Inteligencia Artificial para Sistemas Colaborativos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Noël Ruault Martínez

Tutor/es:
Francisco Javier Ferrández Pastor

Septiembre 2016

Universitat d'Alacant
Universidad de Alicante

ÍNDICE

1. INTRODUCCIÓN

En primer lugar vamos a proceder a presentar una introducción al trabajo realizado, así como la motivación para comenzar el proyecto y los objetivos que se plantearon con la finalidad de dar vida a una Inteligencia Artificial.

1.1. Motivación

En los tiempos que corren la robótica tiene cada vez más importancia y repercusión en el día a día, pero se desconoce una gran parte de la misma, y no se aprovecha el inimaginable potencial que esta tiene debido a razones tanto económicas como de desconocimiento del tema.

Una de las principales razones por las que nace mi interés en el mundo de la robótica y así pues la idea de realizar este proyecto, es la oportunidad que tuve de trabajar con unos de los pioneros en robótica social: Aisoy Robotics.

Durante un año estuve aprendiendo y familiarizándome con todo lo relacionado con la Inteligencia Artificial y la actuación de ésta de un modo autónomo, para así satisfacer las necesidades de un usuario. Esto provocó que quisiera dar forma a una Inteligencia Artificial con la que fuera posible resolver problemas.

1.2. Objetivos

1.2.1. Objetivo general

El objetivo de este proyecto es la programación de una IA con capacidad de interacción con el usuario, es decir, capaz de memorizar y decidir qué hacer en función de un histórico dado por el usuario.

1.2.2. Objetivos específicos

Mediante este trabajo tratamos de llevar a cabo la Programación de una IA. En este caso se ha utilizado una Raspberry Pi2, con las siguientes funcionalidades:

- Utilización del lenguaje Python con el framework ROS para la programación de ésta
- Implementación de un SDK para el código, para agilizar el trabajo.
- Código limpio y buenas praxis, código eficiente.
- Implementación del sistema inteligente Nuance para interactuar con el robot.

1.3. Historia de los robots

El hombre a lo largo de la historia ha desarrollado máquinas y herramientas que permitieran simplificar y facilitar su trabajo, así como mejorar sus condiciones de vida. Actualmente el robot es esa máquina que se encuentra en contínuo desarrollo para ayudar a agilizar nuestras tareas, llegando a ser imprescindible actualmente en nuestra sociedad, ya que se encuentra en cualquier sector de la misma.

Como robot se entiende una maquina electromecánica la cual ha sido programada para moverse, manipular objetos y realizar trabajos a la vez que interacciona con su entorno.

El inicio de la robótica actual puede fijarse en la industria textil del siglo XVIII, cuando Joseph Jacquard inventó en 1801 una máquina textil programable mediante tarjetas perforadas. Con la llegada de la revolución industrial se intentó el desarrollo de máquinas que ayudasen o sustituyesen el hombre en las tareas que éste tenía que desarrollar. Del mismo modo los autómatas, o máquinas semejantes a personas, ya aparecían en los relojes de las iglesias medievales, y los relojeros del siglo XVIII eran famosos por sus ingeniosas criaturas mecánicas.

La primera vez que se utilizó la palabra robot fue en 1920 en una obra llamada ”Los Robots Universales de Rossum”, escrita por el dramaturgo checo Karel Capek. Su trama trataba sobre un hombre que fabricó un robot y luego este último mata al hombre. La palabra checa “Robota” significa servidumbre o trabajo forzado y cuando se tradujo al inglés se convirtió en el termino robot.

Poco después Isaac Asimov comenzó a contribuir con diversas obras. En una de ellas acuñó un término que se utilizaría a lo largo de los años: este termino era Robótica, el cual se refiere a la ciencia o arte relacionada con la inteligencia artificial y con la ingeniería mecánica. También a partir de este término surgen las denominadas ”Tres Leyes de la Robótica”:

- Un robot no puede actuar contra un ser humano o, mediante la inacción, que un ser humano sufra daños.
- Un robot debe de obedecer las órdenes dadas por los seres humanos, salvo que estén en conflictos con la primera ley.
- Un robot debe proteger su propia existencia, a no ser que esté en conflicto con las dos primeras leyes.

Varios factores influyeron en el desarrollo de los primeros robots en la década de los cincuenta. En primer lugar, el desarrollo en la tecnología, donde se incluyen los procesadores electrónicos, los actuadores de control retroalimentados y la transmisión de potencia a través de engranajes han contribuido a flexibilizar los autómatas para realizar las tareas programadas. Además, la investigación en inteligencia artificial desarrolló diferentes maneras de simular el procesamiento de información humana con ordenadores. Las primeras patentes aparecieron en 1946 con los muy primitivos robots para traslado de maquinaria de Devol. En 1954, Devol diseña el primer robot programable. Durante los siguientes años se desarrollan una gran variedad de robots en todos los sectores de la sociedad, principalmente en el ámbito industrial para agilizar las tareas industriales.

En los setenta, la NASA inicio un programa de cooperación con el *Jet Propulsión Laboratory* para desarrollar plataformas capaces de explorar terrenos hostiles.

Actualmente, el concepto de robótica ha evolucionado hacia los sistemas móviles autónomos, que son aquellos capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión.

En general la historia de la robótica se puede clasificar en cinco generaciones: las dos primeras, ya alcanzadas en los ochenta, incluían la gestión de tareas repetitivas con autonomía muy limitada. La tercera generación incluiría visión artificial, en lo cual se ha avanzado mucho en los ochenta y noventa. La cuarta incluye movilidad avanzada en exteriores e interiores y la quinta entraría en el dominio de la inteligencia artificial en lo cual se esta trabajando actualmente.

1.4. Estado del arte.

1.4.1. Smart cities

Hoy en día, las principales ciudades del mundo luchan por ser espacios más tecnológicos, verdes y transitables.

La filosofía de las Smart Cities reside en aunar, mediante una adecuada planificación, todos estos conceptos con la finalidad de convertir las ciudades en espacios sostenibles, innovadores y eficientes, en los que el ciudadano debe ser el eje del cambio y el principal beneficiado del nuevo paradigma urbano.

El propósito final de una Smart City es alcanzar una gestión eficiente en todas las áreas de la ciudad (urbanismo, infraestructuras, transporte, servicios, educación, sanidad, seguridad pública, energía, etc), satisfaciendo a la vez las necesidades de la urbe y de sus ciudadanos, aplicando una gestión eficiente a todas las escalas y **tomando la innovación tecnológica como un importante punto de apoyo.**

1.4.2. El robot "Asimo"

Fabricado por Honda, se trata del robot humanoide más desarrollado del mercado.

Se acerca a su objetivo de convertirse en un autómata para el uso práctico en hogares o allí donde se requiera asistencia. Es capaz de expresarse en lenguaje de señas y predecir hacia dónde se dirige una persona y de este modo evitar chocarse con ella cambiando su propia trayectoria de movimiento.

Honda espera que Asimo sea capaz de ayudar a personas con necesidades especiales, gracias, entre otras cosas, a sus nuevas capacidades, que incluyen un nivel de inteligencia mejorado y una mayor destreza en las manos, lo que le permite abrir una botella y servir una bebida, así como expresarse con el lenguaje de signos.

