Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías de Telecomunicación

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

SISTEMA DE GENERACIÓN DE AUDIO PARA PAQUETES PERDIDOS EN APLICACIONES DISTRIBUIDAS DE INTERPRETACIÓN DE MÚSICA EN TIEMPO REAL

Alumna: **Marta Soler Martínez**

Tutor: Prof. D. Pedro Vera Candeas

Tutor: Prof. D. Pablo A. Cabañas Molero

Depto.: Ingeniería de Telecomunicación

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

SISTEMA DE GENERACIÓN DE AUDIO PARA PAQUETES PERDIDOS EN APLICACIONES DISTRIBUIDAS DE INTERPRETACIÓN DE MÚSICA EN TIEMPO REAL

Alumna: Marta Soler Martínez

Tutor: Prof. D. Pedro Vera Candeas

Tutor: Prof. D. Pablo A. Cabañas Molero

Depto.: Ingeniería de Telecomunicación

Noviembre, 2023

Firma de la autora:	Firma del tutor:	Firma del tutor:
Firmado por SOLER MARTINEZ MARTA - ***5965** el día 06/11/2023	VERA CANDEAS PEDRO - 26228400M Digitally signed by VERA CANDEAS PEDRO - 26228400M Date: 2023.11.06 13:26:10 +01'00'	CABAÑAS MOLERO PABLO ANTONIO - 77342099E Firmado digitalmente por CABAÑAS MOLERO PABLO ANTONIO - 77342099E Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-77342099E, givenName=PABLO ANTONIO, sn=CABAÑAS MOLERO, cn=CABAÑAS MOLERO PABLO ANTONIO - 77342099E Fecha: 2023.11.06 12:38:05 +01'00'

INDICE DE CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

GLOSARIO DE SIGLAS

NMP – *Networked Music Performance*

MIDI – *Musical Instrument Digital Interface*

P2P – *Peer to Peer*

PLC – *Packet Loss Concealment*

PCM – *Pulse Code Modulation*

MICD – *Modulación por Impulsos Codificados Diferencial*

A/D-ADC – *Conversor Analógico-Digital*

FTTH – *Fiber To The Home*

CNMC – *Comisión Nacional de los Mercados y la Competencia*

TCP – *Transmission Control Protocol*

UDP – *User Datagram Protocol*

OOP – *Programación Orientada a Objetos*

D/A-DAC – *Digital Analog Converter*

WAN – *Wide Area Network*

MAN – *Metropolitan Area Network*

RAW – *Audio sin compresión*

WAV – *WAVeform audio file format*

NOT – *Note Onset Time*

AOT – *Acoustic Onset Time*

POT – *Perceptual Onset Time*

PPM-BPM – *Pulsos Por Minuto* / *Beats Per Second*

AWGN – *Additive White Gaussian Noise*

FFT – *Fast Fourier Transform*

DFT – *Discrete Fourier Transform*

RESUMEN

En las últimas décadas, el rápido desarrollo de aplicaciones de Networked Music Performance (NMP), ha llevado la comunicación sincronizada en línea un paso más allá en lo referido a la interpretación musical de forma conjunta. Recientemente, durante la pandemia provocada por el COVID-19, la tarea de garantizar la calidad del audio a través de Internet supuso un reto considerable y las tecnologías audiovisuales basadas en la web demostraron ser insuficientes en cuanto a restricciones de red tales como la latencia. Minimizar este parámetro es un compromiso que sacrifica la robustez en transmisión, lo que se traduce en la pérdida de paquetes. Aquí es donde entra en juego una solución complementaria de *Packet Loss Concealment* (PLC) desarrollada en el presente trabajo.

El software "*Audio_refiller*" tiene como objetivo mitigar los huecos faltantes de información provocados por las pérdidas en la conexión antes de que los paquetes lleguen al receptor, previo a la ejecución del software NMP. Esto se ha logrado mediante la programación de varios modelos de complejidad computacional escalable basados en técnicas de autocorrelación y predicción lineal, así como una solución basada en el algoritmo "*Matching Pursuits*". El continuo desarrollo en esta línea de trabajo podría resultar en posibles colaboraciones con desarrolladores e investigadores en el campo de NMP. Tras la culminación de la herramienta "*Audio_refiller*", el prospecto ha resultado favorable en la consecución de los objetivos propuestos.

ABSTRACT

For decades, the fast development of Networked Music Performance (NMP) has been taking synchronized online communication a step further when it comes to ensemble musical performance. In the context of the recent COVID-19 pandemic, ensuring audio quality through the Internet was no small task and web-based audiovisual technologies have proven insuficient in terms of tight network requirements, such as latency. Minimizing this parameter is a trade-off that sacrifices transmission robustness, which translates into packet loss. This is where a complementary Packet Loss Concealment solution developed in the present work comes into play.

The software called “Audio_refiller” is meant to mitigate the information gaps due to loss before the packets arrive at the receiving end, previous to NMP software execution. This has been achieved programming several models that escalated in computational complexity, based on autocorrelation and linear prediction techniques, as well as a solution based in matching pursuits algorithm. Continuous development in this line of work could result in collaboration with NMP developers and researchers. At last, the prospect after the culmination of “Audio_refiller” tool has deemed to be favorable in the attainment of the proposed objectives.

1. INTRODUCCIÓN

Esta sección pretende presentar el resto del documento, ahondando en el estado del arte de las aplicaciones NMP y en la idea que motivó este proyecto.

1.1. Antecedentes

La cualidad ubicua de la comunicación en las redes telemáticas, la cual presenta Internet, siendo "la red de redes", ha dado lugar a una abundancia de herramientas y mecanismos de colaboración en línea que han facilitado la forma de vida de millones de personas. Más aún, con la integración de la computación en la nube en dichas utilidades y la posibilidad de acceder a la información desde cualquier lugar que disponga de una conexión a Internet.

Por otra parte, la música siempre ha estado asociada de forma irrevocable a su aspecto colaborativo, considérense formaciones musicales o la relación entre quien produce la música y quien la percibe, aspecto sin el cual no se concebiría esta disciplina artística. No resulta sorprendente que los intentos por casar las redes de computadores y lo musical buscasen acentuar las posibilidades creativas de ambos mundos, en la búsqueda constante de nuevos medios de expresión. Uno de los primeros esfuerzos en esta línea puede ponerse de manifiesto con el ejemplo de la Liga de los Compositores de Música Automática en la década de los 70 [1].

Barbosa [2] define una categorización de ejemplos específicos de sistemas de computación para interacciones musicales:

- Redes locales interconectadas con varios músicos tocando de forma entrelazada a la vez que interactúan con instrumentos virtuales.
- Sistemas de composición de música grupales que permiten el intercambio y edición asíncronas de datos MIDI (*Musical Instrument Digital Interface*), o bien la recreación de salas virtuales en línea para sesiones de grabación en remoto basadas en sistemas distribuidos que se conectan a través de servidores centralizados.
- Entornos compartidos basados en redes distribuidas en los que varios intérpretes realizan experimentos de improvisación con eventos para la participación del público.

- ❖ Sistemas en remoto para la interpretación musical síncrona y en tiempo real entre localizaciones geográficas distantes.

En la literatura, las última de las categorías anteriores se engloba dentro del término *Networked Music Performance* (NMP) *applications* [3], que se traduciría como "aplicaciones de interpretación musical en la red". Pueden ser referidas por una variedad de nomenclaturas, como pueden ser "*music rehearsals on the internet*" (ensayos en internet) [4], "*online jamming*" (improvisación en línea) y similares.

Los investigadores Rottondi et al. [3] exponen que NMP se centra en «reproducir condiciones ambientales realistas para un amplio abanico de aplicaciones, que van desde "tele-audiciones", la enseñanza de música y los ensayos a distancia, hasta *jam sessions* distribuidas, pasando por conciertos en línea». Para ello, las claves están en conseguir:

1. Audio de alta fidelidad (*hi-fi* o *high-fidelity).*
2. Un valor bajo de latencia que se encuentre por debajo de 20 ms, ya que el umbral de tolerancia de latencia para ensembles de música se estima entre 20-30 milisegundos (ms).
3. Un *tempo* musical estable que derive de una baja latencia para poder lograr una sincronización entre intérpretes.
4. Desempeño e interacción óptimas para una sincronización entre intérpretes satisfactoria. [3]

Aun así, hay otros aspectos a tener en cuenta, según los autores, que encierran la complejidad de modelar la experiencia de varios músicos en un mismo emplazamiento físico. Estos factores incluyen la reverberación del sonido en el ambiente y, no menos importante, el acto de "darse la entrada" visualmente unos a otros mientras están tocando simultáneamente, a la hora de compartir el sentido del ritmo conjunto. Resulta difícil, cuando menos, reproducir de forma fiable todas las condiciones del problema, pero la prioridad desde NMP siempre residirá en unos requisitos de red muy estrictos en cuanto a parámetros limitantes como la mencionada latencia y la minimización del *jitter*. [3]

Se empezó a usar NMP sobre redes de conmutación de paquetes en los 90 y lleva captando la atención creciente de la comunidad científica desde entonces. El acceso libre y universal a internet y sus servicios, así como su mejora en cuanto a ancho de banda ofrecido y latencia, ha permitido que los músicos puedan comunicarse conjuntamente a distancia. [3]

Si se compara el contexto de NMP con las plataformas de comunicación de audio y vídeo basadas en web, las cuales también asistieron a un desarrollo exponencial paralelo durante el confinamiento por Covid-19, se alza una diferencia muy condicionante entre ambas. Básicamente, las aplicaciones de videoconferencia están basadas en flujos de datos multimedia (en adelante, *media streams*) *peer-to-peer* (P2P) del estándar WebRTC [5] para audio y vídeo de baja latencia en la web. El problema reside en que los *media streams* no permiten configurar la latencia unidireccional que introducen, que debería encontrarse por debajo de 30 ms como máximo para la interacción entre músicos en tiempo real [6].

La situación expuesta es la razón que Sacchetto, M. et al. atribuyen a que prácticamente todas las aplicaciones NMP sean independientes de estándares o nativas, en pos de una mayor flexibilidad en la configuración. Los autores declaran en sus experimentos la solución implementada en la que han tratado de integrar el software Jacktrip con WebRTC [6].

Las herramientas software NMP son muchas y variadas. Entre los ejemplos más sonados se pueden citar: Jacktrip, Soundjack, Jamulus o LOLA. A continuación, se ilustra un resumen fruto del estudio realizado por Rottondi et al. en 2016 [3] de un diverso abanico de tecnologías de esta índole en las que se detallan sus características diferenciadas:

Authors	Name	Architecture	Network range	Network protocols	Supported data type	Nr. of Audio Channels	Multi-stream synchronization	Codec
Saputra *et al.* [91]	BeatME	Client-Server	LAN, WLAN	UDP or OSC [104]	MIDI	16 (input), 1 (output).	none	uncompressed
Kurtisi, Gu *et al.* [86], [104]	-	Client-Server	LAN	RTP,UDP (stream) TCP (session data)	audio	n.a.	NTP	ADPCM, FLAC (real-time) or MP3, MPEG4 (on-demand)
Renwick *et al.* [92]	Sourcenode	Client-Server	LAN	UDP	MIDI	n.a.	none	uncompressed
Stais *et al.* [98]	-	Client-Server or P2P	WAN	n.a.	audio	2	NTP	uncompressed
Kapur *et al.* [85]	Gigapopr	Client-Server	WAN	UDP	audio, video, MIDI	n.a.	n.a.	uncompressed
Wozniewski *et al.* [97]	Audioscape	Client-Server	WLAN	n.a.	audio	1 (input), 2 (output)	GPS	uncompressed
Sawchuk, Zimmermann, Chew *et al.* [36]–[38], [41], [80]	-	Client-Server	WAN	RTP/RTSP, UDP	audio, video, MIDI	16	GPS, CDMA	MPEG1-4
Akoumianakis *et al.* [82], [105]	Musinet	Client-Server or P2P	WAN	SIP (signaling), RTP (stream), HTTP (text)	audio, video	any	none	OPUS (audio), H.264 (video)
Carot *et al.* [81]	Soundjack	P2P	WAN	UDP	audio and video	8	external master clock	ULD, OPUS (audio), uncompressed or JPEG video
Drioli *et al.* [58], [106]	LOLA	P2P	WAN	TCP (control) UDP (stream)	audio, video	8	n.a.	uncompressed audio and video
Lazzaro *et al.* [93]	-	Client-Server (control) P2P (media)	WAN, WLAN	RTP/RTCP,UDP (stream), SIP (signaling)	MIDI	16	RTP/RTCP synchronization tool	MPEG4
El-Shimy *et al.* [84]	-	P2P	LAN		audio, video	n.a.	n.a.	
Fischer *et al.* [63]	Jamulus	Client-Server	WAN	UDP	audio	2	none	OPUS
Caceres *et al.* [59], [88], [107]	Jacktrip	Client-Server or P2P	WAN	UDP	audio	any	software-based audio resampling	uncompressed
Akoumianakis *et al.* [60], [108], [109]	Diamouses	Client-Server or P2P	WAN	RTP, TCP/UDP	audio, video, MIDI	any	internal metronome stream	uncompressed audio, MJPEG video
Gabrielli *et al.* [89], [110]–[112]	WeMust	P2P	LAN, WLAN	TCP or UDP	audio, MIDI	12	software-based audio resampling	uncompressed or CELT
Meier *et al.* [90]	Jamberry	P2P	WAN	UDP	audio	2	external master clock	OPUS
Chafe *et al.* [87]	StreamBD	P2P	WLAN	UDP, TCP	audio	any	none	uncompressed

Figura 1 Resumen de Frameworks NMP. Fuente: Rottondi

A fecha de este Trabajo Fin de Grado, 2023, la tabla de la imagen superior probablemente incluiría algunas aplicaciones NMP más, como la mencionada JamKazam. Seguidamente, se comentan los aspectos más relevantes de las mismas.

ARQUITECTURA DE RED

La arquitectura de red se concibe centralizada o bien descentralizada. La arquitectura centralizada se trata de un modelo de cliente-servidor (C/S), en el cual el servidor es el intermediario de los datos transmitidos, mientras que la arquitectura descentralizada es un modelo tipo malla (*mesh*) o *peer-to-peer*, en el cual todos los intérpretes han de enviar o recibir datos de todos los miembros de la comunicación, haciendo efectivamente de cliente y servidor.

La primera arquitectura es propia de los sistemas Soundjack o LOLA, en los que los diferentes flujos de datos se mezclarían en recepción y cuyo número de participantes por conexión equivale al número de canales menos uno (el propio), con el problema de escalabilidad que ello conlleva [3].

Por otra parte, Jamulus está específicamente orientado a la arquitectura C/S, la cual encara una complejidad computacional y una necesidad de ancho de banda considerables en el servidor, siendo el encargado de mezclar las distintas fuentes de información multimedia antes de enviarla a destino a cambio de permitir una mayor escalabilidad. Su complejidad tiene como fin mejorar el retardo añadido por el nodo extra en la red, problema que no existe en P2P. Sus requerimientos de ancho de banda pueden reducirse permitiendo *multicast* [3].

Aparte, se encuentra el caso de Jacktrip que permite elegir entre ambos modelos según las necesidades de la comunicación NMP. A fecha de 2022, la plataforma online Jacktrip Virtual Studio, que se contextualizará en el próximo subapartado, emplea tecnología *edge computing* optimizada y basada en la nube para arquitectura de red de tipo C/S, requiriendo de un servidor de audio particular o haciendo uso de los suyos propios. Todo ello posibilita márgenes de usuarios ajustables en las sesiones gracias a su escalabilidad, permitiendo cientos de colaboradores en línea al mismo tiempo [7].

RANGO DE RED

El rango de la red está relacionado con la distancia entre puntos geográficos. Se pueden clasificar las redes por este criterio, siendo destacables las redes de área extensa (WAN), las redes de área metropolitana (MAN) y las redes de área local (LAN). Cabe mencionar que, si se deseara implementar una aplicación distribuida de interpretación en tiempo real dentro de una LAN, solo se recomendaría para testeo, debido a sus limitaciones de distancia o cobertura, velocidad y estabilidad. Sourcenode o BeatME están implementadas para LAN [3].

Por lo general, los escenarios y frameworks NMP están contemplados para redes WAN o MAN. La red WAN es el punto de partida desde el cual se estudian los aspectos de red limitantes, como la latencia, y está contemplada en Jacktrip, Jamulus, Jamkazam, etc. Menos recomendables pero utilizadas son las redes LAN inalámbricas (WLAN), soportadas por soluciones como Audioscape. En otros casos, como el de JamKazam, suele notificar al

resto de usuarios de que una conexión inalámbrica está teniendo lugar y, por lo tanto, empeorará la latencia en la conexión [8].

PROTOCOLOS

En cuanto a los protocolos utilizados por este tipo de soluciones, se basan en el conjunto de protocolos de internet (IP). Entre ellos, UDP (*User Datagram Protocol*) es el estándar de transporte de datos prominente en la mayoría de sistemas NMP, frente al protocolo TCP (*Transmission Control Protocol*). La razón por la cual UDP es empleado en Jacktrip o Soundjack recae en una transmisión rápida sin mecanismos de confirmación de entrega y retransmisión que ralentizarían el tránsito si se utilizase TCP en el flujo de información [9]. Puntualmente, TCP se utiliza para un mínimo control de flujo, como ocurre en LOLA. Además, es posible encontrar alternativas de uso con protocolos de capa de aplicación como los relacionados con el *streaming* o la señalización. Se tiene, por ejemplo, que la solución de Akoumianakis et al. está basada a nivel de transporte en RTP y a nivel de aplicación en los protocolos HTTP y SIP. RTP, el protocolo de transporte en tiempo real, ofrece ventajas de sincronización, mientras que SIP y HTTP sobre TCP se emplean para inicialización de sesiones y datos de texto relacionados con los flujos multimedia, respectivamente [3].

TIPO DE DATOS MULTIMEDIA

El tipo de datos empleados en los flujos multimedia es principalmente audio (Jacktrip, Jamulus), pero también puede darse el caso de soportar MIDI, como en Sourcenode, o bien audio y video, tal como en Soundjack, LOLA o JamKazam. A fecha del 28 de agosto de 2023, Jacktrip 2.0 salió a la luz gratuitamente, integrando video, grabación, *live streaming* y configuración de audio para todo participante en una sesión [10].

FORMATO DE AUDIO

El formato de audio en NMP es variable en unas y otras aplicaciones, yendo desde el uso de audio descomprimido para lograr latencias mínimas y ahorrarse procesos de (des)codificación y (des)compresión, hasta códecs optimizados para el mismo fin.

Jacktrip se engloba en el primer grupo, empleando audio sin comprimir en su apuesta por la alta fidelidad [3].

Por otro lado, está Jamulus, que realiza una compresión de audio mediante el códec de audio de baja latencia OPUS, previo a enviar el paquete de audio por la red y, más tarde, antes de enviarse de vuelta desde el servidor, se realiza otra compresión [4]. También hay otros estándares y códecs multimedia, tales como el vídeo descomprimido empleado por LOLA y Jacktrip, el códec ULD de latencia ultra baja que Soundjack tenía implementado en un principio, MPEG, los estándares MPEG-4 y FLAC por Kurtisi et al., etc.

El mencionado ULD, desarrollado por el instituto Fraunhofer, se encuentra obsoleto según su antigua página oficial, que ya no existe [11]. Según un estudio de 2020 de los investigadores Carôt et al., entre los que se encuentra el creador de Soundjack, este último emplea el códec OPUS con unas modificaciones de bitrate [12].

1.2. Motivación

El estudio y desarrollo del presente Trabajo Fin de Grado nace inspirado en la proliferación de aplicaciones para sesiones de música colaborativa en tiempo real, las ya mencionadas NMP, durante la pandemia del virus Covid-19 en 2020. Este tipo de software ya existía y se investigaba previamente a este contexto, pero vio catapultada su relevancia en las circunstancias de una crisis sanitaria y confinamiento mundiales, con todo el mundo recluido en sus hogares, incluidos los músicos. Expandiendo sobre los ejemplos de aplicaciones que incentivaron el proyecto:

Jacktrip [13] fue implementado como un proyecto de código abierto por el Centro para la Investigación de la Computación en la Música y en la Acústica (CCRMA) de la Universidad de Stanford (julio, 2008) y sigue en continua actualización y revisión. En esencia, Jacktrip es un sistema multi-máquina que tiene por objeto reducir el retardo lo máximo posible en interpretaciones musicales conjuntas vía Internet. Puede manejar tantos canales de *streaming* de audio descomprimido bidireccional de alta calidad como puedan soportar los equipos o la red empleados [13]. Sus usuarios reportan buenas experiencias con la baja latencia que puede llegar a lograr y avalan su gran escalabilidad, que puede observarse en el lanzamiento mundial de "*Sing Gently*" del compositor Eric Whitacre, dirigiendo a tres coros distintos dentro del estado de California en 2021 [14].

Desde JamKazam aseguran que tienen lugar, aproximadamente, más de 100000 sesiones de música cada mes [15]. A fecha de septiembre de 2023, en la comunidad de

JamKazam aparecen 5026 miembros que contribuyentes y un total de 394 como los que se encuentran más activos en línea [16]. JamKazam se ha considerado una herramienta valiosa para el jamming virtual, por su oferta de audio de baja latencia para permitir que los músicos hagan interpretaciones en tiempo real conjuntamente, a pesar de encontrarse con dificultades técnicas en cuanto a conseguir la sincronización y calidad de audio ideales.

Jamulus ganó popularidad durante la pandemia, con miles de usuarios a nivel mundial. Por ejemplo, durante la semana del 26 de abril de 2020 hubo más de 12000 descargas a nivel mundial [17]. Jamulus es apreciado por su naturaleza *open-source* y sus capacidades de mínimo retardo perfectas para el jamming en tiempo real, proveyendo de una experiencia colaborativa satisfactoria para los músicos [18].

Soundjack tiene una base de usuarios entregada, pero no se conocen los números que maneja. Quienes han empleado Soundjack reportan, similarmente a otras tecnologías, una experiencia colaborativa de alta calidad con buena sincronización, aunque se reporte algún caso de quien no ha llegado a conseguir establecer un puente entre las cifras de latencia proporcionadas por Soundjack, que giran alrededor de 50 ms, y la experiencia de usuario en redes MAN y WAN. Este último es un ejemplo de uso realizado por Dannenberg, el co-creador del software *Audacity*, en el cual no se han considerado posibles problemas de jitter, solo el retardo [19].

El mencionado sistema de audio contempla la posibilidad de solucionar el problema de los paquetes de información perdidos y ahí es donde este Trabajo Fin de Grado pretende suplir la necesidad de predecir dichos paquetes como si hubieran sido recuperados de forma que quede disimulado el efecto provocado por las pérdidas.

Puesto que el sonido generado a partir de una interpretación musical ha de pasar por varias etapas desde que se envía hasta que se recibe en destino, es imperativo tener en cuenta que, en cada una de esas etapas, se irá añadiendo retardo que va a retrasar la llegada de audio y dificultar, por tanto, la simultaneidad con que tocan los músicos.

En sucesivos apartados se partirá de dos bases: por un lado, las aplicaciones de interpretación colaborativa en la red y, por otro lado, cómo se ven afectadas por la latencia intrínseca a la conexión síncrona. En la próxima sección se profundizará en el concepto de latencia y por qué es un factor limitante a la hora de transmitir música de forma sincronizada a través de una red como lo es Internet.

2. OBJETIVOS

El principal objetivo de este Trabajo Fin de Grado es crear un sistema capaz de generar audio en los paquetes que se pierden en interpretaciones de música a distancia. Para la consecución de dicho objetivo principal, se hace necesario satisfacer una serie de objetivos específicos, detallados a continuación:

- Estudio del funcionamiento de distintas tecnologías distribuidas para la interpretación musical sincronizada en la red.
- Estudio previo de modelos predictivos basados en la autocorrelación de la señal.
- Simulación de un modelo de canal con pérdidas a partir de la probabilidad de error de paquete.
- Implementación de métodos de estimación de paquetes perdidos.
- Programación de una herramienta software que evalúe dichos métodos.
- Selección de la propuesta de predicción más apta en base a la calidad auditiva lograda.
- Generación de una señal sintética que mejore la calidad perceptual en aplicaciones de interpretación musical en la red cuando la pérdida de paquetes sea significativa.

3. MATERIALES Y MÉTODOS

Este capítulo ilustra ampliamente el proceso y desarrollo del trabajo realizado en tres frentes: Fundamentos, Procedimientos y Sistema Propuesto.

Los Fundamentos tienden un puente hacia las nociones clave para entender este trabajo de forma fructífera. En el ámbito de este proyecto, el enfoque se hará sobre el entorno de la red y lo relacionado con ella. Aparte, se estudiarán algunas cuestiones relativas al procesado de señal de audio. Se tratarán el problema de la latencia, el parámetro del *jitter*, la torre de protocolos en NMP, y se explicarán los conceptos de audio descomprimido, autocorrelación y un compendio de términos necesarios desde el punto de vista musical.

