UNIVERSIDAD SAN FRANCISCO DE QUITO

Colegio de Ciencias e Ingeniería

Diseño y Construcción de una pedalera de guitarra digital basada en Raspberry Pi

Marco Antonio Maigua Terán

René Játiva, DEA, Director de Tesis

Tesis de grado presentada como requisito
para la obtención del título de Ingeniero
Electrónico

Quito, agosto de 2015

UNIVERSIDAD SAN FRANCISCO DE QUITO
COLEGIO DE CIENCIAS E INGENIERÍAS

HOJA DE APROBACIÓN DE TESIS

Título de la tesis: Diseño y Construcción de una pedalera de guitarra digital basada en Raspberry Pi

Marco Antonio Maigua Terán

René Játiva, DEA
Director de Tesis _____

Luis Miguel Prócel, M.Sc.
Co-Director de Tesis
Miembro del Comité de Tesis _____

Omar Aguirre, M.Sc.
Miembro del Comité de Tesis _____
Director del Dep. de Ing. Electrónica

Paul Frutos, BCs en Ing. Electrónica
Miembro del Comité de Tesis _____

César Zambrano, Ph.D.
Decano del colegio de Ciencias e Ingenierías _____

Quito, agosto de 2015

Firma:

Nombre: Marco Antonio Maigua Terán

C. I.: 1003202866

Lugar y fecha: Quito, agosto de 2015

A mis padres y a la vida…

RESUMEN

El creciente avance de los sistemas embebidos ha permitido desarrollar dispositivos que se ajustan a aplicaciones específicas en cualquier ámbito social, incluyendo la industria de la música. Con la aparición de dispositivos programables y de alta capacidad de procesamiento la industria de la música ha tomado ventaja sobre los algoritmos DSP para obtener audio de la más alta calidad y a la vez de la más alta manipulación. El presente proyecto pretende implementar conceptos de música, procesamiento digital de señales (DSP), análisis de señales y módulos de hardware de alta abstracción para crear un procesador de audio en tiempo real de una señal de audio proveniente de un instrumento que funciona a base de vibraciones electromagnéticas. En términos de la industria musical, esta aplicación se denomina un pedal digital de guitarra para producir efectos determinados. El proyecto se enfocará en un efecto específico completamente controlable además del control de la calidad del sonido de la salida. La aplicación será desarrollada sobre una plataforma de hardware Raspberry Pi con programación Python para la interfaz y el procesamiento de señal. En conjunto el proyecto emula a productos similares en la industria de la música. Cada año, esta industria innova la tecnología que controla el sonido de muchos artistas en conciertos, estudios e incluso en dispositivos caseros a disposición de todo tipo de usuarios.

ABSTRACT

The recent advance of the embedded systems has led to the development of the devices that adjust to specific applications in any social environment, including the music industry. With the emergence of the programmable and high processing capacity devices, the music industry has taken advantage over the DSP algorithms to obtain the highest quality of audio, and at the same time, the highest manipulation over it. The current project aims the implementation of music, digital signal processing and hardware modules concepts to create a real time audio processor of an audio signal that comes from an instrument of electromagnetic vibrations basis. On the mainstream music industry, it is known as a guitar digital pedal to produce certain effects. The project focuses on a specific effect and the control of the sound quality on the output. The application develops over a Raspberry Pi hardware platform with Python programming language for the interface and the processing algorithms as well. In conclusion, the project emulates similar products of the music industry. Each year, this industry push the bounds of technology that control the sound of many artists on concerts, records, and even at home with simple devices for any kind of user.

TABLA DE CONTENIDO

Antecedentes

Uno de los primeros sistemas embebidos fue la computadora guía Apolo y antes fue la computadora guía Autonetics D-17 para controlar el misil Minuteman en 1961. Desde estas aplicaciones, los sistemas embebidos han reducido en su costo y el poder de procesamiento y funcionalidad ha incrementado. En estos días podemos mencionar la importancia de los sistemas embebidos en casi todos los aspectos de la tecnología moderna. Por ejemplo estos sistemas son usados en automóviles, aviones, trenes, vehículos, herramientas de máquina, cámaras, aparatos electrónicos de consumo, aplicaciones de oficina, aplicaciones de redes, videojuegos, celulares, PDAs, navegación GPS así como también en robots y juguetes.

Justificación e Importancia

Este proyecto pretende integrar sistemas embebidos para aplicaciones específicas y eficientes y cubre necesidades de la industria musical. Con el desarrollo de proyectos de este tipo, se contribuye al desarrollo de tecnologías más integradas y de acceso libre y que abarcan inclusive, el desarrollo de construcciones sociales como el arte y en este proyecto en específico, la música.

Desde el punto de vista de la aplicación, el proyecto será capaz de integrar los fundamentos básicos de comunicaciones, DSP, análisis de señales, conceptos de redes de computación, programación funcional y gráfica. El trabajo en diversos frentes en el campo de la electrónica permitirá desarrollar un producto útil de calidad y posiblemente comerciable. La

aplicación se localiza en la vanguardia de las tecnologías de procesamiento de audio en la industria de la música porque integra las crecientes aplicaciones en dispositivos móviles, la comunicación wireless y la calidad en el procesamiento para audio que en última instancia es lo que todos los usuarios afines a la música desean.

Desde un punto de vista más general con respecto al campo de la electrónica, el uso de sistemas embebidos está impulsando la innovación en aplicaciones humanas y a su vez el desarrollo de la tecnología en general, por ejemplo en el campo de la educación. Gareth Mitchell, un representante de la radio de ciencia y tecnología "Click" en la BBC World Service, expone que ahora la juventud usa estas tarjetas para desarrollar muchas aplicaciones. Sin embargo aunque los jóvenes las utilizan, en realidad se convierten en una suerte de "cajas negras", pues no entienden cómo funcionan. No hay un aprendizaje de la estructura interna de dispositivos electrónicos tales como Arduino. Con estas tarjetas la gente puede fácilmente escribir códigos fuente en el front end para implementar alguna función, sin embargo Mitchell piensa que esto podrías ser un obstáculo en el futuro porque una generación entera crece con estos dispositivos como única solución. Otras tarjetas como Raspberry Pi podrían tener un efecto contrario. Raspberry Pi en efecto es más útil en el sentido que se integran conceptos, técnicas y habilidades de programación. Los críticos piensan que el uso de este tipo de tarjetas podría ayudar a entender las piezas electrónicas primarias y posteriormente facilitar el entender como otros dispositivos más grandes funcionan. Raspberry Pi en la opinión de Mitchell podría innovar la educación de las próximas generaciones. En una mayor escala, este tipo de tarjetas podrían tener el mismo efecto en países en desarrollo como África mejorando

la educación de la tecnología y alcanzando economías de escala así como lo ha hecho el internet (Mitchell, G, 2012).

Por otra parte, ya en su primer aniversario, Raspberry Pi se ha convertido en la manera más barata de obtener poder computacional para varias aplicaciones. Algunas compañías están tratando de obtener un espacio en el mercado del hardware "hobbyist", por ejemplo GizmoSphere o Microsoft han hecho un esfuerzo en exportar hardware de bajo costo a diferentes países. Ya que el código fuente es libre para la mayoría de usuarios, se pueden encontrar maneras de mejorar arquitecturas; y muchas de estas compañías quieren tornar estas mejoras realizadas por usuarios regulares en prototipos de alto nivel industrial. La demanda del Raspberry Pi ha incrementado y bajando su costo hasta $25 dado el telescopio de aplicaciones en la industria de la electrónica, por ejemplo CISCO está usando el Pi como cerebros de cómputo para la automatización de casas inteligentes conectando diferentes dispositivos y usando comunicaciones de radio de baja potencia. Otros ejemplos son las consultorías IT y las firmas tecnológicas como la consultora PA. PA está usando el Pi para construir una estación base GSM compacta como una muestra de que este tipo de hardware puede ser usado para construir productos complejos a un menor costo. El futuro del Raspberry Pi es brillante. Robert Mullis, uno de los fundadores de la fundación Pi dice: "Estamos usando la tecnología que normalmente se vende en cajas negras. Un smartphone moderno ahora tiene
4 procesadores, cada uno corriendo en gigahertz y es una época excitante para transferir la tecnología del móvil hacia la mano de los entusiastas de la tecnología" (Edwards, C, 2013).

Objetivos

Objetivo General:

Integrar el sistema embebido DSP de audio para emular un “Pedal Digital de Guitarra”, un procesador de efectos que se pueda controlar a través de una interfaz gráfica.

Objetivos Específicos:

-Diseñar los filtros que permitan la construcción del procesamiento que se busca con todos los parámetros de control necesarios.

-Familiarizarse con la programación y manejo de los circuitos destinados a usarse, es decir el microprocesador Raspberry Pi y el módulo de audio HiFi Berry DAC+. Comunicar el Raspberry Pi con los módulos necesarios para integrar entradas y salidas de audio.

-Evaluar el alcance de complejidad que se puede explotar desde la integración de estos dos dispositivos electrónicos.

-Conseguir integrar estos dispositivos de forma inalámbrica, de manera que el usuario pueda controlar el sistema integrado desde cualquier dispositivo con capacidad de acceder a internet.

-Desarrollar una interfaz de usuario amigable y eficiente al aplicar los algoritmos DSP que permitan una manipulación de audio de alta calidad.

Revisión Literaria

Debido a la continua innovación en sistemas de comunicación muchas de las subindustrias de la música han tomado ventaja de este arranque de la tecnología para ofrecer productos que beneficien tanto a músicos, a productores y a técnicos de escenarios para ofrecer espectáculos de música de una calidad increíble. En la revista online CreateDigitalMusic en lo referente a la tecnología de las guitarras eléctricas Peter Kirn expone: "¿Estamos entrando a una era en donde toda guitarra, amplificador y pedal en un efecto en cadena se transformaran a grandes sistemas de procesamiento y entorno de edición? O se simplificará todo a un solo dispositivo "*i"*. A continuación se detalla algunos de estos productos que constituyen nuevas tendencias que servirán de base para proponer la aplicación final del proyecto de titulación.

WIFI JACK

A inicios del 2015 el wireless guitar tool WIFI JACK salió a la venta. Este dispositivo permite al músico conectarse inalámbricamente a cualquier dispositivo de salida de audio. Por ejemplo a un típico amplificador de guitarra, a una computadora o a un dispositivo móvil. Este producto ha sido diseñado específicamente para guitarristas y bajistas debido a que este tipo de músicos son los que necesitan una mayor libertad en el escenario para movilizarse de un lado a otro. Como herramientas adicionales este dispositivo puede acceder fácilmente a un software de edición y grabación localizado en una computadora o dispositivo móvil. El WIFI JACK a diferencia de otros sistemas tiene 24 bits de resolución y es descomprimido, compatible con varios dispositivos wireless, con una gran banda ancha y la capacidad de conectarse a múltiples dispositivos al mismo tiempo, además la batería se carga con un cable USB estándar. Es convencionalmente 13 veces más rápido que una conexión bluetooth y 3 veces más rápido

que una conexión “low latency” bluetooth comprimida. Su precio actual es aproximadamente $280 en el mercado y su proceso de diseño y construcción duró 4 años hasta lanzar su prototipo. Ver figura I.1.

Dispositivos “i”.-término para las nuevas tendencias de la tecnología móvil en el estado del arte

Figura I.1: Wifi JACK

En el mercado de la música esta es una gran innovación que permite una proyección de audio en tiempo real entre instrumentos y sus dispositivos de amplificación de audio sin la necesidad de routers, o cables o dispositivos procesadores de audio extras como tarjetas de audio o cajas de conectores. Todo esto sin perder la calidad del sonido. Para la mayoría de músicos de performance usualmente un jack cable para la entrada y salida de audio tiende a ser frágil especialmente cuando se tiene que transportar de un lugar a otro y en el momento de cualquier performance siempre tiende a limitar al músico en el escenario por la longitud del cable.

PEDALES DE GUITARRA WIFI

Un pedal de guitarra es un dispositivo análogo o digital que permite al músico cambiar el sonido que emite la guitarra. Estos efectos producidos por el procesamiento de la señal pueden ser de delay, distorsión, compresión, etc. La combinación de los principales efectos que se pueden realizar permite una gama infinita de posibilidades de ecualización en la salida en tiempo real. Comúnmente esto dispositivos han necesitado una conexión cableada para la

fuente, la salida y la entrada y debido a que el músico no puede llevar el control a la mano siempre ha optado por cambiar los efectos con el pie. Sin embargo en los últimos años se han desarrollado sistemas innovadores que incluyen comodidades como conexión wireless o menor peso. Por ejemplo el H9 de Eventide es un procesador de sonidos que corre algoritmos DSP. La salida de una guitarra es conectada con un cable jack al H9 pero desde aquí este dispositivo puede ser conectado a un dispositivo móvil para ser manipulado remotamente con una aplicación iOS, es decir el usuario puede cambiar el efecto de salida desde un móvil. Aun así el H9 sigue teniendo la necesidad de conectarse con cable jack a un amplificador. Cuesta aproximadamente 500 dólares en el mercado actualmente. Ver figura I.2.

Figura I.2: H9 Eventide

De modo similar el iRIG BlueBoard lleva el concepto de pedaleras inalámbricas a otro nivel. Este dispositivo es capaz de enviar mensajes MIDI a través de una conexión bluetooth que llegan a una aplicación móvil. Esta aplicación móvil permite procesar estos mensajes y transformarlos a cualquier tipo de salida. El Blueboard funciona inalámbricamente y puede ser conectado a una fuente o funcionar con batería, sin embargo la conexión con el dispositivo móvil es inalámbrica via bluetooth. La señal de entrada que puede ser de una guitarra está conectada al dispositivo móvil, en este caso a un dispositivo apple con un adaptador especial.

Luego la salida del audio se hará desde el dispositivo móvil para ser reproducida desde el mismo móvil o a un parlante. En el mercado cuesta aproximadamente 100 dólares. Ver figura I.3:

Figura I.3: iRIG Blueboard

Amplitube de ikmultimedia es un procesador profesional de múltiples efectos y estudio de grabación y edición completo. Transforma a cualquier dispositivo basado en iOS en un procesador móvil de efectos múltiples de guitarra que ofrece versiones digitales de los amplificadores más conocidos. En su nivel más básico es un equipo de guitarra que incluye un sistema de tres switches que simulan pedales análogos y un simulador de amplificador.

Tomando en cuenta estas instancias de nuevos dispositivos y centrándose en la industria de los dispositivos electrónicos que permiten la comunicación entre instrumentos electrónicos y su amplificación se plantea la metodología del proyecto de titulación a continuación.

Metodología del proyecto

-Se selecciona un efecto original tomado de una fusión de los efectos más usados de la industria y con todos los parámetros completamente controlables.

-Se investiga los algoritmos DSP que se requieren para producir el efecto deseado además de sus características de procesamiento, por ejemplo se pretende saber la forma de onda final de la señal, el comportamiento, la frecuencia, etc.

-Se selecciona el hardware requerido para el procesamiento de audio de acuerdo a los parámetros de calidad como por ejemplo la resolución análogo-digital. Como candidato principal de control se pretende usar el Raspberry Pi y como candidato de procesamiento se pretende usar uno de sus módulos, el HiFiberry DAC+ porque posibilita la resolución de audio A/D y D/A hasta 44Khz que es más de lo que necesitamos en términos de calidad de audio.

-Se integra completamente el microprocesador Raspberry Pi, el módulo de audio HiFiBerry DAC+ y conexión a internet para descargar e instalar los softwares requeridos para el desarrollo de la aplicación.

-Integrar la entrada de audio a una amplificación y probar los controles para los cambios del procesamiento desde la interfaz gráfica.}

-Completar módulos de audio necesarios para acondicionar la señal de entrada y obtener una salida de alta calidad.

LISTA DE ACRÓNIMOS

Notches

Puntos mínimos en un espectro de frecuencias producidos artificialmente para producir ciertas características en una señal procesada.

Pick

En la industria de la música son herramientas para golpear los instrumentos de cuerdas. Comúnmente son pedazos de plástico de cierta forma y grosor que al ser tocados en las cuerdas producen sonido.

Jack

Son las terminales de los cables de audio que se usan comúnmente en la industria d ela música. Se conocen también como plugs. En el proyecto se usaron varias denominaciones de este tipo de cables.

Pick Ups

Micrófonos internos de los instrumentos eléctricos de cuerdas. Funcionan a base de vibraciones electromagnéticas y son las únicas responsables de producir el sonido de un instrumento eléctrico de cuerdas.

CAPÍTULO 1: SISTEMAS EMBEBIDOS Y FUNCIONAMIENTO GENERAL DEL RASPBERRY PI

1.1 Concepto de los Sistemas Embebidos

Un sistema embebido es un sistema de computadora con una función dedicada a una aplicación mecánica o eléctrica. En estos días muchos de los dispositivos que usamos usan sistemas embebidos. Las principales ventajas de los sistemas embebidos incluyen bajo consumo de potencia, tamaño reducido, varios rangos de operación y bajo costo por unidad. La mayoría de los sistemas embebidos que se usan hoy en día son hechos a base de microcontroladores y microprocesadores, especialmente en sistemas más complejos. La clave principal de estos sistemas es que desempeñan una función específica. Los diseñadores pueden optimizar estos sistemas, reducir su tamaño y costo de producción e incrementar la confiabilidad y desempeño de los mismos. En gran escala los sistemas embebidos producen economías de escala(Definition of embedded systems, 2014).

La mayoría de estos sistemas usan versiones de los siguientes softwares o sistemas operativos para cumplir sus funciones: Linux, Windows, Mac, así como también otros sistemas operativos comerciales y propietarios especializados en sistemas embebidos. En general estos softwares contienen firmware(software permanente que se encuentra en una memoria de lectura) que están alojados en una memory flash o alguna memoria de lectura. Típicamente estos sistemas usan botones, pantallas LCDs o interfaces gráficas como sistemas de control. En la vanguardia de estos sistemas se están desarrollando maneras de usar

interfaces remotas con ayuda de sistemas de comunicación tales como una red wifi , protocolos seriales cableados, etc. (Llanos, Diego, 2014).

1.2 Comunicación de los sistemas embebidos

Una de las preocupaciones concernientes a los sistemas embebidos es su comunicación con otros sistemas. Estos sistemas pueden hablar con el mundo exterior vía periferias como:

- Serial Communication Interfaces (SCI): RS-232, RS-422, RS-485 etc.
- Synchronous Serial Communication Interface:I2C, SPI, SSC and ESSI (Enhanced Synchronous Serial Interface)
- Universal Serial Bus (USB)
- Multi Media Cards (SD Cards, Compact Flash etc.)
- Networks: Ethernet, LonWorks, etc.
- Fieldbuses: CAN-Bus, LIN-Bus, PROFIBUS, etc.
- Timers: PLL(s), Capture/Compare and Time Processing Units
- Discrete IO: aka General Purpose Input/Output (GPIO)
- Analog to Digital/Digital to Analog (ADC/DAC)
- Debugging: JTAG, ISP, ICSP, BDM Port, BITP, and DP9 ports.

En general los sistemas embebidos requieren residir en máquinas que resistan y operen por largos períodos de tiempo. es por eso que un software de un sistema embebido es usualmente más desarrollado y probado más cuidadosamente.

1.3 Raspberry Pi

El Raspberry Pi es una computadora del tamaño de una tarjeta de crédito que se conecta a un display, un teclado y un ratón para su control. Su uso permite que usuarios de todas las edades aprendan a programar en lenguajes como Scratch o Python. Tiene una estructura ARM y que incluye una interface de entradas y salidas I/O e interfaces de almacenamiento, incluye una salida HDMI, 4 puertos USB, un puerto Ethernet, un slot para una memory SD y una fuente de alimentación con puerto mini USB. Al tener una conexión con el mundo exterior ha sido usado en una gran variedad de proyectos de todo tipo incluyendo máquinas de música, detectores de temperatura para el clima, control de cámaras, en el mundo de la domótica, etc. El Raspberry fue creado por la fundación Raspberry Pi como un proyecto educativo en UK con el objetivo de mejorar la educación de adultos y niños en el campo de las computadoras y tópicos relacionados. (Raspberry Pi Org).

Figura 1.1: Raspberry Pi

1.4 Software y Hardware de Raspberry Pi

1.4.1 Programas Kernel

Kernel general es un programa de computadora que maneja órdenes de entradas y salidas I/O(input/output) desde un software y los traslada a instrucciones de procesamiento de datos hacia la unidad de procesamiento central y otros componentes electrónicos de una computadora. Un programa Kernel es una parte fundamental del sistema operativo de una computadora. Para que el desempeño de un programa Kernel se usa un espacio separado de la codificación del usuario, como por ejemplo en un editor de texto o programas GUI (Interfaces Gráficas de Usuario). Esta separación es necesaria para evitar que los datos del usuario y los datos kernel interfieran disminuyendo el desempeño o causando daños en el sistema hasta volverse inestable. Para el presente proyecto se usarán scripts python para correr los programas que se construirán en lenguaje básico. El sistema Raspbian posee varios programas que permiten escribir estos scripts.

1.4.2 Raspbian basado en Linux

El sistema que se escogió para el proyecto es el sistema Raspbian. Es un sistema operativo libre basado en Debian y optimizado para el hardware del Raspberry pi. Viene con más de 35000 paquetes precompilados y con una fácil instalación. Actualmente es uno de los sistemas más usados para el Raspberry Pi aunque sigue en desarrollo, se lo puede apreciar en la siguiente figura 1.2.(Rasbpian Org).

Figura 1.2: Sistema Raspbian para Raspberry Pi.

1.5 Seteo del Raspberry Pi

Para instalar el sistema operativo del Raspberry Pi se requieren los siguientes recursos:

-Micro SD card: mínimo 4GB

-Un display para visualizar la microcomputadora como por ejemplo una TV o algún proyector con entrada HDMI o algún adaptador del mismo tipo

-Un teclado y mouse con puertos USB

-Un cable con puerto mini USB para la alimentación del Raspberry

Para instalar el sistema operativo en el Raspberry Pi(Raspbian) consultar los pasos en la siguiente dirección:

http://www.raspberrypi.org/help/quick-start-guide/

http://www.raspberrypi.org/documentation/installation/installing-images/README.md

1.5.1 Características del hardware del Raspberry Pi.

El proyecto usará un Raspberry Pi Model B+512 MB. Las salidas básicas del Raspberry Pi son:

-700 Mhz Broadcom BCM2835

-40 pin extended GPIO

-HDMI input

-4 USB ports

-A micro SD card slot

1.6 Wifi del Raspberry Pi para el internet

Debido a que el presente proyecto requiere de librerías Python que no se encuentran en el sistema por default se necesita conexión a Internet para actualizar los paquetes de librerías necesarios. Ya que el sistema Raspbian es basado en Linux, se sigue un proceso similar a otros sistemas operativos basados en Linux como Ubuntu por ejemplo. es decir se hace uso de la terminal para introducir comandos que permiten ver el estado de conexión del dispositivo. Se usó la interfaz gráfica de conexiones del Raspbian y los siguientes comandos para ver el estado de conexión del dispositivo.

```
$ifconfig
```

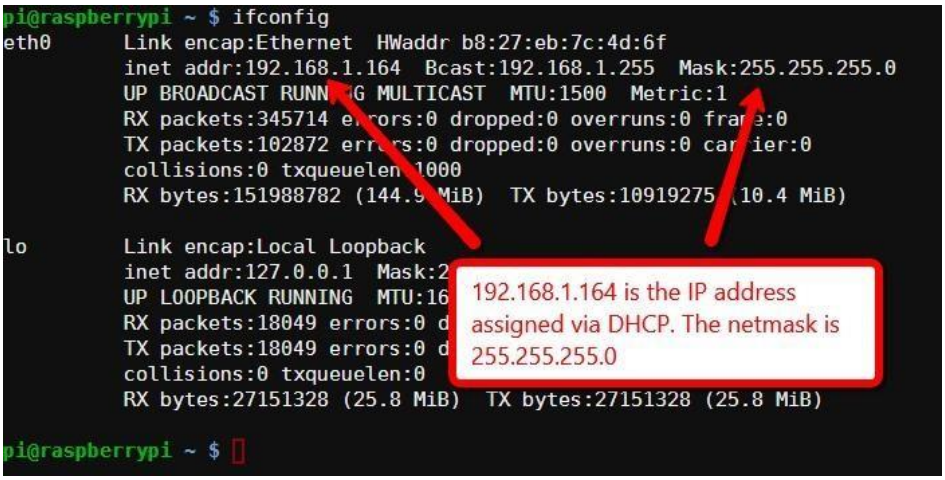


Figura 1.3: Comando para ver las conexiones disponibles

Para el proyecto se necesita un dispositivo USB adaptador de wifi que permite al dispositivo conectarse a la web sin necesidad de un cable ethernet y sin necesidad de instalación de drivers como la que se muestra en la siguiente figura 1.4.

Figura 1.4: Adaptador USB Wifi

Una vez conectado este adaptador se puede ingresar a la interfaz gráfica del sistema Raspbian para seleccionar una conexión específica. Como se ve en la figura 1.5

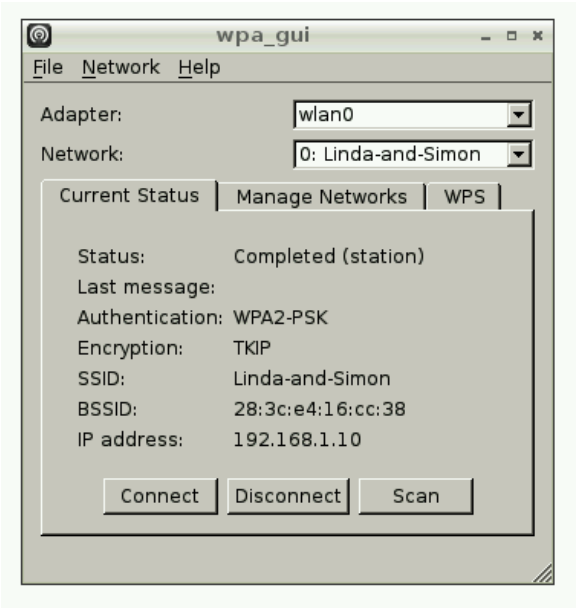


Figura 1.5: Interfaz gráfica para conectarse a una red wifi

CAPÍTULO 2: CONCEPTOS GENERALES DE PROCESAMIENTO DE AUDIO

2.1 Funcionamiento de las guitarras eléctricas y frecuencias relevantes

Las señales de audio de las guitarras eléctricas son creadas por las cuerdas que tienen una tensión específica y cuando son golpeadas por los dedos del músico o algún instrumento de plástico u otro material (picks) causan oscilaciones. Estas oscilaciones de las cuerdas metálicas inducen un campo eléctrico en las bobinas magnéticas inductivas de la guitarra, estos "micrófonos"(bobinas inductivas dentro de una caja) se denominan *pickups*. El campo eléctrico induce una corriente sinusoidal en las bobinas de los cables alrededor de los pickups, la cual viaja desde los cables de las bobinas a través de una red de potenciómetros de volumen y tono y finalmente a la salida del jack de la guitarra como se puede ver en la figura 2.1. La amplitud de estas señales eléctricas varía entre 140 mV y 1.4V y cambia en frecuencia dependiendo de las frecuencias fundamentales de las notas tocadas y sus respectivos armónicos (Osgood Nathaniel, hong June-chi and Richardson, 2008).

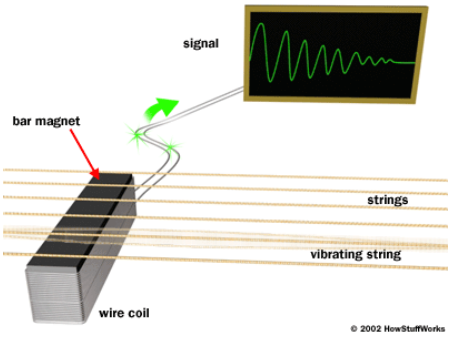


Figura 2.1: Creación del sonido desde los pickups de la guitarra eléctrica

2.1.1 Conceptos generales de las características de las notas musicales

Cuando un instrumento es tocado, es decir cuando se golpea algo, se ve algo o se sopla algo, las características del diseño físico de este instrumento le da todas las a los sonidos que emite(Esto también se aplica al oído humano). Cuando se toca una nota(tono), por ejemplo C4(nota media en el piano), este tiene un pitch. Este es la frecuencia fundamental. En el caso de C4 es 262Hz lo cual significa que envía ondas de sonido a esta frecuencia. El sonido característico de cada instrumento crea lo que se conoce como timbre. La acción de crear sonido crea múltiples armónicos y overtones y lo que es llamado como Envolvente de Amplitud o en otros términos técnicos Envolvente ADSR(Attack, Decay, Sustain, Release), señal que se puede apreciar en la figura 2.2.

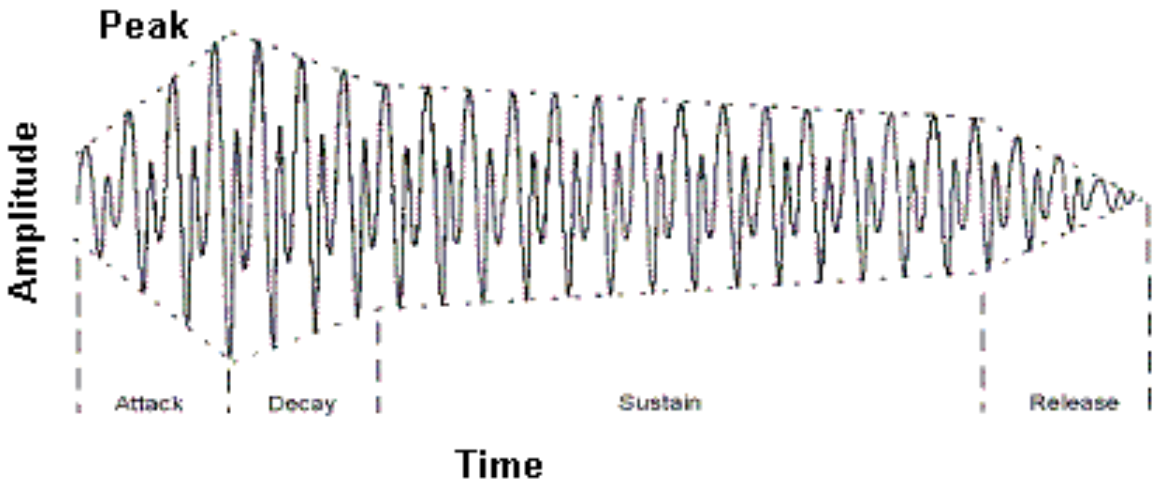


Figura 2.2: Vista en el tiempo de la envolvente ADSR

Cada instrumento tiene una envolvente ADSR diferente (en sus frecuencias fundamentales, armónicos, overtones, etc), es decir, por ejemplo, el tiempo de sustain podría ser más prolongado, el tiempo de ataque diferente, etc.

Timbre.-Cada voz humana tiene un timbre diferente, por esa razón, cada individuo tiene un timbre diferente de voz

Para entender de mejor manera cómo funcionan las notas que representan a las frecuencias que se buscan procesar, se deben tener en cuenta algunos conceptos, los cuales se detallan a continuación.

ADSR (Envolvente de Amplitud).- Es el cambio de en amplitud (mientras la frecuencia permanece constante) en el tiempo y está compuesta por tiempos de ataque, decaimiento, sustain y liberación (ver figura 2.2).

ARMÓNICO.- Cuando una nota es tocada, se genera un pitch (frecuencia fundamental). Pero el instrumento también genera ondas secundarias (armónicos), las cuales tienen pitches (frecuencias fundamentales) múltiplos de 2, 3 etc. de la frecuencia fundamental. A medida que se generan más de estos armónicos se vuelven más débiles en potencia. Cada armónico también posee una envolvente ADSR.

OVERTONE.- Son ondas de sonido que se crean debido al diseño del instrumento pero que no son múltiplos de la frecuencia fundamental(a diferencia de los armónicos), por ejemplo con múltiplos 1.3 o 7.4 de la frecuencia fundamental. En términos de audio es siempre preferible conseguir armónicos a overtones porque los armónicos tienen una relación íntegra con las frecuencias fundamentales al ser múltiplos.

PITCH.- La frecuencia fundamental de un sonido o una nota.

TIMBRE.- Se conoce como la calidad del tono y se conforma por la presencia de la amplitud de los armónicos y overtones que contiene esa nota y la forma de la envolvente de amplitud (ADSR) que posee esa nota (frecuencia fundamental).

POTENCIA DE SONIDO.- El sonido es medido normalmente en dB (medida que no tiene unidad). En acústica, lo que se conoce como dB, significa normalmente SPL (Nivel de presión de sonido). En el mundo digital SPL es representado en número negativos, en donde 0dB es el punto en donde se puede producir distorsión después de que el sonido haya pasado a través de un convertidor(DAC-convertidor digital a análogo) y luego haya sido amplificado en un sistema de reproducción de salida.

2.2.1 Rangos de las notas musicales

La mayor parte de aplicaciones en procesamiento de audio manejan notas musicales; se analizan las siguientes 10 octavas que cubren el oído humano (normalmente entre 20Hz y 20Khz). La siguiente tabla muestra la frecuencia de las notas musicales de más de 10 octavas (una octava es el conjunto de 8 notas en el sistema pitagórico diatónico, ver en la sección de análisis de señales, cada nota está en Hertz). Esta tabla se basa en un modelo que se denomina Modelo Americano Estándar del Pitch donde la nota A4=440hz. Este estándar usa un *intervalo temperado* en cual cada nota es relacionada a la siguiente en una cantidad igual. Cada octava está compuesta por 12 seminotas. Ver figura 2.3:

Note	0	1	2	3	4	5	6	7	8	9	10
C	16	33	65	131	262	523	1047	2093	4186	8372	16744
C#	17	35	69	139	277	554	1109	2217	4435	8870	17740
D	18	37	73	147	294	587	1175	2349	4699	9397	18794
D#	19	39	78	156	311	622	1245	2489	4978	9956	19912
E	21	41	82	165	330	659	1319	2637	5274	10548	21096
F	22	44	87	175	349	698	1397	2794	5588	11175	22351
F#	23	46	93	185	370	740	1480	2960	5920	11840	23680
G	25	49	98	196	392	784	1568	3136	6272	12544	25088
G#	26	52	104	208	415	831	1661	3322	6645	13229	26580
A	28	55	110	220	440	880	1760	3520	7040	14080	28159
A#	29	58	117	233	466	932	1864	3729	7459	14917	29832
B	31	62	123	247	493	988	1976	3951	7902	15804	31604

Figura 2.3: Clasificación de las frecuencias que representan cada nota en Hz.

En la siguiente figura 2.4 se puede comparar el alcance que tiene la guitarra como instrumento en relación con otros instrumentos y cuáles son las notas que representan estas frecuencias.

