Grado en Ingeniería de Tecnologías de Telecomunicación

SISTEMAS DE CODIFICACIÓN Y ALMACENAMIENTO

Codificación de señales de audio en subbandas

PRÁCTICA 4

Universidad de GRANADA

1. Introducción

La codificación en subbandas es una técnica avanzada de compresión de datos ampliamente usada para codificar señales de audio e imágenes. La idea fundamental de esta técnica es la descomposición de una señal en diferentes componentes en frecuencia (subbandas o canales), que son posteriormente procesadas y codificadas de manera independiente. De esta forma es posible aplicar técnicas de compresión específicas a cada banda de frecuencia, lo que resulta en una representación más eficiente de la señal completa. Así, la codificación en subbandas permite aprovechar las propiedades de la percepción humana, representando de manera más precisa las frecuencias a las que nuestros sentidos dan más importancia y reduciendo la precisión de las frecuencias menos perceptibles.

En esta práctica vamos a implementar un codificador en subbandas para señales de audio, similar al del estándar ITU-T G.722. Este codificador divide las señales en dos bandas que son posteriormente procesadas mediante un codificador PCM adaptable. Una vez implementado, evaluaremos su rendimiento en términos de la tasa de bits de la señal codificada y de la relación señal/ruido de la señal reconstruida (tras el proceso de codificación/descodificación en subbandas). Los resultados obtenidos con este esquema serán comparados con los obtenidos por un codificador PCM adaptable, esto es, con un esquema de codificación similar pero que trata a toda la señal por igual, sin dividirla en dos bandas de frecuencia.

2. PCM adaptable (APCM)

La cuantificación adaptable es una variante de la cuantificación uniforme en la que el tamaño del cuanto se adapta, de forma dinámica, a las características de la señal. La implementación física de este enfoque es compleja dado que requiere modificar dinámicamente un parámetro esencial del diseño del cuantificador, el cuanto. Un enfoque conceptualmente similar y más sencillo de implementar, consiste en modificar la señal de forma dinámica, aplicándole una ganancia variable, para que se adapte al rango de operación de un cuantificador uniforme estándar (de cuanto fijo). Así, la señal se adapta al cuantificador, en lugar de adaptar el cuantificador a la señal, obteniendo el mismo efecto.

El parámetro variable, ya sea el cuanto o la ganancia, se calcula a partir de una ventana de muestras de longitud preestablecida y, cuando la adaptación se realiza hacia adelante (basada en la señal de entrada del codificador), ese parámetro debe transmitirse/almacenarse junto a las muestras de la señal para que el descodificador pueda recuperarlas a partir del código. Por tanto, debe ser cuantificado y codificado al igual que las muestras de la señal. Los Algoritmos 1 y 2 muestran respectivamente el pseudocódigo de un codificador y un descodificador PCM adaptable. Observe cómo se usan dos cuantificadores (y dos codificadores) diferentes, uno para las muestras de la señal y otro para las ganancias, con tasas de b y b_g bit/muestra respectivamente.

Algoritmo 1: Codificador PCM adaptable

Input: Vector con las muestras de la señal, s

Tamaño de la ventana, w_{len}

Tasa de bits para los datos, b

Tasa de bits para las ganancias, b_g

Output: Vector con la señal codificada, c

$c \leftarrow \{\}$ // Inicializa el vector de código

for $k \leftarrow 0$ **to** $|s|$ **by** w_{len} **do**

- $w \leftarrow \{s_k, s_{k+1}, ..., s_{k+w_{len}-1}\}$ // Ventana actual: w_{len} elementos de s empezando en k
- $g \leftarrow \text{máx}(\text{abs}(w))$ // Calcula la ganancia como el valor máximo en valor absoluto
- $w \leftarrow \frac{w}{g}$ // Aplica ganancia a los datos de la ventana actual
- $i_w \leftarrow Q_{encode}(b, w)$ // Cuantifica (mapeo del codificador) con b bits los datos
- $i_g \leftarrow Q_{encode}(b_g, g)$ // Cuantifica (mapeo del codificador) con b_g bits la ganancia
- $c_w \leftarrow C_{encode}(b, i_w)$ // Codifica con b bits los datos cuantificados
- $c_g \leftarrow C_{encode}(b_g, i_g)$ // Codifica con b_g bits la ganancia cuantificada
- $c \leftarrow c \cup c_g \cup c_w$ // Acumula el código de los datos y la ganancia de la ventana actual

end

Ejercicio 1

Siguiendo el pseudocódigo de los Algoritmos 1 y 2, implemente un codificador y un descodificador PCM adaptable para señales de audio. Para ello, cree dos funciones con la siguiente interfaz:

```
def encoderAPCM(data, wlen, dataRange, b, bg):
...
code = ...
return code
```

```
def decoderAPCM(code, wlen, dataRange, b, bg):
...
data = ...
return data
```

donde:

