



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Evaluación de un sistema inteligente de reconocimiento de órdenes en el puente de mando de las lanchas de instrucción

Grado en Ingeniería Mecánica

ALUMNO: Álvaro Salvador López
DIRECTORES: Milagros Fernández Gavilanes
Norberto Fernández García
CURSO ACADÉMICO: 2022-2023

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE GRADO

Evaluación de un sistema inteligente de reconocimiento de órdenes en el puente de mando de las lanchas de instrucción

Grado en Ingeniería Mecánica
Intensificación en Tecnología Naval
Cuerpo General

UniversidadeVigo

RESUMEN

La correcta formación de los oficiales de la Armada es un pilar clave para el futuro desempeño de estos. Partiendo desde su centro de formación en la Escuela Naval Militar, se realizan salidas periódicas en las lanchas de instrucción en las que se llevan a cabo ejercicios de maniobra, navegación, simulacros de hombre al agua y otros, intentando recrear el entorno operativo de un navío militar.

Estos ejercicios son realizados en tiempo real y calificados por el oficial al mando in situ, sin tener la posibilidad de realizar una evaluación posterior, la cuál sería de gran interés para detectar potenciales errores y aspectos a mejorar.

A partir de esta premisa, se plantea un proyecto de procesamiento de audio del lenguaje natural para tareas de comunicaciones navales, con el objetivo de realizar un registro del audio de las conversaciones en el puente de mando de las lanchas de instrucción. Con el audio obtenido se realiza un análisis para la detección automática de comunicaciones mediante un sistema entrenado de inteligencia artificial (IA) y procesamiento del lenguaje natural (PLN).

La integración de este sistema junto a diferentes datos de telemetría (rumbo, posición...) permiten la obtención de una bitácora digital que puede ser reproducido para la recreación y análisis del ejercicio realizado.

PALABRAS CLAVE

IA, lanchas, Whisper, Vosk, transcripción de órdenes, puente, datasets, procesamiento de audio, bitácora digital, inteligencia artificial, PLN

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a los profesores implicados en el trabajo, la profesora Milagros Fernández Gavilanes y el profesor Norberto Fernández García, por su apoyo e implicación en la realización del trabajo.

En segundo lugar, me gustaría agradecer el apoyo de mis compañeros de promoción por su colaboración en la aportación de ideas para el trabajo. Además, también me gustaría agradecer la facilidad aportada para la instalación del sistema de grabación por parte de las distintas dotaciones de la lancha de instrucción “Barrutia”.

Por último, quiero dedicar el trabajo a mi familia, en especial a mis padres María del Carmen y Juan Miguel, y a mi hermano, quienes siempre me han apoyado durante toda mi instancia en la “Escuela Naval Militar de Marín”.

CONTENIDO

Contenido	1
Índice de Figuras	5
Índice de Tablas.....	9
1 Introducción y objetivos	11
1.1 Motivación del trabajo	11
1.2 Objetivos	12
1.3 Estructura del trabajo	13
2 Estado del arte	15
2.1 Trabajos previos en el campo de estudio	15
2.1.1 Densidad de información en la comunicación del equipo de puente.....	15
2.1.2 Enseñanza de comunicación marítima basada en SMCP	15
2.1.3 Automatización de simulación de combate basada en habla inteligente.....	16
2.1.4 Aplicación del reconocimiento del habla en comunicación VHF.	16
2.1.5 Proyecto PANNACOTA	16
2.2 Reconocimiento de voz	17
2.2.1 Contexto histórico del reconocimiento de voz	17
2.2.2 Reconocimiento de voz en la actualidad	18
2.3 Aprendizaje automático	19
2.3.1 Machine Learning	19
2.3.2 Deep Learning	21
2.3.3 Redes neuronales convolucionales (CNN)	21
2.3.4 Redes neuronales recurrentes (RNN)	22
2.3.5 Sistemas de redes neuronales.....	23
2.4 Reducción de ruido	25
2.4.1 Filtro Wiener.....	25
2.4.2 Filtros pasa alto, pasa bajo y pasa banda	26
2.4.3 Atenuación adaptativa	27
2.4.4 Puerta espectral o Noisereduce	27
2.4.5 Proyecto RNNoise	27
2.5 Modelos y algoritmos de transcripción de audio	28
2.5.1 Modelo oculto de Markov (HMM).....	29
2.5.2 Algoritmo CTC	30
2.6 Herramientas de transcripción de audios	33

2.6.1 Whisper OpenAI.....	33
2.6.2 Vosk.....	34
2.6.3 Kaldi	35
2.6.4 Pocketsphinx.....	36
2.6.5 Speech Recognition	37
2.7 Extracción automática de palabras clave	38
2.8 Herramientas de simulación y representación	39
2.8.1 SIMDIS.....	39
2.8.2 Folium.....	40
3 Desarrollo del TFG.....	41
3.1 Lenguaje de programación, librerías y algoritmos empleados.....	41
3.1.1 Numpy	42
3.1.2 Soundfile.....	42
3.1.3 Scipy	42
3.1.4 SoundPy	43
3.1.5 Utils.....	43
3.1.6 Jiwer.....	43
3.2 Arquitectura del sistema.....	43
3.3 Proceso de grabación	45
3.4 Procesado de audio.....	46
3.4.1 Reducción de ruido	46
3.4.2 Segmentación de audios	47
3.5 Proceso de transcripción	47
3.5.1 Definición de vocabulario.....	47
3.5.2 Transcripción con Vosk y Kaldi KWS	48
3.5.3 Transcripción con Whisper.....	50
3.5.4 Transcripción con Whisper mejorado.....	50
3.6 Proceso de identificación de órdenes en las transcripciones.....	52
3.6.1 Algoritmos empleados	52
3.6.2 Extracción de características.....	54
3.6.3 Entrenamiento del sistema	55
3.7 Mecanismos de evaluación	57
3.7.1 Evaluación de las transcripciones mediante WER	57
3.7.2 Evaluación de la detección de órdenes	59
3.8 Simulación	59
4 Resultados de las pruebas.....	63

4.1 Resultados obtenidos en la transcripción mediante WER	63
4.1.1 Dataset de grabaciones	63
4.1.2 Evaluación de transcripción 1: transcripción Whisper	63
4.1.3 Evaluación de transcripción 2: Órdenes transcritas por Whisper	64
4.1.4 Evaluación de transcripción 3: Órdenes transcritas por Vosk	64
4.1.5 Evaluación de transcripción 4: Transcripción de Whisper mejorado con Vosk.....	65
4.2 Resultados obtenidos con el sistema detector de órdenes	66
4.2.1 Entrenamiento por transcripciones completas de 2 minutos	66
4.2.2 Entrenamiento por frases de las transcripciones.....	67
4.2.3 Entrenamientos por transcripciones y frases mejoradas	68
5 Conclusiones y líneas futuras	69
5.1 Conclusiones	69
5.2 Líneas futuras	70
6 Bibliografía.....	71
Anexo I: Implicaciones Sociales y Económicas,.....	79
Anexo II: Reflexiones Éticas y Sociales	81
Anexo III: Diccionario de siglas, acrónimos y abreviaturas	83
Anexo IV: Predicciones detector de órdenes.....	85
Anexo V: Unigramas, bigramas y trigramas	89
Anexo VI: Código de transcripción Whisper	93
Anexo VII: Código de Whisper mejorado y evaluaciones 1 y 4.....	97
Anexo VIII: Código de evaluación 3 para Vosk	103
Anexo IX: Código de preparación para SIMDIS	108

ÍNDICE DE FIGURAS

Figura 1-1: Objetivos del TFG. Fuente: propia	13
Figura 2-1: Bell Lab Audrey con la máquina de reconocimiento "AUDREY". Fuente: [11]	17
Figura 2-2: Primer sistema de reconocimiento de voz comercial. Fuente: [11].....	18
Figura 2-3: Modelo comercial de asistente virtual Alexa de Amazon. Fuente: [21]	19
Figura 2-4: Entrenamiento supervisado: clasificación y regresión. Fuente: propia	20
Figura 2-5: Entrenamiento no supervisado: clustering. Fuente: propia	20
Figura 2-6: AI, Machine Learning y Deep Learning. Fuente: [25].....	21
Figura 2-7: Ejemplo de uso CNN para extracción de características de imagen. Fuente: [26].....	22
Figura 2-8: Ejemplo esquemático de red neuronal tipo RNN. Fuente: [26]	22
Figura 2-9:Procesamiento de espectrogramas por CNN. Fuente: propia	23
Figura 2-10:RNN procesa los frames del mapa de propiedades. Fuente: propia	23
Figura 2-11:Obtención de probabilidades de caracteres por pasos de tiempo. Fuente: propia	24
Figura 2-12: Modelo seq2seq que se traduce del mandarín al inglés. Fuente: [44]	25
Figura 2-13: Esquema ilustrativo Filtro Wiener. Fuente: propia	26
Figura 2-14: Ejemplo filtrado Butterworth. Fuente: [48].....	26
Figura 2-15: Ejemplo de filtrado de audio con librería noisereducer. Fuente: [52]	27
Figura 2-16: Comparativa de RNN y GRU. Fuente: [57]	28
Figura 2-17: Ejemplo HMM con palabra right. Fuente: [61].....	29
Figura 2-18: Ejemplo de alineamiento necesario y situación antes del uso de CTC. Fuente: [64]..	30
Figura 2-19: División del audio continuo en frames discretos para procesado RNN. Fuente: [64]	31
Figura 2-20: Algoritmo de decodificación CTC. Fuente: [64].....	31
Figura 2-21: Algoritmo de CTC Loss. Fuente: [64].....	32
Figura 2-22: Esquema de funcionamiento Whisper. Fuente: [65]	33
Figura 2-23: Modelos de Whisper. Fuente: [65]	34
Figura 2-24: Librería Pywhisper. Fuente: [68].....	34
Figura 2-25: Librería Vosk. Fuente: [69]	35
Figura 2-26: Esquema componentes de Kaldi. Fuente: [70]	35
Figura 2-27: Esquema KALDI KWS. Fuente: propia	36
Figura 2-28: Esquema simplificado de funcionamiento Pocketsphinx. Fuente: [77]	37
Figura 2-29: Librería Speech Recognition. Fuente: [78]	38
Figura 2-30: Tipos de búsqueda de palabras clave. Fuente: propia	39
Figura 2-31: Simulaciones SIMDIS. Fuente: [81]	40
Figura 2-32: Librería Folium. Fuente: [82]	40
Figura 3-1: Lenguajes de programación más populares. Fuente: [83]	41

Figura 3-2 : Arquitectura general de sistema. Fuente: propia	43
Figura 3-3:Diagrama de la fase de procesamiento del sistema completo. Fuente: propia	44
Figura 3-4: Instalación del sistema en lancha junto micrófono ambiente. Fuente: propia.....	45
Figura 3-5: Colocación de micrófono de solapa. Fuente: propia.	45
Figura 3-6: Documento de frases normalizadas para las comunicaciones marítimas. Fuente: [5] ..	48
Figura 3-7: Línea de código Kaldirecognizer. Fuente: propia	48
Figura 3-8: Modelos disponibles de Vosk en español. Fuente: [91]	49
Figura 3-9: Transcripción de un chunk en función del vocabulario establecido. Fuente: propia	49
Figura 3-10: Transcripción para los chunks de un archivo y vocabulario <i>all</i> . Fuente: propia	49
Figura 3-11: Línea de código Whisper. Fuente: propia.....	50
Figura 3-12: Transcripción Whisper. Fuente: propia	50
Figura 3-13: Transcripción Whisper mejorado 1. Fuente: propia	51
Figura 3-14:Transcripción Whisper mejorado 2. Fuente: propia	52
Figura 3-15: Comparación CRF de cadena lineal y HMM. Fuente: propia	53
Figura 3-16: Ejemplo modelo BiLSTM-CRF. Fuente: [92].....	54
Figura 3-17: Esquema identificador de órdenes. Fuente: propia.....	55
Figura 3-18: Ejemplo de marcado IOB. Fuente: propia	55
Figura 3-19: Ejemplo de texto de entrenamiento del sistema detector de órdenes. Fuente: propia ..	56
Figura 3-20: Esquema sistema de entrenamiento. Fuente: propia.....	57
Figura 3-21: Resultado del sistema de obtención de órdenes. Fuente: propia	57
Figura 3-22: Cuenta de inserciones, faltas y sustituciones en una transcripción. Fuente: propia	58
Figura 3-23: Evaluaciones del sistema mediante WER. Fuente: propia	59
Figura 3-24: Archivo generador de simulación. Fuente: propia.....	60
Figura 3-25: Cobertura de las cartas electrónicas empleadas. Fuente: propia	61
Figura 3-26: Ejemplo simulación SIMDIS. Fuente: propia	61
Figura 4-1: Resultado de evaluación de la transcripción 1. Fuente: propia	64
Figura 4-2: Ejemplo de evaluación de la transcripción 2. Fuente: propia.....	64
Figura 4-3: Ejemplo de evaluación de la transcripción 3. Fuente: propia.....	65
Figura 4-4: Resultado de evaluación de la transcripción 4. Fuente: propia	65
Figura 4-5: Resultado entrenamiento CRF por archivos. Fuente: propia.....	66
Figura 4-6: Resultado entrenamiento BiLSTM-CRF por archivos. Fuente: propia.....	66
Figura 4-7: Resultado entrenamiento CRF por frases. Fuente: propia.....	67
Figura 4-8: Resultado entrenamiento BiLSTM-CRF por frases. Fuente: propia	67
Figura 4-9: Resultado entrenamiento CRF por transcripciones mejorado 1. Fuente: propia	68
Figura 4-10: Resultado entrenamiento CRF por frases mejorado 1. Fuente: propia.....	68
Figura A1. 1: Reparto de tiempo invertido en el proyecto. Fuente: propia.....	80

Figura A3. 1: Ejemplo de predicción de órdenes 1. Fuente: propia.....	85
Figura A3. 2: Ejemplo de predicción de órdenes 2. Fuente: propia	86
Figura A3. 3: Ejemplo de predicción de órdenes 3. Fuente: propia	87
Figura A4. 1: Ejemplo de archivo de unigramas. Fuente: propia.....	89
Figura A4. 2: Ejemplo de archivo de bigramas. Fuente: propia.....	90
Figura A4. 3: Ejemplo de archivo de trigramas. Fuente: propia	91

ÍNDICE DE TABLAS

Tabla 1: Cartas electrónicas empleadas en la simulación. Fuente: propia	60
Tabla A1. 1: Desglose de costes por fases. Fuente: propia.....	80

1 INTRODUCCIÓN Y OBJETIVOS

A lo largo de este primer capítulo se expondrá una breve contextualización del TFG, indicando la necesidad desde la cual surge su elaboración y se detallarán los objetivos que se persiguen, así como la organización del resto de la memoria.

1.1 Motivación del trabajo

La post-evaluación de actividades de enseñanza militar es un proceso esencial para mejorar el desempeño de estas. A través de la evaluación posterior a la realización de una actividad, es posible obtener información valiosa sobre qué aspectos se han llevado a cabo correctamente y cuáles requieren mejorar. Esto permite a los educadores adaptar y ajustar las actividades para lograr un mayor nivel de aprendizaje en sus estudiantes.

Además de ser útil para mejorar las actividades de enseñanza, la post-evaluación también tiene una serie de ventajas. Por ejemplo, puede ayudar a los educadores a entender mejor cómo procesan los estudiantes la información, lo que les permite adaptar su enseñanza para satisfacer las necesidades individuales de cada estudiante. También puede ayudar a los educadores a detectar problemas temprano, lo que les permite realizar una intervención antes de que se conviertan en problemas mayores.

En el ámbito militar, permite mejorar el adiestramiento del personal al comparar diferentes resultados en actividades realizadas por distintas personas. De esta forma se puede estudiar el rendimiento de cada persona individualmente y detectar automáticamente sus carencias, implementando las mejoras en las siguientes actividades.

Las herramientas de post-evaluación que destacan en los últimos años son las realizadas mediante programas de procesamiento de datos y uso de inteligencia artificial (IA), por su carácter innovador y su flexibilidad para la adaptación a distintas tareas específicas.

Este concepto de mejora de la enseñanza asociado a la innovación sigue la línea de las necesidades actuales del presente Almirante jefe de Estado Mayor de la Armada, que tiene como prioridad la modernización y mejora de la formación del personal, tal y como ha dejado ver en diferentes discursos y ha plasmado en la elaboración del documento Líneas Generales de la Armada [1].

De forma más concreta, un ejemplo de cómo la post-evaluación puede ser utilizada en la enseñanza es mediante el análisis de audios de actividades. A través del uso de técnicas de *audio mining* [2], es posible obtener palabras clave de las transcripciones del audio de las actividades, lo que permite analizar las palabras y frases utilizadas por los estudiantes. Esto puede ayudar a los educadores a detectar patrones en el lenguaje utilizado por los estudiantes, lo que les permite adaptar su enseñanza para asegurar que están cubriendo todos los conceptos necesarios.

En el contexto de la Armada Española, se podría realizar una post-evaluación de los oficiales al mando de buques para recabar información con objeto de la enseñanza de estos. Por ejemplo, se podrían evaluar las órdenes dadas por el oficial con la voz en el puente de mando de un buque, relacionándolas con la trayectoria del buque en cada momento, mediante telemetría. De esta forma, el instructor o persona a evaluar podría, una vez finalizado un ejercicio o actividad en el buque, ver si el oficial dio las órdenes correctas en el momento adecuado, y mediante la derrota seguida por el buque, ver si este gobernó adecuadamente y la tripulación actuó a tiempo con respecto a las órdenes.

Se tendría de esta manera un sistema de gran utilidad para los procesos de enseñanza y calificación de los oficiales de recién ascenso, de los alumnos en las lanchas de instrucción de la Escuela Naval Militar o de las calificaciones rutinarias en los buques a bordo de la flota para su despliegue en misiones internacionales.

Hoy en día esta evaluación de las órdenes se realiza *in situ* por el evaluador dentro del puente de mando o a posteriori mediante la escucha de audios realizados. La primera forma implica tener a una persona física escuchando al oficial de guardia, lo que aumenta la subjetividad de la evaluación. La segunda forma no es eficaz por el tedioso trabajo que supone transcribir manualmente los audios.

Por estos motivos, un sistema que recoja los audios y los transcriba automáticamente para posteriormente buscar las órdenes, sería de gran utilidad para la optimización de recursos y tiempo en el proceso de evaluación.

Una aproximación a la solución del problema está recogida en el proyecto PANNACOTA [3], expuesto en el IX Congreso Nacional de I+D en Defensa y Seguridad de 2022. A pesar de que el sistema planteado no está completo, las ideas y conclusiones obtenidas de él son de gran utilidad para el diseño del modelo completo y la implementación de las diferentes mejoras llevadas a cabo.

1.2 Objetivos

Actualmente no existe en los buques militares ningún sistema que proporcione información sobre la actuación del oficial de guardia ni tampoco almacene las órdenes dadas por él en cada momento, más allá de la crónica de acaecimientos rellena manualmente por una persona en el barco. Esta crónica con todas estas órdenes es transcrita por el oficial de guardia posteriormente al cuaderno de bitácora.

A simple vista, tener a una persona anotando todas las órdenes dadas por el oficial de guardia resulta ser un sistema tremendamente rudimentario que no solo requiere de personal exclusivo para el trabajo, sino que también puede llevar a errores humanos. Además, el análisis de la actuación del oficial mediante la crónica es complicado y muy poco visual.

Con el objetivo de solucionar estos problemas, se propone la realización de un sistema automático de captura y transcripción de conversaciones realizadas en el puente de mando mediante la utilización de las bibliotecas de procesamiento de audio *Vosk* y *Whisper*. El sistema tendrá como base las partes de grabación y transcripción con *Vosk* ya elaboradas en el proyecto PANNACOTA, que serán modificadas para su adaptación de un módulo de transcripción con *Whisper*. Los resultados de este serán combinados con los de *Vosk* y se complementarán al módulo inicial de transcripción para la obtención de mejores resultados.

El objetivo principal del trabajo reside en el análisis, evaluación y mejora del funcionamiento de este sistema mediante grabaciones realizadas en un entorno real, como es el de las lanchas de instrucción de la Escuela Naval Militar. Por tanto, se buscará evaluar el nuevo sistema, y mediante algoritmos supervisados realizar la detección las órdenes producidas en el texto transcrito. Esta evaluación se tendrá que llevar a cabo comparando las salidas producidas por el sistema transcriptor, y las salidas producidas por el detector de órdenes con los correspondientes textos revisados manualmente

Por otro lado, se propone crear una representación visual de lo obtenido sobre la derrota real del buque, facilitando la observación de los errores cometidos por el oficial.

Por tanto, se marcan cuatro hitos principales a conseguir, representados en la Figura 1-1:

- 1- Conseguir suficientes datos de grabaciones y telemetría a partir de las salidas semanales en las lanchas de instrucción.
- 2- Realización de un sistema de transcripción preciso mediante la combinación de *Whisper* y *Vosk* que mejore el actual del proyecto PANNACOTA.
- 3- Entrenamiento supervisado de un sistema detector de órdenes y obtención de una precisión superior al 70 %.
- 4- Simulación de una salida a la mar completa con el sistema terminado.



Figura 1-1: Objetivos del TFG. Fuente: propia

1.3 Estructura del trabajo

El trabajo se organiza en los siguientes capítulos:

- Capítulo 1: Introducción y objetivos. Se expone el contexto del trabajo, definiendo la necesidad de la evaluación de actividades militares para la mejora, la motivación a la realización del trabajo y los objetivos a conseguir.
- Capítulo 2: Estado del arte. En este segundo capítulo se realizará una revisión del marco teórico en el que se engloba este trabajo. Los conceptos teóricos tratados en este capítulo se pueden resumir en los siguientes puntos:
 - o Análisis de los estudios existentes relacionados con el tema.
 - o Historia y actualidad del reconocimiento de voz.
 - o Explicación de los conceptos de *Machine Learning* y *Deep Learning*.
 - o Métodos de reducción de ruido.
 - o Algoritmos y sistemas más empleados en las transcripciones de audio.
 - o Herramientas y librerías de transcripción de audio, indicando las características generales y el funcionamiento de cada una de ellas.

- o Explicación de los conceptos de indexación de palabras clave, *Keyword Extraction* y *Keyword Assignment*, en los cuales está basado el sistema detector de órdenes.
- Capítulo 3: Desarrollo. Se explicará la arquitectura del sistema, detallando el hardware empleado y los procesos realizados en las fases de grabación y procesamiento de audio, así como el uso de las librerías en Python. Se explicarán por apartados los pasos de segmentación de audios, transcripción, detección de órdenes, evaluación y simulación.
- Capítulo 4: Pruebas. Se realizan pruebas para comprobar la influencia de ciertos factores en los resultados de búsqueda de palabras y de transcripción. Se comentan los resultados de la evaluación de las diferentes pruebas realizadas a la hora de realizar las transcripciones, así como los resultados obtenidos en el entrenamiento del detector de órdenes.
- Capítulo 5: Conclusión y líneas futuras. Se realiza un resumen y un análisis de los resultados de las pruebas realizadas y líneas futuras para futuros proyectos relacionados.
- Finalmente se adjunta la bibliografía donde se especifican las fuentes utilizadas a lo largo del trabajo y una serie de anexos que tienen como propósito apoyar el desarrollo del trabajo para una mejor comprensión del mismo.

2 ESTADO DEL ARTE

El presente capítulo tratará de estudiar en qué consiste el reconocimiento de voz, así como el desarrollo de esta tecnología junto con las dos principales ramas de la inteligencia artificial: *Machine Learning* y *Deep Learning*. Se comenzará con un estudio de los trabajos relacionados de interés para el trabajo, para seguir con un estudio de la historia del reconocimiento de voz y los modelos matemáticos y probabilísticos empleados actualmente en este. A continuación, se tratarán de mencionar los sistemas de transcripción de audio más conocidos hoy en día para su uso con el lenguaje de programación Python, junto a las herramientas de simulación consideradas en el trabajo. Por último, se mencionará el funcionamiento de los sistemas de búsqueda de palabras clave en textos.

2.1 Trabajos previos en el campo de estudio

A continuación, se muestra una breve reseña de algunos ejemplos de avances que se han realizado previamente en este campo de estudio y que han servido para enfocar ciertos aspectos de este trabajo.

2.1.1 Densidad de información en la comunicación del equipo de puente

Trabajo realizado por Peter John, Benjamin Brooks, Christoph Wand y Ulf Schriever. A pesar de que no incluye un sistema de transcripción como tal, este trabajo presenta datos de gran interés para los modelos de transcripción específicos para barcos [4].

El artículo presenta un método para medir el flujo de información en la comunicación de equipos de puente, basado en un análisis de la transcripción del registrador de datos de viaje del MV Cosco Busan, que chocó con un pilar de puente en la bahía de San Francisco con densa niebla. El objetivo es evaluar la eficiencia de la comunicación marítima al desarrollar un índice específico adecuado para evaluar el rendimiento de la comunicación individual y de equipos. La colisión investigada en este artículo conduce a la conclusión de que el flujo de información en la comunicación de equipos de puente se puede medir cuantitativamente y que se puede demostrar una correlación con factores situacionales externos.

Además, el estudio es capaz de demostrar la relevancia de las comunicaciones y la viabilidad del estudio de estas con objeto de extraer conclusiones para ámbitos como el de la enseñanza.

2.1.2 Enseñanza de comunicación marítima basada en SMCP

Este taller se centra en la formación en la comunicación marítima basada en el SMCP [5] (Código de Prácticas de Comunicación de la Marina) con la ayuda de sistemas de diálogo (también conocidos como ChatBots). Los participantes adquieren competencias en comunicación a través de la práctica de

la comunicación marítima relevante a su propio ritmo de aprendizaje, lo que permite un enfoque en el estudiante. Por un lado, los ejercicios presentados en el taller se centrarán en la parte obligatoria A del SMCP en su uso en la comunicación por VHF. Por otro lado, una llamada telefónica simulada desafía a los participantes a desarrollar una estrategia de comunicación eficiente para obtener detalles relevantes sobre un accidente náutico. La respuesta de los participantes de la conferencia se utilizará para mejorar aún más los ejercicios desarrollados [6].

El taller, por lo tanto, es capaz de realizar un sistema de mejora a la enseñanza mediante técnicas innovadoras de computación, tal y como pretende este trabajo.

2.1.3 Automatización de simulación de combate basada en habla inteligente.

Este proyecto se basa en las pruebas de simulación de combate, que son el principal medio de evaluación de los sistemas de información de mando. Con el fin de mejorar la inteligencia y eficiencia del experimento de simulación, se propone el método de aplicar la tecnología de interacción por voz inteligente (ISIT) en el proceso de automatización de la simulación. Para ello se diseñan los procesos de reconocimiento de instrucciones de voz, análisis de elementos de instrucción, generación de instrucciones de datos y retroalimentación de síntesis de voz basados en ISIT humano-máquina. Además, tomando como ejemplo la escena de mando y guía de combate aéreo, se construye la base de datos de instrucciones de voz de mando y guía, y se diseña y verifica la simulación de interacción por voz. Los resultados muestran que ISIT puede reducir efectivamente el número de comandantes y operadores en la prueba de simulación de combate aéreo, y mejorar la eficiencia de la prueba [7].

El proyecto pone de relevancia la necesidad del uso del habla como medio de evaluación, y como ya está siendo implementado en el ámbito militar.

2.1.4 Aplicación del reconocimiento del habla en comunicación VHF.

El trabajo se enfoca en la creencia de que en un futuro cercano surgirán barcos autónomos, con proyectos en marcha en empresas como Rolls Royce y en el gobierno japonés. Sin embargo, se espera que exista una mezcla de barcos con tripulación y barcos sin tripulación en el mar durante mucho tiempo o de forma permanente. Por lo tanto, es necesario que los barcos con y sin tripulación tengan medios para comunicarse entre sí. Para ello, el objetivo principal del autor es verificar la precisión de la tecnología de reconocimiento de voz, como la de Google, en la conversación en radio VHF internacional [8].

Este proyecto ha sido realizado por la Universidad de Ciencias Marinas y Tecnología de Tokyo, y aunque no sea de acceso público, nos da una idea del uso de los algoritmos de reconocimiento de voz en el ámbito marino y cómo otros países están interesados en el tema.

2.1.5 Proyecto PANNACOTA

Se trata de un proyecto financiado por el CUD-ENM a través del proyecto propio de investigación Processing Audio and Natural language for NAval COmmunications TAsks (PICUD-2020-09) [3].

Este proyecto ha sido el punto de partida de este trabajo. A partir del sistema de transcripción que elabora, se pretende construir el sistema completo, añadiendo el resto de elementos que hacen posible resolver el problema planteado: Un sistema de evaluación para los futuros oficiales en el puente de mando de las lanchas de instrucción de la Escuela Naval Militar.

Los resultados obtenidos del sistema ya elaborado constituyen la base para la elaboración del sistema robusto completo capaz de generar una simulación a partir de la entrada de audio y datos de telemetría de las lanchas.

2.2 Reconocimiento de voz

A continuación, se procederá a explicar la historia del reconocimiento desde su comienzo hasta la actualidad, lo cual nos aporta una idea de su desarrollo hasta el momento y de las capacidades y limitaciones que tiene hoy en día.

2.2.1 Contexto histórico del reconocimiento de voz

La historia del reconocimiento de voz comienza en 1870 por parte del científico-inventor escocés Alexander Graham Bell, que quiso inventar un dispositivo para representar la palabra visualmente a partir de ondas acústicas [9]. A pesar de que no lo consiguió, sus esfuerzos derivaron en la invención del teléfono.

Más tarde, en 1930, el ingeniero mecánico Tihamer Nemes propuso una máquina para la transcripción automática de voz que fue rechazada y considerada imposible.

Serían 20 años después cuando realmente se intentó la creación de esta máquina, por la empresa de telecomunicación AT&T. En 1952 fue desarrollado un sistema por Bell Laboratories capaz de reconocer números hasta el 9 sobre un ordenador analógico, siendo el precursor de lo que poco después sería el reconocimiento de vocales y consonantes [10]. En la Figura 2-1 se puede ver la exposición de la máquina, en este caso reconociendo el número dos a partir del audio introducido al micrófono.



Figura 2-1: Bell Lab Audrey con la máquina de reconocimiento "AUDREY". Fuente: [11]

Durante la década de 1960, los investigadores expertos en reconocimiento de voz comenzaron a comprender la complejidad de crear una aplicación efectiva. Al principio, se desarrollaron aplicaciones con vocabularios limitados y con palabras habladas de manera intermitente. En ese mismo año, en la Universidad de Carnegie Mellon e IBM [12] empezaron a investigar el reconocimiento de voz continuo. El impacto de sus investigaciones se notó años más tarde.

En la década de 1970, se creó el primer sistema de reconocimiento de voz comercial, cuya portada se encuentra en la Figura 2-2. Se mejoraron las aplicaciones de reconocimiento de voz basadas en el hablante y que requerían una entrada intermitente y un vocabulario reducido. Por otro lado, la *Advanced Research Projects Agency* (DARPA) mostró interés en la investigación del reconocimiento de voz. DARPA comenzó a investigar el habla continua y a utilizar vocabularios más extensos. Además, se mejoró la tecnología para reconocer palabras aisladas en señales de audio continuas, y se desarrollaron técnicas como modelado probabilístico y algoritmos de retropropagación [13].

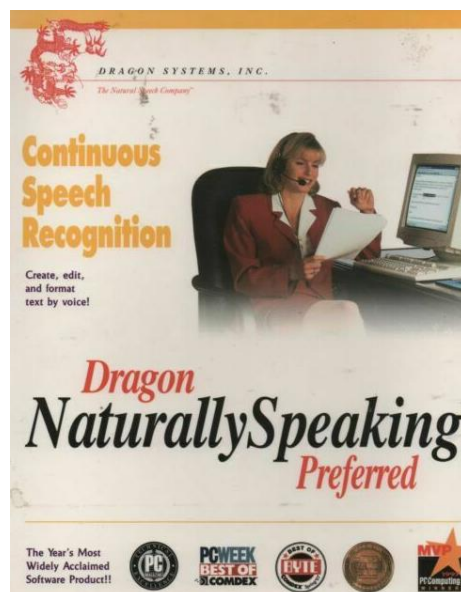


Figura 2-2: Primer sistema de reconocimiento de voz comercial. Fuente: [11]

En la década de 1980, el reconocimiento de voz se vio favorecido por tres factores: el aumento en la popularidad de los ordenadores personales, el apoyo de DARPA; empresa del departamento de investigación de EE. UU, y la disminución en los costos de las aplicaciones comerciales. En ese periodo, el principal interés era desarrollar vocabularios más grandes.

En 1985, un vocabulario de 100 palabras era considerado grande, pero en 1986 se llegó a crear uno con 20,000 palabras. Además, durante esa época hubo importantes avances, ya que se cambió del enfoque basado en reconocimiento de patrones a el uso de métodos probabilísticos, como los Modelos Ocultos de Markov [14], que serían el paso a la creación de sistemas de reconocimiento de voz modernos más parecidos a los usados actualmente.

