Enfoque UTE
ISSN: 1390-6542
Universidad Tecnológica Equinoccial

Jaime, Andrango; Estevan, Gómez
Sistema prototipo actuador por comandos de voz utilizando software libre
Enfoque UTE, vol. 7, núm. 2, 2016, Abril-Junio, pp. 41-54
Universidad Tecnológica Equinoccial

DOI: https://doi.org/10.29019/enfoqueute.v7n2.94

Disponible en: https://www.redalyc.org/articulo.oa?id=572261569004

Enfoque UTE, V.7-N.2, Jun.2016, pp.41 - 54
http://ingenieria.ute.edu.ec/enfoqueute/
e-ISSN: 1390-6542 / p-ISSN: 1390-9363
Recibido (Received): 2016/04/29
Aceptado (Accepted): 2016/06/23

Sistema prototipo actuador por comandos de voz utilizando software libre

(Actuator prototype system by voice commands using free software)

Andrango Jaime[1], Gómez Estevan [2]

Resumen:
El presente sistema prototipo es una aplicación informática que mediante la utilización de técnicas de procesamiento digital de señales, extrae información de la voz del usuario, la cual se utiliza para administrar la activación/desactivación de un actuador periférico del computador personal, cuando el usuario pronuncia las vocales. Se aplica el método de diferencias espectrales. Para el aplicativo se utiliza como actuador aquella información registrada en la dirección de memoria 378H; es decir, el puerto paralelo. La propuesta se ha desarrollado haciendo uso de herramientas de software libre, con la finalidad de dar apertura para que otros investigadores puedan tomar este trabajo como base para otros estudios en fases posteriores y por la versatilidad y dinamismo en las herramientas de la programación de software libre.

Palabras clave: reconocimiento de voz; comandos de voz; diferencias espectrales; Python; aplicaciones de software libre

Abstract:
This prototype system is a software application that through the use of techniques of digital signal processing, extracts information from the user's speech, which is then used to manage the on/off actuator on a peripheral computer when vowels are pronounced. The method applies spectral differences. The application uses the parallel port as actuator, with the information recorded in the memory address 378H. This prototype was developed using free software tools for its versatility and dynamism, and to allow other researchers to base on it for further studies.

Keywords: voice recognition; voice commands; spectral differences; python; free software applications

1. Introducción

El propósito fundamental de este trabajo es estudiar el reconocimiento de voz a través de la creación de un software que permita dicho reconocimiento, utilizando un método comparativo entre la señal pronunciada por el usuario (capturada mediante el micrófono) y otra que se mantiene en una base de información tomada como patrón.

Se parte del hecho de que el espectro en frecuencia de una señal contiene información que en general la diferencia de otra (Bernal, 2000), (Poor, 1985); a pesar de que los autores pueden tener cierta diferencia en sus apreciaciones, en general confluyen en que es posible utilizar un patrón

[1] Universidad de las Fuerzas Armadas, Quito – Ecuador (jfandrango@espe.edu.ec)
[2] Universidad Tecnológica Equinoccial, Quito – Ecuador (estevan.gomez@ute.edu.ec)

del tratamiento de ésta información. En este sentido es de capital importancia encontrar técnicas que permitan obtener patrones invariantes para lograr identificación de lo pronunciado.

“El reconocimiento por voz o parlante, es una modalidad biométrica que utiliza la voz de un individuo con fines de reconocimiento (Difiere de la tecnología del "reconocimiento de discurso", que reconoce las palabras a medida que van siendo articuladas; este no es un dispositivo biométrico)” (Argentina & Biométricos, 2016).

(Sawada, 2014), propone un campo aleatorio condicional (CRF) basado en el enfoque de re-clasificación, que recalcula puntuaciones de detección producidas por un enfoque basado en fonemas de deformación dinámicos en el tiempo (DTW) conocido como STD. Utiliza modelos de detección basados en trifono CRF considerando las características generadas a partir de varios tipos de transcripciones basadas en fonemas. Se entrenan los patrones de error de reconocimiento, tales como confusiones-fonema a fonema en el marco de CRF. Por lo tanto, los modelos se pueden detectar en un trifono, que es uno de trifonos que componen un término de consulta, con la probabilidad de detección.

