UNIVERSIDAD SANTO TOMÁS

IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO DE COMANDOS DE VOZ EN SISTEMAS EMBEBIDOS ORIENTADO A ENTORNOS DOMÓTICOS Y AL DESARROLLO DE UNA GUÍA DE LABORATORIO

Realizado por

David Ricardo Melo Suarez

Sergio Emiro Vargas Cruz

Proyecto presentado en cumplimiento del requisito
para optar por el grado de Ingeniería Electrónica

Grupo de Investigación MEM (Modelado-Electrónica-Monitoreo)
Facultad de Ingeniería Electrónica
División de Ingenierías

Mayo de 2025

IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO DE COMANDOS DE VOZ EN SISTEMAS EMBEBIDOS ORIENTADO A ENTORNOS DOMÓTICOS Y AL DESARROLLO DE UNA GUÍA DE LABORATORIO

Realizado por

David Ricardo Melo Suarez

Sergio Emiro Vargas Cruz

Proyecto enviado en cumplimiento del requisito
para optar por el grado de Ingeniería Electrónica

Dirigido por

Ing. Carlos Javier Mojica Casallas, M.Sc

Grupo de Investigación MEM (Modelado-Electrónica-Monitoreo)
Sistemas de Energía y Automatización
Facultad de Ingeniería Electrónica
División de Ingenierías

Mayo de 2025

Autoridades de la Universidad

RECTOR GENERAL
R.P. FRAY ÁLVARO JOSÉ ARANGO RESTREPO, O.P

VICERRECTOR ADMINISTRATIVO Y FINANCIERO GENERAL
R.P. FRAY HERNÁN YESID RIVERA ROBERTO, O.P

VICERRECTOR ACADÉMICO GENERAL
R.P. FRAY MAURICIO ANTONIO CORTÉS GALLEGO, O.P

SECRETARIO GENERAL
Dra. LUCERO GALVIS CANO

DECANO DIVISIÓN DE INGENIERÍAS
R.P FRAY JAVIER ANTONIO HINCAPIÉ ARDILA, O.P

SECRETARIA DE DIVISIÓN
E.C. LUZ PATRICIA ROCHA CAICEDO

DECANO FACULTAD DE INGENIERÍA ELECTRÓNICA
ING. LEONARDO FABIO YEPEZ ARBELAÉZ.

Nota de aceptación

_____
_____
_____
_____
_____
_____

_____
Firma del tutor

_____
Firma del jurado

_____
Firma del jurado

BOGOTÁ D.C. _____ DE 2025

Advertencia

La Universidad Santo Tomás no se hace responsable de las opiniones y conceptos expresados en el trabajo de grado, solo velará por qué no se publique nada contrario al dogma ni a la moral católica y porque el trabajo no tenga ataques personales y únicamente se vea el anhelo de buscar la verdad científica.

Capítulo III –Art. 46 del Reglamento de la Universidad Santo Tomás.

Dedicatoria

Me gustaría, sobre todo, dedicar esta parte a mis padres, aquellos que hicieron posible, con su esfuerzo y acompañamiento, que yo esté aquí. No solo por brindarme la oportunidad de estudiar, sino también por los valores y principios que me infundieron, los cuales me han permitido llegar hasta este momento. La construcción de este proyecto es, en gran parte, resultado de todo el proceso que hemos recorrido juntos.

David Ricardo Melo Suarez.

Quiero dedicar este proyecto a mis padres, por su apoyo inquebrantable a lo largo de toda la carrera. Tanto este trabajo como mi formación personal son un reflejo directo de los valores que me inculcaron desde pequeño, en especial la disciplina y la perseverancia.

Se los dedico por dar tanto sin buscar nada a cambio y por acompañarme en silencio en tantos días de esfuerzo. Esta meta también es de ustedes.

Sergio Emiro Vargas Cruz

Agradecimientos

Queremos expresar nuestro agradecimiento a todos los profesores de la carrera, quienes no solo compartieron su conocimiento, sino que también nos ayudaron a desarrollar el pensamiento crítico y la capacidad para enfrentar con criterio los desafíos de la ingeniería. Gracias a su enseñanza, hoy contamos con las bases necesarias para abordar con seguridad los problemas que se nos presentan.

De manera especial, agradecemos al director de este proyecto de grado, Carlos Javier Mojica, por su guía, dedicación y valioso apoyo tanto en nuestra formación en procesamiento digital de señales como en el desarrollo mismo del presente trabajo. Su compromiso, claridad y pasión por la materia fueron una fuente de inspiración que nos motivó a profundizar en esta interesante área del conocimiento.

Índice General

Resumen

Este proyecto presenta el diseño e implementación de un sistema de reconocimiento de comandos de voz en un entorno embebido, orientado al control de dispositivos domóticos. Se emplea el procesador digital de señales TMS320C5535, integrando el codec TLV320AIC3204 y una interfaz UART con Arduino para la activación de actuadores mediante comandos de voz. El sistema incluye etapas completas de adquisición de audio, preprocesamiento (eliminación de silencios, normalización, preénfasis y ventaneo), extracción de características mediante Mel-frequency cepstral coeficients (MFCC) y clasificación con tres modelos: Dynamic Time Warping (DTW), Gaussian Mixture Models (GMM) y distancia Euclidiana con normalización CMVN. Los resultados muestran que el modelo GMM ofrece el mejor balance entre precisión (86 % en ambientes ruidosos) y eficiencia computacional. Además, se desarrolló una guía de laboratorio que refuerza el aprendizaje del procesamiento digital de señales, fomentando el interés estudiantil en esta área crítica para la ingeniería electrónica actual.

Abstract

This project presents the design and implementation of a voice command recognition system on an embedded platform aimed at controlling home automation devices. The system is based on the TMS320C5535 digital signal processor, integrating TLV320AIC3204 audio codec and a UART interface with an Arduino to toggle actuators voice commands. It includes complete stages of audio acquisition, preprocessing (silence removal, Z-score normalization, pre-emphasis, and windowing), feature extraction using Mel-frequency cepstral coeficients (MFCC), and classification using three models: Dynamic Time Warping (DTW), Gaussian Mixture Models (GMM), and Euclidean Distance with CMVN normalization. Results indicate that GMM achieves the best trade-off between accuracy (86 % under noisy conditions) and computational efficiency. A complementary lab guide was developed to enhance the learning of digital signal processing and to encourage student engagement in this key area of electronic engineering.

Lista de Figuras

Lista de Tablas

Glosario

Analytic Hierarchy Process (AHP) Método de decisión multicriterio basado en comparaciones por pares y síntesis de prioridades.

Características acústicas Parámetros extraídos de la señal de audio que representan su contenido espectral y temporal, por ejemplo, los coeficientes cepstrales en escala Mel (MFCC).

Cepstral Mean and Variance Normalization (CMVN) Normalización de coeficientes cepstrales restando media y dividiendo por desviación estándar.

Cepstrum Lineal (CC) Transformada cepstral obtenida tras FFT y logaritmo, seguida de IFFT o DCT, sin escala Mel.

Direct Memory Access (DMA) Mecanismo para transferir datos entre periféricos y memoria sin intervención de la CPU.

DSP TMS320C5535 Procesador de señales digitales optimizado para audio en tiempo real.

Dynamic Time Warping (DTW) Algoritmo de alineamiento temporal que mide la similitud entre dos series temporales, permitiendo compresiones o expansiones locales del eje temporal.

Inter-IC Sound (I^2S) Protocolo serie para transmisión de audio PCM.

Linear Predictive Coding (LPC) Técnica que modela la envolvente espectral del habla mediante combinación lineal de muestras anteriores.

Line Spectral Frequencies (LSF) Representación alternativa de los coeficientes LPC como frecuencias angulares extraídas de las raíces de dos polinomios simétricos.

MFCC Coeficientes cepstrales en escala Mel, representación compacta de la envolvente espectral perceptual de la voz.

Modelos de Mezclas de Gaussianas (GMM) Método probabilístico que modela la distribución de un conjunto de vectores de características como la suma ponderada de varias gaussianas.

Patrón Polar Directividad de un micrófono (omnidireccional, cardioide, etc.).

Phase-Locked Loop (PLL) Circuito que sincroniza fase y frecuencia de un oscilador con una señal de referencia.

Señal de audio Variación de presión sonora en el tiempo, caracterizada por parámetros físicos como frecuencia, amplitud y fase; en procesamiento digital se convierte en datos discretos mediante muestreo y cuantización siguiendo el teorema de Nyquist.

Signal-to-Noise Ratio (S/R) Relación en decibelios entre la potencia de la señal útil y la del ruido.

Sound Pressure Level Máximo (SPL Máx.) Nivel de presión sonora máxima que un micrófono puede soportar sin distorsión.

Sistemas embebidos Microcomputadoras integradas en dispositivos físicos con recursos limitados, diseñadas para operar en tiempo real con un bajo consumo energético.

Transformada Discreta del Coseno (DCT) Técnica que descompone una señal o bloque de datos en una suma de funciones coseno de frecuencias crecientes, concentrando la mayor parte de la energía en los primeros coeficientes y facilitando la compresión y el análisis espectral.

UART Interfaz serie para comunicación bit a bit sin reloj compartido.

HMM Modelo estadístico que representa sistemas con estados ocultos mediante cadenas de Markov, útil para reconocimiento de patrones y procesamiento de señales.

ANN Sistema computacional inspirado en redes neuronales biológicas, utilizado para aprendizaje automático y clasificación de datos.

Capítulo 1

Introducción

1.1. Planteamiento del problema

1.1.1. Planteamiento del Problema

En Colombia, la falta de ingenieros es una problemática significativa. Actualmente, el país enfrenta un déficit de más de 42000 profesionales [1]. Según la revista Semana, se gradúan aproximadamente 14000 ingenieros al año en áreas TIC (electrónica, sistemas o computación), muy por debajo de la demanda del mercado [2]. Este déficit se refleja en el hecho de que solo 1 de cada 5 universitarios eligen estudiar ingeniería [2].

Las causas del déficit de ingenieros son diversas y están arraigadas en factores estructurales. Una de las más significativas es la percepción negativa que existe hacia las carreras de ingeniería, consideradas por muchos jóvenes como extremadamente difíciles y exigentes [3]. Esta percepción no es infundada, ya que se relaciona con el bajo rendimiento de los estudiantes en áreas STEM [4]. Según el Ministerio de Educación, existe un desfase en la formación básica en matemáticas y ciencias [5], lo que dificulta no solo el ingreso, sino también el éxito en las carreras técnicas y de ingeniería. Este déficit educativo en etapas tempranas del aprendizaje se traduce en bajos índices de matriculación en áreas esenciales para el desarrollo tecnológico y económico del país [4].

Por otro lado, la permanencia en estas carreras enfrenta desafíos adicionales. Factores como la deserción universitaria, influenciada por problemas económicos, falta de motivación y escaso apoyo institucional, contribuyen significativamente al déficit. Las tasas de deserción en ingeniería son particularmente altas debido a la carga académica y los retos técnicos que estas carreras imponen [6].

Para abordar ambos problemas, es urgente desarrollar estrategias educativas y académicas que fomenten el interés por la ingeniería y, al mismo tiempo, promuevan la permanencia y graduación de los estudiantes. En este sentido, el fortalecimiento de áreas específicas como el procesamiento digital de señales resulta clave para motivar a los estudiantes a involucrarse en disciplinas tecnológicas de gran demanda actual.

Esta problemática se refleja en el desarrollo de la rama de procesamiento digital de señales en la Universidad Santo Tomás, una disciplina clave en áreas tecnológicas como telecomunicaciones, robótica y domótica, esencial para el análisis y manipulación de señales en diversos dispositivos y sistemas [7]. A pesar de su relevancia, en la Universidad Santo Tomás se encuentran pocos trabajos de grado que aborden de manera integral este campo [8], lo que evidencia el bajo interés por esta área y genera una brecha importante en la formación de los estudiantes y en la capacidad de la universidad para desarrollar soluciones tecnológicas de vanguardia [9].

Los dos grupos de investigación MEM y GED de la Facultad de Ingeniería Electrónica de la Universidad Santo Tomás han enfocado sus esfuerzos principalmente en el desarrollo de tecnologías aplicadas a los sistemas de energía eléctrica, a la automatización y, en el caso de GED, en robótica, y no existen registros recientes de proyectos de grado dentro de estos grupos que aborden específicamente el procesamiento digital de señales, específicamente en el título de los proyectos. Por tanto, la posibilidad de realizar investigaciones en áreas emergentes como el procesamiento de señales de audio y otras aplicaciones clave del procesamiento digital de señales es una gran oportunidad para adquirir o profundizar en el conocimiento de estos temas. Esta ausencia amplia las oportunidades para sentar las bases en este tipo de desarrollos, de tal forma que más estudiantes inicien el estudio del reconocimiento de comandos de voz para desarrollar competencias técnicas y adentrarse en una de las áreas más demandadas en la ingeniería actual [10].

En la actualidad, el reconocimiento de voz ha cobrado gran relevancia en una amplia gama de aplicaciones tecnológicas, desde asistentes virtuales hasta sistemas de control por voz en dispositivos electrónicos [11]. Sin embargo, a pesar de los avances significativos en esta área, la implementación de algoritmos de reconocimiento de voz en plataformas de procesamiento digital de señales sigue siendo un desafío técnico [12]. Las DSP, por su capacidad de procesamiento en tiempo real y optimización de recursos, representan una opción ideal para estas aplicaciones, pero su implementación requiere soluciones innovadoras que permitan un procesamiento eficiente y preciso de las señales de audio.

La problemática radica en la falta de estudios y proyectos que aborden la rama de procesamiento digital de señales. Por lo que surge la necesidad de implementar estrategias que incentiven el desarrollo y la apropiación de esta área.

1.1.2. Objetivos

1.1.2.1. Objetivo General

Diseñar un sistema de reconocimiento de comandos de voz en sistemas embebidos para el control de entornos domóticos, integrando el diseño del hardware y la evaluación de algoritmos de procesamiento digital de señales de audio, con el fin de fomentar el aprendizaje y la comprensión en esta área tecnológica.

1.1.2.2. Objetivos Específicos

1. Seleccionar el micrófono necesario para la conexión y funcionamiento en entornos domóticos, integrando y adaptando su señal de entrada a la tensión requerida por el convertidor analógico-digital de la DSC.
2. Implementar tres algoritmos de procesamiento digital de señales de voz, seleccionados a partir del estado del arte, adaptándolos al hardware seleccionado para identificar cuál ofrece el mejor desempeño en términos de precisión y carga computacional en el sistema embebido.
3. Validar el sistema de reconocimiento de comandos de voz mediante pruebas en entornos domóticos, evaluando la precisión, el tiempo de respuesta y el comportamiento en diferentes condiciones de ruido.
4. Aplicar una estrategia educativa a través de la elaboración de una guía de laboratorio, con el fin de fomentar el aprendizaje del Procesamiento Digital de Señales en la Universidad Santo Tomás.

1.2. Justificación

El proyecto aborda la problemática de la deserción universitaria en carreras de ingeniería mediante el diseño de actividades lúdicas y prácticas que incrementan la motivación estudiantil. La creación de una guía de laboratorio facilita el aprendizaje de conceptos complejos de manera interactiva, generando un entorno educativo más dinámico y atractivo. Al involucrar a los estudiantes en la resolución de desafíos reales, se busca transformar el enfoque educativo tradicional, motivando a los estudiantes a enfrentar situaciones prácticas que refuercen su interés y compromiso con su formación académica [13]. De este modo, se espera mejorar la retención de

los estudiantes, reduciendo las tasas de abandono al proporcionarles experiencias educativas significativas que los mantengan involucrados en su desarrollo profesional.

De la misma manera, este proyecto busca establecer las bases para el estudio y desarrollo del procesamiento digital de señales (DSP) en la Universidad Santo Tomás, a través de la implementación de un sistema de reconocimiento de comandos de voz en sistemas embebidos. Se busca crear un precedente que fomente investigaciones futuras en este campo, manteniendo a la universidad competitiva en un entorno donde el DSP es cada vez más relevante. Además, la consolidación de esta línea de estudio permitirá que el grupo de investigación MEM amplíe su enfoque hacia nuevas áreas aparte de las existentes, fortaleciendo su posición dentro de la facultad en términos de innovación tecnológica [10].

El proyecto también abre la posibilidad de desarrollar nuevas colaboraciones interdisciplinarias y potenciar futuras tesis que exploren mejoras en algoritmos de procesamiento, optimización de hardware avanzado y soluciones tecnológicas competitivas. De esta forma, se espera que este esfuerzo no solo diversifique las investigaciones del grupo MEM (Modelado - Electrónica - Monitoreo), sino que impulse la generación de proyectos en el ámbito académico y profesional, maximizando el impacto del PDS en diversas áreas tecnológicas.

Existen múltiples soluciones para el reconocimiento de comandos de voz, utilizando herramientas como DeepSpeech y Vosk, que se ejecutan localmente en un PC, o módulos que implementan esta funcionalidad en microcontroladores. Estas tecnologías permiten desarrollar sistemas autónomos sin depender de la nube, lo que resulta ideal para aplicaciones que requieren baja latencia o altos niveles de privacidad [14, 15]. Sin embargo, la creación de esta solución está orientada a un entorno educativo. Por lo tanto, este proyecto no solo servirá como una implementación funcional, sino también como un proceso de aprendizaje, proporcionando un entorno propicio para el desarrollo de competencias técnicas avanzadas.

La integración de este proyecto conlleva desafíos en la adaptación de un algoritmo en un microcontrolador, ya que este proceso requiere ajustar un modelo teórico existente a las restricciones y requisitos específicos del hardware, como lo describen Gonzales y Murrugarra [16]. Evaluar el rendimiento del algoritmo en este contexto justifica su implementación y enriquece el conocimiento sobre la aplicación de sistemas de reconocimiento de voz en entornos embebidos.

Con lo anteriormente mencionado, se planea emplear un procesador digital de señales, una tarjeta especializada. Al integrar este recurso en el procedimiento, se permitirá el desarrollo del proyecto con un hardware dedicado, ofreciendo una aplicación práctica de esta tecnología en un área de alta demanda.

1.3. Impacto social

El presente proyecto busca fortalecer la educación STEM mediante la incorporación de estrategias educativas, como la guía de laboratorio para el aprendizaje del procesamiento digital de señales. Al abordar el déficit de ingenieros en Colombia, se busca incentivar el interés por áreas tecnológicas clave y proporcionar a los estudiantes herramientas prácticas que mejoren su formación en disciplinas de alta demanda. Esto contribuye a cerrar la brecha educativa en ingeniería, promoviendo la permanencia y el éxito académico, y preparando a futuros profesionales para liderar desarrollos tecnológicos que impulsen el progreso económico y social del país.

Este proyecto contribuye al Objetivo de Desarrollo Sostenible 4: Educación de Calidad [17], al fomentar el estudio del procesamiento digital de señales en la universidad. La apertura de este campo no solo espera incrementar la producción académica y científica, sino que también brinda oportunidades de aprendizaje avanzado para estudiantes y docentes, promoviendo así el desarrollo de competencias investigativas en áreas emergentes.

Capítulo 2

Estado del arte

Para el desarrollo de un algoritmo de reconocimiento de voz mediante un microcontrolador es necesario comprender los avances y metodologías previas en el campo. En esta sección se propone un análisis de los desarrollos y técnicas establecidas en el ámbito del reconocimiento de voz, lo que permite contextualizar las tecnologías actuales y fundamentar las decisiones técnicas que se tomarán.

2.1. Modelos de reconocimiento de voz

Para el reconocimiento de señales de voz, existen diferentes modelos que sirven a un mismo propósito. Entre los modelos más utilizados según Dong Yu y Li Deng [18], se pueden clasificar en dos grandes categorías: los métodos basados en plantillas o patrones, y los modelos estadísticos o de aprendizaje automático. El primer grupo, los métodos basados en plantillas, utiliza técnicas de coincidencia de patrones, donde las señales de voz son comparadas con plantillas pregrabadas o patrones de referencia. Por otro lado, los modelos estadísticos o de aprendizaje automático emplean técnicas estadísticas o de aprendizaje profundo para modelar la relación entre las señales de voz y sus correspondientes palabras o frases.

Los modelos o algoritmos de reconocimiento de comandos de voz más conocidos son DTW, ANN y HMM. El algoritmo DTW, ofrece la oportunidad de comparar dos señales que en tiempo son distintas, pero en contexto son iguales, brindando un mayor porcentaje de aciertos en el reconocimiento de comandos de voz [19].

Cada uno de estos métodos se implementa en diferentes modelos de reconocimiento de voz, dependiendo de sus características y aplicaciones específicas. HMM son utilizados para modelar

secuencias temporales de voz, como en el reconocimiento continuo de palabras y frases donde la variabilidad temporal es crucial. GMM se emplean para representar la distribución de características acústicas mediante combinaciones ponderadas de gaussianas, siendo comunes en sistemas de identificación y verificación de hablantes, así como en la modelización de fonemas. Las Redes Neuronales, incluyendo MLP y RBF, son aplicadas en la clasificación de señales de voz debido a su capacidad para manejar patrones complejos y no lineales, y son particularmente útiles en sistemas modernos de reconocimiento de voz que requieren adaptación a diferentes contextos y hablantes. El KNN ofrece un enfoque más simple pero efectivo para clasificar señales de voz basándose en la similitud con ejemplos de entrenamiento, siendo útil en aplicaciones que requieren una implementación rápida y de bajo costo computacional. Finalmente, el DTW y la VQ son modelos complementarios que mejoran la precisión del reconocimiento al ajustar las secuencias de tiempo y reducir la dimensionalidad de los datos, respectivamente; DTW es útil en aplicaciones donde el tiempo de las secuencias puede variar, como en el reconocimiento de comandos de voz, mientras que VQ se emplea en sistemas que necesitan una reducción de la dimensionalidad para representar datos acústicos de manera compacta [18][20].

2.2. Métodos de extracción de parámetros de voz

Para implementar estos algoritmos se emplean diversos métodos de extracción de parámetros de voz, como lo son Mel Frequency Cepstral Coefficients (MFCC), Linear Predicting Coding (LPC) y Cepstral Coefficients (CC). Estos métodos convierten las señales de voz en parámetros que destacan características específicas de la señal, permitiendo la identificación y diferenciación de patrones acústicos relevantes según el algoritmo utilizado, son ampliamente utilizados en sistemas actuales, por ejemplo los LPC en telefonía celular y telefonía IP, los MFCC son ampliamente utilizados en el reconocimiento de voz para sistemas de dictado. Estos métodos de extracción de parámetros de voz tienen ventajas y desventajas; en cuanto al rendimiento, hay estudios que comprueban que la demanda de procesamiento para LPC es menor con respecto a la exigencia computacional para CC, y que, al aplicar ruido gaussiano a la entrada, el algoritmo que mejor porcentaje de aciertos tiene es MFCC, seguido por LPC y con un rendimiento significativamente inferior CC [21].

Se destacan diversas técnicas de extracción de parámetros de voz y sus aplicaciones en el reconocimiento de voz. La Transformada de Wavelet ha sido aplicada en investigaciones para mejorar la representación de características en señales de voz. En la investigación realizada por Andrés F. Soto se empleó la Transformada de Wavelet para la extracción de características en ambientes ruidosos. Los autores destacaron la capacidad de la Transformada de Wavelet para ofrecer una resolución variable en distintas escalas, lo que permitió una mejor captura de

las características transitorias y una mayor resistencia al ruido en comparación con métodos tradicionales [22].

Los MFCC son ampliamente utilizados debido a su eficacia en la representación de características acústicas relevantes para el reconocimiento de voz. Se emplearon MFCC para mejorar la precisión en el reconocimiento de comandos en entornos ruidosos. Los autores resaltaron que MFCC, al capturar la percepción auditiva humana, proporciona una representación compacta y eficiente de la información espectral que es crucial para la clasificación y la identificación de patrones en señales de voz [16].

Los métodos de extracción de características permiten reducir la cantidad de información de entrada al manejar un menor volumen de datos, descartando parte de la señal original. Este proceso de extracción facilita la identificación de ciertas características relevantes de la señal de voz. Por ejemplo, como detallan los autores, el uso de la Transformada Rápida de Fourier (FFT) permite discriminar frecuencias específicas presentes en las señales de voz, lo que resulta en una disminución del costo computacional y un aumento en la velocidad de procesamiento [21].

Las técnicas basadas en redes neuronales, como los Autoencoders y las Redes Neuronales Convolucionales (CNNs), han sido exploradas en trabajos recientes. Se utilizaron redes neuronales convolucionales para aprender representaciones de voz directamente a partir de datos crudos. Los autores destacaron que estas técnicas permiten la extracción automática de características relevantes, logrando mejoras en la precisión del reconocimiento de voz en comparación con métodos convencionales y mostrando una mayor robustez frente a variaciones en la señal.[23].

2.3. Implementación en microcontroladores

La implementación de algoritmos de reconocimiento de comandos de voz en microcontroladores presenta problemas por las limitaciones en memoria y capacidad de procesamiento. Estudios han demostrado que los microcontroladores de menos de 16 bits suelen tener dificultades significativas para manejar algoritmos complejos por su limitada capacidad de memoria y velocidad de procesamiento. Estas limitaciones pueden restringir la eficacia del prototipo, resultando en tiempos de procesamiento prolongados y una capacidad reducida para codificar algoritmos. [16].

Para abordar estos desafíos, se han propuesto varias soluciones. Una de las estrategias es utilizar microcontroladores de más de 16 bits, que ofrecen mayor capacidad de procesamiento y

memoria suficiente para realizar los algoritmos para reconocer comandos de voz frente a microcontroladores inferiores. Esto permite una implementación más eficiente de los algoritmos, reduciendo significativamente los tiempos de reconocimiento de voz. Además, la optimización de la memoria y el procesamiento, como la utilización de RAM interna en lugar de externa, ha demostrado ser crucial para mejorar la eficiencia del sistema [16].

Un enfoque adicional para superar las limitaciones del microcontrolador es delegar parte del procesamiento a un PC. En el proyecto realizado por Arnold Esteven Ruiz Sierra, se implementó un sistema de reconocimiento de comandos de voz utilizando MATLAB en un PC, que procesaba los comandos y enviaba los resultados al microcontrolador a través de una comunicación inalámbrica. Este método permitió evitar las restricciones de procesamiento del microcontrolador al aprovechar las capacidades avanzadas de MATLAB para el reconocimiento de voz. La comunicación RF entre el PC y el microcontrolador facilitó el control de un prototipo de silla de ruedas, mostrando cómo la integración de PC y microcontrolador puede mejorar significativamente el rendimiento y la funcionalidad de los sistemas de reconocimiento de voz; sin embargo, esta solución limita el funcionamiento del sistema a la conexión y dependencia del PC, ya que la entrada de voz solo se recibe y procesa en el PC, lo que puede restringir la portabilidad y autonomía del dispositivo en situaciones donde el acceso a un ordenador no sea viable [24].

2.4. Reconocedores de voz aplicados a sistemas domóticos

En el contexto del control por voz aplicado a la domótica, los reconocedores de comandos de voz han permitido avances significativos en la automatización del hogar, mejorando tanto la accesibilidad como la usabilidad de estos sistemas. Según Camargo [25], un sistema domótico puede controlar dispositivos del hogar a través de protocolos de comunicación como IEEE 802.15.4 y radios Zigbee, lo que facilita la interacción del usuario mediante comandos hablados para realizar acciones cotidianas como encender luces o activar sistemas de seguridad.

Por otro lado, Panta Martínez [26] aborda el uso de módulos de reconocimiento de voz como el EasyVR, específicamente diseñados para trabajar con plataformas como Arduino. Este enfoque ofrece una solución simple y accesible para implementar control por voz en entornos domóticos, permitiendo al usuario entrenar comandos personalizados que el sistema reconoce para ejecutar funciones básicas del hogar, como el manejo de la iluminación o electrodomésticos. Además, la posibilidad de trabajar en modo normal (escucha y ejecuta los comandos preestablecidos) y modo de administración (entrenar nuevos comandos de voz o borrar entrenamientos anteriores) otorga flexibilidad para adaptarse a las necesidades de cada usuario.

Finalmente, Paucar Robles [27] destaca la integración de sistemas de reconocimiento de voz en entornos virtuales a través de plataformas como OpenHAB, donde el uso de comandos vocales permite controlar dispositivos simulados en Home I/O. La implementación de este sistema, que utiliza Python como intermediario entre OpenHAB y Home I/O, demuestra cómo los comandos de voz pueden simplificar la gestión de un hogar inteligente, facilitando el control remoto desde cualquier dispositivo conectado a internet.

Capítulo 3

Marco Teórico

3.1. Contexto y fundamentos del reconocimiento de voz en sistemas embebidos

El reconocimiento de voz permite interfaces naturales con dispositivos electrónicos, por lo que su adopción en smartphones, asistentes virtuales y sistemas IoT ha crecido notablemente [28, 29]. Sin embargo, ejecutar algoritmos de reconocimiento de voz en dispositivos embebidos plantea desafíos: los recursos de CPU, memoria y energía son limitados, y el procesamiento local debe garantizar baja latencia y privacidad [28, 29]. En muchos dispositivos modernos (móviles, electrodomésticos inteligentes, automóviles) se implementa un sistema de "despertador" por voz: en lugar de reconocer continuamente todo el habla, el sistema está siempre activo escuchando un comando específico (por ejemplo "Ok Google"), de modo que se minimiza el consumo de recursos hasta que se detecta la palabra clave [28]. Este enfoque local reduce la dependencia de la nube y mejora la experiencia de usuario con tiempos de respuesta inmediatos [29]. En este contexto, el reconocimiento de comandos de voz en sistemas embebidos se perfila como un área clave de investigación e ingeniería, con aplicaciones en domótica, salud, industria y electrónica de consumo.

3.2. Definición de conceptos clave

3.2.1. Señal de audio

Una señal de audio representa las variaciones de presión sonora en el tiempo. Se caracteriza por parámetros físicos como la frecuencia (por ejemplo, la voz humana útil se encuentra aproximadamente entre 300 y 3000 Hz) [30], la amplitud (nivel sonoro) y la fase. En el procesamiento digital, la señal analógica se convierte en digital mediante muestreo periódico (típicamente a 8 kHz, 16 kHz o superior) y cuantización, generando datos discretos [31]. El teorema de Nyquist establece que la frecuencia de muestreo debe ser al menos el doble de la máxima frecuencia presente en la señal para evitar el aliasing [31]. La conversión analógica a digital (A/D) impone una resolución (en bits) y un rango dinámico; por ejemplo, un muestreo a 16 kHz y cuantización de 16 bits suelen ser suficientes para aplicaciones de voz con calidad telefónica [31].

3.2.2. Características acústicas

Las características acústicas son parámetros extraídos de la señal de audio que representan su contenido espectral y temporal. Entre las más utilizadas se encuentran los coeficientes cepstrales en frecuencia Mel (MFCC), que proporcionan una representación compacta de la envolvente espectral perceptual. El cálculo de los MFCC implica aplicar un preénfasis, segmentar la señal en ventanas (comunmente de entre 20 ms y 30 ms; el número de muestras por ventana varía según la frecuencia de muestreo), realizar una Transformada Rápida de Fourier (FFT), filtrar con bancos de filtros Mel, tomar el logaritmo de la energía y finalmente aplicar la Transformada Discreta del Coseno (DCT) [32]. Este proceso genera típicamente 12–13 coeficientes por trama y es ampliamente utilizado en sistemas de reconocimiento de voz. Además, se emplean representaciones tiempo-frecuencia como el espectrograma (STFT) para visualizar la energía en función del tiempo y la frecuencia. El cepstrum, obtenido al aplicar la transformada inversa de Fourier al logaritmo del espectro, revela estructuras periódicas asociadas a formantes y tono. Los formantes, que son los picos de resonancia del tracto vocal, caracterizan los sonidos vocálicos y su extracción mediante análisis lineal predictivo fue un método clásico en el reconocimiento de voz.

3.2.3. Modelos de reconocimiento clásicos

Antes del auge del aprendizaje profundo, dos enfoques dominantes en el reconocimiento de voz eran los modelos estadísticos basados en mezclas de Gaussianas (GMM) y los algoritmos

de alineamiento temporal como el Dynamic Time Warping (DTW). Un GMM modela la distribución probabilística de las características acústicas como una combinación de varias distribuciones Gaussianas, permitiendo aproximar cualquier densidad compleja mediante un número adecuado de componentes. Cada clase de fonema o palabra se asocia a un GMM entrenado mediante el algoritmo de Expectation-Maximization (EM) [33]. Por otro lado, el DTW es un algoritmo que alinea dos secuencias temporales (por ejemplo, una plantilla de comando y una entrada actual) permitiendo estiramientos o compresiones temporales para minimizar la distancia global entre ellas. Se implementa mediante programación dinámica sobre una matriz de costos, encontrando la ruta de menor costo que mapea la plantilla al sonido de entrada [34].

3.2.4. Sistemas embebidos

Los sistemas embebidos son microcomputadoras integradas en dispositivos físicos que deben operar en tiempo real con recursos limitados. Ejemplos típicos de hardware incluyen microcontroladores como los ARM Cortex-M y DSPs de Texas Instruments, que cuentan con CPU de baja potencia y memoria limitada. Estos sistemas tienen restricciones estrictas de latencia debido a su capacidad de computo y consumo energético. Por ejemplo, se ha implementado un algoritmo de reconocimiento de voz en un microcontrolador ARM Cortex-M4 (STM32F4), esto ilustra las limitaciones en recursos de cómputo y memoria [35]. Para aplicaciones prácticas, es esencial considerar la capacidad de funcionar sin conexión a internet o un servidor externo y la resistencia al ruido ambiental. Un sistema embebido de reconocimiento de voz se caracteriza por sus limitaciones en frecuencia de reloj, memoria RAM/ROM, consumo de potencia y la necesidad de operar continuamente sin sacrificar exactitud.

3.3. Fundamentos teóricos

3.3.1. Formulación matemática de Dynamic Time Warping (DTW)

Dynamic Time Warping (DTW) es un algoritmo de alineamiento temporal que cuantifica la similitud entre dos secuencias de longitud variable, permitiendo la compresión o expansión local del eje temporal para corregir diferencias de ritmo [34, 31].

Sea

$$X = (x_1, x_2, \ldots, x_N), \quad Y = (y_1, y_2, \ldots, y_M) \tag{3.1}$$

donde:

- X: primera secuencia de vectores de características en $\mathbb{R}^d$.
- Y: segunda secuencia de vectores de características en $\mathbb{R}^d$.
- N: longitud de la secuencia X.
- M: longitud de la secuencia Y.
- x_i: i-ésimo vector de características de la secuencia X.
- y_j: j-ésimo vector de características de la secuencia Y.
- $\mathbb{R}^d$: espacio d-dimensional de números reales, indicando que cada vector de características tiene d componentes.

dos secuencias de vectores de características en $\mathbb{R}^d$. Se define la *distancia local*

$$X = (x_1, x_2, \ldots, x_N) \tag{3.2}$$

$$Y = (y_1, y_2, \ldots, y_M) \tag{3.3}$$

$$d(i,j) = \|x_i - y_j\|_2, \quad 1 \leq i \leq N,\ 1 \leq j \leq M \tag{3.4}$$

donde:

- $d(i,j)$: distancia euclidiana entre el vector x_i de la secuencia X y el vector y_j de la secuencia Y.
- $\|\cdot\|_2$: norma euclidiana (distancia L2).

A partir de $d(i,j)$, se construye la *matriz de costes acumulados* $D \in \mathbb{R}^{N\times M}$ mediante la relación de recurrencia:

$$D(i,j) = d(i,j) + \text{mín} \begin{cases} D(i-1,j), \\ D(i,j-1), \\ D(i-1,j-1), \end{cases} \tag{3.5}$$

En los bordes de la matriz se definen las siguientes condiciones:

$$D(1,1) = d(1,1), \tag{3.6}$$

$$D(i,1) = d(i,1) + D(i-1,1), \quad 2 \leq i \leq N, \tag{3.7}$$

$$D(1,j) = d(1,j) + D(1,j-1), \quad 2 \leq j \leq M. \tag{3.8}$$

donde:

- $D(i, j)$: coste acumulado mínimo para alinear los primeros i elementos de la secuencia X con los primeros j elementos de la secuencia Y.
- $D(i-1, j)$: coste acumulado al avanzar en la secuencia X (insertar un elemento en Y).
- $D(i, j-1)$: coste acumulado al avanzar en la secuencia Y (insertar un elemento en X).
- $D(i-1, j-1)$: coste acumulado al avanzar en ambas secuencias simultáneamente (alinear un elemento de X con uno de Y).

El coste óptimo de alineamiento se encuentra en $D(N, M)$, el cual representa la menor suma de distancias locales necesaria para sincronizar toda la secuencia X con Y. Este valor se utiliza como métrica de similitud: a menor $D(N, M)$, mayor semejanza entre las dos señales.

El correspondiente camino de warping óptimo

$$W = \{(i_k, j_k)\}_{k=1}^{K} \tag{3.9}$$

se recupera retrocediendo desde (N, M) hasta $(1, 1)$, eligiendo en cada paso la celda predecesora que minimizó el coste acumulado. Este camino describe explícitamente cómo alinear cada muestra de X con cada muestra de Y, detallando compresiones y expansiones temporales. En reconocimiento de voz, el warping path permite identificar qué fragmentos de la señal de entrada corresponden a la plantilla de referencia, incluso si la pronunciación es más rápida o lenta que la grabación modelo.