Además, puede subir y bajar escaleras de una manera fluida, o saltar incluso de forma continua con una sola pierna.

El nuevo Asimo cuenta con una serie de capacidades que son el fruto del empeño de Honda en crear robots con habilidades para tomar decisiones, y asistir a los humanos de manera que puedan adaptar su comportamiento al entorno en tiempo real de la manera más humana posible.

Figura 1.1: Asimo.

1.4.3. El robot “Aibo”

Albo es un robot de inteligencia artificial creado por Sony. Este “perro robot” es capaz de desarrollar su propia personalidad, en función del entorno en el que se **encuentre.**

La integración de Aibo en los hogares, ha sido tan positiva que incluso sus usuarios llegaron a confundir el mecanismo de un dispositivo programado para simular el comportamiento animal, con la naturaleza humana.

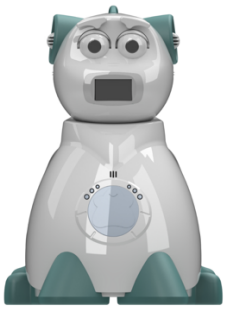
Figura 1.2: Aibo.

1.4.4. El robot “Aisoy1”

Aisoy1 es un robot social, cognitivo y emocional que usa inteligencia artificial. Es capaz de establecer vínculos emocionales con los usuarios e interactuar con el mundo real (físico y digital así como con otros dispositivos electrónicos).

Aisoy es calificado como uno de los primeros robots emocionales para el mercado de consumo y fue creado por la empresa española Aisoy Robotics.

El robot cuenta con un software que le permite interpretar estímulos de su red de sensores para aprender de ellos y tomar decisiones en función de criterios tanto lógicos como emocionales.



Aisoy puede ser programado para hablar, pensar, ver, sentir cuando es tocado y moverse usando herramientas de programación visuales, sencillas e intuitivas.

Figura 1.3: Aisoy1.

1.4.5. El robot “Nao”

Nao resulta de la creación de la compañía Aldebaran Robotics.

Se trata de un robot humanoide con fines educativos y de investigación que se relaciona con el medio a través de sus múltiples sensores y permite ser programado desde muy temprana edad (educación primaria), por lo que está indicado en el ámbito escolar. Asimismo personifica una plataforma evolutiva que se adapta al nivel del usuario (desde los 5 años hasta esferas universitarias y de investigación).

Este robot es capaz de caminar sin chocarse, hablar y mostrar emociones de igual manera que los humanos.

A lo largo de la trayectoria de NAO, ha sido destinado a fines de investigación tales como el autismo, y en ocasiones ha sido considerado una máquina incluso más sociable que el propio ser humano.

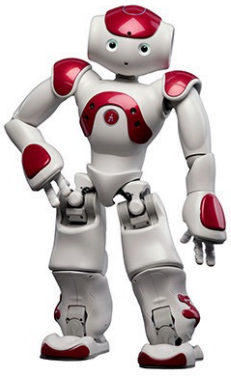


Figura 1.4: Nao.

Todas las tecnologías descritas, hayan sido programadas en forma de humanoide, en forma canina o concretamente en el caso de las Smart Cities sin forma, son capaces de recibir información, procesarla y devolver un resultado que casi en el cien por cien de los casos es una solución muy valiosa para un humano y en la mayor parte de los casos inalcanzable sin este tipo de tecnologías.

Este tipo de mecanismos inteligentes son los que, de una manera u otra, despertaron mi curiosidad y a día de hoy me siento estrechamente ligado a todo el mundo de la Inteligencia Artificial.

2. ELECTRÓNICA Y COMPONENTES HARDWARE

2.1. Raspberry Pi

Raspberry Pi es un pequeño ordenador de tamaño reducido que contiene un procesador central (con capacidad de overclock), un procesador gráfico y memoria RAM. Utiliza una tarjeta SD para el almacenamiento permanente.

Podemos conectar fácilmente la Raspberry Pi a una televisión vía HDMI o RCA y utilizarla con un teclado y un ratón. Tiene conexión a Internet y unos pines GPIO (General Purpose Input/Output), para que podamos interactuar con nuestra placa con sensores, botones, o cualquier dispositivo externo compatible. Por otra parte, reproduce vídeo en alta definición y también sirve como procesador de textos o para jugar, pero realmente está pensada para ser programada.

La Raspberry Pi fue diseñada por la Fundación Raspberry Pi de la Universidad de Cambridge en Reino Unido con el objetivo de estimular el aprendizaje de ciencias de la computación y programación en las escuelas, siendo un proyecto de hardware libre. Desde su lanzamiento la Raspberry Pi se volvió muy popular entre las comunidades de programadores y hardware hackers ya que se pueden hacer infinidad de cosas con este pequeño ordenador.
Este pequeño ordenador será el que gestione toda la actividad del robot, interactuando con el usuario y procesando las señales procedentes de los diferentes sensores. Es el ordenador de placa simple que ha sido utilizado y reprogramado para este proyecto dada la facilidad y no menos importante, la experiencia que he adquirido con estos dispositivos en mi paso por Aisoy Robotics, concretamente con la distribución Raspbian.

2.2. Micrófono

Un micrófono es un dispositivo el cual capta los diferentes sonidos que se producen y transforma las ondas sonoras en energía eléctrica.
Se trata de una herramienta básica para poder tener una interacción más real con un conjunto de dispositivos externos conectados a una Raspberry Pi o en este caso concreto con una IA, pudiendo “escuchar”, ya que procesará los sonidos y los convertirá en texto con un módulo ASR (Automatic Speech Recognition), lo que permitirá tener la posibilidad de tener una dialogo con el robot.

2.3. Altavoz

Un altavoz es un dispositivo que realiza el proceso contrario a un micrófono, es decir, es un transductor electroacústico utilizado para la reproducción de sonido. Es por tanto la puerta por donde sale el sonido al exterior desde los aparatos que posibilitaron su amplificación, su transmisión por medios radioeléctricos, o su tratamiento.

Se trata de un elemento complementario al micrófono, ya que con ambos funcionando podremos comunicarnos con la IA , esta puede "hablar" y responder a lo escuchado por el micrófono. Un módulo básico es el TTS (Text To Speech), que convierte el texto en palabras, las cuales convertiremos en sonido, gracias al sistema inteligente Nuance.

3. SOFTWARE.

En este apartado se explicará el sistema operativo que utiliza el bot, el lenguaje en el que se ha programado la mayor parte del contenido lógico de la aplicación concreta (Python) así como el framework con el que se gestionan las distintas funcionalidades del mismo: ROS (Robot Operating System), nos acercaremos a la definición y posibilidades que nos brinda el lenguaje AIML, junto a una plataforma Web que nos permite gestionarlo (Pandorabots) así como al sistema inteligente de reconocimiento y traducción de texto a sonido y viceversa, Nuance.