Como antecedente, se hace referencia a iniciativas existentes, como los trabajos realizados en (García & Tapia, 2000), (Thomas, Pecham, & Frangoulis, 1989), (Thomas T. , Pecham, Frangoulis, & Cove, 1989), que pretenden identificar una frecuencia fundamental media en el espectro de la señal, obteniéndose así una tendencia a lo largo de la función espectral, que representa a lo pronunciado. El fundamento teórico de las diferencias espectrales pretende obtener información a lo largo del espectro de la señal de voz, creando un método comparativo que logra definir adecuadamente la forma de la función espectral; este mecanismo diferencial establece un factor diferencial entre la señal capturada y el patrón definido, y mientras el diferencial tienda a cero, se habrá logrado un reconocimiento más exacto, es decir que la probabilidad de que la señal capturada versus el patrón sean iguales estará sobre el 90%.

En esta primera fase se analizará la tasa diferencial de la muestra con respecto al patrón; la importancia de la utilización del software libre (para el desarrollo del software) está dada por la utilización de librerías (tanto en C++ como en Phyton) de fácil comprensión y mantenimiento, lo que favorece la oportunidad de trabajos y aplicaciones futuras. Se proyecta para una segunda fase, la obtención de la envolvente del espectro tanto de la señal pronunciada como la utilizada como patrón para la comparación. En esta fase también se espera utilizar una técnica que permita considerar la desviación de estas dos señales. Para cumplir con este objetivo, se crea un prototipo, el mismo que es una aplicación informática, con la funcionalidad de activar y desactivar independientemente los pines del actuador (puerto paralelo del computador) mediante la identificación de una de las cuatro vocales (<A>, <E>, <I>, <O>) pronunciadas por el usuario a través del micrófono del computador personal. A las salidas de los pines del actuador se han conectado diodos led con la finalidad de poder observar la activación (encendido) y desactivación

(apagado) de los mismos. Se ha desarrollado sobre un sistema operativo Linux Debian, utilizando como lenguaje de programación a Python (Phyton, Python GUI Programming (Tkinter), 2016), por la potencialidad que brindan sus librerías de aplicación matemática, además de la posibilidad de generación de ambientes en modo gráfico, y porque permite la integración con librerías desarrolladas por el usuario en código nativo, utilizando el lenguaje C ó C++. Al inicio se utilizó la herramienta Octave (símil a Matlab), pero sobre la marcha se encontró con el inconveniente de que no brinda la posibilidad integrada para crear interfaz gráfica, ni tampoco cuenta con la manipulación del puerto paralelo o usb; es por ello, que se optó en desarrollarlo en Python que sí cuenta con esas herramientas y además dispone de cálculos avanzados con los módulos numpy y scipy, según se tratan en (Hans, 2011), entre otras. La interfaz gráfica se desarrolló con el módulo Tkinter de Phyton (Phyton, Python GUI Programming (Tkinter), 2016) . El módulo para escribir en el puerto paralelo e integrarlo en Python, se desarrolló en el lenguaje C (compilador gcc). Se desarrollaron con la información de (Grayson, 2000).

La comparación de patrones no se efectúa en el dominio temporal sino en la frecuencia; en el desarrollo del sistema se ha considerado la capacidad de detección automática de los momentos en que el usuario empieza a hablar en el micrófono y cuando deja de hacerlo; de forma que la captura de la señal y el almacenamiento corresponda al intervalo en el que realmente habla el usuario.

2. Metodología

A continuación se detalla la propuesta de sistema describiendo la plataforma de desarrollo, el módulo para escribir sobre el puerto 378H, el diseño del aplicativo, y la codificación utilizada para el desarrollo del prototipo.

A. Plataforma de desarrollo

El aplicativo se desarrolló, en la versión 2.7 de Python para Linux Debían (versión Wheezy a 64 bits). Como editor de desarrollo (IDE), se utilizó el programa IDLE (Phyton, Idle Phyton, 2016) . En Python, se utilizaron las librerías Tkinter (Phyton, Python GUI Programming (Tkinter), 2016), (para crear la interfaz gráfica), numpy (que proporciona herramientas avanzadas de cálculo matemático avanzado, similar a Matlab), matplotlib (herramientas para graficación 2D y 3D), pyaudio (permite captura del stream de audio tomada desde el micrófono) y Sndfile y play del audiolab (para la reproducción de audio.wav). También se instaló la librería scikits.audiolab, que puede leer en varios formatos y entrega una estructura numpy. La instalación se realizó como usuario no administrador, por lo que se usó sudo.

