UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

U.A.C.P. y P. I.I.M.A.S.

DESARROLLO DE UN RECONOCEDOR DE VOZ PUBLICO DE PALABRAS AISLADAS

T E S I S

QUE PARA OBTENER EL GRADO DE:
MAESTRO EN CIENCIAS DE LA COMPUTACION
P R E S E N T A :
DANIEL KORNHAUSER EISENBERG

DIRECTOR: DR. JESUS SAVAGE CARMONA

MEXICO, D. F., AGOSTO DE [redacted]

Reconocimientos

- A Adrián, Luis, Rose, por su comprensión y mantener el silencio mientras le hablaba a la computadora.

- Al Dr. Jesús Savage Carmona, director de esta tesis e impulsor de una educación de *calidad* en la Facultad de Ingeniería.

- Al Dr. Fabian Garcia Nocetti y al personal de D.I.S.C.A por su apoyo y por su paciencia a lo largo de esta tesis.

- A Dr. Carlos Rivera Rivera, M. en I. Abel Herrera Camacho y Dr. David Rosenblueth Laguette por aceptar ser sinodales de esta tesis.

- A la Dra. María Garza por su gran apoyo al permitirme usar el laboratorio de electrónica de la Maestría en Ciencias de la Computación y una computadora.

- A la Dra. Hanna Oktaba por su orientación en la maestría.

- A Carlos Velarde Velázquez y Fernando Magarinhos Lamas por su asesoría en LaTeX.

- Al personal de la biblioteca por su *generosidad* en el préstamo de libros.

- A la comunidad de software libre.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Resumen

Se desarrolló la interfaz gráfica para un sistema de reconocimiento de palabras aisladas. Para su codificación se utilizó la filosofía *GNU*[11] con el propósito de lograr una fácil reutilización y para promover el mejoramiento del programa gracias a la distribución pública del código fuente.

El reconocedor de palabras aisladas fue implementado en lenguaje de programación C en el Laboratorio de Interfaces Inteligentes (LII) de la Facutad de Ingeniería.. La interfaz gráfica fue escrita en lenguaje de programación Python con widgets [1] de un toolkitllamado *GNOME*[23] .

El propósito del reconocedor de comandos es proveer un sistema de reconocimiento de palabras aisladas que ejecute comandos cuando se digan palabras previamente entrenadas.

El desarrollo de la tesis se redactó en cinco capítulos.

En el primero se desglosan los antecedentes que se usaron para diseñar el software y para programar los algoritmos. El diseño del software se forja en torno a la filosofía de software libre que se explica en el segundo capítulo, donde se muestra la relevancia que tiene en este trabajo de tesis. La programación de los algoritmos se describe en el tercer capítulo, donde se muestran las diferentes técnicas de procesamiento digital de señales involucradas en el reconocimiento de palabras aisladas de esta tesis. Después se presenta el sistema de reconocimiento de voz en el cuarto capítulo, donde se documenta y se evalua su diseño sugiriendo como se podría mejorarlo. Finalmente en el quinto capítulo se presenta el desempeño del reconocedor de voz y se comenta sobre los resultados obtenidos.

[1] Objetos gráficos utilizado para crear una interfaz gráfica de usuario (ventanas, botones, etc)

1.2 Historia

Al inicio de 1999 cuando empecé la tesis de reconocimiento de voz, no se podía encontrar un reconocedor de voz público disponible en Internet con el cual se pudiera experimentar. Aunque existen varios proyectos de reconocimiento de voz en la WWW, suelen ser muy pequeños y estar mal documentados [5] o demasiado grandes y complicados para ser entendidos por alumnos con poca experiencia en el area [9].

Sin embargo en el Laboratorio de Interfaces Inteligentes (LII) se contaba con un reconocedor de palabras aisladas basado en algoritmos escritos por Andrés Buzo, Carlos Rivera y Jesus Savage [15] [13] [3] [4]. Este reconocedor de palabras aisladas funcionaba en tiempo real al ejecutarse una en una computadora Pentium, sin embargo su operación requiere de la ejecución de varios programas con diferentes opciones a través de la linea de comandos para entrenarlo. Por lo cúal se decidió añadirle una interfaz gráfica de usuario (GUI) para que usuarios interesados en el tema pudieran estudiar un reconocedor de voz sencillo.

La interfaz gráfica se hizo integrada a un desktop llamado GNOME de tal forma que un usuario no necesite capacitación alguna para utilizar el reconocedor de comandos.

El reconocedor de voz del LII se puede dividir en dos partes:

La primera parte se compone de un sistema de análisis espectral de una palabra dicha por el usuario. Este sistema extrae ciertas características de la señal mediante técnicas de procesamiento digital de señales. Posteriormente se comparan la características espectrales extraidas para poder reconocer la palabra.

En una segunda parte del reconocimiento de voz se puede mejorar la elección de de palabras reconocidas mediante reglas de sintaxis y/o semánticas del lenguaje que se quiera reconocer.

El sistema de reconocimiento de voz tratado en esta tesis aborda solo con la primera etapa del reconocimiento de palabras aisladas.

1.3 Objetivos

Los objetivos principales del trabajo son:

Proveer documentación del reconocedor de LII: Se describirá cómo se usan los programas que componen el reconocedor de comandos del LII concentrándose en la técnica de procesamiento digital de señales para reconocimiento de voz . De esta forma se piensa divulgar una técnica de reconocimiento de voz para promover el desarrollo en este campo.

Encapsular las rutinas de reconocimiento de voz del LII: Este proceso consistirá en actualizar las rutinas de reconocimiento de voz para que se pueden accesar desde las interfaces gráficas

Programar una interfaz gráfica para manejar el reconocedor de voz: Integrar las rutinas de entrenamiento y reconocimiento con el *toolkit* de GNOME de tal forma que se puedan manejar fácilmente.

Capítulo 2

Desarrollo del Software Libre

Dado que esta tesis se centra en hacer un ejemplo que se pueda estudiar en una clase de reconocimiento de voz, la filosofía de programación fue central en el desarrollo del sistema. Se decidió de adoptar la filosofía de GNU dado que provee un marco legal que promueve el desarrollo y la distribución de este software. Cabe mencionar que existen otras filosofias de software libre. Sin embargo se optó por adoptar GNU ya que la filosofía GNU es la que tiene el ambiente GNOME.

2.1 Filosofía de GNU

La filosofía de GNU propone que el software se registre bajo una licencia *Copyleft* que contiene las cláusulas necesarias para que :

- Se tenga la libertad de estudiar cómo funciona el programa y adaptarlo a las necesidades que se tengan.
- Se tenga la libertad de redistribuir copias para poder compartirlo con quien se desee.
- La libertad de mejorar el programa, condicionada a liberar éstas mejoras al público de tal forma que toda la comunidad se pueda beneficiar de estas.

2.1.1 Reutilización

La licencia GNU promueve la reutilización de los programas gracias a la distribución obligada del código fuente. Al proveer el código fuente en el Internet se cuenta inmediatamente con una gran base de programadores que comparten la filosofía de GNU. En el caso específico del reconocedor de voz se espera encontrar programadores de universidades que estén interesados en estudiar el

código. Estos programadores podrían poseer las bases teóricas para entenderlo si cursaron o están cursando alguna materia que tenga que ver con reconocimiento de voz. Por otro lado el código podría mejorar sustancialmente gracias a los reportes de fallas y correcciones de esta base de programadores.

Finalmente dado que el código fuente es el único medio por el cual se comunican los programadores, este debe ser obligadamente comprensible y reutilizable.

2.1.2 Importancia didáctica

No se divulga mucho código de reconocimiento de voz debido a la discrecionalidad que se debe de mantener esta industria. Por lo tanto no existe una gran cantidad de material en la WWW que contenga programas que se puedan utilizar fácilmente con su código fuente. Se espera que al poner el código fuente disponible en la red junto con sus ejecutables se pueda dar la oportunidad a personas interesadas en el campo que estudia como se podría programar un reconocedor de palabras.

2.2 Requerimientos

2.2.1 Sistema operativo

Se decidió programar el reconocedor de comandos en el sistema operativo GNU/Linux, dado que se piensa utilizar herramientas GNU que son fácilmente portables entre sistemas UNIX, en particular un desktop llamado GNOME (GNU Network Object Model Environment). Por otro lado, se seleccionó el sistema operativo Linux para portar fácilmente este sistema de reconocimiento a las diferentes arquitecturas que soporta o hasta diferentes sistemas operativos que cumplan con los estándares Posix.

2.2.2 Lenguajes de programación

Se utilizaron dos lenguajes para programar el reconocedor de comandos.

Las rutinas de reconocimiento de palabras aisladas en tiempo real se programaron en C. El lenguaje C se seleccionó debido a las restricciones que impone un sistema de reconocimiento en tiempo real y dado que es uno de los lenguajes más portables entre compiladores y sistemas operativos.

El lenguaje Python se eligió para programar la interfaz gráfica que se integrará dentro del ambiente GNOME. Se seleccionó el lenguaje Python debido también a la gran facilidad y rapidez que presenta para codificar prototipos.

2.2.3 Ambiente GNOME

El ambiente GNOME es un *desktop manager* que provee una interfaz agradable y fácil de usar para el ambientes UNIX. GNOME utiliza el *toolkit Gtk*[2] para construir sus widgets de GUI.

Gracias a GNOME la interfaz gráfica del reconocedor de voz presenta un mismo *Look and Fell* que todas las aplicaciones asociadas al desktop de GNOME facilitando de esta forma el uso intuitivo del reconocedor. Además de la consistencia gráfica el ambiente GNOME establece una serie de vías bien definidas como *CORBA*[17] para que las aplicaciones se puedan comunicar entre sí. De esta manera se piensa que se podrá integrar fácilmente el reconocedor de voz para que pueda interactuar transparentemente con las aplicaciones de GNOME. (hojas electrónicas, procesadores de palabra, manejadores de bases de datos, etc...).

Es importante destacar que existe en México una base relativamente grande de desarrolladores de GNOME. Se espera que contribuyendo en este proyecto se pueda ampliar dicha base y contribuir al desarrollo de software en México.

2.3 Como programar software libre

Existen varios aspectos que se tienen que considerar cuando se desea programar software libre. El primero es tener una idea clara de lo que significa software libre. La programación de software libre surge a partir de una filosofía de una *comunidad* de software sustentada en un tipo de patente que lleva por nombre *Copyleft*. Esta patente sirve para proteger la comunidad de una distribución "inadecuada" del software. La definición exacta de una distribución "inadecuada" se describe en *GPL*[10] .

2.3.1 Como empezar un proyecto de Software libre

Antes de empezar un proyecto, uno debe investigar si existe ya alguien que esté trabajando en un proyecto equivalente dentro de la comunidad GNU. En el caso de que exista se tiene que evaluar cuidadosamente si es más productivo que uno empiece un nuevo proyecto o si convendría contribuir al proyecto en curso.

En el caso del reconocimiento de voz existían varios proyectos, desde proyectos gigantescos como el del *ISIP (Institute for Signal and Information Processing)*[9] hasta proyectos pequeños que se pueden encontrar en varios recopilaciones de *programas de reconocimiento de voz*[5].

Después de una evaluación de los proyectos que se encontraron en el Internet se concluyó que era más productivo utilizar el reconocedor de voz del LII que contribuir a algún proyecto en curso. Se llegó a esta conclusión dado que los proyectos no cumplían los objetivos señalados anteriormente.

Los proyectos gigantescos como el del ISIP no cumplían el objetivo de ser divulgativo. Esto se debe a que no se podrían estudiar en un tiempo razonable su código ni la teoría que necesita el reconocimiento de amplio vocabulario por sílabas. Además, este proyecto se encontraba en una fase experimental ya que su objetivo de crear un reconocedor de vocabulario amplio basado reconocimiento de sílabas es muy ambicioso.

Los otros proyectos de reconocimiento de voz pequeños tampoco cubrían ninguna tarea de divulgación ya que estaban usualmente mal documentados

y sin ningún mantenimiento, por lo que habría sido muy difícil retomarlos. Además, la mayoría de estos reconocedores estaban programados en el lenguaje C++. El lenguaje C++ no se consideró adecuado para el proyecto ya que no debería ser necesario saber programar C++ orientado objetos para entender las rutinas de reconocimiento de voz. Se recuerda que este proyecto tiene algún fin didáctico de ejemplificar como se codifica un reconocedor de voz simple. Dado que el lenguaje más difundido es C, éste se seleccionó para programar los algoritmos.

2.3.2 Selección de tecnologías de programación

Para codificar el reconocedor de voz se seleccionaron dos lenguajes, C y Python.

El reconocedor de voz fue implementado en lenguaje C debido a su rapidez en la ejecución. Era indispensable un lenguaje rápido para implementar los algoritmos de procesamiento digital de señales en tiempo real.

El lenguaje Python se empleó para programar el *front-end* del reconocedor de voz. Este incluye componentes como la interfaz gráfica al usuario, la persistencia de la configuración y la intercomunicación con otras aplicaciones. Python se seleccionó debido a la rapidez en que se puede implementar prototipos en este lenguaje y la facilidad con la que se puede migrar a C en caso de que sea necesario un mejor desempeño. En caso de que se requiera migrar a C la migración no debe presentar problema alguno ya que GNOME esta codificado en C.

El ambiente GNOME es crucial en el desarrollo del reconocedor de palabras aisladas ya que permitiría comunicar fácilmente diferentes aplicaciones a través de CORBA. Debe de quedar muy claro que el uso principal de el reconocedor de palabras aisladas tratado en esta tesis es transmitir comandos a la computadora y no dictarle algún texto. Por lo tanto el reconocedor de palabras aisladas debe tener una interfaz sencilla y bien definida para comunicarse a la aplicación que se dirige el usuario, GNOME provee esta interfaz a través de CORBA. Por otro lado GNOME provee un ambiente gráfico consistente gracias a *toolkit* gráfico GTK. Esto permite que el usuario pueda utilizar sus componentes intuitivamente ya que todos los widgets tiene un comportamiento similar independientemente de la aplicación donde se utilicen.

2.3.3 Codificación

Una buena codificación es indispensable para hacer software público. Como no existe ninguna documentación sobre el diseño del software, toda la documentación de un proyecto GNU se encuentra inicialmente en el código acompañado de un simple archivo de descripción del software. Sin embargo si el código es fácilmente comprensible el proyecto va recibiendo cada vez más contribuciones donde finalmente se incluye un proyecto de documentación.

2.3.4 Documentación

Considero la documentación una de las partes más importante de este proyecto, dado que los proyectos que se encontraron en la WWW [5] no estaban bien documentados. Esto constituye un impedimento para el desarrollo de software libre en el área de reconocimiento de voz ya que la teoría matemática que involucra el procesamiento digital de voz es muy amplia. Por lo tanto, creo que es necesario señalar qué conocimientos son necesarios para que un programador pueda empezar a desarrollar un reconocedor de voz. Por lo tanto a continuación se incluye un capítulo de procesamiento digital de señales donde se presentan algunos de los algoritmos que podrían conformar un reconocedor de palabras aisladas.

Capítulo 3

Procesamiento Digital de Señales

3.1 Introducción

En este capítulo se presentarán los algoritmos de procesamiento digital de señales que componen el reconocedor de palabras aisladas usado en esta tesis. En cada sección se explicará brevemente el algoritmo, después se presentará su programación en Matlab y en C.

Si se desea leer el desarrollo detallado del esquema de reconocimiento de voz se recomienda consultar el capítulo 3 de la tesis de maestría del Dr. Jesús Savage[3]. También se recomienda el libro Fundamentals of Speech Recognition [18] de Rabiner y Huang si surge alguna duda en este capítulo.

Para efectuar reconocimiento de palabras aisladas con el sistema de esta tesis se entrenan las palabras que se quiera reconocer en un proceso que llamaremos *entrenamiento*. Sólo después del entrenamiento se podrá llevar a cabo el *reconocimiento* en tiempo real. Estos dos procesos están basados en el *modelo de producción de voz* presentado a continuación.

3.2 Modelo del aparato vocal

Antes de poder representar paramétricamente la voz mediante predicción lineal surge la necesidad de modelar la producción de voz matemáticamente. Esto se logra mediante un modelo de la producción de voz propuesto por Fant[7] y Flanagan[8] que se puede observar en la figura 3.1.

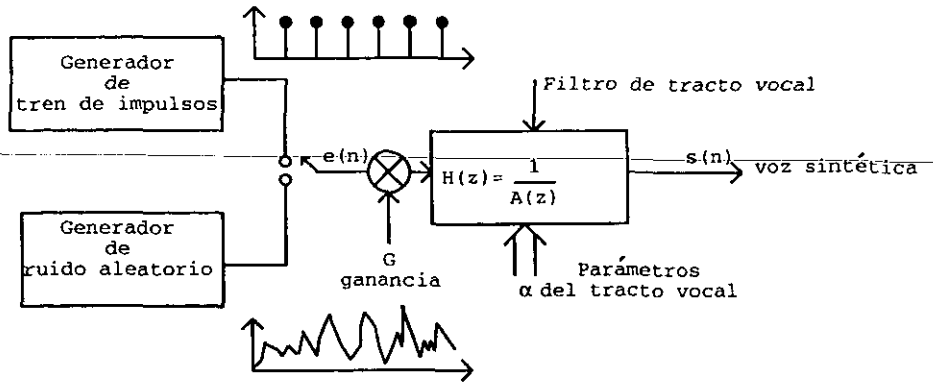


Figura 3.1: Modelo de producción de voz para la codificación por predicción lineal

El modelo de producción de voz presentado a continuación permitirá abstraer un sonido de voz a un conjunto de coeficientes. El sistema de producción de voz se puede dividir en dos partes. La primera parte se puede considerar la excitación de un sistema $S(z)$ que puede ser un tren de pulsos o ruido aleatorio. La segunda parte es modulación del espectro por el trato vocal que se puede representar por un filtro digital $U(z)$ variando en el tiempo.

