Universidad Carlos III de Madrid

INGENIERÍA TÉCNICA INDUSTRIAL:
ELECTRÓNICA INDUSTRIAL

PROYECTO FIN DE CARRERA

Dpto. de SISTEMAS Y AUTOMÁTICA

SÍNTESIS DE VOZ Y RECONOCIMIENTO DEL HABLA. IMPLEMENTACIÓN EN EL ROBOT HOAP-3

Autor:	Pablo Marín Plaza	p.marin.plaza@gmail.com
Tutores:	Daniel Hernández García	dhgarcia@ing.uc3m.es
	Miguel González-Fierro	mgpalaci@ing.uc3m.es

NOVIEMBRE 2011

TITULO: SÍNTESIS DE VOZ Y RECONOCIMIENTO DEL HABLA.
IMPLEMENTACIÓN EN EL ROBOT HOAP-3

AUTOR: PABLO MARÍN PLAZA

TUTORES: DANIEL HERNÁNDEZ GARCÍA
MIGUEL GONZÁLEZ-FIERRO

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 14 de Noviembre de 2011 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

SECRETARIO VOCAL PRESIDENTE

AGRADECIMIENTOS

Dedico este proyecto a mis padres por el apoyo y la confianza que siempre me han dado.

A mis tutores por la paciencia que han tenido y las habilidades que me han enseñado.

RESUMEN

La interacción Humano-Robot es una parte casi tan importante en la creación de los robots como el diseño físico o la programación asociada. Por esta razón surge la necesidad de hacer de esta interacción lo más sencilla y amistosa posible.

Desde la antigüedad, la forma de comunicación utilizada por el ser humano, ha sido mediante sonidos. Estos sonidos se han convertido a lo largo de los años en un sistema complejo con reglas, lógica y palabras para describir el mundo que rodea a los humanos: “EL LENGUAJE”.

Por esta razón, la manera más sencilla de comunicación es mediante la voz. Sin necesidad de una interfaz complicada o un teclado y ratón, la voz se convierte en la herramienta principal de comunicación entre los humanos y los robots.

Gracias al avance tecnológico, hoy en día el ser humano es capaz de dar órdenes a los robots mediante la voz y hacer que los robots se comuniquen mediante la síntesis de voz.

La síntesis de voz consiste en la creación de ondas de sonido artificiales semejantes al habla humana.

El reconocimiento del habla es un sistema capaz de transcribir un mensaje de voz en texto independientemente del hablante. Se basa en la comparación con un modelo acústico recogido en una base de datos.

En este proyecto se ha conseguido la implementación de ambos sistemas (reconocimiento del habla y síntesis de voz) en el robot humanoide HOAP-3.

ABSTRACT

Human-Robot Interaction is almost as important part in developing robots like the physical design or programming associated. For this reason there is a need to make this interaction as simple and user friendly as possible.

Since ancient times, the simplest form of communication used by humans has been using sounds. These sounds have become over the years into a complex system with rules, logic and words to describe the world around humans, "LANGUAGE".

For this reason, the easiest way of communication is by voice. No need for a complicated interface or a keyboard and mouse, the voice becomes the main tool of communication between humans and robots.

Thanks to technological advances, today the human being is able to give commands to the robot by voice and make the robots communicate using speech synthesis.

Speech synthesis (Text-to-speech) is the creation of artificial sound waves similar to human speech. It is based on the concatenation of sound units to build the voice.

Speech recognition is a system capable of transcribing a voice message in text independently of the speaker. It is based on comparison with an acoustic model collected in database.

This project has achieved the implementation of both systems (speech recognition and speech synthesis) in the humanoid robot HOAP-3.

ÍNDICE DE CONTENIDO

1 INTRODUCCIÓN 1

1.1.- INTERACCIÓN HUMANO-ROBOT 1
1.2.- MOTIVACIÓN 1
1.3.- OBJETIVOS 1
1.4.- ESTRUCTURA DEL DOCUMENTO 2

2 FUNDAMENTOS TEÓRICOS 3

2.1.- FUNDAMENTOS PREVIOS DEL SONIDO 3
2.2.- SÍNTESIS DE VOZ 7
2.2.1.- COMPOSICIÓN: 7
2.2.2.- HISTORIA: 7
2.2.3.- SISTEMAS DE SÍNTESIS DE VOZ: 9
2.2.4.- TIPOS DE SÍNTESIS DE VOZ: 10
2.2.4.1.- CONCATENATIVA 10
2.2.4.2.- POR FORMANTES 12
2.3.- RECONOCIMIENTO DEL HABLA 13
2.3.1.- HISTORIA 13
2.3.2.- TIPOS DE RECONOCIMIENTO DEL HABLA 14

3 ESTUDIOS REALIZADOS 16

3.1.- SÍNTESIS DE VOZ 16
3.1.1- SOFTWARE DE LIBRE DISTRIBUCIÓN 16
3.1.1.1.- The Festival Speech Synthesis System 16
3.1.1.2.- Microsoft Speech Application Programming Interface (SAPI) 17
3.1.1.3.- Mbrola 17
3.1.1.4.- Gnuspeech 18
3.1.2.- SOFTWARE CON LICENCIA COMERCIAL 18
3.1.2.1.- Loquendo TTS 18
3.1.2.2.- SodelsCot 19
3.1.2.3.- Cepstral tts voices 19
3.1.2.4.- AT&T Natural Voices 20
3.1.2.5.- Acapela group 21
3.1.2.6.- TextAloud 21
3.1.2.7.- VerbioTTS 22
3.1.2.8.- FonixTalk 6.1 22
3.2.- RECONOCIMIENTO DEL HABLA 23
3.2.1.- SOFTWARE DE LIBRE DISTRIBUCIÓN 23
3.2.1.1.- CMU Sphinx 23
3.2.1.2.- API Android 24
3.2.2.- SOFTWARE CON LICENCIA COMERCIAL 24
3.2.2.1.- FonixVoiceIn 24
3.2.2.2.- Loquendo ASR speech recognition 25
3.2.2.3.- DragonNaturallySpeaking 26

4 IMPLEMENTACIÓN ROBOT HUMANOIDE HOAP-3 27

4.1.- ROBOT HOAP-3 27
4.2.- PROGRAMAS 32
4.2.1.- SÍNTESIS DE VOZ 32
4.2.2.- RECONOCIMIENTO DEL HABLA 36

5 RESULTADOS EXPERIMENTALES 38

5.1.- DEMOSTRACIÓN 38
5.1.1.- SINTESIS DE VOZ 38
5.1.2.- RECONOCIMIENTO DEL HABLA 39
5.2.- ACIERTOS/ERRORES 44
5.2.1.- ANÁLISIS DE PALABRAS CLAVE (SIN FSG) 45
5.2.2.- ANÁLISIS DE FRASES (CON FSG) EXCLUYENDO NÚMEROS 46
5.2.3.- ANÁLISIS NÚMEROS (SIN FSG) 47
5.2.4.- ANÁLISIS DE ORDENES FINAL (CON FSG) INCLUYENDO NÚMEROS. 49

6 CONCLUSIONES 50

6.1.- OBJETIVOS CUMPLIDOS 51
6.2.- OBJETIVO FINAL DEL RECONOCIMIENTO DEL HABLA Y SÍNTESIS DE VOZ. 52

7 REFERENCIAS 53

8 ANEXOS 54

8.1.- PRESUPUESTO 54
8.2.- CÓDIGO FUENTE DEL PROGRAMA DE RECONOCIMIENTO DEL HABLA 57
8.3.- ARCHIVO FSG (FINITE STATE GRAMMAR) 64

ÍNDICE DE FIGURAS

FIG. 2.1.1 ONDA ACÚSTICA "EL ROBOT HUMANOIDE FUE CAPAZ DE HABLAR"..... 3
FIG. 2.1.2 SONIDO SONORO..... 4
FIG. 2.1.3 SONIDO SORDO..... 4
FIG.2.1.4 DISTINCIÓN "F" VS "U"..... 4
FIG. 2.1.4 ONDA DE SONIDO PALABRA "SOY" 5
FIG. 2.3.2 MODELOS OCULTOS DE MARKOV 15
FIG. 4.1.1 ROBOT HUMANOIDE HOAP-3 27
FIG. 4.1.2 GRADOS DE LIBERTAD..... 28
FIG. 4.1.3 SENSORES HOAP-3. 30
FIG. 4.1.4 BATERÍA Y CPU..... 31
FIG. 5.1.1 ARQUITECTURA..... 41

ÍNDICE DE TABLAS

TAB. 2.1.1 VOCALES-FORMANTES. 6
TAB. 2.2.4.2 PARÁMETROS DE LA SÍNTESIS DE VOZ 12
TAB. 4.2.1.1 FRASES SINTETIZADAS 34
TAB. 5.1.2 ÓRDENES LINGUÍSTICAS 42
TAB. 5.1.3 DETECCIÓN DE PALABRAS 43
TAB. 5.2.1 ANÁLISIS PALABRAS CLAVE 45
TAB. 5.2.2 ANÁLISIS FRASES 46
TAB. 5.2.3 ANÁLISIS NÚMEROS 47
TAB. 5.2.4 ANÁLISIS ÓRDENES COMPLETAS 49

1 INTRODUCCIÓN

1.1.- INTERACCIÓN HUMANO-ROBOT

Desde el origen de la era de las máquinas, se ha necesitado comunicarse con los robots para que obedecieran determinadas órdenes.

Al principio, los robots sólo seguían movimientos preestablecidos, haciendo de la interacción humano-robot muy limitada.

Más adelante, con los avances de programación, se necesitaban establecer unas comunicaciones básicas entre el humano y el robot. Estas interacciones por lo general eran pulsando botones o mediante determinados sensores.

Actualmente, la interacción humano-robot (HRI) se ha convertido en una nueva área interdisciplinar que tiene como objetivo modelar y predecir las relaciones entre humanos y robots.

1.2.- MOTIVACIÓN

La idea de trabajar con la síntesis de voz y el reconocimiento del habla, surge a partir de querer interactuar con los robots de una manera más natural. Gracias al avance tecnológico, hoy día el ser humano es capaz de dar órdenes a los robots mediante la voz y hacer que los robots, se comuniquen mediante la síntesis de voz.

En este proyecto se van a exponer las posibilidades que existen de implementar este tipo de comunicación e interacción con los robots.

1.3.- OBJETIVOS

- *ESTUDIO DE SÍNTESIS Y RECONOCIMIENTO DEL HABLA*

Para la primera parte, se realiza un estudio del arte sobre los principales programas que existen tanto de reconocimiento del habla como de síntesis de voz resaltando las cualidades y características de cada uno y dando una valoración.

- *IMPLEMENTACIÓN DE SÍNTESIS DE VOZ*

En la segunda parte, se realizará la implementación de la síntesis de voz mediante el programa llamado FESTIVAL SPEECH SYNTHESIS SYSTEM [1]. Será el encargado de poner voz al robot y expresar lo que el robot tenga que decirnos.

- *IMPLEMENTACIÓN DE RECONOCIMIENTO DEL HABLA*

Dentro de la segunda parte, se continuará con la implementación del sistema de reconocimiento del habla gracias al programa POCKETSPHINX [2]. Este programa realizará la transcripción de las ondas de sonido de la voz a cadenas de caracteres reconocibles por el lenguaje máquina.

- *DEMOSTRACIÓN*

En la tercera parte del proyecto, se hará una breve demostración de ambos sistemas sobre el robot humanoide HOAP-3 y se explicará en que consiste la interacción.

1.4.- ESTRUCTURA DEL DOCUMENTO

Se ha estructurado el proyecto en cuatro partes.

La primera parte intenta introducir al lector en los fundamentos básicos de los sistemas de interacción humano-robot mediante la voz. Haciendo un estudio de las ondas de sonido, del tratamiento de la señal y del procesado tanto para el reconocimiento como para la síntesis de voz.

La segunda parte hace un estudio de los programas que existen actualmente en el mercado y a libre disposición tanto de síntesis como de reconocimiento del habla.

La tercera parte se basa en la implementación de ambos sistemas sobre un robot concreto llamado HOAP-3 donde se pasará a describirlo brevemente y expondremos los resultados experimentales en una demostración.

La cuarta parte se ha dedicado a exponer las conclusiones finales de haber hecho este proyecto.

2 FUNDAMENTOS TEÓRICOS

2.1.- FUNDAMENTOS PREVIOS DEL SONIDO

La voz es una onda de presión acústica producida por el aparato fonador de los humanos. Este aparato fonador se compone entre otras partes anatómicas del aire que sale de los pulmones, pasando por unas cuerdas que vibran provocando distintos sonidos. Gracias a estas cuerdas vocales y a los distintos sonidos que se pueden producir, el ser humano es capaz de comunicarse siempre y cuando se produzcan de manera ordenada y siguiendo unas normas. Según sean estas normas y sonidos se genera un lenguaje u otro.

Los sonidos de la voz humana se pueden clasificar en sonidos sonoros y sonidos sordos [3].

La siguiente gráfica Fig. 2.1.1 muestra una onda de presión acústica producida al decir la frase “El robot humanoide fue capaz de hablar”

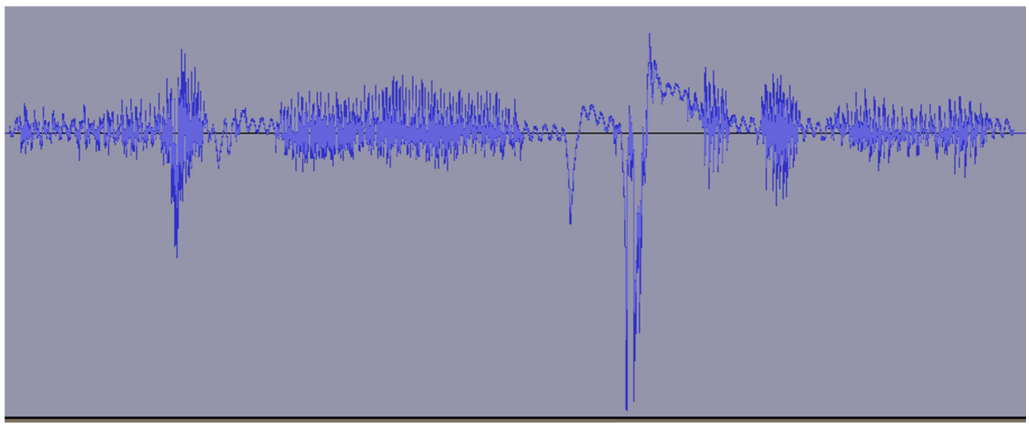


Fig. 2.1.1 Onda acústica “El robot humanoide fue capaz de hablar”

Sonidos sonoros: el aire pasa a través de las cuerdas vocales desde los pulmones provocando unas vibraciones de éstas y produciendo sonidos de una frecuencia entre los 300Hz y los 4000Hz. No existen impedimentos de ningún tipo al paso del aire. Por lo general son sonidos más armónicos y presentan periodicidad temporal. En la Fig. 2.1.2 [27] se muestra el tipo de sonido sonoro:

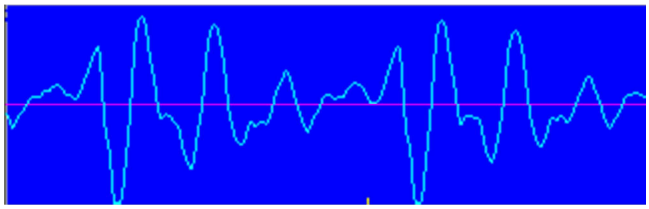


Fig. 2.1.2 Sonido sonoro

Sonidos sordos: las cuerdas vocales permanecen rígidas sin vibrar. El paso del aire está restringido y provoca sonidos de menor amplitud, más ruidosos que los sonidos sonoros. También se conocen como señales fricativas. Se caracterizan por su baja energía y su gran parecido al ruido blanco como se muestra en la Fig. 2.1.3 [28]:

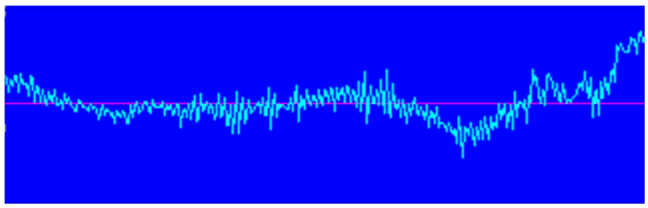


Fig. 2.1.3 Sonido sordo

Para comprobar estos dos tipos de sonidos, basta con tocar suavemente la nuez con los dedos y pronunciar el sonido de la “u”. Acto seguido podemos pronunciar el sonido de la “f”. Se puede apreciar que al pronunciar la “u”, las cuerdas vocales vibran. Esta vibración provoca una frecuencia fundamental llamada pitch y unos armónicos que se repiten de manera equidistante a lo largo de la duración del sonido. En cambio para el sonido de la “f”, al no haber vibración de las cuerdas vocales, no se aprecia el pitch. En la Fig. 2.1.4 [29], se puede apreciar el sonido sordo de la letra “f” y el sonido sonoro de la letra “u”.