B. Módulo para escribir sobre el puerto 378H

Se creó un nuevo módulo para Python (Phyton, Python GUI Programming (Tkinter), 2016), con código en lenguaje C, para la manipulación del puerto paralelo y/o usb. Se crea el archivo pPar.c, como librería externa en lenguaje C. En este archivo, se define la función escribirPuerto(), que recibe dos parámetros enteros, correspondientes al valor que se escribirá en el puerto, y la dirección del mismo (normalmente la dirección 378 en hexadecimal). No se desarrolló una función de lectura, por no requerirse en el aplicativo. La compilación se realizó con la herramienta gcc, de la siguiente manera:

gcc pPar.c -o pPar.so -fPIC -shared -I/usr/include/python2.7

Este proceso genera como resultado el archivo pPar.so, que se copia en la carpeta: /usr/lib/python2.7. Entonces, puede llamarse a la función escribirPuerto() mediante un import al módulo pPar desde una aplicación en python. Detalles de archivo fuente en Tabla 1.

Tabla1. Detalles de Archivo fuente

```
# Archivo fuente pPar.c
 #include <Python.h>
 #include <sys/io.h>
 #include <unistd.h> //para el tiempo
 #include <sched.h> //para la prioridad
static PyObject* pPar_escribirPuerto(PyObject *self, PyObject *args)
{
 int valorPuerto, dirPuerto, estado;      estado=PyArg_ParseTuple(args, "ii", &valorPuerto,
&dirPuerto); if(!estado)  return NULL;

   printf("Valor v:%d, Puerto:%d, estado:%d", valorPuerto, dirPuerto, estado);
struct sched_param sp;
   /* Cambia la prioridad del proceso a tiempo real */
sp.sched_priority=10;
  sched_setscheduler(getpid(), SCHED_RR, &sp);
  fprintf(stderr,"Obteniendo acceso al puerto paralelo.\n");
  if (ioperm(dirPuerto,1,1)) //HABILITO
     { fprintf(stderr,"Error obteniendo acceso al puerto paralelo.\n");     return NULL;
     }
  fprintf(stderr,"Acceso al puerto paralelo.\n");   outb(valorPuerto,dirPuerto);

  if (ioperm(dirPuerto,1,0)) //DESHABILITO
   { fprintf(stderr,"Error obteniendo acceso al puerto paralelo.\n");      return NULL;
   }
  printf("\nTERMINO PROCESO ACCESO PUERTO\n");
  Py_RETURN_NONE;
}
 static PyMethodDef pPar_methods[ ] = {
{"escribirPuerto", pPar_escribirPuerto, METH_VARARGS,
"Documentación pPar.escribirPuerto(int valorPuerto, int direccionPuerto)"},
{NULL, NULL, 0, NULL},/*sentinel*/
} ;
```

C. Diseño del aplicativo

El usuario interactúa con la aplicación móvil a través de la interfaz gráfica como muestra la Figura 1.

Figura 1. Interfaz de usuario

Funcionalidad

Para iniciar el proceso de reconocimiento, se hace clic en el botón Reconocer. Aparece una caja de diálogo que brinda ayuda sobre la captura del audio a reconocer según se indica en la Figura 2.

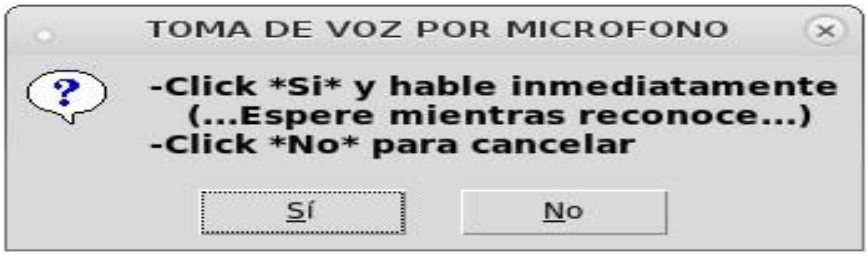


Figura 2. Caja de diálogo que confirma inicio de captura de voz.

Se da clic en el botón “Sí” para iniciar la captura de voz. Cabe indicar que se inicia la grabación únicamente en el momento que se detecte la presencia de señal; antes de ello, el sistema estará latente y no se grabará la voz. Si se detecta un falso disparo en el micrófono, en la etiqueta lblVisor se muestra el mensaje —Repita..

Inmediatamente, se iniciará el procesamiento digital de reconocimiento. En la etiqueta lblVisor, se observará la letra que fue pronunciada en el micrófono. En las etiquetas lblLedRojo y lblLedVerde, se reflejará gráficamente la acción de encendido/apagado de los diodos led rojo y verde, correspondiente a la vocal pronunciada, en conformidad a lo indicado en la Tabla 2.

