Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Reconocimiento de voz con el robot Félix

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Mario Parreño Lara

Tutor: Carlos David Martínez Hinarejos

2016-2017

Resumen

En la actualidad, son cada vez más los sistemas que implementan reconocimiento del habla, ya sea por la comodidad que estos nos ofrecen o por que la tasa de error ha descendido en gran medida desde los últimos años. Así pues, en el siguiente trabajo trataremos de desarrollar un robot capaz de reconocer una serie de palabras que, articulándolas, nos permitan darle órdenes a través de la voz. Para ello explicaremos todo lo necesario, desde un simple modelo 3D hasta cómo mejorar el reconocimiento para que sea más preciso. Además, realizaremos algunas propuestas de mejora y extensión del trabajo que mejorarían la experiencia al usuario de Félix.

Palabras clave: iATROS, HMM, Markov, Viterbi, habla, reconocimiento, Sockets.

Abstract

Currently, there are more and more systems that implement a speech recognition system. Either because of the comfort they provide to us or either because the error rate has sharply dropped to a great extent in the past years. Thus, in the following paper we will try to develop a robot that recognizes a series of words. To do this, when a human articulates a word, an order will be transmitted and understood by the robot. For this, we will explain everything, from a simple 3D prototype to implementations in order to make voice detection more precise. In addition, we will do some implementations proposals and work's extension that would improve the experience of Félix's users.

Keywords: iATROS, HMM, Markov, Viterbi, speech, recognition, Sockets.

Tabla de contenidos

1. INTRODUCCIÓN

Desde siempre he sentido un gran interés por la robótica y la informática y he tenido el deseo de poder realizar un auténtico proyecto en el que pusiera en práctica todo aquello que conocía. Hace un par de años descubrí el campo de la inteligencia artificial y me apasionó. Finalmente, conseguí encontrar un proyecto que combinara estas dos disciplinas, la robótica y la inteligencia artificial, un proyecto que despierta en mí ese interés de investigar y trabajar en él sin verlo como una obligación más. Me decanté por la implementación de un sistema de reconocimiento de voz sobre un pequeño robot que me llamó mucho la atención nada más verlo, Félix[1].

Éste es idóneo para aplicar todo lo aprendido a lo largo de los estudios en el grado en Ingeniería Informática, ya que trataremos temas de redes, diseño de interfaces, bases de datos, sistemas inteligentes o ingeniería del software, como veremos. Además, en él encontramos ese enfoque hacia el mundo comercial, al considerar nuestro robot como un primer prototipo de robot inteligente, de los cuales se prevé un gran aumento para los próximos años [3].

Para abordar el trabajo explicaremos de dónde surge Félix y cómo debemos montarlo. A continuación, pasaremos con el funcionamiento interno del sistema desde el punto de vista programático, explicando cómo calibrar los servomotores que Félix utiliza y cómo enviaremos las órdenes para poder procesarlas fácilmente. Explicaremos las bases del reconocimiento del habla para así poder finalizar siendo capaces de articular un sistema de reconocimiento de voz sobre nuestro cuadrúpedo, estudiando cómo optimizar dicho sistema para errar el menor número de veces posible.

1.1. Motivación

En una primera instancia, el presente proyecto iba a tratar la implantación de un sistema de reconocimiento de voz sobre el robot fabricado por la empresa tecnológica española BQ llamado Zowi [2], pero tras indagar un poco en la escasa documentación, no encontramos una solución sencilla para la una de las tareas subyacentes principales de nuestro sistema, la toma de audio.

A su vez, por la misma época en la que se empezó a plantear el proyecto, se lanzó una impresora de bajo coste nacida como un proyecto que pedía financiación en la página Kickstarter, 101Hero [15], y tratando de imaginar qué podría hacer si adquiría una de éstas, me topé con un modelo de robot cuadrúpedo que me llamo la atención, Félix. Tras plantear las ventajas e inconvenientes que la construcción de éste conllevaría y dialogar con el creador del modelo, me decidí a construirlo con la intención de implementar en éste el sistema de reconocimiento de voz que se resistía en Zowi.

[1] Pinshape. *Quadruped Robot*. `https://pinshape.com/items/27304-3d-printed-quadruped-robot` [Consultado: 7 de octubre de 2016]

1.2. Estado del arte

En la actualidad existen una infinidad de robots de diferentes tipos y formas para llevar a cabo todo tipo de actividades. Y es que la utilización de pequeños asistentes o robots en nuestro hogar está en auge, teniendo todo tipo de finalidades como la de facilitarnos la realización de cualquier tipo de tarea o la de entretenernos. Uno de los dispositivos que podemos destacar es Amazon Echo [13], un dispositivo de la empresa Amazon que es capaz de reconocer órdenes de voz, permitiéndonos realizar tanto tareas relacionadas con el hogar (como enchufar unas luces), contestar a llamadas o reproducir música.

Por otra parte, nos encontramos con otra serie de dispositivos con un enfoque diferente, robots cuya finalidad es la de proporcionar compañía o servir meramente como juguetes. Entre estos podemos destacar el gato robótico desarrollado por la empresa de juguetes Hasbro, Alan. Éste es capaz de reaccionar a caricias y realizar acciones dictadas por voz. Cabe destacar la utilización de estos también para la medicina [1], existiendo varias investigaciones sobre la utilización de éstos al observar mejoría en pacientes que padecen Alzheimer.

Es indiscutible pues, en el mundo actual, que la robótica está presente en todos los ámbitos de nuestra vida, tratando que ésta nos resulte más sencilla y cómoda. Es por esto pues, que surgen afirmaciones como la realizada por la empresa Gartner, una importante empresa consultora y de investigación de las tecnologías de la información, que vaticina, “En un futuro no muy lejano tendremos más conversaciones con robots que con nuestras parejas”[2].

1.3. Objetivos

Los objetivos del siguiente proyecto son: la construcción de un robot cuadrúpedo capaz de tomar ordenes de voz a través de un micrófono incorporado, tratando dichas órdenes y actuando en consecuencia. Para ello, necesitaremos:

- Instalar sobre la placa de proceso C.H.I.P., los elementos software necesarios para hacer que el reconocedor de habla iATROS funcione correctamente y montar un servidor que nos permita, además, calibrar a Félix.
- Estudiar y comprender las bases del reconocimiento de voz y la implementación mediante iATROS.
- Establecer modelos léxicos y del lenguaje que nos permitan, mediante iATROS, ejecutar las acciones que deseamos.
- Optimizar los parámetros del reconocedor de forma que reduzcamos al mínimo el número de errores de reconocimiento.

[2] Citado en El País, 12/2016, `https://goo.gl/tRIYRn`

2. CONSTRUCCIÓN Y ENSAMBLAJE

Félix *The Quadruped Robot* es un modelo de robot cuadrúpedo creado por Ronald Jaramillo. Apareció por primera vez en la página web Pinshape, un repositorio en línea de objetos 3d y luego con más detalle en la página personal de Jaramillo `https://burningservos.com/`. Además Félix se encuentra bajo licencia *Creative Commons – Attribution* que nos da permiso para utilizar el modelo para nuestro proyecto así como adaptarlo si fuera necesario, de forma gratuita.

El correcto montaje de Félix es un aspecto esencial para su buen funcionamiento. En éste capítulo indicaremos todos los componentes necesarios para la construcción del robot, así como una serie de guías para el correcto ensamblado y trucos para realizar un calibrado de forma más sencilla.

2.1. Componentes y presupuesto

En la siguiente sección vamos a tratar, tras la construcción completa de Félix, el coste económico que éste conlleva. No se contemplarán las herramientas software utilizadas, ya que estas son de licencia libre, y explicaremos cuáles necesitamos para cada funcionalidad en secciones posteriores. Por consiguiente, el conjunto de material electrónico e impresión del modelo nos acarreará un coste total de 175.2€ como observamos en la Tabla 2.1.

Tabla 2.1: Desglose del presupuesto para el montaje de Félix

Descripción	**Cantidad**	**Precio unitario**
Controladora de servomotores PCA9685	1	9.95 €
Batería de Litio de 3.7V	1	5.95 €
Caja para 4 baterías AA	1	2.95 €
Cabezales macho de ángulo recto	1	2.95 €
Micro Servomotores – MG90S	8	9.95 €
Cables hembra / hembra	1	1.95 €
4 Baterías AA recargables Energizer	1	7.95 €
Mini micrófono USB	1	4.95 €
C.H.I.P.	1	9.00 €
Impresión modelo 3D	1	49.95 €
Total		**175.2 €**

2.2. Modelo, impresión y unión de las partes

Félix es un modelo sencillo de un robot cuadrúpedo fácil de imprimir y de montar. Como nos indica su creador, el ensamblaje es muy sencillo, ya que no requiere de tuercas o tornillos, todas las piezas encajan (con una tolerancia de error de 0.3mm) tras imprimirlas.

En primer lugar, deberemos coger las cuatro caderas o *hip* y unirlas a las partes del cuerpo delantera y trasera, véase Figura 2.1.

