Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Ingeniería en Sistemas Automotrices

Proyecto Integrador

“Convertidor audio-táctil y audio-visual para asistencia a la conducción de personas con discapacidad auditiva”

Desarrollado por los alumnos:

Juárez Pérez Aracely

López Cinta Javier

Asesores:

Dra. Briseño Tepepa Blanca Rosa

Ciudad de México, a 12 de junio del 2022

INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

UPIITA

CONVERTIDOR AUDIO-TÁCTIL Y AUDIO-VISUAL PARA ASISTENCIA A LA CONDUCCIÓN DE PERSONAS CON DISCAPACIDAD AUDITIVA.

QUE PARA OBTENER EL TÍTULO DE

"INGENIERO EN SISTEMAS AUTOMOTRICES"

PRESENTAN LOS ALUMNOS:

Aracely Juárez Pérez

Javier López Cinta

ASESOR:

p.a. Adolfo Rafael

ASESOR

Dra. Blanca Rosa Briseño Tepepa

SINODALES:

PRESIDENTE

M. en C. Adolfo Rafael Agoración Calderón

SECRETARIO

M. en C. Arodi Rafael Carvallo Domínguez

p.a. Adolfo Rafael

VOCAL

Dra. Blanca Rosa Briseño Tepepa

Instituto Politécnico Nacional
DEPARTAMENTO DE FORMACIÓN INTEGRAL E INSTITUCIONAL
SEP
UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

CIUDAD DE MÉXICO A 12 DE 06 DEL 2022.

upiita·ipn

Autorización de uso de obra

Instituto Politécnico Nacional

P r e s e n t e

Bajo protesta de decir verdad el que suscribe Aracely Juárez Pérez (se anexa copia simple de identificación oficial), manifiesto ser autor (a) y titular de los derechos morales y patrimoniales de la obra titulada Convertidor audio-táctil y audio-visual para asistencia a la conducción de personas con discapacidad auditiva., en adelante "La Tesis" y de la cual se adjunta copia, por lo que por medio del presente y con fundamento en el artículo 27 fracción II, inciso b) de la Ley Federal del Derecho de Autor, otorgo a el Instituto Politécnico Nacional, en adelante El IPN, autorización no exclusiva para comunicar y exhibir públicamente total o parcialmente en medios digitales, Plataforma de la Dirección de Bibliotecas del IPN y/o consulta directa en la Coordinación de Biblioteca de la UPIITA "La Tesis" por un periodo de 5 años contado a partir de la fecha de la presente autorización, dicho periodo se renovará automáticamente en caso de no dar aviso expreso a "El IPN" de su terminación.

En virtud de lo anterior, "El IPN" deberá reconocer en todo momento mi calidad de autor de "La Tesis". Adicionalmente, y en mi calidad de autor y titular de los derechos morales y patrimoniales de "La Tesis", manifiesto que la misma es original y que la presente autorización no contraviene ninguna otorgada por el suscrito respecto de "La Tesis", por lo que deslindo de toda responsabilidad a El IPN en caso de que el contenido de "La Tesis" o la autorización concedida afecte o viole derechos autorales, industriales, secretos industriales, convenios o contratos de confidencialidad o en general cualquier derecho de propiedad intelectual de terceros y asumo las consecuencias legales y económicas de cualquier demanda o reclamación que puedan derivarse del caso.

Ciudad de México, a 12 de Junio de 2022

Atentamente

upiita·ipn

Autorización de uso de obra

Instituto Politécnico Nacional
P r e s e n t e

Bajo protesta de decir verdad el que suscribe Javier López Cinta.
(se anexa copia simple de identificación oficial), manifiesto ser autor (a) y titular de los derechos morales y patrimoniales de la obra titulada Convertidor audio-táctil y audio-visual para asistencia a la conducción de personas con discapacidad auditiva.
_____, en adelante "La Tesis" y de la cual se adjunta copia, por lo que por medio del presente y con fundamento en el artículo 27 fracción II, inciso b) de la Ley Federal del Derecho de Autor, otorgo a el Instituto Politécnico Nacional, en adelante El IPN, autorización no exclusiva para comunicar y exhibir públicamente total o parcialmente en medios digitales, Plataforma de la Dirección de Bibliotecas del IPN y/o consulta directa en la Coordinación de Biblioteca de la UPIITA "La Tesis" por un periodo de 5 años contado a partir de la fecha de la presente autorización, dicho periodo se renovará automáticamente en caso de no dar aviso expreso a "El IPN" de su terminación.

En virtud de lo anterior, "El IPN" deberá reconocer en todo momento mi calidad de autor de "La Tesis". Adicionalmente, y en mi calidad de autor y titular de los derechos morales y patrimoniales de "La Tesis", manifiesto que la misma es original y que la presente autorización no contraviene ninguna otorgada por el suscrito respecto de "La Tesis", por lo que deslindo de toda responsabilidad a El IPN en caso de que el contenido de "La Tesis" o la autorización concedida afecte o viole derechos autorales, industriales, secretos industriales, convenios o contratos de confidencialidad o en general cualquier derecho de propiedad intelectual de terceros y asumo las consecuencias legales y económicas de cualquier demanda o reclamación que puedan derivarse del caso.

Ciudad de México, a 12 de Junio de 2022

Atentamente

_____

AGRADECIMIENTOS Y DEDICATORIAS

Agradecemos profundamente a nuestras familias, quienes nos han brindado todo el apoyo necesario para poder sobrellevar los retos presentados en este tiempo de preparación profesional, sin quienes el poder culminar no hubiera sido posible.

De la misma forma a nuestra Institución, Unidades y profesores que marcaron la diferencia -en cuanto aprendizaje se refiere- durante los periodos académicos cursados.

A aquellas personas que indirectamente alentaban el corazón para continuar, amigos hechos familia durante esa temporada.

Y por, sobre todo, agradecemos a Dios del quién provienen todas las cosas y quien preservó nuestra salud, la de nuestros familiares y amigos, y quien además suplió de todos los recursos para que llegásemos a la meta.

Dedicamos este proyecto terminal a nuestros padres, hermanos y familia en general, por quienes el corazón se inspira y motiva cada día. A aquellos que creyeron en nosotros y aún en los momentos de mayor afrenta decidieron caminar a nuestro lado.

TABLA DE CONTENIDO

RESUMEN

Resumen del documento

Hoy en día el desarrollo tecnológico está vinculado a la vida de cada persona, reducen las distancias entre la gente y contribuyen a que los procesos empresariales sean más efectivos. En esta ocasión nos permitirá ampliar las oportunidades de una persona con discapacidad auditiva.

Es cierto que el oído tiene un valor importante al momento de conducir un automóvil, pero presentar discapacidad auditiva no significa que no sea posible hacerlo.

La conducción es una actividad que permite tener movilidad, independencia y autonomía. Las personas con discapacidad auditiva buscan y desean esas posibilidades, sobre todo, igualdad de oportunidades y una mejor calidad de vida.

Conducir es, principalmente, una actividad visual, más que auditiva. Sin embargo, hay estudios que demuestran que las personas sordas desarrollan mejor visión periférica o aumento del campo de visión, y esto, sí que podría suponer una ventaja para conducir. Por ello, en el presente proyecto se desarrolla el diseño de un prototipo de asistencia al conductor con discapacidad auditiva, permitiéndole a través de la captación de los sonidos de importancia al conducir, la conversión de tales señales acústicas a vibraciones y señales visuales. De esta forma, el mismo conductor podrá estar alerta de sirenas de emergencia y claxon, como parte de los sonidos de mayor relevancia fuera del vehículo y poder así tomar las decisiones correctas al ir al volante.

Con lo anterior, se amplía así la libertad de movilidad a todos los conductores, de manera particular a aquellos que presentan discapacidad auditiva. Este proyecto trasciende a lo meramente tecnológico, ya que está impulsado por la falta de oportunidades que presentan las personas con este tipo de discapacidad, por lo cual, se ha diseñado para asistir de forma complementaria a este grupo selecto de conductores, tomando en cuenta que una persona con discapacidad auditiva agudiza su habilidad para percibir señales visuales y táctiles.

La señal obtenida a través de un instrumento receptor de sonido y su debida conversión permitirán que el conductor sea capaz de distinguir eventos exteriores que requieran su atención, particularmente sirenas de emergencia y claxon. A través de respuestas vibratorias y señales luminosas podrán percibirse las señales de alerta deseadas.

Es posible diversificar la aplicación del presente proyecto, derivado de la amplia necesidad de las personas con discapacidad auditiva, esto en sectores primordialmente de salud, pero además siendo posible la adecuación de este para algún área distinta; modulando los valores de frecuencia deseados y el entrenamiento del algoritmo dispuesto.

La conjunción completa del prototipo permite realizar el filtrado del sonido al rango de frecuencias con las que operan las sirenas y bocinas en el sector automotriz, además de la correcta selección mediante el entrenamiento de un algoritmo de inteligencia artificial. La correcta implementación de los códigos y algoritmos de programación hacen posible el cumplimiento de las necesidades globales planteadas.

Derivado de la actual situación sanitaria, se presenta el prototipo como una propuesta meramente demostrativa, al no ser posible el montaje en un automóvil funcional. Se utiliza como señales fuente, audios pregrabados que simulan la puesta en marcha de un vehículo en movilidad urbana.

Palabras clave: Sonido, audio, frecuencia, filtro, Inteligencia Artificial, algoritmo, red neuronal.

Abstract

Nowadays technological development is linked to the life of each person, they reduce the distances between people and contribute to making business processes more effective. This time it will allow us to expand the opportunities of a person with hearing impairment.

It is true that hearing has an important value in driving, but being hearing impaired does not mean that it is not possible to do so. Driving is an activity that allows mobility, independence and autonomy. People with this disability seek and want these possibilities, above all, equal opportunities and a better quality of life. Driving is primarily a visual activity, rather than an auditory one. However, there are studies that show that deaf people develop better peripheral vision or increased field of vision, and this, it could be an advantage for driving. For this reason, in this project the design of a prototype of assistance to the driver with this type of disability is developed, allowing the conversion of such acoustic signals to vibrations and visual signals, through the capture of sounds of importance when driving. With this, he will be able to be alert for emergency sirens and horns, as part of the most relevant sounds outside the vehicle and thus be able to make the correct decisions when driving.

With the foregoing, the freedom of mobility is thus extended to all drivers, particularly those with hearing disabilities. This project transcends the merely technological, since it is driven by the lack of opportunities that people with this type of disability present, for which reason, it has been designed to assist this select group of drivers in a complementary way, taking into account that a person with a hearing impairment sharpens his ability to perceive visual and tactile signals.

The signal obtained through a sound receiving instrument and its proper conversion will allow the driver to be able to distinguish exterior events that require his attention, particularly emergency sirens and horns. Through vibrational responses and light signals, the desired warning signals can be perceived.

It is possible to diversify the application of this project, derived from the wide need of people with hearing disabilities, this in primarily health sectors, but also being possible to adapt it to a different area; modulating the desired frequency values and training the arranged algorithm.

The complete conjunction of the prototype allows filtering the sound to the range of frequencies with which sirens and horns operate in the automotive sector, as well as the correct selection through the training of an artificial intelligence algorithm. The correct implementation of the programming codes and algorithms make it possible to meet the global needs raised.

Derived from the current health situation, the prototype is presented as a merely demonstrative proposal, as it is not possible to mount it in a functional car. It is used as source signals, pre-recorded audios that simulate the start-up of a vehicle in urban mobility.

Keywords: *Sound, audio, frequency, filter, Machine Learning, algorithm, neural network.*

.

OBJETIVOS

Objetivo General

Implementar un prototipo de asistencia para personas con discapacidad auditiva, que provea la conversión de las señales acústicas esenciales del exterior a respuestas visuales y táctiles, proporcionándole seguridad al conducir un automóvil y a los pasajeros, mediante el filtrado, selección y conversión de las frecuencias deseadas.

Objetivos Particulares

- Seleccionar los sensores necesarios para la correcta captación del sonido.
- Seleccionar los componentes electrónicos debidos para el tratamiento acústico de las señales de entrada.
- Diseñar el circuito eléctrico conveniente para la implementación del prototipo completo y el modo de alimentación de este.
- Elegir las Interfaces de Desarrollo (IDE) adecuados con los debidos componentes electrónicos programables.
- Programar el algoritmo necesario para optimizar la respuesta del prototipo frente a las señales que se reciben.
- Seleccionar el modo más viable para adaptar las señales de salida a alguna zona estratégica para el conductor.
- Montar el sistema completo a algún vehículo.
- Verificar el funcionamiento individual y general del sistema.

1. INTRODUCCIÓN

Para la Organización Mundial de la Salud (OMS), unos 34 millones de niños y unos 432 millones de adultos en el mundo padecen discapacidad auditiva. Entre los adultos, la franja de edad más afectada es la de los mayores de 65 años, estimando con ello que en el 2050 una de cada diez personas sufrirá pérdida auditiva. Una de las causas más frecuentes de la discapacidad auditiva es la hipoacusia, la cual es definida médicamente como la incapacidad total o parcial para escuchar sonidos en uno o ambos oídos.

Se considera que alguien tiene discapacidad auditiva cuando se produce una pérdida auditiva superior a 40dB en el oído con mayor audición en adultos y superior a 30dB en niños.

Para la Dirección General de Tráfico (DGT), como unidad regulatoria en España y quienes mantienen altos estándares de seguridad vial en la Unión Europea, mencionan que una persona con discapacidad auditiva puede conducir turismos o motocicletas siempre y cuando no tenga más del 45% de pérdida de audición entre los dos oídos, lleve o no audífonos. En el caso de tener que necesitar permisos profesionales las normas son más estrictas y la pérdida de audición combinada entre los dos oídos no debe superar el 35%. En los dos casos tampoco puede haber patologías permanentes o intensas que alteren el equilibrio, como vértigo, mareos, vahídos o inestabilidad.

Además, cuando se presenta un déficit sensorial, es obligatorio contar con un mecanismo compensatorio basado en potenciar la capacidad de otras áreas sensoriales, tales como la visión. En el caso de la hipoacusia es obligatoria la utilización de espejos retrovisores exteriores y uno panorámico en el interior del coche. De esta manera se aumenta el campo visual del conductor.

Con ello, es importante recalcar, que el sentido del oído permite al conductor recibir la información necesaria relacionada con la conducción del vehículo. Esta información puede venir desde el exterior del vehículo (pitidos, sirenas de vehículos de emergencias, etc.) como desde el interior (funcionamiento del motor, indicaciones de otros pasajeros, la radio, etc.). Por lo que, una pérdida significativa de la audición puede afectar al rendimiento y la seguridad en la conducción.

Poner todos los sentidos en el momento de la conducción es primordial. El 90% de la información que recibimos al conducir nos llega a través de la vista, sin embargo, una buena salud auditiva también es básica para circular con seguridad. Sin ella no podríamos reaccionar ante la rápida aproximación de vehículos prioritarios, los sistemas de aviso del coche (como por ejemplo el que nos alerta de que no llevamos el cinturón de seguridad) o, simplemente, a los ruidos que emite el vehículo cuando algo no funciona correctamente.

Por ello, el propósito de este proyecto es brindar a las personas con discapacidad auditiva (ya sea en un nivel parcial o total) una conducción más segura de su vehículo, con la asistencia de un desarrollo tecnológico que emite señales visuales y táctiles complementarias que proporcionen confianza al conductor y a sus acompañantes. Dependiendo de la frecuencia propia de cada sonido importante para la conducción (claxon, ambulancia, etc.), la respuesta visual y táctil será proporcional a su propia intensidad.

Al conocer la realidad y la necesidad imperante que tienen las personas con discapacidad auditiva y los aspectos fundamentales de este padecimiento, daremos paso de manera subsecuente al modo de operación del sonido, como este actúa en diferentes campos y la manera en la que este puede ser captado. Es determinante entender la forma en que las ondas de sonido se expanden en el aire para generar diferentes frecuencias y de tal forma poder conocer la frecuencia propia de cada sonido de nuestro interés. Al estar el automóvil comúnmente en un ambiente con diferentes fuentes de sonido, es importante seleccionar un buen instrumento de recepción de sonido, micrófono en este caso, el cual se evalúa y compara para así cumplir con los requerimientos fundamentales para una correcta retención del sonido.

Al haber recibido la señal de audio, será posible hacer un filtrado que nos permita rechazar la mayor parte posible del ruido y frecuencias que no estén dentro del rango de interés, esto mediante programación realizada en Matlab. Así, las frecuencias resultantes del filtrado serán puestas a prueba en una red neuronal previamente entrenada para seleccionar únicamente aquellas frecuencias que sean propias de los sonidos que nos interesan, considerando aquellos efectos físicos presentes en el proceso. Esto, para que finalmente sea posible la conversión de las frecuencias a voltaje y así hacer el señalamiento debido de manera visual a través de elementos luminosos y táctiles por vibración de un motor; lo anterior en base al aumento o disminución de las frecuencias.

2. ANTECEDENTES

2.1 Estado del Arte

TABLA 1. Trabajos recientes relacionados con la conversión de señales acústicas

No	Nombre	Características	País	Instituto / Compañía	Tipo
1	Sistema Acústico Óptico de Sustitución Sensorial	*Instrumentos:* gafas con micrófonos y un visor especial. *Funcionalidad:* recoge la información acústica del ambiente y la transforma en una representación visual.	España	Universidad de La Laguna	Tesis
2	Audio-Visual Conversion (ACV) y Audio-Tactile Conversion (ATC)	*Instrumentos:* sensores, micrófonos y cámaras. *Funcionalidad:* traduce los sonidos producidos en el exterior, mostrando los efectos visuales en el salpicadero y los vibratorios en el volante.	Corea del Sur	Hyundai Motor Company	Proyecto Empresarial
3	Adquisición de Señales Acústicas y de Vibración para el Diagnóstico de Fallos	*Instrumentos:* micrófono, preamplificador. *Funcionalidad:* detección de sonidos y vibraciones particulares en fallas de algunos elementos mecánicos.	Ecuador	Universidad Politécnica Salesiana	Trabajo de Titulación
4	Convertidor de Señales Acústicas en Señales Luminosas Moduladas	*Instrumentos:* micrófono, diodos controlados y regulador de señales. *Funcionalidad:* convierte señales acústicas en señales luminosas de intensidad modulada en función de la intensidad y frecuencia del sonido.	España	Competencia Privada	Proyecto Industrial-Comercial
5	Aparato para el Control de Emisión Sonora	*Instrumentos:* transductores, convertidor ADC, microprocesador, altavoces.	España	Competencia Privada	Proyecto Industrial-Comercial

2.2 Fundamentos Teóricos

FUENTE DE ALIMENTACIÓN

La función de una fuente de alimentación es convertir la tensión alterna en una tensión continua y lo más estable posible, para ello se usan los siguientes componentes (Fig.1): Transformador de entrada; Rectificador a diodos; Filtro para el rizado; Regulador (o estabilizador) lineal, aunque este último no es imprescindible.

FIGURA 1. Principales componentes de una fuente de alimentación

En todo sistema de potencia vamos a necesitar una alta eficiencia la cual casi nunca va a llegar al 100% en la mayoría de los convertidores **DC/DC,** esto se debe a diversos factores, aunque normalmente encontramos un rango de 75% a 95% para la mayoría de los convertidores.

Lo anterior es posible gracias al uso de circuitos de modo conmutado, o chopper, cuyos elementos disipan una potencia insignificante por su atinada configuración (Fig.2). La modulación de ancho de pulso (**PWM**) permite el control y la regulación del voltaje de salida total. Este enfoque también se emplea en aplicaciones que implican alternar corriente, incluidos los convertidores de potencia de CC-CA de alta eficiencia (inversores y amplificadores de potencia), los convertidores de potencia de CA-CA y algunos convertidores de potencia de CA-CC (rectificadores de bajo armónico).

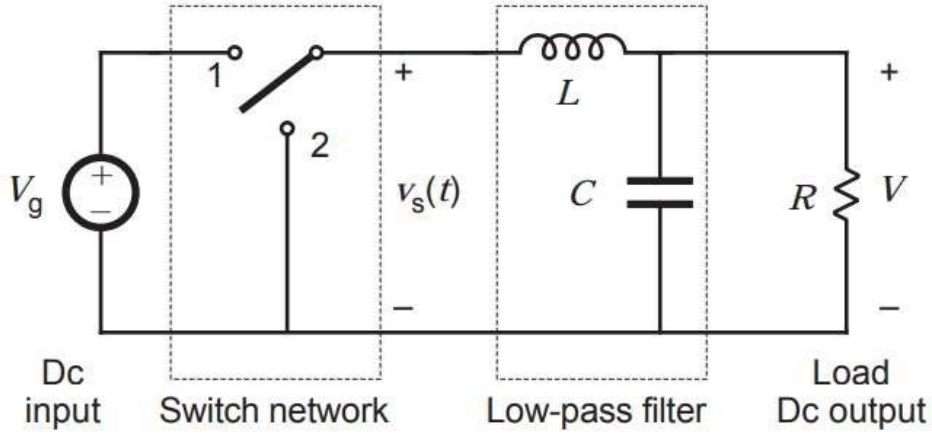


FIGURA 2. Ejemplo de Convertidor Buck

Un interruptor de un solo polo de doble tiro (SPDT) está conectado a la tensión de entrada DC Vg como se muestra. El voltaje de salida del interruptor vs (t) es igual a Vg cuando el interruptor está en la posición 1, y es igual a cero cuando el interruptor está en la posición 2 (Fig.3). La posición del interruptor varía periódicamente, de modo que vs (t) es una forma de onda rectangular que tiene un período Ts y un ciclo de trabajo D. El ciclo de trabajo es igual a la fracción de tiempo que el interruptor está conectado en la posición 1 y, por lo tanto, $0 \leq D \leq 1$. La frecuencia de conmutación fs es igual a 1 / Ts. En la práctica, el interruptor SPDT se realiza utilizando dispositivos semiconductores como diodos, MOSFET de potencia, IGBT, BJT o tiristores. Las frecuencias de conmutación típicas se encuentran en el rango de 1 kHz a 2 MHz, dependiendo de la velocidad de los dispositivos semiconductores.

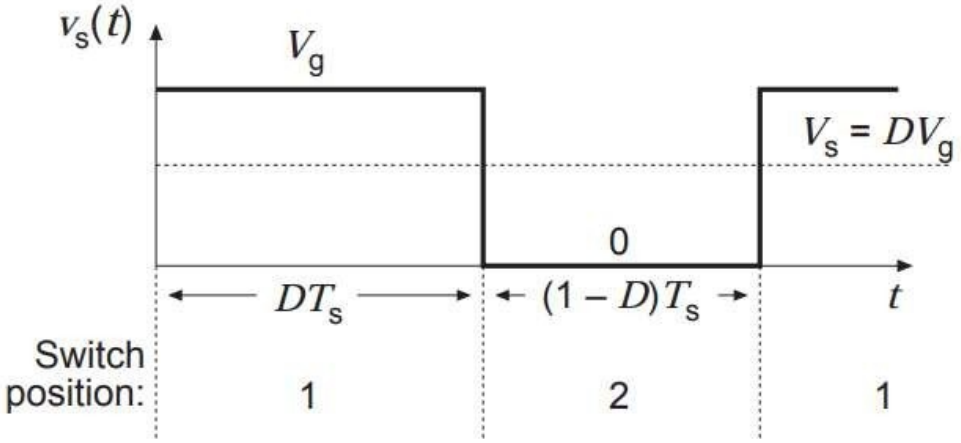


FIGURA 3. Forma de onda del voltaje del Switch

La red del conmutador cambia el componente DC del voltaje. Según el análisis de Fourier, el componente DC de una forma de onda viene dado por su valor promedio. El valor promedio de vs (t) viene dado por:

$$V_s = \frac{1}{T_s}\int_0^{T_s} v_s(t)dt = DV_g$$

La integral es igual al área bajo la forma de onda, o la altura Vg multiplicada por el tiempo DTs. Se puede ver que la red de conmutadores reduce el componente DC del voltaje en un factor igual al ciclo de trabajo D. Desde $0 \leq D \leq 1$, el componente DC de Vs es menor o igual a Vg.

De este modo, el **convertidor Buck** o reductor convierte un voltaje de entrada más alto en un voltaje de salida estabilizado más bajo.

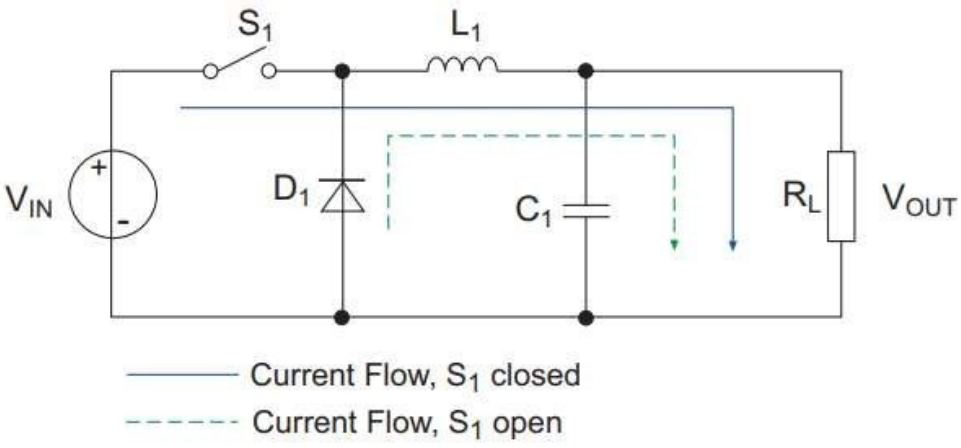


FIGURA 4. Flujo de corriente de acuerdo con el estado del Switch

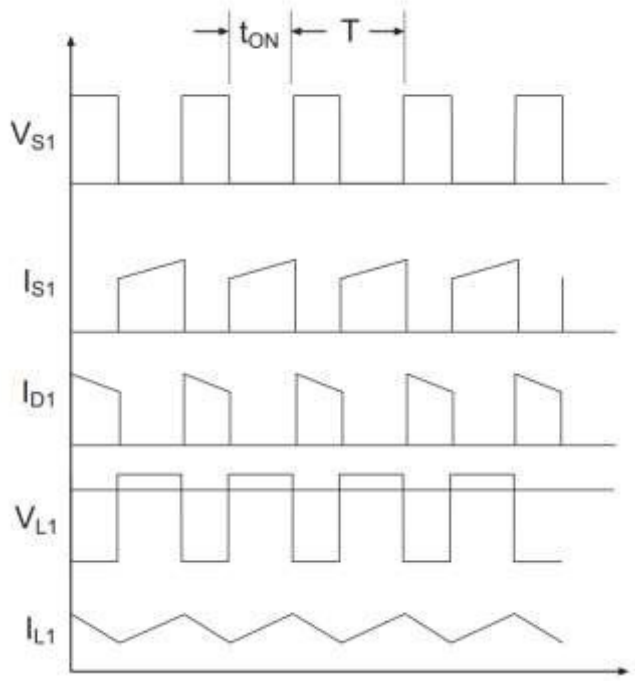


FIGURA 5. Principales formas de onda de corriente y voltaje en un convertidor Buck

Un regulador es un dispositivo o mecanismo que puede regular la potencia de salida constantemente. Existen diferentes tipos de reguladores disponibles en el dominio de la fuente de alimentación. Pero principalmente, en el caso de la **conversión de CC a CC**, hay dos tipos de reguladores disponibles: Lineal o Conmutación.

Un regulador lineal regula la salida usando una caída de tensión resistiva y debido a esto los reguladores lineales proporcionan una menor eficiencia y pierden potencia en forma de calor.

Por otro lado, el regulador de conmutación utiliza inductor, diodo y un interruptor de alimentación para transferir energía desde su fuente a la salida.

Por ello, el regulador Buck figura como un **convertidor conmutado de potencia** DC-DC muy simple que produce un voltaje de salida menor que su entrada, donde el inductor siempre actúa contra el voltaje de entrada. La tensión de salida de un regulador Buck ideal es igual al producto del ciclo de trabajo de conmutación y la tensión de alimentación.

La entrada está directamente conectada a través del interruptor. El inductor y el condensador se conectan a través de la salida, por lo que la carga obtiene una forma de onda de corriente de salida uniforme. El diodo se usa para bloquear el flujo de corriente negativa.

Si suponemos que el interruptor ha estado en posición abierta durante un tiempo prolongado, la corriente en el circuito es 0 y no hay voltaje presente.

En esta situación, si el interruptor se cierra, la corriente aumentará y el inductor creará un voltaje a través de él. Esta caída de voltaje minimiza la tensión de la fuente en la salida, después de unos momentos la velocidad del cambio de corriente disminuye y el voltaje a través del inductor también disminuye lo que eventualmente aumenta el voltaje a través de la carga. El inductor almacena energía usando su campo magnético.

Entonces, cuando el interruptor está encendido, a través del inductor, el voltaje es:

$$VL = V_{in} - V_{out}$$

La corriente en el inductor aumenta a una velocidad de:

$$(V_{in} - V_{out})/L$$

La corriente a través del inductor se eleva linealmente con el tiempo. La tasa de aumento de la corriente lineal es proporcional a la tensión de entrada menos la tensión de salida dividida por la inductancia:

$$\frac{di}{dt} = (V_{in} - V_{out})/L$$

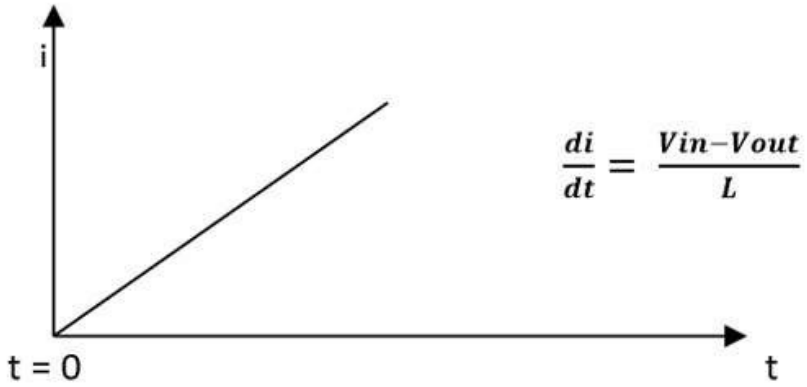


FIGURA 6. Fase de carga del inductor

El gráfico superior muestra la fase de carga del inductor. El eje "x" denota el tiempo y el eje "y" la corriente a través del inductor. La corriente aumenta linealmente con el tiempo cuando el interruptor está cerrado o ENCENDIDO.