Se puede representar este sistema con la siguiente función de transferencia:

$$H(z) = \frac{S(z)}{U(z)} = \frac{G}{1 - \sum_{k=1}^{p} a_k z^{-k}} \tag{3.1}$$

Este sistema es excitado por un tren de impulsos cuando se desea generar un sonidos sonoros o por ruido aleatorio cuando se desea generar sonidos sordos. Los parámetros de este sistema son: sonidos producidos con excitación por tren de impulsos o por ruido aleatorio, periodo que determina el tono en los sonidos generados con un tren de impulsos, parámetro de ganancia G y coeficientes $\{a_k\}$ del filtro digital. Es importante notar que estos parámetros varían lentamente con el tiempo con un muestreo a una velocidad adecuada.

Se observa que se pueden representar satisfactoriamente casi todos los sonidos de la voz si se utiliza un filtro compuesto solamente por polos siempre que se tenga un orden suficientemente alto. La ventaja más importante de este modelo es que el parámetro de ganancia G, y los coeficientes del filtro $\{a_k\}$ se pueden estimar con algoritmos relativamente sencillos y de una forma computacionalmente eficiente gracias al análisis de predicción lineal.

En el caso de el sistema presentado en la figura 3.1, la generación de la voz sintética $s(n)$ esta relacionada a la excitación $u(n)$ por una ecuación en diferencias.

$$s(n) = \sum_{k=1}^{p} a_k s(n-k) + Gu(n) \tag{3.2}$$

3.3 Entrenamiento

En el entrenamiento el usuario graba cada palabra que quiere reconocer varias veces. Posteriormente se calcula un *modelo* de las palabras que se quieren reconocer por medio de técnicas de procesamiento digital de señales.

Se presenta en la figura 3.2 un diagrama de bloques del entrenamiento de palabras. Este diagrama contiene las técnicas de preprocesamiento y análisis [15] que se emplearon en reconocedor del LII (cabe notar que existen muchas otras tecnicas de preprocesamiento y análisis).

Se puede observar que cada bloque contiene el nombre del procesamiento que se le aplica a la señal y el nombre de la rutina que hace ese procesamiento en Matlab en `fuente de teletipo`. A continuación se desglosará cada bloque presentando su función dentro del entrenamiento y se ilustrará con su codificación en Matlab y en C.

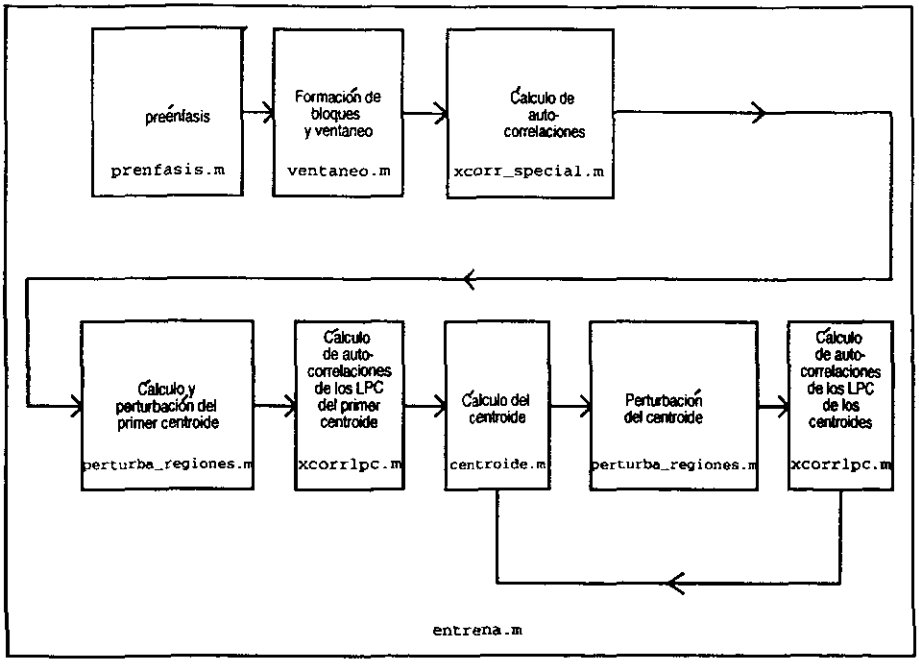


Figura 3.2: Diagrama de bloques del algoritmo de entrenamiento

3.3.1 Preénfasis

El preénfasis es un preprocesamiento sobre la señal que permite resaltar algunas características de la señal que son atenuadas por un transductor como el micrófono. Los micrófonos tienden a tener una mejor respuesta en frecuencias bajas, por lo tanto el preénfasis nos permite aumentar las frecuencias agudas

donde se encuentra gran parte de la información auditiva que se utiliza para reconocer una palabra.

Definición y función del preénfasis

Para hacer el procesamiento de la señal de voz digitalizada menos susceptible a truncamientos, la señal se aplana espectralmente. Esto se logra pasando la señal a través de un filtro paso-bajas de primer orden. Este filtro puede tener coeficientes fijos o ser adaptativo. En el caso de este reconocedor de voz se decidió que los coeficientes serían constantes. Entonces se obtiene sistema de primer orden:

$$H(z) = 1 - \tilde{a}z^{-1}, \qquad 0.9 \leq a \leq 1.0 \tag{3.3}$$

En este caso la salida del sistema de preénfasis, $\tilde{s}(n)$, está relacionada a la entrada del sistema, $s(n)$, por la siguiente ecuación:

$$\tilde{s}(n) = s(n) - \tilde{a}s(n-1) \tag{3.4}$$

Para nuestra implementación seleccionamos $\tilde{a} = 0.97$.

Programación del preénfasis en Matlab

Antes de programar el algoritmo en un lenguaje eficiente se tiene que comprobar que se comprende plenamente el algoritmo. Por lo cual antes de programar en C, se debería programar el algoritmo en un lenguaje que permita programar un prototipo rápidamente y comprobar fácilmente su buen funcionamiento. Usualmente estos lenguajes tiene una ejecución muy lenta pero permiten verificar el buen funcionamiento del algoritmo. A continuación se presenta un programa de Matlab para poder comprobar que el preénfasis esta filtrando la señal de la manera prevista.

En Matlab el algoritmo se podría programar de la siguienta manera:

```
1  function s = prenfasis(y)
2  %Funcion que le aplica el preénfasis a una senal
3  %Entrada : Senal de entrenamiento representada por un vector
4  %          de 1xn.
5  %Salida : Senal de entrenamiento procesada por un filtro paso
6  %         altas representada por un vector de 1xn.
7  %
8  %Descripción: Este programa pasa los archivos de entrenamiento
9  %            a través de un filtro pasa altas para que las
10 %            frecuencias altas que tienen menor amplitud no
11 %            sean despreciadas, cuando se procese el codebook.
12 for n=1:(length(y)-1)
13    s(n+1)=y(n+1)-0.9375*y(n);
14 end
```

En la figura 3.3 generada con el *Signal Processing Toolbox* de Matlab se

muestra un espectrograma formado a partir de una señal de voz grabada digitalmente en formato wav a 8 KHzdonde se dijo *uno*. El espectrograma de la izquierda se observa la señal intacta mientras que en el de la derecha se puede ver la señal después de aplicarle un filtro de preénfasis.

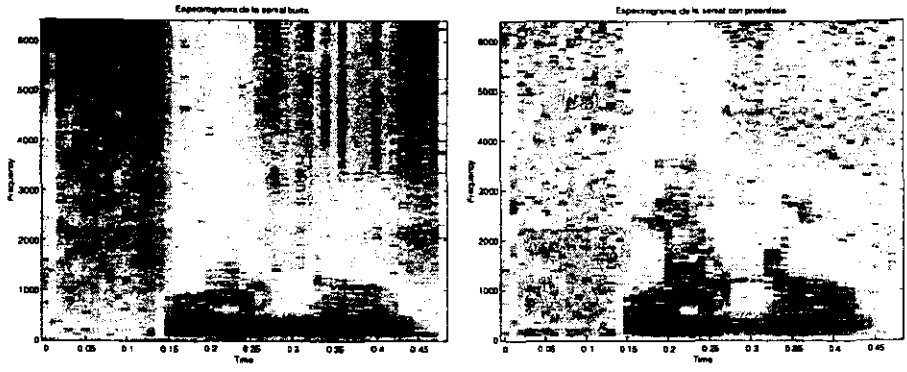


Figura 3.3: Espectrograma de una señal antes y después del preénfasis

Se puede observar que en la figura izquierda se tiene los agudos (parte superior de la gráfica) más atenuados (más claros), mientras que en la figura derecha los agudos tienen una mayor magnitud (son más obscuros).

Por otro lado observa claramente en la figura 3.4 que sistema de preénfasis actua como un filtro paso altas.

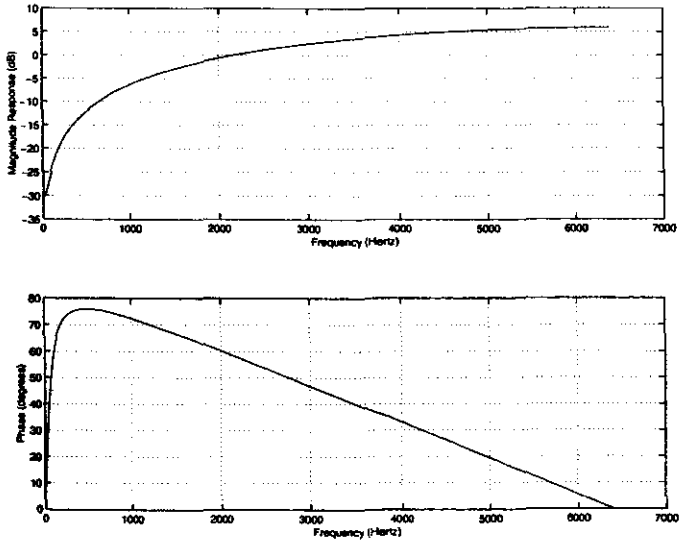


Figura 3.4: Respuesta de un filtro de preénfasis con un coeficiente a=0.97 un muestreo en 12800 Hz

Por consiguiente el filtro aumentó los niveles de las frecuencias agudas para que no sean ignoradas cuando se calculen los coeficientes de los modelos.

Programación del preénfasis en C

El programa en C es una función muy simple la cual recibe los datos de la señal en un apuntador x, y los devuelve procesados en un apuntador y.

```
1  /* Aplica un filtro de preénfasis a los datos */
2  preemfasis(x,y)
3  float x[],y[];
4  {int j;
5   for(j=1;j<WINDOW_SIZE+1;j++){
6       y[j]=x[j]-x[j-1]*PREEM_CONST;
7   }
8  }
```

3.3.2 Formación de bloques y ventaneo

La primera parte del reconocimiento de voz consiste en obtener ciertas características espectrales de la señal de voz a reconocer. Para lograr este fin utilizamos el método de Codificación de Predicción Lineal (LPC) que permite extraer ciertas características espectrales del sonido que queremos identificar .

Sin embargo, este método sólo se puede aplicar a señales estacionarias, lo cual no se cumple directamente en el caso de la señal de voz digitalizada. Por lo tanto, se debe dividir la señal de voz en pequeños segmentos cuasi-estacionarios llamados ventanas antes de calcular el LPC de un sonido. Le llamaremos a este proceso *formación de bloques y ventaneo.*

Antes de ventanear la señal se forman bloques compuestos de N muestras, donde los bloques adyacentes se separan de M muestras. Normalmente se toma $M < N$ para obtener estimaciones suaves de LPC de entre ventana y ventana.

La segmentación de la señal $s(m)$ se logra multiplicándola por otra señal, la ventana $w(m)$, que toma el valor de cero fuera del intervalo de donde queremos extraer la señal. Pero se evita normalmente extraer la señal simplemente multiplicando por cero todo lo que se encuentra fuera de nuestro intervalo de interés, ya que esto distorsiona el espectro en frecuencia. Para prevenir esta distorsión en frecuencia multiplicamos nuestra señal por una ventana de Hamming dada por:

$$w(m) = 0.54 - 0.46cos\frac{2\pi m}{N}, \qquad 0 \leq m \leq N-1, \qquad w(m) = 0, \qquad m < 0 \tag{3.5}$$

donde N es el tamaño de la ventana deseado en tiempo.
De esta forma obtenemos la señal s_n según la siguiente ecuación:

$$s_n(m) = s(n+m)w(m) \tag{3.6}$$

donde n es es desplazamiento de la ventana y m es el índice de las muestras .

Programación del algoritmo ventaneo en Matlab

En Matlab el algoritmo se podría programar de la siguienta manera:

```
1   function [NumVentanas,xw]= ventaneo (s,Fm)
2
3  %Entrada: - Senal preprocesada en un vector 1xn
4  %         - Frecuencia de muestreo en un vector 1x1
5  %Salida: - Senal translapada y ventaneada en un vector de 1xn
6  %         - Numero de ventanas del cual esta compuesta esta
7  %           senal en un vector de 1xn
8  %Descripción:
9  % En el ventaneo se divide la senal en bloques con el
10 % translape deseado y se pasa por una ventana de hamming.
11 Segundos_por_ventana = 0.02;
12 % Se calcula el numero de muestras para una ventana de 20ms
13 NumMuestras=fix(Fm*Segundos_por_ventana);
14 % Num de ventanas que caben en la senal
15 NumVentanas=fix(length(s)/NumMuestras);
16 % Numero de muestras por traslape
17 Translape=fix(NumMuestras/2);
18 % Espacio que se avanza cada ventaneo
19 Avance = NumMuestras-Translape;
20 % Indicador acumulativo que indica el inicio del sig. ventaneo
21 inicio = 0; k=1;
22
23 for l=1:(2*NumVentanas-1),
24   %mult c/vent de la senal p/la vent Hamming
25   for n=1:NumMuestras,
26     xw(k)=s(n+inicio)*(0.54 - 0.46*cos(6.2832*n/NumMuestras));
27     k = k + 1;
28     n+inicio;
29   end
30   inicio = inicio + Avance;
31 end
32
```

Programación de ventaneo en C

El ventaneo esta compuesto por dos funciones, una que multiplica la señal por la ventana, `apply_window(x,w)` y otra que se ocupa de mover el apuntador con

el traslape deseado concatenate_signal(x1,x2,x3).

```
1   /* Aplica una ventana a los datos */
2   apply_window(x,w)
3   float x[];
4   double w[];
5   {
6     int j;
7
8     for(j=1;j<WINDOW_SIZE+1;j++){
9       x[j]=x[j]*w[j];
10    }
```

```
1   /* Concatena la utima parte  y primera parte de x1 y x2 */
2   concatenate_signal(x1,x2,x3)
3   float x1[],x2[],x3[];
4   {
5     int i,j;
6     /* Pon muestra en la mitad del principio del sig. marco */
7     x3[0]=x1[OVERLAP_SIZE];
8
9     for(j=1,i=OVERLAP_SIZE+1;i<WINDOW_SIZE+1;j++,i++){
10     x3[j]=x1[i];
11    }
12    for(i=1,j=OVERLAP_SIZE+1;j<WINDOW_SIZE+1;j++,i++){
13      x3[j]=x2[i];
14    }
15  }
```

3.3.3 Autocorrelación

Después de ser ventaneada la señal es autocorelacionada para obtener

$$r_l(m) = \sum_{N-1-m}^{n=o} \tilde{x}_l(n)\tilde{x}_l(n+m), \quad m = 0, 1, ..., p,$$

donde el valor más alto de la autocorrelación, p, es el orden del análisis LPC.

Como programar el algoritmo de autocorrelación?