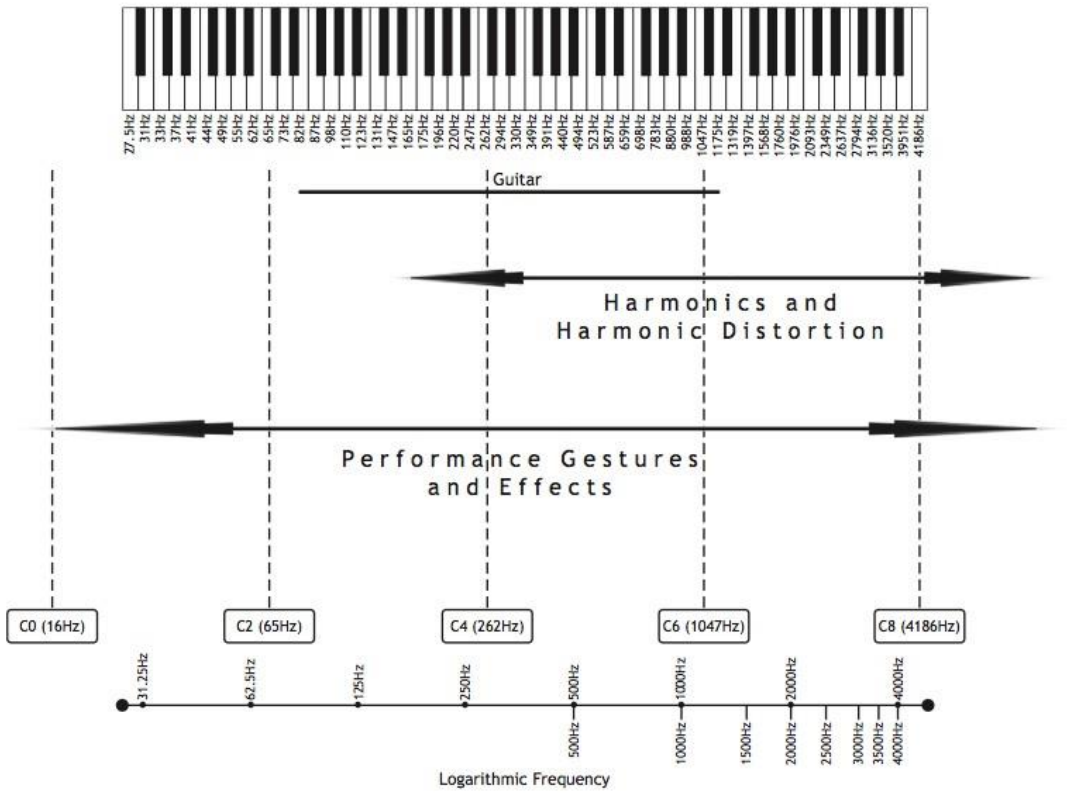


Figura 2.4: Comparación de las notas que se pueden escuchar en un instrumento de cuerdas.

2.1.3 Frecuencias de Audio

Así como las frecuencias fundamentales, la mayoría de instrumentos tienen armónicos y overtones, las cuales son muy difícil de encontrar para su manipulación. Aun así se predice que el rango de frecuencias fundamentales (tonos) de una guitarra eléctrica es de E2(82Hz) a F6(1.397Hz). Este rango también se aplica a las guitarras acústicas. El rango de frecuencias audibles de un amplificador de guitarra eléctrica está entre 100Hz y 6.2KHz y el rango de las frecuencias de las notas disponibles en el brazo de la guitarra es de 82Hz(E2) a 1397Hz(F6) como se mencionó anteriormente. Aun así, los humanos somos más sensibles al sonido en el rango de 400Hz a 5KHz con una sensibilidad particular entre 2 y 4Khz. Esta sensibilidad

podría atribuirse a la evolución de la especie que percibe sonidos de peligro o circunstancias no placenteras.

2.2 Diferencia entre señal de audio análoga y señal de audio digital en pedales de guitarra

Un pedal de efectos para guitarra es un dispositivo análogo o digital que permite procesar la señal de entrada y transformarla para que al salir tenga un sonido diferente en el amplificador. Algo importante de mencionar es la diferencia entre un pedal análogo y un pedal digital en la calidad del sonido aún si procesan el mismo efecto. Por ejemplo un Boss Gt-10 (Figura 2.5) es uno de los pedales digitales más usados en el mercado, al tener un procesamiento digital permite seleccionar cientos de efectos diferentes, a diferencia de, por ejemplo, de los pedales Boss análogos individuales (Figura 2.6) que procesan un sólo efecto por pedal. Al ser análogos necesitan el espacio físico del circuito que procesa la señal. La diferencia está en la calidad del sonido. En los pedales digitales, a pesar de tener un amplio rango de efectos, su sonido es diferente. Para los músicos es muy “digitalizado”, no “no natural”; mientras que los pedales individuales, al estar dedicados a un sólo efecto, este tiene un alto grado de calidad y de control. Por esta razón la mayoría de los músicos profesionales prefieren los pedales análogos a los digitales.

Figura 2.5 Pedal Digital Boss Gt-10

Figura 2.6 Pedales Boss individuales análogos

2.2.1 Diferencia de señales análogas y digitales

La principal diferencia entre señales análogas y señales digitales es que la señal análoga no tiene quiebres(es continua) y que la señal digital se compone de puntos individuales (discretos). Un buen ejemplo de una señal análoga es la voz humana que es continua y no tiene quiebres. Por lo tanto una señal digital producida por cualquier dispositivo tendrá desventajas que los pedales análogos nunca tendrán. Por ejemplo un pedal fuzz análogo suena mucho mejor porque el circuito reacciona a los cambios de voltaje mientras el músico toca. Ya sea

una señal eléctrica de un instrumento o un sonido acústico generado por un instrumento de manera son igualmente señales análogas. En la siguiente figura se presenta la señal análoga (negra) y la señal digital(azul). Ver figura 2.7:

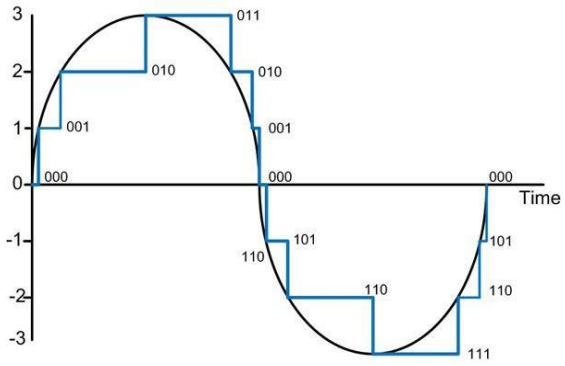


Figura 2.7: Concepto de muestreo en una señal analógica.

Cuando alguna señal es oída por ejemplo a 12 bits, significa que la señal análoga está dividida en 12 bits. Este proceso es conocido como resolución. Aun cuando nuestro oído no puede notar la diferencia un pedal digital de guitarra nunca podrá muestrear la señal hasta compararse con una señal continua que un pedal análogo de guitarra ofrece. Otra desventaja que tienen los procesos de audio digitales es que si la señal no es muestreada de una manera correcta puede producir pérdidas y aliasing de frecuencia. Un procesador digital por ejemplo toma una muestra cada 10 microsegundos y es grabada en la memoria del pedal. Una unidad común para medir la división de la señal análoga es el Hertz que es muestras/segundo. En este ejemplo de 10useg, significa que el tiempo de muestreo es 1/10usec o 100Khz. Lo importante es que nuestro oído solo puede percibir frecuencias debajo de los 20KHz, por eso es bien importante muestrear la señal lo suficientemente rápida para que se complete la señal. Además cuando no se muestrea adecuadamente se pueden producir efectos de aliasing. Aliasing se produce cuando el muestreo no es suficiente rápido para reconstruir la señal

produciendo una señal de diferente frecuencia y por lo tanto un sonido diferente. Si por ejemplo se quiere muestrear la señal de 10KHz a 10KHz, es decir una muestrea cada 0.0001segundos, se obtiene una señal completamente diferente de 500Hz. Ver figura 2.8.

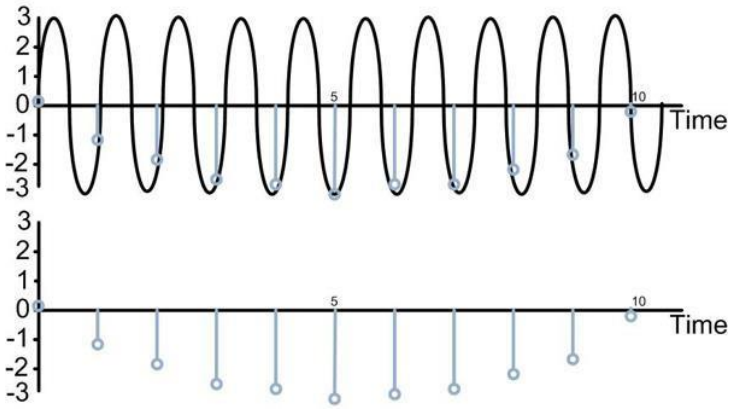


Figura 2.8: Representación del aliasing cuando la señal no es correctamente muestreada.

El teorema de Nyquest sugiere que para lidiar con este problema, se debe muestrear a la señal al menos con el doble de la frecuencia de entrada. Entonces para corregir la señal de 10KHz se necesita muestrear a mínimo 20KHz y a un muestreo mayor si se quiere calidad de sonido. Los CDs de audio en la actualidad reproducen a 44Khz de calidad que quiere decir que es un poco más que 20Khz que es lo que el oído normalmente percibe.

2.3 Latencia y calidad de audio

LATENCIA.- es el intervalo de tiempo entre la simulación o procesamiento y la respuesta, o en general es un retardo de tiempo entre la causa y el efecto. Es consecuencia de la velocidad limitada con cualquier interacción física que se propague. En sistemas de comunicación la latencia limita el máximo ratio en que la información puede ser transmitida. Los posibles contribuidores que hacen que exista una mayor latencia en el procesamiento de audio pueden

ser los procesos de conversión análogo digital, buffering, procesamiento de la señal digital, tiempo de transmisión, conversión digital análogo, y la velocidad del sonido en el aire.

BAJA LATENCIA.- Una latencia baja permite que los delays entre la señal de entrada que es procesada y la correspondiente salida no sean notables al oído humano proporcionando características de tiempo real.

2.4 Selección de efectos más usados

BenchMarking

Para evaluar el mercado en el que este producto tendrá un lugar se debe analizar la posición de hardware y software similares en productos competitivos en otras compañías. Se debe analizar el alcance comercial de este producto en términos de calidad, tiempo, desempeño, etc. Según la revista TopTepReviews Online(2015), entre los mejores pedales digitales de guitarra del 2015 figuran algunos modelos como: el POD HD500 de Line6, el DigiTech RP1000 de Harman, el BOSS Gt-10 de Boss. Se analizarán estos tres pedales digitales físicos en términos de hardware debido a que son los más conocidos en la industria por este ranking y por experiencia personal. Sin embargo estos tres pedales necesitan conexión alámbrica, el controlador se encuentra en el mismo hardware y debido a su capacidad de procesamiento en general son un poco pesados. Además se añade a la lista de análisis el BlueBoard de IkMultimedia que a diferencia de los otros es un controlador de una simple aplicación de móvil que convierte al dispositivo móvil en el procesador de audio. La aplicación BlueBoard iOS permite configurar que cambios de programa y control se quieren asignar a cada botón o banco, luego se cambia los seteos a través de cualquier aplicación MIDI, la aplicación BlueBoard corre en el background recibiendo señales desde el pedal y enviando mensajes MIDI apropiados de regreso. Se añade a la lista este dispositivo porque

tiene muchas similaridades con el objetivo del proyecto que se desarrollará. Con estas aclaraciones se muestra la siguiente tabla de especificaciones técnicas para un posterior análisis:

Nombre del producto	**Capacidad de Hardware**	**Características**	**Mejores efectos calificados**	**Costo**
POD HD500 de Line6	-A/D 24bit -D/A 24bit -44.1KHz, 48KHz, 88.2 KHz, 96KHz	-22 HD Amp models -120 effects -512 presets -Software Editor	-19 delays -23 modulaciones -17 distorsiones -12 compresores -26 filtros(Wah) -12 reverbs	500$ en USA en Amazon
DigiTech RP1000	-A/D 24bit -D/A 24bit -44.1KHz sampling rate -1 Audio DNA2(TM) DSP processor	-160 effects -200 presets -Stompbox -55 Amps	-4 filtros Wah -3 Compresores -2 Noise Gate -21 Distorsiones -7 Chorus -3 Phaser -6 Flanger -5 Pitch -5 Vibrato -4 tremolos -7 envelop -11 Delay -6 Reverb	400$ en USA en Amazon
BOSS GT-10	-A/D 24 bit +AF -A/D 24 bit D/A -Muestreo: 44.1 Khz -impedancia de entrada: 1 M ohm, de retorno: 22 Kohms -Fuente de poder: DC 9V	-400 amps models: 200 presets + 200 por el usuario -40 sec looper -zTone	-26 Distortions and overdrive -11 Delays -7 filters wah	-500$ USA en Amazon
BlueBoard IkMultimedia	-Aplicación iRig para enviar mensajes MIDI a aplicaciones iOS -Conexion bluetooh con la aplicacion iOS AmpliTube -El dispositivo mòvil es el procesador de audio	-MIDI Pedal Board -32 bancos que pueden regularse pero son presets en Amplitube -Afinador Ultra Tuner -Loop Drummer	-Delay -fuzz -overdrive -wah -envelope filter -chorus -flanger -phaser -octave -filtro de ruido + distorsion -5 amplificadores	100$ USA en Amazon. La gran desventaja es que solo es compatible para iOS y disponible para iOS.

Una vez analizadas las características de desempeño de hardware y beneficio/costo de los productos más competitivos del mercado se procede a aclarar el árbol de clasificación de los efectos existentes, los cuales se relacionan con otros en términos de algoritmos DSP, más difíciles de diseñar y construir.

2.5 Clasificación de efectos de guitarra

Tomando en cuenta el modo en que la señal es procesada y la diferencia entre la señal de entrada y salida al final del procesamiento, se pretende categorizar los efectos digitales de guitarra en las siguientes familias:

1. REORGANIZACIÓN DE LA SEÑAL

-Distorsión, overdrive.- Se caracteriza porque los picos de las ondas de la señal se entrelazaban las unas a las otras. En realidad es un ruido de señal controlado dentro de ciertas bandas permitidas.

-Compresión.- Se puede ver como un “limitador” y es cualquier circuito que prevenga que la señal de entrada exceda cierto límite. De alguna manera limpia posibles distorsiones y progresivamente reduce la amplificación de la señal entrada. Reduce el rango dinámico de la ganancia de señales de alta amplitud y manteniendo la ganancia diseñada para señales de amplitud más bajas.

-Modulación.- Toma la señal del instrumento y añade una segunda señal desde un oscilador local o señal fuente. Las dos señales sumadas producen una suma y diferencias de frecuencias.

-Pitch Shifters.- Trasladan la frecuencia base pero necesita mucho procesamiento, se usan para efectos de cambio de nota o armonizadores.

2. AUMENTACIÓN DE LA SEÑAL

-Flanging.- Este efecto involucra dos copias de la misma señal que están desfasadas. Este desfase provocará un retardo pero no lo suficiente para producir dos ondas fuera de fase la una con la otra.

-Phasing.- Es similar al flanging pero sin desplazamiento de frecuencias. Produce un efecto de "remolino".

-Chorus.- Divide la señal en dos para que se encuentren en un ciclo aparte en fase, así se produce un "coro" como dos voces juntas.

-Reverb.- Adhiere copias de la señal original con retardos en tiempo tan cortos que caso no se perciben.

-Delay y Echo.- Adhiere una o más copias que se diferencian en fase y que están tan distanciadas que la forma de onda se repite pero va decayendo en amplitud en cada repetición.

3. FUZZ

Cae dentro de la categoría de distorsiones pero se diferencia también porque se usa un generador de onda cuadrada en la forma de un gatillo Schmitt para introducir una frecuencia

variable para acompañar o incluso reemplazar a la señal original. El circuito Schmiit usa una acción de comparación que depende de la salida. El circuito se puede entender como un multivibrador.

Con el análisis del benchmarking y una idea clara de los efectos que comúnmente se usan en la industria y los que se han usado a lo largo de la historia en guitarras eléctricas se pretende desarrollar la simulación de: Un efecto phaser que permita experimentar un sonido de barrido similar a un eco pero con un tiempo de retardo muy diminuto. Este efecto se produce por el desfase de la señal de salida que posee con respecto a la señal de entrada, cuyo comportamiento se describirá en posteriores capítulos. Adicionalmente se pretende añadir un efecto mínimo de FUZZ, es decir un tipo de distorsión suave como opción para el músico. Agregando a estos dos tramos del algoritmo se pretende conseguir un control completo del nivel de distorsión, del nivel de desfase y control de la ganancia(el volumen).

2.6 Algoritmos DSP

Para el presente proyecto que pretende simular un efecto phaser digital se necesitará diseñar un filtro que emule el efecto tomando los argumentos que el usuario ingresará en el programa. Para este efecto se usarán los conceptos de función de transferencias que nos guiará a encontrar la ecuación a diferencias. La ecuación a diferencias en el dominio del tiempo discreto es la que se adecúa a las necesidades de programación en lenguaje python por ejemplo. Con esto en mente el algoritmo que se pretende llegar a desarrollar es el siguiente:

-Diseñar el filtro PASA TODOS con los parámetros que controlan el nivel de profundidad del efecto.

-Implementar la ecuación a diferencias que representan el filtro en la programación respectiva.

-Desarrollar un programa que permita añadir distorsión a la señal de entrada. Este programa estará separado del filtro porque actuará como una opción del usuario. También poseerá argumentos de control para controlar el nivel de distorsión.

-Desarrollar todos los controles de la señal de salida incluyendo la ganancia (el volumen), los switches para los efectos y sus respectivos controles de nivel.

CAPÍTULO 3: ANÁLISIS DE LOS EFECTOS ESCOGIDOS

3.1 Efecto Phaser

El efecto phaser es un sonido creado por el procesamiento que se le da a una señal. Para conseguir este efecto se utiliza un filtro *pasa todo,* el cual permite que la señal resultante tenga un desfase con respecto a la señal de entrada pero la misma ganancia en todas las frecuencias activas. Para conseguir un efecto más pronunciado se incrementa el número de filtros pasa todo. En la industria de las pedaleras análogas y digitales el número de filtros pasa todo (llamados etapas) que se usan en un efecto phaser varían de acuerdo al modelo, algunos pedales análogos phaser *ofrecen 4, 6, 8, o 12 etapas. Algunos phasers digitales ofrecen 32 etapas o más*.

3.1.1 Historia del Efecto Phaser

Phasing es un efecto popular en la guitarra eléctrica. El término se usó al referirse a un efecto flanging oído en los récords de bandas psicodélicas en los 1960s. Uno de los primeros efectos phaser portátiles fue el MXR Phase. El cual fue lanzado en 1974 dio a conocer a la compañía. También este efecto se empezó a usar en órganos y pianos eléctricos en bandas de diferentes tipos de música. Comúnmente en la industria del cine o la televisión este efecto es usado para modificar la voz humana en una voz robotica.

Figure 3.1: MXR Phasers desarrollados por la industria desde 1974

Los resultados de un retardo de tiempo constante en la respuesta en frecuencia fue conocida por ingenieros en los inicios de 1900 pero se dice que el efecto fue descubierto accidentalmente por Phil Spectre en 1950s cuando estaba produciendo un récord y quería espesar las voces. El reprodujo dos copias de voces grabadas previamente y movió el carrete de cinta para retrasar una de las copias 1 bit. El resultado fue un sonido que además de producir una reverberación y ensanchar el sonido, desplegaba la misma respuesta de filtrado que los ingenieros ya conocían.

.

3.1.2 Primeros Efectos Phaser

Ya que era difícil reproducir este efecto en vivo al principio solo se usó este efecto en los estudios. Luego los ingenieros se percataron que los inductores y capacitores causan retardos de fase. Si se usa de manera adecuada una red de capacitor/resistor o inductor/resistor introducen un retardo de fase dependiente de la frecuencia sin la necesidad de los carretes de cinta. En la figura 3.2 se puede ver un esquema completo del procesamiento de la señal.

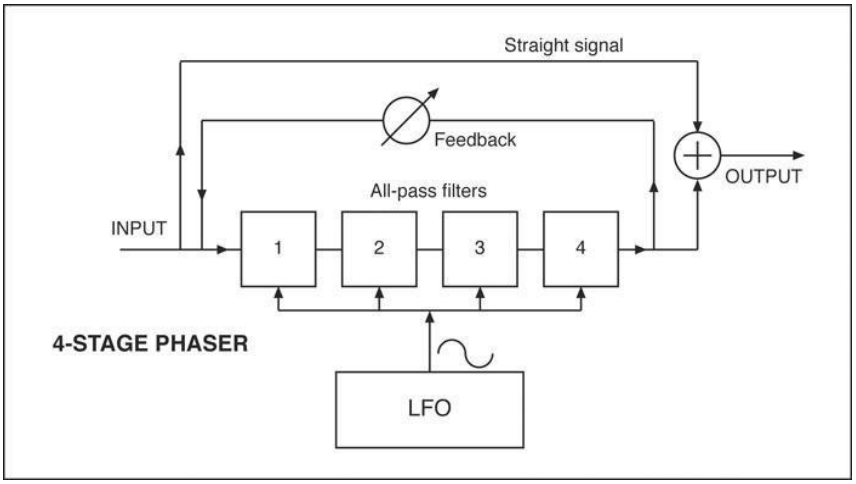
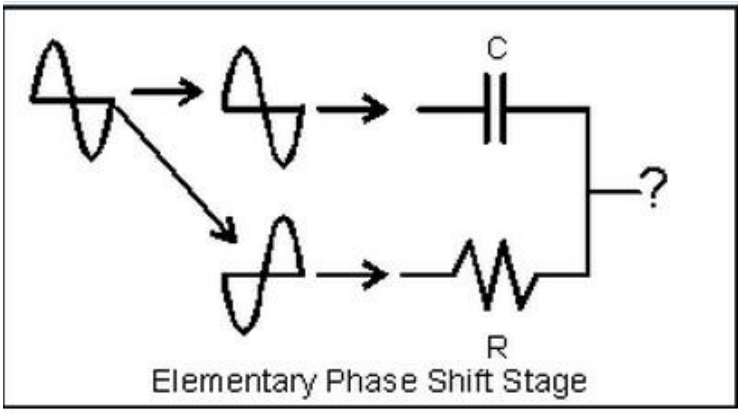


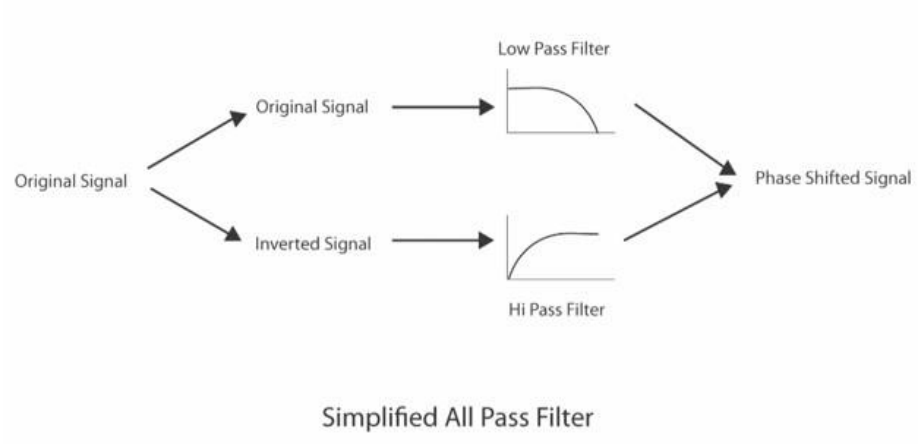
Figura 3.2: Esquema del procesamiento para producir el efecto phaser deseado

3.1.3 Electrónica del efecto phaser

Si se aplica el esquema de la figura 3.2 en un contexto análogo cada bloque representa una abstracción electrónica, por ejemplo un oscilador de frecuencia es usado como portadora, los bloques numerados son los filtros pasatodo antes mencionados. Es importante recalcar que el corazón del procesamiento se encuentra en estos filtros pasa todo. En la figura 3.3 y 3.4 se pueden apreciar una idea de lo que se produce dentro de ellos.



a.



b.

Figura 3.3: Esquema de inductancias y capacitancias que producen retardos de fase dentro del filtro. a) Representación analógica, b) Representación general de lo que se busca

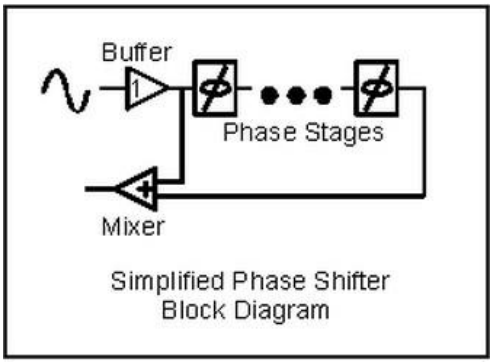


Figura 3.4: Esquema que resume el procesamiento analógico

Se producen dos copias de la señal, la original un la copia desfasada. Se utiliza un oscilador de baja frecuencia para modular la fase de una de estas señales. Una vez que las dos señales ingresan a los bloques de filtrado se produce el siguiente proceso. Se usa una capacitancia que actúa como un filtro pasa altos y un una resistencia que actúa como un filtro pasa bajos .La impedancia del capacitor se reduce al incrementar la frecuencia, es decir a altas frecuencias, el capacitor luce como un cortocircuito, y la señal alimentada a ello domina el resultado en la

unión del capacitor/resistor. En cambio, cuando las frecuencias de la señal disminuyen, la impedancia del capacitor aumenta. A muy bajas frecuencias, el capacitor tiene una impedancia más alta que el resistor, y la señal alimentada al resistor domina la señal en la unión de los dos caminos.

Si el resistor obtuvo la señal no invertida, y el capacitor la invertida, entonces en bajas frecuencias la señal resultante en la unión del resistor y la capacitancia es solo la señal no invertida. A frecuencias altas, la señal resultante en la unión es solo la señal invertida a través de la capacitancia. En señales medias, la señal invertida y la no invertida se añaden de un modo en que depende de la reactancia del capacitor vs la resistencia. Las matemáticas son un poco complicadas, pero se obtiene que la ganancia es la misma a la final del proceso. *Ver más detalles en el capítulo del análisis de la señal.* Aun así la fase es diferente y es controlada por una variable que depende de la frecuencia.

Si se oye la señal en la unión del resistor con la capacitancia, no se oye mucha diferencia porque la amplitud permanece constante mientras la fase cambia. Incluso si añadimos la señal original a la señal desfasada no es mucha la diferencia audible excepto un aumento o decremento de sobreagudo, dependiendo de si el capacitor obtuvo la señal invertida o no. Sin embargo, si colgamos dos de las instancias que provocan el retraso en serie(etapas u orden del filtro), encontramos que la fase en la frecuencia central se ha desfasado en 90 grados por el primer retardo y 90 por el segundo retardo. Entonces la salida de segundo retardo tiene un desfase total de 180 grados. Si añadimos a esto la señal original, ésta se cancela, las dos están totalmente desfasadas. Este es un efecto notable, por ejemplo el MXR(Figura 3.1) con fase de

45 grados trabaja así, con dos retardos concatenados y solo un notch (un solo pico y un solo mínimo en la respuesta en frecuencia, *ver detalles más adelante*). Los phasers más usados como por ejemplo, el MXR phase 90° tiene cuatro etapas(orden, número de filtros) o el MXR phase 100 con 6 etapas. Si se logra mover los notches para arriba y para abajo en frecuencia el efecto es mucho más notorio. Se puede lograr esto tomando en cuenta que cambiando el valor de la resistencia o el capacitor la frecuencia cambia de arriba hacia abajo. Esto produce el efecto deseado.

3.2 Comportamiento en la forma de onda y el espectro de frecuencias.

En esta sección se explicará con más detalle como la señal se comporta a lo largo del procesamiento. Si consideramos una sinusoide con un periodo de 360 grados o 2*pi*rad. de entrada y aplicando el filtro pasa todo la señal se duplicará en dos copias idénticas desfasando una de las copias y si el desplazamiento final de fase es 90°(por ejemplo), se puede ver el siguiente comportamiento en la figura 3.5.

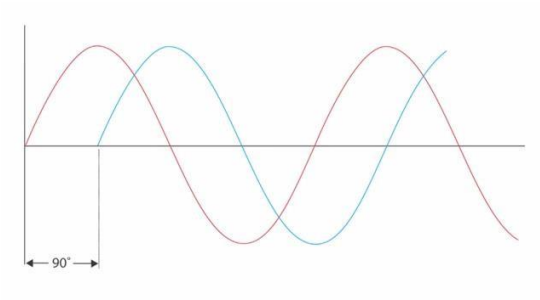


Figura 3.5: Desfase de dos señales sinusoidales con la misma frecuencia pero desfasadas.

En el caso de que el desplazamiento es 180° las ondas son reflejos la una de la otra y cuando se suman se cancelan completamente. En el procesamiento de audio, esta cancelación produce los notches característicos del efecto phaser, ver la figura 3.6:

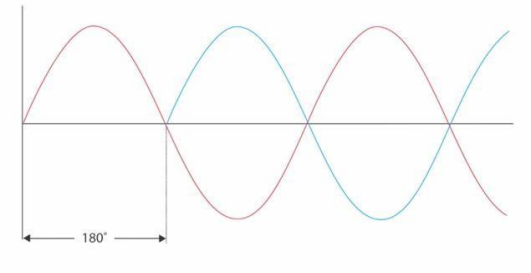


Figura 3.6: Desfase de 180º de dos señales sinusoidales con la misma frecuencia.

Si vemos el comportamiento de la señal resultante después de aplicar un filtro pasa todos, se puede apreciar en el espectro de frecuencias una serie de picos y mínimos. Los picos se producen porque en esas frecuencias la suma algebraica de las dos señales resultantes tienen el mayor valor y el mismo signo, en cambio los ceros mínimos se producen cuando los máximos valores son de signo contrario se elimina completamente. Si hablamos de ganancia se dice que los picos se oirán y los ceros serán las frecuencias que no se oirán. Se predice que estos picos y ceros son los responsables del sonido phaser que se pretende conseguir, a mayores picos, más pronunciado será el efecto. Como las señales son periódicas se predice que estos picos y ceros se producirán a lo largo del espectro de frecuencia siguiendo un patrón y en las frecuencias múltiplos de las primeras frecuencias. Se puede apreciar este comportamiento en la figura 3.7:

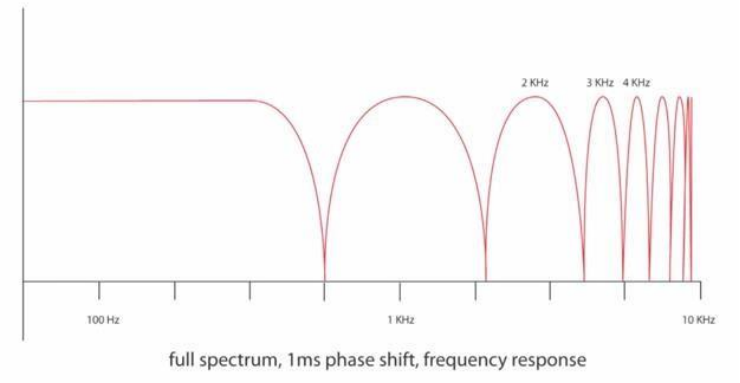


Figura 3.7: Comportamiento de la señal resultante en el espectro de frecuencias.

También se puede analizar el gráfico de la respuesta en fase después de usar un filtro pasa todo a medida que la frecuencia aumenta, se puede apreciar que en este ejemplo. Que el primer desfase significativo de 90 % se produce en 1khz y luego se desfasa completamente 180º en 10khz, ver figura 3.8.

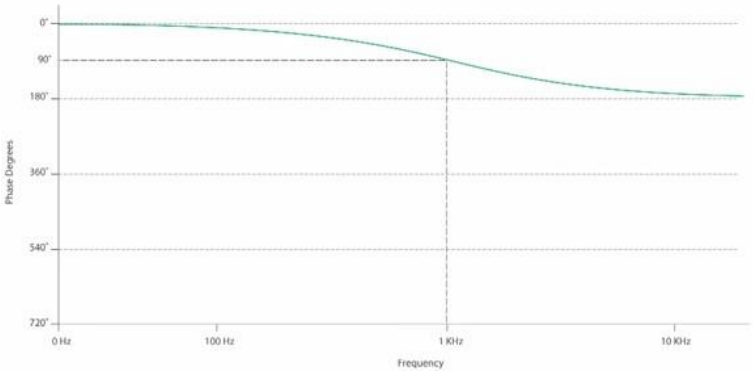


Figura 3.8: Respuesta en fase de la señal resultante

Cuando se utilizan varios filtros pasa todos la curva del desfase es más pronunciada. A continuación se puede ver la respuesta en desfase con diferente número de filtros pasa todo usado para cambiar la respuesta en desfase, ver figura 3.9.

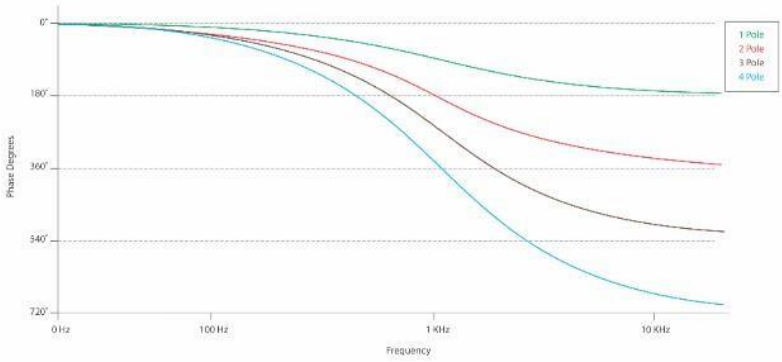


Figura 3.9: Respuesta en fase de la señal resultante con varios filtros pasa todo.

3.3 Objetivo del procesamiento

Como se mencionó, se busca obtener una señal que posea frecuencias con la misma ganancia que la entrada pero cambia la relación de fase con respecto a la señal de entrada. *El cambio de fase es una función de la frecuencia*, a mayor frecuencia se podrán observar diferentes cambios en fase.

Se produce una serie de picos y mínimos en el espectro de la frecuencia producto de la suma algebraica de los valores de las dos señales activas. Estos mínimos en el espectro de frecuencia que son el resultado de la suma de los valores que están desfasados completamente entre la señal de entrada y la señal de salida. Se denominan notches (muescas) y éstas crean el sonido característico del efecto phaser, es decir un sonido de barrido que apreciable en las frecuencias que el oído humano puede percibir. Como se mencionó antes el número de filtros pasa bajos determina las características del sonido. Un phaser con n etapas generalmente tiene n/2 notches en el espectro, así por ejemplo, un phaser con 4 etapas tendrá 2 notches.

