industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Integración en la Nube de un robot móvil con capacidad de interacción natural con los usuarios.

TRABAJO FIN DE MÁSTER

MÁSTER EN INDUSTRIA 4.0

Autor: Roberto Oterino Bono
Directora: Dra. Nieves Pavón Pulido
Codirector: Dr. Jorge Juan Feliú Batlle

Cartagena, a 10 de septiembre de 2022

Universidad Politécnica de Cartagena

Agradecimientos

A mi familia que siempre me ha apoyado por mucho errores que haya cometido. Agradecerle también a la Doctora Nieves Pavón y al Doctor Jorge Juan Feliú por ofrecerme de nuevo la oportunidad de trabajar con ellos.

RESUMEN

El diseño de robots con los que se pueda interactuar de forma sencilla es un requisito indispensable para la Robótica de Servicio y más aún para la Robótica Social. A pesar de que la tecnología ha mejorada y la barrera económica para adquirir un robot con esta característica ha disminuido, todavía no se trata de un producto totalmente maduro.

Este trabajo aborda el diseño y despliegue de un sistema robusto que permita la interacción y la comunicación con un robot móvil autónomo de forma natural y que se pueda configurar remotamente. Para ello, se ha partido del desarrollo previo [1], mejorando sus limitaciones y añadiendo nuevos funcionalidades.

Se ha desarrollado un nuevo sistema de reconocimiento facial partiendo de la inferencia que proporcionaba la detección facial del sistema anterior. Se han implementado varios algoritmos de aprendizaje supervisado y un modelo de aprendizaje profundo para realizar clasificación binaria. De este modo, se ha pasado de una identificación realizada de forma empírica a una objetiva.

En cuanto al reconocimiento de voz se han presentado dos alternativas. Una de ellas proporciona mejores resultados y es más ligera. El modelo anterior presentaba un *Word Error Rate* del 42% y el nuevo del 19%. Además, se ha desarrollado un nuevo módulo de procesamiento de lenguaje natural (Natural Language Processing o NLP), que es fácilmente escalable, ya que utiliza métodos para reconocer patrones en las frases.

Por otro lado, se ha desarrollado una aplicación usando algunas herramientas de Google Cloud que permite almacenar y desplegar configuraciones remotas en el robot, que luego podrá utilizar de manera local. Para simplificar su uso se ha diseñado una página web.

Por último, se han integrado todos los componentes en una plataforma robótica real y se ha configurado el sistema de navegación autónoma de manera que utilice la información que proporcionan los sensores propioceptivos y los sensores exteroceptivos del robot.

Todo el sistema se ha validado adecuadamente mediante un conjunto de pruebas que permiten comprobar cada módulo de forma independiente y la integración de todos ellos.

Índice

Figuras

Tablas

Fragmentos de código

1 Introducción.

En capítulo se describe el contexto en el que se engloba el trabajo propuesto, esto es, el de la Robótica de Servicio. También se explica cómo se pueden abordar los problemas relacionados con la comunicación con el robot y los costes de fabricación. Además, se detallan los objetivos a alcanzar para diseñar y desplegar un sistema robusto que permita comunicarse con un robot móvil autónomo de forma natural y configurarlo remotamente. Por último en el apartado 1.2 se ha escrito un pequeño resumen de cada capítulo de forma independiente.

1.1 Motivación y objetivos.

La Robótica de Servicios está sufriendo un desarrollo tecnológico, impulsado por las innovaciones en Aprendizaje Automático, Inteligencia Artificial o Visión Artificial [2], entre otros Los segmentos de la Robótica de Servicios que más están creciendo son: logístico, médico y Robótica de Campo.

A día de hoy se puede minimizar los costes asociados al diseño de un robot funcional alquilando las capacidades de computación que ofrecen los servicios de la Nube. (*Cloud Services*). Además van apareciendo dispositivos embebidos donde se pueden desplegar modelos de Inteligencia Artificial ligeros que permiten cubrir ciertas funcionalidades como el reconocimiento facial y el procesamiento de lenguaje natural.

Precisamente gracias a esta última funcionalidad se puede cubrir el problema de comunicación con un robot ya que la manera más utilizada entre robot y humanos es la comunicación no verbal (a través de interfaces propias del ámbito de la Informática) [3].

Con este proyecto se quiere romper tanto la barrera económica como el problema de la comunicación. No todo el mundo sabe programar un robot, y la idea no es que todo el mundo sepa. Cada vez se tiene que facilitar más la comunicación entre una persona y un robot.

El objetivo principal será diseñar y desplegar un sistema robusto que permita comunicarse con el robot de forma natural, dotarlo de un sistema de navegación autónoma y desarrollar una aplicación para descargar configuraciones en el robot de manera remota. Este objetivo se logra alcanzando los siguientes subobjetivos:

- Aumentar la robustez del sistema de interacción humano robot.
- Desarrollar una aplicación en la Nube para almacenar y descargar diferentes configuraciones del robot de manera remota. Para facilitar su uso, se diseñará una página web.
- Configurar el paquete de navegación de ROS para que funcione con el controlador del robot.
- Desplegar todo el sistema en el robot.

1.2 Resumen de capítulos.

- **Capítulo 1: Introducción.**

  En este capítulo se ha introducido el tema de la Robótica de Servicios y se han analizado los problemas existentes. Así mismo, se han descrito los objetivos que se quieren alcanzar para diseñar y desplegar un sistema robusto que permita comunicarse con un robot móvil autónomo de forma natural y configurarlo remotamente.

- **Capítulo 2: Estado del arte.**

  En este apartado se detallan los sistemas robóticos que utilizan la Nube y se explican los paradigmas de computación existentes. También se han presentado cuatro robots que tienen un propósito social y utilizan alguno de esos paradigmas.

- **Capítulo 3: Diseño del sistema.**

  En esta sección, se explica la estructura del sistema. Por un lado, la arquitectura hardware, es decir, los componentes del robot y su funcionalidad. Por otro, la arquitectura software, diferenciando los módulos que se desplegarán *on premise* y los módulos que se desplegarán en la Nube de Google. Por último, se explica cómo se han integrado todos los componentes.

- **Capítulo 4: Análisis de los resultados.**

  En este apartado se describen las pruebas realizadas y los resultados obtenidos. Además, se realiza un análisis de los beneficios y limitaciones del sistema.

- **Capítulo 5: Conclusiones y trabajos futuros.**

  En este capítulo se presentan las conclusiones obtenidas y se describen las posibles líneas de desarrollo que se pueden realizar a partir de este trabajo.

2 Estado del arte.

En este apartado se van a describir los modelos de computación que se muestran en la Fig. 1 para entender qué beneficios pueden aportar cada uno de ellos en los sistemas robóticos. Además, se nombrarán las tres plataformas en la Nube líderes del mercado, de las cuales una se describirá en el apartado 3.4 porque será la empleada y tres dispositivos embebidos con gran capacidad de cálculo

Por último, se mencionarán cuatro robots cuyo propósito es social y se apoyan en dichos paradigmas de computación dependiendo de la funcionalidad que realicen.

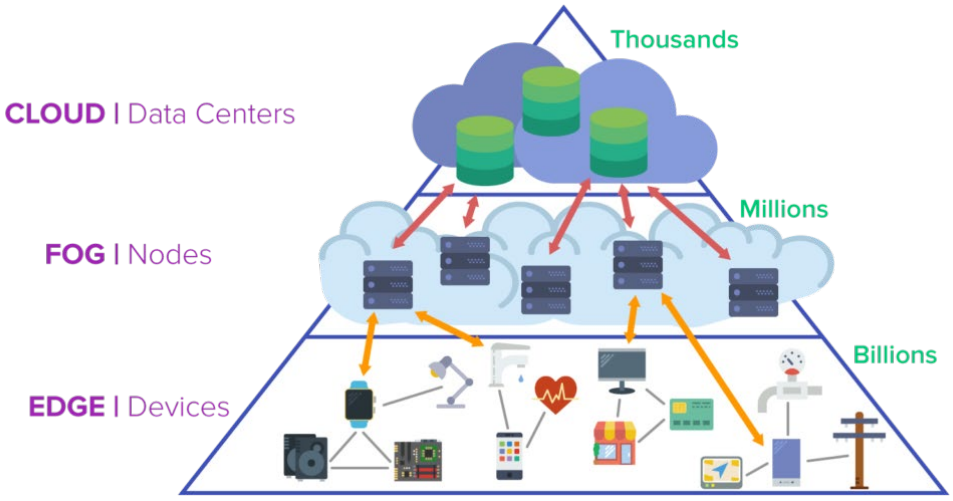

Fig. 1 Modelos de computación.

2.1 Cloud Computing.

El NIST (*National Institute of Standars and Technology),* define Computación en la Nube (*Cloud Computing)* como: "Un modelo tecnológico que permite el acceso ubicuo, adaptado y bajo demanda en red a un conjunto compartido de recursos de computación configurables compartidos (por ejemplo: redes, servidores, equipos de almacenamiento, aplicaciones y servicios), que pueden ser rápidamente aprovisionados y liberados con un esfuerzo de gestión reducido o interacción mínima con el proveedor del servicio." [4] Las características que ofrece son:

- **Autoservicio bajo demanda**: permite al usuario acceder a los servicios contratados de manera automática a medida que los va requiriendo.

- **Pago por uso:** tan solo se factura el uso de los servicios utilizados. De este modo, el cliente paga por los servicios en activo y tiene la posibilidad de ampliar la capacidad de los mismos a medida que sus necesidades incrementan.

Agilidad en la escalabilidad: el aumento de la capacidad de los servicios es transparente al usuario por lo que el cliente no debe preocuparse de la escalabilidad del sistema.

Las plataformas en la Nube líderes del mercado son Amazon Web Service, Microsoft Azure y Google Cloud tal y como muestra la Fig. 2.

Fig. 2 Cuadrante mágico de Gartner Cloud Computing.

2.2 Fog Computing.

El concepto de computación en la niebla (*Fog Computing)*, se define como: "Una arquitectura de red descentralizada en la que tanto los recursos, como los datos y aplicaciones, se alojan en un lugar lógico (nodo o Gateway), entre la Nube y el dispositivo raíz que genera la información." [5] Con esta infraestructura se consigue que el cálculo computacional se aproxime a los extremos de la red.

De este modo el procesamiento tiene lugar en un nodo reduciendo así la cantidad de datos que recorren la red, por lo que se reducen los niveles de latencia y el tiempo de respuesta de la aplicación.

Este paradigma de computación se enfoca a servicios que consumen un elevado ancho de banda por lo que se benefician de los niveles bajos de latencia que proporciona esta arquitectura, permitiendo así un procesado de la información en tiempo real.

2.3 Edge Computing.

La computación en el borde *(Edge Computing)* es un paradigma de computación donde los recursos, datos y aplicaciones se procesan directamente sobre los dispositivos. Así se evitan las restricciones de ancho de banda, se reducen las demoras en la transmisión, se limitan los fallos del servicio y se mantiene la privacidad de los datos [6].

Existen diferentes dispositivos embebidos con una gran capacidad de cálculo computacional. Entre ellos se encuentran:

- **Intel Neural Compute Stick 2** (ver Fig. 3), Permite el desarrollo y prototipado de aplicaciones de inteligencia artificial (IA) en una amplia gama de dispositivos que dispongan de entrada USB. Se basa en la unidad de procesamiento de visión (VPU) Intel Movidius Myriad, que cuenta con un acelerador de hardware para inferencia DNN [7].

Fig. 3 Intel Neural Compute Stick 2.

Fig. 4 Google Coral USB Accelerator.

- **Google Coral USB Accelerator** (ver Fig. 4). Proporciona un coprocesador Edge TPU a los dispositivos, permitiendo desarrollar tareas de aprendizaje automático de alta velocidad en una amplia gama de sistemas, simplemente conectándolo a un puerto USB [8].

- **Nvidia Jetson.** Es un módulo de cálculo con GPU que proporcionan un gran rendimiento y eficiencia energética para ejecutar aplicaciones de inteligencia artificial. Los módulos están disponibles en varias combinaciones de rendimiento, formato y kits de desarrollo [9]. En la Fig. 5 se puede ver el kit de desarrollo denominado Jetson Nano.

Fig. 5 Jetson Nano.