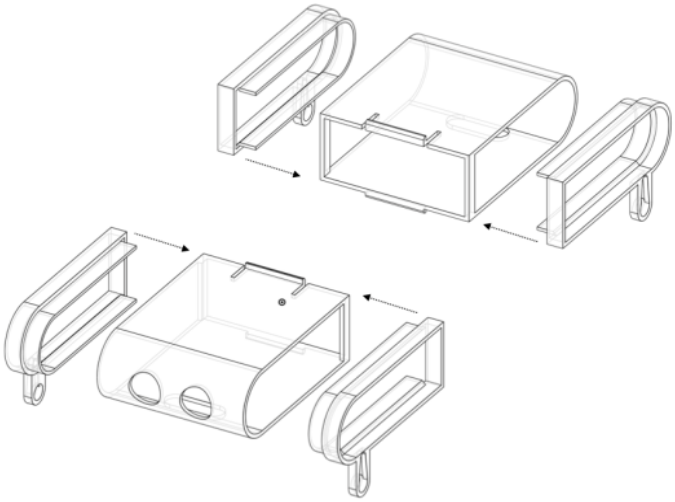

Figura 2.1: Unión de las caderas a parte posterior y anterior de Félix

A continuación, uniremos las dos partes resultantes mediante el cinturón como muestra la Figura 2.2.

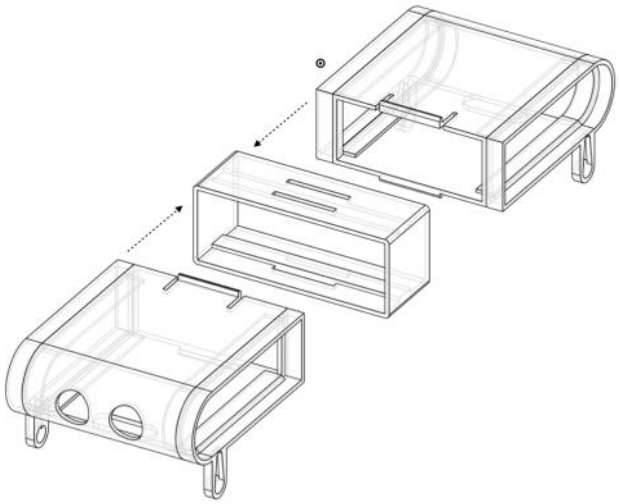

Figura 2.2: Unión de la parte posterior y anterior de Félix con cinturón

Por otra parte, tomaremos los cuatro fémures y los ocho servomotores y los juntaremos de forma que el cable de los servomotores encaje por la parte de los huecos generados por las semicircunferencias como se aprecia en la Figura 2.3. Nótese que, si los servomotores y los muslos quedan un poco sueltos, deberemos hacer que estos queden prietos mediante un poco de cinta adhesiva o algún material similar.

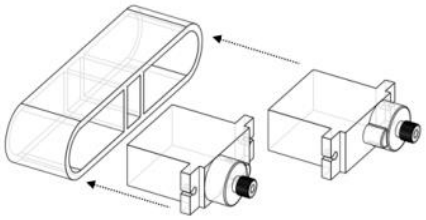

Figura 2.3: Ensamblaje de motores y fémur de Félix

Ahora se insertan los cuernos en los huecos que existen en las caderas y rodillas. Se conectan las caderas con los fémures cómo muestra la Figura 2.4, pero sin apretar todavía los tornillos entre los servomotores y los cuernos, esto lo haremos en la Sección 2.3. Calibrado

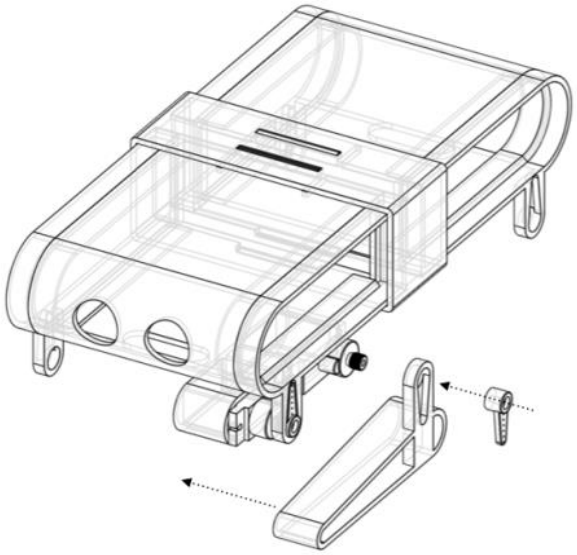

Figura 2.4: Acoplamiento de caderas, fémures y rodillas de Félix

Finalmente, juntaremos todos los muslos con las rodillas de la misma forma, sin apretar los tornillos aún (véase la Figura 2.5); obteniendo como resultado a Félix con las juntas de las piernas sueltas.

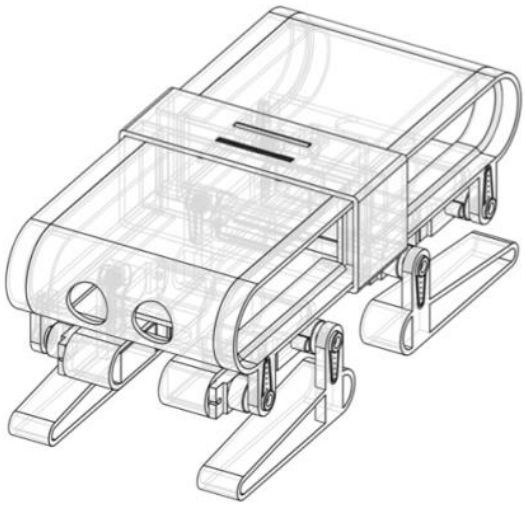

Figura 2.5: Resultado final del montaje de Félix

Con respecto al resto de materiales electrónicos, recomiendo, en primer lugar, mediante algún tipo de cinta adhesiva de doble cara, pegar en el frontal del cuerpo el soporte para la batería de forma que ésta nos servirá para dar mayor estabilidad a Félix.

Tras hacer esto, adheriremos la caja que contendrá las baterías AA a la parte inferior del cinturón. De dicha caja conectaremos su cable rojo al borne etiquetado como V+ de la placa controladora de servomotores PCA9685, y el cable negro al borne GND. Pasaremos todos los cables de los servomotores por los orificios que tenemos en la parte frontal y trasera del cuerpo y conectaremos dichos cables a la controladora de servomotores, de forma que iremos colocándolos contiguamente, a partir de la posición 0, en primer lugar el cable del servomotor superior (correspondiente a la cadera) y después el inferior, de izquierda a derecha guiándonos por la vista expuesta en la Figura 2.6.

Sin llegar a insertar la controladora en el interior del cuerpo, conectaremos los pines y la batería a la placa C.H.I.P., según la referencia expuesta en la Tabla 2.2, y finalmente insertaremos la controladora y C.H.I.P. dentro de Félix.

Tabla 2.2: Referencia conexiones entre controladora de servomotores y C.H.I.P.

Controladora Servomotores PCA9685	**C.H.I.P.**
VCC	3.3V
GND	GND
SCL	TWI2-SCK
SDA	TWI2-SDA

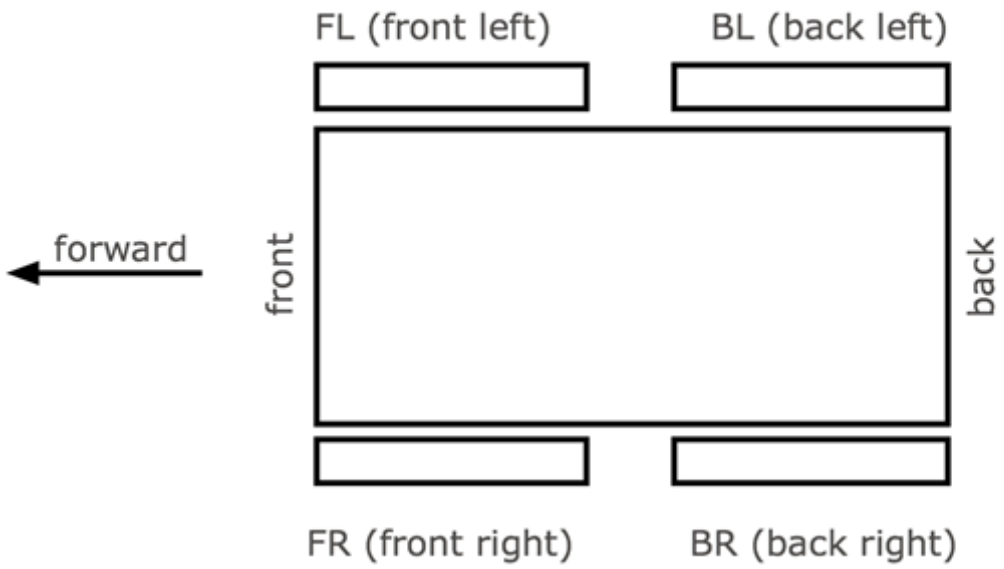

Figura 2.6: Diagrama nomenclatura de las piernas de Félix

2.3. Calibrado

Para concluir este capítulo sobre la construcción y ensamblaje de Félix, vamos a introducir cómo se deben calibrar los diferentes servomotores, para tener así una referencia de sus posiciones al fijar determinados ángulos. Esto nos permitirá saber con relativa exactitud qué consecuencias tendrá posicionar un determinado servomotor en un ángulo u otro y, por consiguiente, ser capaces de establecer poses o una secuencia de éstas (como, por ejemplo, para hacer caminar a Félix).

Así pues, adelantándonos a la inicialización del servidor y cliente web que veremos en el Capítulo 3, arrancaremos nuestro servidor, accederemos a la dirección IP donde se encuentra el cliente y navegaremos en primer lugar a la sección *90º Legs Calibration*. En dicha sección iremos seleccionando las diferentes piernas: FL de *Front Left* para calibrar la pierna frontal izquierda, FR de *Front Right* o pierna frontal derecha, BL de *Back Left* para la pierna trasera izquierda y finalmente BR de *Back Right* para la pierna trasera derecha, siguiendo el esquema de la Figura 2.6. Para cada pierna, iremos seleccionando cada parte: la cadera o *hip* y la rodilla o *knee*.

