

TRABAJO FIN DE GRADO

Desarrollo de un sistema conversacional para robótica de servicios utilizando Amazon Alexa

DEVELOPMENT OF A CONVERSATIONAL SYSTEM FOR SERVICE ROBOTS
USING AMAZON ALEXA

Autor: David Ortega Rubiño

Titulación: Grado en Ingeniería Electrónica, Robótica y Mecatrónica.

Tutor: Cipriano Galindo Andrades

Cotutor: José Raúl Ruiz Sarmiento

Departamento: Ingeniería de Sistemas y Automática.

Área de conocimiento: Ingeniería de Sistemas y Automática.

26 de julio de 2019



**DECLARACIÓN DE ORIGINALIDAD DEL
PROYECTO/TRABAJO FIN DE GRADO**

D./ Dña.: David Ortega Rubiño

DNI/Pasaporte: 74743201D Correo electrónico: davidortega@uma.es

Titulación: Grado en Ingeniería Electrónica, Robótica y Mecatrónica.....

Título del Proyecto/Trabajo: Desarrollo de un sistema conversacional para
robótica de servicios utilizando Amazon Alexa.....

DECLARA BAJO SU RESPONSABILIDAD

Ser autor/a del texto entregado y que no ha sido presentado con anterioridad, ni total ni parcialmente, para superar materias previamente cursadas en esta u otras titulaciones de la Universidad de Málaga o cualquier otra institución de educación superior u otro tipo de fin.

Así mismo, declara no haber trasgredido ninguna norma universitaria con respecto al plagio ni a las leyes establecidas que protegen la propiedad intelectual, así como que las fuentes utilizadas han sido citadas adecuadamente.

En Málaga, a ..26... deJulio..... de 2019...

Fdo.: David Ortega Rubiño

Resumen

En este Trabajo Fin de Grado se lleva a cabo la aplicación del asistente virtual Amazon Alexa en la creación de un sistema conversacional que pueda utilizar un robot de servicios. Para ello se desarrolla una *skill* en la que la conversación que se genera es una elaboración de recetas explicadas paso a paso. Este diálogo recoge las respuestas del usuario desde la consola de desarrollador de Amazon y las preguntas que realiza el sistema (programación lógica) desde el propio ordenador, utilizando el entorno de trabajo de código abierto Jovo para la creación del proyecto, y el lenguaje JavaScript para la programación de dicha parte. El sistema conversacional se integra en ROS mediante un puente web, *rosbridge*, que permite realizar la llamada al servicio que controla al robot móvil, utilizando la librería *roslibjs*. Todo este sistema formado por la programación lógica y la *skill*, junto con el código para su comunicación por ROS, se integrará en un robot móvil de servicios.

Palabras clave

Asistente virtual, Alexa, ROS, Robótica de servicios

Resumen y palabras clave

Abstract

In this End-of-Degree Project, the virtual assistant Amazon Alexa is used to create a conversational system that can be integrated in a service robot. A skill is developed in which the conversation that takes place is the explanation of several recipes step by step. This dialog captures the user answers on the Amazon developer console, and the questions the system asks (logic programming), running on a computer. For that, we use the open source environment Jovo framework in the creation of the project, and the interpreted programming language JavaScript. This conversational system is integrated into ROS through a websocket, rosbridge, which allows us to call a service that controls a mobile robot, using the roslibjs library. All this system formed by the logic programming and the skill, with the code for its communication in ROS, will be integrated into a mobile service robot.

Key words

Virtual assistant, Alexa, ROS, Service robotics

Resumen y palabras clave

Índice

Resumen y palabras clave	1
1. Introducción	7
1.1. Aplicaciones de los asistentes virtuales en robótica	9
1.2. Objetivo	10
1.3. Estructura de la memoria	10
2. Alexa	13
2.1. Marco histórico	13
2.2. Amazon Alexa	15
2.3. Funciones	16
2.4. Privacidad	17
2.5. Alexa Skills Kit y Amazon's AWS Lambda	18
2.6. Alternativas a Alexa	18
3. Desarrollo y herramientas	21
3.1. Activación	21
3.2. Alexa Skills Kit	22
3.2.1. Estructura	23
3.2.2. Intents y Slots	24
3.2.3. Sample Utterances	25
3.2.4. Endpoint	26
3.3. Programación Lógica	26
3.4. Jovo	27
3.5. ROS	28
3.5.1. Diseño modular y entorno colaborativo	28
3.5.2. Componentes	28
3.5.3. Herramientas	29
3.5.4. Características específicas de los robots	30
3.5.5. Infraestructura de comunicación	31
3.5.6. Comunicación de ROS en JavaScript	31
4. Implementación	33
4.1. Desarrollo de Dorota	34

4.1.1. Consola de desarrollador	35
4.1.2. Programación lógica	37
4.2. Comunicación con ROS	39
5. Integración y pruebas con el robot	41
5.1. Instalación de herramientas	41
5.2. Pruebas	41
6. Conclusiones y mejoras futuras	45
Bibliografía	47
Anexo 1	49
1.1. Especificaciones técnicas	49
1.2. Jovo	50
Anexo 2	53
2.1. ROS	53
2.2. Roslibjs	55
Anexo 3	57

1. Introducción

Los asistentes virtuales se encuentran actualmente en auge y cada vez se ganan más adeptos, produciendo una auténtica en 2019 una revolución tecnológica. Siri llevaba siendo la reina de los asistentes virtuales desde su lanzamiento en 2011, hasta que en 2018 se abrieron camino las grandes apuestas de Google y Amazon, los productos Google Home y Amazon Echo (Figura 1), altavoces inteligentes al alcance de todos. Los dos asistentes de estas grandes empresas ya se encontraban en el mercado, Alexa desde 2014 y Google Assistant desde 2016, sin embargo, sus altavoces inteligentes no llegaron a España hasta 2018, con una diferencia de apenas 3 meses adelantándose el gigante Google. Este hecho supuso que en sólo un año se duplicase la cantidad de altavoces inteligentes vendidos en el mundo respecto los datos de 2017, datos que fueron incentivados por la facilidad que ofrecen los dos asistentes en cuanto a desarrollo de nuevas habilidades, que los vuelven mucho más atractivos que su hermana de Apple.



Figura 1: Google Home Mini y Amazon Echo Dot. Fuentes: [5] y [3].

Según Amazon estas nuevas habilidades, conocidas como *skills* “*añaden funcionalidades y permiten personalizar la experiencia con los dispositivos Amazon Echo y otros dispositivos con Alexa integrada*” [1]. Su desarrollo se puede realizar a través de sus respectivas plataformas, pero existe un entorno de trabajo, llamado Jovo [6], desarrollado en código abierto que permite una implementación intuitiva, con una sintaxis simple, todo desde un computador, sin tener que depender de los servicios en la nube y sus restricciones. Jovo fue el primero en permitir la facilidad de programar y almacenar el código

en tu propio ordenador, sin necesidad de conexión a internet, tanto para Amazon Alexa como para Google Assistant además de una completa personalización que lo vuelve muy atractivo para equipos profesionales y usuarios avanzados.

Este entorno permite la creación de un proyecto que contiene diferentes carpetas y archivos, entre ellos, un archivo tipo JavaScript en el que se desarrolla el código de la *skill*. Además Jovo necesita para su correcta instalación y funcionamiento del entorno Node.js que permite ejecutar JavaScript desde el lado del servidor, es decir, las operaciones las realiza el servidor en una conexión cliente-servidor. Es más, el lenguaje de programación JavaScript está especialmente indicado para este tipo de aplicaciones de escritorio debido a su eficiencia, velocidad y naturaleza basada en eventos que evita que se produzcan bloqueos y, al igual que Jovo, es de código abierto. Para poder comunicar una skill con un robot de servicios, siguiendo en el marco de los sistemas de código abierto, el candidato ideal es el entorno flexible de desarrollo de software robot ROS.

A través de todos estos entornos, es posible construir cualquier nueva habilidad que aumente la lista de aplicaciones que tienen los asistentes virtuales, principalmente incluyen la interacción de personas con sus dispositivos móviles, para facilitar su uso, o como controladores de los dispositivos inteligentes del hogar, ayudantes en comercios para resolver dudas a los clientes. En cuanto a la robótica de servicios los asistentes virtuales se incluyen para facilitar la comunicación, ya que se utilizan para colaborar con el ser humano en multitud de tareas, desde tareas del hogar como aspirar hasta situaciones menos amigables como labores de seguridad y vigilancia, o incluso como simple ocio. También se están utilizando como robots de acompañamiento para personas mayores que les recuerden eventos importantes o la medicación que deben tomar, ya que estos robots de servicios aportan la sensación de compañía, acompañándolos de un aspecto amigable y la capacidad de desplazarse, puesto que normalmente son además robots móviles, rodeando al usuario de un elemento móvil con el que interactuar.

Un ejemplo es el robot Giraff (Figura 2), un robot de servicios provisto de una base móvil con motores sobre la que se sostiene un ordenador y conjunto de sensores. Cuenta con una pantalla en la parte superior para que quede a una altura cómoda para el usuario, sujetado a la base mediante un “cuello”, de ahí que su nombre sea una alteración de la palabra Jirafa en inglés (*giraffe*) sin la e. El grupo de investigación MAPIR (MAchine Perception and Intelligent Robotics) de la Universidad de Málaga trabaja con estos robots en sus diferentes líneas de investigación y gracias a su arquitectura basada en ROS pueden comunicarse con personas y evitar obstáculos mientras navegan, utilizando un sensor hokuyo para localizarse y detectar obstáculos cercanos, además de una cámara RGB

para la interacción con personas y otra RGB para los obstáculos 3D.



Figura 2: Robot Giraff del grupo MAPIR utilizado para el desarrollo de este trabajo.

1.1. Aplicaciones de los asistentes virtuales en robótica

Entre las principales ventajas que aportan los asistentes virtuales en robótica podemos destacar las siguientes:

- **Comunicarse fácilmente:** la más obvia es dotar a los robots de voz para que la comunicación con el usuario sea de la manera más natural posible.
- **Controlar mediante comandos de voz:** tanto los movimientos del robot como el funcionamiento de los dispositivos de los que disponga como cámaras, sensores, etc.
- **Asistentes en comercios:** ya se han implantado en algunos comercios de América y Japón para que sean los ayudantes de sus clientes en materia de información y resolución de dudas. O incluso interpretar las expresiones faciales y comportamiento para que su respuesta sea más eficaz.
- **Compañía en el hogar:** desde mantener una conversación amena con aquellas personas que estén solas demasiado tiempo, hasta ser un ayudante de enfermería que recuerda al usuario qué medicamento debe tomar, a qué hora y la dosis correcta. Controlar cualquier dispositivo inteligente e incluso hacer sugerencias para evitar la monotonía.

