UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

DISEÑO DE UN RECONOCEDOR DE COMANDOS DE VOZ PARA EL DSP TMS320C6711

T E S I S

QUE PARA OBTENER EL TÍTULO DE

MAESTRO EN INGENIERÍA

AREA: ELÉCTRICA.

CAMPO: PROCESAMIENTO DIGITAL DE SEÑALES.

P R E S E N T A:

OMAR NIETO CRISÓSTOMO

TUTOR: DR. JOSÉ ABEL HERRERA CAMACHO

CIUDAD UNIVERSITARIA MÉXICO, D.F. ABRIL 2006

A mis padres,
***Álvaro Nieto Mondragón* y**
Dominga Crisóstomo Blas
Que me han apoyado incondicionalmente
durante todo mi desarrollo profesional
y por todos sus desvelos ...

Agradecimientos.

A DIOS por haberme dado la oportunidad de ingresar a este mundo maravilloso de las ciencias.

A mi *alma mater* UNAM por haberme formado como persona y haberme guiado por un sendero rico en conocimiento y oportunidades.

A la Facultad de Ingeniería que me ha ofrecido todas sus instalaciones durante toda mi formación.

A mis profesores por haberme regalado parte de su sabiduría y experiencia.

A mi tutor Dr. José Abel Herrera Camacho por haberme dado todo su apoyo, acompañamiento y consejo durante estos años, y por haberme guiado en el camino de la investigación.

A Maria Angélica Villegas Aguilar por acompañarme todos este tiempo.

Agradezco a CONACyT por apoyarme con una beca durante seis meses.

Índice general

Índice de figuras

Índice de tablas

Introducción

El reconocimiento de voz ha alcanzado el punto de realización comercial viable sobre cualquier computadora. Aunque se ha logrado una madurez en los algoritmos de reconocimiento, aun existe la pregunta ¿Cómo la voz puede usarse en aplicaciones para facilitar la comunicación hombre-máquina?. Un ejemplo de ello, es el uso del reconocimiento de voz en la telefonía móvil, en donde cada vez los teléfonos son más pequeños, y ha orillado a tener interfaces de comunicación diferentes a las de un teclado.

La operación de la máquina por medio de la voz, ha sido uno de los sueños que han tenido la mayoría de las personas. Imagínese que se encuentra en su lugar de trabajo y ha terminado su jornada, sería muy agradable decirle a su vehículo: "*llévame a mi casa*", el automóvil obedecerá sus instrucciones automáticamente, sin replica alguna. Esto cada día se está convirtiendo más y más en realidad con los avances tecnológicos, especialmente en el área de la electrónica, telecomunicaciones y la computación. La realización de esta tecnología involucra varias disciplinas como son: el procesamiento digital de voz, control automático, redes inalámbricas, etc. Esta tesis trata de realizar una interface de voz para dictarle comandos a un móvil cualquiera, la parte electrónica y el control del móvil, se encuentra fuera de los alcances de este trabajo, sin embargo, pueden ser la siguiente etapa del proyecto.

Actualmente, los procesadores tienen una enorme velocidad de ejecución, permitiendo realizar sistemas que trabajan prácticamente en tiempo real. El procesamiento digital de señales, es una de las área que se ha beneficiado de estos avances tecnológicos, con la creación de un procesador especializado, también conocidos como DSP (Procesadores Digitales de Señales), se han podido simplificar enormemente el tratamiento de cualquier señal analógica, en particular, la señal de voz. Por tal motivo, uno de los objetivos de esta tesis es el reconocimiento de comandos de voz en tiempo real, por medio de un procesador digital de señales (DSP), para el control de un móvil cualquiera.

El propósito principal de esta tesis es, efectuar el reconocimiento de ocho palabras aisladas, en tiempo real, por medio del DSP TMS320C6711, empleando

las siguientes técnicas: análisis cepstral con escala en frecuencia Mel; cuantización vectorial (con *K-medias*) y la distorsión euclidiana cuadrática; modelos ocultos de Markov (HMM) para palabras aisladas, utilizando densidades gaussianas (una por estado) y modelos acústicos, representativos a toda la palabra, estos últimos mediante concatenación de modelos de tres estados que representan a los fonema en la palabra.

Las pruebas fueron hechas para las técnicas: cuantización vectorial (VQ) y distancia euclidiana cuadrática; VQ y distancia euclidiana; HMM y probabilidad logarítmica. Se pronunciaron cincuenta repeticiones por cada comando. Los resultados obtenidos fueron los siguientes: para VQ y distancia euclidiana cuadrática, se alcanzó el 100 % de reconocimiento, de los cuales, el 99 % se reconoció bien y el 1 % con un poco de dificultad; para el caso de VQ y distancia euclidiana, se logró el 99.75 % de reconocimiento con un 99.25 % de reconocimiento satisfactorio y 0.5 % de reconocimiento con dificultad, el 0.25 % restante, el sistema lo etiquetó como palabra desconocida; finalmente, el caso del reconocimiento con HMM se alcanzó el 100 % de reconocimiento, de los cuales, el 99.75 % se reconoció bien y el 0.25 % con dificultad.

Los problemas en el reconocimiento de voz

Uno de los aspectos más difíciles de la investigación en el reconocimiento de la voz, es su naturaleza interdisciplinaria. De esta forma, se necesita del procesamiento de señales, la acústica, el reconocimiento de patrones, la lingüística, la fisiología, la teoría de la información, etc., para la realización de sistemas exitosos.

La dificultad en el reconocimiento de la voz cae principalmente en los siguientes factores [Deller, Proakis y Hansen 87]:

- *Dependencia del locutor.* El sistema reconoce a un solo locutor (dependiente del locutor) o a múltiples locutores (independiente del locutor).
- *El tamaño del vocabulario.* Puede ser vocabulario pequeño (1 a 99 palabras), vocabulario mediano (100 a 999 palabras) ó vocabulario grande (1000 palabras ó más).
- *Forma de la pronunciación.* Se pronuncia la voz como unidades discretas, principalmente palabras con distintas pausas entre ellas (reconocimiento de palabras aisladas), o como una pronunciación continua (reconocimiento de palabras conectadas o reconocimiento de continuo).
- El grado de la ambigüedad (por ejemplo: “caza” “casa”) y confusiones acústicas (por ejemplo: “casa”, “masa”, “gasa”).

- *La naturaleza del ruido del entorno*: ambientes controlados o entornos con ruido.

- *Las restricciones lingüísticas*: ¿Cómo se pueden concatenar las unidades fundamentales (fonemas, sílabas o palabras)? ¿En qué orden? ¿En qué contexto? y ¿En qué significado?.

Las aplicaciones de los sistemas de reconocimiento de voz, dependen en gran medida de estos factores, por ejemplo: un sistema de dictado de palabras, necesita un vocabulario muy grande (alrededor de 30,000 palabras), también debe se independiente del locutor y por lo regular un entorno controlado contra el ruido; por otro lado, un sistema de control de aplicaciones, necesita un vocabulario pequeño (alrededor de 10 a 100) comandos, si es necesario, ser independiente del locutor y debe funcionar en condiciones de mucho ruido [Becchetti Ricotti 99]. La figura 1 muestra los problemas de un reconocedor de voz y sus aplicaciones, mediante cuatro ejes: el tamaño del vocabulario, la calidad de la señal de voz, la variabilidad del locutor y la capacidad de cómputo necesaria.

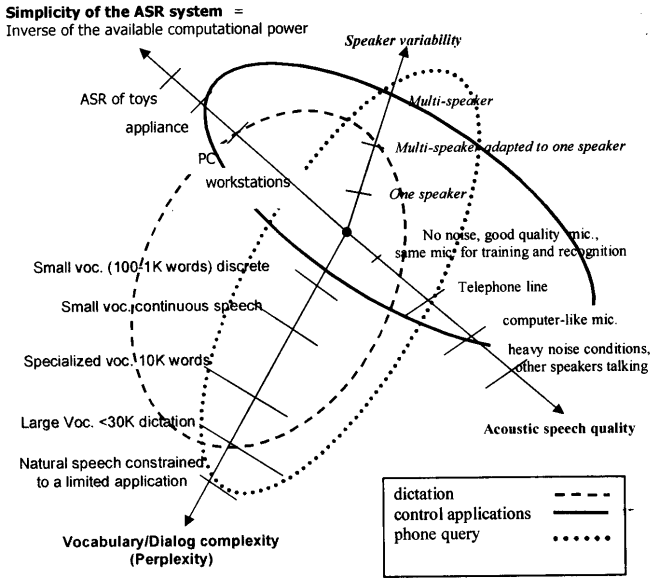


Figura 1: Problemas del reconocimiento de voz y sus aplicaciones.

Componentes de un reconocedor de voz

Para discutir el problema del diseño de un reconocedor de voz, es necesario describirlo matemáticamente. Se denotará como $\mathbf{O}$, a las secuencia de vectores representativos de las pronunciaciones de palabras, y se llamará observaciones acústicas. Debido a que se tiene que manejar en computadoras, se puede suponer sin pérdida de generalidad, que $\mathbf{O}$ es una secuencia de símbolos tomadas de un mismo (y posiblemente muy grande) alfabeto $\mathcal{O}$ [Jelinek 97]:

$$\mathbf{O} = o_1, o_2, \cdots, o_T, \qquad o_t \in \mathcal{O} \tag{1}$$

los símbolos o_t son generados a lo largo del tiempo t.

Sea también, $\mathbf{W}$ como la secuencia de n palabras de un vocabulario fijo y conocido $\mathcal{V}$:

$$\mathbf{W} = w_1, w_2, \cdots, w_n, \qquad w_i \in \mathcal{V} \tag{2}$$

Por tanto, el reconocimiento de voz estima una secuencia de palabras $\hat{\mathbf{W}}$ que es generada por una secuencia de observaciones acústicas $\mathbf{O}$. Usando la probabilidad a posteriori $P(\mathbf{W}|\mathbf{O})$, el reconocimiento de voz puede formularse por la siguiente ecuación

$$P(\hat{\mathbf{W}}|\mathbf{O}) = \max_{\mathbf{W}} P(\mathbf{W}|\mathbf{O}) \tag{3}$$

usando la regla de Bayes, la parte derecha de la ecuación 3 puede escribirse como:

$$P(\mathbf{W}|\mathbf{O}) = \frac{P(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{O})} \tag{4}$$

Debido a que $P(\mathbf{O})$ es independiente de $\mathbf{W}$, en la ecuación 4, entonces la secuencia de palabras estimada, se puede obtener de la siguiente manera:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{O}|\mathbf{W})P(\mathbf{W}), \tag{5}$$

donde $P(\mathbf{O}|\mathbf{W})$ es la probabilidad de una secuencia de observaciones acústicas, condicionada a una secuencia de palabras dada, conocida como la *probabilidad del modelo acústico*. $P(\mathbf{W})$ es la probabilidad de una secuencia de palabras, incluyendo relaciones entre ellas; se le conoce como la *probabilidad de un modelo del lenguaje*. Para esta tesis, solo se considera la probabilidad del modelo acústico y por tanto, no se toma en cuenta la probabilidad del modelado del lenguaje, es decir, la probabilidad $P(\mathbf{W})$ es la misma para todas las palabras.

Descripción general de la tesis

El sistema de reconocimiento de palabras aisladas que se ha implementado con DSP, es la continuación del sistema elaborado en la tesis de [Nieto, López 02]; donde, se implementa un reconocedor de comando de voz en tiempo real, mediante la tarjeta de desarrollo *Starter Kit TMS320C6711* de *Texas Instruments*, usando Cuantización Vectorial (VQ) y vectores de predicción lineal (LPCs). La tesis descrita en este documento, mejora la técnica de reconocimiento de comandos de voz, utiliza vectores Mel Cepstral y las técnicas de entrenamiento VQ y HMM, bajo la misma tarjeta de desarrollo.

El Capítulo 1 trata el proceso de la producción y percepción de la voz, la clasificación de los sonidos de la voz y muestra un modelo del tracto vocal. Por otro lado, el Capítulo 2, describe los fundamentos de procesamiento digital de voz empleados a lo largo de la tesis, como son: características de la señal de voz, análisis de la señal de voz en el dominio de la frecuencia, y la descripción de un algoritmo de detección del inicio y el final de palabras aisladas.

El Capítulo 3 se refiere a la obtención de las características de la señal de voz, mediante el análisis cepstral, en frecuencia con escala Mel. El Capítulo 4 describe la clasificación de patrones utilizados para el entrenamiento: VQ y HMM; en el primero se describe un método de agrupamiento llamado *k-medias*; en el segundo, HMM, se explica los algoritmos para solucionar los tres problemas básicos en los modelos ocultos de Markov, el uso de densidades continuas, el problema de precisión, así como la forma de incluir múltiples observaciones. El Capítulo 5 menciona algunas unidades de reconocimiento y sus modelos HMM.

En el Capítulo 6 se realiza una descripción general de la tarjeta de desarrollo *Starter Kit TMS320C6711* (C6711), el CPU, los periféricos y las interrupciones que maneja; también, se describe las herramientas de desarrollo: Code Composer Studio, DSP/BIOS, etc.

La etapa de entrenamiento del sistema, se explica en el Capítulo 7, en donde se muestra la serie de pasos realizados en la captura de las señales de voz por el DSK C6711 y los algoritmos elaborados para el entrenamiento con VQ y HMM. El Capítulo 8 describe los algoritmos desarrollados para el reconocedor de comandos de voz en tiempo real, en ambas técnicas de entrenamiento. Finalmente en el Capítulo 9, muestra los porcentajes obtenidos de reconocimiento, los tiempos aproximados para los algoritmos más importantes y la memoria utilizada por el sistema; todo esto para las dos técnicas de entrenamiento VQ y HMM; también se describen algunas mejoras que se pueden hacer al sistema, así como las posibles aplicaciones.

Capítulo 1

El proceso de producción y percepción de voz

El proceso de producción (generación) de voz, comienza cuando una persona (el hablante) formula un mensaje en la mente, y este lo desea transmitir por medio de la voz, a un oyente. El siguiente paso en el proceso, es la conversión del mensaje en código de un lenguaje. Una vez que el código del lenguaje es elegido, el hablante debe ejecutar una serie de comandos neuro-musculares para causar que las cuerdas vocales vibren cuando sea apropiado y que el tracto vocal tome la forma adecuada para emitir los sonidos de voz correctos. Los comandos neuro-musculares deben, simultáneamente, controlar todos los aspectos del movimiento articulatorio, incluyendo: controlar los labios, la mandíbula, la lengua y el velo del paladar [Rabiner 93].

Una vez que la señal de voz es generada y propagada hacia el oyente, el proceso de percepción de voz (reconocimiento de voz) comienza. Primero el oyente procesa la señal acústica a través de la membrana basilar en el oído interno, que proporciona un análisis del espectro de la señal de entrada. Un proceso de transducción neuronal, convierte la señal de salida de la membrana basilar en señales activas en el nervio auditivo, correspondiendo aproximadamente a un proceso de extracción de características. En una manera que aún no se ha comprendido, la actividad neuronal a través del nervio auditivo es convertida en un código de lenguaje en los centros mas altos del procesamiento dentro del cerebro, y finalmente es entendido el mensaje (comprensión del significado) [Rabiner 93].

1.1. Aparato fonador

En la producción de la voz intervienen un conjunto de órganos que se conocen con el nombre de *aparato fonador*. Estos se pueden clasificar en tres grupos: Órgano respiratorio o cavidades infraglóticas; Órgano fonador o cavidad laríngea; Cavidades supraglóticas [Quilis 99].

1.1.1. Cavidades infraglóticas

Están formadas por los siguientes órganos: *pulmones*, *bronquios*, *traquea*. Los pulmones son los encargados de la respiración, además, proveer la cantidad de aire suficiente para que se realice el acto de la fonación.

El aire contenido en los pulmones pasa por los bronquios, y de allí a la traquea, órgano formado por anillos cartilaginosos superpuestos, que desemboca a la laringe.

1.1.2. Cavidad laríngea u órgano fonador

La *laringe* (figura 1.1) esta situada por encima de la tráquea, se compone de nueve cartílagos. Tres de ellos son únicos y tres son pares. Los tres cartílagos simples son: el cartílago *tiroides*, el cartílago *epiglótico* (*epiglotis*) y el cartílago *cricoides*. De los cartílagos pares, el cartílago aritenoides es el más importante. Los cartílagos pares *corniculado* y *cuneiforme* son menos importantes [Tortora Anagnostakos 99].

El cartílago tiroides (también llamado *manzana de Adán*) esta compuesto por dos láminas fusionadas, cuya forma se asemeja a escudo.

La epiglotis es un cartílago grande en forma de hoja, se encuentra en la parte superior de la laringe. El "tallo" de la epiglotis se une al cartílago tiroides, la "hoja" o porción libre no se encuentra unida a ninguna estructura, dando lugar a que se pueda mover como una puerta. Durante la deglución, el extremo libre de la epiglotis cierra la glotis. La glotis (figura 1.2) está formada por los pliegues vocales (cuerdas vocales verdaderas) de la laringe y el espacio entre ellos (hendidura glótica).

El cartílago cricoides es un anillo, que forma la pared inferior de la laringe. Se une al primer anillo del cartílago de la tráquea.

Los cartílagos aritenoides son pares y tiene forma piramidal, se sitúan en el

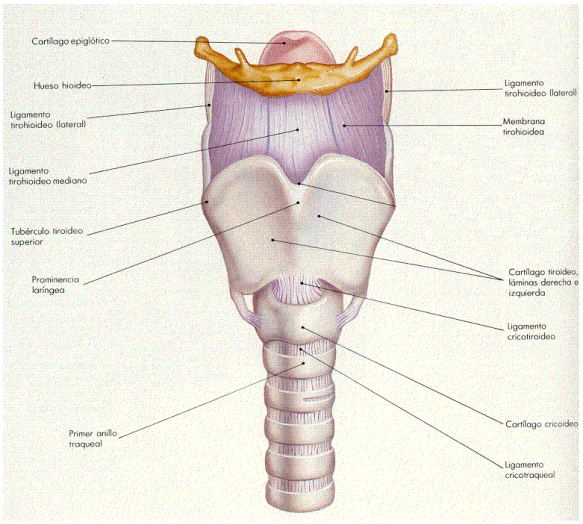


Figura 1.1: Vista anterior de la laringe

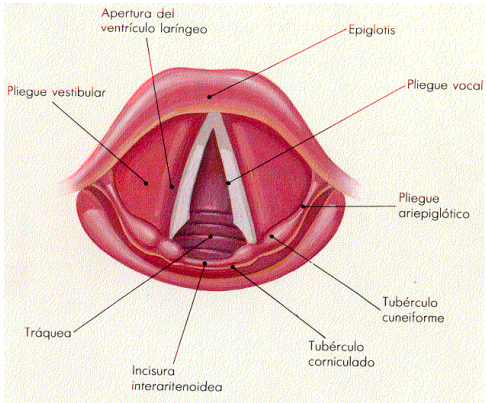


Figura 1.2: Representación de la glotis

borde superior y porción posterior del cartílago cricoides. Se une a los pliegues vocales y a los músculos faríngeos intrínsecos y por su acción pueden mover las cuerdas vocales.

1.1.3. Cavidades supraglóticas

Las principales cavidades supraglóticas son: *la faringe*, *la cavidad oral* y *la cavidad nasal*. Ver figura 1.3.

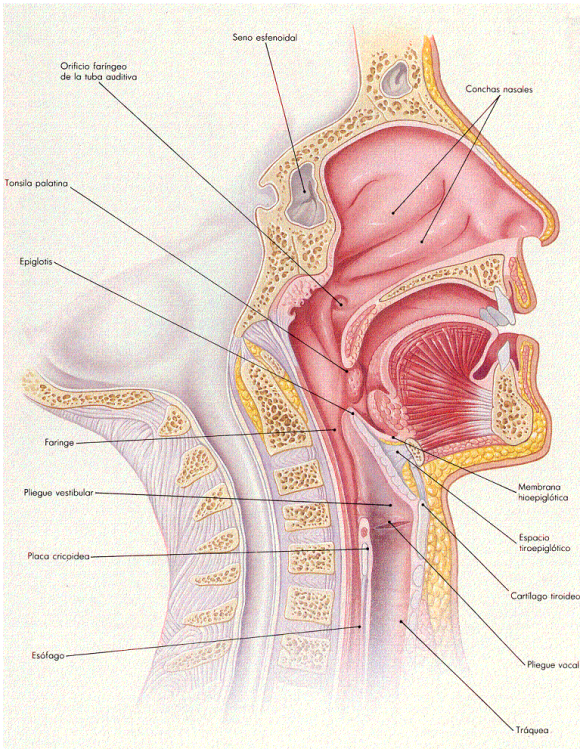


Figura 1.3: Corte sagital de cabeza y cuello que muestra la configuración interior de la laringe, cuerdas vocales, cavidad nasal y cavidad oral.

La faringe o garganta, es un tubo con forma de embudo, conecta la nariz y la boca con la traquea y el esófago. Las funciones de la laringe son, servir de vía de paso al aire y los alimentos, además, proporciona un canal de resonancia para la producción de los sonidos del habla.

La parte superior de la cavidad oral o bucal, está formada por el paladar duro y el paladar blando (velo del paladar). En la sección inferior de la boca se encuentra, la lengua, que constituye una parte importante en la formación de

la voz. La parte anterior de la cavidad oral la conforman los dientes (incisivos superiores e inferiores), los alvéolos dentarios (cavidades existentes en el maxilar y la mandíbula, donde encajan las raíces de las piezas dentarias) y los labios (superior e inferior), que también son importantes en la formación de palabras.

La cavidad nasal se encuentra desde la nariz hasta las aberturas posteriores o *coanas*. Está dividida en dos cavidades por el tabique nasal. La pared lateral de la cavidad nasal contiene tres conchas o cornetes, que son proyecciones óseas cubiertas por mucosa. Abajo y lateral a cada una de estas conchas óseas, se encuentra un espacio llamado meato (superior, medio e inferior). Los senos paranasales son cavidades en los huesos: maxilar, etmoides, esfenoides y frontal; contienen aire y la mayoría desembocan en el meato medio o superior; se piensa que sirven como medio para disminuir el peso de la cabeza ósea y dar resonancia a la voz [Crafts 91].

1.2. Producción de voz

El sonido de la voz es producido cuando pasa una corriente de aire (generada por los pulmones) a través de la cavidad laríngea, e independientemente de la "vibración" o no de las cuerdas vocales[1], se produce una perturbación del aire, en las cavidades supraglóticas, que conllevan a la generación de ondas sonoras.

Los diversos sonidos articulatorios son generados, por las modificaciones en la forma y el volumen de la cavidad bucal, cavidad nasal, así como, en la vibración o no de las cuerdas vocales. Su clasificación se basa principalmente en estos puntos, y se describe en la siguiente sección.

1.3. Clasificación de los sonidos de voz

Se puede clasificar los sonidos de la voz, dependiendo de sus rasgos articulatorios de la siguiente forma [Quilis 99], [Alcina Blecua 01]:

- Por la acción de las cuerdas vocales
- Por la acción del velo del paladar
- Por el modo de articulación

[1] Estrictamente hablando, no es que las cuerdas (o pliegues) vocales vibren, sino que la membrana mucosa que las recubre, realiza un movimiento ondulatorio de abajo hacia arriba, debido a la presión del aire infraglótico [Perello Peres 77].

- Por el lugar o punto de articulación

1.3.1. Por la acción de las cuerdas vocales

Según la vibración de los pliegues vocales, los sonidos de la voz se pueden clasificar en: *sonoros* y *sordos*. Los sonidos sonoros son producidos cuando las cuerdas vocales vibran, por ejemplo, las vocales [i], [o], [a], [e], etc., y algunas consonantes [b], [d], [g], [l], [m], [n], etc. Los sonidos sordos (o no sonoros), son creados sin la vibración de los pliegues vocales; por ejemplo, las consonantes [p], [t], [k], [s], [f], etc.

1.3.2. Por la acción del velo del paladar

Cuando el velo del paladar se encuentra adherido a la pared faríngea, el aire pasa solamente por la cavidad bucal, produciendo los sonidos llamados *orales*, un ejemplo de ello son: [e], [u], [a], [s], [p], [b], etc.

Los sonidos *nasales* se producen cuando el velo del paladar está separado de la cavidad faríngea y la cavidad bucal se encuentra cerrada, de esta forma, sólo el conducto nasal actúa como único resonador, por ejemplo, los sonidos [m], [n]. En otro caso, cuando el conducto oral y el nasal se encuentran abiertos, se le conoce como sonidos *oronasales* (o *vocales nasales*), la vocal [ẽ] en la palabra [ũmãnãmẽnte] *humanamente* actúa de esta forma.

1.3.3. Por el modo de articulación

El modo de articulación es la modificación realizada al flujo de aire, debido a la abertura o cierre de los órganos articulatorios, e independientemente del canal en que se aproximan o entren en contacto. Según esta característica, se pueden clasificar los sonidos de las vocales y consonantes de la siguiente forma:

Vocales

- *Altas*: son generados cuando la lengua se aproxima demasiado al paladar duro o al paladar blando, por ejemplo, [i], [u].
- *Medias*: son sonidos producidos al posicionar la lengua en la parte intermedia de la cavidad bucal, por ejemplo, [e], [o].
- *Bajas*: cuando se sitúa la lengua en la parte mas baja de la bóveda palatal, por ejemplo, [a].

Consonantes

- *Oclusivos*: es cuando se cierran totalmente los órganos articulatorios y se abren en una breve explosión. También se les conoce como sonidos *explosivos* o *plosivos*. Por ejemplo, [p], [b], [t], [d], [k], y [g].
- *Fricativos*: son generados cuando los órganos articulatorios se aproximan demasiado sin que se origine el cierre característico de los sonidos oclusivos. [ƀ], [đ], [θ] y [s].
- *Africados*: también llamados *semioclusivos*, son producidos por un movimiento oclusivo y suavemente se convierte en fricativo. El sonido [ĉ], que corresponde a la letra *ch*, es un ejemplo.
- *Laterales*: el aire pasa por los lados de la lengua, mientras que el ápice de la lengua hace contacto con la parte superior de la cavidad bucal, por ejemplo, [l] en [l]ámina.
- *Vibrantes*: el ápice de la lengua realiza un movimiento de vibración contra los alvéolos dentales. Estos sonidos son generados por [r] y [r̄], por ejemplo, pe[r]o (pero) y pe[r̄]o (perro).
- *Semiconsonantes*: estos sonidos comienzan con una estrechez típica de las consonantes y terminan con la amplitud característica de las vocales. [j̰], [w]: pend[j̰]ente, n[w]evo (pendiente, nuevo).
- *Semivocales*: estos sonidos son lo contrario a las semiconsonantes, comienzan con un sonido vocálico y terminan con un sonido estrecho: [i̯],[u̯]: pe[i̯]ne, ta[u̯]maturgo.

1.3.4. Por el lugar de articulación

El *punto o lugar de articulación* es la zona donde se aproximan o se juntan los órganos articulatorios para producir un estrechamiento o cierre del conducto bucal. La clasificación de los sonidos según el punto de articulación es:

Vocales

- *Anteriores*: también se conocen como *palatales*, son producidos cuando la lengua ocupa la región anterior de la cavidad bucal (zona cubierta por el paladar duro). [i], [e].
- *Centrales*: son producidos cuando la lengua se aproxima a la región posterior de la cavidad bucal, es decir al velo del paladar. También se les conoce como *velares*, un ejemplo son: [u], [o].
- *Posteriores*: cuando la lengua se encuentra en la zona del paladar medio, por ejemplo: [a].

Consonantes

- *Bilabiales*: es realizado cuando se cierran los labios momentáneamente, impidiendo la salida del aire de la cavidad bucal. Un ejemplo de ellos son los sonidos: [p] que corresponde al fonema /p/; [b] y [ɓ] correspondientes al fonema /b/ y [m] que concierne al fonema /m/.
- *Labiodentales*: estos sonidos son producidos con el labio inferior apoyado sobre los incisivos superiores. El sonido [f] que corresponde al fonema /f/, es un ejemplo de este tipo de articulación.
- *Interdentales*: son realizados al posicionar el ápice de la lengua entre los incisivos. Por ejemplo, [θ] del fonema /θ/[2]
- *Dentales*: la articulación se realiza con el ápice de la lengua contra los incisivos superiores. El sonido [t] del fonema /t/ y los sonidos [d], [ɗ] del fonema /d/, son ejemplos de esta articulación.
- *Alveolares*: se realiza la articulación entre el ápice de la lengua y los alvéolos dentales. Por ejemplo: /s/, /n/, /l/, /r/ y /r̄/.
- *Palatales*: la región predorsal de la lengua se adhiere a la zona prepalatal. Los fonemas /ĉ/, /y/, /n̪/ y /l̪/, realizan esta articulación.
- *Velares*: se realizan con el postdorso de la lengua contra el velo del paladar. Los fonemas /k/, /g/ y /x/ realizan esta articulación.

Las tablas 1.1 y 1.2 muestran un resumen de la clasificación de los sonidos con respecto al modo y lugar de articulación para las vocales y consonantes, respectivamente.

Tabla 1.1: Clasificación de las vocales, según el modo y el punto de articulación

	Anteriores o palatales	Central	Posteriores o velares
Altas	/i/		/u/
Medias	/e/		/o/
Bajas		/a/	

1.4. Sistema auditivo

El oído se puede dividir en tres secciones: *oído externo*, *oído medio* y *oído interno*. La figura 1.4 muestra las partes anatómicas más representativas del aparato auditivo [Quilis 99], [Tortora Anagnostakos 99], [Crafts 91], [Fuentes De Lara 97].

[2]Este sonido es típico de España para pronunciar la grafía corresponde a la consonante *c* delante de las vocales *e*, *i* y también a la grafía *z* ante las vocales *a*, *o*, *u*.

Tabla 1.2: Clasificación de las consonantes, según el modo y el punto de articulación

		Labiales	Labiodentales	Interdentales	Dentales	Alveolares	Palatales	Velares
Oclusivas	Sordas	/p/			/t/			/k/
	Sonoras	/b/			/d/			/g/
Africadas	Sordas				/t/		/ĉ/	
Fricativas	Sordas		/f/	/θ/		/s/		/x/
	Sonoras						/y/	
Nasales	Sonoras	/m/				/n/	/ŋ/	

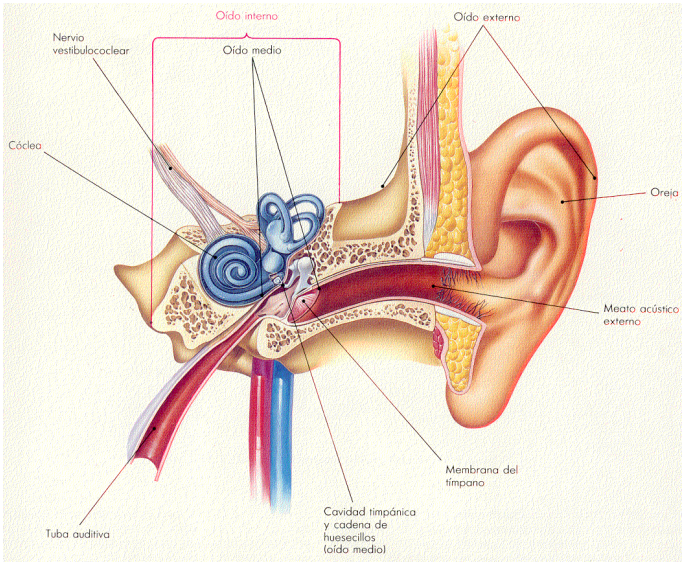


Figura 1.4: Corte transversal del oído

1.4.1. Oído externo

Esta formado por el *pabellón auricular* y *meato* o *conducto auditivo externo*. El pabellón recoge las ondas sonoras y las dirige hacia el conducto auditivo, la unión de los dos pabellones ayuda a identificar el lugar y la distancia de la fuente sonora. El conducto auditivo mide alrededor de 25 mm de largo y 2 mm de diámetro; además de mantener uniforme la temperatura y humedad del aire, actúa como un canal de resonancia[3] para las frecuencias entre 2,500 Hz y 4,000 Hz aproximadamente [Quilis 99].

1.4.2. Oído medio

También se le conoce como *cavidad timpánica*, es una cavidad pequeña revestida de epitelio y llena de aire, se ubica en el hueso temporal (figura 1.5). Está constituida por el *tímpano*, los *huesillos* del oído medio (*martillo*, *yunque* y *estribo*, figura 1.6) y la *trompa de Eustaquio*. Sus funciones principales son: la transmisión de las vibraciones sonoras, la amplificación de las mismas y la protección del oído interno de sonidos muy fuertes.

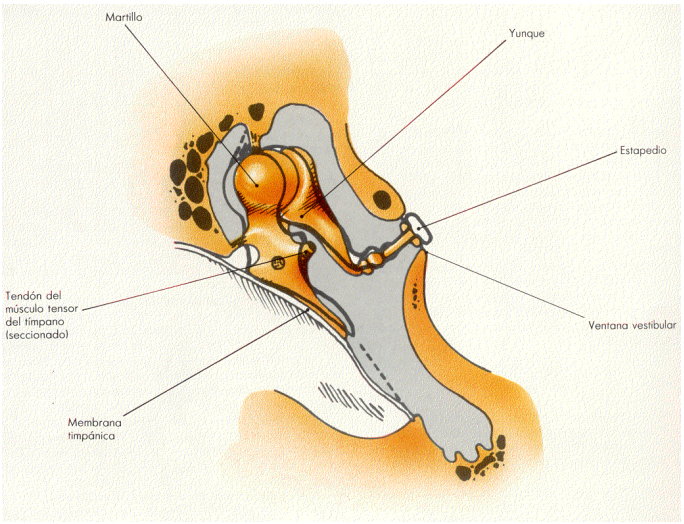


Figura 1.5: Cavidad timpánica

El tímpano es una membrana delgada, elástica y en forma de cono. Esta situado al final del conducto auditivo externo. Cuando las ondas sonoras llegan a través del meato, estas se convierten en ondas mecánicas por la vibración de la membrana timpánica.

[3]El aumento de la presión de las ondas sonoras es alrededor de 10 dB [Perello Peres 77]

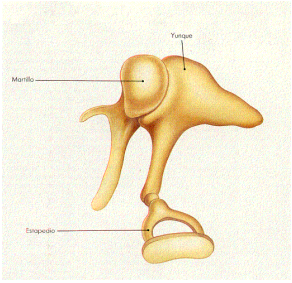


Figura 1.6: Conjunto de huesillos del oído medio

La superficie interna del tímpano esta conectada al primero de los huesillos, el martillo. La cabeza de este último, se articula al cuerpo del yunque (segundo huesillo), y por medio de la *apófisis lenticular*, se enlaza con la cabeza del estribo (tercer huesillo). La base del estribo actúa como tapa de la ventana vestibular, unido en los bordes, por un fino ligamento.

La cadena de huesillos, actúa como un conjunto de palancas para transformar, un movimiento de gran dimensión (vibraciones aéreas), a un movimiento de poca dimensión y gran fuerza (vibraciones líquidas); aumenta catorce veces la presión que llega a la ventana oval, con relación a la presión de las ondas que llegan al tímpano [Quilis 99].

La cavidad timpánica está cerrada por los huesos del cráneo, pero abierta hacia la faringe por medio de la *trompa de Eustaquio*. Sirve para igualar las presiones de aire en el exterior y en el oído medio.

1.4.3. Oído interno

También se le conoce como *laberinto*, por su complicada estructura. Se compone del *laberinto óseo* y el *laberinto membranoso* (figura 1.7). El primero, es un conjunto de cavidades óseas que alojan en su interior al *laberinto membranoso*. Se divide en tres áreas: *vestíbulo*, *cóclea* y tres *conductos semicirculares*. El laberinto membranoso, se encuentra rodeado de un líquido llamado perilinfa, está delimitado con epitelio y contiene un líquido llamado endolinfa.

El vestíbulo consiste de dos sacos llamados: *utrículo*, en donde desembocan los conductos semicirculares membranosos; y el *sáculo*, que se comunica con el primero mediante el conducto llamado utriculosacular, también existe un conducto de unión muy pequeño y delgado hacia la cóclea (ver figura 1.8).

Los tres conductos semicirculares forman cada uno, dos tercios de circulo y

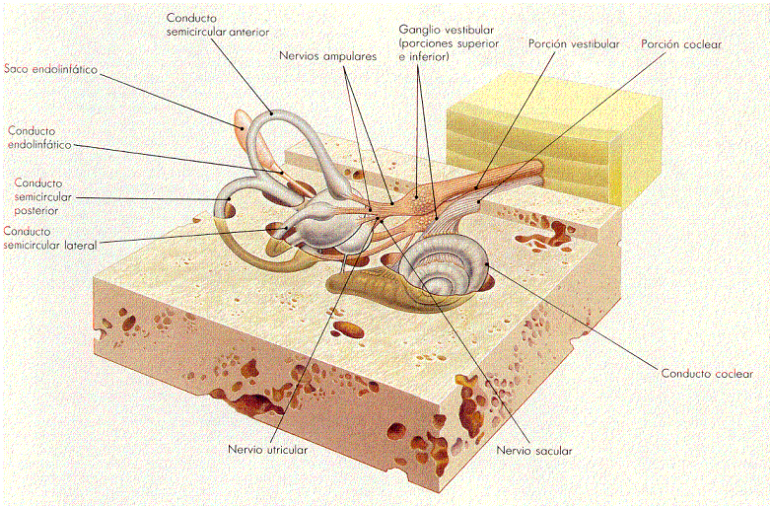


Figura 1.7: Representación del laberinto óseo y laberinto membranosos por medio de un molde plástico.

están situados en planos perpendiculares. Basados en su posición se denominan conductos anterior, posterior y lateral. Contienen *ámpulas* en sus extremos que se comunican con el utrículo. Las ámpulas junto con el sáculo y el utrículo, contienen células sensoriales que detectan el movimiento y hacen que se mantenga el equilibrio (ver figura 1.8).

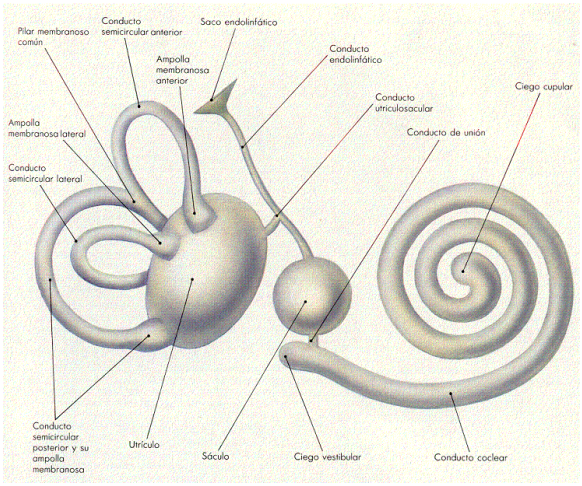


Figura 1.8: Laberinto membranosos.

La cóclea (ver figuras 1.7 y 1.8), también llamada caracol, tiene forma helicoi-

dal con $2\frac{3}{4}$ vueltas al rededor del hueso *mediolo*. Está dividida en tres conductos por medio de la *lámina espiral ósea*, la *membrana basilar* y *membrana vestibular* (membrana de Reissner). El conducto que tiene como extremos la lámina espiral ósea y por la membrana vestibular se le conoce como *rampa vestibular*, está comunicada con el oído medio a través de la ventana oval o vestibular; debajo de ella se encuentra el *conducto coclear* situado entre la membrana basilar y la membrana vestibular; el tercer conducto, *rampa timpánica*, tiene como extremos la membrana basilar y la lámina espiral ósea, termina en la ventana redonda o coclear. La figura 1.9 muestra a la cóclea hipotéticamente extendida.

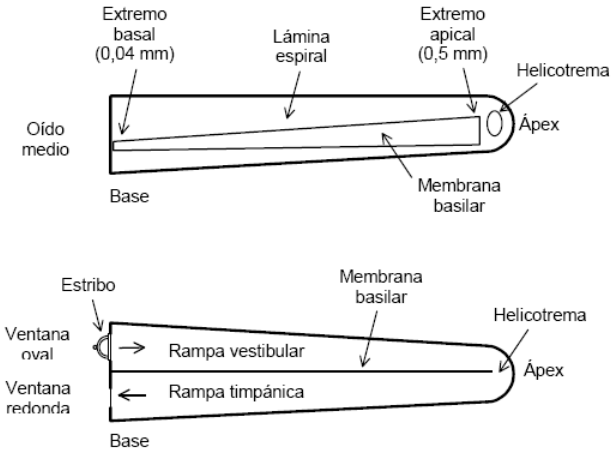


Figura 1.9: Dos vistas de la cóclea hipotéticamente rectificada, arriba vista superior, abajo vista lateral

Sobre la membrana basilar se encuentra el *órgano espiral* u *órgano de Corti* (figura 1.10), que contiene una fila de células pilosas y suspendida sobre ellas la *membrana tectoria*. Algunos autores piensan que, cuando las células pilosas tocan a la membrana tectoria, se produce un impulso nervioso, que es llevado hacia el cerebro e interpretado como un sonido [Crafts 91].

Cuando las ondas sonoras pasan por el pabellón, son dirigidas al tímpano por medio del conducto auditivo externo. Debido a la presión ejercida por las ondas, la membrana timpánica comienza a vibrar, transfiriendo el movimiento al conjunto de huesillos del oído medio. Las oscilaciones son llevadas a la base del huesillo estribo, originando un desplazamiento ondulatorio en la ventana oval y por tanto, en la perilinfa de la rampa vestibular. Esta presión empuja a la membrana vestibular y es transmitida a la membrana basilar por medio del conducto coclear, que a su vez, empuja a la perilinfa en la rampa timpánica, produciendo que la ventana redonda (coclear) sobresalga en el oído medio. Cuando la membrana basilar vibra, las células pilosas del órgano espiral se mueven contra la membrana tectorial, generando impulsos nerviosos que interpretados por

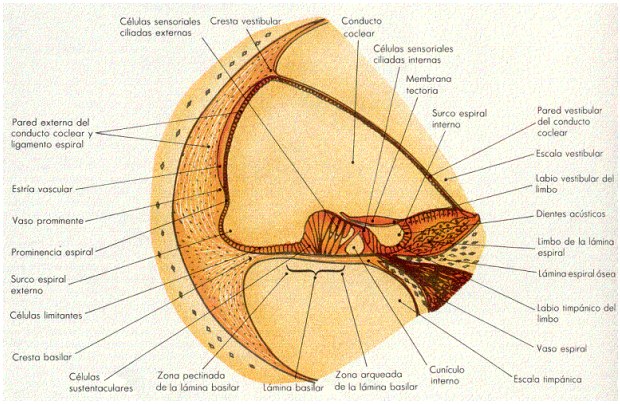


Figura 1.10: Corte transversal del conducto coclear donde se observa el órgano de Corti.

el cerebro, originan el sonido [Tortora Anagnostakos 99].

1.5. Modelo del tracto vocal

El tracto vocal puede ser modelado a partir de un sistema lineal variante en el tiempo, que contiene las propiedades deseadas de la voz, en la salida. Si se considera que para muchos sonidos de voz, las propiedades generales del tracto vocal y su excitación, permanecen fijas en periodos cortos del orden de 10 a 20 ms, entonces, el modelo puede ser representado por un sistema lineal que varia en el tiempo muy lentamente, excitado por una señal cuya naturaleza básica cambia: para sonidos sonoros, por pulsos cuasi-periódicos; y para sonidos sordos, ruido aleatorio [Rabiner Schafer 78]. La figura 1.11 muestra el modelo del tracto vocal propuesto por Rabiner y Schafer.

El modelo de la figura 1.11 trabaja muy bien para sonidos sonoros, y no tan bien para sonidos fricativos u oclusivos. Otras limitaciones del modelo son: la carencia de ceros, trascendental en los sonidos nasales. La división de sonoros y no sonoros es inadecuada para sonidos fricativos sonoros, debido a que la fricación está correlacionada con los picos del flujo glotal. Hay que resaltar que las funciones de transferencia del modelo, son procesadas en tiempo corto; es decir, los parámetros del modelo son considerados constantes, en el intervalo de tiempo alrededor de 19-20 ms.

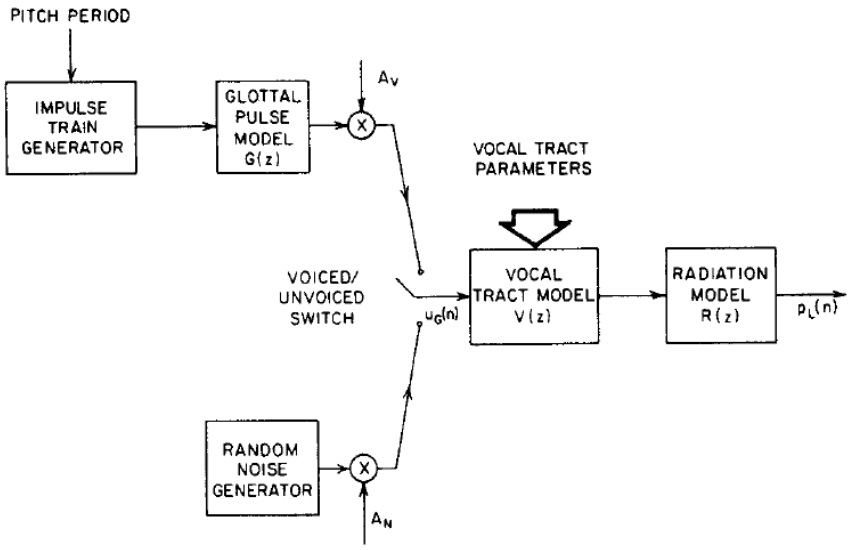


Figura 1.11: Modelo de la producción de la voz.