En el caso de encontrarse habilitada la casilla de verificación chkPuerto, se iluminarán de manera más intensa los led de las etiquetas; además, que se encenderán (o apagarán) físicamente los dos diodos led conectados a los pines 2 y 3 del puerto paralelo. El diagrama circuital de conexión al puerto paralelo puede observarse en la Figura 3.

Tabla 2. Comandos de voz: acción.

LETRA PRONUNCIADA	LED	ESTADO
<**A**>	Color rojo	Encendido
<**E**>	Color rojo	Apagado
<**I**>	Color verde	Encendido
<**O**>	Color verde	Apagado
<**U**>	Rojo y Verde	Ambos apagados

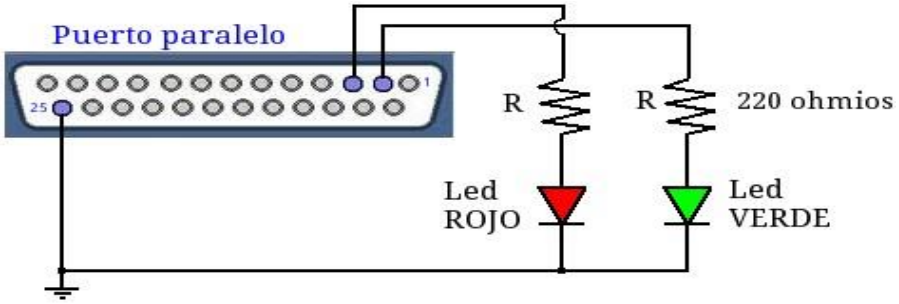
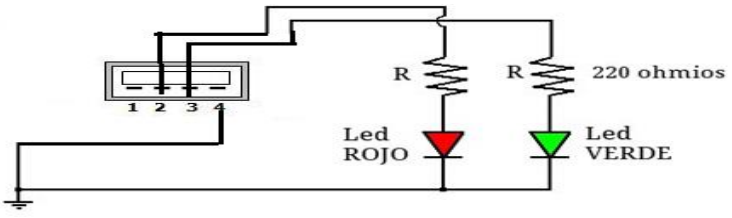


Figura 3. Diagrama circuital. Conexión puerto paralelo a diodos led

Adicionalmente se debe mencionar que el puerto USB ha ido poco a poco reemplazando a otros puertos entre ellos el puerto paralelo; con una estructura muy sencilla, destaca por su gran velocidad de transferencia $480\ \mathrm{Mbit/s}$ con la especificación USB 2.0 (How, 2016). El puerto USB tiene dos canales de transmisión de datos y uno de punto tierra, como se muestra en la Figura 4 para realizar la conexión de igual forma que con el puerto paralelo, de forma tal que la solución se puede trasladar para la utilización del puerto USB sin ningún inconveniente. En cuanto se refiere al software se aplica el mismo criterio, y se realizarán los cambios respectivos de llamado a las funciones pertinentes.



Pin	Señal	Color	Descripción
1	Vcc		+5V
2	D-		Data-
3	D+		Data+
4	GND		Masa

Figura 4. Diagrama circuital. Conexión puerto USB a diodos led

En la Tabla 3, se describe la funcionalidad con sus métodos y variables asociadas a los controles que forman parte de la interfaz de usuario.

