Universidad de la República

Facultad de Ingeniería

Procesamiento Digital de Señales de Audio

Práctica 1

Autor:
Julian O'Flaherty

4 de abril de 2022

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Índice

1. Ejercicio 1

1.1. Parte 1: Muestreo y Aliasing

El aliasing es una distorsión de la señal resultado de muestrear a una frecuencia menor a que la frecuencia de Nyquist:

$$f_s >= f_{Nyquist} = 2W_x \tag{1}$$

donde W_x es el ancho de banda de la señal que se quiere muestrear.

Para sub-muestrear la señal, muestreada a $44100Hz$, se utilizaron 2 filtros decimadores. El primero simplemente tomaba una de cada L muestras, el segundo agregaba un filtro antialias antes del sub-muestreo, evitando así el alias. En la figura 1 se muestran los espectogramas resultantes para dos tonos puros de distinta frecuencia luego del remuestreo con ambos filtros, tomando cada 2 y cada 4 muestras.

Tomando cada 2 muestras, una frecuencia de muestreo de $22050Hz$, la señal no es distorsionada. Sin embargo, cuando tomamos 1 de cada 4 muestras, una frecuencia de muestreo de $11025Hz$ se observa el efecto del aliasing. En el primer filtro (imagen abajo a la izquierda) se observa que el tono cayo en frecuencia, pasando de un valor inicial de $7040HZ$ a estar alrededor de los $4000Hz$, resultando en una modificación de lo escuchado. El filtro que implementa un antialias al principio resulta elimina este tono antes del sub-muestreo, por lo que no se lo observa en el resultado.

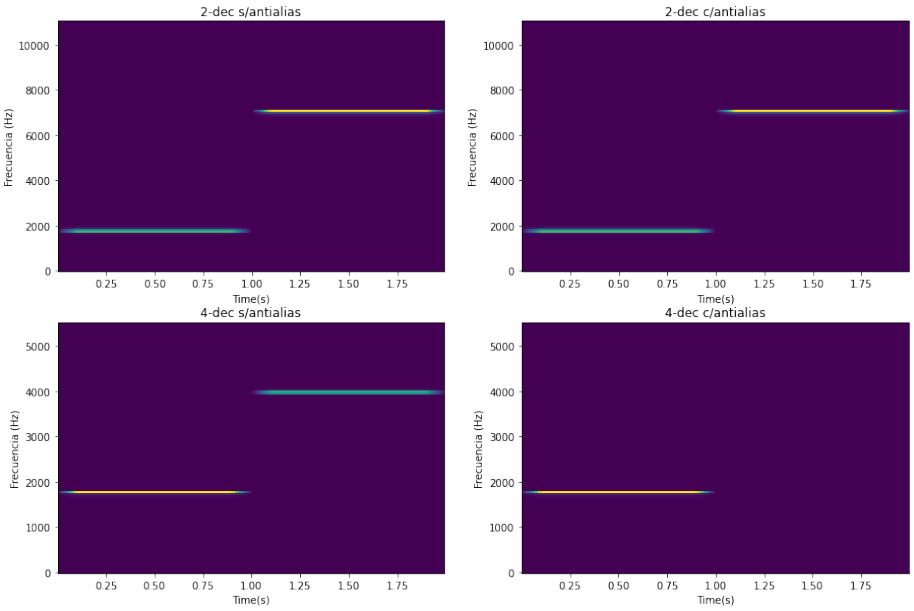
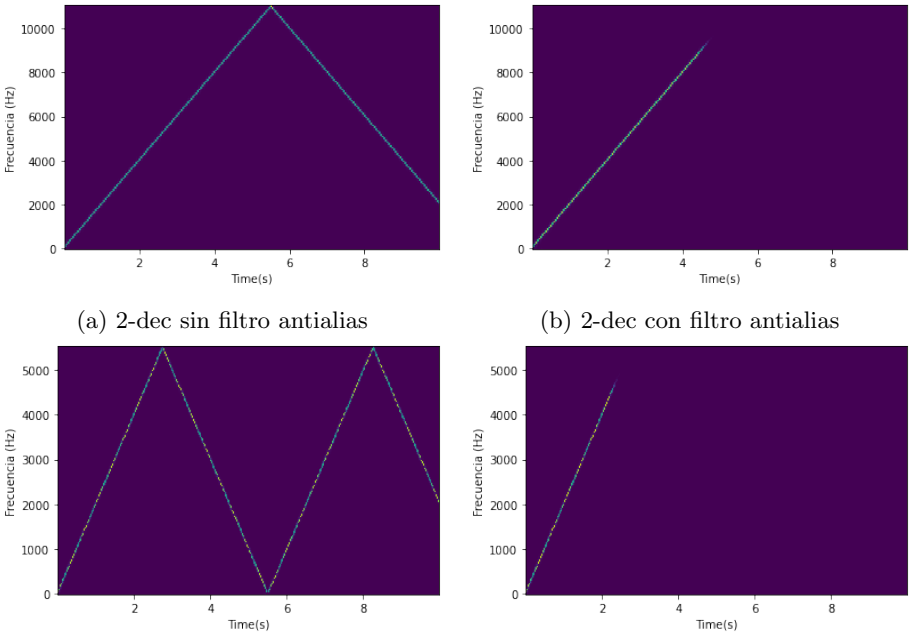


