

Comparación de algoritmos que realizan procesamiento digital de señales (DSP) enfocado al diseño de plug ins de audio.

Carlos Eduardo Segovia Medina - 13224 AED 0117.1

SAE Institute México



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

I Tabla de contenidos

Introducción

Capítulo I: Problema de investigación

1. Objetivo general
2. Objetivos particulares
3. Justificación
4. Metodología
5. Limitaciones
6. Perspectiva
7. Variables

Capítulo II: Marco teórico

- 2.1 C++
- 2.2 Biblioteca estándar de audio para C++
- 2.3 Base teórica
- 2.4 ADC y DAC

Capítulo III: Desarrollo de los algoritmos

- 3.1 Estadística de señales
- 3.2 Convolución
- 3.3 Transformada Discreta de Fourier (DFT)
- 3.4 Transformada Discreta de Fourier Inversa (IDFT)
- 3.5 Transformada Rápida de Fourier (FFT)
- 3.6 Diseño de filtros

Capítulo IV: Pruebas y resultados

- 4.1 Prueba de la media
- 4.2 Prueba de la Desviación estándar
- 4.3 Prueba de la RMS
- 4.4 Prueba del Signal to Noise Ratio
- 4.5 Prueba de la convolución
- 4.6 Prueba de DFT
- 4.7 Prueba de FFT
- 4.8 Prueba de Moving Average Filter
- 4.9 Prueba de Recursive Moving Average Filter
- 4.10 Prueba del Windowed Sync Filter

Capítulo V: Conclusión

Bibliografía



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

II Indice

| | |
|--|-----------|
| Capítulo 1: Problema de investigación | 16 |
| 1-. Objetivo general | 16 |
| Una librería de funciones (objetos) existentes de procesamiento digital de señal para compresión y efectos de tiempo en audio. | 16 |
| 2-. Objetivos particulares | 16 |
| 3-. Justificación: | 17 |
| 4-.Descripción de Metodología: | 18 |
| 5-.Limitaciones | 18 |
| 6-.Perspectiva | 18 |
| 7. Variables | 19 |
| Capítulo 2: Marco Teórico | 19 |
| 2.1 C++ | 19 |
| 2.1.1 Lenguajes de programación. | 19 |
| 2.1.1.1 El lenguaje de máquinas | 19 |
| 2.1.1.2 Lenguaje de ensambladores | 20 |
| 2.1.1.3 Lenguaje de alto nivel | 21 |
| 2.1.2 Breve historia | 21 |
| 2.1.3 Características | 21 |
| 2.1.4 Programación orientada a objetos | 22 |
| 2.1.5 Funciones | 22 |
| 2.1.6 Clases | 23 |
| 2.1.7 Herencia | 23 |
| 2.1.8 Conclusiones de C++ | 23 |
| 2.2. API estándar de audio para C++ | 24 |
| 2.2.1 Breve historia | 24 |
| 2.2.2 Lenguaje de bajo nivel | 24 |
| 2.2.3 Alcance y target | 24 |
| 2.2.4 Conclusión | 25 |
| 2.3. Base teórica. | 25 |
| 2.3.1 Sumatorias | 25 |
| 2.3.2.1 Radianes | 26 |
| 2.3.2.2 Círculo unitario | 27 |
| 2.3.2.3 Razones trigonométricas | 29 |
| 2.4. ADC y DAC | 31 |



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

| | |
|---|-----------|
| 2.4.1 Cuantización | 31 |
| 2.4.2 Teorema de Nyquist | 33 |
| Capítulo 3: Desarrollo de algoritmos | 34 |
| 3.1 Estadística de señales | 34 |
| 3.1.1 El promedio (Mean) | 34 |
| 3.1.1.1 Definición | 34 |
| 3.1.1.2 Algoritmo básico | 35 |
| 3.1.1.3 Algoritmo avanzado | 35 |
| 3.1.2 Standard Deviation | 37 |
| 3.1.2.1 Definición | 37 |
| 3.1.2.2 Algoritmo básico | 37 |
| 3.1.2.3 Algoritmo avanzado | 38 |
| 3.1.3 RMS | 40 |
| 3.1.3.1 Definición | 40 |
| 3.1.3.2 Algoritmo básico | 40 |
| 3.1.3.3 Algoritmo avanzado | 41 |
| 3.1.4 Signal to Noise Ratio | 41 |
| 3.1.4.1 Definición | 41 |
| 3.1.4.2 Algoritmo | 42 |
| 3.2 Convolución | 42 |
| 3.2.1 Definición | 42 |
| 3.2.2 Algoritmo básico | 43 |
| 3.2.3 Algoritmo avanzado | 44 |
| 3.2.3.1 Fórmula matemática | 44 |
| 3.2.3.2 Algoritmo en C++ | 45 |
| 3.3 Transformada Discreta de Fourier (DFT) | 46 |
| 3.3.1 Definición | 46 |
| 3.3.2 Algoritmo | 47 |
| 3.4 Transformada Discreta de Fourier Inversa (IDFT) | 48 |
| 3.4.1 Definición | 48 |
| 3.4.2 Algoritmo | 49 |
| 3.5 Transformada Rápida de Fourier (FFT) | 50 |
| 3.5.1 Definición | 50 |
| 3.5.2 Algoritmo | 51 |
| 3.6 Diseño de filtros | 52 |
| 3.6.1 Moving Average Filter | 53 |
| 3.6.1.1 Algoritmo | 55 |



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

| | |
|---|-----------|
| 3.6.2 Recursive Moving Average Filter | 55 |
| 3.6.2.1 Algoritmo | 56 |
| 3.6.3 Windowed-Sync Filter | 57 |
| 3.6.3.1 Algoritmo | 59 |
| Capítulo 4: Pruebas y resultados | 60 |
| 4.1 Prueba de la media | 61 |
| 4.2 Prueba de la Desviación estándar | 61 |
| 4.3 Prueba de la RMS | 62 |
| 4.4 Prueba del Signal to Noise Ratio | 63 |
| 4.5 Prueba de la convolución | 64 |
| 4.6 Prueba de DFT | 66 |
| 4.7 Prueba de FFT | 69 |
| 4.8 Prueba de Moving Average Filter | 73 |
| 4.9 Prueba de Recursive Moving Average Filter | 74 |
| 4.10 Prueba del Windowed Sync Filter | 76 |
| Capítulo 5: Conclusiones | 80 |
| Bibliografía | 81 |



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Índice de imágenes

Capítulo II: Marco teórico

- Fig. 2.1 Lenguaje de máquina
- Fig. 2.2 Lenguaje de ensamblador
- Fig. 2.3 Lenguaje de alto nivel
- Fig. 2.4 Letra griega sigma que en el campo de la matemática representa la sumatoria
- Fig. 2.5 Descripción de un radian
- Fig. 2.6 Círculo unitario
- Fig. 2.7 Representación de un ángulo de 45 grados sobre el círculo unitario
- Fig. 2.8 Triángulo rectángulo sobre círculo unitario
- Fig. 2.9 El cateto adyacente y opuesto de un triángulo varía dependiendo del ángulo que se tome en cuenta
- Fig. 2.10 Identidades trigonométricas deducidas con el círculo unitario
- Fig. 2.11 Identidades trigonométricas deducidas sobre el círculo unitario
- Fig. 2.12 Señal continua
- Fig. 2.13 Sample & Hold
- Fig. 2.14 Cuantización
- Fig. 2.15 Aliasing

Capítulo III: Desarrollo de los algoritmos

- Fig. 3.1 El promedio de una señal en lenguaje matemático
- Fig. 3.2 Función que calcula el promedio de una señal
- Fig. 3.3 Método Loop Unrolling usado para calcular el promedio de una señal
- Fig. 3.4 Manejo de caso en que tamaño del arreglo no sea múltiplo de 4
- Fig. 3.5 Fórmula de la desviación estándar.
- Fig. 3.6 Función que calcula la desviación estándar de una señal
- Fig. 3.7 Método Loop Unrolling aplicado en el algoritmo avanzado
- Fig. 3.8 Otra fórmula matemática para expresar la desviación estándar.
- Fig. 3.9 Fórmula matemática para expresar el rms
- Fig. 3.10 Función que calcula el rms de una señal
- Fig. 3.11 Método Loop Unrolling aplicado en el cálculo de rms
- Fig. 3.12 Fórmula matemática del Signal to noise Ratio y CV
- Fig. 3.13 Algoritmo de signal to noise ratio y del coeficiente de variación
- Fig. 3.14 fórmula matemática de la convolución
- Fig. 3.15 Algoritmo de convolución
- Fig. 3.16 Fórmula matemática de la convolución desde el punto de vista de la salida.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Fig 3.17 Fórmula de convolución en lenguaje matemático. $h[n]$ es un arreglo de tamaño M.

Fig 3.18 Algoritmo avanzado y formula matemática de convolución

Fig 3.19 Fórmula matemática de la transformada discreta de Fourier

Fig 3.20 Algoritmo de la transformada de Fourier en C++

Fig 3.21 Fórmula matemática de la Transformada Discreta de Fourier Inversa

Fig 3.21 Reconversión de los valores ReX e ImX, y manejo de dos casos especiales.

Fig 3.22 Fórmula matemática de la Transformada Discreta de Fourier Inversa

Fig 3.23 Código en C++ del primer paso de la FFT

Fig 3.24 Código en C++ de la FFT

Fig 3.25 Fórmula matemática del Moving Average Filter

Fig 3.26 Cálculo del filtro para la muestra número 80

Fig 3.27 Antes y después de usar un filtro de 11 puntos en un señal

Fig 3.28 Código en C++ del Moving Average Filter

Fig 3.29 Cálculo del Moving average filter para las muestras número 50 y 51

Fig 3.30 Cálculo recursivo para de la muestra número 51

Fig 3.31 Fórmula matemática recursiva del Moving average filter

Fig 3.32 Código en C++ del Recursive Moving Average Filter

Fig 3.33 Gráfica de la respuesta en frecuencia del Windowed-sinc filter

Fig 3.34 Fórmula matemática del Windowed-sinc filter

Fig 3.35 Gráfica que muestra el papel que juega el valor M

Fig 3.36 Código en C++ del Windowed-sinc filter

Capítulo IV: Pruebas y resultados

Fig 4.1 Resultado en consola del algoritmo de la media

Fig 4.2 Resultado en consola del algoritmo de la media

Fig 4.3 Resultado en consola del algoritmo de la Desviación estándar

Fig 4.4 Resultado en consola del algoritmo de la Desviación estándar

Fig 4.5 Resultado en consola del algoritmo de la RMS

Fig 4.6 Resultado en consola del algoritmo de la RMS

Fig 4.7 Resultado en consola del algoritmo de la Signal to Noise Ratio

Fig 4.8 Resultado en consola del algoritmo de la Signal to Noise Ratio

Fig 4.9 Señal de entrada

Fig 4.10 Señal de impulso

Fig 4.11 Señal de salida de algoritmo 3.2.2

Fig 4.12 Resultado en consola del algoritmo de convolución

Fig 4.13 Señal de salida de algoritmo 3.2.3

Fig 4.14 Resultado en consola del algoritmo de convolución

Fig 4.15 Señal de entrada

Fig 4.16 Gráfica de la parte Imaginaria y la parte Real de nuestra señal de entrada

Fig 4.17 Espectro frecuencial de la señal de entrada.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Fig 4.18 Resultado de la IDFT

Fig 4.18 Resultado en consola del algoritmo DFT

Fig 4.19 Señal de entrada

Fig 4.20 Gráfica de la parte Imaginaria y la parte Real de nuestra señal de entrada

Fig 4.21 Espectro frecuencial de la señal de entrada.

Fig 4.22 Resultado de la IDFT

Fig 4.23 Resultado en consola del algoritmo FFT

Fig 4.24 Señal de entrada

Fig 4.25 Señal de salida

Fig 4.26 Resultado en consola del algoritmo Moving average filter

Fig 4.27 Señal de entrada

Fig 4.28 Señal de salida

Fig 4.29 Resultado en consola del algoritmo Recursive Moving average filter

Fig 4.30 Señal de entrada

Fig 4.31 Señal de impulso

Fig 4.32 Señal de salida

Fig 4.33 Resultado en consola del algoritmo Windowed Syn Filter mas la convolución entre las dos señales.

Fig 4.34 Señal de salida

Fig 4.35 Resultado en consola del algoritmo Windowed Syn Filter más la convolución entre las dos señales usando el algoritmo de la sección 3.2.3.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

IV Anexos

Código:

Anexo 1:

Convolution.cpp - Computa el algoritmo de convolución.

Anexo 2:

DFT.cpp - Computa el algoritmo DFT.

Anexo 3:

FFT.cpp - Computa el algoritmo FFT.

Anexo 4:

IDFT.cpp - Computa el algoritmo IDFT.

Anexo 5:

main.cpp - Contiene la aplicación que compara todos los algoritmos.

Anexo 6:

Moving_average_filter.cpp - Computa el algoritmo Moving Average Filter.

Anexo 7:

Recursive_moving_average.cpp - Computa el algoritmo Recursive Moving Average Filter.

Anexo 8:

Robust_Convolution.cpp - Computa el de convolución avanzado.

Anexo 9:

Robust_mean.cpp - Computa el algoritmo Mean avanzado

Anexo 10:



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Robust_rms.cpp - Computa el algoritmo RMS avanzado.

Anexo 11:

Robust_Standard_Deviation.cpp - Computa el algoritmo Standard deviation avanzado

Anexo 12:

Signal_Statistics.cpp - Computa todos los algoritmos básicos de la sección de estadísticas de señales

Anexo 13:

Windowed_sync_low_pass_filter.cpp - Computa Windowed sync filter.

Archivos

Anexo 14:

Convolution_Output_Signal.dat - Archivo generado de algoritmo de convolución.

Anexo 15:

DFT_IMX.dat - Archivo genera del algoritmo DFT representa la parte imaginaria.

Anexo 16:

DFT_INPUT.dat - Archivo de entrada de algoritmo de DFT.

Anexo 17:

DFT_MAG.dat - Archivo de salida de algoritmo DFT.

Anexo 18:

DFT_REX.dat - Archivo generado del algoritmo DFT representa la parte real.

Anexo 19:

FFT_IDFT_OUTPUT.dat - Archivo de señal de salida del algoritmo IDFT



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Anexo 20:

FFT_IMX.dat - Archivo generado del algoritmo FFT representa la parte imaginaria.

Anexo 21:

FFT_INPUT.dat - Archivo de entrada al algoritmo FFT

Anexo 22:

FFT_MAG.dat - Archivo de salida de algoritmo FFT.

Anexo 23:

FFT_REX.dat - Archivo generado del algoritmo FFT representa la parte real.

Anexo 24:

IDFT_OUTPUT.dat - Archivo de señal de salida del algoritmo IDFT

Anexo 25:

Impulse_Response.dat - Archivo representa un filtro pasabajas de 10 Khz

Anexo 26:

Input_Signal.dat - Archivo representa una señal con un componente de 1 KHz y otro de 15 KHz

Anexo 27:

FFT_IDFT_OUTPUT.dat - Archivo de señal de salida del algoritmo IDFT

Anexo 28:

FFT_IMX.dat - Archivo generado del algoritmo FFT representa la parte imaginaria.

Anexo 29:

MAF_INPUT.dat - Archivo de entrada al algoritmo Moving Average filter

Anexo 30:

MAF_OUTPUT.dat - Archivo de salida al algoritmo Moving Average filter.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Anexo 31:

RECURSIVE_INPUT.dat - Archivo de entrada al algoritmo Recursive Moving Average filter

Anexo 32:

RECURSIVE_OUTPUT.dat - Archivo de salida al algoritmo Recursive Moving Average filter.

Anexo 33:

Robust_Convolution_Output_Signal.dat - Archivo de salida del algoritmo Robust Convolution

Anexo 34:

WINDOWED_FILTER.dat - Archivo que representa el filtro generado por el algoritmo Windowed Sync Filter.

Anexo 35:

WINDOWED_INPUT.dat - Archivo de señal de entrada al algoritmo Windowed Sync Filter.

Anexo 36:

WINDOWED_OUTPUT_ROBUST_CONV.dat - Archivo de salida de convolución entre filtro generado por Windowed sync filter y la señal de entrada

Anexo 37:

WINDOWED_OUTPUT.dat - Archivo de salida de convolución entre filtro generado por Windowed Sync Filter y la señal de entrada.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

V Agradecimientos

Me gustaría agradecer a todas las personas que me han apoyado a en la realización de este trabajo, y a todas las personas que me han apoyado a lo largo de mi etapa universitaria.

Gracias a SAE Institute México, y a todas las personas que lo conforman, por su apoyo y por todas las herramientas que han puesto a mi disposición que me han permitido convertirme en el profesional que soy hoy. Al personal docente por su vocación y su paciencia. Al personal de limpieza, seguridad y mantenimiento por ayudar a hacer de SAE un hogar. A todo el personal de supervisión de audio por su apoyo incondicional a lo largo de estos tres años, en especial a, Juan, Salvador y René. A Ana Sanchez y América Rojas también les agradezco todo su apoyo y asesorías durante el transcurso de mi carrera.

Sobre todo, quiero agradecer a mi familia, mis padres Brayain y Gladys, y mis hermanos Nathalia y Gustavo, por su apoyo incondicional todos estos años. Sin ellos, nada de esto hubiera sido posible, y espero demostrar mi gratitud aprovechando al máximo las oportunidades que se me presenten, para convertirlas en éxitos y logros.

Especial reconocimiento merece Pablo García, por su asesoría y consejos. Rafael Arias, por compartir toda su experiencia, ofrecer su apoyo y su ayuda. Isaac de la Mora por su apoyo, asesoría, ayuda y paciencia.

Finalmente, quiero agradecer a Dante, Jotzil, Aquiles, Angel, Leonardo, Mateo por haberme ayudado tanto en todos los años de mi carrera y por el soporte que me dieron para poder completar mis sueños.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

V Abstract

Digital Signal processing algorithms allow us to create audio plug ins, which can be used in music production either as standalone or through a DAW (Digital Audio Workstation). On the other hand, C++ is the most popular programming language for the low latency required on music production, due to its high performance design. This language is therefore chosen as the center of this work. Its high performance happens because it is just one level above the assembly language of programming.¹

The signal path in a recording scenario is digital since the moment it enters in the audio interface, in this state we can prevent the degradation generated by analog storage and manipulation². This can be notice when comparing the quality of vinyl recordings with compact disk. One of the last steps of music production is the mix down in which several tracks are combined to produce the final product. During this process the use of DSP, represented with a graphical interface that resembles the traditional analog piece of gear more often used, provide a wide variety of tools like filtering, signal addition and subtraction and signal editing for better processing of audio.

The original focus of this thesis was to develop an audio plugin which could be used for music production. But the aim quickly changed to a comparison between several DSP algorithms written in C++ and their mathematical approach as a necessary basis in audio plug in development. This because, just as an analog audio equipment use transistors and capacitors to process the electric signal which represents the wave sound, the equivalent digital audio equipment use DSP algorithms to do the same process to digital signal.

The scope of this thesis is limited to the basic concepts of DSP, according to The Scientist and Engineer's Guide to Digital Signal Processing by Steven W. Smith, Ph.D³, and their further development of the algorithms in C++. The experiments performed during this investigation were done just with one signal of three hundred and twenty samples (320) generated at a sample rate of 48 KHz in Matlab, this signal has two components one of 1 KHz and a second of 15 KHz.

First, the basic concepts of C++ language were studied, the mathematical notions required were revised, some contributions of others programmers were

¹ "C++ Audio Library Options (2018) - Superpowered." <https://superpowered.com/audio-library-list>. Se consultó el 29 nov.. 2019.

² (n.d.). Audio Processing - DSP Guide. Se recuperó el diciembre 12, 2019 de <https://www.dspguide.com/ch1/3.htm>

³ (n.d.). The Scientist and Engineer's Guide to Se recuperó el diciembre 1, 2019 de <http://www.dspguide.com/>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

found and finally digital audio concepts were researched. All that information allows for a comprehension and further development of the basic methods or functions necessary to implement an audio plug in.

The methods analyzed in this thesis were signal statistics (mean, standard deviation, RMS, signal to noise ratio), convolution, DFT, IDFT, FFT and design filters like moving average filter, recursive moving average filter and windowed-sync-filter.

