RECONOCEDOR DE VOCALES.

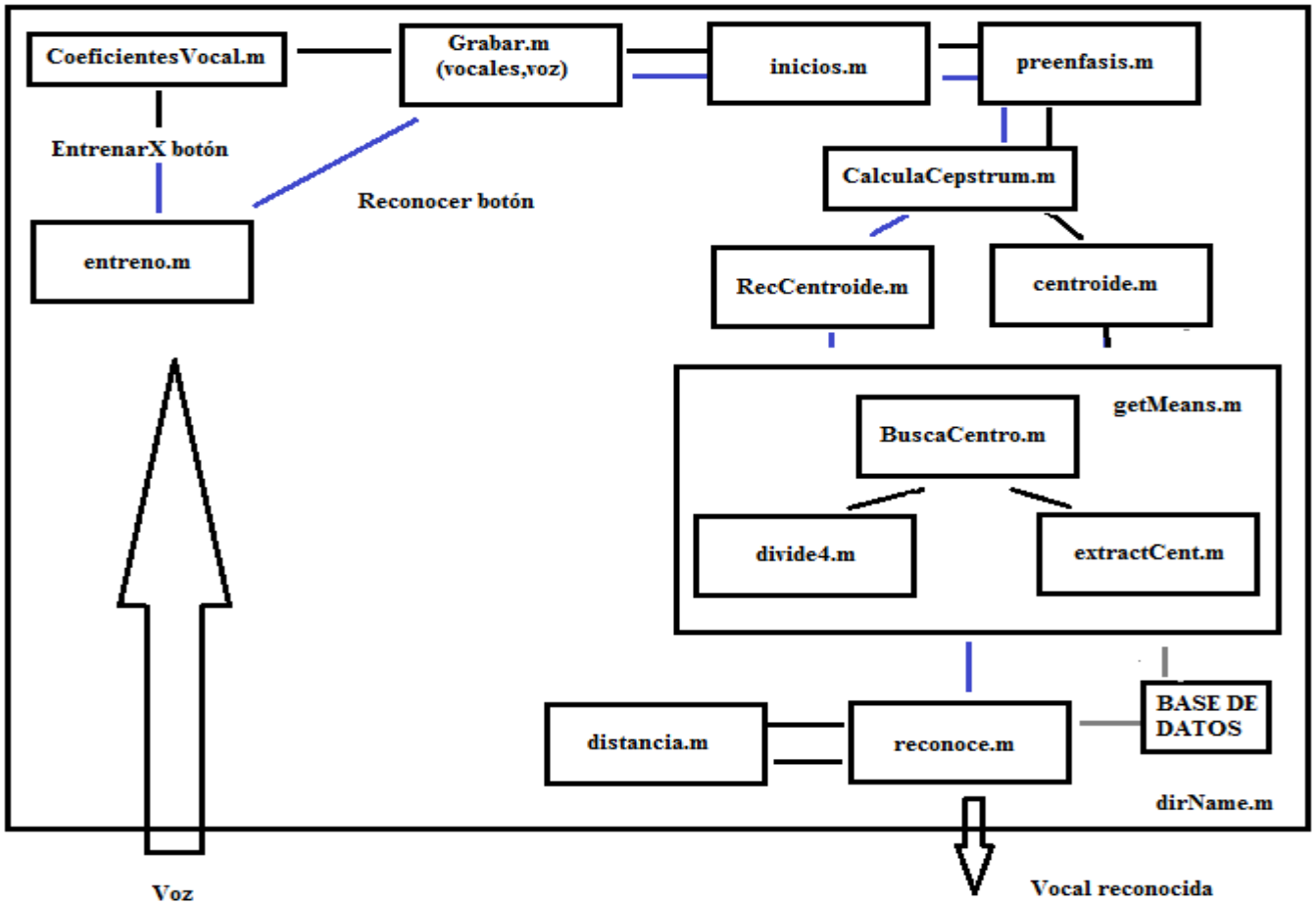
TDSÑ. Grupo 32

Realizado por: Rodrigo Escudero González

Esquema de la solución propuesta

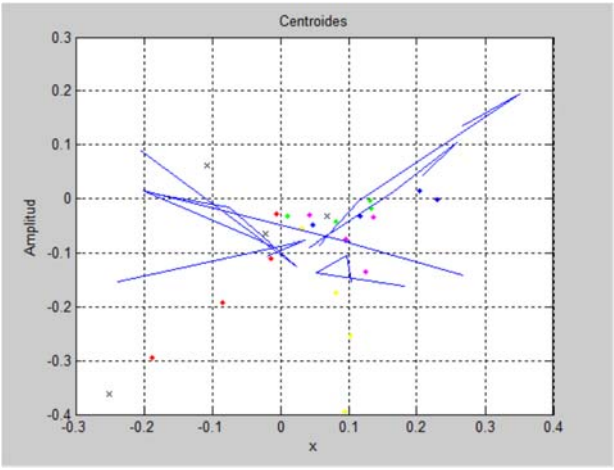
Cuando se pulsa en la interfaz gráfica el botón 'EntrenarX', siendo X el conjunto de vocales, lo que hacemos es llamar a la función 'Grabar.m' que tomará 8000 muestras de la señal, luego pasará a inicios.m donde preenfatizará la señal y se realizará el estudio cepstral. Posteriomente se hallará el centroide y se almacenará en la base de datos junto a los components cepstrales.

Si pulsamos, en cambio, el botón 'Reconocer', se seguirán los mismos pasos que en 'EntrenarX' sólo que al final se llamará a la función 'reconoce.m' mirando en la base de datos y con la ayuda de la función 'distancia.m' nos dará la vocal reconocida.



Pruebas

Esta imagen representa los centroides de las cinco vocales y el de la vocal a reconocer (las cruces grises) en el momento de la decisión.
Se ha grabado como vocal a reconocer la 'A'.



Como podemos observar en la imagen los centroides más cercanos o que tienen menor distancia a las cruces grises (vocal a reconocer) son los puntos rojos que representan los centroides de la vocal 'A'.

Por lo que el proceso de reconocimiento será un éxito.

Funciones empeladas para el reconocedor de vocales.

dirName.m

```
function [ruta] = dirName

% Función que permite establcer la ruta de trabajo. A partir de ahí se debe
% crear la siguinetes estructura de carpetas. ruta\->