Ahora, si el interruptor se abre durante un tiempo mientras la corriente todavía está cambiando, habrá una caída de voltaje en el inductor. El voltaje a través de la carga será menor que el voltaje de entrada. Durante el estado apagado, mientras el interruptor está abierto, la fuente de voltaje de entrada se desconecta y el inductor transferirá la energía almacenada a la carga. El inductor se convertirá en la fuente de corriente para la carga.

MICRÓFONO

Para la correcta selección de un dispositivo de entrada de sonido es necesario conocer sus parámetros técnicos, ya que de estos dependerá la eficiente de su respuesta.

Aunque por la situación actual no fue posible conseguirlo, aunado a su alto coste. Al poder utilizarlo en alguna ocasión futura podrá elevar los estándares de optimización del sistema. Por el momento nos brinda las características técnicas que debemos considerar como base para esta sección del prototipo, y la respuesta en frecuencia que presenta.

De esta manera, cuando se trata de mediciones de sonido en la industria automotriz o aeronáutica, no solamente estamos midiendo el ruido, estamos midiendo la calidad de la

experiencia del usuario final. Y regularmente nos enfrentamos a diversos retos como diversas fuentes de sonido o frecuencias de distracción. Para este tipo de mediciones es necesario un micrófono de multicampo para identificar diferentes fuentes de sonido de manera exacta con un solo micrófono de medición. El micrófono 4961 de Brüel & Kjaer presenta las características más elevadas para un desempeño óptimo. Este micrófono supera a todos los micrófonos de ¼'' en la medición de frecuencias, brindándote una mayor exactitud.

Es el micrófono ideal para mediciones de alta exactitud en campos de sonido libre, difusos o impredecibles. Este micrófono es el único micrófono de ¼'' con un ruido de fondo de 20 dB (A) y una sensibilidad de 60 mV/Pa con un SPL máximo de 130 dB. Que equivale al mismo desempeño que puedes obtener de un micrófono convencional de ½ ".

Gracias a su tamaño pequeño, el micrófono Tipo 4961 posee una respuesta uniforme, incluso en altas frecuencias. Lo que permite usarlo no solo para medir sonido en cabinas de la industria automotriz o aeronáutica sino también en cualquier condición de campo de sonido.

FIGURA 7. Comparación de tamaño del micrófono tipo 4961

Este micrófono es menos sensible al ángulo de incidencia que un micrófono convencional de ½''. Así que, al montarlo podrá ponerse en donde se requiere hacer la medición sin preocupación de donde ubicarlo.

La construcción 100% titanio del micrófono y del preamplificador asegura una máxima resistencia a la corrosión, por lo que no habrá agujeros en el diafragma del micrófono, un problema común en los micrófonos de níquel.

Además, el titanio evita que el micrófono sea afectado en gran medida por campos magnéticos, que son elevados, por ejemplo, del alternador del vehículo.

Cada micrófono tipo 4961 viene con su propia curva de calibración, incluyendo información sobre la sensibilidad a circuito abierto, respuestas en frecuencia en condiciones tanto de campo libre como en campo difuso y el error máximo de medición en cualquier campo de sonido (ver **Anexo I**).

DATOS DE AUDIO EN MATLAB

La señal de audio de un archivo representa una serie de *muestras* que captan la amplitud del sonido a lo largo del tiempo. La *tasa de muestreo* es el número de muestras discretas tomadas por segundo e indicadas en hercios. La precisión de las muestras, medida por la *profundidad de bits* (número de bits por muestra), depende del hardware de audio disponible.

Las funciones de audio de MATLAB leen y almacenan datos de audio de un canal (mono) en un vector columna *m* por 1 y los datos en estéreo en una matriz *m* por 2. En ambos casos, *m* es el número de muestras. Para los datos en estéreo, la primera columna contiene el canal izquierdo y la segunda columna, el canal derecho.

Habitualmente, cada muestra es un valor de doble precisión entre -1 y 1. En algunos casos, en particular cuando el hardware de audio no es compatible con profundidades de bits altas, los archivos de audio almacenan los valores como números enteros de 8 o 16 bits. El rango de los valores de las muestras depende del número de bits disponibles. Por ejemplo, las muestras almacenadas como valores *uint8* pueden oscilar entre 0 y 255 ($2^8 - 1$). Las funciones *sound* y *soundsc* de MATLAB solo son compatibles con valores de precisión simple o doble entre -1 y 1. La función *audioread* es compatible con formatos de archivo de acuerdo a la plataforma en uso (Tabla 2).

TABLA 2. Compatibilidad de plataformas con formatos de archivo

Compatibilidad de las plataformas	Formatos de archivo
Todas las plataformas	WAV (.wav)
	OGG (.ogg)
	FLAC (.flac)
	AU (.au)
	AIFF (.aiff, .aif)
Windows 7 (o posterior), Macintosh y Linux	MP3 (.mp3)
	MPEG-4 AAC (.m4a, .mp4)

FORMATO WAV

El formato WAV es uno de los mejores formatos de archivos de audio debido a su calidad de sonido sin pérdida. El límite superior del archivo WAV es de hasta 22 KHz, que algunas personas con oídos entrenados pueden detectar. Esto puede ser natural con muchas personas para escuchar frecuencias de alrededor de 22 KHz.

Los archivos WAV contienen más detalles y puede obtener un sonido más preciso con él.

En comparación con un archivo MP3, WAV es un formato sin comprimir o sin pérdidas, mientras que MP3 está comprimido o con pérdidas. Técnicamente **.wav** es solo un formato contenedor y puede contener varios tipos de audio comprimido o sin comprimir, pero normalmente verá que contiene audio LPCM sin comprimir (lo mismo que en los CD de audio). Con los archivos **.wav**, esencialmente obtiene una representación de flujo de bits sin formato de la señal de audio en forma digital. Un sonido analógico producido en el mundo real contiene esencialmente una cantidad infinita de información porque es una onda que cambia constantemente. Para llevar estos sonidos al dominio digital, necesita muestrear la señal a varios intervalos para hacer una aproximación del sonido. Para **.wav**, la señal de audio generalmente se muestrea a 44.100 veces por segundo o más, y cada valor muestreado se graba para que la onda de sonido se pueda reproducir.

ONDAS DE SONIDO

La transmisión de las ondas de sonido es compleja y presenta multitud de particularidades, en el punto de recepción puede reducirse a dos elementos simples: presión y tiempo. Estos pueden usarse para definir de manera absoluta cualquier sonido. Aun así, para entender el sonido más en profundidad, se separa en sus componentes, que son:

- Frecuencia, o su inversa, amplitud de onda. La frecuencia es el número de veces que se repite un evento por unidad de tiempo, medida en Hertzios. La longitud de onda es la distancia en la onda donde su forma se repite.

$$f = \frac{v}{\lambda}$$

- Amplitud, presión de sonido o intensidad. La amplitud es la medida de la variación de una variable periódica en un único período. Puede medirse amplitud de pico a pico, de pico, semi-amplitud o amplitud RMS.

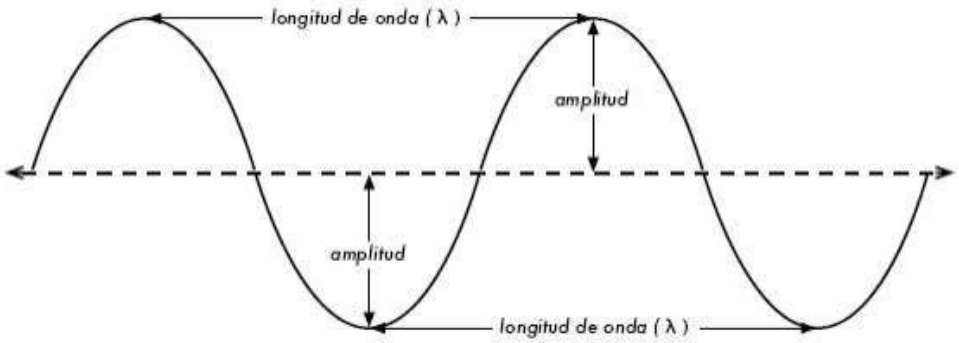


FIGURA 8. Amplitud y longitud de una onda

- Velocidad del sonido.

$$c = \sqrt{\frac{k}{\rho}}$$

Donde k es el módulo de compresibilidad (resistencia a la compresión uniforme) ρ y es la densidad.

- Dirección.

El sonido es perceptible para los seres humano en frecuencias que van desde los 20 Hz hasta los 20 kHz. En el aire, eso corresponde a longitudes de onda en el rango de 17m a 17mm.

El espectro de un sonido muestra las diferentes frecuencias que componen el mismo. La mayoría de los sonidos están compuestos de una compleja mezcla de vibraciones, y el espectro de sonido representa la vibración de cada frecuencia individual. Se presenta normalmente como un gráfico de intensidad o presión de sonido en función de la frecuencia.

INTENSIDAD Y PRESIÓN DE SONIDO

La intensidad de sonido, también conocida como intensidad acústica, es definida como la energía que transportan las ondas de sonido por unidad de área. Las unidades del sistema internacional para la intensidad, incluyendo intensidad acústica, es el vatio por metro cuadrado (W/m^2). Muchas medidas de intensidad acústica se hacen de forma relativa al umbral estándar de intensidad audible I_0:

$$I_0 = 10^{-12} vatios/m^2 = 10^{-16} vatios/cm^2$$

Generalmente se usa la escala en decibelios (dB) para medir la intensidad acústica. Los decibelios miden la relación entre una intensidad dada y el umbral de intensidad audible, de manera que este umbral toma el valor de 0 dB:

$$I\ (dB) = 10 \log_{10} [I\ / I_0]$$

Matemáticamente, la intensidad de sonido se define como:

$$I = p\nu$$

Donde:

- p es la presión de sonido, es decir, el cambio en la presión atmosférica causado por una onda de sonido.
- v es la velocidad de la partícula en el medio.

Tanto la intensidad acústica como la velocidad de la partícula en el medio son vectores, lo que significa que tienen dirección y sentido además de magnitud. La dirección de la intensidad es la

media de la dirección en la que fluye la energía. La intensidad media de un sonido en un período T es:

$$I = \frac{1}{T}\int_0^T p\ (t)v(t)dt$$

O también:

$$I = 2\pi^{\,2}n^2A^2\rho v$$

Donde:

- n es la frecuencia del sonido.
- A es la amplitud de la onda de sonido.
- v es la velocidad del sonido.
- p es la densidad del medio donde el sonido viaja.

TRANSFORMADA DE FOURIER

Toda función periódica puede reducirse de sinusoides, sin que importe el grado de complejidad que presenten, que es lo que establece el teorema de Fourier:

> *Toda función periódica de período P puede descomponerse en una suma de sinusoides armónicas, de amplitudes y fases adecuadas, cuyo primer armónico posea período P.*

Así como este teorema permite descomponer y analizar cualquier función periódica, también habilita la posibilidad de construir señales periódicas complejas a partir de una suma de sinusoides puras. Se puede entonces, reescribir el teorema de la siguiente forma:

> Toda función periódica de período P puede construirse a partir de una suma de sinusoides cuyas frecuencias formen una serie armónica fundamental$f = 1/P$. Cada sinusoide debe poseer la adecuada amplitud y fase, que se determinará a partir de la función periódica a estudiar.

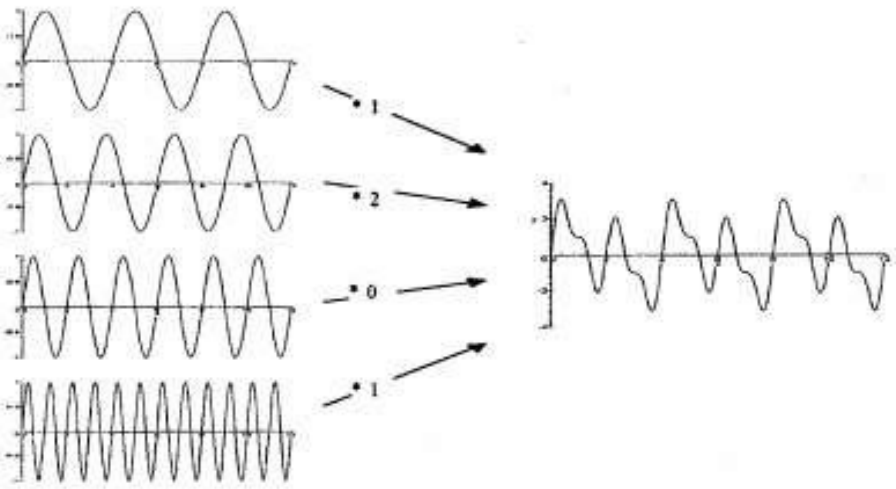


FIGURA 9. Una señal genérica formada por un sumatorio de señales sinusoidales

La disposición de las amplitudes, frecuencias y fases de las sinusoides involucradas en la suma se denomina *espectro de Fourier.*

Para comenzar el teorema establece que una función periódica posee un espectro armónico. Si tenemos una señal periódica y conocemos su período sabemos inmediatamente:

1. Que el espectro correspondiente será armónico.
2. Cuáles serán las frecuencias de cada uno de los armónicos.

Criterio de Nyquist

El criterio de Nyquist puede decirnos si el sistema es estable o inestable al determinar cuántos polos del sistema a lazo cerrado se encuentran en el semiplano derecho:

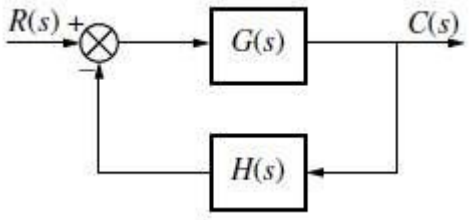


FIGURA 10. Sistema de Control en lazo cerrado

MUESTREO DE UNA SEÑAL

El proceso de **muestreo** se utiliza principalmente para señales de audio y señales de vídeo que son de naturaleza continua, es decir, definidos en todos los instantes de tiempo. Muestrear una señal continua en el tiempo da una señal que sólo tendrá valores en instantes de tiempo en concreto (valores discretos de la señal). El número de muestras por segundo se conoce como *frecuencia de muestreo* (Fs) la cual viene condicionada por el ancho de banda de la señal analógica (Teorema de Nyquist). La frecuencia de muestreo tiene una incidencia directa sobre la cantidad de recursos necesarios en el almacenaje (espacio en disco) y la transmisión (ancho de banda). El no cumplimiento del Teorema de Nyquist introduce un error llamado *aliasing* (un encabalgamiento de la señal provoca que en la reconstrucción no obtengamos la señal original). El concepto de muestreo se puede entender como una multiplicación de la señal original con un tren de deltas (caso ideal) o un tren de pulsos (caso real).

Caso ideal

La onda que muestrea en el ideal es:

$m(t) = \sum_{n=-\infty}^{\infty} \delta(t - nTs)$; donde Ts es el período de muestreo ($Ts = \frac{1}{Fs}$)

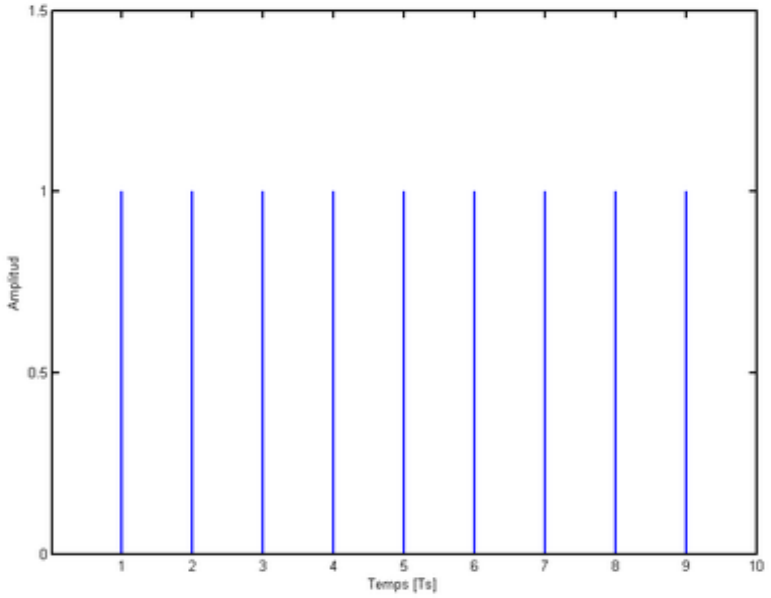


FIGURA 11. Tren de Deltas

Cuando a la entrada se tiene una señal f(t) continua en el tiempo con un ancho de banda finito (W), a la salida obtenemos s(t) = m(t) * f(t):

$$s(t) = \sum_{n=-\infty}^{\infty} f(t)\delta(t - nTs)$$

La transformada de Fourier de la señal de salida es la siguiente:

$$S(f) = \frac{1}{Ts} \sum_{n=-\infty}^{\infty} F(f - nFs)$$

Caso real

Dado que un tren de deltas no es implementable, el modelo para el muestreo utilizado es un tren de pulsos. Su amplitud es 1 durante un intervalo muy pequeño para conseguir un muestreo lo más ideal posible. Por este mismo motivo no podremos recuperar la señal original exacta, debido a que la Transformada de Fourier de un pulso es una función sinc. La multiplicación de una función sinc con el espectro de la señal original dará como resultado el espectro de la señal original ligeramente modificado y repetido cada Fs.

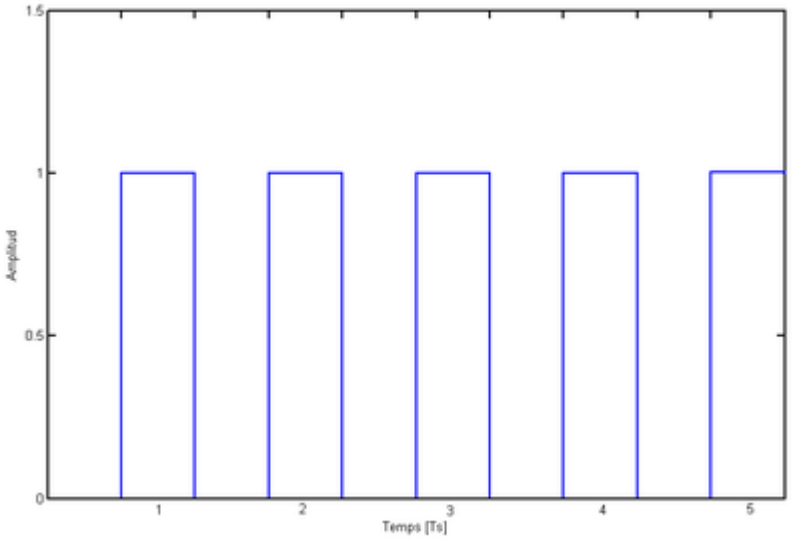


FIGURA 12. Tren de Pulsos

Limitaciones del Sistema Auditivo Humano

El sistema auditivo tiene dos limitaciones:

1. El umbral de audición oscila entre 20 Hz y 20 kHz. Por tanto, todas las frecuencias que están por debajo de 20 Hz o por encima de 20 kHz son inaudibles por el oído humano.

2. Puede haber enmascaramiento frecuencial y/o enmascaramiento temporal.

FILTRO BUTTERWORTH

El filtro de Butterworth es uno de los filtros electrónicos más básicos, diseñado para producir la respuesta más plana que sea posible hasta la frecuencia de corte. En otras palabras, la salida se mantiene constante casi hasta la frecuencia de corte, luego disminuye a razón de 20n dB por década (ó ~6n dB por octava), donde n es el número de polos del filtro.

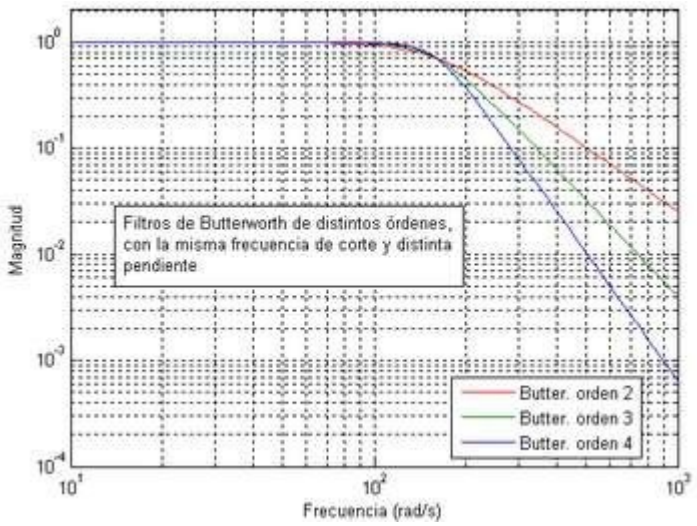


FIGURA 13. Respuesta en frecuencia de un filtro Butterworth de diferentes órdenes

La respuesta en frecuencia de un filtro Butterworth es muy plana (no posee ondulaciones) en la banda pasante, y se aproxima del cero en la banda rechazada. Cuando visto en un gráfico logarítmico, esta respuesta desciende linealmente hasta el infinito negativo. Para un filtro de

primera orden, la respuesta varía en −6 dB por octava (−20 dB por década). (Todos los filtros de primera orden, independientemente de sus nombres, son idénticos y poseen la misma respuesta en frecuencia.) Para un filtro Butterworth de segunda orden, la respuesta en frecuencia varía en −12 dB por octava, en un filtro de tercera orden la variación es de −18 dB, y así por delante. Los filtros Butterworth poseen una caída en su magnitud como una función lineal con ω.

El Butterworth es el único filtro que mantiene el mismo formato para órdenes más elevadas (sin embargo, con una inclinación más íngreme en la banda atenuada) mientras otras variedades de filtros (Bessel, Chevyshev, elíptico) poseen formatos diferentes para órdenes más elevadas.

Comparado con un filtro Chevyshev del Tipo I/Tipo II o con un filtro elíptico, el filtro Butterworth posee una caída relativamente más lenta, y por lo tanto irá a requerir una orden mayor para implementar uno especificación de banda rechazada particular.

Sin embargo, el filtro Butterworth presentará una respuesta en fase más lineal en la banda pasante del que los filtros Chebyshev del Tipo I/Tipo II o elípticos.

INTELIGENCIA ARTIFICIAL

El término “Inteligencia Artificial” fue utilizado por primera vez en el año 1956 por John McCarthy. La inteligencia artificial es la disciplina que se ocupa de la construcción de programas que permite a las computadoras imitar tareas que son características de la inteligencia humana, como la toma de decisiones, el procesamiento de textos, la percepción, el reconocimiento de objetos y sonidos, etc.

El cerebro humano es una máquina fascinante. Viendo todas las capacidades que tiene el ser humano, es posible hacerse una idea de todos los campos.

Machine Learning

El aprendizaje automático es un subcampo de la inteligencia artificial que permite a las máquinas mejorar en una terea determinada, a base de experiencia. Es importante saber que las técnicas de aprendizaje automático son clasificadas como de Inteligencia Artificial pero no toda Inteligencia Artificial puede ser clasificada como Aprendizaje Automático.

El informático Arthur Lee Samuel define el aprendizaje automático como el campo de estudio que provee a las computadoras la habilidad de aprender sin haber sido explícitamente programadas para ello, es decir, engloba todas las técnicas y algoritmos que permiten que los sistemas adquieran su propio conocimiento mediante la extracción de patrones de datos no procesados. Todo esto permite que las máquinas puedan utilizarse para resolver problemas que requieran conocimiento del mundo real.

Deep Learning

El aprendizaje profundo es un campo especializado del Machine Learning que se basa en el entrenamiento de redes neuronales artificiales (ANNs) que utilizan un gran conjunto de datos, como imágenes o textos. Las RNA son modelos de procesamiento de información inspirados en el cerebro humano. El cerebro humano consta de miles de millones de neuronas que se comunican entre sí mediante señales eléctricas y químicas que permiten a los humanos poder ver, sentir y tomar decisiones. El Deep Learning se basa en el entendimiento del mundo como una jerarquía de conceptos, permitiendo a las máquinas aprender conceptos complejos en base a otros más simples con los que tengan relación. A diferencia de las técnicas de aprendizaje automático, el aprendizaje profundo tiene la capacidad de extraer características automáticamente.

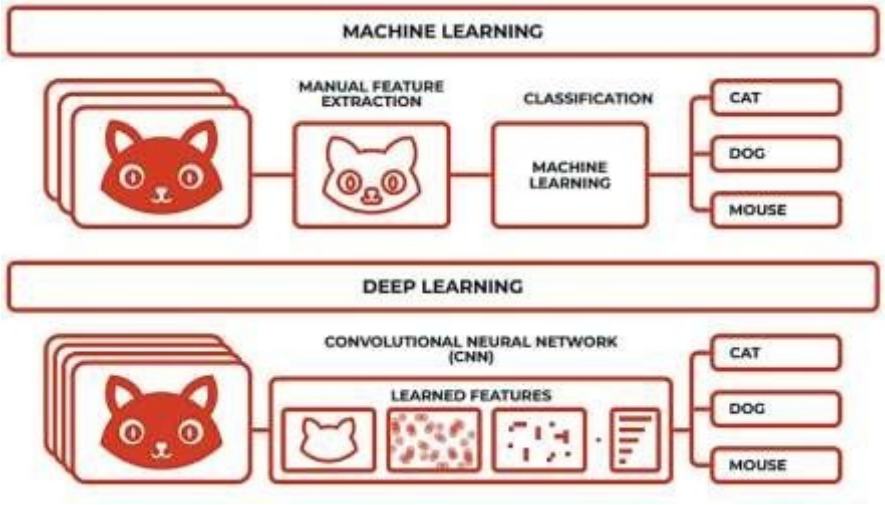


FIGURA 14. Diferencia entre Machine Learning y Deep Learning

RED NEURONAL

En general, las redes neuronales no son más que un tipo de sistema de aprendizaje supervisado para dotar a una máquina con la capacidad de aprender a realizar una tarea. Una red neuronal está compuesta por multitud de elementos simples (neuronas), interconectados de una forma más o menos densa y cuyo funcionamiento en conjunto puede dar lugar a un procesamiento no lineal complejo. Las redes son capaces de adaptarse y ajustar su funcionamiento a partir de los datos y experiencia que se les proporcione.

Una ANN (por sus siglas en inglés) consiste en un conjunto de neuronas separadas en capas y comunicadas entre ellas mediante conexiones.

Conexiones entre neuronas

Las conexiones entre neuronas son equivalentes a los axones en una neurona biológica, en una ANN, estos se definen por los pesos. Estos mismos pueden tratarse de enlaces excitadores o inhibidores en función de si el peso es mayor o menor que cero

Se necesita una, la **capa de entrada** información proveniente de fuentes externas es proporcionada a la red a través de esta capa, la cual se encarga de recibir los datos del modelo.

Cada neurona recibe uno de esos valores iniciales, y ésta lo transforma y lo reenvía a todas las neuronas con las que esté conectada en adelante, es decir, con las **capas ocultas o intermedias**, las cuales ajustarán los pesos para adaptarse a dichos datos. Así mismo, cuenta con una **capa de salida**, la cual es responsable de dar los resultados finales de la red hacia el exterior.

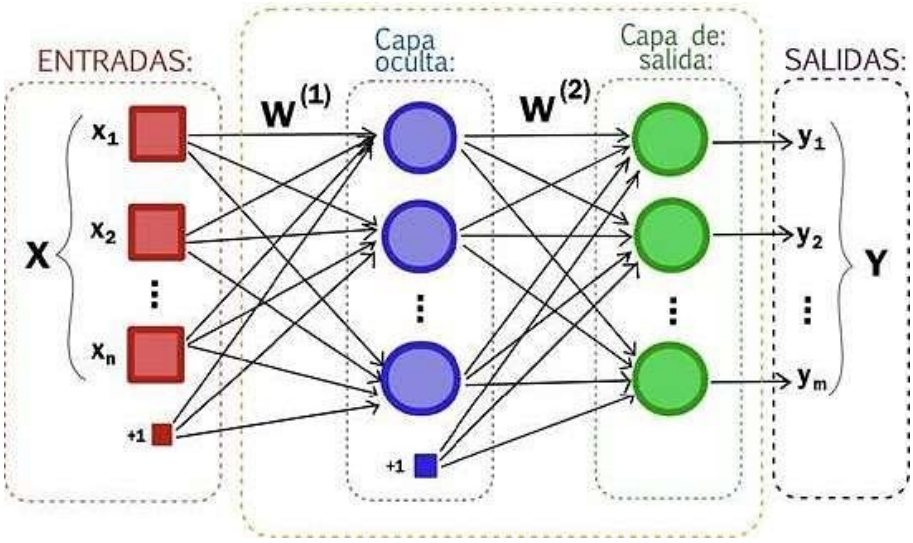


FIGURA 15. Esquema de una red neuronal artificial

Cuantas más capas intermedias tenga una red, tendrá mayor capacidad de detectar los detalles en los datos que se le proporcionen. La fortaleza de las redes neuronales reside en que los pesos de las conexiones pueden ajustarse según si los resultados obtenidos en la capa de salida son correctos o no, es un proceso conocido como entrenamiento. De esta manera, si el resultado obtenido es acertado, se refuerza el comportamiento de la red ante ese tipo de entradas, mientras que, si el resultado es erróneo, se ajustan los pesos para tratar de "acercar" los futuros resultados a una solución satisfactoria. A cada uno de los pasos en el entrenamiento de una red neuronal se le conoce como "época" (o epoch en inglés). Las neuronas de las capas ocultas pueden estar interconectadas de diferentes maneras, lo que determina las distintas topologías de redes neuronales.

