Práctica 1: análisis por bloques

Antonio Bonafonte

21 de febrero de 2018

Resumen

En esta práctica:

- Recordaremos las bases de programación en `C`: compilar, enlazar, ficheros de texto y binarios, struct, punteros, memoria dinámica.
- Veremos como pasar argumentos al programa, por la línea de comandos, mapas de bits, qué significa *little* o *big endian*
- Analizaremos cómo se guarda la información en un fichero `.wav`
- Introduciremos el procesado por bloques (tramos) calculando características temporales sencillas (contorno de potencia, cruces por cero, etc.)
- Integraremos en `wavesufer` la característica calculada.
- Programas: `audacity, wavesurfer, bless, sox, atom, gedit, gcc, gnuplot, ipython`

En esta y en las prácticas siguiente, puede instalar utilizando `sudo` *los programas que necesite y no estén instalados. En el anexo 5.5 encontrará las instrucciones.*

1. Formato de fichero `.wav`

1. Utilizando el programa `audacity` o `wavesurfer`[1] grabe un fichero `.wav`, a 16kHz, 16bits, mono.

   Grabe un párrafo que empiece por su nombre y el del compañero de prácticas y a continuación una o dos frases cortas, con pausas intermedias. Ajuste el nivel de grabación para que la señal tenga un nivel suficiente pero sin saturar. Colóquese adecuadamente el micrófono, para que no aparezcan ruidos por roces o golpes.

   Si ha grabado en otra frecuencia de muestreo, puede aprovechar para conocer `sox`, *Sound eXchange, the Swiss Army knife of audio manipulation*, un programa que permite realizar cambios de formato, y procesar audio con filtros, reducir ruido, cambiar velocidad, mezclar, etc. La siguiente linea asegura que es mono, de 16 kHz.

[1] `audacity` es un programa de grabación y edición de audio multipista. `wavesurfer` es un sistema de análisis de voz, que permite analizar el contorno de potencia, pitch, formantes, etc., aunque es un programa no activo, sin mantenimiento

```
sox input.wav -r 16k -c 1 output-16k-mono.wav
```

2. Utilize el programa `bless` u otro editor hexadecimal para representar el contenido del fichero `wav`. Interprete los bytes de la cabecera. Puede consultar el formato del fichero wave en diversas fuentes, por ejemplo, el documento de atena, obtenido de un curso de Stanford University. Seleccione la opción *little endian*: least-significant byte in smallest address.

   En `bless`, la opción `"Search / Goto offset"` le permite situar el cursor en una posición concreta. Si está activa `"Tools / Conversion table"` se muestra la correspondencia de los bytes que siguen al cursor, con distintos tipos de datos (signed 16 bits, etc.)

2. Extracción de características

2.1. Funciones de análisis

Escriba un *módulo* con tres funciones en `C` para calcular la potencia, la tasa de cruces por cero y la amplitud media. Instale y utilice el programa de edición `atom` o el que suela utilizar. En el anexo 5.1 encontrará algún comentario sobre programas de edición.

Cree tanto el fichero `.c` como el `.h` con la declaración de las funciones, para poder utilizarlas en varios programas.

`pav_analysis.h`

```
#ifndef PAV_ANALYSIS_H
#define PAV_ANALYSIS_H
float compute_power(const float *x, unsigned int N);
float compute_am(const float *x, unsigned int N);
float compute_zcr(const float *x, unsigned int N);
#endif
```

`pav_analysis.c`

```
#include "pav_analysis.h"

float compute_power(const float *x, unsigned int N) {
../..
```

Las expresiones de estas características son:

Potencia (dB FS)	$10 \log \frac{1}{N} \sum_{n=0}^{N-1} x^2[n]$
Amplitud media	$\frac{1}{N} \sum_{n=0}^{N-1} \lvert x[n] \rvert$
Tasa de cruces por cero	$\frac{f_m}{2} \frac{1}{N-1} \sum_n 1 \quad \forall n: 0 < n < N, \text{sgn}\{x[n-1]\} \neq \text{sgn}\{x[n]\}$

siendo f_m la frecuencia de muestreo. Dado que la función no dispone de este valor, la función devuelve la frecuencia discreta (sin multiplicar por f_m) y es el programa que le llama el que debe realizar esta multiplicación.