Cuando seleccionemos una pierna cualquiera y una parte, al variar los parámetros mediante los botones “-” y “+”, los servomotores relacionados con las partes seleccionadas actualizaran sus ángulos a 90 grados. Ahora es cuando debemos atornillar los servomotores a los cuernos con la intención de que al presionar dichos botones, lo que haremos será variar unos parámetros de desviación para tratar de conseguir que las piernas formen 90 grados, como podemos ver en la Figura 2.7.

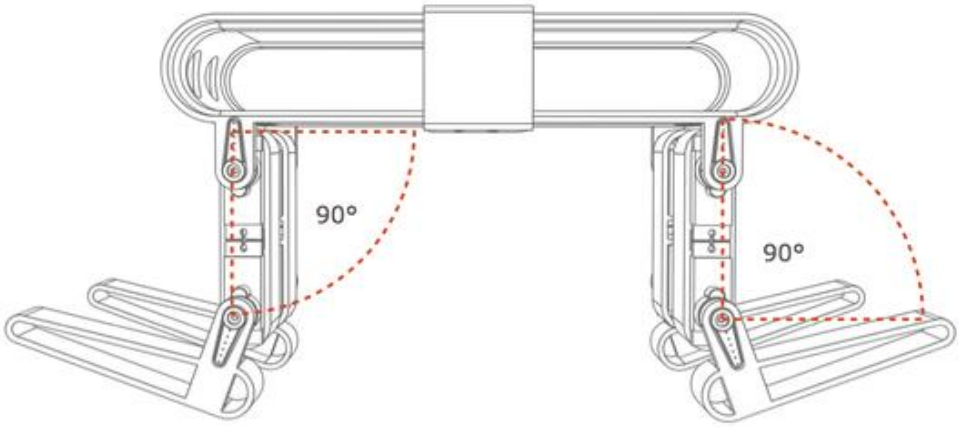

Figura 2.7: Resultado tras la calibración de 90 grados de Félix

Para finalizar con el calibrado de Félix, navegaremos a la siguiente sección: *Home Leg Calibration*. Esta sección funciona de manera similar a la anterior, pero ahora tendremos que calibrar las piernas para llevarlas a las posiciones principales que nos permitirán hacer que Félix ande correctamente. Así pues, tendremos que llevar cada pierna a posiciones como la que se muestra en la Figura 2.8.

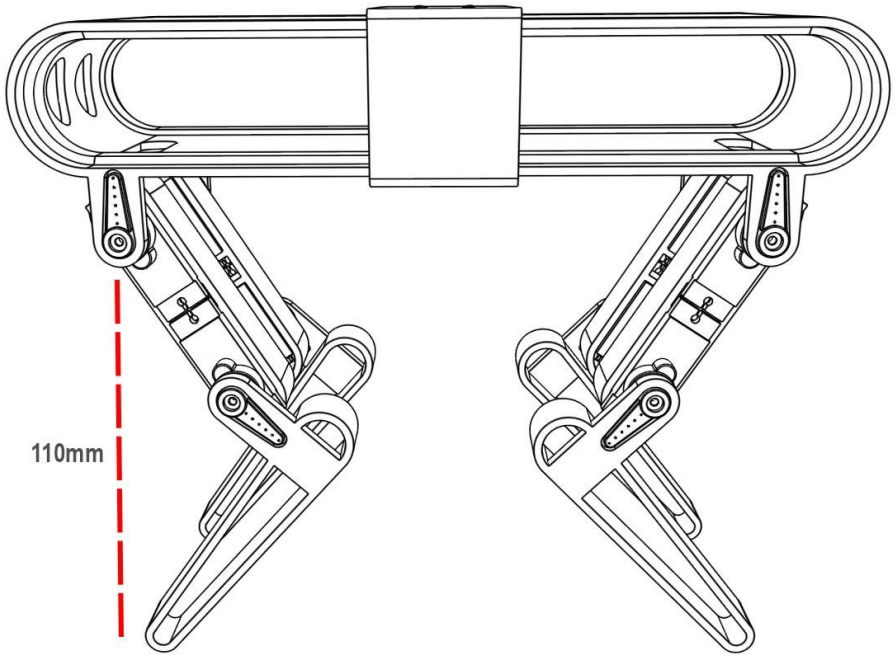

Figura 2.8: Calibración de la posición base de Félix

Tras esto, tendremos a Félix correctamente montado y calibrado.

3. PLANTEAMIENTO Y DISEÑO DEL SISTEMA

Un correcto enfoque sobre cómo debería funcionar nuestro sistema, previamente a la realización de éste, es necesario para que así partamos de una base sólida con la que trabajar, teniendo las ideas claras, pero teniendo la libertad de poder ir realizando pequeños cambios según vayamos avanzando en el proyecto.

Un primer planteamiento que representa a grandes rasgos el funcionamiento de nuestro sistema sería el siguiente: el cliente deberá insertar una dirección IP y un puerto que apuntarán a donde el servidor está inicializado; el cliente Python se encuentra en el mismo Félix, pero el cliente web no tiene por qué, así que, este se intentará conectar al servidor y se quedará esperando un determinado tiempo a que el servidor le envíe un mensaje a través del *socket* abierto, confirmándole que se ha establecido la conexión; una vez establecida la conexión, cada cliente contará con un abanico de órdenes que podrá ejecutar libremente.

3.1. Servidor

Como principal idea, deseamos que desde nuestro cuadrúpedo podamos realizar de forma sencilla tanto la calibración de las piernas como un tratamiento de las órdenes que vayamos reconociendo. Dado este propósito, surge la necesidad de crear algo que nos dé dicho servicio: un servidor local.

3.1.1. Iniciando el servidor

El lado servidor es para Félix una de sus partes fundamentales. En éste, trataremos los diferentes eventos que nos permitirán interactuar con Félix y controlarlo a nuestra voluntad. Para poder llevar esto a cabo hemos optado por utilizar Node.js, un entorno JavaScript del lado servidor basado en eventos. Además, éste cuenta con una serie de *frameworks* que nos facilitarán mucho el trabajo:

- Express.js: Para el servicio de activos estáticos de nuestra aplicación, a través de la función de *middleware* incorporado `express.static`.
- Socket.io: Es uno de los *frameworks* más potentes. Nos permite manejar eventos en tiempo real mediante una conexión TCP. Lo utilizaremos para mandar y recibir mensajes entre el servidor y el cliente.
- Firebase: Nos servirá para establecer la conexión con nuestra base de datos albergada en https://firebase.google.com. Mediante eventos podremos realizar las operaciones básicas de leer, crear, borrar y actualizar datos.
- Chip-io: Nos proporciona una serie de mecanismos para interactuar con C.H.I.P.
- Johnny-Five: Es un *framework* para la robótica que nos ofrece un *kit* básico de control de proyectos hardware como el que manejamos. Gracias a éste podremos crear un objeto que representará a Félix y nos permitirá actuar con sus componentes, en este caso, sus piernas.

Para instalar el servidor únicamente tendremos que descargarnos nuestro proyecto `FelixServer`[3] de GitHub y, en el directorio raíz, descargar mediante NPM las dependencias establecidas en el fichero de configuración `package.json` gracias a la orden `npm install`. Téngase en cuenta que previamente a cualquier despliegue tendremos que haber instalado todo el software necesario en nuestra placa (véase la Subsección 5.1.2). Así pues, ya estará todo listo y solo nos quedará arrancar el servidor mediante la orden `npm start`; el servidor comenzará a inicializarse y, tras una breve demora, obtendremos en la consola en qué dirección IP y puerto se encuentra éste.

Cabe destacar el uso del paquete Nodemon para el desarrollo, una utilidad para nuestro servidor que monitorizará los cambios en nuestro código con la finalidad de reiniciar nuestro servidor automáticamente, y poder así ver los cambios sin tener que reiniciar el servidor cada vez manualmente.

3.1.2. Manejo de eventos

El servidor funciona esencialmente a base de recepción y tratamiento de eventos. Para tratar dichos eventos utilizaremos Socket.io, como indicamos en la Subsección 3.1.1. *Iniciando el servidor* Éste está bastante popularizado y posee una excelente documentación y comunidad que lo respalda.

Tendremos varias formas de comunicarnos y mandar mensajes a Félix. La primera será a través del cliente web que, dependiendo de la sección en la que nos encontremos, enviará un mensaje u otro con diferentes datos. La otra forma de comunicarnos será a través del cliente Python, que tomará una orden de voz y la pasará al servidor como parámetro, en formato JSON, a un evento con determinado identificador (para este caso llamado `voiceCommand`).

Así pues, tras recibir los eventos y los datos de entrada asociados a éstos, según el evento analizaremos qué hacer o configuraremos los parámetros de Félix.

3.1.3. Persistencia

Félix no necesita una compleja arquitectura para la capa de persistencia de datos, ya que únicamente necesitaremos almacenar los parámetros de configuración referentes a los valores de desviación de las caderas y las rodillas de cada pierna (véase la Sección 2.3. Calibrado). Es por esto por lo que hemos decidido emplear la plataforma Firebase de Google. Ésta, con su plan gratuito, nos permite almacenar de forma sencilla y sin tener que instalar ningún sistema de bases de datos, información con un formato de objetos JSON en tiempo real; es decir, mientras variamos los parámetros en nuestro cliente web éstos se respaldarán a su vez en la base de datos definida en Firebase.