2.2.2 Reconocimiento de voz en la actualidad

El reconocimiento automático del habla ha mejorado a un ritmo sorprendente en los últimos años. Este progreso se ha visto facilitado en parte por el acceso a la potencia de cálculo suficiente para entrenar modelos muy grandes, como el *wav2vec 2.0* [15] lanzado por Meta, que tiene 900 millones de parámetros. Sin embargo, el ingrediente clave para el éxito han sido modelos más avanzados, que han reducido significativamente el coste para obtener datos de entrenamiento.

La invención de la pérdida CTC (*Connectionist Temporal Classification Loss*) [16], que se detallará en la sección 2.5.2, ha hecho posible utilizar texto no alineado con señales de audio para el reconocimiento del habla. Anteriormente, los caracteres y/o fonemas debían alinearse manualmente en el tiempo con la señal de audio de entrada, lo que era un proceso lento. De hecho, CTC ha hecho que la preparación de los datos de entrenamiento supervisados [17] sea un 90% más rápida que con los primeros sistemas.

En cuanto a los modelos de lenguaje, las técnicas estocásticas, como el suavizado *Kneser-Ney* de *KenLM* [18] han permitido a los investigadores extraer conocimientos de ortografía y gramática de colecciones de audio y texto muy grandes y sin estructurar.

Los algoritmos de reconocimiento de voz utilizados actualmente se basan en inteligencia artificial y *Deep Learning* [19]. Estos algoritmos son capaces de aprender patrones complejos en los datos de audio y mejorar la precisión de la transcripción. Uno de los más utilizados en el reconocimiento de voz actualmente es la red neuronal recurrente (RNN) [17]. Las RNN son especialmente útiles para modelar

las dependencias a lo largo de las secuencias de audio, lo que les permite manejar la variabilidad en la duración de las palabras y la pronunciación.

Además, los algoritmos de reconocimiento de voz actuales también utilizan redes neuronales convolucionales (CNN) [20] para extraer características relevantes de la entrada de audio. Estas características son luego procesadas por otras capas de la red para producir una transcripción precisa.

Por otro lado, todos estos avances en la tecnología para el reconocimiento de voz también han permitido una mejora en la velocidad y la eficiencia de los sistemas, siendo posible su implementación en sistemas utilizados para asistentes virtuales como Siri o Alexa como el de la Figura 2-3.

Otros ejemplos actuales del uso de sistemas de reconocimiento de voz pueden ser la generación de subtítulos automáticos sobre audio o video o la automatización de procesos industriales, facilitando la búsqueda en bases de datos o el envío de mensajes.



Figura 2-3: Modelo comercial de asistente virtual Alexa de Amazon. Fuente: [21]

2.3 Aprendizaje automático

Como se ha podido observar hasta el momento, los sistemas de reconocimiento de voz actuales están fundamentalmente basados en el uso de técnicas de inteligencia artificial. Además, el uso de esta tecnología se puede ver reflejada en técnicas que se abordarán en este trabajo, como el entrenamiento de un sistema para la búsqueda de palabras clave en textos. Es por ello necesaria una definición de los conceptos de inteligencia artificial asociados a la explicación de los sistemas abordados en el trabajo.

2.3.1 Machine Learning

El *Machine Learning* [13], o aprendizaje automático en español, es una tecnología que se está volviendo cada vez más relevante en el mundo actual. La idea detrás del *Machine Learning* es permitir que las máquinas aprendan y evolucionen con la experiencia, lo que les permite realizar tareas para las que no fueron programadas inicialmente.

Para lograr esto, se utilizan algoritmos que les permiten a las máquinas analizar grandes conjuntos de datos, conocidos como *Big Data*, y hacer predicciones basadas en patrones identificados. Estos algoritmos son capaces de ajustar su comportamiento y rendimiento en función de los resultados de sus acciones anteriores.

Existen diferentes tipos de algoritmos de *Machine Learning*, y se pueden agrupar en tres categorías principales: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

- El aprendizaje supervisado se utiliza para la clasificación y la predicción, y se basa en un conjunto de datos etiquetados. El aprendizaje no supervisado se utiliza para el análisis de datos y la identificación de patrones, y se basa en un conjunto de datos no etiquetados. Por último, el aprendizaje por refuerzo se utiliza para la toma de decisiones y la optimización, y se basa en un proceso de prueba y error [19].

Dentro del aprendizaje supervisado, existen dos tipos principales de aprendizaje representados en la Figura 2-4:

- Clasificación: es utilizada para clasificar los datos de entrada en variables discretas;
- Regresión: se utiliza para predecir un valor de salida a partir de un rango continuo de valores.

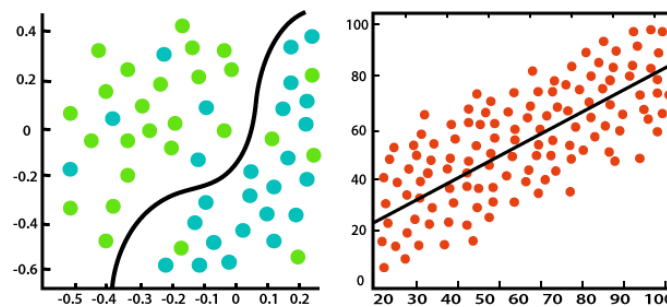


Figura 2-4: Entrenamiento supervisado: clasificación y regresión. Fuente: propia

Un ejemplo de aplicación puede ser los *Conditional Random Fields* (CRF) [22] que son una técnica de aprendizaje supervisado utilizada en *Machine Learning* para la modelización de datos secuenciales. Los CRF permiten considerar múltiples variables de entrada y salida, lo que les permite ser utilizados para una amplia variedad de tareas, como el etiquetado de partes del discurso, el reconocimiento de entidades nombradas y el análisis de sentimientos.

Los CRF se basan en un enfoque probabilístico que modela la probabilidad condicional de una etiqueta dada una secuencia de entrada. Esta técnica ha demostrado ser efectiva en la resolución de tareas de *Natural Language Processing* (PLN) y se ha utilizado con éxito en una variedad de aplicaciones, incluyendo la extracción de información, la traducción automática y la clasificación de texto.

- En el aprendizaje no supervisado, los datos no están etiquetados y se desconoce la salida, por lo que se busca encontrar patrones y estructuras ocultas en los datos. Este tipo de aprendizaje utiliza distintos algoritmos que pueden englobar en [23]:
 - Agrupamiento (*Clustering*)
 - Detección de anomalías
 - Asociación
 - Reducción de la dimensionalidad

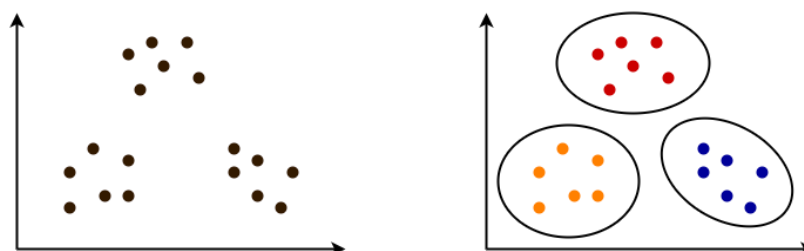


Figura 2-5: Entrenamiento no supervisado: clustering. Fuente: propia

2.3.2 Deep Learning

El avance de la inteligencia artificial en los últimos años ha supuesto un nuevo paradigma. En concreto, en el ámbito del reconocimiento de voz, el *Deep Learning* ha supuesto uno de los mayores avances. Se trata de una de las ramas de la inteligencia artificial caracterizada por el uso de redes neuronales. Estas, son capaces de aprender de forma independiente y resolver tareas propias de los seres humanos, como la identificación del hablante por su tono o identificación de sentimientos en base a expresiones empleadas.

El *Deep Learning* se engloba dentro del ya explicado *Machine Learning* [19], que dentro del campo del inteligencia artificial, pretende crear sistemas con aprendizaje automático a partir de experiencias o entrenamiento, es decir, con el aporte de una gran cantidad de datos del que extraer características. Para ello utiliza redes neuronales con más de tres capas, que pueden ser de distintos tipos. La evolución de estos campos puede ser observada en la Figura 2-6.

En resumen, una red neuronal consiste en un conjunto de expresiones matemáticas recopiladas en las llamadas “neuronas”, que son capaces de coordinarse entre sí para obtener una red. Las más simples son las redes neuronales artificiales (ANN) [24]. Cuando contienen más de tres capas pasan a denominarse redes neuronales profundas. A continuación, se explicarán los tipos de redes profundas de uso extendido en los sistemas de reconocimiento de voz, así como algunos de los sistemas en los que se emplean.

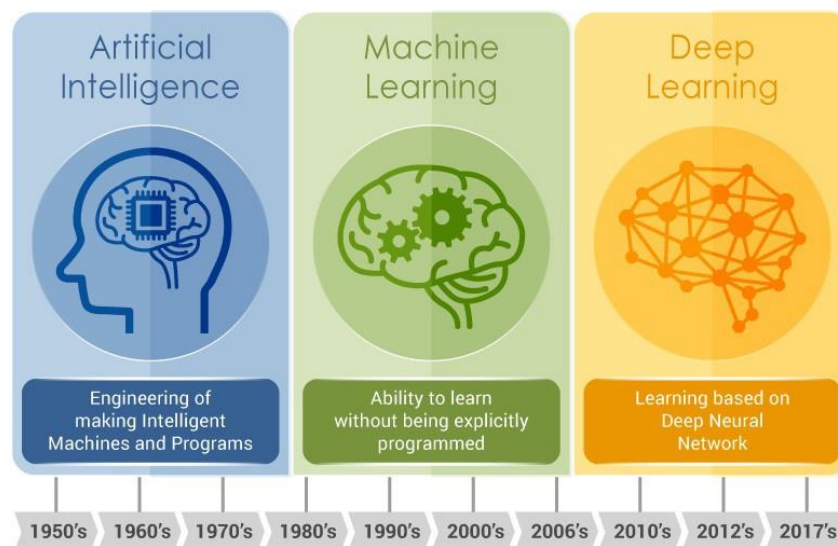


Figura 2-6: AI, Machine Learning y Deep Learning. Fuente: [25]

2.3.3 Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal profunda basado en la biología de la corteza visual, que intenta utilizar el método que utiliza el cerebro para procesar imágenes a través de la retina. Para ello, al igual que nuestro cerebro, utiliza grupos de neuronas que realizan un submuestreo de la capa o imagen para procesar partes específicas, superponiéndose entre sí para formar el campo visual. La imagen utilizada puede ser de muchos tipos, desde un espectrograma de audio a una fotografía. Un ejemplo de extracción de características y submuestreo para una imagen está recogido en la Figura 2-7.

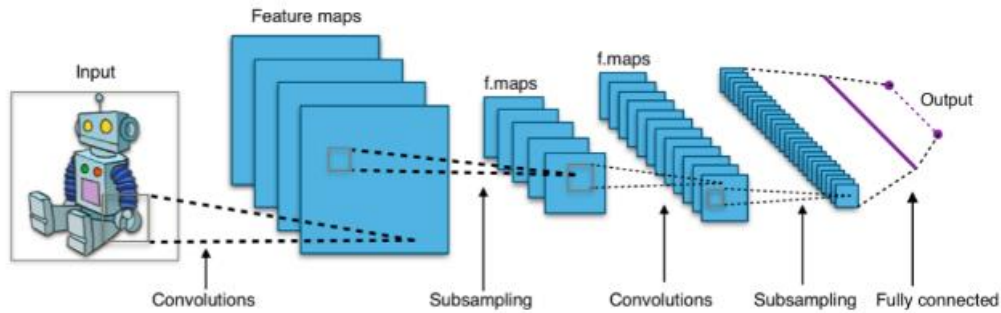


Figura 2-7: Ejemplo de uso CNN para extracción de características de imagen. Fuente: [26]

2.3.4 Redes neuronales recurrentes (RNN)

Este tipo de redes neuronales profundas (RNN) se basan en el uso de una memoria. Las salidas de las neuronas son utilizadas de nuevo la próxima vez que se usa el modelo de red neuronal, cosa que en las redes tradicionales no sucede, asumiendo sucesos independientes. Este proceso es conocido como retroprogragación.

Existen RNN especializadas como las LSTM (*Long short-term memory*) [27] o las GRU (*Gated Recurrent Unit*) [28], que tienen como característica principal la capacidad de mantener estados separados a corto y largo plazo.

Las RNN son adecuadas para problemas de aprendizaje automático que involucran datos secuenciales porque pueden recordar información relevante sobre la entrada que recibieron, lo que les permite ser más precisas en la predicción de lo que vendrá después. La memoria interna permite a las RNN mantener información de contexto a diferencia de otros tipos de redes [29].

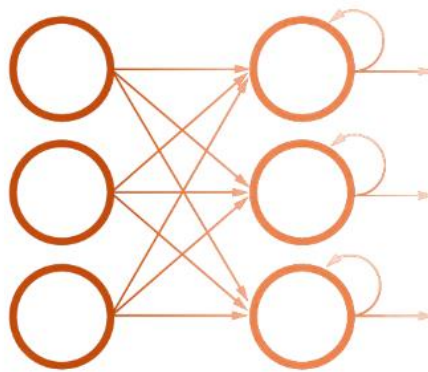


Figura 2-8: Ejemplo esquemático de red neuronal tipo RNN. Fuente: [26]

En el reconocimiento de voz, las redes neuronales recurrentes son capaces de modelar las relaciones entre las características acústicas del habla en secuencia, lo que les permite capturar la información contextual necesaria para identificar y transcribir correctamente las palabras habladas.

En particular, las RNN pueden procesar secuencias de datos de entrada, como una serie de vectores que representan los espectrogramas del habla, y producir una salida en forma de secuencia, como una transcripción fonética de las palabras habladas. Las RNN también son capaces de manejar variaciones en el tiempo de las señales de entrada, lo que las hace útiles en situaciones donde el ritmo o la velocidad del habla pueden cambiar.

El esquema de una RNN simple con tres neuronas y una capa está representado en la Figura 2-8.

2.3.5 Sistemas de redes neuronales

En los últimos años, las redes neuronales de Deep Learning han avanzado en la tecnología de reconocimiento automático de voz (ASR) a gran escala. Las redes neuronales no solo pueden extraer características acústicas, que se utilizan como entradas para modelos tradicionales de ASR como los Modelos de Markov Ocultos (HMM), sino que también actúan como transductores de secuencias, lo que resulta en sistemas de ASR (Automatic Speech Recognition) neuronales de extremo a extremo. Las estructuras más conocidas de estas redes neuronales son dos [30]:

2.3.5.1 CNN+RNN

La primera es una arquitectura de CNN [31] más RNN (Red Neural Recurrente) que utiliza el algoritmo CTC Loss para demarcar cada carácter de las palabras en el habla, por ejemplo, el modelo *Deep Speech* de Baidu [32].

Este ejemplo de estructura cuenta con estas partes [33]:

- Una red convolucional regular que consiste en varias capas residuales CNN que procesan las imágenes de espectrograma de entrada y producen mapas de características de esas imágenes.

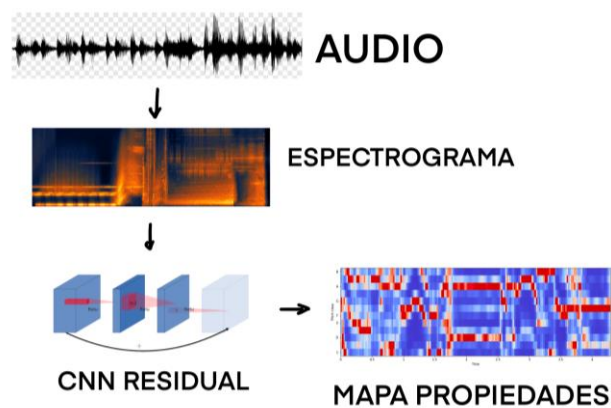


Figura 2-9: Procesamiento de espectrogramas por CNN. Fuente: propia

- Una red recurrente regular que consiste en varias capas Bidireccional LSTM [34] que procesan los mapas de características como una serie de instantes distintos o "fotogramas" que corresponden a la secuencia deseada de caracteres de salida. En otras palabras, toma los mapas de características que son una representación continua del audio y los convierte en una representación discreta.

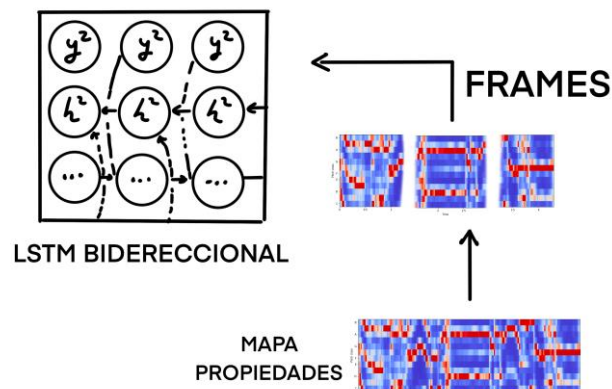


Figura 2-10: RNN procesa los frames del mapa de propiedades. Fuente: propia

- Una capa lineal con función softmax [35] (función que se encarga de comprimir los vectores bidimensionales) que utiliza los vectores de LSTM para producir probabilidades de caracteres para cada instante de la salida.

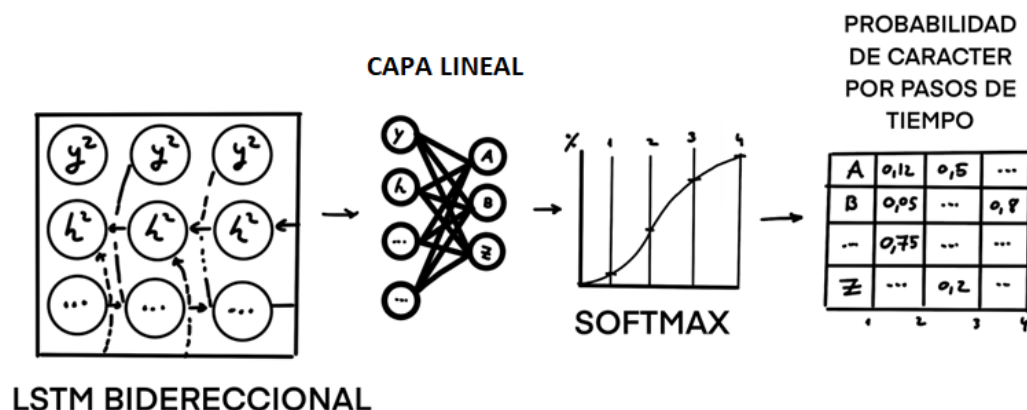


Figura 2-11: Obtención de probabilidades de caracteres por pasos de tiempo. Fuente: propia

El resultado de caracteres y probabilidades debe alinearse con el audio original, para obtener el tiempo de cada palabra. Para ello se usa el algoritmo CTC, explicado en la sección 2.5.2.

2.3.5.2 SEQ2SEQ

La segunda arquitectura más conocida sería una red secuencia a secuencia (seq2seq) basada en RNN que trata cada "rebanada" del espectrograma como un elemento en una secuencia, por ejemplo, el modelo *Listen Attend Spell* (LAS) de Google [36].

Los modelos seq2seq se han vuelto populares en la comunidad de reconocimiento de voz automático (ASR) como una manera de unir los modelos acústicos, de pronunciación y de lenguaje en un solo modelo. Existen varios modelos seq2seq hasta la fecha, como por ejemplo los *Recurrent Neural Network Transducer* (RNNT) [37], *Listen, Attend and Spell* (LAS) [38], *Neural Transducer* [39], *Monotonic Alignments* [40] y *Recurrent Neural Aligner* (RNA) [41]. Aunque estos modelos han demostrado resultados prometedores, aún no está claro si estos enfoques serían prácticos para reemplazar los actuales modelos basados en HMM con redes neuronales acústicas.

El entrenamiento de estos modelos es más sencillo que los sistemas convencionales de ASR y no requieren de árboles de decisión ni de alineaciones de tiempo con CTC. Sin embargo, hasta la fecha, ninguno de estos modelos ha superado a un sistema ASR de última generación en una tarea de reconocimiento de voz continuo y con un gran vocabulario [42].

La arquitectura más común utilizada para construir modelos Seq2Seq e ilustrada en la Figura 2-12 para un traductor es la arquitectura codificador-decodificador [43]:

- **Codificador:** Tanto el codificador como el decodificador son modelos LSTM (o a veces modelos GRU). El codificador lee la secuencia de entrada y resume la información en algo llamado vectores de estado interno o vector de contexto (en el caso de LSTM se llaman vectores de estado oculto y celular). Descartamos las salidas del codificador y solo preservamos los estados internos. Este vector de contexto tiene como objetivo encapsular la información de todos los elementos de entrada para ayudar al decodificador a hacer predicciones precisas.
- **Decodificador:** El decodificador es un LSTM cuyos estados iniciales se inicializan con los estados finales del LSTM Codificador, es decir, el vector de contexto de la celda final del codificador se introduce en la primera celda de la red decodificadora. Usando estos estados

iniciales, el decodificador comienza a generar la secuencia de salida, y estas salidas también se tienen en cuenta para futuras salidas.

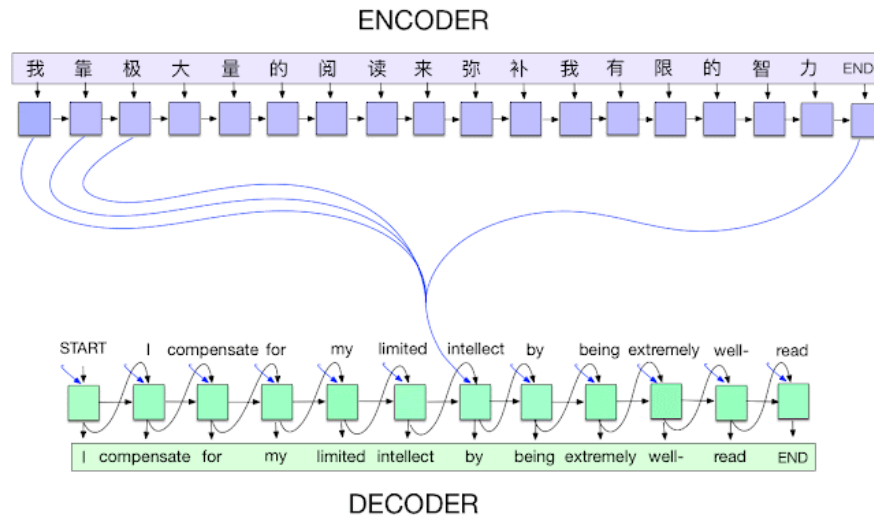


Figura 2-12: Modelo seq2seq que se traduce del mandarín al inglés. Fuente: [44]

2.4 Reducción de ruido

Normalmente, uno de los pasos previos a la introducción de audios ruidosos en los modelos de transcripción es el de reducción de ruido. La reducción de ruido o *speech enhancement* se basa en la separación de la onda acústica relevante del “clutter” u onda no relevante para nuestro modelo de transcripción, como puede ser el ruido meteorológico causado por el viento y la mar. De esta forma, el audio introducido en el mecanismo de transcripción posterior realizará su labor con una mayor precisión y menor número de errores. Es importante que los métodos empleados para este propósito busquen un equilibrio entre reducción del *clutter* y distorsión del habla, de forma que el hablante sea lo más legible posible [45].

Entre los modelos y sistemas más comunes de reducción de ruido se pueden encontrar los que a continuación se detallarán:

2.4.1 Filtro Wiener

El filtrado de Wiener [46] es un estándar de la industria para la eliminación de ruidos en el audio y se utiliza ampliamente en Smartphones y dispositivos de comunicación. Este filtrado requiere tanto del sonido ruidoso como de una muestra de ruido presente en el discurso.

El filtro de Wiener encuentra una estimación estadística del audio limpio a partir del audio ruidoso para minimizar el error cuadrático medio bajo ciertas suposiciones sobre el ruido.

Este filtro es útil en el caso de los Smartphones, donde podemos tener dos micrófonos, uno para grabar nuestro discurso con ruido y otro solo para el ruido. (Normalmente, el micrófono de la parte inferior es para grabar a la persona y el micrófono de la parte superior es para grabar el ruido).

Por lo tanto, el audio capturado por ambos micrófonos puede representarse como:

$$x(t) = s(t) + n(t) \quad (\text{Ec 1})$$

Donde $s(t)$ es la señal de la persona y $n(t)$ la contribución del ruido [46].

Para obtener entonces la señal $s(t)$ limpia, es tan simple como realizar la siguiente operación:

$$w = \frac{E[s(t)^2]}{E[x(t)^2]} \quad (\text{Ec 2})$$

Siendo w la derivada de la señal limpia entre la de la total. Esta relación se utiliza para determinar qué parte de la señal ruidosa se asigna a la señal original y cuánto se asigna a la muestra de ruido.

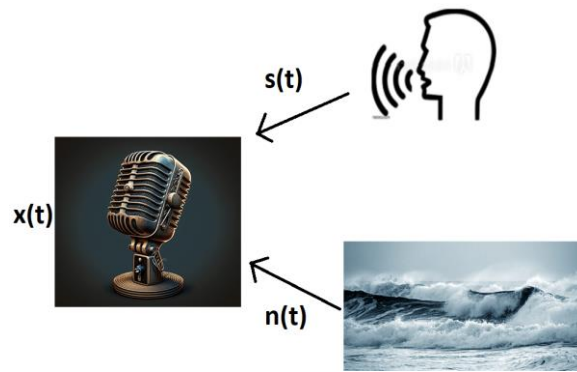


Figura 2-13: Esquema ilustrativo Filtro Wiener. Fuente: propia

2.4.2 Filtros pasa alto, pasa bajo y pasa banda

Un filtro pasa banda es aquel que permite el paso de un rango de frecuencias y la eliminación o atenuación del resto. Existen, además, filtros que simplemente atenúan las frecuencias por encima o debajo de un límite. Estos serían conocidos como los filtros “pasa bajos” o “pasa altos”.

El filtro *Butterworth* es uno de los tipos de filtro pasa bajo diseñado para proporcionar una curva de corte lo más plana posible, dando el máximo control sobre las frecuencias que pasan a través de él. A diferencia de otros filtros, no tiene la opción de usar un filtro pasa alto. Sin embargo, tiene una respuesta más plana, lo que le da un carácter diferente [47]. En la Figura 2-14 se muestra una aplicación del filtro sobre una señal de audio para la frecuencia de 600 Hz.

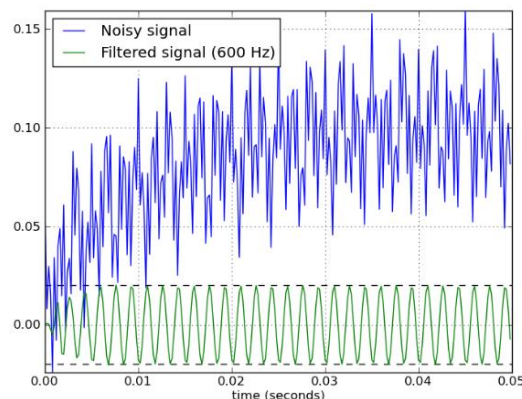


Figura 2-14: Ejemplo filtrado Butterworth. Fuente: [48]

2.4.3 Atenuación adaptativa

El nombre "filtros adaptativos" se puede enfatizar considerando el sistema que intenta ajustarse para responder a algún fenómeno circundante, es decir, ajusta sus parámetros con el objetivo de cumplir con algún objetivo definido que depende del estado del sistema y su entorno. Además, se necesita un procedimiento para llevar a cabo el proceso de adaptación.

Una de las formas de realizar una atenuación adaptativa de un audio es operando por bloques de muestras, atenuando cada bloque en base a su energía media (valor medio del módulo al cuadrado). De esta forma los bloques con mayor energía se ven amplificados (de manera relativa) con respecto a los que tiene menor energía (que se asume son los ruidosos). Por lo tanto, la adaptación dependerá de la energía media de los bloques del sistema, y en función del conjunto de datos utilizado, variará lo que es considerado "bloque ruidoso".

Existen otros métodos de atenuación adaptativa, como puede ser el método Kernel [49], aunque no se abordará su funcionamiento debido a su ineficacia en el entorno de grabación del trabajo.

2.4.4 Puerta espectral o Noisereduce

Trabajo realizado por Tim Sainburg, Marvin Thielk y Timothy Q. Gentner [50]. Su proyecto Noisereduce usa un algoritmo de reducción de ruido en Python que reduce el ruido de señales en el dominio temporal. Se basa en un método llamado "puerta espectral" que es una forma de puerta de ruido [51]. Funciona calculando un espectrograma de una señal (y opcionalmente una señal de ruido) y estimando un umbral de ruido (o puerta) para cada banda de frecuencia de esa señal/ruido. Ese umbral se utiliza para calcular una máscara, que bloquea el ruido por debajo del umbral variable en frecuencia.

La aplicación del proyecto se encuentra disponible para Python mediante la librería Noisereduce [52]

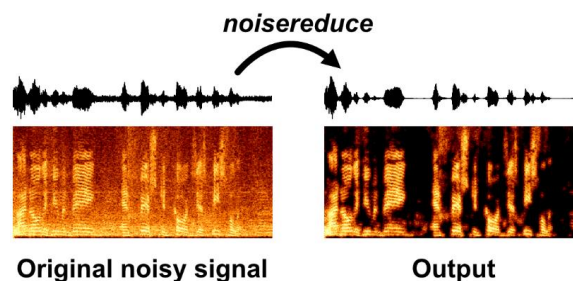


Figura 2-15: Ejemplo de filtrado de audio con librería noisereduce. Fuente: [52]

2.4.5 Proyecto RNNoise

Trabajo realizado por Jean-Marc Valin a partir del estudio de redes neuronales aplicadas a la reducción de ruido [53]. Su trabajo se basa en que a pesar de que la supresión de ruido es un área madura en el procesamiento de señales, todavía es altamente dependiente de la afinación fina de los algoritmos y parámetros del estimador. En su trabajo, demuestra un enfoque híbrido de DSP (procesamiento digital) y *Deep Learning* para la supresión de ruido. Se enfoca en mantener la complejidad lo más baja posible, intentando conseguir una alta calidad en el audio mejorado. Se utiliza una red neuronal recurrente de *Deep Learning* con cuatro capas ocultas para estimar las ganancias ideales de banda crítica, mientras que un filtro de tono más tradicional atenúa el ruido entre las armonías de tono.

El proyecto utiliza una red neuronal recurrente (RNN) [54] con una *gated recurrent unit* (GRU) [55], en lugar de una red neuronal simple o una LSTM [56, 37]. Tal y como se puede ver en la Figura

2-16, las GRU utilizan una estructura de puertas para regular el flujo de información y controlar la memoria a largo plazo, mientras que las RNN utilizan una sola capa de neuronas recurrentes para mantener la memoria. Esto hace capaces a las GRU de aprender automáticamente cuándo deben olvidar información antigua y cuándo deben agregar nueva información a la memoria.

Las RNN son un tipo de red neuronal que se utiliza para procesar secuencias de datos, donde cada elemento de la secuencia tiene una relación con los elementos anteriores. LSTM (Memoria a Corto y Largo Plazo) y GRU (Unidades Recurrentes con Puertas) son dos tipos de RNN que se utilizan para procesar secuencias de datos. Ambas redes son similares en que tienen conexiones de retroalimentación, lo que les permite procesar secuencias completas de datos en lugar de solo puntos de datos individuales.

En general, GRU es una versión simplificada de LSTM que utiliza menos parámetros y es más fácil de entrenar. La retroalimentación en cancelación de ruido es útil para mantener el contexto y la información previa de la señal a medida que se procesa y elimina el ruido. Esto permite que el sistema de cancelación de ruido recuerde la información de la señal previa y la utilice para mejorar la eliminación del ruido en la siguiente iteración.

En comparación con los enfoques *end-to-end* que utilizan una red neuronal de *Deep Learning* para abordar un problema completo, el enfoque híbrido de RNNoise mantiene todo el procesamiento de señales básico necesario y permite a la red neuronal aprender las partes complicadas que requieren ajustes constantes. Este enfoque es más eficiente en términos de recursos y está diseñado para la comunicación en tiempo real en lugar de reconocimiento de voz [57].

Este proyecto se encuentra disponible para implementación en Python mediante la librería `Rnnoise` [58].