Para mejorar el rendimiento en implementaciones embebidas, se suelen aplicar ventanas de restricción (banda de Sakoe–Chiba o paralelogramo de Itakura) que limitan el área de cálculo a una franja de ancho $w \ll \text{mín}(N, M)$, reduciendo la complejidad a $O(Nw)$.

3.3.2. Modelos de Mezclas de Gaussianas (GMM)

Los Modelos de Mezclas de Gaussianas (GMM) son una técnica estadística para describir distribuciones complejas de datos mediante la combinación de varias distribuciones gaussianas simples. En el contexto del reconocimiento de voz, cada GMM se utiliza para modelar la distribución de vectores de características acústicas (por ejemplo, MFCC) correspondientes a un comando concreto [36]. Al representar cada comando como una mezcla de K componentes gaussianas, el GMM puede capturar la variabilidad inherente en la pronunciación de diferentes hablantes o condiciones de grabación.

Un GMM asume que la probabilidad de observar un vector de características $x \in \mathbb{R}^d$ viene dada por

$$p(x) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}\big(x \mid \mu_k, \Sigma_k\big) \tag{3.10}$$

donde cada término es una gaussiana multivariante con media μ_k y covarianza Σ_k, y π_k es el peso de mezcla que indica la importancia de esa componente ($\sum_{k=1}^{K} \pi_k = 1$).

El entrenamiento de un GMM se realiza mediante el algoritmo Expectation–Maximization (EM) para maximizar la verosimilitud de un conjunto de N vectores de entrenamiento $\{x^{(n)}\}$. En la fase E se calculan las responsabilidades

$$\gamma_k^{(n)} = \frac{\pi_k \, \mathcal{N}\big(x^{(n)} \mid \mu_k, \Sigma_k\big)}{\sum_{j=1}^{K} \pi_j \, \mathcal{N}\big(x^{(n)} \mid \mu_j, \Sigma_j\big)} \tag{3.11}$$

y en la fase M se actualizan los parámetros:

$$\pi_k \leftarrow \frac{1}{N} \sum_{n=1}^{N} \gamma_k^{(n)} \tag{3.12}$$

$$\mu_k \leftarrow \frac{\sum_{n=1}^{N} \gamma_k^{(n)} \, x^{(n)}}{\sum_{n=1}^{N} \gamma_k^{(n)}} \tag{3.13}$$

$$\Sigma_k \leftarrow \frac{\sum_{n=1}^{N} \gamma_k^{(n)} \, (x^{(n)} - \mu_k)(x^{(n)} - \mu_k)^{\mathsf{T}}}{\sum_{n=1}^{N} \gamma_k^{(n)}} \tag{3.14}$$

Para el reconocimiento de comandos de voz, se entrena un GMM por cada comando del vocabulario. Dada una nueva secuencia de MFCC $X = \{x^{(n)}\}$, se evalúa la probabilidad conjunta bajo cada modelo:

$$p(X \mid \Theta_c) = \prod_{n=1}^{N} p\big(x^{(n)} \mid \Theta_c\big) \tag{3.15}$$

o equivalentemente su log-verosimilitud, eligiendo como comando reconocido aquel cuyo GMM maximiza dicho valor [37].

Entre sus ventajas destacan su flexibilidad para aproximar densidades arbitrarias y la eficiencia en evaluación, mientras que el coste de entrenamiento (EM) y la suposición de independencia temporal suelen mitigarse realizando el ajuste fuera de línea y añadiendo coeficientes delta y delta–delta para capturar dinamismo temporal.

3.3.3. Formulación teórica de los Coeficientes Cepstrales en Frecuencia Mel (MFCC)

Los coeficientes cepstrales en frecuencia Mel (MFCC) son la representación de características acústicas más utilizada en reconocimiento de voz, tanto en sistemas convencionales como en entornos embebidos. Su principal ventaja radica en capturar la envolvente espectral de la señal de voz de manera compacta y alineada con la percepción humana: el eje de frecuencia se transforma a la escala Mel, en la que las diferencias de altura tonal son percibidas de forma aproximadamente lineal en las bajas frecuencias y logarítmica en las altas [38].

El cálculo de los MFCC combina herramientas de procesamiento de señales y transformadas matemáticas:

- *Preénfasis.* Se aplica un filtro pasa-alto de primer orden a la señal para realzar las componentes de alta frecuencia:

$$y[n] = x[n] - \alpha\, x[n-1], \quad \alpha \approx 0{,}97 \tag{3.16}$$

- *Segmentación en ventanas.* La señal preénfasis se divide en tramas cortas (20–25 ms) con solapamiento (por ejemplo, 10 ms) para asumir que la voz es estacionaria dentro de cada ventana.

- *Ventaneo.* Cada trama se multiplica por una ventana para reducir las discontinuidades en los extremos y minimizar el efecto de filtrado indeseado conocido como leakage en la FFT [39]. Se pueden emplear diferentes tipos de ventanas, como la ventana rectangular, Hanning, Blackman, o Kaiser, cada una con características específicas en cuanto a el ancho del lóbulo principal y la atenuación de los lóbulos laterales. Sin embargo, en el cálculo de MFCC, la ventana de Hamming es la más comúnmente utilizada debido a que ofrece un buen compromiso entre resolución espectral y supresión de lóbulos laterales.

- *Transformada Rápida de Fourier (FFT).* Se calcula la DFT de cada ventana para obtener el espectro de magnitud; normalmente se toma solo la mitad positiva del resultado debido a la simetría.

- *Bancos de filtros Mel.* Al espectro lineal se le aplica un banco de M filtros triangulares espaciados en la escala Mel. La escala Mel se define como

$$\mathrm{Mel}(f) = 2595 \log_{10}\!\Big(1 + \frac{f}{700}\Big) \tag{3.17}$$

  y se emplea su inversa para diseñar las bandas de cada filtro.

- *Logaritmo de energías de banda.* Se toma el logaritmo natural de la energía de salida de cada filtro, modelando la percepción humana de diferencias de nivel.
- *Transformada Discreta del Coseno (DCT).* Se aplica la DCT al vector de log-energías para decorrelacionar las componentes y concentrar la información en los primeros coeficientes. Normalmente se descarta el primer coeficiente (energía global) y se conservan los siguientes 12–13 coeficientes para representar cada ventana [38].

El resultado es una secuencia de vectores de dimensión reducida (por ejemplo $\mathbb{R}^{12}$) que resumen la envolvente espectral perceptual de la voz. Gracias a la DCT, los primeros coeficientes codifican las características más relevantes, mientras que los últimos capturan detalles finos del espectro. Esta representación es adecuada para clasificadores estadísticos (GMM, HMM) o redes neuronales, y es computacionalmente eficiente para sistemas embebidos.

3.4. Marco conceptual

Un diagrama conceptual de un sistema embebido de reconocimiento de comandos engloba los siguientes bloques encadenados.

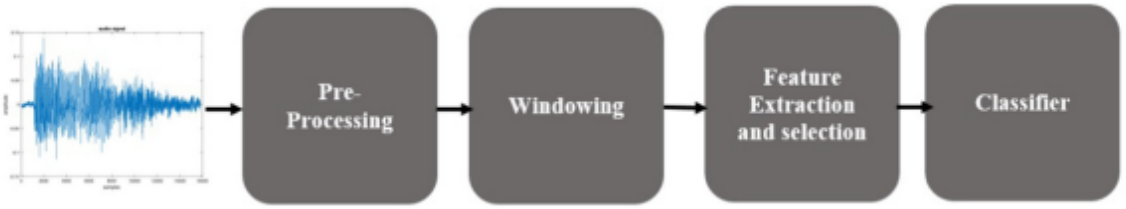

FIGURA 1: Diagrama metodológico de un sistema de reconocimiento de voz.
Tomado de [40].

El proceso de reconocimiento de comandos de voz puede estructurarse como una secuencia de etapas bien definidas, cada una de las cuales cumple una función específica en la transformación de la señal de audio en una decisión clasificada. Esta estructura metodológica, ampliamente reconocida en la literatura sobre procesamiento de señales acústicas, es descrita por Sharma et al. [40] y se ilustra en el diagrama de bloques incluido.

1. **Preprocesamiento (Pre-Processing):**
   En esta primera etapa, la señal de voz cruda es acondicionada para facilitar su posterior análisis. Las tareas comunes incluyen la eliminación de ruido, la reducción de silencios y la normalización de amplitudes. El objetivo es mejorar la relación señal-ruido y asegurar que la señal se encuentre en un rango adecuado para el procesamiento digital posterior.

2. **Ventaneo (Windowing):**
   Dado que las señales de voz son no estacionarias, es necesario dividirlas en tramas o ventanas de corta duración (usualmente entre 20 y 30 ms), dentro de las cuales se puede asumir estacionariedad. Esta segmentación permite capturar de manera precisa las propiedades acústicas que varían en el tiempo.

3. **Extracción y selección de características (Feature Extraction and Selection):**
   En esta etapa se aplica un conjunto de transformaciones para extraer características discriminativas que representen de forma compacta la señal de voz. Métodos como los Coeficientes Cepstrales en Frecuencia Mel (MFCC), LPC o GTCC son comúnmente utilizados. La selección posterior de características busca reducir la dimensionalidad y conservar únicamente aquellas que aportan mayor información para la tarea de clasificación.

4. **Clasificación (Classifier):**
   Finalmente, las características seleccionadas se ingresan a un clasificador —como GMM, HMM, DTW o redes neuronales— que ha sido entrenado previamente con ejemplos etiquetados. El clasificador asigna una etiqueta (por ejemplo, *luz*, *alarma*, etc.) al segmento de audio de entrada, permitiendo traducir la voz humana en una instrucción accionable por el sistema.

Cabe destacar que, tal como se indica en Sharma et al. [40], el clasificador es una parte fundamental del algoritmo de reconocimiento, por lo que al cambiar el clasificador se modifica el algoritmo completo. Esto se debe a que el desempeño y funcionamiento del sistema dependen directamente del método de clasificación empleado, y no solo de etapas previas de procesamiento o extracción de características. Por tanto, aunque el flujo general de procesamiento se mantenga, el cambio del clasificador implica una alteración sustancial del algoritmo global.

De igual manera, en la Universidad Santo Tomás se desarrolló una tesis sobre procesamiento digital de señales de voz [41], en la que se describen las fases principales del procesamiento digital de señales aplicadas al reconocimiento de comandos de voz: captura de voz, preprocesamiento, segmentación, extracción de características y modelos de clasificación. En este proyecto de grados se ha integrado la segmentación dentro de la etapa de preprocesamiento.

Las hipótesis de trabajo habituales asumen condiciones acústicas moderadas (voz clara y ruido de fondo limitado) y uso de hardware embebido de recursos definidos (por ejemplo, MCU ARM Cortex-M con DSP incorporado [42]). Bajo estas premisas, el sistema se evalúa la precisión del comando reconocido manteniendo las restricciones de tiempo real y bajo consumo.

Capítulo 4

Diseño Metodológico

En esta sección se presenta el diseño metodológico del proyecto, el cual incluye el enfoque general adoptado para el desarrollo, evaluación y validación del sistema propuesto. Se mostrará el diagrama de la metodología implementada, así como una descripción detallada de cada una de sus fases. El objetivo es evidenciar el proceso estructurado seguido para alcanzar los objetivos planteados y justificar las decisiones técnicas tomadas durante el desarrollo del proyecto.

El enfoque metodológico adoptado es de tipo cuantitativo y experimental, ya que se busca implementar y evaluar la aplicación de algoritmos de reconocimiento de voz en un sistema embebido, midiendo su rendimiento en términos de precisión y tiempo de respuesta. Asimismo, se trata de una investigación aplicada, en tanto se orienta a la solución de un problema práctico mediante el desarrollo de una solución tecnológica funcional.

El método experimental considera la implementación técnica de los algoritmos seleccionados en el hardware definido, la ejecución de pruebas controladas que permitan validar su funcionamiento y la obtención de datos objetivos para realizar un análisis comparativo del desempeño. De esta manera, se podrá determinar qué algoritmos ofrecen mejores resultados bajo diferentes condiciones de prueba.

El desarrollo metodológico se basa en una combinación de trabajo en paralelo y trabajo retroalimentado. El trabajo en paralelo permite que varias actividades técnicas se ejecuten simultáneamente, como el diseño del hardware, la programación y las pruebas individuales. Por su parte, la retroalimentación entre fases garantiza que los resultados obtenidos en cada etapa puedan influir en decisiones posteriores, permitiendo ajustes o mejoras continuas para optimizar el sistema final.

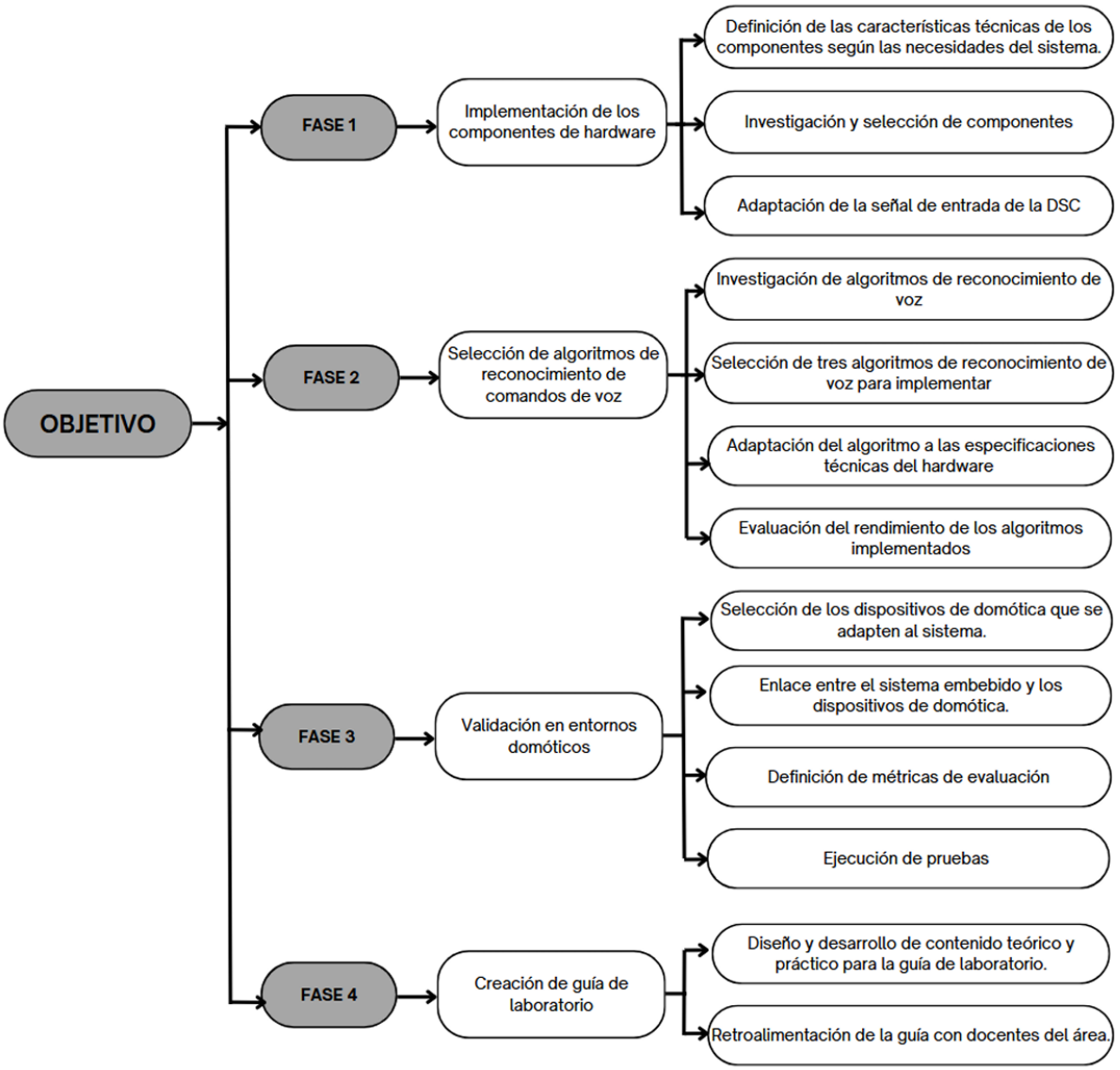

FIGURA 2: Diagrama metodológico.
Tomado de: Autores

4.1. Fase 1: Implementación de los componentes de hardware

En la primera fase, se lleva a cabo una investigación y selección de los componentes, como micrófonos y elementos de acondicionamiento de señal. El foco de la fase está en la configuración e integración del hardware esencial para el funcionamiento del sistema de reconocimiento de voz en un entorno embebido. Inicialmente, se realiza la adaptación de la señal de entrada al procesador digital de señales (DSC), acondicionándola de manera que cumpla con las características necesarias para el periférico del dispositivo. Esta investigación permite identificar los elementos que mejor se adapten a los requisitos de la aplicación del sistema. Para finalizar

esta fase, se definen y documentan las características técnicas de los componentes seleccionados, alineándolos con las necesidades específicas del sistema. Estas actividades aseguran que el sistema tenga una base de hardware adecuada para la ejecución de los comandos de voz.

4.2. Fase 2: Selección de algoritmos de reconocimiento de comandos de voz

La segunda fase se centra en la elección de los algoritmos de reconocimiento de voz y en su adaptación al sistema de hardware previamente establecido. En primer lugar, se realiza una investigación de algoritmos de reconocimiento de voz, analizando diferentes opciones en términos de precisión, carga computacional y compatibilidad. Tras esta investigación, se seleccionan tres algoritmos que, en principio, se consideran viables para su implementación, lo que permitirá una comparación objetiva durante la fase de pruebas. Luego, cada algoritmo es adaptado a las especificaciones técnicas del hardware, realizando ajustes en los parámetros que aseguren su funcionamiento. Finalmente, se evalúa el rendimiento de los algoritmos implementados, con el fin de identificar cuál ofrece el mejor balance entre precisión y tiempo de respuesta, atendiendo a los requisitos específicos del sistema.

4.3. Fase 3: Validación en entornos domóticos

Esta fase consiste en la integración del sistema en un entorno domótico para comprobar su funcionalidad en el control de electrodomésticos. Primero, se lleva a cabo la selección de dispositivos de domótica que sean compatibles y se adapten al sistema. A continuación, se establece un enlace entre el sistema embebido de reconocimiento de voz y los dispositivos de domótica, lo cual permite que los comandos de voz ejecuten las acciones programadas. Luego, se definen métricas de evaluación que permitirán medir el rendimiento del sistema, evaluando su precisión, velocidad de respuesta y tasa de aciertos. Por último, se realizan pruebas prácticas en diferentes entornos para validar el funcionamiento del sistema, midiendo los parámetros seleccionados. Esta fase garantiza medir como opera el prototipo en un entorno final.

4.4. Fase 4: Diseño y Retroalimentación de la Guía de Laboratorio

En la fase final, se lleva a cabo el diseño y desarrollo de contenido teórico y práctico para la guía de laboratorio, que incluye explicaciones detalladas de los conceptos, procedimientos y objetivos de las actividades a realizar. Este contenido debe ser claro, preciso y adecuado al nivel de conocimiento de los estudiantes, asegurando que puedan comprender y aplicar los conceptos en los experimentos prácticos de manera efectiva. Además, se desarrollarán los materiales complementarios, como ilustraciones, tablas, ejercicios y ejemplos prácticos, que faciliten la comprensión y el aprendizaje. Por último, se realiza la retroalimentación con el docente del área.

Capítulo 5

Desarrollo Conceptual

Este capítulo describe detalladamente las actividades, técnicas y procedimientos empleados para el diseño, implementación y evaluación del sistema de reconocimiento de comandos de voz en un entorno embebido.

5.1. Implementación de los componentes de hardware

En esta fase se definen los fundamentos físicos y las características técnicas imprescindibles para el desarrollo del sistema de reconocimiento de voz en un entorno embebido. El objetivo es asegurar que cada componente del sistema en particular, los destinados a la adquisición y procesamiento de la señal de audio, así como los encargados del almacenamiento y la comunicación cumpla con los requisitos de funcionamiento y sean compatibles entre sí.

5.1.1. Descripción de características técnicas

Para realizar la aplicación es necesario cumplir unas especificaciones técnicas para el desarrollo del sistema, estructuradas en función de los requerimientos de adquisición de audio, almacenamiento, procesamiento, y comunicación/expansión:

Captación de Audio

- **Sensibilidad y respuesta en frecuencia:** Se requiere un micrófono capaz de captar señales de voz con alta fidelidad, con una respuesta en frecuencia adecuada para el muestreo de

la voz. En aplicaciones de voz se suele trabajar en un rango de aproximadamente 300 Hz a 3.4 kHz, aunque para capturar matices adicionales puede considerarse un rango extendido hasta 8 kHz [43].

- **Micrófono omnidireccional:** Se debe emplear un micrófono omnidireccional con buena sensibilidad y bajo nivel de ruido propio, capaz de captar señales de voz desde cualquier dirección. Esto es fundamental para asegurar una recepción uniforme del sonido, especialmente en ambientes abiertos o no controlados [44].

Conversión Analógica a Digital (ADC)

- **Resolución:** Se recomienda utilizar un conversor con al menos 12 bits de resolución, aunque 16 bits serían ideales para capturar de forma precisa las variaciones en la señal de voz [44].
- **Frecuencia de muestreo:** La tasa de muestreo debe ser lo suficientemente alta para representar adecuadamente la información de la voz, con un valor típico de 16 kHz en sistemas embebidos orientados al procesamiento de voz. Sin embargo, tasas entre 8 kHz y 16 kHz son comunes para reconocimiento de comandos [31].

Procesamiento en Tiempo Real

- La plataforma debe ser capaz de ejecutar algoritmos de extracción de características (como los coeficientes MFCC o métodos LPC) de forma inmediata para posibilitar el reconocimiento de los comandos en condiciones de operación dinámicas [31].

Capacidades de almacenamiento

- **Expansión mediante almacenamiento externo:** Una interfaz para tarjetas de memoria (como un conector Micro SD) resulta ventajosa para extender el almacenamiento. Esto permitirá registrar datos de pruebas, almacenar bases de datos de comandos o gestionar archivos de entrenamiento sin depender exclusivamente de la memoria interna.

5.1.2. Selección de componentes

A continuación, se plantea la selección de los componentes fundamentales para el desarrollo del sistema, teniendo en cuenta los requisitos técnicos previamente establecidos. Se abordará

la elección del microcontrolador encargado del procesamiento de señales, el micrófono utilizado para la captación de audio considerando criterios de compatibilidad, disponibilidad y adecuación funcional para la aplicación propuesta.

5.1.2.1. Microprocesador

Para esta etapa del proyecto se eligió como plataforma de desarrollo la TMS320C5535 eZdsp, fundamentando la decisión en tres aspectos clave:

- **Disponibilidad institucional:** El laboratorio de la Facultad cuenta con varias unidades de la eZdsp5535, lo cual garantiza acceso inmediato a hardware de prueba sin necesidad de adquisiciones especiales.
- **Concordancia con los requisitos técnicos:**
  - **Procesamiento en tiempo real:** El DSP TMS320C5535 está optimizado para operaciones de señal y puede manejar los cálculos necesarios para extracción de MFCC o LPC en tiempo real, cumpliendo con la baja latencia exigida. Opera a una frecuencia de hasta 100 MHz, lo que le permite ejecutar instrucciones en ciclos de tan solo 10 ns.
  - **Conversión analógica–digital de audio:** Incorpora el códec estéreo **TLV320AIC3204**, que ofrece entrada y salida de audio analógico–digital, permitiendo muestreo de voz con resolución y rango de frecuencia adecuados para comandos de voz.
  - **Almacenamiento interno y externo:** El TMS320C5535 dispone de 320 kB de memoria RAM interna de acceso sin espera (zero-wait state), compuesta por 64 KB de RAM de doble acceso (DARAM) y 256 KB de RAM de acceso simple (SARAM), además de 128 KB de memoria ROM interna, lo que permite almacenar y procesar datos de forma eficiente en sistemas embebidos. Adicionalmente, se incorpora un conector MicroSD con capacidad de 2GB para almacenamiento externo, útil para guardar modelos o registrar datos de voz.
  - **Interfaces de comunicación:** Proporciona puertos USB (para depuración y alimentación), buses serie (SPI, I²C, UART) e interfaces de audio digital (I²S), esenciales para interconectar periféricos y módulos adicionales según se requiera.

5.1.2.2. Micrófono

Para la implementación del sistema de reconocimiento de comandos de voz, la correcta captación de audio es crítica. La selección del micrófono se centró en modelos que cumplieran con

las características esenciales de patrón polar omnidireccional y conector de salida analógica de 3.5 mm.

Se preseleccionaron tres modelos de micrófono para el análisis comparativo, que representan diferentes opciones de disponibilidad y costo, todos dentro del presupuesto establecido de 300.000 COP. Se incluyó el iTalk-01 por su costo nulo y disponibilidad institucional. Los otros dos micrófonos fueron elegidos para cubrir distintos segmentos de precio, reconociendo que la mayoría de los micrófonos omnidireccionales con conector de 3.5 mm en rangos de precios similares presentan características técnicas muy parecidas.

1. **Micrófono iTalk-01:** Disponible en la institución, lo que implica un **costo de adquisición nulo** y disponibilidad inmediata para el proyecto.
2. **Micrófono BOYA BY-M1:** Seleccionado como un representante del segmento de micrófonos **omnidireccionales** de bajo costo, con características técnicas sólidas y amplia disponibilidad en el mercado.
3. **Rode SmartLav+:** Incluido como una opción de gama media-alta, omnidireccional, que permite evaluar si sus prestaciones superiores justifican una mayor inversión.

TABLA 1: Comparación de Micrófonos Analógicos

Parámetro	**iTalk-01 (A)**	**BOYA BY-M1 (B)**	**Rode SmartLav+ (C)**
Sensibilidad	-37 ± 3 dB	-30 dB (± 3 dB)	-35 dB
Respuesta Frecuencia	10 Hz – 10 kHz	65 Hz – 18 kHz	20 Hz – 20 kHz
Relación S/R	70 dB	74 dB+	67 dB
SPL Máx.	60 dB	115 dB	110 dB
Patrón Polar	Omnidireccional	Omnidireccional	Omnidireccional
Conector	3.5mm TRS	3.5mm TRRS (adapt. incl.)	3.5mm TRRS (sin adapt.)
Long. Cable	1.5 m	6 m	1.2 m
Precio adquisición	-	120.000 COP	250.000 COP

Descripción de los parámetros

- **Sensibilidad:** amplitud de señal de salida ante un nivel de presión sonora estándar; valores más altos (dB más cercanos a 0) reducen la necesidad de ganancia.
- **Respuesta en frecuencia:** rango de frecuencias captadas; relevante para reproducir armónicos de la voz.
- **Relación Señal/Ruido (S/R):** diferencia entre señal útil y ruido interno; mayor S/R implica grabaciones más limpias.

- **SPL Máximo:** presión sonora máxima soportada sin distorsión; importante ante variaciones bruscas de volumen.
- **Patrón Polar:** define directividad: omnidireccional capta de todas direcciones, cardioide prioriza el frente.
- **Conector / Salida:** interfaz física para el DSP; aquí todas son TRS de 3.5 mm.
- **Longitud de cable:** extensión disponible para la instalación.
- **Precio:** coste de adquisición; incorporado como criterio *Costo = 0 USD* para iTalk-01.

Para tomar una decisión objetiva mediante el software SUPERDECISIÓN, se modelará un AHP con estos tres micrófonos y los criterios definidos.

Modelado y Evaluación con Super Decisions

Se construyó la estructura jerárquica en Super Decisions, tal como se muestra en la Figura 3 . El objetivo se conecta a los criterios y estos, a su vez, se evalúan respecto a las alternativas disponibles. Los criterios técnicos considerados, derivados de la tabla anterior, fueron: Sensibilidad, Respuesta en Frecuencia, Relación Señal/Ruido (S/R), SPL Máximo y Patrón Polar. Adicionalmente, se incluyó el criterio fundamental de "Costo".

FIGURA 3: Estructura del modelo AHP en Super Decisions.

El siguiente paso consistió en efectuar comparaciones por pares para determinar la importancia relativa de cada criterio. Empleando la escala fundamental de Saaty (1–9), se evaluó cada criterio frente a los demás, con lo que se definieron sus prioridades relativas. De manera destacada, el criterio Costo obtuvo puntuaciones más altas al compararlo con los restantes.

La Tabla 2 resume los pesos obtenidos para cada criterio:

TABLA 2: Pesos obtenidos por criterio mediante el método AHP

Criterio	Peso
Costo	0.38042
Sensibilidad	0.29136
Relación señal/ruido (S/R)	0.17791
Respuesta en frecuencia	0.11387
Presión sonora máxima (SPL Max)	0.03644

Resultados del Análisis AHP

Una vez completadas todas las comparaciones, el software SUPER DECISIONS sintetiza los juicios para obtener las prioridades globales de cada alternativa. Los resultados normalizados, que suman 1 en total, se muestran en la Tabla 3:

TABLA 3: Prioridades globales normalizadas de las alternativas

Micrófono	Normal
iTalk-01	0.418383
Boya BY-M1	0.398988
Rode SmartLav+	0.182629

Según estas prioridades globales:

1. **iTalk-01:** obtiene la prioridad más alta (0.418383), consolidándose como la opción principal debido a su costo nulo y disponibilidad inmediata en la institución.
2. **Boya BY-M1:** se sitúa en un muy cercano segundo lugar (0.398988). Su excelente relación calidad-precio y sus sólidas especificaciones técnicas lo posicionan como una alternativa altamente competitiva y muy viable.
3. **Rode SmartLav+:** ocupa la tercera posición (0.182629), debido a su mayor coste relativo, a pesar de sus altas prestaciones técnicas.

El análisis AHP ponderó el desempeño de cada micrófono de acuerdo con la importancia relativa de sus criterios. De forma destacada, el criterio de Costo obtuvo la mayor ponderación, favoreciendo de manera significativa al iTalk-01. Aunque el Boya BY-M1 muestra un mejor rendimiento en la relación señal/ruido y en el SPL máximo, su prioridad global es prácticamente equiparable a la del iTalk-01, la ventaja decisiva de este último radica en que la institución lo proporciona sin coste alguno.

La proximidad en las prioridades entre el iTalk-01 y el Boya BY-M1 subraya que, si bien el factor económico fue preponderante, las capacidades técnicas del Boya lo convierten en una opción viable. No obstante, dada la disponibilidad del iTalk-01 en la institución y la ausencia de una inversión directa requerida, este micrófono se presenta como la selección más objetiva para el proyecto, sin comprometer los requisitos fundamentales de captación de audio para el reconocimiento de comandos de voz. Se enfatiza que, en ausencia de la ventaja del costo cero, el Boya BY-M1 sería la mejor alternativa.

5.1.3. Adaptación de señal y configuración de periféricos

En esta última actividad de la Fase 1 se llevará a cabo la adaptación de la señal de voz desde su captura analógica hasta su entrada al DSP, así como la configuración de los periféricos asociados. Esto implica describir el codec y el bus de audio, ajustar niveles y filtros necesarios, y parametrizar los pines y protocolos de comunicación para garantizar la correcta adquisición de la señal de voz.

Periférico de entrada de audio Para la adquisición de la señal de voz se emplea el **TLV320AIC3204**, un codec estéreo que actúa como puente entre el dominio analógico y digital del sistema.

- **Canales de entrada:**
  - *LINE2L (AIC_LINE2L):* Canal izquierdo de línea.
  - *LINE2R (AIC_LINE2R):* Canal derecho de línea.
- **Tipo de conector:**
  - Entrada analógica de línea a través del conector J3, compatible con jack estéreo de 3.5 mm.
- **Interfaz digital:**
  - Comunicación con el DSP mediante el bus I²S, que transporta las muestras muestreadas de ambos canales en serie.

A continuación, se describe, a partir de las tablas del *TLV320AIC3204 Application Reference Guide*, qué bloques internos se activan y qué parámetros de reloj, audio y potencia se establecen en esta fase de inicialización.

5.1.3.1. Inicialización del códec TLV320AIC3204

A continuación, se presenta la secuencia de configuración del códec TLV320AIC3204, necesaria para la correcta adquisición y reproducción de señales de voz. Esta configuración incluye el control de potencia, generación de reloj, selección de frecuencia de muestreo, activación de la interfaz I²S y rutado de señales de entrada y salida.

Operación	**Registro (Pág. / Dir.)**	**Campo clave**	**Función**
Software Reset	Page 0 / Reg 1 ($P0_R1$)	D0 = 1	Inicia un software reset de todo el codec, devolviendo cada bloque a su estado por defecto.
Selección de página	Page 0 / Reg 0 y Page 1 / Reg 0	—	Cambia entre mapa de registros "página 0"(sistema/clocks) y "página 1"(potencia/audio).
Conexión AVDDDVDD	Page 1 / Reg 1 ($P1_R1$)	D3 = 1	Deshabilita la conexión pasiva AVDD→DVDD; fuerza que AVDD provenga únicamente de su LDO interno.
LDOs y bloques analógicos	Page 1 / Reg 2 ($P1_R2$)	D3 = 0, D0 = 1	Enciende la LDO de AVDD y habilita todos los bloques analógicos internos.
Power up de la referencia	Page 1 / Reg 123 ($P1_R123$)	D2 D0 = 101	Fuerza el encendido de la referencia bandgap con tiempo de subida de 40 ms.

TABLA 4: Reset y configuración de potencia del códec.

Fuente: *TLV320AIC3204 Application Reference Guide*, Texas Instruments, [45].

Reset y potencia de bloques analógicos: En esta primera etapa se asegura un arranque limpio y estable del códec. Se realiza un reset, se selecciona la página de control de potencia, se configura la fuente AVDD y se activan los bloques analógicos. Finalmente, se establece la referencia interna para garantizar estabilidad antes de pasar a configuraciones más complejas.

Parámetro	Registro	Campo clave	Descripción
Fuente PLLCLK / CODEC_CLKIN	P0_R4	D1 D0 = 11	PLLCLK ← MCLK (12 MHz); CODEC_CLKIN ← salida del PLL
Multiplicador J	P0_R6	D7 D0 = 0x08	J = 8 en F_{PLL} = MCLK · (J + D / 2^{16})
División fraccional D	P0_R7, R8	0x02, 0x00	D = 512 (0x0200) para ajuste fino
Power up del PLL	P0_R5	0x91	Enciende el PLL y espera 20 ms para lock

TABLA 5: Configuración del PLL interno
Fuente: *TLV320AIC3204 Application Reference Guide*, Texas Instruments, [45].

Generación de reloj maestro: Se configura el PLL interno para generar un reloj maestro estable (CODEC_CLKIN), necesario para el procesamiento digital de señales.

Canal	Registro	Campo clave	Comentario
DAC	P0_R11 (NDAC)	0x84	NDAC = 4, activa salida DAC
	P0_R12 (MDAC)	0x8C	MDAC = 12
	P0_R13/14 (DOSR)	0x00, 0x40	DOSR = 64
ADC	P0_R18 (NADC)	0x84	NADC = 4
	P0_R19 (MADC)	0x8C	MADC = 12
	P0_R20 (AOSR)	0x40	AOSR = 64

TABLA 6: Configuración de sobremuestreo y decimación
Fuente: *TLV320AIC3204 Application Reference Guide*, Texas Instruments, [45].

Divisores de muestreo: Aquí se configuran las tasas de muestreo de 16 kHz para los canales de entrada y salida, con sobremuestreo y filtrado digital interno optimizado para calidad de audio.

Parámetro	Registro	Bits clave	Descripción
Modo maestro	P0_R27	D3=1, D2=1, D0=1	BCLK y WCLK generados por el codec
Formato y ancho	P0_R30	0x88	I²S estándar, 16 bits por canal (32 por trama)

TABLA 7: Configuración de la interfaz I²S
Fuente: *TLV320AIC3204 Application Reference Guide*, Texas Instruments, [45].

Interfaz I^2S: Con esta configuración, el codec genera su propio reloj serial y establece el protocolo de audio estándar I²S de 16 bits, listo para transmitir señales PCM al DSP.

Función	Registro	Campo clave	Efecto
Ruteo LDAC/RDAC a HPL/HPR	P1_R12, R13	D3 = 1	Conecta DACs a amplificadores de salida
Volumen digital DAC	P0_R64, R65	0x02, 0x30	Ganancia digital +3 dB
Power up rutas DAC	P0_R63	D7 D6 = 11	Activa NDAC, MDAC y salida a auriculares
Ganancia HPL/HPR	P1_R16, R17	0x00	Amplificadores a 0 dB
Power up HPL/HPR	P1_R9	D5 = 1, D4 = 1	Alimenta los drivers de auriculares

TABLA 8: Configuración de ruta de salida de audio
Fuente: *TLV320AIC3204 Application Reference Guide*, Texas Instruments, [45].

Salida de audio: Esta etapa deja habilitado el canal de salida con ganancia controlada y señal dirigida a los auriculares, lista para reproducir audio.

