Controlador de filtros de audio digitales en lengua Python

Trabajo final de Procesamiento de Señales -
Maestría en Sistemas Embebidos

Autor
Juan Agustin Bassi

Este trabajo fue realizado como trabajo final de la asignatura Procesamiento de Señales de la Maestría en Sistemas Embebidos de la Universidad de Buenos Aires en abril de 2019.

Índice

Introducción **2**

Filtro Comb **3**

Filtro Flanger **8**

Filtro Wah-Wah **12**

Software controlador de filtros **18**

Características principales del software 19

Repositorio público del software 19

Conclusiones **20**

Próximos pasos **20**

Referencias **21**

Introducción

El propósito del presente trabajo fue analizar el comportamiento de un filtro comb matematicamente, entender su funcionamiento y se determinar cómo se comporta. Para ello se utilizaron los contenidos adquiridos en la asignatura Procesamiento de Señales tales como ecuación de transferencia, transformada Z, análisis de polos y ceros, entre otros.

Luego de analizar en detalle el filtro comb, se analizaron los filtros Flanger y Wah-Wah que tienen aplicación en señales de audio.

Para complementar los análisis de los filtros, se desarrolló un programa en lenguaje Python, que de manera interactiva, recibe comandos provistos por el usuario para reproducir las señales de audio, plotear de manera gráfica el comportamiento de los filtros, cambiar los parámetros, entre otros.

Se decidió realizar esta aplicación ya que por un lado se abarcaron varios de los contenidos teóricos de la materia Procesamiento de Señales y por otro se le dio una representación práctica a la teoría adquirida.

Filtro Comb

En el procesamiento de señales, un filtro comb se produce al sumarle a la señal original una versión retrasada en el tiempo de sí misma, causando así interferencia constructiva y destructiva.

Para este trabajo se presentó un filtro comb del tipo FIR, que se implementó en tiempo discreto. En la figura 1.1 se puede ver la estructura del mismo.

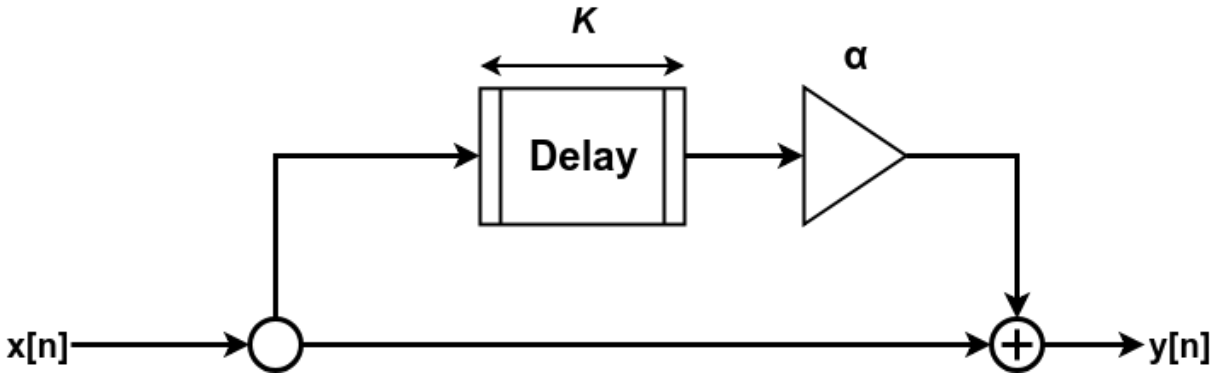

Figura 1.1: Estructura del filtro comb.

La respuesta en frecuencia de un filtro comb consiste en una serie de picos regularmente espaciados, cuya figura se asemeja a la de un peine (comb, en inglés). En la figura 1.2 se puede observar la respuesta en frecuencia genérica de este tipo de filtro.

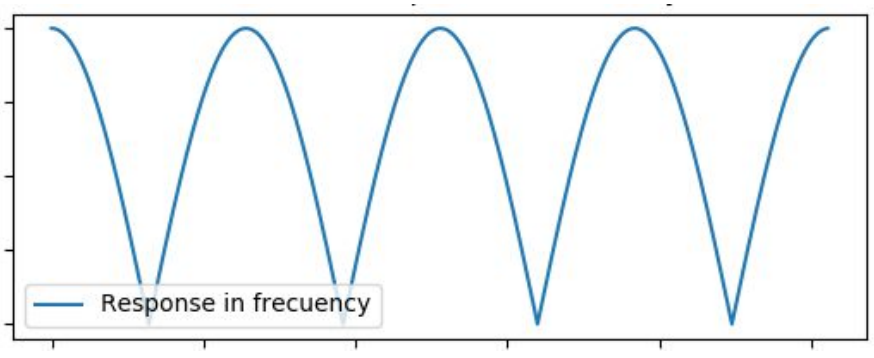

Figura 1.2: Respuesta en frecuencia genérica del filtro comb.

Un detalle interesante que ocurre con este tipo de filtro es que para algunos valores de retraso aplicado a la señal de entrada, ésta puede verse amplificada (interferencia constructiva) o atenuada (interferencia destructiva). La magnitud de la amplificación y atenuación está definida por el factor de escalamiento α.

