eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ Trabajo Fin de Grado ▪

Computación

Creación de diálogos inteligentes en robots antropomorfos

Alexander Triguero Fonseca

Junio 2019

Resumen

Este proyecto comenzó sin tener conocimientos previos por parte del alumno de lo que se podía hacer con el robot NAO. A medida que ha avanzado el desarrollo del proyecto, he descubierto sus capacidades, ampliando las posibilidades y ha aumentado el interés por continuar y conseguir algo más completo.

El planteamiento original de objetivos del proyecto se reducía a explorar las capacidades del robot y hacer un sencillo programa que aprendía pares de pregunta-respuesta. Gracias a la investigación que sobre el tema, ha aumentado mi interés sobre estos sistemas y se profundizado más en el proyecto, yendo más allá de las funcionalidades que se habían pensado en un principio.

La realización del proyecto ha sido todo un desafío, muy diferente a lo que había hecho antes durante toda la carrera: no recibes toda la información que necesita para avanzar nada más empezar, debes trabajar de forma individual y descubrir la forma de solucionar los problemas que ocurren durante la realización del trabajo, desarrollando un alto grado de autonomía en el proceso.

En su conjunto, el proyecto me ha ayudado a aprender a trabajar de forma independiente y sin la necesidad de tener a alguien supervisando y que ayuda en todos los problemas que surgen. La experiencia ha sido perfecta para concluir el grado y prepararse para la vida laboral.

Lista de Figuras y Tablas

FIGURAS

TABLAS

Índice

1. Introducción a Sistemas de Diálogo

Antes de empezar a realizar el proyecto, había tenía que informarme sobre el contexto del mismo, tanto los sistemas que se utilizan para generar este tipo de programas, además de cuales son los sistemas de dialogo más utilizados hoy en día.

1.1. FUNDAMENTOS DE LOS SISTEMAS DE DIÁLOGO

Los sistemas de diálogo[1][2] hablado presentan muy interesantes desafios porque su implementación requiere varias tecnologías para procesar y generar el lenguaje natura con una calidad que permita la interacción natural con un humano, lo que es una tarea compleja. Normalmente, estos sistemas se construyen utilizando las siguientes cinco tecnologías básicas en un proceso que se ilustra en la figura 1 (usamos los acrónimos de los nombres en inglés) :

- Reconocimiento de Voz Automático (Automatic Speech Recognition, ASR)
- Comprensión del Lenguaje Hablado (Spoken Language Understanding, SLU)
- Gestión del Diálogo (Dialogue Management, DM)
- Generación de Lenguaje Natural (Natural Language Generation, NLG)
- Síntesis de Texto a Voz (Text-to-Speech synthesis, TTS)

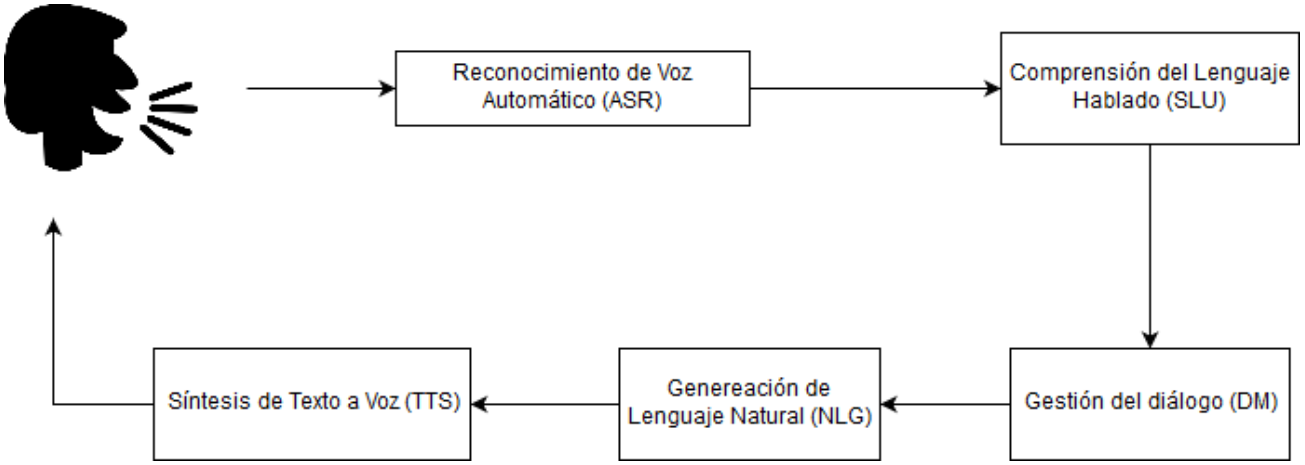

Figura 1: Proceso de dialogo hablado automático y tecnologías involucradas

1 http://loquens.revistas.csic.es/index.php/loquens/article/view/17/47#RF0001_

2 https://www.csie.ntu.edu.tw/~yvchen/doc/DeepDialogue_Tutorial_ACL.pdf

1.1.1. Reconocimiento de Voz Automático (ASR)

El módulo que implementa el ASR se conoce como “reconocedor de voz”, su objetivo es recibir el habla del usuario y generar una hipótesis de reconocimiento, esta hipótesis es una secuencia de palabras más verosímil como interpretación de lo que el usuario ha dicho. Desafortunadamente, estas hipótesis suelen contener errores en muchos casos.

Este tipo de errores pueden darse por varias circunstancias, incluyendo el sonido captado debido al ruido ambiente, ambigüedades debidas a palabras con similitud acústica y fenómenos relacionados con el habla espontánea, como pausas o momentos en los que el hablante duda o se traba.

1.1.2. Comprensión del Lenguaje Hablado (SLU)

Este módulo es el que se encuentra a continuación del reconocedor de voz y recibe como input la respuesta que da el ASR. El objetivo de este módulo es obtener una representación semántica del input obtenido, la cual se suele guardar en una estructura que estará compuesta de un conjunto de espacios. Por ejemplo, la frase *“encuéntrame un buen sitio para comer comida taiwanesa”* tendría los siguientes espacios para poder entender el resultado de este módulo:

TipoDeAccion: “Buscar restaurante”

Localización:-

CalidadDeComida: “buena”

TipoDeComida: “tailandesa”

Con esta correspondencia entre los elementos fonéticos reconocidos en la frase y los espacios equivalentes, el sistema se puede construir una interpretación de lo que el usuario desea expresar. De todas maneras, es posible que la extracción de los elementos fonéticos no se haya realizado correctamente al hacer el reconocimiento de voz, por ello, además de construir la correspondencia semántica, se guarda una medida de confianza de dichas palabras o conjunto de las mismas, que expresa cuán seguro está el sistema de que dicha palabra está correctamente extraída de la señal de voz.

Volviendo al ejemplo de antes, el sistema ya sabe que el usuario busca un restaurante que tenga comida de buena calidad, pero resulta que en el tipo de comida reconocido en la señal de voz es “tailandesa” en lugar de “taiwanesa”. Esto ocurre porque ambas palabras tienen una similitud acústica y ha tomado una de ellas. Al no estar muy seguro de cuál de las dos es la correcta la medida de confianza en el reconocimiento de esta palabra probablemente sea baja. Observar si una palabra tiene una fiabilidad baja puede ayudar a identificar palabras que no se han entendido correctamente o de las cuales haya dudas sobre su significado real.

La tarea que debe realizar el módulo SLU es muy exigente debido a las dificultades específicas inherentes al procesamiento del lenguaje natural, como la ambigüedad entre otras. Para llevar a cabo la SLU, este módulo emplea reglas gramaticales o enfoques estadísticos, o una combinación de

ambos. Además, puede emplear información del módulo de historial de diálogo, el cual mantendría registro de anteriores interacciones y cambios de diálogo del usuario, con el fin de averiguar si anteriormente el usuario ha proporcionado información esencial.

1.1.3. Gestión del Diálogo (DM)

El módulo que implementa la Gestión del Diálogo generalmente se denomina "administrador de diálogo" y recibe su entrada del módulo SLU. El objetivo de este módulo es decidir qué es lo que tiene que hacer el sistema a continuación, ya sea informar al usuario, pedirle que confirme palabras que no se han entendido correctamente o incluso pedirle que reformule la frase. En el ejemplo anterior *"encuéntrame un buen sitio para comer comida taiwanesa"*, el sistema le había dado una confianza baja a la palabra "tailandesa", por tanto, al ver que su fiabilidad es bastante baja, el administrador de diálogo podría decidir generar una solicitud de confirmación para el "TipoDeComida".

Para proporcionar información al usuario, el administrador de diálogo suele consultar una base de datos o buscar información en internet y, como se ha mencionado anteriormente, también tiene en cuenta información que el usuario ha proporcionado en diálogos previos.

Continuando con el mismo ejemplo, el sistema se daría cuenta de que falta información para encontrar un restaurante apropiado, ya que no tiene información acerca de la ubicación. Entonces, podría pedir esta información para rellenar el espacio de "Localización", con la cual ya se podría encontrar un restaurante adecuado.

1.1.4. Generación de Lenguaje Natural (NLG)

La decisión tomada por el administrador de diálogo sobre qué acción debe realizar el sistema a continuación es el input de este módulo. Como la decisión se representa de manera abstracta (por ejemplo *"Solicitar(localización)"*), el objetivo es transformar dicha decisión en una o más oraciones que deben ser gramática y semánticamente correctas, así como coherentes. Para realizar esta transformación se suelen utilizar plantillas para generar varios tipos de oraciones. Algunas partes de estas plantillas son fijas pero otras representan espacios que deben ser rellenados con datos proporcionados por el administrador.

Para generar un diálogo coherente y natural, el módulo NLG debe generar oraciones teniendo en cuenta las interacciones previas que se han producido. Esto implica utilizar algunos recursos dialécticos, como omitir algunas palabras en las oraciones que ya han sido mencionadas previamente, y usar pronombre en lugar de sustantivos. Para llevar a cabo esta tarea, se utiliza el historial de diálogos, que almacena palabras utilizadas recientemente. Este módulo también debe evitar la información redundante en la salida.

1.1.5. Síntesis de Texto a Voz (TTS)

Por último, las oraciones de lenguaje natural generadas por el módulo NLG representadas internamente en formato de texto son el input de este módulo cuya función es transformar las oraciones en texto en señal de voz (audio).

A diferencia de otros métodos simples para la síntesis de voz basados en concatenación de palabras grabadas con anterioridad, el proceso de TTS permite convertir en voz cualquier texto, sin necesidad de tener pregrabadas todas las palabras de la oración.

El proceso de TTS es muy complejo debido a una serie de razones. Una de ellas es la posibilidad de existencia de abreviaciones en las oraciones y otras secuencias de palabras que no puedan ser transformadas directamente en señal de vez mediante transcripción fonética (por ejemplo, números). Otra razón es que la pronunciación de las palabras no es siempre la misma. En español, gracias a las tildes y las reglas de las sílabas tónicas ayudan a realizar la síntesis de la voz de forma natural. En idiomas como el inglés, la generación correcta de la pronunciación no es tan simple.

Por lo tanto, el proceso TTS requiere dos pasos. El primero es reemplazar secuencias de palabras o abreviaturas por transcripciones fonéticas correspondientes. El segundo hace un análisis lingüístico del input para incluir marcas que indiquen la pronunciación adecuada.

1.2. EVOLUCIÓN DE LOS SISTEMAS DE DIÁLOGO

Durante muchos años, el ser humano ha querido poder comunicarse con compañeros mecánicos. Existen muchos ejemplos de esto en películas y literatura, a pesar de que sean ficticios demuestran que se ha tenido esta idea en la cabeza durante bastante tiempo. Es más, en la antigua mitología Griega y Romana aparecían momentos donde ciertos héroes eran capaces de comunicarse con estatuas de dioses o guerreros, lo cual podría tomarse como la primera idea de lo que en el futuro serían los sistemas de diálogo. Los primeros intentos de la creación de estos sistemas comenzaron en los siglos dieciocho y diecinueve, cuando se creó el primer autómata para imitar el comportamiento humano. Estas primeras máquinas eran mecánicas y no fue hasta el final del siglo diecinueve que los científicos concluyeron que el habla podía representada mediante una señal eléctrica. Esta señal podía ser captada mediante membranas que oscilan con la vibraciones del sonido, y generada mediante membranas que se mueven impulsadas por las variaciones en la señal eléctrica.

1.2.1. Primeros sistemas y proyectos de investigación

Al comienzo del siglo veinte se creó la primera máquina capaz de generar sonidos de voz a partid de señales eléctricas y en los años 30, se creó el primer sistema capaz de producir cualquier tipo de sonido. Al mismo tiempo apareció el primer sistema con capacidades de procesamiento del lenguaje natural (PLN), el cual era muy básico y se utilizaba para aplicaciones de traducción automática. En los años cuarenta, se creó la primera computadora y algunos distinguidos científicos como Alan Turing demostraron su potencial para aplicaciones que demandaban inteligencia simbólica más allá de los calculos numéricos de modelos matemáticos.

Este fue el punto de partida que fomentó las iniciativas de investigación que en los años 60 dieron como resultado los primeros sistemas basados en el lenguaje. Por ejemplo, ELIZA, se basaba en localizar palabras clave en las frases de entrada y aplicar plantillas predefinidas que transformaban la entrada que daba el usuario en respuestas del sistema. A pesar de acertar en

varias ocasiones, a la hora de responder, la conversación acababa con respuestas incoherentes por parte de ELIZA.

Beneficiándose de la incesante mejora en el campo de Reconocimiento Automático de Voz (ASR), el Procesamiento de Lenguaje Natural y la síntesis del habla aparecieron en los años 80. El orígen de este área de investigación está vinculado con dos proyectos: DARPA Spoken Language Systems en EEUU y Esprit SUNDIAL en Europa. Estos Proyectos fueron el punto de partida de investigación en MIT y CMU, donde se han creado algunos de los más importante sistemas.

El proyecto DARPA Communicator se destaca como uno de los proyectos de investigación más importantes en los años 90. Este proyecto financiado por el gobierno americano tuvo como objetivo el desarrollo de tecnologías de voz de vanguardia, que podrían emplear como entrada tanto el habla, como otras modalidades.

En la actualidad, los expertos han propuesto objetivos de mayor nivel para desarrollar SDS, como proporcionarles razonamiento avanzado, capacidad de resolución de problemas, capacidad de adaptación, proactividad, inteligencia afectiva, multimodalidad y multilingüismo. Estos objetivos se refieren al sistema de diálogo en su conjunto y representan tendencias importantes que en la práctica se logran a través del trabajo conjunto en diferentes áreas y diferentes componentes del sistema.

1.3. SISTEMAS DE DIÁLOGO ACTUALES

Los sistemas de diálogo automático[3] ya se han establecido en nuestra sociedad, y hay mucha gente que los utiliza en el día a día. Los más conocidos sistemas de diálogo son aquellos conocidos como “Asistente virtual”, los cuales son sistemas de diálogo que se han creado en los últimos años y a los que mucha gente tiene acceso de forma sencilla

La función principal de estos asistentes es crear conversaciones bidireccionales con usuarios para ayudarles a realizar tareas del día a día con mayor comodidad. Estos asistentes tratan de proporcionar una interacción entre persona y máquina de forma natural mediante una comunicación por voz, por lo tanto, estos sistemas deben estar preparados para comprender todo tipo de expresiones propias de los seres humanos. Los asistentes Alexa, Cortana, Siri y el Asistente de Google son los más conocidos, y cualquier persona con un smartphone o ordenador personal tiene acceso a alguno de estos.

La utilidad de estos asistentes no reside únicamente en el sistema de diálogo automático, sino en la mezcla de distintas tecnologías para poder realizar diferente acciones mediante comandos por voz.

3 https://es.wikipedia.org/wiki/Asistente_virtual#Dispositivos_y_objetos_con_asistentes_virtuales

1.3.1 Alexa

Alexa[4] es el asistente virtual desarrollado por Amazon, el cual fue utilizado por primera vez en los altavoces inteligentes de Amazon Echo en 2014.

En junio de 2015, Amazon anunció Alexa Found, un programa que invertiría en compañías que fabrican habilidades y tecnologías de control de voz. En 2016, se anunció el Premio Alexa, para fomentar la tecnología de interacción mediante voz.

Actualmente, Alexa está disponible en varios idiomas, inglés, alemán, japonés, francés, italiano y español. Este asistente es capaz de responder preguntar, reproducir música de servicios como Spotify, Apple Music, Tuneln y Amazon Music, crear listas de comprobación, establecer alarmas o temporizadores, reproducir podcasts y audiolibros, controlar dispositivos inteligentes, proveer información en tiempo real del clima, tráfico y dar resúmenes de noticias actuales.

Además de esto, puede utilizarse para controlar otros dispositivos inteligentes que sean compatibles como cámaras o focos. Mediante la app de Alexa se pueden ampliar sus capacidades, además de crear rutinas para automatizar dispositivos basándose en comandos de voz, hora o ubicación.

Amazon permite a los fabricantes de dispositivos integrar las capacidades de voz de Alexa, para ello tienen que conectarse utilizando el Alexa Voice Service (AVS), un servicio en la nube que permite que las APIs interactúen con Alexa.

1.3.2. Cortana

Cortana[5] es el asistente virtual creado por Microsoft para Windows 10, Windows 10 Mobile, Windows Phone 8.1, altavoz inteligente Invoke, Microsoft Band, Xbox One, iOS y Android entre otros.

El desarrollo de Cortana comenzó en 2009 en el equipo de productos Microsoft Speech, el cual creó un equipo con experiencia para crear los prototipos iniciales de Cortana.