Función	Registro	Campo clave	Efecto
Selección de entrada mic	P1_R52, R55	0x30	Ruta IN2_L/R hacia LADC/-RADC
Terminal negativa PGA	P1_R54, R57	0x03, 0xC0	Conexión a CM_1
Bias de micrófono	P1_R51	0x48	MICBIAS a 2.5 V desde AVDD
Ganancia PGA	P1_R59, R60	0x48	Amplificación de +24 dB
Power up ADC y desmute	P0_R81, R82	0xC0, 0x00	Enciende y activa los ADC

TABLA 9: Configuración de entrada de micrófono
Fuente: *TLV320AIC3204 Application Reference Guide*, Texas Instruments, [45].

Entrada de audio (rutado y ganancia ADC): Se completa así la configuración del canal de entrada, habilitando la captura de voz desde un micrófono con alimentación y ganancia adecuada.

Cada uno de estos pasos corresponde a un bloque de control del codec TLV320AIC3204. Con estas configuraciones:

- El codec arranca con su configuración por defecto.
- Los bloques analógicos están energizados.
- El PLL genera el reloj interno estable.
- Se establece una tasa de muestreo de 16 kHz.
- Se configura la interfaz I²S como maestro.

- Se rutean y ajustan las salidas de auriculares.
- Se rutean y ajustan las entradas de micrófono.

Con esto, se completan las configuraciones esenciales de potencia, reloj y audio del codec, y el sistema está preparado para adquirir y procesar señales de voz.

El código correspondiente a la configuración de periféricos puede consultarse en el archivo 7.1

5.2. Selección de algoritmos de reconocimiento de comandos de voz

En esta segunda fase del proyecto se centra en la elección y adecuación de los algoritmos de reconocimiento de voz al hardware definido previamente. Primero, se lleva a cabo una investigación de diferentes propuestas, valorando su precisión, carga computacional y compatibilidad con el entorno. A partir de ese análisis, se escogen tres algoritmos viables que permitirán realizar una comparación objetiva en la etapa de pruebas. Posteriormente, cada uno de ellos se ajusta a las especificaciones técnicas del sistema, optimizando sus parámetros para asegurar un funcionamiento estable. Por último, se evalúa su rendimiento en términos de precisión y tiempo de respuesta, con el fin de identificar el algoritmo que mejor satisfaga los requisitos del proyecto.

5.2.1. Investigación de algoritmos de reconocimiento de voz

5.2.1.1. Entrada de Audio

La primera etapa de cualquier sistema de reconocimiento de voz consiste en adquirir la señal vía micrófono y convertirla a formato digital. Se emplean habitualmente micrófonos de condensador electret u otros tipos (dinámicos, de condensador a válvula) que ofrecen sensibilidades de –40 a –60 dBV/Pa y una respuesta en frecuencia optimizada para la voz humana (300 Hz – 5 kHz, pudiendo extenderse hasta 8 kHz para mejorar la inteligibilidad) [43, 44]. Antes de la digitalización, la señal pasa por un preamplificador de bajo ruido (<20 dBA) y un filtro antialiasing con pendiente de al menos 18 dB/octava, garantizando que no entren componentes por encima de la mitad de la frecuencia de muestreo.

Siguiendo el teorema de Nyquist, la tasa de muestreo se sitúa típicamente entre 8 kHz (telefonía) y 16 kHz (aplicaciones embebidas de mayor fidelidad), optando por 16 kHz y 16 bits de

resolución [31]. Esta configuración genera flujos de datos de 256 kb/s, que pueden gestionarse con DMA en modo ping-pong sobre un DSP, minimizando latencia y CPU load.

La señal digitalizada resultante con bajo ruido de cuantización y sin aliasing— constituye la base para las fases posteriores de preprocesamiento y extracción de características.

Es fundamental considerar estos parámetros al configurar y seleccionar el hardware, ya que de ellos depende la calidad de la señal de voz capturada y, por ende, la precisión y robustez del reconocimiento de comandos en entornos reales.

5.2.1.2. Preprocesamiento

La señal digital capturada pasa por cuatro operaciones secuenciales todas ejecutadas en el dominio temporal dentro del DSP TMS320C5535 que la acondicionan antes de la extracción de características:

1. **Eliminación de silencios (VAD).** Se detecta actividad de voz comparando la *energía corta* de ventanas de $W = 128$ muestras con un umbral ("SILENCETHRESHOLD" = 165). Cuando la energía de tres ventanas consecutivas cae por debajo del umbral, esas ventanas se descartan. Así se conservan únicamente los fragmentos que contienen habla efectiva, reduciendo la carga computacional posterior y evitando que el clasificador aprenda "ruido de fondo" como característica.

2. **Normalización de amplitud.** Sobre cada bloque que ha superado el VAD se calcula primero la media μ y la desviación típica σ (o, alternativamente, la potencia RMS):

$$X_{\text{RMS}} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} x^2[n]}, \tag{5.1}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} (x[n]-\mu)^2}. \tag{5.2}$$

   Donde:

   - X_{RMS} es el *valor cuadrático medio* (*Root Mean Square*) de la señal discreta $x[n]$.
   - σ es la *desviación estándar*, que cuantifica la dispersión de $x[n]$ respecto a su media.
   - $x[n]$ es el valor de la señal en el instante discreto n.
   - N es el número total de muestras consideradas.
   - μ es la *media aritmética* de la señal, definida como $\mu = \frac{1}{N}\sum_{n=0}^{N-1} x[n]$.

El código aplica *z-score* ('(Muestras[i] - media)/desviacion_estandar') aunque puede conmutarse a RMS si se desea. Con ello se homogeneiza la escala de todas las grabaciones y se elimina el desplazamiento DC.

3. **Pre-énfasis.** Para atenuar la pendiente espectral natural de la voz y realzar formantes altos se filtra la señal con un pasa-altos de primer orden

$$H(z) = 1 - \alpha z^{-1}, \quad 0{,}95 \leq \alpha \leq 0{,}98 \tag{5.3}$$

implementado en `pre_emphasis()` mediante

$$y[n] = x[n] - \alpha\, x[n-1] \tag{5.4}$$

Donde:

- $y[n]$ es la señal resultante tras aplicar la función de preénfasis.
- $x[n]$ representa la señal de entrada en el instante discreto n.
- $x[n-1]$ es el valor anterior de la señal de entrada, es decir, el valor retardado una muestra.
- α es el *coeficiente de preénfasis*, típicamente en el rango $0{,}9 \leq \alpha < 1$, que determina el grado de realce de las componentes de alta frecuencia.

En el firmware se usa $\alpha = 0{,}97$ (`Alpha = 0.97`), codificado en formato Q15 para ejecución en punto fijo.

4. **Ventaneo (Hamming).** El bloque preenfatizado se multiplica por una ventana de Hamming precalculada (`generar_hamming()`) antes de la FFT:

$$w[n] = 0{,}54 - 0{,}46\cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \leq n < N \tag{5.5}$$

La ventana suprime las discontinuidades en los bordes de cada trama, reduciendo la fuga espectral (*spectral leakage*) y mejorando la fidelidad de los MFCC.

En conjunto, estos pasos (eliminación de silencios, normalización, pre-énfasis y ventaneo) producen una señal estabilizada en nivel, libre de segmentos inactivos y con un espectro equilibrado, lista para la extracción de características MFCC. El orden exacto mostrado refleja la secuencia implementada en el código de la plataforma embebida.

5.2.1.3. Extracción de características

La extracción de características es una etapa crítica en el reconocimiento de voz, ya que transforma la señal de audio en parámetros representativos, eliminando redundancias y ruido. Entre los métodos más utilizados están los MFCC, LPC y el cepstrum lineal.

Mel-Frequency Cepstral Coefficients (MFCC): Introducidos por Davis y Mermelstein [46], los MFCC son la representación paramétrica más usada. Su cálculo implica: preénfasis, segmentación en tramas cortas con ventana de Hamming, FFT para obtener el espectro, filtrado en escala Mel, logaritmo de energía por banda, y DCT para obtener coeficientes decorrelacionados. Se suelen conservar 12–13 coeficientes más energía [47]. Son eficaces para capturar propiedades perceptivas del habla y presentan alta precisión [48], aunque son sensibles al ruido [49] y tienen mayor costo computacional (hasta 39 dimensiones incluyendo derivados). No obstante, son viables en sistemas embebidos con DSP optimizados [50].

Linear Predictive Coding (LPC): LPC modela la voz como un filtro lineal excitado por una fuente. Estima coeficientes a_i que predicen cada muestra como combinación lineal de las anteriores, usando autocorrelación y el algoritmo de Levinson-Durbin [47]. Tiene bajo costo computacional y buena compacidad, útil en entornos embebidos con recursos limitados. Sin embargo, sus coeficientes están altamente correlacionados y asume un modelo todo-polo, lo que limita su precisión ante consonantes o ruido. Se suele complementar con LPCC o LSF para mejorar su desempeño [51].

Cepstrum lineal : Se obtiene aplicando logaritmo al espectro y luego IFFT o DCT, produciendo coeficientes que representan la envolvente espectral [52]. Preserva información espectral en escala lineal, lo que puede ser útil en sonidos con energía alta en frecuencias agudas. No obstante, al no reflejar la percepción humana como MFCC, puede ser menos robusto frente al ruido. Aunque computacionalmente similar a MFCC, se prefiere este último por su mayor rendimiento en tareas de reconocimiento [51].

Análisis espectral: FFT vs. wavelet en sistemas embebidos.

En varias de las técnicas anteriores, la Transformada Rápida de Fourier (FFT) constituye la herramienta central para pasar la señal al dominio de frecuencia. La FFT es apreciada por su eficiencia algorítmica $O(N \log N)$ y por disponer de implementaciones optimizadas en hardware y DSP [47]. No obstante, la FFT opera con una resolución tiempo-frecuencia fija determinada por la longitud de ventana: en reconocimiento de voz se emplean ventanas 20 ms como compromiso, lo cual ofrece buena resolución en frecuencias bajas (captura formantes) pero menos

precisión temporal para eventos breves (p.ej., transitorios de explosivas). Una alternativa investigada es la transformada wavelet, que proporciona un análisis multirresolución. Las wavelets utilizan ventanas de distinta duración según la frecuencia: alta resolución temporal en altas frecuencias y alta resolución frecuencial en bajas frecuencias, de forma análoga a la fisiología del oído interno [53]. Esto permite representar mejor la naturaleza no estacionaria de la voz y detectar con mayor fidelidad cambios súbitos en el espectro. De hecho, se ha demostrado que características basadas en wavelets pueden ser más robustas al ruido que las cepstrales tradicionales, al captar información temporal-local que la FFT pasa por alto [54]. Pese a sus ventajas teóricas, las wavelets tienen inconvenientes prácticos en sistemas embebidos: su cálculo es más complejo de implementar eficientemente (especialmente la transformada wavelet continua o paquetes de wavelet con múltiples niveles) y suele implicar mayor carga computacional y de memoria en comparación con una FFT de tamaño equivalente [55]. Además, no existe un estándar único de wavelet para el habla se deben elegir funciones base y niveles de descomposición apropiados. En estudios comparativos, los enfoques wavelet no siempre superan significativamente a MFCC; por ejemplo, Modic et al. [53] encontraron mejoras modestas en tasa de acierto usando wavelet packets frente a MFCC estándar. Por ello, en entornos embebidos donde la simplicidad y la predictibilidad del tiempo de ejecución son críticos, la FFT permanece como la opción predominante para análisis espectral. La transformada wavelet se considera principalmente en aplicaciones especializadas o como complemento (p.ej., combinada con MFCC para mejorar robustez en ruido), más que como reemplazo directo de la FFT en los sistemas de reconocimiento de voz de recursos limitados.

5.2.1.4. Modelos de clasificación

En el panorama del reconocimiento de comandos de voz para sistemas embebidos existen varias familias de clasificadores:

- **Métodos basados en plantillas (DTW, Vector Cuántico, etc.)**: Comparan directamente secuencias de vectores de características (p. ej. MFCC) mediante alineamiento no lineal. En DTW, cada muestra de la señal de prueba se alinea con muestras de un prototipo pregrabado, construyendo una ruta de mínima distancia acumulada que permite compensar variaciones en la velocidad de pronunciación. Aunque su complejidad teórica es $O(N^2)$ en el número de tramas, la aplicación de restricciones de banda (banda Sakoe–Chiba) o poda temprana reduce drásticamente el número de comparaciones, consiguiendo tiempos casi lineales en la práctica [34, 31].
- **Hidden Markov Models (HMM)**: Modelan la señal de voz como una secuencia de estados ocultos, cada uno asociado a una distribución de emisión (típicamente gaussianas). El

entrenamiento (Baum–Welch) ajusta las probabilidades de transición entre estados y los parámetros de emisión, mientras que la decodificación (Viterbi) encuentra la secuencia de estados más probable para una observación dada. Esta estructura captura de forma explícita la dinámica temporal de la voz y mejora la tolerancia al ruido y a la variabilidad de hablantes, a costa de un mayor coste de entrenamiento e inferencia [31].

- **Gaussian Mixture Models (GMM)**: Modelan la distribución estadística de los vectores de características de cada comando como una mezcla de K gaussianas ponderadas. Durante la fase de clasificación se calcula el log-likelihood de la secuencia observada bajo cada modelo y se elige el comando con mayor verosimilitud. Al emplear covarianzas diagonales y optimizaciones en punto fijo, el coste por muestra es $O(K \times D)$, donde D es la dimensión del vector (por ejemplo, 12–14 MFCC), lo que permite ejecutar cientos de evaluaciones por segundo en un DSP embebido con latencias inferiores a 50 ms [37, 31].
- **Redes Neuronales y Aprendizaje Profundo (ANN, CNN, LSTM)**: Las ANN e, igualmente, las CNN y LSTM aprenden de forma automática representaciones jerárquicas de las características de la voz. Una CNN ligera puede extraer patrones espectrales locales, mientras que las LSTM modelan dependencias de largo plazo en la secuencia. Pueden alcanzar altas tasas de acierto y robustez al ruido, pero su despliegue en microcontroladores requiere cuantización, poda de pesos y optimizaciones de inferencia (TensorFlow Lite Micro), así como mayor memoria y ciclos de CPU [56].
- **Máquinas de Vectores de Soporte (SVM)**: Clasifican los vectores de características buscando el hiperplano que maximiza el margen entre clases. Con kernels lineales o no lineales (RBF), ofrecen muy buena precisión en vocabularios reducidos y datos balanceados. Sin embargo, el número de vectores soporte crece con la complejidad del conjunto, aumentando la memoria y el coste de evaluación, lo que limita su escalabilidad en embebidos [57].

5.2.2. Selección de algoritmos de reconocimiento de voz

Tras la investigación teórica, se procedió a seleccionar las técnicas específicas para cada etapa del sistema de reconocimiento de voz, justificando cada elección con base en las ventajas identificadas y la adecuación al contexto del proyecto. A continuación, se describen las decisiones tomadas en cuanto a la entrada de audio, preprocesamiento, extracción de características y ventaneo, destacando por qué fueron elegidas ciertas soluciones frente a las alternativas.

Antes de iniciar con el diseño del algoritmo de reconocimiento de voz, se determinó el sistema de captura de la señal y los parámetros de muestreo:

- **Transductor:** Micrófono omnidireccional condensador electret **iTalk-01**, con sensibilidad de -37 ± 3 dB y alcance efectivo de hasta 3 m [58].
- **Frecuencia de muestreo:** $f_s = 16$ kHz, que cubre componentes espectrales hasta 8 kHz y garantiza la inteligibilidad del habla.
- **Resolución:** 16 bits de cuantización.

5.2.2.1. Preprocesamiento

Dado el entorno domótico y la implementación en el DSP **TMS320C5535**, las técnicas de preprocesamiento fueron seleccionadas tomando como referencia implementaciones previamente documentadas en la literatura, destacadas por su robustez y baja complejidad computacional. Estas características las hacen especialmente adecuadas para su integración en sistemas embebidos. A continuación, se detallan y justifican las técnicas adoptadas, siguiendo el orden en que son aplicadas:

1. Eliminación de silencios. Como primer paso, se suprimen los segmentos de silencio al inicio y final de la señal de voz. Esto se logra mediante detección de actividad de voz (Voice Activity Detection, VAD) basada en la energía a corto plazo de la señal en ventanas temporales. Cada ventana se clasifica como "voz" o "silencio" comparando su energía con un umbral predefinido [59]. En el contexto de reconocimiento de comandos, omitir las partes silenciosas reduce la cantidad de datos a procesar y ahorra recursos computacionales significativos, ya que el motor de reconocimiento solo opera donde efectivamente hay habla [59]. Para una detección robusta, se aplica histéresis temporal, exigiendo múltiples ventanas consecutivas por debajo del umbral antes de marcar el final del comando, evitando así fluctuaciones erráticas [59]. El resultado es una señal recortada a los segmentos útiles de habla, lo que mejora la eficiencia y la robustez del sistema.

2. Normalización por Z-score. Tras eliminar los silencios, la señal de voz se normaliza estadísticamente usando el método Z-score. Esta técnica ajusta la amplitud de la señal para que tenga media cero y varianza unitaria, transformando cada muestra $x[n]$ como $x'[n] = (x[n] - \mu)/\sigma$, donde μ y σ son la media y la desviación estándar del tramo de voz. Esto elimina desplazamientos de la línea base y unifica la escala de energía, haciendo que las grabaciones sean comparables independientemente del volumen con que se habló [60, 61]. Desde el punto de vista de implementación, esta operación es computacionalmente simple y viable en tiempo real

en un DSP de punto fijo como el TMS320C5535. Estudios reportan que esta técnica mejora la robustez del reconocimiento al reducir discrepancias estadísticas entre muestras [60].

3. Filtrado de preénfasis (α = 0.97). Finalmente, se aplica un filtro digital FIR de primer orden con función de transferencia $H(z) = 1 - \alpha z^{-1}$ y $\alpha = 0{,}97$ [62]. Este preénfasis actúa como un pasaaltos suave que compensa la caída espectral natural de la voz, amplificando frecuencias altas donde residen detalles como formantes superiores y transiciones consonánticas [63]. Esto homogeniza el contenido espectral y mejora la relación señal-ruido en bandas relevantes para la identificación fonética. La implementación es directa y de bajo costo computacional, realizando una sola operación por muestra, fácilmente soportada por el TMS320C5535. Esta etapa mejora la calidad de las características extraídas posteriormente, aportando discriminabilidad sin penalizar el rendimiento en sistemas embebidos.

4. Ventaneo. Una vez preprocesada la señal completa, se divide en tramas breves y solapadas mediante el uso de una ventana de Hamming. Este paso es fundamental en la extracción de características en el dominio del tiempo-frecuencia, ya que permite analizar secciones cuasi-estacionarias del habla. La ventana suaviza los bordes de cada trama para evitar discontinuidades, minimizando así el fenómeno de fuga espectral al aplicar la transformada rápida de Fourier (FFT) en etapas posteriores. En la implementación se utilizó un tamaño de ventana de 32ms con solape del 50 %, balanceando resolución temporal y espectral. La ventana de Hamming, al presentar una menor atenuación lateral que la rectangular, permite preservar mejor los picos espectrales sin introducir artefactos indeseados [64]. Este proceso es clave para asegurar la fidelidad espectral de las tramas que alimentan los bancos de filtros Mel, mejorando la discriminación de patrones fonéticos.

5.2.2.2. Extracción de características

Para la etapa de extracción de características, se seleccionó el uso de Mel Frequency Cepstral Coefficients (MFCC) como representación paramétrica de la voz. Esta decisión se fundamentó en múltiples ventajas técnicas frente a alternativas como Linear Predictive Coding (LPC) o los coeficientes cepstrales tradicionales (CC), especialmente en entornos embebidos y aplicaciones en domótica.

Primero, los MFCC demostraron ofrecer mayor precisión en reconocimiento bajo condiciones ruidosas. Estudios clásicos y recientes confirman que, en presencia de ruido no estacionario o variaciones de locutor, los MFCC mantienen una tasa de acierto más alta que LPC o CC [46, 49,

53, 54]. Dado que el sistema se orienta a comandos hablados en ambientes domésticos, donde puede haber interferencias acústicas, esta robustez resultó determinante.

Segundo, los MFCC reflejan mejor la percepción auditiva humana, ya que utilizan un banco de filtros basado en la escala Mel. Esta escala simula cómo el oído humano percibe diferencias tonales: asigna más resolución en bandas de frecuencia bajas y menos en altas, lo que permite conservar la información fonéticamente relevante y descartar redundancias perceptuales [47, 46].

Tercero, los MFCC permiten una compactación eficiente del contenido espectral. Cada frame de voz se resume en 12 o 13 coeficientes, representando con alta fidelidad la envolvente espectral en un espacio de baja dimensión. Esta compacidad reduce significativamente la carga de procesamiento en las etapas de reconocimiento, lo cual es esencial para cumplir los requisitos de tiempo real en un DSP TMS320C5535 [51, 50].

Además, los MFCC son ampliamente aceptados como estándar industrial y académico en sistemas de reconocimiento automático de voz. Esta madurez tecnológica facilita su implementación, ajuste y validación, con abundante bibliografía y soporte disponible [50, 47].

En contraste, los coeficientes LPC, aunque eficientes en compresión, son más sensibles a condiciones adversas, presentan menor capacidad de discriminación en sonidos consonánticos, y asumen un modelo de todo-polo que no se ajusta bien a ciertos fonemas explosivos o fricativos [53, 54]. Por su parte, los coeficientes cepstrales lineales (CC) no incorporan la escala Mel, por lo que pierden la correspondencia perceptual con la audición humana, mostrando un desempeño inferior en tareas de reconocimiento de voz continua [49].

Finalmente, se optó por implementar 12 coeficientes MFCC por ventana de análisis y fue seleccionado como el método de extracción de características por su equilibrio entre precisión, eficiencia computacional y fidelidad perceptual, alineándose perfectamente con los objetivos del sistema embebido de reconocimiento de voz.

5.2.2.3. Modelos de clasificación

De la comparación de las distintas familias de clasificadores, se evalúan los siguientes criterios clave: facilidad de integración en un DSP embebido, coste computacional predecible y requisitos de memoria.

- **Dynamic Time Warping (DTW)**

- *Integración*: sin fase de entrenamiento; basta con almacenar plantillas de MFCC y ejecutar la rutina de alineamiento.
- *Coste computacional*: nominal $O(N^2)$, pero reducido a casi lineal mediante banda Sakoe–Chiba o poda temprana [34, 31].
- *Memoria*: dos buffers ping-pong y la plantilla de referencia; tamaño moderado incluso para comandos de hasta 1–2 s.

- **Gaussian Mixture Models (GMM)**
  - *Integración*: entrenamiento EM off-line; inferencia en punto fijo con sumas y multiplicaciones ponderadas.
  - *Coste computacional*: $O(K \times D)$ por muestra (K mezclas, D dimensión de MFCC), permitiendo decodificar cientos de tramas/s con latencias <50 ms [37, 31].
  - *Memoria*: almacenaje de pesos, medias y varianzas diagonales; escalable controlando K.

- **Hidden Markov Models (HMM)**
  - *Ventajas*: modelado temporal explícito mediante estados y transiciones.
  - *Desventajas*: entrenamiento (Baum–Welch) e inferencia (Viterbi) costosos en CPU y memoria, difícil de encajar en DSP sin aceleración adicional [31].

- **Redes Neuronales (ANN, CNN, LSTM)**
  - *Ventajas*: alta precisión y robustez al ruido.
  - *Desventajas*: requieren frameworks (TensorFlow Lite Micro), cuantización/poda y consumen más RAM y ciclos de CPU [56].

- **Máquinas de Vectores de Soporte (SVM)**
  - *Ventajas*: eficaz en vocabularios muy reducidos.
  - *Desventajas*: el número de vectores soporte crece con el vocabulario, aumentando tiempo de cómputo y memoria [57].

Por su sencillez de implementación, predictibilidad del coste computacional y bajo requerimiento de memoria, los métodos **DTW** y **GMM** son los más adecuados para el sistema embebido de reconocimiento de comandos de voz.

5.2.3. Adaptación e implementación de algoritmos

Con las técnicas seleccionadas, se procedió a la implementación práctica del sistema de reconocimiento de voz en la plataforma embebida basada en el DSP TMS320C5535. En esta sección se describe cómo se llevaron a cabo las etapas de Entrada de Audio, Preprocesamiento y Extracción de características en el hardware, incluyendo consideraciones de bajo nivel como el uso del códec de audio, configuración de DMA y optimizaciones específicas. Asimismo, se incluyen figuras ilustrativas del esquema implementado en cada etapa para una mejor comprensión del flujo del sistema.

5.2.3.1. Entrada de Audio

La implementación de la entrada de audio se realizó utilizando la placa de desarrollo basada en el TMS320C5535, la cual incorpora un DSP de punto fijo de la familia C55x de Texas Instruments y un códec de audio estéreo TLV320AIC3204 integrado. La señal de voz es captada por un micrófono electret (con su circuito de polarización y preamplificador correspondiente) y alimenta la entrada analógica del códec. El TLV320AIC3204 se configuró vía interfaz I²C para operar con las siguientes especificaciones: frecuencia de muestreo de 16 kHz, resolución de 16 bits y modo mono (utilizando el canal del micrófono en la entrada "Line-In/Mic"). Se ajustó una ganancia de entrada apropiada en el preamplificador del códec para optimizar el nivel de la señal digitalizada sin provocar saturación. El códec realiza la conversión A/D de la señal analógica en tiempo real y envía las muestras digitales al DSP mediante la interfaz serie de audio I²S. Para manejar eficientemente el flujo continuo de datos de audio, se implementó un mecanismo de transferencia DMA (Acceso Directo a Memoria). En concreto, el TMS320C5535 cuenta con controladores DMA que permiten transferir bloques de muestras desde el puerto I²S (buffer del códec) hacia la memoria interna del DSP sin intervención constante de la CPU. Se configuró el DMA en modo ping-pong con dos búferes: mientras el DMA llena un búfer con nuevas muestras entrantes, el DSP puede procesar el otro búfer en paralelo. Cuando un búfer se completa (por ejemplo, cada 256 muestras, equivalente a 16ms de audio a 16 kHz), el DMA genera una interrupción al DSP indicando que ese bloque de datos está listo para ser procesado. Este esquema asegura que no se pierdan datos y que el procesador aproveche al máximo su tiempo para cálculo de algoritmos en lugar de gestionar E/S. Cabe destacar que el sincronismo entre el códec y el DSP se mantiene mediante un reloj de bit (bit clock) de la interfaz I²S derivado del reloj maestro del DSP; en nuestro diseño, el DSP actúa como maestro en la generación del reloj de muestreo de 16 kHz para garantizar la estabilidad y sincronía en la adquisición.

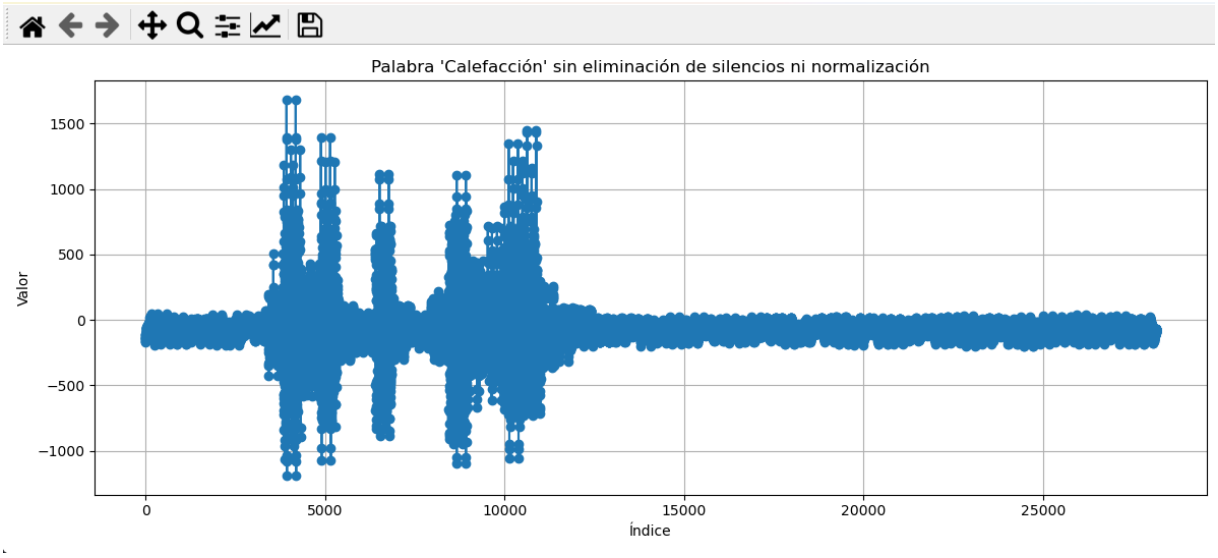

FIGURA 4: Señal de voz original capturada (onda temporal sin normalizar).

5.2.3.2. Preprocesamiento

Una vez se da la captura de audio, se desencadena el proceso de preprocesamiento digital. Este bloque se procesa en cuatro pasos secuenciales: eliminación de silencios, normalización estadística por Z-score, filtrado de preénfasis y segmentación por ventaneo. A continuación, se describen los aspectos clave de su implementación.

1. Eliminación de silencios. La eliminación de silencios se realiza calculando la energía a corto plazo de la señal sobre ventanas deslizantes. Se comparan con un umbral predefinido para clasificar cada sección como voz o silencio. Para evitar fluctuaciones erráticas, se implementa una lógica de histéresis mediante contadores de ventanas consecutivas. El resultado es un vector recortado que contiene únicamente las regiones activas del comando, reduciendo el número de operaciones posteriores. Esta funcionalidad puede consultarse en las líneas **269 a 278** del Código 7.2.

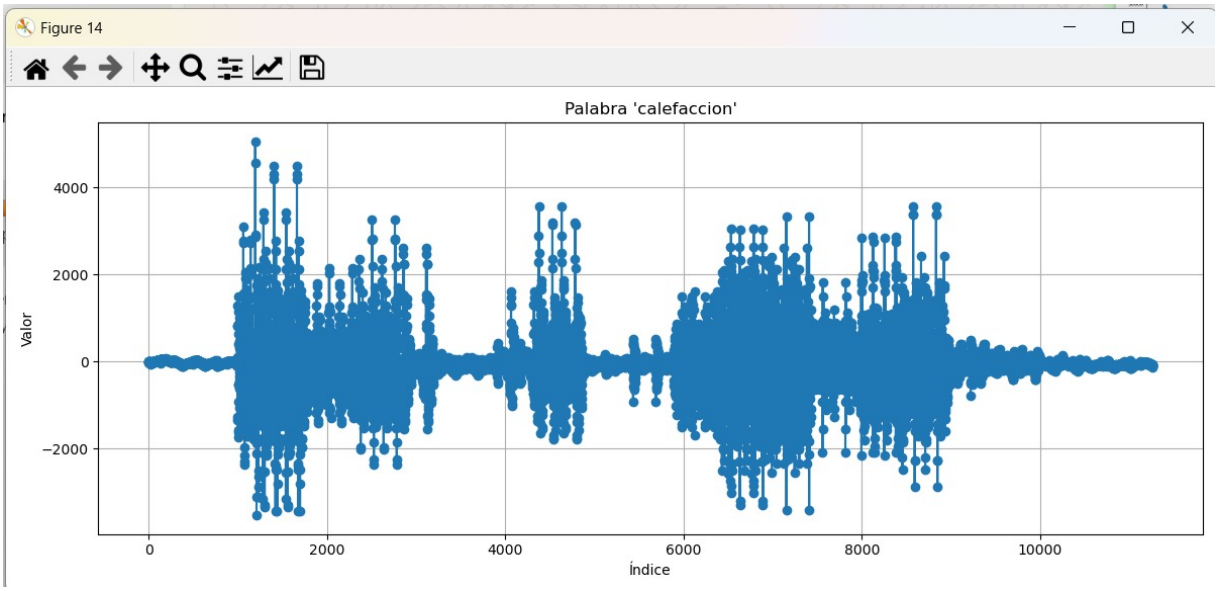

FIGURA 5: Señal original tras eliminación de silencios mediante VAD basada en energía.

2. Normalización por Z-score. El cálculo de la media μ y la desviación estándar σ se realiza usando acumuladores en punto fijo, y se transforma cada muestra según la ecuación $x'[n] = (x[n] - \mu)/\sigma$. Esta operación mejora la homogeneidad estadística entre muestras y reduce el impacto de variaciones de volumen. En el DSP, se implementó la división como multiplicación por el recíproco de σ en Q15, precalculado mediante una rutina de aproximación para eficiencia. Este bloque está implementado entre las líneas **280 a 306** del Código 7.2.

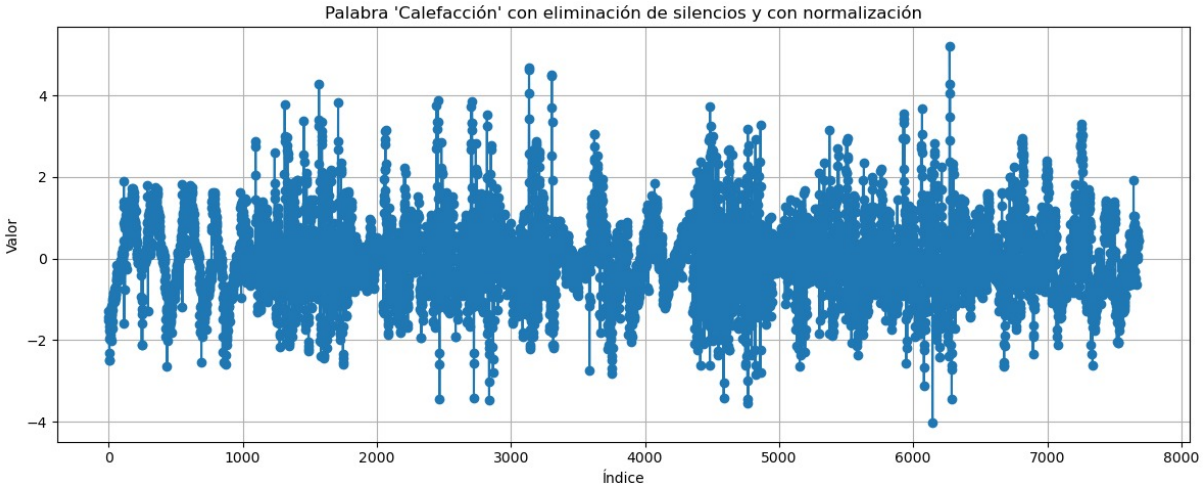

FIGURA 6: Bloque de señal tras la normalización estadística Z-score: media cero y varianza unitaria.

3. Filtrado de preénfasis. Se aplica un filtro de preénfasis con $\alpha = 0{,}97$, que se implementa como una operación de diferencias escaladas entre muestras sucesivas. La fórmula $y[n] = x[n] - \alpha \cdot x[n-1]$ se ejecuta en un ciclo `for` eficiente utilizando variables enteras de 16 bits y la constante α representada en formato Q15 ($\alpha \approx 31785$). El resultado sobrescribe la señal de entrada, aprovechando la baja carga computacional del filtro. Este proceso se realiza inmediatamente después de la normalización, en la línea **309** del Código 7.2.

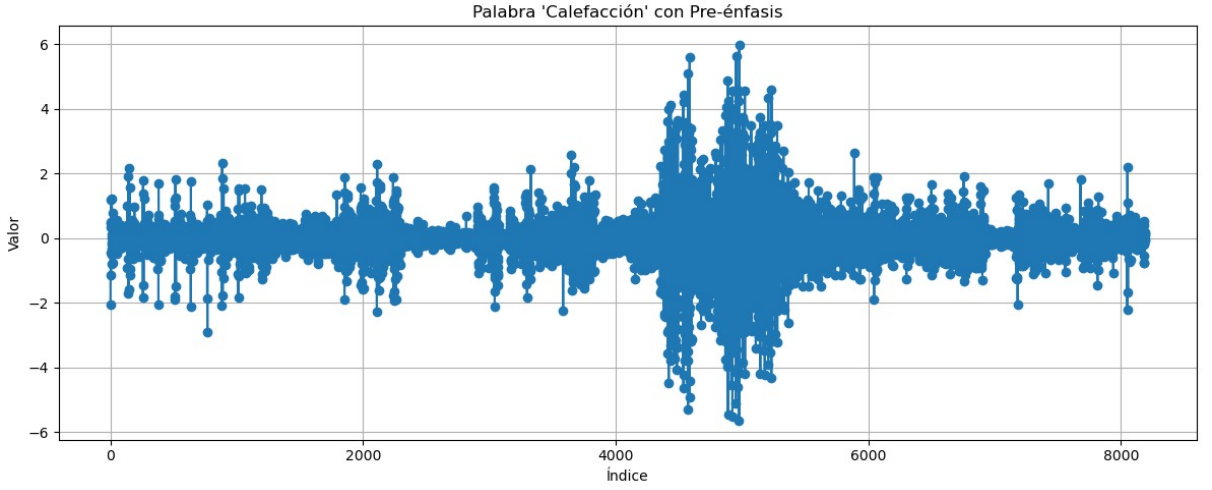

FIGURA 7: Resultado del filtrado de preénfasis: se observa realce de altas frecuencias.