% {cps{a,e,i,o,u,Voz},wavs{a,e,i,o,u,Voz}, funciones {...}}
ruta = 'C:\Users\Rodrigo\Documents\MATLAB\p1';

end
```

coeficientesVocal.m

```
function coeficientesVocal(vocal)
% Función que calcula los centroides del parámetro pasado.
%
dt = Grabar (vocal); % Se graba y almacena la señal.
inicios (vocal,dt);  % Se extraen y almacenan los cepstrum.
centroide (vocal);   % Se extraen y almacenan los centroides.

end
```

Grabar.m

```
function dt = Grabar(vocal,name)

% Función que graba y almacena la señal. Posee dos parámetros de entrada:
% Si llamamos a la función con 1 parámetro (vocal), la salida se almacenará en la vocal que
hayamos introducido como parámetro nombrándose por la fecha de grabación. Se utilizará para
realizar el entrenamiento de las vocales. Cuando llamamos a la función con dos parámetros,se va
a utilizar para el módulo de reconocer y sólo necesitaremos el audio una sóla vez por lo que
podremos sobreescribir el fichero.

Fs=8000;            % Frecuenca de muestreo a utilizar;doble del ancho de banda de la voz.
tiempoGrabacion=1;   % Tiempo de grabación en segundos.
y = wavrecord(tiempoGrabacion*Fs,Fs,1);% Grabación en modo mono durante 1(s). Muestreadas a 8KHz
(8000 muestras).
switch nargin
    case 1
        dt = datestr(now,'yyyymmddHHMMSS'); % LLamar un fichero por la fecha de creación.
    case 2
        dt=name;
end

which=strcat(dirName,'\wavs\',vocal,'\',dt,'.wav');
wavwrite(y,Fs,16,which); % Almacena la señal en donde le hayamos ordenado en 'which'. (Los wavs).
sound(y,Fs);

end
```