Red neuronal convolucional

En las redes convolucionales cada neurona no se une con todas y cada una de las capas siguientes, sino que solo con un subgrupo de ellas, es decir, se especializa, por lo que se reduce

el número de neuronas necesarias y la complejidad computacional para su ejecución. Éstas utilizan como entradas una tabla de datos, lo que las hace idóneas para trabajo con imágenes por su naturaleza bidimensional. Una manera de poder visualizar un audio es mediante un espectrograma, una representación del sonido utilizando ventanas temporales y descomponiendo cada ventana en un espectro de frecuencias.

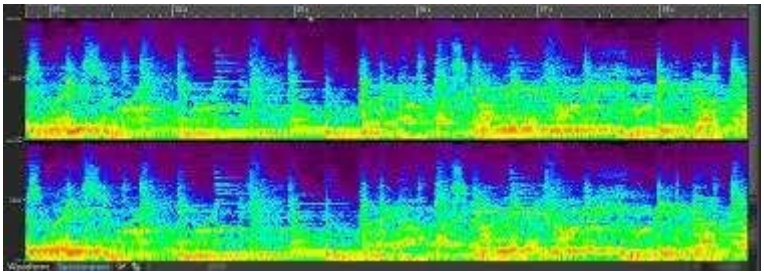


FIGURA 16. Muestra de un espectrograma

COEFICIENTES CEPSTRALES EN LAS FRECUENCIAS DE MEL (MFCC)

Los Coeficientes Cepstrales en las Frecuencias de Mel o MFCCs (Mel Frequency Cepstral Coefficients) son una representación espectral del sonido, la cual deriva de la Transformada rápida de Fourier. Estos se emplean para la representación de sonidos basados en la percepción humana, ya que lo caracterizan en tiempo y frecuencia.

Los MFCCs son una herramienta ampliamente usada en el área de reconocimiento automático de sonidos, estos nos ayudan a crear una imagen que representa las características fundamentales de eventos acústicos, simulando la interpretación del sonido más semejante a la del ser humano. El reconocimiento automático de sonidos requiere de la extracción de características de las señales de audio para la identificación de contenido relevante, así como obviar todas aquellas que poseen información poco valiosa como el ruido de fondo, emociones, volumen, tono, etc. y que no aportan nada al proceso de reconocimiento.

MATRIZ DE CONFUSIÓN

La matriz de confusión nos permite visualizar el desempeño del algoritmo. Cada columna en la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.

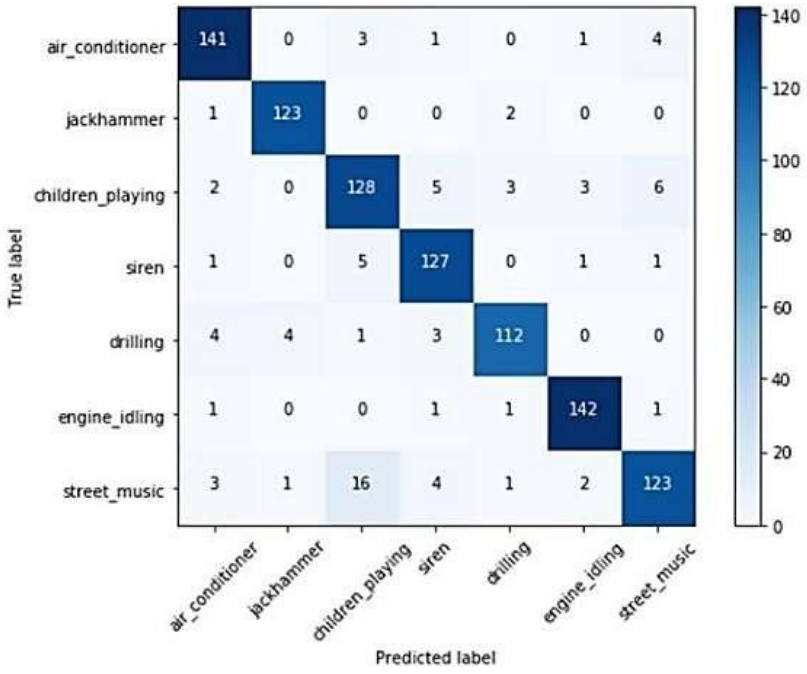


FIGURA 17. Matriz estandarizada

CONVERSOR ANALOGICO-DIGITAL (ADC)

Una señal eléctrica analógica es aquella en la que los valores de la tensión o voltaje varían constantemente y pueden tomar cualquier valor.

Un sistema de control (como un microcontrolador) no tiene capacidad alguna para trabajar con señales analógicas, de modo que necesita convertir las señales analógicas en señales digitales para poder trabajar con ellas.

La señal digital obtenida de una analógica tiene dos propiedades fundamentales:

- Valores: El valor del voltaje define 0 y 1. En nuestro caso es tecnología TTL (0 – 5V)
- Resolución analógica: número de bits que usamos para representar con una notación digital una señal analógica

Arduino Uno tiene una resolución de 10 bits, es decir, unos valores entre 0 y 1023, por lo que el valor de 0 voltios analógico es expresado en digital como B0000000000 (0) y el valor de 5V analógico es expresado en digital como B1111111111 (1023). Por lo tanto, todo valor analógico intermedio es expresado con un valor entre 0 y 1023, es decir, sumo 1 en binario cada 4,883 mV.

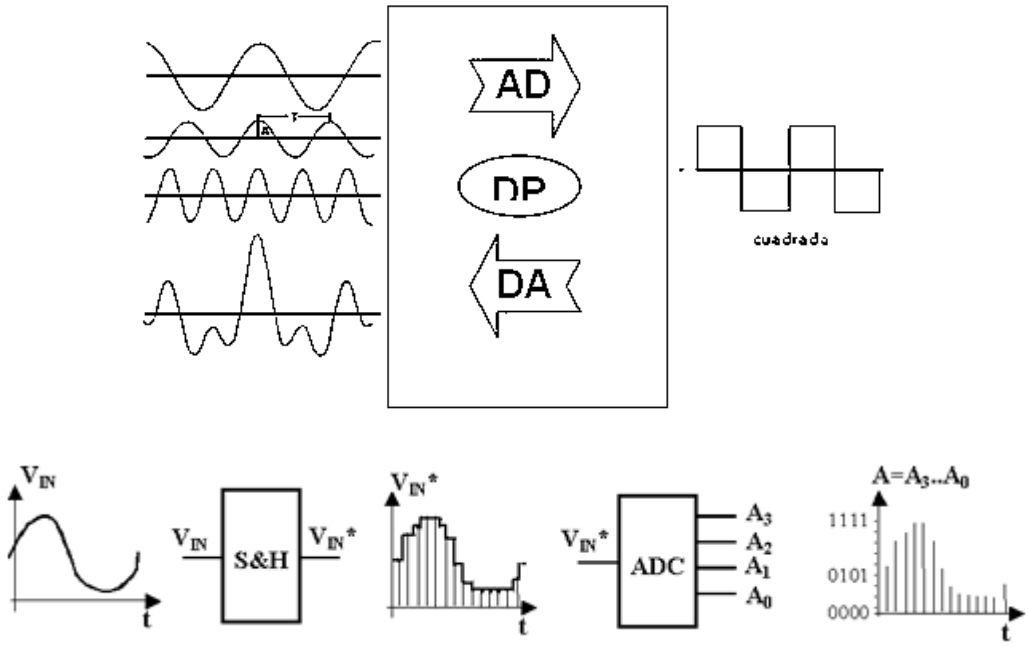


FIGURA 18. Representación gráfica de un Conversor Analógico-Digital

Los microcontroladores de Arduino contienen en la placa un conversor analógico-digital de 6 canales. El conversor tiene una resolución de 10 bits, devolviendo enteros entre 0 y 1023. En las entradas analógicas entran en juego los conversores analógico-digital.

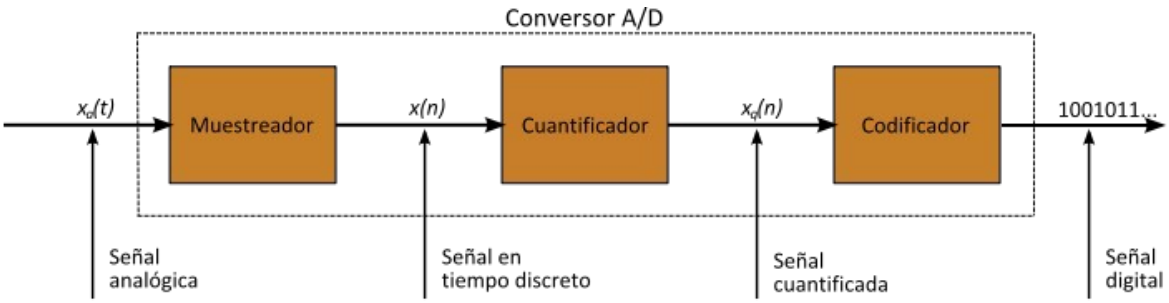


FIGURA 19. Proceso de Conversión Analógica-Digital

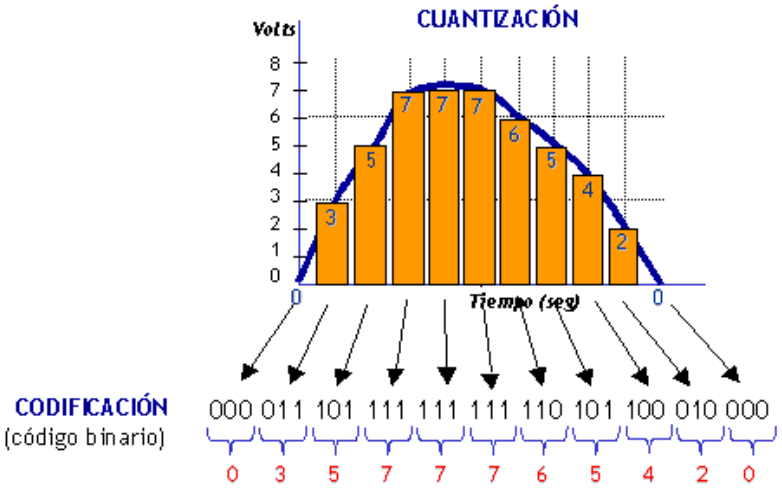


FIGURA 20. Cuantización de un ADC

Esta conversión consiste en la transcripción de señales analógicas en señal digital, con el propósito de facilitar su procesamiento (codificación, compresión, etcétera) y hacer la señal resultante (digital) más inmune al ruido y otras interferencias a las que son más sensibles las señales analógicas.

Como hemos dicho Arduino Uno tiene entradas analógicas que gracias a los conversores analógico digital puede entender ese valor el microcontrolador, pero no tiene salidas analógicas puras y para solucionar esto, usa la técnica de **PWM**.

Algunos pines digitales pueden usarse como salidas analógicas PWM:

Las Salidas PWM (Pulse Width Modulation) permiten generar salidas analógicas desde pines digitales.

La función para hacer una salida PWM en un pin es:

- *analogWrite ( )* – escribe un valor analógico (onda PWM) al pin especificado (no todos los pines digitales tienen accesibilidad al PWM).

En esta técnica se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva en relación con el período.

Duty Cycle = (tiempo que la salida está a uno o HIGH) / (periodo de la función)

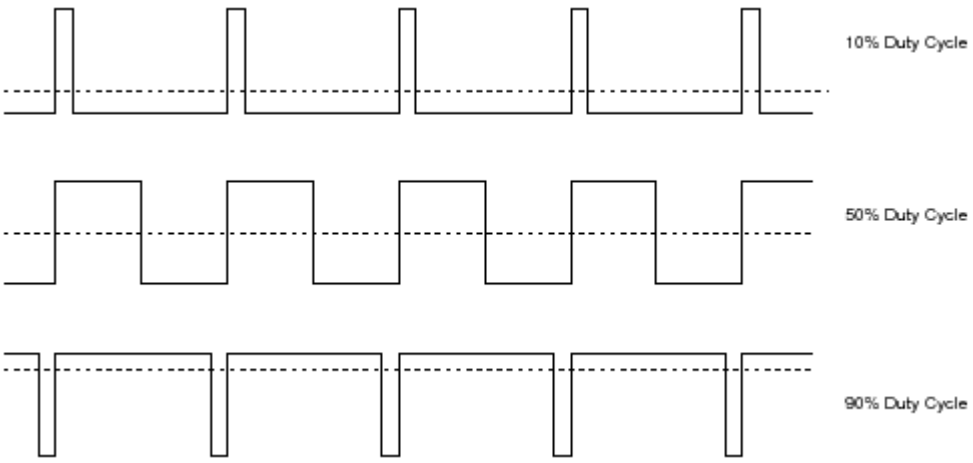


FIGURA 21. Duty Cycle de una señal PWM

TRANSISTOR NPN

El transistor es un componente capaz de amplificar o disminuir la corriente que circula por él. Es como un interruptor, pero se abre y cierra mediante corrientes eléctricos. El transistor NPN tiene tres patas, Colector, Base y Emisor, la mayoría de los transistores NPN tienen las patas en este orden (CBE). Sin embargo, algunos modelos tienen las patas al revés (EBC).

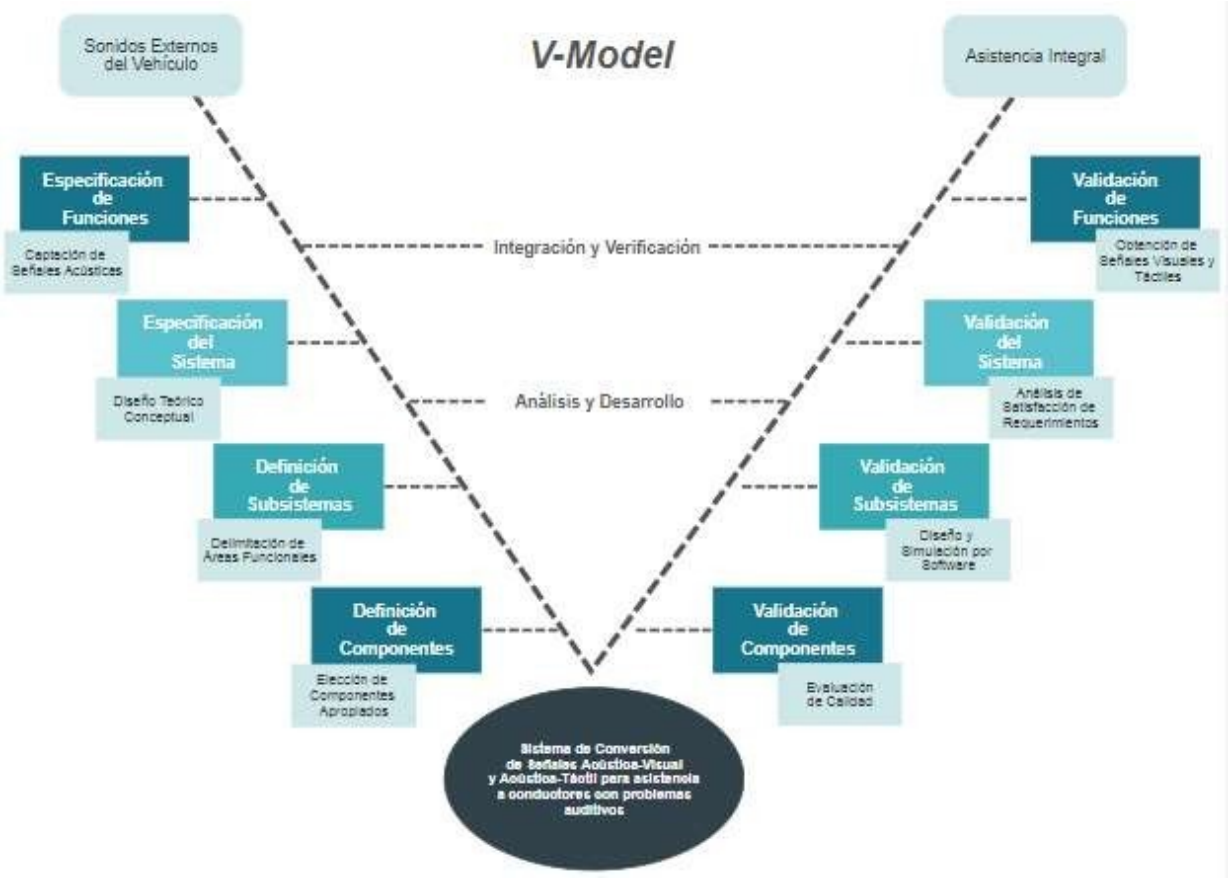
Según la corriente que pase por la Base, el transistor dejará pasar más o menos intensidad del Colector al Emisor, de tal manera que el transistor actúa como un interruptor. Si no hay corriente en la base se dice que el transistor está en Corte. Si la corriente en la Base es muy alta, se dice que el transistor está en Saturación y actuará como un interruptor cerrado.

2.3 Metodologías

En este punto se describen las conclusiones alcanzadas para la toma de decisiones finales que permitieron llevar a cabo el desarrollo del prototipo, tales como el framework y la metodología de procesos a seguir, así como la planificación del trabajo en secciones.

Para empezar a guiar este proyecto, se revisaron distintos artículos sobre el tratamiento de audio con algoritmos de Machine Learning, lo que nos llevó por consiguiente a idear una metodología específica con base en la Metodología de Diseño de Ingeniería V-Model.

FIGURA 22. Diagrama V-Model para Metodología de Diseño de Ingeniería



2.4 Herramientas de Trabajo

PROTEUS

Es una aplicación para la ejecución de proyectos de construcción de equipos electrónicos en todas sus etapas: diseño del esquema electrónico, programación del software, construcción de la placa de circuito impreso, simulación de todo el conjunto, depuración de errores, documentación y construcción.

En el mundo de la formación, Proteus se muestra como una herramienta magnífica porque permite al alumno realizar modificaciones tanto en el circuito como en el programa, experimentando y comprobando de forma inmediata los resultados y permitiéndole de esta forma aprender de forma práctica y sin riesgos de estropear materiales de elevado coste.

MATLAB

(Abreviatura de ***MAT****rix* ***LAB****oratory*, «laboratorio de matrices») es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). . Este lenguaje es interpretado, y puede ejecutarse tanto en el entorno interactivo, como a través de un archivo de script (archivos *.m). Este lenguaje permite operaciones de vectores y matrices, funciones, cálculo lambda, y programación orientada a objetos.

Audio Toolbox proporciona herramientas para el procesamiento de audio, el análisis de la voz y la medición acústica. Incluye algoritmos para el procesamiento de señales de audio (tales como la ecualización y el control del rango dinámico) y la medición acústica (tales como la estimación de la respuesta a impulso, el filtrado de octavas y la ponderación perceptiva). También proporciona algoritmos para la extracción de características de audio y voz (tales como MFCC y tono) y la transformación de señales de audio (tales como el banco de filtros gammatono y el espectrograma con espaciado de Mel).

PYTHON

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad.

ANACONDA/JUPYTER

La elección de este entorno de programación se basó en términos de producción por lo que se empleó Anaconda, una distribución de Python. El entorno se optimizó para poder programar en Python en su versión 3.7.6. Para la escritura de código y compilación, se empleó Jupyter porque permite desarrollar código Python dividiéndolo en celdas, lo que a su vez permite ejecutar de forma independiente cada celda, sin necesidad de ejecutar el script completo, y observar la salida de cada fragmento de código fácilmente. Cada vez que se ejecuta una celda, se guardan sus resultados en memoria para permitir su utilización en las celdas que dependan de la misma.

RASPBERRY PI

El diseño e implementación de modelos de Deep Learning o de Machine Learning requieren una elevada carga computacional. Dichos modelos pueden contar con miles de parámetros, los cuales, al ser modificados, cambian directamente el resultado.

Los requisitos en cuanto a hardware son más exigentes, por lo que para la realización del proyecto se han empleado diferentes dispositivos a través de los cuales se ha podido implementar el sistema.

Por una parte, se empleó un ordenador para el entrenamiento y desarrollo de una red neuronal la cual es capaz de clasificar los fragmentos de audio. Por otra parte, se empleó una Raspberry Pi, la cual se empleó como un módulo sensor con ayuda de un micrófono para poder captar los eventos acústicos que se requieren reconocer y clasificar, así como también se utilizó una tarjeta

de sonido USB externa para poder transformar un puerto USB en entrada y salida de audio por 3.5mm, esto debido a que la Raspberry sólo cuenta con conexión de salida de audio tipo 3.5mm.

Especificaciones del equipo:

- CPU de dos núcleos Intel Core i3-4005U @ 1.70 GHz
- 8.00 GB de RAM.
- Sistema Operativo Windows 10

Aunque la fuente de alimentación recomendada para la RPi 4 es de 3A, esa no es la corriente consumida. La corriente típica consumida por la propia placa sería de 600mA. Si la RPi alimenta a su vez otros periféricos por sus puertos USB, la máxima corriente que se puede suministrar es de 1.2A. Es decir, trabajando de manera individual su consumo seria de 600mA y si alimenta a su vez, por ejemplo, a un disco externo, el consumo podría llegar hasta los 1.8A.

La Raspberry que se ha empleado es una Raspberry Pi Model 4B con 4 GB de memoria RAM.

FIGURA 23. Raspberry Pi 4 Model B

MICRÓFONO

El micrófono empleado es un micrófono omnidireccional, el cual permite captar el sonido en 360°, lo que lo hace ideal para grabar un sonido desde cualquier dirección, independientemente de donde esté situado. Así mismo, es ideal para grabar en un espacio abierto, ya que capta el sonido por igual en todas las direcciones.

FIGURA 24. Micrófono Omnidireccional

Se ha adaptado una tarjeta de sonido que permite la conversión del plug de salida del micrófono con la entrada USB que tiene la Raspberry a utilizar.

FIGURA 25. Tarjeta de sonido USB

ARDUINO UNO

Arduino Uno es una placa de microcontrolador basada en ATmega328P. Tiene 14 pines de entrada / salida digital (de los cuales 6 se pueden usar como salidas PWM), 6 entradas analógicas, un resonador cerámico de 16 MHz (CSTCE16M0V53-R0), una conexión USB, un conector de alimentación, un encabezado ICSP y un botón de reinicio. Contiene todo lo necesario para soportar el microcontrolador; simplemente conéctelo a una computadora con un cable USB o enciéndalo con un adaptador de CA a CC o una batería para comenzar. Puede jugar con su Uno sin preocuparse demasiado por hacer algo mal, en el peor de los casos, puede reemplazar el chip por unos pocos dólares y empezar de nuevo.

"Uno" significa uno en italiano y fue elegido para marcar el lanzamiento de Arduino Software (IDE) 1.0. La placa Uno y la versión 1.0 del software Arduino (IDE) fueron las versiones de referencia de Arduino, ahora evolucionadas a versiones más recientes. La placa Uno es la primera de una serie de placas USB Arduino y el modelo de referencia para la plataforma Arduino; Para obtener una lista extensa de placas actuales, pasadas o desactualizadas, consulte el índice de placas Arduino.

DATASET

Durante el diseño de un modelo de aprendizaje automático, es fundamental elegir correctamente el dataset a utilizar. Para trabajar este proyecto se utilizó una base de datos de sonido urbano “UrbanSound8K”, publicado por los investigadores del proyecto SONYC.

El conjunto de datos contiene 8732 extractos de sonidos etiquetados en formato wav, cuya duración de cada muestra no supera los 4 segundos, clasificados en diez pliegues diferentes, según la publicación “Urban Sound Taxonomy”:

- ***classID = 0*** **air_conditioner**
- ***classID = 1*** **car_horn**
- ***classID = 2*** **children_plying**
- ***classID = 3*** **dog_bark**
- ***classID = 4*** **drill**
- ***classID = 5*** **engine_idling**

- ***classID = 6*** **gun_shot**
- ***classID = 7*** **jackhammer**
- ***classID = 8*** **siren**
- ***classID = 9*** **street_music**

Además de los extractos de sonido, también se proporciona un archivo CSV que contiene metadatos sobre cada extracto.

Esto incluye:

- **slice-file-name:** El nombre del archivo de audio. El nombre toma el siguiente formato: [fsID] - [classID] - [ocurrenceID] - [sliceID] .wav.
- **fsID:** el ID de Freesound de la grabación de la que se toma este extracto (segmento)
- **classID:** Un identificador numérico de la clase de sonido.
- **ocurrenceID:** Un identificador numérico para distinguir diferentes ocurrencias del sonido dentro de la grabación original.
- **sliceID:** Un identificador numérico para distinguir diferentes cortes tomados de la misma ocurrencia.
- **start:** La hora de inicio del corte en la grabación original de Freesound.
- **end:** la hora de finalización del corte en la grabación original de Freesound.
- **salience:** Una calificación de prominencia (subjetiva) del sonido. 1 = primer plano, 2 = fondo.
- **fold:** El número de pliegue (1-10) al que se ha asignado este archivo.
- **classID:** Un identificador numérico de la clase de sonido.
- **class:** El nombre de la clase: aire acondicionado, bocina de coche, niños jugando, ladrido de perro, taladro, motor al ralentí, disparo, martillo neumático, sirena, música callejera.

Para el entrenamiento de la red neuronal, empleamos el indicador de clase (*classID*).

LENGUAJE DE PROGRAMACIÓN

Para el desarrollo de este proyecto, elegimos Python y Matlab como lenguajes de programación para la elaboración de los distintos módulos, puesto que se han convertido los lenguajes de programación más usados en todo el mundo. La mayoría de código desarrollado para Inteligencia Artificial, Machine Learning y Deep Learning, se programa en estos lenguajes.

A pesar de no tener conocimientos previos obtenidos durante la carrera acerca de estos, se ha decidido utilizarlo debido a que son lenguajes con una aplicación amplia en ingeniería y fue posible obtener el aprendizaje de manera autodidacta gracias al previo conocimiento del lenguaje C.

LIBRERIAS

Se han implementado técnicas de aprendizaje automático, dichas técnicas fueron de utilidad para realizar la clasificación e identificación de sonidos.

La clasificación y el reconocimiento de sonidos es una disciplina especialmente compleja, a diferencia del reconocimiento de voz. En este trabajo, identificaremos las características que pueden ser extraídas de los datos de sonidos. Una vez obtenido el conjunto de datos con el que se trabajará el modelo de Machine Learning, se debe dar el formato adecuado, es decir, hacer un preprocesamiento de este para eliminar los datos que no sean de nuestro interés.

Para llevar a cabo el preprocesamiento y manejo de los datos es fundamental saber qué librerías se van a utilizar.

Por defecto, Anaconda cuenta con diversas librerías preinstaladas, de las cuales empleamos las siguientes:

- **Numpy:** Principal librería para la computación científica. Posee funciones para trabajar en los campos del álgebra, transformados de Fourier, matrices N-dimensionales, y el campo de la estadística.
- **Pandas:** Es una librería que posee herramientas para el análisis de datos de manera rápida, potente y fácil de utilizar.

- **Matplotlib:** Es una librería para la representación y visualización de datos de distintas formas, ya sea estática, animada, interactiva, en 2D, 3D, etc.

Necesitamos disponer de herramientas que sean de utilidad para manejar nuestra base de datos, en específico, señales de audio. Principalmente se utilizará la librería de código abierto "Librosa", la cual posee funciones para el análisis de sonido y música y nos ayudará a extraer las características del sonido, descomponer en su parte temporal y frecuencial, así como visualizar su espectrograma, entre otras cosas. En general, esta librería sirve para poder realizar el preprocesamiento de las señales de audio.

Existen diferentes entornos para poder trabajar en el campo del aprendizaje automático, que permiten el desarrollo de modelos de Machine Learning; de los más utilizados y los que nos ayudaron en el desarrollo del proyecto son:

- **TensorFlow**: TensorFlow es una biblioteca de software gratuita y de código abierto que sirve para el flujo de datos la programación diferenciable en una variedad de tareas. Es una biblioteca matemática simbólica y también se utiliza para aplicaciones de aprendizaje automático como las redes neuronales ya que destaca por su sistema de nodos multicapa, lo que permite un rápido entrenamiento de redes neuronales en grandes datasets.
- **Keras:** Keras es una librería que permite construir redes neuronales de alto nivel de una forma sencilla. Es una librería escrita en Python y permite trabajar sobre Theano y TensorFlow, principalmente. Es altamente modular y escalable, además de permitir el prototipado rápido de redes neuronales, debido a su sencillez. Los datos se preparan en tensores y sus modelos se basan en capas, con una capa para los tensores de entrada, otra para los de salida y un número indeterminado de capas ocultas.
- **PyTorch:** PyThorch es un framework de Deep Learning para una experimentación rápida y flexible. Permite la computación con tensores con una potente aceleración por GPU. Usualmente se utiliza como sustituto de NumPy para aprovechar la competencia de cómputo de las GPUs., proporcionando integración con librerías de aceleración de GPU.

- **Theano:** Theano es una librería que define arrays multidimensionales de una forma similar a NumPy, así como operaciones y expresiones matemáticas, por lo que tiene una estrecha integración con NumPy. Sus ajustes de eficiencia y estabilidad permiten obtener resultados mucho más precisos, incluso con valores muy pequeños.
- **SciKit-Learn:** Es un paquete basado en SciPy, haciendo un uso intenso de sus operaciones matemáticas. Proporciona una interfaz concisa y consistente para los algoritmos de Machine Learning más comunes, facilitando su incorporación en sistemas en producción. Combina un código de calidad y una buena documentación, así como una sencillez de uso y un alto rendimiento, lo que la convierte en una librería estándar para el desarrollo de modelos de Machine Learning en Python

3. PLANTEAMIENTO DEL PROBLEMA

3.1 Requerimientos

Los problemas auditivos en las personas han pasado por diferentes estadios, tanto en la posibilidad de obtener un permiso de conducir hasta la negación de algún empleo relacionado con el manejo del automóvil, incluso se ha llegado a limitar la velocidad de circulación y a hacer exámenes especiales, lo cual ha sido categorizado como un problema de exclusión. Y aunque con el paso del tiempo ha cambiado, hoy por hoy se han buscado alternativas que faciliten el acceso a la obtención de un permiso de conducir en igualdad de condiciones.