Capítulo 2

Fundamentos de procesamiento digital de voz

2.1. Características de la señal de voz

Los sonidos se pueden clasificar, de forma genérica, en sonoros y no sonoros o sordos. En los primeros se abren y cierran las cuerdas vocales, modificando el área de la tráquea y produciendo un tren de impulsos casi periódicos. El período o frecuencia fundamental de este tren de impulsos se conoce con el nombre de pitch y su valor está comprendido entre los 50 y 400 Hz para los hombres y es superior en mujeres y niños. En los sonidos no sonoros, el aire fluye libremente hasta alcanzar el tracto vocal, al permanecer abiertas las cuerdas vocales. Finalmente, la variación voluntaria del tracto vocal, junto con el estado variante de las cuerdas, produce la voz.

El tracto vocal actúa como una cavidad resonante para los sonidos sonoros, estando centradas las frecuencias de resonancia para la mayoría de la gente en 500 Hz y sus armónicas pares. Esta resonancia produce grandes picos en el espectro resultante, a los cuales se les llama formantes. Además la señal tiene la naturaleza de un filtro paso bajas y a partir de unos 4000 Hz comienza a predominar el ruido.

En cambio, el segmento de voz no sonoro presenta una estructura ruidosa tanto en el dominio del tiempo como en el de la frecuencia, no teniéndose for-

mantes. Además la energía de la señal es mucho menor que la de los sonidos sonoros [López, 1999].

2.2. Representación de la voz en el dominio del tiempo y la frecuencia

La señal de voz varia lentamente en periodos cortos de tiempo (entre 5 y 100 ms), sus propiedades estadísticas se comportan como una señal estacionaria; sin embargo, sobre periodos largos de tiempo (en el orden de 1/5 segundos o más) las características de la señal cambian para reflejar diferentes sonidos del habla [Rabiner 93].

Existen varias formas de clasificar (etiquetar) los eventos en la voz. El más simple y mas directo es la convención de una representación en tres estados, para la generación de producciones de voz, en el tiempo: (1) *silencios*, cuando no se produce o genera voz, (2) sonidos *no sonoros* o *sordos* y (3) sonidos *sonoros*. Sin embargo, está clasificación no genera segmentos con regiones bien definidas, por lo que representa una dificultad cuando se quieren distinguir regiones dentro de una palabra (figura 2.1).

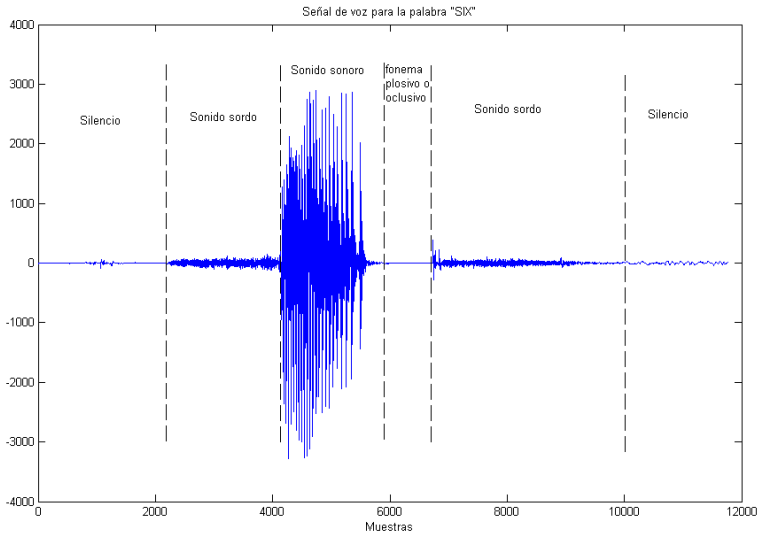


Figura 2.1: Representación de una señal de voz en el dominio del tiempo.

Otra alternativa para caracterizar la señal de voz y representar la información asociada de los sonidos es, utilizando la representación espectral. La forma más común de este tipo de representaciones es el *espectrograma* del sonido. Los

espectrogramas representan en tres dimensiones, la intensidad de la voz en las diferentes bandas de frecuencias y el comportamiento de las frecuencias en el tiempo. La figura 2.2 muestra el espectrograma de la señal de voz para la pronunciación "ALTO". Esta representación de la variación de la señal en el tiempo y la frecuencia es la base para la parametrización del modelo de voz. Sin embargo, el mayor problema que se presenta con este tipo de representaciones es la dificultad para establecer estimaciones confiables de las frecuencias formantes para sonidos sonoros de bajo nivel, así como la dificultad para encontrar las frecuencias formantes de los sonidos sordos y los silencios.

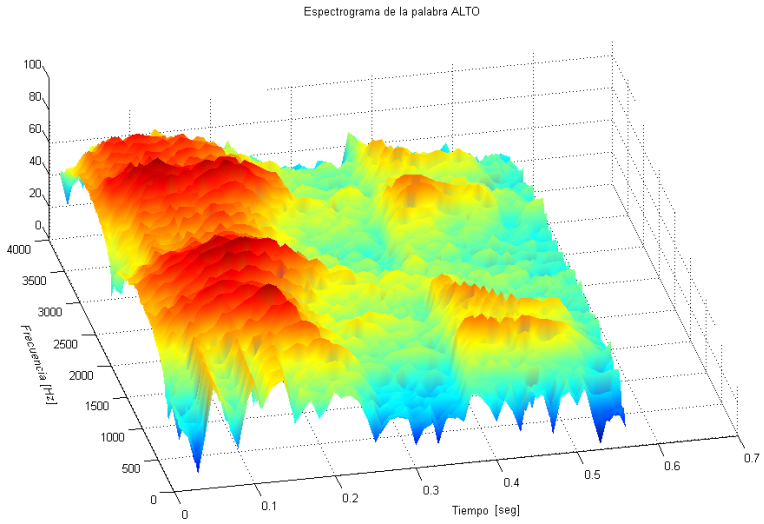


Figura 2.2: Espectrograma de la palabra ALTO. Se visualizan tres dimensiones: horizontal (tiempo), vertical (frecuencia) e intensidad de la señal (coloración)

2.3. Preénfasis

En el espectro de la voz existe una caída de -6 dB/octava, conforme la frecuencia aumenta. Esto se debe a la combinación de una caída de -12 dB/octava ocasionada por la fuente de excitación de la voz y un incremento de +6 dB/octava ocasionado por la radiación de la boca. Es decir, cada vez que la frecuencia aumenta al doble, la amplitud de la señal se reduce en un factor de 16. Por lo que se desea compensar esta caída de -6 dB/octava con una manipulación de la señal de voz que de un incremento de +6 dB/octava en el rango apropiado, de manera que la medición del espectro tenga un rango dinámico similar a lo largo de todo su ancho de banda. A esto se le conoce como preénfasis. En un sistema de procesamiento digital de voz, el preénfasis puede ser implementado de dos formas: ya sea por un filtro analógico paso altas de primer orden, con una frecuencia de

paso de 3 dB, en algún punto entre los 100 Hz y 1 kHz (la posición exacta no es crítica), el cual precede al filtrado *antialiasing* y al convertidor A/D; o con un filtro digital paso altas que procese a la señal de voz digitalizada. Este filtro digital puede ser implementado con la siguiente ecuación en diferencias:

$$y[n] = x[n] - ax[n-1] \tag{2.1}$$

Donde cada muestra leída de la señal se almacena en $x[n]$. Además, $x[n-1]$ representa una muestra inmediatamente anterior en el tiempo. Por otro lado, $y[n]$ es la salida del filtro de preénfasis en el tiempo n. Y finalmente a es una constante, para nuestro caso $a = 0{,}97$, de acuerdo a las recomendaciones de [Owens 93]. Las figura 2.3 muestra el efecto que tiene el filtro de preénfasis sobre una señal de voz "Sigue" en el dominio de la frecuencia.

2.4. Ventanas

En aplicaciones prácticas del procesamiento de señales, es necesario trabajar con pequeñas partes de la señal, a menos que la señal sea de corta duración. Esto es especialmente cierto cuando se utilizan técnicas de análisis convencional sobre señales, con características dinámicas no estacionarias, (como la señal de voz). En este caso, es necesario elegir una parte de la señal que pueda asumirse razonablemente como estacionaria [Deller, Proakis y Hansen 87].

Se nombra ventana en el dominio del tiempo discreto y denotada por $w(n)$, como una secuencia discreta real de tamaño finito, utilizada para seleccionar la muestras de una pequeña sección de la señal original, denotada por $x(n)$, por un proceso de multiplicación punto a punto [Deller, Proakis y Hansen 87]. De esta forma, se asume que la señal es igual a cero fuera del intervalo de interés. La ecuación 2.2 muestra la forma de aplicar una ventana $w(n)$ a la señal $x(n)$.

$$x_w(n) = x(n)w(n) \quad \text{Para } n = 0, 1, \cdots, N-1 \tag{2.2}$$

Existen varios tipos de ventanas, las más comunes se muestran a continuación:

- *Rectangular*

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

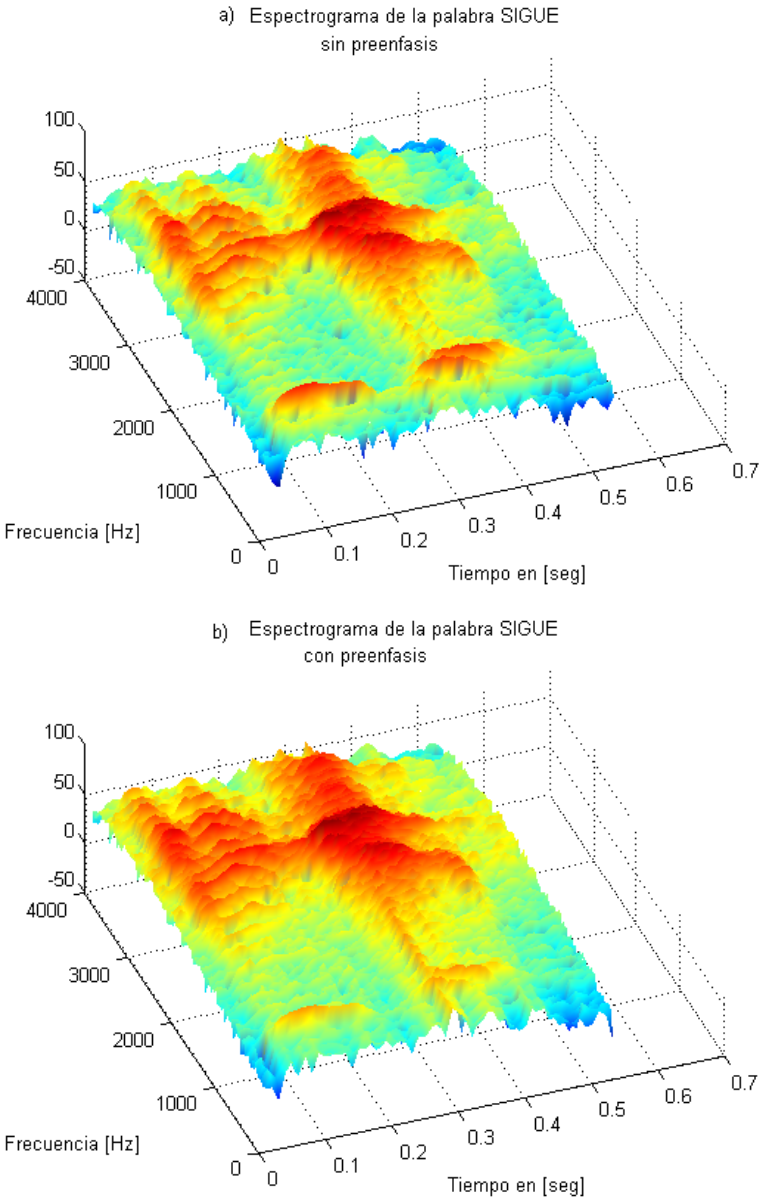


Figura 2.3: Efecto del filtro preénfasis en la señal de voz "SIGUE". a) Señal de voz sin filtrar. b) Señal después de aplicar un filtro preénfasis.

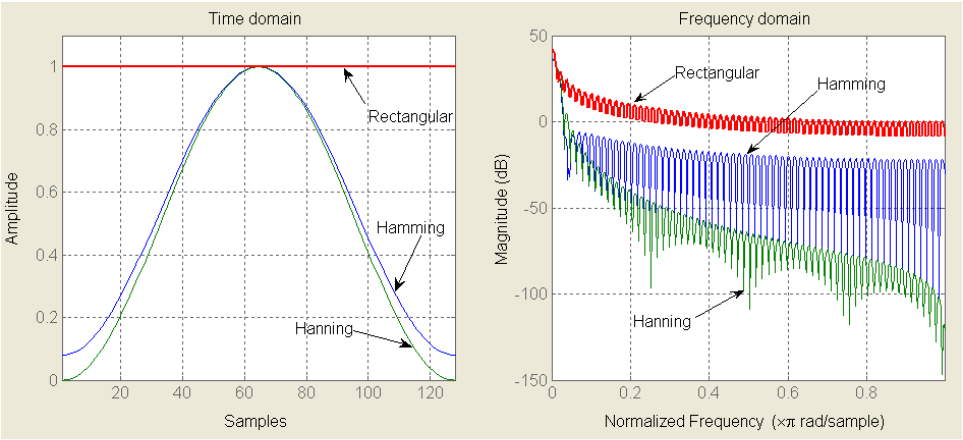
- *Hamming*

$$w(n) = \begin{cases} 0{,}54 - 0{,}46\cos\left(\dfrac{2\pi n}{N-1}\right) & \text{para } n = 0, 1, \cdots, N-1 \\ 0 & \text{otro caso} \end{cases}$$

- *Hanning*

$$w(n) = \begin{cases} 0{,}5 - 0{,}5\cos\left(\dfrac{2\pi n}{N-1}\right) & \text{para } n = 0, 1, \cdots, N-1 \\ 0 & \text{otro caso} \end{cases}$$

La figura 2.4 muestra la representación de las ventanas en el dominio del tiempo discreto y la frecuencia para 128 puntos.



a) Ventanas en el domonio del tiempo b) Ventanas en el dominio de la frecuencia

Figura 2.4: Tipos de ventanas para 128 puntos.

2.5. Transformada Rápida de Fourier (FFT)

En aplicaciones de análisis espectral, a menudo se requiere el cálculo de la Transformada Discreta de Fourier (DFT) en tiempo real, de un conjunto de muestras de entrada. El problema del cálculo de la DFT, es calcular la secuencia $\{X(k)\}$ de N números complejos, dada la secuencia de datos $\{x(n)\}$, de longitud N según la fórmula 2.3

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad \text{Para } k = 0, 1, 2, \cdots, N-1 \tag{2.3}$$

donde

$$W_N^{kn} = e^{-j2\pi/N} \tag{2.4}$$

El cálculo directo de la DFT es básicamente ineficiente, debido a que no explota las propiedades de simetría y periodicidad del factor de fase W_N. En particular, estás dos propiedades son:

Propiedad de simetría: $W_N^{k+N/2} = -W_N^k$
Propiedad de periodicidad: $W_N^{k+N} = W_N^k$

En el desarrollo de algoritmos computacionalmente eficientes para la DFT, se adopta la estrategia de *"divide y vencerás"*. Este método se basa en la descomposición de una DFT de N puntos en DFTs más pequeñas. Con el uso de estos métodos se llega a una familia de algoritmos computacionalmente eficientes, conocidos como algoritmos FFT (Transformada Rápida de Fourier)

2.5.1. Algoritmo del la FFT con decimación en el tiempo

Existen varias aproximaciones para el desarrollo del algoritmo de la FFT, uno de ello es propuesto por [Brigham 74] El algoritmo transforma los índices para la frecuencia y el tiempo en forma binaria en orden para realizar la descomposición. Se comienza con la expansión de la DFT en factores de dos DFTs de tamaño $N/2$, se divide las muestras en muestras pares e impares, por lo que la ecuación 2.3 queda de la siguiente forma.

$$X(k) = \sum_{m=0}^{(N/2)-1} x(2m) W_{N/2}^{2km} + \sum_{m=0}^{(N/2)-1} x(2m+1) W_{N/2}^{(2m+1)k} \tag{2.5}$$

sea

$$f_1(n) = x(2n) \tag{2.6}$$

$$f_2(n) = x(2n+1) \qquad \text{Para } n = 0, 1, 2, \cdots, \frac{N}{2} - 1 \tag{2.7}$$

Sustituyendo las ecuaciones 2.6 y 2.7 en 2.5, y usando la propiedad $W_N^{2n} = W_{N/2}^n$ se tiene.

$$X(k) = \sum_{m=0}^{(N/2)-1} f_1(m) W_{N/2}^{km} + W_N^k \sum_{m=0}^{(N/2)-1} f_2(m) W_{N/2}^{km} \tag{2.8}$$

o lo que es lo mismo

$$X(k) = F_1(k) + W_N^k F_2(k) \qquad \text{Para } k = 0, 1, 2, \cdots, N-1 \tag{2.9}$$

donde $F_1(k)$ y $F_2(k)$ son las DTFs de $N/2$ puntos de la secuencias $f_1(m)$ y $f_2(m)$, respectivamente.

La figura 2.5 muestra gráficamente, la descomposición del cálculo de la DFT en su primera *etapa*, para $N = 8$ puntos.

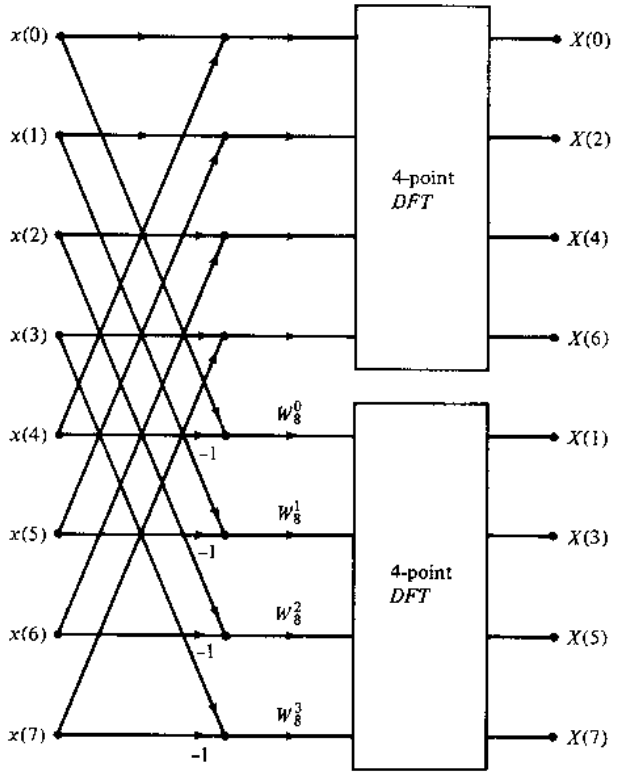


Figura 2.5: Primer paso en el desarrollo de la FFT de 8-puntos con decimación en el tiempo.

Puesto que $F_1(k)$ y $F_2(k)$ son periódicas, con periodo $N/2$, tenemos que

$F_1(k+N/2) = F_1(k)$ y $F_2(k+N/2) = F_2(k)$. Además, $W_N^{k+N/2} = -W_N^k$, por tanto 2.9 se puede expresar como:

$$X(k) = F_1(k) + W_N^k F_2(k) \tag{2.10}$$

$$X(k+\frac{N}{2}) = F_1(k) - W_N^k F_2(k) \tag{2.11}$$

Para $k = 0, 1, 2, \cdots, \frac{N}{2} - 1$.

Las ecuaciones 2.10 y 2.11 son conocida comúnmente como la *mariposa* de la FFT (fig. 2.6). La intersección de los nodos representa de los valores X_1 y X_2 representan la suma compleja.

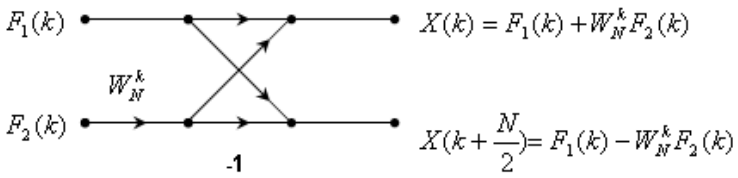


Figura 2.6: Mariposa básica del algoritmo de la FFT, con decimación en el tiempo.

El proceso de descomposición se puede repetir nuevamente para las DFTs de $N/2$ puntos y así sucesivamente hasta obtener DFTs de dos puntos. Cada descomposición es llamada *etapa* y el número total de etapas está dado por la ecuación 2.12.

$$M = \log_2(N) \tag{2.12}$$

Para el cálculo de la FFT de 8 puntos se requiere tres etapas como se muestra en la figura 2.7.

Con el uso de este método se realizan $N \log_2(N)$ sumas complejas y $(N/2)\log_2(N)$ multiplicaciones complejas, comparando con el cálculo directo de la DFT que es N^2, la velocidad en el cálculo se reduce demasiado.

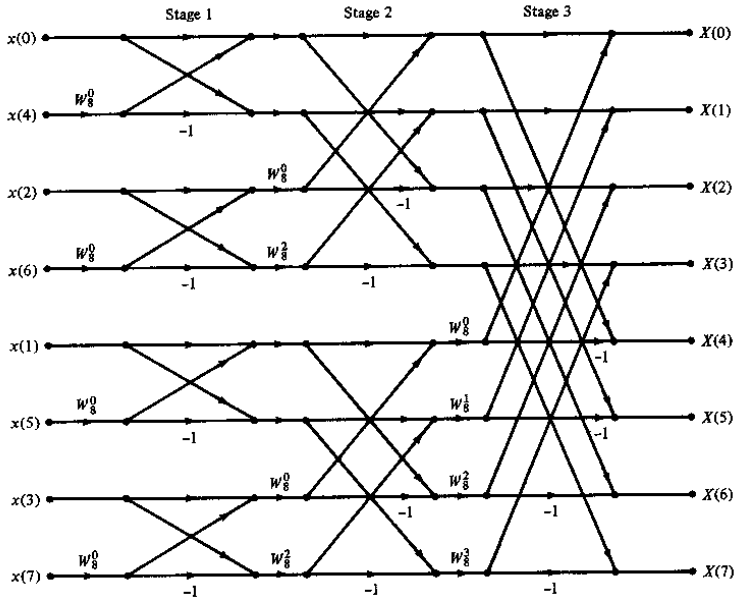


Figura 2.7: La FFT de 8 puntos decimada en el tiempo

2.5.2. Bits en reversa

Para el cálculo de la FFT, es necesario que las secuencias de muestras de entrada sean ordenadas en una forma especial llamada *decimación*. Se puede realizar la decimación mediante la representación de los índices en forma binaria, si los dígitos binarios son colocados en reversa [DeFatta 88]. Uno de los algoritmos para obtener la decimación de una secuencia con el uso de bits en reversa es dado por Rader [Rader 69]. La figura 2.8 muestra la forma de realizar la decimación a partir de la colocación de los bits en reversa.

2.6. Transformada de Coseno Discreta (DCT)

La Transformada Coseno Discreta (DCT) es muy usada en el procesamiento de voz y en la compresión de imágenes. Es muy semejante a la Transformada Discreta de Fourier (DFT), pero con el uso de números reales.

Formalmente, la DCT es una función lineal invertible $F : \mathbf{R}^N \rightarrow \mathbf{R}^N$, donde $\mathbf{R}$ es el conjunto de los números reales. Existen diferentes variantes de la DCT con pequeñas modificaciones en su definición. Los N números $x_0, \cdots, x_{N-1}$

Decimación (bits en reversa)				
Decimal	***Binario***		***Binario***	***Decimal***
0	000		000	0
1	001		100	4
2	010	⟷	010	2
3	011	⟷	110	6
4	100	⟷	001	1
5	101		101	5
6	110		011	3
7	111		111	7

Figura 2.8: Decimación con el uso de bits en reversa.

se transforman en N números reales $f_0, \cdots, f_{N-1}$ a partir de las siguientes ecuaciones:

- **DCT-I**

$$f_j = \frac{1}{2}(x_0 + (-1)^j x_{N-1}) + \sum_{k=1}^{N-2} x_k \cos\left[\frac{\pi}{N-1} jk\right] \qquad \text{Para } j = 0, 1, 2, \cdots, N-1 \tag{2.13}$$

Algunos autores multiplican los términos x_0 y x_{N-1} por $\sqrt{2}$, de igual manera multiplican los términos f_0 y f_{N-1} por $1/\sqrt{2}$.

- **DCT-II**

$$f_j = \sum_{k=0}^{N-1} x_k \cos\left[\frac{\pi}{N} j\left(k + \frac{1}{2}\right)\right] \qquad \text{Para } j = 0, 1, 2, \cdots, N-1 \tag{2.14}$$

La DCT-II es probablemente la más comúnmente usada y a menudo es llamada *"la DCT"*.

- **DCT-III**

$$f_j = \frac{1}{2} x_0 + \sum_{k=1}^{N-1} x_k \cos\left[\frac{\pi}{N}\left(j + \frac{1}{2}\right)k\right] \qquad \text{Para } j = 0, 1, 2, \cdots, N-1 \tag{2.15}$$

Debido a que ésta ecuación es la inversa de DCT-II, algunas veces es llamada como *"la inversa DCT"*.

- **DCT-IV**

$$f_j = \sum_{k=0}^{N-1} x_k \cos\left[\frac{\pi}{N}(j+\frac{1}{2})(k+\frac{1}{2})\right] \qquad \text{Para } j = 0, 1, 2, \cdots, N-1 \quad (2.16)$$

2.7. Tasa de cruces por cero

En el estudio de señales discretas, un cruce por cero ocurre si dos muestras sucesivas tienen signos distintos. La tasa de cruces por cero es una forma muy simple de medir el contenido frecuencial de una señal; lo cual resulta cierto para señales de banda angosta [Gardida 98].

Aunque las señales de voz son señales de banda ancha y la interpretación de la tasa de cruces por cero en promedio, es mucho menos precisa que la energía y magnitud promedio, se pueden estimar las propiedades espectrales de la señal de voz utilizando esta técnica. La tasa de cruces por cero en promedio, se define como [Deller, Proakis y Hansen 87]:

$$Z_n = \frac{1}{N} \sum_{n=m-N+1}^{m} \frac{\left|sgn[s(n)] - sgn[s(n-1)]\right|}{2} w(m-n) \qquad (2.17)$$

donde

$$sgn[x(n)] = \begin{cases} 1 & \text{para } x(n) >= 0 \\ -1 & \text{Otro caso} \end{cases} \qquad (2.18)$$

Debido a que las altas frecuencias involucran una alta tasa de cruces por cero y las bajas frecuencias involucran una baja tasa de cruces por cero, existe una íntima relación entre la tasa de cruces por cero y la distribución de la energía, con la frecuencia.

De acuerdo a la anterior relación se puede hacer una razonable generalización: cuando la tasa de cruces por cero es baja, existe voz sonora y viceversa, cuando la tasa de cruces por cero es alta, existe voz sorda. Tengamos cuidado, la aseveración anterior no es del todo correcta por lo que es necesario hacer otros tipos de consideraciones [Gardida 98].

2.8. Energía y magnitud promedio

La amplitud de los sonidos sordos es mucho menor que la amplitud de los sonidos sonoros. La energía en tiempo corto de la señal de voz proporciona una representación conveniente que refleja estas variaciones de la energía de la señal de voz, la cual se define como:

$$E_n = \sum_{n=m-N+1}^{m} s^2(n)w^2(m-n) \tag{2.19}$$

Una dificultad que se tiene al utilizar la función de energía en tiempo corto es que resulta muy sensible a valores grandes de la señal (debido a que es una sumatoria de términos cuadráticos). Para disminuir esos valore, es más conveniente el uso de la función de magnitud promedio, en vez de la función de energía, que se define como:

$$M_n = \sum_{n=m-N+1}^{m} |s(n)|w(m-n) \tag{2.20}$$

2.9. Detección de inicio y fin de la palabra

El problema de localizar donde inicia y donde termina una palabra, resulta importante en muchas áreas del procesamiento de voz y es de particular importancia en el reconocimiento de palabras aisladas. La detección permite trabajar con sólo las muestras que contienen información, es decir, con muestras que contienen voz, eliminando aquellas que contienen ruido y por tanto, no necesarias.

Un algoritmo muy usado para la detección del inicio y fin de una palabra, es el algoritmo de *Rabiner-Sambur* [Rabiner 75], está basado en la combinación de dos mediciones realizadas en el dominio del tiempo (magnitud promedio y cruces por cero).

2.9.1. Algoritmo de *Rabiner-Sambur*

El algoritmo definen tres tipos de umbrales para la detección del inicio y fin de las palabras aisladas: *umbral superior de energía*, *umbral inferior de energía* y *umbral de cruces por cero*. Estos umbrales son obtenidos de la siguiente forma:

- *UmbSupEnerg*: Umbral superior de energía, se obtiene a partir de la magnitud de las muestras de voz.

$$UmbSupEnerg = 0{,}5 \max_{n}\{M_n\} \tag{2.21}$$

- *UmbInfEnerg*: Umbral inferior de energía. Este valor es obtenido a partir de la magnitud de las muestras del ruido.

$$UmbInfEnerg = \mu_{M_s} + 2\sigma_{M_s} \tag{2.22}$$

donde

μ_{M_s}: es la media de la magnitud del ruido.
σ_{M_s}: es la desviación estándar de la magnitud del ruido.

- *UmbCruCero*: Umbral de cruces por cero. Se obtiene a partir de la media y la desviación estándar del cruce por ceros del ruido.

$$UmbCruCero = \mu_{Z_s} + 2\sigma_{Z_s} \tag{2.23}$$

donde

μ_{Z_s}: es la media de los cruces por cero del ruido.
σ_{Z_s}: es la desviación estándar de los cruces por cero del ruido.

El procedimiento para la detección del inicio y fin de una palabra se describe a continuación:

- Se obtienen por cada trama de n muestras, la magnitud promedio de la señal y su tasa de cruces por cero.
- Se obtiene el umbral superior de energía (*UmSupEnerg*).
- Considerando que las primeras 10 tramas no contienen voz, se calculan los umbrales del ruido: *UmbInfEnerg* y *UmbCruCeros*.
- Enseguida, se busca la trama donde se rebasa, por primera vez, el umbral *UmSupEnerg*. En este punto se asegura la existencia de voz en la trama.
- A partir de la trama localizada en el punto previo, se busca en dirección al inicio (o final) de la grabación, hasta un punto N1 (o N2), donde la magnitud promedio queda por debajo del umbral *UmbInfEnerg*. A este punto se selecciona tentativamente como el inicio (o final) de la palabra.

- El siguiente paso, es la comparación de la tasa de cruces por cero de las tramas y el umbral *UmbCruCeros*. Esto se realiza para las tramas que preceden a N1 (o siguen a N2). Si la tasa de cruces por cero excede el umbral, tres o más veces, el inicio N1 es recorrido hasta el punto en donde el umbral fue sobrepasado por primera vez y es marcado como N1' (inicio de la palabra), en caso contrario el inicio se escoge en el primer punto N1.
- Se realiza un procedimiento similar para determinar el final de la palabra.

Un ejemplo típico de este método se muestra en la figura 2.9.

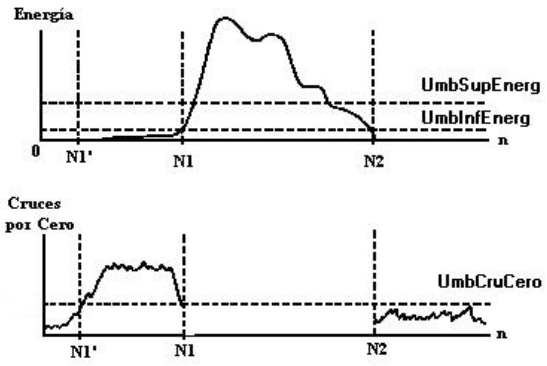


Figura 2.9: Gráficas Típicas de la Magnitud Promedio y los Cruces por Cero

Capítulo 3

Extracción de características

En el contexto del reconocimiento de voz, la meta principal del proceso de extracción de característica, es el cálculo de una secuencia de vectores que representan la señal de voz en una forma compacta.

La extracción de características por lo general se realiza en tres etapas. La primera se llama *análisis de voz* o *acoustic front-end*, en esta etapa se realiza algunos tipos de análisis espectro-temporal de la señal y se generan vectores de características base, los cuales describen la envolvente del espectro de la señal en intervalos cortos. La segunda etapa, recolecta vectores de características extendidos, que se componen de características estáticas y dinámicas (ejemplo: ΔF y $\Delta\Delta$F). Finalmente la última etapa (que no siempre se presenta), transforma los vectores de características extendidos en vectores más compactos y robustos [Martens 00]. En este capitulo se describe en particular el Análisis de la voz por medio de vectores Cepstral, dejando fuera del alcance de la tesis las otras dos etapas.

3.1. Análisis de voz

El primer paso de un extractor de características es destinado para producir alguna representación de la envolvente del espectro en términos cortos. Esta envolvente espectral es una versión suavizada del espectro detallado, de esta forma, expone una variabilidad significante pequeña que el espectro base.

Existen varias formas de obtener los vectores de características, las más comunes son:

- *Análisis LPC*: Este método es muy usado, por ser rápido y simple. Proporciona una buena aproximación a la envolvente espectral del tracto vocal, con el uso de un modelo todo polos; especialmente para las regiones de sonidos sonoros de la voz en estado cuasi estable. Durante las regiones transitorias y en sonidos sordos, el modelo LPC es menos efectivo que en las regiones con sonidos sonoros, pero aun provee un modelo aceptable para propósitos de reconocimiento de voz [Rabiner 93].
- *Análisis Cepstral*: La señal de voz es frecuentemente supuesta como la salida de un sistema *lineal e invariante en el tiempo* (LIT), es decir, es la convolución de una entrada y la respuesta al impuso. Para caracterizar la señal en términos de los parámetros de ese modelo, es necesario un proceso de de-convolución. El análisis cepstral es el procedimiento mas común para tal propósito, representa la envolvente espectral del tracto vocal a partir de una transformación *homomórfica* [Oppenheim 89].

3.2. Análisis Cepstral

En el procesamiento de voz, la aplicación de la técnica del análisis cepstral está basado en la suposición de que la producción de la voz, aun que sea un proceso variante en el tiempo, puede ser modelada en tiempo corto como: la convolución de una función de excitación (un tren de impulsos cuasi-periódicos o como un ruido aleatorio) con la respuesta al impulso del tracto vocal [Rabiner 93]. El análisis cepstral, algunas veces llamado como una *descomposición homomórfica*, es una técnica diseñada para separar componentes de una señal convolucionada por medio de una transformación, en donde en ese dominio, la convolución se convierte en una simple suma [Oppenheim 68].

Sea

$$s(t) = x(t) * y(t) \tag{3.1}$$

donde $*$ denota la convolución. Si obtenemos la transformada de Fourier de ambos lados, se tiene

$$S(w) = X(w)Y(w) \tag{3.2}$$

que denota el espectro de la señal. Por otra parte, si obtenemos el logaritmo en ambos lados, se obtiene

$$\ln S(w) = \ln X(w) + \ln Y(w). \tag{3.3}$$

De esta forma, una convolución en el dominio del tiempo, se ha transformado en una suma de componentes logarítmicas en el dominio de la frecuencia.

Finalmente, para separar las componentes y y x, se debe aplicar una transformada inversa de Fourier para el logaritmo del espectro dado

$$\mathcal{F}^{-1}\{\ln S(w)\} = \mathcal{F}^{-1}\{\ln X(w)\} + \mathcal{F}^{-1}\{\ln Y(w)\}, \tag{3.4}$$

donde $\mathcal{F}\{\cdot\}$ denota la transformada de Fourier (FT) y $\mathcal{F}^{-1}\{\cdot\}$ la transformada inversa de Fourier (IFT). La figura 3.1 muestra el diagrama de bloques de todo el proceso.

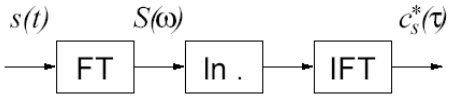


Figura 3.1: Diagrama de bloques del proceso de la transformación homomórfica.

Esta última transformación se encuentra en el dominio del tiempo, pero no es el mismo tiempo que el de la señal original; en efecto, este es una medida de la velocidad del cambio de la magnitud espectral. Este dominio es llamado *cepstral*, y el eje del tiempo es a menudo referenciado como el eje *"quefrency"*

3.2.1. Cepstrum reales y complejos

Los *cepstrum reales* de una señal digital $x[n]$ están definidos como:

$$c[n] = \frac{1}{2\pi}\int_{\pi}^{-\pi} \ln|X(e^{jw})|e^{jwn}dw \tag{3.5}$$

donde $X(e^{jw})$ es la transformada de Fourier de $x[n]$.

De la misma forma, los *cepstrum complejos* de $x[n]$ se definen como

$$\hat{x}[n] = \frac{1}{2\pi}\int_{\pi}^{-\pi} \ln X(e^{jw})e^{jwn}dw \tag{3.6}$$

donde se usa el logaritmo complejo dado por:

$$\hat{X}(e^{jw}) = \ln X(e^{jw}) = |X(e^{jw})| + j\theta(w) \tag{3.7}$$

y la fase $\theta(w)$ esta dada por

$$j\theta(w) = arg[X(e^{jw})] \tag{3.8}$$

Si la señal $x[n]$ es real, el cepstrum real $c[n]$ y el cepstrum complejo $\hat{x}[n]$ son también señales reales. Por tanto el término cepstrum complejo, no se refiere a que este es una señal compleja, sino que se toma el logaritmo complejo. A lo largo de este trabajo sólo se usará el término cepstrum para denotar a los cepstrum reales.

3.2.2. Cepstrum para secuencias finitas

Para secuencias finitas, se puede calcular los cepstrum (real o complejo) con el uso de una DFT, de esta forma

$$X_a[k] = \sum_{n=0}^{N-1} x[n]e^{j2\pi nk/N}, \qquad \text{Para } k = 0, 1, 2, \cdots, N-1 \tag{3.9}$$

$$\hat{X}_a[k] = \ln X_a[k], \qquad \text{Para } k = 0, 1, 2, \cdots, N-1 \tag{3.10}$$

$$\hat{x}_a[n] = \frac{1}{N} \sum_{n=0}^{N-1} \hat{X}_a[k]e^{-j2\pi nk/N}, \quad \text{Para } k = 0, 1, 2, \cdots, N-1 \tag{3.11}$$

El cepstrum complejo obtenido de esta forma no es exacto, debido a que el logaritmo complejo usado en el cálculo de la DFT, es una versión muestreada de $\hat{X}(e^{jw})$, de esta forma, el resultado de la transformación inversa es una versión con *aliasing* del verdadero cepstrum complejo. Por lo que,

$$\hat{x}_a[n] = \sum_{r=-\infty}^{\infty} \hat{x}[n + rN] \tag{3.12}$$

Una aproximación del cepstrum (usando la DFT inversa) es

$$\hat{x}_a[n] = \frac{1}{N} \sum_{n=0}^{N-1} |\hat{X}_a[k]|e^{-j2\pi nk/N}, \qquad \text{Para } k = 0, 1, 2, \cdots, N-1 \tag{3.13}$$

la relación con el verdadero cepstrum es

$$\hat{c}_a[n] = \sum_{r=-\infty}^{\infty} c[n + rN] \tag{3.14}$$

Para minimizar el *aliasing* en el cálculo de $\hat{x}_a[n]$, $c_a[n]$, se necesita un valor largo para N (512 o superior).

3.3. Cepstrum en escala Mel

3.3.1. Escala Mel

Es bien conocido que la resolución espectral del sistema auditivo del ser humano no es constante a lo largo del eje de la frecuencia. El ser humano puede fácilmente discriminar entre tonos de 200 y 250 Hz, pero no lo puede hacer entre los tonos 2000 y 2050 Hz. Para imitar las características espectrales del sistema auditivo del ser humano, se debe usar el análisis espectral con una resolución fija en una escala de frecuencia subjetiva llamada, escala en frecuencia mel (donde *mel* es una unidad de la frecuencia subjetiva). Existe una relación monótona entre la escala en frecuencia mel (en mels) y la escala en frecuencia física (en Hz). La escala en frecuencia mel es lineal hasta 1000 Hz y logarítmica en frecuencia superiores [Quatieri 02]. La siguiente función transforma las frecuencias lineales a frecuencias mel

$$f_{mel} = 2595 \log 10\Big(1 + \frac{f}{700}\Big) \tag{3.15}$$

donde f es la frecuencia en Hz. La figura 3.2 muestra la relación entre la Frecuencia lineal (en Hz) y la Frecuencia Mel (en mels).

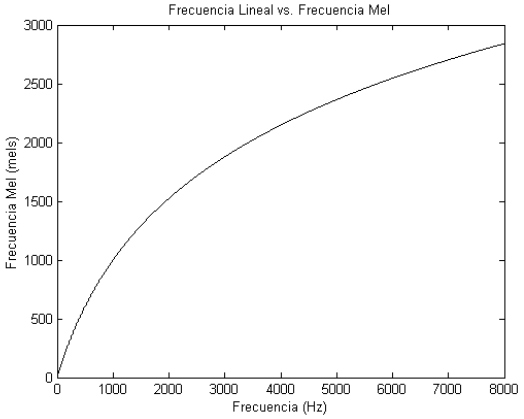


Figura 3.2: Relación entre la Frecuencia Lineal y la Frecuencia Mel.

3.3.2. Enmascaramiento

El fenómeno del enmascaramiento de frecuencia es un hecho en donde, un sonido no puede ser percibido por oído, si otro muy cercano en frecuencia, tiene un nivel bastante alto. Los niveles de la frecuencia de enmascaramiento han sido

determinados empíricamente. La figura 3.3 muestra dos tonos en los cuales sólo se percibe el que tiene mayor intensidad.

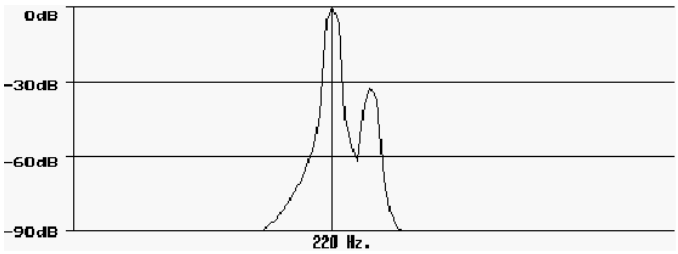


Figura 3.3: Enmascaramiento de un sonido. El tono con mayor intensidad enmascara al segundo tono.

El umbral de enmascaramiento es referenciado comúnmente como la función en escala Bark. La escala Bark proporciona una escala alternativa de percepción similar a la escala Mel (se ha comprobado que estas dos escalas tienen el mismo desempeño para el reconocimiento de voz [Shannon y Paliwal 03]). Cada punto en la membrana basilar puede ser considerado como un filtro paso banda con un ancho de banda igual a una *banda crítica* o un Bark [Seneff 88].

3.3.3. Banco de filtro

El banco de filtros es una técnica clásica en el análisis espectral, que consiste en representar el espectro de una señal, por la energía logarítmica en la salida de un banco de filtros; en donde se usan filtros paso banda traslapados a lo largo del eje de las frecuencias. Esta representación da una burda aproximación de la forma del espectro de la señal.

Considerando el fenómeno del enmascaramiento de las frecuencias del sistema auditivo, se ha implantado el banco de filtros con respuesta en frecuencia triangular, traslapados y con separación uno del otro determinado por un intervalo constante, en frecuencia Mel [Davis y Mermelstein 80]. La figura 3.4 muestra el diseño del banco de filtro propuesto por Davis y Mermelstein.

Existen otras alternativa para el diseño del banco de filtros, una de ellas, calcula el promedio del espectro alrededor de la frecuencia central en cada filtro [Huang, Acero y Hon 01]. Dos respuestas en frecuencia de este tipo de banco de filtros triangulares, están dadas por las ecuaciones 3.16 y 3.17:

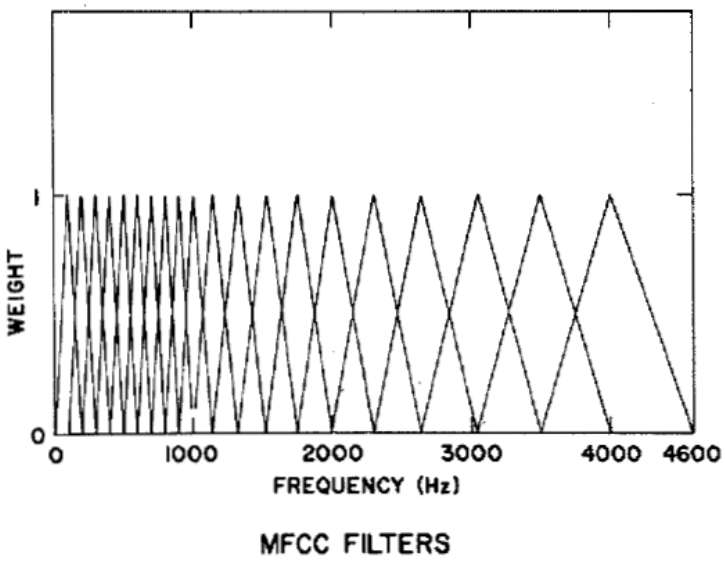


Figura 3.4: Banco del filtro propuesto por Davis y Mermelstein.