3.1. Raspbian.

Raspbian es un sistema operativo libre y gratuito basado en Debian y optimizado para el hardware de la Raspberry Pi. Contiene unos 35 mil paquetes, precompilados, de forma tal que es fácil instalar el que necesitemos en la Raspberry Pi. El punto más fuerte de Raspbian es que hay una comunidad inmensa a las espaldas, la cantidad de información sobre tutoriales o solución de problemas es la razón por la cual es tan dinámico trabajar con esta distribución. Mi experiencia en el ámbito laboral con este sistema operativo, en adición a todo lo anterior es la razón por la cual he desarrollado el proyecto basado en Raspbian.

3.2.ROS

3.2.1. El sistema de archivos de ROS esta compuesto por:

I. **Paquetes** (packages): Paquetes: Los paquetes son la unidad de organización de software de código de ROS. Cada paquete puede contener bibliotecas, archivos ejecutables, scripts u otros artefactos.
II. **Manifiesto** (manifest.xml): Un manifiesto es una descripción de un paquete. Sirve para definir dependencias entre los paquetes y para capturar la información de metadatos acerca del paquete como la versión, "maintainer", licencia, etc...
III. **Pilas** (stacks): Conjuntos de paquetes que forman una librería. Una pila agrupa los paquete que están relacionados entre sí.

3.2.2. Componentes del sistema

El software en ROS está organizado en paquetes.

Un paquete puede contener un nodo, una librería, conjunto de datos, o cualquier cosa que pueda constituir un módulo. Los paquetes pueden organizarse en pilas (stacks).

ROS dispone de una gran variedad de herramientas que hacen más sencilla la comunicación entre los diferentes componentes del sistema. El detalle de los componentes es el siguiente:

3.2.2.1.Nodos

Los nodos son programas ejecutables que realizan funciones concretas. La principal ventaja de la utilización de nodos es su diseño modular, esto es un proceso que realiza algún tipo de computación en el sistema. De este modo un nodo puede controlar un sensor láser, otro los motores de un robot y otro la construcción de mapas.

Los nodos son compilados individualmente unos con respecto a otros, ejecutados y gestionados por un nodo principal (ROS Master). Se comunican entre sí mediante el uso de mensajes y "*topics"* con el fin de lograr un objetivo en común. Al ser un diseño modular y ser cada nodo independiente nos permite afrontar los proyectos de una manera mucho más sencilla.

Cada nodo está escrito usando la librería cliente de ROS. Esta librería es una colección de código que facilita el trabajo a los programadores en ROS, y mediante su uso podemos no solo crear los nodos sino también los publicadores y subscriptores de los mensajes y

"topics" en los nodos, además de los servicios y sus parámetros. Podemos utilizar esta librería tanto en C++ como en Python, para este caso concreto se ha utilizado Python en todos los casos.

La herramienta **rosnode** es una herramienta de línea de comando con la cual podemos mostrar información sobre los nodos, listar los nodos que se están ejecutando o activar/desactivar nodos.

3.2.2.2.Topics

Los nodos necesitan un medio con el que comunicarse. A través de mensajes, unos nodos interactúan con otros y, cuando es necesario, cualquiera de los anteriores con el nodo principal (ROS Master). Cada uno de estos mensajes se denomina "topic" y tiene una arquitectura de comunicación estricta, que se detalla a continuación:

a. Un nodo publica un mensaje en una dirección concreta
b. Varios nodos se subscriben a dicho mensaje, en la estipulada "dirección", y de esta manera reciben la información.

Normalmente las direcciones no son tan ambiguas como parece. A la hora de programar un nodo que publique mensajes se suele utilizar como nombre de la dirección el nombre del nodo que está publicando, y de este modo se evitan malentendidos.
El topic en ROS puede transmitirse usando TCP/IP (TCPROS) o UDP (UDP ROS).
ROS tiene una herramienta para trabajar con tópicos llamada **rostopic**. Es una herramienta de línea de comandos que proporciona información sobre los topics activos

3.2.2.3.Mensajes

Los nodos se comunican entre ellos mediante el paso de mensajes (publicando mensajes en un topic).

Un mensaje es una estructura simple de envío o recepción de datos que puede incluir tipos de datos: primitivos (enteros, flotantes, bool, etc.) y arrays (nuevamente, de tipo primitivo).

3.2.2.4. Servicios

Cuando necesitas comunicarte con nodos y recibir una respuesta no se puede realizar con topics, se debe hacer con servicios.

Los servicios son transacciones síncronas entre nodos que usan la arquitectura servicio/cliente "one-to-one" con requerimiento de respuesta. Las funcionalidades de un servicio pueden ser controladas con el comando **rosservice**, el cual permite entre otras cosas: arrancar un servicio inactivo, transportar datos desde una IA hacia un receptor remoto, listar servicios…

3.2.2.5. ROS Master

El ROS Master es el nodo "maestro" que funciona como núcleo del sistema, haciendo posible la comunicación individual de los diferentes nodos. A su vez, el ROS Master proporciona servicios de nombre y registro al resto de nodos del sistema ROS y realiza el seguimiento de "publishers" y "subscribers" hacia los topics y/o servicios.
Para hacerlo funcionar se utiliza la herramienta **roscore**.

3.3. Nuance

Nuance Communications es una multinacional estadounidense de tecnología software, con sede en Burlington, Massachusetts, Estados Unidos, cuya actividad se centra en el desarrollo de aplicaciones de escáner y voz. Su producción comercial se centra en software de reconocimiento de voz entre otros.

El reconocedor de voz e intérprete Nuance se encuentra en la décima edición, siendo en este momento bajo mi punto de vista el software de reconocimiento de voz con mayor precisión del mercado, con una interpretación de texto a voz increíblemente nítida y humana.
Como caso de éxito, cabe destacar que el motor de reconocimiento de voz de Apple "*Siri"* es proporcionado por Nuance Communications.

Nuance consta de dos partes:

I. ASR

El Automatic Speech Recognition (ASR) o reconocimiento automático del habla es una disciplina de la inteligencia artificial que tiene como objetivo permitir la comunicación hablada entre seres humanos y computadoras.

El problema que se plantea en un sistema de este tipo es el de hacer cooperar un conjunto de informaciones que provienen de diversas fuentes de conocimiento (acústica, fonética, fonológica, léxica, sintáctica, semántica y pragmática), en presencia de ambigüedades, incertidumbres y errores inevitables para llegar a obtener una interpretación aceptable del mensaje acústico recibido.

Este sistema de reconocimiento de voz es una herramienta computacional capaz de procesar la señal de voz emitida por el ser humano y reconocer la información contenida en ésta, convirtiéndola en texto.

II. TTS

El sistema text-to-speech (TTS) o texto a voz. La síntesis de habla es la producción artificial del habla. El sistema computarizado que es usado con este propósito es llamado sintetizador de voz y puede ser implementado en productos software o hardware. Éste convierte el lenguaje de texto normal en habla, en este caso concreto el uso del mismo ha sido crucial en la traducción o interpretación de las decisiones tomadas por el *bot*, de forma que sea capaz de externalizar aquello que quiere decir en forma de voz, logrando así una comunicación bidireccional muchísimo más humana con el usuario.

3.4. AIML

Introducción

El lenguaje de programación AIML fue desarrollado por el Dr. Richard Wallace y la comunidad de código abierto Alicebot entre los años 1995 y 2000.