En primer lugar, deberemos ir la página oficial de Firebase y crear un usuario nuevo. Allí crearemos un proyecto nuevo que albergará nuestra base de datos. Le asignamos el nombre que deseemos (para este proyecto hemos decidido llamarla `felix-config`), y, accediendo al panel de configuración, tendremos que seleccionar el apartado `Database`. Se nos mostrará una especie de árbol, teniendo como nodo raíz un elemento con el nombre que hemos dado al proyecto. A partir de éste crearemos un nodo denominado `leg-config`, del que colgarán cuatro nodos referentes a las piernas y, a su vez, de cada uno de estos cuatro nodos para los diferentes parámetros, quedando la configuración como sigue se ve en la Figura 3.1.

[3] GitHub. *MarioProjects*. `https://github.com/MarioProjects/FelixServer`

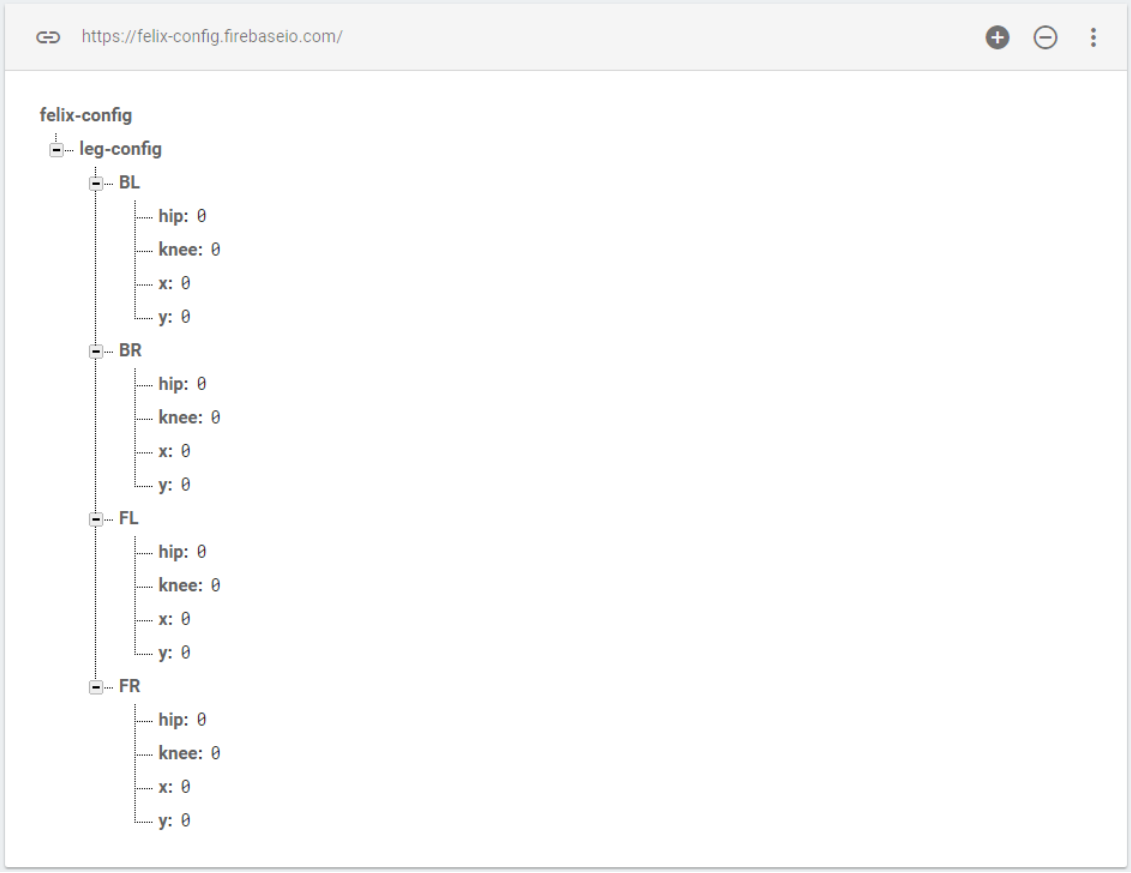

Figura 3.1: Configuración de cada pierna de Félix en Firebase

Para proseguir con la configuración de la conexión, todo lo que necesitaremos será un *token*, el cual obtendremos desde: `Configuración del proyecto`, en la cual haremos clic sobre el botón `Añade Firebase a tu aplicación web` y se nos proporcionará el susodicho *token*.

Por último, como mencionamos en la Subsección 3.1.1. *Iniciando el servidor*, mediante el *framework* Firebase podremos autenticarnos desde nuestro servidor y leer y manipular los datos.

3.2. Cliente web

El cliente web se ubica junto a la parte servidor en el proyecto FelixServer ubicado en nuestro repositorio de GitHub. Nos servirá para tener una mejor experiencia y agilizar el proceso de calibración de los servomotores de Félix. Además, gracias a éste podremos insertar ángulos para cada cadera o rodilla de cada pierna y así poder ir probando poses.

3.2.1. Modelado

Para la creación del cliente web hemos tratado de simplificarlo al máximo, unificando todo en una misma página dividida en secciones, donde cada una de éstas tendrá un propósito específico. Tratando de utilizar las últimas herramientas para facilitar lo referente a maquetación y estilos de la página, hemos optado por las siguientes:

- JQuery: Es una biblioteca JavaScript que simplifica la manera de interactuar con los elementos HTML, así como manejar eventos, desarrollar animaciones y agregar interacción. La utilizaremos para controlar diferentes eventos como el clic en los botones de la configuración para mandar mensajes al servidor, entre otros.
- Bootstrap: Este *framework* de Twitter tiene como objetivo facilitar el diseño web; en concreto, nos proporciona una especie de cuadricula, conocida como *grid system*, que divide nuestra web en doce columnas. En base a estas columnas podemos especificar los tamaños de nuestros elementos y hacerla así adaptable a cualquier tamaño de dispositivo. Además, implementa una serie de clases que dan estilo a los elementos básicos de nuestro HTML.
- Socket.io: Lo utilizaremos para comunicarnos con el servidor.
- Vibrate JS: Se trata de una API especialmente hecha para dispositivos móviles que nos dará la capacidad de, a través de una serie de métodos que la API proporciona, hacer que el dispositivo vibre cuando lo deseemos y como deseemos.
- One page scroll: *Plugin* JQuery que nos proporciona una forma sencilla de añadir transiciones entre las diferentes secciones de nuestra página.
- Font awesome: *Framework* CSS que nos provee de una serie de iconos fáciles de incorporar a nuestro proyecto.

Así pues, como introdujimos, primordialmente la página tiene tres secciones principales:

- *90º Leg Calibration y Home Leg Calibration*: Estas secciones nos sirven para calibrar nuestro cuadrúpedo. Disponemos de varios selectores para elegir las piernas y los parámetros a configurar. Desde JavaScript, utilizando JQuery y los eventos disparados al hacer clic, podremos detectar cuándo variamos el valor de una configuración de pierna, cadera o rodilla seleccionada, para así enviársela a Félix y actualizarla en Firebase.
- *Custom Pose Configuration*: Es la última sección que nos permite interactuar con Félix. Esta es de gran utilidad ya que nos permite establecer los ángulos que deseemos a cada pierna, cadera o rodilla, de forma que podamos ver cómo queda cada uno y así crear poses personalizadas, por ejemplo, para las que luego Félix ejecutará como órdenes.

3.2.2. Conexión al servidor

La idea principal es que tengamos Félix a mano, ya que para la configuración de las piernas explicado en la Sección 2.3, tendremos que ir modificando los valores de los ángulos y observando cómo estos afectan en la posición de las caderas y rodillas, por lo que una conexión remota no tiene bastante sentido; por tanto, se ha decidido que el servidor sirva una página web en un determinado puerto que nos hará la función de cliente web. Así pues, tras arrancar el servidor, en la consola aparecerá la dirección IP y el puerto en el que Félix instancia nuestro cliente, y únicamente tendremos que copiar esta dirección y puerto en un navegador de nuestra preferencia.

Lo primero que observamos al conectarnos a nuestro cliente web es una gran pantalla que nos pregunta por la dirección IP y el puerto en el que está iniciado el servidor (por si deseamos conectarnos desde algún dispositivo que no se encuentre en la misma red que Félix). Como hemos indicado previamente, esta información aparece al arrancar el servidor. Insertaremos los datos requeridos y tratará de crear un *socket* entre el cliente y el servidor. Si ha sido posible, para evitar conexiones erróneas, se le mandará un mensaje al servidor con la intención de que éste responda y saber así que nos hemos conectado realmente a nuestro robot. Si no hemos obtenido respuesta tras un breve periodo de tiempo, se nos volverá a preguntar por la dirección en la que se ubica el servidor de Félix.

Tras conectarnos nos encontraremos con una única página que nos permitirá configurar a nuestro cuadrúpedo de forma sencilla (para más información sobre qué encontraremos véase la Subsección 3.2.1).

3.3. Cliente Python

Para la parte referente a la toma y tratamiento del habla hemos optado por crear un cliente basado en Python, ya que este lenguaje es bastante claro y sencillo y nos proporciona todas las herramientas y mecanismos necesarios para nuestro propósito. Este se ha desarrollado sobre el proyecto `FelixiAtros`[4] en GitHub, donde también almacenamos nuestro reconocedor del habla iATROS.

