
      //Establece la frecuencia deseada
      dsp_write(256UL-(1000000UL/frec));
      }

//Configura constante de tiempo para frecuencia de muestreo de entrada
void set_time_constant_in_4xx(unsigned long frec){
```

```
      dsp_write(0x42);
      //Establece la frecuencia deseada
      dsp_write((frec>>8)&0xFF);//HI
      dsp_write(frec&0xFF);     //LO
      }

//Configura constante de tiempo para la frecuencia de salida
void set_time_constant_out_4xx(unsigned long frec){
      dsp_write(0x41);
      //Establece la frecuencia deseada
      dsp_write((frec>>8)&0xFF);//HI
      dsp_write(frec&0xFF);     //LO
      }

//Configura en el DPS la longitud del bloque para transferencias DMA
//Cuando se alcance la longitud definida, el DSP ejecuta interrupción.
void set_transfer(unsigned int len){
      //Longitud de la transferencia
      dsp_write(0x48);
      dsp_write((len>>8)&0xFF);//HI
      dsp_write(len&0xFF);     //LO
      }

//Detiene transferencia de muestras DMA de 16 bits
void stop_transfer(void){
      dsp_write(0xD5);
      }

//Finaliza Auto-Init DMA Mode de 16 bits
void stop_dma_auto_init_mode_16(void){
      dsp_write(0xD9);
      }

//Activa el altavoz
void spk_on(void){
      dsp_write(0xD1);
      }

//Desactiva el altavoz
void spk_off(void){
      dsp_write(0xD3);
      }

//Configura valores del mezclador del SoundBlaster para entrada
void set_mixer_CT1745(void){
      //Reinicia del mezclador
      outp(0x0224,0x00);outp(0x0225,0x00);
      //Volumen Voice LxR, 4x4 bits, 16 tonos
      outp(0x0224,0x04);outp(0x0225,0x00);
      //Volumen Mic, 3 bits 0-2, 8 tonos
      outp(0x0224,0x0A);outp(0x0225,0x00);
      //Volumen Master LxR, 4x4 bits, 16 tonos
      outp(0x0224,0x22);outp(0x0225,0x00);
      //Volumen MIDI LxR, 4x4 bits, 16 tonos
      outp(0x0224,0x26);outp(0x0225,0x00);
      //Volumen CD LxR, 4x4 bits, 16 tonos
      outp(0x0224,0x28);outp(0x0225,0x00);
      //Volumen LineIn LxR, 4x4 bits, 16 tonos
      outp(0x0224,0x2E);outp(0x0225,_LINEINV);
      //Volumen Master L, 5 bits 3-7, 32 tonos
      outp(0x0224,0x30);outp(0x0225,0x00);
      //Volumen Master R, 5 bits 3-7, 32 tonos
      outp(0x0224,0x31);outp(0x0225,0x00);
      //Volumen Voice L, 5 bits 3-7, 32 tonos
      outp(0x0224,0x32);outp(0x0225,0x00);
```

```
      //Volumen Voice R, 5 bits 3-7, 32 tonos
      outp(0x0224,0x33);outp(0x0225,0x00);
      //Volumen MIDI L, 5 bits 3-7, 32 tonos
      outp(0x0224,0x34);outp(0x0225,0x00);
      //Volumen MIDI R, 5 bits 3-7, 32 tonos
      outp(0x0224,0x35);outp(0x0225,0x00);
      //Volumen CD L, 5 bits 3-7, 32 tonos
      outp(0x0224,0x36);outp(0x0225,0x00);
      //Volumen CD R, 5 bits 3-7, 32 tonos
      outp(0x0224,0x37);outp(0x0225,0x00);
      //Volumen LineIn L, 5 bits 3-7, 32 tonos
      outp(0x0224,0x38);outp(0x0225,_LINEINC);
      //Volumen LineIn R, 5 bits 3-7, 32 tonos
      outp(0x0224,0x39);outp(0x0225,_LINEINC);
      //Volumen Mic, 5 bits 3-7, 32 tonos
      outp(0x0224,0x3A);outp(0x0225,0x00);
      //Volumen PCAltavoz, 5 bits 3-7, 32 tonos
      outp(0x0224,0x3B);outp(0x0225,0x00);
      //Habilitador de salida
      outp(0x0224,0x3C);outp(0x0225,0x00);
      //Habilitador de entrada L
      outp(0x0224,0x3D);outp(0x0225,0x18);
      //Habilitador de entrada R
      outp(0x0224,0x3E);outp(0x0225,0x18);
      //Amplificador entrada L, 2 bits 6-7, 4 tonos
      outp(0x0224,0x3F);outp(0x0225,_AMPLIS);
      //Amplificador entrada R, 2 bits 6-7, 4 tonos
      outp(0x0224,0x40);outp(0x0225,_AMPLIS);
      //Amplificador salida L, 2 bits 6-7, 4 tonos
      outp(0x0224,0x41);outp(0x0225,0x00);
      //Amplificador salida R, 2 bits 6-7, 4 tonos
      outp(0x0224,0x42);outp(0x0225,0x00);
      //Amplificador automático del micrófono (AGC)
      outp(0x0224,0x43);outp(0x0225,0x00);
      //Agudos L, 4 bits 4-7, 16 tonos
      outp(0x0224,0x44);outp(0x0225,0x00);
      //Agudos R, 4 bits 4-7, 16 tonos
      outp(0x0224,0x45);outp(0x0225,0x00);
      //Bajos L, 4 bits 4-7, 16 tonos
      outp(0x0224,0x46);outp(0x0225,0x00);
      //Bajos R, 4 bits 4-7, 16 tonos
      outp(0x0224,0x47);outp(0x0225,0x00);
      /**/
      }

//Asignación de memoria
//Automáticamente sólo se hace una vez
void allocate_memory(void){
      if(sbmem)return;
      //Reserva la memoria y la apunta sbmem
      if((sbmem=(unsigned char *)malloc(_BLOCK))==NULL){
            printf("Error de alojamiento de memoria\n");
            exit(1);
            }
      }

//Alineamiento de memoria
void aligned_memory(void){
      unsigned long physical;
      physical=((unsigned long)FP_OFF(sbmem))+
                        (((unsigned long)FP_SEG(sbmem))<<4);
      physical+=0x0FFFFL;
      physical&=0xF0000L;
      aligned_physical=(physical>>16)&15;
      aligned=(unsigned char *)MK_FP(((unsigned int)
```

```
                  aligned_physical<<12)&0xF000,0);
      }

//Libera la memoria asignada al segmento
//Sólo lo hace una sola vez automáticamente
void free_memory(void){
      if(sbmem)free(sbmem);
      sbmem=NULL;
      }

//Programa el DMA con la dirección inicial y longitud del bloque
void dma_set(unsigned int len){
      //Set mask bit, Canal 1 (Single Mask Register)
      outp(0x0A,0x05);
      //(Clear Byte Pointer Register)
      outp(0x0C,0x00);
      //(Mode Register) Single mode, Increment address,
      //Disable auto initialization, Write transfer, Channel 1
      outp(0x0B,0x45);
      //(Base Address Register of Channel 1)
      outp(0x02,0x00);
      outp(0x02,0x00);
      //(Page Register of Channel 1)
      //Programa al DMA la dirección inicial
      outp(0x83,aligned_physical);
      //(Transfer Count Register of Channel 1)
      //Longitud del bloque
      outp(0x03,len&0xFF);     //LO
      outp(0x03,(len>>8)&0xFF);//HI
      //Clear mask bit, Canal 1 (Single Mask Register)
      outp(0x0A,0x01);
      }

//Lee muestras de 16 bits en Auto-Init DMA Mode
void dma_auto_init_mode_16(unsigned int len){
      //Disminuye en uno
      len--;
      //Programa DMA con dir inicial y long de la transferencia
      dma_set(len);
      //Programa DSP con la longitud del bloque
      set_transfer(len);
      //Auto-Init DMA Mode y buffer FIFO
      dsp_write(0xBE);//Código
      dsp_write(0x10);//Modo, mono con signo
      len/=(unsigned int)(2);//Mitad
      dsp_write(len&0xFF);     //LO
      dsp_write((len>>8)&0xFF);//HI
      }

//Graba muestras de 16 bits y las almacena en archivo
void rec_file(unsigned char *nombre,unsigned long frec,unsigned long
muestras){
      unsigned long bytes,len=0;
      unsigned char cadver[10];
      //Reinicia DSP
      dsp_reset();
      //Obtiene la versión del DSP
      dsp_version(cadver);
      printf("\nVersión de DSP: %s",cadver);
      //Si no es versión 4.xx que se salga
      if(cadver[0]!='4'){
            puts("\nERROR: Este programa fue creado solo para tarjetas
SoundBlaster 4.xx");
            getch();
            exit(1);
```

```
            }
      //Configura mezclador para entrada
      set_mixer_CT1745();
      //Crea el archivo para almacenar las muestras
      create_file(nombre,frec,muestras);
      //Asigna memoria
      allocate_memory();
      aligned_memory();
      //Constante de tiempo
      set_time_constant_in_4xx(frec);
      printf("\nMuestras por segundo: %u",frec);
      printf("\nGrabando %lu muestras...",muestras);
      //Grabando...
      bytes=muestras*2UL;
      dma_auto_init_mode_16(_BLOCK);
      while(bytes||len){
            //Transfiere las muestras al archivo
            if(len)write_file(aligned,len);
            len=(bytes>_BLOCK)?_BLOCK:bytes;
            bytes-=len;
            //Interrumpe proceso
            if(kbhit()){
                  getch();
                  break;
                  }
            //Lee interrupciones del DSP
            while(!dsp_interrupt());
            //Actualiza DMA con dirección y longitud de la transferencia-
1
            dma_set(_BLOCK-1UL);
            }
      //Finaliza Modo Auto-Init
      stop_dma_auto_init_mode_16();
      //Detiene transferencia
      stop_transfer();
      //Salva el archivo
      save_file();
      printf("\nGrabación finalizada exitósamente.");
      }

//Módulo WAVREC
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=4){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nWAVREC [nombre] [frec] [muestras]\n");
            exit(1);
            }
      //Graba muestras en Modo REC Auto-Ini 16 bits
      rec_file(argv[1],atol(argv[2]),atol(argv[3]));
      //Libera memoria halloc
      free_memory();
      }
```

```
//Definición de librería                                          wav.h
#if !defined(__WAV_H)
#define __WAV_H

//Librerías a incluir:
#if !defined(___STRING_H)
#include <string.h>
#endif  /* __STRING_H */
```

```
#if !defined(___STDLIB_H)
#include <stdlib.h>
#endif  /* __STDLIB_H */

#if !defined(___STDIO_H)
#include <stdio.h>
#endif  /* __STDIO_H */

#if !defined(___CONIO_H)
#include <conio.h>
#endif  /* __CONIO_H */

//Estructura de WAVE
typedef struct WAVE {
      unsigned char    nombre[100];   //Nombre del archivo
      unsigned char         rid[5];   //Siempre es "RIFF"
      unsigned long         tamano;   //Tamaño del archivo con todo y
cabecera
      unsigned char         wid[5];   //Siempre es "WAVE"
      unsigned char         fid[5];   //Siempre es "fmt "
      unsigned long           tcab;   //Tamaño de la cabecera hasta este
punto
      int                     efor;   //Etiqueta de formato
      int                     ncan;   //Número de canales
      unsigned long           frec;   //Frecuencia de muestreo
muestras/seg
      unsigned long           byps;   //Número medio de bytes/seg
      int                      adb;   //Alineamiento de bloque
      int                      bpm;   //Bits por muestra (8 o 16)
      unsigned char        data[5];   //Marcador de inicio de datos "data"
      unsigned long           nbym;   //No de bytes muestreados
      fpos_t                 fposd;   //Indica posición inicio de datos
en archivo
      unsigned long       muestras;   //Número de muestras
} WAVE;

//Variables globales
FILE *parchivo=NULL;
WAVE *wave=NULL;

//Libera la memoria usada por el wave
void free_wave(WAVE *wave){
      if(!wave)return;
      if(parchivo)fclose(parchivo);
      parchivo=NULL;
      free(wave);;
      wave=NULL;
      }

//Busca en el archivo el marcador "data" y "fmt "
//Posiciona el token justo al inicio del marcador
fpos_t busca_marcador(FILE *fp,unsigned char marc[5]){
      fpos_t fpos;
      int s;
      do{
            while((s=getc(fp))!=marc[0]);
            if((s=getc(fp))==marc[1])
                  if((s=getc(fp))==marc[2])
                        if((s=getc(fp))==marc[3])
                              break;
                        else fseek(fp,-1,SEEK_CUR);
                  else fseek(fp,-1,SEEK_CUR);
            else fseek(fp,-1,SEEK_CUR);
            }while(s!=EOF);
      fseek(fp,-4,SEEK_CUR);
```

```
      fgetpos(fp,&fpos);
      return fpos;
      }

//Lee la cabecera y datos del wave
WAVE *leer_wave(unsigned char nom_arch[]){
      WAVE *wave=NULL;
      if(!nom_arch[0])return NULL;
      //Abre el archivo para solo lectura binaria
      if(((parchivo)=fopen(nom_arch,"rb"))==NULL){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",nom_arch);
            getch();
            return NULL;
            }
      //Reserva memoria a los miembros
      if((wave=(WAVE *)malloc((unsigned)(sizeof(WAVE))))==NULL){
            printf("\n\nError: No hay espacio suficiente en memoria.");
            getch();
            return NULL;
            }
      //Situa en el comienzo del archivo
      fseek(parchivo,0L,SEEK_SET);
      //Estructura de un *.wave
      /*Nom*/strcpy(wave->nombre,nom_arch);
                  busca_marcador(parchivo,"RIFF");
      /*00*/ fgets(wave->rid,5,parchivo);
      /*04*/ fread(&wave->tamano,4,1,parchivo);
                  busca_marcador(parchivo,"WAVE");
      /*08*/ fgets(wave->wid,5,parchivo);
                  busca_marcador(parchivo,"fmt ");
      /*12*/ fgets(wave->fid,5,parchivo);
      /*16*/ fread(&wave->tcab,4,1,parchivo);
      /*20*/ fread(&wave->efor,2,1,parchivo);
      /*22*/ fread(&wave->ncan,2,1,parchivo);
      /*24*/ fread(&wave->frec,4,1,parchivo);
      /*28*/ fread(&wave->byps,4,1,parchivo);
      /*32*/ fread(&wave->adb,2,1,parchivo);
      /*34*/ fread(&wave->bpm,2,1,parchivo);
                  busca_marcador(parchivo,"data");
      /*36*/ fgets(wave->data,5,parchivo);
      /*40*/ fread(&wave->nbym,4,1,parchivo);
      /*44*/ fgetpos(parchivo,&wave->fposd);
      //Calcula número de muestras
      if(wave->ncan==1){
            if(wave->bpm==8)wave->muestras=wave->nbym;
            else if(wave->bpm==16)wave->muestras=wave->nbym/2L;
            }
      else {
            if(wave->bpm==8)wave->muestras=wave->nbym/2L;
            else if(wave->bpm==16)wave->muestras=wave->nbym/4L;
            }
      return wave;
}