El AIML, o Artificial Intelligence Mark-up Language es un lenguaje de programación basado en XML. Fue diseñado específicamente para ayudar en la creación de la primera entidad chatbot informática de lenguaje artificial online o A.L.I.C.E., en sus siglas en inglés de Artificial Linguistic Internet Computer Entity Chatterbot, (en inglés) Alice). Aunque descrito muy ampliamente, el lenguaje AIML está especializado en la creación de agentes software con lenguaje natural, conocidos como Alicebots.
Personalmente, me he ayudado de muchos de estos Alicebots en Inglés para conseguir comprender el funcionamiento de la mente humana, trasladado a un entorno lógico. Pero en este caso concreto, el cambio de idioma ha sido muy laborioso, dado que la manera de comunicarnos con un bot, varía mucho del Inglés al Español.

Cómo funciona AIML

III. **La etiqueta *<category>*** es la unidad básica del conocimiento de AIML. Una categoría siempre contiene una input "*pattern"* y una respuesta "*template"*.
IV. **La etiqueta *<pattern>*** busca aquello que el usuario dice, es decir, procesa la input del usuario y busca cuál es la respuesta que corresponde a dicha entrada. AIML no hace distinción entre mayúsculas y minúsculas.
V. **La etiqueta *<template>*** define la respuesta del chat*bot* al *pattern* coincidente.

Esta sería la representación en AIML más básica de un chatbot en la interacción con un usuario, con una input: "HOLA", el bot contestaría: "Hola, me alegro de verte!"

```
<category>
    <pattern>HOLA</pattern>
    <template>Hola, me alegro de verte!</template>
</category>
```

Figura 3.1: Ejemplo 01 – Código AIML.

Esto puede complicarse, por ejemplo añadiendo **etiquetas *<random>*** (y etiquetando cada sentencia como elementos de una lista, con la etiqueta <*li*>) de manera que el chat*bot* sea más impredecible, y por lo tanto, más humano.

```
<category>
    <pattern>HOLA</pattern>
    <template><random>
        <li>Hola, me alegro de verte!</li>
        <li>Hola, cómo estás?</li>
    </random></template>
</category>
```

Figura 3.2: Ejemplo 02 – Código AIML.

Otra forma (mucho más avanzada) de establecer un diálogo trazable y complejo con un *chatbot*, es establecer ***topics***, mediante los cuales se puede establecer un contexto que abarque más de una interacción. Con ellos, podemos agrupar categorías o temas de conversación.

```
<category>
    <pattern>HOLA</pattern>
    <template><random>
        <li>Hola, me alegro de verte!</li>
        <li>
            Hola, cómo estás?
            <think><set name="topic">comoestas</set></think>
        </li>
    </random></template>
</category>

<topic name="comoestas">
    <category><pattern>^ BIEN ^</pattern>
        <that>HOLA, CÓMO ESTÁS</that>
        <template>
            <think><set name="topic"></set></think>
            Me alegro mucho
        </template>
    </category>
</topic>
```

Figura 3.3: Ejemplo 03 – Código AIML.

Con la **etiqueta *<think>*** conseguimos que el haber establecido el topic sea completamente invisible para el usuario, es decir, el *bot* lo "piensa" pero no nos lo transmite.

Con la **etiqueta *<that>*** indicamos al *bot* que solamente debe contemplar esa respuesta o *template*, en el caso en el que su respuesta anterior haya sido la contenida en la etiqueta *that*. Es decir:

- *Hola*
- *Hola, me alegro de verte!*
- *Muy bien*
- *...*

En este caso, el *bot* no entiende por qué se le ha dicho "Muy bien", no se había establecido un contexto (*topic*).
En cambio, en este segundo ejemplo, la respuesta es diferente:

- *Hola*
- *Hola, ¿cómo estás?*
- *Muy bien*
- *Me alegro mucho*

En el momento en que el *bot* contesta "Hola, cómo estás?" se establece un contexto o *topic*, el cual le indica al robot que si la respuesta del usuario, después de la pregunta contenida en la etiqueta ***<that>*** que, como recordamos era "Hola, como estás?" es alguna frase que contenga la palabra "Bien", el bot contestará: "Me alegro mucho".

· Debido a (that): *Hola, cómo estás"*

· Por el contexto (topic) con nombre: "*como estás"*

Es relevante mencionar que la etiqueta *that* no tiene en cuenta signos de puntuación (pero sí comas) interrogaciones y exclamaciones. Esa es la razón por la que no se ha añadido al ejemplo.

Además de éstas, hay muchas otras etiquetas que se utilizarán en casos específicos, como bucles (***count***), evaluación de información insertada por el usuario (***eval***), obtención de información del inmediato input del usuario (***star***) y almacenamiento de la misma (***get***) o simplemente obtención de la información almacenada en un momento anterior (***set***).

Por otra parte, una herramienta muy valiosa que tiene este lenguaje son los SET, una lista de elementos únicos como por ejemplo la lista de los colores, o los días de la semana, etc. y los MAP, una lista de pares (el equivalente a un diccionario Python) que pueden servir por ejemplo para hacer una búsqueda sinónimos o capitales del mundo.

El uso de los SET y los MAP, es la manera que tiene el bot de recordar información, contrastarla e incluso poder escoger aquello que en un momento determinado haya sido la mejor opción, así que con tal de esclarecer su manejo, muestro a continuación dos ejemplos prácticos:

SET

Con una previa "base de datos" a modo de archivo "SET" podríamos hacerle este tipo de consultas a nuestro chatbot:

- Humano: Es Mark Twain un escritor?
- Bot: Sí, Mark Twain es un escritor.
- Humano: Es Ernest Hemingway un escritor?
- Bot: Sí, Ernest Hemingway es un escritor.
- Humano: Es mantequilla de cacahuete un escritor?
- Bot: No, mantequilla de cacahuete no es un escritor.

MAP

Gracias a estos archivos, el robot puede comparar con su "base de datos" mediante pares de información.

- Humano: Cuál es la capital de California?
- Bot: Sacramento es la capital de California.
- Humano: Cuál es la capital de New York?
- Bot: Albany es la capital de New York.
- Humano: Cuál es la capital de Texas?
- Bot: Austin es la capital de Texas

4. DESARROLLO DE LA IA.

En este proyecto, todo mi diseño se basa en una interacción humana con un sistema inteligente por reconocimiento de voz y con capacidad de comunicación, de igual manera, con voz. Decidí, después de muchas pruebas con varios motores de este tipo, que el mecanismo de traducción de texto a voz y viceversa fuese Nuance, el más cercano al "Siri" de Apple, que todos conocemos.
Pero esta tecnología necesita de muchísimas líneas de código, para lo que he conseguido a día de hoy: Un sistema inteligente, con el que se puede tener una conversación.

Con el objetivo de construir este sistema he necesitado multitud de engranajes, partiendo de la base de la comunicación interna del robot a sí mismo, para saber cuándo entender y cuándo responder (Python, ROS) pasando por la tecnología que me permitía procesar el habla humana, para poder interactuar con el robot (Nuance) a la toma de decisiones y conocimiento adquirido del robot gracias a la programación implicada en el proyecto (Python, AIML, Pandorabots).