1.2. Objetivo

El objetivo de este trabajo es el desarrollo de un sistema conversacional utilizando el asistente virtual de Amazon Alexa. Para ello la primera fase consiste en crear una skill en la consola de desarrollador que facilita Amazon, implementando una conversación para que el asistente explique paso a paso la elaboración de una receta. Se ha elegido esta temática por su generalidad y abanico de posibilidades, aunque, por supuesto, se podría haber elegido cualquier otra temática. En esta consola se programarán las posibles respuestas del usuario.

A continuación mediante JavaScript se escribirá el código que controlará las preguntas y respuestas que realice Alexa durante la conversación. Este código se realizará a través del entorno Jovo, creando un proyecto que se alojará en la propia computadora, para que se pueda comunicar con la nube y elegir el desarrollo de la conversación en función de las respuestas del usuario. La comunicación con el robot se llevará a cabo mediante ROS, utilizando el puente web que proporciona rosbridge y la librería roslibjs para realizar la llamada al servicio que controlará la tarea de hablar con el usuario y las órdenes que se le den al robot, como por ejemplo avisar a alguien de que la comida está a punto.

Por último se comprobará el correcto funcionamiento de todo el sistema en un robot móvil Giraff, ejecutando el proyecto y comprobando que las respuesta de Alexa se transmiten a través del servicio al robot y éste habla o realiza la acción de movimiento implementada. Por cuestiones de privacidad del usuario se capturará la voz solo cuando lo elija mediante pulsaciones en algún tipo de dispositivo, como pudiera ser un teclado.

1.3. Estructura de la memoria

La memoria de este Trabajo Fin de Grado se estructura de la siguiente forma:

- Una primera parte que se corresponde con la información sobre la historia de los asistentes virtuales, así como una introducción a Alexa y datos relevantes sobre su creación, desarrollo y funciones, además de los servicios que ofrece para desarrolladores como Alexa Skills Kit o Amazon AWS Lambda, y las fundaciones que apoya.
- Una segunda en la que se explica en profundidad el desarrollo de una skill utilizando la consola de desarrollador de Amazon, comentando los elementos que la componen y cómo se utilizan, junto con el código abierto de JavaScript Node.js, el entorno

con el que se crea la aplicación de voz, Jovo, y el sistema operativo robótico, ROS, que se utiliza para la comunicación con el robot.

- La última parte explica la implementación del sistema completo, el proceso que se ha llevado acabo tanto para la skill, como para la comunicación mediante ROS. Seguida de las conclusiones y posibles mejoras que se podrían implementar en el futuro. Una serie de anexos, redactados en forma de guías para el usuario, en los que se muestra paso a paso la instalación de Jovo, de ROS junto con la librería específica para Java y el puente de comunicación, y la ejecución de la skill que da pie a la conversación desarrollada.

2. Alexa

Hoy en día interactuamos con la tecnología de la manera más natural posible: hablando. Alexa es el servicio de voz de Amazon, basado en tecnología de la nube, disponible en sus dispositivos y en otros dispositivos inteligentes, ubicado en su propia nube. Con Alexa se pueden construir experiencias de voz naturales que ofrecen a los usuarios una forma más intuitiva de relacionarse con la tecnología que utilizan diariamente. Lo que se conoce como asistente virtual, un agente de software en el que la interacción entre humano y máquina es de la forma más natural posible, hablando, y que realiza ciertas tareas, que pueden estar automatizadas o no, que suponen una ayuda para el usuario.

2.1. Marco histórico

El primer asistente personal inteligente conocido fue creado por un equipo de personas del SRI International elegidas por Adam Cheyer, mánager del proyecto CALO, que empezó en 2003 y que se mantuvo en desarrollo durante 5 años hasta su finalización en 2008. Dicho proyecto fue impulsado por el servicio militar para crear una inteligencia artificial, de ahí el nombre que forman las siglas de “Asistente cognitivo que aprende y organiza (*Cognitive Assistant that Learns and Organizes*)”. Este asistente no es otro que Siri que sin embargo no fue lanzado por Apple hasta Octubre de 2011.

Pero volviendo al inicio, IBM presentó en la Feria Mundial de Seattle de 1962, la que se introdujo como la primera herramienta que permitía ejecutar reconocimiento digital de voz, siendo capaz de reconocer 16 palabras y los dígitos del 0 al 9, *IBM Shoebox* que había sido lanzada al mercado el año anterior. Este hito se consiguió casi 20 años antes de que la misma compañía introdujese la primera computadora personal. Diez años después de esta presentación, en 1972, en Pittsburg, Pensilvania, la Universidad de Carnegie Mellon completó su programa *Harpy* capaz de entender alrededor de 1000 palabras, gracias a la colaboración de la agencia DARPA del Departamento de Defensa de Estados Unidos.

En la década de 1990 Dragon Systems puso en venta para el consumidor el primer producto de reconocimiento de voz que desarrolló para Microsoft Windows *Dragon Dictate*, misma década en la que dicha compañía introducía su famoso ayudante de Office, *Clippy* que recibía su nombre por tener forma de clip animado y que estuvo disponible para Microsoft Office hasta la llegada en 2001 de Windows XP que contaba ya con un mejorado sistema de reconocimiento de voz para *Office XP*.

Y justo 10 años después, durante la presentación del iPhone 4S el actual gigante Apple, lanzó el primer asistente virtual digital instalado en un teléfono inteligente, *Siri*,

en octubre de 2011, después de adquirir la compañía encargada de su desarrollo sólo un año antes, desde entonces cada año han aparecido nuevos asistentes virtuales. En 2012, tras los rumores que habían surgido sobre las mejoras en el producto Google Voice Search, Google lanzó el primer asistente personal inteligente para dispositivos Android, *Google Now* soportado por primera vez por el Galaxy Nexus con la versión de Android 4.1 “Jelly Bean”. 2013, fue el año en el que Microsoft presentó en su conferencia de desarrolladores en San Francisco, su asistente virtual *Cortana*, planteado para su inclusión en los sistemas operativos de Windows y Windows Phone, que llevaban desarrollando desde 2009.

En noviembre del 2014 la compañía de la A a la Z (Amazon) lanzó el asistente objeto de estudio de este trabajo, junto con los altavoces inteligentes que empezaron siendo accesibles para usuarios Prime. Al año siguiente tanto Alexa como Cortana se expandieron; la expansión de Alexa se dio a través de la introducción de *Alexa Skills Kit* (ASK), un conjunto de facilidades con el que añadir Skills a Alexa de forma sencilla y rápida. Y Cortana amplió sus funcionalidades en diciembre cuando se añadió a las plataformas móviles.

En marzo de 2016 la compañía SoundHound que hasta el momento estaba vinculada al sonido, su aplicación se utilizaba para el reconocimiento de canciones, lanzó su propio asistente virtual por voz, *HOUND*, que pretendía competir con el servicio prestado por Google. Pero dos meses después fue Google quien introdujo la mayor apuesta hasta el momento en los asistentes virtuales que había desarrollado, el *Google Assistant* que empezó siendo parte de la aplicación de mensajería Google Allo, exclusivo para los usuarios de los teléfonos inteligentes Pixel y Pixel XL, hasta que en febrero de 2017 se empezó a implementar en otros dispositivos Android, y que actualmente tienen instalado por defecto.

Para terminar, en octubre de este mismo año, Samsung Electronics, adquirió la empresa Viv Labs que en mayo había debutado en este mercado con un asistente propio, *Viv*, desarrollado en código abierto, lo que permitía acomodar complementos externos escritos para trabajar con el asistente. Todo ello de la mano de los mismos desarrolladores que llevaron a cabo el primer asistente personal inteligente del mercado (Siri) y que llevaban trabajando en este nuevo proyecto desde 2012.

Todo esto queda reflejado en la línea temporal de la Figura 3.

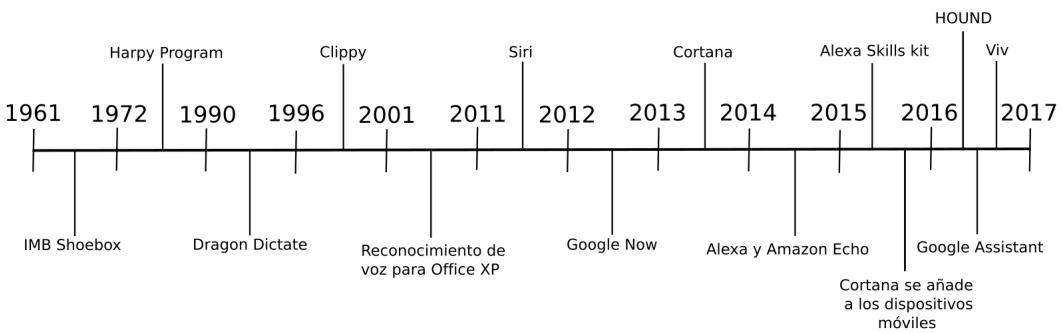


Figura 3: Línea temporal de los asistentes virtuales. (Ilustración propia).

2.2. Amazon Alexa

Alexa es el nombre del asistente virtual desarrollado por Amazon, cuyo desarrollo vino inspirado por el sistema de abordo de la nave *Enterprise* de las películas y series de televisión del universo *Star Trek*. Fue presentado por primera vez en 2014 junto con los dispositivos *Amazon Echo* y *Amazon Echo Dot*, altavoces inteligentes que han sido desarrollados por el *Lab126*, una compañía propiedad de Amazon.com que se encarga de la investigación, desarrollo y hardware computacional, la misma división que creó la línea de tablets y e-readers *Kindle*.

Se escogió este nombre por la gran consonancia de la letra X que facilita un reconocimiento con mayor precisión, aunque también se afirma que fue elegido en honor a la mítica Biblioteca de Alejandría (en inglés *Alexandria*). Tiene la habilidad de reproducir música, hacer listas, programar alarmas y recordatorios, dar información sobre el tráfico, tiempo, deportes o las propias noticias en tiempo real, y su mayor atractivo, controlar los dispositivos inteligentes de las casas como un sistema domótico. Todas estas habilidades pueden ser expandidas por los usuarios instalando “Skills”, aplicaciones desarrolladas por los propios usuarios o por terceros que permiten funcionalidades adicionales como resúmenes de noticias o incluso juegos exclusivamente por audio. De hecho, durante la conferencia Amazon Web Services Re:Invent de Las Vegas se anunció la posibilidad de recibir ingresos por el desarrollo de estas *skills*, además de Alexa para empresas, un servicio premium capaz de utilizar Alexa para programar reuniones o fusionar llamadas de conferencias.

En enero de 2017 se celebró la primera Conferencia Alexa, un encuentro para desarrolladores y entusiastas de todas partes del mundo que tuvo lugar en Nashville, Tennessee; y en mayo del año pasado la compañía declaró que 35.000 nuevas viviendas construidas a lo largo del 2018 contaría con el asistente incluido.