- `data` → Array unidimensional que contiene la señal de audio que vamos a codificar.
- `code` → Array unidimensional con la señal codificada.
- `wlen` → Tamaño de la ventana.
- `dataRange` → Tupla de 2 elementos con el mínimo y el máximo valor posible de las muestras.
- `b` y `bg` → Tasa de bit/muestra de los datos y de las ganancias.

Use estas funciones para codificar y descodificar la señal de audio almacenada en `marlene.wav` con diferentes tasas de bits para los datos, una tasa de 16 bit/muestra para las ganancias y una ventana de 120 muestras. En cada caso, calcule la SNR resultante y compárela con la obtenida por un codificador PCM estándar (no adaptativo) como el implementado en prácticas anteriores.

Algoritmo 2: Descodificador PCM adaptable

Input: Vector con la señal codificada, c
Tamaño de la ventana, w_{len}
Tasa de bits para los datos, b
Tasa de bits para las ganancias, b_g

Output: Vector con las muestras de la señal, s

$s \leftarrow \{\}$ // Inicializa el vector de muestras de la señal

for $k \leftarrow 0$ **to** $|c|$ **by** $w_{len} \cdot b + b_g$ **do**

- $c_g \leftarrow \{c_k, c_{k+1}, ..., c_{k+b_g-1}\}$ // Secuencia de bits correspondiente a la ganancia
- $c_w \leftarrow \{c_{k+b_g}, c_{k+b_g+1}, ..., c_{k+b_g+w_{len} \cdot b-1}\}$ // Secuencia de bits correspondiente a los datos
- $i_w \leftarrow C_{decode}(b, c_w)$ // Descodifica los datos
- $g_w \leftarrow C_{decode}(b_g, c_g)$ // Descodifica la ganancia
- $w \leftarrow Q_{decode}(b, i_w)$ // Cuantifica (mapeo del descodificador) los datos
- $g \leftarrow Q_{decode}(b_g, g_w)$ // Cuantifica (mapeo del descodificador) la ganancia
- $w \leftarrow w \cdot g$ // Aplica ganancia a la ventana actual
- $s \leftarrow s \cup w$ // Acumula los datos de la ventana actual

end

3. Codificación en subbandas

La codificación en subbandas requiere la división de la señal en diferentes componentes en frecuencia, que son procesadas y codificadas de manera independiente. Posteriormente, en la etapa de descodificación, es necesario combinar las componentes en frecuencia en las que fue dividida la señal, para recuperar la señal original. Ambos procesos, la división en componentes y la combinación de las mismas, se llevan a cabo mediante bancos de filtros. Los filtros usados para la división de la señal en componentes en frecuencia se conocen como filtros de análisis, mientras que los usados para combinar las diferentes componentes en frecuencia se conocen como filtros de síntesis.

3.1. Filtros espejo en cuadratura (QMF)

Para que tras los diferentes procesos de filtrado que se realizan en la codificación y descodificación en subbandas de una señal no se distorsione la misma, los bancos de filtros usados deben cumplir algunas condiciones, conocidas como condiciones de *reconstrucción perfecta*. Los filtros espejo en cuadratura (QMF) son un tipo de filtros que cumplen estas condiciones.