En Matlab el algoritmo se podría programar de la siguienta manera:

```
function region = \
                  xcorr_special(xw,num_ventanas,tam_ventana,grado)
%Entrada: - Senal translapada y ventaneada en un vector de 1xn
%        - Numero de ventanas del cual esta compuesta esta
%          senal en un vector de 1xn
%        - Tamaño de ventana en un vector de1x1
%        - Grado del vector de coeficientes de correlación que
%          se desea obtener
%Salida : - Matriz de nxm donde n es el número de ventanas
%          y m el número de coeficientes de correlación.
%Descripción:
% Se calcula la autocorelacion de  ventanas de toda la senal.
% En la matriz region de nxm ( donde n representa el número de
% ventanas y m el número de coeficientes de correlación) se
% almacenarán  los 15 primeros elementos de la correlación de
% todas las ventanas de todas las senales. (tam_ventana = 256;
% xcorr_grado = 15)
function region = xcorr_special(xw,num_ventanas,tam_ventana,\
                                grado)
% Se calcula los 15 primeros coeficientes de la coorelacion
% de las ventanas de 256 datos de todos loas vectores.
for ventana=0:num_ventanas*2-(11)
  % separacion de la ventana y correlacion de la venana
  muestra = xcorr(xw\
      (tam_ventana*ventana+1:tam_ventana*(ventana+1)),'coeff');
  % muestreo de los 15 elementos
  region(1,ventana+1,:) = \
        muestra(tam_ventana:tam_ventana+grado);

end
```

Programación de la autocorrelación en C

```
/* Calcula la funcion de autocorrelacion de la senal */
float calculate_autocorrelation(x,r)
float x[],r[];
{
  int i,j;

  for(i=1;i<=ORDER+1;i++){
    r[i]=0.;
    for(j=1;j<=WINDOW_SIZE-i+1;j++){
      r[i]=r[i]+x[j]*x[j+i-1];
```

```
11              i,r[i],j,x[j],j+i-1,x[j+i-1]);*/
12          }
13      }
14      return(r[1]);
15  }
```

3.3.4 Análisis de predicción lineal

Una de las mejores mejores técnicas de análisis de voz es el método de predicción lineal [16]. Este método se ha establecido como la técnica predominante para estimar los parámetros básicos de la voz (entonación, formantes, espectro, funciones de area del trato vocal). La importancia de este método radica en su habilidad de proveer estimaciones extremadamente precisas de los parámetros de la voz y su rápida computación. En esta sección presentaremos una formulación de los fundamentos de la predicción lineal.

La idea básica detrás del análisis de predicción lineal es que una muestra de voz se puede aproximar como una combinación lineal de muestras anteriores de voz. Lo anterior se logra minimizando la sumatoria del error cuadrático medio (en un intervalo finito) entre las muestras de la voz real y las muestras de la voz predecidas linealmente. Así se logra determinar un conjunto único de coeficientes predictores. (Los coeficientes predictores son los pesos que se les da a coeficientes usados en la combinación lineal que forma la predicción.)

La filosofía detrás de la predicción lineal surge a partir del modelo de síntesis de voz expuesto en la sección 3.2. Se mostró que la voz puede ser modelada como la salida de un sistema lineal variando en el tiempo excitado por trenes de pulsos o ruido aleatorio. La predicción lineal provee un método robusto para estimar los parámetros que caracterizan este sistema lineal variando en el tiempo.

Principios básicos del análisis de predicción lineal

Un predictor lineal con coeficientes de predicción, α_k se define como un sistema que tiene por salida:

$$\tilde{s}(n) = \sum_{k=1}^{p} \alpha_k s(s-k) \tag{3.7}$$

Por lo tanto la función de sistema de un predictor lineal de orden p, se puede escribir como el siguiente polinomio en z:

$$P(z) = \sum_{k=1}^{p} \alpha_k z^{-k} \tag{3.8}$$

Entonces si definimos el error de predicción $e(n)$, con la siguiente ecuación:

$$e(n) = s(n) - \tilde{s}(n) = s(n) - \sum_{k=1}^{p} \alpha_k s(s-k) \tag{3.9}$$

Se puede observar en la ecuación anterior que la secuencia del error de predicción es la salida de un sistema cuya función de transferencia es:

$$A(z) = 1 - \sum_{k=1}^{p} \alpha_k z^{-k} \tag{3.10}$$

Comparando (3.2) y (3.9), se nota que la señal de voz obedece el modelo (3.2), y si $\alpha_k = a_k$, entonces $e(n) = Gu(n)$. Por lo tanto, el *error del filtro de predicción* , $A(z)$, será un *filtro inverso* para el sistema, $H(z)$, obteniendo de (3.1):

$$H(z) = \frac{G}{A(z)} \tag{3.11}$$

El problema básico del análisis de predicción lineal es determinar un conjunto de coeficientes predictores $\{\alpha_k\}$ de la señal de voz de tal manera que se obtenga un buen estimado de la propiedades espectrales de la señal de voz usando (3.11). Como la voz es una señal que varía en el tiempo, los coeficientes predictores deben de ser continuamente estimados de segmentos cortos de voz. La forma básica de atacar este problema es encontrar un conjunto de coeficientes predictores que minimizarán el error cuadrático medio sobre un corto segmento de la onda de voz. Los coeficientes resultantes se asumirán como los parámetros de una función de sistema, $H(z)$, en el modelo de producción de voz.

El error cuadrático medio calculado en un corto tiempo se puede entonces definir como E_n. Donde el rango de las sumatorias de E_n se puede obtener considerando (3.9) y la ecuación del ventaneo aplicada a la señal definida en (3.6). Si $s_n(m)$ es diferente de cero solamente para $0 \leq m \leq M-1$, entonces el error de predicción correspondiente de $e_n(m)$, para un predictor de orden p puede ser diferente de cero en el intervalo $0 \leq m \leq N-1+p$. Por lo tanto, en este caso E_n se puede expresar como:

$$E_n = \sum_{m=0}^{N+p-1} e_n^2(m) \tag{3.12}$$

$$= \sum_{m=0}^{N+p-1} (s_n(m) - \tilde{s}_n(m))^2 \tag{3.13}$$

$$= \sum_{m=0}^{N+p-1} [s_n(m) - \sum_{k=1}^{p} \alpha_k s_n(m-k)]^2 \tag{3.14}$$

donde $s(n)$ es un segmento de voz que fue seleccionado en la vecindad de la muestra n, por ejemplo:

$$s_n(m) = s(m+n) \tag{3.15}$$

Para encontrar los valores de α_k, podemos minimizar E_n en (3.14) haciendo $\partial E_n / \partial \alpha_i = 0$, $i = 1, 2, ..., p$, obteniendo las siguientes ecuaciones:

$$\sum_{m=0}^{N+p-1} s_n(m-i)s_n(m) = \sum_{k=1}^{p} \hat{\alpha}_k \sum_{m} s_n(m-i)s_n(m-k) \quad 1 \leq i \leq p \tag{3.16}$$

donde $\hat{\alpha}_k$ son los valores de α_k que minimizan E_n. (Dado que $\hat{\alpha}_k$ es único dejaremos de usar el acento circunflejo y empezaremos a denotar α_k como los valores que minimizan E_n.)

Si definimos

$$\phi_n(i,k) = \sum_{m=0}^{N+p-1} s_n(m-i)s_n(m-k) \tag{3.17}$$

Entonces (3.16) se puede escribir como:

$$\sum_{k=1}^{p} \alpha_k \phi_n(i,k) = \phi_n(i,0) \qquad i = 1,2,...,p, \tag{3.18}$$

Este conjunto de p ecuaciones con p incógnitas se puede resolver de maneras muy eficientes para los coeficientes predictores desconocidos $\{\alpha_k\}$ que minimicen el promedio cuadrático del error de predicción para un segmento s_m. Usando (3.14) y (3.16), se puede expresar el error cuadrático medio de predicción como :

$$E_n = \sum_{m=0}^{N+p-1} s_n^2(m) - \sum_{k=1}^{p} \alpha_k \sum_{m=0}^{N+p-1} s_n(m)s_n(m-k) \tag{3.19}$$

y usando (3.18) se puede expresar E_n como :

$$E_n = \phi_n(0,0) - \sum_{k=1}^{p} \alpha_k \phi_n(0,k) \tag{3.20}$$

Por lo tanto el error mínimo total consiste en un componente fijo y en un componente que depende de los coeficientes predictores. Para encontrar los coeficientes predictores óptimos, se debe primero computar las cantidades $\phi_n(i,k)$ para $0 \leq k \leq p$. Cuando el paso anterior se encuentre resuelto solo tendremos que resolver (3.18) para obtener las α_k. Sin embargo los detalles de computación de $\phi_n(0,k)$ y la solución subsecuente de las ecuaciones son algo complicados. A continuación expondremos el método de Levison-Durbin que es el que se utilizó en el reconocedor empleado en esta tesis

Solución de las ecuaciones de LPC

El método formal para convertir los coeficientes de autocorrelación a coeficientes LPC se lleva por nombre *método de Durbin-Levinson* y se puede formular como:

$$E^{(0)} = r(0)$$

$$k_i = \frac{\left\{r(i) - \sum_{j=1}^{L-1} \alpha_j^{(i-1)} r(|i-j|)\right\}}{E^{(i-1)}}, \qquad 1 \leq i \leq p$$

$$\alpha_i^{(i)} = k_i$$

$$\alpha_j^{(i)} = \alpha_j^{(i-1)} - k_i \alpha_{i-j}^{(i-1)}$$

$$E^{(i)} = (1 - k_i^2 E^{(i-1)})$$

Donde se omite la sumatoria en la segunda ecuación para $i = 1$. El conjunto de ecuaciones es resuelto recursivamente para $i = 1, 2, ..., p$, y la solución final esta dada por

a_m = coeficientes LPC $\alpha_m^{(p)}$

Programación del cálculo del LPC en Matlab

Para calcular el centroide se necesitan las correlaciones de los LPC de las señales de entrenamiento. El siguiente algoritmo calcula la correlación de los LPC a partir de la función `Levinson` que proviene del *Signal Processing Toolbox* de Matlab.

En Matlab el algoritmo se podría programar de la siguienta manera:

```
1   function corr_lpc_centroide = xcorr_lpc(corr_centroide,grado)
2
3   %Entrada : - Matriz con regiones perturbadas de tamaño jxnxm
4   %            ( donde j = k*2).
5   %Salida : - Matriz de tamaño jxm que esta compuesta por la
6   %          correlación de los vectores LPC calculados para
7   %          cada promedio del los vectores de cada región.
8   %Descripción:
9   % Se calculan con coeficientes "ra" para calcular las
10  % distancias itakura-saito de cada vector a los centroides
11  % de cada region.
12  % Calcular 15 primeros coeficientes de la correlacion del LPC
13  tamanio = size(corr_centroide);
14  for k = 1:1:tamanio(1)
15    lpc_centroide(k,:) =  levinson(corr_centroide(k,:));
16    corr_lpc_centroide_temp(k,:)=xcorr(lpc_centroide(k,:));
17    corr_lpc_centroide(k,:)=\
18                          corr_lpc_centroide_temp(k,16:16+grado);
19  end
```

Programación de LPC en C

El programa en C es una función que recibe los datos de la autocorrelación de señal en un apuntador `r` y el orden del vector que queremos en `m`. La rutina

devuelve el vector LPC en un apuntador que se la pasa a, también devuelve los coeficientes de reflexión en rc y alfa.

```
1   /* Calculo los coeficientes LPC de las autocorrelaciones */
2   calculate_lpc(r,m,a,rc,alfa)
3   float a[],rc[],*alfa,r[];
4   int m;
5   {
6     int minc,ip,ib,mh,i,result;
7     float s,at;
8
9     result=1;
10    if(r[1] <= 0){
11      result=0;
12      return(result);
13    }
14    rc[1]= -r[2]/r[1];
15    a[1]=1.;
16    a[2]=rc[1];
17
18    *alfa=r[1]+r[2]*rc[1];
19
20    for(minc=2;minc<=m;minc++){
21      s=0.;
22      for(ip=1;ip<=minc;ip++){
23        s=s+r[minc-ip+2]*a[ip];
24      }
25
26      rc[minc]= -s/(*alfa);
27      mh=minc/2+1;
28
29      for(ip=2;ip<=mh;ip++){
30        ib=minc-ip+2;
31        at=a[ip]+rc[minc]*a[ib];
32        a[ib]=a[ib]+rc[minc]*a[ip];
33        a[ip]=at;
34      }
35
36      a[minc+1]=rc[minc];
37
38      *alfa= *alfa+ ( rc[minc]*s );
39
40      if(*alfa <= 0){
41        printf("\n unstable lpc filter");
42
43        for(i=1;i<=m;i++){
```

```
44            printf("\n autocorrelations r[%d] %f",i,r[i]);
45          }
46
47          result=0;
48        }
49      }
50      return(result);
51
52    }
```

3.3.5 Cuantización vectorial del LPC

La representación LPC de una señal de voz nos proporciona una serie de vectores que reflejan las características de su espectro variando en el tiempo. Para reconocer una señal de voz se puede comparar una serie de vectores de entrenamiento contra una serie de vectores a reconocer y cuantificar la distorsión entre ellos. Sin embargo esta comparación resulta impráctica ya que implica el manejo de demasiados datos para que se efectúe en tiempo real.

Para reducir la cantidad de datos en la comparación se efectúa una cuantización vectorial [15] al conjunto de vectores LPC de entrenamiento.

Un cuantizador vectorial Q de dimensión k y tamaño N es una transformación de un vector (o un punto) del espacio Euclidiano de dimensión k, $\mathbb{R}^k$, en un conjunto finito $\mathbb{C}$ que contiene N salidas o puntos de reproducción , llamados vectores de código (*code vectors*). Así

$$Q : \mathbb{R}^k \rightarrow \mathbb{C}$$

donde $\mathbb{C} = (y_1, y_2, ..., y_N)$ y $y_i \in \mathbb{R}^k$ para cada $i \in \mathbb{J} = \{1, 2, ..., N\}$. El conjunto $\mathbb{C}$ es llamado el Alfabeto (*Code Book*) y tiene un tamaño N, lo que significa que se tienen N elementos distintos, donde cada uno de ellos es un vector $\mathbb{R}^k$.
La cuantización vectorial ayuda al reconocimiento en dos aspectos:

- Reduce la cantidad de datos que componen los vectores de entrenamiento asignando a cada palabra un vector aproximado creado a partir de todos los vectores que componían todos los entrenamientos de una palabra.

- Crea un vector *promedio* que permite que el usuario no tenga que repetir una palabra exactamente igual a la que fue entrenada. Esto se debe a que el vector cuantizado contiene los vectores que representan varios entrenamientos.

De esta forma se disminuye el tiempo decomputación necesaria para el análisis de la similitud de los vectores espectrales y se obtiene un mejor reconocimiento creando un vector que representa las características principales de la palabra.

A continuación se presentarán los pasos para hacer la cuantización vectorial de una palabra.

Elementos necesarios para efectuar una cuantización vectorial

Para hacer una cuantización vectorial se necesita de:

1. Un gran conjunto de vectores de análisis espectral.
2. Un medida de similitud, o distancia, entre un par de vectores de análisis espectral.
3. Un procedimiento computacional para encontrar un centroide.
4. Un procedimiento de clasificación para vectores arbitrarios de análisis espectral de voz que clasifique el vector del Alfabeto (*Codebook)* que sea más cercano al vector de entrada y que use el índice del Alfabeto como la representación espectral resultante.

Conjunto de vectores de análisis espectral

El conjunto de vectores de análisis espectral se obtiene a partir de entrenamientos de palabras. La calidad de los entrenamientos dependerá de varios factores como, quien habla, en que condiciones habla, que transductores se utilizan y que sistemas de transmisión están involucrados.

Medida de distancia de similitud

La distancia [13] para comparar vectores espectrales v_i y v_j es de la forma:

$$d(v_i, v_j) = d_{ij} \begin{cases} = 0 & \text{si } v_i = v_j \\ > 0 & \text{de otro modo} \end{cases}$$

En el caso de este reconocedor de voz se utilizó la medida de distorsión Itakura-Saito que nos proporciona una medida de distorsión espectral en función de dos densidades espectrales.

Programación de medida de distancia en Matlab

En Matlab el algoritmo se podría programar de la siguienta manera:

```
1   function distancia= itakura(corr_lpc_centroide,region)
2
3   %Entrada : - Matriz de tamaño jxm correlación de LPC
4   %            centroides de coorelación
5   %          - Matriz de jxnxm donde k es el número de regiones
6   %            que existen, n representa el número de ventanas
7   %            y m el número de coeficientes de correlación.
8   %Salida:   - Vector de tamaño jxn que contiene las distancias
9   %            de cada vector a cada centroide.
10  %
```

```
%Descripción:
%       Calcula la distancia itakura-saito de la correlación
%       de cada ventana a cada correlacion LPC de centroides

numero_centroides = size(corr_lpc_centroide);
region = squeeze(region);

% Calculo la distancia para cada ventana
for centroide= 1:numero_centroides(1)
  for ventana = 1:length(region)
    Z0(ventana)=\
         corr_lpc_centroide(centroide,1)*(region(ventana,1));
    Z1(ventana)= 2*\
sum(corr_lpc_centroide(centroide,2:16).*region(ventana,2:16));
    distancia(centroide,ventana) =  Z0(ventana)+Z1(ventana);
  end
end

```

Creación del Alfabeto

La forma en que un conjunto de vectores L puede ser reunidos en otro conjunto de M vectores de código es la siguiente:

1. Inicialización: Se seleccionan arbitrariamente M vectores como el conjunto inicial de palabras de código en el Alfabeto.
2. Búsqueda del vecino más cercano: Para cada vector, se encuentra el vector de código más cercano y se asigna este vector a la celda correspondiente.
3. Actualización de centroide: Se actualiza el vector de código usando el centroide de los vectores de entrenamiento a esta celda.
4. Iteración: Se Repiten los pasos 2 y 3 hasta que la distancia sea menor a un cierto umbral.