CAPÍTULO 4: DESARROLLO DEL SISTEMA DE LA APLICACIÓN DE PROCESAMIENTO DE AUDIO

4.1 Módulos externos de procesamiento de audio

4.1.1 HiFiBerry DAC+

Como módulo de audio se usará el Hifi Berry DAC+ que sirve conversor A/D y D/A y principalmente provee la interface de audio de entradas y salidas análogas. Es una tarjeta de audio para maximizar la calidad del procesamiento de la señal. Acorde a la página oficial de Hifi Berry Dac+, la tarjeta contiene las siguientes características:

- Dedicated 192kHz/24bit high-quality Burr-Brown DAC for best sound quality
- Connects directly to the Raspberry Pi, no additional cables needed
- Compatible with Raspberry
- Directly powered from the Raspberry Pi, no additional power supply
- Ultra-low-noise voltage regulator for optimal audio performance
- Flexible configuration options for output connectors
- Available in different configurations
- Easy to build: Comes as a pre-fabricated kit that includes all needed components. On Raspberry Pi Model A and B, you have to solder a small connector. Make sure that your board features the P5 connector, some very old board (Revision 1) don’t have it.
- Are you looking for a solution that does not need soldering? In this case, you can use our HiFiBerry DAC+ with the new Raspberry Pi model B+.

En la figura 4.1 se puede apreciar el módulo de audio Hifi Berry Dac+.

Figura 4.1: HIFI BERRY DAC+

4.1.1.1. Instalación del módulo HiFi Berry Dac+

Para la conexión entre las dos tarjetas simplemente se deben conectar los pines del Raspberry Pi con los del Hifi Berry Dac+ además de tornillos de soporte. Para más información visitar la siguiente página:

https://www.google.com.ec/webhp?sourceid=chrome-instant&rlz=1C1CHFX_enUS594US594&ion=1&espv=2&ie=UTF-8#q=hifi+berry+dac+installation+vimeo

4.1.1.2. Seteo del software del Hifi Berry Dac+

No basta con la instalación del módulo Hifi Berry. Se deben aplicar algunos comandos en el dispositivo para que reconozca el módulo, para que el módulo se active y para que el módulo sea el dispositivo de salida de audio predeterminado. Para conseguir esto se debe seguir el siguiente proceso después de haber instalado manualmente la tarjeta:

Se debe introducir los siguientes comandos en la terminal para actualizar (usando conexión a internet) el firmware del dispositivo (el cual consiste de la unión del Raspberry Pi y el módulo HiFi Berry Dac+):

```
$sudo rpi-update
$sync
$sudo reboot
```

La parte más importante es la configuración del archivo /etc/modules, al cual le debemos añadir las siguientes líneas de código:

```
$sudo  /etc/modules
```

La línea anterior se usa para acceder al archivo. Posteriormente se añade lo siguiente:

```
$snd_soc_bcm2708
$snd_soc_bcm2708_i2s
$bcm2708_dmaengine
$snd_soc_pcm5102a
$snd_soc_hifiberry_dac
```

Después de esto se debe reiniciar el dispositivo y ejecutar la siguiente instrucción usada anteriormente para ver los dispositivos de salida predeterminados:

```
pi@raspberrypi ~ $ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: sndrpihifiberry [snd_rpi_hifiberry_dac], device 0:
HifiBerry DAC HiFi pcm5102a-hifi-0 []
Subdevices: 1/1
Subdevice #0: subdevice #0
```

Si se obtienen estas líneas de información se asume que el módulo HiFi Berry está activado y es el dispositivo de salida de audio predeterminado.

4.1.2 Acondicionamiento de la señal

En la mayoría de los sistemas de procesamiento de señales se requiere de una entrada limpia e ideal para procesar. Muchos son los factores que pueden alterar la calidad de la señal de entrada antes que pueda ser manipulada en el dispositivo de procesamiento. Debido a esto se usaron en el sistema diversos dispositivos que permitirían la obtención de una entrada lo más pura posible para realizar el procesamiento posterior. A continuación se detallarán estos dispositivos y su uso.

4.1.2.1. Tarjeta de sonido USB

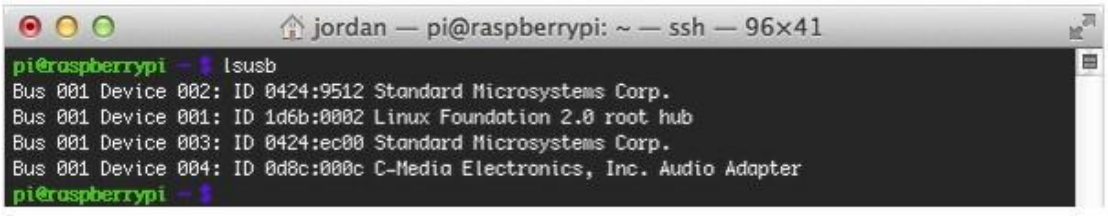
El Raspberry Pi, como se mencionó anteriormente, no posee un puerto de entrada de audio instalado en la tarjeta. Por esta razón se hizo uso de una tarjeta de sonido USB como se muestra en la figura 4.2:

Figura 4.2: Tarjeta de sonido USB

Esta tarjeta de sonido permite añadir un puerto de entrada mono (1 canal) y un puerto estéreo (2 canales) de salida. Posteriormente se debe setear el dispositivo de procesamiento (Raspberry Pi) para que configure sus valores predeterminados de los dispositivos en entrada y salida para que esta tarjeta USB este activa para recibir señales.

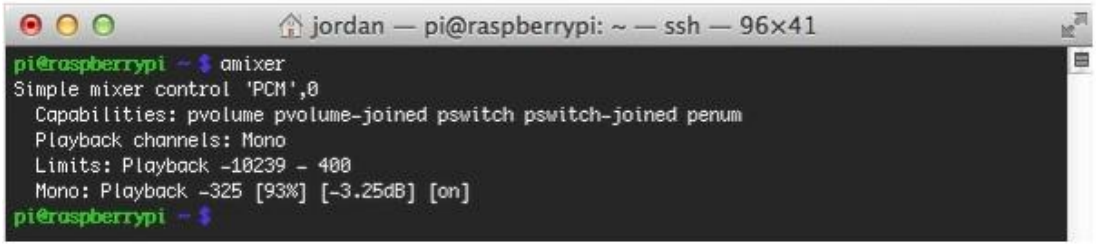
Se sigue la siguiente serie de comandos en la terminal para activar la tarjeta de sonido USB.

```
$ lsusb
```



Este comando permite mostrar los puertos USB disponibles en el dispositivo además de mostrar los dispositivos que están conectados en estos puertos. En cambio, con el siguiente comando se muestran los dispositivos de audio disponible:

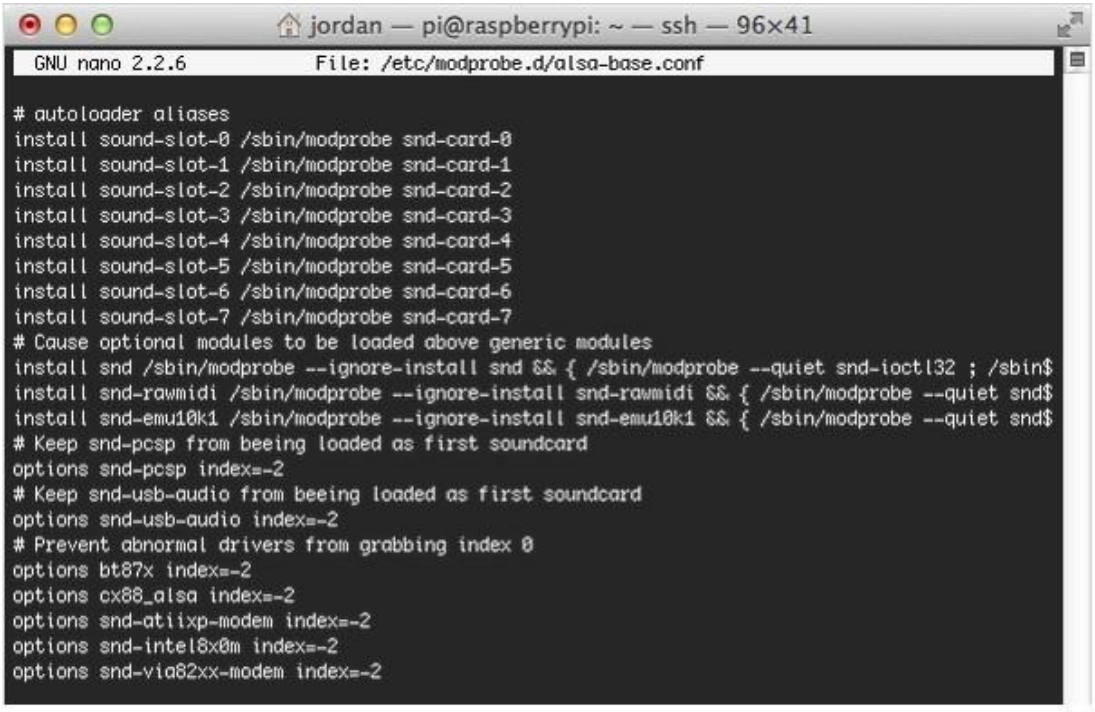
```
$ amixer
```



Al inicio el dispositivo de audio predeterminado es el instalado en el dispositivo, necesitamos que este comando muestre la tarjeta de sonido USB que se usará. Para lograr

esto debemos modificar un archivo interno del dispositivo para que acepte el índice del dispositivo predeterminado al iniciar el dispositivo. Para esto el archivo **alsa-base.conf** ubicado en la carpeta **/etc/modprobe.d/alsa-base.conf** debe ser modificado con la siguiente instrucción:

```
$ sudo nano /etc/modprobe.d/alsa-base.conf
```



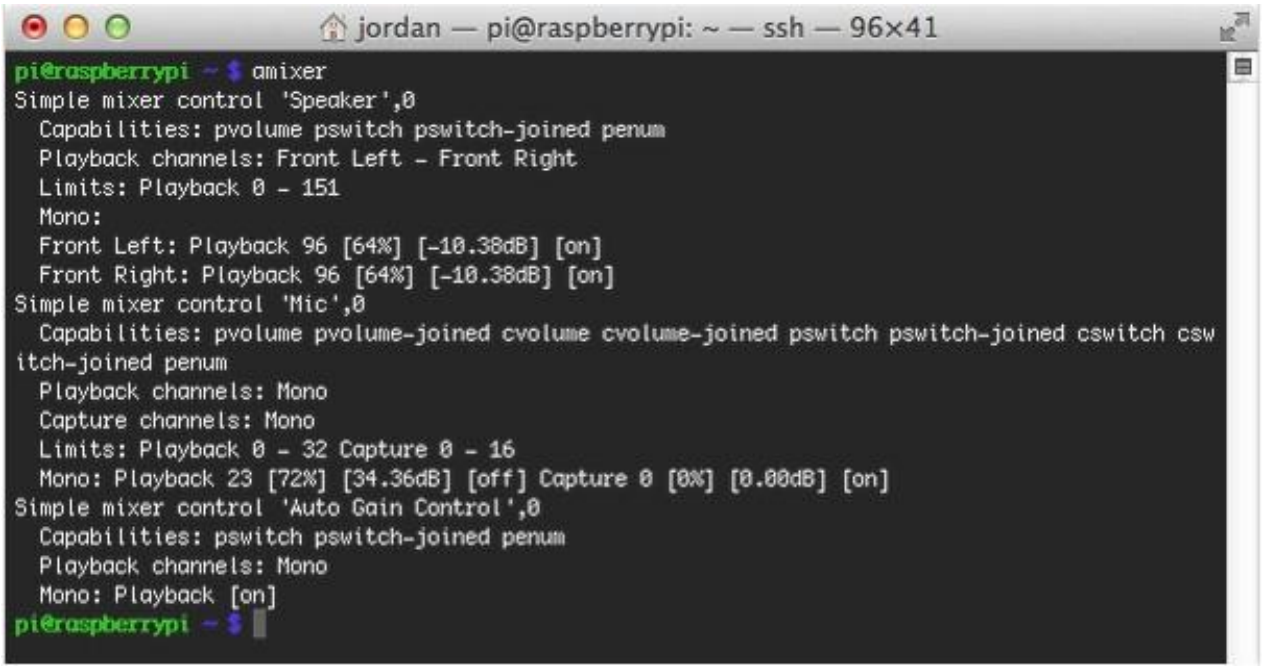
Lo que se busca es cambiar una de las líneas de código para que el dispositivo no la lea. Para esto se añade un carácter de comentario a la siguiente línea.

```
options snd-usb-audio index=-2
```

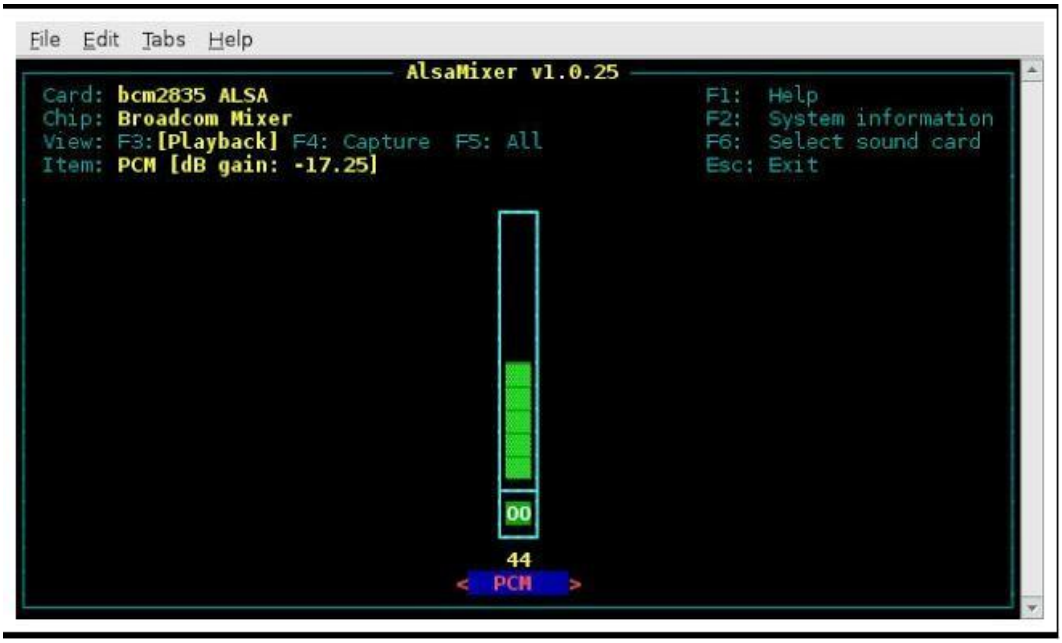
```
# options snd-usb-audio index=-2
```

Una vez cambiado esta línea se guarda el archivo y se sale de la terminal. Se reinicia y

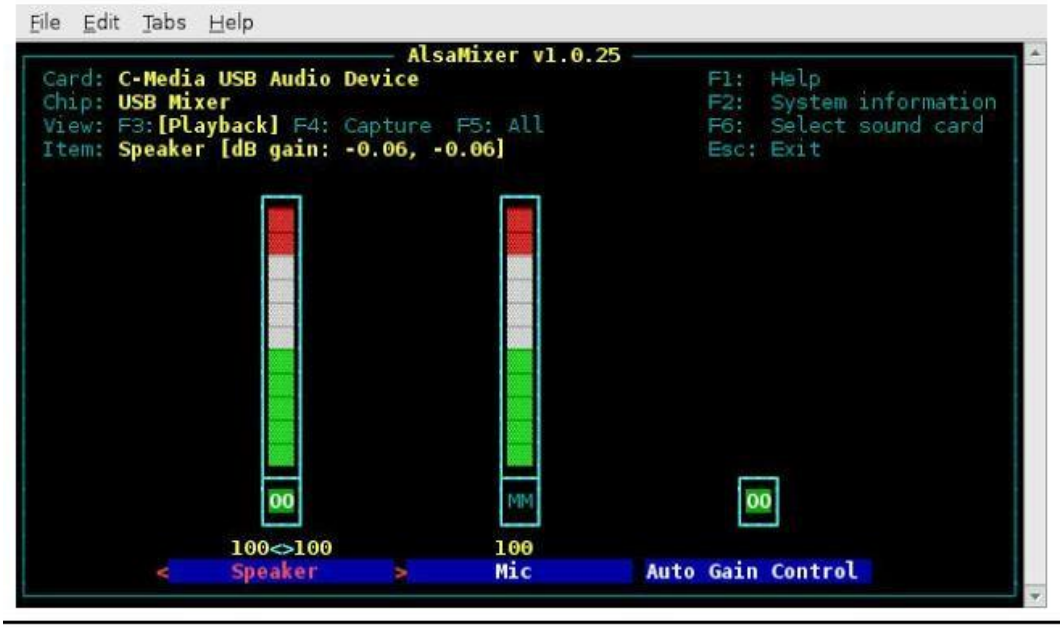
nuevamente se ejecutan los comandos iniciales para ver si los cambios se realizaron de la manera correcta. Si todo se siguió correctamente al mostrar los dispositivos de audio predeterminados se obtendrá la siguiente información en la terminal:



Posteriormente, para modificar los niveles del micrófono se utiliza un programa preinstalado en el dispositivo llamado *alsa mixer*. Con la siguiente instrucción se accede al controlador gráfico de los dispositivos de audio *alsa mixer*:



Se selecciona el dispositivo con las teclas específicas y se ajustan los niveles, se obtendrá una ventana parecida a la siguiente:



El último paso es configurar la computadora para que acepte la tarjeta de audio USB como el dispositivo de salida predeterminado para que siempre que se conecte este dispositivo, la

computadora automáticamente lo selección e interactúe con este. Se ejecutan los siguientes comandos:

```
$pacmd list-sinks
```

Este comando permite ver la lista de los dispositivos de entrada y salida de datos. Con la información relevante como el nombre del dispositivo se sigue las siguientes instrucciones:

```
$pacmd set-default-sink "SINKNAME"
```

```
$pacmd set-default-source "SOURCENAME"
```

En “SINKNAME” y “SOURCENAME” colocar el nombre que aparece en la lista anterior. Por último se debe modificar una línea de código del archivo /etc/pulse/default.pa. Con la siguiente instrucción se abre el editor de texto de este archivo:

```
$sudo nano /etc/pulse/default.pa
```

Una vez abierto el editor se debe cambiar la siguiente instrucción:

```
load-module module-stream-restore
```

se la debe cambiar a:

```
load-module module-stream-restore restore_device=false
```

Finalmente se reinicia la computadora y si los cambios se efectuaron correctamente el dispositivo debería funcionar al conectarse con el microprocesador(Raspberry Pi)

En caso de cualquier complicación consultar las siguiente páginas:

http://computers.tutsplus.com/articles/using-a-usb-audio-device-with-a-raspberry-pi--mac-55876 y http://asliceofraspberrypi.blogspot.com/2013/02/adding-audio-input-device.html

4.1.2.2. Amplificador Análogo Amplug Vox AC30

Este módulo es un producto usado en la industria por músicos para poder oír la señal amplificada por medio de audífonos o por medio de cualquier dispositivo speaker. Este módulo tiene un circuito interno que emula las funciones de los amplificadores de audio análogos regulares pero sin la necesidad que la potencia sea audible. El dispositivo de amplificación Vox Ac30 se puede ver en la figura 4.3:

Figura 4.3: Amplificador de señal

La razón por la que se usa este módulo de ampliación es el hecho que la señal electromagnética que es producida por la guitarra eléctrica es muy débil y necesita ser amplificada previamente antes de entrar al dispositivo de procesamiento de audio (Raspberry Pi). Esto se comprobó de manera experimental.

Se usó el programa *Audacity* para grabar la entrada de la señal directamente desde la guitarra y la señal amplificada con este módulo Vox AC30. Luego de la grabación se usó

un analizador de espectro de frecuencia para visualizar los niveles que posee cada señal. Se obtuvieron las figuras 4.4 y 4.5:

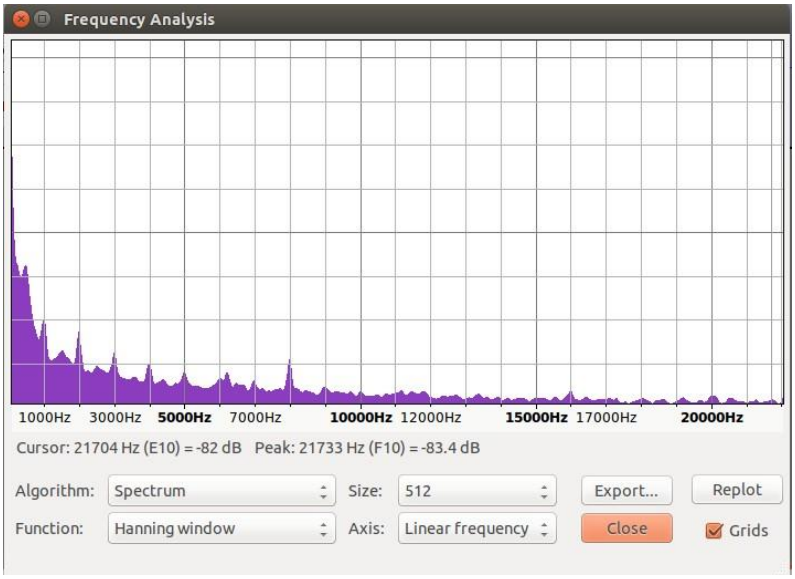


Figura 4.4: Señal sin amplificación

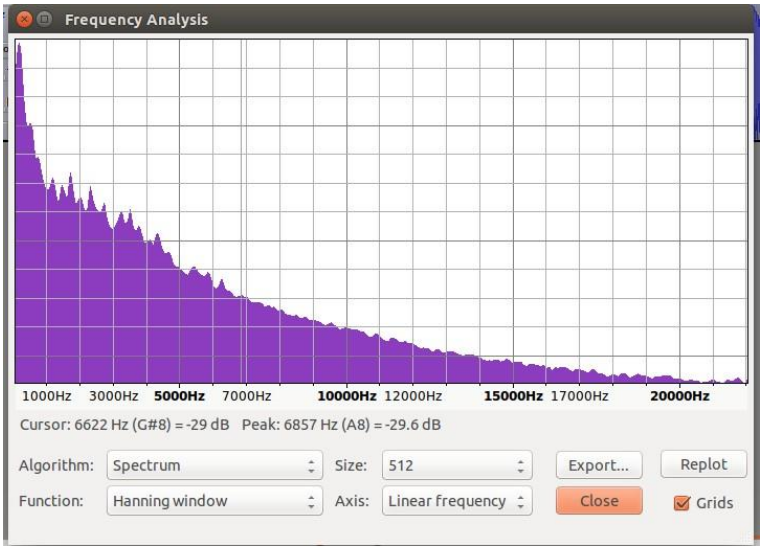


Figura 4.5: Señal con amplificación

Se puede ver en la figura 4.4 que la señal sin amplificación posee un nivel de decibeles promedio de -63dB(nivel que la computadora que no puede leer), mientras que la señal amplificada en la figura 4.5 posee un promedio de 0dB. Por esta razón se hizo necesario añadir este módulo de amplificación al sistema de la aplicación.

4.1.2.3. Adaptador Stereo Mono

Uno de problemas más importantes que se detectaron al inicio de las pruebas del procesamiento fue la presencia de ruido. Hay muchas causas por las que se produce ruido en la conexión de dispositivos. En este caso se piensa tolerar el ruido blanco de la amplificación pero al realizarse las pruebas de procesamiento se detectaron un ruido adicional y de alta potencia. Se llegó a la conclusión que este ruido es el resultado de una conexión al aire de la tierra entre entradas y salidas mono(1 canal) y stereo(2 canales). Se analizó el sistema de entradas y salidas y se llegó a la siguiente conexión ideal para no tener tierras sueltas que puedan producir ruido. Esta se ilustra en la figura 4.6:

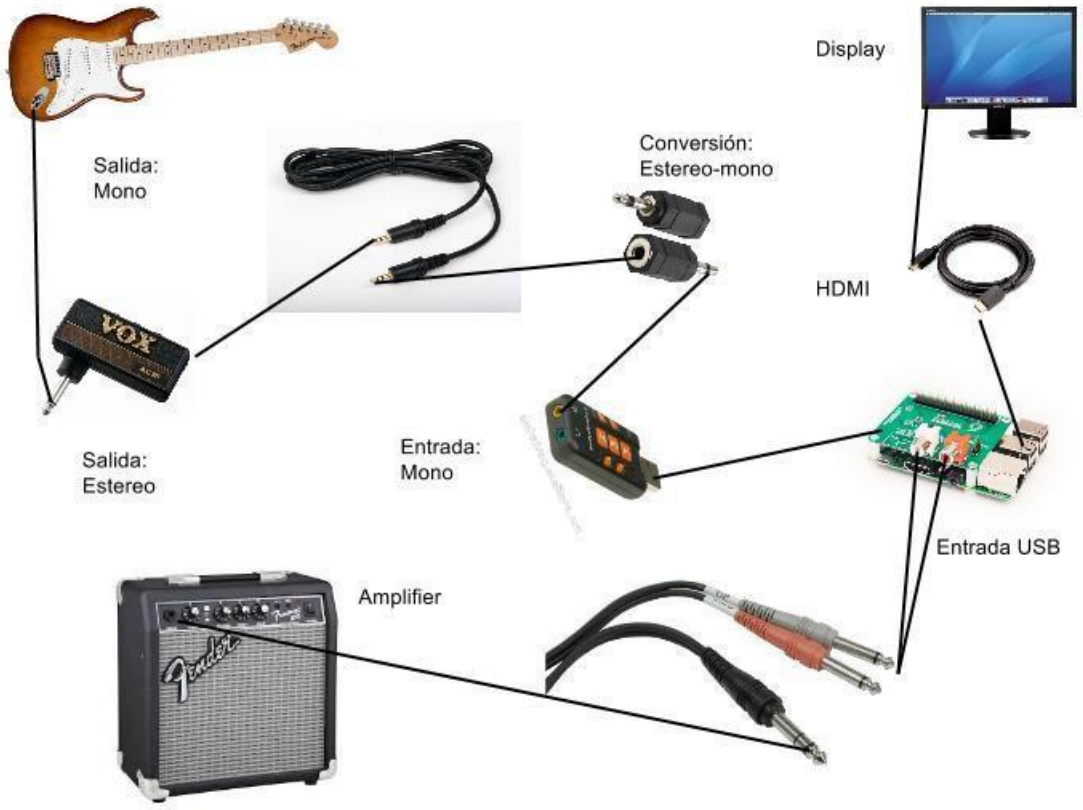


Figura 4.6: Sistema de acondicionamiento de señal completo

El adaptador estereo a mono que se conecta con la tarjeta de audio USB resolvió el problema del ruido causado por la tierra al aire del cable estereo previo a este. Se notó que la potencia de la señal disminuye en cierta medida pero al tener los controles de la salida en el amplificador al final del sistema es irrelevante esta disminución de la potencia. Lo importante es obtener una señal de entrada lo más limpia posible.

4.2 Modelo del proyecto final alcanzado

Debido a que el análisis de las señales con los filtros del procesamiento y la programación del procesamiento y su unión con los controles de la interface llevó más tiempo de lo esperado se optó por suprimir la parte de la comunicación del dispositivo con una interfaz en una página web diseñada. Al final el esquema del sistema se ve a continuación en la figura 4.8.

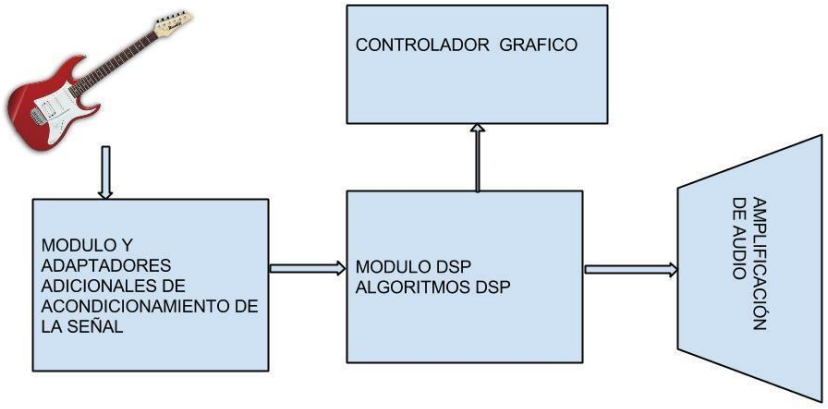


Figura 4.8: Modelo del Proyecto Final

Capitulo 5: Análisis de los filtros Pasa Todo

immediate

August 20, 2015

1 Análisis de la función de transferencia del filtro pasa todo

En procesamiento de audio, un tipo de sistema de gran importancia es el denominado pasa todo y el es responsable de dar el efecto que se busca en el proyecto, se denota como

$$H_{all}(z) = \frac{z^{-1} - a}{1 - az^{-1}} \tag{1}$$

En su forma más simple, tiene una función de transferencia que posee una característica de amplitud constante. Esto quiere decir que la señal filtrada tendrá una ganancia igual a la salida.

$$H_{all}(z) = \left|\frac{z^{-1} - a}{1 - az^{-1}}\right| = 1 \tag{2}$$

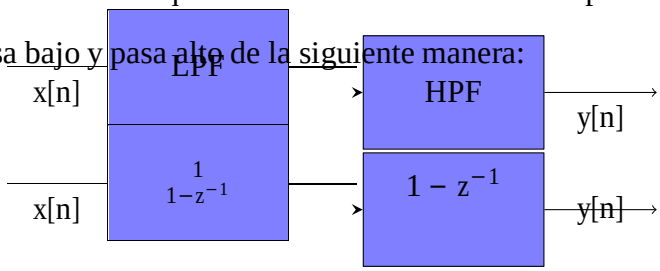
Un sistema paso todo compensa la influencia del polo en a sobre la característica de amplitud con un cero en 1/a*. Podemos construir sistemas paso todo del orden que deseemos sin más que combinar sistemas de orden 1 como en la ecuación 1.

Los filtros genéricos pasa bajos y pasa altos se relacionan en el dominio del tiempo y el dominio de la frecuencia de la siguiente manera:

$$\frac{dx(t)}{dt} \xrightarrow{F} j\omega x(\omega)(HighPassFilter)$$

$$\int x(t)dt \xrightarrow{F} \frac{x(\omega)}{j\omega}(LowPassFilter)$$

Si se analiza la función de transferencia en detalle se puede ver que es una multiplicación de un filtro pasa bajos y un filtro pasa altos. En un sistema de bloques representa la acción de dos bloques seguidos y en el dominio z es una multiplicación de las funciones que representan estos bloques. Con esto se analizan las expresiones genéricas comunes de los filtros pasa bajo y pasa alto de la siguiente manera:



$$(1 - z^{-1})X(z)(\text{PolosAltos})$$
$$(\frac{1}{1 - z^{-1}})X(z)(\text{PolosBa jos})$$

Si queremos encontrar los polos y ceros de la función de transferencia en el dominio z para poder predecir el comportamiento de la ecuación a diferencias que se aplicaran en la programación posterior se sigue los siguientes cálculos.

$$H_{all}(z) = (\frac{z^{-1} - a^* \quad z}{1 - az^{-1}})\frac{}{z} = \frac{1 - a^* z}{z - a} \tag{3}$$

Los polos se encuentran igualando a cero el denominador de la función de transferencia y los ceros con el numerador. Con esto en cuenta se determina que:

Para encontrar el cero: $1 - a^* z = 0 \rightarrow z = \frac{1}{a^*}$
Para encontrar el polo: $z - a = 0 \rightarrow z = a$

Y luego encontramos que:

$$a = e^{jw} \tag{4}$$

$$a^* = e^{-jw} \tag{5}$$

Para encontrar la respuesta impulsiva del filtro se procede a aplicar la transformada inversa

de z:

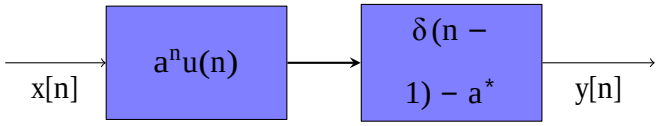
$$X(z) = \mathbf{F}\, x(n) \rightarrow x(n) = \mathbf{F}^{-}1X(z)$$

De aquí obtenemos las expresiones de la respuesta impulsiva del filtro pasa alto y pasa bajo de la manera siguiente:

$$H_{HP}(z) = \frac{1}{1 - az^{-1}} \rightarrow h_{HP}(n) = a^{n}u(n) \tag{6}$$

$$H_{LP}(z) = z^{-1} - a^{*} \rightarrow h_{LP}(n) = \delta(n - 1) - a^{*} \tag{7}$$

Se puede apreciar el procesado de la señal a través del sistema de bloques en la siguiente figura:



Ahora que se puede ver como se comportan las muestras en el tiempo discreto es momento de calcular la ecuación a diferencias de la función de transferencia ya que esta sera la que se implementara en la programación respectiva primero en MatLab y luego en la plataforma del Raspberry Pi. El proceso para encontrar la ecuación a diferencias se describe a continuación:

$$H_{all}(z) = \frac{z^{-1} - a}{1 - az^{-1}} \rightarrow Y(z) - az^{-1}Y(z) = z^{-1}X(z) - a^{*}X(z)$$

Ahora se aplica la transformada inversa de z en esta expresion:

$$y(n) - ay(n - 1) = x(n - 1) - a^{*}x(n)$$

Finalmente se ordena la ecuación y se coloca las condiciones iniciales:

$$y(n) = ay(n - 1) - a^{*}x(n) + x(n - 1) \tag{8}$$

$$y(-1) = 0 \tag{9}$$

$$y(0) = 0 \tag{10}$$

2 Definición de las ecuaciones de diferencias de cada orden

Se calcularon las funciones de transferencia de filtro de orden 1 hasta el orden 5 concatenando el filtro pasa todos de orden 1. Se puede ver que conforme se aumenta el orden del filtro se produce un patrón. Se comprobó que al concatenar el numerador o el denominador y al ser una resta de términos, al elevarlos a alguna potencia, se produce una respuesta binomial que sigue un patrón con coeficientes determinados de acuerdo a la distribución binomial. Posterior a ello, se realizó los cálculos de la ecuación a diferencias así como se explicó en la sección anterior para calcular la ecuación a diferencias del filtro de orden 1. El criterio de estabilidad es el análisis de la región de convergencia ROC. Asumimos que la transformada Z es unilateral porque se trabaja con un vector de entrada finito que empieza en 0. Con esto en mente, se debe buscar que los polos de la función de transferencia se ubiquen dentro del círculo unitario. Por esta razón el argumento a se multiplica con una constante α que tiene un rango de valores entre (0,1), el cual es un rango abierto(no incluye el 0 ni el 1). El valor 1 de α puede causar inestabilidad porque el valor absoluto del argumento a es 1 y al estar en el límite del círculo unitario es propenso a volverse inestable. En los siguientes ejemplos se uso un valor de α cercado a 1 PERO con tres decimales de precisión de manera que no se topa el valor absoluto exacto de 1, pero se acerca bastante.

Para el orden 1, la función de transferencia:

$$H_1(z) = \frac{z^{-1} - a^*}{-az^{-1} + 1} \tag{11}$$

La función de transferencia expresada en sus componentes reales e imaginarias(Para trabajar con valores reales podemos modificar (11), como se ilustra en la nota al pie de página): [1]

$$H_1(z) = \frac{z^{-1} - (0.707 + j0707)}{-(0.707 - j0.707)z^{-1} + 1} \tag{12}$$

La ecuación a diferencias del orden 1:

$$y(n) = ay(n-1) - a^* x(n) + x(n-1) \tag{13}$$

Para el orden 2, la funcion de transferencia:

$$H_2(z) = \left(\frac{z^{-1} - a^*}{-az^{-1} + 1}\right)^2 = \frac{z^{-2} - 2z^{-1}a^* + (a^*)^2}{a^2 z^{-2} - 2az^{-1} + 1} \tag{14}$$

La función de transferencia expresada en sus componentes reales e imaginarias:

$$H_2(z) = \frac{z^{-2} + (-1.414 + j1.414)z^{-1} + (0 - j)}{jz^{-2} + (-1\ 414 - j1\ 414)z^{-1} + 1} \tag{15}$$

. .