Based on the mathematical concepts of mean, standard deviation, RMS, signal to noise ratio and convolution, two groups of algorithms were develop for each method. The first one is really easy to read and to understand the mathematics behind it, while the second group is more complex but proved to be more efficient.

After all the groups of algorithms were programmed, an application was written in order to compared both types of algorithms for every method in time and precision.

The results, as expected, showed that the second group of algorithm was more efficient than the first one. All these comparison were done using the C time library which contains definitions of functions to get and manipulate date and time information⁴.

To quickly recap the main purpose of the comparison between these algorithms, we must remember that the goal is to use these functions and methods to implement audio plugins. The signal statistics algorithms could be applied to develop a plugin that analyze the overall characteristics of an audio program. The convolution algorithm is a fundamental operation through which we can sum signals and develop convolution reverberation based audio plug ins⁵. The DFT will be useful to calculate the signal's frequency spectrum⁶. Also, the filters could be implemented to develop semiparametric equalizers.

Finally, it was concluded that the algorithms developed during this investigation, can be implemented in audio plugins successfully since the efficiency they present will allow us to make real time processing. However, the concepts here presented need to be studied further in order to achieve a better understanding of the capabilities of the algorithms developed. The FFT requires a specialized

⁴ "<ctime> (time.h) - C++ Reference" <http://www.cplusplus.com/reference/ctime/>. Se consultó el 29 nov.. 2019.

⁵ "What is Convolution Reverb? : Ask.Audio." 22 sept.. 2017, <https://ask.audio/articles/what-is-convolution-reverb>. Se consultó el 29 nov.. 2019.

⁶ (n.d.). Applications of the DFT - DSP Guide. Se recuperó el diciembre 5, 2019 de <https://www.dspguide.com/ch9.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

investigation since it is the most important concept of DSP⁷ and its applications will be worked separately in the future.

⁷ (n.d.). FFT: "the most important numerical algorithm of our lifetime".. Se recuperó el diciembre 3, 2019 de http://physics.oregonstate.edu/~grahamat/COURSES/ph421/lec/L14_M.pdf



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Capítulo 1: Problema de investigación

1-. Objetivo general

Hacer un análisis profundo de los conceptos básicos del procesamiento digital de señal, derivando sus fórmulas matemáticas y desarrollando algoritmos rápidos para su computación para posteriormente comparar sus desempeños y evaluar su eficiencia en el desarrollo de plug ins de audio.

Una librería de funciones (objetos) existentes de procesamiento digital de señal para compresión y efectos de tiempo en audio.

2-. Objetivos particulares

2-. Desarrollar algoritmos básicos que realicen análisis de procesamiento digital de señales.

3-. Desarrollar algoritmos eficientes derivados de los algoritmos básicos que realicen el mismo análisis pero en menor tiempo.

4-. Crear un programa que contenga todos estos algoritmos y que puedan servir para futuras aplicaciones relacionadas al audio.

5-. Seleccionar funciones y objetos para la futura elaboración de plugins de compresión, reverberación, eco y *delay*.

3-. Justificación:

Este ejercicio es el primer paso de un proyecto de creación de plug ins de audio. El avance tecnológico ha colocado la producción musical al alcance de cada vez más personas, los softwares de audio han jugado un papel trascendental en este hecho, permitiendo tener en una computadora lo que otrora abarcaba grandes espacios en estudios de grabación de elevados costos.

Esto ha generado un mercado conformado por músicos y aficionados del audio que desean generar contenido musical pero no pueden permitirse rentar un



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

estudio grabacion profesional. El avance tecnológico relacionado a la velocidad de procesamiento permitirá que un software de audio pueda generar el mismo resultado que su contraparte análoga, y su costo sea mucho más bajo.

Debido a la extensión de este proyecto, para los fines de esta tesis, se planea desarrollar solo una librería que contenga las funciones básicas para realizar procesamiento digital de señales.

Más allá del desarrollo de plug ins de audio, proveer a los usuarios de un breve análisis de los conceptos básicos relacionados al procesamiento digital de señales y su relación práctica con el audio.

Es importante porque la industria de plug ins de audio se encuentra muy lejos de estar saturada, y representa una oportunidad de negocio, para cualquiera que entienda del tema, sobretodo en países que carecen de empresas que se dediquen a diseñar estos productos. México juega un papel importante en la economía de Latinoamérica, y solo cuenta con una pequeña empresa (Audio-Assault)⁸, ubicada en Puebla que diseña plug ins de audio.

4-Descripción de Metodología:

- 1-. Entender los conceptos fundamentales relacionados al procesamiento digital de señal, según The Scientist and Engineer's Guide to Digital Signal Processing by Steven W. Smith, Ph.D.
- 2-.Generar algoritmos matemáticos que representen los conceptos fundamentales relacionados al procesamiento digital de señal.
- 3-. Traducir estos algoritmos a C++ de manera exitosa.
- 4-.Desarrollar algoritmos que cumplan las mismas funciones pero que sean más eficientes y compararlos con los primeros en función del tiempo.
- 5-. Agrupar todos las funciones en un solo archivo y generar una interface amigable para su uso.
- 6-. Explicar el papel que jugaran los algoritmos en el desarrollo de plug ins de audio.

⁸ (n.d.). Audio Assault. Se recuperó el diciembre 1, 2019 de <https://audio-assault.com/>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

5.Limitaciones

- 1-. No se tocará ningún aspecto del desarrollo de plug ins de audio fuera de los algoritmos básicos que se planea desarrollar.
- 2-. La velocidad de los algoritmos podría no ser la adecuada para objetivo final.
- 3-. Los algoritmos no se probaran con señales de audio per se, sino con arreglos de un tamaño predefinido generados artificialmente que representen señales de audio en un entorno controlado.

6.Perspectiva

- 1-. Ingeniería en audio
- 2-. Lenguajes de programación C++
- 3-. Procesamiento digital de señales (DSP)
- 4-. Matemáticas
- 5-. Algoritmos de programación

7. Variables

- 1-. Tamaño total de cada archivo
- 2-. Rapidez de los programas
- 3-. Precisión de cálculos.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Capítulo 2: Marco Teórico

2.1 C++

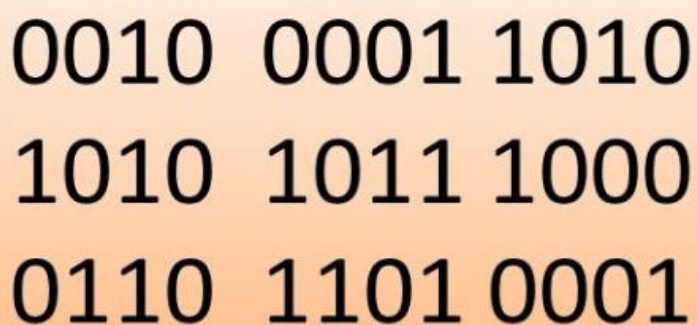
2.1.1 Lenguajes de programación.

En el ámbito de la programación existen una gran cantidad de lenguajes, los cuales pueden ser, de manera general, divididos en tres categorías, cada una difiere de las otras dos en la forma de hacerle llegar las instrucciones escritas por el usuario a la computadora, y cada uno permite al usuario poder escribir dichas instrucciones de una manera cada vez más parecida a la forma que usamos diariamente para comunicarnos entre nosotros. Estos lenguajes son:

- 1-. Lenguaje de máquina.
- 2-. Lenguaje de ensambladores.
- 3-. Lenguaje de alto nivel

2.1.1.1 El lenguaje de máquinas

El lenguaje de máquinas es el lenguaje natural de las computadoras, este es particular de ella y está definido por el diseño del hardware de la misma, se representa como una cadena de números, por lo general unos (1) y (0), en sistema binario, y le indican a la computadora las instrucciones más elementales. Este lenguaje resulta muy difícil de entender para los humanos.



```
0010 0001 1010
1010 1011 1000
0110 1101 0001
```



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Fig. 1.1 Lenguaje de máquina

2.1.1.2 Lenguaje de ensambladores

El lenguaje de ensambladores es una base formada por abreviaturas en inglés, que representan las funciones más básicas de la computadora, luego un programa traduce estas instrucciones en lenguaje ensamblador a lenguaje máquina para que puedan ser leídos por la computadora, este tipo de lenguaje resulta más fácil de entender para los humanos, y mientras no se traduzcan a lenguaje máquina resultan incomprensibles para una computadora.

```

CODESG SEGMENT PARA 'Code'
BEGIN  PROC FAR
        ASSUME SS:STACK,DS:DATASG CS:CODESG
        MOV  AX,DATASG      ;Se asigna direccion de DATASG
        MOV  DS,AX          ; En el registro DS

        MOV  AX,FLDA        ;Mover 0250 a AX
        ADD  AX,FLDB        ;Sumar 0125 a AX
        MOV  FLDC, AX       ;Almacenar suma en FLDC
        MOV  AX,4C00H       ;salida a DOS
        INT  21H

        BEGIN  ENDP        ;Fin del procedimiento
CODESG  ENDS              ;Fin del segmento
        END      BEGIN    ;Fin del programa
    
```

Fig. 1.2 Lenguaje de ensamblador

2.1.1.3 Lenguaje de alto nivel

C++ pertenece al tercer tipo, a los lenguaje de alto nivel, estos permiten que instrucciones individuales realicen tareas importantes para la cual un lenguaje de ensamblador requeriría de muchas instrucciones. Estos lenguajes permiten al usuario escribir instrucciones que se parecen mucho al inglés común, y usar notación matemática común, que luego son traducidas por un programa traductor, que se llama compilador, a lenguaje de máquina.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

ejemplo C: Hola Mundo!

```
#include <stdio.h>

int main()
{
    printf("Hola Mundo!\n");
    return 0;
}
```

Fig 1.3 Lenguaje de alto nivel

2.1.2 Breve historia

C++ es la evolución de C, que es la evolución de BCPL y B. Desarrollado por Bjarne Stroustrup, en los laboratorios Bell en los ochenta no sólo es una versión mejorada del lenguaje C, sino que además introduce el concepto de programación orientada a objetos. (Deitel, Harvey, Paul J. Deitel ,2003) Los objetos son, esencialmente, componentes reutilizables de software que modelan elementos de la vida real, estos tipos de programa resultan más fáciles de comprender, corregir y modificar que la formas de programar anteriores como la programación estructurada.(Deitel, Harvey, Paul J. Deitel ,2003) C++ es un lenguaje híbrido (en el que es posible programar tanto estilo C, como en estilo orientado a objetos , o en ambos)

2.1.3 Características

Las piezas con las que se diseñan programas en C++ se denominan clases y funciones. (Deitel, Harvey, Paul J. Deitel ,2003) Existen dos partes que se deben aprender en el “mundo” de C++. La primera es aprender el lenguaje C++ en sí mismo, la segunda es aprender a utilizar las funciones y clases de la biblioteca estándar de C++.

“Cuando se programa en C++, generalmente se utilizan los siguientes bloques de construcción: clases y funciones de la biblioteca estándar de C++, clases y funciones creadas por uno mismo, y clases y funciones populares de diversos proveedores adicionales.”(Como programar en C++, 2003, p.10)

“La ventaja de crear sus propias funciones y clases es que sabrá exactamente cómo funcionan y podrá examinar el código de C++. La desventaja es el tiempo y esfuerzo que acompañan al diseño, el desarrollo y el mantenimiento para que las nuevas funciones y clases sean correctas y para que operen de manera eficiente.” (Como programar en C++, 2003, p.10)



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

2.1.4 Programación orientada a objetos

La programación orientada a objetos en C++ es más fácil de entender si se usa como analogía el lenguaje que usamos, en donde los sustantivos representan las clases y los verbos representan las funciones, este tipo de enfoque permite crear unidades útiles de software, que suelen ser grandes y estar enfocadas a aplicaciones particulares. Además con este lenguaje se puede reflejar de manera fácil y efectiva a las entidades del mundo real que pueden ser particularmente reutilizables. En el ámbito laboral esto permite diseñar software que son más comprensibles, mejor organizados y fáciles de mantener y corregir. Esto es importante ya que gran parte de los costos de software están relacionados con la continua evolución y mantenimiento del mismo durante su vida útil.

“La ventaja de crear su propio código es que usted sabrá exactamente cómo funciona. La desventaja es el tiempo y esfuerzos que implica el diseño e implementación del nuevo código”.(Como programar en C++, 2003, p.15)

2.1.5 Funciones

En programación se le denomina *divide y vencerás* a la técnica basada en desarrollar a través de piezas y componentes más pequeños y manejables un programa mucho más grande. Estos pequeños componentes se les denomina funciones, estas funciones son tareas específicas reutilizables a lo largo del programa. (Deitel, Harvey, Paul J. Deitel, 2003).

“Una función se invoca (es decir, se hace que realice la tarea para la que fue diseñada) mediante una llamada a la función. La llamada a la función especifica el nombre de la función y proporciona información que la función invocada necesita para hacer su trabajo”.) Una analogía común para esto es la forma jerárquica de administración. Un jefe (la función invocadora, o que invoca) le pide a un trabajador (la función invocada) que realice una tarea y regrese (es decir, le informe), los resultados cuando la tarea está hecha. La función jefe no sabe cómo la función trabajador realiza sus tareas asignadas. El trabajador podría llamar a otras funciones trabajadoras y el jefe no se enteraría de esto.”(Como programar en C++, 2003, p.171)

2.1.6 Clases

“Las clases son paquetes que contienen datos y funciones ambas estrechamente ligados entre sí. Una clase es como un anteproyecto. A partir de un anteproyecto, un constructor puede construir una casa. A partir de una clase, un programador puede crear un objeto. Un anteproyecto se puede reutilizar muchas veces para hacer varias casas. Una clase se puede reutilizar muchas veces para hacer muchos objetos de la misma clase.”(Como programar en C++, 2003, p.406)



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

2.1.7 Herencia

El último concepto necesario para entender cómo funciona de una manera muy general el lenguaje de programación de C++ es **la herencia**, que es una de las características más importantes de la tecnología orientada a objetos, que permite reutilizar software ya existente para crear una clase que absorbe los funciones y datos de otra clase, agregando nuevos parámetros y funciones y modificando en algunos casos los ya existentes, de esta manera se dice que una nueva clase **hereda** los miembros de una clase existente, que se denomina **clase base**.

Un ejemplo muy utilizado para representar la herencia en el mundo real, es el de los automóviles. La *idea* del automóvil, con todas sus características inherentes, sería nuestra *clase*, las acciones que puede realizar este automóvil tales como, encender el motor, acelerar, frenar, doblar a la derecha, serían nuestras *funciones*, el automóvil en el mundo real sería el objeto, y se puede afirmar que un automóvil es un vehículo, pero no es igual de válido decir que todo vehículo es un automóvil, ya que existen diferentes tipos de vehículos, motos, bicicletas, aviones, barcos. En este caso *vehículo* es la clase base de la cual se deriva la *clase automóvil*, y comparte ciertas características con el resto de *clases derivadas* como *moto*, *barco*, *avión*, *bicicleta*, pero realiza las mismas acciones de manera diferente, por ejemplo la *función* acelerar de automóvil, difiere mucho de la manera de acelerar de bicicleta.

2.1.8 Conclusiones de C++

Este ejemplo básico permite entender de manera muy general como funciona C++ y la tecnología orientada a objetos. Lo que se pretende con esta tesis es generar funciones enfocadas a realizar procesamiento digital de señal, que luego puedan formar parte de una **clase base procesador de audio**, de la cual luego se puedan derivar clases como, **ecualizador**, **compresión**, **reverberación**, **delay**. Si la breve introducción sobre C++ ha sido clara no debería ser complicado entender la relación de herencia entre ambas clases, y la importancia de esta tesis.

2.2. API estándar de audio para C++

Es una interfaz de bajo nivel, desarrollada en C++, y su función principal es permitir al usuario interactuar con la tarjeta de sonido de la computadora, además de proveer funciones básicas para el procesamiento de audio. Es relativamente nueva, y muy lejos de estar terminada.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

2.2.1 Breve historia

Fue desarrollada por Guy Somberg, Guy Davidson y Timur Doumler y presentada en enero de 2019, argumentando que hoy en día cualquier producto tecnológico incluye alguna salida de audio, y otros hasta una entrada, aun así la librería estándar de C++ no cuenta con funciones que manejen el audio en bajo nivel, lo que obliga a los programadores a usar API específicas de ciertas plataformas, o usar middleware de otras empresas (FMOD) para realizar funciones de conversión entre plataformas (Windows, Mac, Linux). (Doumler, T., Davidson G. & Somberg G., 2019)

Con esto buscaban estandarizar ciertas prácticas recurrentes de manera que pudieran resultar portables entre diferentes sistemas operativos y facilitar la tarea para las personas que desean aprender de la programación de audio

2.2.2 Lenguaje de bajo nivel

Es un tipo de lenguaje de programación que se encuentra solo un nivel por encima del lenguaje de máquina y permite un trato directo con el hardware, en este caso se usa para pedir información a la computadora sobre las tarjetas de audio disponibles, interna o interfaz externa, sobre la cantidad de canales que maneja, (mono, estéreo, surround), a cuál frecuencia de muestreo trabaja (44.1 KHz, 48 KHz, 88.2 KHz), a qué profundidad de bits (16, 24, 32). Además de proveer acceso a la información de audio en sí misma, esta es representada como un arreglo bidimensional, compuestos por muestras, en la cual una posición **N** del arreglo, contiene la amplitud de dicha muestra en una señal de audio cuantizada.

2.2.3 Alcance y target

Esta librería solo busca proveer al programador con funciones básicas y fáciles de usar para realizar las tareas más básicas relacionadas al audio, como reproducir y grabar, que pueden ser usadas en diferentes hardware (Mac, Windows, Linux), con el fin de ser implementadas en cualquier microcontrolador, teléfono celular, tableta o aparato electrónico extravagante.

Estas funciones son:

- Descubrir que entradas y salidas de audio están disponibles en el sistema
- Recibir notificaciones cuando la configuración anterior cambie
- Pedir y configurar los parámetros básicos de un dispositivo de audio
- Proveer una estructura de buffers de audio que funcione en diferentes plataformas
- Realizar manejo de entrada y salida de audio leyendo o escribiendo estos buffers desde o hacia un dispositivo de audio

Esto permite que cualquier estudiante de audio o persona interesada en programación de bajo nivel, se pueda beneficiar del uso de esta librería.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

2.2.4 Conclusión

Esta librería se encarga de resolver problemas más específicos, que surgirán en una etapa más avanzada del desarrollo de plug ins de audio, encontrándose un nivel de programación por debajo de las funciones a las que esta tesis hará referencia, que en pocas palabras procesarán la señal obtenida a través del uso de las funciones que esta librería de audio ofrece. Como se mencionó en la sección dedicada a C++, **“cuando se programa en C++, generalmente se utilizan los siguientes bloques de construcción: clases y funciones de la biblioteca estándar de C++, clases y funciones creadas por uno mismo, y clases y funciones populares de diversos proveedores adicionales.”(Como programar en C++, 2003, p.10) Estas funciones pertenecen al primer tipo y serán una pieza fundamental para el desarrollo de plug ins pero de ninguna utilidad en esta tesis.**

2.3. Base teórica.

2.3.1 Sumatorias

Es una notación matemática que se usa para simplificar la representación de una suma de números que aumenta de manera periódica una determinada cantidad de veces. Para esta notación se usa la letra griega sigma (Σ).⁹

⁹ (n.d.). Series | Álgebra II | Matemáticas | Khan Academy. Se recuperó el noviembre 26, 2019 de <https://es.khanacademy.org/math/algebra2/sequences-and-series>



ANIMACIÓN



AUDIO



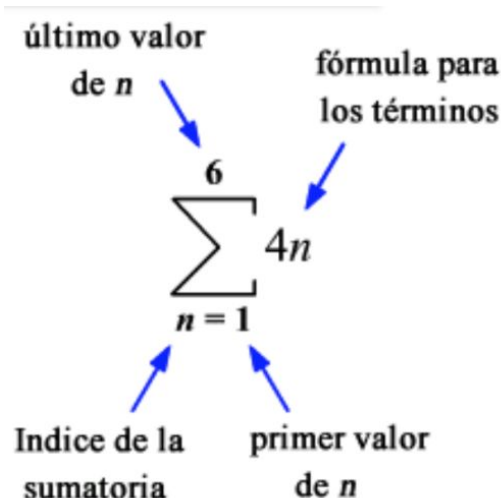
VIDEOJUEGOS



CINE



MUSIC BUSINESS



$$\sum_{n=1}^6 4n$$

Fig 1.4 Letra griega sigma que en el campo de la matemática representa la sumatoria

En la figura, se representa la siguiente suma: $(4 \times 1) + (4 \times 2) + (4 \times 3) + (4 \times 4) + (4 \times 5) + (4 \times 6)$, cuyo resultado es 96, pero de una manera mucho más simple, esta notación resulta útil a medida que el último valor de n se hace cada vez más grande, recordando que hay veces que dicha suma puede tender hacia el infinito. Este tipo de operaciones son relativamente sencillas para la computadora, mientras que para una persona puede requerir más tiempo.