Para demostrar la respuesta en frecuencia del filtro con distintos valores de factor de escalamiento se presentaron los siguientes casos:

a) Valor de $\alpha = 1$**:** En este caso es donde se despliega el comportamiento más notable del filtro. Para un valor de retraso específico, hay una frecuencia donde éste equivale a un período de la señal de entrada. Por lo tanto, la suma de ambas ramas produce una intereferencia constructiva y se duplica la amplitud de la señal. Para otro valor específico de retraso, la señal de entrada siempre está desfasada 180° de la señal original, de manera que al sumarse ambas señales, se produce interferencia destructiva. En la figura 1.3 se muestra el gráfico de la respuesta con este valor.

b) Valor de $\alpha < 1$**:** En este caso la atenuación y amplificación del filtro se ven suavizadas. La intereferencia sobre la señal original no es total, debido a que ambas señales tienen distitna amplitud. En la figura 1.4 se muestra el gráfico de la respuesta con este valor.

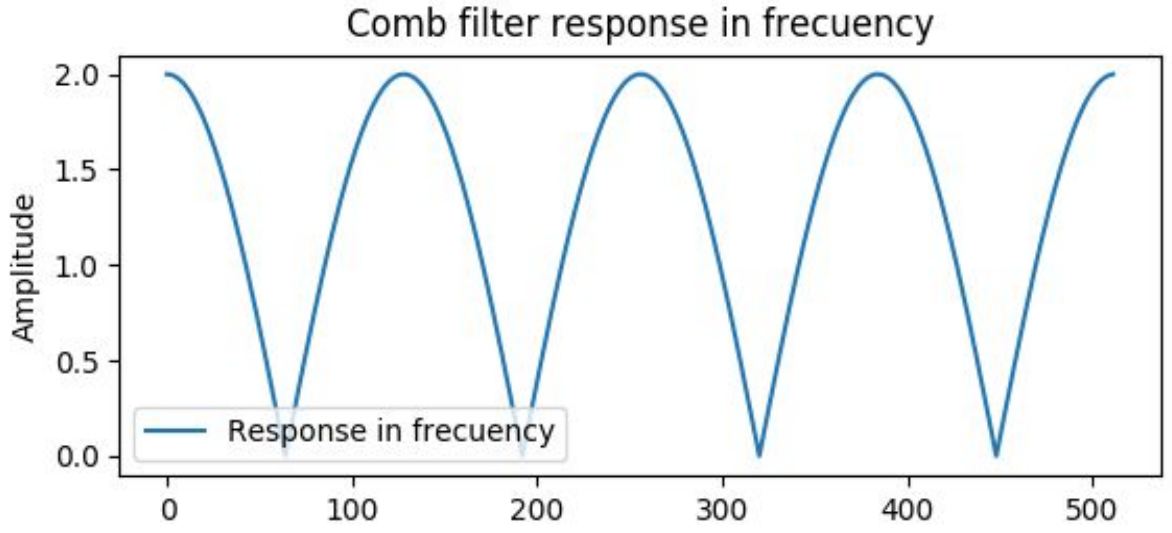

Figura 1.3: Respuesta en frecuencia α = 1.

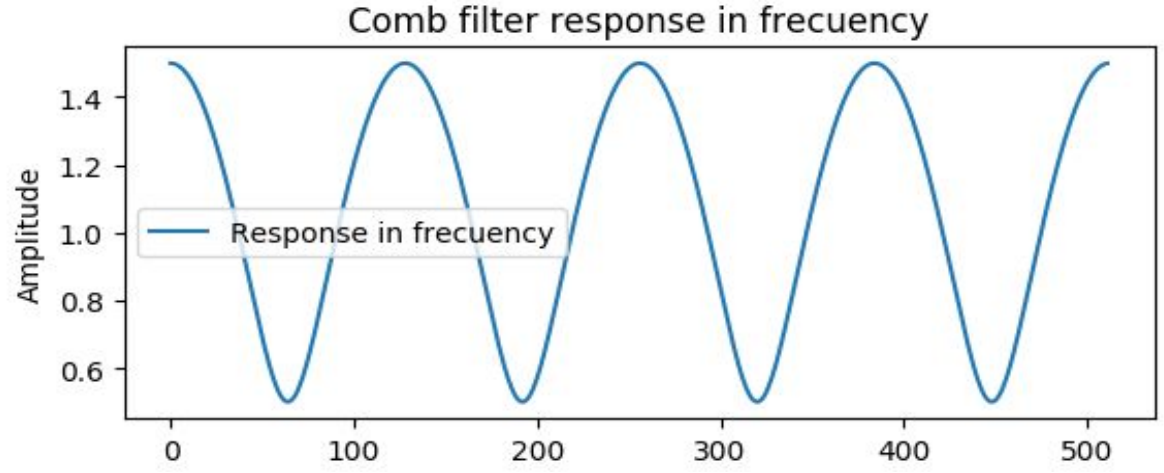

Figura 1.4: Respuesta en frecuencia α < 1.

Se analizaron las características del filtro y se obtuvieron los siguientes detalles:

- La respuesta periódicamente decae hasta un mínimo local (conocido como notch), y luego crece hasta un máximo local (conocido como peak).

- Los niveles máximos y mínimos están siempre equidistantes de 1.

- Cuando **α** es igual a ± 1 , el mínimo tiene amplitud 0. En este caso el mínimo es conocido como cero.

- El máximo de los valores positivos de **α** coincide con el mínimo de los valores negativos de **α**, y viceversa.

La ecuación recurrente de un filtro comb es de la siguiente manera:

$$y[n] = x[n] + \alpha\, x[n-K]$$

Donde *K* es el tamaño del retraso medido en muestras y α es el factor de escalamiento.
La transformada Z de este filtro está representada de la siguiente manera:

$$Y(z) = (1 + \alpha\, z^{-K})\, X(z)$$

A partir de la transformada Z se definió la función transferencia demostrada a continación:

$$H(z) = \frac{Y(z)}{X(z)} = 1 + \alpha z^{-K} = \frac{z^{-K} + \alpha}{z^{K}}$$

Analizando la función transferencia se precisó que el numerador se hace cero cuando $z^{K} = -\alpha$. Estos son los ceros de la función transferencia y se encuentran espaciados equidistes alrededor de un círculo en el plano complejo.