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \frac{2\big(k - f[m-1]\big)}{\big(f[m+1] - f[m-1]\big)\big(f[m] - f[m-1]\big)} & f[m-1] \leq k < f[m] \\ \frac{2\big(f[m+1] - k\big)}{\big(f[m+1] - f[m-1]\big)\big(f[m+1] - f[m]\big)} & f[m] \leq k \leq f[m+1] \\ 0 & k > f[m+1] \end{cases} \tag{3.16}$$

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \frac{\big(k - f[m-1]\big)}{\big(f[m] - f[m-1]\big)} & f[m-1] \leq k < f[m] \\ \frac{\big(f[m+1] - k\big)}{\big(f[m+1] - f[m]\big)} & f[m] \leq k \leq f[m+1] \\ 0 & k > f[m+1] \end{cases} \tag{3.17}$$

en donde $m = 0, 1, 2, \ldots, M-1$ y M es el número de filtros. La figura 3.5 muestra la forma que tiene la respuesta en frecuencia del banco de filtros de la ecuación 3.16.

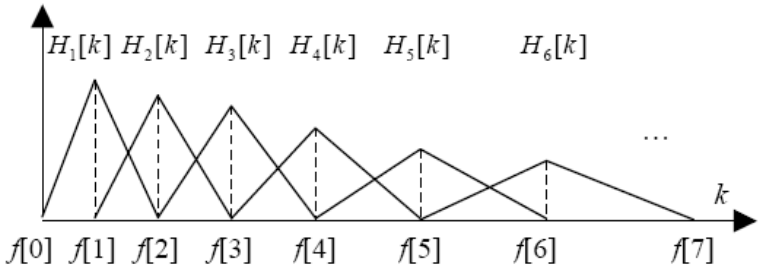


Figura 3.5: Filtros triangulares utilizando el promedio del espectro alrededor de la frecuencia central en cada filtro.

3.3.4. Coeficientes Mel Cepstral

Los *Coeficientes Mel Cepstral* (MFCC, *Mel-Frequency Cepstrum Coefficients*) son una representación definida como el cepstrum real, en la frecuencia Mel, de una señal analizada en tiempo corto. Para calcular los coeficientes MFCC, se realizan los siguientes pasos [Davis y Mermelstein 80]:

- Se calcula el espectro de Fourier de una señal de voz analizada en tiempo corto, mediante la siguiente ecuación

$$X_a[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N} \qquad \text{Para } k = 0, 1, 2, \cdots, N-1 \qquad (3.18)$$

- El siguiente paso es calcular la energía logarítmica en la salida de cada filtro.

$$S[m] = \ln \left[\sum_{k=0}^{N-1} |X_a[k]|^2 H_m[k] \right] \qquad \text{Para } m = 0, 1, 2, \cdots, M-1 \quad (3.19)$$

- Posteriormente, se obtiene los coeficientes mel-cepstral, aplicando la transformada de coseno discreta ($DCT\text{-}II$) a la salida de los M filtros.

$$c[n] = \sum_{m=0}^{M-1} S[m] \cos\left(\pi n(m + 1/2)/M\right) \qquad \text{Para } m = 0, 1, 2, \cdots, M-1 \qquad (3.20)$$

  donde M varía de 24 a 40 para diferentes implementaciones. Para reconocimiento de voz, típicamente sólo se usan los primeros 13 coeficientes mel-cepstral [Huang, Acero y Hon 01].

Es importante notar que la representación de los MFCC ya no es una transformación homomórfica, pero podría serlo si el logaritmo y la sumatoria intercambian de orden en la ecu. 3.20, quedando de la forma:

$$S[m] = \sum_{k=0}^{N-1} \ln\left(|X_a[k]|^2 H_m[k]\right) \qquad \text{Para } m = 0, 1, 2, \cdots, M-1 \qquad (3.21)$$

En la práctica, sin embargo, la representación MFCC es aproximadamente homomórfica para filtros que tienen una función de transferencia uniforme. La ventaja de la representación MFCC usando 3.19 en lugar de 3.21, es debido a que, es más robusta al ruido y a errores de estimación espectral [Huang, Acero y Hon 01].

Capítulo 4

Clasificación de patrones

4.1. Cuantización Vectorial

La cuantización es un proceso de aproximar la amplitud de una señal continua por medio de símbolos discretos. La cuantización de un sólo valor, de una señal o parámetro, se le conoce como *Cuantización Escalar*.

La cuantización vectorial (VQ) es una generalización de la cuantización escalar, pero aplicada a vectores. Se ha usado con muy buenos resultados en la compresión de datos, codificación de voz, codificación de imágenes y reconocimientos de voz [Rabiner 93] y [Hedelin 95].

Un vector se puede utilizar para describir prácticamente cualquier tipo de patrón, como puede ser un segmento de una señal de voz o de una imagen, simplemente al formar un vector con las muestras de la señal de voz o de la imagen. La cuantización vectorial puede aplicarse al reconocimiento de patrones: como un patrón de entrada es comparado y aproximado a alguno de los patrones de referencia almacenados. El reconocimiento permite encontrar el patrón de referencia que más se acopla al patrón de entrada. Por lo tanto, la cuantización vectorial es más que una generalización de la cuantización escalar. En fechas recientes, se ha convertido en la principal herramienta del reconocimiento de voz, además de que sigue utilizándose en la compresión de señales de voz e imágenes [Gersho y Gray 97].

4.1.1. Definición

Partimos de un conjunto de vectores que pertenecen a un espacio K-dimensional, suponiendo que $\mathbf{x}$ es un vector perteneciente a ese conjunto, cuyos componentes $(x_1, x_2, \ldots, x_k)$ son variables aleatorias reales y de amplitud continua. Un cuantizador vectorial Q, de dimensión K y tamaño N, es una transformación de un vector $\mathbf{x}$, del espacio euclidiano de dimensión, en un conjunto finito $\mathbf{C}$ que contiene N salidas o puntos de reproducción, llamados vectores de código (*code vectors*) [Gersho y Gray 97]:

$$Q : \mathbf{R}^K \rightarrow \mathbf{C}, \qquad \mathbf{C} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\} \qquad \text{donde} \qquad \mathbf{y}_i \in \mathbf{R}^K \tag{4.1}$$

Al conjunto $\mathbf{C}$ es llamado libro de códigos (*codebook*) y tiene un tamaño N, esto significa que tiene N distintos elementos, cada uno de ellos dentro del espacio $\mathbf{R}^K$.

Asociado a cada cuantizador vectorial de N puntos, existe una partición de $\mathbf{R}^K$ en N regiones o celdas R_i para $i \in I = \{1, 2, \ldots, N\}$. La i-ésima celda esta definida por:

$$R_i = \left\{\mathbf{x} \in \mathbf{R}^K : Q(\mathbf{x}) = \mathbf{y}_i\right\} \tag{4.2}$$

algunas veces llamada imagen inversa o pre-imagen de $\mathbf{y}_i$ dentro del mapeo Q y denotada de forma más consistente por $R_i = Q^{-1}(\mathbf{y}_i)$.

De la definición de celdas, tenemos que:

$$\bigcup_i R_i = \mathbf{R}^K \qquad \text{y} \qquad R_i \bigcap R_j = \emptyset \qquad \text{para} \qquad i \neq j \tag{4.3}$$

donde las celdas forman una partición de $\mathbf{R}^K$. La figura 4.1 muestra gráficamente las celdas o regiones R_i y los centroides $\mathbf{y}_i$ (asteriscos), de un conjunto de vectores de dos dimensiones.

Una propiedad importante de algunos conjuntos en $\mathbf{R}^K$, es ser convexos. Para el caso de dos o tres dimensiones, un conjunto se dice convexo si, dados dos puntos cualesquiera dentro de él, existe una línea recta que los une y pertenece al mismo conjunto. Esta idea se extrapola a $\mathbf{R}^K$ de la siguiente manera: un conjunto $\mathbf{S} \in \mathbf{R}^K$ es convexo si $\mathbf{a}$ y $\mathbf{b} \in \mathbf{S}$ implica que $\alpha\mathbf{a} + (1 - \alpha)\mathbf{b} \in \mathbf{S}$, $\forall$ $0 < \alpha < 1$.

Un cuantizador vectorial se denomina regular sí:

- Cada celda, R_i, es un conjunto convexo.

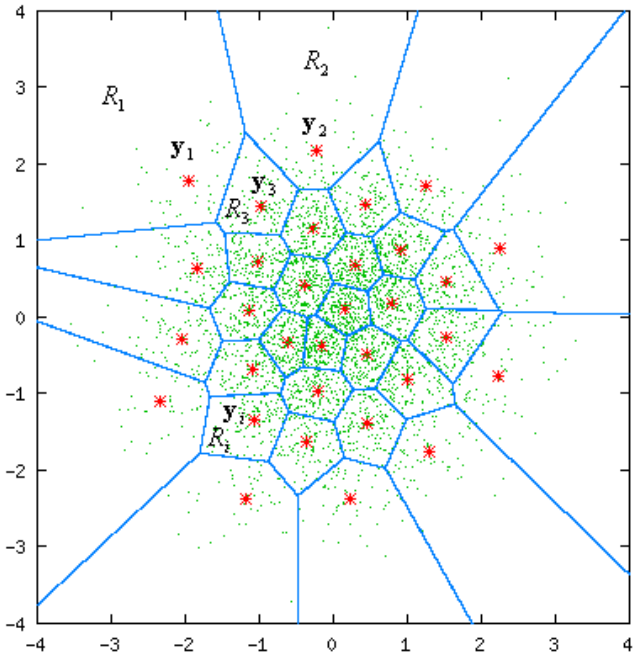


Figura 4.1: Ejemplo de un cuantizador vectorial en dos dimensiones.

- Para cada i, $\mathbf{y}_i \in R_i$.

La tarea de codificación de un cuantizador vectorial es, examinar cada vector de entrada $\mathbf{x}$ e identificar a qué celda, K-dimensional del espacio $\mathbf{R}^K$, pertenece. El codificador vectorial simplemente identifica el índice i de la región y el decodificador vectorial genera el vector del código $\mathbf{y}_i$ que representa a esta región [Gersho y Gray 97].

El conjunto $\mathbf{y}$ es conocido como diccionario de reconstrucción o simplemente diccionario, donde N es el tamaño del diccionario e $\{\mathbf{y}_i\}$ es el conjunto de vectores del código. Los vectores $\mathbf{y}_i$ son conocidos también en la literatura de reconocimiento de patrones, como los patrones de referencia o plantillas. El tamaño N del diccionario, también se conoce como número de niveles, término proveniente de la cuantización escalar. De esta forma, se puede hablar de un diccionario de N niveles. Al proceso de creación del diccionario, también se le conoce como entrenamiento o población del diccionario.

El problema de diseño de un cuantizador es: encontrar un algoritmo efectivo que obtenga el conjunto de centroides $\mathbf{C}$ y las regiones asociadas $\chi = \{R_1, R_2, \ldots, R_N\}$, dado un conjunto de vectores $\tau = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ y una medida de distorsión $d(\mathbf{x}, \mathbf{y})$. El problema se facilita si se tiene la medida de distorsión y el número de centroides N.

Existen varios algoritmos de agrupamiento, entre los que tenemos: *simplex*, *distancia máxima*, *K-Medias*, *ISODATA* y *LBG*; estos dos últimos son variantes del agrupamiento K-Medias, pero con mayor complejidad.

4.1.2. Distancias y medidas de distorsión

Un componente clave en la mayoría de los algoritmos de comparación de patrones, es formular una medida de distorsión entre dos vectores de características. Esta medida de distorsión, puede ser manejada con rigor matemático si los patrones son visualizados en un espacio vectorial.

Suponiendo que se tienen dos vectores de características, $\mathbf{x}$ e $\mathbf{y}$, definidos en un espacio vectorial χ. Se define una *métrica* o *función de distancia*, d, en el espacio vectorial χ, como una función de valor real, sobre el producto cartesiano $\chi \times \chi$, que cumpla las siguientes condiciones [Rabiner 93]:

1. $0 \leq d(\mathbf{x}, \mathbf{y}) < \infty$, para $\mathbf{x}, \mathbf{y} \in \chi$ y $d(\mathbf{x}, \mathbf{y}) = 0$ si y sólo sí $\mathbf{x} = \mathbf{y}$
2. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ para $\mathbf{x}, \mathbf{y} \in \chi$

3. $d(\mathbf{x},\mathbf{y}) \leq d(\mathbf{x},\mathbf{z}) + d(\mathbf{z},\mathbf{y})$ para $\mathbf{x},\mathbf{y},\mathbf{z} \in \chi$

   Además, una función de distancia se le denomina invariante sí:

4. $d(\mathbf{x}+\mathbf{z},\mathbf{y}+\mathbf{z}) = d(\mathbf{y},\mathbf{x})$

Las primeras tres propiedades son conocidas comúnmente como: *positividad* (no negatividad), *simetría* y *desigualdad del triángulo*, respectivamente. Una métrica que contenga estas propiedades, permite un alto grado de manejo matemático. Si una medida de distancia, d, satisface sólo la propiedad de positividad, se le denomina medida de distorsión, particularmente cuando los vectores son representaciones del espectro de la señal.

Para el procesamiento de voz es importante considerar que la definición (o elección), de la medida de distancia, es significativamente subjetiva. Una medida matemática de la distancia, para ser utilizada en el procesamiento de voz, debe tener una alta correlación entre su valor numérico y su distancia subjetiva aproximada, para evaluar una señal real de voz. En el reconocimiento de voz, la consistencia *psicofísica* (los diferentes matices que se le pueden imprimir a una misma palabra o frase), que se desea medir con la distancia, obliga a que se encuentre una medida matemática, ajustada por necesidad, a las características lingüísticas conocidas. Estos requisitos tan subjetivos no pueden ser satisfechos con medidas de distancia que proporcionen manejo matemático. Un ejemplo es la tradicional medida del Error Cuadrático, $d(\mathbf{x},\mathbf{y}) = (\mathbf{x}-\mathbf{y})^2$.

Por otra parte, se hablará de "medidas de distorsión" en vez de "métricas", debido a que se relajan las condiciones de simetría y desigualdad del triángulo. No se debe utilizar el término *distancia* en sentido estricto, acorde a la definición de arriba. Además, se debe mantener la costumbre de la literatura de voz, donde el término distancia es análogo a las medidas de distorsión [Rabiner 93].

El hecho de medir las diferencias entre dos patrones de voz, en términos de promedios o distorsión espectral acumulada, parece ser un camino muy razonable de comparación de patrones, en términos del manejo matemático y su eficiencia computacional. Debido a que muchos estudios psicoacústicos, sobre las diferencias percibidas en el sonido, pueden ser interpretados en términos de diferencias en sus características espectrales, la medida de distorsión espectral puede ser a la vez, un argumento razonablemente matemático y subjetivo. Existen varios tipos de medidas de distorsión, cada una con sus características especiales. Entre ellas tenemos: *Distancia Euclidiana Cuadrática*, *Distorsión del Error Cuadrático Medio*, *Distorsión del Error Cuadrático Ponderado*, *Distancia de Itakura*, etc. A continuación se describe algunas medidas de distorsión más utilizadas.

- *Distancia Euclidiana Cuadrática.* Esta es la medida más conveniente y ampliamente usada para calcular distancias, también se le conoce como

Error Cuadrático. Se define como:

$$d(\mathbf{x},\mathbf{y}) = ||\mathbf{x}-\mathbf{y}||^2 = \sum_{j=1}^{N} \big(x_j - y_j\big)^2 \tag{4.4}$$

La distancia euclidiana entre dos vectores cepstral es una medida razonable de similitud espectral entre dos modelos [Deller, Proakis y Hansen 87].

- *Distorsión Error Cuadrático Medio (MSE).* Es otra de las medidas más utilizadas, se define como:

$$d(\mathbf{x},\mathbf{y}) = \frac{1}{N}\big(\mathbf{x}-\mathbf{y}\big)^T\big(\mathbf{x}-\mathbf{y}\big) = \frac{1}{N}\sum_{j=1}^{N}\big(x_j - y_j\big)^2 \tag{4.5}$$

  esta distorsión está definida por cada dimensión. La popularidad del MSE se basa en su simplicidad.

- *Distancia de Itakura.* Esta distancia se deriva utilizando una interpretación intuitiva del rango de predicción en el error de la energía. Fue obtenida originalmente, utilizando la máxima probabilidad existente entre argumentos similares. La distancia de Itakura es, probablemente, la medida de distorsión más empleada para encontrar la similitud entre dos vectores LPC [Deller, Proakis y Hansen 87].

  Esta distancia se define de la siguiente forma: sea $\mathbf{y}$ el vector aumentado LPC de referencia $[-1 \quad a_1 \quad a_2 \quad \cdots \quad a_p]$ y $\mathbf{x}$ el vector aumentado de los coeficientes LPCs observados $[-1 \quad a'_1 \quad a'_2 \quad \cdots \quad a'_p]$ y sea $\mathbf{R}$ la matriz de autocorrelación con la que se obtuvo el vector $\mathbf{y}$, entonces:

  $\mathbf{xRx}^T$: Energía a la salida del filtro inverso formado por referencia con la voz de entrada.

  $\mathbf{yRy}^T$: Energía mínima posible a la salida de un filtro LP con voz de entrada.

  Se define la distancia de Itakura como:

$$d(\mathbf{x},\mathbf{y}) = \log\left(\frac{E_{\mathbf{x}}}{E_{\mathbf{y}}}\right) \tag{4.6}$$

$$d(\mathbf{x},\mathbf{y}) = \log\left(\frac{\mathbf{xRx}^T}{\mathbf{yRy}^T}\right) \tag{4.7}$$

  en donde, para cualquier vector LP aumentado $\mathbf{a}$, se calcula $\mathbf{aRa}^T$ de la siguiente forma:

$$\mathbf{aRa}^T = \sum_{i=0}^{P} a_i \sum_{j=0}^{P} a_j r_j(|i-j|) \tag{4.8}$$

  En el caso de la autocorrelación, se puede calcular con una complejidad similar a la distancia euclidiana.

4.1.3. Algoritmo *K*-medias

K-medias es uno de los algoritmos más simples sin supervisión, que soluciona el problema del agrupamiento. El procedimiento sigue un método fácil y simple para clasificar un conjunto de datos, por medio de un cierto numero de grupos (supongamos K) establecidos previamente. El algoritmo se describe a continuación:

Sea

K: Numero de grupos

$\mathbf{z}_i$: Centroide para le grupo i.

χ_i: Subconjunto de los vectores de entrenamiento asignado al grupo i

Algoritmo

1. Escoger K centroides iniciales $\mathbf{z}_1(l), \mathbf{z}_2(l), \cdots, \mathbf{z}_K(l)$, en forma aleatoria.
2. En la iteración l, asignar los vectores a los grupos:
   Asignar

   $$\mathbf{x} \quad \text{a} \quad \chi_i(l), \quad \text{si } d(\mathbf{x}, \mathbf{z}_i(l)) \leq d(\mathbf{x}, \mathbf{z}_j(l)), \quad \text{para } j = 1, 2, \cdots, K \quad j \neq i$$

3. Calcular los nuevos centros de grupo:

   $$\mathbf{z}_i(l+1) = \frac{1}{N_i} \sum_{\mathbf{x} \in \chi_i(l)} \mathbf{x}, \quad \text{para} \quad i = 1, 2, \cdots, K$$

   donde N_i es el número de vectores asignados a $\chi_i(l)$
4. Si $\mathbf{z}_i(l+1) = \mathbf{z}_i(l)$ para $i = 1, 2, \cdots, K$; el algoritmo ha convergido y debe terminarse. En caso contrario, regresar al paso 2.

Una característica de este algoritmo es que los centroides o cuantizadores obtenidos, no son los óptimos globales; es decir, se obtienen cuantizadores óptimos locales. Estos dependen de varios factores, como son: asignación inicial de centroides (principalmente), orden de la toma de muestras, propiedades geométricas de los datos, tipo de distorsión empleada, etc. Una forma de poder alcanzar los óptimos globales, es probar con una gran variedad de centroides iniciales y seleccionar los cuantizadores finales que tengan la menor distorsión,

con respecto a los vectores a los cuales representan. Solución poco factible porque existe un gran número de combinaciones para los cuantizadores iniciales. Otra forma es, elegir los centroides iniciales de forma aleatoria, para buscar una distribución homogénea [Deller, Proakis y Hansen 87].

4.2. Modelos Ocultos de Markov(HMM)

Los *modelos ocultos de Markov* es un método estadístico muy usado para caracterizar las propiedades espectrales de una trama o patrón (estos modelos también son conocidos como: fuentes de Markov o funciones probabilísticas de cadenas de Markov). La fundamental suposición de los HMM es que, la señal de voz puede ser caracterizada como un proceso aleatorio paramétrico y que los parámetros del proceso estocástico, puede ser determinado (estimado) de una forma precisa y bien definida [Rabiner 89].

La teoría básica de los modelos ocultos de Markov fue publicada en una serie de *papers* clásicos por Baum y sus colegas [Baum 67], fue implementado en aplicaciones de procesamiento de voz por [Baker 75] en la Universidad Carnegie Mellon (CMU) y por [Jelinek 75] y sus colegas en la IBM.

4.2.1. Procesos de Markov en tiempo discreto

Considere un sistema que puede ser descrito en cualquier tiempo como: situarse en un estado, de un conjunto de N distintos: $s_1, s_2, \cdots, s_N$; tal y como muestra la figura 4.2 (donde $N = 5$, por simplicidad). En tiempos discretos de regular espacio, el sistema experimenta un cambio de estado (posiblemente regresa al mismo estado), conforme a un conjunto de probabilidades asociadas con el estado. Se denotará al instante de tiempo asociado al cambio de estado como $t = 1, 2, \cdots$ y se denotará al estado actual en el tiempo t como q_t. Una descripción probabilística completa del sistema de arriba, en general, requiere de la especificación del actual estado (en el tiempo t), así como todos los estado previos. Para el caso especial del una cadena de Markov discreta de primer orden, la descripción probabilística es truncada para sólo el estado actual y el predecesor, por ejemplo:

$$P[q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \cdots] = P[q_t = S_j | q_{t-1} = S_i]. \tag{4.9}$$

Además, sólo se considera esos procesos en los cuales, el lado derecho de 4.9 es independiente del tiempo, produciendo el conjunto de probabilidades de transiciones de estado a_{ij} de la forma:

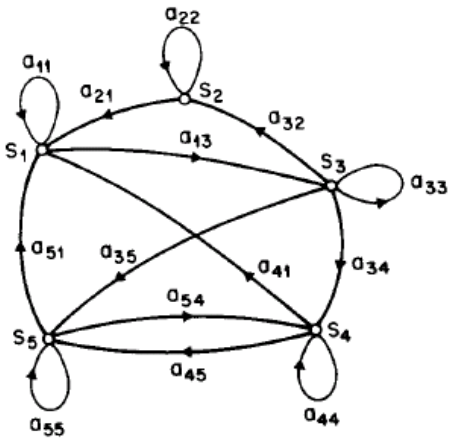


Figura 4.2: Una cadena de Markov con 5 estados (etiquetados con s_1 a s_5) con estados de transición elegidos.

$$a_{ij} = P[q_t = S_j | q_{t-1} = S_i], \quad \text{para} \quad 1 \leq i, j \leq N \tag{4.10}$$

donde los coeficientes de transición de estado, tienen las siguientes propiedades:

$$a_{ij} \geq 0 \qquad \forall j, i \tag{4.11}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \qquad \forall i \tag{4.12}$$

debido a que obedecen a las restricciones probabilísticas.

4.2.2. Definición de los modelos ocultos de Markov

En las cadenas de Markov, cada estado corresponde a un evento observable determinístico, es decir, la salida de la fuente en cualquier estado dado, no es aleatorio. Una extensión natural a las cadenas de Markov es el caso donde, la observación es una función probabilística del estado. A este nuevo modelo se le conoce como *modelo oculto de Markov*, que se puede ver como: un proceso estocástico doblemente incrustado con un proceso aleatorio fundamental, donde este no es observable directamente (es oculto), pero puede ser observable sólo a través de un proceso estocástico que produce la secuencia de observaciones [Rabiner 93].

Un modelo oculto de Markov es básicamente una cadena de Markov, donde la salida es una variable aleatoria X, generada acorde a la salida de la función de probabilidad asociada a cada estado. La figura 4.3 muestra un modelo oculto de Markov para el promedio del Dow Jones. Se observa que cada estado puede generar tres observaciones de salida: *subir*, *bajar* y *sin cambio* de acuerdo a su salida de la función de distribución de probabilidad.

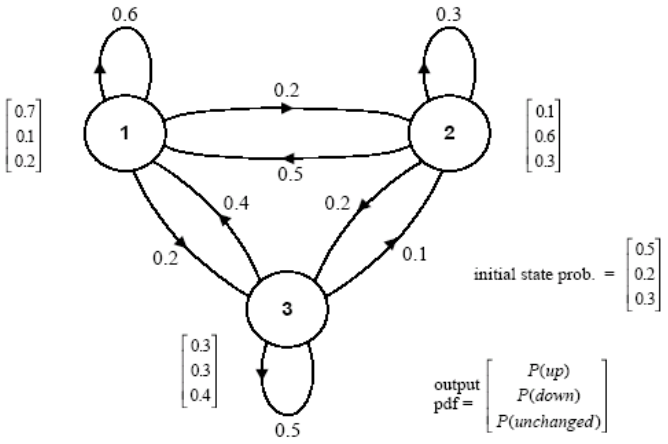


Figura 4.3: Un modelo oculto de Markov para el promedio del Dow Jones, con tres estados y tres probabilidades de símbolos observados.

Un modelo oculto de Markov es caracterizado por lo siguiente [Rabiner 89]:

1. $O = \{o_1, o_2, \ldots, o_M\}$, un alfabeto de salidas observadas[1]. Donde M es el número de distintos símbolos de observación por estado.

2. $\Omega = \{s_1, s_2, \ldots, s_N\}$, un conjunto de estados que representa el espacio de estados. Donde N es el número de estados en el modelo. El estado en el tiempo t se denota como q_t.

3. $\mathbf{A} = \{a_{ij}\}$, una matriz de probabilidades de transición, donde a_{ij} es la probabilidad de realizar una transición del estado i al estado j.

$$a_{ij} = P(q_t = s_j | q_{t-1} = s_i), \quad \text{para} \quad 1 \leq i, j \leq N \tag{4.13}$$

4. $\mathbf{B} = \{b_j(k)\}$, una matriz de distribución de probabilidad de observación de los símbolos, donde $b_j(k)$ es la probabilidad de observar el símbolo o_k, estando en el estado j

$$b_j(k) = P(o_k \; en \; t | q_t = s_j) \quad \text{para} \quad 1 \leq j \leq N, \quad 1 \leq k \leq M \tag{4.14}$$

[1]Si bien, se usan observaciones discretas de salida, se puede extender al caso continuo, en donde se usan funciones de distribución de probabilidad continuas.

5. $\pi = \{\pi_i\}$, Distribución de probabilidad del estado inicial, en donde:

$$\pi_i = P(q_i = s_i) \quad \text{para} \quad 1 \leq i \leq N \tag{4.15}$$

Debido a que a_{ij}, $b_j(k)$, y π_i son probabilidades, deben satisfacer las siguientes propiedades:

$$a_{ij} \geq 0, \quad b_j(k) \geq 0, \quad \pi_i \geq 0, \quad \forall\, i, j, k \tag{4.16}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{4.17}$$

$$\sum_{k=1}^{M} b_j(k) = 1 \tag{4.18}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{4.19}$$

Una completa especificación de un HMM requiere precisar: los dos parámetros del modelo N y M, los símbolos de observación y los tres conjuntos de medidas de probabilidad $\mathbf{A}$, $\mathbf{B}$ y π. Por conveniencia se usará la siguiente notación

$$\lambda = (\mathbf{A}, \mathbf{B}, \pi) \tag{4.20}$$

para indicar el conjunto completo de parámetros del modelo. Este conjunto de parámetros, define la medida de probabilidad para $\mathbf{O}$ como $P(\mathbf{O}|\lambda)$.

4.2.3. Los tres problemas básicos para los HMMs

Dada la forma del HMM, existen tres problemas básicos de interés, que deben ser resueltos para que el modelo sea útil en aplicaciones del mundo real. Esos problemas son los siguientes:

Problema 1: Dada la secuencia de observación $O = O_1, O_2, \cdots, O_T$, y el modelo $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$ ¿como hacer eficiente el cálculo $P(O|\lambda)$, la probabilidad de la secuencia observación, dado el modelo?.

Problema 2: Dada la secuencia de observación $O = O_1, O_2, \cdots, O_T$, y el modelo λ ¿como elegir una secuencia de estados correspondiente $Q = q_1\, q_2 \cdots q_T$ que sea óptima (por ejemplo, la mejor que "explica" a las observaciones)?.

Problema 3: ¿Como ajustar los parámetros del modelo $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$ para maximizar $P(O|\lambda)$?

4.2.4. Solución al problema 1 -Evaluación de la probabilidad

Para calcular la probabilidad de la secuencia de observación $O = O_1, O_2, \cdots, O_T$, dado el modelo λ, es decir $P(O|\lambda)$; la forma más directa es, enumerar cada posible secuencia de estados de longitud T (el número de observaciones). Considere una cierta secuencia de estados fija dada por:

$$Q = q_1 q_2 \cdots q_T \tag{4.21}$$

donde q_1 es el estado inicial. La probabilidad de la secuencia de observación O para la secuencia de estados dada por 4.21 es

$$P(O|Q, \lambda) = \prod_{t=1}^{T} P(O_t|q_t, \lambda) \tag{4.22}$$

donde se ha supuesto independencia estadística de las observaciones. De esta forma, se obtiene

$$P(O|Q, \lambda) = b_{q_1}(O_1) \cdot b_{q_2}(O_2) \cdots b_{q_T}(O_T) \tag{4.23}$$

Por otro lado la probabilidad de una secuencia de estados Q puede ser escrita como

$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \cdots a_{q_{T-1} q_T}. \tag{4.24}$$

La probabilidad conjunta de O y Q, es decir, la probabilidad que O y Q ocurran al mismo tiempo, es el simple producto de los dos términos de arriba

$$P(O, Q|\lambda) = P(O|Q, \lambda) P(Q|\lambda). \tag{4.25}$$

Por tanto, la probabilidad de O (dado el modelo) se obtiene sumando esta probabilidad conjunta, sobre todas las posibles secuencias de estados q, resultando

$$P(O|\lambda) = \sum_{todo\, Q} P(O|Q, \lambda) P(Q|\lambda) \tag{4.26}$$

$$= \sum_{q_1, q_2, \cdots, q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \cdots a_{q_{T-1} q_T} b_{q_T}(O_T). \tag{4.27}$$

El cálculo de $P(O|\lambda)$ con el uso de la ecuación 4.27, involucra operaciones del orden $2T \cdot N^T$; debido a que, en cada $t = 1, 2, \cdots, T$ existen N posibles estado

que se pueden alcanzar, es decir, hay N^T posibles secuencias de estados, y para cada una de ellas, se requieren calcular $2T$ términos en la sumatoria dada por 4.27. Estos cálculos son computacionalmente imposibles, incluso para pequeños valores de N y T. Existe un procedimiento llamado *Forward-Backward*[2] que soluciona este problema.

El procedimiento *hacia adelante* (*forward*)

Considere la variable *hacia adelante* (*forward*) $\alpha_t(i)$ definida como

$$\alpha_t(i) = P(O_1 O_2 \cdots O_t,\, q_t = S_i|\lambda) \tag{4.28}$$

es decir, la probabilidad de la secuencia de observación parcial, $O_1 O_2 \cdots O_t$, (hasta el tiempo t) y el estado S_i en el tiempo t, dado el modelo λ. Se puede resolver $\alpha_t(i)$ de forma inductiva, de la siguiente forma:

1. Inicialización:

$$\alpha_1(i) = \pi_i b_i(O_1), \qquad \text{Para } i = 1, 2, \cdots, N \tag{4.29}$$

2. Inducción:

$$\alpha_{t+1}(j) = \Big[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\Big]\, b_j(O_{t+1}), \quad \begin{aligned} j &= 1, 2, \cdots, N \\ t &= 1, 2, \cdots, T-1 \end{aligned} \tag{4.30}$$

3. Terminación:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i). \tag{4.31}$$

El paso (1) inicializa las probabilidades hacia adelante como la probabilidad conjunta del estado S_i y la observación inicial O_1. El paso (2), *inducción*, es el corazón del cálculo; la figura 4.4(a) muestra como el estado S_j puede ser alcanzado en el tiempo $t+1$, a través de los N posibles estados S_i, donde $1 \leq i \leq N$, en el tiempo t. Debido a que $\alpha_t(i)$ es la probabilidad del evento conjunto donde $O_1 O_2 \cdots O_t$ son observados, y el estado en el tiempo t es S_i; el producto $\alpha_t(i) a_{ij}$ es entonces, la probabilidad de que el evento conjunto $O_1 O_2 \cdots O_t$ sea observado, y el estado S_j es alcanzado en el tiempo $t+1$ por conducto del estado

[2] Estrictamente hablando, sólo es necesario la parte *forward* del procedimiento para resolver el problema 1. Se introduce, en esta sección, la parte *backward* [Baum Egon 67] [Baum Sell 68] ya que se usará para resolver el problema 3

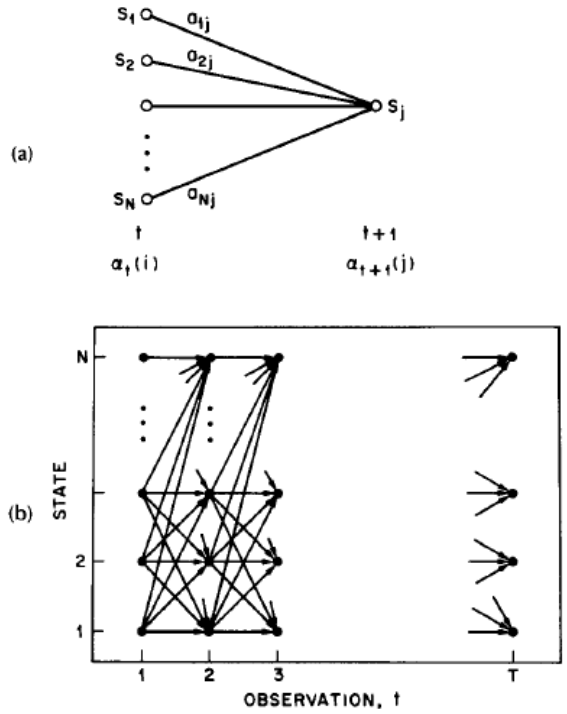


Figura 4.4: (a) Secuencia de operaciones requeridas para el cálculo de la variable $\alpha_{t+1}(j)$. (b) Implementación de los cálculos de $\alpha_t(j)$ en términos de una rejilla (*lattice*) de observaciones t y estados j.

S_i en el tiempo t. Sumando este producto sobre todos los N posibles estados S_i, $1 \leq i \leq N$ en el tiempo t, resulta en la probabilidad de S_j en el tiempo $t+1$ con todo el acompañamiento previo de las observaciones parciales. Una vez que esto es realizado y S_j es conocido, es fácil de ver que $\alpha_{t+1}(j)$ es obtenido calculando para la observación O_{t+1} en el estado j, es decir, multiplicando la cantidad sumada por la probabilidad $b_j(O_{t+1})$. El cálculo para la ecuación 4.30 es ejecutado para todos los estados j, donde $1 \leq i \leq N$, en un dado tiempo t; el cálculo se realiza para $t = 1, 2, \ldots, T-1$. Finalmente, en el paso 3 se obtiene el cálculo de $P(O|\lambda)$, sumando la variable terminal hacia delante (*forward*) $\alpha_T(i)$. Se sabe por definición que:

$$\alpha_T(i) = P(O_1 O_2 \cdots O_T, q_T = S_i|\lambda) \tag{4.32}$$

por tanto $P(O|\lambda)$ es sólo la suma de las $\alpha_T(i)$.

El procedimiento *hacia atrás* (*backward*)

De igual forma se puede considerar la variable *hacia atrás* (*backward*) $\beta_t(i)$ definida como

$$\beta_t(i) = P(O_{t+1}O_{t+2}\cdots O_T|q_t = S_i,\, \lambda) \tag{4.33}$$

es decir, la probabilidad de la secuencia de observación parcial de $t+1$ hasta el final, dado el estado S_i en el tiempo t y el modelo λ. Se puede calcular $\beta_t(i)$ de forma inductiva, de la siguiente forma:

1. Inicialización:
$$\beta_T(i) = 1, \qquad \text{Para } \; i = 1, 2, \cdots, N \tag{4.34}$$

2. Inducción:
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \quad \begin{aligned} t &= T-1, T-2, \cdots, 1 \\ i &= 1, 2, \cdots, N \end{aligned} \tag{4.35}$$

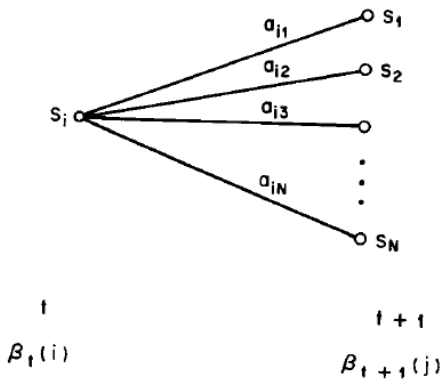


Figura 4.5: Secuencia de operaciones requeridas para el cálculo de la variable *hacia atrás* $\beta_t(i)$.

En el paso (1) establece $\beta_T(i)$ a 1 para todas las i. En el paso (2), el cual se ilustra en la figura 4.5, muestra que para estar en el estado S_i en el tiempo t y considerando la secuencia de observación en el tiempo $t+1$ hacia adelante, se toman todos los posibles estados S_j en el tiempo $t+1$, tomando en cuenta la transición del estado S_i al estado S_j (el término a_{ij}), así como la observación O_{t+1} en el estado j (el término $b_j(O_{t+1})$) y entonces se considera para la secuencia de observación parcial restante, desde el estado j (el término $\beta_{t+1}(i)$).

4.2.5. Solución al problema 2 -Secuencia de estados "óptima"

A diferencia del problema (1) para el cual se pude dar una solución exacta, existen varias formas para resolver el problema (2), esto es, buscar las secuencia de estados "óptima" asociada con la secuencia de observación. La dificultad radica con la definición de la secuencia de estados óptima; es decir, existen varios posibles criterios de optimización. Por ejemplo; un posible criterio de optimización es elegir los estados q_t que son *individualmente* más probables en cada tiempo t. Este criterio de optimización, maximiza el número esperado de los estados individuales correctos.

Aunque este método maximiza el estado más probable en cada t, se podrían tener algunos problemas con la secuencia de estados resultante. Por ejemplo, cuando el HMM tiene transiciones de estado con probabilidad cero ($a_{ij} = 0$ para alguna i y j), la secuencia de estados "optima" puede ser, incluso, una secuencia de estados no valida. Esto es debido a que la solución simplemente determina el estado más probable en cada instante, sin considerar la probabilidad de ocurrencia de la secuencia de estados.

Una mejor solución al problema de arriba, es encontrar la mejor secuencia de estados *única*, es decir, para maximizar $P(Q|O,\lambda)$ es equivalente a maximizar $P(Q,O|\lambda)$. Existe una técnica formal para encontrar esta secuencia de estados única, basada en métodos de programación dinámica y se llama *algoritmo de Viterbi* [Viterbi 67][Forney 73].

El algoritmo de Viterbi

Para encontrar la secuencia de estados única, $Q = \{q_1 q_2 \cdots q_T\}$, para la secuencia de observación dada $O = \{O_1 O_2 \cdots O_T\}$, se necesita definir la cantidad:

$$\delta_t(i) = \max_{q_1 q_2 \cdots q_{t-1}} P[q_1 q_2 \cdots q_{t-1}, q_t = S_i, O_1 O_2 \cdots O_t|\lambda] \tag{4.36}$$

Es decir, $\delta_t(i)$ es el mejor puntaje (máxima probabilidad) a lo largo de una trayectoria o camino, en el tiempo t, que es contabilizado para las primeras t observaciones y finaliza en el estado S_i. Por inducción se tiene

$$\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] \cdot b_j(O_{t+1}) \tag{4.37}$$

Para recuperar la secuencia de estados, se necesita almacenar la pista del argumento que maximiza 4.37, para cada t y j. Se puede hacer esto por medio

del arreglo $\psi_t(j)$. El procedimiento completo para encontrar la mejor secuencia de estados se muestra a continuación

1. Inicialización:

$$\delta_1(i) = \pi_i b_i(O_1), \qquad \text{Para } i = 1, 2, \cdots, N \tag{4.38}$$
$$\psi_1(i) = 0. \tag{4.39}$$

2. Recursión:

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] b_j(O_t), \quad \begin{matrix} 2 \le t \le T \\ 1 \le j \le N \end{matrix} \tag{4.40}$$

$$\psi_t(j) = \arg \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}], \quad \begin{matrix} 2 \le t \le T \\ 1 \le j \le N \end{matrix} \tag{4.41}$$

3. Terminación:

$$p* = \max_{1 \le i \le N} [\delta_T(i)] \tag{4.42}$$
$$q_T* = \arg \max_{1 \le i \le N} [\delta_T(i)]. \tag{4.43}$$

4. Recuperación de la trayectoria (secuencia de estados):

$$q_T* = \psi_{t+1}(q_{t+1}*), \qquad \text{Para } t = T-1, T-2, \ldots, 1. \tag{4.44}$$

Se puede observar que el algoritmo de Viterbi es similar (excepto por la recuperación de la trayectoria) en la implementación del algoritmo *hacia adelante*. La mayor diferencia radica en que, el algoritmo de Viterbi utiliza una maximización en el paso de recursión (ecu. 4.40), mientras que el algoritmo *hacia adelante* utiliza una sumatoria (ecu. 4.30). También es claro que la estructura de enrejado (*lattice*) se implementa eficientemente con el procedimiento de Viterbi.

4.2.6. Solución al problema 3 -Estimación de parámetros

El tercer problema de los HMM, y hasta ahora el más difícil, es determinar un método que ajuste los parámetros del modelo $(\mathbf{A}, \mathbf{B}, \pi)$ para maximizar la probabilidad de la secuencia de observación, dado el modelo. No existe alguna forma analítica conocida, para resolver esta maximización; de hecho, dado cualquier secuencia de observación finita de datos de entrenamiento, no hay alguna forma óptima de estimar los parámetros del modelo. Sin embargo, se puede elegir $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$, tal que $P(O|\lambda)$ sea maximizada localmente usando un procedimiento alternativo, tal como el método de *Baum-Welch*, también conocido como método EM (maximización del valor esperado) [Dempster Laird Rubin 77].

Para describir el procedimiento, es necesario definir una nueva variable $\xi_t(i,j)$, la probabilidad de estar en el estado i en el tiempo t, y en el estado j en el tiempo $t+1$, dado el modelo y la secuencia de observación, es decir:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{4.45}$$

Utilizando la definición de las variables *hacia adelante* (*forward*) y *hacia atrás* (*backward*), se puede escribir $\xi_t(i,j)$ en la forma:

$$\begin{aligned} \xi_t(i,j) &= \frac{P(q_t = S_i, q_{t+1} = S_j, O|\lambda)}{P(O|\lambda)} \\ \xi_t(i,j) &= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \\ \xi_t(i,j) &= \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \end{aligned} \tag{4.46}$$

Esta ecuación se puede interpretar mejor usando la figura 4.6.

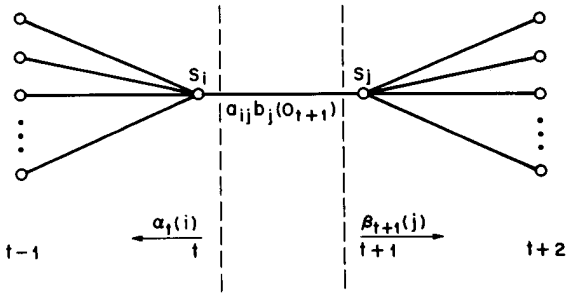


Figura 4.6: Operaciones requeridas para el cálculo de $\xi_t(i,j)$, es decir, la probabilidad de estar en el tiempo t en el estado i y en el tiempo $t+1$ en el estado j.