//Imprime la cabecera del wave
void imprime_cab(WAVE *wave){
      if(wave==NULL)return;
      printf("\n    -> %s\t    Nombre del archivo\n",wave->nombre);
      printf("       %s\t    Bloque de identificación\n",wave->rid);
      printf("       %lu\t    Tamaño del archivo en bytes\n",wave-
>tamano+8);
      printf("       %s\t    Identificador de tipo de archivo\n",wave-
>wid);
      printf("       %s\t    Identificador de formato\n",wave->fid);
```

```
      printf("       %lu\t    Tamaño de la cabecera hasta este
punto\n",wave->tcab);
      printf("       %d\t    Etiqueta de formato PCM\n",wave->efor);
      printf("    -> %d\t    Número de canales\n",wave->ncan);
      printf("    -> %lu\t    Frecuencia de muestreo
(muestras/segundo)\n",wave->frec);
      printf("       %lu\t    Número medio de bytes/segundo\n",wave-
>byps);
      printf("       %d\t    Alineamiento de bloque\n",wave->adb);
      printf("    -> %d\t    Número de bits por muestra\n",wave->bpm);
      printf("       %s\t    Marcador de comienzo de datos de las
muestras\n",wave->data);
      printf("    -> %lu\t    Número de bytes muestreados\n",wave->nbym);
      printf("    -> %lu\t    Número de muestras\n",wave->muestras);
      printf("    -> %ld\t    Posición de los datos en el
archivo\n",wave->fposd);
}

//Crea un wave con valores específicos, requiere free_wave
WAVE *crea_wave(unsigned char *nom,unsigned long frec,unsigned long
muestras,int bpm){
      WAVE *wave=NULL;
      unsigned long cte;
      cte=(unsigned long)(bpm)/8UL;
      //Reserva memoria a los miembros
      if((wave=(WAVE *)malloc((unsigned)(sizeof(WAVE))))==NULL){
            printf("\n\nError: No hay espacio suficiente en memoria.");
            getch();
            return NULL;
            }
      //Dá valores a los miembros
      /*Nom*/strcpy(wave->nombre,nom);
      /*00*/ strcpy(wave->rid,"RIFF");
      /*04*/ wave->tamano=muestras*cte+36UL;
      /*08*/ strcpy(wave->wid,"WAVE");
      /*12*/ strcpy(wave->fid,"fmt ");
      /*16*/ wave->tcab=16;
      /*20*/ wave->efor=1;
      /*22*/ wave->ncan=1;
      /*24*/ wave->frec=frec;
      /*28*/ wave->byps=frec*cte;
      /*32*/ wave->adb=1;
      /*34*/ wave->bpm=bpm;
      /*36*/ strcpy(wave->data,"data");
      /*40*/ wave->nbym=muestras*cte;
      /*fp*/ wave->fposd=44;
      /*nm*/ wave->muestras=muestras;
      return wave;
}

//Comienza a guardar el wave a archivo
int guarda_wave(WAVE *wave){
      if(wave==NULL)return NULL;
      //Crea el archivo para escritura binaria
      if((parchivo=fopen(wave->nombre,"wb"))==NULL){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",wave->nombre);
            getch();
            return NULL;
            }
      //Crea la estructura de un *.wav
      /*00*/ fputs(wave->rid,parchivo);
      /*04*/ fwrite(&wave->tamano,4,1,parchivo);
      /*08*/ fputs(wave->wid,parchivo);
      /*12*/ fputs(wave->fid,parchivo);
```

```
      /*16*/ fwrite(&wave->tcab,4,1,parchivo);
      /*20*/ fwrite(&wave->efor,2,1,parchivo);
      /*22*/ fwrite(&wave->ncan,2,1,parchivo);
      /*24*/ fwrite(&wave->frec,4,1,parchivo);
      /*28*/ fwrite(&wave->byps,4,1,parchivo);
      /*32*/ fwrite(&wave->adb,2,1,parchivo);
      /*34*/ fwrite(&wave->bpm,2,1,parchivo);
      /*36*/ fputs(wave->data,parchivo);
      /*40*/ fwrite(&wave->nbym,4,1,parchivo);
      return 1;
}

//Procesa frecuencia de un wave guardado en archivo y lo guarda en otro
void proc_frec_wave(unsigned char origen[],unsigned char
destino[],unsigned long muestras,unsigned long sub){
      unsigned long tam1,tam2,frec1,frec2,i;
      unsigned char buffer[10000];
      FILE *forigen=NULL;
      size_t bytes,temp;
      WAVE *wave=NULL;
      fpos_t fposd;

      if(!origen[0])return;
      if(!destino[0])return;
      //Obtiene los parámetros
      wave=leer_wave(origen);
      tam1=wave->nbym;
      frec1=wave->frec;
      fposd=wave->fposd;
      free_wave(wave);
      //Abre el archivo origen para solo lectura binaria
      if((forigen=fopen(origen,"rb"))==NULL){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",forigen);
            getch();
            return;
            }
      tam2=(tam1/muestras)*sub+((tam1%muestras)*sub)/muestras;
      frec2=(frec1/muestras)*sub+((frec1%muestras)*sub)/muestras;
      wave=crea_wave(destino,frec2,tam2,8);
      guarda_wave(wave);

      //Situa origen en el comienzo de datos
      fsetpos(forigen,&fposd);
      while((bytes=fread(buffer,1,muestras,forigen))>0){
            if(bytes!=muestras)
                  for(i=bytes;i<muestras;i++)
                        buffer[i]=buffer[bytes-1];
            if(sub<=muestras)
                  fwrite(buffer,1,sub,parchivo);
            else {
                  temp=sub;
                  while(temp){
                        bytes=(temp>muestras)?muestras:temp;
                        temp-=bytes;
                        fwrite(buffer,1,bytes,parchivo);
                        }
                  }
            }
      fclose(forigen);
      free_wave(wave);
      }
```

```
//  ************************************************
//  *  ADAPTACIONES DEL PROGRAMA DE GRABACIÓN  *
//  ************************************************

//Crea el archivo para almacenar las muestras con formato WAV
void create_file(unsigned char *nombre,unsigned long frec,unsigned long
muestras){
      //Crea el wave
      wave=crea_wave(nombre,frec,muestras,16);
      //Empieza a salvar
      guarda_wave(wave);
      }

//Transfiere bytes de las muestras al archivo
size_t write_file(const void *aligned,size_t laa){
      return fwrite(aligned,1,laa,parchivo);
      }

//Cierra el archivo y lo salva
void save_file(void){
      free_wave(wave);
      }

#endif  /* __WAV_H */
```

filtroPW.c

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

//Reserva memoria para los coeficientes del filtro y la ventana
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para el
filtro.");
            getch();
            exit(1);
            }
      return mem;
      }

//Realiza el filtrado pasa banda
int filtropw(unsigned char origen[],unsigned char destino[],long fs,long
fcb,long fca,int K,int N){
      unsigned char header[44];
      double fa,fb,y,*h,*v;
      fpos_t muestras,m;
      FILE *org,*dest;
      int i,*x;
      //Abre el archivo origen para solo lectura binaria
      if(!(org=fopen(origen,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
            getch();
            return 0;
            }
      //Crea al archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            getch();
```

```
            return 0;
            }
      //Reserva memoria para los coeficientes del filtro y la ventana
      h=reservector(N);
      v=reservector(N);
      //Reserva memoria para almacenar muestras de la señal
      if((x=(int *)malloc(N*sizeof(int)))==NULL){
            printf("\n\nError: No se pudo reservar memoria para almacenar
muestras.");
            getch();
            return 0;
            }
      //Calcula los coeficientes de ventana de Blackman
      for(i=0;i<N;i++)
            v[i]=0.42-0.5*cos(2.0*M_PI*(double)(i)/(double)(N-1))
                        +0.08*cos(4.0*M_PI*(double)(i)/(double)(N-1));
      //Calcula las frecuencias de corte digital
      fa=(double)(fca)/(double)(fs);
      fb=(double)(fcb)/(double)(fs);
      //Calcula el coeficiente central del filtro
      h[(N-1)/2]=2.0*(fa-fb);
      //Calcula la primera mitad del filtro
      for(i=(N+1)/2;i<N;i++)
            h[i]=(sin(2.0*M_PI*fa*(double)(i-(N-1)/2))
                    -sin(2.0*M_PI*fb*(double)(i-(N-1)/2)))
                    /(M_PI*(double)(i-(N-1)/2));
      //Copia la otra mitad de los coeficientes
      for(i=0;i<(N-1)/2;i++)
            h[i]=h[(N-1)-i];
      //Aplica la ventana de Blackman y le dá ganancia al filtro
      for(i=0;i<N;i++)
            h[i]*=v[i]*(double)(K);
      //Libera memoria de ventana
      free(v);
      //Situa el comienzo del archivo
      fseek(org,0L,SEEK_SET);
      //Copia datos de cabecera de origen
      fread(&header,44,1,org);
      //Pega datos de cabecera en destino
      fwrite(&header,44,1,dest);
      //Situa el final del archivo
      fseek(org,0L,SEEK_END);
      //Obtiene el tamaño del archivo
      if(fgetpos(org,&muestras))return 0;
      //Calcula el número de muestras
      muestras=(muestras-44)/2;
      //Situa en el comienzo de los datos
      fseek(org,44L,SEEK_SET);
      //Inicializa muestras para el filtrado
      if(muestras<(N+1)/2)return 0;
      //Carga con ceros la primera mitad
      for(i=0;i<(N-1)/2;i++)
            x[i]=0;
      //Carga con datos el resto
      for(i=(N-1)/2;i<N;i++)
            x[i]=getw(org);
      //Realiza el filtrado de la señal
      printf("\nFiltrando con pasa banda la señal %s ",origen);
      for(m=0;m<muestras;m++){
            y=0;
            //Realiza la suma
            for(i=0;i<N;i++)
                  y+=(double)(x[i])*h[i];
            //Coloca la muestra filtrada en destino
            putw((int)(y),dest);
```

```
            //Desplaza las muestras
            for(i=0;i<N-1;i++)
                  x[i]=x[i+1];
            //Carga la siguiente muestra
            x[N-1]=getw(org);
            //Si ya no hay muestras
            if(feof(org))x[N-1]=0;
            }
      //Libera memoria
      free(x);
      free(h);
      //Cierra archivos
      fclose(org);
      fclose(dest);
      printf("\nArchivo filtrado exitósamente.");
      return 1;
      }

//Módulo FILTROPW
void main(int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=8){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nFILTROPW [origen] [destino] [fs] [fcb] [fca]
[ganancia] [orden]");
            exit(1);
            }
      //Realiza el filtrado pasa banda de la señal origen
      if(!filtropw(argv[1],argv[2],atol(argv[3]),atol(argv[4]),atol(argv[
5]),atol(argv[6]),atol(argv[7]))){
            printf("\n\nError: Módulo FILTROPW no concluido
satisfactoriamente.");
            getch();
            }
      }
```

---

filtroPB.c

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

//Reserva memoria para los coeficientes del filtro y la ventana
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para el
filtro.");
            getch();
            exit(1);
            }
      return mem;
      }

//Realiza el filtrado pasa bajas
int filtropb(unsigned char origen[],unsigned char destino[],long fs,long
fca,int K,int N){
      unsigned char header[44];
      double fcd,y,*h,*v;
      fpos_t muestras,m;
      FILE *org,*dest;
      int i,*x;
      //Abre el archivo origen para solo lectura binaria
      if(!(org=fopen(origen,"rb"))){
```

```
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
            getch();
            return 0;
            }
      //Crea al archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            getch();
            return 0;
            }
      //Reserva memoria para los coeficientes del filtro y la ventana
      h=reservector(N);
      v=reservector(N);
      //Reserva memoria para almacenar muestras de la señal
      if((x=(int *)malloc(N*sizeof(int)))==NULL){
            printf("\n\nError: No se pudo reservar memoria para almacenar
muestras.");
            getch();
            return 0;
            }
      //Calcula los coeficientes de la ventana de Blackman
      for(i=0;i<N;i++)
            v[i]=0.42-0.5*cos(2.0*M_PI*(double)(i)/(double)(N-1))
                        +0.08*cos(4.0*M_PI*(double)(i)/(double)(N-1));
      //Calcula la frecuencia de corte digital
      fcd=(double)(fca)/(double)(fs);
      //Calcula el coeficiente central del filtro
      h[(N-1)/2]=2.0*fcd;
      //Calcula la primera mitad de los coeficientes del filtro
      for(i=(N+1)/2;i<N;i++)
            h[i]=sin(2.0*M_PI*fcd*(double)(i-(N-1)/2))
                  /(M_PI*(double)(i-(N-1)/2));
      //Copia la otra mitad de los coeficientes
      for(i=0;i<(N-1)/2;i++)
            h[i]=h[(N-1)-i];
      //Aplica la ventana de Blackman y le dá ganancia al filtro
      for(i=0;i<N;i++)
            h[i]*=v[i]*(double)(K);
      //Libera memoria de ventana
      free(v);
      //Situa en el comienzo del archivo
      fseek(org,0L,SEEK_SET);
      //Copia datos de cabecera de origen
      fread(&header,44,1,org);
      //Pega datos de cabecera en destino
      fwrite(&header,44,1,dest);
      //Situa en el final del archivo
      fseek(org,0L,SEEK_END);
      //Obtiene el tamaño del archivo
      if(fgetpos(org,&muestras))return 0;
      //Calcula el número de muestras
      muestras=(muestras-44)/2;
      //Situa en el comienzo de los datos
      fseek(org,44L,SEEK_SET);
      //Inicializa muestras para el filtrado
      if(muestras<(N+1)/2)return 0;
      //Carga con ceros la primera mitad
      for(i=0;i<(N-1)/2;i++)
            x[i]=0;
      //Carga con datos el resto
      for(i=(N-1)/2;i<N;i++)
            x[i]=getw(org);
```

```
      //Realiza el filtrado de la señal
      printf("\nFiltrando con pasa bajas la señal %s ",origen);
      for(m=0;m<muestras;m++){
            y=0;
            //Realiza la suma
            for(i=0;i<N;i++)
                  y+=fabs(x[i])*h[i];
            //Coloca la muestra filtrada en destino
            putw((int)(y),dest);
            //Desplaza las muestras
            for(i=0;i<N-1;i++)
                  x[i]=x[i+1];
            //Carga la siguiente muestra
            x[N-1]=getw(org);
            //Si ya no hay muestras
            if(feof(org))x[N-1]=0;
            }
      //Libera memoria
      free(x);
      free(h);
      //Cierra archivos
      fclose(org);
      fclose(dest);
      printf("\nArchivo filtrado exitósamente.");
      return 1;
      }