La activación de los dispositivos en los que se encuentre Alexa puede realizarse mediante una única palabra clave (como Alexa o Echo), sin embargo en la aplicación móvil tanto de Android como de iOS, es necesario que el usuario pulse un botón para activar el modo escucha. Esta aplicación es la que se utiliza para instalar las skills en los dispositivos, aunque hay algunas que no necesitan instalación y se pueden utilizar pidiéndole al dispositivo que las abra. También mediante ella se pueden programar con mayor precisión las alarmas, organizar las listas o revisar los controles para la música. Actualmente la comunicación e interacción con el asistente se encuentra disponible en inglés, alemán, francés, japonés, italiano, español y portugués, pero se sigue trabajando para aumentar el catálogo de idiomas.

2.3. Funciones

A través de las skills es como Alexa consigue su versatilidad en cuanto a funciones que realiza, y como cualquier usuario puede desarrollar una skill, estas funciones son muy variadas yendo desde la domótica hasta los pedidos a domicilio. Una capacidad interesante son las rutinas, se pueden configurar en la aplicación y llevan a cabo la ejecución de una serie de funciones al decir un conjunto de palabras concreto. El número de funciones que es capaz de realizar Alexa se quintuplicó en el año 2016 y dos años después, en marzo de 2018, se habilitó una opción configurable llamada “Brief Mode” que hace que la respuesta del dispositivo que confirma la recepción de una orden sea un pitido en lugar de su característico “Okay”.

Las funciones que permiten dar un informe meteorológico o poner una emisora de radio corren a cargo de AccuWeather y TuneIn respectivamente, pero para la gran mayoría se puede elegir el proveedor, como el resumen de noticias diario que requiere tener instalada la skill propia del proveedor como RTVE o ABC (hay infinidad de ellos), o la reproducción de música que se realiza a través de los servicios de *streaming* Amazon Music, Spotify o Deezer (los únicos disponibles en España). Los usuarios que tengan Amazon Prime o Amazon Music Unlimited pueden acceder a millones de canciones, listas de música o emisoras de artistas, además si se tiene un Fire TV se puede controlar la reproducción de contenido multimedia en la televisión.

El control de dispositivos inteligentes se lanzó 8 de abril de 2015, permitiendo a los propios usuarios crear sus propias skills de domótica usando el Alexa Skills Kit, aunque en la mayoría de los casos se necesita tener instalada la aplicación del fabricante del propio dispositivo para conectar el dispositivo a la red WiFi y posteriormente a la aplicación de

Alexa.

Los pedidos a domicilio son posibles desde 2017, y al igual que con el resumen de noticias, hay que tener la skill del servicio al que se vaya a realizar el pedido. Dependiendo del país hay disponibles diferentes opciones, en España tenemos Foster's Hollywood, Telepizza, Just Eat, Domino's Pizza y Burguer King. Gracias a la incorporación de Amazon Key, que permite a Alexa controlar las cerraduras, y Amazon Cloud Cam, los repartidores pueden desbloquear las cerraduras inteligentes y dejar el pedido dentro de la vivienda

En el ámbito de información deportiva, se pueden escuchar las noticias de diferentes ligas que se configuran añadiéndolas a una lista en la sección *Alexa's Sports Update* de la app, que permite hasta ligas.

La última función que viene integrada de forma predeterminada en el asistente de Amazon es la realización de llamadas o envío de mensajes a través de la aplicación o entre dispositivos Echo siempre que el número esté asociado a una cuenta Amazon. Los mensajes que se envían a través de la aplicación son recibidos en forma de texto, pero si se envía un mensaje a través de un Echo, el destinatario lo recibe en forma de mensaje de voz, y por supuesto, no se pueden adjuntar fotos ni vídeos. Esta opción tiene el problema de la privacidad ya que cualquiera puede escuchar los mensajes a través del Echo, ya que no hay forma de reconocer a la persona que está escuchando, y no existe la opción de protegerlos mediante contraseña o PIN. Al igual que en cualquier dispositivo móvil, se puede bloquear temporalmente la alerta de llamadas y mensajes con la opción de No Molestar.

2.4. Privacidad

En lo referente a la invasión de la privacidad hay mucha preocupación por la facilidad que tiene Amazon para acceder a conversaciones privadas y al registro de cualquier sonido que identifique quién está en el mismo lugar que el dispositivo con Alexa, debido a su constante reconocimiento del audio. Esto se da ya que necesita escuchar en todo momento para poder actuar cuando reconozca la *wake word*, único momento en el que se transmiten grabaciones según asegura la compañía. Estas grabaciones se almacenan en la nube y se utilizan para mejorar la respuesta en futuras órdenes, y se pueden borrar fácilmente desde la app siempre que esté asociada a una cuenta de usuario.

De la misma forma en la propia cuenta de usuario del servicio de venta online, se necesita tener guardada una dirección postal, que puede utilizar el asistente cuando necesita utilizar la localización como por ejemplo cuando se le piden recomendaciones de locales

o restaurantes cercanos, o cualquier dirección que requiera de mapeo.

Todas estas grabaciones digitales que Amazon almacena de los usuarios están sujetas a la ley de protección de datos y demanda por cumplimiento de la ley, pudiendo ser utilizadas bajo citación en un proceso judicial, todo esto en un entorno de alta seguridad que evita filtraciones y venta de datos.

2.5. Alexa Skills Kit y Amazon's AWS Lambda

Alexa Skills Kit es un servicio que ofrece Amazon a los desarrolladores para poder construir y publicar Skills de cualquier tipo, que posteriormente pueden ser publicadas para incluirlas en cualquier dispositivo compatible. Cuenta con una herramienta que sirve de guía para desarrollar ciertas skills, *Blueprints*, que se lanzó en abril de 2018 y también una *Smart Home Skill API* para que los fabricantes de dispositivos inteligentes de hogar integren en sus productos, permitiendo que puedan ser controlados con Alexa.

Gran parte de las skills funcionan ejecutando código que se encuentra en la nube, a través del servicio *Amazon's AWS Lambda*. Esta plataforma se introdujo en 2014, está dirigida a eventos y ejecuta el código en función a los mismos, controlando automáticamente los recursos de computación necesarios. Los lenguajes compatibles son C#, Java, Node.js, Python y Go, aunque se puede hacer compatible otro lenguaje solicitándolo. Se diseñó para facilitar la subida de imágenes y vídeos a Amazon S3, actuar en función a las lecturas de un sensor de un dispositivo IoT, reducir la carga de los servidores back-end cuando no estén siendo usados, etc. AWS Lambda es gratuito en un uso de cantidades limitadas durante los 12 primeros meses, después en función de los recursos que sean necesarios para la computación se aplica una tarifa.

2.6. Alternativas a Alexa

Existen algunas alternativas al uso de Alexa que también ofrece Amazon, se trata del AVS (Alexa Voice Service) y de Amazon Lex. El primero permite integrar las capacidades de voz en dispositivos ya desarrollados, proveyendo de reconocimiento automático del habla y entendimiento natural del lenguaje. Los productos que lo integren tienen acceso completo a las capacidades de Alexa que se encuentran en constante crecimiento y a todas las skills, y es completamente gratuito.

El segundo llegó un poco más tarde, pero fue revolucionario, a finales de 2016 se anunciaba que todo el mundo podría acceder a la tecnología que había tras Alexa de reconocimiento de voz y entendimiento del lenguaje, Amazon Lex. Al contrario que AVS,

Lex permite que los usuarios no interactúen con el asistente de Amazon en sí, sino que puedan construir cualquier tipo de asistente o interfaz conversacional, chatbots que pueden interactuar de manera similar a Alexa en una conversación. La respuesta para estos chatbots con Lex puede ser definida por los usuarios en la consola de AWS desde febrero de 2018.

Alexa

3. Desarrollo y herramientas

En este capítulo se va a describir el desarrollo de una skill, su funcionamiento y la estructura que sigue, además de las herramientas necesarias para hacer funcionar todo el sistema en el robot móvil.

3.1. Activación

Un asistente virtual es un software que puede realizar tareas o servicios para un usuario basándose en órdenes dadas por voz. Algunos asistentes son capaces de interpretar voces humanas y responder mediante sintetizadores de voz, de esta forma, los usuarios pueden realizar preguntas a sus asistentes, controlar los dispositivos de automatización de sus casas o reproducir algunos archivos multimedia. Por otro lado también son capaces de controlar otras tareas básicas como el correo electrónico, la lista de cosas que hacer o las entradas del calendario de sus dispositivos inteligentes.

Para que estos asistentes escuchen al usuario normalmente se necesita una palabra de activación. En el caso del asistente de estudio en este trabajo, esa palabra es Alexa, elegida por la sonoridad de la letra “x”, pero aparte de esta palabra, la frase que el asistente escucha tras la palabra de activación, está compuesta por diferentes elementos que se muestran en la figura 4.



Figura 4: Análisis sintáctico de una orden. (Ilustración propia).

Cada palabra está relacionada con una parte de la frase, que sirve un propósito en el procesamiento del código que se realiza en la nube.

- **Principio de frase:** Es la palabra que activa la skill, puede ir acompañada del nombre de invocación e incluso de una petición concreta, aunque no necesariamente tiene que decirse, también se puede invocar la skill sólo con el nombre de invocación.
- **Nombre de invocación:** Esta palabra o grupo de palabras elige la skill. Por lo tanto se debe escoger un nombre que sea fácilmente recordable, reconocible, intuitivo y concordante con la función de la skill, además de único y original puesto que no

se puede escoger un nombre que ya esté en uso. Debe respetar los derechos de propiedad intelectual de una entidad o persona [8].

- Petición: Con esta palabra o conjunto de palabras se consigue que la skill realice directamente lo pedido sin tener que pasar por la parte conversacional de la misma. Únicamente aparece en una de las 3 formas de activar una skill, tal como se ha visto en el punto sobre el principio de frase. Las peticiones (que en inglés se denominan *utterances*) se codifican en el “Alexa Skills Kit”, donde se desarrollan todas las skills que se realizan para Alexa.

3.2. Alexa Skills Kit

Alexa está compuesta entre otras cosas por un conjunto de capacidades conocidas como *skills* que permiten reproducir música, responder a preguntas, consultar el pronóstico del tiempo o el resumen de noticias, e incluso desafiar al usuario con juegos o rompecabezas. Estas skills y otras completamente nuevas pueden ser desarrolladas por los clientes en el Alexa Skills Kit (ASK).

Todas están formadas por una interfaz de voz de usuario *VUI* (front-end) y una programación lógica que implementa la interfaz, hospedada en un servicio web (back-end). El comportamiento que se produce cuando un usuario interactúa con un dispositivo con Alexa es el siguiente (Figura 5):

1. El dispositivo (*front-end*) captura el dialogo del usuario.
2. Se transmite al servicio de Alexa en la nube donde se trata en JSON.
3. Alexa reconoce el intención que representa la petición y la skill que se menciona.
4. La skill accede al intención mencionado y devuelve la información codificada en el *back-end*.
5. Esta respuesta pasa de nuevo por la nube para enviarla al dispositivo desde en que se ha iniciado la conversación.
6. El dispositivo devuelve la respuesta pertinente al usuario mediante un text to speech (TTS).