La ecuación a diferencias del orden 2:

$$y(n) = x(n-2) - 2(a^*)x(n-1) + (a^*)^2 x(n) - a^2 y(n-2) + 2ay(n-1) \tag{16}$$

Para el orden 3, la funcion de transferencia:

$$H_3(z) = \left(\frac{z^{-1} - a^*}{-az^{-1} + 1}\right)^3 = \frac{z^{-3} - 3z^{-2}a^* + 3z^{-1}(a^*)^2 - (a^*)^3}{-a^3 z^{-3} + 3a^2 z^{-2} - 3az^{-1} + 1} \tag{17}$$

---

[1]

$$H_{all}(z) = \frac{z^{-1} - \mathrm{Re}\{a\}}{1 - \mathrm{Re}\{a\}z^{-1}}$$

La función de transferencia expresada en sus componentes reales e imaginarias:

$$H_3(z) = \frac{z^{-3} + (-2.1213 + j2.1213)z^{-2} + (0 - j3)z^{-1} + (0.707 + j0.707)}{(0.707 - j0.707)z^{-3} + (0 + j3)z^{-2} + (-2.1213 - j2.1213)z^{-1} + 1} \qquad (18)$$

La ecuación a diferencias del orden 3:

$$y(n) = x(n-3) - 3(a^*)x(n-2) + 3(a^*)^2x(n-1) - (a^*)^3x(n) + a^3y(n-3) - 3a^2y(n-2) + 3ay(n-1) \qquad (19)$$

Para el orden 4, la función de transferencia:

$$H_4(z) = \left(\frac{z^{-1} - a^*}{-az^{-1} + 1}\right)^4 = \frac{z^{-4} - 4z^{-3}a^* + 6z^{-2}(a^*)^2 - 4z^{-1}(a^*)^3 + (a^*)^4}{a^4z^{-4} - 4a^3z^{-3} + 6a^2z^{-2} - 4az^{-1} + 1} \qquad (20)$$

La función de transferencia expresada en sus componentes reales e imaginarias:

$$H_4(z) = \frac{z^{-4} + (-2.8284 + j2.8284)z^{-3} + (0 - j6)z^{-2} + (2.8284 + j2.8284)z^{-1} + (-1)}{(-1)z^{-4} + (2.8284 - j2.8284)z^{-3} + (0 + j6)z^{-2} + (-2.8284 - j2.8284)z^{-1} + 1} \qquad (21)$$

La ecuación a diferencias del orden 4:

$$\begin{aligned} y(n) = {} & x(n-4) - 4(a^*)x(n-3) + 6(a^*)^2x(n-2) - 4(a^*)^3x(n-1) + (a^*)^4x(n) \\ & -a^4y(n-4) + 4a^3y(n-3) - 6a^2y(n-2) + 4ay(n-1) \end{aligned} \qquad (22)$$

Para el orden 5, la función de transferencia:

$$H_5(z) = \left(\frac{z^{-1} - a^*}{-az^{-1} + 1}\right)^5 = \frac{z^{-5} - 5z^{-4}a^* + 10z^{-3}(a^*)^2 - 10z^{-2}(a^*)^3 + 5z^{-1}(a^*)^4 - (a^*)^5}{-a^5z^{-5} + 5a^4z^{-4} - 10a^3z^{-3} + 10a^2z^{-2} - 5az^{-1} + 1} \qquad (23)$$

La función de transferencia expresada en sus componentes reales e imaginarias:

$$H_5(z) = \frac{z^{-5} + (-3.5355 + j3.5355)z^{-4} + (-j10)z^{-3} + (7.0711 + j7.0711)z^{-2} + (-5)z^{-1} + (0.707 - j0.707)}{(0.707 + j0.707)z^{-5} + (-5)z^{-4} + (7.0711 - j7.0711)z^{-3} + (j10)z^{-2} + (-3.5355 - j3.5355)z^{-1} + 1} \qquad (24)$$

La ecuación a diferencias del orden 5:

$$y(n) = x(n-5) - 5(a^*)x(n-4) + 10(a^*)^2x(n-3) - 10(a^*)^3x(n-2) + 5(a^*)^4x(n-1) - (a^*)^5x(n) + a^5y(n-5) - 5a^4y(n-4) + 10a^3y(n-3) - 10a^2y(n-2) + 5ay(n-1) \quad (25)$$

3 Comprobación de la respuesta simulada y la predicha por los cálculos

Se usó dos métodos en Matlab para comprobar que las ecuaciones a diferencias filtran la señal de la manera predicha por los cálculos. Existe una función FILTER en Matlab que permite filtrar la señal directamente ingresando los vectores de los polinomios del numerador y denominador de la función de transferencia del filtro. Para verificar el comportamiento del filtrado se usó esta función a la vez que las ecuaciones a diferencias y se tomó los 10 puntos iniciales de ambos métodos y los 10 puntos finales de los mismos con un filtro de orden determinado. Se restaron los primeros 10 puntos de la función de Matlab con los de las ecuaciones a diferencias y dio un resultado nulo, es decir los primeros valores fueron iguales.

SIN EMBARGO, ya que los coeficientes que se proponen implementar en la plataforma no son completamente exactos, y al ser estos filtros recursivos en el tiempo se va formando una función de error en comparación con el cálculo exacto de Matlab. Esto se comprobó porque los 10 últimos puntos difieren en mucho de los primeros puntos.

Para ver el impacto que tiene esta función de error se procedió a realizar pruebas de restas entre 10,100,1000, y 10000 puntos de ambos métodos y luego se sumo estos valores para ver la suma de estas variaciones. Se obtuvo la siguiente tabla:

N* of Points	x	y(complex)
10 points	$-2,7523x10^{-}13$	$j2,3384x10^{-}14$
100 points	$-2.33x10^{-}9$	$j1.7792x10^{-}8$
1000 points	0.0011	$j-0.0001$
10000 points	-36.2044	$j-60.9417$

Como se puede ver, hasta los 1000 puntos no se encuentra un error relevante en el procesamiento, sin embargo, luego de los 10000 puntos la suma de las variaciones de las diferencias empieza a ser notorio. Esto podría ser causa de problemas en el procesamiento porque se asume que se trabajara con mucho mas que 10000 puntos, es mas, al ser el procesamiento en tiempo real, las muestras pueden ser tan infinitas mientras dure el audio de entrada.

CAPÍTULO 6: ANÁLISIS DE SEÑALES EN MATLAB

6.1 Objetivos del análisis

Para analizar el procesamiento teórico se realiza una simulación en matlab. Se pretende simular el sonido de los tonos de la guitarra, luego se pretender simular el filtrado teórico durante el procesamiento y finalmente ver la respuesta de la señal procesada. Antes de implementar en la plataforma se procede a definir ciertas metas propuestas en la simulación en matlab, las cuales se detallan a continuación:

-Se pretende simular un tono de la guitarra, es decir emular el verdadero sonido que una guitarra de madera o eléctrica emite al ser ejecutada por el músico.
-A partir del filtro teórico descrito en secciones anteriores se pretende observar y analizar la respuesta en frecuencia teórica de este filtro de un orden determinado.
-Se busca ver el desfase que produce un filtro de determinado orden en la señal de salida con respecto a la señal de entrada.

6.2 Simulación del sonido de la guitarra

El primer paso fue usar matlab para simular el sonido de un tono específico de la guitarra. Para este fin se usó el algoritmo Karplus Strong, el cual es un método de síntesis de modelos físicos que lupea una forma de onda a través de una línea de retardo filtrada para simular el sonido de

una cuerda o algún tipo de percusión. Alexander Strong inventó el algoritmo pero Kevin Karplus realizó el primer análisis que funcionó de manera correcta.

Generalmente funciona de la siguiente manera. Se definen variables de muestreo para el vector de frecuencia, se define la frecuencia del tono, se simula ruido blanco como señal de entrada. Ver la figura 6.1.

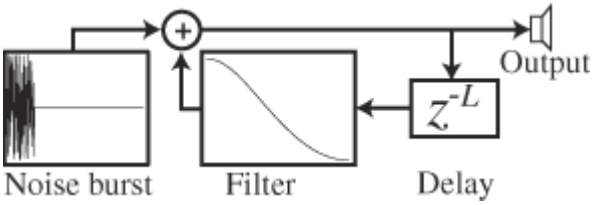


Figura 6.1: Algoritmo Karpus Strong

Es importante recalcar que cuando una guitarra es golpeada, produce una onda de sonido con picos en el dominio de la frecuencia que están igualmente espaciados. Estos se llaman armónicos y dan a cada nota el sonido completo. Se puede generar ondas de sonido con estos armónicos con objetos de filtros en tiempo discreto. Se determinan retrasos en el feedback basados en la primera frecuencia armónica. Se genera un filtro IIR cuyos polos aproximan los armónicos de la frecuencia de la cuerda escogida y finalmente se filtra la señal. Con funciones pre fabricadas en matlab(ver ANEXOS) se puede realizar este procedimiento y se obtiene una respuesta en frecuencia en la siguiente figura 6.2. El archivo *guitarsimulation.m* permite simular una cuerda de guitarra al ser golpeada, especificando la frecuencia fundamental (tono o nota) y ésta función gratificará la fundamental y las frecuencias armónicas que son componentes de la fundamental(En la figura 6.2 solo se muestra los armónicos, la fundamental debería tener una mayor ganancia).

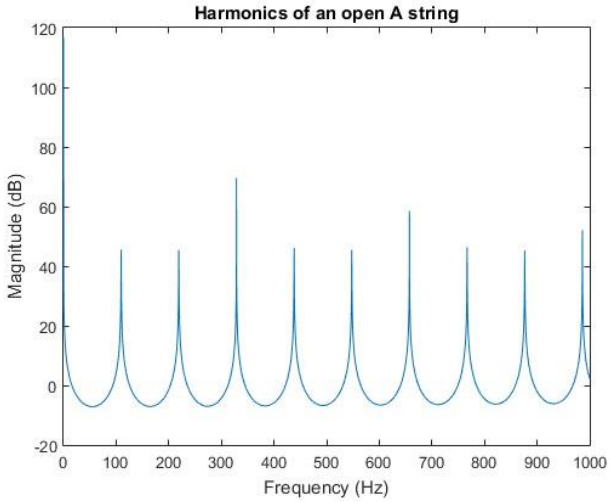


Figura 6.2: Armónicos de la cuerda A(110 Hz)

6.3 Características de la señal de entrada: Señal de la guitarra eléctrica

Con el objetivo del análisis de la señal principal de entrada, la cual es una señal de audio análoga, caracterizada principalmente por contener un contenido espectral donde las frecuencias que tienen mayor magnitud en escala normal y en escala en decibeles, se grabó un archivo de audio de 10 segundos de la señal de la guitarra eléctrica. La señal grabada contiene un par de acorde. Un acorde es el golpe de varios tonos diferentes, que al ser tocados al mismo tiempo producen una sensación agradable en el oído cuando obedecen a reglas musicales. Estas reglas musicales fueron desarrolladas por la tradición europea cientos de años atrás. Conforme ha pasado el tiempo estas reglas han sido usadas para crear música en la mayor parte del mundo. Básicamente estas reglas se basan en 7 tonos principales y medios tonos entre ellos. Este conjunto de tonos se denomina la escala diatónica o escala pitagórica.

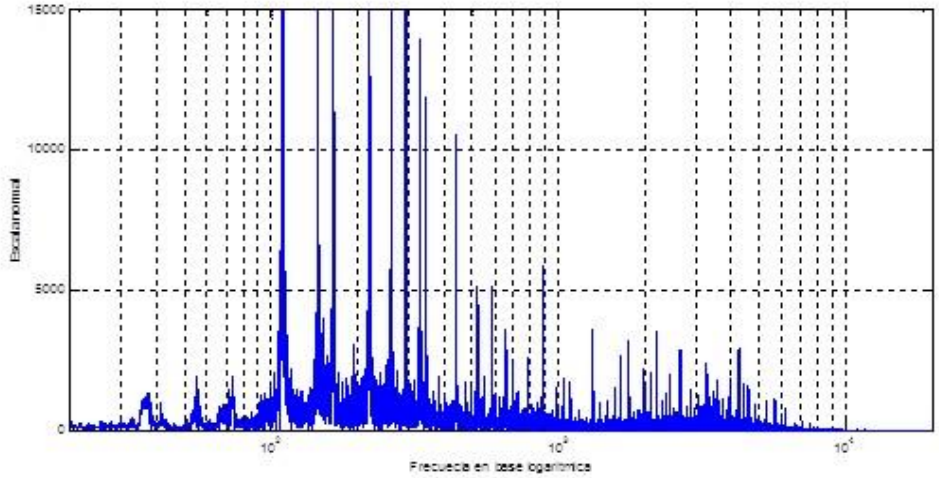
Se pretende comprobar que las frecuencias que dibuja el software MATLAB coinciden con la tabla de clasificación de frecuencias que se mencionó en la sección anterior (ver capítulo 2). Con esto se puede tener una idea más clara de saber cómo afecta los filtros DSP que se pretenden implementar en el presente proyecto a la señal pura.

La función *notas.m*(ver ANEXOS) permite leer un archivo previamente grabado de una guitarra eléctrica y muestra la gráfica de las frecuencias fundamentales tocadas en el archivo, sus armónicos y frecuencias extras que se adhieren por factores físicos del ambiente. El archivo de audio contiene una grabación de 10 segundos de dos acordes (un acorde es el golpe de varias notas al mismo tiempo que obedecen a la escala pitagórica para crear armonía musical). Los dos acordes que se tocaron en los 10 segundos de grabación es A-7 y D-7, las cuales tienen las siguientes frecuencias fundamentales con mayor potencia de acuerdo al Modelo Americano Estándar del Pitch(ver capítulo 2) y que se pueden ver en la siguiente figura son:

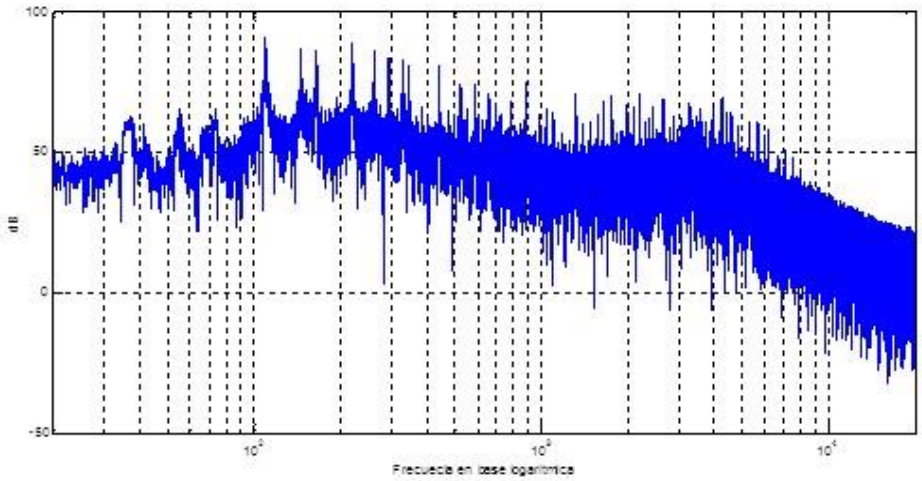
Para A-7: A2(110Hz) E3(165Hz) A3(220Hz) E4(320Hz) A4(440Hz)

Para D-7: D3(147Hz) A3(220Hz) D4(294Hz) F(349Hz) A4(440Hz)

Estas 9 notas (algunas se repiten), son las que tienen los picos más altos y los que se predice que son las que el instrumento interpreta. El resto de frecuencias que tienen un menor valor en potencia se las toma como frecuencias armónicas, ver figura 6.3.



a.



b.

Figura 6.3: Representación de las frecuencias de la grabación en a) escala logarítmica, en b) escala normal y decibeles para el eje y.

6.4 Forma onda de las señales procesadas de entrada y salida usando el filtro pasa todos

Con el objetivo de analizar las señales, se pretende ver la respuesta del filtro pasa todos, la forma de onda de la entrada y salida y su respuesta en frecuencia.

***Respuesta del filtro**

Como lo predicho en las funciones de transferencias calculadas en la sección anterior, se construyó un programa en matlab que permita calcular los polinomios del numerador y el denominador de determinados órdenes de filtros, además de la ubicación de los polos en el plano z y la respuesta en frecuencia del filtro. Para empezar se comprobó que efectivamente los polinomios predichos por los cálculos hechos en el capítulo 5 son los mismos realizados en cómputo con el programa Matlab entonces tenemos que:

*Matlab contienen funciones de análisis de filtros que aceptan los vectores de los polos y ceros para calcular la respuesta del filtro, normalmente se acepta un vector b que representa el numerador de la función de transferencia y sus respectivos ceros y un vector a que representa el denominador de la función de transferencia y sus respectivos polos. Para el presente análisis tal como en el análisis de la sección previas se toma un valor específico para el argumento a, que en este caso es pi/4, y lo que varía son el número de filtros concatenados usados.

```
Para el orden 1:
De acuerdo a la ecuación (12), los vectores de polos y ceros son:
b=[1.0000     -0.7071 - 0.7071i]
a=[ -0.7071 + 0.7071i   1.0000]

Para el orden 2:
De acuerdo a la ecuación (15), los vectores de polos y ceros son:
b=[1.0000  -1.4142-1.4142i   0.0000+1.0000i]
a=[0.0000-1.0000i  -1.4142+1.4142i   1.0000]

Para el orden 3:
De acuerdo a la ecuación (18), los vectores de polos y ceros son:
b=[1.0000           -2.1213 - 2.1213i   0.0000 + 3.0000i     0.7071 - 0.7071i]
a=[0.7071+0.7071i   0.0000-3.0000i  -2.1213+2.1213i     1]
```

```
Para el orden 4:
De acuerdo a la ecuación (21), los vectores de polos y ceros son:
b=[1.0000   -2.8284-2.8284i  0.0000+6.0000i  2.8284-2.8284i  -1]
a=[-1.0000-0.0000i   2.8284+2.8284i   0.0000-6.0000i   -2.8284+2.8284i  1.0000]

Para el orden 5:
De acuerdo a la ecuación (24), los vectores de polos y ceros son:
b=[1.0000  -3.5355-3.5355i   0.0000+10.0000i  7.0711-7.0711i  -5.0000+0.0000i
0.7071+0.7071i]
a=[0.7071-0.7071i  -5.0000-0.0000i  7.0711+7.0711i   0.0000-10.0000i  -3.5355+3.5355i   1]
```

Una vez comprobados los mismos valores predichos por los cálculos se procede a ver el comportamiento del filtro. Se analizará la diferencia del filtro de orden 1 y del filtro de orden 5 para ver la mayor diferencia. El programa *filterresp.m(ver ANEXOS)* permite apreciar los polos y ceros de la función de transferencia de un orden dado del filtro pasa todos. El programa nos arroja los siguientes resultados con un vector de frecuencia de 0 a 2*pi en las figuras 6.4 y 6.5:

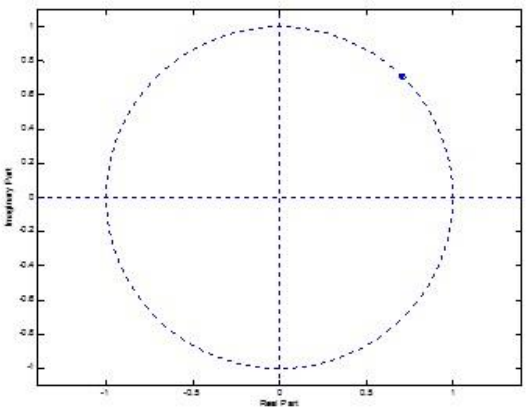


Figura 6.4: Ubicación de los polos y ceros en el plano z de un filtro de orden 1

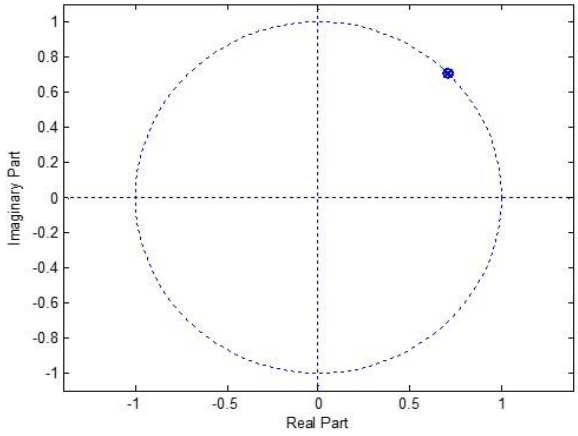


Figura 6.5: Ubicación de los polos y ceros en el plano z de un filtro de orden 5

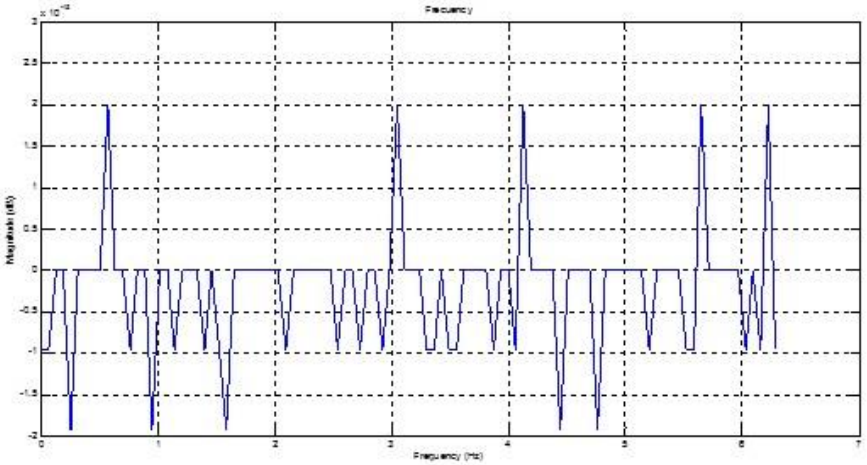


Figura 6.6: Respuesta en frecuencia en magnitud de un filtro de orden 1

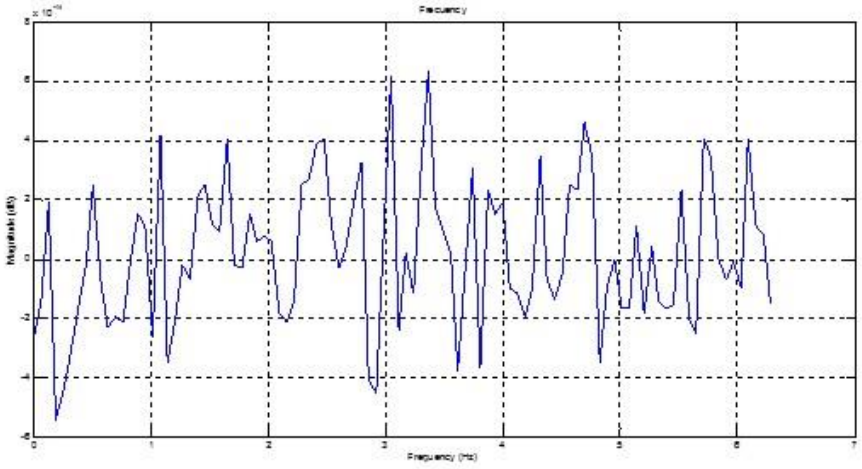


Figura 6.7: Respuesta en frecuencia en magnitud de un filtro de orden 5

En la figura 6.4 y 6.5 se puede ver la ubicación de los polos del filtro de orden 1 y 5. Como se espera, todos los polos y ceros se ubican en el mismo punto porque el valor absoluto de 1 y un ángulo de desfase de pi/4, aunque en el filtro 5 hay un mayor número de polos se encuentran en la misma orientación. Esto se asume que se produce porque el argumento de entrada que determina la orientación de los polos, el argumento a=pi/4, es el mismo en los dos polos. Por otro lado en la figura 6.6 y 6.7 se puede apreciar la respuesta en magnitud de los dos filtros. Se puede ver que en el filtro de orden 1, la respuesta es mucho más simétrica y rectangular, se puede decir que tiene picos negativos y positivos ligeramente espaciados en un patrón, mientras que la respuesta del filtro de orden 5 es mucho más distorsionada, no se puede apreciar un patrón de respuesta y esto se nota en la respuesta audible de la señal procesada, esta se atenúa en un menor tiempo.

6.4.1 Características de la señal de la entrada y salida

En esta sección se analiza las señales de entrada y salida del procesamiento, se busca ver la parte real, imaginaria y absoluta del procesamiento. Para fines del análisis, debido a que la simulación del tono de la guitarra posee valores aleatorios y la señal no tiene una forma definida, se usará una señal básica para el análisis de la forma de onda de la salida y entrada, mientras que la simulación del tono fue utilizada para filtrar la señal y escuchar el efecto audible que produce el procesamiento. Se utilizó un coseno con un vector de frecuencia de f=1000Hz como señal de entrada, la cual se puede ver en la figura 6.7:

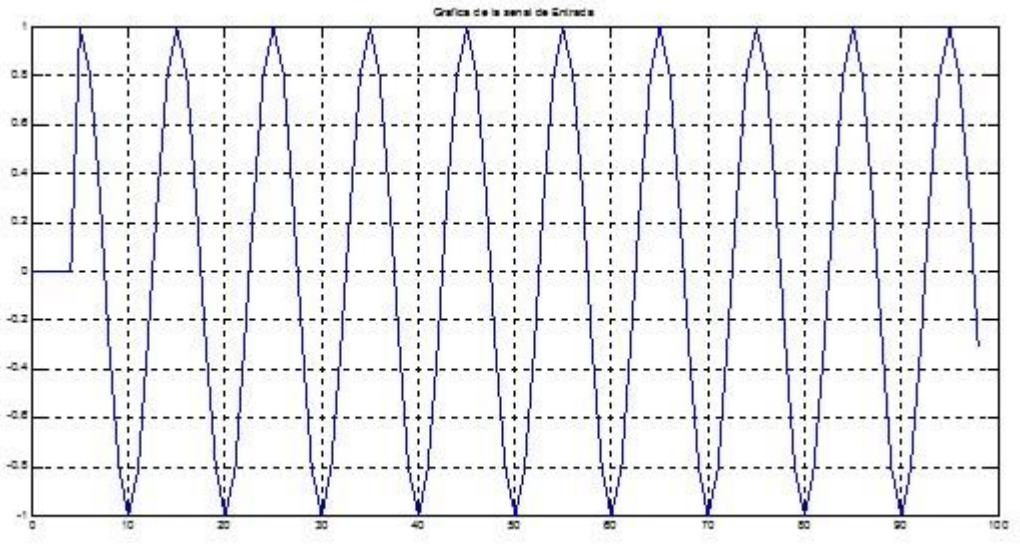


Figura 6.7: Señal de entrada para el análisis

Como se había mencionado, los parámetros que rigen el procesamiento son los polos de la función de transferencia y el orden del filtro que se concatena para producir un efecto más profundo.

Por ejemplo si tomamos un argumento a=exp(j*pi/4) que define el ángulo de los polos y tomamos un filtro de orden 1, se pueden ver las siguientes respuestas en la figura 6.8, 6.9 y 6.10:

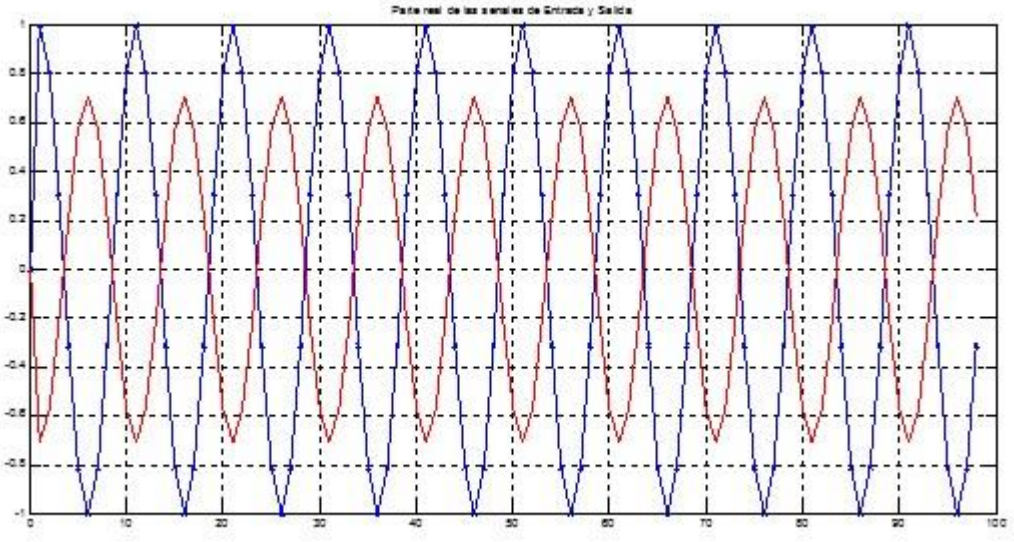


Figura 6.8: Forma de onda de la señal de entrada (azul) y de salida(rojo) del filtro de orden 1

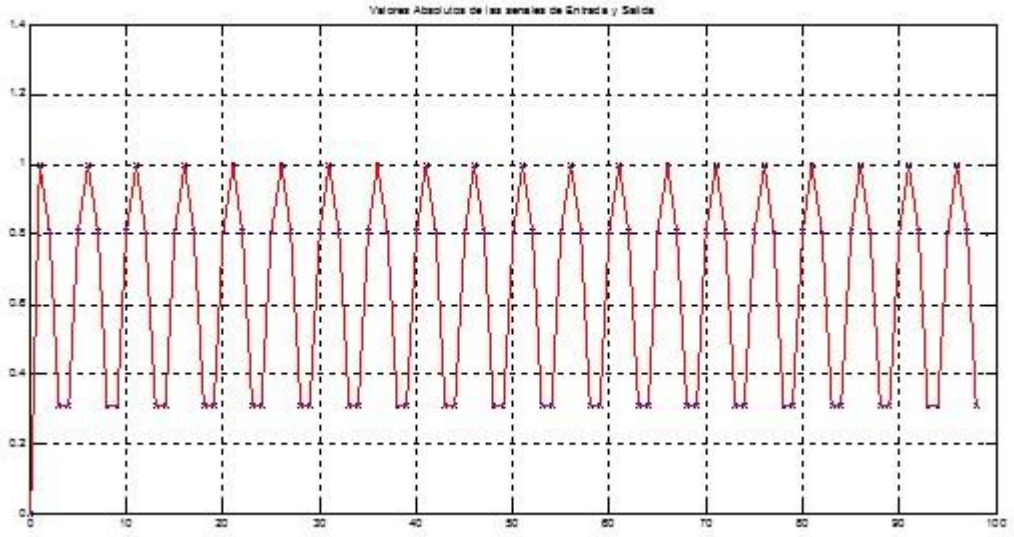


Figura 6.9: Valor Absoluto de la señal de entrada (azul) y de salida(rojo) del filtro de orden 1

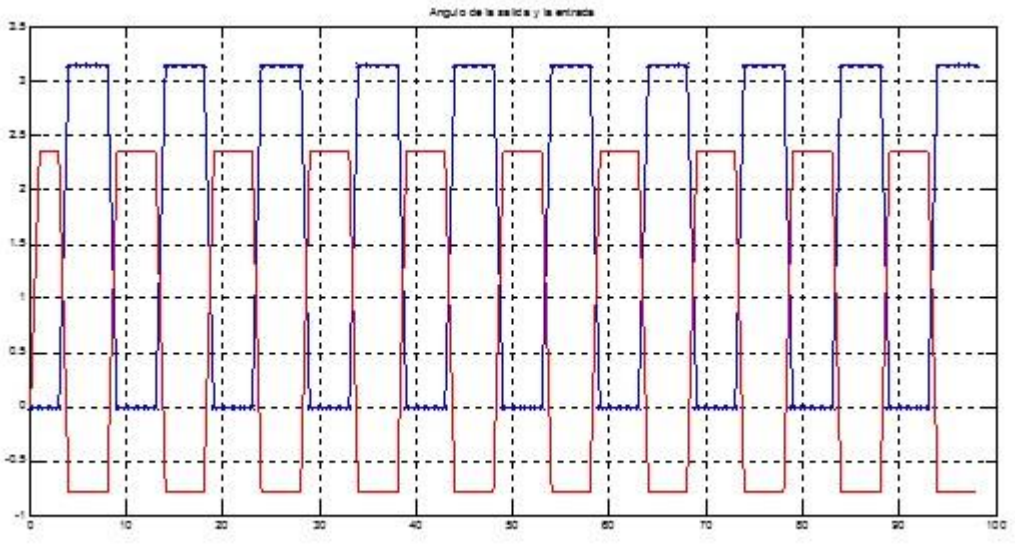


Figura 6.10: Ángulo de la señal de entrada (azul) y de salida(rojo) del filtro de orden 1

En la figura 6.8 se puede apreciar cómo se procesó la señal de salida. Aunque parece tener una amplitud menor que la entrada con la misma forma de onda, se debe recalcar que los valores imaginarios no se pueden ver. En la figura 6.9 efectivamente el valor absoluto de la señal de salida concuerda exactamente con el de la señal de entrada, esto verifica el comportamiento teórico del filtro, el cual predice un valor absoluto de su función de transferencia de 1, lo cual significa que la ganancia al final es la misma que la entrada. A continuación se realizará un breve análisis de la transformada de Fourier FFT y de la transformada discreta de Fourier DFT para entender de mejor manera el comportamiento del efecto que se busca.

6.4.2 Transformada de Fourier continua y discreta

En procesamiento de señales, una función en el tiempo es una representación de una señal con resolución del tiempo muy exacta, pero que no contiene información de la frecuencia, por esta razón, se usa la **Transformada de Fourier (FFT)**, la cual nos proporciona esta información de la resolución de la frecuencia pero con el inconveniente de no representar el tiempo. Un

punto de la magnitud de la transformada de Fourier representa la cantidad de frecuencia que se contiene en ese punto, pero su localización solo es dada en la fase de la misma transformada.