3.3.1. Modelado, lógica y conexión al servidor

Como hemos dicho, el cliente Python se encargará únicamente de lo referente al reconocimiento del habla. Para esta subsección basta con mencionar tres ficheros que utilizaremos para llevar a cabo nuestro propósito:

- `felix_constants.py`: En este fichero inicializaremos todas las constantes que vayamos a utilizar en nuestros archivos Python. Tendremos las constantes ubicadas y será mucho más rápido modificarlas y probar valores.
- `felix_core.py`: Para la declaración de métodos que utilizaremos para varias tareas como la obtención de una frase o la predicción de ésta.
- `felix_brain.py`: Es el archivo principal, donde se encuentra la lógica de funcionamiento de Félix que explicaremos a continuación.

[4] GitHub. *MarioProjects*. `https://github.com/MarioProjects/FelixiAtros`

Con respecto a la lógica de ejecución de nuestro cliente Python, lo primero que haremos será establecer la conexión entre el cliente y el servidor. Para ello, el programa nos preguntará por la dirección IP y el puerto en el que se encuentra el servidor (véase la Subsección 3.1.1) y tratará de crear un *socket* para comunicarse. Al establecer la conexión correctamente, el programa entrará en un bucle infinito, del que solo podremos salir si se produce una desconexión con Félix. En este bucle, en primer lugar:

1. Tomaremos una frase a través del micrófono y la analizaremos, obteniendo una cadena de texto con lo reconocido.
2. A continuación trataremos la orden en el programa Python, ya que el manejo de cadenas es muy simple. Dada la cadena con la frase reconocida, que puede contener varias órdenes, la cortaremos en ordenes individuales para tratarlas una a una. Esto es posible ya que conocemos el conector entre estas órdenes (la “y”).
3. Si la orden lleva una duración asociada, será nuestro cliente Python el que se ocupe de hacer las esperas oportunas para enviar la siguiente orden al servidor o tomar la siguiente frase y comenzar el bucle de nuevo.

4. RECONOCIMIENTO DEL HABLA

Uno de los objetivos del proyecto es el análisis y empleo de un sistema de reconocimiento automático del habla, también conocido por sus siglas como RAH. Dichos sistemas tienen como objetivo permitir la comunicación hablada entre los seres humanos y computadoras [14] teniendo en cuenta una serie de datos de distintos campos (acústica, fonética, fonológica, léxica, sintáctica, semántica y pragmática) para llegar a obtener sistemas con un error de reconocimiento aceptable.

4.1. Bases

Un aspecto fundamental en el diseño de un sistema de reconocimiento del habla es la elección del tipo de aprendizaje que utilicemos para construir la fuente del conocimiento del sistema. En lo que a nuestro proyecto concierne, el aprendizaje de nuestro robot se va a basar en dos tipos de aprendizaje:

- Aprendizaje inductivo: Nuestro fin es que el sistema sea capaz de obtener conocimiento a través de ejemplos, lo que conocemos como muestras de entrenamiento. Esto nos permitirá obtener una predicción más o menos fiable ante una muestra dada.
- Aprendizaje deductivo: Consiste en la transmisión de conocimiento del experto supervisor a la máquina. Para nuestra labor, podríamos mencionar el uso de `eutranscribe` (véase la Subsección 4.2.2), ya que, gracias a éste somos capaces de indicarle al sistema las transcripciones fonéticas que se corresponden a cada palabra y, así pues, el sistema podría ser capaz de comenzar a clasificar.

Además, podemos clasificar nuestro sistema según diversos criterios, pudiendo afirmar que tratamos con un sistema con un dominio reducido de palabras a reconocer (poco más de una veintena). Tiene un grado de robustez medio, ya que, como veremos en la Subsección 5.1.3, se trata el ruido procedente de los servomotores de Félix y ruido de fondo. Permite reconocer habla continua o, aunque no es lo más recomendable, no es estrictamente necesario ajustar nuestro sistema antes de empezar a utilizarlo.

4.1.1. Modelos ocultos de Markov

Los modelos ocultos de Markov o HMM (por sus siglas del inglés Hidden Markov Model) son una pieza fundamental en el reconocimiento que emplea el reconocedor iATROS que utilizamos para Félix. Éstos son especialmente utilizados en el campo que nos movemos sobre el reconocimiento del habla, donde se emplean para modelar fonemas, entre otros [14].

Un modelo oculto de Markov es un modelo estadístico en el que asumimos que el sistema a modelar es un proceso de Markov de parámetros desconocidos. Para comprenderlo mejor debemos definir los procesos de Markov como procesos estocásticos, es decir, procesos aleatorios que varían con el tiempo, con la propiedad de Markov. Dicha propiedad define que la probabilidad condicional sobre el estado presente, futuro y pasado del sistema es independiente; es por esto por lo que también son conocidos como modelos que no tienen memoria.

El uso que daremos a nuestro modelo será el de encontrar la secuencia más probable de estados ocultos que puedan haber generado una secuencia de salida dada. Este problema lo resolveremos mediante el algoritmo de Viterbi, como veremos en la Subsección 4.1.2. La notación más utilizada para definir un Modelo Oculto de Markov es como la de una tupla formada por los siguientes elementos:

- Q: Representa el conjunto de estados.
- V: Conjunto de valores observables en cada estado. Valores de salida.
- π: Probabilidades iniciales.
- A: Conjunto de probabilidades de transiciones entre estados.
- B: Probabilidades de emisión de los símbolos.

4.1.2. Algoritmo de Viterbi

Encontrar la probabilidad de que un modelo oculto de Markov produzca una serie de observaciones es computacionalmente costoso. Es por esto por lo que emplearemos el Algoritmo de Viterbi [14]. Éste nos permite encontrar la secuencia de estados que con mayor probabilidad explica las observaciones del modelo oculto de Markov mediante un algoritmo en sus fundamentos recursivo, pero que debido a su coste se implementa mediante un algoritmo iterativo usando programación dinámica. Gracias a un argumento que almacena, calculará la secuencia de estados que maximizan la ecuación en un instante de tiempo, proviniéndonos finalmente de lo que denominamos predicción.

4.1.3. Extracción del audio

Para el reconocimiento automático del habla necesitamos, en primer lugar, extraer las características de las componentes de una señal de audio de forma que nos quedemos únicamente con aquello que nos aporte información, que sea relevante, debiendo así eludir todo aquello que no sea información valiosa (como el ruido de fondo, el tono, etc.) y que además empobrecen el proceso del reconocimiento [11].

Para llevar a cabo este cometido, iATROS utiliza el método de extracción de características denominado Coeficientes Cepstrales en las Frecuencias de Mel (más conocidos del inglés *Mel Frecuency Cepstral Coefficients* o MFCCs) [4]. Dicha extracción de características busca modelar el habla eliminando cualquier rasgo del propio hablante.

Para ello, la extracción de características comienza con el preénfasis, donde aplicamos a la señal un filtro paso-alto. A la señal resultante, se le aplica la ventana de Hamming, que nos permitirá cortar la señal obteniendo trozos menores cuyos bordes no entorpezcan la extracción de características. A continuación, obtenida la ventana, pasamos el trozo de señal del dominio del tiempo al de la frecuencia gracias a la Transformada Rápida de Fourier o FFT (del inglés *Fast Fourier Transformation*), lo que nos proporciona un banco de filtros uniforme. El oído no discrimina todos los rangos de frecuencias por igual y es por esto que se aplica el banco de filtros correspondiente a la escala Mel. Para finalizar aplicamos el logaritmo para reducir la sensibilidad a los sonidos muy altos o muy bajos y seguidamente aplicamos la transformada discreta coseno o DCT (por sus siglas del inglés *Discrete Cosine Tranform*) al banco de filtros.

Como resultado del proceso descrito, observable en la Figura 4.1, obtendremos una serie secuencias de vectores que cuenta con una serie de parámetros configurables que explicaremos en la Subsección 5.2.1.

Figura 4.1: Proceso para la extracción de características del audio

4.2. iATROS

El reconocedor de voz que utilizaremos para nuestro proyecto es iATROS (del inglés *improved automatically trainable recogniser of speech*). Se trata de un sistema que puede ser utilizado para el reconocimiento de texto manuscrito y habla. Este nos provee de herramientas para el reconocimiento *offline* y *online* del habla basados en módulos ALSA, utilizando como núcleo una búsqueda similar a la de Viterbi en una red de modelos ocultos de Markov.

4.2.1. Modelo acústico

El habla se compone esencialmente por la sucesión de palabras que, en mayor o menor medida, tienen una cohesión y nos permiten transmitir ideas. Estas palabras a su vez podemos verlas como una serie de sonidos conectados que nos permiten articularlos de diferentes formas para formar palabras nuevas. Dichos sonidos, también conocidos como fonemas, deben ser modelados en nuestro sistema para así poder extraer características y posteriormente tener la capacidad de distinguir entre unos u otros.

En lo que a nuestro proyecto respecta, los fonemas han sido representados por Modelos de Markov Ocultos con salidas modeladas por una distribución probabilística formadas por una mixtura de 64 gaussianas para cada estado, obteniendo una relación entre fonemas y símbolos donde un fonema dado estará representado por un símbolo en particular. Por ejemplo, el fonema /v/ estará representado por la “b”. Cabe señalar que para el entrenamiento de nuestro sistema se utilizó el corpus fonético Albayzin [5], una base de datos para el reconocimiento del habla en español.

Para finalizar esta subsección, cabe resaltar, como vemos en el ejemplo del fonema /v/, que la transcripción fonética de una palabra dada no se deriva directamente de su grafía, sino que tendremos que interpretarla fonéticamente. Para llevar a cabo dicha tarea, disponemos de un *script* denominado `eutranscribe`, el cual, dada una palabra, nos proporciona su transcripción fonética.