2.4 Cloud Robotics.

Los sistemas robóticos basados en la Nube aprovechan una amplia gama de servicios que proporciona ésta para ofrecer ventajas tangibles como la reducción de costes, potentes capacidades de cálculo, descarga de datos, etc. Sin embargo, la naturaleza centralizada de la computación en la Nube no se adapta bien a la multitud de servicios que se utilizan hoy en día en los sistemas robóticos y que requieren estrictas garantías de tiempo real y seguridad. La computación en el borde y la computación en la niebla son enfoques complementarios que pretenden mitigar algunos de estos retos proporcionando capacidades de computación más cercanas a los usuarios.

En [10] se pueden encontrar robots que utilizan los diferentes paradigmas de computación. Entre los que presentan un propósito social destacan:

Misty

Fig. 6 Robot Misty.

Es un robot de última generación pensado especialmente para la mejora de la vida de las personas mayores que viven solas y facilitar la tarea de los cuidadores (ver Fig. 6).

Es capaz de crear mapas en 3D y se mueve de forma autónoma y dinámica en respuesta a su entorno. Reconoce caras y objetos y entiende las órdenes de voz. Además, está diseñado para ser fácil de programar e integrar con paquetes de detección, visión y aprendizaje automático [11].

NAO

Es un robot humanoide diseñado para interactuar con las personas (ver Fig. 7). Posee 25 grados de libertad y 7 sensores táctiles que le permiten moverse y adaptarse a su entorno. Además, tiene 4 micrófonos direccionales, altavoces, reconocimiento de voz y diálogo en 20 idiomas por lo que puede interactuar con las personas. Por último, incorpora dos cámaras 2D para reconocer formas, objetos e, incluso, personas.

Se utiliza en la investigación, la educación y la asistencia sanitaria. También es de plataforma abierta y totalmente programable luego se pueden desarrollar nuevas funciones y habilidades [12].

Fig. 7 Robot NAO.

ASIMO

Fig. 8 Robot ASIMO.

Es un robot humanoide que pretende ayudar a las personas que carecen de movilidad completa en sus cuerpos (ver Fig. 8). Tiene la capacidad de detectar el movimiento de las personas gracias a la multitud de sensores que incorpora. Esta funcionalidad le permite anticiparse a obstáculos manteniendo el equilibrio.

También puede realizar tareas que requieran de una elevada precisión ya que tiene la posibilidad mover cada uno de sus dedos de manera independiente [13].

EMIEW3

Es un robot desarrollado para prestar servicios como saludar a los clientes o guiarlos en espacios públicos como estaciones y aeropuertos, así como en empresas privadas (ver Fig. 9). Su objetivo es ayudar en la vida diaria y convivir con los humanos.

Utiliza la computación en la Nube para el reconocimiento de voz, la detección de la ubicación o el procesamiento del lenguaje, entre otros. Sólo se ejecutan en el robot las funciones esenciales que requieren una respuesta en tiempo real, como la evitación de colisiones [14].

Fig. 9 Robot EMIEW3.

3 Diseño del sistema.

En este capítulo se expondrán las diferentes arquitecturas, dispositivos y componentes software empleados para la puesta en marcha del robot.

Por un lado, se van a describir las mejoras que se han realizado en los módulos de interacción humano-robot desarrollados en [1], y la adaptación del paquete de navegación para que funcione con el sistema locomotriz del robot real y con los sensores que incorpora.

Por otro, se va a desarrollar una aplicación "on cloud" que va permitir almacenar los modelos de Machine Learning entrenados y los mapas generados. Además, va a incorporar una funcionalidad para descargar dichas configuraciones en el robot desde la Nube.

Por último, se describirá la integración de todos los componentes desarrollados.

3.1 Arquitectura del sistema.

En cuanto al hardware se ha utilizado un chasis de madera con tres ruedas. Dos de estas son motoras e independientes, permitien al robot girar sobre si mismo. La otra es una rueda loca de tipo castor que proporcionar estabilidad al conjunto. En lo referente a sensores propioceptivos cada rueda cuenta con un encóder [15] que permite estimar el número de revoluciones del eje del motor y, por ende, de las ruedas y, por tanto, calcular la velocidad lineal. Con los datos proporcionados por los encóders, el sistema calcula la odometría usando la técnica de "dead reckoning", que consiste en calcular la pose (posición y orientación), considerando solamente las velocidades lineales y angulares estimadas de las ruedas del sistema motriz. Para controlar las ruedas motoras, se ha utilizado el circuito controlador Devantech MD49, que por medio de un driver permite programar su funcionamiento. Los sensores exteroceptivos que se han empleado son un dispositivo de barrido láser 2D (tipo LIDAR), y un sensor RGB-D. Ambos se han utilizado, principalmente, en tareas de navegación. Para gestionar todos los componentes del robot se ha utilizado una *Raspberry Pi 4* que actúa como elemento central de procesamiento.

En lo referente al software, el módulo *Raspberry Pi 4* ejecuta el sistema operativo Linux, bajo la distribución Ubuntu 20.04, donde se ha instalado el ecosistema de ROS Noetic para controlar y monitorizar el robot. En el espacio de trabajo de ROS se desplegarán los paquetes que permiten utilizar las prestaciones de los sensores exteroceptivos, además de un driver ROS para usar el sistema de locomoción MD49 desarrollado por la Dra. Nieves Pavón, directora de este Trabajo Fin de Máster.

Además, se utiliza un ordenador personal con el sistema operativo Ubuntu 18.04 junto a ROS Melodic, sobre el cual se van a desplegar los paquetes que permiten la interacción humano-robot y los módulos del paquete de navegación, con el fin de facilitar el desarrollo de los experimentos y el análisis de resultados.

Por último, se ha desarrollado una aplicación que permite almacenar las diferentes configuraciones del robot y descargarlas sobre el mismo. Esta aplicación se ha implementado utilizando la plataforma y los servicios que proporciona Google Cloud.

Por lo tanto, se distinguen dos partes bien diferenciadas de la arquitectura del sistema, el hardware y el software que a su vez está dividido en los módulos que ejecutan sobre los dispositivos y los que se despliegan en la Nube. En los apartados 3.2, 3.3, 3.4, 3.5 y 3.6 se explicará cada parte de forma detallada.

3.2 Arquitectura hardware.

Se ha utilizado un chasis de madera (ver Fig. 10), en el que están montados los componentes que se describen en los apartados 3.2.1, 3.2.2, 3.2.3, 3.2.4, 3.2.5, 3.2.6 y 3.2.7. El conexionado de los diferentes elementos se pueden ver en la Fig. 11.

Fig. 10 Robot.

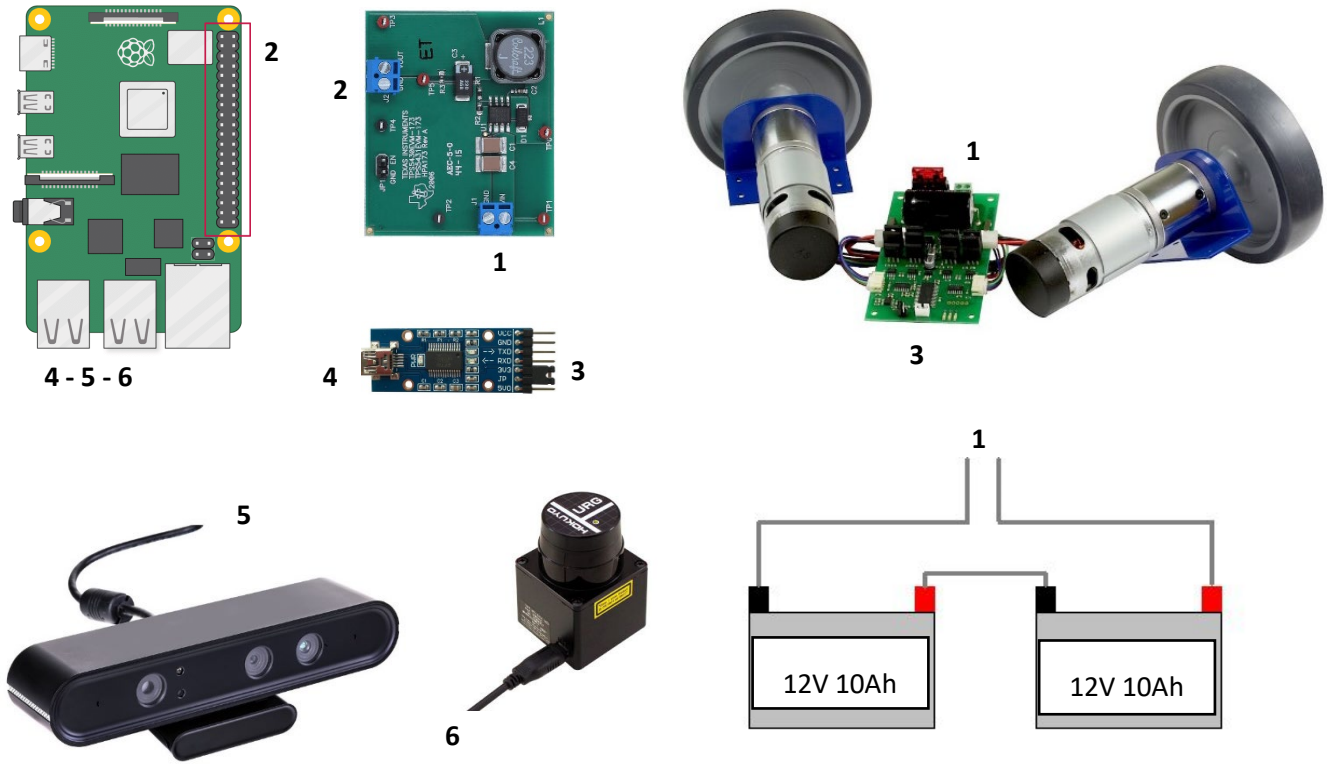

Fig. 11 Esquema conexión arquitectura hardware.

3.2.1 Raspberry Pi 4.

Es un ordenador de placa única (Single Board Computer, SBC) que actúa como elemento central de procesamiento (ver Fig. 12). Se ha utilizado la versión de 8 GB. Las especificaciones se detallan en la Tabla 1.

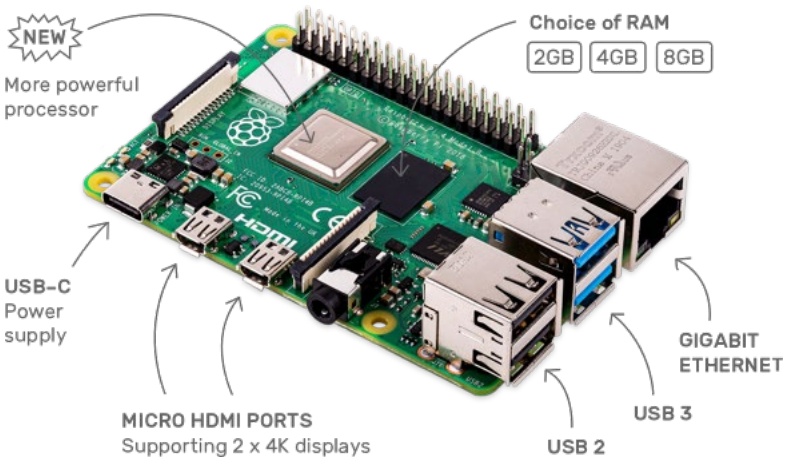

Fig. 12 Raspberry Pi 4.

Tabla 1 Especificaciones Raspberry Pi 4.

Model No.	Raspberry Pi 4
Procesador	CPU ARM Cortex-A72 de 64 bits con cuatro núcleos a 1,5 GHz
Memoria RAM	2 GB, 4 GB o 8 GB de SDRAM LPDDR4
Tarjeta gráfica	Gráficos de VideoCore VI, compatibles con OpenGL ES 3.x
Conectividad	Gigabit Ethernet , red inalámbrica 802.11ac de doble banda y bluetooth 5.0
Vídeo y sonido	Dos puertos micro-HDMI que admiten pantallas de 4K@60Hz a través de HDMI 2.0, puerto de pantalla MIPI DSI, puerto de cámara MIPI CSI, salida estéreo de 4 polos y puerto de vídeo compuesto.
Puertos	Dos puertos USB 3.0 y dos puertos USB 2.0
Alimentación	5V/3A vía USB-C, 5V vía cabezal GPIO

3.2.2 Controlador hardware MD49 de Devantech.

El controlador MD49 (ver Fig. 13), facilita la conexión de dos motores modelo EMG49 para construir un sistema de tracción diferencial que se puede montar en un robot móvil.