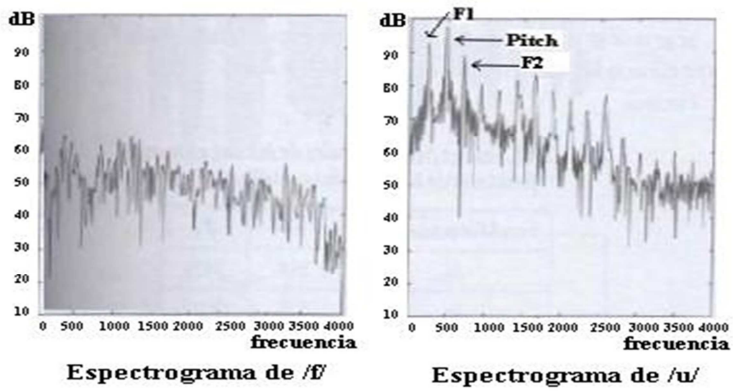


Fig.2.1.4 Distinción “f” vs “u”

Se puede observar otro ejemplo en la Fig. 2.1.4 [30]:

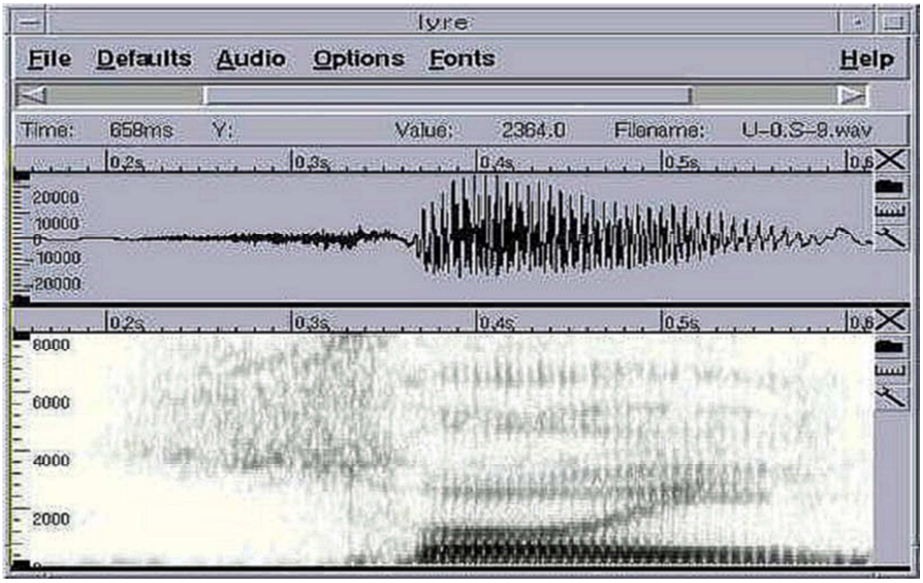


Fig. 2.1.4 Onda de sonido palabra "soy"

Este caso Fig. 2.1.4 corresponde a la onda acústica de la palabra "soy". Desde los 0.25s hasta los 0.35s corresponde con el sonido sordo o fricativo del fonema específico de la letra /s/. Desde los 0.36s hasta los 0.48s se aprecia el sonido sonoro y periódico del fonema de la letra /o/ que es más enérgico que el fonema de la letra /i/ que empieza a partir del 0.48s. Se puede distinguir dos picos en cada fonema sonoro de la /o/ y de la /i/ correspondientes a los formantes (F1 es la frecuencia fundamental y F2 es el primer armónico).

Es posible hacer una distinción para el sonido que producen los hombres y las mujeres. Normalmente, los hombres tienen una frecuencia fundamental (pitch) del orden de entre 50 y 250 Hz y las mujeres del doble, entre 100 y 500 Hz. Por esta razón, las mujeres tienen el tono de voz más agudo.

En la Fig. 2.1.4 también puede apreciarse que hay zonas enfatizadas llamadas resonantes. Estas zonas componen los sonidos de cada fonema que son únicos para cada lenguaje. Pueden sintetizarse mediante filtros. Estas frecuencias características se denominan formantes (o frecuencias formantes) y se conocen por F1, F2, F3, etc. partiendo de la primera frecuencia de resonancia y continuando de forma ordenada hacia frecuencias más elevadas [4].

En el idioma español, las cinco vocales tendrían sus primeros formantes de la siguiente manera Tab.2.1.1

Tab. 2.1.1 Vocales–Formantes.

Vocal\Formante (Hz)	F1	F2
/i/	284	2430
/e/	527	2025
/a/	689	1458
/o/	608	1215
/u/	243	770

De esta manera, el sonido de las vocales se compone de una frecuencia baja F1 y de una frecuencia alta F2 (dependiendo de cada vocal).

Para que estos sonidos tengan algún sentido, dentro de un mismo idioma, siguen unas reglas gramaticales. Al conjunto de sonidos que produce un humano de forma lógica y siguiendo estos patrones y reglas, se le conoce como la voz o el habla.

Las ondas de sonido son muy diversas. Esto se debe a que cada persona tiene sus propias cuerdas vocales, haciendo variar la frecuencia de la onda de sonido (voz grave o aguda), el timbre o la entonación. Estas características hacen de cada onda de sonido única para cada individuo, incluso sufriendo ligeras variaciones dependiendo del estado anímico de la persona.

Para poder establecer una comunicación, es necesario que una de las partes hable (emita sonidos siguiendo unas reglas gramaticales lógicas) y al menos otra parte escuche. La escucha se realiza por medio de los oídos y mediante un sistema de huesos que van transmitiendo vibraciones hasta transformar esas vibraciones en señales eléctricas que el cerebro pueda procesar.

Los ordenadores o los robots no son capaces de trabajar con ondas de sonido analógico. Utilizan unos sensores de presión de ondas sonoras llamados micrófonos. Se encargan de transformar estas ondas de sonido analógicas en señales eléctricas para poder procesarlas. Estas señales pasan a un conversor Analógico-Digital donde se obtiene la información codificada en forma de bytes.

2.2.- SÍNTESIS DE VOZ

2.2.1.- COMPOSICIÓN:

La síntesis de voz consiste en la creación de ondas de sonido artificiales semejantes al habla humana. En inglés se conoce como Text-To-Speech por la conversión de texto a voz.

Los sistemas de síntesis de voz tienen dos partes diferenciadas.

La primera parte se llama front-end. Es la parte encargada de recoger el texto y procesarlo sustituyendo abreviaturas o números en sus equivalentes para facilitar la transformación posterior. A menudo se conoce como normalización del texto o pre-procesado. También hace una distinción entre las partes de la oración en base a la prosodia de la frase (si es interrogación, exclamación, frase afirmativa, negativa, etc.). Separa palabras y les asigna la transcripción fonética.

La segunda parte de los sistemas de síntesis de voz se llama back-end. Consiste en crear las ondas de sonido mediante las configuraciones que han tenido lugar en la front-end.

2.2.2.- HISTORIA:

Mucho antes del desarrollo del procesado de señal moderno, los investigadores de la voz intentaron crear máquinas que produjesen habla humana. El Papa Silvestre II (1003), Alberto Magno (1198-1280) y Roger Bacon (1214-1294) crearon ejemplos tempranos de 'cabezas parlantes'. Autómatas capaces de crear ondas de sonido semejantes a la voz humana.

Por supuesto, estos sonidos eran guturales y casi no se podía entender lo que decían.

En 1779, el científico danés Christian Gottlieb Kratzenstein, que trabajaba en esa época en la Academia Rusa de las Ciencias, construyó modelos del tracto vocal que podían producir las cinco vocales largas (a, e, i, o, u). Wolfgang von Kempelen de Vienna, Austria, describió en su obra menschliche Sprache Mechanismus Beschreibung der sprechenden Maschine ("mecanismo del habla humana con descripción de su máquina parlante", J.B. Degen, Wien) una máquina accionada con un fuelle. Esta máquina tenía, además, modelos de la lengua y los labios, para producir consonantes, así como vocales. En 1837 Charles Wheatstone produjo una 'máquina parlante' basada en el diseño de von Kempelen, y en 1857 M. Faber construyó la máquina 'Euphonia'. El diseño de Wheatstone fue resucitado en 1923 por Paget.

En los años 30, los laboratorios Bell Labs. desarrollaron el VOCODER, un analizador y sintetizador del habla operado por teclado que era claramente inteligible. Homer Dudley refinó este dispositivo y creó VODER, que exhibió en

la Exposición Universal de Nueva York de 1939.

Los primeros sintetizadores de voz sonaban muy robóticos y eran a menudo inteligibles a duras penas. Sin embargo, la calidad del habla sintetizada ha mejorado en gran medida, y el resultado de los sistemas de síntesis contemporáneos es, en ocasiones, indistinguible del habla humana real.

A pesar del éxito de los sintetizadores puramente electrónicos, sigue investigándose en sintetizadores mecánicos para su uso en robots humanoides. Incluso el mejor sintetizador electrónico está limitado por la calidad del transductor que produce el sonido, así que en un robot un sintetizador mecánico podría ser capaz de producir un sonido más natural que un altavoz pequeño.

El primer sistema de síntesis computerizado fue creado a final de la década de 1950 y el primer sistema completo texto a voz se finalizó en 1968.

Hacia finales de los años 70, aparecieron las primeras máquinas capaces de convertir texto tecleado en voz (conversor texto-voz), que junto con los programas de reconocimiento óptico de caracteres (Optical carácter Recognition en inglés) produjeron los primeros sistemas comerciales para leer libros en voz alta. Uno de los más famosos es la Kurzweil Reading Machine, que por su precio en aquella época, sólo estaba accesible en algunas bibliotecas importantes del mundo, en particular la del MIT (Massachussets Institute of Technology). Fue precisamente en esa Universidad donde se desarrolló uno de los primeros conversores texto-voz del mundo (EL MIT-Talk).Este sistema, fue convertido en producto por la Empresa Telesensory Speech Systems (más tarde Speech Plus Inc y después adquirida por Centigram). El producto se llamaba Prose 2000 y convertía en voz todo texto enviado a su puerto serie en formato ASCII. La primera versión funcionó sólo para el idioma inglés americano. Otros sistemas le siguieron como el DEC-Talk, el Klat-talk, el Infovox, y muchos otros.

Unos años más tarde empezaron a aparecer conversores texto-voz en otros idiomas, español, francés, sueco, alemán, italiano... etc.

Posteriormente, ya a finales de los años 80, las principales operadoras telefónicas del mundo tomaron cartas en el asunto, y produjeron sus propios conversores texto a voz, en un conjunto de idiomas diverso. Cabe citar Bell Labs de ATT, más tarde escindida en Lucent Technologies y ATT Research, British Telecom., France Telecom., Deutsche Telecom., CSELT, NTT y, por descontado, Telefónica [5].

El interés de todas estas últimas centrado sobre todo en la automatización de servicios de información telefónica, en los que los datos disponibles están sobre todo almacenados en el ordenador en modo texto.

Precisamente los servicios de información y atención telefónica automática son uno de los pilares económicos importantes de todos los desarrollos actuales de la Tecnología del Habla [6].

2.2.3.- SISTEMAS DE SÍNTESIS DE VOZ:

A la hora de crear una onda de voz hay que tener en cuenta dos aspectos fundamentales que tiene que tener el resultado final.

La naturalidad y la inteligibilidad. Los sistemas de síntesis, buscan un equilibrio de ambas características.

La naturalidad es la cualidad por la que una onda sintética se parece a la voz humana (conjunto factores que afectan la pronunciación de una manera global, como la entonación, el ritmo y la intensidad del habla).

La inteligibilidad es la cualidad de poder entender la onda de voz sintética sin necesidad de esfuerzo.

Un conversor de texto a voz consta de tres bloques [7]:

- *Análisis lingüístico del texto:*

Como se ha dicho previamente, se realiza un pre-procesado para insertar correctamente los equivalentes tanto de abreviaturas como de números, etc.

También se divide la frase por palabras y se separan las partes gramaticales. Finalmente este bloque también engloba la conversión del texto en los símbolos fonéticos basándose en las reglas gramaticales del idioma.

Dependiendo de la sofisticación del software, pueden haber hasta cuatro niveles de conversión basándose en el nivel fonético (el más sencillo que atiende a la pronunciación de las palabras), en el nivel sintáctico (atendiendo a la estructura gramatical), en el nivel semántico (atendiendo al significado de la frase a procesar) o al nivel pragmático (atendiendo al significado del discurso).

Para mejorar la naturalidad del resultado final, se puede crear una prosodia concreta de cada palabra dependiendo del lugar que ocupa en la frase a procesar.

- *Estudio de la frase y su entonación o prosodia*

Este bloque central se encarga de estudiar la frase en su conjunto para poder darle una entonación correcta. También se encarga de estudiar los aspectos relacionados con la prosodia como son la entonación, la melodía (ritmo), y la intensidad del texto. Estos aspectos se han ido alcanzando y perfeccionando por medio del ensayo y error hasta conseguir una entonación y melodía adecuados.

Actualmente tienen bases de datos que alcanzan la prosodia correcta de las frases automáticamente por medio de métodos estadísticos sobre una base de datos preestablecida.

Este apartado es importante para generar una voz artificial lo más natural posible ya que una buena entonación y la melodía ayudan a la calidad final de la onda sintetizada.

La entonación es la evolución de la frecuencia fundamental (pitch) a lo largo del texto y la melodía o el ritmo incluye tanto las duraciones de cada uno de los segmentos como de las pausas entre palabras.

En el idioma español, la entonación consta de una rama ascendente que comprende desde el primer sonido hasta el primer acento tónico (rama intensiva) y a partir de aquí se mantiene subiendo y bajando la entonación hasta la parte del último acento (rama distensiva).

La elevación de la rama distensiva significa que la frase no está completa. Si esta rama es de entonación descendente, significa que la frase finaliza (rama conclusiva).

Finalmente si se produce una elevación y un descenso en la misma frase, significa que la frase es interrogativa.

- *Síntesis de voz*

El último bloque es el sistema de síntesis de voz. En este bloque existen dos tecnologías principales: la síntesis concatenativa que se basa en segmentos grabados de voz humana real y la síntesis por formantes que se basa en crear una onda de sonido partiendo de cero configurando las frecuencias y las demás características.

Una vez encontradas las unidades mínimas tanto en el de síntesis concatenativa como en el de síntesis por formantes, se aplica un post-procesado segmental.

Este post-proceso tiene la función de asegurar la continuidad de la entonación, la intensidad y la melodía al pasar de una unidad mínima a otra, manteniendo el tono constante sin saltos ni defectos fonéticos.

2.2.4.- TIPOS DE SÍNTESIS DE VOZ:

2.2.4.1.- CONCATENATIVA

Existen tres tipos de síntesis concatenativas:

- Síntesis por selección de unidades:

En este tipo de síntesis concatenativa trae consigo una gran base de datos donde están almacenadas las grabaciones del habla.

La base de datos se compone de unidades que pueden ser fonemas, sílabas, palabras, frases u oraciones. Con estos datos, se modela la onda de voz atendiendo a los datos analizados anteriormente.

El problema de esta tecnología era que para cada prosodia determinada de la frase se tendría que elaborar y parametrizar dicha frase para unir los distintos segmentos. La forma de unir los segmentos se revolucionó a partir de la aparición del procesamiento segmental llamado PSOLA (Pitch-Synchronous Over-Lap and Add).

Permite modificar la frecuencia fundamental de una señal de un modo sencillo actuando directamente sobre la representación de la misma en el dominio del tiempo, pudiendo utilizarse por tanto para formar la curva prosódica de los mensajes a emitir.

Consiste en la separación de las distintas unidades, duplicando su periodo al doble y superponiendo las unidades para que no tengan saltos cuantitativos a la hora de la unión. Se utiliza para aumentar la naturalidad de la voz sintética.

La síntesis por selección de unidades de propósito general (sin un tema específico) necesita una base de datos extensa para tener una buena calidad y naturalidad.

Es posible crear una voz natural e inteligible con una base de datos pequeña si es con un propósito concreto sobre un determinado tema, como por ejemplo, el sistema de aviso de una terminal de autobuses que está determinado a decir lo mismo una y otra vez sin salirse del guión establecido.

- Síntesis por difonos:

Utiliza una base de datos más compacta.

Contiene todos los difonos del lenguaje. A partir de estos difonos y atendiendo a los parámetros obtenidos en los procesos anteriores, se realiza una búsqueda estadística, seleccionando el difono que más se ajuste.

Como los difonos ya contienen una característica prosódica del lenguaje, no es necesario realizar el proceso segmental de la unión por unidades pero si conviene hacer un post-procesado para suavizar la unión de los mismos.

El resultado final es una voz robótica poco natural y parcialmente inteligible. Por este motivo no se usa en aplicaciones comerciales pero sí en investigación ya que tiene programas de libre distribución.

- Síntesis específica para un dominio:

En esta síntesis, se tiene una base de datos con palabras y oraciones grabadas directamente de la voz humana. Se utiliza con fines sobre temas muy concretos, como por ejemplo un sistema de sonido que de las horas.

En este sistema se nota mucho el cambio de entonación e intensidad entre palabras. En cambio es uno de los sistemas más naturales que existen puesto que la prosodia y entonación de las palabras como unidades son las de las grabaciones originales. No es válido para oraciones de propósito general (para cualquier tema) debido a la mínima base de datos.

2.2.4.2.- POR FORMANTES

Esta tecnología crea zonas de concentración de energía en el espectro del sonido sintetizado, lo que imita el sonido de la voz.

La síntesis de formantes no usa muestras de habla humana en tiempo de ejecución. En lugar de eso, la salida se crea usando un modelo acústico. Parámetros como la frecuencia fundamental y los niveles de ruido se varían durante el tiempo para crear una forma de onda o habla artificial.

Se basa en la utilización de filtros para crear los distintos efectos sobre las ondas que provocarían los aparatos fonadores humanos.

Estos filtros modifican los siguientes parámetros acústicos como muestra la siguiente tabla Tab. 2.2.4.2:

Tab. 2.2.4.2 Parámetros de la síntesis de voz

Parámetro	Base fisiológica	Acústica
AV amplitud de la sonoridad	Amplitud de la vibración de las cuerdas vocales	Intensidad de la onda sonora
Fofrecuencia fundamental	Frecuencia de vibración de las cuerdas vocales	Frecuencia fundamental
F1 primer formante	Primera resonancia del tracto vocal	Frecuencia del primer formante
F2 segundo formante, F3 tercer formante...	Segunda resonancia del tracto vocal, tercera resonancia del tracto vocal...	Frecuencia del segundo formante, frecuencia del tercer formante...
B1 amplitud de banda del primer formante, B2, B3...	Configuración del tracto vocal	Ampitud de banda del primer formante...
AK Amplitud del ruido de fricción	Tipo de constricción	Intensidad de los componentes aperiódicos
K1, K2 Frecuencia del ruido de fricción	Tipo de constricción	Frecuencia de los componentes aperiódicos
AH Amplitud de la aspiración	Fricción en el tracto vocal	Componentes aperiódicos de los formantes
N1 Frecuencia del formante nasal	Resonancia del tracto nasal	Frecuencia del formante nasal
AN Amplitud de la nasalidad	Resonancia del tracto nasal	Intensidad de la resonancia nasal

2.3.- RECONOCIMIENTO DEL HABLA

En este apartado del trabajo se hace necesario hacer una distinción clara sobre reconocimiento de voz y reconocimiento del habla.