4.1. Primera parte: Utilización de ROS, caso específico.

1. Un nodo se enciende y se encarga de que la escucha esté activa. Cada vez que el robot detecta algún tipo de sonido, se lo manda a Nuance, y éste intenta obtener un resultado.
2. Si el audio que ha obtenido el nodo es un resultado válido (realmente se considera una palabra y no ruido de ambiente), el nodo manda a Nuance la información.
3. Nuance devuelve el audio en formato texto, dando así solución a la sentencia dicha por el usuario o, dicho de otra manera, reconociendo la voz del usuario para convertirla en formato texto.
    a. La solución que nos da Pandorabots será procesada por la programación de la IA, basada en lenguaje AIML.
4. Sea lo que sea lo que decida el motor de diálogo, se envía a Nuance para que éste pueda transcribir el texto escogido por la IA a voz, y ésta nos conteste en lenguaje natural.

4.2. Segunda parte: Reconocimiento de voz con Nuance (ASR) y conexión con Pandorabots.

Puesta en marcha de Nuance

El proceso de instalación de Nuance en una Raspberry no había sido documentado en el momento de la ejecución de este proyecto y, pese a la multitud de necesidades que una buena instalación del mismo requiere, finalmente se consiguió que funcionase con total normalidad.

Por tanto, simplemente descargando e instalando las dependencias de Nuance (requirements.txt), no conseguimos nada, dado que en Raspbian, bastantes dependencias con otras librerías no venían instaladas por defecto.

Un primer paso es instalar pip, pyaudio y sox.

```
sudo apt-get install python-pyaudio python-pip sox
sudo easy_install pip
```

Figura 4.1: Código de terminal, instalación de Nuance.

Después de esto, ya podemos realizar la descarga del cliente Nuance:

```
sudo wget http://github.com/nuancedev/ndev-python-http-cli/archive/master.tar.gz && tar -xvzf master.tar.gz
rm -rf master.tar.gz && mv ndev-python-http-cli-master/ Nuance
cd Nuance/
```

Figura 4.2: Código de terminal, instalación de Nuance.

E instalar las dependencias que éste requiere, mediante su instalador.

```
sudo pip install -r requirements.txt
sudo python setup.py install
```

Figura 4.3: Código de terminal, instalación de Nuance.

A continuación muestro el árbol de directorios del cliente de Nuance:

Figura 4.4: Árbol de directorios del cliente Nuance.

Una vez instalado satisfactoriamente, accedemos al cliente y empezamos a realizar las pruebas.

```
cd /opt/nuance/ndev-python-http-cli-master
```

Figura 4.5: Código de terminal, ejecución de Nuance.

Para grabar un fragmento de audio tenemos que recurrir al archivo "record.wav" que está dentro de la carpeta "bin", en este caso concreto, hemos llamado a la grabación de prueba "test.wav".

```
GRABAR:
(GRABACION DE UN ARCHIVO AUDIO)
    python bin/record_wav.py test.wav
```

Figura 4.6: Código de terminal, ejecución de Nuance.

Con tal de hacer una transducción eficiente, Nuance necesita que el archivo de audio sea de 8k o 16k, la siguiente imagen nos muestra cómo hacer esta conversión.

```
CONVERSION:
(CONVERSION DE UN ARCHIVO CUALQUIERA a 8k o 16k)
    ./bin/resample.sh test.wav 8k
    ./bin/resample.sh test.wav 16k
```

Figura 4.7: Código de terminal, ejecución de Nuance.

Una vez convertido a 8k o 16k, (el propio conversor renombra el "test.wav" a "test_8k.wav" para este caso concreto de conversión a 8k) ejecutamos el archivo "asr.py", nuevamente ubicado en la carpeta "bin", el cual se encargará de mandarle a *Nuance* el audio e intentar hallar una traducción de este en formato texto.

```
SPEECH RECOGNITION:
(ENVIO DE UN AUDIO Y RECEPCION DE UN STRING)
    ./bin/asr.py test_8k.wav -l es_ES
```

Figura 4.8: Código de terminal, ejecución de Nuance.

Con tal de que *Nuance* nos permita operar con esta comunicación bidireccional, es estrictamente necesario haber pedido unas credenciales (en el archivo "credentials.json"), que sirven para tener conectada en todo momento la comunicación con *Nuance* de forma codificada.

Una vez ya tenemos Nuance configurado, y teniendo en cuenta todo lo que necesitamos para construir este mecanismo inteligente de reconocimiento de voz y habla, ha sido necesario implementar una librería encargada del módulo ASR.
Estas son las dos funciones más importantes de la librería encargada de su gestión:

1. Función ***listen.***

   Esta función se encarga de encender o apagar el nodo que espera el input del usuario, en función de si el archivo dado es correcto o incorrecto, existente o inexistente, y cualquier combinación de estas cuatro variables. También se controla el tiempo de funcionamiento (timeout).

```
def listen(self, performance_callback):
    resp=""
    if self.last_recog_correct:
        performance_callback("listen_on")
    (filename,timeout) = self.recorder.listen(save_file=True)
    if timeout:
        self.last_recog_correct=False
    if self.last_recog_correct:
        performance_callback("listen_off")
    if filename !="" and os.path.isfile(filename):
        performance_callback("recognizing_on")
        resp = self.recognize(filename)
        performance_callback("recognizing_off")
        self.recorder.remove_file(filename)
        if resp!="":
            performance_callback("understand")
            self.last_recog_correct=True
        else:
            performance_callback("not_understand")
           # self.last_recog_correct=False
    return resp
```

Figura 4.9: Código fuente de la función "listen".

2. Función ***recognize.***

   Si aquello que ha reconocido el nodo es válido (y no nulo o inexistente) se realiza la consulta al servidor de Nuance ASR.

```
"""
recognizes a given an audio file using the Nuance Cloud.
"""
def recognize(self, filename):
    if filename is None:
        return None

    resp = self.cloud_request.get_json_request(filename, "asr", self._get_final_lang())
    print filename
    if resp != "" and resp != None:
        resp=resp.get('recognition')
        return resp
    return ""
```

Figura 4.10: Código fuente de la función "recognize".

4.3. Tercera parte: Interpretación del lenguaje y toma de decisiones (AIML).

Como con si se tratase de un "hola mundo" voy a explicar la más sencilla interacción con la IA. Nos disponemos a probar un diálogo exactamente igual al siguiente:

- *Usuario: HOLA*
- *IA: Hola, cómo estás?*
- *Usuario: MUY BIEN*
- *IA: Me alegro mucho.*

Para ello, debemos indicar cuál es el camino que la IA deberá seguir, dada una input "HOLA" estipulada por el usuario…

En este caso concreto de pregunta-respuesta he contemplado la posibilidad de no siempre preguntar al usuario "cómo estás?" y hay un 33% de probabilidades de hacerlo. De todos modos, en el momento en el que esta pregunta sea lanzada, el bot estará preparado para seguir un tema de conversación, en este caso acerca del estado anímico del usuario.

```
<category><pattern>HOLA</pattern>
<template><random>
<li>Hola, me alegro de verte!</li>
<li>Hola, es un placer volver a verte!</li>
<li>
    <think><set name="topic">comoestas</set></think>
    Hola, cómo estás?
</li>
</random></template>
</category>
<topic name="comoestas">
    <category><pattern>^ BIEN ^</pattern>
    <that>HOLA, CÓMO ESTÁS</that>
    <template>
        <think><set name="topic"></set></think>
        Me alegro mucho</template>
    </category>
    <category><pattern>^ MAL ^</pattern>
    <that>HOLA, CÓMO ESTÁS</that>
    <template>
        <think><set name="topic"></set></think>
        Intenta ser un poco más positivo
    </template>
    </category>
    <category><pattern>#</pattern>
    <that>HOLA, CÓMO ESTÁS</that>
    <template>
        <think><set name="topic"></set></think>
        Me inquietas...
    </template>
    </category>
    <think><set name="topic"></set></think>
</topic>
```

Figura 4.11:Código AIML de ejemplo.