Respecto al denominador, este se convertiera en cero cuando $z^{K} = 0$, resultando en *K* polos cuando $z = 0$. La figura 1.5 demuestra los polos y ceros de un filtro comb con $z = 8$ y $\alpha = 0.5$.

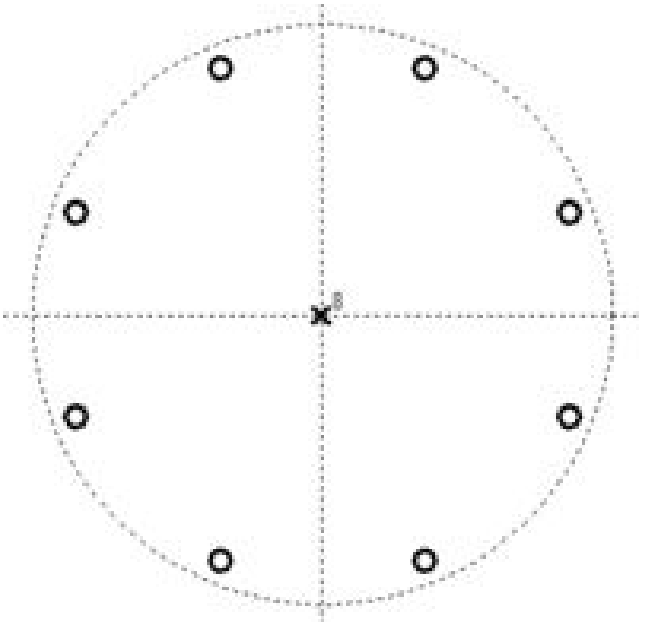

Figura 1.5: Polos y ceros de un filtro comb con $z = 8$ y $\alpha = 0.5$.

Filtro Flanger

Un derivado del filtro comb es el filtro Flanger, que encuentra su aplicación en señales de audio. En la figura 2.1 se muestra una descripción del mismo.

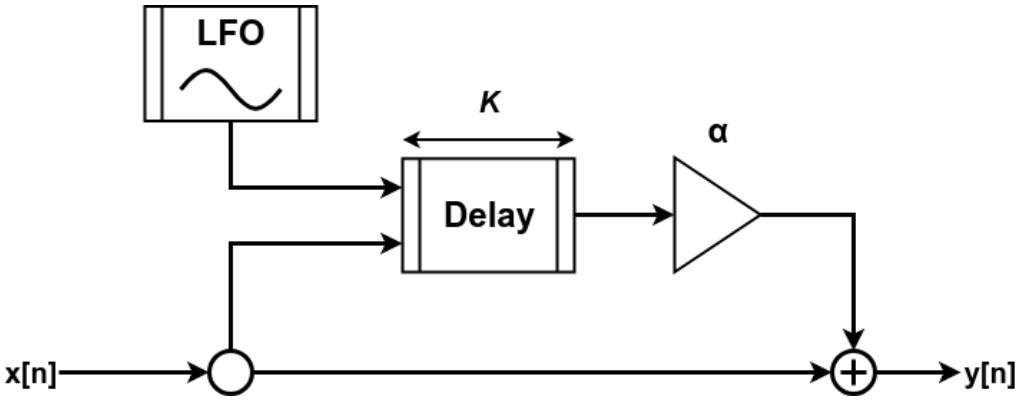

Figura 2.1: Diagrama en bloques de un filtro flanger.

Este filtro actua sobre la señal de entrada aplicando un retraso y luego sumándolo a la salida, pero a diferencia del filtro comb original, el retraso es variable en el tiempo mediante la implementación de un oscilador de frecuencias bajas. El resultado es filtro comb móvil, en el que el peine constantemente se abre y se cierra, ofreciendo una coloración característica como resultado. Esta variación del retraso produce un sonido metalizado oscilante sobre la señal original. Su acción es demostrada en la figura 2.2.

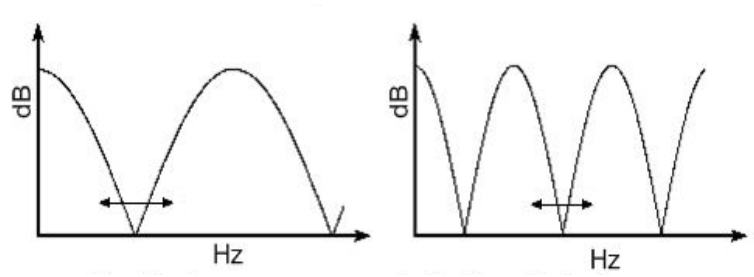

Figura 2.2: Variación del retraso en filtro flanger.

Los valores de retraso más comunes varian entre 1-10 milisegundos. La variacion del valor del retraso se aplica a frecuencias lentas, generalmente de 0 a 3 hz.

A partir de sus características se establecieron los parámetros necesarios para controlar el filtro flanger, entre los que se encuentran:

- **Retraso:** es el valor máximo del retraso aplicado a la señal original, se suele expresar en milisegundos. También llamado profundidad.

- **Frecuencia:** es la frecuencia de oscilación de la modulación del retraso.

- **Factor de escalamiento:** es la proporción de la señal retrasada que se suma a la original.

La implementación del filtro se puede presentar de manera analógica y digital, así como también puede tener realimentación o no. Para este trabajo el análisis se realizó sobre un filtro digital FIR sin realimentacion.

