Escola Politècnica Superior
d'Enginyeria de Vilanova i la Geltrú
UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROJECTE FI DE CARRERA

TÍTOL: Diseño e implementación de un sistema de control por voz

AUTOR: Jordi Montero Mata

TITULACIÓ: Enginyeria Tècnica Industrial Electrònica Industrial

DIRECTOR: Dr. Abel Torres Cebrián

DEPARTAMENT: ESAII – Enginyeria de Sistemes, Automàtica i Informàtica Industrial

DATA:

TÍTOL: Diseño e implementación de un sistema de control por voz

COGNOMS: Montero Mata **NOM: Jordi**

TITULACIÓ: Enginyeria Tècnica Industrial

ESPECIALITAT: Electrònica Industrial **PLA: 95**

DIRECTOR: Dr. Abel Torres Cebrián

DEPARTAMENT: ESAII – Enginyeria de Sistemes, Automàtica i Informàtica Industrial

QUALIFICACIÓ DEL PFC

TRIBUNAL

PRESIDENT **SECRETARI** **VOCAL**

DATA DE LECTURA:

Aquest Projecte té en compte aspectes mediambientals: □ Sí □ No

PROJECTE FI DE CARRERA

RESUM (màxim 50 línies)

El presente proyecto final de carrera consiste en el diseño e implementación de un sistema de reconocimiento de voz con una interfaz amigable que permita al usuario un manejo sencillo del sistema, Este sistema tiene como objetivo proporcionar una ayuda a las personas con alguna disminución física.

El software elegido para la implementación del sistema el entorno de programación Matlab utilizando las librerias de tratamiento de señales y adquisición de datos. Las razones por las cuales se ha escogido Matlab son las ventajas que presenta respecto a otros entornos algunas de estas ventajas pueden ser: potencia de cálculos, posibilidad de programar una interfaz gráfica fácilmente que permita al usuario ejecutar el programa que sea necesario que tenga conocimientos de programación, etc...
Para llevar a cabo la implementación del sistema se han tenido en cuenta las dos principales tecnicas que se utilizan para el reconocimiento del habla que son: reconocimiento utilizando Redes Neuronales artificiales y reconocimento utilizando Modelos Ocultos de Markov.

Paraules clau (màxim 10):

Reconocimiento	voz	Matlab	Redes Neuronales Artificiales
Interfaz	HMM	Coeficientes ceptrum	LPC

Índice

1. Introducción

Se denomina reconocimiento del habla al proceso de extraer información lingüística de una señal de voz. Este proceso que el ser humano es capaz de llevar a cabo automáticamente y casi inconscientemente es extremadamente complejo prueba de ello son los estudios realizados sobre la psicofísica de la percepción humana de habla que ponen de manifiesto esta complejidad y demuestran que en este proceso intervienen complejos procesos cerebrales relacionados con el lenguaje, son estos procesos precisamente los que convierten el ser humano en el sistema de reconocimiento del habla más robusto de hecho el ser humano es capaz de extraer información lingüística en ambientes ruidosos, con falta de información o incluso con información errónea.

El reconocimiento del habla es aplicable a una gran variedad de situaciones donde se requiera una comunicación hombre-máquina algunos de estos ejemplos de aplicación serian: la redacción de textos sin teclado, telefonistas automáticas, ayuda a discapacitados físicos, etc…
Uno de los objetivos del presente proyecto es precisamente el de propocionar ayuda a las personas discapacitadas.
Los dèficits corporales, de carácter visual, auditivo o motriz, debidos a una enfermedad, un accidente o al propi ciclo de la vida, frecuentemente comportan la aparición de un conjunto de limitacinones en las actividades cotidianas de las personas que las padecen. Estas limitaciones se traducen en una restricción y una reducción de la participación de los colectivos de las personas con discapacidades o de la gente mayor en la sociedad, produciendose a demás, la dependencia de terceras personas.

El presente proyecto se podría implementar sobre un sistema de agentes portables, autónomos, sin mantenimiento e inteligentes que cooperen entre si y puedan adaptarse a cualquier tipo de usuario con la intención de mejorar la autonomía y el acceso a la información de colectivos con déficits y limitaciones concretas.
El sistema que interactuaria con el usuario dispondría de un transmisor-receptor de corto alcance. Este sistema se situaría en una pulsera que se colocaría el usuario en la muñeca; de forma que cuando este acerque la mano a un objeto, que dispone de una etiqueta inteligente, pueda identificarlo y/o localizarlo mediante un mensaje oral. Las etiquetas inteligentes estarian formadas por memoria, sensores, sistema recolector de energía (sin batería), sistema transmisor-receptor por radiofrecuencia y un sistema de control. Cuando la etiqueta inteligente detectase una transmisión del sistema de usuario, enviaría la información de la memoria y sensores al usuario.

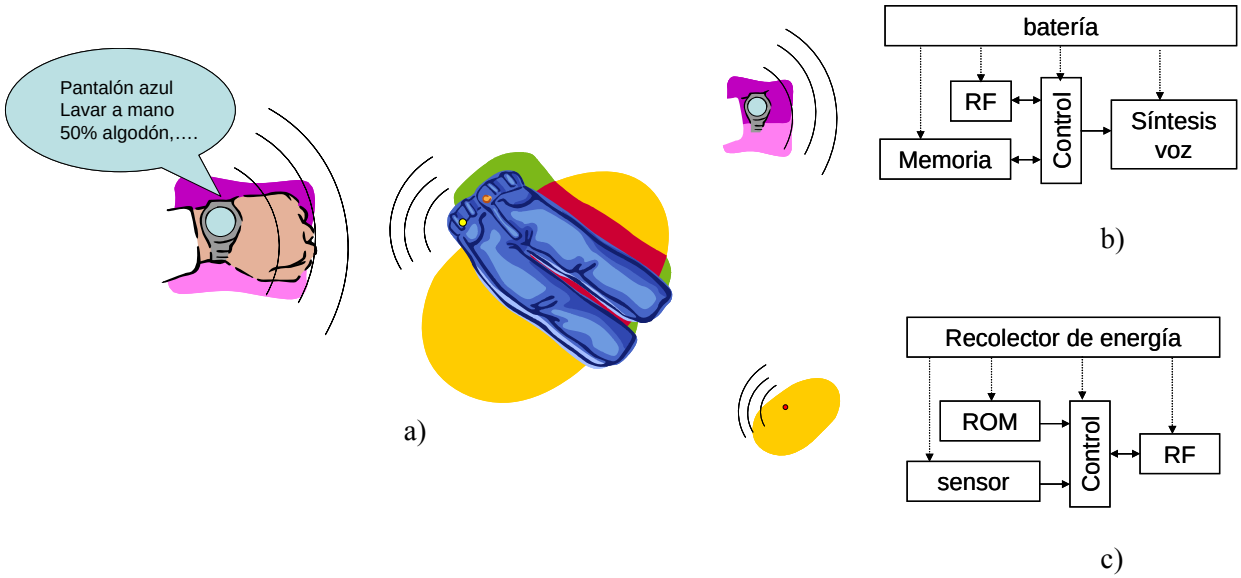


Figura 1. a) Esquema de funcionamient de la implementación en un sistema de objetos portables B) Diagrama de bloques del sistema de interactuación con el usuario. C) Diagrama de bloques de una etiqueta inteligente

En definitiva la disponibilidad de la tecnología que se plantea desarrollar presenta innumerables aplicaciones prácticas para todos aquellos ciudadanos con algún tipo de déficit visual en esta fase inicial del proyecto para ampliarlo a personas con diversos grados de dependencia. Entre estas aplicaciones se destacan las relacionadas con el área de accesibilidad a la información y autonomía personal. A continuación se detallan algunas de las más interesantes:

Identificación i localización de productos en comercios o el hogar

La aplicación consiste en incrustar una etiqueta inteligente en productos comerciales. Esta etiqueta permitirá a un usuario, que utilice una pulsera complementaria, localizar y/o identificar de que artículo se trata, así como de recibir alarmas de si se está utilizando de forma inadecuada. Por ejemplo, si se tratara de un producto alimenticio que requiriese refrigeración, el sistema permitiría localizarlo en un supermercado, informa al usuario de su valor nutricional y fecha de caducidad, así como activar una alarma en el caso de que se dejase fuera del refrigerador. Otro ejemplo ilustrativo consistiría en ubicar una etiqueta en una prenda; el usuario podría conocer el color de la prenda simplemente acercando su mano a ésta y si la pusiera en una lavadora que recibiera una alarma en el caso de necesitar un programa delicado.

Identificación e interactuación con servicios de transporte público

Existen incontables aplicaciones para el transporte público de pasajeros. Por ejemplo, en el sistema de autobuses de cualquier ciudad. Los ciudadanos con déficit visual tienen problemas

para identificar que autobús se acerca o que ruta sigue dicho autobús. Además podría activar el sistema de plataforma para silla de ruedas automáticamente, en el caso de un déficit motriz. Permitiría también compensar las variaciones en los horarios y accesos en las estaciones de tren o aeropuertos; ya que dichas variaciones pueden no ser percibidas , ni saber resolverse de forma autónoma por tener limitada la lectura de los paneles informativos.

Posología y seguimiento de medicación recetada

Esta tecnología se podría aplicar en los tratamientos médicos si en las farmacias el farmacéutico programara las etiquetas inteligentes, a partir de las especificaciones de las recetas médicas, con la posología que debe realizar el paciente. De esta forma un ciudadano con déficit visual, por ejemplo, no tan solo podría localizar cada medicamento recetado, si no que recibiría un conjunto de alarmas sobre cuando tomárselo, el doctor podría contrastar también el seguimiento del tratamiento.

Prevención de problemas

Si estos dispositivos inteligentes y autónomos se implantasen los ciudadanos con dependencia de terceros. Los sensores de los agentes podrían ser sus constantes vitales, movimientos y ubicación para generar alarmas y transmitirlas al familiar o cuidador responsable en caso de prever una situación de riesgo.

Una característica importante del reconocimiento de la voz es su naturaleza multidisciplinar. Los problemas a resolver en el diseño de un sistema reconocedor están relacionados con una gran variedad de disciplinas, tales como la acústica, el proceso de la señal, reconocimiento de patrones, fonética, lingüística e informática.

El problema del reconocimiento del habla puede afrontarse desde dos enfoques distintos: reconocimiento conducido por los datos o por el conocimiento. Esta distinción se refiere a la forma en que se tiene en cuenta el conocimiento necesario para llevar a cabo la tarea del reconocimiento. En el enfoque estadístico (Modelos de Markov o Redes Neuronales Artificiales), el conocimiento se extrae a partir de ejemplos por medio de técnicas de aprendizaje. En el otro extremo, en el enfoque basado en el conocimiento (Inteligencia Artificial) se intenta razonar sobre todas las fuentes de conocimiento (fonética, léxico, sintaxis, semántica, etc.) para finalmente obtener una interpretación asociada al conjunto de características acústicas medidas.

1.1 Objetivos del Proyecto

El objetivo principal del proyecto consiste en el diseño y la implantación de un sistema de control por voz para un sistema de etiquetaje inteligente, que permita la interacción entre el usuario y el sistema mediante mensajes orales. A su vez, además de poder realizar etiquetaje inteligente deberá ser capaz de controlar otros dispositivos como pueden ser una silla de ruedas, el puntero del ratón o en definitiva cualquier otro objeto que proporcione una ayuda a las personas discapacitadas. A continuación se exponen los objetivos marcados para el presente proyecto:

Definición del problema y búsqueda de información sobre los diferentes sistemas de reconocimiento de voz existentes.
Diseño e implementación sobre un entorno de programación MATLAB de un sistema de reconocimiento de voz.
Creación de una interfaz gráfica que proporcione al usuario un entorno amigable
Comprobación del funcionamiento del sistema

El presente proyecto forma parte del proyecto SAPIENS que se enmarca dentro de las actuaciones de la cátedra de Accesibilidad –arquitectura, diseño y tecnología para todos de la UPC. En concreto el proyecto SAPIENS se orienta de manera específica a las personas con déficit visual, a través del desarrollo de un sistema que hace de interfaz con el usuario, de forma amigable, para la identificación, localización e interacción con cualquier objeto de la vida cotidiana.

2 La voz humana

2.1 Anatomía del aparto fonatorio

El aparato fonatorio está constituido por la laringe constituida por nueve cartílagos tres impares (cricoides, tiroides, epiglotis) y tres pares (aritenoides, corniculados y cuneiformes).En los cartílagos aritenoides, específicamente en sus apófisis vocales, se encuentran las cuerdas vocales las cuales se unen al cartílago tiroides.

Las cuerdas vocales en realidad son dos membranas en la laringe orientadas de adelante hacia atrás la abertura entre las cuerdas se denomina glotis.

Cuando las cuerdas vocales se encuentran separadas, la glotis adopta una forma triangular. El aire pasa libremente y prácticamente no se produce sonido. Es el caso de la respiración.

Cuando la glotis comienza a cerrarse, el aire que la atraviesa proveniente de los pulmones experimenta una turbulencia, emitiéndose un ruido de origen aerodinámico conocido como aspiración (aunque en realidad acompaña a una espiración o exhalación).

Esto sucede en los sonidos denominados “aspirados” (como la h). Al cerrarse más, las cuerdas vocales comienzan a vibrar a modo de lengüetas, produciéndose un sonido tonal, es decir periódico. La frecuencia de este sonido (denominada también pitch) depende de varios factores, entre otros del tamaño y la masa de las cuerdas vocales, de la tensión que se les aplique y de la velocidad del flujo del aire proveniente de los pulmones. A mayor tamaño, menor frecuencia de vibración, lo cual explica por qué en los varones, cuya glotis es en promedio mayor que la de las mujeres, la voz es en general más grave. A mayor tensión la frecuencia aumenta, siendo los sonidos más agudos.

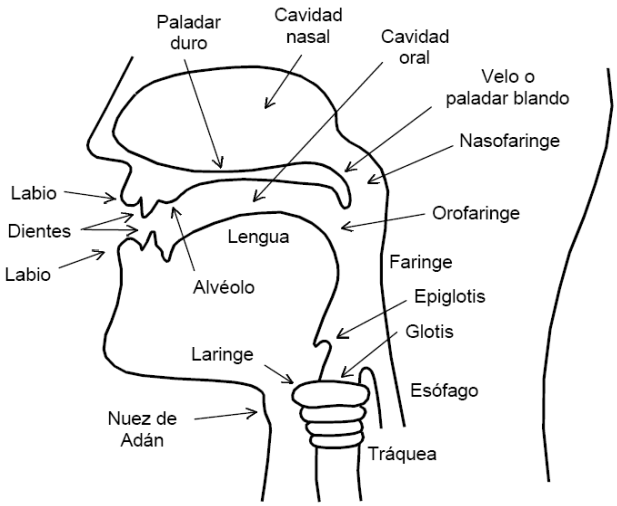


Figura 2.1. Corte esquemático del aparato fonador humano

Cuando la glotis esta cerrada totalmente no se produce ningún sonido. Sobre la glotis se encuentra la epiglotis un cartílago que permite tapar la laringe la glotis durante la ingesta de alimentos para evitar que estos pasen al tracto respiratorio. Durante la respiración y la fonación (emisión de sonido) la epiglotis está separada de la glotis permitiendo la circulación del flujo de aire. Durante la deglución, en cambio, la laringe ejecuta un movimiento ascendente de modo que la glotis apoya sobre la epiglotis.

Las cavidades situadas por encima de la laringe es decir cavidades nasal, oral junto con los elementos articulares se denomina cavidad supraglotica mientras que espacios por debajo de la laringe, es decir la tráquea, los bronquios y los pulmones, se denominan cavidades infraglóticas.

Algunos de los elementos articulares de la cavidad supraglotica se pueden mover a nuestra voluntad modificando el sonido producido por la vibración de las cuerdas vocales e incluso producir sonidos propios. Todo esto se produce por dos mecanismos principales el filtrado y la articulación.

El filtrado actúa modificando el espectro del sonido. Tiene lugar en las cuatro cavidades supraglóticas principales: la faringe, la cavidad nasal, la cavidad oral y la cavidad labial. Las mismas constituyen resonadores acústicos que enfatizan determinadas bandas frecuencia les del espectro generado por las cuerdas vocales, conduciendo al concepto de formantes, es decir una serie de picos de resonancia ubicados en frecuencias o bandas de frecuencias que son características de cada tipo de sonido.

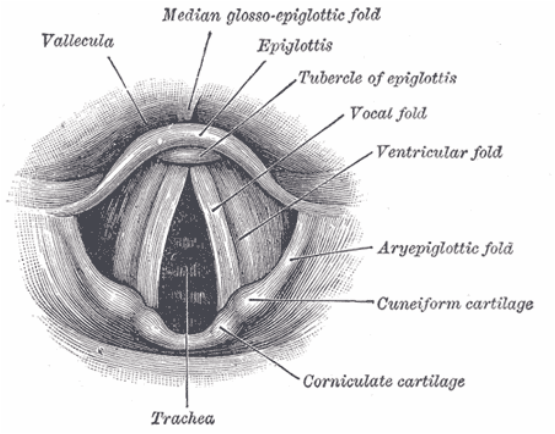


Figura 2.2. Dibujo de las cuerdas vocales según la anatomía de Gray

La articulación es una modificación temporal de los sonidos, y esta relacionada con la emisión de estos y con los fenómenos transitorios que los acompañan. Se caracteriza por el lugar del tracto vocal donde tiene lugar, por los elementos que intervienen y por el modo en que se produce dando lugar a otro tipo de clasificación que se verá más adelante.

2.2 Clasificación de los sonidos

Los sonidos se pueden clasificar teniendo en cuenta sus características de emisión, de esta manera podemos hacer la siguiente clasificación:

Según sean vocales o consonantes.
Según su oralidad nasalidad.
Según su carácter tonal (sonoro) o sordo.
Según el lugar de articulación
Según el modo de articulación
Según la posición de los órganos articulatorios

2.2.1 Vocales o consonantes.

En el caso de los sonidos vocálicos podemos decir que el aire sale de los pulmones sin ningún tipo de impedimento de manera que el sonido se produce únicamente por la vibración de las cuerdas vocales. Las consonantes, por el contrario, se emiten interponiendo algún obstáculo formado por los elementos articulatorios. Los sonidos correspondientes a las consonantes pueden ser tonales o no dependiendo de si las cuerdas vocales están vibrando o no. Funcionalmente, en el castellano las vocales

pueden constituir palabras completas, no así las consonantes.

2.2.2 Oralidad y nasalidad

Los fonemas en los que el aire pasa por la cavidad nasal se denominan nasales, en tanto que aquéllos en los que sale por la boca se denominan orales. La diferencia principal está en el tipo de resonador principal por encima de la laringe (cavidad nasal y oral, respectivamente). En castellano son nasales sólo las consonantes “m”, “n”, “ñ”.

2.2.3 Tonalidad

Los fonemas en los cuales hay una vibración de las cuerdas vocales se denominan Tonales o sonoros esto implica que dichos sonidos contienen una componente frecuencial casi periódica. Como se ha dicho anteriormente todas las vocales son sonoras pero existen también una serie de consonantes que poseen esta característica un ejemplo sería: “b”, “d”, “m”. Los fonemas en los cuales no hay una vibración de las cuerdas vocales de denominan sordos un ejemplo de estos sonidos son: “s”, “f”, “c”.

2.2.4 Lugar y modo de articulación (Consonantes)

La articulación es el proceso mediante el cual alguna parte del aparato fonatorio interpone un obstáculo para la circulación del flujo de aire. Las características de la articulación permitirán clasificar las consonantes. Los órganos articulatorios son los labios, los dientes, las diferentes partes del paladar (alvéolo, paladar duro, paladar blando o velo), la lengua y la glotis. Salvo la glotis, que puede articular por sí misma, el resto de los órganos articula por oposición con otro. Según el lugar o punto de articulación se tienen fonemas:

-Bilabiales: oposición de ambos labios ejemplos: “p”, “b”, “m” etc...

-Labiodentales: oposición de los dientes superiores con el labio inferior ejemplos: “f”

-Linguodentales: oposición de la punta de la lengua con los dientes superiores ejemplos: “z”, “d”.

-Alveolares: oposición de la punta de la lengua con la región alveolar ejemplos: “n”, “s”.

-Palatales: oposición de la lengua con el paladar duro ejemplos: “j”, “l”.

-Velares: oposición de la parte posterior de la lengua con el paladar blando ejemplos: “k”, “g”, “x”

-Glotales: articulación en la propia glotis ejemplo: “h”.

A su vez, para cada punto de articulación ésta puede efectuarse de diferentes modos,
dando lugar a fonemas: Oclusivos (la salida de aire se cierra momentáneamente por completo), Fricativos (el aire sale atravesando un espacio muy estrecho), Africados (oclusión seguida por fricación), Laterales (la lengua obstruye el centro de la boca y el aire sale por los lados), Vibrantes (la lengua vibra cerrando el paso del aire intermitentemente), Aproximantes (la obstrucción muy estrecha que no llega a producir turbulencia).

2.2.5 Posición de los órganos articulatorios (vocales)

En el caso de las vocales los órganos articulatorios que intervienen en la variación de los formantes de la voz son la lengua (tanto la profundidad como la altura donde esta este situada), los labios y el paladar blando.
De esta manera tanto la abertura de la boca como la posición de la lengua influyen sobre el primer y segundo formante respectivamente como se puede observar en la fig 2.3

Posición vertical	Tipo de vocal	Posición horizontal (avance)		
		Anterior	Central	Posterior
Alta	Abierta	i		u
Media	Media	e		o
Baja	Cerrada		a	

Tabla 1. Clasificación de las vocales en función de la posición de la lengua

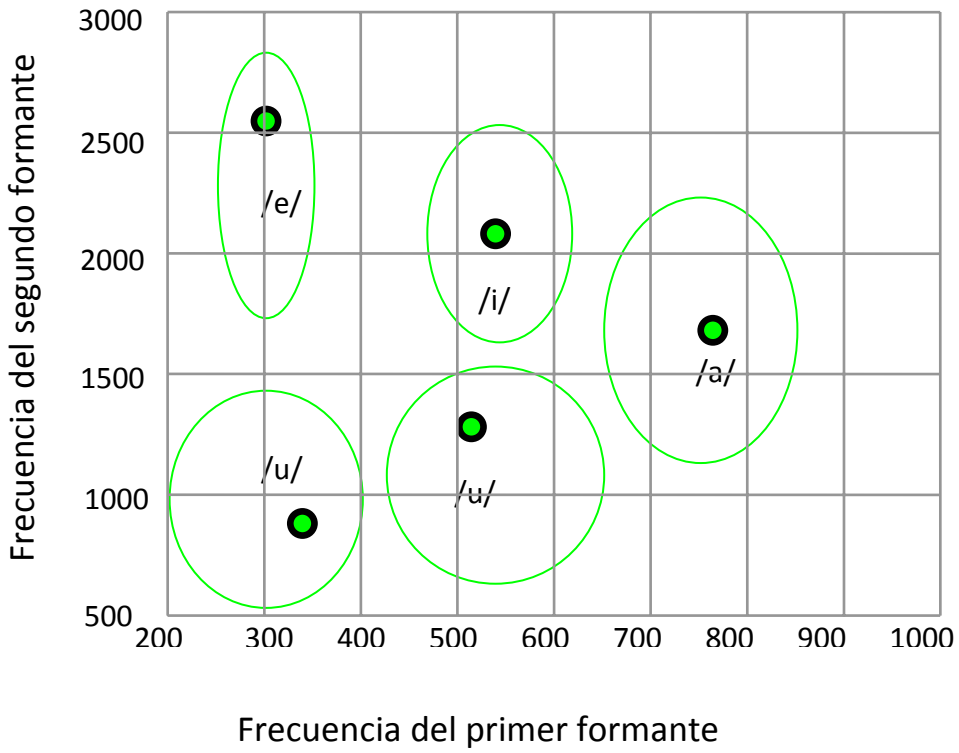


Figura 2.3. Carta de formantes

Otra cualidad controlable en los fonemas vocálicos es la posición de los labios que es conocido como labialización, las vocales labializadas también denominadas redondeadas son las que para pronunciarlas hay que redondear los labios extendiendo así el tracto vocal estas vocales son la /u/ y la /o/.

3. Técnicas de análisis de la señal de voz

3.1 Enventanado de la señal

La señal de voz es un proceso aleatorio y no estacionario. Esto supone un inconveniente a la hora de analizar la señal, no obstante es posible salvar este problema si se tiene en cuenta que a corto plazo de tiempo (del orden de ms) la señal es casi-estacionaria esto da lugar a un tipo de análisis donde se obtienen segmentos o TRAMAS de señal de pocos ms denominado análisis localizado. A este proceso donde se obtienen TRAMAS o segmentos consecutivos de señal se le denomina enventanado.

El enventanado requiere que cada una de las TRAMAS sea multiplicada por una función limitada en el tiempo de tal manera que su valor fuera de su intervalo sea nulo.

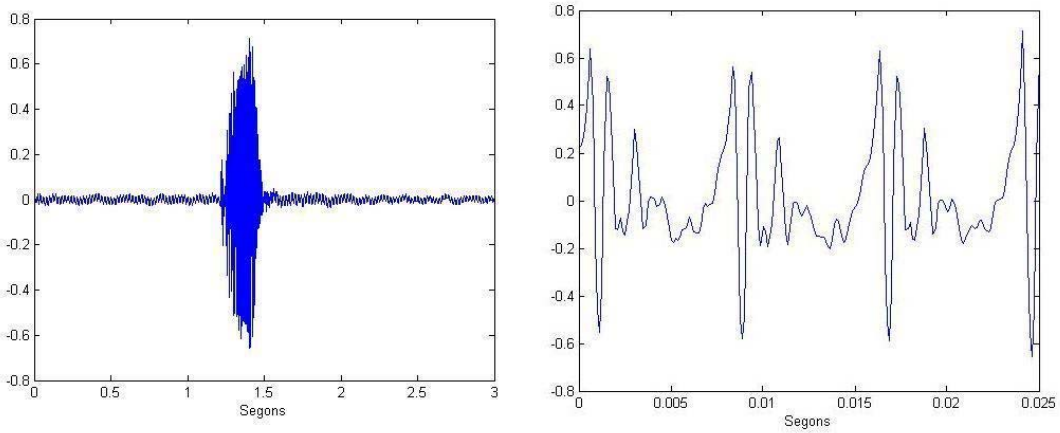


Fig. 3.1 A la izquierda se puede ver el oscilograma de la pronunciación de la vocal /a/ y a la derecha se puede observar como en corto espacio de tiempo la señal es casi-

De los diferentes tipos de ventana podemos destacar dos: Ventana rectangular y Ventana Hamming.