Y este es el resultado en la lógica de la IA, utilizando *Pandorabots* como plataforma web para trabajar con nuestros ficheros AIML.

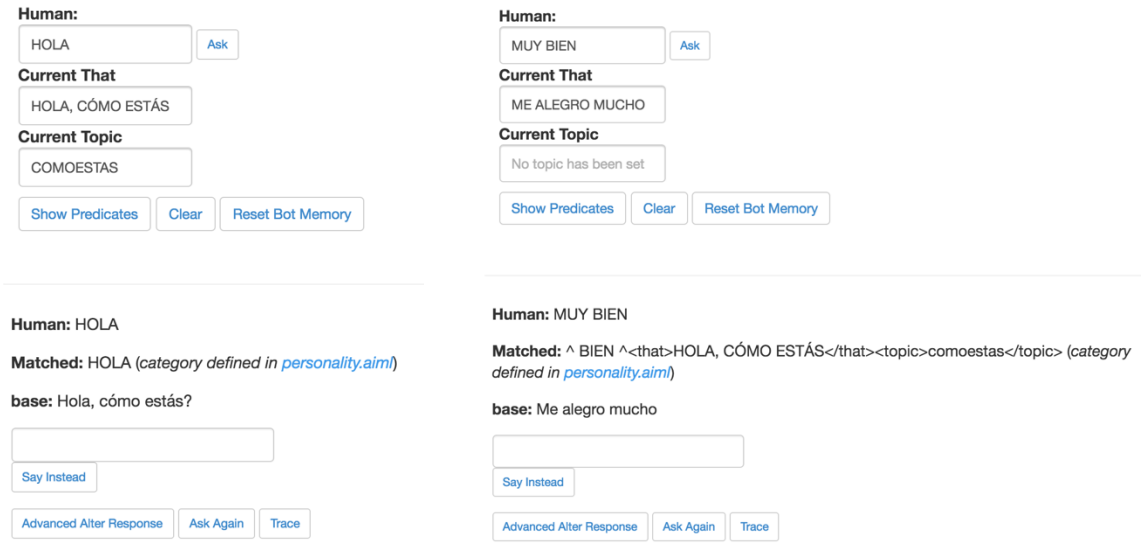


Figura 4.12:Capturas ejemplo de la plataforma Pandorabots.

Como se puede observar, el *bot* también está preparado para recibir una respuesta negativa, como por ejemplo, que el usuario está mal (anímicamente hablando). He aquí la prueba:

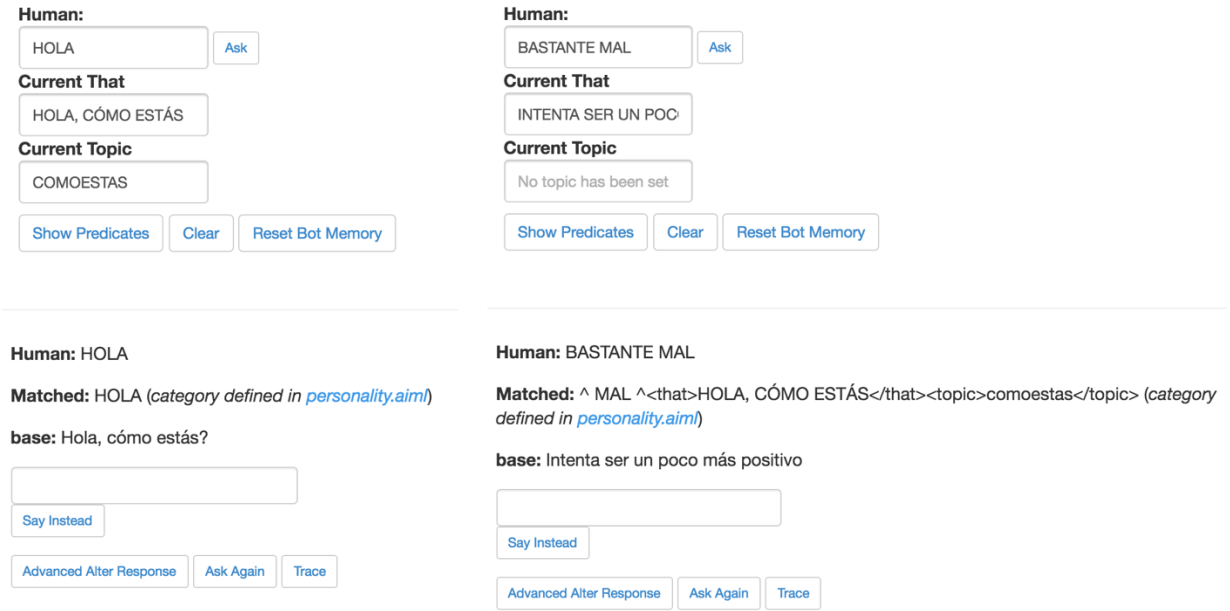


Figura 4.13:Capturas ejemplo de la plataforma Pandorabots.

Si nos centramos en la trazabilidad de la conversación, en las cuatro imágenes se puede observar cómo el *bot* detecta que ha preguntado acerca del estado anímico del usuario, y se mantiene en el *topic* "comoestas", preparado para la respuesta del cliente, pero una vez recibida la respuesta, es capaz de cambiar de tema, y haber resuelto la incógnita (en este caso bastante sencilla) acerca del usuario.