Tabla 3. Funcionalidad de los controles de interfaz

CONTROL	FUNCIONALIDAD
btnReconocer t: Button m: Reconocer(), y Reconocer1()	En la función Reconocer( ), se llama a la función Record_to_file('miVoz.wav') del módulo grabarNumpyTerminado.py. Esta última captura la señal de voz desde el micrófono del PC en el instante mismo cuando el usuario empieza a hablar (antes no), la normaliza, e inserta silencios al inicio y al final de la señal capturada. Finalmente, se almacena en el archivo "miVoz.wav" en formato .wav. También controla **falsos disparos en el micŕofono**, en tal caso, muestra el mensaja "Repita" en lblVisor. El método Reconocer1( ), se encarga de hallar la FFT y valor absoluto de la señal almacenada en el archivo "miVoz.wav"; luego, mediante resta de arreglos, se realiza una comparación con sus similares (FFT y valor
	absoluto) de las vocales <A>, <E>, <I> y <O> (previamente calculadas en e método Inicio( )); resultados, a los cuales se halla el valor absoluto y la media. El texto reconocido corresponde, al valor menor de las medias halladas. A través de la función escribirPuerto( ) del módulo pPar.py se enciende o apaga el led correspondiente.
btnReproducir t: Button m: Reproducir()	Llama al método Reproduce( ) del módulo canta.py, el cual permite reproducción de audio, del archivo pasado como parámetro. Al final se grafica la señal reproducida.
chkPuerto t: Checkbutton v: selPuerto m: ActivaPuerto()	Este método habilita o deshabilita las etiquetas lblLedRojo y lblLedVerde como un indicativo visual de activación/deshactivación del puerto paralelo. El método Reconocer1( ) consulta si el control chkPuerto esta deshabilitado o no, para acceder al puerto.
btnSalir t: Button m: Salir()	Termina el aplicativo; previo, encera el puerto.
btnAcerca t: Button m: AcercaDe()	Caja una carga de diálogo, con los datos informativos del aplicativo.
lblL2 t: Label	Etiqueta en la que se tutoría al usuario respecto a la vocal con la que se enciende (vocal <I>) o apaga (vocal < O >) el led de color verde.
lblL1 t: Label	Etiqueta en la que se tutoría al usuario respecto a la vocal con la que se enciende (vocal <A>) o apaga (vocal <E>) el led de color rojo.
lblLedVerde t: Label	Etiqueta en la cual se carga las imágenes imgLedVerde (carga el archivo: ledVerde_ON.ppm) ó imgLedOFF (carga el archivo: led_OFF.ppm), según corresponda el encendido/apagado del led de color verde.
lblLedRojo t: Label	Etiqueta en la cual se carga las imágenes imgLedRojo (carga el archivo: ledRojo_ON.ppm) ó imgLedOFF (carga el archivo: led_OFF.ppm), según corresponda el encendido/apagado del led de color rojo.
lblVisor t: Label v: Visor (tipo StringVar)	Muestra los textos: "Letra_A", "Letra_E", "Letra_I", "Letra_O", de acuerdo al reconocimiento de la vocal pronunciada por el usuario.

La codificación de la interfaz de usuario, en modo gráfico, fue realizada mediante widgets Tkinter (Phyton, Python GUI Programming (Tkinter), 2016).

D. Codificación- Detalles técnicos.

Se trabajó con una frecuencia de muestro de 44100 Hz, canal mono, con lectura de tramas de 1024 bytes, y archivos de audio en formato —.wavll, con cuantización a 16 bits.

D.1 Adecuación de la señal capturada del micrófono:

Con la finalidad de optimizar el proceso de captura y almacenamiento de la señal captada desde el micrófono, se realizaron los siguientes procedimientos: 1) Detección automática de los momentos en que el usuario empieza a hablar en el micrófono y cuando deja de hacerlo; de forma que la captura de la señal y el almacenamiento corresponda a la señal en el intervalo cuando realmente habla. Para ello, en cada trama (de 1024 bytes) se analiza si se ha superado el nivel empíricamente establecido del THRESHOLD (500) a través del método is_silent( ). Este método resulta afirmativo cuando se supera el THRESHOLD de 500, y esto pasará cuando el usuario empiece a hablar. A partir de este momento, se concatenan las posteriores capturas del micrófono; parándose cuando no se supere nuevamente el THRESHOLD establecido por el lapso de más de 20 tramas capturadas, siendo este un indicativo de que el usuario paró de hablar, como se muestra en la Tabla 4.

Tabla 4. Codificación Adecuación de la señal capturada del micrófono

```
.....

p = pyaudio.PyAudio( ) stream = p.open(format=FORMAT, channels=CANALES,
rate=RATE, input=True, output=True, frames_per_buffer=CHUNK_SIZE)

num_silent = 0
snd_started = False
r = np.array([],np.int16)

while 1:
   snd_data = np.fromstring(stream.read(CHUNK_SIZE),np.int16)
   if byteorder == 'big':    snd_data.byteswap()
silent = is_silent(snd_data)
   if not silent and not snd_started: # si no silencioso y no inicio
 snd_started=True
      r=np.concatenate((r,snd_data))
 if not silent and snd_started:
   r=np.concatenate((r,snd_data))
if silent and snd_started:
      num_silent+=1
 if snd_started and num_silent > 20:
 # posibilidad que el usuario paró de
        #hablar
  num_silent=0
```

2) Se realiza un recorte de los espacios en blanco al inicio y final. El método Trim(snd_data), ejecuta esta actividad. 3) Luego del proceso anterior se procede a normalizar la señal, mediante el método normalize(snd_data), que toma como referencia la constante MAXIMUM = 16384. 4) Posteriormente, se añaden (vía concatenación del vector que contiene información de la señal de audio) silencios al inicio y al final del stream total capturado, mediante el método add_silence(snd_data, seconds). El tiempo de silencio se pasa como segundo parámetro (en segundos). En pruebas, con Ts=0.3 segundos se logró una mejor respuesta. Con Ts=0.1 segundos, el tiempo de reconocimiento se acorta pero se tienen menos aciertos en el reconocimiento. 5) Finalmente, las muestras se graban en archivo, para lo cual se hace uso de la funcionalidad del módulo wave. En la Figura 4, se puede observar un ejemplo de la señal correspondiente a la vocal <I>) luego del tratamiento que se aplica a la señal capturada del micrófono, utilizando el procedimiento que antecede.