3.1.1 Ventana Hamming

La utilización de una ventana rectangular tiene el inconveniente que en los extremos de dicha ventana la función decae rápidamente y esto da lugar a que se produzca el fenómeno de Gibbs. Para evitar este fenómeno es necesario una ventana cuya función tenga una caída suave en sus extremos una posible solución es utilizar una ventana Hamming.

Función que define la ventana Hamming:

$$w(n)\begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) & para\ 0 \le n \le N-1 \\ 0 & resto \end{cases}$$

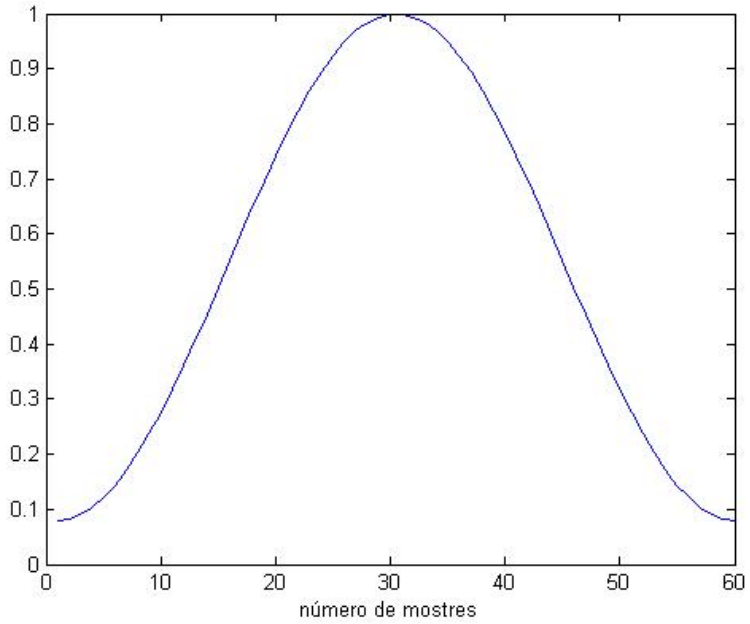


Fig. 3.2 Ventana del tipo Hamming de 60 muestras

Al utilizar una ventana Hamming es preciso tener en cuenta que las muestras en los extremos de la ventana sufrirán una ponderación a diferencia de las muestras de la zona central cuyo valor no experimentara ningún cambio, para compensar este efecto de ponderación se hace necesario un solapamiento de las ventanas de tal manera que el desplazamiento de la ventana sea inferior a la longitud de esta.

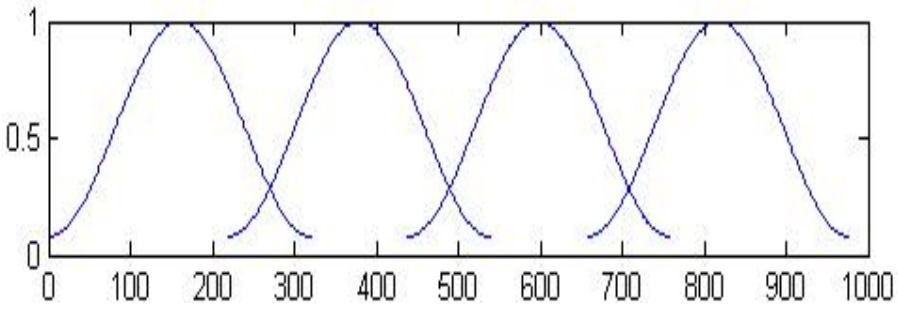


Fig. 3.3 Solapamiento entre ventanas Hamming

3.2 Extracción de características

El objetivo del análisis localizado es la extracción de una serie de características de la señal que nos ayuden no solo a reconocer el sonido de que se trata sino a determinar si la señal analizada es realmente una señal de voz y en el caso que efectivamente lo sea reconocer el fonema del que se trata. Para extraer dichas características existen dos tipos de análisis:

-Análisis en dominio temporal.

-Análisis en dominio frecuencial.

3.2.1 Análisis en dominio temporal

Uno de los problemas que se nos presenta a la hora de analizar una señal de voz es determinar cuando empieza y cuando finaliza la frase o palabra que estamos analizando es decir eliminar las Tramas que no contienen señal de voz. El análisis en dominio temporal no nos resulta muy útil a la hora de reconocer los fonemas por el contrario si que nos es muy útil para llevar acabo un preprocesado de la señal y de esta manera hacer una primera clasificación de Tramas que contienen sonido y Tramas que no contienen nada. De este tipo de analiza podemos extraer tres características interesantes:

-Energía localizada.

-Tasa de cruces por cero.

-Autocorrelación.

3.2.1.1 Energía localizada

La energía localizada viene determinada por la siguiente expresión:

$$E = \sum_{m=0}^{N-1} [W(m)X(n-m)]^2$$

donde W(n) es una función de peso o ventana que selecciona un segmento de X(n), y N es el número de muestras de la ventana. Una buena ventana para el cálculo de la energía debe tener una longitud entre 10 y 20 ms, si es más pequeña, menor de un periodo fundamental (pitch) la energía fluctúa muy rápidamente dependiendo de los detalles exactos de la onda. Si el intervalo de la ventana es más grande, de varios periodos fundamentales, la energía varía muy poco y no se reflejan las propiedades del cambio de la señal. El mayor significado de la energía es que proporciona una buena medida para separar segmentos de la voz vocálicos de no vocálicos. En el caso de que la señal sea de buena calidad, la energía se puede utilizar también para separar la voz no vocálica del silencio.

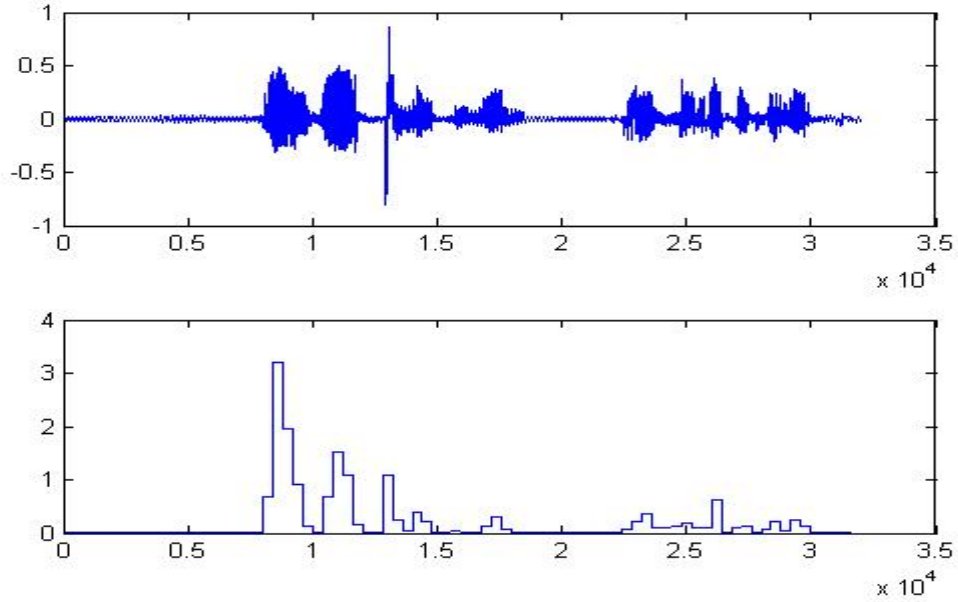


Fig. 3.4 Arriba se puede ver el oscilogarma de la frase “el golpe de timón fue sobrecogedor”. Abajo la energía localizada de la señal

3.2.1.2 Tasa de cruces por cero

El hecho de que una señal una vez discretizada cambie de signo entre dos muestras consecutivas es consecuencia de que la señal continua de la cual proviene ha pasado forzosamente por cero teniendo en cuenta esta observación podemos obtener un valor medio de las veces que la señal pasa por cero en cada ventana y así obtener la Tasa de cruces por cero.

$$Tc = \frac{1}{N}\sum_{n}\frac{1}{2}\,|\,\mathrm{sgn}[x(n)] - \mathrm{sgn}[x(n-1)]\,|\,w(m-n)$$

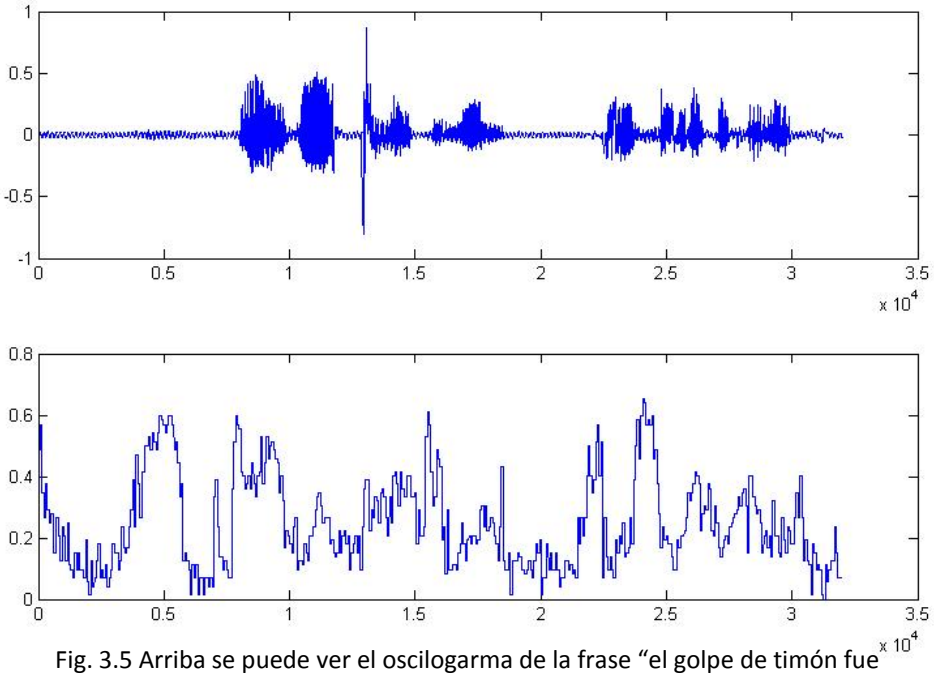


Fig. 3.5 Arriba se puede ver el oscilogarma de la frase "el golpe de timón fue sobrecogedor". Abajo la tasa de cruces por cero de la frase

La tasa de cruces por cero nos da información sobre el carácter sonoro o sordo de la señal ya que cuando el sonido es sordo la frecuencia de la señal aumenta.

3.2.1.3 Autocorrelación

La función de autocorrelación es el resultado de comparar la señal de voz consigo misma desplazada m muestras, una de las posibles aplicaciones de dicha comparación podría ser determinar el valor de la frecuencia fundamental o pitch en el caso de que la señal de voz sea sonora teniendo en cuenta que la autocorrelación adquiere su valor máximo cuando el desplazamiento es igual al valor de este.

$$R_m(k) = \sum_{n=0}^{N-1}\{w[m-n]x[n]\}\{w[m-(n+k)]x[n+k]\}$$

$$k = 0, 1, 2, \ldots \mathrm{p.}$$

Para calcular la autocorrelación es necesario que el tamaño de la ventana sea superior al periodo de la frecuencia fundamental si esto no se cumple no será posible calcular el valor de esta a partir de la función de Autocorrelación.

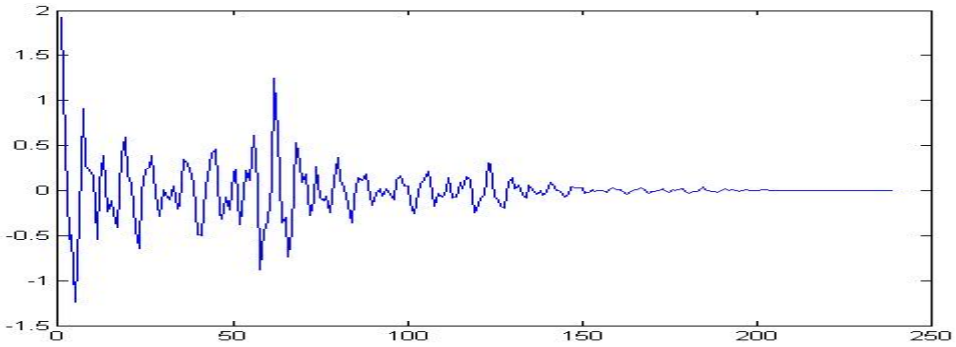


Fig. 3.6. En la imagen se puede ver la función de autocorrelación de una Trama de señal sonora como se puede observar hay un pico que corresponde al "pitch" o frecuencia fundamental.

Hasta ahora las técnicas comentadas para el análisis de señal voz se llevan acabo en el dominio temporal pero para extraer más características que nos ayuden a diferenciar los diferentes sonidos correctamente es necesario obtener la componente frecuencial de dichas señales.

3.2.2 Análisis en dominio frecuencial

3.2.2.1 Transformada localizada de fourier

La transformada de fourier es una herramienta que descompone la señal en sus componentes frecuenciales pero no determina el momento en el que se producen una solución a este problema es calcular la transformada de cada una de las ventanas y de esta manera realizar una Transformada Localizada de Fourier.

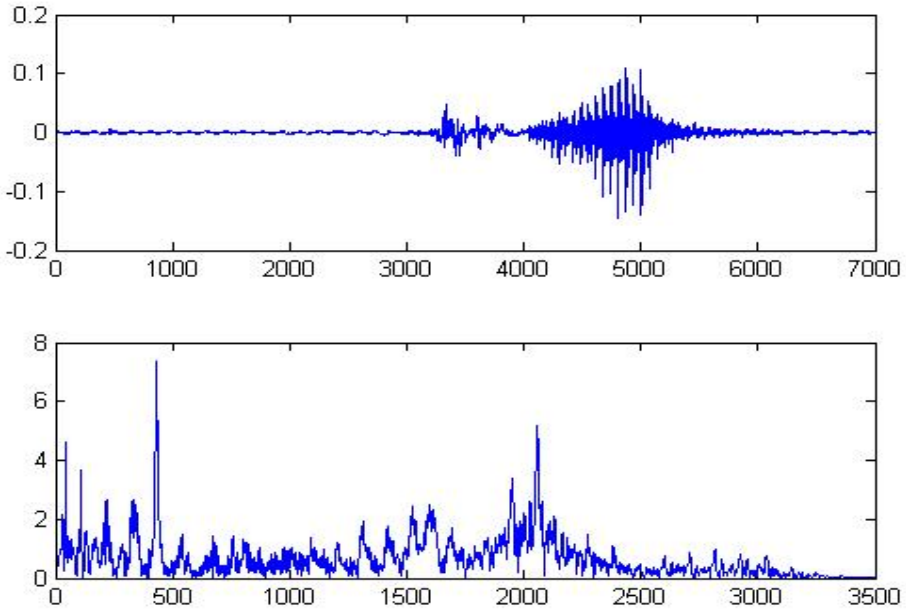


Fig. 3.7 Arriba se puede ver el oscilogarma de la palabra " fue ". Abajo la energía la transformada de fourier de la palabra, como se puede observar no nos proporciona la variación del espectro en función del tiempo.

3.2.2.2 Espectrograma

otra posible solución al problema que presenta utilizar la transformada de fourier para analizar el espectro de la señal es utilizar un espectrograma.
El espectrograma nos proporciona información sobre el espectro frecuencial por medio de una escala de grises o en otros casos en una escala de color sobre el eje vertical diferenciando la intensidad frecuencial en diferentes tonalidades y la variación frecuencial la altura sobre dicho eje. Otra característica del espectrograma es que sobre el eje horizontal se representan las ventanas de tiempo de la señal teniendo en cuenta esto podríamos distinguir entre dos tipos de espectrogramas:

Espectrograma de banda ancha: en este espectrograma se utiliza ventana muy ancha en consecuencia tendremos una muy buena resolución temporal, pero una mala resolución frecuencial.

Espectrograma de banda estrecha: en este espectrograma se utiliza una ventana estrecha y por lo tanto tendremos una muy buena resolución frecuencial pero una mala resolución temporal.

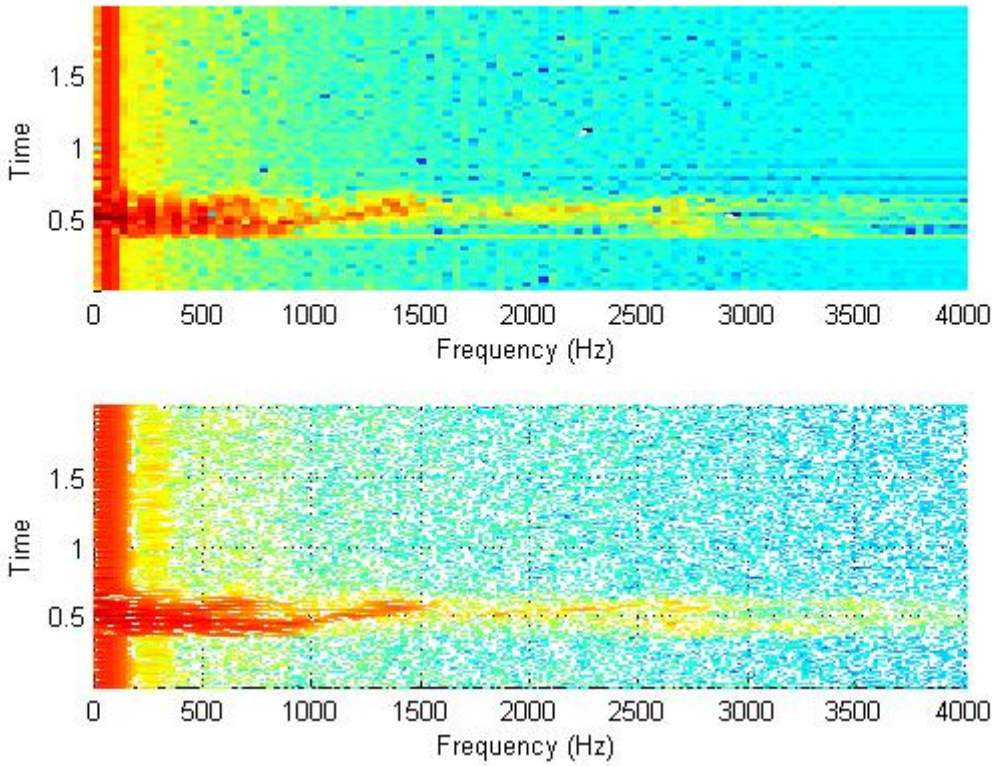


Fig. 3.8 Arriba se puede ver un espectrograma de banda ancha por lo tanto tenemos una buena resolución temporal, abajo en cambio tenemos un espectrograma de banda estrecha por lo que tendremos una buena resolución frecuencial

3.2.2.3 Coeficientes lpc

El aparato fonador se puede modelar como un sistema lineal. Este modelo cuya entrada puede ser un tren de impulsos en el caso que sea un sonido sonoro o bien ruido gaussiano en el caso que sea sordo. El predictor se puede expresar mediante la transformada Z según la expresión:

$$A(z) = a_1 z^{-1} + a_2 z^{-2} + \cdots + a_p z^{-p}$$

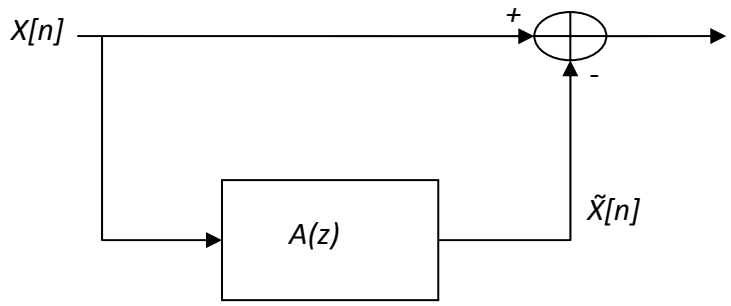


Fig. 3.9 Esquema de un predictor lineal a corto plazo

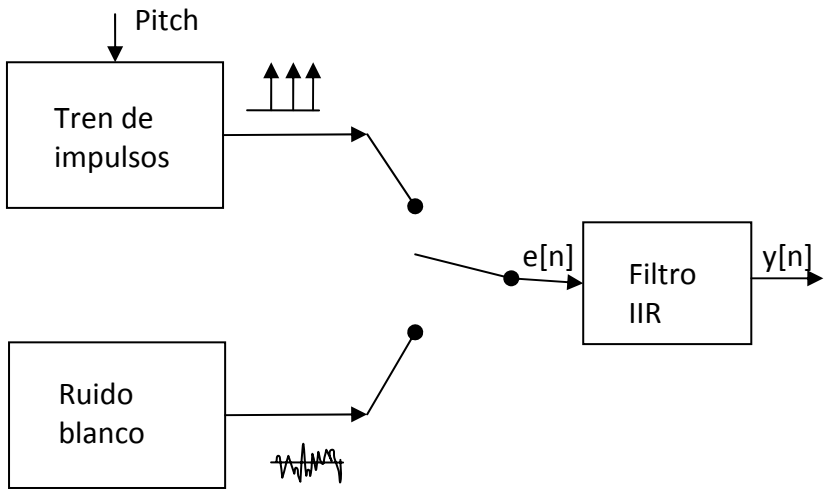


Fig. 3.10 Modelo del tracto vocal. Como se puede ver la entrada puede ser un tren de impulsos en caso de una señal sonora y ruido blanco en caso de que sea una señal sorda

Este modelo tiene una función de transferencia de un filtro IIR por lo tanto la salida depende de la entrada actual y de muestras anteriores de esta manera se hace una predicción lineal de la salida.

Para el cálculo de los coeficientes del filtro podemos utilizar el método de correlación

$$R_S(|K|) = \sum_{K=1}^{P} a_i R(|K-i|)$$

$$R_s(|1|) = a_1 R_s(|0|) + a_2 R_s(|1|) + ... + a_P R_s(|P-1|)$$
$$R_s(|2|) = a_1 R_s(|1|) + a_2 R_s(|2|) + ... + a_P R_s(|P-2|)$$
$$\cdots\cdots \qquad \cdots\cdots$$
$$R_s(|P|) = a_1 R_s(|P-1|) + \qquad ..... + a_P R_s(|0|)$$

Y expresado en forma Matricial quedaría:

$$\begin{vmatrix} R_s(|1|) \\ R_s(|2|) \\ \vdots \\ R_s(|P|) \end{vmatrix} = \begin{vmatrix} R_s(|0|) & R_s(|1|) & \cdots & R_s(|P-1|) \\ R_s(|1|) & R_s(|0|) & \cdots & R_s(|P-2|) \\ \vdots & \vdots & \vdots & \vdots \\ R_s(|P-1|) & \cdots & \cdots & R_s(|0|) \end{vmatrix} \begin{vmatrix} a_1 \\ \vdots \\ \vdots \\ a_p \end{vmatrix}$$

El objetivo de obtener un modelo del aparato fonador en el reconocimiento del habla es adquirir la envolvente del espectro de la señal de voz, es decir si aplicásemos la transformada localizada de Fourier en vez de este sistema observaríamos que la componente frecuencial coincide con la respuesta en frecuencia del modelo con la característica que dicha respuesta tiene una forma más "suave" que la obtenida con la TDF. Los picos de la respuesta en frecuencia corresponden a los Formantes de la señal, son estos Formantes los que posteriormente se utilizaran para clasificar los sonidos.

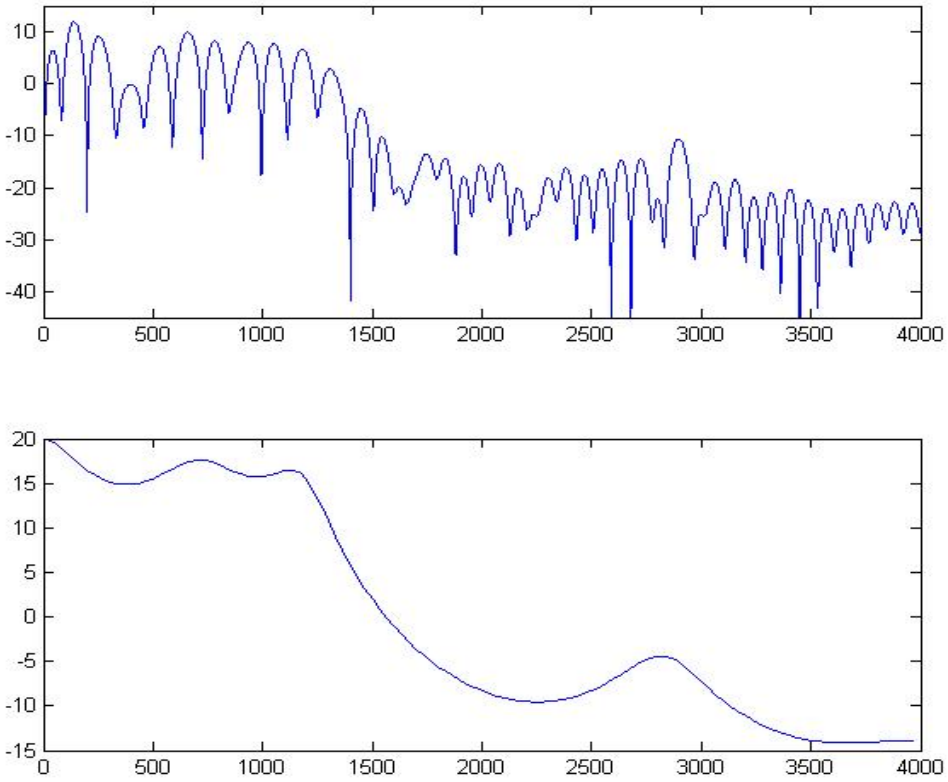


Fig. 3.11 Arriba se puede ver la transformada localizada de fourier de la letra /a/ mientras que abajo se puede ver la respuesta en frecuencia del modelo utilizando coeficientes LPC. Como se puede observar los coeficientes LPC nos proporcionan la envolvente de la señal.

3.2.2.4 Análisis Cepstral

Al considerar que el modelo del tracto vocal se comporta como un sistema lineal hemos de suponer que la salida es el resultado del producto de convolución con la función de transferencia del sistema.

El Análisis Cepstral tiene el objetivo de desconvolucionar la salida para separar la entrada de la función de transferencia del sistema para ello tiene en cuenta la siguiente observación:

$$s(t) = e(t) * h(t)$$

Calculamos la transformada de fourier

$$S(\omega) = E(\omega) \cdot H(\omega)$$

A continuación aplicamos las propiedades de los logaritmos y obtenemos:

$$\log|S(\omega)| = \log|E(\omega)| + \log|H(\omega)|$$

Antitransformamos

$$c[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} \log|S(\omega)| d\omega$$

$$c[n] = c_{excitación}[n] + c_{tracto}[n]$$

Si conseguimos eliminar la excitación y transformamos de nuevo regresando al dominio frecuencial obtendremos un al igual que con los coeficientes LPC un espectro suavizado del la señal. Este tipo de filtrado se denomina filtrado Homomorfico.

