Memoria Práctica 5

Marcos Rico Guerra

Introducción

En esta práctica, nuestro objetivo es aprender a trabajar con ficheros de audio(mas concretamente con ficheros de tipo WAV), para poder analizarlos(El número de muestras, su frecuencia de sampleo,etc.) y manipularlos para unirlos a otros audios, cortarlos, o incluso darle efectos como añadir eco o darles la vuelta.

Para realizar el trabajo, se usará el lenguaje R, usando las bibliotecas “tuneR”, “seewave” y “audio”.

Lectura de ficheros

Para empezar, lo primero que necesitamos para trabajar con ficheros de audio es conseguir que el programa los reconozca y los guarde en una variable con la que podamos usar las funciones correspondientes. Para ello usaremos la función readWave de tuneR donde pasamos como parámetro el nombre del archivo de audio que queremos leer. El resto de parámetros se dejan por defecto, ya que nos interesan así (Aunque podríamos cambiarlos para leer solo una parte del audio por ejemplo)

```
nombre <- readwave('Nombre.wav')
apellido <- readWave('Apellido.wav')
```

Para el ejercicio, he creado dos audios, uno donde digo mi nombre y otro mi apellido.

Como vemos, usamos la función indicada para guardar los audios en dos variables.

Análisis de los audios

Lo siguiente que hacemos es analizar los audios que acabamos de leer. Tenemos varias posibilidades:

Podemos simplemente poner el nombre de la variable y el programa hace un print de sus parámetros:

```
nombre
apellido
```

```
> nombre

Wave Object
        Number of Samples:      86400
        Duration (seconds):     1.8
        Samplingrate (Hertz):   48000
        Channels (Mono/Stereo): Stereo
        PCM (integer format):   TRUE
        Bit (8/16/24/32/64):    16
```

Otra forma es mirando la cabecera con la función str():

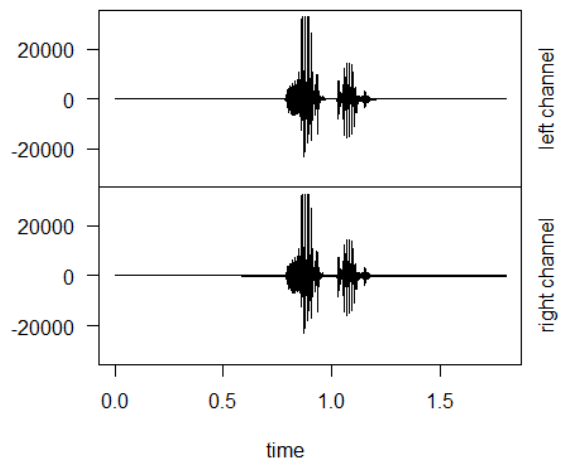
```
str(nombre)
str(apellido)
```

```
> str(nombre)
Formal class 'Wave' [package "tuneR"] with 6 slots
  ..@ left     : int [1:86400] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ right    : int [1:86400] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ stereo   : logi TRUE
  ..@ samp.rate: int 48000
  ..@ bit      : int 16
  ..@ pcm      : logi TRUE
```

Como vemos ambas opciones nos dan casi los mismos datos

Otra forma de analizar nuestro archivo de audio es ver su forma de onda con la función plot que sirve para mostrar gráficos simples, y aplicando la función extractWave al archivo para obtener su forma de onda:

```
plot(extractWave(nombre, from=1, to=86400))
plot(extractWave(apellido, from=1, to=110880))
```

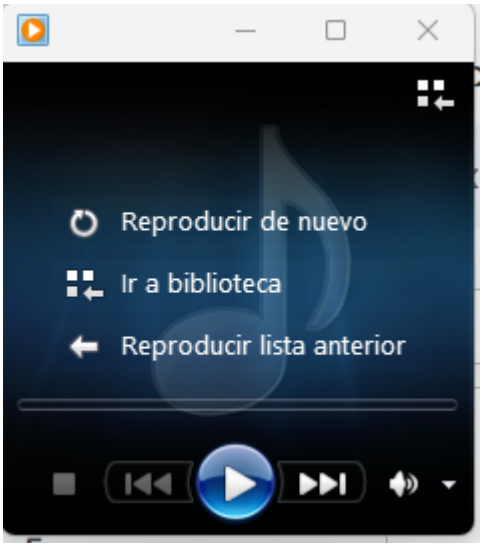


En los parámetros debemos indicar que fragmento queremos. Para coger todo el audio, indicamos el índice de la última muestra de audio que viene indicada por ejemplo en Number of Samples.