//Módulo FILTROPB
void main(int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=7){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nFILTROPB [origen] [destino] [fs] [fca] [ganancia]
[orden]\n");
            exit(1);
            }
      //Realiza el filtrado pasa bajas de la señal origen
      if(!filtropb(argv[1],argv[2],atol(argv[3]),atol(argv[4]),atol(argv[
5]),atol(argv[6]))){
             printf("\n\nError: Módulo FILTROPB no concluido
satisfactoriamente.");
             getch();
            }
      }
```

```
#include <stdlib.h>                                                cut.c
#include <stdio.h>
#include <conio.h>

//Retorna la componente de directa de la señal
double foffset(unsigned char nombre[]){
      fpos_t muestras,i;
      double Mp=0;
      FILE *arch;
      if(nombre==NULL)return NULL;
      //Abre el archivo para solo lectura binaria
      if(!(arch=fopen(nombre,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",nombre);
            getch();
            return NULL;
            }
```

```
      printf("\nCalculando componente de directa del archivo
%s...",nombre);
      //Situa en el final del archivo
      fseek(arch,0L,SEEK_END);
      //Obtiene el tamaño del archivo
      if(fgetpos(arch,&muestras))return NULL;
      //Calcula el número de muestras
      muestras=(muestras-44)/2;
      //Situa en el comienzo de los datos
      fseek(arch,44L,SEEK_SET);
      //Realiza la sumatoria
      for(i=0;i<muestras;i++)
         if((Mp+=(double)(getw(arch)))==EOF && feof(arch)) return NULL;
      //Calcula Mp a partir del valor de la suma
      Mp/=(double)(muestras);
      printf("\nMp=%lf",Mp);
      //Cierra el archivo
      fclose(arch);
      return Mp;
      }

//Determina la posición del inicio de señal
long MnL(unsigned char envol[],int limMn){
      FILE *arch;
      long i=-1;
      int mact;
      if(envol==NULL)return -1;
      //Abre el archivo para solo lectura binaria
      if(!(arch=fopen(envol,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",envol);
            getch();
            return -1;
            }
      printf("\nBuscando inicio de señal del archivo %s...",envol);
      //Situa en el comienzo de los datos
      fseek(arch,44L,SEEK_SET);
      do{
            //Contador
            i++;
            //Lee la muestra por comparar
            if((mact=getw(arch))==EOF && feof(arch)) return -1;
            //Criterio de inicio de señal
            }while(mact<limMn);
      //Cierra el archivo
      fclose(arch);
      printf("\nInicio de señal localizado en la muestra %ld.",i);
      return i*2L+44L;
      }

//Determina la posición del fin de señal
long MnR(unsigned char envol[],int limMn){
      fpos_t fpos;
      FILE *arch;
      int mact;
      if(envol==NULL)return -1;
      //Abre el archivo para solo lectura binaria
      if(!(arch=fopen(envol,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",envol);
            getch();
            return -1;
            }
      printf("\nBuscando fin de señal del archivo %s...",envol);
      //Situa en el final de los datos
```

```
      fseek(arch,0L,SEEK_END);
      //Obtiene la posición para el contador
      if(fgetpos(arch,&fpos))return -1;
      do{
            //Contador
            fpos-=2;
            //Lee la muestra por comparar
            if(fsetpos(arch,&fpos))return -1;
            if((mact=getw(arch))==EOF && feof(arch)) return -1;
            //Criterio de final de señal
            }while(mact<limMn);
      //Cierra el archivo
      fclose(arch);
      printf("\nFin de señal localizado en la muestra
%ld.",(long)(fpos));
      return (long)(fpos);
      }

//Recorta el ruido de los extremos de la señal
int recorta(unsigned char origen[],unsigned char destino[],long ini,long
fin){
      unsigned long mfinales,nbym,i;
      unsigned char header[44];
      FILE *org,*dest;
      if(origen==NULL)return 0;
      if(destino==NULL)return 0;
      if(ini==-1 || fin==-1)return 0;
      //Abre el archivo origen para solo lectura binaria
      if(!(org=fopen(origen,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
            getch();
            return 0;
            }
      //Crea al archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            getch();
            return 0;
            }
      printf("\nCreando archivo %s con señal sin ruido en los
extremos...",destino);
      //Situa en el comienzo del archivo
      fseek(org,0L,SEEK_SET);
      //Copia datos de cabecera de origen
      fread(&header,44,1,org);
      //Pega datos de cabecera en destino
      fwrite(&header,44,1,dest);
      //Calcula muestras totales del destino
      mfinales=(unsigned long)(fin-ini+2L)/2UL;
      //Situa en el inicio de señal
      fseek(org,ini,SEEK_SET);
      //Imprime muestras válidas de la señal
      for(i=0;i<mfinales;i++)
            putw(getw(org),dest);
      //Coloca los nuevos valores en cabecera
      //Número de bytes muestreados
      nbym=mfinales*2UL;
      fseek(dest,40L,SEEK_SET);
      fwrite(&nbym,4,1,dest);
      //Tamaño del archivo (nbym+cabecera-8)
      nbym+=36UL;
      fseek(dest,4L,SEEK_SET);
```

```
      fwrite(&nbym,4,1,dest);
      //Cierra los archivos
      fclose(org);
      fclose(dest);
      printf("\nArchivo recortado exitósamente.");
      return 1;
      }

//Módulo CUT
void main (int argc, char *argv[]){
      double offset;
      long ini,fin;
      int limMn;
      //Argumentos del programa
      if(argc!=6){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nCUT [origen PB] [origen PW] [destino PB] [destino
PW] [%tol]\n");
            exit(1);
            }
      //Obtiene componente de directa de la envolvente
      offset=foffset(argv[1]);
      //Calcula nivel de comparación
      limMn=(int)((double)(atoi(argv[5]))*offset/100.0);
      //Obtiene posición de inicio y fin de la señal en el archivo
      ini=MnL(argv[1],limMn);
      fin=MnR(argv[1],limMn);
      printf("\nSeñal encontrada entre las posiciones %ld y %ld del
archivo.",ini,fin);
      //Recorta el ruido de los extremos de la envolvente
      if(!recorta(argv[1],argv[3],ini,fin)){
            printf("\n\nError: Módulo CUT no concluido
satisfactoriamente.");
            getch();
            }
      //Recorta el ruido de los extremos de la modulada
      if(!recorta(argv[2],argv[4],ini,fin)){
            printf("\n\nError: Módulo CUT no concluido
satisfactoriamente.");
            getch();
            }
      }
```

---

norm.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Lee un archivo y retorna el máximo valor
int maximo(unsigned char nombre[]){
      int muestra,max=0;
      FILE *arch;
      if(nombre==NULL)return 0;
      //Abre el archivo solo para lectura binaria
      if(!(arch=fopen(nombre,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",nombre);
            getch();
            return 0;
            }
      printf("\nEn busca del máximo valor del archivo %s...",nombre);
      //Situa en el comienzo de los datos
      fseek(arch,44L,SEEK_SET);
      //Comienza a leer y comparar con máximo actual
```

```
      while((muestra=getw(arch))!=EOF || !feof(arch)){
            muestra=abs(muestra);
            if(max<muestra)max=muestra;
            }
      printf("\nEl máximo valor encontrado fue: %d.",max);
      //Cierra el archivo
      fclose(arch);
      //Retorna el máximo
      return max;
}

//Crea un archivo con los datos datos normalizados
int normaliza(unsigned char origen[],unsigned char destino[]){
      unsigned char header[44];
      int muestra,max;
      double factor;
      FILE *org,*dest;
      if(origen==NULL)return 0;
      if(destino==NULL)return 0;
      //Obtiene el máximo del archivo origen
      if(!(max=maximo(origen)))return 0;
      //Calcula el factor de escalamiento
      factor=(double)(32767)/(double)(max);
      //Abre el archivo origen para solo lectura binaria
      if(!(org=fopen(origen,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
            getch();
            return 0;
            }
      //Crea el archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            getch();
            return 0;
            }
      printf("\nCreando archivo %s con muestras
normalizadas...",destino);
      //Situa en el comienzo del archivo
      fseek(org,0L,SEEK_SET);
      //Copia datos de cabecera de origen
      fread(&header,44,1,org);
      //Pega los datos de cabecera en destino
      fwrite(&header,44,1,dest);
      //Lee muestra y la copia normalizada en destino
      while((muestra=getw(org))!=EOF || !feof(org)){
            muestra=(int)((double)(muestra)*factor);
            putw(muestra,dest);
            }
      //Cierra los archivos
      fclose(org);
      fclose(dest);
      printf("\nArchivo normalizado exitósamente.");
      return 1;
}

//Módulo NORM
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=3){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nNORM [origen] [destino]\n");
            exit(1);
```

```
            }
      //Crea archivo con muestras normalizadas
      if(!normaliza(argv[1],argv[2])){
            printf("\n\nError: Módulo NORM no concluido
satisfactoriamente.");
            getch();
            }
      }
```

expand.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Crea un archivo con muestras normalizadas en longitud
int expande(unsigned char origen[],unsigned char destino[],unsigned long
mfinales){
      unsigned long nbym,muestras,cont,pos;
      unsigned char header[44];
      double factor;
      FILE *org,*dest;
      if(origen==NULL)return 0;
      if(destino==NULL)return 0;
      //Abre el archivo origen para solo lectura binaria
      if(!(org=fopen(origen,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
            getch();
            return 0;
            }
      //Lee el número de bytes muestreados
      fseek(org,40L,SEEK_SET);
      fread(&nbym,4,1,org);
      //Calcula el número de muestras
      muestras=nbym/2UL;
      //Verifica que pueda expandirse
      if(muestras>mfinales){
            printf("\n\nError: No se pudo expandir el archivo %s por
tener más de %lu muestras.",origen,mfinales);
            fclose(org);
            getch();
            return 0;
            }
      //Crea al archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            getch();
            return 0;
            }
      //Calcula el factor de proporcionalidad
      factor=(double)(muestras)/(double)(mfinales);
      printf("\nCreando archivo %s de muestras expandidas...",destino);
      //Situa en el comienzo del archivo
      fseek(org,0L,SEEK_SET);
      //Copia datos de cabecera de origen
      fread(&header,44,1,org);
      //Pega datos de cabecera en destino
      fwrite(&header,44,1,dest);
      //Imprime muestras proporcionalmente
      for(cont=0;cont<mfinales;cont++){
            pos=(unsigned long)((double)(cont)*factor);
            if(fseek(org,pos*2UL+44UL,SEEK_SET))return 0;
```

```
            putw(getw(org),dest);
            }
      //Coloca los nuevos valores en cabecera
      //Número de bytes muestreados
      nbym=mfinales*2UL;
      fseek(dest,40L,SEEK_SET);
      fwrite(&nbym,4,1,dest);
      //Tamaño del archivo (nbym+cabecera-8)
      nbym+=36UL;
      fseek(dest,4L,SEEK_SET);
      fwrite(&nbym,4,1,dest);
      //Cierra los archivos
      fclose(org);
      fclose(dest);
      printf("\nArchivo expandido exitósamente.");
      return 1;
      }

//Módulo EXPAND
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=4){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nEXPAND [origen] [destino] [muestras]\n");
            exit(1);
            }
      //Crea archivo con muestras normalizadas en longitud
      if(!expande(argv[1],argv[2],atol(argv[3]))){
            printf("\n\nError: Módulo EXPAND no concluido
satisfactoriamente.");
            getch();
            }
      }
```

---

cap.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Indica error de división entre cero
void error(void){
      printf("\nERROR: Intento dividir entre cero en módulo CAP en
función \"crout\"");
      getch();
      exit(1);
      }

//Reserva memoria para matrices y vectores
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para vectores en
proceso LU.");
            getch();
            exit(1);
            }
      return mem;
      }

//Soluciona sistemas de ecuaciones por descomposición LU según Crout
int crout(double *a,double *b,double *x,int n){
      double *lu,*y,suma;
      int i,j,k;
      if(a==NULL)return 0;
```

```
	if(b==NULL)return 0;
	if(x==NULL)return 0;
	if(n==0)return 0;
	//Reserva memoria
	lu=reservector(n*n);
	y=reservector(n);
	//Calcula LU
	for (j=0;j<n;j++){
		for (i=j;i<n;i++){
			suma=0;
			for (k=0;k<j;k++)
				suma+=(lu[n*i+k]*lu[n*k+j]);
			lu[n*i+j]=a[n*i+j]-suma;
			}
		for (i=j+1;i<n;i++){
			suma=0;
			for (k=0;k<j;k++)
				suma+=(lu[n*j+k]*lu[n*k+i]);
			if(!lu[n*j+j])error();
			lu[n*j+i]=(a[n*j+i]-suma)/lu[n*j+j];
			}
		}
	for(i=0;i<n;i++){
		suma=0;
		for(j=0;j<=i-1;j++)
			suma+=(lu[n*i+j]*y[j]);
		if(!lu[n*i+i])error();
		y[i]=(b[i]-suma)/lu[n*i+i];
		}
	//Solucióna el sistema de ecuaciones:
	for(i=n-1;i>=0;i--){
		suma=0;
		for(j=i+1;j<n;j++)
			suma+=(lu[n*i+j]*x[j]);
		x[i]=y[i]-suma;
		}
	//Libera memoria
	free(lu);
	free(y);
	return 1;
	}

//Retorna el cuadrado de un valor
double powint(int base,int exp){
	double val=1;
	int i;
	for(i=0;i<exp;i++)
		val*=(double)(base);
	return val;
	}

//Calcula los fij
double fij(int i,int j,int N){
	double suma=0;
	int n;
	for(n=0;n<N;n++)
		suma+=powint(n,i+j);
	return suma;
	}

//Calcula los bi
double bi(int *x,int i,int N){
	double suma=0;
	int n;
	for(n=0;n<N;n++)
```

```
            suma+=(double)(x[n])*powint(n,i);
      return suma;
      }

//Calcula los coeficientes CAP por segmentos de la señal
int cap(unsigned char origen[],unsigned char destino[],int polos,long
segmentos){
      long tamseg,contseg;
      FILE *org,*dest;
      double *a,*b,*x;
      int *data,i,j;
      fpos_t fpos;
      if(origen==NULL)return 0;
      if(destino==NULL)return 0;
      //Abre el archivo origen para solo lectura binaria
      if(!(org=fopen(origen,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
            getch();
            return 0;
            }
      //Crea el archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            getch();
            return 0;
            }
      printf("\nCalculando coeficientes CAP del archivo %s...",origen);
      //Reserva memoria para matriz A, vector B de TI y solución x
      a=reservector(polos*polos);
      b=reservector(polos);
      x=reservector(polos);
      //Situa el final del archivo
      fseek(org,0L,SEEK_END);
      //Obtiene el tamaño del archivo
      if(fgetpos(org,&fpos))return 0;
      //Calcula el número de muestras
      tamseg=((long)(fpos)-44L)/2L;
      //Calcula el tamaño de los segmentos
      tamseg/=segmentos;
      //Reserva memoria para almacenar muestras del segmento
      if((data=(int *)malloc((tamseg)*sizeof(int)))==NULL){
            printf("\n\nError: No se pudo reservar memoria para almacenar
muestras.");
            fclose(dest);
            fclose(org);
            getch();
            return 0;
            }
      //Salva número de segmentos en archivo destino
      fwrite(&segmentos,sizeof(long),1,dest);
      //Salva número de polos por segmento
      putw(polos,dest);
      //Situa el comienzo de los datos
      fseek(org,44L,SEEK_SET);
      //Calcula los CAP
      for(contseg=0;contseg<segmentos;contseg++){
            //Lee el segmento en cuestión
            if(fread(data,sizeof(int),tamseg,org)!=tamseg)return 0;
            //Calcula la matriz A y el vector b de TI
            for(i=0;i<polos;i++){
                  b[i]=bi(data,i,tamseg);
                  for(j=0;j<polos;j++)
```

```
                        a[polos*i+j]=fij(i,j,tamseg);
                  }
            //Resuelve el sistema Ax=b
            if(!crout(a,b,x,polos))return 0;
            //Guarda en archivo los CAP calculados
            if(fwrite(x,sizeof(double),polos,dest)!=polos)return 0;
            }
      //Libera memoria
      free(data);
      free(a);
      free(b);
      free(x);
      //Cierra archivos
      fclose(org);
      fclose(dest);
      printf("\nCálculo de coeficientes CAP concluido exitósamente.");
      return 1;
      }

//Módulo CAP
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=5){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nCAP [origen] [destino] [polos] [número de
segmentos]\n");
            exit(1);
            }
      //Calcula los CAP del archivo origen
      if(!cap(argv[1],argv[2],atol(argv[3]),atol(argv[4]))){
            printf("\n\nError: Módulo CAP no concluido
satisfactoriamente.");
            getch();
            }
      }
```