Para mejorar el procedimiento anterior se puede modificar el paso 1 de la siguiente forma. En vez de seleccionar arbitrariamente M vectores como el conjunto inicial de palabras de código se crea un centroide a partir de todos los vectores, este centroide se divide creando centroides hasta el orden requerido en el Alfabeto. Los pasos a seguir para encontrar estos centroide serían los siguientes:

1. Diseñar un Alfabeto de 1 vector que es el centroide de todos los vectores.
2. Duplicar el tamaño del Alfabeto partiendo cada Alfabeto actual y_n siguiendo la regla:

$$y_n^+ = y_n(1+\epsilon)$$

$$y_n^- = y_n(1-\epsilon)$$

donde n varia de 1 al tamaño del Alfabeto, y ϵ es un parámetro para partir el Alfabeto.

3. Usar el algoritmo descrito anteriormente para obtener el conjunto de centroides para el Alfabeto y crear nuevas regiones a partir de estos centroides.

4. Iterar los pasos 2 y 3 hasta que el Alfabeto de tamaño M sea diseñado.

El algoritmo para la creación de un Alfabeto se puede observar en la figura 3.5.

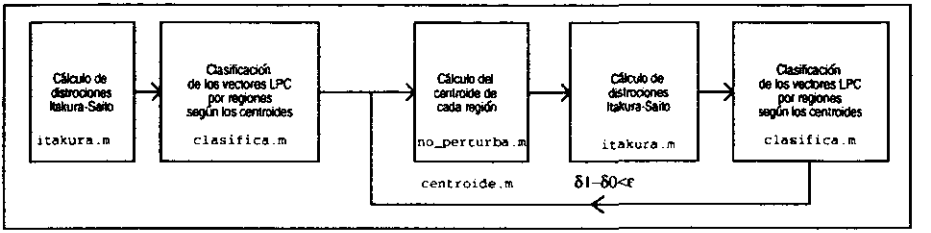


Figura 3.5: Diagrama de bloques del algoritmo de cuantización vectorial

A continuación se presenta el programa centroides.m que genera el cuantizador vectorial de una palabra.

```
1   function  [num_vec_regiones,regiones] = \
2                           centroide(corr_lpc_centroide,region)
3
4   mensaje = \
5   '----------------- INICIA CALCULO CENTROIDES -------------'
6
7   % Calculos las distancias de cada centroide a cada vector
8   distancia= itakura(corr_lpc_centroide,region);
9
10  % Extraigo los distancias minimos con sus indices
11  [minimo,indice_minimo] = min(distancia);
12  numero_regiones = size(corr_lpc_centroide)
13
14  % Clasifico en las dos nuevas regiones mis centroides
15  [num_vec_regiones,regiones] = \
16          clasifica(numero_regiones(1),indice_minimo,region);
17
18  % Se vuelven a calcular los centroides con las regiones
19  % ya creadas a partir de las dos regiones iniciales hasta que
20  % dif_nor_sum_dist_min > ( 0.001 * 2^(fix(iteracion/20))
21  dif_nor_sum_dist_min = 1
```

```
22  iteracion = 0
23  corr_lpc_centroide_back = corr_lpc_centroide;
24
25  while  dif_nor_sum_dist_min > \
26                          abs((0.001 * 2^(fix(iteracion/20))))
27    mensaje =
28
29  '----------------- ITERANDO CENTOIDES  -------------------'
30
31    % Condiciones iniciales de cada iteracion
32    limite = 0.001 * 2^(fix(iteracion/20))
33    iteracion = iteracion + 1
34
35    % Calculos las distancias de cada centroide a cada vector
36    corr_lpc_centroide = no_perturba(regiones,num_vec_regiones);
37
38    % Calculos las distancias de cada centroide a cada vector
39    distancia_bis = itakura(corr_lpc_centroide,region);
40
41    % Se saca cuales fueron las distancias minimas y sus indices
42    [minimo_bis,indice_minimo_bis] = min(distancia_bis);
43
44    % Clasifico en las dos nuevas regiones mis centroides
45    [num_vec_regiones_bis,regiones_bis] = \
46        clasifica(numero_regiones(1), indice_minimo_bis,region);
47
48    % y se comparan
49    dif_nor_sum_dist_min = \
50                  abs((sum(minimo)-sum(minimo_bis))/sum(minimo));
51
52    % Se pasan los centroides de region nueva a region vieja
53    regiones = regiones_bis;
54    num_vec_regiones = num_vec_regiones_bis
55
56    % Se guarda la nueva distancia minima
57    minimo = minimo_bis;
58
59  end
```

3.4 Reconocimiento

Para reconocer una palabra se ejecutan los siguientes pasos:

1. Se pasa la señal de la palabra a reconocer a través del preprocesamiento mencionado en los puntos 3.3.1 y 3.3.2 para obtener una señal con un espectro aplanado y estacionaria.

2. Se calcula la distancia de cada vector de la palabra preprocesada a todos los vectores de cada Alfabeto.

3. Se toma la distancias mínimas de cada vector de la palabra a reconocer a cada vector de los Alfabetos.

4. Se suman las distancias mínimas que se obtuvieron de cada Alfabeto.

5. Se selecciona como Alfabeto que representa la palabra el que obtiene la menor sumatoria

A continuación se puede observar el diagrama de bloques de como interactuan varias funciones programadas en Matlab para reconocer una palabra.

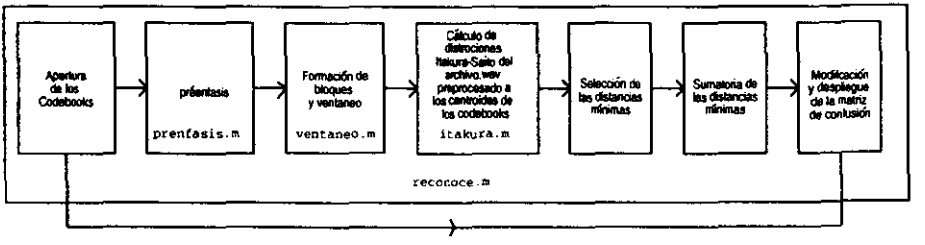


Figura 3.6: Diagrama de bloques del algoritmo de reconocimiento

En Matlab el algoritmo se podría programar de la siguienta manera:

```
% Rutina para reconocer  la senal

% Se borran todas la variables anteriores
clear

% Abrimos los libros de codigo
fid=fopen('W0.cod','r');
[CodeBook_cero,count]=fread(fid,[32,16],'float');
fid=fopen('W1.cod','r');
[CodeBook_uno,count]=fread(fid,[32,16],'float');
fid=fopen('W2.cod','r');
[CodeBook_dos,count]=fread(fid,[32,16],'float');

% Se cargan el archivo a reconocer
% y se guarda su Frecuencias de muestreo en Fm
% Nota: Se supone que se muestrean todas las senales
% a la misma frecuencia
nombre = dir('wavs');
tam = size(nombre);
matriz_de_confusion = zeros(3,3);

```

```
22  for numero = 3:tam(1),
23    archivo = nombre(numero).name;
24    ruta = ('wavs/');
25    ruta(1,6:length(archivo)+5) = archivo(1,:)
26    [y,Fm] = wavread(ruta);
27
28    %I.- PRENFASIS
29
30    %1. CALCULO DEL PREÉNFASIS
31    s = prenfasis(y);
32
33    %2.- FORMACION DE BLOQUES DE VENTANAS DE HAMMING
34    [NumVentanas,xw] = ventaneo(s,Fm);
35
36    % Se calcula la autocorelacion del las ventanas de toda la
37    % senal
38
39    % Se calcula los 15 primeros coeficientes de la correlacion
40    % de las ventanas de 256 datos de todos las vectores.
41    for ventana=0:NumVentanas*2-(2);
42      % separacion de la ventana y correlacion de la ventana
43      muestra = xcorr(xw(256*ventana+1:256*(ventana+1)),'coeff');
44      % muestreo de los 15 elementos
45      region(1,ventana+1,:) = muestra(256:271);
46    end
47
48    % Calculos las distancias de cada centroide a cada vector
49    distancia_cero = itakura(CodeBook_cero,region);
50    distancia_uno = itakura(CodeBook_uno,region);
51    distancia_dos = itakura(CodeBook_dos,region);
52
53    minimo0 = min(distancia_cero);
54    minimo1 = min(distancia_uno);
55    minimo2 = min(distancia_dos);
56
57    distancia(1) = sum(minimo0);
58    distancia(2) = sum(minimo1);
59    distancia(3) = sum(minimo2);
60
61    [distancia_minimo,inidice_distancia_minimo] = min(distancia);
62    [inidice_distancia_minimo-1,distancia_minimo,distancia];
63    matriz_de_confusion(nombre(numero).name(2)-47, ...
64                        inidice_distancia_minimo)=\
65    1+matriz_de_confusion(nombre(numero).name(2)-47,...
66                        inidice_distancia_minimo)
67  end
```

Capítulo 4

Prototipo de un reconocedor de palabras aisladas

4.1 Reconocedor de palabras aisladas

El reconocedor de palabras descrito a continuación fue programado en lenguaje C. El reconocedor esta compuesto por tres programas `get_smpl`, `get_vqs` y `recognize`.

4.1.1 Descripción de programas

`get_smpl`: El entrenador se utiliza para recopilar el sonido de una palabra. Gracias a este programa el usuario le asigna a una palabra un sonido. Este entrenamiento se puede efectuar varias veces de tal manera que se pueda reconocer la palabra gracias a un sonido *promedio*.

`get_vqs`: El modelador calcula la cuantización vectorial de los LPC de una palabra gracias a los archivos de entrenamiento. La cuantización vectorial devuelve una serie de vectores que representa las características espectrales de la palabra.

`recognize`: El reconocedor reconoce en tiempo real las palabras que dice un usuario. El reconocimiento se obtiene gracias a la comparación de las características espectrales del sonido de la palabra dicha con los diferentes modelos de las palabras creadas con `get_vqs`.

4.1.2 Descripción de bibliotecas

Los programas anteriores utilizan las funciones de las bibliotecas descritas a continuación. Normalmente una función se utiliza en varios programas por lo

cual la modularización simplifica considerablemente el código de los programas.

`allocate.c`: Esta biblioteca contiene las funciones que sirven para hacer malloc de diferentes tipos(char, int, double) y tamaños (vector, matrix,hypermatrix).

`audio_linux.c`: Esta biblioteca contiene las funciones para configurar, ver el status, escribir y leer el dispositivo de audio.

`audio.h`: Header de algunas variables de audio_linux.

`basics.c`: Esta biblioteca contiene funciones matemáticas

`basics.h`: Header de algunas variables de basics (definición de constantes) como el preénfasis, el ventaneo, y otras funciones que se utilizan a menudo en el procesamiento digital de voz.

`global.h`: Header de algunas variables globales (definición de constantes).

`gsNetPoll.c`: Biblioteca que sirve para utilizar sockets (se utiliza para mandar las palabras reconocidas entre diferentes programas).

`gsNetPoll.h`: Header de algunas variables de gsNetPoll.c.

`sockets_lib.c`: Biblioteca que se utiliza por gsNetPoll.c para accesar rutinas básicas de sockets.

`lpc_lib.c` Bibliotecas donde se encuentran las rutinas para calcular los LPC.

`key_lpc.c` Bibliotecas donde se encuentran las rutinas para hacer la cuantización vectorial de los LPC.

4.1.3 Descripción de los archivos de información y configuración

Los archivos de datos que genera la versión actual se pueden dividir en los archivos que contienen información de una señal y en los archivos que informan de la asociación de palabras a señales.

Archivos del procesamiento digital de señales

archivos con extensión .wav Este archivo contiene en formato wav el sonido de una palabra de entrenamiento, es generado cuando se entrena una palabra. Su nombre se forma de la siguiente forma:

"W_"+nombre_de_la_palabra+"_"+número del entrenamiento+".wav"

Este archivo se puede oír en muchas utilerias reproductoras de archivos en formato "wav", como por ejemplo el SoundStudio.

archivos con extensión .cor Este archivo contiene la correlación de los datos numéricos del archivo -wav, se genera cada vez que se entrena una palabra. Su nombre se genera de la siguiente forma :

'W_"+nombre_de_la_palabra+"_"+número del entrenamiento+".cor"

archivos con extensión .ar* Existen cuatro tipos de archivos que tienen como extensión .ar:

nombre_de_la_palabra+"ar"+"_"+número

nombre_de_la_palabra+"ar"+"_"+número+"_dis"

nombre_de_la_palabra+"ar"+"_"+número+"_sim"

nombre_de_la_palabra+"ar"+"_"+número+"_spa"

Estos archivos contienen los resultados de los cálculos que se efectuán para obtener el modelo de una palabra.

4.2 Interfaz gráfica de usuario

La interfaz gráfica de usuario se ha convertido en una parte crucial de casi cualquier programa. En el caso de un reconocedor de voz, ésta es especialmente importante, ya que para que el reconocedor facilite realmente la interacción del usuario con el sistema, es indispensable que tenga una interfaz amigable con el usuario.

Características de una interfaz amigable

Está comprobado que una de las características más importantes de un software desde el punto de vista del usuario es que lo pueda manejar fácilmente. Esta facilidad de manejo se forja a partir de:

- Una amplia gama de widgets que se adecuen a las acciones que quiera realizar el usuario.
- Consistencia en el ambiente gráfico que le permita al usuario moverse intuitivamente en el GUI gracias a comportamientos similares en todos sus componentes.
- Una comunicación transparente entre los diferentes componentes del ambiente gráfico.

El *toolkit* gráfico Gtk+ dentro de un ambiente GNOME conjuga todas estas características.

4.2.1 Un *toolkit* gráfico completo y fácil de programar: Gtk+

Gtk+ es un *toolkit* para crear GUI bajo licencia GNU. El diseño orientado a objetos de Gtk+ y su licencia permiten que se desarrolle libremente con él sin tener que estar sujeto a comprar licencias. Cabe mencionar que uno de los problemas de la versión anterior del reconocedor de voz es que utilizaba un *toolkit* de Motif propietario, lo cual hacia imposible distribuirlo. Otra de las grandes ventajas de Gtk+ es que uno no esta ligado a ningún lenguaje para usarlo aunque este está escrito en C. En el caso de este sistema se utilizó unas extenciones de Python y toda la programación se hizo con el lenguaje Python. Finalmente el Gtk+ implementa un sistema de señales fácilmente utilizable y muy flexible lo cual permite una fácil comunicación entre los widgets. Esta fácil comunicación se generaliza dentro del ambiente gráfico que contiene Gtk+: GNOME.

4.2.2 Una fácil integración a un desktop: GNOME

GNOME es un ambiente amigable al usuario que le permite utilizar y configurar fácilmente sus computadoras. GNOME encierra un conjunto de convenciones que fácilitan la cooperación y consistencia entre aplicaciones. En consecuencia el usuario puede utilizar en conjunto varias aplicaciones sin tener que aprender el manejo de varias interfaces gráficas.

Por otro lado GNOME provee varias ventajas a los desarrolladores. GNOME se puede utilizar desde varios lenguajes ya que se diseñó con este objetivo en mente. Además GNOME utiliza CORBA (*Common Objetc Request Broker Architecture)*, para permitir que los componentes de software interoperen transparentemente independientemente de los lenguajes en que estén implementados o de la arquitectura en que estén ejecutándose.

Es importante recalcar que la interfaz de CORBA puede ayudar mucho en el desarrollo de un sistema de manejo de GUI por voz ya que podría proveer un mecanismo robusto y bien estructurado para comunicar el reconocedor de voz con cualquier aplicación.

4.2.3 Bindings de Python a GNOME: pygnome

Pygnome es un conjunto de lazos (*bindings*) del lenguaje Python al conjunto de widgets que compone GNOME Y GTK. Pygnome provee una interfaz orientada a objetos que tiene un nivel un poco más alto que la interfaz en C. Pygnome hace automáticamente los castings de los tipos y cuenta las referencia que normalmente se deberían de hacer con GNOME y GTK cuando se programan en C.

4.2.4 Desarrollo de la interfaz gráfica

Programación de interfazs gráficas con Python

Python es un buen lenguaje para programar interfaces gráficas de usuario ya que posee las características de un lenguaje orientado a objetos, como clases herencia múltiple y funciones virtuales, pero es simple de programar. La simpleza de la programación ayuda al programador a desarrollar interfaces rápidamente con respecto al tiempo de desarrollo de la misma interfaz implementada en C. Además surgió una ventaja inesperada: Ya que el GNOME estaba en desarrollo durante la codificación de prototipo el API cambiaba constantemente y era muy difícil programar con GNOME ya que las llamadas en C cambiaban constantemente de nombre y de argumentos. Sin embargo al usar los lazos se creó una capa intermedia donde el programador de los lazos modifica constantemente los lazos para que funcionen con los nuevos prototipos y argumentos pero mantenía la sintaxis de las llamadas en Python. De esta forma se evita al programador estar al tanto de los cambios del API de programación en C y se puede concentrar en la programación de la interfaz.

Módulos de la interfaz gráfica

La interfaz gráfica esta compuesta por cuatro módulos:

icepick: Módulo donde se crean las instancias de las clases necesarias para iniciar la aplicación.

data: Interfaz de Python con los programas del reconocedor en C.

gui: Interfaz gráfica de usuario.

communication: Interfaz para comunicar el GUI y los procesos de reconocimiento y entrenamiento a través de sockets.