Figura 1: espectogramas resultantes del filtrado de dos tonos puros

Se repitió el experimento utilizando el audio *chirp.wav*, una señal que aumenta de frecuencia linealmente con el tiempo.

Luego del sub-muestreo se obtienen los espectrogramas de la figura 2. Dividiendo la frecuencia de muestreo entre 2, a partir de la ecuación 1 se espera que cuando la señal supere los $f_s^{nueva} = \frac{f_s}{2} = 11025Hz$ se comience a afectar el aliasing. En la figura 2a se observa cómo sin el filtro antialias las altas señales por arriba de la cota establecida se empiezan a ubicarse en $f_{alias} = f_s^{nueva} - \text{mín}_{k\in\mathbb{N}}\{|kf_s - f|\}$, formándose el triangulo que se ve. Cuando se agrega el filtro antialias, se suprime la señal a partir de la frecuencia donde se comenzaría a obvservar el aliasing.

Cuando se divide la frecuencia de muestreo entre 4, se observa el mismo efecto que antes, solo que ahora a partir de $f_s^{nueva} = \frac{f_s}{4} = 5512{,}5Hz$.



(a) 2-dec sin filtro antialias

(b) 2-dec con filtro antialias

(c) 4-dec sin filtro antialias

(d) 4-dec con filtro antialias

Figura 2: Resultados del sub-muestreo

1.2. Parte 2: cuantización y dithering

1.2.1. Signal to error ratio

Podemos escribir el resultado de la cuantización como:

$$y[n] = x[n] + e[n] \tag{2}$$

Se asume un error de cuantización uniforme, es decir, $e[n] \sim \mathcal{U}(\frac{-Q}{2}, \frac{Q}{2})$, donde Q es el ancho del de los bins de cuantización. Se asume también que la señal varía lo suficientemente rápido para que el ruido y la señal sean independientes, por lo que podemos calcular sus potencias por separado.

Tomemos la potencia de la señal como P_x. La potencia del ruido del ruido se puede calcular de la siguiente manera:

$$P_e = E\left[e^2\right] = \int_{-\infty}^{\infty} e^2 \rho(e) de = \int_{-\frac{Q}{2}}^{\frac{Q}{2}} e^2 \frac{1}{Q} de = \left.\frac{1}{Q}\frac{e^3}{3}\right|_{-\frac{Q}{2}}^{\frac{Q}{2}} = \frac{Q^2}{12}$$

Es trivial ver que $Q = \frac{2X_{max}}{2^b}$, donde X_{max} es el máximo valor que queremos que nuestro cuantizador puede procesar y b es el número de bits. Por lo tanto:

$$P_e = \frac{X_{max}^2}{12 \times 2^{2b-2}} = \frac{X_{max}^2}{3 \times 2^{2b}} \tag{3}$$

Por lo tanto, la SNR de la señal es:

$$\begin{gathered} SNR = \frac{P_x}{P_e} = \frac{3P_x 2^{2b}}{X_{max}} \\ SNR_{dB} = 10\log\left(3P_x\right) - 10\log(X_{max}) + 20b\log(2) \end{gathered} \tag{4}$$

Fácilmente se observa que aumentar un bit mejora en $20\log(2)$ la SNR de nuestra cuantización.

