Práctica 3: guía para el desarrollo del código fuente

Profesores de la asignatura

octubre de 2022

El código fuente a desarrollar en esta práctica, y también de las siguientes, sigue una estructura semejante a los típicos ejercicios de la ESO en las que el enunciado es una frase en la que el profesor ha dejado palabras en blanco para que el alumno escoja la adecuada.

Esta es una _____ con _____ para que el _____ escriba la _____ correcta

En este caso, sin embargo, en lugar de espacios en blanco tenemos cláusulas `\TODO`. El problema es que tenemos siete `\TODO`s repartidos en dos ficheros, y no es evidente en qué orden resolverlos, cómo hacerlo y, casi lo más importante, cómo comprobar que se ha hecho correctamente. Este documento pretende servir de guía para completar los huecos dejados en el código.

Puesta en marcha inicial.

Los programas de la práctica se compilan e instalan en el directorio `~/PAV/bin` con la orden `make release`:

```
usuario:~/PAV/P3/$ make release
meson --buildtype=release --prefix=/home/albino/PAV --libdir=lib build/release
The Meson build system
Version: 0.61.2
Source dir: /home/albino/PAV/P3/proba/P3
Build dir: /home/albino/PAV/P3/proba/P3/build/release
Build type: native build
Project name: Práctica 3 de PAV - detección de pitch
Project version: v2.0
C++ compiler for the host machine: c++ (gcc 11.2.0 "c++ (Ubuntu                    ...
↪  11.2.0-19ubuntu1) 11.2.0")
C++ linker for the host machine: c++ ld.bfd 2.38
Host machine cpu family: x86_64
Host machine cpu: x86_64
Program doxygen found: YES (/usr/bin/doxygen)
Configuring Doxyfile using configuration
Build targets in project: 5

Práctica 3 de PAV - detección de pitch v2.0

  User defined options
    buildtype: release
    libdir   : lib
    prefix   : /home/albino/PAV
```

```
Found ninja-1.10.1 at /usr/bin/ninja
ninja install -C build/release
ninja: Entering directory `build/release'
[12/13] Installing files.
Installing src/pav/libpav.a to /home/albino/PAV/lib
Installing src/get_pitch/get_pitch to /home/albino/PAV/bin
Installing src/get_pitch/pitch_evaluate to /home/albino/PAV/bin
```

Si se obtiene un error al intentar instalar los programas, es posible que sea debido a que el directorio `~/PAV/bin` no existe. Si es así, puede crearse con la orden `mkdir -p ~/PAV/bin`.

Si tenemos el directorio `~/PAV/bin` en la variable de entorno `PATH`, podremos ejecutar el programa `get_pitch` directamente[1]:

```
usuario:~/PAV/P3/$ get_pitch
Arguments did not match expected patterns

get_pitch - Pitch Estimator

Usage:
    get_pitch [options] <input-wav> <output-txt>
    get_pitch (-h | --help)
    get_pitch --version

Options:
    -h, --help  Show this screen
    --version   Show the version of the project

Arguments:
    input-wav   Wave file with the audio signal
    output-txt  Output file: ASCII file with the result of the estimation:
                    - One line per frame with the estimated f0
                    - If considered unvoiced, f0 must be set to f0 = 0
```

Vemos que, por ahora, el programa requiere dos argumentos: el fichero `WAVE` de entrada y el fichero `f0` en el que se escribirá el resultado de la estimación de pitch.

Ejecutamos el programa con los argumentos correctos. Para ello, podemos usar el fichero `prueba.wav`:

```
usuario:~/PAV/P3/$ get_pitch prueba.wav prueba.f0
```

Si visualizamos el fichero generado `prueba.f0`, usando por ejemplo `less` u otro programa semejante, vemos que consiste de 200 líneas iguales a 0. Eso es así debido a que al programa, en su estado actual, le faltan las funciones necesarias para llevar a cabo la estimación. Evidentemente, si analizamos el resultado con el programa `pitch_evaluate`, éste es un desastre:

```
usuario:~/PAV/P3/$ pitch_evaluate prueba.f0ref
--- Compare prueba.f0ref and prueba.f0
```

[1] Y si `~/PAV/bin` no está incluida en `PATH`, es un buen momento para modificar el fichero de arranque de sesión (habitualmente, `~/.bashrc` o `~/.profile`) para que lo esté.

```
Num. frames:    200 = 113 unvoiced + 87 voiced
Unvoiced frames as voiced:      0/113 (0.00 %)
Voiced frames as unvoiced:      87/87 (100.00 %)
Gross voiced errors (+20.00 %): 0/0 (0.00 %)
MSE of fine errors:     0.00 %

===>    prueba.f0:      0.00 %
--------------------------
```

Vemos que todas las tramas de señal se han considerado como sordas. Por tanto, la tasa de confusiones de sordo por sonoro es cero, pero la tasa de confusiones de sonoro por sordo es del 100 %, con lo que la tasa total es 0 %.

