

PAV
MEMORIA PRÁCTICA 2

Marc Dulcet Isart
Pablo Lozano Gómez

2021 QT

1. Introducción

Memoria de la práctica 2 de PAV. Al final leímos que la entrega se realiza en el propio README. Sin embargo, elaborar este documento nos ayudó a encontrar alguna información más rápido respecto a buscar en el propio enunciado de 35 páginas.

2. Desarrollo de la práctica

2.1. Detección de voz usando umbrales

Tareas:

Para analizar la capacidad de un detector de voz basado en medidas estadísticas de la señal, como las calculadas en la primera práctica, utilice una señal de voz (16kHz, mono) que contenga pausas internas (puede utilizar la señal grabada en la primera práctica o grabar una nueva al efecto).

Nombre al fichero `pav_ggLD.wav` (gg: grupo 11, 41, etc; L: id. del pc; D: índice de señal, si graba más de una: 1, 2, ...). Los ficheros grabados por todos los alumnos del curso se utilizarán para evaluar los distintos sistemas de detección de voz.

1. Tal y como hizo en la primera práctica, visualice en el programa `wavesurfer` la señal y las características potencia y tasa de cruces por cero. Cree un nuevo panel *transcription* y utilícelo para etiquetar los segmentos de silencio (S) y de voz (V). Para ello, sitúese al final de cada segmento de voz o silencio e introduzca la etiqueta que corresponda. Sólo considere como silencio los segmentos de una cierta duración; no pausas cortas que no se perciban claramente.

Guarde las transcripciones (fichero `.lab`) y suba los ficheros `.wav` y `.lab` a Atenea, para contribuir a la base de datos de evaluación.

2. Defina unas condiciones *razonables* para detectar la presencia de voz o silencio. Para ello, observe la señal y determine:

- Nivel de potencia y tasa de cruces por cero en el silencio inicial de la señal.
- Incremento aproximado del nivel de potencia, respecto al valor en el silencio inicial, que se tiene para distintos tipos de fonema: fricativas sordas, consonantes sonoras, vocales, etc.
 - ¿Es capaz de determinar un valor mínimo del incremento de nivel tal que, si se supera, podamos tener una cierta seguridad de que se trata de voz, pero, si no se hace, podamos considerar que se trata de silencio?
- Duración mínima razonable de los segmentos de voz y de silencio.

3. Visualice con `less` o `cat` el fichero con la transcripción, e interprete el significado de su contenido.

Usaremos la grabación de la práctica 1 porque consideramos que puede resultar una grabación útil para este ejercicio, puesto que dispone de bastantes tramos de silencio y sonido.

Como explica el enunciado, usamos el programa WaveSurfer para etiquetar los segmentos de silencio (s) y de voz (v). Sólo consideramos como silencio aquellos segmentos que tienen una cierta duración, tal y como se muestra a continuación:

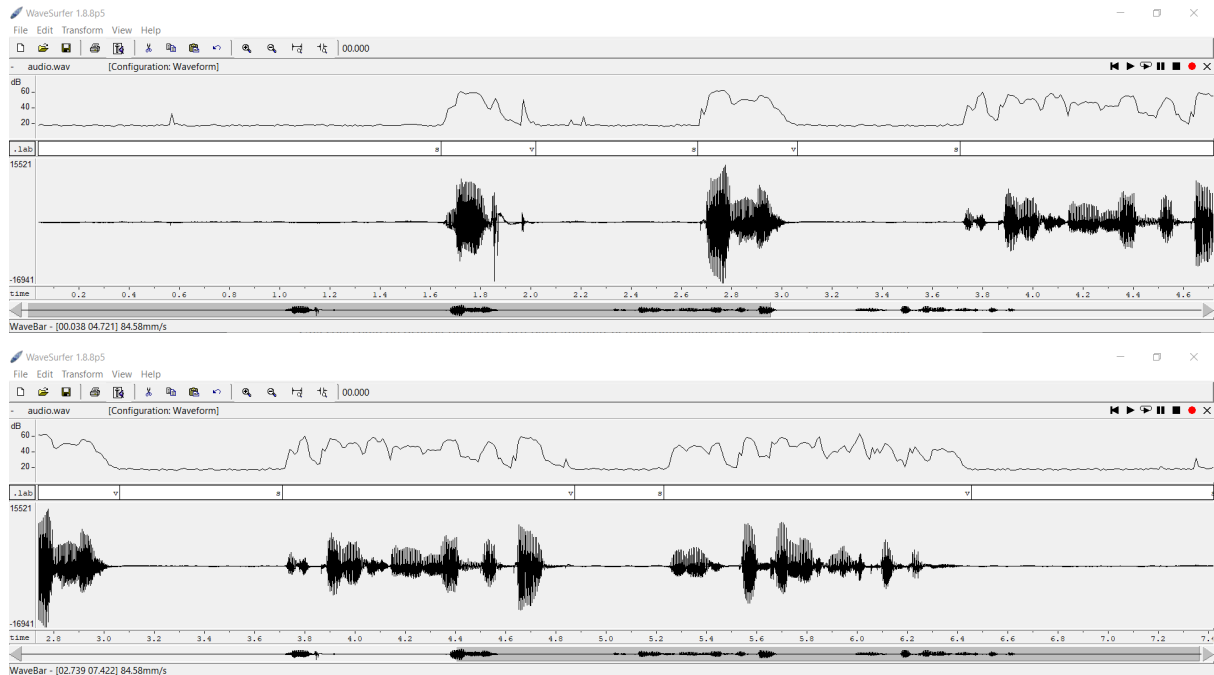


Figura 1: captura de WaveSurfer con los segmentos de voz (v) y silencio (s)

Observando el gráfico de potencia, a priori parece que podríamos establecer un umbral para detectar si estamos ante sonido o silencio. Por ejemplo, los segmentos iguales o mayores a 40 o 50 dB podrían considerarse sonido, mientras que el resto sería silencio. Este criterio podría funcionar con nuestra grabación, pero no tiene por qué funcionar con el resto de grabaciones de la base de datos, ya que algunas podrían tener segmentos de sonido con menos potencia, mayor ruido de fondo, etc.

Por otra parte, si nos fijamos de nuevo en el gráfico y, tenemos en cuenta la frase de voz que representa la señal: “Marc, Pablo, estamos en el lab de PAV y nos pican los mosquitos”. Podemos llegar a comprobar que, en cada segmento de sonido, se pueden distinguir los sonidos sonoros de los sordos, puesto que los sonidos sonoros tienen menos cruces por cero (zcr) y mayor energía; por contra, los sonidos sordos tienen más cruces por cero y muy poca energía.

Para finalizar esta sección, con el comando ‘less’ o ‘cat’ podemos observar cuánto dura en segundos cada segmento etiquetado en WaveSurfer:

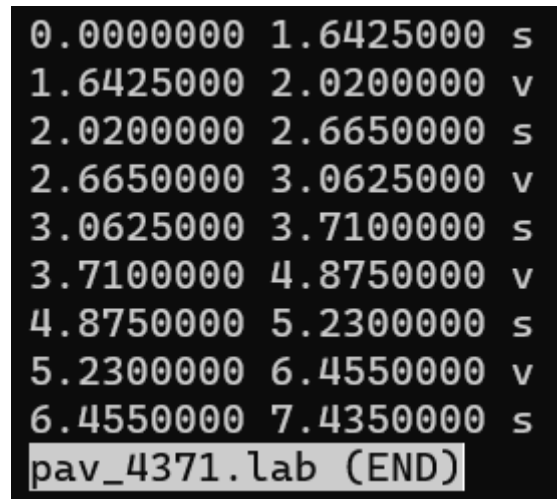


Figura 2: duración de los segmentos de voz (v) y silencio (s)

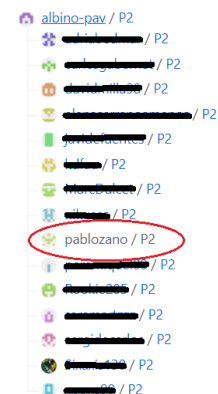
En cada fila tenemos información sobre cada uno de los estados identificados: instante en segundos del inicio del estado, instante del final y estado identificado.