1.2.2. Dithering

El dithering es una técnica para mejorar la SNR de una cuantización agregando un ruido independiente a la señal antes de cuantizarla:

$$y[n] = x[n] + w[n] + e[n]$$

En este caso, la SNR la podemos calcular como:

$$SNR = \frac{P_x}{P_e + P_w}$$

Calculemos la potencia del ruido para distintas distribuciones del ruido:

Triangular:

$$\begin{aligned} P_w^{triangular} = E[e^2] &= \int_{-\infty}^{\infty} e^2 \rho(e) = \int_{-Q}^{0} e^2 \frac{e+Q}{Q^2} + \int_{0}^{Q} e^2 \left(\frac{-e+Q}{Q^2}\right) \\ &= \left(\frac{e^4}{4Q^2} + \frac{e^3}{3Q}\right)\Big|_{-Q}^{0} + \left(-\frac{e^4}{4Q^2} + \frac{e^3}{3Q}\right)\Big|_{0}^{Q} = \frac{2Q^2}{3} - \frac{Q^2}{2} \end{aligned}$$

$$P_e^{triangular} = \frac{Q^2}{6}$$

Por lo tanto, la potencia de ruido de cuantización y el SNR para un dithering triangular son:

$$P_e = \frac{Q^2}{6} + \frac{Q^2}{12} = \frac{Q^2}{4} \qquad SNR = \frac{4P_x}{Q^2} \tag{5}$$

Cuadrada:

$$P_e^{cuadrada} = E[e^2] = \int_{-\infty}^{\infty} e^2 \rho(e) = \int_{-\frac{Q}{2}}^{\frac{Q}{2}} e^2 \frac{1}{Q} = \frac{e^3}{3Q}\Big|_{-\frac{Q}{2}}^{\frac{Q}{2}} = \frac{Q^2}{12}$$

Por lo tanto, la potencia de ruido de cuantización y el SNR para un dithering cuadrado son:

$$P_e = \frac{Q^2}{12} + \frac{Q^2}{12} = \frac{Q^2}{6} \qquad SNR = \frac{6P_x}{Q^2} \tag{6}$$

Gaussiana:

$$P_e^{cuadrada} = E[x^2] = \int_{-\infty}^{\infty} x^2 \rho_e(x) dx = \int_{-\infty}^{\infty} x^2 \frac{2}{\sqrt{2\pi}Q} e^{-\frac{4x^2}{Q^2}}$$

Esta integral no tiene primitiva definida. Para resolverla se la introdujo en una https://www.integral-calculator.com/. El resultado es:

$$P_e^{cuadrada} = \frac{Q^2}{4}$$

Por lo tanto, la potencia de ruido de cuantización y el SNR para un dithering gaussiano son:

$$P_e = \frac{Q^2}{12} + \frac{Q^2}{4} = \frac{Q^2}{3} \qquad SNR = \frac{3P_x}{Q^2} \tag{7}$$

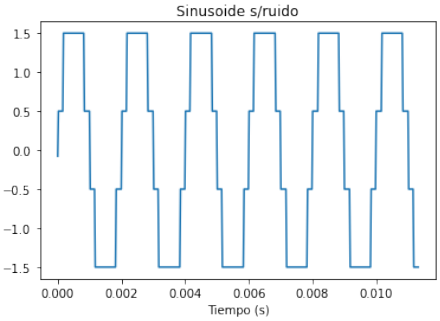
1.2.3. Implementación

Para estudiar el efecto del dithering se cuantiza en 4 nivelves una sinusoide pura de amplitud 2 y frecuencia $500Hz$. En la figura 3 se muestra el resultado de la cuantización con distintos ditherings.

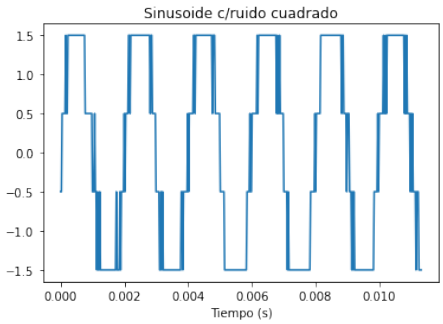
Auditivamente, cuantizar la señal sin dithering resulta en un sonido poco armónico, del estilo que uno normalmente asocia con los juegos clásico de 8-bits. Al agregar el dihering, el sonido parece más armónico, asemejándose más al original, pero contiene una componente importante de ruido blanco, por lo que la SNR decae. Empíricamente se ve claramente en los resultados mostrado en la tabla 1.