Ya se ha definido previamente que no hay dos personas que tengan los mismos componentes armónicos, entonación, timbre o intensidad en la voz. En esto se basa precisamente el reconocimiento de voz. Compara y determina estos parámetros buscando en la base de datos las ondas grabadas previamente de esa persona. Se usa en sistemas de seguridad.

El reconocimiento del habla, es un sistema capaz de transcribir un mensaje oral en texto o emitiendo órdenes acordes, independientemente del hablante. Se basa en la comparación con un modelo acústico recogido en una base de datos.

Su implantación comercial nos ha llegado en forma de compañías telefónicas, acceso a discapacitados visuales, etc.

2.3.1.- HISTORIA

En la época de 1870, Alexander Graham Bell inventó el teléfono gracias a querer implementar un sistema capaz de ayudar a las personas con discapacidades físicas auditivas.

AT&T Bell Laboratories, en los años 1910 inventan un sistema electrónico capaz de reconocer los 10 primeros números en inglés con un índice de acierto del 99%. Este sistema necesitaba de una gran configuración interna para cada locutor.

A mediados de los años 60 los sistemas empiezan a incorporar técnicas de normalización del tiempo (que dos palabras iguales dichas más o menos rápidas pudieran reconocerse igual), se centran en locutores individuales y con un vocabulario muy reducido (alrededor de 50 palabras).

Estos sistemas tenían en común el segmentado del habla, es decir, las palabras se pronunciaban con cierto espacio entre sí para que el sistema pudiera reconocerlas.

A principios 1970 se produce el primer producto comercial de reconocimiento de voz, el **VIP100** de Threshold Technology Inc. que utiliza un vocabulario pequeño, dependiente del locutor, y reconocía palabras discretas [8].

2.3.2.- TIPOS DE RECONOCIMIENTO DEL HABLA

Existen dos tipos de reconocimiento automático del habla: *DirectVoice Input* (DVI), que se basa en el reconocimiento de unos cientos de palabras para trasladar esa información a otra aplicación, y el otro es el *Large vocabulary continuous speech recognition* (LVCSR) que se usa en la creación de documentos pudiendo reconocer frases extensas [9].

Ambos sistemas se basan en la comparación de fonemas.

En la base de datos se almacenan los fonemas que componen el lenguaje y lo comparan con la palabra o frase que el locutor emite.

Esta comparación es compleja ya que utilizan unas técnicas basadas en modelos ocultos de Markov (Hidden Markov Models – HMM [10]).

Su característica principal es que está compuesto de estados y a estos estados se llega o se cambia dependiendo de la probabilidad de unos parámetros de entrada.

La calidad de un sistema de reconocimiento del habla se mide en el porcentaje de error y velocidad.

El error se da en tanto por ciento de la frase a procesar. Así si se está procesando una frase con 10 palabras, tendremos un error en función de las palabras reconocidas correctamente, de las que están cambiadas de sitio, de las que no están y de las que aparecen pero no se han pronunciado.

Este error es comúnmente conocido como Word Error Rate (%) (WER). La velocidad es un factor a tener en cuenta en los sistemas de tiempo real.

En estos sistemas cumplir los plazos es esencial por ello el software asociado al reconocimiento de voz tiene que ser lo más rápido posible.

Para poder procesar la onda de sonido, se divide períodos de 20 ms y se analizan los parámetros en el dominio del tiempo tales como frecuencia, amplitud, silencios y armónicos y otros parámetros en dominio discreto (transformadas de Fourier).

Se hace un primer barrido de esos 20 ms y se detectan los silencios para poder separar las palabras. En la base de datos se almacenan patrones de comparación de los distintos fonemas del lenguaje.

Con los datos obtenidos y por medio de un sistema estadístico complejo se obtienen unos resultados aproximados.

Algunos programas muy sofisticados llegan a hacer varios barridos de este tipo para aumentar el porcentaje de aciertos.

Estos modelos estadísticos son muy diversos pero los dos predominantes son las cadenas ocultas de Markov (HMM) y las redes neuronales.

El primero se basa en la comparación de una secuencia de patrones generada por medio de un sistema de estados. Esta secuencia de estados tiene la característica que no se puede observar o determinar directamente.

Los estados pueden ser Left-to-right (de izquierda a derecha) o pueden ser ergódicos (se puede llegar desde cualquier estado). En la Fig. 2.3.2 [31] se detallan los tipos de estados de los modelos ocultos de Markov.

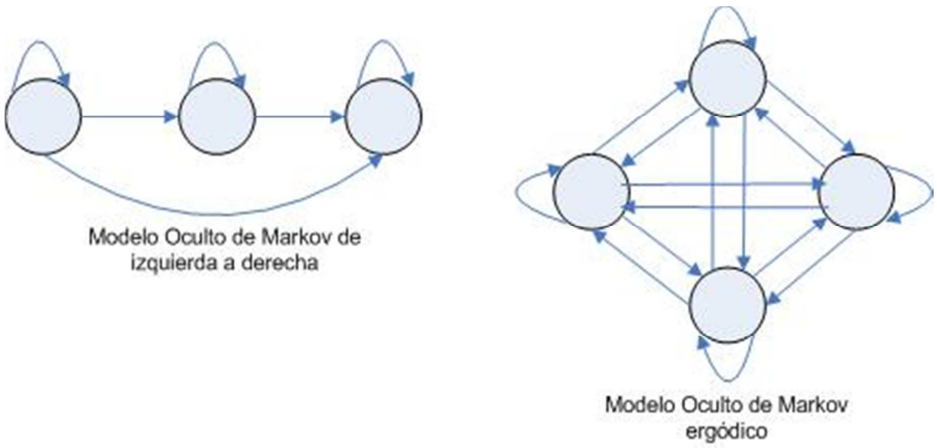


Fig. 2.3.2 Modelos Ocultos de Markov

En ambos casos, se modifica el estado por medio de las probabilidades estadísticas de unos parámetros en las ondas procesadas.

Las redes neuronales se basan en unos objetos de programación que reciben entradas (parámetros de las transformadas de Fourier) y dependiendo del peso y del algoritmo interno de estos objetos darán una señal de salida para poder hacer conexiones con otros objetos formando una red.

Estos objetos tienen un umbral de comparación para generar una salida acorde a los patrones a comparar.

El objetivo es el mismo que el del sistema de los modelos ocultos de Markov: comparar los patrones de la onda de sonido con otros patrones almacenados para decidir si es o no la palabra o frase correcta que se quiere procesar.

Cada estado lleva asociado unos determinados argumentos que en conjunto dan como resultado el texto de la frase procesada.

3 ESTUDIOS REALIZADOS

3.1.- SÍNTESIS DE VOZ

En el mercado existen diversos programas encargados de sintetizar voz a partir de un texto ya sea mediante una interface del mismo programa o por medio de la consola de Linux. En este trabajo se ha intentado hacer un análisis superficial de cada programa viendo su grado de calidad y posibilidad de empotrar el software para su uso en otros sistemas o programas.

3.1.1- SOFTWARE DE LIBRE DISTRIBUCIÓN

3.1.1.1.- The Festival Speech Synthesis System

Festival [1] es un sintetizador de texto a voz pensado para tres tipos de personas: los usuarios que quieren simplemente traducir un texto arbitrario a voz ya sea por ayudar a los discapacitados o simplemente por ocio, los desarrolladores que usan este software como medio para conseguir un objetivo en su programa (este es el caso del presente proyecto), o los investigadores que profundizan en el campo de la síntesis para mejorar o avanzar en sus proyectos.

Está escrito enteramente en C++ y usa Edinburgh Speech Tools Library junto con una interfaz de comandos escrita en el terminal.

Tiene una licencia de libre distribución X-11 para uso cotidiano no comercial sin restricción y comercial con licencia de pago para dar soporte.

Se basa en la tecnología concatenativa por difonos.

Su síntesis final es una voz robótica perfectamente entendible y de buena calidad.

Sus aplicaciones suelen ser de investigación más que para el ámbito comercial. Está disponible para español, inglés británico y americano.

Este software incluye documentación completa de las funciones a utilizar (API) facilitando la programación. Además incluye librerías para poder utilizar en programación C, C++ y JAVA.

3.1.1.2.- Microsoft Speech Application Programming Interface (SAPI)

SAPI [11] soporta los motores TTS3000 de Lernout y Hauspie para español, inglés americano, inglés británico, holandés, francés, alemán, italiano, japonés, coreano, ruso y portugués de Brasil.

Está escrito en C y C++ en las versiones anteriores a 5 y en JAVA en las posteriores.

Es de licencia particular y sólo se puede usar en Windows.

Su uso básico es incorporar voz a distintas aplicaciones de Microsoft. Desde la primera versión SAPI 1 para Windows 95 y NT 3.51 ha ido evolucionando incorporando nuevas voces con más claridad y más funciones. En la última versión de SAPI 5.4 este software es capaz tanto de sintetizar voz realmente parecida a la voz humana como reconocerla.

Tiene soporte para Windows 7 y todos los anteriores.

3.1.1.3.- Mbrola

Mbrola [12] es un sistema de concatenación de difonos para unas 25 lenguas.

Aunque no use motor propio para la síntesis del habla, no acepta texto para convertir a voz, tiene una gran base de datos que usan otros programas. Utiliza un conjunto de fonemas que junto con las formas prosódicas correspondientes a cada lenguaje, obtiene una señal acústica de 16 bits a la salida emulando una voz robótica sintética de alta calidad.

Su principal idea es facilitar el estudio de la síntesis de voz para la investigación académica, obteniendo los patrones de la mayor cantidad posible de voces sintéticas para el máximo posible de idiomas.

La distribución es libre para uso no comercial y no militar.

3.1.1.4.- Gnuspeech

Gnuspeech [13] Es un paquete extensible de texto a voz basado en síntesis por reglas articulatorias en tiempo real. Se basa en Tube Resonance Model (TRM) es decir en simular un tracto fónico humano a partir de filtros de sonido.

Utiliza diccionarios léxicos y modelos de entonación y ritmo para transformar los descriptores fonéticos en parámetros acústicos sintéticos de bajo nivel.

Finalmente crean la onda de sonido agrupando estos descriptores fonéticos en tiempo real.

TMR representa con realismo las propiedades del tracto fónico humano incluyendo la intensidad del sonido, la simulación de la cavidad nasal como oral así como también la distorsión producida por labios, nariz y lengua.

La base de datos (modelos acústicos, diccionarios léxicos y formantes) está basada en la investigación de KTH Royal Institute of Tecnology en Estocolmo y la Universidad de Calgary (Canada).

Únicamente tiene diccionario para el idioma inglés y algunas palabras en francés.

3.1.2.- SOFTWARE CON LICENCIA COMERCIAL

3.1.2.1.- Loquendo TTS

Este software tiene diccionarios para español (varios acentos), catalán, holandés, inglés, portugués, italiano, francés, alemán, griego, sueco y chino. Hay disponibles demostraciones interactivas vía web.

Loquendo TTS [14] dispone de voces verdaderamente naturales que pueden leer cualquier texto y comando de las aplicaciones vocales basadas en servidor, multimedia, embebidas y multimodales.

Además de incorporar una voz clara y agradable, tiene base de datos para masculino y femenino e incluso para distintos acentos de un mismo idioma (Español Castellano-Argentino-Mejicano, Inglés Británico-Americano, etc.).

Tiene soporte para Windows y Linux. Su motor de transcripción es muy potente y para ello los desarrolladores han impuesto unos requerimientos del sistema:

-Loquendo TTS Multimedia: 10 MB RAM para el motor, 50 MB por voz, 3 MB por canal.

-Loquendo TTS Multimedia Compacto: 10 MB RAM para el motor, 20 MB por voz.

Las voces de Loquendo son tan logradas que la mayoría de los contestadores automáticos de las grandes compañías telefónicas utilizan este software e incluso han llegado a sustituir operadores telefónicos humanos.

3.1.2.2.- SodelsCot

Este Software ha sido diseñado por la empresa Sodels Factory S.L.

El texto puede proceder de editores, navegadores, libros digitales, correos electrónicos, etc. Entre otras opciones, SodelsCot [15] puede escuchar páginas web, correos electrónicos, etc., escuchar el texto tecleado en cualquier momento, grabar documentos o libros electrónicos en mp3, estudiar textos o perfeccionar la pronunciación, publicar contenidos con voces profesionales, utilizar la síntesis de voz en sus aplicaciones.

Utilizan la voz creada por Loquendo (Jorge) para la versión empresarial y la voz creada por RealSpeak (Mónica) para la versión profesional. También este software es compatible con voces TTS de Cepstral, AT&T o NeoSpeech.

La interfaz gráfica es realmente fácil de usar y tiene opciones tan simples como es copiar el texto y con el botón derecho seleccionar la opción de reproducir.

Su licencia es comercial. Tienen a disposición del usuario una versión completa de prueba de 7 días.

3.1.2.3.- Cepstral tts voices

Cepstral es una empresa dedicada únicamente a la síntesis de voz. Han sacado varios programas entre ellos Cepstral Voices [16] y Cepstral LLC.

Ambos programas dedicados a la síntesis de voz tanto masculinas y femeninas en varios idiomas, entre ellos inglés, español y francés.

Las aplicaciones de este software son muy variadas, desde telefonía, móviles, asistencia a discapacitados, lectura de archivos, navegación web, juegos digitales, aplicaciones en organismos públicos, industria y servicios de asistencia médica (tanto en aparatos médicos como información al ciudadano).

En estas diversas aplicaciones este software cumple con los estándares de programación tanto en empotramiento como en portabilidad. Es multiplataforma, natural y sencillo.

Es compatible con las voces de Microsoft SAPI a partir de la versión SAPI 5. Tiene una interfaz visual muy intuitiva y la configuración puede hacerse desde el panel de control de Windows o Linux.

Una versión completa incluye un SDK (set development kit) con las distintas funciones, librerías, APIs y ejemplos para ayudar al empotramiento de

este software en nuestros programas y sistemas ya sean escritos en C o C++. Incluso para los proyectos escritos en JAVA es posible usar librerías específicas para TTS pero no dan soporte oficial para este lenguaje.

Tiene soporte técnico bajo licencia comercial explícita aunque puedes adquirir únicamente el software sin soporte.

Es necesario unos requisitos mínimos para el programa Cepstral TTS voices: CPU: Intel Pentium II, AMD K6, 64 Mb RAM, 25-110 Mb por voz.

Estos requisitos son únicamente para el programa con interfaz visual.

3.1.2.4.- AT&T Natural Voices

AT&T Natural Voices [17] se centra en dos ramas: El desarrollo de aplicaciones (Desktop Edition) y el desarrollo de aplicaciones telefónicas (Server-Lite/ Server Edition). Ambos ámbitos usan un motor de síntesis con soporte en inglés británico y americano con voces en masculino y femenino, español latino americano con una voz femenina, alemán tanto femenina como masculina y francés con ambas voces masculinas y femeninas.

Está escrito en Java y da soporte tanto a Java como a C++. Tiene unas librerías a disposición del usuario y un repertorio de funciones recogidas en los APIs de ambos tipos de lenguaje de programación lo que facilita el empotramiento en otros sistemas o programas.

El motor de búsqueda y síntesis es el mismo o similar que usa Microsoft en su versión SAPI 4.

Para la versión Desktop están disponibles unas voces de mayor claridad (16KHz) mientras que para aplicaciones telefónicas son de (8KHz).

Se distribuye bajo licencia comercial y dan soporte técnico tanto a usuarios como a desarrolladores.

Este software requiere unos mínimos de hardware para el correcto funcionamiento del programa: Requisitos mínimos: RAM: 128 Mb (mínimo), CPU: 300 MHz III, Disco duro: 500 MB.

3.1.2.5.- Acapela group

Acapela group [18] se dedica únicamente a trabajar con sistemas tanto de síntesis de voz como de reconocimiento de voz. Tiene diversos programas para las distintas plataformas (Linux, Windows, Pocket-PC, etc.).

Su software es de fácil empotramiento y está dedicado al ensamblaje en aplicaciones y proyectos gracias a su gran variedad de información en forma de APIs para distintos tipos de lenguajes.

Las voces varían en calidad, frecuencias de 8KHz, 11KHz, 16KHz y 22KHz en una gran variedad de idiomas: árabe, catalán, checo, inglés americano y británico, finlandés, francés, alemán, italiano, español castellano y latino, sueco y turco. En varios de los idiomas existen voces tanto femeninas como masculinas y mayor o menor calidad.

Entre sus aplicaciones existe una que lee directamente el contenido que haya almacenado en el portapapeles de Windows o Linux. Otras aplicaciones tienen una interfaz gráfica y sencilla. Su punto fuerte son los SDK (Set development Kit) para los programadores en una multitud de plataformas tanto si son servers o mobile-PC, incluso tienen una línea de comercialización de un hardware específico de TTS para SCADA. Es totalmente compatible con Microsoft SAPI 5 y pueden usarse sus voces.

Su licencia es comercial con soporte técnico parcial. Existe una versión que incorpora servicio técnico completo para desarrolladores y programadores. Los usuarios tienen una versión de prueba de 15 días de un programa sencillo para poder escuchar las voces.

3.1.2.6.- TextAloud

La empresa NextUp Technologies tiene una línea comercial dedicada a la síntesis del habla.

El software distribuido es TextAloud [19], uno de los más conocidos.

Es capaz de leer cualquier texto que provenga de páginas web con solo seleccionarlo. Una función que proporciona este software es la posibilidad de generar una onda de sonido en formato mp3.

Puede usar (y descargar) voces de AT&T Natural Voices, NeoSpeech, Cepstral, Acapela, todas ellas compatibles con SAPI 4 y SAPI 5.

Por tanto es ideal para una gran variedad de países o sistemas que necesiten ser multilingües. Las voces que se pueden seleccionar son el inglés, español, alemán y francés entre muchas otras. Este software no es empotrable en ningún otro sistema.

La licencia comercial es directamente su programa sin posibilidad de utilizar sus funciones o librerías para otros sistemas. Tiene soporte técnico para TextAloud.

3.1.2.7.- VerbioTTS

Verbio [20] Technologies S.L., con sede en Barcelona (España), es una empresa especializada en el desarrollo de tecnologías del habla, básicamente síntesis de voz y reconocimiento del habla.

