

# Universidad de Alcalá

## Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación.



### Trabajo Fin de Grado

Implementación de comunicaciones mediante voz  
en un robot social.

ESCUELA POLITECNICA

**Autor:** Alberto Palomo Alonso.

**Tutor/es:** Saturnino Maldonado Bascón.

2020







# Índice de contenido.

1. Abstract.....	9
2. Introducción.....	10
3. Palabras clave. ....	11
4. Glosario.....	12
5. Estudio de modelos. ....	15
5.1. Modelos online.....	15
5.2. Modelos offline (Neural Network Models).....	17
5.3. Modelos offline (Hidden Markov Models).....	19
6. Objetivos y modelo seleccionado. ....	21
6.1. Objetivos del proyecto.....	21
6.2. Modelo seleccionado.....	22
7. Descripción del sistema.....	24
7.1. Descripción genérica.....	24
7.2. Descripción hardware.....	25
7.3. Sistema operativo. ....	27
7.4. Dependencias.....	27
7.5. SphinxBase.....	28
7.6. PocketSphinx.....	28
7.7. Librería de Python.....	28
7.8. ALSA.....	28
7.9. Script estático.....	29
7.10. Scripts modificables, parámetros y diccionarios.....	29
8. Instalación del modelo.....	30
8.1. Instalación del sistema operativo. ....	30
8.2. Instalación vía software. (Online).....	35
8.3. Instalación manual. (Offline).....	38
8.4. Detección y funcionamiento de dispositivos de entrada de audio en ALSA y PocketSphinx.....	41
9. Sistema de archivos.....	44
10. Scripting.....	46
10.1. Script estático.....	46
10.2. Manejador de señales.....	47
10.3. Mapa de correcciones.....	48

10.4.	Diccionario.....	49
10.5.	Parámetros del programa. ....	50
11.	Manual de usuario.....	51
11.1.	Puesta en marcha del programa. ....	52
11.2.	Ejecución de DEMO. ....	53
12.	Manual del programador. ....	54
12.1.	Manejador de señales en Raspberry.....	54
12.2.	Manejador de señales vía comandos.....	55
12.3.	Corrección de errores.....	56
12.4.	Diccionario personalizado.....	56
13.	Condiciones de funcionamiento. ....	58
13.1.	Condiciones de distancia y ruido. ....	59
13.2.	Pruebas de velocidad.....	62
14.	Conclusión personal del proyecto. ....	62
15.	Referencias.....	64
16.	Anexo I. Scripts.....	67
16.1.	Instalador del programa en bash.....	67
16.2.	Sampler.py (Script estático). ....	68
16.3.	FSMSignal.py (Manejador de señales / DEMO). ....	72
16.4.	CMap.py (Mapa de correcciones / DEMO).....	73
16.5.	DICT.txt (Diccionario / DEMO).....	74
16.6.	PARAMETERS.txt (Parámetros).....	74
17.	Anexo II. Árbol.....	75
18.	Anexo III. Links.....	75
19.	Anexo IV. Matlab publish.....	76
	Pruebas de distancia. ....	76
	Pruebas de ruido.....	78
	Pruebas de corrección. ....	80
20.	Anexo V. Manual del programa. ....	82







# Índice de figuras.

1. Figura 5.1.1.- Esquema de funcionamiento de un modelo online.	[16]
2. Figura 5.2.1.- Esquema de conexiones de un NNM.	[18]
3. Figura 5.3.1.- Esquema HMM.	[20]
4. Figura 6.2.1.- Componentes de CMU Sphinx.	[22]
5. Figura 6.2.2.- Diagrama de funcionamiento CMU Sphinx.	[23]
6. Figura 7.1.1.- Esquema de conexión de todos los sistemas involucrados.	[25]
7. Figura 7.2.1.- Micrófono USB.	[25]
8. Figura 7.2.2.- Raspberry Pi 4.	[26]
9. Figura 7.2.3.- Motor 12V.	[26]
10. Figura 7.2.4.- Tarjeta microSD.	[26]
11. Figura 8.1.1.- Archivos descargados.	[31]
12. Figura 8.1.2.- Imagen del sistema operativo en la carpeta.	[31]
13. Figura 8.1.3.- Wizard para Raspberry Pi Imager.	[31]
14. Figura 8.1.4.- Finalización de instalación del Wizard.	[32]
15. Figura 8.1.5.- Raspberry Pi Imager.	[33]
16. Figura 8.1.6.- Selección de la imagen a instalar.	[33]
17. Figura 8.1.7.- Ruta config.txt.	[34]
18. Figura 8.1.8.- Línea a modificar en config.txt.	[35]
19. Figura 8.2.1.- Keryx.	[38]
20. Figura 8.4.1.- Null dispositivo prefijado en ALSA.	[42]
21. Figura 8.4.2.- Default PulseAudio en ALSA.	[42]
22. Figura 8.4.3.- PARAMETERS.txt.	[43]
23. Figura 9.1.- Esquema de ficheros del proyecto.	[44]
24. Figura 9.2.- Ficheros de Python.	[45]
25. Figura 11.2.- Esquema usuario-programa.	[54]
26. Figura 12.1.- Raspberry Pi (GPIO).	[55]
27. Figura 13.1.1.- Fallos en función de la distancia.	[59]
28. Figura 13.1.2.- Fallos en función del nivel de ruido.	[60]
29. Figura 13.1.3.- Comparación de fallos con CMAP.	[61]
30. Tabla 13.- Especificaciones de pruebas.	[58]
31. Tabla 13.2.- Tiempo de procesado y carga del programa.	[62]



## 1. Abstract.

Since 20th century, speech recognition methods have been under research. It was not until 1987 when practical models were discovered, introducing n-gram language models. Not only the language models are important in this type of feature, but the algorithms are an essential part of the whole model. Machine learning discipline, which took a huge development in the 90s, are the most applied algorithms which combine with n-gram models in order to create a speech recognition model.

Since then, many models were developed; notwithstanding, the major part of that models are online-based models, which use heavy-equipped servers that perform this machine learning methods in great timestamps. In this project, an offline model is implemented in a social robot designed to help people with any kind of disabilities, so the major part of the models can't be implemented in this case, as the robot can reach areas where the internet is not available.

CMU Sphinx is the solution adopted in order to reach perform a precise and real time voice commands, which implements an offline machine learning model based on Hidden Markov Model and a Spanish language model. The robot will be available to recognize basic commands and act depending on the user needs.

## 2. Introducción.

Una de las tendencias de estos últimos años en empresas como Google, Amazon y otros gigantes de la tecnología, que enfocan sus productos al público ordinario, es el reconocimiento de palabras mediante voz con objetivo de que una máquina, generalmente nuestros dispositivos móviles, sean capaces de entender lo que estamos diciendo y poder actuar con un simple comando de voz; sin necesitar que el usuario realice alguna acción directa sobre el dispositivo.

Esto por norma general, suele ser una comodidad más que una necesidad; que más allá de proporcionar al usuario una sensación de completitud tecnológica, puede llegar a ahorrar tiempo y en algunos casos permitir su uso como es el caso de los vehículos y manos libres.

Además, este tipo de características pueden ser muy útiles en distintos oficios donde se utilizan dispositivos que no tienen entradas o salidas fáciles de instalar y recurren a este tipo de características para la comunicación del usuario con la máquina.

Sin embargo, estos avances permiten a personas con discapacidad o dependencia poder utilizar los dispositivos que sin esta característica no podrían utilizar, al menos de forma flexible y fluida. En este proyecto, enfocado más a este tipo de personas con dependencia, abordaremos una solución tecnológica en la cual el usuario podrá dar órdenes mediante voz a un robot social destinado a asistir a este tipo de personas.

Los primeros intentos de reconocimiento de audio vienen de 1952, donde tres investigadores (Stephen Balasek, R. Biddulph, K. H. Davis) [2] construyeron un sistema de reconocimiento de dígitos mediante el espectro de audio. Además, empresas tecnológicas como IBM ya estaban haciendo sus investigaciones en esas fechas, dado que en 1962 se presentó en el *World's Fair* un modelo capaz de reconocer hasta 16 palabras. Sin embargo, no es hasta finales de los 80 que se consiguen diseñar modelos prácticos basados en probabilidades condicionadas de n-gramas (Katz's back-off model). A día de hoy, se utilizan modelos de inteligencia artificial, conocido como Machine Learning, que permiten una eficiencia mucho mayor que los modelos tradicionales basados en probabilidades condicionadas y análisis espectral del audio.

Los modelos más empleados son los Modelos Ocultos de Markov (HMMs) y Redes Neuronales Atrificales (NNMs). Estas últimas no sólo se utilizan para reconocimiento de voz, si no para un muy amplio abanico de disciplinas basadas en inteligencia artificial. Por ejemplo, DeepMind (equipo de Google) lanzó un modelo de inteligencia artificial llamado AlphaZero en 2017 que logró un nivel de juego sobrehumano en ajedrez en 24 horas, demostrando que estos modelos acontecen un desarrollo tecnológico exagerado. [3]

Este proyecto se enfoca más a convertir el audio a señales de manejo del robot (movimiento, por ejemplo) más que al manejo de estas mismas señales, dicha elección se deja a necesidad del usuario. Por tanto, el enfoque es usuario-sistema dado que no entraré a definir un manejo de las acciones (señales usuario-acción). Como, por ejemplo, el movimiento del robot tras recibir una orden de desplazamiento, debido a que depende mucho del uso que se le vaya a dar al robot en el que implementemos esta característica. El proyecto se queda en definir la señal usuario-acción y no en la interpretación, uso y manejo de esta. De todas formas, en el proyecto se implementa de forma simple un ejemplo de manejo de las señales usuario-acción en modo de una DEMO, tanto por hardware como por software y se proporciona al programador archivos aislados y bien definidos donde configurar las señales y parámetros que desee de forma muy sencilla.

Cabe destacar que las señales de las que se habla en esta introducción son señales producidas por un programa de reconocimiento de audio hacia un programa destino o manejador de dispositivo con el fin de realizar cierta acción, estas señales serán, en función de cómo las defina el usuario, señales del sistema operativo o acciones directas en un dispositivo.

### 3. Palabras clave.

Machine Learning.

Python.

Speech recognition.

Linux.

Hidden Markov Models (HMMs).

Raspberry.

## 4. Glosario.

A.

ALSA: Advanced Linux Sound Architecture. Sistema de sonido del sistema operativo Linux.

Amazon: Empresa tecnológica.

B.

Bash: Lenguaje de órdenes para un sistema operativo UNIX.

C.

CMU Sphinx: Término general para describir un conjunto de sistemas de reconocimiento de voz en particular, desarrollado por la Universidad de Carnegie Mellon.

5G: Estándar de comunicaciones móviles. Generalmente para conexión a internet.

Controlador: Véase 'driver'.

D.

Debian: Sistema operativo y distribución de software.

Driver: Programa que se encarga de gestionar un periférico.

E.

Ethernet: Estándar de conexión en redes locales alámbricas.

G.

Google: Empresa tecnológica.

GPIO: Puerto de propósito general (entrada o salida).

GPU: Unidad de procesamiento gráfico.

H.

Hardware: Conjunto de elementos físicos o materiales que componen un sistema.

Hash Map: Mapa de búsqueda, popular en programación.

HDMI: Puerto de entrada/salida de audio y vídeo en HD.

Hidden Markov Models: Modelo de Machine Learning basado en estadística.

I.

IBM: Empresa tecnológica.

J.

K.

Kernel: Núcleo del sistema operativo.

L.

Linux: Sistema operativo para máquinas y ordenadores, de libre uso.

M.

Machine Learning: Disciplina mediante la cual se le enseña a una máquina a realizar una determinada tarea.

Matlab: Programa matemático.

MicroSD: Dispositivo de almacenamiento en una tarjeta de reducido tamaño.

P.

PocketSphinx: Programa que implementa un HMM para el reconocimiento de audio.

Python: Lenguaje de programación famoso por su fácil uso, cantidad de librerías y flexibilidad.

R.

Raspberry: Ordenador de bajo presupuesto y pequeño tamaño que implementa las funcionalidades básicas del mismo.

Redes neuronales artificiales: Disciplina de Machine Learning mediante un modelo que se 'parece' a las neuronas de un cerebro.

Relé: Interruptor controlado por un dispositivo eléctrico.

S.

Script: Programa escrito directamente sobre un lenguaje de programación.

Sistema embebido o sistema empotrado: Sistema de computación enfocado a realizar una o pocas funciones dedicadas.

Software: Conjunto de elementos informáticos o programas que componen un sistema.

Speech Recognition: Disciplina mediante la cual se le enseña a una máquina a reconocer palabras y/o frases mediante audio.

SphinxBase: Paquete de programas informáticos enfocados al reconocimiento de audio.

T.

Transistor: Componente electrónico usado para controlar señales (potencia o dirección).

U.

UNIX: Sistema operativo.

W.

WiFi:

Protocolo de conexión inalámbrica en áreas locales.

Windows 10:

Sistema operativo.

Wizard:

Asistente software para la instalación de un programa.



## 5. Estudio de modelos.

Para implementar la característica de reconocimiento de comandos en el robot social, se disponen de varios modelos posibles de implementar en un robot genérico que tenga labor de comunicación social o asistencia a personas discapacitadas. Los modelos más eficientes son los modelos online, mientras que los modelos offline actualmente abarcan mucho cómputo en los procesadores del robot y limitan ligeramente la cantidad de programas que pueden correr en paralelo.

A continuación, se estudian algunos modelos que se pueden implementar en el robot añadiendo sus pros y sus contras, para posteriormente elegir un modelo capaz de cubrir las necesidades en un robot social.

### 5.1. Modelos online.

El modelo más eficiente es el modelo online, dado que este tipo de modelos recurren a servidores externos que están altamente equipados para correr modelos de Machine Learning, que suelen incluir un tipo de GPU muy cara que sería inviable implementar en un robot social. Es por esto por lo que este tipo de modelos son muy eficientes, pero tienen la desventaja de obligar al robot comunicarse vía a internet al servidor dedicado a este tipo de característica; cosa que no siempre es posible o viable, dado que los robots con capacidad de desplazarse pueden salirse del área de cobertura de la red u otros factores.

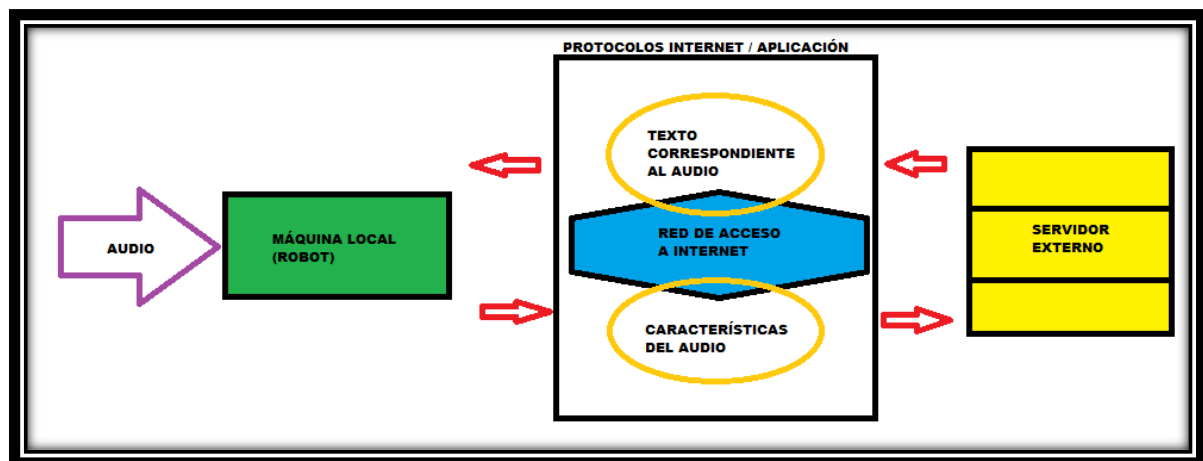
Estos modelos online consisten en el intercambio de datos entre el servidor y el usuario, proporcionando el usuario las características del audio o el mismo audio, y recibiendo a cambio el resultado en modo de texto. Estos modelos, en principio, son gratuitos, pero no comercializables, dado que se recurre a un servidor externo que no pertenece al usuario y pertenece a una empresa o entidad externa.

Uno de los modelos online consiste en la llamada a una función del módulo ‘SpeechRecognition’ de Python [1]. La función de reconocimiento por defecto de esta librería conecta con un servidor de Google para procesar el audio y sacar el texto correspondiente. Es muy eficiente en la lengua inglesa, aunque también funciona para la lengua española o Castellano. Si se desea especificar una función de esta librería la función sería la siguiente:

```
recognizer_instance.recognize_google_cloud()
```

NOTA: Cabe destacar que esta función es de la librería SpeechRecognition, y que es llamada mediante un script en Python.

Este módulo de Python es capaz de correr modelos offline, pero que deben de ser instalados o modelos creados por el usuario. Crear un modelo personalizado es un trabajo muy grande, dado que hay que disponer de una base de datos enorme de audios (en Castellano o en inglés) y conocimientos de entrenamiento de Machine Learning, por lo que crear un modelo personalizado con este módulo resultaría en un modelo no tan eficiente; además que esto al ser un modelo que se ejecuta en la máquina local y no en una remota, ocupa la capacidad de cómputo del robot.



*Figura 5.1.1.- Esquema de funcionamiento de un modelo online.*

## 5.2. Modelos offline (Neural Network Models)

En cuanto a los modelos offline se refiere, como se ha mencionado en el capítulo anterior, tienen la ventaja de que no necesitamos tener una conexión a internet constante para ejecutar el programa que permite implementar la característica de reconocimiento de audio, pero tiene la desventaja que todo el procesamiento del reconocimiento se ejecuta en la máquina local, por lo que puede limitar la capacidad de cómputo del robot si se ejecutan más programas en paralelo, que es lo más común en robot social.

Uno de los modelos offline más eficientes que existen son los modelos de Redes Neuronales Artificiales, que es un tipo de modelo que utiliza teoría de Machine Learning para asociar las características del audio a texto.

Este tipo de modelos se les llama Machine Learning (aprendizaje de la máquina) porque la máquina es capaz de aprender por sí misma. Uno de los modelos es el modelo de Redes Neuronales Artificiales; se les denomina así por su estructura en forma de neuronas, aunque las sinapsis neuronales tienen muchas más conexiones que los que se implementan en máquinas, la estructura es parecida.

Coloquialmente, podemos definirla como un pastel por capas, cada 'neurona' tiene una función asociada (función de activación), que transforma los valores y, en algunos casos, impone un límite máximo y mínimo de los valores numéricos que les atraviesan. Las neuronales están conectadas por enlaces, los enlaces contienen un valor numérico por el cual se multiplica el valor que les atraviesa, estos enlaces conectan dos neuronas, se extrae el valor de una neurona y mediante el enlace, el valor es desplazado de una neurona a otra como muestra la figura 5.2.1. Los valores van fluyendo por las neuronas mediante la *función de red*<sup>1</sup> hasta la última capa, donde se decide el resultado en la capa de salida. Esta red se elabora mediante el ajuste de pesos, funciones y valores a lo largo de la red, mediante métodos de ajuste de dichos valores, conocidos como métodos de aprendizaje; entre ellos está el famoso 'backpropagation', que reajusta todos los valores desde la salida (capa de salida) hacia la entrada (capa de entrada), con objetivo de dejar la red preparada para que, ante entradas con características parecidas, se obtenga una salida correcta. A este fenómeno se le denomina entrenamiento de la red, y se necesitan de varias iteraciones para crear un modelo adecuado.

Al pie de página vemos la función de red, donde  $g_i(x)$  es la composición de funciones de la entrada 'x' en el enlace 'i',  $w_i$  es el vector de pesos del enlace número 'i' y 'k' es la función de activación del en la neurona 'x'. Una de las funciones de activación más famosas y utilizadas es la de la tangente hiperbólica o la función sigmoide.

---

<sup>1</sup>Función de red:  $f(x) = k(\sum_i^N w_i g_i(x))$

Normalmente, en este tipo de modelos se dispone de un set de datos bastante amplio para la fase de entrenamiento; es muy importante el tipo de datos que se le introduce dado que deben componer toda la variedad de características posibles, intentando no repetirse, dado que existen problemas de sobre ajuste (overfitting), que consisten en, básicamente, fiarse mucho de un tipo de dato en particular, sobre ajustándose al mismo. Un ejemplo muy simple inventado, disponemos de círculos de colores blanco y negro, a la red se le enseñan 30 círculos negros y uno blanco.; la red conoce ambos tipos, pero al introducir un círculo grisáceo casi blanco, se decantará por el negro, dado que existe un sobre ajuste en la cantidad de datos de una característica y la red ha sido entrenada para detectar mejor el negro que el blanco, y solo detectará el último al no ser que el círculo sea lo suficientemente blanco como para detectarlo. Este ejemplo refleja en la gran importancia de los datos de entrenamiento, mucho más que en los datos aleatorios que puedan usarse en el funcionamiento cotidiano de la red. **Es por ello por lo que no cualquier base de datos con palabras en español aleatorias sirve para el entrenamiento de una red.**

En la Figura 5.2 vemos el esquema de una red neuronal artificial [26], donde vemos las capas de entrada, salida e intermedias.

En nuestro caso, tenemos una entrada con pocas neuronas, dado que del audio se pueden extraer pocas características, en comparación a la salida, que es todo el repertorio de palabras de un idioma. Nuestra red tendrá forma de triángulo con la punta apuntando hacia la entrada, dado que las neuronas de entrada (características procesadas del audio) son menos que las de salida (todas las palabras de una lengua). Al final, obtenemos un número en cada capa de salida, que representará una probabilidad o puntuación de la palabra respecto a la entrada; nos quedaremos con la palabra mejor puntuada. Sin embargo, si en vez de las características reducidas de un audio, entrenamos la red a raíz de las muestras, tenemos otro esquema distinto.