Así también, se puede definir la variable de probabilidad aposteriori

$$\gamma_t(i) = P(q_t = S_i | O, \lambda) \tag{4.47}$$

es decir, la probabilidad de estar en el estado S_i en el tiempo t, dada la secuencia de observación O y el modelo λ. La ecuación 4.47 puede expresarse en términos de las variables *hacia adelante* y *hacia atrás* (*forward-backward*),

de la siguiente forma

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{4.48}$$

Debido a que $\alpha_t(i)$ contabiliza la secuencia de observación parcial $O_1 O_2 \cdots O_t$ y el estado S_i en t, mientras que $\beta_t(i)$ contabiliza la secuencia de observación restante $O_{t+1} O_{t+2} \cdots O_T$, dado el estado S_i en t. El factor de normalización $P(O|\lambda) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i)$ hace que $\gamma_t(i)$ sea una medida de probabilidad y por tanto

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \tag{4.49}$$

Por otra lado se puede mejorar, de forma iterativa, los parámetros del modelo HMM $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$, por medio de, maximizar la probabilidad $P(O|\lambda)$ en cada iteración. Para realizar esto, es necesario definir la función-Q auxiliar,

$$Q(\lambda, \overline{\lambda}) = \sum_{\forall Q} \frac{P(O, Q|\lambda)}{P(O|\lambda)} \log P(O, Q|\overline{\lambda}) \tag{4.50}$$

donde $\overline{\lambda} = (\overline{\mathbf{A}}, \overline{\mathbf{B}}, \overline{\pi})$ representa al nuevo modelo estimado. Por otro lado $P(O, Q|\lambda)$ y $\log P(O, Q|\overline{\lambda})$ pueden ser expresados como:

$$P(O, Q|\lambda) = \pi_{q_0} \prod_{t=1}^{T} a_{q_{t-1} q_t} b_{q_t}(O_t) \tag{4.51}$$

$$\log P(O, Q|\lambda) = \log \pi_{q_0} + \sum_{t=1}^{T} \log a_{q_{t-1} q_t} + \sum_{t=1}^{T} \log b_{q_t}(O_t) \tag{4.52}$$

de esta forma, se puede reescribir la ecuación 4.50 como

$$Q(\lambda, \overline{\lambda}) = Q_\pi(\lambda, \overline{\pi}) + \sum_{i=1}^{N} Q_{a_i}(\lambda, \overline{\mathbf{a}}_i) + \sum_{i=1}^{N} Q_{b_i}(\lambda, \overline{\mathbf{b}}_i) \tag{4.53}$$

donde

$$\overline{\pi} = [\overline{\pi}_1, \overline{\pi}_2, \ldots, \overline{\pi}_N],$$
$$\overline{\mathbf{a}}_i = [\overline{a}_{i1}, \overline{a}_{i2}, \ldots, \overline{a}_{iN}],$$

$\overline{\mathbf{b}}_i$ es un vector con parámetros definido como $b_i(\cdot)$

y

$$Q_\pi(\lambda, \overline{\pi}) = \sum_{i=1}^{N} P(O, q_0 = S_i|\lambda) \log \overline{\pi}_i \tag{4.54}$$

$$Q_{a_i}(\lambda, \overline{\mathbf{a}}_i) = \sum_{j=1}^{N} \sum_{t=1}^{T} P(O, q_{t-1} = S_i, q_t = S_j|\lambda) \log \overline{a}_{ij} \tag{4.55}$$

$$Q_{b_i}(\lambda, \overline{\mathbf{b}}_i) = \sum_{t=1}^{T} P(O, q_t = S_i|\lambda) \log \overline{b}_i(O_t) \tag{4.56}$$

Debido a que se ha separado la función-Q en tres términos independientes, el procedimiento de maximización de $Q(\lambda, \overline{\lambda})$ se puede realizar, maximizando cada término separadamente, sujeto a las restricciones de probabilidad

$$\sum_{j=1}^{N} \overline{\pi}_j = 1 \tag{4.57}$$

$$\sum_{j=1}^{N} \overline{a}_{ij} = 1, \quad \text{para }, j = 1, 2, \ldots, N \tag{4.58}$$

$$\sum_{k=1}^{N} \overline{b}_j(k) = 1 \quad \text{para }, j = 1, 2, \ldots, N. \tag{4.59}$$

$$\tag{4.60}$$

Dado que todas las funciones auxiliares individuales tiene la forma

$$\sum_{j=1}^{N} w_j \log y_j \tag{4.61}$$

sujeto a las restricciones $\sum_{j=1}^{N} y_j = 1$, $y_j \geq 0$. Utilizando los multiplicadores de *Lagrange*, la función 4.61 alcanza su máximo valor en

$$y_j = \frac{w_j}{\sum_{i=1}^{N} w_i}, \qquad \text{para }, j = 1, 2, \ldots, N \tag{4.62}$$

Utilizando esta función, se obtiene el modelo estimado $\overline{\lambda} = (\overline{\pi}, \overline{\mathbf{A}}, \overline{\mathbf{B}})$ para

$$\overline{\pi}_i = \frac{P(O, q_0 = S_i|\lambda)}{P(O|\lambda)} \tag{4.63}$$

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T} P(O, q_{t-1} = S_i, q_t = S_j|\lambda)}{\sum_{t=1}^{T} P(O, q_{t-1} = S_i|\lambda)} \tag{4.64}$$

$$\overline{b}_i(k) = \frac{\sum_{t=1}^{T} P(O, q_t = S_i|\lambda)\delta(O_t, o_k)}{\sum_{t=1}^{T} P(O, q_t = S_i|\lambda)} \tag{4.65}$$

donde se define

$$\delta(O_t, o_k) = \begin{cases} 1 & \text{si } O_t = o_k \\ 0 & \text{otro caso} \end{cases}$$

Si se utiliza la definición de la variable *hacia adelante*, $\alpha_t(i) = P(O_1, O_2, \cdots, O_t, q_t = S_i|\lambda)$ y la variable *hacia atrás* $\beta_t(i) = P(O_{t+1}, O_{t+2}, \cdots, O_T|q_t = S_i, \lambda)$, se puede escribir los parámetros estimados como

$$P(O, q_t = S_i|\lambda) = \alpha_t(i)\beta_t(i)$$

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_t(i)\beta_t(i) = \sum_{i=1}^{N} \alpha_T(i)$$

$$P(O, q_{t-1} = S_i, q_t = S_j|\lambda) = \alpha_{t-1}(i)a_{ij}b_j(O_t)\beta_t(j)$$

por tanto

$$\overline{\pi}_i = \frac{\alpha_0(i)\beta_0(i)}{\sum_{j=1}^{N}\alpha_T(j)} = \gamma_0(i) \tag{4.66}$$

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T}\alpha_{t-1}(i)a_{ij}b_j(O_t)\beta_t(j)}{\sum_{t=1}^{T}\alpha_{t-1}(i)\beta_{t-1}(i)} = \frac{\sum_{t=1}^{T}\xi_{t-1}(i,j)}{\sum_{t=1}^{T}\gamma_{t-1}(i)} \tag{4.67}$$

$$\overline{b}_i(k) = \frac{\sum_{t=1}^{T}\alpha_t(i)\beta_t(i)\delta(O_t,o_k)}{\sum_{t=1}^{T}\alpha_t(i)\beta_t(i)} = \frac{\sum_{t=1,\text{para } O_t=o_k}^{T}\gamma_t(i)}{\sum_{t=1}^{T}\gamma_t(i)} \tag{4.68}$$

Conforme al procedimiento EM, el algoritmo *forward-Backward* (*Baum-Welch*) garantiza un mejoramiento monótono de la verosimilitud (*likelihood*) en cada iteración, eventualmente converge a un máximo local. El algoritmo *forward-Backward* puede ser descrito en una forma similar al algoritmo EM, como se describe a continuación:

Paso 1 Inicialización: Elegir un estimación inicial del modelo λ.

Paso 2 Paso E: Calcular la función auxiliar $Q(\lambda, \overline{\lambda})$ basados en λ.

Paso 3 Paso M: Calcular $\overline{\lambda}$ basados en la estimación de las ecuaciones 4.66, 4.67 y 4.68, para maximizar la función-Q auxiliar.

Paso 4 Iteración: Establecer $\lambda = \overline{\lambda}$, repita desde el paso 2 hasta que converja el algoritmo.

4.2.7. Observaciones con densidades continuas en HMM

Hasta el momento, se han considerado sólo observaciones que se caracterizan con símbolos discretos de un alfabeto finito, y por tanto, se utilizan densidades de probabilidad discretas dentro de cada estado en el modelo. El problema con esta aproximación es que, en la mayoría de las aplicaciones, las observaciones son señales (o vectores) continuas. Aunque es posible convertir la representación de señales continuas en una secuencia discreta de símbolos, utilizando cuantización vectorial u otros métodos, podría tener una severa degradación asociada a la discretización de la señal continua.

Para usar observaciones con densidades continuas, se deben establecer algunas restricciones en la forma de funciones de densidad de probabilidad (fdp), para asegurar que los parámetros de la fdp pueden ser estimados de manera consistente. La representación mas general de la fdp, para el cual se ha formulado un algoritmo de re-estimación, es una combinación finita de densidades de la forma:

$$b_j(O) = \sum_{k=1}^{M} c_{jk} \mathcal{N}(O, \mu_{jk}, \mathbf{U}_{jk}) \tag{4.69}$$

donde O es el vector de observación siendo modelado, c_{jk} es el coeficiente de combinación (*mixture coefficient*) para la k-ésima densidad en el estado j y $\mathcal{N}$ es cualquier densidad simétrica elíptica o cóncava-logarítmica [Levinson, Rabiner 83], por ejemplo, una Gaussiana. Sin pérdida de generalidad se asume que $\mathcal{N}$ es una Gaussiana, con media μ_{jk} y matriz de covarianza $\mathbf{U}_{jk}$ para el k-ésimo componente en el estado j. Los coeficientes de combinación c_{jk} satisfacen las restricciones estocásticas

$$\sum_{k=1}^{M} c_{jk} = 1, \quad \text{para }, j = 1, 2, \ldots, N \tag{4.70}$$

$$c_{jk} \geq 0, \quad \text{para } 1 \leq j \leq N,\ 1 \leq k \leq M, \tag{4.71}$$

de tal forma que la fdp es normalizada apropiadamente,

$$\int_{-\infty}^{\infty} b_j(O) dO = 1, \quad \text{para } 1 \leq j \leq N \tag{4.72}$$

La figura 4.7 muestra, que un estado HMM con una combinación de densidades, es equivalente a un modelo multi-estado con una sola densidad [Juang Levinson Sondhi 86].

Se puede demostrar que las formulas de re-estimación para los coeficientes de densidades combinadas (c_{jk}, μ_{jk} y $\mathbf{U}_{jk}$), son

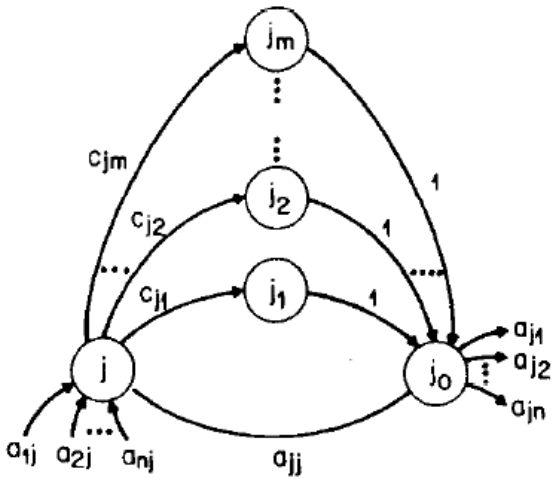


Figura 4.7: Equivalencia de un estado con densidades múltiples, a un multi-estado con una sola distribución de densidad.

$$\overline{c}_{jk} = \frac{\sum_{t=1}^{T} \rho_t(j,k)\beta_t(j)}{\sum_{t=1}^{T} \alpha_t(j)\beta_t(j)}, \tag{4.73}$$

$$\overline{\mu}_{jk} = \frac{\sum_{t=1}^{T} \rho_t(j,k)\beta_t(j)O_t}{\sum_{t=1}^{T} \rho_t(j,k)\beta_t(j)}, \tag{4.74}$$

$$\overline{U}_{jk} = \frac{\sum_{t=1}^{T} \rho_t(j,k)\beta_t(j)(O_t-\mu_{jk})(O_t-\mu_{jk})^T}{\sum_{t=1}^{T} \rho_t(j,k)\beta_t(j)} \tag{4.75}$$

Para $1 \leq i, j \leq N$ y $1 \leq k \leq M$, donde

$$\rho_t(j,k) = \begin{cases} c_{jk}\dfrac{\partial b_j}{\partial c_{jk}}\Big|_{O_t} & \text{para } t = 1 \\ \displaystyle\sum_{i=1}^{N} \alpha_{t-1}(i)a_{ij}c_{jk}\frac{\partial b_j}{\partial c_{jk}}\Big|_{O_t} & \text{para } 1 < t \leq T \end{cases}$$

4.2.8. Solución al problema de precisión -Escalamiento

Para realizar el procedimiento de re-estimación de los modelos HMM, es necesario realizar operaciones como la variable *hacia adelante* $\alpha_t(i)$, que consiste en sumas de un gran número de términos, cada uno de la forma,

$$\left(\prod_{s=1}^{t-1} a_{q_s a_{s+1}} \prod_{s=1}^{t} b_{q_s}(O_s)\right)$$

Debido a que cada término a y b es menor o igual a uno, y además, t por lo general es un valor grande (por ejemplo, $t = 100$ ó más), los valores de los términos $\alpha_t(i)$ tienden a cero exponencialmente conforme se incrementa el número de observaciones. Si se desea calcular estos valores en una computadora cualquiera, excederán el rango de precisión para cualquier máquina (incluso, si se utiliza con doble precisión). Por lo que, la única forma de realizar los cálculos, es incorporar un procedimiento de escalamiento.

El procedimiento básico de escalamiento, multiplica $\alpha_t(i)$ por un coeficiente independiente de i (este sólo depende de t). De la misma forma, es necesario un escalamiento para los coeficientes $\beta_t(i)$ (ya que, también, tienden a cero exponencialmente).

Considerando la fórmula de re-estimación para los coeficientes de transición de estados a_{ij} en el algoritmo *forward-backward*, dada por

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}, \tag{4.76}$$

Además, si se considera $\hat{\alpha}$ como las αs escaladas y $\hat{\hat{\alpha}}$ como una versión local de α antes del escalamiento; entonces, el procedimiento para la re-estimación de los coeficientes de transición a_{ij} es [Rabiner 93]:

- Inicialización, para $t = 1$, se calcula $\alpha_1(i)$ usando la ecuación 4.29 y se calcula los siguiente valores

$$\hat{\hat{\alpha}}_1(i) = \alpha_1(i), \quad c_1 = \frac{1}{\sum_{i=1}^{N} \alpha_1(i)}, \quad \hat{\alpha}_1(i) = c_1 \alpha_1(i)$$

- Para cada t, donde $2 \leq t \leq T$, se calcula $\hat{\hat{\alpha}}_t(i)$ conforme a la ecuación 4.30 de la página 61, en términos de $\hat{\alpha}_t(i)$; es decir,

$$\hat{\hat{\alpha}}_t(i) = \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji} b_i(O_t) \tag{4.77}$$

se determina el coeficiente de escalamiento c_t como

$$c_t = \frac{1}{\sum_{i=1}^{N} \hat{\hat{\alpha}}_t(i)} \tag{4.78}$$

entonces se obtiene $\hat{\alpha}_t(i)$ como

$$\hat{\alpha}_t(i) = c_t \hat{\hat{\alpha}}_t(i) \tag{4.79}$$

de la ecuaciones 4.77 a 4.79, se puede escribir $\hat{\alpha}_t(i)$ como

$$\hat{\alpha}_t(i) = \frac{\sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji} b_i(O_t)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji} b_i(O_t)} \tag{4.80}$$

- Cálculo del término $\beta_t(i)$ del la recursión *hacia atrás*. Se utiliza el *mismo* factor de escalamiento c_t para el cálculo de $\beta_t(i)$, de la siguiente forma

$$\hat{\beta}_t(i) = c_t \beta_t(i) \tag{4.81}$$

- Calculo de los coeficientes de transición de estados a_{ij}

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j)}, \tag{4.82}$$

El mismo procedimiento de escalamiento se aplica para la re-estimación de los coeficientes π o $\mathbf{B}$. En el caso del cálculo de $P(O|\lambda)$, se tiene

$$\prod_{t=1}^{T} c_t \sum_{i=1}^{N} \alpha_T(i) = C_T \sum_{i=1}^{N} \alpha_T(i) = 1 \tag{4.83}$$

donde

$$\prod_{t=1}^{T} c_t = C_T \tag{4.84}$$

de esta forma, tenemos

$$\prod_{t=1}^{T} c_t P(O|\lambda) = 1 \tag{4.85}$$

o

$$P(O|\lambda) = \frac{1}{\prod_{t=1}^{T} c_t} \tag{4.86}$$

o

$$\log[P(O|\lambda)] = \sum_{t=1}^{T} \log c_t \tag{4.87}$$

de esta manera, se puede calcular el logaritmo de P, pero no P, ya que este podría estar fuera del rango dinámico de la máquina en cualquier caso.

4.2.9. Múltiples secuencias de observación

Aunque el algoritmo descrito en la sección 4.2.6, está basado en una secuencia de observación, este se puede generalizar fácilmente para múltiples secuencias de observación de entrenamiento. La modificación del procedimiento de re-estimación es directa y es la siguiente.

Se denotará al conjunto de K secuencias de observación como:

$$\mathbf{O} = [O^{(1)}, O^{(2)}, \cdots, O^{(K)}] \tag{4.88}$$

donde $O^{(k)} = (O_1^k, O_2^k, \cdots, O_{T_k}^k)$ es la k-ésima secuencia de observación. Se hace la suposición de que cada secuencia es independiente de cada una de las

otras secuencias de observación. Como la meta principal es ajustar los parámetros del modelo a maximizar, entonces

$$P(\mathbf{O}|\lambda) = \prod_{k=1}^{K} P(O^{(k)}|\lambda) \tag{4.89}$$

$$= \prod_{k=1}^{K} P_k. \tag{4.90}$$

Debido a que las fórmulas de re-estimación están basadas en la frecuencia de ocurrencias de varios eventos, las fórmulas de re-estimación para múltiples secuencias de observación se modifican, agregando las frecuencias individuales de ocurrencia para cada secuencia. Por lo tanto, las fórmulas de re-estimación de $\overline{a}_{ij}$ y $\overline{b}_j(l)$ son:

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) a_{ij} b_j(O_{t+1}^k) \beta_{t+1}^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \tag{4.91}$$

$$\overline{b}_i(\ell) = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i) \delta(O_t, o_\ell)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \alpha_t^k(i) \beta_t^k(i)} \tag{4.92}$$

π no es estimado debido a que $\pi_1 = 1$ y $\pi_i = 0$, para $i \neq 1$.

El escalamiento de las ecuaciones 4.91 y 4.92 es fácil de hacer, ya que cada secuencia de observación tiene su propio factor de escalamiento. La idea clave es eliminar el factor de escalamiento de cada término, antes de sumarlos. Esto se puede realizar, escribiendo las ecuaciones de re-estimación en términos de las

variables escaladas, es decir:

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) a_{ij} b_j(O_{t+1}^k) \hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)} \tag{4.93}$$

$$\overline{b}_i(\ell) = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i) \delta(O_t, o_\ell)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)} \tag{4.94}$$

Así, para cada secuencia $O^{(k)}$, aparecerá el mismo factor de escalamiento en cada suma sobre t como aparece en el término P_k y por tanto, se cancelará. De esta forma, usando los valores escalados de las αs y βs, resulta en un valor de $\overline{a}_{ij}$ no escalada.

Capítulo 5

Modelo acústico

El propósito del modelo acústico es proveer un método para calcular la verosimilitud de cualquier secuencia de vectores de observación O, dado una palabra w. El modelado acústico juega un papel muy importante en el mejoramiento de la exactitud, y es sin duda, la parte central en cualquier sistema de reconocimiento de voz. Lo constituye: la *extracción de características*, en el cual la señal de voz es procesada para generar un conjunto de vectores que describen a la señal, frecuentemente en términos de la envolvente espectral; las *unidades de reconocimiento*, que describe la unidad a reconocer (fonemas, trifonemas, sílabas ó palabras completas); y la *representación específica de cada unidad*, que determina la manera de representar cada unidad mediante HMM.

5.1. Extracción de características

Como ya se ha descrito en el capítulo 3, la etapa de extracción de características, procesa la señal de estrada para producir un conjunto de vectores que contienen información: acerca de la identidad del hablante, el genero, las características del tracto vocal, el acento, la edad, así como la información del entorno acústico y el canal en que se transporta la voz. Se ha demostrado que la mejor representación de la señal de voz, para un sistema de reconocimiento de voz, son los vectores Mel-Cepstral (MFCC).

5.2. Unidades de reconocimiento

Existen varias formas de representar las palabras con unidades de reconocimiento, en las cuales se encuentran, la representación por medio de fonemas, difonemas, trifonemas, sílabas e incluso toda la palabra. Sin embargo existen algunas ventajas y desventajas para el uso de ellas, dependiendo del sistema de reconocimiento de voz. Por ejemplo, para un sistema de reconocimiento de voz de palabras aisladas y sistemas de reconocimiento de voz para palabras conectadas, es recomendable usar modelos que contemplen toda la palabra, debido a que se encuentra bien defina su representación acústica y la variabilidad acústica ocurre principalmente en la región del inicio y fin de la palabra; otra ventaja es que se evita el léxico de la palabra, haciendo la estructura del reconocimiento muy simple [Rabiner 93].

Sin embargo, para sistemas de reconocimiento de voz continua, el uso de palabras como unidad de reconocimiento, no es apropiado, debido a que se necesitan una gran cantidad de palabras a modelar, así como el número de pronunciaciones de cada una de ellas en el conjunto de entrenamiento. Una mejor forma de modelar a las unidades de reconocimiento, en los sistemas de reconocimiento de voz continua, es con el uso de fonemas, en los cuales sólo se necesitan unas cuantas unidades (alrededor de 40 a 50 fonemas). Sin embargo, con esta representación, no se contempla el contexto de los fonemas.

Con la representación de trifonemas como unidades de reconocimiento, para los sistemas de reconocimiento de voz continua, mejora la exactitud, sin embargo, se presenta otro problema para los trifonemas no vistos en el entrenamiento y que aparecen en el reconocimiento. Esto involucra la necesidad de usar técnicas para la solución de la insuficiencia de datos, como por ejemplo, *Back-off* que combina la representación de otros modelos (difonemas y fonemas).

5.3. Modelado de las unidades de reconocimiento basado en HMM

La forma mas popular de modelar la voz es, con los modelos ocultos de Markov. La figura 5.1 (a), representa el modelo de una palabra con el uso de HMM, de N estados y donde N puede ser un valor fijo (de 5 a 10 para cada palabra). La figura 5.1 (b) muestra la representación de los modelos HAM para un fonemas (difonema o trifonemas), en el cual, el número de estados puede ser variable con el número de sonidos (fonemas) en la palabra, o puede ser establecido el número promedio de tramas en la palabra.

Para la estimación de las unidades de reconocimiento, es necesario un con-

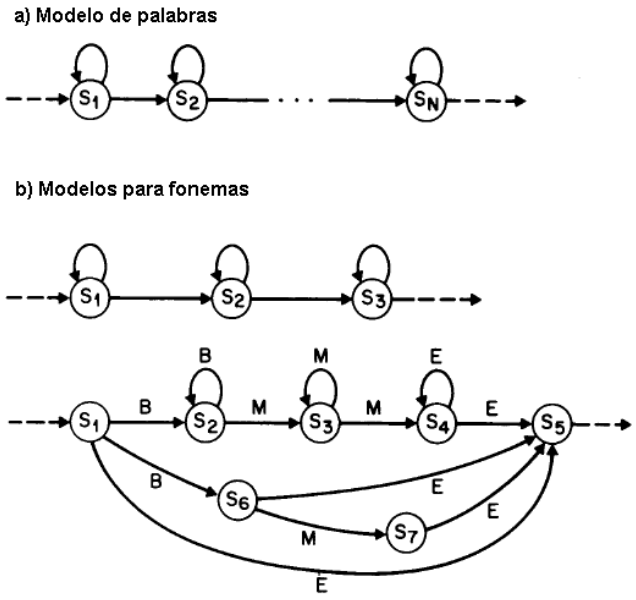


Figura 5.1: Representación de HMM para modelar una palabra (a) y el modelo de fonemas (b).

junto de sentencias de entrenamiento, en donde cada sentencia consiste de la señal de voz y su transcripción en palabras. Además, se debe contar con un diccionario del conjunto de entrenamiento, para la transcripción de cada palabra con las unidades de entrenamiento correspondientes; por ejemplo, para la palabra "puerta", se tiene una transcripción en sus fonemas como /p/, /u/, /e/, /r/, /t/, /a/. Se asume que el silencio puede (no necesariamente) preceder o seguir de una palabra dentro de la sentencia, es decir, con un silencio en el inicio y fin de cada sentencia. Basados en esta suposición, una sentencia típica en el conjunto de entrenamiento se puede transcribir como:

$$S_W : W_1 W_2 W_3 \dots W_L$$

en donde cada W_i, $1 \leq i \leq L$ es una palabra en el diccionario. Por tanto la sentencia "Hola a todos", es una sentencia de tres palabras $W_1 =$ "Hola", $W_2 =$ a, $W_3 =$ "todos". Cada palabra puede ser transcrita en unidades de entrenamiento. Por lo tanto, la sentencia S puede ser escrita en sus unidades de sub-palabras

como

$$\begin{aligned} S_W : U_1(W_1)U_2(W_1)U_3(W_1)\ldots U_{Tam(W_1)}(W_1) \quad \oplus \\ U_1(W_2)U_2(W_2)U_3(W_2)\ldots U_{Tam(W_2)}(W_2) \quad \oplus \\ \cdots \oplus U_1(W_L)U_2(W_L)U_3(W_L)\ldots U_{Tam(W_L)}(W_L) \end{aligned} \tag{5.1}$$

donde $Tam(W_i)$ es la longitud (en unidades) de la palabra W_i. Finalmente, se reemplaza cada unidad de sub-palabra por su HMM (el modelo de tres estados, como se muestra la figura 5.1) y se incorpora la suposición de un silencio entre palabras, para obtener un HMM extendido en cada sentencia.

El proceso descrito previamente se muestra en la figura 5.2, para la sentencia particular "Hola a todos". Se pude observar que una sentencia es representada como una red de estados finitos (FSN) donde los arcos son: palabras, silencio o los arcos nulos (donde la transición nula ϕ es requerida para saltar el silencio alternativo). Cada palabra es representada como un FSN de unidades de sub-palabra, y cada unidad de sub-palabra es representada como un HMM de tres estados, la figura 5.3 muestra un modelo HMM de tres estado de un fonema, dada una secuencia de vectores acústicos $\mathbf{Y}$.

Una característica importante en la implementación de la figura 5.2, es que se usa un HMM de un sólo estado para modelar el silencio. Esto se usa debido a que, el silencio es generalmente estacionario y no tiene una estructura temporal para explotar.

Una vez que se crea una sentencia compuesta de FSN, para cada frase en el conjunto de entrenamiento, el problema de entrenamiento se convierte en la estimación de parámetros para los modelos de las unidades de sub-palabras, que maximice la verosimilitud (*maximum likelihood*) de los modelos, de todos los datos de entrenamiento. La máxima verosimilitud de los parámetros puede ser encontrada usando cualquiera de los siguientes procedimiento: el procedimiento *forward-backward* o el algoritmo de entrenamiento segmental *K-Medias*.

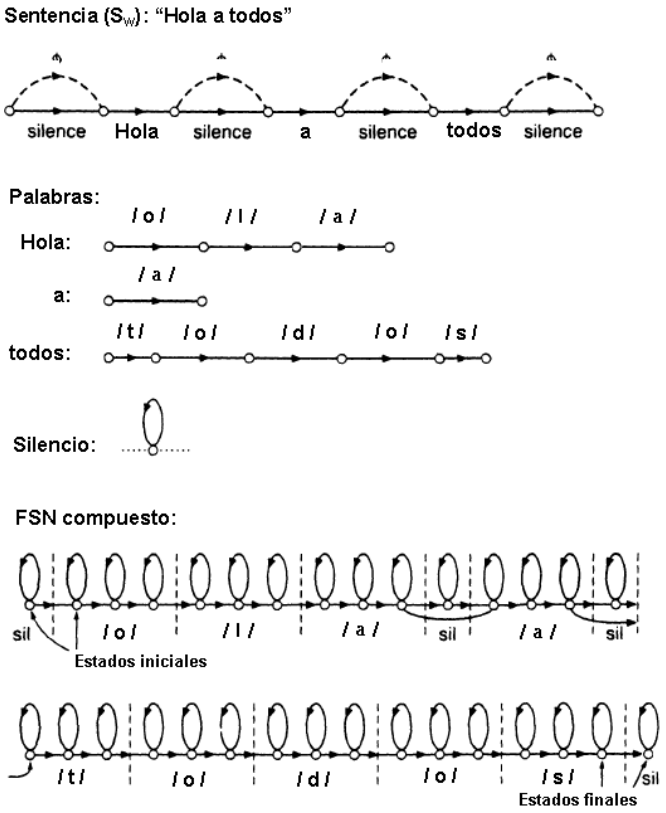


Figura 5.2: Creación de un FSN para la secuencia "Hola a todos".

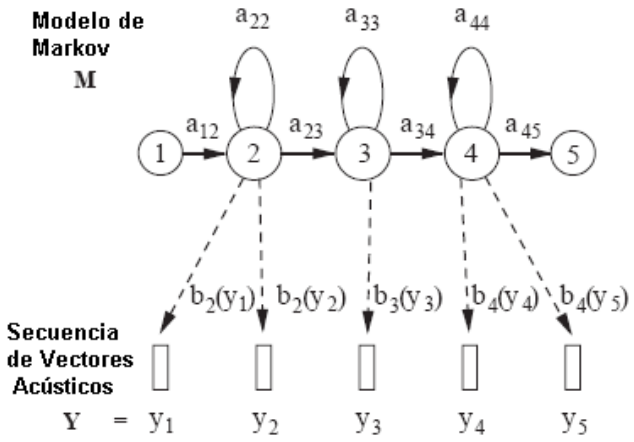


Figura 5.3: Modelo de un fonema basado en HMM.

Capítulo 6

Descripción de la tarjeta y las herr. de desarrollo

6.1. Introducción

Un procesador digital de señales (DSP) es un tipo de microprocesador que manipula los datos en tiempo real. Esta capacidad de tiempo real, hace que un DSP sea perfecto para aplicaciones que no pueden tolerar algún retardo. Por ejemplo, el DSP dentro de un teléfono celular, procesa el sonidos tan rápido que las personas que se encuentran hablando, no se percatan de que las señales del sonido son transformadas a señales digitales, filtradas y comprimidas para su transmisión, además de ser procesadas en sentido inverso para la recepción (Figura 6.1).

Figura 6.1: Representación gráfica del proceso de señales digitales en un teléfono celular.

Algunas de las ventajas del diseño con DSPs sobre otros microprocesadores

se listan a continuación:

- Operaciones de multiplicación y acumulación (MACs) en un sólo ciclo
- Ejecución en tiempo real, simulación y emulación
- Flexibilidad
- Confiable
- Incremento en el desempeño del sistema
- Reducción del costo del sistema

En la actualidad los DSPs se usan en varias áreas, como por ejemplo: en redes inalámbricas LAN, sistemas de telefonía multicanal, sistemas de seguridad en casas mediante el reconocimiento del rostro ó huella digital, realidad virtual gráficos 3D, reconocimiento de voz, audio, radar, teléfonos celulares, Modems inalámbricos, Cámaras digitales, Control de motores, DVDs, etc.

6.2. Características de la tarjeta de desarrollo

La tarjeta de desarrollo empleada a lo largo de esta tesis es el kit de iniciación del DSP C6711 (*C6711 DSP Starter Kit*, DSK), fabricada por *Texas Instruments*. La figura 6.2 muestra la tarjeta de desarrollo.

El DSK C6711 tiene las siguientes características:

- Contiene el DSP TMS320C6711, con una frecuencia de 150 MHz, capaz de ejecutar 900 millones de operaciones de punto flotante por segundo (MFLOPS)
- Soporta reloj dual: 150 MHz en el CPU y 100 MHz en la interface de memoria externa (EMIF)
- Interface de controlador de puerto paralelo (PPC), para puertos paralelos estándar en una PC (soporta EEP y bi-direccional SPP)
- Contiene un banco de memoria RAM dinámica síncrona (SDRAM) de 4M x 32-bits palabras, que opera a 100 MHz (10-ns)
- Tiene 128K bytes de memoria flash programable asíncrona (flash EPROM). Con esta memoria, el DSK puede iniciar automáticamente una aplicación cuando se enciende o se reinicia.

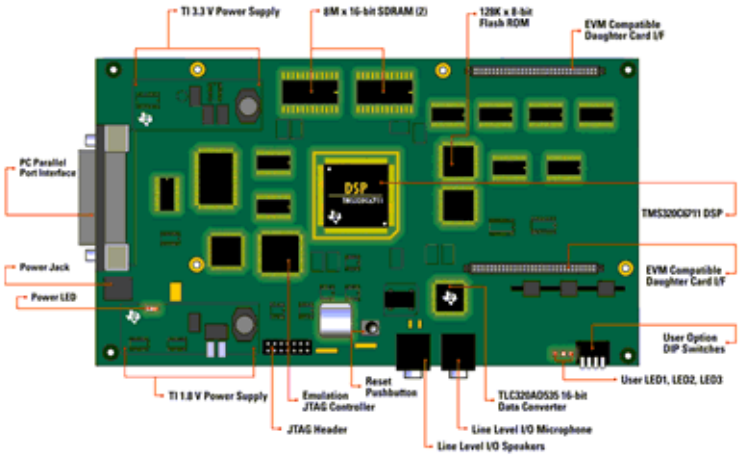


Figura 6.2: Tarjeta de desarrollo DSK C6711.

- Puerto de E/S mapeado en memoria de 8 bits
- Emulación JTAG embebida por medio del puerto paralelo y soporte externo XDS510
- Interface de puerto host (HPI) accede a toda la memoria del DSP por medio del puerto paralelo
- La tarjeta DSK incluye un *codec* de canal dual voz/datos TLC320AD535[1], que está conectado con el McBSP del DSP. Este circuito de interface analógica (AIC) tiene las siguientes características:
  - Procesamiento de señal de 16-bits
  - Canales independientes de voz y de datos (sólo el canal de voz es utilizado en el DSK)
  - Frecuencia máxima de muestreo de 11.025 KHz (la tarjeta mantiene fija la frecuencia de muestreo a 8 KHz)
  - Modo de ahorro de energía por software separado para los dos canales
  - Amplificadores de ganancia programable
  - Manejador de audífonos de 60 Ω con amplificador de ganancia programable
  - Soporte para micrófono *electret* con voltaje de polarización (*bias*)
- La tarjeta contiene dos conectores que permite acoplar una tarjeta de expansión (*daughtercard*). La tarjeta de expansión puede ser usada para

[1] Una descripción completa del *codec* se puede encontrar en la literatura [SLAS202B]

ampliar las capacidades del DSK y para proveer de una aplicación específica de E/S personalizada. A continuación se describe los dos conectores:

- El conector de la interfaz de expansión de memoria (J1) provee las señales de control EMIF asíncronas del DSP, y los buses de direcciones y de datos.
- El conector de la interfaz de expansión de periféricos (J3) proporciona el acceso a los periféricos del DSP y a sus señales de control/estado.

6.3. Características del DSP TMS320C6711

El DSP TMS320C6711[2] de *Texas Instruments*, forma parte de la familia de DSPs de punto flotante en la plataforma TMS320C6000. Los dispositivos C67x están basados en alto desempeño, arquitectura avanzada de palabras de instrucción muy largas (VLIW, *very-long-instruction-word*) desarrollado por *Texas Instruments* (TI).

Estos dispositivos (C67x) ejecutan mas de 900 millones de operaciones de punto flotante por segundo (MFLOPS), a una velocidad de reloj de 150 MHz. El DSP C6711/C6711B posee la flexibilidad operacional de controles de alta velocidad y la capacidad numérica de arreglos de procesadores. Este procesador tiene 32 registros de propósito general con un tamaño de palabra de 32-bits, además contiene ocho unidades funcionales muy independientes. Estas ocho unidades proporcionan cuatro ALUs de punto flotante/fijo, dos ALUs de punto fijo y dos multiplicadores de punto flotante/fijo. El DSP C6711/C6711B puede producir dos MACs por ciclo para un total de 300 MMAC/seg.

Los dispositivos C67x usan una arquitectura basada en caché de dos niveles. La caché de programa Nivel 1 (L1P) es una caché de 32-Kbits de mapeo directo, la caché de datos Nivel 1 (LID) es una caché asociativa de 32-Kbits de 2-vías. La memoria/caché Nivel 2 (L2) consiste de un espacio en memoria de 512-Kbits que es compartido entre espacio de datos y programa. La memoria L2 puede ser configurada como: memoria mapeada, caché o una combinación de ambas.

Para una mayor descripción de las características del DSP C6711, consultar la guía de referencia de Texas Instruments [SPRU733].

Además, estos dispositivos contiene un poderoso conjunto de periféricos, que incluyen dos puertos seriales con buffer y multicanal (McBSPs), dos timer de propósito general, una interface de puerto-host (HPI) y una interface de memoria externa (EMIF): unidad capaz de conectar a una SDRAM, SBSRAM

[2] En adelante se llamará de forma particular C6711, y de forma general TMS320C67x o C67x para denotar a todas las versiones.

y periféricos asíncronos. Una descripción más detallada de los periféricos del DSP C6711, se puede consultar en la guía de referencia de Texas Instruments [SPRU190D].

El DSP C6711 tiene un completo conjunto de herramientas de desarrollo que incluyen: un compilador de C, un optimizador de ensamblador para simplificar la programación y planeación, además de una interfaz de desarrollo integrada para el desarrollo de aplicaciones de DSPs en tiempo real. La figura 6.3 muestra un diagrama de bloques de la arquitectura de los dispositivos C67x/C62x. Para tener mayor información referente a estas herramientas, consultar las guías de usuario [SPRU328B], [SPRU423], [SPRU186I] y [SPRU198F].

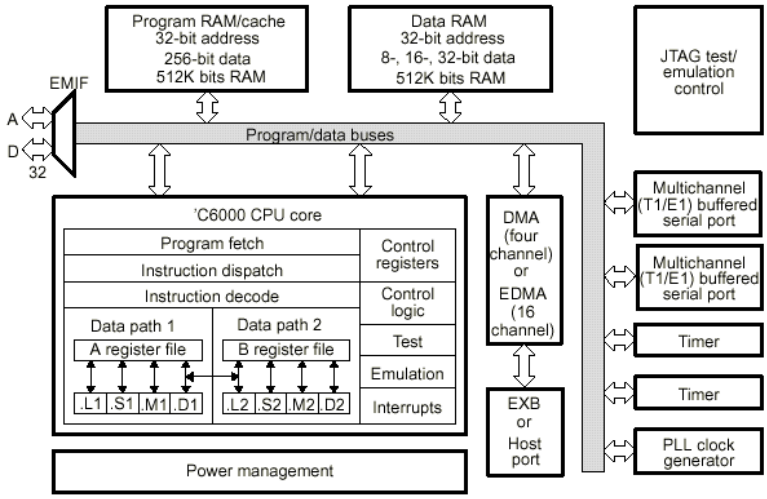


Figura 6.3: Arquitectura de los dispositivos TMS320C62x/C67x.

6.4. Descripción del CPU (núcleo del DSP)

El CPU reúne palabras de instrucciones muy extensas (VLIW) (256 bits de ancho) para suministrar ocho instrucciones de 32 bits para las ocho unidades funcionales durante cada ciclo de reloj.

Las unidades funcionales en el CPU se dividen en dos conjuntos (lado A y lado B), cada conjunto contiene cuatro unidades y una fila de registros. Un conjunto contiene las unidades funcionales .L1, .S1, .M1 y .D1; el otro conjunto contiene las unidades funcionales .D2, .M2, .S2 y .L2. Las dos filas de registros contienen cada una, 16 registros de 32-bits (A0-A15 para la fila A y B0-B15 para la fila B) que dan un total de 32 registros de propósito general (ver figura

6.4). Las cuatro unidades funcionales de cada lado del CPU, pueden compartir libremente los 16 registros correspondientes a ese lado. Además, cada lado ofrece un sólo bus de datos conectado a todos los registros del lado opuesto.

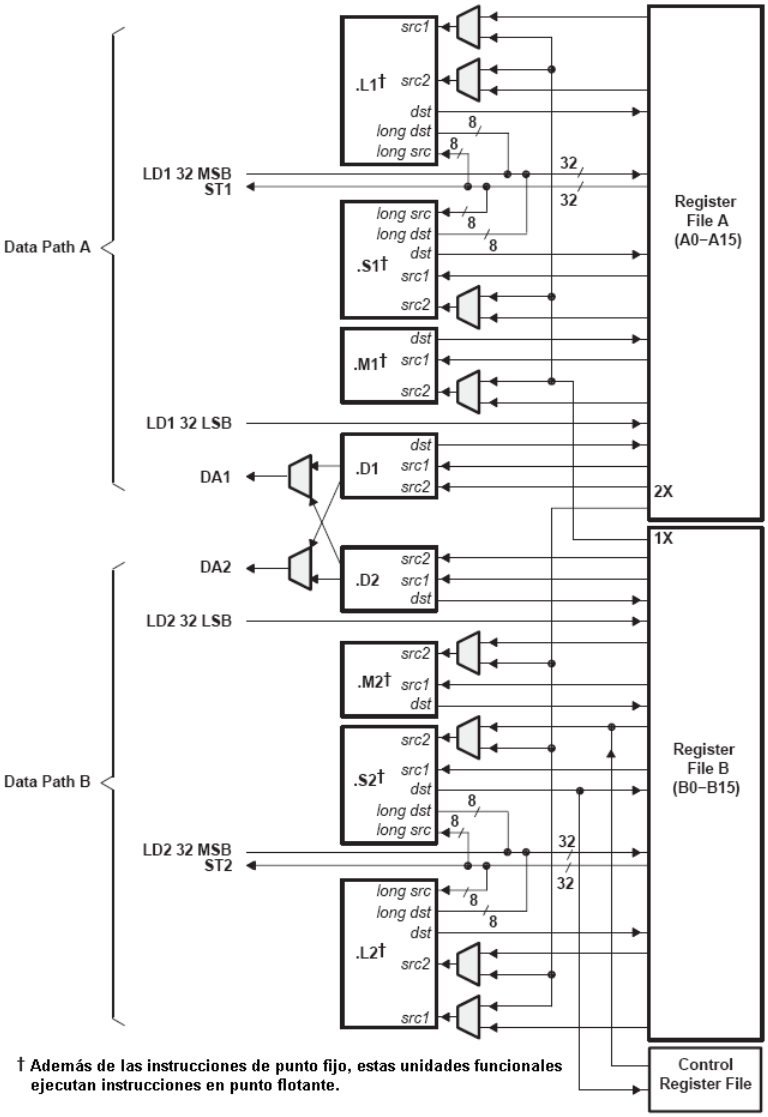


Figura 6.4: Trayectorias de los datos del CPU TMS320C67x (núcleo del DSP).

El CPU C67x ejecuta todas las instrucciones de los dispositivos C62x. Además de las instrucciones de punto fijo del C62x, seis de ocho unidades funcionales (.L1, .S1, .M1, .M2, .S2 y .L2) también ejecutan instrucciones de punto flotante. Las restantes unidades funcionales (.D1 y D2) también ejecutan la nueva

instrucción LDDW que lee 64 bits por lado, dando un total de 128 bits por ciclo.

Otra característica del CPU C67x es la arquitectura lectura/almacenamiento, donde todas las instrucciones operan con registros. Dos conjuntos de unidades funcionales (.D1 y .D2) son responsables de toda la transferencia de datos entre las filas de registros y la memoria. El manejo de la dirección de datos, por la unidad .D, permite direccionar datos generados de un registro para usarse en la carga y almacenamiento de datos hacia/desde otro registro, en la misma fila de registros. El CPU C67x soporta una variedad de modos de direccionamiento indirecto usados para direccionamiento circular y lineal con 5 o 15 bits de *offset*. Todas las instrucciones son condicionales y la mayoría puede acceder a cualquiera de los 32 registros. Las dos unidades funcionales .M son dedicadas para la multiplicación. Las unidades .S y .L ejecutan un conjunto general de funciones aritméticas, lógicas y de saltos con resultados disponibles en cada ciclo de reloj. Una descripción detallada del CPU en los dispositivos C67x se encuentra en la literatura [SPRU733]

6.4.1. Filas de registros de propósito general

Como ya se ha mencionado, el CPU tiene dos filas de registros de propósito general, cada una de esas filas contiene 16 registros de 32-bits (A0-A15 para la fila A y B0-B15 para la fila B). Estos registro pueden ser usados para datos, puntero a direcciones de datos o registros de condición (ver figura 6.4).