2.3.2 Funciones trigonométricas

2.3.2.1 Radianes

Si trazamos un círculo cuyo radio vale r , y luego trazamos un ángulo θ , que se genera a subtender el radio a lo largo de la circunferencia, podemos definir esta angulo θ como un radian, de ahí que su nombre se derive de la palabra radio. Esta unidad nos puede servir como equivalencia al sistema que usamos para medir grados que va desde 0 grados a 360. Según esta equivalencia, 360 grados sería igual a 2π *radianes, es decir, una vuelta entera al círculo, 180 grados, es decir media vuelta sería igual a π *radianes, estas son las equivalencias más importantes entre estos dos sistemas.¹⁰

¹⁰ (n.d.). Trigonometría | Álgebra II | Matemáticas | Khan Academy. Se recuperó el noviembre 26, 2019 de <https://es.khanacademy.org/math/algebra2/trig-functions>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

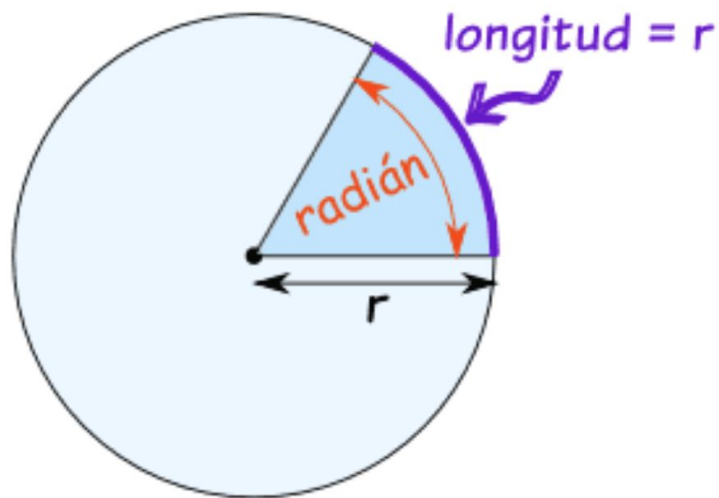


Fig 1.5 Descripción de un radian

2.3.2.2 Círculo unitario

Si trazamos un círculo de radio 1, sobre un plano de coordenadas cartesianas, tenemos el denominado círculo unitario.

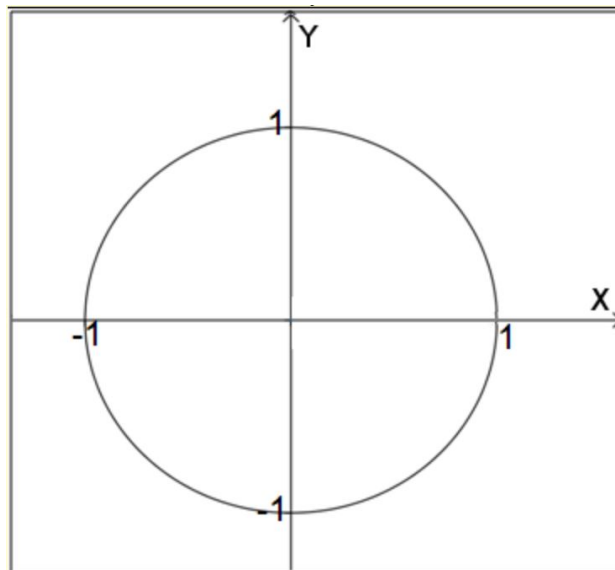


Fig 1.6 Círculo unitario

La convención usada para trazar ángulos en dicho círculo, es la siguiente:

-Ángulos positivos: los que se generan en sentido antihorario



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

-Ángulos negativos: los que se generan en sentido horario.

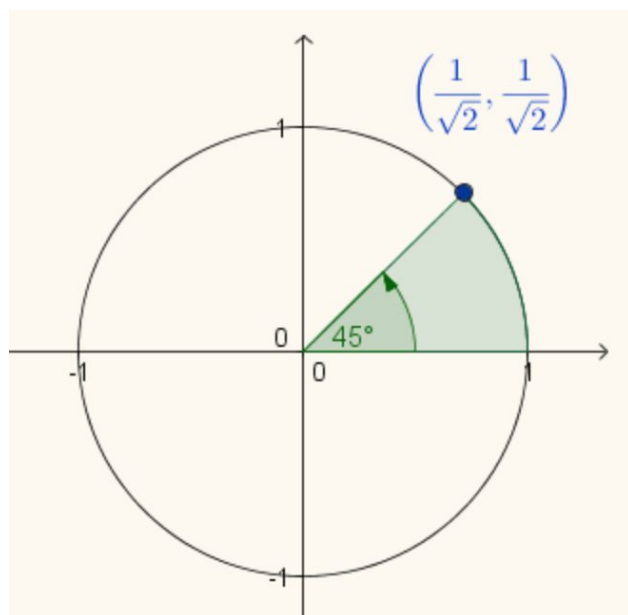


Fig 1.7 Representación de un ángulo de 45 grados sobre el círculo unitario

Ahora si usáramos dicho ángulo para trazar un triángulo rectángulo, podríamos definir el valor de la hipotenusa como el valor del radio, en este caso valdría 1. El valor de los catetos sería el valor de la coordenada **y** y **x**.¹¹

¹¹ (2013, mayo 31). El círculo unitario (video) | Trigonometría | Khan Academy. Se recuperó el noviembre 26, 2019 de <https://es.khanacademy.org/math/algebra2/trig-functions/unit-circle-definition-of-trig-functions-alg2/v/unit-circle-definition-of-trig-functions-1>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

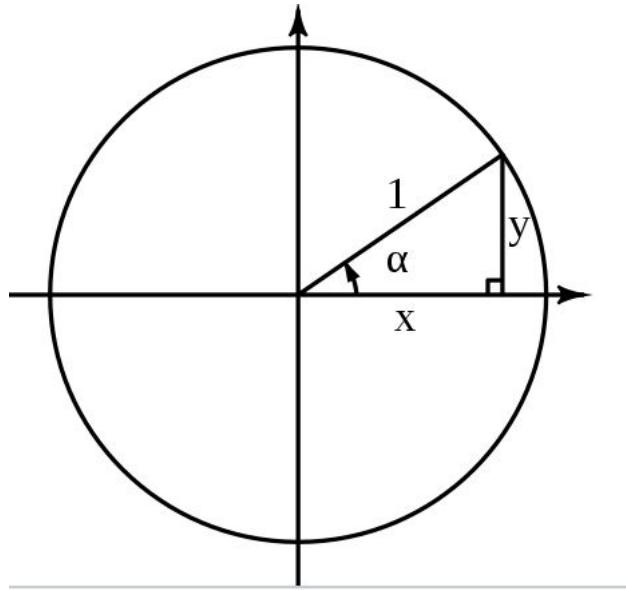


Fig 1.8 Triángulo rectángulo sobre círculo unitario

El círculo unitario es un gran recurso para entender las identidades trigonométricas como seno, coseno y tangente.

2.3.2.3 Razones trigonométricas

Si se divide un lado de un triángulo rectángulo entre el lado más largo (Hipotenusa), el resultado sería básicamente el mismo independientemente del tamaño del triángulo, siempre y cuando la relación de sus ángulos se mantenga igual, a esta relación entre los catetos y la hipotenusa se definen como seno y coseno, y se cumple para cualquier triángulo rectángulo, y es dependiente del ángulo y no del tamaño del cateto y de la hipotenusa.

La hipotenusa siempre va a ser el lado más largo de triángulo mientras que el cateto opuesto y el adyacente dependen del ángulo que se analice, que pueden ser uno de los dos diferentes del ángulo recto.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

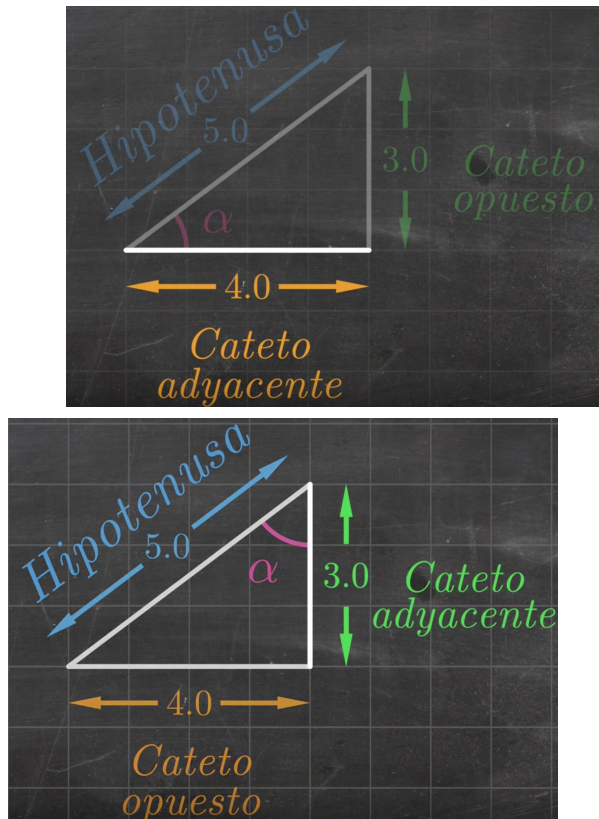


Fig 1.9 El cateto adyacente y opuesto de un triángulo varía dependiendo del ángulo que se tome en cuenta

Si se divide el cateto adyacente, entre la hipotenusa, el valor de su relación será siempre el mismo y se le denomina **coseno α (alfa)**, independientemente de su tamaño siempre y cuando se mantengan el valor de sus ángulos.

Si se divide el cateto opuesto, entre la hipotenusa, el valor de su relación se le denomina **seno**.

Ahora si realizamos estos cálculos en el círculo unitario explicado anteriormente, donde la hipotenusa vale 1, se obtiene que el coseno es igual al cateto adyacente y que el seno es igual al cateto opuesto.¹²

$$\sin \theta = \frac{y}{r} = \frac{y}{1} = y$$

$$\cos \theta = \frac{x}{r} = \frac{x}{1} = x$$

¹² (n.d.). The unit circle definition of sine, cosine, and tangent | Khan Se recuperó el noviembre 26, 2019 de <https://www.khanacademy.org/math/trigonometry/unit-circle-trig-func>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Fig 1.10 Identidades trigonométricas deducidas con el círculo unitario

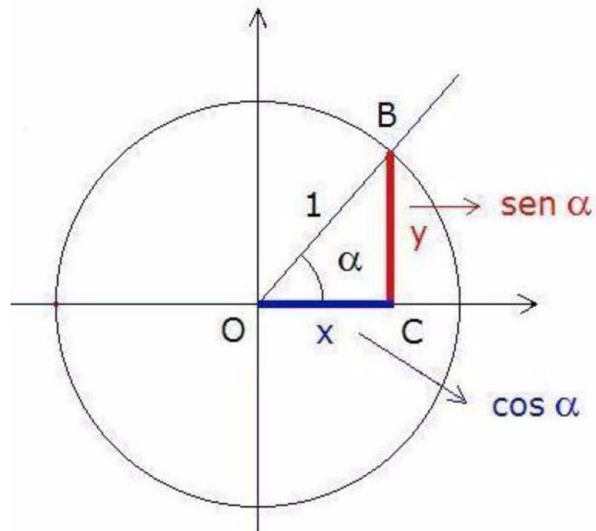


Fig 1.11 Identidades trigonométricas deducidas sobre el círculo unitario

Esto es importante porque ahora podríamos decir que las coordenadas de cualquier punto en el círculo pueden ser escritas de la forma $(\cos \alpha, \sin \alpha)$.

2.4. ADC y DAC

2.4.1 Cuantización

El sonido en el mundo real es una señal continua, esto quiere decir que su voltaje, al entrar una interface de audio luego de ser preamplificada, con el fin de que las computadoras la puedan manipular, es necesario que a través de un proceso denominado cuantización, un convertidor análogo-digital (ADC, Analog Digital Converter en inglés), las convierte en señales discretas.¹³

¹³ (n.d.). A Standard Audio API for C++: Motivation, Scope ... - open-std. Se recuperó el noviembre 26, 2019 de <http://93.90.116.65/JTC1/SC22/WG21/docs/papers/2019/p1386r0.pdf>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

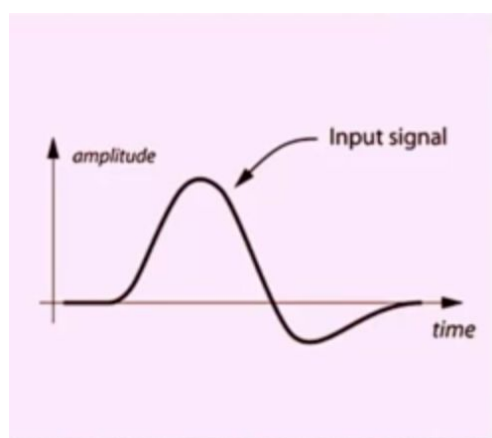


Fig 1.12 Señal continua

Al proceso inverso, se le conoce como conversión digital análogo, y permite traducir señales discretas dentro de una computadora a señales continua en el mundo real, es decir, sonido.

Este proceso dentro del ADC, se lleva a cabo en dos módulos, el S/H y el cuantizador. El S/H, Sample & Hold, mantiene el voltaje constante mientras la señal está siendo cuantizada, y solo cambia en intervalos periódicos de tiempo. Las variaciones de voltaje que ocurran en la señal entre esos periodos de tiempo, es totalmente ignorada por el cuantizador, de esta manera se convierte la variable independiente, es decir, el tiempo, de su forma continua a su forma discreta. (Doumler, T., Davidson G. & Somberg G.. 2019)

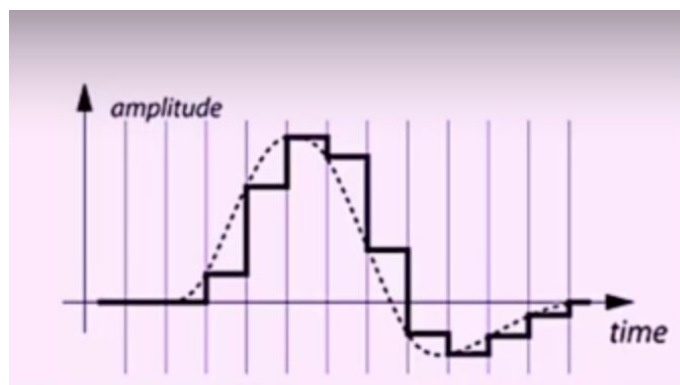


Fig 1.13 Sample & Hold

Luego el cuantizador redondea la señal, de por ejemplo 0.234764 V, a valores predeterminados con los que la computadora puede trabajar, digamos por ejemplo, 0.235V, luego convierte este valor a número binario y lo almacena, es importante mencionar este redondeo porque implica que señales cuyo voltaje sea 0.2345 V, 0.2346 V, 0.2347V, 0.2348V, 0.2349 V,



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

serán ahora iguales a una señal de 0.2345 V, lo que establece un límite de resolución de esta señal del mundo real en el mundo computacional.

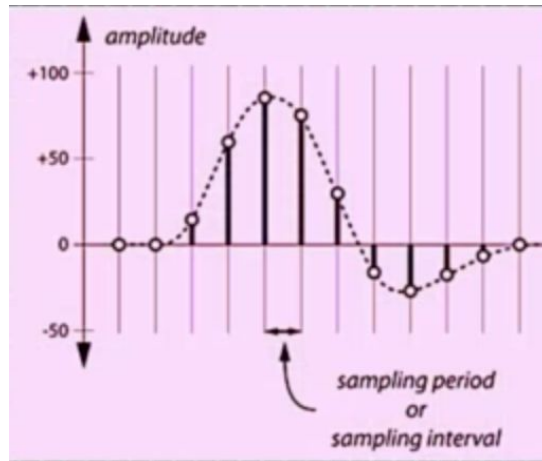


Fig 1.14 Cuantización

2.4.2 Teorema de Nyquist

En el mundo del audio se conoce como sampling, a la habilidad de reconstruir señales análogas basándonos solo en sus muestras o sampler

El teorema de Nyquist o teorema de sampleo, dicta que con el fin de reproducir de manera correcta dicha señal no debe contener frecuencias por encima de la mitad de la frecuencia de muestreo (Sample rate), en este caso el sample rate sería la cantidad de veces que el S/H cambia su voltaje en un segundo, en audio se usan valores fijos como 44.1KHz, 48 KHz, 88.2 KHz, y mientras más alto el sample rate más fielmente podremos reproducir una señal de audio. Esto quiere decir que la frecuencia más alta que podemos reproducir si el sample rate es de 44.1KHz, sería de 22,05 KHz.

Se puede notar que la elección del sample rate no fue aleatoria, ya que permite reproducir señales de hasta 2 KHz por encima del rango de escucha del ser humano (20Hz - 20KHz). Si hubiera una frecuencia por encima de 22.05 KHz en nuestra señal, se generaría un **alias**, esto ocurre cuando una señal sinusoidal asume otra frecuencia diferente a la que le pertenece ante la computadora, así como una persona asume el nombre de otra. Para evitar que esto ocurra es necesario pasar nuestra señal a través de un filtro pasa bajas que elimina todo el contenido por encima de la mitad del valor de la frecuencia de muestreo.

14

¹⁴ (n.d.). The Sampling Theorem - DSP Guide. Se recuperó el noviembre 26, 2019 de <http://www.dspguide.com/ch3/2.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

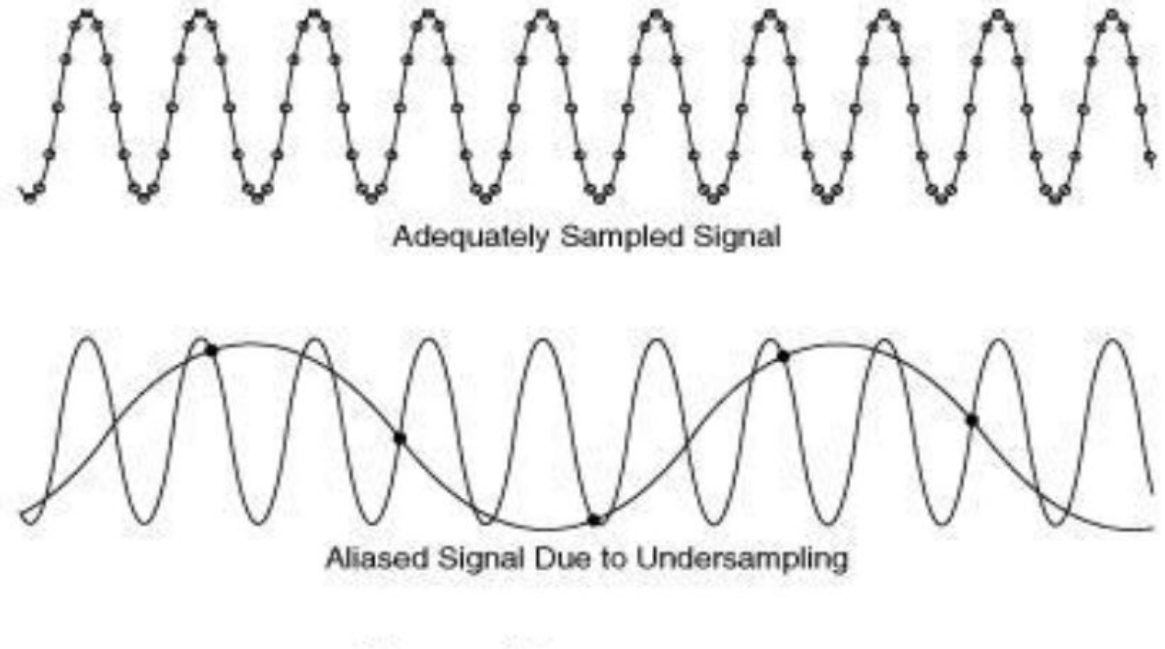


Fig 1.15 Aliasing

Capítulo 3: Desarrollo de algoritmos

3.1 Estadística de señales

3.1.1 El promedio (Mean)

3.1.1.1 Definición

El procedimiento para calcular el promedio de una señal es el siguiente:

- 1-. Se suman todos los puntos pertenecientes a la señal que se quiere medir.
- 2-. Se divide dicha suma entre el número de puntos que conforman la señal que se quiere medir.