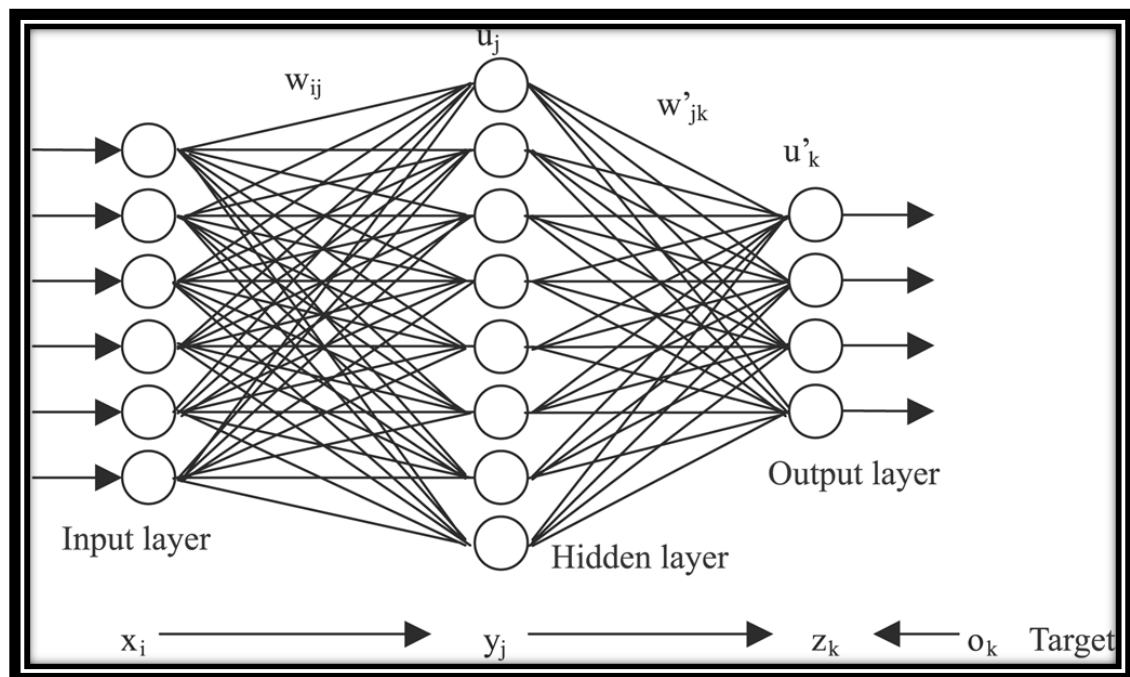


Figura 5.2.1.- Esquema de conexiones de un NNM. [27]

A la hora de implementar un el reconocimiento de audio de este tipo podemos recurrir a diferentes librerías o modelos ya elaborados y entrenados como los siguientes:

1. **KALDI:** [29] Es un kit de herramientas desarrollado en el lenguaje de programación C++ que permite el reconocimiento de audio y procesamiento de señales.
2. **Speech Recognition:** [1] Como mencionamos en el capítulo anterior, este módulo puede implementar cualquier tipo de modelo que entrenemos nosotros. Existen modelos para el reconocimiento de audio en inglés de este tipo.
3. **DeepSpeech:** [28] Este proyecto utiliza la librería de Google de Tensorflow (librería diseñada para la elaboración de redes neuronales artificiales) para implementar un modelo de redes neuronales artificiales. Como los anteriores, necesitan de un modelo, aunque los que se pueden encontrar actualmente son en inglés. Se basa en un paper de investigación elaborado por la Universidad de Cornell [28].

La desventaja de estos modelos es que son en inglés, y en caso de buscar una solución en Castellano, tendríamos que recurrir a entrenar una red propia, por lo que necesitaríamos una base de datos apropiada para el entrenamiento de gran tamaño; cosa que muchas veces es muy difícil de obtener o se podría obtener pagando. Además de todo el trabajo que implica el entrenamiento y diseño de una red de este calibre.

### 5.3. Modelos offline (Hidden Markov Models)

Dentro del Machine Learning, existe un modelo probabilístico llamado Hidden Markov Model (HMM) o Modelo Oculto de Markov [4]. Dicho modelo entra en la categoría de Machine Learning y la ventaja respecto a los anteriores radica en que existen modelos funcionales en Castellano, lo que en algunos robots sociales es fundamental.

Para estos modelos, se asume que el proceso que está sucediendo es un proceso de Markov, es decir, un proceso estocástico basado en estados donde la probabilidad de transición a otros estados o al mismo estado depende únicamente del estado en el que el proceso se encuentre. Un ejemplo muy básico de un proceso de Markov sería el siguiente:

Llamemos  $X$  a la probabilidad de que hoy llueva, y llamemos  $Y$  al estado actual (si hoy ha llovido o si no ha llovido).  $X$  depende de  $Y$ , imaginemos que si ha llovido la posibilidad de que llueva mañana es mayor a la que si no ha llovido, por ende,  $X$  sería mayor si  $Y$  está en el estado 'ha llovido hoy' que si está en el estado 'no ha llovido hoy'. Este ejemplo básico muestra cómo el estado actual influye en la probabilidad de transición a otros estados, es decir un modelo de Markov.

Esto es aplicable al reconocimiento de voz, dado que la probabilidad de obtener una palabra u otra también depende de las palabras que le preceden. Por ejemplo, después de un artículo hay muchas probabilidades de que la siguiente palabra sea un sustantivo, esto el modelo de Markov lo tiene en cuenta a la hora de generar el texto desde el audio.

Al igual que las redes neuronales artificiales (NNM), este modelo se basa en un modelo por capas que puede ser entrenado en base a un modelo de lenguaje (refiérase al capítulo anterior para ver el modelo por capas), en este caso en vez de ajustar pesos y funciones, se ajustan las probabilidades de transición entre estados, donde la mayoría están ocultos y no son observables, es por eso por lo que se le denomina modelo oculto de Markov, y al ajustar valores internos de la red mediante un entrenamiento, se considera como Machine Learning.

La Figura 5.3.1 muestra un esquema de este tipo de modelos.

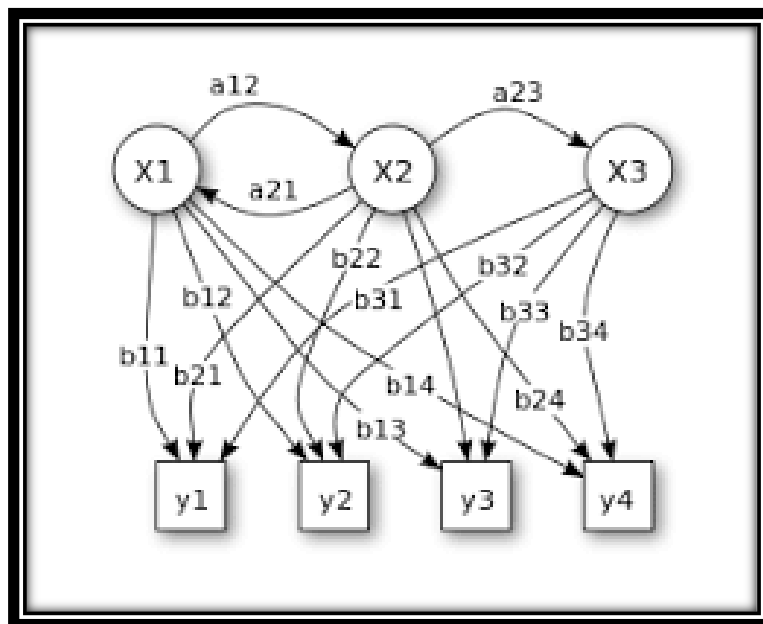


Figura 5.3.1.- Esquema HMM [30].

## 6. Objetivos y modelo seleccionado.

### 6.1. Objetivos del proyecto.

Para el modelo que estamos buscando necesitamos que cumpla una serie de características:

1.- Debe de ser un modelo que pueda funcionar sin conexión a internet. Por tanto, los modelos online del apartado 5.1 quedan totalmente descartados. Que se requiera de conexión a internet para la instalación del modelo no es un problema para el modelo que se desea establecer. Pero sí debe de ser capaz de funcionar en entornos donde no disponga de dicha conexión, puesto que en principio el robot puede desplazarse fuera del área de cobertura WiFi y mucho menos la disposición de un cable Ethernet hacia al mismo.

Sin embargo, existe la posibilidad de que con el desarrollo del 5G pueda incorporarse algún tipo de conexión a internet móvil. Es por eso por lo que se menciona, pese a que no estemos buscando dicha solución.

2.- El modelo debe poder funcionar en Castellano (español – España). Encontrar un modelo en español no es sencillo, debido a que la mayoría de los modelos son resultado de proyectos de investigación fuera de España o modelos desarrollados por ingenieros que no trabajan en español. Es mucho más común encontrarlos en inglés, particularmente no he encontrado ningún NNM funcional en castellano, aunque sí he encontrado diccionarios para HMM.

Sin embargo, existe la posibilidad de que en un futuro se desarrolle o se encuentre, en caso de que ya esté desarrollado, un modelo más eficiente mediante NNM, es por eso por lo que se tiene en cuenta también.

3.- El modelo debe poder funcionar en una Raspberry Pi 3, por lo que no debe de tener un cómputo interno que el procesador de dicha placa no pueda mantener en pie de manera fluida. En el caso de HMMs y NNMs el cómputo del programa a la hora de pasar el audio a texto es poco, dado que el peso computacional radica más en entrenar o enseñar a la máquina (acción realizada antes de la ejecución del programa) que a procesar el audio. Por lo que estos modelos trabajan en tiempo real y son completamente funcionales en Raspberry.

## 6.2. Modelo seleccionado.

Aplicando las restricciones del apartado anterior (6.1 Objetivos del proyecto) se selecciona un HMM. En nuestro caso se trata de un modelo llamado PocketSphinx, que utiliza SphinxBase para su funcionamiento. Todo esto ha sido desarrollado bajo el término de CMU Sphinx, en la universidad de Carnegie Mellon [5]. Que corresponde a todo el kit de herramientas para el reconocimiento de audio.

SphinxBase contiene los modelos y el entrenador de estos (SphinxTrain).

PocketSphinx es la librería que proporciona una interfaz para SphinxBase y poder utilizar los modelos en sistemas embebidos. Además de disponer de librerías para Python. En la figura 6.2.1 se muestra un diagrama funcional extraído de [6] que explica los procesos de comunicaciones entre las aplicaciones y el modelo. En la figura 6.2.2 se especifica más el diagrama funcional.

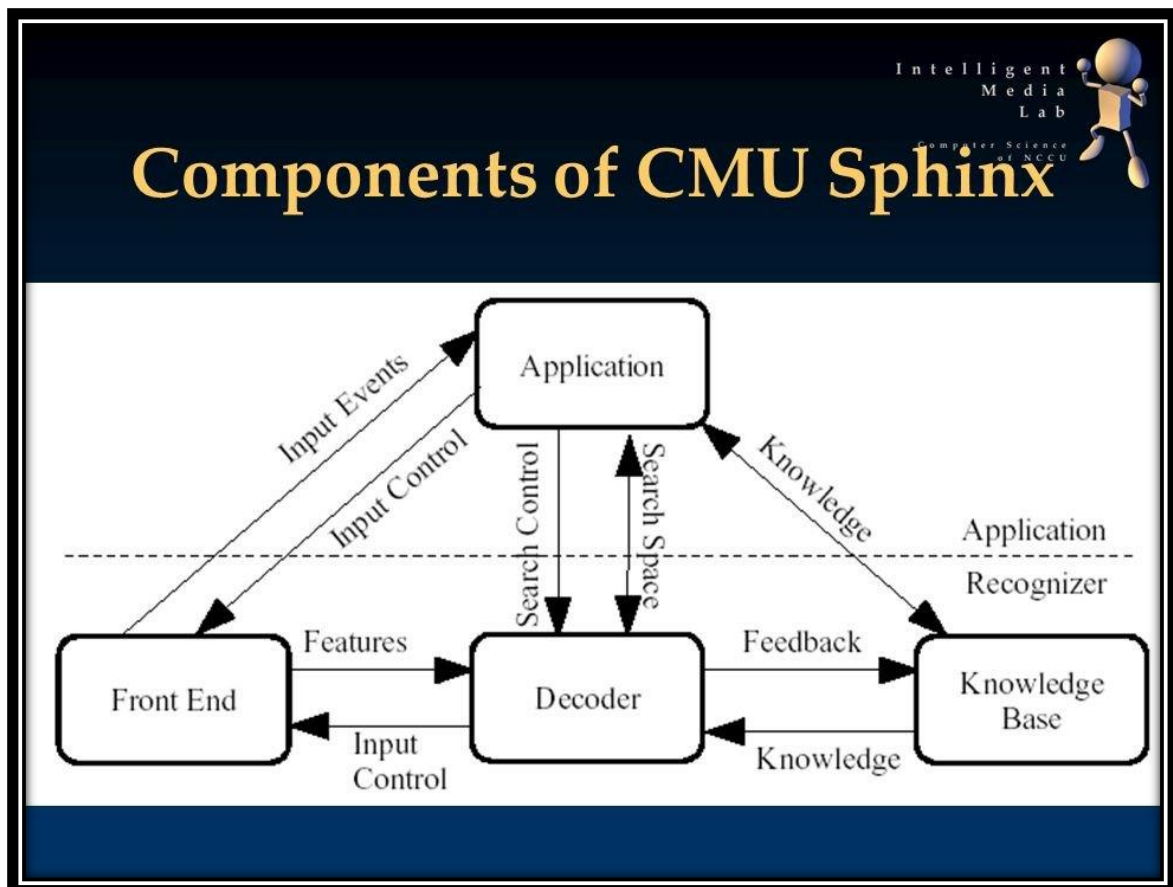


Figura 6.2.1. Componentes de CMU Sphinx. [6]



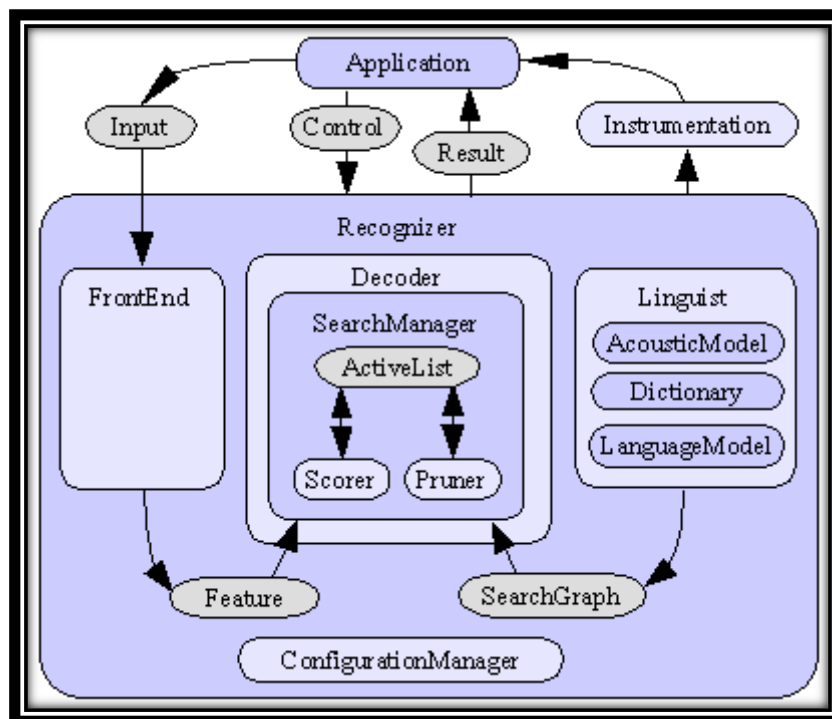


Figura 6.2.2. Diagrama de funcionamiento CMU Sphinx. [6]

Principalmente, se dispone de un sistema FrontEnd que proporciona una interfaz al decodificador. Es decir, traduce la entrada de la aplicación a características que se le pasan al modelo. Se utiliza un modelo HMM con un 'scorer' que junto con los diccionarios y los modelos que representan el lenguaje utilizado y consiguen obtener la palabra que más se parece. Coloquialmente, podemos decir que la aplicación le da un audio al programa en FrontEnd, esta lo traduce a unas características y el decodificador lo compara con las características de los modelos que tiene almacenados. Tras la comparación el decodificador produce un resultado de la comparación (una palabra o frase), que se le devuelve a la aplicación.

Todo esto es posible de manejar de manera muy superficial y a muy alto nivel con la librería PocketSphinx de Python. Gracias a su sencillez, los scripts resultantes son cortos y flexibles; aunque por dentro, SphinxBase utiliza modelos tan complejos de inteligencia artificial.

## 7. Descripción del sistema.

### 7.1. Descripción genérica.

El sistema consta de cinco partes definidas de las cuales se involucran tanto software como hardware. En la Figura 7.1.1. se muestra un esquema de conexión a nivel funcional de todos los sistemas involucrados en el funcionamiento de la característica de reconocimiento de palabras implementada en el robot social. El robot constará de más sistemas además de los expuestos en la figura y en el proyecto, por lo que el diagrama muestra los sistemas necesarios para la implementación de dicha característica.

Primero tenemos un micrófono para poder captar audio, vale con cualquier micrófono que disponga de un driver para Linux, conectado al puerto USB de la Raspberry Pi 3 o 4 que se disponga para controlar el robot. ALSA (Advanced Linux Sound Architecture) se encargará de gestionar, junto con los drivers pertinentes, la entrada de audio del micrófono. Cabe recalcar que entrar dentro de esta arquitectura se escapa de los límites de este proyecto.

ALSA está sobre Linux (en nuestro caso es Raspbian de 64 bits) por lo que el sistema operativo automáticamente se encarga de la gestión del audio. Mediante Python nos encargamos de recolectar las características del audio y transmitirlas a PocketSphinx que hará uso de SphinxBase y el modelo en castellano para comparar las características de la ráfaga de audio<sup>2</sup> y devolver las palabras identificadas en el proceso, devolviéndolas mediante la librería en Python de PocketSphinx hacia el script estático.

El script estático hará uso de los parámetros y un diccionario de palabras seleccionadas por el programador del sistema. Cabe recalcar que en el diccionario deben de estar las palabras que el modelo pueda reconocer. Adicionalmente se utiliza un mapa de corrección de errores que mejora el rendimiento del sistema.

El script estático se encarga de traducir las palabras detectadas a señales internas del programa, no confundir con las señales que se mandan mediante el sistema operativo, por lo tanto, dichas señales se le pasa al manejador de señal (el cuál traduce estas señales internas del programa a señales del sistema operativo) para realizar las tareas oportunas en función de las palabras detectadas. Dicho manejador se deja a elección del usuario, dado que no sabemos que funcionalidad quiere darle ni que palabras quiere incluir en el diccionario.

El manejador de señal interactúa con el sistema operativo para conectar mediante GPIO u otra forma de comunicación a los periféricos del robot, que serán motores, luces, etc.

---

<sup>2</sup> Se define ráfaga de audio como todas las muestras de audio realizadas desde el inicio de la una frase hasta que esta termina.

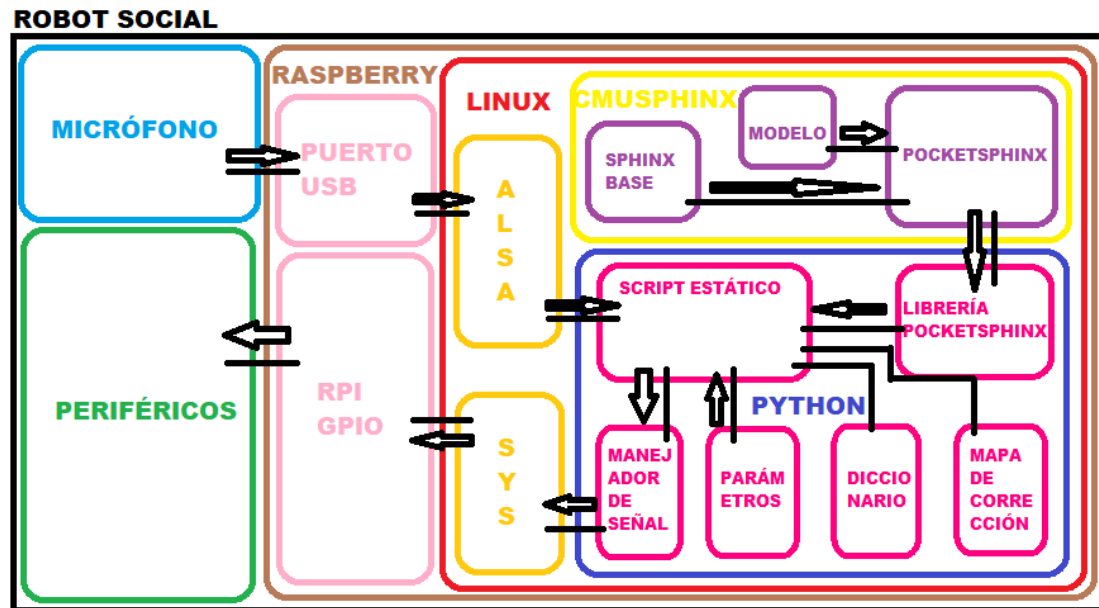


Figura 7.1.1.- Esquema de conexión de todos los sistemas involucrados.