2.2. Estructura del proyecto

Siga las instrucciones del fichero README.md en el repositorio GitHub de la [Práctica P2](#) para clonar los ficheros de la práctica en su directorio raíz de las prácticas ~/PAV.

Esta práctica se distribuye a través del repositorio GitHub [P2] (<https://github.com/albino-pav/P2>), y una parte de su gestión se realizará mediante esta web de trabajo colaborativo.

Para ello, primero debemos crear una nueva cuenta en GitHub, que nos servirá para gestionar la práctica actual así como el resto de prácticas del curso. Una vez tenemos nuestra cuenta, debemos acceder al repositorio de GitHub [P2] y, mediante el botón de “Fork”, se creará en nuestra cuenta un proyecto con el mismo nombre (P2). Esto sólo lo hará un miembro del equipo de laboratorio, que añadirá al otro miembro como “Collaborator” y así también podrá gestionar el repositorio. A continuación, localizamos el botón “Branch: master”, en la página principal del repositorio, y lo usaremos para crear una rama nueva con nuestros primeros apellidos.



Ahora abrimos una sesión de Bash y nos situamos en el directorio “/PAV”. Desde aquí ejecutamos:

```
$ git clone https://github.com/albino-pav/P2.git
```

A continuación nos movemos al directorio de la práctica “/PAV/P2” y añadimos la etiqueta de “origin” para facilitar los “push” y “pull” al repositorio original:

```
$ git remote add origin https://github.com/albino-pav/P2.git
```

Cambiamos a la rama “Lozano-Dulcet” con la orden:

```
$ git checkout Lozano-Dulcet
```

A partir de ahora, cada vez que queramos subir cambios locales al repositorio de GitHub, debemos confirmar los cambios en nuestro directorio local de la siguiente forma:

```
$ git add .  
$ git commit -m “Mensaje del commit”  
$ git push -u origin Lozano-Dulcet
```

Por tanto, de ahora en adelante trabajaremos en nuestro directorio local “/PAV/P2”, que ha sido clonado del directorio principal de GitHub [P2]:

```
lozano@LAPTOP-SU8TMLJK:~/PAV/P2$ ls  
MARKDOWN.md  bin      docopt_c  meson.build  pav_4371.lab  pav_4371.wav  src  
README.md    db.v4    img       p2_vad.pdf   pav_4371.vad  scripts
```

El código fuente fundamental está ubicado en el directorio PAV/P2/src, y está formado por los ficheros main_vad.c, vad.h y vad.c. La gráfica siguiente muestra la estructura del código fuente:

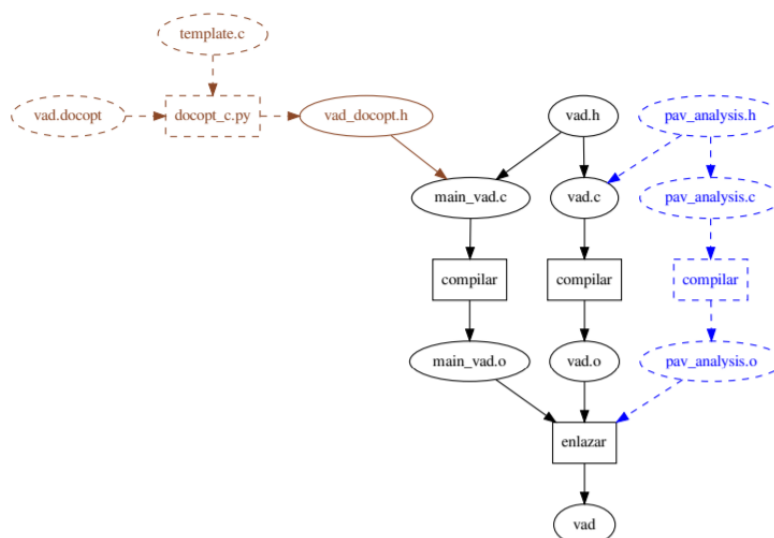


Figura 3: estructura de código fuente

Para usar las funciones definidas en la primera práctica para el análisis de la señal, será necesario incorporar al proyecto los ficheros `pav_analysis.c` y `pav_analysis.h` de la primera práctica.

```
lozano@LAPTOP-SU8TMLJK:~/PAV/P2/src$ ls
main_vad.c  pav_analysis.c  pav_analysis.h  vad.c  vad.dcopt  vad.h  vad.dcopt.h
```

Cabe decir que también debemos incluir el fichero “`pav_analysis.c`” en el fichero de dependencias “`meson.build`” para la correcta ejecución de nuestro código.

En el siguiente apartado profundizaremos más en el mantenimiento del proyecto mediante Meson/Ninja.

2.3. Mantenimiento del proyecto usando Meson/Ninja.

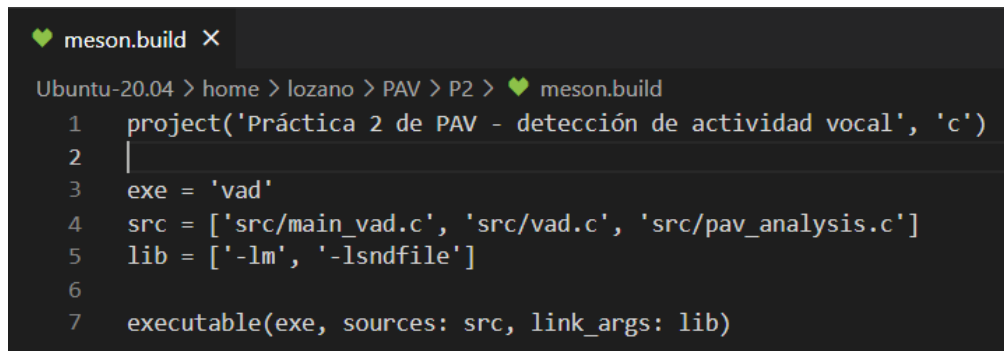
Tareas:

- Escriba el fichero `meson.build`.
- Ejecute `meson` y `ninja` para generar el programa en el directorio `bin`.
 - Compruebe que `ninja` realiza correctamente el mantenimiento del proyecto, modificando alguno de los ficheros y comprobando que `ninja` sólo realiza las tareas necesarias para mantener el programa actualizado.

El programa “meson” se basa en ficheros de dependencias, semejantes a los “makefile” de “make”, denominados “`meson.build`”. Como en el caso de makefile, `meson.build` debe existir en el directorio raíz del código que deseamos mantener. Algunas de las características principales de “meson”, que debemos tener en cuenta para esta práctica, son las siguientes:

- El fichero “`meson.build`” está escrito en un lenguaje propio semejante a Python.
- Cualquier fichero “`meson.build`” debe iniciar con una línea con la descripción básica del proyecto: indicando el nombre del proyecto, en nuestro caso ‘Práctica 2 de PAV - detección de actividad vocal’, y el lenguaje de programación usado, en nuestro caso ‘c’.
- La compilación y enlazado se realizan en un directorio distinto al que se encuentra el fichero “`meson.build`”. Así podemos realizar distintas versiones del proyecto o eliminar completamente una compilación con sólo eliminar el directorio que la contiene, todo ello sin preocuparnos de modificar los directorios con el código original.
- No es necesario especificar los ficheros incluidos por otros, como por ejemplo las cabeceras “.h” de los ficheros “.c”. Por ello, nosotros sólo hemos incluido

'pav_analysis.c' dentro del directorio fundamental para la elaboración de esta práctica (PAV/P2/src).