Por último, podemos hacer aquello para lo que están hechos los audios; escucharlos:

```
listen(nombre)
listen(apellido)
```

Esto abrirá un reproductor de audio que reproducirá el audio en cuestión



Unión de sonidos

Lo siguiente que debemos hacer es unir ambos sonidos en uno solo. Como ambos han sido grabados con la misma aplicación, tienen exactamente la misma frecuencia de muestreo por lo que podemos unirlos sin problemas.

Para unirlos, haremos uso de la función pastew(). Esta añade el primer sonido que recibe justo al terminar el segundo sonido que recibe como parámetro. Ademas, indicamos que el output será otro sonido en formato “wave”.

```
nom_ape <- pastew(apellido, nombre, output = "wave")
nom_ape
```

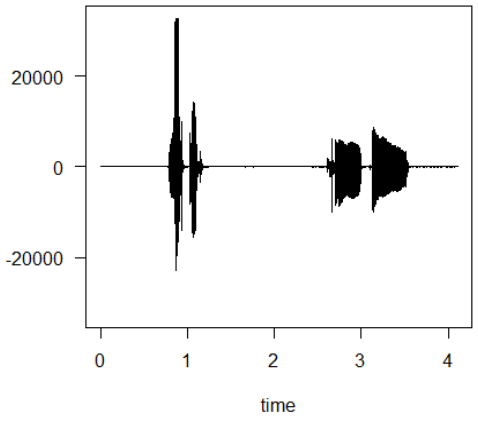
```
> nom_ape

Wave Object
        Number of Samples:      197280
        Duration (seconds):     4.11
        Samplingrate (Hertz):   48000
        Channels (Mono/Stereo): Mono
        PCM (integer format):   TRUE
        Bit (8/16/24/32/64):    16

>
```

Si nos fijamos tanto la duración como el número de muestras es la suma de estos parámetros de los dos audios que lo componen:

```
> nombre

Wave Object
        Number of Samples:      86400
        Duration (seconds):     1.8
        Samplingrate (Hertz):   48000
        Channels (Mono/Stereo): Stereo
        PCM (integer format):   TRUE
        Bit (8/16/24/32/64):    16

> apellido

Wave Object
        Number of Samples:      110880
        Duration (seconds):     2.31
        Samplingrate (Hertz):   48000
        Channels (Mono/Stereo): Stereo
        PCM (integer format):   TRUE
        Bit (8/16/24/32/64):    16
```

Por último comprobamos su forma de onda, y veremos que también es igual a la concatenación de las dos formas de los audios que lo componen:

```
plot(extractWave(nom_ape, from=1, to=197280))
```



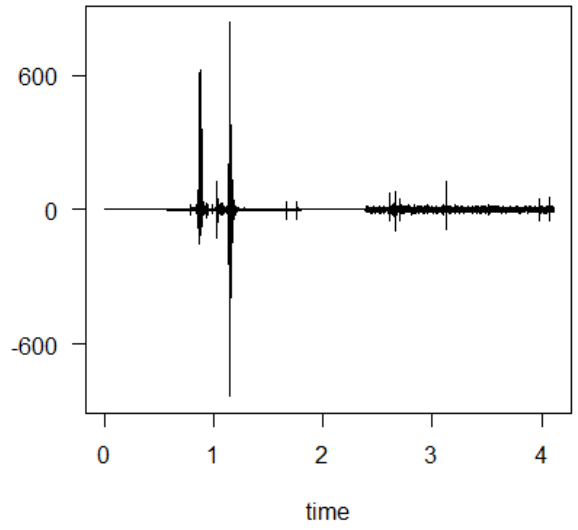
Filtro de frecuencia

Otra cosa que podemos hacer es aplicar filtros de frecuencia para eliminar sonidos de ciertas frecuencias.