NOTA: Referirse a la Figura 11.1. para ver qué ficheros puede modificar el usuario.

## 7.2. Descripción hardware.

El hardware necesario para que el sistema pueda desempeñar la función de reconocimiento de frases y palabras son los siguientes:

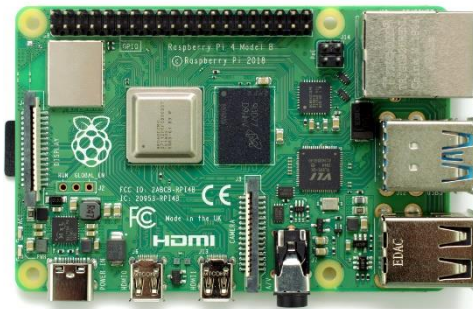
1.- **Micrófono.** Basta con un micrófono o un dispositivo de grabación de audio genérico que disponga de un controlador para Linux. Cabe recalcar que en función de la calidad del micrófono obtendremos unas prestaciones u otras.

- Necesitaremos un micrófono con una sensibilidad capaz de reconocer palabras a 1 metro de distancia como mínimo para frecuencias inferiores a 24kHz, para un audio de calidad media.

- Cuanta menos directividad para esta aplicación mejor será el micrófono, dado que la posición del robot será aleatoria y el sonido puede venir de cualquier posición.



Figura 7.2.1.- Micrófono USB. [7]



*Figura 7.2.2.- Raspberry Pi 4. [8]*



*Figura 7.2.3.- Motor 12V. [9]*



*Figura 7.2.4.- Tarjeta microSD. [10]*

## 2.- Raspberry Pi 3 o Raspberry Pi 4.

Para ambas versiones de la placa el sistema funciona correctamente, aunque se recomienda el modelo 4 si se desean correr muchos procesos en paralelo a la característica de reconocimiento de palabras y frases. Es imprescindible tener un puerto USB para el micrófono y alimentación para la placa ya sea vía batería o vía alimentación por USB tipo C. La ocupación de la característica es de un puerto.

3.- Periféricos. En función del enfoque que tenga el robot se utilizarán unos periféricos u otros, eso se deja a elección del usuario. Un periférico puede llegar desde un simple LED controlado por GPIO como un controlador de un motor de 12V; como se muestra en la figura.

Normalmente el control de robots en este tipo de sistemas se realizan mediante ROS (Robot Operating System), pudiendo realizar las operaciones de manera mucho más flexible y programable.

4.- Tarjeta microSD. Raspberry Pi utiliza tarjetas SD como discos duros del sistema operativo. Es necesario disponer de un mínimo de 8GB para el sistema operativo y el software que incorpora el proyecto. Con una tarjeta microSD de 64 GB sirve y sobra, añadiendo los programas que quiera añadir el usuario. Lo ideal es usar una tarjeta tipo 10 A1, esto influye en la velocidad de lectura en la tarjeta, que a su vez depende de la necesidad del usuario en el robot.

El presupuesto en hardware mínimo para implementar dicha característica puede rondar los 100€ - 117\$ si se compran unos periféricos básicos, como LEDs y resistencias. Pudiendo subir a los 300€ - 350\$ si se está diseñando un Robot con muchos periféricos y características.

### 7.3. Sistema operativo.

El sistema operativo utilizado es Raspbian Buster (versión de 64 bits) [11]. Este sistema operativo es el sistema operativo por defecto de la Raspberry y es el más optimizado para ello, dado que el diseño de este sistema operativo ha sido elaborado exclusivamente para los procesadores de dichas placas.

Adicionalmente, se puede añadir un sistema operativo para el robot (ROS), que se encargará de manejar todo lo correspondiente a las señales de control y procesado de este. Es muy común el uso de ROS en proyectos donde un robot tiene varias tareas, por lo que, si se desea instalar un ROS, se recomienda del sistema operativo orientado a robots de NOETIC [30], que será el que se instale en el robot al que va dirigido el proyecto.

### 7.4. Dependencias.

Una vez instalado el sistema operativo se instalarán las dependencias que utiliza el programa, dichas dependencias son las siguientes:

- |                      |  |
|----------------------|--|
| 1. Python:           | Intérprete o lenguaje de programación.   |
| 2. Pip:              | Instalador de librerías y módulos de Python.   |
| 3. Wheel:            | Manejador de privilegios en Linux.   |
| 4. Setuptools:       | Kit de programas para la configuración en Linux.   |
| 5. Libasound2-dev:   | Librería de sonido (extensión).  |
| 6. Swig:             | Interfaz para Linux orientada al desarrollo de programas.  |
| 7. Python-dev:       | Extensiones de Python.   |
| 8. Libpulse-dev:     | Extensión de la librería de audio en Linux.  |
| 9. Git:              | Sistema de control de versiones, necesario para descargar programas.   |
| 10. Autoconf:        | Kit de programas para configuración automática en Linux.   |
| 11. Libtool:         | Kit de programas para manejo de librerías en Linux.  |
| 12. Automake:        | Kit de programas para instalación de software en C.  |
| 13. Bison:           | Programa generador de análisis sintácticos.  |
| 14. Build-essential: | Kit de programas para instalación de software en Linux.  |
| 15. PULSEAUDIO:      | Esta dependencia suele estar instalada en Ubuntu pero no en Debian, es muy importante asegurarse de tenerla, de lo contrario se experimentarán problemas con la detección del micrófono. |

## 7.5. SphinxBase.

Es un módulo que implementa la base de PocketSphinx, se instala antes que este y proporciona la implementación de toda la teoría, entrenamiento e interfaz de Linux de PocketSphinx.

## 7.6. PocketSphinx.

Es un módulo que proporciona una interfaz de SphinxBase a las librerías de Python de Shpinx. Contiene los modelos del programa en el idioma que se haya entrenado, los parámetros de este y configuración específica del usuario.

En los próximos capítulos de hablará más de estos programas.

## 7.7. Librería de Python.

Existe una librería de Python que proporciona la interfaz a programación de cara al usuario, con esto, el usuario llamará a funciones que se encuentran en el módulo de PocketSphinx, esta librería se instala vía 'pip' y contiene funciones de una única llamada, lo que proporciona una interfaz muy simple en Python. La librería se llama pocketsphinx y la función de interés LiveSpeech(), que recoge el audio del micrófono configurado y devuelve texto.

## 7.8. ALSA.

ALSA es la estructura o arquitectura de Linux para sonido (Advanced Linux Sound Architecture), para el uso que le vamos a dar, llamará a PulseAudio, otra arquitectura de Linux orientada al procesamiento de sonido, que es más eficiente que ALSA, que implementa únicamente las funciones básicas de la arquitectura. PulseAudio será llamado por ALSA para el procesamiento de audio. Para más información vaya al capítulo 8.4, donde se explica la detección de micrófono en ALSA.

## 7.9. Script estático.

Además de los programas anteriores, se dispone de un script estático que llama a la librería de Python y manejará el sistema por dentro con un programa en Python, por lo que el usuario no deberá de intervenir en este script. El control del sistema importa todas las librerías necesarias, proporciona una interfaz gráfica vía línea de comandos, controla la apertura y cierre de ficheros y del programa, utiliza las funciones de la librería de Python para traducir el audio a texto, importa los modelos en español y controla el flujo del programa.

El script se encuentra en el Anexo I (Capítulo 16.2 Sampler.py (Script estático)).

## 7.10. Scripts modificables, parámetros y diccionarios.

Junto al script estático se encuentran scripts que el usuario debe modificar. Refiérase al Capítulo 11 Manual de usuario y Capítulo 12 Manual del programador para más información sobre estos programas y ficheros. Entre ellos podemos encontrar los siguientes:

1. FSMSignal: Manejador de señales generados por el programa principal, que reside en el script estático del sistema.
2. DICT: Diccionario personalizado por el usuario, define las señales en función de las palabras que quiere el usuario detectar.
3. PARAMETERS: Parámetros del programa.
4. CMap: Mapa de corrección del sistema. Corrige errores comunes dentro del sistema. Debe de ser ajustado empíricamente.

## 8. Instalación del modelo.

Para la instalación de la característica necesitamos un sistema operativo basado en Debian, funcional para la Raspberry y que sea lo suficientemente ligero como para poder ser instalado en una tarjeta microSD. En el manual de instalación, supondremos que no se dispone de un sistema operativo; es decir, el robot está completamente vacío en cuanto a software se refiere.

Los pasos a seguir para la instalación son los siguientes:

- 1.- Instalación del sistema operativo.
- 2.- Instalación de dependencias.
- 3.- Instalación de CMU Sphinx.
- 4.- Instalación de los scripts.
- 5.- Ajuste de las variables de entorno.
- 6.- Diseño de un manejador de señales personalizado.

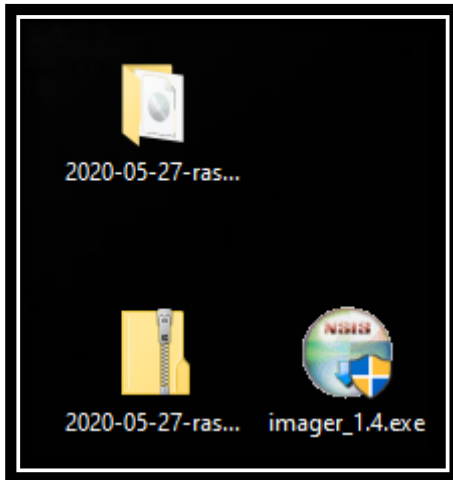
Cabe destacar que se dispone de un instalador que combina los pasos [2,3,4,5] en una simple ejecución de un programa en bash que se proporciona junto al paquete de instalación. En los siguientes apartados veremos cómo se puede instalar la característica por medio del instalador y vía manual, sin instalador.

### 8.1. Instalación del sistema operativo.

El sistema operativo, como se ha mencionado en el Capítulo 7.3, será Raspbian Buster de 64 bits, disponible por descarga oficial de Raspberry Pi [11]. Además, dispondremos de un instalador oficial de [12] la fundación Raspberry (Raspbian Pi Imager) que nos permitirá instalar el sistema operativo que hemos descargado en la tarjeta microSD que actuará de disco duro.

Dicho esto, descargaremos el archivo zip [1] y el instalador del programa Raspberry Pi Imager [12]. Descomprimiremos el archivo zip [1] y se nos mostrará en una carpeta como muestra la Figura 8.1.1.






Dentro de la carpeta que hemos obtenido tras la extracción del fichero zip [11] tendremos la imagen que hay que instalar del sistema operativo. La imagen ocupa 3624 MB, algo más de 3GB y medio, por lo que, para instalar el sistema operativo en la microSD, la tarjeta debe de tener como mínimo 4GB de almacenamiento dedicado al sistema operativo, más el espacio que necesitaremos para instalar aplicaciones y dependencias que nos permitan ejecutar la característica en el robot social.

Ejecutaremos `imager_1.4.exe` para instalar el programa que nos permitirá añadir la ya mencionada imagen del sistema operativo a la tarjeta microSD.

*Figura 8.1.1.- Archivos descargados.*

Nombre	Fecha de modificación	Tipo	Tamaño
 2020-05-27-raspbian-buster-arm64.img	20/10/2020 18:04	Archivo de image...	3.624.960 KB

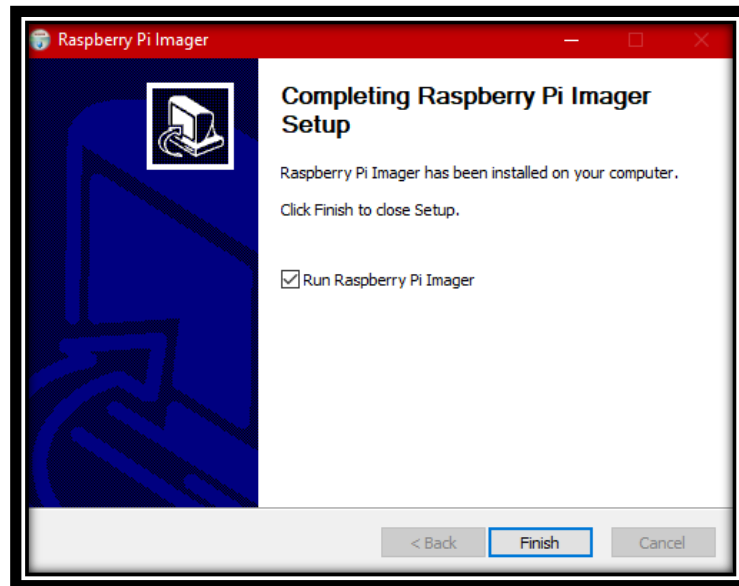
*Figura 8.1.2.- Imagen del sistema operativo en la carpeta.*

Se mostrará a continuación la instalación del programa que transcribirá la imagen a la tarjeta en un ordenador con Windows 10 instalado. Lo primero, deberemos ejecutar el programa con permisos de administrador y se nos abrirá el Wizard como muestra la Figura 8.1.3.



*Figura 8.1.3.- Wizard para Raspberry Pi Imager.*

Esperaremos a que se cargue la barra de instalación y finalizaremos el Wizard como muestra la Figura 8.1.4, y daremos clic en el botón de Finish o botón de finalizar la instalación. A continuación, abriremos la aplicación Raspberry Pi Imager, manteniendo la caja “Run Raspberry Pi Imager” con un ‘tick’ a la hora de cerrar el Wizard.



*Figura 8.1.4.- Finalización de la instalación del Wizard.*

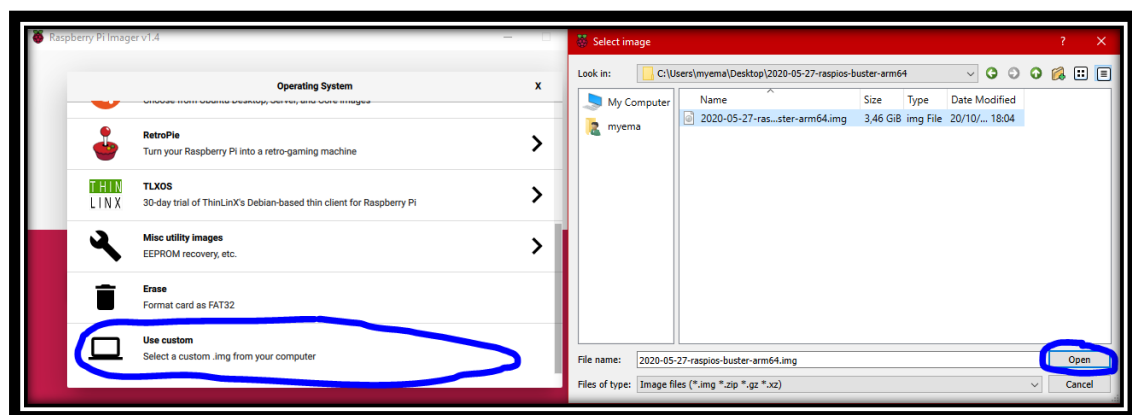
Una vez dentro del programa, insertaremos la tarjeta microSD en el lector de tarjetas de nuestro ordenador en el cual tenemos descargados tanto la imagen como el programa. Y seleccionaremos primero: CHOOSE A SD CARD. Una vez reconocida nuestra tarjeta, seleccionaremos la misma. Se recomienda tener únicamente conectada la tarjeta microSD sin otro tipo de dispositivos de almacenamiento (ya sea USB o discos) que nos dificulten la identificación de la tarjeta.

A continuación, seleccionaremos CHOOSE OS -> USE CUSTOM, que se encuentra como la última opción de la lista. Luego, seleccionaremos nuestra imagen en la carpeta que nosotros hayamos dejado la misma y clicaremos en OPEN.

Finalmente seleccionaremos la opción de WRITE, cabe destacar que esta opción borrará todos los datos que la tarjeta microSD tenía anteriormente, por lo que, si se desea mantenerlos, debemos hacer una copia de seguridad de esta. Seleccionaremos que sí a todos los recuadros que aparezcan y esperaremos a que la barra de escritura se complete. Luego automáticamente expulsará la tarjeta microSD, ya casi preparada para insertar en la Raspberry Pi.



*Figura 8.1.5.- Raspberry Pi Imager.*



*Figura 8.1.6.- Selección de la imagen a instalar.*

Una vez expulsada la tarjeta se recomienda volver a insertarla al ordenador si se va a utilizar el puerto HDMI / uHDMI, dado que la mayoría de las veces, no se podrá detectar automáticamente la televisión o monitor que se conecta a la placa. Si dicho problema se da, podemos arreglarlo la mayoría de las veces accediendo a la tarjeta mediante nuestro ordenador con lector de tarjetas en la ruta:

boot(D:)/config.txt

Buscaremos la línea donde ponga:

#hdmi\_force\_hotplug=1

y borraremos el carácter # de tal manera que ponga:

hdmi\_force\_hotplug=1

Nombre	Fecha de modificación	Tipo	Tamaño
overlays	27/10/2020 15:08	Carpeta de archivos	
bcm2708-rpi-b.dtb	27/10/2020 14:58	Archivo DTB	25 KB
bcm2708-rpi-b-plus.dtb	27/10/2020 14:58	Archivo DTB	25 KB
bcm2708-rpi-b-rev1.dtb	27/10/2020 14:58	Archivo DTB	25 KB
bcm2708-rpi-cm.dtb	27/10/2020 14:58	Archivo DTB	25 KB
bcm2708-rpi-zero.dtb	27/10/2020 14:58	Archivo DTB	25 KB
bcm2708-rpi-zero-w.dtb	27/10/2020 14:58	Archivo DTB	26 KB
bcm2709-rpi-2-b.dtb	27/10/2020 14:58	Archivo DTB	26 KB
bcm2710-rpi-2-b.dtb	27/10/2020 14:58	Archivo DTB	26 KB
bcm2710-rpi-3-b.dtb	27/10/2020 14:59	Archivo DTB	28 KB
bcm2710-rpi-3-b-plus.dtb	27/10/2020 14:59	Archivo DTB	28 KB
bcm2710-rpi-cm3.dtb	27/10/2020 14:59	Archivo DTB	26 KB
bcm2711-rpi-4-b.dtb	27/10/2020 14:59	Archivo DTB	47 KB
bcm2711-rpi-cm4.dtb	27/10/2020 14:59	Archivo DTB	47 KB
bootcode.bin	27/10/2020 15:09	Archivo BIN	52 KB
cmdline.txt		Documento de te...	1 KB
config.txt	20/10/2020 19:44	Documento de te...	2 KB
COPYING.linux	27/10/2020 14:59	Archivo LINUX	19 KB
fixup.dat	27/10/2020 15:09	Archivo DAT	8 KB
fixup_cd.dat	27/10/2020 15:09	Archivo DAT	4 KB
fixup_db.dat	27/10/2020 15:09	Archivo DAT	11 KB
fixup_x.dat	27/10/2020 15:09	Archivo DAT	11 KB

*Figura 8.1.7.- Ruta config.txt*

```
# uncomment if hdmi display is not detected and composite is being output
hdmi_force_hotplug=1
```

*Figura 8.1.8.- Línea a modificar en config.txt*

Si se experimentan otras anomalías durante la instalación del sistema operativo en la tarjeta microSD, refiérase a cualquier tutorial de instalación de imágenes en Raspberry Pi, dado que no es competencia de este proyecto la solución de errores en instalaciones de sistemas operativos en Raspberry y aquí únicamente se aporta un tutorial simple de dicha instalación; dado a que existe una gran variedad de tarjetas, monitores y dispositivos que pueden dar lugar a errores en la instalación. Solucionados dichos errores y teniendo ya disponible el sistema operativo en su Raspberry Pi, puede seguir con la línea principal de este manual de instalación.

## 8.2. Instalación vía software. (Online)

Tras la instalación del sistema operativo en la tarjeta microSD (Capítulo 8.1), procederemos a instalar la característica de reconocimiento de voz en el robot. Si el robot dispone de una red de acceso a internet podemos continuar con este capítulo. Si no se dispone de una red de acceso a internet refiérase al siguiente capítulo (Capítulo 8.3), que explica la instalación manual para dispositivos sin acceso a internet.

Para este capítulo utilizaremos un programa en bash que instala automáticamente la característica de reconocimiento de voz con todos los scripts y dependencias además de generar el sistema de archivos.