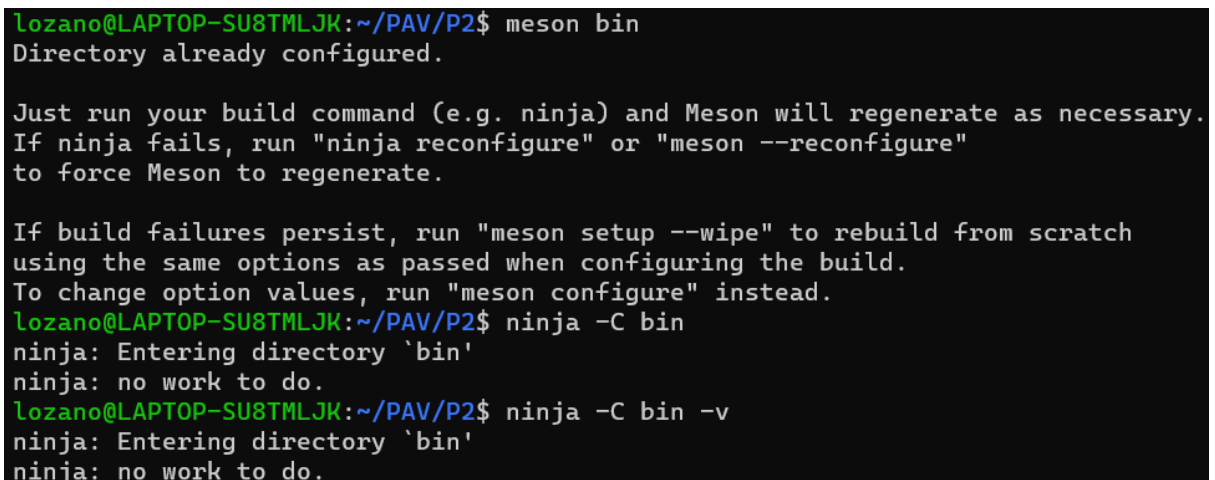
```
meson.build X
Ubuntu-20.04 > home > lozano > PAV > P2 > meson.build
1 project('Práctica 2 de PAV - detección de actividad vocal', 'c')
2 |
3 exe = 'vad'
4 src = ['src/main_vad.c', 'src/vad.c', 'src/pav_analysis.c']
5 lib = ['-lm', '-lsndfile']
6
7 executable(exe, sources: src, link_args: lib)
```

Figura 4: contenido de meson.build

Para facilitar la legibilidad y mantenimiento, definimos una serie de variables con los elementos que participan en la creación del programa:

- **exe**: nombre del programa a generar, que es el objeto del proyecto.
- **src**: los códigos fuentes que forman el programa.
- **lib**: las librerías con las que queremos enlazarlo. Estas son:
 - '-lm': es la misma librería matemática que usamos para la primera práctica.
 - '-lsndfile': la gestión de la lectura y escritura de los ficheros de audio en formato WAVE se realiza usando funciones de la librería "sndfile".

Una vez hemos escrito el fichero "meson.build", ahora debemos ejecutar "meson" y "ninja" para generar el programa en el directorio "bin".



```
lozano@LAPTOP-SU8TMLJK:~/PAV/P2$ meson bin
Directory already configured.

Just run your build command (e.g. ninja) and Meson will regenerate as necessary.
If ninja fails, run "ninja reconfigure" or "meson --reconfigure"
to force Meson to regenerate.

If build failures persist, run "meson setup --wipe" to rebuild from scratch
using the same options as passed when configuring the build.
To change option values, run "meson configure" instead.
lozano@LAPTOP-SU8TMLJK:~/PAV/P2$ ninja -C bin
ninja: Entering directory `bin'
ninja: no work to do.
lozano@LAPTOP-SU8TMLJK:~/PAV/P2$ ninja -C bin -v
ninja: Entering directory `bin'
ninja: no work to do.
```