---

lpc.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Indica error de división entre cero
void error(void){
      printf("\nERROR: Intento de dividir entre cero en módulo LPC
función \"crout\"");
      getch();
      exit(1);
      }

//Reserva memoria para matrices y vectores
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para vectores en
proceso LU.");
            getch();
            exit(1);
            }
      return mem;
      }

//Soluciona sistemas de ecuaciones por descomposición LU según Crout
int crout(double *a,double *b,double *x,int n){
      double *lu,*y,suma;
```

```
	int i,j,k;
	if(a==NULL)return 0;
	if(b==NULL)return 0;
	if(x==NULL)return 0;
	if(n==0)return 0;
	//Reserva memoria
	lu=reservector(n*n);
	y=reservector(n);
	//Calcula LU
	for (j=0;j<n;j++){
		for (i=j;i<n;i++){
			suma=0;
			for (k=0;k<j;k++)
				suma+=(lu[n*i+k]*lu[n*k+j]);
			lu[n*i+j]=a[n*i+j]-suma;
			}
		for (i=j+1;i<n;i++){
			suma=0;
			for (k=0;k<j;k++)
				suma+=(lu[n*j+k]*lu[n*k+i]);
			if(!lu[n*j+j])error();
			lu[n*j+i]=(a[n*j+i]-suma)/lu[n*j+j];
			}
		}
	for(i=0;i<n;i++){
		suma=0;
		for(j=0;j<=i-1;j++)
			suma+=(lu[n*i+j]*y[j]);
		if(!lu[n*i+i])error();
		y[i]=(b[i]-suma)/lu[n*i+i];
		}
	//Soluciona el sistema de ecuaciones:
	for(i=n-1;i>=0;i--){
		suma=0;
		for(j=i+1;j<n;j++)
			suma+=(lu[n*i+j]*x[j]);
		x[i]=y[i]-suma;
		}
	//Libera memoria
	free(lu);
	free(y);
	return 1;
	}

//Calcula los fij
double fij(int *data,int i,int j,int polos,int N){
	double suma=0;
	int n;
	for(n=0;n<N;n++)
		suma+=(double)(data[polos+n-i])*
				(double)(data[polos+n-j]);
	return suma;
	}

//Calcula los coeficientes LPC por segmentos de la señal
int lpc(unsigned char origen[],unsigned char destino[],int polos,long
segmentos){
	long tamseg,contseg;
	FILE *org,*dest;
	double *a,*b,*x;
	int *data,i,j;
	fpos_t fpos;
	if(origen==NULL)return 0;
	if(destino==NULL)return 0;
	//Abre el archivo origen para solo lectura binaria
```

```
      if(!(org=fopen(origen,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
            getch();
            return 0;
            }
      //Crea al archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            getch();
            return 0;
            }
      printf("\nCalculando coeficientes LPC del archivo %s...",origen);
      //Reserva memoria para matriz A, vector B de TI y solución x
      a=reservector(polos*polos);
      b=reservector(polos);
      x=reservector(polos);
      //Situa en el final del archivo
      fseek(org,0L,SEEK_END);
      //Obtiene el tamaño del archivo
      if(fgetpos(org,&fpos))return 0;
      //Calcula el número de muestras
      tamseg=((long)(fpos)-44L)/2L;
      //Calcula el tamaño de los segmentos
      tamseg/=segmentos;
      //Reserva memoria para almacenar muestras del segmento
      if((data=(int *)malloc((tamseg+polos)*sizeof(int)))==NULL){
            printf("\n\nError: No se pudo reservar memoria para almacenar
muestras.");
            fclose(dest);
            fclose(org);
            getch();
            return 0;
            }
      //Salva número de segmentos en archivo destino
      fwrite(&segmentos,sizeof(long),1,dest);
      //Salva número de polos por segmento
      putw(polos,dest);
      //Situa en el comienzo de los datos
      fseek(org,44L,SEEK_SET);
      //Carga ceros en las primeras p muestras
      for(i=0;i<polos;i++)
            data[i]=0;
      //Calcula de los LPC
      for(contseg=0;contseg<segmentos;contseg++){
            //Lee el segmento en cuestión desplazandolo p muestras
            if(fread(data+polos,sizeof(int),tamseg,org)!=tamseg)return 0;
            //Calcula la matriz A y el vector b de TI
            for(i=0;i<polos;i++){
                  b[i]=fij(data,i+1,0,polos,tamseg);
                  for(j=0;j<polos;j++)
                        a[polos*i+j]=fij(data,i+1,j+1,polos,tamseg);
                  }
            //Resuelve el sistema Ax=b
            if(!crout(a,b,x,polos))return 0;
            //Guarda en archivo los LPC calculados
            if(fwrite(x,sizeof(double),polos,dest)!=polos)return 0;
            //Coloca las últimas p muestras al inicio de la memoria
            for(i=0;i<polos;i++)
                  data[i]=data[tamseg+i];
            }
      //Libera memoria
      free(data);
```

```
      free(a);
      free(b);
      free(x);
      //Cierra archivos
      fclose(org);
      fclose(dest);
      printf("\nCálculo de coeficientes LPC concluido exitósamente.");
      return 1;
      }

//Módulo LPC
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=5){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nLPC [origen] [destino] [polos] [número de
segmentos]\n");
            exit(1);
            }
      //Calcula los LPC del archivo con la señal
      if(!lpc(argv[1],argv[2],atol(argv[3]),atol(argv[4]))){
            printf("\n\nError: Módulo LPC no concluido
satisfactoriamente.");
            getch();
            }
      }
```

envprom.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Genera la envolvente promedio de un grupo menor o igual a 10 firmas
int promenv10(unsigned char baseorg[],unsigned char destino[],int
archivos){
      unsigned long n,muestras;
      unsigned char cad[100];
      FILE *dest,*org[10];
      double prom;
      int i,mues;
      if(baseorg==NULL)return 0;
      if(destino==NULL)return 0;
      if(!archivos || archivos>10)return 0;
      //Abre los archivos de las envolventes
      for(i=0;i<archivos;i++){
            //Genera cadena con nombre del archivo
            sprintf(cad,"%s%0.1d.wav",baseorg,i);
            //Abre el archivo para solo lectura binaria
            if(!(org[i]=fopen(cad,"rb"))){
                  printf("\n\nError: No se ha podido abrir el archivo
\"%s\"",cad);
                  getch();
                  return 0;
                  }
            //Situa en el comienzo de los datos
            fseek(org[i],44L,SEEK_SET);
            }
      //Crea el archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            getch();
            return 0;
            }
```

```
      printf("\nGenerando archivo %s con envolvente
promedio...",destino);
      //Situa en el comienzo del primer archivo
      fseek(org[0],0L,SEEK_SET);
      //Copia sus datos de cabecera
      fread(&cad,44,1,org[0]);
      //Pega datos de cabecera en destino
      fwrite(&cad,44,1,dest);
      //Lee el número de bytes muestreados
      fseek(org[0],40L,SEEK_SET);
      fread(&muestras,4,1,org[0]);
      //Calcula el número de muestras
      muestras/=2UL;
      //Situa en el comienzo de los datos
      fseek(org[0],44L,SEEK_SET);
      //Promedia muestras correspondientes y guarda en destino
      for(n=0;n<muestras;n++){
            prom=0;
            for(i=0;i<archivos;i++){
                  if((mues=getw(org[i]))==EOF && feof(org[i]))return 0;
                  prom+=(double)(mues);
                  }
            prom/=(double)(archivos);
            putw((int)(prom),dest);
            }
      //Cierra destino
      fclose(dest);
      //Cierra archivos origen
      for(i=0;i<archivos;i++)
            fclose(org[i]);
      printf("\nArchivo de envolvente promedio generado exitósamente.");
      return 1;
      }

//Genera la envolvente promedio total de un grupo de firmas múltiplo de
10
int promenv(unsigned char baseorg[],unsigned char destino[]){
      unsigned char cad1[100],cad2[100];
      int k,archivos=0;
      FILE *arch;
      if(baseorg==NULL)return 0;
      if(destino==NULL)return 0;
      do{//Cuenta los archivos a promediar
            sprintf(cad1,"%s%0.2d.wav",baseorg,archivos);
            if((arch=fopen(cad1,"rb"))!=NULL)archivos++;
            if(arch)fclose(arch);
            }while(arch);
      //Si no hay archivos se sale
      if(!archivos){
            printf("\n\nError: No se encontraron archivos con base
%s.",baseorg);
            getch();
            return 0;
            }
      //Si no es múltiplo de 10 se sale
      if(archivos%10){
            printf("\n\nError: Número de archivos \"%d\" no es múltiplo
de 10.",archivos);
            getch();
            return 0;
            }
      printf("\nSe encontraron %d archivos con base
%s.",archivos,baseorg);
      //Crea promedio de grupos de 10 firmas
      for(k=0;k<archivos/10;k++){
```

```
            sprintf(cad1,"%s%0.1d",baseorg,k);
            sprintf(cad2,"%sk%0.1d.wav",baseorg,k);
            if(!promenv10(cad1,cad2,10)){
                  printf("\n\nError: No se pudo crear archivo de
envolvente de grupo\"%s\"",cad2);
                  getch();
                  return 0;
                  }
            }
      printf("\nGenerando archivo %s con envolvente promedio
total...",destino);
      sprintf(cad1,"%sk",baseorg);
      if(!promenv10(cad1,destino,archivos/10)){
            printf("\n\nError: No se pudo crear archivo de envolvente
promedio total\"%s\"",destino);
            getch();
            return 0;
            }
      printf("\nArchivo de envolvente promedio total generado
exitósamente.");
      return 1;
      }

//Módulo ENVPROM
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=3){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nENVPROM [base origen] [destino]\n");
            exit(1);
            }
      //Genera la envolvente promedio total
      if(!promenv(argv[1],argv[2])){
            printf("\n\nError: Módulo ENVPROM no concluido
satisfactoriamente.");
            getch();
            }
      }
```

---

pat2vec.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Copia coeficientes CAP o LPC de uno de los archivos origen al único
destino
//Retorna número de vectores agregados y comprueba compatibilidad de
polos
long copia(unsigned char origen[],FILE *fdest,int npg){
      unsigned char buffer[1024];
      long nvec;
      size_t bytes;
      FILE *forg;
      int polos;
      if(origen==NULL)return 0;
      if(fdest==NULL)return 0;
      //Abre el archivo origen para solo lectura binaria
      if(!(forg=fopen(origen,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
            getch();
            return 0;
            }
      //Situa en el inicio del archivo
```

```
      fseek(forg,0L,SEEK_SET);
      //Obtiene número de vectores de coeficientes CAP o LPC
      if(fread(&nvec,sizeof(long),1,forg)!=1)return 0;
      //Obtiene la dimensión de los vectores
      if((polos=getw(forg))!=npg){
            printf("\n\nError: Número de polos de \"%s\" (%d) diferente
al global (%d).",origen,polos,npg);
            fclose(forg);
            getch();
            return 0;
            }
      printf("\nCargando archivo %s en archivo de vectores...",origen);
      //Vacía el contenido de uno al archivo de vectores
      bytes=fread(buffer,1,1024,forg);
      while(bytes){
            fwrite(buffer,1,bytes,fdest);
            bytes=fread(buffer,1,1024,forg);
            }
      //Cierra archivo
      fclose(forg);
      printf("\nArchivo cargado exitósamente.");
      return nvec;
      }

//Concatena archivos de coeficientes CAP o LPC en un solo archivo
int pat2vec(unsigned char tipopat[],unsigned char baseorg[],unsigned char
destino[],int npg){
      long vectotales=0;
      unsigned char cad[100];
      int i,archivos=0;
      FILE *dest;
      if(baseorg==NULL)return 0;
      if(destino==NULL)return 0;
      do{//Cuenta los archivos por vaciar
            sprintf(cad,"%s%0.2d.%s",baseorg,archivos,tipopat);
            if((dest=fopen(cad,"rb"))!=NULL)archivos++;
            if(dest)fclose(dest);
            } while(dest);
      if(!archivos){
            printf("\n\nError: No se encontraron archivos con base
%s.",baseorg);
            getch();
            return 0;
            }
      printf("\nSe encontraron %d archivos con base
%s.",archivos,baseorg);
      //Crea el archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            getch();
            return 0;
            }
      printf("\nGenerando archivo de vectores %s...",destino);
      //Salva número de vectores provisional en destino
      fwrite(&vectotales,sizeof(long),1,dest);
      //Salva dimensión de los vectores
      putw(npg,dest);
      //Carga archivos en destino y actualiza número de vectores
      for(i=0;i<archivos;i++){
            sprintf(cad,"%s%0.2d.%s",baseorg,i,tipopat);
            if(!(vectotales+=copia(cad,dest,npg)))return 0;
            }
      //Situa en el inicio del destino
      fseek(dest,0L,SEEK_SET);
```

```
      //Salva número de vectores definitivo en destino
      fwrite(&vectotales,sizeof(long),1,dest);
      //Cierra archivo
      fclose(dest);
      printf("\nArchivo de vectores generado exitósamente.");
      return 1;
}