4. Ventaneo. Finalmente, se segmenta el bloque resultante en tramas de 32ms con 50 % de solape, aplicando a cada trama, una ventana de Hamming almacenada como arreglo en memoria. Este ventaneo se realiza por multiplicación punto a punto, y su implementación aprovecha las instrucciones de acceso rápido del DSP a través de punteros indexados. Las tramas generadas son almacenadas en buffers circulares que luego alimentan el módulo de extracción de características. Esta lógica está distribuida en las líneas **315 a 372** del Código 7.2, dentro de los bloques 'if (Flag == 1)' y 'else', que gestionan el solapamiento y almacenamiento de las ventanas.

5.2.3.3. Extracción de características

Finalmente, se desarrolló la etapa de extracción de características MFCC adaptada al DSP TMS320C5535. La Figura 9 muestra el diagrama general del proceso implementado para convertir la señal de voz preprocesada en coeficientes MFCC en tiempo real. A continuación, se describen los pasos realizados para cada porción de audio (preenfatizada y normalizada) que se procesa:

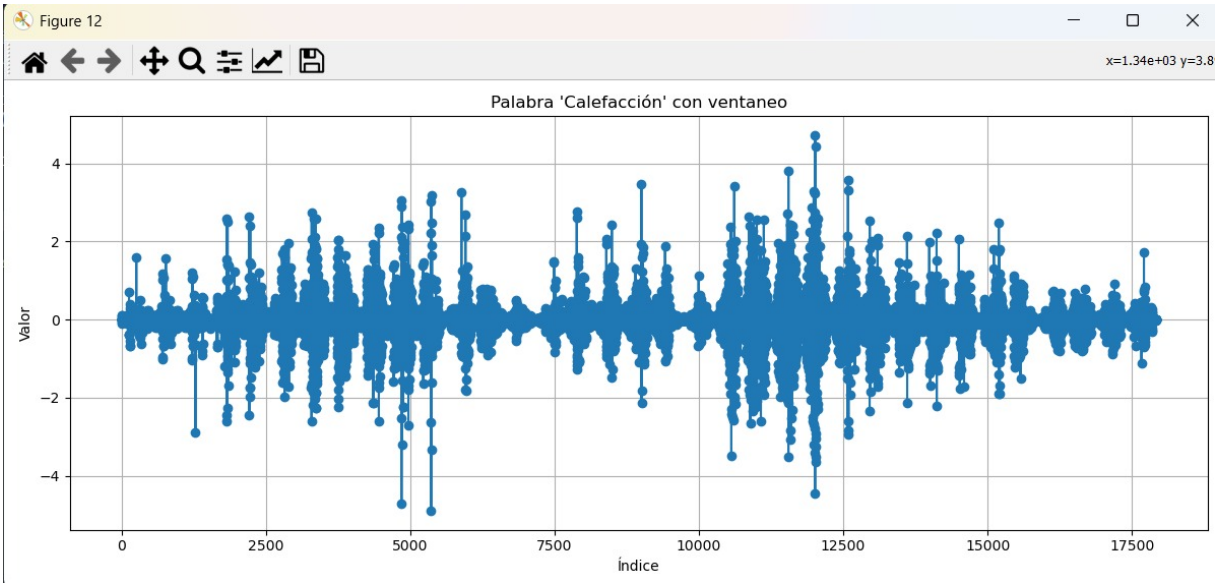

FIGURA 8: Segmentación de la señal en tramas solapadas mediante ventana de Hamming.

- **Segmentación en tramas (framing)**: El flujo de muestras de audio se segmenta en tramas cortas de duración fija para realizar un análisis de corto plazo. En nuestro caso, se utilizan tramas de $N = 256$ muestras (equivalente a 16 ms a 16 kHz) con un solapamiento del 50 % entre tramas consecutivas. Para comandos muy breves que quepan en una sola trama, se procesa únicamente esa trama; si el comando abarca varias tramas, se procesan todas y sus características se combinan después. El solapamiento mejora la suavidad temporal de las características extraídas.

- **Aplicación de ventana (windowing)**: A cada trama $z[n]$ se le aplica una ventana de Hamming de longitud N:

$$w(n) = 0{,}54 - 0{,}46\cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \leq n < N, \tag{5.6}$$

  lo cual reduce los bordes abruptos y minimiza el leakage espectral. Esta ventana se aplica multiplicando cada muestra de la trama por su correspondiente valor precalculado en una tabla.

- **Transformada de Fourier (FFT)**: Se calcula la FFT de 256 puntos para cada trama. Aprovechando el acelerador de FFT y las librerías optimizadas de TI, se obtiene el espectro complejo $X(k)$. Posteriormente, se calcula el espectro de potencia como:

$$P(k) = |X(k)|^2 = \Re(X_k)^2 + \Im(X_k)^2, \quad k = 0, \ldots, 127, \tag{5.7}$$

considerando sólo la mitad de los bins por simetría.

- **Banco de filtros Mel**: El espectro $P(k)$ se pasa por un banco de 26 filtros triangulares distribuidos en la escala Mel. Cada filtro acumula la energía en su banda como:

$$E_i = \sum_{k=0}^{127} P(k) \cdot H_i(k) \tag{5.8}$$

donde $H_i(k)$ representa la respuesta del filtro i para el bin k, precalculada y almacenada como tabla.

- **Logaritmo**: Se calcula el logaritmo natural de cada energía E_i, usando una aproximación polinomial en punto fijo para optimizar rendimiento en el DSP:

$$\log E_i, \quad i = 1, \ldots, 26. \tag{5.9}$$

- **Transformada Coseno Discreta (DCT)**: Finalmente, se aplica la DCT de tipo II sobre los $\log E_i$ para obtener los coeficientes cepstrales:

$$c_n = \sum_{i=1}^{26} \log E_i \cdot \cos\left(\frac{\pi n}{26}(i - 0{,}5)\right), \quad n = 1, \ldots, 12. \tag{5.10}$$

Opcionalmente, se incluye c_0 como indicador de energía total de la trama.

El resultado es un vector de 12 (o 13 si se incluye c_0) coeficientes que representa la envolvente espectral de cada trama. Para comandos compuestos por varias tramas, se promedian los vectores MFCC obtenidos para formar un único vector representativo del comando. Esta estrategia mantiene bajo el costo computacional y se adapta bien a comandos aislados de pronunciación estable.

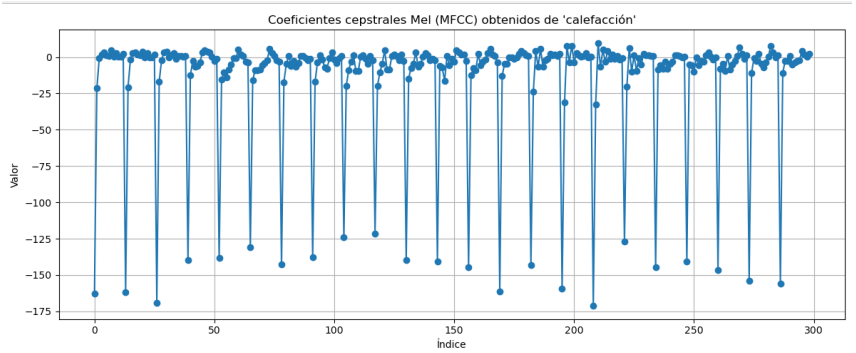

FIGURA 9: Ejemplo de coeficientes cepstrales Mel (MFCC) obtenidos de un comando de voz.

Toda esta secuencia, desde la FFT, el banco de filtros Mel, el logaritmo y la DCT, es ejecutada tras cada ventana procesada, y su implementación puede consultarse en las líneas **578 a 893** del Código 7.2.

5.2.3.4. Modelos de clasificación

Tras haber extraído los coeficientes MFCC de la señal de voz, el siguiente bloque funcional corresponde a la clasificación, cuyo objetivo es asignar cada secuencia de características a uno de cinco comandos predefinidos. Para ello, el sistema dispone de cinco modelos entrenados —uno por comando— que comparan la información espectral de la entrada con los ejemplos de referencia y calculan una métrica de similitud o probabilidad para decidir el mejor candidato.

En lo que sigue describiremos paso a paso la implementación de estos clasificadores: cómo se cargan y representan internamente las plantillas o parámetros entrenados, de qué manera se recorre la estructura de datos para evaluar la señal de prueba, cómo se calculan las puntuaciones para cada comando y finalmente cómo se selecciona la salida óptima entre las cinco alternativas disponibles.

Dynamic Time Warping (DTW) Una vez explorado el funcionamiento en el marco teórico, se procede a plantear el diagrama de flujo, el cual servirá como base para la construcción del algoritmo. Este seguirá el procedimiento planteado anteriormente.

Carga y representación de plantillas : Cada comando de voz dispone de una plantilla de referencia generada durante el entrenamiento, que contiene la secuencia característica de coeficientes MFCC. Estas plantillas se cargan en memoria al inicio del proceso de clasificación y se representan como arreglos multidimensionales que permiten un acceso durante la comparación.

Recorrido de la estructura de datos : Una vez cargadas las plantillas, la señal de prueba es evaluada frente a cada una de ellas. Este proceso sigue el flujo indicado en el diagrama, recorriendo las secuencias temporales mediante una matriz de distancias. Durante este recorrido se calculan las distancias locales entre cada ventana de la señal de prueba y cada elemento de la plantilla de referencia, preparando así la matriz para el cálculo acumulado.

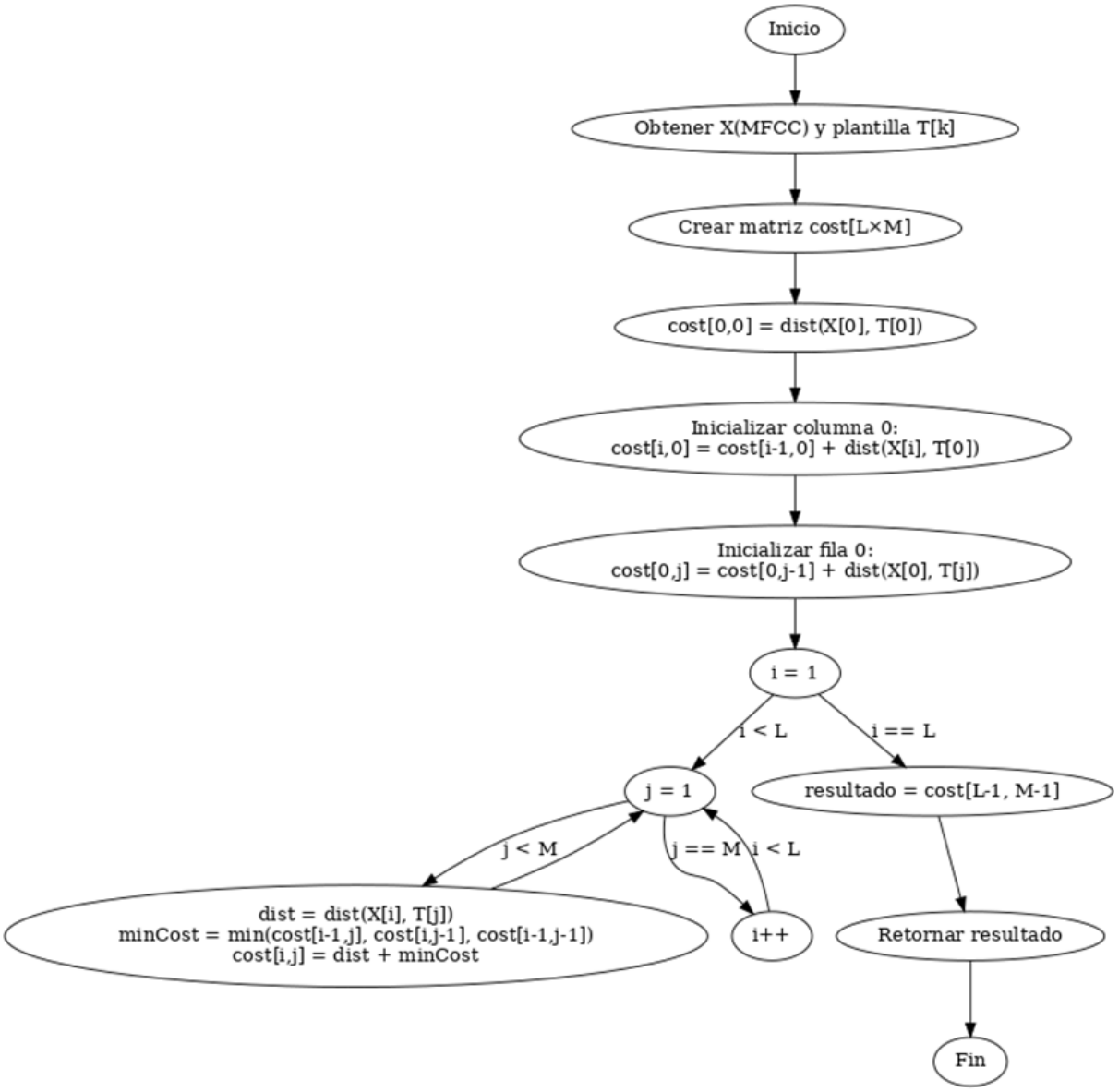

FIGURA 10: Diagrama de flujo del algoritmo DTW

Cálculo de puntuaciones : Siguiendo con el diagrama de flujo, se llena la matriz de coste acumulado aplicando la regla de minimización definida en el modelo teórico. El resultado final de este cálculo para cada plantilla corresponde a una puntuación que indica el grado de similitud entre la señal de entrada y el comando de referencia.

Selección de la salida óptima : Una vez obtenidas las puntuaciones para las cinco plantillas, se procede a seleccionar la que presenta la menor distancia acumulada, lo cual indica que es la coincidencia más probable. Esta decisión finaliza el proceso de clasificación y determina cuál comando será ejecutado por el sistema.

La implementación completa de este modelo de clasificación puede consultarse en el archivo de código fuente 7.4.

Gaussian Mixture Models (GMM)

Como se explicó en el marco teórico, el modelo de Gaussian Mixture Models (GMM) permite modelar la distribución de los vectores MFCC extraídos de los comandos de voz. A continuación, se detalla paso a paso cómo se implementó este método en el sistema:

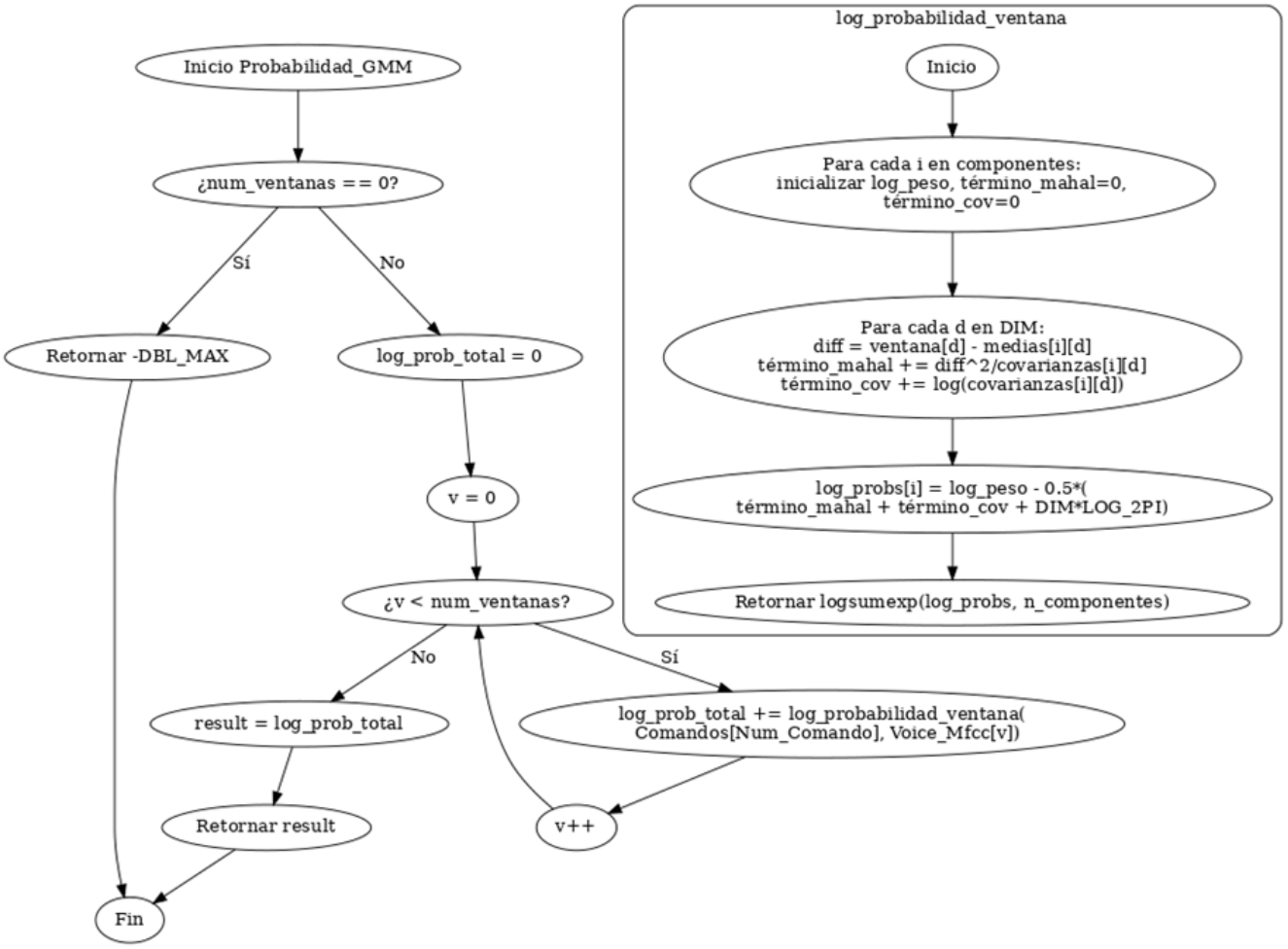

FIGURA 11: Diagrama de flujo del proceso de GMM Cálculo

Carga y representación de parámetros entrenados Para cada uno de los cinco comandos definidos, se entrenó previamente un modelo GMM utilizando la biblioteca scikit-learn, guardando

los parámetros resultantes (pesos, medias y covarianzas) en archivos de configuración. Al inicio de la ejecución, el sistema carga estos parámetros y reconstruye los objetos correspondientes para que estén listos para la fase de evaluación.

Recorrido de la estructura de datos Una vez extraídos los vectores MFCC de la señal de prueba, el sistema recorre cada comando (y su respectivo modelo GMM) y evalúa secuencialmente todas las ventanas de características que componen el comando. Este recorrido se realiza de manera eficiente, procesando lote a lote (batch) las ventanas para reducir el tiempo de cómputo.

Cálculo de puntuaciones para cada comando En cada modelo GMM, se calcula la probabilidad logarítmica de cada ventana de características mediante la combinación de las probabilidades individuales de los componentes, usando la técnica numéricamente estable de *logsumexp*. Posteriormente, las probabilidades de todas las ventanas del comando se suman para obtener la probabilidad logarítmica total de la secuencia bajo ese modelo. Este valor se almacena como puntuación final para el comando.

Selección de la salida Tras evaluar los cinco modelos correspondientes a los comandos, se comparan las puntuaciones obtenidas y se selecciona el comando cuya probabilidad logarítmica total es máxima. Este comando es finalmente reportado como el resultado del reconocimiento. La implementación completa de este modelo de clasificación puede consultarse en el archivo de código fuente 7.3.

Clasificación basada en distancia Euclidiana con CMVN y desplazamiento

A diferencia de DTW o GMM, este modelo emplea directamente la distancia Euclidiana entre secuencias MFCC normalizadas mediante Cepstral Mean and Variance Normalization (CMVN), y explora pequeños desplazamientos temporales para mejorar la alineación.

- **CMVN por coeficiente:** Para cada ventana y cada coeficiente MFCC se calcula la media y la desviación típica tanto de la señal de voz como de la plantilla. Luego, cada coeficiente se normaliza restando su media y dividiendo por su desviación (más un pequeño valor constante para evitar división por cero).

- **Desplazamiento temporal (S):** Se permite un corrimiento de $-S$ a $+S$ ventanas entre la señal de prueba y la plantilla, con el fin de compensar ligeras variaciones en la velocidad de pronunciación.
- **Distancia Euclidiana promedio:** Para cada desplazamiento se calcula el promedio de los cuadrados de las diferencias normalizadas; la raíz de este valor es la puntuación de distancia para ese shift. Finalmente se elige el desplazamiento que minimiza dicha distancia.

A continuación, se describe el flujo de ejecución, apoyado en el diagrama de flujo de la Figura 12:

1. Carga y representación de plantillas Al inicio, se cargan las plantillas MFCC entrenadas para los cinco comandos y se almacenan como arreglos en memoria, junto con los parámetros necesarios para CMVN (medias y desviaciones precomputadas o calculables en tiempo real).

2. Normalización CMVN Para cada comando se recorre la matriz MFCC de la señal de prueba y la de la plantilla, calculando dinámicamente (o recuperando) las medias y desviaciones de cada coeficiente, y aplicando la normalización:

$$\hat{x}_{w,c} = \frac{x_{w,c} - \mu_c}{\sigma_c + \varepsilon} \quad , \quad \hat{t}_{w,c} = \frac{t_{w,c} - \mu_c}{\sigma_c + \varepsilon} \tag{5.11}$$

donde ε es un valor pequeño para estabilidad numérica.

3. Recorrido con desfase temporal Se itera el parámetro de desfase desde $-S$ hasta $+S$. Para cada desfase:

- Se alinean ventanas de prueba y plantilla con el corrimiento especificado.
- Se suman los cuadrados de la diferencia $(\hat{x}_{iv,c} - \hat{t}_{w,c})^2$ sobre todos los coeficientes y ventanas coincidentes.
- Se divide por el número de comparaciones para obtener la distancia promedio.

4. Cálculo de puntuaciones La puntuación de cada comando es la raíz cuadrada de la mínima distancia promedio obtenida entre todos los desfases temporales:

$$d_{\text{cmd}} = \min_{-S \le s \le S} \sqrt{\frac{1}{N} \sum_{(w,c)} \left(\hat{x}_{w+s,c} - \hat{t}_{w,c}\right)^2} \tag{5.12}$$

5. Selección de la salida Tras evaluar los cinco comandos, se selecciona aquel cuyo d_{cmd} sea menor, indicando la mayor similitud cepstral normalizada.

La implementación completa de este modelo de clasificación puede consultarse en el archivo de código fuente 7.5.

Diagrama de flujo La Figura 12 resume gráficamente estas etapas.

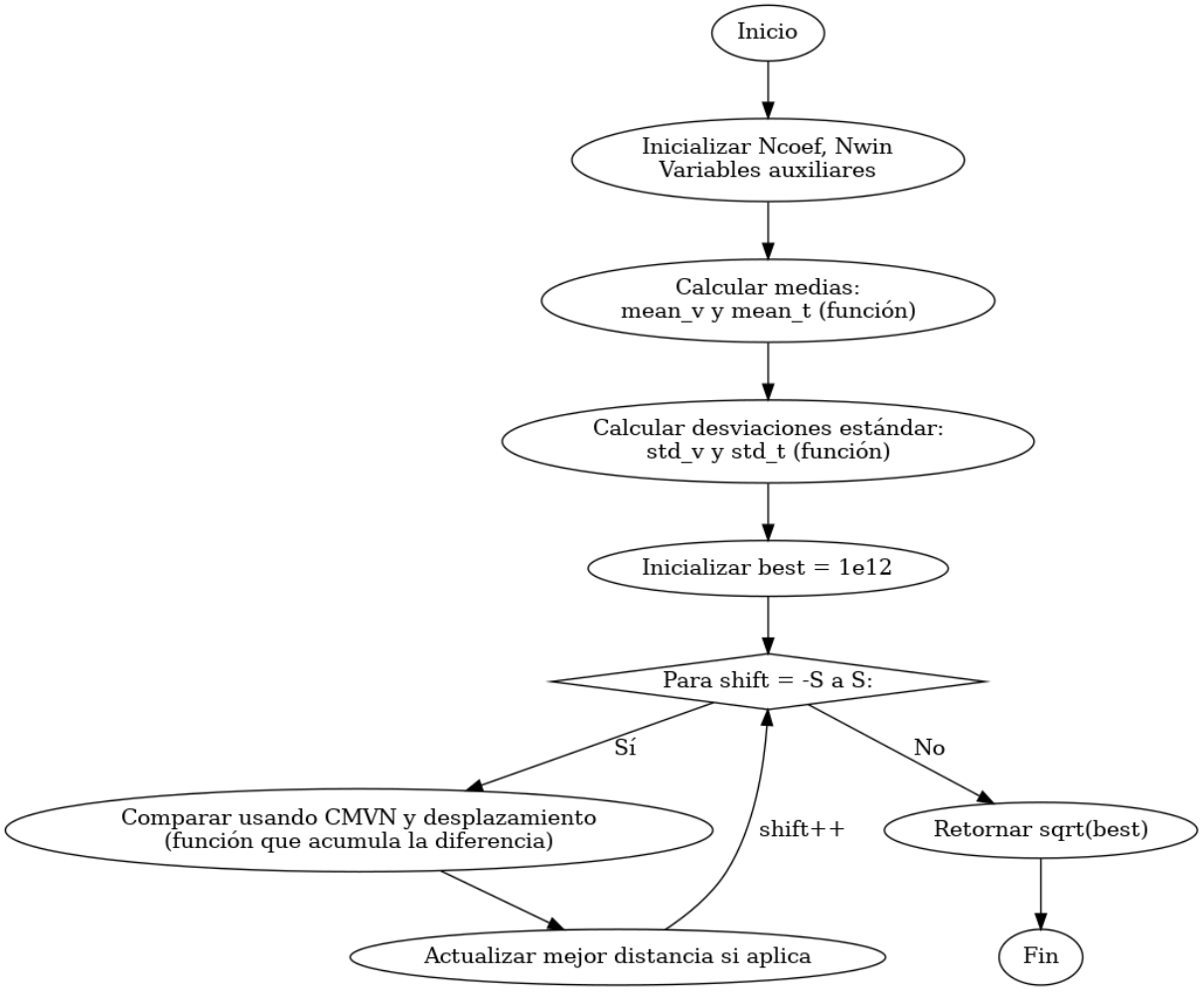

FIGURA 12: Diagrama de flujo de la clasificación por distancia Euclidiana con CMVN y desplazamiento

5.2.4. Evaluación de rendimiento de algoritmos

En esta fase se evaluó la capacidad de tres modelos de clasificación (GMM, DTW y Distancia Euclidiana con CMVN) para reconocer comandos de voz en ambientes silenciosos (ver Figuras 27, 28 y 26). Los resultados se analizaron mediante matrices de confusión y métricas de

precisión, cuyos detalles se presentan en el Anexo A.

Resultados en ambiente silencioso

Todos los modelos alcanzaron una precisión cercana al $100\,\%$, con errores menores a $4\,\%$ en cada caso. Las matrices de confusión (ver Figuras 27, 28 y 26) muestran que los errores fueron aleatorios y no sistemáticos, indicando un buen desempeño bajo condiciones ideales.

Resultados en ambiente ruidoso (50 dB)

La introducción de ruido afectó significativamente el rendimiento de los algoritmos, revelando diferencias en su robustez:

5.2.4.1. Modelo GMM

- **Distribución de errores (Tasa de acierto:** $86\,\%$**)**: Los errores se dispersaron uniformemente entre todos los comandos (ver Figura 13). Ningún comando tuvo menos de 7 aciertos sobre 10 ensayos, lo que sugiere una tolerancia generalizada al ruido. Su estructura probabilística parece mitigar el impacto del ruido, evitando sesgos hacia errores específicos.

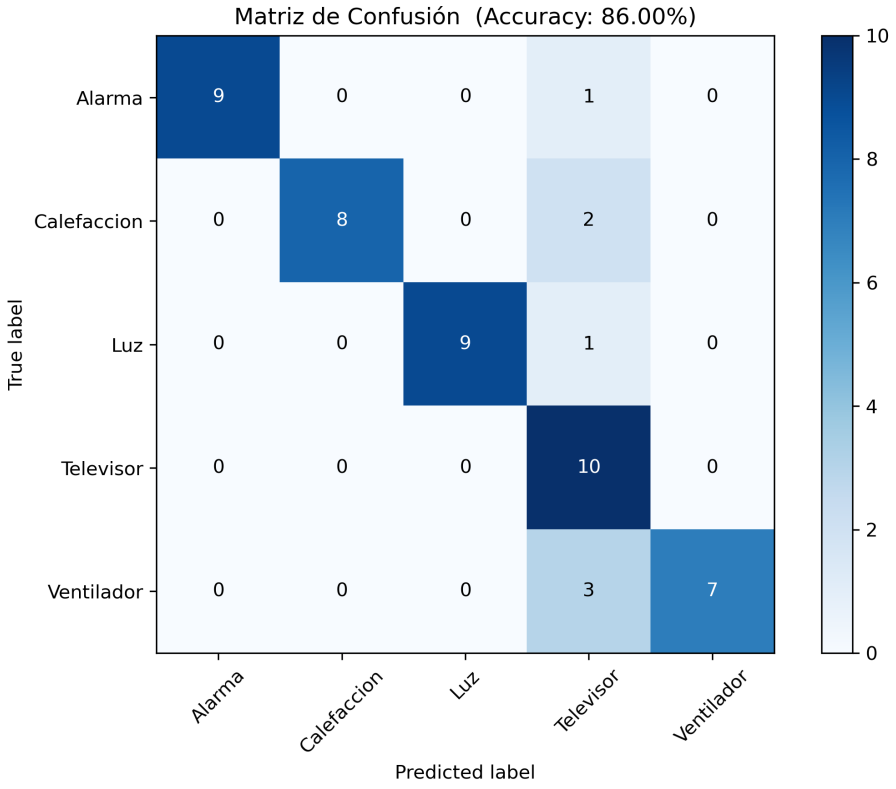

FIGURA 13: Matriz de confusión del modelo GMM en ambiente ruidoso (50 dB).

5.2.4.2. Modelo DTW

-**Distribución de errores (Tasa de acierto:** 84 %): La matriz de confusión (ver Figura 14) muestra que el comando **"calefacción"** fue el más vulnerable: - Solo 3 de 10 ensayos se clasificaron correctamente. - Errores concentrados en "televisor" (5 casos) y "alarma" (1 caso). El DTW, basado en alineación temporal, podría ser sensible a variaciones en la forma de onda causadas por el ruido, especialmente en comandos con patrones complejos como "calefacción".

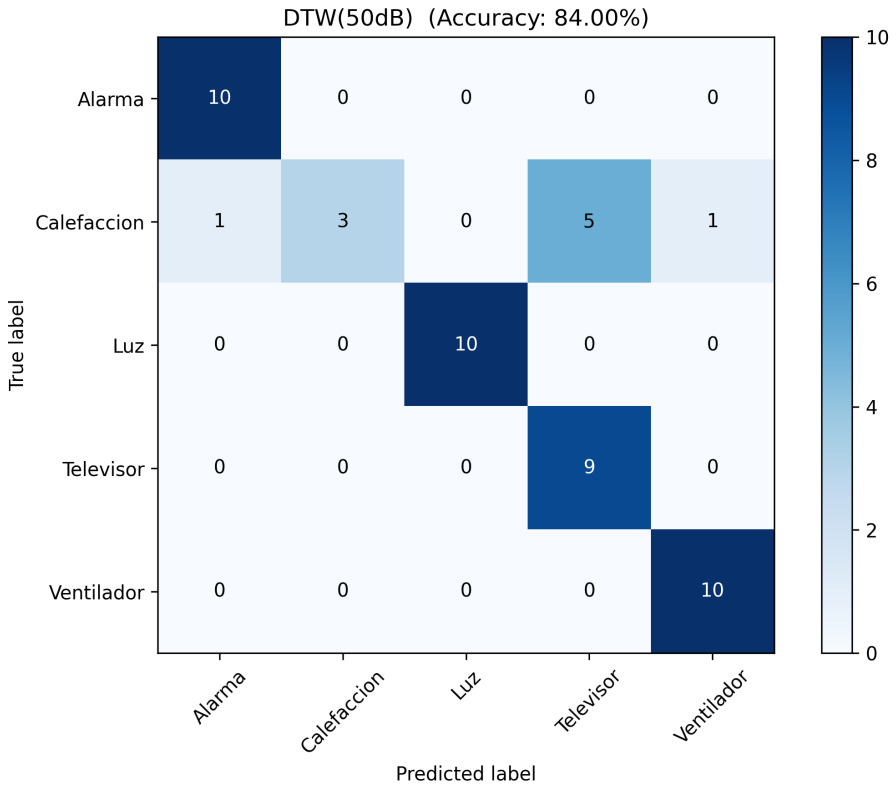

FIGURA 14: Matriz de confusión del modelo DTW en ambiente ruidoso (50 dB).

5.2.4.3. Distancia Euclidiana con CMVN

- **Distribución de errores (Tasa de acierto:** 78 %**)**: La matriz de confusión (ver Figura 15) evidencia una caída más pronunciada en la precisión global:

- Comandos como luz y calefacción sufrieron múltiples errores:
- Luz: 6 aciertos, 1 error en alarma, 1 en calefacción, 1 en televisor y 1 en ventilador.
- Alarma también mostró fragilidad, con 1 error en "ventilador".
- Calefacción: 4 aciertos, 3 errores en alarma, 2 en televisor y 1 en ventilador.

La normalización CMVN no compensó completamente el ruido, especialmente en comandos cortos o con características acústicas similares a otros (ej.: luz vs. alarma).

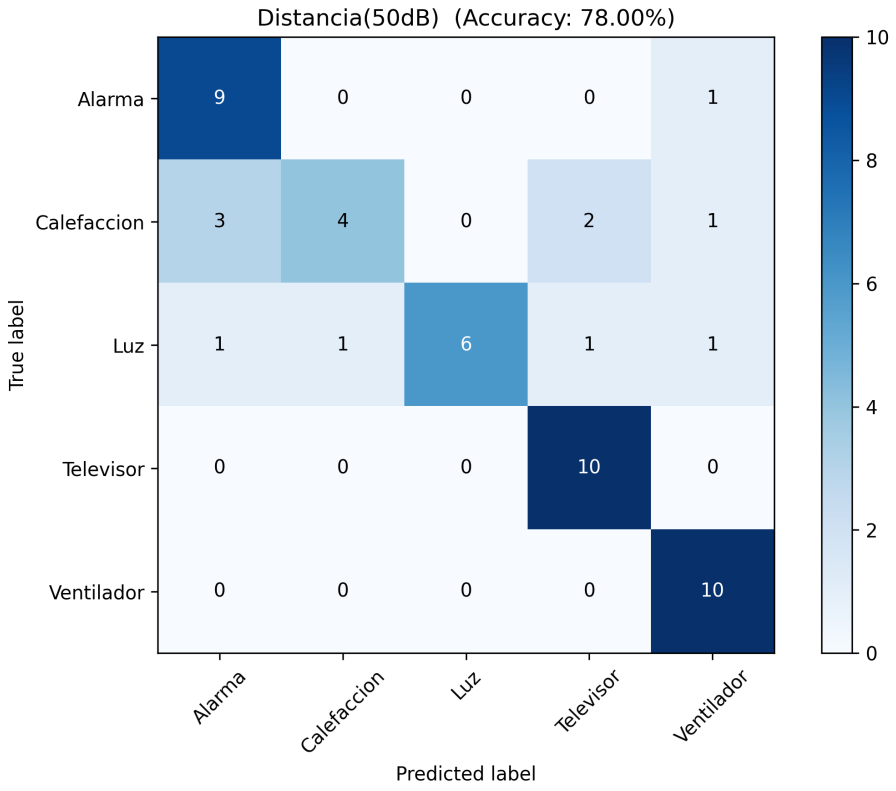

FIGURA 15: Matriz de confusión de Distancia Euclidiana con CMVN en ambiente ruidoso (50 dB).

"

Comparación cualitativa

TABLA 10: Comparación cualitativa de la robustez y comportamiento de los modelos

Métrica	**GMM**	**DTW**	**Distancia Euclidiana**
Robustez al ruido	Alta	Media	Baja
Errores sistémicos	No	Sí (“calefacción”)	Sí (“luz”, “calefacción”)
Sensibilidad a variabilidad	Moderada	Alta	Muy alta

Conclusiones

- **GMM** demostró la mejor estabilidad ante ruido, distribuyendo errores de manera homogénea.

- **DTW** requiere ajustes para comandos con patrones temporales complejos (ej.: “calefacción”).

- **Distancia Euclidiana con CMVN** es el menos robusto, especialmente en entornos ruidosos.

Tiempos de ejecución

Para completar el análisis, medimos el número de ciclos de máquina promedio (± desviación estándar) que tarda cada algoritmo en procesar un comando, tanto en ambiente silencioso (35 dB) como en ruidoso (50 dB). Los resultados se ilustran en las Figuras X.1 y X.2.