En Procedimientos, tras haber logrado una comprensión de los conceptos necesarios, se plantean y estudian exhaustivamente distintas vías y algoritmos para lograr la consecución de los objetivos perseguidos.

En Sistema Propuesto, se explican el desarrollo y la implementación de la herramienta software "***Audio_refiller***", basada en los algoritmos del apartado previo.

3.1. Fundamentos

3.1.1. Latencia o retardo

Latencia, también referida como retardo (*delay*), es un término vinculado pero no circunscrito a las redes telemáticas. En general, puede ser descrita como el tiempo que se demora la información en propagarse de un punto a otro de la red, siendo una consecuencia de todos los retardos introducidos por los diferentes bloques en un sistema [3].

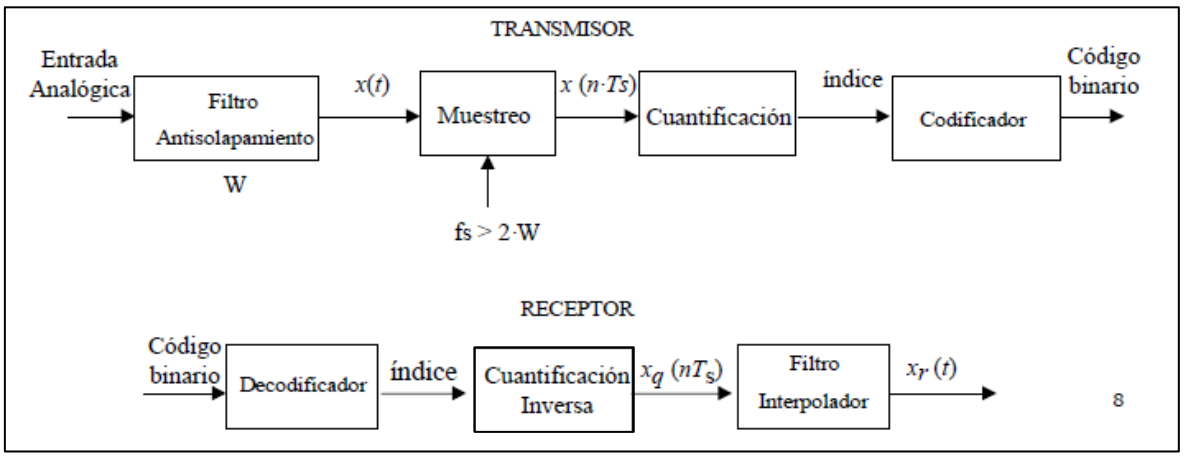

Figura 2 Sistema de transmisión digital basado en un codificador MICD o PCM. Fuente: Cañadas

En la imagen superior, se tiene un sistema de transmisión digital con distintos bloques que realizan funciones específicas [20]. Por ejemplo, en un sistema de codificación PCM (*Pulse Code Modulation*), también conocido como conversor analógico-digital (A/D), que consiste en un filtro antisolapamiento, un muestreador, un cuantificador y un codificador binario. En cada una de esas etapas, se introduce retardo y, al transmitir la señal, se introducirá un retardo intrínseco al canal o medio de transmisión elegido (como la fibra óptica). Análogamente, se sumarán más demoras en los bloques del receptor.

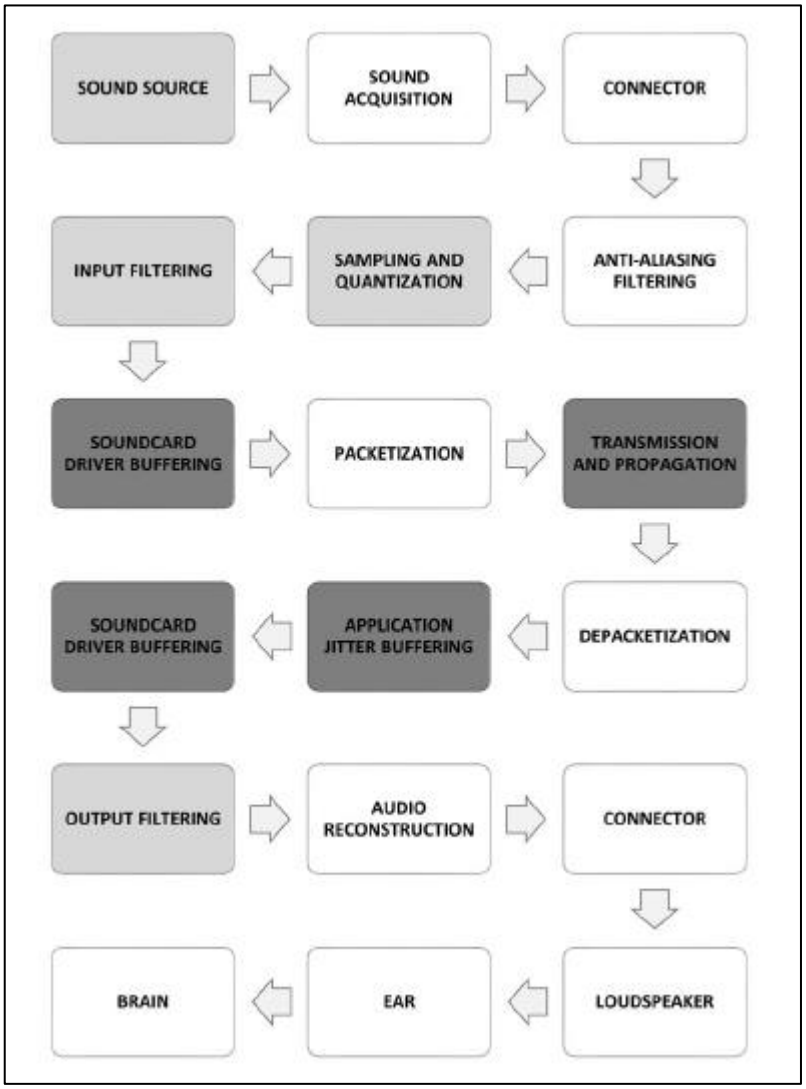

Figura 3 Contribuciones al retardo en un sistema. Fuente: Rottondi

Dejando el proceso de codificación a un lado y profundizando en los bloques específicos al campo del audio en las interpretaciones musicales vía internet, se puede

examinar el sistema de la figura de arriba [3], en el que todas y cada una de las etapas introducen latencia en mayor o menor medida (en escala de mayor o menor sombreado del gráfico), complicando la tesitura. Se tiene también en cuenta el aspecto físico y el electroacústico de la transmisión de audio. Se estudiará el problema con el caso de dos músicos que quieran conectarse desde Valencia y La Palma, respectivamente:

- Por un lado, existe una **distancia geográfica** considerable entre ambas localizaciones, tanto terrestre como marina. Esto se traduce, observando la red de internet mundial, en distintos saltos de red a lo largo de kilómetros de cables de fibra óptica a través de tierra y mar. A mayor distancia, mayor número de saltos en puntos de interconexión y mayor latencia. En un área metropolitana grande se pueden tardar unos 10-12 ms en recorrer una distancia de un punto a otro. [21]

- Otro factor a tener en cuenta es el **plan o velocidad de internet contratada** por cada intérprete en su casa, que se asumirá en adelante que es del tipo *Fiber to the Home* o fibra hasta la casa (FTTH), teniendo en cuenta los datos de junio de 2023 de la CNMC sobre el aumento a 14,3 millones de líneas de FTTH en España [22]. FTTH computaría un retardo de unos 2 ms [21].

- Se considera mala práctica conectarse a través de una red inalámbrica, ya que los routers y demás **equipos de red añaden una latencia considerable**, entre otras cosas. Con lo cual, estos dos músicos habrían de usar un cable Ethernet, que añade alrededor de 1 ms de retardo [21], para conectarse a la red desde sus equipos.

- Internamente, desde el punto de vista electrónico, los equipos empleados por los músicos tienen en su haber una **tarjeta de sonido** incorporada o externa con un conversor ADC/DAC (analógico-digital o viceversa, que son intercambiables según quién transmita y quién reciba la información musical) que introduce *delay* en un rango variable. En estos casos, se recomienda emplear una tarjeta de sonido diseñada específicamente para este fin [21].

❖ Por último, pero igual de importante, se encuentra el **factor acústico**, fruto de la distancia física del emisor humano al transductor electroacústico inverso, o sea, el micrófono. De forma análoga, tiene en cuenta la distancia entre el transductor electroacústico directo, o altavoz, hasta el receptor. Sus valores pueden encontrarse entre 1 y 3 ms [21]. En la figura inferior se muestran los factores electroacústicos del retardo:

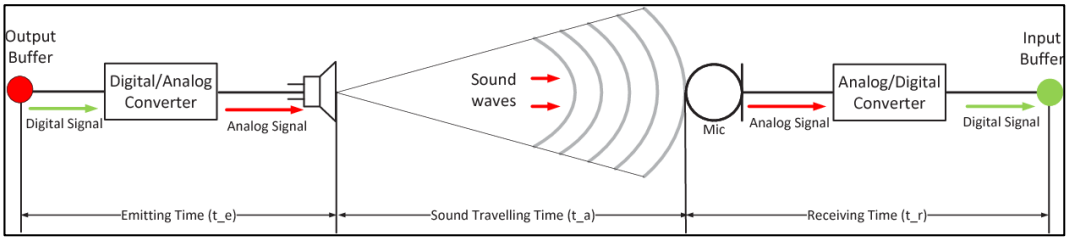

Figura 4 Latencia acústica. Fuente: Le

Cabe preguntarse, incluso después de este análisis de factores limitantes, el por qué la latencia es digna de estudio en el campo de la música en la red. En [21] se describe el comúnmente denominado "Efecto Feliz Cumpleaños", o sea, la sucesión en *canon* de voces descompasadas que intentan cantar al unísono a la vez que escuchan el resto de voces para adaptarse unas a otras. Esto, en términos más técnicos, es un problema de ritmo, de **sentido del pulso** musical (concepto que se explorará en este apartado). La situación descrita ya resulta un problema de forma presencial cuando se trata de una agrupación de músicos, ya sea un conjunto de cámara, una orquesta o una *big band*. Si al factor acústico del tiempo que tarda la música en llegar a oídos del intérprete para tocar al son del resto, se le superponen los factores de latencia mencionados, surge una verdadera dificultad.

A ritmos o *tempi* (véase en el último subapartado de este bloque) más pausados, la latencia afectará de forma más tolerable. A *tempi* rápidos, resultará en una mezcla descoordinada sin pies ni cabeza.

Así, soluciones a las que se han llegado en lo referido a aplicaciones NMP tienen como prioridad el reducir la latencia en cada una de las fases involucradas del sistema de audio, para así reducir el cómputo global, como se visualiza en la imagen.

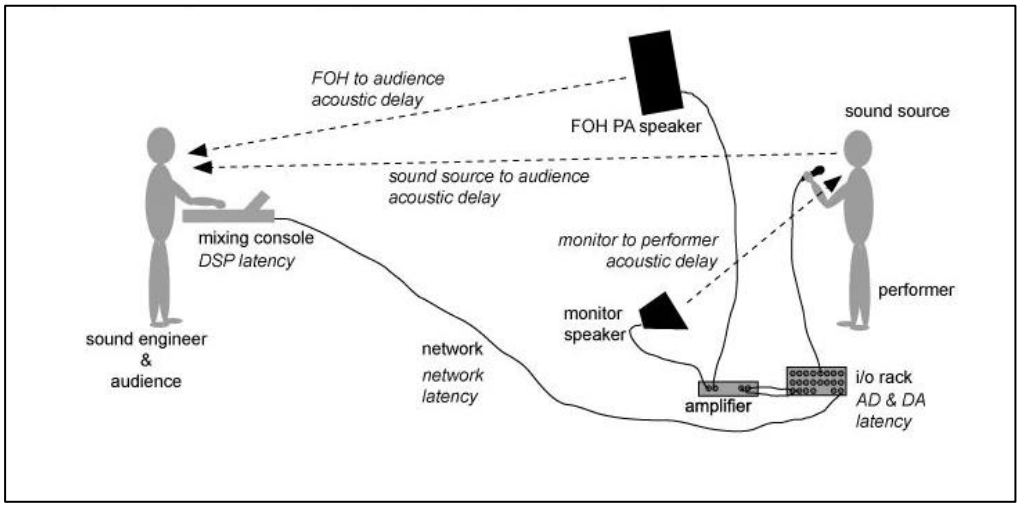

Figura 5 Diferentes tipos de latencia que pueden darse en un sistema de audio conectado en vivo. Fuente: Yamaha

La fundación Jacktrip, que no únicamente el sistema de código abierto, propuso dos soluciones para esta coyuntura: una software y otra hardware. La primera, ya conocida, no es otra que Jacktrip Virtual Studio [23], que deja de lado el aspecto de la comunidad para adentrarse en un modelo de suscripción (con opción gratuita) y distintos beneficios. Este sistema adopta la tecnología IoT "*edge computing*" o "computación en la frontera", que pretende abarcar la distancia entre la nube y los dispositivos del Internet de las Cosas, usualmente relegados a recogida y monitorización de datos, para tener un papel más activo, más "inteligente", acercando la nube, que suele encontrarse en servidores muy distantes. Esto, desde el punto de vista de la latencia, resulta tremendamente útil a partir de la organización eficiente de la red.

Asimismo, si se desea reducir la latencia debida a las tarjetas de sonido, Jacktrip lanzó Jacktrip Analog Bridge para interfaces RCA y Jacktrip Digital Bridge para interfaces USB [24]. Cumple la doble función de reducir aún más el retardo en la comunicación y facilitar la tarea de configuración que muchos usuarios reportan como excesivamente técnica, teniendo en cuenta que entre el público objetivo de las aplicaciones NMP se encuentran músicos, docentes y alumnado sin conocimientos profundos sobre red y configuración hardware/software. En esencia, cumple la labor de

Jacktrip *open-source*, con la configuración facilitada a través de su propia app para *smartphones*, logrando los requisitos mínimos de latencia de red solo con conectar todos los equipos de audio involucrados en el proceso.

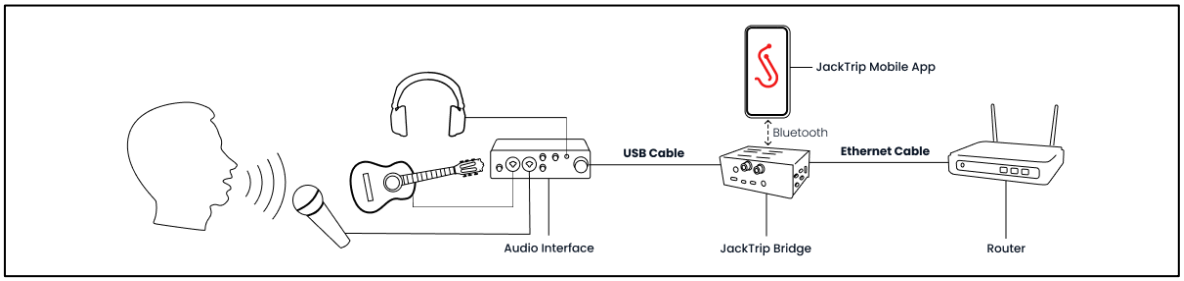

Figura 6 Caso de uso de Jacktrip Digital Bridge. Fuente: Jacktrip

3.1.2. Jitter

Se entiende por *jitter* de red a la fluctuación en el tiempo de llegada de paquetes en una red, de ahí que también sea comúnmente denominado *jitter* de paquetes . En otras palabras, pequeñas variaciones en los intervalos de envío entre un paquete y el siguiente provocan que estos lleguen tarde al receptor. En una interpretación musical en tiempo real, una llegada tardía a su destino supone la pérdida de ese paquete, quedando descartado tras saltarse la marca de tiempo que tuviera programada. Esto se traduce, en el mejor de los casos, en una ralentización momentánea del ritmo musical y, en el peor de ellos, en una pérdida completa del sentido del ritmo de un ensemble de músicos, quedando efectivamente descompasados [3].

En su definición más general, el jitter está asociado al hardware y a la descompensación entre los relojes internos de los equipos al realizar el proceso de conversión ADC/DAC y se suelen clasificar los tipos de jitter en torno al causante de esta fluctuación temporal, ya sea inducido por la interfaz o inducido por el muestreo. El origen del jitter de red se atribuye a congestiones y cuellos de botella en el flujo de información, el empleo de enlaces de banda estrecha en redes WAN, pérdidas debidas al balanceo de cargas, etc. [25]

El jitter en la red está relacionado con la latencia en tanto en cuanto las variaciones en los tiempos afectan a la consistencia y estabilidad del retardo, además

de empeorar el rendimiento de la red. En conjunto, se llegan a provocar distorsiones notables en la información de amplitud, frecuencia y fase de las señales [3].

Anteriormente, se ha hablado de las comunicaciones entre áreas geográficas extensas y cómo afectan a los paquetes. En las redes WAN o de área extendida y, en menor medida, en las redes de área metropolitana (MAN) se precisa de un *buffer* [26], solución que se comenta a modo de curiosidad, ya que no entra dentro del ámbito de este TFG el introducir temas relacionados con la sincronización desde el punto de vista del hardware.

Un *buffer* es una reserva de espacio en memoria para almacenamiento temporal de paquetes, para tomar el problema del *jitter* en consideración. En condiciones críticas, más propensas a error, la cola del *buffer* debe ser lo más baja posible ya que colas demasiado largas incrementan los retardos de transmisión. Como se ha descrito previamente, un paquete que se demora de tal manera en la llegada como para haber colapsado la cola del *buffer* es, a todas luces, un paquete perdido que, eventualmente, causará un hueco en el playback de audio en recepción y cuya ausencia será muy notable si no se mitiga adecuadamente con técnicas de *loss concealment* [3].

A la hora de asegurar unas condiciones óptimas para la sincronización conjunta de agrupaciones de músicos, cada paquete tiene un periodo de vencimiento muy estricto de forma que se satisfagan los requerimientos máximos de latencia y *jitter* en recepción. En resumidas cuentas, un proceso exhaustivo de ensayo y error a la hora de minimizar los parámetros limitantes de la red no va a evitar íntegramente las pérdidas provocadas por los mismos, por lo que ha de tenerse en cuenta dentro de un rango realista, para poder centrarse en otras técnicas de mitigación más avanzadas de pérdida de paquetes [26].

3.1.3. NMP Protocol Stack

Como quedó establecido en los antecedentes, las tecnologías NMP hacen uso de una serie de protocolos sobre los que fundamentan las comunicaciones en la red y sus procesos. Dichos protocolos se integran parcialmente en el modelo OSI, constando de varias capas, posicionadas una sobre otra en forma de pila o torre. Estas

son la capa física, la de acceso al medio, la de red o internet, la de transporte, la de sesión y, por último, la de aplicación [27]. En NMP, el foco se pone sobre los dos niveles superiores sin contar la capa de sesión, ya que los protocolos de acceso al medio y de internet dependen más del entorno de red y quedan relegados a un segundo plano desde este enfoque. Seguidamente, se van a explicar tres protocolos de transporte y dos de aplicación empleados en interpretaciones en vivo.

En primer lugar, UDP es un protocolo de datagramas de usuario de la capa de transporte no orientado a conexión (*connectionless*) implementado por encima del protocolo de red IP [28]. Se emplea en conexiones no fiables, por lo tanto, no garantiza la entrega de paquetes ni tampoco que estos lleguen ordenados. En general, los procedimientos de este protocolo son mínimos a la hora de transmitir mensajes de unos programas a otros en la capa de aplicación [28]. Respecto a los campos existentes en la cabecera de un datagrama, se conforman por los puertos de origen y destino, la longitud del paquete, el *checksum* y la carga de datos en forma de octetos, tal como muestra la imagen de abajo:

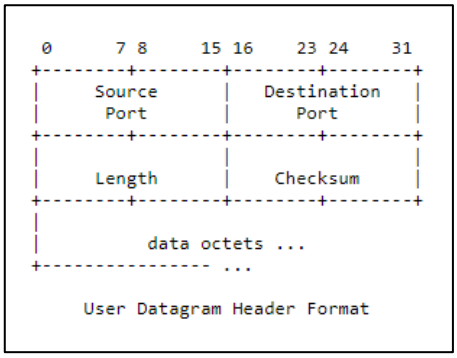

Figura 7 Campos cabecera UDP. Fuente: RFC

Con un uso más reducido en NMP, el protocolo de control de transmisión TCP está orientado a conexión (*connection-oriented*) aunque no tiene capacidad de determinar si algo se está transmitiendo en vivo. Provee de un transporte de datos fiable y ordenado a las aplicaciones. Dicha fiabilidad consiste en una detección de las pérdidas de paquetes y errores a través de números de secuencia y *checksums* por segmento, respectivamente. Además, corrige errores mediante la retransmisión de paquetes [9]. En la siguiente representación, se puede observar la diferencia de tamaño de la cabecera TCP con respecto a UDP.

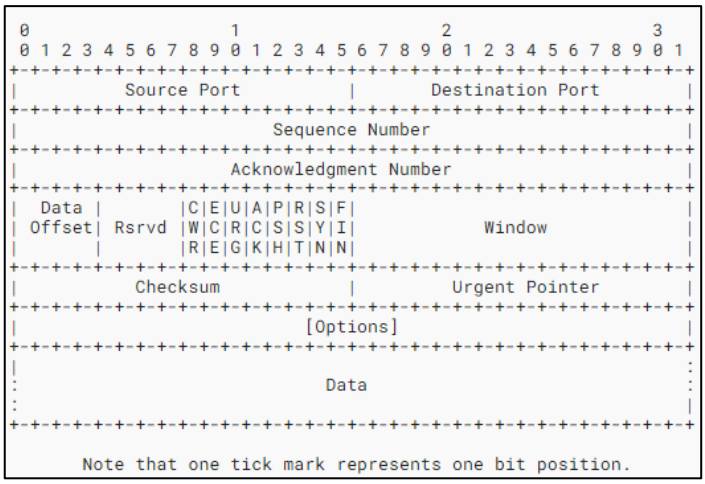

Figura 8 Campos cabecera TCP. Fuente: RFC

Resulta conveniente para este proyecto resaltar las razones por las cuales se prefiere el uso de UDP en aplicaciones en tiempo real y por qué se considera propicio en ciertos casos de uso, como el *streaming* de audio [29] [30]:

- **Menor latencia:** Una de las razones clave por las que UDP es preferido en *streaming* de audio es su menor latencia en comparación con TCP, debido a que no tiene mecanismos de confirmación de entrega ni retransmisión de paquetes, lo que se traduce en un envío más veloz [29], crucial para el envío de paquetes en tiempo real sin demasiados retrasos perceptibles.
- **Menor sobrecarga:** TCP tiene un proceso de establecimiento de conexión y requiere un intercambio de mensajes para asegurar la entrega fiable de datos. Esto implica una mayor cantidad de datos de control y una mayor latencia en general [30]. En contraste, UDP no necesita de ese proceso y, si comparamos las figuras de arriba, tiene menos sobrecarga en el encabezado del paquete, lo que resulta en una transmisión más eficiente para flujos de información continuos.
- **Control de la aplicación:** Cuando se trata de interpretaciones musicales en tiempo real y sistemas de audio distribuido en la red, a menudo es la aplicación (y su correspondiente capa de la pila de protocolos) la que maneja la lógica de corrección de errores y la gestión de la calidad de la transmisión [29]. Si se produjeran pérdidas

de paquetes o errores, la aplicación podría decidir cómo manejarlos con una mayor libertad de configuración de parámetros gracias a UDP.

Además de TCP y UDP, existe otro protocolo de transporte específico para transmisión de datos en tiempo real, el ya mencionado RTP. Los datos que transporta de punto a punto en la red son particulares a este tipo de comunicación (audio, video…). Por otra parte, no reserva recursos ni ofrece calidad de servicio, debido a su propia naturaleza, similar a la de UDP. No es de extrañar que RTP funcione sobre el protocolo de datagramas sin conexión, aunando sus funcionalidades de checksum con las de UDP, además de aportar la multiplexación [31]. Seguidamente, se muestra la cabecera RTP con los campos comunes a todos los paquetes de tiempo real (a excepción de los identificadores CSRC) en la imagen de abajo:

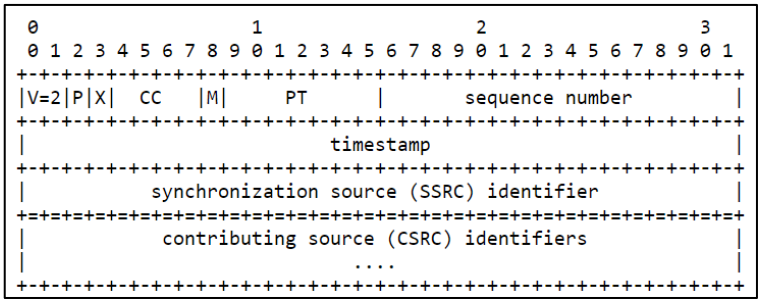

Figura 9 Campos cabecera RTP. Fuente: RFC

Conjuntamente, existe un protocolo de control, RTCP, que funciona al mismo nivel que RTP, monitorizando los datos de transporte de forma escalable y aportando control e identificación de los mismos. Tanto RTP como RTCP no dependen de las capas inferiores [31]. En cuanto a los servicios real-time que proporciona RTP son de identificación del tipo de carga de datos (payload), numeración por secuencia, datación y monitorización de envío. En la última versión de RTP, se mide el momento idóneo para enviar paquetes de control de transporte para reducir el excedente de recursos en transmisión cuando varios músicos se unen simultáneamente a una sesión NMP [31].