//Módulo PAT2VEC
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=5){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nPAT2VEC [tipo patrones] [base origen] [destino]
[dimensión]\n");
            exit(1);
            }
      //Concatena archivos de coeficientes CAP o LPC en un solo archivo
      if(!pat2vec(argv[1],argv[2],argv[3],atol(argv[4]))){
            printf("\n\nError: Módulo PAT2VEC no concluido
satisfactoriamente.");
            getch();
            }
      }
```

```
#include <stdlib.h>                                                capmeans.c
#include <stdio.h>
#include <conio.h>
#include <math.h>

//Indica error de división entre cero
void error(void){
      printf("\nERROR: Intento de dividir entre cero en módulo
CAPMEANS");
      getch();
      exit(1);
      }

//Reserva memoria para matrices y vectores
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para
vectores.");
            getch();
            exit(1);
            }
      return mem;
      }

//Realiza el agrupamiento de vectores en clases por método de CapMeans
int capmeans(unsigned char origen[],unsigned char destino[],int clases){
      long nvec,contv,nz;
      FILE *org,*dest;
      int i,p,polos;
      double *x,*Z;

      if(origen==NULL)return 0;
      if(destino==NULL)return 0;
      //Abre el archivo origen para solo lectura binaria
      if(!(org=fopen(origen,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
```

```
            getch();
            return 0;
            }
      //Crea el archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            getch();
            return 0;
            }
      printf("\nAgrupando patrones CAP en clases del archivo
%s...",origen);
      //Situa en el inicio del archivo
      fseek(org,0L,SEEK_SET);
      //Obtiene el número de vectores
      fread(&nvec,sizeof(long),1,org);
      //Obtiene la dimensión de los vectores
      polos=getw(org);
      //Comprueba compatibilidad
      if(nvec%(long)(clases)){
            printf("\n\nError: No. de patrones de \"%s\" (%ld) no
múltiplo del no. de clases (%d).",origen,nvec,clases);
            fclose(org);
            fclose(dest);
            getch();
            return 0;
            }
      //Reserva memoria para almacenar patrones
      x=reservector(polos);
      //Matriz que alverga los centroides
      Z=reservector(clases*polos);
      //Inicia con ceros la suma de patrones
      for(i=0;i<clases*polos;i++)
            Z[i]=0;
      //Suma los patrones en las clases
      for(contv=0;contv<nvec;contv++){
            //Lee al patron CAP en turno
            if(fread(x,sizeof(double),polos,org)!=polos)return 0;
            //Suma patron en su respectiva clase
            for(p=0;p<polos;p++)
                  Z[(int)(contv%(long)(clases))*polos+p]+=x[p];
            }
      //Vectores por clase, checa que no de cero
      if(!(nz=nvec/(long)(clases)))error();
      //Calcula los centroides (promedio de los xi)
      for(i=0;i<clases*polos;i++)
            Z[i]/=(double)(nz);
      //Salva número de centroides
      nvec=(long)(clases);
      fwrite(&nvec,sizeof(long),1,dest);
      //Salva dimensión de centroides
      putw(polos,dest);
      //Salva los centroides de las clases en archivo
      if(fwrite(Z,sizeof(double),clases*polos,dest)!=clases*polos)return
0;
      //Libera memoria
      free(x);
      free(Z);
      //Cierra archivos
      fclose(org);
      fclose(dest);
      printf("\nAgrupación de patrones CAP concluida exitósamente.");
      return 1;
}
```

```
//Módulo CAPMEANS
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=4){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nCAPMEANS [origen] [destino] [clases]\n");
            exit(1);
            }
      //Agrupa patrones CAP en clases
      if(!capmeans(argv[1],argv[2],atol(argv[3]))){
            printf("\n\nError: Módulo CAPMEANS no concluido
satisfactoriamente.");
            getch();
            }
      }
```

kmeans.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>

//Indica error de división entre cero
void error(void){
      printf("\nERROR: Intento de dividir entre cero en módulo KMEANS.");
      getch();
      exit(1);
      }

//Reserva memoria para matrices y vectores
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para
vectores.");
            getch();
            exit(1);
            }
      return mem;
      }

//Retorna el cuadrado de un valor
double cuad(double val){
      return val*val;
      }

//Realiza el agrupamiento de vectores en clases por método de K-Means
int kmeans(unsigned char origen[],unsigned char destino[],int clases){
      double mod,min,*x,*Z,*Z2,*temp;
      int i,p,polos,W,sig,*nz;
      long nvec,contv;
      FILE *org,*dest;
      if(origen==NULL)return 0;
      if(destino==NULL)return 0;
      //Abre el archivo origen para solo lectura binaria
      if(!(org=fopen(origen,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origen);
            getch();
            return 0;
            }
      //Crea al archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
```

```
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            getch();
            return 0;
            }
      printf("\nAgrupando vectores en clases del archivo %s...",origen);
      //Situa en el inicio del archivo
      fseek(org,0L,SEEK_SET);
      //Obtiene número de vectores
      fread(&nvec,sizeof(long),1,org);
      //Obtiene la dimensión de los vectores
      polos=getw(org);
      //Reserva memoria para almacenar bloque de vectores
      x=reservector(polos);
      //Matrices que alvergan a los centroides de las clases
      Z=reservector(clases*polos);
      Z2=reservector(clases*polos);
      //Inicia con valores arbitrarios, primeros k vectores
      if(fread(Z,sizeof(double),clases*polos,org)!=clases*polos)return 0;
      //Reserva memoria para cuenta de vectores por clase
      if((nz=(int *)malloc(clases*sizeof(int)))==NULL){
            printf("\n\nError: No se pudo reservar memoria para cuenta de
vectores.");
            getch();
            return 0;
            }
      putchar('\n');
      do{//Inicio de ciclo iterativo
            putchar('@');//Indicador de iteraciones
            //Inicia con ceros la suma de vectores
            for(i=0;i<clases*polos;i++)
                  Z2[i]=0;
            //Inicializa contador de vectores por clase
            for(i=0;i<clases;i++)
                  nz[i]=0;
            //Situa en el comienzo de los datos
            fseek(org,6L,SEEK_SET);
            //Calcula los centroides
            for(contv=0;contv<nvec;contv++){
                  //Lee bloque de vectores
                  fread(x,sizeof(double),polos,org);
                  //Obtiene la menor magnitud y asigna vector a clase
                  min=1.7E+308;//Valor auxiliar
                  for(i=0;i<clases;i++){
                        mod=0;
                        for(p=0;p<polos;p++)
                              mod+=cuad(Z[i*polos+p]-x[p]);
                        mod=sqrt(mod);
                        if(mod<min){
                              min=mod;
                              W=i;
                              }
                        }
                  //Suma vector en su respectiva clase
                  for(p=0;p<polos;p++)
                        Z2[W*polos+p]+=x[p];
                  //Incrementa cuenta de vectores de dicha clase
                  nz[W]++;
                  }
            //Calcula de los nuevos centroides (promedio de los xi)
            for(i=0;i<clases;i++)
                  for(p=0;p<polos;p++){
                        if(!nz[i])error();
                        Z2[i*polos+p]/=(double)(nz[i]);
```

```
                        }
            //Cambia a los nuevos centroides (via punteros)
            temp=Z;
            Z=Z2;
            Z2=temp;
            //Criterio de comparación entre centroides
            sig=0;
            for(i=0;i<clases*polos;i++)
                  if(Z2[i]!=Z[i])
                        sig=1;
      }while(sig);//Fin de ciclo iterativo
      //Salva número de centroides
      nvec=(long)(clases);
      fwrite(&nvec,sizeof(long),1,dest);
      //Salva dimensión de centroides
      putw(polos,dest);
      //Salva los centroides de las clases en archivo
      if(fwrite(Z,sizeof(double),clases*polos,dest)!=clases*polos)return
0;
      //Libera memoria
      free(x);
      free(Z);
      free(Z2);
      free(nz);
      //Cierra archivos
      fclose(org);
      fclose(dest);
      printf("\nAgrupación de vectores en clases concluida
exitósamente.");
      return 1;
}

//Módulo KMEANS
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=4){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nKMEANS [origen] [destino] [clases]\n");
            exit(1);
            }
      //Agrupa vectores en clases por K-Means
      if(!kmeans(argv[1],argv[2],atol(argv[3]))){
            printf("\n\nError: Módulo KMEANS no concluido
satisfactoriamente.");
            getch();
            }
      }
```

---

entrena.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Sistema de entrenamiento para EP, CAP y LPC
int entrena(unsigned char baseorg[],int cpol,long cseg,int lpol,long
lseg){
      unsigned char cad[200];
      int i,archivos=0;
      FILE *org;
      if(baseorg==NULL)return 0;
      do{//Cuenta los archivos
            sprintf(cad,"%s%0.2d.wav",baseorg,archivos);
            if((org=fopen(cad,"rb"))!=NULL)archivos++;
            if(org)fclose(org);
```

```
            } while(org);
      if(!archivos){
            printf("\n\nError: No se encontraron archivos con base
%s.",baseorg);
            getch();
            return 0;}
      printf("\nSe encontraron %d archivos con base
%s.",archivos,baseorg);
      printf("\n@Inicia proceso de entrenamiento:");

      //Preprocesamiento mixto
      for(i=0;i<archivos;i++){
            //Filtra las firmas con filtro paso banda
            sprintf(cad,"FILTROPW %s%0.2d.wav %sww%0.2d.wav 8000 800 1000
1 501",baseorg,i,baseorg,i);
            system(cad);
            //Filtra las firmas con filtro paso bajas para obtener sus
envolventes
            sprintf(cad,"FILTROPB %sww%0.2d.wav %swb%0.2d.wav 8000 10 6
501",baseorg,i,baseorg,i);
            system(cad);
            //Recorta el ruido de los extremos de las firmas
            sprintf(cad,"CUT %swb%0.2d.wav %sww%0.2d.wav %snb%0.2d.wav
%snw%0.2d.wav 10",baseorg,i,baseorg,i,baseorg,i,baseorg,i);
            system(cad);
            //Crea archivo de señal modulada normalizado en amplitud
            sprintf(cad,"NORM %snw%0.2d.wav
%sw%0.2d.wav",baseorg,i,baseorg,i);
            system(cad);
            //Crea archivo de envolvente normalizado en amplitud
            sprintf(cad,"NORM %snb%0.2d.wav
%seb%0.2d.wav",baseorg,i,baseorg,i);
            system(cad);
            //Crea archivo de envolvente normalizado en amplitud y
longitud
            sprintf(cad,"EXPAND %seb%0.2d.wav %sb%0.2d.wav
64000",baseorg,i,baseorg,i);
            system(cad);
            //Genera los CAP de las envolventes de las firmas
            sprintf(cad,"CAP %sb%0.2d.wav %sb%0.2d.cap %d
%ld",baseorg,i,baseorg,i,cpol,cseg);
            system(cad);
            //Genera los LPC de las señales moduladas
            sprintf(cad,"LPC %sw%0.2d.wav %sw%0.2d.lpc %d
%ld",baseorg,i,baseorg,i,lpol,lseg);
            system(cad);
            //Agrupa patrones LPC en clases por K-Means
            sprintf(cad,"KMEANS %sw%0.2d.lpc %sw%0.2d.kms
16",baseorg,i,baseorg,i);
            system(cad);}

      //Entrenamiento por envolvente promedio
      //Crea la envolvente promedio del grupo de firmas
      sprintf(cad,"ENVPROM %sb %sbp.wav",baseorg,baseorg);
      system(cad);
      //Calcula estadísticos de distancias de ECM de envolventes
      sprintf(cad,"STAT wav %sb %d",baseorg,archivos);
      system(cad);

      //Entrenameinto por coeficientes CAP
      //Crea archivo de vectores a partir de los de patrones CAP
      sprintf(cad,"PAT2VEC cap %sb %sb.vec %d",baseorg,baseorg,cpol);
      system(cad);
      //Agrupa vectores de patrones CAP en clases por Capmeans
      sprintf(cad,"CAPMEANS %sb.vec %sb.kms %d",baseorg,baseorg,cseg);
```

```
      system(cad);
      //Calcula estadísticos de distancias euclidianas de centroides CAP
      sprintf(cad,"STAT cap %sb %d",baseorg,archivos);
      system(cad);

      //Entrenamiento por coeficientes LPC
      //Crea archivo de vectores a partir de los de patrones LPC
      sprintf(cad,"PAT2VEC lpc %sw %sw.vec %d",baseorg,baseorg,lpol);
      system(cad);
      //Agrupa vectores de patrones LPC en clases por K-Means
      sprintf(cad,"KMEANS %sw.vec %sw.kms 16",baseorg,baseorg);
      system(cad);
      //Calcula estadísticos de distancias euclidianas de centroides LPC
      sprintf(cad,"STAT kms %sw %d",baseorg,archivos);
      system(cad);

      //Valida firmas del entrenamiento por EP
      printf("\n\nFirmas del entrenamiento por EP:");
      for(i=0;i<archivos;i++){
            sprintf(cad,"VALIDA %sbp %sb%0.2d.wav",baseorg,baseorg,i);
            system(cad);
            getch();}

      //Valida firmas del entrenamiento por CAP
      printf("\n\nFirmas del entrenamiento por CAP:");
      for(i=0;i<archivos;i++){
            sprintf(cad,"VALIDA %sb %sb%0.2d.cap",baseorg,baseorg,i);
            system(cad);
            getch();}

      //Valida firmas del entrenamiento por LPC
      printf("\n\nFirmas del entrenamiento por LPC:");
      for(i=0;i<archivos;i++){
            sprintf(cad,"VALIDA %sw %sw%0.2d.kms",baseorg,baseorg,i);
            system(cad);
            getch();}

      printf("\n@Proceso de entrenamiento culminado exitósamente.");
      return 1;
      }

//Módulo ENTRENA
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=6){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nENTRENA [base origen] [cpol] [cseg] [lpol]
[lseg]\n");
            exit(1);
            }
      //Sistema de entrenamiento para EP, CAP y LPC
      if(!entrena(argv[1],atol(argv[2]),atol(argv[3]),atol(argv[4]),atol(
argv[5]))){
            printf("\n\nError: Módulo ENTRENA no concluido
satisfactoriamente.");
            getch();} }
```

```
#include <string.h>                                                stat.c
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
```

```
//Importa valores provenientes de otros módulos
int fgetval(void *ptr, size_t size, size_t n){
      FILE *file;
      //Abre archivo de retorno como archivo binario
      if(!(file=fopen("return.val","rb"))){
            printf("\n\nError: No se ha podido leer archivo
\"return.val\"");
            getch();
            return 0;
            }
      //Lee valores a importar
      if(fread(ptr,size,n,file)!=n)return 0;
      fclose(file);
      return n;
      }

//Reserva memoria para matrices y vectores
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para
vectores.");
            getch();
            exit(1);
            }
      return mem;
      }

//Retorna el cuadrado de un valor
double cuad(double val){
      return val*val;
      }