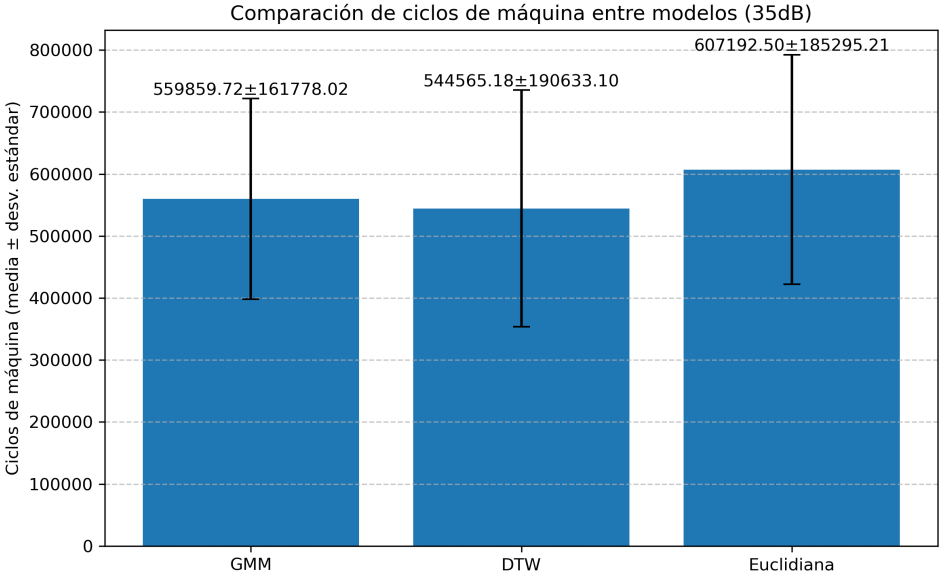

FIGURA 16: Comparación de ciclos de máquina entre modelos en ambiente silencioso (35 dB).

Análisis (35 dB):

En esta condición, el *GMM* completa cada comando en aproximadamente 559.9k ciclos con una desviación de 161.8k. El *DTW* tarda 544.6k ± 190.6k, mientras que la *Euclidiana* usa 607.2k ± 185.3k. Aunque DTW tiene una media ligeramente inferior a GMM, su mayor desviación indica picos de latencia impredecibles; Euclidiana resulta la más lenta.

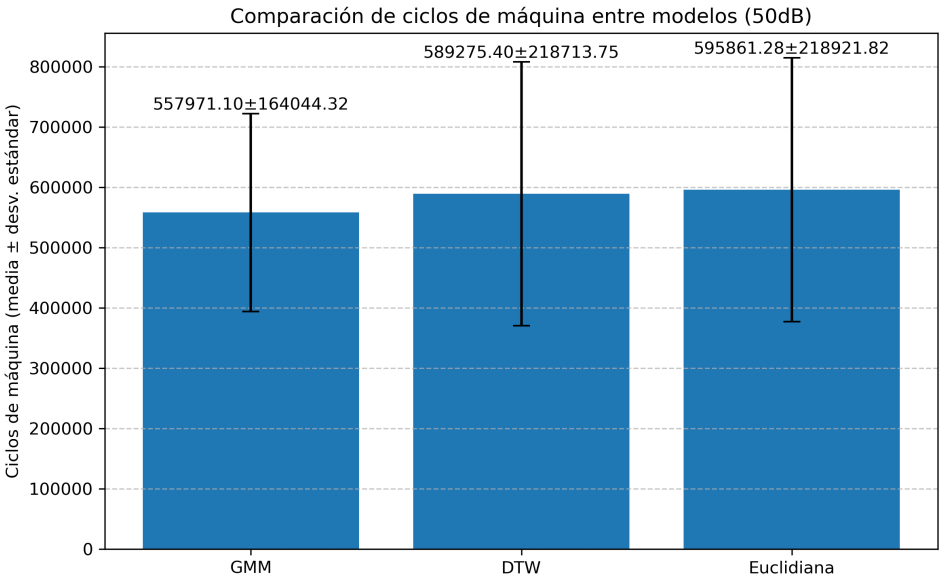

FIGURA 17: Comparación de ciclos de máquina entre modelos en ambiente ruidoso (50 dB).

Análisis (50 dB):
Bajo ruido, GMM mantiene su rendimiento (558.0k ± 164.0k). DTW y Euclidiana suben a 589.3k ± 218.7k y 595.9k ± 218.9k, respectivamente. El ruido introduce más sobrecarga y variabilidad en estos métodos, mientras que GMM sigue siendo el más consistente.

- *GMM* conserva el menor coste computacional en ambos escenarios.
- *DTW* y *Euclidiana* requieren entre un 5 % y un 10 % más de ciclos que GMM, y además presentan mayor variabilidad en tiempo de ejecución.
- Además, en el caso de *DTW* la longitud de la matriz de distancias crece con el número de muestras del comando: comandos más largos o con más ruido (lo que implica más frames útiles) aumentan notablemente el tiempo de procesamiento al elevarse la complejidad de la alineación temporal.

Selección del modelo

Tomando en cuenta tanto la precisión como el tiempo de ejecución:

- **GMM** alcanza un 86 % de precisión en ruido, distribuye sus errores de forma homogénea y requiere el menor tiempo de cómputo.

- **DTW** iguala la precisión de GMM, pero incurre en una penalización de hasta un 7 % más de ciclos y muestra alta variabilidad temporal.
- **Distancia Euclidiana con CMVN** cae al 78 % de precisión y además es el más lento en promedio.

Teniendo en cuenta todos los resultados **modelo GMM** ofrece el *mejor compromiso* entre robustez al ruido, uniformidad de errores y eficiencia computacional, por lo que se recomienda como la solución óptima para el reconocimiento de comandos en entornos de voz con niveles de ruido hasta 50 dB.

5.3. Validación en entorno domótico

En esta tercera fase del proyecto se procede a la integración completa del sistema de reconocimiento de voz en un escenario real de automatización. El objetivo es comprobar no solo la precisión y rapidez del algoritmo en condiciones controladas, sino su capacidad para interactuar de forma fiable con dispositivos domóticos . Para ello, en primer lugar se seleccionan los equipos que resulten compatibles con el microcontrolador y el protocolo de comunicación empleado. A continuación, se implementa el enlace físico y lógico entre el sistema embebido y los dispositivos, de modo que cada comando de voz desencadene la acción correspondiente en el entorno doméstico. Sobre esta base, se definen un conjunto de métricas de evaluación —incluyendo tasa de aciertos, latencia de respuesta y robustez frente a interferencias— que permitirán cuantificar el rendimiento del prototipo. Finalmente, se ejecutan pruebas prácticas en distintos escenarios de uso diario, midiendo y analizando cada métrica para validar la operatividad y fiabilidad del sistema en un entorno real de domótica.

5.3.1. Selección de los dispositivos de domótica

El objetivo de esta sección es que el sistema de reconocimiento de voz sea capaz de controlar, mediante una señal binaria de encendido/apagado (ON/OFF), los cinco dispositivos vinculados a los comandos definidos: *Luz*, *Calefacción*, *Ventilador*, *fuente* y *Televisor*. El sistema iniciará con todos los dispositivos en estado *apagado*, y al detectar alguno de los comandos, ejecutará una acción de *toggle* sobre el estado actual del dispositivo correspondiente (encendiendo si estaba apagado o apagando si estaba encendido).

Para implementar este esquema de control, se optó por utilizar módulos de relé mecánico estándar, los cuales permiten conmutar la alimentación eléctrica de cada dispositivo de manera sencilla y segura. Los dispositivos seleccionados para la validación son los siguientes:

- **Luz:** Bombilla LED de 12 W.
- **Calefacción:** Hornilla eléctrica de resistencia de 26Ω alimentada a 120V, con una potencia aproximada de 554Wh.
- **Ventilador:** Motor DC de baja potencia.
- **Fuente:** Cargador de teléfono.
- **Televisor:** Televisor de laboratorio.

Cada uno de estos dispositivos será conectado mediante los módulos de relé, los cuales son capaces de manejar tensiones y corrientes típicas de cargas domésticas de baja potencia. La elección de este método garantiza que el sistema pueda ejecutar acciones reales sobre los aparatos controlados, replicando un entorno domótico básico y facilitando la validación experimental del desempeño del prototipo.

A continuación, se presenta una fotografía ilustrativa del montaje realizado para la integración de los dispositivos domóticos con el sistema embebido:

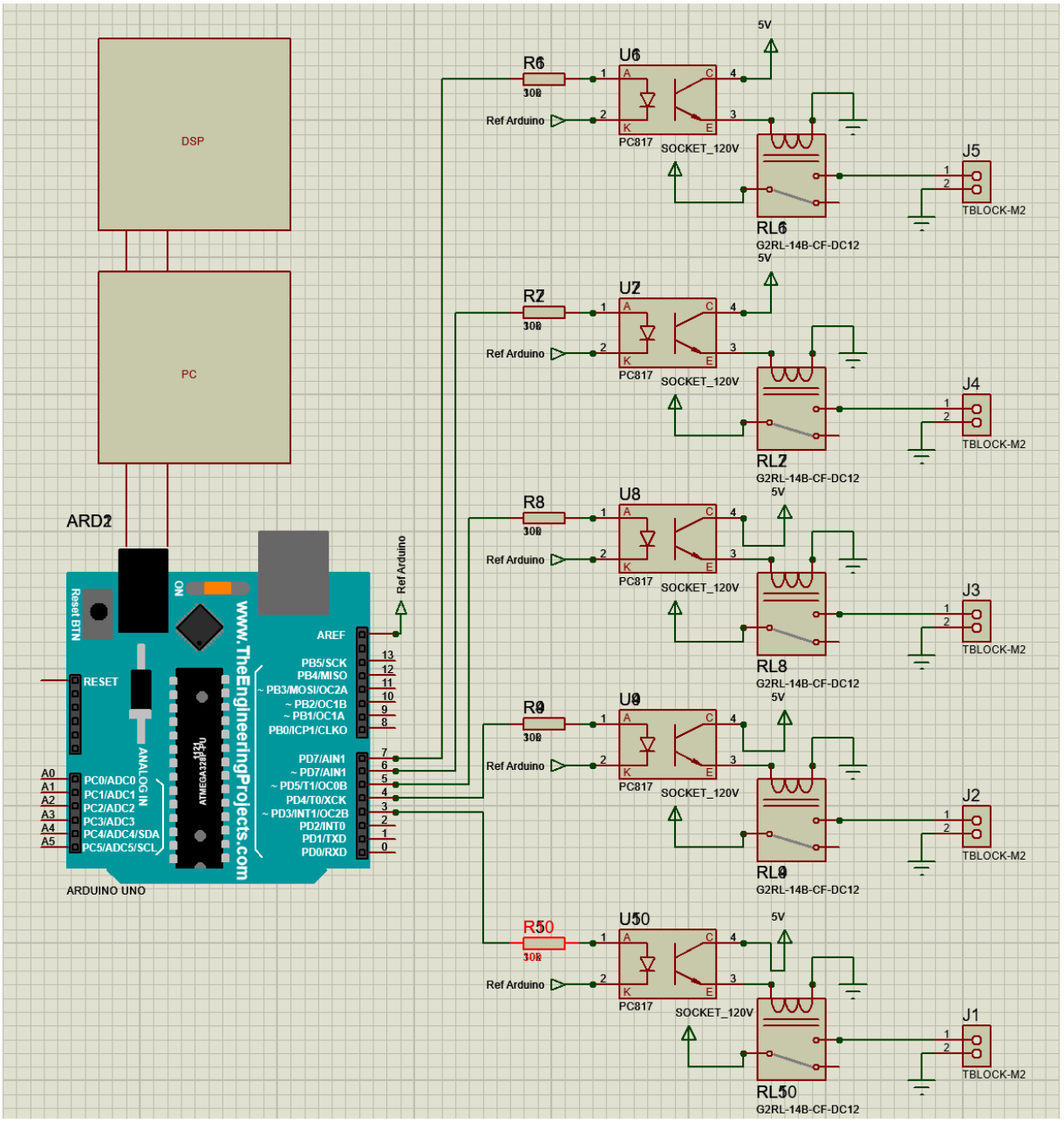

FIGURA 18: Montaje experimental de los dispositivos domóticos conectados mediante módulos de relé.

Para establecer la conexión física y lógica entre la DSP de reconocimiento de voz y los módulos de relé que controlan los cinco actuadores domóticos, se evaluaron dos alternativas:

1. **Acceso directo a puertos de expansión de la DSP.** La tarjeta EZDSP5535 incluye un conector de expansión con señales GPIO que, en principio, permitirían disparar relés sin intermediarios. Sin embargo, este conector requiere un adaptador hembra específico (ya descatalogado) para soldar un cable plano y acceder a los pines. Dada la antigüedad de

la placa y la imposibilidad de conseguir localmente el adaptador, esta opción fue descartada.

2. **Comunicación UART hacia un microcontrolador intermedio.** Se optó por aprovechar el puerto serie UART integrado en la DSP como canal de control hacia un Arduino. En esta arquitectura, el PC actúa como puente entre la DSP y el Arduino, permitiendo redirigir los datos de la UART de la DSP hacia el microcontrolador a través de un monitor serial o una aplicación intermedia. El flujo de datos es:

   - La DSP captura el comando de voz y lo clasifica internamente.
   - Inmediatamente después de la clasificación, la DSP envía un único carácter por UART (por ejemplo, 'L' para "Luz", 'C' para "Calefacción", etc.).
   - El PC recibe este carácter por su puerto serie y lo reenvía directamente al Arduino, que está conectado a otro puerto serial del PC.
   - El Arduino, constantemente escuchando en su puerto serie, recibe ese carácter, lo interpreta y activa o desactiva el relé correspondiente (toggle ON/OFF).

   Este esquema desacopla completamente la lógica de reconocimiento —que permanece sobre la DSP— de la gestión de hardware de potencia —que corre sobre el Arduino y sus relés azules clásicos—. Además, la comunicación serial es robusta, ampliamente documentada y permite, en un futuro, ampliar el número de comandos o añadir bidireccionalidad (por ejemplo, confirmaciones o diagnósticos) sin modificar el hardware base.

De esta forma, el sistema embebido mantiene su eficiencia en procesamiento de voz y queda preparado para interactuar de forma segura y escalable con cualquier dispositivo domótico conmutado por relé.

5.3.2. Definición de métricas de evaluación

Para la fase de validación en entornos domóticos, se seleccionaron dos métricas clave que permiten evaluar el rendimiento del sistema en condiciones reales de uso:

- **Precisión (tasa de aciertos):** Se mantiene la métrica utilizada en las fases anteriores, que consiste en la proporción de comandos correctamente reconocidos sobre el total de comandos emitidos. Esta métrica permite comparar la robustez del sistema frente a condiciones acústicas reales y variabilidad en la pronunciación.

- **Tiempo de respuesta total:** Se mide el tiempo transcurrido desde la emisión del comando de voz hasta la activación efectiva del dispositivo domótico correspondiente (por ejemplo, encendido de una luz para el comando "luz"). Esta métrica refleja directamente la eficiencia del sistema en una aplicación embebida práctica y es clave para evaluar la experiencia de usuario.

Ambas métricas permiten contrastar el desempeño teórico medido en laboratorio con el comportamiento real del prototipo en su entorno de aplicación final.

5.3.3. Ejecución de pruebas

Las pruebas de validación final se realizaron en dos entornos domóticos reales con salidas físicas conectadas al sistema mediante módulos de relé. Durante estas pruebas, se midió el tiempo transcurrido desde la emisión verbal del comando hasta la activación o desactivación efectiva del electrodoméstico correspondiente.

Entorno ETM-2: Pruebas en un ambiente con condiciones acústicas complejas debido a la presencia de eco, con niveles de ruido ambiente controlados. Se probaron cinco comandos de voz asociados a los dispositivos: luz, calefacción, ventilador, fuente y televisor.

TABLA 11: Tiempos de respuesta por comando en entorno ETM-2 (milisegundos)

Comando	**Tiempo medio de respuesta de encendido**	**Tiempo medio de apagado**
Luz	6120	7800
Calefacción	11100	10600
Ventilador	10250	11800
Fuente	7470	8230
Televisor	10200	10900

Entorno ETM-7: Pruebas en un ambiente con ruido de fondo generado por personas conversando y el sonido de la alarma, los cuales modificaron significativamente el entorno domótico. Además, se sustituyó el televisor por una alarma como electrodoméstico para cambiar el entorno domótico.

TABLA 12: Tiempos de respuesta por comando en entorno ETM-7 (milisegundos)

Comando	Tiempo medio de respuesta de encendido	Tiempo medio de apagado
Luz	5850	7600
Calefacción	13200	11300
Ventilador	11050	12400
Fuente	9300	9700
Alarma	11500	-

Además, se llevó a cabo un análisis cualitativo de la precisión del reconocimiento, observándose que las únicas fallas ocurrieron cuando la alarma estaba encendida. En dichas situaciones, el ruido generado por la alarma impedía capturar el audio de la persona, afectando la correcta interpretación de los comandos de voz.

5.4. Creación de guía de laboratorio

Esta guía está dirigida específicamente a los estudiantes del espacio académico de Procesamiento Digital de Señales, ya que sus contenidos y actividades prácticas han sido diseñados para alinearse con los temas abordados en dicho curso. En el marco de los objetivos educativos del proyecto, se elaboró una guía de laboratorio orientada a introducir a los estudiantes en el procesamiento digital de señales de audio, utilizando la plataforma DSP TMS320C5535 como herramienta principal. Esta experiencia fue complementada con una clase expositiva, en la que se presentó el proyecto de grado, se explicó el funcionamiento del sistema de reconocimiento de comandos de voz, y se revisaron los conceptos fundamentales del procesamiento de señales de audio en sistemas embebidos.

Durante la clase se detalló el flujo de adquisición, preprocesamiento, extracción de características y clasificación de señales de voz, y se mostró en funcionamiento el sistema embebido desarrollado. Se hizo especial énfasis en el impacto del diseño eficiente de algoritmos sobre plataformas con recursos limitados y la relevancia de esta área en aplicaciones actuales como la domótica.

Se elaboró una guía de laboratorio titulada *Inicialización de la DSP y Procesamiento de Señales de Audio* (ver Guía de Laboratorio: Inicialización de la DSP y Procesamiento de Señales de Audio), enfocada en la configuración inicial del TMS320C5535, la normalización por Z-score y la eliminación de silencios. La guía está estructurada en tres pasos principales inicialización, normalización y detección de voz activa y tiene como objetivos facilitar el uso básico de la plataforma DSP, implementar etapas esenciales del procesamiento de voz y motivar a los estudiantes a

explorar aplicaciones prácticas mediante herramientas accesibles como GitHub y tutoriales en video.

Finalizada la clase, se aplicó una encuesta diagnóstica a los estudiantes asistentes con el fin de evaluar el interés despertado por el procesamiento de señales de audio. Esta encuesta buscaba validar la hipótesis inicial del proyecto: que esta temática, poco explorada hasta ahora en el contexto institucional, podría resultar atractiva y formativa para los estudiantes. Los resultados fueron ampliamente positivos, confirmando la disposición de los estudiantes para aprender sobre procesamiento de voz, incluso cuando se parte desde cero en conocimientos de señales acústicas.

Teniendo en cuenta los resultados de la encuesta se puede afirmar que la clase y la guía de laboratorio lograron cumplir con el objetivo específico de fomentar el interés por el procesamiento digital de señales de audio entre los estudiantes de ingeniería electrónica. Esta intervención no solo generó una experiencia formativa enriquecedora, sino que también demostró el potencial de esta línea temática para consolidarse como un eje innovador dentro de la formación académica e investigativa de la Universidad Santo Tomás.

Evaluación del impacto educativo

A continuación, se presenta la evaluación del impacto educativo de la clase demostrativa y de la guía de laboratorio, realizada a través de una encuesta diagnóstica dirigida a los estudiantes participantes. Esta evaluación tuvo como propósito medir el nivel de interés, comprensión y percepción general del contenido abordado. En total, participaron en la encuesta 8 de las 10 personas presentes en la actividad.

Seguidamente, se detallan las preguntas aplicadas junto con los resultados obtenidos para cada una:

1. ¿Cómo calificaría la claridad de los temas presentados?

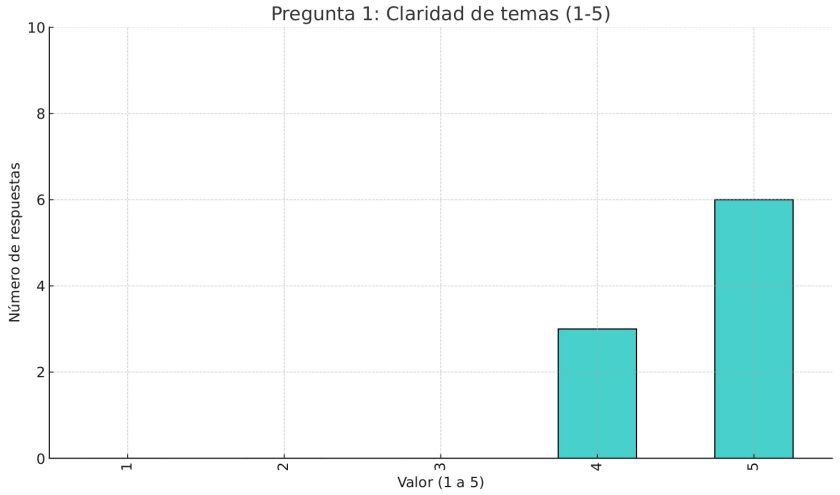

FIGURA 19: Claridad de los temas presentados.

2. ¿Considera que la clase le brindó un acercamiento al procesamiento digital de señales de audio?

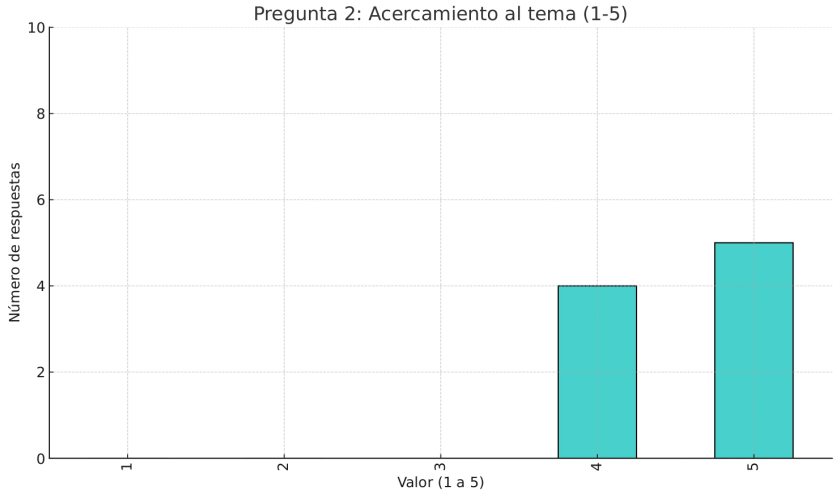

FIGURA 20: Percepción del acercamiento al procesamiento de señales de audio.

3. ¿El nivel de profundidad fue adecuado?

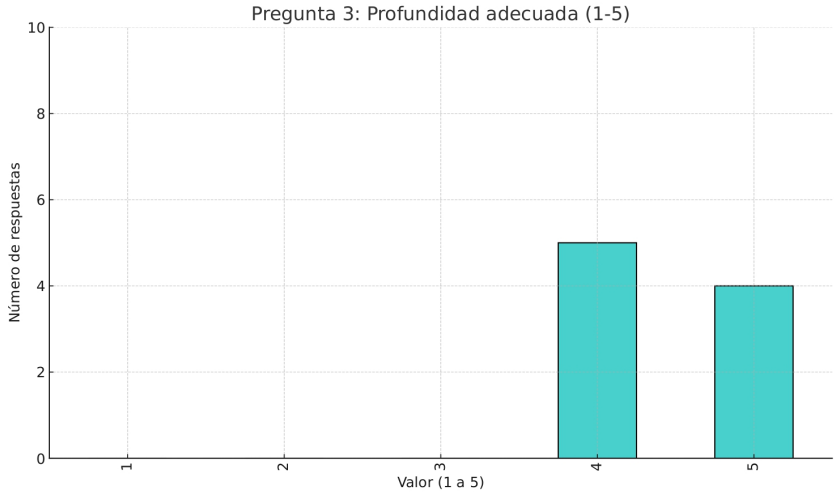

FIGURA 21: Valoración del nivel de profundidad en la clase.

4. ¿Qué tan útil le pareció la herramienta lúdica presentada para aprender procesamiento de señales?

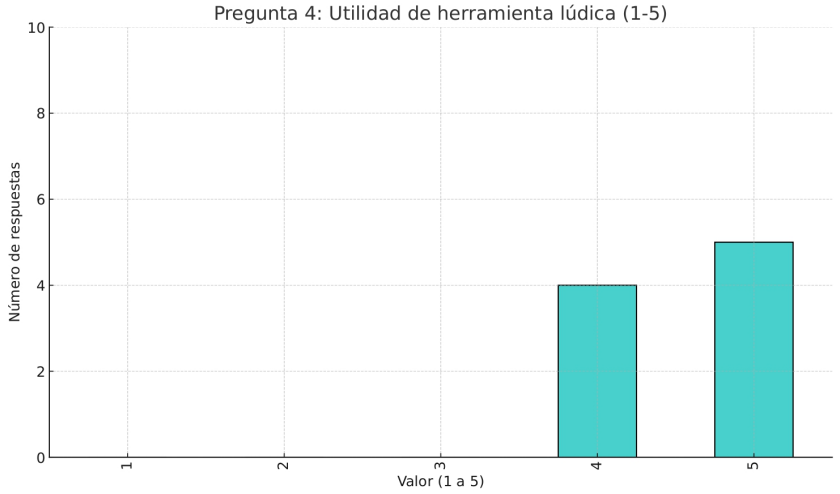

FIGURA 22: Utilidad percibida de la herramienta lúdica.

5. ¿Después de la clase, se siente más motivado(a) a explorar el área de procesamiento de señales?

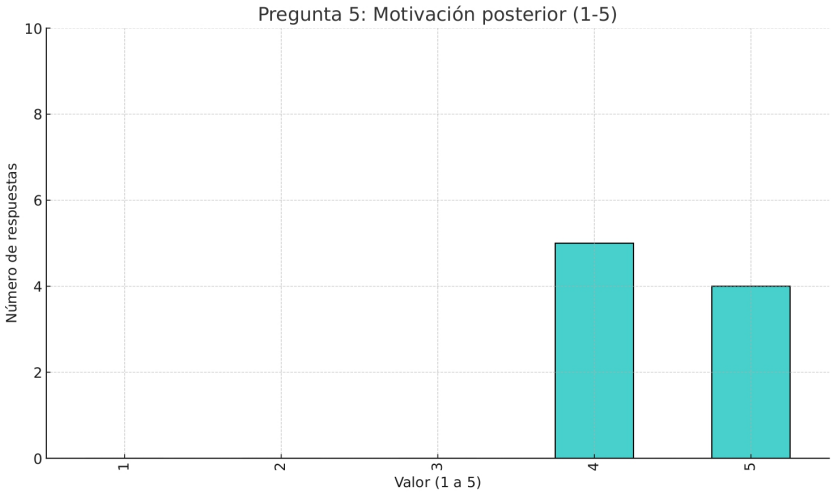

FIGURA 23: Motivación posterior a la clase.

6. ¿La dinámica de la clase fue participativa e interactiva?

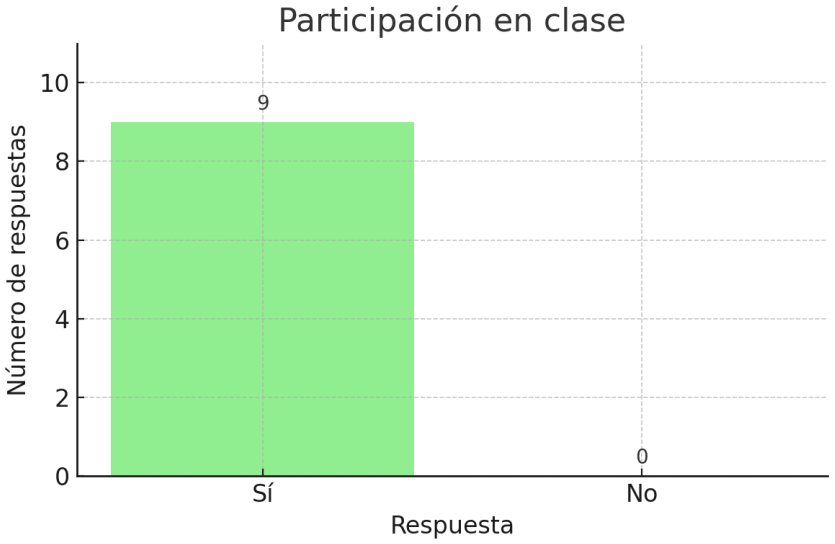

FIGURA 24: Percepción de participación activa durante la clase.

7. ¿Considera que esta herramienta podría ser implementada en sus cursos regulares o proyecto de grado?

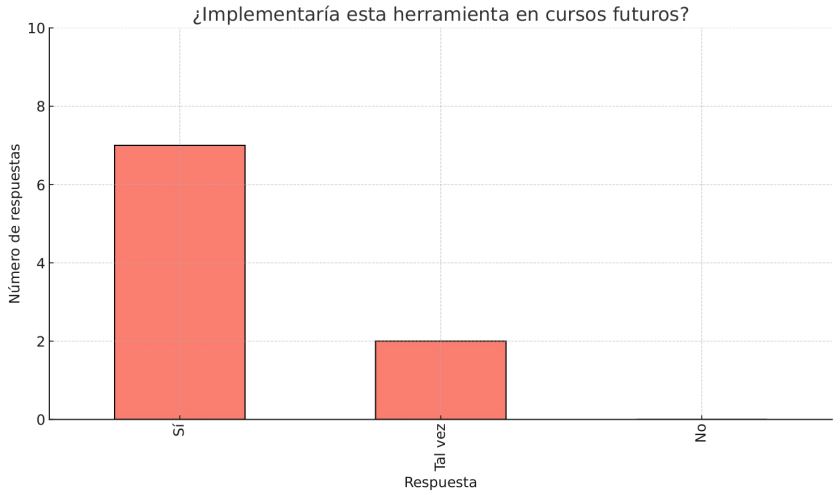

FIGURA 25: Opinión sobre la implementación futura de la herramienta.

TABLA 13: Opinión sobre la implementación futura de la herramienta

Respuesta	**Número de respuestas**
Sí	7
Tal vez	2
No	0

Los resultados muestran que la mayoría de los estudiantes que estuvieron presentes en la actividad calificó positivamente los temas abordados, el enfoque participativo y la utilidad de la herramienta desarrollada. En particular:

- El 100 % indicó que la clase fue participativa e interactiva.
- El 100 % otorgó puntuaciones de 4 o 5 a la claridad de la explicación, la profundidad y la herramienta lúdica.
- El 100 % expresó sentirse motivado a explorar el área.
- Un 77.8 % indicó que **sí** implementaría la herramienta en sus cursos o proyectos, mientras que el 22.2 % restante respondió **tal vez**.

Estos resultados permiten concluir que la estrategia pedagógica no solo fue efectiva en transmitir los contenidos técnicos del procesamiento digital de señales de audio, sino que también despertó un interés real y aplicable en los estudiantes. Esta validación directa refuerza la pertinencia académica de la guía desarrollada y del enfoque del presente proyecto de grado.

Capítulo 6

Resultados y discusión

Los resultados obtenidos en este proyecto evidencian una clara diferenciación en el desempeño de los modelos de clasificación de comandos de voz evaluados, reflejando tanto fortalezas como limitaciones relevantes para aplicaciones prácticas en sistemas embebidos. Los detalles pueden observarse en el apartado 5.2.4, titulado Evaluación de rendimiento de algoritmos, donde se presenta una comparación directa entre GMM, DTW y Distancia Euclidiana con CMVN bajo diferentes condiciones de ruido.

En condiciones ideales, con bajo nivel de ruido ambiental, los tres modelos alcanzaron una precisión cercana al 100 %, indicando que la metodología propuesta es adecuada para ambientes controlados y silenciosos.

Sin embargo, la introducción de ruido ambiental moderado (50 dB) puso en evidencia las diferencias en la robustez y tolerancia al ruido de cada algoritmo. El modelo basado en Mezcla Gaussiana (GMM) demostró la mayor estabilidad y menor sensibilidad a variaciones acústicas, manteniendo una precisión del 86 % y distribuyendo los errores de forma homogénea entre los comandos, lo que sugiere una mejor capacidad para generalizar en presencia de interferencias. Por el contrario, el modelo basado en Distancia Euclidiana con Normalización (CMVN) presentó una caída más significativa en la precisión, especialmente en comandos acústicamente similares o cortos, lo que limita su aplicabilidad en entornos ruidosos. El modelo DTW, aunque alcanzó una precisión similar a la del GMM, mostró vulnerabilidad frente a comandos con patrones temporales complejos y presentó mayor variabilidad en tiempos de procesamiento.

Desde una perspectiva computacional, el modelo GMM también se destacó por requerir el menor número de ciclos de máquina para procesar los comandos, lo que es crucial para su implementación en plataformas con recursos limitados como DSP embebidos. Esta combinación

de eficiencia y robustez posiciona al GMM como el modelo óptimo para sistemas domóticos controlados por voz en entornos con ruido ambiental moderado.

No obstante, se identificaron limitaciones inherentes a los modelos ante niveles de ruido superiores a 65 dB, especialmente cuando el ruido proviene de otras fuentes vocales, lo que genera dificultad en la discriminación y puede afectar la confiabilidad del sistema. Además, el análisis cualitativo revela que ciertos comandos presentan mayor susceptibilidad a errores, lo que sugiere la necesidad de ajustes o entrenamientos específicos para mejorar su reconocimiento.

Finalmente, los resultados educativos derivados de la implementación y evaluación del sistema corroboran que la estrategia pedagógica utilizada no solo facilitó la comprensión técnica del procesamiento digital de señales de audio, sino que también motivó y generó un interés significativo en los estudiantes. Esta contribución técnica y formativa fortalece la pertinencia y el impacto del proyecto, abriendo caminos para futuras investigaciones y desarrollos en la materia.

Discusión

La superioridad del modelo GMM en ambientes con ruido ambiental moderado se explica principalmente por su capacidad para modelar la distribución probabilística de las características acústicas de los comandos, lo que permite una mayor tolerancia a variaciones e interferencias no lineales. Esta propiedad está alineada con los fundamentos teóricos del procesamiento de señales, donde los métodos basados en estadística pueden capturar la variabilidad inherente a señales de voz en condiciones reales [47]. Por el contrario, el enfoque CMVN, aunque eficaz para normalizar y comparar características, carece de mecanismos intrínsecos para modelar ruido y variabilidad compleja, lo que se traduce en una menor robustez en escenarios no ideales.

En cuanto al algoritmo DTW, su capacidad para alinear patrones temporales lo hace adecuado para comandos con duración variable; sin embargo, su vulnerabilidad frente a ruido se incrementa debido a la sensibilidad a distorsiones locales en la señal, además de su mayor costo computacional, limitando su uso en dispositivos con recursos restringidos. Esta observación es consistente con estudios previos que documentan la eficiencia y limitaciones de DTW en aplicaciones embebidas [65].

Los hallazgos obtenidos concuerdan con investigaciones recientes que destacan la eficacia de modelos probabilísticos como GMM para reconocimiento de voz en entornos ruidosos y sistemas embebidos [66, 67]. Por ejemplo, Huang et al. [66] demostraron que la incorporación de

modelos estadísticos mejora significativamente la precisión en condiciones acústicas adversas, respaldando la elección metodológica adoptada en este proyecto.

Simultáneamente, se confirma que los métodos basados en distancia euclidiana y normalización presentan limitaciones similares a las observadas en esta investigación, lo que sugiere la necesidad de complementarlos con técnicas adicionales, como filtros adaptativos o aprendizaje profundo, para lograr una mayor robustez [68].

Estas coincidencias y diferencias refuerzan la validez de los resultados aquí presentados y aportan un marco sólido para futuras mejoras y aplicaciones prácticas en sistemas domóticos controlados por voz.

Capítulo 7

Conclusiones y Trabajos Futuros

- Se seleccionó el micrófono omnidireccional iTalk-01, el cual demostró ser la opción más adecuada para el sistema debido a su sensibilidad de $-37 \pm 3,$ dB, respuesta en frecuencia de 10Hz a 10kHz y patrón polar omnidireccional, permitiendo una captación clara de voz en entornos cerrados sin necesidad de orientación directa. Su compatibilidad con el codec TLV320AIC3204 y la disponibilidad institucional (sin costo de adquisición) lo convirtieron en una alternativa técnica y económicamente ideal. La señal fue adaptada correctamente a los niveles requeridos por el convertidor A/D, garantizando fidelidad en la adquisición.

- Los modelos de clasificación evaluados (GMM, DTW y Distancia Euclidiana con CMVN) mostraron un desempeño sobresaliente en ambientes bajo condiciones silenciosas (35 dB). En presencia de ruido moderado (50 dB), el modelo GMM destacó por su mayor estabilidad, logrando una precisión del 86 % con baja variabilidad entre ejecuciones. No obstante, cuando el nivel de ruido supera los 65 dB, especialmente ante interferencias vocales simultáneas, la confiabilidad del sistema se reduce considerablemente, dificultando la correcta identificación de los comandos.

- La validación del sistema se llevó a cabo con cinco electrodomésticos reales (luz, calefacción, ventilador, alarma y televisor) operados mediante módulos de relé controlados por un Arduino, el cual recibe comandos desde la DSP a través de comunicación serial UART. Si bien inicialmente se planeaba utilizar directamente las salidas GPIO de la DSP, la falta de disponibilidad de conectores hembra compatibles motivó la adopción de esta solución alternativa basada en comunicación serial.