Para comprobar que “ninja” realiza correctamente el mantenimiento del proyecto, podemos, por ejemplo, hacer una modificación tonta en el fichero “main_vad.c” y ver qué operaciones realiza “ninja” en cada paso mediante el siguiente comando:

```
lozano@LAPTOP-SU8TMLJK:~/PAV/P2$ ninja -C bin -v
ninja: Entering directory `bin'
[1/2] cc -Ivad@exe -I. -I.. -fdiagnostics-color=always -pipe -D_FILE_OFFSET_BITS=64 -Wall -Winvalid-pch -g -MD -MQ 'vad@exe/src_pav_analysis.c.o' -MF 'vad@exe/src_pav_analysis.c.o.d' -o 'vad@exe/src_pav_analysis.c.o' -c ../src/pav_analysis.c
../src/pav_analysis.c: In function 'compute_power':
../src/pav_analysis.c:5:9: warning: unused variable 'i' [-Wunused-variable]
     5 |     int i = 0;
       |         ^
../src/pav_analysis.c: At top level:
../src/pav_analysis.c:20:1: warning: data definition has no type or storage class
     20 |     pepito = 69;
       |     ^~~~~~
../src/pav_analysis.c:20:1: warning: type defaults to 'int' in declaration of 'pepito' [-Wimplicit-int]
[2/2] cc -o vad 'vad@exe/src_main_vad.c.o' 'vad@exe/src_vad.c.o' 'vad@exe/src_pav_analysis.c.o' -Wl,--as-needed -Wl,--no-undefined -Wl,--start-group -lm -lsndfile -Wl,--end-group
```

2.4. Utilización del VAD

- Compruebe que bin/vad funciona conforme a lo esperado.

```
lozano@LAPTOP-SU8TMLJK:~/PAV/P2$ bin/vad --help
VAD - Voice Activity Detector

Usage:
vad [options] -i <input-wav> -o <output-vad> [-w <output-wav>]
vad (-h | --help)
vad --version

Options:
-i FILE, --input-wav=FILE    WAVE file for voice activity detection
-o FILE, --output-vad=FILE   Label file with the result of VAD
-w FILE, --output-wav=FILE   WAVE file with silences cleared
-l FLOAT, --alpha1=FLOAT    Ganancia para obtener el umbral de deteccion 1 [default: 9]
-v, --verbose               Show debug information
-h, --help                  Show this screen
--version                   Show the version of the project
```

Tareas:

- Ejecute vad con la señal que grabó y etiquetó al principio de la práctica.
- Abra la señal con wavesurfer y cree dos paneles de transcripción: uno con la segmentación manual (fichero hola.lab) y otro con la generada automáticamente (hola.vad).
- Compare el resultado obtenido con el resultado teórico.
 - Como el programa, en su estado actual, asigna un etiquetado aleatorio, no es de extrañar que no acierte ni una...

```
lozano@LAPTOP-SU8TMLJK:~/PAV/P2$ bin/vad -i pav_4371.wav -o pav_4371.vad|
```

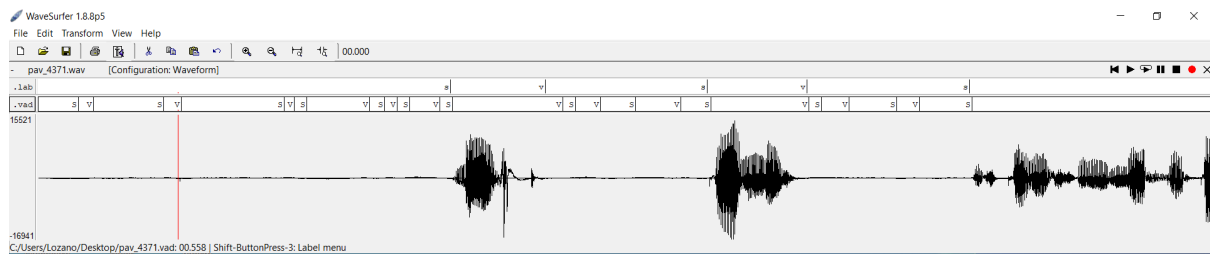



Figura 5: comparación .lab con .vad cuando tenemos 89%

Cuando tenemos nuestras señales “.lab” y “.vad” ya podemos realizar el alineado entre el resultado y la referencia y calcular la media geométrica de $F_1(V)$ y $F_{1/2}(S)$ para evaluar nuestro sistema:

$$F_{mix} = \sqrt{F_2(V) \cdot F_{1/2}(S)}$$

Para ello, disponemos del script Perl “scripts/vad_evaluation.pl” que, cuando ejecutamos con el nombre de nuestro fichero “.lab”, obtenemos un resultado cercano al 50%, el cual tiene sentido debido a la aleatoriedad de la implantación actual del VAD.