La ecuación recurrente del flanger es similar a la del filtro comb, pero en este caso el valor del retraso es aplicado a la señal original es variable.

$$y[n] = x[n] + \alpha\, x[n - k[n]]$$

Donde a es el factor de escalamiento, y *k[n]* es el valor de retraso aplicado a cada muestra. La determinación de k{n} está dada de la siguiente manera.

$$k[n] = R * sin(2\pi \frac{fc}{fs} n)$$

En donde R es el valor máximo de retardo aplicado, fc es la frecuencia de modulación y fs es la frecuencia de muestreo de la señal de entrada.

El efecto que estos parámetros tienen sobre el filtro, tanto para su respuesta en frecuencia como para el diagrama de polos y ceros, son idénticos a sus los expresados en el filtro comb, mencionados en la sección anterior.

Luego del análisis matemático del filtro se realizó su implementación en lenguaje Python, descripta en el algoritmo 2.1. Para demostrar el comportamiento se plotearon imágenes de funcionamiento y se reprodujeron señales de audio. En la figura 2.3 se muestra la respuesta del filtro (color naranja) sobre la señal original (color azul) que consiste en un tono de 1 khz. Los parámetros aplicados sobre el filtro fueron: $R = 3\,ms, fc = 1\,hz, \alpha = 1$

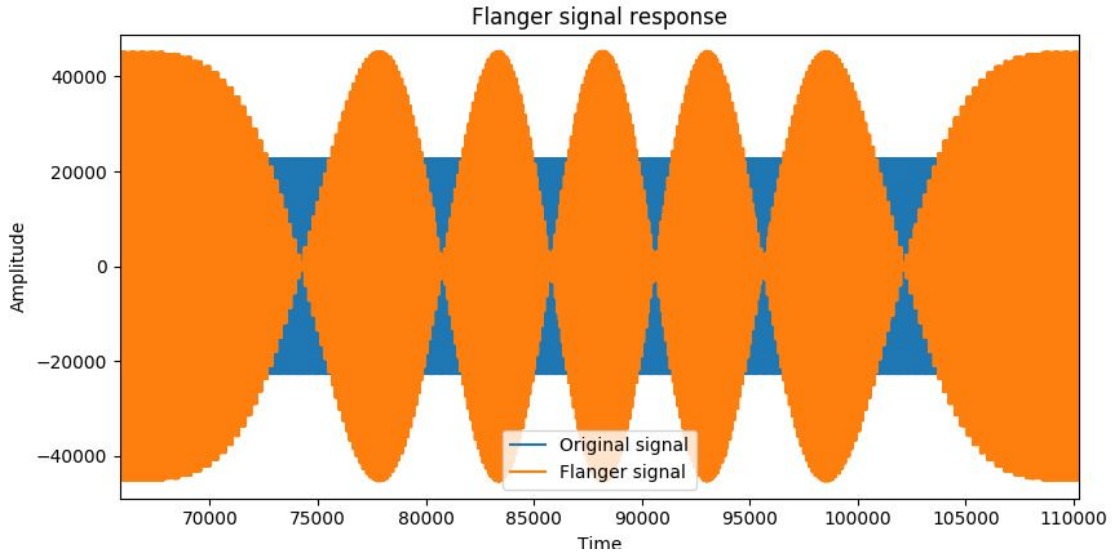

Figura 2.3: Respuesta del filtro flanger sobre señal original 1.

Como se puede apreciar en la figura 2.3, los valores de retraso de la señal varían constantemente. En la figura 2.4 se demuestra cómo se produce una interferencia destructiva absoluta sobre la señal de entrada.