- La guía de laboratorio diseñada demostró ser una herramienta efectiva para introducir a los estudiantes en el procesamiento digital de señales de audio. Los resultados de la

encuesta aplicada evidenciaron un aumento significativo en la motivación y el interés por esta área poco explorada en el contexto institucional. Se concluye que la universidad, al contar con tarjetas DSP TMS320C5535 un procesador digital especializado en aplicaciones de audio por su arquitectura optimizada para señales en tiempo real tiene la capacidad de fomentar activamente la formación en procesamiento de voz, estimulando líneas de investigación en semilleros y futuros trabajos de grado.

Trabajos Futuros

- Dado que el sistema pierde confiabilidad en entornos con niveles de ruido superiores a 65 dB, particularmente cuando se trata de ruido vocal, se propone como trabajo futuro la incorporación de técnicas de cancelación adaptativa de ruido, con el objetivo de mejorar la presición del reconocimiento en condiciones acústicas complejas.
- Implementar el inicio de la ejecución del sistema mediante un comando clave en lugar de un botón físico, para permitir un control más natural e intuitivo, facilitando la interacción manos libres y mejorando la experiencia del usuario.

Referencias bibliográficas

[1] Universidad de la Costa. *Solo 1 de cada 5 profesionales en ingeniería de sistemas es mujer*. Blog CUC. Feb. de 2024. URL: https://virtual.cuc.edu.co/blog/solo-1-de-cada-5-profesionales-en-ingenieria-de-sistemas-es-mujer.

[2] Semana. «¿Y dónde están los ingenieros?» En: *Semana.com* (15 de sep. de 2014). URL: https://www.semana.com/tecnologia/articulo/y-donde-estan-los-ingenieros/402945-3/.

[3] J. Sebastián et al. *Percepciones sobre la ingeniería en estudiantes de colegios públicos en Bogotá*. Paper. Accedido: 2024-11-12. ACOFI Papers, 2019. URL: https://antiguo.acofipapers.org/index.php/eiei2019/2019/paper/viewFile/3396/1357.

[4] Politécnico Grancolombiano. *¿Qué es STEM? y cómo sacarle provecho en Colombia*. Blog Poli.edu.co. Ago. de 2024. URL: https://www.poli.edu.co/blog/poliverso/que-es-stem-y-como-sacarle-provecho-en-colombia.

[5] Ministerio de Educación Nacional de Colombia. *Pruebas PISA 2022: Colombia, un sistema educativo resiliente...* Comunicado de prensa. 2022. URL: https://www.mineducacion.gov.co/portal/salaprensa/Comunicados/417751:Pruebas-PISA-2022-Colombia-un-sistema-educativo-resiliente-que-requiere-cambios-estructurales-para-mejorar-su-calidad.

[6] M. R. Rodríguez. «Factores que inciden en la deserción estudiantil de la Carrera Ingeniería en Sistemas de Información en la FAREM-Estelí, UNAN-Managua». En: *Revista Científica de FAREM-Estelí* 33 (mar. de 2020), págs. 35-51. DOI: 10.5377/farem.v0i33.9607.

[7] M. Vetterli, J. Kovačević y V. Goyal. *Foundations of Signal Processing*. Cambridge University Press, 2014. URL: https://assets.cambridge.org/97811070/38608/frontmatter/9781107038608_frontmatter.pdf.

[8] Universidad Santo Tomás. *Buscar: DSP*. Repositorio USTA. 2022. URL: https://repository.usta.edu.co/handle/11634/85/discover?query=DSP.

[9] E. Blass y P. Hayward. «Innovation in higher education; will there be a role for 'the academe/university' in 2025?» En: *European Journal of Futures Research* 2.1 (mayo de 2014). DOI: 10.1007/s40309-014-0041-x.

[10] Engineer Choice. *What Is The Future Of DSP Engineering?* Blog. Nov. de 2023. URL: https://engrchoice.com/what-is-the-future-of-dsp-engineering/.

[11] Grupo Atico34. *Reconocimiento de voz: Qué es, cómo funciona y tipos que existen*. Protección Datos LOPD. Jun. de 2024. URL: https://protecciondatos-lopd.com/empresas/reconocimiento-de-voz/.

[12] Peerdh. *Optimización de algoritmos de reconocimiento de voz para baja latencia en dispositivos de borde*. Blog. URL: https://peerdh.com/es/blogs/programming-insights/optimizing-voice-recognition-algorithms-for-low-latency-processing-on-edge-devices-2.

[13] P. De Enseñanza et al. *La importancia de la lúdica como estrategia didáctica en el*. Inf. téc. Accedido: 2024-11-12. Universidad Militar Nueva Granada. URL: https://repository.unimilitar.edu.co/server/api/core/bitstreams/5bff0044-a1eb-45a1-bdfa-00b51da184da/content.

[14] Amnistía Internacional. *Big Tech privacy poll shows people worried*. Comunicado. Dic. de 2019. URL: https://www.amnesty.org/es/latest/press-release/2019/12/big-tech-privacy-poll-shows-people-worried/.

[15] J. Li et al. «Security and privacy problems in voice assistant applications: A survey». En: *Computers Security* (2023). DOI: 10.1016/j.cose.2023.103448.

[16] C. Gonzalez-Cadenillas y N. Murrugarra-Llerena. «Isolated Words Recognition Using a Low Cost Microcontroller». En: *III Brazilian Symposium on Computing Systems Engineering*. Nov. de 2013. DOI: 10.1109/sbesc.2013.28.

[17] United Nations. *Transforming our world: The 2030 Agenda for Sustainable Development*. Resolution A/RES/70/1. Sep. de 2015. URL: https://www.un.org/sustainabledevelopment/sustainable-development-goals/.

[18] D. Yu y L. Deng. *Automatic Speech Recognition: A Deep Learning Approach*. London: Springer, 2015. URL: https://zhaoshuaijiang.com/file/Signals_and_Communication_Technology_Aut.pdf.

[19] Instituto Tecnológico de Aguascalientes. *DSC para reconocimiento de voz*. Redalyc. Accedido: 2024-08-23. URL: https://www.redalyc.org/pdf/944/94403203.pdf.

[20] B. Víctor et al. *Facultad de Ingeniería, Arquitectura y Urbanismo: Escuela Académico Profesional de Ingeniería de Sistemas*. Repositorio USS. Accedido: 2024-09-19. URL: https://repositorio.uss.edu.pe/bitstream/handle/20.500.12802/5250/Ruiz%20Vargas.pdf.

[21] Universidad de Montevideo. «Evaluación comparativa de sistemas de reconocimiento de locutor basados en LPC, CC y MFCC». En: *Revista de Ingeniería* (2024). URL: https://revistas.um.edu.uy/index.php/ingenieria/article/view/390/479.

[22] L. Cañete, A. Pereira y C. Alvarez. *Algoritmo para el reconocimiento de comandos de voz*. ResearchGate. 2012. URL: https://www.researchgate.net/publication/281241777_Algoritmo_para_el_reconocimiento_de_comandos_de_voz.

[23] B. Víctor et al. *FACULTAD DE INGENIERÍA, ARQUITECTURA Y URBANISMO Escuela Académico Profesional de Ingeniería de Sistemas*. Accessed: 2024-09-19. 2024. URL: https://repositorio.uss.edu.pe/bitstream/handle/20.500.12802/5250/Ruiz%20Vargas.pdf?sequence=1&isAllowed=y.

[24] A. E. Ruiz Sierra y cols. *Diseño e implementación de un sistema de reconocimiento de voz para sillas de ruedas mediante Arduino*. Repositorio Unisucre. URL: https://repositorio.unisucre.edu.co/server/api/core/bitstreams/d667e014-2d36-4728-ba56-e4c6806f0421/content.

[25] J. Camargo, E. Gaona y L. García. *Reconocimiento de voz humana aplicado a la domótica*. Dialnet. URL: https://dialnet.unirioja.es/descarga/articulo/5038438.pdf.

[26] J. Panta Martínez. *Control domótico por voz*. Repositorio UPV. URL: https://riunet.upv.es/bitstream/handle/10251/17631/Memoria.pdf.

[27] C. Paucar Robles y cols. *Asistente domótico de control por voz para Home I/O basado en OpenHAB*. Biblus US. URL: https://biblus.us.es/bibing/proyectos/abreproy/72088/fichero/TFM-2088+PAUCAR+ROBLES%2C+CARLOS+ANDR%C3%89S.pdf.

[28] Andrei Kolesau y Dušan Šešok. «Voice Activation Systems for Embedded Devices: Systematic Literature Review». En: *Informatica* 31.1 (2020), págs. 65-88.

[29] Jing Chen et al. «A Novel Single-Word Speech Recognition on Embedded Systems Using a Convolution Neural Network with Improved Out-of-Distribution Detection». En: *Electronics* 13.3 (2024), pág. 530.

[30] International Telecommunication Union (ITU-T). *Pulse Code Modulation (PCM) of Voice Frequencies*. Recommendation G.711. Geneva, Switzerland, 1988.

[31] L. R. Rabiner y B. H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[32] GeeksforGeeks. *Mel-frequency Cepstral Coefficients (MFCC) for Speech Recognition*. https://www.geeksforgeeks.org/mel-frequency-cepstral-coefficients-mfcc-for-speech-recognition/. Accessed: 2025-04-29. 2023.

[33] MathWorks. *Speaker Verification Using Gaussian Mixture Model*. https://www.mathworks.com/help/audio/ug/speaker-verification-using-gaussian-mixture-model.html. Accessed: 2025-04-29. 2023.

[34] H. Sakoe y S. Chiba. «Dynamic Programming Algorithm Optimization for Spoken Word Recognition». En: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (feb. de 1978), págs. 43-49. DOI: 10.1109/TASSP.1978.1163055.

[35] University of Maryland. *Voice Recognition on Simple Microcontrollers*. https://www.cs.umd.edu/~dchou/papers/818w_paper.pdf. Accessed: 2025-04-29. 2020.

[36] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[37] Douglas A. Reynolds y Ronald C. Rose. «Robust text-independent speaker identification using Gaussian mixture speaker model». En: *IEEE Transactions on Speech and Audio Processing* 3.1 (1995), págs. 72-83.

[38] S. B. Davis y P. Mermelstein. «Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences». En: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.4 (1980), págs. 357-366.

[39] Julius O. Smith III. *Spectral Audio Signal Processing*. Online book, accessed via https://ccrma.stanford.edu/~jos/sasp/. W3K Publishing, 2011. URL: https://ccrma.stanford.edu/~jos/sasp/.

[40] Garima Sharma, Kartikeyan Umapathy y Sridhar Krishnan. «Trends in audio signal feature extraction methods». En: *Applied Acoustics* 158 (2020), pág. 107020. DOI: 10.1016/j.apacoust.2019.107020.

[41] David Barragán Bermúdez. «Sistema de reconocimiento de comandos de voz como interfaz de una habitación con tres atmósferas lumínicas». Proyecto de grado. Tesis de mtría. Universidad Santo Tomás, 2022.

[42] Julio Benítez, César Gutiérrez y Camilo Perdomo. «Implementación de un sistema embebido para reconocimiento de vocales mediante MFCC y clasificación GMM en un microcontrolador ARM Cortex-M». En: *Ingeniería y Universidad* 23.2 (2019), págs. 197-220.

[43] Brian B. Monson et al. «The perceptual significance of high-frequency energy in the human voice». En: *Frontiers in Psychology* 5 (2014), pág. 587. DOI: 10.3389/fpsyg.2014.00587. URL: https://www.frontiersin.org/articles/10.3389/fpsyg.2014.00587/full.

[44] Texas Instruments. *Audio Pre-processing System Reference Design for Voice Based Applications Using C5517*. Application Report SPRABM0. TI, 2012. URL: https://www.ti.com/lit/ug/tiducy1c/tiducy1c.pdf?ts=1748472124142.

[45] Texas Instruments. *TLV320AIC3204 Application Reference Guide*. https://www.ti.com/lit/pdf/slaa557. Literature Number: SLAA557. Nov. de 2012.

[46] Steven B. Davis y Paul Mermelstein. «Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences». En: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.4 (1980), págs. 357-366. DOI: 10.1109/TASSP.1980.1163420.

[47] Lawrence R. Rabiner y Biing-Hwang Juang. *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice Hall, 1993. ISBN: 978-0-13-015157-3.

[48] Vivek S. Tiwari. «MFCC and Its Applications in Speech Recognition». En: *International Journal of Emerging Technologies and Applications in Engineering, Technology and Sciences* 4.2 (2010). ISSN 0974-3588, págs. 227-231.

[49] Sadaoki Furui. «Cepstral analysis technique for automatic speaker verification». En: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29.2 (1981), págs. 254-272. DOI: 10.1109/TASSP.1981.1163536.

[50] Veton Z. Kepuska y Gamal Bohouta. «Next-Generation of Speech Recognition System for Controlling VHF/UHF Radios aboard Aircraft». En: *Proceedings of the International Multi-Conference on Engineering and Technological Innovation (IMETI)*. 2014, págs. 123-129.

[51] Haizhou Li y En Chang. «A comparative study of MFCC and LPCC features for speaker verification». En: *Proceedings of the International Conference on Machine Learning and Cybernetics* 4 (2007), págs. 2791-2794. DOI: 10.1109/ICMLC.2007.4370669.

[52] Douglas O'Shaughnessy. *Speech Communications: Human and Machine*. 2nd. ISBN 978-0780353867. Piscataway, NJ: IEEE Press, 2003.

[53] Janez Modic, Borut Batagelj y Borut Kos. «Wavelet packet based method for phoneme recognition». En: *EURASIP Journal on Applied Signal Processing* 2003.11 (2003), págs. 1081-1092. DOI: 10.1155/S111086570321105X.

[54] William M. Chan y Saeed V. Vaseghi. «Wavelet Speech Feature Extraction for Robust Speech Recognition». En: *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. 2004, I-581-I-584. DOI: 10.1109/ICASSP.2004.1326036.

[55] Julius O. Smith. *Introduction to Digital Signal Processing: A Practical and Applied Approach*. 1st. ISBN 978-0-9745607-4-8. Stanford, CA: W3K Publishing, 2018.

[56] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016. ISBN: 978-0262035613.

[57] Nello Cristianini y John Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000. ISBN: 978-0511801389.

[58] *MICROPHONE CONDENSER OMNIDIRECTIONAL 3.5MM ITALK-01*. https://www.blibli.com/p/tyless-360-degree-microphone-table-conference-meeting-studio-italk-01/ps--HES-70026-00048. Consultado en mayo de 2025. 2025.

[59] Martti Vainio. *Speech Processing Book - Voice Activity Detection*. https://speechprocessingbook.aalto.fi/. Consultado en 2025. 2021.

[60] Xeridia. *Normalización de datos en Machine Learning: Z-score vs MinMax*. https://pitch.xeridia.com/normalizacion-z-score-vs-minmax/. Consultado en 2025. 2022.

[61] Microsoft. *What is z-score normalization in ML?* https://learn.microsoft.com/en-us/azure/machine-learning/algorithm-module-reference/z-score-normalization. Consultado en 2025. 2023.

[62] UPIITA-IPN. «Técnicas de preénfasis en señales de voz». En: *Boletín de Ingeniería* (2020). Disponible en http://boletin.upiita.ipn.mx/preenfasis-voz.

[63] J. Gómez. «Preénfasis y análisis espectral en voz». En: *Dialnet* (2019). https://dialnet.unirioja.es/descarga/articulo/1234567.pdf.

[64] Alan V. Oppenheim, Ronald W. Schafer y John R. Buck. *Discrete-Time Signal Processing*. 2nd. Upper Saddle River, New Jersey: Prentice Hall, 1999.

[65] Meinard Müller. «Dynamic Time Warping». En: *Information Retrieval for Music and Motion* (2007), págs. 69-84. DOI: 10.1007/978-3-540-74048-3_4.

[66] X. Huang, J. Li y L. Deng. «Deep Learning for Speech Recognition: Recent Advances and Future Trends». En: *IEEE Signal Processing Magazine* 29.6 (2014), págs. 82-97. DOI: 10.1109/MSP.2012.2205597.

[67] H. Lee et al. «Unsupervised Feature Learning for Audio Classification Using Convolutional Deep Belief Networks». En: *Advances in Neural Information Processing Systems* 22 (2015), págs. 1096-1104.

[68] Z. Zhao, D. Wang e Y. Zhang. «Robust Voice Activity Detection Based on Deep Neural Networks and Uncertainty Modeling». En: *IEEE Transactions on Audio, Speech, and Language Processing* 25.12 (2017), págs. 2347-2358. DOI: 10.1109/TASLP.2017.2752296.

Anexos

7.1. Guía de Laboratorio: Inicialización de la DSP y Procesamiento de Señales de Audio

A continuación, se presenta la guía de laboratorio desarrollada como parte del presente proyecto. Esta guía introduce al estudiante en el entorno de desarrollo de la DSP TMS320C5535 y en las primeras fases del procesamiento de señales de audio, incluyendo normalización por Z-score y eliminación de silencios.

UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
FACULTAD INGENIERÍA ELECTRÓNICA

Inicialización de la DSP y Procesamiento de Señales de Audio

1º David Ricardo Melo Suárez
Estudiante. Ingeniería Electrónica
Universidad Santo Tomas
Bogotá, Colombia
davidmelos@usantotomas.edu.co

2º Sergio Emiro Vargas Cruz
Estudiante. Ingeniería Electrónica
Universidad Santo Tomas
Bogotá, Colombia
sergiovargasc@usantotomas.edu.co

3º Carlos Javier Mojica Casallas
Docente. Ingeniería Electrónica
Universidad Santo Tomas
Bogotá, Colombia
carlosmojica@usta.edu.co

OBJETIVOS

- Adquirir conocimientos sobre la configuración inicial de la DSP TMS320C5535.
- Implementar las primeras etapas del procesamiento de voz: Normalización y eliminación de silencios.
- Prepararse para las siguientes fases de procesamiento digital de señales de audio para reconocimiento de voz.

COMPETENCIAS A DESARROLLAR

- Comprender la importancia de la normalización en señales de voz y cómo implementarla en la DSP.
- Entender la técnica de eliminación de silencios y cómo utilizarla para mejorar la eficiencia del procesamiento.
- Hacer uso del entorno de desarrollo en la DSP TMS320C5535 para proyectos de procesamiento de señales de audio.
- Utilizar herramientas y recursos disponibles en GitHub para implementar los primeros pasos en procesamiento de señales.

MATERIALES

- DSP TMS320C5535.
- Archivos y librerías disponibles en el repositorio de GitHub: https://github.com/DavidMelo2003/VoiceRecognizedTMS320C5535/tree/main.
- Video tutorial para la inicialización de la DSP: https://youtu.be/cg2vYDqPq-A.
- Computadora con software de desarrollo adecuado para la DSP.
- Micrófono y sistema de captura de audio.

PROCEDIMIENTO

Paso 1: Inicialización de la DSP

- Primero, consulte el video tutorial donde se explica la inicialización de la DSP TMS320C5535.
- Asegúrese de seguir los pasos mostrados en el video, utilizando el código y las librerías proporcionadas en el repositorio de GitHub. Este repositorio contiene todo lo necesario para comenzar con la inicialización.
- Realice la conexión de la DSP a su computadora y configure la comunicación para el proceso de programación.

Paso 2: Normalización de la Señal de Voz

- La normalización ajusta la amplitud de la señal de voz para asegurar que todas las señales tengan un nivel de energía uniforme.
- Implementar la normalización como se describe en el código fuente del repositorio.
- Verifique el resultado utilizando las herramientas de depuración de la DSP para asegurarse de que la señal está correctamente normalizada.

Paso 3: Eliminación de Silencios

- La eliminación de silencios se realiza para eliminar segmentos de la señal que no contienen habla.
- Revise el funcionamiento y asegúrese de que los silencios se eliminan correctamente antes de proceder al siguiente paso.

Cuadro I
REGISTRO DE RESULTADOS DE LA ELIMINACIÓN DE SILENCIOS

Comando a reconocer	**Número de muestras antes de la eliminación de silencios**	**Número de muestras después de la eliminación de silencios**
1	_____	_____
2	_____	_____
3	_____	_____

RESULTADOS ESPERADOS

- La DSP debe estar correctamente inicializada y configurada para el procesamiento de audio.
- La señal de voz debe estar normalizada y libre de silencios, lo que optimiza el rendimiento del sistema de reconocimiento de voz.
- Se debe obtener una secuencia de señales de voz lista para el análisis espectral en las siguientes etapas del procesamiento de audio.

CONCLUSIÓN

Al finalizar esta guía de laboratorio, los estudiantes habrán completado con éxito las primeras etapas del procesamiento de voz: la normalización y la eliminación de silencios. Estos son pasos fundamentales para cualquier sistema de procesamiento de señales de voz, y proporcionan la base para futuras implementaciones y mejoras en sistemas embebidos de reconocimiento de voz.

7.2. Configuración de periféricos (C)

```c
#include "ezdsp5535.h"
#include "ezdsp5535_gpio.h"
#include "ezdsp5535_i2c.h"
#include "ezdsp5535_i2s.h"
#include "stdio.h"

#include "csl_i2s.h"

#include <csl_mmcsd.h>
#include <csl_intc.h>
#include <csl_general.h>
#include <soc.h>

/* SD card Buffer size in Bytes                                          */
#define BUFFER_MAX_SIZE    (1024u)
/* SD card physical address with respect to sector number                */
#define CARD_START_ADDR    (0x0)

/* Macros used to calculate system clock from PLL configurations         */
#define CSL_PLL_DIV_000    (0)
#define CSL_PLL_DIV_001    (1u)
#define CSL_PLL_DIV_002    (2u)
#define CSL_PLL_DIV_003    (3u)
#define CSL_PLL_DIV_004    (4u)
#define CSL_PLL_DIV_005    (5u)
#define CSL_PLL_DIV_006    (6u)
#define CSL_PLL_DIV_007    (7u)
#define CSL_PLL_CLOCKIN    (32768u)

#define CSL_SD_CLOCK_MAX_KHZ      (20000u)
#define AIC3204_I2C_ADDR 0x18

CSL_MMCControllerObj    pMmcsdContObj;
CSL_MmcsdHandle         mmcsdHandle;
CSL_MMCCardObj          mmcCardObj;
CSL_MMCCardIdObj        sdCardIdObj;
CSL_MMCCardCsdObj       sdCardCsdObj;

CSL_Status   mmcStatus;
Uint32       cardStatus;

Uint32 cardAddr;

CSL_Status CSL_sdConfig();
CSL_Status CSL_sdClose();

Uint32 getSysClk(void);
Uint16 computeClkRate(void);
Int16 preprocess_sample(Int16 raw_sample);
Int16 AIC3204_rset( Uint16 regnum, Uint16 regval);
Int16 AIC3204_rset( Uint16 regnum, Uint16 regval );
```

```
52 Int16 Configuracion( );
53 void Close(void);
54
55
56 void aic3204_Config(void)
57 {
58     /* Configure AIC3204 */
59     AIC3204_rset(0, 0x00);   // Pagina 0
60     AIC3204_rset(1, 0x01);   // Reset del codec
61     EZDSP5535_waitusec(1000);
62     AIC3204_rset(0, 0x01);   // Pagina 1
63     AIC3204_rset(1, 0x08);   // AVDD desde LDO (no desde DVDD)
64     AIC3204_rset(2, 0x01);   // Habilitar bloques anal gicos
65     AIC3204_rset(123, 0x05); // Encender referencia anal gica
66     EZDSP5535_waitusec(100000); // Espera 100 ms (para estabilidad)
67     AIC3204_rset(0, 0x00);   // Pagina 0
68
69     // ------------------------------------------------
70     // Configuraci n de PLL y Divisores para 16 kHz
71     // ------------------------------------------------
72     AIC3204_rset(4, 0x03);   // PLL habilitado, CLKIN = MCLK (12 MHz)
73     AIC3204_rset(6, 0x08);   // J = 8
74     AIC3204_rset(7, 0x02);   // D = 512 (High Byte: 0x0200)
75     AIC3204_rset(8, 0x00);   // D = 512 (Low Byte)
76     AIC3204_rset(5, 0x91);   // Encender PLL (P=1, R=1)
77     EZDSP5535_waitusec(20000); // Esperar 20 ms
78
79     // Divisores DAC (16 kHz)
80     AIC3204_rset(11, 0x84);  // NDAC = 4 (Power up + valor)
81     AIC3204_rset(12, 0x8C);  // MDAC = 12
82     AIC3204_rset(13, 0x00);  // DOSR High = 64
83     AIC3204_rset(14, 0x40);  // DOSR Low = 64
84
85     // Divisores ADC (16 kHz)
86     AIC3204_rset(18, 0x84);  // NADC = 4
87     AIC3204_rset(19, 0x8C);  // MADC = 12
88     AIC3204_rset(20, 0x40);  // AOSR = 64
89
90     // ------------------------------------------------
91     // Configuraci n de I2S (BCLK y WCLK para 16 kHz)
92     // ------------------------------------------------
93     AIC3204_rset(27, 0x0D);  // Master mode, BCLK y WCLK como salidas
94     AIC3204_rset(30, 0x88);  // 32 bits por trama (16 bits por canal)
95
96     // ------------------------------------------------
97     // Ruteo y Ganancia del DAC
98     // ------------------------------------------------
99     AIC3204_rset(0, 0x01);   // Pagina 1
100    AIC3204_rset(12, 0x08);  // LDAC     HPL
101    AIC3204_rset(13, 0x08);  // RDAC     HPR
102    AIC3204_rset(0, 0x00);   // Pagina 0
103    AIC3204_rset(64, 0x02);  // Volumen izquierdo = derecho
104    AIC3204_rset(65, 0x30);  // Ganancia DAC a +3 dB (0x20 = +3dB)
```

```
105     AIC3204_rset(63, 0xD4);  // Formato I2S, 16 bits, DACs encendidos
106     AIC3204_rset(0, 0x01);   // Pagina 1
107     AIC3204_rset(16, 0x00);  // HPL a 0 dB (0x08 = 0 dB)
108     AIC3204_rset(17, 0x00);  // HPR a 0 dB
109     AIC3204_rset(9, 0x30);   // Encender HPL y HPR
110
111     // ------------------------------------------------
112     // Ruteo y Ganancia del ADC
113     // ------------------------------------------------
114     AIC3204_rset(0, 0x01);   // Pagina 1
115     AIC3204_rset(52, 0x30);  // IN2_L     LADC_P (Micr fono)
116     AIC3204_rset(55, 0x30);  // IN2_R     RADC_P
117     AIC3204_rset(54, 0x03);  // CM_1     LADC_M
118     AIC3204_rset(57, 0xC0);  // CM_1     RADC_M
119     AIC3204_rset(51, 0x48);  // BIAS de micr fono a 2.5V
120     AIC3204_rset(59, 0x40);  // MIC_PGA_L a 24 dB (0x18 = 24 dB)
121     AIC3204_rset(60, 0x40);  // MIC_PGA_R a 24 dB
122     AIC3204_rset(0, 0x00);   // Pagina 0
123     AIC3204_rset(81, 0xC0);  // Encender ADC
124     AIC3204_rset(82, 0x00);  // Desmutear ADC
125     EZDSP5535_waitusec(100 );  // Wait
126     /* Initialize I2S */
127
128     EZDSP5535_I2S_init();
129 }
130
131 void Close()
132 {
133     CSL_sdClose();
134     EZDSP5535_I2S_close();    // Disble I2S
135     AIC3204_rset( 1, 0x01 );  // Reset codec
136 }
137
138 CSL_Status CSL_sdConfig()
139 {
140     Uint16       actCard;
141     Uint32       sectCount;
142     Uint16       clockDiv;
143     Uint16       rca;
144
145
146     sectCount = 0;
147
148     /* Initialize data buffers */
149
150     /* Get the clock divider value for the current CPU frequency */
151     clockDiv = computeClkRate();
152
153     /* Initialize the CSL MMCSD module */
154     CSL_Status mmcStatus = MMC_init();
155     if(mmcStatus != CSL_SOK)
156     {
157         printf("API: MMC_init Failed\n");
```

```
158         return(mmcStatus);
159     }
160
161     /* Open the MMCSD module in POLLED mode */
162
163 #ifdef C5515_EZDSP
164     mmcsdHandle = MMC_open(&pMmcsdContObj, CSL_MMCSD1_INST, CSL_MMCSD_OPMODE_POLLED, &
        mmcStatus);
165 #else
166     mmcsdHandle = MMC_open(&pMmcsdContObj, CSL_MMCSD0_INST, CSL_MMCSD_OPMODE_POLLED, &
        mmcStatus);
167 #endif
168     if(mmcStatus != CSL_SOK)
169     {
170         printf("API: MMC_open Failed\n");
171         return(mmcStatus);
172     }
173     else
174     {
175         printf("\n");
176     }
177
178     /* Send CMD0 to the card */
179     mmcStatus = MMC_sendGoIdle(mmcsdHandle);
180     if(mmcStatus != CSL_SOK)
181     {
182         printf("API: MMC_sendGoIdle Failed\n");
183         return(mmcStatus);
184     }
185
186     /* Check for the card */
187     mmcStatus = MMC_selectCard(mmcsdHandle, &mmcCardObj);
188     if((mmcStatus == CSL_ESYS_BADHANDLE) || (mmcStatus == CSL_ESYS_INVPARAMS))
189     {
190         printf("API: MMC_selectCard Failed\n");
191         return(mmcStatus);
192     }
193
194     /* Verify whether the SD card is detected or not */
195     if(mmcCardObj.cardType == CSL_SD_CARD)
196     {
197         printf("\n");
198
199         /* Check if the card is high capacity card */
200         if(mmcsdHandle->cardObj->sdHcDetected == TRUE)
201         {
202             printf("SD card is High Capacity Card\n");
203             printf("Memory Access will use Block Addressing\n\n");
204
205             /* For the SDHC card Block addressing will be used.
206                Sector address will be same as sector number */
207             cardAddr = sectCount;
208         }
```

```
209         else
210         {
211             printf("\n");
212             /* For the SD card Byte addressing will be used.
213                Sector address will be product of  sector number
214                and sector size */
215             cardAddr = (sectCount)*(CSL_MMCSD_BLOCK_LENGTH);
216         }
217     }
218     else
219     {
220         /* Check if No card is inserted */
221         if(mmcCardObj.cardType == CSL_CARD_NONE)
222         {
223             printf("No Card Detected!\n");
224         }
225         else
226         {
227             printf("SD card is not Detected!\n");
228         }
229 
230         printf("Please Insert SD card!!\n");
231         return(CSL_ESYS_FAIL);
232     }
233 
234     /* Set the init clock */
235     mmcStatus = MMC_sendOpCond(mmcsdHandle, 100);
236     if(mmcStatus != CSL_SOK)
237     {
238         printf("API: MMC_sendOpCond Failed\n");
239         return(mmcStatus);
240     }
241 
242     /* Send the card identification Data */
243     mmcStatus = SD_sendAllCID(mmcsdHandle, &sdCardIdObj);
244     if(mmcStatus != CSL_SOK)
245     {
246         printf("API: SD_sendAllCID Failed\n");
247         return(mmcStatus);
248     }
249 
250     /* Set the Relative Card Address */
251     mmcStatus = SD_sendRca(mmcsdHandle, &mmcCardObj, &rca);
252     if(mmcStatus != CSL_SOK)
253     {
254         printf("API: SD_sendRca Failed\n");
255         return(mmcStatus);
256     }
257 
258     /* Read the SD Card Specific Data */
259     mmcStatus = SD_getCardCsd(mmcsdHandle, &sdCardCsdObj);
260     if(mmcStatus != CSL_SOK)
261     {
```

```
262         printf("API: SD_getCardCsd Failed\n");
263         return(mmcStatus);
264     }
265
266     /* Set bus width - Optional */
267     mmcStatus = SD_setBusWidth(mmcsdHandle, 1);
268     if(mmcStatus != CSL_SOK)
269     {
270         printf("API: SD_setBusWidth Failed\n");
271         return(mmcStatus);
272     }
273
274     /* Disable SD card pull-up resistors - Optional */
275     mmcStatus = SD_configurePullup(mmcsdHandle, 1);
276     if(mmcStatus != CSL_SOK)
277     {
278         printf("API: SD_configurePullup Failed\n");
279         return(mmcStatus);
280     }
281
282     /* Set the card type in internal data structures */
283     mmcStatus = MMC_setCardType(&mmcCardObj, mmcCardObj.cardType);
284     if(mmcStatus != CSL_SOK)
285     {
286         printf("API: MMC_setCardType Failed\n");
287         return(mmcStatus);
288     }
289
290     /* Set the card pointer in internal data structures */
291     mmcStatus = MMC_setCardPtr(mmcsdHandle, &mmcCardObj);
292     if(mmcStatus != CSL_SOK)
293     {
294         printf("API: MMC_setCardPtr Failed\n");
295         return(mmcStatus);
296     }
297
298     /* Get the number of cards */
299     mmcStatus = MMC_getNumberOfCards(mmcsdHandle, &actCard);
300     if(mmcStatus != CSL_SOK)
301     {
302         printf("API: MMC_getNumberOfCards Failed\n");
303         return(mmcStatus);
304     }
305
306     /* Set clock for read-write access */
307     mmcStatus = MMC_sendOpCond(mmcsdHandle, clockDiv);
308     if(mmcStatus != CSL_SOK)
309     {
310         printf("API: MMC_sendOpCond Failed\n");
311         return(mmcStatus);
312     }
313
314     /* Set Endian mode for read and write operations */
```

```
315     mmcStatus = MMC_setEndianMode(mmcsdHandle, CSL_MMCSD_ENDIAN_LITTLE,
316                                   CSL_MMCSD_ENDIAN_LITTLE);
317     if(mmcStatus != CSL_SOK)
318     {
319         printf("API: MMC_setEndianMode Failed\n");
320         return(mmcStatus);
321     }
322
323     /* Set block length for the memory card
324      * For high capacity cards setting the block length will have
325      * no effect
326      */
327     mmcStatus = MMC_setBlockLength(mmcsdHandle, CSL_MMCSD_BLOCK_LENGTH);
328     if(mmcStatus != CSL_SOK)
329     {
330         printf("API: MMC_setBlockLength Failed\n");
331         return(mmcStatus);
332     }
333
334     Uint32 cardStatus;
335     do {
336         MMC_getCardStatus(mmcsdHandle, &cardStatus);
337     } while (cardStatus & 0x8000); // Bit 15 = busy
338
339     CSL_MMCSetupNative setup;
340     setup.rspTimeout = 0xFFFF; // M ximo timeout
341     setup.dataTimeout = 0xFFFF;
342     MMC_setupNative(mmcsdHandle, &setup);
343
344     return mmcStatus;
345 }
346
347
348
349 CSL_Status CSL_sdClose()
350 {
351     /* Get card stataus */
352     mmcStatus = MMC_getCardStatus(mmcsdHandle, &cardStatus);
353     if(mmcStatus != CSL_SOK)
354     {
355         printf("API: MMC_getCardStatus Failed\n");
356         return(mmcStatus);
357     }
358
359     /* Deselect the SD card */
360     mmcStatus = MMC_deselectCard(mmcsdHandle, &mmcCardObj);
361     if(mmcStatus != CSL_SOK)
362     {
363         printf("API: MMC_deselectCard Failed\n");
364         return(mmcStatus);
365     }
366
367     /* Clear the MMCSD card response registers */
```

```
368     mmcStatus = MMC_clearResponse(mmcsdHandle);
369     if(mmcStatus != CSL_SOK)
370     {
371         printf("API: MMC_clearResponse Failed\n");
372         return(mmcStatus);
373     }
374 
375     /* Send CMD0 to the SD card */
376     mmcStatus = MMC_sendCmd(mmcsdHandle, 0x00, 0x00, 0xFFFF);
377     if(mmcStatus != CSL_SOK)
378     {
379         printf("API: MMC_sendCmd Failed\n");
380         return(mmcStatus);
381     }
382 
383     /* Close the MMCSD module */
384     mmcStatus = MMC_close(mmcsdHandle);
385     if(mmcStatus != CSL_SOK)
386     {
387         printf("API: MMC_close Failed\n");
388         return(mmcStatus);
389     }
390     else
391     {
392         printf("API: MMC_close Successful\n");
393     }
394     return mmcStatus;
395 }
396 
397 Uint16 computeClkRate(void)
398 {
399     Uint32    sysClock;
400     Uint32    remainder;
401     Uint32    memMaxClk;
402     Uint16    clkRate;
403 
404     sysClock  = 0;
405     remainder = 0;
406     memMaxClk = CSL_SD_CLOCK_MAX_KHZ;
407     clkRate   = 0;
408 
409     /* Get the clock value at which CPU is running */
410     sysClock = getSysClk();
411 
412     if (sysClock > memMaxClk)
413     {
414         if (memMaxClk != 0)
415         {
416             clkRate   = sysClock / memMaxClk;
417             remainder = sysClock % memMaxClk;
418 
419             /*
420              * If the remainder is not equal to 0, increment clock rate to make
```

```
421             * sure that memory clock value is less than the value of
422             * 'CSL_SD_CLOCK_MAX_KHZ'.
423             */
424            if (remainder != 0)
425            {
426                clkRate++;
427            }
428
429            /*
430             * memory clock divider '(2 * (CLKRT + 1)' will always
431             * be an even number. Increment the clock rate in case of
432             * clock rate is not an even number
433             */
434            if (clkRate%2 != 0)
435            {
436                clkRate++;
437            }
438
439            /*
440             * AT this point 'clkRate' holds the value of (2 * (CLKRT + 1).
441             * Get the value of CLKRT.
442             */
443            clkRate = clkRate/2;
444            clkRate = clkRate - 1;
445
446            /*
447             * If the clock rate is more than the maximum allowed clock rate
448             * set the value of clock rate to maximum value.
449             * This case will become true only when the value of
450             * 'CSL_SD_CLOCK_MAX_KHZ' is less than the minimum possible
451             * memory clock that can be generated at a particular CPU clock.
452             *
453             */
454            if (clkRate > CSL_MMC_MAX_CLOCK_RATE)
455            {
456                clkRate = CSL_MMC_MAX_CLOCK_RATE;
457            }
458        }
459        else
460        {
461            clkRate = CSL_MMC_MAX_CLOCK_RATE;
462        }
463    }
464
465    return (clkRate);
466}
467
468/**
469 *  \brief  Function to calculate the clock at which system is running
470 *
471 *  \param    none
472 *
473 *  \return   System clock value in KHz
```

```
474  */
475
476 #if (defined(CHIP_C5505_C5515) || defined(CHIP_C5504_C5514))
477
478 Uint32 getSysClk(void)
479 {
480     Bool      pllRDBypass;
481     Bool      pllOutDiv;
482     Uint32    sysClk;
483     Uint16    pllVP;
484     Uint16    pllVS;
485     Uint16    pllRD;
486     Uint16    pllVO;
487
488     pllVP = CSL_FEXT(CSL_SYSCTRL_REGS->CGCR1, SYS_CGCR1_VP);
489     pllVS = CSL_FEXT(CSL_SYSCTRL_REGS->CGCR1, SYS_CGCR1_VS);
490
491     pllRD = CSL_FEXT(CSL_SYSCTRL_REGS->CGICR, SYS_CGICR_RDRATIO);
492     pllVO = CSL_FEXT(CSL_SYSCTRL_REGS->CGOCR, SYS_CGOCR_OD);
493
494     pllRDBypass = CSL_FEXT(CSL_SYSCTRL_REGS->CGICR, SYS_CGICR_RDBYPASS);
495     pllOutDiv   = CSL_FEXT(CSL_SYSCTRL_REGS->CGOCR, SYS_CGOCR_OUTDIVEN);
496
497     sysClk = CSL_PLL_CLOCKIN;
498
499     if (0 == pllRDBypass)
500     {
501         sysClk = sysClk/(pllRD + 4);
502     }
503
504     sysClk = (sysClk * ((pllVP << 2) + pllVS + 4));
505
506     if (1 == pllOutDiv)
507     {
508         sysClk = sysClk/(pllVO + 1);
509     }
510
511     /* Return the value of system clock in KHz */
512     return(sysClk/1000);
513 }
514
515 #else
516
517 Uint32 getSysClk(void)
518 {
519     Bool      pllRDBypass;
520     Bool      pllOutDiv;
521     Bool      pllOutDiv2;
522     Uint32    sysClk;
523     Uint16    pllVP;
524     Uint16    pllVS;
525     Uint16    pllRD;
526     Uint16    pllVO;
```

```
527     Uint16    pllDivider;
528     Uint32    pllMultiplier;
529
530     pllVP = CSL_FEXT(CSL_SYSCTRL_REGS->CGCR1, SYS_CGCR1_MH);
531     pllVS = CSL_FEXT(CSL_SYSCTRL_REGS->CGICR, SYS_CGICR_ML);
532
533     pllRD = CSL_FEXT(CSL_SYSCTRL_REGS->CGICR, SYS_CGICR_RDRATIO);
534     pllVO = CSL_FEXT(CSL_SYSCTRL_REGS->CGOCR, SYS_CGOCR_ODRATIO);
535
536     pllRDBypass = CSL_FEXT(CSL_SYSCTRL_REGS->CGICR, SYS_CGICR_RDBYPASS);
537     pllOutDiv   = CSL_FEXT(CSL_SYSCTRL_REGS->CGOCR, SYS_CGOCR_OUTDIVEN);
538     pllOutDiv2  = CSL_FEXT(CSL_SYSCTRL_REGS->CGOCR, SYS_CGOCR_OUTDIV2BYPASS);
539
540     pllDivider = ((pllOutDiv2) | (pllOutDiv << 1) | (pllRDBypass << 2));
541
542     pllMultiplier = ((Uint32)CSL_PLL_CLOCKIN * ((pllVP << 2) + pllVS + 4));
543
544     switch(pllDivider)
545     {
546         case CSL_PLL_DIV_000:
547         case CSL_PLL_DIV_001:
548             sysClk = pllMultiplier / (pllRD + 4);
549         break;
550
551         case CSL_PLL_DIV_002:
552             sysClk = pllMultiplier / ((pllRD + 4) * (pllVO + 4) * 2);
553         break;
554
555         case CSL_PLL_DIV_003:
556             sysClk = pllMultiplier / ((pllRD + 4) * 2);
557         break;
558
559         case CSL_PLL_DIV_004:
560         case CSL_PLL_DIV_005:
561             sysClk = pllMultiplier;
562         break;
563
564         case CSL_PLL_DIV_006:
565             sysClk = pllMultiplier / ((pllVO + 4) * 2);
566         break;
567
568         case CSL_PLL_DIV_007:
569             sysClk = pllMultiplier / 2;
570         break;
571     }
572
573     /* Return the value of system clock in KHz */
574     return(sysClk/1000);
575 }
576 #endif
577
578
579 Int16 AIC3204_rget(  Uint16 regnum, Uint16* regval )
```

```
580 {
581     Int16  retcode = 0;
582     Uint16 cmd[2];
583 
584     cmd[0] = regnum & 0x007F;       // 7-bit Device Register
585     cmd[1] = 0;
586 
587     retcode |= EZDSP5535_I2C_write( AIC3204_I2C_ADDR, cmd, 1 );
588     retcode |= EZDSP5535_I2C_read( AIC3204_I2C_ADDR, cmd, 1 );
589 
590     *regval = cmd[0];
591     EZDSP5535_wait( 10 );
592     return retcode;
593 }
594 
595 
596 Int16 AIC3204_rset( Uint16 regnum, Uint16 regval )
597 {
598     Uint16 cmd[2];
599     cmd[0] = regnum & 0x007F;       // 7-bit Device Register
600     cmd[1] = regval;                // 8-bit Register Data
601 
602     EZDSP5535_waitusec( 300 );
603 
604     return EZDSP5535_I2C_write( AIC3204_I2C_ADDR, cmd, 2 );
605 }
606 
607 
608 Int16 Configuracion( )
609 {
610     /* Initialize I2C */
611     EZDSP5535_I2C_init( );
612     aic3204_Config();
613     CSL_sdConfig();
614 
615     return 0;
616 }
```