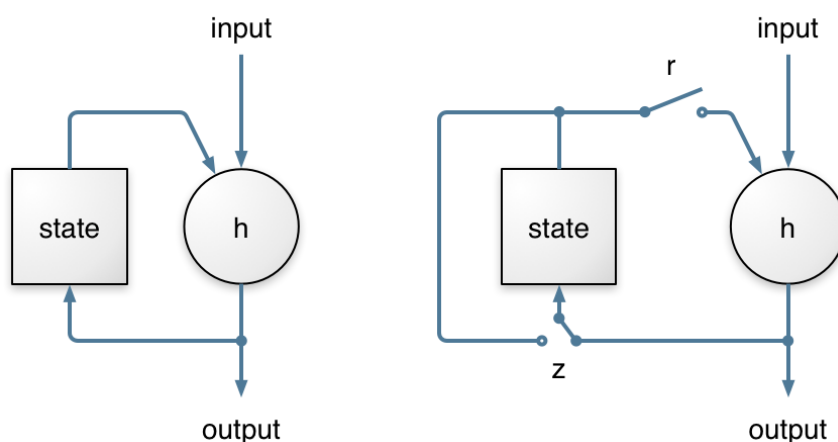


Figura 2-16: Comparativa de RNN y GRU. Fuente: [57]

2.5 Modelos y algoritmos de transcripción de audio

A continuación, se mencionarán los modelos y algoritmos más famosos y empleados en la transcripción de audio a texto, así como su funcionamiento. Este apartado servirá como apoyo al entendimiento de la arquitectura de los sistemas empleados en las librerías de transcripción que abordadas en el capítulo 2.6.

2.5.1 Modelo oculto de Markov (HMM)

Una vez con los archivos de audio listos para su procesamiento, se puede afrontar el problema de la transcripción con distintos algoritmos y modelos, o con el uso de una combinación de estos. El más común y precursor de las redes neuronales actuales es el modelo oculto de Markov (HMM, por sus siglas en inglés) [59].

Un Modelo Oculto de Markov (HMM) es un modelo probabilístico que se utiliza para representar y analizar secuencias de observaciones, como audio. En un sistema de reconocimiento de voz, por ejemplo, un HMM puede ser entrenado en una variedad de hablantes y luego utilizado para identificar el hablante en una nueva secuencia de audio. Los HMMs asignan probabilidades a las secuencias de observaciones y a los estados ocultos subyacentes que generan las observaciones, y se utilizan comúnmente para tareas de clasificación y reconocimiento [60].

Este enfoque se basa en la suposición de que una señal de habla, cuando se ve en una escala de tiempo lo suficientemente corta (digamos, diez milisegundos), puede ser razonablemente aproximada como un proceso estacionario, es decir, un proceso en el que las propiedades estadísticas no cambian con el tiempo.

En un HMM típico, la señal de habla se divide en fragmentos de 10 milisegundos. El espectro de potencia de cada fragmento, que es esencialmente una trama de la potencia de la señal como función de la frecuencia, se mapea a un vector de números reales conocido como coeficientes cepstrales. La dimensión de este vector suele ser pequeña, a veces tan baja como 10, aunque los sistemas más precisos pueden tener una dimensión de 32 o más. La salida final del HMM es una secuencia de estos vectores.

Para decodificar la voz en texto, se comparan grupos de vectores con uno o más fonemas, la unidad fundamental del habla. Este cálculo requiere un proceso complejo y entrenamiento, ya que el sonido de un fonema varía de hablante a hablante e incluso varía de una emisión a otra del mismo hablante. Luego se aplica un algoritmo especial para determinar la palabra o palabras más probable que produzcan la secuencia dada de fonemas.

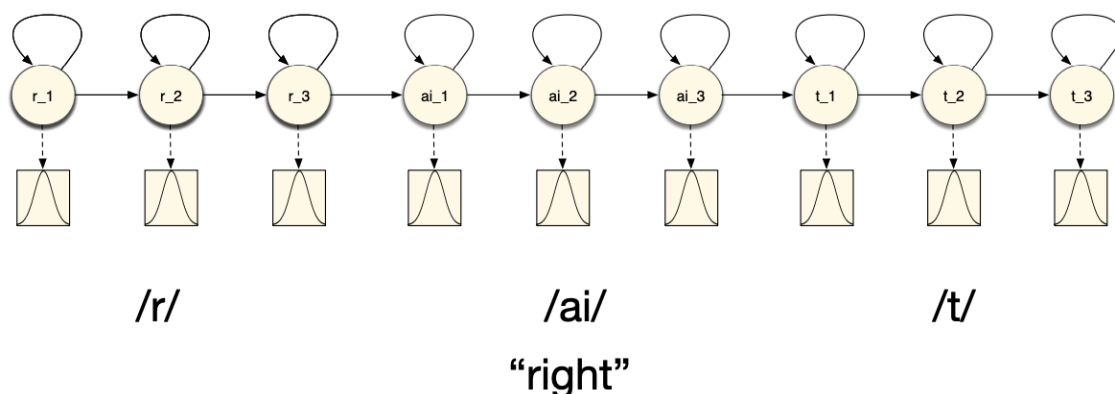


Figura 2-17: Ejemplo HMM con palabra right. Fuente: [61]

Estos procesos pueden observarse en el ejemplo de la Figura 2-17, donde una señal de audio se ha descompuesto en fragmentos a los que se han asignado probabilidades y asociado a fonemas, obteniendo así la palabra “right” del audio.

Se puede imaginar que todo este proceso puede ser computacionalmente costoso. En muchos sistemas modernos de reconocimiento de voz se utilizan detectores de actividad vocal (VAD) para

reducir una señal de audio solo a las partes que probablemente contengan habla, lo que evita que el reconocedor desperdicie tiempo analizando partes innecesarias de la señal. [62].

Hoy en día, gracias a otros modelos como los modelos de mezcla Gaussiana (GMM), es posible incluso identificar al hablante por su acento y tono [63], cosa que el modelo HMM no es capaz de hacer.

2.5.2 Algoritmo CTC

Cuando se realiza una transcripción a texto mediante la red neuronal CNN explicada en la sección 2.3.3, se tiene un problema, conocido como el problema de conexión temporal del contexto.

El problema CTC se refiere a la dificultad que tiene la red neuronal para alinear correctamente las etiquetas de salida con las entradas de audio correspondientes. En otras palabras, cuando la red neuronal CNN transcribe el audio en texto, puede haber momentos en que la transcripción no coincide con lo que se ha dicho en el audio, lo que puede generar errores y reducir la precisión del modelo.

Nuestro objetivo final es asignar momentos o "fotogramas" a personas individuales de la transcripción.

Para ello, el modelo de red neuronal decodifica las probabilidades de caracteres para producir la salida final, pero para un espectrograma particular no sabemos ni cuántos fotogramas hay ni cuáles son sus límites.

Como se puede ver en la Figura 2-18, en la señal de la palabra "Good" de la derecha se necesita un método para calcular los fotogramas y asignar cada uno a un carácter con una cierta probabilidad de acierto, de forma que se pueda conocer el comienzo y fin de cada palabra.

El audio y las imágenes de espectrogramas no están presegmentados para darnos esta información, por lo que necesitamos alguna forma para obtener esa asignación de fotogramas a transcripción [64].

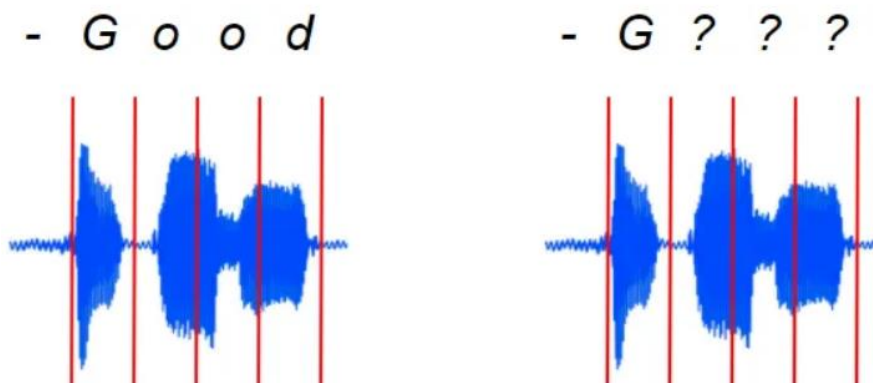


Figura 2-18: Ejemplo de alineamiento necesario y situación antes del uso de CTC. Fuente: [64]

El algoritmo CTC se utiliza para alinear las secuencias de entrada y salida cuando la entrada es continua y la salida es discreta, y no hay claros límites de elementos que se puedan usar para asignar la entrada a los elementos de la secuencia de salida.

Para ello, el audio continuo se corta en marcos discretos y se trasladan a la RNN (Red Neuronal Recurrente), tal y como aparece representado en la Figura 2-19. El trabajo del algoritmo CTC es tomar estas probabilidades de caracteres y derivar la secuencia correcta.

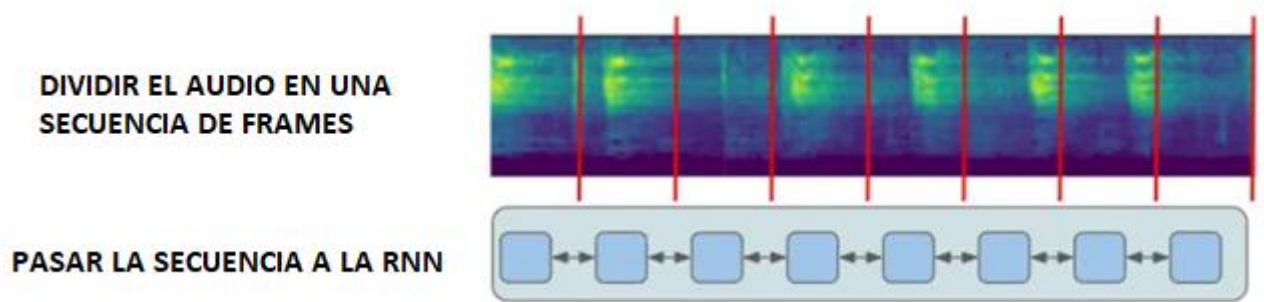


Figura 2-19: División del audio continuo en frames discretos para procesamiento RNN. Fuente: [64]

Para ayudar a manejar el desafío del alineamiento y aparición de caracteres repetidos, introduce el concepto de un carácter "en blanco" (denotado por "-") en el vocabulario. Por lo tanto, las probabilidades de caracteres generadas por la red también incluyen la probabilidad del carácter en blanco para cada marco.

Hay que tener en cuenta que un espacio en blanco no es lo mismo que un espacio. Un espacio es un carácter real mientras que un blanco significa la ausencia de cualquier carácter, algo así como un "nulo" en la mayoría de los lenguajes de programación. Se utiliza solo para demarcar el límite entre dos caracteres. CTC funciona en dos modos que pasamos a detallar a continuación. El primero, *CTC Decoding*, es comúnmente usado durante el entrenamiento de modelos de lenguaje; mientras que el segundo, *CTC Loss*, es empleado en la transcripción de audios.

2.5.2.1 CTC Decoding

Tiene como base una transcripción y trata de entrenar la red para maximizar la probabilidad de producir esa transcripción correcta.

Se utilizan las probabilidades de caracteres para elegir el carácter más probable para cada fotograma, incluidos los espacios en blanco.

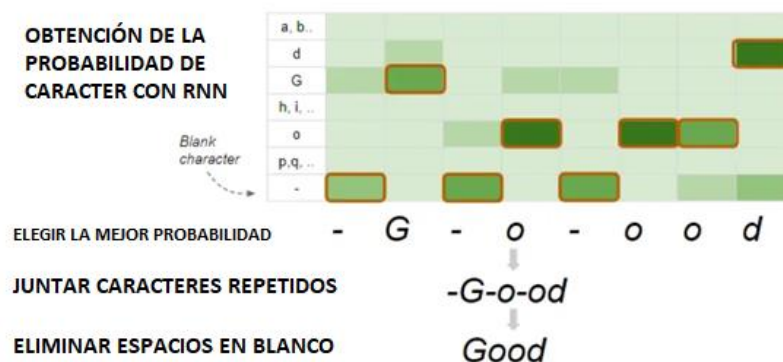


Figura 2-20: Algoritmo de decodificación CTC. Fuente: [64]

En su funcionamiento, como se puede observar en la Figura 2-20, tras la obtención de probabilidades de carácter en el espectrograma con RNN, se combinan los caracteres que se repiten y no están separados por un espacio en blanco. Por ejemplo, podemos combinar las "oo" en una sola "o", pero no podemos combinar las "o-oo".

De esta manera, CTC es capaz de distinguir que hay dos "o" separadas y producir palabras con caracteres repetidos, como por ejemplo "G-o-od". Finalmente, ya que los espacios en blanco han cumplido su función, se eliminan todos los caracteres de espacio en blanco.

2.5.2.2 CTC Loss

Aquí no tenemos una transcripción como base a la que referimos y tenemos que predecir la secuencia más probable de caracteres.

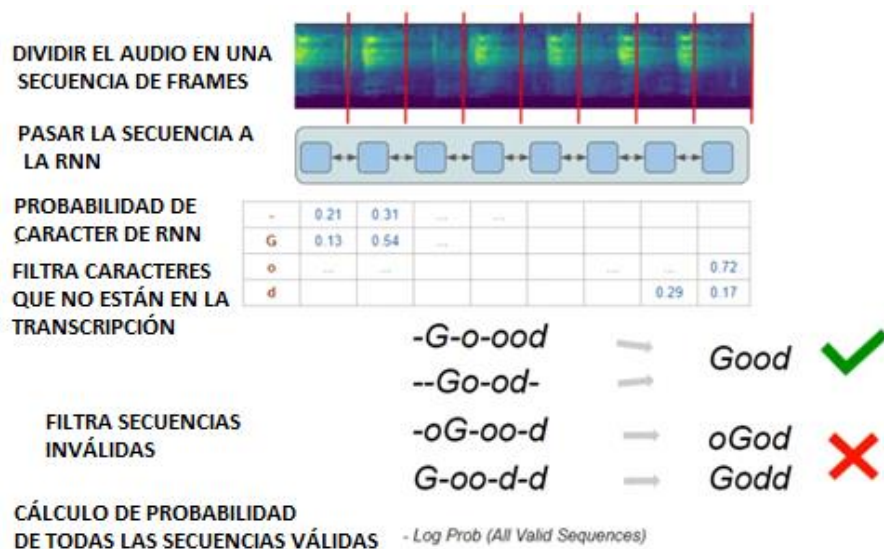


Figura 2-21: Algoritmo de CTC Loss. Fuente: [64]

La pérdida se calcula como la probabilidad de que la red prediga la secuencia correcta. Para hacer esto, el algoritmo lista todas las secuencias posibles que la red puede predecir y, a partir de eso, selecciona el subconjunto que coincide con la transcripción objetivo.

Para identificar ese subconjunto del conjunto completo de secuencias posibles, el algoritmo reduce las posibilidades de la siguiente manera, tal y como aparece en el ejemplo de la Figura 2-21:

- Manteniendo solo las probabilidades de los caracteres que ocurren en la transcripción base y descartando el resto. En nuestro ejemplo, solo mantiene las probabilidades para "G", "o", "d" y "-".
- Usando el subconjunto filtrado de caracteres, para cada fotograma, seleccionando solo aquellos caracteres que ocurren en el mismo orden que la transcripción base. Por ejemplo, aunque "G" y "o" son caracteres válidos, un orden de "Go" es una secuencia válida mientras que "oG" es una secuencia inválida.

Con estas restricciones en su lugar, el algoritmo ahora tiene un conjunto de secuencias de caracteres válidos, todas las cuales producirán la transcripción correcta. Por ejemplo, utilizando los mismos pasos que se utilizaron durante la inferencia, " -G-o-ood " y " - Go-od- " ambos resultarán en una salida final de "Good".

Luego, utiliza las probabilidades de caracteres individuales para cada fotograma para calcular la probabilidad general de generar todas esas secuencias válidas. El objetivo de la red es aprender cómo maximizar esa probabilidad y, por lo tanto, reducir la probabilidad de generar cualquier secuencia inválida.

En términos estrictos, ya que una red neuronal minimiza la pérdida, la pérdida CTC se calcula como el logaritmo negativo de la probabilidad de todas las secuencias válidas. A medida que la red minimiza esa pérdida a través de la retropropagación durante el entrenamiento, ajusta todos sus pesos para producir la secuencia correcta [64].

2.6 Herramientas de transcripción de audios

2.6.1 Whisper OpenAI

Whisper es un sistema Open Source de reconocimiento de voz automático (ASR) entrenado con 680.000 horas (77 años) de datos supervisados multilingües recolectados de la web. El uso de un conjunto de datos tan grande y diverso conduce a una mayor robustez en identificación de acentos, ruido de fondo y lenguaje técnico. Además, permite la transcripción en varios idiomas, así como la traducción de esos idiomas al inglés, debido a los diferentes lenguajes con los que está entrenado [65].

La arquitectura de Whisper es un enfoque sencillo de extremo a extremo, implementado como una red neuronal igual a las que llevan existiendo desde hace más de 5 años en el mundo de la transcripción de audios [66]. Lo que hace a Whisper “especial” está en la cantidad de datos y como han estructurado su entrenamiento.

Respecto a su funcionamiento, el audio de entrada se divide en fragmentos de 30 segundos, se convierte en un espectrograma y luego se pasa a un codificador. Un decodificador se entrena para predecir el subtítulo de texto correspondiente, que mezclado con los llamados tokens especiales (secuencias numéricas para funciones específicas) dirigen al modelo único a realizar tareas como la identificación de idiomas, marcas de tiempo a nivel de frase, transcripción de discurso multilingüe y traducción de discurso a inglés.

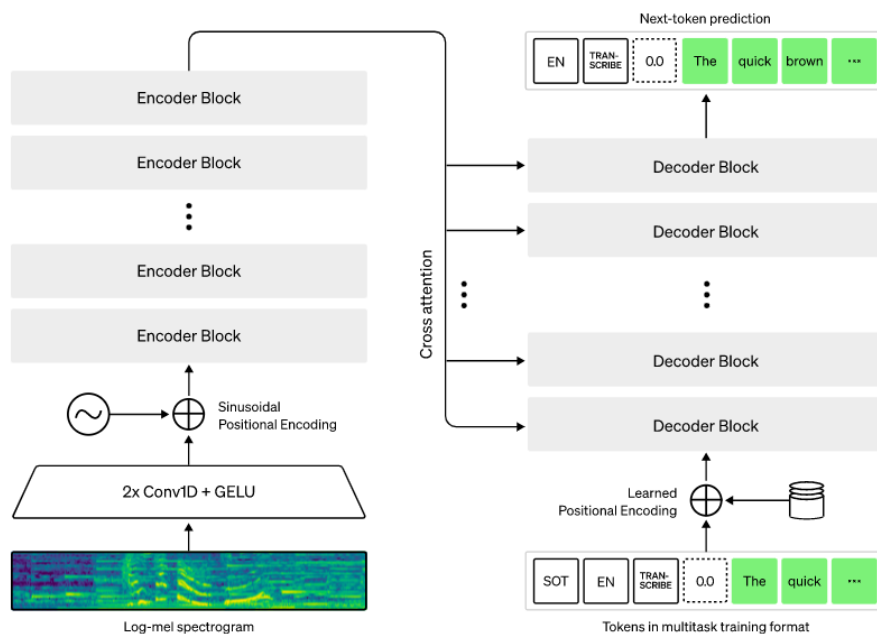


Figura 2-22: Esquema de funcionamiento Whisper. Fuente: [65]

Debido a que Whisper se entrenó con un conjunto de datos grande y diverso, que no se ajustó a ninguno específico, no supera a otros modelos que usan vocabulario especializado en un ámbito específico. Sin embargo, al medir el rendimiento de Whisper a través de muchos conjuntos de datos

diversos, se ve que es mucho más robusto y comete un 50% menos de errores que otros modelos. Aproximadamente un tercio del conjunto de datos de audio de Whisper es en idiomas distintos al inglés. [67]

Whisper además incluye 5 modelos Open Source en función del nivel de precisión deseado, reflejados en la Figura 2-23. En los modelos superiores la precisión obtenida será mayor, pero el tiempo de procesamiento y peso del modelo se verán afectados [65]

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	<code>tiny.en</code>	<code>tiny</code>	~1 GB	~32x
base	74 M	<code>base.en</code>	<code>base</code>	~1 GB	~16x
small	244 M	<code>small.en</code>	<code>small</code>	~2 GB	~6x
medium	769 M	<code>medium.en</code>	<code>medium</code>	~5 GB	~2x
large	1550 M	N/A	<code>large</code>	~10 GB	1x

Figura 2-23: Modelos de Whisper. Fuente: [65]

Para su implementación en el entorno de trabajo Python de Pycharm, existe la librería `Pywhisper`, que permite la utilización offline de Whisper una vez descargado el modelo a utilizar.



Figura 2-24: Librería Pywhisper. Fuente: [68]

2.6.2 Vosk

Vosk es una herramienta de reconocimiento de voz, cuyos aspectos más destacables son [69]:

- Soporta más de 20 idiomas y dialectos, como inglés, inglés indio, alemán, francés, español, portugués, chino, ruso, turco, vietnamita, italiano, neerlandés, catalán, árabe, griego, farsi, filipino, ucraniano, kazajo, sueco, japonés, esperanto, hindi, checo, polaco y uzbeko. Se agregarán más en el futuro.
- Funciona sin conexión, incluso en dispositivos de baja potencia como Raspberry Pi, Android e IOS.
- Se instala fácilmente.
- Los modelos por idioma son solo 50Mb cada uno, pero hay modelos de servidor más grandes disponibles.
- También hay enlaces para diferentes lenguajes de programación, como Java, C# y JavaScript.
- Permite una reconfiguración rápida del vocabulario para obtener la mejor precisión.
- Además de reconocimiento de voz simple, también soporta la identificación del orador.



Figura 2-25: Librería Vosk. Fuente: [69]

Por otra parte, Vosk se basa en un método de redes neuronales o *Deep Learning*. Para transcribir audio, Vosk primero divide el audio de entrada en fragmentos más pequeños y luego los procesa de forma secuencial para producir transcripciones precisas, como hacía Whisper. Además, utiliza técnicas de lenguaje para mejorar la precisión, como el modelado del contexto.

2.6.3 Kaldi

Kaldi es un toolkit de código abierto especializado en tareas de reconocimiento de voz. El proyecto Kaldi comenzó en 2009 en un taller en la Universidad Johns Hopkins, con el objetivo de crear un sistema de reconocimiento de voz de alta calidad a bajo costo para nuevos idiomas y entornos. En 2010, los participantes del taller se reunieron en Brno, República Checa, para desarrollar una herramienta de propósito general que fue presentada en 2011. Entre los creadores de Kaldi se encuentran expertos de diferentes instituciones como Microsoft Research, Saarland University, Centre de Recherche Informatique de Montréal, Brno University of Technology y Tsinghua University [70].

Para su funcionamiento utiliza las bibliotecas OpenFst y BLAS/LAPACK de álgebra numérica, donde destacan los modelos HMM, SGMM [71] y GMM [72] utilizados para la representación de fonemas y búsqueda de palabras. La recopilación de todas las bibliotecas empleadas en Kaldi aparecen representadas en la Figura 2-26.

Además, incorpora un módulo de detección de palabras clave, Kaldi KWS, el cual es usado por Vosk en la búsqueda y análisis de flujos de audio.

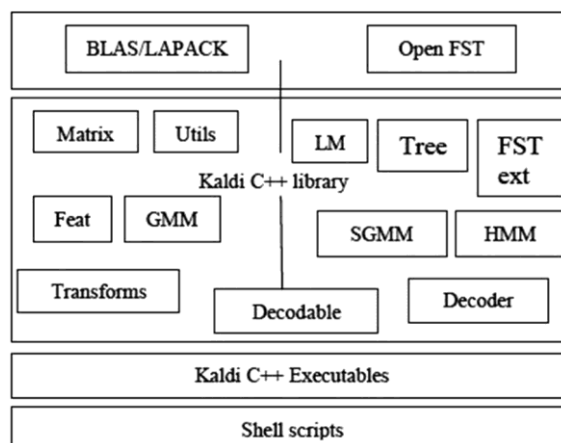


Figura 2-26: Esquema componentes de Kaldi. Fuente: [70]

Kaldi KWS es un módulo implementado para el sistema de reconocimiento de voz Vosk cuyo objetivo es la búsqueda de palabras clave (*Keyword Spotting*) [73]. Para esto, realiza la indexación de estructuras llamadas celosías generadas por el sistema de reconocimiento de voz LVCSR (*Large Vocabulary Continuous Speech Recognizer*) del sistema original. Además, consta de un sistema para la

obtención rápida de palabras clave con la resolución de problemas de vocabulario (OOV, *out-of-vocabulary*) [74].

El sistema LVCSR extrae características acústicas dimensionales y realiza un procesamiento acústico mediante HMM y SAT (*Speaker Adaptive Training*) usando un modelo gaussiano (SGMM, *Singular Gauss-Markov Model*), para obtener transcripciones de una señal de audio. [75].

Las celosías generadas se procesan mediante técnicas de indexación para luego almacenar la información sobre tiempo de inicio-fin y probabilidad posterior de celosía como un coste. Estos datos se pueden comparar con un índice invertido que permite buscar menciones de palabras clave.

Si una palabra clave no está en el vocabulario LVCSR, no aparecerá en las celosías de búsqueda. Para solucionar este problema, Kaldi KWS utiliza un sistema de subpalabras que busca palabras del vocabulario acústicamente similares a las OOV para usarlas como proxy de la palabra clave sustituyendo a la original, lo que también mejora el rendimiento en palabras clave que sí están en el vocabulario LVCSR [70].

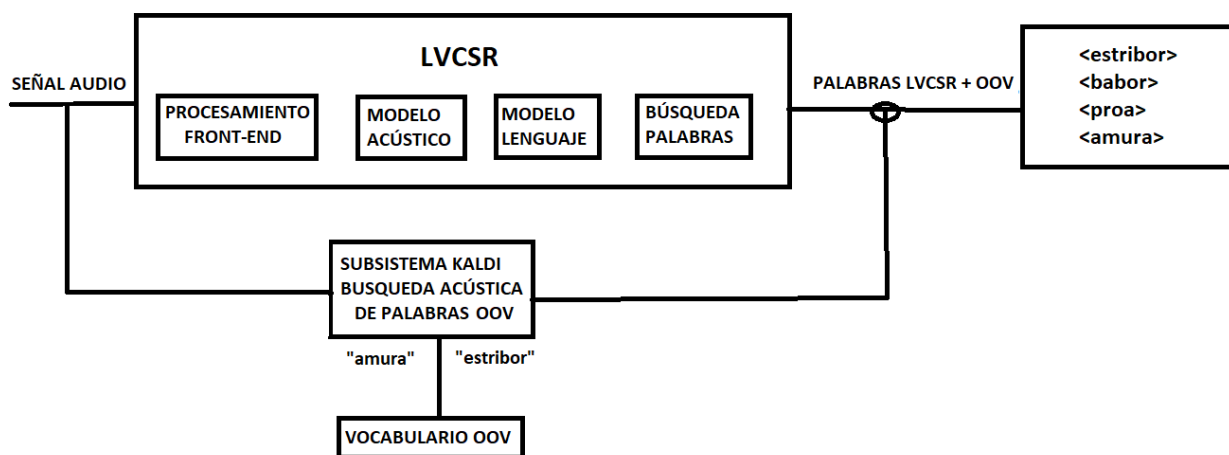


Figura 2-27: Esquema KALDI KWS. Fuente: propia

En la Figura 2-27 se puede ver el proceso anteriormente descrito, observándose la posibilidad de reconocer palabras que el sistema LVCSR no recoge gracias al vocabulario externo introducido.

En resumen, Kaldi sirve como ayuda al sistema de reconocimiento de voz de Vosk a la hora de identificar palabras de un vocabulario externo no recogido por el sistema de reconocimiento de audio usado. Estas palabras son marcadas en la transcripción obtenida.

2.6.4 Pocketsphinx

Pocketsphinx es una librería de Python que brinda servicios de reconocimiento de voz en tiempo real. Para funcionar, requiere la instalación de la biblioteca Sphinxbase [76].

La creación de Pocketsphinx proviene del proyecto Sphinx original desarrollado por la Universidad Carnegie Mellon en 2000, que consistía en una serie de aplicaciones de reconocimiento de voz y un entrenador acústico. La primera versión de Pocketsphinx para Python fue liberada en febrero de 2015 y la última versión es la 5.0.0 lanzada en 2020. En la actualidad, la librería se encuentra en fase alfa de desarrollo, por lo que se esperan futuras mejoras e implementaciones.

El sistema CMU Sphinx utiliza el modelo estadístico ya explicado HMM para reconocer voz. En este sistema, los fonemas (unidades básicas del sonido, en una palabra) se representan mediante una

distribución de muestras que lo componen, creando así un modelo acústico. De este modo, el sistema transforma los fonemas recibidos en vectores que representan las características de la señal y los compara con el modelo acústico previamente entrenado para estimar la probabilidad de que un conjunto de vectores corresponda a un fonema conocido.

El reconocimiento de voz implica el procesamiento de la señal y la comparación con el modelo acústico en tiempo real para encontrar la secuencia de vectores que mejor coincide con la señal de entrada. Este proceso requiere la construcción de un grafo HMM compuesto por fonemas siguiendo un orden específico, que requiere la aplicación de un diccionario que descomponga las palabras en fonemas y un modelo acústico que obtenga el HMM de los fonemas.

Una vez construido el grafo, el sistema compara la secuencia de vectores con las diferentes rutas del grafo para encontrar la mejor coincidencia y generar un resultado con las palabras reconocidas. Estas palabras reconocidas se pueden utilizar para realizar diferentes funciones, como reconocer una palabra dicha por el micrófono y mostrarla en la pantalla en tiempo real o entrenar el sistema para reconocer palabras específicas [76].

La Figura 2-28 representa el funcionamiento interno del sistema partiendo de una entrada de audio hasta la obtención de una salida de palabras, gracias al empleo de los modelos de lenguaje y acústico, así como la extracción de características del espectrograma inicial.

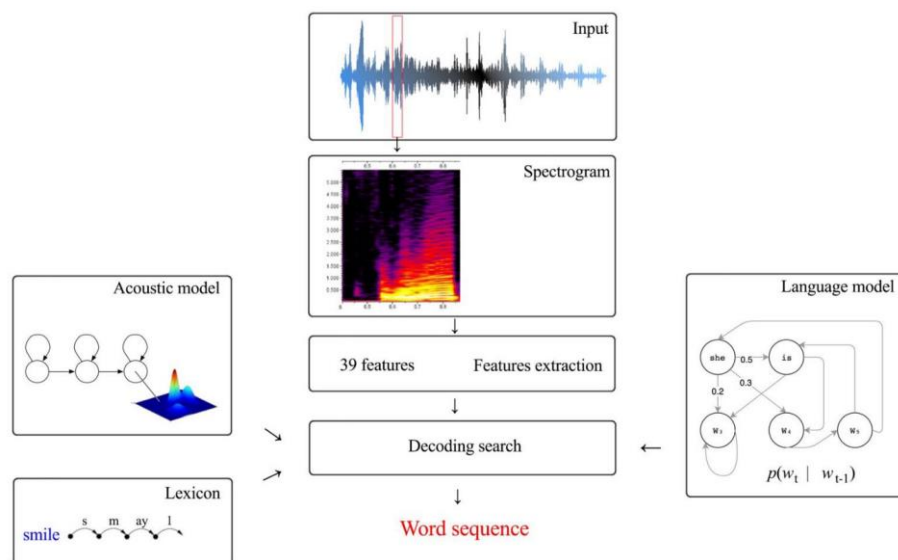


Figura 2-28: Esquema simplificado de funcionamiento Pocketsphinx. Fuente: [77]

2.6.5 Speech Recognition

Speech Recognition es una librería de reconocimiento de voz en Python, con capacidad de uso en tiempo real dependiendo del modelo usado. La versión actual es la 3.9.0, representada en la Figura 2-29. Speech Recognition puede utilizar varios modelos y bibliotecas de reconocimiento de voz, incluyendo, según la versión actual [78]:

- CMU Sphinx (offline),
- Google Speech Recognition,
- Google Cloud Speech API,
- Wit.ai,
- Microsoft Azure Speech,

- Houndify API,
- IBM Speech to Text,
- Snowboy Hotword Detection (offline),
- Tensorflow,
- Vosk API (offline),
- OpenAI whisper (offline).

La librería permite transcribir archivos de audio en formato WAV, FLAC y MP3 a texto. Utiliza una combinación de modelos de procesamiento de lenguaje natural y algoritmos de reconocimiento de voz para convertir el audio en texto. Puede ser utilizada para tareas como transcripción automática, reconocimiento de comandos de voz, conversación por voz y más.

Como se puede observar, entre los modelos que puede utilizar se incluyen la mayoría de los ya comentados. De forma general, se puede decir que esta librería facilita el uso de los modelos existentes y el intercambio entre ellos sin necesidad de descargarlos individualmente y mediante un código común.

La mayor desventaja del uso de esta herramienta reside en la incapacidad del uso de modelos entrenados con datasets propios. Por ello, el usuario debe adaptarse a la calidad de los modelos implementados, teniendo en ocasiones resultados deficientes con el uso de palabras fuera del vocabulario de los modelos o en entornos de ruido.