Por último, se definen dos protocolos de aplicación empleados en la solución NMP diseñada por Akoumianakis et al.: SIP y HTTP.

SIP, por una parte, es un protocolo de control de esta capa que permite crear, modificar y terminar sesiones con uno o más participantes que incluyen, entre otros ejemplos de sesiones, conferencias multimedia en tiempo real. Para crear una sesión SIP, se precisa de una invitación con descripción de sesión que ayuda a que los participantes puedan establecer acuerdos en el tipo de datos multimedia a enviar [32]. Para lograr esto, SIP emplea servidores proxy que se encargan de encaminar peticiones a donde se encuentre el usuario y que puedan autenticarse, además de tener autorización para los servicios que ofrece el protocolo. Cuando se usa SIP sobre TCP (puede darse con UDP), como es el caso, el primero es responsable de administrar conexiones persistentes o de coordinar TLS sobre TCP [32]. Los campos de la cabecera de un mensaje INVITE de SIP se pueden ver en la imagen inferior. Cabe resaltar el campo 'User-Agent', que describe el equipo origen desde el cual se inició la conexión sobre RTP:

```
INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 2 INVITE
Contact: <sip:alice@client.atlanta.example.com;transport=tcp>
Diversion: Carol <sip:carol@atlanta.example.com>;privacy=off;reason=no-
answer;counter=1;screen=no
Remote-Party-ID: Alice <sip:alice@atlanta.example.com>
P-Asserted-Identity: Alice <sip:alice@atlanta.example.com>
P-Charge-Info: <sip:eve@atlanta.example.com>
P-Source-Device: 216.3.128.12
Content-Type: application/sdp
Content-Length: 151
X-BroadWorks-DNC: network-address=sip:+9876543210@127.0.0.101;user=phone
User-Agent: X-Lite release 1104o stamp 56125 v=0 o=alice 2890844526 2890844526 IN IP4
client.atlanta.example.com s=- c=IN IP4 192.0.2.101 t=0 0 m=audio 49172 RTP/AVP 0 a=rtpmap:0
PCMU/8000
```

Figura 10 Campos mensaje INVITE de SIP. Fuente: Transnexus

HTTP, por otra parte, es un protocolo sin estado que, por defecto, suele funcionar sobre TCP y tiene diversas aplicaciones más allá del hipertexto, haciendo uso de sus métodos de peticiones (GET, POST, PUT…), códigos de error y cabeceras. Una utilización de HTTP involucra la escritura y negociación de la

representación de datos, lo cual garantiza una independencia de los sistemas sobre los datos que transmiten y cómo se presentan. Así ocurre en el mencionado sistema NMP, en el cual los datos de texto sobre los flujos multimedia son independientes del envío de los mismos en la aplicación [33]. HTTP se presenta como dos tipos de mensaje: petición o respuesta. Abajo se presentan una petición HTTP con método GET y una respuesta con un mensaje de estado 200 OK como ejemplo. Se pueden observar claramente delimitadas la cabecera y la carga de datos en la respuesta:

Figura 11Mensaje petición GET HTTP en Wireshark. Fuente: Elaboración propia

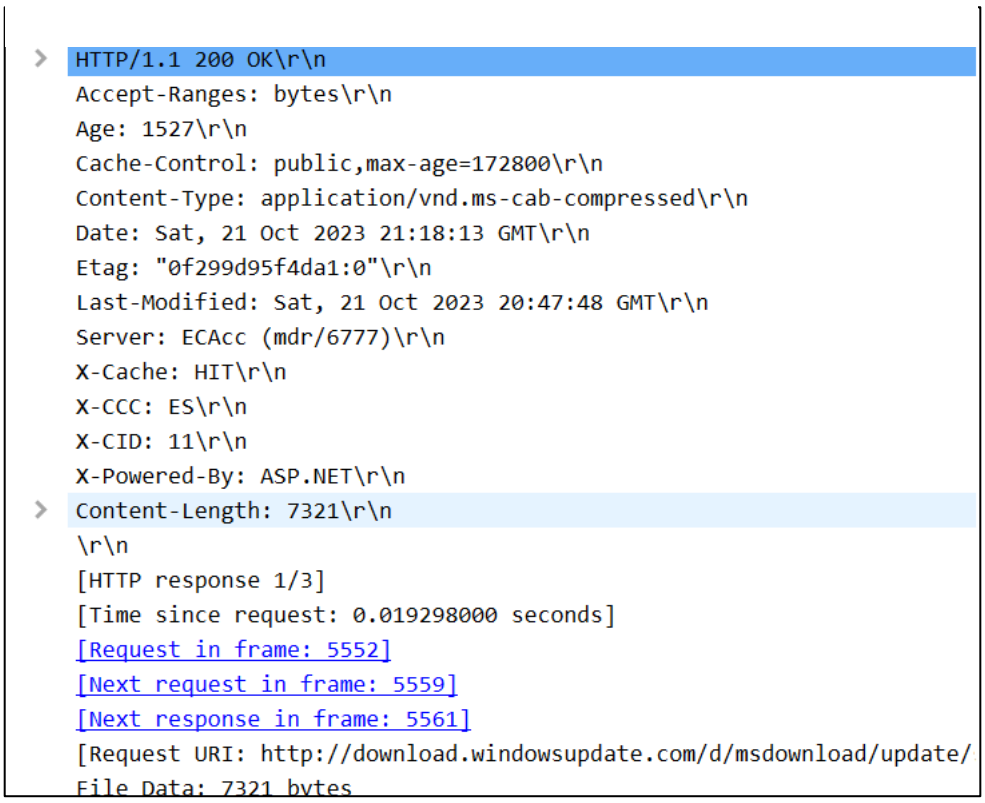

Figura 12 Mensaje respuesta HTTP en Wireshark. Fuente: Elaboración propia

3.1.4. Formatos de audio

Los formatos de audio pueden agruparse en dos categorías: descomprimidos y comprimidos, según el ratio de bits de audio que contengan. Los primeros, como su propio nombre indica, engloban el audio sin comprimir, también denominado audio RAW ("crudo"). Los segundos pueden clasificarse, a su vez, según tengan o no pérdidas en lossy o lossless.

Una forma de evitar que desaparezcan sutilezas cruciales para la calidad del sonido es almacenar los datos de audio como valores de modulación por código de pulso (PCM), es decir, en RAW, sin procesar las cabeceras de metadatos de los paquetes, solo conservando la información relevante para la calidad de la musicalidad [34]. Los archivos RAW relacionados con señales de audio o muestreadas son formatos que pueden ser utilizados por algunas aplicaciones o dispositivos integrados para ingresar, procesar y/o emitir información. La información se almacena tal como está porque el formato es ad-hoc para un solo uso y también porque los formatos de intercambio existentes no pueden conservar la información o corresponden a una determinada organización de la memoria sin decodificación [34].

Generalmente, el aspecto más distintivo de un archivo de audio sin formato real es la falta de firma y metadatos [35]. Este aspecto hace que determinar el formato de un archivo de audio sin formato sea mucho más difícil, así como la determinación de sus parámetros de audio. Aparte, lo más común es que los archivos RAW porten audio sin comprimir, lo que no siempre es cierto, con una profundidad de 4 bits a n bits, una cantidad indeterminada de canales y datos con o sin entrelazado [34].

Concentrándose en NMP y sus ejemplos de uso, se tiene que al menos un formato de audio se aplica a cada categoría: la extensión WAV en audio descomprimido, el códec FLAC en audio comprimido sin pérdidas, y los códecs OPUS y ULD en el comprimido con pérdidas. Se puede examinar una comparativa de sus características clave en el cuadro a continuación:

	WAV	**FLAC**	**OPUS**	**ULD (Ultra Low Delay)**
Compresión	Ninguna (audio descomprimido)	Compresión sin pérdidas (lossless)	Compresión con pérdidas (lossy)	Compresión con pérdidas (lossy)
Profundidad de bit	Varias (8, 16, 24, 32)	Hasta 32	Hasta 32	N/A

Frecuencia de muestreo	Varias	Varias	8 kHz - 48 kHz	8 kHz o 16 kHz
Tamaño de archivo	Grande	Aprox. la mitad de tamaño que WAV [igorsramos]	Más pequeño que ULD	Pequeño
Ejemplo de uso	Archivo, alta calidad	Archivo de alta calidad	Streaming, VoIP	VoIP en tiempo real
Compatibilidad	Amplia	Común	Ampliamente compatible	Aplicaciones en tiempo real

Tabla 1. Comparación formatos de audio NMP. Fuente: Elaboración propia

Ramos [igorsramos] argumenta las diferencias de tamaño entre WAV y FLAC con unas mediciones realizadas para un software voz-a-texto, obteniendo que un archivo FLAC ocupa, de media, aproximadamente la mitad que un archivo WAV. La diferencia radica en que WAV mantiene la información de bit redundante, mientras que FLAC solo contiene información relevante que puede recuperar su estado RAW realizando una descompresión lossless [36]. A más bajo nivel, ULD y OPUS mantienen únicamente la información necesaria de acuerdo a determinados estándares basados en la percepción del oído humano, alterando el material original de forma irreversible [37].

Respecto al formato WAV, también conocido como WAVE, fue desarrollado por Microsoft e IBM y se ha convertido en uno de los principales formatos de audio sin comprimir. Almacena audio a aproximadamente 10 MB por minuto a una frecuencia de muestreo de 44,1 kHz utilizando muestras estéreo de 16 bits, lo que hace que sea el tipo de archivo de audio más grande y también el de mayor calidad, por definición. Esta calidad se puede sacrificar por el tamaño del archivo ajustando la frecuencia de muestreo, el ancho de bit de los datos y el número de canales (hasta 2 para estéreo) [38]. En cuanto a la latencia que puede llegar a introducir, al no estar comprimido, no necesita de procesos de codificación/descodificación para su tratamiento, lo cual, si nos remontamos al esquema de Rottondi et al. [3] [figura X] acerca de los procesos que contribuyen al retardo, supone un ahorro de uso de CPU considerable. Un archivo WAV tiene un peso que acarrea otro tipo de problemas – ancho de banda, por ejemplo – pero el hecho de no contar con metadatos en su cabecera, aunque pueda dificultar el poder extraer sus parámetros, agiliza de igual manera su transmisión. En la siguiente tabla se puede echar un vistazo a la cabecera de un paquete de audio de un archivo con extensión WAV.

Description	Size - Data Type (Windows)	Usual contents
"RIFF" file description header	4 bytes - FOURCC	The ASCII text string "RIFF". mmsystem.h provides the macro FOURCC_RIFF for this purpose.
size of file	4 bytes - DWORD	The file size LESS the size of the "RIFF" description (4 bytes) and the size of file description (4 bytes). This is usually file size - 8.
"WAVE" description header	4 bytes - FOURCC	The ASCII text string "WAVE". Check out the mmioFOURCC macro in mmsystem.h.
"fmt " description header	4 bytes - FOURCC	The ASCII text string "fmt " (note the trailing space). Check out the mmioFOURCC macro in mmsystem.h.
size of WAVE section chunck	4 bytes - DWORD	The size of the WAVE type format (2 bytes) + mono/stereo flag (2 bytes) + sample rate (4 bytes) + bytes/sec (4 bytes) + block alignment (2 bytes) + bits/sample (2 bytes). This is usually 16 (or 0x10).
WAVE type format	2 bytes - WORD	Type of WAVE format. This is a PCM header, or a value of 0x01.
mono/stereo	2 bytes - WORD	mono (0x01) or stereo (0x02)
sample rate	4 bytes - DWORD	Sample rate.
bytes/sec	4 bytes - DWORD	Bytes/Second
Block alignment	2 bytes - WORD	Block alignment
Bits/sample	2 bytes - WORD	Bits/Sample
"data" description header	4 bytes - FOURCC	The ASCII text string "data". Check out the mmioFOURCC macro in mmsystem.h.
size of data chunk	4 bytes - DWORD	Number of bytes of data is included in the data section.
Data	Unspecified data buffer	Your data.

Tabla 2. Encabezado general del formato WAV. Fuente: RingThis [39]

En segundo lugar en importancia, aunque no de uso extendido en sistemas de audio NMP, se encuentra el FLAC, el más rápido y el más ampliamente soportado de todos los formatos lossless comprimidos [40]. Es resistente a errores, es de código abierto, es flexible en cuanto a soporte de metadatos, ampliamente documentados en su página oficial [41].

Dejando a un lado los formatos sin pérdidas, OPUS es un formato de codificación de audio *lossy* desarrollado por la Fundación Xiph y estandarizado por el IETF, diseñado para codificar de manera eficiente el habla y el audio general en un solo formato, sin abandonar una latencia lo suficientemente baja para la comunicación interactiva en tiempo real y una

complejidad asequible para procesadores integrados de gama baja. OPUS es la actualización de Vorbis y Speex para nuevas aplicaciones [37].

Un paquete OPUS cuenta con la cabecera de datos a continuación:

Description	Size	Contents
Magic Signature	4 bytes	"Opus" (ASCII) + null termination (0x00)
Version	1 byte	Version number (e.g., 0x01)
Channel Count	1 byte	Number of audio channels (e.g., 1 or 2)
Pre-skip	2 bytes	Number of samples to skip at the beginning
Input Sample Rate	4 bytes	Original sample rate in Hz
Output Gain	2 bytes	Gain applied during decoding
Channel Map	Variable	Mapping of channels to speaker positions
Stream Count	1 byte	Number of Opus streams in the file
Coupled Stream Count	1 byte	Number of coupled streams (e.g., for stereo)

Tabla 3. Encabezado de un paquete OPUS. Fuente: OPUS[37]

Cuando se habla del códec (no formato) ULD, se está refiriendo al desarrollado por el instituto Fraunhofer. ULD es un esquema de compresión de datos de audio con pérdida que solo introduce una cantidad muy pequeña de retardo en la señal de audio en comparación con los codificadores de audio comúnmente conocidos como MP3 o AAC. Esta propiedad es especialmente útil para fines de comunicación (como llamadas de voz, videoconferencias o hacer música a través de Internet), para lo cual no solo se necesitan altas relaciones de compresión, sino que también es fundamental una baja latencia.

3.1.5. Autocorrelación

La correlación es una herramienta matemática que comprende dos secuencias de señal en cada una de sus dos vertientes: la **correlación cruzada** y la **autocorrelación** [42]. En su vertiente de correlación cruzada, se conciben dos señales diferentes y, en el segundo caso, se tienen en cuenta la señal y su duplicada; es decir, la autocorrelación es un caso especial de correlación cruzada en la que, dadas dos señales x[n] e y[n], $x[n] = y[n]$.

La función de autocorrelación es una técnica de procesado de señal que equipara copias desplazadas de la señal consigo misma. El resultado es una función que muestra cuán parecida es la señal a sí misma.

La autocorrelación de una señal discreta y finita x[n] se define como:

$$r_{xx}[l] = \sum_{n=i}^{N-|k|-1} x[n]x[n-l] = \sum_{n=i}^{N-|k|-1} x[n+l]x[n] \quad (1)$$

Donde i=l, k=0 para l≥0, e i=0, k=l para l<0.

Algunas propiedades importantes de esta herramienta son [43]:

- Si la señal está en fase consigo misma, el producto de ambas y su salida son positivas.
- Si la señal está desfasada consigo misma, el producto puede ser negativo y la salida puede computar valores positivos pequeños o negativos considerables.
- $r_{xx}[0] = E_x$, el valor máximo de la autocorrelación se encuentra en n=0 y concentra la energía promedio de la señal. Si la función de autocorrelación se encuentra normalizada, su valor en n=0 es 1.
- Asumiendo $x[n]\,\epsilon\,\Re, r_{xx}[l] = r_{xx}[-l]$ (2), la autocorrelación tiene simetría par.
- Si x[n] es una señal periódica entonces su autocorrelación también es periódica con igual longitud de periodo que la entrada.

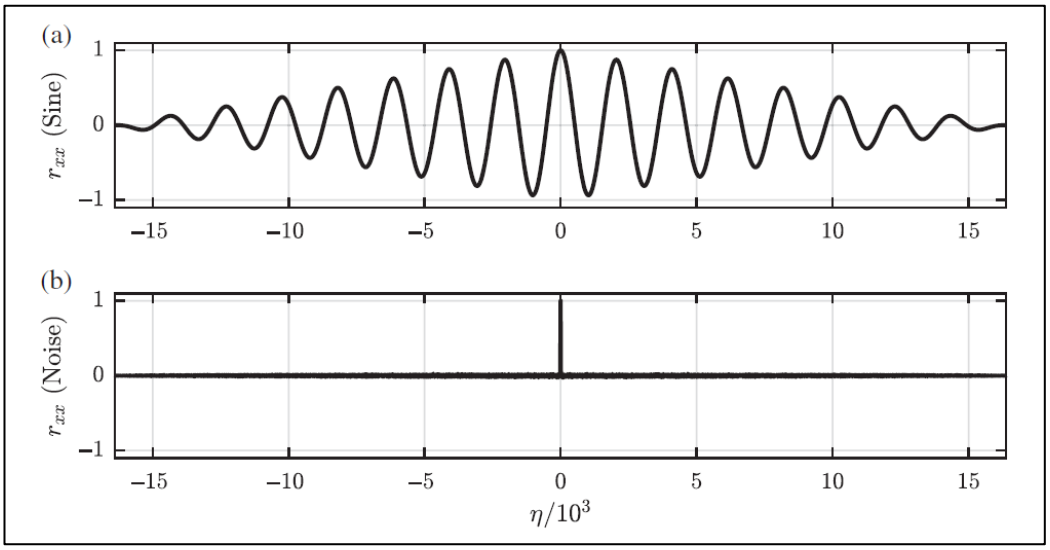

Figura 13 Funciones de autocorrelación de una sinusoide y de ruido blanco. Fuente: Lerch

En el gráfico superior quedan ilustrados dos ejemplos de autocorrelación, $r_{xx}(\tau)$ con desplazamiento τ, de señales continuas definidas en potencia . La primera

es de tipo periódico debido a la propia naturaleza de la función seno, con valores elevados positivos o negativos según la señal se encuentre completamente en fase consigo misma o con un desfase de $\varphi = \tau = \frac{\pi}{2}$ (3).

La segunda, tiene un único valor de 1 en $\tau = 0$. Esto ocurre porque el ruido blanco es una señal altamente incorrelada y, por tanto, solo se asemeja a sí misma cuando el desplazamiento es nulo.

Como se podrá ver en el desarrollo de esta memoria, la autocorrelación es la base de numerosos algoritmos de tratamiento de señal, entre los que se encuentra el de Levinson Durbin, o la base de las ecuaciones de Yule Walker, ambos métodos empleados para resolver el modelo autorregresivo. Como se detallará más adelante, se puede emplear en la predicción de muestras de señal, para encontrar patrones repetidos en la música o para hallar la frecuencia fundamental de una secuencia basándose en su periodicidad, entre muchas otras aplicaciones.

3.1.6. Conceptos musicales: Onset, tempo, beat y ritmo

Puesto que se tocan algunas pinceladas de tratamiento de la señal musical, se hace necesario definir el concepto de *onset* y lo que engloba. ***Onset*** es el inicio de un evento de sonido musical. En el contexto de la música occidental y en el alcance de este Trabajo Fin de Grado, se puede hablar de segmentación en cuanto a la percepción humana del sonido. En realidad, cuando se habla de comienzo de un sonido musical, se concibe como un periodo de tiempo denominado **tiempo de ataque** o tiempo inicial transitorio (*attack time*/*rise time*), o sea, el tiempo desde la primera oscilación instrumental hasta que se alcanza un pico [44].

Lerch distingue entre el tiempo de *onset* de nota (NOT), el acústico (AOT) y el perceptual (POT) que, en ocasiones, se mezclan. Ocurren en ese orden y, desde el punto de vista del procesado de audio, los más interesantes son los dos últimos: el tiempo teórico en el que se puede medir la señal musical y el tiempo en el cual es percibida, resultando la diferencia entre ambos en el tiempo de ataque [44]. Dependiendo de la aplicación, uno u otro pueden ser anecdóticos y resulta conveniente tener en mente que la precisión no es el parámetro a mejorar, sino el desempeño en la detección de picos.

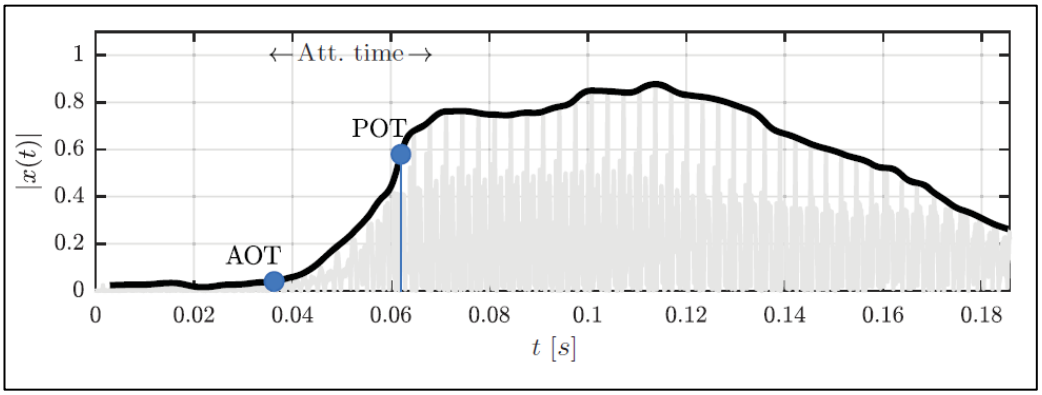

Figura 14 Visualización de una envolvente, el tiempo de ataque, y los tiempos de onset acústico (AOT) y perceptual (POT). Fuente: Lerch

Además del *onset* y sus variantes, cabe subrayar la importancia de distinguir entre tres conceptos musicales más, antes de entrar en el procesado de señal: el *tempo*, el *beat* y el ritmo. ***Tempo*** hace referencia a un índice que mide la velocidad de una pieza musical en un número fijo de pulsos regulares o ***beats***. Su unidad es pulsos/*beats* por minuto [PPM/BPM]. Por otro lado, el ritmo es la agrupación o división sistemática de notas en patrones reproducibles a un *tempo* específico. Por ejemplo, una pieza de música puede tocarse a un *tempo Allegro*, que supone alrededor de 120 ppm, a ritmo/patrón de corchea (♪=120).

Dentro del pulso, podemos hacer una distinción especial para el *downbeat* o pulso fuerte. Este es el primer golpe periódico que existe en un compás, una de las subdivisiones equitativas de una obra basada en un patrón rítmico específico, como podría ser la medida de 4 negras o $\frac{4}{4}$, que queda reflejada en el gráfico a continuación [44]:

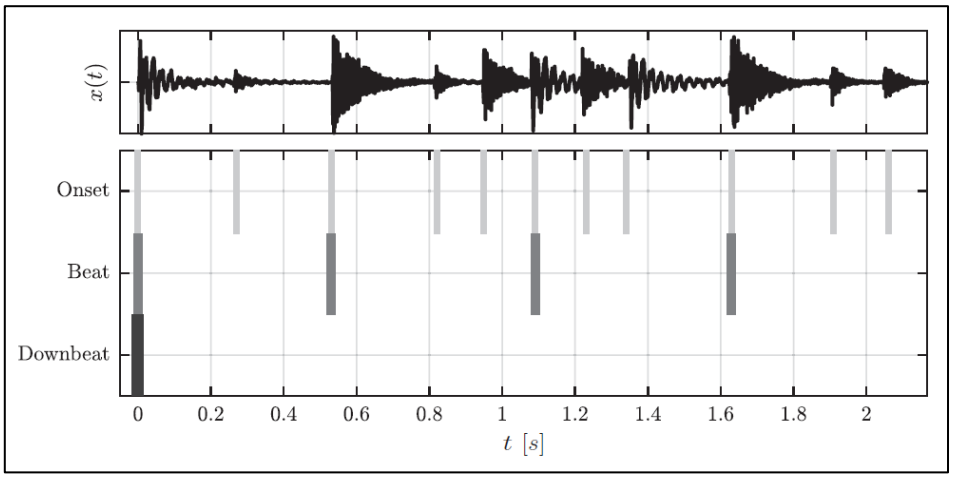

Figura 15 Conceptos de onset, beat, downbeat. Fuente: Lerch

Con todo, incidir en que la música es inherentemente estructural en su percepción y que elementos como las sucesiones o progresiones armónicas, el fraseo (secciones autocontenidas con sentido propio desde el punto de vista musical), cambios en la dinámica o intensidad, o la propia rendición del intérprete, entre muchos otros factores, pueden dar lugar a agrupaciones o límites dentro de una composición musical.