Tiene aplicaciones en el campo de la telefonía: Call Center, mensajería unificada, operadoras automáticas, en el campo de domótica: confirmación de la información o lectura de datos del sistema, en el campo de la industria: ayuda a la automatización industrial de procesos mediante la voz, y también en el campo de la ayuda y soporte a los discapacitados a distintos niveles.

Es capaz de soportar arquitecturas de cliente- servidor y monopuesto (desktop). Está habilitado para poder trabajar tanto en Linux como en Windows.

Los idiomas son muy variados centrándose en el castellano (con varios acentos) pero también da soporte al idioma inglés con pronunciación americana y británica, francés, catalán, gallego, euskera, latino mejicano, colombiano y latino con tonos neutros, portugués y brasileño.

En algunos idiomas es posible elegir la voz masculina o femenina. Para mayor calidad cuenta con unas voces en 16KHz y otras de menos calidad para empotramientos compactos de 8KHz.

Este software es totalmente empotrable en nuestros proyectos. La interfaz es SAPI 4 y SAPI 5 pero gracias a su conjunto de librerías y funciones recogidas en su API, con el SDK se puede programar fuera de las SAPI de microsoft. Su SDK (Set Development Kit) para desarrolladores es específico del entorno de desarrollo o la plataforma ya sea Dialogic, Eicon, CETADE, AvayaIR o genérico. Está escrito en C/C++ pero con posibilidad de adaptarlo a lenguajes tipo Visual Basic, Delphi o Java.

3.1.2.8.- FonixTalk 6.1

Este software distribuido bajo licencia comercial, está creado por la compañía SpeechFX Inc.[21]

Su principal línea de desarrollo es con sistemas de voz, tanto reconocimiento como síntesis. Tiene aplicaciones en el mercado de video-juegos.

Es un sistema multilenguaje en inglés americano y británico, francés, alemán, italiano y español latino y castellano. También está disponible en koreano y chino mandarín. Tiene opción a poner la voz en masculino o femenino para algunos idiomas (no todos) y una voz de niño en inglés. Hay dos calidades de voz, la de 16 Khz y la de 8 Khz.

Es completamente empotrable en aplicaciones o proyectos gracias a los SDK y las APIs. Su interfaz es la de Microsoft SAPI 4 y SAPI 5, aunque con los APIs puedes utilizar entornos de desarrollo y emplear las librerías de FonixTalk 6.1.

3.2.- RECONOCIMIENTO DEL HABLA

Para ayudar en el reconocimiento del habla, existen diversos programas ya sean con licencia gratuita o comercial en el mercado capaces de escribir en texto lo que un locutor diga por el micrófono.

3.2.1.- SOFTWARE DE LIBRE DISTRIBUCIÓN.

3.2.1.1.- CMU Sphinx

Este software es de libre distribución creado por la universidad Carnegie Mellon en Pittsburgh, Pennsylvania (USA).

El motivo de que sea libre es incentivar el estudio y la investigación de sistemas basados en la voz, más concretamente en el reconocimiento de voz.

Las primeras versiones de Sphinx [22] (Sphinx-1, Sphinx-2 y Sphinx-3) se han quedado obsoletas, siendo sustituidas por Sphinx-4 escrita enteramente en JAVA que es el máximo exponente de este software.

También crearon una versión de código más compacta (pocketsphinx) para poder adaptar mejor el software al proyecto escrita en C. Gracias a las APIs la programación es bastante sencilla salvo por la falta de ejemplos y manuales parcialmente incompletos.

Está adaptado al reconocimiento del habla en inglés americano y británico. Al ser de libre distribución existen bases de datos (FestVox) con los diccionarios necesarios para poder cambiar el idioma a español, francés, árabe, portugués, etc.

Estos diccionarios con sus correspondientes modelos léxicos y acústicos se crean a partir de donaciones de voz de gente anónima de todo el mundo.

Pocketsphinx es el sistema de reconocimiento del habla más rápido del proyecto CMU Sphinx, haciéndole el perfecto candidato para los desarrolladores. Este software ha ido incorporándose poco a poco al mundo de la telefonía móvil con aplicaciones de dictado, y escritura automática de mensajes gracias a la tecnología “SmartPhone”.

Ambas versiones basan sus sistema de motor de búsqueda en los estados ocultos de Markov (HMM) que es precisamente la base de datos que se necesita para hacer funcionar el programa.

Esta base de datos llamada diccionario, junto con el modelo acústico y el modelo léxico más la posibilidad de incorporar una prosodia específica y de acentos, se carga en el motor de búsqueda y comienza el reconocimiento.

Si se quiere cambiar el idioma, basta con cargar otros diccionarios y modelos en el motor de búsqueda.

3.2.1.2.- API Android

Android [23] es un Sistema Operativo para los teléfonos móviles llamados “Smartphone”. Este sistema operativo diseñado por la compañía Google está habilitado para que cualquier persona pueda programar en JAVA programas y aplicaciones de cualquier índole, desde aplicaciones matemáticas como calculadoras, hasta localizadores GPS.

Este sistema operativo tiene una gran variedad de documentación en su página web donde explica paso a paso a utilizar todas las librerías disponibles. Una de estas librerías está dedicada a la voz, ya sea TTS o reconocimiento del habla. Una aplicación muy famosa que ya está en funcionamiento en los dispositivos Smartphone es la redacción de mensajes con la voz.

Estas librerías y funciones se encuentran en unas APIs de Google (desde la versión 2.1 de Android llamadas “Google APIs”) para el reconocimiento de voz.

Existe un emulador del Sistema Operativo de Android que puede ejecutarse tanto en Windows como en Linux. La propia página de Android brinda a los desarrolladores un manual completo y un “getting started” con el entorno de desarrollo ECLIPSE.

Este emulador está muy limitado y no puede ejecutar todas las aplicaciones igual que un teléfono móvil, el micrófono graba información pero no es válida debido a que no se puede codificar correctamente. Si la codificación adecuada, los datos no sirven y no se pueden ejecutar aplicaciones de reconocimiento de voz.

3.2.2.- SOFTWARE CON LICENCIA COMERCIAL

3.2.2.1.- FonixVoiceIn

Al igual que su versión de TSS, la compañía SpeechFX Inc. [24]. Se dedica también al reconocimiento del habla. Han creado un programa llamado VoiceIn en su versión 4.1 que es capaz de reconocer automáticamente el dictado del locutor y transcribirlo a texto para poder procesarlo posteriormente.

Los idiomas pueden elegirse entre los siguientes: español castellano y latino, inglés británico y americano, francés, alemán, japonés, coreano, sueco e italiano. Las voces en algunos idiomas está disponible en versión masculino o femenino y con distintos diccionarios léxicos y acústicos de mayor complejidad ASR (ideal para locuciones sin tema específico y de larga duración), hasta diccionarios con unas pocas palabras sueltas y de manera controlada descritas previamente.

Tiene soporte en distintas plataformas como Windows o Linux. Está escrito en C y C++. Las librerías de funciones se pueden encontrar en su API

para los SDK (Set Development Kit) donde hace una explicación corta de cada función, las variables que necesita y el objetivo de cada una.

Esta información está relativamente completa salvo por unas variables y funciones específicas que usa del Sistema Operativo (en el caso del presente proyecto de las funciones de ALSA en Linux) para poder calibrar el ruido registrado por el micrófono a la vez que se toman los datos del locutor.

Están escritos unos ejemplos del código para facilitar la comprensión y el potencial de este software.

El programa se distribuye bajo licencia comercial y se puede adquirir también soporte técnico para empresas. Uno de los mayores logros de esta compañía ha sido la participación en videojuegos interactivos para Xbox 360.

3.2.2.2.- Loquendo ASR speech recognition

La empresa Loquendo también tiene una rama de reconocimiento del habla. El producto para este fin se llama Loquendo ASR [25] (Automatic Speech Recognition). Es uno de los más potentes y rápidos del mercado.

Gracias a su gran resistencia al ruido y a la utilización de gramáticas y modelos estadísticos del lenguaje, son muchas las empresas que utilizan los servicios de Loquendo ya que es una de las pioneras del sector. ASR es apto para cualquier ámbito aplicativo, tanto telefónico como web, desktop, programas embebidos, automoción o “SmartPhone” y siempre dotado de los modelos acústicos para cualquier condición requerida.

Es independiente del hablante, por lo que no requiere formación previa para poder utilizarlo.

Disponible en numerosos idiomas (inglés británico, americano y australiano, español latino y castellano, franceś y francés canadiense, italiano, alemán, holandés, gallego, catalán, valenciano, sueco, danés, finlandés, árabe), el reconocedor vocal de Loquendo está a la base de servicios interactivos que administran millones de llamadas al día, como call centers automatizados, portales de voz y aplicaciones móviles.

Loquendo ASR soporta opcionalmente la funcionalidad de verificación del hablante.

3.2.2.3.- DragonNaturallySpeaking

La empresa Nuance ha saco la versión 11.5 de reconocimiento del habla llamado Dragon Naturally Speaking [26]. Con este software cualquier persona es capaz de comunicarse con el PC sin necesidad de teclado, simplemente con hablar. Este programa está pensado únicamente para hacer dictados en el PC con una hoja de texto, redactar mensajes de correo electrónico, navegar en internet o configurar el ordenador para que responda a determinadas palabras abriendo alguna aplicación o ejecutando algún programa.

Es muy sencillo y fácil de usar pero requiere configurar cada palabra a una acción. Es un software muy robusto, sin faltas de ortografía y lo puede usar cualquier persona ya que no distingue las voces. Está especialmente pensado para personas con leves discapacidades y ayudarles a integrarse en la sociedad con las nuevas tecnologías.

Es de licencia comercial y sólo se puede comprar el programa ya hecho. No hay posibilidad de comprar las funciones y librerías para empotrarlo en otros proyectos, no tiene a disposición APIs para desarrolladores.

Existen tres versiones: Dragon NaturallySpeaking 11.5 HOME, PREMIUM y PROFESSIONAL. La versión HOME no funciona en Excel o PowerPoint y tampoco es muy efectiva frente a vocabularios extensos. La versión PREMIUM sí es capaz de trabajar con dictados extensos aumentando mucho el porcentaje de acierto. Es más recomendable para estudiantes y para las empresas. La versión PROFESSIONAL incluye aplicaciones de auto transcripción, guardado automático del fichero de audio de la transcripción, control completo de aplicaciones por medio de la voz y un gran diccionario con vocabularios técnicos de trabajos específicos (lenguaje médico, ingenieril, etc)

Se puede adquirir en distintos idiomas: español, inglés, japonés, francés, italiano, alemán.

4 IMPLEMENTACIÓN ROBOT HUMANOIDE HOAP-3

En esta parte del proyecto, se pretende poner en práctica todos los conocimientos adquiridos a lo largo del presente trabajo. La plataforma elegida es el robot humanoide HOAP-3 cuyas siglas corresponden al acrónimo inglés Humanoid for Open Architecture Platform. Un robot diseñado por Fujitsu Automation en colaboración Fujitsu Lab. en su tercera versión.

4.1.- ROBOT HOAP-3

En la imagen 4.1.1 [32] se puede ver al robot HOAP-3

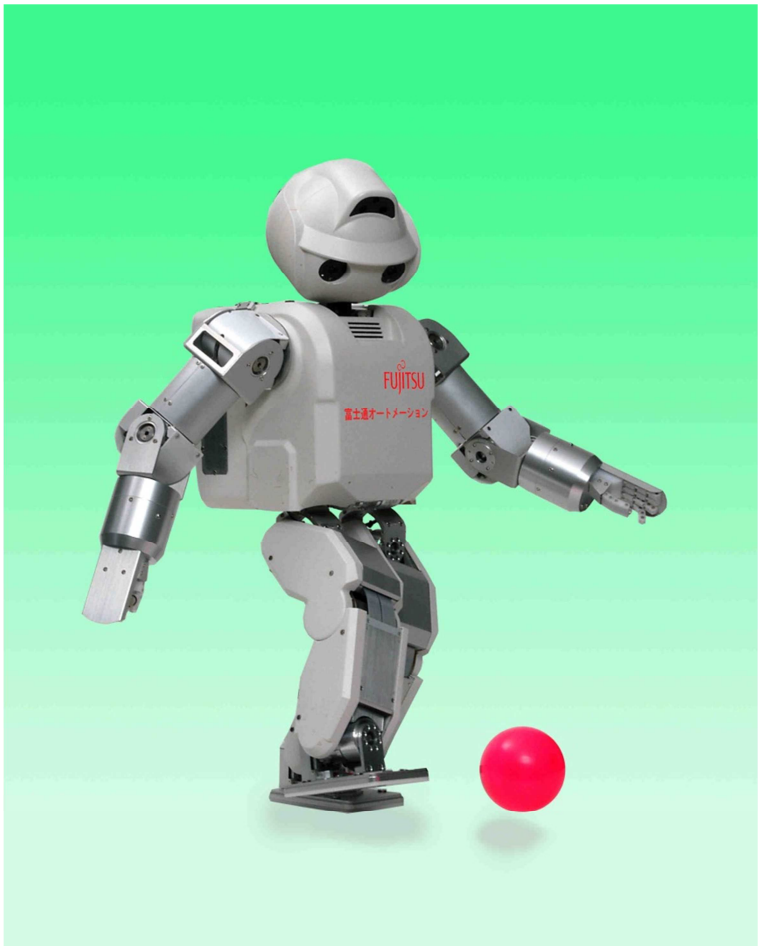


Fig. 4.1.1 Robot humanoide Hoap-3

Este robot tiene una altura de 60cm y un peso aproximado de 8'8 Kg.

Tiene 28 GDL para poder realizar los movimientos básicos como puede apreciarse en la imagen Fig. 4.1.2. [33]

Estas articulaciones con todos sus grados de libertad, permite el robot HOAP-3 realizar diversos movimientos. La longitud y la configuración de cada eslabón o articulación ha sido pensada a conciencia para asemejarse lo máximo posible a un ser humano y poder realizar movimientos típicos de los seres humanos como caminar, girar, mover brazos y piernas, girar la cabeza, etc.

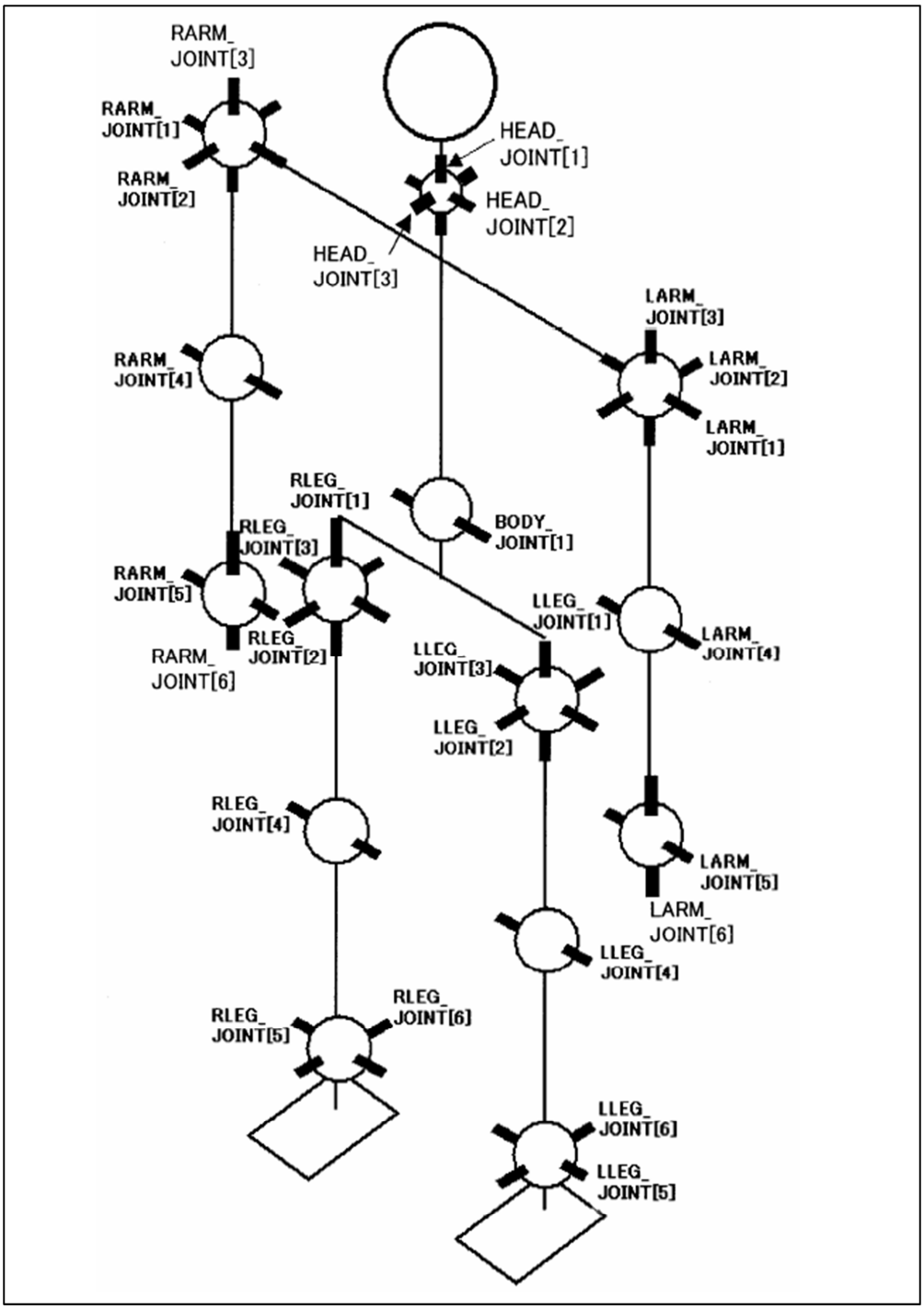


Fig. 4.1.2 Grados de libertad

Los dispositivos de control de todos los componentes (motores, baterías, leds, altavoz, micrófono, etc.) pasan por un PC embebido con un procesador Pentium M de 1'1GHz. El robot tiene instalado el Sistema Operativo Linux Fedora Core 1 (Kernel 2.4).

Se llama Robot porque este dispositivo es capaz de interrelacionarse con su entorno, es autónomo, y toma sus propias decisiones configuradas por hardware sin un operario que le diga qué hacer en cada momento. Para relacionarse con el entorno, tiene incorporados unos sensores que recogen información del exterior.