Ruido	Potencia	SNR
sin	-22.56dB	19.09
cuadrado	-19.01dB	13.38
triangular	-16.13dB	10.03
gaussiano	-13.70dB	7.87

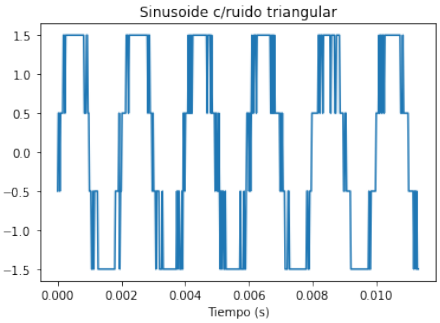
Cuadro 1: potencias del ruido de cuantización con el dithering



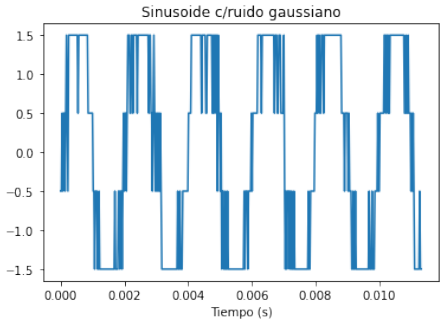
(a) sinusoide sin ruido



(b) sinusoide con ruido con distribución cuadrada



(c) sinusoide con ruido con distribución triangular



(d) sinusoide con ruido con distribución gaussiana

Figura 3: resultados del dithering

2. Ejercicio 2: características de tiempo corto

2.1. Parte 1

Se definen las magnitudes E_n y M_n de la señal $x[n]$ como:

$$E_n = \sum_{m=-\infty}^{\infty} x^2[m]w[n-m] \quad M_n = \sum_{m=-\infty}^{\infty} |x[m]|w[n-m]$$

Donde $w[n]$ es una ventana de análisis temporal.

2.1.1. Ventana w_a

Tomemos la ventana:

$$w_a[n] = \begin{cases} a^n, & n \geq 0 \\ 0, & n < 0. \end{cases}$$

Por lo tanto:

$$E_n = \sum_{m=-\infty}^{\infty} x^2[m]w_a[n-m] = \sum_{m=-\infty}^{n} a^{n-m}x^2[m] = a\sum_{m=-\infty}^{n} a^{n-1-m}x^2[m]+x^2[n]$$

$$E_n = aE_{n-1} + x^2[n] \tag{8}$$

$$M_n = \sum_{m=-\infty}^{\infty} |x[m]|w_a[n-m] = \sum_{m=-\infty}^{n} a^{n-m}|x[m]| = a\sum_{m=-\infty}^{n} a^{n-1-m}|x[m]|+|x[n]|$$

$$M_n = aM_{n-1} + |x[n]| \tag{9}$$

2.1.2. Ventana w_R

Tomemos la ventana:

$$w_R[n] = \begin{cases} 1, & 0 \leq n \leq N-1 \\ 0, & \text{en otro caso.} \end{cases}$$

Hallemos las expresiones de recurrencia:

$$E_n = \sum_{m=-\infty}^{\infty} x^2[m]w_R[n-m] = \sum_{m=n-N+1}^{n} x^2[m] = x^2[n]+\sum_{m=n-N}^{n-1} x^2[m]-x^2[n-N]$$

$$E_n = E_{n-1} + x^2[n] - x^2[n-N] \tag{10}$$

$$M_n = \sum_{m=-\infty}^{\infty} |x[m]|w_R[n-m] = \sum_{m=n-N+1}^{n} |x[m]| = |x[n]|+\sum_{m=n-N}^{n-1} |x[m]|-|x[n-N]|$$

$$M_n = M_{n-1} + |x[n]| - |x[n-N]| \tag{11}$$

Implementando las recursiones en python (anexo 4.1), calculamos la energía y la magnitud de la señal en el tiempo. El audio cuenta hasta 10, dejando un tiempo entre número y número. Viendo los resultados en la figura 4 se pueden identificar claramente los primeros 6 números. Al ser todos monosilábicos, solo se observa un pico por número, siendo el seis ('six') el de menor amplitud y duración, coherente con la fonética. El siete ('seven') es el primer número con más de una silaba, por lo que comenzamos a observar más picos por número. Esto complica la identificación, pudiendo hacerlo si se sabiendo que están más o menos equiespaciados en el tiempo.