La definición en tiempo continuo de la FFT es:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}\,dt \qquad \text{forward transform} \tag{1}$$

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}\,d\omega \qquad \text{inverse transform} \tag{2}$$

X(w) es la FFT de una señal en el dominio en el tiempo x(t), la frecuencia angular w se define como 2*pi*f, donde f es la frecuencia análoga de una señal x(t). También se puede usar el proceso inverso para calcular la señal en el tiempo x(t) a partir de una FFT X(w). Al tener una componente compleja la FFT tiene la forma F(w)=a(w)+ib(w), la FFT usualmente se la representa en términos de magnitud y fase, es decir:

$$F(\omega) = |\,F(\omega)\,|\,e^{i\Phi(\omega)},$$ (F(w) es la fft de un función f(t), f(t) es la señal) (3)

Es conveniente definir la magnitud y ángulo porque son los necesarios para el análisis del efecto buscado.

$$|\,F(\omega)\,| = \sqrt{a^2 + b^2}$$ (Magnitud) (4)

$$\Phi(\omega) = \tan^{-1}\left(\frac{b}{a}\right).$$ (Fase) (5)

En el presente proyecto sin embargo, se necesita de la **forma discreta de la transformada de Fourier DFT** para representar la FFT en puntos discretos. La DFT está definida como:

$$X_k \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i k n / N}, \quad k \in \mathbb{Z} \text{ (integers)} \tag{6}$$

Xn es la señal muestreada que representa una señal continua x(t), N es el número de puntos que se utilizan para muestrear la señal Xn. La frecuencia de la señal es K ciclos cada N muestras. También se definen la magnitud y fase en tiempo discreto de la siguiente manera.

$$|X_k|/N = \sqrt{\mathrm{Re}(X_k)^2 + \mathrm{Im}(X_k)^2}/N \tag{7}$$

$$\arg(X_k) = \operatorname{atan2}\big(\mathrm{Im}(X_k), \mathrm{Re}(X_k)\big) = -i \ln\left(\frac{X_k}{|X_k|}\right) \tag{8}$$

Con estos conceptos se pretende analizar el espectro de frecuencias del efecto phaser. Con la misma señal analizada anteriormente, se ve el comportamiento de la transformada de Fourier con los mismos parámetros. En la figura 6.11 se ve que el valor absoluto de la magnitud de la FFT (eje y, escala normal) es igual en la entrada Xf (figura 6.11, figuras de la izquierda) y en la salida Yf en magnitud. Este respuesta era de esperarse debido a que el efecto predice una respuesta en magnitud de 1(ver capítulo 5).

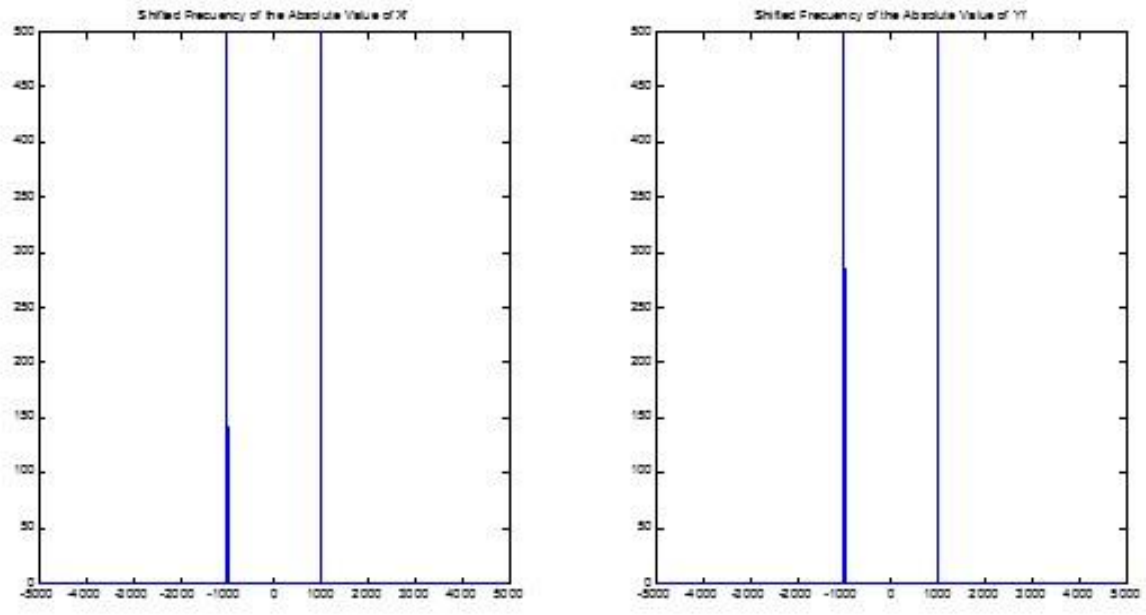


Figura 6.11: Comportamiento del valor absoluto de la FFT de la entrada y salida en un corrimiento de frecuencia. Filtro de orden 1.

Para el comportamiento de la fase de la FFT se encontró un problema. Ya que se definió a la fase como un arco tangente que relaciona el valor real e imaginario de cada muestra de la DFT en tiempo discreto, y al tener una señal con poco contenido espectral(1 frecuencia fundamental), el software MATLAB no puede interpretar de una buena manera los valores y arroja basura informática por cálculos no determinados. Es necesario usar una señal con mayor contenido espectral, el archivo *pruebas.m* (ver ANEXOS) permite aumentar una variación al coseno inicial para tener una mejor visualización del ángulo de la FFT. La nueva señal tiene la siguiente forma en el tiempo (ver figura 6.12).

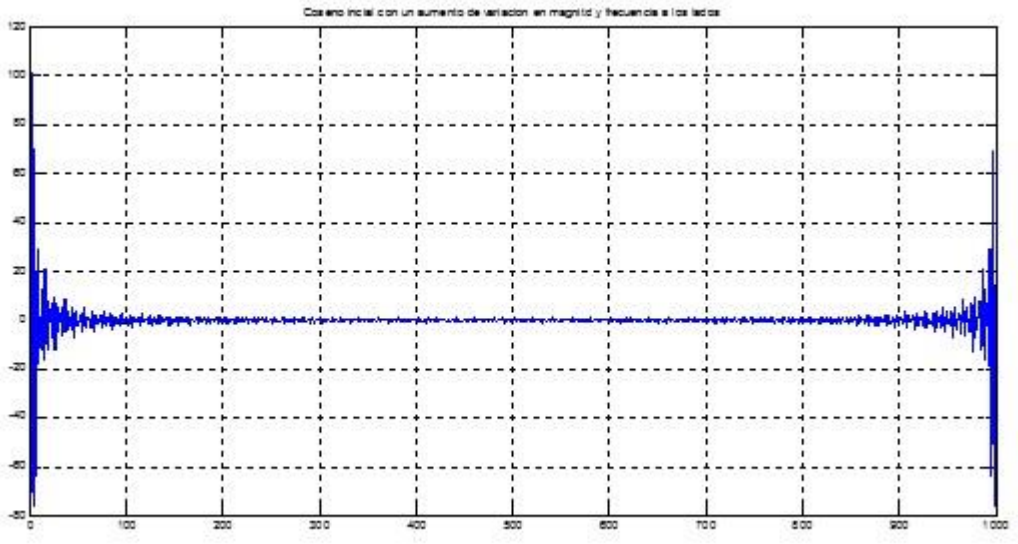


Figura 6.12: Coseno inicial con una variación de magnitud y frecuencia.

Esta señal permite que una banda de frecuencias tenga la misma magnitud, por lo tanto tendrá una respuesta en magnitud diferente al coseno inicial, esto se aprecia en la siguiente figura 6.13 (Se calculó la entrada Xf y la salida filtrada con el efecto Yf).

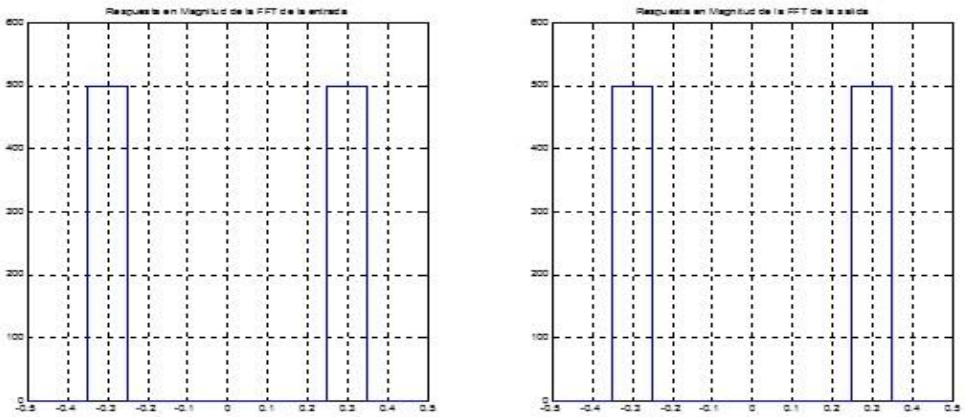


Figura 6.13: Respuesta en magnitud de la FFT del nuevo coseno, la banda entera de frecuencias tienen la misma magnitud en la entrada Xf y salida filtrada Yf(frecuencia normalizada)

Ahora que se tiene una banda de frecuencias más amplia se puede analizar el comportamiento de la fase de la FFT en la siguiente figura 6.14.

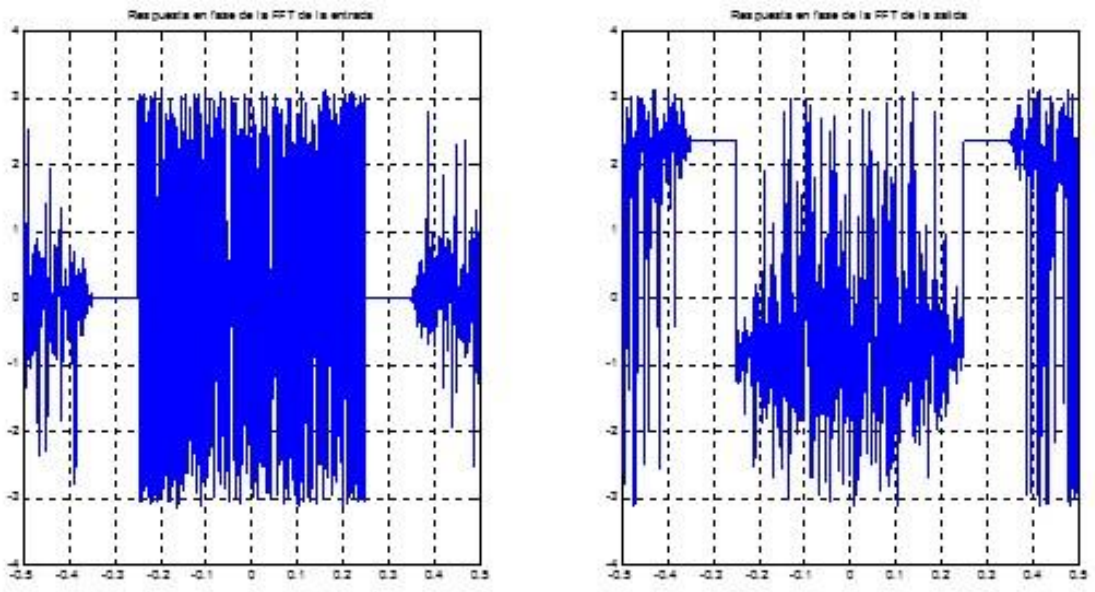


Figura 6.14: Respuesta de la fase de la FFT en la entrada Xf y la salida filtrada Yf.(frecuencia normalizada)

Como se puede ver, la banda entera de frecuencias se desplazó en fase a un punto superior de 2.356 radianes, lo que representa un desplazamiento de 135 grados. Más adelante se detallara la razón de este desplazamiento debido al filtro.

Ahora es recomendable analizar un filtro de mayor orden para ver su comportamiento en relación al filtro de orden 1. Se tomó un filtro de orden 5, con el mismo argumento a=exp(j*pi/4) , se obtuvieron las siguientes gráficas. Figuras 6.15-6.20.

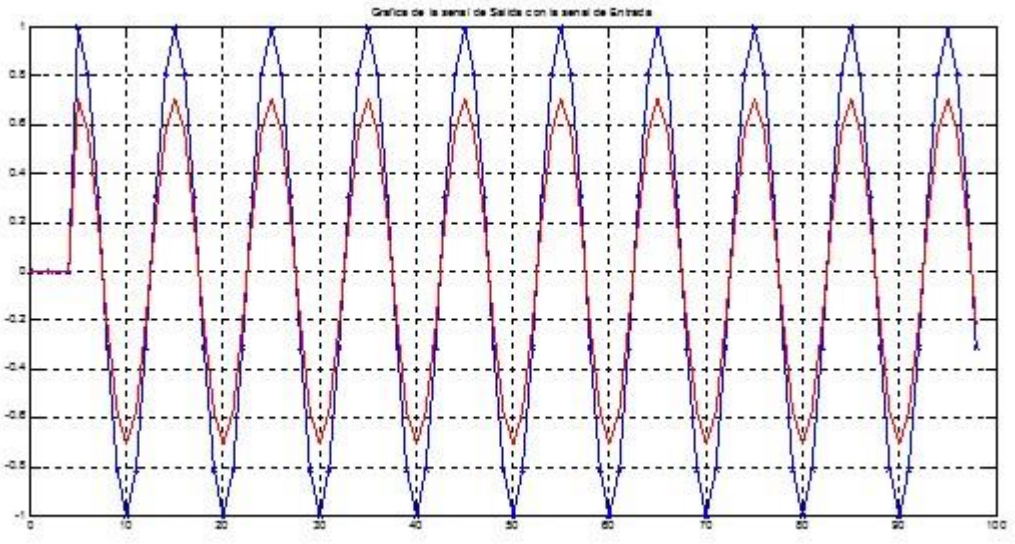


Figura 6.15: Forma de onda de la señal de entrada (azul) y de salida(rojo) del filtro de orden 5

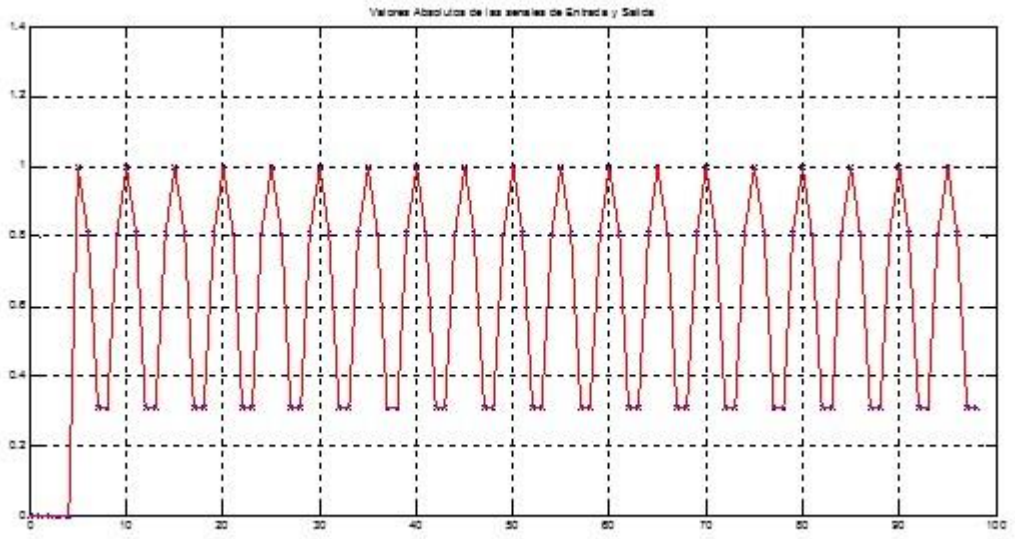


Figura 6.16: Valor Absoluto de la señal de entrada (azul) y de salida(rojo) del filtro de orden 5

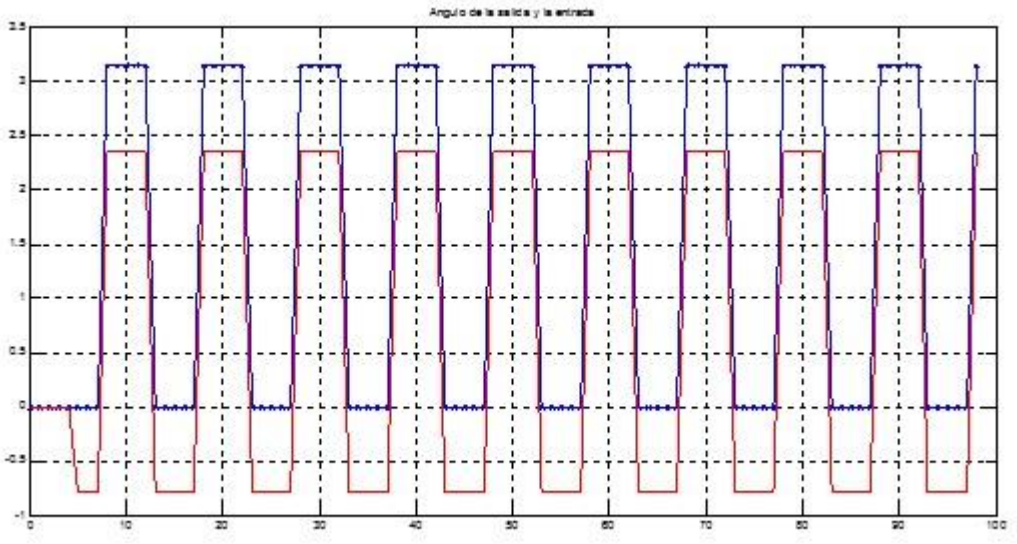


Figura 6.17: Ángulo de la señal de entrada (azul) y de salida(rojo) del filtro de orden 5

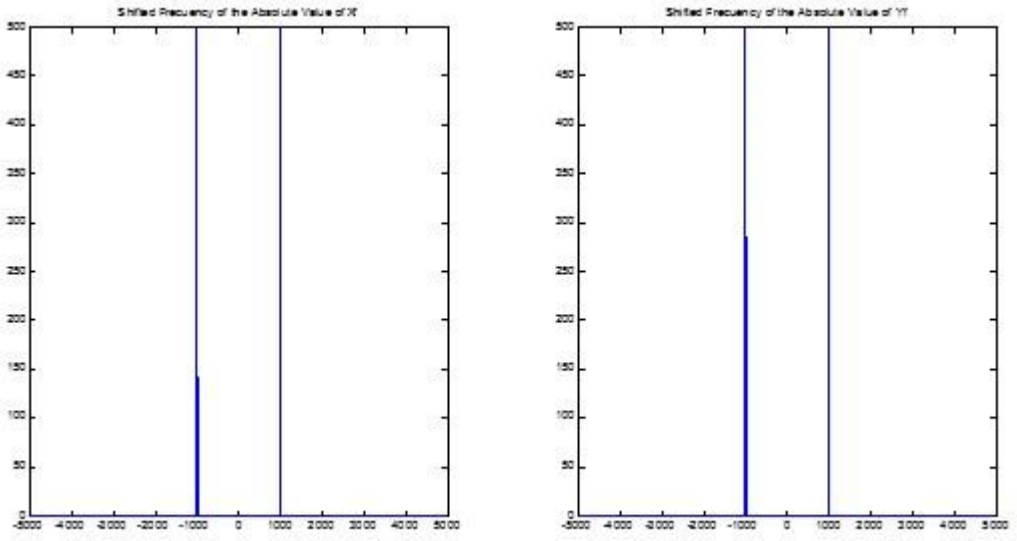


Figura 6.18: Comportamiento del valor absoluto de la entrada y la salida en su frecuencia original, y la frecuencia desplazada del filtro de orden 5.

Las figuras 6.15-6.17 muestran el comportamiento en el tiempo del coseno simple y la figura 6.18 muestra su respuesta en magnitud. Al igual que el filtro de orden 1, el filtro de orden 5 también nos brinda una respuesta de 1 en magnitud, es decir la magnitud es la misma en la

entrada y salida. Ahora, como se procedió en el filtro de orden 1, se usó la variación del coseno para visualizar el comportamiento de la magnitud y fase de una mejor manera.

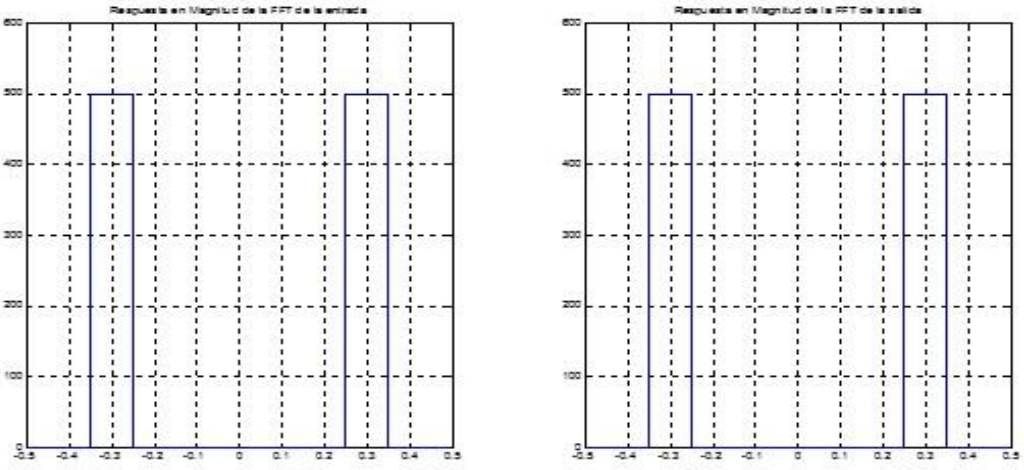


Figura 6.19: Comportamiento de la magnitud de la FFT del coseno modificado en la entrada y salida. (Frecuencia Normalizada)

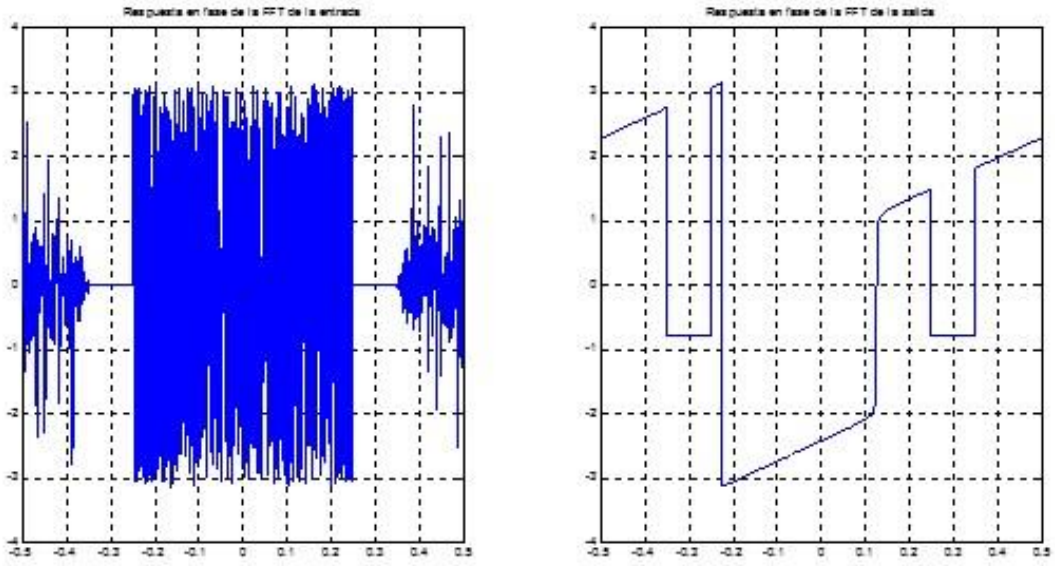


Figura 6.20: Comportamiento de la fase de la FFT del coseno modificado en la entrada y salida. (Frecuencia Normalizada)

Ahora el corrimiento de la fase con un filtro de orden 5 es -0.7854 radianes en la señal filtrada, lo que equivale a -45 grados. Al parecer cada orden de filtro desfasa la señal un ángulo diferente por orden.

El corazón de este análisis es ver el comportamiento de la fase con respecto a los parámetros de entrada. Las figura 6.10 (fase en el dominio del tiempo del filtro de orden 5) y figura 6.17(fase en el dominio del tiempo del filtro de orden 5) nos permitirán analizar el patrón de desfase que tiene una señal al ser aplicada un filtro pasa todos. Se usaron estas gráficas en los mismos filtros de orden 1 y orden 5 para ver cuál es el patrón de comportamiento del desfase tomando en cuenta los parámetros de entrada, es decir el argumento de los polos a.

Se tomaron 4 puntos, dos puntos de la salida y dos puntos de la entrada en posiciones diferentes en el tiempo en la gráfica de la fase(figura 6.10 para el filtro 1 y figura 6.17 para el filtro 5). Se usó el siguiente procedimiento para transformar estos puntos en radianes a grados y se los grafico a mano para encontrar un patrón.

Para el filtro 1,

Con a=exp(j*pi/4), es decir con teta_in=45°(el ángulo de entrada)

p1(4,3.142)(entrada x)
p2(4,-0.7854)(salida y)
p3(9,0)(entrada x)
p4(9,2.359)(salida y)
Variación_teta=135°

Con a=exp(j*pi/3), es decir con teta_in=60°(el ángulo de entrada)

p1(4,3.142)(entrada x)
p2(4,-1.047)(salida y)
p3(9,0)(entrada x)
p4(9,2.094)(salida y)

Variación_teta=120°

Con a=exp(j*pi/6), es decir con teta_in=30°(el ángulo de entrada)

p1(4,3.142)
p2(4,-0.5236)
p3(9,0)
p4(9,2.618)
Variación_teta=150°

Los puntos de la señal de entrada y salida representados en coordenadas polares del filtro de orden 1 se pueden ver en la figura 6.21 con diferentes ángulos de entrada, es decir , el argumento *a*(el punto 1 y 3 representan puntos de la entrada y los puntos 2 y 4 representan puntos de la salida).

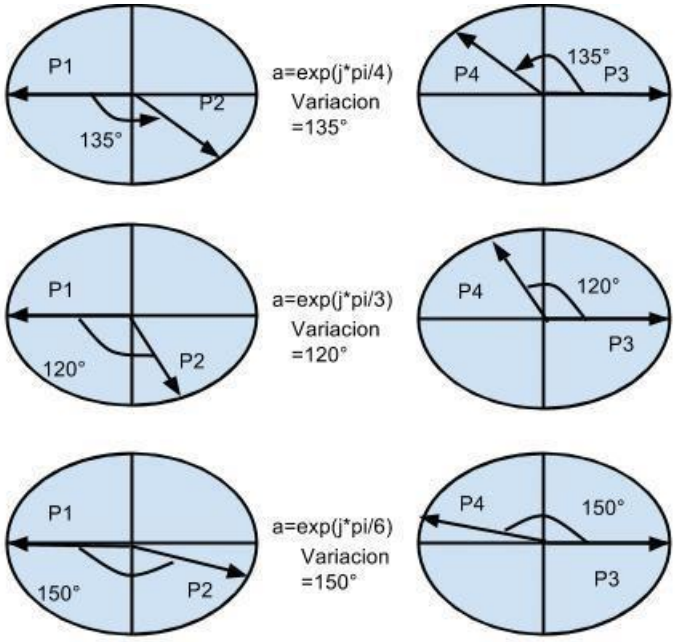


Figura 6.21: Desfases del filtro de orden 1 con diferentes ángulos de entrada (argumento a)

Ahora nos interesa saber cuál patrón sigue estas variaciones con respecto al ángulo de entrada. *Se propone que debido a que si se resta de 180° al ángulo de entrada, el resultado absoluto es la variación entre la señal de entrada y salida para determinado ángulo entrada.*

Ahora se realiza el mismo procedimiento de análisis de la variación de fase entre la señal de entrada y salida pero con un filtro de orden 5.

Para el filtro 5,

Con a=exp(j*pi/4), es decir con teta_in=45°(el ángulo de entrada)

p1(8,3.142)(Entrada x)
p2(8,2.356)(Salida y)
p3(13,0)(Entrada x)
p4(13,-0.7854)(Salida y)
Variación_teta=315°

Con a=exp(j*pi/3), es decir con teta_in=60°(el ángulo de entrada)

p1(8,3.142)(Entrada x)
p2(8,1.047)(Salida y)
p3(13,0)(Entrada x)
p4(13,-2.094)(Salida y)
Variación_teta=240°

Con a=exp(j*pi/6), es decir con teta_in=30°(el ángulo de entrada)

p1(8,3.142)(Entrada x)
p2(8,-2.618)(Salida y)
p3(13,0)(Entrada x)
p4(13,0.5236)(Salida y)
Variación_teta=30°

Los puntos de la señal de entrada y salida representados en coordenadas polares del filtro 5 se pueden ver en la figura 6.2 con diferentes ángulos de entrada (el punto 1 y 3 representan puntos de la entrada y los puntos 2 y 4 representan puntos de la salida).

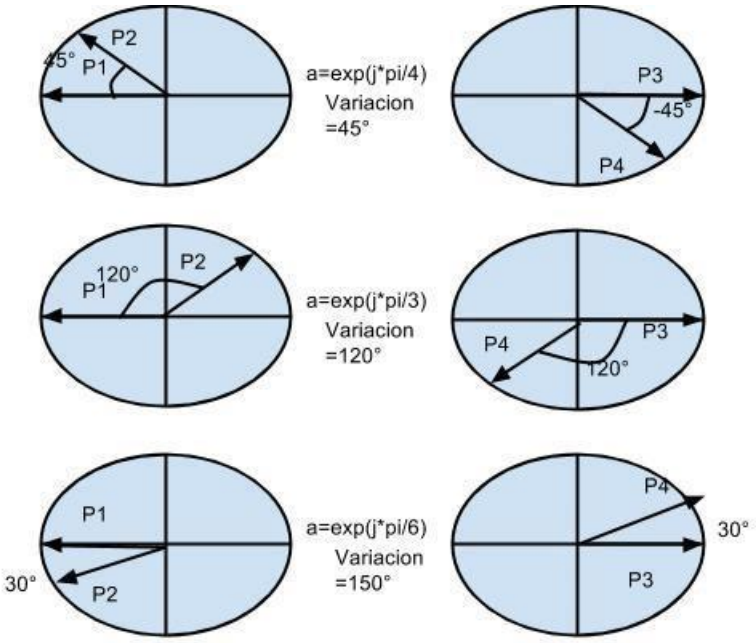


Figura 6.22: Desfases del filtro de orden 5 con diferentes ángulos de entrada(argumento a)

No se encontró un patrón de la variación de ángulo con relación al argumento a. Aun así la variación parece ser la misma entre el punto 1 y 2 y el punto 3 y 4.

Lo que se puede calcular, es una relación del orden del filtro con un valor a determinado. Se procedió a comparar las gráficas de fase de los 5 órdenes de filtro con un solo valor a=exp(j*pi/4), ver figura 6.23. Se encontró que si hay un patrón que relaciona estos órdenes con la variación entre la entrada y la salida. Se encontró que la ubicación de un punto en la salida con respecto a la entrada en el gráfico de la fase cambia de posición angular. La constante que mueve ese punto de la salida es efectivamente la variación de la entrada y salida Var_teta. Es decir que si la ubicación del primer punto de la salida es por ejemplo, -45 grados

para el filtro 1, la ubicación de ese punto en un filtro de orden 2 se ubicara en ese punto más 135 grados (variación de la entrada y la salida), y así sucesivamente se sumará 135 grados para los órdenes siguientes. Al menos para el filtro de orden 1 se concluye que el orden del filtro determina la ubicación inicial del desfase.

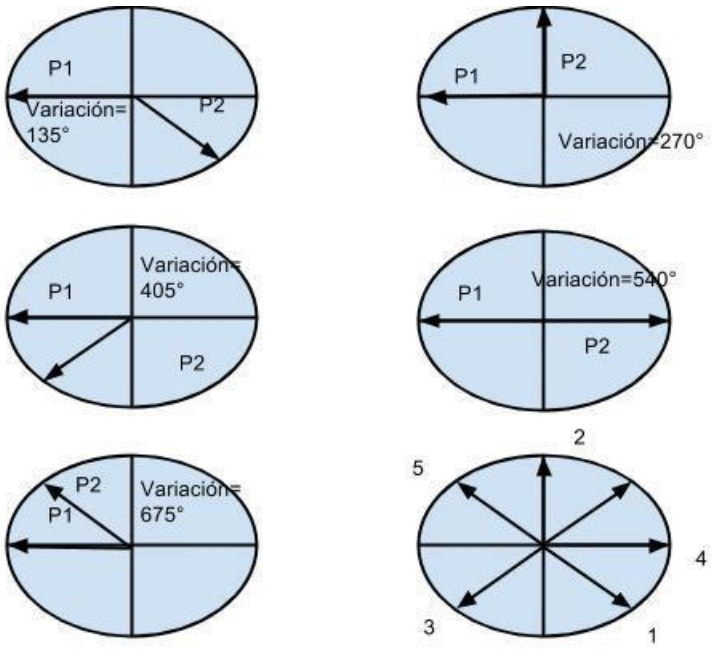


Figura 6.23: Patrón de desfases para todos los filtros con un solo valor angular de entrada

6.5 Filtros de ecualización

Con el objetivo de aplicaciones principales al proyecto, se diseñaron filtros de ecualización. Estos algoritmos son consultados del paper: ”*Digital Filter Design for Audio Processing*” de Ethan Elenberg, Anthony Hsu, Marc L´Heureux, Stephanie Ng, Alaap

Parikh, E.J. Thiele, Michelle Yu (Ver bibliografía). El algoritmo principal es desarrollar filtros que atenúan o incrementen potencia en las bandas especificadas en ciertas frecuencias centrales que se definirán a continuación.

Como se mencionó antes, el rango de frecuencias de la guitarra eléctrica es de 82Hz a 1400Hz aproximadamente. Aun así el rango más usado entre los guitarristas es de 82Hz a 700Hz aproximadamente porque son las notas que más se usan. Se probaron varias frecuencias centrales y anchos de banda de cada frecuencia pero al final se llegó al consenso de elegir las siguientes frecuencias centrales y sus respectivas bandas.

-Para filtrar las frecuencias bajas: Frecuencia central: 150Hz, Banda: 400Hz

-Para las frecuencias medias: Frecuencia central: 300Hz, Banda: 400Hz

-Para las frecuencias altas: Frecuencias altas: 700Hz, Banda: 600Hz

Ahora que se han definido estas frecuencias el algoritmo consiste en calcular los coeficientes de los términos de la función de transferencia del filtro de acuerdo con los parámetros que se requieren y se detallaron anteriormente. La función de matlab *parameq.m(ver anexos),* permite calcular estos coeficientes introduciendo la ganancia que se requiere aumentar o disminuir de acuerdo a la siguiente fórmula:

```
beta = tan(Dw/2) * sqrt(abs(GB^2 - G0^2)) / sqrt(abs(G^2 - GB^2));
b = [(G0 + G*beta), -2*G0*cos(w0), (G0 - G*beta)] / (1+beta);
a = [1, -2*cos(w0)/(1+beta), (1-beta)/(1+beta)];

%Donde Dw es la banda, GB es la ganancia que se busca reducir o aumentar,
G0 es la ganancia de base de referencia, G es la ganancia tope de referencia
y w0 es la frecuencia central.
```

Posteriormente se convulsionan los coeficientes de los tres filtros, de la banda de bajos, medios y altos, para obtener el filtro final que obedece a los mismos comportamientos antes descritos. El archivo *effect.m(ver anexos)* realiza todos los pasos anteriores con argumentos de entrada que se dan por el usuario. En el caso de este análisis se querrá disminuir -5dB a la banda de bajos, aumentar 5dB a la banda de medios y aumentar 10 dB en la banda de altos. La figura 6.24 muestra la respuesta en frecuencia del filtro, la señal de entrada (archivo de audio) y la señal de salida(señal filtrada).