Hay que destacar la librería `Google Speech Recognition`, capaz de transcribir audio a texto en tiempo real, pero sin capacidad offline. `Google Speech Recognition` fue lanzada en 2012 como parte del módulo de reconocimiento de voz de Google. Desde entonces, ha sido actualizada y mejorada para proporcionar una experiencia de reconocimiento de voz más precisa y fácil de usar. En la actualidad, es una de las bibliotecas de reconocimiento de voz más populares y ampliamente utilizadas [79].



Figura 2-29: Librería Speech Recognition. Fuente: [78]

2.7 Extracción automática de palabras clave

La búsqueda de palabras clave (KI, por sus siglas en inglés) se define como la tarea que identifica automáticamente un conjunto de términos que mejor describen el tema de un documento. Se utiliza diferente terminología para estudiar los términos que representan la información más relevante contenida en el documento: frases clave, segmentos clave, términos clave o simplemente palabras clave. Todos los sinónimos mencionados tienen la misma función: caracterizar los temas discutidos en un documento. Extraer un pequeño conjunto de unidades, compuesto por uno o más términos, de un solo documento es un problema importante en la minería de texto (MT), la recuperación de información (RI) y el procesamiento del lenguaje natural (PLN).

Es importante saber y distinguir entre los dos métodos de búsqueda de palabras clave existentes hoy en día. Las dos categorías son la asignación de palabras clave (*Keyword Assignment*) y la extracción de palabras clave (*Keyword Extraction*). Ambas giran en torno al mismo problema: seleccionar la mejor palabra clave [80].

En la asignación de palabras clave, se eligen palabras de un vocabulario o diccionario controlado de términos y se seleccionan las coincidencias, tal y como hace el módulo Kaldi KWS.

En la extracción de palabras clave, sin embargo, se analiza un documento y las palabras que se mencionan explícitamente en el texto. Las palabras que aparecen en el documento se analizan para identificar las más representativas, normalmente explorando las propiedades de origen (es decir, frecuencia, longitud). Por lo general, la extracción de palabras clave no utiliza un vocabulario predefinido para determinar las palabras, sino que necesita de entrenamiento con textos etiquetados en los que se señalen las palabras importantes. De esta forma, a partir de un texto sería capaz de realizar la selección de forma autónoma, evaluando otros criterios como posibles repeticiones de palabras o frecuencia de aparición.

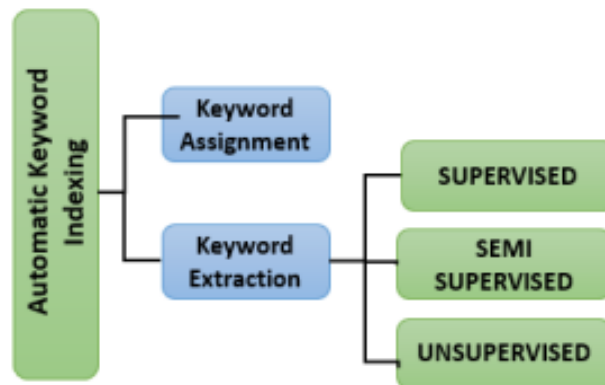


Figura 2-30: Tipos de búsqueda de palabras clave. Fuente: propia

Para la extracción de palabras clave se suelen utilizar sistemas con redes neuronales de Deep Learning. Estos sistemas pueden ser supervisados, no supervisados o semi-supervisados [19], en función del tipo de datos que se han utilizado para su entrenamiento. El sistema abordado en este trabajo será explicado en la sección 3.6, siendo un sistema supervisado con datos de entrenamiento etiquetados.

En la Figura 2-30 se encuentra la representación de un esquema con los tipos de búsqueda de palabras y entrenamientos.

2.8 Herramientas de simulación y representación

2.8.1 SIMDIS

SIMDIS agrupa un conjunto de herramientas elaboradas por Code 5770 en NRL (Laboratorio de investigación Naval de Estados Unidos). Su implementación permite obtener representaciones gráficas en 2D y 3D de datos y simulaciones en vivo.

SIMDIS es un sistema de visualización de simulación que integra múltiples fuentes de datos en una sola imagen. Ha sido utilizado con éxito para la formación y la evaluación y entrenamiento por varias organizaciones militares de los Estados Unidos y extranjeras. Tiene una base de usuarios grande, alrededor de 25,000, y ha sido independientemente certificado y acreditado. Además, permite a los usuarios personalizar y extender el software existente a través de APIs. La versión actual es SIMDIS 10 SR5, lanzada en marzo de 2019 y es compatible con los sistemas operativos Windows 7, 10 y RHEL 6, 7.

SIMDIS puede ser utilizado en visualización de pruebas del sistema de combate, pruebas navales, revisión conjunta después de la acción, monitoreo en tiempo real de la situación durante ejercicios o evaluación del rendimiento de vehículos. Algunas simulaciones se reflejan en la Figura 2-31.

Los requisitos mínimos de hardware para su funcionamiento incluyen un procesador dual-core a 1.6 GHz, 2 GB de RAM, 100 GB de espacio en disco duro y una tarjeta gráfica de 256 MB. Los requisitos recomendados incluyen un procesador quad-core a 3.0 GHz, 8 GB de RAM (16 GB para sistemas de 64 bits), un disco duro sólido de 1 TB y una tarjeta gráfica de 1 GB.

El software del simulador es de código abierto y está disponible para descarga en su sitio web, junto con una mesa de ayuda y una sección para informar problemas y solicitar características adicionales. También está disponible un registro de claves de licencia en el sitio web. [81]

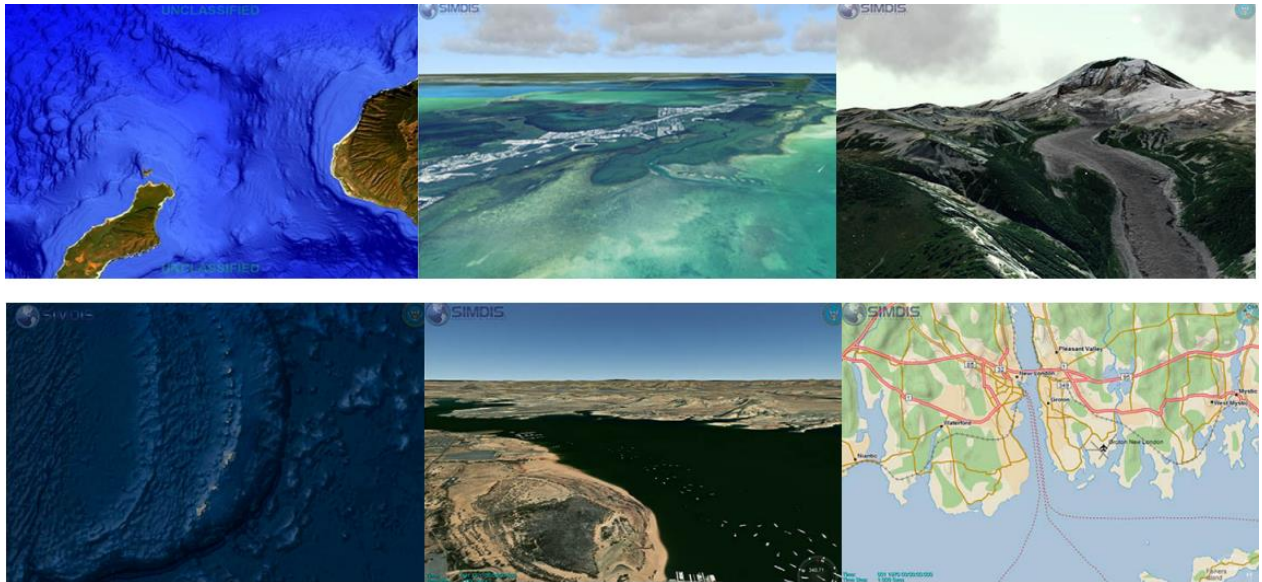


Figura 2-31: Simulaciones SIMDIS. Fuente: [81]

2.8.2 Folium

Folium es una biblioteca de Python que combina la capacidad de manipulación de datos de Python con la funcionalidad de visualización de mapas de la biblioteca leaflet.js. Folium permite enlazar datos con un mapa para visualizaciones de mapas de áreas y también permite incluir visualizaciones de vectores, imágenes, vídeo, GeoJSON y TopoJSON en el mapa. Folium incluye conjuntos de mapas integrados de OpenStreetMap, Mapbox y Stamen y admite conjuntos de mapas personalizados con claves API de Mapbox o Cloudmade.



Figura 2-32: Librería Folium. Fuente: [82]

3 DESARROLLO DEL TFG

En los siguientes apartados se detallará la arquitectura del sistema, profundizando en la elaboración del código Python para cada una de las etapas de grabación, procesamiento de audio, transcripción, búsqueda de órdenes, evaluación y simulación.

Sin embargo, antes de explicar el sistema completo, se procederá a realizar una justificación de las herramientas y librerías empleadas y qué ventajas aportan respecto a otras.

3.1 Lenguaje de programación, librerías y algoritmos empleados

De todos los lenguajes de programación disponibles, se ha elegido Python como entorno para el desarrollo del proyecto por ser el de mayor crecimiento a nivel mundial gracias, entre otras cosas, a la extensa comunidad de la que dispone y a su facilidad a la hora de realizar la interpretación y la escritura de código, como se puede observar en la Figura 3-1. En concreto se ha seleccionado la versión de 3.8 de Python, debido a que las versiones más actuales no son compatibles con algunas de las librerías empleadas.

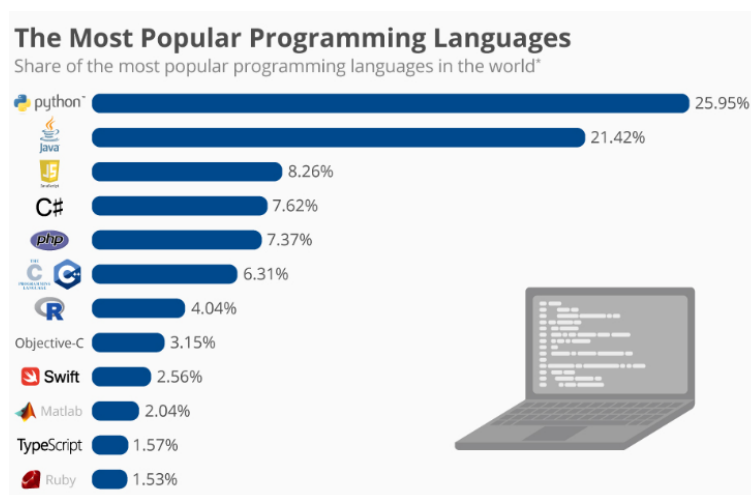


Figura 3-1: Lenguajes de programación más populares. Fuente: [83]

En relación con el sistema de transcripción se ha utilizado dos herramientas en combinación:

- Vosk con Kaldi KWS: El motivo de su elección ha sido que tiene la capacidad, gracias a Kaldi, de realizar la tarea transcripción de una forma superior a otros modelos ya que permite

trabajar con una definición previa de un vocabulario a detectar. Se obtienen solamente las palabras de interés y sin fallos. Esta implementación de Kaldi es una característica de gran interés en entornos de grabación no ideales, como es este caso.

Concretamente, el modelo de Vosk empleado ha sido el “*small*” de español, debido a que el modelo más amplio aún no permite añadir un vocabulario externo mediante Kaldi.

- Whisper: El motivo de su elección es que tiene mejor detección de sonido y transcripción. Sin embargo, no permite su configuración y personalización. Whisper transcribe todo el audio sin la posibilidad de añadir un vocabulario específico, como uno náutico. Es por ello por lo que es quizás menos preciso, y una misma palabra puede ser detectada y transcrita de distintas maneras si la calidad de la grabación no es lo suficientemente elevada.

La combinación de ambos sistemas permite obtener una mejor captura de los audios y un menor error en las transcripciones. La implementación usando el sistema Vosk ha sido obtenido del proyecto PANNACOTA, mientras que el de Whisper ha sido creado e implementado sobre este para conseguir una mejora en los resultados.

Por otra parte, en relación con el sistema de obtención de órdenes, se ha optado por entrenar un sistema que sea capaz de distinguir, a partir de las transcripciones, las órdenes o palabras clave que ahí se encuentren en base a su contexto. Para ello, se han utilizado algoritmo como son el CRF o el BiLSTM, que una vez entrenados, proporcionen buenos resultados, basándose en evaluaciones realizadas en trabajos previos de análisis de sentimientos [84].

Por último, otras librerías utilizadas han sido empleadas al observar un mejor desempeño con respecto a otras utilizadas. Este es el ejemplo de la librería de reducción de ruido, que se ha seleccionado tras probar más de 10 tipos distintos descritos en el apartado 3.4.1. Para cada uno se han realizado evaluaciones de las transcripciones, comparando resultados y llegando a la conclusión de que, a pesar de tener resultados muy similares, los sistemas que hacen uso de RNN suelen tener mejor desempeño.

Además de las librerías relacionadas con la reducción de ruido, se han utilizado otras que han sido de utilidad para el proyecto y que se pasan a detallar a continuación.

3.1.1 Numpy

La librería `Numpy` [85] de Python sirve para trabajar con arreglos y matrices numéricas de forma eficiente y rápida. Proporciona una amplia gama de funciones matemáticas y herramientas para trabajar con estos arreglos, incluyendo operaciones de álgebra lineal, transformadas de Fourier, estadísticas y generación de números aleatorios. Numpy es ampliamente utilizado en la ciencia de datos, el aprendizaje automático, la ingeniería y otras áreas que requieren cálculos numéricos intensivos.

3.1.2 Soundfile

La librería `Soundfile` [86] de Python es una biblioteca que proporciona funciones para leer y escribir archivos de audio en diferentes formatos, incluyendo WAV, FLAC y Ogg Vorbis. Esta librería es útil para trabajar con archivos de audio en Python, lo que permite a los desarrolladores leer y manipular datos de audio para aplicaciones como procesamiento de señales de audio, reconocimiento de voz y análisis de sonido.

3.1.3 Scipy

La librería `Scipy` [48] de Python es una biblioteca muy útil para la computación científica y la manipulación de datos en Python. Ofrece una amplia gama de herramientas para realizar tareas avanzadas como integración numérica, análisis estadístico, procesamiento de señales, procesamiento de

imágenes, resolución de sistemas de ecuaciones lineales, entre otros. Es ampliamente utilizada en diferentes campos científicos y tecnológicos como la física, la ingeniería, la biología, la química y las ciencias sociales. Además, es capaz de utilizar los arrays numéricos obtenidos de la librería Numpy.

3.1.4 SoundPy

La biblioteca SoundPy [87] es utilizada principalmente para el procesamiento de señales de audio. Se puede utilizar para visualizar y filtrar señales de audio antes y después de la extracción de características, con diversos tipos de características, como señales en crudo, STFT, FBANK y MFCC. También puede ser utilizada para la preparación y manipulación de audio, como la conversión de archivos de audio, extracción de características, aumentar o reducir la velocidad y el tono, agregar ruido, filtrar ruido y detectar la actividad de voz en una señal, así como medir la frecuencia dominante y básica en una señal.

3.1.5 Utils

Esta librería [88] incluye una serie de funciones y herramientas útiles para tareas comunes en el desarrollo de software, como el manejo de archivos y directorios, el procesamiento de arrays o cadenas de audio y manipulación de datos general.

3.1.6 Jiwer

La librería Jiwer [89] es una biblioteca de Python que proporciona herramientas para medir la calidad de la transcripción de voz a texto. Para esto emplea funciones que calculan métricas comunes de evaluación, como la tasa de error de palabras WER, la tasa de error de caracteres CER, y la tasa de error de secuencias SER. Además, también proporciona herramientas para calcular métricas de alineación de texto, como la precisión de la alineación y la medición de los tiempos de inicio y fin.

3.2 Arquitectura del sistema



Figura 3-2 : Arquitectura general de sistema. Fuente: propia

La arquitectura general del sistema está dividida en dos fases. La primera es la toma de muestras realizada *in situ* mediante la grabación del oficial en el puente de mando. La segunda se basa en el procesamiento posterior de todos los datos obtenidos en la primera parte, hasta llegar a la obtención de las órdenes y evaluación de la calidad de la transcripción. Por lo tanto, es importante aclarar que no se trata de un sistema en tiempo real.

En los posteriores apartados se detallarán las 6 partes que componen las fases, desde la grabación (fase 1); hasta la transcripción, búsqueda de órdenes, simulación y evaluación (fase 2).

El diagrama de funcionamiento del sistema que se procederá a explicar es el de la Figura 3-3. En él, las 6 partes se encuentran señaladas en cajas, siendo las dos primeras las obtenidas del proyecto PANNACOTA, a partir de las cuales se ha construido el resto del sistema.

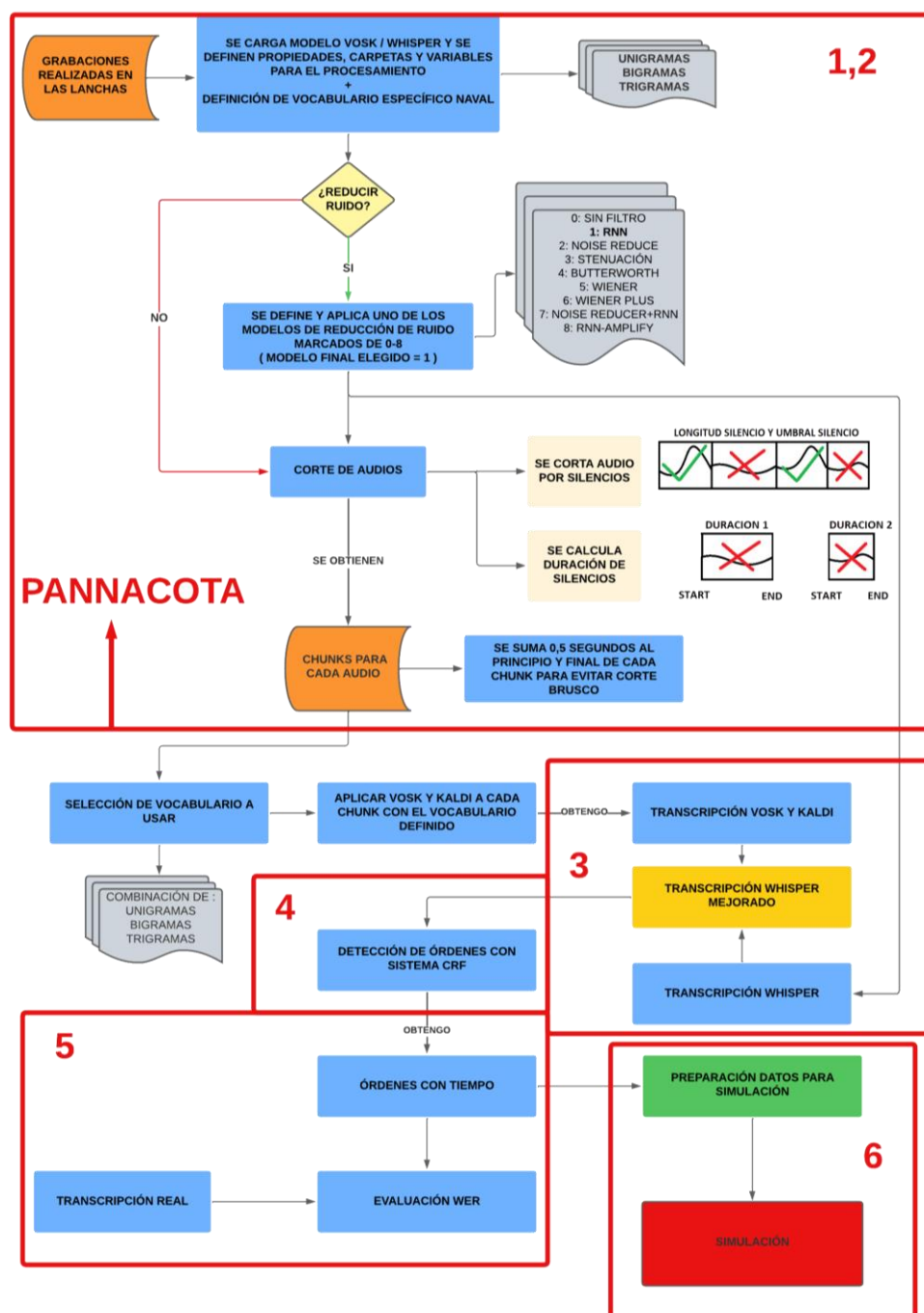


Figura 3-3:Diagrama de la fase de procesamiento del sistema completo. Fuente: propia

3.3 Proceso de grabación

Con el fin de recoger audio en las lanchas de instrucción, se ha decidido emplear una Raspberry Pi como dispositivo hardware y conectarla, en un caso, a un micrófono de ambiente ubicado en el panel de control del puente mando, y en otro, a un micrófono Bluetooth portátil de solapa colocado en el alumno encargado del ejercicio en la lancha (Figura 3-4 y Figura 3-5). Todos los sonidos capturados se guardan en fragmentos temporales de dos minutos en un formato WAV de modulación de código de pulso mono (PCM) y con una frecuencia de 16000Hz. Este proceso de fragmentación se lleva a cabo tanto para evitar pérdidas en la captura de audio en caso de un error durante la grabación de algún fragmento, como para evitar el procesado *in situ* debido a la alta carga computacional que ejerce el proceso de transcripción [3].

Tras la realización de las pruebas de transcripción realizadas, finalmente se han escogido las transcripciones del micrófono de solapa para el sistema debido a su mejor relación señal-ruido. El micrófono ambiente a pesar de proporcionar buenos resultados cuando el oficial está cerca de él, captura con la misma intensidad sonora las distintas voces dentro del puente de mando, lo que dificulta la tarea de transcripción posterior.



Figura 3-4: Instalación del sistema en lancha junto micrófono ambiente. Fuente: propia.



Figura 3-5: Colocación de micrófono de solapa. Fuente: propia.

Para el proceso de grabación cada dos minutos la Raspberry utiliza un *crontab* que ejecuta automáticamente el código de grabación, junto con una tarjeta SD de 64 GB para guardar los datos de los audios. El *crontab* es un proceso en segundo plano que le permite a la Raspberry Pi ejecutar scripts en momentos específicos mientras se encuentre encendida.

3.4 Procesado de audio

3.4.1 Reducción de ruido

La utilización de ambos micrófonos viene motivada por la necesidad de evitar una degradación del rendimiento del sistema en presencia de ruido. Para combatir este efecto, existen varios métodos, entre los que se encuentran:

- 1- La realización de un paso de procesamiento previo del audio que reduzca el ruido y las distorsiones (conocido como *speech enhancement*)
- 2- Adaptando los parámetros de los modelos de transcripción de voz en entornos ruidosos.

Dado lo complicado y variado del entorno contemplado en este proyecto, parece razonable inclinarse por la primera opción, ya que proporciona una solución más flexible. Así, en este caso concreto, se aplicaron técnicas de eliminación de ruido (*denoising*) en los audios capturados. El objetivo no es otro que tratar de eliminar/atenuar el ruido ambiente debidos al motor, oleaje o el mal tiempo. Las técnicas probadas han sido variadas: atenuación adaptativa, aplicación de filtro de paso banda de Butterworth, filtro de Wiener o reducción de ruido mediante la aplicación de técnicas de aprendizaje profundo y redes neuronales RNN, siendo esta una de la más efectivas.

A continuación, se procederá a detallar la implementación de todos los filtros utilizados:

- RNN: Mediante la librería `Rnnoise` [58].
- Noise Reduce: Mediante la librería `Noisereduce` [52].
- Atenuación adaptativa: Se ha realizado un código que implementa un filtro de reducción de ruido adaptativo para un archivo de audio en formato WAV. Para ello se han utilizado las librerías `Numpy` y `Soundfile` de Python. El proceso consta de varios pasos, que incluyen la lectura del archivo de audio, el cálculo de la máscara, el suavizado de la máscara, la aplicación de la máscara y la escritura del archivo de audio filtrado. La máscara se calcula dividiendo el audio en bloques y se utiliza una cantidad del 85% de la energía en la señal para evitar la saturación.
La energía media se utiliza como una medida de la señal de voz o de música y la señal de ruido. Luego, se aplica una máscara a cada bloque de audio y se suaviza para evitar artefactos. Por último, se escribe la señal de audio filtrada en un archivo WAV, normalizándola nuevamente en el rango `[-32768, 32767]`, que es el rango para el formato de muestreo de 16 bits PCM.
- Butterworth: Se ha implementado mediante un código que realiza el filtro Butterworth a un archivo de audio en formato WAV. El filtro es un filtro pasa banda, con un ancho de banda que va desde 300 Hz hasta 3400 Hz, lo que simula un teléfono analógico. El código utiliza la biblioteca de Python `Scipy` [48] para calcular el filtro y filtrar la señal de audio. El filtro se configura especificando los límites de frecuencia de la banda de paso, el orden del filtro y la tasa de muestreo del archivo de audio. La señal de audio se lee en un "buffer" y se filtra utilizando la función `sosfilt` de `Scipy`. Luego, la señal filtrada se escribe en un nuevo archivo WAV y se guarda.
- Wiener: Utilizando la librería `SoundPy` [90]. El código lee el archivo de audio en un "buffer" y se utiliza la biblioteca `soundfile` para cargar la señal de audio en la memoria. La señal de audio se filtra utilizando la función `soundpy.filtersignal` de la biblioteca `soundpy`. La función toma como entrada la señal de audio, la tasa de muestreo y el tipo de filtro a aplicar (en este caso, un filtro Wiener).
- Wiener plus: Se utiliza el mismo código que el utilizado en Wiener, pero con los parámetros de `Soundpy` `apply_postfilter` y `filter_scale` que son dos parámetros opcionales que se utilizan en la función `soundpy.filtersignal`, para realizar un procesamiento posterior del audio con mayor o menor intensidad en función de `filter_scale`.

- Noise-Reduce + RNN: Combinación de ambos filtros.
- RNN + Amplify: Filtro Rnoise y amplificación de señal obtenida.

3.4.2 Segmentación de audios

Una vez obtenidos los audios filtrados, podríamos pensar que ya se encuentran listos para su transcripción con Vosk o Whisper. Sin embargo, es aquí donde surge una de las principales diferencias entre los dos modelos de transcripción. Mientras Whisper funciona mejor con fragmentos de audio no demasiado cortos, a Vosk le sucede todo lo contrario. Vosk es capaz de procesar los audios de manera más eficaz si la duración de estos es de pocos segundos, conteniendo pocas palabras.

Por ello, se ha adoptado la solución de cortar los fragmentos de 2 minutos de las lanchas en fragmentos basados en silencios para su procesamiento posterior por Vosk. El funcionamiento es el explicado a continuación. La función toma el segmento de audio de 2 minutos y mediante la librería `utils` realiza la segmentación indicando los siguientes parámetros:

- `min_silence_len`: La duración mínima de tiempo que tiene que durar el silencio (en milisegundos) que se usará para realizar la división.
- `silence_thresh`: El umbral máximo de ruido (en dBFS) que se considerará como silencio. Cualquier cosa más silenciosa que este umbral será considerada como una sección de silencio.
- `keep_silence`: Tiempo de silencio (en milisegundos o True/False) que se dejará al principio y al final de los segmentos resultantes. Esto permite que el sonido no parezca cortado abruptamente. Si se especifica True, se mantendrá todo el silencio, si se especifica False, no se mantendrá ningún silencio.
- `seek_step`: Tamaño de segmento iterado en milisegundos. Se utiliza para controlar el tamaño de la porción de audio que se evalúa en cada iteración durante el proceso de detección de secciones silenciosas en el audio. Si el `seek_step` es grande, la detección será más rápida, pero puede no ser tan precisa.

Además, también existe una función para la aproximación de la duración del silencio cortado, con el objetivo de tener, una vez hecha la transcripción, el tiempo absoluto en el que se dijo cada palabra con respecto al comienzo de la grabación.

3.5 Proceso de transcripción

Para la transcripción de los archivos de audio se han realizado tres métodos. El primero ha sido mediante Vosk y Kaldi KWS junto a un vocabulario náutico previamente definido. En el segundo se ha utilizado Whisper y su modelo más preciso de transcripción. Por último, se ha diseñado un tercer método que combina los dos anteriores, mejorando de esta forma la precisión obtenida. A continuación, se detallarán los modelos en profundidad, detallando además el vocabulario establecido para Kaldi KWS.

3.5.1 Definición de vocabulario

Concluido el procesamiento inicial del audio, los motores de Whisper y de Vosk estarían preparados para la transcripción de los audios sin y con corte respectivamente, obteniendo los distintos resultados para el mismo audio.

A pesar de esto, la búsqueda de órdenes que se realiza en Vosk, y sobre la que se ha utilizado la herramienta de Kaldi KWS para búsqueda de órdenes, necesita un diccionario base que compare con el audio, encontrando así las similitudes. Vosk de por sí incorpora un vocabulario en su modelo general,

sin embargo, no está adaptado al entorno naval, siendo necesaria la definición de un vocabulario especializado que utilizar para las palabras OOV del sistema Kaldi KWS (sección 3.5.1).

Para ello se han establecido tres archivos de texto identificados como unigramas, bigramas y trigramas. El contenido de estos ficheros se encuentra en el Anexo V: Unigramas, bigramas y trigramas. Estos archivos permiten al sistema encontrar órdenes incluso a pesar de que no estén en el vocabulario de Vosk, gracias a la descomposición de las órdenes largas en segmentos menores de palabras y búsqueda por similitud acústica con el audio.

El código Python encargado de pasar los diccionarios a Vosk está configurado con cinco posibilidades de uso: No usar diccionarios específicos (diccionario general Vosk), usar unigramas, bigramas, trigramas, o los tres a la vez.

Es necesario aclarar que los diccionarios se han obtenido a partir de la Resolución A.918 elaborada por la Organización Marítima Internacional (IMO) [5], cuya portada está representada en la Figura 3-6, en la que define las frases normalizadas para las comunicaciones marítimas, con su respectiva traducción al español. Esta información se ha contrastado con su uso práctico en los buques militares, añadiendo o modificando ciertas órdenes en desuso.

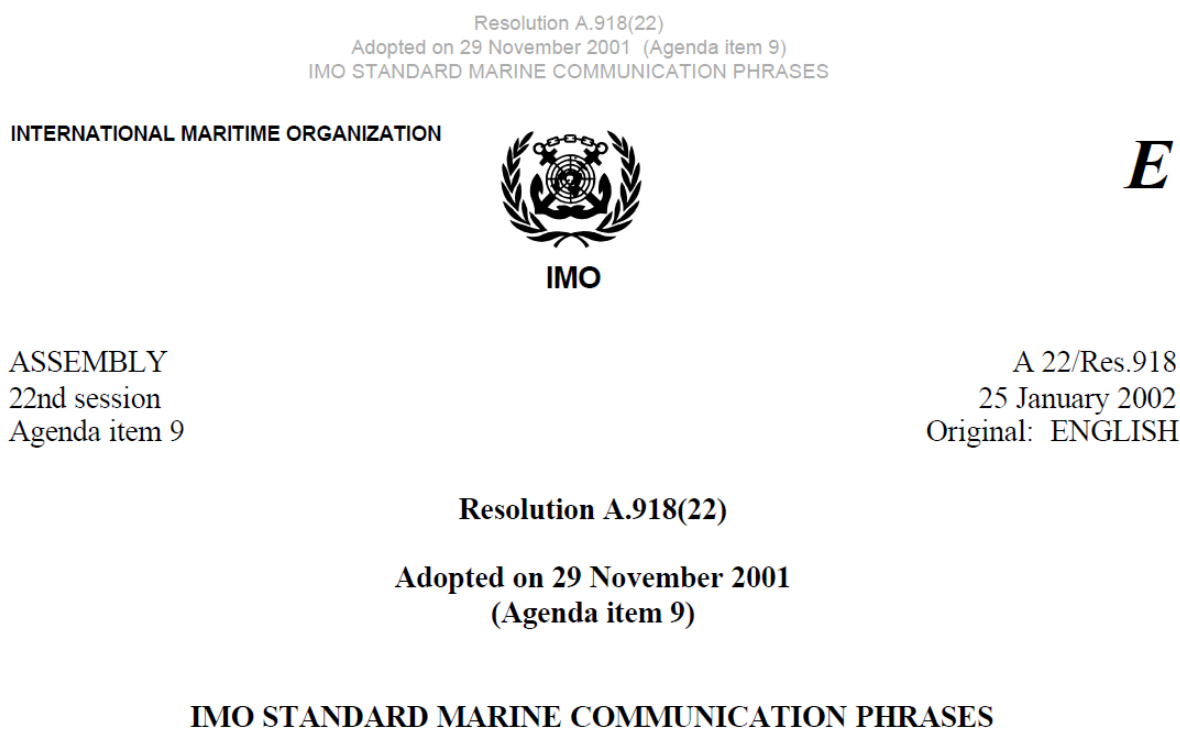


Figura 3-6: Documento de frases normalizadas para las comunicaciones marítimas. Fuente: [5]

3.5.2 Transcripción con Vosk y Kaldi KWS

Una vez finalizados los pasos anteriores, para realizar la transcripción mediante Vosk y su módulo Kaldi KWS, se le pasan los fragmentos cortados llamados *chunks* de cada audio de 2 minutos. Para conseguirlo, se utiliza la siguiente línea de código para cada fragmento:

```
rec = KaldiRecognizer(model, wf.getframerate(), json)
```

Figura 3-7: Línea de código Kaldirecognizer. Fuente: propia

siendo el argumento *model* el modelo *small* de Vosk en español de la Figura 3-8 utilizado para la transcripción; y el argumento *json* el documento de texto con el vocabulario de ámbito naval para Kaldi KWS, es decir, con los unigramas, bigramas, trigramas, o una combinación de estos. Este vocabulario servirá para realizar la búsqueda de palabras (*Keyword Assignment*), que se encargará de encontrar y marcar las palabras coincidentes entre vocabulario y audio, teniendo en cuenta que las palabras fuera de este vocabulario no son transcritas.