Los registros de los DSPs C67x soportan rangos de datos desde 16-bits hasta 40-bits en punto fijo y 64-bits para punto flotante. Los valores mayores de 32 bits, tales como cantidades de 40-bits y 64-bits, son almacenados en pares de registros. En donde los 32 bits menos significativos (LSBs) del dato, son colocados en un registro con número par y los 8 ó 32 bits restantes mas significativos (MSBs) en el siguiente registro impar (este es siempre un registro con número impar). Todas las operaciones de lectura y escritura para registros de 64-bits son ejecutadas en dos ciclos de reloj. La tabla 6.1 muestra los registros para almacenar valores mayores a 32 bits.

6.4.2. Unidades funcionales

Las ocho unidades funciones en el C67x, se pueden dividir en dos grupos de cuatro: El primer grupo tiene una terminación de 1 (.L1, .S1, .D1 y .M1) corresponden a la fila de registros A; mientras que las que terminan en 2 (.L2, .S2, .D2 y .M2) corresponde a la fila de registros B (ver figura 6.4). Cada unidad funcional tiene dos puertos de 32-bits de lectura para los operandos fuente *src1*

Tabla 6.1: Pares de registros de 40-bits/64-bits

Filas de Registros	
A	**B**
A1:A0	B1:B0
A3:A2	B3:B2
A5:A4	B5:B4
A7:A6	B7:B6
A9:A8	B9:B8
A11:A10	B11:B10
A13:A12	B13:B12
A15:A14	B15:B14

y *src2*. Las cuatro unidades .L1, .L2, .S1 y .S2 tienen un puerto extra de 8-bits de ancho para escribir y leer palabras de 40-bits. Debido a que cada unidad tiene su propio puerto de 32-bits para escribir, se pueden usar las ocho unidades en paralelo en cada ciclo, cuando se ejecutan operaciones de 32-bits. Las unidades funcionales se describen en la tabla 6.2.

6.4.3. Caminos cruzadas de datos entre las filas de registro

Cada unidad funcional lee directamente y escribe directamente a la fila de registro que le corresponde. Esto es, las unidades .L1, .S1, .D1 y .M1 escriben a la fila de registros A y las unidades .L2, .S2, .D2 y .M2 escriben a la fila de registros B. Las filas de registros están conectadas a las unidades funcionales de la fila de registros opuesta, por medio de caminos cruzados 1X y 2X. Estas trayectorias cruzadas permiten a las unidades funcionales acceder a operandos de 32-bits de la fila de registros opuesta. El camino cruzado 1X permite a las unidades funcionales del camino de datos A, leer su operando fuente de la fila de registros B; y el camino cruzado 2X permite a las unidades funcionales de camino de datos B, leer el operando fuentes de la fila de registros A (ver figura 6.4).

En los DSPs C67x, seis de las ocho unidades funcionales tiene acceso a la fila de registros opuesta, por medio del camino cruzado. El operando 2 (*src2*) de las unidades .M1, .M2, .S1 y .S2 pueden usar la fila de registros opuesta a través del camino cruzado, o la fila de registros que se encuentra en el mismo lado. En el caso de las unidades .L1 y .L2, los operandos de entrada *src1* y *src2* pueden utilizar la fila de registros opuesta mediante el camino cruzado, o la fila de registros en el mismo lado.

Existen sólo dos caminos cruzados en la arquitectura C68000, 1x y 2x. De esta forma, el límite de lectura de filas de registros opuesta es de dos por ciclo.

Tabla 6.2: Unidades funcionales y sus operaciones de ejecución

Unidad Funcional	**Operaciones de punto fijo**	**Operaciones de punto flotante**
Unidad .L (.L1, .L2)	Operaciones aritméticas de 32/40-bits y comparación Operaciones lógicas de 32-bits Conteo de 1 ó 0 mas a la izquierda para 32 bits Conteo de normalización para 32 y 40 bits	Operaciones Aritméticas Operaciones de conversión DP→SP, INT→DP, INT→SP
Unidad .S (.S1, .S2)	Operaciones aritméticas de 32 bits Operaciones de corrimiento de 32/40-bits y operaciones de campo de bit de 32-bits Operaciones lógicas de 32-bits Saltos Generación de constantes Transferencia de registro desde/hacia filas de registros de control (sólo .S2)	Comparaciones Operaciones de recíproco y raíz cuadrada recíproca Operaciones de valor absoluto Operaciones de conversión SP→DP Substracción y sumas de SP y DP Substracción invertida SP y DP (src2 - src1)
Unidad .M (.M1, .M2)	Operaciones de multiplicación de 16x16 bits Operaciones de multiplicación de 32x32 bits	Operaciones de multiplicación con punto flotante Operaciones de multiplicación de precisión mixta
Unidad .D (.D1, .D2)	Suma y resta de 32 bits Cálculo del direccionamiento circular y lineal Lectura y almacenamiento con offset constante de 5-bits Lectura y almacenamiento con offset constante de 15-bits (Sólo .D2)	Lectura de doble ancho de palabra con offset constante de 5-bits

En el DSP C67x, sólo una unidad funcional, por camino de datos, puede recibir operandos de la fila de registros opuesta.

6.4.4. Interrupciones del CPU

Típicamente, los DSPs trabajan en un entorno que contiene múltiples eventos externos asíncronos. Esos eventos requieren que el DSP ejecute tareas, cuando estos ocurren. Una interrupción es un evento que detiene el proceso actual en el CPU, de tal forma que pueda atender a la tarea necesaria debida al evento. Esta fuente de interrupción puede ser dentro del chip o externo a él, tal como temporizadores (*timers*), convertidores analógico digital o periféricos.

Tipos de interrupciones

Existen tres tipos de interrupciones en los CPUs de los DSPs C6000.

- Reset
- Mascarables
- No mascarables

Esos tres tipos son diferentes por sus prioridades, como muestra la tabla 6.3. La interrupción de *reset* tiene la mas alta prioridad y corresponde a la señal $\overline{RESET}$. La interrupción no mascarable tiene la segunda mas alta prioridad y corresponde a la señal NMI. Las interrupciones de baja prioridad son las interrupciones 4-15 que corresponden a las señales $INT4 - INT15$ que se encuentran mapeadas a los pines del C6000. Algunas señales de interrupción $INT4 - INT15$ son usadas por periféricos internos y algunas pueden estar no disponibles o pueden usarse bajo control de software. Todas las interrupciones de los dispositivos C67x se describen en la literatura [SPRU733].

Habilitar e inhabilitar interrupciones globalmente

El registro de control de estado (CSR) contiene dos bits que controlan las interrupciones: GIE (habilita interrupciones globalmente) y PGIE (valor previo de GIE). El bits GIE permite habilitar e inhabilitar todas las interrupciones mascarables. Cuando el bit GIE se enciende (GIE = 1) se habilita las interrupciones y cuando GIE = 0, se inhabilitan. El propósito del bit PGIE es guardar el valor del bit GIE durante el comienzo del proceso de la interrupción

Tabla 6.3: Prioridades de las interrupciones

Prioridad	**Nombre de la interrupción**	**Tipo de interrupción**
Alta	Reset	Reset
	NMI	No mascarable
	INT4	Mascarable
	INT5	Mascarable
	INT6	Mascarable
	INT7	Mascarable
	INT8	Mascarable
	INT9	Mascarable
	INT10	Mascarable
	INT11	Mascarable
	INT12	Mascarable
	INT13	Mascarable
	INT14	Mascarable
Bajo	INT15	Mascarable

Habilitar e inhabilitar interrupciones individualmente

Se puede activar y desactivar las interrupciones por medio del registro IER (registro de habilitador de interrupciones). Una interrupción puede activarse sólo si el bit correspondiente en el IER está encendido. El bit 0 en el IER corresponde a la interrupción de *reset*, es de sólo lectura y tiene el valor de 1, por lo que la interrupción de *reset* siempre está habilitada. Los bits IE4-IE15 activan las interrupciones mascarables con el número correspondiente, el valor 1 las habilita y 0 las inhabilita. Cuando el bit NMIE del registro IER está en bajo (NMIE = 0), todas las interrupciones están inhabilitadas excepto la interrupción de reset.

El registro de banderas de interrupción (IFR) contiene el estado de las interrupciones INT4-INT15 y NMI. Cada bit en el registro IFR, corresponde a una interrupción, si este bit esta activado (1) entonces la interrupción ocurre; en otro caso, el bit tiene un valor a 0.

Para activar o desactivar las interrupciones mascarables manualmente, se utiliza el registro ISR (registro de activación de interrupciones) y el registro ICR (registro para desactivar la interrupción). Escribiendo un valor de 1 en los bits IS4-IS15 en el registro ISR, causa que la bandera de la interrupción correspondiente, en el registro IFR se active. De la misma forma, si se escribe un 1 en un bit en el registro ICR, causa que la bandera de interrupción correspondiente se desactive. Si se escribe un 0 en cualquier bit de los registros ISR ó ICR, no tiene algún efecto.

6.5. Periféricos del DSP C67x

La figura 6.5 muestra un diagrama de bloques de los periféricos de los dispositivos C67x. Estos periféricos se describen a continuación (todos los periféricos de los dispositivos C67x se describen a detalle en la literatura [SPRU190D]):

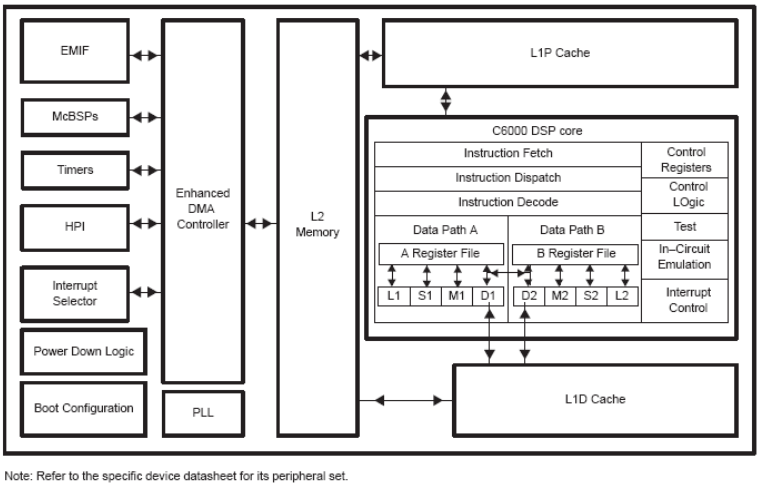


Figura 6.5: Diagrama de bloques de los DSPs C621x/C671x.

- **Controlador de acceso directo a memoria mejorado (EDMA)**: El controlador EDMA transfiere datos, entre un rango de direcciones en el mapa de memoria, sin la intervención del CPU. Maneja todas las transferencias de datos entre el controlador de memoria/caché de nivel dos (L2) y los dispositivos periféricos. Tiene 16 canales independientes con prioridad programables y la habilidad de ligar y encadenar transferencias de datos. El EDMA permite movimientos de datos desde/hacia cualquier espacio de memoria direccionable, incluyendo memoria interna (L2, SRAM), periféricos y memoria externa.
- **Interface de puerto host (HPI)**: El HPI es un puerto paralelo de 16 bits, en el cual, un procesador *host* puede acceder directamente al espacio de memoria del CPU. El dispositivo host funciona como maestro para la interface, que facilita el incremento de acceso. El host y el CPU pueden intercambiar información por medio de la memoria interna o externa. Además, el host tiene acceso directo a los periféricos mapeados en memoria.
- **Interface de memoria externa (EMIF)**: El EMIF soporta una interface *glueless* de 32-bits[3] para varios dispositivos externos, incluyendo los siguientes:

[3]El término *glueless*, se refiere a que el dispositivo (DSP) tiene todas las señales de con-

  - SRAM síncrona de ráfaga (*burst*), (SBSRAM)
  - DRAM síncrona (SDRAM)
  - Dispositivos asíncronos, incluyendo SRAM, EPROM, ROM y FIFOs
  - Un dispositivo externo de memoria compartida

- **Configuración de inicio**: Los dispositivos C6000 proporcionan una variedad de configuraciones de inicio, las cuales determinan las acciones ejecutadas por el DSP después de que el dispositivo se ha restablecido y preparado para la inicialización. Este incluye carga de código de un espacio de ROM externo en el EMIF y carga de código a través del HPI de un host externo.

- **McBSP**: El puerto serial de multicanal con buffer (McBSP) está basado en el estándar de interface de puerto serie encontrado en los dispositivos de las plataformas C2000 y C5000. Además, el puerto puede almacenar muestras seriales en memoria automáticamente con la ayuda del controlador EDMA. También tiene la capacidad multicanal compatible con los estándar de redes T1, E1, SCSA y MVIP. Este puerto tiene las siguientes capacidades:

  - Comunicación *full-duplex*
  - Registros de datos de doble-buffer que permite un flujo de datos continuo.
  - Tramado independiente y sincronización por medio de reloj en la recepción y transmisión
  - Interface directa a *codecs* (codificador-decodificador) estándar de la industria, chips de interface analógica (AICs) y otros dispositivos analógico/digital (A/D) y digital/analógico (D/A) conectados serialmente.

    Además, el McBSP tiene las siguientes capacidades:

  - Interface directa a:
    - Tramas T1/E1
    - Dispositivos conforme a ST-BUS
    - Dispositivos conforme a IOM-2
    - Dispositivos conforme a AC97
    - Dispositivos conforme a IIS
    - Dispositivos SPI
  - Transmisión y recepción multicanal hasta 128 canales
  - Una gran selección de tamaño de datos que incluye: 8, 12, 16, 20, 24 y 32 bits.

trol necesarias para una memoria particular, sin la necesidad de usar alguna lógica externa adicional.

  - Compansión ley-μ y ley-A
  - transferencia de datos de 8-bits con LSB (bit menos significativo) o MSB (bit más significativo) primero.
  - Polaridad programada de sincronización de tramas y reloj de datos
  - Reloj interno y generador de tramas altamente programables

- **Timer**: Los dispositivos C6000, contienen *timers* de 32-bits de propósito general que pueden ser usados como:
  - Eventos de tiempo
  - Eventos de conteo
  - Generador de pulsos
  - Interrupción de CPU
  - Envío de eventos de sincronización para el controlador EDMA

- **Selector de interrupciones**: El conjunto de periféricos de los C60000 tiene hasta 32 fuentes de interrupción. Sin embargo, el CPU tiene disponible 12 interrupciones para usarse. El Selector de interrupciones permite al usuario elegir y priorizar, cual de las 12 interrupciones de 32, se necesita en el sistema. El selector también permite cambiar la polaridad de la interrupciones externas.

- **Lógica para bajo consumo de energía**: La lógica para bajo consumo de energía permite reducir el gasto de la energía. La mayoría de la potencia de operación de la lógica CMOS se disipa durante la conmutación de un estado lógico a otro. Los modos de bajo consumo de energía pueden ser usados para lograr un significante ahorro de energía sin la pérdida de datos o el contexto de operación. En los DSP C6000 tiene tres modos de bajo consumo de energía.

6.6. Code composer Studio

El Code Composer Studio (CCStudio) es una de las herramientas de desarrollo de software de Texas Instruments, que incluye características necesarias para llevar cada una de las fases del ciclo de desarrollo para aplicaciones embebidas con DSPs. La figura 6.6 muestra las etapas de desarrollo en aplicaciones con DSP y las características incluidas en el CCStudio en cada fase.

El CCStudio tiene los siguientes componentes:

- Herramientas de generación de código de los DSPs TMS3206000

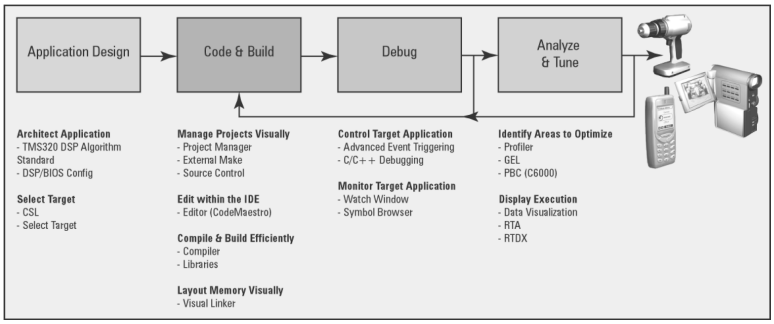


Figura 6.6: Características incluidas en el CCStudio para las fases desarrollo de aplicaciones con DSPs.

- Un entorno de desarrollo integrado (IDE)
- DSP/BIOS *Plug-ins* y API
- RTDX *plug-in*, API e interface host.

Estos componentes, trabajando en conjunto, se muestran en la figura 6.7.

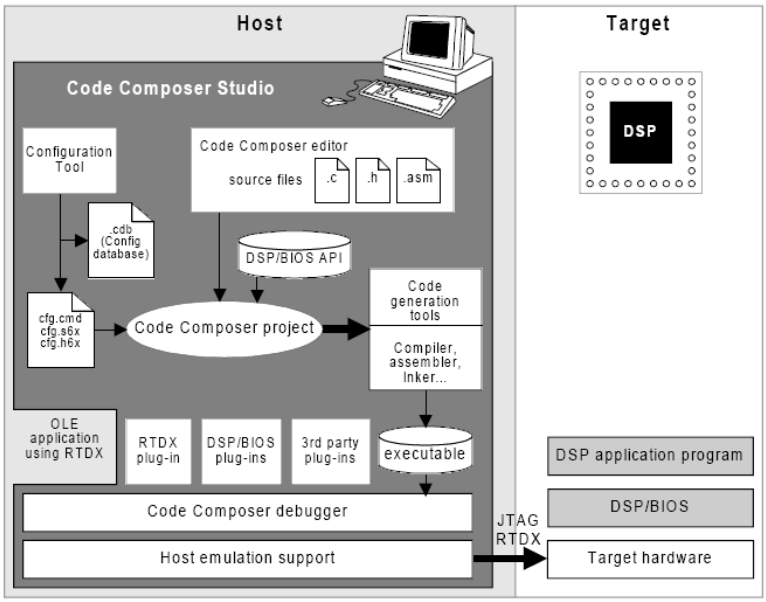


Figura 6.7: Interacción de los componentes del CCStudio para la elaboración de una aplicación.

6.6.1. Herramientas de generación de código

El conjunto de herramientas de generación de código proporcionado por el CCStudio incluye: una optimización del compilador de C/C++, un optimizador de ensamblado, un ensamblador, un ligador y un conjunto surtido de utilerías.

La figura 6.8 muestra el flujo de desarrollo de software para las aplicaciones con DSPs C6000. La parte sombreada representa la ruta más común del desarrollo, las demás partes son opcionales y representan funciones periféricas que mejorar el proceso de desarrollo.

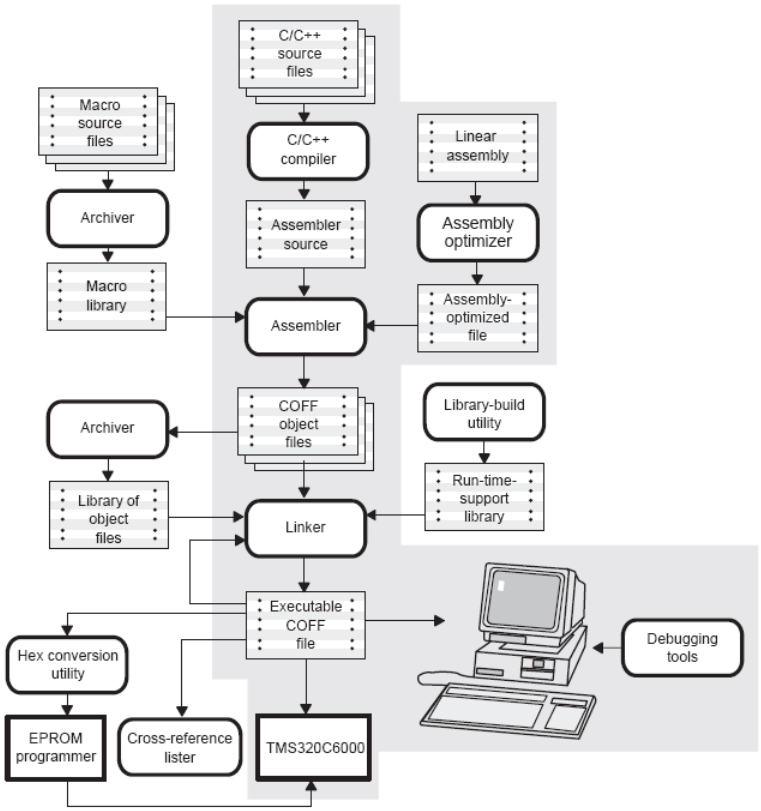


Figura 6.8: El flujo de desarrollo para aplicaciones con los DSP C6000.

- El **optimizador de ensamblado** (*assembly optimizer*), permite escribir código en lenguaje ensamblador lineal o secuencial, sin importar la estructura del pipeline o la asignación de registros. Este acepta código en ensamblador en donde no se han asignado registros. El optimizador de ensamblado asigna los registros y usa optimización de ciclos, para con-

vertir el código de lenguaje ensamblador lineal en lenguaje ensamblador altamente paralelo y que toma las ventajas del pipeline de software.

- El **Compilador C/C++**. Acepta código fuente en lenguaje C/C++ y produce código en lenguaje ensamblador para el C6000. El compilador incluye además: un *programa de shell*, un *optimizador* y una *utilería de interlistado*.

  - El *programa de shell* habilita el compilador, ensamblador y liga los módulos fuente en un sólo paso. Si hay un archivo de entrada con extensión .sa, el programa de shell invoca al optimizador de ensamble.
  - El *optimizador* modifica el código para mejorar la eficiencia en los programas en C.
  - La *utilería de interlistado* entreteje las sentencias fuente en C/C++ con la salida del lenguaje ensamblador.

- El **ensamblador** (*assembler*) traduce los archivos fuente en lenguaje ensamblador a archivos objeto en lenguaje máquina. El lenguaje máquina esta basado en el formato de archivos de objeto común (COFF, *Common Object File Format*).

- El **ligador** (*linker*). Combina los archivos objeto en un sólo archivo ejecutable para el DSP. Acepta archivos de librerías y módulos de salidas, creados por una ejecución previa.

- El **archivador** (*archiver*). Permite juntar un grupo de archivos dentro de un sólo archivo, llamado librería. Por ejemplo, se puede juntar distintas macros, dentro de una librería de macros; también se puede usar el archivador para reunir un grupo de archivos objeto en una librería.

- También se puede usar la **utilería de construcción de librerías** (*library-build utility*), para formar librerías de soporte en tiempo real personalizadas.

- El **listado absoluto** (*absolute lister*) es una herramienta para depuración de programas. Aceptan como entrada, archivos objeto ligados y crea archivos con extensión .abs, que son ensamblados para producir una lista que muestra las direcciones absolutas del código objeto.

- La **utilería de conversión hexadecimal** (*hex conversion utility*). Convierte un archivo objeto de tipo COFF a un archivo cuyo formato objeto se puede seleccionar entre los siguientes: TI-Tagged, ASCII hexadecimal, Intel, Motorola-S o Tektronix. El archivo convertido, puede ser almacenado en un programador de EPROM.

- El **listador de referencias cruzadas** (*cross-reference lister*). Usa archivos objeto para producir un listado de referencias cruzadas; mostrando símbolos, definiciones y sus referencias en el archivo fuente vinculado.

6.6.2. Entorno de desarrollo integrado (IDE) CCStudio

El IDE CCStudio proporciona un programas donde se puede realizar todo el proceso de generación de código como: editar, compilar, depurar y cargar programas en el DSP. Además incluye: soporte para el sistema operativo de tiempo real DSP/BIOS, capacidades de análisis en tiempo real, herramientas de optimización un administrador de proyectos de forma visual y una variedad de simuladores y controladores de emulación. La figura 6.9 muestra la ventana de control del CCStudio.

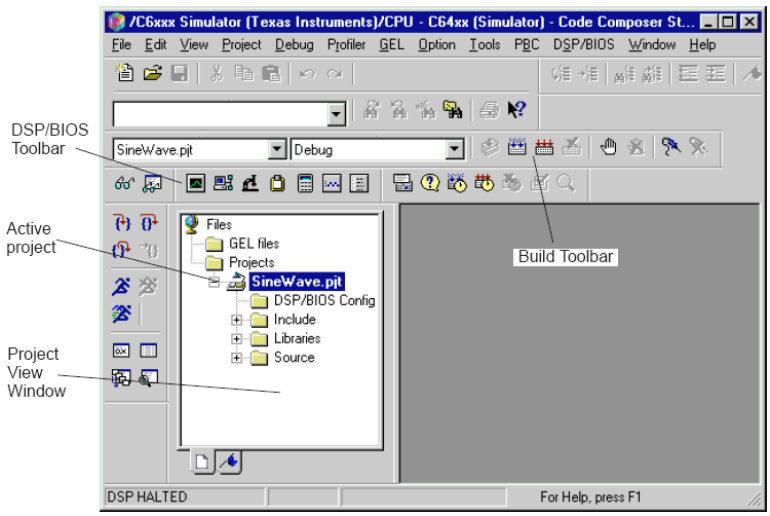


Figura 6.9: Ventana de control del Code Composer Studio.

Características del editor de código de programas

El IDE CCStudio incluye un editor para la escritura de programas en C, C++ y código en ensamblador para el DSP. El editor contiene las características estándar tales como: resaltado de palabras clave, impresión, cortar y pegar, arrastrar y soltar, etc.

Las barras de herramientas flotantes soportan avanzadas operaciones tales como: la búsqueda del paréntesis o corchete, de apertura o cierre, que le corresponde; sangrado de texto, función de búsqueda y reemplazo; ayuda sensible al contexto.

También soporta: marcadores (*bookmarks*), para buscar y guardar las ubicaciones clave en los archivos fuente; edición de columna, para seleccionar, cortar, copiar, pegar y borrar columnas de texto.

El editor tiene una sección de margen que, por medio de iconos, identifica los marcadores, los puntos de interrupción, los puntos de prueba, la localización del contador de programas, entre otras cosas. Esta sección se puede desactivar, identificando los puntos por medio de líneas de color.

El editor soporta la vista de modo combinado de código. En esta vista, se muestra el código fuente escrito en lenguaje C/C++, entrelazado con las instrucciones en ensamblador apropiadas para un DSP particular. La figura 6.10 muestra este tipo de vista.

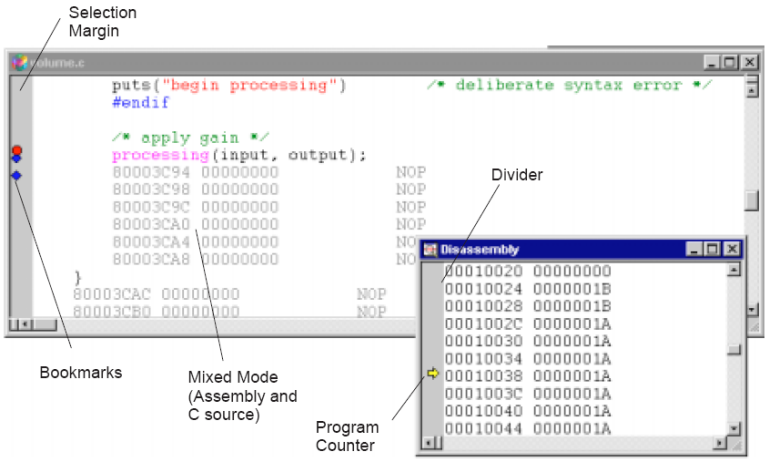


Figura 6.10: Vista de modo de combinación de código.

El editor provee dos modos de seleccionar texto e insertar espacios: el modo de flujo y el modo de espacio en blanco virtual. En el primero (modo flujo), el editor provee la selección de texto e inserción de forma similar a la mayoría de los procesadores de palabras. En el segundo modo (espacio en blanco virtual), el ratón o las teclas de flechas pueden ser usados para mover el cursor a cualquier localización dentro de la ventana de documento. Cuando se escribe, cualquier espacio en blanco entre el final de la línea y el cursor es automáticamente llenada con espacios. De forma similar, cuando se selecciona texto, cualquier área después del fin de línea es automáticamente llenado con espacios.

El editor resalta en diferente color las palabras clave, comentarios, cadenas, directivas de ensamblador y comandos GEL (Lenguaje de Extensión General). Estas palabras clave son resaltadas sólo para archivos C, C++, ensamblador y GEL (con extensión .c, .cpp, .asm y .gel, respectivamente). Se pude personalizar las palabras clave resaltadas, con sólo crear un archivo de texto.

El IDE del CCStudio soporta el uso de un editor de texto externo (de otro fabricante) en lugar de el editor integrado por defecto. Después que el editor

externo es configurado y habilitado, el editor externo es llamado cuando, un nuevo documento es creado o cuando se abre un archivo existente. Una desventaja al usar un editor externo, es que sólo se puede editar, no se puede depurar programas.

Administrador de proyectos

El administrador de proyectos organiza los archivos en carpetas para archivos: fuente, librerías, de configuración, de encabezado (*include*) y archivos generados. Con esto facilita la navegación entre los archivos de código. El administrador de proyectos automáticamente revisa dependencias. La figura 6.11 muestra el administrador de proyectos.

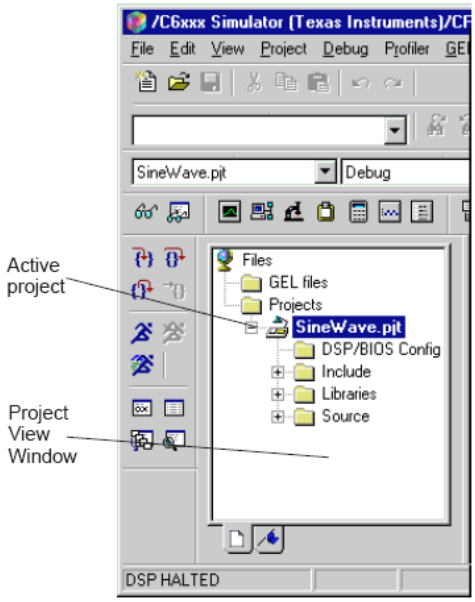


Figura 6.11: Administrador de proyectos.

El administrador de proyectos soporta configuraciones múltiples para construcción de código. Como por ejemplo, configuración de depuración (*Debug*) y una configuración de liberación (*Release*).

Otras características que tiene el administrador de proyectos es: la apertura de múltiples proyectos simultáneamente, la especificación para determinar si el proyecto es un ejecutable o una librería, establecer las opciones de construcción de archivos en forma individual, etc.

El depurador del CCStudio

El depurador del CCStudio ayuda a localizar y reparar errores en programas embebidos de tiempo real. Los comandos del depurador permiten controlar la ejecución del programa. Las ventanas de depurador, permiten ver el código fuente, el valor de variable en el programa, la memoria y registros. La utilización de puntos de interrupción, permite detener la ejecución en un lugar especificado para poder examinar el estado del programa.

- *Ventana de memoria*. La ventana de memoria (figura 6.12-(a)) permite ver y editar el contenido de la memoria a partir de una dirección dada, así como cambiar el formato de visualización.
- *Ventana de registros*. Con esta ventana se puede ver y editar el contenido de los registros del CPU y los registros de los periféricos (figura 6.12-(b)).

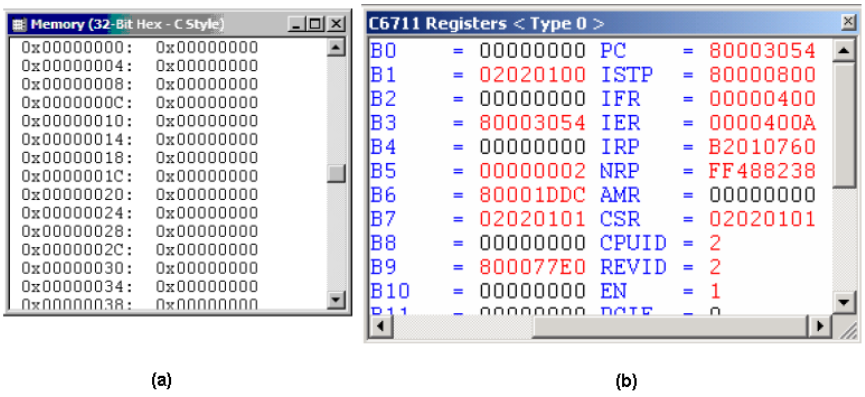


Figura 6.12: (a) Ventana de memoria. (b) Ventana de registros.

- *Ventana de desensamble*. En esta ventana se muestra las instrucciones desensambladas e información de símbolos necesarias para la depuración (figura 6.13-(a)). El desensamblado es el proceso inverso de ensamblar, y permite desplegar el contenido de la memoria en lenguaje ensamblador.
- *Ventana de llamadas a la pila*. El uso de esta ventana permite examinar las llamadas a función (figura 6.13-(b)).
- *Ventana de visualización de variables*. La ventana de visualización de variables (*Watch Window*), permite inspeccionar los valores de variables locales y globales, además de expresiones en C/C++ (figura 6.14).

Además de los puntos de interrupción, el depurador del CCStudio contiene los puntos de prueba (*Probe Point*). Cuando la ejecución de un programa alcanza

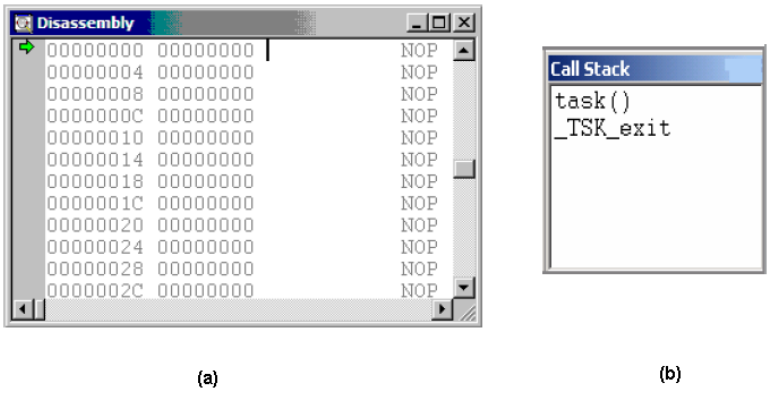


Figura 6.13: (a) Ventana de desensamble. (b) Ventana de llamadas a la pila.

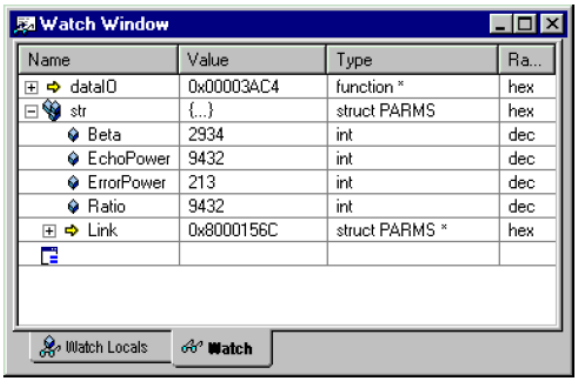


Figura 6.14: Ventana de visualización de variables.

a un punto de prueba, el objeto conectado (un archivo, una gráfica o la ventana de memoria) es actualizado.

El depurador permite transferir datos desde/hacia un archivo al DSP, usando la característica de archivos de Entrada/Salida. Esto permite simular el código con valores de muestra conocidas. La características de archivos de E/S usa los puntos de prueba para transferir cada uno de los datos hacia/desde el DSP.

El Manejador de depuración paralelo (PDM) permite sincronizar múltiples procesadores. Si se tiene varios procesadores y manejadores de dispositivos que lo soportan, el PDM es habilitado cuando se inicia el CCStudio. Con la barra de menú PDM, se puede abrir ventanas individuales para controlar cada uno de los procesadores o los comandos de transmisión para un grupo de procesadores.

Análisis del código de la aplicación

Un análisis del código de la aplicación puede revelar muchas oportunidades para mejorar la eficiencia, especialmente cuando se sabe donde se observa. Las herramientas proporcionadas por el CCStudio para este fin son:

- **Visualización de datos**. La visualización de datos es útil cuando se desarrolla aplicaciones para comunicaciones, inalámbricos, procesamiento de imágenes, así como aplicaciones de propósito general.

  Existe una gran variedad de formas para graficar los datos procesados en el CCStudio. Los tipos de gráficas que se encuentran disponibles son: tiempo/frecuencia, diagrama de constelación, diagrama de vista e imágenes. La figura 6.15 muestra el cuadro de dialogo para configurar las propiedades de las gráficas. La figura 6.16 muestra la gráfica de una señal en el dominio del tiempo.

- **Análisis del Simulador**. La herramienta de análisis del simulador reporta la ocurrencia de eventos particulares del sistema, de tal forma que se pueda monitorizar con precisión y medir el desempeño del programa. La figura 6.17 muestra la ventana de análisis del simulador.

- **Análisis del emulador**. La herramienta de análisis del emulador (figura 6.18) permite establecer, monitorizar y contar eventos, y puntos de interrupción de hardware.

- **Herramientas de análisis en tiempo real (RTA) del DSP/BIOS**. La característica de análisis en tiempo real del DSP/BIOS (RTA), proporciona la capacidad de probar, seguir y monitorizar una aplicación en el DSP, durante el curso de su ejecución. Estas herramientas se describen en mayor detalle más adelante en la siguiente sección.

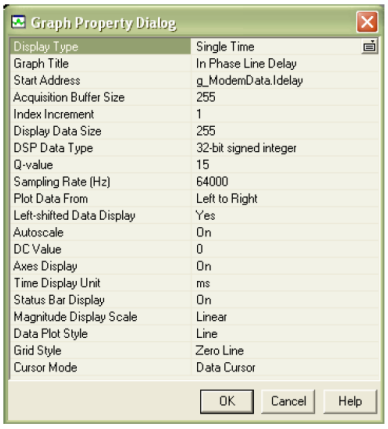


Figura 6.15: Cuadro de dialogo para configurar las propiedades de las gráficas.

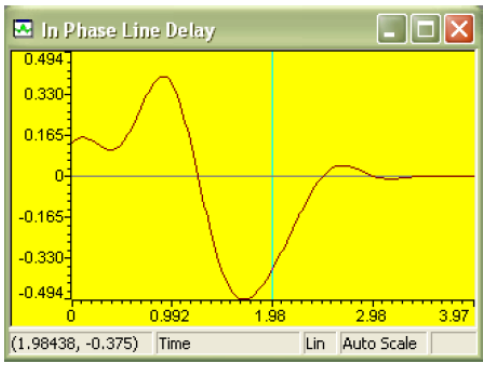


Figura 6.16: Ventana de visualización de la gráfica.

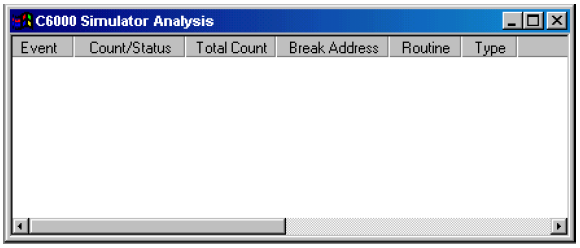


Figura 6.17: Ventana de análisis del simulador.

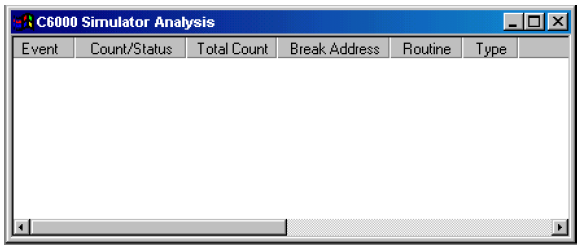


Figura 6.18: Ventana de análisis del simulador.

6.6.3. El examinador del código de ejecución (*profiler*)

CCStudio permite recolectar estadísticas de ejecución en áreas específicas del código. A esto se le llama examinación (*profiling*). El examinador (*profiler*) analiza la ejecución del programa y muestra el consume de tiempo del código. De esta forma, ayuda a identificar rápidamente los cuellos de botella. Por ejemplo, un análisis del examinador puede reportar cuantos ciclos de reloj toma una función para ejecutarse, así como, que tan a menudo se invoca.

El examinador soporta el análisis de funciones escritas en lenguaje C o C++, aun no soporta funciones en lenguaje ensamblador. La información mostrada para esta área de examinación es: el tamaño del código, el número de veces que se ha invocado la función, el numero de ciclos totales transcurridos en las llamadas de la función, el promedio de ciclos consumidos por llamada, etc.

El segundo tipo de área de examinación, es el rango. Un rango es una sección de código establecida por un inicio y un final de área. La información presentada por este tipo de área de examinación, es similar a la mostrada en las funciones.

La figura 6.19 muestra la ventana de una sesión del examinador.

6.6.4. DSP/BIOS

El DSP/BIOS es un *kernel* escalable. Se diseño para aplicaciones que requieren programación y sincronización en tiempo real, comunicación del host (PC) a la tarjeta e instrumentación en tiempo real. El DSP/BIOS proporciona multi-procesos con prioridades, abstracción de hardware, análisis en tiempo real y herramientas de configuración.

El DSP/BIOS incluye los siguientes componentes:

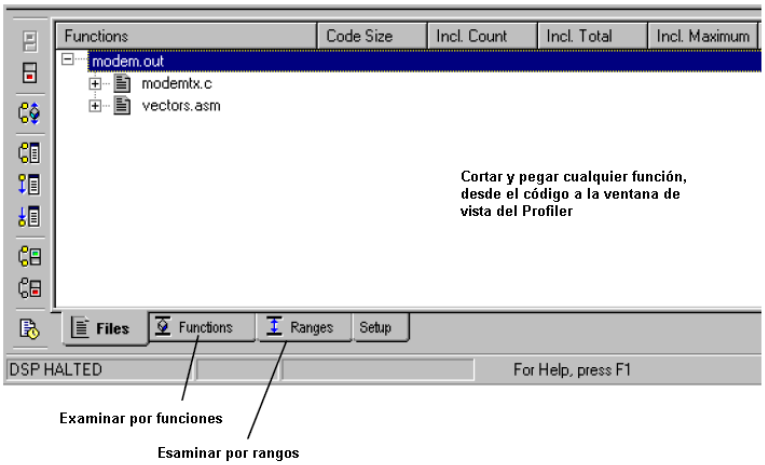


Figura 6.19: Ventana del examinador.

- **La API del DSP/BIOS**. La API (Interfaz de Programación de Aplicaciones) del DSP/BIOS proporciona un conjunto de funciones en C, C++ y en lenguaje ensamblador; para el manejo de los objetos del DSP/BIOS.
- **Herramienta de configuración del DSP/BIOS**. Esta herramienta permite crear y configurar los objetos del DSP/BIOS usados en el programa. También se puede usar para configurar la memoria, propiedades de procesos, manejador de interrupciones y configuración por medio de la librería del soporte del chip (CSL).
- **Herramientas de análisis en tiempo real del DSP/BIOS**. Estas herramientas permiten observar la actividad del programa en tiempo real, supervisar la carga del CPU, el tiempo, mensajes (*logs*), ejecución de hilos (*thread*[4]) y más.

El API del DSP/BIOS

El API del DSP/BIOS es un conjunto de funciones en lenguaje C, C++ y ensamblador para el manejo de los objetos del DSP/BIOS. Estas funciones están divididas en módulos, para que sólo la parte utilizada, sea ligada al programa ejecutable (el nombre de las funciones, comienzan con el código correspondiente a su módulo); también, se encuentran optimizadas con el menor número posible de instrucciones por ciclo, con una parte significante implementada en lenguaje ensamblador. Los módulos en que se divide el API del DSP/BIOS son:

[4] *thread* se refiere a cualquier de los siguientes hilos de ejecución: interrupciones por hardware, interrupciones por software, tareas o funciones ociosas (*idle*)

- **ATM**: En este módulo se encuentran un conjunto de funciones en lenguaje ensamblador, que son usadas para manipular variables con interrupciones desactivadas. Estas funciones pueden ser usadas en datos compartidos entre tareas, y datos compartidos entre rutinas de interrupción y tareas.

- **BUF**: Contiene un conjunto de funciones para el manejo de bloques de buffers de tamaño fijo. Los bloques de buffer pueden ser creados de forma estática o dinámica.

- **C62**: Este módulo proporciona funciones específicas del DSP, como por ejemplo, las definiciones de los registros del CPU, definición de estructuras, macros en lenguaje ensamblador para salvar y restaurar registros en las rutinas de interrupción.

- **CLK**: Este módulo es el administrador del reloj del sistema. Provee un reloj de tiempo real, con funciones que acceden a este reloj en dos resoluciones (alta y baja). El contador se almacena en un valor de 32-bits para ambas resoluciones.

- **DEV**: Proporciona la interface del dispositivo. El DSP/BIOS provee dos modelos de manejadores de dispositivo, que habilitan la comunicación entre la aplicación y los periféricos del DSP: IOM y SIO/DEV.

  Los componentes del modelo IOM separan las capas independientes del hardware (*class driver*) y dependiente del hardware (*mini driver*). La primer capa, administra peticiones de dispositivo, sincronización y serialización de solicitudes de Entrada/Salida. La segunda capa, es una clase de manejador que usa un dispositivo específico para su operación particular (Para mayor información consulte la guía de desarrollo [SPRU616]).

  El modelo SIO/DEV provee una interface de flujo de E/S. En este modelo, la aplicación invoca indirectamente a funciones DEV implementadas por el Administrador del manejador del dispositivo físico particular unido al flujo, usando funciones genéricas proporcionadas por el módulo SIO (para mayor información consulte la guía de usuario [SPRU423]).