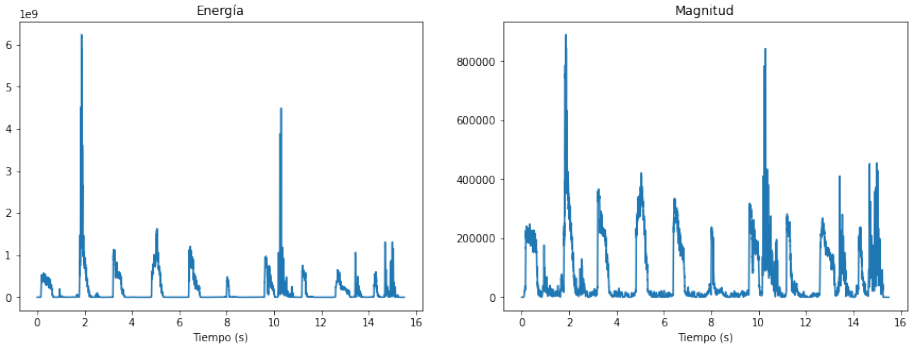


Figura 4: energía y magnitud de la señal voice.wav con ventana rectangular de 10 ms

2.2. Parte 2: tasa de cruces por cero

Matemáticamente, la tasa de cruces por cero se define como:

$$Z_m = \sum_{m=-\infty}^{\infty} |\text{sign}(x[m]) - \text{sign}(x[m-1])| w[n-m]$$

donde:

$$\text{sign}(x[m]) = \begin{cases} 1, & x[m] \geq 0 \\ -1, & x[m] < 0. \end{cases} \quad w[n] = \begin{cases} \frac{1}{2N}, & 0 \leq n \leq N-1 \\ 0, & \text{en otro caso} \end{cases}$$

2.2.1. Recursión

$$\begin{aligned} Z_n &= \sum_{m=-\infty}^{\infty} |\text{sign}(x[m]) - \text{sign}(x[m-1])| w[n-m] \\ &= \frac{1}{2N} \sum_{m=n-N+1}^{n} |\text{sign}(x[m]) - \text{sign}(x[m-1])| \end{aligned}$$

$$\begin{aligned} Z_n = Z_{n-1} + \frac{1}{2N} \{ &|\text{sign}(x[n]) - \text{sign}(x[n-1])| \\ &- |\text{sign}(x[n-N]) - \text{sign}(x[n-N-1])| \} \end{aligned} \tag{12}$$

2.2.2. Implementación

Utilizando la recursión es posible crear un programa que calcula eficientemente la tasa de cruces por zero. El código de la función se adjunto en el anexo 4.2. El resultado de aplicarla a la señal se observa en la figura 5.

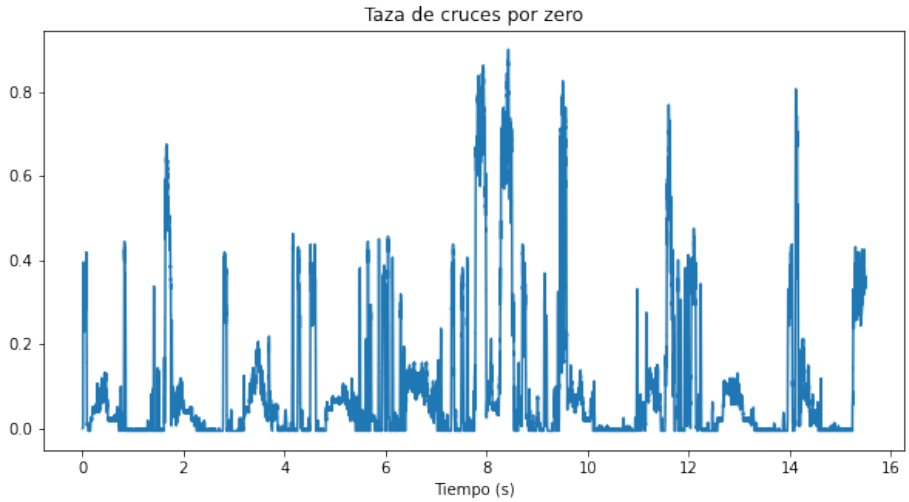


Figura 5: taza de cruces por cero de la señal *voice.wav* con ventana de 10ms

Comparando con la figura 4, se observa que los picos de los cruces por cero se dan cuando se está en silencio o en camino a una consonante como primer letra ('two' por ejemplo).

2.3. Parte 3: detección de palabras

Utilizando la energía y la tasa de cruces por cero se puede intentar detectar la ubicación de las palabras en la señal. El código implementado se encuentra en el Anexo 4.3. Los parámetros del código se calibraron utilizando el audio *voice.wav*, audio sobre el cual se calcularon las características.

Para detectar las palabras se jugo con los umbrales de energía y tasa de zeros. En el caso de la energía, de tomaron dos umbrales para que tuviera histeresis: una vez superado el umbral inicial para considerarlo palabra, el umbral mínimo para empezar a considerar que terminó la palabra es varias decadas menor. En simultáneo se evalúa que la tasa de cruces por cero sea menor que otro umbral, seleccionado en 0,3 (no depende de la señal), acorde con lo observado anteriormente.