Sin embargo, para la correcta evaluación de nuestro sistema necesitaremos disponer de una base de datos, ya que si sólo trabajamos con nuestra señal actual, podríamos ajustar los parámetros del FSA para obtener unos resultados óptimos para esa señal, pero para otra señal podría ser totalmente diferente según sus condiciones de grabación, por ejemplo. Por tanto, usaremos una base de datos dada de 81 señales (db.v4).

Tarea:

Edite el fichero `scripts/run_vad.sh` para que el script llame al programa `vad`, y ejecute `scripts/run_vad.sh` y `scripts/vad_evaluation.pl` con los ficheros de la base de datos `db.v4` para comprender su funcionamiento.

```
$ run_vad.sh M X
home > lozano > PAV > P2 > scripts > $ run_vad.sh
1  #!/bin/bash
2
3  # Be sure that this file has execution permissions:
4  # Use the nautilus explorer or chmod +x run_vad.sh
5
6  # Write here the name and path of your program and database
7  DIR_P2=$HOME/PAV/P2
8  DB=$DIR_P2/db.v4 # Base de Datos (Data Base)
9  CMD="$DIR_P2/bin/vad --alpha1=$1"
10
11 for filewav in $DB/*/*wav; do
12     # echo
13     echo "***** $filewav *****"
14     if [[ ! -f $filewav ]]; then
15         echo "Wav file not found: $filewav" >&2
16         exit 1
17     fi
18
19     filevad=${filewav/.wav/.vad}
20
21     $CMD -i $filewav -o $filevad || exit 1
22
23     # Alternatively, uncomment to create output wave files
24     # filewavOut=${filewav/.wav/.vad.wav}
25     # $CMD $filewav $filevad $filewavOut || exit 1
26
27 done
28
29 # vad_evaluation.pl con los ficheros de la bade de datos db.v4
30 scripts/vad_evaluation.pl $DB/*/*lab
31
32 exit 0
33
```

2.5. Ejercicios

Ejercicios básicos:

1. Complete el código de los ficheros `main_vad.c` y `vad.c` para que el programa realice la detección de actividad vocal. Escriba las funciones de análisis o incorpore al proyecto los ficheros de la primera práctica `pav_analysis.c` y `pav_analysis.h`. Recuerde incorporar las cabeceras necesarias en los ficheros correspondientes.

Tiene completa libertad para implementar el algoritmo del modo que considere más oportuno, pero el código proporcionado puede ser un buen punto de partida para hacerlo usando un autómata de estados finitos (FSA). Encontrará en los ficheros sugerencias para ello, marcadas con la palabra **TODO** (del inglés *to do*, a realizar).

Ayúdese de la visualización de la señal y sus transcripciones usando `wavesurfer`, de la opción `-verbose` del programa `vad`, de la colocación de *chivatos* y de cualquier otra técnica que se le pueda ocurrir para conseguir que el programa funcione correctamente. Recuerde que el fichero de salida sólo debe incluir las etiquetas V y S.

2. Optimice los algoritmos y sus parámetros de manera que se maximice la puntuación de la detección de la base de datos de desarrollo (`db.v4`).

Ejercicios de ampliación:

1. Complete el código del fichero `main_vad.c` para que el programa, en el caso de que se indique un fichero `.wav` de salida (opción `--output-wav`), escriba en él la señal de la entrada, sustituyendo por cero los valores de los intervalos identificados como silencio.
2. Modifique los ficheros necesarios para permitir que los parámetros del FSA (nivel de los umbrales, α_0 , α_1 ...; duraciones mínimas y máximas; etc.) sean accesibles desde la línea de comandos (ver el Anexo II).