Dentro de cada módulo se encuentran las clases que los componen, a excepción de Icepick que sólo contiene funciones para crear las instancias iniciales. En ;a figura 4.1 se puede ver un árbol de las clases que componen el sistema que fue generado a partir del código gracias a un IDE llamado OO-Browser. En la parte superior del árbol se pueden ver la clases del GUI creadas a partir de clases de GTK. En la parte inferior se puede observar el Recognizer_server que es un servidor de sockets. Gracias a este servidor la rutinas de reconocimiento de voz se pueden comunicar con el Recognizer_gui como se puede observar en la siguiente figura. La invocación de estas rutinas de voz se hacen gracias a las clases Voice y Word que implementan una interfaz entre las rutinas de C y el Recognizer_gui.

4.3 Evaluación del software

En esta sección se analizará las cualidades y los defectos de los programas que componen el reconocedor de palabras aisladas para extraer que modificaciones

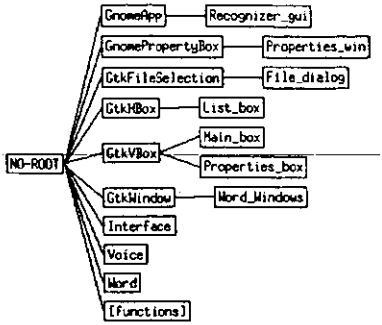


Figura 4.1: Árbol de clases del *front-end* del reconocedor de voz

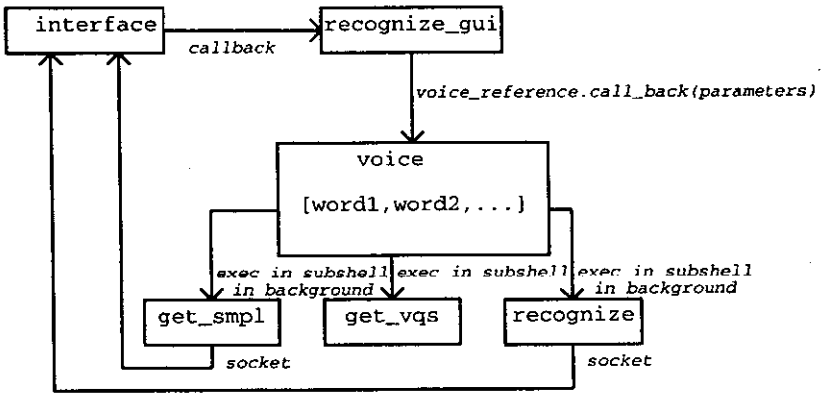


Figura 4.2: Diagrama de bloques de las clases de front-end del reconocedor de voz

se le deben de hacer para que cumplan los estándares de software de dominio público.

4.3.1 Cualidades

El reconocedor de palabras aisladas fue escrito con los siguiente objetivos en mente: "Que funcionara en tiempo real" y que se pudiera modificar configurar fácilmente los parámetros del reconocimiento (velocidad muestreo, tiempo de muestreo, número de muestras, etc...). Esto se debe a que el propósito principal del software era comprobar que las rutinas de reconocimiento de voz funcionavan en tiempo real en una microcomputadora y ver cuál era la configuración óptima de calidad de reconocimiento vs. el cómputo requerido.

Tiempo real

El software desarrollado procesa la señal en tiempo real en dos casos . El primer caso sucede cuando uno está entrenando una palabra para ser posteriormente reconocida. El programa `get_smpl` detecta automáticamente cuando el usuario empezó a decir una palabra y cuando terminó de decirla. El segundo caso sucede en el reconocimiento. En este caso el programa `recognize` además de poder detectar cuando se dijo una palabra, indica inmediatamente (desde el punto de vista de un usuario) que palabra se dijo .
Se observó a lo largo de todas las pruebas de entrenamiento y de reconocimiento que el usuario nunca tuvo que esperar procesamiento alguno.

Modularización

Existe una modularización de los diferentes bloques que componen el reconocedor de voz. Se tiene en cada biblioteca agrupadas las funciones que atienden un diferente bloque del sistema. Las funciones que se usan a menudo en el procesamiento de voz se encuentran en la biblioteca de `basics`, mientras que las que calculan los LPC y su cuantización vectorial se encuentran en `lpc_lib` y `key_lib`. Por otro lado las rutinas de comunicación por sockets se encuentran en `gsNetPoll` y `sockets_lib`.

4.3.2 Defectos

Cuando se programaron las rutinas del reconocedor no se tenia como prioridad que las rutinas se liberarán como software público. En consecuencia no se abordaron varios puntos necesarios para convertirlo en software público como documentación y portabilidad.

Manejo directo de archivos

Otra de las características del software público es que se procura que los archivos de configuración sean fácilmente legibles y modificables. Esto es un poco complicado cuando se generan muchos archivos por palabra, por lo tanto se podría generar un sólo archivo de dato por palabra. Esto se puede hacer fácilmente gracias a bibliotecas GNU como *LibPropList*.

Manejo directo de dispositivos

Una de las áreas donde GNU/Linux a estado muy activo es en el soporte a dispositivos. Esto se debe a que en el caso de otros sistemas operativos el fabricante del dispositivo se encarga de programar el driver. Sin embargo en GNU/Linux los manejadores son programados por voluntarios que a menudo no cuentan con la documentación necesaria. Por lo tanto el soporte a dispositivos es a veces escaso y no robusto. En el caso del reconocedor de LII uno se ve en la obligación de trabajar con tarjetas Sound Blaster ya que la configuración de tarjetas de otras marcas pueden resultar complicadas o imposibles de configurar.

Sin embargo, como se verá mas adelante ya se esta liberando manejadores de dispositivos de sonido robustos que soportan muchas tarjetas lo que liberará al usuario de la configuración.

Manejo directo de comunicación entre procesos con mecanismos y protocolos robustos.

La comunicación con aplicaciones es una de las partes medulares del reconocedor de voz. No sirve de nada reconocer voz si no existe una forma fácil de acoplarlo con otras aplicaciones. En el caso del reconocedor que se utilizó, sólo se contaba con una capa de sockets que trabajaban con un protocolo unidireccional sin confirmación. Esto puede dificultar la integración del reconocedor a un sistema. Para una buena interacción con el sistema se requiere que el sistema pueda comunicarse de manera fácil y robusta con la aplicación a la cual le manda el comando asignado a la palabra reconocida. De esta forma la aplicación le puede confirmar que recibió el comando y que es un comando válido. En caso de que comando enviado por el reconocedor de voz no sea válido este podría avisar al usuario que su petición no fue exitosa.

4.4 Sugerencias para mejorar las rutinas de reconocimiento de voz

La licencia GNU es sólo una parte que contribuye a liberar el software público; sirve para estar protegido legalmente de los malos usos que se le puedan dar. Sin embargo un requisito clave para hacer un software público es que pueda ser entendido, modificado y portado fácilmente. Para que se cumplan estos puntos el software debe de tener una buena documentación y estar correctamente modularizado . Para lograr la portabilidad existen varias utilerias GNU que dispensan al programador de tener que lidiar con diferentes sistemas operativos y arquitecturas.

4.4.1 Utilización de bibliotecas GNU

Una de las grandes ventajas de programar con licencia GNU es que uno puede utilizar sus bibliotecas. Estas bibliotecas abarcan un espectro muy amplio de funciones que son generalmente robustas y que están bien documentadas. De esta forma el programador puede concentrase en el problema que resuelve su programa y no tiene que estar programando su sistema desde cero. Por ejemplo en el caso del reconocedor de voz ya existen bibliotecas que proveen estructuras de datos portables, funciones para generar y leer archivos de configuración, funciones para leer los argumentos con los que se invoca el programa y muchas otras bibliotecas más. Gracias a lo anterior el programador se puede concentrar en la programación del procesamiento digital que requiere el reconocimiento de voz sin preocuparse sobre la portabilidad, persistencias de configuración y otros componentes del software.

4.4.2 Una biblioteca de estructuras de datos en C reusable: *glib[1]*

Dado que la portabilidad cobra cada vez más importancia ya que no se sabe sobre qué plataforma podría estar trabajando un contribuidor a un proyecto GNU, resulta indispensable que el reconocedor se pueda compilar en cualquier plataforma. Sin embargo la falta de definiciones exactas de los tipo de datos en el ANSI C resulta un impedimento que puede afectar severamente la distribución de un software. Por lo tanto surgió en GNU la *glib* que se diseño para resolver algunos de los problemas de portabilidad que tiene el lenguaje C y provee otras funcionalidades que la mayoría de los programas requieren. Glib se utiliza por GDK, GTK, entre otras aplicaciones.

Glib define un número de tipos usados comúnmente, que puede ser dividido en cuatro grupos:

- Nuevos tipos que no se parte del C estándar : gboolean, gsize, gssize.
- Tipos de enteros que se garantiza que tendrán el mismo tamaño en todas las plataformas: gint8, guint8, gint16, guint16, gint32, guint32, gint64, guint64.
- Tipos que son más legibles y más fáciles de usar que sus similes en el C estándar: gpointer, gconstpointer, guchar, guint, gushort, gulong.
- Tipos que corresponden exactamente a los tipos estándares de C pero que se incluyeron por completitud gchar, gint, gshort, glong, gfloat, gdouble.

Sería por lo tanto un mejora sustancial en el programa de reconocimiento de voz utilizar esto tipo de datos portables.

4.4.3 Generación y lectura Archivos de configuración: *libPropList[*, libPropList*/LIBPROPLIST*

La generación de archivos de configuración fácilmente legibles se ha vuelto una necesidad dentro de la comunidad GNU dado que esto facilita el desarrollo en grupo. Hacer un programa robusto en C que genere estos archivos de configuración puede ser muy tardado y muy difícil. Por lo tanto existe una biblioteca de C disponible que lleva por nombre libPropList. Esta biblioteca maneja estructuras opacas en C que nos permiten generar y leer fácilmente archivos de configuración que un desarrollador pueda examinar fácilmente. Al usar esta biblioteca el desarrollador se podrá concentrar en mejorar el núcleo del reconocedor de voz y no preocuparse en tareas que pueden demandar mucho tiempo.

4.4.4 Lectura de argumentos de línea: getopt *getopt[12]*

La función `getopt` se encuentra implementada dentro de la biblioteca de C de GNU. Esta función sirve para poder obtener los argumentos de la línea de comandos. La función `getopt` presenta una forma robusta y fácilmente

configurable de obtener los argumentos con los cuales fue invocado el programa. En el caso del los programas que componen el reconocedor de palabras aisladas se notó que el manejo de los comandos de línea era complicado dado que se escribieron muchas funciones para obtener los parámetros.

4.4.5 Manejo de dispositivo de sonido: *ALSA[14]*

El manejador de dispositivo de sonido actual es el *OSS/Lite[22].* El OSS tiene varias desventajas, la principal es que no esta bajo licencia GNU. La licencia del OSS es de tipo shareware, donde se provee un manejador con capacidades restringidas (OSS/Lite) mientras no se compre la versión plenamente funcional (OSS).

El manejador ALSA (arquitectura avanzada de sonido para Linux) será distribuido bajo la versión 2 de la licencia de GPL y la licencia LGPL. El proyecto se originó gracias al proyecto: *The Linux Ultra Sound Project*

Sus metas principales son:

1. Crear un manejador de sonido totalmente modularizado que soporte kerneld y kmod.
2. Crear un API de ALSA que sobrepase el API actual de OSS.
3. Mantener la compatibilidad con los binarios de OSS/Lite (versión de OSS gratis).
4. Crear una biblioteca de ALSA (C, C++), que simplifique el desarrollo de las aplicaciones de ALSA.
5. Crear un Administrador de ALSA: un programa de configuración interactivo para el manejador.

Actualmente ya se pueden observar varias ventajas de ALSA con respecto al OSS/Lite como el de full duplex que permitiría que el usuario estuviera oyendo música con audifonos al mismo tiempo que ejecuta los programas de reconocimiento.

4.4.6 Un formato eficiente para almacenar sonido: mp3

El formato con el que se guarda actualmente los archivos de sonido de los entrenamientos es .wav. Este formato crea grandes archivos ya que no tiene ningún tipo de compresión. Si uno desea utilizar el reconocedor de comandos con muchos comandos los archivos podrían tomar un espacio excesivo de almacenamiento. Pero ejemplo en el caso de 100 palabras a reconocer se necesitan 34 Mega-bytes para almacenar los datos del entrenamiento.

Por lo tanto para disminuir el espacio de almacenamiento de los archivos de entrenamiento se recomienda guardarlos con el formato mp3 que reduce en un factor de $\frac{1}{10}$ el espacio requerido. Existen varios codificadores y decodificadores de mp3 bajo licencia GNU que podrían cumplir este propósito.

4.4.7 Una implementación de CORBA para GNOME : *ORBit[20]*

En el futuro las aplicaciones de GNOME exportarán sus funcionalidad a través de CORBA para que cualquiera pueda usarlas independientemente del lenguaje o de la plataforma que utilice. Dado que los programadores del proyecto GNOME no encontraron ninguna implementación de las especificaciones de CORBA que les pareciera adecuada para su proyecto decidieron programar una implementación de CORBA llamada ORBit. ORBit esta programado en lenguaje C y la linea de desarrollo se concentra en que utilice poca memoria y sea rápido. Gracias a ORBit se podría obtener un simple mecanismo mediante el cual el reconocedor de voz pueda llamar las funciones de una aplicación. Sin esto sería imposible pensar una forma sencilla de integrar el reconocedor de voz al desktop.

4.4.8 Una mecanismo para manejar un ambiente gráfico de usuario con voz: *a2x[19]*

El software de dominio público a2x[1], fue desarrollado en el MIT por Bob Scheifler uno de los primeros autores de Xwindows. Desgraciadamente tuvo que dejar el proyecto debido a que empezó a padecer de RSI[2]. El sistema a2x permitía al usuario comunicarse a la computadora a través de comandos de voz reconocidos gracias a DragonDictate, un sistema de reconocimiento de voz. El reconocedor de voz le mandaba cadenas de caracteres en ASCII a a2x que las convertía en eventos provenientes de una teclado o de un mouse en una terminal X. Dado que el manejo de a2x se hace con cadenas de caracteres ASCII este se podría acoplar muy fácilmente con el reconocedor de voz. Gracias a este programa se podría manejar con el vocabulario adecuado cualquier aplicación gráfica ya que se podría enviar cualquier señal producida por el teclado o el mouse gracias a un comando de voz.

Por otro lado a2x maneja un mecanismo de contextos donde se pueden ejecutar diferentes acciones con el mismo comando. El usuario indica en que contexto quiere estar y a2x manda diferentes señales previamente programadas para el contexto indicado. Por ejemplo si el usuario quiere desplazarse hacia abajo en un editor de texto, a2x mandara diferentes señales según el usuario le indique y si se encuentra escribiendo en emacs o en vi.

Este mecanismo de contextos se puede mejorar de varias formas:

La primera es que a2x sea automáticamente informado que ventana está activa y que aplicaciones esta corriendo esto es posible con los manejadores de ventanas actuales.

La segunda mejoría consistiría en restringir el espacio de búsqueda de las palabra a reconocer según los comandos de la aplicación que se esté utilizando. Esto se podría lograr gracias a GNOME dado que uno de sus widgets permite la generación automática de menús.

[1] a2x es un acrónimo para "ASCII to X"

[2] Repetivie Strain Injurie, una enfermedad que imposibilita al programador escribir a máquina debido a una inflamación de los nervios de la muñeca y la mano.

4.4.9 Un ejemplo de manejo de ambiente gráfico gracias a la voz: *Xtalk[6]*

Xtalk es un sistema que se desarrolló en el centro de inteligencia artificial de *SRI[21]*. El propósito de Xtalk fue desarrollar una forma de permitir a los trabajadores de SRI comunicarse con sus computadoras gracias a la voz. De esta forma los empleados que estaban afectados por RSI podrían disminuir el tecleo de sus manos lastimadas. En la documentación de Xtalk se describen problemas y posibles soluciones en la implementación de una interfaz de voz a ambiente gráfico. Por ejemplo se describen gramáticas para el manejo de un procesador de texto, un manejador de archivos y un navegador de Internet entre otros. Se podría implementar el manejo de estas gramáticas para que el usuario pudiera comunicase plenamente con la computadora.

Capítulo 5

Experimentos

5.1 Matriz de confusión

5.1.1 Construcción de las matrices de confusión

Para evaluar el desempeño del sistema de reconocimiento se hicieron las matrices de confusión con diferentes tamaños de vocabularios entrenados tres, diez y quince veces.

Una matriz de confusión es un arreglo donde se puede observar el comportamiento de un sistema de reconocimiento. En el caso de un reconocedor de voz la matriz esta compuesta por renglones que representan la palabras que la computadora le indica decir a un usuario y columnas que representan la palabra que la computadora reconoció a lo largo de una prueba de reconocimiento.

La prueba se desarrolla de la siguiente manera:

1. La computadora escribe en la pantalla la palabra que desea que el usuario diga como se puede observar en la figura 5.1.