Existen en el mundo más países donde se prohíbe la conducción a personas con discapacidad auditiva que países que lo permiten. Por tal motivo se considera trascendental el desarrollo de este proyecto, ya que proporciona herramientas alternas al conductor, las cuales se verían directamente reflejadas en la seguridad de la conducción.

Para la implementación de este sistema de asistencia al conductor se presentan los retos siguientes:

- Diseño y construcción del sistema que permita la conversión del sonido externo del vehículo en señales visuales y táctiles proporcionales a la señal acústica recibida.
- Selección de los sensores adecuados para la captación de señales con frecuencias específicas.
- Programación adecuada para el tratamiento acústico mediante inteligencia artificial.
- Adaptación de las señales de salida a alguna parte estratégica del vehículo para la fácil visualización de estas.
- Acoplar una comunicación efectiva entre los elementos electrónicos involucrados.

El cumplimiento de tales puntos permitirá al conductor con discapacidad auditiva acceder a mejores trabajos, tener una mejor calidad de vida y decidir sobre su movilidad con total libertad.

3.2 Propuesta de Solución

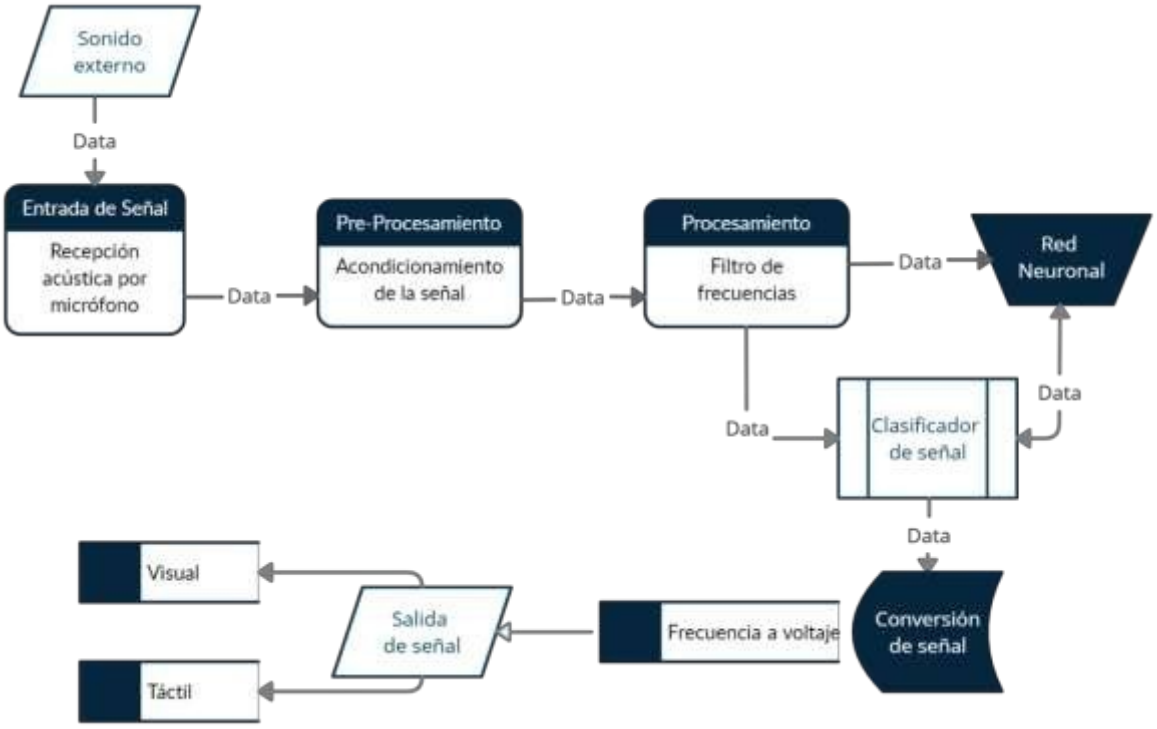


FIGURA 26. Diagrama de Proceso del Sistema

Los sonidos urbanos percibidos al momento de la conducción son bastos, por lo que cada uno de ellos entran de manera directa a través de una recepción acústica por micrófono, el cual por medio del módulo de entrada de sonido de una tarjeta podrá ingresar al ordenador, el cual hará el acondicionamiento de tal señal para poder así hacer el procesamiento de audio y a través de fórmulas y funciones en Matlab, nos permitirá observar el espectro del audio obtenido y a su vez realizar un filtrado de frecuencias, permitiendo continuar con el proceso a aquellas cuyo rango sea el de nuestro interés.

Estos datos de frecuencia son almacenados y llevados al algoritmo que conforma una red neuronal previamente entrenada, la cual clasificará las frecuencias de acuerdo con los rangos en la base de datos de sonidos recabados. Al clasificarlos dentro de los sonidos de nuestro interés, sirenas de emergencia y claxon, estos datos de frecuencia son convertidos directamente a voltaje, y así podamos visualizar por leds y sentir por vibración la respuesta correspondiente al aumento o disminución de tales frecuencias.

4. DESARROLLO DE LA SOLUCIÓN

4.1 Módulo de Alimentación

Considerando que la tarjeta que utilizaremos es una Raspberry Pi 4 y un Arduino Uno que requieren una alimentación de 5V vía USB-C o 5V vía cabezal GPIO, con una corriente típica de consumo de 600mA y pensando en que su aplicación es automotriz, es necesario realizar un conversor de tensión de los 12V -entregados por la batería del automóvil- a los 5V requeridos por las tarjetas de desarrollo a ocupar.

Para este fin se ocupa la topología Buck/Step-Down la cual permite reducir el valor de tensión respecto a la tensión de entrada.

El circuito regulador tipo Buck puede encontrarse dentro del circuito integrado MC34063, donde la salida de 5V se genera a partir de un voltaje de entrada de 12V.

Se realizan los cálculos necesarios de los componentes, basándonos en la configuración dada por la compañía "ON Semiconductor" que elabora el circuito integrado y tomando como referencia las fórmulas proporcionadas en el Data Sheet del mismo (ver **Anexo II**).

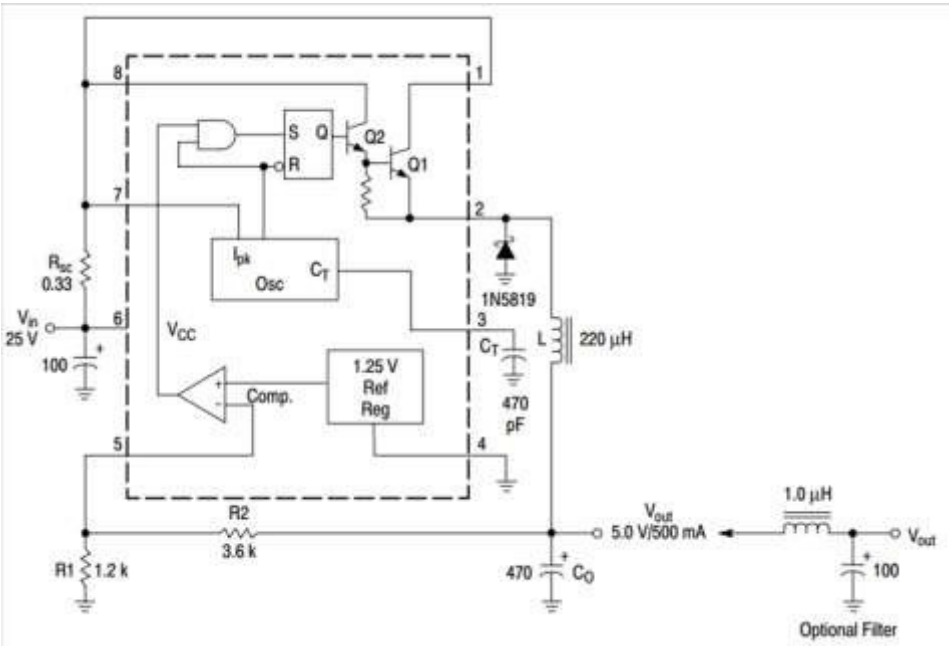


FIGURA 27. Topología Buck/Step-Down

En seguida se muestran los requisitos de diseño:

- V_{IN} = 12 V, proveniente de la batería del automóvil.
- V_{OUT} = 5 V, necesarios para alimentar la tarjeta de desarrollo.
- I_{OUT} = 600 mA, corriente típica de consumo de la tarjeta de desarrollo.
- F_{OSC} = 10 kHz, hasta 100 kHz.
- V_F = 0.45 V, tensión de codo del diodo Schottky utilizado (LM 5818).
- $V_{ripple(pp)}$ = 180 mV, tensión de risado pico a pico.

1. Relación del tiempo del transistor en conducción y el tiempo en que permanece apagado:

$$\frac{T_{on}}{T_{off}} = \frac{V_{out} + V_f}{V_{in} - V_{sat} - V_{out}} = \frac{5 + 0.45}{12 - 0 - 5} = 0.778$$

El voltaje de saturación (Vsat) en este caso es el propio del transistor externo del circuito integrado para elevar la corriente de salida >1.5 A.

2. Periodo de la señal:

$$(T_{on} + T_{off}) = \frac{1}{f} = \frac{1}{10000} = 100\ \mu s$$

3. Tiempo de apagado:

$$T_{off} = \frac{T_{on} + T_{off}}{\frac{T_{on}}{T_{off}} + 1} = \frac{100\mu s}{0.778 + 1} = 56.24\ \mu s$$

4. Tiempo de encendido:

$$T_{on} = (T_{on} + T_{off}) - T_{off} = 43.76\ \mu s$$

5. Condensador de oscilación:

$$C_T = 4x10^{-5} \times T_{on} = 1.75\ nF$$

6. Corriente de pico de la bobina:

$$I_{pk} = 2 \times I_{out} = 1.2\ A$$

7. Resistencia de oscilación:

$$R_{sc} = \frac{0.3}{I_{pk}} = 0.25\ \Omega$$

8. Bobina:

$$L = \left(\frac{V_{in} - V_{sat} - V_{out}}{I_{pk}}\right) \times T_{on} = 255\mu H$$

9. Condensador del filtro de salida:

$$C = \frac{I_{pk} \times (T_{on} + T_{off})}{8 \times V_{ripple(pp)}} = 120\ \mu F$$

10. Resistencias de Feedback:

Fijando R_2=10KΩ y despejando R_1 de la siguiente fórmula:

$$|V_{out}| = 1.25\ \left(1 + \frac{R_2}{R_1}\right)$$

$$R_1 = \frac{R_2}{\frac{V_{out}}{1.25} - 1} = 3.33\ k\Omega$$

TABLA 3. Valores calculados/estandarizados de los componentes

Valores de los componentes calculados para un DC-DC 12 a 5 V	Valores comerciales reales para simulación y armado
C_T=1.75 nF	C_T=2 nF
R_{SC}=0.25 Ω	R_{SC}=0.33 Ω
L=255 µH	L=280 µH
C=120 µF	C=100 µF
R1=3.33 kΩ	R1=3.3 kΩ
R2=10 kΩ	R2=10 kΩ

Los valores estandarizados nos dan un amplio rango de tolerancia para soportar el aumento de voltaje que pudiera presentar la bateria del automóvil.

Se presenta el armado del circuito eléctrico con los valores estandarizados obtenidos.

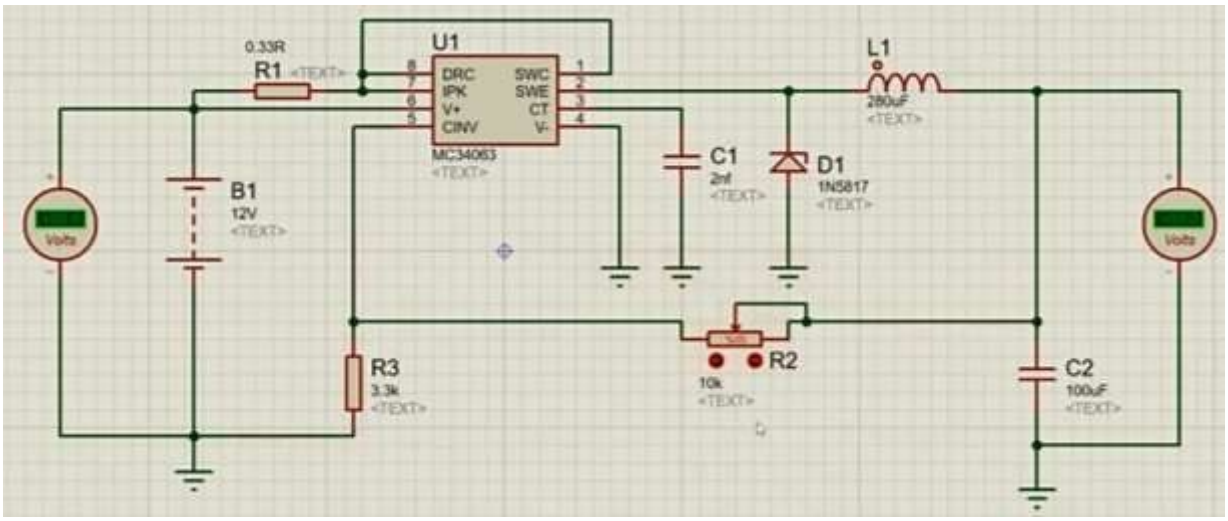


FIGURA 28. Circuito eléctrico del regulador Buck/Step-Down en Proteus

4.2 Módulo de Recepción

El nodo de recepción se implementa en una Raspberry, la cual permite la captura de un evento acústico y a través de las características del sonido se realiza la clasificación posterior.

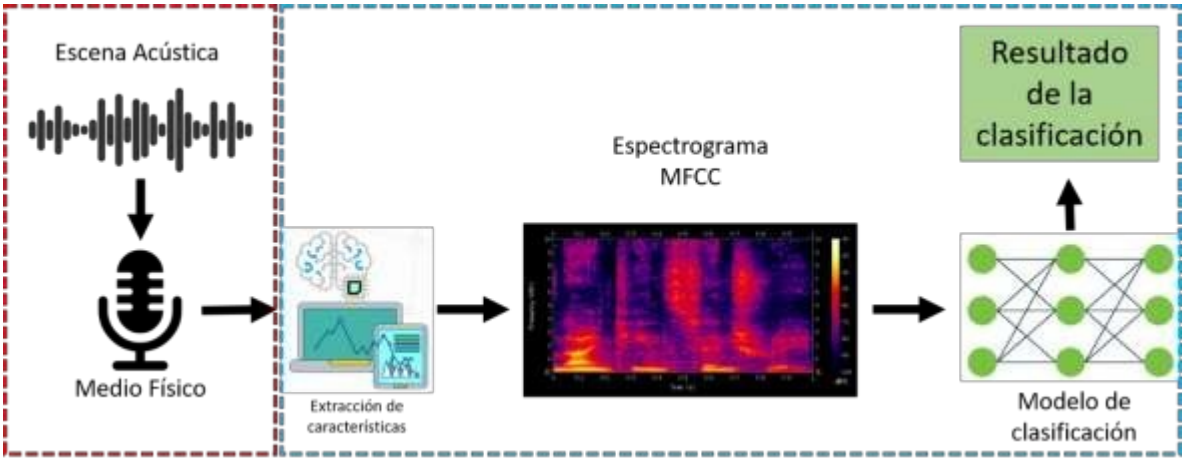


FIGURA 29. Esquema general para clasificación

Se ha conectado un micrófono a la Raspberry Pi el cual funciona como nodo de recepción. Este nodo capta los sonidos, el fragmento de audio captado se almacenará en un archivo en formato .wav. que después del módulo de filtrado, el módulo de clasificación se encarga de extraer las características del archivo de audio, que a su vez son evaluadas y finalmente clasificadas para el uso final.

Para ello es necesario conectar el extremo micro-USB del cable USB a la Raspberry Pi y el extremo USB normal del cable USB a la computadora y esperar hasta que el LED PWR en el hardware comience a parpadear.

En seguida se realiza la conexión del micrófono a uno de los puertos USB del hardware. A través de la pantalla de configuración de hardware se configura la red Raspberry Pi.

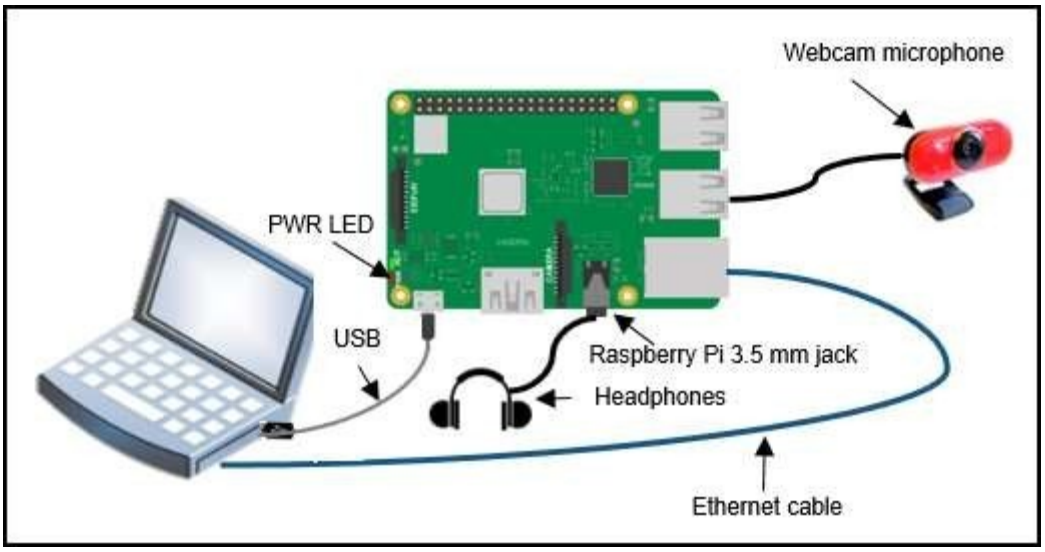


FIGURA 30. Esquema de conexión para la recepción de la señal de audio

Se crea un objeto *raspi* que representa una conexión del software MATLAB a la placa Raspberry Pi. Para interactuar con la placa Raspberry Pi se utiliza este objeto con las funciones enumeradas en Funciones de objeto.

```
mypi = raspi

mypi = raspi(ipaddress,username,password)
mypi = raspi(hostname,username,password)

mypi = raspi(name)
mypi = raspi(serialnumber)
mypi = raspi(__,"Timeout",time)
```

Consecutivamente, se crean los objetos del sistema *captureObj* y *playerObj* para el procesamiento de audio en el hardware Raspberry Pi. El *captureObj* realiza una conexión al dispositivo fuente de audio. La entrada de audio se captura a una frecuencia de muestreo de 48000 y muestras por cuadro de 4800.

Además, el objeto *Audiocapture* representa una conexión entre un dispositivo de entrada de audio y un hardware Raspberry Pi. Es posible interactuar con el dispositivo de audio utilizando las funciones enumeradas en Funciones de objeto.

```
audioCaptureObj = audiocapture(mypi,DeviceName)
audioCaptureObj = audiocapture(mypi,DeviceName,Name,Value)
```

La función *capture* recoge datos del dispositivo de entrada de audio conectado al hardware Raspberry Pi. Esta función utiliza el marco del controlador de Arquitectura de sonido avanzada de Linux (ALSA) para leer datos de audio. Tales datos leídos son guardados como un archivo de formato particular, para su utilización posterior.

4.3 Módulo de Preprocesamiento

Al recibir los sonidos externos por el micrófono y a través de la tarjeta de desarrollo, se direcciona el audio obtenido a la carpeta en que se ejecuta el algoritmo en Matlab, como un archivo en formato .wav para poder muestrear la señal de audio análoga obtenida del ambiente.

Al muestrear la señal de audio se obtiene una equivalencia matemática igual a la multiplicación por un tren de deltas; esta multiplicación en el dominio del tiempo equivale a una convolución en el dominio de la frecuencia. La transformada de un tren de deltas cuya distancia entre deltas es T es otro tren de deltas cuya separación es $\frac{1}{T}$.

Tal convolución de la función por un tren de deltas da como resultado una copia de la función en la posición de cada delta. Nos queda que la transformada de una señal muestreada es un tren de funciones, cada una de las cuales representa la transformada de la señal sin muestrear, separadas por sus centros en la frecuencia de muestreo.

Para conservar la transformada de la función original se debe muestrear a una frecuencia en la que no se produzcan solapamientos, siendo ésta a más del doble del ancho de banda de la señal o frecuencia de corte; se conoce a este requisito como criterio de NYQUIST o teorema del muestreo de SHANNON. Así que, para estudiar el espectro de una función muestreada sólo tendremos en cuenta el rango de frecuencias que sea menor a la frecuencia máxima o de corte.

De manera concreta el teorema establece que "puede descomponerse" o que "puede construirse" una función periódica cualquiera, lo cual no significa que resulte sencillo, sin embargo, a través de Matlab se obtienen directamente los resultados gracias a las funciones incluidas: *transformada rápida de Fourier* (fft) y *transformada inversa de Fourier* (ifft).

El proceso de programación en Matlab utilizado para la obtención del espectro del audio -que es la base para el posterior filtrado- es el siguiente:

1. Importación del audio: A través de la función de Matlab *audioread,* la cual nos entrega un vector de datos del audio y una frecuencia de muestreo.

```
1     % Audio Import
2
3 -   [a , fs] = audioread("Audio.wav");
```

2. Duración de audio: A través de la función *length* podemos conocer la longitud del vector de datos del audio, que al dividir entre la frecuencia de muestreo resulta en la duración en segundos del audio recibido.

```
4 -     d = length(a)/fs;
```

3. Conversión de estéreo a monofónico: El vector de datos nos indica que el audio es recibido en estéreo y para procesarlo necesitamos procesar canal por canal.
   Para ello sumamos los valores del vector de audio en su posición 1 (columna 1) y su posición 2 (columna 2), lo cual resulta en el doble de la amplitud, por ello es necesario multiplicar todo por ½.
   Lo anterior permite que el vector de nuestro audio tenga el mismo número de elementos, pero en una sola posición, es decir, en mono.

```
5 -     a_m = 0.5*(a(:,1) + a(:,2)).';
```

4. Gráfica de la forma de onda: Para ello se requiere de un vector de tiempo, obtenido por la función *linspace,* desde 0 hasta la duración del audio (d) y con la cantidad de muestras que tiene nuestro sonido.

```
7       % Waveform Plot
8
9 -     t = linspace(0, d, length(a_m));
10
11 -    figure();
12 -    plot(t, a_m);
13 -    title("Audio Waveform");
14 -    xlabel("Time [s]");
15 -    ylabel("Amplitude");
16 -    grid on;
```

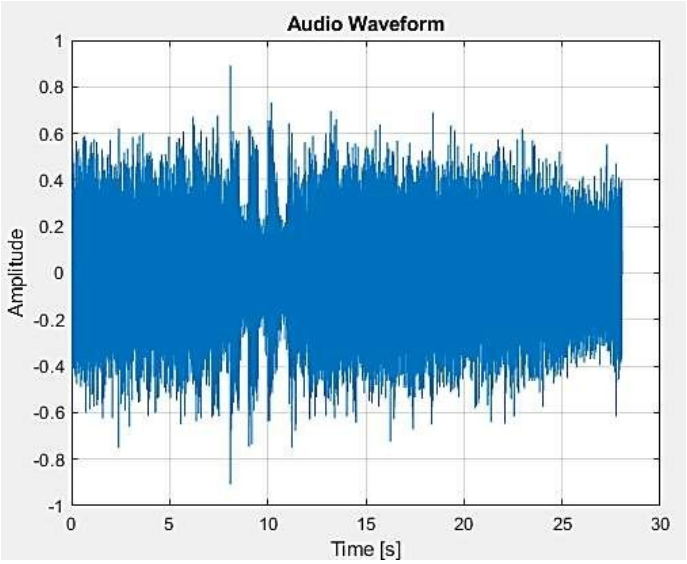


FIGURA 31. Forma de audio original

5. Análisis de frecuencia: Haciendo uso de la Transformada rápido de Fourier que Matlab implementa a través de la función *fft* para realizar un análisis en el dominio de la frecuencia.

    - Espectro del audio en un solo canal. Utilizando la función *fftshift* centramos el espectro resultante en 0.

```
18      %% Frecuency
19
20 -    A_m = fftshift(fft(a_m));
```

    - Eje de frecuencia.

```
22 -    f = linspace(-fs/2, fs/2, length(A_m));
```

    - La Transformada de Fourier nos arroja valores complejos, y para conocer la magnitud del espectro, realizamos lo siguiente:

```
24 -    mag_A = abs(A_m);
```

La onda de sonido obtenida del audio se puede caracterizar, en términos de las amplitudes de las ondas sinusoidales componentes que la conforman. Este conjunto de números, indica el contenido de armónicos de un sonido, y es referido como el espectro armónico del sonido, el cual se obtiene de este módulo de preprocesamiento.

4.4 Módulo de Filtrado

Los filtros digitales son fundamentales para casi todos los sistemas de procesamiento de señales. Los filtros eliminan los artefactos no deseados de las señales para mejorar su calidad y prepararlas para su posterior procesamiento. Los filtros digitales en este caso permiten la eliminación de ruido y valores atípicos, suavizado de la señal, y el paso de las señales entre 400 Hz y 2000 Hz, rango en el que los sonidos de interés en este proyecto se encuentran.

Se realiza en conjunto un filtro pasa-bajas que converge con un filtro pasa-altas para evaluar su respuesta al espectro resultante, además de un filtro Butterworth pasa-banda, el cual optimiza el filtrado por sus propiedades.

Filtro Pasa-Bajas

Este filtro paso-bajo permite el paso de señales por debajo de la frecuencia de corte indicada (2000 Hz), y atenúa señales por encima de la frecuencia de corte (banda de atenuación).

El filtro paso bajo, mediante la función *low pass filter (lpf)* de Matlab produce un cambio lento en los valores de salida para facilitar la observación de tendencias y aumentar la proporción de señal a ruido con una degradación mínima de la señal.

```
40 -    lpf = 1.*(abs(f)<2000);
```

A través de la línea anterior se crea un filtro ideal paso-bajo que manda un 0 lógico a la señal a la banda de atenuación y un 1 a la banda de paso, esto en base al valor absoluto de las frecuencias del espectro del audio, considerando que esta tiene valores negativos y positivos.

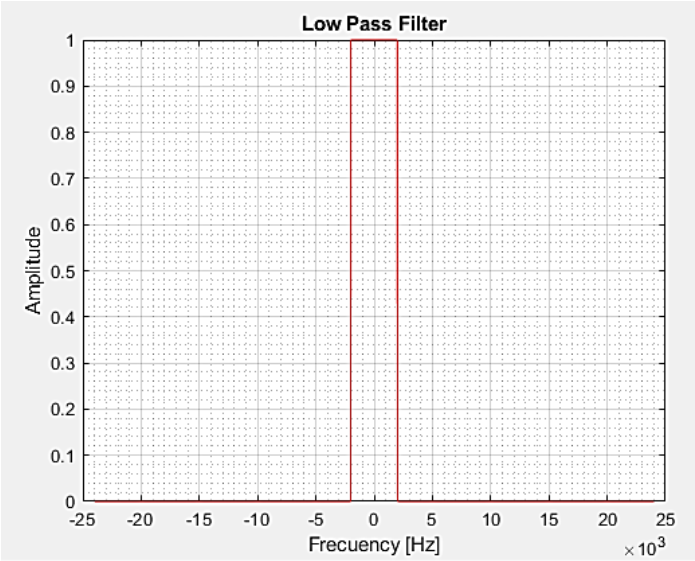


FIGURA 32. Filtro Pasa-Bajas

De esta manera, las frecuencias absolutas mayores a 2000 Hz del espectro serán rechazadas y solo se preservaran hasta este punto aquellas de 0 a 2000 Hz.