Puede comprobar que no hay errores de sintaxis, compilando (véase anexo 5.2):

```
gcc -I. -c pav_analysis.c
```

2.2. Programa principal

Escriba ahora un programa que lea las muestras del fichero `wav` y escriba, para cada tramo de 10 ms, las características anteriores.

La llamada debe ser (el argumento entre corchetes, es opcional):

```
pav_analysis input.wav [output.txt]
```

A grandes rasgos, el algoritmo sería:

```
open input and output files (fopen)
while(1) {
  read N samples associated to 10ms; (fread)
  if not enough samples: break out of while
  normalize amplitude to +/- 1
  compute features
  write features (fprintf)
}
close files (fclose)
```

- El programa debe comprobar que recibe uno o dos argumentos (`argc = 2 o 3`). El primero es el nombre del fichero de entrada `.wav` y el segundo, opcional, el nombre del fichero de salida, `.txt`.

```
int main(int argc, const char *argv[]) {
  ../..
  if (argc != 2 && argc != 3) {
     fprintf(stderr,"%s: inputfile.wav [outputfile.txt]\n", argv
[0]);
     return -1;
  }
  ../..
```

  Para considerar si se indica o no el nombre del fichero de salida, una opción es utilizar para escribir las características `fprintf(fout, ...)`, y según el número de argumentos asignar `fout` a `stdout` o a un fichero llamando a `fopen()`.

- Puede analizar la información de la cabecera de fichero (44 bytes, si no hay etiquetas *metadatos*), leyendo primero 44 bytes en una variable `char header[44]` y de allí extraer frecuencia de muestreo, número de canales, etc.[2] Alternativamente, puede realizar un programa específico para el fichero que ha grabado, ignorando la cabecera y asumiendo conocidas las características de este fichero particular: PCM, 16kHz, 16 bits/muestra, mono). En ese caso, para descartar la cabecera puede utilizar la función `fseek`.

- La señal se guarda en un fichero binario, con 16 bits por muestra. Por ello, deberá leerla usando la función `fread` guardando los datos en un buffer de `short`. Para muchas aplicaciones de procesado, es conveniente convertirlas en el tipo `float`: normalizando por el valor máximo posible de las muestras, se obtiene una señal entre ± 1.

[2] Por ejemplo: `int rate, *pt = (int *) (header + 24); rate = *pt;`, o con notación vector, `int rate, *pt = (int *) &header[24]; rate = pt[0];`

```
   #define NSAMPLES 160 /* 10 ms, fs=16kHz */
   int main(int argc, const char *argv[]) {
   short buffer[NSAMPLES];
   float x[NSAMPLES];
   int i;
   float norm_factor = 1.0/ (float) 0x8000; /* 16 bits: 0x8000 =
2^15 */

   ../..
   for (i=0; i<NSAMPLES; i++) x[i] = (float) buffer[i] *
norm_factor;
```

- En el fichero de salida, escriba una línea por tramo con cuatro columnas separadas por tabulador (`'\t'`), con un contador de tramo, potencia, amplitud media y tasa de cruces por cero.

Una vez finalizado, compile el programa.

3. Representación de las características

3.1. Representación mediante programas gráficos

Ejecute su programa, guardando las características en un fichero de texto con el mismo nombre que el fichero `.wav`, pero con otra extensión (por ejemplo, `.feat`). Puede ejecutar el programa pasando como argumento el fichero de salida, o bien utilizar la opción de Linux, de redireccionar la salida mediante `'>'`. Por ejemplo,

```
./pav_analysis mifichero.wav > mifichero.feat
```

Visualice con un editor de texto (o mediante el comando `less mifichero.feat`) el fichero generado.

En Linux, es fácil seleccionar una de las columnas del fichero, que corresponde a cada una de las características, mediante redireccionado o *pipeline* al programa `cut`. Por ejemplo:

```
cut -f 3 mifichero.feat > mifichero.am
```

o directamente ejecutando el programa:

```
./pav_analysis mifichero.wav | cut -f 3 > mifichero.am
```

Represente las características mediante algún programa externo (véase el anexo 5.4).

3.2. Representación integrada en `wavesurfer`

Represente ahora la potencia integrada en `wavesurfer`, sincronizada con la señal. Para ello, tenemos que separar las columnas que queremos representar en cada panel: en este caso, como tienen distinta escala, cada característica iría en un panel distinto, por tanto lo necesitamos en un fichero distinto.