2. El usuario lee la palabra y la dice en voz alta.

3. El reconocedor de palabras comunica a una subrutina la palabra reconocida.

4. Esta subrutina forma la matriz de confusión sumando uno al número que tiene por índices :

   - El número de columna que representa la palabra escrita en la pantalla en el punto 1.
   - El número de reglón que representa la palabra escrita en la reconocida en el 3.

Por lo tanto si la palabra que se reconoció es la misma que la que se leyó, entonces el índice de la columna y el índice del renglón son iguales.

De lo anterior se puede apreciar la calidad del reconocimiento observando la diagonal del arreglo:

- Si sólo hay números mayores a cero en la diagonal de la matriz de confusión entonces el reconocimiento es perfecto.

- Si existen números fuera de la diagonal de la matriz entonces el reconocimiento está imperfecto.

Por otro lado también se puede evaluar el desempeño del sistema calculando el porcentaje de las palabras reconocidas. Este se puede obtener sumando todos los números que se encuentran el la diagonal del arreglo y dividendo el sumando entre el número total de palabras que pidió por el sistema.

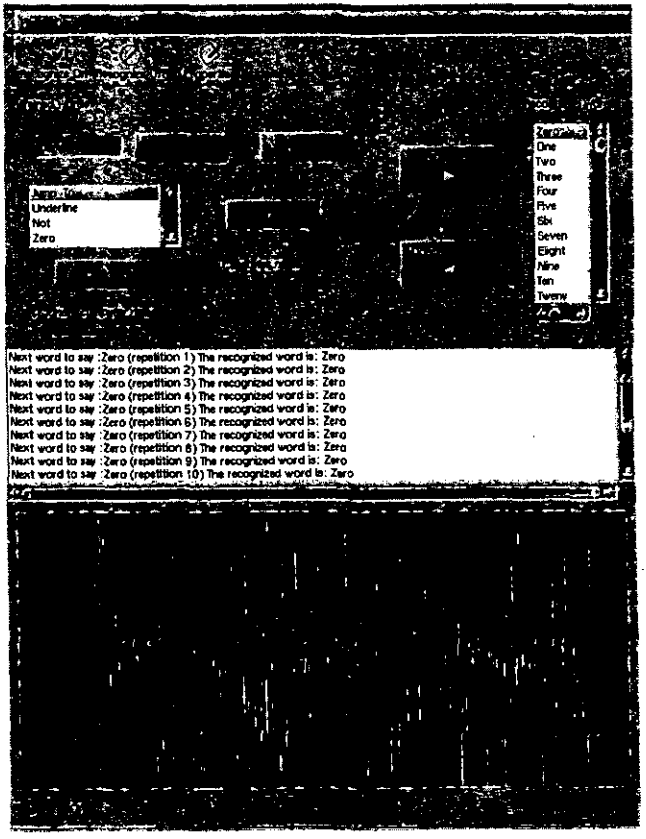


Figura 5.1: Ventana de prueba para hacer matriz de confusión en Icepick

5.2 Resultados de experimentos

5.2.1 Parámetros de los experimentos

Se pueden variar dos parámetros cuando se elabora una matriz de confusión:

- El número de palabras que se prueban que está indicado por el orden n de una arreglo de $n * n$.
- El número de repeticiones que se hacen de las palabras.

También se debe notar que el tipo de palabras que se reconocen afecta el porcentaje de reconocimiento. Cuando se esta trabajando con vocablos monosílabos se tiene un reconocimiento menor que cuando se esta trabajando con palabras compuestas por varias sílabas. Además se tiene que tomar en cuenta que tanto se parecen las palabras que se reconocen debido a que si estas están compuestas por las mismas vocales el reconocedor tiende a confundirlas.

5.2.2 Pruebas de reconocimiento

En la siguientes secciones presentaremos las matrices de confusión que se obtuvieron reconociendo diferentes conjuntos de palabras.

Un conjunto que suele ser utilizado para probar el desempeño de un sistema de reconocimiento de voz son las palabras que representan los números del cero a diez. Se efectuaron experimentos para el conjunto de número en español:

- cero, uno , dos , tres , cuatro , cinco , seis , siete , ocho , nueve

y para el conjunto de números en inglés:

- zero , one , two , three , four , five , six , seven , eight , nine

Otro conjunto que se probó fue un conjunto de palabras que permitiría al usuario manejar una computadora. Este conjunto de palabras se obtuvo de una lista que contiene todas las palabras que tienen un icono preasignado dentro de GNOME. GNOME utiliza un conjunto de iconos asignados a varias palabras que se requieren usualmente en una aplicación como (open, close, move), también contiene todas las palabras necesarias para utilizar mail, un navegador o un editor de texto.

5.2.3 Pruebas de reconocimiento de 10 palabras para un solo usuario con 3 entrenamientos

Números en español

	cero	uno	dos	tres	cuatro	cinco	seis	siete	ocho	nueve
cero	10	0	0	0	0	0	0	0	0	0
uno	0	10	0	0	0	0	0	0	0	0
dos	0	0	10	0	0	0	0	0	0	0
tres	0	0	0	10	0	0	0	0	0	0
cuatro	0	0	0	0	10	0	0	0	0	0
cinco	0	0	0	0	0	10	0	0	0	0
seis	0	0	0	1	0	0	9	0	0	0
siete	1	0	0	0	0	4	0	5	0	0
ocho	0	0	0	0	0	0	0	0	10	0
nueve	0	0	0	0	0	0	0	0	0	10

Porcentaje de reconocimiento: 94.0 %

Tabla 5.1: Matriz de confusión de 10 palabras (números en español) entrenados 3 veces

Números en inglés

	cero	uno	dos	tres	cuatro	cinco	seis	siete	ocho	nueve
cero	10	0	0	0	0	0	0	0	0	0
uno	0	6	0	0	0	0	0	0	0	4
dos	1	0	9	0	0	0	0	0	0	0
tres	0	0	0	10	0	0	0	0	0	0
cuatro	0	0	0	0	10	0	0	0	0	0
cinco	0	0	0	0	0	9	0	0	0	1
seis	0	0	0	0	0	0	10	0	0	0
siete	0	0	0	0	0	0	0	10	0	0
ocho	0	0	0	0	0	0	0	0	10	0
nueve	0	0	0	0	0	0	0	0	0	10

Porcentaje de reconocimiento: 94.0 %

Tabla 5.2: Matriz de confusión de 10 palabras (números en inglés) entrenados 3 veces

5.2.4 Prueba de reconocimiento de 10 palabras para un solo usuario con 10 entrenamientos

Números en español

	cero	uno	dos	tres	cuatro	cinco	seis	siete	ocho	nueve
cero	10	0	0	0	0	0	0	0	0	0
uno	0	10	0	0	0	0	0	0	0	0
dos	0	0	10	0	0	0	0	0	0	0
tres	0	0	0	10	0	0	0	0	0	0
cuatro	0	0	0	0	10	0	0	0	0	0
cinco	0	0	0	0	0	10	0	0	0	0
seis	0	0	0	0	0	0	10	0	0	0
siete	0	0	0	0	0	0	0	10	0	0
ocho	0	0	0	0	1	0	0	0	9	0
nueve	0	0	0	0	0	0	0	0	0	10

Porcentaje de reconocimiento: 99.0 %

Tabla 5.3: Matriz de confusión de 10 palabras (números en español) entrenados 10 veces

Números en inglés

	cero	uno	dos	tres	cuatro	cinco	seis	siete	ocho	nueve
cero	10	0	0	0	0	0	0	0	0	0
uno	0	5	0	0	0	0	0	0	0	5
dos	2	0	8	0	0	0	0	0	0	0
tres	0	0	0	10	0	0	0	0	0	0
cuatro	0	0	0	0	10	0	0	0	0	0
cinco	0	0	0	0	0	9	0	0	0	1
seis	0	0	0	0	0	0	10	0	0	0
siete	1	0	0	0	0	0	0	9	0	0
ocho	0	0	0	0	0	0	0	0	10	0
nueve	0	0	0	0	0	0	0	0	0	10

Porcentaje de reconocimiento: 91.0 %

Tabla 5.4: Matriz de confusión de 10 palabras (números en inglés) entrenados 10 veces

5.2.5 Prueba de reconocimiento de 10 palabras para un solo usuario con 15 entrenamientos

Números en español

	cero	uno	dos	tres	cuatro	cinco	seis	siete	ocho	nueve
cero	10	0	0	0	0	0	0	0	0	0
uno	0	10	0	0	0	0	0	0	0	0
dos	0	0	10	0	0	0	0	0	0	0
tres	0	0	0	10	0	0	0	0	0	0
cuatro	0	0	0	0	10	0	0	0	0	0
cinco	0	0	0	0	0	10	0	0	0	0
seis	0	0	0	0	0	0	10	0	0	0
siete	0	0	0	0	0	0	0	10	0	0
ocho	0	0	0	0	0	0	0	0	10	0
nueve	0	0	0	0	0	0	0	0	0	10

Porcentaje de reconocimiento: 100.0 %

Tabla 5.5: Matriz de confusión de 10 palabras (números en español) entrenados 15 veces

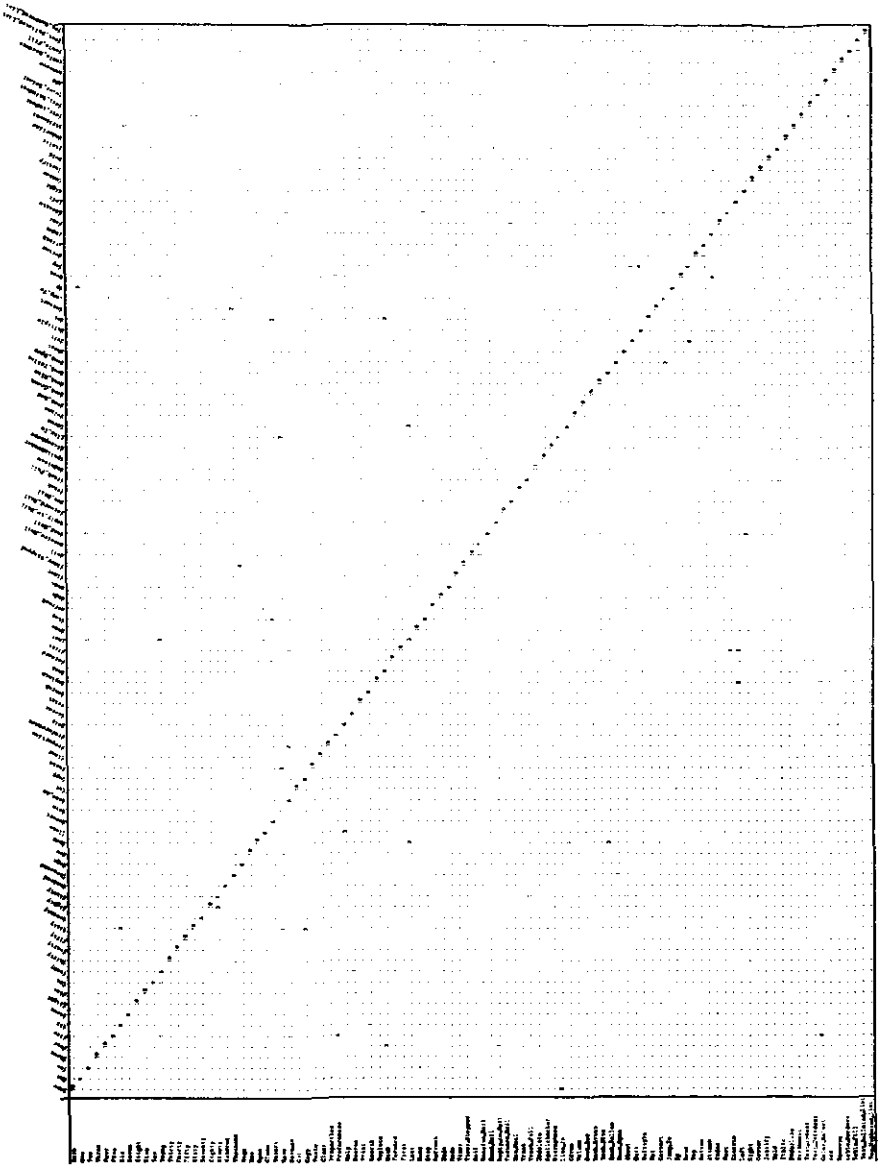
Números en inglés

	cero	uno	dos	tres	cuatro	cinco	seis	siete	ocho	nueve
cero	10	0	0	0	0	0	0	0	0	0
uno	0	8	0	0	0	0	0	0	0	2
dos	0	0	10	0	0	0	0	0	0	0
tres	0	0	0	10	0	0	0	0	0	0
cuatro	0	0	0	0	10	0	0	0	0	0
cinco	0	0	0	0	0	10	0	0	0	0
seis	0	0	0	0	0	0	10	0	0	0
siete	0	0	0	0	0	0	0	10	0	0
ocho	0	0	0	0	0	0	0	0	10	0
nueve	0	2	0	0	0	0	0	0	0	8

Porcentaje de reconocimiento: 96.0 %

Tabla 5.6: Matriz de confusión de 10 palabras (números en inglés) entrenados 15 veces

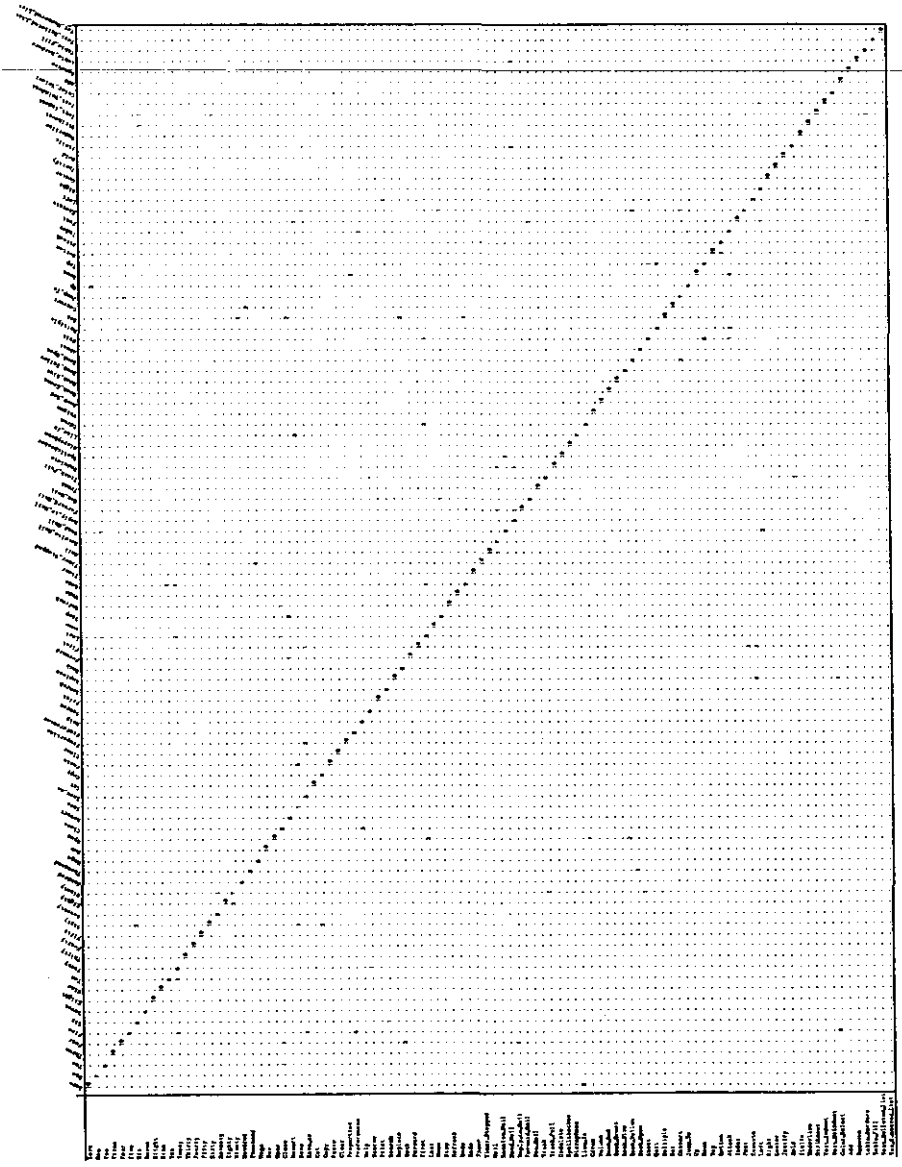
5.2.6 Prueba de reconocimiento de 100 palabras para un solo usuario con 3 entrenamientos



Porcentaje de reconocimiento: 66.9 %

Tabla 5.7: Matriz de confusión de 100 palabras entrenadas 3 veces

5.2.7 Prueba de reconocimiento de 100 palabras para un solo usuario con 10 entrenamientos



Porcentaje de reconocimiento: 85.5 %

Tabla 5.8: Matriz de confusión 100 palabras entrenados 10 veces

5.3 Interpretación de resultados

La interpretación de los resultados no es simple dado que el reconocimiento de palabras puede ser afectado por muchas situaciones:

- El ruido ambiental afecta el reconocimiento, en le caso de las pruebas se procuro mantenerlo a un mínimo.
- Los fonemas que componen las palabras también afectan el reconocimiento. Se debe de procurar utilizar un mínimo de palabras monosilábicas dado que estas tienen a confundirse fácilmente si están compuestas de las mismas vocales.
- También afecta la pronunciación que tiene el usuario si el usuario hace énfasis en la dicción de las características que distinguen una palabra, esta se puede reconocer mucho mejor.