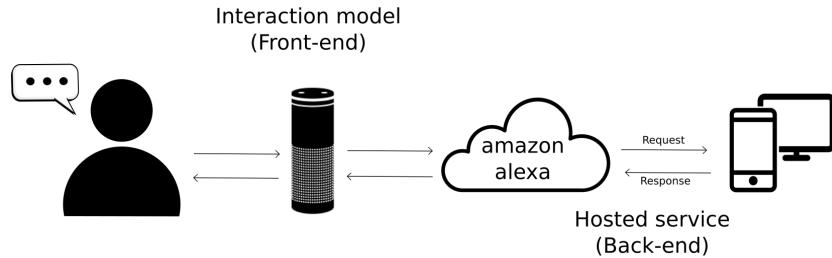


Figura 5: Flujo de comunicación en la ejecución de una skill. (Ilustración propia).

3.2.1. Estructura

Para utilizar el ASK hay que acceder a la consola de desarrollador [1], lo que permite crear diferentes tipos de skills, la mayoría con un modelo predeterminado para realizar acciones específicas como las indicadas anteriormente. Pero la más interesante es la opción personalizada (Custom), que da lugar a un sin fin de posibilidades siendo la mayor baza de este asistente inteligente. En cualquiera de los casos se puede seleccionar el nombre de publicación de la skill, además de poder cambiarlo posteriormente, y el idioma por defecto en el que se va a desarrollar, pudiendo siempre añadir idiomas adicionales.

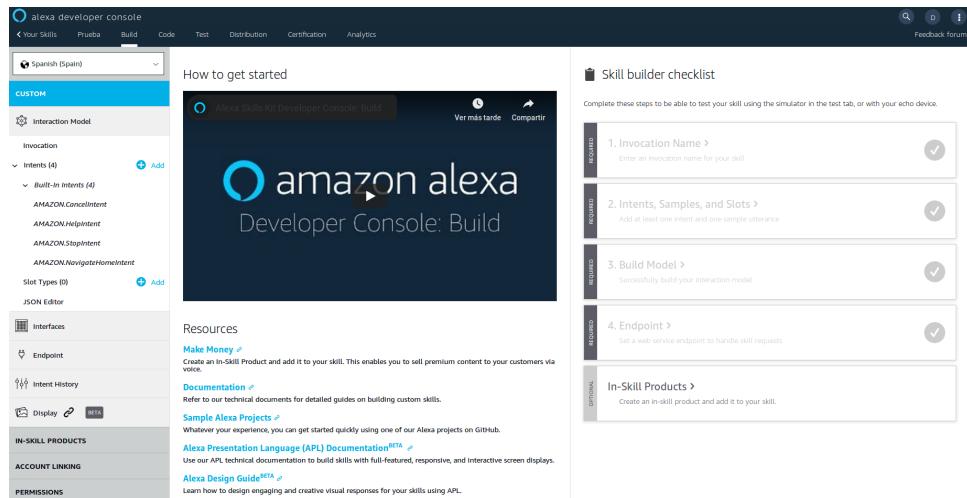


Figura 6: Elementos de una skill personalizada al crearla en la consola de desarrollador. Fuente: [1].

Cuando se diseña y construye una skill personalizada, se crea lo que aparece en la Figura 6:

- *Intents* (intenciones) que representan las acciones que se pueden realizar, la funcionalidad principal de la skill.

- Un conjunto de expresiones (*utterances*) que especifican las palabras o frases que puede decir el usuario para invocar esos *intents*, y que se asignados a los mismos.
- Un nombre de invocación que identifica la skill, que el usuario menciona para activarla.
- Un conjunto de imágenes, archivos de audio y archivos de vídeo opcionales que deben almacenarse en un sitio de acceso público para que cada elemento sea accesible mediante una URL única.
- Un servicio en la nube que reconoce los *intents* como solicitudes estructuradas y actúa sobre ellos, accesible a través de internet. Se tiene que proporcionar un *backend* para el servicio al configurar la skill.
- Una configuración que reúne todo lo anterior para que Alexa pueda interpretar las solicitudes para la skill, creada en la consola de desarrollador.

3.2.2. Intents y Slots

Un intent es un posible comportamiento de la skill que sirve para entrenar el modelo del lenguaje, puede tener asignados argumentos llamados slots (Figura 7), que son los elementos que capturan las variables. Los slots deben incluir todas las palabras que se quieren hacer entender, aunque no se vayan a utilizar, porque si se dice una palabra que no se encuentra registrada en un slot, no se entenderá y provocará que Alexa repita la pregunta.

Existen una serie de intents y slots predefinidos que se pueden incluir en cualquier skill para que realice algunas tareas básicas. De hecho cuando se crea una nueva, vienen incluidos algunos de ellos que son: AMAZON.FallbackIntent, AMAZON.CancelIntent, AMAZON.HelpIntent, AMAZON.StopIntent y AMAZON.NavigateHomeIntent, accesibles en todo momento durante la ejecución de la skill.

El nombre de un intent sólo puede contener letras mayúsculas o minúsculas, teniendo especial cuidado de no solaparlo con el nombre de un slot, por ello los predefinidos incluyen la palabra Intent. Por ejemplo, un intent para saludar podría llamarse *SaludoIntent* mientras que el slot podría ser *saludo*, que incluiría palabras como: hola, buenas, qué tal, buenos días, etc. Si se indica que uno de los slots es requerido para completar el intent, el servicio tiene que pedir ese dato específico para poder continuar.

Slot Types / name

VALUE	ID (OPTIONAL)	SYNONYMS (OPTIONAL)
Juan de Dios	Enter ID	Add syn +
Carmen Mari	Enter ID	Add syn +
Nuria	Enter ID	Add syn +
Jesus	Enter ID	Add syn +

Figura 7: Slot de nombres. Fuente: [1].

3.2.3. Sample Utterances

Las Utterances son las posibles frases a decir por los usuarios cuya función principal es entrenar a Alexa, por eso cada intent tiene varias utterances asociadas y son estas las que contienen los slot. De esta forma cuantas más utterances, mejor enterá al usuario y por lo tanto, mejor funcionamiento tendrá la skill (Figura 8).

Sample Utterances (5)
What might a user say to invoke this intent?
+ It's {names}
{names}
I am {names}
My name is {names}
I'm {names}

Figura 8: Ejemplo de utterances. Fuente: [1].

3.2.4. Endpoint

Nada del procesamiento del código de una skill se realiza en el dispositivo, se hace en la nube. Por eso el endpoint donde se almacene la programación lógica de la skill tiene que:

- Ser accesible a internet.
- Acatar la interfaz del servicio ASK.
- Usar HTTP a través de SSL/TLS en el puerto 443.
- Ser un servicio web o AWS Lambda.

Si se opta por un servicio web, se debe configurar un certificado SSL, mientras que si se utiliza AWS Lambda, ya tiene integrados los certificados y compatibilidades de seguridad. Este servicio facilita un identificador conocido como Amazon Resource Name (ARN) que se debe introducir en la interfaz de voz de usuario; con el servicio web se tiene que introducir la URL.

El endpoint tiene 8 segundos para responder tanto al usuario como a Alexa. Al usuario se le dan dos oportunidades, si no responde en 8 segundos en ambas se acaba la conversación.

3.3. Programación Lógica

La programación lógica es el código que al ser ejecutado controla el funcionamiento de la skill junto con la interfaz de voz de usuario. Dependiendo del tipo de skill se requieren diferentes servicios, todos ellos basados en la nube que controla las solicitudes para cada uno. Los tres tipos de skills disponibles para desarrollar en español son:

- Una *Smart Home* skill que controle los dispositivos inteligentes de la casa como luces, termostatos, altavoces, etc, gestionada por la API Smart Home Skill sin necesidad de toda la configuración del apartado anterior.
- Una skill personalizada, la baza más importante de Alexa, que permite al desarrollador realizar una aplicación con total libertad, que sea capaz de lo que se le ocurra. Junto con la anterior, la programación lógica se puede realizar en el servicio de Amazon AWS Lambda utilizando Node.js, Java, Python, C#, o Go; o, en un servicio web hospedado en un proveedor de alojamiento en la nube, que acepta las solicitudes a través de HTTPS.

- Una *Flash Briefing* skill, que proporciona el resumen de noticias, controlada por la API Flash Briefing, sin tener que configurar intents, slots o utterances. Desde la aplicación se tienen que seleccionar los canales que proporcionan el contenido, y este puede ser de audio que reproduzca Alexa para el cliente o de texto, poseyendo los derechos para poder distribuirlo.

3.4. Jovo

Jovo es el primer entorno de código abierto que permite crear aplicaciones de voz para Amazon Alexa y Google Assistant donde la programación lógica se aloja en un computador que, al ejecutar el proyecto, se conecta con la nube del asistente [6] (Figura 9). Su objetivo principal es hacer el entorno lo más fácil posible para principiantes, con una sintaxis simple y clara, manteniéndolo personalizable y ampliable para equipos profesionales y usuarios avanzados. Incluyendo potentes integraciones que ayudan a construir aplicaciones profesionales más rápido, además de desarrollo multiplataforma.



Figura 9: Integración de Jovo. Fuente: [6].

Las principales características del entorno Jovo son:

- Jovo Framework: desarrollar aplicaciones de voz para Amazon Alexa y Google Assistant.
- Jovo CLI: crear, generar e implementar proyectos Jovo.
- Jovo Webhook: desarrollar y depurar aplicaciones de voz en su computadora local.
- Jovo Debugger: probar y depurar aplicaciones de voz en tu navegador.
- Jovo Language Model: un modelo de lenguaje consolidado que se puede convertir en Alexa Interaction Models y Dialogflow Agents.

3.5. ROS

El *Robot Operarting System* (ROS) es un marco de trabajo flexible para escribir software robótico. Es una colección de herramientas, bibliotecas y acuerdos que tienen como objetivo simplificar la tarea de crear un comportamiento robótico complejo y robusto en una amplia variedad de plataformas robóticas, ya que crear software robótico verdaderamente robusto y de propósito general es difícil.

ROS se construyó para dar facilidad y plataforma a la colaboración internacional en el desarrollo de software robótico, ya que tratar de resolver los problemas desde la perspectiva del robot es una tarea que difícilmente puede realizar un individuo por su cuenta. De tal forma que diferentes laboratorios colaboren entre sí para ayudarse mutuamente en sus proyectos, teniendo cada uno expertos en temas diferentes que puedan complementar el trabajo de los demás.

3.5.1. Diseño modular y entorno colaborativo

Se optó por un diseño distribuido y modular, de tal forma que los diseñadores elijan las partes que les son útiles y las que quieren implementar. Esto ha provocado que haya una gran cantidad de paquetes (más de 3000 en el último recuento), variados en sus funciones, que aportan valor al sistema principal de ROS. Toda esta infraestructura compartida por los usuarios, proporciona herramientas de desarrollo, bibliotecas útiles y acceso a controladores de hardware, además de la oportunidad de colaborar con especialistas de todo el mundo.