LISTING 7.1: Configuración de perifericos

7.3. Preprocesamiento y Extracción de características (C)

```
1 #include "stdio.h"
2 #include "ezdsp5535.h"
3 #include "ezdsp5535_i2s.h"
4 #include "csl_i2s.h"
5 #include <csl_mmcsd.h>
6 #include <csl_intc.h>
7 #include <csl_general.h>
8 #include <soc.h>
```

```
#include "csl_gpt.h"
#include <math.h>

#define Time 2

#define N 512

#define M_PI 3.14159265358979323846


//  P a r metros  configurables

#define FRAME_SIZE 256

#define ENERGY_THRESHOLD 0.01

#define HP_COEFF 0.98

#define Solapamiento 256



// SD card Buffer size in Bytes

#define BUFFER_MAX_SIZE    (1024u)

// Coeficientes del filtro

const double cn[] = {0.9870 , -2.9610 ,   2.9610,   -0.9870};

const double cd[] = {1.0000,   -2.9738  ,  2.9480 ,  -0.9742};

#define FILTER_ORDER 3

// Buffers para las  s e ales

float input_buffer[11] = {0,0,0,0,0,0,0,0,0,0,0};

float output_buffer[11] = {0,0,0,0,0,0,0,0,0,0,0};

float input_signal = 0;

float output_signal = 0;


Int16 sec, msec;
Uint16 RpReadBuff[BUFFER_MAX_SIZE/2];
Uint16 RpWritedBuff[BUFFER_MAX_SIZE/2];
Uint16 LpReadBuff[BUFFER_MAX_SIZE/2];
Uint16 LpWritedBuff[BUFFER_MAX_SIZE/2];


```

```
62 double Muestras[BUFFER_MAX_SIZE/2];
63 double Hop[Solapamiento];
64 double Ventana[BUFFER_MAX_SIZE/2];
65 double PreProcesSamples[1024];
66 double Voice_Mfcc[78][13];
67 Uint32 Num_Windows;
68
69 Uint16 preprocess_sample(Uint16 Addr);
70 double apply_filter(double input);
71 Int16 aic3204_listen( );
72 Int16 CSL_gptIntr(void);
73 interrupt void gpt0Isr(void);
74 extern void VECSTART(void);
75 void pre_emphasis();
76
77 ///////FFT
78
79 void fft(double input[N]);
80 void Suma_comp(Uint16 salto);
81 void Mult_w(Uint16 salto);
82 void Suma_Real();
83 void fill_w();
84 Uint16 inverso(Uint16 val);
85
86 float X[N];
87 float W_R[N], W_I[N];
88 float Y_R[N], Y_I[N];
89 float output_real[N];
90 float output_imag[N];
91 double power_spectrum[N/2];
92
93 ///////MFCC
94
95
96
97 #define N_MEL_FILTERS 20      //  N mero  de filtros Mel
98 #define N_CEPS 13             // Coeficientes MFCC a extraer
99
100 void compute_mfcc();
101 void init_mel_filter_bank(float Fs);
102 void apply_mel_filters();
103 void compute_dct(const float log_energy[N_MEL_FILTERS]);
104 void generar_hamming(double hammingWindow[512]);
105 void Save_plantilla();
106
107
108
109 float mel_filters[N_MEL_FILTERS][N/2]; // Banco de filtros Mel (precalculado)
110 float mfcc[N_CEPS];           // Coeficientes MFCC finales
111 float log_energy[N_MEL_FILTERS];
112
113 ///////
114
```

```
115
116
117 #define SILENCE_THRESHOLD   165
118 #define WINDOW_SIZE         128
119 #define MIN_SILENCE_WINDOWS 4
120
121 //prueba
122
123 Int16 prueba[512]={0};
124 Int16 prueba2[512]={0};
125 Uint16 flag=0;
126 extern CSL_MmcsdHandle  mmcsdHandle;
127 CSL_Handle    hGpt;
128 Int16 Hit = 0;
129 Int16  data1, data2;
130 Int16 ciclos=0;
131 Int16 ciclos2=0;
132
133 Int16 aic3204_listen( )
134 {
135
136     Int16 sec, msec, sample;
137     Uint16 i = 0;
138     Uint32 Addr = 0;
139
140     CSL_gptIntr();
141
142     /* Play Loop for 5 seconds */
143
144     for ( sec = 0 ; sec < Time ; sec++ )
145     {
146         for ( msec = 0 ; msec < 1000 ; msec++ )
147         {
148             for ( sample = 0 ; sample < 16; sample++ )
149             {
150                 while(Hit!=1)
151                 {
152
153                 }
154
155                 Hit=0;
156
157                 /* Write 16-bit left channel Data */
158
159                 EZDSP5535_I2S_writeLeft(data1);
160                 EZDSP5535_I2S_writeRight(data1);
161
162                 RpWritedBuff[i]=(Uint16)(((Int32)data1)+32768);
163                 i++;
164
165                 if(i>=512)
166                 {
167                     MMC_write(mmcsdHandle, Addr*512, BUFFER_MAX_SIZE, RpWritedBuff);
```

```
168
169                     Addr++;
170                     i=0;
171                 }
172             }
173         }
174     }
175
176     Addr=preprocess_sample(Addr);
177
178     Uint32 k;
179     for(k=0;k<Addr;k++)
180     {
181         MMC_read(mmcsdHandle, (k+200)*512, BUFFER_MAX_SIZE, RpReadBuff);
182
183         for(i=0;i<512;i++)
184         {
185             Int16 processed_sample = (Int16)(((Int32)RpReadBuff[i])-32768);
186             // Escribir muestra procesada al DAC
187             while(Hit!=1)
188             {
189
190             }
191             Hit=0;
192
193             EZDSP5535_I2S_writeLeft(processed_sample);
194             EZDSP5535_I2S_writeRight(processed_sample);
195
196         }
197
198     }
199
200     IRQ_globalDisable();
201     IRQ_clearAll();
202     IRQ_disableAll();
203     GPT_stop(hGpt);
204     GPT_reset(hGpt);
205     GPT_close(hGpt);
206
207     return 0;
208
209 }
210
211 Uint16 por_saber;
212
213 Uint16 preprocess_sample(Uint16 Addr)
214 {
215
216     double hammingWindow[512];
217     Uint32 u = 0, l = 0;
218     Int32 energy = 0;
219     Uint16 ventana = 128;
220     Uint16 inicio = 7;
```

```
221     Uint16 umbral = 200;
222     Uint16 silent_count = 0;
223     Uint16 i,h,t;
224     Uint16 Coeficientes_Index = 0;
225     Uint16 Flag = 1;
226 
227     generar_hamming(hammingWindow);
228 
229     init_mel_filter_bank(16000);
230 
231     for(u = 0; u < 1024; u++)
232         PreProcesSamples[u] = 0.0;
233 
234     for(h=0;h<78;h++)
235     {
236         for(t=0;t<N_CEPS;t++)
237         {
238             Voice_Mfcc[h][t]= 0.0;
239         }
240     }
241 
242     Uint16 N_VENTANAS = 0;
243 
244     for(u = inicio; u < Addr ; u++)
245 
246     {
247 
248         MMC_read(mmcsdHandle, u * 512, BUFFER_MAX_SIZE, RpReadBuff);
249 //        for(i = 0; i < 512; i++)
250 //
251 //            printf("%f,", RpReadBuff[i]);
252 //
253 //        printf("\n");
254         // Calcular energa
255 
256         energy = 0;
257 
258         for(l = 0; l < ventana; l++)
259             energy += abs((Int16)(((Int32)RpReadBuff[l]) - 32768));
260 
261         energy /= ventana;
262 
263 //        if(u<(inicio+5))
264 //        {
265 //            umbral = (energy + umbral)*0.5;
266 //            continue;
267 //        }
268 
269         if (energy < (umbral)) {
270 
271             silent_count++;
272 
273             if (silent_count >= 3)
```

```
274
275                 continue;
276         } else {
277             silent_count = 0;
278         }
279
280         double suma=0.0, suma_cuadrados=0.0;
281
282         for(l = 0; l < 512; l++)
283         {
284             double valor =  (double)((Int16)(((Int32)RpReadBuff[l]) - 32768));
285             Muestras[l] = valor;
286             suma += valor;
287             suma_cuadrados += valor * valor;
288         }
289 //         printf("\n");
290
291                 // Calcular energa
292
293         double media = suma / 512;
294
295         double varianza = (suma_cuadrados / 512) - (media * media);
296
297         double desviacion_estandar = sqrt(varianza);
298
299         double rms = sqrt(suma_cuadrados / 512);
300
301         for(l = 0; l < 512; l++)
302         {
303             //Muestras[l] = valor/rms;//normalizacion rms
304             Muestras[l] = (Muestras[l] - media) / desviacion_estandar; //normalizacion z-score
305 //             printf("%f,", Muestras[l]);
306         }
307 //         printf("\n");
308
309         pre_emphasis();
310
311         if(Flag == 1)
312         {
313             for(i = 0; i < 512; i++)     Ventana[i] = Muestras[i] * hammingWindow[i];
314
315 //             for(l = 0; l < 512; l++)
316 //             {
317 //                 printf("%f,", Ventana[l]);
318 //             }
319 //             printf("\n");
320
321             fft(Ventana);
322             compute_mfcc();
323
324             for(i = 0; i < N_CEPS; i++)
325             {
326 //                 printf("%f,", mfcc[i]);
```

```
327                 Voice_Mfcc[N_VENTANAS][i] = mfcc[i];
328                 PreProcesSamples[Coeficientes_Index++] = mfcc[i];
329             }
330 
331             N_VENTANAS++;
332 
333             // Guardar mitad final para siguiente solapamiento
334 
335             for(i = Solapamiento; i < 512; i++) Hop[i - Solapamiento] = Muestras[i];
336 
337             Flag = 0;
338 
339         }
340 
341         else
342 
343         {
344 
345             // Ventana solapada: Hop + nueva mitad
346 
347             for(i = 0; i < 512; i++)
348 
349             {
350                 if(i < Solapamiento)
351                     Ventana[i] = Hop[i] * hammingWindow[i];
352                 else
353                     Ventana[i] = Muestras[i - Solapamiento] * hammingWindow[i];
354             }
355 
356 //            for(l = 0; l < 512; l++)printf("%f,", Ventana[l]);
357 //            printf("\n");
358 
359             fft(Ventana);
360             compute_mfcc();
361 
362             for(i = 0; i < N_CEPS; i++) {
363 //                printf("%f,", mfcc[i]);
364                 Voice_Mfcc[N_VENTANAS][i] = mfcc[i];
365                 PreProcesSamples[Coeficientes_Index++] = mfcc[i];
366             }
367 
368             N_VENTANAS++;
369 
370             //Ventana completa actual
371 
372             for(i = 0; i < 512; i++)     Ventana[i] = Muestras[i] * hammingWindow[i];
373 //            for(l = 0; l < 512; l++)printf("%f,", Ventana[l]);
374 //            printf("\n");
375 
376             fft(Ventana);
377             compute_mfcc();
378 
379             for(i = 0; i < N_CEPS; i++)
```

```
380             {
381 //                printf("%f,", mfcc[i]);
382                 Voice_Mfcc[N_VENTANAS][i] = mfcc[i];
383                 PreProcesSamples[Coeficientes_Index++] = mfcc[i];
384             }
385 
386             N_VENTANAS++;
387 
388             // Guardar mitad final para siguiente bloque
389 
390             for(i = Solapamiento; i < 512; i++)  Hop[i - Solapamiento] = Muestras[i];
391 
392         }
393 
394     }
395 
396     Num_Windows = N_VENTANAS;
397     //Save_plantilla();
398 
399 //    for(h=0;h<78;h++)
400 //    {
401 //        printf(",");
402 //        printf("{");
403 //        for(t=0;t<N_CEPS;t++)
404 //        {
405 //            if(t<12)
406 //                printf("%f, ",Voice_Mfcc[h][t]);
407 //            else
408 //                printf("%f ",Voice_Mfcc[h][t]);
409 //        }
410 //        printf("}");
411 //    }
412 
413     // copiar python
414 //    printf("El numero de ventanas es %hu \n", N_VENTANAS);
415 //    printf("[\n");
416 //    for (h = 0; h < N_VENTANAS; h++) {
417 //        printf("  [");
418 //        for (t = 0; t < N_CEPS; t++) {
419 //            if (t < N_CEPS - 1)
420 //                printf("%f, ", Voice_Mfcc[h][t]);
421 //            else
422 //                printf("%f", Voice_Mfcc[h][t]);
423 //        }
424 //        if (h < 78 - 1)
425 //            printf("],\n");
426 //        else
427 //            printf("]\n");
428 //    }
429 //    printf("]\n");
430 
431     return 0;
432 
```

```
433 }
434
435 void double_to_uint16_array(double input, Uint16 output[2]) {
436     union {
437         double d;
438         Uint16 arr[2];
439     } u;
440     u.d = input;
441     output[0] = u.arr[0];
442     output[1] = u.arr[1];
443 }
444
445 double uint16_array_to_double2(const Uint16 input[2]) {
446     union {
447         double d;
448         Uint16 arr[2];
449     } u;
450     u.arr[0] = input[0];
451     u.arr[1] = input[1];
452     return u.d;
453 }
454
455 void Save_plantilla()
456 {
457     Uint32 index, j;
458     Uint32 sector = 174;
459     Uint16 array[2];
460     Uint32 buff_index = 0;
461
462     for(index = 0; index < 1024; index++)
463     {
464         printf("%f ",PreProcesSamples[index]);
465         double_to_uint16_array(PreProcesSamples[index], array);
466         for(j = 0; j < 2; j++)
467         {
468             RpWritedBuff[buff_index++] = array[j];
469         }
470         if(buff_index >= 512)
471         {
472             MMC_write(mmcsdHandle, sector * 512, BUFFER_MAX_SIZE, RpWritedBuff);
473             sector++;
474             buff_index = 0;
475         }
476     }
477     printf("Comando Guardado");
478 }
479
480
481
482 void generar_hamming(double hammingWindow[512])
483
484 {
485
```

```
    Uint16 i;

    for (i = 0; i < 512; i++)

    {

        hammingWindow[i] = 0.54 - 0.46 * cos((2 * M_PI * i) / (512 - 1));

    }

}



void pre_emphasis() {

    double Alpha = 0.97;

    double anterior = Muestras[0];

    double actual;

    int i;

    for (i = 1; i < 512; i++) {

        actual = Muestras[i];

        Muestras[i] = Muestras[i] - Alpha * anterior;

        anterior = actual;

    }

}




double apply_filter(double input) {

    // Shift buffers

    int i;

    for (i = FILTER_ORDER; i > 0; i--) {

        input_buffer[i] = input_buffer[i - 1];

        output_buffer[i] = output_buffer[i - 1];

    }
```

```
539
540
541
542     // Agregar nueva entrada
543
544     input_buffer[0] = input;
545
546
547
548     // Calcular salida del filtro
549
550     double output = 0.0;
551
552     for (i = 0; i <= FILTER_ORDER; i++) {
553
554         output += cn[i] * input_buffer[i];
555
556         if (i > 0) {
557
558             output -= cd[i] * output_buffer[i];
559
560         }
561
562     }
563
564
565
566     // Guardar salida actual
567
568     output_buffer[0] = output;
569
570
571
572     return output;
573
574 }
575
576
577
578 ////////////////////////////////////////FFT
579
580
581
582 void fft(double input[N]) {
583
584     Uint16 i,salto;
585
586     // Aplicar bit-reverso al arreglo de entrada
587
588     for ( i = 0; i < N; i++) {
589
590         X[inverso(i)] = input[i];
591
```

```
592     }
593
594
595
596     // Calcular factores twiddle
597
598     fill_w();
599
600
601
602     // Primera etapa de la FFT
603
604     Suma_Real();
605
606
607
608     // Etapas posteriores de la FFT
609
610     for ( salto = 4; salto <= N; salto *= 2) {
611
612         Mult_w(salto);
613
614         Suma_comp(salto);
615
616     }
617
618
619
620     // Normalizar los resultados
621
622     for ( i = 0; i < N; i++) {
623
624         output_real[i] = Y_R[i] / N;
625
626         output_imag[i] = Y_I[i] / N;
627
628     }
629
630     for ( i = 0; i < N/2; i++) {
631
632         power_spectrum[i]=(output_real[i]*output_real[i]) + (output_imag[i]*output_imag[i]);
633 //         printf("%f, ", power_spectrum[i]);
634     }
635
636 }
637
638
639
640 Uint16 inverso(Uint16 val) {
641
642     Uint16 inv = 0;
643
644     Uint16 M1, M2;
```

```
645
646     for (M1 = 1, M2 = (N >> 1); M2 > 0; M1 <<= 1, M2 >>= 1) {
647
648         if (val & M1) {
649
650             inv |= M2;
651
652         }
653
654     }
655
656     return inv;
657
658 }
659
660
661
662 void fill_w() {
663
664     Uint16 i;
665
666     for (i = 0; i < N; i++) {
667
668         float angle = 2 * M_PI * i / N;
669
670         W_R[i] = cos(angle);
671
672         W_I[i] = -sin(angle);
673
674     }
675
676 }
677
678
679
680 void Suma_Real() {
681
682     Uint16 i;
683
684     for ( i = 0; i < N; i += 2) {
685
686         Y_R[i] = X[i] + X[i + 1];
687
688         Y_R[i + 1] = X[i] - X[i + 1];
689
690         Y_I[i] = 0.0;
691
692         Y_I[i + 1] = 0.0;
693
694     }
695
696 }
697
```

```
698
699
700 void Mult_w(Uint16 salto) {
701
702     Uint16 s = salto >> 1;
703
704     Uint16 j, k, r;
705
706     for ( j = s; j < N; j += 2 * s) {
707
708         for ( k = 0; k < s; k++) {
709
710             r = (N * k) / salto;
711
712             float tmp1 = Y_R[j + k] * W_R[r] - Y_I[j + k] * W_I[r];
713
714             float tmp2 = Y_R[j + k] * W_I[r] + Y_I[j + k] * W_R[r];
715
716             Y_R[j + k] = tmp1;
717
718             Y_I[j + k] = tmp2;
719
720         }
721
722     }
723
724 }
725
726
727
728 void Suma_comp(Uint16 salto) {
729
730     Uint16 s = salto >> 1;
731
732     Uint16 k,m,j,i;
733
734     for ( i = 0; i < N; i += salto) {
735
736         for ( m = i, k = 0; k < s; k++, m++) {
737
738             j = m + s;
739
740             float tmp1 = Y_R[m] + Y_R[j];
741
742             float tmp2 = Y_I[m] + Y_I[j];
743
744             Y_R[j] = Y_R[m] - Y_R[j];
745
746             Y_I[j] = Y_I[m] - Y_I[j];
747
748             Y_R[m] = tmp1;
749
750             Y_I[m] = tmp2;
```

```
751
752         }
753
754     }
755
756 }
757
758
759
760 ////////////////////////////////////MFCC
761
762
763
764 void compute_mfcc() {
765
766     apply_mel_filters(power_spectrum, log_energy);
767
768     compute_dct(log_energy);
769
770 }
771
772
773
774 void init_mel_filter_bank(float Fs) {
775
776     float Nyquist = Fs / 2.0;
777
778     float mel_low = 0.0;
779
780     float mel_high = 2595.0 * log10f(1.0 + Nyquist / 700.0);  // Convertir Hz a Mel 2840
781
782     Uint16 i;
783
784
785
786     // Puntos equidistantes en escala Mel
787
788     float mel_points[N_MEL_FILTERS + 2];
789
790     for (i = 0; i < N_MEL_FILTERS + 2; i++) {
791
792         mel_points[i] = mel_low + (mel_high - mel_low) * i / (N_MEL_FILTERS + 1);
793
794     }
795
796
797
798     // Convertir Mel a Hz
799
800     float hz_points[N_MEL_FILTERS + 2];
801
802     for (i = 0; i < N_MEL_FILTERS + 2; i++) {
803
```

```
804         hz_points[i] = 700.0 * (powf(10.0, mel_points[i] / 2595.0) - 1.0);
805 
806     }
807 
808 
809 
810     // Crear filtros triangulares
811 
812     for (i = 0; i < N_MEL_FILTERS; i++) {
813 
814         float left = hz_points[i];
815 
816         float center = hz_points[i + 1];
817 
818         float right = hz_points[i + 2];
819 
820         Uint16 bin;
821 
822 
823 
824         for (bin = 0; bin < N/2; bin++) {
825 
826             float freq_bin = (bin * Fs) / N;
827 
828             if (freq_bin < left || freq_bin > right) {
829 
830                 mel_filters[i][bin] = 0.0;
831 
832             } else if (freq_bin <= center) {
833 
834                 mel_filters[i][bin] = (freq_bin - left) / (center - left);
835 
836             } else {
837 
838                 mel_filters[i][bin] = (right - freq_bin) / (right - center);
839 
840             }
841         }
842 
843     }
844 
845 }
846 
847 
848 
849 void apply_mel_filters() {
850 
851     Uint16 i;
852 
853     for (i = 0; i < N_MEL_FILTERS; i++) {
854 
855         log_energy[i] = 0.0;
856 
```

```
857         double energy = 0.0000000000000000;
858 
859         Uint32 bin;
860 
861         for (bin = 0; bin < N/2; bin++) {
862 
863             energy += power_spectrum[bin] * mel_filters[i][bin];
864 
865         }
866 
867         log_energy[i] = logf(energy+ 1e-6);  // Evitar log(0)
868 
869     }
870 
871 }
872 
873 
874 
875 void compute_dct(const float log_energy[N_MEL_FILTERS]) {
876 
877     Uint16 c;
878 
879     for (c = 0; c < N_CEPS; c++) {
880 
881         mfcc[c] = 0.0;
882 
883         Uint16 j;
884 
885         for (j = 0; j < N_MEL_FILTERS; j++) {
886 
887             mfcc[c] += log_energy[j] * cosf((c * (j + 0.5) * M_PI / N_MEL_FILTERS));
888 
889         }
890 
891     }
892 
893 }
894 
895 
896 
897 ////////////////////////////////Timer
898 
899 
900 
901 Int16 CSL_gptIntr(void)
902 
903 {
904 
905     CSL_Status    status;
906 
907     CSL_Config    hwConfig;
908 
909     CSL_GptObj    gptObj;
```

```
910
911
912
913     status   = 0;
914
915
916
917     hGpt = GPT_open (GPT_0, &gptObj, &status);
918
919     if((NULL == hGpt) || (CSL_SOK != status)) {
920
921         printf("GPT Open Failed\n");
922
923     }
924
925
926
927     status = GPT_reset(hGpt);
928
929     if(CSL_SOK != status) {
930
931         printf("GPT Reset Failed\n");
932
933     }
934
935
936
937     IRQ_clearAll();
938
939     IRQ_disableAll();
940
941     IRQ_setVecs((Uint32)(&VECSTART));
942
943     IRQ_plug(TINT_EVENT, &gpt0Isr);
944
945     IRQ_enable(TINT_EVENT);
946
947
948
949     // Configurar GPT para 16 kHz
950
951     hwConfig.autoLoad     = GPT_AUTO_ENABLE;
952
953     hwConfig.ctrlTim      = GPT_TIMER_ENABLE;
954
955     hwConfig.preScaleDiv  = GPT_PRE_SC_DIV_0; // Prescaler de 2
956
957     hwConfig.prdLow       = (100000 / 32) - 1; // Periodo ajustado
958
959     hwConfig.prdHigh      = 0x0000;
960
961
962
```

```
963     status =  GPT_config(hGpt, &hwConfig);
964 
965     if(CSL_SOK != status) {
966 
967         printf("GPT Config Failed\n");
968 
969     }
970 
971 
972 
973     IRQ_globalEnable();
974 
975     GPT_start(hGpt);
976 
977     return 0;
978 
979 }
980 
981 
982 
983 interrupt void gpt0Isr(void)
984 
985 {
986 
987     GPT_stop(hGpt);
988 
989     Hit=1;
990 
991     /* Read 16-bit left channel Data */
992 
993     EZDSP5535_I2S_readLeft(&data1);
994 
995 
996 
997     IRQ_clear(TINT_EVENT);
998 
999 
1000 
1001     CSL_SYSCTRL_REGS->TIAFR = 0x01;
1002 
1003     GPT_start(hGpt);
1004 
1005 }
```

LISTING 7.2: Preprocesamiento y Extracción de características

7.4. GMM (C)

```
1 #include "stdio.h"
2 #include "ezdsp5535.h"
```

```
#include "ezdsp5535_sar.h"
#include <math.h>
#include <float.h>
#include <string.h>  // Para memcpy

#define DIM 13            // Coeficientes MFCC por ventana
#define MAX_COMP 3       //  M ximo   n mero  de componentes
#define LOG_2PI 1.837877  // log(2*PI)
#define MAX_TEMPLATE_SIZE 78

typedef struct {
    Uint16 n_componentes;
    double pesos[MAX_COMP];
    double medias[MAX_COMP][DIM];
    double covarianzas[MAX_COMP][DIM];
} ModeloGMM;

extern double Voice_Mfcc[MAX_TEMPLATE_SIZE][DIM];

double result;

ModeloGMM Comandos[5];

double Probabilidad_GMM(Uint16 Num_Comando, Uint16 num_ventanas);



void inicializarGMM()
{
    // Inicializacin para Comandos[0] Luz
    Comandos[0].n_componentes = 3;

    double pesos1[3] = {0.3655, 0.4581, 0.1765 };
    memcpy(Comandos[0].pesos, pesos1, sizeof(pesos1));

    double medias1[3][DIM] = {
          { -145.0801, -37.0897, 3.7786, 7.0585, -5.7370, 2.0840, 2.4976, -2.1958, -2.2924, 2.2024, 0.9074, -0.7304, 3.3127 },
          { -157.5847, -6.7434, 1.7608, -1.9055, -1.4887, -3.3281, -2.7673, -6.4725, -4.7070, -2.0830, 0.6700, 1.1405, 0.2584 },
          { -173.8009, -15.8509, 1.5926, 1.6722, 2.5772, 3.4031, 2.1246, 2.4712, 1.0185, 1.1483, 1.2903, 1.4142, 0.7005 },
      };
    memcpy(Comandos[0].medias, medias1, sizeof(medias1));

    double covariancias1[3][DIM] = {
        { 53.7838, 48.6930, 10.2383, 13.6943, 7.9478, 6.4677, 5.0577, 12.3599, 3.8467, 6.4216, 2.1129, 3.9274, 2.2266 },
        { 111.2821, 24.3301, 6.0765, 8.1492, 42.7556, 9.3496, 4.6594, 4.4550, 5.0795, 3.9239, 1.7239, 1.4971, 2.7870 },
        { 157.4753, 8.8189, 1.6769, 2.0971, 2.4716, 2.3107, 1.6181, 2.3207, 1.3563, 1.1944, 1.0820, 1.0626, 1.3096 },
    };
```

```
50    memcpy(Comandos[0].covarianzas, covariancias1, sizeof(covariancias1));
51
52    // Inicializacin para Comandos[1] Calefaccion
53    Comandos[1].n_componentes = 3;
54
55    double pesos2[3] = {0.3014, 0.4676, 0.2310 };
56    memcpy(Comandos[1].pesos, pesos2, sizeof(pesos2));
57
58
59    double medias2[3][DIM] = {
60          { -168.5168, -9.3870, 2.7958, 0.6194, 2.2595, 0.5253, -1.4830, 0.9320, 0.8461,
      0.3154, 0.8198, 0.5469, -0.3068 },
61          { -156.6826, -3.4862, -0.8319, -10.0074, -0.8542, -5.9247, -4.6549, -3.9437, 0.8876,
       0.8571, 2.3384, 0.1545, -0.7204 },
62          { -156.2146, -29.7319, 7.4096, 0.0990, -3.8798, 1.6371, -7.5241, 1.7557, -1.5082,
      -0.6474, 0.7529, 0.0527, 0.0050 },
63        };
64    memcpy(Comandos[1].medias, medias2, sizeof(medias2));
65
66    double covariancias2[3][DIM] = {
67        { 113.5799, 30.8750, 10.5302, 9.5363, 5.1883, 4.8979, 7.8708, 2.6081, 2.2795, 2.6383,
      2.3167, 1.8272, 1.4462 },
68        { 97.1647, 17.6053, 9.5258, 26.0922, 9.6438, 9.8069, 5.0103, 4.1021, 4.6016, 2.8420,
      2.4828, 3.7158, 2.3159 },
69        { 122.0512, 220.2825, 7.8109, 21.3114, 4.4081, 20.2872, 12.7826, 7.4592, 3.1512,
      9.6345, 1.6543, 1.2609, 1.9626 },
70    };
71    memcpy(Comandos[1].covarianzas, covariancias2, sizeof(covariancias2));
72
73    // Inicializacin para Comandos[2] Ventilador
74    Comandos[2].n_componentes = 3;
75
76    double pesos3[3] = {0.2518, 0.3533, 0.3949 };
77    memcpy(Comandos[2].pesos, pesos3, sizeof(pesos3));
78
79    double medias3[3][DIM] = {
80          { -164.8666, -11.2537, 0.9894, -0.2940, 1.9222, -1.1440, -0.6002, 1.5182, 0.5016,
      0.6598, 1.1853, 1.9464, 0.6795 },
81          { -161.6815, -1.1845, 1.2710, -10.8064, -4.6970, -8.5107, -3.1478, -3.9936, 1.5988,
      -0.5260, 3.6765, 0.0843, -1.8618 },
82          { -155.1440, -10.1232, 1.5422, 3.4376, -0.4770, -7.4508, -7.9156, -2.7244, -0.6845,
      -2.1861, -0.6620, 1.3361, -1.0534 },
83        };
84    memcpy(Comandos[2].medias, medias3, sizeof(medias3));
85
86    double covarianzas3[3][DIM] = {
87          { 152.0659, 26.7334, 8.4406, 20.0885, 7.2702, 12.8651, 6.5618, 3.2494, 3.6973,
      4.4235, 2.8102, 2.6540, 1.7468 },
88          { 103.0581, 23.3116, 8.3918, 8.1137, 6.2710, 6.0156, 5.2940, 3.6143, 2.8028,
      4.5510, 2.9357, 3.9646, 6.5833 },
89          { 122.4395, 17.0079, 8.3247, 17.0025, 7.9754, 8.8793, 7.0360, 3.6463, 5.7383,
      3.6235, 2.2032, 1.7757, 2.0538 },
90       };
```

```
91      memcpy(Comandos[2].covarianzas, covarianzas3, sizeof(covarianzas3));
92      // Inicializacin para Comandos[3] Alarma
93      Comandos[3].n_componentes = 3;
94
95      double pesos4[3] = {0.3467, 0.4700, 0.1833 };
96      memcpy(Comandos[3].pesos, pesos4, sizeof(pesos4));
97
98      double medias4[3][DIM] = {
99                               { -171.6021, 0.9096, 5.0474, -2.0948, 2.1004, -3.7219, -2.9520,
        -2.5075, 0.8709, -0.9661, 1.3223, 0.8933, -0.9162 },
100                              { -152.7677, -5.7856, -0.4801, -3.6242, 0.5425, -4.7846,
        -4.8572, -3.6000, 0.3574, -0.5253, 0.3908, 0.8740, -0.9315 },
101                              { -157.9114, -11.5405, -1.9480, -2.1868, 0.3164, 0.0907,
        -0.6086, 1.1544, 1.0059, 1.9159, 1.7631, 1.1941, 0.1778 },
102                         };
103     memcpy(Comandos[3].medias, medias4, sizeof(medias4));
104
105     double covarianzas4[3][DIM] =  {
106                                     { 61.9841, 8.0571, 3.4415, 6.2255, 4.4830, 4.1766, 2.3800,
         2.6465, 2.6387, 2.3173, 2.4663, 1.7732, 1.2909 },
107                                     { 49.8754, 19.5961, 7.3000, 44.6895, 14.7630, 5.8789,
        9.3590, 10.2387, 2.4876, 4.4051, 2.5374, 3.1566, 2.0793 },
108                                     { 164.4528, 8.6377, 4.9585, 4.8651, 3.0416, 2.1084,
        1.7112, 1.7625, 1.1489, 2.0808, 1.2159, 1.2460, 2.0567 },
109                               };
110     memcpy(Comandos[3].covarianzas, covarianzas4, sizeof(covarianzas4));
111
112     // Inicializacin para Comandos[4] Televisor
113     Comandos[4].n_componentes = 3;
114
115     double pesos5[3] = {0.1398, 0.5466, 0.3136 };
116     memcpy(Comandos[4].pesos, pesos5, sizeof(pesos5));
117
118     double medias5[3][DIM] = {
119           { -154.6295, -32.8322, 6.6130, 5.5814, -6.6053, 6.2811, -6.5847, -2.2098, 0.8440,
        -2.5062, 0.0838, -0.3332, -1.7649 },
120           { -149.3892, -7.1691, -1.2474, -5.2177, -1.7105, -5.6835, -5.0730, -2.7644, -0.1204,
         -1.5216, 1.1007, 1.1290, -1.1094 },
121           { -164.2144, -10.5825, 2.0374, 2.9357, 1.3826, -5.0025, -5.9170, -2.2460, -0.9691,
        -1.6064, -0.9568, 0.5716, -0.2580 },
122       };
123     memcpy(Comandos[4].medias, medias5, sizeof(medias5));
124
125     double covarianzas5[3][DIM] = {
126            { 82.4274, 63.0583, 16.1156, 14.8128, 5.2516, 4.8741, 8.8600, 4.9055, 1.8979,
        5.5364, 4.1187, 1.4553, 1.4555 },
127           { 78.6988, 27.0690, 10.9947, 21.2471, 18.0785, 6.0128, 10.1054, 17.8421, 6.1618,
        6.1701, 3.3563, 3.1654, 2.9682 },
128           { 97.3366, 23.7007, 6.1533, 10.6161, 6.1701, 33.9602, 27.8779, 14.8796, 7.0802,
        8.6340, 4.3261, 1.2769, 2.0759 },
129        };
130     memcpy(Comandos[4].covarianzas, covarianzas5, sizeof(covarianzas5));
131
```

```
}


double logsumexp(double logprobs[3], Uint16 n) {
    double max = -DBL_MAX;
    Uint16 i;
    for(i = 0; i < n; i++) {
        if(logprobs[i] > max) max = logprobs[i];
    }

    double sum = 0.0;
    for(i = 0; i < n; i++) {
        sum += exp(logprobs[i] - max);
    }

    return max + log(sum);
}

double log_probabilidad_ventana(ModeloGMM modelo, double ventana[DIM]) {
    double log_probs[MAX_COMP];
    Uint16 i, d;

    for(i = 0; i < modelo.n_componentes; i++) {
        double log_peso = log(modelo.pesos[i]);
        double termino_mahal = 0.0;
        double termino_cov = 0.0;

        for(d = 0; d < DIM; d++) {
            double diff = ventana[d] - modelo.medias[i][d];
            termino_mahal += (diff * diff) / modelo.covarianzas[i][d];
            termino_cov += log(modelo.covarianzas[i][d]);
        }

        log_probs[i] = log_peso - 0.5 * (termino_mahal + termino_cov + DIM * LOG_2PI);
    }

    return logsumexp(log_probs, modelo.n_componentes);
}

double Probabilidad_GMM(Uint16 Num_Comando, Uint16 num_ventanas)
{
    if(num_ventanas == 0) result = -DBL_MAX;

    double log_prob_total = 0.0;
    Uint16 v;

    for(v = 0; v < num_ventanas; v++) {
        log_prob_total += log_probabilidad_ventana(Comandos[Num_Comando], Voice_Mfcc[v]);
    }

    return log_prob_total;
}
```