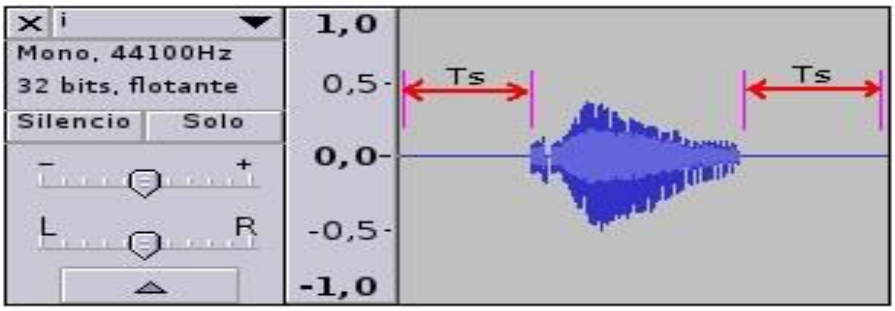


Figura 4. Señal capturada y tratamiento inicial.

E. Reconocimiento de vocales pronunciadas por el usuario:

La técnica de reconocimiento utilizada consiste en aplicar un método comparativo entre la señal pronunciada por el usuario (capturada mediante el micrófono) y otra que se mantiene en una base de información tomada como patrón (Tabla 5)

Tabla 5. Código para reconocimiento de vocales

El método FFTyABS(ARCHIVO) del módulo transforma.py, devuelve la transformada rápida de Fourier (FFT) normalizada, algoritmo básico para el proceso de reconocimiento.

```
def FFTyABS(ARCHIVO): #(método del módulo transforma.py)
sound=Sndfile(ARCHIVO,'r')    n1 = sound.nframes    data =
sound.read_frames(n1)
   yfft = np.abs(np.fft.rfft(data, data.size))
#Se obtiene la FFT de la señal y    #su módulo
   yfft1 = yfft/float(max(yfft))
  #Se normaliza
  sound.close( )
   return yfft1[1:6000] #mejora, se puede bajar el tono de voz
```

Cabe acotar que la comparación no se efectúa en el dominio temporal sino en la frecuencia. Dicha comparación se efectúa utilizando el método de diferencias espectrales entre las señales que se comparan. Algorítmicamente, se hace un barrido comparativo espectral de la señal capturada del micrófono ('miVoz.wav') con las señales patrón almacenadas ('a.wav', 'e.wav', 'i.wav', 'o.wav, 'u.wav'). Las gráficas temporales de las señales patrón se muestran en la Figura 5, las cuales se obtuvieron con el propio programa y utilizando la voz del usuario.