3.5.2. Componentes

La comunicación de ROS se realiza mediante el mecanismo *peer to peer* (publicación-suscripción) en el que el maestro establece una comunicación nodo a nodo entre todos aquellos que publican, suscriben o usan un servicio. El maestro tiene todos los procesos registrados y nombrados, controla la comunicación por los topics y las actualizaciones del servidor de parámetros.

Los nodos son la base de la programación de ROS ya que la mayoría del código cliente tiene esta forma y están organizados en paquetes. Las publicaciones y suscripciones se realizan a través de los topics, que son los canales por donde reciben y envían la información los nodos. Para poder enviar un mensaje, el nodo tiene que publicar en ese topic y para recibirlo tiene que estar suscrito. En realidad un nodo no sabe qué nodos envían y cuales reciben, solo el topic que transmite la información.

Un servicio (Figura 10) permite la recepción de una respuesta cuando se realiza una petición mediante el envío y recepción de un par de mensajes, cosa que los topics no son capaces de hacer, el modelo de publicar/suscribir es unidireccional. Gracias a los servicios cuando un nodo solicita una acción a otro, éste le devuelve un único resultado, y al igual que los mensajes, no pasan a través del maestro. Los servicios se utilizan en operaciones que tienen principio y fin, y se les llama de uno en uno. No se cuenta con servicios estándar por lo que tienen que ser creados por los usuarios, se definen usando archivos *srv* y un nombre *string*.

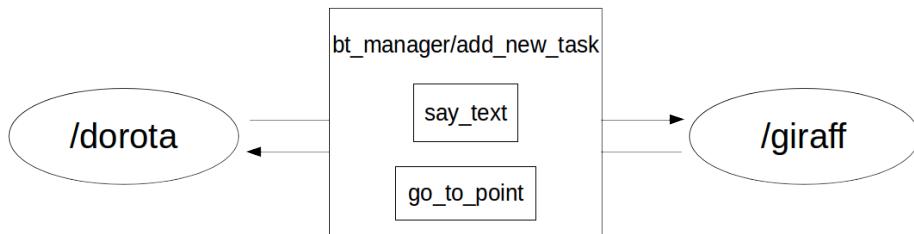


Figura 10: Comunicación mediante el servicio *Add_Task* con el robot. (Ilustración propia).

Existe una base de datos que incluye toda información que no se someta a cambios o se mantenga prácticamente invariable, conocida como servidor de parámetros.

Para trabajar en un proyecto con ROS hay que crear un entorno de trabajo que compilará los paquetes, formado por uno o más nodos.

3.5.3. Herramientas

Las herramientas del entorno de desarrollo de software robot permiten la depuración, introspección, trazado y visualización del estado del sistema que se está desarrollando.

Hay accesibles más de 45 herramientas de líneas de comandos, entre las que se encuentran las herramientas gráficas rviz y rqt.

- rviz (Figura 11) es un visualizador tridimensional con diferentes tipos de visualizaciones y complementos que se utiliza para los robots definidos en URDF, los datos de los sensores y los entornos en los que trabajan.
- rqt, basado en Qt, es un entorno de trabajo para desarrollar interfaces gráficas para el robot. Se pueden introducir nuevos componentes de interfaz creando complementos de rqt o utilizar los existentes en la librería.

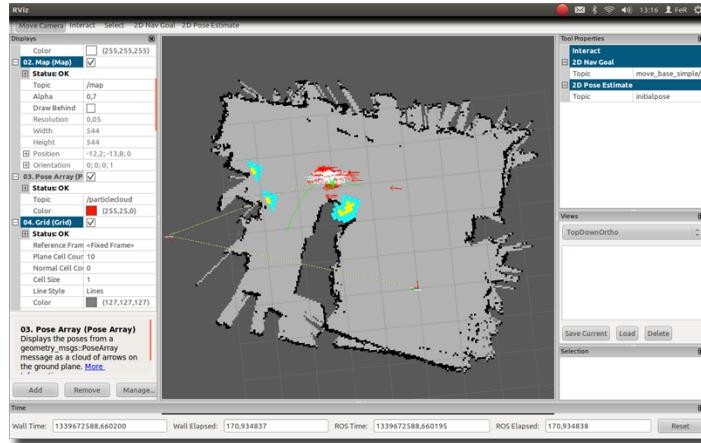


Figura 11: Trayectoria de navegación de un robot por un mapa en RVIZ. Fuente: [7].

3.5.4. Características específicas de los robots

ROS proporciona una serie de herramientas y librerías comunes específicas.

- Existen definiciones de mensajes para diferentes conceptos o instrumentos que forman un conjunto de formatos de mensajes estándar cubriendo la mayoría de usos comunes en robótica.
- La biblioteca `tf` (*transform*), rastrea cada parte del sistema robótico, permitiendo definir transformaciones estáticas y dinámicas, lo que resuelve el problema de conocer la posición de los componentes del robot respecto a otros.
- El lenguaje de descripción son todas aquellas herramientas que se utilizan para describir y modelar el robot, utilizando el formato *Unified Robot Description Format* (URDF), un documento XML donde se describen las propiedades físicas el robot.
- Una forma estándar de producir, recopilar y agregar diagnósticos sobre el robot para que se puedan abordar los problemas a medida que surgen.
- Las acciones pueden informar del progreso antes de devolver la respuesta final, lo que las diferencia de los servicios, además de que pueden ser anuladas por el que llama.
- Paquetes que resuelven los problemas robóticos básicos como la localización en un mapa, la estimación de la posición, la construcción de un mapa e incluso la navegación móvil, permitiendo hacer más, con menos esfuerzo.

3.5.5. Infraestructura de comunicación

ROS ofrece una interfaz de intercambio de mensajes que permite la comunicación entre procesos a nivel más bajo. Esta interfaz proporciona diferentes servicios como:

- El envío de mensajes con los que, mediante el mecanismo de publicación/suscripción anónima, gestiona los detalles de la comunicación entre procesos.
- La grabación y reproducción de mensajes que se puede realizar sin cambios en el código gracias al sistema anónimo y asíncrono.
- Las llamadas de procedimientos remotos permiten las interacciones síncronas de solicitud/respuesta entre procesos usando servicios.
- Los sistemas de parámetros distribuidos proporcionan una forma para que las tareas compartan información de la configuración, además de permitir modificar fácilmente la configuración de las tareas e incluso que unas cambien la configuración de otras.

3.5.6. Comunicación de ROS en JavaScript

Principalmente el lenguaje de programación por excelencia para ROS es C++, pero es posible utilizar otros lenguajes como por ejemplo Python. O como en el caso de este trabajo, HTML y JavaScript para conectarse con ROS a través de rosbridge. Para ello se importa la librería roslibjs, la principal de JavaScript para interactuar con ROS desde el navegador. Utiliza WebSockets para realizar las funciones esenciales de ROS.

4. Implementación

Se ha desarrollado un sistema conversacional utilizando el asistente de voz de Amazon Alexa y realizando la comunicación entre la skill desarrollada y la programación lógica mediante el entorno Jovo. Como conversación de ejemplo en las pruebas, se ha decidido optar por un dialogo de ayudante de cocina que vaya explicando una receta paso a paso, que evolucione según las respuestas del usuario, utilizando algunas variables guardadas como el nombre de la persona con la que conversa. Además en la misma conversación se han preparado ciertas tareas para que el robot realice cuando se le llame, como por ejemplo, moverse y avisar a alguien. La conversación sigue el diagrama de flujo que se muestra en la Figura 12.

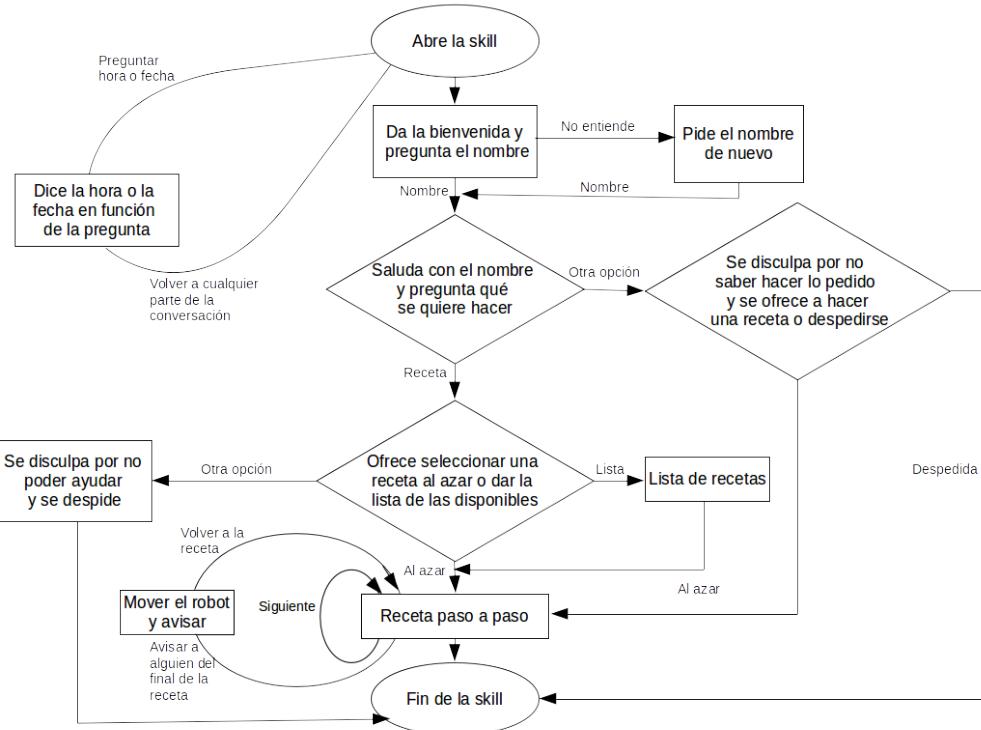


Figura 12: Diagrama de flujo de la conversación.

Como nombre del sistema en general se ha elegido Dorota por su sonoridad y facilidad de pronunciación, en honor a la conocida ama de llaves de la serie americana *Gossip Girl*. Este es el mismo nombre con el que se ha creado el proyecto Jovo para escribir el código de la programación lógica.

4.1. Desarrollo de Dorota

El desarrollo de la skill consta de dos partes: la interfaz de voz de usuario y la programación lógica como explica la Figura 13.



Figura 13: Diagrama de comunicaciones entre la interfaz de usuario (posibles respuestas del usuario) y la programación lógica (respuestas y acciones del robot).

La interfaz de usuario se ha realizado desde la consola de desarrollador de Amazon, utilizando una cuenta de cliente como la de la mayoría de usuarios. Cuenta con versión inglesa y española (Figura 14), ya que durante las primeras fases de desarrollo todavía no estaba disponible el español y se decidió proceder con una aplicación en inglés, hasta que tras finalizar una primera versión, se optó por desarrollar también la versión española que ya sí se encontraba disponible.