El altavoz está situado en la parte superior del pecho. Con él Se puede sintetizar la voz o incluso sonidos de cualquier tipo, música, etc. ya que el altavoz no tiene limitaciones de ancho de banda y es igual que los altavoces de casa o del trabajo.

El micrófono está situado en la parte superior de la cabeza. Es un micrófono de entorno, lo que quiere decir que no hace falta situarse al lado para que sea sensible a las perturbaciones sonoras. Este tipo de micrófonos son muy sensibles al ruido de fondo ya que deben percibir desde distancias medias.

En ambos dedos de las manos, tiene unos sensores de presión. Estos sensores recogen información de si el robot tiene sujeto algo en las manos o no detecta nada. En las plantas de los pies, tiene unos sensores de presión para determinar el grado de fuerza que está ejerciendo el robot en el suelo, básicamente, el peso de cada pie sobre el suelo. Estos sensores recogen información cuantitativa para facilitar la programación de caminar, girar o retroceder.

Tiene un sistema de dos cámaras situadas en la cabeza simulando los ojos. Esta configuración de ambas cámaras determina la visión estereoscópica para realizar programación más compleja de movimiento y profundidad de objetos.

También incorpora un sensor infrarrojo de distancia al lado del micrófono para evitar colisiones con paredes, etc.

Un giróscopos y acelerómetros para los 3 ejes situados en la cintura del robot completan el sistema de interacción con el mundo real del HOAP-3.

En la Fig. 4.1.3 se aprecian los distintos tipos de componentes arriba mencionados.

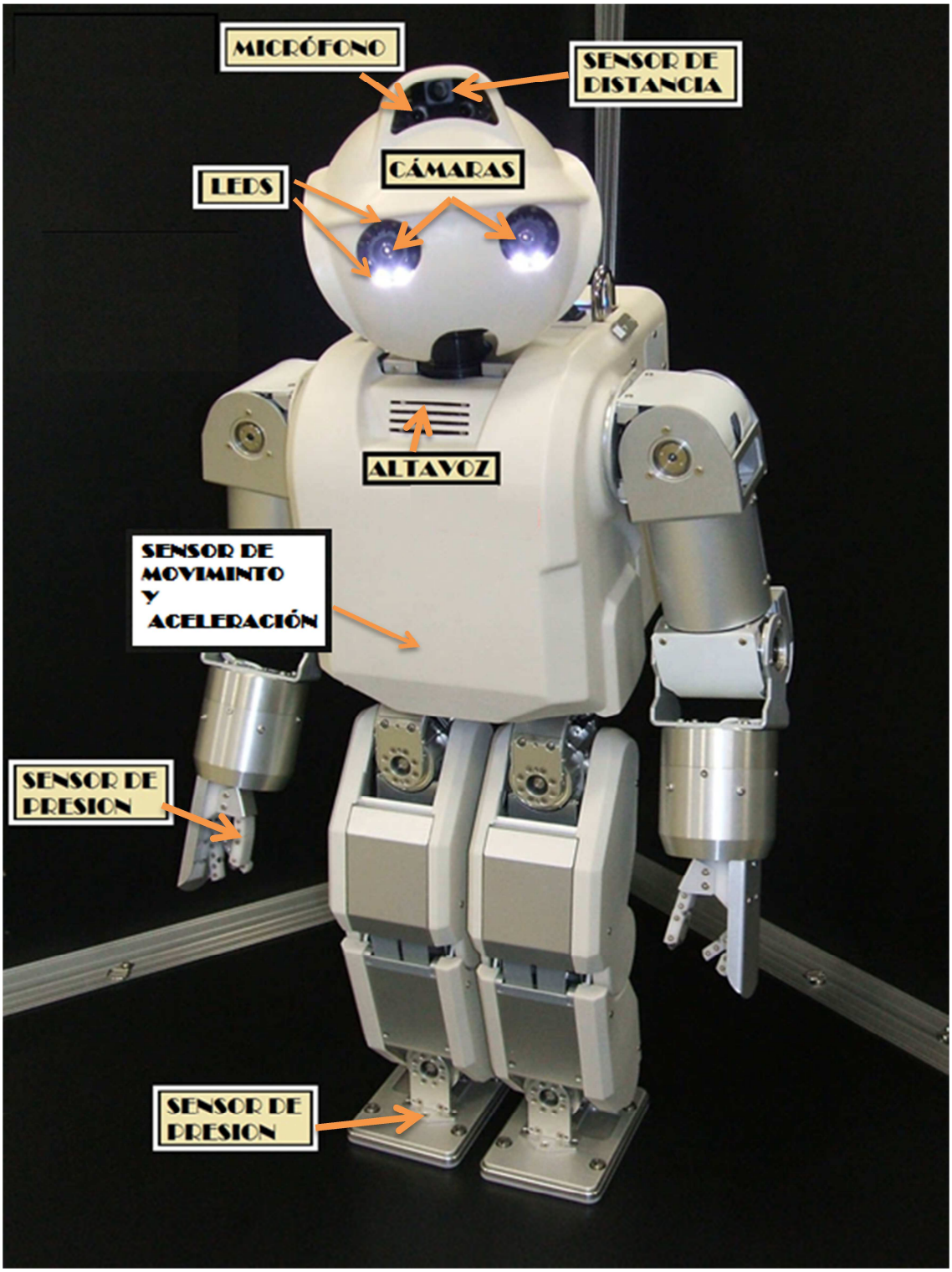


Fig. 4.1.3 Sensores HOAP-3.

Incorpora un sistema de comunicación por WIFI. Esto se traduce en un manejo más versátil y cómodo ya que desde una distancia media se puede tener comunicación bilateral con el robot ya sea para darle órdenes, comandos específicos o incluso hacer una parada de emergencia sin necesidad de cables.

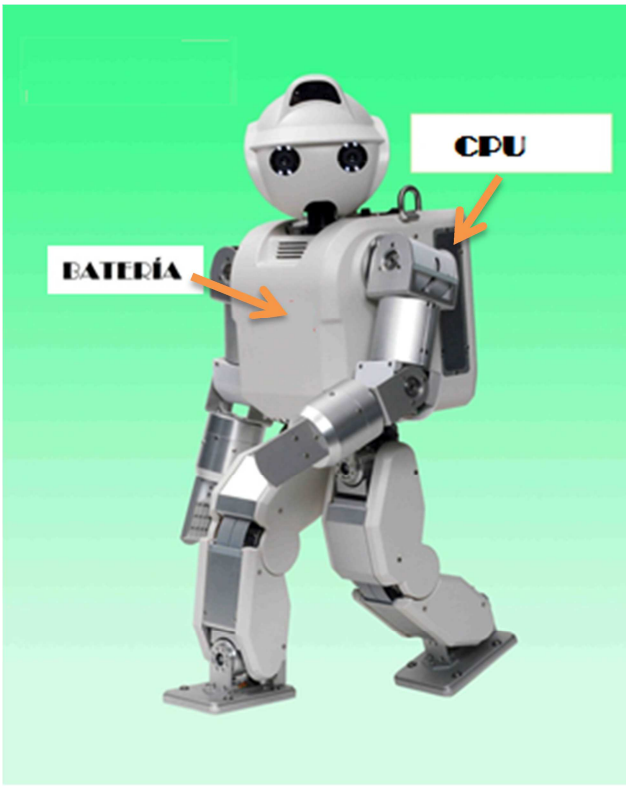
La capacidad de memoria de este robot es de 512 MB de RAM y una memoria Compact Flash de 1GB. Esta memoria se usa tanto para el SO como para el resto de software.

El Sistema Operativo de Linux utiliza un controlador llamado ALSA para configurar y manejar todo el sistema de sonido del robot. ALSA (Advanced Linux Sound Architecture) tiene librerías y funciones recogidas en unos APIs para programadores.

El Robot HOAP-3 dispone solamente de 23 motores. Se pueden controlar en posición y en velocidad gracias a unos encoders relativos ligados a cada motor. 21 de los 23 motores, controlan el movimiento de los brazos y las piernas y los restantes son para el agarre y rotación de la muñeca y la cabeza.

Tiene una batería de Ni-Mh de 24V para alimentar tanto a los motores como a la circuitería interna. Esta batería está dentro de su pecho. La autonomía máxima es de aproximadamente 30 minutos. La CPU está camuflada en la mochila que lleva colgada en la espalda.

En la Fig. 4.1.4 vemos la distribución de la batería y la CPU.



10 Fig. 4.1.4 Batería y CPU

4.2.- PROGRAMAS

4.2.1.- SÍNTESIS DE VOZ

Los desarrolladores de Fujitsu incorporaron en su versión comercial un software de síntesis de voz y reconocimiento de voz llamado VORERO. Este programa es capaz de sintetizar voz robótica a partir de un texto plano. El único requisito es que el texto esté en japonés. Tiene varios ejemplos donde cuenta los números, saluda y dice la compañía que le ha creado.

En un principio se intentó utilizar este software analizando las funciones y su contenido pero sin el diccionario en inglés, ni el modelo acústico, se hizo una tarea imposible.

Se decidió entonces probar el código de libre distribución llamado *The Festival Speech Synthesis System*. Utiliza la arquitectura de sonido ALSA configurándola y adaptándola a las necesidades de reproducción cuando es necesario. Como el programa es tan sencillo, se ha instalado directamente en el sistema operativo Fedora (Linux) en su versión más ligera, se ha cargado un diccionario y un modelo acústico en inglés americano y se ha probado desde el terminal de Linux.

Posee una interfaz en el terminal para introducir comandos específicos para ejecutar órdenes de síntesis. Uno de los comandos consiste en poner en la interfaz del programa la siguiente línea:

```
festival>SayText "TEXTO"
```

Donde pone la palabra TEXTO se sustituye por la frase que se quiere sintetizar, en nuestro caso al estar preseleccionado el idioma inglés, deberá ser una frase en inglés.

Otra forma de usar el programa es mediante scripts que tiene este software. Los scripts son líneas de código que forman un programa que se ejecuta independientemente, sin necesidad de usar la interfaz de Festival. Desde el terminal se puede introducir el siguiente comando:

```
$ festival – tts archivo
```

Donde pone ***archivo*** se tiene que poner el nombre de un documento de texto plano acabado en .scm por ejemplo ***hola.scm*** creado aparte que se ubique en la misma carpeta que el ejecutable del programa en curso y que contenga el siguiente texto:

```
## hola.scm ##
(SayText "TEXTO")
```

Por tanto para ejecutar este script de festival tenemos que poner:

```
$ festival – tts hola.scm
```

El funcionamiento de este script es sencillo, abre el programa festival de forma oculta con << festival – tts ***archivo*** >> y escribe en la interfaz de festival lo que contenga el archivo, es este caso << (SayText "***TEXTO***") >>. Finalmente cierra la interfaz de festival y finaliza el script habiendo sintetizado y reproducido lo que contenía ***TEXTO***.

A lo largo de la creación del presente proyecto, se ha decidido que la mejor forma de tratar el punto de la síntesis de voz, es recurrir a esta programación, así cuando el robot ejecute una orden, se lanzará el script correspondiente a esa orden. Este modelo de programación no es el más adecuado ya que necesita un script para cada tarea que se programe (con un archivo .scm) distinto para cada orden de voz. Por ejemplo, si el robot recibe la orden de saludar, aparte de los movimientos oportunos del saludo, tiene que sintetizar el saludo. Es necesario el archivo ***hola.scm*** que contiene (SayText "***hello my friend***"). No es el mejor debido a su difícil modificación archivo por archivo si se necesita cambiar el programa de síntesis o alguna modificación menor.

En la Tab. 4.2.1.1, hay una lista con todas las posibles órdenes que el robot puede ejecutar y que se ha tenido en cuenta:

Tab. 4.2.1.1 Frases sintetizadas

hello.scm
(SayText "***hello my friend***")

goodbye.scm
(SayText "***goodbye my friend***")

moveHeadDown.scm
(SayText "***command received. move head down***")

moveHeadUp.scm
(SayText "***command received. move head up***")

moveHeadLeft.scm
(SayText "***command received. move head left***")

moveHeadRight.scm
(SayText "***command received. move head right***")

moveWalkingFoward.scm
(SayText "***command received. move walking foward***")

moveWalkingBackward.scm
(SayText "***command received. move walking backward***")

moveTurningRight.scm
(SayText "***command received. move turning right***")

moveTurningLeft
(SayText "***command received. move turning left***")

notNumbers.scm
(SayText "***You must tell me the number***")

notUnderstand.scm
(SayText "***I don't understand, Can you repeat that?***")

Al llegar a un punto crítico donde el robot deba realizar una acción, se ejecuta el script de festival abriendo alguno de estos archivos ".scm" y se sintetiza claramente la voz robótica referente a la correspondiente orden.

Para ejecutar el script desde el programa en ejecución del HOAP-3 (en lenguaje C), se hace una llamada a la función system. Esta función se encarga de abrir un terminal oculto y ejecutar exactamente la línea escrita a continuación. Por ejemplo:

```
if (a!=0)
{
        system("festival –tts hello.scm);
}
```

System escribe en el terminal:

```
$ festival –tts hello.scm
```

Esta acción lanza el programa festival y ejecuta lo que contiene hello.scm, es decir, sintetizará la voz robótica del texto "***hello my friend***". El robot dirá por los altavoces con una voz clara hello my friend.

Para poder ejecutar los archivos .scm, hay que darle la ruta donde se encuentran al ejecutar el script. As si está en la ruta /home/Desktop/hoap/scm el comando final será:

```
$ festival –tts hello.scm /hoap/scm
```

Una vez visto el funcionamiento del programa, para poder implementarlo en el robot HOAP-3, se ha utilizado un código ya existente. Este código contiene todas las rutinas necesarias para realizar los movimientos ya programados tales como caminar, girar, saludar y mover la cabeza hacia los distintos lados.

En líneas muy concretas del código, antes de ejecutar la acción o después (dependiendo de la acción) se ha introducido el comando *system* y la orden concreta para sintetizar el archivo correspondiente a la acción asociada. Así por ejemplo, cuando se deba ejecutar la acción de saludar, justo en las siguientes líneas de código se ha introducido:

```
If (…
{
        Do…
        System(“festival –tts saludo.scm/hoap/scm);
}
```

Con estas líneas conseguimos que el robot sintetice exactamente lo que necesitemos en el momento oportuno.

4.2.2.- RECONOCIMIENTO DEL HABLA

VORERO también se puede usar para reconocer palabras mediante la voz y transformarlas a texto plano. Este programa lleva incorporado un sofisticado software que reduce el ruido ambiente del micrófono del HOAP-3 y procesa la onda de audio. En un principio, cuando se descubrió el potencial de este software se hizo un pequeño estudio de sus funciones y del código correspondiente. Una vez visto el código, VORERO tiene un motor de búsqueda para comparar la onda de voz procesada y troceada con elementos acústicos (fonemas) en una base de datos llamado diccionario. El problema de utilizar este código es que se dispone únicamente del diccionario en japonés.

A partir de ese momento se decidió utilizar CMU SPHINX en la versión de código C más sencilla y rápida (aunque un poco menos precisa). La memoria del robot HOAP-3 está reservada por programas de visión y movimientos, por eso se pensó en su momento utilizar un procesador aparte donde esté instalado CMU Pocketsphinx. Este PC no está conectado directamente al robot HOAP-3, se comunica con el robot HOAP-3 mediante el protocolo de comunicación TelNet (TELecommunication NETwork) vía WI-FI.

El programa pocketsphinx es capaz de realizar un reconocimiento automático del habla. Esto quiere decir que según se vayan dando órdenes al robot, este programa las procesa sin necesidad de reiniciar el programa con cada orden.

Funciona de la siguiente manera:

En un principio, es necesario inicializar todas las funciones de ALSA y configurar el micrófono. Esto se hace gracias a unas funciones de inicialización de pocketsphinx que se encargan de todo. Acto seguido para poder lanzar un reconocimiento automático continuo, es necesario tener en cuenta una variable de las ondas de audio llamada TimeStamp. Esta variable se encuentra en una estructura de pocketsphinx. Cada vez que se graba una onda de audio del micrófono, esta variable cambia si no se detecta una perturbación mínima. Va aumentando de valor si “no se escucha nada” o si lo que se está grabando es “silencio”. Cuando percibe una perturbación significativa, esta variable vuelve a ser 0, aumentando de valor con los “silencios”. Gracias a esta variable, podemos configurar el programa para que admita órdenes separadas por intervalos de tiempo.

Si por ejemplo, queremos darle al robot la orden “saluda” y la orden “avanza”.

Como el sistema es de reconocimiento automático del habla que además es continuo, se tiene que dar primero una orden y esperar un segundo para que el programa de reconocimiento del habla procese la información y se la mande al robot. Cuando el robot haya acabado su tarea, el programa retoma el control y se le puede enviar la siguiente orden de avanzar.

Para este programa, ejecutamos un bucle donde estamos continuamente grabando del micrófono. Cuando la variable TimeStamp llegue a

un valor determinado (en este caso el necesario para que sea 1 segundo de silencio) sale del bucle y procesa el texto. Este texto sufre un post-procesado antes de enviar la orden definitiva al robot. Este post-procesado, hace una comparación palabra por palabra para ver qué orden estamos queriendo enviar y qué cantidad (de pasos, de ángulos) es la indicada y si está o no en el rango.

Este tipo de programación es únicamente por seguridad tanto del robot como de evitar situaciones no deseadas (como por ejemplo enviar sin querer que avance 90 pasos).

Una vez ejecutado el programa de reconocimiento del habla con el software de libre distribución pocketsphinx, recogemos valores en cadena de caracteres que enviamos al HOAP-3 después del pequeño post-procesado del texto.

El envío final de las órdenes se hace mediante la conexión WIFI utilizando sockets TCP.

Como diccionario seleccionado se ha elegido el diccionario en inglés.

5 RESULTADOS EXPERIMENTALES

En este apartado muestro los resultados obtenidos de haber implementado tanto la síntesis de voz como el reconocimiento del habla en el robot HOAP-3 de FUJITSU.