Spanish			
vosk-model-small-es-0.42	39M	16.02 (cv test) 16.72 (mtedx test) 11.21 (mls)	Lightweight wideband model for Android and RPi
vosk-model-es-0.42	1.4G	7.50 (cv test) 10.05 (mtedx test) 5.84 (mls)	Big model for Spanish

Figura 3-8: Modelos disponibles de Vosk en español. Fuente: [91]

El resultado del proceso de transcripción sería el de un archivo de texto para los *chunks* como los mostrados a continuación, donde se representan tres columnas que indican el vocabulario empleado, el resultado para cada vocabulario y los tiempos de inicio de cada palabra. En el caso de la Figura 3-10 se muestra el resultado para un archivo completo de 2 minutos con el vocabulario *all* seleccionado. Se representan los tiempos de inicio y fin de cada palabra y *chunk*, así como la transcripción obtenida para cada uno de ellos dentro del archivo de 2 minutos.

```

none      faldilla lo haga todo  [ 0.78 1.32 1.41 1.
words     toldilla              [ 0.75 ]          [ 1.32 ]
bigrams   toldilla larga todo   [ 0.75 1.32 1.62 ]
trigrams  toldilla larga todo   [ 0.75 1.32 1.62 ]
all       toldilla larga todo   [ 0.75 1.32 1.62 ]

```

Figura 3-9: Transcripción de un chunk en función del vocabulario establecido. Fuente: propia

DATE	-TIME	START AUDIO	END AUDIO	START 1WORD	END LWORD	CHUNK	SENTENCE		
2021-07-22-09:46:01		0.0	1.8329251700680271	0.12	0.96	chunk0.txt	babor estribor de guardia		
2021-07-22-09:46:01		2.5329705215419502		3.1429478458049886		2.058532	2.330532	chunk1.txt	larga través
2021-07-22-09:46:01		3.7519274376417235		5.031927437641723		3.600751	3.990751	chunk2.txt	larga spring
2021-07-22-09:46:01		5.381950113378685		9.765986394557823		6.020381	6.469381	chunk3.txt	larga el largo
2021-07-22-09:46:01		5.381950113378685		9.765986394557823		8.240381	9.080381	chunk3.txt	castillo larga todo
2021-07-22-09:46:01		10.216961451247165		12.130975056689342		10.750216	11.620216	chunk4.txt	toldilla larga todo
2021-07-22-09:46:01		12.58095238095238		13.676916099773242		12.30058	12.72058	chunk5.txt	cambio de bandera
2021-07-22-09:46:01		14.08	16.231927437641723	15.13908		16.06908		chunk6.txt	atrás poca las dos
2021-07-22-09:46:01		16.76997732426304		22.508934240362812		20.740769	21.490769	chunk7.txt	atrás poca babor
2021-07-22-09:46:01		23.303945578231293		24.336961451247166		23.004303	23.750303	chunk8.txt	atrás poca estribor
2021-07-22-09:46:01		25.33895691609977		26.336961451247166		25.000338	25.840338	chunk9.txt	avante poca las dos
2021-07-22-09:46:01		27.280997732426304		28.42893424036281		27.00028	27.81028	chunk10.txt	avante poca estribor
2021-07-22-09:46:01		29.06793650793651		29.855963718820863		29.000067	29.660067	chunk11.txt	avante poca babor
2021-07-22-09:46:01		36.6249433106576		36.752925170068025		none	none	chunk12.txt	
2021-07-22-09:46:01		39.09795918367347		44.86693877551021		none	none	chunk13.txt	
2021-07-22-09:46:01		39.09795918367347		44.86693877551021		none	none	chunk13.txt	
2021-07-22-09:46:01		46.43791383219955		49.09396825396826		none	none	chunk14.txt	
2021-07-22-09:46:01		49.49297052154195		50.21197278911565		none	none	chunk15.txt	
2021-07-22-09:46:01		50.99392290249433		52.593922902494334		none	none	chunk16.txt	
2021-07-22-09:46:01		53.02793650793651		53.15891156462585		none	none	chunk17.txt	
2021-07-22-09:46:01		53.651972789115646		55.0249433106576		none	none	chunk18.txt	
2021-07-22-09:46:01		55.49297052154195		55.701950113378686		none	none	chunk19.txt	
2021-07-22-09:46:01		56.09995464852608		56.130975056689344		none	none	chunk20.txt	
2021-07-22-09:46:01		56.52	56.53197278911565	none	none	chunk21.txt			
2021-07-22-09:46:01		57.30195011337869		59.903945578231294		none	none	chunk22.txt	

Figura 3-10: Transcripción para los chunks de un archivo y vocabulario *all*. Fuente: propia

3.5.3 Transcripción con Whisper

Para realizar la transcripción mediante Whisper, se ha aplicado el siguiente código por cada audio de dos minutos grabado:

```
model= pywhisper.load_model("large", device="cpu")  
  
out = model.transcribe(path_to_saved_file)
```

Figura 3-11: Línea de código Whisper. Fuente: propia

En primer lugar, se carga el modelo de Whisper es cargado mediante la librería `Pywhisper`, mencionada en la sección 2.6.1, y se utiliza la función `.transcribe` indicando el archivo de audio correspondiente para obtener el texto en una variable que se puede pasar a un archivo de texto.

En este caso no se utilizan los audios segmentados ya que, tras distintas pruebas, se ha podido observar que la calidad de la transcripción es mayor cuanto más largo es el audio, ya que permite conocer su contexto.

El resultado de la transcripción da lugar a un *json* con los distintos elementos: el texto total, las frases contenidas, el instante temporal de inicio y fin de frase, y los tokens que ha empleado Whisper. En la Figura 3-12 se pueden observar estos datos obtenidos a partir de una grabación realizada en una lancha de instrucción.

```
{'text': ' Avante poco vapor Para todo Atrás poco vapor Para todo Listo comandante  
Se libera en prueba de máquinas Con su permiso comienzo de  
maniobra de desastres aquí Vale Castillo y Tordilla aligerar Eva Por ordenes generales  
Ligerar Urko Urko Espera que está ahí Vale, aligerados  
Vale, pues Hey Tord, dame una escuadra O un cartabos Ah, genial Rápido, eh Con su  
permiso comandante Castillo y Tordilla Ligerados Largo Tordilla  
y me preparo para hacer cabeza Venga, venga, que va Venga, venga, que va Venga,  
venga, que va Venga, venga, sigue, sigue Castillo y Tordilla aligerados  
Sigo la secuencia Enterada Eva ',  
'segments': [{ 'id': 0, 'seek': 0, 'start': 0.0, 'end': 2.0, 'text': ' Avante poco vapor  
'tokens': [50364,  
11667, 2879, 10639, 20358, 50464, 50464, 11107, 5149, 50564, 50564,  
1711, 17168, 10639, 20358, 50664, 50664, 11107, 5149, 50764, 50764, 441,  
9334, 395, 474,  
2879, 50864, 50864, 1100, 6774, 64, 465, 48241, 368, 39701, 17094, 50964,  
50964, 2656, 459, 4784, 19227, 395, 1053, 4765, 368, 587, 72, 24393, 368, 730, 525,  
495, 6661, 51064, 51064, 26415, 51164, 51164, 11019, 15831, 288, 314, 765,  
5291, 51264, 51264, 441, 4810, 289, 29377, 51364, 51364, 5269, 28615, 279, 2674, 279,  
51464, 51464, 11019, 15831, 288, 314, 765, 5291, 441, 4810, 289, 51564],  
'temperature': 0.0, 'avg_logprob': -0.34115958999801466, 'compression_ratio': 1.5443037,  
'no_speech_prob': 0.5503456592559814}, { 'id': 1, 'seek': 0, 'start': 2.0, 'end': 4.0}
```

Figura 3-12: Transcripción Whisper. Fuente: propia

3.5.4 Transcripción con Whisper mejorado

Tras comprobar el rendimiento de los dos modelos de transcripción anteriores por separado (que se verán en detalle en la sección 4.1), se ha optado por realizar una mejora en el sistema de transcripción. Esta mejora lleva consigo la utilización dual de las salidas de Vosk y Whisper. En este punto hay que destacar que Whisper devuelve una transcripción total del audio, mientras que Vosk únicamente detecta

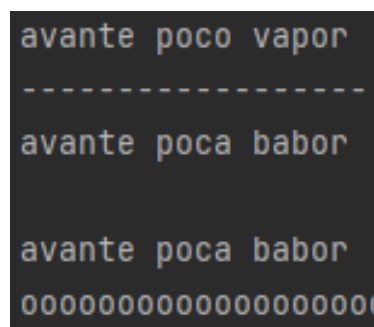
aquellas palabras/bigramas o trigramas proporcionadas. Por tanto, se opta por mejorar la salida de Whisper a partir de las frases reconocidas por Vosk. De este modo, se solucionan posibles errores en palabras náuticas.

Para realizar este proceso de mejora, es necesario aplicar un procedimiento de comparación entre palabras de Whisper y Vosk utilizando la distancia *Levenshtein*. Esta distancia se define como el número mínimo de operaciones de edición (insertar, borrar o reemplazar un carácter) necesarias para transformar una cadena en otra.

Para llegar a las frases mejoradas, se hace uso de los tiempos de frase proporcionados por Whisper y Vosk en sus respectivas transcripciones, y se realizan los siguientes pasos generales:

- Se leen los tiempos de inicio de frase proporcionados por Whisper en un array.
- Para cada tiempo de frase del array, se guarda la transcripción de Whisper en otro array.
- Se comparan los tiempos de inicio de frase proporcionados por Vosk con los de Whisper, y se asocian. Para ello, se busca el tiempo inmediatamente inferior o igual al de Vosk en Whisper, y se guarda en un array. Se realiza el proceso con todas las frases de Vosk, obteniendo un array con los tiempos de Whisper en los que deberían encontrarse las mismas frases que Vosk.
- Para cada tiempo del array del paso anterior, se comparan las frases de Vosk y Whisper palabra a palabra. Si una palabra de Vosk tiene distancia *Levenshtein* menor a 0.5 con otra de Whisper, se supone que es la misma palabra, y se sustituye. Por lo tanto, suponemos que la palabra correcta es la de Vosk, al utilizar un vocabulario. También se estudia si una palabra tiene distancia *Levenshtein* mayor a 0.8 y no está en Whisper, y se incluye, para el caso de que Vosk consiguiese capturar algo que Whisper no.
- Por último, se compara la transcripción mejorada completa con un archivo de trigramas, iterando sobre grupos de tres palabras de la transcripción y sustituyéndolas por las palabras de los trigramas si su distancia *Levenshtein* con ellos es menor de 0.2.

La Figura 3-13 muestra un ejemplo del proceso descrito en Python para dos frases del mismo instante de tiempo de Whisper y Vosk obtenidas en el paso 3. Se indica la transcripción por Whisper en la parte superior, a continuación, la misma transcripción realizada por Vosk, y por último la salida de Whisper mejorada en la parte inferior, combinando las dos frases anteriores con los pasos 4 y 5 explicados anteriormente.



```
avante poco vapor
-----
avante poca babor
avante poca babor
oooooooooooooooooooo
```

Figura 3-13: Transcripción Whisper mejorado 1. Fuente: propia

La Figura 3-14 muestra el mismo proceso, pero en el caso en el que Vosk aporta información adicional a la transcripción de Whisper. Es decir, en este caso en el paso 4 la distancia *Levenshtein* calculada es mayor a 0.8 para la palabra *para*, por lo que se incluye en la frase mejorada.

Estos ejemplos han sido obtenidos a través del código que se puede encontrar en el Anexo VII: Código de Whisper mejorado y evaluaciones 1 y 4.

```
cinco de caña a estribor a quedar rumbo 020
-----
[unk1] para
cinco de caña a estribor a quedar rumbo 020 [unk1] para
oooooooooooooooooooooooooooooooooooooooo
```

Figura 3-14: Transcripción Whisper mejorado 2. Fuente: propia

3.6 Proceso de identificación de órdenes en las transcripciones

Como ya se ha comentado, hasta el momento se ha realizado la identificación de terminología náutica mediante Kaldi KWS a la hora de hacer la transcripción desde el audio. Esto es lo que se conoce como *Keyword Assignment*. Sin embargo, a la hora de trabajar con una transcripción completa es necesario buscar una alternativa. Por este motivo, se ha elaborado un sistema supervisado de *Keyword Extraction*, previamente entrenado con los audios y transcripciones creados a partir de las grabaciones capturadas en las lanchas de instrucción.

Que el modelo sea supervisado implica que se utilizan conjuntos de datos etiquetados para entrenarlo. Es decir, cada ejemplo de entrenamiento tiene una etiqueta o respuesta conocida asociada a él. El objetivo del modelo es aprender a predecir las etiquetas correctas para nuevos ejemplos no vistos. En lo referente al algoritmo seleccionado, este lo ha sido basándose en proyectos y publicaciones dedicados al ABSA (*Aspect Based Sentiment Analysis*) o análisis de sentimientos en textos a partir de palabras clave [84]; y al etiquetado de secuencias [92].

En los posteriores apartados se procederá a explicar los algoritmos empleados en el sistema, y el mecanismo de entrenamiento.

3.6.1 Algoritmos empleados

3.6.1.1 Algoritmo CRF

El modelo CRF (*Conditional Random Fields*) es una clase de método de modelado estadístico que se aplica frecuentemente en el reconocimiento de patrones y el aprendizaje automático, y se utilizan para la predicción estructurada. Mientras que un clasificador predice una etiqueta para una sola muestra sin tener en cuenta las muestras "vecinas", un CRF puede tomar en cuenta el contexto del texto. Como ejemplo, podemos comparar el esquema de los CRFs de cadenas lineales simples con el del modelo HMM, sustituyendo las líneas unidireccionales que conectan distintas predicciones por nodos.

En la Figura 3-15 se observa como una red HMM de sentido único pasa a ser una red CRF al sustituir las flechas por nodos de doble dirección. En un HMM, las flechas representan la transición de un estado oculto a otro, mientras que, en un CRF, los nodos representan las etiquetas de las observaciones y las etiquetas de los estados. La transformación de una HMM a un CRF requiere la creación de nodos para representar tanto las etiquetas de observación como las etiquetas de estado, y la adición de conexiones entre estos nodos. Esta transformación permite que las observaciones influyan en las predicciones de las etiquetas de estado adyacentes y permite una mayor flexibilidad en la modelización de las dependencias entre las etiquetas de estado y las observaciones.

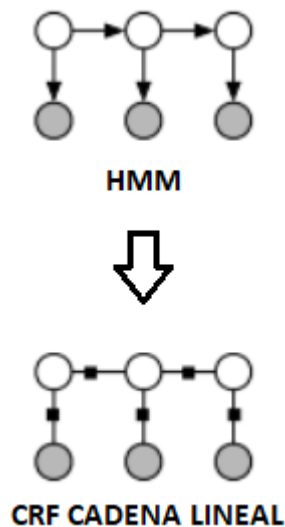


Figura 3-15: Comparación CRF de cadena lineal y HMM. Fuente: propia

3.6.1.2 Algoritmo BiLSTM-CRF

El sistema utilizado para el mapeado de características de los audios e identificación de palabras en base a un entrenamiento estructurado ha sido un modelo de redes neuronales BiLSTM en combinación con una capa de análisis estadístico CRF.

Por una parte, la principal ventaja del modelo de red neuronal usado BiLSTM respecto al LSTM regular es que permite que la red neuronal tenga en cuenta tanto la información pasada como futura de la secuencia de texto de entrada. Esto permite mejorar la precisión del modelo en tareas que dependen de la comprensión de secuencias de datos.

El modelo LSTM bidireccional, o BiLSTM, es un modelo de procesamiento de secuencias que consiste en dos LSTMs: uno que toma la entrada en una dirección hacia adelante y el otro en una dirección hacia atrás. Las BiLSTMs aumentan efectivamente la cantidad de información disponible para la red, mejorando el contexto disponible para el algoritmo (por ejemplo, saber qué palabras siguen inmediatamente y preceden a una palabra en una oración).

Combinar una red BiLSTM y una red CRF para formar un modelo BiLSTM-CRF permite usar eficientemente características de entrada pasadas a través de la capa BiLSTM y la información de etiquetado a nivel de frase a través de la capa CRF. La capa CRF se representa por líneas que conectan capas de salida consecutivas [92].

A modo de resumen, el sistema utiliza el modelo BiLSTM para extraer gracias al *Deep Learning* características de los textos, analizando las normas comunes entre ellos y simplificando su clasificación posterior. La capa CRF se implementa sobre el sistema para mejorar la eficiencia de este con la capacidad de predicción en base al contexto y al entrenamiento realizado con anterioridad.

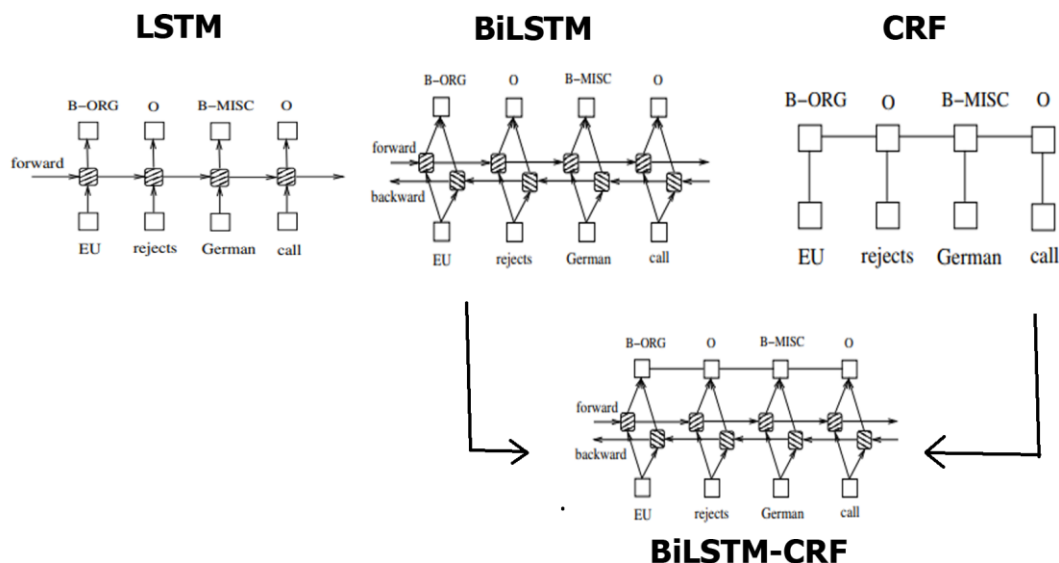


Figura 3-16: Ejemplo modelo BiLSTM-CRF. Fuente: [92]

3.6.2 Extracción de características

La utilización de los algoritmos descritos mediante distintas librerías de Python ha dado lugar a un modelo secuencial no dirigido capaz de modelar la probabilidad condicional $p(Y|X)$ sobre la secuencia de texto oculta Y a partir de la secuencia de observación X .

El objetivo del sistema es reconocer si lo marcado es una orden o no. Para ello se utilizan las etiquetas IOB [22] y se tienen en cuenta las siguientes características:

- **Forma y formas de sus contextos:** Se tiene en cuenta como características de una palabra dada, una ventana alrededor suya de - 2, es decir, dos palabras anteriores. Podemos decir que es un pequeño contexto para la palabra a analizar.
- **Etiqueta léxica:** Se tienen en cuenta las etiquetas léxicas de cada palabra de la ventana, es decir, si se tratan de sustantivos, adjetivos, conjunciones, etc. Para ello se ha utilizado la librería de Python `Spacy` [93], una librería de Python para PLN que permite usar un modelo general para español. Además de para el español, ofrece modelos para 24 idiomas.
- **Lema:** El lema es por convenio la palabra se acepta como representante de todas las formas flexionadas de una misma palabra. Por ejemplo, *cayendo* es una forma cuyo lema es *caer*, pero también lo es de *cae* o *caemos*. Si la forma es un número, su lema se indica con NUM.
- **Análisis sintáctico:** Se analizan sintácticamente las oraciones, dividiéndolas en sujeto, predicado, etc.

El resultado de esta fase es un vector de características, en el que se ha utilizado `Spacy` para el español para obtener los valores de las características descritas, las cuales serán utilizadas para el entrenamiento del sistema de identificación de órdenes, y posteriormente para su detección.

Como veremos en el siguiente apartado, tras el entrenamiento, el sistema debe ser capaz de identificar las órdenes en la transcripción. Además, las repeticiones de las órdenes del oficial en el puente de mando por parte del resto de dotación también se localizarían, no siendo marcadas.

Gracias al uso del sistema entrenado se consigue un alto nivel de precisión, descartando frases como órdenes por el contexto, a pesar de que Kaldi KWS las transcribió al encontrarse dentro de su vocabulario

de órdenes y fueron etiquetadas. Por ejemplo, si el comandante dijese “*Recomiendo caer a babor*”, el sistema entrenado sería capaz de discernir la recomendación de una orden, no marcándola.

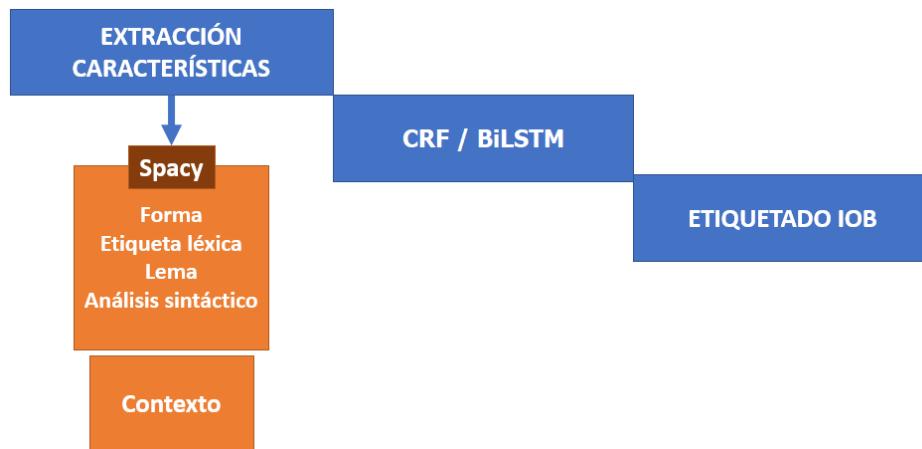


Figura 3-17: Esquema identificador de órdenes. Fuente: propia

3.6.3 Entrenamiento del sistema

Para llegar al funcionamiento automático descrito, se ha tenido que realizar un entrenamiento previo para la obtención del modelo del sistema. Para ello, se ha separado el texto en tokens convertidos a minúsculas. A continuación, cada token se ha marcado con una etiqueta que sigue el esquema IOB (B: *Begin*, I: *Inside*, O: *Outside*), que es el más utilizado actualmente en problemas similares de etiquetado de secuencias [22]. Concretamente, las etiquetas B e I indican el comienzo y longitud de la secuencia y la etiqueta O marcan secuencias que no corresponden a la búsqueda, es decir, que no son órdenes. Un ejemplo puede encontrarse en la Figura 3-18.

castillo	larga	todo	todo	largado	castillo...	avante	poca	babor	babor	en	avante	poca
B	I	I	O	O	O	B	I	I	O	O	O	O

Figura 3-18: Ejemplo de marcado IOB. Fuente: propia

Para realizar el entrenamiento, se han utilizado grabaciones obtenidas en las lanchas tras la obtención de sus transcripciones mediante Whisper sobre las que se realizó un proceso de revisión manual. La transcripción resultante contiene por cada línea una frase. Sobre ella, se ha realizado manualmente un proceso de etiquetado de las órdenes mediante marcadores “<” y “>”, y de las repeticiones de estas por el timonel con “{“y “}”.

Es importante aclarar que las transcripciones de Whisper han sido empleadas como ayuda al entrenamiento y evaluación del sistema de detección debido a que, a pesar de que Whisper no incorpora un sistema como Kaldi KWS para la búsqueda de palabras, es capaz de captar toda la conversación en los audios sin importar la intensidad del sonido recibido. Esto hace posible crear las transcripciones de los archivos por frase de los audios de forma más rápida que si fuesen realizadas por una persona. Sin embargo, los textos requieren revisión para la corrección de errores en palabras específicas.

El resultado del archivo final para el entrenamiento del sistema es como el que se puede ver en la Figura 3-19 a continuación. Como vemos, las órdenes y respuestas quedan marcadas de la forma ya mencionada:

```
Eee <diez grados de caña a babor a quedar a rumbo tres cuatro cero>
{rumbo Tres cuatro cero}
si
tengo torpedos
A tus órdenes mi oficial según carta se recomienda poner un rumbo tres
Tres uno cinco
Si si
Tienes que hacer una guiñada y ya incorporarte
Eee quédate
{Rumbo Tres cuatro cero}
<Diez grados de caña a babor a quedar a rumbo tres uno cinco>
{Quedar a rumbo tres uno cinco}
Siempre mirar eh si hace falta salir
todos de barrutia control
Chereguini
Rull
Segunda se recomienda poner rumbo del tramo
<Rumbo del tramo>
Chereguini rull de barrutia les recibo fuerte y claro
<Quince grados de caña a babor a quedar a rumbo dos cuatro cero>
Según radar nos vamos a quedar a estribor de
Eee <Quince grados de caña a babor a quedar a rumbo dos cuatro cinco>
Dos copiado
{rumbo dos cuatro cinco}
```

Figura 3-19: Ejemplo de texto de entrenamiento del sistema detector de órdenes. Fuente: propia

Los archivos de entrenamiento han sido utilizados con dos algoritmos con el objetivo de tener dos modelos entrenados que comparar y elegir el más eficiente.

Los modelos han sido los siguientes:

- Entrenamiento realizado con `Scikit-learn` [94], una biblioteca Python de código abierto para el aprendizaje automático y CRF, al tiempo que se usa un método de optimización `'lbfgs'` [95].
- Entrenamiento realizado con un modelo BiLSTM-CRF similar al descrito en el apartado 3.6.1, mediante la librería `Torch` [96], y basado en el modelo descrito en [92]

Tras varias pruebas, se ha determinado que el modelo con el que mejores resultados se obtiene es el primero, que a pesar de ser más simple y no emplear redes neuronales, presenta resultados más robustos con menos errores para el número de datos de entrenamiento empleados, siendo el aplicado finalmente.

Sin embargo, no se descarta el uso del modelo BiLSTM-CRF en entrenamientos futuros con más datos, puesto que teóricamente es capaz de predecir patrones de forma más precisa que el CRF simple gracias al uso de redes neuronales.

Por otra parte, se han llevado a cabo dos tipos de entrenamiento para cada uno de los modelos descritos:

- Entrenamiento por transcripciones completas de dos minutos.
- Entrenamiento por frases dentro de las transcripciones de dos minutos.

La diferencia principal entre entrenar un modelo por transcripciones de texto con frases o entrenarlo por cada una de las frases es la granularidad de los datos de entrenamiento.

Entrenar un modelo por transcripciones de texto completas implica que se tiene un conjunto de frases dentro de cada transcripción, y que el modelo es entrenado con toda la transcripción como una única unidad. En este caso, el modelo está aprendiendo a etiquetar cada una de las frases dentro de la

transcripción, tomando en consideración la información contextual de todo el archivo, es decir, la secuenciación seguida.

Por otro lado, entrenar un modelo CRF por frases implica que cada frase es una unidad separada de entrenamiento. En este caso, el modelo está aprendiendo a etiquetar cada frase de forma independiente, sin considerar la información contextual de otras frases.

El esquema representativo de los pasos de entrenamiento es el que aparece en la siguiente figura:

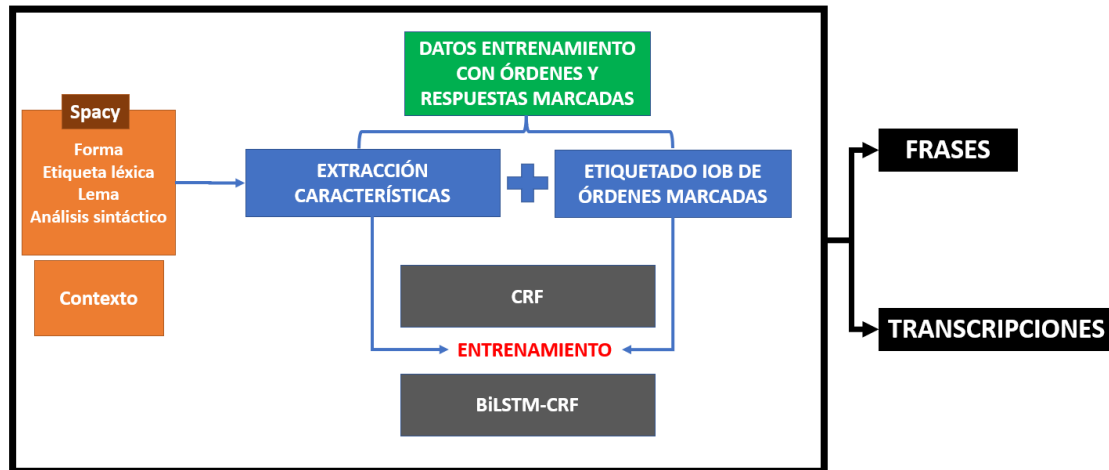


Figura 3-20: Esquema sistema de entrenamiento. Fuente: propia

Para la obtención de la evaluación del entrenamiento, el sistema utiliza la comparación del 20% de los datos y sus predicciones, mientras que el 80% restante son usados como datos de entrenamiento.

El sistema entrenado da como resultado un archivo de texto con las predicciones etiquetadas automáticamente con el formato IOB, que se explicará en la sección 3.6.3. El resultado para una frase está representado en la Figura 3-21.

caña	tag prediction: B
a	tag prediction: I
la	tag prediction: I
vía	tag prediction: I
y	tag prediction: O
tienes	tag prediction: O
que	tag prediction: O
tomar	tag prediction: O
una	tag prediction: O

Figura 3-21: Resultado del sistema de obtención de órdenes. Fuente: propia

3.7 Mecanismos de evaluación

A continuación, se tratarán los mecanismos de evaluación tanto de las transcripciones creadas como del sistema de detección de órdenes.

3.7.1 Evaluación de las transcripciones mediante WER

Una vez transcrito nuestro texto y alineado, debemos comprobar su rendimiento y nivel de acierto.

La técnica más conocida es el cálculo del *word error ratio* (WER) [97], que compara la salida y la transcripción real palabra por palabra (o carácter por carácter) para determinar el número de diferencias entre ellas.

Las posibles diferencias podrían ser:

- Una palabra que está presente en la transcripción pero que falta en la salida (una falta),
- Una palabra que no está en la transcripción, pero ha sido agregada a la salida (una inserción),
- Una palabra que ha sido alterada entre la predicción y la transcripción (una sustitución).

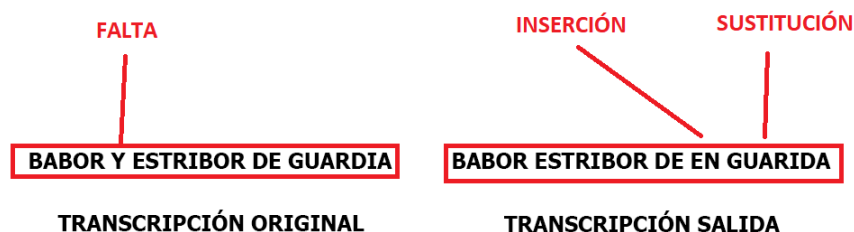


Figura 3-22: Cuenta de inserciones, faltas y sustituciones en una transcripción. Fuente: propia

La fórmula de la métrica representada en la (Ec 3) es bastante sencilla. Es el porcentaje de diferencias relativo al número total de palabras.

$$WER = \frac{Sustituciones + Faltas + Inserciones}{Total\ palabras} \quad (Ec\ 3)$$

Existen otros tipos de fórmulas para obtener ratios de evaluación similares al WER, como el CER, basado en caracteres en lugar de palabras. El empleo de uno u otro dependerá del tipo de sistema a evaluar y el resultado esperado.