En enero de 2015, Microsoft anunció la disponibilidad de Cortana para equipos de escritorio y dispositivos móviles con Windows 10 como parte de la fusión de Windows Phone en el sistema operativo general. En este mismo año, se estableció una versión de Android e iOS. Durante el E3 del mismo año, Microsoft anunció que Cortana se añadiría a la Xbox One.

Cortana puede establecer recordatorios, reconocer voz natural y responder preguntas utilizando información de motor de búsqueda de Bing.

Actualmente, Cortana está disponible en inglés, portugués, francés, alemán, italiano, español, chino y japonés.

4 https://es.wikipedia.org/wiki/Amazon_Alexa
5 https://es.wikipedia.org/wiki/Microsoft_Cortana

1.3.3. Siri

Siri[6] es un agente dotado de inteligencia artificial con funciones de asistente personal creada para iOS, macOS, tvOS y watchOS.

Siri es la primera aplicación del mundo que funcionó como asistente personal virtual, fue creada por Diri INC, una empresa de tecnología que surgió a partir de SRI Internacional. En 2010 la empresa fue comprada por Apple y Siri fue incorporada en los iPhone.

Esta aplicación utiliza procesamiento de lenguaje natural para responder preguntas, hacer recomendaciones y realizar acciones mediante la delegación de solicitudes hacia un conjunto de servicios web.

Entre las cualidades destacadas por la campaña de mercado de la aplicación se afirma que Siri es capaz de adaptarse con el paso del tiempo a las preferencias del usuario, personalizando búsquedas web y realizando tareas como reservar mesa en un restaurante, pedir un taxi, predecir el clima, o escribir mensajes de WhatsApp.

Actualmente, está disponible en una gran variedad de idiomas: inglés, francés, ruso, alemán, japonés, coreano, chino, italiano, español, neerlandés, tailandés, portugués, noruego, danésm turco, sueco, árabe, finés, malayo y hebreo.

1.3.3. Asistente de Google

Como el nombre indica, el Asistente de Google[7] es un asistente virtual desarrollado por Google que está disponible en dispositivos inteligentes móviles y domésticos.

El asistente de Google fue presentado por primera vez durante la conferencia de desarrolladores de Google en 2016, como parte de la presentación del Google Home y la nueva aplicación de mensajería Allo.

Después de un periodo de exclusividad en teléfonos inteligentes Pixel y Pixel XL, comenzó a implementarse en otros dispositivos Android en febrero de 2017 y se lanzó como una aplicación independiente en el sistema operativo de iOS en mayo.

Mayormente, los usuarios interactúan con el asistente mediante voz, aunque también admite entradas de teclado. De la misma forma que Google Now, el Asistente puede buscar en Internet, programar eventos y alarmas, ajustar la configuración de hardware en el dispositivo del usuario y mostrar información de su cuenta de Google.

En cuanto a idiomas, se puede utilizar en inglés, francés, hindi, indonesio, alemán, italiano, japonés, coreano, portugués y español.

---

6 https://es.wikipedia.org/wiki/Siri

https://www.nextu.com/blog/la-historia-detras-de-siri-y-su-creador-el-hombre-que-nacio-desarrollador-de-aplicaciones/

7 https://es.wikipedia.org/wiki/Asistente_de_Google

2. Objetivos y Recursos del Proyecto

A continuación, se analizarán los objetivos que se tomaron en las dos fases que tiene el proyecto, además de analizar las herramientas que se han utilizado, así como diferentes paquetes o incluso el mismo robot, con sus cualidades y funciones.

2.1. Objetivos del Proyecto

El objetivo de este proyecto ha evolucionado a medida que este avanzaba, ya que al principio el alumno no conocía las limitaciones y capacidades del Robot NAO. Por esto, el desarrollo del proyecto se divide en dos fases, las cuales tienen sus propios objetivos. La primera fase se centra más en aprender a utilizar herramientas básicas y crear un sistema sencillo que aprende y luego es capaz de utilizar lo aprendido. La segunda fase es una extensión de la anterior, utilizando herramientas ya conocidas se añaden algunas nuevas para crear un sistema que utiliza inteligencia artificial para responder adecuadamente.

2.1.1. Primera Fase

Como he mencionado anteriormente, esta primera fase se centra más en el aprendizaje de los recursos harware y software disponibles, en esta fase se tenía que aprender el funcionamiento del SDK del Robot NAO principalmente, el cual tiene sus limitaciones y no permite realizar ciertas funciones que se resultan imprescindibles en el sistema de diálogo natural. Por lo tanto, también se añadió como objetivo adicional la búsqueda y aprendizaje de herramientas de procesamiento de lenguaje natural en español para realizar reconocimiento de voz.

Por otro lado, el objetivo principal de esta fase era realizar un sistema capaz de aprender de forma literal pares de preguntas y sus respectivas respuestas. Para realizar esto era totalmente necesario realizar previamente un aprendizaje de las tecnologías necesarias.

En resumen, el sistema se ocuparía de aprender que cada pregunta y como responderla, es decir, que cada pregunta tendría una respuesta correspondiente, como se ilustra en la figura 2.

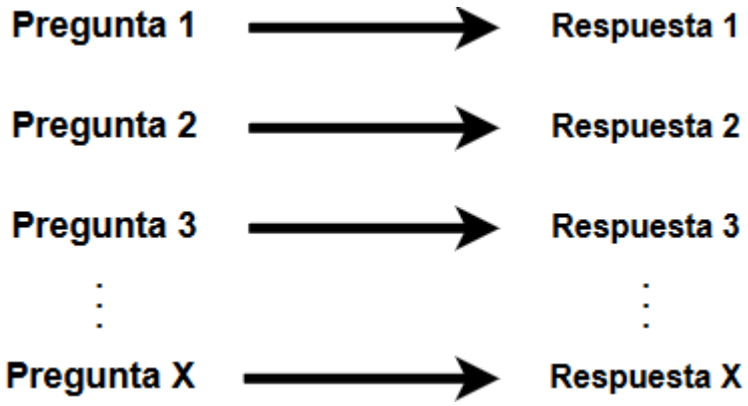

Figura 2: esquema inicial de pregunta-respuesta

2.1.1. Segunda Fase

La segunda fase, se centra en añadir cierta inteligencia al Robot NAO. Para ello, el objetivo era crear un sistema capaz de obtener un input de voz, identificar el significado de la pregunta y responderlo

en consecuencia. Para lograrlo era necesario atribuir a un tipo a cada conjunto de preguntas y respuestas, de tal manera que un conjunto de preguntas tendría un conjunto de respuestas que concuerdan en el mismo contexto (figura 3), por ejemplo, las preguntas de tipo “saludo” como “hola” o “buenos días” tengan como respuestas saludos similares.

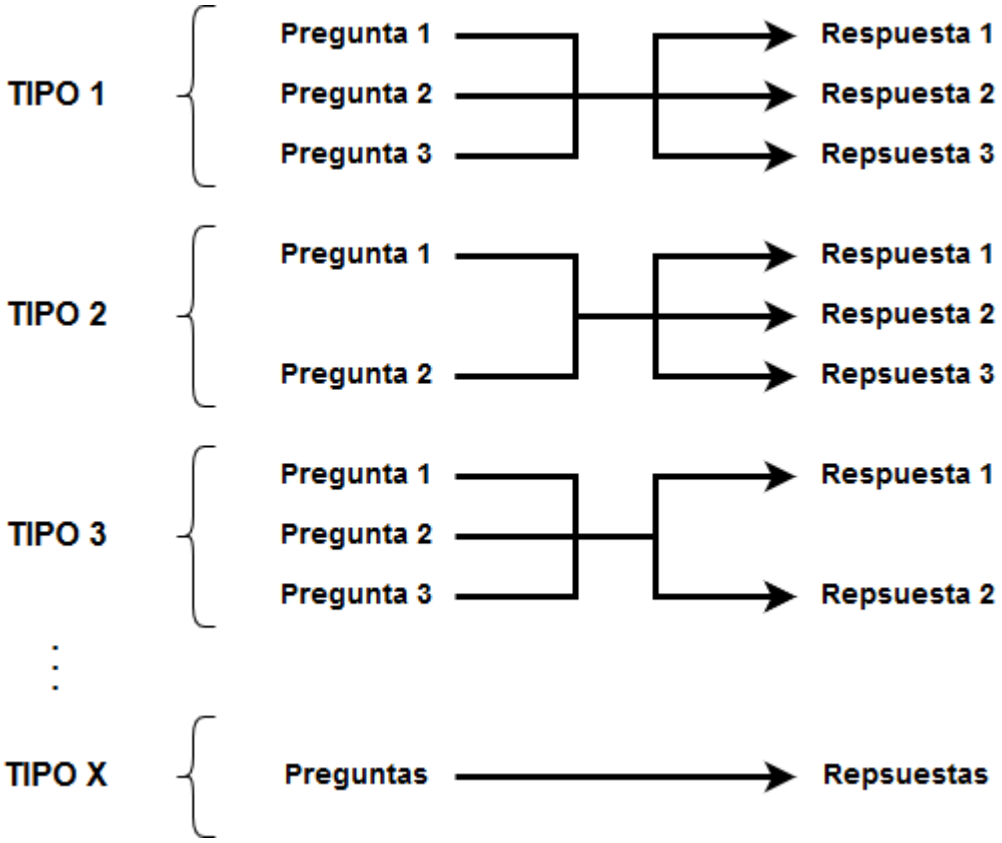

Figura 3 Categorización de preguntas equivalentes que dan lugar a respuestas similares para ser generadas de forma arbitraria causando efecto de sorpresa

2.2. Tecnologías y Herramientas utilizadas

Este proyecto se ha realizado en python, por lo tanto las librerías y paquetes que se mencionan a continuación pertenecen al entorno del lenguaje de programación python.

2.2.1. Robot Nao

Nao es un robot humanoide programable y autónomo, desarrollado por Aldebaran Robotics, una compañía de robótica francesa, la cual fue adquirida por SoftBank Mobile en 2013 y fue renombrada como SoftBank Robotics en 2015.

Figura 4: Robot Nao

2.2.1.1. Historia de Nao

El desarrollo de robots comenzó en 2004 con el lanzamiento del "Proyecto Nao" por la mano de Bruno Maisonnier, el cual en 2005 creo Aldebaran Robotics. Entre 2005 y 2007 se diseñaron 6 versiones diferentes de NAO. En 2007 Nao reemplazó al perro robot Aibo de Sony como el robot usado en RoboCupStandard Platform League (SPL), que es una competición internacional de fútbol para robots. En el año 2008 se lanzó la primera versión de producción del robot para participar en SPL, además de lanzarse en el mismo año una versión académica. También fue utilizado en la competición SPL de 2009, y el NaoV3R fue elegido como la plataforma para la competición de 2010.

En octubre de 2010 La universidad de Tokyo compro 30 robots Nao para un laboratorio, con la esperanza de convertirlos en asistentes de laboratorio activos. En diciembre del mismo año, se demostró que un robot Nao podía hacer un monólogo de comedia y se lanzó una nueva versión de este, con motores mejorados. En 2011, se anunció que saldría al código fuente para controlar el Nao y que sería código abierto. En 2013 Aldebaran fue adquirido por SoftBank Mobile por 100 millones de dólares estadounidenses.

Varias versiones de este robot se han lanzado desde 2008. La versión Académica fue lanzada para universidades y laboratorios con fines de investigación y educativos. Este salio en 2008 para instituciones y se hizo público en 2011. Nao ha tenido varias mejoras de la plataforma desde que se lanzó al mercado, incluyendo Nao Next Gen de 2011 y Nao Evolution de 2014. El Nao Next Gen tenía mejoras tanto de software como de hardware, con cámaras de gran resolución, mejorado la robustez, añadiendole un sistema anticolisiones y mayor velocidad a la hora de caminar. En el Nao Evolution, mejoraron la durabilidad y la síntesis de habla plurilingüe, se le mejoró la detección de formas y rostros mediante nuevos algoritmos, y se le mejoró la localización de sonidos gracias a usar cuatro micrófonos direccionales. La última versión de Nao es conocida como NAO[6] y recibe varias mejoras tanto de software como de hardware lo que facilita trabajar con el. Esta última versión es la que se ha utilizado en el proyecto.

2.2.1.2. NAOqi SDK

Naoqi[8] es el nombre de la librería de NAO, que viene con métodos para su control que permiten una gran variedad de implementaciones. En la página web de Softbank Robotics se puede descargar este SDK tanto para Python 2, C++ y Java, siempre y cuando se tenga un cuenta para poder acceder a dicha página. Este SDK cuenta con una gran cantidad de módulos, los cuales están diseñados para llevar a cabo distintas tareas, aún así se pueden combinar para realizar distintas acciones.

- Módulo “NAOqi Core”: Como dice el nombre, es módulo núcleo, con el cual se puede manejar el sistema del robot, controlar los recursos que este utiliza incluso crear nuevos módulos propios.
- Módulo “NAOqi Motion”: Este módulo se utiliza para controlar los movimientos del robot, desde movimientos de los brazos o adoptar posturas, hasta hacer que camine.
- Módulo “NAOqi Audio”: Su función principal está relacionado con el audio, para manejar los audios de entrada como de salida, este módulo cuenta con métodos para grabar sonidos, hacer hablar al robot, detectar sonidos, incluso reconocimiento de voz en varios idiomas.
- Módulo “NAOqi Vision”: Este módulo se usa para para controlar las cámaras del robot, nos permite grabar videos, hacer fotos, incluso detectar objetos, formas, movimiento y códigos de barras.
- Módulo “NAOqi PeoplePerception”: A diferencia de otros módulos no se centra en partes concretas del robot, utiliza varias de ellas para analizar a la gente que se encuentra alrededor del robot. Entre otras cosas, da la capacidad de detectar caras, personas sentadas, incluso es capaz de estimar características de las personas (como edad o sexo) basándose en su cara.
- Módulo “NAOqi Sensors”: Su función es dar información de varios sensores del robot, con los cuales puede obtener datos de su batería, infrarrojos, parachoques y botón del pecho entre otros, Además da la posibilidad de controlar los leds que tiene el robot.
- Módulo “NAOqi Trackers”: Permite al robot hacer seguimiento de diferentes objetivos.

8 http://doc.aldebaran.com/2-1/naoqi/sensors/dcm.html

- Módulo “ALDiagnosis”: Permite detectar si hay algún problema en el hardware.
- Módulo “DCM”: Se encarga de la comunicación de casi todos los dispositivos electrónicos del robot, excepto el sonido y cámaras.

La instalación de este SDK hay que hacerla de forma manual, bajando el archivo desde la página oficial, no se pueden utilizar gestores de paquetes como pip o conda para hacerlo.

2.2.1.3. Componentes y capacidades de NAO

El Robot NAO[9] cuenta con 25 grados de libertad, lo que se traduce en una autonomía de movimiento sorprendente, se desplaza gracias a sus extremidades articuladas en cualquier dirección, gira y mantiene posiciones naturales.

Sus manos prensiles le permiten sujetar objetos y desplazarlos de un lado a otro, es capaz de levantar objetos de hasta 600 gramos.

Además de esto, NAO puede comunicarse a través de Wifi o Ethernet. Puede ser utilizado tanto bajo protocolo WPA como WEP para que se conecte en lugares de trabajo u hogares.

A través de los cuatro micrófonos que contiene el hardware del robot, Nao es capaz de reconocer sonido casi desde cualquier ángulo y es capaz de comunicarse, ahora mismo reconoce y habla un total de 19 idiomas.

NAO tienes dos cámaras que le permiten analizar visualmente su entorno. A través de ellas detecta objetos y reconoce personas. Las cámaras no sólo permiten ver objetos o persona, sino que es capaz de desplazarse por el entorno con seguridad ya que una de las cámaras está situada en la boca para identificar lo que hay debajo de él.

NAO tiene bumpers o “parachoques” en los pies, lo que le permite detectar obstáculos. Además el diseño de sus pies le permite levantarse con facilidad si se cae.

Además de esto, el robot cuenta con nueve sensores táctiles, telémetro, medidor de inercia. dos sensores ultrasónicos y ocho sensores de presión.

2.2.1.4. Poner en marcha a NAO

Antes de encender el robot, hay un par de cosas que hay que tener en cuenta. En primer lugar, sujetarlo de la cintura es la mejor manera de manejarlo y evitar que pille el dedo a alguien. A continuación, es importante colocarlo en la postura “Sentado” o “Agachado”. Una vez está en una de estas dos posturas, se presiona una vez el pecho del robot, y hay que esperar de dos a cinco minutos para que se inicie, cuando está listo suena un ruido que suena como “GNUK GNUK” y el robot se coloca de pie.

9 https://aliverobots.com/nao/

Si se presiona el botón del pecho durante tres segundos este vuelve a realizar el sonido “GNUK GNUK” y se apaga forzosamente, volviendo a la postura “Agachado”. Si se realiza este apagado se pierden todos los datos.