LISTING 7.3: GMM

7.5. DTW (C)

```c
#include "stdio.h"
#include <stdlib.h>
#include <float.h>
#include <math.h>
#include <stdint.h>

#define MAX_TEMPLATE_SIZE 78
#define MAX_MFCC_COEFS 13

// Matrices externas
extern double Voice_Mfcc[MAX_TEMPLATE_SIZE][MAX_MFCC_COEFS];
double Templest[5][MAX_TEMPLATE_SIZE][MAX_MFCC_COEFS];
double result;

// Funcin para calcular la distancia euclidiana
double euclidean_distance(double vec1[MAX_MFCC_COEFS], double vec2[MAX_MFCC_COEFS], double
    size)
{
    double sum = 0.0;
    double i;
    for (i = 0; i < size; i++) {
        double diff = vec1[i] - vec2[i];
        sum += diff * diff;
    }
    return sqrt(sum);
}

// Implementacin del algoritmo DTW
double dtw_custom(Uint16 len1, Uint16 len2, Uint16 num_coefs, Uint16 index) {
    double cost[MAX_TEMPLATE_SIZE][MAX_TEMPLATE_SIZE];
    double i, j;
    double dist;

    // Inicializar primera celda
    cost[0][0] = euclidean_distance(Voice_Mfcc[0], Templest[index][0], num_coefs);

    // Inicializar primera columna
    for (i = 1; i < len1; i++) {
        cost[i][0] = cost[i - 1][0] + euclidean_distance(Voice_Mfcc[i], Templest[index][0],
    num_coefs);
    }

    // Inicializar primera fila
    for (j = 1; j < len2; j++) {
```

```
43          cost[0][j] = cost[0][j - 1] + euclidean_distance(Voice_Mfcc[0], Templest[index][j],
    num_coefs);
44      }
45
46      // Llenar el resto de la matriz
47      for (i = 1; i < len1; i++) {
48          for (j = 1; j < len2; j++) {
49              dist = euclidean_distance(Voice_Mfcc[i], Templest[index][j], num_coefs);
50
51              // Buscar el  m nimo  sin usar fmin
52              double min_cost = cost[i - 1][j];
53              if (cost[i][j - 1] < min_cost) {
54                  min_cost = cost[i][j - 1];
55              }
56              if (cost[i - 1][j - 1] < min_cost) {
57                  min_cost = cost[i - 1][j - 1];
58              }
59
60              cost[i][j] = dist + min_cost;
61          }
62      }
63      return cost[len1 - 1][len2 - 1];
64  }
```

LISTING 7.4: DTW

7.6. Distancia Euclidiana con CMVN (C)

```
1  #include "stdio.h"
2  #include <stdlib.h>
3  #include <float.h>
4  #include <math.h>
5  #include <stdint.h>
6
7  #define MAX_TEMPLATE_SIZE 78
8  #define MAX_MFCC_COEFS 13
9
10 // Matrices externas
11 extern double Voice_Mfcc[MAX_TEMPLATE_SIZE][MAX_MFCC_COEFS];
12 double Templest[5][MAX_TEMPLATE_SIZE][MAX_MFCC_COEFS];
13 double result;
14
15 double distancia(Uint32 comando);
16
17 #define S 2  // desplazamiento  m ximo
18
19 double distancia(uint32_t comando)
20 {
21     // Constantes de  t a m a o
22     Int16 Ncoef = MAX_MFCC_COEFS;
```

```
23      Int16 Nwin  = MAX_TEMPLATE_SIZE;
24
25      // Arrays para CMVN
26      double mean_v[MAX_MFCC_COEFS];
27      double std_v [MAX_MFCC_COEFS];
28      double mean_t[MAX_MFCC_COEFS];
29      double std_t [MAX_MFCC_COEFS];
30
31      // Variables auxiliares
32      Int16 c = 0, w = 0;
33      double sum = 0.0;
34
35      // 1) Calcular mean_v y mean_t por coeficiente
36      for (c = 0; c < Ncoef; c++) {
37          sum = 0.0;
38          for (w = 0; w < Nwin; w++) {
39              sum += (double)Voice_Mfcc[w][c];
40          }
41          mean_v[c] = sum / Nwin;
42
43          sum = 0.0;
44          for (w = 0; w < Nwin; w++) {
45              sum += (double)Templest[comando][w][c];
46          }
47          mean_t[c] = sum / Nwin;
48      }
49
50      // 2) Calcular std_v y std_t por coeficiente
51      for (c = 0; c < Ncoef; c++) {
52          sum = 0.0;
53          for (w = 0; w < Nwin; w++) {
54              double dv = (double)Voice_Mfcc[w][c] - mean_v[c];
55              sum += dv * dv;
56          }
57          std_v[c] = sqrt(sum / Nwin);
58
59          sum = 0.0;
60          for (w = 0; w < Nwin; w++) {
61              double dt = (double)Templest[comando][w][c] - mean_t[c];
62              sum += dt * dt;
63          }
64          std_t[c] = sqrt(sum / Nwin);
65      }
66
67      // 3) Comparacin con desplazamiento (shift) y CMVN
68      double best = 1e12;
69      Int16 shift = 0;
70      Int16 iv = 0;
71      Int16 count = 0;
72      double suma_sq = 0.0;
73      double v_norm = 0.0, t_norm = 0.0, diff = 0.0, avg = 0.0;
74
75      for (shift = -S; shift <= S; shift++) {
```

```
76          suma_sq = 0.0;
77          count   = 0;
78          for (w = 0; w < Nwin; w++) {
79              iv = w + shift;
80              if (iv < 0 || iv >= Nwin) continue;
81              for (c = 0; c < Ncoef; c++) {
82                  // CMVN + suavizado de divisor
83                  v_norm = ((double)Voice_Mfcc[iv][c] - mean_v[c]) / (std_v[c] + 1e-6);
84                  t_norm = ((double)Templest[comando][w][c] - mean_t[c]) / (std_t[c] + 1e-6);
85                  diff   = v_norm - t_norm;
86                  suma_sq += diff * diff;
87                  count++;
88              }
89          }
90          if (count > 0) {
91              avg = suma_sq / count;
92              if (avg < best) {
93                  best = avg;
94              }
95          }
96      }
97
98      return sqrt(best);
99  }
```

LISTING 7.5: Distancia Euclidiana con CMVN

7.7. Main (C)

```
1  #include "stdio.h"
2  #include "ezdsp5535.h"
3  #include <csl_mmcsd.h>
4  #include "ezdsp5535_sar.h"
5  #include <float.h>
6  #include <math.h>
7  #include "ezdsp5535_uart.h"
8  #include "csl_gpio.h"
9  #include "ezdsp5535_gpio.h"
10
11 #define MAX_TEMPLATE_SIZE 78   /* T a m a o   m ximo  de plantilla (ventanas) */
12 #define MAX_MFCC_COEFS 13      /*  N mero   m ximo  de coeficientes MFCC por ventana */
13 #define MAX_TEMPLATES 20       /*  N mero   m ximo  de plantillas (ejemplos) */
14 #define BUFFER_MAX_SIZE    (1024u)
15 #define MAX_MFCC_COEFS 13
16
17 typedef struct {
18     Uint32 sector_location;   // Sector donde  e s t n  guardados los MFCCs en SD
19     int label;                // Etiqueta del comando
20     char name[32];            // Nombre descriptivo
21 } Template;
```

```
22
23 static Template templates[MAX_TEMPLATES];
24 static int num_templates = 0;
25 extern CSL_MmcsdHandle  mmcsdHandle;
26 Uint16 RReadBuff[BUFFER_MAX_SIZE/2];
27 double tempBuff[BUFFER_MAX_SIZE];
28
29 double Mfcc_voice[MAX_TEMPLATE_SIZE][MAX_MFCC_COEFS];
30 double Mfcc_template[MAX_TEMPLATE_SIZE][MAX_MFCC_COEFS];
31 Uint16 command;
32
33 extern Int16 Configuracion( );
34 extern void Close();
35 extern Int16 aic3204_listen( );
36 extern Int16 recognize_voice_command( Uint32 audio_sector);
37 extern void init_templates();
38
39 double uint16ToDouble(Uint16 value);
40 Uint16 extract_mfcc_from_sd(Uint32 sector_start, double output_buffer[MAX_TEMPLATE_SIZE][
       MAX_MFCC_COEFS]);
41 Int16 classify_command();
42 void init_templates(void);
43 double uint16_array_to_double(const Uint16 input[2]);
44 void LecturaMemoria(Uint32 sector_start);
45
46 extern double Voice_Mfcc[78][13];
47 extern Uint32 Num_Windows;
48 extern double Templest[5][78][13];
49 extern double result;
50
51 ////////////////////////////////////////////////////////////////////////
52 //extern dtw_custom(Uint16 len1, Uint16 len2, Uint16 num_coefs, Uint16 i);
53 //extern double distancia(Uint32 comando);
54 extern void inicializarGMM();
55 extern double Probabilidad_GMM(Uint16 Num_Comando, Uint16 num_ventanas);
56 ////////////////////////////////////////////////////////////////////////
57
58
59 void main( void )
60   {
61     /* Initialize BSL */
62     EZDSP5535_init( );
63     if(EZDSP5535_SAR_init())
64     {
65         printf("Error al inicializar el  m dulo  SAR\n");
66         return;
67     }
68     Configuracion();
69     init_templates();
70     inicializarGMM();
71
72     printf("Presione el SW1 para escuchar comando\n");
73     while(1)
```

```
74      {
75          if(EZDSP5535_SAR_getKey() == SW1)
76          {
77              aic3204_listen( );
78              printf("Pensando\n");
79              command = classify_command();
80              EZDSP5535_UART_open();
81              switch(command)
82              {
83              case 0:
84                  printf("Luz");
85                  EVM5515_UART_putChar('L');    // Write L
86                  break;
87              case 1:
88                  printf("Calefaccion");
89                  EVM5515_UART_putChar('C');    // Write C
90                  break;
91              case 2:
92                  printf("Ventilador");
93                  EVM5515_UART_putChar('V');    // Write V
94                  break;
95              case 3:
96                  printf("Fuente");
97                  EVM5515_UART_putChar('A');    // Write A
98                  break;
99              case 4:
100                 printf("Televisor");
101                 EVM5515_UART_putChar('T');    // Write T
102                 break;
103             }
104             printf("\nPresione el SW1 para escuchar comando\n");
105             EZDSP5535_init( );
106             EZDSP5535_SAR_init();
107             Configuracion();
108         }
109     }
110 }
111
112 Int16 classify_command()
113 {
114     double similitud;
115     int best_match = -1;
116     Uint32 i;
117
118     Uint16 WindowsVoice = Num_Windows;
119
120 //    similitud = DBL_MAX;// DTW y distance
121     similitud = -DBL_MAX;//GMM
122
123     printf("");
124
125     for (i = 0; i < num_templates; i++)
126     {
```

```
127 ///////////////distance
128
129 //         double valor = distancia(i);
130 //
131 //         if (valor < similitud) {
132 //             similitud = valor;
133 //             best_match = templates[i].label;
134 //         }
135
136 ////////////dtw
137
138 //         double valor = dtw_custom(WindowsVoice, templates[i].sector_location, 13, i);
139 //
140 //         /* Si la distancia es menor que un umbral, considerar como coincidencia */
141 //         if (valor < similitud) {
142 //             similitud = valor;
143 //             best_match = templates[i].label;
144 //         }
145
146 ///////////////GMM
147
148         double valor =Probabilidad_GMM( i, WindowsVoice);
149
150         if (valor > similitud) {
151             similitud = valor;
152             best_match = templates[i].label;
153         }
154
155         printf("", templates[i].name, valor);
156
157     }
158
159     /* Si la mejor distancia es muy alta, p o d r a  no ser  n i n g n  comando conocido */
160 //     if (similitud > 10000.0) { /* Ajustar este umbral  s e g n  tus datos */
161 //         return -1;
162 //     }
163     return best_match;
164 }
165
166 double uint16_array_to_double(const Uint16 input[2]) {
167     union {
168         double d;
169         Uint16 arr[4];
170     } u;
171     u.arr[0] = input[0];
172     u.arr[1] = input[1];
173     u.arr[2] = 0;
174     u.arr[3] = 0;
175     return u.d;
176 }
177
178 void LecturaMemoria(Uint32 sector_start)
179 {
```

```
180     Uint16 array[2],index,i;
181     Uint32 block = sector_start*512;
182 
183     MMC_read(mmcsdHandle, block, 1024, RReadBuff);
184 
185     index = 0;
186 
187     for (i = 0; i < 1024; i++) {
188         array[0] = RReadBuff[(i-index)*2];
189         array[1] = RReadBuff[((i-index)*2)+1];
190         tempBuff[i] = uint16_array_to_double(array);
191         printf("%f ", tempBuff[i]);
192         if((i % 256) == 0 && i > 0) {
193             sector_start++;
194             block = sector_start*512;
195             MMC_read(mmcsdHandle, block, 1024, RReadBuff);
196             index += 256;
197         }
198     }
199     printf("Lectura Completada");
200 }
201 
202 Uint16 extract_mfcc_from_sd(Uint32 sector_start, double output_buffer[MAX_TEMPLATE_SIZE][
        MAX_MFCC_COEFS]) {
203     Uint16 numValidWindows = 0;
204     Uint32 i, j, pos;
205 
206     LecturaMemoria(sector_start);
207 
208     int maxWindows = BUFFER_MAX_SIZE / MAX_MFCC_COEFS;
209     if (maxWindows > MAX_TEMPLATE_SIZE) maxWindows = MAX_TEMPLATE_SIZE;
210 
211     pos = 0;
212     for (i = 0; i < maxWindows; i++)
213     {
214         int nonZeroValues = 0;
215         for (j = 0; j < MAX_MFCC_COEFS && pos + j < BUFFER_MAX_SIZE; j++) {
216             if (tempBuff[pos + j] != 0.0) nonZeroValues++;
217         }
218         if (nonZeroValues == 0) break;
219 
220         for (j = 0; j < MAX_MFCC_COEFS && pos + j < BUFFER_MAX_SIZE; j++) {
221             output_buffer[i][j] = tempBuff[pos + j];
222         }
223 
224         numValidWindows++;
225         pos += MAX_MFCC_COEFS;
226     }
227 
228     return numValidWindows;
229 }
230 
231 void init_templates() {
```

```
232     // Plantilla 1 - Sector 300 y 301
233     Template plantilla1 = {25, 0, "Luz"};
234 
235 
236     // Plantilla 2 - Sector 302 y 303
237     Template plantilla2 = {39, 1, "Calefaccion"};
238 
239     Template plantilla3 = {41, 2, "Ventilador"};
240 
241     Template plantilla4 = {35, 3, "Alarma"};
242 
243     Template plantilla5 = {39, 4, "Televisor"};
244 
245 
246     templates[num_templates++] = plantilla1;
247     templates[num_templates++] = plantilla2;
248     templates[num_templates++] = plantilla3;
249     templates[num_templates++] = plantilla4;
250     templates[num_templates++] = plantilla5;
251 }
252 
253 double uint16ToDouble(Uint16 value) {
254     /* Implementacin especfica segn el formato de datos */
255     double result;
256 
257     union {
258         Uint16 uint16Value;
259         double doubleValue;
260     } converter;
261 
262     converter.uint16Value = value;
263     result = converter.doubleValue;
264 
265     return result;
266 }
```

LISTING 7.6: Main

7.8. Comunicación Con Electrodomésticos (C)

```
1  void setup() {
2      Serial.begin(115200);
3  
4      // LED integrado
5      pinMode(13, OUTPUT);
6  
7      // Salidas asociadas a cada letra
8      pinMode(3, OUTPUT); // Luz
9      pinMode(4, OUTPUT); // Calefacci n
10     pinMode(5, OUTPUT); // Ventilador
```

```
11    pinMode(6, OUTPUT); // Alarma
12    pinMode(7, OUTPUT); // Televisor
13
14    // Aseguramos que todas las salidas empiecen apagadas
15    digitalWrite(3, HIGH);
16    digitalWrite(4, HIGH);
17    digitalWrite(5, HIGH);
18    digitalWrite(6, HIGH);
19    digitalWrite(7, HIGH);
20
21    Serial.println("Arduino listo...");
22  }
23
24  void loop() {
25    if (Serial.available()) {
26      String msg = Serial.readStringUntil('\n');
27      msg.trim();  // Elimina espacios en blanco y saltos de l nea
28
29      if (msg.length() > 0) {
30        Serial.println("Recibido: " + msg);
31        digitalWrite(13, HIGH); // LED indicador de actividad
32
33        char c = msg.charAt(0); // Toma la primera letra
34        int pin = -1;
35
36        if (c == 'L') pin = 3; // Luz
37        else if (c == 'C') pin = 4; // Calefacci n
38        else if (c == 'V') pin = 5; // Ventilador
39        else if (c == 'A') pin = 6; // Alarma
40        else if (c == 'T') pin = 7; // Televisor
41
42        if (pin != -1) {
43          // Lee estado actual y cambia al opuesto
44          int currentState = digitalRead(pin);
45          digitalWrite(pin, !currentState);
46        }
47
48        delay(200);  // Breve pausa visual para el LED
49        digitalWrite(13, LOW); // Apaga LED indicador
50      }
51    }
52  }
```

LISTING 7.7: Comunicación Con Electrodomésticos

7.9. Entrenamiento de GMM (Python)

```
1 import numpy as np
2 from sklearn.mixture import GaussianMixture
3
```

```
# Supongamos que tienes varias grabaciones, cada una con sus propios MFCCs.
# Cada grabaci n es una lista de vectores MFCC (cada vector puede representar un frame de audio).
# Ejemplo para dos grabaciones:
grabacion1 =
grabacion2 =
grabacion3 =
grabacion4 =
grabacion5 =
grabacion6 =
grabacion7 =
grabacion8 =
grabacion9 =
grabacion10 =

# Combinar todas las grabaciones en una sola lista
mfccs = grabacion1 + grabacion2 + grabacion3 + grabacion4 + grabacion5 + grabacion6 + grabacion7 + grabacion8 + grabacion9 + grabacion10

# Convertir a array de numpy
mfccs = np.array(mfccs)

# Entrenar GMM con 2 componentes (por ejemplo)
gmm = GaussianMixture(n_components=3, covariance_type='diag')
gmm.fit(mfccs)

# Funci n para generar el c digo C con los par metros del GMM
def print_gmm_for_c(gmm, num_coeffs):
    print("// Par metros GMM en C")
    print(f"#define NUM_COMPONENTS {gmm.n_components}")
    print(f"#define NUM_COEFFS {num_coeffs}\n")

    # Pesos
    print("float weights[] = {", end="")
    print(", ".join(f"{w:.4f}" for w in gmm.weights_), "};\n")

    # Medias
    print("float means[][NUM_COEFFS] = {")
    for mean in gmm.means_:
        print("    {", ", ".join(f"{m:.4f}" for m in mean), "},")
    print("};\n")

    # Covarianzas
    print("float covariances[][NUM_COEFFS] = {")
    for cov in gmm.covariances_:
        print("    {", ", ".join(f"{c:.4f}" for c in cov), "},")
    print("};")

# Generar el c digo C utilizando 13 coeficientes (por ejemplo)
print_gmm_for_c(gmm, num_coeffs=13)
```

LISTING 7.8: Entrenamiento de GMM (Python)

7.10. Comunicación entre puertos (Python)

```python
import serial
import serial.tools.list_ports

def detectar_puertos_disponibles():
    disponibles = []
    for p in serial.tools.list_ports.comports():
        try:
            s = serial.Serial(p.device)
            s.close()
            disponibles.append((p.device, p.description))
        except:
            pass  # No se puede abrir, est  ocupado
    return disponibles

# === Listar puertos disponibles ===
puertos = detectar_puertos_disponibles()

if not puertos:
    print("       No hay puertos disponibles.")
    exit()

for idx, (device, desc) in enumerate(puertos):
    print(f"[{idx}] {device} - {desc}")

# === Selecci n de puertos ===
idx_dsp = int(input("Selecciona  ndice  del puerto para la DSP: "))
idx_arduino = int(input("Selecciona  ndice  del puerto para el Arduino: "))

try:
    dsp = serial.Serial(puertos[idx_dsp][0], baudrate=115200, timeout=1)
    arduino = serial.Serial(puertos[idx_arduino][0], baudrate=115200, timeout=1)
    print("    Conexi n exitosa. Escuchando...")

    while True:
        # DSP     Arduino
        if dsp.in_waiting > 0:
            data = dsp.readline().decode('utf-8', errors='ignore').strip()
            if data:
                print(f"[DSP] {data}")
                arduino.write((data + '\n').encode('utf-8'))

        # Arduino     PC (monitor serial)
        if arduino.in_waiting > 0:
            msg = arduino.readline().decode('utf-8', errors='ignore').strip()
            if msg:
                print(f"[ARDUINO] {msg}")

except Exception as e:
    print(f"[ERROR] {e}")

finally:
```

```
    try:
        dsp.close()
        arduino.close()
    except:
        pass
```

LISTING 7.9: Comunicación entre puertos (Python)

7.11. Matriz de confusión Distancia Euclidiana con CMVN(35dB)

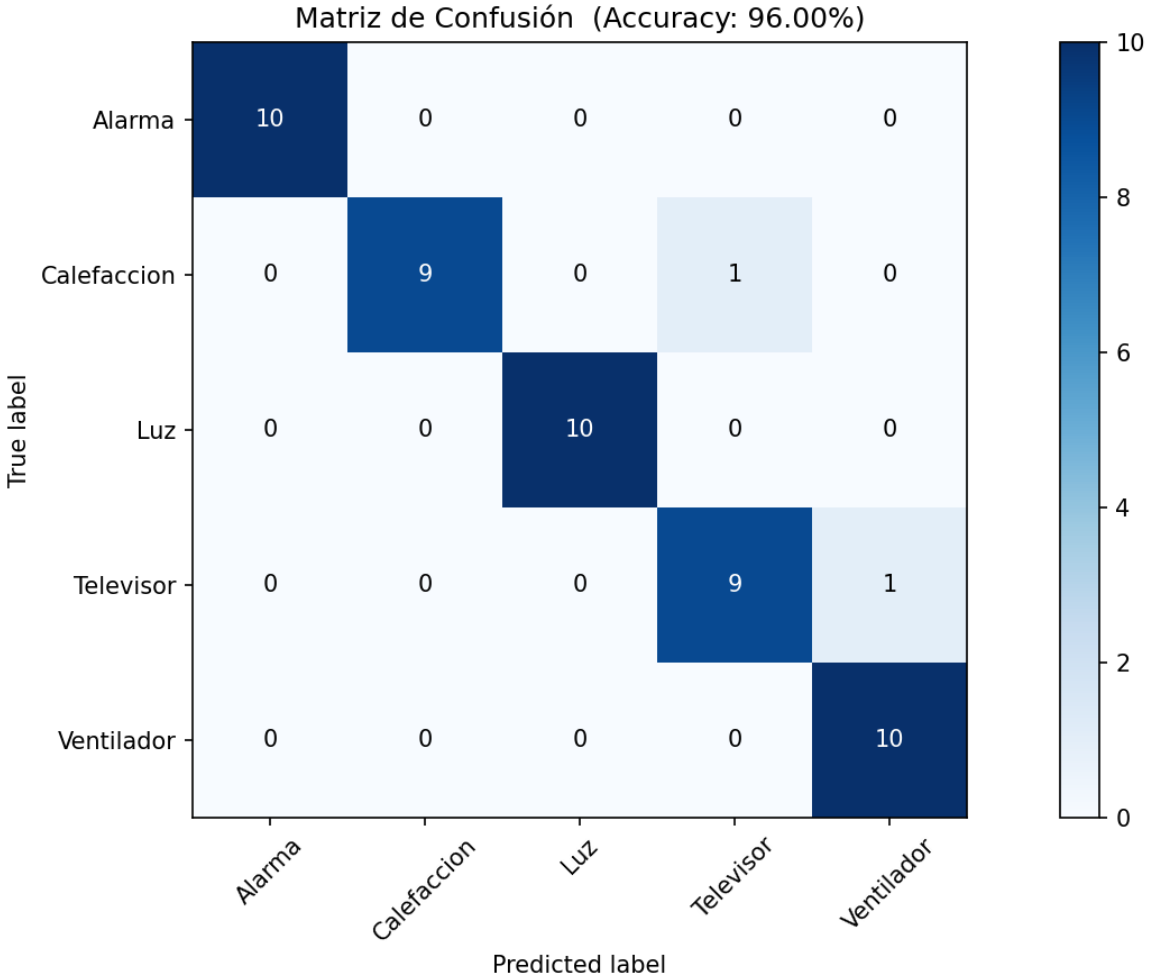

FIGURA 26: Diagrama metodológico 1.
Tomado de: Autores

7.12. Matriz de confusión GMM(35dB)

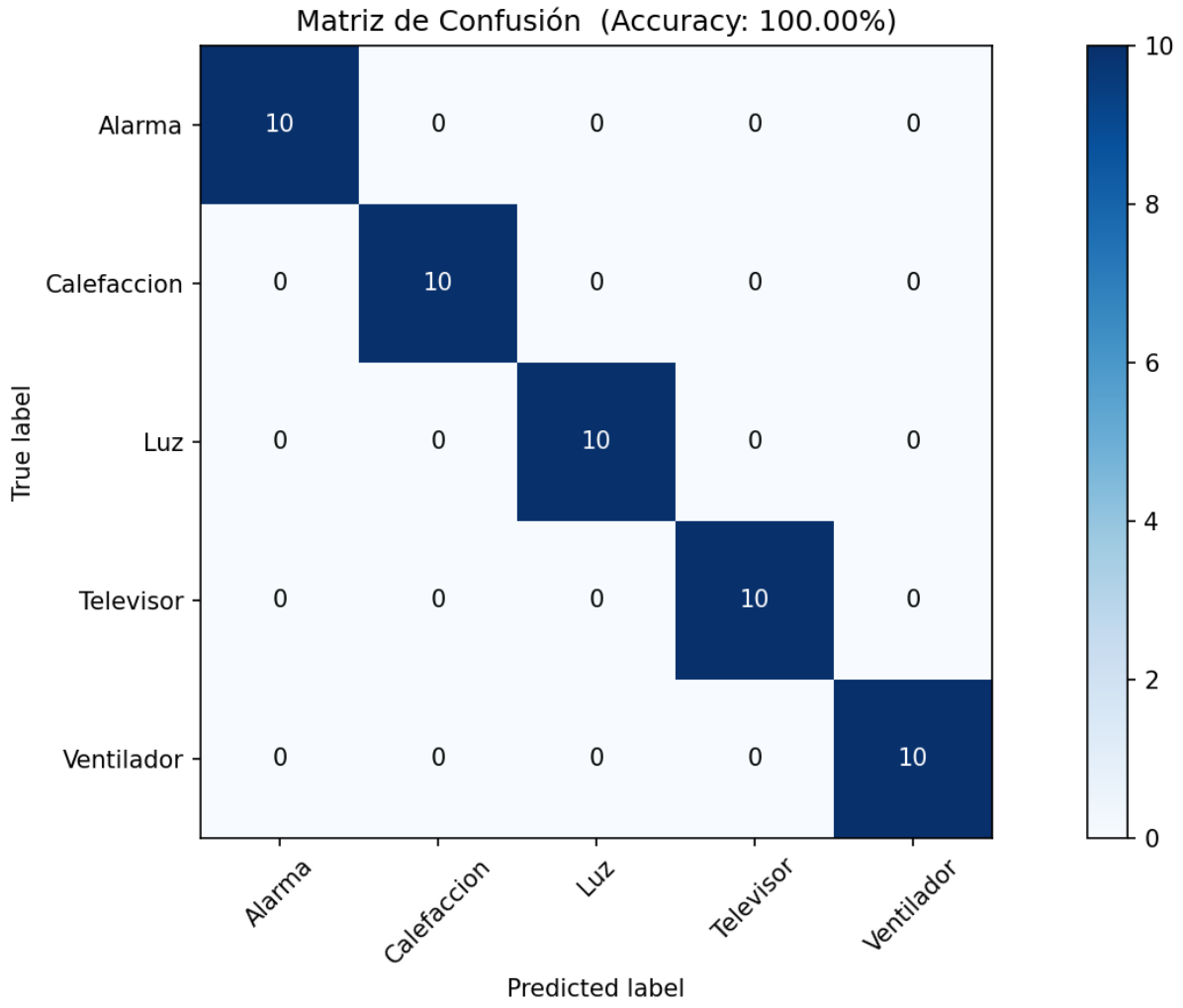

FIGURA 27: Diagrama metodológico 2.
Tomado de: Autores

7.13. Matriz de confusión DTW(35dB)

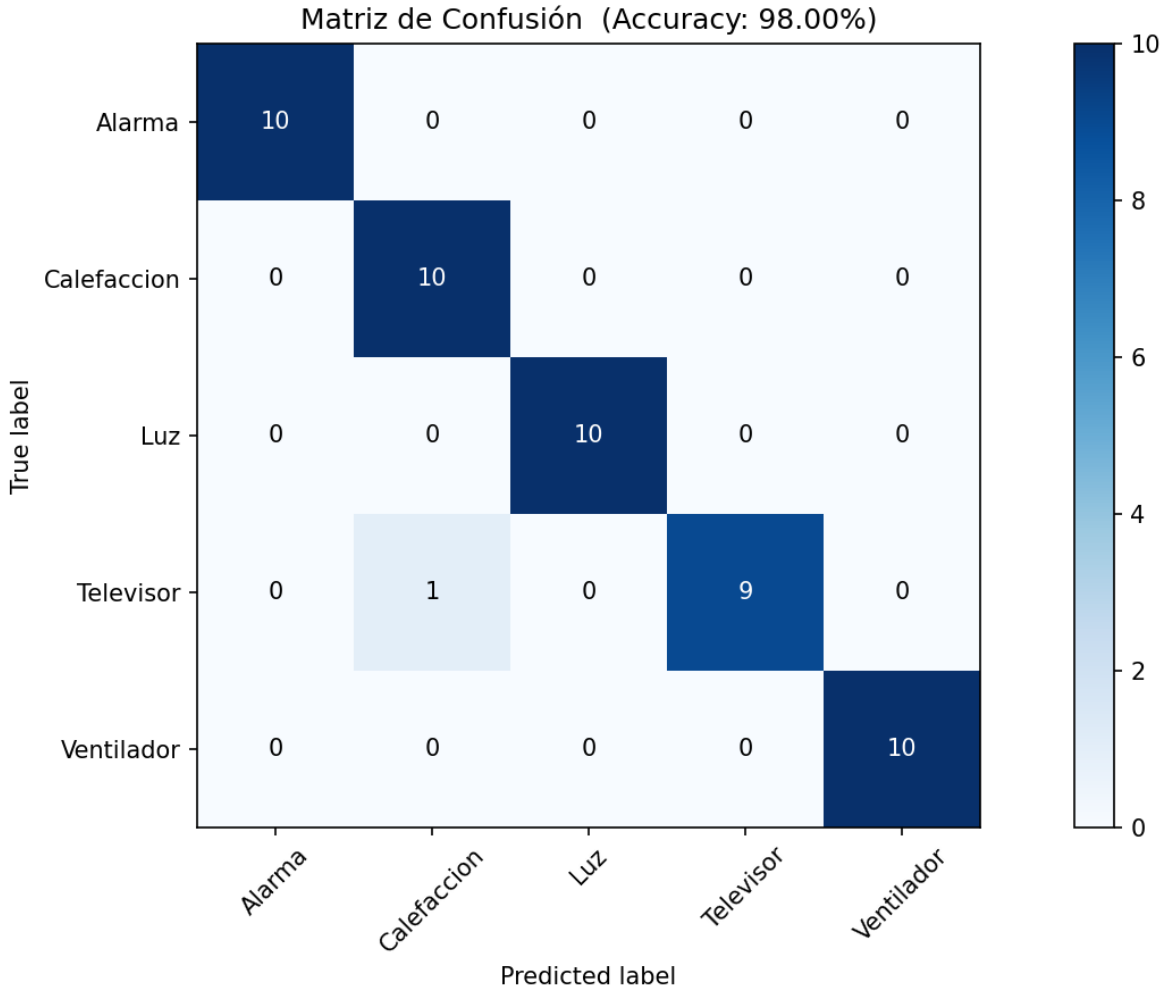

FIGURA 28: Diagrama metodológico 3.
Tomado de: Autores