Además, dependiendo de si el sistema ha utilizado unigramas, bigramas, trigramas o ninguno, se puede observar la variación del WER y rendimiento del sistema, aunque normalmente se usarán todos a la vez para la obtención de mejores resultados.

La librería empleada para el cálculo WER ha sido *Jiwer*, obteniendo las palabras eliminadas, añadidas o modificadas con respecto a la transcripción real, que son posteriormente divididas entre el número de palabras totales para la obtención del índice de error.

Se han realizado cuatro tipos evaluaciones mediante WER:

- 1- La primera evalúa la transcripción completa realizada por Whisper con la transcripción real, con el objetivo de estimar la eficiencia del modelo Whisper en el entorno de grabación.
- 2- La segunda evalúa únicamente las frases con órdenes de la transcripción entre Whisper y la real, para comprobar la calidad en la transcripción de estas obviando el resto audio,
- 3- La tercera evalúa igualmente las frases con órdenes de la transcripción real y la de Vosk, comprobando cuántas ordenes ha encontrado el sistema y cómo es la calidad de estas.
- 4- Por último, se ha realizado un sistema que compara las transcripciones de Whisper mejorado y las reales, facilitando la comparación con los otros modelos de transcripción para determinar si realmente los mejora.

También cabe destacar que la transcripción real se ha obtenido a partir de la transcripción de los archivos con Whisper, siendo revisada y corregida manualmente.

El código generador de las evaluaciones puede encontrarse en el Anexo VII: Código de Whisper mejorado y evaluaciones 1 y 4. y en el Anexo VIII: Código de evaluación 3 para Vosk.

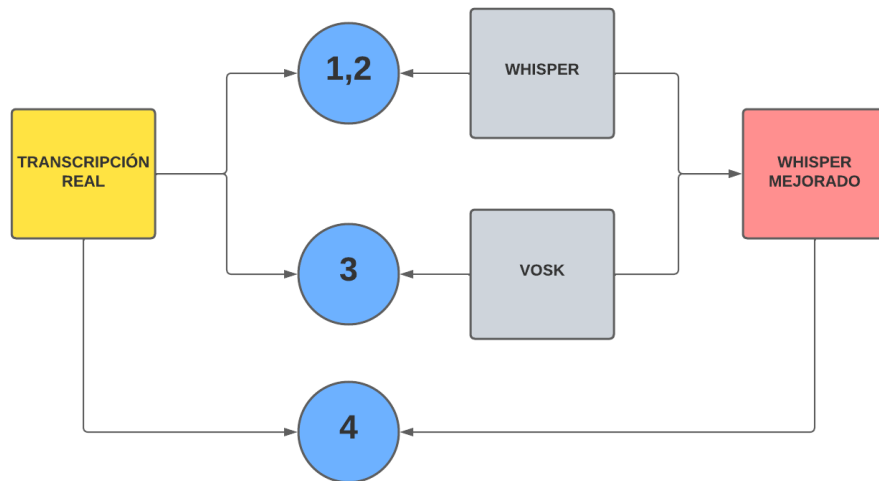


Figura 3-23: Evaluaciones del sistema mediante WER. Fuente: propia

3.7.2 Evaluación de la detección de órdenes

En cuanto a las medidas de evaluación del sistema de detección de órdenes, se ha utilizado principalmente la precisión, la cobertura y la medida *F1-score*.

En este sentido, la precisión mide la proporción de predicciones positivas correctas realizadas por el modelo en relación con el total de predicciones positivas realizadas. La precisión se enfoca en la exactitud de las predicciones positivas, sin tener en cuenta si el modelo logró identificar correctamente todos los casos positivos en el conjunto de datos. Por otro lado, la cobertura (*recall*) mide la proporción de casos positivos que el modelo ha identificado correctamente con respecto a todas las que son positivas.

Por otro lado, el *F1-score* es una medida de evaluación combinada que tiene en cuenta tanto la precisión como la cobertura. El *F1-score* se enfoca en encontrar un equilibrio entre la precisión y la cobertura, lo que significa que tiene en cuenta tanto la capacidad del modelo para identificar correctamente casos positivos como para evitar falsos positivos.

3.8 Simulación

Terminada la explicación completa del sistema, se procederá a explicar cómo se ha realizado la simulación de los resultados del sistema para, de una forma visual, observar las ordenes obtenidas en una situación real.

El software utilizado para ello ha sido SIMDIS, descrito en la sección 2.8.1. Los parámetros que ha tenido en cuenta y que se han tenido que aportar han sido:

- Posiciones GPS de la lancha obtenidas durante las salidas.
- Datos de las órdenes.
- Tiempo de cada orden respecto al inicio de la grabación.
- Fecha de grabación y hora de comienzo de la grabación.

Estos datos se han recopilado en un archivo de simulación como el de la Figura 3-24, que es convertido a formato “.asi” junto con las posiciones GPS y sus tiempos para ser procesado en SIMDIS. El código generador del archivo puede encontrarse en el Anexo IX: Código de preparación para SIMDIS.

```

DATE      TIME START      SENTENCE
2023-02-13-11:02:01      0.0      avante poca babor
2023-02-13-11:02:01      120.78   para avante babor
2023-02-13-11:02:01      128.78   para [unk1]
2023-02-13-11:02:01      419.26000000000005      avante ochocientas diez grados de caña a estribor a que
cero
2023-02-13-11:02:01      497.12   avante ocho nudos para [unk3]
2023-02-13-11:02:01      593.9    cinco grados de caña estribor a quedar rumbo dos nueve dos [unk3]
2023-02-13-11:02:01      669.9    diez grados de caña estribor a quedar rumbo uno diez cero [unk1]
2023-02-13-11:02:01      712.68   cinco de caña a babor a quedar rumbo cero cero cinco avante diez nudos
2023-02-13-11:02:01      733.68   cinco gradso de caña a estribor a quedar rumbo cero dos cero [unk1]
2023-02-13-11:02:01      885.4599999999999      diez grados de caña a babor a rumbo cero ocho cero
2023-02-13-11:02:01      975.2399999999999      a quedar rumbo cero tres cero [unk1]
2023-02-13-11:02:01      1011.2399999999999      cinco grados de caña estribor a diez
2023-02-13-11:02:01      1013.2399999999999      a diez grados de caña a estribor a quedar rumbo cero cu
2023-02-13-11:02:01      1031.2399999999998      cinco grados de caña diez grados de caña a quedar rumbo
2023-02-13-11:02:01      1041.2399999999998      caña a la via para todo
2023-02-13-11:02:01      1045.2399999999998      avante poca estribor
2023-02-13-11:02:01      1054.2399999999998      atrás poca estribor
2023-02-13-11:02:01      1058.2399999999998      atras toda atras toda estribor
2023-02-13-11:02:01      1066.2399999999998      todo la caña a babor para todo [unk1]
2023-02-13-11:02:01      1075.02   dos y medio avante

```

Figura 3-24: Archivo generador de simulación. Fuente: propia

Además, en SIMDIS se utilizan cartas náuticas electrónicas online del *Web Map Service* (WMS), proporcionadas por el Instituto Hidrográfico de la Marina (IHM) en su página web [98]. Las cartas empleadas han sido descritas en la Tabla 1.

NOMBRE	PROPÓSITO	ESCALA
cartaENCp5	5	1:4000 – 1:22000
cartaENCp4	4	1:22000 – 1:90000
cartaENCp3	3	1:90000 – 1:350000
cartaENCp2	2	1:350000 – 1:500000

Tabla 1: Cartas electrónicas empleadas en la simulación. Fuente: propia

El WMS es un estándar del *Open Geospatial Consortium* (OGC) [99] que se utiliza para compartir mapas en la web. Un WMS es un servicio web que permite a los usuarios solicitar mapas en línea y visualizarlos en un navegador web.

Los mapas que se proporcionan a través de este son imágenes georreferenciadas, que se generan dinámicamente a partir de capas de información geográfica almacenadas en un servidor. El servidor WMS toma estas capas de información y las combina en un mapa final que se puede mostrar en un navegador web.

En la siguiente imagen se puede ver que queda cubierta toda la zona de tránsito de las lanchas de instrucción e incluso el territorio perteneciente a la zona económica exclusiva del territorio español:

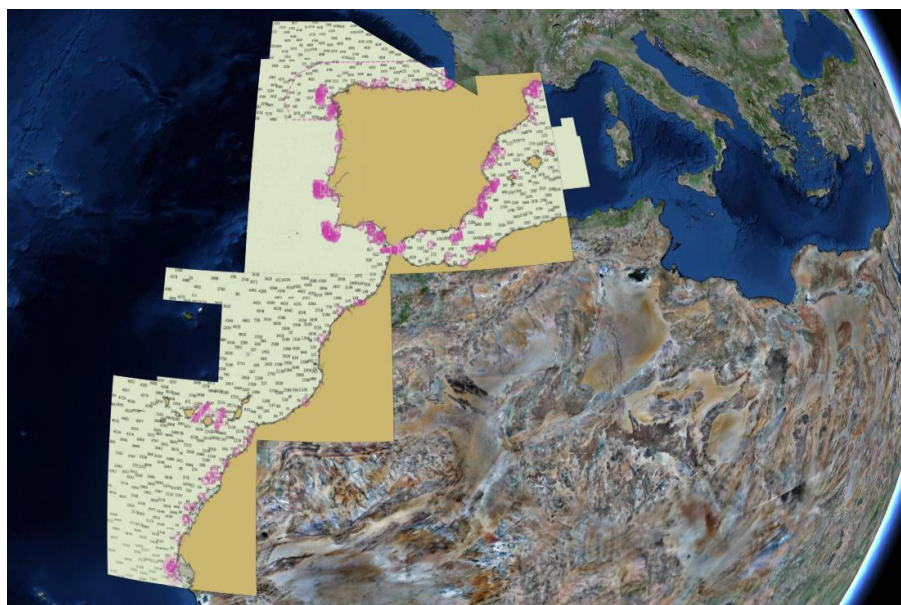


Figura 3-25: Cobertura de las cartas electrónicas empleadas. Fuente: propia

Por último, en la Figura 3-26 se muestra un ejemplo de simulación para una salida de ejemplo.



Figura 3-26: Ejemplo simulación SIMDIS. Fuente: propia

4 RESULTADOS DE LAS PRUEBAS

En los siguientes apartados se detallarán y explicarán los resultados obtenidos en las distintas pruebas realizadas con el sistema, como pueden ser la validación y medición de la precisión de transcripción de los modelos de Vosk y Whisper, los resultados de entrenamiento del detector de órdenes y los inconvenientes encontrados durante todo el proyecto.

4.1 Resultados obtenidos en la transcripción mediante WER

A continuación, se mostrarán los resultados obtenidos en las distintas evaluaciones explicadas en el apartado 3.7.1 para los audios de solapa de una de las salidas grabadas. Es necesario destacar que los resultados pueden variar de unas grabaciones a otras en función del hablante y ruido introducido al micrófono por efectos meteorológicos de viento y lluvia.

4.1.1 Dataset de grabaciones

La creación de un dataset de grabaciones amplio ha sido uno de los objetivos del trabajo, constituyendo uno de los pilares clave para el sistema detector de órdenes y las evaluaciones realizadas.

Mediante salidas en la lancha de instrucción Barrutia de la Escuela Naval Militar, se han realizado grabaciones de 6 salidas a la mar con una duración media de 3 horas por salida, lo que equivale a unos 90 archivos de 2 minutos. Las salidas se ha realizado a lo largo de los meses de febrero y marzo de 2023, con la autorización expresa de los alumnos implicado.

En total, las 6 salidas han producido aproximadamente 500 archivos de transcripción por frases mediante Whisper, que han sido revisados y corregidos manualmente para formar el dataset con las transcripciones reales. Este ha sido el empleado en el entrenamiento del sistema detector de órdenes.

Por último, es necesario destacar que se han empleado 20 de los audios con más información del dataset para la evaluación de las transcripciones de Vosk, Whisper y Whisper mejorado, haciendo un total de 40 minutos de grabación que se consideran representativos para el conjunto del dataset.

4.1.2 Evaluación de transcripción 1: transcripción Whisper

La primera de las pruebas ha tenido como objetivo evaluar la calidad de transcripción proporcionada por Whisper para su posible utilización en el sistema. Para ello, se han comparado las transcripciones completas de Whisper con las obtenidas tras la revisión manual.

El WER medio de todos los archivos obtenido ha sido del 26 % aproximadamente, lo que implica que la transcripción general ha sido relativamente buena. Además, se aprecia que el sistema es capaz de captar sonidos lejanos y transcribirlos, como las respuestas a las órdenes del oficial. Un ejemplo para el dataset de evaluación con los 20 audios es el que se observa en la Figura 4-1. Se puede observar el WER medio comentado y las palabras sustituidas y eliminadas respecto a la transcripción real.

un parece que ya estamos si lo que vas viendo ya en el ultimo momento juegas con
viendo cinco grados de caña diez grados de caña a quedar rumbo cero cuatro cinco
cero caña a la via para todo caña a la via todo parado avante poca estribor ava
para caña a la via atras poca estribor atras poca estribor estribor en atras poc
estribor vale para para todo parado toda la caña a babor para todo dos nudos v
la via o sin caña la voz sin caña la voz cinco grados de caña a babor cinco gra
de esta de oficial de guardia si ha dicho que queria ir a oracion si entre que ll
creo que no porque tardamos diez minutos en llegar otros diez lcinco en atracar c
complica eh yo creo que hasta te lo he dicho eh no pero yo lo veo yo creo el futu
recogemos de tacha bueno esperabamos cuando se pueda recoger esta tacha ahi salen

Word Error Rate: 25.7327%

Substituciones: 1449 Eliminaciones: 1496 Inserciones: 0

Figura 4-1: Resultado de evaluación de la transcripción 1. Fuente: propia

4.1.3 Evaluación de transcripción 2: Órdenes transcritas por Whisper

La siguiente prueba realizada es igual a la primera, pero con la salvedad de que en este caso no se utiliza la transcripción completa, sino solamente las frases que son órdenes y están marcadas en la transcripción revisada. De esta forma, se puede conocer la calidad de la transcripción de estas, obviando el resto del audio.

Archivo: 2023-02-13_18-34-00-bt.txt

Órdenes Whisper revisado: avante ochocientas diez grados de caña a estribor a quedar rumbo dos cinco cero

Frases de ordenes Whisper no revisado: avante lochocientas diez grados de caña estribola a quedar rumbo dos
cinco cero dos cuatro cero corrijo

Word Error Rate : 50.0000%

Substituciones: 16 Eliminaciones: 13 Inserciones: 0

WER MEDIO = 35.495786416972855

Figura 4-2: Ejemplo de evaluación de la transcripción 2. Fuente: propia

El WER medio obtenido en este caso ha sido del 35 %. Se puede llegar a la conclusión de que estas transcripciones que realiza de estas no son precisas. En este sentido, Whisper no ha sido entrenado con vocabulario náutico y algunas de las palabras utilizadas son confundidas por el sistema por otras similares. Por ejemplo, palabras como “babor”, “estribor” o “avante” son transcritas como “vapor”, “estribos” y “avance”.

4.1.4 Evaluación de transcripción 3: Órdenes transcritas por Vosk

Continuando con las evaluaciones, la tercera se ha enfocado en evaluar el rendimiento de Vosk. Para ello se han comparado las órdenes encontradas y transcritas por Vosk con las revisadas manualmente, pudiendo observar si ha conseguido capturar todas ellas y cómo ha sido la calidad obtenida.

El código para la realización del WER hace empleo de una lista numerada *hash* que guarda las frases de Vosk y busca para cada una su transcripción asociada en Whisper revisado en función del tiempo de comienzo de frase, tal y como realizada el sistema de transcripción mejorado del apartado 3.5.4.

Esto permite evaluar la calidad de las transcripciones de las órdenes de Vosk con respecto a la transcripción ideal revisada, pero además da la posibilidad de saber cuántas órdenes del audio no ha conseguido reconocer, aunque en la transcripción ideal sí estén marcadas.

Archivo: 2023-02-16_16-08-01-bt.txt

Órdenes Vosk: dos y medio adelante [unk1] caña a la vía [unk3] adelante [unk1b] quince grados de caña babor

Órdenes ideal: <dos y medio adelante> <caña la vía> <caña la vía> <cinco grados de caña a babor>
<cinco grados de caña a babor>

Word Error Rate sin contar órdenes no encontradas: 55.5556%

Substituciones: 10 Eliminaciones: 11 Inserciones: 0

Órdenes encontradas Vosk / Órdenes en Whisper = 3 / 5

Word Error Rate contando órdenes no encontradas: 64.5455%

WER MEDIO = 45.790959160571234

Figura 4-3: Ejemplo de evaluación de la transcripción 3. Fuente: propia

El WER medio obtenido ha sido de 45 %. Se puede apreciar que Vosk solamente captura la información que posee en su vocabulario, pero además no capta la mayoría de las respuestas a las órdenes, ya que con el micrófono de solapa no se aprecia con claridad esas respuestas, y Vosk desprecia todo aquel audio que no supera un umbral determinado en base al sistema de reducción de ruido empleado.

La ventaja de esto es que no se capturan sonidos lejanos ni conversaciones que no pertenezcan a las del oficial del puente de mando. Sin embargo, tiene como inconveniente la pérdida de posibles respuestas del timonel que pueden ser de utilidad para el sistema detector de órdenes CRF.

4.1.5 Evaluación de transcripción 4: Transcripción de Whisper mejorado con Vosk

Por último, se ha realizado la evaluación de las frases mejoradas de Whisper con las de Vosk y la transcripción completa corregida manualmente. En los resultados se puede observar que la calidad de la transcripción mejora significativamente, reduciendo el problema que tiene Whisper para el reconocimiento de palabras del diccionario náutico, cosa que Vosk sí hace gracias a Kaldi.

Las transcripciones mejoradas, aunque siguen contando con errores, son mucho más uniformes y la mayoría de las faltas ortográficas en palabras náuticas están solucionadas en caso de que Vosk las haya capturado.

babor a quedar rumbo norte venga lo quedar 4 minutos para dar guías lo damos adelante adelante cero nudos diez grados de caña a babor a quedar rumbo adelante poca a quedar rumbo cero tres cero al final lo que u maniobras con los ejes no tanto con las cañas la caña cuando vas tan parado no te hace nada no te hace cinco grados de cañas tribora a quedar rumbo 040 diez por lo menos a rumbo 040 vale vale si a rumbo caña estribor a diez grados de caña a estribor a quedar rumbo cero cuatro cinco a rumbo cero cuatro cinco parece que ya estamos si lo que vas viendo ya en el ultimo momento juegas con las palancas vale pa viendo cinco grados de caña diez grados de caña a quedar rumbo cero cuatro cinco pues al cero cuatro cero caña a la vía para todo caña a la vía todo parado adelante poca estribor adelante poca estribor est para caña a la vía atras poca estribor atras poca estribor estribor en atras poca atras todo atras t estribor vale para para todo parado toda la caña a babor para todo dos nudos vale dos y medio avante la vía o sin caña la voz sin caña la voz cinco grados de caña a babor cinco grados de caña a babor de esta de oficial de guardia si ha dicho que queria ir a oracion si entre que llegamos y atracamos p creo que no porque tardamos diez minutos en llegar otros diez cinco en atracar cinco diez ya ya pero complica eh yo creo que hasta te lo he dicho eh no pero yo lo veo yo creo el futuro tengo un domo ya recogemos de tacha bueno esperabamos cuando se pueda recoger esta tacha ahi salen que lo vean

Word Error Rate: 21.6700%

Substituciones: 1449 Eliminaciones: 1496 Inserciones: 0

Figura 4-4: Resultado de evaluación de la transcripción 4. Fuente: propia

El WER en este caso es un 22%. Por lo tanto, con Whisper mejorado se obtienen los mejores resultados, siendo el motivo por el que se ha considerado la opción de transcripción para el sistema.

4.2 Resultados obtenidos con el sistema detector de órdenes

4.2.1 Entrenamiento por transcripciones completas de 2 minutos

El sistema detector de órdenes estudiado en la sección 3.6 y su módulo de entrenamiento del apartado 3.6.3 han sido entrenados gracias al dataset de entrenamiento mencionado en la sección 4.1.1 a través de las transcripciones de los audios de dos minutos. Para ello, los audios del dataset se han transcrito, corregido y marcado según los procedimientos explicados en apartados anteriores.

Inicialmente el entrenamiento se realizó en los dos modelos disponibles, BiLSTM-CRF y CRF simple. A pesar de que el tiempo de entrenamiento del modelo BiLSTM-CRF es de varias horas con respecto al modelo CRF que dura varios minutos, la precisión obtenida en este es menor.

Respecto al tamaño del conjunto de entrenamiento BiLSTM-CRF, se han utilizado cinco particiones de los datos de entrenamiento (`k_folds`) y se han recorrido 150 veces para minimizar la pérdida (`epochs`).

La predicción teniendo en cuenta toda la transcripción de cada archivo de dos minutos da lugar a resultados que tienen en cuenta el contexto de las órdenes para ser marcados, facilitando la identificación de las respuestas a las órdenes. Finalmente, la precisión aproximada obtenida en ambos sistemas con respecto a las etiquetas I y B (las que indican donde comienza y termina una orden) es del 73% y 52% respectivamente, con una medida F alrededor del 65 %.

	precision	recall	f1-score	support
B	0.68	0.56	0.62	275
I	0.75	0.63	0.68	961
micro avg	0.73	0.61	0.67	1236
macro avg	0.71	0.60	0.65	1236
weighted avg	0.73	0.61	0.67	1236

Figura 4-5: Resultado entrenamiento CRF por archivos. Fuente: propia

	precision	recall	f1-score	support
B	0.528	0.743	0.617	214
I	0.513	0.908	0.655	687
micro avg	0.516	0.869	0.647	901
macro avg	0.520	0.826	0.636	901
weighted avg	0.516	0.869	0.646	901

Figura 4-6: Resultado entrenamiento BiLSTM-CRF por archivos. Fuente: propia

Tras estos resultados se ha decidido utilizar el modelo entrenado CRF en el sistema. Además, tras comprobar dónde se producen los errores que producen esa precisión y que órdenes no son detectadas, se ha llegado a la conclusión de que son las siguientes:

- 1- Órdenes no dichas con precisión por el oficial de guardia. Por ejemplo, la orden “*Toda la caña a babor a quedar rumbo dos uno cero*”, no ha sido detectada en la mayoría de los casos si se ha dado como “*al dos uno cero*”, “*cae al dos uno cero*” o “*rumbo dos uno cero*”, puesto que el modelo entrenado ha considerado estas frases como respuestas a la orden, descartándolas.
- 2- Órdenes erróneas, tal y como “*Avante todo las dos*” en lugar de “*Avante toda las dos*”. Sin embargo, este error se podría subsanar con más datos de entrenamiento que marcasen este error como una orden.
- 3- Cuando una orden se da repetidamente, puede ser considerada como respuesta. Como ejemplo, la orden “*Palada avante babor*”, si fuese dicha dos veces de forma consecutiva, el sistema podría considerar la segunda como una respuesta a la primera orden, no siendo marcada.
- 4- Órdenes dadas por el comandante de la lancha al corregir al alumno. Por ejemplo “*Toca babor y estribor de guardia*” puede llegar a ser marcada. Sin embargo, el sistema entrenado puede tener en cuenta palabras en el contexto como “*manda*”, “*toca*” o “*cae*”, por lo que este error también puede minimizarse con un mayor número de datos de entrenamiento.

Como se puede ver, los problemas mencionados pueden ser solucionados con un mayor número de transcripciones de salidas a la mar, que por motivos de tiempo y calendario no han sido posibles.

4.2.2 Entrenamiento por frases de las transcripciones

Para la realización del entrenamiento por frases se ha seguido el mismo procedimiento que el realizado por archivos. En este caso, los resultados de precisión obtenidos para los dos sistemas de entrenamiento CRF y BiLSTM-CRF han sido del 65% y 62% respectivamente. Como se puede observar, ambos son valores inferiores a los obtenidos con el entrenamiento por archivos.

	precision	recall	f1-score	support
B	0.62	0.64	0.63	1288
I	0.66	0.68	0.67	4612
micro avg	0.65	0.67	0.66	5900
macro avg	0.64	0.66	0.65	5900
weighted avg	0.65	0.67	0.66	5900

Figura 4-7: Resultado entrenamiento CRF por frases. Fuente: propia

	precision	recall	f1-score	support
B	0.610	0.656	0.632	250
I	0.618	0.769	0.685	861
micro avg	0.616	0.743	0.674	1111
macro avg	0.614	0.712	0.659	1111
weighted avg	0.616	0.743	0.673	1111

Figura 4-8: Resultado entrenamiento BiLSTM-CRF por frases. Fuente: propia

4.2.3 Entrenamientos por transcripciones y frases mejoradas

Una vez realizadas las pruebas de entrenamientos comentadas en apartados anteriores, el resultado máximo ha sido el obtenido con CRF por transcripciones con un 73 % de precisión.

Con objeto de mejorar la máxima precisión obtenida hasta el momento se han aplicado una serie de mejoras sobre los modelos de entrenamiento por frases y por transcripciones, dando lugar a los dos modelos mejorados.

En primer lugar, la modificación que se ha introducido ha sido la implementación de un archivo de vocabulario náutico. El objetivo de este es realizar un filtrado de las posibles órdenes obtenidas en el entrenamiento, descartando aquellas que no tengan relación con el vocabulario.

Tras realizar un entrenamiento con los mismos datos que los modelos anteriores, el resultado de precisión obtenido para el entrenamiento de transcripciones ha sido de un 76% con un puntaje del 68%, lo que mejora los resultados hasta el momento.

	precision	recall	f1-score	support
B	0.73	0.50	0.59	22
I	0.77	0.65	0.71	83
micro avg	0.76	0.62	0.68	105
macro avg	0.75	0.58	0.65	105
weighted avg	0.76	0.62	0.68	105

Figura 4-9: Resultado entrenamiento CRF por transcripciones mejorado 1. Fuente: propia

En el caso de la implementación para el entrenamiento por frases, se ha eliminado la etiqueta B suponiendo que la mayoría de las órdenes se darán en frases separadas, por lo que no hace falta marcar el comienzo de la orden. Sin embargo, en el caso de que en una frase se encontrase más de una orden, toda la frase quedaría marcada con la etiqueta I sin hacer distinción entre ambas. Por otra parte, también se ha añadido el vocabulario ya implementado en el entrenamiento por archivos.

Los resultados obtenidos han sido de un 66 % de precisión y un 68 % de puntaje.

	precision	recall	f1-score	support
I	0.66	0.71	0.68	5900
micro avg	0.66	0.71	0.68	5900
macro avg	0.66	0.71	0.68	5900
weighted avg	0.66	0.71	0.68	5900

Figura 4-10: Resultado entrenamiento CRF por frases mejorado 1. Fuente: propia

La conclusión obtenida es que el mejor modelo para nuestros datos de entrenamiento ha sido el mejorado por archivos utilizando CRF, siendo el utilizado finalmente.

Las predicciones obtenidas en una de las transcripciones de prueba pueden observarse en el Anexo IV: Predicciones detector de órdenes.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

El *Deep Learning* y la inteligencia artificial están transformando el mundo y acelerando el avance de la tecnología a un ritmo sin precedentes. La capacidad de estas tecnologías para procesar enormes cantidades de datos y aprender de ellos, ha permitido avances en áreas como la visión por computadora, la robótica y la atención médica. Pero uno de los campos donde la implementación de estos está teniendo un mayor impacto es el reconocimiento de voz. Con sistemas cada vez más precisos como Whisper, que ha sido el último avance en este campo, se abre un nuevo paradigma para la comunicación entre humanos y máquinas. En un futuro no muy lejano, estos sistemas podrían permitir el control de dispositivos electrónicos solo con la voz, lo que sería de gran utilidad para personas con discapacidades físicas o para situaciones donde las manos no están disponibles. Además, esta tecnología podría tener un gran impacto en el sector empresarial y en la productividad, permitiendo la automatización de tareas y la reducción de errores.

La principal conclusión que se extrae de este trabajo es que el uso de las nuevas tecnologías está haciendo que campos como el reconocimiento de voz, que tenían escasa evolución hasta principios de siglo, ahora lo hagan de forma exponencial, teniendo repercusión en otras áreas que pueden hacer uso de él.

Respecto a los objetivos planteados en el capítulo 1, se han cumplido satisfactoriamente los 4 hitos planteados.

En primer lugar, se han conseguido realizar las grabaciones necesarias a bordo de las lanchas de instrucción para entrenar y probar el sistema. Además, el sistema de transcripción de Whisper mejorado ha conseguido superar la precisión de los modelos de Whisper y Vosk por separado, capturando la mayor parte de las órdenes.

Por otra parte, gracias a las cinco grabaciones realizadas con un total de diez horas de audio transcritas y revisadas manualmente con la ayuda de Whisper, se ha obtenido una precisión del 76 % en el sistema detector de órdenes. A pesar de no ser un mal resultado, un mayor número de grabaciones podría haber aumentado la precisión.

Por último, se ha conseguido realizar la simulación de una de las salidas con éxito, gracias al software de SIMDIS y las cartas náuticas descargadas proporcionadas por el Instituto Hidrográfico de la Marina.

El principal inconveniente observado durante la elaboración del trabajo ha sido el tiempo de procesamiento, tanto por Whisper en la tarea de transcripción usando su modelo más preciso, como por el sistema de entrenamiento del detector de órdenes BiLSTM-CRF. Por la cantidad de transcripciones a usar, algunos entrenamientos y transcripciones han llegado a durar más de cinco horas.

Además, tras estos entrenamientos se ha notado la importancia de la calidad en los datos de entrenamiento. Malas transcripciones o etiquetado de los datos puede dar lugar a resultados completamente distintos.

Por último, a título personal he aprendido la importancia del uso de las nuevas tecnologías en el ámbito militar y la inmensa cantidad de posibilidades que abre para mejorar la eficiencia y la precisión de los sistemas actuales. He adquirido conocimientos acerca de las herramientas y sistemas empleados hoy en día para las tareas de reconocimiento de voz, NLP y *Deep Learning*. También me he dado cuenta de que el uso de estas nuevas herramientas es clave para mantener el sector de la defensa a la vanguardia tecnológica, y estoy emocionado de ver cómo estas tecnologías seguirán evolucionando y cambiando la forma en que se llevan a cabo las operaciones militares en el futuro.

5.2 Líneas futuras

Tras haber conseguido la elaboración de un sistema capaz de simular una salida a la mar a partir de grabaciones y telemetría, se proponen las siguientes líneas futuras para continuar con el desarrollo del trabajo o aplicar mejoras sobre él:

- Mejora de la eficiencia del sistema de transcripción mediante cambios en el sistema de transcripción de Whisper o Vosk. Whisper podría añadir modelos de transcripción más precisos en el futuro. Además, el modelo más grande en español de Vosk no ha sido utilizado al no admitir la implementación de Kaldi, cosa que en un futuro podría cambiar.
- Mejora del sistema detector de órdenes a través del aumento de datos de entrenamiento o cambios en el código.
- Afinamiento de uno de los modelos de Whisper a través un mayor número de datos, reduciendo el error de transcripción en términos náuticos.
- Creación de un sistema complementario para capturar órdenes en cubierta o zonas ajenas al puente de mando.

En resumen, las mejoras se basan en reducir los errores del sistema y aumentar la precisión de este, incrementando la fiabilidad de las simulaciones realizadas. Por otro lado, se puede mejorar el alcance del sistema si se consiguen capturar las órdenes en cubierta durante las distintas maniobras. Con estas, se podría realizar una representación en una pantalla en el puente de mando para información del resto de personal.

6 BIBLIOGRAFÍA

- [1] MINSITERIO DE DEFENSA, «El almirante general Antonio Martorell toma posesión como jefe de Estado Mayor de la Armada.,» *Revista Española de Defensa*, pp. 6-7, Marzo 2021.
- [2] R. Curry, *Audio mining - Unabridged Guide*, Brisbane: Emereo Publishing, 2012.
- [3] N. F. G. Á. H. M. G.-C. J. P. y. T. G. R. Milagros Fernández-Gavilanes, «Procesamiento de audio y lenguaje natural para el análisis de ejercicios en lanchas de instrucción,» *IX Congreso Nacional de I+D en Defensa y Seguridad*, 2022.
- [4] P. J. & B. B. & C. W. &, «Information density in bridge team communication and miscommunication,» World Maritime University, 2013.
- [5] IMO, *IMO STANDARD MARINE COMMUNICATION PHRASES - Resolution A.918(22)*, IMO, 2001.
- [6] A. N. T. B. Peter John, «Using Computer Dialogue Systems for Providing aStudent-Centred Teaching Approach in SMCP,» de *International Maritime English Conference*, Johor Bahru, 2015.
- [7] H. S. T. C. a. H. L. Xiang Fu, «Application Research on Combat Simulation Automation based on Intelligent Speech Interaction,» *Journal of Physics: Conference Series*, 2021.
- [8] G. Okamoto, *APPLICATION OF SPEECH RECOGNITION ALGORITHM TO INTERNATIONAL VHF COMMUNICATION*, Tokyo: Tokyo University of Marine Science And Technology (TUMSAT), 2020.
- [9] K. Wiles, «Deafness, 'Visible Speech' and Alexander Graham Bell,» *HISTORY TODAY*, p. 1, 26 01 2015.
- [10] M. Pinola, «PCWORLD,» 2 11 2011. [En línea]. Available: https://www.pcworld.com/article/477914/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html. [Último acceso: 31 01 2023].
- [11] D. Spicer, «<https://computerhistory.org/>,» 09 06 2021. [En línea]. Available: <https://computerhistory.org/blog/audrey-alexa-hal-and-more/>. [Último acceso: 31 01 2023].