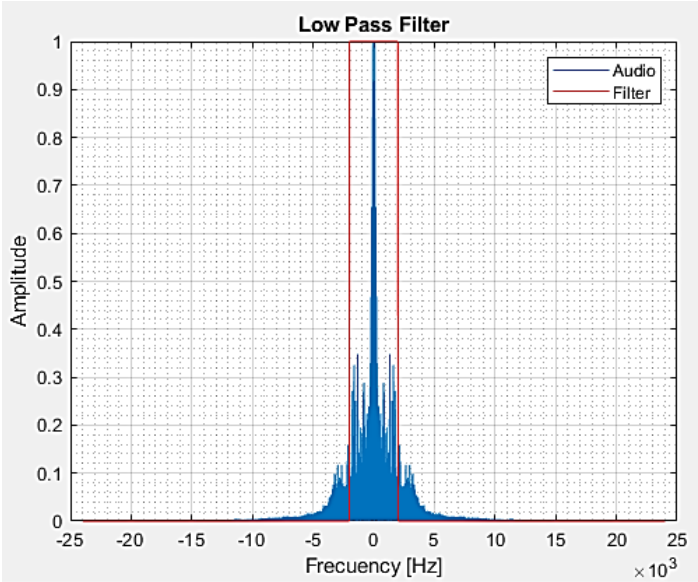


FIGURA 33. Espectro y Filtro Pasa-Bajas

Filtro Pasa-Altas

Si el valor absoluto de los elementos del vector de frecuencia es mayor o igual a 400 Hz, este enviará un valor en alto, preservando esos valores de frecuencia; y enviando a 0 los elementos por debajo de 400 Hz, impidiendo su avance.

```
101 -    hpf = 1.*(abs(f)>=400);
```

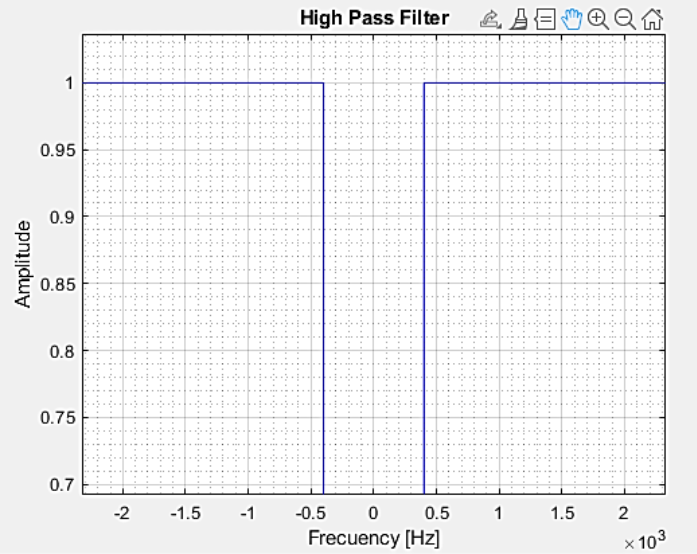


FIGURA 34. Filtro Pasa-Altas

Para poder visualizar el espectro del audio con la implementación del filtro pasa-altas al igual que con el pasa-bajas, requerimos al siguiente código:

```
115 -    figure();
116 -    plot(f, mag_A/max(mag_A));
117 -    hold on;
118 -    plot(f, hpf, 'r');
119 -    legend("Audio", "Filter");
120 -    title("High Pass Filter");
```

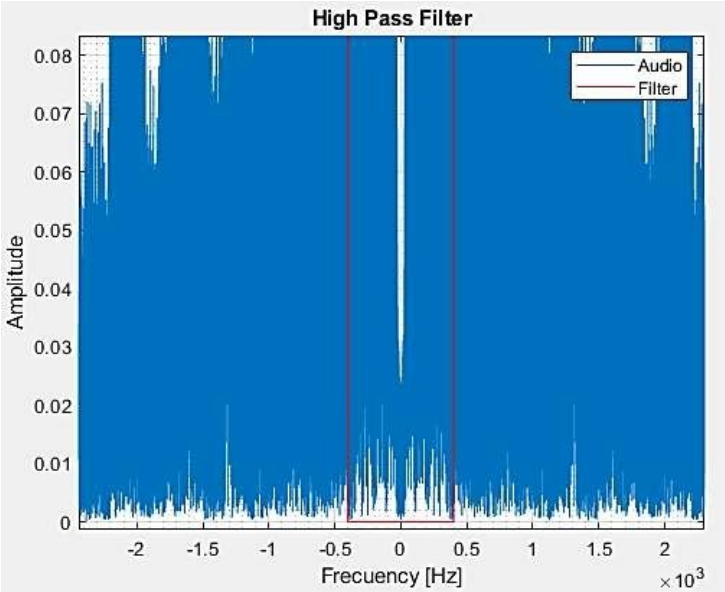


FIGURA 35. Espectro con Filtro Pasa-Altas

Para completar el proceso de filtrado, es necesario multiplicar punto a punto el vector que contiene los datos del filtro y el vector que contiene los datos del audio.

```
130 -    A_hpf = A_m .* hpf; % Filtering
```

Se procede a visualizar el filtrado completo, el cual nos permite apreciar el rechazo de la banda debajo del valor establecido.

```
131 -    figure();
132 -    plot(f, abs(A_hpf)/max(abs(A_hpf)));
133 -    hold on;
134 -    plot(f, hpf, 'r');
135 -    legend("Filtered Audio", "Filter");
136 -    title("High Pass Filter");
```

Es necesario regresar del dominio de la frecuencia al dominio del tiempo, por lo cual el paso del audio por el filtro en el tiempo será igual a la transformada inversa de Fourier del espectro obtenido después del filtrado, a través de la función integrada en Matlab *ifft*.

```
146 -    a_hpf = ifft(fftshift(A_hpf));
```

Al ejecutar la secuencia anterior el vector **a_hpf** resulta un valor complejo, sabiendo con ello que no es posible una señal compleja en el tiempo.

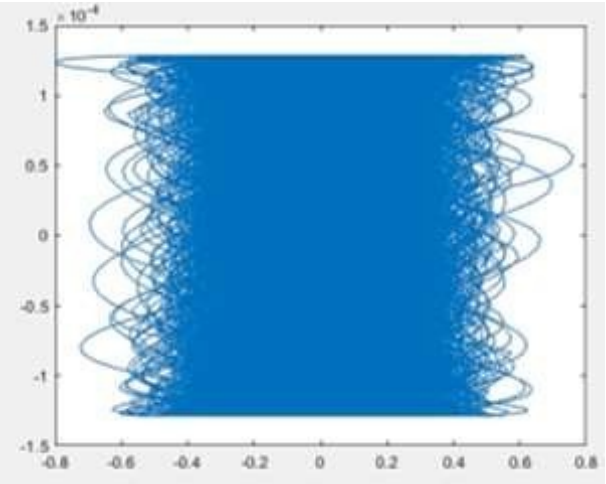


FIGURA 36. Resultado de la Transformada Inversa de Fourier después del filtrado

Es posible ver que, en el eje real la parte real está entre -0.8 y 0.8, mientras que la parte compleja oscila entre $-1.5x10^{-4}$ y $1.5x10^{-4}$. Así, la parte compleja es mucho más pequeña que la real, por lo que puede suponerse una pequeña inexactitud en los algoritmos de la transformada inversa de Fourier. Por este motivo, la parte compleja al ser muy pequeña simplemente podría despreciarse de la siguiente manera:

```
147 -    a_hpf = real(a_hpf);
```

Finalmente, para escuchar el audio original y el resultante después del filtrado, realizamos lo siguiente:

```
155 -    sound(a_hpf, fs);
156 -    pause(d+1);
157
158 -    sound(a_m, fs);
159 -    pause(d+1);
```

Filtro Butterworth Pasa-Banda

Se utiliza la herramienta de Matlab “Filter Designer” para diseñar el filtro requerido de acuerdo con las frecuencias que se desean preservar y rechazar aquellas que no.

Hay diversos parámetros que se deben configurar en la ventana emergente para el diseño del filtro. Inicialmente se indica el tipo de respuesta Pasa Banda, con un método de diseño para un filtro análogo o **Infinite Impulse Response (IIR)** a través del filtro de Butterworth.

Al saber que el rango de frecuencia deseado está entre los 400 Hz (frecuencia mínima de claxon) y hasta 2000 Hz (frecuencia máxima de una sirena de ambulancia), se toman estos valores como frecuencia inicial de paso y frecuencia terminal de paso respectivamente. Se ingresa la frecuencia menor de corte a 300 Hz y se permite disminuir hasta 2100 Hz.

En las especificaciones de la magnitud se mantienen las atenuaciones predeterminadas por Matlab. Con estos valores procedemos al Diseño del Filtro, el cual al crearse nos indicará el orden que Matlab calcula adecuado para el filtro.

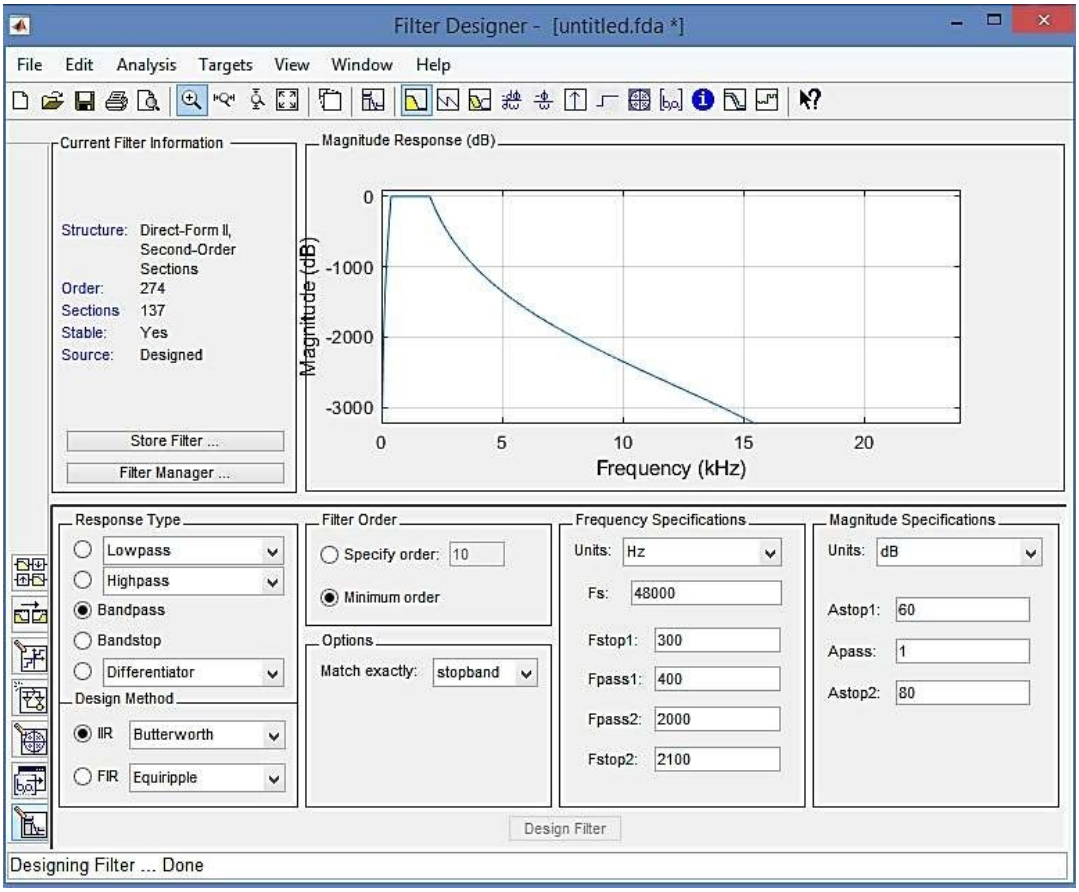


FIGURA 37. Configuración de diseño de filtro Butterworth Pasa-Banda

Se comprueba que en alguna de las frecuencias de corte se encuentren los 60 u 80 dB de la magnitud. Dado a que la exactitud del filtro está dada en la banda de corte, iremos a la

frecuencia de corte final (2100 Hz) que deberá corresponder a los 80 dB, esto simplemente para corroborar la exactitud del filtro diseñado.

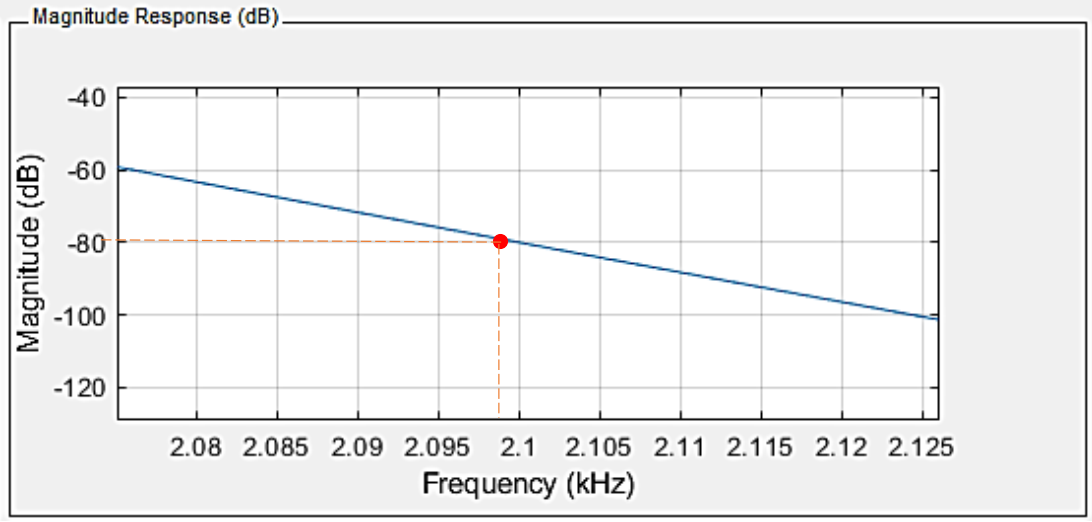


FIGURA 38. Frecuencia de corte superior con su magnitud correspondiente para corroborar exactitud del filtro diseñado

Se exporta el diseño del filtro como un objeto, para su utilización posterior.

FIGURA 39. Exportación del filtro diseñado

De esta forma se realiza el filtrado completo de la señal, por un lado, los filtros ideales pasa-bajas y pasa-altas cortan tajantemente la banda de rechazo, mientras que el filtro Butterworth pasa-banda va atenuando la banda de rechazo sin ser tan agresivo el corte, lo que para esta aplicación conviene. Se muestra el código completo del filtrado como un anexo al reporte (ver **Apéndice I**).

4.5 Módulo de Clasificación

A través de Python en el entorno de desarrollo de Jupyter se clasifican e identifican los sonidos ambientales, por lo que se emplean algoritmos de aprendizaje automático para llevar a cabo el tratamiento de archivos de audio. Este tipo de algoritmos se basan en la predicción y clasificación de datos, lo que hace que la obtención de información útil se vuelva una tarea relativamente fácil.

La complejidad de trabajar con sonidos radica en la forma en que los seres humanos lo perciben, ya que se debe analizar y tomar en cuenta la intensidad, así como las propiedades espectrales y temporales; finalmente, la caracterización de la señal debe englobar toda esa información de la misma forma en que lo percibe el ser humano.

Las señales de audio se recogen como muestras de amplitudes en función del tiempo, y se visualizan como secuencias consecutivas de valores. El dominio de la frecuencia permite identificar qué 'tonos' (frecuencias) componen una señal de audio, es decir, se obtiene la forma en que cambian los tonos a través del tiempo; como no se trata de una frecuencia periódica, está limitada en el tiempo.

Por ello se ha empleado un modelo de redes neuronales convolucionales (CNN), lo que nos permite dotar al ordenador con una capacidad de observación.

FIGURA 40. Flujo de trabajo para aprendizaje automático

Exploración de Datos

Una vez obtenida y cargada la base de datos a emplear (UrbanSound8K) se observó que esta cuenta con diez carpetas que contienen muestras de audio en formato .wav; así como una carpeta de metadatos (UrbanSound8K.csv) con información para cada archivo de audio.

Posteriormente, se comienza con un análisis exploratorio básico de los datos de audio. De manera inicial se instalan y cargan los paquetes faltantes (Keras, Librosa, SciKit-Learn) e

importar las librerías a emplear, así como cargar el *dataset,* todo lo anterior necesario para poder llevar a cabo el análisis de los datos.

Para la lectura del archivo de datos se ejecuta el siguiente código para lograr la importación del *dataset*.

```
In [6]: #Lectura del archivo de datos
        sonidos=pd.read_csv('dataset_sounds/UrbanSound8K.csv')
        print(sonidos.shape)

        (8732, 8)
```

El preprocesado de datos dentro de este módulo es una parte fundamental del diseño de modelos de aprendizaje automático y **Librosa** nos permite llevar a cabo el análisis de archivos de audio y de esta forma extraer las características de los datos de audio.

Se procede a cargar una carpeta de datos para análisis, cada carpeta cuenta con alrededor de 870 muestras.
A continuación, se requiere leer los datos de audio en memoria para obtener su frecuencia de muestreo original (número de muestras de audio transportadas por segundo, medido en Hz), se realiza esto ya que existen archivos de poca longitud.

Las señales de audio deben ser transformadas por un filtro pasa bajo para convertir la frecuencia de muestreo a 22.05 kHz que maneja Librosa de forma predeterminada y así reducir el número de canales a 1 (mono) para que de esta forma, los datos se normalicen, es decir, para que oscilen entre -1 y 1.

Extracción de características

Existe una técnica interesante para poder visualizar los datos de audio, esta es la generación de espectrogramas. Con esta técnica, a partir de los datos de audio, podemos hacer una representación visual de los espectros de frecuencias. Para el proyecto, usaremos una técnica muy similar conocida como Coeficientes Cepstrales en las Frecuencias de Mel (MFCC). La principal diferencia es que un espectrograma usa una escala de frecuencia espaciada lineal (por

lo que cada intervalo de frecuencia está espaciado a un número igual de Hertz), mientras que un MFCC usa una escala de frecuencia espaciada cuasi-logarítmica, que es más similar al modo en que el sistema auditivo humano procesa sonidos. Para esto, necesitamos de una función que sea capaz de convertir dichas señales entre el dominio del tiempo y el dominio de la frecuencia.

La "Transformada de Fourier", nuevamente es la función que nos permite realizar la transformación matemática y nos ayuda a extraer el espectro de frecuencias para poder obtener las características de las señales. Entendiendo en esta parte, que la Transformada de Fourier realizada en el módulo de filtrado fue propia para ese proceso en específico, y al tener que regresar la señal filtrada al dominio del tiempo, es necesario nuevamente realizar esta conversión, pero para un fin distinto.

Para cada archivo de audio en el conjunto de datos se debe extraer un MFCC, es decir, se debe representar en imagen cada muestra de audio.

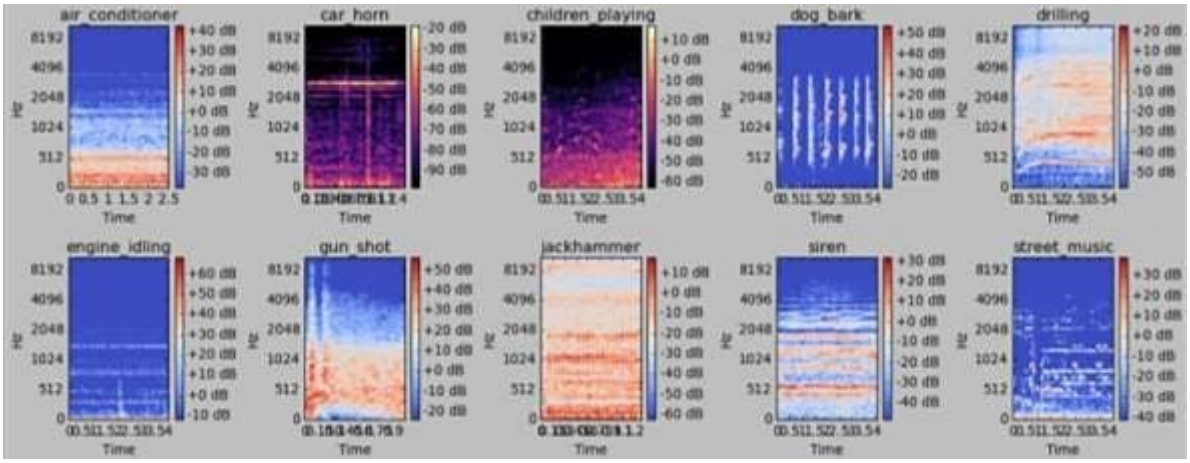


FIGURA 41. MFCC de las muestras de audio tomadas

Con la escala de Mel, los espectrogramas son mucho más fácil de distinguir.

A partir de los datos de audio sin procesar se deben extraer características sólidas para poder construir el modelo de clasificación.

Extracción de características del *dataset* completo

Una vez cargado el conjunto de datos, se extraen las características de cada muestra (8732 muestras de audio). Se realiza de esta forma, la siguiente secuencia para la extracción de tales características.

```
feature = []
label = []
# Funcion para cargar los archivos y extraer caracteristicas
def parser(row):
    for i in range(8732):
        file_name = 'dataset_sounds/fold' + str(sonidos["fold"][i]) + '/' + sonidos["slice_file_name"][i]
        #kaiser_fast es una técnica usada para la extración rápida de características
        X, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
        # Extraer la función mfcc de los datos
        mels = np.mean(librosa.feature.melspectrogram(y=X, sr=sample_rate).T,axis=0)
        feature.append(mels)
        label.append(sonidos["classID"][i])
    return [feature, label]

temp = parser(sonidos)
```

Construcción del modelo de clasificación

Para llevar a cabo la construcción del modelo clasificador, es necesario dividir el *dataset* original en un conjunto de entrenamiento y un conjunto de test. Esta división es necesaria ya que el modelo debe ser probado para saber si el algoritmo puede llegar a funcionar con datos que no formen parte del *dataset* original.

Se divide la información ejecutando el código siguiente y se utiliza el 75% de los datos para crear un modelo (fase de entrenamiento del algoritmo) para entrenar a la máquina y así sea capaz de aprender los rasgos generales, las relaciones entre las variables independientes y un modelo que sea capaz de predecir la variable dependiente.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state = 1)

#Fase de entrenamiento
X_train = X_train.reshape(6549, 16, 8, 1)
#Fase de testing para evaluar el modelo
X_test = X_test.reshape(2183, 16, 8, 1)
```

Posteriormente se prueba el algoritmo, utilizando el 25% restante para evaluar (fase de testing). De esta manera podrá observarse el rendimiento con datos que no se han utilizado para entrenar,

es decir, los que formen parte del conjunto de test. Esta fase de prueba asegura que cualquier señal de entrada puede ser ingresada a la red sin tener que pertenecer al conjunto de datos previamente cargados, de este modo podrán ingresarse los datos de la señal obtenida después del módulo de filtrado y así facilitar el trabajo de la red.

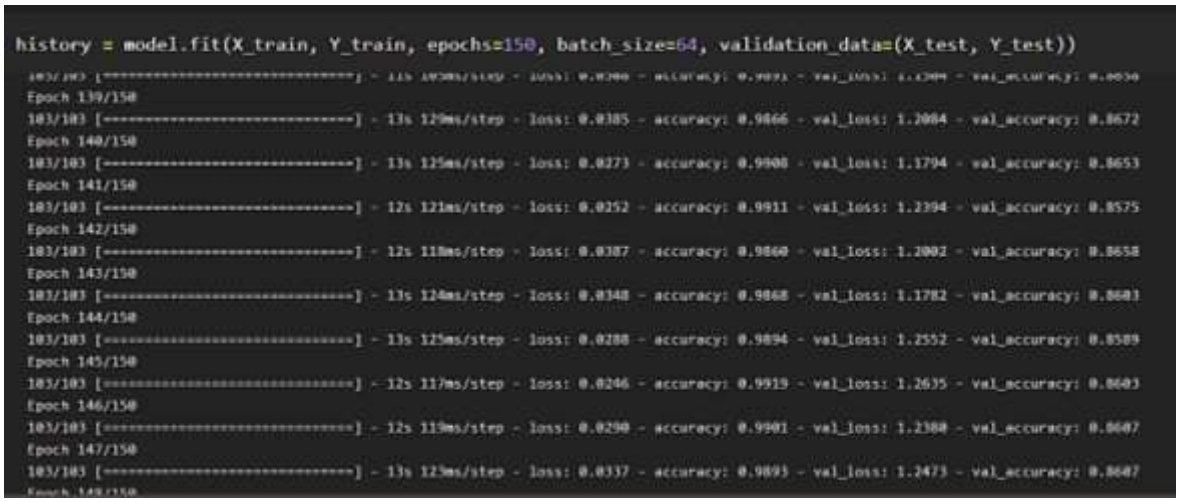
Ahora se construye el clasificador de aprendizaje profundo utilizando una red completamente conectada que tiene cuatro capas ocultas. Para esto se utilizan los componentes habituales como la *deserción* para evitar el sobreajuste y el *optimizador de Adam* con parámetros predeterminados para el modelo. Los detalles de la arquitectura del modelo se muestran en el siguiente código:

```
#add
#Método para agregar capas al modelo
#Capa NN densamente conectada
model.add(Conv2D(64, (3, 3), padding = "same", activation = "tanh", input_shape = input_dim))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), padding = "same", activation = "tanh"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.1))
model.add(Flatten())
model.add(Dense(1024, activation = "tanh"))
model.add(Dense(10, activation = "softmax"))

#Optimizador con parametros predeterminados
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

Proceso de formación y entrenamiento del modelo

El siguiente paso es entrenar el modelo. Este modelo fue entrenado durante 150 épocas con un tamaño de lote de 64. Estos parámetros pueden ser ajustados para obtener un modelo más robusto, pero debe ser un procedimiento cuidadoso para evitar el sobreajuste o bajo ajuste de la red. En seguida se muestra el debido proceso de entrenamiento.



Con lo anterior se muestra que la red programada tiene una precisión de validación cercana al 87%, lo cual indica un buen rendimiento en conjunto (ver **Apéndice III**).

4.6 Módulo de Conversión y Respuesta

Después del filtrado y clasificación del sonido, es menester la conversión de las frecuencias obtenidas a los dos modos de salida requeridos: visual y táctil. Por lo que se recurre a una conversión analógica-digital (ADC), a través de la placa Arduino Uno.

Una de las características claves del convertidor, es su número de bits, que define la resolución con la que se puede cuantificar esa conversión a digital. En el caso de Arduino, son 10 los bits que maneja en las puertas analógicas, lo que significa que su resolución es 1024 posibles valores. Cuanto mayor sea esta resolución mejor es la capacidad de aproximación al valor real cuya conversión buscamos. Así, el rango de frecuencias de entrada corresponderá a un valor entre 0 y 1024.

El ADC realizará la comparación sucesiva de la señal que queremos cuantificar en la entrada, con una tensión de referencia contra la que hace las comparaciones. No proporcionará valores absolutos, sino una comparación cuantificada con relación a un valor de referencia.

En este caso, se ha decidido trabajar con Arduino Uno puesto que su microcontrolador ATMega328p, al igual que toda la gama ATMega de Atmel y otros microcontroladores, tienen un ADC integrado y no necesita ningún hardware adicional, esto nos permite simplemente conectar una entrada analógica. Lo anterior en comparación con una Raspberry Pi que necesita de un ADC externo como el MCP3008.

Aunque el ATmega328P tiene 6 pines que son capaces de ser utilizados como pines de entrada analógicos (Port C), sólo hay un ADC en el microcontrolador, pero entre el ADC y los pines hay un multiplexor analógico, esto permite que podamos elegir qué pin está conectado al ADC, es decir, aunque podemos utilizar todos los pines, sólo se puede leer el valor de uno de ellos a la vez lo cual será suficiente para la entrada analógica de frecuencias que recibe.

Aunque la configuración del ADC en Arduino está previamente establecida, se muestran los registros que se configuran en el microcontrolador.

Primero debe darse un reloj al ADC para establecer la velocidad de muestreo, pero esta tiene un máximo de 200kHz, este valor se establece mediante los *pre-scaler*. Estos son configurados por

el registro ADCSRA y los bits relacionados que son ADPS2, ADPS1 and ADPS0 con valores de pre-scaler de 2 a 128. Como el microcontrolador funciona a 16 MHz se usará el prescaler 128 para que el reloj del ADC funcione a 125 KHz (ver **Anexo III**).

El siguiente paso es configurar el voltaje de referencia usado por el ADC, en este caso es 5V. Este valor de voltaje de referencia es configurado en el registro ADMUX con los bits REFS1 y REFS0. En este caso es el valor por defecto y no hace falta modificar.

El ADC está casi configurado, solo hace falta iniciarlo (por defecto está apagado para consumir menos) y para ello se debe poner a 1 el bit ADEN del registro ADCSRA y luego poner a 1 el bit ADSC para comenzar la conversión en el mismo registro. En el registro ADCSRB los bits ADTS2, ADTS1 and ADTS0 determinan como una nueva conversión comienza, por defecto está en mode free running. También debe ponerse a 1 el bit ADATE para que en modo free running comience la conversión y active el auto trigger porque solo estamos leyendo de un pin analógico.

Ahora solo queda leer el valor devuelto por el ADC en los registros **ADCH** y **ADCL.**

Se anexa el código en Atmel Studio para la configuración de los registros para el ADC del ATmega328p (ver **Apéndice II**).

Aunque pareciera sencillo, es necesario sintetizar los valores de conversión del ADC y su proporcional salida PWM tomando en cuenta sus respectivos rangos para entender el modo en que se realiza la conversión.

TABLA 4. Valores proporcionales del ADC con el PWM

%	ADC	PWM
0	0	0
1	10.23	2.55
2	20.46	5.1
3	30.69	7.65
⋮	⋮	⋮
98	1002.54	249.9
99	1012.77	252.45
100	1023	255

Así, se denota la linealidad de correspondencia en la que opera de manera interna el PWM en respuesta al ADC.

SEÑAL VISUAL

Para poder enviar una señal de salida visual, se han utilizado luces LED que trabajan a 12V, las cuales serán adaptadas a una funda de volante, lo que permitirá al conductor una fácil visualización. Al obtener la conversión de la señal analógica de las frecuencias a valores entre 0-1024, se procederá a utilizar las salidas con PWM, la cual se encargará de regular la tensión de salida con valores de entre 0-255 y de esta forma aumentar o disminuir la intensidad de las luces LED.

Debido a que la intensidad de luz en un LED es proporcional a la intensidad de corriente que le recorre, se recurre a un transistor para controlar su flujo.

Para ello utilizaremos un Darlington TIP120 que es un transistor NPN de potencia, que soporta hasta 5 Amperios y una tensión de 60V. Es el transistor ideal por sus características para que el Arduino controle los LEDs con una mínima carga para nuestro microcontrolador (ver **Anexo IV**).

Entonces, el transistor se encarga de controlar la carga que pasa a la tira de LEDs mientras Arduino con un pin PWM regula el paso mediante la señal que entrega en la base del transistor.

Circuito de control de tira de LEDS

Vamos en primer lugar con el circuito correspondiente al transistor de potencia media, el TIP 120, cuya base gobernaremos mediante el pin digital 9 de Arduino.