En un sistema de codificación en subbandas con 2 canales, como el mostrado en la Figura 1, se necesitan 4 filtros: 2 de análisis (un filtro paso-baja, h_1, y un filtro paso-alta, h_2) y 2 de síntesis (un filtro paso-baja, k_1, y un filtro paso-alta, k_2). Estos filtros cumplirán las condiciones de reconstrucción perfecta si h_1 tiene fase lineal (coeficientes «simétricos») y el resto de filtros satisfacen:

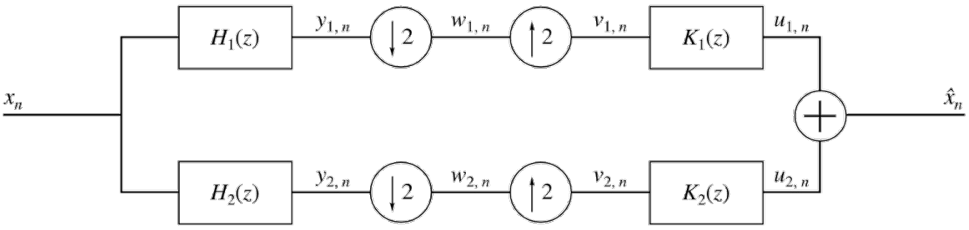

Figura 1: Esquema de un sistema de codificación en subbandas con 2 canales.

$$H_2(z) = H_1(-z) \tag{1}$$

$$K_1(z) = 2 \cdot H_2(-z) = 2 \cdot H_1(z) \tag{2}$$

$$K_2(z) = -2 \cdot H_1(-z) = -2 \cdot H_2(z) \tag{3}$$

donde $H_2(z)$, $K_1(z)$ y $K_2(z)$ son las transformadas Z de h_2, k_1 y k_2. En el dominio del tiempo, las ecuaciones 1, 2, 3 se convierten en:

$$h_2(n) = (-1)^n h_1(n) \tag{4}$$

$$k_1(n) = 2 \cdot h_1(n) \tag{5}$$

$$k_2(n) = -2 \cdot (-1)^n h_1(n) = -2 \cdot h_2(n) \tag{6}$$

donde $h_1(n)$, $h_2(n)$, $k_1(n)$ y $k_2(n)$ representan la respuesta al impulso de los 4 filtros del sistema.

Por tanto, una vez calculados los coeficientes de la respuesta al impulso de h_1, de forma tal que sea un filtro de fase lineal, es sencillo obtener el resto de filtros de un sistema con 2 canales. En Python, podemos usar la función `firwin` del módulo `signal` de `scipy` para construir un filtro FIR con fase lineal mediante el método de enventanado:

```
from scipy import signal
from scalib import bode

h = signal.firwin(24, 0.5, window='hamming')
bode(h)
```

El ejemplo anterior almacena en `h` un array con los coeficientes de la respuesta al impulso de un filtro paso-baja de 24 coeficientes con frecuencia de corte $\pi/2$ rad/muestra. La respuesta en frecuencia del filtro (obtenida con la función `bode`, incluida en `scalib`) se muestra en la Figura 2.

Ejercicio 2

Implemente una función que calcule los coeficientes de la respuesta al impulso de los 4 filtros QMF necesarios en un sistema de codificación en subbandas de 2 canales. Use la función `firwin` de `scipy.signal` y las ecuaciones 4, 5 y 6. A continuación obtenga la respuesta en frecuencia de esos filtros e interprete el resultado.

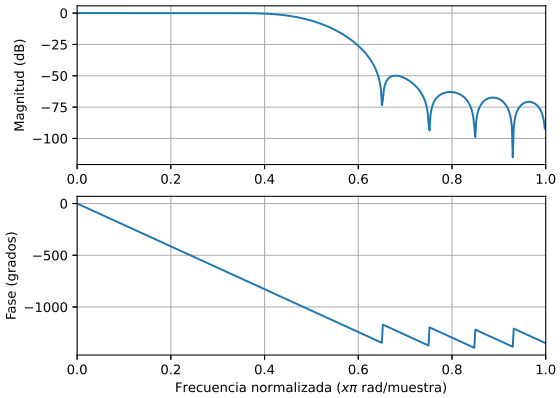

Figura 2: Respuesta en frecuencia del filtro FIR paso-baja con frecuencia de corte $\pi/2$ rad/muestra.

3.2. Estándar G.722: Codificación de audio en subbandas

En un sistema de codificación en subbandas de 2 canales, la señal recuperada (tras el proceso de codificación y descodificación) ha atravesado 4 filtros. Para comprobar si la magnitud o la fase de la señal se ven alteradas tras este proceso, podemos calcular la respuesta al impulso del sistema completo, $g(n)$, y a partir de ella, calcular y representar su respuesta en frecuencia.

$$g(n) = \frac{h_1(n) * k_1(n) + h_2(n) * k_2(n)}{2} \tag{7}$$

donde el operador $*$ representa la operación de convolución.