En lenguaje matemático esto se puede expresar de la siguiente manera¹⁵:

¹⁵ (n.d.). Mean and Standard Deviation - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch2/2.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

$$\mu = \frac{1}{N} \sum_{i=0}^{N-1} x_i$$

fig 3.1 El promedio de una señal en lenguaje matemático

La media se representa con la letra griega mu(μ), el número de muestras que contiene la señal se representa con N, y x es el valor de un punto en una posición i , que puede ir desde 0 hasta N -1, es decir cualquier punto de nuestra señal.

3.1.1.2 Algoritmo básico

El algoritmo básico para calcular el promedio de una señal en C++ sería el siguiente:

```

11 double calc_signal_mean (double* sig_src_arr, int sig_lenght){
12     double mean = 0.0;
13
14     for( int i = 0; i < sig_lenght; i++)
15     {
16         mean += sig_src_arr [i];
17     }
18
19     mean = mean/ (double)sig_lenght;
20
21     return mean;
22 }

```

Fig 3.2 Función que calcula el promedio de una señal

Esta función cuenta con un ciclo **for** que itera a través de todo un arreglo de tipo double y acumula cada valor en otra variable llamada **mean**, que luego se divide su valor acumulado entre la cantidad de valores, para al final ser devuelta al usuario.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

3.1.1.3 Algoritmo avanzado

Para hacer este algoritmo más eficiente se puede hacer uso del método **Loop Unrolling**¹⁶. Este método consiste en evitar lo más posible, las verificaciones que tiene que realizar una computadora para determinar si una condición se cumplió y permite así realizar operaciones de una manera más rápida.

```

20     blkCnt = Blocksize / 4;      //Se divide el arreglo en cuatro partes
21
22     while (blkCnt>0) {
23
24         in1 = *sig_src_arr++;      //in1 = sig_src_arr [1]
25         in2 = *sig_src_arr++;      //in2 = sig_src_arr [2]
26         in3 = *sig_src_arr++;      //in3 = sig_src_arr [3]
27         in4 = *sig_src_arr++;      //in4 = sig_src_arr [4]
28
29         sum += in1;
30         sum += in2;
31         sum += in3;
32         sum += in4;
33
34         blkCnt--;
35     }
36

```

fig 3.3 Método Loop Unrolling usado para calcular el promedio de una señal

La variable **blkCnt**, representa el tamaño del arreglo dividido en cuatro partes, y las variables in1, in2, in3, in4 almacenan los valores del arreglo, de a cuatro valores mientras que **blkCnt**, sea mayor que 0. Este algoritmo aunque más difícil de entender es mucho más eficiente que el primero.

```

38     blkCnt = Blocksize%4;
39
40     while (blkCnt > 0) {
41         sum += *sig_src_arr++;
42         blkCnt--;
43     }
44

```

fig. 3.4 Manejo de caso en que tamaño del arreglo no sea múltiplo de 4

¹⁶ (2017, mayo 3). How to unroll a short loop in C++ - Stack Overflow. Se recuperó el diciembre 2, 2019 de <https://stackoverflow.com/questions/2382137/how-to-unroll-a-short-loop-in-c>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Esta parte del algoritmo sucederá si el tamaño de nuestro arreglo no es múltiplo de 4, lo que significa que podrían quedar de 1 a 3 valores del arreglo sin ser procesados. Esta parte del algoritmo no debería suceder nunca, pero se coloca de manera preventiva.

Este algoritmo realiza dicha operación de una manera más eficiente a cambio de ocupar más espacio en la computadora, podemos notar a simple vista que nuestro código es más largo, y que usamos más variables que el algoritmo básico.

Si comparamos el algoritmo básico con el avanzado, podemos notar que para un arreglo de 16 elementos, la verificación que tiene que realizar la computadora para deducir si el ciclo for o while, aún resulta verdadero, se realiza 16 veces para el básico y solo 4 para el avanzado, estas verificaciones toman tiempo, lo que nos permite deducir que nuestro algoritmo avanzado si es más eficiente.

3.1.2 Standard Deviation

3.1.2.1 Definición

Luego de obtener el valor promedio de la señal que estamos midiendo, otro dato que nos puede ayudar a entender su carácter es saber cuánto difiere cada muestra de dicho promedio, y más aún saber cual es el promedio de esa diferencia, y si además realizamos este procedimiento con la potencia de cada diferencia en vez de con su amplitud, representaríamos más fielmente cómo se suman las señales en el mundo real, para finalizar tendríamos que sacar la raíz cuadrada de dicho valor para compensar la elevación de potencia. A este resultado se le llama **la desviación estándar** y su fórmula matemática es la siguiente¹⁷:

$$\sigma^2 = \frac{1}{N-1} \sum_{i=0}^{N-1} (x_i - \mu)^2$$

fig 3.5 Fórmula de la desviación estándar.

¹⁷ (n.d.). Mean and Standard Deviation - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch2/2.htm>



3.1.2.2 Algoritmo básico

El algoritmo básico para calcular la desviación estándar en C++ sería el siguiente:

```
25
26 double calc_signal_std (double* sig_src_arr, double sig_mean, int sig_lenght)
27 {
28     double variance = 0.0;
29     double std = 0.0;
30
31     for (int i = 0; i < sig_lenght; i++) {
32         variance += pow((sig_src_arr[i] - sig_mean), 2);
33     }
34
35     variance /= (sig_lenght - 1);
36
37     std = sqrt(variance);
38
39     return std;
40 }
41
```

fig. 3.6 Función que calcula la desviación estándar de una señal

Esta función requiere que el usuario suministre, además del arreglo que se va a analizar y su tamaño, el promedio de dicha señal, este se obtiene usando la función descrita en el apartado anterior.

3.1.2.3 Algoritmo avanzado

Para hacer este algoritmo más eficiente se hará uso del método **Loop Unrolling**, que se usó previamente en el algoritmo pasado.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

```

32     blkCnt = Blocksize/4;
33
34     while (blkCnt>0) {
35
36         in = *pSig_src_arr++;
37         sum += in;
38         sumofSquare += in * in;
39
40         in = *pSig_src_arr++;
41         sum += in;
42         sumofSquare += in * in;
43
44         in = *pSig_src_arr++;
45         sum += in;
46         sumofSquare += in * in;
47
48         in = *pSig_src_arr++;
49         sum += in;
50         sumofSquare += in * in;
51
52         blkCnt--;
53     }
54

```

fig 3.7 Método Loop Unrolling aplicado en el algoritmo avanzado

Además se hará uso de una equivalencia matemática de la fórmula de la figura 3.5.

$$\sigma^2 = \frac{\sum X^2}{N} - \mu^2$$

fig 3.8 Otra fórmula matemática para expresar la desviación estándar.

El uso del método de Loop Unrolling y el uso de esta equivalencia matemática hace que nuestro algoritmo avanzado sea significativamente más rápido que nuestro algoritmo básico.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

3.1.3 RMS

3.1.3.1 Definición

La fórmula matemática de rms (root mean square), es básicamente sumar todos los cuadrados de la señal, y obtener la raíz de dicho resultado. Matemáticamente se puede expresar de la siguiente manera¹⁸:

$$x_{rms} = \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}$$

fig 3.9 Fórmula matemática para expresar el rms

3.1.3.2 Algoritmo básico

El algoritmo básico para calcular el rms de una señal en lenguaje C++ sería el siguiente:

```

42
43 double calc_signal_rms (double* sig_src_arr, int sig_lenght)
44 {
45     double rms = 0.0;
46
47     for (int i = 0; i < sig_lenght; i++) {
48
49         rms += sig_src_arr[i] * sig_src_arr[i];
50
51     }
52
53     rms = sqrt(rms/(double)sig_lenght);
54
55
56     return rms;
57 }
58

```

fig 3.10 Función que calcula el rms de una señal

A través de un ciclo **for** una variable de tipo double de nombre **rms** almacena el cuadrado de una posición en el arreglo **sig_src_arr** iterando el valor de **i**, hasta que sea igual al valor del **sig_lenght**. Luego en la línea 53 **rms** es igual a la raíz de su actual valor dividido entre el tamaño del arreglo.

¹⁸ (n.d.). Mean and Standard Deviation - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch2/2.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

3.1.3.3 Algoritmo avanzado

Para realizar un algoritmo más eficiente se hizo uso del método **Loop Unrolling**, como ya se explicó este método busca reducir la cantidad de veces que la computadora realiza verificaciones para saber cuando salir de un ciclo iterativo.

```

21
22     blkCnt = Blocksize/4;           //Se divide el tamaño del arreglo entre 4
23
24     while (blkCnt > 0) {
25         in = *pSig_src_arr++;       //pSig_src_arr [0]
26         sum += in * in;             //pSig_src_arr [0] al cuadrado
27
28         in = *pSig_src_arr++;       //pSig_src_arr [1]
29         sum += in * in;             //pSig_src_arr [1] al cuadrado
30
31         in = *pSig_src_arr++;       //pSig_src_arr [2]
32         sum += in * in;             //pSig_src_arr [2] al cuadrado
33
34         in = *pSig_src_arr++;       //pSig_src_arr [3]
35         sum += in * in;             //pSig_src_arr [3] al cuadrado
36
37         blkCnt--;
38     }
39
40

```

fig 3.11 Método Loop Unrolling aplicado en el cálculo de rms

Gracias a este método se pudo desarrollar un algoritmo que es más eficiente a simple vista, aunque pueda resultar muy difícil de leer y dar mantenimiento, además debe recordarse que estamos obteniendo rapidez a cambio de espacio, ya que este tipo de algoritmos ocupan más espacio en disco.

3.1.4 Signal to Noise Ratio

3.1.4.1 Definición

Para algunas aplicaciones se puede entender el promedio como lo que se está midiendo y la desviación estándar representa ruido, o interferencia. Lo realmente útil de ambos términos es como se relacionan, esta relación se define como **Signal to noise ratio** que se obtiene dividiendo el promedio entre la desviación estándar. Otra manera de calcular la relación es a través del coeficiente de variación (CV), esto se obtiene dividiendo la desviación estándar entre el promedio, y luego multiplicando el resultado por 100.¹⁹

¹⁹ (n.d.). Mean and Standard Deviation - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch2/2.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

$$SNR = \frac{\mu}{\sigma} \quad CV = \frac{\sigma}{\mu} \times 100$$

fig 3.12 Fórmula matemática del Signal to noise Ratio y CV

3.1.4.2 Algoritmo

El algoritmo para calcular tanto el noise ratio como el coeficiente de variación requiere que el usuario suministre el promedio y la desviación estándar, estas se obtienen a través de los algoritmos presentados en los apartados anteriores

```

59
60 double calc_signal_noise(double* mean, double* std_dev)
61 {
62     double signal_noise_ratio = *mean/ *std_dev;
63
64     return signal_noise_ratio;
65 }
66
67
68 double calc_signal_CV(double* mean, double* std_dev)
69 {
70     double coefficient_variation = (*std_dev / *mean) * 100;
71
72     return coefficient_variation;
73 }
74

```

fig 3.13 Algoritmo de signal to noise ratio y del coeficiente de variación

3.2 Convolución

3.2.1 Definición

La convolución es una operación matemática al igual que la multiplicación, potenciación, suma etc. Así como la suma toma dos números y entrega un tercero, la convolución toma dos señales y entrega una tercera, cuando se trabaja con



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

sistemas lineales la convolución nos sirve, entre otras cosas, para describir la relación entre tres señales, a saber: la señal de entrada, el impulso unitario, y la señal de salida.

$$x[n] * h[n] = y[n]$$

fig 3.14 fórmula matemática de la convolución

El signo que se usa para indicar la convolución es un asterisco (*), sin embargo, en la gran mayoría, si no es que todos los lenguajes de programación cuando se usa un asterisco, nos referimos a la multiplicación, no convolución.

Si no conociéramos $h[n]$, podríamos obtenerlo, asignando un valor arbitrario a $x[n]$, digamos un valor que sea el equivalente al uno en la multiplicación o al cero en la suma, y realizar la convolución con $h[n]$, nuestro resultado sería un $y[n]$ igual a $h[n]$.²⁰

A través de esta operación podemos conocer el comportamiento de cualquier sistema en su totalidad, por eso es uno de los conceptos más importantes dentro del procesamiento digital de señales y resulta muy útil diseñar nuestro propio algoritmo para dicha tarea, que entendamos cómo funciona y podamos ser capaces de hacerle mantenimiento cuando sea necesario.

3.2.2 Algoritmo básico

El algoritmo de convolución en lenguaje de C++ es el siguiente:

```

26
27     //algoritmo de convolución
28     for (i = 0; i < src_sig_lght; i++) {
29         for (j = 0; j < imp_resp_lght; j++) {
30             sig_dest_arr[i + j] += sig_src_arr[i] * imp_resp_arr[j];
31         }
32     }
33

```

fig 3.15 Algoritmo de convolución

²⁰ (n.d.). Convolution - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch6/2.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

El algoritmo básico hace uso de dos estructuras **for** para iterar un punto de un arreglo A a lo largo de todo un arreglo B, y almacenarlo en una posición del arreglo C.

3.2.3 Algoritmo avanzado

3.2.3.1 Fórmula matemática

Lo interesante de la matemática es que solo invirtiendo el orden en que escribimos la fórmula, sin alterar en ningún momento la igualdad, podemos deducir cosas sobre la convolución.

$$y[n] = x[n] * h[n]$$

fig 3.16 Fórmula matemática de la convolución desde el punto de vista de la salida.

La fórmula de la figura 3.16 nos indica como calcular cada muestra de la señal de salida de manera independiente, esta es la segunda forma de entender la convolución, ver cómo cada elemento de la señal de entrada contribuye con un solo elemento de la señal de salida. Esto es útil no solo desde un punto de vista matemático sino también práctico.

$$y[i] = \sum_{j=0}^{M-1} h[j] x[i-j]$$

fig 3.17 Fórmula de convolución en lenguaje matemático. **$h[]$** es un arreglo de tamaño M.

Si iteramos **i** desde **0** hasta **n** , siendo **n** el tamaño del arreglo obtenemos la señal de salida. El algoritmo presentado en la siguiente sección hace uso de esta fórmula y no del de la figura 3.14.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

3.2.3.2 Algoritmo en C++

El siguiente algoritmo hace uso del método **Loop Unrolling**, usado en los algoritmos anteriores. No se pretende explicar detalladamente cómo funciona cada parte del algoritmo pero se hará mención de cuáles secciones del código hacen referencia a ciertas partes de la fórmula matemática de la fig 3.17.

```

55 while(Blocksize1>0)
56 {
57     sum = 0.0;
58
59     k = count/4;
60
61     while (k > 0 ) {
62         sum += *px++ * *py--;
63         sum += *px++ * *py--;
64         sum += *px++ * *py--;
65         sum += *px++ * *py--;
66
67         k--;
68     }
69
70     //Con el siguiente ciclo while nos encargamos del caso que count no se multiplo de 4
71
72     k = count%4;
73
74     while (k>0) {
75         sum += *px++ * *py--;
76         k--;
77     }
78
79     *pOut++ = sum;           //Output Signal      // Itera y[0], y[1], y[2]...
80
81     py = pIn2 + count;      // Input Signal      //Itera a[0], a[1], a[2]....
82     px = pIn1;              // Impulse response //Reposiciona px al principio del arreglo
83
84     count++;
85     Blocksize1--;
86 }
87

```

$$y[i] = \sum_{j=0}^{M-1} h[j] x[i-j]$$

fig 3.18 Algoritmo avanzado y fórmula matemática de convolución

El código de colores que se usará de ahora en adelante a lo largo de esta tesis para representar la relación entre los códigos y las fórmulas matemáticas ha sido tomado



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

de la tesis de Rafael Arias "Comparación de algoritmos rápidos de la transformada de Fourier enfocados en un diseño económico de análisis de respuesta de frecuencia de un cuarto.", que ha servido de inspiración para esta tesis.

3.3 Transformada Discreta de Fourier (DFT)

3.3.1 Definición

Una transformada es sencillamente un procedimiento mediante el cual convertimos un valor en otro, así como $y = 2x + 1$, es una función que para un determinado valor x entrega un determinado valor y , a través de una serie de pasos en los cuales primero multiplicamos x por 2 y luego le **sumamos 1**, mediante una transformada podemos tomar una señal de 100 muestras y convertirlas en otra señal de 100 muestras, depende qué transformada usemos se realizarán diferentes procedimientos.

El procedimiento por el cual Fourier pudo colocar su nombre a una transformada, nos permite representar una señal periódica y continua, como una suma de ciertas frecuencias sinusoidales. La utilidad de descomponer una señal en muchas senoidales, radica en que es más sencillo manipular una senoidal.

Aunque podemos dividir la transformada de Fourier en cuatro categorías dependiendo el tipo de señal que estemos manipulando (Periódica o no periódica, continua o discreta), a nosotros sólo nos interesa la transformada que nos permite manipular señales **periódicas y discretas**, a saber la **Transformada Discreta de Fourier**.

Una transformada de Fourier toma una señal, representada como un cambio de amplitud a lo largo del tiempo, medido en N número de muestras que ocurren a intervalos iguales de tiempo y la transforma en dos grupos cada uno con $N/2 + 1$ señales, un grupo contiene las amplitudes de los cosenos, y el otro las amplitudes de los senos.

Así como el espectro frecuencial puede ser representado en un diagrama donde el **eje x** representa la frecuencia en Hertz y el **eje y** la amplitud en Db's, en el campo del procesamiento digital de señales la información del **eje x** también puede ser representada en muestras, como números enteros que van de 0 a $N/2$, donde N es el número de muestras de la señal original, esta manera es más intuitiva para los programadores. Otra manera el eje x va desde 0 a π , siendo más fácil de entender en el área de la matemática, si recordamos que hay 2π radianes en un



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

círculo y que el resultado final de una DFT es un arreglo con senos y otro con cosenos.

Aunque la forma de representar la señal resulta muy intuitiva en el área del audio, cabe destacar que a diferencia de ir de 20 Hz a 20 KHz, el espectro de frecuencia irá de 0 a la mitad de la tasa de sampleo que estemos usando, por el Teorema de Nyquist.²¹

$$ReX[k] = \sum_{i=0}^{N-1} x[i] \cos(2\pi k i / N)$$

$$ImX[k] = - \sum_{i=0}^{N-1} x[i] \sin(2\pi k i / N)$$

fig 3.19 Fórmula matemática de la transformada discreta de Fourier

La transformada discreta de Fourier divide una señal de entrada en dos señales una parte **ReX** y otra **ImX**, cada una de tamaño **k** que es igual a la mitad de la señal original, se puede intuir que estos arreglos almacenan un valor relacionado a la frecuencia, que se calculan multiplicando cada valor de la señal original (**x[i]** yendo desde **0** a **N-1** donde **N** es el tamaño de la señal), por el coseno o seno y acumulando toda la suma en dicha posición de nuestro arreglo.

Puede resultar útil al lector ir a la sección 2.3 si desea entender un poco las expresiones matemáticas que aparecen en la figura 3.19 y entender porque nos interesa precisamente el coseno y el seno de una señal.