3.2. Procedimientos

3.2.1. Elección de entorno y lenguaje de programación

El presente Trabajo Fin de Grado incluye dentro de sus objetivos la creación de una herramienta software para evaluar los diferentes métodos y algoritmos propuestos. Desde un primer momento, se plantearon dos alternativas de lenguaje de programación: Python y MATLAB.

Por un lado, Python es un lenguaje de programación y scripting con una serie de ventajas muy interesantes si son enfocadas a este proyecto. En primer lugar, su

portabilidad en cualquier sistema operativo sin necesidad de adaptar los *scripts* lo hace atractivo y, más aún, teniendo en cuenta que Python Software Foundation [45] lo mantiene como un proyecto de código abierto accesible a todo el mundo.

En segundo lugar, se considera un lenguaje intuitivo de aprender, con una sintaxis sencilla y consistente, lo cual permite centrarse en el propósito para el que se está desarrollando el software. Dentro de este enfoque, Python dispone de un amplísimo repositorio de librerías que cubren un gran espectro de aplicaciones, desde modelos estadísticos (*statsmodels*) hasta automatización de procesos, y una comunidad en constante crecimiento (es el segundo tag más usado en el sitio de Stack Overflow) [46]. Más concretamente, Python es ideal para implementar aplicaciones relacionadas con el análisis de datos y ML, que se encuentran dentro del planteamiento de este proyecto y que expandiremos en el siguiente subapartado. Por último, pero no menos importante, la versatilidad de este lenguaje permitiría poder contactar con los desarrolladores de Jacktrip y proponer una posible colaboración de cara a futuro.

Por otro lado, tenemos MATLAB, un lenguaje basado en matrices que, con este enfoque, resulta poderoso y extremadamente veloz cuando se vectorizan los datos y se trabaja con las matrices como unidad computacional. Asimismo, se ha recurrido a MATLAB para soluciones prácticas a lo largo del plan de estudios del Grado de Tecnologías de Telecomunicación, lo que lo convierte en un aliado familiar, con un completo repositorio preinstalado que incluye la librería o *toolbox* de procesado de señal y su pormenorizado abanico de comandos para la representación gráfica de funciones. A considerar también dispone de una herramienta interactiva para diseñar una interfaz gráfica de usuario (GUI), entre otras cosas, aparte de un entorno de programación (IDE) propio.

Si comparamos MATLAB y Python, se tiene que, si bien el primero funciona de forma muy rápida y eficiente trabajado desde el punto de vista del álgebra matricial, no llega al nivel de versatilidad y accesibilidad del segundo, que es gratuito y libre. Tomando todos los puntos favorables, se inició este Proyecto Fin de Grado con Python en mente para el trazado de su hoja de ruta, pero se siguió manteniendo MATLAB dentro de los objetivos por su viabilidad. Como Python no cuenta con un IDE de antemano, se instaló la distribución libre de paquetes Anaconda, que dispone

de herramientas como el cuaderno compilable Jupyter o el entorno Spyder, idóneo para la ciencia de datos y su análisis, y con una interfaz muy similar al espacio de trabajo de MATLAB.

Igualmente, es posible tender un puente entre MATLAB y Python gracias a la comunidad generada por cada uno de estos lenguajes y sus respectivos catálogos. Por ejemplo, en Python se puede instalar la librería *Matplotlib*, que está basada, tanto a nivel de comandos como en lo gráfico, en el motor de representación de MATLAB; sin olvidar las archiconocidas librerías *SciPy y NumPy*, que cubren y amplían el ámbito de la *toolbox* de procesado de señal de MATLAB. Todo lo anterior ha resultado de ayuda para la transición de uno a otro. En la imagen inferior se establecen una serie de diferencias en algunas sentencias básicas para cada lenguaje, posibilitando una aproximación a ese puente que se comentaba.

Python vs. Matlab - General Syntax

	Matlab	Python
Element index	1	0
Comment	%	#
Print variable contents to screen	disp(x)	print(x)
Print string	'Hello Everyone!'	print "hello Everyone!"
Find help on a function	help func	Help(func)
Script file extension	.m	.py
Import library functions	Must be in MATLABPATH	from func import *
Matrix dimensions	size(x)	x.shape
Line continuation	...	\

Figura 16 Comparativa de la sintaxis de Python y MATLAB. Fuente: Intro2Python

Como breve inciso, se conoce que MATLAB opera a nivel de vectores y matrices enteras dentro de la clasificación de array [47], que es su forma de representación fundamental. Es decir, a pesar de que se pueden crear variables más reducidas en tamaño que dichos objetos, su unidad base son los arrays en forma de vectores renglón y columna (matriz traspuesta del renglón). Esta distinción se hace porque los vectores están expresados y comprendidos dentro de una matriz.

Retornando a Python, los objetos son su abstracción a nivel de datos. O sea, todo en Python está representado por objetos o relaciones entre ellos, siendo un lenguaje de programación orientada a objetos (OOP) [48]. Si no se entiende Python desde este enfoque, trabajando con clases, métodos e instancias, la experiencia puede no ser fructífera, como ha sido el caso en este trabajo. Python tiene el objeto array unidimensional dentro del módulo array. Sin embargo, no contiene arrays multidimensionales por defecto, ya que estos se encuentran dentro de la librería *NumPy* como *ndarray*. Sí posee, en cambio, las listas, las tuplas, los sets y los diccionarios, que pueden ser clasificados en objetos mutables o invariables, entre otras distinciones.

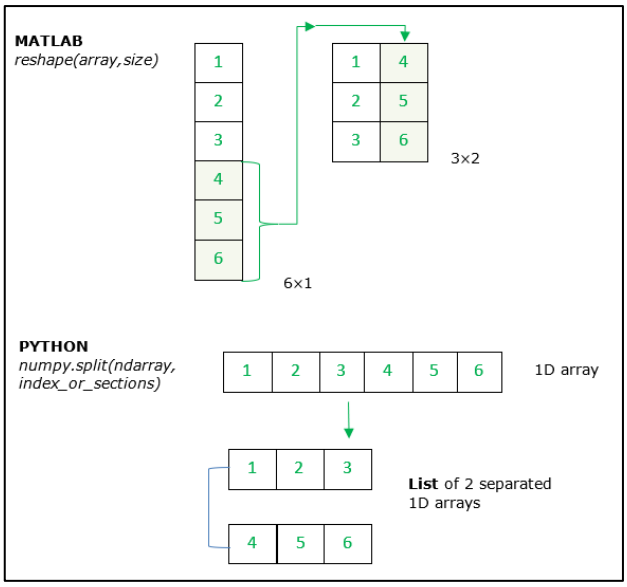

Figura 17 Comparativa MATLAB y Python en la lógica de Arrays. Fuente: Elaboración propia

Véase el ejemplo de la figura superior, que ilustra el significado de trabajar a nivel de array: MATLAB dispone de la función *reshape*, que permite remodelar un array en base a un tamaño especificado. Si se quiere reformular un vector columna de dimensión 6x1 en una matriz de dimensión 3x2, MATLAB toma los 3 últimos elementos del primer array y los dispone en una segunda columna para crear la matriz, rellenando así por columnas. Python, en cambio, toma el array unidimensional de tamaño 6 (no diferencia entre filas y columnas, sino entre índices y ejes) y, llamando al método *split* de *NumPy*, lo convierte en una lista de 2 arrays de tamaño 3 cada uno (esta función permite divisiones de vectores con distinto tamaño).

En lo que se refiere a comenzar trabajando con Python partiendo de una base de MATLAB, dentro del proyecto se han encontrado una serie de dificultades en el transcurso del mismo que han cambiado las tornas que se establecieron en un principio.

Para empezar, se encuentra la curva de aprendizaje de Python, alabada por tratarse de un lenguaje muy legible desde el punto de vista semántico y sintáctico de su código. Puesto que los scripts de las funciones de partida relativas al procesado de señal estaban escritos en MATLAB, había que abordar algo más que las disparidades sintácticas para realizar la "traducción".

Al ir un paso más allá, la verdadera cuestión se encuentra en el funcionamiento y lógica internas de estos lenguajes, así como la existencia de comandos en ambos lenguajes que, a priori y a alto nivel pueden asemejarse debido a sintaxis similar o parece que pueden realizar la misma función, devuelven resultados muy distintos. Este paradigma se refleja en el supuesto de que se quieran correlacionar dos funciones, o correlacionar una señal consigo misma cambiando sus parámetros de escalado, una tarea necesaria para este proyecto.

En MATLAB es posible gracias a la función nativa *xcorr*, que puede especificar si se trata de una correlación cruzada o una autocorrelación variando sus argumentos. Uno de los argumentos de *xcorr* es *scaleopt*, que, si se especifica con el valor '*unbiased*' (literalmente "sin sesgo"), permite desnormalizar la función tomando en cuenta que la correlación en la práctica no es cíclica o circular, sino lineal [https://ccrma.stanford.edu/~jos/st/Unbiased_Cross_Correlation.html].

En Python existen múltiples funciones para el cálculo de la correlación. Por una parte, en el módulo *matplotlib.pyplot* existen *acorr* y *xcorr* para la autocorrelación y la cruzada, pero ninguna de las dos permite calcular la correlación '*unbiased*', más allá de permitir desnormalizar su cálculo. En la librería *statsmodels*, centrada, como su nombre indica, más en modelos estadísticos que en procesado de señal. Cuenta con herramientas de representación del modelo y personalización de funciones y parámetros que escapan del ámbito de este TFG. Por último, tenemos la función *correlate*, que se encuentra tanto en *NumPy* como en el módulo *scipy,signal*, encontrándose más optimizada en el primero que en el segundo debido al uso de la

transformada rápida de Fourier (FFT) en los cálculos, así que se tomó como referencia la correlación más eficiente de NumPy. Como no tiene parámetros de normalización, es la única función que permitía añadir en el código, de forma manual, el término para la correlación normalizada lineal (sin sesgo).

Ello ejemplifica cómo cada pequeño paso que se pueda añadir en el programa en Python puede suponer un retraso considerable en el transcurso y planificación de su desarrollo, al estar constantemente traduciendo instancias de MATLAB y procurando acortar este proceso cada vez. En suma, la experiencia en Python se ha visto dificultada a medida que el programa iba abarcando modelos más complejos de generación de paquetes de audio.

Cabe mencionar que la manera en la que MATLAB y Python realizan operaciones difiere mucho en tiempos de procesado. Mientras que MATLAB puede calcular una autocorrelación y representarla en pocos segundos (o menos) gracias a su potente motor gráfico y su rapidez característica a nivel de operaciones optimizadas con matrices; Python es mucho más lento, y la única librería bien optimizada de las mencionadas es *NumPy* (aunque no al nivel de MATLAB), con su *core* escrito en lenguaje C, conocido por su celeridad.

Como se puede constatar, este proyecto se ha apoyado en el funcionamiento específico de las herramientas proporcionadas por MATLAB y las complicaciones que acarreaba tender un puente constantemente entre este y Python resultó en una transición que no terminó de asentarse. Aun así, el aprendizaje y entendimiento logrado de Python ha ayudado a proyectar otra visión del despliegue de la aplicación final y abre nuevas vías de estudio.

3.2.2. Estudio exhaustivo de los algoritmos contemplados

3.2.2.1. ALGORITMO DE PREDICCIÓN LINEAL

Un predictor lineal es un sistema que se basa en el pasado de la señal discreta x[n], para estimar la muestra actual de la señal, $\hat{x}[n]$, siendo el resultado de la estimación. Para realizar la misma se emplea un filtro solo ceros en el plano z, P(z) [49]. El modelo general para este sistema aparece en la figura siguiente:

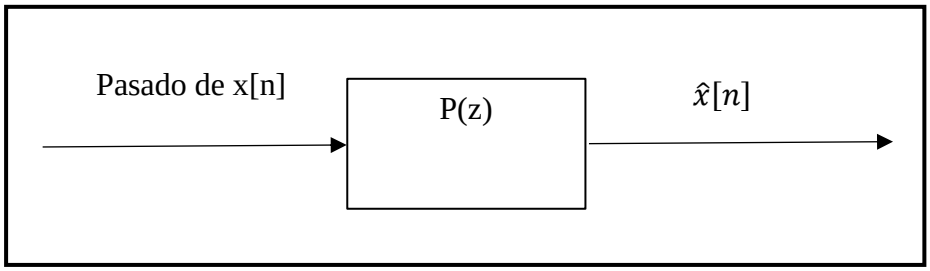

Figura 18 Estimación espectral paramétrica. Fuente: Vera

$$P(z) = \sum_{k=1}^{P} a_P\,[k] z^{-k} \qquad (4)$$

La salida $\hat{x}[n]$ queda desglosada a continuación:

$$\hat{x}[n] = -\sum_{k=1}^{P} a_P\,[k] x[n-k] \qquad (5)$$

Según Oppenheim y Schafer (2011) [50], se puede observar que, para obtener la muestra actual predicha, se emplea una combinación lineal de las muestras anteriores de la señal de partida. El modificador de la ecuación son los denominados coeficientes de predicción de orden P-ésimo, $a_P[k]$.

Teniendo esto en cuenta, se puede cuantificar la fiabilidad del predictor mediante el error de predicción o residuo. Este se expresa como la diferencia entre la señal original, x[n], y la estimada, $\hat{x}[n]$, [50]:

$$e^{+}[n] = x[n] - \hat{x}[n] \qquad (6)$$

Sustituyendo $\hat{x}[n]$ por su valor en la ecuación, queda [49]:

$$e^{+}[n] = x[n] + \sum_{k=1}^{P} a_P\,[k] x[n-k] \qquad (7)$$

O lo que es lo mismo [49]:

$$e^{+}[n] = \sum_{k=0}^{P} a_P\,[k] x[n-k] \qquad (8)$$

El predictor lineal, pues, se puede plantear como un sistema todo-ceros donde la entrada es la señal original y la salida el error de predicción, de esta forma [49]:

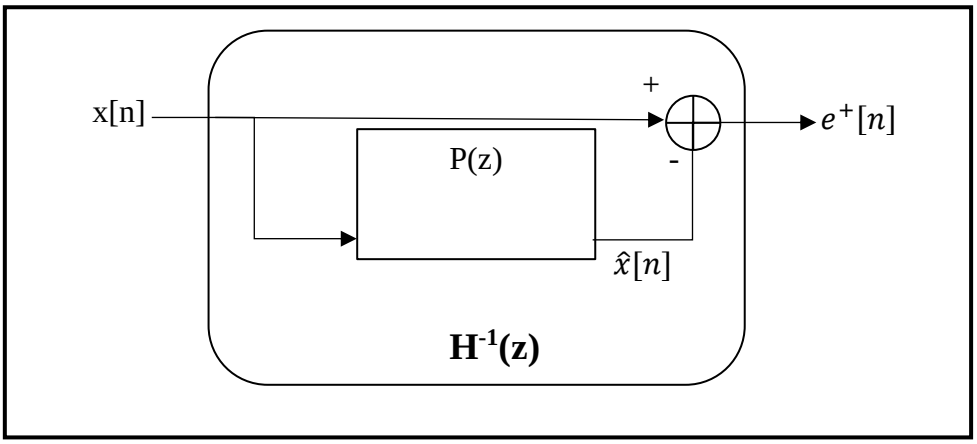

Figura 19 Predictor Lineal como inverso del modelo paramétrico todo-polos. Fuente: Vera

Observando la figura, se tiene que el sistema se corresponde con el filtro inverso de la estimación espectral paramétrica [51], de la cual parte la predicción lineal.

El modelado paramétrico no es más que la representación de la señal mediante un número concreto de parámetros matemáticos. Oppenheim y Schafer (2011) [50] establecen que, si la elección de modelo es adecuada, es posible lograr una representación bastante fiel con un conjunto limitado de parámetros.

El filtro todo-polos parte de su inverso, $H^{-1}(z)$ [50]:

$$H^{-1}(z) = 1 - P(z) = \sum_{k=0}^{P} a_P\,[k]z^{-k}, \text{ siendo } a_P[0] = 1. \qquad (9)$$

Habiendo establecido el paralelismo con la estimación paramétrica, se procede a partir del escenario general para definirla. El escenario general consiste en un sistema lineal como el estudiado en las figuras, en el cual tenemos el filtro H(z) (inverso de $H^{-1}(z)$), [51]:

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{k=0}^{Q} b_Q[k]z^{-k}}{\sum_{k=0}^{P} a_P[k]z^{-k}} \qquad (10)$$

Siendo $b_Q[k]$ los coeficientes del filtro B(z), P el número de polos y Q el número de ceros.

Este escenario general, contando con los polos y los ceros se denomina modelo autorregresivo y de media móvil o *moving average* (ARMA).

Partiendo del mismo, sabiendo que el escenario a estudiar es un filtro todo-polos, se simplifica como [51]:

$$H(z) = \frac{1}{A(z)} = \frac{1}{\sum_{k=0}^{P} a[k]z^{-k}} \qquad (11)$$

Convirtiendo la expresión al dominio del tiempo, tenemos la ecuación en diferencias de la respuesta al impulso, tal que:

$$h[n] = -\sum_{k=1}^{P} a\,[k]h[n-k] + \delta[n]\text{ , con a[0]=1.} \qquad (12)$$

La respuesta impulsiva h[n] depende de h[n-k], sus muestras pasadas, de ahí que se denomine a este modelo todo-polos como autorregresivo (AR), empleado para describir procesos dependientes del tiempo. En el caso concerniente, se han de predecir las muestras actuales de una señal musical de *jamming*, la cual se propaga en la red en tiempo real, basándose en material anterior, por lo que el modelo AR se adecúa a lo que se pretende estudiar.

Sustituyendo $e^{+}[n]$ por su valor en la expresión del error cuadrático medio, E_p^{+} [49]:

$$E_p^{+} = r_{xx}[0] + \sum_{k=1}^{P} a_P{}^{*}[k]r_{xx}[k] + \sum_{k=0}^{P} a_P[k]\, r_{xx}{}^{*}[k] +$$

$$\sum_{m=1}^{P}\sum_{l=1}^{P} a_P{}^{*}[l]a_P[m]r_{xx}[l-m] \qquad (13)$$

Acto seguido, se minimiza el valor de E_p^{+} y resolviendo la derivada del error cuadrático medio respecto a los coeficientes $a_P[k]$, se obtiene un sistema de ecuaciones que, al resolverlo, devuelve [49]:

$$r_{xx}[k] = -\sum_{l=1}^{P} a_P{}^{*}[l]r_{xx}[k-l], k > 0 \qquad (14)$$

El error de predicción óptimo se expresa como [49]:

$$E_p^{+} = r_{xx}[0] + \sum_{l=1}^{P} a_P\,[l]r_{xx}[-l] \qquad (15)$$

Si ejemplificamos todo el proceso de predicción lineal basándonos en el modelo AR con los parámetros que se obtienen de la estimación, se tiene lo siguiente:

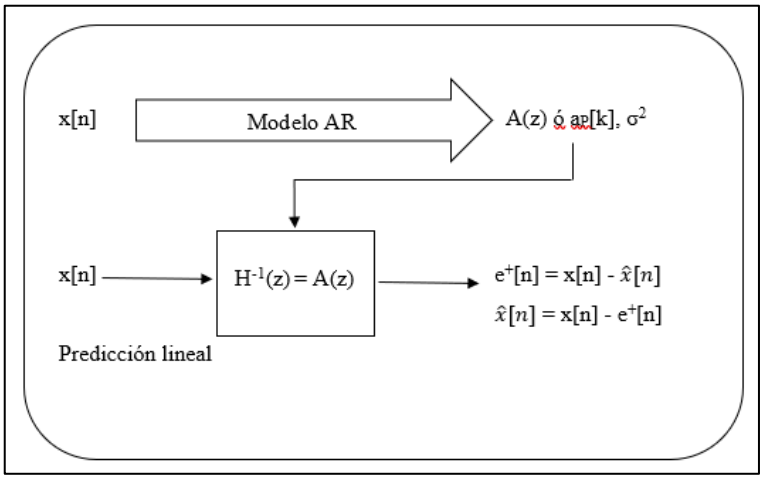

Figura 20 Predictor lineal

Finalmente, operando con el algoritmo de Schur-Cohn, se obtienen los coeficientes del filtro expresado como estructura en celosía (k_P) a partir de los coeficientes de la estructura directa (a_P), previamente calculados con el algoritmo recursivo de Levinson-Durbin [49]. Disponiendo de los coeficientes en celosía es posible discriminar el orden óptimo del filtro, es decir, el último para el cual el sistema es estable, ofreciendo salidas acotadas para entradas acotadas (BIBO, Bounded Input – Bounded Output) [50]. Si a la salida de un sistema con una entrada de valor finito se expulsa una función que tiende a infinito, se generan muestras con predicciones no fiables e involucrando unos costes de diseño que pueden evitarse.

Se indican las ecuaciones de Schur-Cohn [49]:

$$k_P = a_P[P] \qquad (16)$$

$$a_{P-1}[0] = 1 \qquad (17)$$

$$a_{P-1}[\mathrm{k}] = \frac{a_P[k] - k_P a_P[\mathrm{P} - \mathrm{k}]}{1 - k_P^2}, \mathrm{k} = 1, \ldots, \mathrm{P} - 1 \qquad (18)$$

3.2.2.2. ALGORITMO DE VOCODER

Un vocoder es un codificador de voz basado en el modelo de análisis-síntesis. Tal como exponen Martínez et al., el modelo de análisis-síntesis consiste en realizar un examen de la señal, musical en el caso a estudiar, para obtener una serie de parámetros que la modelan, definiéndola. La intención detrás de este proceso de generación de señal no radica en asemejarse a su forma de onda, sino, más bien, a su sonoridad. Esto se logra conformando el procedimiento por el que el oído humano percibe el sonido, que se puede simplificar como una división por bandas de frecuencia que se pueden procesar de manera independiente [52].

La señal de voz puede ser percibida como tal por sus formantes o picos espectrales, que forman una envolvente trazada sobre el espectro de la señal. Si observamos su espectro, se verá que los picos se concentran en un determinado rango de frecuencias, las perceptibles por el oído humano, y la agrupación y posicionamiento de estos formantes constituyen un fonema, identificado unívocamente. El fonema, asimismo, puede ser de tipo sordo – modelándose como ruido blanco – o sonoro – modelándose como un tren de deltas –, lo cual afectará al esqueleto de señal generado mediante el vocoder [53].

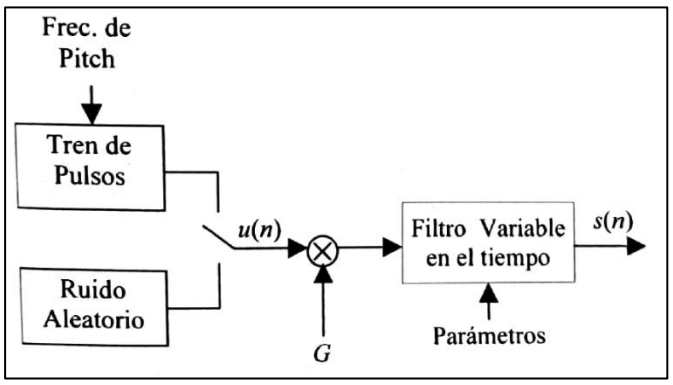

Figura 21 Generación del fonema con modelo de vocoder.

En el sistema de arriba, la salida y la entrada se interconectan en la siguiente ecuación[52]:

$$s[n] = \sum_{k=1}^{P} a_P\,[k]s[n-k] + Gu[n] \qquad (19)$$

Puesto que el empleo de predicción lineal en la codificación del habla ofrece resultados muy positivos, partiendo de una combinación lineal de muestras de señal previas para obtener la muestra de voz actual, se tomarán en consideración las

ecuaciones del algoritmo anterior. Según Martínez et al. [52], «la salida de cualquier sistema depende de la excitación que ataca a éste y de las características de respuesta del mismo». En el sistema a diseñar, la entrada no se trata de ruido (u[n]), sino de un residuo que actúa de excitación del vocoder, e[n], provocando una respuesta, que es la señal. Si se refleja este cambio en la ecuación preliminar y se desprecia la ganancia, se obtiene una expresión general equivalente a la del predictor lineal [49]:

$$x[n] = \sum_{k=1}^{P} a_P\,[k]x[n-k] + e[n] \qquad (20)$$

Es necesario analizar la sonoridad del fonema de entrada para clasificarlo como sonoro o sordo. El fonema sonoro vibra a una determinada frecuencia de pitch que se extrae para generar un tren de deltas equivalente, y para el fonema sordo se modela paramétricamente de forma directa generando ruido blanco [51]. Este residuo clasificado según contenga material sonoro o sordo porta información de las muestras anteriores.