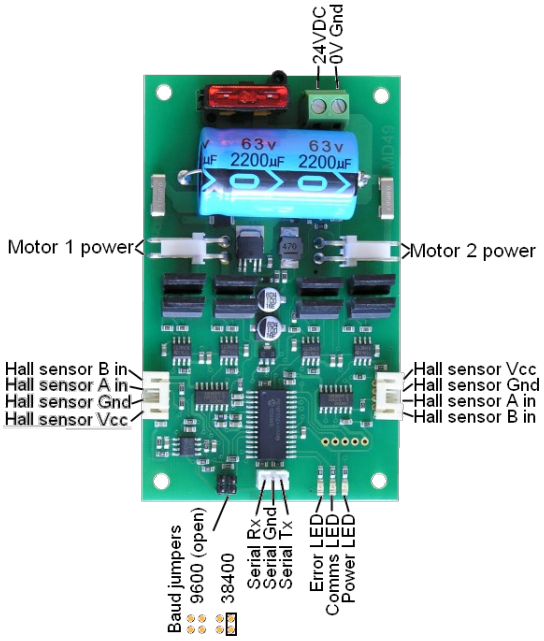

Fig. 13 Controlador hardware MD49 de Devantech.

El modelo EMG49 es un motor (ver Fig. 14) a 24 V equipado con encóders y con reducción 49:1. Entre sus características destacan un par de 16 Kg/cm, una velocidad promedio de 122 rpm y una intensidad promedio de 2100 mA. En vacío (sin carga), alcanza 143 rpm y consume 500 mA. Proporciona una potencia de salida media de 37.4 W. Los encóders proveen 980 tics por vuelta, por lo que la resolución es de 2*π/980 radianes. Las características completas pueden consultarse en [16].

Fig. 14 Motor EMG49.

Habitualmente, el conjunto de tracción, suele comprarse como el kit de locomoción RD03, que incluye dos ruedas y soportes para las mismas como muestra la Fig. 15.

Fig. 15 Kit de locomoción RD03.

El módulo del controlador MD49 requiere 24 V para alimentar el módulo, las características completas pueden consultarse en [17]. El controlador opera con un bus serie TTL y no se puede conectar directamente a un puerto RS232. Dicha conexión debe llevarse a cabo mediante un convertidor de nivel de voltaje, como un ST232. El protocolo para intercambiar información entre una aplicación y el controlador se describe a continuación.

El formato de intercambio de información se materializa enviando un conjunto de tramas (escritura), a través del puerto serie, y leyendo la respuesta proporcionada por el controlador. El patrón de típico de tramas a intercambiar es:

Byte de sincronismo	Byte de operación	Byte de dato

El número de bytes retornados por el controlador dependerá del tipo de operación solicitada. Por ejemplo, para enviar un comando de velocidad al motor 1 (de los dos motores conectados al controlador), se utilizará la trama siguiente:

Byte de sincronismo	Byte de operación	Byte de dato
0x00	0x31	0XFF

El controlador no enviará ningún dato de respuesta, en este caso. La velocidad indicada viene dada por un byte (255 si se usa un formato de números sin signo, y 127 si se considera un formato de números con signo). En cualquier caso, 0xFF es la velocidad máxima hacia adelante.

Si, por el contrario, se desea consultar la velocidad real, es posible realizar una lectura de los dos encóders simultáneamente. En este caso, la trama a enviar es:

Byte de sincronismo	Byte de operación	Byte de dato
0x00	0x25	

En este caso, no se envía ningún byte de dato, y el controlador retornará 8 bytes que representan las lecturas de ambos encóders:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Cuenta del encóder 1				Cuenta del encóder 2			

Para realizar correctamente la interpretación de la respuesta recibida, es necesario construir un número de 4 bytes, es decir, de 32 bits, con signo, para representar la cuenta de cada encóder. En un lenguaje como C, estas operaciones se realizan utilizando los operadores lógicos a nivel de bit y de desplazamiento. Es importante resaltar que cada cuenta de encóder debe almacenarse en número entero de 32 bits con signo. Por ejemplo, en C, es posible utilizar el tipo int para este propósito, ya que, habitualmente dicho tipo representa un número con signo de 32 bits. Sin embargo, el tamaño (sizeof), de un int dependerá del compilador, por lo que es necesario

asegurarse de que usamos un tipo entero de estrictamente 32 bits. En los paquetes ROS diseñados se usa el tipo int32_t.

Para calcular el valor de rpm (vueltas), es necesario conocer el tiempo transcurrido entre dos lecturas de los encóders. Así, si tomamos la hora del sistema al hacer una lectura (tics1), por ejemplo, t1, y la hora del sistema al hacer la segunda lectura (tics2), por ejemplo, t2, el número de vueltas por minuto se calculará como:

$$vueltas = \frac{(tics2 - tics1) \cdot 60}{res \cdot (t2 - t1)}$$

Donde res es la resolución del encóder (980 para el controlador MD49).

Se ha usado la siguiente regla de tres simple: si durante t2-t1 segundos, se ha obtenido un número de revoluciones de (tics2-tics1)/res, entonces durante 60 segundos (1 minuto), obtenemos vueltas rpm.

Para evitar que el valor de las cuentas de los encóders se incremente sin control, se ha implementado una puesta a cero de los encóders, inmediatamente después de hacer una lectura de los mismos. De ese modo, tics1 se supone que es siempre 0. Así, desde el instante en que se ponen a cero los encóders en t1, hasta la siguiente lectura realizada (tics2) en t2, el número de vueltas por minuto o rpm se calcula como:

$$vueltas = \frac{(tics2) \cdot 60}{res \cdot (t2 - t1)}$$

Por ejemplo, en el controlador MD49, si el número de tics calculados durante 250 ms es de 230, el número de vueltas por minuto es de:

$$vueltas = \frac{(230) \cdot 60}{980 \cdot 0.2} = \frac{13800}{49} = 281.63$$

Pero realmente, más que el valor de rpm, es necesario conocer el ángulo que el eje ha descrito durante esos 250 ms. Tomando el valor de rpm, se puede calcular el ángulo haciendo las operaciones oportunas, sin embargo, se puede simplificar el cálculo, considerando, directamente la resolución de los encóders directamente para calcular dicho ángulo. En el controlador MD49 se tiene una resolución de 980 tics por vuelta, por tanto, cada tic se corresponde con 360/980 grados, es decir 0,36 grados o, lo que es lo mismo 0,36*π/180 radianes.

Es posible calcular el ángulo como:

$$\phi = tics \cdot resangular$$

Siendo la resolución angular de 0.00628 radianes para el controlador MD49.

Dado que se conoce el tiempo transcurrido entre que se pusieron los encóders a cero y se midió el número de tics, pasado ese tiempo t, la velocidad angular se puede calcular como:

$$\frac{\phi}{t} = \frac{(tics \cdot resangular)}{t} = \omega$$

Por otra parte, conociendo el ángulo, es posible saber el espacio s recorrido por una rueda de radio r, conectada directamente al eje del motor:

$$s = \phi \cdot r$$

La velocidad lineal v, para una rueda de radio r, se calcularía como s/t, por tanto:

$$v = \frac{s}{t} = \frac{\phi \cdot r}{t} = \frac{tics \cdot resangular \cdot r}{t} = \omega \cdot r$$

3.2.3 Hokuyo URG-04LX-UG01.

Es un sensor de barrido láser (ver Fig. 16) de pequeño tamaño (50x50x70mm), y 160g de masa. Funciona con una conexión miniUSB/USB que sirve tanto para la alimentación del dispositivo como para la transferencia de datos recogidos por el sensor y el envío de comandos al microcontrolador interno [18].

Fig. 16 Hokuyo URG-04LX-UG01.

El láser tiene un rango de ±120° aproximadamente. La resolución angular es de 0.36° luego el rango está dividido en 666 arcos, en cada uno de los cuales el dispositivo proporciona una medida de la distancia entre el entorno y él con un alcance máximo de 5600 mm. La fuente de luz del sensor es un diodo semiconductor láser infrarrojo de 785nm de longitud de onda con seguridad de "clase 1", es decir, el sensor es seguro para cualquier condición de funcionamiento y no requiere medidas de seguridad. Hay que tener en cuenta que para un correcto funcionamiento ha de ser utilizado sólo en interiores.

El conjunto de las características se pueden ver en la Tabla 2.

Tabla 2 Características Hokuyo URG-04LX-UG01.

Model No.	URG-04LX-UG01
Power source	5VDC±5%(USB Bus power)
Light source	Semiconductor laser diode(λ=785nm), Laser safety class 1
Measuring area	20 to 5600mm(white paper with 70mm×70mm), 240°
Accuracy	60 to 1,000mm : ±30mm 1,000 to 4,095mm : ±3% of measurement
Angular resolution	Step angle : approx. 0.36°(360°/1,024 steps)
Scanning time	100ms/scan
Noise	25dB or less
Interface	USB2.0/1.1[Mini B](Full Speed)
Command System	SCIP Ver.2.0
Ambient illuminance	Halogen/mercury lamp: 10,000Lux or less, Florescent: 6000Lux(Max)
Ambient temperature/humidity	-10 to +50 degrees C, 85% or less(No condensation, no icing)
Vibration resistance	10 to 55Hz, double amplitude 1.5mm each 2 hour in X, Y and Z directions
Impact resistance	196m/s^2, Each 10 time in X, Y and Z directions
Weight	Approx. 160g

Su implementación es ROS es directa ya que cuenta con una paquete ya desarrollado denominado hokuyo node [19], el cual permite la comunicación con el ordenador mediante la publicación de la nube de puntos que se obtiene del láser Hokuyo en el *topic scan.*

3.2.4 ORBBEC ASTRA PRO.

Es una cámara RGB-D (ver Fig. 17) de tamaño 160x30x40mm, y 300g de masa. Cuenta con una cámara RGB que proporciona una imagen de 1920x1080 a 30 fps y una imagen de profundidad de 640x480 a 30 fps que percibe la profundidad de los objetos entre 0.6 metros y 8 metros. También tiene 2 micrófonos [20].

Fig. 17 Orbbec Astra Pro.

El conjunto de las características se pueden ver en la Tabla 3.

Tabla 3 Características Orbbec Astra Pro.

Product Name	Astra Pro Plus
Range	0.6m – 8m
FOV	60°H x 49.5°V x 73°D
RGB Image Resolution	1920 x 1080 @30fps
Depth Image Resolution	640 x 480 @30fps
Size	165mm x 30mm x 40mm
Temperature	10 – 40°C
Power Supply	USB 2.0
Power Consumption	< 2.5 W
Operating Systems	Android / Linux / Windows7/8/10
SDK	Astra SDK or OpenNI 2 or 3rd Party SDK
Precision	+/- 1 – 3mm @1 m
Microphones	2 Built-in

Su uso está destinado principalmente para las siguientes aplicaciones:

- Control por gestos.
- Robótica.
- Digitalización 3D.
- Desarrollo de Nubes de puntos.
- Sistemas interactivos.
- VR/AR.

Su implementación en ROS es directa ya que cuenta con una paquete ya desarrollado denominado astra_camera [21] el cual proporciona tanto imagen como Nube de puntos.

3.2.5 Reductor CC/CC TPS5430EVM-173.

El rango de voltaje de entrada de este módulo (ver Fig. 18) es de 9.0 V a 36 V y el voltaje de salida está preestablecido a 5 V. La tensión de salida puede ajustarse fácilmente cambiando un divisor de resistencias [22]. Se ha utilizado para proporcionar la alimentación de la *Raspberry Pi 4*. Esto es debido a que la batería que se emplea para alimentar al robot es de 24V, mientras que la *Raspberry Pi 4* soporta 5V.

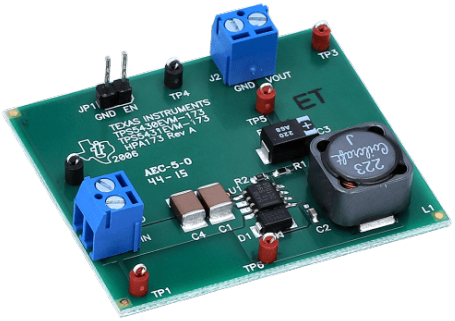

Fig. 18 TPS5430EVM-173.

3.2.6 Adaptador serie TTL a USB.

Adaptador (ver Fig. 19) que permite conectar y comunicar la placa MD49 con la Raspberry Pi 4, mediante la conexión serie disponible.

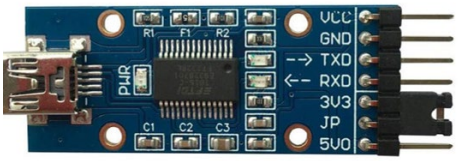

Fig. 19 Adaptador serie TTL a USB.