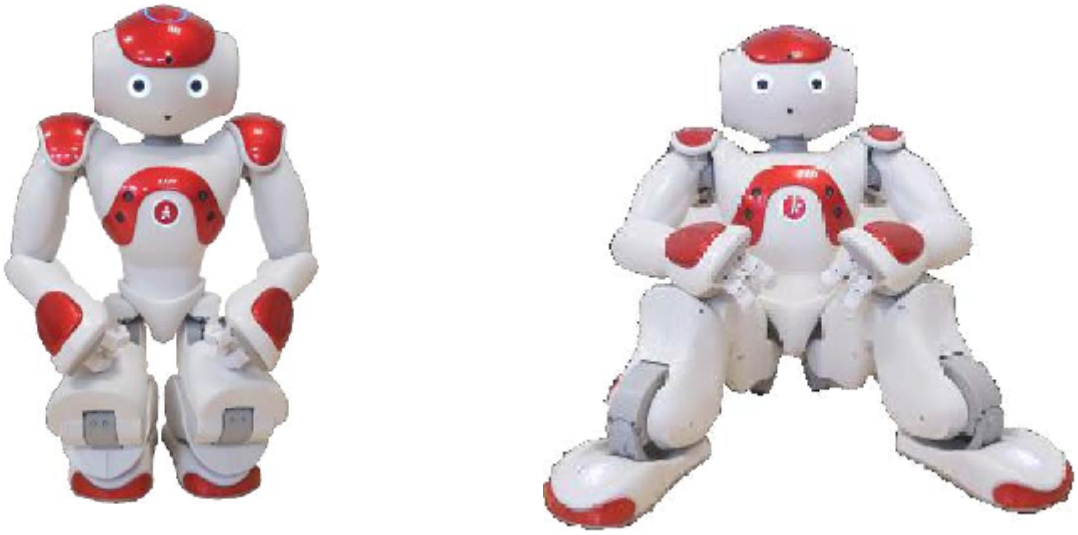

Figura 5: posturas de agachado y sentado

Cuando está encendido, el color del botón del pecho indica diferentes cosas, en el caso de que el color sea verde, significa que no hay ningún problema, si es amarillo es un aviso de alerta y si el color es rojo significa que ha ocurrido un error. Pulsando el botón del pecho una sola vez el robot comunica el IP de internet al que está conectado y en el caso de que el botón este amarillo o rojo comentará una notificación sobre lo que sucede.

En la parte trasera de su cabeza tiene una tapa, si se quita dicha tapa, deja al descubierto un puerto USB y otro para conectar el cable Ethernet.

En la parte trasera de su tronco hay un conector que sirve para enchufar el cargador del robot.

Sus dedos son muy frágiles y se recomienda no tirar de ellos ya que se pueden romper fácilmente. Mientras el robot está encendido tampoco hay que manejar manualmente ninguna de sus extremidades o partes móviles, ya que esto podría dañar sus motores, en cambio, cuando está apagado se pueden manejar libremente. Por último, pero no menos importante, se recomienda no dejarlo al borde de las mesas u otras superficies por el riesgo que supone que se caiga.

2.2.2. paramiko

Paramiko[10] es una implementación en python (2.7, 3.4+) del protocolo SSHv2, que provee la función de cliente como la de servidor. A pesar de que utilice una extensión de Python C para criptografía de nivel bajo, Paramiko en sí misma es una interfaz pura de Python sobre conceptos de redes SSH.

10 http://www.paramiko.org/

Aunque no es un paquete que se haya utilizado mucho en el desarrollo del proyecto, ha sido imprescindible para llevar a cabo este proyecto, ya que hacía falta hacer conexión con el robot para compartir información. Tanto las grabaciones y el documento que guarda el conjunto de preguntas y respuestas, el cual debe ubicarse en el robot para que este pueda utilizarlo.

paramiko puede instalarse mediante pìp sin ningún problema y se puede encontrar en GitHub y PyPI.

2.2.3. spaCy

spaCy[11] es una librería para Procesamiento de Lenguaje Natural disponible para Python y Cython. Se basa en las últimas investigaciones y se diseñó desde el primer momento para ser utilizado en productos reales. spaCy viene con modelos estadísticos pre-entrenados y vectores de palabras, y actualmente admite tokenización para más de 49 idiomas.

Cuenta con modelos de redes neuronales convolucionales de alta tecnología para el etiquetado, el análisis y el reconocimiento de entidades nombradas, así como una integración de aprendizaje profundo y fácil. Es un software comercial de código abierto, publicado bajo la licencia MIT.

spaCy es una de las librerías más importantes en este proyecto, su ayuda a sido imprescindible para poder avanzar en el proyecto y ha sido utilizada para tokenizar y lematizar las frases en español. Puede instalarse de forma sencilla utilizando pip o conda. También puede encontrarse en GitHub y PyPI

2.2.4. Gensim

Gensim[12] es una librería gratuita de Python (2.7, 3.5 y 3.6) diseña para extraer de forma automática temas semánticos de los documentos, de la manera más eficiente.

Gensim está diseñado para procesar textos digitales sin estructurar, es decir, no hace falta tener textos estructurados de formas específicas.

Gensim cuenta con una gran cantidad de algoritmos como Word2Vec, Doc2Vec, FastText, Latent Semantic Analysis, Latent Dirichlet Allocation entre otros, los cuales descubren automáticamente la estructura semántica de los documentos examinado los patrones de coocurrencia estadística dentro de un campo de documentos de entrenamiento. Estos algoritmos no son supervisados, lo que significa que no es necesario intervención humana, solo se necesita un corpus de documentos de texto sin formato.

Una vez se encuentran estos patrones estadísticos, cualquier documento de texto sin formato se puede expresar en la nueva representación semántica y consultar la similitud tópica con otros documentos.

---

11 **https://github.com/explosion/spaCy**

12 https://radimrehurek.com/gensim/intro.html

Al igual que spaCy, esta librería ha sido imprescindible a la hora de llevar a cabo el proyecto, ya que es la que se ha utilizado para entrenar un modelo y poder estimar el significado de las preguntas que se le hacen al sistema. Puede instalarse mediante conda y pip de forma sencilla.

Gensim depende de algunas otras librerías y a pesar de no haberlas usado directamente, son necesarias para poder utilizar Gensim:

- NumPy >= 1.11.3
- ScyPy >= 0.18.1
- Six >= 1.5.0
- smart_open >= 1.2.1

2.2.5. speech_recognition

speech_recognition[13], como el nombre indica, es una librería de Python (2.6,2.7 y 3.3+) para llevar a cabo reconocimiento de voz, es decir, sirve para convertir mensajes de audio a texto (SpeechToText). Si se quiere llegar a utilizar todas sus funcionalidades hay que instalar otras librerías adicionales:

- Pyaudio (siempre y cuando se necesite un input de micrófono)
- PocketSphinx (para utilizar el reconocedor de Sphinx)
- Librería de cliente de la API de Google(si se quiere utilizar Google Cloud Speech API)

Esta librería ha sido utilizada para convertir audio en texto para luego poder procesar dichos textos. Se puede instalar fácilmente mediante pip.

2.2.6. PyCharm

Pycharm[14] es un entorno de desarrollo integrado (Integrated Development Environment, IDE) utilizado para programación de computadores, sobre todo para el lenguaje de Python. Está desarrollado por JetBrains, una compañía checa. Proporciona análisis de código, un debugger gráfico y un comprobador de unidades integrado. PyCharm es multiplataforma, con versiones disponibles para Windows, macOS y Linux. La Community Edition se publica bajo la Licencia de Apache, y también existe la Professional Edition, publicada bajo licencia del propietario.

PyCharm es una herramienta muy útil para desarrolladores, ya que facilita programación y tiene un sistema para instalar paquetes de forma cómoda.

A la hora de crear un nuevo proyecto, o en cualquier momento del desarrollo del proyecto, el usuario puede crear nuevos entornos con distintos intérpretes, los cuales podrá ir cambiando siempre y cuando lo necesite. Para esto se puede elegir que tipo de entorno se quiere crear, un entorno python, de conda o de pip (figura 6).

13 https://github.com/Uberi/speech_recognition#readme
14 https://www.jetbrains.com/pycharm/

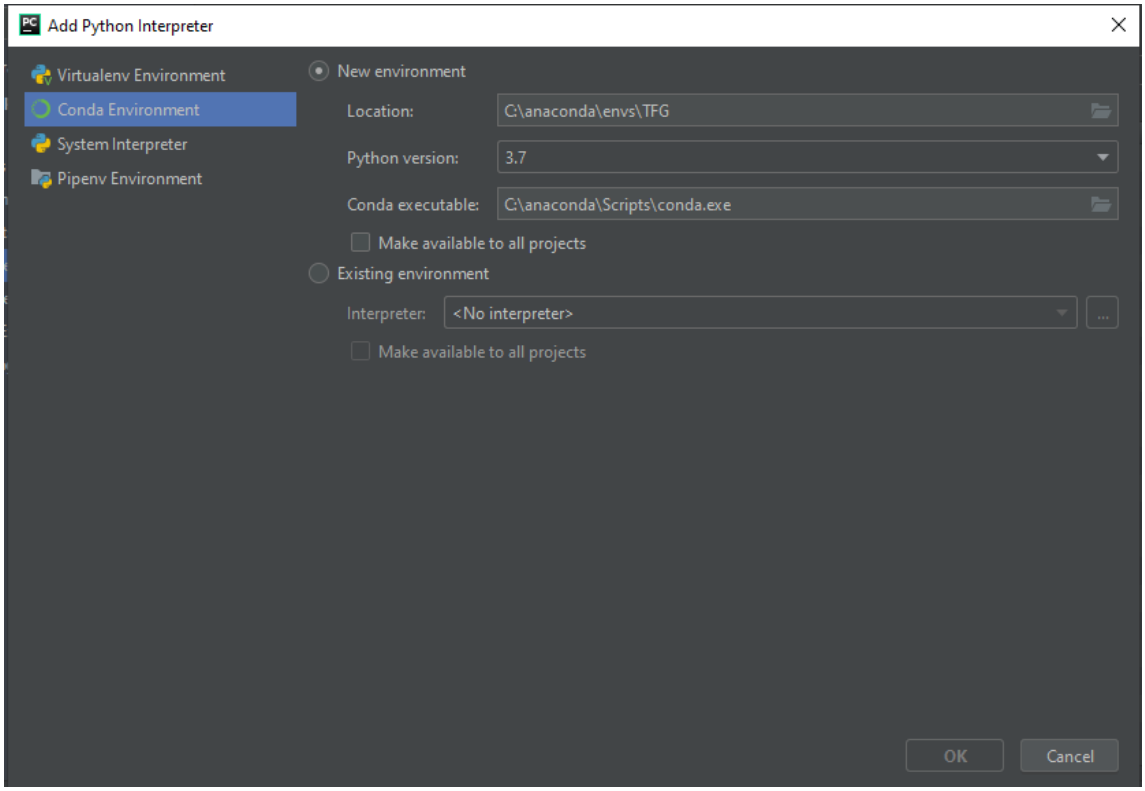

Figura 6: Selección del entorno

Si se necesita instalar un paquete se puede hacer de forma rápida, Seleccionando File→ Settings en la barra de herramientas, se abrirá una ventana, en esta parecerá nuestro intérprete actual y las distintas librerías que tiene instaladas, dandole al "+" de la derecha se puede instalar los paquetes que hagan falta, con "-" se pueden desinstalarlos y con "▲" se pueden actualizar. Además, a la hora de instalarlos, se puede seleccionar la versión deseada (figura 7).

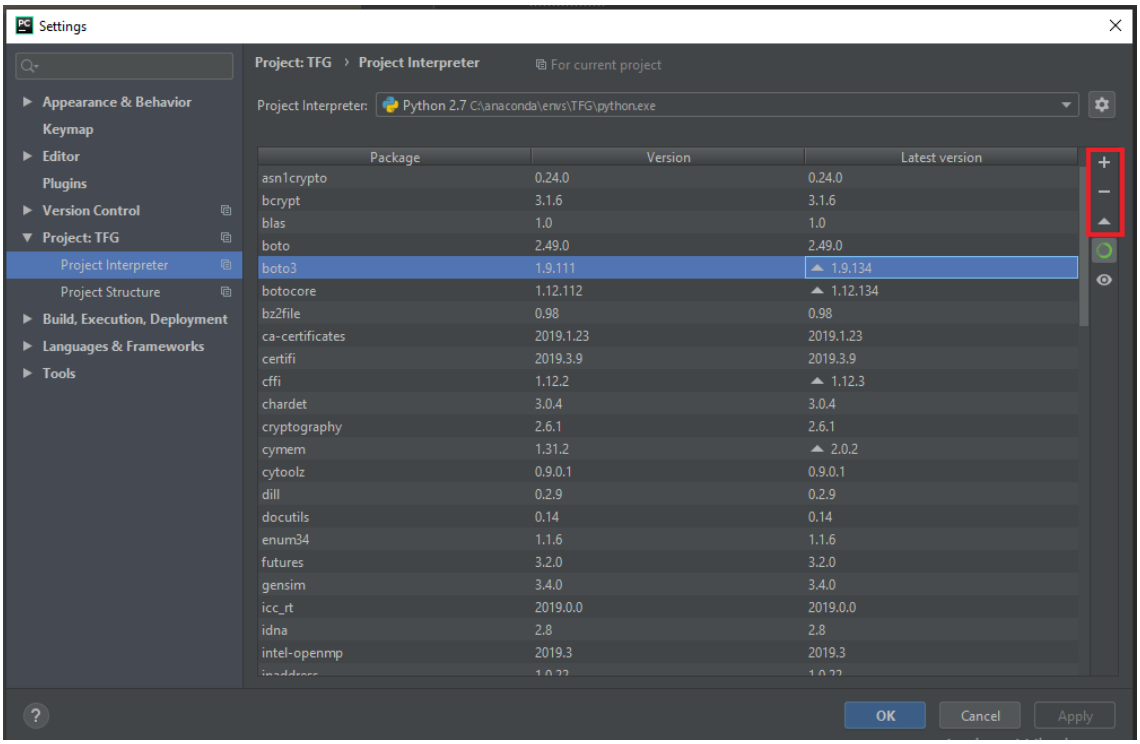

Figura 7: manejo de paquetes

Por último, hay que comentar que proporciona ayudas a la hora de programar (figura 8), ya que analiza el código escrito hasta el momento y avisa de errores que hay en este.

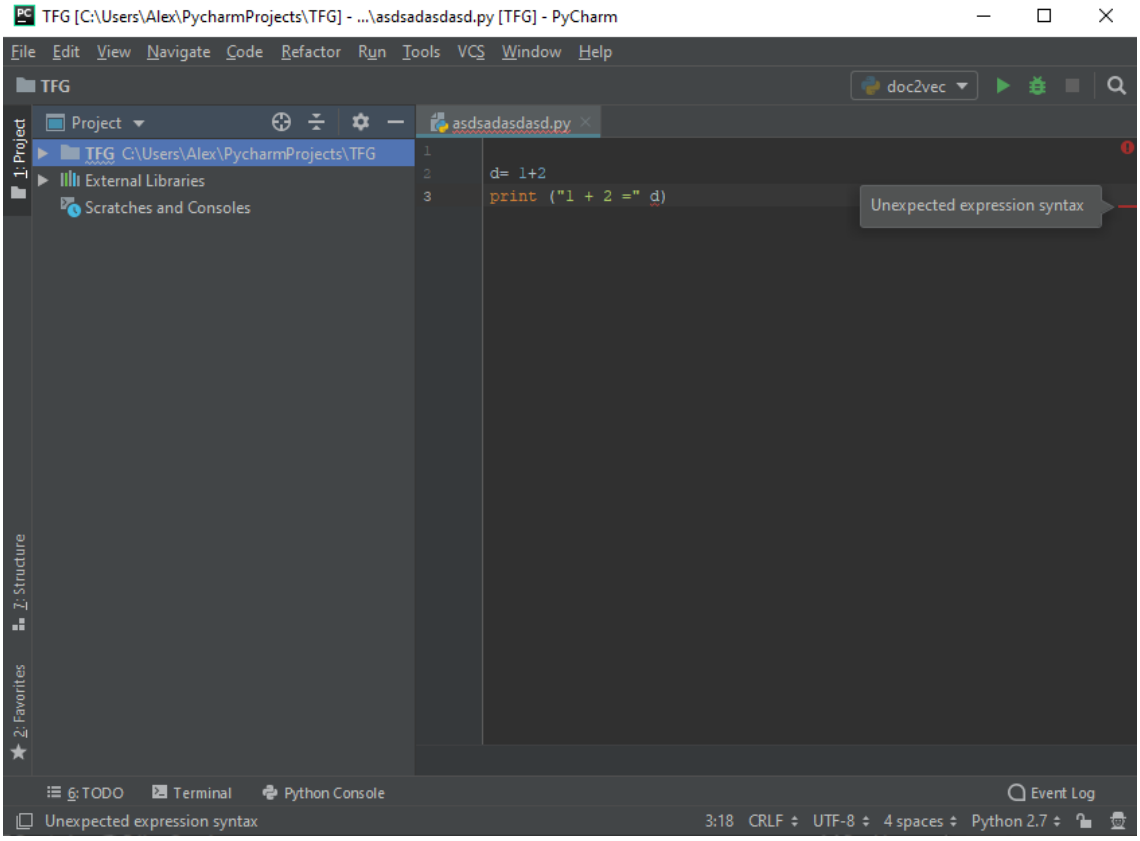

Figura 8: Advertencia de error sintactico

2.2.7. Doc2Vec

Doc2Vec[15] es un concepto que se ha utilizado desde de la librería gensim que ha servido para entrenar el modelo que detecta el tipo de las preguntas que se le realizan al robot. Doc2Vec es una herramienta de procesamiento de Lenguaje Natural para representar documentos como vectores y es una generalización del método Word2Vec. Para entender cómo funciona Doc2Vec es necesario saber cómo funciona Word2Vec previamente.

Como dice el nombre, word2vec se utiliza para generar vectores de representación a partir de palabras. Por lo general, cuando se quiere realizar un modelo usando palabras, etiquetarlas y codificarlas es una forma plausible de hacerlo, pero al hacerlo de esta manera, las palabras pierden su significado, de tal manera que no se tiene en cuenta la relación que pueden tener dos palabras entre sí. Para solucionar este problema, word2vec realiza una representación numérica para cada palabra, que puede capturar las relaciones entre palabras. Estos pueden encapsular relaciones como sinónimos, antónimos o analogías entre otros.