A la hora de codificar la información, el espectro de la señal vocal se caracteriza por tener tres formantes que se ven reflejadas en tres zonas paso-banda. Esto se traduce en que, como mínimo, se necesitarán 6 polos para un modelado correcto de estas señales. Y, a mayor número de polos, mayor precisión tendrá el filtro. Sin embargo, ello resulta en una creciente complejidad computacional en cuanto a diseño y, lo que es más, las mejoras en el modelado no son significativas a partir de 14 polos. Por tanto, lo común en un vocoder es el empleo de 12 polos para el filtrado.

Sintetizando lo anterior, para caracterizar al vocoder basta con extraer, o bien la frecuencia de pitch para el material sonoro, o la potencia de ruido en el caso sordo, la envolvente del fonema a aplicar, la excitación y los coeficientes del filtro LPC. Todos estos parámetros «varían muy lentamente en el tiempo» [52].

Volviendo a la temática central de este estudio, se plantea la posibilidad de cambiar el enfoque del modelo para adaptarlo a una señal musical generalizada. Por una parte, las señales musicales de improvisación pueden contar con una fuente de voz, pero son principalmente instrumentales, como puede ocurrir en el jazz o en *jamming sessions* sobre temas de música pop o rock. Por otra parte, el vocoder está destinado en su concepción a aplicaciones con la voz humana y se basa en la

eficiencia de la predicción lineal para modelar la misma, traduciendo la fisiología del tracto vocal humano.

En lo conceptual, la voz es una señal de naturaleza aleatoria y no estacionaria. A pesar de esto último, es plausible asumirla estacionaria al dividirla en partes más pequeñas, de ahí la segmentación según el tipo de fonema [51]. Asimismo, la señal de audio es de naturaleza aleatoria y no estacionaria, y buena parte de la información que transporta se encuentra codificada dentro de esta última característica, al igual que en la voz [54]. Debido a su carácter armónico y la estructura interna de los sonidos que la componen, la música como flujo de datos también puede seccionarse para ser modelada como si fuera estacionaria según contenga material sonoro o sordo. Internamente, la señal musical es concebida como una suma de componentes sinusoidales o tonos que se denominan armónicos, múltiplos de una frecuencia fundamental [54].

Después de realizar las suposiciones pertinentes que confirman que se puede adaptar un vocoder a otros usos más allá de la voz humana, cabe indagar en los cambios a realizar en su configuración de base. Principalmente, habría de aumentarse el rango de frecuencias a estudiar, ya que el audio abarca un espectro de frecuencias más extenso que el vocal. El rango de la voz humana a nivel conversacional se encuentra en un valor medio entre los 100 y 200 Hz para la población general [55], siendo un valor más alto para las mujeres. El rango de la señal de audio, en cambio, comprende cerca de 20 kHz en total solo para el espectro audible por el ser humano. Además de aumentar la frecuencia de pitch mínima y máxima, habría que focalizar el algoritmo en la detección de fase y no tanto en los parámetros

3.2.2.3. ALGORITMO DE MODELO SINUSOIDAL

El modelo a tratar, tal como en el apartado anterior, está basado en el esquema de análisis-síntesis de señal, con una fase previa que se detallará seguidamente. Puesto que se va a desarrollar en el dominio de la frecuencia, se debe trabajar con tonos que siguen unas trayectorias que se extienden en las muestras de la señal de audio próximas [56].

A alto nivel, el algoritmo sinusoidal comprende una primera etapa de análisis detección de picos espectrales fundamentándose en la herramienta *matching pursuits* (en la cual no se profundizará y se comenta solo como curiosidad) implementada eficientemente con la FFT, donde se realiza un análisis de la señal tonal y una segunda etapa de síntesis. La primera se denomina análisis de tonos. El gráfico siguiente refleja la arquitectura del modelo:

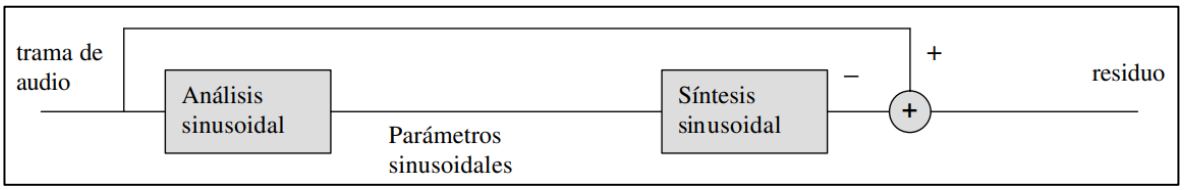

Figura 22 Modelo sinusoidal. Fuente: Vera

Tras la detección de picos, se han extraído una serie de parámetros clave por cada tono – amplitud, frecuencia y fase – que, a continuación, se sintetizan en el tiempo más allá de la ventana de análisis. La información proporcionada por la amplitud, la frecuencia y la fase de una trama de paquetes anteriores se usa, por tanto, para rellenar los huecos erróneos actuales de la señal.

La habitual inercia de la señal de audio hace posible que el modelo de tonos se adapte a las siguientes muestras de la señal de entrada. El modelo sinusoidal se basa en el modelo matemático dado en la siguiente ecuación [56]:

$$x[n] \approx \hat{x}[n] = \sum_{k=1}^{K} A_k[n] \cdot cos(\omega_k[n] \cdot n + \phi_k[n]) \qquad (21)$$

$A_k[n]$, $\omega_k[n]$ y $\phi_k[n]$ se analizan y adquieren de los tonos de cada trama. Vera (2006) [56] pone de manifiesto que, al variar estos parámetros en cada uno de los tonos y en el tiempo discreto (n), se hace necesario el empleo de un banco de osciladores – tantos como frecuencias haya – que realicen la fase de síntesis temporal de la información.

Finalmente, tanto en el proceso de análisis como en el de síntesis, se realizaría un enventanado de cada partición de la señal. En la primera fase, las ventanas a utilizar son rectangulares, que consiguen evitar el solapamiento producido por otras como la triangular, que es del 50%, desembocando en una mayor cifra de tonos por muestra. La única desventaja de la rectangular, en este caso, es la generación de "artefactos audibles" [56] en los límites entre ventanas, problema que se puede

abarcar solapando ligeramente las tramas colindantes al sintetizarlas en el receptor. Ello es posible gracias a la existencia de ventanas trapezoidales que puedan lograr un efecto de *crossfade* (desvanecimiento cruzado) en las transiciones [56]. Como se verá más adelante, el *crossfade* debe utilizarse en todos los algoritmos de generación predictiva de señal tratados en este Trabajo Fin de Grado.

El modelo sinusoidal permite predecir las muestras siguientes de la señal de entrada. Así, en el momento que un paquete erróneo llega al sistema, se completa el hueco con una serie de tonos desde la fase en la que llegó el último correcto.

3.2.2.4. ALGORITMO DE MODELO SINUSOIDAL CON MODELO DE RUIDO

En este apartado culminan los algoritmos sobre los que se construye el software que es el núcleo del presente proyecto. Ello resulta en un vocoder colocado a la salida de un sistema de modelado sinusoidal, tal como se muestra en el esquema de la figura.

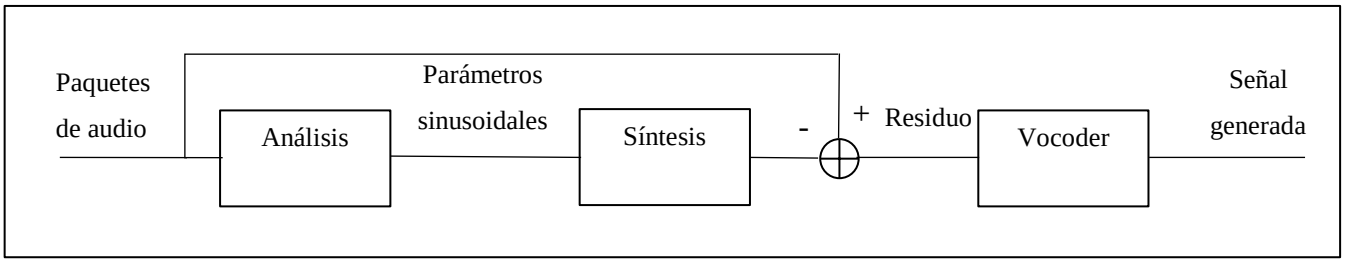

Figura 23 Modelo sinusoidal con modelo de ruido. Elaboración propia

En otras palabras, el vocoder recoge la diferencia entre la señal y el modelo sinusoidal, que da lugar al residuo del modelo sinusoidal. El vocoder se encarga de parametrizar esta excitación y producir un modelo de señal con una sonoridad similar que se insertará dentro del hueco causado por el paquete o paquetes erróneos [49][56].

$$x[n] \approx \hat{x}[n] = \sum_{k=1}^{K} A_k[n] \cdot cos(\omega_k[n] \cdot n + \phi_k[n]) \qquad (22)$$

$$x[n] - \hat{x}[n] = \hat{e}[n] \qquad (23)$$

En la ecuación se muestra el nuevo residuo que se genera, $\hat{e}[n]$, tras sustraer la señal predicha a la señal que llega al receptor. El residuo se modela siempre como sordo en el vocoder, puesto que la excitación vocal se elimina mediante el modelo sinusoidal.

La ventaja de este algoritmo es que permite un mejor modelado de señales compuestas por tonos más ruido, si bien no podrá modelar los transitorios, que en cualquier caso son difíciles de predecir por cualquier método que explote la correlación (o inercia) de la señal de audio.

3.3. Sistema Propuesto: "Audio_Refiller"

Tras haber comprendido algunos de los algoritmos que pueden emplearse en la generación de paquetes de audio para sesiones de música colaborativa en tiempo real y haber estudiado en detalle qué lenguaje de programación se adaptaba mejor a las necesidades de este Trabajo Fin de Grado, ha llegado el momento de presentar y explicar el despliegue de la aplicación desarrollada en MATLAB para este proyecto.

El objeto de estudio es la creación y desarrollo de un software denominado "***Audio_refiller***". ***Audio_refiller*** es, en su naturaleza, un software complementario al empleo de aplicaciones NMP, que opera como una etapa posterior en el equipo receptor. Se encarga de mitigar los efectos que provocan las pérdidas del canal en los paquetes de llegada, reconstruyendo fragmentos de señal a partir de técnicas de PLC. Su funcionamiento está limitado por tasas de error menores del 10% en total y ha sido programado como una simulación, ya que su integración con NMPs no ha sido testada, debido a la flexibilidad de configuración (por la variedad de alternativas existentes) y complejidad de implementación que ello requiere de esta herramienta.

El rol del software propuesto en un sistema de aplicaciones distribuidas de música en tiempo real. El esquema está simplificado a una única parte del intercambio de información, ya que, en realidad, la comunicación es bidireccional y cada usuario final es transmisor y receptor a la vez. Sin embargo, con esta simplificación se clarifica que la ejecución de ***Audio_refiller*** tiene lugar en recepción, antes de que el flujo de audio con pérdidas llegue al usuario final y se

puedan enmascarar, hasta cierto punto, los efectos audibles de la ausencia de información.

La estructura de la aplicación desplegada podría definirse como escalonada en cinco etapas diferenciadas. Se parte de una base que constituye el simulador de canal con errores para muestras de audio y, desde la misma, se irá construyendo el resto de ***Audio_refiller***.

En una primera instancia, el programa parte de una selección de diez *tracks* extraídos de la base de datos MUSDB18HQ [57], la versión descomprimida de la misma. El *dataset* consiste en 2 *subsets* de 150 *tracks* en total separados por fuentes de audio, incluyendo la mezcla principal. Puesto que el propósito del experimento se aleja de lo relacionado con la separación de fuentes musicales, solo se trabajará con mezclas. Para las pistas seleccionadas, solo se considerará la mezcla principal '*mixture.wav*' con voz para la 1 y la 8, para el resto de pistas se sumarán todas las partes instrumentales sin la voz:

	Artista	**Tema**
1	Angels In Amplifiers	I'm Alright
2	Ben Carrigan	We'll Talk About It All Tonight
3	Cristina Vane	So Easy
4	Georgia Wonder	Siren
5	Lindsey Ollard	Catching Up
6	Secretariat	Borderline
7	Signe Jakobsen	What Have You Done To Me
8	Speak Softly	Broken Man
9	The Long Wait	Dark Horses
10	Zeno	Signs

Tabla 4. Temas seleccionados para la evaluación.

Los diversos algoritmos que integran el software emplean lo que se ha denominado como parámetros generales. En el transcurso de esta memoria se distinguirá entre los parámetros (o argumentos de entrada) generalizables a todos los modelos y los específicos de cada uno. Hay argumentos que derivan de otros mediante una operación sencilla, en cuyo caso se especificarán.

Entre los parámetros globales, tenemos:

- La **señal de entrada *mono*** extraída de las diez pistas musicales referenciadas en la tabla superior. Cada una de las pistas se ha acotado en un fragmento de aproximadamente 13 s (o 574281 muestras) por igual.

Posterior a la predicción, se eliminarán los dos primeros segundos (88201 muestras iniciales) de todos los *tracks* generados, para obviar los transitorios iniciales en la escucha. También se eliminarán de las pistas originales, para que, en la evaluación de resultados, todas las señales tengan la misma longitud.

- La **frecuencia de muestreo (f_s)** común a todos los *tracks*.
- El **tiempo en segundos de audio por paquete a transmitir**.
- El **número de muestras en un paquete**, derivado de la frecuencia de muestreo y los segundos por paquete.
- El **número de paquetes** en que se puede dividir la entrada de audio, derivado del número de muestras por paquete y las muestras totales de la señal. (*) Nota: este argumento no tendría sentido en un flujo continuo de paquetes, pero para el presente proyecto se tiene en cuenta simulando una señal dividiéndose en fragmentos manipulables.
- La probabilidad máxima de pérdida del paquete.
- La **tasa de error máxima** que se permitirá en el canal.
- La **máxima longitud de ráfaga de error** en bits, siendo cada bit equivalente a un paquete.

3.3.1. Simulador de canal con pérdidas

Puesto que el modelo en cuestión no tiene parámetros específicos al ser la base del resto de etapas del programa, pasarán a justificarse las elecciones de configuración en esta y los siguientes escenarios desarrollados.

Para empezar, se toma la entrada fragmentada en paquetes de igual tamaño simulando un flujo de música continuo. Basta con que la señal dure lo suficiente como para que se puedan apreciar los errores y su posterior enmascaramiento, además de ocupar un espacio en memoria limitado para facilidad de procesamiento en la recreación de un entorno no real.

Seguidamente, se detallan las variables de partida necesarias para este y el resto de escenarios. Cabe resaltar que se trata de variables configurables e independientes, de las que dependen todas las demás existentes en el script del programa:

- La **frecuencia de muestreo** (f_s) depende de las diferentes pistas de entrada quedando en un valor de 44,1 kHz invariable en todas las instancias del software.
- Los **segundos de paquete de audio** o el **número de muestras a transmitir por paquete** es un parámetro que se ha establecido en 2 ms (0.002 s) y es configurable, basándose en la literatura. Por un lado, para el experimento realizado por Verma et al. (2020) sobre PLC, se trabaja con paquetes de 8 ms (128 muestras). En esta línea, Rottondi et al. (2016) establecen que la paliación de errores mediante sustitución por silencio funciona razonablemente bien para tamaños de paquete típicos de entre 4 ms y 16 ms (dependiendo de la f_s) [3]. Un tamaño de 2 ms o su equivalente de 88 muestras, sobre el cual se ha decidido trabajar, puede resultar pequeño, pero teniendo en cuenta que no se están empleando flujos de audio reales, es suficiente para observar los efectos del canal y la mitigación de pérdidas de cada modelo.
- La **tasa de error máxima** debe ser lo más reducida posible y se ha fijado en un valor de umbral de 0.019 (alrededor de 0.01) para obtener una tasa estimada de errores del 1%. En el caso de buscar una tasa del 5% se fijaría en un valor aproximado por encima de 0.05.
- La **máxima longitud de ráfaga** se ha configurado en un máximo de 4 paquetes perdidos consecutivos. Para tasas de error por encima del 1%, puede darse el caso de que se unan varias ráfagas de error seguidas, dando lugar a pérdidas mucho mayores.

En el diagrama de flujo inferior se puede seguir el funcionamiento del software paso a paso:

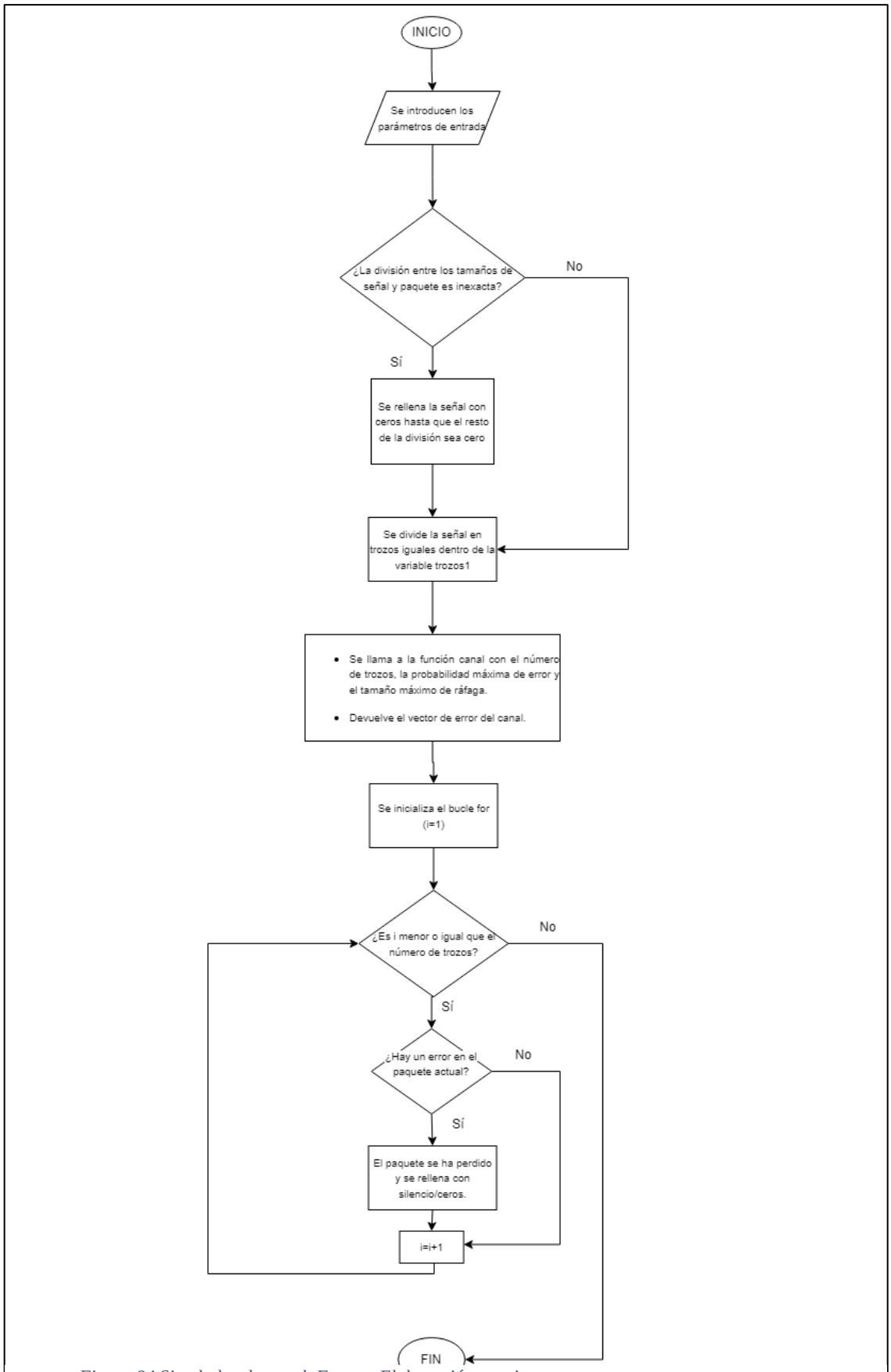

Figura 24 Simulador de canal. Fuente: Elaboración propia

3.3.2. Predictor lineal

El script del predictor lineal se monta sobre el escenario base, equivalente a un canal con pérdidas, configurando las variables que representan sus funcionalidades distintivas.

De esta parte del software derivan dos factores que serán reutilizados en sucesivas iteraciones:

- ✓ Las **muestras de promediado para fading** al inicio y final de cada trama de paquetes. La técnica de fading empleada se trata del *crossfade*, que consiste en una atenuación progresiva de un pequeño tramo final (de longitud variable) de la señal predicha y una acentuación *in crescendo* de un tramo inicial de igual tamaño del siguiente trozo de señal correcto, solapándose. Se ha determinado su valor en 88 muestras, o lo que es lo mismo, la longitud de paquete que se fijó para el conjunto de ***Audio_refiller***. Debe tener una duración adecuada para que el desvanecimiento cruzado haga efecto, evitando, a su vez, los perjuicios de la inserción de material nuevo. Si se prolonga el material generado en la predicción en 88 muestras que se solapan de forma paulatina con el paquete siguiente de material original, tendrá lugar este fenómeno. El promediado se puede apreciar en la imagen a continuación:

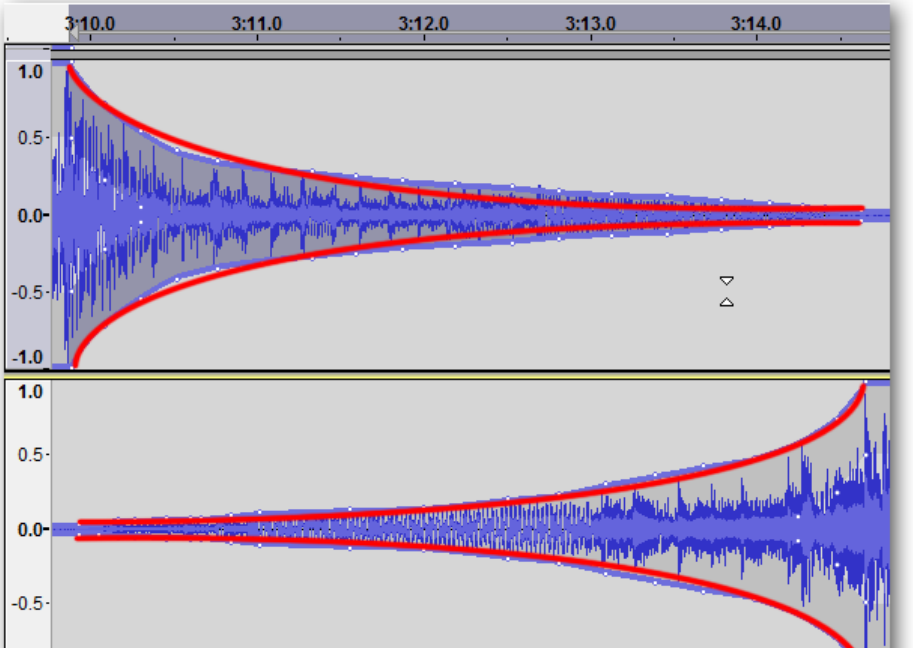

Figura 25 Crossfade en Audacity. Fuente: HowToGeek

- **Número mínimo de trozos o paquetes anteriores a partir de los que se estima.** Su valor se encuentra en 48 paquetes compuestos por 4224 muestras o una duración de 96 ms, que son suficientes para que el predictor pueda estimar con mayor precisión. Si se toma en cuenta que las pistas de audio elegidas tienen una extensión de aproximadamente 7512 paquetes, debe elegirse un valor sabiendo que:
  - ✓ Será el total de paquetes que se desprecien al inicio de la ejecución. Es decir, si se pierde el paquete 11, no se va a poder recuperar, ya que se necesitarían 48 paquetes anteriores que almacenar para la predicción y 11 es inferior a ese número. Supone un riesgo considerable fijar un valor demasiado grande, ya que no solo se descartarían los transitorios iniciales, sino también buena parte de la información que se transmite en la red
  - ✓ Se pueden elegir menos paquetes de estimación en cualquier escenario, pero, si es una cifra pequeña, la estimación no tomará en cuenta variaciones en la señal y podrá no ser acertada, dependiendo del flujo de paquetes en el que se encuentre.