En Linux, si tenemos que ejecutar varias ordenes, se suelen agrupar en un fichero o *script*. Edite por ejemplo `run.sh`, y escriba en él unas líneas como las siguientes:

```
#!/bin/bash

# ponga aqui el nombre de su fichero.wav sin extension
```

```
F=mifichero

./pav_analysis $F.wav $F.feat
cut -f 2 $F.feat > $F.pot
cut -f 3 $F.feat > $F.am
cut -f 4 $F.feat > $F.zcr
exit 0
```

Cambie los permisos para que sea ejecutable (con el navegador gráfico, o mediante el comando `chmod +x run.sh`) y ejecute `./run.sh`

Abra `wavesurfer` con su fichero `.wav` (`padsp wavesurfer mifichero.wav &`) y cree un panel del tipo `data plot`, indique en `properties` el espaciado temporal en los datos de su fichero de característica (10 ms), y la extensión del fichero. Puede crear varios paneles para las distintas características, y guardar la configuración de `wavesurfer`, que podría utilizar en otro momento con ese u otros ficheros.

Compárela con la potencia que calcula el propio. `wavesurfer`, mediante el panel `power plot`. Consulte las propiedades de este panel y compare con su el método que hemos utilizado.

4. Ampliación:

Para evitar *ruido* debido a los bordes del tramo, típicamente la potencia se calcula utilizando una *ventanas* $w[i]$, por ejemplo, Hamming, solapadas. Es decir, cada M muestras se selecciona el tramo de señal $l = 0, 1, \ldots,$ de duración N muestras (ej.: $M = 0{,}01 \cdot f_m; N = 2M$), se enventana y se calcula la potencia del tramo l:

$$x_l[n] = x[lM + n] \quad 0 \leq n < N$$

$$p[l] = 10 \log_{10} \frac{\sum_{i=0}^{N-1} (x_l[i]w[i])^2}{\sum_{i=0}^{N-1} w^2[i]}$$

Modifique su programa para calcular el contorno de potencia utilizando ventanas solapadas. No cambie la función `compute_power` sino que añada dos funciones nuevas:

- una función para definir ventana en una variable (`w[n]`), así como el valor de denominador, $-10 \log_{10} \sum_{i=0}^{N-1} w^2[i] = -p_w$, que se ejecuta una única vez por programa;
- y otra, para aplicar la ventana a las muestras, a cada tramo de señal.

5. Anexo

En este anexo vamos a comentar algunos aspectos básicos para el desarrollo en Linux, edición, compilado, ejecución, representación gráfica e instalación de programas.

5.1. Edición de código fuente

Para editar código fuente existen `IDE`, entornos de desarrollo como `eclipse, codeblocks, codelite`, etc., que pemiten editar, compilar, ejecutar, depurar. Sin embargo, en esta asignatura invocaremos explícitamente las herramientas básicas.

Algunas de las opciones para editar en Linux son:

`vim` Es un editor muy potente, que existía en unix (en la era *pre-mouse*) y permite utilizarlo sin entorno gráfico, así que si se conoce puede utilizarse en conexión a servidores, etc. En la versión no gráfica, hay que recordar comandos.

`emacs`[3] Otro potente editor *pre-mouse*, que permite configurarlo para multitud de tareas. Por ejemplo, para ajustar automáticamente la sangría de los programas, compilar, comparar ficheros, etc.

`gedit` Es el que se abrirá por defecto para ficheros de texto. Configúrelo para mostrar números de línea, no permitir ajuste de línea (*wrapping*), resaltar *matching brackets*, espacio de tabulación 3, permitir sangrado automático (*indentation*).

`sublime` Es un editor más moderno y potente, disponible en distintas plataformas (linux, windows, os x). No es gratuito, pero puede probarse para su evaluación. Puede descargarse el paquete `.deb` desde su página web e instalarlo (doble click o `dpkg -i fichero.deb`)

`atom` *A hackable text editor for the 21st Century.* Moderno, gratuito, configurable, fácil de integrar con `github` ... Descarga el paquete para Linux: `https://atom.io/download/deb` e instalalo con `sudo dpkg -i` *nom_fichero_atom.deb*

5.2. Compilar y enlazar (link)

El compilador que utilizaremos para compilar `C` es `gcc` y para `C++`, `g++`.