Alexa Skills					Create Skill
SKILL NAME	LANGUAGE	TYPE	MODIFIED	STATUS	
Dorota(ES) <small>View Skill ID</small>	Spanish (ES)	Custom	2019-06-09	In Dev	
Dorota <small>View Skill ID</small>	English (US)	Custom	2019-06-07	In Dev	

Figura 14: Creación de la skill Dorota en la consola de desarrollador de Amazon. [1]

4.1.1. Consola de desarrollador

Lo primero que se ha elegido es el nombre de invocación. Teniendo en cuenta el nombre de la skill, se ha decidido que por facilidad y por la función que desempeña, sea *assistant dorota* en la versión inglesa y *asistente dorota* en la española. Es recomendable que tenga dos palabras para evitar en medida de lo posible que exista otra skill con el mismo nombre de invocación.

NAME	UTTERANCES	SLOTS	TYPE	ACTIONS
AnswerIntent	3	1	Custom	Edit Delete
MyNameIntent	5	1	Custom	Edit Delete
MealIntent	1	1	Custom	Edit Delete
ContIntent	1	1	Custom	Edit Delete
ActionIntent	2	1	Custom	Edit Delete
HourIntent	1	1	Custom	Edit Delete
AMAZON.NavigateHomeIntent	-	-	Required	Edit
AMAZON.StopIntent	-	-	Required	Edit
AMAZON.HelpIntent	-	-	Required	Edit
AMAZON.CancelIntent	-	-	Required	Edit

Figura 15: *Intents* de la skill Dorota. Fuente: [1].

Teniendo en cuenta la conversación que se ha planteado en el diagrama de flujo, se necesita un *intent* por cada pregunta que tiene que responder el usuario (Figura 15). Cada uno de los *intents* se han implementado con nombres relacionados con la función que desempeñan, de esta forma los nombres son los siguientes:

- **MyNameIntent:** Es el *intent* que se encarga de recoger el nombre del usuario para utilizarlo, dándole a la conversación un carácter más familiar. La captura del nombre se realiza a través del *Intent slot* names que tiene asociado el tipo de slot AMAZON.US_FIRST_NAME en la versión inglesa. Este slot ya viene integrado en Alexa y es capaz de reconocer cualquier nombre que se diga, facilitando así la tarea de capturar el nombre, mientras que en la versión española se ha tenido que escribir cada uno de los posibles nombres.
- **AnswerIntent:** Este *intent* recoge la respuesta del usuario ante la pregunta de qué se quiere hacer. En él es donde se encuentra el slot que recoge si el usuario quiere realizar una receta o cualquier otra cosa. En caso de elegir una receta se pasará al siguiente intent, en caso contrario el sistema intentará reconducir la conversación diciendo que no sabe realizar esa acción, y sugiriendo realizar una receta de pasta o dejar la conversación despidiéndose.

- **MealIntent:** En él se recoge la decisión de que se presente una receta aleatoria o la lista de recetas disponibles, a través del *intent meal*. Si se dice *let's eat* o *vamos a comer* se devolverá una de las recetas de la batería que hay almacenada, si se elige *list* o *lista*, entonces se devolverá el nombre de todas las recetas de esta batería. En caso de no escoger ninguna de las dos entonces se acabará la conversación.
- **ContIntent:** El *intent* de continuar simplemente se utiliza para poder seguir la receta al ritmo que se considere oportuno, es decir cada vez que el usuario diga *got it* o *continúa*, o cualquiera de los sinónimos que se han añadido en el slot *cont*, Dorota revelará el siguiente paso de la receta, pero mientras tanto se mantendrá a la espera, guardando cuál ha sido la última instrucción.
- **ActionIntent:** Este *intent* está enfocado para el robot de servicios que va a contener este sistema. El slot *action* recoge una serie de acciones como moverse, ir a algún sitio, o decir algo. De esta forma Dorota sólo responderá *Okay, coming right up* o *Vale, ahora mismo*, cuando se le haga alguna petición de este tipo, y la comunicación por ROS actuará para mover al robot a que realice la petición.
- **HourIntent:** Recoge la petición del usuario de conocer la fecha o la hora a través del slot *hour*.

Toda la configuración de los *intents* y los slots (Figura 16) que se ha desarrollado se encuentra en un único archivo en el apartado JSON Editor de la consola de desarrollador, por lo que es bastante simple de reproducir. Simplemente se necesita copiar todo el archivo y pegarlo en una skill nueva, en cuanto se pulse el botón de construir el modelo, al terminar el proceso, aparecerán los elementos tal y como estaban.

Slot Types			
	SLOT VALUES	TYPE	ACTIONS
action	3	Custom	Edit Delete
AMAZON.US_FIRST_NAME	-	Built-In	Edit Delete
answer	5	Custom	Edit Delete
cont	1	Custom	Edit Delete
hour	2	Custom	Edit Delete
meal	6	Custom	Edit Delete
name	9	Custom	Edit Delete

Figura 16: Slots asignados a las *intents*. Fuente: [1].

4.1.2. Programación lógica

El código que se comunica con la interfaz de usuario se ha elegido desarrollar a través de un servicio web y hospedarlo en el computador, por la facilidad que supone a la hora de escribirlo y porque se trata de un servicio gratuito. AWS Lambda de Amazon también es gratuito pero tiene ciertas restricciones: es válido sólo durante los primeros 12 meses siempre que no se supere cierta cantidad de tráfico de datos, además el lenguaje de programación pese a ser el mismo, utiliza diferentes comandos que son menos intuitivos y de fácil equivocación. Por todas estas razones se ha elegido el servicio Jovo, que permite una programación intuitiva y gratuita desde cualquier computador. Su instalación es muy simple y rápida, de hecho en este trabajo se ha desarrollado el primer Anexo como guía que cubre los pasos a seguir.

Se ha creado un nuevo proyecto Jovo con el mismo nombre que se ha elegido en la skill, para mantener así la concordancia y que sea fácil de identificar (Figura 17). La creación de este proyecto genera una carpeta con el nombre que se ha escogido en cuyo interior se encuentran diferentes archivos y carpetas necesarios para realizar la comunicación, pero la carpeta más importante es la llamada *src* donde se encuentra un archivo JavaScript llamado *app*, el archivo donde se realiza la programación lógica de la skill. La creación de este archivo incluye la configuración de la skill y dos ejemplos de *intents* predeterminados que son: el *intent* de lanzamiento (lo que devuelve el sistema al lanzar la skill) y el *intent* de nombre, que incluye la utilización de un slot como variable para incluirlo en la conversación.



```
david@dorotha:~$ jovo new Dorota
I'm setting everything up
✓ Creating new directory /Dorota
✓ Downloading and extracting template helloworld
✓ Installing npm dependencies

Installation completed.
```

Figura 17: Creación del proyecto Dorota con el entorno Jovo.

Para el desarrollo del código, por cada *intent* que se ha incluido en la consola de desarrollador se ha asociado una función con el nombre del *intent*, y como parámetro, el slot que se ha utilizado como variable para tomar las decisiones que sigue el diagrama de flujo de la Figura 12.

Antes de codificar cada *intent* se ha creado una batería de recetas utilizando el tipo de variables que tiene la skill de trivial. Se crea una variable global en la que se añaden las

recetas identificando nombre, ingredientes y seis pasos distintos, cada receta debe estar entre llaves y separada de la siguiente por comas. Además se crea una variable que guarda el número de recetas que hay y otra que genera un número aleatorio entre 1 y el número de recetas, que se utilizaran en los *intents* de *Meal* y *Cont* respectivamente.

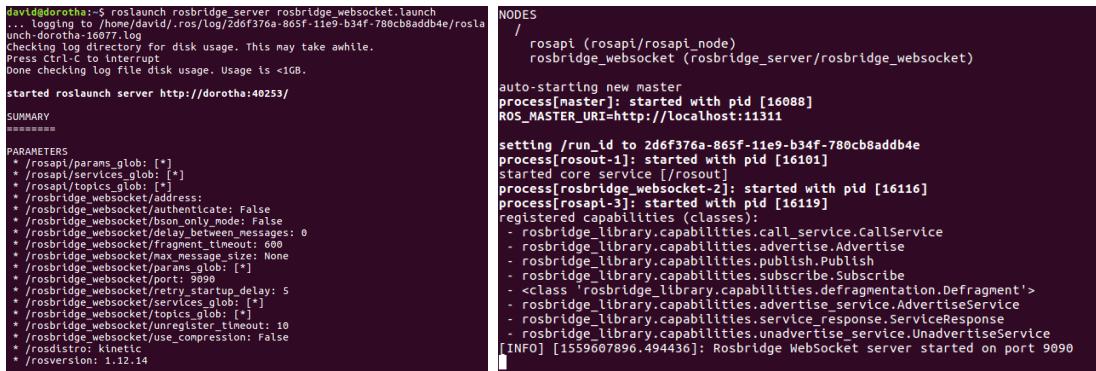
Las funciones de los *intents* se han implementado en el código de la siguiente manera:

- **HelloWorldIntent:** El *intent* de lanzamiento de la skill simplemente devuelve una frase de bienvenida y pregunta por el nombre del usuario.
- **MyNameIntent:** Obtiene el nombre de la persona a través del slot de la consola de desarrollador y lo utiliza para saludar al usuario y preguntar qué quiere hacer.
- **AnswerIntent:** Es el primero que se codifica en función a la respuesta escogida de la pregunta anterior. Mediante una comparación se escoge entre hacer una receta, en cuyo caso se le propondrá al usuario saber los nombres de las disponibles o dejar que el sistema elija una al azar. O no hacer una, lo que supondrá decirle al usuario que no sabe hacer eso y que puede ofrecer realizar una receta o dejar la conversación despidiéndose.
- **MealIntent:** Sigue la misma estructura que el anterior, porque también hay que escoger entre diferentes opciones, pero en este caso también se asignan variables con nombre sencillo, al nombre de la receta y a los ingredientes. Según la comparación si se elige que el sistema muestre una receta aleatoria, se empieza diciendo cuáles son su nombre e ingredientes, por eso se guardaban en una variable al principio. Mientras que si se decide por saber cuáles son las recetas de la base de datos, simplemente se devuelve el nombre de todas las recetas disponibles. En caso de no pedir ninguna de estas dos opciones, el sistema devolverá una disculpa por no poder ayudar y se despedirá.
- **ContIntent:** Para continuar con la receta que ha empezado el sistema en el *intent* anterior, se asigna una variable a cada paso de la misma, y se realiza una cuenta por cada vez que se diga que se puede continuar, para poder decir el siguiente paso. Este mecanismo también permite que si en cualquier momento se decide realizar otra acción, cuando se retomen los pasos de la receta, continúe por donde se ha quedado. Al finalizar todos los pasos, el sistema avisa que la receta se ha terminado y se despide del usuario.