3.2.7 Batería de plomo ácido de 12V 10Ah.

Un par de baterías (ver Fig. 20) en serie para obtener 24 V. Se emplea para suministrar corriente tanto al MD49 y el sistema de motorización como a la *Raspberry Pi 4*.

Fig. 20 Batería plomo ácido 12V 10Ah.

3.3 Módulos Edge Computing.

Por un lado en los apartados 3.3.1, 3.3.2 y 3.3.3 se detallan los módulos que permiten la interacción humano robot.

Por otro, se emplea el paquete de Navegación de ROS junto con el controlador MD49 y los paquetes ya existentes para el sensor láser Hokuyo y la cámara RGB-D Orbbec para dotar al robot de capacidad para navegar de forma autónoma.

3.3.1 Reconocimiento facial.

Para desarrollar el método de reconocimiento facial se ha partido del trabajo expuesto en [1]. El resultado de la inferencia mediante el modelo *FaceNet* es un tensor de características de 512 elementos. En [1] se comparaba dicho tensor con cada uno de los almacenados anteriormente utilizando la distancia euclídea. Si dicha distancia era inferior a un umbral (se seleccionó empíricamente 1,24), se reconocía al usuario.

Para aumentar la robustez del método de reconocimiento se han entrenado una serie de modelos evitando así que la identificación de la persona se realice empíricamente. Ahora bien, la problemática común a la hora de entrenar un modelo de Deep Learning (la falta de datos), ha sido evidente. Por ello, para mejorar la calidad del *dataset*, se ha incrementado su volumen y la variedad de las imágenes que lo forman, empleando técnicas de *Data Augmentation*. Esta estrategia consiste en aplicar diferentes transformaciones a las imágenes originales y generar otras nuevas de manera sintética.

El proceso completo que se ha seguido se observa en la Fig. 21.

Fig. 21 Flujo entrenamiento reconocimiento facial.

1. En primer lugar se extraen los rostros de las personas que se quieren identificar (se ha utilizado el mismo modelo *Caffe* que se usó en [1]), y se guardan automáticamente en una carpeta con su correspondiente nombre. Asimismo, se extraen los rostros de imágenes de personas ajenas a la identificación para utilizarlas en el entrenamiento de los modelos de clasificación.
2. Luego se aplican técnicas de *Data Augmentation* de forma que se consiguen imágenes que pueden ser fieles a la realidad. El nuevo *Dataset* generado se guarda en otra carpeta, manteniendo el nombre de la persona para mantener la trazabilidad. En concreto se han aplicado las transformaciones que se pueden ver en la Fig. 22 usando la librería *imgaug* [23] tal y como muestra el Fragmento de código 1.

Fig. 22 Técnicas Data Augmentation.

```
import imgaug as ia
from imgaug import augmenters as aug

def reflect(image):
    image_aug = aug.Fliplr(1)(image=image)
    return image_aug

def bright(image):
    brightness = iaa.AddToBrightness((-30, 30))
    image_brightness = brightness.augment_image(image)
```

```
    return image_brightness


def apply_color(image):
    color = iaa.Add((-30, 30), per_channel=1)
    image_color = color.augment_image(image)
    return image_color

def motion_blur(image):
    blur = iaa.AverageBlur(k=(5))
    image_blur = blur.augment_image(image)
    return image_blur
```

Fragmento de código 1 Técnicas Data Augmentation.

3. Posteriormente se obtienen los vectores de características normalizados (salida del modelo *FaceNet* utilizado en [1]), de las personas que se quieren identificar y se les asigna el número 0. También se obtienen los vectores de características del resto de personas asignándoles el número 1. De este modo se consigue diferenciar entre el sujeto que se quiere reconocer y el que no. Estos datos se guardan de forma automáticamente en una tabla Excel utilizando la librería *Pandas* [24], que se usará para el entrenamiento de los modelos.
4. Con los datos obtenidos se entrenan los siguientes modelos:

   a. Red neuronal profunda (DNN).

Se han realizado diferentes pruebas variando el número de capas ocultas de la red manteniendo la entrada (vector de características de 512 elementos) y la salida con una neurona dado que se trata de un problema de clasificación binaria. Las funciones de activación utilizadas son las siguientes:

Función ReLU: para la capa de entrada y las capas ocultas. El comportamiento de esta función se muestra en la Fig. 23. Esta función generará una salida igual a cero cuando la entrada sea negativa, y una salida igual a la entrada cuando sea positiva [25]. Se trata de la función más utilizada debido a que:

- No existe la saturación por lo que el algoritmo del gradiente descendente converge rápidamente, facilitando así el entrenamiento.
- Requiere un cálculo computacional menor que otras funciones.

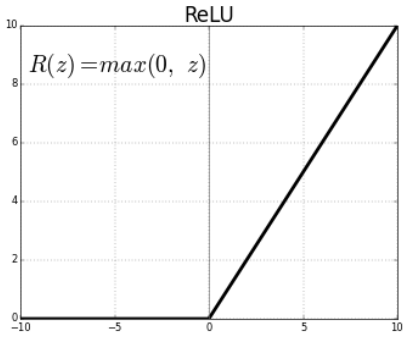

Fig. 23 Función activación ReLU.

Función sigmoidal: para la capa de salida. Esta función de activación toma cualquier rango de valores a la entrada y los mapea al rango de 0 a 1 a la salida [25]. Dicho comportamiento se muestra en la Fig. 24. La aplicación principal de esta función es la clasificación binaria.

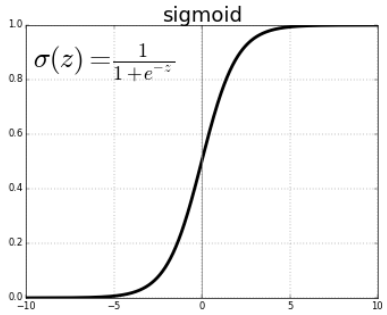

Fig. 24 Función de activación sigmoidal.

La obtención de un conjunto de pesos para que la solución tenga un error nulo de forma analítica no suele ser posible, por ello se emplean algoritmos iterativos de optimización para acercarse a la solución ideal. Las opciones son muy variadas, pero casi todas tienen en común el uso de la variación del error con respecto a la variación de los pesos como en un descenso de gradiente, aunque cada uno lo implementa con una fórmula distinta en el proceso de optimización [1]. En este caso se ha utilizado el descenso de gradiente estocástico (SGD) [26] y la estimación adaptativa de momentos (ADAM) [27]. En el Fragmento de código 2 se puede ver como se crea la red neuronal y se entrena.

```
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

def classification_model():
    model = Sequential()
```

```
    model.add(Dense(5, activation='relu', input_shape=(512,)))
    model.add(Dense(3, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='SGD',
metrics=['accuracy'])
    return model


model = classification_model()


history = model.fit(X_train_sc, y_train, validation_split=0.2, epochs=20,
verbose=2)
```

Fragmento de código 2 Entrenamiento DNN.

b. XGBoost.

Es una biblioteca de aprendizaje automático supervisado distribuida y escalable de árboles de decisión con potenciación de gradiente (GDBT). Es la principal biblioteca de aprendizaje automático supervisado para los problemas de regresión y clasificación [28].

El aprendizaje automático supervisado utiliza algoritmos para entrenar un modelo con el fin de encontrar patrones en un conjunto de datos con etiquetas y características y, a continuación, utiliza el modelo entrenado para predecir las etiquetas en las características de un nuevo conjunto de datos como se puede ver en la Fig. 25.

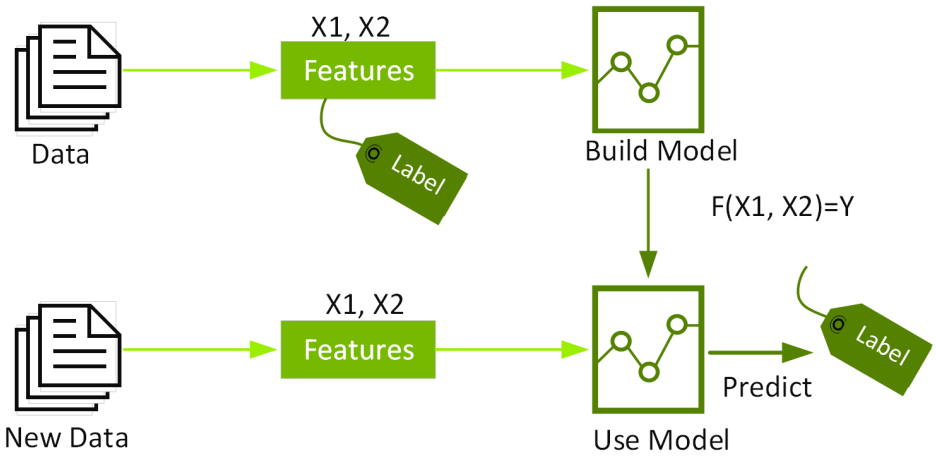

Fig. 25 Aprendizaje automático supervisado.

Los árboles de decisión crean un modelo que predice la etiqueta mediante la evaluación de condicionales (Fig. 26), y la estimación del número mínimo de condiciones necesarias para evaluar la probabilidad de tomar una decisión correcta.

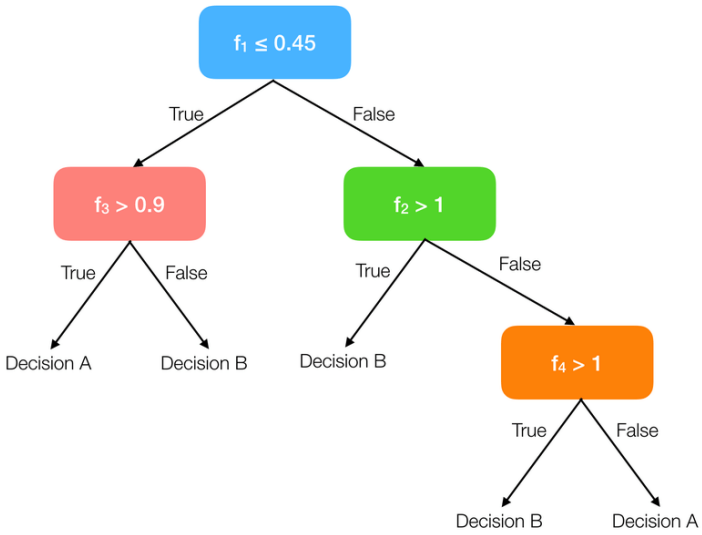

Fig. 26 Árbol de decisión.

GDBT es un algoritmo de aprendizaje de conjunto de árboles. La idea detrás de la potenciación (boosting) es generar múltiples modelos de predicción débiles secuencialmente, y que cada uno de estos tome los resultados del modelo anterior, para generar un modelo más fuerte, con mejor poder predictivo y mayor estabilidad en sus resultados [29].

Para conseguir un modelo más fuerte a partir de estos modelos débiles, se emplea el algoritmo de optimización del descenso de gradiente. Durante el entrenamiento, los parámetros de cada modelo débil son ajustados iterativamente tratando de encontrar el mínimo de una función objetivo, que puede ser la proporción de error en la clasificación, el área bajo la curva (AUC o Area Under the Curve), o la raíz del error cuadrático medio (RMSE), entre otros.

Cada modelo se compara con el anterior. Si un nuevo modelo tiene mejores resultados, entonces se toma este como base para realizar modificaciones. Si, por el contrario, tiene peores resultados, se regresa al mejor modelo anterior y se modifica ese de una manera diferente. Qué tan grandes son los ajustes de un modelo a otro es uno de los hiperparámetros que debe definir el usuario.

Este proceso se repite hasta llegar a un punto en el que la diferencia entre modelos consecutivos es insignificante, lo cual nos indica que hemos encontrado el mejor modelo posible, o cuando se llega al número de iteraciones máximas definido por el usuario.

El punto más complejo a la hora de usar este algoritmo es seleccionar los hiperparámetros ya que son elevados y el rendimiento de un modelo depende en gran medida de ellos Los hiperparámetros son parámetros ajustables que permiten controlar el proceso de entrenamiento de los modelos.