- **HST**: El módulo HST administra los objetos del canal *host*, que permiten a una aplicación transmitir flujo de datos entre el DSP y un host. Los canales host son configurados estáticamente para entrada o salida.

- **HWI**: El módulo de HWI administra las interrupciones de hardware. Con la herramienta de configuración del DSP/BIOS, se puede asignar funciones que atiendan a las interrupciones de hardware. Estas rutinas se pueden escribir completamente en C, en ensamblador o una mezcla de ambos.

  El módulo provee funciones específicas para habilitar y desactivar interrupciones de hardware, así como una función para desactivar de forma global las interrupciones; también provee dos macros, en lenguaje ensamblador, para salvar y restaurar los registros del CPU, en rutinas que atienden a interrupción de hardware (ISR) del DSP/BIOS.

- **IDL** El módulo IDL gestiona los hilos de nivel más bajo en una aplicación. Además de las funciones creadas por el usuario, el módulo IDL ejecuta funciones del DSP/BIOS que manejan la comunicación de host y el cálculo de la carga del CPU.

  Existen cuatro tipos de hilos que pueden se ejecutados en los programas de DSP/BIOS: interrupciones por hardware (módulo HWI), interrupciones por software (módulo SWI), tareas (módulo TSK) e hilos en segundo plano (módulo IDL). Los hilos en segundo plano (*background threads*) tienen la más baja prioridad y se ejecutan sólo si ninguna interrupción por hardware, interrupciones por software o tareas necesitan ejecutarse.

- **LCK**: El módulo bloqueo, contiene un conjunto de funciones que manipulan los objetos *lock*, accedidos a través de manejadores (*handles*). Cada *lock* implícitamente corresponde a un recurso compartido global, y es usado para arbitrar el acceso al recurso, entre varias tareas que compiten por él.

  El módulo LCK contiene un par de funciones para adquirir y abandonar la propiedad del recurso bloqueado mediante tareas. Esas funciones son usadas para delimitar secciones de código, que requieren acceso mutuamente excluyente a un recurso en particular. También contiene funciones para crear y eliminar objetos correspondientes a este módulo.

- **LOG**: El módulo LOG (registro), es usado para capturan eventos en tiempo real, mientras el programa es ejecutado en la tarjeta. Se puede usar el registro de sistema o se puede crear registros definidos por el usuario.

  Este módulo contiene funciones para habilitar y desactivar el mecanismo de registro de mensajes, además contiene funciones para enviar mensajes de error al registro de sistema, enviar mensajes formateados e inicializar el mecanismo de registros.

- **MBX**: El módulo MBX, dispone de un conjunto de funciones que manipulan objetos *mailbox* (buzón). Los buzones pueden usarse para enviar mensajes de una tarea a otra, en el mismo DSP.

  El módulo contiene funciones para crear y eliminar objetos buzón. También contiene una función para leer un mensaje del un buzón, en la cual, si no haya mensajes disponibles (el buzón se encuentra vacío), bloquea a la tarea hasta que un determinado tiempo se cumpla o indefinidamente hasta que se envía un nuevo mensaje al buzón.

- **MEM**: El módulo MEM proporciona un conjunto de funciones para asignar memoria de forma dinámica, a partir de uno o más segmentos de memoria disjuntos.

- **PIP**: Este módulo maneja tuberías de datos (*pipe*), que son usadas como flujos de buffers para entrada y salida de datos. Estas tuberías proporcionan una estructura de datos consistente, para manejar entradas/salidas entre el DSP y otros dispositivos periféricos, en tiempo real.

- **PRD**: El módulo de manejo de funciones periódicas, administra objetos de este tipo. Permite crear objetos PRD que programan ejecuciones periódicas para funciones especificadas. El período puede ser manejado por el módulo CLK o por llamadas a la función *PRD_tick* cuando ocurre un evento.

- **QUE**: El módulo QUE (colas[5]) contiene un conjunto de funciones para manipular objetos de este tipo. Cada cola contiene una secuencia ordenada de cero o más elementos referenciados a través de variables de tipo *QUE_elem*. Contiene funciones para: crear, borrar, obtener, insertar, vaciar, etc., elementos de la cola.

- **RTDX**: El intercambio de datos en tiempo real (RTDX), permite a las aplicaciones transferir datos entre la computadora host y los dispositivos DSP, sin interferir con la aplicación en el sistema del DSP. Los datos se pueden ser analizados y visualizados en el host, usando cualquier automatización cliente OLE.

  El módulo RTDX provee los tipos de datos y funciones para: enviar datos del DSP hacia el host, enviar datos del host al DSP, proveer buffer para almacenar los datos en el DSP, entre otras.

- **SEM**: El DSP/BIOS provee un conjunto esencial de funciones, para la sincronización y comunicación entre las tareas, basados en *semáforos*. Los semáforos son usados para coordinar el acceso a recursos compartidos, entre un conjunto de tareas que compiten ellos.

  El módulo SEM proporciona funciones que manipulan objetos semáforo. Los objetos SEM son semáforos contadores, que pueden ser usados para sincronización de tareas y mutua exclusión. Guardan un contador interno con el número del recursos correspondiente disponible. Cuando el contador es mas grande que cero, las tareas no bloquean al adquirir un semáforo. El módulo tiene funciones para: obtener el contador del semáforo, crear y eliminar un objetos semáforo, iniciar y restablecer semáforos, suspender y activar tareas, si el contador del semáforo es cero.

- **SIO**: Un flujo (*stream*) es un canal en donde fluyen datos entre una aplicación y dispositivos de E/S. Los flujos proveen una interface universal y simple para todos los dispositivos de E/S, permite que la aplicación ignore completamente el detalle de la operación de un dispositivo individual. El módulo SIO incluye un conjunto de funciones para administrar objetos de flujo como: crear y eliminar flujos, insertar y obtener datos del flujo, entre otras más.

[5] Una cola es una estructura de datos, caracterizada por ser una secuencia de elementos en la que, la operación de inserción *push* se realiza por un extremo y la operación de extracción *pop* por el otro. También se le llama estructura *FIFO* (del inglés *First In First Out*), debido a que el primer elemento en entrar, será también el primero en salir

- **STS**: El módulo STS administra objetos llamados acumuladores estadísticos. Los objetos STS capturan contadores, máximos, el total y el promedio de valores para cualquier variable en tiempo real.

  Con la ayuda del panel de control RTA, se pude habilitar el seguimiento de los módulo: HWI, que acumula estadísticas sobre valores supervisados dentro de las interrupciones de hardware; PIP, acumula el conteo del numero de tramas leídas y escritas a la tubería de datos; PRD, cuenta el numero de *ticks* transcurridos de tiempo, en donde el objeto PRD está listo a correr, hasta que se ha finalizada la ejecución; SWI, cuenta los ciclos de instrucción transcurridos desde el tiempo anunciado hasta la terminación; TSK, cuenta los ciclos transcurridos desde el tiempo que la tarea esta lista para correr, hasta que la aplicación llama a TSK_deltatime.

- **SWI**: Este módulo administra las interrupciones por software, que son hilos con menor prioridad que las interrupciones por hardware y mayor prioridad que las tareas. Las interrupciones por software pueden se activadas por medio de llamadas a funciones del modulo SWI. El DSP/BIOS usa prioridades para determinar si una interrupción es ejecutada o espera a que un hilo, con mayor prioridad, termine. Cada objeto SWI contiene un buzón (*mailbox*) de 32-bits, que puede ser usado para determinar, si es anunciada una interrupción por software o como un valor, que puede ser evaluado dentro de la función de interrupción.

- **SYS**: El módulo SYS contiene un conjunto de funciones de propósito general, que proporciona servicios básicos de sistema, tales como la detención de ejecución del programa e impresión de texto formateado. En general, las funciones SYS son similares a funciones que se encuentran en la librería estándar de C.

- **TRC**: El módulo de *trace* administra un conjunto de bits de control de seguimiento, que verifican la captura en tiempo real de información del programa, a través de eventos *logs* y acumuladores estadísticos.

- **TSK**: El módulo TSK provee de un conjunto de funciones que manipulan objetos de tareas. Las tareas tienen una prioridad más alta que los hilos *idle* y más baja que las interrupciones por hardware y software.

  Las tareas representan hilos independientes de control, que de forma conceptual, ejecutan funciones en paralelo dentro de un sólo programa en C. Se puede asignar hasta 15 niveles de prioridad a las tareas por medio del TSK.

Herramienta de configuración del DSP/BIOS

La herramienta de configuración del DSP/BIOS permite a los desarrolladores seleccionar módulos del *kernel* y controlar un gran rango de parámetros de

configuración, accedidos por el *kernel* del DSP/BIOS en tiempo de ejecución.

La herramienta de configuración del DSP/BIOS (figura 6.20) sirve como un editor visual de propósito especial, para crear y asignar atributos a objetos individuales del *kernel*, en tiempo de ejecución (*threads*, *streams*, etc.), usados en programas de aplicación en conjunto con llamadas de APIs del DSP/BIOS. La herramienta de configuración permite a los desarrolladores la capacidad de declarar, en forma estática, y configurar los objetos del *kernel* durante el desarrollo del programa. La declaración de los objetos del *kernel* usando la herramienta de configuración, produce objetos estáticos, que existen durante todo el programa. También se puede crear y eliminar, en forma dinámica, a muchos objetos del *kernel* durante la ejecución. Sin embargo, la creación de objetos de forma dinámica requiere de código adicional para soportar las operaciones dinámicas. La declaración de objetos en forma estática, minimiza la memoria, debido a que no incluyen código adicional de creación.

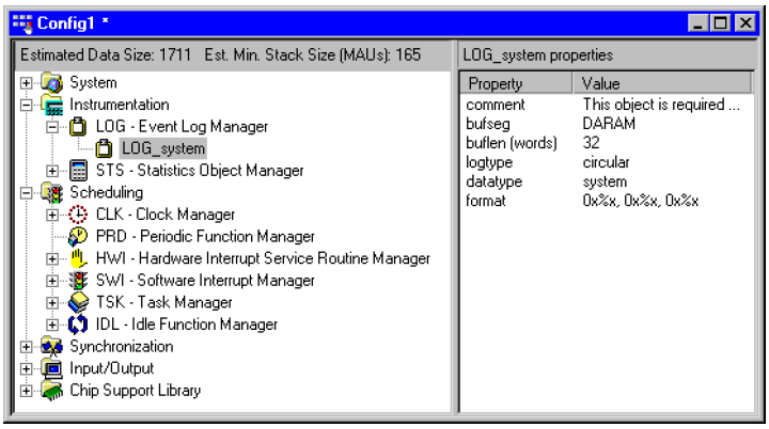


Figura 6.20: Ventana de configuración del DSP/BIOS.

Herramienta de análisis en tiempo real del DSP/BIOS

La característica de análisis en tiempo real del DSP/BIOS (RTA), mostrada en la figura 6.21, proporciona la capacidad de probar, seguir y monitorizar una aplicación en el DSP, durante el curso de su ejecución. Estas utilidades se respaldan en una conexión física JTAG (empleada por el debugger) y se emplea como un enlace de comunicación de baja velocidad (aunque en tiempo real) entre la tarjeta y la PC.

El RTA del DSP/BIOS requiere la presencia del *kernel* del DSP/BIOS dentro de la tarjeta. Además para proveer servicios en tiempo real para la aplicación, el *kernel* del DSP/BIOS proporciona soporte para comunicación en tiempo real

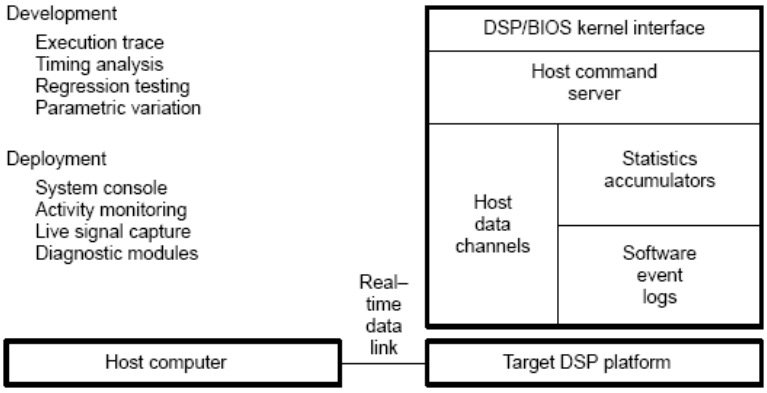


Figura 6.21: Análisis y captura en tiempo real.

con el host (PC) a través de su enlace físico. Con una simple estructuración de la aplicación alrededor de las APIs del DSP/BIOS y objetos de estadística creados (que proporcionan soporte básico de multitarea y de entrada y salida), los desarrolladores equipan automáticamente la tarjeta para la captura y transferencia de información, en tiempo real, controladas por las herramientas visuales de análisis en el IDE del CCStudio.

Las herramientas de análisis en tiempo real del DSP/BIOS, pueden ser accedidas a través de la barra de herramientas del DSP/BIOS (figura 6.22).

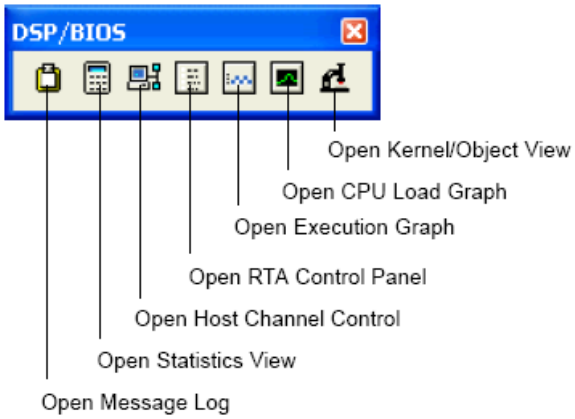


Figura 6.22: Barra de herramientas de RTA DSP/BIOS.

La descripción de cada uno de los elementos es:

- **Message Log**. Despliega una secuencia ordenada (en el tiempo) de even-

tos escritos a objetos LOG de *kernel* por hilos independientes, en tiempo real. Este es útil para seguir todo el flujo de control del programa. La figura 6.23 muestra la ventana de visualización de los eventos LOG en el Code Composer Studio (CCS).

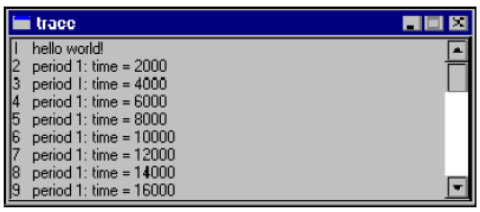


Figura 6.23: Ventana de mensajes LOG.

- **Statistics View**. Despliega un resumen estadístico reunido en objetos de acumulación de *kernel*, reflejando en elementos dinámicos del programa, con un contador simple y valores de datos cambiantes en el tiempo, en intervalos de procesamiento transcurridos para hilos independientes. El programa, en el DSP, acumula las estadísticas explícitamente a través de llamadas de APIs del DSP/BIOS o de forma implícita por el *kernel*, cuando se programa los hilos para ejecución o realizar operaciones de E/S. La figura 6.24 muestra la ventana para visualizar las estadísticas en el CCStudio.

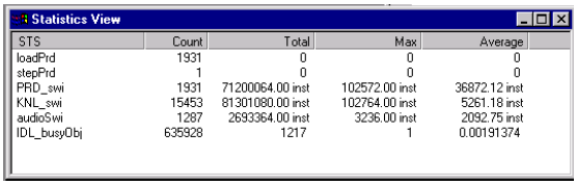


Figura 6.24: Ventana vista de estadísticas.

- **Host Channel Control**. Despliega los canales host definidos por su programa. Se puede usar esta ventana para unir archivos a los canales, comenzar transferencia de datos sobre los canales y monitorizar la cantidad de datos transferidos. La figura 6.25 muestra la ventana para controlar los canales del host.

- **RTA Control Panel**. Controla los seguimientos en tiempo real y las acumulaciones de estadísticas en los programas en el DSP. De hecho, este permite a los desarrolladores controlar el grado de visibilidad en la ejecución del programa en tiempo real. Por defecto, todos los seguimientos se encuentran habilitados. La figura 6.26 muestra el cuadro de dialogo del panel de control del RTA.

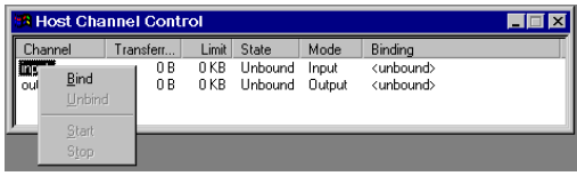


Figura 6.25: Ventana de control de canal host.

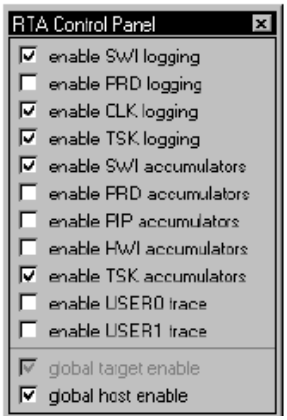


Figura 6.26: Cuadro de dialogo del panel de control del RTA.

- **Execution Graph**. La gráfica de ejecución es una gráfica especial usada para desplegar información de los procesos en ejecución (SWI, PRD, TSK, SEM y CLK). Mediante esta gráfica se puede ver el tiempo y el orden en el cual los procesos se ejecutan. Las líneas azules (más oscuras) indican que el proceso está corriendo actualmente, es decir, el proceso utiliza el CPU. La figura 6.27 muestra la ventana de la gráfica de ejecución.

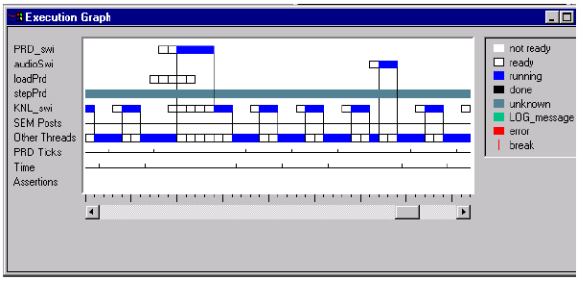


Figura 6.27: Ventana de la gráfica de ejecución.

- **CPU Load Graph**. Despliega una gráfica de la carga de procesamiento del CPU. La carga del CPU es definida como porcentaje de ciclos de instrucción que el CPU consume cuando la aplicación se encuentra trabajando. Es decir, el porcentaje del total de veces que el CPU está:

  - Corriendo interrupciones de hardware, interrupciones de software, tareas o funciones periódicas.
  - Realizando E/S con el host.
  - Ejecutando cualquier rutina del usuario.

  La ventana de carga del CPU se muestra en la figura 6.28

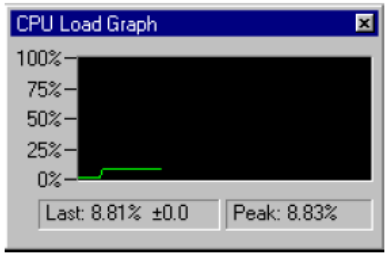


Figura 6.28: Ventana de carga del CPU.

- **Kernel/Object View**. Despliega la configuración, estado y estatus de los objetos del DSP/BIOS que actualmente se corren en la aplicación. Esta herramienta muestra los objetos configurados en forma dinámica y estática. La figura 6.29 muestra la vista de objetos del kernel.

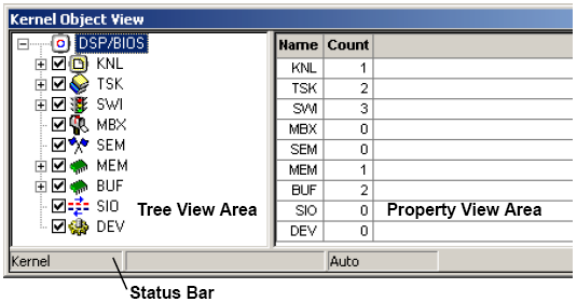


Figura 6.29: Ventana de carga del CPU.

6.6.5. Intercambio de datos en tiempo real (RTDX)

El intercambio de datos en tiempo real (RTDX), proporciona una visibilidad continua de la forma que operan las aplicaciones con DSPs, en el mundo real. Los *plug-ins* del RTDX permiten a los desarrolladores de sistemas, transferir datos entre una computadora host y el dispositivo DSP sin interferir con la aplicación. Los datos pueden ser analizados y visualizados en la computadora host usando un cliente de automatización OLE. Esto reduce el tiempo de desarrollo, proporcionando una representación realista de la manera en que opera el sistema.

El RTDX se compone de las partes que se encuentran en la computadora host y en la aplicación con el DSP. Una librería pequeña de software RTDX se ejecuta en el DSP. La aplicación DSP llama a las funciones de API en la librería, para la transferencia o recepción de datos. La librería hace uso de un emulador para mover los datos, desde o hacia la plataforma host vía una interface JTAG. La transferencia de datos al host ocurre en tiempo real, mientras la aplicación DSP se encuentra corriendo.

En la plataforma host, opera una librería host RTDX en conjunto con el Code Composer Studio. Las herramientas de despliegue y análisis se comunican con RTDX, por medio de una API de Modelo Objeto componente (COM), para recibir y/o enviar datos a la aplicación DSP. De esta forma, se pueden usar cualquier paquete de software de despliegue estándar, como por ejemplo:

- LabVIEW de *National Instruments*
- Herramientas de graficación en tiempo real de *Quinn-Curtis*
- Microsoft Excel

De forma alternativa, se puede desarrollar una aplicación en Visual Basic o

Visual C++, para diseñar la forma de visualizar los datos.

El flujo de datos RTDX

El RTDX forma una tubería de dos vías entre la aplicación con DSPs y el cliente host. Esta tubería de datos consiste en una combinación de componentes de hardware y software como muestra la figura 6.30.

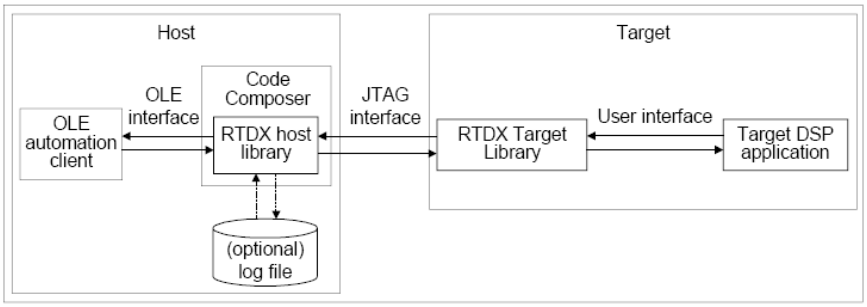


Figura 6.30: El flujo de datos RTDX entre el host y la aplicación DSP.

Flujo de datos de la aplicación DSP al host

Para guardar datos en la aplicación DSP, se debe declarar un canal de salida y escribir los datos a este, usando rutinas definidas en la interface de usuario. Los datos son grabados inmediatamente en un buffer RTDX, definido en la librería RTDX de la aplicación DSP. El dato en el buffer es enviado a la computadora host por medio de la interface JTAG.

La librería host RTDX recibe el dato a través de la interface JTAG y lo almacena. La computadora host guarda el dato en un buffer de memoria o en un archivo log RTDX (dependiendo del modo de almacenamiento RTDX especificado).

El dato puede ser recuperado por cualquier aplicación host que sea un cliente de automatización OLE. Algunos ejemplos típicos de aplicaciones host con capacidad OLE son:

- Aplicaciones en Visual Basic
- Aplicaciones en Visual C++
- Lab View

- Microsoft Excel
- Matlab

Típicamente, un cliente de automatización OLE RTDX es una interface gráfica que permite visualizar los datos en una mejor forma.

Flujo de datos del host a la aplicación DSP

Para que la aplicación DSP reciba los datos del host, se debe declarar primero un canal de entrada y pedir los datos a través de este, usando rutinas definidas en la interface de usuario. La solicitud del dato es almacenada en el buffer RTDX en la aplicación DSP y enviada al host por medio de la interface JTAG.

Un cliente de automatización OLE puede enviar los datos a la aplicación DSP, usando la interface OLE. Todos los datos a ser enviados al DSP, son escritos en un buffer de memoria dentro de la librería RTDX del host. Cuando esta librería recibe y lee la petición de la aplicación DSP, el dato en el buffer del host es enviado al DSP por medio de la interface JTAG. El dato es escrito en la localidad de petición en el DSP en tiempo real. El host notifica a la librería RTDX de la aplicación DSP cuando la operación esta completada.

La interface de usuario de la librería RTDX para aplicación DSP

La interface de usuario de la librería RTDX de la aplicación DSP proporciona un método seguro para intercambiar datos entre la aplicación DSP y la librería RTDX del host.

Los tipos de datos y las funciones definidas por la interface de usuario, manejan las siguientes acciones:

- Capacita a la aplicación DSP para enviar datos a la librería RTDX del host.
- Habilita a la aplicación DSP para pedir un dato a la librería RTDX del host.
- Proveer del buffer de datos en la aplicación DSP. Una copia del dato es almacenado en un buffer en la aplicación DSP, previo a ser enviado al host. Esta acción ayuda para asegurar la integridad del dato y minimizar la interferencia del tiempo real.

- Proporciona una interrupción segura. Se puede llamar a la rutina que se define en la interface del usuario para el manejo de la interrupción.
- Asegura la correcta inicialización del mecanismo de comunicación. Es un requerimiento, que sólo un *datum*[6] a la vez, puede ser intercambiado entre el host y la aplicación DSP, usando la interface JTAG. Las rutinas definidas en la interface de usuario, manejan el tiempo de llamadas en la interface de bajo nivel.

La interface OLE del host RTDX

La interface OLE describe los métodos que habilitan un cliente, de automatización OLE, para comunicarse con la librería RTDX del host.

Las funciones definidas en la interface OLE son:

- Habilitan un cliente de automatización OLE para acceder al dato, que fue grabado en un archivo log RTDX o almacenado en un buffer por la librería host RTDX
- Capacita un cliente de automatización OLE para enviar un dato a la aplicación DSP por medio de la librería host RTDX

[6] Un *datum* es la unidad mínima direccionable para un procesador, que puede ser leída o almacenada en memoria.

Capítulo 7

Entrenamiento

El sistema de reconocimiento de voz de palabras aisladas mediante el DSP, se dividió en dos etapas para su construcción: la etapa de entrenamiento y la etapa de reconocimiento. La etapa de entrenamiento se basa principalmente, en la obtención de patrones de referencia de un conjunto de palabras de entrenamiento. La etapa de reconocimiento, utiliza el conjunto de patrones de referencia, y por medio de una probabilidad o distorsión, se obtiene los patrones que más se aproximan a una palabra pronunciada, que finalmente, estos determinarán la palabra reconocida.

Se utilizaron dos técnicas para implementar el sistema de reconocimiento de voz en el DSK: una utiliza cuantización vectorial (VQ); y la otra, modelos ocultos de Markov (HMM).

En este capítulo se describirá todos los procedimientos que se llevaron a cabo en la etapa de entrenamiento. En el capitulo siguiente, se describirá la etapa de reconocimiento con más detalle.

7.1. Descripción general de la etapa de entrenamiento

La etapa de entrenamiento se realizó, una parte en el DSK y otra en una computadora personal (PC). En el DSK, se obtuvieron los vectores de características (Coeficientes Mel-Cepstral) de cada una de las palabras de entrenamiento, estos fueron almacenaron en archivos independientes y enviados como entrada, a los algoritmos de entrenamiento implementados en la PC. En la computadora

personal, se realizaron los algoritmos de entrenamiento VQ y HMM, estos toman los archivos con los vectores de características y obtienen los patrones de comparación, utilizados en la etapa de reconocimiento. La figura 7.1 describe, de forma general, los pasos realizados en la etapa de entrenamiento.

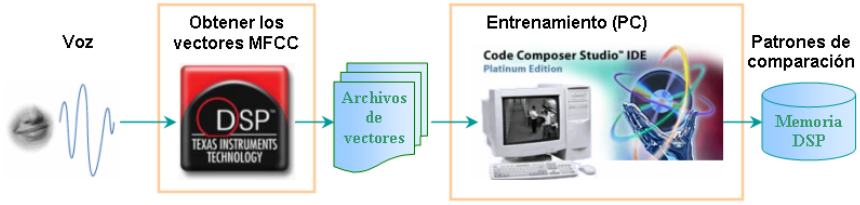


Figura 7.1: Diagrama general de la etapa de entrenamiento.

7.1.1. Descripción de los módulos en el DSK

La primera fase en la etapa de entrenamiento, es obtener los coeficientes Mel-Cepstral (MFCC) mediante el DSK. Debido a que el sistema de reconocimiento, debe realizarse en tiempo real, es necesario que se obtengan los vectores MFCC de la misma manera. De forma general, los pasos para obtener los coeficientes MFCC en el DSK, se muestran en la figura 7.2; a continuación se describen:

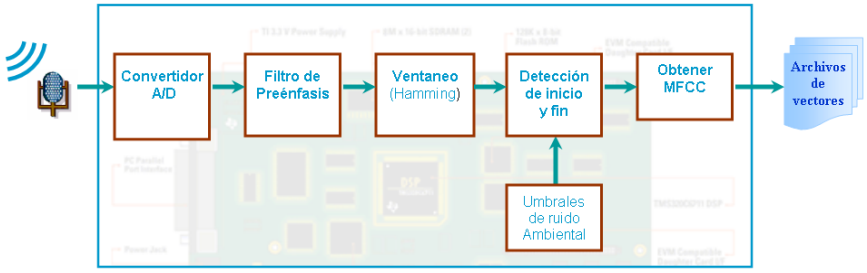


Figura 7.2: Diagrama de bloques de los procesos realizados en el DSK.

- **Convertidor A/D**. Este módulo convierte la señal analógica de la voz, en una señal digital.
- **Filtro de preénfasis**. En este módulo, se aplica un filtro de preénfasis a la señal de voz. Debido a que las altas frecuencias tienen muy poca energía, en los sonidos fricativos, es necesario realzarlos por medio de este filtro.
- **Ventaneo**. En esta etapa, la señal de voz es dividida en intervalos de tiempo corto con el uso de ventanas. Como la señal de voz varían muy poco

sus propiedades estadísticas en intervalos de tiempo pequeños, entonces, se puede suponer como un proceso estacionario para este intervalo de tiempo. Esto conduce a métodos de procesamiento en tiempo corto, en los cuales sólo una parte de la señal, es aislada y procesada en pequeños segmentos a los que se denomina tramas. Se pueden utilizar varios tipos de ventanas (por ejemplo: *Hamming*, *Hanning*, etc.), para tener una mejor representación frecuencial, en particular se utiliza una ventana *Hamming*.

- **Umbrales del ruido ambiental**. Para almacenar y procesar sólo muestras de voz, es necesario obtener parámetros que caractericen al ruido ambiental. Este módulo calcula los umbrales del ruido utilizados en la detección del inicio y el fin de la palabra.

- **Detección de inicio y fin**. Este módulo recorta la señal de voz a límites bien definidos, donde se tiene la seguridad de la existencia de voz. Con esto, se evita necesidad de realizar cálculos y almacenar tramas que no contienen información. Es decir, sólo se procesa la pronunciación de una palabra.

- **Obtener MFCC**. Las tramas de la señal de voz, son caracterizadas por los coeficientes Mel-Cepstral que son una variante de la transformación homomórfica. Este módulo realiza todos los procedimientos apropiados para la obtención de los coeficientes MFCC en cada trama.

7.1.2. Descripción de los módulos en la PC

En la computadora personal, se realizó el entrenamiento a partir de los vectores de características MFCC, se emplearon varias pronunciaciones de las palabras de entrenamiento. Las técnicas utilizadas en esta fase son: Cuantización Vectorial (VQ) y Modelos Ocultos de Markov (HMM). La primera emplea los vectores de características, para obtener un conjunto de centroide por medio de el algoritmo de agrupamiento *K-Medias*. En la segunda técnica (HMM), se modela la variaciones de los vectores de características, por medio de dos procesos estocásticos incrustados, y se obtiene los parámetros estadísticos de los modelos acústicos con la maximización de la verosimilitud.

La figura 7.3 muestra el diagrama de bloques para el entrenamiento utilizando VQ. El primer módulo se encarga de segmentar los archivos de coeficientes MFCC en forma lineal. Todos los vectores que pertenecen a cada segmento, son enviados a un algoritmo de agrupamiento (*K-Medias*) en el módulo Cuantización vectorial. Los patrones de comparación se obtienen para cada uno de los segmentos.

El segundo método de entrenamiento HMM (figura 7.4), además de utilizar los vectores de características MFCC, necesita información fonética de las pa-

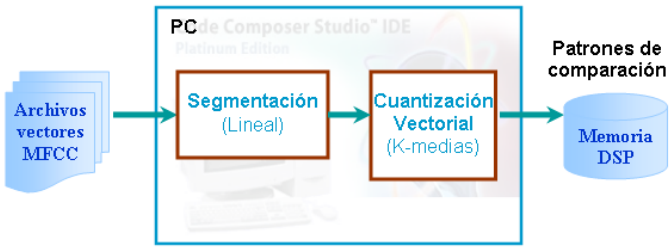


Figura 7.3: Diagrama de bloques de los procesos realizados para el entrenamiento con VQ.

labras de entrenamiento (dada en el diccionario de palabras) y la definición del modelo acústico para cada una de las palabras en diccionario. El primero bloque, se encarga de inicializar, de forma plana, los parámetros estadísticos empleados en el algoritmo *Baum-Welch*. El bloque Algoritmo de *Baum-Welch*, estima los parámetros estadísticos utilizando el algoritmo de máxima similitud, también se le conoce como algoritmo *Forward-Backward*. Una vez que ha convergido el procedimiento, los parámetros estadísticos, de cada palabra, serán los patrones de comparación en la etapa de reconocimiento.

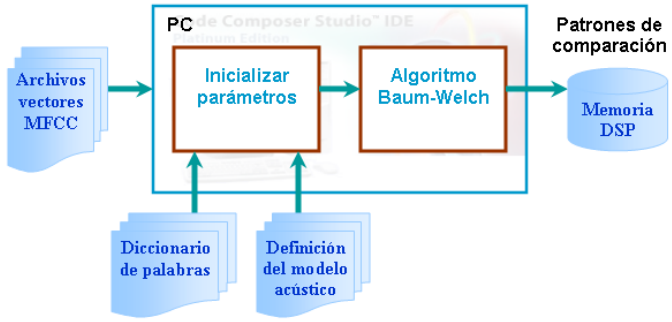


Figura 7.4: Diagrama de bloques de los procesos realizados para el entrenamiento con HMM.

7.2. Algoritmo realizado en el DSK

Para la implementación en la tarjeta de desarrollo (DSK), se elaboró un algoritmo siguiendo el diagrama de estados mostrado en la figura 7.5. El programa detecta la existencia de voz y procesa la señal en tiempo real, para obtener los vectores MFCC correspondientes a la pronunciación. Conforme se articula la palabra, el sistema obtiene los vectores de características, hasta que se detecta el

final de la palabra. Este algoritmo es una modificación del algoritmo elaborado por [Nieto, López 02] para obtiene los vectores LPC en tiempo real.

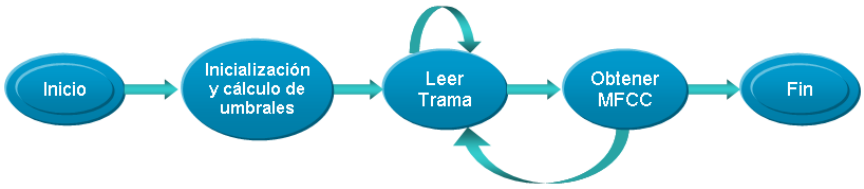


Figura 7.5: Diagrama de estados del algoritmo implementado en el DSK.

Cuando inicia el programa en el DSK, el sistema entra de inmediato al estado *Inicialización y cálculo de umbrales*. En este estado, se configura el puerto serie y el *codec* para poder leer la señal digital. También, calcula los umbrales del ruido ambiental, utilizados para detectar la presencia de voz.

Después de la configuración e inicialización, el sistema cambia al estado *Leer trama*, que es una rutina de atención al puerto serie. Cada vez que llega una muestra de la señal de voz, esta rutina la procesa, con un filtro de preénfasis, y la almacena en un buffer temporal. Continua el proceso hasta que se almacena una trama completa, en donde se activa una interrupción por software y el sistema cambia al estado siguiente.

En el estado *Obtener MFCC*, determina si la trama capturada es parte de una palabra, si es así, la procesa para obtener los coeficientes MFCC, de lo contrario se desecha. Además, determina si la trama es el final de la palabra, en cuyo caso detendrá la ejecución; si de lo contrario, la trama no es el final de la palabra, el sistema cambiará al estado *Leer trama*, para continuar nuevamente con la lectura de la trama.

A continuación se describen con mayor detalle los procesos que se ejecutan en cada estado:

- **Estado *Inicialización y cálculo de umbrales***: Se realiza la configuración del puerto serie McBSP y establecer una comunicación con el convertidor TLC320AD535 incluido en el DSK. El *codec* es configurado para trabajar a 8 KHz y 16 bits como ancho de palabra. El manual [SLAS202B] de *Texas Instruments*, proporciona una mayor descripción del convertidor TLC320AD535.

  En el cálculo de los umbrales del ruido, se capturan 16 tramas del ruido ambiental. Cada trama tiene una duración aproximada de 0.025625 [s] (205 muestras). Las muestras son pasadas por un filtro de preénfasis cuyo coeficiente α es igual a 0.97. Posteriormente, se obtiene la magnitud y los cruces por cero para cada una de las tramas. Basándose en el algoritmo de

Rebiner-Sambur, en la detección de inicio y fin de un palabra, se calcula el umbral inferior de energía y el umbral de cruce por ceros.

Cuando se ha terminado el proceso de iniciación, se activa la interrupción de hardware para cambiar al siguiente estado.

- **Estado *Leer trama***: Una vez que se tienen los umbrales del ruido, sigue la lectura de las muestras de la señal de voz, el filtrado de la señal y el tramado. En este estado se realizan estas operaciones para cada una de las tramas.

  El estado es realizado con la interrupción de hardware de recepción del puerto serie: cada vez que el convertidor A/D y D/A lee una muestra, esta se pasa a través de un filtro de preénfasis ($\alpha = 0{,}97$) y se almacena en un buffer temporal. Cuando se completa una trama, se activa una interrupción de software para cambiar al siguiente estado.

  Las tramas almacenadas tienen una duración de 0.25625 [s], es decir, 205 muestras para una frecuencia de muestreo de 8 KHz y utilizan un corrimiento de 0.01 [s] (80 muestras de corrimiento ó 125 muestras de traslape), por lo que se activa la interrupción de software cada 0.01 [s].

- **Estado *Obtener MFCC***: Cuando se ha completado una trama, se activa el estado *Obtener MFCC* por medio de una interrupción de software. En este estado, se determina si la trama capturada contiene voz, en cuyo caso se obtienen los vectores MFCC; de lo contrario, si se detecta que la trama contiene ruido, se desecha.

  Para detectar si una trama contiene voz o ruido ambiental, se obtiene la magnitud promedio y los cruces por cero de la trama, que junto con los umbrales del ruido ambiental calculados en el primer estado, se determina si la trama contiene voz o silencio, también se identifica el inicio o final de una pronunciación. El algoritmo de identificación del inicio y fin de la palabra se describe con mayor detalle en la tesis elaborada por [Nieto, López 02], que es una modificación al algoritmo de *Rabiner-Sambur* para que funcione en tiempo real.

  La figura 7.6 muestra el diagrama de bloques para obtener los coeficientes MFCC a partir de la trama dada. Los procedimientos son los siguientes:

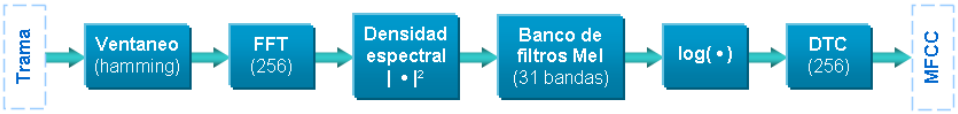


Figura 7.6: Diagrama de bloque para obtener de los vectores MFCC.

- *Ventaneo*: la trama es multiplicada punto a punto por una ventana de hamming de 205 muestras.

- *FFT*: este bloque calcula el espectro en frecuencia mediante la Transformada Rápida de Fourier (FFT). Se empleó una FFT *radix-2* con 256 puntos y con decimación en frecuencia. En este mismo bloque se realizó el ordenamiento de los datos por medio del algoritmo de *bits reverse*.
- *Espectro de potencia*: este bloque calcula el espectro de potencia de la trama, utiliza los coeficientes de Fourier del punto previo. Se obtiene a través de la ecuación 7.1.

$$S[k] = real\big(X[k]\big)^2 + imag\big(X[k]\big)^2 \tag{7.1}$$

- *Banco de filtros en escala Mel*: este bloque calcula el promedio del espectro de potencia, alrededor de las frecuencias centrales en escala mel. Para ello se utilizan 31 filtros triangulares de área unitaria traslapados (ecu. 7.2) que filtran al espectro de potencia en 31 bandas. La figura 7.7 muestra el mapeo de la frecuencia lineal y la frecuencia Mel, para los 31 filtros.

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \dfrac{2\big(k - f[m-1]\big)}{\big(f[m+1] - f[m-1]\big)\big(f[m] - f[m-1]\big)} & f[m-1] \le k < f[m] \\ \dfrac{2\big(f[m+1] - k\big)}{\big(f[m+1] - f[m-1]\big)\big(f[m+1] - f[m]\big)} & f[m] \le k \le f[m+1] \\ 0 & k > f[m+1] \end{cases} \tag{7.2}$$

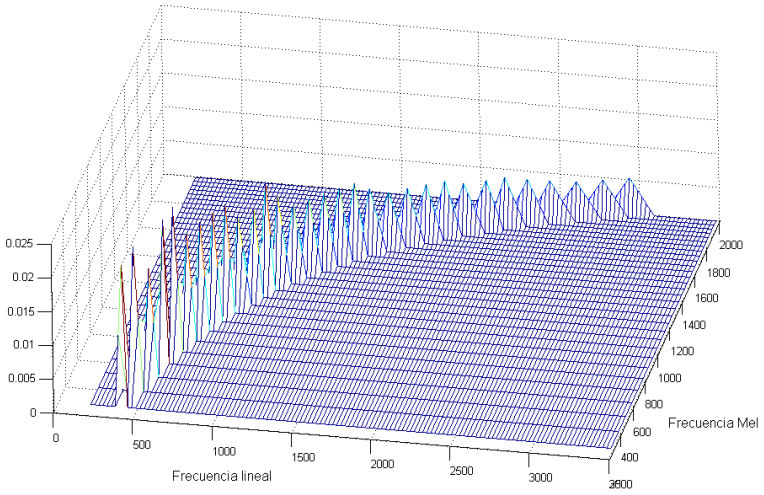


Figura 7.7: Filtros triangulares utilizan el promedio del espectro alrededor de la frecuencia central en cada filtro.

La forma de realizar el filtrado es, multiplicar el espectro de potencia para cada uno de los filtros triangulares, como muestra la ecuación 7.3.

$$\widetilde{S}[m] = \left[\sum_{k=0}^{N-1} S[k] H_m[k] \right] \qquad \text{Para } m = 0, 1, 2, \cdots, M-1 \qquad (7.3)$$

donde:

N : Es el número de puntos de resolución del espectro de potencia ($256/2 = 128$ puntos)

M : Es el número de bandas en el banco de filtros (31).

- *Logaritmo del espectro de potencia*: se obtiene el logaritmo natural, a la intensidad de las bandas del espectro de potencia filtrado.
  a cada uno de los puntos del espectro de potencia filtrado, se le obtiene el logaritmo natural.
- *DTC*: finalmente, para obtener los vectores Mel cepstral, se calcula la Transformada Coseno Discreta Tipo II (DCT-II), como muestra la ecuación 7.4.

$$MelCepst_t[n] = \sum_{m=0}^{M-1} \ln\left(\widetilde{S}[m]\right) \cos\left(\pi n (m+1/2)/M\right) \quad n = 0, 1, \cdots, D-1 \qquad (7.4)$$

donde:

D: Es el número de coeficientes mel cepstral, (13 coeficientes)

t: Número de la trama.

$MelCepst_t$: Vector de coeficientes Mel Cepstral para la trama t sin normalizar.

Cuando la trama no es parte de una pronunciación de voz se desecha y el programa regresa al estado *Leer trama*. En el caso de que la trama sea parte de una palabra de voz, se obtiene sus coeficientes MFCC y el programa regresa al estado *Leer trama*; cuando se detecta que la trama es el fin de la palabra, el sistema cambia al estado *Fin*.

- **Estado *Fin***: En este estado, se normalizan los vectores MFCC como lo indica las siguientes ecuaciones:

$$media[n] = \frac{1}{T} \sum_{t=1}^{T} MelCepst_t[n] \qquad \text{Para } n = 0, 1, \cdots, D-1 \qquad (7.5)$$

$$MFCC_t[n] = MelCepst_t[n] - media[n] \qquad \begin{matrix} n = 0, 1, \cdots, D-1 \\ t = 1, 2, \cdots, T \end{matrix} \tag{7.6}$$

donde:

D: Es el número de coeficientes mel cepstral, (13 coeficientes)

T: Número total de tramas en la pronunciación.