3.2.2.5 Coeficientes MEL

Las técnicas comentadas hasta ahora enfatizan todas las escalas de frecuencia por igual a menudo es conveniente utilizar una escala de frecuencias no lineal. La escala que se utiliza normalmente se denomina escala mel esta escala imita el proceso auditivo siendo lineal hasta los 1000 Hz y logarítmica a partir de entonces.
En general se utiliza un banco de filtros triangulares esto no solo se logra aproximar la escala MEL sino reducir la cantidad de información a procesar en etapas posteriores.

3.3 Técnicas de reconocimiento

3.3.1 Cuantificación Vectorial

3.3.1.1 ¿Que es la cuantificación vectorial?

La cuantificación, a nivel general, es una técnica de compresión de datos que tiene como objeto el reducir el espacio ocupado por una serie de parámetros, pero tratando de
mantener la fidelidad requerida de los datos.

En los sistemas de reconocimiento de habla cuantificamos un vector de parámetros, a diferencia de la cuantificación escalar, en la que se cuantifica un sólo valor .En el caso de utilizar modelos ocultos de Markov discretos, el módulo de cuantificación vectorial realiza la correspondencia

entre los parámetros cepstrales y los símbolos pertenecientes a la técnica de modelado de Markov.

3.3.1.2 Codebook

Se realiza mediante un proceso de aprendizaje a partir de un conjunto de vectores de entrenamiento. Estos vectores serán una muestra del tipo de vectores que luego querremos cuantificar por ejemplo los coeficientes cepstrum. Nuestro objetivo será encontrar los L vectores que mejor representen a estos vectores de entrenamiento (de modo que la distorsión media total sea mínima). Para llegar a tal objetivo una solución muy buena seria utilizar el algoritmo LBG.

El algoritmo LBG, lleva su nombre debido a sus autores Y. Linde, A. Buzo y R. M. Gray, en él se elige 1 codeword inicial de entre los vectores de datos a clasificar, luego se utiliza el algoritmo de división binaria para duplicar el número de codewords, los vectores de observación se agrupan en torno a los codewords que les presentan menor
distancia, se recalculan los codewords como la media multidimensional de cada sector y se agrupan nuevamente los datos, el proceso se detiene cuando el codebook no presenta variación significativa y al llegar al número de codewords deseados.
Este algoritmo de gran popularidad (que utiliza el algoritmo k-Means) produce codebooks que logran un mínimo local en la función de error por distorsión.
Para generar un codebook de M sectores o palabras de código:
En primer lugar designando un codeword inicial para luego utilizando una técnica de división llegar a obtener un codebook inicial, luego iterando la misma técnica de división en los codewords hasta que llegamos a obtener el número de codewords igual a M que va a ser el tamaño del codebook deseado.

3.3.2 Alineamiento temporal

Uno de los problemas en el reconocimiento de voz es que una palabra pronunciada repetidamente no se pronuncia nunca igual ya que la duración en el tiempo de cada uno de los fonemas varía en cada repetición y por lo tanto se hace difícil una comparación de patrones de cada palabra contenida en un diccionario.

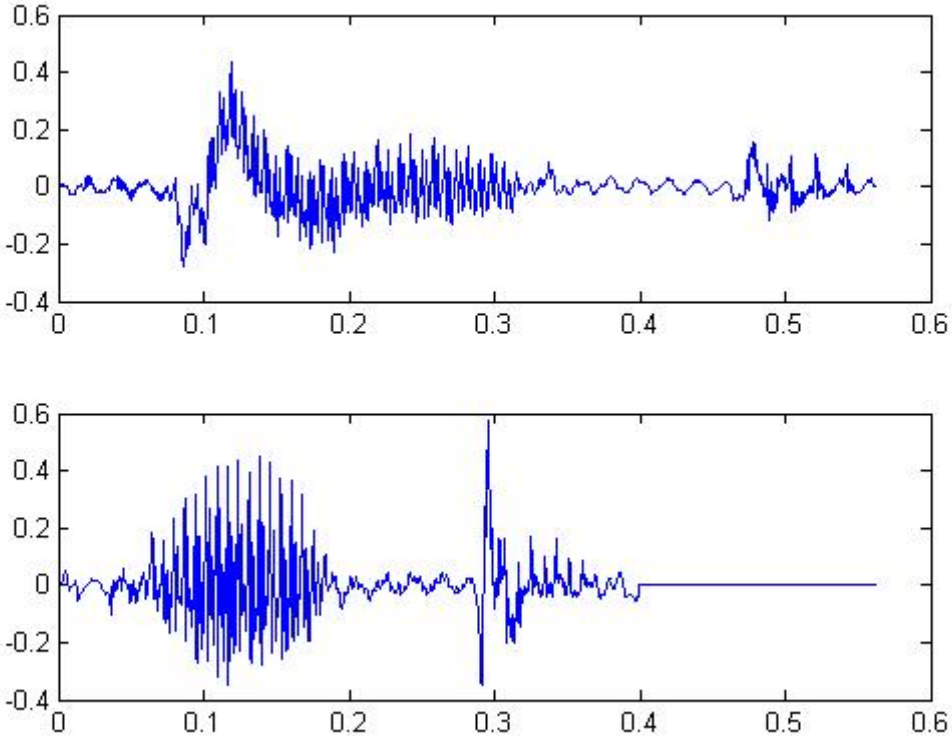


Fig. 3.12 Oscilograma de la palabra "golpe". Como podemos ver en la imagen donde la misma palabra es pronunciada dos veces, está claramente desalineada

Una de las técnicas utilizadas en reconocimiento del habla que resuelve este problema es DTW (Alineamiento Dinámico en el Tiempo). Esta técnica que fue una de las primeras en conseguir un porcentaje de acierto relativamente elevado consiste en alinear dos señales que previamente habrán sido divididas en cuadros o Frames en ingles posteriormente de cada uno de estos cuadros se extraerán los coeficientes cepstrales ya que en la practica no resulta efectivo la comparación directamente del espectro de la señal la alineación temporal debe llevarse acabo por medio de una distorsión temporal. Para entender el significado de este concepto pondré un ejemplo si nosotros deseamos comparar dos señales a (t) y b (t) que habremos dividido en na y nb cuadros y de cada cuadro habremos obtenido los coeficientes cepstral estos coeficientes formaran dos vectores Ai correspondiente al i-ésimo vector y un segundo vector Bj correspondiente al j-ésimo vector una primera solución podría ser una función tal que:

$$j = \phi(i)$$

Esta solución no resuelve todas las situaciones ya que en ocasiones un fonema del vector A puede ocupar más de un cuadro del vector B. Se hace necesario la utilización de un índice k=1,..T correspondiente a un tiempo normalizado y hacer j y i dependan de k:

$$j = \phi_a(k)$$
$$i = \phi_b(k)$$

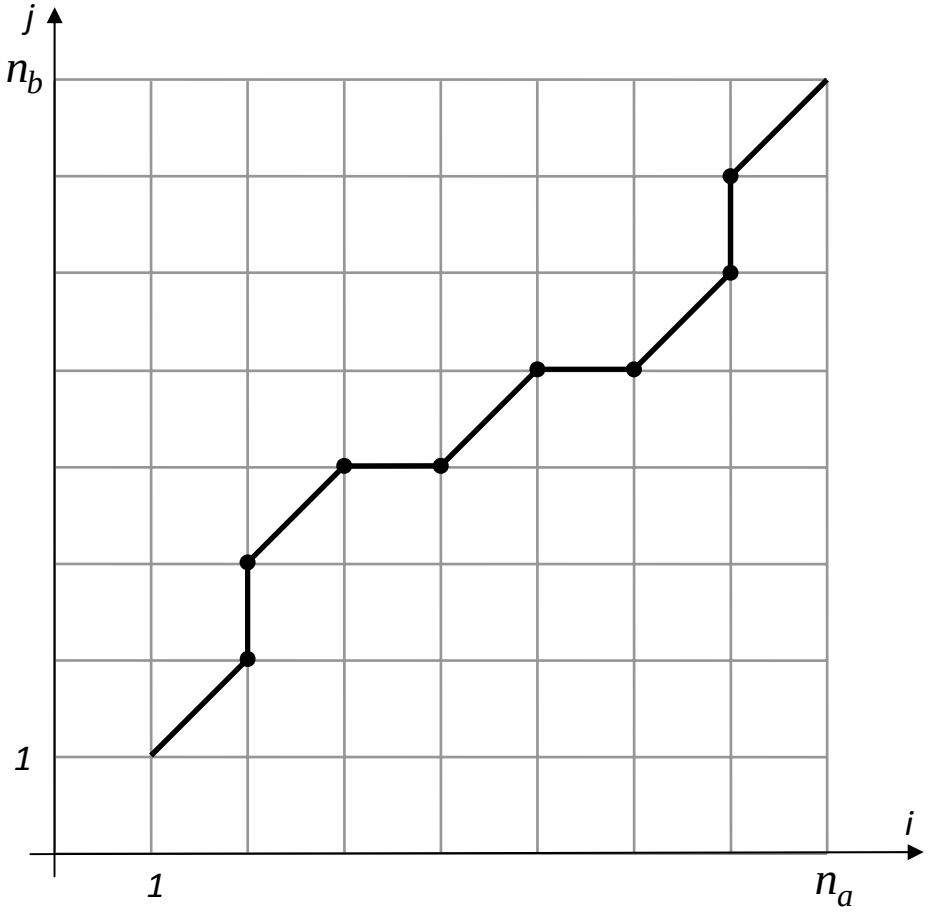


Figura 2.3. Carta de formantes

Nuestro objetivo es alinear dos vectores de coeficientes cepstrum iguales a partir de dicha alineación podemos definir una distancia igual a la suma de los cuadros iguales. Es decir,

$$d_\phi(A,B) = \sum_{k=1}^{T} d(a_{\phi_{a(k)}}, b_{\phi_{b(k)}})$$

Hemos introducido el concepto de la función de distorsión temporal así como la distancia entre A y B de cada una ellas pero existe un numero muy elevado de ellas y se hace necesario un criterio para la elección de la optima que será aquella que hace mínima la distancia entre A y B.

$$\phi_{optima} = \arg\left(\min\left\{d_\phi(A,B)\right\}\right)$$

No profundizaremos más este método ya que no se ha utilizado en el presente proyecto ya que los métodos empleados para el reconocimiento del habla son HMM y redes neuronales artificiales.

3.3.3 Modelos ocultos de Markov

Uno de los métodos más útiles empleados para aplicaciones de reconocimiento del habla son los HMM(Modelos Ocultos de Markov) en inglés Hidden Markov Model este método a diferencia de otros descritos anteriormente tiene un enfoque puramente estadístico. Una de las ventajas frente a las redes neuronales es que nos se hace necesaria la segmentación de las palabras ya que la utilización de estos modelos realiza una segmentación en el mismo proceso de búsqueda de la secuencia más probable.

3.3.3.1 Procesos de Markov

Consideremos un sistema que en un instante determinado se encuentra dentro de un estado de un conjunto de N estados distintos: S1,S2,…,SN consideremos también la probabilidad de transición de un estado a otro en intervalos de tiempo espaciados regularmente.

Denotamos:

Instantes de tiempo asociados con los cambios de estados: t = 1, 2,…
Estado en el tiempo $t : q_t$

Una descripción probabilística completa del sistema requeriría especificar el estado actual (en el tiempo t) y todos los estados precedentes.
Una secuencia de estados S1,S2,… es una Cadena de Harkov, si posee la propiedad de Markov:

$$P(q_t = S_j \mid q_{t-1} = S_i, q_{t-2} = S_k, ...) = P(q_t = S_j \mid q_{t-1} = S_i)$$

De esta manera podemos definir un conjunto de probabilidades de transición de estados a_{ij} de la forma:

$$a_{ij} = P(q_t = S_j \mid q_{t-1} = S_i)$$

$$1 \le i, j \le N$$

La salida del proceso es el conjunto de estados en cada instante de tiempo, donde cada estado corresponde a un evento observable. El proceso de Harkov se denomina entonces observable.

3.3.3.2 Ejemplo de un modelo con dos estados observables

Supongamos que una habitación hay una cortina y en un lado de la cortina estamos situados nosotros y al otro lado hay una persona que lanza una moneda al aire y que únicamente nos comunica el resultado del experimento, es decir si la moneda a caído del lado Cara o por el contrario a caído del lado Cruz.
En este ejemplo existen únicamente dos estados que corresponden con cada lado de la moneda es decir es un modelo de Harkov con dos estados observables.

O = C X X C X C ·····
S = 1 2 2 1 2 1 ·····

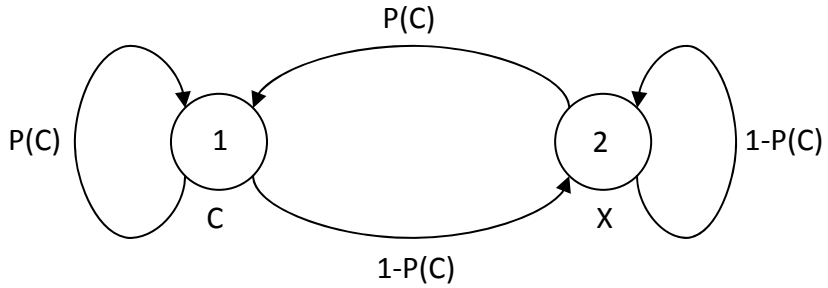


Fig. 3.13 Modelo de Markov con dos estados observables, donde se lanza una moneda al aire la C representa cara y la X cruz. En este caso la secuencia de estados coincide con la secuencia de observación

3.3.3.3 Ejemplo de un modelo con 2 estados ocultos

Pensemos en el ejemplo anterior donde alguien detrás de una cortina nos comunica el resultado del lanzar una moneda al aire pero en este caso esta persona realiza el experimento con dos monedas y únicamente nos comunica si la moneda a caído del lado de Cara o del de Cruz sin proporcionarnos información sobre cual es la moneda que a lanzado al aire que por otra parte habrá sido escogida aleatoriamente. En este caso se trata de un HMM ya que es un proceso doblemente estocástico, donde hay un proceso que no es observable (es decir, es oculto), y solo puede ser observado a través de otro conjunto de procesos estocásticos que producen la secuencia de observación.

O = C X X C X C ·····
S = 2 1 1 2 1 2 ·····

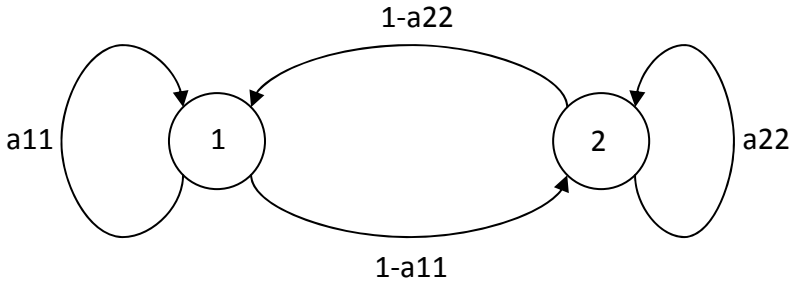


Fig. 3.14 Modelo de Markov con dos estados ocultos, donde se lanza dos moneda al aire la C representa cara y la X cruz. En este caso la secuencia de estados no coincide con la secuencia de observación

3.3.3.4 Elementos que componen un HMM

Conjunto de N estados

$$S = \{S_1, S_2, ..., S_N\}$$

Conjunto M de símbolos observables que puedan ser producidos por un HMM

$$V = \{V_1, V_2, ..., V_M\}$$

3) una matriz de probabilidades de transición de estados, $A = \{a_{ij}\}$. Esta matriz es cuadrada de dimensión N y cada elemento a_{ij} corresponde a la probabilidad de transición del estado S_i al estado S_j.

$$a_{ij} = P(q_t = S_j \mid q_{t-1} = S_i)$$

$$1 \le i, j \le N$$

4) Distribución de probabilidad del símbolo de observación en el estado 'j'

$$B = \{b_j(k)\}$$

donde

$$b_j(k) = P(V_k \ en \ t | q_t = S_j), \quad 1 \le j \le N, \quad 1 \le k \le M$$

Un conjunto de probabilidades del estado inicial $\pi = \{\pi_i\}$

donde

$$\pi_i = P(q_1 = S_i), \quad 1 \le i \le N$$

De esta manera quedan completamente especificadas las tres medidas de probabilidad A, B, y π que definen un HMM que generalmente se denota.

$$\lambda = (A, B, \pi)$$

3.3.3.5 Los tres problemas de un HMM

La utilización de HMM requiere la solución de tres problemas:

Problema 1: Teniendo una secuencia observación $O = O_1, O_2 \cdots O_T$ y un modelo $\lambda = (A, B, \pi)$ como calcular la probabilidad de que dicho modelo haya generado dicha secuencia de observación.

Problema 2: Teniendo una secuencia de observación $O = O_1, O_2 \cdots O_T$ y un modelo $\lambda = (A, B, \pi)$ como escogemos una secuencia de estados $Q = q_1 \, q_2 \cdots q_T$ que mejor explique dicha secuencia de observación.

Problema 3: Teniendo una secuencia de observación $O = O_1, O_2 \cdots O_T$ como elegir los parámetros del modelo $\lambda = (A, B, \pi)$ para que la generación de dicha secuencia por el modelo sea la más probable.

Supongamos un diagrama de estados como el de la figura 5 con un parámetro $B = \{b_j(k)\}$ que describe la probabilidad de que el estado j observe el símbolo k del conjunto de salidas. En estas condiciones nunca se podrá saber con certeza en que estado esta el modelo observando solamente su salida. Se trata de un HMM este tipo de modelo es uno de los más utilizados en el reconocimiento de voz y poseen una estructura muy simple llamada de derecha a izquierda de esta manera es mas fácil el calculo de sus parámetros debido a esta restricción.

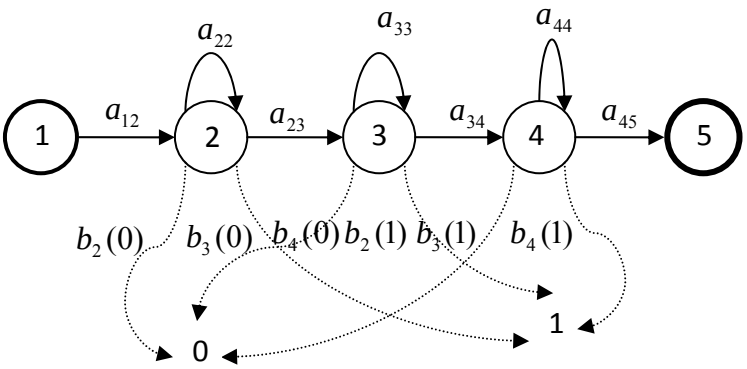


Fig. 3.14 Diagrama de estados de un modelo de Harkov, este modelo se utiliza comúnmente en el reconocimiento de voz. las líneas en flechas continuas indican las transiciones entre estados, las líneas discontinuas indican la probabilidad de observación en cada estado. En esta configuración se puede observar la particularidad de que las transiciones se dan solamente de derecha a izquierda.

Ahora supongamos que para el modelo de la figura 5 tenemos los siguientes parámetros:

$$A = \begin{vmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1/4 & 1/4 & 1/2 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 0 & 0 \end{vmatrix} \qquad B = \begin{vmatrix} 0 & 1/3 & 1/5 & 2/3 & 0 \\ 0 & 2/3 & 4/5 & 1/3 & 0 \end{vmatrix}$$

Teniendo en cuenta todo esto podemos platear una pregunta ¿que probabilidad existe de que este modelo genere la secuencia 0, 0, 1, 0? La Respuesta no es tan obvia ya que se trata de un modelo oculto, para resolver este problema es necesario analizar todas las posibles secuencias que pasen por 4 estados emisores y sus probabilidades asociadas la respuesta a esta pregunta seria la solución los problemas 1 y 2 de los HMM.

Secuencia de estados	Probabilidades de transición	Probabilidades de observación	Probabilidades de la secuencia
1, 2, 2, 2, 4, 5	$1\frac{1}{4}\frac{1}{4}\frac{1}{2}\frac{1}{2}=\frac{1}{64}$	$\frac{1}{3}\frac{1}{3}\frac{2}{3}\frac{2}{3}=\frac{4}{81}$	$\frac{1}{1296}$
1, 2, 2, 3, 4, 5	$1\frac{1}{4}\frac{1}{4}\frac{1}{2}\frac{1}{2}=\frac{1}{64}$	$\frac{1}{3}\frac{1}{3}\frac{4}{5}\frac{2}{3}=\frac{8}{135}$	$\frac{1}{1080}$
1, 2, 2, 4, 4, 5	$1\frac{1}{2}\frac{1}{4}\frac{1}{2}\frac{1}{2}=\frac{1}{32}$	$\frac{1}{3}\frac{1}{3}\frac{1}{3}\frac{2}{3}=\frac{2}{81}$	$\frac{1}{1296}$
1, 2, 2, 3, 4, 5	$1\frac{1}{2}\frac{1}{4}\frac{1}{2}\frac{1}{2}=\frac{1}{32}$	$\frac{1}{3}\frac{1}{5}\frac{4}{5}\frac{2}{3}=\frac{8}{225}$	$\frac{1}{900}$
1, 2, 3, 4, 4, 5	$1\frac{1}{2}\frac{1}{4}\frac{1}{2}\frac{1}{2}=\frac{1}{32}$	$\frac{1}{3}\frac{1}{5}\frac{1}{3}\frac{2}{3}=\frac{2}{135}$	$\frac{1}{2160}$
1, 4, 4, 4, 4, 5	$1\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}=\frac{1}{16}$	$\frac{1}{3}\frac{2}{3}\frac{1}{3}\frac{2}{3}=\frac{4}{81}$	$\frac{1}{324}$

Probabilidad total $\sum=\frac{77}{10800}$

Tabla 1. En esta tabla podemos ver todos los caminos para una secuencia de 4 observaciones del ejemplo

Este calculo directo resulta inviable así pues se hace necesario reducir el numero de cálculos empleados para encontrar la secuencia más probable. Con este fin existen algoritmos como el algoritmo de Viterbi que permiten ahorrar muchos cálculos. En este algoritmo la idea principal es recorrer el diagrama de transiciones de estados a través del tiempo, almacenando para cada estado solo ala máxima probabilidad acumulada y el estado anterior desde el que se llega.

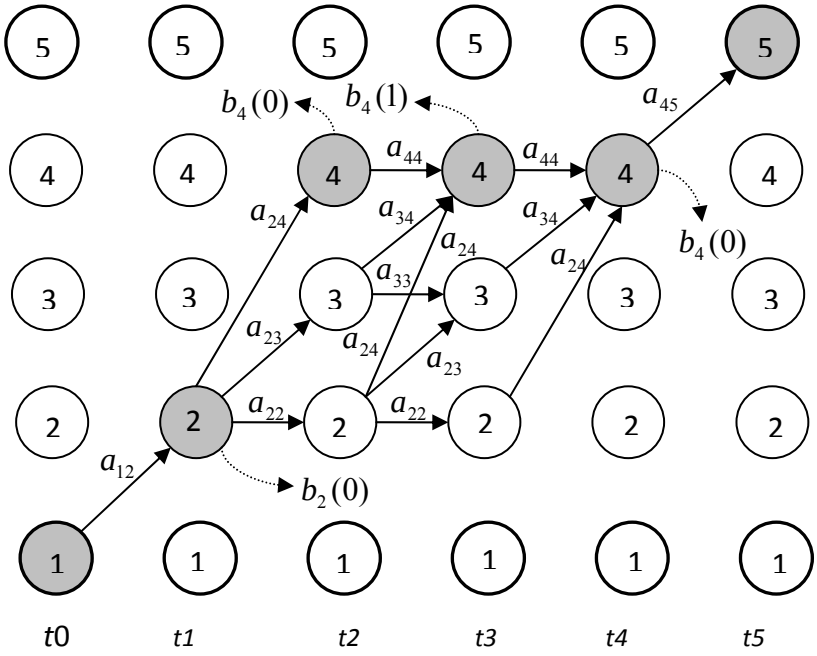


Fig. 3.15. Diagrama de transiciones de estado. En este diagrama podemos ver todos los caminos posibles y el más probable encontrado utilizando el algoritmo de viterbi

Teniendo en cuenta el diagrama de la figura 3.15 calcularemos la secuencia de estados utilizando el algoritmo de viterbi para una secuencia para la misma secuencia de observación del ejemplo anterior es decir 0, 0, 1, 0.