3.3.2 Algoritmo

La figura 3.20 muestra el código en lenguaje C++ de la fórmula matemática de la figura de 3.19. Se usaron colores para hacer más fácil al lector encontrar la relación entre el código y la matemática.

²¹ (n.d.). DFT Basis Functions - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch8/4.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS


```

29 // // DFT // //
30
31 for(k = 0; k < (sig_lenght / 2) + 1; k++) // Para cada frecuencia
32 {
33     for (i = 0; i < sig_lenght; i++) { //Calculamos la contribucion de todos los
34         sig_dest_rex_arr[k] = sig_dest_rex_arr[k] + pSrc_arr[i] * cos(2 * M_PI * k * i / sig_lenght); //puntos en el dominio del tiempo
35         sig_dest_imx_arr[k] = sig_dest_imx_arr[k] - pSrc_arr[i] * sin(2 * M_PI * k * i / sig_lenght);
36     }
37 }
38
39 }
40

```

$$\begin{aligned}
 \text{Re}X[k] &= \sum_{i=0}^{N-1} x[i] \cos(2\pi k i / N) \\
 \text{Im}X[k] &= - \sum_{i=0}^{N-1} x[i] \sin(2\pi k i / N)
 \end{aligned}$$

Fig 3.20 Algoritmo de la transformada de Fourier en C++

Este algoritmo resulta muy práctico para entender el concepto de la transformada de Fourier pero no es el más eficiente. En una sección posterior se desarrollará una versión más rápida de este algoritmo denominado FFT.

3.4 Transformada Discreta de Fourier Inversa (IDFT)

3.4.1 Definición

Al proceso descrito en la sección anterior se le denomina también **análisis**, y al proceso de sumar senoidales de diferentes frecuencias y amplitudes se le denomina **síntesis**, o Transformada Discreta de Fourier Inversa. En el área del audio se usa el término síntesis cuando se trabaja con sintetizadores modulares, y dicha síntesis no dista mucho de la aquí presentada.

$$x[i] = \sum_{k=0}^{N/2} \text{Re}\bar{X}[k] \cos(2\pi k i / N) + \sum_{k=0}^{N/2} \text{Im}\bar{X}[k] \sin(2\pi k i / N)$$

fig 3.21 Fórmula matemática de la Transformada Discreta de Fourier Inversa



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Hay que recordar que i es un entero que va desde 0 a $N-1$, donde N es el tamaño de nuestro arreglo, también que cada valor que toma i representa una muestra en el tiempo tomada a intervalos iguales, de una señal compleja.

La fórmula de la figura 3.21 nos permite calcular cada punto sumando las diferentes amplitudes de las senoidales que componen dicha señal.²²

Nótese que las variables de los arreglos que se usan no son precisamente ReX ni ImX , sino valores que se derivan de ellos de la siguiente manera

$$\begin{aligned} Re\bar{X}[k] &= \frac{ReX[k]}{N/2} & Re\bar{X}[0] &= \frac{ReX[0]}{N} \\ Im\bar{X}[k] &= -\frac{ImX[k]}{N/2} & Re\bar{X}[N/2] &= \frac{ReX[N/2]}{N} \end{aligned}$$

fig 3.21 Reconversión de los valores ReX e ImX , y manejo de dos casos especiales.

3.4.2 Algoritmo

La figura 3.22 muestra el código en lenguaje C++ de la fórmula matemática de la figura de 3.21. Se usaron colores para hacer más fácil al lector encontrar la relación entre el código y la matemática.

²² (n.d.). Synthesis, Calculating the Inverse DFT - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch8/5.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

```

34         //          IDFT          //
35
36
37     for (k= 0; k<idft_lenght/2; k++) {
38
39         for (i = 0; i < idft_lenght; i++) {
40
41             //          Manejo del caso especial donde          //
42             if (i== 0 || i== idft_lenght) {
43                 idft_out_arr [i] = idft_out_arr [i] + (sig_src_rex_arr[k] /(idft_lenght)) * cos(2 * M_PI * k * i/idft_lenght);
44                 idft_out_arr [i] = idft_out_arr [i] + (sig_src_imx_arr[k] /(idft_lenght)) * sin(2 * M_PI * k * i/idft_lenght);
45             }
46
47             idft_out_arr [i] = idft_out_arr [i] + (sig_src_rex_arr[k] /(idft_lenght/2)) * cos(2 * M_PI * k * i/idft_lenght);
48             idft_out_arr [i] = idft_out_arr [i] + (sig_src_imx_arr[k] /(idft_lenght/2)) * sin(2 * M_PI * k * i/idft_lenght);
49         }
50     }
51 }
52

```

$$x[i] = \sum_{k=0}^{N/2} \text{Re}\bar{X}[k] \cos(2\pi ki/N) + \sum_{k=0}^{N/2} \text{Im}\bar{X}[k] \sin(2\pi ki/N)$$

fig 3.22 Fórmula matemática de la Transformada Discreta de Fourier Inversa

3.5 Transformada Rápida de Fourier (FFT)

3.5.1 Definición

La Transformada rápida de Fourier, se divide en cuatro pasos. Primero se descompone una señal de un tamaño N, que se encuentra en función del tiempo, en N cantidad de señales todas compuestas por un solo punto. Luego, se calcula el espectro de frecuencia para estas N cantidad de señales. Por último, se sintetizan todas estas señales en un solo espectro de frecuencia.

El proceso de descomposición, para una señal de 16 muestras, sería primero una división entre dos señales cada una con 8 muestras, separándolas según pares o impares, luego volvemos a dividir cada señal de 8 muestras en dos señales, de 4 muestras, usando el mismo criterio. Estas 4 señales cada una de 4 muestras, se dividen una vez más, y ahora tenemos 8 señales de 2 muestras cada una, y una última vez nos deja con 16 señales cada una con una sola muestra. En total serán, Log2 (N) de divisiones.

El siguiente paso, calcular el espectro de frecuencia de cada una de las 16 señales, resulta muy fácil, no requiere que hagamos nada, solo recordar que la señal ya no se encuentra en función del tiempo, sino que ahora es un espectro de frecuencia.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

El último paso, es sumar todos estos espectros de frecuencia de manera inversa a como realizamos la descomposición, 16 espectros de frecuencia de 1 muestra cada uno se convierten en 8 de 2 muestras cada uno, estos 8 espectros de frecuencia luego se convierten 4 de 4 muestras cada uno, y así sucesivamente hasta obtener un solo espectro de frecuencia de 16 muestras.²³

El algoritmo de la figura 3.20, que representa la DFT consta de dos ciclos **for** anidados lo que da una cantidad de operaciones igual a N^2 . Por otro lado el algoritmo de la FFT, cuenta con $\text{Log2}(N) \times N$, $\text{Log2}(N)$ representa el número de divisiones ya mencionado y N , la cantidad de multiplicaciones que se realiza en cada una. El carácter exponencial de la DFT, nos permite intuir que la FFT en teoría si requiere menos de tiempo de computación. Además como se realizan menos operaciones, la cantidad de redondeos también disminuye esto nos permite tener más precisión lo que se refleja en menos ruido en nuestra señal .

Esto se pretende demostrar pasando una señal elegida arbitrariamente por ambos algoritmos (DFT y FFT) y luego, y usando el algoritmo de la sección 3.1.2 (standard deviation), para medir la diferencia de ruido entre cada una.

3.5.2 Algoritmo

El primer paso, mencionado anteriormente se lleva a cabo con este algoritmo

```

38 //Bit reversal sorting//
39 for ( int i = 1; i < Sig_L/2; i++)
40 {
41     double TR, TI; //acumulador temporal de la variable real
42     int a;
43
44     a = bit_Reversal(i, Sig_L); //a es el numero i invertido
45
46     TR = ReX[i]; //Acumula temporalmente la variable
47     TI = ImX[i]; //Acumula temporalmente la variable
48
49     ReX[i]= ReX[a]; //intercambio de posicion de la parte real
50     ReX[a] = TR; //intercambio de posicion de la parte real
51
52     ImX[i] = ImX[a]; //intercambio de posicion de la parte imaginaria
53     ImX[a] = TI; //intercambio de posicion de la parte imaginaria
54 }

```

fig 3.23 Código en C++ del primer paso de la FFT

A través de la inversión de bits de los valores que representan los números de muestras podemos reordenar las muestras de nuestra señal, y

²³ (n.d.). FFT Programs - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch12/3.htm>



llevar a cabo de una manera práctica una descomposición que realmente es teórica.

```

57 // FFT
58 for (int L = 1; L < m; L++) {
59     int LE = pow(2, L);
60     int LE2 = LE/2;
61     double ur = 1;
62     double ui = 0;
63     double sr = cos (Pi/ LE2);           //calculo del valor del coseno
64     double si = -sin(Pi/ LE2);          //calculo del valor del seno
65     double tr, ti;
66
67     for (int j = 1; j < LE2; j++) {
68         int jm1 = j - 1 ;
69         for (int i = jm1; i < NM1; i += LE) {
70             int ip = i + LE2;
71             tr = ReX[ip] * ur - ImX[ip] * ui; //Butterfly calculations
72             ti = ReX[ip] * ui - ImX[ip]*ur;   //Butterfly calculations
73
74             ReX[ip] = ReX[i] - tr;
75             ImX[ip] = ImX[i] - ti;
76
77             ReX[i] = ReX[i] + tr;
78             ImX[i] = ImX[i] + ti;
79         }
80
81         tr = ur;
82         ur = tr * sr - ui*si;
83         ui = tr * si + ui*sr;
84     }

```

fig 3.24 Código en C++ de la FFT

No se puede explicar como ocurre cada paso mencionado en la sección anterior en el código presentado en la figura, sin tener que entrar forzosamente en el campo de la matemática, lo que no es el fin de esta tesis. Lo que se puede mencionar es que dicho algoritmo reemplaza las señales de entrada que se encuentra en función del tiempo con la señales que contienen el espectro de frecuencia, usando el mismo contenedor. Evitando así tener que crear otro, lo que implicaría que el algoritmo ocuparía más espacio.

3.6 Diseño de filtros

El diseño de filtros digitales es uno de los temas más importantes del procesamiento digital de señales. Tienen dos usos principales: separar señales, cuando tenemos una señal contaminada con interferencia, ruido o alguna otra señal. O restaurar señales, cuando tenemos una señal distorsionada de alguna manera, como en grabaciones realizadas con equipos de baja calidad.

Los filtros pueden ser o análogos, transistores y resistencia, o digitales, código en algún lenguaje de programación, los primeros son baratos, rápidos y poseen un amplio rango dinámico y frecuencial, los segundos por otro lado, son extremadamente superiores en cuanto su desempeño, donde se pueden alcanzar



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

transiciones muy abruptas de amplitud en distancias frecuenciales muy cortas(1 Hz), esta clase de resultados no se pueden alcanzar con filtros análogos.

La manera más sencilla de implementar un filtro, es haciendo convolución entre una señal de entrada y la respuesta de impulso de un filtro. Cuando la respuesta de impulso se usa con este propósito, se le denomina filtro kernel.

Cuando se habla de filtros es normal representarlos sobre un plano cartesiano cuyo **eje x** representa el tiempo, esto debido a que la mayoría de las veces, las señales se obtienen registrando valores de amplitud a intervalos iguales de tiempo. Cuando realizamos mediciones de un evento a lo largo del tiempo la señal que obtenemos nos da información del momento en que ocurrió, su duración y su desarrollo, cada muestra por separado nos da información sin tener que compararlo con otra muestra, si solo tuviéramos una muestra de la señal, aun así tendríamos información del evento, si bien no es la única manera, es la más común y por ende una de las que se maneja.

La otra manera de representarlos es sobre un plano cartesiano cuyo **eje x** representa el espectro frecuencial, esta manera no es tan intuitiva como la primera. Y se obtiene de medir la frecuencia, fase y amplitud de los movimientos periódicos que presentan los objetos en nuestro universo, y así obtener información del sistema que los genera. Este dominio se adapta perfectamente al audio, cuando tomamos muestras de un sonido, la frecuencia fundamental y sus armónicos nos proveen información de la masa y elasticidad del objeto que lo generó, una sola muestra no nos da ninguna información sobre la señal, la información está en la relación entre varios puntos en la señal.

Los filtros pasa altas, los elimina banda y los pasa banda, derivan todos del filtro pasa bajas, por eso solo se desarrollaran este tipo de filtros.

3.6.1 Moving Average Filter

El Moving Average filter es el filtro más común en el procesamiento digital de señales, en gran parte por lo fácil que es de entender y de usar. A pesar de su simplicidad funciona de una manera bastante óptima para tareas básicas como reducir el ruido.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j]$$

fig 3.25 Fórmula matemática del Moving Average Filter

Al diseñar esta clase de filtros, tenemos que determinar el número de puntos que queremos usar, este representaría nuestro valor **M**, ahora lo que haremos será calcular, cada valor de la señal de salida **i**, que será del mismo tamaño que la señal de entrada, sumando al valor de entrada los valores adyacentes, tantos como **M**, y luego dividiendo este valor entre **M**. Se puede observar que solo se realiza un cálculo de promedio²⁴.

$$y[80] = \frac{x[80] + x[81] + x[82] + x[83] + x[84]}{5}$$

fig 3.26 Cálculo del filtro para la muestra número 80

Si **M**, fuera igual a 5, y la fórmula para calcular **y[80]**, sería sumando **x[80]** hasta **x[84]**, y luego dividiendo este resultado en 5.

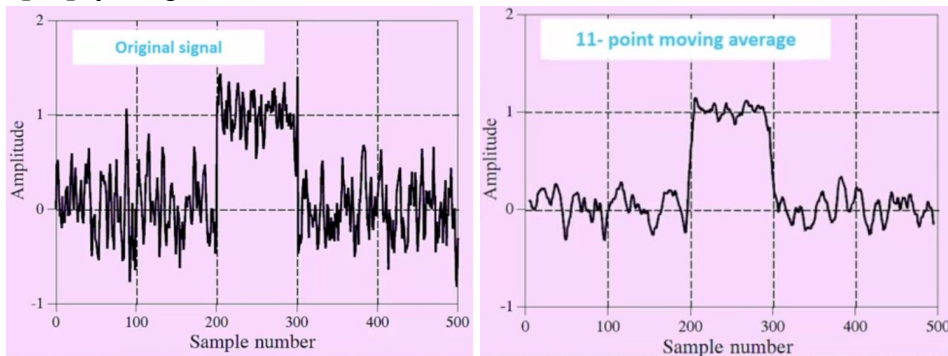


fig 3.27 Antes y después de usar un filtro de 11 puntos en una señal

²⁴ (n.d.). Implementation by Convolution - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch15/1.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

3.6.1.1 Algoritmo

El desarrollo de la fórmula matemática de dicho filtro en C++ es el siguiente:

```

13 void moving_average( double * pSig_src_arr, double *pSig_out_arr, int signal_lenght, int filter_pts)
14 {
15     int i, j; //contadores
16
17     for(i =(int) floor(filter_pts/2) ; i <(int) (signal_lenght - floor(filter_pts/2) - 1); i++ )
18     {
19         pSig_out_arr [ i ] = 0;           //inicializacion del arreglo de salida en 0
20
21         for ( j =(int) -floor(filter_pts/2); j < (int) floor(filter_pts/2); j++) {
22             pSig_out_arr [i] = pSig_out_arr [i]+ pSig_src_arr [ i + j];
23         }
24         pSig_out_arr [i] = pSig_out_arr [ i]/filter_pts;
25     }
26 }
27

```

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j]$$

fig 3.28 Código en C++ del Moving Average Filter

3.6.2 Recursive Moving Average Filter

Ahora supongamos que estamos calculando **y[50]** además de **y[51]** con un filtro cuyo **M** es igual a **7**. Las ecuaciones lucirían en un principio así:

$$y[50] = x[47] + x[48] + x[49] + x[50] + x[51] + x[52] + x[53]$$

$$y[51] = x[48] + x[49] + x[50] + x[51] + x[52] + x[53] + x[54]$$

fig 3.29 Cálculo del Moving average filter para las muestras número 50 y 51

Comparando ambas sumatorias, podemos notar, que ambos resultados se componen de una gran cantidad de valores similares, **6** para ser precisos. Esto nos permite deducir que podríamos obtener **y[51]**, solo restando **x[47]**, para luego sumar **x[54]**, al valor previamente calculado **y[50]**. Esto podríamos decirlo en lenguaje matemático de la siguiente manera.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

$$y[51] = y[50] + x[54] - x[47]$$

fig 3.30 Cálculo recursivo para de la muestra número 51

Esta fórmula matemática se le denomina recursiva, ya que cuenta con muestras de la señal de entrada y valores de la señal de salida previamente calculados. Esta recursividad no se debe confundir con las funciones recursivas que se usan en computación²⁵.

Podríamos escribir una fórmula general, para poder calcular cualquier valor de la señal de salida de esta manera.

$$y[51] = y[50] + x[54] - x[47]$$

$$y[i] = y[i - 1] + x[i + p] - x[i - q]$$

$$p = (M - 1) / 2$$

$$q = p + 1$$

fig 3.31 Fórmula matemática recursiva del Moving average filter

Ahora analicemos las ventajas que nos ofrece este método. Primero, sólo realizamos dos cálculos independientemente del tamaño de **M**. Segundo, el resultado se obtiene solo usando sumas y restas. Tercero, el cálculo de los índices, (los valores dentro **[]**), resultan muy sencillos de obtener.

3.6.2.1 Algoritmo

A continuación, se presenta la fórmula matemática descrita previamente en lenguaje de programación C++.

²⁵ (n.d.). Recursive Implementation. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch15/5.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

```

12 void recursive_moving_average( double * pSig_src_arr, double *pSig_out_arr, int signal_lenght, int filter_pts)
13 {
14     int i;
15     double acc;
16     acc = 0;
17
18     for(i = 0; i < filter_pts - 1; i++)
19     {
20         acc = acc + pSig_out_arr [ i ] ;
21     }
22     pSig_out_arr [(filter_pts- 1 / 2)] = acc / filter_pts;
23
24     for (i = (int)floor(filter_pts/2); i < (int) (signal_lenght - floor(filter_pts/2) - 1); i++) {
25         acc = acc + pSig_src_arr [ i + (int) floor((filter_pts - 1) / 2)] - pSig_src_arr [ i - (int) floor(filter_pts/2)];
26         pSig_out_arr [ i ] = acc/filter_pts;
27     }
28 }
29
30
31

```

$$y[i] = y[i - 1] + x[i + p] - x[i - q]$$

fig 3.32 Código en C++ del Recursive Moving Average Filter

Este algoritmo aunque más eficiente, es realmente muy sencillo. Además de la notación con colores que se ha venido usando a lo largo de esta tesis, se agregó un cuadrado, dentro de este cuadrado se encuentra el código del cálculo de **y[0]**, este es el único punto de la señal de salida que se calcula como el algoritmo de la sección 3.7.7.1, el resto de los puntos de la señal de salida, se calculan en el ciclo **for** que le sigue.

3.6.3 Windowed-Sync Filter

Sobre este tipo de filtros pasa baja no se desarrollarán las implicaciones matemáticas que permiten su derivación, ya que eso excede el alcance de esta tesis. Y es posible hacer un uso básico de este filtro sin necesidad de poseer un entendimiento profundo del mismo, que es lo que se pretende en esta tesis.