4.2.2. Modelo léxico

Con respecto a la construcción de nuestro reconocedor del habla, en primer lugar tendremos que definir el dominio de clasificación que éste va a tener, es decir, qué palabras seremos capaces de reconocer. En nuestro caso este dominio versará sobre una serie de acciones definidas que hemos implementado para Félix, así como los números desde el 1 al 9, unidades de tiempo y el conector "y".

Definido el dominio y ayudados por el *script* mencionado en la subsección anterior `eutranscribe`, iremos obteniendo palabra a palabra su transcripción fonética y construiremos nuestro modelo léxico sobre el fichero `felix.lx`, en el que indicamos la palabra completa, la probabilidad (generalmente 1.0) y, finalmente, la transcripción fonética con los fonemas separados.

Así pues, una palabra la podemos considerar como un autómata en el que las transiciones son ejecutadas por los fonemas que la componen, donde el conjunto de estos autómatas formaría nuestro modelo léxico. Un ejemplo de estos autómatas sería el siguiente, representando la palabra "camina", la cual, al ser procesada por `eutranscribe` nos devuelve que su transcripción fonética es `kamina`:

Figura 4.2: Ejemplo de autómata de fonemas para el modelo léxico

4.2.3. Modelo del lenguaje

Por último, nos encontramos con el modelo del lenguaje. En éste definiremos todas las secuencias de frases aceptadas por nuestro reconocedor a partir de las palabras definidas en el modelo léxico. Al igual que el modelo léxico, el modelo del lenguaje está formado por un autómata finito determinista, conocido por sus siglas como AFD, con un único estado inicial y dos posibles estados finales, donde las transiciones entre dichos estados son las palabras reconocibles.

La Figura 4.3 muestra una simplificación del autómata que utilizamos para nuestro sistema:

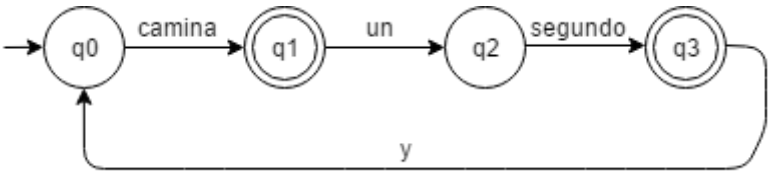

Figura 4.3: Simplificación del autómata utilizado en nuestro sistema

4.2.4. Configuración

Con lo referente a la configuración de nuestro sistema, contamos con una serie de parámetros que debemos variar, tanto para la adquisición y carga de nuestras frases como para la búsqueda a la hora de realizar predicciones. Éstos parámetros los encontramos en los siguientes ficheros:

- `felix.cnf`: Fichero de configuración para el reconocimiento. En él encontramos las rutas de nuestro reconocedor hacia los modelos acústicos, léxicos y del lenguaje, y una serie de parámetros utilizados para la búsqueda que iATROS realiza para el algoritmo de Viterbi. Algunos de sus parámetros que hemos necesitado modificar para optimizar la búsqueda (véase la Subsección 5.2.1) son los que exponemos en la Tabla 4.1:

Tabla 4.1: Descripción de algunos parámetros de configuración de iATROS

Parámetro	**Descripción**
`hmm`	Ruta relativa del fichero del modelo acústico.
`lexicon`	Ruta relativa del fichero del modelo léxico.
`lexicon-type`	Indicador del tipo de modelo léxico utilizado.
`grammar`	Ruta relativa del modelo del lenguaje.
`grammar-type`	Indicador del tipo de gramática utilizada.
`beam`	Dada una transición, si supera la puntuación actual más este valor, será eliminada. Criterio para la poda en la búsqueda.
`grammar-scale-factor`	A mayor valor, el reconocedor concederá mayor importancia al modelo del lenguaje que al modelo acústico.
`word-insertion-penalty`	A mayor valor favoreceremos las secuencias de palabras más largas.
`histogram-prunning`	Controla el tamaño del *heap*, que, si supera este valor, eliminará la hipótesis menos probable.

- `conf.feat`: Fichero de configuración para extraer características del audio. En éste encontramos los parámetros relacionados con la adquisición/carga y preprocesado de la señal de audio [10], de los cuales podemos resaltar los presentados en la Tabla 4.2:

Tabla 4.2: Parámetros relacionados con el tratamiento del audio de iATROS

Parámetro	**Descripción**
`SampleFrec`	Frecuencia de muestreo a la que realizamos las grabaciones.
`SilencesThreshold`	Umbral de energía por debajo del cual consideramos que hay silencio.
`SecondsSilence`	Segundos que han de transcurrir para considerar que existe un silencio.

5. IMPLEMENTACIÓN DEL TRABAJO

Para la implementación del trabajo se ha optado por la utilización de una placa de bajo coste denominada C.H.I.P. Esta placa apareció el siete de mayo de 2015 en la plataforma de financiamiento de proyectos Kickstarter, ofreciendo una placa más asequible que sus competidoras como la familia Raspberry y con unos atributos muy prometedores.

Por otra parte, para realizar un control de las versiones y tener copias de seguridad de los avances que vamos realizando con los cambios referentes a nuestro código fuente, hemos optado por la utilización de una plataforma de desarrollo software que nos permite esto de forma sumamente sencilla: GitHub. Aquí hemos creado un usuario con dos repositorios, donde cada uno nos sirve para almacenar el servidor y cliente web y otro para la parte del reconocedor y cliente Python. Cabe destacar la utilización del editor de código Visual Studio Code gratuito de Microsoft que nos ha facilitado el desarrollo gracias a su integración con GitHub.

5.1. C.H.I.P.

"C.H.I.P. nace de la necesidad de tener una placa que proporcionara menos problemas que sus coetáneos modelos de Raspberry Pi al integrarse con OTTO, una cámara que deseaba comercializar Next Thing Co antes que C.H.I.P. (Tony, Next Thing Co, correo electrónico, 12 de mayo de 2017)" Además, se distinguía de sus competidores por su bajo coste y contar con wifi y Bluetooth incorporado, sin necesidad de añadir dispositivos extra.

5.1.1. Descripción, ventajas y limitaciones

C.H.I.P. es una placa capaz tanto de realizar tareas para las que utilizamos un ordenador normalmente, como pueden ser navegar en internet, editar documentos o jugar a videojuegos, como de sernos útil para nuestros proyectos a la hora de controlar dispositivos como leds o servomotores, entre otros, que conectemos a través de la cabecera de pines.

Las principales ventajas de C.H.I.P. frente a sus competidores [7], y las cuales nos han hecho decantarnos por éste, han sido la conectividad, ya que C.H.I.P. cuenta con wifi y Bluetooth incorporado como comentamos anteriormente, cosa que en sus competidoras, como Raspberry Pi Zero, no ocurre, y debe añadirse a través de otros dispositivos como adaptadores wifi USB. Otro elemento que cabe destacar es que cuenta con un almacenamiento interno de 4GB sin necesidad de adquirir discos duros externos o tarjetas SD. También contaremos con 80 pines de entrada/salida de propósito general o GPIO (del inglés *General Purpose Input/Output*) que nos permitirán controlar todos los componentes de Félix que conectemos a C.H.I.P. Es por esto y su reducido coste por lo que finalmente decidimos utilizar esta placa en nuestro proyecto.

En su contra, nuestra placa no posee una comunidad de usuarios tan extensa y activa como la de la computadora Raspberry. Además, C.H.I.P. es un dispositivo basado en Unix, el cual puede obstaculizar la ejecución de nuestro código en cualquier momento para manejar una interrupción o realizar tareas administrativas, entre otras. Esto nos ha impedido la utilización del sensor de proximidad HC_SR04, ya que su funcionamiento se basa en el envío y recepción de ultrasonidos para posteriormente procesar cuánto se ha tardado en recibir el ultrasonido y estimar la proximidad del obstáculo que se tiene enfrente; pero si el sistema roba el control de la ejecución de este proceso, las medidas obtenidas no son lo suficientemente exactas.

5.1.2. Instalación y puesta a punto

Previamente a toda instalación de software sobre nuestra placa, tendremos que grabar la imagen que hemos utilizado para nuestro proyecto, que ha funcionado sin mayores problemas. Para hacer esto, la empresa desarrolladora de la placa, Next Thing Co, nos pone a disposición una herramienta web a la que debemos acceder desde un navegador Chrome. Así pues, accediendo a dicha herramienta ubicada en la página `https://flash.getchip.com/ls`, se nos pedirá instalar unos controladores para poder reconocer en nuestro sistema la placa correctamente. Finalmente, y tras seguir los pasos bien detallados de la página, llegaremos a una pantalla donde se nos pedirá seleccionar una imagen, que será la que se grabará en la placa. La imagen utilizada para el proyecto es “Headless kernel 4.4”.

Tras esto ya podremos conectarnos a Félix a través de conexión serie como se indica en su página de configuración [8]; aquí encontraremos cómo conectarnos a una red wifi y así poder acceder a Félix de forma remota mediante SSH. Esto nos facilitará mucho el trabajo, ya que, a la hora de asentar la placa en el interior de Félix, será más cómodo no tenerla presa de un cable conectado y poder así Félix moverse con libertad.

Una vez conectados a Félix y con conexión a internet, podemos comenzar con la instalación de todos los paquetes software que vamos a requerir para nuestro proyecto. Esto lo haremos a través del gestor de paquetes APT y NPM (del inglés Node Package Manager) el gestor de paquetes Javascript de NodeJS. Cabe destacar que hemos utilizado la versión de NodeJS 4.4.7, ya que ésta no nos ha ocasionado ningún problema con la instalación de dependencias. Para instalar dicha versión de NodeJS descargaremos la versión mediante la orden `wget https://nodejs.org/dist/v4.4.7/node-v4.4.7-linux-armv7l.tar.xz`. A continuación, deberemos instalarlo mediante `sudo tar -C /usr/local --strip-components 1 -xJf node-v4.4.7-linux-armv7l.tar.xz`.