5.3.1 Resultados de las pruebas

En las pruebas de los incisos 5.2.3, 5.2.4 y 5.2.5, se notó que al repetir una prueba sus resultados variaban considerablemente. Por lo tanto se repitió tres veces cada prueba y se tomó el promedio de estas como el porcentaje de reconocimiento en el siguiente análisis de los resultados de las pruebas.

Los resultados con tres entrenamientos son pasables. El porcentaje de reconocimiento que se obtuvo de las pruebas con tres entrenamientos fue de 91% en español y de 93% en inglés. Se nota una mejoría en los resultados con diez entrenamientos , el porcentaje de reconocimiento de las tres pruebas suba a 97% en español y sube a 86% en inglés. Finalmente en los resultados con quince entrenamientos se llega a un porcentaje de reconocimiento casi perfecto de 99.3% en español y de 96.3% en inglés. Sin embargo en las pruebas de 100 palabras se nota un franca disminución en el porcentaje de reconocimiento dado que con tres entrenamientos solo se alcanzó 66.9% con tes entrenamientos y un 85.5% con diez entrenamientos

En estas pruebas se puede observar cómo aumenta el reconocimiento cuando se reduce el espacio de búsqueda de las palabras. Esta puede llegar hasta un 99% en el caso de lo números en español con quince entrenamiento. Cabe señalar que el 99% de los números puede ser representativo para muchos otros casos dado que en el conjunto de uno a diez se tienen cinco palabras monosílabas, y también bastante parecidas como "tres" y "seis".

Estos resultados nos pueden llevar a un buen reconocimiento si se respalda con mecanismos de corrección de error y de contexto (restringiendo el espacio de palabras a seleccionar según los menús disponible).

Capítulo 6

Conclusiones

6.1 Conclusiones

El objetivo de esta tesis era crear un sistema básico de reconocimiento de voz que permitiera ejecutar comandos cuando se dijera una palabra y que fuera de dominio público. Al iniciar el proyecto se asumió que era suficiente añadir una interfaz gráfica a un reconocedor de voz para lograr este objetivo. Sin embargo pronto nos percatamos que era necesario cierto contexto para interpretar correctamente los comandos que se tenia previstos (como un lector de correo electrónico). Esto fue confirmado al terminar de programar la interfaz gráfica que facilitaba mucho el proceso de entrenamiento y la ejecución de reconocimiento pero no proveía un medio eficaz de comunicación oral con la computadora. Sin embargo se investigaron las alternativas que se podrían tener en caso de que se programara la interfaz entre la voz y el ambiente gráfico.

Se encontró una extensión de X windows, a2x, que provee la funcionalidad básica para convertir cadenas de caracteres de ASCII (que le comunica un reconocedor de voz) a señales de *X*. También se encontró el programa Xtalk que describía algunas gramáticas para el uso de algunas aplicaciones (administrador de archivos, navegador, correo electrónico).

Evaluación de reconocedor de voz

A continuación presento las desventajas y ventajas que presenta el reconocedor de palabras aisladas. El entrenamiento de las palabras a reconocer se podría ver como una desventaja del reconocedor de voz, sin embargo el entrenamiento permite que el reconocedor se puedo entrenar con palabras de cualquier lenguaje con cualquier acento. En el caso de los reconocedores preentrenados esto no es posible dado que solamente se entrenan las palabras o los fonemas del lenguaje que se tiene que reconocer. Sin embargo considero necesario que se pueda hacer reconocimiento de voz continua para que el sistema pueda ser utilizable ya que hablar segmentando las frases en palabras aisladas puede ser muy agobiante para el usuario.

Evaluación de la interfaz gráfica del reconocedor de voz

La interfaz gráfica no cumplió su propósito de proveer una canal práctico para que el usuario pudiera manejar su computadora a través de comandos de voz. Sin embargo el GUI facilita inmensamente el proceso de entrenamiento y reconocimiento ya que la generación de los archivos de configuración se efectúa automáticamente. Por otro lado la inclusión de la interfaz gráfica con un *toolkit* gráfico de GNU permite distribuir el software sin ningúna necesidad de tener que instalar algún *toolkit* de GUI propietario como era necesario cuando el software estaba ligado a un *toolkit* propietario de Motif.

6.2 Futuras lineas para desarrollo del reconocedor de voz

El sistema de reconocimiento de voz se debe modificar en tres areas.

La primera es la área de codificación donde se debe hacer un esfuerzo por mejorar el código con una programación bien estructurada, documentación y utilización de librerías GNU. La segunda área es el procesamiento digital de señales, donde se pueden añadir varios módulos al reconocedor actual para mejorar su desempeño como modelos escondidos de Markov o cancelación de eco. Finalmente se debe construir una buena interfaz que permita una interacción entre el reconocedor de voz y el GUI del tal forma que se reduzca el espacio de búsqueda de palabras mediante contextos.

Una característica clave para mejorar el reconocimiento de voz es la posibilidad del obtener contexto en el reconocimiento para reducir el espacio de búsqueda de palabras. En el manejo de una interfaz gráfica uno puede reducir fácilmente el número de palabras posibles a reconocer mientras el usuario utilice un esquema de menús dependiendo de la aplicación que tenga activa y de la tarea que este efectuando.

Anteriormente no existía una forma de interacción con el GUI y sólo se podía mandar comandos a través de señales de X. Ahora gracias a GNOME existe la posibilidad de consultar las cadenas de caracteres que componen cada menú de cualquier aplicación que se desarrolle con el *toolkit* de GNOME. De esta forma se puede crear un interacción que limite eficazmente el conjunto de palabras posibles dentro de un contexto para mejorar el reconocimiento.

La interacción se lograría gracias a dos mecanismos que serían transparentes al programador. Por un lado las señales de X generadas por cadenas de caracteres que comunicaría el reconocedor de voz. Por otro lado un contexto que sería generado automáticamente a partir de los menús de la aplicación.

Apéndice A

Guía rápida de usuario de icepick

A.1 Por donde empezar:

Antes de empezar a utilizar el reconocedor de comandos se debe de tener alguna idea empírica de su funcionamiento de tal forma que se entienda plenamente cómo se puede utilizar y las limitaciones que tiene. En las siguiente secciones trataré de explicar los mecanismos que utiliza el reconocedor de comandos para reconocer una palabra aislada.

A.1.1 ¿Qué significa reconocer una palabra aislada?

Este reconocedor de comandos sólo puede reconocer palabras aisladas. Se define una palabra aislada como: una palabra que se dice entre dos pausas. Sucede que cuando uno habla normalmente uno no deja espacios de silencio entre las palabras sino que las encadena una después de otra conforme habla. El reconocedor de palabras necesita que la palabra que uno diga se encuentre entre dos silencios para que pueda reconocer el segmento de señal que tiene que procesar en el reconocimiento.

A.1.2 ¿Cómo funciona?

En este reconocedor de palabras aisladas la palabra a reconocer se debe entrenar para que se reconozca. En el entrenamiento el usuario repite la palabra que quiere entrenar varias veces. Se aconseja un rango de entrenamiento que va desde 3 repeticiones hasta quince repeticiones; en este rango entre más se entrena la palabra se obtiene una mejor probabilidad de ser reconocida.

Después de que el usuario entrena con una palabra a Icepick hace un modelo espectral de ella. Se debe de enfatizar que Icepick toma sólo el **sonido** que fue

emitido entre dos silencios para crear un modelo; Icepick no tiene capacidad alguna de distinguir sílabas, fonemas, consonantes o vocales.

Cuando el proceso de reconocimiento empieza, cada palabra aislada será comparada contra los modelos de las palabras entrenadas, y el modelo que sea el más parecido será seleccionado como el que representa la palabra que se quiere reconocer.

A.1.3 ¿Puedo entrenar palabras en cualquier lenguaje?

Se puede a Icepick entrenar palabras en cualquier lenguaje. Esto se debe a que para reconocer la palabra, Icepick utiliza un modelo que se basa en las propiedades espectrales de los sonidos de la palabra y no utiliza ningún tipo de propiedades fonéticas, sintácticas o semánticas para reconocer la palabra.

Por ejemplo se podría entrenar un grito, un gruñido y una palabra y Icepick reconocería cada uno, sin importar si el grito o el gruñido tienen representación escrita alguna. Se recomienda que no se entrene a Icepick con los sonidos anteriores, dado que la gente podría empezar a tratarlo en forma diferente si lo ve todo el día usando este tipo de vocabulario con la computadora :-). Aunque se puede entrenar a Icepick con cualquier sonido se recomienda que se utilicen palabras largas para un mejor reconocimiento.

A.1.4 ¿Qué sucede si se dice una palabra que no fue entrenada?

Si se dice una palabra con la que no fue previamente entrenado Icepick selecciona el modelo que le parezca más semejante a la palabra dicha, este comportamiento es impredecible desde el punto de vista de el usuario.

A.2 Utilización de Icepick

A.2.1 Entrenamiento de una palabra

Para entrenar una palabra primero se le tiene que asignar un nombre. Se escribe el nombre que se quiere asociar con la palabra a ser entrenada en el campo de entrada que se encuentra en el marco de Trained Words.

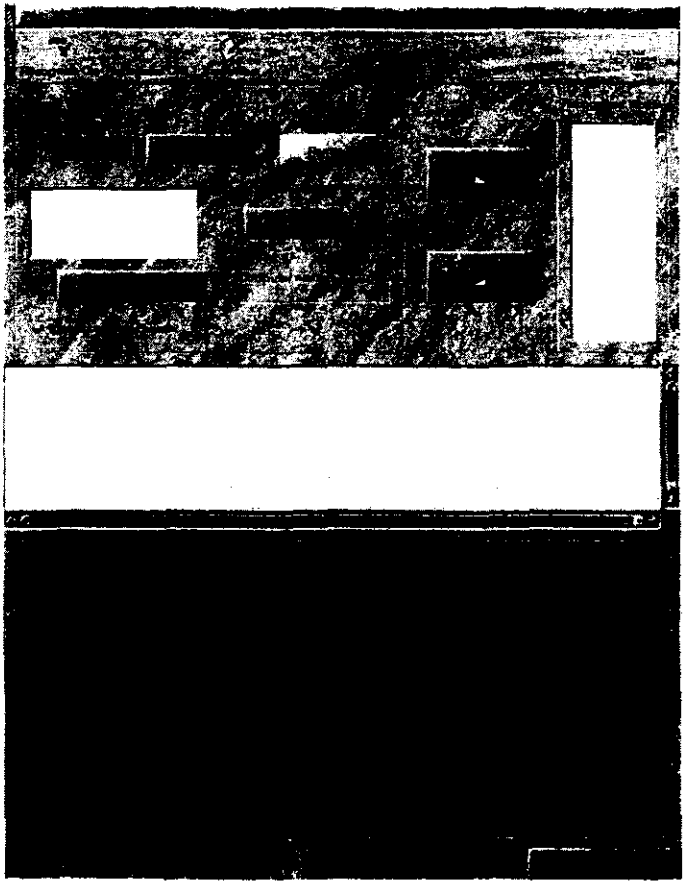


Figura A.1: Ventana principal de Icepick

Después se presiona el botón Add y una nueva ventana aparecerá. Esta es la ventana de entrenamiento donde el usuario repetirá varias veces la palabra a ser entrenada.

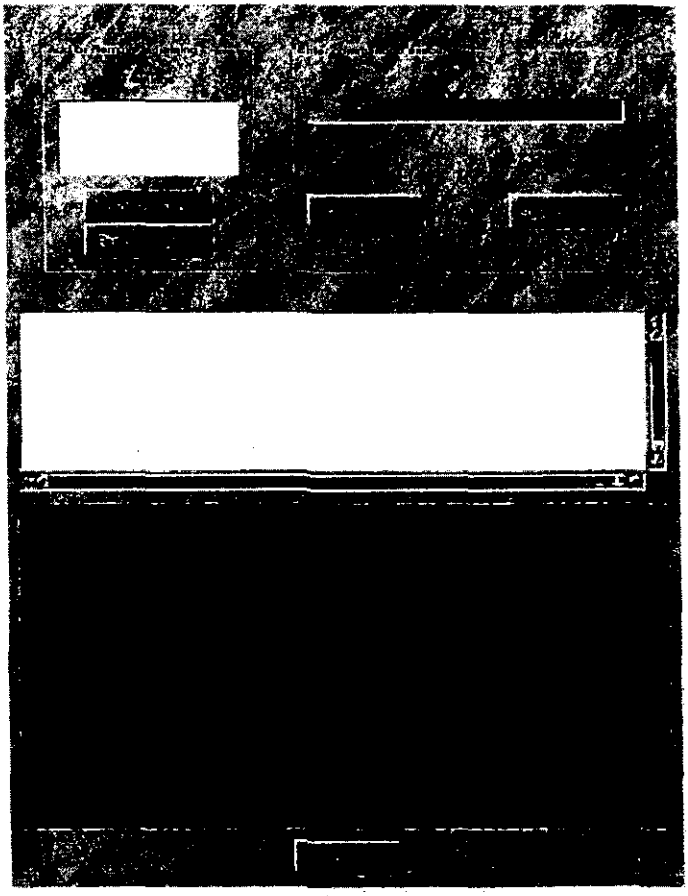


Figura A.2: Ventana de entrenamiento de Icepick (Entrenando la palabra xterm)

En esta ventana se localiza el botón Add Training, cuando se presione este botón se indicará que a continuación se va a captar el sonido de fondo. Se suplica que se mantenga silenció mientras la ventana lo indique (aproximadamente de 2 a 4 segundos).

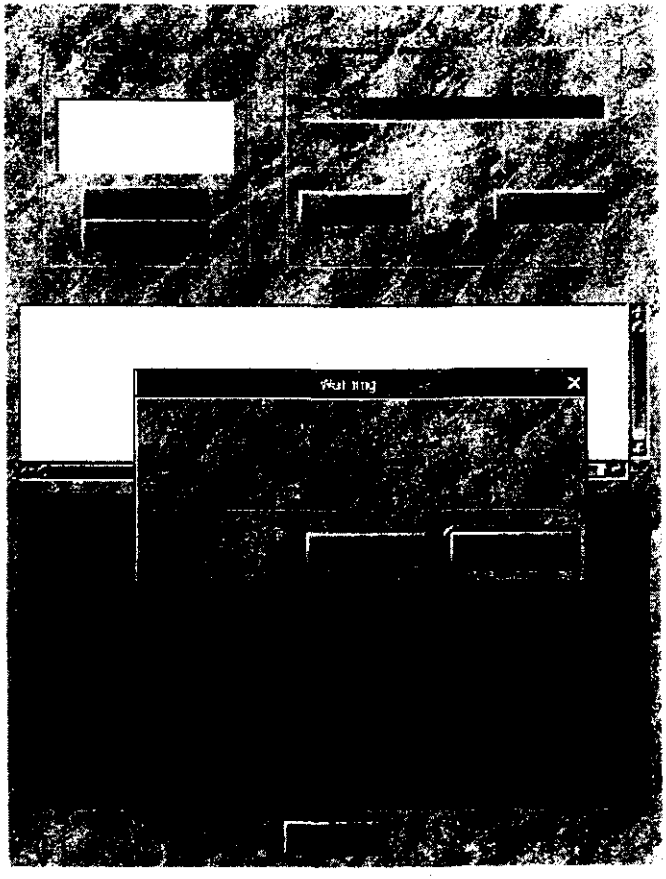


Figura A.3: Ventana de entrenamiento de Icepick mientras se muestra el sonido de fondo

Finalmente una aparecerá otra ventana donde se le indicará de decir la palabra. Se deberá de decir la palabra de la manera más natural posible pero cuidando que se tenga una buena dicción.

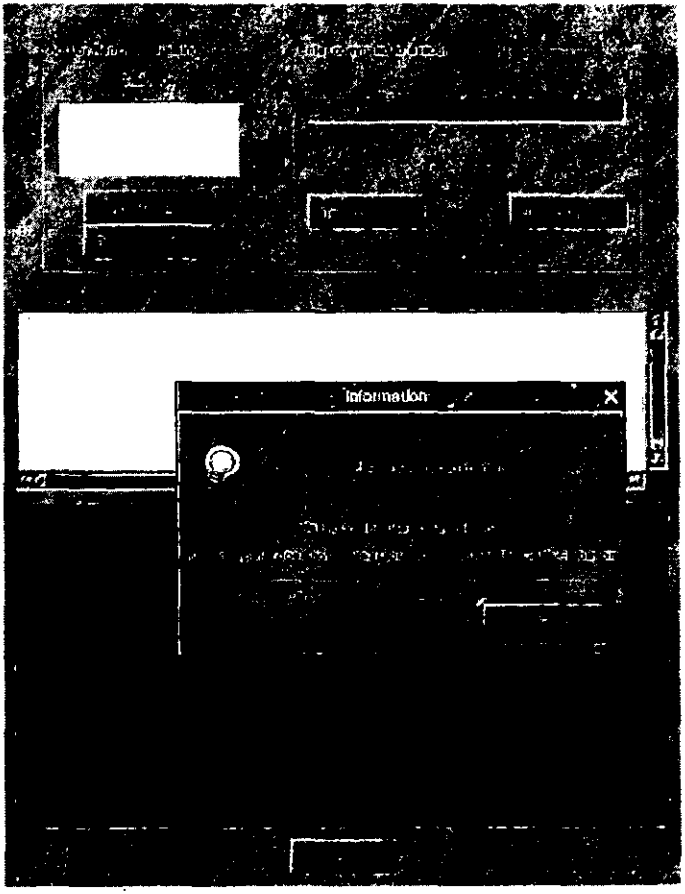


Figura A.4: Ventana de entrenamiento de Icepick mientras se dice una palabra

Después de que se diga la palabra, aparecerá un oscilograma de ésta en la parte inferior de la ventana de entrenamiento. En el oscilograma el usuario podrá notar si la computadora pudo capturar bien la palabra. Después se le preguntará al usuario si quiere volver a entrenar a Icepick con la palabra. Como se señaló anteriormente se recomienda que el usuario entre la palabra entre tres y quince veces.