Para ejecutarlo primero, necesitamos configurar la red de acceso a internet. Por lo que enchufaremos un cable ethernet a la Raspberry Pi o configuraremos la red WiFi. Tras haber configurado el acceso a la red, clonaremos el repositorio [13] en el escritorio de nuestra Raspberry Pi ( /home/pi/Desktop/ ) con los siguientes comandos:

(Home)/(Usuario)/\$	<b>cd Desktop</b>
(Home)/(Usuario)/\$	<b>sudo apt-get install git</b>
(Home)/(Usuario)/Desktop/\$	<b>git clone <a href="https://github.com/iTzAlver/SpeechRecog">https://github.com/iTzAlver/SpeechRecog</a></b>
(Home)/(Usuario)/Desktop/\$	<b>mv SpeechRecog/install/installer.sh installer.sh</b>

A continuación, podemos eliminar todo el repositorio y ejecutar el script que acabamos de desplazar al escritorio del sistema.

```
(Home)/(Usuario)/Desktop/$      rm -r SpeechRecog
(Home)/(Usuario)/Desktop/$      ./installer.sh
```

Esperaremos a que la instalación se realice, y a continuación podemos eliminar el script de instalación y comprobar mediante el comando Lola si la característica está instalada correctamente.

```
(Home)/(Usuario)/Desktop/$      rm installer.sh
(Home)/(Usuario)/Desktop/$      Lola
```

Si recibimos el error de que el comando Lola no se reconoce, podemos arreglar ese error fácilmente con el siguiente comando que actualiza los comandos para el usuario actual:

```
(Home)/(Usuario)/Desktop/$      source (Home)/.bashrc
(Home)/(Usuario)/Desktop/$      Lola
```

Si aun así no se consigue ejecutar el comando Lola se recomienda realizar una instalación manual (Capítulo 8.3) y ver exactamente dónde reside el error. Si el programa se ejecuta y el sistema devuelve un error parecido al siguiente refiérase al Capítulo 8.4, ya que es un problema de detección de dispositivo de entrada de audio, que es un problema del sistema operativo, ALSA o PulseAudio:

**Error opening audio device (null) for capture connection refused**

Resumiendo, estos son los comandos a realizar para completar la instalación sobre el sistema operativo seleccionado:

```
(Home)/(Usuario)/$ cd Desktop
(Home)/(Usuario)/$ sudo apt-get install git
(Home)/(Usuario)/Desktop/$ git clone https://github.com/iTzAlver/SpeechRecog
(Home)/(Usuario)/Desktop/$ mv SpeechRecog/install/installer.sh installer.sh
(Home)/(Usuario)/Desktop/$ rm -r SpeechRecog
(Home)/(Usuario)/Desktop/$ ./installer.sh
(Home)/(Usuario)/Desktop/$ rm installer.sh
(Home)/(Usuario)/Desktop/$ source (Home)/.bashrc
(Home)/(Usuario)/Desktop/$ cd ..
(Home)/(Usuario)/$ Lola
```

El instalador realizará las siguientes acciones:

1. Instalación de dependencias.
2. Instalación de PulseAudio.
3. Crear un directorio sobre (Home)/(Usuario)/ llamado sphinx
4. Descargar, configurar e instalar SphinxBase. [14]
5. Descargar, configurar e instalar PocketSphinx. [15]
6. Instalar más dependencias. (pip, Wheel y setuptools)
7. Instalar librería de PocketSphinx en Python.
8. Descargar el repositorio con los scripts.
9. Instalar los modelos en español.
10. Instalar el comando Lola.
11. Ajustar variables del sistema.
12. Borrar los residuos de la instalación.

El script de instalación puede encontrarse en el Anexo I. (Capítulo 15.1)

### 8.3. Instalación manual. (Offline)

En este tipo de instalación, procederemos a instalar todos los paquetes a falta del instalador. Lo primero será descargar los tres repositorios en un ordenador que sí disponga de conexión a internet. Por lo tanto, descargaremos SphinxBase [14], PocketSphinx [15] y el repositorio de scripts [13]. A continuación, buscaremos una herramienta que nos permita instalar paquetes vía USB, por lo que **necesitaremos un pendrive** para este tipo de instalaciones. En nuestro caso, usaré un simple tutorial de Keryx [17][16], que es un agente de descargar offline en GNU/Linux diseñado precisamente para este tipo de situaciones. En el enlace [16] se proporciona un tutorial más extenso sobre este tema.

Se pueden utilizar otro tipo de herramientas que permiten la instalación en Linux de paquetes sin necesidad de disponer de una red de acceso a internet. Aunque siempre que sea posible se recomienda utilizar una red de acceso momentánea para la instalación de los paquetes necesarios, dado que las instalaciones offline suelen ser más complicadas y aparatosas, especialmente en sistemas operativos basados en Debian.



*Figura 8.2.1.- Keryx. [16]*



Los pasos que debemos seguir para realizar una instalación sobre Keryx son los siguientes:

1. Instalar Keryx en Windows 10 u otro sistema operativo con conexión a internet.
2. Conectar un dispositivo USB.
3. Crear un proyecto en Keryx.
4. Añadir los siguientes paquetes de instalación a Keryx:
  - 4.1. Python. (2.7 o 3)
  - 4.2. Python-dev.
  - 4.3. Python-pip
  - 4.4. Build-essential
  - 4.5. Swig
  - 4.6. Libpulse-dev
  - 4.7. Libasound2-dev
  - 4.8. Autoconf
  - 4.9. Libtool
  - 4.10. Automake
  - 4.11. Bison
  - 4.12. PulseAudio
  - 4.13. Pip-Wheel
  - 4.14. Pip-Setuptools
  - 4.15. Pip-Pip
  - 4.16. Pip-pocketsphinx
5. Extraer el USB e insertarlo en el robot.
6. Instalar los paquetes seleccionados.
7. Extraer el USB e insertarlo en el ordenador con acceso a internet.
8. Formatear el USB e insertar los repositorios de SphinxBase, PocketSphinx, SpeechRecog.
9. Extraer el USB e insertarlo en el robot.
10. Crear una carpeta en el directorio raíz: /home/usuario/sphinx.
11. Copiar las carpetas sphinxbase, pocketsphinx y speechrecog en la carpeta sphinx.
12. Ejecutar los siguientes comandos en la línea de comandos de Linux:

<b>(Home)/(Usuario)/\$</b>	<b>cd sphinx</b>
<b>(Home)/(Usuario)/sphinx\$</b>	<b>cd sphinxbase</b>
<b>(Home)/(Usuario)/sphinx/sphinxbase\$</b>	<b>./autogen.sh</b>
<b>(Home)/(Usuario)/sphinx/sphinxbase\$</b>	<b>./configure</b>
<b>(Home)/(Usuario)/sphinx/sphinxbase\$</b>	<b>make</b>
<b>(Home)/(Usuario)/sphinx/sphinxbase\$</b>	<b>sudo make install</b>
<b>(Home)/(Usuario)/sphinx/sphinxbase\$</b>	<b>cd ..</b>

```
(Home)/(Usuario)/sphinx/$      cd pocketsphinx
(Home)/(Usuario)/sphinx/pocketsphinx$ ./autogen.sh
(Home)/(Usuario)/sphinx/pocketsphinx$ ./configure
(Home)/(Usuario)/sphinx/pocketsphinx$ make
(Home)/(Usuario)/sphinx/pocketsphinx$ sudo make install
(Home)/(Usuario)/sphinx/pocketsphinx$ cd ..
```

13. Extraer los ficheros situados en la ruta:  
(Home)/(Usuario)/sphinx/speechRecog/DICTS/
14. Renombrar es-20k.lm a es-20k.lm.bin.
15. Renombrar la carpeta dentro de cmusphinx-es.5.2/voxforge\_es\_sphinx.cd\_ptm\_4000 a es-es y moverla junto con es-20k.lm.bin y es.dict.
16. Mover todos los archivos (es-es, es.dict y es-20k.lm.bin) a:
  - 16.1. /home/usuario/.local/lib/python2.7/site-packages/pocketsphinx/model si estamos en Python 2.7.
  - 16.2. /home/usuario/.local/lib/python3/site-packages/pocketsphinx/model si estamos en Python 3.
17. Ir al archivo .bashrc situado en /home/usuario/.
18. Escribir en la última línea lo siguiente:

Python 2:

```
alias Lola="python /home/usuario/sphinx/SpeechRecog/scripts/sampler.py"
```

Python 3:

```
alias Lola="python3 /home/usuario/sphinx/SpeechRecog/scripts/sampler.py"
```

**NOTA:** Si el comando a ejecutar queremos que no se llame Lola si no que queremos que se llame de otra forma, basta con cambiar la palabra Lola de este archivo por la palabra que queremos que ejecute el programa.

19. Borrar los dictectorios: install y DICTS de la carpeta SpeechRecog, ubicada en sphinx, para una instalación limpia.
20. Ejecutar los siguientes comandos para actualizar las variables del sistema:

```
$ source /home/usuario/.bashrc
$ export LD_LIBRARY_PATH=/usr/local/lib
```

**NOTA:** Pueden ser ejecutadas en cualquier directorio, son llamadas al sistema.

## 8.4. Detección y funcionamiento de dispositivos de entrada de audio en ALSA y PocketSphinx.

Una vez instalado todo, podemos experimentar dificultades para ejecutar el programa, especialmente si recibimos el siguiente mensaje durante la ejecución de este como ya se ha mencionado capítulos atrás:

**Error opening audio device (null) for capture connection refused**

Esto significa que ALSA no es capaz de identificar un dispositivo de entrada de audio predeterminado y tendremos que configurarlo manualmente. Este problema incluye un abanico muy grande de posibilidad de errores, por lo que solucionarlo no es una tarea sencilla. A continuación, expongo las soluciones más comunes de este caso. Si esto no funciona, refiérase a un manual de Linux/Debian donde explique el funcionamiento de ALSA (Advanced Linux Sound Architecture).

Comenzaremos empleando el siguiente comando:

```
$ aplay -L
```

Este comando lista todos los dispositivos de entrada de audio, si su micrófono no se encuentra en la lista, significa que ALSA no es capaz de identificar su micrófono. Probablemente, no tenga instalados los drivers del micrófono y necesite instalarlos manualmente; esto depende enormemente del micrófono del que se disponga.

Si su micrófono se encuentra listado, pero ve lo que la Figura 8.4.1. señala, significa que PulseAudio no se ha configurado correctamente y se recomienda su reinstalación, dado que PulseAudio al instalarse, genera carpetas dentro de el sistema de archivos de ALSA que modifican su comportamiento. La Figura 8.4.2. señala lo que debe de aparecer si PulseAudio se ha instalado correctamente, y, por tanto, PulseAudio detectará su micrófono por defecto. Cabe destacar que PocketSphinx utiliza PulseAudio y no ALSA, por lo que, si no queremos modificar archivos internos de PocketSphinx para arreglar este error, debemos de utilizarlo. Existe la posibilidad de utilizar ALSA en vez de PulseAudio, pero el rendimiento del sistema decae enormemente, además de que se requiere de un conocimiento avanzado en ALSA para configurarlo correctamente. Para reinstalar PulseAudio:

```
$ sudo apt-get remove pulseaudio
$ sudo apt-get install pulseaudio --fix-missing
$ sudo reboot
```

```
alver@alver-VirtualBox:~$ aplay -L
null
    Discard all samples (playback) or generate zero samples (capture)
pulse
    PulseAudio Sound Server
default:CARD=I82801AAICH
    Intel 82801AA-ICH, Intel 82801AA-ICH
    Default Audio Device
sysdefault:CARD=I82801AAICH
    Intel 82801AA-ICH, Intel 82801AA-ICH
    Default Audio Device
front:CARD=I82801AAICH,DEV=0
    Intel 82801AA-ICH, Intel 82801AA-ICH
    Front speakers
surround21:CARD=I82801AAICH,DEV=0
    Intel 82801AA-ICH, Intel 82801AA-ICH
    2.1 Surround output to Front and Subwoofer speakers
surround40:CARD=I82801AAICH,DEV=0
    Intel 82801AA-ICH, Intel 82801AA-ICH
    4.0 Surround output to Front and Rear speakers
surround41:CARD=I82801AAICH,DEV=0
    Intel 82801AA-ICH, Intel 82801AA-ICH
    4.1 Surround output to Front, Rear and Subwoofer speakers
surround50:CARD=I82801AAICH,DEV=0
```

*Figura 8.4.1.- Null dispositivo prefijado en ALSA.*

```
alver@alver-VirtualBox:~$ aplay -L
default
    Playback/recording through the PulseAudio sound server
null
    Discard all samples (playback) or generate zero samples (capture)
pulse
    PulseAudio Sound Server
sysdefault:CARD=I82801AAICH
    Intel 82801AA-ICH, Intel 82801AA-ICH
    Default Audio Device
front:CARD=I82801AAICH,DEV=0
    Intel 82801AA-ICH, Intel 82801AA-ICH
    Front speakers
surround21:CARD=I82801AAICH,DEV=0
    Intel 82801AA-ICH, Intel 82801AA-ICH
    2.1 Surround output to Front and Subwoofer speakers
surround40:CARD=I82801AAICH,DEV=0
    Intel 82801AA-ICH, Intel 82801AA-ICH
    4.0 Surround output to Front and Rear speakers
surround41:CARD=I82801AAICH,DEV=0
    Intel 82801AA-ICH, Intel 82801AA-ICH
    4.1 Surround output to Front, Rear and Subwoofer speakers
surround50:CARD=I82801AAICH,DEV=0
    Intel 82801AA-ICH, Intel 82801AA-ICH
```

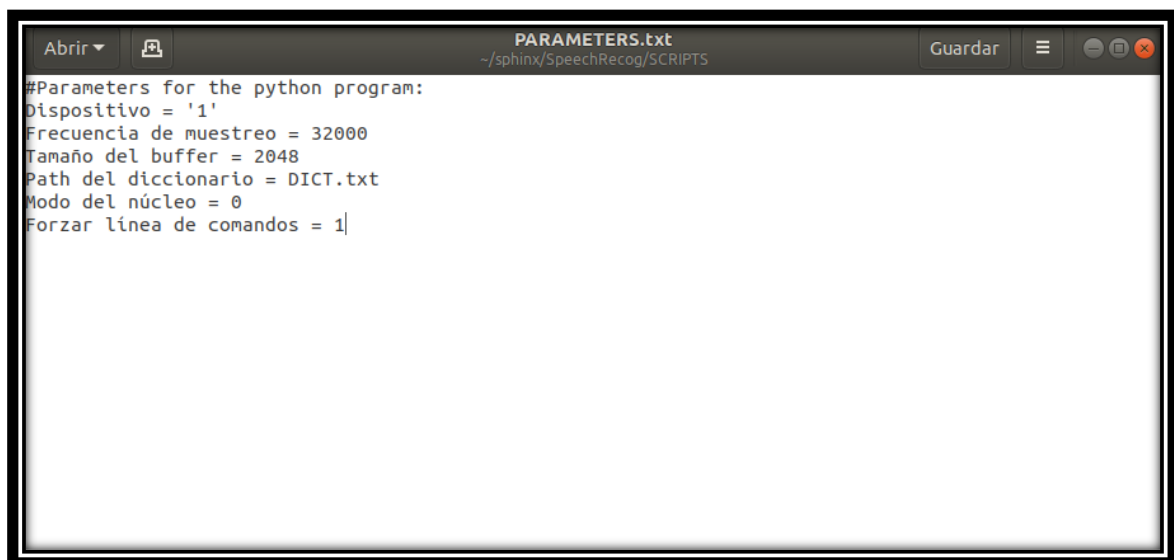
*Figura 8.4.2.- Default PulseAudio en ALSA.*

Si pese a desinstalar e instalar de nuevo PulseAudio sigue experimentando errores se recomienda modificar los siguientes ficheros dentro de los parámetros personales del script, para ver si el micrófono es detectado por PocketSphinx manualmente. Este fichero se encuentra en /home/usuario/sphinx/SpeechRecog/SCRIPTS/PARAMETERS.txt.

Dentro de este fichero habrá que modificar la variable 'Modo del núcleo', poniendo el número a '1' en vez de a '0' como viene por defecto. Importante no modificar los espacios y las líneas en las que se encuentran las variables.

Si usted está utilizando PulseAudio, en la variable 'Dispositivo' debe de buscar un índice, mayor o igual que 0, que identifique dentro de PulseAudio a su micrófono. Si aun así no detecta micrófono alguno, refiérase a la documentación de PulseAudio [18] para intentar arreglar el error.

Si usted está utilizando ALSA y aún no es capaz de detectar el micrófono, en la variable 'Dispositivo' debe de buscar una tarjeta y dispositivo que se refiera a su micrófono de la siguiente manera: 'plughw:CARD,DEVICE' donde CARD se refiere a la tarjeta del controlador de ALSA asociada a su micrófono y DEVICE el dispositivo asociado a esa tarjeta. Si no entiende o no es capaz de detectar el micrófono de esta manera, refiérase a la documentación de ALSA [19].



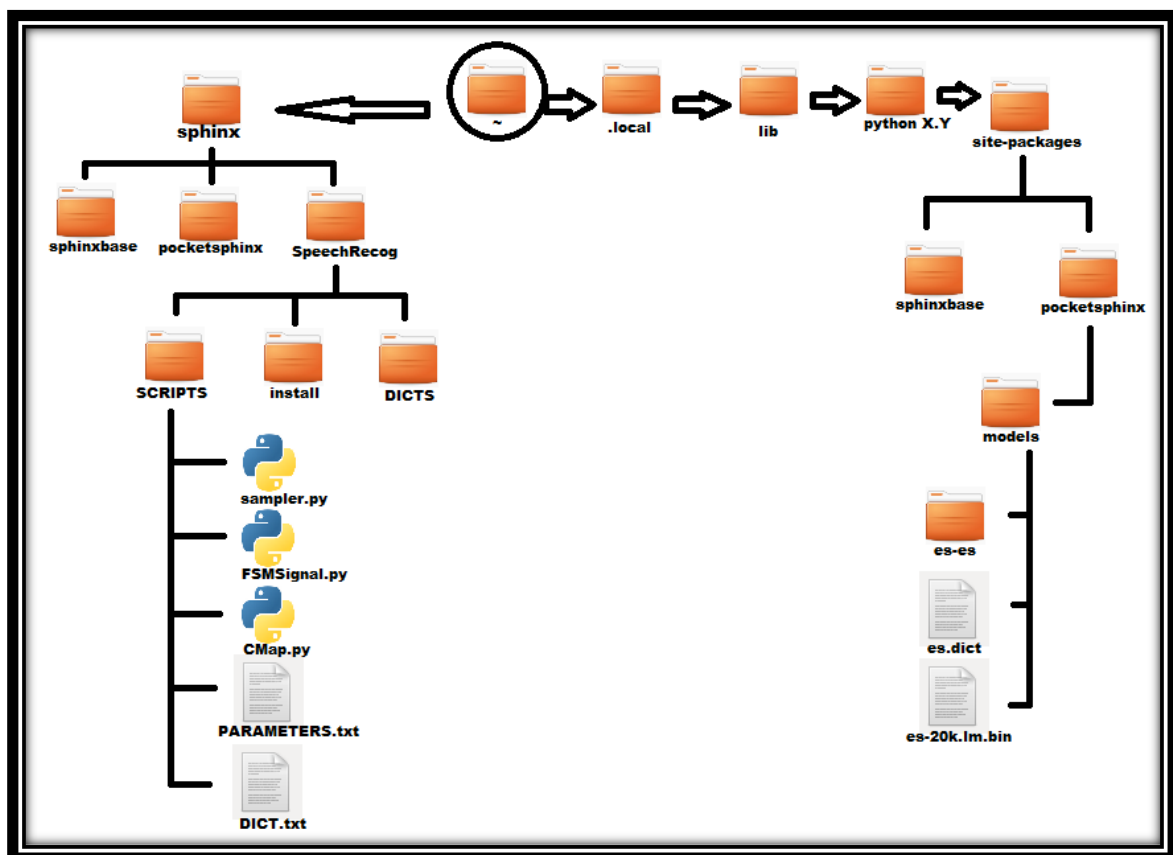
*Figura 8.4.3.- PARAMETERS.txt*

## 9. Sistema de archivos.

El sistema de archivos que compone el programa se representa en la Figura 9.1, un esquema de árbol que representa los ficheros que hacen uso de la característica de reconocimiento de audio. No se entra en detalle a observar las carpetas dedicadas a PocketSphinx ni a SphinxBase, si no únicamente a las que son interesantes modificar, como el modelo de lenguaje y los diccionarios.

Existen dos ramas principales; en `/home/usuario/.local/` y en `/home/usuario/`. La primera rama instala PocketSphinx y ShpinxBase en directorios accesibles y la segunda contiene las carpetas del script que el programador vaya a modificar.

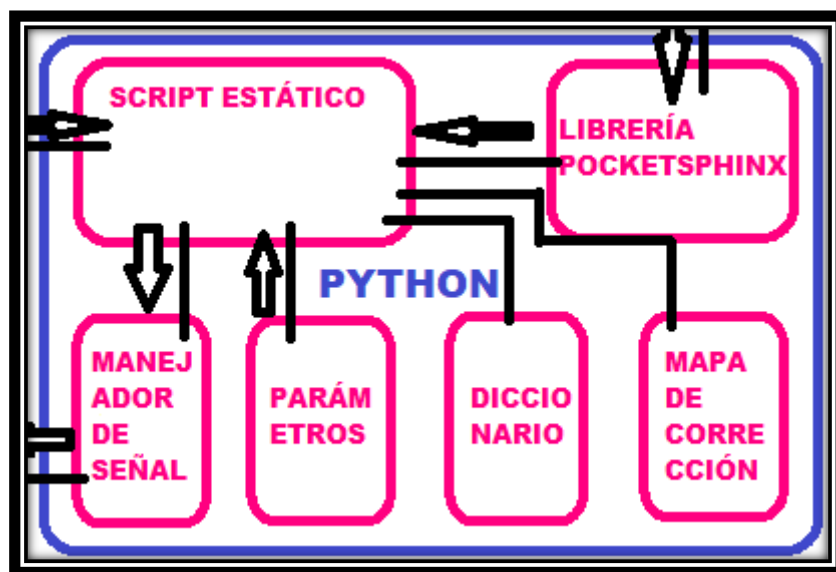
Cabe destacar que no es necesario acceder a las carpetas de sphinxbase ni pocketsphinx si se ha instalado correctamente la característica. Es más, pueden ser eliminadas, pero es aconsejable mantenerlas en el árbol por si necesitamos llamar a la orden de configuración, que se encuentra en las mismas.