Comenzamos en el estado 1 cuya probabilidad acumulada es P1 = 1 y al pasar al estado 2 la probabilidad acumulada es:

$$p_{12} = b_2(0)[p_1 a_{12}] = \frac{1}{3}[1 \times 1] = \frac{1}{3}$$

Desde el estado 2 se puede pasar al estado 2, 3 o 4:

$$p_{122} = b_2(0)[p_{12} a_{22}] = \frac{1}{3}[\frac{1}{3}\frac{1}{4}] = \frac{1}{36}$$

$$p_{122} = b_2(0)[p_{12} a_{22}] = \frac{1}{3}[\frac{1}{3}\frac{1}{4}] = \frac{1}{60}$$

$$p_{124} = b_4(0)[p_{12} a_{24}] = \frac{2}{3}[\frac{1}{3}\frac{1}{2}] = \frac{1}{9}$$

Desde el estado 2 en el tiempo t2 se puede pasar a los estados 2, 3, 4

$$p_{1222} = b_2(1)[p_{212}a_{22}] = \frac{2}{3}[\frac{1}{36}\frac{1}{4}] = \frac{1}{216}$$

$$p_{1223} = b_3(1)[p_{112}a_{22}] = \frac{4}{5}[\frac{1}{36}\frac{1}{4}] = \frac{1}{180}$$

$$p_{1222} = b_2(1)[p_{212}a_{22}] = \frac{2}{3}[\frac{1}{36}\frac{1}{4}] = \frac{1}{216}$$

Desde el estado 3 en tiempo t2 se puede pasar a los estados 3 y 4

$$p_{1233} = b_3(1)[p_{123}a_{33}] = \frac{4}{5}[\frac{1}{60}\frac{1}{2}] = \frac{1}{600}$$

$$p_{1234} = b_4(1)[p_{123}a_{34}] = \frac{1}{3}[\frac{1}{60}\frac{1}{2}] = \frac{1}{360}$$

Y desde el estado 4 en el tiempo t2 solo se puede pasar al estado 4:

$$p_{1224} = b_4(1)[p_{124}a_{44}] = \frac{1}{3}[\frac{1}{9}\frac{1}{2}] = \frac{1}{54}$$

Una vez llegado al tiempo t3 desde cualquier estado solo es posible pasar al estado 4:

$$p_{1222} = b_4(0)[p_{1222}a_{24}] = \frac{1}{3}[\frac{1}{216}\frac{1}{2}] = \frac{1}{1296}$$

$$
\begin{aligned}
p_{12?34} &= b_4(0)\max\{[p_{1223}a_{34}][p_{1233}a_{34}]\} \\
&= b_4(0)\max\{p_{1223}, p_{1233}\}a_{34} \\
&= \frac{1}{5}\max\left\{\frac{1}{216}, \frac{1}{600}\right\}\frac{1}{2} \\
&= \frac{1}{5}\frac{1}{216}\frac{1}{2} = \frac{1}{2160} \\
&= p_{12434}
\end{aligned}
$$

$$
\begin{aligned}
p_{12?44} &= b_4(0)\max\{[p_{1224}a_{44}][p_{1234}a_{44}][p_{1244}a_{44}]\} \\
&= b_4(0)\max\{p_{1224}, p_{1234}, p_{1244}\}a_{44} \\
&= \frac{2}{3}\max\left\{\frac{1}{216}, \frac{1}{360}, \frac{1}{54}\right\}\frac{1}{2} \\
&= \frac{1}{5}\frac{1}{54}\frac{1}{2} = \frac{1}{160} \\
&= p_{12444}
\end{aligned}
$$

Finalmente en el tiempo t4 la única opción es pasar al estado 5.

$$
\begin{aligned}
p_{12??45} &= \max\{[p_{12224}a_{45}][p_{12234}a_{45}][p_{12444}a_{45}]\} \\
&= \max\{p_{12224}, p_{12234}, p_{1244}4\}a_{45} \\
&= \max\left\{\frac{1}{1296}, \frac{1}{2160}, \frac{1}{162}\right\}\frac{1}{2} \\
&= \frac{1}{162}\frac{1}{2} = \frac{1}{324} \\
&= p_{124445}
\end{aligned}
$$

De esta manera llegamos a la misma conclusión que en la tabla 1 es decir el camino más probable es 1, 2, 4, 4, 4, 5 y todo esto realizando un numero mucho menor de cálculos además hay que tener en cuenta que esta diferencia de cálculos aumenta a mediada que se incrementa el numero de estados del modelo.

3.3.3.6 Determinar los parámetros de un modelo

Hasta ahora hemos calculado la secuencia de estados a partir de un modelo cuyos parámetros eran conocidos pero queda pendiente uno de los problemas más importantes, estimar el valor de dichos parámetros, a este proceso se le denomina entrenamiento.

Si a través del algoritmo de Viterbi obtenemos la secuencia de estados a partir de un conjunto símbolos de observación es posible estimar las probabilidades de transición y observación a partir de los símbolos que han quedado asignados a cada estado. Si se tiene un conjunto de secuencias de observación para el entrenamiento se pueden encontrar todas las secuencias de estados más probables y contabilizar las veces que se ha pasado del estado j a partir del estado i. De esta manera se obtiene una buena aproximación de las probabilidades de transición entre estados aij.
De forma similar se puede encontrar el parámetro bj(k) contando las veces que el k-ésimo símbolo observable ha sido asignado al j-ésimo estado del modelo.

3.3.4 Redes Neuronales artificiales

3.3.4.1 Neuronas biológicas

Las neuronas son células especializadas en tratar la información. Estas células están interconectadas masivamente para hacernos una idea cada una esta conectada con aproximadamente entre 1000 y 10000 neuronas, cada de estas neurona recibe información en forma de un impulso eléctrico a través de una ramificaciones denominadas dentritas y envía las señales creadas en la célula a través de otro tipo de ramificación denominada Axón. Las neuronas tratan con la información a una velocidad del orden de los 100 Hz una velocidad muy inferior a los circuitos electrónicos actuales pero en cambio procesan tal cantidad de información en paralelo que es impensable su implementación con un circuito electrónico.

3.3.4.2 Redes neuronales artificiales

Las redes neuronales artificiales intentan emular el comportamiento cerebral. Una red neuronal artificial consiste en un conjunto de unidades computacionales simples denominadas nodos estos nodos están conectados a otras unidades simples utilizando unas conexiones denominadas pesos estas conexiones se refuerzan mediante un procedimiento de aprendizaje. Al igual que las redes neuronales artificiales la RNA cada nodo cambia de estado en función del valor de las entradas.

3.3.4.3 Modelo de neurona

Los nodos son los encargados de determinar el comportamiento de la red de manera que cada uno tiene una única salida que se activará si la suma de sus entradas xi multiplicada previamente por su correspondiente peso Wij supera un cierto umbral este umbral es determinado por una función f(u).

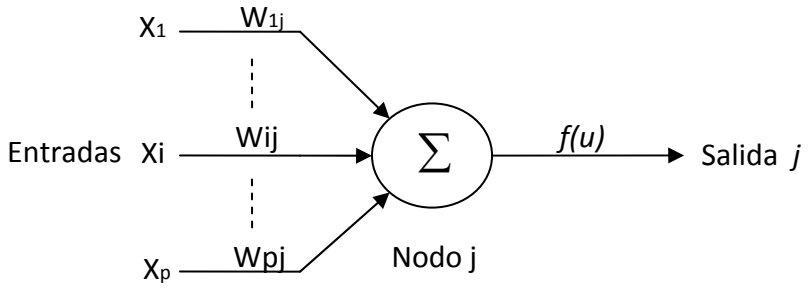


Fig 3.16. Modelo de Neurona Artificial

	Función salida
Identidad	$y = x$
Senoide	$y = Asen(\alpha x + \vartheta)$
Escalón	$y = signo(x)$ $y = H(x)$
Lineal por tramos	$y = \begin{cases} -1 & si\ x \le -1 \\ x & si + 1 \le x \le -1 \\ 1 & si\ z \ge 1 \end{cases}$
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$
Gaussiana	$y = Ae^{-Bx^2}$

Tabla 2. Tipos de funciones de transferencia que se pueden encontrar en algunos modelos neuronales.

3.3.4.4 Arquitectura de la red

La arquitectura de las redes neuronales artificiales depende del algoritmo de entrenamiento escogido dicho esto podemos clasificar según el tipo de neuronas:

Neuronas de entrada (reciben las señales del entorno)
Neuronas ocultas (sus entradas y salidas están dentro de la RNA y no son visibles desde el exterior.
Neuronas de salida (envían la señal fuera de la red neuronal)

Además pueden organizarse en niveles o capas Un nivel de neuronas es un conjunto de neuronas cuyas entradas provienen de la misma fuente y cuyas salidas se dirigen al mismo destino.

También es posible clasificarlas según su tipo de conexión:

Conexión hacia delante (la salida de una neurona solo puede ser la entrada de una neurona de nivel superior
Conexión hacia atrás (la salida puede ser la entrada del mismo nivel, posterior o anterior)

3.3.4.5 Aprendizaje

Una definición de aprendizaje aplicado a las RNA es la realizada por Haykin (1994):

Aprendizaje es el proceso mediante el cual los parámetros libres de la red neuronal son adaptados a medida que esta es estimulada por el entorno en que la red neuronal se encuentra inmersa.

Es decir como parámetros libres tenemos los pesos Wij y para el ajuste de dichos pesos es necesario un algoritmo de entrenamiento, existen dos tipos de entrenamiento:

3.3.4.5.1 Entrenamiento supervisado

En el entrenamiento supervisado el reajuste de los pesos se realiza a partir de la comparación de la salida obtenida con la salida esperada, es decir la red minimiza el error.

3.3.4.5.2 Entrenamiento no supervisado

Es un aprendizaje autoorganizado sin necesidad de un agente externo que supervise el desempeño de la RNA

4. Implementación en Matlab

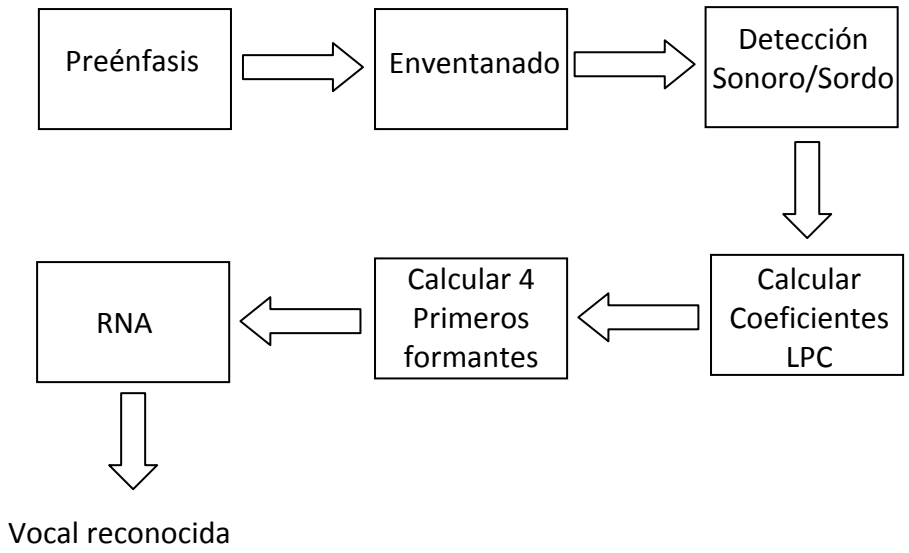
4.1 Introducción

La herramienta escogida para la implementación del presente proyecto es MATLAB. Matlab es un lenguaje de programación de alto nivel basado en matrices y vectores que gracias a su gran potencia de cálculo y a su entorno agradable que incorpora la posibilidad de visualización gráfica de los resultados hacen que sea la herramienta ideal para el procesamiento de señales.

Matlab puede incorporar una gran variedad de programas denominados toolboxes que extienden la cantidad de funciones contenidas en el programa principal. Una de las toolboxes que resulta muy útil para la implementación de este proyecto es Signal Processing Toolbox.
Matlab a su vez dispone también de una toolbox de adquisición de datos que nos permitirá adquirir datos a través entrada de micrófono de la tarjeta de sonido del PC y a su vez también nos permitirá sacar datos a través de la salida de altavoces.

4.2 Implementación de un modulo de reconocimiento de vocales

Uno de los objetivos del presente proyecto es el de implementar un sistema de reconocimiento del habla robusto. Para llegar a este propósito primero se ha llevado a cabo la implementación de un modulo de reconocimiento de vocales utilizando para ello redes neuronales artificiales y como parámetros de entrada a la red se han utilizado los 4 primeros formantes a partir del calculo de los coeficientes LPC a continuación se describe el algoritmo utilizado.



4.2.1 Filtro Preenfasis

Debido a que la señal de voz sea atenúa a medida que aumenta la frecuencia es necesario incrementar la relevancia de las frecuencias altas esto se consigue mediante un filtro denominado filtro de preenfasis cuyo coeficiente es normalmente $\alpha = 0.95$.

4.2.2 Enventanado

Para el enventanado se ha utilizado una ventana hamming con una longitud de 320 muestras y un desplazamiento de la de 80 muestras, la frecuencia de muestreo utilizada para la adquisición de las muestras es de 8000 Hz esta es la frecuencia mínima a la que se puede muestrear una señal de voz ya que una frecuencia menor produciría el efecto aliasing, teniendo en cuenta esta frecuencia podemos calcular la longitud en tiempo de la ventana así como su desplazamiento:

$$T = \frac{1}{Fs} \cdot n^o de\ muestras = \frac{1}{8000} \cdot 320 = 0{,}004s$$

$$T = \frac{1}{8000} \cdot 80 = 0.001s$$

Es decir la longitud de la ventana es de 40 ms y el desplazamiento de la misma es de 10 ms.

4.2.3 Detección sonoras/sordas

Una vez enventanada la señal es necesario determinar donde esta el inicio y el final de la palabra o en este caso el inicio y fin de la vocal a reconocer.
En el caso de reconocimiento de vocales el hecho de que todas las vocales sean sonoras facilita esta clasificación ya que únicamente es necesario clasificar las tramas según su energía y establecer un umbral por encima del cual la señal es sonora. En este caso no es necesaria una detección de bordes ya que separaremos todas las tramas sonoras de las tramas sordas y procesaremos únicamente las primeras porque cuando se pronuncia una vocal las tramas sonoras son consecutivas a diferencia de la pronunciación una palabra donde puede existir un fonema sordo entre dos sonoros.

4.2.4 Calculo de los coeficientes LPC

Una vez clasificadas las tramas entre sonoras y sordas se calcula los coeficientes LPC de las tramas. Para el cálculo de estos coeficientes se ha utilizado una función que incorpora la toolbox de procesamiento de señales de Matlab, esta función utiliza el algoritmo Levinson-Durvin para el cálculo de los coeficientes. Normalmente suele utilizarse un orden de predicción de orden 4 para los sonidos sordos y uno de orden 10 para los sonidos sonoros y es precisamente este el que se ha escogido para la clasificación de vocales.

4.2.5 Calculo de los cuatro primeros formantes

Los cuatro primeros formantes son los que nos aportan más información para realizar un reconocimiento de la vocal como se puede observar en la figura 4.1 los formantes están relacionados con los polos de la respuesta en frecuencia del filtro IIR con coeficientes LPC.

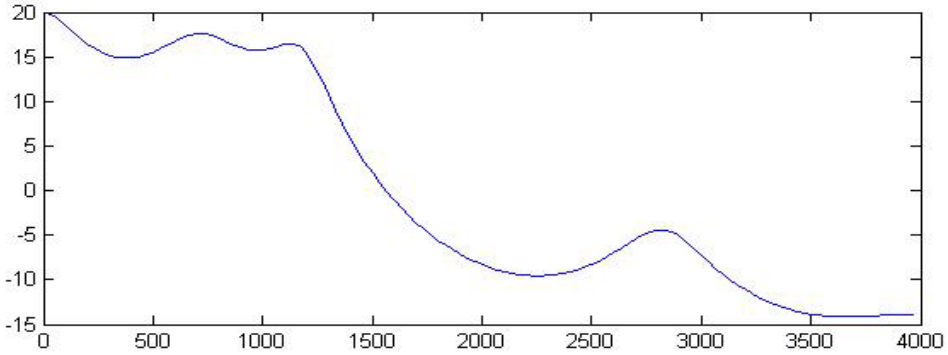


Fig. 4.1 Respuesta en frecuencia del filtro IIR con coeficientes calculados para la vocal /a/.

Para aproximar los formantes se han calculado las raíces para calcular las del la función de transferencia del filtro IIR. Para el calculo de las raíces se ha utilizado la función roots () de Matlab una vez calculadas se separan las que tienen parte imaginaria positiva ya que si la raíz es compleja la función roots () nos devuelve también el conjugado. Una vez se tienen las raíces se calcula el valor absoluto y se orden de mayo a menor, de esta manera tendremos una aproximación de los formantes de la vocal. El calculo de los formantes se calcula sobre cada una de las tramas de las cuales hemos calculado los coeficientes LPC es decir sobre tramas sonoras teniendo en cuenta esto podemos hacer la media de cada uno de los formantes calculados sobre todas las tramas ya que el valor de los formantes debe ser aproximadamente el mismo y de esta manera tendremos un único vector de 3 componentes que será la entrada a la red neuronal.

4.2.6 Red neuronal artificial

Por último para clasificar las vocales una primera idea fue teniendo en cuenta la carta de los formantes ¿podrían separarse linealmente las vocales únicamente teniendo en cuenta el primer y segundo formante?

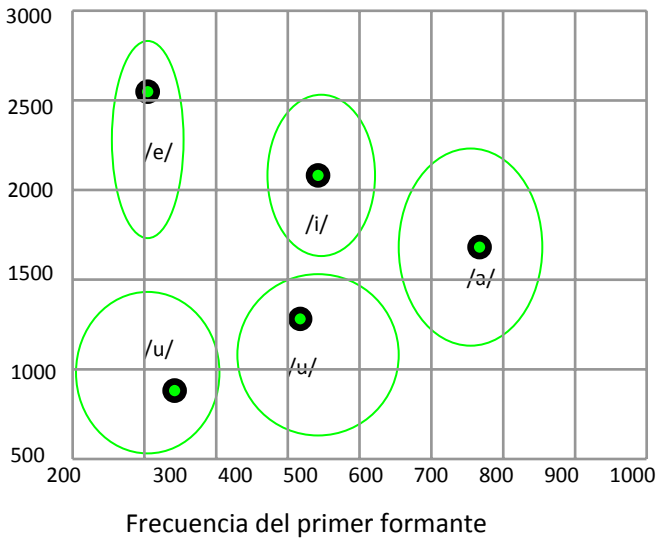


Fig. 4.2 Carta de formantes de las vocales castellanas

Para probarlo se implemento un sencillo algoritmo de clasificación cuyo diagrama de flujo se describe a continuación:

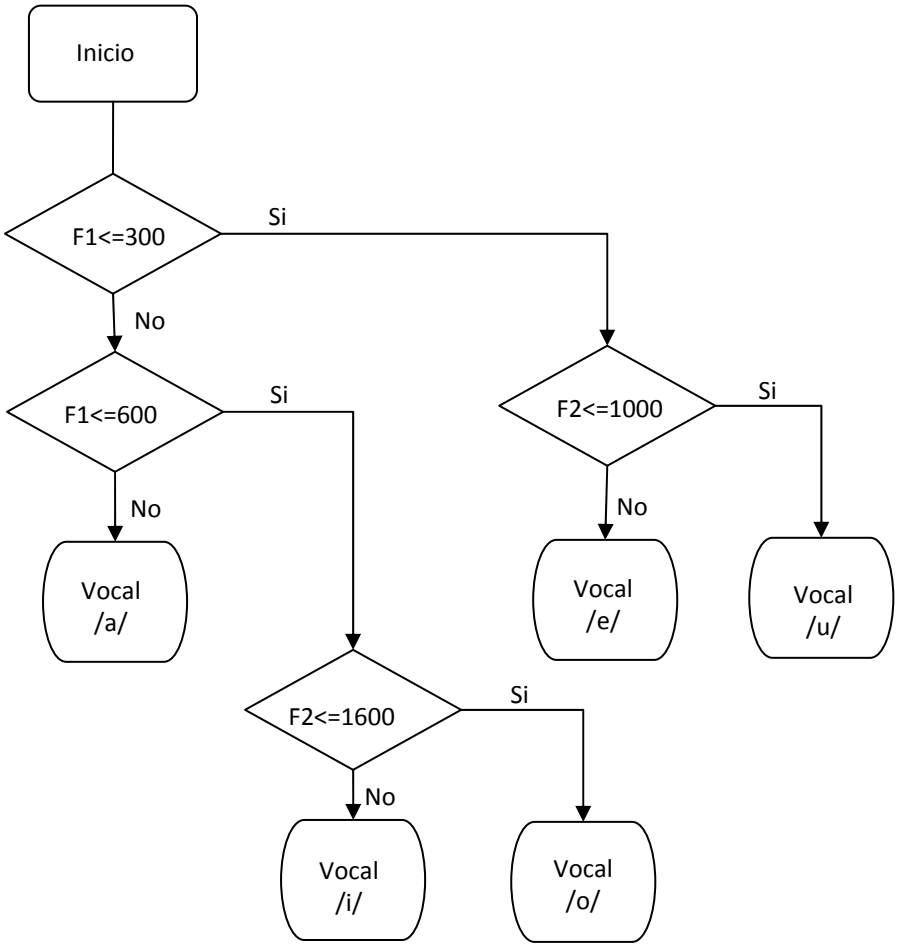


Fig. 4.3 Diagrama de flujo del algoritmo para clasificar vocales a partir de sus dos primeros formantes

Este algoritmo no resulta efectivo ya que aunque teóricamente el valor de los formantes es separable este valor varía lo suficiente como para establecer un límite lineal. Una de las posibles soluciones seria realizar una clasificación utilizando redes neuronales artificiales.

Para la implementación del reconocedor se ha hecho servir una red neuronal utilizando la toolbox de matlab nnet que permite crear una red eligiendo entre las diferentes arquitecturas que existen. La red escogida y que ha demostrado mas efectividad es una red con conexión hacia atrás con tres neuronas de entrada que corresponden a los tres primeros formantes y en la salida cinco neuronas que corresponden a cada una de las vocales para la creación de esta red se utiliza la función newff () a continuación se describen los parámetros de entrada de dicha función:

newff (PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

PR : Rx2 Matriz de valores máximos y mínimos de cada uno de las R neuronas de entrada.
Si: Número de neuronas para cada una de las capas.
TFi: Función de transferencia a utilizar en cada una de las capas, por defecto utiliza tansig
BTF: Algoritmo de entrenamiento a utilizar, por defecto utiliza trainlm
BLF: Función de actualización de los pesos, por defecto utiliza learngdm.
PF: Función para evaluar el desempeño de la red, por defecto utiliza mse.

Para el entrenamiento de la red se utilizo una base de datos que contenía la pronunciación 200 veces de las cinco vocales por diferentes locutores, para el aprendizaje se utilizo el algoritmo que viene por defecto en la función newff de matlab es decir trainlm.
El algoritmo trailm actualiza los pesos y las ganancias de acuerdo a la optimización de Levenberg-Marquardt. Es el algoritmo más rápido para redes Backpropagation; tiene la desventaja de requerir de un set de entrenamiento lo más estándar posible, pues de otra forma solo aproximará correctamente valores que se encuentren dentro de los patrones de aprendizaje.

4.3 Implementación de un modulo de reconocimiento de palabras

4.3.1 Introducción

Una segunda posibilidad para implementar un sistema de reconocimiento del habla es la utilización de los modelos ocultos de Markov de manera que en el presente proyecto se ha llevado a cabo también esta opción es decir reconocer palabras utilizando HMM y en este caso los vectores de características utilizados son los coeficientes cepstrum en escala Mel. No es necesario que los coeficientes estén en esta escala pero si es recomendable ya que al aplicar un filtrado la cantidad de datos a procesar es menor.
El objetivo del programa es escoger entre los modelos de HMM el que tenga más probabilidad de haber generado la secuencia de observación, modelos cuyos parámetros $\lambda = (A, B, \pi)$ habrán sido escogidos previamente durante la fase de entrenamiento. El diagrama de bloques del reconocedor será el siguiente:

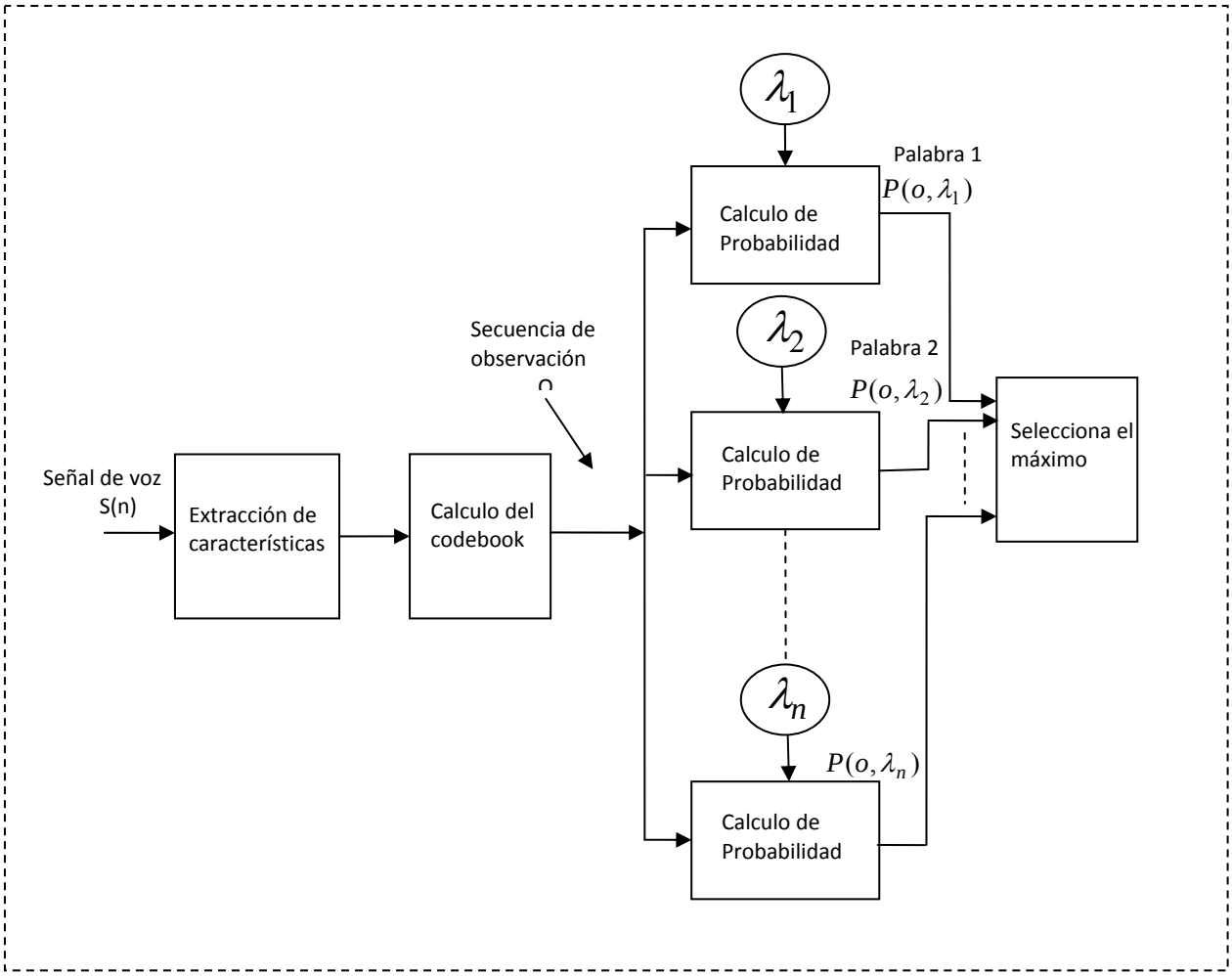


Fig. 4.4. Diagrama de bloques de un modulo de reconocimiento de palabras utilizando modelos ocultos de Harkov.

4.3.2 Extracción de los coeficientes cepstrum.

Para la extracción de los coeficientes MFFC que son necesarios tanto para el reconocimiento de la palabra como para el entrenamiento de los parámetros que componen los modelos HMM se ha utilizado el siguiente algoritmo.