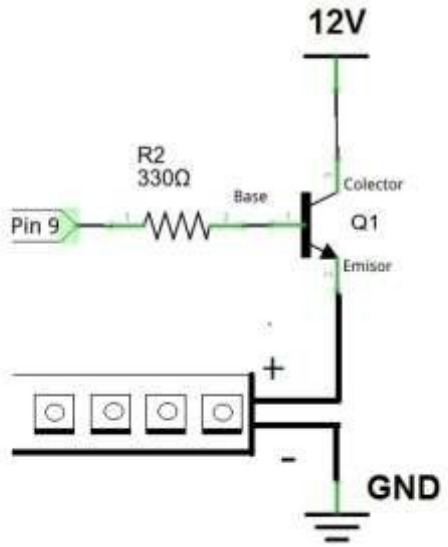


FIGURA 42. Circuito de control de tira LED

El transistor TIP 120, regula el flujo de corriente a través de 12V y por la tira LED en función de la tensión que pongamos en su base, desde Saturación (pasa todo) con 5V en la base y a corte (no pasa nada) cuando ponemos 0V en la base.

La lectura a través de la conversión analógica-digital en el pin analógico A0, con su proporcional salida PWM en el pin 9 de Arduino define cuanta tensión es dada a la base del transistor.

SEÑAL TÁCTIL

Para cumplir con la respuesta en vibración de nuestro proyecto, utilizaremos un motor DC, el cual es similar al funcionamiento del motor para vibración de un teléfono celular.

Para alcanzar la vibración, en el eje del motor se le pone una carga metálica semi-cilíndrica. Esta carga hace que cuando el motor gire, se genere un cabeceo de la carga con lo cual se produce la vibración. Aprovecharemos la misma lectura de la señal analógica de entrada en A0, para obtener un PWM proporcional en el pin 3 de Arduino.

Circuito de control de motor

Debido a que Arduino proporciona una corriente de salida de 50mA y un motor DC estándar consume unos 1000mA, se utiliza un controlador de motores. Este presenta un comportamiento similar al de un interruptor, cuando se cierra deja circular corriente a través de sí. Sólo que, en vez de activarse manualmente, es activado mediante un microcontrolador y además es capaz de regular la velocidad del motor. Para este fin en particular, se ha ocupado el transistor NPN 2N2222 que por sus características nos permitirá poder controlar la velocidad de giro del motor (ver **Anexo IV**).

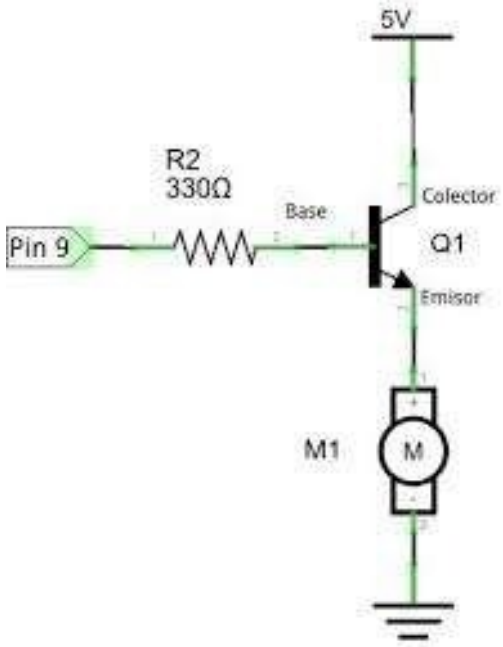


FIGURA 43. Circuito de control de motor

CIRCUITO GENERAL

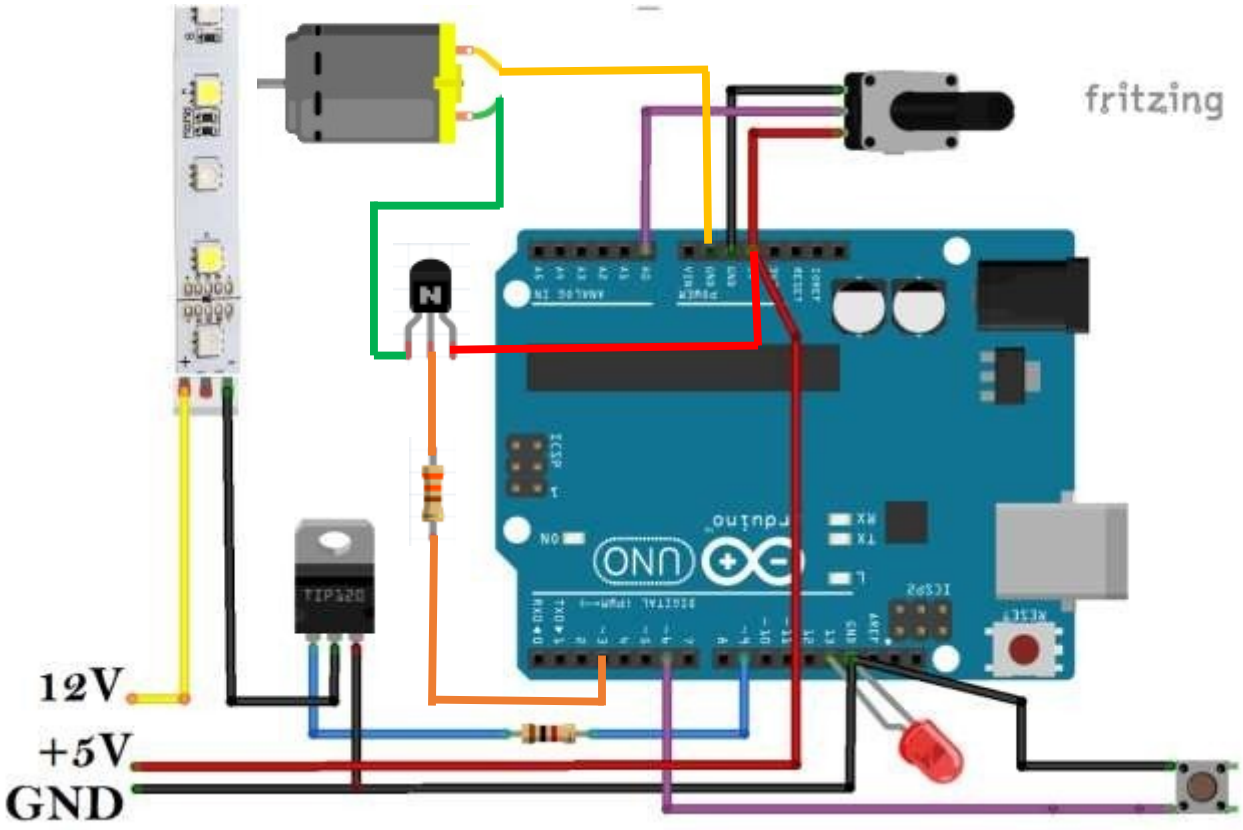


FIGURA 44. Diagrama de conexión general de respuesta

PROGRAMA DE CONTROL

```
#include "arduinoFFT.h" // Realiza la Transformada Rápida de Fourier

uint16_t j = 0;
double vReal []; // Tensión Real
double vImag []; // Tensión Imaginaria

arduinoFFT FFT = arduinoFFT(A0); // Configuración para Transformada Rápida de Fourier
MotorControl mc = MotorControl(3); // Configuración del Control de Motor
LedControl lc = LedControl(9); // Configuración del Control de Tira de Led

void setup(){
   Serial.begin(9600);
   pinMode(FFT, INPUT); // Entrada análoga de frecuencias
   pinMode(mc, OUTPUT); // Pin PWM para control de Velocidad de Motor
   pinMode(lc, OUTPUT) // Pin PWM para control de Intensidad de Leds
   delay(1000); // Retraso para estalizar tensión
}

void loop(){
   int lectura = analogRead(FFT); // Lectura del muestreo

   for(int i=0; i<lectura; i++){
      vReal[i] = analogRead(FFT); // Lectura de señal de entrada en arreglo de 0-1023
      vImag[i] = 0; // Despreciando parte imaginaria
   }

   FFT.Windowing(vReal, lectura, FFT_WIN_TYP_RECTANGLE, FFT_FORWARD); // Configuración de
ventana
   FFT.Compute(vReal, vImag, lectura, FFT_FORWARD); // Configuración de procesamiento
   FFT.ComplexToMagnitude(vReal, vImag, samples); // Transformada Z para conversión de
magnitud compleja a real

   j = lectura; // Frecuencia normalizada a rango 0-1023

   for(int k=0; k < 1023; k++){
      if((byte)vReal[j] <= 1023){
         analogWrite (mc, vReal[k]/4); // Salida 0-255 a motor proporcional a la entrada
         analogWrite (lc, vReal[k]/4); // Salida 0-255 a leds proporcional a la entrada
      }
   }
}
```

5. VALIDACIÓN DEL SISTEMA

5.1 Validación del funcionamiento del módulo de alimentación

Si se disminuye la resistencia de referencia (R2) a su valor mínimo (0%) se obtiene una tensión a salida fija de 1.25V, voltaje minimo de salida del circuito.

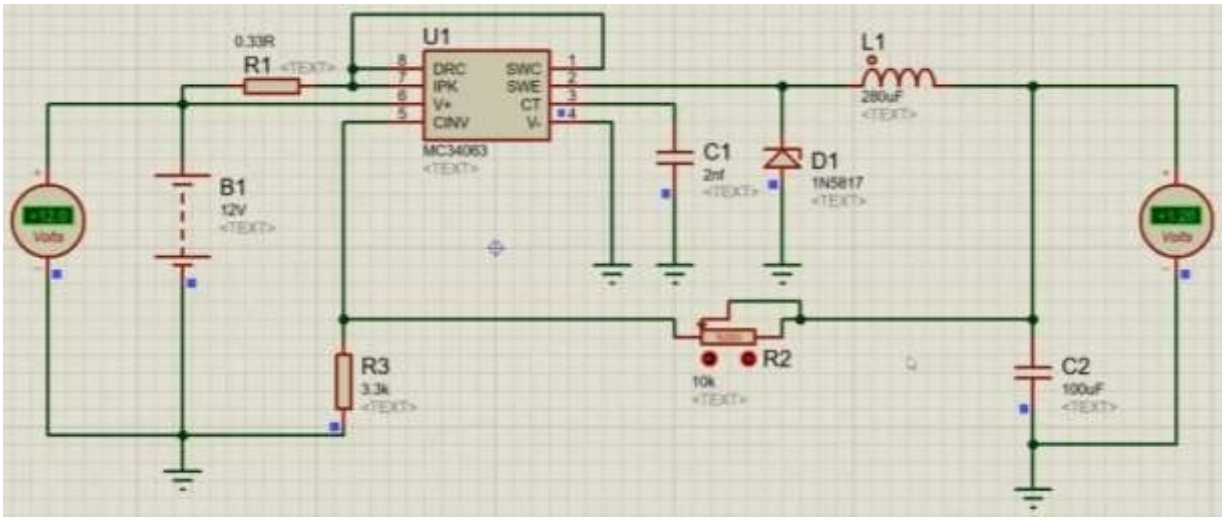


FIGURA 45. Circuito con voltaje de salida mínimo (1.25V)

Ahora bien, al regular la resistencia de referencia (R2) al 50% se obtiene una tensión a salida fija de aproximadamente 3.5 V, valor medio de salida del circuito.

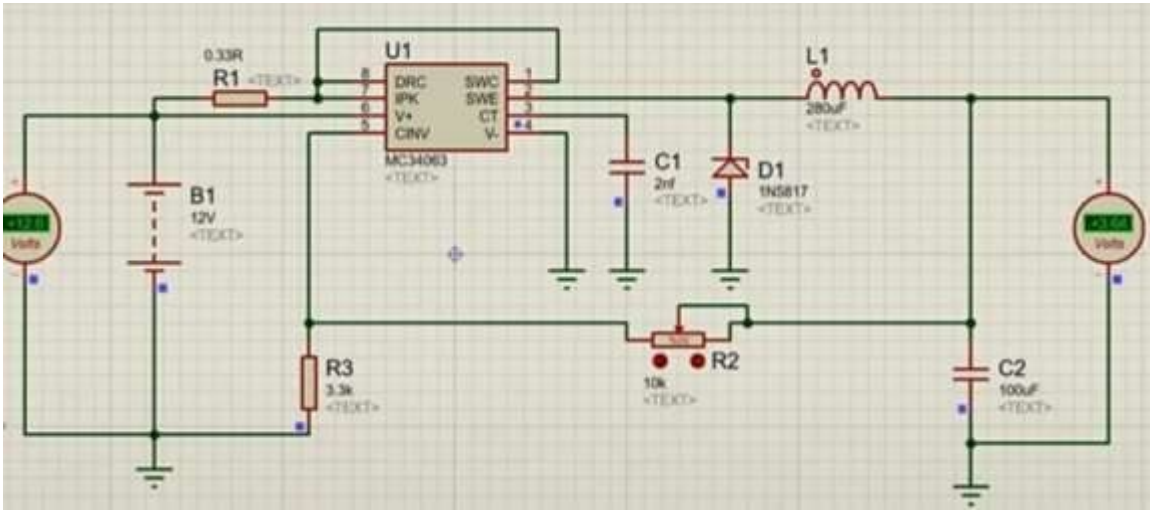


FIGURA 46. Circuito con voltaje de salida medio (3.5V)

Si elevamos la resistencia de referencia (R2) a su máximo valor (100%) se obtiene una tensión a salida fija de aproximadamente 5V, cumpliendo con el voltaje requerido por la tarjeta de desarrollo utilizada para la obtención de la señal de entrada del módulo siguiente.

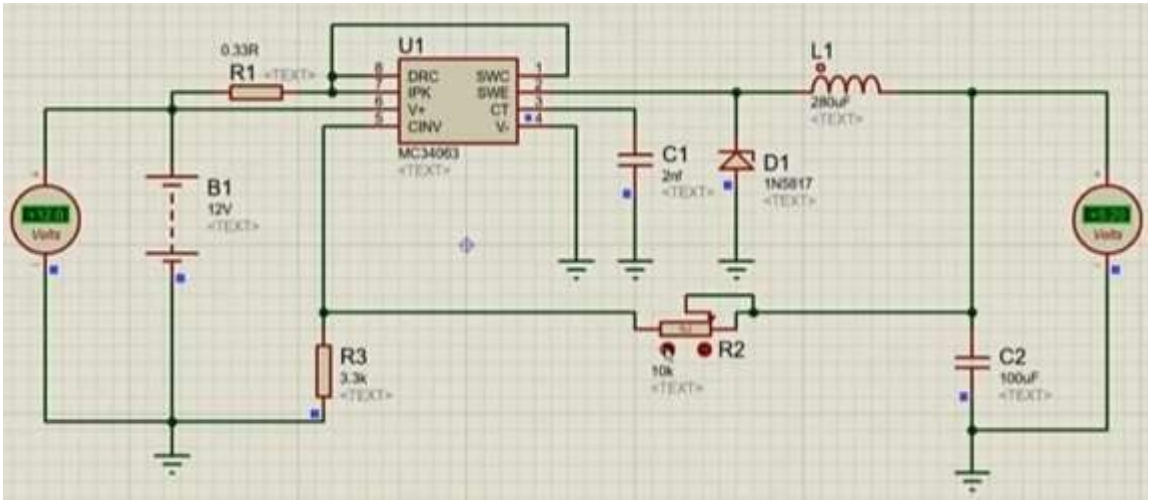


FIGURA 47. Circuito con voltaje de salida máximo (5V)

5.2 Validación del funcionamiento del módulo recepción

Después de haber realizado el proceso inicial para la recepción de señales de audio, utilizando un micrófono conectado a la Raspberry Pi como instrumento y realizando la configuración en Matlab para la captación de este, se visualiza el espectro del audio recibido en un monitor alterno que permite constatar que el módulo de recepción ha funcionado.

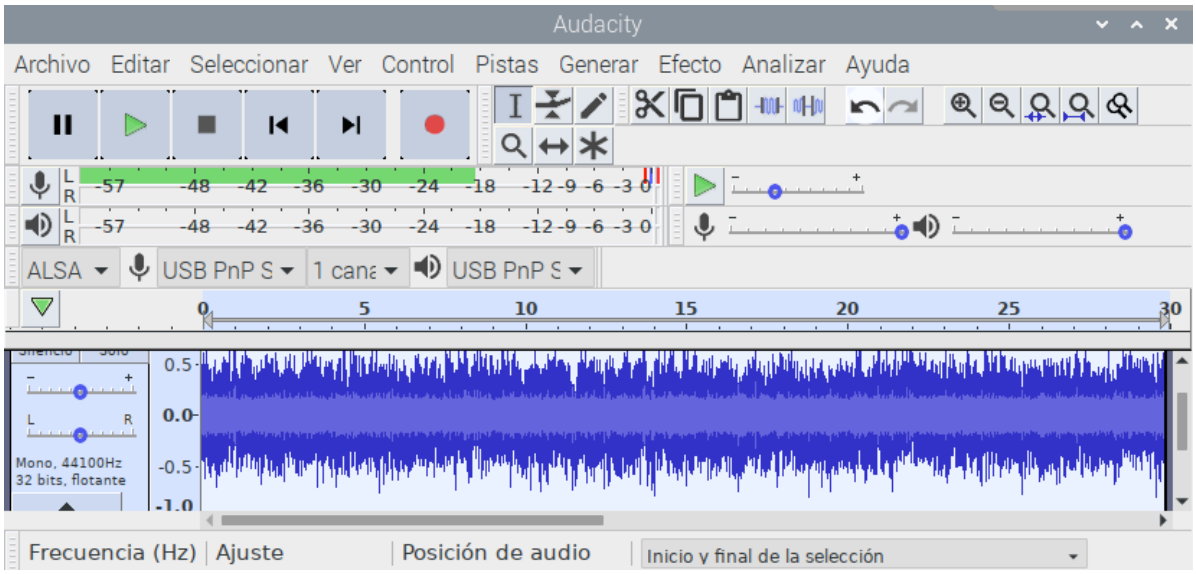


FIGURA 48. Señal de audio recibido a través de la Raspberry al ordenador principal

Cabe destacar que tal señal de entrada necesita ser guardado como un archivo, que en este caso será del tipo .wav, el cual brinda características que resultan convenientes para esta aplicación. Lo anterior es requerido por Matlab para poder realizar el análisis de la señal y los debidos procesos consiguientes.

5.3 Validación del funcionamiento del módulo de preprocesamiento

A través del análisis de Fourier de la función periódica, otorgada por la forma de onda del audio recibido se han extraido las series de senos y cosenos que cuando se superponen, reproducen la función original. La transformada rápida de Fourier (FFT) como método matemático ha permitido la transformación de una función del tiempo en una función de frecuencia, descrita como la transformación del dominio del tiempo al dominio de frecuencia. Esto es muy útil para el análisis de los fenómenos dependientes del tiempo, lo cual nos abre el camino para la implementación de los filtros requeridos.

Así, la FFT desarrollada de manera automática por los algoritmos de Matlab permite una representación de la distribución de energía sonora del sonido recibido en función de la frecuencia. Este espectro es importante porque la percepción auditiva del sonido es de naturaleza predominantemente espectral.

Dependiendo del tamaño de la ventana que se utiliza para el análisis de Fourier tendremos diferentes niveles de resolución del espectrograma. Al aplicar una ventana muy grande se obtiene un espectrograma muy detallado, pero a costa de incrementar el tiempo de cálculo necesario para esta operación. Para el caso de una ventana demasiado pequeña el efecto es el inverso y no seremos capaces de distinguir los diferentes armónicos si están muy juntos en el espectrograma. Por ello se ha aplicado una ventana estándar que nos permita una atinada visualización sin tener que llegar a ninguno de los extremos.

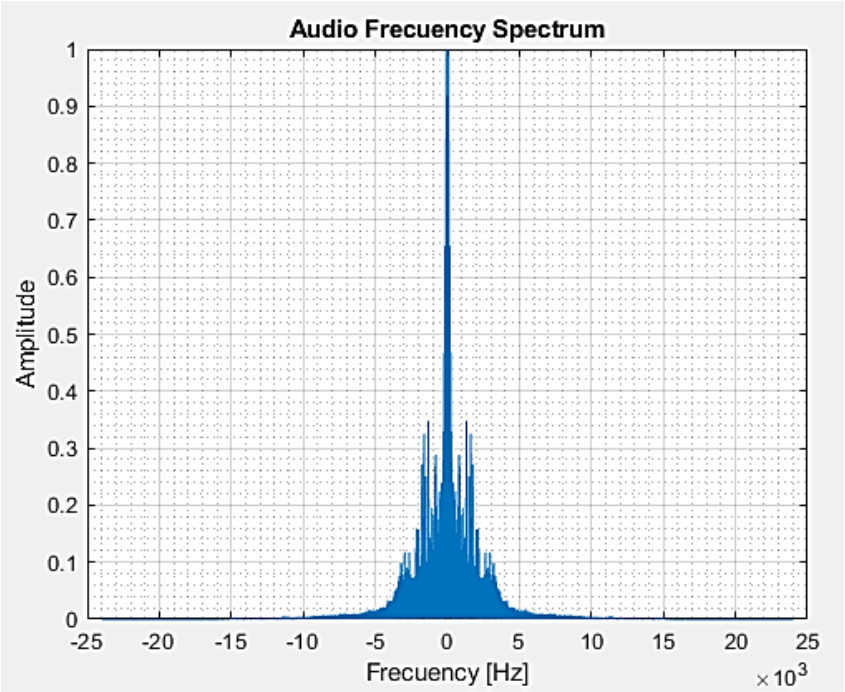


FIGURA 49. Espectro de frecuencia de la señal de entrada

Al ejecutar el programa de preprocesamiento y filtrado, se obtienen los valores de las variables inmersas en el programa, con su valor, número de vías y tipo de dato. Lo anterior para verificar el cambio de las variables de acuerdo con la secuencia en las que se ejecutan.

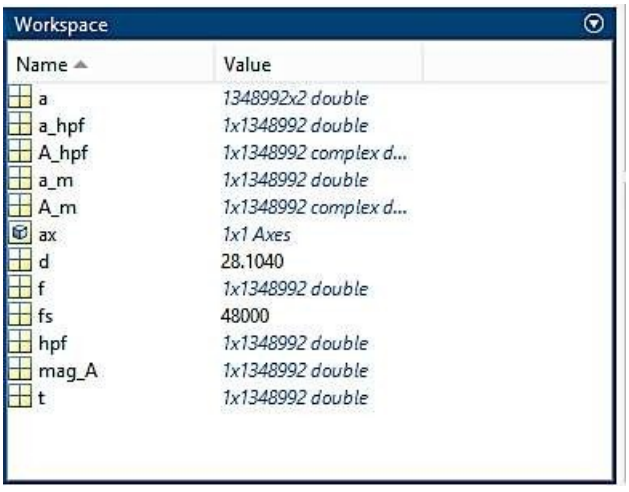


FIGURA 50. Variables tras la ejecución del programa en Matlab

Puede visualizarse la utilización de cada una de las variables dentro del programa anexo a este proyecto (Apéndice C).

5.4 Validación del funcionamiento del módulo de filtrado

El filtro pasa-bajas implementado ha realizado el primer paso de este módulo: rechazar o atenuar las señales por encima de los 2000 Hz y dar paso a aquellas frecuencias por debajo de esta frecuencia de corte indicada.

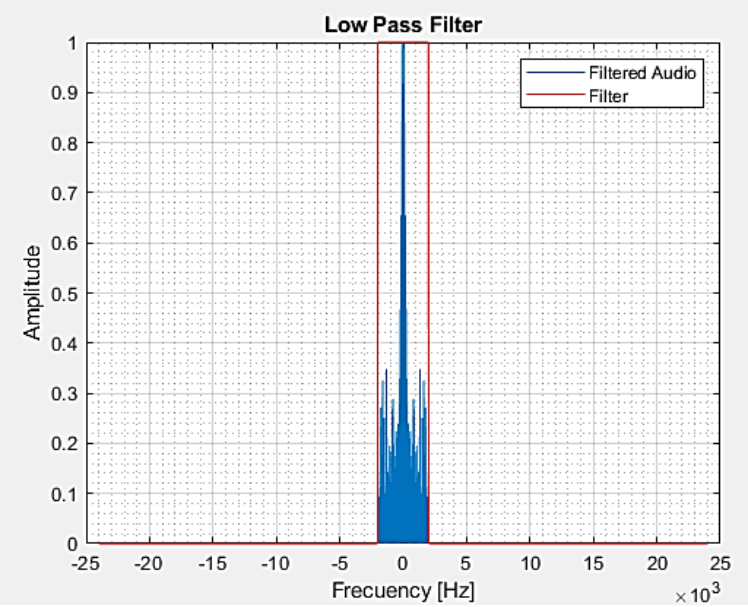


FIGURA 51. Filtro Pasa-Bajas completo

Por consiguiente, el filtro pasa-altas realiza la segunda parte de este proceso, eliminar aquellas frecuencias por debajo de los 400 Hz.

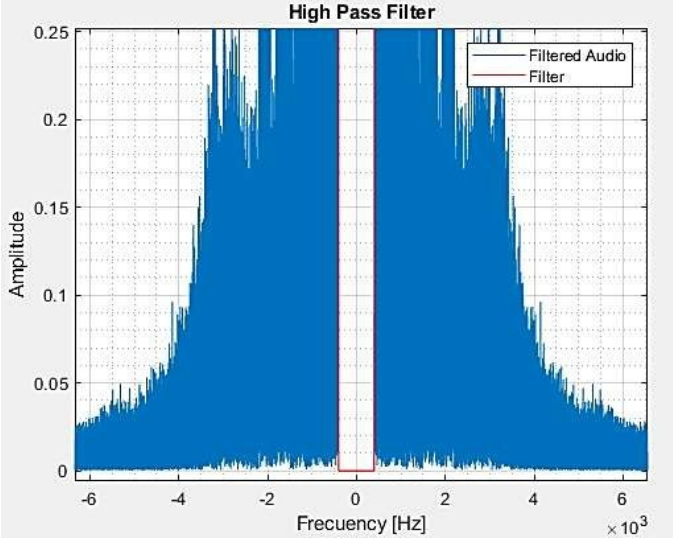


FIGURA 52. Filtro Pasa-Altas completo

Por otro lado, para el diseño del filtro Butterworth pasa-banda e ingresando los valores de las frecuencias de corte y frecuencias de paso, se muestra la forma de respuesta del mismo.

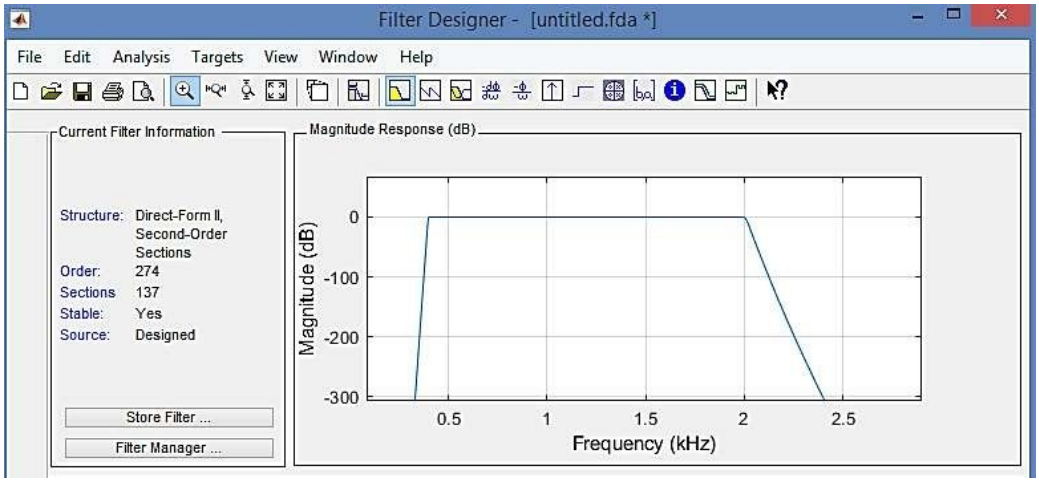


FIGURA 53. Forma del filtro Butterworth Pasa-Banda diseñado

La respuesta del filtro Butterworth arroja un buen comportamiento en general. Presenta una máxima uniformidad de amplitud en la respuesta en frecuencia en la banda de paso. Y muestra una mejor respuesta a entrada de pulsos que Chebyshev y una mejor pendiente de atenuación que Bessel, por ello la elección de este filtro para esta aplicación en particular.

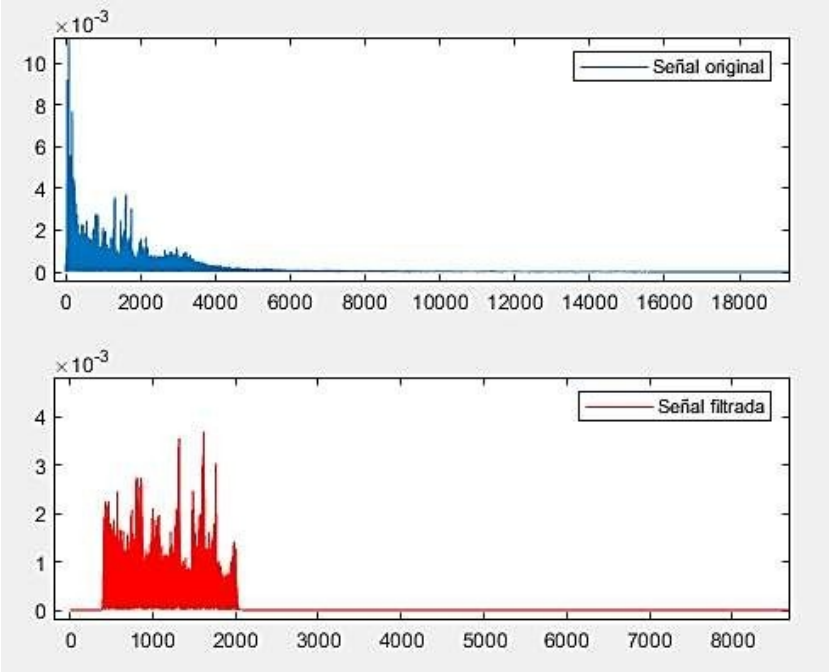


FIGURA 54. Filtro Butterworth Pasa-Banda completo

5.5 Validación de funcionamiento del módulo de clasificación

Inicialmente es posible la visualización de la distribución de datos de cada categoría.

	index	jackhammer	children_playing	dog_bark	drilling	street_music	air_conditioner	engine_idling	siren	car_horn	gun_
0	fold1	120	100	100	100	100	100	96	86	36	35
1	fold2	120	100	100	100	100	100	100	91	42	35
2	fold3	120	100	100	100	100	100	107	119	43	36
3	fold4	120	100	100	100	100	100	107	166	59	38
4	fold5	120	100	100	100	100	100	107	71	98	40
5	fold6	68	100	100	100	100	100	107	74	28	46
6	fold7	76	100	100	100	100	100	106	77	28	51
7	fold8	78	100	100	100	100	100	88	80	30	30
8	fold9	82	100	100	100	100	100	89	82	32	31
9	fold10	96	100	100	100	100	100	93	83	33	32

FIGURA 55. Distribución de datos en el dataset

A través de la forma de onda es posible visualizar las muestras de audio recabadas.