- **ActionIntent:** Permite lanzar una acción para el robot de servicios donde se ha integrado esta skill, devolviendo al usuario un *Okay, coming right up* o *Vale, ahora mismo*.
- **HourIntent:** Se realiza la captura de la fecha completa y la hora mediante la función *Date* propia de JavaScript, se separa la fecha completa de la hora y se devuelven al usuario en función de lo que pregunte.

4.2. Comunicación con ROS

La comunicación de la skill con el robot de servicios en el que se ha implantado se ha realizado a través de ROS. Para ello como se ha mencionado en el apartado de Desarrollo y herramientas en el robot se necesita, además de tener instalado el sistema operativo robótico, la librería propia de JavaScript *roslibjs*, para poder realizar la llamada al servicio que hará que el robot móvil se comunique y se mueva. Pero además para que esta conexión se lleve a cabo se necesita instalar el paquete rosbridge (Figura 18), para poder interactuar con la web. La instalación se puede llevar a cabo siguiendo la guía desarrollada en el Anexo 2.



```
davidd@dorothea:~$ rosrun rosbridge_server rosbridge_websocket.launch
... logging to /home/david/.ros/log/2d6f376a-865f-11e9-b34f-780cb8addb4e/rosla
nd
[rosapi] Starting log file at /home/david/.ros/log/2d6f376a-865f-11e9-b34f-780cb8addb4e/rosla
nd.log
[rosapi] Checking log directory for disk usage. This may take awhile.
[rosapi] Press Ctrl-C to interrupt
[rosapi] Done checking log file disk usage. Usage is <1GB.

started rosbridge server http://dorothea:40253

SUMMARY
=======
PARAMETERS
+ /rosapi/params.glob: []
+ /rosapi/services.glob: [*]
+ /rosapi/topics.glob: [*]
+ /rosbridge_websocket/address:
+ /rosbridge_websocket/authenticate: False
+ /rosbridge_websocket/bson_only_mode: False
+ /rosbridge_websocket/call_timeout: 0
+ /rosbridge_websocket/fragment_timeout: 600
+ /rosbridge_websocket/max_message_size: None
+ /rosbridge_websocket/params.glob: [*]
+ /rosbridge_websocket/port: 9090
+ /rosbridge_websocket/retry_startup_delay: 5
+ /rosbridge_websocket/topic_glob: [*]
+ /rosbridge_websocket/unregister_timeout: 10
+ /rosbridge_websocket/use_compression: False
+ /rosdistro: kinetic
+ /rosversion: 1.12.14

NODES
/
  rosapi (rosapi/rosapi_node)
  rosbridge_websocket (rosbridge_server/rosbridge_websocket)

auto-starting new master
process[master]: started with pid [16088]
ROS_MASTER_URI=http://localhost:11311

setting /run.id to 2d6f376a-865f-11e9-b34f-780cb8addb4e
process[rosout-1]: started with pid [16101]
started core service [/rosout]
process[rosbridge_websocket-2]: started with pid [16116]
process[rosapi-3]: started with pid [16119]
registered capabilities (classes):
- rosbridge_library.capabilities.call_service.CallService
- rosbridge_library.capabilities.advertise.Advertise
- rosbridge_library.capabilities.publish.Publish
- rosbridge_library.capabilities.subscribe.Subscribe
- <class 'rosbridge_library.capabilities.defragmentation.Defragment'>
- rosbridge_library.capabilities.advertise_service.AdvertiseService
- rosbridge_library.capabilities.service_response.ServiceResponse
- rosbridge_library.capabilities.unadvertise_service.UnadvertiseService
[INFO] [1559607896.494436]: Rosbridge WebSocket server started on port 9090
```

Figura 18: Lanzamiento del websocket con rosbridge.

En el código, en el apartado de la configuración de la aplicación, se crea un objeto de nodo ROS para comunicarse con el servidor rosbridge, se agregan oyentes para los eventos de conexión, error y cierre, monitorizando así la conexión al servidor rosbridge. Por último se declara la llamada al servicio indicando el nombre del mismo (*bt_manager/add_new_task*) y el tipo (*bt_manager/AddTask*).

Ya sólo es necesario declarar una variable de petición del servicio después de cada interacción de respuesta del sistema con la misma información que se manda a la consola

de desarrollador como argumentos, especificando la tarea a desempeñar, ya sea hablar (*say*) o moverse (*go_to_point*) y realizar la llamada. De esta forma cada vez que se acceda a un *intent*, cuando el sistema devuelva la respuesta, también se la comunicará al robot.

El resultado final es que si se sigue el Anexo 3 para poner en marcha la skill, se consigue que se comunique la información gracias al servicio, haciendo que el robot hable para mantener una conversación con el usuario y se mueva cuando se le pida que vaya a avisar de que la comida está lista.

Para que el proyecto lleve a cabo la comunicación ROS correctamente es necesario lanzar el websocket antes de ejecutarlo, en caso contrario se produce el error que se muestra en la Figura 19, dando lugar a que la skill funcione en la consola de desarrollador pero no tenga en cuenta la parte de ROS.

```
david@dorotha:~$ cd Dorota
david@dorotha:~/Dorota$ jovo run
ROSLib uses utf8 encoding by default. It would be more efficient to use ascii (if
possible)
Example server listening on port 3000!
Error connecting to websocket server: { Error: connect ECONNREFUSED 127.0.0.1:9
090
    at TCPConnectWrap.afterConnect [as oncomplete] (net.js:1191:14)
```

Figura 19: Error en la comunicación con el rosbridge.

5. Integración y pruebas con el robot

En este capítulo se procede a la explicación de las pruebas realizadas e integración en un robot de servicios.

5.1. Instalación de herramientas

Para la comprobación del correcto funcionamiento de todo el sistema, éste se integra en un robot de servicios Giraff del grupo de investigación de la Universidad de Málaga MAPIR, provisto de arquitectura basada en ROS para poder entablar conversación y navegar evitando obstáculos. Para ello se han tenido que instalar las herramientas que se han utilizado durante el desarrollo, siguiendo el Anexo 1 paso a paso para la instalación del entorno Jovo, instalando primero el sistema de gestión de paquete npm y Node.js.

Al utilizar sólo el comando de instalación de Node.js, cuando se procede a instalar Jovo aparece un problema y no se completa la instalación porque la versión de Node.js es inferior a 8.10 que es la mínima requerida para la instalación de Jovo v2. Por ello se ha tenido que limpiar la caché, volver a instalar Node.js y actualizar la versión a la más reciente posible vinculando el último archivo instalado al *path* de las versiones de node de la carpeta de usuario local.

Una vez instalado Jovo se han creado dos nuevos proyectos Dorota y Dorota_ES, en los que se ha sustituido el archivo app.js de la carpeta src por las versiones inglesa y española respectivamente del sistema desarrollado.

5.2. Pruebas

Una vez están las versiones en el ordenador del Giraff, siguiendo el Anexo 3 se han ejecutado para proceder a las pruebas de reconocimiento de voz y órdenes. La realización de dichas pruebas se ha llevado a cabo en el laboratorio del grupo MAPIR durante un día de trabajo normal, por lo que el laboratorio estaba repleto de personas trabajando, comentando opiniones entre sí, tecleando, generando el ruido ambiente normal de un entorno de trabajo que dificultó el reconocimiento de sonido por parte del micrófono.

El robot cuenta con dos micrófonos como se puede apreciar en la Figura 20, los dos situados en las cámaras kinect que tiene integradas, uno en la superior y otro en la que se encuentra debajo de la pantalla. Se han realizado pruebas de sonido con ambos para comprobar cuál tenía mejor precisión y en un primer momento se había decidido utilizar el de la cámara de la parte superior porque desde los ajustes de sonido se podía apreciar

que reconocía el sonido mejor, sin embargo a la hora de empezar las pruebas con el sistema conversacional, el asistente no fue capaz de reconocer la voz del usuario por lo que se ha tenido que optar por otra solución.



Figura 20: Fotograma del vídeo donde se aprecian las cámaras del robot.

El sistema está desarrollado de forma que cualquier dispositivo con acceso a internet y pantalla puede actuar sobre él, desde un *smartphone* hasta un ordenador, por lo que se ha decidido que para las pruebas con el robot se va a utilizar un ordenador portátil que va a actuar como micrófono para reconocer la voz del usuario.

Ya que el robot está siendo utilizado por otras personas, no era posible realizar cambios en los ajustes o incluir más dispositivos en el propio robot que podían dificultar o entorpecer su trabajo. El hecho de tener que usar un dispositivo externo al robot y que no se encuentra a la altura idónea para que entienda fácilmente al usuario, ha generado que a veces el asistente no reconozca con precisión la respuesta u orden que se le ha transmitido. Debido a que el micrófono del ordenador portátil además de captar la voz del usuario, capta el sonido ambiente de la sala en la que se encuentra, dificulta el reconocimiento de ciertas palabras, que en cierta medida se puede solucionar elevando un poco más el volumen de la voz al hablar.

Las pruebas han consistido en mantener una conversación con el robot de servicios para comprobar el funcionamiento del sistema desarrollado, finalizando con la ejecución de una orden de movimiento que consiste en la petición de moverse a un punto específico desde la posición en la que se encuentra el robot. Tanto para realizar esta orden de movimiento como para que el propio robot hable, se necesita realizar una llamada al servicio `bt_manager/add_new_task`. Este servicio se encarga de asignar al robot diferentes tareas para que las ejecute, que pueden ir desde hablar a moverse o garantizar que la batería del robot esté siempre en buen estado. Cuando se realiza la llamada al servicio,

necesita recibir los siguientes parámetros:

- *name*: tipo de tarea.
- *priority*: prioridad del 0(baja) al 9(alta).
- *permanence*: *False* para que la tarea se detenga después de una ejecución y *True* para que no se detenga nunca.
- *args*: Parámetros que tengan cada tarea (Ej.: texto para hablar o coordenadas a las que moverse).

Tras diferentes tomas y ajustes del volumen de voz del usuario, las pruebas se han llevado a cabo con éxito y el robot Giraff ha seguido la conversación como se esperaba, acatando la orden de movimiento como muestran las Figuras 21 y 22.



(a) El usuario le indica la orden y el robot comienza la navegación.
 (b) El robot acercándose al punto designado.

Figura 21: Fotograma del vídeo donde el robot se mueve por petición del usuario.

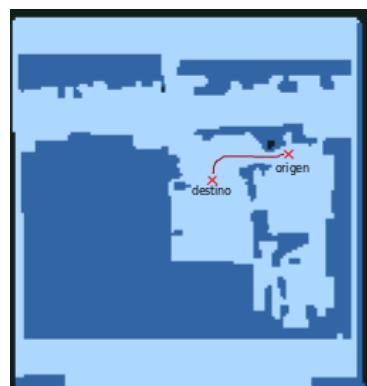


Figura 22: Captura de pantalla de rviz con la trayectoria que sigue el robot.