Algoritmo 3: Codificador en subbandas de 2 canales

Input: Vector con las muestras de la señal, s
Tamaño de la ventana, w_{len}
Tasa de bits para el canal de frecuencias bajas, b_l
Tasa de bits para el canal de frecuencias altas, b_h
Tasa de bits para las ganancias, b_g

Output: Vector con la señal codificada, c

$h_1 \leftarrow getG722lpf()$ // Recupera los coeficientes almacenados del filtro paso-baja
$h_2 \leftarrow (-1)^n h_1(n)$ // Calcula el filtro paso-alta como el espejo del filtro paso-baja
$s_l \leftarrow h_1 * s$ // Filtra s con el filtro h_1 para obtener el canal de frecuencias bajas
$s_h \leftarrow h_2 * s$ // Filtra s con el filtro h_2 para obtener el canal de frecuencias altas
$s_l \leftarrow \{s_{l_0}, s_{l_2}, s_{l_4}, ...\}$ // Reduce a la mitad la frecuencia de muestreo del canal de frecuencias bajas
$s_h \leftarrow \{s_{h_0}, s_{h_2}, s_{h_4}, ...\}$ // Reduce a la mitad la frecuencia de muestreo del canal de frecuencias altas
$c_l \leftarrow APCM_{encode}(s_l, w_{len}, b_l, b_g)$ // Codifica el canal de frecuencias bajas con APCM
$c_h \leftarrow APCM_{encode}(s_h, w_{len}, b_h, b_g)$ // Codifica el canal de frecuencias altas con APCM
$c \leftarrow c_l \cup c_h$ // Concatena el código de los dos canales

Ejercicio 3

Represente la respuesta en frecuencia del sistema completo formado por los filtros obtenidos en el Ejercicio 2 e interprete el resultado. Para realizar la operación de convolución, use la función `convolve` de `numpy`. Calcule también el retardo introducido por el sistema[1].

Repita el calculo de los filtros y de la respuesta en frecuencia del sistema completo usando, para h_1, una frecuencia de corte entre un 5 % y un 10 % superior. ¿Cómo cambia la respuesta en frecuencia del sistema completo? Justifique el resultado.

Por último, repita los cálculos anteriores usando para h_1 los coeficientes de la respuesta al impulso definidos en el estándar G.722, que puede obtener mediante la función `getG722lpf` de `scalib`. ¿Qué diferencias encuentra con los casos anteriores?

Para que la codificación en subbandas pueda producir tasas de bits reducidas es necesario reducir la frecuencia de muestreo de los canales en los que el codificador divide la señal. Esta reducción de la frecuencia de muestreo puede llevarse a cabo sin perdida de información si se hace por un factor inferior o igual a la reducción del ancho de banda de cada canal respecto del de la señal completa. Además, la disminución de la frecuencia de muestreo debe deshacerse (aumentando dicha frecuencia) durante el proceso de descodificación. Los Algoritmos 3 y 4 muestran el proceso completo llevado a cabo, respectivamente, por un codificador y un descodificador en subbandas de 2 canales, similares a los definidos en el estándar G.722. El descodificador debe determinar cuantos bits de la cadena de bits de entrada correponden a cada canal. Conociendo el tamaño de la ventana y las tasas de bits empleadas el cálculo es sencillo y puede realizarse usando la siguiente función:

```
def getBitsPerChannel(nBits, wlen, bLB, bHB, bG):
    bGLB = bG if bLB > 0 else 0
    bGHB = bG if bHB > 0 else 0

    wSize = wlen*(bLB + bHB) + bGLB + bGHB
    nSamples = (nBits // wSize * wlen + max(nBits % wSize - bGLB - bGHB, 0) // (bLB+bHB))
    nLB = int(nSamples * bLB + np.ceil(nSamples / wlen) * bGLB)
    nHB = nBits - nLB

    return nLB, nHB
```

[1] Puede usar la función `filterDelay` incluida en `scalib`.