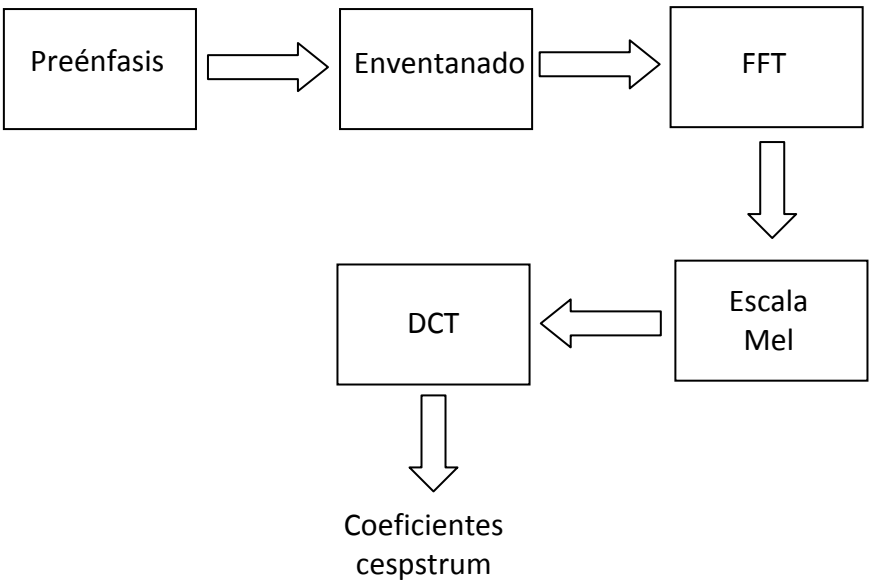


Fig. 4.5. Diagrama de bloques del algoritmo de extracción de los coeficientes cepstrum

4.3.2.1 Preprocesado de la señal

Previamente al igual que sucede con el reconocimiento utilizando redes neuronales artificiales es necesario un preprocesado de la señal es decir un prefiltrado con un filtro de preenfasis con un $\alpha = 0.95$, el enventanado de la señal utilizando una ventana Hamming de 40 ms con un desplazamiento de 10 ms.

4.3.2.2 Aislamiento de la palabra

Hasta ahora el preprocesado de la señal de voz es el mismo que en las RNA, pero a partir de aquí se hace necesario otro sistema para aislar la palabra ya que en el caso RNA únicamente era necesario clasificar las tramas entre sonoras y sordas ya que todas las vocales son sonoras, en este caso en cambio puede existir un sonido que no sea sonoro por ejemplo una consonante como la “s” o la “f”.
Para solucionar este problema se ha utilizado además de la energía de la señal un filtro pasa bajos.

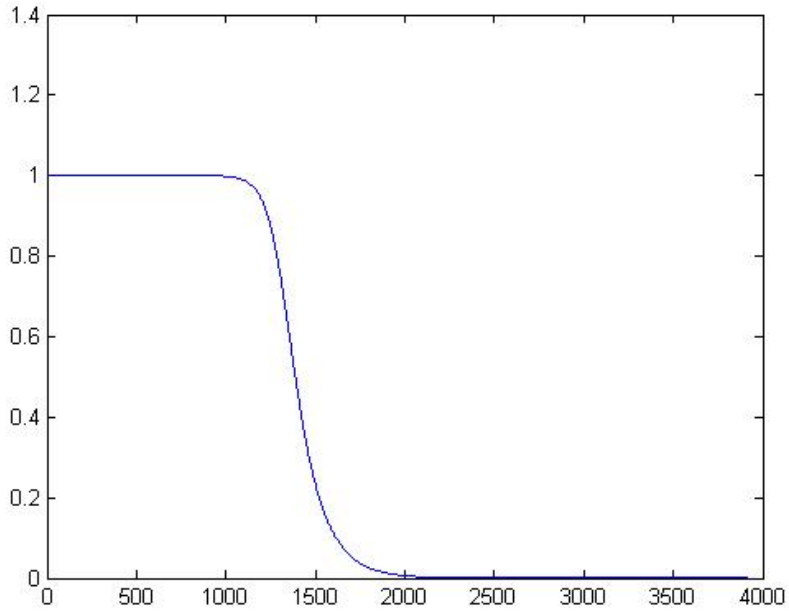


Fig. 4.6. Respuesta en frecuencia del filtro pasabajos utilizado para aislar la palabra

El filtrado se realiza en cada una de las tramas posteriormente se suman todos los valores de la señal filtrada y si esta supera un cierto umbral la trama contiene sonido y por lo tanto se le asigna el valor lógico “1” que caso contrario será de “0”

Una vez aislada la palabra se pasa del dominio temporal al dominio frecuencial utilizando para ello la transformada discreta de fourier. Esto se lleva a cabo utilizando la función FFT() de la toolbox Signal Processing de Matlab el numero de puntos elegidos para el calculo de la TFT es de 512. La salida de esta función esta en forma compleja por lo tanto hay que calcular el modulo de la transformada con la función abs ().

4.3.2.3 Banco de filtros MEL

Se a utilizado un banco de filtros Mel cuya frecuencia más baja es 133,33 Hz la segunda frecuencia esta espaciada 66,66 Hz este espacio continua siendo el mismo para el resto de frecuencias hasta los 1000 Hz a partir de entonces el espacio aumenta logarimicamente. La altura del triangulo también disminuye proporcionalmente a la separación entre frecuencias.

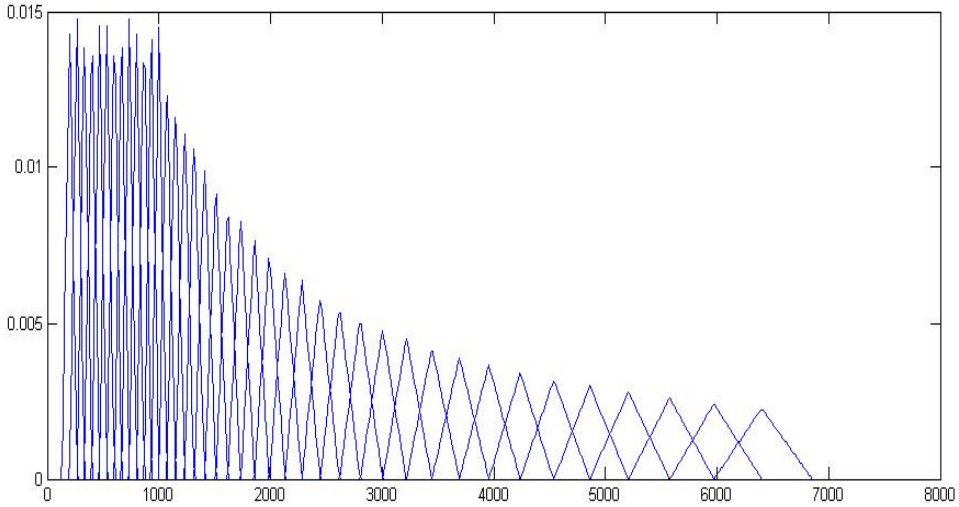


Fig. 4.7 Banco de filtros MEL. Este banco es lineal hasta los 1000 hz y logarítmico a partir de esa frecuencia

4.3.2.4 DCT

Como último paso se calcula la DCT(Transformada Discreta del Coseno) que no es más que la parte real de la transformada discreta de fourier una vez calculada esta transformada con la función de Matlab dct() obtendremos a la salida una matriz de vectores MxN donde M son los 13 primeros coeficientes y N los cuadros de tiempo.

4.3.3 Obtención de los modelos del HMM

4.3.3.1 Calculo de codebook

Una vez tenemos los vectores de características es necesario calcular el codebook. Para determinar cual es el codebook en primer lugar se agrupan todos los vectores de características en un único vector de todas las palabras que se utilizan en el entrenamiento, a partir de este vector se calculan los centroides utilizando el algoritmo k-means que es el siguiente:

1) Inicialización: elegir algún método para hallar un codebook inicial {zi, $1 \leq i \leq L$}.

2) Clasificación: clasificar cada vector de entrenamiento {xk} en uno de los clusters Ci Eligiendo el centroide zi más cercano (x Є Ci, si i $d(x,zi) \leq d(x,zj)$ para todo j<>i). Esta Clasificación se llama clasificador de mínima distancia.

3) Actualización del codebook: calcular el centroide de cada cluster como la media de los vectores de entrenamiento contenidos en ese cluster {zi = cent(Ci), $1 \leq i \leq L$}.

4) Terminación: si la variación en la distorsión media global D entre esta iteración y la anterior es inferior a un umbral, PARAR. En caso contrario, ir al Paso 2.
En definitiva, este proceso de minimizar la distorsión media se divide en dos pasos básicos:

1) Reasignar los vectores: Asumiendo que se ha encontrado el centroide zi del cluster Ci, entonces la minimización consiste en asignar el conjunto de los vectores de entrenamiento a su cluster más cercano de acuerdo a la medida de distancia utilizada.

2) Recolocar los centroides: Por otro lado, dadas las particiones, para minimizar se encuentra el nuevo centroide de cada cluster de modo que se minimice la distorsión media dentro del cluster, que como ya hemos visto es el vector media de los vectores del cluster.
Iterando sobre estos dos pasos puede obtenerse un valor de la distorsión media global D más pequeño que el de la iteración anterior.

4.3.3.2 Calculo de los parámetros del modelo

Una vez se tiene el codebook y en el caso que estemos en el proceso de entrenamiento, se pueden calcular los parámetros que definen el modelo para la palabra entrenada y de esta manera creamos una matriz del tipo cell array para cada uno de los parámetros donde cada uno de los elementos de esta matriz sea el parámetro correspondiente a una palabra.

En el caso que estemos en el proceso de reconocimiento se calculara la probabilidad de todos los modelos y se elegirá el que más alta probabilidad tenga de haber generado la secuencia de observación.

Para el ajuste de los parámetros y teniendo una secuencia de observación $O = \{O_1, O_2, ..., O_t\}$ que corresponde con el vector de características que utilizaremos para el proceso de entrenamiento se calculan dichos parámetros utilizando el algoritmo Baum-welch que se explica a continuación:
Antes de describir el proceso de estimación, necesitamos conocer:
-el número esperado de transiciones desde el estado i en O
-el número esperado de transiciones desde el estado i al estado j en O
Previamente se define ξt(i,j) como la probabilidad de estar en el estado i en el instante t y en el estado **j** en el instante t + 1, dado una observación O y el modelo λ.

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

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)}$$

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{k=1}^{N}\sum_{l=1}^{N} \alpha_t(k) a_{kl} b_l(o_{t+1}) \beta_{t+1}(l)}$$

Donde los valores $\alpha_t(i)$ y $\beta_t(i)$ se calculan utilizando el algoritmo avance-retroceso:

Calculo de $\alpha_t(i)$

Consideramos la variable $\alpha_t(i)$ como:

$$\alpha_t(i) = P(o_1, o_2, \ldots, o_t, q_t = i|\lambda)$$

1. Inicialización

$$\alpha_t(i) = \pi_i b_i(o_1) \quad 1 \le i \le N$$

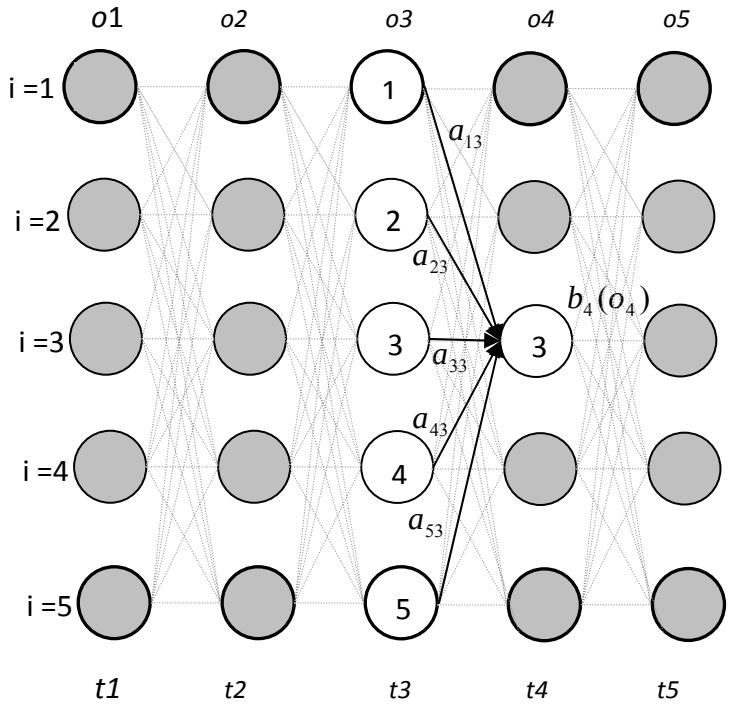
2. Recursión

$$\alpha_t(i) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}) \quad t = 1,2,\ldots,T-1, 1 \le j \le N$$

3. Terminación

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

Ejemplo de calculo de $\alpha_4(3)$



$$\alpha_3(4) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{i3} \right] b_3(o_4)$$

Calculo de $\beta_t(i)$

Consideramos la variable $\beta_t(i)$ como:

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T, q_t = i | \lambda)$$

1. Inicialización

$$\beta_T(i) = 1 \quad 1 \le i \le N$$

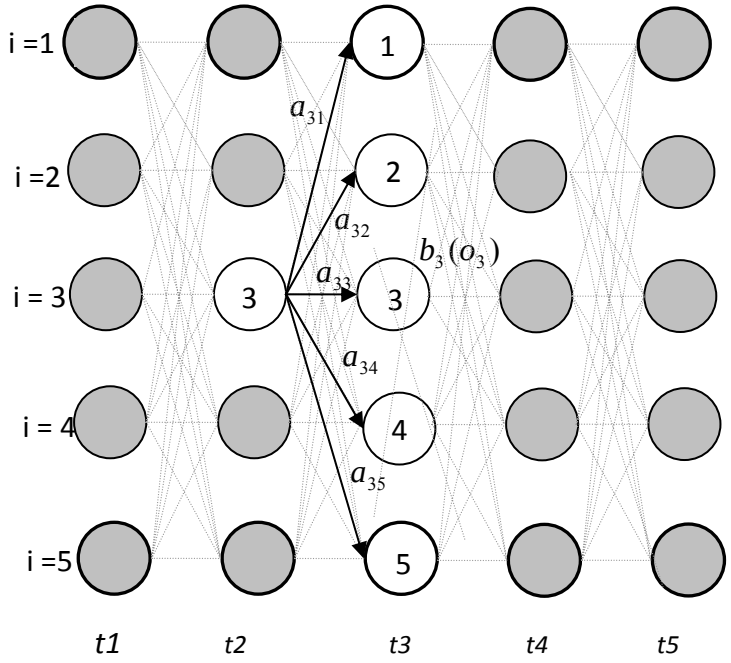
2. Recursión

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} \beta_{t+1}(j) b_j(o_{t+1}) \quad t = T-1, T-2, \ldots, 1, 1 \le i \le N$$

3. Terminación

$$P(o|\lambda) = \sum_{i=1}^{N} \beta_1(i)\pi_i b_i(o_1)$$

Ejemplo de calculo de $\beta_2(3)$



$$\beta_2(3) = \sum_{j=1}^{5} a_{3j}\beta_3(j)b_j(o_3)$$

Definimos también γt(i) como la probabilidad de estar en el estado i en el instante t

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

Sumando cada γt(i) en cada instante de tiempo obtenemos:
El numero esperado de transiciones desde el estado i a la observación o

$$\sum_{t=1}^{T-1} \gamma_t(i)$$

y haciendo lo mismo con cada ξt(i,j):
- el numero esperado de transiciones desde cada estado i hasta cada estado j en la observación o.

$$\sum_{t=1}^{T-1} \xi_t(i, j)$$

Proceso de reestimación

El algoritmo del procedimiento reiterativo es el siguiente:

Se selecciona un modelo inicial elegido aleatoriamente.
Se calculan las transiciones y los símbolos de emisión más probables para el modelo inicial.
Se crea un nuevo modelo en el que se mejora la probabilidad de las transiciones y símbolos determinados en el primer paso, de tal manera que el modelo nuevo tendrá una probabilidad mayor que el modelo antiguo.

Este algoritmo se repite varias veces hasta que no exista mejora entre el modelo anterior y el actual.
Basándose en las expresiones anteriores se obtienen las siguientes formulas recursivas:

Probabilidad de estar en el estado i en el instante t=1:

$$\overline{\pi}_i = \gamma_1 \quad 1 \le i \le N$$

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \qquad \overline{b}_j(k) \frac{\sum_{t=1 \; s.t. o_t = o_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

4.4 Interfaz grafica

4.4.1 Introducción

Existe la posibilidad de incorporar a los programas creados en matlab una interfaz grafica con el objetivo de que el usuario encuentre un entorno interactivo y amigable esta interfaz grafica se denomina GUI (Graphic User Interface) podemos optar por dos opciones programar desde cero las GUI o a través de un entorno de programación que recibe el nombre de GUIDE. Este entorno de programación tiene todas las características de otros lenguajes de programación como pueden ser Visual Basic o Visual C++.

4.4.2 creando una interfaz nueva

Para crear una GUI en el entorno de programación GUIDE que incorpora Matlab hemos de introducir en la línea de comandos el comando:

>> GUIDE

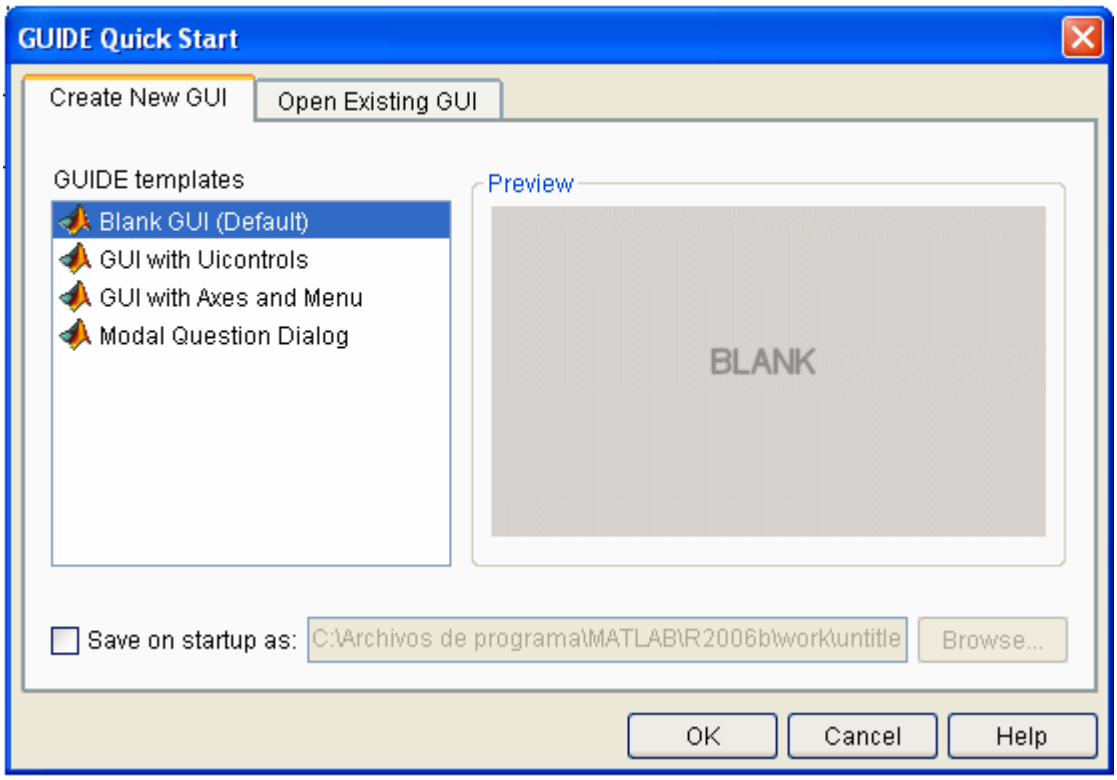


Fig. 4.8 Ventana de inicio de GUI

Una vez aparezca la ventana de inicio de GUI podremos escoger el tipo que queremos en nuestro caso escogeremos la primera opción Blank GUI (Default) que corresponde a una interfaz grafica en blanco.

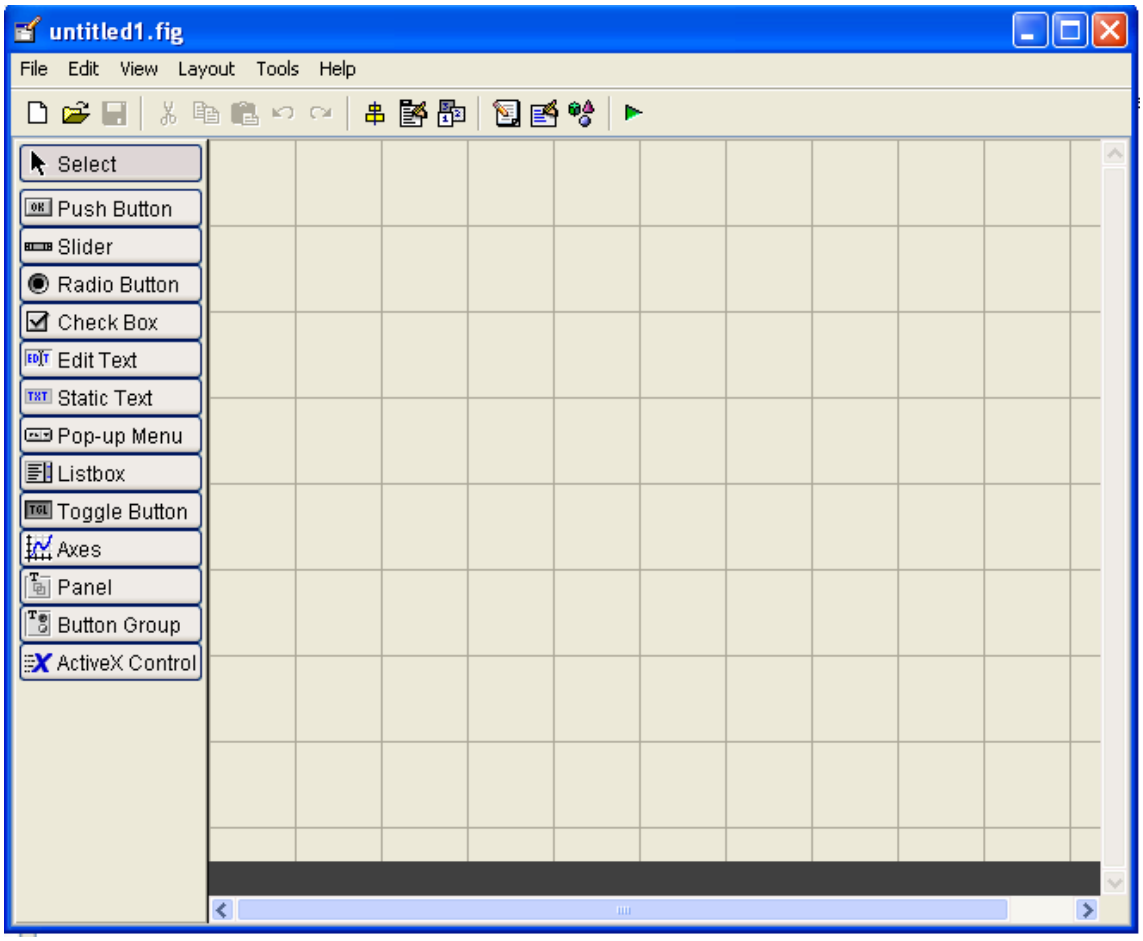


Fig. 4.9 Entorno de diseño de GUI

Llegados a este punto donde ya tenemos el entorno de diseño en la pantalla podemos comenzar a diseñar nuestra GUI. A continuación se hace una descripción de los diferentes componentes que componen en la paleta de diseño:

Objeto	Descripción
Check box	Indica el estado de una opción o atributo
Editable Text	Caja para editar texto
Pop-up menu	Provee una lista de opciones
List Box	Muestra una lista deslizable
Push Button	Invoca un evento inmediatamente
Radio Button	Indica una opción que puede ser seleccionada
Toggle Button	Solo dos estados, “on” o “off”
Slider	Usado para representar un rango de valores
Static Text	Muestra un string de texto en una caja
Panel button	Agrupa botones como un grupo
Button Group	Permite exclusividad de selección con los radio button

Como cualquier lenguaje de programación orientado a objetos cada uno de los objetos tiene unas propiedades que definen su estado y procedimientos ligados a eventos.
Podemos ver las propiedades de cada objeto seleccionándolo y eligiendo la opción Property Inspector del menú View.

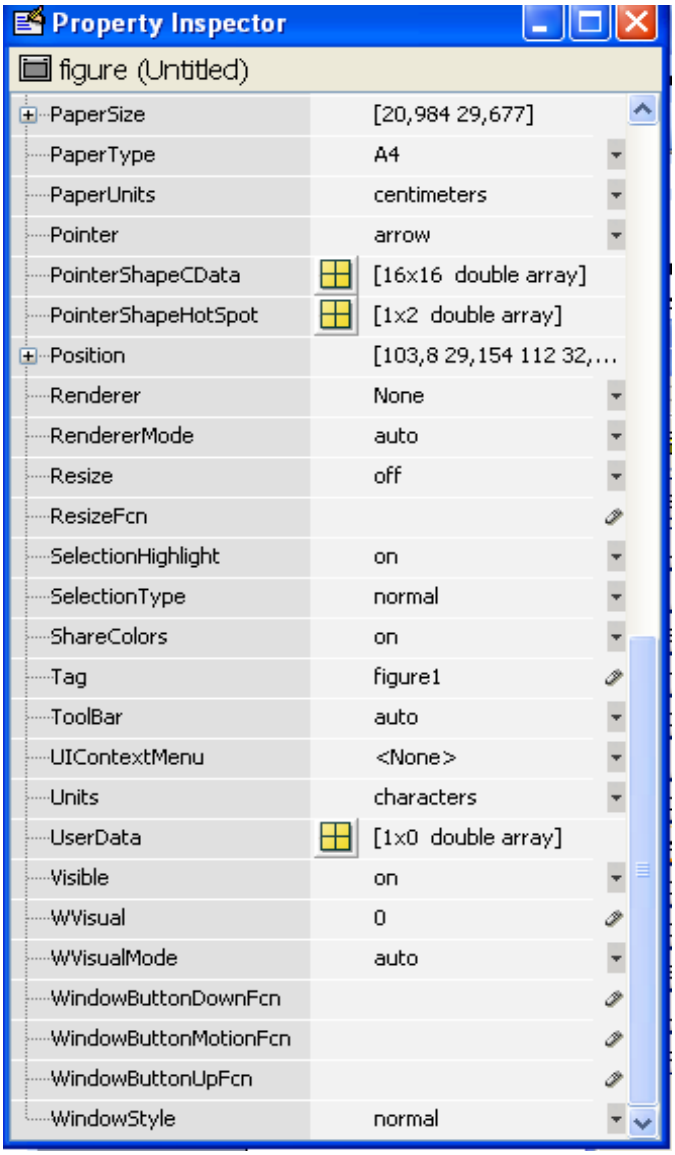


Fig. 4.10 propiedades de los objetos

Una vez diseñada nuestra GUI podemos guardarla, al guardarla se generan dos archivos uno con la extensión (.fig) y el otro con la extensión (.m).