Los parámetros específicos del modelo son:

- El **orden máximo del filtro** de predicción lineal, que se ha limitado nuevamente a 88 muestras, ya que, si se encontrase por debajo, no predeciría a nivel de paquete, sino a uno inferior. Y, si se encontrase por encima, provocaría conflictos entre vectores y su longitud. Debe ser un valor entero positivo múltiplo de la longitud de paquete. En este caso, mide lo mismo que un único ítem.
- **Umbral de valor óptimo para los coeficientes 'k' del algoritmo de Levinson-Durbin**. Este parámetro, que no tiene que ver con los índices 'k' de los coeficientes del modelo AR – aunque sí con el vector de coeficientes y la autocorrelación de la señal, como se indicó en los Fundamentos –, ha de configurarse para que siempre valga menos que la unidad, según el criterio de estabilidad de Schur-Cohn. Cuanto más se acerque al valor de uno, más inestable se volverá el filtro. Lo ideal es que sea una cifra decimal muy

cercana a 1 y correlacionarla con los coeficientes del filtro. Se ha establecido el umbral en 0,999.

El esquema de más abajo refleja cómo opera esta parte de la herramienta software:

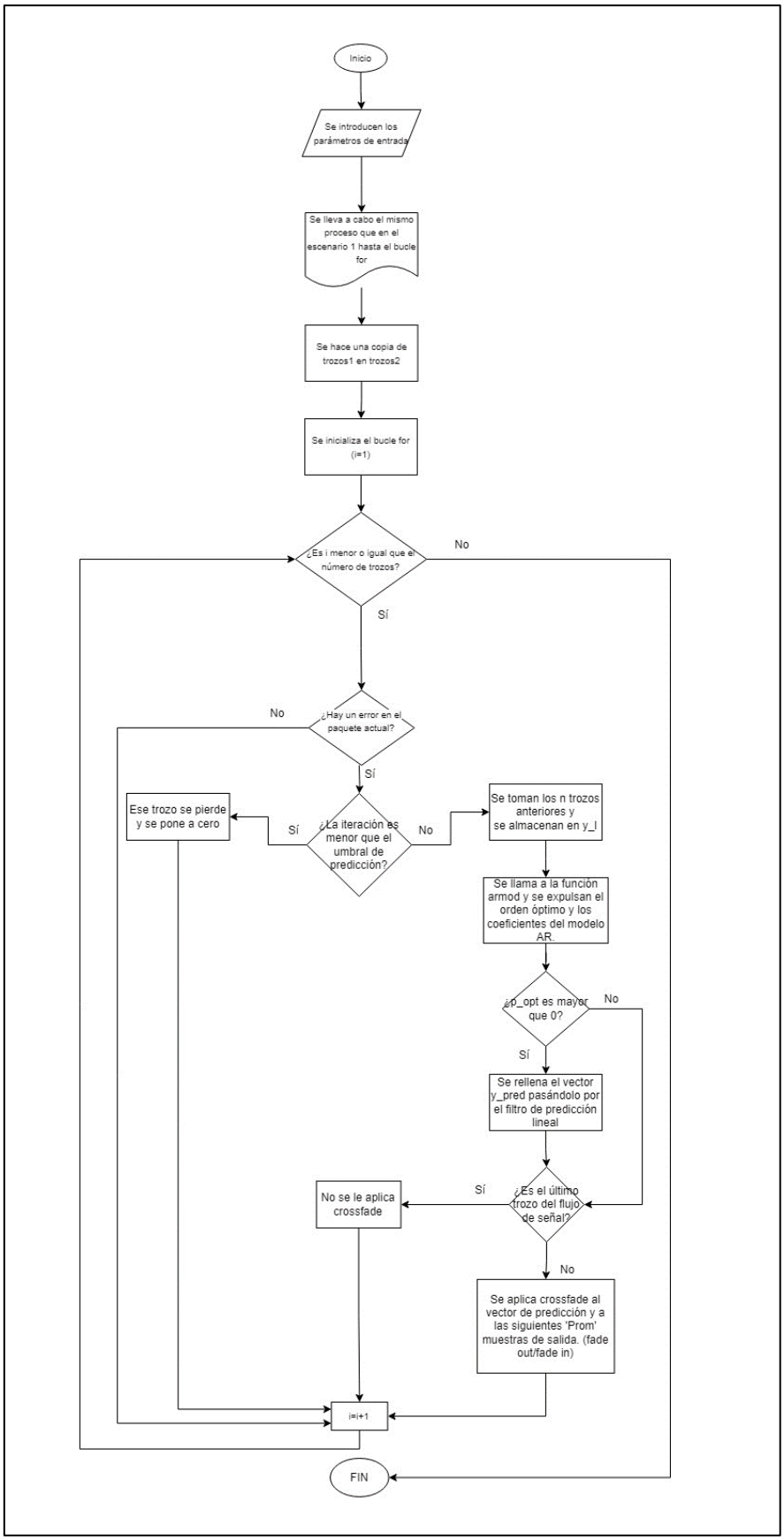

Figura 26 Algoritmo de predicción lineal. Fuente: Elaboración propia

3.3.3. Vocoder

El único parámetro específico del vocoder es el **número de polos** que emplea, cuyo valor se ha establecido en 12, citando a los autores Martínez et al. Como se comentó en los procedimientos, el número de polos mejora la calidad del filtro a la vez que aumenta la complejidad computacional, aunque para más de doce polos no se obtienen mejoras reseñables.

Dentro del bloque que conforma el vocoder, se obtiene primeramente la excitación: los coeficientes a_k del filtro de predicción lineal se extraen a partir de las muestras anteriores de señal, s[n-k], y los 12 polos definidos en las variables iniciales. Una vez conseguidos dichos coeficientes mediante el modelo AR, se define el filtro inverso por el que pasarán los paquetes de entrada. En lugar de calcular la función de transferencia del sistema, H(z), normalmente, los coeficientes A(z) obtenidos de polos se colocan en el numerador del polinomio, permutándose con B(z), que tiene un valor igual a 1, ya que no hay ceros (no es un filtro ARMA).

Hasta aquí, el procedimiento equivale a la ecuación general del vocoder, incluyendo la excitación. Después, esta pasará por un proceso de detección de pitch. De obtener un pitch distinto de cero, significa que existe componente tonal y que, por tanto, se trata de un fonema sonoro que hay que modelar como tal con un tren de deltas de Dirac. De lo contrario, se trataría de un fonema sordo y habría de modelarse como ruido blanco. Esto, en ***Audio_refiller***, se traslada a un tratamiento diferente de la excitación en cada caso:

- Por un lado, en el material sonoro se reparte la excitación en las diferentes frecuencias de pitch, ignorando las doce primeras muestras correspondientes a los polos. Ya repartida, se calcula el material sobrante y sustrayéndolo del periodo total de pitch se tiene la muestra a partir de la cual se perdieron los paquetes. Calculada la fase para la inserción, se concatena la excitación con estas muestras en fase.
- Por otro lado, en el material sordo se realiza la concatenación de la excitación con la excitación invertida como técnica para generar material nuevo de naturaleza ruidosa.

La nueva excitación tratada vuelve a pasar por un filtro con los coeficientes a_k para su codificación. El material a partir del cual se regenera el fragmento de señal

comienza a partir de la longitud de la primera excitación. Lo anterior a este material se emplearía para un *crossfade* inicial.

En el esquema siguiente se desglosa el codificador de voz para generación de señal de audio:

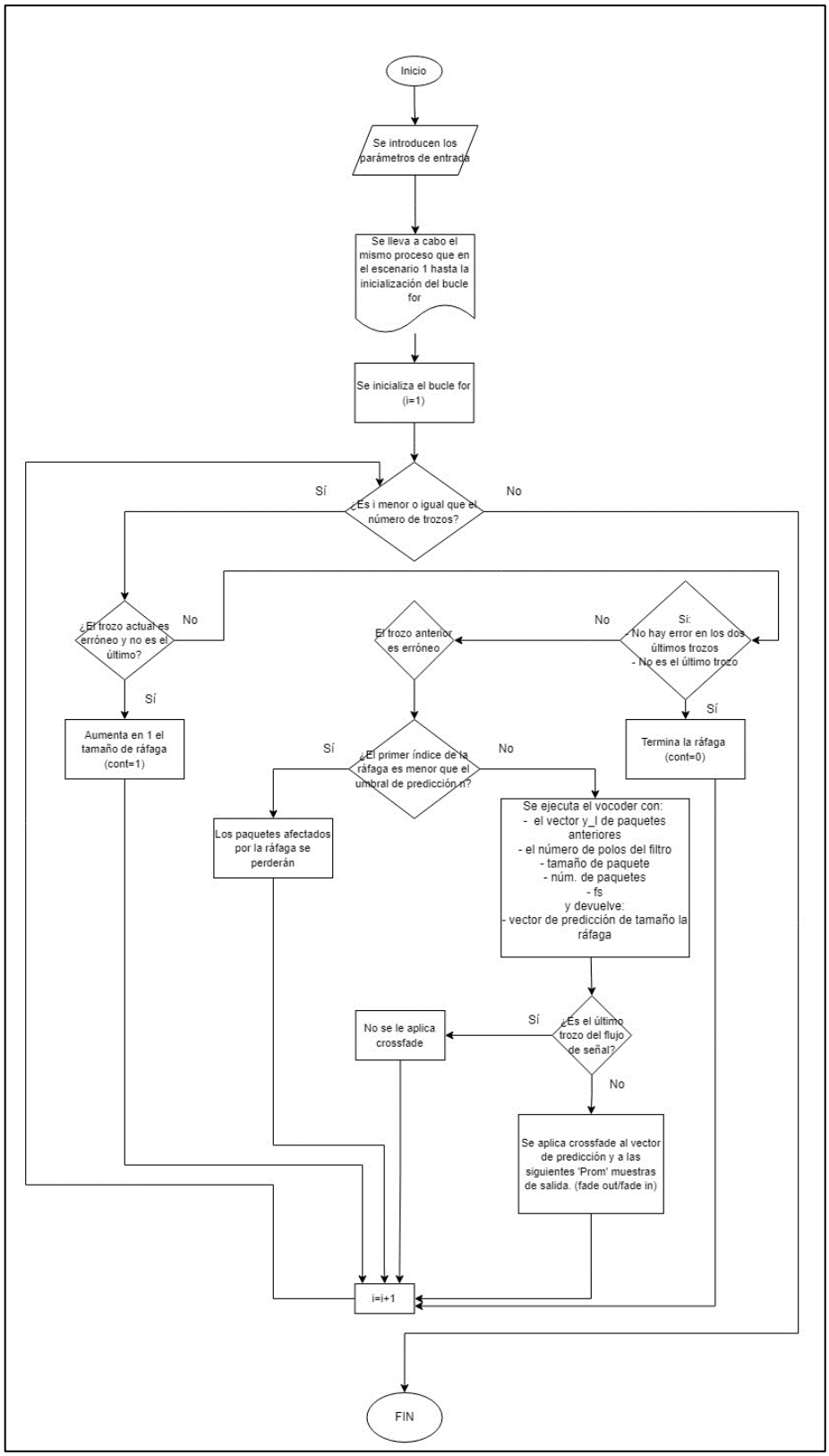

Figura 27 Algoritmo de vocoder. Fuente: Elaboración propia

3.3.4. Modelo sinusoidal

El **número de muestras a inventar** (48*88), que depende del número mínimo de paquetes que se quieran estimar (48) y la longitud de los mismos (88). En total suman el doble de paquetes que se pueden generar (el doble de la frecuencia).

La **longitud en frecuencia** se establece en el mismo número de muestras a generar.

Número máximo de tonos se establece en 700.

Longitud de ventana se establece en el tamaño de paquete.

Abajo se detalla la implementación del algoritmo para el modelo sinusoidal:

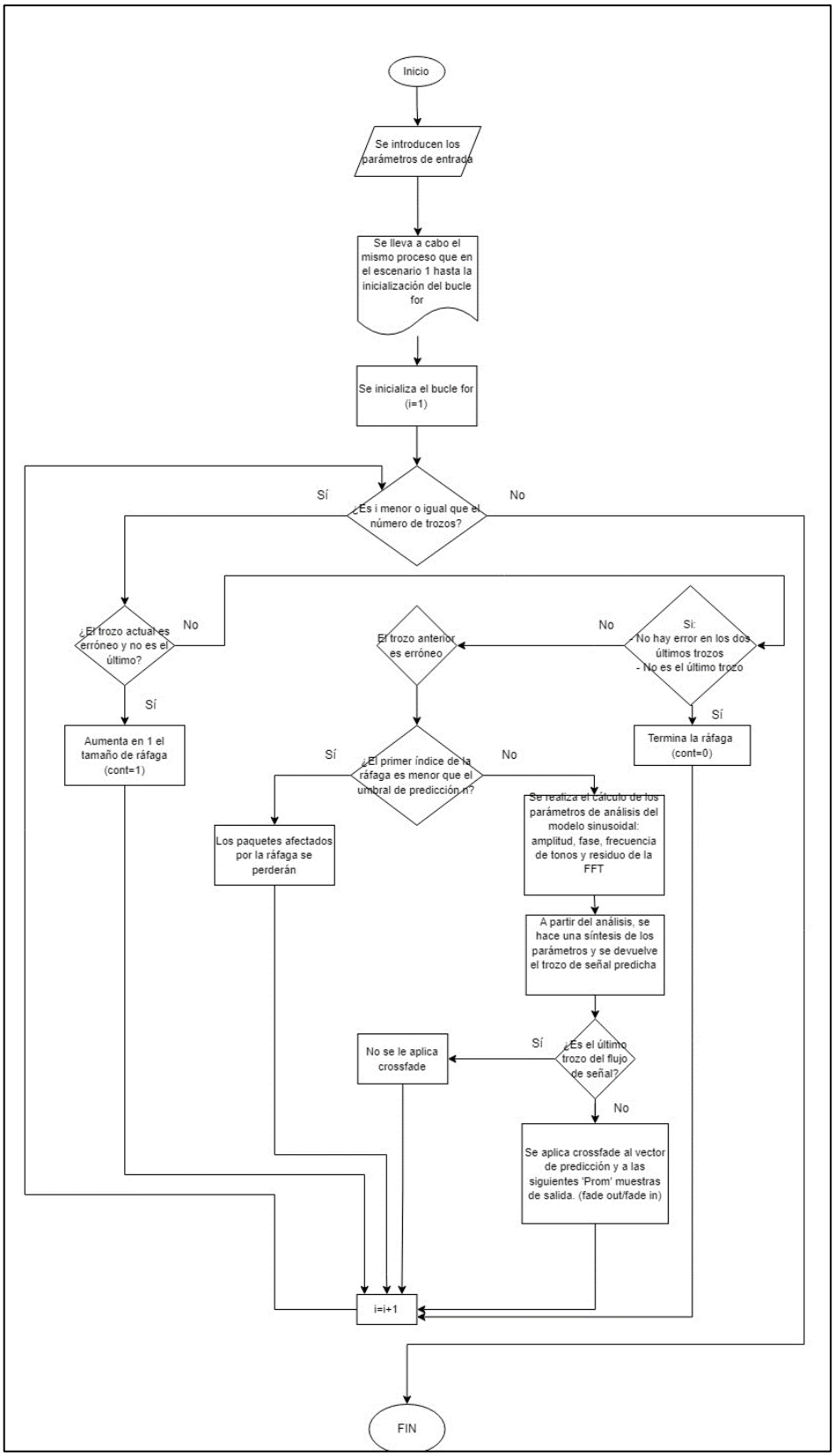

Figura 28 Algoritmo para modelo sinusoidal. Fuente: Elaboración propia

3.3.5. Integración de modelo sinusoidal con vocoder

Los parámetros específicos son un compendio de los dos modelos anteriores. Cabe resaltar que es una mejora sobre el modelo sinusoidal que aprovecha su error de predicción. Este residuo se encarga de excitar el vocoder, para ser codificado como ruido sintético, que, al volver a sumarse con el material estimado, perfecciona su técnica predictiva, acercándose cada vez más a un sonido con menos errores perceptibles.

Se puede ver esta integración de ambos modelos en la imagen inferior:

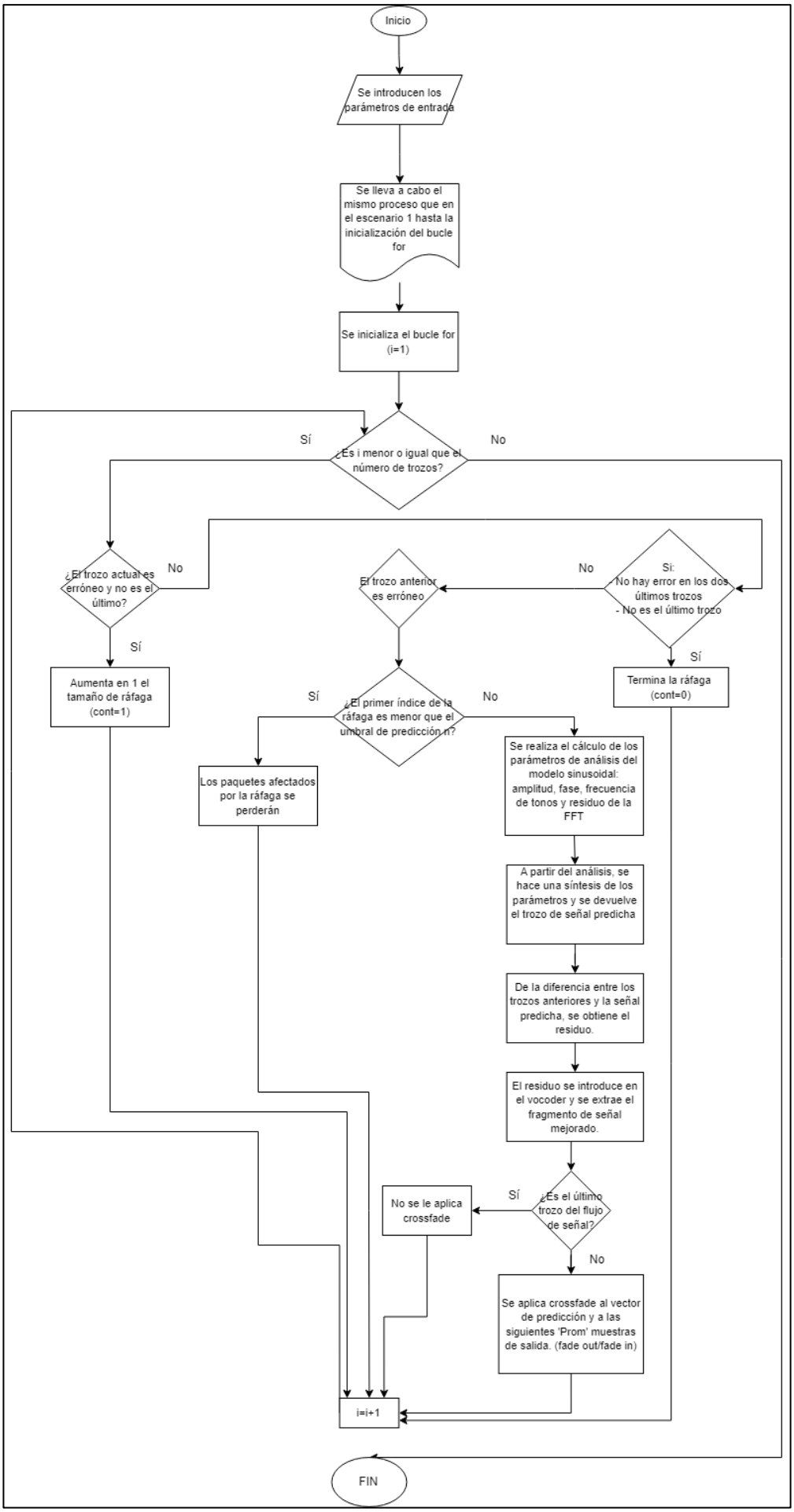
Inicio
Se introducen los parámetros de entrada
Se lleva a cabo el mismo proceso que en el escenario 1 hasta la inicialización del bucle for
Se inicializa el bucle for (i=1)
¿Es i menor o igual que el número de trozos?
Sí
No
¿El trozo actual es erróneo y no es el último?
No
Sí
Aumenta en 1 el tamaño de ráfaga (cont=1)
El trozo anterior es erróneo
No
Sí:
No hay error en los dos últimos trozos
- No es el último trozo
Sí
Termina la ráfaga (cont=0)
¿El primer índice de la ráfaga es menor que el umbral de predicción n?
Sí
No
Los paquetes afectados por la ráfaga se perderán
Se realiza el cálculo de los parámetros de análisis del modelo sinusoidal: amplitud, fase, frecuencia de tonos y residuo de la FFT
A partir del análisis, se hace una síntesis de los parámetros y se devuelve el trozo de señal predicha
De la diferencia entre los trozos anteriores y la señal predicha, se obtiene el residuo.
El residuo se introduce en el vocoder y se extrae el fragmento de señal mejorado.
¿Es el último trozo del flujo de señal?
Sí
No se le aplica crossfade
No
Se aplica crossfade al vector de predicción y a las siguientes 'Prom' muestras de salida. (fade out/fade in)
i=i+1
FIN

Figura 29 Algoritmo para modelo sinusoidal con modelo de ruido. Fuente: Elaboración propia

4. COMPARATIVA DE RESULTADOS

En el transcurso del Proyecto Fin de Grado se ha tenido en cuenta una selección de métodos que se consideraban adecuados para la tarea de generar paquetes de audio perdidos en la situación de partida. Es aquí, en el presente apartado, en el cual desembocan todos los esfuerzos por encontrar la propuesta más apta en cuanto a la calidad sonora deseada, habiendo desplegado una herramienta software diseñada para tal fin. Se relata, en esta línea, cómo se han extraído los datos de salida del software para su análisis y estudio comparativo.

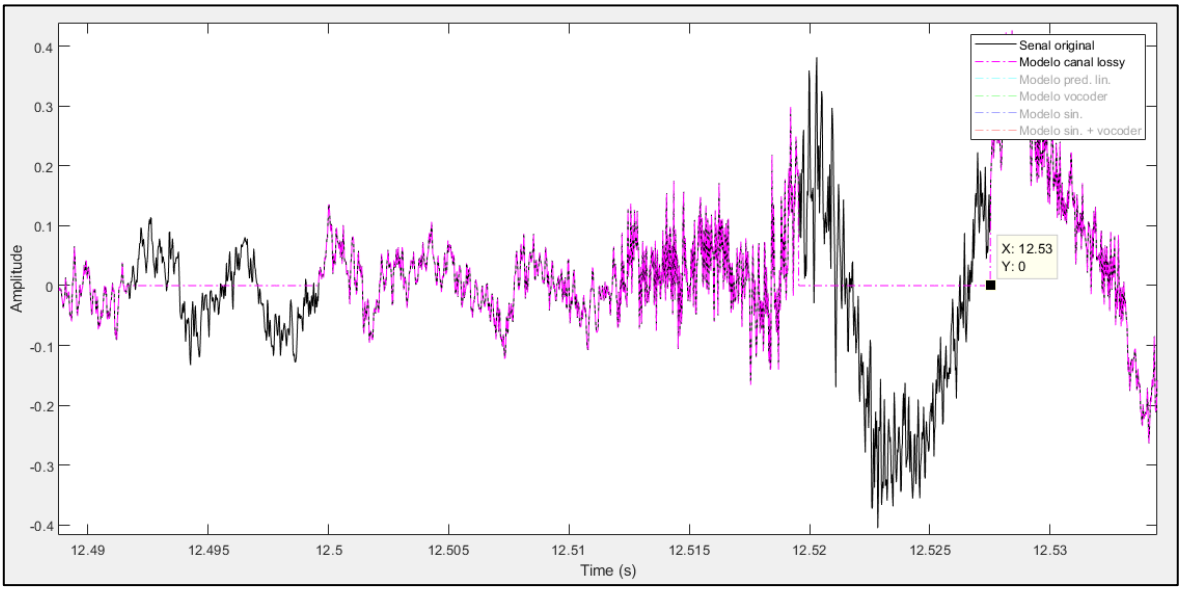

Figura 30 Modelo de canal con pérdidas frente a la señal original. Fuente: Elaboración propia

En la figura superior, se toma un fragmento de la ejecución del simulador de canal en el que se han perdido los paquetes de audio que se encuentran en las marcas de tiempo de 12,49 a 12,50 segundos y de 12,52 a 12,53 segundos, aproximadamente. Se contraponen las gráficas de este primer escenario (magenta) y de la señal original (negro) y se observa que, efectivamente, el primer escenario no realiza ninguna acción sobre el material perdido y simplemente inserta silencio o pone a cero estos huecos.