Finalmente, se detiene el algoritmo para poder almacenar los coeficientes MFCC correspondientes a la pronunciación, en un archivo de computadora.

7.3. Algoritmo realizado en la computadora

El entrenamiento se realizó utilizando dos técnicas diferentes: *Cuantización Vectorial* (VQ) y *Modelos Ocultos de Markov* (HMM). Cada una ellas fueron programadas con *scripts* hechos en MATLAB.

En ambas técnicas se utilizaron ocho palabras de entrenamiento o comandos: *Abajo*, *Adelante*, *Alto*, *Arriba*, *Atrás*, *Derecha*, *Izquierda* y *Sigue*. Cada palabra fue pronunciada veinte veces y procesada en el DSP con el fin de obtener los vectores de características MFCC. Posteriormente, cada repetición se guardo en archivos individuales dentro de una PC.

7.3.1. Entrenamiento con VQ

Esta técnica de entrenamiento utiliza Cuantización Vectorial (VQ) para representar, un conjunto grande de vectores, a través de un conjunto pequeño llamados centroides; utiliza para ello una medida de distorsión.

La figura 7.8 muestra todo el proceso de entrenamiento de una palabra, utilizando VQ. Los archivos de vectores de características con las repeticiones de una palabra, se dividieron en segmentos temporales, de forma lineal. Todos los vectores de un mismo segmento, son agrupados y enviados al algoritmo *K-Medias* para obtener los centroides correspondientes al mismo segmento.

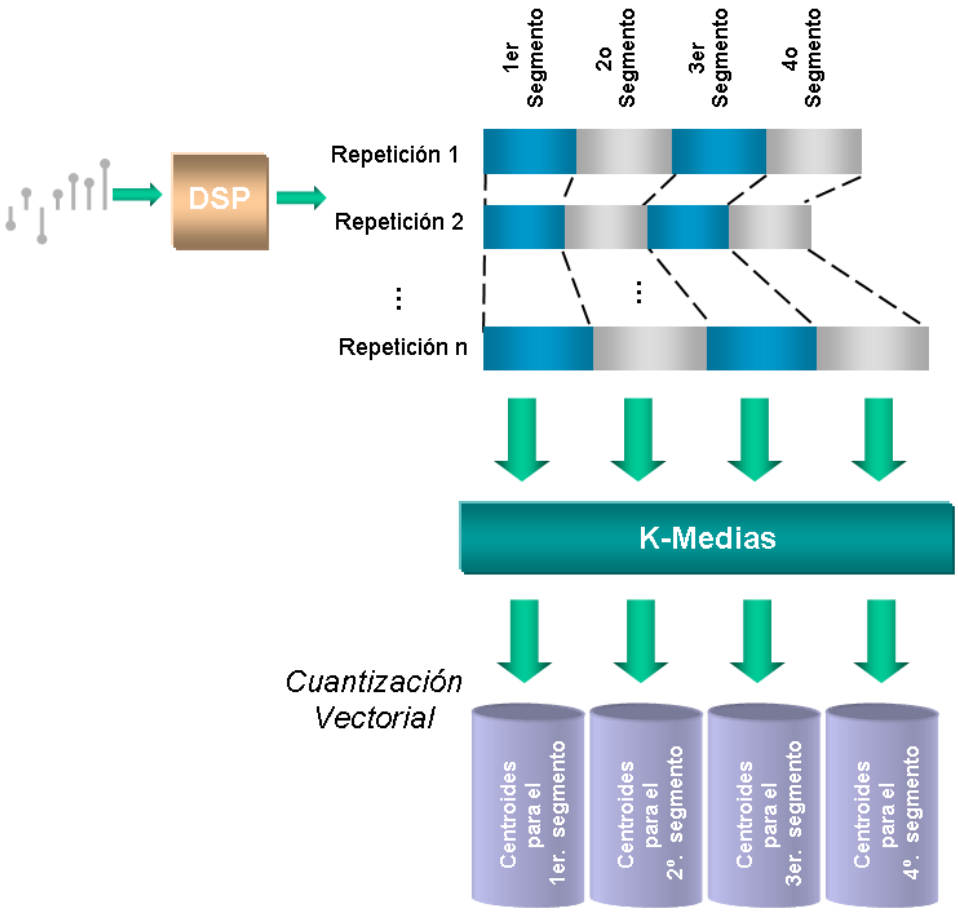


Figura 7.8: Esquema del proceso de entrenamiento utilizando VQ.

El entrenamiento con VQ, básicamente se divide en dos fases: *Segmentación* y *Cuantización Vectorial* (figura 7.3, pág. 128).

- **Segmentación**. Para cada una de las palabras de entrenamiento, se tomaron los archivos con los vectores MFCC de las repeticiones. Se dividieron los vectores en cuatro segmentos lineales en el tiempo, y se concatenaron todos los coeficientes pertenecientes a un mismo segmento, para formar los vectores de entrada en el algoritmo *K-Medias*.

- **Cuantización vectorial**. Los vectores pertenecientes a un mismo segmento, se utilizan como vectores de entrada en el algoritmo de agrupación *K-Medias*[1]. Este algoritmo calcula los vectores centroides representativos de cada segmento.

  Se utiliza $K = 16$ centroides en cada segmento; los centroides iniciales se obtuvieron de forma aleatoria.

  Finalmente, los centroides son almacenados en la memoria del DSP y empleados como patrones de comparación en la etapa de reconocimiento. Con el almacenamiento de los centroides, termina la fase de entrenamiento para esta técnica.

7.3.2. Entrenamiento con HMM

La segunda técnica realiza el entrenamiento con los Modelos Ocultos de Markov (HMM). Esta técnica constituye una de las más utilizadas y de mayor éxito en el reconocimiento de voz. Principalmente, ha permitido modelar adecuadamente la gran variabilidad de la señal de voz en el tiempo.

Para realizar el entrenamiento con HMM se utilizan: los archivos con los *vectores MFCC*; un *Diccionario de Palabras*, que contiene una lista con todas las palabras de entrenamiento y la cantidad de fonemas de las palabras; también incluye la *Definición del Modelo Acústico*, que determina el modelo HMM de las palabras de entrenamiento, es decir, la cantidad de estados utilizados para modelar a un palabra, las probabilidades de transición entre ellos y las densidades de probabilidad de observación para cada estado. Este conjunto de parámetros sirven como entrada a la fase de *Inicializar parámetros*. Finalmente, los parámetros estadísticos son estimados en la fase *Algoritmo Baun-Welch* (ver figura 7.4).

[1] El algoritmo *K-Medias* se describe detalladamente en la sección 4.1.3 en la página 55

Definición del modelo acústico

Las palabras de entrenamiento son modeladas tomando en cuenta su composición fonética. Cada uno de los fonemas es representado por un modelo oculto de Markov de tres estados, tal como muestra la figura 7.9. El modelo completo de una palabra, se obtiene concatenando los modelos HMM de cada uno de los fonemas que conforman la palabra. Por ejemplo, el comando "ABAJO" se compone de cinco fonemas (/a/ /b/ /a/ /x/ /o/), por tanto, el modelo tendrá 15 estados más un estado no emisor que marcará el final del modelo. La figura 7.10 muestra el modelo HMM utilizado para la palabra "ABAJO". El apéndice A muestra los modelos HMM utilizados para todas las palabras de entrenamiento.

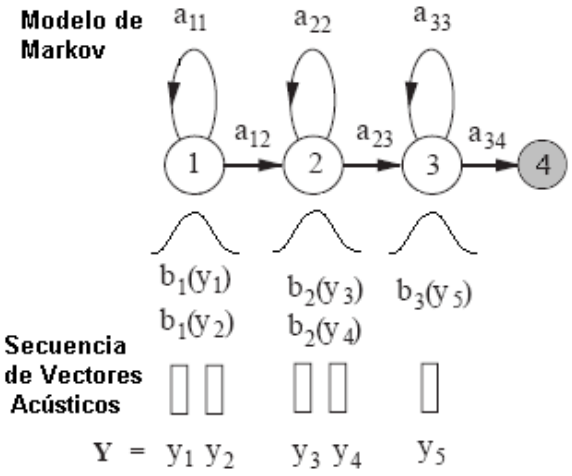


Figura 7.9: Modelo de Markov para representar a un fonema.

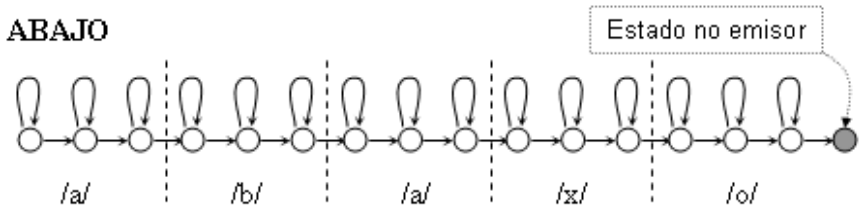


Figura 7.10: Modelado de la palabra "Abajo" usando HMM.

Se utilizaron densidades de probabilidad gaussianas (definidas por la ecuación 7.7) para representar las probabilidades de observación de los estados.

$$b_j(O) = \frac{1}{\sqrt{(2\pi)^d |\mathbf{U}_j|}} \exp\big(-\frac{1}{2}(\mathbf{x}_t - \mu_j)'\mathbf{U}_j^{-1}(\mathbf{x}_t - \mu_j)\big) \tag{7.7}$$

donde:

d : Es la dimensión de los vectores MFCC. En este caso, se utilizaron 13 coeficientes mel cepstral.

$\mathbf{U}_j$: Matriz de covarianza con elementos $var_{j,i}$.

Si se toman los elementos de la diagonal principal de la covarianza $\mathbf{U}_j$:, se obtiene;

$$b_j(O) = \frac{1}{\sqrt{(2\pi)^d \prod_{i=1}^{d} var_{j,i}}} \exp\big(-\frac{\sum_{i=1}^{d}(x_{t,i} - \mu_{j,i})^2}{2var_{j,i}}\big) \tag{7.8}$$

donde:

$var_{j,i}$: representa la varianza del estado i en la dimensión i.

Debido a que se reduce considerablemente los cálculos en 7.8, el entrenamiento utiliza esta densidad para representar las probabilidades de observación en los modelos de las palabras.

Inicialización de parámetros

Antes de realizar el entrenamiento, todos los parámetros son inicializados con valor plano. Debido a que en cada estado existen sólo dos transiciones (una transición al mismo estado y otra al estado siguiente), sólo los elementos $a_{i,i}$ y $a_{i,i+1}$ de la matriz $\mathbf{A}$ son utilizados, y por tanto, inicializados con el valor 1/2.

Para inicializar los parámetros de las densidades gaussianas (la media $\mu_{p,j}$ y la varianza $\mathbf{U}_{p,j}$), se calculo la media global ($\mu_{glob,p}$) y la varianza global ($\mathbf{U}_{glob,p}$), que se obtuvieron con los vectores MFCC de las repeticiones de cada palabra de entrenamiento. Las ecuaciones 7.9 y 7.10 muestran el proceso de cálculo de la media y la varianza globales para cada palabra.

$$\mu_{glob,p} = \frac{1}{Q_p} \sum_{r=1}^{R} \sum_{m=1}^{M_{p,r}} \mathbf{X}_{p,r,m} \quad \text{para } p = 1, \cdots, P \tag{7.9}$$

$$\mathbf{U}_{glob,p} = \frac{1}{Q_p} \sum_{r=1}^{R} \sum_{m=1}^{M_{p,r}} \left(\mathbf{X}_{p,r,m} - \mu_{glob,p}\right)^2 \quad \text{para } p = 1, \cdots, P \tag{7.10}$$

donde:

$\mathbf{X}_{p,r,m}$: Vector de coeficientes MFCC de la palabra p, repetición r en el tiempo m.

R : Número de repeticiones de cada palabra (se utilizaron 20 repeticiones en el sistema).

$M_{p,r}$: Número de vectores MFCC de la repetición r en la palabra p.

Q_p : Número total de vectores de coeficientes MFCC en todas las repeticiones de la palabra p, es decir $Q_p = \sum_{r=1}^{R} M_{p,r}$.

P : Número de palabras de entrenamiento

Utilizando las ecuaciones 7.9 y 7.10, se inicializaron los valores de la media y la varianza de la siguiente forma:

$$\mu_{p,j} = \mu_{glob,p} \quad \text{para } j = 1, \cdots, N \tag{7.11}$$

$$\mathbf{U}_{p,j} = \mathbf{U}_{glob,p} \quad \text{para } j = 1, \cdots, N \tag{7.12}$$

donde:

N : es el número de estados en la palabra p.

Algoritmo Baum-Welch

Una vez iniciadas los parámetros, el siguiente paso es, el entrenamiento propiamente dicho. El método usado para este fin es el procedimiento llamado *forward-backward*, también conocido como algoritmo *Baum-Welch*. En los siguientes puntos se describen todos pasos realizados en este algoritmo.

En cada una de las palabras $p = 1, \cdots, P$; se realiza los siguientes pasos:

1. Para cada repetición $r = 1, \cdots, 20$

   a) Se calcula la variable *hacia adelante* $\alpha_t(i)$, con forme al procedimiento *forward* descrito en la sección 4.2.4 con la densidad de probabilidad de la ecuación 7.8 (pág. 137).

   b) Se calcula el factor de escalamiento c_t de la siguiente forma:

$$c_t = \frac{1}{\underset{i=1,\cdots,N}{\text{máx}} \alpha_t(i)} \quad \text{para } t = 1, \cdots, T \tag{7.13}$$

   donde:

   T : es el número de vectores MFCC de observación.

   c) Utilizando el factor de escalamiento, se calcula $\hat{\alpha}_t(i)$

$$\hat{\alpha}_t(i) = c_t \alpha_t(i) \quad \text{para } t = 1, \cdots, T \quad \text{e} \quad i = 1, \cdots, N \tag{7.14}$$

   d) El siguiente punto es el cálculo de la variable *hacia atrás* $\beta_t(i)$, con forme a la sección 4.2.4 y la densidad de probabilidad gaussiana de la ecuación 7.8 (pág. 137). Así mismo, se realiza el escalamiento de $\beta_t(i)$ en el instante t, de la siguiente forma:

$$\hat{\beta}_t(j) = c_t \beta_t(j) \quad \text{para } j = 1, \cdots, N \tag{7.15}$$

   e) Mientras se calculan los valores de $\hat{\beta}_t(j)$ en el instante t, se realizan la acumulación de: $tacc_{p,i,j}$, $mediaAcc_{p,j}$ y $varAcc_{p,j}$. Estos valores se utilizan para el cálculo de la estimación de la matriz de transición, la media y varianza respectivamente. Es importante aclarar que estas variables no sólo acumulan los valores para la observación en curso (**O**) (repetición actual), también acumulan los valores de las demás repeticiones.

   Las siguientes ecuaciones, resumen todo el proceso realizado en la acumulación de los valores.

   - Para el instante $t = T$ y para cada $i = 1, \cdots, N$ y $j = 1, \cdots, N$

$$IniTacc_{p,i,j} = \begin{cases} \hat{\alpha}_T(i) a(i,j) \hat{\beta}_T(j) / \hat{\alpha}_T(N) & \text{Donde } j \text{ es un estado no emisor} \\ 0 & \text{Otro caso} \end{cases}$$

   - Para $i = 1, \cdots, N$ y $j = 1, \cdots, N$

$$tacc_{p,i,j} = IniTacc_{p,i,j} + \sum_{t=T-1}^{1} \hat{\alpha}_t(i) a(i,j) b_j(\mathbf{O}_{t+1}) \hat{\beta}_{t+1}(j) / \hat{\alpha}_T(N) \tag{7.16}$$

- Para $t = T-1, \cdots, 1$ y para cada estado $j = 1, \cdots, N$

$$\xi_p(t,j) = \sum_{i=1}^{N} \hat{\alpha}_t(i) a(i,j) b_j(\mathbf{O}_{t+1}) \hat{\beta}_{t+1}(j) / \hat{\alpha}_T(N) \tag{7.17}$$

- Para cada estado $j = 1, \cdots, N$ y para cada elemento $f = 1, \cdots, 13$

$$mediaAcc_{p,j,f} = \sum_{t=T-1}^{1} \xi_p(t,j) O_{t+1,f} \tag{7.18}$$

$$varAcc_{p,j,f} = \sum_{t=T-1}^{1} \xi_p(t,j) \big(O_{t+1,f} - \mu_{p,j,f}\big)^2 \tag{7.19}$$

f) Se obtiene el logaritmo de la probabilidad de la observación $\mathbf{O}$ en la repetición r y el estado $q = N$ (estado final), dado el modelo λ.

$$logLik(r) = \log\big(P(\mathbf{O}, q = N|\lambda)\big) = \log\big(\hat{\alpha}_T(N)\big) - \sum_{t=1}^{T} \log\big(c_t\big) \tag{7.20}$$

Además, se almacena la cantidad de vectores de observación y la acumulación de $\xi_p(t,j)$ con respecto a t y para cada j:

$$numVect(r) = T \tag{7.21}$$

$$\xi_p(j) = \sum_{t=1}^{T} \xi_p(t,j) \tag{7.22}$$

2. Una vez que se han obtenido todas las acumulaciones para todas las repeticiones, se obtiene los nuevos valores estimados para la media, varianza y matriz de transiciones, de la siguiente forma:

- Para cada estado $i = 1, \cdots, N$ y para cada elemento $f = 1, \cdots, 13$

$$\overline{\mu}_{p,i,f} = \frac{mediaAcc_{p,i,f}}{\xi_p(i)} \tag{7.23}$$

$$\overline{var}_{p,i,f} = \frac{varAcc_{p,i,f}}{\xi_p(i)} \tag{7.24}$$

- Para cada estado $i = 1, \cdots, N$ y para $j = 1, \cdots, N$

$$\overline{a}_{p,i,j} = \frac{tacc_{p,i,j}}{\xi_p(i)} \tag{7.25}$$

3. Se obtiene la verosimilitud logarítmica (*log-likelihood*) por trama, en la iteración *iter*:

$$logLikTrama(iter) = \frac{\sum_{r=1}^{20} logLik(r)}{\sum_{r=1}^{20} numVect(r)} \tag{7.26}$$

4. Se actualizan los parámetros actuales con los parámetros estimados:

$$\mu_{p,i,f} = \overline{\mu}_{p,i,f}, \qquad var_{p,i,f} = \overline{var}_{p,i,f}, \qquad a_{p,i,j} = \overline{a}_{p,i,j} \tag{7.27}$$

   Si es la primera iteración ($iter == 1$), entonces regresa al punto 1, de lo contrario, continua con los puntos siguientes.

5. Se calcula la relación de convergencia, a través del incremento de la verosimilitud con respecto a la iteración previa, de la siguiente forma:

$$rationLik = \frac{(logLikTrama(iter) - logLikTrama(iter-1))}{|logLikTrama(iter-1)|} \tag{7.28}$$

   Si $rationLik < 0{,}02$ entonces ha terminado el entrenamiento para la palabra p, prosiguiendo con las palabras restantes; de lo contrario, se incrementa la *iter* en uno y regresa al paso 1

Cuando se ha terminado el entrenamiento de todas las palabras, los patrones de comparación son almacenados en la memoria del DSP para usarse posteriormente en la etapa de reconocimiento, descrita en el siguiente capitulo.

Capítulo 8

Reconocimiento

8.1. Descripción general de la etapa de reconocimiento.

La segunda fase en la construcción del sistema, es el reconocimiento de comandos en tiempo real usando el DSP C6711. En esta fase, se utilizó el conjunto de patrones de referencia obtenidos en la etapa de entrenamiento, y por medio de una medida de distorsión o una probabilidad, se obtiene los patrones que más se aproximan a una palabra pronunciada. La figura 8.1 muestra, de forma general, el funcionamiento del reconocedor de comandos de voz.

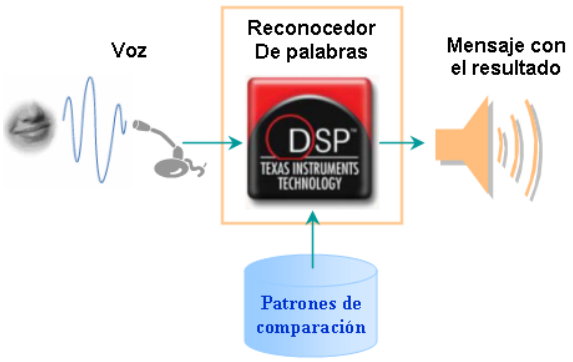


Figura 8.1: Esquema general del sistema de reconocimiento de palabras en el DSP.

Cuando una persona comienza a producir una palabra, el sistema detecta

inmediatamente la existencia de voz, iniciando así, el proceso de obtener sus vectores MFCC. Cuando se detecta el final de la palabra, el sistema inhabilita la captura de la señal de voz, ocupándose sólo realizar la comparación de los vectores MFCC con respecto a los patrones de referencia. Finalmente, se identifica el modelo que más se aproxima a la pronunciación, que será el que determinará la palabra reconocida.

8.2. Procedimiento realizado en el reconocimiento.

La figura 8.2 muestra el diagrama de estados del sistema de reconocimiento de voz en el DSK. Como se observa en el diagrama, es muy parecido al realizado en la fase de entrenamiento, para obtener los vectores de características MFCC en tiempo real (sección 7.2), por tal motivo, los módulos ya conocidos se describe brevemente, y se explica con más detalle el estado *Reconocimiento*.

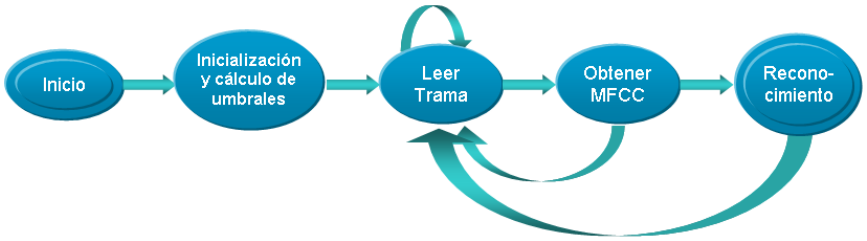


Figura 8.2: Diagrama de estados para el algoritmo de reconocimiento.

Cuando el sistema se enciende, entra inmediatamente al estado *Inicialización y cálculo de umbrales*, en donde establece toda la inicialización necesaria para configurar el McBSP y el codec en el DSK, también se calculan los umbrales del ruido ambiental para la detección del inicio y fin de la palabra. Una vez que se ha terminado con configuración de los periféricos, el sistema cambia al estado *Leer trama*, mediante la habilitación de la interrupciones del McBSP de recepción. Para una descripción más detallada del estado, consulte la sección 7.2 en la página 128.

El estado *Leer trama*, es realizado con una interrupción por hardware y se ejecuta cada vez que llega una muestra de la señal de voz. El estado se encarga de capturar tramas de la señal de voz y realizar un filtrado pre-énfasis. Cuando se ha completado una trama de voz, se habilita una interrupción por software, lo que activa el estado *Obtener MFCC*. Las tramas capturadas presentan un traslape de 125 muestras. Una mejor descripción de lo que realiza el estado se encuentra en la sección 7.2.

El estado *Obtener MFCC*, realiza todos los procedimientos para obtener los vectores Mel cepstral de la trama actual. También, incluye un algoritmo de detección de inicio y fin de la palabra, en tiempo real. Cuando se detectado el final de pronunciación, se activa el estado *Reconocimiento* mediante una interrupción por software. Para mayor información consulte la sección 7.2.

Cuando se ha detectado el final de una palabra, el sistema entra al estado *Reconocimiento*. El procedimiento inhabilita las interrupciones por hardware, evitando la captura de la señal de voz, hasta que termine el reconocimiento. En seguida, se realiza la normalización de los vectores MFCC y posteriormente, se compara los patrones de referencia con respecto a la palabra pronunciada. Se utilizaron dos técnicas para el reconocimiento: la primera obtiene la *mínima distorsión promedio* de la pronunciación, con respecto a un conjunto de centroides, obtenidos en la fase de entrenamiento con VQ. La segunda, calcula la *máxima verosimilitud* de la palabra, con relación a los modelos HMM, previamente entrenados. Bajo estos últimos criterios, se identifica la palabra pronunciada y se emite un mensaje con los resultados obtenidos: "*Palabra desconocida*", cuando las medidas de comparación son mayores a umbrales preestablecidos; "*Repetir palabra*", cuando las medidas de comparación son muy cercanas y puede haber una confusión en el reconocimiento; finalmente el mensaje "*Palabra reconocida...*", se emplea para anunciar la palabra que se ha identificado como la palabra pronunciada. Finalmente, se habilitan las interrupciones de hardware, por lo que el sistema regresa al estado *Leer trama*, listo para reconocer una nueva palabra.

Las técnicas empleadas en el reconocimiento, se realizaron por separado, por tal motivo, se describen a detalle en apartados diferentes en las siguientes secciones.

8.3. Reconocimiento usando VQ.

El reconocedor de comandos de voz usando VQ, utiliza la *mínima distorsión promedio* obtenida a partir de la palabra pronunciada y los patrones de referencia [Rabiner 93]. La figura 8.3 muestra el diagrama de bloques empleada para realizar el reconocedor con VQ.

Cuando se obtienen todos los vectores MFCC de una palabra, estos son comparados con los centroides de las palabras de entrenamiento. Se obtiene la distancia mínima y la distancia próxima del resultado de la comparación, y se aplica un criterio de rechazo para obtener los mensajes correspondientes.

El procedimiento para el reconocimiento con VQ es detallado a continuación:

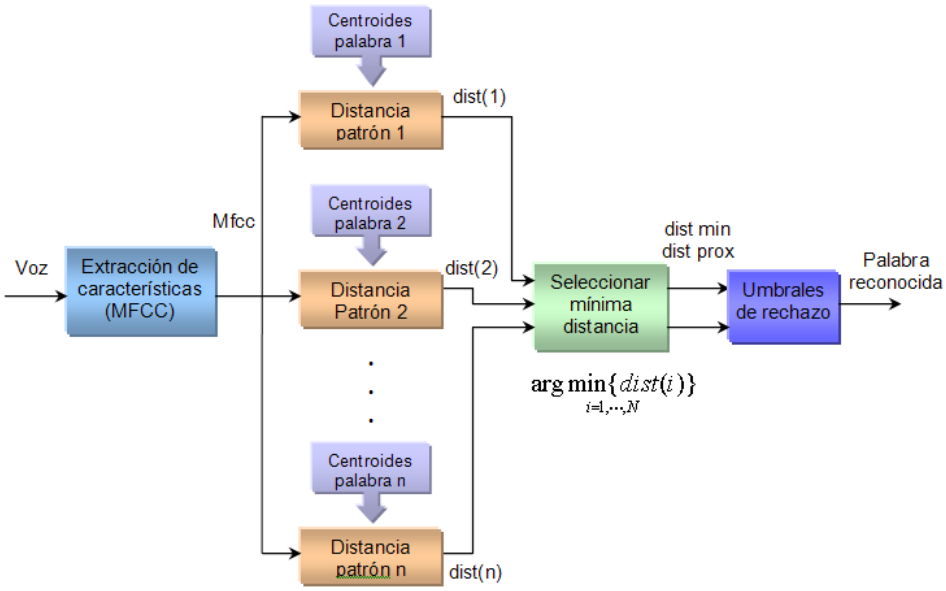


Figura 8.3: Diagrama de bloques para el reconocimiento con VQ.

1. Primeramente, se normalizan los vectores MFCC como lo indica las siguientes ecuaciones:

$$media[i] = \frac{1}{T}\sum_{t=1}^{T} MelCepst_t[i] \qquad i = 0, 1, \cdots, D-1 \quad (8.1)$$

$$MFCC_t[i] = MelCepst_t[i] - media[i] \qquad i = 0, 1, \cdots, D-1$$
$$t = 1, 2, \cdots, T \quad (8.2)$$

donde:

D: Es el número de coeficientes Mel cepstral, (13 coeficientes)

T: Número de tramas.

$MelCepst_t$: Vector Mel cepstral no normalizado en el instante t.

$MFCC_t$: Vector Mel cepstral normalizados en el instante t.

2. Se divide los vectores MFCC de la palabra pronunciada, en segmentos lineales en el tiempo. La figura 8.4 muestra el proceso de segmentación.

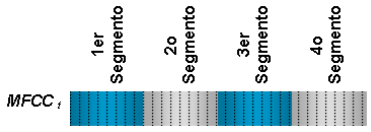


Figura 8.4: División de los vectores MFCC en segmentos lineales en el tiempo.

3. Se obtiene la distancia de los vectores, que se encuentran en cada segmento de la palabra pronunciada, con respecto a los centroides correspondientes al mismo segmento, para cada una de las palabras de reconocimiento de la siguiente forma.

   Sea la distancia euclidiana cuadrática dado por la ecuación 8.3:

$$distEucl(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{D} \big(x_i - y_i\big)^2 \tag{8.3}$$

   donde:

   $\mathbf{X}, \mathbf{Y}$: Son dos vectores con dimensión D

   x_i, y_i: Elementos i-ésimos de los vectores $\mathbf{X}, \mathbf{Y}$ respectivamente.

   Sea Q_s el conjunto de vectores MFCC que pertenecen al segmento s, entonces, para cada instante $t = 1, \cdots, T$, y para $MFCC_t \in Q_s$, se obtiene:

$$dist_p(t) = \min_{k=1,\cdots,K} \Big\{ distEucl\big(Centr_p(s,k),\ MFCC_t\big) \Big\} \tag{8.4}$$

   donde:

   K: Es el número de centroides o *clusters*, en este caso se utilizaron 16 centroides

   S: Número de segmentos. Se emplearon 4 segmentos por palabra.

   $MFCC_t$: El vector Mel cepstral en el instante t.

   $Centr_p(s,k)$: Centroide k de la palabra p, en el segmento s.

4. Se obtiene la distancia promedio para todas las palabras $p = 1, \cdots, P$, donde P es el número de palabras a reconocer (en este caso $P = 8$).

$$dist(p) = \frac{1}{T}\sum_{t=1}^{T} dist_p(t) \tag{8.5}$$

5. Se obtiene la distancia mínima así como su índice $p*$ correspondiente, de igual forma, se obtiene la distancia más próxima a la mínima y su respectivo índice.

$$\begin{aligned} dist_{min} &= \min_{p=1,\cdots,P}\{dist(p)\} \quad p* = \arg\min_{p=1,\cdots,P}\{dist(p)\} \\ dist_{prox} &= \min_{\substack{k=1,\cdots,P \\ k\neq p*}}\{dist(k)\} \quad k* = \arg\min_{\substack{k=1,\cdots,P \\ k\neq p*}}\{dist(k)\} \end{aligned}$$

6. El siguiente punto es la emisión de los mensajes. Esto se realiza utilizando los umbrales de rechazo: *FrontMax* (frontera máxima), *UmbRiesgo* (umbral de riesgo), *FrontSup(p)* (frontera superior en la palabra p) y *promedio(p)* (promedio de la distancia en la palabra p). En el apéndice B se describe la forma de obtener los umbrales de rechazo.

   El rechazo de las palabras se realiza con el siguiente procedimiento:

   - Si $(dist_{min} \geq FrontMax)$ entonces se emite el mensaje "*Palabra desconocida*", en este caso se anuncia que la palabra pronunciada, no es parte del conjunto de palabras de reconocimiento.
   - Por otro lado, si $\big((dist_{min} + UmbRiesgo) < dist_{prox}\big)$ y $\big(dist_{min} < FrontSup(p*)\big)$, se emite el mensaje con la palabra reconocida, por ejemplo: "*Reconoció: Izquierda*".
   - Si no cumple con las dos condiciones previas, se compara la distancia mínima y la distancia próxima, con respecto a sus niveles promedio; se utilizan esos valores para determinar si es una palabra reconocida o una palabra que es confusa.

$$\begin{aligned} offSetDist_{min} &= |promedio(p*) - dist_{min})| && (8.6) \\ offSetDist_{prox} &= |promedio(k*) - dist_{prox})| && (8.7) \\ & && (8.8) \end{aligned}$$

   - Si $\big(offSetDist_{min} < offSetDist_{pox}\big)$, emite el mensaje con la palabra reconocida con dificultad, por ejemplo: "*Reconoció: Derecha ***"
   - Para otro caso, se emite el mensaje ("*Repetir palabra*"), en donde no se sabe si la palabra, con la mínima distorsión, es la palabra dicha o una palabra incorrecta. Debido a la cercanía de las distancias, es preferible que el usuario repita la palabra, en lugar de tener una incertidumbre del resultado.

7. Finalmente, el sistema activa las interrupción por hardware para recibir nuevamente muestras de una nueva señal de voz. Con esto, el sistema cambia al estado *Leer trama*, iniciando nuevamente el proceso de captura de la palabra.

8.4. Reconocimiento usando HMM.

El reconocimiento de palabras aisladas usando HMM, emplea la probabilidad del modelo acústico $P(\mathbf{X}|\mathbf{W})$, para determinar la palabra reconocida. Utiliza para ello, el procedimiento *hacia adelante* (*forward*), descrito en la sección 4.2.4 (pág. 61). La búsqueda consiste en un problema simple de reconocimiento de patrones, en donde la palabra $\hat{\mathbf{W}}$ con más alta probabilidad *forward*, es la palabra reconocida [Huang, Acero y Hon 01]. La figura 8.5 muestra el diagrama de bloques en el reconocimiento de palabras aisladas con HMM.

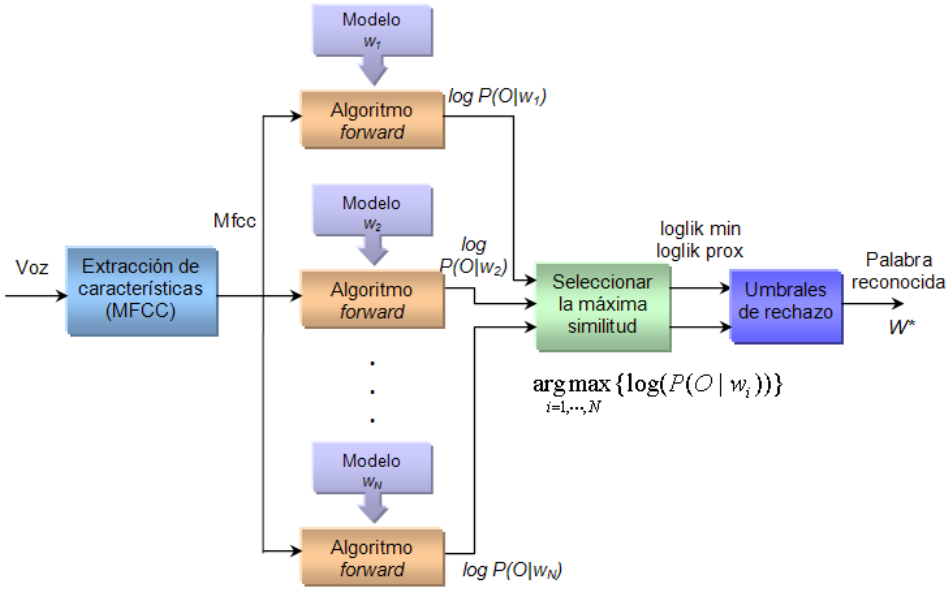


Figura 8.5: Diagrama de bloques para el reconocimiento con HMM.

El procedimiento se detalla a continuación:

1. Se normalizan los vectores MFCC, de la siguiente forma:

$$media[i] = \frac{1}{T}\sum_{t=1}^{T} MelCepst_t[i] \qquad i = 0, 1, \cdots, D-1 \quad (8.9)$$

$$MFCC_t[i] = MelCepst_t[i] - media[i] \qquad i = 0, 1, \cdots, D-1 \quad t = 1, 2, \cdots, T \quad (8.10)$$

donde:

D: Es el número de coeficientes mel cepstral, (13 coeficientes)

T: Número de tramas.

$MelCepst_t$: Vector Mel cepstral no normalizado en el instante t.

$MFCC_t$: Vector Mel cepstral normalizados en el instante t.

2. Para cada palabra p, en el conjunto de entrenamiento, se obtiene la $LogLik_p = \log\left(P(\mathbf{X}|\mathbf{W}_p)\right)$ correspondiente. Esta probabilidad logarítmica es producida con la variable *hacia adelante* $\alpha_t(i)$, por medio del algoritmo *forward* descrito en la sección 4.2.4. Para generar las probabilidades de observación, el procedimiento utiliza las densidades gaussianas definidas por la ecuación 7.8 (pág. 137).

   La forma de obtener $LogLik_p$, dada la observación y al modelo HMM de la palabra p, se realiza lo siguiente:

   - Para cada instante t, se obtiene $\alpha_t(i)$, como se indica en el algoritmo *forward* (sección 4.2.4).
   - Se calcula el factor de escalamiento c_t de la siguiente forma:

$$c_t = \frac{1}{\max\limits_{i=1,\cdots,N} \alpha_t(i)} \quad \text{para } t = 1, \cdots, T \quad (8.11)$$

     donde:

     T : es el número de vectores MFCC de observación.

   - Utilizando el factor de escalamiento, se calcula $\hat{\alpha}_t(i)$

$$\hat{\alpha}_t(i) = c_t\alpha_t(i) \quad \text{para } t = 1, \cdots, T \quad \text{e} \quad i = 1, \cdots, N \quad (8.12)$$

- Cuando se han procesado todas las observaciones T, se calcula la probabilidad logarítmica *forward* para cada palabra p, de la siguiente manera:

$$logLik_p = \log\left(\hat{\alpha}_T(N)\right) - \sum_{t=1}^{T} \log\left(c_t\right) \tag{8.13}$$

3. El siguiente paso es obtener el valor máximo de 8.13, denotado por $logLik_{max}$; también, se extrae el valor más cercano a $logLik_{max}$, llamado $logLik_{prox}$. Las siguientes expresiones describen la forma de obtener estos dos valores:

$$\begin{aligned} logLik_{max} &= \underset{p=1,\cdots,P}{\text{máx}}\{logLik_p\} \quad p* = \arg\underset{p=1,\cdots,P}{\text{máx}}\{logLik_p\} \\ logLik_{prox} &= \underset{\substack{k=1,\cdots,P \\ k\neq p*}}{\text{máx}}\{logLik_k\} \quad k* = \arg\underset{\substack{k=1,\cdots,P \\ k\neq p*}}{\text{máx}}\{logLik_k\} \end{aligned}$$

4. A continuación se realiza la emisión de los mensajes. Se utiliza los umbrales de rechazo: *FrontMin* (frontera mínima), *UmbRiesgo* (umbral de riesgo), *FrontInf(p)* (frontera inferior de la palabra p) y *promedio(p)* (promedio de la distancia en la palabra p). La forma de obtener los umbrales de rechazo para HMM, se describe en el apéndice C.

   En seguida se muestran el criterio establecido para la emisión de mensajes:

   - Si $(logLik_{max} \leq FrontMin)$ entonces se emite el mensaje "*Palabra desconocida*", que significa que la palabra pronunciada no es parte del conjunto de palabras de reconocimiento.
   - Por otro lado si $\left((logLik_{max} - UmbRiesgo) > logLik_{prox}\right)$ y $\left(logLik_{max} > FrontInf(p*)\right)$, entonces emite el mensaje con la palabra reconocida, por ejemplo: "*Reconoció: Izquierda*".
   - De lo contrario, se compara la probabilidad máxima y la probabilidad próxima con respecto a sus niveles promedio, para determinar si es reconocida con confusión ó si es necesario repetir la palabra. Se obtienen los siguientes valores,

$$\begin{aligned} offSetLogLik_{max} &= |promedio(p*) - logLik_{max})| \quad (8.14) \\ offSetLogLik_{prox} &= |promedio(k*) - logLik_{prox})| \quad (8.15) \end{aligned}$$

     - Si $\left(offSetLogLik_{max} < offSetLogLik_{pox}\right)$, emite el mensaje con la palabra reconocida con confusión, por ejemplo: "*Reconoció: Derecha* **"
     - De lo contrario, se emite el mensaje ("*Repetir palabra*"), En este caso, la palabra pronunciada se aproxima demasiado a dos patrones, por lo que puede haber error en el reconocimiento y se recomienda repetir nuevamente la palabra.

5. Finalmente, el sistema activa las interrupción por hardware para recibir nuevamente muestras de la señal de voz. Con esto, el sistema cambia al estado *Leer trama* comenzando nuevamente el proceso de reconocimiento.

Capítulo 9

Resultados y conclusiones

El sistema de reconocimiento de palabras aisladas en tiempo real, mediante el DSP C6711, se diseñó para identificar ocho palabras diferentes de un mismo locutor, con el uso de vectores Mel Cepstral y dos técnicas de entrenamiento: VQ y HMM.

En la fase de entrenamiento se utilizaron 20 repeticiones de cada palabra. El proceso de extracción de características empleó vectores Mel Frequency Cepstral (MFCC) con 13 elementos.

En la primera técnica de entrenamiento, VQ, se dividieron las repeticiones de cada palabra en 4 segmentos lineales en el tiempo. Cada segmento fue agrupado en 16 centroides por medio del algoritmo *K-Medias* y la distancia euclidiana cuadrática.

Por otro lado, el entrenamiento con HMM utilizó la palabra, como unidad de reconocimiento. Los modelos acústicos fueron creados, por medio de la concatenación de modelos HMM independientes: que representan a los fonemas transcritos de la palabra. Los modelos HMM, de cada fonema, se representaron con tres estados y dos transiciones: una transición al mismo estado y otra al estado siguiente. Se utilizó una densidad gaussiana por cada estado, para interpretar las probabilidades de observación. El entrenamiento, de los modelos HMM, se realizó por medio del algoritmo *Baum-Welch* con una razón de convergencia de 0.02.

9.1. Porcentajes de reconocimiento

El sistema se probó con 50 pronunciaciones de cada una de las palabras de entrenamiento, con un entorno de ruido semicontrolado[1]. Se realizaron tres tipos de pruebas: una empleando VQ y la distancia euclidiana; la segunda utiliza VQ y la distancia euclidiana cuadrática; finalmente, la tercera emplea HMM y probabilidad logarítmica.

La tabla 9.1 muestran los porcentajes de reconocimiento obtenidos usando VQ y la distancia euclidiana: el 99.5 % de las palabras, fueron reconocidas sin confusión; el 0.5 % de las palabras, se reconocieron con dificultad; el 0.25 % se clasificaron como palabras desconocidas; finalmente, no hubo la necesidad de repetir la palabra, tampoco reconoció palabras erróneas.

Tabla 9.1: Resultados en el entrenamiento con VQ y distancia euclidiana

Palabra	**Reconocida** (bien)	**Reconocida** (dificultad)	**Repetir palabra**	**Error al reconocer**	**Palabra desconocida**
Alto	49	1	0	0	0
Sigue	49	0	0	0	1
Izquierda	50	0	0	0	0
Derecha	50	0	0	0	0
Arriba	50	0	0	0	0
Abajo	50	0	0	0	0
Adelante	50	0	0	0	0
Atrás	49	1	0	0	0
Totales	**397**	**2**	**0**	**0**	**1**
Porcentajes	**99.25 %**	**0.5 %**	**0 %**	**0 %**	**0.25 %**

El segundo experimento se realizó empleando VQ y la distancia euclidiana cuadrática. Los resultados son muy parecidos al experimento previo. Se obtuvo el 99 % de reconocimiento sin dificultad, el 1 % con un poco de confusión; no se obtuvieron palabras desconocidas, palabras reconocidas con error, tampoco hubo la necesidad de repetir la palabra. La tabla 9.2 muestra los resultados obtenidos para esta prueba.

El tercer experimento fue realizado con HMM y la probabilidad logarítmica. La tabla 9.3 muestra los resultados obtenidos con esta técnica. Como se observa, se obtuvo el 99.75 % de reconocimiento sin confusión y el 0.25 % se reconoció con un poco de dificultad; no hubo algún evento para las demás opciones.

[1] Es semicontrolado por que no hubo ruidos de gran potencia, solo un pequeño sonido constante producido por un ventilador.

Tabla 9.2: Resultados obtenidos usando VQ y distancia euclidiana cuadrática.

Palabra	**Reconocida** (bien)	**Reconocida** (dificultad)	**Repetir palabra**	**Error al reconocer**	**Palabra desconocida**
Alto	50	0	0	0	0
Sigue	49	1	0	0	0
Izquierda	48	2	0	0	0
Derecha	49	1	0	0	0
Arriba	50	0	0	0	0
Abajo	50	0	0	0	0
Adelante	50	0	0	0	0
Atrás	50	0	0	0	0
Totales	**396**	**4**	**0**	**0**	**0**
Porcentajes	**99 %**	**1 %**	**0 %**	**0 %**	**0 %**

Tabla 9.3: Resultados logrados con HMM

Palabra	**Reconocida** (bien)	**Reconocida** (dificultad)	**Repetir palabra**	**Error al reconocer**	**Palabra desconocida**
Alto	50	0	0	0	0
Sigue	50	0	0	0	0
Izquierda	50	0	0	0	0
Derecha	50	0	0	0	0
Arriba	50	0	0	0	0
Abajo	50	0	0	0	0
Adelante	49	1	0	0	0
Atrás	50	0	0	0	0
Total	399	1	0	0	0
Porcentajes	99.75 %	0.25 %	0 %	0 %	0 %

9.2. Tiempos de ejecución

Una característica importante en los sistemas de reconocimiento, es determinar el tiempo que se tardan en efectuar el reconocimiento. Para este trabajo, se considera el tiempo de reconocimiento como: el intervalo transcurrido desde el término de la pronunciación de una palabra, hasta la obtención de los resultados. Por lo cual, no contempla el lapso transcurrido entre la pronunciación de la palabra, ni la emisión del mensaje de resultado, debido a que este último, está en función de la tarea desempeñada por el comando.