15 https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Para saber como funciona word2vec, hay que saber que se compone de la combinación de dos algoritmos, Continuous Bag of Words (CBOW) y Skip_Gram.

Continuous Bag of Words crea una ventana deslizante alrededor de la palabra actual para predecirla del contexto. Los vectores que representan palabras similares están cerca en diferentes métricas de distancia y encapsulan relaciones numéricas.

El algoritmo Skip gram es todo lo contrario a CBOW, en lugar de predecir una palabra a partir de su contexto se intenta predecir el contexto a partir de una palabra. Skip gram es mucho más lento que CBOW, pero da mejores resultados al usarse palabras poco frecuentes.

A diferencia de word2vec que realiza una representación numérica de cada palabra, el objetivo de doc2vec es crear una representación numérica de un documento sin importar su longitud.

Para conseguir esto, se le añadió otro vector al modelo word2vec. Este vector es un vector de características. Por lo tanto, cuando se entrenan los vectores de palabras, el vector de documentos también se entrena y al final del entrenamiento contiene una representación numérica del documento.

Mientas que los vectores de palabras representan el concepto de una palabra, el vector de documento intenta representar el concepto de todo el documento.

2.2.7. Otras herramientas

Además de las herramientas que se han mostrado anteriormente, las cuales han sido utilizadas en el código final o han sido muy importantes para llevarlo a cabo, se han utilizado otras, que a pesar de no haber sido muy relevantes, han servido para hacer pruebas o han sido descartadas por no satisfacer ciertas necesidades, a pesar de ser buenas herramientas que podrían servir para otros proyectos.

2.2.7.1. Choregraphe

Choregraphe[16] es una aplicación de escritorio multiplataforma que permite crear animaciones, diálogos y comportamientos para un robot, o simularlos en uno virtual. Además, permite monitorizar y controlar dicho robot. Por último, se puede combinar con el código python para obtener una experiencia más enriquecedora. En resumen, Choregraphe permite crear aplicaciones con diálogos, servicios comportamientos, interacciones con personas, bailes o envío de correos electrónicos entre otras capacidades, las cuales se pueden realizar sin escribir ni una línea de código, ya que permite una programación de forma gráfica (figura 9), mediante unión de distintos módulos. Esta herramienta se utilizó al principio del proyecto para testear las capacidades del robot NAO.

16 http://doc.aldebaran.com/2-1/software/choregraphe/choregraphe_overview.html

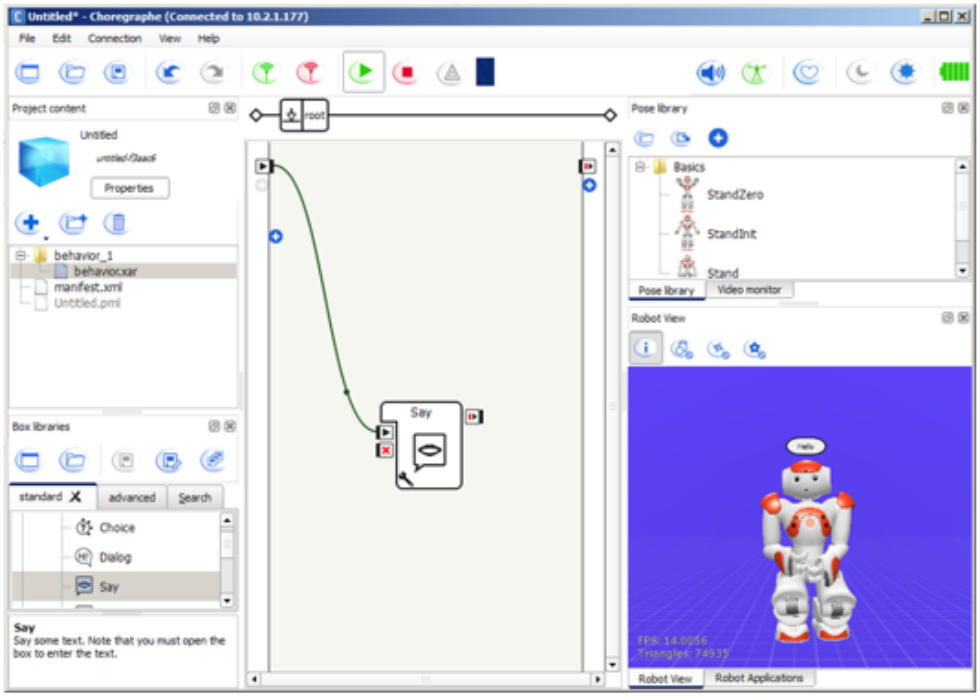

Figura 9: Interfaz del Choreographe

2.2.7.2. Dialogflow

Dialogflow[17] es una herramienta que permite crear herramientas de conversación sin necesidad de realizar procesamiento de lenguaje natural. La cual se centra en crear varios conjuntos de preguntas y respuestas, de tal manera que un conjunto de preguntas tenga como respuestas las respuestas del mismo conjunto. Básicamente, crea un modelo para responder a preguntas de diferentes maneras.

Al principio se pensó en añadir esta herramienta al proyecto para llevar a cabo la segunda fase y tener ya entrenado un conjunto de preguntas y respuestas, pero a decir verdad fue descartado rápidamente, ya que ofrecía lo mismo que el sistema de diálogo del SDK de NAO.

A pesar de no haberse usado directamente, esta herramienta permite crear un conjunto de preguntas (sin respuestas) más comunes a la hora de crear bots conversacionales, las cuales se utilizaron para crear el corpus de preguntas.

17 https://dialogflow.com/

2.2.7.3. NLTK

NLTK (Natural Language Toolkit)[18] es una plataforma para crear programas que trabajan con lenguaje natural. Esta herramienta ofrece simples interfaces además de unos 50 corpus y recursos léxicos como Wordnet, además un conjunto de librerías de procesamiento de texto para clasificación, tokenización, derivación, etiquetado, análisis y razonamiento semántico.

A pesar de tratarse de una herramienta muy útil, fue descartada porque los servicios que ofrecía para español no eran los suficientemente buenos para utilizarlos, pero si el trabajo se hubiese realizado para oraciones en inglés, probablemente habría resultado una herramienta muy eficiente.

18 https://www.nltk.org/

3. Desarrollo del Proyecto

El desarrollo del proyecto ha sido complicado ya que me ha sido necesario investigar sobre tecnologías que no había utilizado antes así como encontrar herramientas compatibles con mi sistema y sus restricciones. El proyecto ha evolucionado a medida que se trabajaba en él y podría dividirse en dos fases, la primera sería la que corresponde a cumplir los objetivos establecidos al principio del desarrollo del proyecto y la segunda fase corresponde a lo que vino después de cumplir los objetivos iniciales, lo realizado en la primera fase parecía demasiado simple para dejarlo ahí, por lo cual, se decidió extender el desarrollo y añadir funciones que no se habían pensado al principio. A continuación, se analizarán ambas fases por separado y se expondrán las dificultades que han habido, las decisiones que se han tenido que tomar y cómo se han llevado a cabo.

3.1. Primera Fase

3.1.1. Desarrollo y Dificultades

El primer paso de esta fase del proyecto era preparar el entorno con el que se trabajaría (en el ordenador que se había facilitado para la realización del TFG) y familiarizarse con las herramientas disponibles. Por eso mismo, en las primeras semanas me centré en estudiar el manual del robot NAO, para conocer cómo utilizarlo adecuadamente y no dañarlo, ya que tiene partes sensibles que se pueden romper con facilidad. Con el fin de familiarizarme con las capacidades y limitaciones del robot NAO, fue de ayuda instalar el programa Choregraphe, el cual permite programar el robot sin necesidad de saber programar, da muchas facilidades que permite programar de forma gráfica uniendo distintos módulos para generar un programa completo. A la hora de instalarlo daba problemas en el sistema operativo de linux, y fue necesario instalar windows 10 para probar Choregraphe. Al mismo tiempo, tuve que informarme sobre el SDK de python, los métodos y módulos que tenía y cómo utilizarlos.

Gracias a Choregraphe, se hicieron las primeras pruebas del reconocimiento de voz que tenía el NAO, las cuales funcionaban correctamente pero mostraron un problema, a la hora de hacer el reconocimiento de voz mediante Choregraphe o el SDK de NAO, el sistema espera que se le pasen un conjunto de palabras que tenía que esperar, denominado “vocabulario”. En resumen, en sistema era incapaz de identificar palabras que no estuviesen en el vocabulario, por lo tanto no podía convertir cualquier mensaje de voz en texto. El objetivo principal que se tenía para esta fase era realizar un sistema capaz de aprender de lo escuchado y responder en consecuencia, y el hecho de necesitar saber cuales iban a ser las palabras o frases entrantes constituían un problema, ya que no se podían recibir oraciones desconocidas como entrada.

El siguiente paso fue buscar herramientas adicionales para hacer la conversión speech to text, la solución que se encontró fue combinar las librerías speech_recognition y paramiko, de manera que el primero NAO graba el audio utilizando el SDK, después se comparte el archivo grabado mediante paramiko, y por último se transcribe en español mediante el paquete speech_recognition.

Una vez se completada la búsqueda y aprendizaje de las herramientas necesarias para realizar esta fase, comenzó la implementación. Dicha implementación se produjo según lo previsto, no hubo demasiados problemas a parte de un pequeño problema con carácteres especiales en español como la ‘ñ’ o letras con tilde ‘á’, ’é’, ’í’, ’ó’ y ’ú’ que se soluciono mediante cambio de la estructura de los strings.

Cuando un usuario desea cambiar de fase puede utilizar comandos por voz, el robot tiene un sistema que guarda cual es la última palabra del vocabulario que ha escuchado, por lo tanto, se utiliza para elegir cual es el cambio de estado que se ha seleccionado. Esta herramienta ha sido muy útil y funcionaba sin problemas en la primera iteración, pero en la segunda iteración daba problemas porque todavía mantenía guardado la última palabra que se había utilizado la última vez, por lo tanto volvía a entrar en la misma selección y tomé la decisión de que al empezar y terminar de usar esta herramienta siempre se reiniciaría para evitar problemas.

3.1.2. Implementación

El sistema que se consiguió en esta fase se compone de 4 estados, INITIAL_STATE, DIALOGUE_STATE, LEARNING_STATE y SHUTDOWN_STATE. El cambio de un estado se realiza en la mayoría de casos mediante interacción por voz, es decir, hay que decir palabras concretas en estados concretos para cambiar de estado (figura 10).

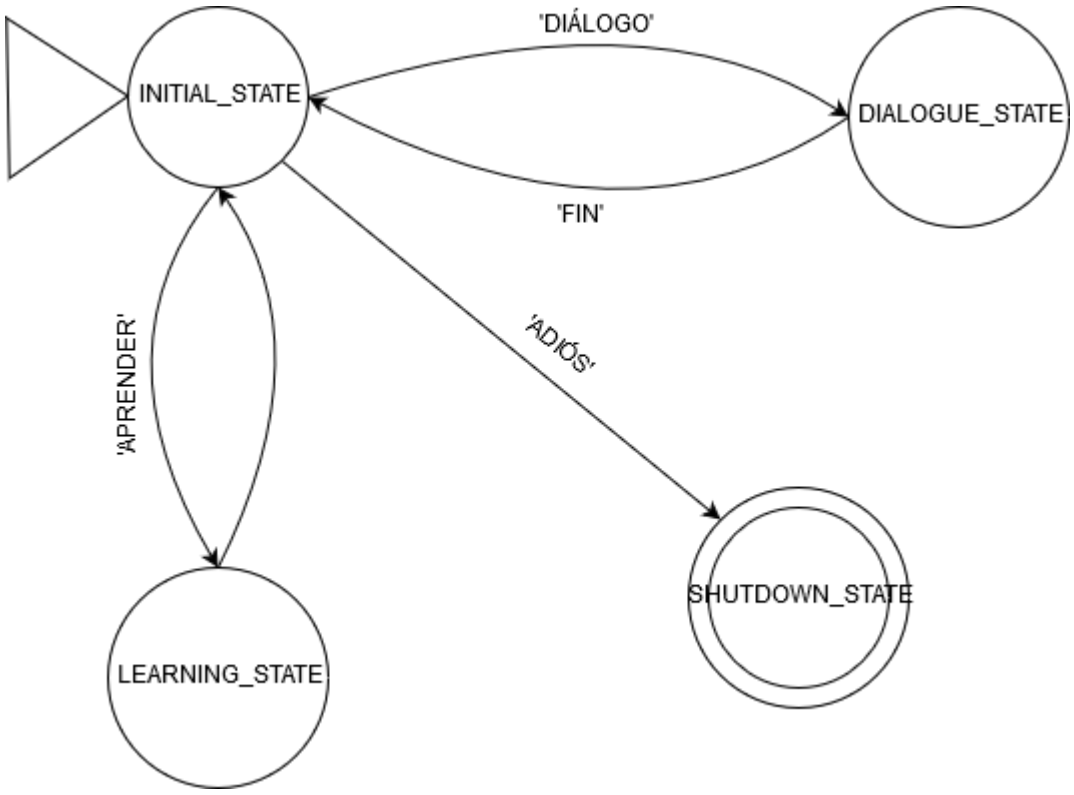

Figure 10: diagrama de estados y transiciones del sistema diseñado en la primera fase

El método principal del programa hace una configuración inicial para establecer comunicación con el robot NAO, para ello es necesario saber su IP y su puerto, una vez hecho esto, llama al método “main”, que es el que se encarga de manejar el cambio de estados y qué hacer en cada uno de ellos.

A continuación se analizarán todos los métodos del fichero “NAODialog.py” que se implementaron en esta fase.

3.1.2.1 Método “main”

Lo primero que hace este método es llamar al método “downloadfile” para descargar del robot las preguntas-respuestas que tiene aprendidas y las añadirá a una matriz con el método “creatematrix”, después inicializa algunas variables que se necesitarán más adelante y se establece INITIAL_STATE como estado actual. A continuación, entra en un bucle que se mantendrá siempre y cuando es estado no sea SHUTDOWN_STATE, lo que asegura que en la primera iteración siempre se ejecutará. El siguiente paso es realizar una acción u otra dependiendo de estado en el estado en el que se encuentre el sistema.

Si el estado es INITIAL_STATE se llamará al método “ChangeState”, en el caso de estar en LEARNING_STATE se llamará 2 veces a el método “getrRecordedText”, para obtener una pregunta y su respuesta y guardarlos en una matriz (la cual tendrá todas las preguntas y respuestas) y actualizará el contador que guarda la cantidad de preguntas-respuestas que hay. Si el estado es DIALOGUE_STATE, se llamará a los métodos “createfile”, “uploadfile” y “dialogue”, en este orden. Tanto al acabar LEARNING_STATE como DIALOGUE_STATE se establecerá INITIAL_STATE como siguiente estado. Por último, en el caso de que el estado fuese SHUTDOWN_STATE se termina el bucle, se llamará a los métodos “createfile” y “uploadfile”, se apaga el robot mediante el método “ShutDown” y termina el proceso.

3.1.2.2. Método “downloadfile”

Este método se encarga de descargar el archivo “dialogo.top” que está guardado en el robot NAO, en el cuál se guardan los conjuntos de preguntas y respuestas aprendidos y se utilizan para hablar con el. Este método no obtiene ningún valor de entrada y no devuelve nada. Lo único que hace es conectarse mediante ssh (gracias a paramiko) al robot NAO y descargar el archivo “dialogo.top”.

3.1.2.3. Método “creatematrix”

Este método abre el fichero “dialogo.top” descargado del robot y guarda sus valores en una matriz, además devuelve el número de líneas que tiene el fichero, el cual siempre será el número de preguntas-respuestas más dos, ya que las dos primeras líneas del archivo siempre serán las mismas. Por lo tanto el método guardará en una matriz las preguntas y respuestas por parejas.

3.1.2.4. Método “ChangeState”

El objetivo de este método es decidir cuál será el siguiente estado y siempre se ejecutará desde el estado INITIAL_STATE. El método no admite ningún input y devuelve como salida cuál será el siguiente estado. El método comienza con establecer el vocabulario que se utilizará para elegir estado, mediante las palabras “Diálogo”, “Aprender”, y “Adiós”, además se subscribe a ciertos eventos del SDK de NAO para poder detectar que se dicen estas palabras. A continuación se iniciará un bucle que no acabará hasta que escuche una de estas palabras. Por último, dependiendo de cuál de ellas se haya interpretado devolverá un estado u otro, SHUTDOWN_STATE para “Adiós”, LEARNING_STATE para “Aprender” y DIALOGUE_STATE para “Diálogo”.

3.1.2.5. Método “getrRecordedText”

Este método sirve para obtener el texto de cualquier frase, es decir, se encarga de hacer Speech to Text. Recibe como input valores de parámetros necesarios para conectarse con NAO (IP, usuario y contraseña) así como un atributo llamado “text” que guarda una oración, la cual se sirve para diferenciar si el robot pide una pregunta o una respuesta, ya que le preguntará al usuario sobre la pregunta o respuesta. La salida es la frase que ha escuchado del usuario en texto.

Tras establecer conexión con NAO y subscribirse a los eventos necesarios comienza un bucle del cual no se podrá salir hasta que el sistema haya entendido lo que el usuario haya dicho y lo haya convertido en texto. A continuación el robot se pondrá a grabar continuamente al usuario hasta que deje de escucharlo, en el caso de que el usuario no haya dicho nada en 2 segundos desde que empezó a grabar, vuelve a empezar a hacerlo. El siguiente paso, es obtener la grabación que se ha guardado en NAO, para ello se utiliza conexión ssh de paramiko.