//Calcula estadísticos de distancias entre centroides o envolventes
int stat(unsigned char ext[],unsigned char baseorg[],int archivos){
      double *d,coefvar,desv=0,prom=0;
      unsigned char cad[100];
      int i,ecm=0;
      FILE *dest;
      if(ext==NULL)return 0;
      if(baseorg==NULL)return 0;
      if(archivos<=0)return 0;
      //Declara bandera indicadora de ECM
      if(!strcmp(strlwr(ext),"wav"))ecm=1;
      //Crea el archivo destino como archivo binario
      if(ecm)sprintf(cad,"%sp.stt",baseorg);
      else sprintf(cad,"%s.stt",baseorg);
      if(!(dest=fopen(cad,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",cad);
            getch();
            return 0;
            }
      printf("\nCalculando estadisticos de distancias para %s...",ext);
      //Reserva memoria para distancias
      d=reservector(archivos);
      //Calcula las distancias
      for(i=0;i<archivos;i++){
            //Calcula la distancia entre centroides o envolventes
            if(ecm)sprintf(cad,"ERRORCM %s.wav %s%0.2d.wav
r.wav",baseorg,baseorg,i);
            else sprintf(cad,"EUCLIDES %s.kms
%s%0.2d.%s",baseorg,baseorg,i,ext);
            system(cad);
            //Carga la distancia resultante en memoria
```

```
            if(fgetval(&d[i],sizeof(double),1)!=1)return 0;
            //Suma las distancias
            prom+=d[i];
            }
      //Imprime las distancias
      for(i=0;i<archivos;i++)
            printf("\nd(%d)=%lf",i,d[i]);
      //Obtiene el promedio
      prom/=(double)(archivos);
      printf("\nProm=%lf",prom);
      //Obtiene la desviación estándar de la muestra
      for(i=0;i<archivos;i++)
            desv+=cuad(d[i]-prom);
      desv=sqrt(desv/(double)(archivos-1));
      printf("\nDesv=%lf",desv);
      //Obtiene el coeficiente de variación
      coefvar=desv/prom;
      printf("\nCvar=%lf",coefvar);
      //Imprime no de datos en destino
      putw(archivos,dest);
      //Imprime la distancia promedio
      fwrite(&prom,sizeof(double),1,dest);
      //Imprime la desviación estándar
      fwrite(&desv,sizeof(double),1,dest);
      //Imprime el coeficiente de variación
      fwrite(&coefvar,sizeof(double),1,dest);
      //Imprime las distancias resultantes
      if(fwrite(d,sizeof(double),archivos,dest)!=archivos)return 0;
      //Libera memoria
      free(d);
      //Cierra archivo
      fclose(dest);
      printf("\nCálculo de estadísticos concluido exitósamente.");
      return 1;
      }

//Módulo STAT
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=4){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nSTAT [ext asociada] [base origen] [número de
archivos]\n");
            exit(1);
            }
      //Calcula estadísticos de las distancias
      if(!stat(argv[1],argv[2],atol(argv[3]))){
            printf("\n\nError: Módulo STAT no concluido
satisfactoriamente.");
            getch();
            }
      }
```

---

euclides.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>

//Exporta valores para uso de otros módulos
int fputval(const void *ptr, size_t size, size_t n){
      FILE *file;
      //Crea archivo de retorno como archivo binario
      if(!(file=fopen("return.val","wb"))){
```

```
            printf("\n\nError: No se ha podido crear archivo
\"return.val\"");
            getch();
            return 0;
            }
      //Imprime valores a exportar
      if(fwrite(ptr,size,n,file)!=n)return 0;
      fclose(file);
      return n;
      }

//Reserva memoria para matrices y vectores
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para
vectores.");
            getch();
            exit(1);
            }
      return mem;
      }

//Retorna el cuadrado de un valor
double cuad(double val){
      return val*val;
      }

//Compara centroides de clases por distancia euclidiana
int euclides(unsigned char prototipo[],unsigned char muestra[]){
      double *ZP,*ZM,mod,total=0;
      long i,clasesp,clasesm;
      int p,polosp,polosm;
      FILE *fpro,*fmues;
      if(prototipo==NULL)return 0;
      if(muestra==NULL)return 0;
      //Abre el archivo prototipo para solo lectura binaria
      if(!(fpro=fopen(prototipo,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",prototipo);
            getch();
            return 0;
            }
      //Abre el archivo muestra para solo lectura binaria
      if(!(fmues=fopen(muestra,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",muestra);
            fclose(fpro);
            getch();
            return 0;
            }
      printf("\nComparando clases de %s contra %s...",muestra,prototipo);
      //Situa en el inicio del prototipo
      fseek(fpro,0L,SEEK_SET);
      //Obtiene número de vectores
      if(fread(&clasesp,sizeof(long),1,fpro)!=1)return 0;
      //Obtiene la dimensión de los vectores
      polosp=getw(fpro);
      //Situa en el inicio de la muestra
      fseek(fmues,0L,SEEK_SET);
      //Obtiene número de vectores
      if(fread(&clasesm,sizeof(long),1,fmues)!=1)return 0;
      //Obtiene la dimensión de los vectores
      polosm=getw(fmues);
      //Comprueba compatibilidad
```

```
      if(clasesp!=clasesm || polosp!=polosm){
            printf("\n\nError: No son compatibles \"%s\" y
\"%s\".",prototipo,muestra);
            fclose(fmues);
            fclose(fpro);
            getch();
            return 0;
            }
      //Vectores para los centroides
      ZP=reservector(polosp);
      ZM=reservector(polosp);
      //Calcula para cada centroide
      for(i=0;i<clasesp;i++){
            //Carga par de centroides y los compara
            if(fread(ZP,sizeof(double),polosp,fpro)!=polosp)return 0;
            if(fread(ZM,sizeof(double),polosp,fmues)!=polosp)return 0;
            //Calcula la magnitud de la diferencia
            mod=0;
            for(p=0;p<polosp;p++)
                  mod+=cuad(ZM[p]-ZP[p]);
            mod=sqrt(mod);
            total+=mod;
            }
      //Calcula la distancia promedio
      total/=(double)(clasesp);
      printf("\nMDT=%lf",total);
      //Exporta valor para el módulo que lo requiera
      if(fputval(&total,sizeof(double),1)!=1)return 0;
      //Libera memoria
      free(ZP);
      free(ZM);
      //Cierra archivos
      fclose(fpro);
      fclose(fmues);
      printf("\nComparación de clases concluida exitósamente.");
      return 1;
      }

//Módulo EUCLIDES
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=3){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nEUCLIDES [prototipo] [muestra]\n");
            exit(1);
            }
      //Compara centroides de clases por distancia euclidiana
      if(!euclides(argv[1],argv[2])){
            printf("\n\nError: Módulo EUCLIDES no concluido
satisfactoriamente.");
            getch();
            }
      }
```

---

errorcm.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>

//Exporta valores para uso de otros módulos
int fputval(const void *ptr, size_t size, size_t n){
      FILE *file;
      //Crea archivo de retorno como archivo binario
```

```
      if(!(file=fopen("return.val","wb"))){
            printf("\n\nError: No se ha podido crear archivo
\"return.val\"");
            getch();
            return 0;
            }
      //Imprime valores a exportar
      if(fwrite(ptr,size,n,file)!=n)return 0;
      fclose(file);
      return n;
      }

//Retorna el cuadrado de un valor
double cuad(double val){
      return val*val;
      }

//Calcula el error cuadrático medio y crea archivo con la resta de las
señales
int errorcm(unsigned char original[],unsigned char estimado[],unsigned
char destino[]){
      unsigned char header[44];
      FILE *org,*est,*dest;
      fpos_t muestras;
      int morg,mest;
      double E=0;
      if(original==NULL)return 0;
      if(estimado==NULL)return 0;
      if(destino==NULL)return 0;
      //Abre el archivo original para solo lectura binaria
      if(!(org=fopen(original,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",original);
            getch();
            return 0;
            }
      //Abre el archivo estimado para solo lectura binaria
      if(!(est=fopen(estimado,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",estimado);
            fclose(org);
            getch();
            return 0;
            }
      //Crea el archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(org);
            fclose(est);
            getch();
            return 0;
            }
      printf("\nCalculando ECM y creando archivo %s de resta de
señales...",destino);
      //Situa en el final del original
      fseek(org,0L,SEEK_END);
      //Obtiene el tamaño del original
      if(fgetpos(org,&muestras))return 0;
      //Calcula el número de muestras
      muestras=(muestras-44)/2;
      //Situa en el comienzo del original
      fseek(org,0L,SEEK_SET);
      //Copia datos de cabecera
      fread(&header,44,1,org);
```

```
	//Pega datos de cabecera en destino
	fwrite(&header,44,1,dest);
	//Situa en comienzo de muestras del estimado
	fseek(est,44L,SEEK_SET);
	//Lee la muestra original y la estimada
	while((morg=getw(org))!=EOF || !feof(org)){
		if((mest=getw(est))==EOF && feof(est)) return 0;
		//Guarda la diferencia
		putw(morg-mest,dest);
		//Suma los cuadrados de las diferencias
		E+=cuad(morg-mest);
		}
	//Error cuadrático medio, escalado con raiz para reducir valores
	E=sqrt(E/(double)(muestras));
	//Cierra los archivos
	fclose(org);
	fclose(est);
	fclose(dest);
	printf("\nDistancia de Error Cuadrático Medio, E=%lf",E);
	//Exporta valor para el módulo que lo requiera
	if(fputval(&E,sizeof(double),1)!=1)return 0;
	printf("\nArchivo de resta de señales creado exitósamente.");
	return 1;
	}

//Módulo ERRORCM
void main (int argc, char *argv[]){
	//Argumentos del programa
	if(argc!=4){
		printf("\nPor favor inténtelo del siguiente modo:");
		printf("\nERRORCM [original] [estimado] [destino]\n");
		exit(1);
		}
	//Calula el error cuadrático medio y crea archivo con la resta de
las señales
	if(!errorcm(argv[1],argv[2],argv[3])){
		printf("\n\nError: Módulo ERRORCM no concluido
satisfactoriamente.");
		getch();
		}
	}
```

```
#include <stdlib.h>                                              reconoce.c
#include <stdio.h>
#include <conio.h>

//Quita la extensión de la ruta del archivo
void quitaext(unsigned char *ruta){
	int i;
	//Encuentra la posición del punto
	for(i=0;ruta[i]!='.'&&ruta[i];i++);
	//Coloca caracter de fin de cadena
	if(ruta[i]=='.')ruta[i]=0;
	}

//Sistema de reconocimiento para EP, CAP y LPC
int reconoce(unsigned char baseprot[],unsigned char muestra[],int
cpol,long cseg,int lpol,long lseg){
	unsigned char cad[200];
	int i;
	if(baseprot==NULL)return 0;
	if(muestra==NULL)return 0;
	//Quita la extensión
```

```
      quitaext(muestra);
      printf("\n@Inicia proceso de reconocimiento:");

      //PREPROCESAMIENTO
      //Filtra la firma con filtro paso banda
      sprintf(cad,"FILTROPW %s.wav %sww.wav 8000 800 1000 1
501",muestra,muestra);
      system(cad);
      //Filtra la firma con filtro paso bajas para obtener su envolvente
      sprintf(cad,"FILTROPB %sww.wav %swb.wav 8000 10 6
501",muestra,muestra);
      system(cad);
      //Recorta el ruido de los extremos de la firma
      sprintf(cad,"CUT %swb.wav %sww.wav %snb.wav %snw.wav
10",muestra,muestra,muestra,muestra);
      system(cad);
      //Crea archivo de señal modulada normalizado en amplitud
      sprintf(cad,"NORM %snw.wav %sw.wav",muestra,muestra);
      system(cad);
      //Crea archivo de envolvente normalizado en amplitud
      sprintf(cad,"NORM %snb.wav %seb.wav",muestra,muestra);
      system(cad);
      //Crea archivo de envolvente normalizado en amplitud y longitud
      sprintf(cad,"EXPAND %seb.wav %sb.wav 48000",muestra,muestra);
      system(cad);
      //Genera los CAP de la envolvente de la firma
      sprintf(cad,"CAP %sb.wav %sb.cap %d
%ld",muestra,muestra,cpol,cseg);
      system(cad);
      //Genera los LPC de la señal modulada
      sprintf(cad,"LPC %sw.wav %sw.lpc %d
%ld",muestra,muestra,lpol,lseg);
      system(cad);
      //Agrupa patrones LPC en clases por K-Means
      sprintf(cad,"KMEANS %sw.lpc %sw.kms 16",muestra,muestra);
      system(cad);

      //Reconocimiento de firma por EP
      printf("\n\nReconocimiento de firma por EP:");
      sprintf(cad,"VALIDA %sbp %sb.wav",baseprot,muestra);
      system(cad);
      //Reconocimiento de firma por CAP
      printf("\n\nReconocimiento de firma por CAP:");
      sprintf(cad,"VALIDA %sb %sb.cap",baseprot,muestra);
      system(cad);
      //Reconocimiento de firma por LPC
      printf("\n\nReconocimiento de firma por LPC:");
      sprintf(cad,"VALIDA %sw %sw.kms",baseprot,muestra);
      system(cad);
      printf("\n@Proceso de reconocimiento culminado exitósamente.");
      return 1;
      }

//Módulo RECONOCE
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=7){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nRECONOCE [baseprot] [muestra] [cpol] [cseg] [lpol]
[lseg]\n");
            exit(1);
            }
      //Sistema de reconocimiento para EP, CAP y LPC
      if(!reconoce(argv[1],argv[2],atol(argv[3]),atol(argv[4]),atol(argv[
5]),atol(argv[6]))){
```

```
            printf("\n\nError: Módulo RECONOCE no concluido
satisfactoriamente.");
            getch();
            }
       }
```

valida.c

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Importa valores provenientes de otros módulos
int fgetval(void *ptr, size_t size, size_t n){
      FILE *file;
      //Abre archivo de retorno como archivo binario
      if(!(file=fopen("return.val","rb"))){
            printf("\n\nError: No se ha podido leer archivo
\"return.val\"");
            getch();
            return 0;
            }
      //Lee valores a importar
      if(fread(ptr,size,n,file)!=n)return 0;
      fclose(file);
      return n;
      }

//Retorna 1 si la extensión es wav para caso ECM
int ecm(unsigned char arch[]){
      int i;
      for(i=0;arch[i]!='.'&&arch[i];i++);
      return !strcmp(strlwr(arch+i+1),"wav");;
      }

//Valida firma de acuerdo a estadísticos de distancia
int valida(unsigned char baseprot[],unsigned char muestra[]){
      double prom,desv,d,max,min;
      unsigned char cad[100];
      FILE *org;
      if(baseprot==NULL)return 0;
      if(muestra==NULL)return 0;
      //Abre el archivo de estadísticos
      sprintf(cad,"%s.stt",baseprot);
      if(!(org=fopen(cad,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",cad);
            getch();
            return 0;
            }
      printf("\nValidando firma \"%s\"...",muestra);
      //Situa en inicio de información
      fseek(org,2L,SEEK_SET);
      //Lee promedio
      fread(&prom,sizeof(double),1,org);
      //Lee desviación estándar
      fread(&desv,sizeof(double),1,org);
      //Cierra archivo
      fclose(org);
      //Calcula máximo del intervalo
      max=prom+2.0*desv;
      printf("\nmax=%lf",max);
      //Calcula mínimo del intervalo
      min=prom-2.0*desv;
```

```
      printf("\nmin=%lf",min);
      //Calcula la distancia entre centroides o envolventes
      if(ecm(muestra))sprintf(cad,"ERRORCM %s.wav %s
r.wav",baseprot,muestra);
      else sprintf(cad,"EUCLIDES %s.kms %s",baseprot,muestra);
      system(cad);
      //Carga la distancia resultante a la memoria
      if(fgetval(&d,sizeof(double),1)!=1)return 0;
      printf("\n%s\td=%lf\t  ",muestra,d);
      //Validación de la firma
      if(d>=min && d<max) printf("VALIDA");
      else printf("NO VALIDA");
      printf("\nValidación de firma concluida exitósamente.");
      return 1;
      }

//Módulo VALIDA
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=3){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nVALIDA [base prototipo] [muestra]\n");
            exit(1);
            }
      //Valida firma de acuerdo a estadísticos de distancia
      if(!valida(argv[1],argv[2])){
            printf("\n\nError: Módulo VALIDA no concluido
satisfactoriamente.");
            getch();
            }
      }
```