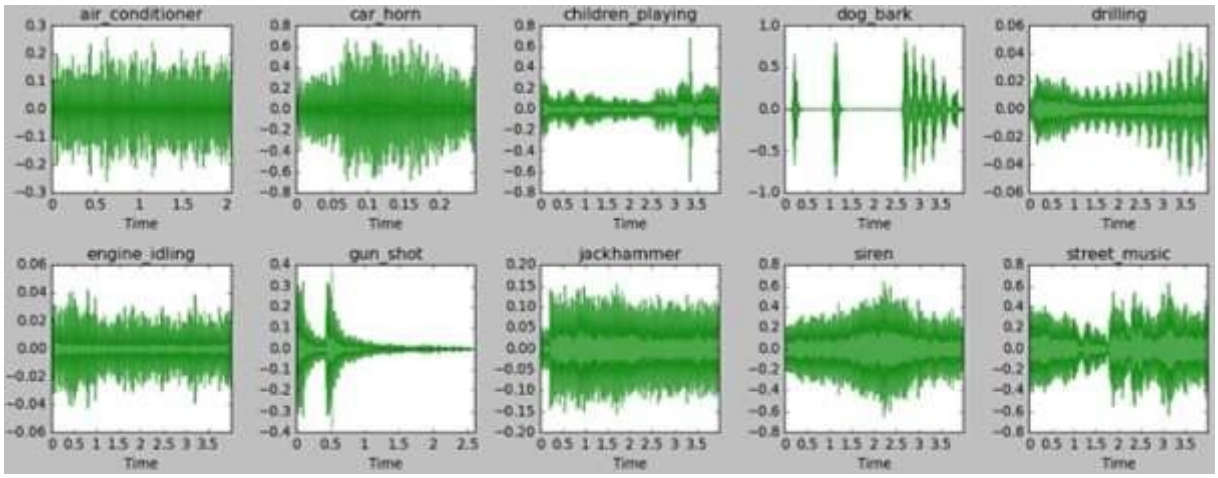


FIGURA 56. Forma de onda tomadas de las muestras de audio

A través de la forma de onda sólo podemos observar el 'volumen del sonido' (variación de amplitud en el tiempo). Como se mencionó anteriormente, la tarea de identificar patrones mediante formas de onda se vuelve difícil e incluso es complicado visualizar la diferencia entre algunas de las clases.

Ahora bien, se visualizan las muestras de audio como espectrogramas:

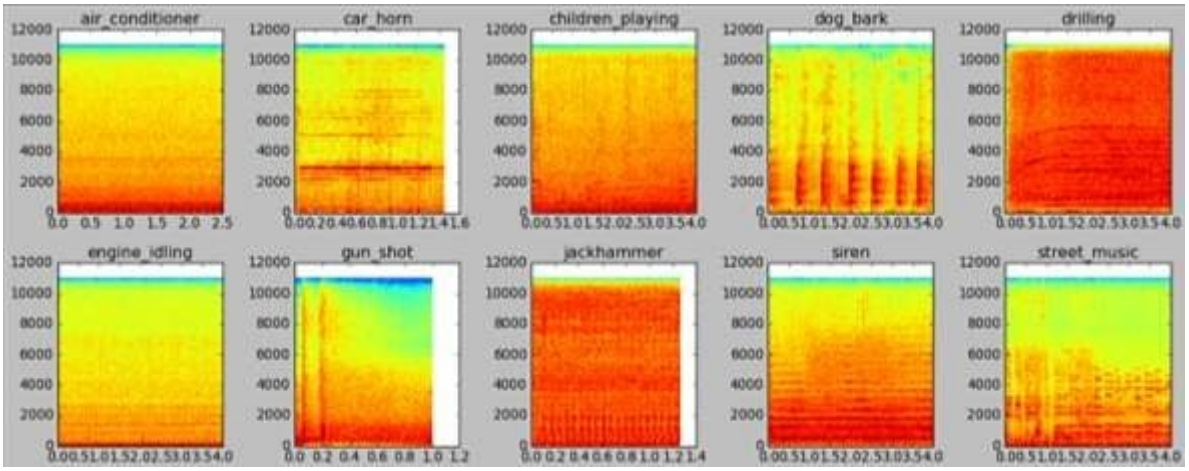


FIGURA 57. Visualización de las muestras de audio como espectrogramas

Se pueden observar algunas diferencias marcadas en los espectrogramas entre las diferentes fuentes de audio. Sin embargo, el espectrograma de Mel es mejor que el espectrograma básico, ya que su escala se basa en las comparaciones de tono. La escala Mel es una escala perceptiva de tonos.

Después de los procesos de formación y entrenamiento del modelo se traza la precisión general y los gráficos de pérdidas del modelo para poder tener un panorama visual del resultado:

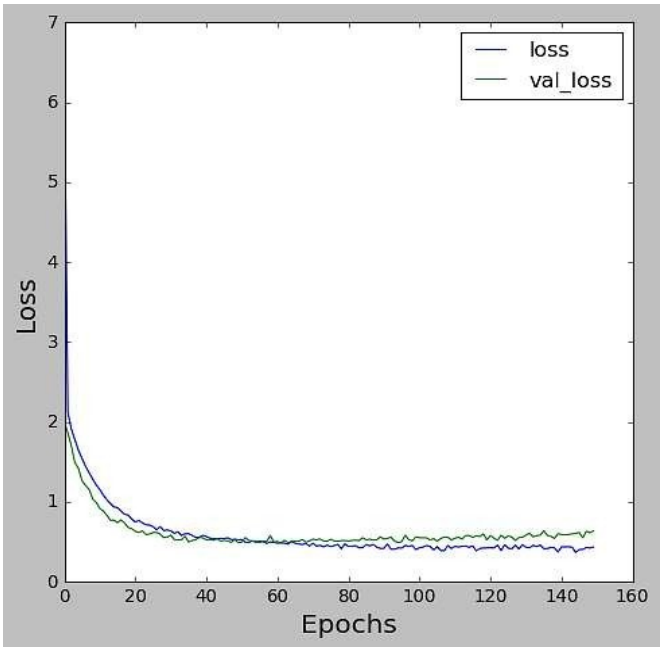


FIGURA 58. Evolución del error del modelo

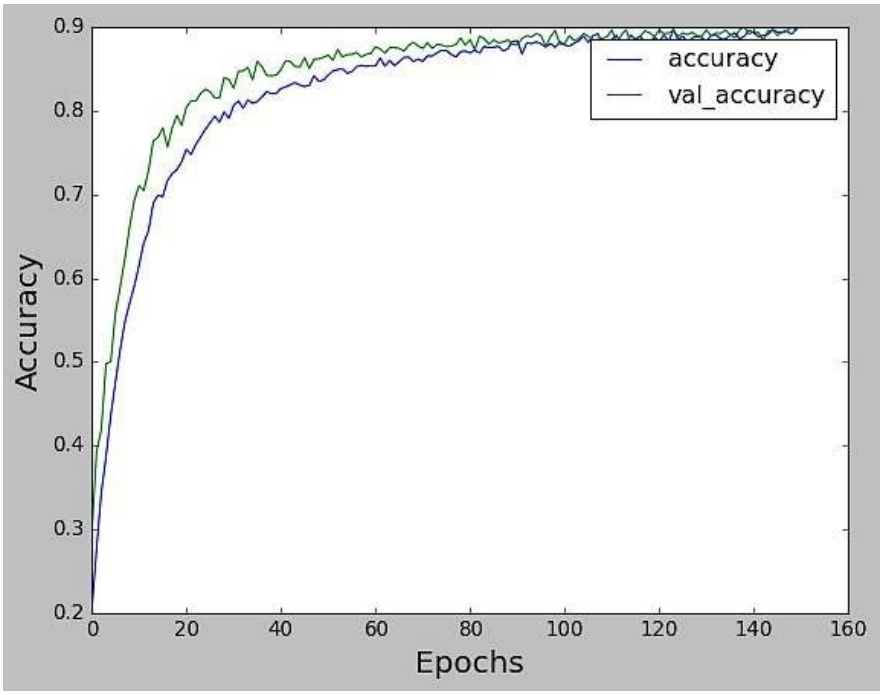


FIGURA 59. Evolución de la precisión del modelo

Se logra apreciar que la pérdida y precisión del modelo entre el entrenamiento y la validación es bastante consistente; tal vez se sobreajuste ligeramente, pero es insignificante considerando que su diferencia es bastante baja.

Es necesario poner a prueba el modelo para evaluar su desempeño en los datos de prueba, para ello deben hacerse predicciones. Para ello, se obtienen las métricas generales que ayudan a comprender dicho rendimiento.

```
Recall: [0.95723684 0.91851852 0.93793103 0.7826087  0.93493151 0.94637224
 0.81355932 0.98920863 0.92114695 0.76623377]
Precision: [0.97324415 0.98412698 0.66019417 0.92125984 0.97153025 0.95238095
 0.92307692 0.96830986 0.93115942 0.87732342]

 clasification report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       304
           1       0.98      0.92      0.95       135
           2       0.66      0.94      0.77       290
           3       0.92      0.78      0.85       299
           4       0.97      0.93      0.95       292
           5       0.95      0.95      0.95       317
           6       0.92      0.81      0.86       118
           7       0.97      0.99      0.98       278
           8       0.93      0.92      0.93       279
           9       0.88      0.77      0.82       308

    accuracy                           0.90      2620
   macro avg       0.92      0.90      0.90      2620
weighted avg       0.91      0.90      0.90      2620
```

FIGURA 60. Reporte de clasificación

Lo anterior resulta en una precisión general del modelo y una puntuación f1 de 90%, que es excelente y coherente con lo obtenido en el conjunto de datos de validación.

A continuación, se visualiza el rendimiento del modelo por clase, el cual se ha graficado en una matriz de confusión.

```
confussion matrix:
[[291   0   7   0   0   3   0   1   0   2]
 [  0 124   4   0   0   2   0   0   2   3]
 [  0   1 272   2   0   2   3   0   2   8]
 [  1   0  36 234   2   1   4   0  10  11]
 [  0   0   8   0 273   0   0   6   0   5]
 [  1   0   8   2   1 300   0   0   4   1]
 [  0   0  12   7   0   1  96   1   0   1]
 [  0   0   0   0   3   0   0 275   0   0]
 [  1   0  14   4   0   1   0   0 257   2]
 [  5   1  51   5   2   5   1   1   1 236]]
```

FIGURA 61. Rendimiento del modelo

En una matriz estandarizada se puede observar de mejor manera el comportamiento en cuanto a desempeño del modelo clasificador. Esto nos ayudó para entrenar y ajustar el modelo y de esta forma poder corregir los errores en donde hay menor precisión en las predicciones.

5.6 Validación del funcionamiento del módulo de conversión y respuesta

Se ha simulado la respuesta de la señal previamente filtrada y clasificada, a través del pin de conversión analógico-digital A0 de Arduino Uno y con dos monitores de salida para visualizar la respuesta del ciclo de trabajo de la señal PWM después de su conversión y su correspondiente valor PWM entre 0-255.

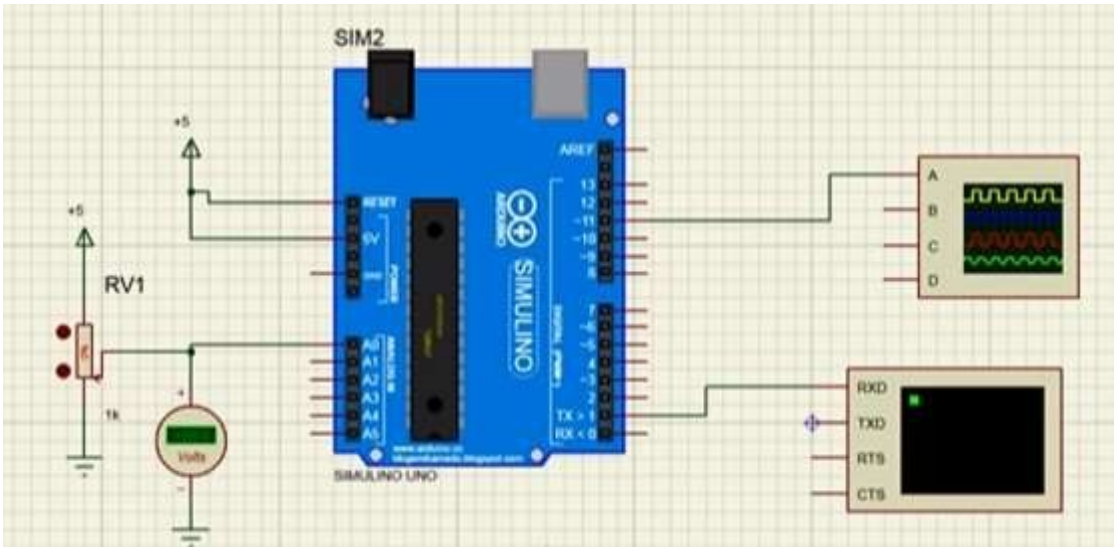


FIGURA 62. Diagrama esquemático de respuesta del sistema

Al convertir la señal analógica con una resolución de 10 bits se devuelven enteros de entre 0 y 1023 que corresponden a la salida PWM de 8 bits, con un rango de entre 0-255.

De esta manera, si las frecuencias de entrada arrojan un 0% del rango que cubre el ADC, el ciclo de trabajo (Duty Cycle) se mantiene en 0.

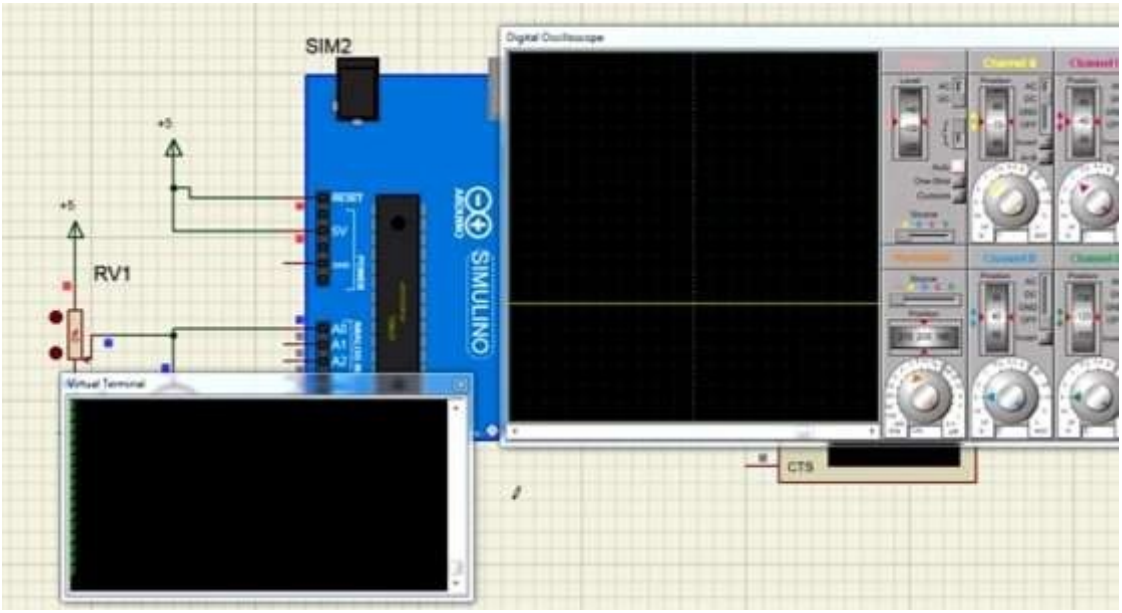


FIGURA 63. Respuesta del sistema al recibir un Ciclo de Trabajo de 0%

Si la señal analógica representa a un tercio (33.3%) del rango del ADC, es decir, un 341 de los 1023, corresponderá a un Ciclo de Trabajo con el mismo porcentaje en el rango del PWM, es decir, un 85 de 255.

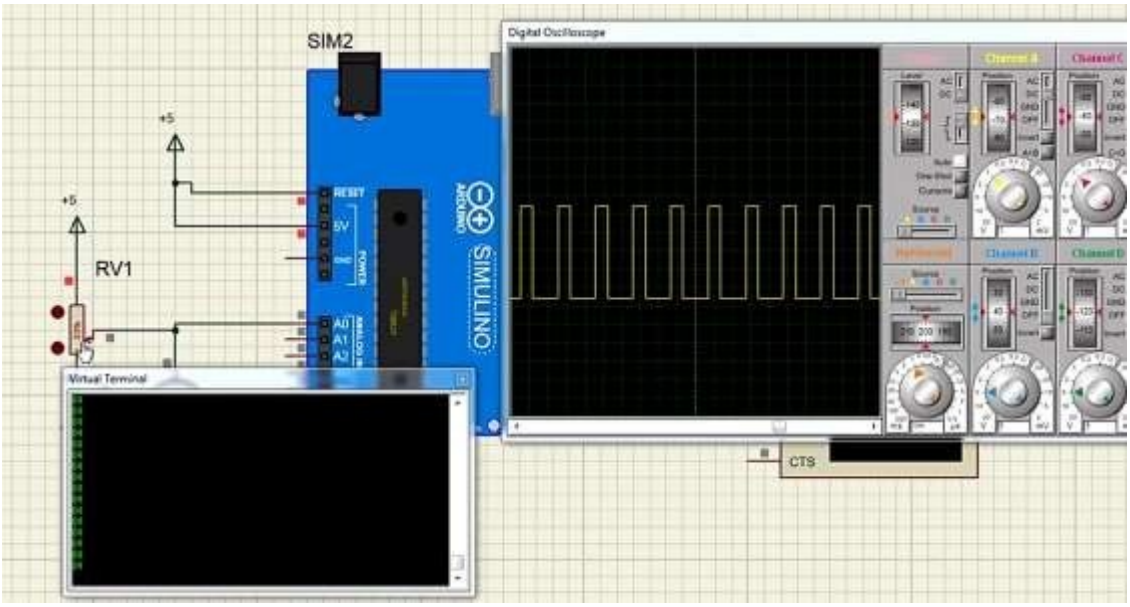


FIGURA 64. Respuesta del sistema al recibir un Ciclo de Trabajo de 33.3%

Así, si la señal de entrada representase la mitad (50%) del rango del ADC con un valor de aproximadamente 512, la respuesta del Ciclo de Trabajo del PWM sería proporcional, es decir, un valor de entre 127 y 128.

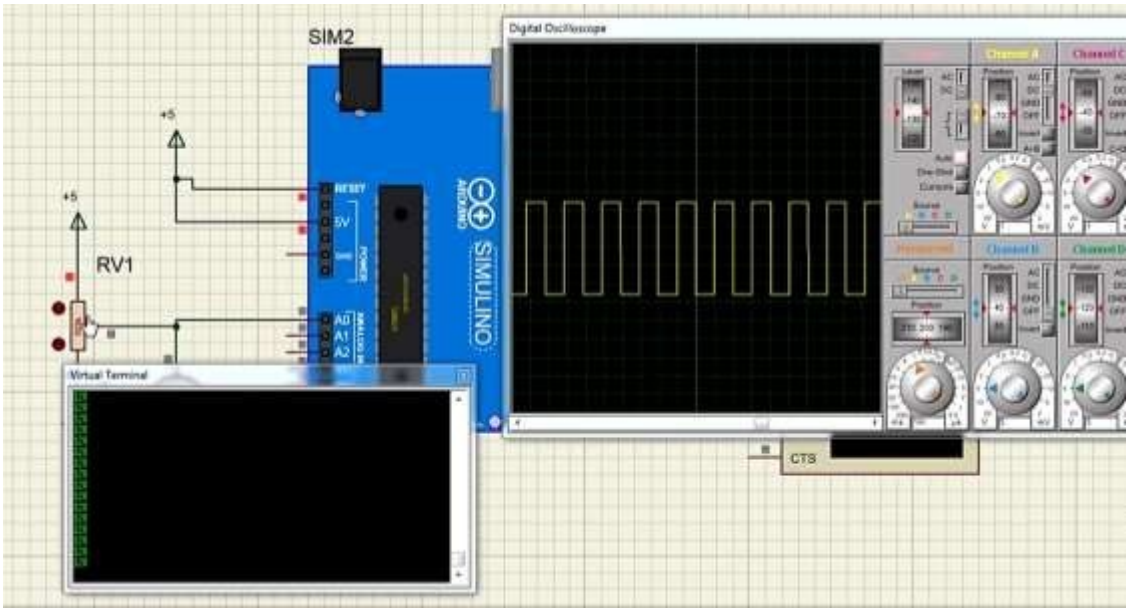


FIGURA 65. Respuesta del sistema al recibir un Ciclo de Trabajo de 50%

Observamos un Ciclo de Trabajo del PWM de aproximadamente 75%, con un valor de 191 en su propio rango, derivando así una señal de entrada por ADC de 767 aproximadamente.

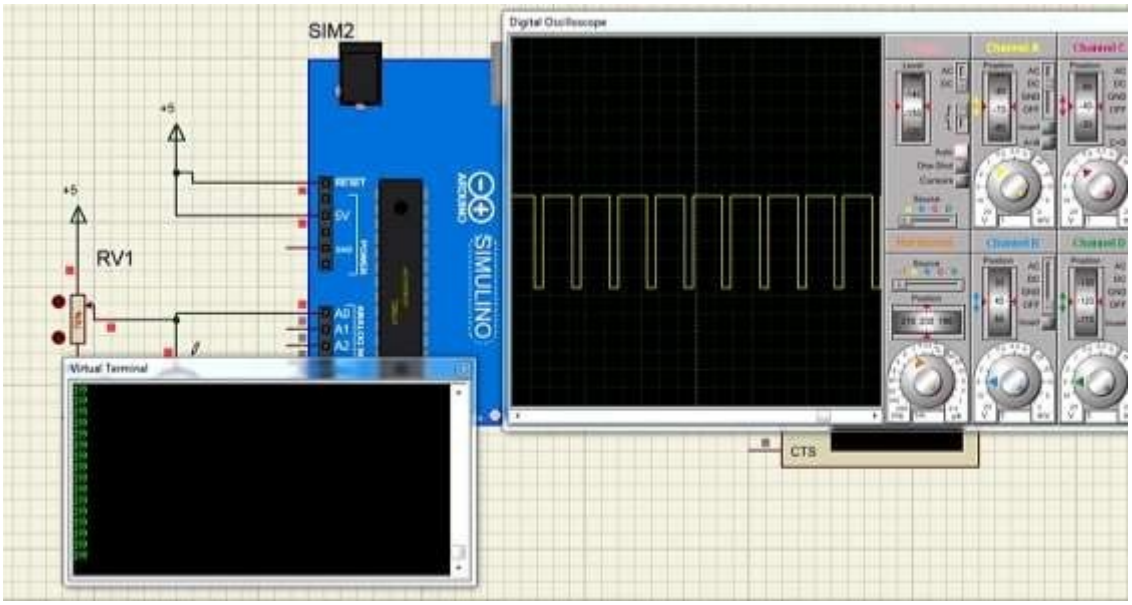


FIGURA 66. Respuesta del sistema al recibir un Ciclo de Trabajo de 75%

Si la frecuencia de entrada fuera las más elevada después del proceso de filtrado y clasificación, correspondiente a un 100% en el rango del convertidor ADC (1023) y Ciclo de Trabajo del PWM (255), se visualiza un estado en alto (HIGH) constante.

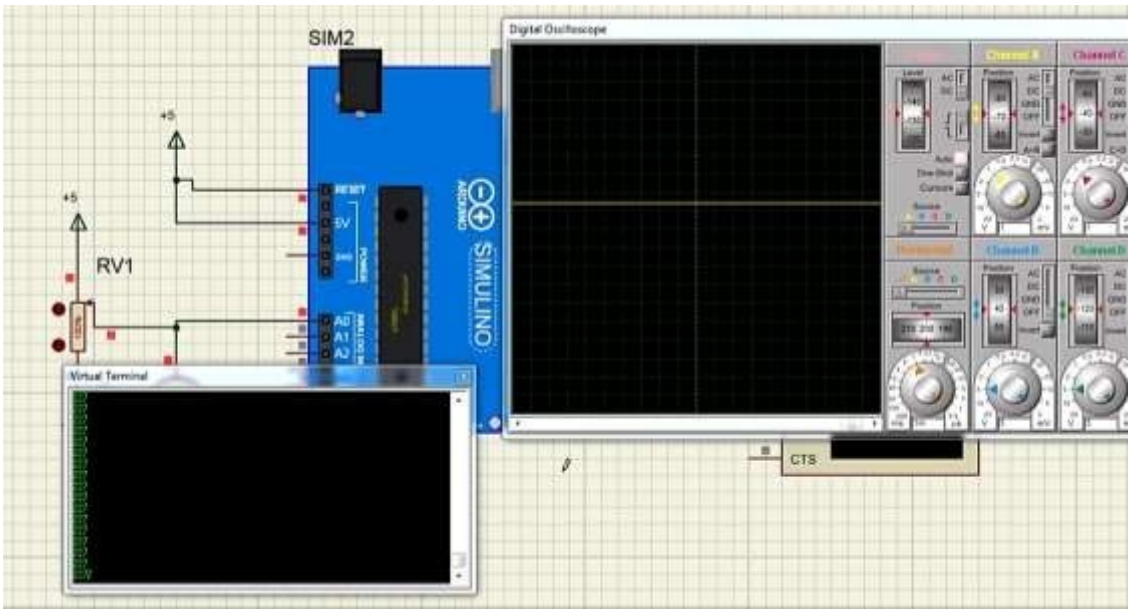


FIGURA 67. Respuesta del sistema al recibir un Ciclo de Trabajo de 100%

De este modo, la intensidad luminica en la tira LED y la velocidad del motor para la vibración, serán proporcionales al porcentaje del Ciclo de Trabajo del PWM.

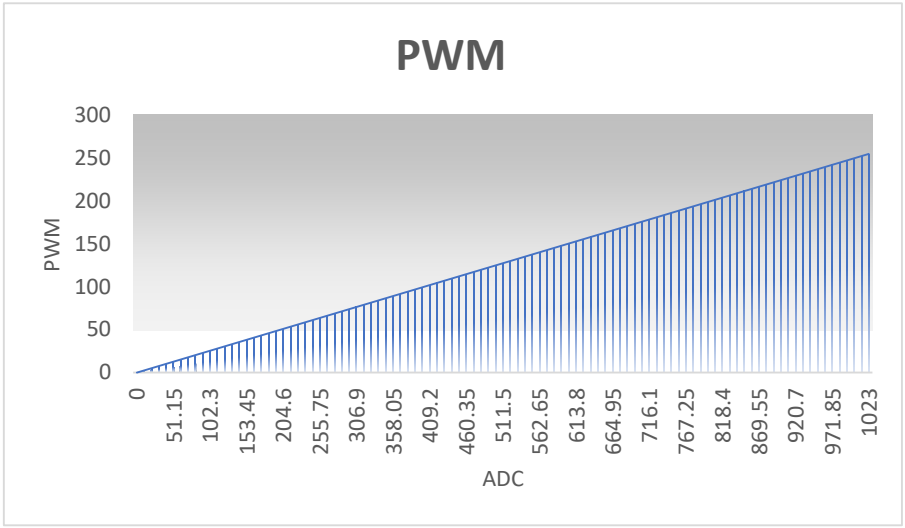


FIGURA 68. Relación proporcional del ADC con PWM

6. CONCLUSIONES

En el presente proyecto se ha podido implementar el proceso completo de conversión de señales acústicas a señales visuales y táctiles, partiendo desde la recepción de los sonidos externos, los que a su vez pasaron por etapas de filtrado y clasificación hasta la obtención de las señales de salida deseadas.

Como se ha señalado en el desarrollo del proyecto, se utilizaron instrumentos que, aunque no fueron los más avanzados tecnológicamente, tenían las prestaciones para suplir nuestras necesidades, tal es el caso del micrófono utilizado.

Si bien, al momento de pensar este proyecto se tenía la intención de realizar la sección del filtrado utilizando componentes electrónicos tangibles, lo cual resultó imposible por la falta de material, si fue posible realizarlo a través de programación, por lo que esa parte ha quedado cubierta. Además de que fue posible utilizar diferentes recursos tales como la Transformada Rápida de Fourier en Matlab que optimizaron los procesos.

Y aunque no fue posible el montaje y prueba en un vehículo, se ha procurado la ejecución integral del prototipo considerando los parámetros reales del ambiente en que se está inmerso. Iniciando con una sección de alimentación que suponía la batería del vehículo como el medio de alimentación principal de nuestro sistema, simulando el mismo, para garantizar el funcionamiento de cada módulo posterior.