Pero ¿y si no contestamos al robot algo coherente, o que no se espera, acerca de cómo estamos? Las imágenes a continuación, clarificarán esta pregunta:

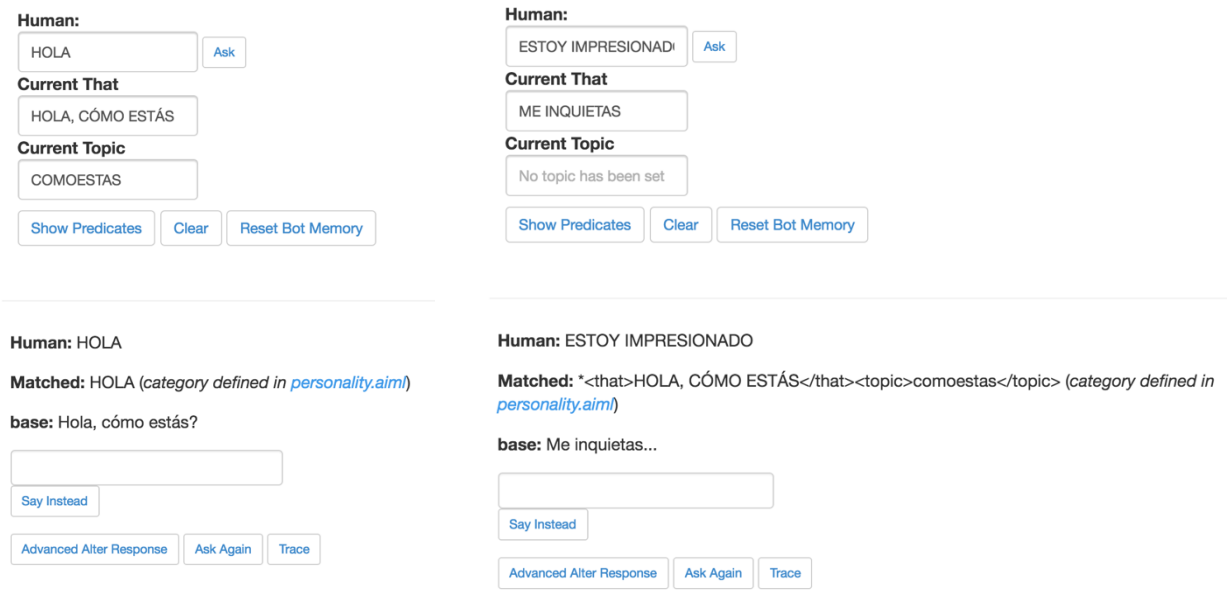


Figura 4.14:Capturas ejemplo de la plataforma Pandorabots.

Como se puede apreciar, la capacidad de este motor de diálogo es enorme y, a su vez, extremadamente delicada.

Por otra parte, ha sido necesaria una librería para controlar las interacciones del resto de nodos con *Pandorabots* (la herramienta utilizada para procesar nuestro código AIML)

```
import requests
host_base = "https://"

def talk(user_key, app_id, host, botname, input_text, session_id=False,
        client_name=False, recent=False, reset=False, trace=False, clientID=False,atalk=False):
    path = '/talk/' + app_id + '/' + botname
    if not session_id and not client_name:
        path = '/atalk/' + app_id + '/' + botname
    url = host_base + host + path
    query = {"user_key": user_key,
             "input": input_text
             }
    if recent:
        query['recent'] = recent
    if session_id:
        query['sessionid'] = session_id
    if client_name:
        query['client_name'] = session_id
    if reset:
        query['reset'] = reset
    if trace:
        query['trace'] = trace
    if clientID:
        query['client_name'] = clientID
    response = requests.post(url, params=query)
    result = response.json()
    status = result['status']
    output = {}
    if status == 'ok':
        output["response"] =  result['responses'][0]
        if reset:
            output["output"] = 'Bot has been reset.'
        if trace:
            trace_text = result['trace']
            trace_string = 'Trace: '
            for elt in trace_text:
                if 'status' in elt.keys():
                    trace_string +='Level: ' + str(elt['level'])
                    trace_string += ' Sentence to process: ' + ' '.join(elt['input']) + ' '
                    trace_string += 'Matched pattern: ' + str(elt['matched'][0])
                    trace_string += ' from file: ' + elt['filename']
                    trace_string += ' template: ' + elt['template'] + '\n'
            output["trace"] = trace_string
        if 'sessionid' in result:
            output["sessionid"] = result['sessionid']
        if 'client_name' in result:
            output["client_name"] = result['client_name']
    else:
        output["response"] = result['message']
    return output
```

Figura 4.15:Código fuente de la función "talk" del nodo ROS.

Una descripción *grosso modo* de la figura anterior sería que en base a una serie de identificadores de nombre de usuario, direcciones y sesión, esta tecnología es capaz de comunicarse con nuestro *bot* en la nube, enviarle los datos recogidos por Nuance y que éste decida cuál será la siguiente pregunta o respuesta que hay que darle al usuario. Todo ello, como se puede observar, con una trazabilidad lo más exhaustiva posible (patrón encontrado, nombre del fichero que se ha utilizado, sentencia procesada… etc.)

4.4. Cuarta parte: Interpretación de Pandorabots hacia Nuance (TTS).

La operación inversa con *Nuance* (texto a audio) es muchísimo más sencilla que la anterior, ya que la única operación que necesitamos realizar es un envío de texto al archivo tts.py (contenido en la carpeta "bin"), con los parámetros de idioma y "bitrate".

```
CREAR UN .wav MEDIANTE UN STRING:
(CREACION DE UN AUDIO PASANDO UN STRING)
    ./bin/tts.py test.wav "text" -l en_US -r 8000
    ./bin/tts.py test.wav "text" -l en_US -r 16000

    ./bin/tts.py test.wav "texto" -l es_ES -r 8000
    ./bin/tts.py test.wav "texto" -l es_ES -r 16000
```

Figura 4.16: Código de terminal, ejecución de Nuance.

Por tanto, obteniendo las *output* de Pandorabots, la operación que debemos hacer es convertir aquello que ha procesado el motor de diálogo y exteriorizarlo en forma de audio, para así finalizar el segmento de comunicación con el usuario.
Esto llevado a un entorno ROS, con el cual trabajar automáticamente, de manera ágil, conlleva el desarrollo de una librería.

Esta es la función principal de la librería que hace llamadas al servicio Cloud de Nuance TTS:

1. Función ***say.***

   Es la encargada de recibir el mensaje del servicio de Nuance TTS (archivo de audio) y mandarlo a la cola de reproducción, creando una copia de seguridad en un archivo temporal

```
"""
Split sentences, get the audio and send it to a queue to play it.
"""
def say(self, sentence, wait=True):
    self.__set_desired_tts_lang__()

    sentence = self.__add_pause_to_symbols(sentence)

    for elem in sentence.split("."): #añadir más símbolos
        if elem != "":
            elem=elem.strip()
            filename = tempfile.NamedTemporaryFile().name+"."+self.AUDIO_TYPE #creates a temporal file to save the audio
            #SystemHelper.create_file(filename)
            if self.__get_audio__(elem, filename):  #it records the audio
                self.queue.put(filename)            #adds the audio to be played in a queue

    while wait and not self.queue.empty():
        time.sleep(0.1)
```

Figura 4.16: Código fuente de la función "say" del nodo ROS.

botbase.properties

Este es el fichero que almacena toda la información del robot. Es invisible al usuario, por lo que es lo más cercano a un fichero de configuración. Además de simplemente establecer valores de personalidad, también podemos controlar los marcadores de final de frase (splitters), la capacidad máxima de aprendizaje y algunos "por defecto" muy interesantes.

bot_profile.aiml

En este fichero se puede acceder a toda la información acerca del robot mediante las preguntas adecuadas: Nombre, edad, género… Todas estas respuestas serán extraídas del fichero "botbase.properties".