Programas utilizados como herramientas de desarrollo

verhexa.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Visualiza un archivo binario en hexadecimal
int verhexa(unsigned char nombre[]){
      int val,ch,cont=0;
      FILE *arch;
      //Abre el archivo origen para solo lectura binaria
      if(!(arch=fopen(nombre,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",nombre);
            getch();
            return 0;
            }
      clrscr();
      while(((val=getw(arch))!=EOF || !feof(arch)) && ch!=27){
            printf("%0.4X ",val);
            if(cont++==383){
                  ch=getch();
                  clrscr();
                  cont=0;
                  }
            }
      if(ch!=27)getch();
      //Cierra archivo
      fclose(arch);
```

```
	return 1;
	}

//Módulo VERHEXA
void main (int argc, char *argv[]){
	//Argumentos del programa
	if(argc!=2){
		printf("\nPor favor inténtelo del siguiente modo:");
		printf("\nVERHEXA [archivo]\n");
		exit(1);
		}
	//Visualiza un archivo binario en hexadecimal
	if(!verhexa(argv[1])){
		printf("\n\nError: Módulo VERHEXA no concluido
satisfactoriamente.");
		getch();
		}
	}
```

---

header.c

```
#include <wav.h>

//Corrige la cabecera del wav para que los datos comiencen en la 44L
int retype_header(unsigned char origen[],unsigned char destino[]){
	unsigned char buffer[1024];
	unsigned long frec,muestras;
	FILE *forigen=NULL;
	size_t bytes;
	fpos_t fposd;
	if(!origen[0])return 0;
	if(!destino[0])return 0;
	//Obtiene los parámetros
	wave=leer_wave(origen);
	//Sale si no es 16b y 1 canal
	if(wave->ncan!=1)return 0;
	if(wave->bpm!=16)return 0;
	frec=wave->frec;
	muestras=wave->muestras;
	fposd=wave->fposd;
	free_wave(wave);
	//Abre el archivo origen para solo lectura binaria
	if((forigen=fopen(origen,"rb"))==NULL){
		printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",forigen);
		getch();
		return 0;
		}
	printf("\nCreando archivo %s con cabecera corregida...",destino);
	//Crea el archivo con la nueva cabecera
	wave=crea_wave(destino,frec,muestras,16);
	guarda_wave(wave);
	//Situa origen en el comienzo de datos
	fsetpos(forigen,&fposd);
	//Vacía el contenido de uno al otro archivo
	bytes = fread(buffer, 1, 1024, forigen);
	while(bytes){
		fwrite(buffer, 1, bytes, parchivo);
		bytes = fread(buffer, 1, 1024, forigen);
		}
	//Cierra el archivo
	fclose(forigen);
	//Libera el wave
	free_wave(wave);
	printf("\nArchivo corregido exitósamente.");
```

```
      return 1;
      }

//Módulo HEADER
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=3){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nHEADER [origen] [destino]\n");
            exit(1);
            }
      //Crea archivo con cabecera corregida
      if(!retype_header(argv[1],argv[2])){
            printf("\n\nError: Módulo HEADER no concluido
satisfactoriamente.");
            getch();
            }
      }
```

graph.c

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <wav.h>

//Resolución grafica:
#define  _CEJE      LIGHTBLUE
#define  _CSENAL    LIGHTRED
#define  _CCUADROS  GREEN
#define  _MODO      _1024x768
#define  _NX        7
#define  _NY        5
//Definición de los modos
#define  _320x200   0
#define  _640x400   1
#define  _640x480   2
#define  _800x600   3
#define  _1024x768  4

//Resolución gráfica por defecto
int huge func(void){
      return _MODO;
      }

//Inicia el modo gráfico
void inicia_graficos(void){
      //Autodecta
      int tarjeta=DETECT,modo,error_cod;
      //Instala SVGA 256 colores
      installuserdriver("Svga256",func);
      //Inicializa gráficos
      initgraph(&tarjeta,&modo,"");
      //Lee resultado de la inicialización
      error_cod=graphresult();
      //En caso de error
      if (error_cod != grOk){
            printf("\n\nError de gráficos:
%s\n",grapherrormsg(error_cod));
            printf("Presione tecla para salir");
            getch();
            exit(1);
            }
      }
```

```
//Finaliza el modo gráfico
void finaliza_graficos(void){
      closegraph();
      }

//Cuadricula la pantalla
void cuadricula(void){
      int maxx,maxy,i;
      float incx,incy;
      //Coord máximas
      maxx=getmaxx();
      maxy=getmaxy();
      incx=(float)(maxx)/(float)(_NX);
      incy=(float)(maxy)/(float)(_NY);
      //Limpia pantalla
      cleardevice();
      //Pinta cuadro
      setcolor(_CCUADROS);
      rectangle(0,0,maxx,maxy);
      //Pinta grids
      for(i=1;i<_NY;i++)
            line(0,i*incy,maxx,i*incy);
      for(i=1;i<_NX;i++)
            line(i*incx,0,i*incx,maxy);
      //Pinta eje
      setcolor(_CEJE);
      line(0,maxy*0.5,maxx,maxy*0.5);
      }

//Coloca la imagen en pantalla
void pinta_wave(WAVE *wave){
      unsigned long cont;
      float incx,incy;
      int maxx,maxy;
      //Si no hay wave sale
      if(wave==NULL)return;
      if(wave->ncan!=1)return;
      if(wave->bpm!=16)return;
      //Coord máximas
      maxx=getmaxx();
      maxy=getmaxy();
      //Calcula los incrementos
      incx=(float)(maxx)/(float)(wave->muestras-1);
      incy=(float)(maxy)/65535.0;
      cuadricula();
      //Situa en el inicio de datos
      fseek(parchivo,44L,SEEK_SET);
      //Pinta señal
      setcolor(_CSENAL);
      moveto(0,maxy-32768*incy);
      for(cont=0;cont<wave->muestras&&!kbhit();cont++)
            lineto(cont*incx,maxy-(getw(parchivo)+32768)*incy);
      }

//Coloca los puntos en pantalla
void imprime_wave(WAVE *wave){
      unsigned long i;
      int val,ch;
      clrscr();
      if(wave==NULL)return;
      if(wave->ncan!=1)return;
      if(wave->bpm!=16)return;
      //Situa en el origen de los datos
      fseek(parchivo,44L,SEEK_SET);
```

```
      for(i=0;i<wave->muestras&&ch!=27;i++){
            val=getw(parchivo);
            printf("\n\tm[%0.10lu] = %0.4X->%0.5d-
>%0.4f",i,val,val,(float)(val)/32768.0);
            if(!((i+1)%25))ch=getch();
            }
      }

//Módulo GRAPH
void main(int argc, char *argv[]){
      //Apuntador a  wave
      WAVE *wave;
      //Argumentos del programa
      if(argc!=2){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nGRAPH [archivo]\n");
            exit(1);
            }
      //Carga el wave
      if(!(wave=leer_wave(argv[1])))return;
      //Muestra cabecera
      imprime_cab(wave);
      getch();
      //Imprime las muestras
      imprime_wave(wave);
      //Pinta las muestras
      inicia_graficos();
      //Grafica un archivo wav
      pinta_wave(wave);
      getch();
      finaliza_graficos();
      //Libera el wave
      free_wave(wave);
      }
```

---

pat2txt.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Reserva memoria para matrices y vectores
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para vectores en
proceso LU.");
            getch();
            exit(1);
            }
      return mem;
      }

//Crea archivo de patrones en modo texto
int pat2txt(unsigned char origenp[],unsigned char destino[]){
      FILE *orgp,*dest;
      long nvec,contv;
      int p,polos;
      double *x;
      if(origenp==NULL)return 0;
      if(destino==NULL)return 0;
      //Abre el archivo origen polos para solo lectura binaria
      if(!(orgp=fopen(origenp,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origenp);
```

```
            getch();
            return 0;
            }
      //Crea el archivo destino como archivo de texto
      if(!(dest=fopen(destino,"wt"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(orgp);
            getch();
            return 0;
            }
      printf("\nImprimiendo coeficientes en modo texto de
%s...",origenp);
      //Reserva memoria para x
      x=reservector(polos);
      //Situa en el inicio del archivo de coeficientes
      fseek(orgp,0L,SEEK_SET);
      //Obtiene número de vectores de coeficientes
      if(fread(&nvec,sizeof(long),1,orgp)!=1)return 0;
      //Obtiene la dimensión de los vectores
      polos=getw(orgp);
      //Inicia la impresión en modo texto
      for(contv=0;contv<nvec;contv++){
            //Lee de archivo los coeficientes
            if(fread(x,sizeof(double),polos,orgp)!=polos)return 0;
            //Impresión del patrón
            for(p=0;p<polos;p++)
                  fprintf(dest,"a(%d)=%0.15lf\n",p,x[p]);
            fprintf(dest,"\n");
            }
      //Libera memoria
      free(x);
      //Cierra archivos
      fclose(orgp);
      fclose(dest);
      printf("\nImpresión concluida exitósamente.");
      return 1;
      }

//Módulo PAT2TXT
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=3){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nPAT2TXT [origen patrones] [destino txt]\n");
            exit(1);
            }
      //Crea archivo de patrones en modo texto
      if(!pat2txt(argv[1],argv[2])){
            printf("\n\nError: Módulo PAT2TXT no concluido
satisfactoriamente.");
            getch();
            }
      }
```

---

capi.c

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

//Reserva memoria para matrices y vectores
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
```

```
            printf("\nError: No se pudo reservar memoria para
vectores.");
            getch();
            exit(1);
            }
      return mem;
      }

//Retorna el cuadrado de un valor
double powint(int base,int exp){
      double val=1;
      int i;
      for(i=0;i<exp;i++)
            val*=(double)(base);
      return val;
      }

//Reconstruye la señal a partir de los coeficientes CAP
int capi(unsigned char origend[],unsigned char origenp[],unsigned char
destino[],long tamvent){
      unsigned char header[44];
      FILE *orgd,*orgp,*dest;
      int n,j,polos,*data;
      double muestra,*x;
      long nvec,contv;
      if(origend==NULL)return 0;
      if(origenp==NULL)return 0;
      if(destino==NULL)return 0;
      //Abre el archivo origen data solo para lectura binaria
      if(!(orgd=fopen(origend,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origend);
            getch();
            return 0;
            }
      //Abre el archivo origen polos para solo lectura binaria
      if(!(orgp=fopen(origenp,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origenp);
            fclose(orgd);
            getch();
            return 0;
            }
      //Crea el archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(orgd);
            fclose(orgp);
            getch();
            return 0;
            }
      printf("\nPrediciendo señal a partir de coeficientes CAP
%s...",origenp);
      //Reserva memoria para x
      x=reservector(polos);
      //Reserva memoria para almacenar muestras de la ventana
      if((data=(int *)malloc(tamvent*sizeof(int)))==NULL){
            printf("\n\nError: No se pudo reservar memoria para almacenar
muestras.");
            fclose(orgd);
            fclose(orgp);
            fclose(dest);
            getch();
            return 0;
```

```
            }
      //Situa en el inicio del archivo de coeficientes
      fseek(orgp,0L,SEEK_SET);
      //Obtiene el número de vectores de coeficientes CAP
      if(fread(&nvec,sizeof(long),1,orgp)!=1)return 0;
      //Obtiene la dimensión de los vectores
      polos=getw(orgp);
      //Situa en el inicio del archivo de datos
      fseek(orgd,0L,SEEK_SET);
      //Copia datos de cabecera de origen
      fread(&header,44,1,orgd);
      //Pega datos de cabecera en destino
      fwrite(&header,44,1,dest);
      //Inicia predicción de muestras
      for(contv=0;contv<nvec;contv++){
            //Lee la ventana en cuestión
            if(fread(data,sizeof(int),tamvent,orgd)!=tamvent)return 0;
            //Lee de archivo los coeficientes CAP
            if(fread(x,sizeof(double),polos,orgp)!=polos)return 0;
            //Estimación
            for(n=0;n<tamvent;n++){
                  muestra=0;
                  for(j=0;j<polos;j++)
                        muestra+=x[j]*powint(n,j);
                  putw((int)(muestra),dest);
                  }
            }
      //Rellena hueco de ventana incompleta del final
      if((contv=fread(data,sizeof(int),tamvent,orgd))>0L)
            fwrite(data,sizeof(int),contv,dest);
      //Libera memoria
      free(data);
      free(x);
      //Cierra archivos
      fclose(orgd);
      fclose(orgp);
      fclose(dest);
      printf("\nPredicción concluida exitósamente.");
      return 1;
}

//Módulo CAPI
void main (int argc, char *argv[]){
      //Argumentos del programa
      if(argc!=5){
            printf("\nPor favor inténtelo del siguiente modo:");
            printf("\nCAPI [origen datos] [origen cap] [destino]
[tamvent]\n");
            exit(1);
            }
      //Reconstruye la señal a partir de coeficientes CAP
      if(!capi(argv[1],argv[2],argv[3],atol(argv[4]))){
            printf("\n\nError: Módulo CAPI no concluido
satisfactoriamente.");
            getch();
            }
      }
```

```
#include <stdlib.h>                                                    lpci.c
#include <stdio.h>
#include <conio.h>

//Reserva memoria para matrices y vectores
```

```
double *reservector(size_t n){
      double *mem;
      if((mem=(double *)malloc(n*sizeof(double)))==NULL){
            printf("\nError: No se pudo reservar memoria para
vectores.");
            getch();
            exit(1);
            }
      return mem;
      }

//Genera el archivo a partir de los coeficientes LPC y muestras
anteriores
int lpci(unsigned char origend[],unsigned char origenp[],unsigned char
destino[],long tamvent){
      unsigned char header[44];
      FILE *orgd,*orgp,*dest;
      int i,j,polos,*data;
      double futura,*x;
      long nvec,contv;
      if(origend==NULL)return 0;
      if(origenp==NULL)return 0;
      if(destino==NULL)return 0;
      //Abre el archivo origen data para solo lectura binaria
      if(!(orgd=fopen(origend,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origend);
            getch();
            return 0;
            }
      //Abre el archivo origen polos para solo lectura binaria
      if(!(orgp=fopen(origenp,"rb"))){
            printf("\n\nError: No se ha podido localizar el archivo
\"%s\"",origenp);
            fclose(orgd);
            getch();
            return 0;
            }
      //Crea el archivo destino como archivo binario
      if(!(dest=fopen(destino,"wb"))){
            printf("\n\nError: No se ha podido crear el archivo
\"%s\"",destino);
            fclose(orgd);
            fclose(orgp);
            getch();
            return 0;
            }
      printf("\nPrediciendo señal a partir de coeficientes LPC
%s...",origenp);
      //Reserva memoria para x
      x=reservector(polos);
      //Reserva memoria para almacenar muestras de la ventana
      if((data=(int *)malloc(tamvent*sizeof(int)))==NULL){
            printf("\n\nError: No se pudo reservar memoria para almacenar
muestras.");
            fclose(orgd);
            fclose(orgp);
            fclose(dest);
            getch();
            return 0;
            }
      //Situa en el inicio del archivo de coeficientes
      fseek(orgp,0L,SEEK_SET);
      //Obtiene número de vectores de coeficientes LPC
      if(fread(&nvec,sizeof(long),1,orgp)!=1)return 0;
```

```
	//Obtiene la dimensión de los vectores
	polos=getw(orgp);
	//Situa en el inicio del archivo de datos
	fseek(orgd,0L,SEEK_SET);
	//Copia datos de cabecera de origen
	fread(&header,44,1,orgd);
	//Pega datos de cabecera en destino
	fwrite(&header,44,1,dest);
	//Inicia predicción de muestras
	for(contv=0;contv<nvec;contv++){
		//Lee la ventana en cuestión
		if(fread(data,sizeof(int),tamvent,orgd)!=tamvent)return 0;
		//Lee del archivo los coeficientes LPC
		if(fread(x,sizeof(double),polos,orgp)!=polos)return 0;
		//Guarda las primeras p muestras
		if(fwrite(data,sizeof(int),polos,dest)!=polos)return 0;
		//Estimación
		for(i=polos;i<tamvent;i++){
			futura=0;
			for(j=0;j<polos;j++)
				futura+=x[j]*(double)(data[i-j-1]);
			putw((int)(futura),dest);
			}
		}
	//Rellena hueco de ventana incompleta del final
	if((contv=fread(data,sizeof(int),tamvent,orgd))>0L)
		fwrite(data,sizeof(int),contv,dest);
	//Libera memoria
	free(data);
	free(x);
	//Cierra archivos
	fclose(orgd);
	fclose(orgp);
	fclose(dest);
	printf("\nPredicción concluida exitósamente.");
	return 1;
	}

//Módulo LPCI
void main (int argc, char *argv[]){
	//Argumentos del programa
	if(argc!=5){
		printf("\nPor favor inténtelo del siguiente modo:");
		printf("\nLPCI [origen datos] [origen lpc] [destino]
[tamvent]\n");
		exit(1);
		}
	//Genera archivo a partir de los LPC y muestras anteriores
	if(!lpci(argv[1],argv[2],argv[3],atol(argv[4]))){
		printf("\n\nError: Módulo LPCI no concluido
satisfactoriamente.");
		getch();
		}
	}
```