Una vez detectada la palabra, para detectar si se mantiene en al misma palabra se pide o que la energía sea mayor que el segundo umbral, o que el zero-crossing-rate sea menor que 0,2.

El segundo audio, cuyo análisis se observa en la figura 7, es más complejo fonéticamente. Esto resulta en nuestro clasificador simple fallando. Por ejemplo, ün águila"se la escucha como una sola palabra sin separación (ünaguila"). La conexión de palabras vecinas sin pausa, posible por las características fonéticas de cada una, hace que la tarea sea mucho más difícil y requiera un procesamiento más fino que lo que se hará en este trabajo.

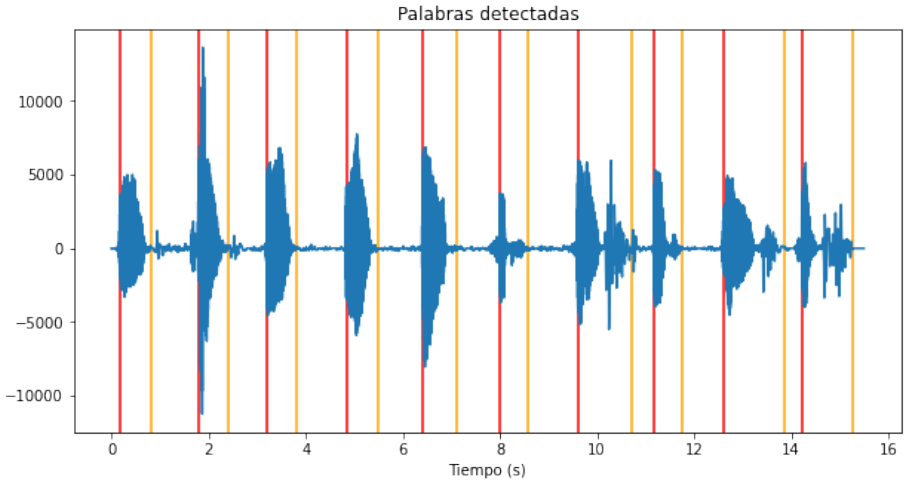


Figura 6: señal temporal del audio marcando el comienzo (rojo) y el final (naranja) de las palabras

Cuando se aplico sobre el audio con dos tonos puros, no se detectaron palabras, lo cual es coherente con la tarea.

3. Ejercicio 3

3.1. Parte 1

Se define la autocorrelación de tiempo corto como:

$$R_n[k] = \sum_{m=-\infty}^{\infty} x[m]w[n-m]x[m+k]w[n-k-m] \tag{13}$$

3.1.1. función par

$$\begin{aligned} R_n[-k] &= \sum_{m=-\infty}^{\infty} x[m]w[n-m]x[m-k]w[n+k-m] \\ &\overset{l=m-k}{=} \sum_{l=-\infty}^{\infty} x[l+k]w[n-l-k]x[l]w[n-l] = R_n[k] \end{aligned}$$

Por lo tanto:

$$R_n[k] = R_n[-k] \tag{14}$$

la función de autocorrelación de tiempo corto es par.

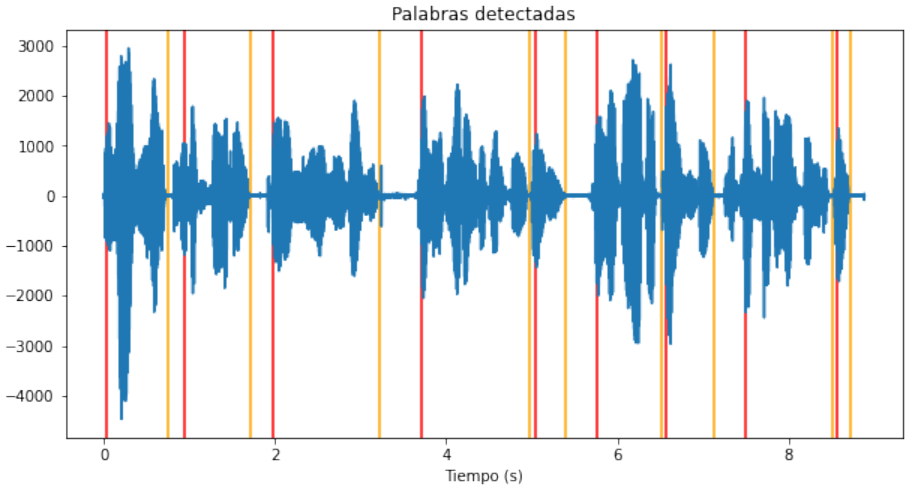


Figura 7: detección de palabras sobre en audio *voice2.wav*

3.1.2. forma compacta

Definamos $h_k[n] = w[n]w[n+k]$. Observar que:

$$\begin{aligned} R_n[k] &= \sum_{m=-\infty}^{\infty} x[m]w[n-m]x[m+k]w[n-k-m] \\ &\overset{l=m+k}{=} \sum_{l=-\infty}^{\infty} x[l-k]w[n-l+k]x[l]w[n-l] \\ &= \sum_{l=-\infty}^{\infty} x[l]x[n-l]h_k[n-l] \end{aligned}$$

Por lo que podemos reescribir la ecuación 13 como:

$$R_n[k] = \sum_{m=-\infty}^{\infty} x[m]x[n-m]h_k[n-m] \tag{15}$$

3.2. Parte 2

A partir del procedimiento presentado en la descripción del problema se genero un código en python (anexo 4.4) para estimar la frecuencia fundamental de la señal en el tiempo. Lo primero que hace el código es aplicar un filtro pasa-bajos para atenuar frecuencias por encima de los $1kHz$ para acentuar las frecuencias fundamentales y reducir el efecto de las formantes, y un filtro pasa-altos con frecuencia de corte $10Hz$ para eliminar las componentes constantes de la señal. Ambos filtros son Butterworth de grado 8.

Luego de los filtros se procede con lo sugerido, separando en frames de 30ms cada 10ms y estudiando la correlación en los primero 250 puntos. Se toma que la frecuencia fundamental en un frame dado es el primer máximo local que supera el 60 % de la auto correlación en cero

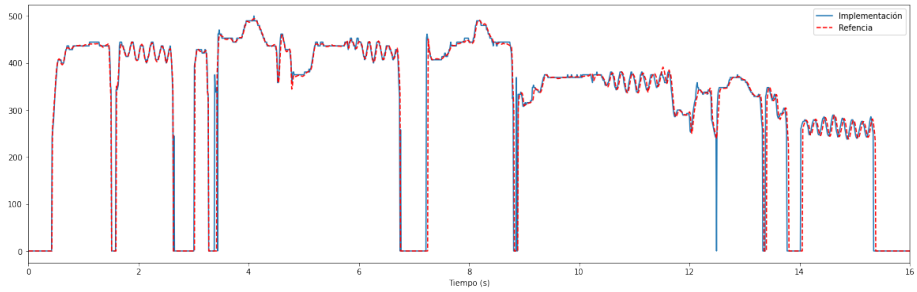
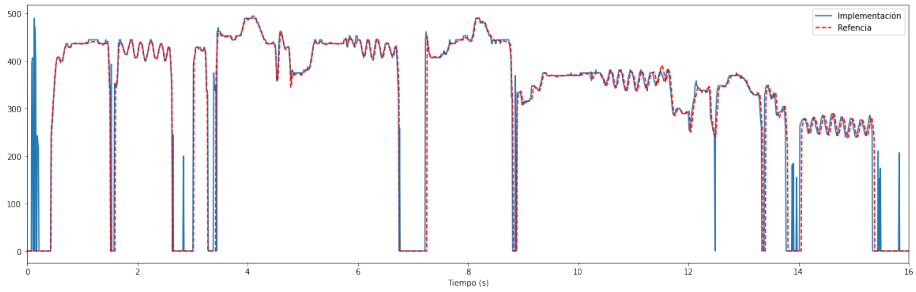


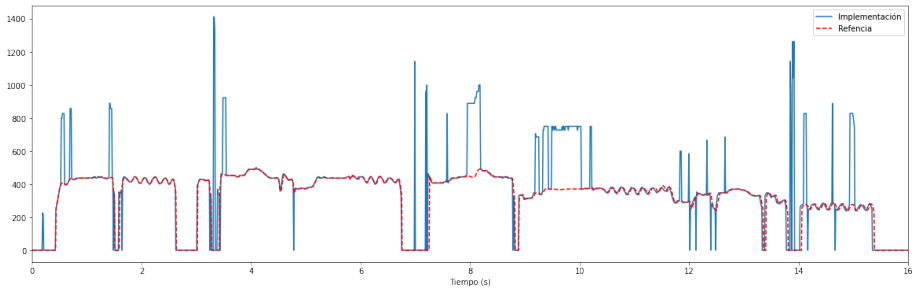
Figura 8: resultado del estudio de las frecuencias fundamentales con la referencia superpuesta

Modificando los parámetros del filtro se puede observar el impacto que estos tienen. En la figura 9 se observa el resultado de cambiar la frecuencia de corte de los filtros. En la figura 9a se eligió una frecruencia muy cercana a cero, y se comienzan a ver ciertas inestabilidades que parecen ser producto del filtro y sus tiempos de respuesta.