Algunas instrucciones básicas:

- `gcc fichero.c -o miprograma`:
  compila y enlaza, creando el ejecutable `miprograma`. Para ejecutarlo, si en la variable de entorno `PATH` está el directorio donde se encuentra el programa, basta con escribir su nombre. Si no lo está, se ha de indicar, bien de forma absoluta (ej. `/home/pav113/p1/miprograma`) o relativa (ej. `./miprograma`, ó `../p1/miprograma`)

- `gcc fichero.c -lm -o miprograma`:
  `gcc fichero.c /usr/lib/libm.a -o miprograma`:
  En ocasiones, al enlazar, hay que indicar librerías con funciones auxiliares, que pueden ser nuestras o del sistema. Por defecto, `gcc` incluye muchas de las funciones de

[3] ¿`vim` o `emacs`? No tiene sentido perpetuar esta guerra de editores, ... pero `emacs` es mejor ;-) Como dice Richard Hendricks, fundador de *Pied Piper*: ¿`vim`? *God, help us!*. `https://www.youtube.com/watch?v=3r1z5NDXU3s`

la librería estandar, como `fprintf` o `calloc`. Sin embargo, para las funciones matemáticas, como `log10` o `cos`, se debe indicar explícitamente que se utilice la librería `libm.a`.

- `gcc *.c -lm -o miprograma`
  En este caso se compilan y enlazan todos los ficheros del directorio. Si se incluyen ficheros propios con declaraciones (*headers*) se debe indicar el directorio en que se encuentran con la opción `-I`. Por ejemplo:
  `gcc -I. -Iinclude-dir *.c -lm -o miprograma`

- `gcc -c fichero_principal.c`
  `gcc -c fichero_axiliar.c`
  `gcc fichero_principal.o fichero_auxiliar.o -lm -o miprograma`
  Cuando los programas están compuestos de varios ficheros fuentes, es muy ineficiente compilar todos ellos cada vez que hacemos un cambio en uno. Los comandos anteriores separan el procedimiento entre compilación (opción `-c`), que produce los ficheros `fichero_principal.o` y `fichero_auxiliar.o`, y el enlazado, que produce el programa ejecutable. Si por ejemplo, cambiamos algo en `fichero_auxiliar.c`, deberemos ejecutar el segundo y tercer comando.

- `gcc -g fichero.c -lm -o miprograma`
  Se guardan unos símbolos en el ejecutable que permite depurarlo con herramientas como la gráfica `ddd` o la de consola `gdb`

- `gcc -O fichero.c -lm -o miprograma`
  El compilador intenta optimizar el código para que ocupe menos y sea más rápido. Hay opciones especificas sobre la optimización. Por ejemplo, `-O3`, optimiza lo máximo, aunque supondrá más tiempo de compilación.

5.3. Ejecutar Programas desde el terminal

Desde el terminar, debe escribir el nombre del programa. Si el directorio en que se encuentra el programa no está incluido en la variable de entorno `PATH` entonces deberá indicar el directorio de forma explícita, bien de forma absoluta o relativa. Por ejemplo:

`/home/pav112/practica1/miprograma`	
`./miprograma`	(en directorio actual)
`../practica1/miprograma`	(en directorio `practica1`, que es parte de directorio *padre* del actual.)

El directorio actual suele aparecer en el *prompt*, o puede consultarlo usando el comando `pwd`.

Es muy habitual configurar el sistema para que al menos el directorio actual forme parte de los directorios de búsqueda. Para ello, edite el fichero `.bashrc`, en su directorio `home` y escriba en la variable `PATH` los directorios que quiera que formen parte de la búsqueda de programas, separados por ":". Por ejemplo, añadir el directorio actual y los predefinidos:

```
export PATH=".:$PATH"
```

Los cambios en `.bashrc` tendrán lugar en las sesiones de terminal nuevas, o bien ejecute `source .bashrc`.