Podemos separar la instalación en dos apartados diferentes dependiendo de sus propósitos:

- Reconocedor: Todos los paquetes relacionados con el reconocedor y la toma de audio para Félix. Los paquetes utilizados, instalados mediante el gestor APT, y una breve descripción de estos los podemos ver en la Tabla 5.1.
- Servidor: Aquí se encuentran los paquetes referentes a la instanciación del servidor y el manejo de los componentes de nuestro robot (véase la Tabla 5.2).

Tabla 5.1: Paquetes necesarios para el funcionamiento del reconocedor

Paquete	**Descripción**
flex / bison	Utilizados en el procesado de los ficheros de configuración y los modelos del reconocedor.
cmake	Generará ficheros para la compilación.
rpm	Herramienta de administración de paquetes necesaria para la puesta a punto del reconocedor.
pkg-config	Provee una interfaz unificada para llamar bibliotecas instaladas a la hora de la compilación de nuestro reconocedor.
build-essential	Incluye todos los paquetes necesarios para compilar paquetes Debian. Incluye los compiladores gcc y gcc+, además de librerías y otras utilidades.
fftw3 / fftw3-dev / libfftw3 / libfftw3-dev / libasound2-dev	Para el procesado de la señal de audio por el reconocedor.
python / python3 / python-pandas	Nos permitirá ejecutar nuestros programas escritos en el lenguaje Python y que manejan la lógica del reconocimiento en Félix.
valgrind	Conjunto de herramientas libres que ayuda en la depuración de problemas de memoria y rendimiento de programas. Utilizado a la hora de compilar el código del reconocedor.
subversion / git	Para el control de versiones de nuestro código.
sox	Toma y manipulación de los ficheros de audio que grabará nuestro reconocedor.

Tabla 5.2: Paquetes necesarios para el funcionamiento del servidor

Paquete	**Gestor**	**Descripción**
xz-utils	APT	Nos permitirá descomprimir correctamente el paquete de NodeJS.
bower	NPM	Para instalar las versiones de los paquetes que requerimos y sus dependencias.
serialport	NPM	Necesario para el paquete de Johnny-five y chip-io, nos provee una interfaz para manipular los puertos de C.H.I.P.
Johnny-five / chip-io	NPM	Nos proporcionan una forma sencilla de manipular los componentes conectados a Félix.
express / http / socket.io	NPM	Necesarios para instanciar nuestro servidor y crear sockets para recibir y mandar órdenes.

Ya tenemos todo el software necesario para ejecutar nuestro analizador, servidor y cliente, pero nos queda descargar el código fuente de éstos. Para ello, únicamente tendremos que clonarlo de nuestros repositorios, mediante las ordenes que Github nos provee, y ya tendremos todo lo necesario para que Félix comience a funcionar.

5.1.3. Gestión del audio

La toma y tratamiento de audio en nuestro proyecto es un aspecto esencial, donde, en primer lugar, tendremos que analizar nuestras necesidades para optar por un software de captación y tratamiento de audio u otro. Contamos con un sistema, con lo referente al audio, cuya principal característica es la capacidad de ser capaces de reconocer cuándo el interlocutor emite una frase y cuándo éste la finaliza, de forma que será entonces cuando pasemos a analizarla. Podemos afirmar que necesitamos una herramienta software que nos permita comenzar a grabar y, al captar estos silencios, finalizar la grabación, obteniendo entonces una palabra o serie de estas que están conectadas por pausas lo suficientemente breves como para poder afirmar que constituyen una misma frase. Además, nos exponemos a un medio ruidoso, principalmente causado por la fricción de los motores.

Tras buscar entre las diferentes opciones que Linux nos brinda, hemos optado por la utilidad multiplataforma SoX. Esta tiene una ventaja crucial para nuestro objetivo de toma de frases ante sus competidores, y es la opción de detectar silencio durante la grabación de un fichero de audio como necesitamos. Esto lo haremos gracias a su parámetro `silence` [12], que nos da la posibilidad de establecer un umbral para comenzar o terminar la grabación y cuánto tiempo de retraso queremos al empezar de grabar o parar. Otro problema que SoX nos soluciona es la eliminación de ruido de fondo. Gracias a las opciones `noiseproof` y `noisered` podemos tomar una muestra del ruido generado por el cuadrúpedo para posteriormente generar un perfil de audio que utilizaremos para eliminar el ruido en cada toma de audio.

5.2. Ajuste del sistema

Cada proyecto es diferente del resto, cada sistema tiene unas necesidades diferentes. Con lo que respecta a la parte del reconocedor para nuestro sistema, por ejemplo, contamos con un dominio de palabras a reconocer bastante pequeño, donde nuestro objetivo es el de poder reconocer habla continua, frases enteras para poder analizarlas en un entorno con un ruido de fondo continuo generado por los motores. Este sistema pues, dista bastante de un sistema utilizado para transcribir habla continua a texto, donde contamos con centenares de miles de posibles palabras a reconocer y quizás la tolerancia a un ruido continuo sea menor.

Es por esto que es vital ajustar nuestro sistema, adaptarlo a nuestras necesidades, a las características de éste. Para ello, analizados los parámetros que debemos modificar, será relativamente sencillo estudiar cómo se comporta éste al variar el valor de dichos parámetros. Con este fin, deberemos crear un corpus de prueba e idear una forma de automatizar la búsqueda de los parámetros que mejoren el reconocimiento en nuestro sistema.

5.2.1. Toma de muestras y optimización de parámetros

Con el objetivo de minimizar la tasa de error de nuestro sistema, iATROS, como introdujimos en la Subsección 4.2.4, cuenta con una serie de parámetros que nos permiten configurar el reconocedor y tendremos que ir variando para ver cuál es su impacto en la tasa de acierto, y poder así quedarnos con unos u otros valores.

Para ir probando dichos valores, en primer lugar debemos construir un corpus de muestras sobre el que aplicar estos parámetros, y que así los resultados sean más fiables y flexibles a las diferentes entradas que podemos obtener. Esto lo haremos gracias al *script* `felix_inifite_recorder.py` que nos permite realizar grabaciones de forma sencilla mediante SoX con una frecuencia de muestreo de 16 kHz, como especificamos para el

parámetro `SampleFrec` en el archivo de configuración `conf.feat` (véanse las Subsecciones 5.1.3 y 4.2.4), e ir almacenándolas para el posterior entrenamiento.

A continuación, para la obtención del valor para los parámetros que mejore nuestra tasa de acierto, hemos creado un *script* al que hemos llamado `felix_adjuster.py`. Éste realiza un barrido sobre aquellos que deseemos modificar para observar cómo se comporta el reconocedor. Estos parámetros a variar son `beam`, `gramar-scale-factor` y `word-insertion-penalty` y, esencialmente, el funcionamiento de dicho *script* es el siguiente:

- En primer lugar, preguntamos por la ruta relativa donde se encuentra el corpus de entrenamiento.
- Si éste contiene ficheros de audio, entonces pediremos al usuario que introduzca los rangos para los que quiere variar cada parámetro, así como los saltos de valores entre el valor mínimo y máximo del rango proporcionado.
- A continuación, se realizará un barrido en el que recorreremos todas las posibles combinaciones de valores dados los rangos para los parámetros y sus saltos, de forma que, para cada iteración, calcularemos los coeficientes cepstrales de cada uno de los ficheros de audio y se los pasaremos al reconocedor.
- Éste nos devolverá la predicción y entonces tendremos que determinar si el análisis ha sido exitoso o no. Para ello hemos optado por el etiquetado de los ficheros a través de su nombre para conocer cuál es su valor.
- Tras analizar el corpus por completo, almacenaremos cuáles son los valores de los parámetros actuales y cuál es la tasa de acierto global obtenida para nuestro corpus. Cabe mencionar que vamos a extraer dos tasas de acierto, una a nivel léxico, donde la frase tiene que estar perfectamente predicha, y una tasa de acierto a nivel semántico, donde daremos por buenas frases donde “segundo” se haya fallado y predecimos “segundos” o “minuto”, “minutos”.
- Tras finalizar el barrido, obtendremos una colección de elementos compuestos por los valores de los parámetros y la tasa de acierto mencionadas, donde únicamente tendremos que buscar cuáles son los valores de los parámetros para el elemento con mayor tasa de acierto semántico.

Tras crear nuestro corpus y expuesto el funcionamiento de nuestro entrenador, en lo que a Félix respecta, hemos realizado un barrido con rangos para el `Beam` definidos entre 100 y 580, con saltos de 20 en 20, rangos para `Grammar-Scale-Factor` entre 1 y 49 con saltos de 4 en 4, y rangos para `Word-Insertion-Penalty` entre -20 y 20 con saltos de 5 en 5, realizando así un total de 2925 iteraciones. En la Tabla 5.3 podemos observar parte de estos resultados, así como en la Tabla 5.4 la combinación de valores que maximizan la tasa de acierto semántico.

Cabe mencionar que, como es usual en el ajuste de parámetros, hemos realizado múltiples barridos con rangos de valores más cerrados cercanos a los que han obtenido mejores porcentajes de acierto, pero, para nuestro caso no se han encontrado configuraciones que mejoraran significativamente la tasa de acierto y se ha optado por no incluir dichas búsquedas.