Por último, se transcribe la frase grabada mediante a speech_recognition, en el caso de que de error, el robot expresa que no ha entendido la frase y se repite el proceso. En caso de que no se produzca error y haya entendido algo, el robot le dice al usuario lo que speech_recognition ha interpretado y le pregunta si es correcto. Para comprobar la respuesta de tipo “sí” y “no” que de el usuario, se ejecuta el método “SioNo” y dependiendo de la respuesta terminará el método o volverá a preguntar por la oración.

3.1.2.6. Método “SioNo”

Las preguntas de respuesta “sí” o “no” se utilizan más de una vez en el programa, por lo tanto, este método creó con el fin de no implementarlo más de una vez. No tiene ningún input pero devuelve como salida el en número 1 en caso de escuchar “sí” y 0 en caso de escuchar “no”. Se trata de un método muy simple, en el cual se establecen las palabras “sí” y “no” en el vocabulario y el sistema estará en bucle hasta que escuche una de las dos.

3.1.2.7. Método “createFile”

Este método se utiliza para crear un fichero a partir de los datos que hay en la matriz de preguntas y respuestas. Obtiene como entrada el número de líneas que tendrá el fichero que se va a crear, el cual siempre es el número de parejas pregunta-respuesta más dos, ya que la primeras dos líneas son siempre las mismas.

3.1.2.8. Método “uploadfile”

Este método es muy similar a “downloadfile”, la diferencia es que en lugar de descargar el archivo “dialogo.top” del NAO en el ordenador, se hace al revés, es decir, se sube este fichero al NAO para que pueda utilizarlo al hacer el diálogo. Para ello, se utiliza paramiko para hacer conexión ssh.

3.1.2.9. Método “dialogue”

Este método se utiliza para establecer conversación con el robot, es decir, en este estado se pone en práctica lo aprendido en el estado LEARNING_STATE, utilizando las frases que están en “dialogo.top”.

Básicamente, lo que este método hace es cargar el archivo “dialogo.top” y activarlo. Mientras está activado, el robot responde a todo lo que el usuario diga usando las frases de dicho documento, si el usuario usa una frase que el robot no conoce, da la respuesta que el sistema considera más parecida, lo cual se decide mediante una puntuación. El fichero de preguntas y respuestas siempre debe contiener la palabra “terminar”, la cual se establece como vocabulario y en el momento que la escucha, se finaliza el diálogo, por lo tanto se desactiva el documento y no si vuelve a tener en cuenta.

3.1.2.10. Método “Shutdown”

Este método obtiene como entrada el IP, el nombre de usuario y la contraseña necesarias para conectarse con NAO, que se necesita para llamar al método “shutdown” que apaga el robot.

3.2. Segunda Fase

3.2.1. Desarrollo y Dificultades

En principio, una vez acabada la primera fase del proyecto no se tenía planeado continuar, pero visto el trabajo realizado y las posibilidades que había, se decidió extender el proyecto y realizar algo más sofisticado usando machine learning. Mientras pensaba como insertar machine learning en el sistema, realizé una mejora en el código: encendía los leds de los ojos del robot en el momento que espera que le hablen, de esta manera quedaba más claro para el interlocutor cuando estaba escuchando la señal audio o simplemente esperando.

Finalmente se decidió implementar un chatbot entrenado mediante machine learning, el cual sería capaz de procesar una mayor cantidad de frases en comparación a la primera fase, la cual se limita a frases concretas. Una vez decidido esto, se tenía que encontrar una forma adecuada de realizarlo, por eso se empezó a investigar acerca del tema, empezando por Dialogflow, el cual da ayudas a la hora de realizar chatbots. Tras ver que no servía para nuestros propositos, ya que no ofrecía un aprendizaje y se reducía a lo mismo que realizaba el robot NAO en la primera fase del proyecto, se descartó por completo. De todas formas, una pequeña parte de la ayuda que daba esta herramienta acabó siendo útil.

Antes de realizar un entrenamiento de un modelo, es imprescindible realizar un preproceso de las frases de entrada. Como preproceso, se pensó en realizar una tokenización de las frases, una lematización y por último eliminación de palabras que no dan información relevante para realizar una clasificación apropiada. Por ello, era necesario encontrar herramientas tanto para tokenizar y lematizar palabras en español.

La primera herramienta que probé para realizar esto fue NLTK, la cual permite realizar tanto la lematización y tokenización en diferentes idiomas además de dar herramientas útiles para procesamiento de lenguaje natural, el problema es que no contaba con un lematizador en español, lo cual era muy importante, ya que gracias a esta herramienta se pueden generalizar ciertas palabras (gracias al lematizador, las palabras “hablaba” y “hablaré” se convierten en “hablar”, lo que facilita su proceso posterior).

El siguiente paso, fue buscar otra herramienta que ofreciera una lematización apropiada, la respuesta a esta necesidad fue “spacy”, una librería que cumple las mismas funciones que necesitaba de NLTK, pero a diferencia de la primera, permite usar lematización en español, por lo tanto dejé de lado NLTK para centrarme en spacy. Desafortunadamente, en el ordenador que se estaba utilizando para realizar el proyecto, spacy no se instalaba correctamente y no funcionaba, por lo tanto decidí cambiar el entorno de python por uno de conda, lo que me obligó a instalar nuevamente todas las librerías necesarias.

Finalmente, contaba con unas herramientas para realizar el preproceso del corpus de preguntas y me faltaba elegir un método para crear un modelo. El método elegido fue doc2vec, que se podía utilizar desde la librería gensim. Con las herramientas necesarias para realizar el proyecto, comencé con la implementación de sistemas, además de la creación de un corpus de preguntas, las cuales se clasificarían por tipo de pregunta. Una vez realizado el corpus y el código necesario para entrenar el modelo, se contaron las repeticiones de las palabras, las cuales se ordenaron por frecuencia de aparición. De esta manera se podía saber que palabras aparecían en demasiadas frases por lo que no podrían ser decisivas a la hora de decidir de qué tipo sería la frase entrante. Las eliminamé para que no produjeran problemas.

A continuación, se procedió a mejorar el modelo que se había creado , para ello se analizaron las variables que utilizaba y se cambiaron sus valores para observar qué cambios se producen al cambiar una u otra. De esta manera se incrementó la cantidad de aciertos que tenía el modelo.

Por último, se añadió el módelo al sistema previamente creado, editando ligeramente ciertas partes para integrar esta nueva función, además de crear un corpus de respuestas, ligado a los tipos de preguntas del otro corpus, de tal manera que una al detectar que una pregunta de cierto tipo, se dé una respuesta del mismo tipo.

3.2.2. Implementación

El nuevo sistema que se consiguió en esta fase, es una extensión del anterior, el cual cuenta con un estado más, CHATBOT_STATE. Al igual que el resto de estados, se accede y se sale de este estado mediante comandos de voz (figura 11).

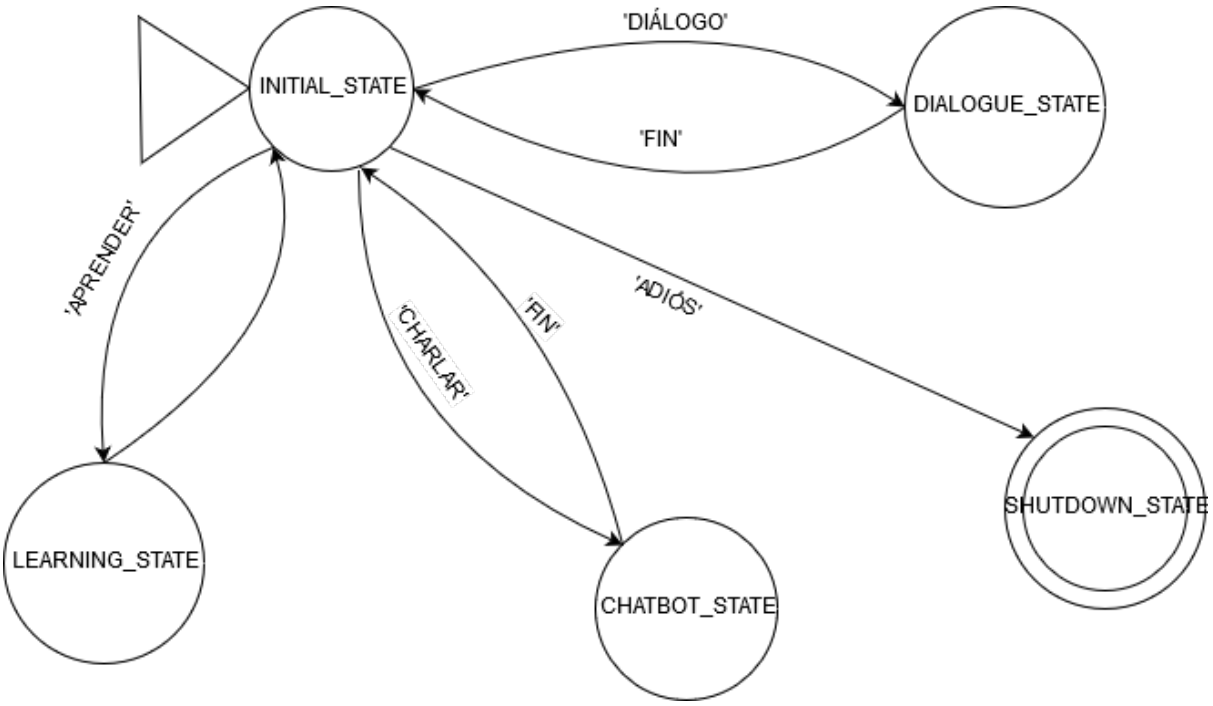

Figura 11: diagrama de estados y transiciones del sistema en la segunda fase.

A diferencia de la anterior fase, la cual tenía todos sus métodos en el mismo fichero, en esta fase se han separado en tres ficheros diferentes, por un lado está "NAODialog.py", que es el fichero principal, el que enlaza el resto de tecnologías con NAO; "doc2vec.py", que se ocupa de realizar el entrenamiento del modelo, además de obtener datos de el modelo creado y manejar el corpus de preguntas; y "loadModel.py", que carga el modelo guardado y obtiene una respuesta de este.

A continuación se analizan los métodos que se han implementado en esta fase, además de los cambios que se han podido realizar sobre el código de la anterior fase.

3.2.2.1. Método "main"

Este método funciona prácticamente igual que en la primera fase, con unos pequeños cambios para funcionar con el nuevo estado. En primer lugar, nada más en especial se le llama al método "loadResp" el cual se encarga de cargar el conjunto de respuestas de un fichero. Por otro lado, al haber un nuevo estado, se añade que se debe hacer en el caso de que el estado actual sea "CHATBOT_STATE": llamar al método "chat" y cambiar el estado actual a INITIAL_STATE.

3.2.2.2. Método "loadResp"

Este método se encuentra en el fichero "NAODialog.py" y se encarga de cargar las respuestas del fichero "respuestas.txt" y guardarlos en el diccionario global "resps".

3.2.2.3. Método “chat”

Este método se encuentra en el fichero “NAODialog.py” y su función es realizar la interaccón de chatbot. Su implementación es muy parecida al método “RecordedText” que se implementó en la primera fase. Al principio se subscribe a los eventos pertinentes y comienza la iteración principal, de la cual no se saldrá hasta que el usuario diga “terminar”. Dentro de esta iteración se encuentra otra que se encarga de obtener un archivo de voz que se graba, aquí se comprueba cada dos segundos si el usuario sigue hablando, si este para de hacerlo se procede a continuar con la ejecución. El siguiente paso es realizar una conexión ssh mediante a la librería “paramiko” para obtener la oración y poder transcribirla mediante speech_recognition. En el caso en el que el usuario diga “terminar”, se termina la conversación con NAO, por lo contrario, si se dice cualquier otra cosa, se llama al método “getAnswer” del fichero “loadModel.py” para obtener una respuesta, la cual el robot expresa por voz. Por último, en el caso de que el robot no haya escuchado nada, se repite todo el proceso.

3.2.2.4. Método “ChangeState”

Este método funciona prácticamente igual que en la anterior fase, se encarga de decidir cuál será el próximo estado. Como hay un nuevo estado en esta fase, se ha añadido el caso en el que si se dice “Charlemos” a la hora de elegir estado, el siguiente estado será “CHATBOT_STATE”.

3.2.2.5. Método “load”

Este método se encuentra en el fichero “doc2vec.py” y su función es cargar el corpus de preguntas en las listas “frase” para las preguntas y “etiq” para los tipos de estás, de tal manera que que el mismo índice representa la pareja de pregunta y su tipo. No recibe ningún dato de entrada ni genera datos de salida, edita las listas globales “frase” y “etiq”, además de “linecant” que guarda la cantidad de líneas del corpus.

3.2.2.6. Método “lema”

Este método se encuentra en el fichero “doc2vec.py” y su función es lematizar cada palabra de la frase que tiene como input. Para ello, se carga la herramienta en español de spacy, la cual permite tokenizar y lematizar en español las frases, de esta manera se analiza todas las palabras de la oración, se lematizan cada una de ellas y elimina las palabras innecesarias. Devuelve como resultado una lista que contiene cada una de las palabras lematizadas que se consideran necesarias. En el caso de querer usar el método “counRep” para contar las repeticiones de cada palabra, el momento adecuado se encuentra en este método.

3.2.2.7. Método “countRep”

Este método se encuentra en el fichero “doc2vec.py” y su función es contar el número de apariciones de cada palabra. Como el método no consta de una iteración para pasar por todas las palabras, es necesario utilizarlo en el método “lema”, de tal manera que cuente todos los token. Recibe como entrada un string, el cual será una palabra lematizada, en el caso de que en la lista “palabrasrep” (a lista global) no se encuentre dicha palabra, se añadirá esta palabra y en la misma

posición de la lista “repc” (lista global) se añadirá el valor 1, que representa cuántas veces ha aparecido esa palabra y se actualizará el valor de “np” (global), el cual cuenta el numero de palabras que hay. En caso contrario, es decir, si esta palabra se encuentra en la lista “palabrasrep”, se aumentará el contador correspondiente a dicha palabra.

3.2.2.8. Método “printRep”

Este método se encuentra en el fichero “doc2vec.py” y su función es imprimir en un fichero cada una de las palabras y sus repeticiones en el fichero “palabras.txt”. no tiene datos de entrada, ni de salida, pero utiliza las listas “palabrasrep” y “repc” y “np” para iterar cada una de las líneas que se van a escribir.

3.2.2.9. Método “calcPorcent”

Este método se encuentra en el fichero “doc2vec.py” y su función es calcular el porcentaje de aciertos del modelo guardado. Para ello, carga el modelo “d2v.model” y pasa cada una de las frases de la lista de preguntas “frase” por el modelo entrenado y obtiene una estimación del tipo al que pertenece, si dicho tipo es el mismo que hay en la lista “etiq”, se incrementa la cantidad de aciertos. Tras analizar cada una de las preguntas, se calcula el porcentaje de aciertos del modelo.

3.2.2.10. Método “pInecesaria”

Este método se encuentra en el fichero “doc2vec.py” y su función es determinar si la palabra entrante es necesaria o no, de tal manera que si se encuentra en la lista “palabras”, que guarda el conjunto de palabras innecesarias, se devuelve un True, en caso contrario se devuelve un False.

3.2.2.11. Método “printFrases”

Este método se encuentra en el fichero “doc2vec.py” y su función es imprimir en el fichero “frases.txt” cada una de las frases lematizadas y con las palabras innecesarias eliminadas, por sí en algún caso se quisiera ver como han quedado cada una de las frases tras lematizarlas y eliminar las palabras innecesarias. La función principal del fichero es identificar frases del mismo tipo que han quedado iguales tras lemtización y eliminación de palabras innecesarias y así eliminar redundancias del corpus.

3.2.2.12. Método “train”

Este método se encuentra en el fichero “doc2vec.py” y su función es entrenar el modelo mediante doc2vec, para ello se crea un modelo y se entrena con las frases y sus etiquetas, tras realizar el entrenamiento, el modelo se guarda en “d2v.model”, una vez guardado se puede volver a cargar para continuar entrenandolo

3.2.2.13. Método “getType”

Este método se encuentra en el fichero “loadModel.py” y su función es obtener el tipo de la pregunta que se le pasa como entrada. Por lo tanto, carga el modelo “d2v.model” lo lematiza usando el método “lema” y devuelve como salida el resultado de la estimación que se ha realizado para dicha oración.

3.2.2.14. Método “getAnswer”

Este método se encuentra en el fichero “loadModel.py” y su función es obtener una respuesta del mismo tipo de la pregunta que se le pasa como entrada. Para ello, a parte de la pregunta, obtiene como entrada un diccionario, el cual guarda los diferentes tipos y las respuestas que corresponden a cada uno de estos tipos. Lo que hace este método es llamar a “getAnswer” para obtener el tipo correspondiente de la pregunta y devolver una respuesta aleatoria a corde con el tipo que se ha estimado.

3.3. Guía de uso del sistema

Nada más arrancar el programa tardará unos segundos en estar a punto ya que tiene que cargar varios archivos. Una vez realizada la carga de datos, el robot preguntará qué es lo que el usuario quiere realizar, dándole cuatro opciones.

“Elige una opción, aprender, diálogo o charlar. En el caso de que quieras que me apague di adiós”

En el caso de decir “*adiós*”, robot se despide con un *“Buenas noches”* y a continuación se apagará.