*Figura 9.1.- Esquema de ficheros del proyecto.*

En la carpeta de SCRIPTS, dentro de la rama /home/usuario/sphinx/SpeechRecog/SCRIPTS, encontramos los siguientes archivos:

1. PARAMETERS.txt: Es el archivo de configuración de los parámetros, dentro podemos configurar variables como el dispositivo de audio a utilizar, la frecuencia de muestreo, el tamaño del buffer, ruta del diccionario respecto al archivo actual, si queremos que el dispositivo sea detectado automática o no (Modo del núcleo) y si se desea que las señales sean mostradas por salida estándar (línea de comandos) o por GPIO.
2. DICT.txt: Es el diccionario relativo, dado a que existen dos, el diccionario del modelo (2000 palabras en castellano) y el relativo o de usuario, que es el que el usuario modifica con el fin de adaptar el programa a sus necesidades. Las señales se generan a partir de este diccionario, si una palabra de las que se dicen corresponde con las escritas en este diccionario se genera una señal con el identificador de la línea de texto en la que se encuentre la palabra. Si por ejemplo en la línea 27 del diccionario está escrita la palabra: ejemplo, al procesarse la palabra ejemplo se creará una señal con identificador 27. Esto es muy importante tenerlo en cuenta a la hora de crear un manejador de señales.
3. CMap.py: Es el script que contiene la variable de corrección de errores. Este archivo sirve para mejorar el rendimiento de nuestras aplicaciones.
4. FSMSignal.py: Es el script que contiene las funciones de manejador de la señal. Está diseñado para crear una máquina de estados en su interior, dado que se le pasa el parámetro 'estado', el parámetro 'señal' y 'número', siendo el 'estado' el estado asociado a la máquina de estados, 'señal' la señal generada por el reconocimiento de voz y 'número' un buffer de almacenamiento en formato *string*.
5. Sampler.py: Es el script que se ejecuta cuando se llama al comando guardado en .bashrc. Este script se recomienda no modificarlo y es el que maneja todo el contexto del programa, tanto el reconocimiento de audio, como los ficheros y las señales.



*Figura 9.2.- Ficheros de Python.*

## 10. Scripting.

El script que ejecuta el comando almacenado en `.bashr`, es por defecto `sampler.py`, que gestiona todo el sistema.

Dentro de los scripts que contiene el proyecto, donde está el grueso es en archivo `sampler.py`, el nombre indica que ahí es donde se obtienen las muestras. La funcionalidad de este archivo es principalmente lanzar la ejecución en tiempo real de el reconocimiento de audio y traducir las palabras a señales utilizando la función residente en `FSMSignal.py`.

Los scripts que el usuario debe modificar para ajustar sus necesidades a la ejecución del programa son `CMap.py` (para mejorar el rendimiento del programa) y `FSMSignal.py` (para definir un manejo de las señales). Además, debemos configurar el diccionario (`DICT.txt`) para definir que palabras son las que queremos detectar. Por último, modificar el archivo `PARAMETERS.txt` para ajustar parámetros internos como detección del micrófono. Ver Figura 9.2 para ver de manera funcional el sistema.

### 10.1. Script estático.

El script estático o `sampler.py` de encarga de configurar el reconocimiento de audio mediante el archivo `PARAMETERS.txt`, archivo que el usuario puede modificar para ajustar el programa a sus necesidades. Además, el usuario dispone del archivo `DICT.txt`, en el cuál puede escribir palabras, que dentro de este script serán traducidos a señales en función de la línea en la que esté escrita. Dichas señales son redirigidas al manejador de señales, que es definido por el usuario.

Primero, se leen los parámetros de `PARAMETERS.txt` y se almacena en variables en el programa. En cuanto el programa no es capaz de abrir estos archivos devuelve un error y cancela la ejecución, por lo que su presencia es imprescindible para la ejecución del programa.

Al iniciar, se ejecuta una interfaz gráfica por la línea de comandos como muestra la Figura 10.1.1, que describe brevemente lo que está sucediendo en el programa y presenta el mismo. Si el programa no se está ejecutando en una Raspberry, no tendrá instalado por defecto `Rpi.Gpio` y el script identificará que no existe esa librería en el sistema, por lo que todas las señales irán sobre línea de comandos.



Tras ejecutar la interfaz gráfica, el programa intentará abrir el diccionario (DICT.txt), por lo que, si falla en abrirlo, el programa termina. Tras la interfaz, se declaran las variables 'state' (estado de la máquina) y 'number' (buffer en formato string, principalmente pensado por si se desea escribir algún número vía audio, aunque puede servir para otras aplicaciones, que definirá el usuario).

El programa hace uso de la función `__FSMSignal()` (`__FSMSignalRPI()` si estamos en una Raspberry) definido en el archivo `FSMSignal.py`, por lo que su presencia es imprescindible. Si dichas funciones devuelven -1, se llamará a una función que cierra de manera segura el programa. Por defecto, cuando se escucha la palabra 'adiós' el programa se cierra de dicha manera, igual que si por teclado se emplea la interrupción CTRL + Z. Ambas por defecto terminan el programa.

El programa identifica en bucle todas las frases pronunciadas por el usuario. Le quita las tildes y las pasa a minúsculas. Posteriormente, se aplica el mapa de correcciones, y, tras ello, se busca la palabra resultante en el diccionario, traduciendo dicha palabra a una señal en función de la línea en la que se encuentre dentro del diccionario. Esa señal se le pasa a la función `__FSMSignal()` (o `__FSMSignalRPI()`) y el manejador de señales hará lo oportuno con dicha señal.

Cabe destacar que la función de eliminar tildes se ha obtenido de [20]. El script está en el Capítulo 15.2, en el Anexo I.

## 10.2. Manejador de señales.

El manejador de señales se compone de dos funciones:

1. `__FSMSignalRPI()`. Esta función será llamada por el sistema si se detecta que la librería `Rpi.Gpio` está instalada y está diseñado para llamar a las funciones de GPIO de la Raspberry en ella.
2. `__FSMSignal()`. Esta función será llamada por el sistema si se detecta que no hay librería `Rpi.Gpio` instalada o si se define en los parámetros del programa que se desea llamar a esta función siempre.

Por defecto viene escrita una DEMO con unas pocas palabras, vea el Anexo I, Capítulo 15.2 para ver el código de la DEMO, que viene instalada por defecto.

El manejador de señales tiene como objetivo manejar las señales que el sistema le proporciona desde el programa `sampler.py` y actuar como máquina de estados si el usuario lo desea, es por ello por lo que las funciones tienen como entrada las siguientes variables:

1. State: Estado de la máquina de estados del usuario.
2. Signal: Señal detectada por el sistema.
3. Number: Buffer antes de la llamada de la señal.

Y devuelve las siguientes variables:

1. State: Estado de la máquina de estados tras la ejecución de la función, el usuario decide si implementar o no la máquina de estados, pero siempre debe de haber retorno de algún estado. Es necesario devolvérselo al script estático, que es el que llama a la función.
2. Number: Buffer después de la llamada de la señal. Siendo el buffer una variable en la cuál el usuario tiene la posibilidad de almacenar un valor o recurso, es necesario devolvérselo al script estático, que es el que llama a esta función.

**NOTA:** Cabe recalcar que si el estado 'state' retornado por la función es -1, el programa termina. Además, cabe la posibilidad de implementar las acciones asociadas a los comandos en esta función.

### 10.3. Mapa de correcciones.

El mapa de correcciones tiene como objetivo mejorar el rendimiento del sistema. Actúa como un simple corrector, traduciendo las palabras erróneas en palabras correctas. Dentro del programa `CMap.py`, tenemos la variable `HMAP`, que es un Hash Map<sup>3</sup> que traduce nuestras palabras erróneas en palabras correctas. Esto a veces es necesario debido a que PocketSphinx tiene un rendimiento bajo para identificar ciertas palabras. Un simple ejemplo es el siguiente:

Al decir la palabra 'nueve' el reconocimiento de palabras identifica dicha palabra con la palabra 'haré' un 90% de las ocasiones, por lo que en la variable `HMAP` implementaremos la línea:

**'hare' : 'nueve'**

El sistema traducirá las palabras erróneas en palabras correctas. **Esto se podrá hacer siempre y cuando la palabra confundida no corresponda con una de nuestras palabras del diccionario. La aplicación de esta característica es opcional, pero mejora considerablemente el rendimiento del sistema.**

**NOTA:** Las palabras introducidas al Hash Map deben de ser **TODAS** en minúsculas y sin tildes, debido a que el procesado de la palabra ya ha sido efectuado antes de llamada al Hash Map.

---

<sup>3</sup> Hash Map: [https://es.wikipedia.org/wiki/Tabla\\_hash](https://es.wikipedia.org/wiki/Tabla_hash)

## 10.4. Diccionario.

El diccionario personalizado por el usuario es un fichero donde se almacenan las palabras que van a actuar como activadores de señal, es decir, aquí se escribirán las palabras que queremos que se reconozcan a la hora de mandar órdenes al robot.

Es muy importante saber que el número de la línea en la cual escribamos una palabra será igual al identificador de la señal generada al ser reconocida. Por ejemplo, si en el documento (DICT.txt) se escribe la palabra 'ejemplo' en la línea 27, cuando el usuario diga la palabra 'ejemplo', el programa mandará al manejador de señales la señal 27, para que el usuario defina un manejador de señal a su gusto y necesidad.

El formato debe de ser el siguiente:

1. Las palabras **deben** de ir en **minúsculas**, sin tildes.
2. Las palabras no deben de contener espacios.
3. Las palabras deben de estar deparadas por el carácter de salto de línea, es decir, las palabras deben de estar cada una en una línea del archivo.
4. **La señal generada para cada palabra es igual a la línea en la que se encuentra la palabra en este archivo.**
5. No deben de existir huecos entre líneas, es decir, todas las líneas del archivo deben tener **una única palabra**, sin espacios.

Se proporciona un ejemplo a modo de demo, para que el usuario no confunda el formato del diccionario, incluyendo palabras aleatorias con su manejador de señal correspondiente.

Es posible ubicar el diccionario en una ruta diferente, pero esto hay que especificarlo en los parámetros del programa como se verá en el Capítulo 10.5. Por defecto, el fichero DICT.txt, que es nuestro diccionario, viene ubicado en /home/usuario/sphinx/SpeechRecog/SCRIPTS/, ver figura del Anexo II para más información sobre el sistema de archivos o referirse al Capítulo 9.

Si el programa principal no es capaz de encontrar la ruta del diccionario del usuario (DICT.txt), lanzará un mensaje de error y hará imposible la ejecución de la característica, por lo que la correcta definición de la ruta es imprescindible para que el sistema funcione correctamente.

## 10.5. Parámetros del programa.

Existe un archivo llamado PARAMETERS.txt, situado junto a los scripts en Python de las variables del sistema. Este archivo contiene varias líneas que definen dichos parámetros internos y el comportamiento del reconocimiento de palabras. Refiérase al Anexo I para ver el contenido por defecto de este archivo.

El archivo tiene como contenido las siguientes variables:

1. Dispositivo: Es el dispositivo de audio que el programa va a utilizar, sólo se tiene en cuenta si el modo del núcleo está a 1. Refiérase al Capítulo 8.4 para más información acerca del reconocimiento de dispositivos en ALSA y PulseAudio.
2. Frecuencia de muestreo: Es la frecuencia de muestreo del audio, se recomienda dejarlo en 32000 para la Raspberry, que experimentalmente es el que mejor funciona para este modelo. Refiérase al Capítulo 13 para más información sobre el rendimiento en ciertas frecuencias.
3. Tamaño del buffer: Tamaño de almacenamiento del buffer del audio, no influye tan drásticamente en el rendimiento como la frecuencia de muestreo, pero se recomienda para la Raspberry Pi 4 un valor 2048.
4. Path del diccionario: Es la ruta en la cual se puede hallar el archivo DICT.txt, si se cambia de ruta hay que especificarlo en este parámetro, de lo contrario el programa no podrá ejecutarse correctamente. Se recomienda mantener el diccionario la ruta predeterminada, pero en caso de que el usuario desee cambiarlo, es posible gracias a este parámetro.
5. Modo del núcleo: Es el modo de reconocimiento del dispositivo de entrada de audio en el núcleo del sistema. Si vale 0, el sistema operativo buscará el dispositivo de audio por defecto, es decir, el primer dispositivo listado por el comando: \$ aplay -L en el intérprete de comandos de Linux. Refiérase al Capítulo 8.4 para más información acerca del reconocimiento de dispositivos en ALSA y PulseAudio. Si vale 1, el núcleo buscará el dispositivo especificado en el primer parámetro de este archivo.
6. Forzar línea de comandos: Fuerza al programa a usar la función \_\_FSMSignal() en lugar de la función \_\_FSMSignalRPI() en el script FSMSignal.py, puede llegar a ser útil para el usuario si este desea manejar dos funciones en vez de una en el manejador de señales. Además, poner a 0 esta característica, deja de imprimir las palabras detectadas por pantalla.
7. Eco: Activa o desactiva el eco de reconocimiento de palabras, es decir, repite lo que el usuario ha dicho en modo de texto. El valor 0 desactiva esta característica y el valor 1 la activa.

## 11. Manual de usuario.

El sistema de reconocimiento de voz se compone de una parte estática y otra que el usuario debe modificar. En este capítulo sólo nos centraremos en las partes que el usuario debe de modificar. Tras instalar el programa (Capítulo 8) el usuario puede pasar a modificar los ficheros para configurar el robot en función de sus necesidades.

Lo primero que debe de hacer el usuario es definir las palabras clave que quiere que el sistema identifique. En el caso de la DEMO proporcionada las palabras se encuentran en el Anexo I (16.2 DICT.txt), que implementa un pequeño conjunto de palabras añadidas en el archivo DICT.txt, **que es el diccionario relativo o diccionario de usuario, no confundir con el diccionario del modelo (que contiene 20000 palabras en castellano)**. Al añadir palabras al diccionario, es importante recordar el orden de estas, dado que cada palabra introducida en el diccionario se le asocia un número de señal, en función de la línea del diccionario en la que se encuentre. La palabra posicionada en la primera línea del diccionario tendrá el número de señal **interna**<sup>4</sup> 1, la segunda 2, la tercera 3... así sucesivamente sin un límite teórico.

El sistema reconocerá automáticamente dichas palabras, escritas en el diccionario, por lo que genera una señal correspondiente a la palabra del diccionario según lo expuesto en el párrafo anterior y pasará dicha señal al manejador de señales. Cabe destacar, que al ser una asociación el orden es indiferente, pero debe de tenerse en cuenta al programar el fichero FSMSignal.py.

El manejador de señales, situado en el archivo FSMSignal.py consta de 3 partes:

1. Señal: Número de la palabra detectada. **Esta es la señal que hemos denominado señal interna del programa.**
2. Estado: Es una variable que permite la generación de una máquina de estados. Si el usuario devuelve el estado -1, el programa terminará.
3. Número: Buffer en forma de cadena de caracteres, principalmente diseñado para almacenar números, pero puede almacenar cualquier tipo de cadena.

Además, hay dos funciones, una diseñada para el GPIO de Raspberry y otra diseñada para el terminal de Linux o señales dentro de programas o sistema operativo. El usuario importará las librerías pertinentes es este archivo con objetivo de mandar señales por dentro del sistema operativo o las librerías para el GPIO de Raspberry. El sistema por defecto detecta automáticamente si se disponen de las librerías GPIO de Raspberry. El usuario definirá la función `__FSMSignal()` / `__FSMSignalGPIO()` del archivo FSMSignal.py para crear una reacción a su gusto de las palabras detectadas. En la Figura 11 se muestra un esquema simple de los archivos y funciones básicas del programa.

---

<sup>4</sup> Existen dos tipos de señales: internas (producidas por el script estático al reconocer una palabra en el diccionario) y del sistema operativo (producidas por el manejador de señal hacia el sistema operativo al reconocer una acción asociada a una señal interna).

### 11.1. Puesta en marcha del programa.

Una vez configurados los parámetros e instalado el programa (referirse al capítulo 8, instalación del modelo para realizar la instalación) podemos realizar la puesta en marcha. Es recomendable que el usuario modifique los parámetros anteriormente mencionados (diccionario y manejador de señal) para que el uso que le dé sea personalizado. Sin embargo, es posible poner en marcha la DEMO del programa sin modificar dichos archivos.

Para ello, podemos lanzar desde cualquier directorio del usuario el siguiente comando:

```
$ Lola
```

Con este comando ejecutaremos la puesta en marcha del reconocimiento de voz, que correrá en paralelo a los programas que vayamos a estar ejecutando en el robot social. En caso de que el sistema operativo no reconozca el comando, puede arreglar el error con el siguiente:

```
$ source /.bashrc
```

```
$ Lola
```

Si aun así el programa sigue sin ejecutarse puede acceder al archivo 'sampler.py' y ejecutarlo manualmente con el intérprete de Python, refiérase al Anexo II para localizar los archivos del sistema que desee.

Una vez realizada la ejecución del programa, el manejador de señales se ejecutará en el programa y realizará las acciones pertinentes. Por defecto, la DEMO está diseñada para una interfaz en línea de comandos. Esto, como ya hemos mencionado en otros capítulos, podemos desactivarlo en el archivo 'parameters.txt'.

## 11.2. Ejecución de DEMO.

En este programa, se dispone de una DEMO para poder probar la característica de reconocimiento de audio. Para ello, debemos poner en marcha el programa como muestra el capítulo anterior. Una vez puesto en marcha el programa y sin haber recibido ningún tipo de error, como puede ser la detección de un micrófono, podemos realizar pruebas sobre la misma.

Comencemos dejando cargar el modelo hasta que se muestre el mensaje de:

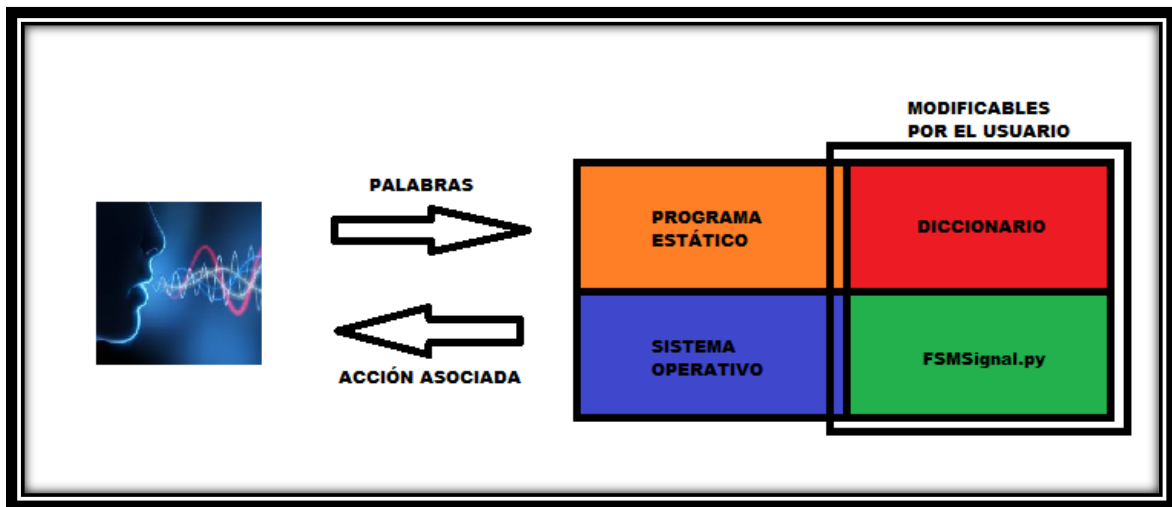
`Cancele la ejecucion en cualquier momento con ctrl+Z.`

Posteriormente, el programa hará eco de todas las palabras que reconozca, es probable que interprete situaciones de ruido como ciertas palabras aleatorias. Tenga en cuenta que el rendimiento del programa es de aproximadamente un 70%, por lo que hay 30% de posibilidades de que la palabra que usted pronuncie no sea correctamente detectada en el rango de 0 a 4 metros de distancia del micrófono.

Pruebe a pronunciar las siguientes palabras, en caso de no ser reconocidas, pruebe a pronunciarlas varias veces:

- Hola.  
Respuesta: Hola, buenas, soy Lola.
- Uno.  
Respuesta: Numero: 1.
- Dos.  
Respuesta: Numero: 12.
- Nueve:  
Respuesta: Numero: 129.
- Llama.  
Respuesta: Llamando a : 129.
- Richard.  
Respuesta: [Numero de Richard].
- Vámonos a la:  
Respuesta interna, genera un estado en la máquina de estados.
- Cocina:  
Respuesta: Me estoy moviendo a la cocina.
- Adiós.  
(Fin de ejecución del programa).

Decir números consecutivos provoca que se almacenen en el buffer, decir las palabras 'no' o 'cancelar' eliminan el número del buffer. Probados estos elementos de la DEMO, podemos concluir que el sistema funciona correctamente si las respuestas fueron las esperadas. El usuario puede comprobar que palabras han sido efectuadas.