Tabla 5.3: Tasas de acierto del barrido de ajuste de parámetros

			Word-Insertion-Penalty			
		GSF	-20	-10	10	20
Beam	100	15	62.2	63.1	60.5	59.2
		30	55.6	58.1	58.1	57.7
	300	15	54.4	55.4	54.2	50.5
		30	55.7	55.3	52.8	52.5
	450	15	40.6	41.6	38.7	38.6
		30	38.3	40.7	39.6	33.7

Tabla 5.4: Parámetros que maximizan la tasa de acierto de nuestro sistema

Beam	100
Grammar-Scale-Factor	15
Word-Insertion-Penalty	-10
Tasa de acierto	63.1%

5.2.2. Gestión del reconocimiento

Aunque en la Subsección 3.3.1 hemos explicado levemente la lógica para con lo referente al reconocimiento del habla, este es uno de los aspectos principales de nuestro proyecto y por esto ahondaremos más en esta subsección. Para ello explicaremos paso a paso desde la toma del audio hasta como lo procesamos para que Félix actúe en consecuencia. Tras establecer la conexión con Félix, que explicaremos en la Subsección 5.2.3 con más detalle, realizaremos en bucle hasta que detengamos el *script* o se corte la conexión establecida con Félix lo siguiente:

1. Gracias a los mecanismos que SoX nos proporciona para la toma del audio, tomaremos una frase invocando un proceso que grabe hasta que detectemos un silencio de 1,45 segundos, valor que hemos obtenido tras la realización de múltiples pruebas hasta llegar a un valor que funcionara bien con las características del medio (para más información relacionada con SoX véase la Subsección 5.1.3).
2. Comprobaremos si está activo el proceso encargado de la toma de cepstrales para generar la predicción, lanzándolo si no lo está. Dicho proceso se queda esperando la aparición de un archivo que contenga los cepstrales en una ruta determinada para así generar las predicciones.
3. Dado el fichero que almacena la frase grabada e iniciado el proceso encargado de realizar el reconocimiento, generaremos los cepstrales a partir del fichero con la frase.
4. El reconocedor detectará la aparición del archivo con los cepstrales a reconocer y sobre el proceso que maneja nuestro reconocedor se imprimirá la predicción encapsulada por los delimitadores <s> y </s>. En este punto, nuestro *script* Python estará constantemente leyendo los cambios en la salida del proceso con el reconocedor y cuando detecte la aparición de los limitadores mencionados, tomara la frase que retienen.

5. Dadas las acciones a realizar solo nos faltaran tratarlas. Para ello nos valemos de la potencia y simpleza de Python para tratar cadenas ya que, conociendo el conector entre ordenes, somos capaces de enviar a nuestro servidor (que es quien maneja la parte mecánica de Félix), las acciones únicamente, realizando la lógica de las esperas de tiempo y cómo debemos actuar en Python.

5.2.3. Envío y tratamiento de ordenes

El envío de órdenes, con lo que respecta al reconocimiento del habla, se realiza de manera unidireccional del cliente al servidor, donde en este último es donde es donde procesaremos cada acción.

El envío de ordenes encapsula nuestra lógica para el reconocimiento del habla (véase la Subsección 5.2.2). Es posible gracias a la librería Socket.io, explicada en la Subsección 3.1.2, que nos permitirá abrir una conexión TCP entre nuestro cliente y el servidor. Será entonces cuando, tras realizar la lógica necesaria y obtenida la orden, enviaremos un mensaje al servidor con la orden concreta que debe realizar Félix, indicando que debe disparar el evento llamado `voiceCommand`. Cuando se lanza dicho evento en el lado servidor, se analiza que orden es simplemente a través del texto recibido y en consecuencia se lanzan unos métodos, que únicamente conllevan una serie de transiciones de poses, u otros hasta que recibamos nuevas órdenes.

6. CONCLUSIONES Y PROPUESTAS DE MEJORA

Dado un tiempo determinado para realizar el proyecto, al cual hay que ceñirse casi estrictamente para no comprometer otras tareas de importancia, es comprensible que se nos ocurran formas en las que podríamos extender la funcionalidad que Félix ofrece, así como aspectos que podríamos mejorar, pero que por desgracia o porque podríamos realizar un proyecto entero con estas ideas, no podemos llevar a cabo. Así pues, en el presente capítulo trataremos un par de propuestas que mejorarían la experiencia de uso y ampliarían la funcionalidad de nuestro cuadrúpedo.

6.1. Conclusiones

Para la realización de este trabajo hemos visto como, aunque estemos tratando con un proyecto relativamente pequeño, ha sido necesario abordar un gran espectro de competencias relacionadas con los estudios en el grado de Ingeniería Informática. Nos ha servido para afianzar conocimientos y darnos cuenta de la importancia que tiene plantear un proyecto y cada punto de éste antes de empezar a realizarlo.

Hemos obtenido experiencia en el modelado de una solución real, infravalorado tal vez en un principio por no haber realizado un proyecto de tal envergadura. Resaltando en todo momento el interés que éste me despertaba por investigar y seguir trabajando en él.

Respecto a la evolución del trabajo, cabe destacar que el arranque de este ha sido un poco lento al principio, principalmente por la utilización de la placa C.H.I.P., lo que en un primer lugar nos llevó a la decepción y el pensamiento de que tal vez no conseguiríamos tener acabado el proyecto en el plazo determinado. Pero poco a poco, tras ir solucionando los pequeños problemas y ver como comenzábamos a tener resultados, nos impulsaba a seguir continuando.

Finalmente, cabe destacar la importancia de la implantación del reconocimiento de voz en el sistema, ya que es éste quien dota realmente de funcionalidad a nuestro robot y lo hace más interesante al usuario.

6.2. Diseñador de animaciones

Una tarea que ha requerido más tiempo del que hubiésemos deseado ha sido la de articular la secuencia de movimientos que Félix debe realizar para obtener las diferentes poses. Para esto contamos con una pequeña sección en nuestro cliente web donde éramos capaces de determinar los ángulos que las caderas y rodillas debían poner para, posteriormente, ir anotándolos y formar una secuencia que nos llevara a una de las poses predefinidas. Este proceso se podría agilizar en gran medida si construyéramos una aplicación en la que se nos presentara una imagen tridimensional del robot y fuéramos capaces de, con ayuda del cursor, mover cada una de las caderas y rodillas de Félix y guardar estas posiciones, creando una lista que posteriormente podríamos ejecutar.

6.2. Félix y IoT

Además de utilizar a Félix como nuestra mascota personal, podemos dotarla de nuevas funcionalidades relacionadas con lo que se conoce como internet de las cosas o IoT (del inglés *Internet of Things*). Esto lo podríamos realizar de forma relativamente sencilla a través de un servicio web ofrecido por Amazon, Alexa [6]. Gracias a éste podremos realizar todo tipo de peticiones mediante la voz a nuestro cuadrúpedo como podrían ser encargar una pizza, reproducir canciones si le conectamos unos altavoces, o que apagara las luces (siempre y cuando contemos con bombillas adecuadas, como las Hue de Philips capaces de conectarse a la red wifi [9]).

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

BIBLIOGRAFÍA

[1] BBC News. *Meet the robotic cat for the elderly*. http://www.bbc.com/news/technology-35310200 [Consultado: 26 de mayo de 2017].

[2] BQ. Zowi, el robot inteligente y educativo para niños. https://www.bq.com/es/zowi [Consultado: 28 de noviembre de 2016]

[3] Business Insider. *Growth Statistics For Robots Market.* http://www.businessinsider.com/growth-statistics-for-robots-market-2015-2 [Consultado: 25 de mayo de 2017].

[4] ESTÁNDAR ETSI. *Speech Processing, Transmission and Quality Aspects*. ETSI ES 202 050 V1.1.5, 2007.

[5] Liceu. *Albayzín, Base de datos para el reconocimiento del habla en español.* http://liceu.uab.cat/~joaquim/language_resources/spoken_res/Corp_oral_esp.html [Consultado: 28 de mayo].

[6] LifeHacker. *How to Build Your Amazon Echo with a Raspberry Pi.* http://lifehacker.com/how-to-build-your-own-amazon-echo-with-a-raspberry-pi-1787726931 [Consultado: 1 de junio de 2017].

[7] Makezine. *C.H.I.P. vs Pi Zero: Which Sub-$10 Computer Is Better?* http://makezine.com/2015/11/28/chip-vs-pi-zero/ [Consultado: 25 de octubre de 2016].

[8] Next Thing Co. *Next Thing Co Documentation*. https://docs.getchip.com/chip.html [Consulta: 10 de diciembre de 2016].

[9] Philips. *Philips Hue.* http://www2.meethue.com/es-es [Consultado: 1 de junio de 2017].

[10] PRHLT. *Audio Notes.* https://www.prhlt.upv.es/software/iatros/doc/speech/audio.html [Consultado: 22 de mayo de 2017].

[11] PRHLT. *Technical Documentation*. https://www.prhlt.upv.es/software/iatros/doc/speech/technical.html [Consultado: 21 de mayo de 2017].

[12] Sourceforge. *SoX*. http://sox.sourceforge.net/sox.html [Consultado: 12 de abril de 2017].

[13] Wikipedia. *Amazon Echo.* https://es.wikipedia.org/wiki/Amazon_Echo [Consultado: 25 de mayo de 2017].

[14] X. D. Huang, Y. Ariki, M. A. Jack1 1990. *Hidden Markov Models for Speech Recognition*. Edinburgh University Press.

[15] 101Hero. *3D Printers & More*. http://www.101hero.com/ [Consultado: 3 de octubre de 2016].