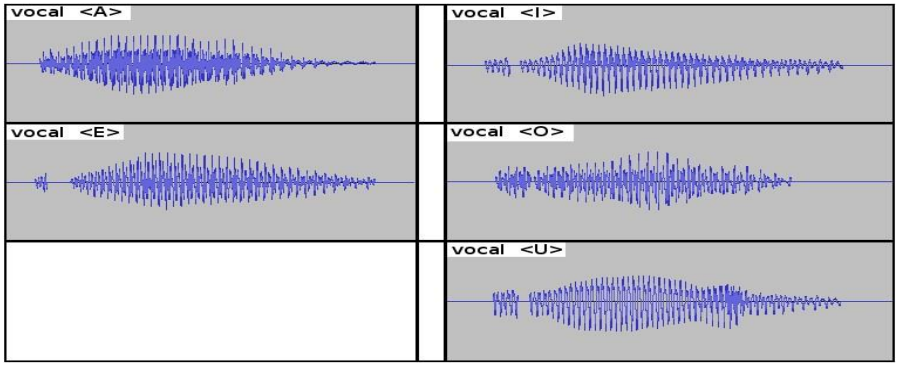


Figura 5. Vocales patrón en el dominio del tiempo

Se utilizó el programa en software libre Audacity (Audacity, 2016) para la edición de estas señales temporales. Es importante observar que el lapso de tiempo de estas señales es diferente. Esta variante temporal, determina que el resultado de aplicación de la FFT también varíe en el número de muestras en el dominio de la frecuencia; por tanto dependerá del tiempo que el usuario demore en pronunciar cada vocal. En este sentido es necesario acotar los arreglos que contienen las muestras en frecuencias a un número de muestras igual en todas las vocales, para efectos de poder realizar la resta con el arreglo que contiene la información espectral de la señal pronunciada en el micrófono a reconocer; ya que deben tener la misma dimensión para el cálculo. En la Figura 6, se indica la gráfica de la FFT normalizada de las cinco vocales patrón. Por simple observación, la información espectral significativa aparentemente se encuentra hasta una trama de aproximadamente 4000 muestras (Tabla 6).

Figura 6. Espectro normalizado de la vocales patrón (frecuencia)

Tabla 6. Comparación espectral de la voz

```
def Reconocer1( ):
 miVoz = transforma.FFTyABS('miVoz.wav')  v = np.zeros(5)
letra = np.array(["Letra_A","Letra_E","Letra_I","Letra_O","Letra_U"])
v[0] = np.mean(np.abs(vocalA-miVoz))
v[1] = np.mean(np.abs(vocalE-miVoz))  v[2] = np.mean(np.abs(vocalI-miVoz))
 v[3] = np.mean(np.abs(vocalO-miVoz))  v[4] = np.mean(np.abs(vocalU-miVoz))
 elegida = letra[ v.argmin( ) ] #el método argmin( ) detecta el menor valor de un   arreglo.
 print ("La letra reconocida es: ",elegida)  visor.set(elegida) #Se muestra la letra
reconocida en la etiqueta lblVisor

 global dato #En la variable dato se almacena el último estado del puerto paralelo.
 if elegida=="Letra_A":
   lblLedRojo.configure(image=imgLedRojo)

   dato= (dato & 1) | 2 #dato = (dato AND 00000001) OR (00000010), solo
                        #se enmascara encendido led Rojo

 if elegida=="Letra_E":
   lblLedRojo.configure(image=imgLedOFF)     dato=(dato & 1) #dato = (dato AND
00000001), solo se enmascara
                      #apagado del led Rojo

 if elegida=="Letra_I":
   lblLedVerde.configure(image=imgLedVerde)     dato=(dato & 2) | 1 #dato = (dato AND
00000010) OR (00000001), solo                    #se enmascara encendido led
Verde

 if elegida=="Letra_O":
   lblLedVerde.configure(image=imgLedOFF)     dato=dato & 2 #dato = (dato AND
00000001), solo se enmascara
                     #apagado del led Verde

 if elegida=="Letra_U":
   lblLedRojo.configure(image=imgLedOFF)
lblLedVerde.configure(image=imgLedOFF)     dato=0
 print "dato: ",dato

 if selPuerto.get( )==1:     try:
        pPar.escribirPuerto(dato,0x378)     except:
       print ("Error en el Puerto. Verifique")
```

Utilizar las 12000. (Nf) muestras (en la comparación espectral de la señal capturada y los patrones) no mejora notablemente el reconocimiento, pero si genera mayor tiempo en el proceso. Empíricamente, con 6000 muestras trabaja aceptablemente. Con 4000 muestras, se tiene mayor tasa de error en el número de aciertos en reconocimiento. Mediante la función Reconocer1( ), se realiza la comparación espectral de la voz pronunciada por el usuario y las vocales consideradas patrón, que fueron grabadas anteriormente por el mismo usuario. La comparación es efectuada

por la operación de resta de los arreglos, en las que se almacenan los datos espectrales de cada una de las vocales (vocalN-miVoz), resultado al cual también se hace necesario hallar el módulo (abs) para no tener que trabajar con imaginarios. En el arreglo v[ ], de cinco elementos, se almacena el promedio de los módulos de las diferencias espectrales de cada vocal. El elemento del arreglo v[ ] que contenga el menor valor, corresponde a la vocal pronunciada por el usuario; siendo así, como se hace el reconocimiento. Finalmente, se realiza enmascaramientos de la variable dato (que contiene la información del último valor escrito en el puerto paralelo) con la finalidad de que la modificación actual de uno de los diodos led, no afecte al estado anterior del otro.