5.1.- DEMOSTRACIÓN

5.1.1.- SINTESIS DE VOZ

Los primeros resultados de la síntesis de voz fueron en el laboratorio de robótica en un PC. Desde el sistema operativo UBUNTU 10.04 (Linux) se lanzó el programa FESTIVAL TEXT TO SPEECH. Se utilizó una voz de varón en el idioma inglés americano (*voiceKal-diphone*). La primera impresión del sonido de la "voz" de este programa es bastante pobre. Es muy robótica pero se entiende a la perfección las palabras y hace las pausas precisas.

Al tratarse de un sistema que utiliza concatenación por formantes, el resultado que se podría esperar sería una voz más depurada con algunos aspectos fónicos más claros, mejor prosodia y más parecidos a la voz humana. Como FESTIVAL da la posibilidad de cambiar la "voz" a la de otro varón o a la de una mujer, se hicieron pruebas en el laboratorio antes de experimentar con el robot HOAP-3. Los resultados de estas pruebas sí fueron más alentadores ya que se utilizó una "voz" menos robótica, de mujer y mucho más clara, con mejor prosodia. Esta voz utilizada se recogió de MBROLA (*voice us1-mbrola*). Sigue estando en inglés americano.

Probadas las voces y elegida (*voice us1-mbrola*) se pasó a probar la "voz" en español (v*oice El-diphone*). Esta voz es aceptable pero ligeramente peor que la voz inglesa. Tiene una entonación neutra y sin depuración de prosodia ninguna. Se entienden perfectamente todas y cada una de las frases sintetizadas, se hacen las pausas correctas. El único error que se le puede sacar es que no está contemplada la letra "ñ" en el sistema de síntesis de esta "voz" y al probarla, nos dice directamente el número en código ASCII de la letra ñ, pero no la letra en sí.

Después de hacer estas pruebas en un PC del laboratorio, con el código en C dispuesto, lo llevamos al robot HOAP-3. Se instaló el programa FESTIVAL Text-to-Speech en el sistema operativo del robot con su versión más simplificada para ahorrar el máximo espacio de memoria y debido a las limitaciones de FEDORA core 1 (Kernel 2.4) con respecto a UBUNTU 10.04.

Antes de lanzar el código creado en C donde se usaron librerías de FESTIVAL, se probó que FESTIVAL se instaló correctamente. Se Lanzó la interfaz de festival y se probó que funcionaba todo correctamente (el altavoz y el software). Desde la consola de Linux y con la interfaz de FESTIVAL, se introdujo simplemente el siguiente comando:

```
festival>(SayText “HELLO WORLD”)
```

Este comando introducido directamente, hizo “hablar” al robot por primera vez en inglés (ya había hablado previamente pero sólo en japonés). Después de las pruebas iniciales y viendo lo fácil que resultaron estas pruebas, se decidió entonces a cambiar las voces para mejorar el resultado.

Al tener instalada la versión más ligera de FESTIVAL las voces de MBROLA no eran ya compatibles con esta versión y por tanto quedaron descartadas al instante. De las voces disponibles se decidió instalar (*voiceKal-diphone*) y (voice El-diphone) debido al gasto de memoria que requerirían más voces.

Ya que el software de libre distribución de FESTIVAL funcionaba. Se decidió probar más adelante el programa creado en C utilizando librerías de FESTIVAL.

Se crearon los archivos .scm donde se encontraban las frases a sintetizar con el diccionario instalado por defecto.

5.1.2.- RECONOCIMIENTO DEL HABLA

En esta parte del objetivo final del proyecto, se ha invertido prácticamente todo el tiempo. Las pruebas fueron realizadas en el PC del laboratorio. Se instaló el software de libre distribución POCKETSPHINX en la distribución de Linux UBUNTU 10.04.

Este programa realiza un análisis de la onda sonora recogida por el micrófono y guarda en una variable la cadena de texto aproximada de esa onda de sonido. No importa la voz del locutor, si es masculino o femenino o si es adulto o niño.

El código creado en este programa, realiza un reconocimiento continuo y automático del habla, quiere decir que no hace falta intervenir para que continúe reconociendo la voz.

Se cargó la base de datos de estados ocultos de Markov (HMM) más completo del idioma inglés. Este sistema estadístico se ha formado gracias a las donaciones anónimas de todo el mundo que quiera colaborar comparando sus voces con las ondas de audio de cada sonido. Incluso se ha adaptado para las distintas pronunciaciones (norteñas o sureñas de América o británica).

Además de la base de datos estadística (HMM), es necesario un diccionario léxico para unir los sonidos con las palabras que quieres que se guarden en el array de caracteres que guarda el resultado del reconocimiento.

Este diccionario es muy importante ya que si no recoge alguna palabra, aunque se recoja en el micrófono, nunca será reconocida.

Otro archivo que se puede cargar en el programa es una GRAMÁTICA DE ESTADOS FINITOS (FSG). Este archivo determina el orden de salida de las palabras y mejora el reconocimiento de las frases sencillas. Su funcionamiento es el siguiente: se crean variables que contienen palabras. Esas variables se ponen en un determinado orden de la posible frase a generar.

Por ejemplo si la frase es: *"Robot avanza 2 pasos"* o *"Robot retrocede 3 pasos"*, se ve que tienen en común que empiezan y acaban con las mismas palabras. Por lo tanto estas palabras son fijas. Luego avanza y retrocede están en la misma posición, esa será la primera variable que contendrá a las dos palabras y como segunda variable contendrá los números. De esta manera el programa sabe cuál es la palabra que tiene que buscar y el orden de estas palabras haciendo más sencillo el reconocimiento del habla. Este tipo de archivos sólo se usan para reconocimiento de frases concretas y de habla específica sobre un tema, no es válido para habla de propósito general.

El archivo utilizado se puede encontrar en los documentos anexos [8.3].

Una vez en marcha el programa, el resultado del reconocimiento de voz sin la gramática de estados finitos (FSG) es muy pobre. El número de errores es elevadísimo y esto se debe a que el idioma elegido es el inglés y la pronunciación influye mucho si se utiliza un idioma no nativo para el reconocimiento.

Una vez ajustado el FSG y cargado al programa, el resultado es bastante aceptable. En el laboratorio se ha conseguido un reconocimiento del habla para un tema muy determinado con unas frases concretas con un alto acierto.

Para asegurar todavía más los aciertos y el orden de las palabras, se ha hecho un post-procesado del texto reconocido. Este post-procesado normalmente es necesario en los programas que usa reconocimiento del habla debido a la posibilidad real de ejecutar acciones no deseadas por una mala transcripción de voz a texto. Este post-procesado se encarga de hacer un análisis rápido palabra por palabra guardada en el array de caracteres. Si se detecta una palabra descolocada o sin sentido, el programa envía al robot una orden para que sintetice la frase *"I don't understand, Can you repeat that?"* Si el post-procesado detecta una frase que sí contempla, envía un socket TCP con la información precisa al robot para que ejecute la orden correcta dicha por el locutor.

Como el robot tiene una memoria muy limitada, se decidió utilizar un ordenador portátil que hiciera el trabajo de reconocimiento del habla. Se cargó el programa creado en C en el PC auxiliar. Mientras, en el robot HOAP-3 se ejecutaba un programa que está a la espera de recibir órdenes.

Este programa cargado en el robot, tiene como objetivo principal ejecutar las tareas de los movimientos de los motores ejecutando scripts ya escritos y probados, así cuando llegado el momento, se necesite realizar la tarea de saludar, este programa es el encargado de lanzar ese script.

En la Fig. 5.1.1 se puede ver la composición del sistema final y un pequeño flujograma para generar una idea aproximada de la composición del sistema:

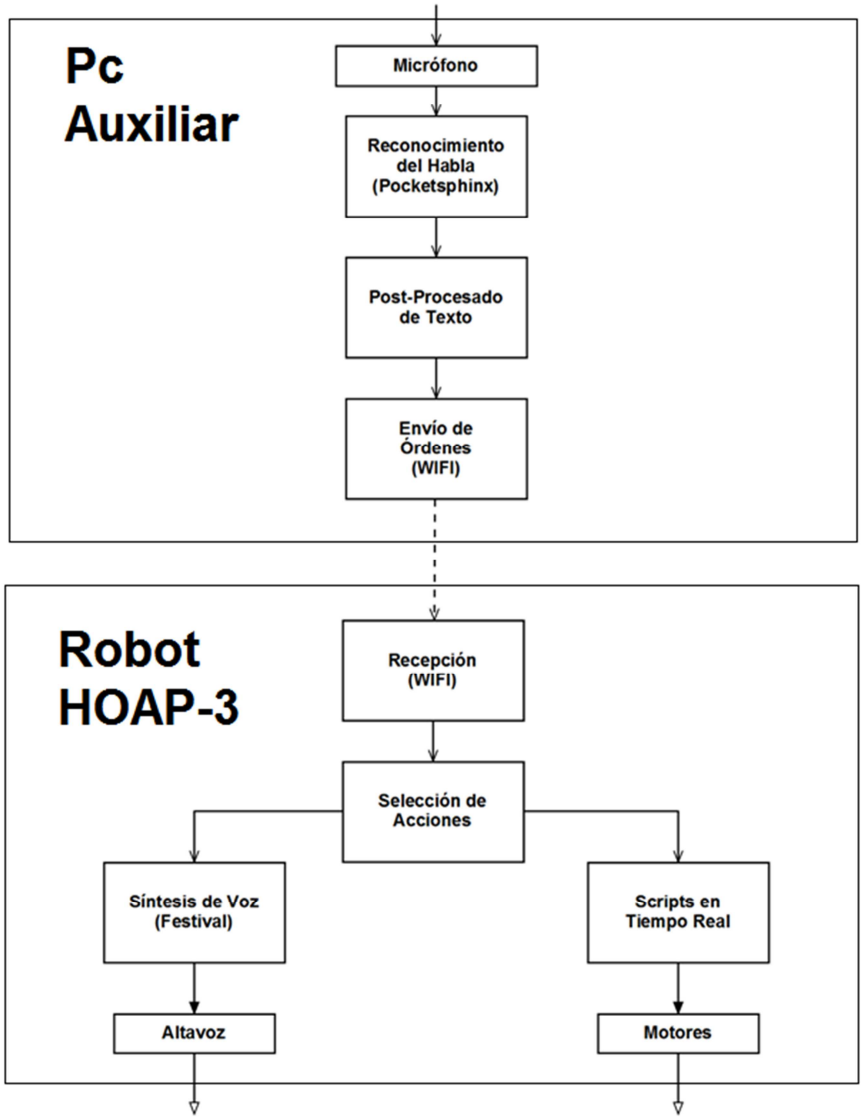


Fig. 5.1.1 Arquitectura

Debido a los motivos antes expuestos, es necesario hacer el post-procesado del texto reconocido mediante el habla. Este post-procesado realiza una comparación con todas y cada una de las posibles órdenes dadas al robot y que están contempladas en la programación del mismo. Incluso se ha tenido en cuenta la cantidad introducida en números y comparado con cada número por separado.

En el primer caso, el locutor desea que el robot salude a los espectadores. Para ello, la primera orden debe ser "ROBOT" para que sepa que están hablando con él. Si esta palabra no aparece en la orden, el robot no debe ejecutar ninguna acción. Acto seguido, el locutor debe mencionar la orden expresa, como por ejemplo, “HELLO”, esta orden, lanza el script de saludar, cuando ejecuta el saludo, por los altavoces se genera la frase “HELLO MY FRIEND”.

Debido a los movimientos programados, se deben dar unas órdenes específicas para cada movimiento si lo que queremos es que ande hacia delante o gire o mueva la cabeza. Las órdenes se aprecian en la siguiente Tab. 5.1.2

Tab. 5.1.2 Órdenes Linguísticas

ROBOT	HELLO			
	MOVE	WALKING	FOWARD	Nº STEPS
			BACKWARD	Nº STEPS
		TURNING	LEFT	Nº DEGREES
			RIGHT	Nº DEGREES
		HEAD	UP	Nº DEGREES
			DOWN	Nº DEGREES
			LEFT	Nº DEGREES
			RIGHT	Nº DEGREES
	GOODBYE			

Si el deseo es que el robot mueva la cabeza hacia arriba 10 grados se tendrá que dar la orden siguiente:

“ROBOT MOVE HEAD UP 10 DEGREES”

Si el reconocimiento ha ido según lo esperado, el post-procesado de este texto deberá recorrer cada palabra por separado haciendo una comparación y asegurándose de seguir el camino correcto de la orden dada. Comparará en primer momento si está la palabra ROBOT o no en el inicio de la frase, acto seguido comparará la siguiente palabra con las tres únicas opciones que hay: “HELLO”, “MOVE” o “GOODBYE”. En este caso aparece “ROBOT MOVE” y por lo tanto el siguiente paso es chequear cuál de las 3 posibilidades dentro de “MOVE” se ha reconocido. En el caso de no reconocer ninguna de las tres opciones posibles, saltará a un estado en el cual el reconocimiento no ha tenido éxito y dirá “I DON’T UNDERSTAND, CAN YOU REPEAT THAT?”. Con este post-procesado se van recorriendo cada una de las palabras del array.

Una vez llegados al número de pasos que queremos que ande o al número de grados que queremos que gire, se hace una comparación número por número hasta un máximo de 99 unidades. Este algoritmo de búsqueda de los números es más sencillo de lo que parece ya que en el lenguaje inglés, los 9 primeros números se repiten para formar los siguientes 99 (al igual que el español). Por tanto si queremos formar el 35 chequearemos la palabra "THIRTY" y si se ha encontrado, la siguiente palabra será uno de esos 9 primeros dígitos: "ONE", "TWO", etc. Si no se encuentra ninguno de esos números entonces estaremos en el caso del número 30.

Este mecanismo ha hecho posible ahorrarse muchas líneas de código. Como comodidad, se ha despreciado el post-procesado del tipo de unidades. Si se llega a la conclusión en el post-procesado que el locutor quiere que el robot mueva la cabeza 10 grados hacia arriba siguiendo la trayectoria: "ROBOT MOVE HEAD UP 10", ya no sigue buscando si la siguiente palabra son grados "DEGREES" o pasos "STEPS" ya que no es necesario para la orden aunque el locutor se confunda en tipo de unidades. Como en las órdenes no hay ninguna que trabaje a la vez con pasos o con grados, no hay lugar a confusión de unidades y se puede despreciar en el post-procesado.

Para poder separar las palabras una por una del puntero devuelto por el programa de reconocimiento de voz, se ha usado una función que trocea la cadena con toda la frase en una matriz que contiene una única palabra por línea y las letras de esa palabra por cada fila.

En la siguiente Tab 5.1.3 se muestra un par de ejemplos:

Tab. 5.1.3 Detección de palabras

	LETRAS							
Nº PALABRA	R	O	B	O	T	'/0 '		
	H	E	L	L	O	'/0'		
	R	O	B	O	T	' /0'		
	M	O	V	E	'/0'			
	H	E	A	D	'/0'			
	U	P	'/0 '					
	T	E	N	'/0 '				
	D	E	G	R	E	E	S	'/0'

El algoritmo es muy sencillo, se va copiando lo que contiene la cadena, letra por letra, en la matriz en el primer espacio y en la primera fila. Cuando el puntero que realiza el barrido, encuentra un espacio (el correspondiente carácter en código ASCII), guarda esa dirección y a partir de ahí empieza a copiar la siguiente palabra en la siguiente fila de la matriz, sin olvidar de ponerle un "NULL" o final de carro para saber que la palabra se ha acabado.

5.2.- ACIERTOS/ERRORES

Como el sistema utilizado para el reconocimiento de voz se basa en estados o modelos ocultos de Markov (HMM), es un sistema puramente probabilístico. Esto se traduce en que existen posibilidades en las cuales el reconocimiento no sea del todo acertado. Por eso es necesario hacer un pequeño estudio para ver el porcentaje de aciertos y errores que tiene el sistema de libre distribución SPHINX en su versión para código C más compacta y portable llamada POCKETSPHINX.

Este reconocimiento no es de propósito general y se han tenido en cuenta solo algunas palabras concretas y el orden de éstas. Se ha decidido hacer el estudio para todos los posibles casos de reconocimiento de estas palabras.

El análisis se ha realizado en cuatro aspectos:

1º Mediante la repetición de palabras clave por separado (sin el FSG cargado en el motor de búsqueda).

2º Mediante la repetición de las frases enteras contempladas en la programación y post-procesado excluyendo números (Cargando el FSG en el motor de búsqueda).

3º Creando un programa aparte para el reconocimiento de números.

4º Haciendo un barrido de órdenes íntegras que son aceptadas en el post-procesado (con el FSG cargado en el motor de búsqueda).

Este análisis ha sido realizado en una habitación con un nivel bajo de ruido externo y en completo silencio. Se ha usado un micrófono comercial de baja calidad y sin atenuación de ruido por software.

5.2.1.- ANÁLISIS DE PALABRAS CLAVE (SIN FSG)

Tab. 5.2.1 Análisis palabras clave

PALABRA	REPETICIONES	ACIERTOS
ROBOT	20	18
HELLO	20	20
GOODBYE	20	16
MOVE	20	19
TURNING	20	10
WALKING	20	14
HEAD	20	15
LEFT	20	18
RIGHT	20	17
FOWARD	20	14
BACKWARD	20	10
UP	20	18
DOWN	20	17

Como puede observarse en la Tab. 5.2.1, el porcentaje de aciertos de algunas palabras varía muchísimo llegando desde el 50% de aciertos hasta el 100% para determinadas palabras.

Esta variación puede deberse a muchos aspectos tanto del programa como del locutor. Uno de los aspectos más importantes es la pronunciación. El diccionario cargado en el motor de búsqueda junto con el diccionario usado con la pronunciación británica, hace este número de aciertos realmente bajo si quiere usarse en frases que contengan más de una de estas palabras (ya que si no falla una en ese caso, falla otra incrementando exponencialmente el número real de fallos). Otro aspecto a destacar es que el % de aciertos disminuye si una persona trabaja en el reconocimiento del habla en un lenguaje que no es el materno, ya que la pronunciación no tiende a ser exacta, produciendo un descenso considerable en el número de aciertos.

También cabe decir que el número de aciertos del programa de reconocimiento automático del habla creado para este propósito aumenta después de las repeticiones de la misma palabra. Es debido a la ligera modificación tanto de entonación como de la pronunciación por parte del locutor.