El ajuste de hiperparámetros, también denominado optimización de hiperparámetros es el proceso de encontrar la configuración de estos que produzca el mejor rendimiento. Normalmente, el proceso es manual y costoso desde el punto de vista computacional, sin embargo existen librerías como *Scikit-learn* [30] que tienen clases que permiten automatizar esta optimización. Se han utilizado dos:

- ***GridSearchCV:*** Para construir los modelos se utilizan todas las permutaciones posibles de los hiperparámetros. Se evalúa el rendimiento de cada modelo y se selecciona el de mejor rendimiento.
- ***RandomizedSearchCV***: En lugar de proporcionar un conjunto discreto de valores para explorar en cada hiperparámetro, se proporciona una distribución estadística o lista de hiperparámetros. Los valores de los diferentes hiperparámetros se conforme a esta distribución. En el Fragmento de código 3 se puede ver como se usa este mñetodo de optimización.

```
import xgboost as xgb


xgb_model = xgb.XGBClassifier(objective="binary:logistic",
use_label_encoder=False, random_state=1)
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)


xgb_grid = RandomizedSearchCV(xgb_scikit,params,cv=5,n_jobs=-2,n_iter=20)
xgb2 = xgb_grid.best_estimator_
xgb2.fit(X_train, y_train)
y_pred2 = xgb2.predict(X_test)
```

Fragmento de código 3 Entrenamiento XGBoost.

c. Algoritmo de k vecinos más cercanos (KNN).

Es un método de aprendizaje supervisado utilizado para clasificar objetos en función de la proximidad a agrupaciones de espacios de características [31]. Un objeto se clasifica por el voto mayoritario de sus vecinos, es decir, se asigna a la clase que se repite con más frecuencia entre sus k vecinos, donde k es un número entero positivo (ver Fig. 27). En el algoritmo KNN, variando el parámetro k se obtienen resultados diferentes, el valor se suele fijar tras un proceso de pruebas.

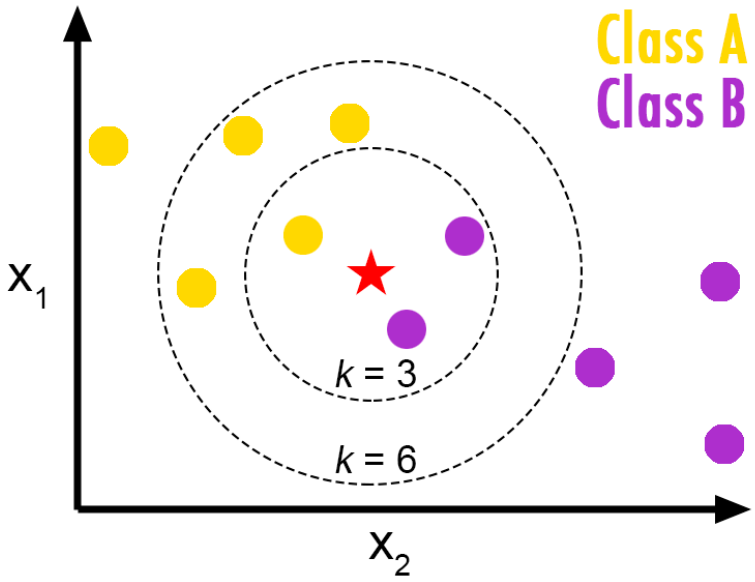

Fig. 27 KNN clasificador binario.

Se ha utilizado la librería *Scikit-learn* [30] que proporciona la clase *KNeighborsClassifie*r y se han hecho pruebas modificado el valor de k. En el Fragmento de código 4 se observa cómo se crea y se entrena este modelo.

```
from sklearn.neighbors import KNeighborsClassifier


KNN = KNeighborsClassifier(n_neighbors=2)
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
```

Fragmento de código 4 Entrenamiento KNN.

d. Máquina de soporte vectorial (SVM).

Es un algoritmo de aprendizaje supervisado que permite clasifica datos en dos o más clases incluso cuando la separación entre clases no es lineal. Durante la fase de entrenamiento, SVM genera un hiperplano de forma iterativa hasta encontrar el hiperplano marginal máximo que separa las diferentes clases [32] (ver Fig. 28). En el caso de que las clases no sean linealmente separables, es decir, no exista un hiperplano que las divida, se puede aplicar un kernel para aumentar la dimensión del espacio.

En este caso también se ha utilizado la librería *Scikit-learn* [30] que proporciona la clase SVM y se ha seleccionado un kernel lineal y polinómico. El Fragmento de código 5 muestra cómo se crea este modelo y se entrena.

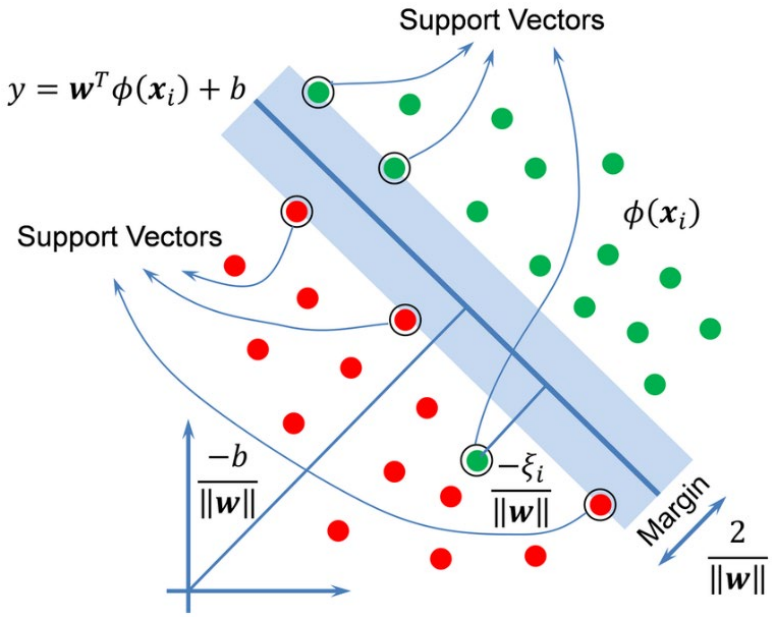

Fig. 28 SVM clasificador binario.

```
from sklearn.svm import SVC

SVM = SVC(kernel='linear')
SVM.fit(X_train, y_train)
y_pred = SVM.predict(X_test)
```

Fragmento de código 5 Entrenamiento SVM.

5. Por último, se validan los modelos para comprobar cuál es el que obtiene el mejor resultado. Para ello se ha obtenido la matriz de confusión (ver Fig. 29), que es una herramienta que permite analizar los resultados de cómo trabaja un algoritmo de aprendizaje supervisado. Esta matriz se presenta siempre en forma de tabla, de manera que en cada columna aparece el número de predicciones de cada clase, mientras que cada fila muestra el número real de instancias de cada clase. Los datos que proporciona son los siguientes:

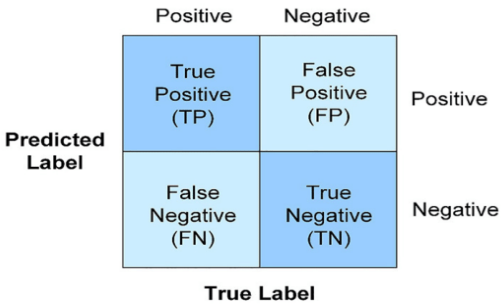

Fig. 29 Matriz de confusión.

TP (*True Positive*): son los valores que el algoritmo clasifica como positivos y que realmente son positivos.

TN (*True Negative*): son valores que el algoritmo clasifica como negativos y que realmente son negativos.

FP (*False Positive*): son valores que el algoritmo clasifica como positivo cuando realmente son negativos.

FN (*False Negative*): son valores que el algoritmo clasifica como negativo cuando realmente son positivos.

Esta matriz es la base sobre la que se construyen las métricas de clasificación:

- **Exactitud (*Accuracy*):** representa el porcentaje total de valores correctamente clasificados, tanto positivos como negativos.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precisión (*Precision*)**: se utiliza para saber qué porcentaje de valores que se han clasificado como positivos son realmente positivos.

$$Precision = \frac{TP}{TP + FP}$$

- **Sensibilidad (*Recall*):** se utiliza para saber cuántos valores positivos son correctamente clasificados.

$$Recall = \frac{TP}{TP + FN}$$

- **Índice F1 (*F1-score*)**: se calcula como la media armónica entre la precisión y la sensibilidad. Es una métrica equilibrada entre tener pocos falsos positivos y pocos falsos negativos.

$$F1\text{–}Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

La librería *Scikit-learn* [30] también proporciona clases para obtener la matriz de confusión y las diferentes métricas. El Fragmento de código 6 muestra cómo se obtienen.

```
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score


print(confusion_matrix(y_test, y_pred))
```

```
print(classification_report(y_test, y_pred, digits=4))
print(accuracy_score(y_test, y_pred))
```

Fragmento de código 6 Matriz de confusión y métricas.

3.3.2 Reconocimiento de voz.

En los sistemas de reconocimiento de voz (STT) se consideran tres tipos diferentes de errores:

- **Sustituciones (S):** la inferencia proporciona una palabra incorrecta en la oración reconocida.
- **Borrados (B):** la inferencia no es capaz de devolver una palabra en la frase reconocida.
- **Inserciones (I):** la inferencia devuelve una palabra extra en la oración reconocida.

Para evaluar la precisión de un sistema STT se utilizan estos tres errores, la métrica se conoce como *Word Error Rate* (WER) [33].

$$WER = \frac{S + B + I}{M}$$

donde M representa el conjunto de palabras correctas en la oración reconocida. En general, para calcular el WER primero es necesario alinear la secuencia de palabras reconocidas con la secuencia de palabras correcta [34].

Para comprobar las prestaciones de un sistema STT se pueden utilizar conjunto de datos de comunicación, en este caso se ha comprobado la métrica WER utilizando el *Dataset MediaSpeech* [35]. Las pruebas de rendimiento de varios modelos se ven en la Tabla 4.

Tabla 4 WER Modelos STT.

Modelo	*Wit*	*Azure*	*Quartznet*	*Vosk*	*Google*	*wav2vec*	*SIlero*	*Deepspeech*
WER Español	0.0879	0.1296	0.1826	0.1970	0.2176	0.2469	0.3070	0.4236

La mayor debilidad que se encontró en el Trabajo Fin de Grado que se detalla en [1] radica en el proceso de reconocimiento de voz. Las pruebas de rendimiento anteriores corroboran dicha flaqueza. Para solventarlo se ha comprobado el funcionamiento de los modelos *Silero* y *Vosk* que también se ejecutan en modo offline:

Silero

Es un modelo de conversión de voz a texto y de texto a voz pre entrenado. Ofrece uno de los mejores modelos de STT en circulación que, según los desarrolladores, alcanzaría un rendimiento similar, si no superior, a los modelos ofrecidos por Google. Dado que el modelo está pre entrenado (Fig. 30), no es posible realizar mucha experimentación con él [36], [37].

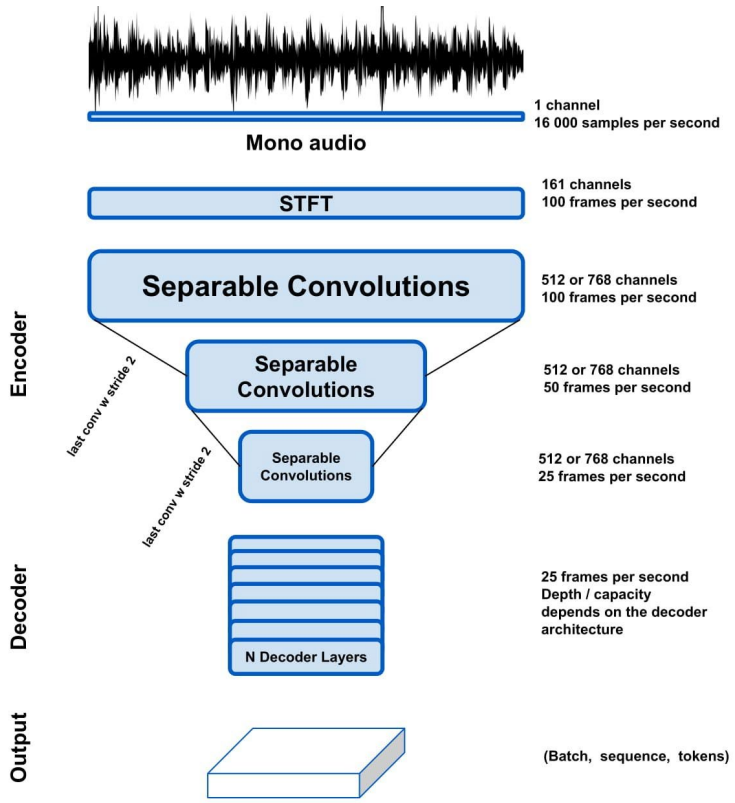

Fig. 30 Arquitectura modelos Silero.