Se trata de una implementación en la que la tasa máxima de error en el canal está configurada en un 5%, con lo cual el número de pérdidas es importante, dando lugar a una gran frecuencia de huecos en el sonido. Estos, además, pueden tener una mayor extensión debido a una convergencia de varios intervalos erróneos, como se explicó en los parámetros del modelo de predictor lineal. Dado el caso de una tasa de error alta, aunque se delimite el número de paquetes erróneos seguidos en el canal, es posible que dos o más ráfagas coincidan consecutivamente, dando lugar a una ráfaga de gran magnitud.

Si se mide el primer tramo erróneo en muestras, se obtienen 352, que, divididas entre una longitud de 88 muestras por paquete, se tiene que se han perdido 4 paquetes. En el siguiente tramo se han perdido el mismo número de muestras y paquetes, llegando al máximo de paquetes perdidos con una separación de menos de 30 ms entre un fallo y el siguiente.

En cualquier caso, un índice de error por encima de un 1% obstaculiza la audición y, por lo tanto, la interpretación en conjunto, teniendo en cuenta tanto el marco teórico, como el perceptible (práctico). Se puede extraer del análisis que la sustitución de pérdidas por silencios puede encubrir las primeras hasta cierto punto si hay una separación considerable entre paquetes erróneos en el canal y su longitud es, idealmente, de uno o dos paquetes a lo sumo. Seguirán escuchándose pequeños desajustes, pero, al tocar en grupo y mezclar las distintas fuentes de audio musical antes de llegar al receptor, es posible que los intérpretes no reparen en la presencia de estos "clics".

Consolidado el escenario base, se efectúa una evaluación de los modelos subsecuentes llevados a la práctica justificando la figura inferior:

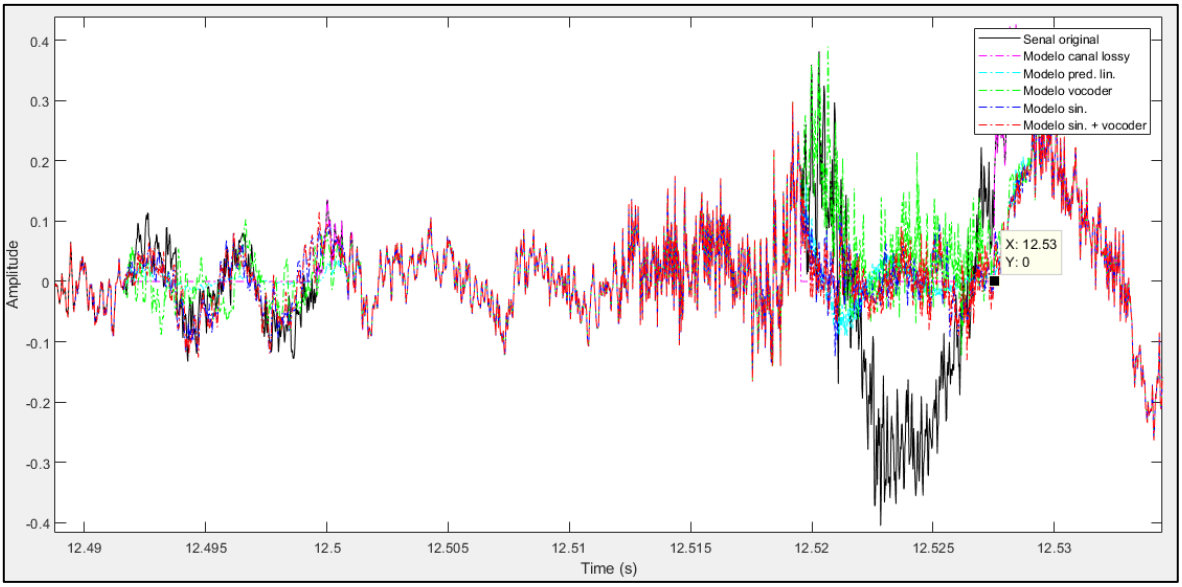

Figura 31 Comparativa de los modelos implementados. Fuente: Elaboración propia

En ella se disciernen los cuatro escenarios que se construyen sobre el canal *lossy*, siendo: el predictor lineal (cian), el vocoder (verde), el modelo sinusoidal (azul) y el modelo sinusoidal modificado con modelo de ruido (rojo).

4.1. Análisis del predictor lineal

El comportamiento del predictor en el primer tramo podría describirse como uniforme. Parte desde el primer cero del canal con pérdidas y continúa en la trayectoria creciente de la señal original sin llegar a acercarse, con poca variabilidad. En general, la

predicción depende en gran medida de las muestras previas. Antes de los paquetes perdidos, se puede observar la tendencia de la señal original y cómo eso se traduce en el escenario en una suerte de promedio de lo anterior. Se puede decir que se adapta a la dirección de la señal de partida, según tenga pendiente positiva o negativa.

En el segundo tramo también hay un cierto seguimiento del descenso de la señal de base, para después cambiar a un curso creciente, abandonando la caída escarpada del material principal. Continúa variando ligeramente de forma cuasi-periódica, propiciada por la autocorrelación. En el descenso abrupto que se mencionaba, cabe destacar que ningún modelo consigue anteponerse a este transitorio, ya que no hay información previa en la que basarse, y todos los modelos están limitados en este sentido, sean más o menos precisos en su adaptabilidad.

El modelo AR (en lo que respecta a este trabajo, es lo mismo que llamarlo modelo de predictor lineal) se apoya principalmente en la autocorrelación y en el cálculo de los coeficientes mediante los algoritmos pertinentes. Está limitado por los coeficientes y el valor óptimo del filtro, que puede variar radicalmente, ya que su cálculo implica las muestras actuales frente a las pasadas de la autocorrelación. Si distan mucho entre sí, dará lugar a valores de k (para estructura en celosía) mayores que uno, teniendo que seleccionar el último valor del filtro antes de que se vuelva inestable.

La forma en la que se va desvaneciendo progresivamente cuando el tramo de pérdidas finaliza se debe al *crossfade*.

Citando a Verma et al., el modelo AR funciona relativamente bien para muestras recientes y es útil en el sentido de sentar las bases para incorporarle añadidos que potencien su predicción [26]. Pese a no ofrecer un resultado preciso, la perspectiva es favorable para los algoritmos que le sucedan en complejidad.

4.2. Análisis del vocoder

El empleo de un vocoder basado en un filtro de predicción lineal supone un salto de calidad a la hora de generar muestras que se acerquen a las de la señal de entrada al sistema.

Destacan dos características cuando se observa la variabilidad del material que genera este modelo. Primeramente, la rapidez con la que es capaz de anteponerse a las fluctuaciones en el tiempo y, en segundo lugar, su insuficiencia en cuanto a permanecer en fase con la señal auténtica, llegando a comportarse ligeramente peor que el predictor lineal en el primer intervalo con pérdidas de la gráfica. Pese a que los esfuerzos en la

implementación de este modelo han ido destinados a imbricar los componentes sonoros o sordos en la fase a partir de la cual existe una pérdida en el paquete, han resultado insuficientes. Para el material sonoro se realizaba una aproximación tomando el periodo de pitch como referencia y sustrayendo manualmente las muestras sobrantes, para insertar la señal generada en el hueco correspondiente. Para el material sordo, simplemente se realizaba una inversión de la excitación al vocoder y se concatenaba con la misma.

A pesar de lo ya expuesto, este escenario consigue adaptarse de forma exitosa en el segundo tramo erróneo. Se puede advertir que en este flujo de paquetes perdidos no existe una tendencia periódica y, aunque no es capaz de estimar el descenso repentino en la señal, llega a solaparse tanto al inicio como al final del intervalo de predicción, a pesar de que no emplea la técnica *crossfade* al comienzo.

4.3. Análisis del modelo sinusoidal

El cambio de paradigma se da en este escenario de la implementación. Las estimaciones comienzan a ser satisfactorias y se está cerca de encontrar el modelo de estimación más adecuado para los objetivos planteados.

El modelo sinusoidal es capaz de anteponerse a la señal y ofrecer una estimación razonable con una inserción en fase sobresaliente de las muestras generadas. No se encuentra solapada con la señal de entrada, pero sí hay un seguimiento fiel de su trayectoria, incluso en algunos transitorios. Al estar basada en el componente sinusoidal de la señal, funciona de forma correcta en fragmentos cuasi-periódicos y no tanto cuando hay cambios inesperados en los transitorios, como ocurre en la marca de tiempo 12.498 s.

El empleo de enventanado trapezoidal (no simétrico) conjuntamente con interpolación de parámetros mejora el procedimiento de análisis-síntesis del modelo previo. La elección de valores para los parámetros da buenos resultados, desde los 48 paquetes anteriores y los 700 tonos considerados para la estimación, hasta la longitud del *crossfade* configurada al tamaño de paquete. Se puede observar el efecto del desvanecimiento cruzado al final del intervalo *lossy*, logrando un solapamiento muy ajustado a la señal original.

4.4. Análisis del modelo sinusoidal con modelo de ruido

Finalmente, la fase de implementación culmina en el escenario que garantiza una reproducción lo más cercana posible a la señal desde la que se parte. Si esta señal se

fragmenta en paquetes o ventanas donde contenga información estacionaria, ofrecerá los mejores resultados.

Si comparamos con el modelo anterior, la primera diferencia es que el desvanecimiento cruzado sí tiene lugar con una ventana trapezoidal simétrica, es decir, se aplica al inicio y al final, superponiéndose, en parte, con la señal real.

La diferencia más notable es que, al tomar en cuenta el residuo del modelo sinusoidal e introducirlo en un vocoder para aprovechar la información que contiene, consigue anteponerse más rápido (característica del vocoder) e incluso mitigar errores en la propia estimación cuando hay variaciones repentinas en la señal de audio. Es decir, la gráfica roja se yuxtapone más a menudo con la original que la gráfica azul y no cuenta con picos muy prominentes cuando no consigue predecir correctamente, cosa que en el modelo anterior sí ocurre al no contar con un vocoder que pueda enmascarar algunos transitorios y modele el ruido existente. Aun así, resulta complicado prever todos los transitorios y los que existen al principio de una señal se ignoran para cualquier modelo predictivo, al no contar con un histórico de muestras pasadas en el que apoyarse.

4.5. Evaluación de experimentos

Antes de proceder con la experimentación y comprobaciones, se plantea la situación: se van a introducir diez pistas de audio al sistema. El parámetro de entrada se puede configurar como dos tasas de error del 1% y el 5%. El sistema de mitigación de errores consiste en cinco escenarios diferentes, que corresponden a los cuatro modelos del software ***Audio_refiller*** y a no realizar acción alguna (modelo canal lossy).

En primer lugar, para caracterizar las cinco opciones de sistema existentes, se ha optado por escoger uno solo de entre todos los *tracks* y verificar que los algoritmos cumplen su función. La pista de audio seleccionada es '*track1.wav*', que ha obtenido la nota media más baja en todos los modelos estudiados para los dos índices de error – valorado con una nota de 67,5 para la tasa 1 y otra de 38,07 para la tasa 2 –. Este proceso se detalla en los siguientes subapartados destinados a cada una de las fases de la prueba.

En segundo lugar, se realiza el test de evaluación perceptual de la calidad de audio (PEAQ) para todas y cada una de las posibles configuraciones que ofrece el escenario escogido. Se van a ofrecer las pistas originales como guía (demarcadas) y se pueden escuchar cuantas veces se desee. Por otra parte, el original se esconderá entre los elementos evaluables

como referencia – idealmente se evaluaría con una puntuación de 100 sobre 100 –. Así, el máximo número de combinaciones para diez pistas con dos índices de error diferentes evaluadas con cinco modelos es de 100 posibilidades. Si además se cuentan las referencias (seis modelos) para las dos tasas, se obtienen 20 posibilidades más, por duplicado. En total suman 120 posibilidades, y 110 sin contar duplicados.

La herramienta de evaluación de calidad auditiva por la que se ha optado es el test WebMUSHRA [58] basado en una API y concebido para la experimentación. MUSHRA es el acrónimo para *"Multiple Stimuli with Hidden Reference and Anchor"*, traducido como "Ensayo multiestímulo con referencia y patrón ocultos" por la ITU-R en la Recomendación BS.1534-3 (10/2015) [59]. En la recomendación se demuestra que la prueba es apta para evaluar la calidad de audio intermedia con resultados «precisos y fiables» tras ensayos satisfactorios.

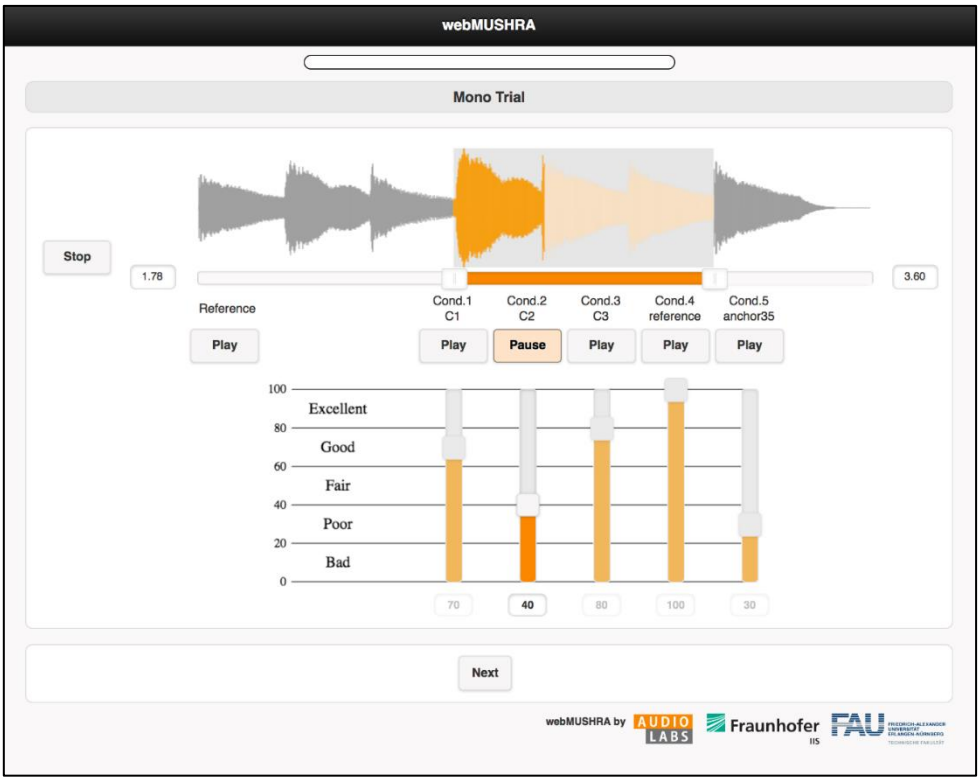

Figura 32 Demo de WebMUSHRA. Fuente: WebMUSHRA

Este test puntúa en un rango de 0 a 100 la calidad de generación de paquetes de audio que se pierden en interpretaciones de música a distancia en la red. El proceso incluye una serie de instrucciones, detallando el tiempo a dedicarle a la escucha y el criterio de valoración. Se emite una puntuación baja si se considera que el modelo se escucha

entrecortado o con cortes audibles molestos en la reproducción, y viceversa si no se han percibido estas interrupciones.

El test ha sido realizado por un total de siete participantes: tres mujeres y cuatro hombres. Sus edades están en el rango de 24 a 39 años, siendo dos personas de 24 años, tres de 25, una de 31 y otra de 39. Solo tres de los sujetos tenían experiencia en procesado de audio y/o estudios de música, contando con un profesional del primer campo entre ellos. Los otros cuatro participantes no tenían ningún trasfondo o estudios en este aspecto.

Contextualizando, si un paquete de información llega erróneo, va a ser descartado por el sistema receptor y eso dará lugar a un hueco en la señal, un silencio en la reproducción de música que se percibirá como un chasquido en paquetes de pocos milisegundos,

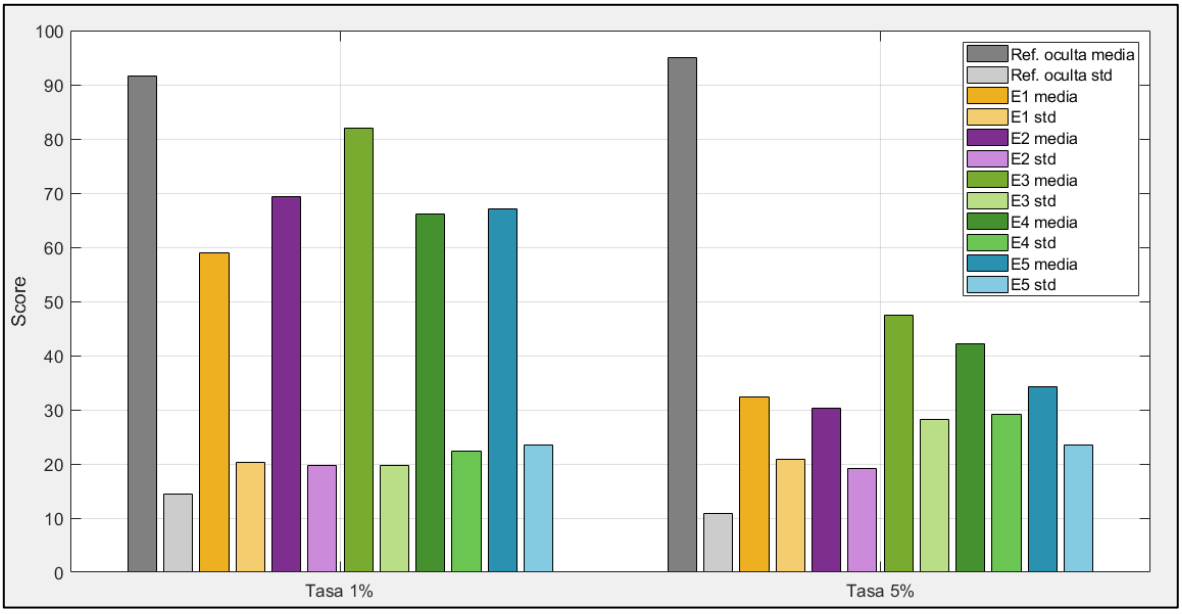

Figura 33 Medias y desviaciones típicas de los escenarios según su tasa de error. Fuente: Elaboración propia

encontrándose su rango de actuación típico en paquetes de duraciones de entre 4 y 16 ms [3]. En el caso de análisis, se han tomado paquetes de duración de 2 ms para una frecuencia de muestreo de 44100 Hz y una longitud máxima de ráfaga de 4 paquetes, con lo cual, el máximo de duración de ráfaga son 8 ms. Sin embargo, para ratios de error de más del 1%, puede ocasionar que más de cuatro paquetes perdidos consecutivos se fusionen, superando el máximo de ráfaga configurado y dando lugar a “clics” que comienzan a entorpecer la escucha y la interpretación sincronizada, en suma.

Escenarios	Tasas de error	Medias	Desviaciones típicas

Referencia oculta	1%	91,6429	14,3951
	5%	94,9857	10,9366
Escenario 1 (E1): Canal *lossy*	1%	58,9	20,3214
	5%	32,3	20,7904
Escenario 2 (E2): Predictor lineal	1%	69,3286	19,6452
	5%	30,2429	19,1201
Escenario 3 (E3): Vocoder	1%	81,9286	19,7237
	5%	47,5429	28,1804
Escenario 4 (E4): Modelo sinusoidal	1%	66,1571	22,3228
	5%	42,1	29,209
Escenario 5 (E5): Modelo sin. + modelo ruido	1%	67,1143	23,5382
	5%	34,3286	23,4334

Tabla 5. Datos estadísticos de cada uno de los escenarios.

En relación a la figura y tabla superiores, como era de esperar, la referencia ha obtenido la nota más alta de todos los escenarios, por encima de un 90% de media en ambas pruebas con tasas de error diferentes. Curiosamente, a pesar de que la señal de referencia es igual en ambos escenarios, si se comparan, en el índice de error más alto (5%) no solo hay mayor diferencia de calidad de escucha en el *track* original con respecto al resto de estímulos, sino que además este ha obtenido 3.35 puntos más de media que el mismo *track* de referencia en un planteamiento con una tasa de error más baja. Sin embargo, los resultados de los distintos escenarios no se corresponden con lo esperado, a excepción del escenario en el que no se hacen mejoras respecto a los errores, en el que es de sobra conocido que no va a mitigar adecuadamente las pérdidas para los índices de error que se manejan.

Cabe pensar que, para un índice del 5% de pérdida, la diferencia entre las señales que contienen errores y la señal correcta resulta mucho más notable. Tomando la referencia y el estímulo con la nota más deficiente, que es el modelo de predicción lineal, estudiándolos para un 5% de fallos, hay un salto de 64.75 puntos, que demuestra una calidad de escucha pésima. En contraposición, con un 1% de tolerancia a fallos ese salto puede ser más abarcable, puesto que la diferencia es de 32.74 en nota equiparando la referencia y el simulador de canal con errores, que tiene la puntuación más baja en este caso.

Si se toma el resultado perceptual más favorable de entre todos los estímulos, se tiene un hecho sorprendente, y es que el modelo de vocoder ha sido elegido por los participantes en el test MUSHRA como el modelo que mejor calidad auditiva media consigue en comparación a los otros cuatro escenarios. Aunque no debería serlo, ya que, objetivamente,

los modelos sinusoidal y sinusoidal con vocoder añadido deberían funcionar mejor. Se ha de tener en cuenta la experiencia en audio de los participantes y también el tamaño reducido de la muestra estadística (7 individuos), pero, para comprender por qué ocurre este fenómeno, la importancia reside en la propia naturaleza de las pistas de audio escogidas.

En este caso, las señales tienen el componente tonal mezclado con ruido, ya sea de percusión o debido a otras fuentes. Para este tipo de señales, el vocoder realmente destaca por su eficacia y, aunque su gráfica pueda no parecerse a la original, se trata de conseguir replicar la sonoridad mediante el modelado, no el mismo material. En esta línea, el modelo sinusoidal no funciona tan bien como cabría esperar, ya que depende mucho del componente tonal (por ejemplo, señales de audio instrumental sin percusión en las que la fuente sea un piano, o instrumentos similarmente tonales) y de que este se distinga claramente en los fragmentos con información estacionaria. Si se presta más atención, el modelo sinusoidal con modelo de ruido sintético funciona un poco mejor que el modelo sinusoidal básico, y esto es gracias al vocoder que incorpora.

Concluyendo, el modelo de vocoder es el que mejor han valorado los sujetos del test y podría funcionar incluso mejor de haber hecho más hincapié en la fase. Aunque lo cierto es que para el tipo de señales consideradas en el estudio, la información de fase puede resultar menos relevante que si fueran exclusivamente tonales. No obstante, no hay que despreciar que en los dos últimos escenarios, la desviación típica se mantiene para ambas tasas de error, lo cual es muy positivo porque demuestra la robustez del modelo sinusoidal frente a los errores.

5. CONCLUSIONES Y LÍNEAS FUTURAS

Examinando el nivel de consecución de los objetivos propuestos, el desarrollo de la herramienta software para este proyecto y los principales puntos expuestos en él, es posible extraer algunas conclusiones y plantear una serie de posibles continuaciones y mejoras al trabajo efectuado.

Este Trabajo Fin de Grado se basó en implementar un sistema capaz de mitigar errores en la señal de audio generando nuevos fragmentos de la misma mediante técnicas de predicción en *jamming sessions* hospedadas en aplicaciones colaborativas de música en tiempo real. En esta línea, se evalúan los objetivos planteados:

- Se ha creado un sistema capaz de generar audio en los paquetes que se pierden en interpretaciones de música a distancia mediante algoritmos predictivos basados en la predicción lineal, la codificación de voz y el modelado sinusoidal.
- Se ha estudiado el funcionamiento de las tecnologías NMP de interpretación musical sincronizada en la red y se han comprendido su importancia, sus antecedentes y limitaciones. Entre ellas, se encuentran Jacktrip, Jamulus, JamKazam, Soundjack, etc.
- Se ha simulado un modelo de canal con pérdidas a partir de la probabilidad de error de paquete como escenario base en el que no se realiza nada para paliar esas pérdidas.
- Se han implementado métodos de estimación de paquetes perdidos fundamentados en los algoritmos estudiados para tal fin y se ha configurado una herramienta software preexistente que evalúa dichos métodos mediante un test experimental, siendo esta la aplicación WebMUSHRA, la cual funciona a través de una API del instituto Fraunhofer.
- Se ha seleccionado la propuesta de predicción más apta en base a la calidad auditiva lograda, que es el modelo de vocoder.
- Se ha generado una señal sintética que mejore la calidad perceptual en aplicaciones de interpretación musical en la red cuando la pérdida de paquetes sea significativa mediante el modelo sinusoidal con modelo de ruido.