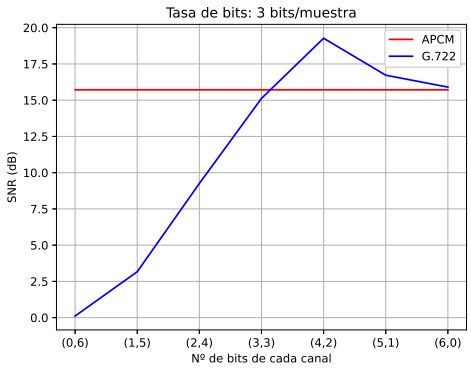

Figura 3: Comparación entre la SNR obtenida por un codificador APCM y uno similar al estándar G.722.

Algoritmo 4: Descodificador en subbandas de 2 canales

Input: Vector con la señal codificada, c

Tamaño de la ventana, w_{len}

Tasa de bits para el canal de frecuencias bajas, b_l

Tasa de bits para el canal de frecuencias altas, b_h

Tasa de bits para las ganancias, b_g

Output: Vector con las muestras de la señal, s

$k_1 \leftarrow 2 \cdot getG722lpf()$ // Construye el filtro paso-baja de síntesis a partir del filtro paso-baja de análisis

$k_2 \leftarrow (-1)^{n+1} k_1(n)$ // Calcula el filtro paso-alta como el espejo del filtro paso-baja

$n_{LB}, n_{HB} \leftarrow getBitsPerChannel(|c|, w_{len}, b_l, b_h, b_g)$ // Núm. de bits correspondiente a cada canal

$c_l \leftarrow \{c_0, c_1, ..., c_{n_{LB}-1}\}$ // Bits correspondiente al canal de frec. bajas

$c_h \leftarrow \{c_{n_{LB}}, c_{n_{LB}+1}, ..., c_{|c|-1}\}$ // Bits correspondiente al canal de frec. altas

$s_l \leftarrow APCM_{decode}(c_l, w_{len}, b_l, b_g)$ // Descodifica el canal de frecuencias bajas con APCM

$s_h \leftarrow APCM_{decode}(c_h, w_{len}, b_h, b_g)$ // Descodifica el canal de frecuencias altas con APCM

$s_l \leftarrow \{s_{l_0}, 0, s_{l_1}, 0, s_{l_2}, 0, ...\}$ // Duplica la frecuencia de muestreo del canal de frecuencias bajas

$s_h \leftarrow \{s_{h_0}, 0, s_{h_1}, 0, s_{h_2}, 0, ...\}$ // Duplica la frecuencia de muestreo del canal de frecuencias altas

$s_l = k_1 * s_l$ // Filtra el canal de frec. bajas con el filtro k_1

$s_h = k_2 * s_h$ // Filtra el canal de frec. altas con el filtro k_2

$s \leftarrow s_l + s_h$ // Combina los dos canales para obtener la señal completa

Ejercicio 4

Siguiendo el pseudocódigo de los Algoritmos 3 y 4, implemente un codificador y un descodificador en subbandas de 2 canales. Para ello, cree dos funciones con la siguiente interfaz:

```
def encoderG722(data, wlen, dataRange, bl, bh, bg):
...
code = ...
return code
```

```
def decoderG722(code, wlen, dataRange, bl, bh, bg):
...
data = ...
return data
```

donde:

- `data` → Array unidimensional que contiene la señal de audio que vamos a codificar.
- `code` → Array unidimensional con la señal codificada.
- `wlen` → Tamaño de la ventana.
- `dataRange` → Tupla de 2 elementos con el mínimo y el máximo valor posible de las muestras.
- `bl`, `bh` y `bg` → Tasa de bits/muestra del canal de frecuencias bajas, del canal de frecuencias altas y de las ganancias.

Ejercicio 5

Use las funciones implementadas en el Ejercicio 4 para codificar y descodificar la señal de audio almacenada en `marlene.wav` con una ventana de 120 muestras, una tasa de 16 bit/muestra para las ganancias y todas las combinaciones posibles de una tasa de 3 bit/muestra (6 bits para el canal 1 y 0 bits para el canal 2; 5 bits para el canal 1 y 1 bits para el canal 2; etc.) para los datos. En cada caso, calcule la SNR resultante y compárela con la obtenida por un codificador PCM adaptable como el implementado en el Ejercicio 1. Tenga en cuenta el retardo introducido por el sistema. Represente las SNR obtenidas en un gráfico similar al de la Figura 3. Analice los resultados.