Archivo extensión (.fig): este archivo es el que hemos diseñado hasta ahora contiene los botones cajas de texto, etiquetas, etc.

Archivo extensión (.m): este archivo contiene el código ejecutable de matlab en este código están las subrutinas que van asociadas a cada elemento del archivo (.fig) por ejemplo la subrutina que se tiene que ejecutar al apretar un botón pero también contiene el código referente a las propiedades de los objetos es decir: tamaño, color, etc.

4.4.3 Interfaz del programa

Como se ha dicho anteriormente matlab dispone de la posibilidad de crear una interfaz grafica amigable que permita a un usuario sin conocimientos de programación ejecutar un programa de una manera sencilla. A continuación se explica con detalle la interfaz grafica del programa que implementa un sistema de reconocimiento del habla utilizando modelos ocultos de Harkov.

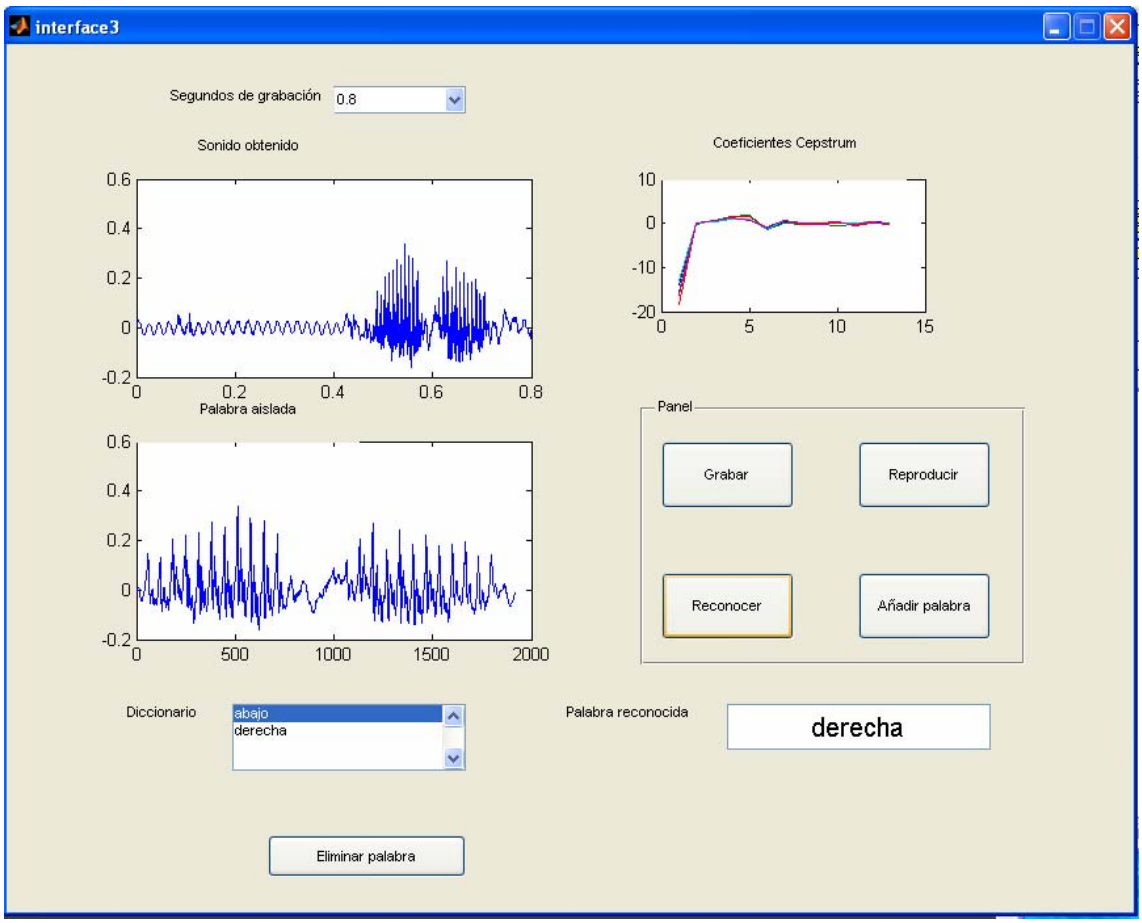


Fig. 4.11 Interfaz gráfica del programa

Al ejecutar el programa nos aparece la ventana que se puede ver en la figura 4.11, como se puede observar se puede elegir los segundos de grabación dependiendo de la palabra a grabar se elegirá uno u otro. Las dos opciones más que se permiten ejecutar son: Grabar y Añadir palabra. Si elegimos la primera opción es decir Grabar se grabarán tantos segundos voz como hayamos fijado anteriormente, en el caso que no se detecte sonido la aplicación nos advertirá de que no se ha detectado sonido. En la imagen también se puede observar que existen dos ejes estos ejes nos mostrarán la señal grabada, en el eje superior se nos mostrará toda la señal en el eje inferior en cambio se nos mostrará la señal recortada es decir la palabra aislada donde a detectado los bordes.

Una vez grabada se activara la posibilidad de escucharla y de reconocerla si elegimos reconocerla el programa reconocer la palabra aislada entre las palabras que tiene en el diccionario y se podrá ver en la caja de texto con la etiqueta "palabra reconocida" las palabras que contiene el diccionario podemos verlas en una lista que hay en la parte inferior izquierda de la ventana. También está disponible la posibilidad de eliminar una palabra del diccionario, para ello seleccionamos la palabra que queremos eliminar y clikamos sobre "Eliminar palabra".

Otra opción que nos ofrece el programa es la de entrenar otra palabra y añadirla al diccionario, para llevar a cabo esto se debe elegir la opción añadir palabra de la ventana principal, una vez elegida esta opción aparecerá la ventana que se puede observar en la fig 4.12

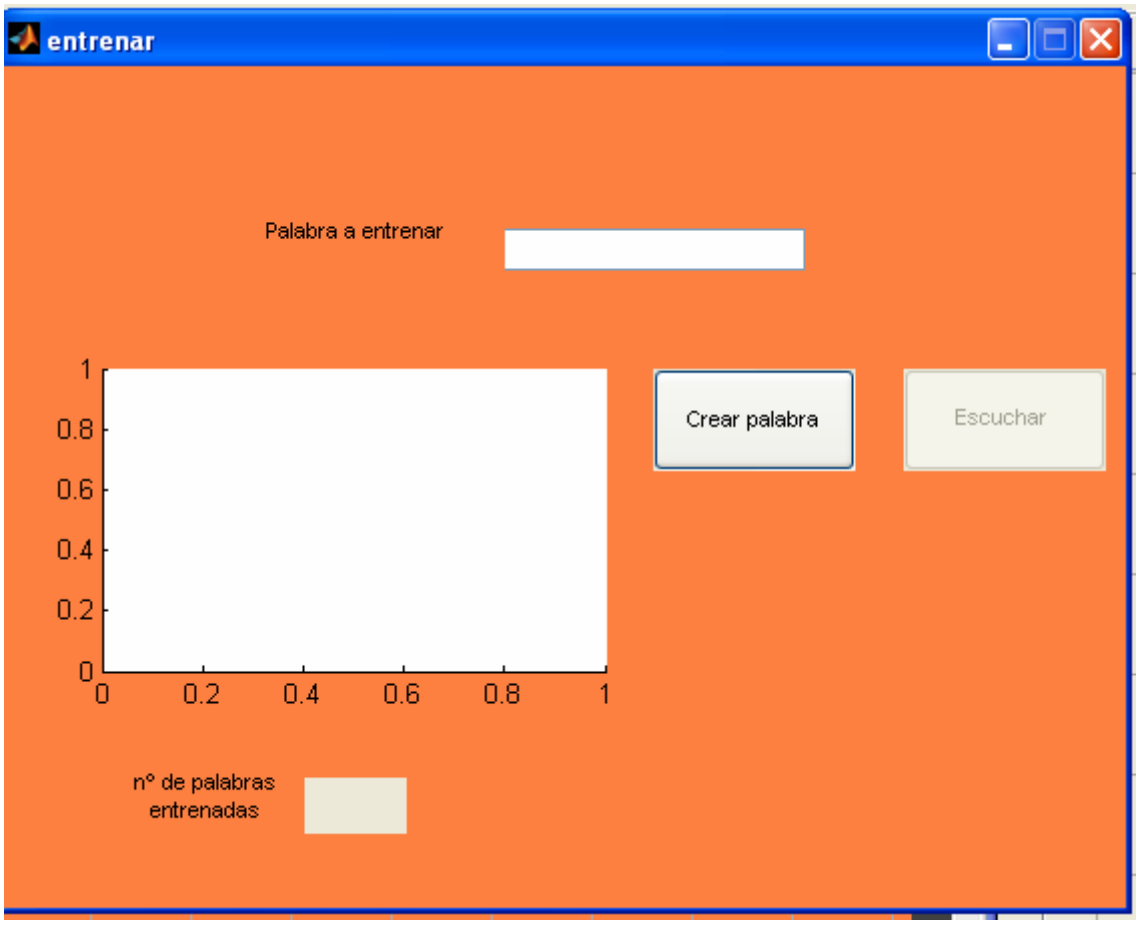


Fig. 4.12 Ventana que aparece al seleccionar la opción "añadir palabra"

5. Conclusiones

El presente proyecto final de carrera consta de cuatro fases para su elaboración, una primera fase para la búsqueda de información, una segunda fase para el diseño e implementación de un sistema de reconocimiento del habla, una tercera fase para la prueba del sistema y una cuarta fase para la elaboración de la presente memoria.

De las cuatro fases podemos destacar una en lo que a tiempo invertido se refiere se trata de la fase de diseño e implementación, dicha fase a su vez podemos dividirla en dos fases una primera fase en la que se diseño un sistema de reconocimiento de vocales utilizando redes neuronales artificiales y una segunda fase donde se diseño un sistema de reconocimiento de palabras utilizando modelos ocultos de Markov, este segundos sistema se implemento tanto en off-line como en tiempo real.

Uno de los problemas que más dificultades a planteado para la elaboración de este proyecto es el hecho de que cuando se pronuncia una palabra se hace muy difícil segmentar los diferentes fonemas para clasificarlos, de hecho este problema es el que n ha permitido que se utilizaran redes neuronales para reconocer palabras si no únicamente vocales es decir fonemas aislados , el problema de la segmentación por el contrario tiene mejor solución en el sistema de reconocimiento utilizando modelos ocultos de Markov ya que la utilización de este sistema conlleva una segmentación puesto que se reconoce la probabilidad de la secuencia de estados en una palabra.

Otro de los problemas que han surgido durante este proyecto es la dificultad que supone el reconocimiento en tiempo real ya que se hace necesario que el procesamiento de los datos sea rápido es decir sin demasiados cálculos.

Podemos concluir diciendo que los objetivos planteados en un principio se han cumplido al menos parcialmente, puesto que por un lado podemos decir que se ha conseguido implementar un sistema de reconocimiento del habla que se puede considerar robusto pero por otra parte se propuso la posibilidad de que el sistema respondiera a los mensajes hablados utilizando una síntesis de voz este objetivo no se ha llevado a cabo debido a que durante el presente proyecto han surgido problemas cuya solución era compleja y que en un principio no parecía que para su resolución se invirtiera tanto tiempo si bien cabe decir que la implementación de dicha síntesis en principio no debería comportar demasiada dificultad.

5.1 Realizaciones futuras

Como ya se ha comentado anteriormente una de las posibles ampliaciones del programa podría consistir en añadirle una síntesis de voz que resultaría interesante en cuanto a la utilización por parte de las personas con déficit visual. Otra posible realización futura podría ser la implementación del programa en orto Hardware como podría ser una DSP en un principio uno de los objetivos del presente proyecto era precisamente este, implementar un sistema de reconocimiento de voz utilizando el TMS320C6713-OE DSP Starter Kit de Texas Intruments para ello sería necesario programar los algoritmos que se han utilizado en el proyecto en leguaje de programación C, en el kit TMS320C671-OE se incorpora un compilador de este lenguaje

6. Bibliografía

[1] O'Shaughnessy, Douglas. Speech communication human and machine (1987).

[2] Sen M.Kuo, Bob.H.Lee and Wenshun Tian. Real-Time Digital Processing Implementation and Applications (2006).

[3] Rulph Chassaing. Digital signal Processing and applications with the C6713 and C6416 DSK (2005).

[4] Jorge Luis Villar Santos, Paz Morillo Bosch. Métodos numéricos con Matlab. Edicions UPC (2003).

[5] Eduard Bertrán Albertí. Señales y sistemas de tiempo discreto. Edicions UPC (2003).

[6] Howard Demeth, Mark Beale. Neural Network Toolbox. For use with Matlab.

[7] L.Rabiner, B. Juang. Fundamentals of speech Recognition. Prentice-Hall (1993).

[8] Duane Hanselman y Bruce Littelfield. Matlab edición estudiante 4 guía de usuario con tutorial. Edición Prentice Hall (1996).

[9] Manuel Berenguel Soria y Teodoro Álamo Cantarero. Tutorial de analisis y control de sitemas usando Matlab .

[10] José B.Mariño, ed. Francesc Vallverdú José A.Rodriguez y Asunción Moreno. Tranamiento de la señal, una introducción experimental. Edicion UPC (1999).

[11] Fernando Martinez Gustavo Portale Hernan Klein Osvaldo Olmos. Reconocimiento de voz, apuntes de cátedra para Introducción a la Inteligencia Artificial.

[12] A.T. Barucha Reid. Elements of the theory of Markov processes and their applications Reid. McGraw Hill (1960).

[13] M. C. José Jaime Esqueda Elizondo. Interfaces Gráficas en Matlab

Usando GUIDE

[14] Prof. Luis A. Hernández Gómez. Tratamiento Digital de Voz

[15] PLP and RASTA (and MFCC, and inversion) in Matlab using melfcc.m and invmelfcc.m. . http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/

[16] Rabiner, L.R. & Schafer, R.W. Digital Processing of Speech Signals, Prentice Hall, Englewood Cliffs, N.J (1978).

[17] Procesamiento Digital de Señales de Voz.
http://www.fceia.unr.edu.ar/prodivoz/home_index.html

[18] Módulo de Reconocimiento de Voz.
http://www.consciousrobots.com/raul/voz/rec_voz.htm

Anexo A : codigo de Matlab de la interfaz gráfica del programa off-line

```
function varargout = interface3(varargin)
% INTERFACE3 M-file for interface3.fig
%      INTERFACE3, by itself, creates a new INTERFACE3 or raises the existing
%      singleton*.
%
%      H = INTERFACE3 returns the handle to a new INTERFACE3 or the handle to
%      the existing singleton*.
%
%      INTERFACE3('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INTERFACE3.M with the given input arguments.
%
%      INTERFACE3('Property','Value',...) creates a new INTERFACE3 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before interface3_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to interface3_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help interface3

% Last Modified by GUIDE v2.5 24-Jan-2008 18:09:00

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @interface3_OpeningFcn, ...
                   'gui_OutputFcn',  @interface3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
```

```
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before interface3 is made visible.
function interface3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to interface3 (see VARARGIN)

% Choose default command line output for interface3
handles.output = hObject;
load etiqueta

 l=size(etiqueta);

 r=etiqueta(1,:);
 for n=2:l(1)
     t=etiqueta(n,:);
     r=[r ; t];
end
set(handles.listbox1,'String',r);
% Update handles structure

set(handles.axes1,'xlim',[0 0.8])
guidata(hObject, handles);

% UIWAIT makes interface3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = interface3_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

axes(handles.axes1);


popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
```

```
    case 1
        v=0.8;
    case 2
        v=1.6;


end
senyal=grabar(round(v));
t=0:1/8000:v;
l=length(t);
if v==0.8
t=t(1:l-1);
else
t=t(1:l-2);
end
handles.senyal=senyal;
plot(t,senyal);
[voiced_msf] = filtro_pasa_bajos(senyal, 8000, 40e-3);
sortida=detec_son(voiced_msf);
y2 = filter([1 -.95], 1, senyal);
[bool_energia] =energia_media(y2, 8000);
if bool_energia<=0.000001,
    msgbox('La senyal no contiene Tramas de sonido','Advertencia');
else

    x=senyal(sortida(1):sortida(2));

    axes(handles.axes2);
    plot(x);
    handles.x=x;
    set(handles.pushbutton2,'enable','on');
    coef = mfc2(x',40e-3,10e-3,8000);
    coef=coef(:,1:13)';
    axes(handles.axes3);
    plot(coef(:,:));
    set(handles.pushbutton2,'enable','on');
    set(handles.pushbutton3,'enable','on');
end
guidata(hObject, handles);

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
%        double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
% load etiqueta
% l=size(etiqueta);
% r=etiqueta(1,:);
% for n=2:l(1)
%     t=etiqueta(n,:);
%     r=[r t];
% end


if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on key press over edit1 with no controls selected.
function edit1_KeyPressFcn(hObject, eventdata, handles)
%set(handles.pushbutton1,'enable','on');
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)




% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

sound(handles.senyal,8000);







% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
%v = {'0.8';'1.6';'2.4'};
popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
    case 1
        v=0.8;
```

```
    case 2
        v=1.6;


end
set(handles.axes1,'xlim',[0 v]);
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load modelos
load etiqueta
load data
load cb
pal{1}{1}=handles.x';

[logp,guess] = hmmrecog2(pal{1}(1),A_m,B_m,pi_m,cb);
set(handles.edit1,'string',etiqueta(guess,:));

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDAT
entrenar
close interface3;




% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = get(hObject,'String') returns listbox1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox1


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on key press over popupmenu1 with no controls selected.
function popupmenu1_KeyPressFcn(hObject, eventdata, handles)

% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)




% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=get(handles.listbox1,'value');
ans=questdlg('Seguro que quiere eliminar la palabra del
diccionario?','reconocedor','Si','No','No');
if strcmp(ans,'No')


    return;
end
load data
load modelos
load etiqueta
set(handles.edit1,'string',n);
etiqueta(n,:)=[];
data(:,n)=[];
A_m(:,n)=[];
B_m(:,n)=[];
pi_m(:,n)=[];
loglike_m(:,n)=[];
d=lon_cd(data,10e-3,40e-3,8000)
if d>=23,
    d=23;
```

```
end
[cb,K,T,dist] = hmmcodebook2(data,20);

save data data
save etiqueta etiqueta
save cb cb
save modelos A_m B_m pi_m loglike_m
```

Anexo B : codigo de Matlab de la interfaz gráfica del programa on-line

```
function varargout = interface3(varargin)
% INTERFACE3 M-file for interface3.fig
%      INTERFACE3, by itself, creates a new INTERFACE3 or raises the existing
%      singleton*.
%
%      H = INTERFACE3 returns the handle to a new INTERFACE3 or the handle to
%      the existing singleton*.
%
%      INTERFACE3('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in INTERFACE3.M with the given input arguments.
%
%      INTERFACE3('Property','Value',...) creates a new INTERFACE3 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before interface3_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to interface3_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help interface3

% Last Modified by GUIDE v2.5 11-Jan-2008 12:11:14

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @interface3_OpeningFcn, ...
                   'gui_OutputFcn',  @interface3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
global valor
valor=1;

% --- Executes just before interface3 is made visible.
function interface3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to interface3 (see VARARGIN)

% Choose default command line output for interface3
handles.output = hObject;
load etiqueta

 l=size(etiqueta);

 r=etiqueta(1,:);
 for n=2:l(1)
     t=etiqueta(n,:);
     r=[r ; t];
end
set(handles.listbox1,'String',r);
% Update handles structure

set(handles.axes1,'xlim',[0 0.8])
guidata(hObject, handles);

% UIWAIT makes interface3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = interface3_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global senyal
global nuevo
```

```
global t
global valor
global tot
global flag
flag=1;
prueba=1;

tot=1;

nuevo=handles;
axes(handles.axes1);


popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
   case 1
      v=0.8;
   case 2
      v=1.6;


end
%grabar(handles);


numero_buffer=1;
ai=analoginput('winsound');
addchannel(ai, 1);

%flag=0;


set(ai, 'SampleRate',8000);
set(ai, 'SamplesPerTrigger',inf);
set(ai, 'TriggerRepeat', 1);
set(ai, 'TriggerType', 'manual');

set(ai, 'TimerPeriod', 0.1);
 set(ai, 'SamplesAcquiredFcnCount', 1024*8);

 set(ai, 'SamplesAcquiredFcn', @localfftShowData);

Fs=8000;
toffset=40e-3;
offset=Fs*40*10^(-3);
set(ai,'userdata',handles);

start(ai);
trigger(ai);



[bool_energia] =0;

t=0:1/8000:v;
```

```
l=length(t);
if v==0.8
t=t(1:l-1);
else
t=t(1:l-2);
end


handles.senyal=tot;



guidata(hObject, handles);

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
%        double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
% load etiqueta
% l=size(etiqueta);
% r=etiqueta(1,:);
% for n=2:l(1)
%     t=etiqueta(n,:);
%     r=[r t];
% end


if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on key press over edit1 with no controls selected.
function edit1_KeyPressFcn(hObject, eventdata, handles)
%set(handles.pushbutton1,'enable','on');
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

sound(handles.senyal,8000);




% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
%v = {'0.8';'1.6';'2.4'};
popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
   case 1
      v=0.8;
   case 2
      v=1.6;


end
set(handles.axes1,'xlim',[0 v]);
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load modelos
load etiqueta
```

```
load data
load cb
pal{1}{1}=handles.x';

[logp,guess] = hmmrecog2(pal{1}(1),A_m,B_m,pi_m,cb);
set(handles.edit1,'string',etiqueta(guess,:));

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDAT

entrenar
close interface3;

% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listbox1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox1


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on key press over popupmenu1 with no controls selected.
function popupmenu1_KeyPressFcn(hObject, eventdata, handles)

% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)




function localfftShowData(ai,event)
global senyal
```

```
global tot
global valor
global flag
%%
% Get the handles.
datos = ai.UserData;
senyal = peekdata(ai,1024*8);
h=get(datos.figure1);
n=get(datos.axes1);
longitud=get(ai, 'SamplesPerTrigger');
set(datos.axes1, 'HandleVisibility','on');
plot(datos.axes1,senyal);



 [voiced_msf] = filtro_pasa_bajos(senyal, 8000, 40e-3);
sortida=detec_son(voiced_msf);
y2 = filter([1 -.95], 1, senyal);
[bool_energia] =energia_media(y2, 8000);
if bool_energia>=0.001 && (sortida(2)- sortida(1))>=3000,


   x=senyal(sortida(1):sortida(2));


   plot(datos.axes2,x);
   handles.x=x;

   coef = mfc2(x,40e-3,10e-3,8000);
   coef=coef(:,1:13)';
 load modelos
load etiqueta
load data
load cb
pal{1}{1}=x;

[logp,guess] = hmmrecog2(pal{1}(1),A_m,B_m,pi_m,cb);
set(datos.edit1,'string',etiqueta(guess,:));
else

end

drawnow;
```

Anexo C : Codigo de Matlab de la interfaz gráfica “Entrenar”

```
function varargout = entrenar(varargin)
% ENTRENAR M-file for entrenar.fig
%      ENTRENAR, by itself, creates a new ENTRENAR or raises the existing
%      singleton*.
%
%      H = ENTRENAR returns the handle to a new ENTRENAR or the handle to
%      the existing singleton*.
%
%      ENTRENAR('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ENTRENAR.M with the given input arguments.
%
%      ENTRENAR('Property','Value',...) creates a new ENTRENAR or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before entrenar_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to entrenar_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help entrenar

% Last Modified by GUIDE v2.5 22-Dec-2007 15:42:36

% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @entrenar_OpeningFcn, ...
                   'gui_OutputFcn',  @entrenar_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before entrenar is made visible.
function entrenar_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to entrenar (see VARARGIN)

% Choose default command line output for entrenar
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes entrenar wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = entrenar_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global n_repeticiones;
global paso;
h=get(handles.seleccion,'enable');
c=strcmp(h,'on');
if c==1,
    paso=1;
    nombre=get(handles.seleccion,'string');
    handles.nombre=nombre;
    load etiqueta
    handles.etiqueta=etiqueta;
    load data
    handles.data=data;
    n_palabras=size(etiqueta);
    n_palabras=n_palabras(1);
    handles.n_palabras=n_palabras+1;
```

```
    n_repeticiones=1;
    %handles.n_repeticiones=n_repeticiones;
    handles.lon=length(handles.nombre);
    %       handles.primero=0;
else
    %handles.n_repeticiones=(handles.n_repeticiones)+1;
    n_repeticiones=n_repeticiones+1;
    load data
end
%handles.n_palabras=n_palabras;

handles.data=data;
nombre=get(handles.seleccion,'string');
%handles.n_repeticiones=handles.n_repeticiones+1;
palabra=wavrecord(8000*3,8000);
%palabra=palabra';
[voiced_msf] = filtro_pasa_bajos(palabra, 8000, 40e-3);
sortida=detec_son2(voiced_msf);
palabra=palabra(sortida(1):sortida(2));
handles.palabra=palabra;
set(handles.seleccion,'enable','off');
set(handles.pushbutton1,'enable','off');
set(handles.pushbutton2,'enable','on');
axes(handles.axes1);
plot(palabra);
guidata(hObject, handles);

function seleccion_Callback(hObject, eventdata, handles)
% hObject    handle to seleccion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of seleccion as text
%        str2double(get(hObject,'String')) returns contents of seleccion as a double


% --- Executes during object creation, after setting all properties.
function seleccion_CreateFcn(hObject, eventdata, handles)
% hObject    handle to seleccion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end




% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
global n_repeticiones
global paso
set(handles.pushbutton3,'visible','on');
sound(handles.palabra);
%questdlg('Quiere añadir la palabra al diccionario?',' reconocedor');
ans=questdlg('Desea añadir la palabra al diccionario?','reconocedor','Si','No','No');
if strcmp(ans,'No')
   set(handles.pushbutton1,'string','Siguiente');
   set(handles.pushbutton1,'enable','on');
   %if n_repeticiones>=2 %&& paso==1,
      n_repeticiones=n_repeticiones-1;
     % paso=0;

   %end
   return;
end

% load data
% load etiqueta
% handles.n_palabras=size(etiqueta);
% handles.n_palabras=handles.n_palabras(1);
% handles.n_repeticiones=length(data{handles.n_palabras});
% lon=length(handles.nombre);
% if handles.primero==0,
%     data{(handles.n_palabras+1)}{1}=handles.palabra;
%     etiqueta((handles.n_palabras)+1,1:lon)=handles.nombre;
%     handles.primero=1;
%     save data data
%     save etiqueta etiqueta
% else
   %handles.data{(handles.n_palabras)}{((handles.n_repeticiones))}=handles.palabra;
%paso=1;
handles.data{(handles.n_palabras)}{((n_repeticiones))}=handles.palabra;
handles.etiqueta(handles.n_palabras,1:handles.lon)=handles.nombre;
set(handles.text2,'string',n_repeticiones);
data=handles.data;
etiqueta=handles.etiqueta;
save data data
save etiqueta etiqueta
% end
%guidata(hObject, handles);
set(handles.pushbutton1,'string','Siguiente');
set(handles.pushbutton1,'enable','on');
guidata(hObject, handles);

%


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)

ans=questdlg('Desea entrenar los datos?','reconocedor','Si','No','No');
if strcmp(ans,'No')
```

```
    clear,clc,close entrenar
    interface3
    return;
end
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
data=handles.data;
etiqueta=handles.etiqueta;
save data data
save etiqueta etiqueta
load modelos
% load data
% load etiqueta

%d=lon_cd(data,10e-3,40e-3,8000)
%if d>=23,
%    d=23;
%end
[cb,K,T,dist] = hmmcodebook2(data,20);
longitud1=length(A_m);
longitud2=size(etiqueta);
longitud2=longitud2(1);
%for i=longitud1+1:longitud2+1,
[A_m1,B_m1,pi_m1,loglike_m1] = entrenamiento_hmm(data,6,longitud2,cb,500,1e-
3,8000,10e-3,40e-3);
A_m=[A_m A_m1{longitud2}];
B_m=[B_m B_m1{longitud2}];
pi_m=[pi_m pi_m1{longitud2}];
loglike_m=[loglike_m loglike_m1{longitud2}];
%end

save modelos A_m B_m pi_m loglike_m
save cb cb
close entrenar
interface3
```