*Figura 11.2.- Esquema usuario – programa. [21].*

## 12. Manual del programador.

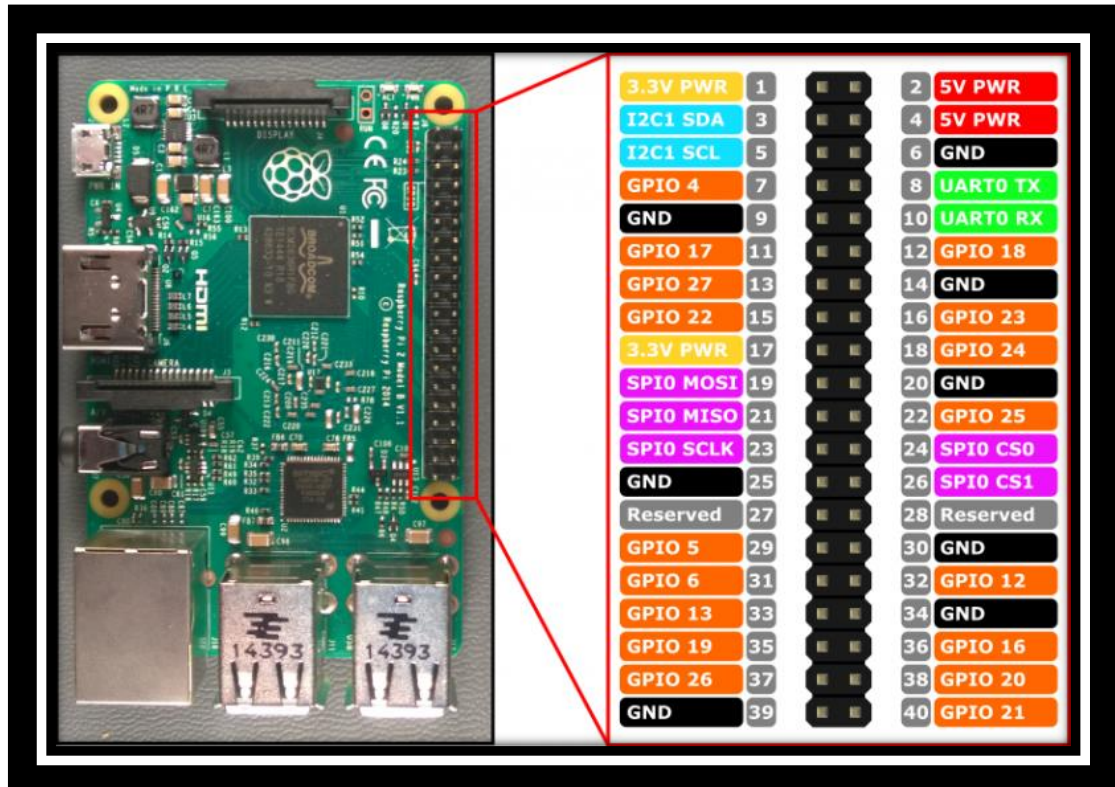
Para modificar el sistema acorde a otras funciones que no sean las funciones básicas del usuario definidas en el Capítulo 11, se proporciona un manual más avanzado. La definición de parámetros del sistema se encuentra en el archivo PARAMETERS.txt, que incluye un repertorio de opciones acorde a las necesidades del usuario. En el capítulo 10.5 ya se ha explicado el funcionamiento de este archivo. Refiérase a él para más información.

### 12.1. Manejador de señales en Raspberry.

La forma más sencilla de manejar las señales en una Raspberry es mediante su interfaz en Python de puertos de entrada y salida genéricos. En la Figura 12.1 se muestra una imagen sencilla de los puertos GPIO (Generic Port Input/Output) de una Raspberry. Estos puertos mandan señales en forma de voltaje físico por los pines de la placa, por lo que es ideal para controlar otros dispositivos como controladores de potencia de motores como, por ejemplo, basados en transistores o relés. Normalmente, los sistemas operativos enfocados a Raspberry tienen estas librerías instaladas por defecto. De lo contrario puede instalarlo con el siguiente comando: [22]



\$ `pip install RPi.GPIO`



*Figura 12.1 Raspberry Pi (GPIO)*

## 12.2. Manejador de señales vía comandos.

Mediante la librería SIGNAL de Python podemos enviar señales a programas que se estén ejecutando en paralelo al nuestro, por lo que podremos enviar señales a otros programas por medio del sistema operativo cuando escuchemos la palabra clave. Evidentemente, podemos juntar manejador de GPIO junto con un manejador de sistema de señales del sistema operativo. Esto es útil para programas que manejan por medio de USB, internet u otras formas de conexión; el robot. [23]

### 12.3. Corrección de errores.

Para mejorar el rendimiento del sistema podemos usar un mapa de corrección de errores. Dado que el programa puede malinterpretar palabras y producir el mismo error en la misma palabra varias veces, una solución para programas de pocas palabras consiste en usar un corrector, por lo que la palabra fallo (palabra reconocida que no corresponde con la palabra pronunciada) se traduce a la palabra pronunciada y el sistema la reconocerá correctamente. Esto puede usarse si la palabra fallo no pertenece al conjunto de las palabras del diccionario. Si se desea una mayor eficiencia para un diccionario muy extenso, hay que buscar otro modelo más eficiente, lo que compone una complejidad mucho mayor o realizar un diccionario personalizado, que requiere un trabajo de modificación de un fichero. Refiérase al siguiente capítulo para dicho caso.

Para la corrección de estos errores, hay que acceder al programa CMap.py, que contiene una variable llamada HMAP, el sistema utilizará esta variable para corregir errores; con el siguiente formato en forma de lista en Python:

PALABRA FALLO : PALABRA CORRECTA

El número de palabras es ilimitado, dado que la complejidad de un Hash Map es lineal ( $O(1)$  ( $O(n)$  en el peor de los casos, pero muy improbable y no va a suceder)) [24], por lo que podemos añadir todas las palabras que queramos dentro de los límites de nuestro diccionario.

Refiérase al Capítulo 10.3 para una explicación de esta característica. Refiérase el Capítulo 16.4 para ver el ejemplo proporcionado a modo de DEMO, junto a las palabras proporcionadas en el diccionario, Capítulo 16.5.

### 12.4. Diccionario personalizado.

Si el usuario desea personalizar el diccionario localizado en la carpeta de PocketShpinx, puede hacerlo. Si el usuario desea un rendimiento muy alto del sistema, puede editar el fichero es.dict para dejar únicamente las palabras que el usuario quiera reconocer. La ruta de este archivo es la siguiente:

**/home/usuario/local/lib/python2.7/site-packages/pocketsphinx/models/es.dict**

Para modificar dicho archivo para que el sistema reconozca únicamente las palabras que desee, debe modificarlo. Se recomienda mantener en segundo plano el diccionario antiguo, modificando el nombre de este a `es.dict.old`:

```
$ cd /home/usuario/local/lib/python2.7/site-packages/pocketsphinx/models/es.dict
$ sudo cp es.dict es.dict.new
$ sudo mv es.dict es.dict.old
$ sudo mv es.dict.new es.dict
```

Ahora modificamos el fichero `es.dict` nuevo y borramos todas las palabras que no vayamos a utilizar en el diccionario. Esto es muy laborioso si el diccionario es muy amplio, pero mejora el rendimiento del programa, dado que no se generan palabras erróneas, esto es una solución alternativa al mapa de correcciones, que es una solución más rudimentaria y menos laboriosa.

**NOTA:** Si se añade una palabra que no existe al diccionario, no será reconocida, debido a que los parámetros están diseñados para el diccionario primario (`es.dict.old`). Por tanto, únicamente podemos borrar palabras de este.

Si se deseara crear un diccionario personalizado con sus parámetros asociados para una línea futura de trabajo o investigación, dispone de un tutorial en la página oficial de CMU Sphinx [25]. Refiérase al Anexo III para obtener un enlace vía online a dicho tutorial.

Cabe la posibilidad de activar diferentes diccionarios de forma dinámica, lo que implica una recarga del modelo en el script estático, que debería de modificarse en ese caso, (que como veremos en el Capítulo 13.2 Pruebas de velocidad, implican alrededor de 10 segundos por recarga), esto elimina la posibilidad de un reconocimiento fluido de audio si se utilizan varios diccionarios asignados de forma dinámica, aunque para un reconocimiento no exija un nivel de fluidez alto, puede llegar a ser una opción.

## 13. Condiciones de funcionamiento.

En esta Capítulo se reúnen experimentos y condiciones de funcionamiento del sistema. En el Capítulo 13.1 se verán limitaciones físicas, mientras que en el Capítulo 13.2 se verán limitaciones de cómputo o software del sistema y de la característica en general. Para los experimentos se han utilizado las especificaciones de la Tabla 13:

CARACTERÍSTICA	TABLA 13. ESPECIFICACIONES DE PRUEBAS					
Palabras	Hola	Cocina	Tres	Nueve	Dos	Sí
Placa	Raspberry Pi 4					
Micrófono	Plantronics HW520					
Distancia	1 metro					
Ruido	-30 dB (110Hz)					
CMAP	Activado					
Diccionario	Estándar					

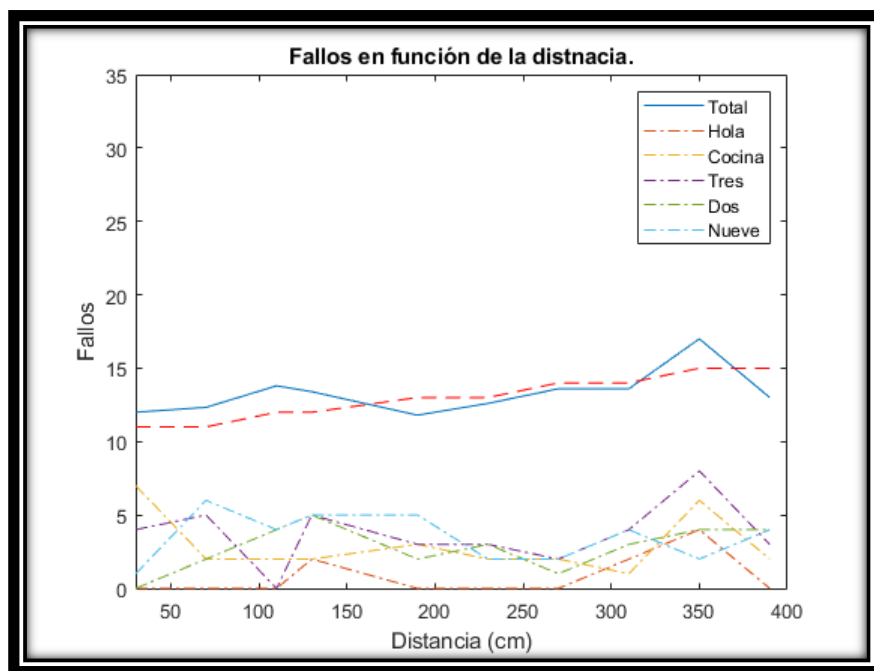
Se utilizará un micrófono auricular Plantronics HW520, una Raspberry Pi 4, el repertorio de 5 palabras: [Hola, Cocina, Sí, Tres, Nueve, Dos] a un metro de distancia del micrófono con la corrección de mapa activada. El nivel de ruido se encuentra 30 [25-35] dB por debajo del audio proporcionado por el locutor, dicho ruido se genera por el ventilador de la Raspberry Pi 4 en 110Hz (se ha medido con una aplicación de análisis espectral de audio). El diccionario del modelo no se ha modificado.

Se procederán a modificar las variables de las pruebas a raíz de la Tabla 13. Se producirán 10 iteraciones de cada palabra y se contará el número de fallos en cada una de ellas, para conocer el rendimiento del sistema en función de las componentes físicas.

Además, se dispone de un 'live script' de Matlab para la representación de gráficas asociadas a los datos de las pruebas.

### 13.1. Condiciones de distancia y ruido.

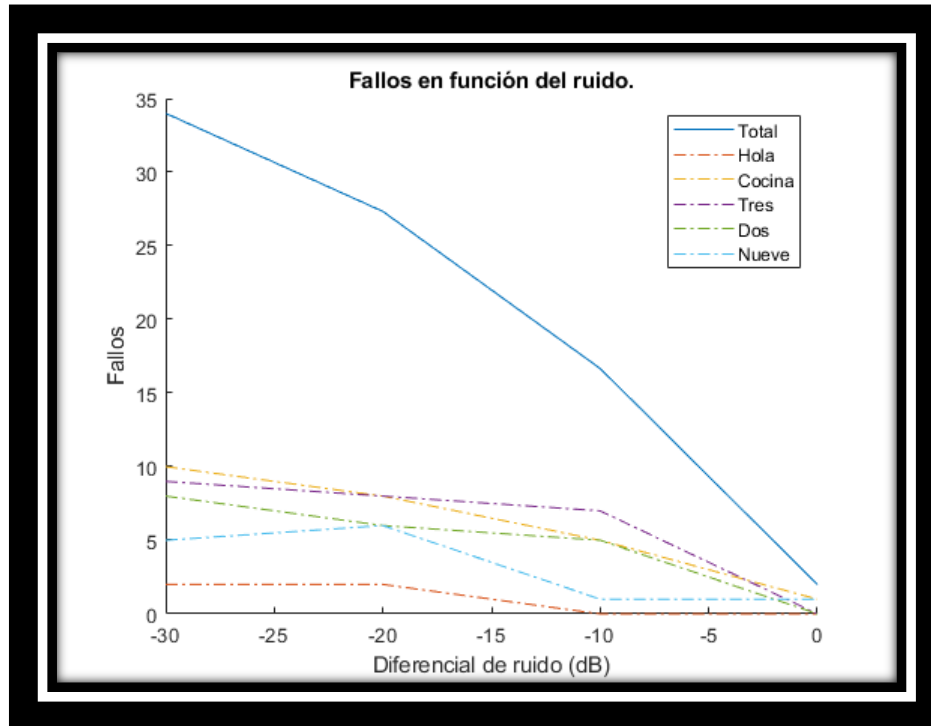
Dado que el tipo de placa no influye en el rendimiento del programa, se probará todo sobre una Raspberry Pi 4. Variando los diferentes factores de locución (distancia, ruido y locutor). En la figura 13.1.1 podemos observar cómo varía en número de fallos con la distancia, mientras que en la Figura 13.1.2 podemos ver cómo varía el número de fallos con el ruido de la sala.



*Figura 13.1.1.- Fallos en función de la distancia.*

En la figura se muestra que, aunque el porcentaje de fallos aumenta con la distancia, se obtiene un rendimiento aceptable en el rango de 0 a 4 metros de distancia. Por lo que una persona que esté lo suficientemente cerca del robot, podrá comunicarse con él con una eficiencia aceptable.

El rendimiento medio del sistema en función de la distancia es de un **72.6%**, lo que implica que, de cada 100 palabras, no detectará alrededor de 28 de media, esto con el mapa de corrección activado. Cada metro aumenta la media de fallos en 1 de cada 10, es decir, el rendimiento decae un 10% por metro, lo que puede resultar molesto para robots que se comuniquen con el usuario a más del rango definido para el experimento. Dado que este proyecto está enfocado a robots que se muevan por habitaciones o sitios cerrados, 4 metros es un buen límite.



*Figura 13.1.2.- Fallos en función del nivel de ruido.*

Como muestra la figura, cuando la diferencia entre el ruido y la voz es nula, se obtiene un número de fallos muy grande, el rendimiento del sistema puede considerarse desastroso (32%), mientras que a medida que el ruido decrece, nos aproximamos a un modelo con un rendimiento del 85% aproximadamente a distancia de un metro.

Esta prueba se ha realizado según las especificaciones de la Tabla 13, variando el nivel de ruido con un generador de ruido blanco gaussiano, que genera ruido aleatorio en todas las frecuencias.

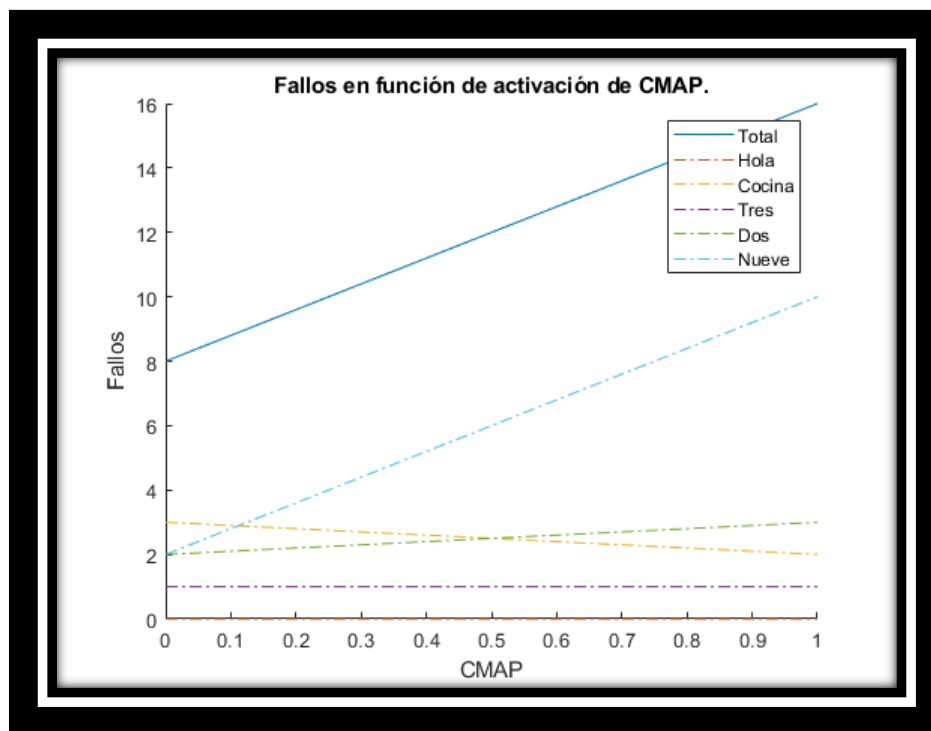
Se define el diferencial de ruido como:

$$DR = NV - NR - 30dB$$

- NV: Nivel de decibelios de la voz.
- NR: Nivel de decibelios del ruido.
- 30dB: Diferencia de ruido en silencio en una sala estándar.

Se le resta 30dB para representar los datos de manera limpia, dejando el diferencial de ruido en 0 para el silencio en la sala. Este valor deja el eje centrado en el silencio.

Si desactivamos la corrección con el Hash Map del archivo CMap.py, podemos observar lo que nos muestra la Figura 13.1.3, que, para ciertas palabras, es imprescindible el uso del mapa, dado que el sistema reconoce para el locutor una palabra errónea un amplio porcentaje del tiempo. Por ejemplo, durante las pruebas, la palabra nueve fue corregida desde las palabras 'haré' o 'muere', que no están incluidas en el diccionario, y, por tanto, pueden ser corregidas. El rendimiento con la capa de correcciones **aumenta en un 16%** si se utiliza, permitiendo el reconocimiento de palabras que, sin él, no serían posibles reconocer.



*Figura 13.1.3.- Comparación de fallos con CMAP.*

En cuanto al cambio de locutor, no influye al rendimiento del sistema en general, aunque sí puede influir en palabras específicas. Se probó un cambio de locutor y la prueba demostró que el reconocimiento de ciertas palabras era ligeramente mejor que otras, mientras que palabras que antes no daban problemas para un locutor ahora empiezan a darlos. Por ello, se recomienda únicamente que el mapa de correcciones esté adaptado a cada usuario del robot, mejorando así el rendimiento en general para todos los locutores. Si no se hiciese, el rendimiento podría decaer hasta un 16%.

## 13.2. Pruebas de velocidad.

Dependiendo del dispositivo que esté ejecutando la característica de reconocimiento de audio, los tiempos de espera entre que el programa carga los modelos o los tiempos que tarda el programa en procesar el audio pueden aumentar. En la Tabla 13.2 podemos ver una comparación de las velocidades en función del dispositivo que ejecute el programa.

DISPOSITIVO	TIEMPO DE CARGA	TIEMPO DE PROCESADO
RASPBERRY PI 3 (ms)	18330	2420
RASPBERRY PI 4 (ms)	10360	1990
INTEL CORE i7 (VBM) (ms)	7620	920

*Tabla 13.2.- Tiempos de procesado y carga del programa.*

El sistema es funcional y rinde en todos los dispositivos en los que se ha probado, aunque si deseamos un reconocimiento instantáneo, la versión de la placa debe de ser superior, y por tanto, más eficiente.

## 14. Conclusión personal del proyecto.

En cuanto a reflexión personal, creo que este proyecto puede llegar a ser muy útil para un gran número de robots basados en Raspberry Pi, dado que, gracias al instalador, no se requiere de mucho trabajo para configurar la placa ni tampoco se requiere de mucho trabajo para crear las funciones y añadir palabras en el diccionario. Si se sabe lo que hay que hacer, el sistema puede estar instalado en 20 minutos (proceso automático) y configurado en otros 20, escribiendo unas pocas líneas de código. Fácilmente en menos de una hora podrías tener funcional el reconocimiento del audio y manejo de señales, por lo que estoy orgulloso de eso. Por ejemplo, alguien que maneje un robot por comandos o por un programa en Python podría trasladar su proyecto a voz en una hora con un único ejecutable, dando los comandos por voz en vez de teclado o botones.

Dado que el proyecto lo he enfocado a la placa en sí (Raspberry), la cantidad de proyectos que pueden acoplar este proyecto al suyo son muchísimos, dando la posibilidad a mucha gente de tener esta característica ya no en un robot, si no en un proyecto genérico en Raspberry Pi.



Esta placa a la que está enfocada el proyecto presume de tener más de 12.5 millones de unidades vendidas en 2017, más todos los proyectos que se hayan podido desarrollar gracias a ella. En la actualidad, se han vendido más de 30 millones de unidades (finales de 2019) y se estima que con el lanzamiento de la Raspberry 4, esta marca ha aumentado como poco a 35 millones.

Además, este proyecto pese a ser muy atractivo para una Raspberry, también se puede ejecutar en servidores, ordenadores convencionales y otro tipo de placas mientras tengan instalados el sistema operativo de Linux, aunque como ya he dicho queda mucho más atractivo en Raspberry.

Otra cosa que me ha sorprendido es el rendimiento del programa, que a priori estimaba que más de la mitad de las palabras no las iba a reconocer. Sin embargo, si se pronuncia lo suficientemente bien y no se habla demasiado rápido el rendimiento supera el 70%, lo que me ha sorprendido a la hora de hacer las pruebas. Otro dato que me ha sorprendido es que el rendimiento no ha decaído exageradamente cuando el locutor se aleja 4 metros del micrófono, lo que es un plus para el proyecto.