En el caso del filtro pasa-altos, al aumentar la frecuencia de corte permitimos que la formantes impacten en la detección de la frecuencia fundamental. Es por esto que vemos los saltos bruscos a frecuencias altas en la figura 9b.



(a) resultado con $0{,}1nHz$ de frecuencia de corte en el HPF



(b) resultado con $2500Hz$ de frecuencia de corte en el LPF

Figura 9: implementación con distintos parametros en los filtros

Anexo

4.1. Implementación calculo de energia y magnitud tiempo corto

```
def energia_ventana_rect(x, fs, ws):
    N = int(fs*ws)
    e = np.empty(x.shape)
    e[:N-1] = 0
    e[N-1] = np.sum(x[:N]**2)
    for i in range(N, len(x)):
        e[i] = e[i-1] + x[i]**2 - x[i-N]**2
    return e

def magnitud_ventana_rect(x, fs, ws):
    N = int(fs*ws)
    M = np.empty(x.shape)
    M[:N-1] = 0
    M[N-1] = np.sum(abs(x[:N]))
    for i in range(N, x.shape[0]):
        M[i] = M[i-1] + abs(x[i]) - abs(x[i-N])
```

```
    return M
```

4.2. Implementación tasa de cruces por cero

```
def signo(a):
    return 1 if a>=0 else -1

def taza_cruces_por_ceros(x, fs, ws):
    N = int(fs * ws)
    z = np.empty(x.shape)
    z[:N+1] = 0
    for i in range(1, N):
        z[N] += abs(signo(x[i]) - signo(x[i-1]))
    z[N] /= (2*N)

    for i in range(N+1, len(x)):
        z[i] = z[i-1] + (1/(2*N)) * (abs(signo(x[i])-signo(x[i-1])) \
                - abs(signo(x[i-N])-signo(x[i-N-1])))

    return z
```

4.3. Código detección de palabras

```
def detect_words(x, fs):
"""
    Devuelve un array con 1s donde comienzan las palabras y 2s cuando terminan
"""
    min_samples_between_words = 5
    threshold_up = 3e8
    threshold_down = 0.4e5

    e = energia_ventana_rect(x, fs, 10e-3)
    s_rate = taza_cruces_por_ceros(x, fs, 10e-3)

    z = np.zeros(e.shape)

    word = False
    down_counter = 0
    for i in range(len(e)):
        if not word and (e[i]>threshold_up and s_rate[i]<0.5):
            z[i] = 1
            word = True
            down_counter = 0
        elif word and e[i] < threshold_down:
            down_counter += 1
            word = down_counter >= min_samples_between_words
            if not word:
                z[i] = 2
        elif word and (e[i] > threshold_down or s_rate[i]<0.2):
            down_counter = 0

    return z
```

4.4. Detección de la frecuencia fundamental

```
def calculo_frecuencia_fundamental(x, fs, ws=30e-3, hs=10e-3, max_k=250):
    # filtrado y preprocesamiento de la señal
    # la frecuencia fundamental de la voz no supera los 1000Hz
    hpf = signal.butter(8, 2500/fs, 'highpass', output='sos')
    x = signal.sosfilt(hpf, x)
    # eliminacion de ruido de continua
    lpf = signal.butter(8, 50/fs, 'lowpass', output='sos')
    x = signal.sosfilt(lpf, x)

    wn = int(fs*ws)
    hn = int(fs*hs)
```

```
n_frames = int((x.shape[0] - wn) / hn)

f = np.zeros(n_frames, dtype='float64')
t = np.linspace(0, n_frames*hs, n_frames) + ws/2

ar = np.empty(max_k)

for i in range(n_frames):
    ind_l = i*hn
    ind_h = i*hn + wn

    for k in range(max_k):
        ar[k] = x[ind_l+k:ind_h] @ x[ind_l:ind_h-k]

    r0 = ar[0]

    j = 1
    while (j<len(ar)): # busca maximo
        if ar[j] > r0*0.7 and ar[j] > ar[j-1] and ar[j] > ar[j+1]:
           break
        j += 1
    if j>1 and j!= 250:
        f[i] = fs/j

return f,t
```