En el caso de decirle *“Charlar”*, el robot dirá *“Charlemos”* y comenzará el sistema de chatbot, donde el robot responderá a todas las preguntas que se le hagan. En el momento que se quiera detener esta charla, el usuario debe decir *“terminar”*, de tal manera, NAO expresa que ha entendido con *“Me lo he pasado bien, hablemos otro día”* y volverá a preguntar sobre lo que el usuario desea hacer.

En el caso de decir *“diálogo”*, el robot contesta con *“Hablemos pues”* y comienza el sistema de diálogo de NAO que utiliza el fichero con las frases que se han aprendido. Mientras se encuentra en este estado, el NAO responde a las preguntas de dicho archivo de forma correcta, y cuando el usuario diga *“terminar”* terminará el diálogo y volverá a preguntar acerca lo que el usuario quiere hacer.

Por último, en el caso de que se le diga *“aprender”,* el robot contesta con *“Vamos a aprender”* y continuará con *“Dime la frase que quieres que aprenda”* a lo que el usuario deberá responder. En el caso de que el robot no haya podido grabar correctamente lo que el usuario ha dicho lo comunica *“No te he entendido, repite por favor”*, en caso contrario, reproduce lo que el usuario ha dicho.

"Has dicho _____. ¿Es correcto?"

A lo que el usuario deberá responder con un *"sí"* o *"no"*, en el caso de ser un *"no"*, dice *"Has dicho que no. Repite por favor"* y se tendrá que volver a expresar la frase, si es un *"sí"*, responde *"Has dicho que sí"* y procede a preguntar por la respuesta correspondiente *"Dime cómo debería responder a eso"* y repite el mismo proceso que a la hora de aprender la pregunta.

Observando la figura 12 uno puede aprender como utilizar el sistema para realizar las acciones que desea, además de como reaccionará el robot en consecuencia.

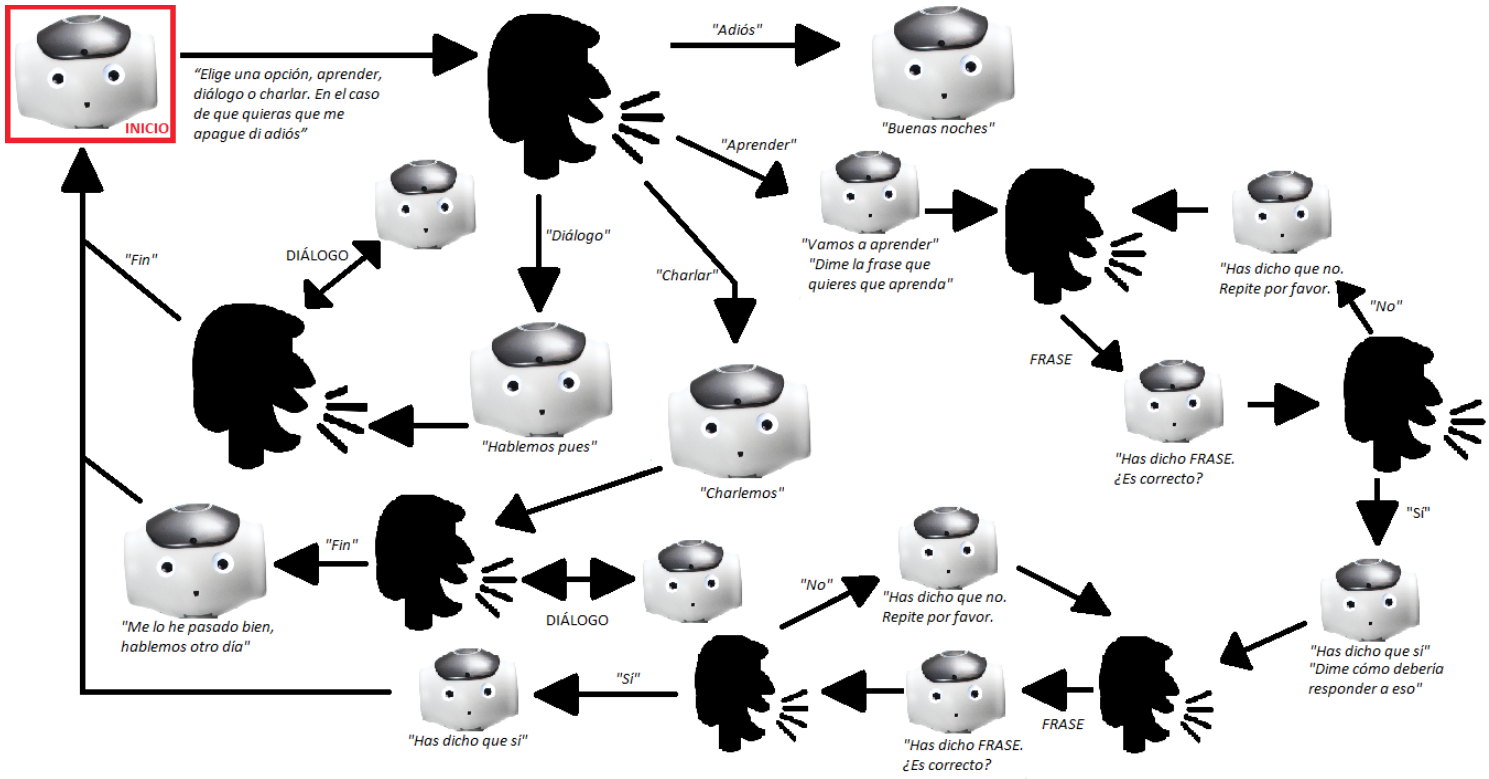

Figura 12: iteración

3.4. Archivos y formatos

En total, el sistema utiliza tres ficheros para manejar la información. Por un lado, está el archivo "dialogo.top", sirve para guardar parejas de preguntas y respuestas, por otro lado, tenemos los archivos "chatbotnltk.txt" y "respuestas.txt" que tienen la misma estructura pero uno guarda las preguntas y el otro las respuestas.

3.4.1 Archivo “dialogo.top”

Como se ha mendionado anteriormente, este archivo guarda parejas de pregunta-respuesta, las cuales son aquellas que aprende el robot. El SDK de NAO da la capacidad de utilizar archivos de diálogo, los cuales tienen ya una estructura determinada y para poder utilizarlos los archivos deben encontrarse en el propio robot, es decir, si el archivo se encuentra en el computador que ejecuta el programa no se podrá utilizar. Por esta razón, siempre que se quiere realizar la función de “dialogo” del robot se sube este archivo, de esta manera, si se han añadido nuevas parejas pregunta-respuesta” estarán actualizadas. El formato del fichero es simple, la primera línea representa el nombre del tema y se escribe “topic: ~example_topic()”. La segunda línea corresponde al idioma que se utilice, en el caso del español se escribe “language: spe”. el resto de lineas son parejas de preguntas y respuestas escritas “u: (PREGUNTA) RESPUESTA” .

```
topic: ~example_topic()
language: spe
u: (fin) "  "
u: (no digas más) me da igual
u: (hola qué tal estás) estoy bien gracias por preguntar
```

3.4.2. Archivos “chatbotnltk.txt” y “respuestas.txt”

Estos dos archivos tienen la misma estructura, la cual es muy simple, en cada línea se encuentra una oración y el tipo al que corresponde, separados mediante “#”. El archivo “chatbotnltk.txt” guarda las preguntas y sus tipos correspondientes, los cuales se cargan en dos listas (una para las preguntas y la otra para sus tipos) que se utilizan para entrenar el modelo. Por otro lado, el fichero “respuestas.txt” guarda las parejas de respuestas con su tipo correspondiente, los cuales se cargan en un diccionario de tal manera que a cada tipo le corresponden varias respuestas, esta estructura facilita la selección de una respuesta a partir de un tipo.

```
hola#1
buenos días#1
buenas tardes#1
qué tal#2
cómo estás#2
cómo te encuentras#2
buenas noches#1
buenas#1
hey#1
```

```
qué tal estás#2
qué tal te encuentras#2
cómo te sientes#2
```

En total, el sistema cuenta con 66 tipos diferentes los cuales están representados por números (0-65) pro en realidad tienen su propio significado.

```
0 edad    1 saludo    2 qtal      3 conocerte       4 molesto
5 respondeme    6 malo      7 estudia   8 guapo     9 cumpleaños
10 aburrido           11 jefe     12 ocupado  13 ayuda    14 chatbot
15 listo  16 loco     17 despedido      18 gracioso       19 bueno
20 feliz  21 hobby    22 hambre         23 casarse        24 amigo
25 ocupación    26 origen         27 preparado      28 real
29 residencia   30 verdad   31 seguro   32 hablar   33 estar
34 mal    35 bien     36 nproblem       37 gracias        38 denada
39 bienhecho    40 espera         41 abrazo   42 nimporta
43 perdon       44 wtf      45 incorrecto     46 adios
47 encantado    48 verte    49 hablarte       50 qhay
51 enfadado           52 volver   53 ndormir        54 nhablar
55 entusiasmo   56 cama     57 broma    58 gustar   59 solo
60 amar   61 echardmenos    62 consejo  63 triste   64 sueño
65 cansado
```

4. Conclusiones y Resultados

Para concluir veremos los resultados que se han obtenido en la segunda fase, observaremos que mejoras podría hacerse al robot de cara al futuro y se analizará donde puede llegar a ser útil este robot.

4.1. Resumen del trabajo realizado

El camino que he tomado para acabar el trabajo ha sido largo y con varios obstáculos, desde problemas de compatibilidad de las diferente herramientas que se han probado, hasta problemas a la hora de entender el funcionamiento de algunas de estas herramientas.

La decisión que se tomó en mitad del desarrollo del proyecto de ampliar los objetivos e intentar introducir más profundidad al proyecto fue muy afortunada. Le dio más riqueza al proyecto, añadiendole contenido, y me ofreció la oportunidad de trabajar con otras tecnologías a parte de las que ya había utilizado.

El producto final cumple sus funciones, pero no es perfecto, el sistema realizado en la primera fase, a pesar de ser bastante sólido no es demasiado útil. Tener que utilizar preguntas fijas hace que sea un sistema que no da demasiada flexibilidad.

Por otro lado, la función que se añadió en la segunda fase ofrece justamente lo contrario, es decir, no es igual de preciso que el primero, ya que a pesar de hacerle una pregunta que se hubiese presentado en el entrenamiento no hay garantía de que la vaya a reconocer correctamente, por lo tanto no es del todo fiable. Por otro lado, el hecho de haber entrenado el sistema para reconocer las intenciones del usuario hace que aumente la cantidad de frases que es capaz de reconocer e interpretar correctamente.

La metodología utilizada para desarrollar el proyecto ha sido adecuada. En la primera fase, antes de realizar nada relacionado con programación, me tenía que formar acerca del funcionamiento de los sistemas de diálogo y como se podía crear uno sencillo.

Una vez hecho esto, se buscaron las tecnologías adecuadas para seguir adelante. Después, se probaron para averiguar cómo funcionaban, si realmente satisfacían las necesidades que habían y si eran compatibles con el resto de tecnologías. En varias ocasiones se tuvieron que hacer más búsquedas, ya que algunas herramientas no eran adecuadas.

A continuación, había que diseñar un sistema para realizar los objetivos que se habían marcado en la primera fase. Para ello, se analizaron cada una de las funciones que debía tener y se generó un sistema que permitiese navegar de una a otra sin necesidad de insertar comandos por escrito, es decir, que funcionase completamente por comando de voz.

En la segunda fase se siguieron los mismos pasos, lo primero fue añadir nuevos objetivos se buscó la manera de llevarlos, así como que herramientas se necesitaban para poder realizarlo, teniendo en cuenta las restricciones que el NAO tenía. A continuación, se probaron nuevamente estas tecnologías, y se descartaron las que no era útiles.

Después se analizó cómo integrar la nueva función sin alterar en gran medida lo realizado en la fase anterior. Una vez hecho esto, se implementó la nueva función, la cual, al igual que las anteriores tendría se utilizaría mediante comandos por voz.

Finalmente, el sistema que se ha conseguido cumplia sus objetivos, pero no es perfecto, y todavía se puede mejorar. La primera de sus carencias reside en el reconocimiento de voz, el cual complica las cosas, ya que a la hora de decirle que hacer (aprender, diálogo, charlar o adiós), el reconocimiento de voz puede interpretar de forma errónea lo que uno a dicho, por lo tanto, puede ocurrir que el sistema entre un estado no deseado. En el caso de “LEARNING_STATE” y

“SHUTDOWN_STATE” resulta molesto que los interprete de forma inadecuada, ya que si pasa a “SHUTDOWN_STATE” el robot se apagará y se finaliza el proceso y en si entra en “LEARNING_STATE”, no se podrá salir de este hasta que se añada un conjunto de pregunta y respuesta.

4.2. Resultados obtenidos

A diferencia de la primera fase, la cual responde únicamente a las preguntas que tiene en el fichero, en la segunda fase se puede hacer un entrenamiento de modelo, el cual estima cual es el tipo de la pregunta que se le ha hecho. Por lo tanto, no acierta en todos los casos. El acierto depende del entrenamiento del modelo, así como del corpus que se utiliza para realizar dicho entrenamiento.

Para entrenar el modelo mediante doc2vec de la librería gensim, se utilizan las variables “max_epoachs”, vec_size, alpha, min_alpha, min_count, dm y workers.

max_epochs: Cantidad de iteraciones máximas que se realizan en el corpus, empieza en uno y termina con max_epoch, de tal manera que estos van aumentando mientras se entrena el modelo, y cada vez se itera más veces el corpus.

vec_size: Tamaño del los vectores de características.

alpha: Tasa inicial de aprendizaje.

min_alpha: A medida que se entrena el modelo, la tasa inicial de aprendizaje va bajando hasta llegar a min_alpha.

min_count: en esta variable se indica el mínimo de veces que debe aparecer una palabra para tenerla en cuenta, por ejemplo, si la palabra “hola” solo aparece una vez en todo el corpus de entrenamiento y min_count es dos, entonces la palabra “hola” no se tendría en cuenta. Para evitar problemas de este tipo, min_count se establece en 1, de tal manera que todas las palabras se tienen en cuenta.

dm: Si dm es 1, se utiliza el algoritmo “distributed memory”. En caso contrario, se utiliza “distributed bag of words”.

workers: es el número de hilos que se utilizarán para realizar el entrenamiento, lo cual influye en la velocidad en la que se realice el entrenamiento.

La primera vez que se entrenó el modelo, se hizo con valores estándares y sin eliminación de palabras innecesarias del corpus, por lo tanto el porcentaje de aciertos que se consiguió fue decepcionante, con un 50.34% de aciertos. Este y el resto de de pruebas que se han realizado para ver el porcentaje de acierto que se conseguía se han hecho utilizando el mismo corpus que se ha utilizado para entrenar el modelo.

Lo que había que hacer a continuación, era mejorar ese porcentaje, de tal manera que fuese un índice de aciertos aceptable, para ello, había que eliminar las palabras que no diesen información suficiente, y modificar las variables que se utilizaban en el entrenamiento del modelo.

Primero se probó cambiando el algoritmo que se utilizaba para ver cual era mejor y el resultado demostró que “distributed bag of words” daba mejores resultados. Aun así, se hicieron más pruebas

con “distributed memory”, ya que era posible que al cambiar otras variables esta mejorase enormemente. El resultado fue decepcionante, reduciendo el índice de aciertos hasta un 24,48%

A continuación, se amplió vecsize de 20 a 50 y 100 para ver los cambios con ambos algoritmos, el cambio que se obtuvo demostró que aumentar esta variable mejoraba la capacidad de predicción del modelo y se acabó descartando el algoritmo “distributed memory” ya que los resultados que daban era mucho peores.

vecsize	20	50	100
aciertos dm=0	50,34%	62,50%	65,92%
aciertos dm=1	24,48%	34,08%	36,47%

Tabla 1: resultados de clasificación con distintos tamaños de vectores para “distributed memory” y “distributed bag of words”

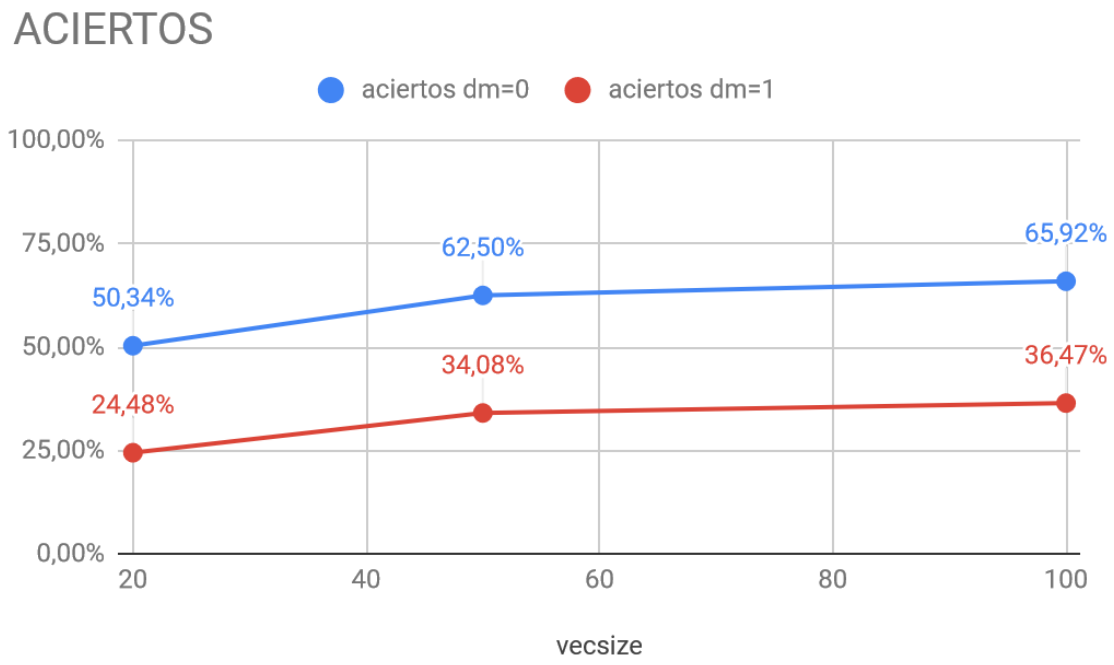

Figura 13: aciertos en función del tamaño de vector para “distributed memory” y “distributed bag of words”

Una vez se llegó a este punto, se analizaron la aparición de las palabras en el corpus, y se localizaron cuales eran las más utilizadas con el fin de eliminarlas antes de hacer el entrenamiento. La razón de realizar esto reside en intentar que las palabras con mucha apariciones no interfirieran a la hora de predecir el tipo de la pregunta, porque hay ciertas palabras que no dan demasiada información ya que aparecen en tantas frases de diferentes tipos que no resultan nada útiles a la hora de elegir el tipo al que puede pertenecer. Al eliminar estas palabras, los resultados mejoraron hasta un 71,4%

Llegado a este punto, se siguió haciendo pruebas aumentado el vecsize para ver hasta qué punto podrían mejorar los resultados, y observar en qué punto dejan de aumentar.

vecsize	aciertos
100	71,40%
150	71,40%
200	72,77%
250	72,43%
300	71,90%
350	72,43%
364	71,90%
400	72,43%
450	72,08%
500	71,74%

Tabla 2: aciertos frente a tamaño de vector para “distributed bag of words”

El corpus cuenta con 364 palabras diferentes, por lo tanto, no se esperaba que a partir de ese número el índice de aciertos mejorase. A partir del vecsize 100, los resultados obtenidos no cambian demasiado, y aunque en el vecsize 364 se esperaba que tuviese mejor resultado que en otros vecsize, no ha resultado como se esperaba.