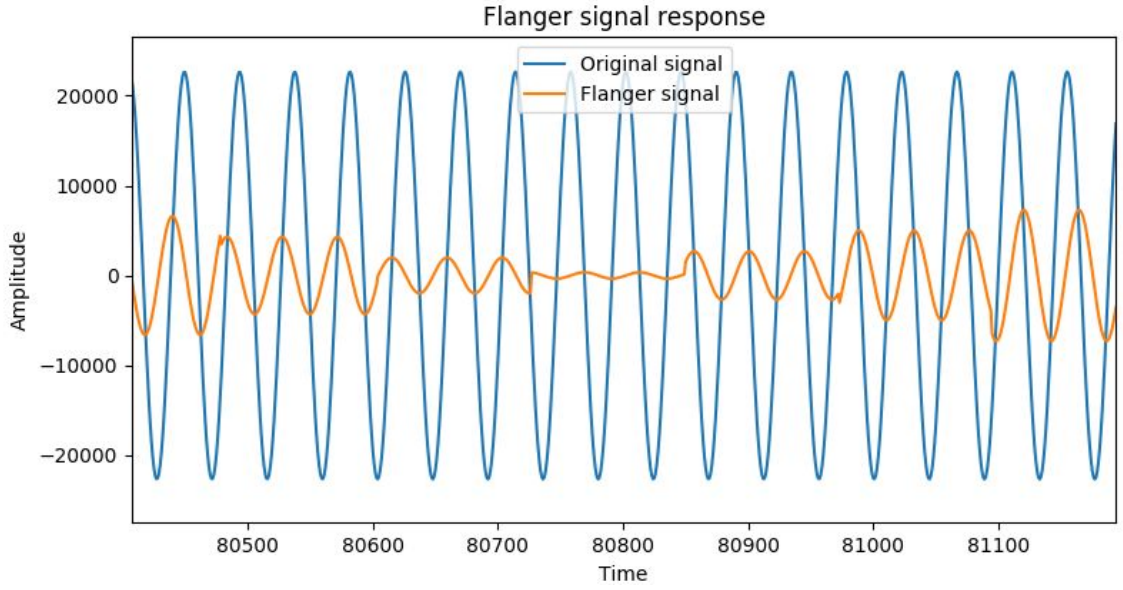

Figura 2.4: Interferencia destructiva absoluta sobre la señal original.

```
def apply_filter(self, original_signal, fs):
    flanger_signal = None
    if len(original_signal) > 0:
        # Create a lambda to call it when process delay later
        def sinus_reference(index): return math.sin(2 * math.pi * index *
                                                    (self.__rate / fs))
        # Convert delay in ms to max delay in samples
        max_delay_sample = round(self.__max_delay * fs)
        # Copy original signal into new one that will be returned
        flanger_signal = numpy.zeros(len(original_signal))

        i = max_delay_sample + 1
        while i < len(original_signal):
            current_sinus = sinus_reference(i)

            current_delay = max_delay_sample * abs(current_sinus)

            flanger_signal[i] = ((original_signal[i]) +
                                 (self.__scale *
                                 original_signal[i - int(current_delay)]))
            i += 1

        # Obtain the max value of flanger
        max_flanger = numpy.amax( numpy.absolute(flanger_signal) )
        # Establish a relation between max flanger value and INT 16 max value
        normalized_relation = FlangerFilter.MAX_INT16_VALUE/max_flanger
        # adapt flanger signal to original signal amplitude
        normalized_flanger = [int(x * normalized_relation) for x in flanger_signal]
        # create an np array to reproduce it then
        flanger_signal = np_array(normalized_flanger)

    return flanger_signal
```

Algoritmo 2.1: Implementación del filtro flanger en lenguaje Python.

Filtro Wah-Wah

Wah-wah es una palabra imitativa (onomatopeya) para el sonido de alterar la resonancia de las notas musicales para extender la expresividad, sonando como una voz humana diciendo la sílaba “wah”. El efecto wah-wah es un deslizamiento espectral, una modificación de la calidad de la vocal de un tono, también conocido como filtro pasa-banda. El efecto se produce al subir y bajar periódicamente las frecuencias pasa-bandas del filtro mientras se mantiene una nota.

El diagrama en bloques generalizao del filtro está representado en la figura 3.1.

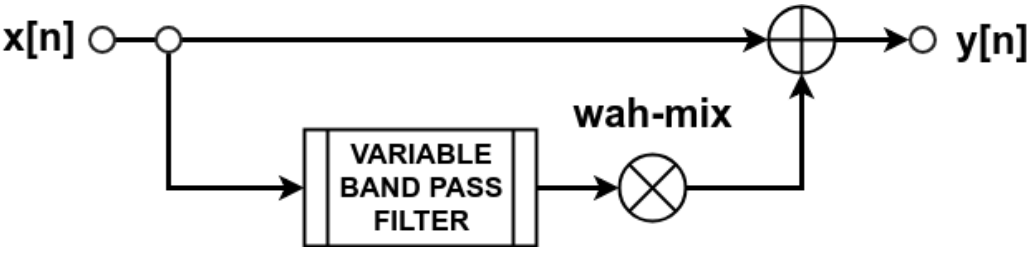

Figura 3.1: Diagrama resumido del filtro wah-wah.

La aplicación del filtro consiste en sumar la señal de entrada actual junto con la señal de entrada a la cual se la pasa por un filtro pasabanda variable.

En algunos instrumentos musicales, como por ejemplo la guitarra eléctrica, la frecuencia del filtro pasa-banda se cambia accionando manualmente un pedal. En este trabajo, se decidió cambiar la frecuencia del filtro automáticamente, con el fin de mostrar su funcionamiento sin interacción manual.

Los parámetros involucrados en el filtro wah-wah son los siguientes:

- **Damping**: Factor de amortiguación del filtro pasa-banda. Valor nominal 0.01-0.07.
- **Min cutoff:** Frecuencia mínima que se va a filtrar. Valor nominal 300 hz.
- **Max cutoff:** Frecuencia máxima que se va a filtrar. Valor nominal 4500 hz.
- **Wah-Wah frequency:** Frecuencia a la que se aplica el efecto. Valor nominal 0.1-2 hz.

La implementación del filtro se basó en un de filtro de variable de estados, con coeficientes variables en el tiempo. Posee dos parámetros que permiten controlar la frecuencia de corte y la amplitud de la banda pasante. Si bien en el efecto wah-wah sólo interesa un filtro pasa-banda, utilizando este tipo de filtro se puede obtener simultaneamente una salida pasa-bajos, una pasa-banda y otra pasa-altos. En la figura 3.2 se puede apreciar un diagrama detallado del filtro de variable de estados implementado.