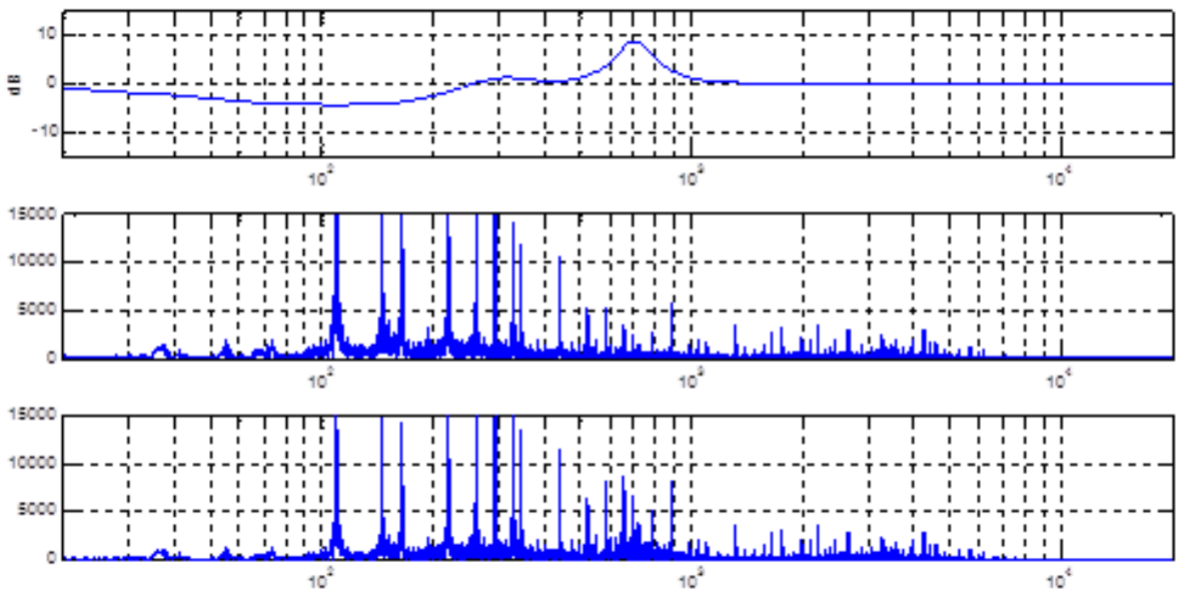


Figura 6.24: Respuesta de los filtros de ecualización para atenuar o aumentar los decibeles de una banda deseada.

Como se puede ver, las frecuencias se atenúan o aumentan en potencia conforme se manipule la cantidad de ganancia que se desea aumentar o disminuir.

Después de que el programa predice este comportamiento del filtro se tradujo estos códigos a lenguaje python para ser usados por el raspberry pi. El argumento de entrada de los filtros son los decibeles que se pretenden aumentar o disminuir en una rango de -5dB a 5dB en cada banda (ver interface en secciones posteriores).

CAPÍTULO 7: DESARROLLO DE LA INTERFAZ Y CONTROL REMOTO

7.1 Esquema de la interfaz gráfica y funcionamiento de su entorno

7.1.1 Parámetros

Una interfaz permite al usuario interactuar y controlar las funciones de cualquier programa. Para el desarrollo de la interfaz gráfica es relevante determinar los parámetros que impactan sobre el efecto deseado porque estos serán los argumentos que el usuario ingresará en el programa. Con el objetivo de tener completo control sobre el efecto se definen los siguientes parámetros de control y a su vez se detalla el tipo de comandos que se usarán para controlar dicho parámetro:

SECCIÓN DE FEEDBACK y PHASING

Se usará un **botón de filtro** que activa el procesamiento desarrollado a partir del filtro teórico. Se incluirá **barra deslizamiento** para regular el desfase controlando el parámetro del cual depende la profundidad del filtro porque cambia la función de transferencia con cada nuevo valor. Al ser las señales periódicas, este argumento solo alcanzar un valor límite, luego de este no será necesario porque se repite la profundidad del efecto. Por ejemplo, a 0° los valores extremos de la modulación son alcanzados simultáneamente para todos los canales. 180° o -180° es igual la distancia más grande posible de las modulaciones de fase de los canales. Además se incluirá una **barra de deslizamiento** el orden del filtro, y una **barra de deslizamiento** que controle la ganancia del efecto. Adicionalmente se añade en la interface un LED que permite ver al usuario cuando el efecto está en funcionamiento y cuando no.

SECCIÓN DE LA DISTORSIÓN Y VOLUMEN

Se añadirá un **botón de distorsión** que activará esta opción con su respectiva **barra de desplazamiento** para controlar el nivel de distorsión. Finalmente se añade una **barra de desplazamiento** adicional para controlar el nivel de la ganancia de la salida. Finalmente se añade un LED para indicar la activación o inactivación del efecto en el sonido entrante.

En los últimos días del proyecto se adicionaron **tres barras de deslizamiento que controlan los filtros de ecualización**. Las longitudes de estas barras son de un valor de -5 hasta 5, que representan decibeles de atenuación o aumentación de la ganancia de esas bandas de frecuencia.

Tkinter es un paquete para desarrollar GUIs(Interfaces de Usuario Gráficas). Es una capa orientada a objetos por lo que no se necesita construir sus objetos que representan funcionalidades básicas para la construcción de GUIs. Es decir se debe leer la documentación de las funciones de la librería y simplemente usar esas funciones para construir la aplicación. Se optó por usar esta librería como base porque integra objetos que no ocupan mucho espacio y lo que se busca es optimizar todos los recursos para que el procesamiento de la señal no tenga retardos.

Tkinter para el desarrollo de la Interface

WIDGETS

-LABEL.-Es una texto o una imagen en la pantalla. Se pueden desplegar contenidos, pero es recomendable usar canvas.

-FRAME.-Es una región rectangular en la pantalla. Es usado como un widget maestro, es decir el widget de referencia, para contener otros widgets. Se usa principalmente dentro de los labels.

-PACK.-Este paquete se encarga de administrar la posición de los widgets. Se manejan los widgets con un widget maestro padre. Es recomendable usarlo para los siguientes fines:

-Ubicar un widget en un frame y rellenarlo en el frame entero

-Ubicar un número de widgets arriba o abajo de otros widgets

-Ubicar un número de widgets lado a lado.

-SCALE.-Permite al usuario seleccionar un valor numérico moviendo un slider a través de una barra. Se pueden controlar los valores máximos y mínimos así como también la resolución.

La siguiente imagen representa el prototipo final de la interface del proyecto con los componentes que antes se detallaron (ver figura 7.1)

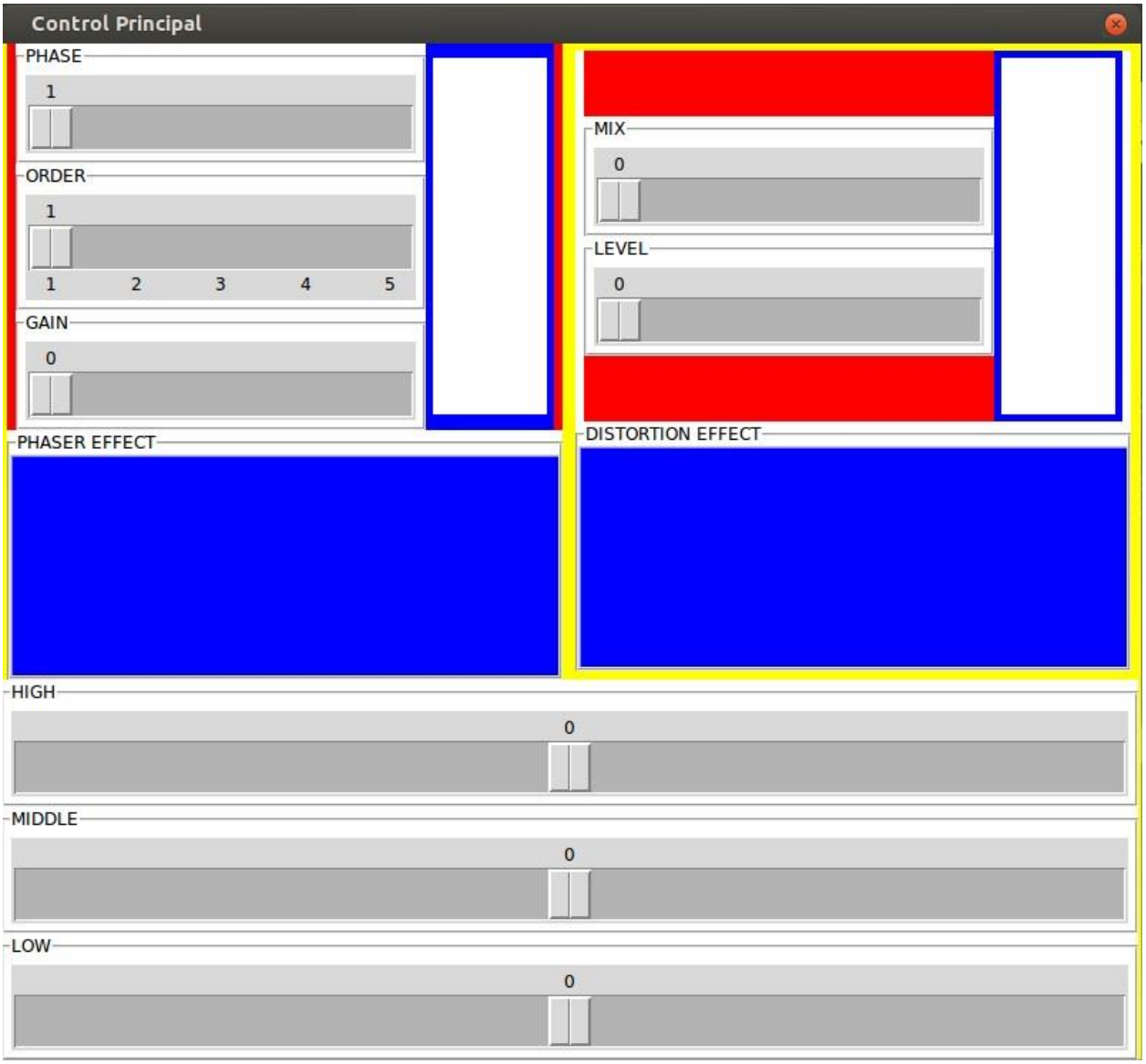


Figura 7.1: Interfaz principal de control del proyecto

7.2 CONTROL REMOTO DESDE LA COMPUTADORA

En muchas ocasiones se necesita controlar los dispositivos de procesamiento de manera rápida y fácil y muchas veces no se dispone de un teclado, un ratón y una pantalla display para dicho dispositivo (raspberry pi). En esas ocasiones es conveniente usar la computadora para manejar directamente al dispositivo. Para esto se usa el control remoto que es el uso de softwares especializados y protocolos de comunicación para usar la computadora

como medio de comunicación y control. Para conseguir este objetivo es necesario saber conceptos básicos de redes de computadoras, tales como servidores, direcciones IP y protocolos de acceso remoto, los cuales se detallan en breve a continuación.

SERVIDOR.- Un servidor es el procesamiento de una aplicación software capaz de aceptar peticiones de un cliente y dar respuestas acorde con la petición. Las computadoras de mesa tradicionales tienen servidores locales porque son capaces de aceptar respuestas y enviar peticiones. Los servidores operan en una arquitectura cliente-servidor. Esto facilita al cliente compartir datos, información o cualquier recurso hardware o software que se requiera. Los clientes normalmente se conectan al servidor a través de una red. En el contexto de un protocolo IP, un servidor es un programa que opera como un receptor socket (un punto final de un proceso de flujo de comunicación a través de una red de computadoras). Ejemplos típicos de servidores de data son: servidores de datos, servidores de archivos, servidores de mail, servidores de web, servidores de juegos, etc. En teoría cualquier proceso computarizado que comparte un recurso a uno o más clientes es un servidor. En el contexto de hardware, un servidor se designa a modelos computacionales que alojan aplicaciones software bajo una alta demanda de entorno de red.

COMUNICACIÓN IP.- La dirección IP es un arreglo de números separados por puntos que identifica cada computadora usando el protocolo de Internet para comunicar esta computadora sobre una red específica. Pueden existir dos tipos de IP debido a la clasificación de redes públicas o privadas.

IP pública.- Es cualquier dirección o número que puede ser accesada en el internet. Los grupos reguladores estándares del Internet como el Centro de Información de Redes(Network Information Center, NIC) o la Autoridad de Números Asignados de Internet (Internet Assigned Numbers Authority, IANA), son las únicas organizaciones responsables de registrar los rangos IP a los Proveedores de Servicios de Internet (ISPs).

IP privada.- Es cualquier dirección o número que se le asigna a un dispositivo electrónico o a una red de área local TCP/IP y aquel número es sólo accesible dentro de la red de área local.

Lo que es útil al presente proyecto es la dirección IP privada en una red local para comunicar una laptop o computadora de mesa con el dispositivo procesador de audio.

Protocolo y servidor SSH.- Un servidor SSH es un programa de software que usa un protocol secure shell para aceptar conexiones remotas de computadoras externas. Los transferidores de archivos SFTP/SCP y conexiones remotas de terminales son populares para un servidor SSH.

Servidor VNC(Virtual Network Computing-Computación de red virtual).- Este es un programa de software que en esencia es un escritorio gráfico remoto de compartición de sistema que usa el protocolo Buffer de Frames Remotos (RFB-Remote Frame Buffer) para controlar remotamente otra computadora.

7.2.1 IP estática para comunicación

Este proyecto hace uso de direcciones IP privadas pero estas direcciones deben ser configuradas en el raspberry pi de manera que sean estáticas, es decir que no cambien continuamente, se producen muchos contratiempos por las direcciones Ip dinámicas porque siempre el usuario tiene que estar en busca de la dirección IP actual. Para conseguir este objetivo se debe modificar algunos archivos del sistema raspberry porque el protocolo DHCP(The Dynamic Host Configuration **Protocol**) para asignación automática de IPs es el que está predeterminado en el dispositivo.

Para empezar, se debe obtener la información de las direcciones IPs actuales, si se quiere configura las vías ethernet o LAN se obtener estas direcciones, para este objetivo el comando *ifconfig.* La siguiente figura 7.2 muestra este comando en la terminal.

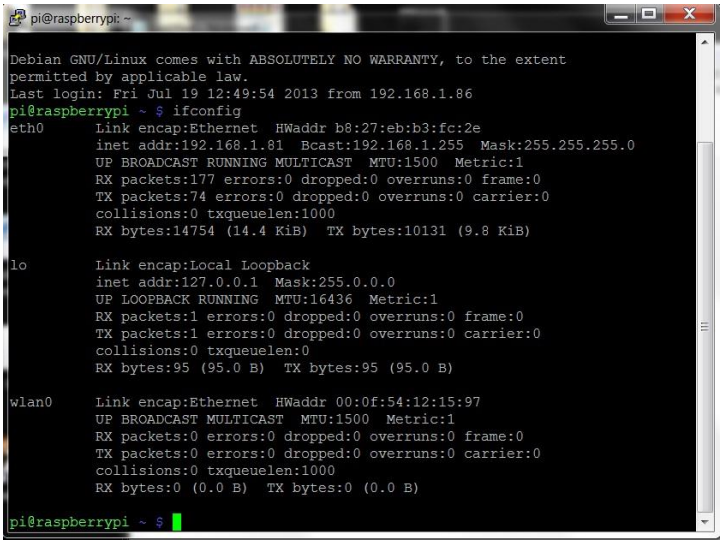


Figura 7.2: Comando Ifconfig para ver el estado de la red del dispositivo

Se deben anotar los siguientes datos vía LAN y vía ethernet: Dirección IP, Submascara, Broadcast.

Ahora se debe editar el siguiente archivo con el siguiente comando:

$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

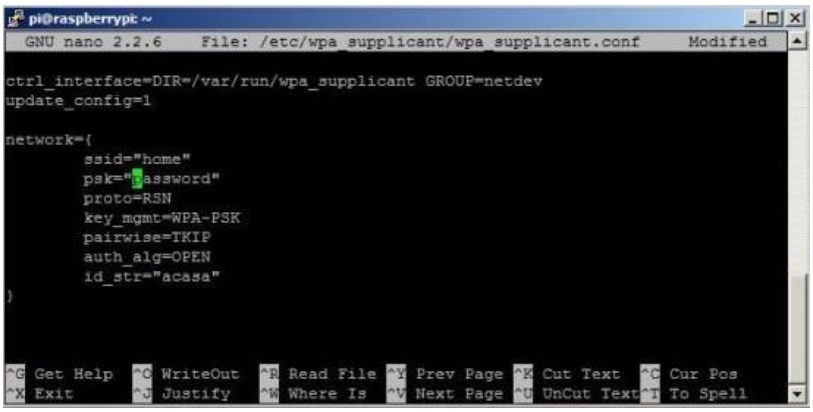


Figura 7.3 Archivo wpa_supplicant para editar características de red

En este caso, la red LAN proporciona la configuración del acceso, lo principal es aumentar la última línea de código *id_str=”acasa”*, este comando sirve para darle una identificación estática a la red LAN.

Finalmente se debe modificar el archivo: **sudo nano /etc/network/interfaces**. Se debe añadir las líneas que faltan como se detalla a continuación (cambiar las direcciones IP, broadcast, mascara, gateway).

```
auto lo
auto eth0
iface lo inet loopback
iface eth0 inet static
address 192.168.0.108
gateway 192.168.0.1
netmask 255.255.255.0

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface acasa inet static
address 192.168.0.104
netmask 255.255.255.0
gateway 192.168.0.1
```

```
iface default inet dhcp
```

Para acceder al destinatario (destination) y la puerta de acceso, usar el siguiente comando:

```
$netstat -nr
```

Si todo se ha efectuado correctamente al iniciar el sistema las redes se configurarán de manera que el sistema use IPs estáticas.

7.2.2 Control Remoto Raspberry Pi(En el lado del servidor para VNC)

Se puede configurar al raspberry Pi para que sea el servidor de clientes remotos ya sea para que los clientes remotos solo manipulen la terminal de comandos del dispositivo (servidor SSH) o para que se use un escritorio remoto(servidor VNC).

En el caso del servidor SSH se requiere una configuración previa en el raspberry pi, se debe acceder a la interfaz de configuración de la tarjeta con el siguiente comando:

```
$sudo raspi-config
```

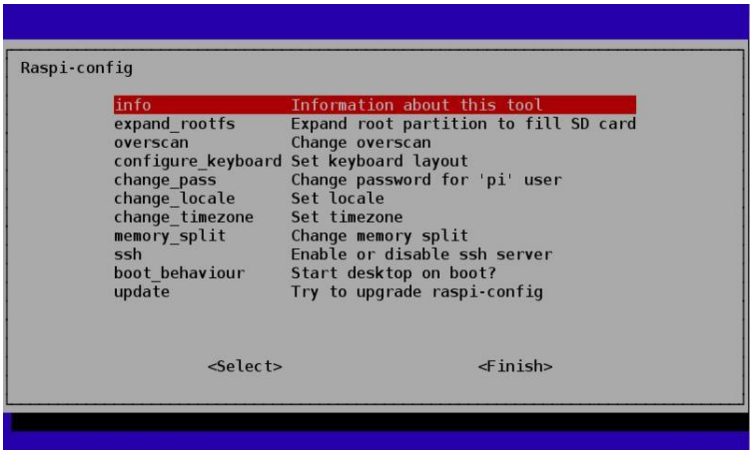


Figura 7.4: Menú principal de configuración del Raspberry Pi.

Con este comando ir a la sección de opciones avanzadas y habilitar la opción SSH(es recomendable reiniciar el sistema después de esto).

Para lograr una conexión remota satisfactoria con un servidor VNC(Escritorio remoto) se debe configurar al raspberry pi para que sea capaz de aceptar clientes remotos, para esto se sigue el siguiente proceso en la siguiente página oficial del raspberry pi:

https://www.raspberrypi.org/documentation/remote-access/vnc/README.md

Para ambos servidores es recomendable poseer una IP estática como se detalló en la sección anterior

7.2.3 Control Remoto en Windows (En el lado del cliente)

Para lograr un control remoto con una computadora con sistema operativo Windows (en el proyecto se usó una computadora windows y una en linux), se debe instalar un software previos para un servidor SSH o un servidor VNC.

SSH para Windows

La documentación oficial de raspberry pi sugiere descargar el programa PUTTY, el cual puede ser descargado en la siguiente página:

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Además se debe, para usar este programa, simplemente introducir la dirección IP estática que se detalló en la sección anterior; y se podrá acceder a la terminal de comandos del raspberry pi de manera satisfactoria. En esta terminal se debe introducir el usuario y contraseña del dispositivo (en caso de haberlos) y se puede manipular finalmente el dispositivo.

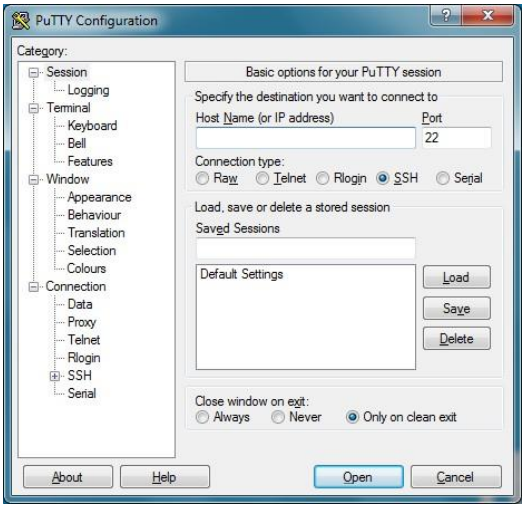


Figura 7.5: Interfaz de la comunicación entre servidor SSH y el Raspberry Pi en Windows.

VNC para Windows

La documentación oficial de raspberry pi sugiere el programa TightVNC para el servidor VNC, el cual puede ser descargado en la siguiente página:

http://www.tightvnc.com/download.php

Una vez instalado el programa, el proceso es simple, introducir en el programa la dirección estática IP e introducir la contraseña del raspberry pi para acceder a la ventana VCN, la cual muestra el escritorio del Raspbian en el caso de una conexión satisfactoria.

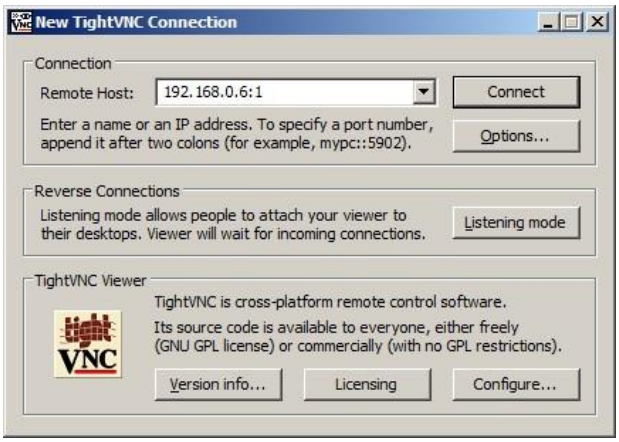


Figura 7.6: Interfaz del servidor VNC de Windows para controlar el Raspberry Pi remotamente.

7.2.4 Control Remoto en sistemas UNIX (En el lado del cliente)

Los sistemas UNIX (Ubuntu, Raspberry pi) tienen mayor libertad para la comunicación, para el proyecto fue muy necesario manipular rápidamente el raspberry pi desde el sistema ubuntu, ya que desde la computadora basada en Ubuntu se desarrollaban los archivos para la aplicación. Para obtener este tipo de comunicación se asume que se configuró al raspberry pi con una IP estática ya sea vía ethernet o vía LAN.

Para el caso del cliente SSH no se necesita softwares previos en el lado del cliente, sólo se requiere ingresar a la terminal (en este caso, la terminal de comandos de ubuntu), e ingresar la siguiente línea de código:

```
$ssh pi@ "introducir la dirección ip"
```

Si la dirección Ip está correcta, la terminal pedirá las credenciales del dispositivo, una vez introducidos el usuario y la contraseña se podrá ver que la sintaxis de la entrada de comandos cambiará a verde y será el nombre del usuario del raspberry pi, esto quiere decir que la terminal ahora se ha convertido en la terminal del raspberry pi.

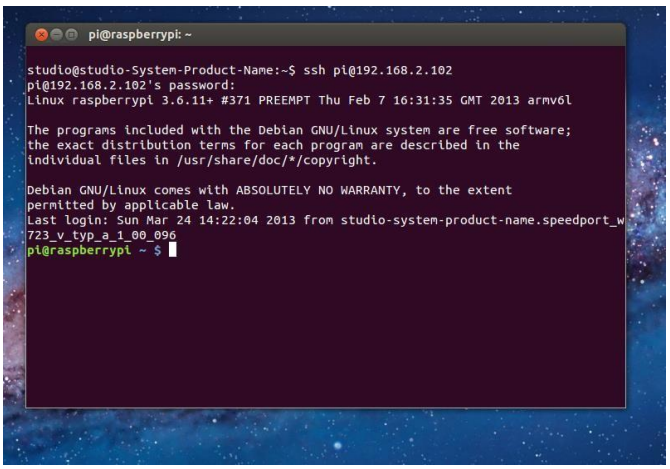


Figura 7.7: Comunicación SSH en Ubuntu Linux para controlar Raspberry Pi remotamente.

Para el caso del cliente VNC, la última versión de ubuntu (versión 15), la cual está instalada en la máquina del presente proyecto posee una interfaz gráfica instalada por default en el sistema, esta interfaz se puede acceder en la interfaz de búsqueda del sistema ubuntu, el programa se llama REMMINA, como se puede ver en la siguiente página.

Figura 7.8: Programa Remmina para controlar el Raspberry Pi con un servidor VNC en ubuntu-linux.

Luego introducir en la sección de *comunicación básica* la dirección IP estática del raspberry, y la contraseña del dispositivo. Si todo se efectuó con éxito, se podrá acceder finalmente al escritorio del raspberry pi.

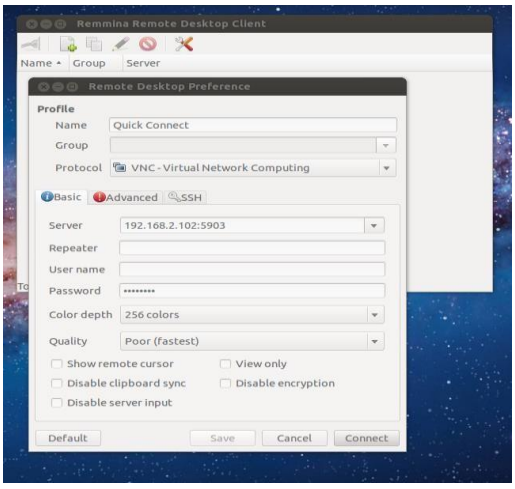


Figura 7.9: Interfaz del programa Remmina(Servidor VNC) para controlar el Raspberry Pi remotamente.

Para más información, visitar la página oficial del control remoto del raspberry:

https://www.raspberrypi.org/documentation/remote-access/

CAPÍTULO 8: PROGRAMACIÓN

8.1 Controles Principales

8.1.1 Comandos Principales del Raspberry Pi

Raspberry Pi es controlado principalmente con comandos Shell. Es decir se introducen comandos específicos para las principales tareas que se necesitan en un microprocesador o una computadora personal. Por ejemplo, a través de la terminal de comandos se instalan nuevos programas, se actualiza el software del sistema operativo, se manipula el estado de las conexiones del dispositivo, se instalan drivers de los puertos de entrada y salida y se realizan las principales tareas de control como inicio del sistema y apagado del sistema.

Una vez que el raspberry pi inicia, la única manera de iniciar el desktop del dispositivo (Raspbian para el raspberry pi), es introducir el siguiente comando para que la ventana del Raspbian aparezca, ver figura 8.1.

```
$startx
```

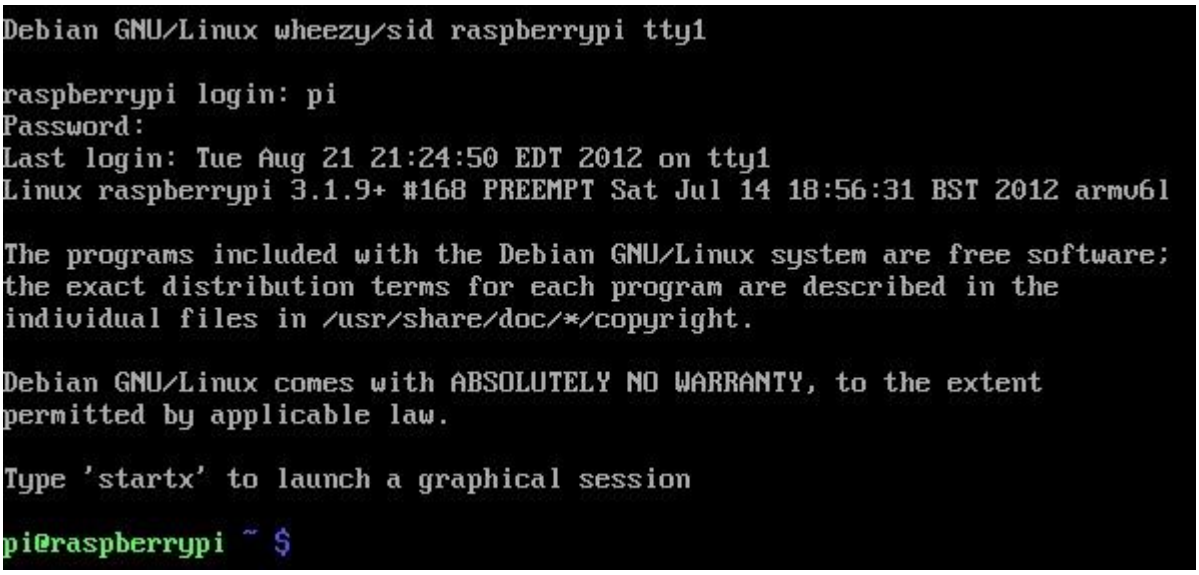


Figura 8.1: Inicio del sistema Raspbian

En el caso que el Raspberry Pi se haya configurado con nombre de usuario y contraseña ingresarlos y luego se iniciará el sistema.

Por otro lado, una vez que se haya terminado con las tareas en el dispositivo, lo más recomendable es apagar el sistema con la terminal de comandos o con la interface Raspbian. Con el siguiente comando se apaga el dispositivo

```
$shutdown now -h now
```

Algo que se debe realizar cada cierto tiempo es actualizar los paquetes de las librerías del sistema operativo, la comunidad Linux actualiza los archivos UNIX constantemente, entonces el siguiente comando debe ejecutarse cada cierto tiempo:

```
$sudo apt-get update
```

8.1.2 Principales librerías y actualizaciones del raspberry pi

En algunas ocasiones debido a errores humanos o a errores técnicos de hardware y software es necesario restaurar las configuraciones de fábrica del hardware y software de cualquier tarjeta. En el presente proyecto hubo muchas ocasiones que se dio este caso guiando a la conclusión de la necesidad de un reinicio de sistema. Como se mencionó en la sección del seteo inicial del raspberry pi se debe crear una imagen del sistema en una tarjeta micro SD pero ya que el proyecto necesita de un IDE(Entorno de programación), librerías específicas para el proyecto y los drivers de los módulos de audio(Hifi Berry Dac+), se especifica los comandos directos para la instalación de estas librerías y actualizaciones.

-Para la instalación del IDE geany:

```
sudo apt-get install geany
```

-Para la instalación de las librerías específicas para el proyecto:

```
sudo apt-get install python-pyaudio
sudo apt-get install python-numpy python-scipy
sudo apt-get install python-imaging
sudo apt-get install python-tk
sudo apt-get install python-qt4
sudo apt-get install python-imaging-tk
```

8.2 Lenguaje de Desarrollo

8.2.1 Desarrollo en Python

Se usó el entorno Python27 debido a que es el lenguaje usado comúnmente para interactuar con la comunicación externa de los sensores digitales y análogos del raspberry pi. Python es un lenguaje de programación de alto nivel y para propósitos generales. La filosofía del diseño enfatiza la lectura apropiada del código y su sintaxis permite a los programadores expresar conceptos en pocas líneas de código que podrían ser posibles en lenguajes como C++ o Java. Incluye múltiples paradigmas de programación como orientación a objetos, programación imperativa y funcional o estilos de procedimiento. Puede ser ejecutado en una gran variedad de sistemas, en este caso para el proyecto se usó linux ubuntu.

Se optó usar Python porque los archivos .py son scripts que consumen muy poca memoria y al usar un lenguaje tan básico no requieren de software extra para ejecutarse. El proyecto se desarrolló en diferentes archivos .py que están divididos estructuralmente para facilidad de programador. Para ejecutar cualquier archivo python generalmente se sigue el siguiente procedimiento:

-Abrir la terminal en el sistema basado en linux, en este caso el computador con sistema ubuntu o el raspbian para el raspberry pi

-Localizar la carpeta que contiene el archivo .py a ejecutarse.