Gracias a la correcta selección de los programas informáticos (Software) fue posible la ejecución de los módulos con mayor demanda de procesamiento, tanto para el filtrado como para la clasificación. Aun recurriendo a ellos para realizar las simulaciones necesarias para suplir las partes que por la situación actual no fueron posibles realizar físicamente.

El algoritmo programado para la clasificación tuvo un rendimiento considerablemente bueno, el cual junto con la base de datos que permitió el entrenamiento de la red, permitieron que este modulo en particular resultará óptimo.

Fue necesario improvisar alguna forma para obtener las señales de salida propuestas, y aunque los materiales no eran enteramente especializados para ello, fue posible alcanzar el resultado. Por un lado, con la utilización de LED’s para suplir la señal visual, y a falta de las celdas de

vibración, un motor DC que impulsa movimiento provocando la vibración requerida para la señal táctil.

La mayor complicación enfrentada fue la ejecución en tiempo real, ya que, al estar sujetos a los diferentes Software, estos necesitan formatos específicos para poder realizar su funcionamiento. Así que, por esto el proyecto queda abierto para futuras propuestas, en donde puedan utilizarse instrumentos de alta gama que permitan el almacenamiento de los programas realizados y una ejecución secuencial y en tiempo real.

Por último, cabe decir que los objetivos se han alcanzado con éxito, realizando un prototipo de conversión de señales versátil. Permite incorporar bases de datos, cambiar los valores de las frecuencias de corte y evaluar clasificadores utilizando algoritmos de Aprendizaje Automático, diversificando así su aplicación con cambios sumamente sencillos.

7. REFERENCIAS

[1] J. Salamon, C. Jacoby and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research", 22nd ACM International Conference on Multimedia, Orlando USA, Nov. 2014.

[2] Gorgolewski, C. “UrbanSound8K”. URL: https://www.kaggle.com/chrisfilo/urbansound8k

[3] CS231n: Convolutional Neural Networks for Visual Recognition. URL: https://cs231n.github.io/convolutional-networks/

[4] McClelland , C. 2017. The DDifference Between Artificial Intelligence, Machine Learning, and Deep Learning. Medium. URL: https://medium.com/iotforall/the-difference-between-artificial-intelligence-machine-learning-and-deep-learning-3aa67bff5991.

[5] M. Al-Maathidi M and Francis F. L. (2015) “Audio Content Feature Selection and Classification” en IEEE International Conference on Progress in Informatics and Computing (PIC). Nanjing, China. December 2015

[6] S. Chachada and C.-C. J. Kuo, “Environmental sound recognition: A survey,” en Proc. Asia Pac. Signal Inf. Process. Assoc. Annu. Summit Conf., Kaohsiung, Taiwan, 2013, pp. 1–9.

[7] Gil-Pita, R., Ayllón, D., Ranilla, J., Llerena-Aguilar, C., & Díaz, I. (2015). “A computationally efficient sound environment classifier for hearing aids”. IEEE Transactions on Biomedical Engineering. Vol. 62, n. 10, October 2015 pp. 2358-2368.

[8] Center for Urban Science and Progress (NYU). Urban Sound Datasets.

[9] Playrec, Multi-Channel Matlab Audio. Playrec.

[10] Bickford, Anita; “Articulatory Phonetics: Tools For Analyzing The World's Languages” (4th ed.); Summer Institute of Linguistics.

[11] Shewan, Robert; "Voice Classification: An Examination of Methodology"; The NATS Bulletin. 35: 17–27.

[12] Atal, B. S. and Hanauer, S. L. ; "Speech Analysis and Synthesis by Linear Prediction of the Speech Wave"; J. Acoust. Soc. Am., 50, 637-655.

Anexo I: Micrófono Tipo 4961

Características Técnicas

PRODUCT DATA

Multi-field Microphone Type 4961

1/4″ Microphone with TEDS

Mulit-field Microphone Type 4961 is optimised for multi-field response, which means that it can be used to perform measurements in a wide range of sound fields.

Multi-field Microphones are ideal for any situation in which the nature of the sound field is unpredictable, or when the direction of the dominant noise source is difficult to pinpoint or shifts over time.

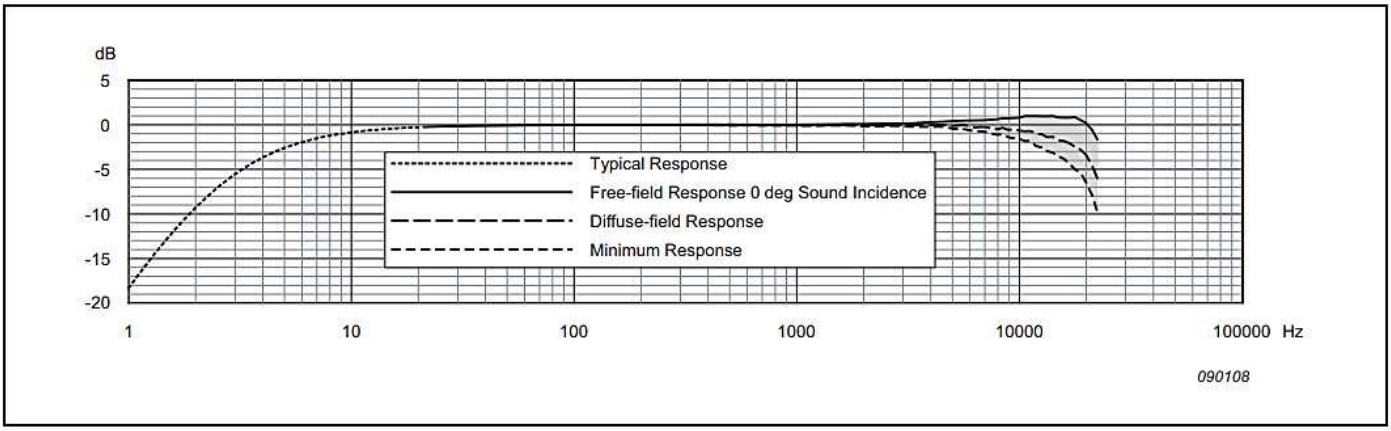
Uses

- Measurements in unpredictable sound field conditions
- Cabin noise measurements
- Near-field measurements
- Ad-hoc sound measurements

Features

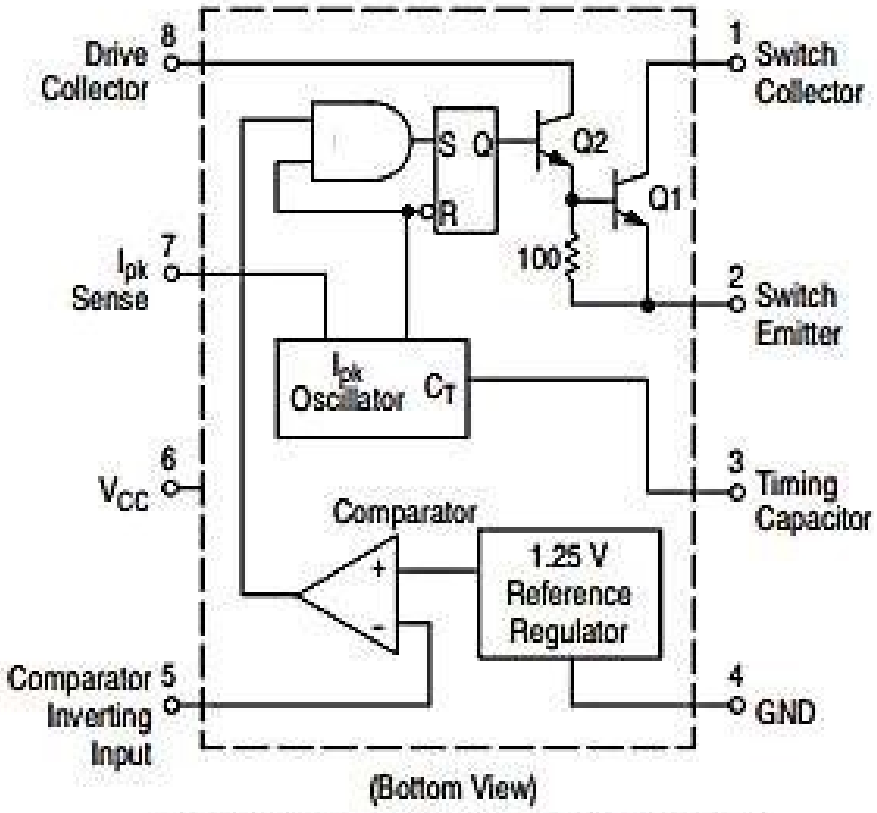
- Sensitivity: 60 mV/Pa
- Frequency range: 5 Hz–20 kHz
- Dynamic Range: 20–130 dB
- TEDS: UTID No. 769, IEEE P1451.4 V0.9
- TEDS IEEE 1451.4 V 1.0 can be ordered
- Temperature: – 20 to +70°C (– 4 to +158°F)
- Connects directly to DeltaTron® (CCLD) input

Curva de Respuesta



Anexo II: Circuito Integrado MC34063A

Diagrama Interno



This device contains 79 active transistors.

Tabla de Fórmulas de Diseño

MC34063A, MC33063A, SC34063A, SC33063A, NCV33063A

Calculation	Step-Up	Step-Down	Voltage-Inverting
t_{on}/t_{off}	$\frac{V_{out} + V_F - V_{in(min)}}{V_{in(min)} - V_{sat}}$	$\frac{V_{out} + V_F}{V_{in(min)} - V_{sat} - V_{out}}$	$\frac{\lvert V_{out}\rvert + V_F}{V_{in} - V_{sat}}$
$(t_{on} + t_{off})$	$\frac{1}{f}$	$\frac{1}{f}$	$\frac{1}{f}$
t_{off}	$\frac{t_{on} + t_{off}}{\frac{t_{on}}{t_{off}} + 1}$	$\frac{t_{on} + t_{off}}{\frac{t_{on}}{t_{off}} + 1}$	$\frac{t_{on} + t_{off}}{\frac{t_{on}}{t_{off}} + 1}$
t_{on}	$(t_{on} + t_{off}) - t_{off}$	$(t_{on} + t_{off}) - t_{off}$	$(t_{on} + t_{off}) - t_{off}$
C_T	$4.0 \times 10^{-5}\, t_{on}$	$4.0 \times 10^{-5}\, t_{on}$	$4.0 \times 10^{-5}\, t_{on}$
$I_{pk(switch)}$	$2I_{out(max)}\left(\frac{t_{on}}{t_{off}} + 1\right)$	$2I_{out(max)}$	$2I_{out(max)}\left(\frac{t_{on}}{t_{off}} + 1\right)$
R_{sc}	$0.3/I_{pk(switch)}$	$0.3/I_{pk(switch)}$	$0.3/I_{pk(switch)}$
$L_{(min)}$	$\left(\frac{(V_{in(min)} - V_{sat})}{I_{pk(switch)}}\right) t_{on(max)}$	$\left(\frac{(V_{in(min)} - V_{sat} - V_{out})}{I_{pk(switch)}}\right) t_{on(max)}$	$\left(\frac{(V_{in(min)} - V_{sat})}{I_{pk(switch)}}\right) t_{on(max)}$
C_O	$9\,\frac{I_{out} t_{on}}{V_{ripple(pp)}}$	$\frac{I_{pk(switch)}(t_{on} + t_{off})}{8V_{ripple(pp)}}$	$9\,\frac{I_{out} t_{on}}{V_{ripple(pp)}}$

V_{sat} = Saturation voltage of the output switch.
V_F = Forward voltage drop of the output rectifier.

The following power supply characteristics must be chosen:

V_{in} – Nominal input voltage.
V_{out} – Desired output voltage, $|V_{out}| = 1.25\left(1 + \frac{R2}{R1}\right)$
I_{out} – Desired output current.
f_{min} – Minimum desired output switching frequency at the selected values of V_{in} and I_O.
$V_{ripple(pp)}$ – Desired peak-to-peak output ripple voltage. In practice, the calculated capacitor value will need to be increased due to its equivalent series resistance and board layout. The ripple voltage should be kept to a low value since it will directly affect the line and load regulation.

NOTE: For further information refer to Application Note AN920A/D and AN954/D.

Anexo III: Configuración ADC

Tabla de selección de Prescaladores

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Tabla de selección de Modo de Activación

ADTS2	ADTS1	ADTS0	Trigger source
0	0	0	Free running mode
0	0	1	Analog comparator
0	1	0	External interrupt request 0
0	1	1	Timer/counter0 compare match A
1	0	0	Timer/counter0 overflow
1	0	1	Timer/counter1 compare match B
1	1	0	Timer/counter1 overflow
1	1	1	Timer/counter1 capture event

Anexo IV: Transistores de Potencia

Valores Máximos TIP120

Rating	Symbol	TIP120, TIP125	TIP121, TIP126	TIP122, TIP127	Unit
Collector–Emitter Voltage	V_{CEO}	60	80	100	Vdc
Collector–Base Voltage	V_{CB}	60	80	100	Vdc
Emitter–Base Voltage	V_{EB}	5.0			Vdc
Collector Current – Continuous – Peak	I_C	5.0 8.0			Adc
Base Current	I_B	120			mAdc
Total Power Dissipation @ T_C = 25°C Derate above 25°C	P_D	65 0.52			W W/°C
Total Power Dissipation @ T_A = 25°C Derate above 25°C	P_D	2.0 0.016			W W/°C
Unclamped Inductive Load Energy (Note 1)	E	50			mJ
Operating and Storage Junction, Temperature Range	T_J, T_{stg}	–65 to +150			°C

Valores Máximos 2N2222A

Characteristic	Symbol	Value	Unit
Collector – Emitter Voltage	V_{CEO}	40	Vdc
Collector – Base Voltage	V_{CBO}	75	Vdc
Emitter – Base Voltage	V_{EBO}	6.0	Vdc
Collector Current – Continuous	I_C	600	mAdc
Total Device Dissipation @ T_A = 25°C Derate above 25°C	P_D	625 5.0	mW mW/°C
Total Device Dissipation @ T_C = 25°C Derate above 25°C	P_D	1.5 12	W mW/°C
Operating and Storage Junction Temperature Range	T_J, T_{stg}	–55 to +150	°C

Apéndice I: Código de Preprocesamiento y Filtrado en Matlab

```
%% Audio Import

[a , fs] = audioread("Audio.wav");
d = length(a)/fs;
a_m = 0.5*(a(:,1) + a(:,2)).';

%% Waveform Plot

t = linspace(0, d, length(a_m));

figure();
plot(t, a_m);
title("Audio Waveform");
xlabel("Time [s]");
ylabel("Amplitude");
grid on;

%% Frecuency

A_m = fftshift(fft(a_m));

f = linspace(-fs/2, fs/2, length(A_m));

mag_A = abs(A_m);

figure();
plot(f, mag_A/max(mag_A));
title("Audio Frecuency Spectrum");
xlabel("Frecuency [Hz]");
ylabel("Amplitude");
grid on, grid minor;

ax = gca;
ax.XAxis.Exponent = 3;


%% Low Pass Filter

lpf = 1.*(abs(f)<2000);

figure();
plot(f, lpf, 'r');
title("Low Pass Filter");
xlabel("Frecuency [Hz]");
ylabel("Amplitude");
grid on, grid minor;

ax = gca;
```

```
ax.XAxis.Exponent = 3;

%% Filter and Audio

figure();
plot(f, mag_A/max(mag_A));
hold on;
plot(f, lpf, 'r');
legend("Audio", "Filter");
title("Low Pass Filter");
xlabel("Frecuency [Hz]");
ylabel("Amplitude");
grid on, grid minor;

ax = gca;
ax.XAxis.Exponent = 3;

%% Filtering

A_lpf = A_m .* lpf; % Filtering
figure();
plot(f, abs(A_lpf)/max(abs(A_lpf)));
hold on;
plot(f, lpf, 'r');
legend("Filtered Audio", "Filter");
title("Low Pass Filter");
xlabel("Frecuency [Hz]");
ylabel("Amplitude");
grid on, grid minor;

ax = gca;
ax.XAxis.Exponent = 3;

%% Inverse FFT

a_lpf = ifft(fftshift(A_lpf));
a_lpf = real(a_lpf);


%% High Pass Filter

hpf = 1.*(abs(f)>=400);

figure();
plot(f, hpf, 'b');
title("High Pass Filter");
xlabel("Frecuency [Hz]");
ylabel("Amplitude");
grid on, grid minor;

ax = gca;
ax.XAxis.Exponent = 3;
```

```
%% Filter and Audio

figure();
plot(f, mag_A/max(mag_A));
hold on;
plot(f, hpf, 'r');
legend("Audio", "Filter");
title("High Pass Filter");
xlabel("Frecuency [Hz]");
ylabel("Amplitude");
grid on, grid minor;

ax = gca;
ax.XAxis.Exponent = 3;

%% Filtering

A_hpf = A_m .* hpf; % Filtering
figure();
plot(f, abs(A_hpf)/max(abs(A_hpf)));
hold on;
plot(f, hpf, 'r');
legend("Filtered Audio", "Filter");
title("High Pass Filter");
xlabel("Frecuency [Hz]");
ylabel("Amplitude");
grid on, grid minor;

ax = gca;
ax.XAxis.Exponent = 3;

%% Inverse FFT

a_hpf = ifft(fftshift(A_hpf));

figure();
plot(a_hpf);

a_hpf = real(a_hpf);


%% Butterworth Filter

NFFT = 2^nextpow2(L);
Y = fft(a,NFFT)/L;
f = fs/2*linspace(0,1,NFFT/2+1);

y = filter(Filtro,a);
Y1 = fft(y,NFFT)/L;

subplot(2,1,1);
plot(f,2*abs(Y(1:NFFT/2+1)));
```

```
legend('Señal original');

subplot(2,1,2);
plot(f,2*abs(Y1(1:NFFT/2+1)),'r');
legend('Señal filtrada');


%% Play Audios

sound(a_hpf, fs);
pause(d+1);

sound(a_lpf, fs);
pause(d+1);

sound(Y1, fs);
pause(d+1);
```

Apéndice II: Código de Configuración de ADC en AtmelStudio

```
#include <avr/io.h>

int adc_value; // Variable utilizada para almacenar el valor leído del convertidor ADC

int main(void){

  DDRB |= (1<<PB5); // PB5 digital 13 es una salida
  ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)); // Prescaler a 128, por lo que tenemos
                                                   una fuente de reloj de 125Khz
  ADMUX |= ~(1<<REFS0);
  ADMUX &= ~(1<<REFS1); // Avcc(+5v) como voltaje de referencia
  ADCSRB &= ~((1<<ADTS2)|(1<<ADTS1)|(1<<ADTS0)); // ADC en modo free-running
  ADCSRA |= (1<<ADATE); // Fuente de señal, en este caso es el free-running
  ADCSRA |= (1<<ADEN); // Enciende el ADC
  ADCSRA |= (1<<ADSC); // Comienza conversión

     for(;;){
       if(adc_value > 512){ // Condicional de acuerdo a aplicación
         PORTB |= (1<<PB5); // Si el valor de ADC es superior al valor manda un HIGH
       }
       else {
         PORTB &= ~(1<<PB5); // De lo contrario un LOW
       }
     }
  return 0;
}
```

Apéndice III: Código para Módulo de Clasificación

```
#!pip install numba
```

In [1]:

```
#Importamos la librerías y dependencias que vamos a utilizar
import numba as nb
import pandas as pd
import numpy as np
import os
import glob
import skimage
import librosa
import librosa.display
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import models
import IPython.display as ipd
plt.style.use("seaborn-dark")

pd.plotting.register_matplotlib_converters()
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPool2D, Dropout
from tensorflow.keras.utils import to_categorical

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Análisis de datos

Tipo y formato

In [2]:

```
#Importar el dataset
sonidos=pd.read_csv('dataset_sounds/UrbanSound8K.csv')
```

In [3]:

```
#Visualizamos la distribución de datos de cada categoria
appended = []
for i in range(1,11):
    appended.append(sonidos[sonidos.fold == i]['class'].value_counts())


class_distribution = pd.DataFrame(appended)
class_distribution = class_distribution.reset_index()
class_distribution['index'] = ["fold"+str(x) for x in range(1,11)]
class_distribution
```

Out[3]:

In [4]:

```
#Informacion del dataset
sonidos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8732 entries, 0 to 8731
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   slice_file_name  8732 non-null   object
 1   fsID             8732 non-null   int64
 2   start            8732 non-null   float64
 3   end              8732 non-null   float64
 4   salience         8732 non-null   int64
 5   fold             8732 non-null   int64
 6   classID          8732 non-null   int64
 7   class            8732 non-null   object
dtypes: float64(2), int64(4), object(2)
memory usage: 545.9+ KB
```

Analizar una muestra de sonido usando librosa

In [5]:

```
samp1, sampling_rate1 = librosa.load('dataset_sounds/fold5/100032-3-0-0.wav')
```

In [6]:

```
plt.figure(figsize=(20, 10),facecolor="white")
D = librosa.amplitude_to_db(np.abs(librosa.stft(samp1)), ref=np.max)
plt.subplot(4, 2, 1)
librosa.display.specshow(D, y_axis='linear')
```

```
plt.colorbar(format='%+2.0f dB')
plt.title('Linear-frequency power spectrogram')
```

Out[6]:

```
Text(0.5, 1.0, 'Linear-frequency power spectrogram')
```

Observar las diferencias en las formas de onda

In [7]:

```
arr = np.array(sonidos["slice_file_name"])
fold = np.array(sonidos["fold"])
cla = np.array(sonidos["class"])


for i in range(192, 197, 2):
    path = 'dataset_sounds/fold' + str(fold[i]) + '/' + arr[i]
    data, sampling_rate = librosa.load(path)
    plt.figure(figsize=(10, 5),facecolor='white')
    D = librosa.amplitude_to_db(np.abs(librosa.stft(data)), ref=np.max)
    plt.subplot(4, 2, 1)
    librosa.display.specshow(D, y_axis='linear')
    plt.colorbar(format='%+2.0f dB')
    plt.title(cla[i])
```

Extracción de características y creación de base de datos

In [8]:

```
#Ejemplo
samp1, sampling_rate1 = librosa.load('dataset_sounds/fold5/100032-3-0-0.wav')
arr = librosa.feature.melspectrogram(y=samp1, sr=sampling_rate1)
arr.shape
```

Out[8]:

```
(128, 14)
```

Extraer las características MFCC de los archivos wav

Leer archivos de sonido wav, y extraer características mfcc, así como etiquetas

In [9]:

```
feature = []
label = []
# Funcion para cargar los archivos y extraer características
def parser(row):
```

```
    for i in range(8732):
        file_name = 'dataset_sounds/fold' + str(sonidos["fold"][i]) + '/' +
sonidos["slice_file_name"][i]
        #kaiser_fast es una técnica usada para la extraxión rápida de
características
        X, sample_rate = librosa.load(file_name, res_type='kaiser_fast')
        # Extraer la función mfcc de los datos
        mels = np.mean(librosa.feature.melspectrogram(y=X,
sr=sample_rate).T,axis=0)
        feature.append(mels)
        label.append(sonidos["classID"][i])
    return [feature, label]
```

In [10]:

```
temp = parser(sonidos)
```

```
C:\Users\aracely\anaconda3\lib\site-packages\librosa\core\spectrum.py:224: UserWar
ning: n_fft=2048 is too small for input signal of length=1323
  n_fft, y.shape[-1]
C:\Users\aracely\anaconda3\lib\site-packages\librosa\core\spectrum.py:224: UserWar
ning: n_fft=2048 is too small for input signal of length=1103
  n_fft, y.shape[-1]
C:\Users\aracely\anaconda3\lib\site-packages\librosa\core\spectrum.py:224: UserWar
ning: n_fft=2048 is too small for input signal of length=1523
  n_fft, y.shape[-1]
```

In [11]:

```
temp = np.array(temp)
data = temp.transpose()
```

```
C:\Users\aracely\anaconda3\lib\site-packages\ipykernel_launcher.py:1: VisibleDepre
cationWarning: Creating an ndarray from ragged nested sequences (which is a list-o
r-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprec
ated. If you meant to do this, you must specify 'dtype=object' when creating the n
darray
  """Entry point for launching an IPython kernel.
```

In [12]:

```
X_ = data[:, 0]
Y = data[:, 1]
print(X_.shape, Y.shape)
X = np.empty([8732, 128])
```

```
(8732,) (8732,)
```

In [13]:

```
for i in range(8732):
    X[i] = (X_[i])
```

In [14]:

```
Y = to_categorical(Y)
```

In [15]:

```
print(X.shape)

print(Y.shape)
```

```
(8732, 128)
(8732, 10)
```

Dividimos el conjunto de datos en entrenamiento y test

75% de los datos se ocuparán para entrenamiento y el 25% restante para testing

In [16]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state = 1)
```

In [17]:

```
#Fase de entrenamiento

X_train = X_train.reshape(6549, 16, 8, 1)

#Fase de testing para evaluar el modelo

X_test = X_test.reshape(2183, 16, 8, 1)
```

In [18]:

```
input_dim = (16, 8, 1)
```

Creación y definición del modelo y pruebas de Keras

CNN 2D con 64 unidades y función de activación tanh. MaxPool2D con ventana 2 *2. CNN 2D con 128 unidades y activación tanh. MaxPool2D con ventana 2* 2. Capa de abandono con 0,2 de probabilidad de caída. DL con 1024 unidades y activación tanh. Unidades DL 10 con activación softmax. Optimizador de Adam con función de pérdida categorical_crossentropy. Se han utilizado 50 épocas.

In [19]:

```
#Define una pila lineal de capas de red

model = Sequential()
```

In [20]:

```
#add

#Método para agregar capas al modelo

#Capa NN densamente conectada

model.add(Conv2D(64, (3, 3), padding = "same", activation = "tanh", input_shape =
input_dim))

model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), padding = "same", activation = "tanh"))

model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Dropout(0.1))

model.add(Flatten())

model.add(Dense(1024, activation = "tanh"))

model.add(Dense(10, activation = "softmax"))
```

In [21]:

```
#Optimizador con parametros predeterminados

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
['accuracy'])
```

Proceso de formación y entrenamiento del modelo

Este modelo se entrenó en una instancia de AWS p2.x durante cincuenta épocas con un tamaño de lote de 128.

In [22]:

```
history = model.fit(X_train, Y_train, epochs=150, batch_size=64,
validation_data=(X_test, Y_test))
```

```
Epoch 1/150
103/103 [==============================] - 32s 261ms/step - loss: 1.9312 - accurac
y: 0.3853 - val_loss: 1.5473 - val_accuracy: 0.4993
Epoch 2/150
103/103 [==============================] - 9s 87ms/step - loss: 0.0477 - accuracy:
0.9816 - val_loss: 1.0510 - val_accuracy: 0.8607
Epoch 149/150
. . .
103/103 [==============================] - 12s 122ms/step - loss: 0.0302 - accurac
y: 0.9895 - val_loss: 1.2223 - val_accuracy: 0.8644
Epoch 150/150
103/103 [==============================] - 13s 127ms/step - loss: 0.0221 - accurac
y: 0.9918 - val_loss: 1.2152 - val_accuracy: 0.8639
```

In [24]:

```
#Visualizamos el proceso de formación

plt.style.use("classic")

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

t = f.suptitle('Deep Neural Net Performance', fontsize=12)

f.subplots_adjust(top=0.85, wspace=0.2)

epochs = list(range(1,151))

ax1.plot(epochs, history.history['accuracy'], label='Train Accuracy')

ax1.plot(epochs, history.history['val_accuracy'], label='Validation Accuracy')

ax1.set_ylabel('Accuracy Value')

ax1.set_xlabel('Epoch')

ax1.set_title('Accuracy')

l1 = ax1.legend(loc="best")

ax2.plot(epochs, history.history['loss'], label='Train Loss')

ax2.plot(epochs, history.history['val_loss'], label='Validation Loss')

ax2.set_ylabel('Loss Value')

ax2.set_xlabel('Epoch')

ax2.set_title('Loss')
```

```
l2 = ax2.legend(loc="best")
```

In [26]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 8, 64)         640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 4, 64)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 4, 128)         73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 2, 128)         0
_________________________________________________________________
dropout (Dropout)            (None, 4, 2, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              1049600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                10250
=================================================================
Total params: 1,134,346
Trainable params: 1,134,346
Non-trainable params: 0
_________________________________________________________________
```

In [27]:

```
#Precisión de la predicción
#Visualización de métricas
def acc(y_test,prediction):
    cm = confusion_matrix(y_test, prediction)
    #Capacidad del modelo para recordar todas las instancias relevantes
    recall = np.diag(cm) / np.sum(cm, axis = 1)
    #Capacidad del modelo de devolver sólo instamcias relevantes
    precision = np.diag(cm) / np.sum(cm, axis = 0)

    print ('Recall:', recall)
    print ('Precision:', precision)
    #Informe de clasificación
    print ('\n clasification report:\n', classification_report(y_test,prediction))
    #Matriz de confución
    print ('\n confussion matrix:\n',confusion_matrix(y_test, prediction))
```

```
    ax = sns.heatmap(confusion_matrix(y_test, prediction),linewidths=
0.5,cmap="YlGnBu")
```

Evaluamos el rendimiento del modelo en los datos de prueba

In [28]:

```
predictions = model.predict(X_test)

score = model.evaluate(X_test, Y_test)

print(score)
```

```
69/69 [==============================] - 2s 21ms/step - loss: 1.2152 - accuracy: 0
.8639
[1.215211033821106, 0.8639487028121948]
```

In [29]:

```
preds = np.argmax(predictions, axis = 1)
```

In [30]:

```
result = pd.DataFrame(preds)

result.to_csv("UrbanSound8kResults.csv")
```

ÍNDICE DE FIGURAS

Página

ÍNDICE DE TABLAS

Página