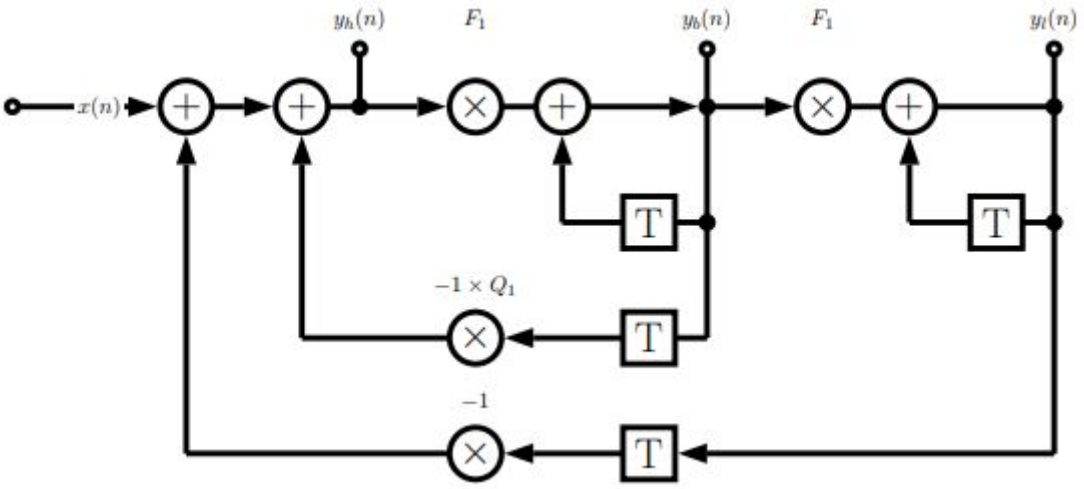

Figura 3.2: Diagrama en bloques del filtro de variables de estado implementado.

Para la figura 3.2, $x(n)$ representa la señal de entrada $yl(n)$ representa la salida pasa-bajos, $yb(n)$ la salida pasa-banda e $yh(n)$ la salida pasa-altos. Las equaciones diferenciales están dadas de la siguiente manera:

$$yl[n] = f1 * yb[n] + yl[n - 1]$$

$$yb[n] = f1 * yh[n] + yb[n - 1]$$

$$yh[n] = x[n] - yl[n - 1] - Q1 * yb[n - 1]$$

Para calcular los parámetros del filtro a utilizar en cada muestra, primero se estableció una frecuencia de corte fc (proveniente del pedal en caso de una guitarra, o una señal triangular para este trabajo). La señal fc oscila entre los valores min/max cutoff, a una frecuencia *Wah-wah frequency* fija. El parámetro $Q1$, que controla el ancho de banda del filtro, se dejó constante para proveer el factor de damping deseado. El valor del parámetro $f1$ para obtener la frecuencia de corte del filtro deseada, surgió de análisis existentes sobre el comportamiento del filtro, y están demostrados de la siguiente manera:

$$f1 = 2 * sin(\pi * fc[n]/fs)$$
$$Q1 = 2 * damping$$

Donde $fc[n]$ es la frecuencia del filtro pasa-banda que se aplica a cada muestra de la señal de entrada, fs es la frecuencia de muestreo de la señal de audio de entrada y $damping$ es la amplitud de la banda pasante.

De manera programática, la implementación del filtro wah-wah consiste en tres partes principales:

- Crear una onda triangular para modular la frecuencia central del filtro pasa-banda.
- Implementar filtro de variables de estado.
- Recalcular la frecuencia central del filtro-pasabanda para cada muestra.

Para crear la onda triangular que modula frecuencia central del filtro pasa-banda, se creó la función ***_create_triangle_signal*** que se demuestra en el algoritmo 3.1 y que toma en cuentalos siguientes parámetros:

- Cantidad de muestras de la señal original que se quiere modular.
- Frecuencia de muestreo de la señal original.
- Frecuencia de corte inferior del filtro wah-wah.
- Frecuencia de corte superior del filtro wah-wah.
- Frecuencia de modulación del filtro wah-wah.

```
def __create_triangle_waveform(self, original_signal_lenght, fs):
        # establish signal period from fs and wah wah frecuency
        signal_period = fs/self.__frequency
        # steps which triangle signal will do, considering the minummum
        # and max value that it has to reach. Also signal period is taken
        # and finally it is multiplied by 2, because the signal will have to
        # increase and decrease it's value in each signal period
        step = ((self.__max_cutoff - self.__min_cutoff)/signal_period) * 2
        # This is internal function that will create the signal, from the min
        # value to max value. It's a generator to make it memory efficient.
        def generator():
            index = 0
            # loop until to reach the lenght of original signal
            while index < original_signal_lenght:
                # each iteration start with an initial value
                signal_value = self.__min_cutoff
                # create ascendent part of triangle
                while signal_value < self.__max_cutoff:
                    signal_value += step
                    index += 1
                    yield signal_value
                # create desscendent part of triangle
                while signal_value > self.__min_cutoff:
                    signal_value -= step
                    index += 1
                    yield signal_value

        # Call the generator of the function and create a list from it
        triangle_signal = list(generator())
        # Trim triangle signal to lenght of original signal
        triangle_signal = triangle_signal[:original_signal_lenght]
        # return triangle singla.
        return triangle_signal
```

Algoritmo 3.1: Creación de señal triangular del filtro wah-wah.

En la figura 3.3 se puede apreciar la creación de esta señal de manera detallada para una señal de entrada de 143.325 muestras, muestreadas a 11,025 Khz (duración 13 segundos), con un min cutoff de 300 hz, un max cutoff de 4500 hz y una frecuencia de modulación del wah wah de 0,5 hz. Es de interés notar cómo, para una señal de entrada de 13 segundos y una frecuencia de modulación de 0.5 hz, la señal de modulación presenta 6,5 períodos.