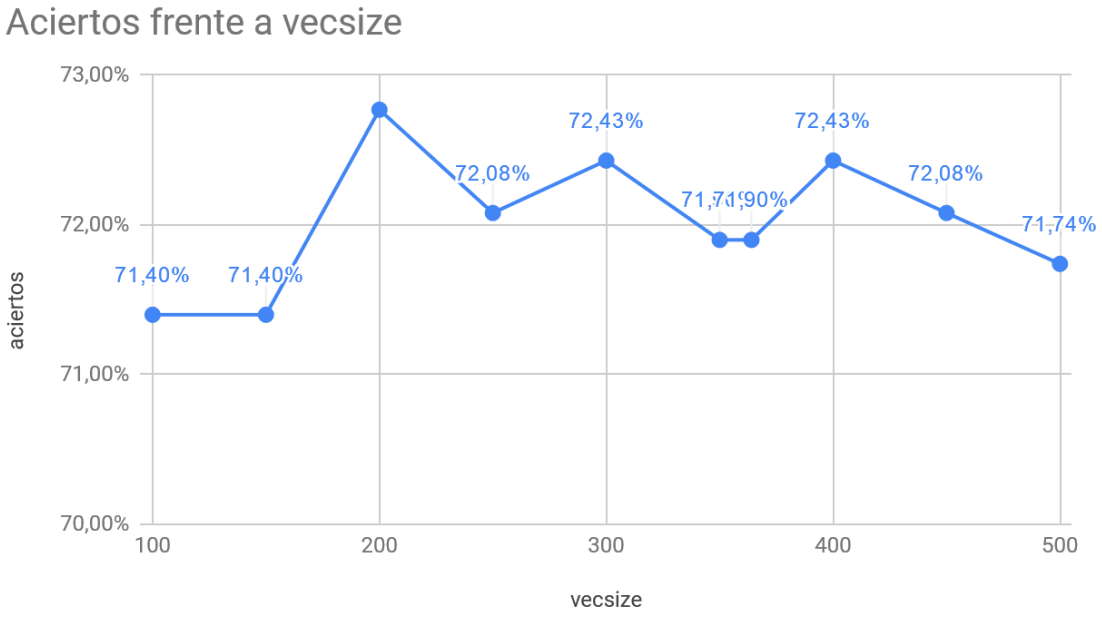

Figura 14: aciertos frente a vecsize para “distributed bag of words”

El siguiente paso fue probar a cambiar max_epoach, alpha y min_alpha, con la finalidad de ver qué ocurría. Los resultados obtenidos no fueron nada buenos, tanto al subirlos como al bajarlos el porcentaje de aciertos se redujo. Como no se comprendía cómo funcionaban exactamente se decidió no continuar con estas pruebas.

Durante todas las pruebas se mantuvo el número de workers constante ya que no afecta a el resultado, solo a la velocidad en la que se obtiene.

En cuanto a min_count, se mantuvo en 1 en todo momento, ya que en el corpus hay ciertas palabras que solo aparecen una vez y a pesar de eso son relevantes a la hora de predecir el tipo, por lo que no era conveniente eliminarlas, hacerlo solo reduciría el índice de aciertos.

4.3. Propuesta de Mejoras y Siguientes Pasos

Como se ha mencionado antes, una de las mayores problemas del sistema es el reconocimiento de voz, ya que en ocasiones se equivoca, por lo tanto podría mejorarse para reducir las posibilidades de confundir lo que uno dice.

Por otro lado, ahora mismo el conjunto de tipos de frases que entiende la función “charlar” no es demasiado elevado, con 66 tipos diferentes, se podría ampliar esta capacidad a una mayor, de tal manera que pudiese responder a una mayor cantidad de preguntas. En el caso de querer utilizarlo solo para preguntas concretas se podría reducir la cantidad de tipos de preguntas, incluso se podría eliminar todas las que ya existen y reemplazarlas por otras completamente distintas. Por ejemplo, si se desea usar este sistema como asistente de una tienda, no es necesario que tenga el tipo de frase “estudia”, ya que las personas que lo vayan a utilizar no van a exigir al NAO que se ponga a estudiar. Si se quiere utilizar el sistema para situaciones concretas es mejor utilizar una cantidad pequeña de tipos preguntas, de esta manera el porcentaje de aciertos será mayor, pero si la cantidad de preguntas que se vayan a realizar es muy pequeña, se recomienda usar la función “diálogo” en lugar de “charlar” ya que es más fiable para preguntas concretas.

El entrenamiento que se ha realizado para crear el modelo que se utiliza en la función “charlar” no ha sido el mejor, se podría trabajar en esto para refinar sus aciertos.

Otra mejora que se le podría añadir al sistema es darle la capacidad de realizar análisis sintácticos de las frases entrantes, de tal manera que pudiese identificar las personas implicadas en la frase y responde en consecuencia. El sistema actual trata las oraciones “soy guapo” y “eres guapo” por igual, ya que al lematizar las palabras “soy” y “eres”, ambas se convierten en “ser”, por lo tanto la frase que se utiliza para identificar el tipo de pregunta sería “ser guapo” en ambos casos y el tipo que se predice también sería el mismo.

A pesar de que no representa un problema demasiado importante, se podría mejorar el sistema de estados, de tal manera que se pueda salir del estado “entrenar” en el caso de entrar en él accidentalmente.

También se pensó en añadir una función para hacer olvidar al robot las frases aprendidas, finalmente no se añadió esta función ya que el hecho de realizar esto por comandos de voz podría ser tedioso, ya que habría que identificar las parejas de pregunta y respuesta que se quieren borrar

sin eliminar el resto. Para ello el usuario debería saber cuales son todas las frases que parejas de pregunta-respuesta tiene guardadas, pero el robot no puede mostrarlas, de manera que la única forma de informar de las preguntas-respuestas que tiene sería recitarlas y se podría tardar demasiado en realizarlo.

Al igual que existentes asistentes virtuales, se podría añadir la capacidad de hacer búsquedas en internet acerca de las preguntas que se le hacen, y conseguir una capacidad de respuesta mucho mayor.

4.4. Utilidad del proyecto

El proyecto en sí consiste en un sistema que crea una charla artificial con un robot que muestra cierta inteligencia, este sistema se centra en responder a preguntas de poca relevancia y en principio no tiene una utilidad demasiado práctica, ya que solo sirve como entretenimiento, pero solo es un proyecto, y su finalidad es mostrar que se puede hacer con este sistema.

En realidad, tal y como se ha dicho antes, el proyecto es un sistema que es capaz de responder a las preguntas que se le hacen, a las cuales se le responden de la manera en la que se le ha enseñado. Entonces, si se le enseña a responder a preguntas concretas para situaciones concretas podría ser un sistema útil de cara al público. Por ejemplo, se podría aplicar este sistema en tiendas, de tal manera que tenga guardadas las preguntas más frecuentes que se puedan realizar. Obviamente, para realizar esto, anteriormente hay que hacer un análisis de la situación y comprobar cuales pueden ser estas preguntas.

Siguiendo con el mismo ejemplo, podrían añadirse tipos de preguntas relacionados con secciones concretas de la tienda de manera que el sistema sea capaz de indicar a los usuarios en qué lugar de la tienda se encuentra lo que buscan o dar respuestas sobre los productos que tienen.

4.5. Conclusión Final

A pesar de no haberse realizado todo el desarrollo como uno hubiese querido, finalmente la experiencia que se ha obtenido ha sido mucho más gratificante y enriquecedora. El hecho de que haya habido algún que otro problema al que se ha tenido que enfrentar obliga al desarrollador a solucionar sus propios problemas, cuando durante el resto de la carrera había siempre algún profesor que ayudaba.

La experiencia que se recibe al realizar este proyecto es muy enriquecedora, ya que uno aprende a trabajar por sus propios medios, investigar acerca de lo que quiere hacer y cómo puede llevarlo a cabo. Es un buen primer paso antes de comenzar la vida laboral, gracias a esta experiencia uno se da cuenta de cómo debe afrontar los problemas y se puede hacer una idea de cómo se trabaja en proyectos verdaderos.

Este proyecto en concreto, al principio no parecía que diese para mucho, ya que las capacidades del robot NAO no eran tan avanzadas como se esperaba. De todas formas, a medida que se investigó sobre el tema apareció se encontró más información, la cual demostraba todo lo que se

podía hacer. Gracias a estos datos, se extendió lo que sería el desarrollo, para lograr un sistema más complejo y que realmente demostrase cierta inteligencia.

Por último, a pesar de que el proyecto ha terminado, es importante decir que este desarrollo podría haberse alargado y haber acabado siendo algo más completo, ya que la cantidad de funcionalidades o complejidad que se le puede añadir al sistema es enorme, y de ser una especie de chatbot, podría pasar a ser algo más útil.

Bibliografía

[1] Rámon López-Cózar, Zoraida Callejas, David Griol y José F. Quesada: *Review of spoken dialogue systems* http://loquens.revistas.csic.es/index.php/loquens/article/view/17/47#RF0001
[2] *Deep Learning of Dialogue Systems*
https://www.csie.ntu.edu.tw/~yvchen/doc/DeepDialogue_Tutorial_ACL.pdf
[3] *Asistente virtual, Amazon Alexa, Microsoft Cortana, Siri, Asistente de google y Robot NAO :* Wikipedia
[4] *La historia de Siri y su creador* https://www.nextu.com/blog/la-historia-detras-de-siri-y-su-creador-el-hombre-que-nacio-desarrollador-de-aplicaciones/
[5] Documentación de paquete NAOqi http://doc.aldebaran.com/2-1/naoqi/sensors/dcm.html
[6] Información adicional sobre NAO, https://aliverobots.com/nao/
[7] Guía de bolsillo de NAO
[8] paramiko http://www.paramiko.org/
[9]spaCY https://github.com/explosion/spaCy
[10]gensim https://radimrehurek.com/gensim/intro.html
[11]speech recognition https://github.com/Uberi/speech_recognition#readme
[12]pycharm https://www.jetbrains.com/pycharm/
[13]Gidi Shperber, *A gentle introduction to Doc2Vec https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e*
[14]*Word2Vec Turotial- The Skip-Gram Model,* http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
[15]Choregraphe http://doc.aldebaran.com/2-1/software/choregraphe/choregraphe_overview.html
[16]Dialogflow https://dialogflow.com/
[17]NLTK https://www.nltk.org/