inicios.m

```
function inicios(vocal,fileName)

% Función que permite almacenar los coeficioentes Cepstrum de la vocal
% elegida.
%
[signalInicial,fs] = wavread(strcat (dirName,'\wavs\',vocal,'\',fileName,'.wav')); % Lectura en
'wavs'.
signal=preenfasis(signalInicial);       % Preénfasis de la señal.
N=length(signal);                       % Longitud de la señal.
duracionTrama=256;                      % Duración de 32 (ms).
numeroTramas = floor(fs/duracionTrama); % Redondeo del número de tramas.
SUP_signal=signal;                      % Se almacena la señal entera.
INF_signal=signal(128:N);               % Se almacena la señal desde la muestra 128 hasta el
final.Solape 16 ms.
```

```
CoefCesptrum_SUP = zeros(numeroTramas,10);  % Matriz para almacenar los coeficientes Cepstrum
de cada trama de la señal.
CoefCesptrum_INF = zeros(numeroTramas-1,10);% Igual que CoefCesptrum_SUP; se le resta 1 porque
se empieza a almacenar en la muestra 128. Excedería los límites.

for i=1:numeroTramas
    CoefCesptrum_SUP (i,:) = CalculaCepstrum (SUP_signal((i-1)*256+1:i*256)); % Se llama a
CalculaCepstrum y se le pasan tramas 256 muestras de la señal.
    if i < numeroTramas
        CoefCesptrum_INF (i,:) = CalculaCepstrum (INF_signal((i-1)*256+1:i*256));
    end;
end
save (strcat
(dirName,'\cps\',vocal,'\',fileName,'.cps'),'CoefCesptrum_SUP','CoefCesptrum_INF'); %
Almacenamiento de todos los coeficientes Cepstrum de la señal.(Los cps).

End
```

preenfasis.m

```
function [y]=preenfasis(x)

% Función que preenfatiza la señal.
% Es un filtro IIR paso alto que amplifica las componentes de alta frecuencia
% de la señal con el fin de mantener una respuesta constante a través para
% todo el rango de frecuencias.
%
b=[1 -0.95];
y=filter(b,1,x); % Filter utiliza la transpuesta de la segunda forma directa.

end
```

CalculaCepstrum.m

```
function [ CoefCesptrum ] = CalculaCepstrum( frame )

% Función que calcula los 10 primeros coeficientes Cepstrum de la trama que
% se le pase como parámetro.
%
TEMP=ifft(log(abs(fft(frame.*hamming(256))))); % Enventanado de Hamming de 256 muestras (32ms)
de cada trama.
% CoefCesptrum = TEMP (1:10);
CoefCesptrum = TEMP (2:11); % Se desprecia el primer coeficiente por no ser significativo.

end
```

centroide.m

```
function centroide (vocalName)

%Cálculo de los centroides de un directorio vocal.
numeroTramas = 31;                                  % Resultado del floor de 'inicios'.
CoefCesptrum_SUP = zeros(numeroTramas,10);          % Coeficientes Cepstrum rama superior.
CoefCesptrum_INF = zeros(numeroTramas-1,10);        % Coeficientes Cepstrum rama inferior.
dirlist = dir(strcat (dirName,'\cps\',vocalName,'\*.cps'));    % Listado del directorio de cada
vocal.
m62=zeros(61,10);        % Matriz donde se guardarán los Cepstrum tanto de la rama superior
como de la inferior.
m62result= zeros(61,10); % Matriz donde se guardará la media de los Cepstrum de cada cocal que
se haya grabado.
for i = 1:length(dirlist)
    filename=dirlist(i).name;                % Extracción del nombre del fichero.
    load (strcat (dirName,'\cps\',vocalName,'\',filename),'-mat','CoefCesptrum_SUP','CoefCesptrum_INF'); %
Se cargan los Cepstrum.

    for j = 1:31;
        m62(j*2-1,:)=CoefCesptrum_SUP(j,:); % Se cargan en m62, en las filas impares, los
        if j<31;                             % CoefCepstrum_SUP, y en las pares los
            m62(j*2,:)=CoefCesptrum_INF(j,:);% CoefCepstrum_INF. Se ordenan en el tiempo.
        end;
    end;
```

```
    m62result = m62result+m62;  % Suma de todas las matrices de cada elemento de dirList.
end
m62result=m62result/i;          % Media del bucle anteior.
%plot (m62result,'.');
ctrs=zeros(4,10);               % Matriz donde se guardarán los centroides.
[Rcentroide, ctrs]=kmeans(m62result,4);  % Extracción de los 4 centroides de todos los
coeficientes Cepstrum. Rcentroide: qué puntos pertenecen a qué K.

save (strcat (dirName,'\cps\',vocalName,'\centroide.cnt'),'Rcentroide','ctrs','m62result'); %
Almacenamiento datos de interés.

end
```