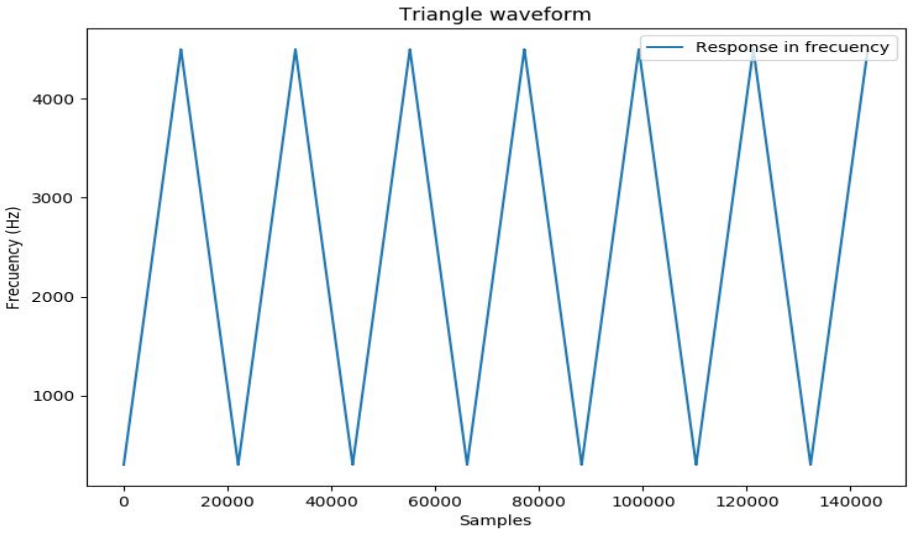

Figura 3.3: Creación de onda triangular para modulación del filtro pasa-banda.

Para la demostración de funcionamiento del filtro se utilizó una señal de entrada correspondiente al sonido de guitarras, muestreadas a una frecuencia de 11,025 Khz con una duración de 13 segundos. En la figura 3.4 se puede observar la implementación del filtro (color naranja) sobre la señal original (color azul). El algoritmo 3.2 muestra la implementación del filtro en una función de lenguaje Python:

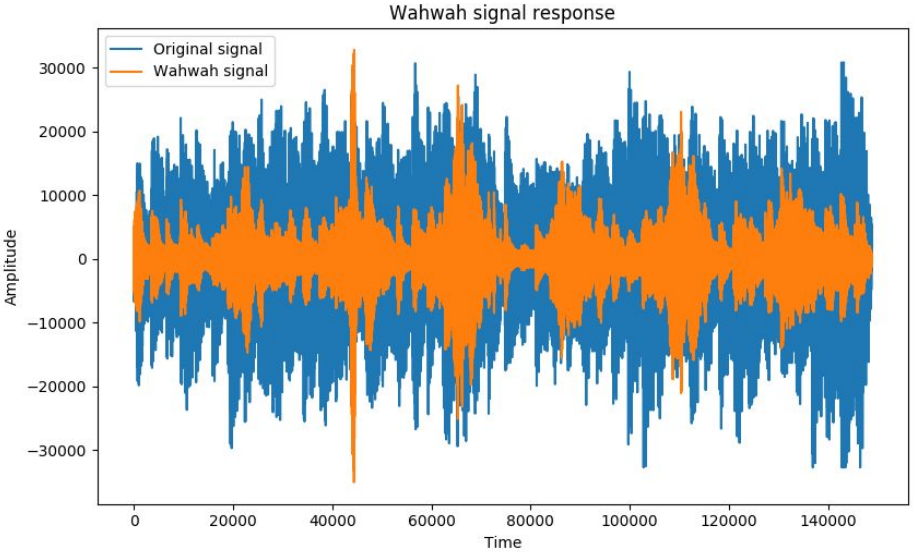

Figura 3.4: Señal del filtro wah-wah sobre señal original.

```
def apply_filter(self, original_signal, fs):
        # Create triangle signal
        cuttoff_frequencies = self.__create_triangle_waveform(len(original_signal), fs)
        # equation coefficients
        f1 = 2 * math.sin((math.pi * cuttoff_frequencies[0])/fs)
        # size of band pass filter
        q1 = 2 * self.__damping
        # initialize filters arrays with zero values
        highpass = numpy.zeros(len(original_signal))
        bandpass = numpy.zeros(len(original_signal))
        lowpass = numpy.zeros(len(original_signal))
        # assign first values
        highpass[0] = original_signal[0]
        bandpass[0] = f1 * highpass[0]
        lowpass[0] = f1 * bandpass[0]
        # loop to reach the lenght of original signal
        for n in range (1, len(original_signal)):
            highpass[n] = original_signal[n] - lowpass[n-1] - (q1 * bandpass[n - 1])
            bandpass[n] = (f1 * highpass[n]) + bandpass[n - 1]
            lowpass[n] = (f1 * bandpass[n]) + lowpass[n - 1]
            # recalculate equation coefficients
            f1 = 2 * math.sin((math.pi * cuttoff_frequencies[n])/fs)
        # Obtain the max value of YB
        max_bandpass = numpy.amax(bandpass)
        # Establish a relation between max YB value and INT 16 max value
        normalized_relation = WahWahFilter.MAX_INT16_VALUE/max_bandpass
        # adapt wahwah signal to original signal amplitude
        normalized_bandpass = [int(x * normalized_relation) for x in bandpass]
        # create an np array to reproduce it then
        wahwah_signal = np_array(normalized_bandpass)

        return wahwah_signal
```

Algoritmo 3.2: Implementación del filtro Wah-wah.

Software controlador de filtros

Para realizar el control de los filtros, se utilizó el lenguaje de programación Python. Mediante este lenguaje y la utilización de las librerías numpy, scipy y matplotlib, además de clases implementadas ad-hoc se realizó un control total de los mismos.

Para manejar adecuadamente los datos y las funciones que el programa debe realizar se utilizó el patrón de diseño de software modelo-vista-controlador. Este patrón de diseño permite separar de manera concreta y unívoca las funciones que debe realizar cada parte del software.

- El modelo se encarga de acceder a los datos, y toda la lógica que se debe realizar con los mismos, por ejemplo instanciar cada uno de los filtros, leer, escribir y actualizar los parámetros de configuración de cada uno, de aplicar cada uno de los filtros sobre las señales, entre otros. También provee acceso para leer y escribir los parámetros de configuración de manera persistente.