Para el procesamiento de las tramas en tiempo real, el sistema invoca varias funciones específicas como son: aplicar una ventana, cálculo de la *FFT*, etc., por tal motivo, el tiempo que tardan cada una de estas funciones, es de suma importancia, si se desea reconocer palabras en tiempo real. La tabla 9.4 muestra la duración aproximada (para cada trama) de las funciones más importantes, que van, desde la captura de la señal, hasta el cálculo de los vectores MFCC; contiene las siguientes columnas: *Nombre de la función*, *Ciclos de reloj promedio* y *Tiempo aproximado*. El promedio de los ciclos de reloj fue obtenido a través del *Profiler*, en la herramienta de desarrollo *Code Composer Studio*. Se invocó 50 veces cada función, para obtener el promedio de los ciclos de reloj; el tiempo aproximado fue calculado, contemplando la frecuencia de trabajo del DSP, de 150 MHz.

Tabla 9.4: Duración aproximada de los procesos más importantes.

Nombre de la función	**Ciclos de reloj promedio** (por trama)	**Tiempo aprox.** (por trama)
Aplicar ventana hamming	15449	0.1029933 ms.
Cálculo de la *FFT*	228851	1.5256733 ms.
Ordenación en *bit-reverse*	31591	0.2106066 ms.
Obtener los vectores MFCC	226162	1.5077466 ms.
Energía y cruces por cero	12583	0.0838866 ms.
Totales	**514636**	**3.430907 ms.**

Como la frecuencia de muestreo del convertidor A/D, es de 8 KHz, y el tamaño de una trama es de 205 muestras con corrimientos de 80 muestras, entonces, el tiempo para procesar una trama debe ser menor a 80 [muestras] / 8000 [muestras/segundo] = 10 [ms], comparando este resultado con el obtenido en la tabla 9.4 de 3.430907 [ms], entonces el sistema tiene la capacidad suficiente para realizar el procesamiento en tiempo real, incluso de realizar tareas adicionales. Por otro lado, no hay que dejar de considerar que el tiempo obtenido es una aproximación y sólo contempla las funciones principales, no se toma en cuenta el tiempo de llamadas a las interrupciones, tampoco la gestión que hace

el DSP/BIOS para su operación, aunque sólo realice muy pocas tareas, a fin de cuenta, es tiempo extra que se debe agregar.

Considerando que los valores para procesar una trama es de 3.430907 [ms], se puede obtener el tiempo que tarda el DSP en calcular los coeficientes MFCC, para una palabra que dura un segundo:

Tiempo MFCC_{1seg} = (3.430907)[ms/tramas] x (98.43)[tramas]

Tiempo MFCC_{1seg} = 0.3377299 [s]

Por consiguiente, el DSP tarda aproximadamente 0.3377299 segundos en ejecutar los cálculos necesarios para obtener los coeficientes MFCC de una palabra que dura un segundo.

Por otra parte, en el proceso de comparación de patrones se detiene la captura de datos, dedicándose exclusivamente el DSP a realizar el reconocimiento.

Las tablas 9.5 y 9.6 muestran los ciclos de reloj aproximados, por trama, para el reconocimiento con VQ y las distancias: euclidiana y euclidiana cuadrática, respectivamente. Los valores fueron obtenidos a partir de una sola pronunciación de las palabras. Las tablas muestran el número de ciclos de reloj aproximados, que tarda el DSP en realizar la comparación de cada comando en particular, el número de tramas que existió en la pronunciación de una palabra y el número de ciclos de reloj que se ejecutan en cada trama (este dato se obtuvo dividiendo el número total de ciclos, entre la cantidad de tramas en la palabra). De igual forma, la tabla 9.7 muestra los ciclos de reloj aproximados para el reconocimiento de las palabras, en el caso de la técnica HMM. Es importante aclarar que los ciclos de reloj fueron obtenidos, empleando el reloj del *Profiler* en la herramienta de desarrollo *Code Composer Studio* y que puede variar, dependiendo de la limpieza y/o conflictos del *pipeline*, entre otras cosas.

Tabla 9.5: Ciclos de reloj para el reconocimiento con VQ y la distancia euclidiana.

Palabra	**Ciclos**	**Num. Tramas**	**Ciclos/Trama**
Alto	42851811	51	840231.59
Sigue	51280179	61	840658.67
Izquierda	71431329	85	840368.58
Derecha	57943465	69	839760.36
Arriba	60475138	72	839932.47
Abajo	52067127	62	839792.37
Adelante	73105632	87	840294.62
Atrás	57105803	68	839791.22
Totales	**466260484**	**555**	**840108.98**

Tabla 9.6: Ciclos de reloj en el reconocimiento con VQ y la distancia euclidiana cuadrática.

Palabra	**Ciclos**	**Num. Tramas**	**Ciclos/Trama**
Alto	5984763	51	104995.84
Sigue	7344399	61	104919.99
Izquierda	8599465	82	104871.52
Derecha	6611604	69	104946.10
Arriba	7763295	74	104909.39
Abajo	7029679	67	104920.58
Adelante	9331236	89	104845.35
Atrás	7762284	74	104895.73
Totales	**60426725**	**576**	**104907.51**

Tabla 9.7: Ciclos de reloj para el reconocimiento con HMM y la probabilidad logarítmica.

Palabra	**Ciclos**	**Num. Tramas**	**Ciclos/Trama**
Alto	47725845	53	900487.64
Sigue	56309534	61	923107.11
Izquierda	83944887	87	964883.76
Derecha	64863098	69	940044.90
Arriba	64600752	69	936242.78
Abajo	61464008	66	931272.85
Adelante	84864869	88	964373.51
Atrás	66745346	71	940075.30
Totales	**530518339**	**564**	**940635.35**

Con estos resultados, se puede obtener una aproximación del tiempo que tardar el DSP, en reconocer una palabra con duración de un segundo. Se considera los siguientes datos para realizar los cálculos:

- La frecuencia de trabajo del DSP es de 150 MHz.
- El *codec* trabaja a una frecuencia de muestreo de 8 KHz.
- Se emplea un corrimiento de ventana de 80 muestras (0.01 s).
- Cada trama contiene 0.025625 segundos de la señal (205 muestras por trama).
- Una palabra, de aproximadamente un segundo, tiene 98 tramas (1 + (8000 - 205)/80).

Por tanto, para reconocer una palabra de un segundo, con VQ y distancia euclidiana, el DSP tarda:

$$\begin{aligned} \text{Tiempo VQ y dist. Euc.} &= \frac{98\,[\text{tramas}]\ \text{x}\ 840108{,}98\,[\text{ciclos/trama}]}{150\ \text{MHz}} \\ \text{Tiempo VQ y dist. Euc.} &= 0{,}5488712\,s. \end{aligned}$$

Si se cambia la medida de distorsión por la distancia euclidiana cuadrática, se obtiene:

$$\begin{aligned} \text{Tiempo VQ y dist. Euc. Cuad.} &= \frac{98\,[\text{tramas}]\ \text{x}\ 104907{,}51\,[\text{ciclos/trama}]}{150\ \text{MHz}} \\ \text{Tiempo VQ y dist. Euc. Cuad.} &= 0{,}0685395\,s. \end{aligned}$$

Como se puede comprobar, el tiempo de reconocimiento para el caso de la distancia euclidiana cuadrática, es aproximadamente ocho veces más rápida, que el tiempo empleando en la distancia euclidiana.

Por otro lado, en la técnica con Modelos Ocultos de Markov, se tiene un tiempo de reconocimiento de:

$$\begin{aligned} \text{Tiempo HMM.} &= \frac{98\,[\text{tramas}]\ \text{x}\ 940635{,}35\,[\text{ciclos / trama}]}{150\ \text{MHz}} \\ \text{Tiempo HMM.} &= 0{,}6145484\,s. \end{aligned}$$

Por los que el DSP tarda aproximadamente 0.6145484 segundos, adicionales a la captura de la señal, en realizar la comparación de una palabra que dura un segundo, empleando HMM. La tabla 9.8 resume los tiempos que tarda el DSP, en reconocer una palabra, de un segundo, para cada técnica de reconocimiento.

Tabla 9.8: Resumen de tiempos que tarda el DSP, en reconocer una palabra de un segundo.

Técnica de reconocimiento	**Duración**
VQ y dist. euclidiana	0.5488712 s.
VQ y dist. euclidiana cuadrática	0.0685395 s.
HMM y prob. logarítmica	0.6145484 s.

9.3. Memoria de datos utilizada

La memoria utilizada por el sistema se puede dividir en tres partes: *memoria de buffer*, en donde se almacenan las muestras temporales de la señal de voz; *memoria Mel cepstral*, empleada para el cálculo de los vectores MFCC; y la *memoria de los patrones*, que es básicamente la cantidad de espacio utilizada para almacenar los patrones de comparación.

9.3.1. Memoria de buffer y memoria Mel cepstral

La memoria buffer es donde se van almacenando temporalmente las muestras de la señal de voz, conforme se capturan del convertidor A/D. El buffer tiene un tamaño para almacenar 12,000 muestras de voz de 16 bits, por lo que la memoria utilizada para este fin es de 24,000 bytes. Cuando la palabra pronunciada tiene una duración mayor de 1.5 s., los datos restante son almacenados al inicio del buffer, por lo que se considera como un buffer circular controlado por software. Es importante aclarar que el sistema está diseñado para que trabaje con un buffer más pequeño, no menor a 1,184 bytes (592 muestras), con lo que se reduciría considerablemente esta memoria.

Otro espacio de memoria utilizado en la memoria buffer, es donde se guardan los vectores Mel Cepstral de la palabra pronunciada. Se ha reservado memoria para almacenar 200 vectores MFCC, esto es alrededor de 2 segundos de voz, considerando un corrimiento de ventana de 80 muestras (o un traslape de 125 muestras) y cada ventana almacena 205 muestras. Por tanto, la memoria utilizada para este fin es de 200 x 13 x 4 = 10,400 bytes.

En el cálculo de los vectores MFCC también se ocupó memoria para llevar

a cabo esta tarea. Primeramente se emplearon 102 valores de 4 bytes para almacenar la ventana *Hamming*, es decir 102 x 4 = 408 bytes. Para el cálculo de la *FFT* de una trama se empleó una variable compleja de doble precisión para almacenar 256 valores, esto es 256 x 2 x 8 = 4,096 bytes; una variable de doble precisión para guardar el espectro de potencia, 256 x 8 = 2,048 bytes; se almacenaron los coeficientes complejos para el cálculo de la *FFT*, 128 x 2 x 8 = 2,048 bytes. En la aplicación del banco de filtros con escala mel, se utilizaron 31 filtros y cada uno de ellos contiene un máximo de 11 valores, por lo que se reservó memoria para guardar 31 x 11 x 4 = 1,364 bytes para los coeficientes; 31 x 4 = 124 bytes y 62 bytes para determinar los límites de cada filtro. Finalmente, los coeficientes para calcular la transformada discreta coseno, ocuparon 31 x 13 x 4 = 1,612 bytes.

La tabla 9.9 resume la cantidad de memoria empleada para todo el proceso de obtención de los vectores Mel Cepstral.

Tabla 9.9: Memoria utilizada para el buffer temporal y para el cálculo de los vectores MFCC.

	Tipo de almacenamiento	**Memoria (bytes)**
Memoria buffer	Almacenar señal de voz	24,000
	Buffer de vectores MFCC	10,400
Memoria Mel Cepstral	Ventana hamming	408
	Espectro de la señal	4,096
	Espectro de potencia la trama	2,048
	Coef. complejos	2,048
	Coef. banco de filtros	1,364
	Límites del banco de filtros	186
	Coef. Transf Coseno	1,612
	Total	46,162

9.3.2. Memoria de los patrones de comparación

La mayoría de los patrones de comparación fueron almacenados con números flotantes de 32 bits. En ambas técnicas de entrenamiento, se utilizaron vectores Mel Cepstral de dimensión 13.

En el caso del entrenamiento con VQ, se utilizaron 16 centroides por segmento, 4 segmentos por palabra y se entrenaron 8 comandos; por lo que, la memoria utilizada para almacenar los patrones de referencias se desglosa de la siguiente manera:

$$\begin{aligned}\text{Mem. patrones} = {} & 4\{\text{bytes por elemento}\} \text{ x } 13\{\text{dimensión MFCC}\} \text{ x} \\ & 16\{\text{centroides}\} \text{ x } 4\{\text{segmentos}\} \text{ x } 8\{\text{palabras}\} \\ \text{Mem. patrones} = {} & 26,624 \text{ bytes}\end{aligned}$$

Para el caso del entrenamiento con HMM, el cálculo de la memoria se obtuvo contemplando varios factores de realización del algoritmo. Debido a que se ha dado la prioridad para que la ejecución sea más rápida, la memoria no fue optimizada. Por ejemplo, para obtener la probabilidad de observación, se almacenó el factor de normalización, evitando así realizar cálculos innecesarios. También, debido a que las palabras de entrenamiento tienen diferentes fonemas y por tanto, diferente cantidad de estados, se emplearon variables extras para tener un mayor manejo de la información.

En cada palabra de entrenamiento se creó un modelo acústico, con el número de estados dependiente de la cantidad de sus fonemas. Se emplearon 3 estados para representar cada fonema, por lo que el número total de estado utilizados fue 135. La tabla 9.10 muestra la cantidad de estados utilizados para cada palabra de entrenamiento y la tabla 9.11 muestra la cantidad de memoria utilizada para almacenar los patrones de comparación para HMM.

Tabla 9.10: Número de estados por palabra

Palabra	**Núm. Estados**
Alto	4 x 3 = 12
Sigue	4 x 3 = 12
Izquierda	8 x 3 = 24
Derecha	6 x 3 = 18
Arriba	5 x 3 = 15
Abajo	5 x 3 = 15
Adelante	8 x 3 = 24
Atrás	5 x 3 = 15
Total	**135**

Es necesario aclarar que no se considera la memoria utilizada por el *stack* (memoria que se usa para manejo de variables locales, llamadas a funciones, etc.). Tampoco se toma en cuenta la memoria ocupada por el código de los programas. Por tanto, la cantidad de memoria total utilizada para el sistema de reconocimiento es: 72,786 bytes para la técnica con Cuantización Vectorial (VQ) y 64,534 bytes para Modelos Ocultos de Markov.

Tabla 9.11: Memoria utilizada para almacenar los patrones con HMM

Variable	**Memoria (bytes)**
Media	135 x 13 x 4 = 7020
Varianza	135 x 13 x 4 = 7020
Apuntadores de la media	800
Apuntadores de la varianza	800
Norm Gauss.	135 x 4 = 540
Apunt. de la Norm Gauss.	8 x 4 = 32
Prob de transición (A)	135 x 2 x 4 = 1080
Índices para A	135 x 2 x 4 = 1080
Total	**18,372**

9.4. Mejoras

El sistema de reconocimiento de comandos de voz, puede ser perfeccionado en varios aspectos, como por ejemplo: se puede modificar el sistema para que sea más rápido, para que funcione en entornos ruidosos, con mayor vocabulario, etc. Los siguientes puntos describen algunas mejoras que se pueden implementar para perfeccionar el sistema:

- Para mejorar la velocidad de ejecución, se pueden elaborar algunos algoritmos en lenguaje ensamblador del DSP. Sin embargo, con esto también aumenta la complejidad del sistema.

- Para que el ruido de fondo no interfiera demasiado con el reconocimiento, se puede realizar un filtro que elimine el ruido ambiental, con el uso de otro micrófono. Esto ayudaría enormemente al reconocimiento en lugares no controlados. Sin embargo, la incorporación de un micrófono extra, conduce a la integración de una nueva tarjeta llamada *daughter card*, y por tanto, una mayor inversión.

- Los umbrales de ruido pueden ser calculados de manera dinámica, utilizando los lapsos de tiempo en que el DSP permanece ocioso. Es decir, se puede activar un procedimiento capaz de detectar esta situación y efectuar un nuevo cálculo de umbrales del ruido. La realización de esto no es muy complicado, con la ayuda de la herramienta de desarrollo *DSP/BIOS* a través de un hilo tipo *idle*.

- El sistema de reconocimiento está realizado para que sea dependiente del locutor. Sin embargo, se puede modificar para que sea independiente del locutor, principalmente en la técnica HMM, usando más palabras de entrenamiento con distintos locutores.

- Otra posible mejora es la integración de adaptación del locutor, es decir, que el sistema se adapte a la persona que dicta los comandos de voz.

- Además, se puede enriquecer el diccionario de palabras, siempre teniendo como limitante la capacidad de memoria del DSP. Aumentar las palabras de reconocimiento se incrementa el número de comparaciones necesarias y por tanto, disminuye la velocidad de respuesta.

- Se empleó como unidad de reconocimiento en el modelado acústico, la palabra. Si se desea incorporar una cantidad mucho más grande de comandos, se puede emplear otra unidad de reconocimiento como es el difonema o trifonema, lo que llevaría a necesitar una enorme cantidad de palabras de entrenamiento, también conocido como un corpus más grande.

- Una característica del sistema, es que no incluye el modelado del lenguaje. Para el reconocimiento de palabras continuas en tiempo real, el modelo del lenguaje es de suma importancia, junto con el cambio de las unidades de reconocimiento, por ejemplo: difonemas o trifonemas; sin embargo, la cantidad de memoria que tiene la tarjeta, no es suficiente para este tipo de reconocimiento, por lo que es necesario agregarle más memoria.

9.5. Posibles aplicaciones

El sistema de reconocimiento de comando de voz tiene muchas aplicaciones, tantas como la imaginación lo permita. Por ejemplo, se puede utilizar el sistema para controlar una silla de ruedas de una persona minusválida; así como, para controlar el nivel de inclinación de las camas de hospital.

Otra aplicación pueden ser dentro de un automóvil, por ejemplo: el conductor de un vehículo puede comunicarse con el tablero de controles del vehículo, para solicitar el nivel de gasolina, encender las luces o abrir una ventana.

La telefonía móvil ha tenido un crecimiento en los últimos años y cada vez se están haciendo más pequeños los dispositivos, lo que llevaría a la integración de interfaces de operación diferentes, por ejemplo, la incorporación de un sistema de reconocimiento de voz, para su control.

Existen muchas otras aplicaciones como son: en edificios inteligentes, el control de luces en una casa, el control de las máquinas de venta automática, el cobro del transporte público, el control automático del pedido de una pizzería, etc.

9.6. Conclusiones

Los resultados obtenidos, desde un punto de vista particular, fueron buenos, debido a que en versiones previas del sistema [Nieto, López 02], se alcanzó un máximo de 99 % de reconocimiento[2]. En este sistema se logró el 99.75 % de reconocimiento para VQ y la distancia euclidiana, y solo el 0.25 % de las palabras se detectaron como desconocidas. Para el caso de VQ y la distancia euclidiana cuadrática, se obtuvo el 100 % de reconocimiento; todas las palabras de prueba (400), se reconocieron con éxito, de las cuales el 1 % se reconoció con un poco de dificultad. Por otro lado, el reconocimiento con HMM también se obtuvieron muy buenos resultados, el 99.75 % de las palabras se reconocieron bien y solo el 0.25 % con un poco de dificultad, dando el 100 % de reconocimiento.

Como se comprobó, la memoria empleada para guardar los patrones de comparación para HMM es menor que la utilizada para VQ, también la calidad del reconocimiento es mejor para HMM que VQ; sin embargo, el tiempo de reconocimiento para HMM es mayor que VQ con distancia euclidiana, y casi nueve veces más que VQ y distancia media euclidiana. Por lo que si se quiere eficiencia en cuanto al tiempo de reconocimiento, es mejor emplear VQ y una distancia euclidiana cuadrática, sin embargo si se quiere precisión y un poco menos de memoria, HMM es excelente.

Se mejoró el tiempo de reconocimiento, mediante el desarrollo de algoritmos, capaces de efectuar operaciones, conforme se captura la señal de voz. Gracias a lo cual, al término de la pronunciación de la palabra, sólo se ejecutan las comparaciones con los patrones existentes.

Los tipos de discriminación implementados para la comparación de patrones (umbrales de rechazo), a pesar de ser obtenidos empíricamente (apoyados en estadísticas de las diversas palabras), son aceptables. Se puede trabajar más con ellos para mejorarlos en un futuro.

Por otro lado, existe el inconveniente siguiente: para el caso de palabras distintas a las contenidas en el diccionario de palabras, existe el riesgo latente de reconocimiento erróneo. Esto puede ser debido a que, es necesario modificar los umbrales de rechazo, limitando las regiones de reconocimiento a franjas mas pequeñas.

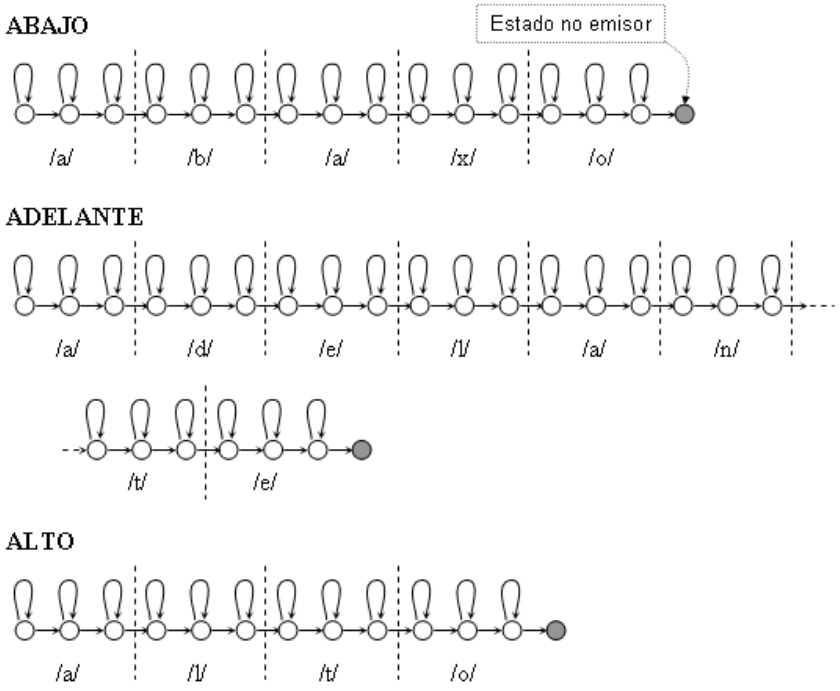
Además, los umbrales de ruido son calculados al inicio de la sesión y permanecen constantes durante todo el tiempo que se efectúa el reconocimiento. Debido a esto, un aumento en los niveles de ruido, puede afectar la precisión en el reconocimiento.

[2]La comparación no es muy correcta debido a que en el sistema previo se entrenó con dos locutores, además de realizar algunas operaciones diferentes en el proceso de extracción de características.

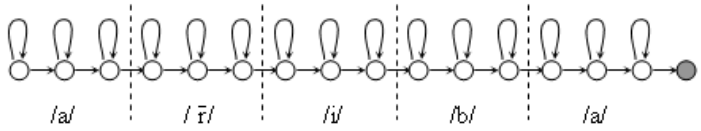
Apéndice A

Modelado HMM de las palabras de entrenamiento

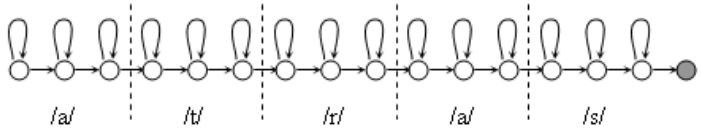
En este apéndice se muestra los modelados HMM empleados en cada una de las palabras de entrenamiento.



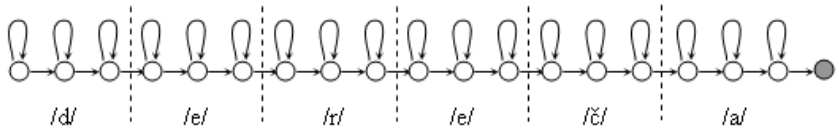
ARRIBA



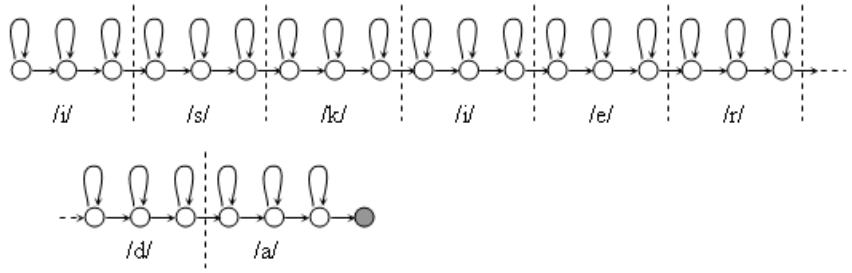
ATRÁS



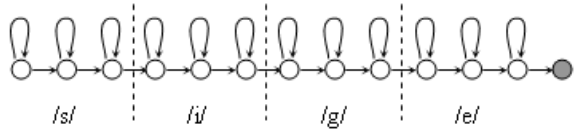
DERECHA



IZQUIERDA



SIGUE



Apéndice B

Calculo de los umbrales de riesgo para VQ

Los umbrales de riesgo empleados en el reconocimiento con VQ, se obtuvieron mediante un conjunto de datos estadísticos, para ello se necesitó almacenar las distancias mínimas ($dist_{min}$) y la distancias próximas ($dist_{prox}$) de veinte repeticiones, en cada una de las palabras en el entrenamiento.

El procedimiento realizado para calcular los umbrales de rechazo se describe a continuación:

- *promedio(p)*: El promedio de la palabra p, se obtiene con la medía aritmética de las distancias mínimas

$$promedio(p) = \frac{1}{M}\sum_{i=1}^{M} dist_{min}(p,i) \quad \text{Para } p = 1, \cdots, P \tag{B.1}$$

  donde:

  M: Es el número de repeticiones usadas, para este caso $M = 20$.

  P: Número de palabras de reconocimiento. Se emplearon $P = 8$ palabras.

- *FrontMax*: El umbral frontera máxima se encarga de delimitar la máxima distancia, para considerarla una palabra dentro del vocabulario. Se calculó con la ayuda de las distancias mínimas de todas las palabras. Las ecuaciones siguientes describen el procedimiento.

$$FrontMax = \mu + 5{,}5 \cdot \sigma \tag{B.2}$$

donde:

$$\mu = \frac{1}{M \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{M} dist_{min}(p,i) \tag{B.3}$$

$$\sigma = \sqrt{\frac{(M \cdot P) \sum_{p=1}^{P} \sum_{i=1}^{M} dist_{min}(p,i)^2 - \left(\sum_{p=1}^{P} \sum_{i=1}^{M} dist_{min}(p,i) \right)^2}{(M \cdot P)(M \cdot P - 1)}} \tag{B.4}$$

- *UmbRiesgo* El umbral de riesgo establece la mínima diferencia entre las distancias mínimas y las distancias próximas, para que se considere una palabra reconocida con exactitud. El umbral se obtiene por medio de las siguientes ecuaciones:

$$UmbRiesgo = \mu - 2{,}2 \cdot \sigma \tag{B.5}$$

donde:

$$\mu = \frac{1}{M \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{M} DifDist(p,i) \tag{B.6}$$

$$DifDist(p,i) = (dist_{prox}(p,i) - dist_{min}(p,i)) \tag{B.7}$$

$$\sigma = \sqrt{\frac{(M \cdot P) \sum_{p=1}^{P} \sum_{i=1}^{M} DifDist(p,i)^2 - \left(\sum_{p=1}^{P} \sum_{i=1}^{M} DifDist(p,i) \right)^2}{(M \cdot P)(M \cdot P - 1)}} \tag{B.8}$$

- *FrontSup(p)* Este umbral determina la frontera superior en cada palabra p, delimita una región de la distancia mínima de cada palabra para que se considera como una palabra dentro del vocabulario. La forma de obtener este umbral se describe en las siguientes ecuaciones:

$$FrontSup(p) = \mu_p + 3 \cdot \sigma_p \tag{B.9}$$

donde:

$$\mu_p = \frac{1}{M}\sum_{i=1}^{M} dist_{min}(p,i) \tag{B.10}$$

$$\sigma_p = \sqrt{\frac{M\sum_{i=1}^{M} dist_{min}(p,i)^2 - \left(\sum_{i=1}^{M} dist_{min}(p,i)\right)^2}{M(M-1)}} \tag{B.11}$$

La figura C.1 muestra la gráfica de los umbrales de rechazo, en comparación con las distancias mínimas y distancias próximas de la palabra "*Abajo*".

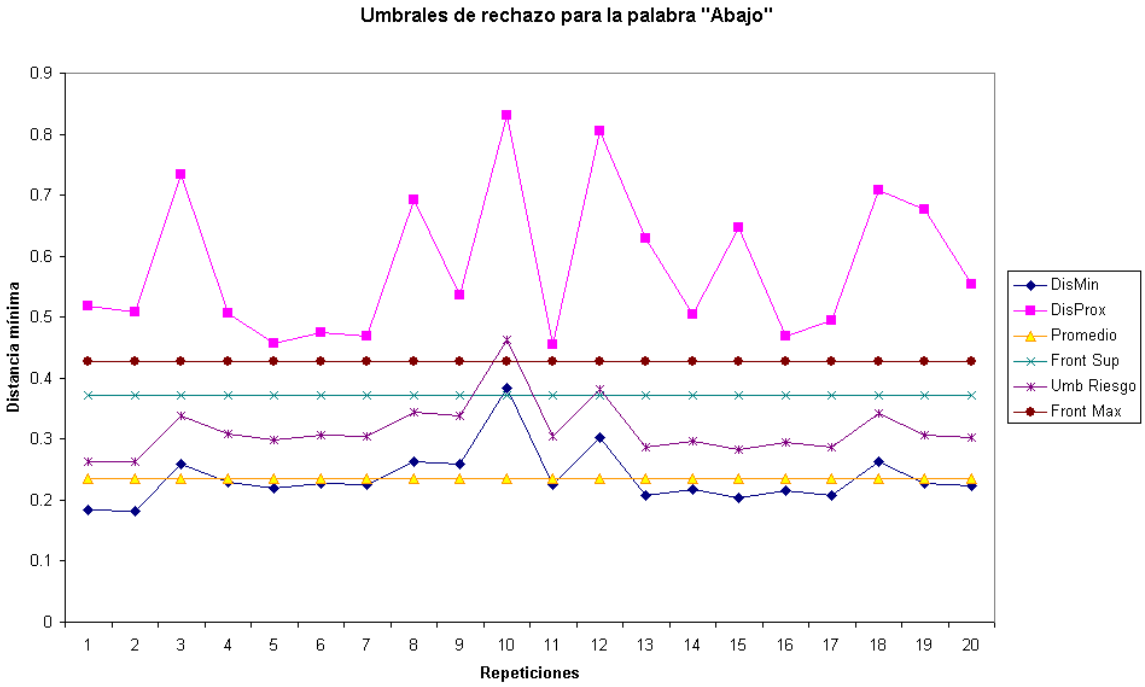


Figura B.1: Comparación de los umbrales de rechazo, con las distancias mínimas y próximas de la palabra "*Abajo*".

Apéndice C

Calculo de los umbrales de riesgo para HMM

Los umbrales de riesgo empleados en el reconocimiento con HMM, se obtuvieron de forma similar a los empleados para VQ. Para ello se emplearon las probabilidades logarítmicas máximas ($logLik_{max}$) y las probabilidades logarítmicas próximas ($logLik_{prox}$) de veinte repeticiones, para cada una de las palabras en el entrenamiento.

En seguida se describe la forma de obtener los umbrales:

- *promedio(p)*: El promedio de la probabilidad logarítmica p, se obtiene con la medía aritmética como se describe a continuación:

$$promedio(p) = \frac{1}{M}\sum_{i=1}^{M} logLik_{max}(p,i) \quad \text{Para } p = 1, \cdots, P \qquad \text{(C.1)}$$

  donde:

  M: Es el número de repeticiones usadas, para este caso $M = 20$.

  P: Número de palabras de reconocimiento, en este caso $P = 8$ palabras.

- *FrontMin*: El umbral frontera mínima, se obtiene con las probabilidades logarítmicas máximas de todas las palabras. Las ecuaciones siguientes describen el procedimiento.

$$FrontMin = \mu - 4 \cdot \sigma \tag{C.2}$$

donde:

$$\mu = \frac{1}{M \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{M} logLik_{max}(p,i) \tag{C.3}$$

$$\sigma = \sqrt{\frac{(M \cdot P) \sum_{p=1}^{P} \sum_{i=1}^{M} logLik_{max}(p,i)^2 - \left(\sum_{p=1}^{P} \sum_{i=1}^{M} logLik_{max}(p,i) \right)^2}{(M \cdot P)(M \cdot P - 1)}} \tag{C.4}$$

- *UmbRiesgo* El umbral de riesgo establece la mínima dispersión utilizada para asegurar la palabra reconocida. El umbral se calcula usando las siguientes ecuaciones:

$$UmbRiesgo = \mu + 3{,}5 \cdot \sigma \tag{C.5}$$

donde:

$$\mu = \frac{1}{M \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{M} difLogLik(p,i) \tag{C.6}$$

$$difLogLik(p,i) = (logLik_{max}(p,i) - logLik_{prox}(p,i)) \tag{C.7}$$

$$\sigma = \sqrt{\frac{(M \cdot P) \sum_{p=1}^{P} \sum_{i=1}^{M} difLogLik(p,i)^2 - \left(\sum_{p=1}^{P} \sum_{i=1}^{M} difLogLik(p,i) \right)^2}{(M \cdot P)(M \cdot P - 1)}} \tag{C.8}$$

- *FrontInf(p)* Este umbral determina la frontera superior en cada palabra p. La manera de obtener este umbral se describe en las siguientes ecuaciones:

$$FrontInf(p) = \mu_p - 2{,}1 \cdot \sigma_p \tag{C.9}$$

donde:

$$\mu_p = \frac{1}{M} \sum_{i=1}^{M} logLik_{max}(p,i) \tag{C.10}$$

$$\sigma_p = \sqrt{\frac{M\sum_{i=1}^{M} logLik_{max}(p,i)^2 - \left(\sum_{i=1}^{M} logLik_{max}(p,i)\right)^2}{M(M-1)}} \qquad \text{(C.11)}$$

La figura C.1 muestra la gráfica de comparación de los umbrales de riesgo con respecto a las probabilidades logarítmicas máximas de la palabra "*Abajo*".

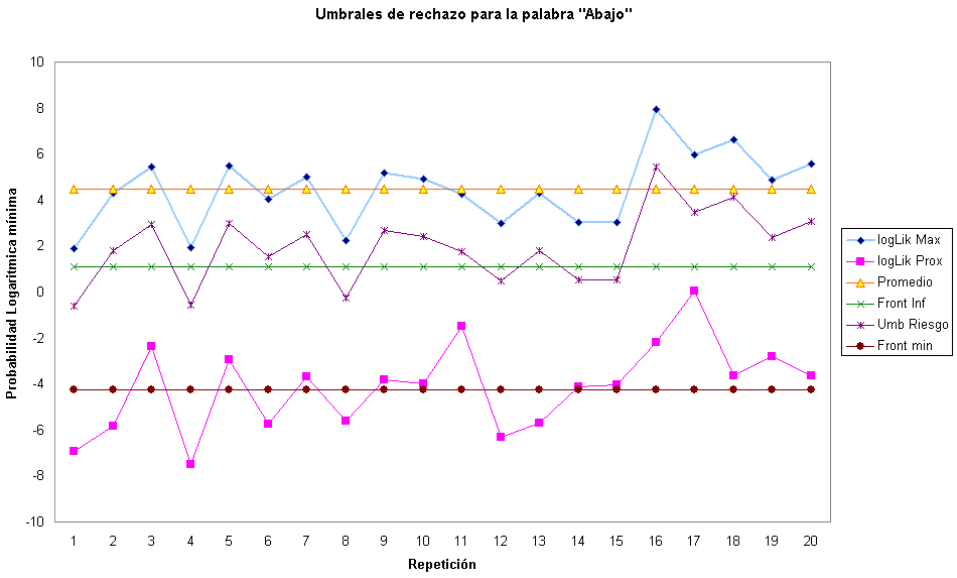


Figura C.1: Comparación de los umbrales de riesgo con las probabilidades logarítmicas máximas de las repeticiones para la palabra "*Abajo*".

Bibliografía

[Nieto, López 02] Nieto Omar, López Víctor, *"Reconocimiento de Comandos verbales en DSP's"*, Tesis, octubre 2002, Facultad de Ingeniería, UNAM.

[Rabiner 93] L. Rabiner, B. H. Juang, *"Fundamentals of Speech Recognition"*, Editorial Prentice-Hall, 1993

[Quilis 99] Quilis Antonio, *"Tratado de fonología y fonética españolas"*, Segunda edición, Editorial Gredos, Madrid España, 1999.

[Alcina Blecua 01] Alcina J., Blecua J. M., *"Gramática española"*, 11ª edición, Editorial Ariel, Barcelona España, 2001.

[Tortora Anagnostakos 99] G. J. Tortora, N. P. Anagnostakos *"Principios de anatomía y fisiología"*, Sexta edición, Oxford University Press, 2002.

[Crafts 91] Crafts R. C. *"Anatomía humana funcional"*, Editorial Noriega, 1991.

[Perello Peres 77] Perello J., Peres J., *"Fisiología de la comunicación oral"*, Segunda edición, Editorial Científico-Médica, 1977.

[Fuentes De Lara 97] Fuentes S. R., De Lara S. G. *"Corpus: Anatomía Humana General"*, Vol II, Editorial Trillas, México, Mayo 1997.

[Rabiner Schafer 78] L. R. Rabiner, R. W. Schafer, *"Digital Processing of Speech Signals"*, Editorial Prentice-Hall, 1978

[Becchetti Ricotti 99] C. Becchetti, L. P. Ricotti, *"Speech Recognition: Theory and C++ Implementation"*, Editorial Wiley, 1999.

[Jelinek 97] F. Jelinek, *"Statistical Methods for Speech Recognition"*, MIT Press, Massachusetts London, England, 1997.

[Martens 00] Jean-Pierre Martens, *"Continuous Speech Recognition over the Telephone"*, Final Report of COST Action 249, Electronic & Information Systems, Ghent University, May 2000.

[Rabiner 75] L. R. Rabiner y M. R. Sambur, *"An algorithm for determining the endpoints of isolated utterance"*, Bell Syst. Tech. J., vol. 54 no. 2, pp. 297-315, 1975

[Brigham 74] E. O. Brigham, *"The Fast Fourier Transform"*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

[Rader 69] B. Gold y C. M. Rader, *"Digital Processing of Signals"*, McGraw-Hill, New Rotk, 1969.

[DeFatta 88] D. J. Deffatta, J. G. Lucas y W. S. Hodgkiss *"Digital Signal Processing"*, John Wiley, New Rotk, 1988.

[Oppenheim 89] A. V. Oppenheim and R. W. Schafer, *"Discrete-Time Signal Processing"*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[Gardida 98] Gardida Degollado Arturo *"Reconocimiento de Palabras Aisladas Utilizando Cuantización Vectorial"*, Tesis de Ingeniería en Telecomunicaciones, FI, UNAM, 1998.

[Oppenheim 68] A. V. Oppenheim, R. W. Schafer, T.G. Stockham Jr., *"Nonlinear filtering of multiplied and convolved signals"*, Proc. IEEE, vol. 56, pp. 1264-1291, Aug. 1968.

[Quatieri 02] T. F. Quatieri, *"Discrete Time Speech Signal Processing: Principles and Practice"*, Upper Saddle River NJ, Prentice Hall, 2002.

[Davis y Mermelstein 80] Davis, S. y P. Mermelstein, *"Comparison of Parametric Representations for Monosyllabic Word Recognition"*, IEEE Trans. on Acoustics, Speech and Signal Processing, 1980, 28(4), pp. 357-366.

[Shannon y Paliwal 03] B. J. Shannon, K. K. Paliwal *"A comparative Study of Filter Bank Spacing for Speech Recognition"*, Microelectronic Engineering Research Conference, 2003

[Seneff 88] Stephanie Seneff *"A join synchrony/mean-rate model of auditory speech processing"*, Proc. J. of Phonetics, vol. 16, pp. 55-76, January 1988.

[Huang, Acero y Hon 01] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon, *"Spoken Language Processing: A Guide to Theory, Algorithm and System Development"*, 1st Prentice Hall, Abril 2001.

[Hedelin 95] Hedelin, P., P. Knagenhjelm, and M. Skoglund, *"Vector Quantization for Speech Transmission"*, in Speech Coding and Synthesis, W.B. Kleijn and K.K. Paliwal, eds. 1995, Amsterdam, pp. 311-396.

[Gersho y Gray 97] Gersho, A., Gray, R. M., *"Vector Quantization and Signal Compression"*, Sexta Edición. Editorial Kluwer Academic Publishers. Norwell, Massachusetts, U.S.A., 1997.

[Deller, Proakis y Hansen 87] Deller, Proakis y Hansen. *"Discrete-Time Processing of Speech Signals"*, Tercera Edición. Editorial Prentice Hall. New Jersey, U.S.A., 1987

[Owens 93] Owens F.J., *"Signal Processing of Speech"*, Editorial Magraw Hill. New York, U.S.A., 1993

[MacQueen 67] J. B. MacQueen, *"Some Methods for classification and Analysis of Multivariate Observations"*, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press, 1967.

[Baum 67] L.E. Baum, J.A. Eagon, *"An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology"*, Bulletin of American Mathematical Society, 1967, 73, pp. 360-363.

[Baker 75] J. K. Baker, *"The dragon system -An overview"*, IEEE Trans. Acoustics, Speech, Signal Proc., Febrero de 1975.

[Jelinek 75] F. Jelinek, L. P. Bahl, R. L. Mercer, *"Design of a linguistic statistical decoder for the recognition of continuous speech"*, IEEE Trans. Information Theory, 1975.

[Rabiner 89] Rabiner, L.R., *"A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition"*, Proc. of IEEE, 1989, 77(2), pp. 257-286.

[Baum Egon 67] L. E. Baum y J. A. Egon, *"An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology"*, Bull. Amer. Meteorol. Soc., vol 73, pp360-363, 1967.

[Baum Sell 68] L. E. Baum y G. R. Sell, *"Growth functions for transformations on manifolds"*, Pac. J. Math., vol 27, no. 2, pp211-227, 1968.

[Viterbi 67] A. J. Viterbi, *"Error bounds for convolutional codes and an asymptotically optimal decoding algorithm"*, IEEE Transactions on Information Theory, vol. IT-13, pp. 260-269 Abril de 1967.

[Forney 73] G. D. Forney, *"The Viterbi algorithm"*, Proc. IEEE vol. 61, pp 268-278 Marzo de 1973.

[Dempster Laird Rubin 77] A. P. Dempster, N. M. Laird y D. B. Rubin, *"Maximum likelihood from incomplete data via the EM algorithm"*, J. Roy-Stat. Soc., vol. 39, no. 1, pp. 1-38, 1977.

[Levinson, Rabiner 83] S. E. Levinson, L. R. Rabiner, *"An Introduction to the Application of the theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition"*, Bell system Tech. J. pp 1035-1074, Abril de 1983.

[Juang Levinson Sondhi 86] B. H. Juang, S. E. Levinson y M. M. Sondhi, *"Maximum likelihood estimations for multivariate mixture observations of Markov chains"*, IEEE Trans. Information Theory, pp, 307-309, Marzo de 1986.

[SPRU733] *"TMS320C67x/C67x+ DSP CPU and Instruction Set Reference Guide"*, Texas Instruments, Num. de literatura: SPRU733, Mayo de 2005.

[SPRU190D] *"TMS320C6000 Peripherals Reference Guide"*, Texas Instruments, Num. de literatura: SPRU190D, Febrero de 2001.

[SPRU328B] *"Code Composer Studio User's Guide"*, Texas Instruments, Num. de literatura: SPRU328B, Febrero de 2000.

[SPRU423] *"TMS320 DSP/BIOS User's Guide"*, Texas Instruments, Num. de literatura: SPRU423, Febrero de 2001.

[SPRU186I] *"TMS320 Assembly Language Tools User's Guide"*, Texas Instruments, Num. de literatura: SPRU186I, Abril de 2001.

[SPRU198F] *"TMS320C6000 Programmer's Guide"*, Texas Instruments, Num. de literatura: SPRU198F, Febrero de 2001.

[SLAS202B] *"TLC320AD535C/I Dual Channel Voice/Data Codec Data Manual"*, Texas Instruments, Num. de literatura: SLAS202B, 2000.

[SPRU616] *"DSP/BIOS Driver Developer's Guide"*, Texas Instruments, Num. de literatura: SPRU616, Noviembre 2002.

[SPRU423] *"TMS320 DSP/BIOS User's Guide"*, Texas Instruments, Num. de literatura: SPRU423, Noviembre 2002.