Anexo A: NAODialog.py

```
# -*- coding: utf-8 -*-
import speech_recognition as sprec
import paramiko
import time
import os
from naoqi import ALProxy
from naoqi import ALBroker
from optparse import OptionParser
from loadModel import getAnswer



# Global variable to store the SelectState module instance
SelectState = None
matrix=None
linequant=0
INITIAL_STATE=0
DIALOGUE_STATE=1
LEARNING_STATE=2
SHUTDOWN_STATE=3
CHATBOT_STATE=4
resps={}

#rellena el fichero dialogo.top con la información que hay en "matrix"
def createfile(c):
    global matrix
    f = open("dialogo.top", "w")
    f.write("topic: ~example_topic()\n")
    f.write("language: spe\n")
    i=0
    while i<c-2 :
        p=matrix[i][0]
        r=matrix[i][1]
        ltowrite="u: ("+p+") "+ r + "\n"
        print ltowrite
        f.write(ltowrite)
        i=i+1
    f.close()


#lee el fichero "dialogo.top" y rellena matrix con su información
def creatematrix():
    global matrix

    f = open("dialogo.top", "r")

    c = 0
    # contar numero de lineas
```

```
    for line in open("dialogo.top"):
        c = c + 1
    lines = [line.rstrip('\n') for line in open("dialogo.top")]


    matrix = [[0 for x in range(2)] for y in range(c-2)]
    i = 0
    for line in lines:
        if (i==0 or i==1):
            #las primeras 2 lineas no son necesarias
            leer = f.readline()
            i=i+1
        else:
            #lee la linea y añade la pregunta-respuesta a "matrix"
            leer = f.readline()
            p, r = leer.split(")")
            p = p[4:]
            r = r[1:-1]
            matrix[i-2][0] = p
            matrix[i-2][1] = r
            i = i + 1
    f.close()
    return c

#este metodo descarga del nao el fichero "dialogo.top" y lo guarda en un
fichero local, de esta manera se puede editar sin restricciones
def downloadfile():
    NAO_IP = "192.168.1.140"
    NAO_USERNAME = "nao"
    NAO_PASSWORD = "nao"
    pport = 9559

    #conectarse con el robot
    ssh = paramiko.SSHClient()
    ssh.load_system_host_keys()
    ssh.connect(NAO_IP, username=NAO_USERNAME, password=NAO_PASSWORD)

    #descargar el fichero
    sftp = ssh.open_sftp()
    localpath = "dialogo.top"
    remotepath = "/home/nao/dialogo.top"
    sftp.get(remotepath, localpath)
    sftp.close()
    ssh.close()

#Sube el fichero "dialogo.top" al Nao, es necesario subirlo al robot para
que este pueda utilizarlo
def uploadfile():
    NAO_IP = "192.168.1.140"
    NAO_USERNAME = "nao"
    NAO_PASSWORD = "nao"
    pport = 9559
    localpath = "dialogo.top"
```

```
    remotepath = "/home/nao/dialogo.top"

    #sube el archivo
    transport=paramiko.Transport((NAO_IP,22))
    transport.connect(username=NAO_USERNAME,password=NAO_PASSWORD)
    sftp=paramiko.SFTPClient.from_transport(transport)
    sftp.put(localpath,remotepath)
    #ssh = paramiko.SSHClient()
    #ssh.load_system_host_keys()
    #ssh.connect(NAO_IP, username=NAO_USERNAME, password=NAO_PASSWORD)

    #sftp = ssh.open_sftp()
    #sftp.get(remotepath, localpath)
    sftp.close()
    transport.close()
    #ssh.close()

#El robot hará varias preguntas de respuesta si o no, por ello existe
este metodo, para no tener que implementarlo en varias ocasiones.
def SioNo():
    sr = ALProxy("ALSpeechRecognition")
    memory = ALProxy("ALMemory")
    tts = ALProxy("ALTextToSpeech")
    leds=ALProxy("ALLeds")

    #suscribirse a eventos y definir el vocabulario
    memory.subscribeToEvent("WordRecognized", "onWordRecognized",
"SpeechDetected")
    memory.insertData("WordRecognized","nothing")
    vocabulary = ["Si","No","nothing"]

    #Especificar idioma y insertar vocabulario.
    sr.pause(True)
    tts.setLanguage("Spanish")
    sr.setVocabulary(vocabulary, False)
    sr.pause(False)

    # voice = memory.getData("SpeechDetected")
    leds.fadeRGB("FaceLeds", 0, 1, 0, 0)
    #esperar hasta escuchar algo, un si o un no.
    data="nothing"
    while data=="nothing":
        time.sleep(2)
        try:
            data = memory.getData("WordRecognized")
        except:
            memory.insertData("WordRecognized", "nothing")
    listened = data[0]
    leds.reset("FaceLeds")
    memory.insertData("WordRecognized", "nothing")

    #
```

```
    #evuelve 1 en el caso de que haya escuchado "si" y o en el caso de
escuchar "no"
    #print "Data " +data[0]
    if listened == "Si":
        return 1
    elif listened == "No":
        return 0

#Obtiene un texto a partir de un audio que graba
def getrRecordedText(NAO_IP,NAO_USERNAME,NAO_PASSWORD,text):
    global sr
    record = ALProxy("ALAudioRecorder", NAO_IP, 9559)
    aup = ALProxy("ALAudioPlayer", NAO_IP, 9559)
    leds=ALProxy("ALLeds")
    memory = ALProxy("ALMemory")

    memory.subscribeToEvent("SpeechDetected", "onSpeechDetected",
"WordRecognized")

    voice = True

    tts = ALProxy("ALTextToSpeech")
    tts.say(text)
    #Mientras no se confirme que se ha entendido lo que el usuario ha
dicho, el bucle continuará
    fin=False
    while fin==False:
        #print("START RECORDING...")
        #time.sleep(2)
        #print("Start recording...")
        #empieza a grabar
        leds.fadeRGB("FaceLeds", 0, 1, 0, 0)
        record.startMicrophonesRecording("/home/nao/record.wav", 'wav',
16000, (0, 0, 1, 0))

        #seguirá grabando hasta que el usuario este 2 segundos sin
hablar.
        while voice == True:
            time.sleep(2)
            voice = memory.getData("SpeechDetected")

        record.stopMicrophonesRecording()
        leds.reset("FaceLeds")
        #print("END RECORDING")

        # SPEECH TO TEXT

        #Se hace consexión con Nao para obtener el audio grabado
        ssh = paramiko.SSHClient()
        ssh.load_system_host_keys()
        ssh.connect(NAO_IP, username=NAO_USERNAME, password=NAO_PASSWORD)

        #se descarga el audio grabado para utilizarlo
```

```
        sftp = ssh.open_sftp()
        localpath = "prueba.wav"
        remotepath = "/home/nao/record.wav"
        sftp.get(remotepath, localpath)
        sftp.close()
        ssh.close()

        #a continuación se hace el reconocimiento de voz, en el caso de
error, Nao pedira que se repita la frase,
        #si no ha ocurrido ningún error, se le preguntará al usuario si
lo que el reconocimiento de voz ha entendido es correcto o no
        #si no es correcto se repetirá el proceso otra vez.
        r = sprec.Recognizer()
        harvard = sprec.AudioFile("prueba.wav")
        with harvard as source:
            audio = r.record(source)
        try:
            ema=(r.recognize_google(audio,language="es-ES"))
            n = 'Has dicho, ' + ema.encode("utf-8") + ' .¿Es correcto?'
            tts.say(n)
            son=SioNo()
            if son== 0:
                fin=False
                voice=True
                tts.say("Has dicho que no. Repite la frase por favor.")
            else:
                fin = True
                tts.say("Has dicho que si.")
        except:
            fin=False
            tts.say("No te he entendido, repite por favor.")
            voice=True
            pass
    return ema

#se produce un dialogo con el robot Nao, este utilizará la información
que se le ha enseñado y está en el fichero "dialogo.top"
def dialogue():

    ALDialog = ALProxy("ALDialog")
    leds=ALProxy("ALLeds")
    ALDialog.setLanguage("Spanish")
    leds.fadeRGB("FaceLeds", 0, 1, 0, 0)

    # cargar fichero
    topic_path = "/home/nao/dialogo.top"
    topf_path = topic_path.decode('utf-8')
    topic_name = ALDialog.loadTopic(topf_path.encode('utf-8'))
    # activar la información del fichero (active topic), esto hará que
cuando escuche una pregunta que reconoce responda
    #automaticamente.
    ALDialog.activateTopic(topic_name)
```

```
    memory = ALProxy("ALMemory")
    memory.insertData("WordRecognized","nothing")
    print memory.getData("WordRecognized")
    memory.subscribeToEvent("WordRecognized", "onWordRecognized",
"SpeechDetected")


    #en el caso que escuche "terminar" se terminará el dialogo

    listened = "nothing"
    while (listened != "terminar"):
        data = memory.getData("WordRecognized")

        try:
            listened = data[0]
        except:
            listened = "nothing"
            memory.insertData("WordRecognized", "nothing")
        #print listened
    print listened
    #desactivar el topico para que no siga respondiendo
    ALDialog.deactivateTopic(topic_name)
    ALDialog.unloadTopic(topic_name)
    memory.insertData("WordRecognized","nothing")
    leds.reset("FaceLeds")
    #print "SE ACABO EL DIALOGAR"

#Teniendo en cuenta el estado actual del sistema se seleccionará cual es
el siguiente.
def ChangeState():
    global SelectState
    global INITIAL_STATE
    global DIALOGUE_STATE
    global LEARNING_STATE
    global SHUTDOWN_STATE
    global CHATBOT_STATE

   # SelectState = SelectStateModule("SelectState")
    sr=ALProxy("ALSpeechRecognition")
    memory=ALProxy("ALMemory")
    tts=ALProxy("ALTextToSpeech")
    leds=ALProxy("ALLeds")

    #suscribirse a eventos, insertar vocabulario e idioma

memory.subscribeToEvent("WordRecognized","onWordRecognized","SpeechDetect
ed")
    memory.insertData("WordRecognized", "nothing")
    print memory.getData("WordRecognized")
    vocabulary = ["Diálogo", "Aprender", "Adios","Charlar","nothing"]

    sr.pause(True)
    tts.setLanguage("Spanish")
```

```
    sr.setVocabulary(vocabulary, False)
    sr.pause(False)
    time.sleep(5)

    #Se le pregunta al usuario que quiere hacer, dependiendo de la
respuesta pasará a un estado u otro.
    voice = False
    tts.say("Elige una opción, aprender, diálogo o charlar. En caso de
que quieras que me apague di adios")
    leds.fadeRGB("FaceLeds", 0, 1, 0, 0)
    data="nothing"
    while data=="nothing":
        time.sleep(2)
        try:
            data = memory.getData("WordRecognized")
            print data
        except:
            memory.insertData("WordRecognized", "nothing")
    listened = data[0]
    memory.insertData("WordRecognized", "nothing")
    leds.reset("FaceLeds")
    #print data[0]
    if listened == "Adios":
        tts.say("Buenas Noches")
        #memory.unsubscribeToEvent("WordRecognized","SpeechDetected")
        return SHUTDOWN_STATE
    elif listened == "Aprender":
        tts.say("Vamos a aprender")
        #memory.unsubscribeToEvent("WordRecognized","SpeechDetected")
        return LEARNING_STATE
    elif listened == "Diálogo": #Diaglogo
        tts.say("Hablemos pues")
        #memory.unsubscribeToEvent("WordRecognized","SpeechDetected")
        return  DIALOGUE_STATE
    else: #Charlar
        tts.say("Charlemos")
        return CHATBOT_STATE

def chat(NAO_IP,NAO_USERNAME,NAO_PASSWORD):
    global sr
    global resps
    record = ALProxy("ALAudioRecorder", NAO_IP, 9559)
    aup = ALProxy("ALAudioPlayer", NAO_IP, 9559)
    leds = ALProxy("ALLeds")
    memory = ALProxy("ALMemory")

    memory.subscribeToEvent("SpeechDetected", "onSpeechDetected",
"WordRecognized")


    tts = ALProxy("ALTextToSpeech")
    fin = False
    while fin == False:
```

```
        # print("START RECORDING...")
        # time.sleep(2)
        # print("Start recording...")
        leds.fadeRGB("FaceLeds", 0, 1, 0, 0)
        record.startMicrophonesRecording("/home/nao/record.wav", 'wav',
16000, (0, 0, 1, 0))
        voice = True
        while voice == True:
            time.sleep(2)
            voice = memory.getData("SpeechDetected")

        record.stopMicrophonesRecording()
        leds.reset("FaceLeds")
        # print("END RECORDING")

        # SPEECH TO TEXT

        ssh = paramiko.SSHClient()
        ssh.load_system_host_keys()
        ssh.connect(NAO_IP, username=NAO_USERNAME, password=NAO_PASSWORD)

        sftp = ssh.open_sftp()
        localpath = "prueba.wav"
        remotepath = "/home/nao/record.wav"
        sftp.get(remotepath, localpath)
        sftp.close()
        ssh.close()

        r = sprec.Recognizer()

        harvard = sprec.AudioFile("prueba.wav")
        with harvard as source:
            audio = r.record(source)
        try:
            ema = (r.recognize_google(audio, language="es-ES"))
            if(ema=="terminar"):
                fin = True
                tts.say("Me lo he pasado bien, hablemos otro día.")
            else:
                resultado = getAnswer(ema, resps)
                tts.say(resultado)
        except:
            pass

def loadResp():
    global resps
    resps = {}
    lines = [line.rstrip('\n') for line in open("respuestas.txt")]
    for line in lines:
        rt=line.split("#")
        if(rt[1]in resps):
            resps[rt[1]].append(rt[0])
```

```
        else:
            resps[rt[1]] = [rt[0]]


#El robot se apaga
def ShutDown(NAO_IP,NAO_USERNAME,NAO_PASSWORD):
    sd = ALProxy("ALSystem", NAO_IP, 9559)
    sd.shutdown()

def main():
    global linequant
    global matrix
    downloadfile()
    linequant=creatematrix()
    loadResp()

    NAO_IP = "192.168.1.140"
    NAO_USERNAME = "nao"
    NAO_PASSWORD = "nao"
    pport=9559

    global INITIAL_STATE
    global DIALOGUE_STATE
    global LEARNING_STATE
    global SHUTDOWN_STATE
    global CHATBOT_STATE

    STATE=INITIAL_STATE
    while STATE!=SHUTDOWN_STATE:
        #print "State: " + str(STATE)
        if STATE==INITIAL_STATE:
            STATE=ChangeState()
        elif STATE==LEARNING_STATE:
            pregunta=getrRecordedText(NAO_IP, NAO_USERNAME,
NAO_PASSWORD,"Dime la frase que quieres que aprenda.")
            respuesta=getrRecordedText(NAO_IP, NAO_USERNAME,
NAO_PASSWORD,"Dime como deberia responder a eso.")
            #print pregunta
            #print respuesta
            #pr= "u: ("+pregunta+") "+ respuesta+"/n"
            new=[pregunta.encode("utf-8"),respuesta.encode("utf-8")]
            matrix.append(new)
            linequant=linequant+1
            os.remove("prueba.wav")
            STATE = INITIAL_STATE
        elif STATE==DIALOGUE_STATE:
            createfile(linequant)
            uploadfile()
            dialogue()
            STATE=INITIAL_STATE
        elif STATE == CHATBOT_STATE:
            chat(NAO_IP,NAO_USERNAME,NAO_PASSWORD)
            STATE = INITIAL_STATE
```

```
    createfile(linequant)
    uploadfile()
    os.remove("dialogo.top")
    ShutDown(NAO_IP, NAO_USERNAME, NAO_PASSWORD)
```

```
if __name__ == '__main__':
    """ Main entry point

        """
    parser = OptionParser()
    parser.add_option("--pip",
                      help="Parent broker port. The IP address or your
robot",
                      dest="pip")
    parser.add_option("--pport",
                      help="Parent broker port. The port NAOqi is
listening to",
                      dest="pport",
                      type="int")
    parser.set_defaults(
        pip="192.168.1.140",
        pport=9559)

    (opts, args_) = parser.parse_args()
    pip = opts.pip
    pport = opts.pport

    # We need this broker to be able to construct
    # NAOqi modules and subscribe to other modules
    # The broker must stay alive until the program exists
    myBroker = ALBroker("myBroker",
                        "0.0.0.0",  # listen to anyone
                        0,  # find a free port and use it
                        pip,  # parent broker IP
                        pport)  # parent broker port

    main()
```

Anexo B: doc2vec.py

```
# -*- coding: utf-8 -*-
from gensim.models.doc2vec import TaggedDocument
from gensim.models import Doc2Vec

import spacy
import codecs

frase=[]
palabrasrep=[]
repc=[]
etiq=[]
linecant=0
np=0
palabras = [u"que",u"qué",u"ser", u"querer" ]


#carga las preguntas que se utilizan para entrenar el modelo doc2vec
def load():
    global frase
    global linecant
    global etiq
    #guarda la cantidad de lineas del documento
    for line in open("chatbotnltk.txt"):
        linecant = linecant + 1
    #obtiene las lineas en "lines"
    lines = [line.rstrip('\n') for line in open("chatbotnltk.txt")]
    #a cada linea se le aplica, una separación de tipo y frase, además de
lematizar la frase.
    for line in lines:
        pregresp=line.split("#")
        pregunta=lema(unicode(pregresp[0], "utf-8"))
        etiq.append(pregresp[1])
        frase.append(pregunta)

#lematiza y tokeniza la frase entrante para obtener una versión
simplificada de las palabras
def lema(string):
    #carga el lematizador/tokenizdor
    nlp = spacy.load('es_core_news_sm')
    text1=string
    ema=[]
    #para cada token de la frase, en el caso de que no sea una palabra
innecesaria se guarda en la lista
    for token in nlp(text1):
        #countRep(token.lemma_)
        #print(token.text, token.lemma_, token.pos_)
        if(pInecesaria(token.lemma_)==False):
            ema.append(token.lemma_)
    return ema
```

```
#entrena el modelo usando las frases y etiquetas cargadas anteriormente
def train():
    global frase
    global etiq

    #etiq[i]
    tagged_data = [TaggedDocument(words=frase[i], tags=[etiq[i]]) for i
in range(linecant)]

    print tagged_data
    max_epochs = 100
    vec_size = 200
    alpha = 0.02

    model = Doc2Vec(
                    vector_size=vec_size,
                    alpha=alpha,
                    min_alpha=0.00025,
                    min_count=1,
                    dm =0,
                    workers=20
                    )

    model.build_vocab(tagged_data)


    for epoch in range(max_epochs):
        print('iteration {0}'.format(epoch))
        model.train(tagged_data,
                    total_examples=model.corpus_count,
                    epochs=model.iter)
        # decrease the learning rate
        model.alpha -= 0.0002
        # fix the learning rate, no decay
        model.min_alpha = model.alpha


    model.save("d2v.model")
    print("Model Saved")

#Dandole un string mira en la lista palabrasrep, en el caso de aparecer,
aumenta el numero de apariciones de este
#en caso contrario añade dicha palabra a la lista y marca 1 como el
numero de apariciones.
def countRep(string):
    global palabrasrep
    global repc
    global np
    if(string in palabrasrep):
        i=palabrasrep.index(string)
        repc[i]=repc[i]+1
    else:
```

```
        palabrasrep.append(string)
        repc.append(1)
        np=np+1

#imprime en el fichero "palabras.txt" las palabras y las repeticiones que
tienen para identificar cuales pueden ser innecesarias
def printRep():
    global palabrasrep
    global repc
    global np
    file = codecs.open("palabras.txt", 'w', 'utf8')
    for index in range(np):
        file.write(palabrasrep[index])
        file.write(":")
        file.write(str(repc[index]))
        file.write("\n")

#calcula el porcentaje de acietos del modelo "d2v.model"
def calcPorcent():

    global frase
    global etiq
    #carga el modelo
    model = Doc2Vec.load("d2v.model")
    i=0
    aciertos=0
    #pasa cada frase del entrenamiento por el modelo para obtener una
repsuesta e identificar si a acertado el tipo
    for f in frase:
        v1 = model.infer_vector(f)
        similar_doc = model.docvecs.most_similar([v1])
        """print (str(similar_doc[0][0]) + "A")
        print (str(etiq[i]) + "A")
        print(str(similar_doc[0][0]) == str(etiq[i]))"""
        if(str(similar_doc[0][0]) == str(etiq[i])):
            aciertos=aciertos+1
        i=i+1
    #cal cula el indice de aciertos
    percent=(float(aciertos)/float(i))*100
    print "Se ha acerado un " + str(percent) + "%"

#comprueba si la palabra "s" es necesaria o no, para ello toma en cuenta
las palabras que entás en la lista "palabras"
def pInecesaria(s):
    global plablras
    if s in palabras:
        return True
    else:
        return False

#imprime las frases lematizadas, tokenizadas y con palabras innecesarias
eliminadas en "frases.txt"
def printFrases():
```

```
    global frase
    file = codecs.open("frases.txt", 'w', 'utf8')
    for f in frase:
        for p in f:
            file.write(p)
            file.write(" ")
        file.write("\n")
    file.close()
if __name__ == '__main__':
    load()
    print "-------------------------------------------------"
    #printRep()
    printFrases()
    train()
    calcPorcent()
```

Anexo C: loadModel.py

```
# -*- coding: utf-8 -*-
import random

from gensim.models.doc2vec import Doc2Vec
from doc2vec import lema

#Dandole una pregunta, utiliza el modelo creado para identificar que tipo
de pregunta es
def getType(pregunta):
    #Cargar el modelo
    model= Doc2Vec.load("d2v.model")
    #lematizar la pregunta para que conincida más fácilmente con lo
entrenado
    test_data = lema(pregunta)
    #print test_data
    #aplicar la pregunta al modelo para obtener el tipo de pregunta
    v1 = model.infer_vector(test_data)
    #print("V1_infer", v1)

    # obtener el tipo
    similar_doc = model.docvecs.most_similar([v1])
    #print similar_doc
    num=similar_doc[0][0]
    return num

#dandole una pregunta y un conjunto de respuestas enlazadas a los tipos
de preguntas (dicc), obtiene una respuesta que corresponde ocn el tipo de
la pregunta
def getAnswer(pregunta,dicc):
    #obtiene el tipo utilizando el método "getType"
    n=getType(pregunta)
    ##############CON EL NUMERO OBTENER RESPUESTA##############
    #obtiene una respuesta aleatoria del mismo tipo que la pregunta, del
conjunto de respuestas "dicc"
    return random.choice(dicc[str(n)])



# to find vector of doc in training data using tags or in other words,
printing the vector of document at index 1 in training data

if __name__ == '__main__':
    dict = {"1": ["hola","buenos días"], "2": ["a"], "3": ["e"], "4":
["i"]}
    r="hola"
    resp=getAnswer(unicode(r,"utf-8"),dict)
    print resp
```