Se ha utilizado el modelo para el idioma español que se encuentra en [37] y se ha seguido el proceso que muestra la Fig. 31.

Fig. 31 Flujo STT.

En primer lugar se carga el modelo pre entrenado de reconocimiento de voz utilizando el framework de *PyTorch* [36]. A continuación, se configuran algunas características antes de captar el audio y se inicia la grabación utilizando los métodos que proporciona la librería *PyAudio* [38]. Por último se inicia la inferencia utilizando el decodificador del modelo para obtener la transcripción del audio tal y como se puede ver en el Fragmento de código 7.

```
def run_inference(self, audio_path):
    device = torch.device('cpu')
    model, decoder = init_jit_model(self.__model_path, device=device)
    batch = read_batch([audio_path])
```

```
    input = prepare_model_input(batch, device=device)
    output = model(input)
    return decoder(output[0].cpu())
```

Fragmento de código 7 Inferencia STT Silero.

Vosk

Es un sistema de reconocimiento de voz de código abierto para 17 idiomas incluido el español. Funciona con *Kaldi Recognizer*, el cual se compone de un kit de herramientas para el reconocimiento de la voz y que se encuentra escrito en lenguaje C++ [39]. Debido a que *Vosk* se encuentra aún en desarrollo se pueden producir ciertos errores durante su ejecución conduciendo a una mala precisión. Sin embargo, proporciona cuatro niveles de adaptación para mejorar la precisión:

- Actualización del vocabulario en tiempo real durante el reconocimiento.
- Actualización *offiline* del modelo del idioma. Dependiendo de la mejora que se quiera alcanzar se puede compilar el modelo a partir de una o varias fuentes de datos (diccionario, modelo acústico y modelo de lenguaje)
- Actualización del modelo lingüístico y del diccionario en modelos grandes.
- Adaptación del modelo acústico con ajuste fino. Tan solo hay que recopilar datos (mínimo 1 hora) y transformarlos al formato que utiliza *Kaldi* [40].

En este caso se ha utilizado el modelo para el idioma español ya entrenado [41] que está optimizado para dispositivos que usen Android y para Raspberry Pi. El proceso que se ha seguido es el mimo que el anterior (ver Fig. 31), pero *Vosk* tiene su propia librería en Python que proporciona sus propios métodos para cargar el modelo. En el Fragmento de código 8 se puede ver como se realiza la inferencia.

```
def inference(self,path):
    process = subprocess.Popen(['ffmpeg', '-loglevel', 'quiet', '-i',
          path,'-ar', str(self.__rate) , '-ac', '1', '-f', 's16le', '-'],
          stdout=subprocess.PIPE)
    while True:
        data = process.stdout.read(4000)
        if len(data) == 0:
            break
        if self.__model.AcceptWaveform(data):
            print(self.__model.Result())
```

```
        else:
            print(self.__model.PartialResult())
    print(self.__model.FinalResult())
```

Fragmento de código 8 Inferencia STT Vosk.

Además, se han desarrollado dos funciones nuevas que se pueden ver en el Fragmento de código 9. La primera permite que la grabación no se inicie hasta que se detecte un umbral de sonido personalizado. La segunda recorta los silencios del inicio y del final del audio. Con estas dos funciones se obtiene un mejor resultado.

```
def trim(self,snd_data):
    def _trim(snd_data):
        snd_started = False
        r = array('h')
        for i in snd_data:
            if not snd_started and abs(i)>self.__threshold:
                snd_started = True
                r.append(i)
            elif snd_started:
                r.append(i)
        return r
    snd_data = _trim(snd_data)
    snd_data.reverse()
    snd_data = _trim(snd_data)
    snd_data.reverse()
    return snd_data

def add_silence(self,snd_data, seconds):
    silence = [0] * int(seconds * self.__rate)
    r = array('h', silence)
    r.extend(snd_data)
    r.extend(silence)
    return r
```

Fragmento de código 9 Umbral de ruido STT.

3.3.3 Procesamiento de lenguaje natural.

En [1] se utilizó la librería *spaCy-udpipe* con el modelo *spanish-gsd-ud-2.5-191206.udpipe (es-gsd).* Se logró el objetivo de analizar frases sencillas cuya característica común era que implicaban cierto movimiento. Sin embargo, en esta ocasión se busca ampliar el vocabulario y el principal problema de utilizar la librería *Spacy-udpipe* es que para mejorar el sistema experto hay que desarrollarlo de cero. Para conseguir que el sistema sea escalable se ha optado por utilizar la nueva evolución de *spaCy* (ver Fig. 32) que tiene implementado métodos para identificar patrones en las frases.

Las funciones que permiten realizar el procesamiento de lenguaje natural son, entre otras:

- **Segmentación de palabras (tokenización):** consiste en identificar todos los elementos de un texto (*tokens*).
- **Etiquetado gramatical (POS):** asigna a cada palabra su categoría gramatical.
- **Análisis de dependencia**: permite obtener el análisis sintáctico de una oración, es decir, como están relacionadas las palabras entre sí.
- **Lematización**: asigna a cada palabra su forma canónica (lema).
- **Detección de fronteras**: permite separar las oraciones de un texto.
- **Reconocimiento de entidades nombradas (NER)**: para clasificar las entidades de un texto en categorías predefinidas.

Fig. 32 Arquitectura *spaCy.*

spaCy es una librería *open-source* para procesamiento del lenguaje natural (NLP) en Python gratuita. Empleando las funciones definidas anteriormente permite detectar la similitud entre textos comparando las palabras del mismo, clasificar textos asignando categorías y buscar secuencias de *tokens* basándose en patrones lingüísticos similares a las expresiones regulares [42].

En *spaCy* las anotaciones lingüísticas están disponibles como atributos de *token*, *spaCy* codifica todos los *strings* como valores hash para reducir el uso de memoria y mejorar la eficiencia. Para que el *string* sea legible se le añade al final del nombre una barra baja [43]. Por ejemplo, si se utiliza token.lemma se accede al valor numérico (hash) de la palabra. En el caso de utilizar token.lemma_ se obtendría la forma canónica de la palabra sin codificar.

Por otro lado, hay que tener en cuenta que las anotaciones se rigen por el proyecto *Universal Dependencies* (UD) cuyo objetivo es proporcionar un conjunto universal de categorías y directrices para facilitar la anotación coherente de construcciones similares en todas las lenguas, al tiempo que se permiten extensiones específicas para cada lengua cuando sea necesario [44].

Además, *spaCy* permite realizar entrenamientos estadísticos para mejorar las predicciones. Es un proceso iterativo en el que las predicciones del modelo se comparan con las anotaciones de referencia para estimar el gradiente de pérdida (ver Fig. 33). Este se usa para ver cómo los valores de peso deben modificarse para que las predicciones del modelo se asemejen a las etiquetas de referencia. Es importante entrenar a los modelos con textos que sean significativos para el contexto donde se va a utilizar el modelo [45].

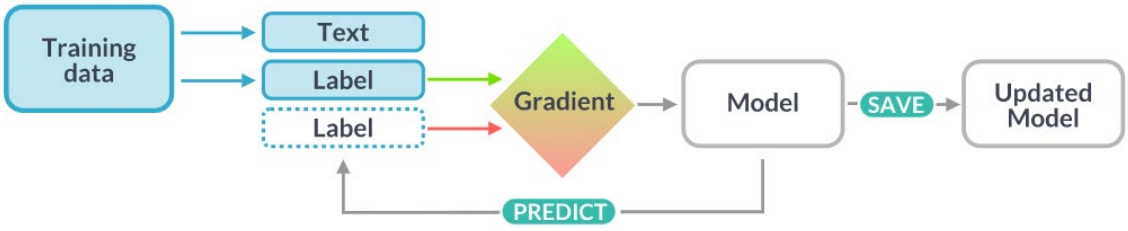

Fig. 33 Entrenamiento en *spaCy*.

spaCy ofrece *pipelines* que ya están entrenadas y permiten a los usuarios acceder a las diferentes funciones que tiene [46]. Para la aplicación de procesamiento de lenguaje natural (NLP) se ha utilizado el modelo en español ya entrenado *es_dep_news_trf* [47] , el procedimiento completo se muestra en la Fig. 34.

Se ha utilizado la clase *Matcher* [48], para el análisis sintáctico, que permite utilizar reglas que describen los atributos de *token*. En particular se han buscado verbos, lugares (nombres), objetos (nombres) y adjetivos usando las anotaciones universales *obl, obj, nmod/amod* para las tres últimas categorías sintácticas tal y como se ve en el Fragmento de código 10.

Una vez obtenido ese conjunto es posible identificar los complementos circunstanciales de lugar y los objetos directos de una oración, para ello se ha desarrollado un nuevo sistema experto (Fragmento de código 11). El funcionamiento es sencillo, en el objeto creado de la clase *Matcher* se almacenan las palabras que coinciden con los patrones de las reglas definidas. Luego es cuestión de recorrer todos los elementos almacenados y compararlos con las categorías que se quieren obtener para guardarlos ya clasificados de forma independiente. En el caso de los verbos, se guarda la forma canónica, es decir, el infinitivo.

Por otro lado, se ha definido un diccionario con los verbos, los lugares y los objetos que realmente el robot es capaz de reconocer. Una vez analizada la frase, se relaciona con acciones predefinidas que dependerán directamente del verbo identificado. En este caso, se han contemplado sólo acciones que impliquen movimiento, pero se podría ampliar para realizar otros tipos de acciones añadiendo al diccionario verbos como "tener" que puede implicar una necesidad "Tener frío", "Tener calor" o "Quiero escuchar música", entre otras.

Fig. 34 Flujo Procesamiento NLP.

```
matcher = Matcher(self.__model.vocab)
pattern_verbos = [{"POS": "VERB"}]
pattern_lugar = [{"DEP": "obl"}]
pattern_lugar_donde = [{"DEP": "nsubj"}]
pattern_objeto = [{"DEP": "obj"}]
pattern_adjetivo = [{"DEP": {"IN" : ["nmod",  "amod"]}}]

matcher.add("Verbos", [pattern_verbos])
matcher.add("Lugar", [pattern_lugar])
matcher.add("Objeto", [pattern_objeto])
matcher.add("Adjetivo", [pattern_adjetivo])
```

Fragmento de código 10 *Matcher Spacy*.

```
for i in range(len(matches)):
    if self.__model.vocab[matches[i].label].text == "Verbos":
        verbos.append(matches[i].lemma_)
    elif self.__model.vocab[matches[i].label].text == "Lugar":
        lugares.append(str(matches[i]))
    elif self.__model.vocab[matches[i].label].text == "Objeto":
        objetos.append(str(matches[i]))
    elif self.__model.vocab[matches[i].label].text == "Adjetivo":
        if self.__model.vocab[matches[i-1].label].text == "Lugar":
            lugares.remove(str(matches[i-1]))
            lugares.append(str(matches[i-1]) + str(matches[i]))
        elif self.__model.vocab[matches[i-1].label].text == "Objeto":
            objetos.remove(str(matches[i-1]))
            objetos.append(str(matches[i-1]) + str(matches[i]))
```

Fragmento de código 11 Sistema experto.

Por último, se obtienen las coordenadas de los lugares y objetos identificados en la oración siempre y cuando estén incluidos en el diccionario. En la Fig. 35 se observa que las localizaciones se definen en hojas Excel que posteriormente se transforman a formato *JSON*.

Hoja cuartobano

objeto	Coordenada_x	Coordenada_y
puerta	3	1
cepillodientes	7	8

Hoja habitacionpadres

objeto	Coordenada_x	Coordenada_y
puerta	2	2
libroazul	5	6

Hoja cocina

objeto	Coordenada_x	Coordenada_y
puerta	5	6
vasoleche	1	1

```
"cuartobano":[
 {
  "objeto": "puerta",
  "Coordenada_x": 3,
  "Coordenada_y": 1
 },
 {
  "objeto": "cepillodientes",
  "Coordenada_x": 7,
  "Coordenada_y": 8
 }
]
```

Fig. 35 Ejemplo registro coordenadas.

3.3.4 Controlador MD49.

El paquete ROS, previamente desarrollado por la directora de este Trabajo Fin de Máster, la Dra. Nieves Pavón, implementa la comunicación con el circuito controlador MD49 e integra un conjunto de módulos software destinados a facilitar la conexión e intercambio de datos con el dispositivo controlador a través del puerto serie. Tales módulos son:

- **serialport.cpp**: Implementa la comunicación con el puerto serie usando termios.h.
- **protocoloMD49TTL.cpp**: Implementa una clase que facilita la construcción de las tramas serie necesarias para realizar acciones sobre el controlador hardware.
- **odometria.cpp**: Proporciona funciones para obtener la pose del robot y aplicar las consignas de velocidad lineal y angular.
- **MD49controller.cpp**: Implementa una clase que facilita el intercambio de información entre la aplicación y el controlador hardware.