5.2.2.- ANÁLISIS DE FRASES (CON FSG) EXCLUYENDO NÚMEROS

Tab. 5.2.2 Análisis frases

FRASE	REPETICIONES	ACIERTOS
ROBOT HELLO	20	19
ROBOT GOODBYE	20	19
ROBOT MOVE WALKING FOWARD	20	17
ROBOT MOVE WALKING BACKWARD	20	15
ROBOT MOVE TURNING RIGHT	20	14
ROBOT MOVE TURNING LEFT	20	13
ROBOT MOVE HEAD UP	20	16
ROBOT MOVE HEAD DOWN	20	16
ROBOT MOVE HEAD RIGHT	20	15
ROBOT MOVE HEAD LEFT	20	13

Los resultados de la Tab. 5.2.2 son excelentes comparados con las palabras sueltas debido al FSG cargado en el motor de búsqueda. Para programas de propósito general, es imposible configurar un FSG, pero para programas específicos que tengan pocas opciones para contemplar, es una herramienta realmente apropiada para la tarea porque aumenta con mucha diferencia el número de aciertos del programa de reconocimiento de voz.

Se han eliminado los números ya que son muchos los casos que se tendrían que contemplar para cada frase (99 veces cada frase para los distintos números y 20 repeticiones de cada una).

5.2.3.- ANÁLISIS NÚMEROS (SIN FSG)

Tab. 5.2.3 Análisis números

NÚMERO	REPETICIONES	ACIERTOS
1	20	11
2	20	6
3	20	9
4	20	7
5	20	10
6	20	18
7	20	14
8	20	8
9	20	16
10	20	18
11	20	16
12	20	14
13	20	13
14	20	13
15	20	14
16	20	15
17	20	13
18	20	13
19	20	15
20	20	17
30	20	14
40	20	15
50	20	14
60	20	11
70	20	11
80	20	12
90	20	13

Este análisis se ha realizado en la siguiente Tab. 5.2.3 con los números en inglés. Existe una gran diferencia respecto a algunos números debido a la similitud de palabras como por ejemplo “TWO – TO” o “THREE – TREE” realmente la pronunciación es algo similar pero no tanto como para que un locutor los confunda. Sobre todo, el sistema de reconocimiento ha tenido problemas con los números acabados en –teen (del 13 al19) y las decenas, acabados en –ty (del 20 al 90).

El programa tiene la ventaja de que haya reconocido o no la palabra correctamente, siempre devuelve una palabra al menos. Esto quiere decir que al realizar el estudio en ningún momento el programa se quedó sin valor y sin reconocer nada.

Si cotejamos los resultados obtenidos de este reconocimiento, sacamos la conclusión que el reconocimiento de los números es algo más complejo y no es tan obvio para una máquina que para nosotros. Si contamos también con la composición de los números tales como “22” “TWENTY TWO” que son dos palabras separadas, el porcentaje de acierto de nuevo disminuye exponencialmente al tener la necesidad de acertar ambas palabras (si una no falla, puede fallar la otra).

Este problema no se ha podido solucionar en este trabajo, el acierto es realmente bajo y así se ha implementado en el robot HOAP-3. El post-procesado de los números se ha hecho por bloques, en el primer bloque se tienen los 19 primeros números que son de una sola palabra, y los demás como una composición de una decena y un número del 1 al 9.

Este post-procesado no corrige el número mal reconocido por el programa POCKETSPHINX, únicamente comprueba si realmente es un número lo que el locutor ha dicho y el programa ha reconocido. En algunos casos se ha reconocido una palabra que no es un número y el post-procesado tiene como único objetivo evitar que se envíe la orden.

5.2.4.- ANÁLISIS DE ORDENES FINAL (CON FSG) INCLUYENDO NÚMEROS.

Con este propósito se ha elegido el número 10, el 15 y el 20 para el número de ángulos girado por la cabeza o el cuerpo y números bajos (1, 2, 3, 4, 5) para pasos acorde a las órdenes más comunes que se van a ejecutar para este robot. Los resultados se muestran en la Tab. 5.2.4.

Tab. 5.2.4 Análisis órdenes completas

ORDEN	REPETICIONES	ACIERTOS
ROBOT HELLO	20	17
ROBOT GOODBYE	20	18
ROBOT MOVE WALKING FOWARD ONE STEP	20	13
ROBOT MOVE WALKING FOWARD TWO STEPS	20	10
ROBOT MOVE WALKING FOWARD THREE STEPS	20	11
ROBOT MOVE WALKING FOWARD FOUR STEPS	20	11
ROBOT MOVE WALKING FOWARD FIVE STEPS	20	11
ROBOT MOVE WALKING BACKWARD ONE STEP	20	13
ROBOT MOVE WALKING BACKWARD TWO STEPS	20	12
ROBOT MOVE WALKING BACKWARD THREE STEPS	20	13
ROBOT MOVE WALKING BACKWARD FOUR STEPS	20	10
ROBOT MOVE WALKING BACKWARD FIVE STEPS	20	13
ROBOT MOVE TURNING RIGHT TEN DEGREES	20	9
ROBOT MOVE TURNING RIGHT FIFTEEN DEGREES	20	10
ROBOT MOVE TURNING RIGHT TWENTY DEGREES	20	9
ROBOT MOVE TURNING LEFT TEN DEGREES	20	14
ROBOT MOVE TURNING LEFT FIFTEEN DEGREES	20	15
ROBOT MOVE TURNING LEFT TWENTY DEGREES	20	12
ROBOT MOVE HEAD UP TEN DEGREES	20	17
ROBOT MOVE HEAD UP FIFTEEN DEGREES	20	15
ROBOT MOVE HEAD UP TWENTY DEGREES	20	14
ROBOT MOVE HEAD DOWN TEN DEGREES	20	13
ROBOT MOVE HEAD DOWN FIFTEEN DEGREES	20	12
ROBOT MOVE HEAD DOWN TWENTY DEGREES	20	13
ROBOT MOVE HEAD RIGHT TEN DEGREES	20	14
ROBOT MOVE HEAD RIGHT FIFTEEN DEGREES	20	14
ROBOT MOVE HEAD RIGHT TWENTY DEGREES	20	15
ROBOT MOVE HEAD LEFT TEN DEGREES	20	15
ROBOT MOVE HEAD LEFT FIFTEEN DEGREES	20	13
ROBOT MOVE HEAD LEFT TWENTY DEGREES	20	14

El número de aciertos en la ejecución de las acciones es mayor, pero he tenido en cuenta cuando el motor de reconocimiento de voz no acertaba el número tanto de grados como de pasos.

6 CONCLUSIONES

Después de haber trabajado con reconocimiento del habla y síntesis de voz, se puede concluir que no se trata de un sistema 100% fiable. De los programas probados: PocketSphinx y Fonix para el reconocimiento del habla, el primero no ha dado más que un porcentaje bastante significativo pero nada fiable. El segundo no se ha conseguido configurar el ruido del micrófono ya sea por los problemas a la hora de tratar con ALSA o por la baja calidad del micrófono.

Para el apartado de la síntesis de voz, el programa Festival, aun siendo de libre distribución, se ha conseguido un resultado óptimo para el uso que se le ha dado en este proyecto. La voz es robótica pero clara y muy fácilmente entendible.

Una buena forma de tratar el tema el reconocimiento del habla es usar el que ya está funcionando en los "SmartPhone" gracias a la tecnología de Google para el sistema operativo Android. En los nuevos iPhone 4, existen a aplicaciones de reconocimiento de voz que son muy fiables teniendo un alto índice de aciertos. Ambos sistemas deben estar conectados a internet. Comparan las ondas de voz recogidas por el micrófono con una base de datos donde están almacenados los patrones. Esta búsqueda se hace en tiempo real y el tiempo de cómputo es mínimo. Los aciertos son muy elevados en comparación con cualquier sistema que se ha probado en este proyecto.

Para el entorno de desarrollo de todo el software, se ha usado un programa llamado Qt-Creator bajo el sistema operativo de Ubuntu en la versión 10.04. Este programa tiene todo lo necesario para poder trabajar en código C o C++ (usado en este proyecto).

Para poder usar las funciones ya escritas de otros programas, es necesario cargar las librerías correctamente y enlazarlas con nuestro programa. Para este cometido se ha utilizado una herramienta llamada Cmake en el terminal de Linux.

Siempre que se usen las librerías de cualquier software, ya sea bajo licencia comercial o de libre distribución, es necesario hacer un estudio previo para utilizar las librerías. Este estudio es posible gracias a las APIs. En esta documentación se encuentran todas las funciones, los valores necesarios para usarlas y los distintos valores devueltos junto con el objetivo de cada una. La conclusión obtenida en este apartado es que antes de ponerse a programar es necesario este paso ya que se ahorra mucho tiempo y esfuerzo pudiendo aprovechar todas las funciones que sean necesarias sin necesidad de programarlas uno mismo.

6.1.- OBJETIVOS CUMPLIDOS

El primer objetivo propuesto en este proyecto, fue en el marco teórico, realizar un estudio de cómo se produce la síntesis de voz y el reconocimiento del habla. Este apartado se ha cumplido en el capítulo 2 donde se recogen las bases y fundamentos que hacen posible la capacidad de sintetizar y reconocer la voz. En el capítulo 3 se hace un estudio de los programas que hacen posible la implementación.

El siguiente objetivo fue sintetizar voz usando el robot humanoide HOAP-3. Este paso se ha conseguido en el capítulo 4, donde se recogen los aspectos necesarios que han hecho posible tal tarea.

Para el objetivo de implementar el reconocimiento del habla implementado en el HOAP-3 se ha conseguido parcialmente ya que por desgracia, se tiene una memoria muy escasa en el robot HOAP-3 y se ha necesitado un PC auxiliar que haga el algoritmo de reconocimiento, el post-procesado y el envío de información de las órdenes oportunas. Sin embargo, se ha implementado un sistema completo de reconocimiento del habla y síntesis de voz (algoritmo de reconocimiento, archivos reprogramables, etc.) usando el HOAP-3 que era el objetivo final del proyecto.

Finalmente en el objetivo de la demostración se ha grabado un video del robot HOAP-3 usando tanto el reconocimiento del habla como el sistema de síntesis de voz realizando las tareas contempladas en la programación satisfactoriamente.

“Voz HOAP Uc3m PFC Pablo Marín Subtitulado” [34]

6.2.- OBJETIVO FINAL DEL RECONOCIMIENTO DEL HABLA Y SÍNTESIS DE VOZ.

En el presente, ya se usan sistemas automáticos de reconocimiento del habla y síntesis de voz en el ámbito de las centralitas de telefonía (averías, pedidos, comerciales, etc.) y también en el ámbito de la medicina (para pedir citas). Estos sistemas deben ser 100% fiables, sin opción a error. Existen también robots que hablan de una manera muy precisa y muy humana (con programas bajo licencia) y que reconocen casi todo lo que se le dice en un reconocimiento automático del habla en ámbito general (lo consiguen gracias a la carga dinámica de gramáticas).

Por lo tanto el objetivo final del reconocimiento y síntesis del habla, es precisamente para poder interactuar de una manera más intuitiva y sencilla (sin necesidad de códigos que haya que aprenderse) con los aparatos, máquinas, sistemas y robots. En un futuro puede llegar a ser posible la programación de todos los aparatos mediante la voz. Este sistema de interactuar por medio de la voz también permitirá más movilidad en las tareas e incluso poder hacer varias tareas a la vez. Por ejemplo una persona que quiera activar el aparato reproductor de música con un disco específico almacenado en la memoria del aparato. Ahora es necesario ir hacia el aparato, buscar el disco y darle a reproducir. Es necesario centrarse en esa tarea exclusivamente. Con un reconocimiento de voz avanzado, simplemente dirigiéndose con la voz a este aparato, diciendo el disco que quiere que se reproduzca, podría mientras estar realizando otras tareas y sin la necesidad de ir hacia el reproductor.

Sin duda será un medio de vida para las personas discapacitadas, haciendo su vida realmente más cómoda.

7 REFERENCIAS

[1]http://www.cstr.ed.ac.uk/projects/festival/ (last visit: Nov-2011).
[2]http://cmusphinx.sourceforge.net/wiki/ (last visit: Nov-2011).
[3]http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_05_06/io3/public_html/conceptos/voz/voz_03.html (last visit: Nov-2011).
[4]http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_05_06/io3/public_html/conceptos/voz/voz_04.html (last visit: Nov-2011).
[5]http://es.wikipedia.org/wiki/S%C3%ADntesis_de_habla (last visit: Sep-2011).
[6] Administración de un call center de cobranza (2006) http://bibdigital.epn.edu.ec/handle/15000/505 (last visit: Nov-2011).
[7] An introduction to text-to-speech synthesis [Dutoit, Thierry – 1997 Kluwer Academic]
[8] http://ict.udlap.mx/people/ingrid/Clases/IS412/index.html (last visit: Nov-2011).
[9] Speech Recognition: Statistical Methods (L R Rabiner, Rutgers University, B-H Juang, Georgia Institute of Technology) – 2006
[10] Statistical methods for speech recognition [Jelinek, Frederick – 1998 The MIT Press]
[11]http://www.microsoft.com/en-us/tellme/ (last visit: Nov-2011).
[12]http://tcts.fpms.ac.be/synthesis/mbrola.html (last visit: Nov-2011).
[13]http://www.gnu.org/s/gnuspeech/ (last visit: Nov-2011).
[14]http://www.loquendo.com/es/ (last visit: Nov-2011).
[15]http://www.sodelscot.com (last visit: Nov-2011).
[16]http://cepstral.com/ (last visit: Nov-2011).
[17]http://www.naturalvoices.att.com/# (last visit: Nov-2011).
[18]http://www.acapela-group.com/index.html (last visit: Nov-2011).
[19]http://www.nextup.com/index.html (last visit: Nov-2011).
[20]http://www.verbio.com/webverbio3/html/index.php (last visit: Nov-2011).
[21]http://www.speechfxinc.com/fonixtalk61.php (last visit: Nov-2011).
[22]http://cmusphinx.sourceforge.net/ (last visit: Nov-2011).
[23]http://developer.android.com/index.html (last visit: Nov-2011).
[24]http://www.speechfxinc.com/voicein_se.php (last visit: Nov-2011).
[25]http://www.loquendo.com/es/productos/reconocimiento-de-voz/ (last visit: Nov-2011).
[26]http://nuance.com/dragon/index.htm (last visit: Nov-2011).
[27]http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_05_06/io3/public_html/procesamiento/reconocimiento/grafreconoc/sonidsonoro.gif (last visit: Nov-2011).
[28]http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_05_06/io3/public_html/procesamiento/reconocimiento/grafreconoc/sonidsordo.gif (last visit: Nov-2011).
[29]http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_05_06/io3/public_html/conceptos/voz/Vozgraf/espectro_f_u.jpg (last visit: Nov-2011).
[30]http://www.monografias.com/trabajos901/fundamentos-basicos-reconocimiento-voz/Image2576.jpg (last visit: Nov-2011).
[31]http://advancedtech.wordpress.com/2008/07/03/modelos-ocultos-de-markov-arquitectura/
[32]http://home.comcast.net/~jtechsc/ (last visit: Nov-2011).
[33] Manual HOAP-3
[34]http://www.youtube.com/watch?v=q8rNuqHW1zo (last visit: Nov-2011).

8 ANEXOS

8.1.- PRESUPUESTO

MATERIALES

Concepto	Valor por unidad (€)	I.V.A.	Coste total (€)
Equipo PC-auxiliar(1)	507.62	18.00%	599
Cascos-Micrófono	3.39	18.00%	4
Sistema Operativo(2)	0	0	0
Software(3)	0	0	0
Total			**603**

(1). Laptop Acer ASPIRE 5750G.
(2). Linux Ubuntu 10.04
(3). Festival TTS; PocketSphinx.