Tras haber dimensionado el conjunto de este TFG, se dejan planteadas algunas posibles vías de estudio en próximos trabajos de investigación sobre este tema:

- Las pistas empleadas no se corresponden con un flujo continuo de paquetes, por lo tanto, su aplicabilidad es limitada. ¿Qué ocurre cuando la señal se prolonga en el tiempo en una sesión de *jamming*? ¿Se establecerían límites de errores permitidos en los paquetes contenidos en el transcurso de un segundo o de cualquier otra unidad de tiempo?
- Efectuar un estudio de la latencia aportada en todos los procesos de la red, incluyendo la ejecución del software y tomando en cuenta, por ejemplo, cuánto tardan en cargar las funciones para la reconstrucción predictiva y cómo optimizarlas.
- Desarrollar una solución centrada exclusivamente en el método predictivo de Machine Learning y redes neuronales para generar paquetes perdidos de señal musical.

6. BIBLIOGRAFÍA

[1] «The League of Automatic Music Composers - Free Music Archive». Accedido: 3 de septiembre de 2023. [En línea]. Disponible en: https://freemusicarchive.org/music/The_League_of_Automatic_Music_Composers/bio

[2] Á. Barbosa, «Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation», *Leonardo Music Journal*, vol. 13, pp. 53-59, dic. 2003, doi: 10.1162/096112104322750791.

[3] C. Rottondi, C. Chafe, C. Allocchio, y A. Sarti, «An overview on networked music performance technologies», *IEEE Access*, vol. 4, pp. 8823-8843, 2016, doi: 10.1109/ACCESS.2016.2628440.

[4] V. Fischer, «Case Study: Performing Band Rehearsals on the Internet With Jamulus».

[5] «WebRTC». Accedido: 4 de septiembre de 2023. [En línea]. Disponible en: https://webrtc.org/?hl=es-419

[6] M. Sacchetto, A. Servetti, y C. Chafe, «JackTrip-WebRTC: Networked music experiments with PCM stereo audio in a Web browser», 2021. [En línea]. Disponible en: https://github.com/JackTrip-webrtc/JackTrip-webrtc.

[7] «How to Make Music Online with Jacktrip Virtual Studio - YouTube». Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://www.youtube.com/watch?v=aGJOtKhc9gU

[8] «JamKazam | Live, In-Sync Music Jamming Over the Internet». Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://jamkazam.com/

[9] «RFC 9293: Transmission Control Protocol (TCP)». Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://www.ietf.org/rfc/rfc9293.html

[10] «Introducing JackTrip 2.0 - Product Updates - JackTrip Community Forums». Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://community.jacktrip.org/t/introducing-jacktrip-2-0/652

[11] «Fehlerseite 404 - Fraunhofer IDMT». Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://www.idmt.fraunhofer.de/de/projects/expired_publicly_financed_research_projects/ultra_low_delay.html

[12] A. Carôt, C. Hoene, H. Busse, y C. Kuhr, «Results of the Fast-Music Project-Five Contributions to the Domain of Distributed Music», doi: 10.1109/ACCESS.2020.2979362.

[13] «JackTrip». Accedido: 10 de junio de 2023. [En línea]. Disponible en: https://jacktrip.github.io/jacktrip/

[14] «Eric-Whitacre». Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://www.jacktrip.com/eric-whitacre

[15] «Remote Music Rehearsals | JamKazam». Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://jamkazam.com/remote-music-rehearsals/

[16] «JamKazam Forums». Accedido: 3 de septiembre de 2023. [En línea]. Disponible en: https://forum.jamkazam.com/

[17] «Jamulus - Internet Jam Session Software download | SourceForge.net». Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://web.archive.org/web/20200426235011/https://sourceforge.net/projects/llcon/

[18] «Jamulus – Toca música online.» Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://jamulus.io/es/

[19] «Roger B. Dannenberg | Audio Latency». Accedido: 4 de noviembre de 2023. [En línea]. Disponible en: https://www.cs.cmu.edu/~rbd/blog/latency-blog29jul2020.html

[20] F. J. Cañadas, «Temario Técnicas de Codificación y Transmisión. Tema 1: Técnicas de transformación.»

[21] «JackTrip Labs | Technology». Accedido: 4 de septiembre de 2023. [En línea]. Disponible en: https://www.jacktrip.com/technology

[22] «Las líneas de fibra óptica hasta el hogar alcanzaron los 14,3 millones en junio | CNMC». Accedido: 4 de septiembre de 2023. [En línea]. Disponible en: https://www.cnmc.es/prensa/mensual-telecos-junio-20230811

[23] «Virtual Studio: Frequently Asked Questions (FAQs) – JackTrip». Accedido: 6 de noviembre de 2023. [En línea]. Disponible en: https://help.jacktrip.org/hc/en-us/articles/360055332753-Virtual-Studio-Frequently-Asked-Questions

[24] «How to Join with a JackTrip bridge – JackTrip». Accedido: 6 de noviembre de 2023. [En línea]. Disponible en: https://help.jacktrip.org/hc/en-us/articles/6356359766291

[25] «Jitter explained - Part 1.3 [English]». Accedido: 4 de septiembre de 2023. [En línea]. Disponible en: https://www.tnt-audio.com/clinica/jitter1_e.html

[26] P. Verma, A. I. Mezza, C. Chafe, y C. Rottondi, «A Deep Learning Approach for Low-Latency Packet Loss Concealment of Audio Signals in Networked Music Performance Applications», jul. 2020, [En línea]. Disponible en: http://arxiv.org/abs/2007.07132

[27] R. Braden, «Requirements for Internet Hosts - Communication Layers», oct. 1989, doi: 10.17487/RFC1122.

[28] «RFC 768: User Datagram Protocol». Accedido: 4 de septiembre de 2023. [En línea]. Disponible en: https://www.rfc-editor.org/rfc/rfc768

[29] Austen Lenihan, «TCP vs UDP´-Which is Better for Streaming», Dacast. Accedido: 2 de junio de 2023. [En línea]. Disponible en: https://www.dacast.com/blog/tcp-vs-udp/#:~:text=User Datagram Protocol (UDP) is,because it's very latency efficient

[30] Team Pico, «UDP vs TCP», PICO Knowledgebase.

[31] H. Schulzrinne, S. Casner, R. Frederick, y V. Jacobson, «RTP: A Transport Protocol for Real-Time Applications», jul. 2003, doi: 10.17487/RFC3550.

[32] J. Rosenberg *et al.*, «SIP: Session Initiation Protocol», jun. 2002, doi: 10.17487/RFC3261.

[33] R. Fielding *et al.*, «Hypertext Transfer Protocol -- HTTP/1.1», jun. 1999, doi: 10.17487/RFC2616.

[34] FMTZ team, «RAW AUDIO FILE FORMATS INFORMATION», FMTZ FILE FORMATS INFORMATION AND SPECIFICATIONS. Accedido: 16 de octubre de 2023. [En línea]. Disponible en: https://web.archive.org/web/20160525083851/http:/www.fmtz.com/misc/raw-audio-file-formats

[35] B. M. Uranga y A. Lamacraft, «Schr\"odingeRNN: Generative Modeling of Raw Audio as a Continuously Observed Quantum State», vol. 107, n.º 2018, pp. 74-106, 2019, [En línea]. Disponible en: http://arxiv.org/abs/1911.11879

[36] «Comparison of WAV, FLAC and OGG audio formats: size and latency - Igor S. Ramos». Accedido: 6 de noviembre de 2023. [En línea]. Disponible en: https://web.archive.org/web/20230803225529/https://igorsramos.com/wav-flac-ogg-audio-formats/

[37] «Opus Codec». Accedido: 6 de noviembre de 2023. [En línea]. Disponible en: https://opus-codec.org/

[38] «Multimedia Programming Interface and Data Specifications 1.0». Accedido: 6 de noviembre de 2023. [En línea]. Disponible en: http://www.tactilemedia.com/info/MCI_Control_Info.html

[39] «Ring This... Wave File Format». Accedido: 6 de noviembre de 2023. [En línea]. Disponible en: https://www.ringthis.com/dev/wave_format.htm

[40] «Lossless comparison - Hydrogenaudio Knowledgebase». Accedido: 6 de noviembre de 2023. [En línea]. Disponible en: https://wiki.hydrogenaud.io/index.php?title=Lossless_comparison#Comparison_Table

[41] «FLAC - Format overview». Accedido: 6 de noviembre de 2023. [En línea]. Disponible en: https://www.xiph.org/flac/documentation_format_overview.html

[42] «Digital Signal Processing (4th Edition) by John G. Proakis, Dimitris K Manolakis (z-lib.org)».

[43] «Teoría de la Comunicación (2º grado)», 2019.

[44] A. Lerch, «An Introduction to Audio Content Analysis».

[45] Python Software Fundation, «The Python Language Reference», Python Documentation. Accedido: 12 de febrero de 2023. [En línea]. Disponible en: https://docs.python.org/3/reference/index.html

[46] «Tags - Stack Overflow». Accedido: 5 de septiembre de 2023. [En línea]. Disponible en: https://stackoverflow.com/tags

[47] Mathworks, «Matlab. Documentación», *Mathworks*, 2023, [En línea]. Disponible en: https://es.mathworks.com/help/pdf_doc/matlab/index.html

[48] «The Python Tutorial — Python 3.12.0 documentation». Accedido: 6 de noviembre de 2023. [En línea]. Disponible en: https://docs.python.org/3/tutorial/index.html

[49] P. Vera-Candeas, «Predicción lineal», 2020.

[50] A. V. Oppenheim, R. W. Schafer, y J. Portillo, *Tratamiento de señales en tiempo discreto*. Prentice-Hall, 2011.

[51] P. Vera-Candeas, «Estimación espectral paramétrica», 2018.

[52] D. Martínez Muñoz, *Transmisión digital* . en Techné. Jaén: Universidad de Jaén, Servicio de Publicaciones, 2009.

[53] «5.8. Vocoder — Introduction to Speech Processing». Accedido: 5 de septiembre de 2023. [En línea]. Disponible en: https://speechprocessingbook.aalto.fi/Modelling/Vocoder.html

[54] A. Meynard y B. Torrésani, «Spectral analysis for nonstationary audio», 2018.

[55] M. Berg, M. Fuchs, K. Wirkner, M. Loeffler, C. Engel, y T. Berger, «The Speaking Voice in the General Population: Normative Data and Associations to Sociodemographic and Lifestyle Factors», *Journal of Voice*, vol. 31, n.º 2, pp. 257.e13-257.e24, mar. 2017, doi: 10.1016/J.JVOICE.2016.06.001.

[56] P. Vera-Candeas, «Desarrollo de técnicas de codificación de audio basadas en modelos de señal paramétricos».

[57] Z. and Rafii, A. and Liutkus, F.-R. and Stöter, S. I. and Mimilakis, y R. Bittner, «MUSDB18-HQ - an uncompressed version of MUSDB18», Zenodo. Accedido: 30 de agosto de 2023. [En línea]. Disponible en: https://doi.org/10.5281/zenodo.3338373

[58] M. Schoeffler *et al.*, «webMUSHRA - A comprehensive framework for web-based listening tests», *J Open Res Softw*, vol. 6, n.º 1, 2018, doi: 10.5334/jors.187.

[59] Uit-r, «Método para la evaluación subjetiva del nivel de calidad intermedia de los sistemas de audio Serie BS Servicio de radiodifusión (sonora)», 2023, Accedido: 5 de septiembre de 2023. [En línea]. Disponible en: http://www.itu.int/publ/R-REC/es

ANEXOS

Para lograr una comprensión plena del documento, se ha compilado contenido adicional para que el lector que lo desee pueda compilar el programa, así como profundizar en la lógica interna de sus módulos.

1. Manual de usuario de “Audio_refiller”

Aquí se detallan los pasos previos a seguir para el funcionamiento del software desarrollado como objetivo principal de este TFG.

Escenario 1

Esta fase del programa consiste en una simulación de un canal *lossy* en el que se perderán paquetes, los cuales llegarán erróneos y serán descartados por el programa, lo cual dará lugar a huecos en el *stream* de datos. Primeramente, se estudiará la ejecución del *script* principal a vista de pájaro en **main.m** y después se examinará la función **canal.m** internamente.

Ejecución del archivo main

El programa lee un fichero de audio descomprimido en formato WAV. Debido a que es sonido en estéreo, uno de los dos canales se eliminará y se tendrá una señal *mono* que se modificará ligeramente para poder convertirse en la entrada al sistema. Previo a operar, se hacen necesarios unos parámetros iniciales:

- los **segundos de audio que se desean transmitir (tx)** en un paquete,
- la **tasa de error máxima (P)** que se permitirá en el canal,
- la **máxima longitud de ráfaga de error (M)** en bits.

Se ha de pensar en la entrada como un flujo continuo de paquetes de audio. Con esto en mente, se calculan las **muestras por paquete (S)** multiplicando la frecuencia de muestreo por los segundos de transmisión. Una vez se tienen las muestras, se puede obtener el **número de paquetes (num)** de información en que se dividirá la entrada (muestras totales/muestras de paquete), el cual es, obviamente, un número entero. Se procede a concatenar la entrada con muestras nulas de cara a realizar una división exacta en **trozos** iguales. Así, se remodela la señal de entrada en una variable **trozos1** con **S** filas y **num** columnas.

A continuación, los paquetes de señal pueden ser propagados por el canal, que tendrá unos parámetros **P** y **M** concretos. Del canal con pérdidas a la salida se computará el vector de error (**Err**) en forma de array booleano.

Después de llamar a la función "canal" de **canal.m**, se inicializa un vector de salida **y** igual a **trozos1**. Sabiendo que el vector de error está formado por unos y ceros, se recorre elemento a elemento y desembocará en dos vías:

- Si el elemento **Err** vale '1', el paquete actual se ha perdido y se anula la muestra de llegada **y** en el receptor.
- Si el elemento **Err** vale '0', se considera que el paquete actual se ha recuperado con éxito y la ejecución continúa.

Si se computa la media de aciertos del vector de error, se podrá conocer la tasa de error final existente en el canal. También se puede realizar un test de calidad de audio midiendo la relación señal a ruido (**SNR**) existente entre la salida y el error en el medio de transmisión, o sea, en la fibra óptica a través de la red.

Tras simular un canal ruidoso, se escuchan las diferencias entre la señal original y la señal con pérdidas que llega al receptor.

Función simulador de canal

canal.m tiene la sintaxis siguiente:

Err = canal(num, P, M)

Tiene como argumentos de entrada el número entero de paquetes correspondiente a la señal que se quiere propagar, el umbral máximo permitido de tasa de error en el canal y la longitud de ráfaga de error del canal. Expulsa un vector con los bits a '1' o '0' según correspondan a paquetes erróneos o correctos.

La función canal solo genera el escenario del mismo, como tal no se encarga de descartar o aprovechar paquetes, ya que esa es una característica del receptor reservada a la instancia principal del escenario 1 de ***audio_refill***.

En resumidas cuentas, se inicializa el vector de error del canal que tendrá el tamaño igual a **num**. Seguidamente, se genera un array de números aleatorios del mismo tamaño y se crea una variable para la probabilidad de ráfaga, **p_rafaga**, la cual acotará los índices donde se producirá una ráfaga con una probabilidad por debajo del límite máximo, **M**. A partir de entonces, un bucle generará cada vez una longitud de ráfaga aleatoria por debajo de **M** y recorrerá el vector **Err** en base a los índices guardados en **p_rafaga**, con ráfagas de

longitud **l_rafaga**. En los índices que corresponda, el vector de error se configurará como un ‘1’ lógico.

En los sucesivos modelos, se mencionarán solo las variables de entrada relevantes, pues el resto se reutilizarán, así como se omitirá lo relativo al canal de datos. Cada apartado se ceñirá al material nuevo que se haya añadido en cada caso.

Escenario 2

El algoritmo del predictor lineal en este escenario se ha desplegado conjuntamente dentro de la función armod.m y dentro del archivo principal, para después integrar los paquetes a la salida del sistema en este último.

Ejecución del archivo main

Los argumentos de entrada acumulados en esta etapa son:

- El orden del filtro de predicción lineal (O).
- Umbral de predicción para el algoritmo de Levinson-Durbin.
- Muestras de promediado para realizar técnicas de fading al inicio y final de paquete a la hora de superponerlos con huecos.
- Número mínimo de trozos o paquetes a partir de los cuales realizar la estimación.

Sustentándose en lo construido en el simulador de canal de transmisión con errores, esta vez se generan dos vectores de paquetes basados en la señal original dividida: trozos1 y trozos2. trozos1 equivaldría a la entrada sin modificaciones y trozos2 computaría el efecto del canal ruidoso en la señal, originando huecos o tramos de material descartado.

A partir de trozos2 se construye matricialmente la salida, y una vez superado el límite superior para la predicción, n, se toman los n últimos trozos previos al erróneo, almacenados en y_l, la entrada al modelo paramétrico.

Tras ejecutar armod.m, se inicializa un vector de predicción de tamaño el doble de muestras del paquete actual (con las muestras de promediado) y se va rellenando con las ecuaciones de predicción lineal teniendo en cuenta el orden óptimo. Una vez construido el vector y_pred, se realizan un fade in y un fade out al principio y al final del paquete predicho para integrarlo con los paquetes pasados y futuros de la señal de salida.

Función armod

Sintaxis:

[Kp,ar]=armod(x0,pmax,umbral)

Dentro de la función se ejecuta el modelo autorregresivo de predicción a partir de la autocorrelación. A partir de los coeficientes del modelo AR, se calculan los coeficientes k del modelo Levinson-Durbin que permiten evaluar en qué orden el filtro de predicción será inestable y no realizará buenas estimaciones. Es decir, a partir de un valor de k mayor o igual que el umbral, se rechaza el orden actual del filtro y se toma el orden inmediatamente anterior, que sería el óptimo para estimar los coeficientes que se emplearán en el filtro de predicción lineal.

La función expulsa, pues, el valor de k del orden óptimo y los coeficientes del filtro.

Escenario 3

La tercera etapa corresponde con el desarrollo de un sistema de codificación de voz, un vocoder modificado para señales instrumentales que pueden incluir o no fuente vocal.

Ejecución del archivo main

Si el penúltimo trozo al actual es erróneo, se convierte la longitud de ráfaga en muestras (long). Si el número de bits de la ráfaga actual es menor que el total de trozos anteriores empleado para predecir (n), quiere decir que no se dispone de material de predicción, con lo cual los paquetes afectados por dicha ráfaga se van a perder (se establecen en cero). En cambio, si los bits de ráfaga actuales se encuentran por encima del umbral de predicción (n), se ejecuta el vocoder y se obtiene el paquete predicho, al cual se le realizará el promediado de muestras correspondiente para integrar un fade in al inicio y un fade out al final.

Función vocoder

Sintaxis:

[y_pred]=vocoder2(y_l,fs,polos,S,n)

Se ejecuta la función de detección de pitch y, se realiza un despliegue en base a que el fonema sea sonoro o sordo. Si el pitch devuelto por pitchdetec es distinto de 0, se considera que el fonema es sonoro y se realiza una detección de fase a partir del resto entre la muestra de la excitación (la entrada al vocoder) a partir de la cual se ha perdido la información restándole los 12 polos iniciales, y el valor del pitch.

Se hace una concatenación de la excitación con la excitación a partir de la muestra en fase calculada y se transfiere por el filtro de predicción para obtener la salida del vocoder.

Función pitchdetec

Sintaxis:

[pitch,rs]=pitchdetec(x,Fs)

Basándose en el rango de frecuencias de 40 a 4000 Hz, que es ligeramente más dilatado que el correspondiente a la voz humana, se obtiene el periodo de pitch a partir de los máximos locales. Con este parámetro se puede comprobar si el fonema o material es de tipo sonoro o sordo y cómo se puede modelar.

Escenario 4

En el cuarto escenario, se despliega el modelo sinusoidal basado en la transformada de Fourier. Su ejecución se halla dividida en dos bloques: uno de análisis y otro de síntesis. Cada bloque esta representado por una función, sin_analisis_atom2 y sin_sintesis_ext_rap, respectivamente. Dentro del main, se inicializa primero la función de análisis y después la de síntesis.

Ejecución del archivo main

- Condición 1 – Si el trozo actual es erróneo y la iteración del bucle es menor que el número de paquetes (Err==1 y j<num), aumenta el tamaño de la ráfaga en la variable contador.
- Condición 2 – Si no hay bit de error, la iteración no es igual al número de paquetes u no hay bit de error en la iteración anterior ((Err(j)==0 || Err(num)==1) && Err(j-1)==0) termina la ráfaga reseteando el contador.
- Condición restante – el penúltimo trozo al actual es erróneo:
  Inicialmente, se convierte la longitud de ráfaga en muestras (long). Si el número de bits de la ráfaga actual es menor que el total de trozos anteriores empleado para predecir (n), quiere decir que no se dispone de material de predicción, con lo cual los paquetes afectados por dicha ráfaga se van a perder (se establecen en cero). En cambio, si los bits de ráfaga actuales se encuentran por encima del umbral de predicción (n), se ejecuta el modelo sinusoidal.

Función de análisis sinusoidal

Sintaxis de la función:

[frec, amp, fases, fftresiduo] = sin_analisis_atom2(senal_ven, TF_uno, longitud_frec, ventana, valorfin, criterio)

Como es habitual, antes de comenzar se señalan las variables de entrada. Además de y_l, el vector formado por n paquetes anteriores, se introducen en la función sin_analisis_atom2 los cinco siguientes parámetros:

- La longitud en frecuencia (longitud_frec), o sea el número de puntos que tendrá la transformada de Fourier.
- El número máximo de tonos (deltas de Dirac) en frecuencia que se emplearán en la estimación (maxtonos).
- La ventana rectangular que se aplicará a y_l a la entrada del sistema
- La transformada de Fourier rápida (FFT) de la ventana TF_uno con el doble de puntos que la longitud en frecuencia.
- El criterio para localizar los picos está establecido como el número máximo (‘numeromax’) comparando con maxtonos. Cuando el contador interno de frecuencias (contfrec) alcance el máximo de tonos que se quieren calcular en este modelo, sesale de la ejecución.

A la salida, una vez completado el análisis, se extraen los parámetros a continuación:

- Vector con las frecuencias a las que se encuentran los máximos locales
- Vector de amplitudes de los máximos.
- Vector de fases de los máximos encontrados.
- Residuo de la FFT sobre el cual se predecirá el paquete perdido.

sin_analisis_atom2 hace la TF de la señal de entrada para trabajar en el dominio de la frecuencia y así poder buscar los picos o tonos donde se localice la información a extraer. Del máximo relativo con el cual opere en cada momento registra sus parámetros principales: su amplitud, la fase en la que se encuentra y la frecuencia a la que está el pico.

Tras haber realizado la transformada FFT de la ventana rectangular, se la resta al pico para obtener el residuo de la transformada. Esta operación se repite hasta que haya terminado con todos los tonos especificados en maxtonos.

Función de síntesis sinusoidal

La sintaxis de la función sin_sintesis_ext_rap se expresa de la forma:

[senal_rec] = sin_sintesis_ext_rap(frec, amp, fases, longitud_frec, N, ext, T)

De los parámetros que se obtuvieron en la fase de análisis, ahora se integran en el último bloque de este algoritmo. La división del algoritmo en dos etapas permite atacar el problema de la predicción de manera eficaz aislando dos procesos distintos.

A partir del cómputo de tonos (sinusoides en el dominio espectral), recorre las frecuencias y genera un tono con la información de entrada relativa a la amplitud, la fase y frecuencia, además de convertirlo al dominio temporal.

Los parámetros de entrada a sin_sintesis_ext_rap, sin incluir longitud_frec, son:

- Las salidas del análisis conforman la entrada al proceso de síntesis: la frecuencia, la amplitud y las fases.
- El número de muestras de las que se parten para emplear en la predicción (N)
- El número de muestras predichas que el modelo generará (ext).
- El término de normalización de la serie de Fourier, que se suele expresar como 1/T en la ecuación inversa para el dominio del tiempo (IFFT). Por facilidad de nomenclatura de variables, se calcula como T.

A la salida el modelo expulsa la señal recibida estimada.

Escenario 5

Esta etapa es una continuación de las dos etapas anteriores, integrando el algoritmo de vocoder y el algoritmo de la interpolación por transformada rápida de Fourier.

Ejecución del archivo main

La novedad en este algoritmo reside en su estructura, ya que aprovecha el residuo resultante de la diferencia entre el conjunto de paquetes anteriores empleados en la predicción (y_l) y la salida predicha (y_pred), del mismo tamaño. Este residuo es de tipo ruidoso y es la entrada al vocoder, que obtiene una salida de predicción mucho más afinada, a pesar de no contar con adaptabilidad a los transitorios de la señal. Al remontarse al funcionamiento de un vocoder, se comprueba que realmente destaca en modelar paramétricamente señales de entrada ruidosas, que encuentran su equivalente en fonemas sordos y se modelan como ruido blanco aditivo y gaussiano.

[insertar diagrama]

Se integran en este programa tanto las funciones de vocoding como las de análisis síntesis de Fourier.