Lo principal es entender que con esta fórmula matemática se puede desarrollar una señal, que actúa como un filtro pasa bajas cuando se realiza una convolucion con otra señal. Este tipo de filtros nos permite determinar la frecuencia de corte a partir de la cual el resto de frecuencias será atenuada.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

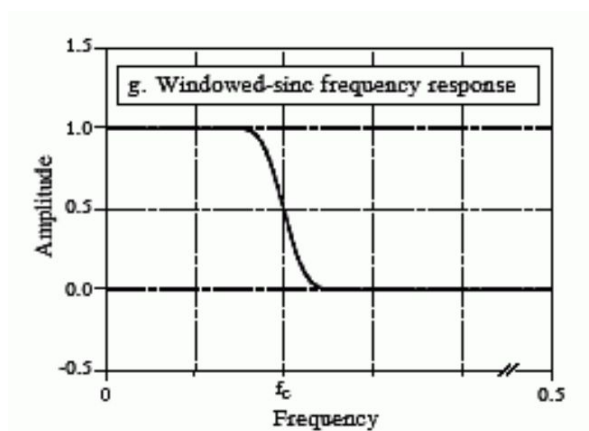


fig 3.33 Gráfica de la respuesta en frecuencia del Windowed-sinc filter

La gráfica de la figura 3.33, muestra cómo actúa el filtro sobre el espectro de frecuencia de una señal de entrada. Todas las frecuencias desde 0 Hz hasta f_c , tienen una amplitud de 1, es decir su amplitud permanece inalterada, mientras que las frecuencias desde f_c hasta, 0.5 de la tasa de sampleo tendrán una amplitud de 0.

$$h[i] = K \frac{\sin(2\pi f_c (i - M/2))}{i - M/2} \left[0.42 - 0.5 \cos\left(\frac{2\pi i}{M}\right) + 0.08 \cos\left(\frac{4\pi i}{M}\right) \right]$$

fig 3.34 Fórmula matemática del Windowed-sinc filter

La figura muestra la fórmula para desarrollar esta señal que actúa como filtro. La variable f_c representa la frecuencia de corte, y como ya se ha venido trabajando a lo largo de esta tesis, va desde 0 hasta 0.5 de la tasa de sampleo. M representa el tamaño de nuestra señal, es decir, de cuántas muestras se va a componer, este número juega un papel importante como se explicará más adelante. Por último, el valor i representa la muestra de la señal del filtro que se está calculando, y va desde 0 hasta M . K es una constante²⁶.

²⁶ (n.d.). Designing the Filter - DSP Guide. Se recuperó el noviembre 26, 2019 de <https://www.dspguide.com/ch16/2.htm>



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

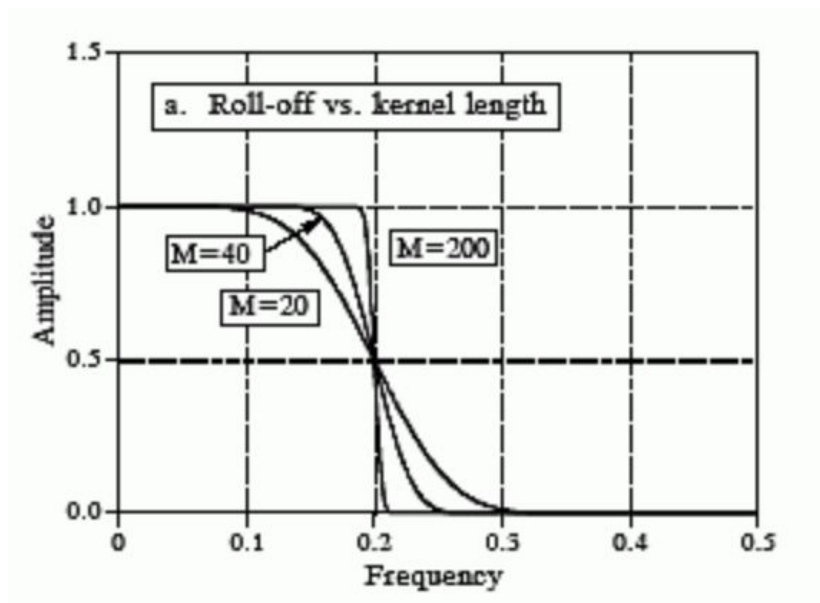


fig 3.35 Gráfica que muestra el papel que juega el valor M

La figura 3.35 muestra la relación entre el tamaño de nuestro filtro, y la inclinación de la banda de transición. A medida que nuestro filtro se hace más grande la banda de transición se va haciendo más empinada.

3.6.3.1 Algoritmo

La fórmula matemática de la figura 3.35, se puede escribir en lenguaje C++ de la siguiente manera:

```
for (int i = 0; i < filtr_lght; i++)
{
    if(i-filtr_lght%2==0)
    {
        filtr_kernel_dest_arr [i] = 2*M_PI*cutoff_freq;
    }

    if (i - filtr_lght%2 !=0) {
        filtr_kernel_dest_arr[i] = sin(2*M_PI*cutoff_freq* (i - filtr_lght/2))/(i - filtr_lght/2);
        filtr_kernel_dest_arr [ i] = filtr_kernel_dest_arr [ i] * (0.42 - 0.5 * cos( (2 * M_PI * i)/ filtr_lght) * 0.08 * cos(4 * M_PI * i - filtr_lght));
    }
}
```

$$h[i] = K \frac{\sin(2\pi f_c (i - M/2))}{i - M/2} \left[0.42 - 0.5 \cos\left(\frac{2\pi i}{M}\right) + 0.08 \cos\left(\frac{4\pi i}{M}\right) \right]$$

fig 3.36 Código en C++ del Windowed-sinc filter



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Se ha elegido el sistema de colores para hacer más sencillo de entender la relación entre la fórmula matemática y el código.

Capítulo 4: Pruebas y resultados

El siguiente paso, es medir qué tan eficientes son los algoritmos avanzados, para tener un conocimiento más profundo de las herramientas diseñadas y saber que resultados esperar a la hora de usar estos algoritmos para implementarlos en aplicaciones especializadas en el procesamiento de audio. En este análisis comparativo se midió la eficiencia en tiempo, precisión y utilidad de cada algoritmo programado. En total se desarrollaron 11 algoritmos: la media, la desviación estándar, RMS, Signal to Noise Ratio, convolución, DFT, IDFT, FFT, Moving Average Filter, Recursive Moving Average Filter, Windowed Sync Filter. Para la media, la desviación estándar, RMS, convolución se planea comparar ambos algoritmos diseñados en cada sección. El Signal to Noise Ratio que es simplemente la división entre la media y la desviación estándar se comparará dividiendo la media y la desviación obtenidas a través de los algoritmos denominados básico y la media y la desviación estándar obtenidas a través de los algoritmos denominados avanzados. La DFT se comparará contra FFT, no solo en velocidad, sino también en precisión usando, el algoritmo Signal to Noise Ratio para calcular la diferencia de ruido de ambas señales de salida, el Moving Average Filter se comparará con el Recursive Moving Average Filter y con el Windowed Sync Filter.

-Variables:

-Precisión: En la mayoría de los casos se comparan dos algoritmos que realizan la misma función, ambos deberían arrojar los mismos resultados.

-Tiempo: se medirá el tiempo de ejecución de cada algoritmo en segundos, y se espera una diferencia considerable de tiempo de ejecución entre cada una.

Los clicks son unidades constantes de tiempo pero dependiente del sistema computacional que se esté usando²⁷. Este valor para efectos didácticos no nos resulta útil, sin embargo, se hará una conversión a segundos, que ilustrará de manera más entendible las diferencias en la variable a medir.

²⁷ (n.d.). time_t - C++ Reference - Cplusplus.com. Se recuperó el noviembre 26, 2019 de http://www.cplusplus.com/reference/ctime/time_t/



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

4.1 Prueba de la media

```
Calculating mean  
The mean of the signal is:0.0371119  
It took me 57 clicks (5.7e-05seconds)
```

Fig 4.1 Resultado en consola del algoritmo de la media

El algoritmo desarrollado en la sección 3.1.1.2 arroja un valor en consola de 0.0371119, recordemos que el arreglo que representa la señal de entrada son en este caso 320 valores que van desde -1 hasta 1. Luego entrega el número de clicks, en este caso 57, y la conversión en segundos. A la computadora le tomó 5.7×10^{-5} segundos.

```
Calculating robust mean  
The mean of the signal is:0.0371119  
It took me 20 clicks (2e-05seconds)
```

Fig 4.2 Resultado en consola del algoritmo de la media

El algoritmo desarrollado en la sección 3.1.1.3 arroja un valor en consola de 0.0371119, este valor es igual que el de la figura 4.1. Luego entrega el número de clicks, en este caso 20, y la conversión en segundos. A la computadora le tomó 2.0×10^{-5} segundos.

4.2 Prueba de la Desviación estándar

```
Calculating standard deviation  
The Standard Deviation of the signal is:0.787502  
It took me 49 clicks (4.9e-05seconds)
```

Fig 4.3 Resultado en consola del algoritmo de la Desviación estándar

El algoritmo desarrollado en la sección 3.1.2.2 arroja un valor en consola de 0.787502. Luego entrega el número de clicks, en este caso 49, y la conversión en segundos. A la computadora le tomó 4.9×10^{-5} segundos.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

```
Calculating robust standard Deviation
The Standard Deviation of the signal is:0.788379
It took me 12 clicks (1.2e-05seconds)
```

Fig 4.4 Resultado en consola del algoritmo de la Desviación estándar

El algoritmo desarrollado en la sección 3.1.2.2 arroja un valor en consola de 0.787502. Luego entrega el número de clicks, en este caso 12, y la conversión en segundos. A la computadora le tomó 1.2×10^{-5} segundos.

4.3 Prueba de la RMS

```
Calculating RMS
The RMS of the signal is:0.787146
It took me 11 clicks (1.1e-05seconds)
```

Fig 4.5 Resultado en consola del algoritmo de la RMS

El algoritmo desarrollado en la sección 3.1.3.2 arroja un valor en consola de 0.787146. Luego entrega el número de clicks, en este caso 11, y la conversión en segundos. A la computadora le tomó 1.1×10^{-5} segundos.

```
Calculating the Robust RMS
The Robust RMS of the signal is:0.787146
It took me 12 clicks (1.2e-05seconds)
```

Fig 4.6 Resultado en consola del algoritmo de la RMS

El algoritmo desarrollado en la sección 3.1.3.3 arroja un valor en consola de 0.787146. Luego entrega el número de clicks, en este caso 12, y la conversión en segundos. A la computadora le tomó 1.2×10^{-5} segundos.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

4.4 Prueba del Signal to Noise Ratio

```
Calculating the Signal to Noise Ratio  
The Signal to Noise Ratio of the signal is:0.0471261  
It took me 11 clicks (2.4e-05seconds)
```

Fig 4.7 Resultado en consola del algoritmo de la Signal to Noise Ratio

El algoritmo desarrollado en la sección 3.1.4 arroja un valor en consola de 0.0471261. Luego entrega el número de clicks, en este caso 11, y la conversión en segundos. A la computadora le tomó 2.4×10^{-5} segundos.

```
Calculating the Robust Signal to Noise Ratio  
The Signal to Noise Ratio of the signal is:0.0470737  
It took me 4 clicks (4e-06seconds)
```

Fig 4.8 Resultado en consola del algoritmo de la Signal to Noise Ratio

El algoritmo desarrollado en la sección 3.1.4 arroja un valor en consola de 0.0470737. Luego entrega el número de clicks, en este caso 4, y la conversión en segundos. A la computadora le tomó 4.0×10^{-6} segundos.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

4.5 Prueba de la convolución

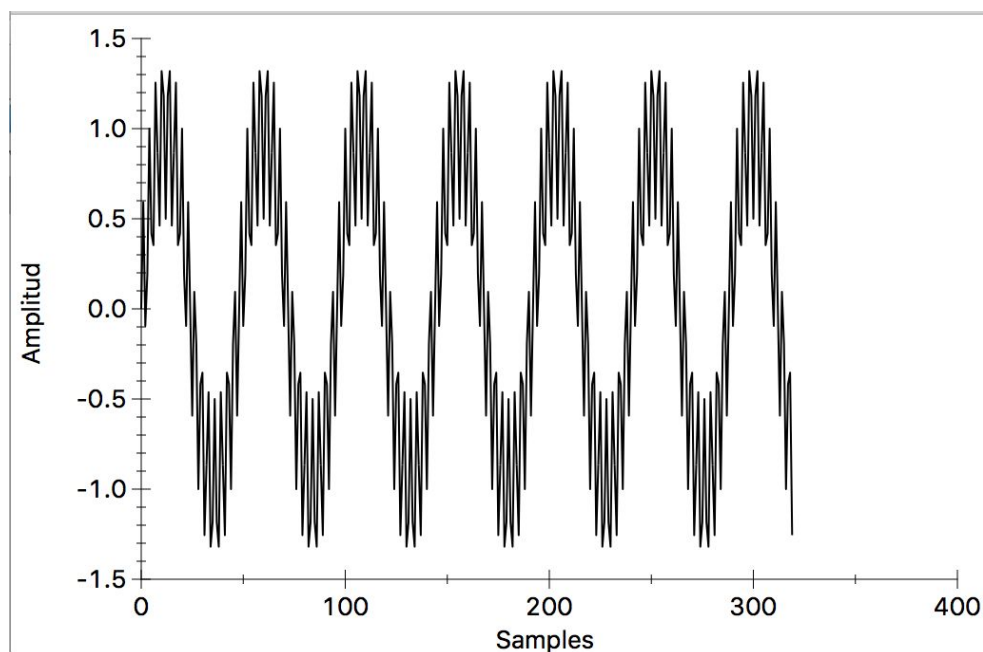


Fig 4.9 Señal de entrada

La figura 4.9 muestra una señal de 1 KHz y 15 KHz, en el **eje y** está en función de la amplitud y el **eje x** esta en función de muestras .

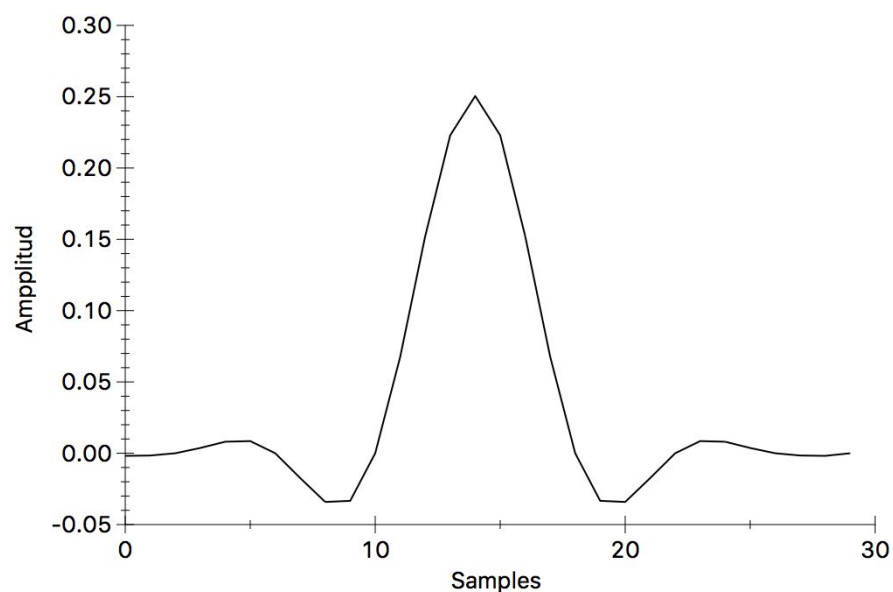


Fig 4.10 Señal de impulso



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

La figura 4.10 representa una respuesta de impulso que debe actuar como un filtro pasa bajas con una frecuencia de corte de 10 KHz.

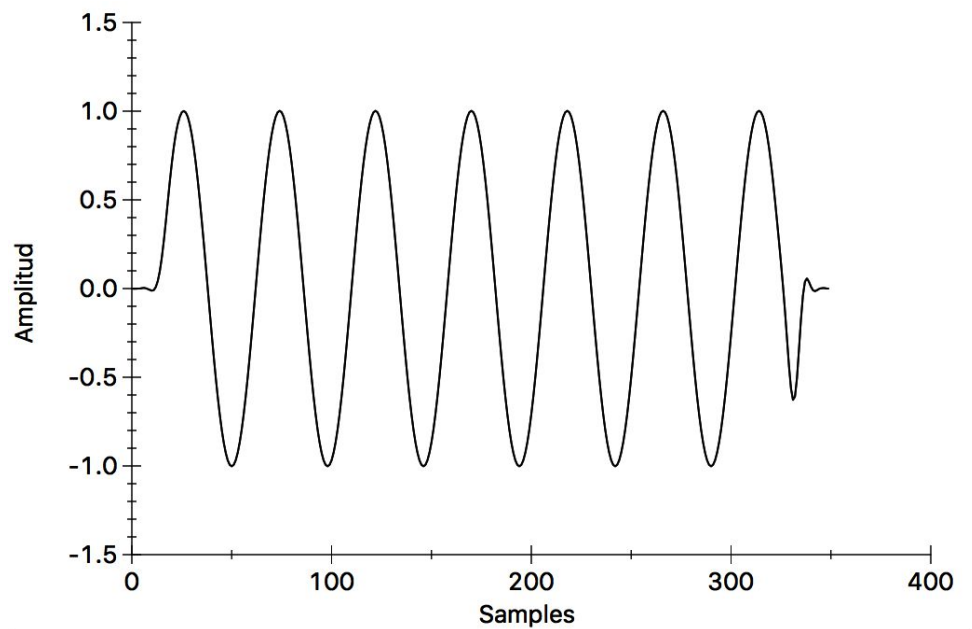


Fig 4.11 Señal de salida de algoritmo 3.2.2

La figura 4.11 representa la señal de salida después de la convolución entre las dos señales previamente señaladas a través del algoritmo de la sección 3.2.2. Se puede observar que la frecuencia de 15KHz ha sido removida.

**Convolution successfully executed
It took me 66 clicks (6.6e-05seconds)**

Fig 4.12 Resultado en consola del algoritmo de convolución

La figura 4.12 muestra el resultado del algoritmo de la sección 3.2.2. El resultado en consola entrega el número de clicks, en este caso 66, y la conversión en segundos. A la computadora le tomó 6.6×10^{-5} segundos.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

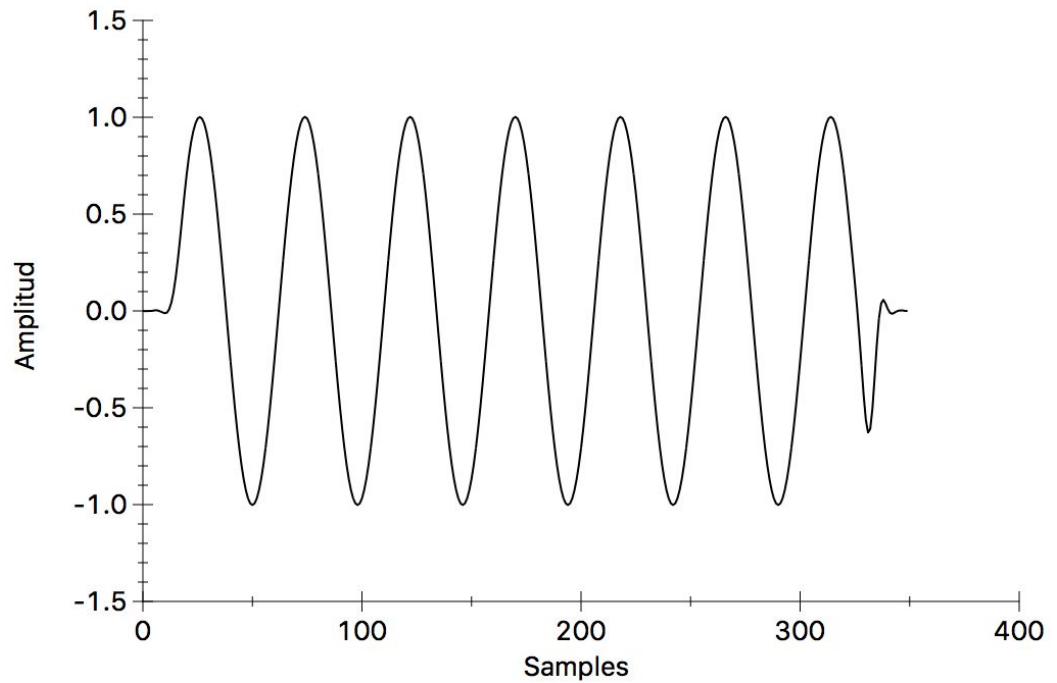


Fig 4.13 Señal de salida de algoritmo 3.2.3

La figura 4.13 representa la señal de salida después de la convolución entre las dos señales previamente señaladas a través del algoritmo de la sección 3.2.3. Se puede observar que la frecuencia de 15KHz ha sido removida.