Una cosa que no me ha gustado mucho es tener que utilizar una solución tan rudimentaria como el mapa de correcciones para corregir errores del modelo instalado. Pero como es una solución opcional, me parece un buen aporte, pudiendo mejorar el rendimiento un 16% gracias a él. El usuario decide si usarlo o no, dado que puede dejarlo vacío.

La velocidad de reconocimiento de audio me parece bastante aceptable para reconocimiento de audio en tiempo real, dado que de media en una Raspberry Pi 4 se tardan 2 segundos en procesar la señal desde que se pronuncia la palabra clave. Si se usase un modelo online, debido a la latencia de la conexión, los 2 segundos serían una velocidad más que aceptable para un modelo de reconocimiento de audio.

Otro punto negativo del proyecto es que el usuario debe pronunciar bien las palabras clave a detectar y con una velocidad moderada, por lo que, si el usuario habla con fluidez y pronuncia una frase larga, el modelo puede fallar bastante. Esto es normal, dado que hoy en día no existe una tecnología que reconozca el audio con fluidez. Sólo hace falta observar a Google y Amazon, con los dispositivos que lanzan al mercado. Hoy en día esos dispositivos fallan a la hora de reconocer una frase entera pronunciada con fluidez, por lo tanto, que este proyecto tenga ese punto negativo es bastante aceptable.

## 15. Referencias.

- [1] A. Zhang. (2017, Dic 5). SpeechRecognition · PyPi. [Online]. Available: <https://pypi.org/project/SpeechRecognition/>
- [2] Wikipedia. (2020, Nov 3). Speech Recognition. [Online]. Available: [https://en.wikipedia.org/wiki/Speech\\_recognition#Pre-1970](https://en.wikipedia.org/wiki/Speech_recognition#Pre-1970)
- [3] Wikipedia. (2020, Nov 3). DeepMind. [Online]. Available: <https://es.wikipedia.org/wiki/DeepMind>
- [4] Matthew J. Beal, Zoubin Ghahramani, Carl Edward Rasmussen. The Infinite Hidden Markov Model. (2002). [Online]. Available: <http://papers.nips.cc/paper/1956-the-infinite-hidden-markov-model.pdf>
- [5] Wikipedia. (2020, Nov 3). CMU Shpinx. [Online]. Available: <https://es.wikipedia.org/wiki/DeepMind> [https://es.wikipedia.org/wiki/CMU\\_Sphinx](https://es.wikipedia.org/wiki/CMU_Sphinx)
- [6] CMUSphinx. Figura 6.2. [Online]. Available: <https://steemhunt.com/@amr008/cmu-sphinx-open-source-speech-recognition-toolkit>
- [7] Imagen de micrófono USB. [Online]. Available: <https://www.kubii.es/teclados-y-usb-periferico/1936-microfono-usb-para-raspberry-pi-kubii-3272496007772.html>
- [8] Imagen de Raspberry Pi 4. [Online]. Available: <https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-4-73e5afbcd54b>
- [9] Motor de 12 voltios. [Online]. Available: <https://www.transmotec.es/product/rf-500tb-12560-mv/>
- [10] Tarjeta microSD. [Online]. Available: <https://www.worten.es/productos/moviles-smartphones/accesorios/tarjetas-de-memoria-moviles/tarjeta-de-memoria-micro-sdxc-128gb-sandisk-ultra-adaptador-sd-6328763>
- [11] Raspberry Pi Foundation. Raspbian Buster 64 bits. [Online]. Available: [https://downloads.raspberrypi.org/raspios\\_arm64/images/raspios\\_arm64-2020-05-28/2020-05-27-raspios-buster-arm64.zip](https://downloads.raspberrypi.org/raspios_arm64/images/raspios_arm64-2020-05-28/2020-05-27-raspios-buster-arm64.zip)
- [12] Raspberry Pi Foundation. Raspberry Pi Imager. [Online]. Available: [https://downloads.raspberrypi.org/imager/imager\\_1.4.exe](https://downloads.raspberrypi.org/imager/imager_1.4.exe)
- [13] Alberto Palomo Alonso, CMU Sphinx. Repositorio GIT del Proyecto. [Online]. Available: <https://github.com/iTzAlver/SpeechRecog>
- [14] CMU Sphinx. Repositorio SphinxBase. [Online]. Available: <https://github.com/cmuspinx/sphinxbase>

- [15] CMU Sphinx. Repositorio PocketSphinx. [Online]. Available:  
<https://github.com/cmusphinx/pocketsphinx>
- [16] Keryx. Tutorial para instalación de paquetes offline. [Online]. Available:  
<http://beatofthegEEK.com/2011/07/keryx-tutorial-download-software-for.html>
- [17] Keryx. (Nov, 5). Keryx project. [Online]. Available:  
<https://launchpad.net/keryxproject>
- [18] PulseAudio. (Nov, 5). PulseAudio documentation. [Online]. Available:  
<https://www.freedesktop.org/wiki/Software/PulseAudio/Documentation/>
- [19] ALSA. (Nov, 5). ALSA documentation. [Online]. Available:  
<https://www.alsa-project.org/alsa-doc/alsa-lib/>
- [20] Victorino. (Jan, 2017). Script de eliminación de tildes. [Online]. Available:  
<https://gist.github.com/victorono/7633010>
- [21] Imágen voz artificial. [Online]. Available:  
<https://ainews4u.com/can-ais-capability-to-copy-human-voice-be-misused/>
- [22] Croston. (Jul 21,2019). RPiGPIO [Online]. Available:  
<https://pypi.org/project/RPi.GPIO/>
- [23] Python. (Jul 21,2019). SIGNAL library. [Online]. Available:  
<https://docs.python.org/3/library/signal.html>
- [24] StackOverflow. (Nov, 2020) HashMap complexity Python. [Online]. Available:  
<https://stackoverflow.com/questions/1963507/time-complexity-of-accessing-a-python-dict#:~:text=6%20Answers&text=See%20Time%20Complexity,in%20a%20lot%20of%20collisions.>
- [25] CMU Sphinx. (Nov, 2020) How to build a custom model. [Online].  
<https://cmusphinx.github.io/wiki/tutoriallm/>
- [26] Wikipedia. (2020, Nov 3). Red Neuronal Artificial. [Online]. Available:  
[https://es.wikipedia.org/wiki/Red\\_neuronal\\_artificial](https://es.wikipedia.org/wiki/Red_neuronal_artificial)
- [27] Esquema NNM. [Online]. Available:  
<https://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they>

- [28] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, Andrew Y. Ng. Deep Speech: Scaling up end-to-end speech recognition. (17 Dec 2014). [Online]. Available: <https://arxiv.org/abs/1412.5567>
- [29] Daniel Povey. KALDI. [Online]. Available: <https://kaldi-asr.org/doc/index.html>
- [30] NOETIC. ROS NOETIC. [Online]. Available: <http://wiki.ros.org/noetic>

## 16. Anexo I. Scripts.

### 16.1. Instalador del programa en bash.

```
#!/bin/bash
sudo echo Instalador de Lola.
echo Intalando dependencias...
sudo apt-get install -qq python python-dev python-pip build-essential swig git libpulse-
dev libasound2-dev autoconf libtool automake bison
sudo apt-get -y install pulseaudio
echo Dependencias instaladas.
echo Creando carpeta para Sphinx...
cd ~
mkdir sphinx
cd sphinx
git clone https://github.com/cmusphinx/sphinxbase
git clone https://github.com/cmusphinx/pocketsphinx
cd sphinxbase
./autogen.sh
./configure
make
sudo make install
cd ..
cd pocketsphinx
./autogen.sh
./configure
make
sudo make install
cd ..
echo Sphinxbase y pocketsphinx instalados.
pip install --upgrade pip setuptools wheel
pip install --upgrade pocketsphinx
echo Librerías python instaladas.
git clone https://github.com/iTzAlver/SpeechRecog
cd SpeechRecog
cd DICTS
tar xzf cmusphinx-es-5.2.tar.gz
rm cmusphinx-es-5.2.tar.gz
gunzip es-20k.lm.gz
mv es-20k.lm ~/.local/lib/python2.7/site-packages/pocketsphinx/model/es-20k.lm.bin
mv es.dict ~/.local/lib/python2.7/site-packages/pocketsphinx/model/es.dict
cd cmusphinx-es-5.2
cd model_parameters
mv voxforge_es_sphinx.cd_ptm_4000 ~/.local/lib/python2.7/site-
packages/pocketsphinx/model/es-es
echo Modelo español copiado correctamente.
echo Instalando comando Lola:
cat ~/sphinx/SpeechRecog/install/add2bashrc.txt >> ~/.bashrc
echo Actualizando:
echo Borrando residuos...
rm -r ~/sphinx/SpeechRecog/install
echo Cambiando variables...
sleep 5
export LD_LIBRARY_PATH=/usr/local/lib
sleep 10
source ~/.bashrc
echo Terminado.
echo
echo Instalación completada, ejecute el comando: Lola para ejecutar el programa, si el
sistema operativo no reconoce el comando ejecute la siguiente acción:
echo source ~/.bashrc
```

## 16.2.Sampler.py (Script estático).

```
#-----  
--  
#  
#   @author      Alberto Palomo Alonso  
#   @date        28/10/2020  
#   @file        .py  
#  
#-----  
--  
#-----  
--  
#  
#   @libraries  
#  
#-----  
--  
try:  
    import os  
    import sys  
    import signal  
    import unicodedata  
    import re  
except:  
    print("Error al importar librerías del sistema.")  
    sys.exit(0)  
try:  
    from pocketsphinx import LiveSpeech, get_model_path  
except:  
    print("Error al importar librerías de pocketsphinx.")  
    sys.exit(0)  
  
raspi = False  
try:  
    import RPi.GPIO as GPIO  
    raspi = True  
except:  
    print("Warning: No hay interfaz GPIO, se senaliza por consola.")  
try:  
    import FSMSignal as FSMS  
    import CMap as CM  
except:  
    print("Falta el archivo FSMSignal o CMap...")
```

```
sys.exit(0)

THIS_FOLDER = os.path.dirname(os.path.abspath(__file__))
#-----
--
#
# @parameters
#-----
--
try:
    param = open(os.path.join(THIS_FOLDER, 'PARAMETERS.txt'), "r")
    param.readline()
    device = ((param.readline()).split())[2]
    srate = int(((param.readline()).split())[4])
    bsize = int(((param.readline()).split())[4])
    dict_path = ((param.readline()).split())[4]
    core_mode = int(((param.readline()).split())[4])
    force_command = int(((param.readline()).split())[5])
    force_echo = int(((param.readline()).split())[2])
    param.close()
except:
    print("Error al acceder al archivo PARAMETERS.txt")
    sys.exit(0)
#-----
--
#
# @handler          ctrlc_handler()
#-----
--
def ctrlc_handler( signal , frame ):
    print("")
    print("\033[1;35;40m
=====")
    print("\033[1;35;40m = Se ha finalizado el programa por medio de una
interrupcion.=")
    print("\033[1;35;40m
=====")
    print("\033[1;0;0m")
    dict_file.close()
    sys.exit(0)
    return

signal.signal(signal.SIGTSTP, ctrlc_handler)
#-----
--
#
# @handler          exit_handler()
#-----
--
def exit_handler():
    print("")
    print("\033[1;35;40m
=====")
    print("\033[1;35;40m = Se ha finalizado el programa por medio de un comando.
=")
    print("\033[1;35;40m
=====")
    print("\033[1;0;0m")
    dict_file.close()
    sys.exit(0)
    return

signal.signal(signal.SIGTSTP, ctrlc_handler)
#-----
--
#
# @function          elimina_tildes()
# @ref              https://gist.github.com/victorono/7633010
#-----
--
def elimina_tildes(cadena):
```

```
s = ''.join((c for c in unicodedata.normalize('NFD', unicode(cadena)) if
unicodedata.category(c) != 'Mn'))
return s.decode()

#-----
--
#
# @function      __setup()
#
#-----
--
def __setup( rate , size ):
    model_path = get_model_path()
    try:
        if (core_mode == 0):
            speech = LiveSpeech(
                verbose=False,
                sampling_rate=rate,
                buffer_size=size,
                no_search=False,
                full_utt=False,
                hmm=os.path.join(model_path, 'es-es'),
                lm=os.path.join(model_path, 'es-20k.lm.bin'),
                dic=os.path.join(model_path, 'es.dict')
            )
        else:
            speech = LiveSpeech(
                audio_device = device,
                verbose=False,
                sampling_rate=rate,
                buffer_size=size,
                no_search=False,
                full_utt=False,
                hmm=os.path.join(model_path, 'es-es'),
                lm=os.path.join(model_path, 'es-20k.lm.bin'),
                dic=os.path.join(model_path, 'es.dict')
            )
    except:
        print('Error al importar el dispositivo de audio o modelos: ', device, ' ',
model_path)
        sys.exit(0)
    return speech

#-----
--
#
# @function      __dicrSearch()
#
#-----
--
def __dictSearch( word , file ):
    signal_number = 1
    file.seek(0)
    for line in file:
        if (line[:-1] == word):
            return signal_number
        else:
            signal_number = signal_number + 1
    return 0

#-----
--
#
# @function      __wordHandler()
#
#-----
--
def __wordHandler( word ):
    # Lower.
    h_word = word.lower()
    h_word = elimina_tildes(h_word.decode('utf-8'))
    # HASHMAP
    #
    HMAP = CM.HMAP
    #
    #
    h_word = HMAP.get(h_word, h_word)
    return h_word
```



```
#-----
--
#
# @main
#
#-----
--
if __name__ == '__main__':
    # Console interface.
    print("\033[1;33;40m")
    print("\033[1;33;40m
=====")
    print("\033[1;33;40m = Ejecutando LOLA speech recognition.
=)
    print("\033[1;33;40m = Autor: Alberto Palomo Alonso.
=)
    print("\033[1;33;40m
=====")
    print("\033[1;33;40m")
    print("\033[1;31;40m
=====")
    print("\033[1;31;40m = Cargando modelo ...
=)
    speech = __setup(srate, bsize)
    print("\033[1;31;40m = Cargado el modelo.
=)
    print("\033[1;31;40m
=====")
    print("\033[1;33;40m")
    print("\033[1;33;40m
=====")
    print("\033[1;33;40m = Abriendo diccionario...
=)
    try:
        dict_file = open(os.path.join(THIS_FOLDER,dict_path), "r")
    except:
        print("Error al abrir el archivo DICT.txt")
        sys.exit(0)
    print("\033[1;33;40m = Diccionario abierto.
=)
    print("\033[1;33;40m
=====")
    print("\033[1;33;40m")
    print("\033[1;33;40m
=====")
    print("\033[1;33;40m = Ejecutando programa...
=)
    print("\033[1;33;40m = Cancele la ejecucion en cualquier momento con ctrl+Z.
=)
    print("\033[1;33;40m
=====")
    print("\033[1;31;40m")
    # Computing loop:
    state = 0
    number = ''
    for phrase in speech:
        if force_echo == 1:
            print(phrase)
        words = str(phrase).split()
        for word in words:
            mysignal = __dictSearch(__wordHandler(word), dict_file)
            if(mysignal > 0):
                if((raspi == True) and (force_command == 0)):
                    [state, number] = FSMS.__FSMSignalRPI(mysignal, state, number)
                else:
                    [state, number] = FSMS.__FSMSignal(mysignal, state, number)
            if state == -1:
                exit_handler()
#-----
--
#
# @endfile
#
#-----
--
```

### 16.3.FSMSignal.py (Manejador de señales / DEMO).

```
#-----
--
#
# @function      __FSMSignal/RPI/()
#
#-----
--
def __FSMSignalRPI( signal , state , number):
    #Manejador GPIO
    return [0, number]

def __FSMSignal( signal , state , number):
    #Manejador de estado:
    if(signal == 1 or signal==25):
        print('=== > Hola, buenas, soy Lola.')
    if(signal == 2):
        return [1, number]
    if(signal == 3):
        print('=== > Llamando a : ' + number)
    if(signal == 5):
        number = '[Numero de bernardo]'
    if(signal == 6):
        number = '[Numero de luisa]'
    if(signal == 7):
        number = '[Numero de juana]'
    if(signal == 8):
        number = '[Numero de richard]'
    if(signal == 9):
        print('=== > Que pase un buen dia.')
        return [-1,number]
    if(signal >= 10 and signal <= 19):
        number = number + str(signal-10)
        print('=== > Numero : ' + number)
    if(signal == 20 or signal == 22):
        number = ''
    if(signal == 21):
        print("=== > Si?")
    if(signal == 23):
        return [2,number]
    if(signal == 24):
        return [3,number]
    # -----
    if(state == 1):
        if(signal == 4):
            print("=== > Me estoy moviendo a la cocina.")
    if(state == 2):
        if(signal == 4):
            print(" === > Encendiendo cocina.")
    if(state == 3):
        if(signal == 4):
            print(" === > Apagando cocina.")

    return [0, number]
```

## 16.4.CMap.py (Mapa de correcciones / DEMO).

```
HMAP = {  
'buenas' : 'hola',  
'chao' : 'adios',  
'siente' : 'siete',  
'pero' : 'cero',  
'hare' : 'nueve',  
'pico' : 'cinco',  
'unos' : 'uno',  
'cinico' : 'cinco',  
'diana' : 'llama',  
'mueren' : 'nueve',  
'bernard' : 'bernardo',  
'muere' : 'nueve',  
'acaba' : 'apaga',  
'muevete' : 've',  
'cortina' : 'cocina',  
'acabar' : 'apagar'  
}
```

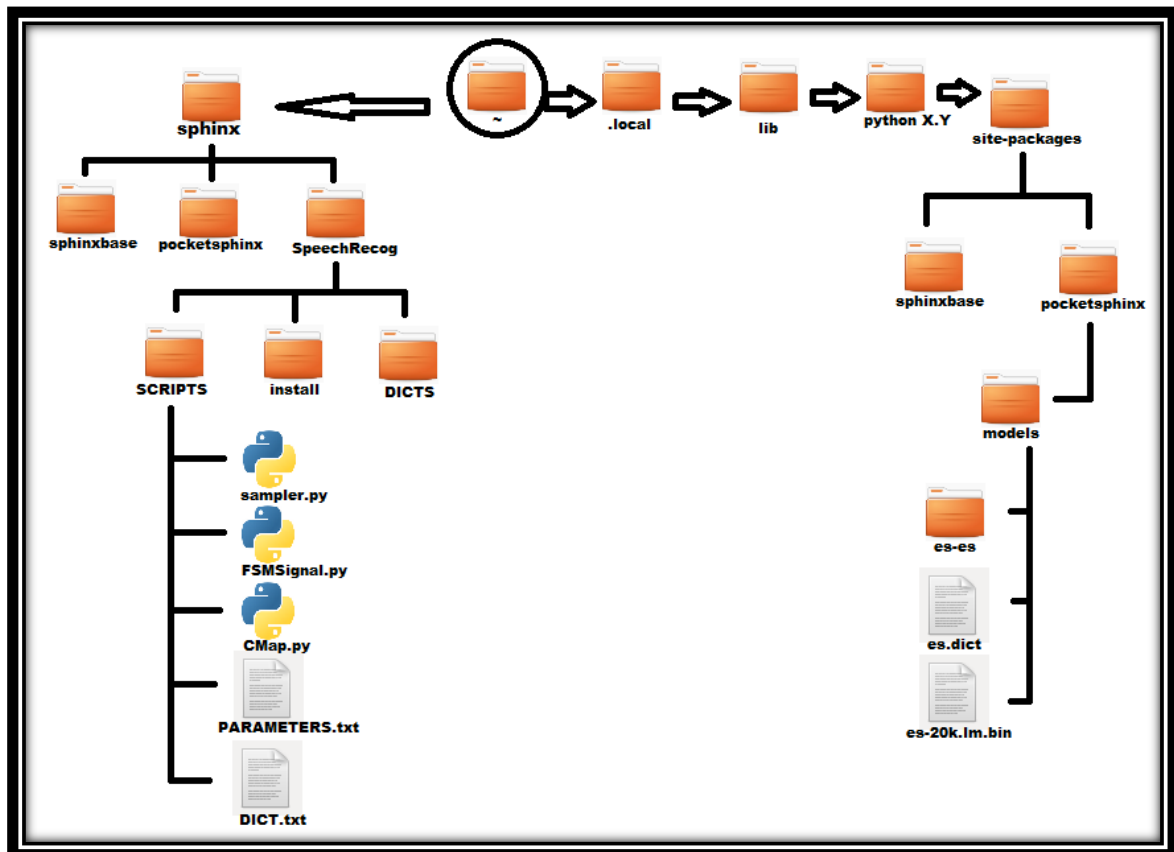
## 16.5.DICT.txt (Diccionario / DEMO).

hola  
vamonos  
llama  
cocina  
bernardo  
luisa  
juana  
richard  
adios  
cero  
uno  
dos  
tres  
cuatro  
cinco  
seis  
siete  
ocho  
nueve  
no  
si  
cancelar  
encender  
apagar  
ven

## 16.6.PARAMETERS.txt (Parámetros).

```
#Parameters for the python program:  
Dispositivo = '1'  
Frecuencia de muestreo = 32000  
Tamaño del buffer = 2048  
Path del diccionario = DICT.txt  
Modo del núcleo = 0  
Forzar línea de comandos = 1  
Eco = 0
```

## 17. Anexo II. Árbol.



## 18. Anexo III. Links.

- [.1] <https://github.com/iTzAlver/SpeechRecog.git>
- [.2] <https://github.com/cmusphinx/sphinxbase.git>
- [.3] <https://github.com/cmusphinx/pocketsphinx.git>
- [.4] <https://cmusphinx.github.io/wiki/tutoriallm/>

## 19. Anexo IV. Matlab publish.

### Pruebas de reconocimiento de audio en un robot social.

En este script de Matlab, se procede a generar un repertorio de pruebas donde se proceden a contar el número de fallos de las palabras para conocer el rendimiento del sistema.