RecCentroide.m

```
function RecCentroide (vocal,fileName)

% Función idéntica a centroide con la salvedad que aquí no hace falta el
% bucle for del listado, no se va a crear una base de datos, ya está hecha,
% sólo se va aemplear este audio para compararlo con la base de datos ya
% creada.
CoefCesptrum_SUP = zeros(numeroTramas,10);
CoefCesptrum_INF = zeros(numeroTramas-1,10);
Vm62=zeros(61,10);
fileNameCps = strcat(dirName,'\cps\',vocal,'\',fileName,'.cps');
load (fileNameCps,'-mat','CoefCesptrum_SUP','CoefCesptrum_INF');
for j = 1:31;
    Vm62(j*2-1,:)=CoefCesptrum_SUP(j,:);
    if j<31;
        Vm62(j*2,:)=CoefCesptrum_INF(j,:);
    end;
end
plot (Vm62,'.');

[Vcentroide,Vctrs]=kmeans(Vm62,4);

save (strcat (dirName,'\cps\',vocal,'\',fileName,'.cnt'),'Vcentroide','Vctrs','Vm62');

end
```

Reconoce.m

```
function [sol,min]= Reconoce()

%Función que devuelve la vocal reconocida.
%
inicios('Voz','voz');               % Características del
RecCentroide('Voz','voz');          % Audio a comparar.
vocales='aeiou';                    % Array de posibles soluciones
min= intmax('int64');               % Mayor valor integer (64 bit)
sol='a';

for rv=1:length(vocales)            % Bucle que devuelve la vocal más cercana
    dist=distancia(vocales(rv));    % a la señal grabada.
    if dist< min
        min=dist;
        sol=vocales(rv);
    end

end
```

Distancia.m

```
function [retorno] = distancia(vocal)
% Función que calcula la distancia euclídea entre dos matrices.
%
fileName = strcat (dirName,'\cps\Voz\voz.cnt');                % Destinada a almacenar
el centroide de la señal a comparar (Vctrs).
```

```
vocalName= strcat (dirName,'\cps\',vocal,'\centroide.cnt');    % Destinada a almacenar
el centroide de la base de datos creada (ctrs).
load (fileName,'-mat','Vcentroide','Vctrs','Vm62');            % Carga de los datos
necesarios.
load (vocalName,'-mat','Rcentroide','ctrs','m62result');       % Carga de los datos
necesarios.
diferenciaSimple = Vctrs - ctrs;                               % Diferencia de los
centroides.
retorno = sqrt(sum(diag(diferenciaSimple * diferenciaSimple')));   % Distancia de estas
matrices. Se define como la raíz cuadrada de la traza
del producto de las matrices de los centroides. (Una conjugada)
end
```

getMeans.m

```
function centroides = getMeans(m62result)
 % Extrae la matriz de 4x10 centroides de m62result

centroides=zeros(4,10);
for i=1:10
    %i=1
    [x y]=BuscaCentro(m62result(:,i));
    [cuadrantes,indices]=divide4(m62result(:,i),x,y);
    for j=1:4
        % j=1;
        [cx,cy]=extractCent( m62result(:,i), cuadrantes,j);
        centroides(j,i)=cy;
    end
end
end
```

divide4.m

```
function [ cuadrantes ordinal]  = divide4( nube,x,y )

% Divide la nube de puntos en cuatro regions, los cuadrantes.

cuadrantes=zeros(1,length(nube));
ordinal=zeros(4,1);
for i = 1:length(nube)
    cmpx = i - x;
    cmpy = nube(i) - y ;

    if  (cmpx<0) && (cmpy<0)
        ind=3;
    else
        if  cmpx<0 && cmpy>0
            ind=2;
        else
            if  cmpx>0 && cmpy>0
                ind=1;
            else
                ind=4;
            end
        end
    end
    cuadrantes(1,i)=ind;
    ordinal(ind) = ordinal(ind) + 1;
end

end
```

extractCent.m

```
function [ cx cy ] = extractCent( vector,indices,cuadrante )

% Extrae la los puntos donde se hallan los centroides

nube(1:length(vector),1) = realmax;%nube trabajo

for i=1:length(nube)

    if indices(i)== cuadrante
        nube(i)=vector(i);
    end
end

[cx cy]=BuscaCentro(nube);

end
```