**Robust convolution successfully executed
It took me 28 clicks (2.8e-05seconds)**

Fig 4.14 Resultado en consola del algoritmo de convolución

La figura 4.14 muestra el resultado del algoritmo de la sección 3.2.3. El resultado en consola entrega el número de clicks, en este caso 28, y la conversión en segundos. A la computadora le tomó 2.8×10^{-5} segundos.

4.6 Prueba de DFT



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

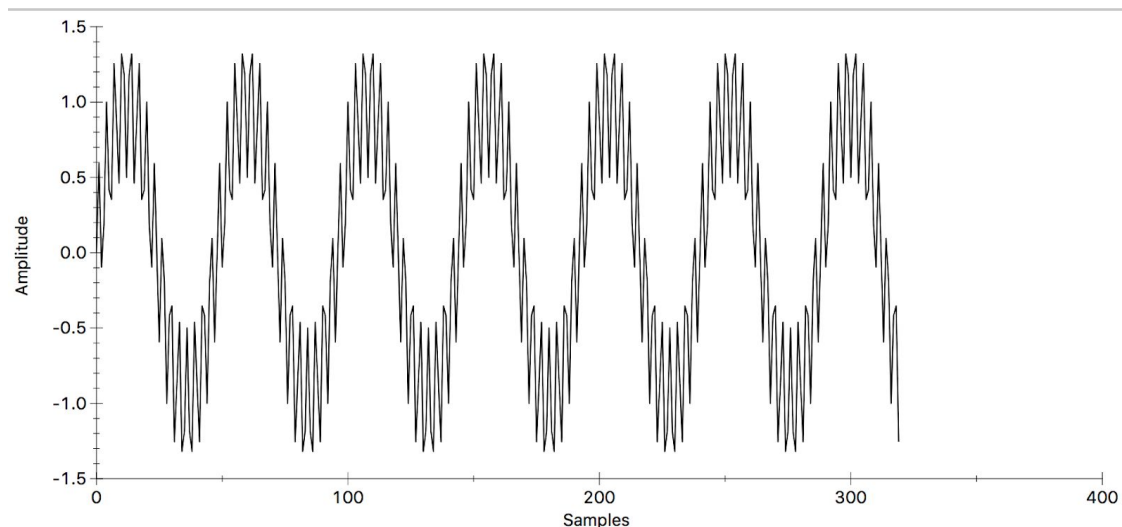


Fig 4.15 Señal de entrada

La figura 4.15 muestra una señal de 1 KHz y 15 KHz, en el **eje y** está en función de la amplitud y el **eje x** está en función de muestras .

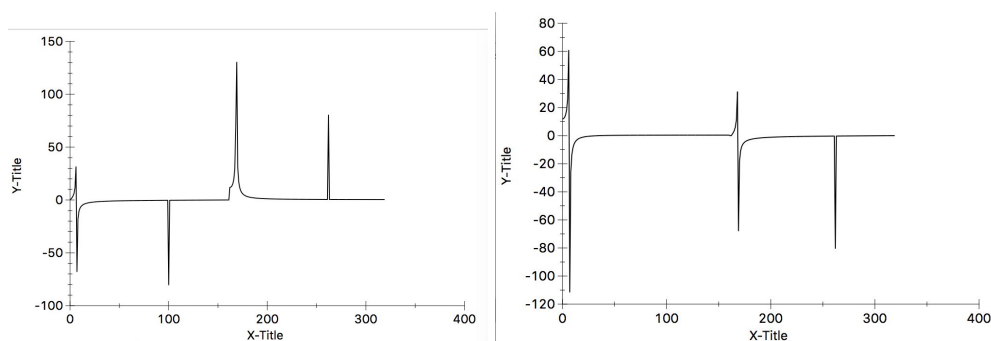


Fig 4.16 Gráfica de la parte Imaginaria y la parte Real de nuestra señal de entrada

La figura 4.16 muestra las gráficas tanto de la parte imaginaria como de la parte real de la señal de entrada.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

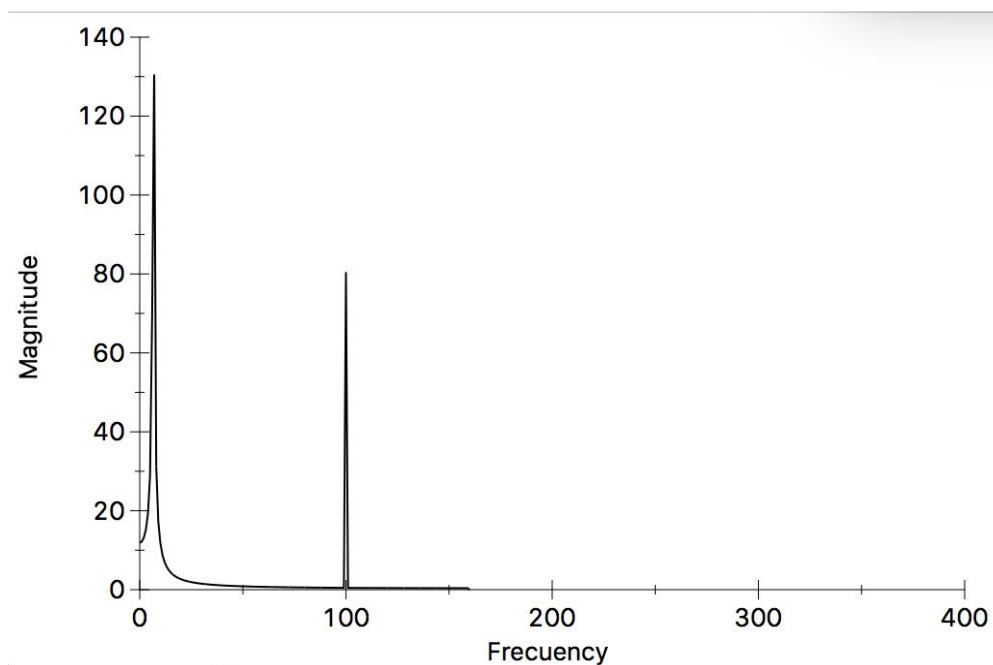


Fig 4.17 Espectro frecuencial de la señal de entrada.

La figura 4.17 muestra el espectro de frecuencia que resulta del cálculo de la magnitud ($Magnitud = \sqrt{Real^2 + Imaginaria^2}$). El **eje x** va desde 0 hasta 160, siendo 160 la frecuencia Nyquist, en nuestro caso ya que la señal de entrada fue generada a 48 KHz de S/R, sería igual a 24KHz. Los dos picos representan la frecuencia de 1KHz y la de 15KHz



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

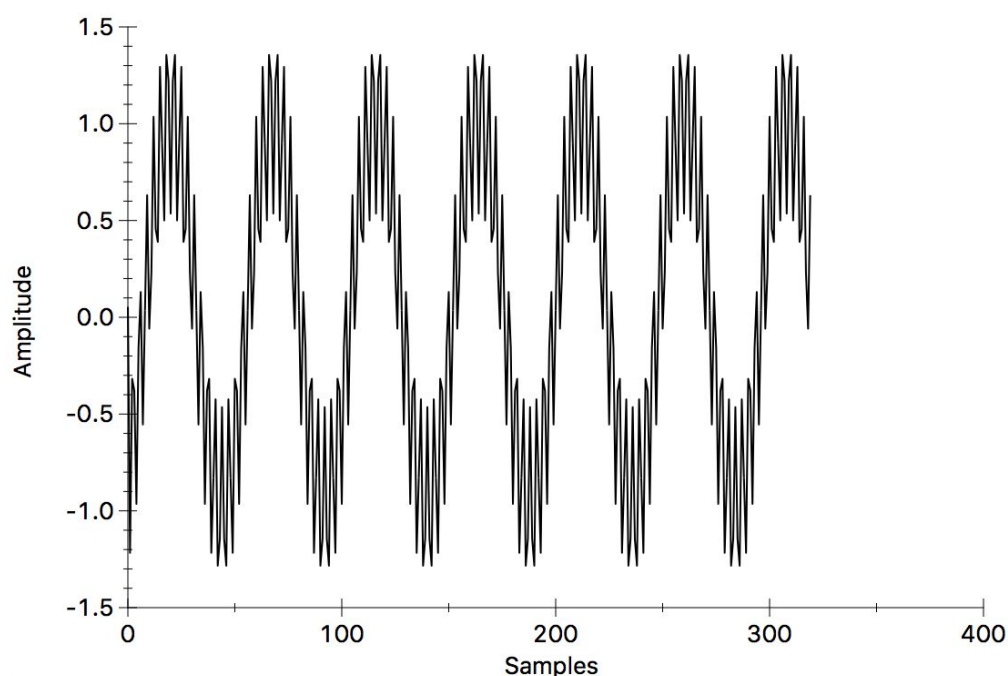


Fig 4.18 Resultado de la IDFT

La figura 4.18 muestra el resultado del proceso inverso de la DFT, por medio del cual se toman los valores la figura 4.16 y se sintetiza la señal de nuevo, podemos notar, que pese a un ligero cambio de fase la señal aún posee ambos componentes las frecuencias de 1KHz y 15KHz.

**DFT successfully executed
It took me 2241 clicks (0.002241seconds)**

Fig 4.18 Resultado en consola del algoritmo DFT

La figura 4.18 muestra el resultado del algoritmo de la sección 3.3.2. El resultado en consola entrega el número de clicks, en este caso 2241, y la conversión en segundos. A la computadora le tomo 0.002241 segundos.

4.7 Prueba de FFT



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

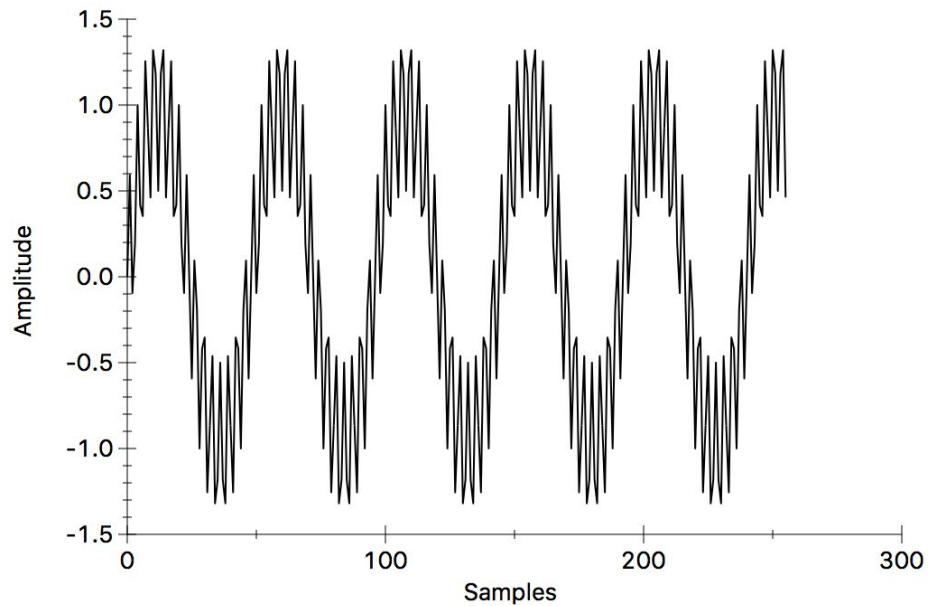


Fig 4.19 Señal de entrada

La figura 4.19 muestra una señal de 1 KHz y 15 KHz, en el **eje y** está en función de la amplitud y el **eje x** está en función de muestras.

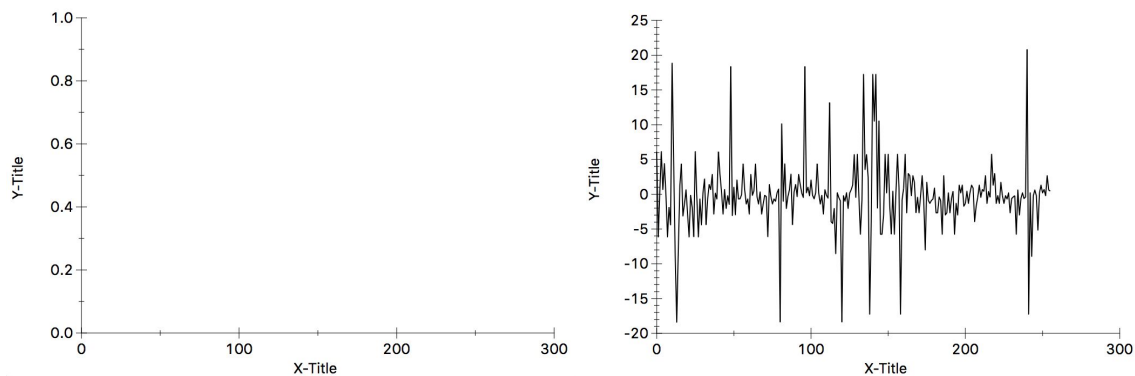


Fig 4.20 Gráfica de la parte Imaginaria y la parte Real de nuestra señal de entrada

La figura 4.20 muestra las gráficas tanto de la parte imaginaria como de la parte real de la señal de entrada. Se puede observar que la gráfica de la izquierda no posee valores esto puede indicar que el algoritmo desarrollado falla alguna parte.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

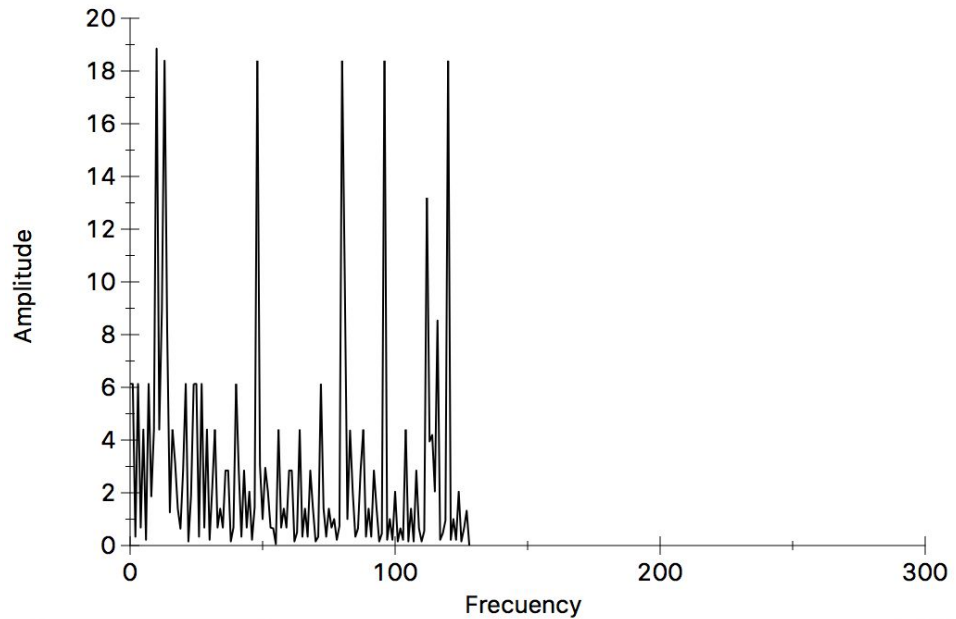


Fig 4.21 Espectro frecuencial de la señal de entrada.

La figura 4.21 muestra el espectro de frecuencia que resulta del cálculo de la magnitud ($Magnitud = \sqrt{Real^2 + Imaginaria^2}$). El **eje x** va desde 0 hasta 160, siendo 160 la frecuencia Nyquist, en nuestro caso ya que la señal de entrada fue generada a 48 KHz de S/R, sería igual a 24KHz. Se puede observar que hay una gran cantidad de ruido en la señal y se pueden observar hasta seis picos en la señal, lo que corrobora la idea de que el algoritmo falla en alguna parte.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

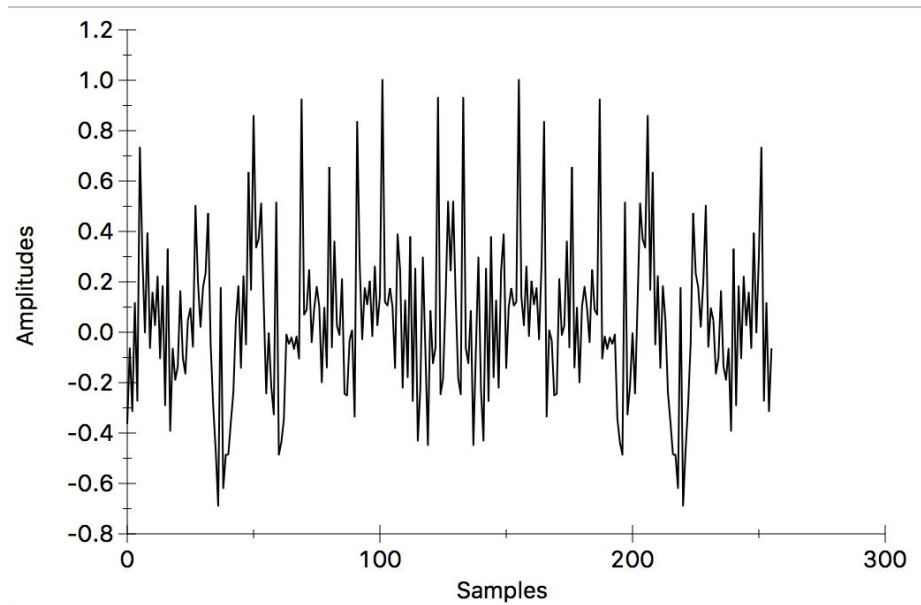


Fig 4.22 Resultado de la IDFT

La figura 4.22 muestra el resultado del proceso inverso de la FFT, por medio del cual se toman los valores la figura 4.20 y se sintetiza la señal de nuevo. Como se estableció previamente, la señal de salida confirma que el algoritmo desarrollado para la FFT falla el alguna parte del código.

FFT successfully executed
It took me 87 clicks (8.7e-05seconds)

Fig 4.23 Resultado en consola del algoritmo FFT

La figura 4.23 muestra el resultado del algoritmo de la sección 3.5. El resultado en consola entrega el número de clicks, en este caso 87, y la conversión en segundos. A la computadora le tomó 8.7×10^{-5} segundos.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

4.8 Prueba de Moving Average Filter

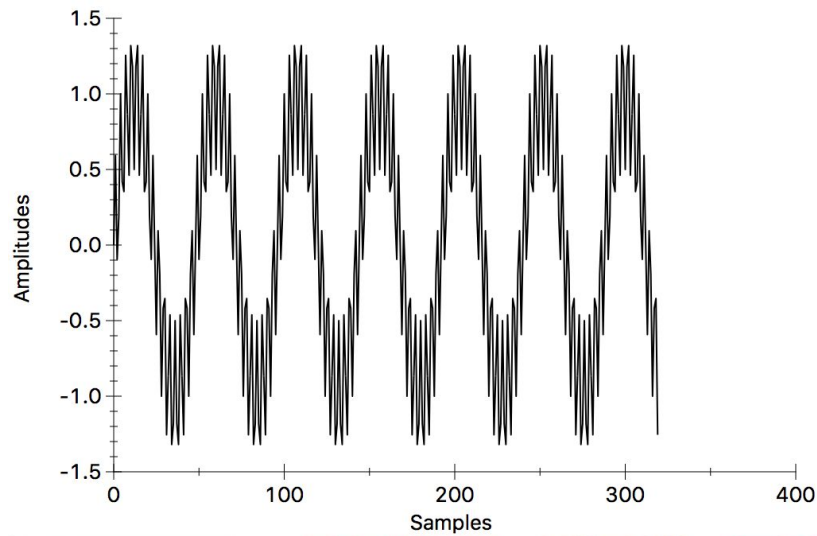


Fig 4.24 Señal de entrada

La figura 4.24 muestra una señal de 1 KHz y 15 KHz, en el **eje y** está en función de la amplitud y el **eje x** está en función de muestras .