Anexo D : Funciones del programa

Funciones utilizadas en el reconocimiento utilizando HMM

```
function [A_m,B_m,pi_m,loglike_m] = hmmtrain2(data,cb,S,maxiter,tol)
%función entrenamiento de los parametros HMM

[dim,K] = size(cb);
R = length(data);

A_m  = cell(R,1);
B_m  = cell(R,1);
pi_m = cell(R,1);
loglike_m = cell(R,1);

for (i = 1:R)
  L = length(data{i});

  ytrain = cell(L,1);
  for (j = 1:L)
    file_s = data{i}{j};

    s=file_s;

    y=mfc2(s,11025);
    [dim,T] = size(y);

    yk = zeros(T,1);
    for (t = 1:T)
      [dist,k] = min(sum((cb-repmat(y(:,t),1,K)).^2));
      yk(t) = k;
    end;
    ytrain(j) = {yk};
  end

  [A,B,pi1,loglike] = hmmfb(ytrain,S,K,maxiter,tol);

  A_m(i)  = {A};
  B_m(i)  = {B};
  pi_m(i) = {pi1};
  loglike_m(i) = {loglike};
end



function [bool_energia] =energia_media(x, fs);
```

```
%función que dectecta las tramas con sonido


m=1;
frame=length(x);
for n=10:20,
    resto(m)=rem(frame,n*0.001*8000);
   m=m+1;
end
small=min(resto);
minimo=find(resto==small);
x=x(1:frame-min(resto));

fsize=(minimo(1)+9)*0.001;
 frame_length = round(fs .* fsize);

N= frame_length - 1;

for b=1 : frame_length : (length(x)),
    y1=x(b:b+N);
      y = filter([1 -.9378], 1, y1);

    ener(b:(b + N)) = energia (y);

end



bool_energia=sum(ener)/length(ener);



function [coef] = mfc2(x,desplazamiento,long_ventana,fs)

%función que extrae el vector de caracteristicas de una señal de voz

%parametros de entrada
%x=señal de entrada
%long_ventana=longitud de la ventana Hamming
%desplazamiento=desplazamiento de cada cuadro, para que exista solapamiento
long_ventana>desplazamiento
%fs=frecuencia de muestreo
%salida
%coef=matriz de coeeficientes cepstrales
coef=[];
fftSize=512;

desp=desplazamiento*fs;
window=long_ventana*fs;


frame=length(x);
tamany = round(frame/desp);
for i=1:tamany,
```

```
   if i*desp+window>=frame,
    y(i*desp+1:frame)=0;
   else
   y1=x((i*desp)+1:(i*desp)+window-1);
   y3= y1 .* hamming(length(y1));
   y = filter([1 -.95], 1, y3);
   y2=abs(fft(y,fftSize));

    banco_filtros=frecuencias(fs,fftSize,frame);
    cepstrum=log10(banco_filtros*y2);
%       coeficientes_cepstrum(i,:)=dct(cepstrum);

     coef(i,:)=dct(cepstrum);

   end

end

function [cb,K,T,dist] = hmmcodebook2(data,Kmax)

% funcion que calcula el codebook

R = length(data);
Y = [];
for (i = 1:R)
  L = length(data{i});
  for (j = 1:L)
    file_s = data{i}{j};

    s=file_s;


    y = mfc2(s,10e-3,30e-3,8000)
    y=y(:,1:13)';
    Y = [Y y];
  end
end

[Yc,c,errlog] = kmeans(Y',Kmax,5000);
cb = Yc(unique(c),:)';

[N,K] = size(cb);
[N,T] = size(Y);
dist  = sqrt(errlog(end))/T;



function [Yc,c,errlog] = kmeans(Y,K,maxiter)

%función que calcula los centroides utilizando le algoritmo k-means
[M,N] = size(Y);
if (K > M)
  error('Mas centroides que vectores')
end
```

```
errlog = zeros(maxiter,1);
perm = randperm(M);
Yc   = Y(perm(1:K),:);


d2y = (ones(K,1)*sum((Y.^2)'))';

for (i = 1:maxiter)

  Yc_old = Yc;


  d2 = d2y + ones(M,1)*sum((Yc.^2)') - 2*Y*Yc';


  [errvals,c] = min(d2');


  for (k = 1:K)
    if (sum(c==k)>0)
      Yc(k,:) = sum(Y(c==k,:))/sum(c==k);
    end
  end


  errlog(i) = sum(errvals);

  if (max(max(abs(Yc - Yc_old))) < 10*eps)
    errlog = errlog(1:i);
    return
  end
end



function banco_filtros = frecuencias(Fs,fftSize,windowSize);
% banco de filtros en escala MEL, esta función calcula las Frecuencias que
% estan espaciadas linealmente hasta los 1000 Hz y desde los 1000 a los
% 4000 hz su destancia es logaritmica
lowestFrequency = 133.3333;
linearFilters = 13;
linearSpacing = 66.66666666;
logFilters = 27;
logSpacing = 1.0711703;

cepstralCoefficients = 13;
windowSize = 400;
windowSize = 256;


totalFilters = linearFilters + logFilters;



freqs = lowestFrequency + (0:linearFilters-1)*linearSpacing;
```

```
freqs(linearFilters+1:totalFilters+2) = ...
            freqs(linearFilters) * logSpacing.^(1:logFilters+2);

lower = freqs(1:totalFilters);
center = freqs(2:totalFilters+1);
upper = freqs(3:totalFilters+2);

mfccFilterWeights = zeros(totalFilters,fftSize);
triangleHeight = 2./(upper-lower);
fftFreqs = (0:fftSize-1)/fftSize*Fs; %escala de frecuencias de la FFT

for chan=1:totalFilters
    banco_filtros(chan,:) = ...
  (fftFreqs > lower(chan) & fftFreqs <= center(chan)).* ...
   triangleHeight(chan).*(fftFreqs-lower(chan))/(center(chan)-lower(chan)) + ...
  (fftFreqs > center(chan) & fftFreqs < upper(chan)).* ...
   triangleHeight(chan).*(upper(chan)-fftFreqs)/(upper(chan)-center(chan));
end
```