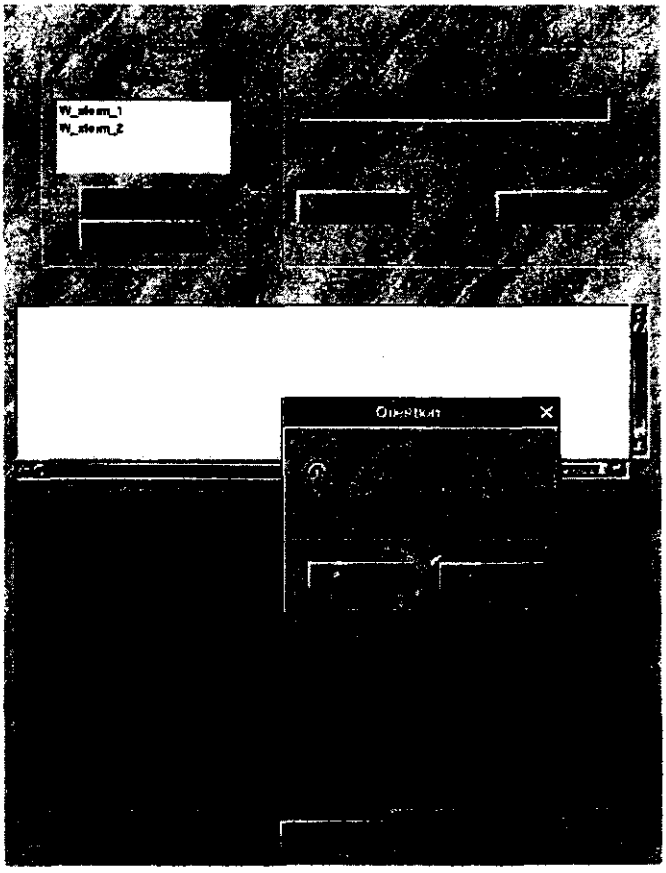


Figura A.5: Ventana de entrenamiento después de que el usuario dijo una palabra

Se puede opcionalmente asignar un comando para que lo ejecute Icepick cuando se reconozca la palabra.
Después de que se acabe de hacer todos los entrenamientos se presiona el botón Done y entonces se creará un modelo de la palabra.

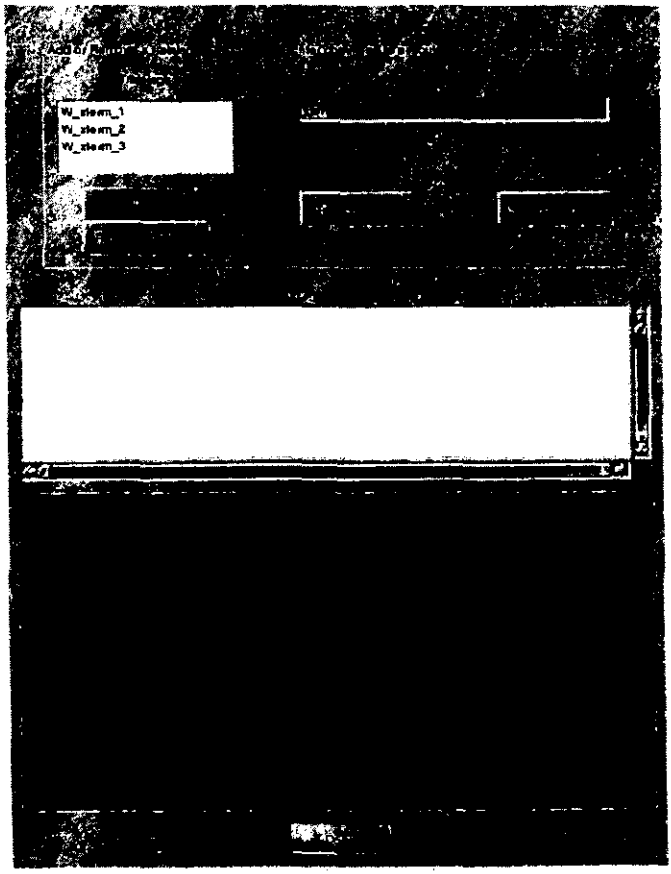


Figura A.6: Ventana de entrenamiento de Icepick, asignando un comando y acabando de entrenar

A.2.2 ¿Cómo empiezo a reconocer?

Después de que se entrenaron las palabras (que se encuentran en la lista del marco que dice Trained words) se deben añadir a la lista de las palabras reconocidas (que se encuentran en la lista del marco que dice Recognized words) para indicarle a Icepick cúales de las palabras entrenadas se quiere reconocer.

Debe haber por lo menos una palabra dentro de la lista de Recognized words para poder iniciar el reconocimiento. Se debe notar que si solo hay una palabra a reconocer, *cualquier* palabra que se diga será asociada a la *única* palabra en la lista de Recognized words. El reconocimiento utiliza una asociación relativa, lo que quiere decir que reconoce la palabra seleccionando el modelo que se parezca más a la palabra dicha. Por lo tanto si sólo hay una palabra en el reconocimiento, entonces su modelo será *siempre* el que se parezca más.

Después de que se seleccionen las palabras que se quiera reconocer con los botones que dicen recognize y Don't recognize, se puede empezar el reconocimiento presionando el botón que dice Recognize on.

Entonces estará esperando que se le digan palabras aisladas, después de calcular el sonido de fondo. Cuando detecte que se le diga algo, desplegará el oscilograma del sonido y imprimirá en el marco de texto la palabra que esta reconociendo. También imprimirá si esta ejecutando algún comando asociado a la palabra si se da el caso.

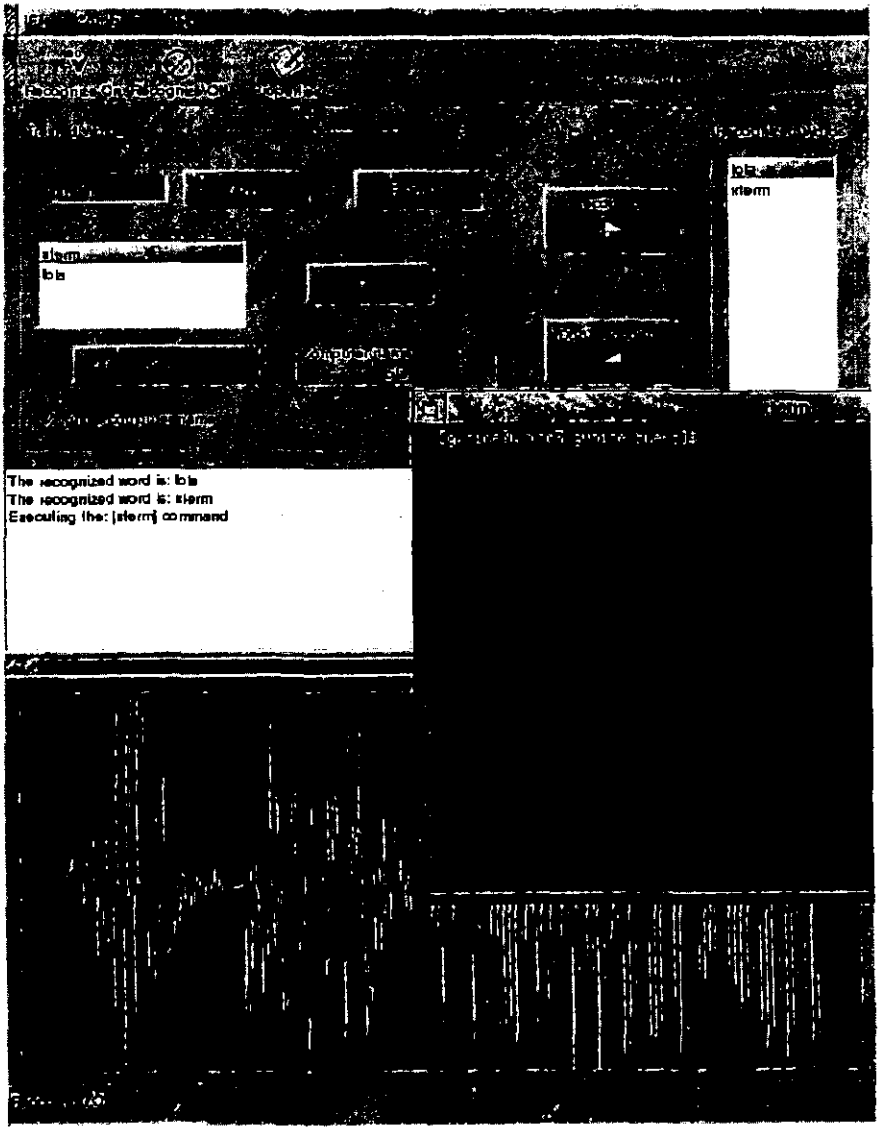


Figura A.7: Ventana principal de Icepick: Reconociendo un comando

Antes de empezar a proceso de reconocimiento se pueden seleccionar un número de opciones:

Ejecución opcional de comandos

En la parte inferior derecha de la ventana principal de Icepick se encuentra un botón que dice Executing commands off. Si se presiona este botón la etiqueta que contiene cambiará a Executing commands on. Este botón desactiva la ejecución de los comandos asociados a las palabras que se reconozcan.

Asignación de nombre a la computadora

Se puede asignar un nombre a la computadora. En el caso que se asigne y que se este en modo de ejecución de comandos los comandos sólo se ejecutarán si previamente se reconoció la palabra de la computadora. En el caso de la siguiente figura se entrenó a Icepick con una palabra que es *xterm*, y la computadora tiene por nombre *Lola*. Entonces solo después de que se diga Lola se instanciará una terminal X.

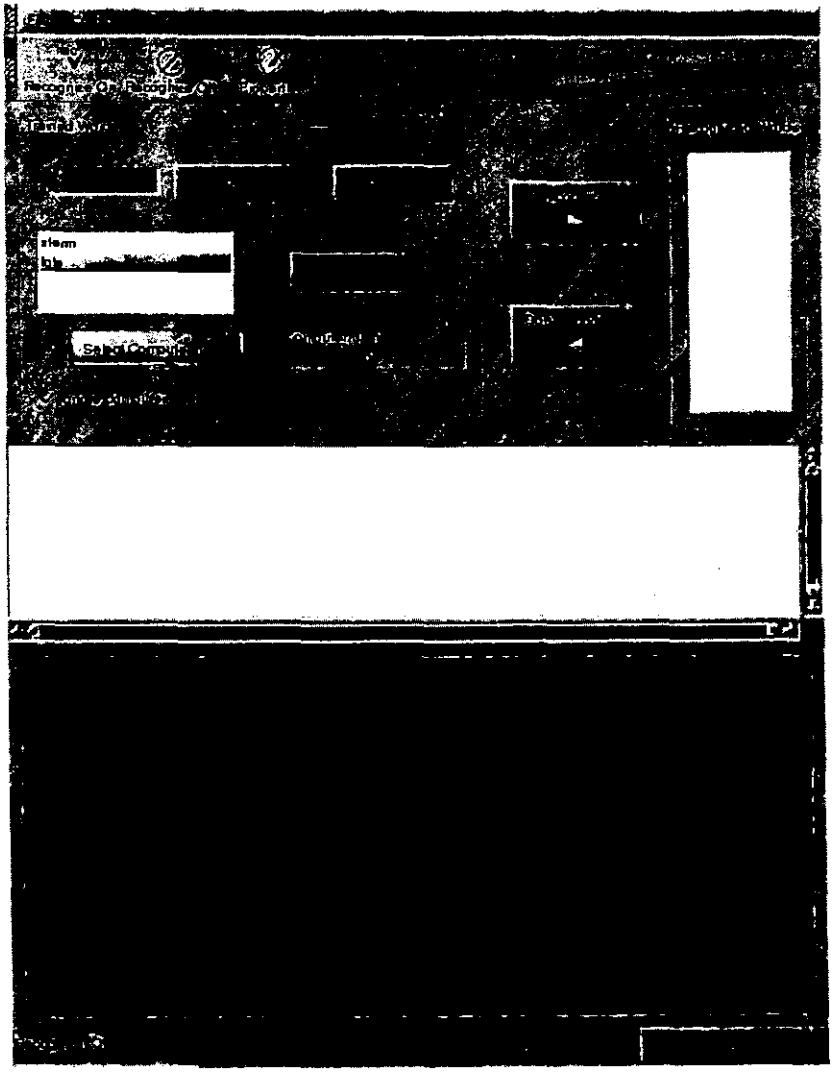


Figura A.8: Ventana principal de Icepick, asignando un nombre a la computadora

Bibliografía

[1] Shawn T. Amundson. The gimp toolkit. HTML WWW, 1999. La documentación de esta biblioteca se puede encontrar en http://www.gtk.org/documentation.html

[2] Shawn T. Amundson. The gimp toolkit. HTML WWW, 1999. Esta página se puede encontrar en http://www.gtk.org.

[3] Jesús Savage Carmona. Implantación de algoritmos de procesamiento digital de señales con micropocesadores. Master's thesis, Universidad Autónoma de México, D.E.P.F.I, 1989.

[4] Jesús Savage Carmona. *A Hybrid System with Simbolic AI and Statistical Methods for Speech Recognition.* PhD thesis, University of Washington, 1995.

[5] Depósitos donde se encuentran varios programas de reconocimiento de voz. HTML WWW, 1999. Existe un sitio de ftp que tiene almacenados varios programas ftp://svr-ftp.eng.cam.ac.uk/pub/pub/comp.speech/recognition y por otro lado existe otra página donde tambien se pueden accesar http://www.isip.msstate.edu/resources/speech/software/index.html.

[6] Simon Crosby. Xtalk. HTML, WWW, 1999. Esta página se puede encontrar en http://www.ai.sri.com/ cheyer/xtalk/xtalk-help.html

[7] G. Fant. *Acoustic theory of speech production.* The Hague, 1970.

[8] J. L. Flanagan. *Speech Analysis, Synthesis and Perception.* Springer-Verlag, 2d edition, 1972.

[9] The Institute for Signal and Information Processing. Large vocabulary conversational speech recognition. HTML WWW, 1999. Esta página se puede encontrar en http://www.isip.msstate.edu/projects/speech_recognition/.

[10] Free Software Foundation (FSF). GNU general public license. HTML WWW, 1999. Esta página se puede encontrar en http://www.gnu.org/copyleft/gpl.html.

[11] Free Software Foundation (FSF). Philosophy of the gnu project. HTML WWW, 1999. Esta página se puede encontrar en http://www.gnu.org/philosophy/philosophy.html.

[12] GNU. Gnu c library. HTML WWW, 99. Esta página se puede encontrar en http://www.gnu.org/software/libc

[13] Rober M. Gray, Andrés Buzo, Augustine H. Gray, and Yasuo Matzuyama. Distorsion measures for speech processing. *IEEE Transactions on Acoustics Speech and Signal Processing*, ASSP-28(4):367–376, August 1980.

[14] Jaroslav Kysela. Advanced linux sound arquitecture. HTML, WWW, 1999. Esta página se puede encontrar en http://alsa.jcu.cz/

[15] Yoseph Linde, Andrés Buzo, and Rober M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, COM-28(1):84–95, January 1980.

[16] J.D. Markel and A.H. Gray. *Linear Prediction of Speech.* Springer Verlag, 1976.

[17] Inc Object Management Group. What is corba ? HTML, WWW, 1998.

[18] L.R. Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition.* Signal Processing Series. Prentice Hall, 1993.

[19] Bob Scheifler. Manual de a2x. FTP, WWW, 1999. El manual de el programa a2x se puede encontrar en ftp://ftp.ai.sri.com/pub/cheyer/a2x.ps.Z

[20] Red Hat Software. Orbit. HTML, WWW, 1998. Esta página se puede encontrar en http://www.labs.redhat.com/orbit/

[21] SRI. Arficial intellligence center, 1990. Esta página se puede encontrar en http://www.ai.sri.com/.

[22] 4Front Technologies. Programmer's guide to oss. HTML, WWW, 1999. Esta página se puede encontrar en http://www.se.opensound.com/pguide/

[23] Tigert. GNOME project. HTML WWW, 1999. Esta página se puede encontrar en http://www.gnome.org.