APÉNDICE B
CONSTANCIAS Y RECONOCIMIENTOS

En este apéndice se muestra la primera página del trabajo presentado en el XVIII congreso de instrumentación SOMI, realizado en la Torre de Ingeniería UNAM, del 6 al 10 de Octubre del año 2003, con el cual se obtuvo Mención Honorífica por haber presentado el mejor trabajo en el área de materiales, sensores y películas delgadas. Además, se incluyen las constancias de participación en el congreso y las de vinculación con el Instituto de Investigaciones en Materiales.

APÉNDICE B
CONSTANCIAS Y RECONOCIMIENTOS

En este apéndice se muestra la primera página del trabajo presentado en el XVIII congreso de instrumentación SOMI, realizado en la Torre de Ingeniería UNAM, del 6 al 10 de Octubre del año 2003, con el cual se obtuvo Mención Honorífica por haber presentado el mejor trabajo en el área de materiales, sensores y películas delgadas. Además, se incluyen las constancias de participación en el congreso y las de vinculación con el Instituto de Investigaciones en Materiales.

PLUMA MAGNETOELASTICA PARA RECONOCIMIENTO DE FIRMA AUTOGRAFA

I. Betancourt Reyes, M. Ramírez Álvarez y P. F. Padilla Monroy
Instituto de Investigación en Materiales, UNAM. Apdo Pos 70-360, México D.F. 04510
israelb@correo.unam.mx, mra@tutopia.com, polofpm@mailbanamex.com

RESUMEN

En este trabajo, se describe el funcionamiento de un dispositivo basado en transductor mecánico-eléctrico (en forma de alambre magnético) que es capaz de reconocer la señal eléctrica generada al plasmar una firma autógrafa de una persona, verificando la autenticidad de la misma con base a características propias de la firma y del movimiento de la mano del individuo. Para lograr implementar dicho dispositivo se utilizaron técnicas y algoritmos de reconocimiento de patrones, principios básicos de electrónica analógica, conocimientos de la arquitectura de computadoras y del software para su programación.

ABSTRACT

In this report, we describe a device able to recognize personal signatures. The system is based on a novel transducer in the form of magnetoelastic wire. The recognition process is based on signature´s particularities and on personal characteristic hand movements. Recognition patterns techniques were used together with analogic electronics, computer arquitecture and programming.

1 INTRODUCCIÓN

El reconocimiento de patrones como desarrollo tecnológico ha tenido un gran impulso desde los años 50´s del siglo XX debido a la aparición de las computadoras digitales y a su creciente capacidad para procesar información, lo que ha permitido programar y poner a prueba algoritmos de reconocimiento que por su elevado coste computacional no habían podido ser implementados previamente [1-3]. Algunos de los primeros esfuerzos en el campo del reconocimiento de patrones fueron tentativas de programar computadoras para tomar decisiones automáticas y desarrollar hardware especializado en leer patrones tales como caracteres alfa numéricos [1]. Un ejemplo de estos desarrollos incipientes fue el Perceptrón de Rossenblatt, el cual estaba basado en el algoritmo Perceptrón, que describe la forma de almacenar la información en el cerebro [1]. Aunado al desarrollo del reconocimiento de patrones está el avance en el estudio de la inteligencia artificial, cuyo primer reporte fue realizado por W. McCulloch & W.Pitts en 1943 [1]. Ellos propusieron un modelo construido por neuronas artificiales. Posteriormente, D.Hebb en 1949 demostró una sencilla regla de actualización para modificar las conexiones entre neuronas de manera que ocurriera el aprendizaje [2]. A principios de la década de los 50's C. Shannon & A. Turing escribieron programas de ajedrez para computadoras convencionales mientras que M. Minsky & D. Edmonds construyeron la primera computadora de red neuronal [2]. Estos esfuerzos iniciales en la resolución de problemas se basaban en un mecanismo de búsqueda de propósito general en el que se entrelazaban pasos de razonamiento elementales para encontrar así soluciones completas. El programa DENDRAL construido por E. Feigenbaum, B. Buchanan & J. Lederberg en 1969 [2] constituye un ejemplo típico de éste enfoque. La necesidad de aplicaciones prácticas impulsó el aumento en la demanda de esquemas de representación del conocimiento, para lo cual se desarrollaron varios lenguajes de representación como PROLOG [2], que era una PROgramación LÓGica diseñada para la manipulación de símbolos [3]. En 1981 los japoneses anunciaron el proyecto de la "quinta generación", un plan de 10 años para construir computadoras inteligentes en las que corriera PROLOG[2]. Desde entonces, han sido muchas y muy variadas las aplicaciones que se le han dado al reconocimiento de patrones. La primera máquina lectora de Kurzweil (KRM, 1976) por ejemplo, consiste en una máquina exploradora de imágenes [4]. El reconocimiento auditivo ha recibido también mucha atención ya que los sonidos son fundamentales

XVIII CONGRESO DE INSTRUMENTACIÓN

LA SOCIEDAD MEXICANA DE INSTRUMENTACIÓN OTORGA LA PRESENTE.

MENCIÓN HONORÍFICA

a:

Manuel Ramírez Alvarez , Polo Fco. Padilla Monroy e Israel Betancourt Reyes

Por presentar el mejor trabajo en el área de:

Materiales, Sensores y Películas Delgadas

con el trabajo

Pluma magnetoelástica para reconocimiento de firma autógrafa

Torre de Ingeniería, Cd. Universitaria, México D.F. Octubre 2003

SOMI

Dr. Felipe Lara Rosano
PRESIDENTE

CCADET
CENTRO DE CIENCIAS APLICADAS Y DESARROLLO TECNOLOGICO

SOCIEDAD MEXICANA DE INSTRUMENTACIÓN, A.C.

MEXICAN SOCIETY OF INSTRUMENTATION

SOMI

LA SOCIEDAD MEXICANA DE INSTRUMENTACIÓN

otorga la siguiente

CONSTANCIA

al

RAMÍREZ ALVAREZ MANUEL

Por su participación en la sesión cartel con su trabajo:

PLUMA MAGNETOELÁSTICA PARA RECONOCIMIENTO DE FIRMA AUTÓGRAFA

En el Congreso de Instrumentación, efectuado del
6 al 10 de octubre de 2003,
Torre de Ingeniería, Cd. Universitaria
UNAM, México, D.F.

Dr. Ovsei Gelman Muravchik
Secretario Ejecutivo

Apdo. Postal 70-186, C.P. 04510, Coyoacán, México, D.F. Tel: (52) 5622-8635, Fax: (52) 5622-8620
Email: somi@aleph.cinstrum.unam.mx http://somi.cinstrum.unam.mx

SOCIEDAD MEXICANA DE INSTRUMENTACIÓN, A.C.

MEXICAN SOCIETY OF INSTRUMENTATION

SOMI

LA SOCIEDAD MEXICANA DE INSTRUMENTACIÓN

otorga la siguiente

CONSTANCIA

al

PADILLA MONROY POLO FRANCISCO

Por su participación en la sesión cartel con su trabajo:

PLUMA MAGNETOELÁSTICA PARA RECONOCIMIENTO DE FIRMA AUTÓGRAFA

En el Congreso de Instrumentación, efectuado del
6 al 10 de octubre de 2003,
Torre de Ingeniería, Cd. Universitaria
UNAM, México, D.F.

Dr. Ovsei Gelman Muravchik
Secretario Ejecutivo

CIUDAD UNIVERSITARIA, MÉXICO, D.F.
2003

Apdo. Postal 70-186, C.P. 04510, Coyoacán, México, D.F. Tel: (52) 5622-8635, Fax: (52) 5622-8620
Email: somi@aleph.cinstrum.unam.mx http://somi.cinstrum.unam.mx

INSTITUTO DE INVESTIGACIONES EN MATERIALES
COORDINACIÓN DE FORMACIÓN DE
RECURSOS HUMANOS

VNIVERSIDAD NACIONAL
AVTÞNOMA DE
MEXICO

A quien corresponda:

Por este medio hago constar que el alumno ***MANUEL RAMÍREZ ALVAREZ,*** de la carrera de Ingeniería Eléctrica Electrónica, de la Facultad de Ingeniería de la UNAM, con número de cuenta 95254029, se encuentra realizando su tesis de licenciatura titulada "*Pluma magnetoelástica para reconocimiento de firma autógrafa*" en este instituto, bajo la asesoría del Dr. José Israel Betancourt Reyes, durante el periodo del 01 de abril al 31 de diciembre de 2003.

Atentamente,
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cd. Universitaria, D. F., a 17 septiembre de 2003.

EL COORDINADOR

INSTITUTO DE INVESTIGACIONES
EN MATERIALES

DR. TATSUO AKACHI MIYAZAKI

VNIVERSIDAD NACIONAL
AVTONOMA DE
MEXICO

INSTITUTO DE INVESTIGACIONES EN MATERIALES
COORDINACIÓN DE FORMACIÓN DE
RECURSOS HUMANOS

A quien corresponda:

Por este medio hago constar que el alumno ***POLO FRANCISCO PADILLA MONROY,*** de la carrera de Ingeniería Eléctrica Electrónica, de la Facultad de Ingeniería de la UNAM, con número de cuenta 95096904, se encuentra realizando su tesis de licenciatura titulada "*Pluma magnetoelástica para reconocimiento de firma autógrafa*" en este instituto, bajo la asesoría del Dr. José Israel Betancourt Reyes, durante el periodo del 01 de abril al 31 de diciembre de 2003.

Atentamente,
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cd. Universitaria, D. F., a 17 septiembre de 2003.

EL COORDINADOR

INSTITUTO DE INVESTIGACIONES
EN MATERIALES

DR. TATSUO AKACHI MIYAZAKI

BIBLIOGRAFÍA

1. J. T. Tou & R. C. Gonzales, ”Pattern Recognition Principles”, Addison-Wesley, 1974, 377p.

2. S. J. Russell & P. Norving, “Inteligencia artificial”, Prentice-Hall, 1996, 953p.

3. J. Mochón, R. Aparicio, F Trigueros & C. Castaños, “Inteligencia artificial, evolución histórica y perspectivas de futuro”, Boixareu Editores, 1987, 284p.

4. R. Kurz, “Máquinas inteligentes”, Sirius, 1994, 603p.

5. T. Dean, J. Allen & Y. Aloimonds, “Artificial intelligence”, Benjamin Cummings, 1995, 559p.

6. M. Friedman & A. Kandel, “Introduction to pattern recognition”, World Scientific, 1999, 325p.

7. “Physica B”, vol 299, 2001, 302-313.

8. J. Reitz F. Milford, “Fundamentos de la teoría electromagnética”, Uteha, 1981, 270p.

9. R. Coughlin & F. Driscoll, “Amplificadores operacionales y circuitos integrados lineales”, Prentice-Hall, 1999, 517p.

10. L. R. Rabiner & R. W. Schafer, “Digital Processing of Speech Signals”, Prentice-Hall, 1978, 509p.

11. J. Barrios Romano, “Introducción a los filtros digitales ”, UAM, 1992, 240p.

12. J. G. Proakis & D. G. Manolakis, “Digital Signal Processing”, Macmillan, 1992, 969p.

13. Michael T. Bruno J. ,“PC interno 5, Programación de Sistemas”, Marcombo, 1996, 1373p.

14. Murray Spiegel, “Probabilidad y Estadística”, McGraw-Hill, 1976, 372p.