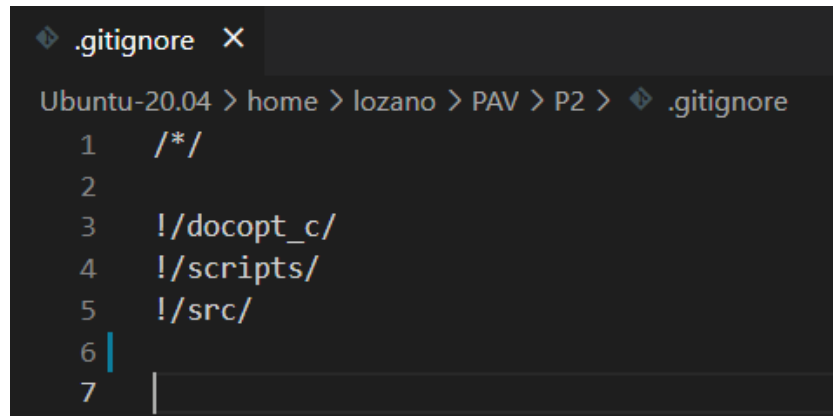
Esta ampliación, de hecho, le puede resultar muy útil para optimizar el resultado de la detección.

2.6. Fichero `~/.gitignore`

Para gestionar las versiones del software con Git, sólo nos interesan los ficheros y directorios que realmente forman el código. Por ello, crearemos un fichero de texto en el directorio del repositorio con el nombre `“.gitignore”`, donde queremos que se excluya de la gestión Git todos los directorios salvo los tres que nos interesan: `“src”`, `“scripts”` y `“docopt_c”`.

Tareas:

- Escriba el fichero `.gitignore` de manera que Git sólo gestione los directorios `P2/src`, `P2/docopt_c` y `P2/scripts`, y los ficheros del directorio `P2`.
- Realice la primera confirmación (*commit*) con los ficheros originales de la práctica.

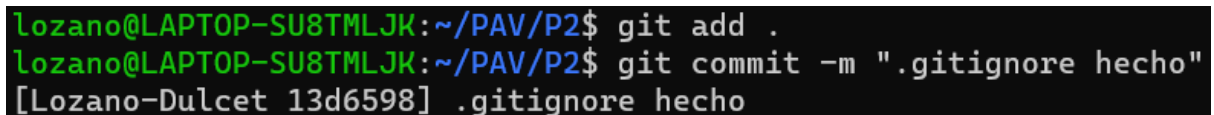


A screenshot of a code editor showing a file named `.gitignore`. The editor's title bar shows a diamond icon, the filename `.gitignore`, and a close button. The path bar shows `Ubuntu-20.04 > home > lozano > PAV > P2 > .gitignore`. The file content is as follows:

```
1  /*/
2
3  !/docopt_c/
4  !/scripts/
5  !/src/
6
7  |
```

La primera línea de “.gitignore” indica que no debe realizarse el seguimiento de ningún subdirectorio del proyecto. Por otro lado, las otras tres líneas que vienen a continuación indican que el directorio especificado por su nombre es una excepción a la regla anterior, por tanto, “docopt_c”, “scripts” y “src” no serán excluidos.

Ahora, cada vez que queramos almacenar una nueva versión, todo lo que tenemos que hacer es repetir los comandos “git add .” y “git commit” con un mensaje explicativo de los cambios incorporados en ella y Git sólo gestionará aquellos directorios que no han sido excluidos en “.gitignore”.



A screenshot of a terminal window showing the following commands and output:

```
lozano@LAPTOP-SU8TMLJK:~/PAV/P2$ git add .
lozano@LAPTOP-SU8TMLJK:~/PAV/P2$ git commit -m ".gitignore hecho"
[Lozano-Dulcet 13d6598] .gitignore hecho
```