funciones utilizadas en el reconocimento utilizando Redes Neuronales

```
%Funcion que reconoce vocales

valor=1;
numero_buffer=1;
ai=analoginput('winsound');
addchannel(ai, 1);


set(ai, 'SampleRate',8000);
set(ai, 'SamplesPerTrigger',6400);
set(ai, 'TriggerRepeat', 1);
set(ai, 'TriggerType', 'Immediate');
set(ai, 'TimerPeriod', 0.1);
Fs=8000;
toffset=40e-3;
offset=Fs*40*10^(-3);
segundos=2;

while valor<=segundos


 start(ai);

 numero_de_muestras = ai.SamplesAvailable;
 while numero_de_muestras<6400,
   numero_de_muestras = ai.SamplesAvailable;
 end
 stop(ai);

 [d,t] =getdata(ai);
 longitud=length(d);
 y = filter([1 -.9378], 1, d);
```

```
 if valor==1,
   senyal((valor):((valor+longitud-1)))= y;
 else
    senyal(((valor-1)*longitud):(((((valor-1)*longitud)+longitud-1))))=y;
 end
 valor=valor+1;
end
senyal(12800)=zeros;
rangom=min(senyal);
rangoM=max(senyal);
valoresx=[min(senyal) max(senyal)];


plot(senyal);

%[h_media] =entropia_media(senyal,valoresx, 8000);
[voiced_msf] = filtro_pasa_bajos(senyal, 8000, toffset);
sortida=detec_son2(voiced_msf);
[bool_energia] =energia_media(senyal, 8000);
%sortida2=detec_son(voiced_msf);
x=senyal(sortida(1):sortida(2));
%[h_media] =entropia_media(x,valoresx, 8000);
figure;

plot(voiced_msf);
grid;
[coeficientes_cepstrum] =mfc(x, 8000);

[voiced_zc] = cruces_por_cero(x, Fs);


total=on_line(x,Fs,toffset);%calcula els 4 primers formants del senyal
for n=1:3,
    salida_fin(n)=mean(total(n,:));
end
out = output(salida_fin,1);%entrada de la red neuronal la media de los 4 formantes obtenidos



function total=on_line(x,fs,toffset)
% función que calcula los 4 primeros formantes de la vocal
m=1;
frame=length(x);
for n=30:40,
    resto(m)=rem(frame,n*0.001*8000);
   m=m+1;
end
small=min(resto);
minimo=find(resto==small);
x=x(1:frame-min(resto));

fsize=(minimo(1)+29)*0.001;
 frame_length = round(fs .* fsize);

N= frame_length - 1;
```

```
for z=1:frame_length:(length(x)),

   tama=hamming(N+1)'.*x(z:z+ N);


   %if z==1,

%    if voiced((z+frame_length)-2)==1,
      coef=lpc(tama,10);

      raizes=roots(coef);
      loc = raizes ;
      ind = find(imag(loc)>0) ;% posicion en raices que tiene parte imaginaria positiva
      rai = loc(ind) ;
      [urai irai]=sort(abs(rai)) ;
      rai= rai(fliplr(irai')')   ;
      numraiz= size(rai);
      rai = rai(1:4) ;
      [xra xrai]=sort(real(rai)) ;
      rai = rai(fliplr(xrai')' ) ;
      fsred=fs/(2*pi);

      k=size(rai);

      for j=1:k,
%Cálculo de los Formantes
         r = rai(j) ;
         formante(j)=atan2(imag(r),real(r))*fsred;
         a(j)=log10(abs(rai(j)));
%plot([1 2 3 4] ,formante)

         representacion(j,(z:(z+frame_length-1)))=formante(j);
     end

end
total=representacion;


function out = output(salida_fin,n);
%función que clasifica la vocal utilizando la RNA previamente entrenada
out=0;
load 'redentrenada.mat';

%formante1=salida_fin(n,1:3);
formante1=salida_fin;
clas=sim(red,formante1');
media_clas= clas';
vmax=max(media_clas);
for r=1:5
   if media_clas(r)~=vmax
      media_clas(r)=0;
   end
end
index=find(media_clas>0);
```

```
switch (index)
   case 1
      out='vocal a';
   case 2
      out='vocal e';
   case 3
      out='vocal i';
   case 4
      out='vocal o';
   case 5
      out='vocal u';
end


function en = entrena_red(res,fs, vocal)

%función que entrena la red neuronal artificial

vocal_a=[0 0 0 0 1; 0 0 0 0 1; 0 0 0 0 1; 0 0 0 0 1; 0 0 0 0 1];
vocal_e=[0 0 0 1 0; 0 0 0 1 0; 0 0 0 1 0; 0 0 0 1 0; 0 0 0 1 0];
vocal_i=[0 0 1 0 0; 0 0 1 0 0; 0 0 1 0 0; 0 0 1 0 0; 0 0 1 0 0];
vocal_o=[0 1 0 0 0; 0 1 0 0 0; 0 1 0 0 0; 0 1 0 0 0; 0 1 0 0 0];
vocal_u=[1 0 0 0 0; 1 0 0 0 0; 1 0 0 0 0; 1 0 0 0 0; 1 0 0 0 0];

formante = vectfun(fs, res);

formantes = formante(1:5,1:2);

if vocal== 1
   salida=vocal_a';
elseif vocal== 2
    salida=vocal_e';
elseif vocal== 3
   salida=vocal_i';
elseif vocal== 4
   salida=vocal_o';
else
   salida=vocal_u';
end

entrena=formantes';

red=train(red,entrena,salida);


function vocal = clasificacion(fs, datos);
%funcion que clasifica las vocales a partir de una separación de los
%formantes
formante = vectfun(fs, datos);
mediaformante=mean(formante);
media1=mediaformante(1,1);
media2=mediaformante(1,2);
if media1 <= 250
   if media2 <= 1000
```

```
      fprintf('vocal u')
   else
   fprintf('vocal i')
   end
elseif media1 <= 600
   if media2 <= 1500
     fprintf('vocal o')
   else
     fprintf('vocal e')
  end
else
   fprintf('vocal a')

end


function crea_red=red()

%funcion que crea la red neuronal artificial con arquitectura
%backpropagation
in=[ 0 2500; 300 3000; 0 4000];
crea_red= newff(in,[3 5],{'purelin' 'purelin'},'trainlm');


%crea un objeto que contiene una base de dato de las cinco vocales
load('vocales2.mat');
entrada = 0;
salida = 0;
for j=1:3
j=4;
perioa=a(j).datos;
   formantea = vectfun(8000, perioa);
   lona = length(formantea);
   formantea1 = formantea(1:lona,1:2);

   perioe=e(j).datos;
   formantee = vectfun(8000, perioe);
   lone = length(formantee);
   formantee1 = formantee(1:lone,1:2);

   perioi=i(j).datos;
   formantei = vectfun(8000, perioi);
   loni= length(formantei);
   formantei1 = formantei(1:loni,1:2);

   perioo=o(j).datos;
   formanteo = vectfun(8000, perioo);
   lono= length(formanteo);
   formanteo1 = formanteo(1:lono,1:2);

   periou=u(j).datos;
   formanteu = vectfun(8000, periou);
   lonu = length(formanteu);
   formanteu1 = formanteu(1:lonu,1:2);
   if entrada == 0
```

```
    entrada = [formantea1' formantee1' formantei1' formanteo1' formanteu1'];
  else
    entrada = [entrada formantea1' formantee1' formantei1' formanteo1' formanteu1'];
  end
  %for n=1:lona
   salidaa = zeros(lona,5);
   salidaa(:,1)=1;
   %salidaa(1:lona, ) = [1 0 0 0 0];
   %end

  %for n=1:lone
   salidae = zeros(lone,5);
   salidae(:,2)=1;
   %end

  %for n=1:loni
    salidai = zeros(lone,5);
    salidai(:,3)=1;
   %end

  %for n=1:lono
   salidao = zeros(lono,5);
   salidao(:,4)=1;
   %end

  %for n=1:lonu
   salidau = zeros(lonu,5);
   salidau(:,5)=1;
   %end
  if salida == 0
    salida = [salidaa' salidae' salidai' salidao' salidau'];
  else
    salida = [salida salidaa' salidae' salidai' salidao' salidau'];
  end
%  red=train(red,entrada,salida);

end
```

```
function b=dct(a,n)
%DCT  Discrete cosine transform.
%
%   Y = DCT(X) returns the discrete cosine transform of X.
```

```
%   The vector Y is the same size as X and contains the
%   discrete cosine transform coefficients.
%
%   Y = DCT(X,N) pads or truncates the vector X to length N
%   before transforming.
%
%   If X is a matrix, the DCT operation is applied to each
%   column.  This transform can be inverted using IDCT.
%
%   See also FFT, IFFT, IDCT.

%   Author(s): C. Thompson, 2-12-93
%              S. Eddins, 10-26-94, revised
%   Copyright 1988-2004 The MathWorks, Inc.
%   $Revision: 1.7.4.2 $  $Date: 2004/12/26 22:15:37 $

%   References:
%   1) A. K. Jain, "Fundamentals of Digital Image
%      Processing", pp. 150-153.
%   2) Wallace, "The JPEG Still Picture Compression Standard",
%      Communications of the ACM, April 1991.


if nargin == 0,
    error('Not enough input arguments.');
end

if isempty(a)
   b = [];
   return
end

% If input is a vector, make it a column:
do_trans = (size(a,1) == 1);
if do_trans, a = a(:); end

if nargin==1,
  n = size(a,1);
end
m = size(a,2);

% Pad or truncate input if necessary
if size(a,1)<n,
  aa = zeros(n,m);
  aa(1:size(a,1),:) = a;
else
  aa = a(1:n,:);
end

% Compute weights to multiply DFT coefficients
ww = (exp(-i*(0:n-1)*pi/(2*n))/sqrt(2*n)).';
ww(1) = ww(1) / sqrt(2);

if rem(n,2)==1 || ~isreal(a), % odd case
  % Form intermediate even-symmetric matrix
  y = zeros(2*n,m);
```

```
  y(1:n,:) = aa;
  y(n+1:2*n,:) = flipud(aa);

  % Compute the FFT and keep the appropriate portion:
  yy = fft(y);
  yy = yy(1:n,:);

else % even case
  % Re-order the elements of the columns of x
  y = [ aa(1:2:n,:); aa(n:-2:2,:) ];
  yy = fft(y);
  ww = 2*ww;  % Double the weights for even-length case
end

% Multiply FFT by weights:
b = ww(:,ones(1,m)) .* yy;

if isreal(a), b = real(b); end
if do_trans, b = b.'; end


function net = newff(pr,s,tf,btf,blf,pf)
%NEWFF Create a feed-forward backpropagation network.
%
%  Syntax
%
%    net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)
%
%  Description
%
%    NEWFF(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF) takes,
%      PR  - Rx2 matrix of min and max values for R input
elements.
%      Si  - Size of ith layer, for Nl layers.
%      TFi - Transfer function of ith layer, default = 'tansig'.
%      BTF - Backprop network training function, default =
'trainlm'.
%      BLF - Backprop weight/bias learning function, default =
'learngdm'.
%      PF  - Performance function, default = 'mse'.
%    and returns an N layer feed-forward backprop network.
%
%    The transfer functions TFi can be any differentiable
transfer
%    function such as TANSIG, LOGSIG, or PURELIN.
%
%    The training function BTF can be any of the backprop
training
%    functions such as TRAINLM, TRAINBFG, TRAINRP, TRAINGD, etc.
%
%    *WARNING*: TRAINLM is the default training function because
it
%    is very fast, but it requires a lot of memory to run.  If
you get
%    an "out-of-memory" error when training try doing one of
these:
```

```
%
%    (1) Slow TRAINLM training, but reduce memory requirements,
by
%        setting NET.trainParam.mem_reduc to 2 or more. (See
HELP TRAINLM.)
%    (2) Use TRAINBFG, which is slower but more memory efficient
than TRAINLM.
%    (3) Use TRAINRP which is slower but more memory efficient
than TRAINBFG.
%
%    The learning function BLF can be either of the
backpropagation
%    learning functions such as LEARNGD, or LEARNGDM.
%
%    The performance function can be any of the differentiable
performance
%    functions such as MSE or MSEREG.
%
%  Examples
%
%    Here is a problem consisting of inputs P and targets T that
we would
%    like to solve with a network.
%
%      P = [0 1 2 3 4 5 6 7 8 9 10];
%      T = [0 1 2 3 4 3 2 1 2 3 4];
%
%    Here a two-layer feed-forward network is created.  The
network's
%    input ranges from [0 to 10].  The first layer has five
TANSIG
%    neurons, the second layer has one PURELIN neuron.  The
TRAINLM
%    network training function is to be used.
%
%      net = newff(minmax(P),[5 1],{'tansig' 'purelin'});
%
%    Here the network is simulated and its output plotted
against
%    the targets.
%
%      Y = sim(net,P);
%       plot(P,T,P,Y,'o')
%
%    Here the network is trained for 50 epochs.  Again the
network's
%     output is plotted.
%
%      net.trainParam.epochs = 50;
%      net = train(net,P,T);
%      Y = sim(net,P);
%       plot(P,T,P,Y,'o')
%
%  Algorithm
%
```

```
%    Feed-forward networks consist of Nl layers using the
DOTPROD
%    weight function, NETSUM net input function, and the
specified
%    transfer functions.
%
%    The first layer has weights coming from the input.  Each
subsequent
%    layer has a weight coming from the previous layer.  All
layers
%    have biases.  The last layer is the network output.
%
%    Each layer's weights and biases are initialized with
INITNW.
%
%    Adaption is done with TRAINS which updates weights with the
%    specified learning function. Training is done with the
specified
%    training function. Performance is measured according to the
specified
%    performance function.
%
%  See also NEWCF, NEWELM, SIM, INIT, ADAPT, TRAIN, TRAINS

% Mark Beale, 11-31-97
% Copyright 1992-2005 The MathWorks, Inc.
% $Revision: 1.1.6.2 $

if nargin < 2
  net = newnet('newff');
  return
end

% Defaults
if nargin < 4, btf = 'trainlm'; end
if nargin < 5, blf = 'learngdm'; end
if nargin < 6, pf = 'mse'; end

% Error checking
if isa(pr,'cell') && all(size(pr)==[1 1]), pr = pr{1,1}; end
if (~isa(pr,'double')) | ~isreal(pr) | (size(pr,2) ~= 2)
  error('Input ranges is not a two-column matrix, or cell array
with a single two-column matrix.')
end
if any(pr(:,1) > pr(:,2))
  error('Input ranges has values in the second column larger in
the values in the same row of the first column.')
end
if isa(s,'cell')
  if (size(s,1) ~= 1)
    error('Layer sizes is not a row vector of positive
integers.')
  end
  for i=1:length(s)
    si = s{i};
```

```
    if ~isa(si,'double') | ~isreal(si) | any(size(si) ~= 1) |
any(si<1) | any(round(si) ~= si)
      error('Layer sizes is not a row vector of positive
integers.')
    end
  end
  s = cell2mat(s);
end
if (~isa(s,'double')) | ~isreal(s) | (size(s,1) ~= 1) | any(s<1)
| any(round(s) ~= s)
  error('Layer sizes is not a row vector of positive integers.')
end

% More defaults
Nl = length(s);
if nargin < 3, tf = {'tansig'}; tf = [tf(ones(1,Nl))]; end

% Architecture
net = network(1,Nl);
net.biasConnect = ones(Nl,1);
net.inputConnect(1,1) = 1;
[j,i] = meshgrid(1:Nl,1:Nl);
net.layerConnect = (j == (i-1));
net.outputConnect(Nl) = 1;
net.targetConnect(Nl) = 1;

% Simulation
net.inputs{1}.range = pr;
for i=1:Nl
  net.layers{i}.size = s(i);
  net.layers{i}.transferFcn = tf{i};
end

% Performance
net.performFcn = pf;

% Adaption
net.adaptfcn = 'trains';
net.inputWeights{1,1}.learnFcn = blf;
for i=1:Nl
  net.biases{i}.learnFcn = blf;
  net.layerWeights{i,:}.learnFcn = blf;
end

% Training
net.trainfcn = btf;

% Initialization
net.initFcn = 'initlay';
for i=1:Nl
  net.layers{i}.initFcn = 'initnw';
end
net = init(net);


function [a,e]=lpc(x,N)
```

```
%LPC  Linear Predictor Coefficients.
%   A = LPC(X,N) finds the coefficients, A=[ 1 A(2) ... A(N+1)
], of an Nth
%   order forward linear predictor.
%
%      Xp(n) = -A(2)*X(n-1) - A(3)*X(n-2) - ... - A(N+1)*X(n-N)
%
%   such that the sum of the squares of the errors
%
%      err(n) = X(n) - Xp(n)
%
%   is minimized.  X can be a vector or a matrix.  If X is a
matrix
%   containing a separate signal in each column, LPC returns a
model
%   estimate for each column in the rows of A.  N specifies the
order of
%   the polynomial A(z) which must be a positive integer.  N
must be less
%   or equal to the length of X.  If X is a matrix, N must be
less or equal
%   to the length of each column of X.
%
%   If you do not specify a value for N, LPC uses a default N =
%   length(X)-1.
%
%   [A,E] = LPC(X,N) returns the variance (power) of the
prediction error.
%
%   LPC uses the Levinson-Durbin recursion to solve the normal
equations
%   that arise from the least-squares formulation.  This
computation of the
%   linear prediction coefficients is often referred to as the
%   autocorrelation method.
%
%   See also LEVINSON, ARYULE, PRONY, STMCB.

%   Author(s): T. Krauss, 9-21-93
%   Modified:  T. Bryan 11-14-97
%   Copyright 1988-2004 The MathWorks, Inc.
%   $Revision: 1.12.4.4 $  $Date: 2004/12/26 22:16:18 $

error(nargchk(1,2,nargin))

if isempty(x)
    error('Input vector X should not be empty');
end

[m,n] = size(x);
if (n>1) && (m==1)
    x = x(:);
    [m,n] = size(x);
end

if nargin < 2,
```

```
    N = m-1;
elseif N < 0,
    % Check for N positive
    error(generatemsgid('negativeOrder'), ...
        'Order of the predictor should be a positive integer.');
end

if (N > m),
    error(generatemsgid('orderTooLarge'), '%s\n%s\n%s', ...
        'X must be a vector with length greater or equal to the
prediction order.', ...
        'If X is a matrix, the length of each column must be
greater or equal to', ...
        'the prediction order.');
end

% Compute autocorrelation vector or matrix
X = fft(x,2^nextpow2(2*size(x,1)-1));
R = ifft(abs(X).^2);
R = R./m; % Biased autocorrelation estimate

[a,e] = levinson(R,N);

% Return only real coefficients for the predictor if the input
is real
for k = 1:n,
    if isreal(x(:,k))
        a(k,:) = real(a(k,:));
    end
end

function [c,lags] = xcorr(x,varargin)
%XCORR Cross-correlation function estimates.
%   C = XCORR(A,B), where A and B are length M vectors (M>1), returns
%   the length 2*M-1 cross-correlation sequence C. If A and B are of
%   different length, the shortest one is zero-padded.  C will be a
%   row vector if A is a row vector, and a column vector if A is a
%   column vector.
%
%   XCORR produces an estimate of the correlation between two random
%   (jointly stationary) sequences:
%          C(m) = E[A(n+m)*conj(B(n))] = E[A(n)*conj(B(n-m))]
%   It is also the deterministic correlation between two deterministic
%   signals.
%
%   XCORR(A), when A is a vector, is the auto-correlation sequence.
%   XCORR(A), when A is an M-by-N matrix, is a large matrix with
%   2*M-1 rows whose N^2 columns contain the cross-correlation
%   sequences for all combinations of the columns of A.
%   The zeroth lag of the output correlation is in the middle of the
%   sequence, at element or row M.
%
%   XCORR(...,MAXLAG) computes the (auto/cross) correlation over the
%   range of lags:  -MAXLAG to MAXLAG, i.e., 2*MAXLAG+1 lags.
%   If missing, default is MAXLAG = M-1.
%
%   [C,LAGS] = XCORR(...)  returns a vector of lag indices (LAGS).
%
```

```
%   XCORR(...,SCALEOPT), normalizes the correlation according to
SCALEOPT:
%     'biased'   - scales the raw cross-correlation by 1/M.
%     'unbiased' - scales the raw correlation by 1/(M-abs(lags)).
%     'coeff'    - normalizes the sequence so that the auto-
correlations
%                  at zero lag are identically 1.0.
%     'none'     - no scaling (this is the default).
%
%   See also XCOV, CORRCOEF, CONV, COV and XCORR2.

%   Author(s): R. Losada
%   Copyright 1988-2004 The MathWorks, Inc.
%   $Revision: 1.16.4.2 $  $Date: 2004/12/26 22:17:15 $
%
%   References:
%     S.J. Orfanidis, "Optimum Signal Processing. An Introduction"
%     2nd Ed. Macmillan, 1988.

error(nargchk(1,4,nargin));

[x,nshift] = shiftdim(x);
[xIsMatrix,autoFlag,maxlag,scaleType,msg] = parseinput(x,varargin{:});
error(msg);

if xIsMatrix,
    [c,M,N] = matrixCorr(x);
else
    [c,M,N] = vectorXcorr(x,autoFlag,varargin{:});
end

% Force correlation to be real when inputs are real
c = forceRealCorr(c,x,autoFlag,varargin{:});


lags = -maxlag:maxlag;


% Keep only the lags we want and move negative lags before positive
lags
if maxlag >= M,
    c = [zeros(maxlag-M+1,N^2);c(end-M+2:end,:);c(1:M,:);zeros(maxlag-
M+1,N^2)];
else
    c = [c(end-maxlag+1:end,:);c(1:maxlag+1,:)];
end

% Scale as specified
[c,msg] =
scaleXcorr(c,xIsMatrix,scaleType,autoFlag,M,maxlag,lags,x,varargin{:})
;
error(msg);

% If first vector is a row, return a row
c = shiftdim(c,-nshift);

%-----------------------------------------------------------------
function [c,M,N] = matrixCorr(x)
% Compute all possible auto- and cross-correlations for a matrix input
```

```
[M,N] = size(x);

X = fft(x,2^nextpow2(2*M-1));

Xc = conj(X);

[MX,NX] = size(X);
C = zeros(MX,NX*NX);
for n =1:N,
    C(:,(((n-1)*N)+1):(n*N)) = repmat(X(:,n),1,N).*Xc;
end

c = ifft(C);

%--------------------------------------------------------------------
function [c,M,N] = vectorXcorr(x,autoFlag,varargin)
% Compute auto- or cross-correlation for vector inputs

x = x(:);

[M,N] = size(x);

if autoFlag,
    % Autocorrelation
    % Compute correlation via FFT
    X = fft(x,2^nextpow2(2*M-1));
    c = ifft(abs(X).^2);

else
    % xcorrelation
    y = varargin{1};
    y = y(:);
    L = length(y);

    % Cache the length(x)
    Mcached = M;

    % Recompute length(x) in case length(y) > length(x)
    M = max(Mcached,L);

    if (L ~= Mcached) && any([L./Mcached, Mcached./L] > 10),

        % Vector sizes differ by a factor greater than 10,
        % fftfilt is faster
        neg_c = conj(fftfilt(conj(x),flipud(y))); % negative lags
        pos_c = flipud(fftfilt(conj(y),flipud(x))); % positive lags

        % Make them of almost equal length (remove zero-th lag from
neg)
        lneg = length(neg_c); lpos = length(pos_c);
        neg_c = [zeros(lpos-lneg,1);neg_c(1:end-1)];
        pos_c = [pos_c;zeros(lneg-lpos,1)];

        c = [pos_c;neg_c];

    else
        if L ~= Mcached,
```

```
            % Force equal lengths
            if L > Mcached
                x = [x;zeros(L-Mcached,1)];

            else
                y = [y;zeros(Mcached-L,1)];
            end
        end

        % Transform both vectors
        X = fft(x,2^nextpow2(2*M-1));
        Y = fft(y,2^nextpow2(2*M-1));

        % Compute cross-correlation
        c = ifft(X.*conj(Y));
    end
end

%---------------------------------------------------------------------
function [c,msg] = scaleXcorr(c,xIsMatrix,scaleType,autoFlag,...
    M,maxlag,lags,x,varargin)
% Scale correlation as specified

msg = '';

switch scaleType,
case 'none',
    return
case 'biased',
    % Scales the raw cross-correlation by 1/M.
    c = c./M;
case 'unbiased',
    % Scales the raw correlation by 1/(M-abs(lags)).
    scale = (M-abs(lags)).';
    scale(scale<=0)=1; % avoid divide by zero, when correlation is
zero

    if xIsMatrix,
        scale = repmat(scale,1,size(c,2));
    end
    c = c./scale;
case 'coeff',
    % Normalizes the sequence so that the auto-correlations
    % at zero lag are identically 1.0.
    if ~autoFlag,
        % xcorr(x,y)
        % Compute autocorrelations at zero lag
        cxx0 = sum(abs(x).^2);
        cyy0 = sum(abs(varargin{1}).^2);
        scale = sqrt(cxx0*cyy0);
        c = c./scale;
    else
        if ~xIsMatrix,
            % Autocorrelation case, simply normalize by c[0]
            c = c./c(maxlag+1);
        else
            % Compute the indices corresponding to the columns for
which
            % we have autocorrelations (e.g. if c = n by 9, the
autocorrelations
```

```
            % are at columns [1,5,9] the other columns are cross-
correlations).
            [mc,nc] = size(c);
            jkl = reshape(1:nc,sqrt(nc),sqrt(nc))';
            tmp = sqrt(c(maxlag+1,diag(jkl)));
            tmp = tmp(:)*tmp;
            cdiv = repmat(tmp(:).',mc,1);
            c = c ./ cdiv; % The autocorrelations at zero-lag are
normalized to
            % one
        end
    end
end

%-----------------------------------------------------------------
-
function [xIsMatrix,autoFlag,maxlag,scaleType,msg] =
parseinput(x,varargin)
%    Parse the input and determine optional parameters:
%
%    Outputs:
%       xIsMatrix - flag indicating if x is a matrix
%       AUTOFLAG  - 1 if autocorrelation, 0 if xcorrelation
%       maxlag    - Number or lags to compute
%       scaleType - String with the type of scaling wanted
%       msg       - possible error message

% Set some defaults
scaleType = '';
autoFlag = 1; % Assume autocorrelation until proven otherwise
maxlag = [];

errMsg = 'Input argument is not recognized.';

switch nargin,
case 2,
    % Can be (x,y), (x,maxlag), or (x,scaleType)
    if ischar(varargin{1}),
        % Second arg is scaleType
        scaleType = varargin{1};

    elseif isnumeric(varargin{1}),
        % Can be y or maxlag
        if length(varargin{1}) == 1,
            maxlag = varargin{1};
        else
            autoFlag = 0;
            y = varargin{1};
        end
    else
        % Not recognized
        msg = errMsg;
        return
    end
case 3,
    % Can be (x,y,maxlag), (x,maxlag,scaleType) or
(x,y,scaleType)
```

```
    maxlagflag = 0; % By default, assume 3rd arg is not maxlag
    if ischar(varargin{2}),
        % Must be scaletype
        scaleType = varargin{2};

    elseif isnumeric(varargin{2}),
        % Must be maxlag
        maxlagflag = 1;
        maxlag = varargin{2};

    else
        % Not recognized
        msg = errMsg;
        return
    end

    if isnumeric(varargin{1}),
        if maxlagflag,
            autoFlag = 0;
            y = varargin{1};
        else
            % Can be y or maxlag
            if length(varargin{1}) == 1,
                maxlag = varargin{1};
            else
                autoFlag = 0;
                y = varargin{1};
            end
        end
    else
        % Not recognized
        msg = errMsg;
        return
    end

case 4,
    % Must be (x,y,maxlag,scaleType)
    autoFlag = 0;
    y = varargin{1};

    maxlag = varargin{2};

    scaleType = varargin{3};
end

% Determine if x is a matrix or a vector
[xIsMatrix,m] = parse_x(x);



if ~autoFlag,
    % Test y for correctness
    [maxlag,msg] = parse_y(y,m,xIsMatrix,maxlag);
    if ~isempty(msg),
        return
    end
```

```
end

[maxlag,msg] = parse_maxlag(maxlag,m);
if ~isempty(msg),
    return
end


% Test the scaleType validity
[scaleType,msg] =
parse_scaleType(scaleType,errMsg,autoFlag,m,varargin{:});
if ~isempty(msg),
    return
end


%-------------------------------------------------------------------
----
function [xIsMatrix,m] = parse_x(x)


xIsMatrix = (size(x,2) > 1);

m = size(x,1);


%-------------------------------------------------------------------
----
function [maxlag,msg] = parse_y(y,m,xIsMatrix,maxlag)
msg = '';
[my,ny] = size(y);
if ~any([my,ny] == 1),
    % Second arg is a matrix, error
    msg = 'B must be a vector (min(size(B))==1).';
    return
end

if xIsMatrix,
    % Can't do xcorr(matrix,vector)
    msg = 'When B is a vector, A must be a vector.';
    return
end
if (length(y) > m) && isempty(maxlag),
    % Compute the default maxlag based on the length of y
    maxlag = length(y) - 1;
end

%-------------------------------------------------------------------
----
function [maxlag,msg] = parse_maxlag(maxlag,m)
msg = '';
if isempty(maxlag),
    % Defaul hasn't been assigned yet, do so
    maxlag = m-1;
else
    % Test maxlag for correctness
```

```
    if  length(maxlag)>1
        msg = 'Maximum lag must be a scalar.';
        return
    end
    if maxlag < 0,
        maxlag = abs(maxlag);
    end
    if maxlag ~= round(maxlag),
        msg = 'Maximum lag must be an integer.';
    end
end

%-------------------------------------------------------------------
-----
function c = forceRealCorr(c,x,autoFlag,varargin)
% Force correlation to be real when inputs are real

forceReal = 0; % Flag to determine whether we should force real
corr

if (isreal(x) && autoFlag) || (isreal(x) &&
isreal(varargin{1})),
    forceReal = 1;
end


if forceReal,
    c = real(c);
end

%-------------------------------------------------------------------
-----
function [scaleType,msg] =
parse_scaleType(scaleType,errMsg,autoFlag,m,varargin)
msg = '';
if isempty(scaleType),
    scaleType = 'none';
else
    scaleOpts = {'biased','unbiased','coeff','none'};
    indx = find(strncmpi(scaleType, scaleOpts,
length(scaleType)));

    if isempty(indx),
        msg = errMsg;
        return
    else
        scaleType = scaleOpts{indx};
    end

    if ~autoFlag && ~strcmpi(scaleType,'none') && (m ~=
length(varargin{1})),
        msg = 'Scale option must be ''none'' for different
length vectors A and B.';
        return
    end
end
```

```
% EOF


function [net,tr,Y,E,Pf,Af]=train(net,P,T,Pi,Ai,VV,TV)
%TRAIN Train a neural network.
%
%  Syntax
%
%    [net,tr,Y,E,Pf,Af] = train(NET,P,T,Pi,Ai,VV,TV)
%
%  Description
%
%    TRAIN trains a network NET according to NET.trainFcn and
%    NET.trainParam.
%
%    TRAIN(NET,P,T,Pi,Ai) takes,
%      NET - Network.
%      P   - Network inputs.
%      T   - Network targets, default = zeros.
%      Pi  - Initial input delay conditions, default = zeros.
%      Ai  - Initial layer delay conditions, default = zeros.
%      VV  - Structure of validation vectors, default = [].
%      TV  - Structure of test vectors, default = [].
%    and returns,
%      NET - New network.
%      TR  - Training record (epoch and perf).
%      Y   - Network outputs.
%      E   - Network errors.
%      Pf  - Final input delay conditions.
%      Af  - Final layer delay conditions.
%
%    Note that T is optional and need only be used for networks
%    that require targets.  Pi and Pf are also optional and need
%    only be used for networks that have input or layer delays.
%    Optional arguments VV and TV are described below.
%
%    TRAIN's signal arguments can have two formats: cell array
or matrix.
%
%    The cell array format is easiest to describe.  It is most
%    convenient for networks with multiple inputs and outputs,
%    and allows sequences of inputs to be presented:
%      P  - NixTS cell array, each element P{i,ts} is an RixQ
matrix.
%      T  - NtxTS cell array, each element P{i,ts} is an VixQ
matrix.
%      Pi - NixID cell array, each element Pi{i,k} is an RixQ
matrix.
%      Ai - NlxLD cell array, each element Ai{i,k} is an SixQ
matrix.
%      Y  - NOxTS cell array, each element Y{i,ts} is an UixQ
matrix.
%      E  - NtxTS cell array, each element P{i,ts} is an VixQ
matrix.
```

```
%      Pf - NixID cell array, each element Pf{i,k} is an RixQ
matrix.
%      Af - NlxLD cell array, each element Af{i,k} is an SixQ
matrix.
%    Where:
%      Ni = net.numInputs
%      Nl = net.numLayers
%      Nt = net.numTargets
%      ID = net.numInputDelays
%      LD = net.numLayerDelays
%      TS = number of time steps
%      Q  = batch size
%      Ri = net.inputs{i}.size
%      Si = net.layers{i}.size
%      Vi = net.targets{i}.size
%
%    The columns of Pi, Pf, Ai, and Af are ordered from the
oldest delay
%    condition to most recent:
%      Pi{i,k} = input i at time ts=k-ID.
%      Pf{i,k} = input i at time ts=TS+k-ID.
%      Ai{i,k} = layer output i at time ts=k-LD.
%      Af{i,k} = layer output i at time ts=TS+k-LD.
%
%    The matrix format can be used if only one time step is to
be
%    simulated (TS = 1).  It is convenient for network's with
%     only one input and output, but can be used with networks
that
%     have more.
%
%    Each matrix argument is found by storing the elements of
%    the corresponding cell array argument into a single matrix:
%      P  - (sum of Ri)xQ matrix
%      T  - (sum of Vi)xQ matrix
%      Pi - (sum of Ri)x(ID*Q) matrix.
%      Ai - (sum of Si)x(LD*Q) matrix.
%      Y  - (sum of Ui)xQ matrix.
%      E  - (sum of Vi)xQ matrix
%      Pf - (sum of Ri)x(ID*Q) matrix.
%      Af - (sum of Si)x(LD*Q) matrix.
%
%    If VV and TV are supplied they should be an empty matrix []
or
%    a structure with the following fields:
%      VV.P,  TV.P  - Validation/test inputs.
%      VV.T,  TV.T  - Validation/test targets, default = zeros.
%      VV.Pi, TV.Pi - Validation/test initial input delay
conditions, default = zeros.
%      VV.Ai, TV.Ai - Validation/test layer delay conditions,
default = zeros.
%    The validation vectors are used to stop training early if
further
%    training on the primary vectors will hurt generalization to
the
```

```
%    validation vectors.  Test vector performance can be used to
measure
%    how well the network generalizes beyond primary and
validation vectors.
%    If VV.T, VV.Pi, or VV.Ai are set to an empty matrix or cell
array,
%    default values will be used. The same is true for TV.T,
TV.Pi, TV.Ai.
%
%    Not all training functions support validation and test
vectors.
%    Those that do not ignore the VV and TV arguments.
%
%  Examples
%
%    Here input P and targets T define a simple function which
%    we can plot:
%
%      p = [0 1 2 3 4 5 6 7 8];
%      t = [0 0.84 0.91 0.14 -0.77 -0.96 -0.28 0.66 0.99];
%      plot(p,t,'o')
%
%    Here NEWFF is used to create a two layer feed forward
network.
%    The network will have an input (ranging from 0 to 8),
followed
%    by a layer of 10 TANSIG neurons, followed by a layer with 1
%    PURELIN neuron.  TRAINLM backpropagation is used.  The
network
%    is also simulated.
%
%      net = newff([0 8],[10 1],{'tansig' 'purelin'},'trainlm');
%      y1 = sim(net,p)
%      plot(p,t,'o',p,y1,'x')
%
%    Here the network is trained for up to 50 epochs to a error
goal of
%    0.01, and then resimulated.
%
%      net.trainParam.epochs = 50;
%      net.trainParam.goal = 0.01;
%      net = train(net,p,t);
%      y2 = sim(net,p)
%      plot(p,t,'o',p,y1,'x',p,y2,'*')
%
%  Algorithm
%
%    TRAIN calls the function indicated by NET.trainFcn, using
the
%    training parameter values indicated by NET.trainParam.
%
%    Typically one epoch of training is defined as a single
presentation
%    of all input vectors to the network.  The network is then
updated
%    according to the results of all those presentations.
```

```
%
%    Training occurs until a maximum number of epochs occurs,
the
%    performance goal is met, or any other stopping condition of
the
%    function NET.trainFcn occurs.
%
%    Some training functions depart from this norm by presenting
only
%    one input vector (or sequence) each epoch. An input vector
(or sequence)
%    is chosen randomly each epoch from concurrent input vectors
(or sequences).
%    NEWC and NEWSOM return networks that use TRAINR, a training
function
%    that does this.
%
%  See also INIT, REVERT, SIM, ADAPT

%  Mark Beale, 11-31-97
%  Copyright 1992-2004 The MathWorks, Inc.
%  $Revision: 1.11.4.2 $ $Date: 2004/08/17 21:42:13 $

% CHECK AND FORMAT ARGUMENTS
% --------------------------

if nargin < 2
  error('Not enough input arguments.');
end
if ~isa(net,'network')
  error('First argument is not a network.');
end
if net.hint.zeroDelay
  error('Network contains a zero-delay loop.')
end
switch nargin
  case 2
    [err,P,T,Pi,Ai,Q,TS,matrixForm] = trainargs(net,P);
  VV = [];
  TV = [];
  case 3
    [err,P,T,Pi,Ai,Q,TS,matrixForm] = trainargs(net,P,T);
  VV = [];
  TV = [];
  case 4
    [err,P,T,Pi,Ai,Q,TS,matrixForm] = trainargs(net,P,T,Pi);
  VV = [];
  TV = [];
  case 5
    [err,P,T,Pi,Ai,Q,TS,matrixForm] = trainargs(net,P,T,Pi,Ai);
  VV = [];
  TV = [];
  case 6
    [err,P,T,Pi,Ai,Q,TS,matrixForm] = trainargs(net,P,T,Pi,Ai);
  if isempty(VV)
    VV = [];
```

```
  else
    if ~hasfield(VV,'P'), error('VV.P must be defined or VV must
be [].'), end
    if ~hasfield(VV,'T'), VV.T = []; end
    if ~hasfield(VV,'Pi'), VV.Pi = []; end
    if ~hasfield(VV,'Ai'), VV.Ai = []; end
    [err,VV.P,VV.T,VV.Pi,VV.Ai,VV.Q,VV.TS,matrixForm] = ...
      trainargs(net,VV.P,VV.T,VV.Pi,VV.Ai);
      if length(err)
      disp('??? Error with validation vectors using ==> train')
      error(err)
    end
  end
  TV = [];
  case 7
    [err,P,T,Pi,Ai,Q,TS,matrixForm] = trainargs(net,P,T,Pi,Ai);
  if isempty(VV)
    VV = [];
  else
    if ~hasfield(VV,'P'), error('VV.P must be defined or VV must
be [].'), end
    if ~hasfield(VV,'T'), VV.T = []; end
    if ~hasfield(VV,'Pi'), VV.Pi = []; end
    if ~hasfield(VV,'Ai'), VV.Ai = []; end
    [err,VV.P,VV.T,VV.Pi,VV.Ai,VV.Q,VV.TS,matrixForm] = ...
      trainargs(net,VV.P,VV.T,VV.Pi,VV.Ai);
      if length(err)
      disp('??? Error with validation vectors using ==> train')
      error(err)
    end
  end
  if isempty(TV)
    TV = [];
  else
    if ~hasfield(TV,'P'), error('TV.P must be defined or TV must
be [].'), end
    if ~hasfield(TV,'T'), TV.T = []; end
    if ~hasfield(TV,'Pi'), TV.Pi = []; end
    if ~hasfield(TV,'Ai'), TV.Ai = []; end
    [err,TV.P,TV.T,TV.Pi,TV.Ai,TV.Q,TV.TS,matrixForm] = ...
      trainargs(net,TV.P,TV.T,TV.Pi,TV.Ai);
      if length(err)
      disp('??? Error with test vectors using ==> train')
      error(err)
    end
  end
end
if length(err), error(err), end

% TRAIN NETWORK
% -------------

% Training function
trainFcn = net.trainFcn;
if ~length(trainFcn)
  error('Network "trainFcn" is undefined.')
```

```
end

% Delayed inputs, layer targets
Pc = [Pi P];
Pd = calcpd(net,TS,Q,Pc);
Tl = expandrows(T,net.hint.targetInd,net.numLayers);

% Validation and Test vectors
doValidation = ~isempty(VV);
doTest = ~isempty(TV);
if (doValidation)
  VV.Pd = calcpd(net,VV.TS,VV.Q,[VV.Pi VV.P]);
  VV.Tl = expandrows(VV.T,net.hint.targetInd,net.numLayers);
end
if (doTest)
  TV.Pd = calcpd(net,TV.TS,TV.Q,[TV.Pi TV.P]);
  TV.Tl = expandrows(TV.T,net.hint.targetInd,net.numLayers);
end

% Train network
net = struct(net);
flatten_time = (net.numLayerDelays == 0) & (TS > 1) &
(~strcmp(trainFcn,'trains'));
if flatten_time
  Pd = seq2con(Pd);
  Tl = seq2con(Tl);
  if (doValidation)
    VV.Pd = seq2con(VV.Pd);
    VV.Tl = seq2con(VV.Tl);
    VV.Q = VV.Q*VV.TS;
    VV.TS = 1;
  end
  if (doTest)
    TV.Pd = seq2con(TV.Pd);
    TV.Tl = seq2con(TV.Tl);
    TV.Q = TV.Q*TV.TS;
    TV.TS = 1;
  end
  [net,tr,Ac,El] = feval(trainFcn,net,Pd,Tl,Ai,Q*TS,1,VV,TV);
  Ac = con2seq(Ac,TS);
  El = con2seq(El,TS);
else
  [net,tr,Ac,El] = feval(trainFcn,net,Pd,Tl,Ai,Q,TS,VV,TV);
end
net = class(net,'network');

% Network outputs, errors, final inputs
Y = Ac(net.hint.outputInd,net.numLayerDelays+[1:TS]);
E = El(net.hint.targetInd,:);
Pf = Pc(:,TS+[1:net.numInputDelays]);
Af = Ac(:,TS+[1:net.numLayerDelays]);

% FORMAT OUTPUT ARGUMENTS
% ----------------------

if (matrixForm)
```

```
  Y = cell2mat(Y);
  E = cell2mat(E);
  Pf = cell2mat(Pf);
  Af = cell2mat(Af);
end

% CLEAN UP TEMPORARY FILES
% ------------------------
nntempdir=fullfile(tempdir,'matlab_nnet');
if exist(nntempdir)
    rmpath(nntempdir);
    tf=dir(nntempdir);
    for i=1:length(tf)
        if (~tf(i).isdir)
          delete(fullfile(nntempdir,tf(i).name));
        end
    end
    rmdir(nntempdir)
end

% ===============================================================
function [s2] = expandrows(s,ind,rows)

s2 = cell(rows,size(s,2));
s2(ind,:) = s;

% ===============================================================
function [err,P,T,Pi,Ai,Q,TS,matrixForm] =
trainargs(net,P,T,Pi,Ai);

err = '';

% Check signals: all matrices or all cell arrays
% Change empty matrices/arrays to proper form
switch class(P)
  case 'cell', matrixForm = 0; name = 'cell array'; default =
{};
  case {'double','logical'}, matrixForm = 1; name = 'matrix';
default = [];
  otherwise, err = 'P must be a matrix or cell array.'; return
end
if (nargin < 3)
  T = default;
elseif ((isa(T,'double')|isa(T,'logical')) ~= matrixForm)
  if isempty(T)
    T = default;
  else
    err = ['P is a ' name ', so T must be a ' name ' too.'];
    return
  end
end
if (nargin < 4)
  Pi = default;
elseif ((isa(Pi,'double')|isa(Pi,'logical')) ~= matrixForm)
  if isempty(Pi)
    Pi = default;
```

```
  else
    err = ['P is a ' name ', so Pi must be a ' name ' too.'];
    return
  end
end
if (nargin < 5)
  Ai = default;
elseif ((isa(Ai,'double')|isa(Ai,'logical')) ~= matrixForm)
  if isempty(Ai)
    Ai = default;
  else
    err = ['P is a ' name ', so Ai must be a ' name ' too.'];
    return
  end
end

% Check Matrices, Matrices -> Cell Arrays
if (matrixForm)
  [R,Q] = size(P);
  TS = 1;
  [err,P] = formatp(net,P,Q); if length(err), return, end
  [err,T] = formatt(net,T,Q,TS); if length(err), return, end
  [err,Pi] = formatpi(net,Pi,Q); if length(err), return, end
  [err,Ai] = formatai(net,Ai,Q); if length(err), return, end

% Check Cell Arrays
else
  [R,TS] = size(P);
  [R1,Q] = size(P{1,1});
  [err] = checkp(net,P,Q,TS); if length(err), return, end
  [err,T] = checkt(net,T,Q,TS); if length(err), return, end
  [err,Pi] = checkpi(net,Pi,Q); if length(err), return, end
  [err,Ai] = checkai(net,Ai,Q); if length(err), return, end
end

% ============================================================
```