PERSONAL

Concepto	Nº horas	Coste por hora (€)	Coste total (€)
Empleado	720	15	10.800
Total			**10.800**

PRESUPUESTO TOTAL

Concepto	Coste total (€)
Materiales	603
Personal	10.800
Total	**11.403**

8.2.- CÓDIGO FUENTE DEL PROGRAMA DE RECONOCIMIENTO DEL HABLA

```
#include <iostream>
#include "reconContinuo.h"
#include "reconContinuo.cpp"


#include "sock.h"
#include "command.h"

int main(int argc, char *argv[]) {

    static ps_decoder_t *ps;
    static cmd_ln_t *config;

    config = cmd_ln_init(NULL, ps_args(), TRUE,
                         "-hmm", MODELDIR "/hmm/en_US/hub4wsj_sc_8k",
//                         "-lm", "/usr/share/pocketsphinx/model/hoap/hoap3.lm" ,
                         "-dict", "/usr/share/pocketsphinx/model/hoap/hoap3.dic",
                         "-jsgf","/usr/share/pocketsphinx/grammar/HoapGrammar.jsgf",
//                         "-hmm", MODELDIR "/hmm/en_US/voxforge_en_sphinx.cd_cont_5000",
//                         "-mdef", MODELDIR "hmm/en_US/voxforge_en_sphinx.cd_cont_5000/mdef",
//                         "-hmm", MODELDIR "/hmm/zh/tdt_sc_8k",
//                         "-hmm", MODELDIR "/hmm/en/tidigits",
//                         "-lm", MODELDIR "/lm/en/turtle.DMP",
//                         "-dict", MODELDIR "/lm/en/turtle.dic",

                        NULL);

    ps = ps_init(config);
    if (ps == NULL)
        return 1;
    ad_rec_t *ad;
    int16 adbuf[4096];
    int32 k, ts, rem;
    const char *hyp;
    char const *uttid;

    char *t1;
    char cadena[FILAS][50],orden[50];
    int i=0,f=0,numero,valor;

  // Sock commandServer = Sock(argv[1], "hoapcommand", "tcp");
  // command commServ = command(&commandServer);

    cont_ad_t *cont;
    /* cont_ad_t es una estructura con los siguientes parametros:
       -cont_ad_t::ad
           A/D device argument for adfunc.
       -cont_ad_t::state
           State of data returned by most recent cont_ad_read call; CONT_AD_STATE_SIL or CONT_AD_STATE_SPEECH.
       -cont_ad_t::read_ts
           Absolute timestamp (total numbers of raw samples consumed upto the most recent cont_ad_read call, starting from the very
beginning). Note that this is a 32-bit integer; applications should guard against overflow. char word[256];
       -cont_ad_t::seglen
           Total number of raw samples consumed in the segment returned by the most recent cont_ad_read call. Can be used to detect
silence segments that have stretched long enough to terminate an utterance
       -cont_ad_t::siglvl
           Max signal level for the data consumed by the most recent cont_ad_read call (dB range: 0-99).
       -cont_ad_t::sps
           Samples/sec; moved from ad->sps to break dependence on ad by N.
       -cont_ad_t::tail_state
           State at the end of its internal buffer (internal use): CONT_AD_STATE_SIL or CONT_AD_STATE_SPEECH.
       -cont_ad_t::rawfp
           If non-NULL, raw audio input data processed by cont_ad is dumped to this file.
       -cont_ad_t::logfp
           If non-NULL, write detailed logs of this object's progress to the file.
     Realmente los que se actualizan son:
     state, read_ts, seglen, siglvl. con la funcion cont_ad_read.
*/

    if ((ad = ad_open_dev(cmd_ln_str_r(config, "-adcdev"),(int)cmd_ln_float32_r(config, "-samprate"))) == NULL)
        E_FATAL("Failed top open audio device\n");


    /* Initialize continuous listening module */
    if ((cont = cont_ad_init(ad, ad_read)) == NULL)
        E_FATAL("Failed to initialize voice activity detection\n");//cont_ad_init es para configurar el nivel de umbral.
                                                   //se ha ajustado ad con el dispositivo de audio y con el samprate por defecto,
                                                   //si queremos modificarlo hay que cambiar el samprate.
```

```
/*
        Parameters:
           ad              In: The A/D source object to be filtered
           adfunc  In: adfunc = source function to be invoked to obtain raw A/D data. See ad.h for the required prototype definition.
           ad_read es una callback */
  if (ad_start_rec(ad) < 0)
     E_FATAL("Failed to start recording\n");
  if (cont_ad_calib(cont) < 0)//calibramos el umbral con los parametros anteriores
     E_FATAL("Failed to calibrate voice activity detection\n");

  for (;;) {
     /* Indicate listening for next utterance */
     printf("READY....\n");
     fflush(stdout);
     fflush(stderr);

     /* Wait data for next utterance */
     while ((k = cont_ad_read(cont, adbuf, 4096)) == 0)
        sleep_msec(100);
/*  Despues de esta funcion, ya tenemos en adbuf una ristra de datos digitales que son los procesados por
    A/D (conversor analogico-digital) para poder procesarlos.
    Las siguientes variables se actualizan con la llamada:
    cont_ad_t.state, cont_ad_t.read_ts, cont_ad_t.seglen, cont_ad_t.siglvl.
        K = nº de muestras leidas.*/

     if (k < 0)
        E_FATAL("Failed to read audio\n");

     /*
      * Non-zero amount of data received; start recognition of new utterance.
      * NULL argument to uttproc_begin_utt => automatic generation of utterance-id.
      */
     if (ps_start_utt(ps, NULL) < 0)
        E_FATAL("Failed to start utterance\n");
     ps_process_raw(ps, adbuf, k, FALSE, FALSE); //adbuf es el buffer dnd esta el archivo raw que contiene lo que queremos
decodificar.
     printf("Listening...\n");
     fflush(stdout);

     /* Note timestamp for this first block of data */
     ts = cont->read_ts;

     /* Decode utterance until end (marked by a "long" silence, >1sec) */
     for (;;) {
        /* Read non-silence audio data, if any, from continuous listening module */
        if ((k = cont_ad_read(cont, adbuf, 4096)) < 0) //se actualiza cont->read_ts con el nuevo valor y luego se compara
           E_FATAL("Failed to read audio\n");
        if (k == 0) {
           /*
            * No speech data available; check current timestamp with most recent
            * speech to see if more than 1 sec elapsed.  If so, end of utterance.
            */
           if ((cont->read_ts - ts) > DEFAULT_SAMPLES_PER_SEC) // samples per sec = 16000
              break;
        }
        /*Esta funcion compara el valor actual de timestamp guardado en cont->read_ts con el valor 16.000 que son las muestras que
tiene 1 segundo.
           Si queremos que deje de grabar a los 5 segundos x ejemplo tendriamos q compararlo con el valor 80.000
           lo hace de la siguiente manera: Si seguimos hablando mientras esta grabando, el timestamp valdrá lo que valía antes mas lo
que ha decodificado ahora
           por tanto ese valor va en aumento. Cuando dejamos de hablar, read_ts actual sigue aumentando pero el timestamp anterior
no se actualiza por tanto la diferencia
           entre ambos será mayor. si pasa de los 16.000 es que hemos dejado de hablar durante 1 seg y sale del bucle.*/


        else {
           /* New speech data received; note current timestamp */
           ts = cont->read_ts;//actualizamos el timestamp
        }

        /*
         * Decode whatever data was read above.
         */
        rem = ps_process_raw(ps, adbuf, k, FALSE, FALSE);//decodifica lo que tenga el buffer de datos digitales llegados desde el
conversor A/D

        /* If no work to be done, sleep a bit */
        if ((rem == 0) && (k == 0))
           sleep_msec(20);
     }

     /*
      * Utterance ended; flush any accumulated, unprocessed A/D data and stop
      * listening until current utterance completely decoded
```

```
        */
        ad_stop_rec(ad);
        while (ad_read(ad, adbuf, 4096) >= 0);
                                //espera hasta que ad_read devuelva un valor mayor o igual a 0.
                                //ad_read: source function to be invoked to obtain raw A/D data
        cont_ad_reset(cont); //resetea los parametros de cont

        printf("Stopped listening, please wait...\n");
        fflush(stdout);
        /* Finish decoding, obtain and print result */
        ps_end_utt(ps);
        hyp = ps_get_hyp(ps, NULL, &uttid); // uttid: numero de veces que se ha procesado el audio hyp:
                                    // cadena de caracteres con las posibles coincidencias
        printf("%s: %s\n", uttid, hyp);
        fflush(stdout);
/*****************************************************************************************************
/*                      Ya tenemos la cadena de texto para procesar.
/*      Separamos las palabras de la cadena en un array para trabajar más cómodos
*****************************************************************************************************/

        for(f=0;f<i;f++)
        {
            sprintf(cadena[f],"%s","\0");
        }
        i=0;
        for ( t1 = strtok((char*)hyp," ");t1 != NULL;t1 = strtok(NULL, " ") )
        {
            printf("t1:%s\n",t1);
            sscanf(t1, "%s", cadena[i]);
            i++;
        }
        for (f=0; f<i; f++)
        {
            printf("cadena:%s\n", cadena[f]);
        }
/****************** Buscamos la cadena de numeros para traducirla   ******************
*****************************************************************************************/
        numero=0;
        for(f=0;f<i;f++)
        {
            valor=delunoalnueve(cadena[f]);
            if(valor != 0)
            {
                numero=valor;
                break;
            }
            else if(strcmp(cadena[f],"ten") == 0){
                numero=10;
                break;
            }
            else if(strcmp(cadena[f],"eleven") == 0){
                numero=11;
                break;
            }
            else if(strcmp(cadena[f],"twelve") == 0){
                numero=12;
                break;
            }
            else if(strcmp(cadena[f],"fourteen") == 0){
                numero=14;
                break;
            }
            else if(strcmp(cadena[f],"fifteen") == 0){
                numero=15;
                break;
            }
            else if(strcmp(cadena[f],"sixteen") == 0){
                numero=16;
                break;
            }
            else if(strcmp(cadena[f],"seventeen") == 0){
                numero=17;
                break;
            }
            else if(strcmp(cadena[f],"eighteen") == 0){
                numero=18;
                break;
            }
            else if(strcmp(cadena[f],"nineteen") == 0){
                numero=19;
                break;
            }
            else if(strcmp(cadena[f],"twenty") == 0){
                valor=delunoalnueve(cadena[f+1]);
```

```
    if (valor != 0)
    {
        numero=20+valor;
        break;
    }
    else
    {
        numero=20;
        break;
    }
}
else if(strcmp(cadena[f],"thirty") == 0){
    valor=delunoalnueve(cadena[f+1]);
    if (valor != 0)
    {
        numero=30+valor;
        break;
    }
    else
    {
        numero=30;
        break;
    }
}
else if(strcmp(cadena[f],"forty") == 0){
    valor=delunoalnueve(cadena[f+1]);
    if (valor != 0){
        numero=40+valor;
        break;
    }
    else
    {
        numero=40;
        break;
    }
}
else if(strcmp(cadena[f],"fifty") == 0){
    valor=delunoalnueve(cadena[f+1]);
    if (valor != 0){
        numero=50+valor;
        break;
    }
    else
    {
        numero=50;
        break;
    }
}
else if(strcmp(cadena[f],"sixty") == 0){
    valor=delunoalnueve(cadena[f+1]);
    if (valor != 0){
        numero=60+valor;
        break;
    }
    else
    {
        numero=60;
        break;
    }
}
else if(strcmp(cadena[f],"seventy") == 0){
    valor=delunoalnueve(cadena[f+1]);
    if (valor != 0){
        numero=70+valor;
        break;
    }
    else
    {
        numero=70;
        break;
    }
}
else if(strcmp(cadena[f],"eighty") == 0){
    valor=delunoalnueve(cadena[f+1]);
    if (valor != 0){
        numero=80+valor;
        break;
    }
    else
    {
        numero=80;
        break;
    }
}
```

```
        else if(strcmp(cadena[f],"ninety") == 0){
            valor=delunoalnueve(cadena[f+1]);
            if (valor != 0){
                numero=90+valor;
                break;
            }
            else
            {
                numero=90;
                break;
            }
        }
    }
    printf("numero:%i\n",numero);
/******************** Ya tenemos el numero concreto ************************/

    if (strcmp(cadena[0], "robot") == 0 || strcmp(cadena[0], "HOAP") == 0 || strcmp(cadena[0], "HOAP3") == 0)
    {
        if (strcmp(cadena[1], "goodbye") == 0 || strcmp(cadena[1], "bye") == 0)
        {
            printf("I have recognized the word goodbye :-)\n");
            system("festival -b goodbye.scm");
            commServ.sendCommand("BYE\n");
        }else if (strcmp(cadena[1], "hello") == 0 || strcmp(cadena[1], "hi") == 0 || strcmp(cadena[1], "greetings") == 0)
        {
            printf("I have recognized the word hello :-)\n");
            system("festival -b hello.scm");
            commServ.sendCommand("HELLO\n");
        }else
/************************** Concatenamos cadenas para las unidades ************************/
/*                En numero en teoria nos encontramos las unidades                      */
/****************************** Pero lo comprobamos ***************************************/
        if (strcmp(cadena[1], "move") == 0)
        {
            if(strcmp(cadena[2], "turning") == 0)
            {
                if(strcmp(cadena[3], "left") == 0)
                {
                    if(numero!=0){
                        printf("I have recognized MOVE TURNING LEFT\n");
                        sprintf(orden, "%s %i %s","MOVE TURNING LEFT", numero,"DEGREES\n");
                        printf("Esta es la orden: %s",orden);
                        system("festival -b moveTurningLeft.scm");
                        commServ.sendCommand(orden);
                    }else
                    {
                        printf("You don't tell me number!\n");
                        system("festival -b nonumbers.scm");
                        commServ.sendCommand("NOTNUMBERS");
                    }

                }else if(strcmp(cadena[3], "right") == 0)
                {
                    if(numero!=0)
                    {
                        printf("I have recognized MOVE TURNING RIGHT\n");
                        sprintf(orden, "%s %i %s","MOVE TURNING RIGHT", numero,"DEGREES\n");
                        printf("Esta es la orden: %s",orden);
                        system("festival -b moveTurningRight.scm");
                        commServ.sendCommand(orden);
                    }else
                    {
                        printf("You don't tell me number!\n");
                        system("festival -b nonumbers.scm");
                        commServ.sendCommand("NOTNUMBERS");
                    }
                }else
                {
                    printf("I don't have recognized anything\n");
                    system("festival -b notUnderstand.scm");
                    commServ.sendCommand("NOTUNDERSTAND");
                }
            }
            else if(strcmp(cadena[2], "walking") == 0)
            {
                if (strcmp(cadena[3], "forward") == 0)
                {
                    if(numero!=0)
                    {
                        printf("I have recognized MOVE WALKING FORWARD\n");
                        sprintf(orden, "%s %i %s","MOVE WALKING FORWARD", numero,"STEPS\n");
                        printf("Esta es la orden: %s",orden);
                        system("festival -b moveWalkingForward.scm");
                        commServ.sendCommand(orden);
```

```
            }else
            {
                printf("You don't tell me number!\n");
                system("festival -b nonumbers.scm");
                commServ.sendCommand("NOTNUMBERS");
            }
        }else if(strcmp(cadena[3], "backward") == 0)
        {
            if(numero!=0)
            {
                printf("I have recognized MOVE WALKING BACKWARD\n");
                sprintf(orden, "%s %i %s","MOVE WALKING BACKWARD", numero,"STEPS\n");
                printf("Esta es la orden: %s",orden);
                system("festival -b moveWalkingBackward.scm");
                commServ.sendCommand(orden);
            }else
            {
                printf("You d¡dn't tell me number!\n");
                system("festival -b nonumbers.scm");
                commServ.sendCommand("NOTNUMBERS");
            }
        }
    }
    else if(strcmp(cadena[2], "head") == 0)
    {
        if (strcmp(cadena[3], "left") == 0)
        {
            if(numero!=0)
            {
                printf("I have recognized MOVE HEAD LEFT\n");
                sprintf(orden, "%s %i %s","MOVE HEAD LEFT", numero,"DEGREES\n");
                printf("Esta es la orden: %s",orden);
                system("festival -b moveHeadLeft.scm");
                commServ.sendCommand(orden);
            }else
            {
                printf("You d¡dn't tell me number!\n");
                system("festival -b nonumbers.scm");
                commServ.sendCommand("NOTNUMBERS");
            }
        }else if(strcmp(cadena[3], "right") == 0)
        {
            if(numero!=0)
            {
                printf("I have recognized MOVE HEAD RIGHT\n");
                sprintf(orden,"MOVE HEAD RIGHT");
                sprintf(orden, "%s %i %s","MOVE HEAD RIGHT", numero,"DEGREES\n");
                printf("Esta es la orden: %s",orden);
                system("festival -b moveHeadRight.scm");
                commServ.sendCommand(orden);
            }else
            {
                printf("You d¡dn't tell me units!\n");
                system("festival -b nonumbers.scm");
                commServ.sendCommand("NOTNUMBERS");
            }
        }else if (strcmp(cadena[3], "up") == 0)
        {
            if(numero!=0)
            {
                printf("I have recognized MOVE HEAD UP\n");
                sprintf(orden, "%s %i %s","MOVE HEAD UP", numero,"DEGREES\n");
                printf("Esta es la orden: %s",orden);
                system("festival -b moveHeadUp.scm");
                commServ.sendCommand(orden);
            }else
            {
                printf("You d¡dn't tell me number!\n");
                system("festival -b nonumbers.scm");
                commServ.sendCommand("NOTNUMBERS");
            }
        }else if(strcmp(cadena[3], "down") == 0)
        {
            if(numero!=0)
            {
                printf("I have recognized MOVE HEAD DOWN\n");
                sprintf(orden, "%s %i %s","MOVE HEAD DOWN", numero,"DEGREES\n");
                printf("Esta es la orden: %s",orden);
                system("festival -b moveHeadDown.scm");
                commServ.sendCommand(orden);
            }else
            {
                printf("You d¡dn't tell me number!\n");
                system("festival -b nonumbers.scm");
```

```
                    commServ.sendCommand("NOTNUMBERS");
                }
            }else
            {
                printf("I don't have recognized anything\n");
                system("festival -b notUnderstand.scm");
                commServ.sendCommand("NOTUNDERSTAND");
            }
        }else
        {
            printf("I don't have recognized anything\n");
            system("festival -b notUnderstand.scm");
            commServ.sendCommand("NOTUNDERSTAND");
        }
    }else
    {
        printf("I don't have recognized anything\n");
        system("festival -b notUnderstand.scm");
        commServ.sendCommand("NOTUNDERSTAND");
    }
}else
{
    printf("You don't speak with me\n");
    //system("festival -b other.scm");
    commServ.sendCommand("OTHER");
}
/* Resume A/D recording for next utterance */
if (ad_start_rec(ad) < 0)
    E_FATAL("Failed to start recording\n");
if(strcmp(cadena[0], "goodbye") == 0)
    break;
}

cont_ad_close(cont);
ad_close(ad);
ps_free(ps);
return 0;
}


int delunoalnueve(char *cadena)
{
if(strcmp(cadena,"one") == 0)
    return 1;
else if(strcmp(cadena,"two") == 0)
    return 2;
else if(strcmp(cadena,"three") == 0)
    return 3;
else if(strcmp(cadena,"four") == 0)
    return 4;
else if(strcmp(cadena,"five") == 0)
    return 5;
else if(strcmp(cadena,"six") == 0)
    return 6;
else if(strcmp(cadena,"seven") == 0)
    return 7;
else if(strcmp(cadena,"eight") == 0)
    return 8;
else if(strcmp(cadena,"nine") == 0)
    return 9;
else
    return 0;
}
```

8.3.- ARCHIVO FSG (FINITE STATE GRAMMAR)

```
#JSGF V1.0;
	grammar gram;
	public <gram> = [<rob>] <order> | [<rob>] <order> <location> <side> <quantity>;

<rob> = robot
;

<order> =  move
                        | hello
                        | hi
                        | goodbye
                        | bye
;

<location> = walking
                        | head
                        | turning
;
<side> = left
                        | right
                        | up
                        | down
                        | forward
                        | backward
;
<quantity> = <digit>| <below20>| <tens> [<digit>];

<digit> = one
                        | two
                        | three
                        | four
                        | five
                        | six
                        | seven
                        | eight
                        | nine
;
<below20> = ten
                        | eleven
                        | twelve
                        | thirteen
                        | fourteen
                        | fifteen
                        | sixteen
                        | seventeen
                        | eighteen
                        | nineteen
;
<tens> = twenty
                        | thirty
                        | forty
                        | fifty
                        | sixty
                        | seventy
                        | eighty
                        | ninety
;
<unit> =  step
                        | steps
                        | degrees
;
```