Cada uno de estos módulos expone las clases, tipos y funciones necesarias a través del correspondiente archivo **.h**.

Además, se incluye un archivo **definiciones.h** que permite definir los parámetros necesarios para configurar cualquier tipo de robot con tracción diferencial usando el controlador mencionado.

Finalmente, **rosmd49_node.cpp** implementa el programa principal.

Gráficamente el diagrama modular del paquete rosmd49 se muestra en la Fig. 36 .

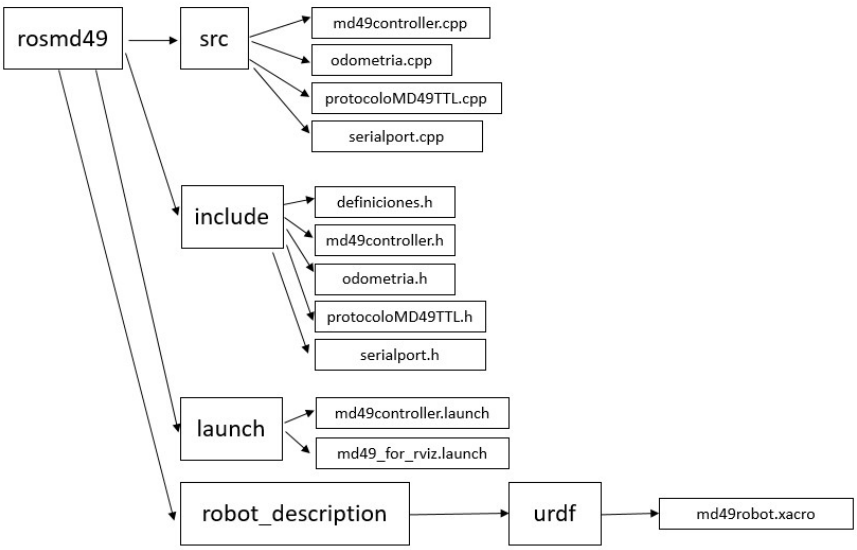

Fig. 36 Diagrama modular del paquete rosmd49.

3.3.5 Componente navegación.

Previamente hay que realizar las transformaciones de coordenadas entre el sensor láser, la cámara RGB-D y la base del robot. La transformación indica las traslaciones y rotaciones necesarias para convertir un sistema de referencia en otro diferente. De ese modo se consigue que todo este referenciado al mismo cuerpo, en este caso, a la base del robot.

ROS proporciona un paquete llamado tf [49] para manejar estas transformaciones. En el Fragmento de código 12 se puede ver como se utiliza y la aplicación del caso utilizado.

```
static_transform_publisher x y z yaw pitch roll frame_id child_frame_id
period_in_ms


<node pkg="tf" type="static_transform_publisher"
name="static_transform_publisher_laser" args="0 0 0.5 0 0 0 base_link
laser 100" />


<node pkg="tf" type="static_transform_publisher"
name="static_transform_publisher_camera" args="0 0 0.9 0 0 0 base_link
camera_link 100" />
```

Fragmento de código 12 Transformaciones coordenadas.

Para la navegación se ha usado el paquete que proporciona ROS (ver Fig. 37), ya utilizado anteriormente en [1]. Tan solo hay que adaptar los parámetros de velocidad, aceleración y odometría al nuevo sistema y renombrar los topics para que coincidan con las publicaciones de los sensores propioceptivos del sistema de tracción visto en el apartado 3.2.2 y los sensores exteroceptivos vistos en los apartados 3.2.3 y 3.2.4.

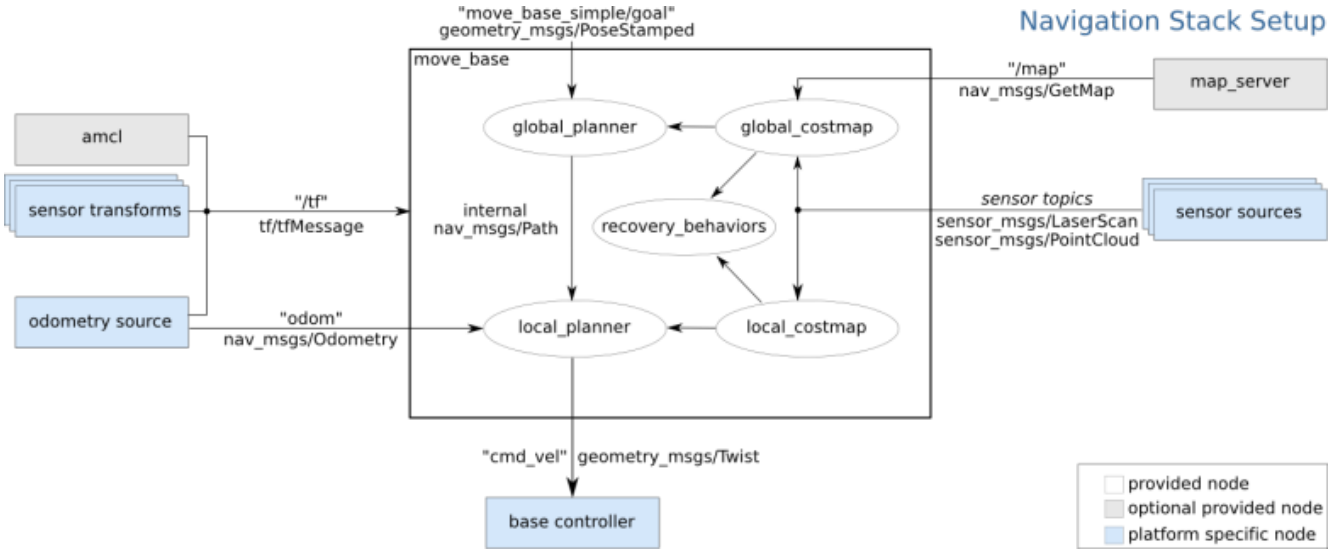

Fig. 37 Paquete navegación ROS.

Por otro lado, al disponer de una cámara RGB-D se ha añadido una nueva funcionalidad a la navegación. Consiste en aprovechar la Nube de puntos que proporciona esta cámara para crear una cuadrícula de ocupación de vóxeles. Para ello se ha utilizado el paquete *spatio_temporal_voxel_layer* [50], [51] que es totalmente integrable en el paquete de navegación.

3.4 Google Cloud Platform.

Google Cloud Platform (GCP), es una plataforma de computación en la Nube que ofrece diversos servicios de *Infrastructure as a Service (IaaS), Platform as a Service (PaaS) y Software as a Service (SaaS)*. Estos servicios ofrecen características de escalabilidad, control automático y gestión de los recursos, herramientas de monitorización y sobre todo la simulación o virtualización de recursos hardware sin la necesidad de adquirirlos. Esta última característica es una de las principales para que este tipo de tecnología haya sido seleccionada. Esto permite que no sean necesarias tareas de mantenimiento del hardware, debido a que no habrá necesidad alguna de adquirir hardware específico para el servidor, que no haya que preocuparse por el consumo de energía, o de otras tareas de mantenimiento. Tampoco será necesario preocuparse por la utilización de recursos, o la necesidad repentina de tener que aumentar la potencia de procesamiento o almacenamiento según la demanda, ya que esta herramienta permite adaptarse con bastante facilidad a estos eventos [52]. Estos servicios se conocen como IaaS.

En cuanto a los servicios de *PaaS,* GCP proporciona herramientas que permiten el desarrollo, despliegue y mantenimiento de aplicaciones que se alojan en la nube, incluso su monitorización por si se requiere aumentar el rendimiento en el caso de que los requisitos incrementen [53]. Además, también ofrece mecanismo sencillos para acceder a los servicios que ya se han desplegado desde múltiples plataformas.

Por último como ya se ha comentado, GCP también proporciona servicios *SaaS* para que el usuario no invierta tiempo en el desarrollo de las aplicaciones [54]. El cliente tan solo tiene que activar las funcionalidades que quiera usar y el propio sistema se encargará de ampliar los recursos en caso de que sea necesario.

Hay que tener en cuenta que la tarifa que se aplica cuando se consume cualquiera de los servicios varía dependiendo de los recursos que se utilicen y de la distribución geográfica. Por ello, es recomendable desactivar aquellas prestaciones que no se vayan a utilizar.

Una buena parte de los recursos que ofrece GCP se explican en los apartados 3.4.1, 3.4.2, 3.4.3 y, en 3.4.4, aquellos utilizados.

3.4.1 Google App Engine.

Es una plataforma como servicio (PaaS) que permite almacenar y desplegar aplicaciones en los servidores propios de Google. Las aplicaciones se pueden implementar mediante los lenguajes de programación Python, Java, Go y PHP. En este caso se ha utilizado Java como lenguaje de programación. Una de las grandes ventajas que proporciona es la de desarrollar aplicaciones que escalan automáticamente, además de numerosos servicios entre los que se encuentra los que se verán en los apartados 3.4.2, 3.4.3 y 3.4.4.

Google App Engine (GAE), cuenta, además, con la posibilidad de utilizar un servidor local que simula el funcionamiento del servidor remoto. Esto permite poder probar las aplicaciones con mayor seguridad ya que, si ocurre cualquier error, basta con restaurar el fichero que almacena la base de datos local y volver a empezar. Además, se evita enviar tráfico al servidor remoto a través de internet.

Por otro lado, existe una consola de administración que permite al desarrollador monitorizar el estado de las aplicaciones de una forma intuitiva, realizar tareas de mantenimiento para corregir posibles errores y gestionar un control de versiones.

3.4.2 Google Cloud Endpoints.

Son un conjunto de herramientas que permiten, de forma sencilla, generar las *APIs* para poder comunicar las aplicaciones clientes (webs y móviles) con una web *BackEnd* [55], [56]. Es decir, permite disponer de unos mecanismos sencillos para poder crear, exponer y consumir la *API REST* con la que se compartirá los datos entre las aplicaciones clientes y el *BackEnd* almacenado y gestionado en el GAE.

3.4.3 Almacenamiento.

Para las aplicaciones que se desarrollan en GAE se recomienda el uso de tres servicios de almacenamiento que no son mutualmente excluyentes:

1. ***Cloud Datastore*** [57]: es un servicio de base de datos *NoSQL* creado para escalar de manera automática, presentar un alto rendimiento y facilitar el desarrollo de aplicaciones. Los datos que se encuentran en el *Datastore* son accedidos mediante las denominadas transacciones, que cumplen las características *ACID* de atomicidad, consistencia, aislamiento y persistencia. Ofrece consistencia fuerte cuando se trata de operaciones que se realizan mediante claves y ancestros, y consistencia eventual en el resto de los casos. Estas transacciones también ofrecen todas las operaciones *CRUD (Create, Read, Update, Delete),* de crear, leer, actualizar y eliminar. *Google Datastore* proporciona numerosos mecanismos para realizar estas transacciones, desde una librería *JSON*, clientes de código abierto o herramientas mantenidas por los usuarios como *NDB* u *Objectify*.

2. ***Cloud SQL*** [58]: es un servicio que facilita la configuración, gestión, almacenamiento y mantenimiento de bases de datos relacionales en la Nube. Se puede usar con *MySQL* o *PostgreSQL*. Este servicio realiza conmutaciones por error y crea copias de seguridad y réplicas, lo que permite que la base de datos sea segura, de alta disponibilidad y flexible para ofrecer un gran rendimiento
3. ***Cloud Storage*** [59]: ofrece un servicio de almacenamiento de objetos de alta durabilidad que se puede escalar y proporciona un acceso instantáneo a los datos. La característica más destacable es la capacidad de almacenar cualquier tipo de archivo, CSV, JSON, AVRO, hasta imágenes en diferentes clases de almacenamiento y adaptarlas según el uso para reducir costes.