-Ejecutar el siguiente comando: *python3 name_of_the_file.py*

Adicionalmente en el proyecto se usó un IDE, el cual se detallará a continuación, para tener una mayor comodidad en la escritura de archivos python

8.2.2 IDE PyCharm y Geany

Un IDE(Integrated Development Environment) es una aplicación de software que provee herramientas que facilitan el desarrollo de software a los programadores. Principalmente consiste en un editor de código fuente, automatización de compilación y herramientas de debug. Algunas veces el IDE contiene interfaces GUI para crear otras interfaces, las acciones de este tipo de interfaces se traducen a fuente código y es más fácil conseguir rápidos resultados. En el presente proyecto se usó PYCHARM para la edición de los archivos de la aplicación en computadoras externas. Estos archivos luego serían trasladados al dispositivo de procesamiento(Rapsberry Pi) para ejecutarse a través de otro IDE llamado GEANY, el cuál no es muy pesado. Ambos IDEs contienen herramientas básicas para el desarrollo de software, pero sobretodo fue de gran utilidad fue el panel de control que sirve para ver errores de compilación y el reconocimiento de caracteres en el código fuente, es decir, el IDE es capaz de resaltar características de objetos, clases, etc. Esta utilidad es de gran ayuda cuando el código es muy grande porque ayuda al programador a depurar errores. El IDE es capaz de reconocer los errores de sintaxis antes que se intente correr el programa, lo cual es muy útil. Ver figura 8.2 y 8.3

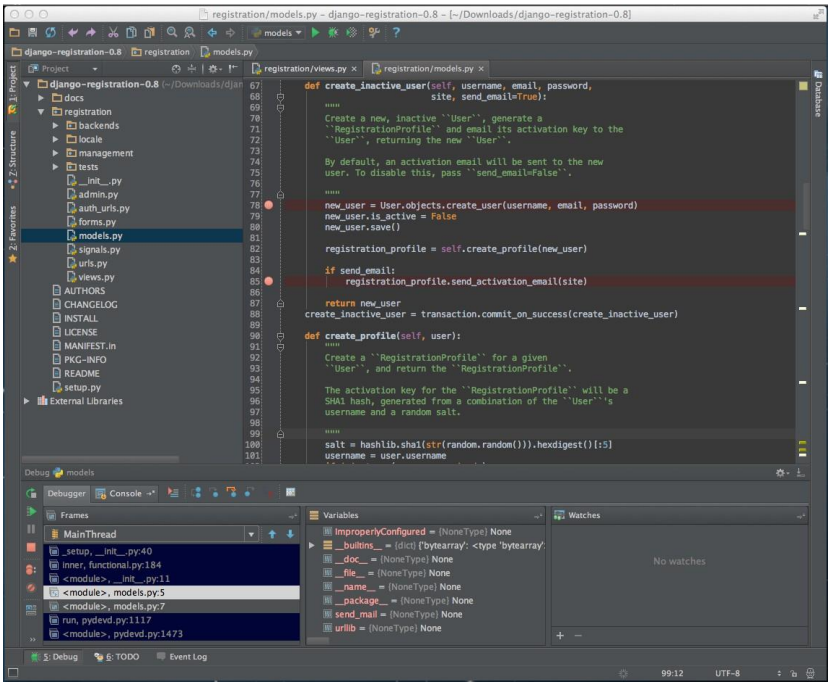


Figura 8.2: IDE PyCharm para Computadora

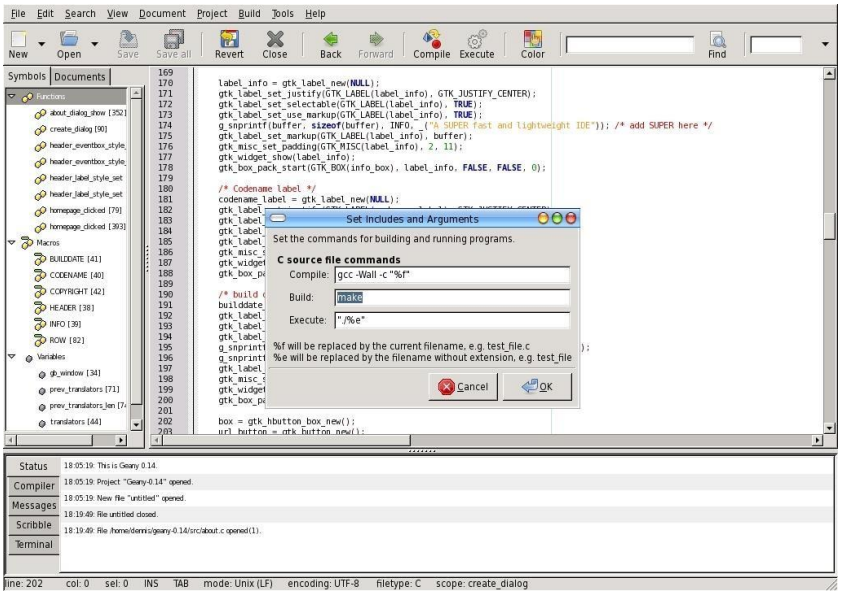


Figura 8.3: IDE Geany para Raspberry Pi

8.2.3 Librerías Usadas

Para el desarrollo de la aplicación se requieren de ciertas librerías python que comúnmente no se encuentran en el sistema operativo en desarrollo. Para el desarrollo del

proyecto se construyó la aplicación desde una computadora basada en linux, por motivos de experimentación previos a la implementación del código en la plataforma raspberry pi. Las librerías necesarias para desarrollar la interfaz descrita en la sección previa son:

PyAudio y PortAudio para lectura de Audio

PyAudio provee un acceso y unión a la librería PortAudio(Disponible para todos los sistemas operativos). Es una librería diseñada para escribir programas de audio simples en C o C++. PortAudio provee APIs simples para grabar o reproducir sonidos usando funciones prediseñadas o interfaces de lectura y escritura. Los usos más comunes de esta librería son la reproducción de ondas sinusoidales, procesamiento de entradas de audio, grabación de archivos de audio, etc.

Scipy para el procesamiento de la señal digital y Numpy para el manejo correcto del tipo de data de los arreglos

SciPy es una colección de paquetes para tareas científicas y es desarrollado por una comunidad de científicos orientados al desarrollo de software científico. Es una colección de algoritmos numéricos y cajas de herramientas de dominios específicos, incluyendo procesamiento de señal, optimización, estadísticas y mucho más. Numpy es el paquete fundamental de esta colección para computación numérica. Este paquete define tipos de arreglos y matrices numéricas y operaciones básicas entre ellos.

Math para caracteres matemáticos especiales

Math es un paquete interno por default en el paquete predeterminado de python, sin embargo es necesario actualizarlo igual que todos los paquetes. Este provee acceso a las funciones matemáticas definidas por el lenguaje C estándar. Si se requiere de números complejos, se debe hacer uso de la librería cmath(complex math).

Tkinter para el desarrollo de la Interface

Se explicó en la sección anterior

Image para mostrar imágenes en los frames de la aplicación

El módulo IMAGE que viene en el paquete de python por default, provee una clase que es usada para representar una imagen PIL(Python Imaging Library). El módulo también provee un número de funciones de fábrica, incluyendo funciones para cargar imágenes de archivos o crear nuevas imágenes.

Threads para programación en hilos

En ciencia computacional, un thread de ejecución es un pedazo de código que puede ser manejado independientemente por el programa maestro. Múltiples threads o procesos pueden ejecutarse a la vez si se maneja de manera correcta la comunicación entre ellos, el inicio y final y el tiempo de ejecución de cada uno. Más adelante se explicará la importancia de esta librería en el desarrollo de esta aplicación.

8.3 Programación de la Aplicación

En esta sección se explicará de manera breve los principales comandos que intervienen en la aplicación principal y cómo funcionan.

8.3.1 Importación de librerías

Al inicio de todo archivo python, así como en otros lenguajes como C, C++, android, etc, se debe importar las librerías o archivos que contienen funciones abstraídas que cumplen tareas específicas necesarias para la aplicación. Como se mencionó en la sección anterior, las siguientes librerías fueron necesarias para que la aplicación pueda funcionar de manera correcta.

```
import pyaudio
import numpy as np
import scipy.signal
from PIL import ImageTk
import time
from threading import Thread
import Tkinter as tk
import math
from PIL import Image
from Tkinter import *
import PIL
```

8.3.2 Definición de variables relevantes

Las siguientes variables definen la configuración del muestreo que se pretende realizar, características tales como el tamaño del buffer en cada iteración, el formato de lectura de datos, el número de canales del puerto de entrada, la frecuencia de muestreo, etc.

```
global CHUNK
CHUNK = 2048
global FORMAT
FORMAT = pyaudio.paInt16
```

```
global DTYPE
DTYPE = np.int16
global CHANNELS
CHANNELS =1L
global RATE
RATE = 48000
global RECORD_SECONDS
RECORD_SECONDS = 20
```

La siguiente variable *flag* fue esencial para la activación de los switches de la aplicación, se usó un arreglo de valores void True y False, los cuales cambiarán de acuerdo a una función de control ligada a los WIDGETS de la interface, así el programa cambiará automáticamente el efecto de procesamiento que se elija.

```
global flag
flag=[False,False]
```

8.3.3 Estructura de la Aplicación GUI y variables de los WIDGETS

Una aplicación GUI, para facilitar el funcionamiento y rendimiento del sistema, se debería estructurar a manera de una clase. En esta clase se definen funciones locales, variables locales, etc. En la siguiente figura se puede ver en donde va el código de la interface.

```
class main_window(Frame):

   def __init__(self,master):

       #AQUÍ SE DESARROLLA LA APLICACIÓN…
```

En el programa general se llama a la clase de la interface definiendo una variable, con la función **.mainloop()** la aplicación se ejecuta constantemente a no ser que el usuario salga del programa.

```
def main():
   while True:
       global app
       root=Tk()
       app=main_window(root)
```

```
  root.protocol("WN_DELETE_WINDOW", onclosing)
       root.mainloop()
       while True:
           pass
if __name__ == '__main__':
   main()
```

Dentro de la interface, las variables locales que controlan los parámetros de los efectos son variables que provienen de la librería tkinter(GUI) para que sean capaces de vincularse con los WIDGETS que se usarán.

```
global var_phaser_level
var_phaser_level=tk.DoubleVar()
global var_phaser_order
var_phaser_order=tk.DoubleVar()
global var_phaser_gain
var_phaser_gain=tk.DoubleVar()

global var_distortion_level
var_distortion_level=tk.DoubleVar()
global var_distortion_mixer
var_distortion_mixer=tk.DoubleVar()
global var_distortion_gain
var_distortion_gain=tk.DoubleVar()

global var_high_level
var_high_level=tk.DoubleVar()
global var_middle_level
var_middle_level=tk.DoubleVar()
global var_low_level
var_low_level=tk.DoubleVar()
```

8.3.4 Programación en hilos

La programación en hilos fue fundamental para el funcionamiento de la aplicación. la razón es que se necesita que el procesamiento de la señal no se detenga en ningún momento, es decir debe existir una función que corra independientemente de la interface o cualquier otro proceso. al inicio hubo muchos problemas de programación sin el uso de esta librería ya que el

programa se quedaba atascado en un loop infinito que no permitía a la aplicación ejecutar la interfaz. Pero el loop infinito es necesario en el procesamiento porque incluso sin el procesamiento de los efectos la señal debe ser leída y debe transmitirse a la salida. Al final con el uso de esta librería se creó una función **get_phaser_level()** que llama a un thread(proceso) que corre todo el tiempo, esta función se ubica en la interfaz de la siguiente manera así como su definición.

```
get_phaser_level(None)

def get_phaser_level(var):
   t1 = Thread(target=processing)
   t1.start()
```

Luego de declarar la función que llama a la función de procesamiento **processing()**, se debía tener un control de switches de los efectos que se usaban, para eso se creó una función que cambiaba la variable que hacía las veces de semáforo en la aplicación, como se mencionó en la sección 8.2.2. Esta función **negar()**, por ejemplo, está vinculada con la interfaz de la manera siguiente.

```
button_distortion=Button(distortion_frame,text="DISTORTION
EFFECT",width=380, height=150,bg="blue",image=img_distortion)
button_distortion.bind('<Button-1>', lambda event:negar(flag,1))
button_distortion.pack()
```

Al aplastar el botón la función cambia los valores de esta variable flag, esta variable se leerá a continuación en la sección de procesamiento.

```
def negar (flag, i):
   flag[i] = not flag[i]
   app.cambiar_color(i)
```

8.3.5 Procesamiento de la señal

Para iniciar la lectura de datos se debe abrir el puerto de entrada con la instrucción **.PyAudio().** la instrucción **.open()** permite abrir el buffer con las especificaciones necesarias que se detallaron anteriormente en la definición de variables las cuales tienen que ver con la configuración de la lectura de datos digitales.

```
global p
p = pyaudio.PyAudio()
global stream stream =
p.open(format=FORMAT,channels=CHANNELS,rate=RATE,input=True,output=Tr
ue,frames_per_buffer=CHUNK,input_device_index=2)
```

Una vez que se llega al programa **processing()**, este leerá los valores de los WIDGETS de la interfaz de la siguiente manera.

```
highvar=float(var_high_level.get())
middlevar=float(var_middle_level.get())
lowvar=float(var_low_level.get())
highvar=10**(highvar/20)
middlevar=10**(middlevar/20)
lowvar=10**(lowvar/20)

#variables para phasing
gain_phaser=float(var_phaser_gain.get())
order=float(var_phaser_order.get())
theta = float(var_phaser_level.get())
```

Aquí se definen los parámetros de los efectos como variables de lectura con la función **.get()**, los valores que se cambian constantemente en la interfaz se enviarán a estas variables para el procesamiento posterior.

8.3.6 Lectura de Datos

Ahora que las variables son obtenidas, se procede a realizar una lectura de continua, la función **processing()** se encuentra en un loop infinito, en el cual cada iteración tratará de leer la

entrada en vectores del tamaño de buffer especificado en las variables iniciales. Para evitar que se lean vectores vacíos, se añade una excepción de error de manera que si el vector está vacío, lo que se lee es un vector de ceros.

```
try:
   string_audio_data = stream.read(CHUNK)
   #print(str(len(string_audio_data)))
except IOError as ex:
   if ex[1] != pyaudio.paInputOverflowed:
       raise
   string_audio_data= '\x00' * CHUNK*2
```

En la lectura es necesario la conversión de unidades. El comando .read() lee el tamaño del buffer en bytes por muestra. Es decir, si se leen 2048 de tamaño de buffer, los cuales son representados como INT16(16 bits), entonces se tiene 32768 bits en cada loop. El vector que se lee continuamente es un vector de strings de longitud 2048. Ahora este vector de strings no puede ser procesado porque necesitamos representación en números. Para esto se usa la función **fromstring()** de la librería **numpy** el cual permite la conversión a números enteros de 16 bits dándonos un vector final de 2048 números enteros, como se quería.

8.3.7 Procesamiento limpio

Ahora que se tiene un vector de información, se procede a elegir el tipo de procesamiento de acuerdo a las banderas de la variable **flag**. Si **flag** es igual a False, False, el procesamiento es limpio, es decir, la señal se lee normalmente y se escribe de la misma manera con la instrucción **write()** de la librería pyaudio.

8.3.8 Procesamiento Phasing

En el caso que flag == True, False , significa que el switch del procesamiento del efecto phaser está activado y se procede a realizar los siguientes cálculos. Se usa el parámetro del desfase obtenido de la escala de la interfaz. Con la librería **numpy** se puede hacer uso de

números complejos y con la librería **math** se puede hacer uso de funciones trigonométricas, las cuales son necesarias para definir el parámetro final que está presente en los vectores a y b de la función de transferencia que filtra la señal. También se lee el orden del filtro, si el orden es mayor que 2, la función multiplica estos vectores para obtener la función de transferencia final.

```
if flag==[True,False]:

      #variables para phasing
      gain_phaser=float(var_phaser_gain.get())
      order=float(var_phaser_order.get())
      theta = float(var_phaser_level.get()) r=1
      cor_x=r*np.round(math.cos(theta),decimals=4)
      cor_y=r*np.round(math.sin(theta),decimals=4)

      a=cor_x+j*cor_y
      a_cong= np.conjugate(a)
      af=np.array([-a_cong, 1])
      bf=np.array([1, -a])

      if order>=2:
          afinal=af
          bfinal=bf
          for ii in range(0,int(order)-1):
              afinal=np.array(scipy.signal.convolve(afinal,af))
              bfinal=np.array(scipy.signal.convolve(bfinal,bf))
      else:
          afinal=af
          bfinal= bf
```

Posteriormente se procede a filtrar la señal con el comando **.lfilter()** de la librería scipy, la cual tiene las funciones semejantes a la función filter de MATLAB. Es decir devuelve el vector filtrado asumiendo que se ingresan el numerador y denominador de la función de transferencia. Para mayor precisión en los cálculos se usó una conversión al tipo double.

Finalmente se convierte el vector de números double a los strings originales que fueron leídos. Con la instrucción **tostring()** se convierte el tipo de data y está listo para escribirse con la instrucción **.write().** Esta última sección es compartido con todos los tipos de procesamiento.

```
y = np.double(scipy.signal.lfilter(bequa,aequa, audio_data))

y=np.int16(y)

string_audio_data = y.tostring()

stream.write(string_audio_data)
```

8.3.9 Procesamiento Distorsión

En esta sección se habla rápidamente en que se basa el efecto distorsión ya que no es de mayor dificultad que la implementación de la siguiente ecuación (Udo Zolzer):

$$f(x) = \frac{x}{|x|}\left(1 - e^{\frac{x^2}{|x|}}\right) \quad (1)$$

Esta función pretende usar una variable de ganancia que es la cantidad de distorsión en el sonido y mix que es el nivel de la mezcla con la señal original y la señal distorsionada. Primero se aplica la ganancia a la entrada y luego se aplica la ecuación. Si mix = 0, la señal tiene toda la señal distorsionada mezcla a y si mix =0, la señal está limpia.

```
if flag==[False,True]:
      #variables para distortion
      gain=float(var_distortion_level.get())
      mix=float(var_distortion_mixer.get())
      audio_data = np.fromstring(string_audio_data, dtype=DTYPE)
      audio_data=np.double(audio_data)
      maximum_in=np.double(max(np.absolute(audio_data)))
      q=np.double(audio_data*gain/maximum_in)
      z=np.double(np.sign(-q)*(1-np.exp(np.sign(-q)*q)*1.1))
```

```
maximum_out=np.double(max(np.absolute(z)))
m=mix*np.array(z)*maximum_in/maximum_out+(1-mix)*audio_data
maximum_m=np.double(max(np.absolute(m)))
y=np.double(np.array(m)*maximum_in/maximum_m)
#rewritting y=np.int16(y)
string_output=y.tostring()
stream.write(string_output)
```

8.3.10 Procesamiento de ambos efectos

En caso que flag==True, True se procede a realizar los dos efectos simplemente juntando las instrucciones de los efectos anteriores y escribiéndolos de la misma manera. Se debe tener cuidado con la ganancia de cada uno de los efectos en la programación.

8.4 Flujo de Datos

A continuación se detallará los parámetros más relevantes que se usaron para este procesamiento en tiempo real de una señal análoga. Para empezar se utilizó una frecuencia de muestreo fs=48000. Con esto se puede deducir que el tiempo de muestreo es ts=1/fs=20us. Es decir, cada 20us se toma una muestra de la señal análoga y se la interpreta. Posteriormente se utilizaron varios tamaños de buffer, es decir varios valores del número de muestras que se utilizarían por loop de procesamiento. Después de las pruebas se llegó a un número de n=2048. Este valor nos sirve para calcular un parámetro muy relevante que es el período de muestreo T=n/fs=42.67ms. Esto quiere decir que en teoría, la lectura de 2048 muestras cada 20us llevaría 42.67ms en cada loop. Este valor será relevante en futuros cálculos.

Ahora es conveniente realizar los cálculos del manejo de bits que utiliza el dispositivo de procesamiento de acuerdo a los parámetros dados. En el presente proyecto, por motivos de calidad de audio y reducción mínima de costo de procesamiento, se eligió un formato de

lectura del puerto de entrada de enteros de 16 bits(Int16) de la librería pyAudio, es decir cada muestra contiene 16 bits de representación. Luego de esto se hace un cálculo del número de bits por segundo que el dispositivo va a procesar. La velocidad de bits es Rb=fs*#bits=48000(muestras/seg)*16(bits/muestra)=768000bits/seg. Ahora es conveniente calcular el número de bits en nuestro tamaño de buffer n=2048:

2048muestras*(16bits/muestra)=32768bits.

Esto quiere decir que 32768bits son leídos en cada loop de procesamiento de tamaño de buffer n=2048 muestras. Ahora finalmente se puede predecir el tiempo que se demora en leer en cada loop de procesamiento de la siguiente manera:

32768bits*(seg/768000bits)=42.67ms

Esto quiere decir que en teoría se toma 42.67 ms en lectura de 2048 muestras con 16bits por muestra, tal como se predijo en el cálculo del muestreo anterior. Este tiempo es un valor referente en el cual se debiera optimizar la programación de manera que todo el procesamiento en cada loop para transformar la señal de entrada en una salida filtrada, sea menor que el tiempo de lectura que se predice.

8.4.1 Flujo de Datos con comandos específicos

En el código anterior se mostró el método processing() el cual es encargado del procesamiento, ahora se muestra un flujo de datos en los comandos más principales del programa, explicando el funcionamiento de cada uno.

Primero se definen variables globales, en especial el tamaño del buffer es un parámetro importante.

```
CHUNK=1024
```

Una vez que se especifica el tamaño del buffer se especifica el número de bits por muestra así como se mencionó en la sección anterior.

```
FORMAT=pyAudio.paInt16
```

Estos dos parámetros son claves a la hora de setear las características de la lectura del puerto de entrada, con la siguiente instrucción se identifica el formato, el número de canales del dispositivo, la frecuencia de muestreo(RATE) y el tamaño del buffer.

```
stream=p.open(FORMAT,CHANNELS,RATE,BUFFER=CHUNK)
```

Posteriormente se lee este tamaño de buffer con la siguiente sentencia, la cual interpreta la entrada en un vector de strings porque la computadora interpreta esos valores como símbolos aleatorios.

```
string_audio_data=stream.read(CHUNK)#vector de strings, símbolos
```

Como se van a manejar operaciones matemáticas para el procesado, se requiere convertir estos símbolos en números. Se procede a convertirlos en números enteros de 16 bits.

```
audio_data=np.fromstring(string_audio_data,DTYPE=np.Int16)
```

Para mayor precisión en las operaciones matemáticas se utiliza doble precisión, aunque con un costo de mayor contenido de procesamiento.

```
audio_data=np.double(audio_data)(tipo float64)
```

Con este vector de doble precisión se efectúa una serie de operaciones para llegar a la señal filtrada. Por último se utiliza el proceso inverso de conversión con el que fue leído, es decir se usan comandos de conversión similares previos a la lectura de la salida.

```
y=np.int16(audio_data)
```

Ahora la información en este loop en específico está listo para ser enviada a los speakers con el mismo formato con el que entraron el puerto de lectura. Así mismo se utiliza un proceso de conversión inverso.

```
stream.write(y)
```

Estas operaciones son constantes en cada loop y son las que provocan que la aplicación cumpla con las características con las cuales fue construida.

8.5 Acondicionamiento de Señal.

Fue necesario visualizar los dispositivos de entrada y salida y sus características, en particular su número de canal y su número index para especificarlos en los seteos de la lectura en el programa principal. Se creó un mini programa que imprimía los dispositivos de entrada y salida de audio. El siguiente código imprimirá los dispositivos que el IDE(interface de texto) lee.

```
import pyaudio
p = pyaudio.PyAudio()
for i in range(p.get_device_count()):
 dev = p.get_device_info_by_index(i)
print((i,dev['name'],dev['maxInputChannels'],dev['maxOutputChannels']
))
```

La salida de este programa es la siguiente:

```
(0, u'Hifi Berry Dac+', 2L, 2L)
(1, u'HDA Intel PCH: ALC3239 Analog (hw:1,0)', 2L, 0L)
(2, u'USB PnP Sound Device: Audio (hw:2,0)', 1L, 2L)
(3, u'hdmi', 0L, 8L)
(4, u'pulse', 32L, 32L)
(5, u'default', 32L, 32L)
```

El número index *2* de la tarjeta de audio USB será de importancia en el código ademas de su número de canales, el cual es en este caso 1L. El index 0 fue la entrada hifi berry Dac+ del raspberry pi.

CAPÍTULO 9: ERRORES DETECTADOS

9.1 Cambio de algoritmo en el procesamiento

El proyecto pretendía implementar las ecuaciones a diferencias traducidas a lenguaje de programación como se mencionó en secciones anteriores. Sin embargo, se optó por cambiar el algoritmo de programación y usar en cambio algunos comandos que tienen las funciones de procesamiento de señal pre diseñadas. Este cambio se lo ejecutó con el objetivo de optimizar el tiempo de procesamiento que se requería para obtener un efecto auditivo audible en tiempo real dado el poder de procesamiento limitado del dispositivo en uso (Raspberry Pi). Por ejemplo, para el filtro 1 la implementación de las ecuaciones a diferencias es:

```
if order==1:
    for m in range(0, length_audio_data):
                audio_output[m]=a*audio_output[m-1]+audio_data[m-1]-
a_cong*audio_data[m]
absolute[m]=np.absolute(audio_output[m])
audio_input[m]=audio_data[m]


y = np.double(scipy.signal.lfilter(bequa,aequa, audio_data))
```

9.2 Lectura de Datos

Se realizaron pruebas archivos de audio(formato wav) que contenían los mismos códigos de procesamiento que se estan desarrollando en este proyecto pero orientados a procesar un archivo de audio guardado en algún lugar de la computadora. Esto se realizó con el objetivo de tener el código de procedimiento asegurado antes de lanzarse a pruebas con micrófonos y señales análogas.

Se dedicó un tiempo considerable al seteo de los parámetros de lectura para el procesamiento digital de la señal. Estas dificultades iniciales tenían como relación, por

ejemplo, la frecuencia de muestreo mínima necesaria para optimizar la lectura y a su vez que no perturbe las capacidades de procesamiento del dispositivo. Se probaron diferentes tamaños de buffer para asegurar que el procesamiento sea lo suficientemente rápido y que se lea en tiempo real sin producir notches(pausas debido al procesamiento).

9.3 Underrun de ALSA: Principal problema

El principal problema en el proyecto fue el siguiente error que se imprimía aleatoriamente en cada loop de procesamiento en el programa:

```
ALSA lib pcm.c:7843:(snd_pcm_recover) underrun occurred
```

La característica de este error es que al imprimir este error en la consola, se corta la señal por ese loop determinado, el quiebre de la señal es audible y disturba un sonido continuo. También se caracteriza porque que no viene de errores en la programación de la aplicación, sino en los archivos de las librerías que controlan la escritura y lectura de las señales internas en la computadora, más adelante se detallará las librerías específicas.

Se realizó una búsqueda exhaustiva para encontrar una solución definitiva. Muchos de los sitios que hacían referencia al problema eran foros de programas de còdigo abierto, no se llegó a concluir una respuesta definitiva, algunos usuarios al parecer piensan que este error se dá debido a que el dispositivo de procesamiento no tiene la capacidad de procesar los datos a la misma velocidad que la escritura especificada, otros que se debe a un mal seteo de las caracterìsticas de lectura como por ejemplo la frecuencia de muestreo en la aplicación, otros que se debe modificar los archivos de la biblioteca ALSA para la escritura de datos y otros que piensan que estas librerías son muy sensibles a llamar funciones externas por lo que pueden

producirse delays en la ejecución del programa. Después de probar todas estas posibles soluciones se concluyó que fue un overcloking que se explicará más adelante en el detalle de los tiempos de retardo.

9.3.1 Interface PCM (Pulse Code Modulation, Modulación del código de pulso)

Es la librería interna que controla la lectura y escritura de las señales análogas, esta librería es interna y está escrita en lenguaje C para los sistemas basados en código abierto(Ubuntu, Raspberry Pi, etc). Esta librería es la responsables de manejar los convertidores análogo-digital digital-análogo, independientemente de las aplicaciones o programas que se usen para solicitar la reproducción o grabación de audio.

9.3.2 Tiempos de retardo que ocasionaron el underrun

Se desarrolló un programa que permite obtener tiempos de lectura y escritura en un tiempo en una determinada línea del código. Se usó estas líneas de código para analizar las posibles causas del error mencionado anteriormente. Se encontraron relaciones y posibles explicaciones a este problema. Sin embargo es necesario conocer las características del hardware de la entrada y salida de la aplicación para sumar esto valores a los cálculos. La siguiente información es del hardware solamente, es decir, sin contar el tiempo que la aplicación toma para procesar datos:

```
ENTRADA: USB PNP SOUND DEVICE
default sample rate: 44100 HZ
default low input latency:0.0116099
```

```
default high input latency: 0.046439909
max input channels: 1

SALIDA: HIFI BERRY DAC+
default sample rate: 44100 HZ
default low output latency: 0.0116099
default high output latency: 0.046439909
max output channels: 2
```

Posteriormente se vio las siguientes líneas de código para recolectar información relevante. El método *processing* es la encargada de realizar el loop infinito y filtrar la señal. En las siguientes líneas de código sólo se imprimen los tiempos de análisis de la señal sin ningún efecto pero con filtros de ecualización.

```
1. def processing():
2.     while True:
3.         print("-------------------------------------------------")
4.         start_stream= stream.get_time()
5.         start=time.clock()
6.         print "Lectura disponible antes de leer: " + str(
   stream.get_read_available())
7.
8.         #lectura=stream.get_read_available()
9.
10.              try:
11.                  string_audio_data = stream.read(CHUNK)
12.              except IOError as ex:
13.                  if ex[1] != pyaudio.paInputOverflowed:
14.                      raise
15.                  string_audio_data= '\x00' * CHUNK*2
16.                  print "Lectura disponible despues de leer: " + str(
   stream.get_read_available())+"\n"
      a. # for clean sound
17.              if flag==[False,False]:
18.                  audio_data = np.fromstring(string_audio_data,
   dtype=DTYPE)# of length 2048 and each element is numpy.int16
19.
20.                  audio_data=np.double(audio_data)
21.
22.                  highvar=float(var_high_level.get())
23.                  middlevar=float(var_middle_level.get())
24.                  lowvar=float(var_low_level.get())
25.                  highvar=10**(highvar/20)
26.                  middlevar=10**(middlevar/20)
27.                  lowvar=10**(lowvar/20)
28.
```

```
29.      #Aquí se definen los parámetros de ecualización
30.                  f1 = 150
31.                  df1 = 400
32.                  w1 = 2 * math.pi * f1 / RATE
33.                  dw1 = 2 * math.pi * df1 / RATE
34.                   b1, a1 = parmeq(1, lowvar, 1 / math.sqrt(2), w1,
   dw1)
35.
36.                  f2 = 300
37.                  df2 = 400
38.                  w2 = 2 * math.pi * f2 / RATE
39.                  dw2 = 2 * math.pi * df2 / RATE
40.                  b2, a2 = parmeq(1, middlevar, 1 / math.sqrt(2), w2,
   dw2)
41.
42.                  f3 = 700
43.                  df3 =600
44.                  w3 = 2 * math.pi * f3 / RATE
45.                  dw3 = 2 * math.pi * df3 / RATE
46.                  b3, a3 = parmeq(1, highvar, 1 / math.sqrt(2), w3,
   dw3)
47.
48.                  bequa = np.array(scipy.signal.convolve(b1,b2))
49.                  bequa = np.array(scipy.signal.convolve(bequa,b3))
50.                  aequa = np.array(scipy.signal.convolve(a1,a2))
51.                  aequa = np.array(scipy.signal.convolve(aequa,a3))
52.
53.                  y = np.double(scipy.signal.lfilter(bequa,aequa,
   audio_data))
54.
55.                  y=np.int16(y)
56.                  audio_data=np.int16(audio_data)
57.                  string_audio_data = y.tostring()
58.                  string_audio_data = audio_data.tostring()
59.
60.                  print "Escritura Antes de escribir: " + str(
   stream.get_write_available())
61.                  stream.write(string_audio_data)
62.                  end=time.clock()
63.                  ttime=end-start
64.                  print("tiempo de lectura y
   escritura"+str(ttime*1000)+"ms")
65.                  print "Escritura Despues de escribir: " + str(
   stream.get_write_available())+"\n"
66.
67.                  print "Latencia de la entrada: " + str(
   stream.get_input_latency())
68.                  print "Latencia de la salida: " + str(
   stream.get_output_latency())
69.                  end_stream= stream.get_time()
70.                  time_stream=end_stream-start_stream
71.                  print("tiempo stream:"+str(time_stream*1000)+"ms")
```

El código de la parte superior *imprime* tiempos relevantes para evaluar el desempeño de la lectura y escritura de un loop en particular. Se llama a comandos específicos de la librería *time* para poder calcular la diferencia de tiempos en una línea de código en particular y también se usa comandos de la librería PyAudio para evaluar datos disponibles para leer y para escribir en una línea de código y tiempo específicos. El problema underrun que se puede ver en la siguiente impresión se produce cuando el buffer que intenta leer un número de muestras excede las muestras disponibles para leerse (este problema tiene relación con el overcloking de los dispositivos de fábrica):

```
Lectura disponible antes de leer: 1860
Lectura disponible después de leer: 2068

^ALSA lib pcm.c:7339:(snd_pcm_recover) underrun occurred
#principal problema

Escritura Antes de escribir: 4096
tiempo de lectura y escritura: 20.0ms
Escritura Después de escribir: 2048

Latencia de la entrada: 0.0853333333333
Latencia de la salida: 0.128

tiempo stream:97.051ms
```

El tiempo de lectura y escritura es sólo un reloj entre una línea y otra, en cambio el tiempo de stream, según la documentación oficial de pyAudio da el tiempo que se demora el puerto en leer los datos y cambia constantemente. Se pudo ver que aun si este tiempo de stream que en este loop en particular fue 97.051ms es mayor que el tiempo de lectura predicho por los cálculos, se deben considerar los tiempos de latencia de entrada y salida, los cuales son constantes. Así por ejemplo, en este loop en particular los tiempos se calcularán de la siguiente manera(asumiendo una latencia alta de los dispositivos que se analizaron anteriormente).

Tiempo teórico de procesamiento: 46.439909ms(latencia alta de la entrada usb)+42.67ms(predicho por los cálculos con 16bits por muestra)+46.439909ms(latencia alta de la salida dac+)=135.55ms

Tiempo real de procesamiento de un loop en particular: 46.439909ms(latencia alta de la entrada usb)+97.051ms(tiempo real usado por el número de operaciones en el código)+46.439909ms(latencia alta de la salida dac+)=189.93ms

Como se puede ver, en este loop en particular el tiempo real de procesamiento (189.93ms) es mayor que el teórico (135.55ms) y también se lo puede comprobar en la impresión de los frames listos para ser leídos. Los frames disponibles antes de ser leídos en la línea 1 y 2 son menores al tamaño del buffer. El programa trata de leer un tamaño de buffer menor al disponible y ocasiona en la mayoría de los casos el error que se mencionó en secciones anteriores. Este error es aleatorio y depende de la potencia del procesador en un tiempo dado. Por este motivo se intentaron realizar varios métodos de optimización de código.

9.4 Calidad de sonido final (sección de errores)

Se realizaron varias pruebas de los archivos finales de la aplicación, se ajustaron algunos parámetros de la interfaz, como por ejemplo el orden máximo del efecto phaser. Esto se dio debido a que el raspberry pi empezaba a consumir mucho poder de procesamiento intentando calcular órdenes mayores debido a que el número de cálculos incrementa conforme se subía este parámetro. Aun así las pruebas en la computadora resultaron claras, la computadora es capaz de soportar esta carga(hasta un orden de 5), en

cambio el raspberry pi es capaz de soportar hasta un orden de 3(sin distorsionar la señal). Otro ejemplo es el efecto de distorsión, el cual lastimosamente fue demasiado pesado en términos de procesamiento para el raspberry pi, aun así la computadora es capaz de llevar esta carga. Se pretende posponer estos algoritmos del efecto de distorsión para futuro trabajo. Por último, después de todas las pruebas, tomando en cuenta los parámetros máximos que acepta cada dispositivo(la computadora y el raspberry pi), se concluyó por análisis subjetivo(oído de músico) que el raspberry pi, con la ayuda del módulo de audio Hifi Berry DAC+, tiene un sonido de calidad mucho mayor al de la computadora. Se piensa que es debido a que el conversor D/A del DAC+ tiene una mayor resolución que los speakers internos de la computadora.

El error underrun de la librería ALSA PROJECT que se mencionó anteriormente ocurrió de manera más continua en la computadora porque tenía menos memoria física que el raspberry en el momento (300MB), a diferencia del raspberry pi(más de 3.5GB). Aun así cuando se corre el programa en el raspberry pi y se desplaza el mouse para cambiar los valores en la interfaz o si se usa otro proceso que consuma memoria, el error tiende a ser continuo. Por ese motivo se aconseja que hasta que se optimice aún más el proyecto, se intente no mover el mouse en el entorno del sistema operativo o usar cualquier otra aplicación que consuma memoria, con el objetivo de tener un audio procesado continuo y son quiebres de señal.