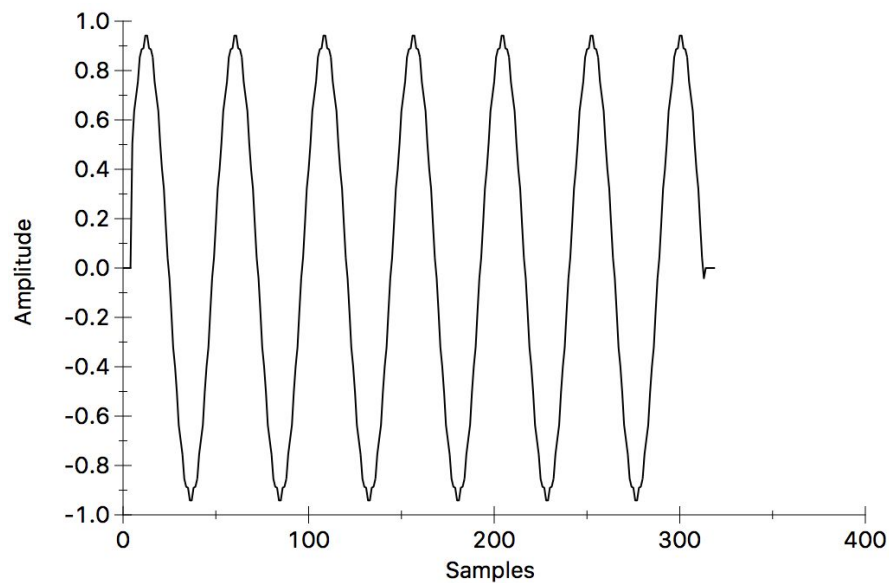


Fig 4.25 Señal de salida

La figura 4.25 muestra la señal de salida, se puede observar que la frecuencia de 15 KHz ya no se encuentra y solo se encuentra la frecuencia de 1 KHz.

**Moving average filter successfully executed
It took me 35 clicks (3.5e-05seconds)**

Fig 4.26 Resultado en consola del algoritmo Moving average filter

La figura 4.26 muestra el resultado del algoritmo de la sección 3.6.1. El resultado en consola entrega el número de clicks, en este caso 35, y la conversión en segundos. A la computadora le tomó 3.5×10^{-5} segundos.

4.9 Prueba de Recursive Moving Average Filter

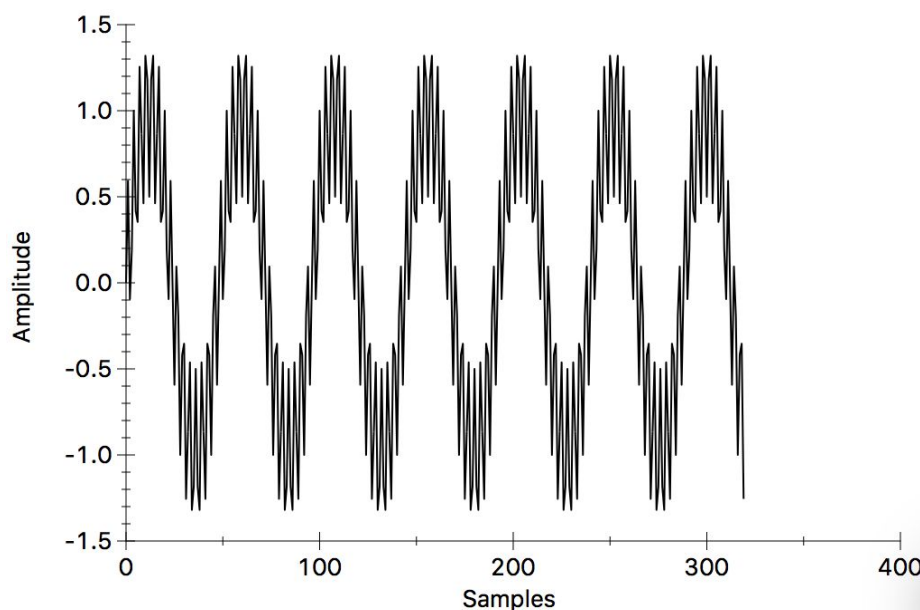


Fig 4.27 Señal de entrada

La figura 4.27 muestra una señal de 1 KHz y 15 KHz, en el **eje y** está en función de la amplitud y el **eje x** está en función de muestras.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

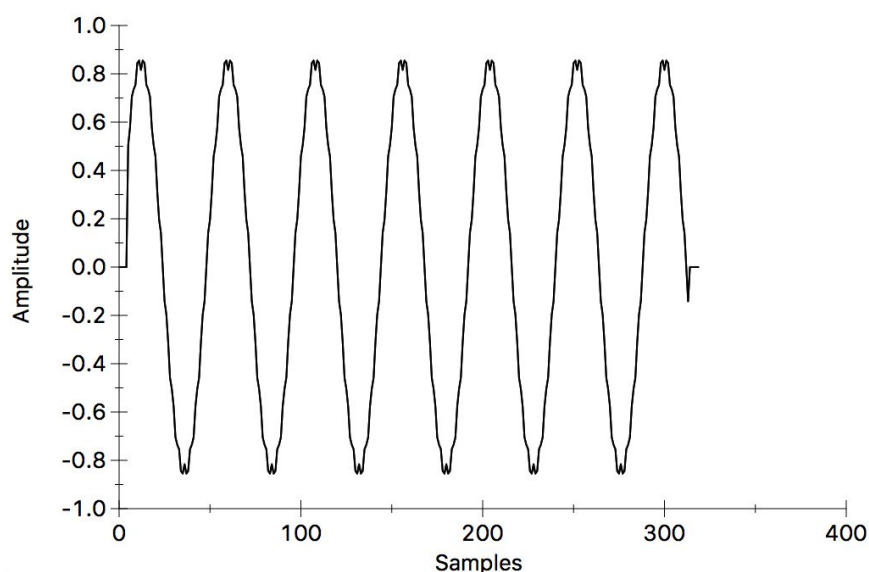


Fig 4.28 Señal de salida

La figura 4.28 muestra la señal de salida, se puede observar que el la frecuencia de 15 KHz ya no se encuentra y solo se encuentra la frecuencia de 1KHz.

**Recursive moving average filter successfully executed
It took me 18 clicks (1.8e-05seconds)**

Fig 4.29 Resultado en consola del algoritmo Recursive Moving average filter

La figura 4.29 muestra el resultado del algoritmo de la sección **3.6.2**. El resultado en consola entrega el número de clicks, en este caso 18, y la conversión en segundos. A la computadora le tomó 1.8×10^{-5} segundos.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

4.10 Prueba del Windowed Sync Filter

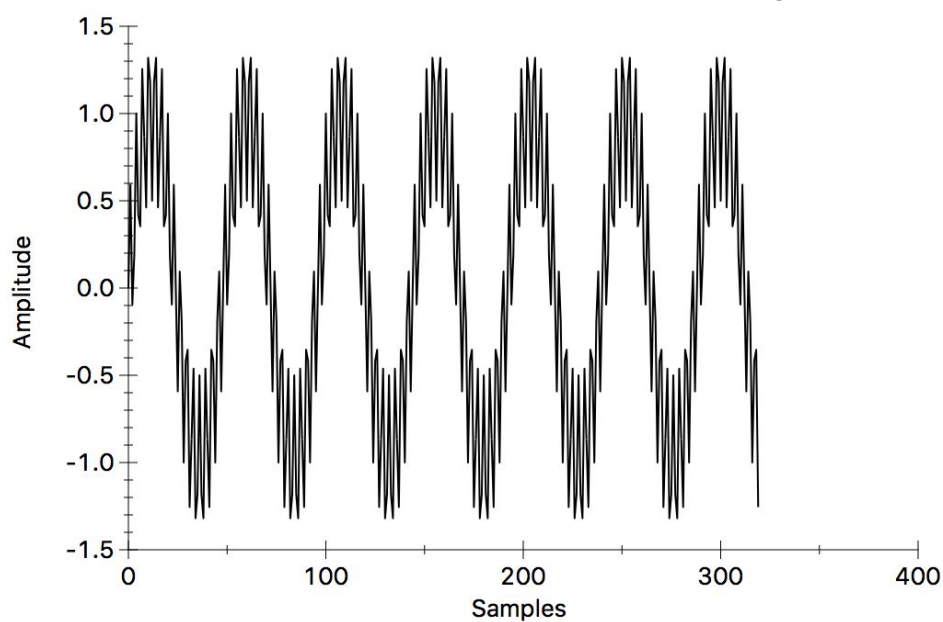


Fig 4.30 Señal de entrada

La figura 4.30 muestra una señal de 1 KHz y 15 KHz, en el **eje y** está en función de la amplitud y el **eje x** está en función de muestras.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

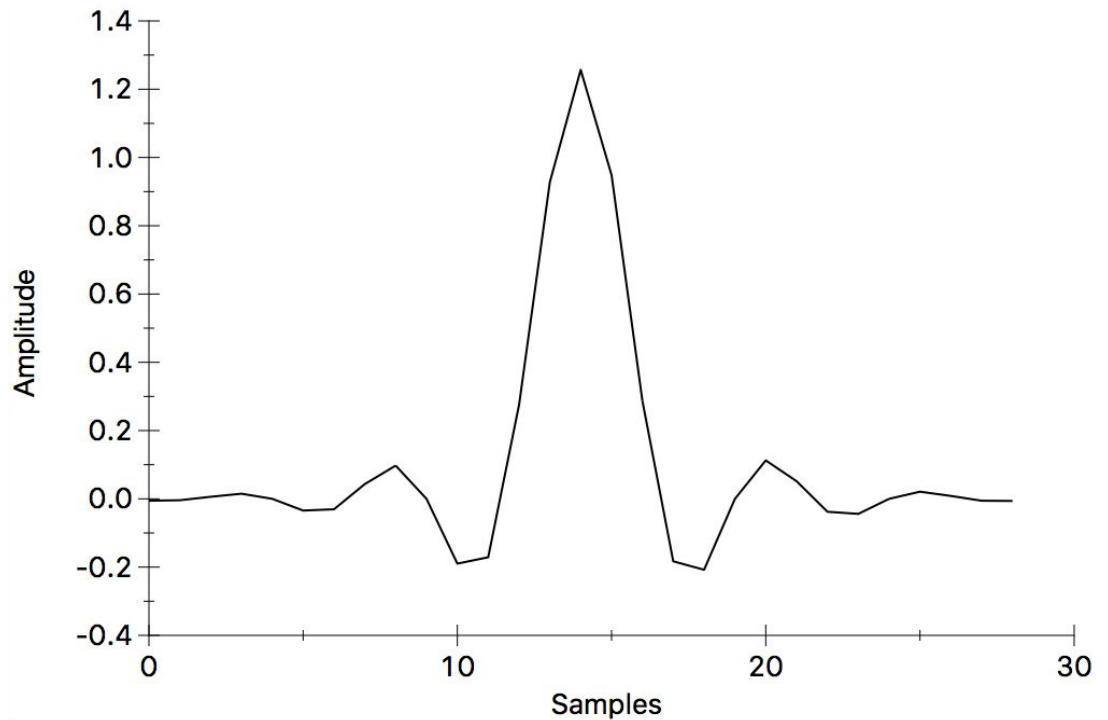


Fig 4.31 Señal de impulso

La figura 4.31 representa una respuesta de impulso que debe actuar como un filtro pasa bajas con una frecuencia de corte de 10 KHz. Este filtro es el resultado del algoritmo. Ahora se hará uso del algoritmo de convolución entre la señal de la figura 4.30 y la respuesta de impulso 4.31, se espera filtrar todas las frecuencias por encima de 10 KHz de la señal de entrada.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

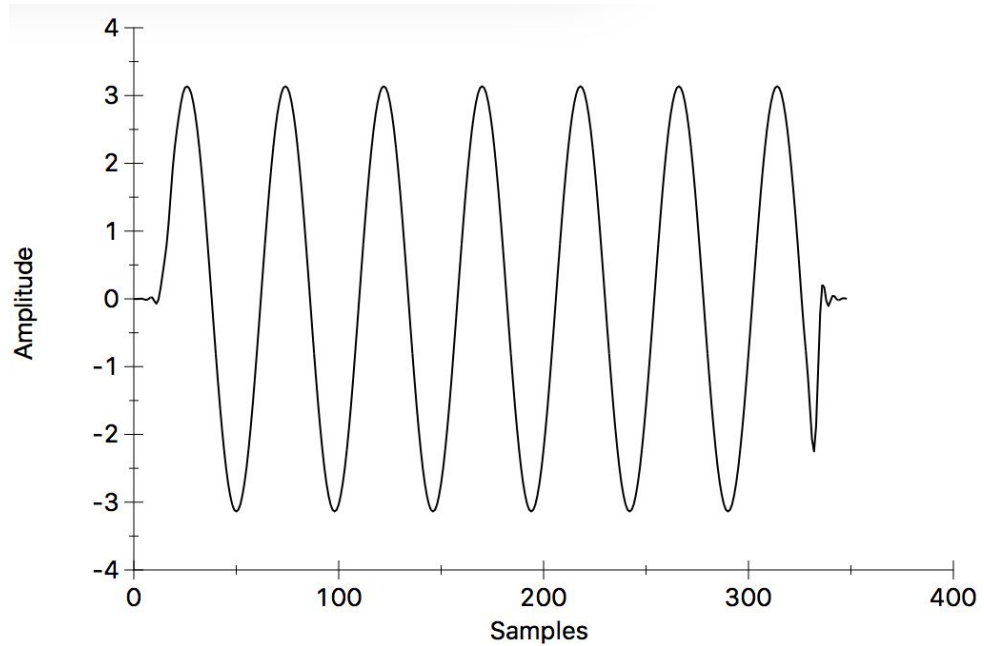


Fig 4.32 Señal de salida

La figura 4.32 muestra la señal de salida, se puede observar que la frecuencia de 15 KHz ya no se encuentra y solo se encuentra la frecuencia de 1 KHz.

```
Windowed Sync filter successfully generated
It took me 4 clicks (4e-06seconds)
```

```
Convolution between Input signal and Windowed sync filter successfully executed
It took me 64 clicks (6.4e-05seconds)
```

```
The total time is: 68 clicks (6.8e-05seconds)
```

Fig 4.33 Resultado en consola del algoritmo Windowed Syn Filter mas la convolución entre las dos señales.

La figura 4.33 muestra el resultado del algoritmo de la sección 3.6.3. El resultado en consola entrega el número de clicks, en este caso 4, solo para la generación del filtro, y la conversión en segundos. A la computadora le tomó 4.0×10^{-6} segundos. La segunda parte de la prueba tiene similitud con el resultado arrojado en consola de la figura 4.12.

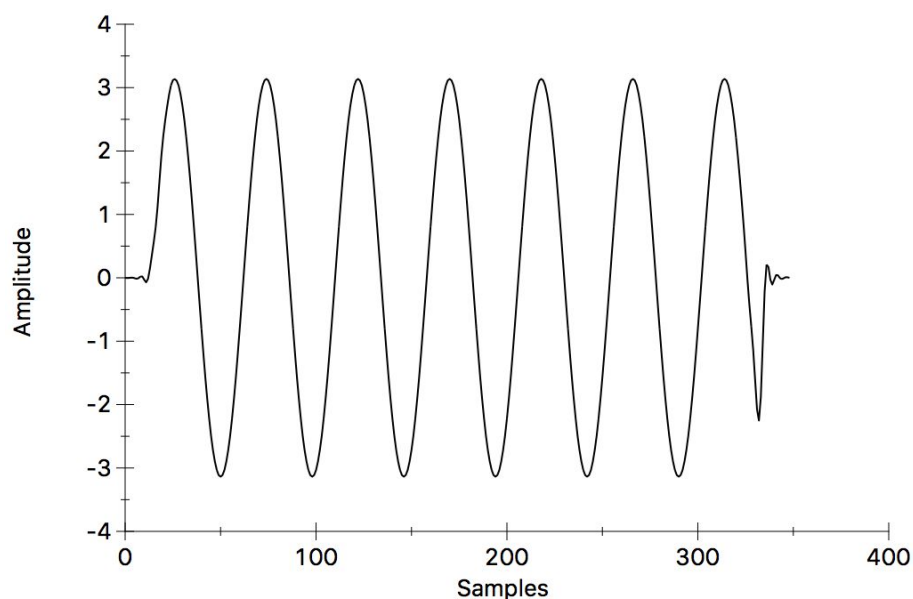


Fig 4.34 Señal de salida

La figura 4.34 muestra la señal de salida, se puede observar que el la frecuencia de 15 KHz ya no se encuentra y solo se encuentra la frecuencia de 1 KHz.

Windowed Sync filter successfully generated
It took me 6 clicks (6e-06seconds)

Robust Convolution between Input signal and Windowed sync filter successfully executed
It took me 52 clicks (5.2e-05seconds)

The total time is: 58 clicks (5.8e-05seconds)

Fig 4.35 Resultado en consola del algoritmo Windowed Syn Filter más la convolución entre las dos señales usando el algoritmo de la sección 3.2.3.

La figura 4.35 muestra el resultado del algoritmo de la sección 3.6.3. El resultado en consola entrega el número de clicks, en este caso 6, solo para la generación del filtro, y la conversión en segundos. A la computadora le tomó 6.0×10^{-6} segundos. La segunda parte de la prueba no tiene similitud con el resultado arrojado en consola de la figura 4.14.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Capítulo 5: Conclusiones

Uno de los objetivos de esta tesis, que era obtener un entendimiento pleno de los conceptos básicos en el procesamiento digital de señales, no se logró de acuerdo a las expectativas deseadas.

Todos los conceptos mencionados en esta tesis poseen propiedades que no fueron analizadas ni mencionadas en esta tesis. Cabe mencionar que aunque el desarrollo teórico de la FFT fue exitoso, el algoritmo no funcionó, como se esperaba. Resolver el algoritmo requerirá una investigación mucho mayor de la realizada para esta tesis, que aunque ha sido extensa y necesaria no ha sido suficiente.

A pesar de eso, la meta del proyecto, lograr una derivación matemática de los conceptos básicos del procesamiento digital de señal y desarrollar una aplicación básica que permita evaluar y comparar los algoritmos obtenidos, fue cumplida. Se investigaron los conceptos planteados, se desarrollaron los algoritmos necesarios y se realizó una implementación exitosa que permitió una comparación bastante precisa corroborando que la teoría fue desarrollada correctamente.

Por último, se debe recordar la intención final del proyecto: implementar plug ins de audio que sean accesibles a través de un sitio web y que realicen el procesamiento digital de la señal en la nube e implementar un DAW que funcione de la misma manera.

Todos los algoritmos denominados como avanzados probaron ser igual de precisos y más eficientes que los denominados como básicos, esto se comprobó de manera precisa. Los algoritmos básicos sin embargo permiten entender la implementación de conceptos abstractos en el mundo real, y son por ende más útiles para fines didácticos.

Dentro de este contexto, se puede concluir que los algoritmos avanzados, excluyendo la FFT, presentan mayores ventajas a la hora de realizar procesamiento digital de señales en tiempo real, que es el requerido cuando se trabaja con softwares de audio.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS

Bibliografía

- Steven W. Smith (1997): *The Scientist & Engineer's Guide to Digital Signal Processing* California, USA: California Technical Pub. Recuperado el 15 de junio de 2018 de <https://www.dspguide.com/pdfbook.htm>
- Deitel, Harvey, Paul J. Deitel (2003): *Como programar en C++* Estado de Mexico: Pearson Educación. Consultado el 11 de febrero de 2019.
- Doumler, T., Davidson G. & Somberg G.. (2019). A Standard Audio API for C++: Motivation, Scope, and Basic Design. Recuperado el 7 de junio de 2019, de Open Standards Sitio web: <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1386r0.pdf>
- Burg, Jennifer, Jason Romney, Eric Schwartz (2016): *Digital Sound & Music: Concepts, Applications, and Science* Portland, Oregon: Franklin, Beedle & Associates Inc. Recuperado el 15 de noviembre de 2019 de <http://digitalsoundandmusic.com/>
- The C++ Resources Network. (2000-2019). Cplusplus. Recuperado el 10 octubre de noviembre de 2019 de <http://www.cplusplus.com/>.
- Davis, Gary, Ralph Jones (1988): *Sound Reinforcement Handbook* Milwaukee, Wisconsin: Hal Leonard Corporation. Consultado el 7 de octubre de 2019
- Stack Exchange Inc. (2019). StackOverflow. US.. Recuperado el 15 febrero de 2019 de <https://stackoverflow.com/>.



ANIMACIÓN



AUDIO



VIDEOJUEGOS



CINE



MUSIC BUSINESS