- [12] «An Overview of the SPHINX Speech Recognition System,» *IEEE TRANSACTIONS ON ACOUSTICS SPEECH AND SIGNAL PROCESSING*, vol. 38, n° 1, p. 11, 1990.
- [13] W. Telecom, «Brief History of Automatic Speech Recognition,» de *Speech and Audio Signal Processing: Processing and Perception of Speech and Music*, Wiley, 2011, pp. 40-58.
- [14] A. A. Larios, Diccionario español/inglés para el aprendizaje de vocabulario utilizando una interfaz de voz, Puebla, México: Departamento de Ingeniería en Sistemas Computacionales, 1999.
- [15] H. Z. A. M. M. A. Alexei Baevski, «wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations,» Cornell University, 2020.
- [16] S. W. Jaesong Lee, «Intermediate Loss Regularization for CTC-based Speech Recognition,» ICASSP 2021, 2021.
- [17] P. K. F. K. M. M. R. M. A. M. Huy Phan, «Audio Scene Classification with Deep Recurrent Neural Networks,» arxiv, 2017.
- [18] I. P. H. H. C. Philipp Koehn, «Scalable Modified Kneser-Ney Language Model Estimation,» de *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013.
- [19] G. R. Casajús, Predicción de tipo de buque utilizando datos AIS y técnicas de inteligencia artificial, Marín: Centro Universitario de la Defensa, 2022.
- [20] R. N. Keiron O'Shea, «An Introduction to Convolutional Neural Networks,» arxiv, 2015.
- [21] AMAZON, «AMAZON,» [En línea]. Available: https://www.amazon.es/echo-dot-3-generacion-altavoz-inteligente-con-alexa-tela-de-color-antracita/dp/B07PHPXHQS/ref=d_p13n_ds_purchase_sim_1p_dp_desktop_sccl_1_3/262-1966738-7846952?pd_rd_w=Oxt7v&content-id=amzn1.sym.dc5985cb-cfda-4321-9e2d-5adc044b9d37&p. [Último acceso: 02 02 2023].
- [22] M. Chernyshevich, «IHS R&D Belarus: Cross-domain extraction of product features using CRF,» vol. Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014), p. 309–313, 2014.
- [23] J. L. Gonzalez, «medium,» 08 02 2018. [En línea]. Available: <https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2>. [Último acceso: 11 03 2023].
- [24] W. Khepri, «Medium,» 2 11 2018. [En línea]. Available: <https://medium.com/@williamkhepri/redes-neuronales-que-son-a64d022298e0>. [Último acceso: 22 02 2023].
- [25] P. Yadav, «medium,» 30 08 2018. [En línea]. Available: <https://medium.com/@mail2princeyadav/machine-learning-vs-deep-learning-b5c5a4fc5c>. [Último acceso: 16 02 2023].
- [26] DATADEMIA, «datademia,» 2023. [En línea]. Available: <https://datademia.es/blog/que-es-deep-learning-y-que-es-una-red-neuronal>. [Último acceso: 16 02 2023].
- [27] intellipaat, «intellipaat.com,» 4 03 2023. [En línea]. Available: <https://intellipaat.com/blog/what-is->

- lstm/#:~:text='%20LSTM%20stands%20for%20long%20short,especialmente%20in%20sequence%20prediction%20problems.. [Último acceso: 21 03 2023].
- [28] S. Kostadinov, «towardsdatascience,» 16 03 2017. [En línea]. Available: <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>. [Último acceso: 21 03 2023].
- [29] H. Pedamallu, «Medium,» 14 11 2020. [En línea]. Available: <https://medium.com/analytics-vidhya/rnn-vs-gru-vs-lstm-863b0b7b1573>. [Último acceso: 22 02 2023].
- [30] J. C. R. C. A. C. Y. G. Y. L. H. L. S. S. D. S. A. S. Z. Z. Eric Battenberg, «Exploring Neural Transducers for End-to-End Speech Recognition,» Cornell University. Arxiv, 2017.
- [31] aprendemachinelearning, «www.aprendemachinelearning.com,» 29 11 2018. [En línea]. Available: <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>. [Último acceso: 16 02 2023].
- [32] BAIDU, «<http://research.baidu.com/>,» 31 10 2017. [En línea]. Available: <http://research.baidu.com/Blog/index-view?id=90>. [Último acceso: 05 02 2023].
- [33] E. B. a. S. A. B. Bahmei, «CNN-RNN and Data Augmentation Using Deep Convolutional Generative Adversarial Network for Environmental Sound Classification,» *IEEE Signal Processing Letters*, vol. 29, pp. 682-686, 2022.
- [34] N. J. a. A. -r. M. A. Graves, «Hybrid speech recognition with Deep Bidirectional LSTM,» *IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 273-278, 2013.
- [35] I. Goodfellow y Y. Bengio, «6.2.2.3 Softmax Units for Multinoulli Output Distributions,» MIT Press, Courville, Aaron, 2016.
- [36] W. Chan, N. Jaitly, Q. Le y O. Vinyals, «Listen, attend and spell: A neural network for large vocabulary conversational speech recognition,» de *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2016.
- [37] A. Graves, «Sequence Transduction with Recurrent Neural Networks,» de *First published in the International Conference of Machine Learning (ICML) 2012 Workshop on Representation Learning*, 2012.
- [38] N. J. Q. V. L. O. V. William Chan, «Listen, Attend and Spell,» ARXIV, Cornell University, 2015.
- [39] D. S. Q. V. L. O. V. I. S. S. B. Navdeep Jaitly, «A Neural Transducer,» arxiv, Cornell University, 2015.
- [40] H.-G. K. M.-J. L. J. L. S. O. H. L. Y. S. C. Tae Gyoong Kang, «Learning Monotonic Alignments with Source-Aware GMM Attention,» de *ICLR 2021 Conference Blind Submission*, 2021.
- [41] M. S. K. R. F. B. Hasim Sak, «Recurrent Neural Aligner: An Encoder-Decoder Neural Network Model for,» de *INTERSPEECH 2017*, 2017.
- [42] T. N. S. Y. W. R. P. P. N. Z. C. A. K. R. J. W. K. R. E. G. N. J. B. L. J. C. M. B. Chung-Cheng Chiu, «State-of-the-art Speech Recognition With Sequence-to-Sequence Models,» Cornell University, ARXIV, 2018.

- [43] analyticsvidhya, «analyticsvidhya.com,» 9 10 2020. [En línea]. Available: <https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/>. [Último acceso: 05 02 2023].
- [44] A. G. a. D. Britz, «ai.googleblog.com,» GOOGLE, 11 04 2017. [En línea]. Available: <https://ai.googleblog.com/2017/04/introducing-tf-seq2seq-open-source.html>. [Último acceso: 05 02 2023].
- [45] M. d. Gangi, «towardsdatascience,» 17 01 2023. [En línea]. Available: <https://towardsdatascience.com/introduction-to-speech-enhancement-part-1-df6098b47b91>. [Último acceso: 21 03 2023].
- [46] J. L. Roux y E. Vincent, «Consistent Wiener Filtering for Audio Source Separation,» *IEEE Signal Processing Letters*, vol. 20, nº 3, pp. 217-220, 2013.
- [47] S. Butterworth, « On the Theory of Filter Amplifiers,» *Experimental Wireless and the Wireless Engineer*, nº 7, pp. 536-541, 1930.
- [48] Scipy, «Scipy-cookbook,» 26 04 2020. [En línea]. Available: <https://scipy-cookbook.readthedocs.io/items/ButterworthBandpass.html>. [Último acceso: 13 02 2023].
- [49] F. F. Rosângela Coelho, «Adaptive Gain Methods to Improve Speech Intelligibility under Reverberation,» de *XXXVI Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, 2018.
- [50] M. T. a. T. Q. G. Tim Sainburg, «Finding, visualizing, and quantifying latent structure across diverse animal vocal repertoires,» *PLoS computational biology*, vol. 16, nº 10, 2020.
- [51] WIKIPEDIA, «Wikipedia.org,» 11 12 2021. [En línea]. Available: https://en.wikipedia.org/wiki/Noise_gate. [Último acceso: 13 02 2023].
- [52] timsainb, «GITHUB,» 21 09 2022. [En línea]. Available: <https://github.com/timsainb/noisereduce>. [Último acceso: 13 02 2023].
- [53] J.-M. Valin, «A Hybrid DSP/Deep Learning Approach to Real-Time Full-Band Speech Enhancement,» *Proceedings of IEEE Multimedia Signal Processing*, 2018.
- [54] WIKIPEDIA, «wikipedia.org,» 06 12 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Red_neuronal_recurrente. [Último acceso: 16 02 2023].
- [55] WIKIPEDIA, «wikipedia.org,» 30 01 2023. [En línea]. Available: https://en.wikipedia.org/wiki/Gated_recurrent_unit. [Último acceso: 16 02 2023].
- [56] WIKIPEDIA, «wikipedia.org,» 16 02 2023. [En línea]. Available: [https://en.wikipedia.org/wiki/Long_short-term_memory#:~:text=Long%20short%2Dterm%20memory%20\(LSTM,networks%2C%20LSTM%20has%20feedback%20connections..](https://en.wikipedia.org/wiki/Long_short-term_memory#:~:text=Long%20short%2Dterm%20memory%20(LSTM,networks%2C%20LSTM%20has%20feedback%20connections..) [Último acceso: 16 02 2023].
- [57] Mozilla Emerging Technologies, «jmvalin.ca,» 2017. [En línea]. Available: <https://jmvalin.ca/demo/rnnoise/>. [Último acceso: 03 02 2023].
- [58] J.-M. Valin, «GITLAB,» 2022. [En línea]. Available: <https://gitlab.xiph.org/xiph/rnnoise>. [Último acceso: 13 02 2023].

- [59] L.-Y. D. J. C. L.-T. C. C. X. & Q. T. Min Xu, «HMM-Based Audio Keyword Generation,» de *Advances in Multimedia Information Processing - PCM 2004, 5th Pacific Rim Conference on Multimedia*, Tokyo, Japan, 2004.
- [60] P. Bell, Context-dependent phone models, ASR Lecture 2, Edimburgh: University of Edimburgh, 2019.
- [61] P. Bell, Context-dependent phone models, ASR Lecture 6, University of Edimburgh, 2019.
- [62] D. Amos, «realpython.com,» 2018. [En línea]. Available: <https://realpython.com/python-speech-recognition/>. [Último acceso: 29 01 2023].
- [63] F. Bimbot, «A Tutorial on Text-Independent Speaker Verification,» *EURASIP Journal on Advances in Signal Processing*, n° Article number: 101962, p. 22, 2004.
- [64] K. Doshi, «towardsdatascience.com,» 25 03 2021. [En línea]. Available: <https://towardsdatascience.com/audio-deep-learning-made-simple-automatic-speech-recognition-asr-how-it-works-716cfce4c706>. [Último acceso: 02 02 2023].
- [65] J. W. K. T. X. G. B. C. M. I. S. Alec Radford, «arxiv.org,» 6 12 2022. [En línea]. Available: <https://arxiv.org/abs/2212.04356>. [Último acceso: 29 01 2023].
- [66] D. Rothman, Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more, Birmingham: Packt Publishing, 2021.
- [67] OPENAI, «OPENAI,» 21 9 2022. [En línea]. Available: <https://openai.com/blog/whisper/>. [Último acceso: 29 01 2023].
- [68] OPENAI, «pypi.org,» 26 10 2022. [En línea]. Available: <https://pypi.org/project/pywhisper/>. [Último acceso: 29 01 2023].
- [69] VOSK, «pypi.org,» 15 12 2022. [En línea]. Available: <https://pypi.org/project/vosk/>. [Último acceso: 29 01 2023].
- [70] KALDI, «kaldi.org,» 8 2 2020. [En línea]. Available: <https://kaldi-asr.org/doc/kws.html>. [Último acceso: 29 01 2023].
- [71] P. M. a. H. B. IS. Madikeri, «Combining SGMM speaker vectors and KL-HMM approach for speaker diarization,» de *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, South Brisbane, Australia, 2015.
- [72] M. Vyas, «A GAUSSIAN MIXTURE MODEL BASED SPEECH,» *Signal & Image Processing : An International Journal (SIPIJ)*, vol. 4, n° 4, pp. 1-10, 2013.
- [73] Z. -H. T. J. H. L. H. a. J. J. I. López-Espejo, «Deep Spoken Keyword Spotting: An Overview,» *IEEE Access*, vol. 10, pp. 4169-4199, 30 12 2021.
- [74] K. B. Marín, Herramientas para la detección de menciones a palabras clave, Marín: Centro Universitario de la Defensa, 2020.
- [75] S. K. D. P. J. T. D. Y. a. O. Y. Guoguo Chen, «QUANTIFYING THE VALUE OF PRONUNCIATION LEXICONS FOR KEYWORD SEARCH,» *ICASSP*, Baltimore, 2013.

- [76] cmusphinx, «cmusphinx,» 13 10 2022. [En línea]. Available: <https://cmusphinx.github.io/>. [Último acceso: 29 01 2023].
- [77] J. Hui, «jonathan-hui.medium,» 9 9 2019. [En línea]. Available: <https://jonathan-hui.medium.com/speech-recognition-gmm-hmm-8bb5eff8b196>. [Último acceso: 29 1 2023].
- [78] SpeechRecognition, «pypi.org,» 4 12 2022. [En línea]. Available: <https://pypi.org/project/SpeechRecognition/>. [Último acceso: 29 01 2023].
- [79] GOOGLE, «cloud.google,» [En línea]. Available: <https://cloud.google.com/speech-to-text?hl=es>. [Último acceso: 29 01 2023].
- [80] S. Beliga, «Keyword extraction: a review of methods and approaches,» University of Rijeka, Department of Informatics, Radmile Matejčić 2, 51 000 Rijeka, Croatia.
- [81] U.S. Naval Research Laboratory, «simdis.nrl.navy.mil,» 04 04 2019. [En línea]. Available: <https://simdis.nrl.navy.mil/SimdisPresentation.aspx>. [Último acceso: 30 01 2023].
- [82] PYPI, «pypi.org/,» 12 12 2022. [En línea]. Available: <https://pypi.org/project/folium/>. [Último acceso: 30 01 2023].
- [83] S. Feldman, 8 01 2019. [En línea]. Available: <https://www.statista.com/chart/16567/popular-programming-languages/>. [Último acceso: 02 03 2023].
- [84] thisisnic, «GITHUB,» 10 01 2021. [En línea]. Available: <https://github.com/thisisnic/ABSA>. [Último acceso: 21 02 2023].
- [85] J. Hugunin, «PYPI.ORG,» 5 02 2023. [En línea]. Available: <https://pypi.org/project/numpy/>. [Último acceso: 14 02 2023].
- [86] Bastibe, «PYPI,» 27 09 2022. [En línea]. Available: <https://pypi.org/project/soundfile/>. [Último acceso: 14 02 2023].
- [87] A. Rose, «PYPI,» 9 04 2021. [En línea]. Available: <https://pypi.org/project/soundpy/>. [Último acceso: 14 02 2023].
- [88] J. Schlinkert, «PYPI,» 08 02 2020. [En línea]. Available: <https://pypi.org/project/utlis/>. [Último acceso: 14 02 2023].
- [89] nikvaessen, «PYPI,» 06 09 2022. [En línea]. Available: <https://pypi.org/project/jiwer/>. [Último acceso: 14 02 2023].
- [90] SoundPy, «aislynrose.bitbucket.io,» 2020. [En línea]. Available: https://aislynrose.bitbucket.io/auto_examples/plot_filter_out_noise.html. [Último acceso: 13 02 2023].
- [91] VOSK, «ALPHACEPHEI,» [En línea]. Available: <https://alphacephei.com/vosk/models>. [Último acceso: 14 02 2023].
- [92] W. X. K. Y. Zhiheng Huang, «Bidirectional LSTM-CRF Models for Sequence Tagging,» arXiv:1508.01991v1, 2015.
- [93] spaCy, «spacy.io,» [En línea]. Available: <https://spacy.io/models/es>. [Último acceso: 23 02 2023].

- [94] scikit-learn, «scikit-learn.org,» [En línea]. Available: <https://scikit-learn.org/stable/>. [Último acceso: 23 02 2023].
- [95] wikipedia, «wikipedia.org,» 7 12 2022. [En línea]. Available: <https://es.wikipedia.org/wiki/L-BFGS>. [Último acceso: 23 02 2023].
- [96] pytorch, «pytorch.org,» [En línea]. Available: <https://pytorch.org/>. [Último acceso: 23 02 2023].
- [97] S. P. K. V. S. C. G. Youngja Park, «An Empirical Analysis of Word Error Rate and Keyword Error Rate,» de *International Conference on Spoken*, Brisbane, Australia, 2008.
- [98] Instituto Hidrográfico de la Marina, «ideihm,» [En línea]. Available: <https://ideihm.covam.es/portal/>. [Último acceso: 10 03 2023].
- [99] Open Geospatial Consortium, «ogc.org,» [En línea]. Available: <https://www.ogc.org/>. [Último acceso: 10 03 2023].
- [100] talent.com, «talent.com,» [En línea]. Available: <https://es.talent.com/salary?job=programador#:~:text=El%20salario%20programador%20promedio%20en,%E2%82%AC%2014%2C62%20por%20hora..> [Último acceso: 23 03 2023].
- [101] T. Bäckström, «wiki.aalto.fi,» 03 09 2019. [En línea]. Available: <https://wiki.aalto.fi/display/ITSP/Wake-word+and+keyword+spotting>. [Último acceso: 29 01 2023].
- [102] M. D. S. L. V. A. G. K. M. I. Mahbubul Alam, «Survey on Deep Neural Networks in Speech and Vision Systems,» Tennessee State University, Nashville, 2019.
- [103] E. B. Hajo Nils Krabbenhöft, «TEVR: IMPROVING SPEECH RECOGNITION,» Cornell University, 2022.
- [104] NUMPY, «PYPI,» 26 12 2022. [En línea]. Available: <https://pypi.org/project/numpy/>. [Último acceso: 03 02 2023].
- [105] M. S. T. K. Xuechen Liu, «Learnable MFCCs for Speaker Verification,» ISCAS, 2021.
- [106] META, «paperswithcode.com,» [En línea]. Available: <https://paperswithcode.com/method/bilstm>. [Último acceso: 21 02 2023].

ANEXO I: IMPLICACIONES SOCIALES Y ECONÓMICAS

El desarrollo de un sistema de transcripción y simulación como el del presente trabajo requiere principalmente tiempo, tanto para la elaboración del código como para los procesos de grabación y recopilación de audios. Por otro lado, también se requieren componentes físicos de grabación como los dos micrófonos empleados, la Raspberry Pi o el cableado.

Realizando una estimación económica del coste de diseño y elaboración del proyecto, se tienen que tener en cuenta las siguientes partes:

- La fase de investigación, búsqueda de librerías, herramientas y algoritmos a emplear ha sido una fase de gran importancia, dedicándole más de 80 horas, principalmente en la búsqueda de alternativas a la transcripción y detección de órdenes. También se han investigado herramientas de reducción de ruido para encontrar la que más se ajusta al modelo. Supone el 25,81 % del proyecto.
- La fase de grabación ha requerido la instalación de los micrófonos antes de cada salida y la recogida de estos al final. En el primera grabación también se estuvo presente en la lancha durante toda la salida para comprobar el funcionamiento y posibles errores. En total se corresponde con unas 15 horas dedicadas, lo que supone un 4,84 % del trabajo.
- La siguiente fase ha sido la de programación, sin duda la más extensa, suponiendo unas 180 horas que se han repartido en la programación del sistema de transcripción de Whisper mejorado, los sistemas de evaluación de la transcripción, el detector de órdenes y código para simulación. En total se corresponde con el 58,06 % del trabajo realizado.
- Aunque no se tendrá en cuenta en el presupuesto, si hay que considerar el tiempo de procesamiento del ordenador para la obtención de las transcripciones con Whisper y el entrenamiento del detector de órdenes ha sido muy elevado, superando las 100 horas de trabajo continuo de CPU.
- Por último, se incluye la fase de lectura y comprensión del código del proyecto PANNACOTA y la corrección de errores del sistema final, así como las pruebas realizadas, que suponen un del proyecto con 35 horas y un 11,29 % del proyecto.

En la Figura A1. 1 se puede ver el reparto de los distintos tiempos dedicados a cada tarea, y en la Tabla A1. 1 el presupuesto total desglosado al incluir los costes por hora, asignados respecto a una valoración personal y con base en otros proyectos similares [100].

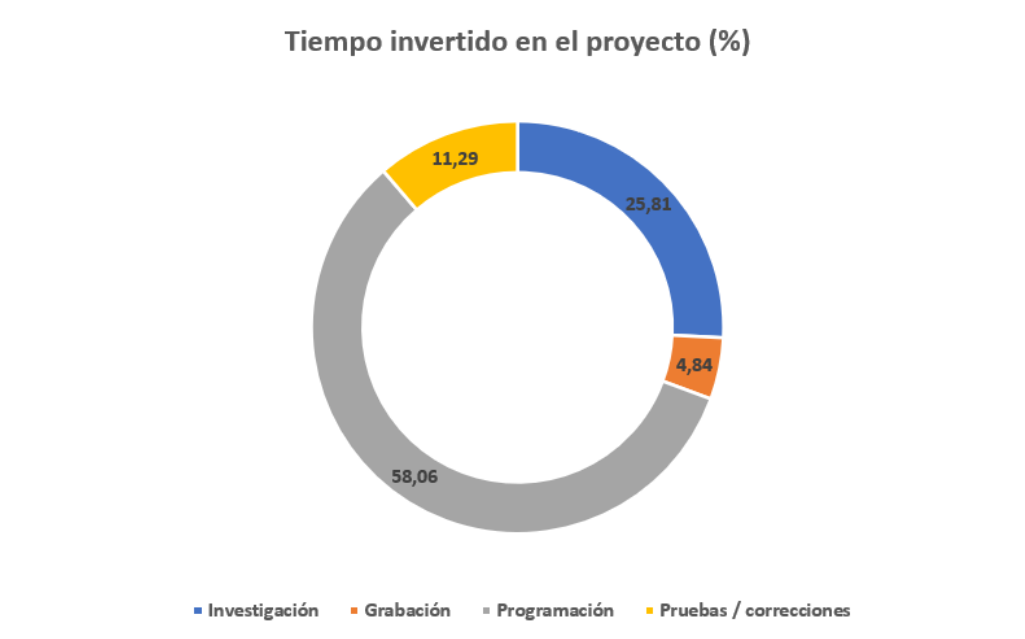


Figura A1. 1: Reparto de tiempo invertido en el proyecto. Fuente: propia

Fase	Horas	Coste/hora (€/h)	Coste total (€)
Investigación	80	5	400
Grabación	15	35	525
Programación	180	15	2700
Pruebas/correcciones	35	5	175

Tabla A1. 1: Desglose de costes por fases. Fuente: propia

Por lo tanto, el coste total asciende a 3.800 €, que junto al coste de los micrófonos (120 €) y de la Raspberry Pi (150€), hacen un total de 4.070 €.

Respecto a las implicaciones sociales, el sistema planteado en este trabajo es capaz de tener una repercusión directa en la formación de los alumnos y su aprendizaje, lo que afecta indirectamente a la mejora de la defensa nacional. Además, podría ser llevado a otros ámbitos ajenos al entorno militar. Con los ajustes adecuados en el código y vocabulario establecido, el sistema sería capaz de realizar la búsqueda de palabras en conferencias o cualquier tipo de audio con voz.

ANEXO II: REFLEXIONES ÉTICAS Y SOCIALES

El uso de tecnología avanzada en el ámbito militar permite mejorar la precisión y eficacia en la evaluación de ejercicios realizados a bordo de los buques. Sin embargo, este sistema plantea diversas reflexiones éticas y sociales que deben ser consideradas.

En primer lugar, se debe tener en cuenta el derecho a la privacidad de los miembros de la tripulación, quienes pueden sentirse incómodos al saber que sus conversaciones están siendo grabadas y analizadas. Es importante establecer protocolos claros y transparentes sobre el uso y almacenamiento de estas grabaciones, así como garantizar la confidencialidad de la información recopilada. A pesar de que el sistema ha sido utilizado en un entorno de enseñanza con autorización verbal del personal, sería conveniente la firma de una cláusula de confidencialidad, de acuerdo con la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.

En virtud del artículo 5 de esta misma ley, se especifica:

Los interesados a los que se soliciten datos personales deberán ser previamente informados de modo expreso, preciso e inequívoco.

Se recoge además la información que deberá recibir el personal implicado, destacando los siguientes puntos:

- De la existencia de un fichero o tratamiento de datos de carácter personal, de la finalidad de la recogida de éstos y de los destinatarios de la información.
- Del carácter obligatorio o facultativo de su respuesta a las preguntas que les sean planteadas.
- De las consecuencias de la obtención de los datos o de la negativa a suministrarlos.
- De la posibilidad de ejercitar los derechos de acceso, rectificación, cancelación y oposición.
- De la identidad y dirección del responsable del tratamiento o, en su caso, de su representante.

Además, es importante considerar el impacto que este sistema puede tener en la cultura organizacional y en las relaciones interpersonales entre los miembros de la tripulación. La presencia de un sistema de vigilancia constante puede generar desconfianza y tensión en el ambiente laboral, lo que puede afectar negativamente el desempeño de la tripulación.

Otro aspecto que considerar es la posibilidad de que se utilice esta tecnología para fines ilegales o inmorales, como el espionaje o la discriminación. Es importante establecer medidas de control y supervisión para garantizar que este sistema se utilice únicamente con fines legítimos y éticos.

ANEXO III: DICCIONARIO DE SIGLAS, ACRÓNIMOS Y ABREVIATURAS

- ABSA: Análisis de opinión basado en aspectos (*Aspect-Based Sentiment Analysis*).
- AJEMA: Almirante Jefe del Estado Mayor de la Armada
- ANN: Redes neuronales artificiales (*Artificial Neural Networks*).
- ASR: Reconocimiento automático del habla (*Automatic Speech Recognition*).
- CER: Tasa de error de caracteres (*Character Error Rate*).
- CNN: Redes neuronales convolucionales (*Convolutional Neural Networks*).
- CRF: Campos aleatorios condicionales (*Conditional Random Fields*).
- CTC: Conexión de marcos de tiempo continuo (*Connectionist Temporal Classification*).
- CUD: Centro Universitario de la Defensa
- DARPA: Agencia de proyectos de investigación avanzados de defensa (*Defense Advanced Research Projects Agency*).
- ENC: Carta electrónica digital (*Electronical Navigational Chart*).
- ENM: Escuela Naval Militar.
- FBANK: Banco de filtros de Mel frecuencia (*Mel Frequency Filter Bank*).
- GMM: Modelo de mezcla gaussiana (*Gaussian Mixture Model*).
- GRU: Unidad recurrente de compuerta (*Gated Recurrent Unit*).
- HMM: Modelo oculto de Markov (*Hidden Markov Model*).
- IHM: Instituto Hidrográfico de la Marina.
- IBM: Corporación Internacional de Maquinaria y Computación (*International Business Machines Corporation*).
- IOB: Dentro-Fuera-Beginning (*Inside-Outside-Beginning*).
- ISIT: Teoría de la información y estadística (*Information Theory and Statistics*).
- KI: Indexación de palabras (*Keyword Indexing*).
- KWS: Búsqueda de palabras clave (*Keyword Spotting*).
- LAS: *Listen, Attend and Spell*.
- LVCSR: Reconocimiento del habla continuo de vocabulario grande (*Large Vocabulary Continuous Speech Recognition*).
- LSTM: Memoria a largo plazo (*Long Short-Term Memory*).
- MFCC: Coeficientes cepstrales en frecuencia de Mel (*Mel Frequency Cepstral Coefficients*).
- MT: Traducción automática (*Machine Translation*).
- NRL: Laboratorio de investigación naval (*Naval Research Laboratory*).
- OGC: Consorcio de Geoinformación Abierta (*Open Geospatial Consortium*).
- OOV: Palabra fuera del vocabulario (*Out-of-Vocabulary*).

- PANNACOTA: *Processing Audio and Natural language for NAval COmmunications Tasks*.
- PLN: Procesamiento del lenguaje natural (*Natural Language Processing*).
- RNN: Red neuronal recurrente (*Recurrent Neural Network*).
- RNNT: Transductor de red neuronal recurrente (*Recurrent Neural Network Transducer*).
- RI: Indización de recuperación (*Retrieval Indexing*).
- SAT: Adaptación de características de audio (*Speaker Adaptation Techniques*).
- STFT: Transformada de Fourier de tiempo corto (*Short-Time Fourier Transform*).
- VAD: Detección de actividad de voz (*Voice Activity Detection*).
- VHF: Frecuencia muy alta (*Very High Frequency*).
- WER: Tasa de error de palabras (*Word Error Rate*).
- WMS: Servicio de mapas web (*Web Map Service*).