6. Conclusiones y mejoras futuras

Los asistentes virtuales y en concreto Alexa, que ha sido objeto de estudio en este Trabajo Fin de Grado, seguirán desarrollándose y creciendo durante los próximos años volviéndose cada vez más “inteligentes” y llegando así a más hogares, incluso llegando a estar incluidos en construcciones futuras. Aunque para que sean realmente útiles en el hogar es necesario el desarrollo de más dispositivos inteligentes y que sean más accesibles para el usuario medio. El avance de la inteligencia artificial conseguirá que las conversaciones que se puedan tener sean cada vez más fluidas, llegando incluso a adquirir tal capacidad de conocimiento que sea posible conversar con un asistente virtual como si se estuviese manteniendo una conversación con una persona real. O que en función a ciertas situaciones que sean capaces de identificar a través de sensores, actúen tomando decisiones en función al conocimiento adquirido.

El objetivo del trabajo era desarrollar un sistema que pudiese integrarse en un robot de servicios, y se ha llevado a cabo con éxito, sin embargo las funcionalidades de la skill pueden ampliarse cuanto se quiera. Además de la funcionalidad de recetas, trabajos futuros a realizar pueden ser la inclusión de más idiomas o más tipos de conversaciones, como por ejemplo sobre el tiempo o el fútbol. Aumentar las posibles respuestas del usuario (*utterances*), aumentando así la inteligencia del mismo y así conocer la respuesta a cualquier petición del usuario. Incluso podría añadirse cualquier mejora que tenga que ver con el reconocimiento de imágenes, como un sistema que sea capaz de reconocer objetos de un habitación, distribuido por todas las habitaciones de una casa y así poder preguntar a Alexa donde están las llaves del coche o el teléfono móvil por ejemplo.

Bibliografía

- [1] Alexa. <https://developer.amazon.com/es>. 08/03/2018.
- [2] Alexa repositories. <https://github.com/alexa/>. 08/03/2018.
- [3] Amazon echo dot. <https://www.amazon.es/dp/B07PHPXHQ5>. 21/07/2019.
- [4] Custom skills. <https://developer.amazon.com/es/docs/custom-skills/>. 08/03/2018.
- [5] Google home mini. https://store.google.com/es/product/google_home_mini. 21/07/2019.
- [6] Jovo. <https://www.jovo.tech/>. 31/07/2018.
- [7] Mapa y trayectoria de navegación con rviz. <http://robotica.unileon.es/index.php/Fernando-TFM-ROS02>. 21/07/2019.
- [8] Policy testing for an alexa skill. <https://developer.amazon.com/docs/custom-skills/policy-testing-for-an-alexa-skill.html#cert-tm-ip-brands>. 30/05/2019.
- [9] Ros libjs. <http://wiki.ros.org/roslibjs>. 06/03/2019.
- [10] Ros packages. packages.ros.org. 05/03/2019.
- [11] Ros wiki. <http://wiki.ros.org/>. 22/04/2019.
- [12] Simulador echo. <https://echosim.io/welcome?next=%2F/>. 08/03/2018.
- [13] HOY, M. B. Alexa, siri, cortana, and more: An introduction to voice assistants. *Medical reference services quarterly* 37, 1 (2018), 81–88.
- [14] JOSEPH, L. *ROS Robotics Projects*. Packt Publishing, 2017.
- [15] KEPUSKA, V., AND BOHOUTA, G. Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home). In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)* (2018), IEEE, pp. 99–103.

- [16] LÓPEZ, G., QUESADA, L., AND GUERRERO, L. A. Alexa vs. siri vs. cortana vs. google assistant: a comparison of speech-based natural user interfaces. In *International Conference on Applied Human Factors and Ergonomics* (2017), Springer, pp. 241–250.
 - [17] TEIXEIRA, P. *Professional Node.js: Building Javascript based scalable software*. John Wiley Sons, 2012.
- [1–17]

Anexo 1

Instalación de Jovo

Tutorial para la instalación y creación de un proyecto donde escribir el código de la skill y ejecutarlo desde un servidor remoto con el Jovo Webhook.

1.1. Especificaciones técnicas

Antes de proceder con la instalación del entorno JOVO hay que tener en cuenta que este necesita para su correcto funcionamiento *Node.js*, en una versión igual o superior a 8.10, y su sistema de gestión de paquetes por defecto *npm*. Para su instalación hay que seguir en orden los siguientes comandos:

1. Comprobar que los paquetes actualmente instalados en la computadora estén actualizados.

```
$ sudo apt-get update
```

2. Tanto la instalación de jovo-cli como la de node se realiza a través de el sistema de gestión de paquetes, por lo que hay que instalarlo antes.

```
$ sudo apt-get install npm
```

3. Si no se había instalado antes no debería dar problemas, pero es recomendable realizar una limpieza del caché de npm forzado para evitar que se instalen versiones antiguas.

```
$ sudo npm cache clean -f
```

4. Una vez realizada la limpieza de caché se instalan los módulos n usando el comando de gestión de paquetes.

```
$ sudo npm install -g n
```

5. Es necesario comprobar la versión de node instalada para cerciorarse de que sea superior a la 8.10.

```
$ node -v
```

6. En caso de ser inferior, se puede actualizar la versión usando el modulo n. En el que aparece la versión a la que se actualiza

```
$ sudo n stable
```

7. Vincular el último archivo nodejs instalado, escribiendo la versión que aparece al utilizar el comando anterior.

```
$ sudo ln -sf /usr/local/n/versions/node/[  
version_que_haya_aparecido]/bin/node /usr/bin/node
```

1.2. Jovo

1. Instalación de Jovo-CLI.

Las herramientas de líneas de comandos de Jovo ofrecen un muy buen punto de partida para la creación de aplicaciones de voz.

```
$ npm install -g jovo-cli
```

Con esto deberá estar descargado e instalado, pero se puede comprobar si la instalación ha terminado correctamente y funciona sin fallos con el siguiente comando, que devuelve la versión actual del CLI:

```
$ jovo -v
```

2. Crear un nuevo proyecto.

Como se puede ver con el comando de abajo es posible crear un nuevo proyecto:

```
$ jovo new [nombre_del_proyecto]
```

El modelo *helloworld* (Figura 23) es la plantilla por defecto que clonará la aplicación ejemplo Jovo en el directorio especificado:

```
david@dorotha:~$ jovo new nombre_del_proyecto
I'm setting everything up
✓ Creating new directory /nombre_del_proyecto
✓ Downloading and extracting template helloworld
✓ Installing npm dependencies

Installation completed.
```

Figura 23: Creación de un nuevo proyecto

3. Ejecución desde un servidor remoto con el Jovo Webhook.

Una vez modificado el fichero donde se aloja la skill a desarrollar, para comprobar si funciona el código hay que dirigirse desde consola al fichero del proyecto y ejecutar el comando:

```
$ jovo run
```

Lo que iniciará el servidor express que devuelve la url necesaria en la Consola de desarrollador de Amazon (Figura 24).

```
david@dorotha:~$ cd nombre_del_proyecto
david@dorotha:~/nombre_del_proyecto$ jovo run
Example server listening on port 3000!
This is your webhook url: https://webhook.jovo.cloud/e0ca859f-208e-44fc-89c4-e8cecfcd18c9
```

Figura 24: Lanzamiento del servidor local

La skill ya está ejecutándose localmente y lista para probarla.

Anexo 2

Instalación de ROS y roslibjs

Tutorial para la instalación de ROS Kinetic Kame y la librería roslibjs necesaria para comunicar la skill a través de un nodo ROS.

2.1. ROS

La instalación de ROS [11] se puede realizar en diferentes sistemas operativos (Ubuntu y Debian), en este caso se ha optado por el sistema de Linux Ubuntu. Lo primero que hay que hacer es habilitar los repositorios de Ubuntu “restricted”, “universe” y “multiverse”.

1. Configurar el ordenador para que acepte software de packages.ros.org [10]:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/roslatest.list'
```

2. Configurar las claves.

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

Si aparecen problemas al conectar con el servidor de claves, se puede intentar sustituir hkp://pgp.mit.edu:80 o hkp://keyserver.ubuntu.com:80 en el comando anterior.

3. Instalación.

Primero hay que asegurar que el paquete Debian está actualizado:

```
$ sudo apt-get update
```

Hay diferentes librerías y herramientas en ROS. Existen cuatro configuraciones por defecto con las que empezar, aunque también se pueden instalar paquetes de ROS individualmente. Por lo tanto hay que elegir el tipo de instalación entre los siguientes

- ROS desktop instalación completa: ROS, rqt, rviz, librerías robot-generic, simuladores 2D/3D, navegación y percepción 2D/3D.

```
$ sudo apt-get install ros-kinetic-desktop-full
```

- ROS desktop: ROS, rqt, rviz, librerías robot-generic.

```
$ sudo apt-get install ros-kinetic-desktop
```

- ROS-Base desktop instalación completa: paquete ROS, construcción y librerías de comunicación. Sin herramientas GUI.

```
$ sudo apt-get install ros-kinetic-ros-base
```

- Instalar paquetes específicos.

```
$ sudo apt-get install ros-kinetic-PACKAGE
```

Se puede utilizar el siguiente comando para encontrar los paquetes disponibles:

```
$ apt-cache search ros-kinetic
```

4. Inicialización de rosdep.

Antes de poder iniciar ROS, hay que inicializar rosdep. Permite instalar fácilmente las dependencias del sistema para la fuente que se desea compilar y se requiere para ejecutar algunos componentes centrales de ROS:

```
$ sudo rosdep init  
$ rosdep update
```

5. Configuración del entorno:

Es conveniente comprobar si las variables del entorno de ROS se han añadido automáticamente al archivo bashrc y si no añadirlas manualmente.

```
$ gedit ~/.bashrc  
Incluir al final: source /opt/ros/kinetic/setup.bash
```

2.2. Roslibjs

Ya que el código desde el que trabaja la skill está programado en node.js, se necesita instalar la librería ROS de Javascript [9] para poder subscribirse y publicar en un topic al que se conecta el robot para la interactuación con el usuario. Para ello es necesario ejecutar el siguiente comando:

```
$ sudo npm i roslib
```

Y para que la estructura escrita en ROS en el código se pueda ejecutar hay que tener instalado un rosbridge:

```
$ sudo apt-get install ros-kinetic-rosbridge-suite
```


Anexo 3

Ejecución de la skill

1. Primero hay que lanzar el main que permite que el servicio al que se llama funcione:

```
$ roslaunch missions_pkg movecare_main.launch
```

2. Para elegir el punto al que se va a mover el robot hay que abrir el remote, donde se pueden ver las coordenadas que hay que introducir:

```
$ roslaunch missions_pkg movecare_remote.launch
```

3. Se lanza el puente entre ros y la web, para que se pueda ejecutar la estructura de ROS integrada en el código.

```
$ roslaunch rosbridge_server rosbridge_websocket.launch
```

4. Por último en otra pestaña, se abre la carpeta en la que se encuentra el proyecto y se ejecuta:

```
$ cd nombre_del_proyecto  
$ jovo run
```

Ahora sólo hay que abrir el simulador o la consola de desarrollador para poder conversar.