Los argumentos, que se reciben en `C` en las variables `argc, argv` de la función `main()`, se introducen a continuación:

```
miprograma 12 fichero.txt salida.txt
```

Muchos programas, suelen permitir argumentos opcionales mediante letras precedidas por guión (o doble guión y nombre completo de opción). Por ejemplo, `ls *.c` muestra los ficheros `.c` del directorio. `ls -l -h -t *.c` (o equivalentemente `ls -lht .c`) muestra atributos del fichero, con formato *más humano*, y ordenados por fecha de modificación. La opción `-h` (o en ocasiones ejecutar programa sin argumento) suele mostrar un pequeño mensaje de ayuda. Puede probar `cut -h` o `apt-get -h` o `apt-get`. Hay varias funciones en `C` que ayudan a programar el tratamiento de opciones. La más sencilla, `getopt` o `getopt_long`

5.4. Representación gráfica

Existen multitud de programas para representar gráficas. Comentemos alguno de ellos. Para ilustrarlo generaremos en `matlab` el fichero `polynomial.txt`, de 4 columnas, con x, x^2, x^3 y x^4, con $-1 \leq x \leq 1$.

```
x = [-1:0.1:1]';
dat = [x, x.^2, x.^3, x.^4];
save  '-ascii' 'polynomial.txt' dat
```

5.4.1. Python + Matplotlib

Python es un lenguaje de programación interpretado que permite expresar programas complejos con pocas instrucciones. Se apoya en multitud de paquetes ya compilados, lo que en muchos casos es muy eficiente. En cierto sentido es similar a `matlab`, pero es un lenguaje de programación más moderno y estructurado (incluyendo clases, etc.) y además es libre y gratuito. [4]

Ejecute `ipython3` (*iterative python, version 3*)

A continuación carge las librería para representación `matplotlib` y el paquete `numpy`, para cálculo numérico y matricial.

Si al usarlos no los encuentra, puede instalarlos. Para `python3`:

```
sudo apt install python3-matplotlib python3-numpy
```

Se recomienda realizar un tutorial sobre estos paquetes [5], pero para representar características es suficiente que utilize, desde el *prompt* de `ipython3` la función `plt.plotfile`. Por ejemplo:

[4] Recomendamos usar Python pues es un lenguaje cada vez más utilizado, siendo por ejemplo el más extendido en aplicaciones de *deep learning*, al existir APIs a librerías muy eficientes (como *Tensorflow*, *pytorch*, etc.).

[5] https://matplotlib.org/users/pyplot_tutorial.html

```
import numpy as np
import matplotlib.pyplot as plt
plt.plotfile('polynomial.txt', cols=(0,0,1,2,3), delimiter=' ',
             subplots=False, names = ('x', 'x2', 'x3', 'x4'))
plt.show()
```

Utiliza la primera columna (identificador 0) en eje abcisas, y en eje de ordenadas la misma columna (x), la segunda (x^2), etc. Si `subplots` es `False` se dibujan todas las gráficas en los mismos ejes. Si es `True`, se utilizan `subplots`, con varias gráficas.

Puede obtener más información, desde `ipython`, usando `help(plt.plotfile)`

Si el fichero sólo es de una columna, puede prescindirse de muchas opciones. Por ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
plt.plotfile('fic_datos.txt')
plt.show()
```

5.4.2. Octave

`Octave` es un programa libre y gratuito que emula `matlab`, aunque más limitado que este. Un ejemplo:

```
load "polynomial.txt"
x = polynomial;
plot(x(:,1), x)
legend('x', 'x2', 'x3', 'x4')
```

5.4.3. gnuplot

Es un pequeño programa específico para representar funciones. Como los anteriores, puede indicarse de forma analítica o a partir de un fichero. Ejecutar `gnuplot` y desde el prompt:

```
plot "polynomial.txt" using 1:1 with lines title "x", \
                      '' using 1:2 with lines title 'x2',\
                      '' using 1:3 with lines title 'x3',\
                      '' using 1:4 with lines title 'x4'
```

Si el fichero sólo es de una columna, puede usar:

```
plot "fic_datos.txt" with lines
```

5.5. Instalación de programas

Muchos de los programas que utilizaremos estarán instalados en su ordenador. Los que no estén, si están en el repositorio de ubuntu, puede instalarlos mediante las herramientas

gráficas de administración del sistema, o desde la consola mediante `apt-get`. Por ejemplo, para instalar `g++`:

```
sudo apt-get update; sudo apt-get install g++;
```

Al ejecutar `sudo` le pedirá su contraseña.

Si no está en el repositorio pero tiene un paquete de formato *debian*, `.deb`, puede utilizar

```
sudo dpkg -i fichero.deb
```

Si dispone de ficheros fuentes, suelen incluir instrucciones de instalación. Frecuentemente lo que debe hacer es, configurar los ficheros de compilación para su ordenador, compilar, e instalar en directorio del sistema:

```
./configure
make
sudo make install
```