- La vista se encarga de presentar los datos así como también capturar los datos ingresados el usuario. Los datos ingresados están asociados directamente a acciones del programa, por ejemplo plotear la señal de un filtro, reproducir una señal filtrada, mostrar las configuraciones actuales, cambiar los parámetros de los filtros, entre otros. Si bien la vista del programa fue realizada por la terminal de comandos, está totalmente desacoplada del resto del código, por lo que puede ser reemplazada por una interfaz gráfica sin necesidad de cambiar el resto del programa.

- El controlador se encarga de llevar a cabo las peticiones del usuario. Esta clase es el nexo entro los métodos provistos por el modelo y por la vista. De esta manera se genera desacople de estas clases, es decir, desde el modelo no se utiliza ninguna pieza de software de la vista, y en la vista no se utiliza ninguna del modelo. Todo el intercambio de información es llevado a cabo por el controlador.

A su vez, a parte de generar el código que da aplicación, se realizó test unitario del código para comprobar el correcto funcionamiento del mismo bajo diferentes escenarios.

Por otro lado, todo el código fue versionado con el controlador de versiones Git mediante la herramienta visual Gitkraken, y el mismo fue subido a un repositorio público de Github.

Características principales del software

- Implementación del patrón de diseño de software modelo-vista-controlador.
- Test unitario con la biblioteca estándar de Python unittest.
- Interfaz con el cliente mediante terminal de comandos.
- Compatible con el standard de Python pep8 mediante agresivo analizador de código.
- Instalación automatizada para sistemas Linux basados en Debian.
- Archivo README con descripción detallada de funcionamiento.
- Ejecución del programa sobre entorno virtual de Python.
- Reproducción de archivos con extensión WAV.
- Ploteo de gráficos de señales mediante biblioteca de Python matplotlib.
- Acceso al sistema de archivos del host para guardar datos de configuración.
- Manejo y visualización de errores en pantalla.
- Modificación de parametros de filtros en tiempo de ejecución.
- Restauración de configuraciones por defecto en caso de configuraciones erróneas.
- Versionado bajo controlador de versiones Git.
- Publicado en repositorio público de Github.

Repositorio público del software

El software está publicado en un repositorio de Github en esta URL:

https://github.com/agustinBassi/dsp_controller/

Conclusiones

Este trabajo consolidó una gran experiencia, ya que por un lado se aplicaron conceptos relacionados con procesamiento de señales y por otro se aplicaron técnicas avanzadas de programación que permitieron darle una estructura sólida a la aplicación de software.

Las principales conclusiones son:

- Realizar una búsqueda en libros digitales y páginas afines es clave para comenzar la investigación.
- Una vez que se recolecta la información se puede proceder con el análisis de los filtros.
- El lenguaje de programación Python posee todas las herramientas necesarias para realizar procesamiento de señales.
- La biblioteca de Python matplotlib provee todas las funciones necesarias para representar el comportamiento de los filtros de manera visual.
- Mediante la representación auditiva de señales a través de las biblioteca scipy se pude percibir el comportamiento de los filtros.

Próximos pasos

Se espera que la aplicación de software creada se utilice como herramienta para trabajos relacionados con el procesamiento de señales. También se espera que haya colaboradores que puedan contribuir al crecimiento y mejora de la aplicación.

Algunas actividades candidatas para los próximos pasos son:

- Encontrar una comunidad a la cual sirva la aplicación.
- Encontrar desarrolladores que quieran colaborar en el crecimiento de la aplicación.
- Encontrar una forma sencilla de agregar nuevos filtros.
- Cambiar la interfaz de consola por interfaz gráfica.
- Instalación automatizada para múltiples sistemas operativos.

Referencias

- Oppenheim Signals and Systems. *A. V. Oppenheim.*
https://www.academia.edu/5922058/Oppenheim_Signals_and_Systems_2nd_Edition_Solutions
Disponible: 2018-04-23.

- Representación filtro WahWah MIDI. *Steinberg.*
https://steinberg.help/cubase_plugin_reference/v9/es/_shared/topics/plug_ref/wahwah_r.html
Disponible: 2018-04-23.

- Sound project using Python. *Ijera.*
https://www.ijera.com/papers/Vol4_issue5/Version%201/B45010811.pdf
Disponible: 2018-04-23.

- Time varying delay effects. *Standford.*
https://ccrma.stanford.edu/~jos/pasp/Time_Varying_Delay_Effects.html
Disponible: 2018-04-23.

- Wah Wah. *Marion, Jean Guy Bruno.*
https://ses.library.usyd.edu.au/bitstream/2123/10578/2/Marion%2C%20Bruno%20-%20Wah%20Wah.pdf
Disponible: 2018-04-23.

- Proyecto fin de carrera, Universidad de Madrid. *Sergio Carrero Jordán.*
http://arantxa.ii.uam.es/~jms/pfcsteleco/lecturas/20140722SergioCarreroJordan.pdf
Disponible: 2018-04-23.

- C code DSP snippets. *DSP related.*
https://www.dsprelated.com/code-2/mp/C/all.php
Disponible: 2018-04-23.

- Galería de imágenes con matplotlib. *Rasbt.*
https://github.com/rasbt/matplotlib-gallery
Disponible: 2018-04-23.

- Digital Audio Effects. *Dave Marshall.*
https://ses.library.usyd.edu.au/bitstream/2123/10578/2/Marion%2C%20Bruno%20-%20Wah%20Wah.pdf
Disponible: 2018-04-23.