BuscaCentro.m

```
function [centroX centroY] = BuscaCentro (nube)

% Extrae el centro de una nube de puntos aplicando dicotomía.

xMax=length(nube);
xMin=1;

while true
    centroX = floor((xMax + xMin)/2);
    %disp('ITERAMOS X');

    resultX=zeros(3,1);

    for i = 1:length(nube)
        if nube(i)<realmax
            if i > centroX
                ind=2;  %derecha
            else
                if i < centroX
                    ind=1; %izquierda
                else
                    ind=3; % Coincidentes
                end
            end
            resultX(ind) =  resultX(ind) +1;
        end
    end
    %if abs(resultX(1)-resultX(2)) <= (resultX(3)+1)
    if abs(resultX(1)-resultX(2)) <= 1

        %disp ('Nos vamos');
        break;
    end

    if resultX(1) < resultX(2)
        xMin=centroX;
    else
        xMax=centroX;
    end
```

```
end

%iteramos
elMx=-realmax;
for i =1:length(nube)
    if nube(i)<realmax
        if nube(i) > elMx
            elMx=nube(i);
        end
    end
end
elMn=min(nube);
while true
    centroY = (elMx + elMn)/2;
    %disp('ITERAMOS Y');
    %centro=[centroX, centroY];

    resultY=zeros(3,1);

    for i =1:length(nube)
        if nube(i)<realmax
            if nube(i) > centroY
                ind=1;
            else
                if nube(i) < centroY
                    ind=2;
                else
                    ind=3;
                end
            end
            resultY(ind) =  resultY(ind) +1;
        end
    end
    if abs(resultY(1)-resultY(2)) <= 1
        %  if abs(resultY(1)-resultY(2)) <= (resultY(3)+1)
        %disp ('Nos vamos');
        break;
    end

    if resultY(1) < resultY(2)
        elMx=centroY;
    else
        elMn=centroY;
    end
end
end
```

Interfaz de usuario (entreno.m)

```
function varargout = entreno(varargin)

      % ENTRENO MATLAB code for entreno.fig
      %      ENTRENO, by itself, creates a new ENTRENO or raises the existing
      %      singleton*.
      %
      %      H = ENTRENO returns the handle to a new ENTRENO or the handle to
      %      the existing singleton*.
      %
      %      ENTRENO('CALLBACK',hObject,eventData,handles,...) calls the local
      %      function named CALLBACK in ENTRENO.M with the given input arguments.
      %
      %      ENTRENO('Property','Value',...) creates a new ENTRENO or raises the
      %      existing singleton*.  Starting from the left, property value pairs are
      %      applied to the GUI before entreno_OpeningFcn gets called.  An
```

```
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to entreno_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help entreno
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @entreno_OpeningFcn, ...
    'gui_OutputFcn',  @entreno_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

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


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

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

% Update handles structure
guidata(hObject, handles);


% UIWAIT makes entreno wait for user response (see UIRESUME)


% uiwait(handles.figure1);

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

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


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

```
% --- Executes on button press in pushbuttonI.
function pushbuttonI_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
coeficientesVocal ('i')                                    % Características vocal
i


% --- Executes on button press in pushbuttonU.
function pushbuttonU_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonU (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
coeficientesVocal ('u')                                    % Características vocal
u


% --- Executes on button press in pushbuttonE.
function pushbuttonE_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonE (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
coeficientesVocal ('e')                                    % Características vocal
e


% --- Executes on button press in pushbuttonO.
function pushbuttonO_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonO (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
coeficientesVocal ('o')                                    % Características vocal
o


% --- Executes on button press in pushbuttonReconocer.
function pushbuttonReconocer_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonReconocer (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.Resultado,'String','');
Grabar('Voz','voz');
mvocal=zeros(5,1);


for i=1:30
    [sol,min]=Reconoce();
    disp(strcat ('reconozco :',sol));
    switch sol
        case 'a'
            ind=1;
        case 'e'
            ind=2;
        case 'i'
            ind=3;
        case 'o'
            ind=4;
        case 'u'
            ind=5;
        otherwise
            ind=9;
    end
    mvocal(ind) = mvocal(ind) + 1;
end
solucion=-1;
```

```
for i=1:5
    disp (strcat ('indice ',num2str(i),' es ',num2str(mvocal(i))));
    if (solucion < mvocal(i))
        solN=i;
        solucion=mvocal(i);
    end
end
switch solN
    case 1
        sol='a';
    case 2
        sol='e';
    case 3
        sol='i';
    case 4
        sol='o';
    case 5
        sol='u';
    otherwise
        sol='m';

end
%solucion =sprintf('Sol = %s %f',sol,min);
solucion =sprintf('%s',sol);
set(handles.Resultado,'String', solucion);


% --- Executes on button press in pushbuttonValidar.
function pushbuttonValidar_Callback(hObject, eventdata, handles) %Función auxiliar
% hObject    handle to pushbuttonValidar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sol = get(handles.Resultado,'String');
%disp (['Encontramos ',sol]);
RenameVoz(sol);
```