9.5 Intentos de optimización

A pesar de usar varios métodos de optimización, como por ejemplo transportar la función que calcula los coeficientes de la función de transferencia del efecto phaser, de esta manera, no se calcularían continuamente estos valores. Aun así, no se consiguió un gran tiempo de optimización. Esto se da porque el buscar un valor que se no se altere en otra función también consume tiempo de procesamiento. También se intentó reducir el número de operaciones al máximo, pero aun así las operaciones necesarias seguían consumiendo tiempo de procesamiento. Como en la sección anterior se mencionó el diagrama de flujo de los datos, se intentó cambiar el formato de lectura de la librería pyAudio de paInt16 paFloat32 para manejar los datos precisos de manera directa, sin necesidad de cambiarlos a double precisión, pero con el costo de leer más bits desde un principio. Aun así se consiguió mucho mejoramiento. Se piensa que el único cambio significativo fue debido a tres optimizaciones:

-Se transportaron las variables globales sólo a donde necesitaban ser leídas de manera que el programa no tenga la necesidad de leer variables innecesarias en procesos externos.

-Se usó un comando time.sleep() de la librería de time que sirve para que el procesador descanse por un tiempo determinado en cada loop. Este es un intento forzado de descanso del procesador con librerías específicas de procesos complejos. Se usó un tiempo de 1 milisegundo después de cada iteración.

-Se definieron vectores de la función de transferencia determinados. Es decir, se declararon vectores fijos en lugar de calcular la función de transferencia en cada ciclo. De esta

manera, las barras de desplazamiento que controlan los argumentos de entrada servirían para escoger los valores en una tabla de vectores y filtrar posteriormente la señal con los vectores escogidos.

9.6 Compatibilidad de Módulos de Audio

Un problema que ocurrió en los últimos días del proyecto fue, que debido a que el sistema se perdió en algunas ocasiones y fue necesario reiniciar el sistema operativo Raspbian, en un momento dado, el módulo de audio Hifi Berry Dac+ no fue reconocida por el Raspberry Pi. Se intentaron instalar los drivers del módulo tal como se detalló en la sección anterior pero algunos de los drivers no fueron leídos correctamente por el sistema. Se intentaron varios métodos para resolver el problema, ya que sin el módulo Dac+ no habría una salida de audio. Al final se encontró el problema, el dispositivo si fue leído por el Raspberry Pi en fechas anteriores, lo que se concluyó es que la nueva versión del sistema Kernel del Raspberry, el cual es 4.0 en su versión más reciente, no reconocía los módulos del Hifi Berry Dac+, los cuales funcionan con versiones anteriores, como por ejemplo, linux 3.18.x. Al parecer la página de soporte del Hifi Berry Dac+ no ha actualizado este bug, por lo que la solución consistió en volver a una versión anterior del sistema kernel del raspberry. Para conseguir éste objetivo se utilizó los siguientes comandos:

-Para saber la versión actual el kernel que está instalado en el dispositivo, se realiza:

```
$ uname -a
```

-En el caso del presente proyecto, la versión era la más actual, linux raspberrypi 4.0, para cambiar de kernel, se ejecuta el siguiente comando:

```
$ sudo rpi-update <version>
```

-Para encontrar el código del sistema deseado, visitar la siguiente página: https://github.com/Hexxeh/rpi-firmware/commits/master

-En el caso del presente proyecto, por ejemplo, el comando fue:

```
4f95c894c6a96b265d98f04a889e469833f7e996
```

-Una vez terminado la actualización, reiniciar el sistema, comprobar los pasos que se describieron en la sección previa del módulo y el problema debería haberse solucionado.

9.7 Cambio de definición del filtro pasa todos.

En los últimos días del proyecto se llegó a un código cuya eficiencia era bastante aceptable. Sin embargo el mayor problema fue que la respuesta del filtro producía efectos secundarios en la señal. Después de un filtro de orden 3, la ecuación a diferencias incrementan los valores de la salida en gran proporción, hasta el punto de distorsionar la señal completamente. Esto se da porque las ecuaciones a diferencias dependen de los valores anteriores en cada iteración. Se optó por cambiar la definición de la función de transferencia del filtro de orden 1(filtro prototipo para los siguientes órdenes). Se omitió la parte imaginaria del numerador y denominador de la función de transferencias. Es decir, el nuevo filtro sólo contiene términos REALES. Esto se da en el filtro de orden 1 y en el resto de los órdenes ya que dependen del filtro de orden 1. Según la forma del filtro se predijo que el crecimiento de los valores de la señal en cada iteración incrementaría a menor ritmo y como consecuencia el filtro llegaría a operar a órdenes superiores a 3.

CAPÍTULO 10: CONCLUSIONES Y RECOMENDACIONES

10.1 Conclusiones

-El Raspberry pi es un poderoso microprocesador con capacidades computacionales similares a la de una laptop pequeña. Al estar basado en un sistema de código abierto como es linux, el raspberry pi tiene una amplia gama de aplicaciones. La facilidad con la que se comunica con el exterior, es decir la interface de entrada y salida, es suficiente para alcanzar proyectos ambiciosos. El hecho que tenga acceso a varios protocolos de comunicación, entradas USB, display y salidas de audio, lo convierte en una computadora de bolsillo y ayuda a entender a un nivel mucho más básico el funcionamiento de los dispositivos de la vanguardia tecnológica en términos de aplicaciones para la vida cotidiana, así como reproducción de audio, internet, telefonía móvil, etc. En resumen el Raspberry Pi, bajo la configuración y programación indicada puede transformarse en cualquier dispositivo tecnológico. Esta característica convierte al programador en un auténtico inventor de hardware y software y lo pone a la vanguardia de la tecnología mundial porque el código fuente pertenece a una comunidad global de programadores ansiosos de impulsar el ingenio humano y artístico.

-La comunicación entre dispositivos bajo protocolos específicos tales como dirección IP, protocolo SSH, protocolo VNC, etc permiten que los dispositivos puedan ser fácilmente internados en una red local de manejo, de manera que un usuario fácilmente puede acceder a varios sistemas de control en un solo puerto de comunicación, dándole ventaja en tiempo y herramientas para desarrollar proyectos de software y hardware mucho más avanzados.

-Raspbian basado en linux es un sistema operativo de código abierto, que se usó en el presente proyecto. Tiene una capacidad ilimitada de comunicación entre el usuario programador y la estructura interna de cualquier dispositivo que use este sistema operativo. Con esto en mente se puede decir que linux permite a los usuarios entender como los archivos que conforman una arquitectura de computadora interactúan los unos con los otros. Por ejemplo se actualizó el sistema operativo innumerables veces, se instalaron bibliotecas que modificaban el comportamiento del sistema, se modificó algunos archivos internos del dispositivo para poder operar el proyecto con los requerimientos necesarios, etc. Todas estas modificaciones del sistema solo se realizan en la terminal de linux que funciona con comandos shell, es decir con comandos de texto que usan sintaxis específicas para manipular el dispositivo. En resumen, linux es necesario para los usuarios que desean saber cómo funcionan las computadoras a un nivel de abstracción más grande, incluso al nivel de entender como el hardware y software se comunican en todos los dispositivos tecnológicos que usamos a diario.

-Python es un lenguaje de programación muy poderoso, al ser el lenguaje de preferencia de los sistemas basados en UNIX de código abierto permite fácilmente conectarse con los drivers, puertos externos, sensores, etc de los dispositivos basados en esta plataforma. El lenguaje, además de ser muy legible por los compiladores, permite al programador desarrollar algoritmos efectivos y precisos que lo obligan a pensar en desarrollar arquitecturas de código para facilidad de lectura y edición. Python posee una cantidad casi ilimitada de librería que se adecuan a los requerimientos del proyecto o aplicación a tal nivel que es difícil predecir si es uno de los lenguajes más básicos y a la vez más complejos en la actualidad.

-El procesamiento de audio digital se enfoca en una correcta programación de código fuente, en términos de procesamiento y de comunicación con el usuario(interfaz gráfica). Se deduce desde el presente proyecto que se pueden aplicar cientos de efectos diferentes que obedecen a diferentes algoritmos teóricos aplicando las líneas de código correctas que obedecen a estos algoritmos. Los archivos que contienen las aplicaciones de procesamiento de audio no son pesados, es decir el código va directo a los puertos. Sin embargo el uso de comandos de librerías específicas y el número de operaciones usadas para el procesamiento causan retardos por el overcloking (intento de usar procesos más rápido que la frecuencia del reloj de fábrica del dispositivo) que se produce.

-Con el presente proyecto se identifican los puntos importantes en la fabricación de productos procesadores de audio digital, por ejemplo se deduce que algoritmos similares que generan efectos de sonido característicos como por ejemplo la distorsión o el delay son empleados para fabricar pedales físicas de procesamiento para músicos y productores de la industria musical. Se concluye que si se dispusiera de mayor capital, se podría llegar al estado del arte de estas bases de construcción de software y hardware para comercializar productos de alta calidad de procesamiento de audio que pudieran competir con las empresas internacionales de la industria de la música.

-En procesamiento de audio en tiempo real, el tiempo de procesamiento que el dispositivo toma en cada iteración, de un tamaño de buffer determinado, no es constante. Es decir, el procesador no siempre funciona al 100% de su capacidad ni tampoco a un 50% de su capacidad. Esta conclusión es trivial pero fue la causa de muchos errores en la recuperación de

los puertos de lectura en la entrada. Se comprobó que incluso los servicios básicos del sistema operativo consumen memoria RAM que afecta la eficiencia de la aplicación causando overclokings. Para este tipo de problemas siempre es recomendable preparar el hardware de manera que se prevenga posibles calentamientos, como por ejemplo usar disipadores de calor y mantener el dispositivo en espacios frescos mientras se usa la aplicación.

-El Hifi Berry DAC+ es un convertidor digital-análogo de grandes capacidades. El presente proyecto usó una resolución de 16 bits, lo cual para el oído humano es bastante caro en términos de calidad de sonido. La calidad de sonido se pudo apreciar a priori, por el oído humano(oído de músico y productor). El mismo código se probó en la computadora y el raspberry pi con el módulo hifi berry dac+ y el sonido efectivamente era mucho más claro. Esta resolución es clave en la industria musical. En grandes conciertos donde la amplificación es de cientos de watts de potencia esta resolución puede determinar la calidad del sonido para la audiencia así también como en los dispositivos DSP para grabaciones en estudios.

10.2 Recomendaciones

-En los sistemas basados en UNIX hay que tener mucho cuidado con las actualizaciones que se requieren porque en algunos casos se necesita una versión del kernel específico. Por ejemplo el error que se detalló en la sección anterior de errores de programación tenía que ver con la compatibilidad del sistema operativo con los drivers con el módulo de audio Hifi Berry Dac+. Antes de realizar cualquier intento de prueba con aplicaciones es necesario documentar la compatibilidad de librerías con sistemas operativos y con drivers específicos.

-Muchos foros de errores de programación como por ejemplo stackoverflow arreglan muchas situaciones específicas de programación, sin embargo no se especifican las versiones del lenguaje, o del software compilador o el sistema operativo.

-Estar alerta a copiar y pegar cualquier código de corrección de errores en foros sin tener en cuenta la documentación oficial que aún si es extensa y lleva tiempo pero que brinda soluciones y explicación de los errores.

-En la programación, estar consciente del funcionamiento de las funciones prediseñadas por librerías específicas, así como también de los argumentos que aceptan como entrada, las variables que retornan de estas funciones y principalmente el tipo de dato que manejan. En el presente proyecto, llevo un tiempo considerable encontrar la falla de programación, la cuál era un manejo erróneo del tipo de conversión de dato que se usaba con librerías externas.

-En procesamiento de audio estar muy atento a la configuración de entradas y salidas de audio. En muchas ocasiones, porque no se configuraron las entradas de una manera adecuada se produjeron estallidos de audio con feedback a un alto volumen debido que las frecuencias son extremadamente altas. Por ejemplo, ocurrió que el los parlantes de la laptop, al tener contacto con el puerto de la entrada de micrófono causaban in feedback con frecuencias muy altas que perturbaban el oído.

-Tener cuidado con la alimentación de los dispositivos, asegurar los cables de conexión y prevenir posibles cortocircuitos por errores humanos. Por ejemplo se pensó que el raspberry pi

se había dañado en los días finales del proyecto. Se pensó que era debido a un problema de actualización del sistema operativo, luego se pensó que era un problema de la alimentación y hasta se pensó que la tarjeta se dañó por un cortocircuito producido por una descarga estática de los dedos al manipular la tarjeta. Al final se concluyó que era un problema del cable adaptador HDMI que se había dañado y no mostraba ninguna señal. Estar atento a que todo el sistema ese asegurado antes de proceder con las pruebas.

-Es recomendable configurar las computadoras que se comunican con el dispositivo base de cualquier proyecto. La comunicación remota fue de gran utilidad para casos de emergencia donde no se disponía de una pantalla para el display, un ratón y un teclado. Es más fácil usar una laptop a la mano y configurar su control remoto con el dispositivo que buscar todas estas herramientas para el manejo de las tarjetas. Además el control remoto permite identificar redes de comunicación más amplias y sirven de práctica para familiarizarse con los protocolos de comunicación en general.

-Usar breakpoints y códigos de información de tipo de dato y forma de dato. En diversas ocasiones se arrojaron errores de programación que se basaban en manejo erróneo del tipo de dato que se manejaba, o el intento de operar entre vectores y valores de diferente tipo de dato. En un proyecto como este, en el que se necesita puertos de entrada y salida para un procesamiento en tiempo real, es recomendable desarrollar códigos de información de los puertos de entrada y salida, es decir obtener información como la latencia de éstos dispositivo, los índices con el que el dispositivo los reconoce, el número de canales, el muestreo predeterminado. etc.

-Preparar cualquier tipo de hardware que maneje un procesamiento alto y continuo. Los disipadores previenen que el dispositivo se caliente y las aplicaciones que se desarrollan en este tengo un alto nivel de rendimiento. En el caso del raspberry pi fue necesario aumentar tres pequeños disipadores de calor para que la aplicación mejore su rendimiento.

BIBLIOGRAFÍA

Apple Documentation: "Phaser Effect". Recuperado el 15 de febrero de 2015, de https://documentation.apple.com/en/logicstudio/effects/index.html#chapter=9%26section=6%26tasks=true

Como dar una Ip estática a tu Raspberry Pi. Recuperado el 6 de Junio de 2015, de http://www.modmypi.com/blog/tutorial-how-to-give-your-raspberry-pi-a-static-ip-address

Digital Audio Effects. Recuperado el 20 de Julio de 2015, de http://www.cs.cf.ac.uk/Dave/CM0268/PDF/10_CM0268_Audio_FX.pdf

Documentación del HiFi Berry Dac+. Recuperado el 10 de Enero de 2015, de https://www.hifiberry.com/dac/

Documentación PortAudio. Recuperado el 10 de marzo de 2015, de http://www.portaudio.com/

Documentación Pyaudio. Recuperado el 6 de marzo de 2015, de https://people.csail.mit.edu/hubert/pyaudio/

Documentación Python. Recuperado el 20 de febrero de 2015, de https://docs.python.org

Edwards, C. (2013). *Not so humble Raspberry Pi gets big ideas*. Engineering & Technology (17509637), 8(3), 30-33.

Elenberg Ethan, Hsu Anthony, L´Heureux marc, Ng Stephanie, Parikh Alaap, Thiele E.J., Yu Michelle. "*Digital Filter Design for Audio Processing*". Research Gate. Retrieved from: www.researchgate.net/publictopics.PublicPostFileLoader.html?id...key…

Fruity Phaser. Recuperado el 10 de Junio de 2015,de https://www.image-line.com/support/FLHelp/html/plugins/Fruity%20Phaser.htm

Guitars. Recuperado el 35 de mayo de 2015, de http://recordingology.com/in-the-studio/guitars/

Heeks, R., & Robinson, A. (2013). *Ultra-Low-Cost Computing and Developing Countries*. Communications Of The ACM, *56*(8), 22-24. doi:10.1145/2492007.2492016

Holy City Sound. *All about all pass filtering.* Retrieved from: https://holycitysound.wordpress.com/2014/12/10/all-about-all-pass-filtering/

Ip estática wireless. Recuperado el 10 de Junio de 2015, de http://www.electroschematics.com/9496/static-manual-ip-wireless-raspberry-pi/

Llanos, Diego. *Teaching Embedded Operating Systems using Raspberry Pi and Virtual Machines*. Universidad de Valladolid, Spain, 2014.

Mitchell, G. G. (2012). *The Raspberry Pi single-board computer will revolutionize computer science teaching*. Engineering & Technology (17509637), 7(3), 26. doi:10.1049/et.2012.0300

Organización Raspbian. Recuperado el 18 de febrero de 2015, de https://www.raspbian.org/

Organización Raspbian. *Acceso Remoto*. Retrieved from: https://www.raspberrypi.org/documentation/remote-access/

Osgood Nathaniel, Hong June-chi and Richardson Jacqui(2008). *Signal Processing for the Electric Guitar(Published Bachelor Thesis).* Worcester Polytechnic Institute.

PC MAGAZINE. *Definition of embedded systems*. Recuperando en febrero 8 de 2015, de http://www.pcmag.com/encyclopedia/term/42554/embedded-system

Raspberry Pi Foundation. *Quick Start Guide*. Recuperado de http://www.raspberrypi.org/help/quick-start-guide/

Raspberry Pi Foundation. *What is a raspberry Pi?.* Published on 2014. Recuperado de http://www.raspberrypi.org/help/what-is-a-raspberry-pi/

Tech Stuff Frequency Ranges. Recuperado el 5 de marzo de 2015, de http://www.zytrax.com/tech/audio/audio.html

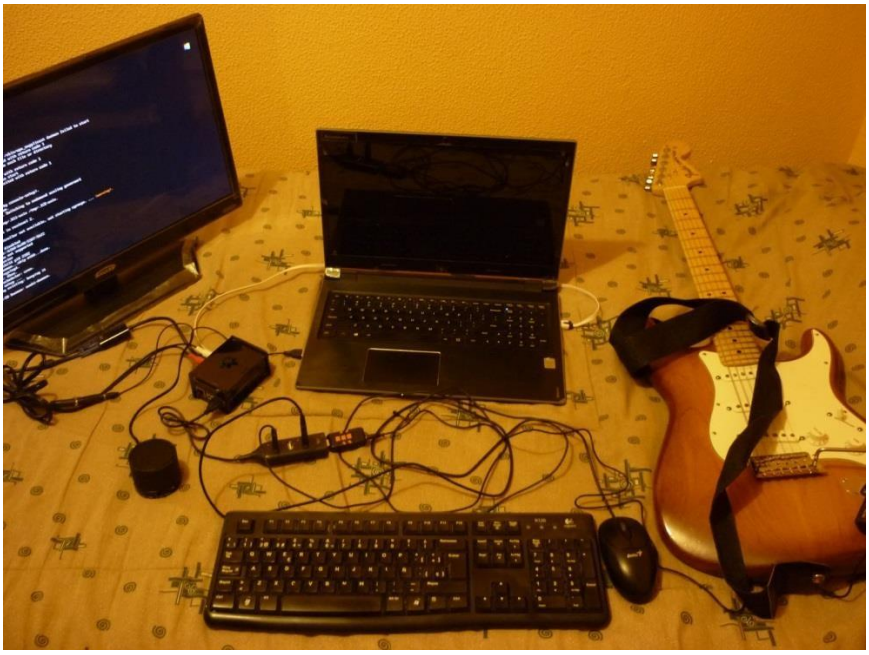
TestTone, Music Technology Reviews. *What is a Phaser Effect*. Retrieved from: http://testtone.com/fundamentals/what-phaser-effect

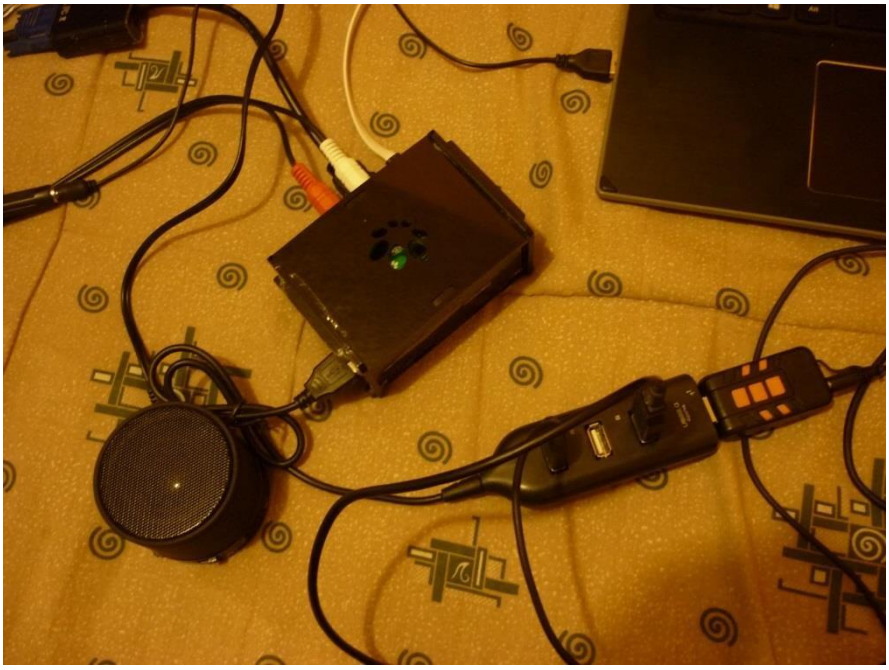
Thakur, Sofia. (2013). *The Olympic Cauldron, Raspberry Pi and Microsoft's Windows Phone 8 are battling to be named the Design of the Year in a competition run by a museum in south London*. Engineering and Technology Magazine.Engv

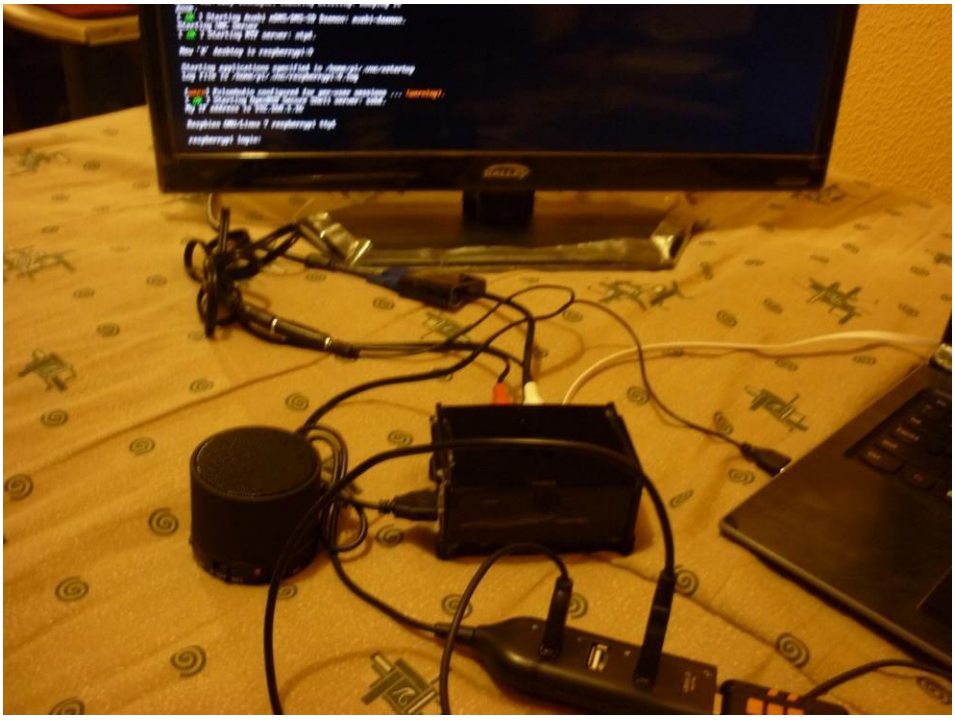
The technology of phase shifters and flangers. Recuperado el 5 de Junio de 2015, de http://www.geofex.com/Article_Folders/phasers/phase.html

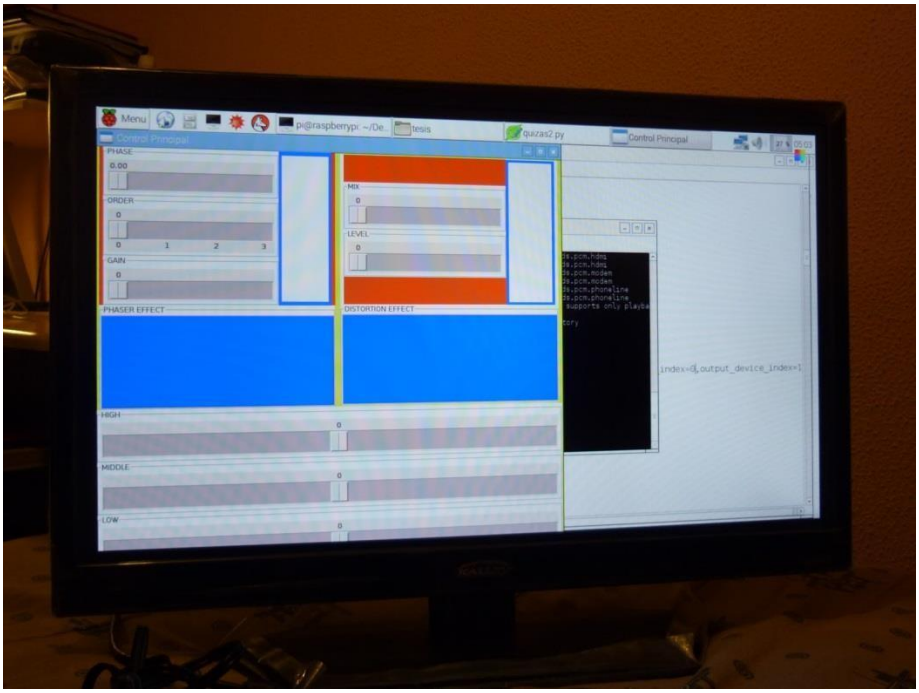
Zolzer Udo, Wiley John & Sons. *DAFX: Digital Audio Effects*. Ltd ISBNs: 0-471-49078-4 (Hardback); 047084604-6 (Electronic). Copyright q 2002

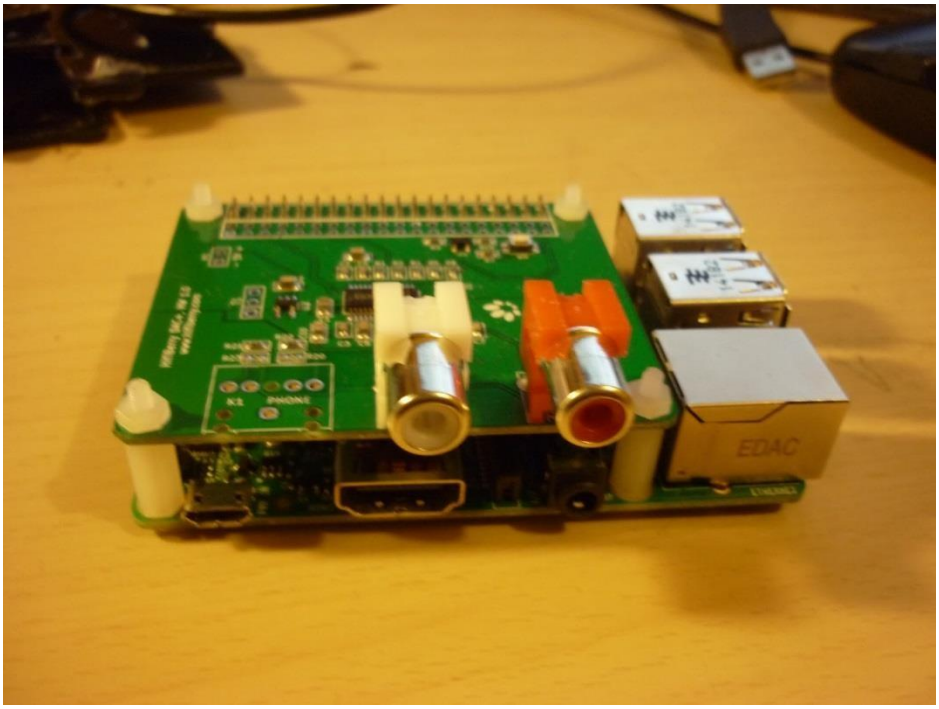
ANEXOS








Menu
Control Principal
PHASE
ORDER
GAIN
PHASER EFFECT
MIX
LEVEL
DISTORTION EFFECT
HIGH
MIDDLE
LOW


EDAC

ANEXOS DE CÓDIGO

```
function notas()
[samp,fs] = wavread('analisis_guitarra.wav');
figure(1) xf=fft(samp);
lf=ceil(length(xf)/2);
xf=xf(1:lf);
fplot=linspace(0,round(fs/2),lf);
semilogx(fplot,abs(xf))
xlabel('Frecuecia en base logaritmica')
ylabel('Escala normal')
axis([20 20000 0 15000])
grid on
figure(2)
semilogx(fplot,20*log10(abs(xf)))
xlabel('Frecuecia en base logaritmica')
ylabel('dB')
axis([20 20000 -50 100])
grid on

--------------------------------------------------------------------------
function [b, a, beta] = parmeq(G0, G, GB, w0, Dw)
beta = tan(Dw/2) * sqrt(abs(GB^2 - G0^2)) / sqrt(abs(G^2 - GB^2));
b = [(G0 + G*beta), -2*G0*cos(w0), (G0 - G*beta)] / (1+beta);
a = [1, -2*cos(w0)/(1+beta), (1-beta)/(1+beta)];

--------------------------------------------------------------------------
function effect(f1,f2,f3,df1,df2,df3,gdb1,gdb2,gdb3)
%effect(150,300,700,400,400,600,-5,5,10)
[samp,fs] = wavread('analisis_guitarra.wav');
%f1 = 150;
%df1 = 400;
w1 = 2 * pi * f1 / fs;
dw = 2 * pi * df1 / fs;
%GdB = -5;
G = 10 ^ (gdb1 / 20);[b1, a1] = parmeq(1, G, 1 / sqrt(2), w1, dw);
%Reduces muddieness (-.25)
%f2 = 300;
%df2 = 400;
w1 = 2 * pi * f2 / fs;
dw = 2 * pi * df2 / fs;
%GdB = 5;
G = 10 ^ (gdb2 / 20);
[b2, a2] = parmeq(1, G, 1 / sqrt(2), w1, dw); %Evens mid-frequencies (-
.6)
%f3 = 700;
%df3 = 600;
w1 = 2 * pi * f3 / fs;
dw = 2 * pi * df3 / fs;
```

```
%GdB = 10;
G = 10 ^ (gdb3 / 20);
[b3, a3] = parmeq(1, G, 1 / sqrt(2), w1, dw); %Raised the distortion in
the
%high frequencies.
%(2.2)
b = conv(b1,b2);
b = conv(b,b3)
a = conv(a1,a2);
a = conv(a,a3)
filt = filter(b,a,samp);
%len=length(filt)
%spectrum(filt,fs,len)
wavwrite(filt,fs,'distorsion2');
% mag response in db
im=[1 zeros(1,fs-1)];
yim=filter(b,a,im);
yimf=fft(yim);
lff=ceil(length(yimf)/2);
yimf=yimf(1:lff);
fplotf=linspace(0,round(fs/2),lff);
figure(1) subplot(3,1,1)
semilogx(fplotf,20*log10(abs(yimf)))
ylabel('dB')
axis([20 20000 -15 15])
grid on
% input signal xf=fft(samp);
lf=ceil(length(xf)/2);
xf=xf(1:lf);
fplot=linspace(0,round(fs/2),lf);
subplot(3,1,2)
semilogx(fplot,abs(xf))
axis([20 20000 0 15000])
grid on
%output signal yf=fft(filt);
ylf=ceil(length(yf)/2);
yf=yf(1:ylf);
fploty=linspace(0,round(fs/2),ylf);
subplot(3,1,3)
semilogx(fploty,abs(yf))
axis([20 20000 0 15000])
grid on
--------------------------------------------------------------------------
function [note,F]=guitarsimulation(string)
%string=6;
Fs       = 44100;
A        = 110; % The A string of a guitar is normally tuned to 110 Hz
Eoffset  = -5;
```

```
Doffset  = 5;
Goffset  = 10;
Boffset  = 14;
E2offset = 19;
E=82;
A=110;
D=147;
G=196;
B=247;
e=330;
if string==6
st=E;
end
if string==5
st=A;
end
if string==4
st=D;
end
if string==3
st=G;
end
if string==2
st=B;
end
if string==1
st=e;
end
F = linspace(1/Fs, 1000, 2^12);
length(F)
x = zeros(Fs*4, 1);
delay = round(Fs/st);
b  = firls(42, [0 1/delay 2/delay 1], [0 0 1 1]);
a  = [1 zeros(1, delay) -0.5 -0.5];
[H,W] = freqz(b, a, F, Fs);
figure(1)
plot(W, 20*log10(abs(H)));
title('Harmonics of an open A string');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;
zi = rand(max(length(b),length(a))-1,1);
note = filter(b, a, x, zi);
l_W=length(W)
l_note=length(note)
note = note-mean(note);
note = note/max(abs(note));
hplayer = audioplayer(note, Fs);
play(hplayer);
```

```
function filterresp(n,m,string);
%INPUT3: STRING SIMULATION-----------------------------------------------
fs       = 44100;
% The A string of a guitar is normally tuned to 110 Hz

%f = linspace(1/fs, 1000, 2^12);
f = linspace(0, 2*pi);
length(f)
%construction of the filter
a=exp(j*pi/m);
bf=[-a 1]
af=[1 -a']
if n>=2
   afinal=af;
   bfinal=bf; for
i=1:n-1
bfinal=conv(bfinal,bf)
afinal=conv(afinal,af)
end
else
    bfinal= bf
  afinal=af

end
%SELECCION DE METODOS
%PRUEBA CON LA FUNCION FILTER
[H1,W1] = freqz(bfinal, afinal, f, fs);
[H2,W2] = freqz(bfinal, afinal, 1024,'whole',fs);
%figura de la respuesta del filtro
figure(1)
plot(W1, 20*log10(abs(H1)));
title('Frecuency');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on
figure(2)
zplane(bfinal, afinal)
figure(3)
freqz(bfinal,afinal)
figure(4)
plot(W1/pi,20*log10(abs(H1)))
xlabel('Normalized Frequency (\times\pi rad/sample)')
ylabel('Magnitude (dB)')

-------------------------------------------------------------------------
function pruebas()
fo=0.3;
spamf=[-50:1:50]/1000;
f=fo+spamf;
x=zeros(1,1000);
t=0:999;
```

```
for k=1:101
   x=x+cos(2*pi*f(k)*t);
end
Xf=fft(x);
[x,y1,y]=rene_comp(1,x,4);
Yf=fft(y1);
figure(1)
plot(x)
title('Coseno incial con un aumento de variacion en magnitd y frecuencia
a los lados')
grid on
fs=1/(t(2)-t(1));
df=fs/length(t);
fplot=-fs/2:df:fs/2-df;
figure(2) subplot(1,2,1)
plot(fplot,fftshift(abs(Xf)))
title('Respuesta en Magnitud de la FFT de la entrada')
grid on subplot(1,2,2)
plot(fplot,fftshift(abs(Yf)))
title('Respuesta en Magnitud de la FFT de la salida')
grid on
figure(3)
subplot(1,2,1)
plot(fplot,fftshift(angle(Xf)))
title('Respuesta en fase de la FFT de la entrada')
grid on subplot(1,2,2)
plot(fplot,fftshift(angle(Yf)))
title('Respuesta en fase de la FFT de la salida')
grid on
```