client_profile.aiml

Este fichero almacena la personalidad del usuario: color favorito, nombre, edad, género…

default.aiml

Este es uno de los archivos más importantes en la personalidad de la IA. Ha sido sometido a una adaptación lingüística, comparando ficheros en otros idiomas de modo que contiene un porcentaje muy elevado de preguntas y respuestas a los temas más comunes que se han hablado con un bot en el paso de los tiempos. Con una extensión de catorce mil líneas de código AIML, está preparado para contestar a la mayoría de inserciones del usuario.

si.set y **no.set**

Estos sets son utilizados para contemplar todas las variaciones de "sí" y "no" que se han considerado posibles. De esta manera, no hay código duplicado para distintas formas de decir "sí" o "no".

personality.aiml

El fichero personality o fichero de personalidad, es el fruto de una depuración y un trabajo exhaustivo. En función de las inserciones que los usuarios han ido haciendo, las mías propias y el simple hecho de haber pensado en multitud de posibles combinaciones y haberlas hecho realidad en el código. Todo ello hace que este fichero sea el más

característico del robot, mediante el cual, la personalidad del robot saldrá a la luz, con preguntas-respuestas como:

- *Usuario: Crees en Papá Noel?*
- *Bot: Quién te dice que no me ha diseñado él?*

Microframes

Gracias a documentación referente a patrones de conducta en el ámbito lingüístico, he podido saber cómo funciona el cerebro humano en las diferentes fases de una conversación.

De esta manera, he podido denotar cambios emocionales en la personalidad del bot, silencios o estrategias verbales cuando el bot no ha entendido algo (entre otras muchas cosas).A esto, lo he denominado *Microframe* y en base a la información contrastada, he catalogado varios tipos basados en:

1. Microframes del bot.
    a. Estrategias interactivas.
       El bot le pide al usuario que interactúe con él, pidiéndole que haga algo.

    b. Estrategias Metalingüísticas o pragmáticas.
       Estrategias verbales del bot cuando no ha entendido nada del input.

    c. Estrategias verbales.
       El bot le hace preguntas al usuario para sacarle tema de conversación o simplemente hace reflexiones.

2. Microframes de patrones conductuales
    a. Patrones paralingüísticos.
       Un ejemplo de este tipo de patrón sería un bostezo, un ronquido o un carraspeo. Hay patrones de este tipo insertados en el proyecto.

    b. Patrones emocionales.
       El expresar felicidad, miedo o simplemente reír, son algunos de los ejemplos que he podido implementar en el motor de diálogo.

3. Microframes del usuario.

Sin datos de un corpus de usuario, es más difícil modelar su conducta. En el futuro, con una hipotética aplicación chatbot online podremos saber de qué cosas le gusta a la gente hablar con sus amigos bots, pero de momento, hemos seguido estas tres estrategias:

a. Preguntas reflejo. Hemos sobreentendido que el usuario tenderá a devolverle las preguntas personales al bot cuando este las realice. De tal manera que si el bot pregunta la edad, es probable que el usuario pregunte seguidamente: ¿y tú? ¿Cuántos años tienes? Esto puede dar bastante juego, ya que a veces, podemos re direccionar la conversación a nuestro favor.

b. Palabras léxicas más frecuentes en español. La RAE tiene una lista de 1000 palabras más usadas en español. Me he inspirado en ellas para modelar algunos microframes.

c. Interacción directa con el usuario. El usuario le pide al bot que haga cosas.

d. Preguntas sobre el usuario. El usuario puede preguntarle al robot cosas sobre él que ya le ha contado, tipo: cómo me llamo? Cuántos años tengo? Quién soy yo? …etc. (todo esto, está controlado por el fichero aiml: "*client_profile"*)

4. Microframes de temas concretos.
   a. Microframe de robótica.

      El motor de diálogo, puede ser una poderosa herramienta, no sólo para tener una conversación de temas dispares, sino para informar o razonar respecto a un tema concreto. En este caso, el tema que escogí para su estudio y desarrollo fue el de robótica.

      Para ponernos en situación y profundizar un poco más, el robot conoce del tema del que se está hablando, puede seguir una conversación compleja en este caso de hasta 3 niveles (entendiendo como niveles de una conversación, la capacidad que tiene el chatbot de seguir el hilo de una conversación de 2^3 maneras diferentes) dependiendo de las input del usuario.

6. REFLEXIÓN PERSONAL

El objetivo de este proyecto, ha sido moldear una Inteligencia Artificial capaz de colaborar con el ser humano y tomar decisiones en base a la información previamente obtenida del usuario. Así como tomar sus propias decisiones de forma autónoma. Después de la automatización del procesamiento de comunicación con el *chatbot*, el sistema lógico del *bot* es capaz de almacenar, procesar y decidir aquello que es más conveniente en un momento determinado en base a la programación previamente realizada.

Esto podría tener aplicaciones prácticas de beneficio potencial para la sociedad, en múltiples y variadas funciones, tales como ayuda social para personas con Alzheimer que necesitan un apoyo real o virtual que les esté ayudando a saber qué han hecho o qué deben hacer, dentro de unas pautas, que podrían haber sido programadas y personalizadas en función de cada caso.

Por contraposición, para lograr que un sistema de este tipo fuese eficiente, necesitaríamos una inmensa base de datos, y bastantes años de documentación y desarrollo, en este caso focalizados en el Alzheimer.

De alguna manera, este tipo de reflexiones nos hacen darnos cuenta de que la tecnología, y concretamente la Inteligencia Artificial, es un campo muy poderoso con el cual podemos mejorar nuestra calidad de vida en diferentes ámbitos, tanto en temas de salud como de cualquier tipo.

Personalmente creo que es la ciencia del mañana, o dicho de otra manera, aquello por lo que la sociedad va a apostar en un futuro no muy lejano y como prueba de ello, hago referencia a aquello por lo que las empresas más grandes en la actualidad están implicando su esfuerzo, como son las Smart Cities, el asistente de Apple “Siri” o el asistente de Google “Google Now” asimismo el gigante de la comunicación Facebook, también posee un largo abanico de Inteligencia Artificial, capaz de recolectar todo tipo de información de sus usuarios y procesarla exitosamente.

7. BIBLIOGRAFÍA

Enlaces:

The Robot Operating System (ROS) Documentation (versión febrero de 2014). Recuperado de*: http://wiki.ros.org/*

Pandorabots Documentation. How to build a bot using the Playground UI (versión junio de 2016). Recuperado de*: https://playground.pandorabots.com/en/tutorial/*

Libros:

Richard S. Wallace. *(Marzo de 2005). Be Your Own Botmaster. 2nd Edition: Foundation, Inc.*

Enerlis, Ernst and Young, Ferrovial and Madrid Network. *(Septiembre de 2012). El libro blanco de las Smart Cities. España: Imprintia.*

Nuance Cloud Services. *(Diciembre de 2013). HTTP Services 1.0 Programmer's Guide. Nuance, LTD.*

Kathrin Haag. *(Mayo de 2016). Pandorabot Playground Documentation. Edinburgh: University of Edinburgh.*

Revistas:

Ana González Ledesma. *(Septiembre de 2013). Patrones de conducta lingüística.*

Mark C. Marino. *(Diciembre de 2014). The Racial Formation of Chatbots. CLCWeb: Comparative Literature and Culture, volumen 16 , número 5.*