3. Resultados y Discusión:

Se diseñó la experimentación para determinar la relación de fallos en la identificación de la vocal pronunciada respecto a la tasa de muestreo seleccionada; para lo cual, como primera acción se graba las vocales que servirán como patrón; luego se realizan veinte pruebas con una determinación aleatoria de las vocales a pronunciar. Para el efecto, se establecen tres grupos de prueba, determinados por la tasa de muestreo: 4000 Hz, 5000 Hz, y 6000 Hz.

Bajo las mismas condiciones (de cuando se tomaron las vocales patrón) de configuración del control de volumen del micrófono, el mismo tono y duración en la pronunciación de las vocales, la tasa de fallos en reconocimiento experimentado se indica en la Tabla 7.

Tabla 7. Tasa de fallos por número de muestras.

Nf (Número de muestras espectrales)	Número fallos en 20 pruebas	Tasa de fallos
4000	7	35%
5000	5	25%
6000	3	15%

La tasa de fallos se incrementa aproximadamente en un 5%, cuando se captura la voz de una persona diferente a la que generó las vocales que se usan como patrón.

Esta tasa se incrementa notoriamente cuando el usuario prolonga la pronunciación de la vocal más allá del doble de tiempo respecto al utilizado en las señales patrón, o en su defecto cuando se pronuncia de forma débil.

4. Conclusiones y Recomendaciones

El sistema de aplicación informática para administrar la activación/desactivación de un actuador periférico del computador personal mediante el reconocimiento de las vocales de voz se ha desarrollado, y se encuentra funcionando. Al aplicar el método de diferencias espectrales,

utilizando un muestreo de 4000 Hz, se genera mayor error en el reconocimiento; esto sucede cuando el usuario pronuncia en tono normal (no alto) o un tanto diferente a las señales de las vocales que se utilizan como patrón de comparación; Generalmente, en los tres grupos de prueba, se observa que no se reconoce la letra <O>, confundiéndose con la letra <U>. Con 5000 muestras, no se presenta ese problema, pero se tiene que alzar el tono de la voz al pronunciarla. Con 6000 muestras, permite bajar el tono de la voz.

Para una segunda fase, se recomienda la obtención de la envolvente del espectro tanto de la señal pronunciada como la utilizada como patrón para la comparación. Es importante que también se considere la desviación de estas dos señales que se comparan.

Otra alternativa recomendable a experimentar, en esta línea de reconocimiento por señal patrón, sería la aplicación de la transformada wavelet, con la desventaja que esta técnica requiere de un considerable esfuerzo computacional

Bibliografía

A. Larcher, K.-A. L. (2013). “Phoneticallyconstrained PLDA modeling for text-dependent speaker verification with multiple short utterances”. *IEEE Internationall Conference on Acoustics, Speech, and Signal ,Processing (ICASSP)*.

Audacity. (29 de 04 de 2016). *Audacity Español*. Obtenido de Audacity Español: http://audacity.es/

Bernal, J. (2000). *Reconocimiento de Voz y Fonética Acústica.* Madrid: Ra-Ma.

Chen, Y. L.-M. (2015). “Locallyconnected d and convolutional neural networks for smalL footprint speaker recognition,”. *Interspeech*.

García, C., & Tapia, D. (2000). Estudio de la Frecuencia Fundamental de la Voz y de sus Efectos en el. *Proyecto de Fin de Carrera E.T.S.I.* Madrid, España: U. Politécnica de Madrid.

Grayson, J. (2000). *Phyton and Tkinter Programming*‖*, .* Manning Publications Co.

H. Aronowitz, R. H. (2011). “New developments in voice biometrics for user. *n Interspeech, Florence*, 17-20.

Hans, P. (2011). *A Primer on Scientific Programming with Python.* New York: Springer.

Phyton. (29 de 04 de 2016). *Idle Phyton*. Obtenido de Idle Phyton: https://docs.python.org/2/library/idle.html

Phyton. (29 de 04 de 2016). *Python GUI Programming (Tkinter)*. Obtenido de Python GUI Programming (Tkinter): http://www.tutorialspoint.com/python/python_gui_programming.htm

Poor, H. (1985). An Introduction to Signal Detection and Estimation. New York: Springer- Verlag.

Thomas, T., Pecham, J., & Frangoulis, E. (1989). A Determination of the Sensitivity of Speech Recognisers to Speaker Variability. *Proceedings of ICASSP*, (págs. 544-547). Glasgow.

Thomas, T., Pecham, J., Frangoulis, E., & Cove, J. (14989). A The Sensitivity of Speech Recognisers to Speaker Variability and Speaker Variation. *Proc of Eurospeech*, (págs. 408-411). Paris.