ANEXO IV: PREDICCIONES DETECTOR DE ÓRDENES

Tabla de predicción de órdenes y respuestas con el modelo entrenado CRF por transcripciones mejorado para audio de prueba. Marcado en amarillo las ordenes, en verde las respuestas y en rojo los errores de detección.

uno	tag prediction: O	prueba	tag prediction: O
dos	tag prediction: O	estribor	tag prediction: O
tres	tag prediction: O	uno	tag prediction: O
probando	tag prediction: O	dos	tag prediction: O
uno	tag prediction: O	tres	tag prediction: O
dos	tag prediction: O	probando	tag prediction: O
tres	tag prediction: O	uno	tag prediction: O
probando	tag prediction: O	dos	tag prediction: O
uno	tag prediction: O	tres	tag prediction: O
dos	tag prediction: O	probando	tag prediction: O
tres	tag prediction: O	cuarto	tag prediction: O
probando	tag prediction: O	de	tag prediction: O
uno	tag prediction: O	rota	tag prediction: O
dos	tag prediction: O	prueba	tag prediction: O
tres	tag prediction: O	a	tag prediction: O
probando	tag prediction: O	abor	tag prediction: O
uno	tag prediction: O	uno	tag prediction: O
dos	tag prediction: O	dos	tag prediction: O
tres	tag prediction: O	tres	tag prediction: O
probando	tag prediction: O	probando	tag prediction: O
uno	tag prediction: O	uno	tag prediction: O
dos	tag prediction: O	dos	tag prediction: O
tres	tag prediction: O	tres	tag prediction: O
probando	tag prediction: O	probando	tag prediction: O
estoy	tag prediction: O	uno	tag prediction: O
en	tag prediction: O	dos	tag prediction: O
el	tag prediction: O	tres	tag prediction: O
cuarto	tag prediction: O	probando	tag prediction: O
de	tag prediction: O	alerón	tag prediction: O
rota	tag prediction: O	abor	tag prediction: O
al	tag prediction: O	uno	tag prediction: O
fondo	tag prediction: O	dos	tag prediction: O
estribor	tag prediction: O	tres	tag prediction: O
uno	tag prediction: O	probando	tag prediction: O
dos	tag prediction: O	uno	tag prediction: O
tres	tag prediction: O	dos	tag prediction: O
probando	tag prediction: O	tres	tag prediction: O
uno	tag prediction: O	probando	tag prediction: O
dos	tag prediction: O	alerón	tag prediction: O
tres	tag prediction: O	estribor	tag prediction: O
probando	tag prediction: O	creo	tag prediction: O
cuarto	tag prediction: O	que	tag prediction: O
de	tag prediction: O	grabó	tag prediction: O
rota	tag prediction: O	todo	tag prediction: O

Figura A3. 1: Ejemplo de predicción de órdenes 1. Fuente: propia

espera	tag prediction: O	palada	tag prediction: O
que	tag prediction: O	avante	tag prediction: O
duró	tag prediction: O	estribor	tag prediction: O
más	tag prediction: O	palada	tag prediction: B
de	tag prediction: O	avante	tag prediction: I
un	tag prediction: O	estribor	tag prediction: I
minuto	tag prediction: O	palada	tag prediction: O
al	tag prediction: O	avante	tag prediction: O
final	tag prediction: O	estribor	tag prediction: O
vamos	tag prediction: O	castillo	tag prediction: B
a	tag prediction: O	larga	tag prediction: I
probar	tag prediction: O	todo	tag prediction: I
ahora	tag prediction: O	castillo	tag prediction: O
toldilla	tag prediction: B	larga	tag prediction: O
larga	tag prediction: I	todo	tag prediction: O
todo	tag prediction: I	caña	tag prediction: B
toldilla	tag prediction: O	a	tag prediction: I
larga	tag prediction: O	la	tag prediction: I
todo	tag prediction: O	vía	tag prediction: I
avante	tag prediction: B	caña	tag prediction: O
poca	tag prediction: I	a	tag prediction: O
estribor	tag prediction: I	la	tag prediction: O
avante	tag prediction: B	vía	tag prediction: O
poca	tag prediction: I	atrás	tag prediction: O
estribor	tag prediction: I	poca	tag prediction: O
atrás	tag prediction: B	las	tag prediction: O
media	tag prediction: I	dos	tag prediction: O
abor	tag prediction: I	atrás	tag prediction: B
atrás	tag prediction: O	poca	tag prediction: I
media	tag prediction: O	las	tag prediction: I
abor	tag prediction: O	dos	tag prediction: I
toda	tag prediction: B	quince	tag prediction: B
la	tag prediction: I	grados	tag prediction: I
caña	tag prediction: I	de	tag prediction: I
a	tag prediction: I	caña	tag prediction: I
abor	tag prediction: I	a	tag prediction: I
toda	tag prediction: O	abor	tag prediction: I
la	tag prediction: O	quince	tag prediction: O
caña	tag prediction: O	grados	tag prediction: O
a	tag prediction: O	de	tag prediction: O
abor	tag prediction: O	caña	tag prediction: O
palada	tag prediction: B	a	tag prediction: O
avante	tag prediction: I	abor	tag prediction: O
estribor	tag prediction: I	para	tag prediction: B

Figura A3. 2: Ejemplo de predicción de órdenes 2. Fuente: propia

todo	tag prediction: I	vía	tag prediction: O
para	tag prediction: O	y	tag prediction: O
todo	tag prediction: O	ienes	tag prediction: O
avante	tag prediction: B	que	tag prediction: O
poca	tag prediction: I	tomar	tag prediction: O
las	tag prediction: I	una	tag prediction: O
dos	tag prediction: I	enfilación	tag prediction: O
avante	tag prediction: O	y	tag prediction: O
poca	tag prediction: O	ienes	tag prediction: O
las	tag prediction: O	que	tag prediction: O
dos	tag prediction: O	tomar	tag prediction: O
toda	tag prediction: O	una	tag prediction: O
la	tag prediction: O	enfilación	tag prediction: O
caña	tag prediction: O	regular	tag prediction: O
a	tag prediction: O	diría	tag prediction: O
babor	tag prediction: O	yo	tag prediction: O
toda	tag prediction: O	ya	tag prediction: O
la	tag prediction: O	es	tag prediction: O
caña	tag prediction: O	decir	tag prediction: O
a	tag prediction: O	que	tag prediction: O
babor	tag prediction: O	se	tag prediction: O
toda	tag prediction: B	nota	tag prediction: O
la	tag prediction: I	que	tag prediction: O
caña	tag prediction: I	hay	tag prediction: O
a	tag prediction: I	mucho	tag prediction: O
estribor	tag prediction: I	menos	tag prediction: O
toda	tag prediction: O	ruido	tag prediction: O
la	tag prediction: O	pero	tag prediction: O
caña	tag prediction: O	distorsiona	tag prediction: O
a	tag prediction: O	la	tag prediction: O
estribor	tag prediction: O	voz	tag prediction: O
caña	tag prediction: B	veremos	tag prediction: O
a	tag prediction: I	como	tag prediction: O
la	tag prediction: I	se	tag prediction: O
vía	tag prediction: I	lo	tag prediction: O
caña	tag prediction: O	toma	tag prediction: O
a	tag prediction: O	vosk	tag prediction: O
la	tag prediction: O		

Figura A3. 3: Ejemplo de predicción de órdenes 3. Fuente: propia

ANEXO V: UNIGRAMAS, BIGRAMAS Y TRIGRAMAS

A continuación, se incluyen fragmentos de los archivos de unigramas, bigramas y trigramas utilizados como vocabulario para el módulo Kaldi KWS.

[unk]	barboten	despacio
a	cabeza	diecinueve
cae	cambio	dieciocho
abajo	caña	dieciseis
activa	castillo	diecisiete
activas	catorce	diez
activo	cero	doce
activos	ciaboga	dos
afirmar	cien	doscientas
agua	cientas	doscientos
al	cientos	eje
aleta	cinco	el
alfa	cincuenta	en
aleron	clara	entre
aligerar	claro	escoben
amura	con	esta
ancla	cpa	estamos
anclas	cuarenta	estan
arco	cuatro	este
arrancada	cuarto	estribor
arriba	cuatrocientas	fallo
atras	cuatrocientos	fallos
avante	de	falta
abor	del	faltan
bandera	demora	faltar

Figura A4. 1: Ejemplo de archivo de unigramas. Fuente: propia

a babor	a tres	cinco minutos
a catorce	a uno	cincuenta grados
a cien	a veinte	cincuenta
a cinco	a quedar	minutos
a cincuenta	activos de	cincuenta y
a cuarenta	afirmar y	cuarenta grados
a cuatro	ancla a	cuarenta
a diez	arco alfa	minutos
a doce	atras poca	cuarenta y
a dos	avante como	cuatro grados
a estribor	avante poca	cuatro minutos
a la	babor de	de babor
a noventa	babor para	de bandera
a nueve	babor y	de castillo
a ochenta	babor a	de caña
a ocho	estribor a	de estribor
a once	cambio de	de fondeo
a quince	castillo larga	de guardia
a rumbo	catorce grados	de seguridad
a seis	catorce minutos	del eje
a sesenta	caña a	demora de
a setenta	cien grados	demora guia
a siente	cien minutos	diecinueve
a trece	ciento y	grados
a treinta	cinco grados	diecinueve
		minutos

Figura A4. 2: Ejemplo de archivo de bigramas. Fuente: propia

a babor estribor	atras despacio	avante toda
a babor a	babor	estribor
avante diez	atras despacio	avante toda las
nudos	estribor	babor del eje
avante doce	atras despacio	babor estribor
nudos	las	para
avante ocho	atras poca babor	babor para
nudos	atras poca	quedar
avante seis	estribor	babor y estribor
nudos	atras poca las	cambio de
a quedar rumbo	atras toda babor	bandera
a babor para	atras toda	cambio de
babor a quedar	estribor	rumbo
estribor a	atras toda las	caña a babor
quedar	avante despacio	caña a estribor
a estribor para	babor	caña a la
a la via	avante despacio	castillo larga
afirmar y	estribor	todo
reforzar	avante despacio	castillo y toldilla
agua por babor	las	cuarto de rota
agua por	avante poca	de caña a
estribor	babor	de la canal
al agua por	avante poca	despacio las dos
alfa a babor	estribor	dos tres
alfa a estribor	avante poca las	probando
arriba y clara	avante toda	eje de la
	babor	

Figura A4. 3: Ejemplo de archivo de trigramas. Fuente: propia

ANEXO VI: CÓDIGO DE TRANSCRIPCIÓN WHISPER

```
import os
from os import walk
import pywhisper
from moviepy.editor import *
import codecs
from pydub import AudioSegment
import re
import datetime
from os import walk

def save_subtitles(list_of_subtitles, fichero):
    path_to_save =
    f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/transcripciones/{fichero}"

    path_to_time = f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/tiempos/{'time_'+fichero}"

    with codecs.open(path_to_save, mode="a+", encoding='utf-8') as f:
        for i in list_of_subtitles:
            #f.write(f"ID:{i['id']} TRANSCRIPCION: {i['text']} START:
            {i['start']}\n\n".format(i))
            # TODO EL TEXTO A MINÚSCULAS
            i['text'] = i['text'].lower()

            # REEMPLAZAR TILDES"
            reemplazos = {
                'á': 'a',
                'é': 'e',
                'í': 'i',
                'ó': 'o',
                'ú': 'u',
                'ü': 'u'
            }

            # REEMPLAZAR LAS TILDES Y SIGNOS
            for tilde, sin_tilde in reemplazos.items():
                i['text'] = i['text'].replace(tilde, sin_tilde)

            i['text'] = re.sub(r'[^a-zA-Z\sñÑ0-9]', '', i['text'])
            # f.write(f"{i['text'].lower()}\n".format(i))
            f.write(
                f"{i['text'].lower()}\n".format(
                    i))

    f.close()

    with codecs.open(path_to_time, mode="a+", encoding='utf-8') as x:
        for i in list_of_subtitles:
            x.write(f" {i['start']}\n".format(i))
    x.close()
```

```
def creacion_dataset(texto):
    #TODO EL TEXTO A MINÚSCULAS
    texto = texto.lower()

    #REEMPLAZAR TILDES"
    reemplazos = {
        'á': 'a',
        'é': 'e',
        'í': 'i',
        'ó': 'o',
        'ú': 'u',
        'ü': 'u'
    }

    #REEMPLAZAR LAS TILDES Y SIGNOS
    for tilde, sin_tilde in reemplazos.items():
        texto = texto.replace(tilde, sin_tilde)

    texto = re.sub(r'[^a-zA-Z\sñÑ0-9]', '', texto)

    with codecs.open(f"audios_para_procesado/{carpeta}/Transcripcion_Whisper.txt",
mode="a+", encoding='utf-8') as f:
        f.write(f"{texto} ")

def listDir(p):
    for (path, ficheros, archivos) in walk(p):
        print(ficheros)
        continue
    r = re.compile(r"(\d+-\d+-\d+_\d+-\d+-\d+)")
    archivos.sort(key=lambda x: str(r.search(x).group(1)))
    return archivos

print("INTORDUCE EL NOMBRE DE LA CARPETA DE AUDIOS, RECUERDA QUE CONTENGA LAS
CARPETAS AUDIOS Y TRANSCRIPCIONES")
carpeta= input()
print(f"Tu carpeta es {carpeta}")

audios=
f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/audios/"
lista_ficheros=listDir(audios)
print(lista_ficheros)
path=f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/audios/"
start=0
end=0
id_counter=0
numero=1

for fichero in lista_ficheros:
    # print(fichero)
    final_list_of_text = []
    f = str(fichero).replace(".wav", ".txt")
    path_to_saved_file= path+fichero
    # print(path_to_saved_file)
    audio_clip= AudioFileClip(path_to_saved_file)
    duration = audio_clip.duration
    audio_clip.close()
    from transformers import AutoModel
    from transformers import AutoModelForMaskedLM

    model= pywhisper.load_model("large", device="cpu")
```

```
out = model.transcribe(path_to_saved_file)
print(out)

list_of_text = out["segments"]

for line in list_of_text:
    line ["start"] += start
    line ["end"] += start
    end= line["end"]
    final_list_of_text.append(line)
start += duration

save_subtitles(list_of_subtitles=final_list_of_text, fichero=f)

creacion_dataset(texto=out["text"])

#LIMPIAR TRANSCRIPCIONES

# Ruta de la carpeta con los archivos
carpeta =
'C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/solapall/transcripciones'

# Itera sobre los archivos de la carpeta
for nombre_archivo in os.listdir(carpeta):
    # Verifica que el archivo sea de tipo .txt
    if nombre_archivo.endswith('.txt'):
        ruta_archivo = os.path.join(carpeta, nombre_archivo)
        with open(ruta_archivo, 'r', encoding='utf-8') as archivo:
            # Lee el contenido del archivo
            contenido = archivo.read()
            # Elimina los caracteres especificados y convierte el texto a
mayúsculas
            contenido_limpio = re.sub(r'[.,?]', '', contenido).lower()
            with open(ruta_archivo, 'w') as archivo:
                # Sobrescribe el archivo con el contenido limpio
                archivo.write(contenido_limpio)

print("stop")
```


ANEXO VII: CÓDIGO DE WHISPER MEJORADO Y EVALUACIONES 1 Y 4.

```
def encontrar_whisper(archivo):
    numeros_whisper = []
    archivo2 = "time_" + archivo

    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/transcrip
ciones/{archivo}") as f:
        lineas = f.readlines()

    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/tiempos/{
archivo2}") as f:
        lineas2 = f.readlines()

    for i, linea in enumerate(lineas):

        numeros_whisper.append(float(lineas2[i]))
        # print("ye")
        # print(numeros_whisper)

    return numeros_whisper

import wave

def buscar_similitudes(palabra, lista_palabras):
    for p in lista_palabras:
        distancia = Levenshtein.distance(palabra, p)
        if distancia / max(len(palabra), len(p)) < 0.1:
            return p
    return palabra

def buscar_similitudes2(palabra1, palabra2, lista_bigramas):
    bigramas = [palabra1 + " " + palabra2, palabra2 + " " + palabra1]
    for b in lista_bigramas:
        distancia = Levenshtein.distance(b, bigramas[0])
        if distancia / max(len(b), len(bigramas[0])) < 0.1:
            return b.split()[1]
        distancia = Levenshtein.distance(b, bigramas[1])
        if distancia / max(len(b), len(bigramas[1])) < 0.1:
            return b.split()[0]
    return palabra1, palabra2

def encontrar_por_tiempos(tiempos_hash, archivo, contador, duracion2):
    numeros_redondeados = []
    # Abrir el archivo de números
    archivo2="time_"+archivo
    with
wave.open(f'audios_para_procesado/solapa11/audios/{archivo.replace(".txt",".wav") }
', 'r') as archivo:
        frames = archivo.getnframes()
        rate = archivo.getframerate()
```

```

        duracion = frames / float(rate)

    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/tiempos_v
osk/{archivo2}") as f:
    # Leer cada línea del archivo y convertirlo a un número de coma flotante
    lineas = f.readlines()

    for linea in lineas:
        # Convertir la línea en un número entero
        numero = float(linea.strip())
        if contador!=0:
            numero=numero+duracion2

        # Buscar el resultado inmediatamente inferior en el array de números
        resultado = None
        for n in tiempos_hash:
            if n <= numero:
                resultado = n
            else:
                continue
        if resultado==None:
            resultado=n
        numeros_redondeados.append(resultado)
    return numeros_redondeados,duracion

duracion=0
contador=0
total=""
wer_medio=0
numero_archivos=0
hash = {}
for archivos in transcripciones:
    reference_all= ""
    hypothesis_all= ""

    with open(transcripciones_carpeta + archivos, "r") as y:
        for g, linea in enumerate(y):
            hash[g] = linea.strip()

    with open(tiempos_carpeta + "time_" + archivos, 'r') as archivo:
        tiempos_hash = [float(linea.strip()) for linea in archivo.readlines()]

    if contador==0:
        duracion2=0

    #CON LAS TRANSCRIPCIONES DE VOSK OBTENGO LOS TIEMPOS PARA CADA FRASE DE
WHISPER
    numeros_redondeados,duracion2=
encontrar_por_tiempos(tiempos_hash,archivos,contador,duracion)
    contador = contador + 1
    duracion=duracion+duracion2

    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/vosk/{arc
hivos}") as f:
    # Leer todas las líneas del archivo y almacenarlas en un array
    vosk = f.readlines()
    array_tiempos_ordenes_whisper = []

```

```
array_tiempos_ordenes_vosk = []

with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/transcrip
ciones/{archivos}") as f:
    whisper=f.readlines()

    for i, elemento in enumerate(tiempos_hash):
        print(elemento)
        posicion = tiempos_hash.index(elemento)
        if elemento in numeros_redondeados:
            posicion2=numeros_redondeados.index( (elemento))
            # print(posicion)
            print(hash[posicion])
            print("-----")
            print(vosk[posicion2])
            # lista de palabras de cada texto
            palabras1 = hash[posicion].split()
            # print(palabras1)
            palabras2 = vosk[posicion2].split()
            # print(palabras2)

            for x in range(len(palabras1)):
                # verifica si la distancia de Levenshtein es menor al 50%
                for palabra2 in palabras2:
                    if Levenshtein.distance(palabras1[x], palabra2) /
max(len(palabras1[x]), len(palabra2)) < 0.5:
                        palabras1[x] = palabra2 # sustituye la palabra del primer
texto por la del segundo

                        for palabra2 in palabras2:
                            # sale del bucle interno si se encuentra una palabra adecuada
                            if Levenshtein.distance(palabras1[x], palabra2) /
max(len(palabras1[x]),len(palabra2)) > 0.8 and not palabra2 in palabras1:
                                palabras1.append(palabra2) # sustituye la palabra del
primer texto por la del segundo

                                # crea el texto completo utilizando la lista de palabras actualizada
                                reference = " ".join(palabras1)
                                reference=reference.replace("<", "").replace(">", "")
with
open("audios_para_procesado/solapall/whisper_mejorado/whisper_mejorado_frases.txt"
, "a+", encoding="utf-8") as whisper2:

        whisper2.write(reference + "\n")

reference_all = reference_all + " " + reference
array_tiempos_ordenes_vosk.append(elemento)

hypothesis = whisper[posicion]
hypothesis_all = hypothesis_all + " " + hypothesis
continue

if elemento not in numeros_redondeados:
    palabras1 = whisper[posicion].split()
    reference = " ".join(palabras1)
    reference=reference.replace("<", "").replace(">", "").lower()
```

```

with
open("audios_para_procesado/solapall/whisper_mejorado/whisper_mejorado_frases.txt"
, "a+",
    encoding="utf-8") as whisper2:
    # IMPLEMENTACION BUSQUEDA DE PALABRAS DICCIONARIO.
    with open("words.txt") as f:
        lista_palabras = f.read().splitlines()
    with open("bigrams.txt") as f:
        lista_bigramas = f.read().splitlines()
    with open('trigrams.txt', 'r', encoding="utf-8") as f:
        trigramas = [line.rstrip('\n') for line in f]

    palabras = reference.split()

    # Recorre las palabras del texto en grupos de tres
    r = 0
    while r < len(palabras) - 2:
        trigram = ' '.join(palabras[r:r + 3])

        # Busca similitudes con los trigramas del archivo
        for t in trigramas:
            if Levenshtein.distance(trigram, t) / max(len(trigram), len(t)) <
0.2:
                palabras[r:r + 3] = t.split()
                break

        r += 3

    # Une las palabras del texto con espacios
    reference = ' '.join(palabras)
    reference= reference.replace("á","a").replace("í","i")

    whisper2.write(reference + "\n")
with
open("audios_para_procesado/solapall/whisper_mejorado/whisper_mejorado_tiempos.txt"
, "a+",
    encoding="ISO-8859-1") as whisper3:
    whisper3.write(f"{elemento}" + "\n")
    total = total + " " + reference

    print("oooooooooooooooooooooooooooooooooooooooo")

    # reference = hash[posicion].replace("<","").replace(">","")
    reference_all = reference_all + " " + reference
    array_tiempos_ordenes_vosk.append(elemento)

    hypothesis = whisper[posicion]
    hypothesis_all = hypothesis_all + " " + hypothesis

    print(reference)
    continue

    contador_encontradas=0

    array_tiempos_ordenes_whisper= encontrar_whisper(archivos)

    whisper_frases=""
    reference_all_2=""

```

```
for i, elemento in enumerate(array_tiempos_ordenes_whisper):
    posicion = tiempos_hash.index(elemento)
    whisper_frases= hash[posicion]
    reference_all_2 = reference_all_2 + " " + whisper_frases
    # print(reference_all_2)

for elemento in array_tiempos_ordenes_vosk:
    if elemento in array_tiempos_ordenes_whisper:
        contador_encontradas = contador_encontradas+1

hypothesis_all = hypothesis_all.replace('\n', '')
wer_result = wer(reference_all, hypothesis_all)
wer_result_2 = wer(reference_all_2, hypothesis_all)
substitution, deletion, insertion = wer2(reference_all, hypothesis_all)
wer_medio = wer_medio + wer_result
numero_archivos = numero_archivos + 1

with
open("audios_para_procesado/solapall/whisper_mejorado/whisper_mejorado.txt","a+",e
ncoding="ISO-8859-1") as whisper:

total=total.replace("<","").replace(">","").replace("\n","").replace("[unk1]","").
replace("[unk1b]","").replace("[unk3]","")
    whisper.write(total)

whispermolineas2= []
for archivos in
os.listdir(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/tra
nscipciones/"):
    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/transcrip
ciones/{archivos}") as f:
        whispermolineas=f.readlines()
        for lineas in whispermolineas:
            whispermolineas2.append(lineas)
            whispermolineas3 = " ".join(whispermolineas2)
            whispermolineas3 = whispermolineas3.replace("<", "").replace(">",
"").replace("\n", "")

whisperlineas2=[]
for archivos in
os.listdir(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/whi
sper/"):
    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/whisper/{
archivos}","r",encoding="ISO-8859-1") as f:
        whisperlineas=f.readlines()
        for lineas in whisperlineas:
            whisperlineas2.append(lineas)
            whisperlineas3 = " ".join(whisperlineas2)
            whisperlineas3 = whisperlineas3.replace("<", "").replace(">",
"").replace("\n", "").replace("Ã±","ñ")

wer_result_3 = wer(whisperlineas3, total)
wer_result_4 = wer(whispermolineas3, whisperlineas3)
substitution2, deletion2, insertion2 = wer2(whisperlineas3, total)
```

```

with open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/" +
"evaluacion_WER_B1_mejorado.txt",
        "a+") as file3:
    file3.write(f"\nÓrdenes Whisper mejorado: {total}\n\n\nÓrdenes Whisper:
{whisperlineas3}\n\n")
    file3.write("Word Error Rate: {:.4f}%\n".format(wer_result_3 * 100))
    file3.write(f"Substituciones: {substitution2}    Eliminaciones: {deletion2}
Inserciones: {insertion2}\n\n")

with open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/" +
"evaluacion_WER_B1_nomejorado.txt",
        "a+") as file3:
    file3.write(f"\nTranscripción Whisper: {whispernolineas3}\n\n\nRevisado
Whisper: {whisperlineas3}\n\n")
    file3.write("Word Error Rate: {:.4f}%\n".format(wer_result_4 * 100))
    file3.write(f"Substituciones: {substitution2}    Eliminaciones: {deletion2}
Inserciones: {insertion2}\n\n")

print("WER 1 y 4 REALIZADOS CON EXITO!")

```

ANEXO VIII: CÓDIGO DE EVALUACIÓN 3 PARA VOSK

```
import codecs
from pydub import AudioSegment
import re
import datetime
from os import walk
#WER B1 - ORDENES DE VOSK Y CORRESPONDIENTES FRASES REVISADAS

def listDir(p):
    for (path, ficheros, archivos) in walk(p):
        # print(ficheros)
        continue
    r = re.compile(r"(\d+-\d+-\d+ \d+-\d+-\d+)")
    archivos.sort(key=lambda x: str(r.search(x).group(1)))
    return archivos

print("INTORDUCE EL NOMBRE DE LA CARPETA DE AUDIOS, RECUERDA QUE CONTENGA LAS  
CARPETAS AUDIOS Y TRANSCRIPCIONES")
carpeta= input()
print(f"Tu carpeta es {carpeta}")

#CREACION HASH Y ARRAY TIEMPOS
import Levenshtein
transcripciones_carpeta =
f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/whisper/"
transcripciones = listDir(transcripciones_carpeta)
tiempos_carpeta =
f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/tiempos/"
tiempos= listDir(tiempos_carpeta)

def wer(reference, hypothesis):
    """
    Calculates the Word Error Rate (WER) between two text transcripts.
    """
    reference = reference.replace('<', ' ').replace('>', ' ')
    hypothesis = hypothesis.replace('<', ' ').replace('>', ' ')
    reference = reference.split()
    hypothesis = hypothesis.split()
    d = Levenshtein.distance(reference, hypothesis)
    if reference==[]:
        wer= 0
        print("ERROR, LAS ORDENES DE VOSK NO SE ENCUENTRAN EN WHISPER")
        return wer
    else:
        wer = float(d) / len(reference)
        return wer

def wer2(original, resultado):
    original=original.replace('<', ' ').replace('>', ' ')
    resultado=resultado.replace('<', ' ').replace('>', ' ')
    original_words = original.split()
    resultado_words = resultado.split()
    substitution = 0
    deletion = 0
    insertion = 0
    pointer_original = 0
    pointer_resultado = 0
    while pointer_original < len(original_words) and pointer_resultado <
```

```
len(resultado_words):
    if original_words[pointer_original] == resultado_words[pointer_resultado]:
        pointer_original += 1
        pointer_resultado += 1
    else:
        substitution += 1
        pointer_resultado += 1
while pointer_original < len(original_words):
    deletion += 1
    pointer_original += 1
while pointer_resultado < len(resultado_words):
    insertion += 1
    pointer_resultado += 1
return substitution, deletion, insertion

def encontrar_whisper(archivo):
    numeros_whisper = []
    archivo2 = "time_" + archivo

    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/whisper/{
archivo}") as f:
    lineas = f.readlines()

    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/tiempos/{
archivo2}") as f:
    lineas2 = f.readlines()

    for i, linea in enumerate(lineas):
        if "<" in linea and i < len(lineas2):
            numeros_whisper.append(float(lineas2[i]))
            # print("ye")
            # print(numeros_whisper)

    return numeros_whisper

import wave

def encontrar_por_tiempos(tiempos_hash, archivo, contador, duracion2):
    numeros_redondeados = []
    # Abrir el archivo de números
    archivo2 = "time_" + archivo
    with
wave.open(f'audios_para_procesado/solapall/audios/{archivo.replace(".txt", ".wav")}'
, 'r') as archivo:
        frames = archivo.getnframes()
        rate = archivo.getframerate()
        duracion = frames / float(rate)

    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/tiempos_v
osk/{archivo2}") as f:
        # Leer cada línea del archivo y convertirlo a un número de coma flotante
        # numeros = [float(linea.strip()) for linea in f]
        lineas = f.readlines()

    for linea in lineas:
```

```
# Convertir la línea en un número entero
numero = float(linea.strip())
if contador!=0:
    numero=numero+duracion2

# Buscar el resultado inmediatamente inferior en el array de números
resultado = None
for n in tiempos_hash:
    if n <= numero:
        resultado = n
    else:
        continue
if resultado==None:
    resultado=n
numeros_redondeados.append(resultado)
# print(numeros_redondeados)
return numeros_redondeados,duracion

duracion=0
contador=0
reference_all= ""
hypothesis_all= ""
reference_all_2 = ""
contador_encontradas=0
array_tiempos_ordenes_whisper = []
array_tiempos_ordenes_vosk = []
for archivos in transcripciones:

    # print(archivos)
    hash = {}
    with open(transcripciones_carpeta + archivos, "r") as y:
        for g, linea in enumerate(y):
            hash[g] = linea.strip()

    with open(tiempos_carpeta + "time_" + archivos, 'r') as archivo:
        tiempos_hash = [float(linea.strip()) for linea in archivo.readlines()]
        # print(tiempos_hash)
        # print(tiempos_hash)

    if contador==0:
        duracion2=0

    #CON LAS TRANSCRIPCIONES DE VOSK OBTENGO LOS TIEMPOS PARA CADA FRASE DE
    WHISPERsol
    numeros_redondeados,duracion2=
    encontrar_por_tiempos(tiempos_hash,archivos,contador,duracion)
    # print(numeros_redondeados)
    contador = contador + 1
    duracion=duracion+duracion2
    #print(numeros_redondeados)

    with
open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/vosk/{arc
hivos}") as f:
    # Leer todas las líneas del archivo y almacenarlas en un array
    vosk = f.readlines()

    for i, elemento in enumerate(numeros_redondeados):
```

```
if elemento in tiempos_hash:
    posicion = tiempos_hash.index(elemento)
    # print(posicion)
    if "<" in hash[posicion]:
        print(hash[posicion])
        print("-----")
        print(vosk[i])
        # lista de palabras de cada texto
        palabras1 = hash[posicion].split()
        # print(palabras1)
        palabras2 = vosk[i].split()
        # print(palabras2)

        # recorre todas las palabras del primer texto
        for x in range(len(palabras1)):
            # verifica si la distancia de Levenshtein es menor al 50%
            for palabra2 in palabras2:
                if Levenshtein.distance(palabras1[x], palabra2) /
max(len(palabras1[x]), len(palabra2)) < 0.5:
                    palabras1[x] = palabra2 # sustituye la palabra del
primer texto por la del segundo
                    break # sale del bucle interno si se encuentra una
palabra adecuada

                for palabra2 in palabras2:
                    if Levenshtein.distance(palabras1[x], palabra2) /
max(len(palabras1[x]), len(palabra2)) > 0.75:
                        palabras1.append(palabra2) # agrega la palabra al
final del texto completo
                        break # sale d

        # crea el texto completo utilizando la lista de palabras
actualizada
        reference = " ".join(palabras1)
        print(reference)

        # reference = hash[posicion].replace("<", "").replace(">", "")
        reference_all = reference_all + " " + reference
        array_tiempos_ordenes_vosk.append(elemento)
        hypothesis = vosk[i]
        hypothesis_all= hypothesis_all+" " + hypothesis

array_tiempos_ordenes_whisper1= encontrar_whisper(archivos)
print(array_tiempos_ordenes_whisper1)
for palabras in array_tiempos_ordenes_whisper1:
    array_tiempos_ordenes_whisper.append(palabras)

print(array_tiempos_ordenes_whisper)

whisper_frases=""

for i, elemento in enumerate(array_tiempos_ordenes_whisper1):
    posicion = tiempos_hash.index(elemento)
    whisper_frases= hash[posicion]
    reference_all_2 = reference_all_2 + " " + whisper_frases
    # print(reference_all_2)
```

```
for elemento in array_tiempos_ordenes_vosk:
    if elemento in array_tiempos_ordenes_whisper1:
        contador_encontradas = contador_encontradas+1

hypothesis_all = hypothesis_all.replace('\n', '')

print(reference_all)
print(hypothesis_all)
wer_result = wer(reference_all, hypothesis_all)

wer_result_2 = wer(reference_all_2, hypothesis_all)
substitution, deletion, insertion = wer2(reference_all, hypothesis_all)

with open(f"C:/Users/alvar/Desktop/WhisperTFG/audios_para_procesado/{carpeta}/" +
"evaluacion_WER_3.txt",
"a+") as file3:
    file3.write(f"Archivo: {archivos}\nÓrdenes Vosk: {hypothesis_all}\nÓrdenes
Whisper: {reference_all_2}\n\n")
    file3.write("Word Error Rate sin contar órdenes no encontradas:
{:.4f}%\n".format(wer_result * 100))
    file3.write(f"Substituciones: {substitution}   Eliminaciones: {deletion}
Inserciones: {insertion}\n\n")
    file3.write(
        f"Órdenes encontradas Vosk / Órdenes en Whisper = {contador_encontradas}
/ {len(array_tiempos_ordenes_whisper)}\n")
    file3.write("Word Error Rate contando órdenes no encontradas:
{:.4f}%\n\n\n\n".format(wer_result_2 * 100))

print("WER 3 REALIZADO CON EXITO!")
```

ANEXO IX: CÓDIGO DE PREPARACIÓN PARA SIMDIS

```

import codecs
import os

from pydub import AudioSegment
import re
import datetime
from os import walk
import re
import datetime

def listDir(p):
    for (path, ficheros, archivos) in walk(p):
        # print(ficheros)
        continue
    r = re.compile(r"(\d+-\d+-\d+_\d+-\d+-\d+)")
    archivos.sort(key=lambda x: str(r.search(x).group(1)))
    return archivos

archivos= listDir("audios_para_procesado/solapall/audios/")
nombre= archivos[0]
fecha_original= nombre.replace(".wav","")
fecha_original=fecha_original.replace("-bt","")
print(fecha_original)
# Formato de fecha original
formato_original = "%Y-%m-%d_%H-%M-%S"

# Convertir a objeto datetime
fecha_objeto = datetime.datetime.strptime(fecha_original, formato_original)

# Formato deseado
formato_deseado = "%Y-%m-%d-%H:%M:%S"

# Convertir a texto de fecha con el formato deseado
fecha_deseada = fecha_objeto.strftime(formato_deseado)

with
open("audios_para_procesado/solapall/whisper_mejorado/whisper_mejorado_tiempos.txt", "r") as archivo_numeros,
open("audios_para_procesado/solapall/whisper_mejorado/whisper_mejorado_frases.txt", "r") as archivo_nombres,
open("audios_para_procesado/solapall/whisper_mejorado/SIMDIS.txt", "w") as
archivo_resultados:
    lista_numeros = archivo_numeros.readlines()
    lista_nombres = archivo_nombres.readlines()

    archivo_resultados.write("DATE\tTIME START\tSENTENCE\n")

    for i in range(len(lista_numeros)):
        linea_resultado =
("{}\t{}\t{}".format(fecha_deseada, lista_numeros[i].strip(),
lista_nombres[i].strip())
        archivo_resultados.write(linea_resultado + "\n")

```