- Se utilizarán 5 palabras: Hola, Cocina, Tres, Dos, Nueve.
- Cada palabra se mencionará 10 veces.
- Se contará el número de palabras falladas sobre el total para elaborar un rendimiento.
- Se realizan pruebas de las siguientes limitantes físicas:

- |                          |          |
|--------------------------|----------|
| 1. Distancia.            | (Fallos) |
| 2. Ruido.                | (Fallos) |
| 3. Raspberry Pi 3.       | (Tiempo) |
| 4. Raspberry Pi 4.       | (Tiempo) |
| 5. Virtual Box (Ubuntu). | (Tiempo) |
| 6. Mapa de correcciones. | (Fallos) |

#### Pruebas de distancia.

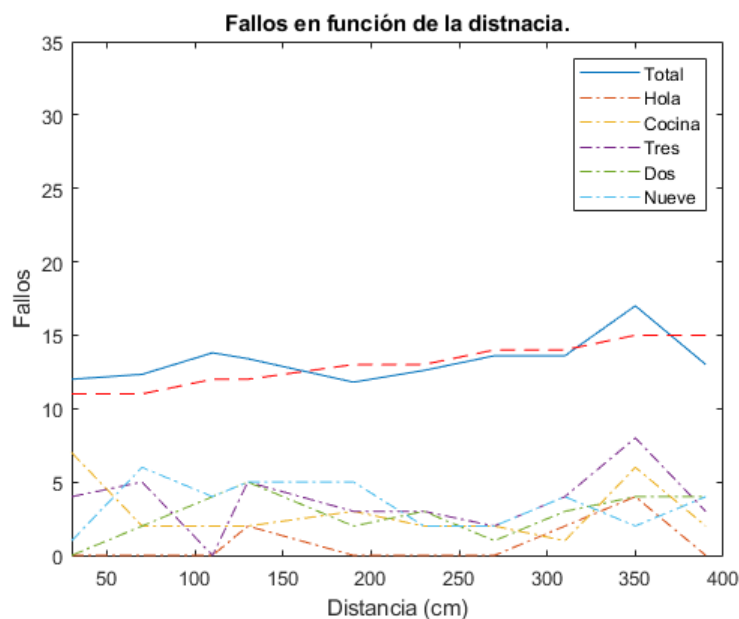
En esta sección se procede a contar el número de fallos y a generar un rendimiento en base a 5 palabras en 10 distancias diferentes, por lo que tenemos un total de 500 palabras. Todo en función de la distancia.

```
x = [30 70 110 130 190 230 270 310 350 390];
Hola = [0 0 0 2 0 0 0 2 4 0];
Cocina = [7 2 2 2 3 2 2 1 6 2];
Tres = [4 5 0 5 3 3 2 4 8 3];
Dos = [0 2 4 5 2 3 1 3 4 4];
Nueve = [1 6 4 5 5 2 2 4 2 4];
AVERAGE = [11 11 12 12 13 13 14 14 15 15];
Total = Hola + Cocina + Tres + Dos + Nueve;

plot(x, smooth(Total), '-')
hold on
plot(x, Hola, '-')
plot(x, Cocina, '-')
plot(x, Tres, '-')
plot(x, Dos, '-')
plot(x, Nueve, '-')
plot(x, AVERAGE, '-r')
hold off

legend('Total','Hola','Cocina','Tres','Dos','Nueve')
title('Fallos en función de la distnacia.')
xlabel('Distancia (cm)')
ylabel('Fallos')

xlim([30 400])
ylim([-0.1 35])
```



```
rendimiento = (1 - (sum(Total)/500))*100;  
disp("Rendimiento total en el rango 0-4 metros (%):")
```

Rendimiento total en el rango 0-4 metros (%):

```
disp(rendimiento)
```

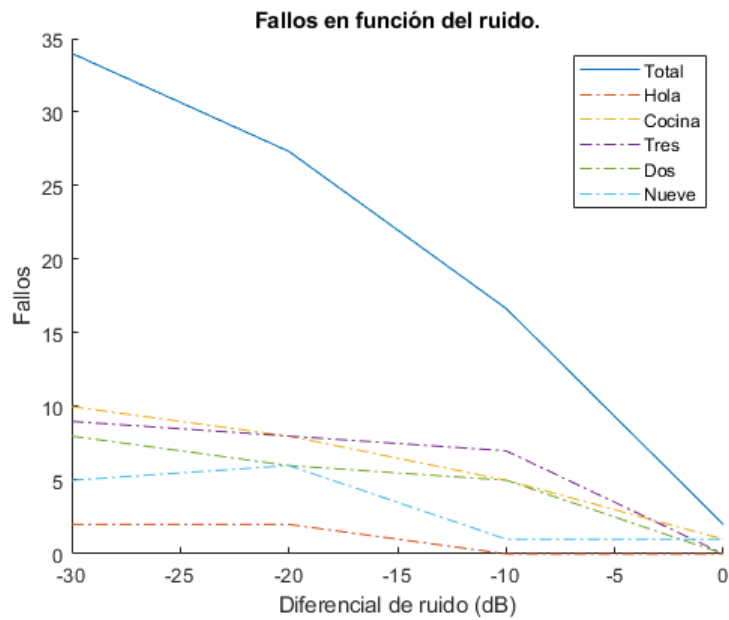
72.6000

### Pruebas de ruido.

En esta sección se procede a contar el número de fallos y a generar un rendimiento en base a 5 palabras en 10 distancias diferentes, por lo que tenemos un total de 500 palabras. Todo en función del ruido.

```
x = [-30 -20 -10 0];  
Hola = [2 2 0 0];  
Cocina = [10 8 5 1];  
Tres = [9 8 7 0];  
Dos = [8 6 5 0];  
Nueve = [5 6 1 1];  
Total = Hola + Cocina + Tres + Dos + Nueve;  
  
figure  
hold on  
plot(x, smooth(Total))  
plot(x, Hola, '-.')  
plot(x, Cocina, '-.')  
plot(x, Tres, '-.')  
plot(x, Dos, '-.')  
plot(x, Nueve, '-.')  
hold off  
  
legend('Total', 'Hola', 'Cocina', 'Tres', 'Dos', 'Nueve')  
title('Fallos en función del ruido.')  
xlabel('Diferencial de ruido (dB)')  
ylabel('Fallos')
```





```
rendimiento = (1 - (sum(Total)/(10*6*5)))*100;
disp("Rendimiento total en el rango -30 - 0 dB por encima del audio
(%):")
```

Rendimiento total en el rango -30 - 0 dB por encima del audio (%):

```
disp(rendimiento)
```

## Pruebas de corrección.

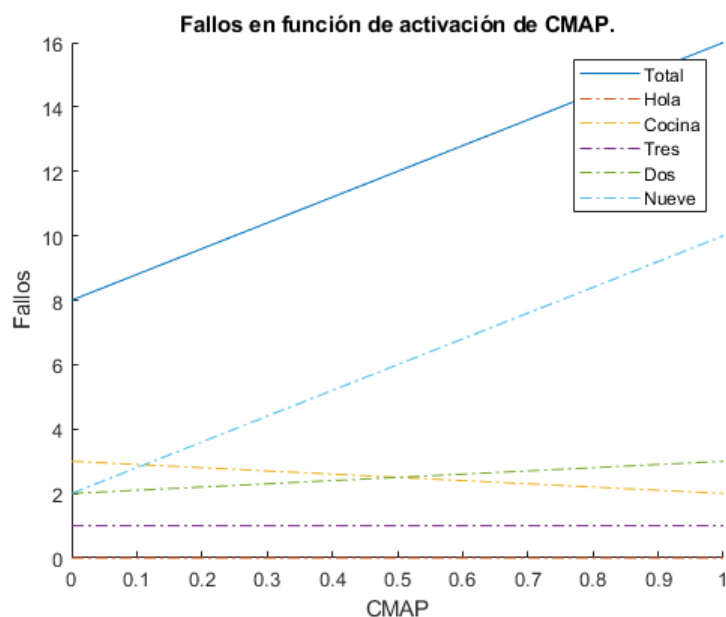
Se realizan pruebas del rendimiento del sistema sin el mapa de correcciones.

```

x = [0 1];
Hola    = [0 0];
Cocina  = [3 2];
Tres    = [1 1];
Dos     = [2 3];
Nueve   = [2 10];
Total = Hola + Cocina + Tres + Dos + Nueve;

figure
hold on
plot(x, smooth(Total))
plot(x, Hola, '-.')
plot(x, Cocina, '-.')
plot(x, Tres, '-.')
plot(x, Dos, '-.')
plot(x, Nueve, '-.')
hold off

legend('Total','Hola','Cocina','Tres','Dos','Nueve')
title('Fallos en función de activación de CMAP.')
xlabel('CMAP')
ylabel('Fallos')
  
```



```
rendimiento = (16/50 - 8/50)*100;  
disp('El rendimiento aumenta un (%) con el mapa de correcciones:')
```

El rendimiento aumenta un (%) con el mapa de correcciones:

```
disp(rendimiento)
```

16

## 20. Anexo V. Manual del programa.

# RECONOCIMIENTO DE COMANDOS VIA PYTHON

Este repositorio contiene un programa capaz de realizar señales al sistema operativo a partir de comandos mediante voz. Ha sido diseñado con objetivo de controlar un robot, manteniendo una ejecución en paralelo. Este modelo ha sido diseñado para Python 2.7.

*Autor: Alberto Palomo Alonso.*

*Tutor: Saturnino Maldonado Bascón.*

*Universidad de Alcalá de HERNANDES. Escuela Politécnica Superior.*

## 1.- INSTALACIÓN DEL MODELO

```
1.- Instalación:      $ git clone
https://github.com/iTzAlver/SpeechRecog
                        $ cd SpeechRecog/install/
                        $ ./installer.sh
                        $ source /.bashrc

2.- Eliminar residuos: $ rm -r SpeechRecog

3.- Reiniciar el sistema: $ sudo reboot
```

## 2.- EJECUCIÓN DE DEMO:

Existen dos modos de ejecutar el programa:

- Ejecución simple:                      \$            Lola
- Ejecución de archivo:                \$            python  
~/sphinx/SpeechRecog/SCRIPTS/sampler.py

Tras la llamada al programa, aparecerá un mensaje de carga del modelo y detección de micrófono junto con la presentación del programa. Si el programa no ha recibido errores, como la detección de micrófono u otros errores posibles, recibirá el siguiente mensaje en la línea de comandos:

***Ejecutando programa...***

***Cancele la ejecucion en cualquier momento con ctrl+Z.***

## 3.- USO DE LA DEMO:

Durante la demo, todas las palabras que el sistema reconozca serán devueltas a la línea de comandos en forma de eco. Si el sistema no devuelve las palabras que usted ha pronunciado, compruebe el micrófono o refiérase a la documentación del proyecto para más información.

### **NOTA IMPORTANTE: DURANTE LA CARGA EL USUARIO DEBE PERMANECER EL SILENCIO**

Pruebe a pronunciar el siguiente conjunto de palabras:

- Hola  
Respuesta: Hola, buenas, soy Lola.
- Uno  
Respuesta: Numero : 1
- Dos  
Respuesta: Numero : 12
- Nueve  
Respuesta: Numero: 129
- Llama  
Respuesta: Llamando a : 129

- Richard  
Respuesta: [Numero de Richard]
- Vámonos a la cocina (Cambio de estado de la máquina de estados.) +  
Respuesta: Me estoy moviendo a la cocina.
- Adiós  
Fin del programa.

Decir números consecutivos provoca que se almacenen en el buffer, decir las palabras 'no' o 'cancelar' eliminan el buffer. Si las respuestas han sido las adecuadas, significa que el sistema está listo para ser adaptado a su proyecto y utilizarse.

## 4.- ADAPTACIÓN DEL SISTEMA:

Como usuario, deberá modificar los siguientes ficheros (en los siguientes apartados se profundizará):

- 1.- Fichero de diccionario de usuario:  
~/sphinx/SpeechRecog/SCRIPTS/DICT.txt
- 2.- Fichero de parámetros del sistema:  
~/sphinx/SpeechRecog/SCRIPTS/PARAMETERS.txt
- 3.- Fichero de manejador de señales:  
~/sphinx/SpeechRecog/SCRIPTS/FSMSignal.py

Como programador, deberá modificar, adicionalmente, los siguientes ficheros:

- 4.- Fichero de corrección de errores:  
~/sphinx/SpeechRecog/SCRIPTS/CMap.py
- 5.- Fichero de diccionario del modelo: ~/local/lib/python2.7/site-packages/pocketsphinx/models/es.dict

### 4.1.- Fichero de diccionario de usuario:

Existen dos tipos de señales:

- **Señales internas:** Son las señales que genera el sistema (sampler.py) hacia el manejador de señales en función de la palabra reconocida.
- **Señales del sistema:** Son las señales que genera el manejador de señal hacia el sistema operativo en función de la acción asociada.

En este fichero debemos de introducir las palabras que queremos que se traduzcan a señales internas. Por lo que todas las palabras que queremos reconocer deben de estar incluidas en este diccionario con el siguiente formato:

- \* Sin acento.
- \* En minúsculas.
- \* Sin espacios.
- \* Sin ningún tipo de carácter que no sea una letra.

\* Con retorno de carro al final de cada línea.

Cabe destacar que para que la palabra sea reconocida, debe de estar también incluida en el diccionario del modelo, que contiene 20.000 palabras en castellano.

#### 4.2.- Fichero de parámetros del sistema:

El sistema tiene varios parámetros modificables y son los siguientes:

- 1.- Dispositivo: Es el micrófono a utilizar, esto es útil únicamente si existen varios micrófonos conectados al dispositivo. Sólo se tiene en cuenta si el modo del núcleo es 1.
- 2.- Frecuencia de muestreo: Frecuencia a la que se extraen las muestras del audio. El valor óptimo en una Raspberry es de 3200.
- 3.- Tamaño del buffer: Tamaño del buffer de audio. El valor óptimo en una Raspberry es de 2048 ó 1024.
- 4.- Path del diccionario: Ubicación del diccionario de usuario en el sistema de archivos, puede modificarse la ruta del mismo, aunque no se recomienda. La ruta de este parámetro es relativa al directorio de SCRIPTS.
- 5.- Modo del núcleo: Modo de reconocimiento de dispositivo en el kernel. Si su valor es 0, utilizará el dispositivo predeterminado por ALSA; si su valor es 1, utilizará el dispositivo con el índice del valor que esté puesto en 'Dispositivo'.
- 6.- Forzar línea de comandos: Escoge la función del manejador de señales.
  - \*Si el valor es 1: El manejador de señales utilizará la función `__FSMSignal()` forzosamente. El valor 1 corresponde por defecto a la DEMO.
  - \*Si el valor es 0: El manejador de señales escogería la función `__FSMSignal()` si el dispositivo NO es un Raspbian y usará la función `__FSMSignalRPI()` si SÍ nos encontramos en un Raspbian.
- 7.- Eco: Activa o desactiva la función de eco. El eco repite las palabras dichas por el usuario en modo de texto en la DEMO. Se recomienda solo para DEBUG.

#### 4.3.- Fichero de manejador de señales:

El manejador de señales, traduce las señales recibidas por los scripts (señales internas) en señales al sistema operativo (señales del sistema). Estas funciones debe definir las el usuario, dado que existe un amplio abanico de posibilidades de mandar señales, generar máquinas de estados y muchas más cosas que no dependen del sistema de reconocimiento de audio y sí dependen de parámetros como: el lenguaje en el que está escrito el programa principal o; qué sistema operativo se está ejecutando. Existen dos funciones principales:

\* `__FSMSignalRPI()`: Esta se ejecuta cuando estamos en Raspbian y el valor 'Forzar línea de comandos' del fichero de parámetros del sistema vale 0.

\* `__FSMSignal()`: Esta se ejecuta cuando NO estamos en Raspbian y el valor 'Forzar línea de comandos' del fichero de parámetros del sistema vale 1.

Sea cual sea la función que queramos utilizar, aquí debe de ir el código que el usuario debe de desarrollar. Las posibilidades que existen para mandar señales por el sistema operativo son las siguientes:

- \* Señales de procesos. (cualquier lenguaje)
- \* Tuberías del sistema operativo. (cualquier lenguaje)
- \* Librerías como 'multiprocessing', 'zmq' y otras. (python en cualquier versión)
- \* Cualquier comunicación manual desarrollada por el usuario. (cualquier lenguaje)
- \* Implementación directa de código sobre la función. (sólo python 2.7)

Estas funciones contienen dos variables importantes, AMBAS DEBEN DE SER DEVUELTAS AL SCRIPT PRINCIPAL CON UN RETURN [state,number]:

- \* `state`: Estado de la máquina de estados, en caso de que el usuario quiera desarrollarla. Si se devuelve -1 al script principal, este termina la ejecución del programa.
- \* `number`: Buffer en forma de string. En caso de que el usuario necesite un buffer.

**NOTA IMPORTANTE:** Para determinar si estamos en Raspbian, se importa la librería **RPiGPIO** de Python, que viene por defecto instalada en Raspbian. Si esta se instala en otro sistema operativo, será evaluado como Raspbian.

#### 4.4.- Fichero de corrección de errores:

Dado que el rendimiento del modelo es de aproximadamente un 55% en las especificaciones ambientales adecuadas (un valor un poco bajo) debido a la colisión entre palabras, cabe la posibilidad de corregir las palabras NO utilizadas y que se estén confundiendo con las que queremos pronunciar en las palabras que queremos pronunciar. Por tanto, se añade un fichero que contiene un HashMap que corrige estos errores. Si por ejemplo al pronunciar la palabra 'nueve' el sistema la detecta como 'muere' podemos añadir la siguiente línea al HashMap: 'muere' : 'nueve', y por ende, 'muere' pasa a ser sinónimo de 'nueve'. Esta característica mejora el rendimiento del sistema a aproximadamente un 75%, por lo que por muy coloquial que pueda parecer, es necesaria en la mayoría de los casos que querramos un sistema de buen rendimiento. Además, podemos realizar labor de sinónimos con este HashMap. Añadir una palabra al HashMap con el siguiente formato:

- 'PALABRA ERRÓNEA' : 'PALABRA CORRECTA',
- (Referirse al ejemplo proporcionado en la DEMO para más información).



#### 4.5.- Fichero de diccionario del modelo:

Este fichero contiene 20.000 palabras en castellano, por lo que, si la palabra que quieres añadir a tu repertorio de comandos no se encuentra aquí, no será reconocida. Dado que este diccionario utiliza un Modelo Oculto de Markov con los parámetros ya adaptados, podemos eliminar palabras del diccionario QUE NO UTILIZEMOS para mejorar LIGERAMENTE (hasta un 5%) el rendimiento del sistema, dado que evitaremos ciertas colisiones entre palabras. Dado que el modelo ya tiene los parámetros entrenados NO podremos añadir palabras. Recomendando no modificar este diccionario y utilizar en su lugar el fichero de CMap.py para evitar colisiones.

#### 5.- Especificaciones ambientales:

El sistema está diseñado para soportar las siguientes condiciones ambientales:

- Rango de distancia del locutor del micrófono entre 0 y 6 metros.
- Ruido blanco gaussiano como máximo a 6dB por debajo del umbral de voz del locutor.
- Se recomienda eliminar todo ruido lingüístico. (palabras aleatorias que el sistema pueda detectar).
- La distancia del locutor del micrófono no disminuye drásticamente el rendimiento. (66% a 6 metros vs 72% a 1 metro)
- El ruido ambiental disminuye drásticamente el rendimiento. (-6dB rendimiento de 50% vs 75% a -30dB)

#### 6.- Cambio de nombre al comando:

Para cambiar el nombre al comando basta con sustituir en la siguiente línea del fichero:

- `~/bashrc`

La palabra Lola por la que deseemos que se llame nuestro comando en la línea:

- `alias Lola = 'python ~/sphinx/SpeechRecog/SCRIPTS/sampler.py'`

Posteriormente actualizar el fichero con el siguiente comando:

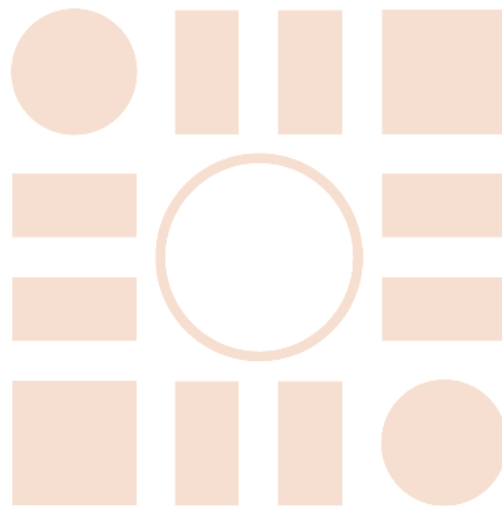
- `$ source ~/bashrc`

Hecho esto, podremos ejecutar el programa desde cualquier directorio.





Universidad de Alcalá  
Escuela Politécnica Superior



ESCUELA POLITECNICA  
SUPERIOR



Universidad  
de Alcalá