Construcción de un primer estimador de pitch operativo.

Cálculo de la autocorrelación.

El primer \TODO que se debe implementar es del cálculo de la autocorrelación en el método `autocorrelation()` de la clase `PitchAnalyzer`, definido en el fichero `src/get_pitch/pitch_analyzer.cpp`. Existen distintas alternativas para el cálculo de la autocorrelación (sesgada, no sesgada, covarianza, etc.). El método más estándar es la autocorrelación sesgada, que, para señales reales, tiene la forma:

$$r[l] = \sum_{n=0}^{N} x[n]x[n+l] \tag{1}$$

Determinación del máximo de la autocorrelación.

A continuación, es necesario determinar la posición del máximo de la autocorrelación fuera del lóbulo principal de la misma (\TODOdel método `compute_pitch()` de la clase `PitchAnalyzer`. Existen distintas alternativas: uso de un bucle al estilo de C, uso de un iterador de C++, uso de la función `std::max_element()` definida en el estándar de C++... Aparentemente, el código proporcionado favorece el uso de iteradores, o sea que se recomienda usarlos.

Implementación de la decisión sonoro/sordo.

Finalmente, el método `unvoiced()` de esa misma clase debe devolver `true` si se considera que la trama es sorda y `false` si se considera que es sonora. Esta decisión se toma en base a la potencia de la señal, la autocorrelación normalizada de 1 y la autocorrelación en el máximo de la autocorrelación. Tal y como se proporciona, el código devuelve siempre el valor `true`, con lo que se considera que todas las tramas son sordas.

Como *apaño* provisional, hacemos que `unvoiced()` devuelva siempre `false`. Es decir, consideramos que todas las tramas son sonoras (excepto la primera y la última que, por cómo está implementado el programa principal, siempre se consideran sordas).

Ejecución y evaluación del estimador de pitch inicial.

Al ejecutar el programa con estas modificaciones ya se obtiene algo más o menos coherente:

```
usuario:~/PAV/P3/$ get_pitch prueba.wav prueba.f0
usuario:~/PAV/P3/$ pitch_evaluate prueba.f0ref
--- Compare prueba.f0ref and prueba.f0
Num. frames:    200 = 113 unvoiced + 87 voiced
Unvoiced frames as voiced:      111/113 (98.23 %)
Voiced frames as unvoiced:      0/87 (0.00 %)
Gross voiced errors (+20.00 %): 2/87 (2.30 %)
MSE of fine errors:     3.16 %

===>    prueba.f0:      13.99 %
---------------------------
```

Ahora el problema es que (casi) todas las tramas han sido calificadas como sonoras.

Resto de código a implementar.

Una vez tenemos un primer estimador de pitch, el objetivo es optimizar sus prestaciones. Para ello, debemos implementar el resto de \TODOs:

- Mejorar la decisión sordo/sonoro en el método `unvoiced()` para que tenga en cuenta los argumentos que se le pasan (y otros que se puedan considerar oportunos, como la tasa de cruces por cero). Se trata de construir una regla de decisión a partir de uno o más de ellos. Por ejemplo, podemos poner un umbral mínimo a los tres argumentos y decidir que la trama es sonora sólo si se superan los tres, o si se superan dos de ellos, o si se supera uno de ellos concreto y, como mínimo, uno de los otros dos, o si...
  - Para optimizar estos umbrales, y otros parámetros del estimador de pitch, puede ser conveniente añadir opciones al programa empleando la biblioteca `docopt` (lo cual constituye otro \`TODO` opcional).
- Implementar el envantanado de Hamming y comprobar si éste aporta algún beneficio.
- Implementar métodos de preprocesado y postprocesado adecuados.
- Implementar métodos alternativos al de la autocorrelación, como el cepstrum o el AMDF.
- Cualquier otra idea que resulte en una mejora de la estimación. Recuérdese que la nota dependerá, en gran medida, del resultado obtenido en la evaluación ciega de la misma.