En este caso, usamos la función bwfilter, que elimina las frecuencias entre 2 valores dados

```
filtrado <- bwfilter(nom_ape, f=48000, from=10000, to=20000,
                     bandpass=TRUE, listen = FALSE, output = "Wave")
> filtrado

Wave Object
        Number of Samples:      197280
        Duration (seconds):     4.11
        Samplingrate (Hertz):   48000
        Channels (Mono/Stereo): Mono
        PCM (integer format):   TRUE
        Bit (8/16/24/32/64):    16
```

Mirando en sus parámetros no notamos nada distinto, pero si vemos su forma de onda veremos que ha cambiado:



Por último, guardaremos este sonido en un archivo que llamaremos “mezcla.wav”. Esto se hará con la función writeWave(). Antes de eso, normalizamos el sonido para que no nos de errores a la hora de guardarla

```
filtrado <- normalize(filtrado, unit="16", center = TRUE, level = 1)

writeWave(filtrado, file.path("mezcla.wav") )
```

Otros efectos

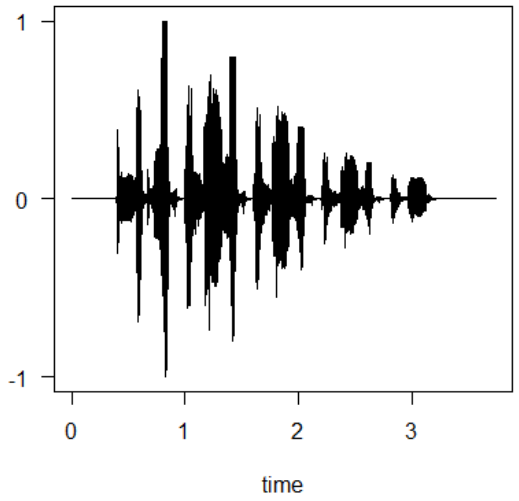
Para finalizar la práctica hemos aplicado ciertos efectos a otro sonido.
En concreto, le hemos añadido eco y se le ha dado la vuelta.

Para lo primero, hemos usado la función echo

```
burger <- readwave('burger.wav')
burgerECHO <- echo(burger,f=48000,amp=c(0.8,0.4,0.2),delay=c(0.6,1.2,1.8),
                   output="wave")
```

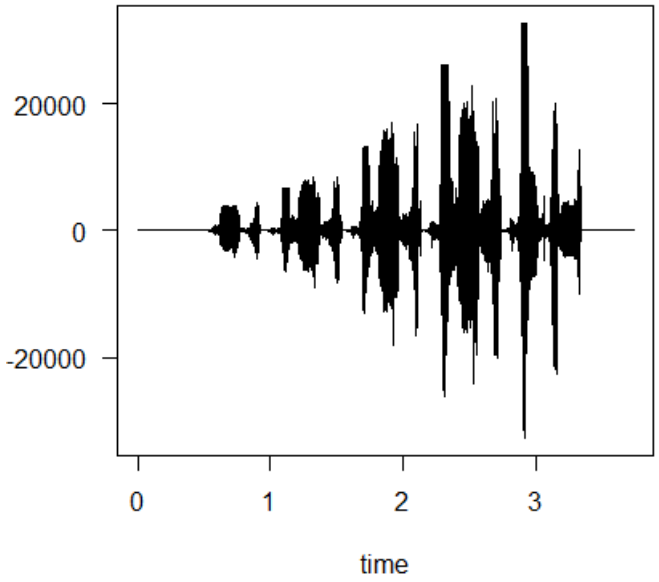
En esta, primero establecemos el vector amp donde indicamos el volumen de los sucesivos ecos, en este caso el primer eco tendrá un 80% de la amplitud de sonido que el sonido original, el segundo un 40% y el tercero un 20%. No estamos restringidos a 3 ecos, podríamos ampliar el vector todo lo que quisiéramos.
En el vector delay, indicamos cuando se va a reproducir cada eco, siendo en nuestro caso cada 0.6 segundos. Es necesario que tanto el vector amp como delay tengan la misma longitud.

```
> burgerECHO

Wave Object
        Number of Samples:      179520
        Duration (seconds):     3.74
        Samplingrate (Hertz):   48000
        Channels (Mono/Stereo): Mono
        PCM (integer format):   TRUE
        Bit (8/16/24/32/64):    16
```



Tras esto, procedemos a darle la vuelta con la función revw()

```
alreves <- revw(burgerECHO, output="Wave")
```



La forma de onda, como vemos, está al revés

Ahora solo nos queda guardarla en un archivo .wav, lo que haremos igual que anteriormente:

```
alreves <- normalize(alreves, unit="16", center = TRUE, level = 1)
writeWave(alreves, file.path("alreves.wav") )
```