El servicio de almacenamiento que se ha utilizado en el desarrollo de la aplicación es *Cloud Datastore* utilizando *Objectify* que es una API para el acceso a datos en Java diseñada para Google App Engine Datastore [60]. Sus objetivos son:

- Curva de aprendizaje sencilla.
- Soporta todas las funciones nativas del *Datastore* de manera intuitiva.
- Modelo sofisticado, fuerte tipificación, estructuras de datos polimórficas.
- Habilita lógica transaccional tipo EJB modular.
- Aumenta el rendimiento y reduce el coste a través de un inteligente uso de la caché en las transacciones.
- Permite realizar migraciones de esquemas al momento sin tener que parar el servidor.
- Puede coexistir con otras herramientas del Datastore: el *API* de Java de bajo nivel, JDO/JPA, *Twig, Go, Python DB y NDB.*

3.4.4 Google Cloud Pub/Sub.

Es un servicio de mensajería para intercambiar datos de eventos entre aplicaciones y servicios. Permite crear sistemas de productores y consumidores de eventos, llamados publicadores y suscriptores. Los publicadores se comunican con los suscriptores de forma asíncrona mediante la transmisión de eventos. *Google Cloud Pub/Sub* ofrece una mensajería duradera y de baja latencia para implementar flujos de trabajo asíncronos, distribuir notificaciones de eventos y transmitir datos desde varios procesos o dispositivos [61]. En general, un mensaje sigue estos pasos:

1. Un publicador transmite un mensaje.
2. El mensaje se almacena para garantizar la entrega
3. El servidor envía una confirmación al publicador de que recibió el mensaje y que garantiza su entrega a todas las suscripciones adjuntas.

4. Al mismo tiempo que escribe el mensaje en el almacenamiento, el servidor lo envía a los suscriptores.
5. Los suscriptores una vez que procesan el mensaje envían una confirmación (*ack*) a *Pub/Sub*.
6. Una vez que al menos un suscriptor por cada suscripción confirmó la recepción del mensaje, el mensaje se borra de forma asíncrona.

3.5 Desarrollo de la aplicación Cloud.

El objetivo es desarrollar una aplicación con una interfaz sencilla y que disponga de una serie de funcionalidades para comunicarse con los servicios que proporciona *Google*. Se ha desarrollado un *BackEnd* y un *FrontEnd*.

El *BackEnd* es el área que se dedica a la parte lógica donde se programan las funciones que tendrá la aplicación que se despliegue en la Nube. Las funciones son *APIs* que el *FrontEnd* consume.

Para desarrollarlo se ha utilizado el entorno de desarrollo Eclipse junto al complemento *Cloud Tools for Eclipse de Google* [62].

El *FrontEnd* es la parte del desarrollo web desde la estructura hasta los estilos. Se suelen utilizar el *lenguaje HTML y CSS* para darle estructura y estilo a la web, y *Javascript* para dar dinamismo y comunicarse con el *BackEnd*.

Para el diseño del *FrontEnd* se ha utilizado *Bootstrap* que es un *framework CSS y Javascript* que proporciona interactividad en una página web, por lo que ofrece una serie de componentes que facilitan la comunicación con el usuario, como menús de navegación, controles de página, barras de progreso u otros elementos [63], [64].

Además, permite la construcción de sitios web responsive, es decir, las páginas están diseñadas para que se visualicen en múltiples dispositivos móviles (ordenadores, móviles, tablets) de manera muy simple y organizada y no se produzcan errores durante su uso.

3.5.1 BackEnd.

Una vez finalizada la instalación de Google Cloud Tools en Eclipse [62], ya se puede iniciar el desarrollo de la aplicación. Para ello, para no partir de cero es recomendable crear una Aplicación estándar de *App Engine* y crearla como proyecto *Maven*. En este paso se pueden incluir ya las librerías que se van a utilizar, en este caso *App Engine* , *Google Cloud Endpoint y Objectify*. Este proceso se puede ver en la Fig. 38.

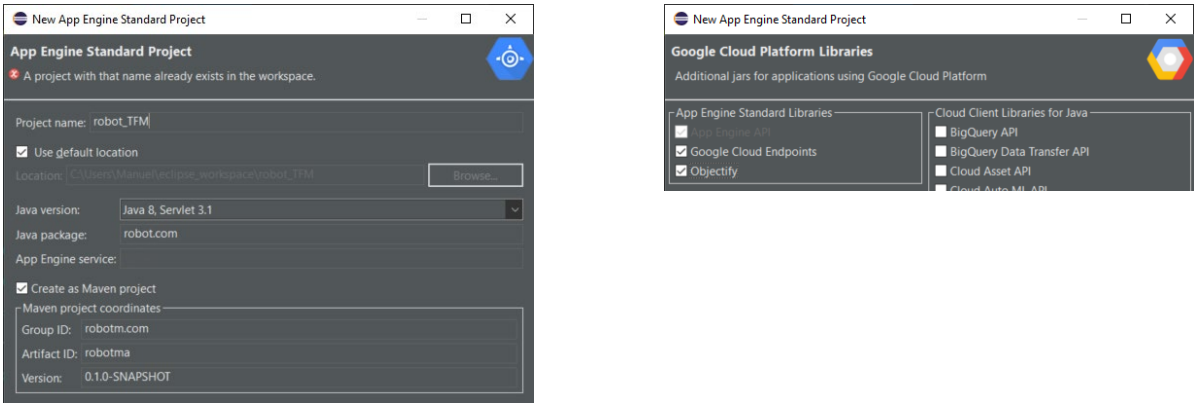

Fig. 38 Creación Aplicación estándar App Engine.

Una vez inicializada se creara una estructura de carpetas, de las cuales nos centraremos en el árbol que se muestra en la Fig. 39.

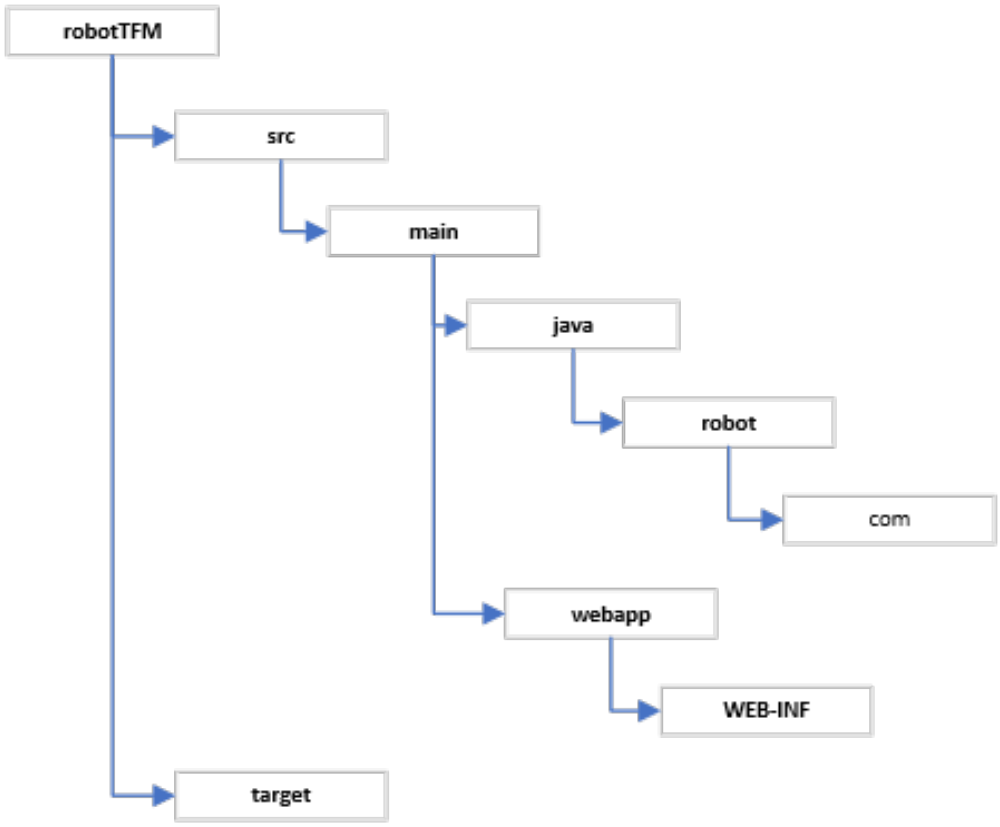

Fig. 39 Estructura carpetas aplicación App Engine.

Dentro de la carpeta *WEB-INF* hay un archivo denominado web.xml que hay que modificar para especificar los componentes que se van a utilizar, en este caso un *EndpointServlet* (ver Fragmento de código 13) para que la aplicación pueda recibir peticiones y procesarlas utilizando los métodos que se creen en el *Endpoint*..

```
<servlet>
    <servlet-name>EndpointsServlet</servlet-name>
    <servlet-class>com.google.api.server.spi.EndpointsServlet</servlet-
class>
    <init-param>
        <param-name>services</param-name>
        <param-value>robot.com.Endpoint_robot</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>EndpointsServlet</servlet-name>
    <url-pattern>/_ah/api/*</url-pattern>
</servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Fragmento de código 13 EndpointServlet web.xml.

En la carpeta *robot.com* hay que crear el *Endpoint* que se va a utilizar y las diferentes clases. Por un lado hay que crear una clase Modelo, que será una instancia *Objectify* que se guardará en la base de datos, luego habrá que utilizar la anotación correspondiente. La clase con sus atributos se pueden ver en el Fragmento de código 14. Los índices tienen que ser unívocos para diferenciar los datos que se guarden en el *Datastore*.

```
@Entity
public class Modelo {
     @Id Long id;
     @Index String familia;
     @Index String persona;
     @Index String tipo;
     @Unindex String modelo;
```

Fragmento de código 14 Clase Modelo.

Del mismo modo hay que crear una clase Mapa como se puede ver en el Fragmento de código 15. En el atributo nombre se indicará tanto el tamaño del plano como la escala. El atributo plano es de tipo String porque la imagen está codificada en base64.

```
@Entity
public class Mapa {
	@Id Long id;
	@Index String nombre;
	@Unindex String plano;
	@Unindex String coordenadas;
```

Fragmento de código 15 Clase Mapa.

Por otro lado, hay que crear las clases que contengan los métodos que indican la respuesta al almacenar, descargar y listar objetos de tipo Modelo y de tipo Mapa, incluyen un atributo *res* para poder obtener una trazabilidad en el caso de que se produzca un error. A continuación, se crea el *Endpoint* que proporcionará los servicios a los que se pueden acceder desde *http*. Para ello hay que seguir la anotación que se puede ver en el Fragmento de código 16. En este caso se ha indicado que se va a utilizar el método *http* de tipo *Post* y sus argumentos. Los métodos que se han desarrollado se utilizan para guardar, listar y descargar entidades del *Datastore*.

```
@Api(name = "endpoint_robot",
version = "v1",
namespace = @ApiNamespace(ownerDomain = "robot.com",
                                ownerName = "robot.com",
                                packagePath = ""
                                )
)
public class Endpoint_robot {


//ModelosIA
	@ApiMethod(name = "almacenar_modelo", httpMethod =
ApiMethod.HttpMethod.POST, path = "url_almacenar_modelo")
	public Respuesta_almacenar almacenar_modelo(@Named ("familia")
String familia, @Named("persona") String persona, @Named("tipo") String
tipo, @Named("modelo") String modelo) {
```

Fragmento de código 16 Clase Endpoint.

Como se van a realizar transacciones con el *Datastore* hay que seguir utilizando la anotación *Objectify*. Las instrucciones más importantes utilizadas se pueden ver en el Fragmento de código 17.

```
md.crear_id(new Timestamp(System.currentTimeMillis()).getTime());


Key<Modelo> clave = ofy().save().entity(md).now();


List<Modelo> lofy = ofy().load().type(Modelo.class).filter("familia ==",
familia).filter("tipo ==", tipo).list();


Modelo ofy = ofy().load().type(Modelo.class).filter("familia ==",
familia).filter("persona ==", persona).filter("tipo ==",
tipo).first().now();
```

Fragmento de código 17 Transacciones Datastore con objectify.

La primera se ha utilizado para crear ese indicador unívoco mencionado anteriormente, para que no se repita se ha usado la marca temporal (*timestamp*). El segundo permite guardar una entidad en el *Datastore*, este a su vez devuelve una clave para comprobar si la transacción se ha realizado correctamente. El tercero y el cuarto permiten recuperar una entidad o una lista de ellas según una serie de condiciones. En todas estas transacciones, en caso de producirse un error, se asigna un número al atributo *res* de las clases respuestas mencionado anteriormente, de este modo se puede identificar el fallo.

Ahora bien para que se puedan utilizar las entidades *POJO* que son las clases que se han utilizado con la anotación *Objectify* hay que registrarlas en *ObjectifyWebListener.java* que también está en la carpeta *robot.com* de la forma que indica en el Fragmento de código 18.

```
  @Override
  public void contextInitialized(ServletContextEvent event) {
    ObjectifyService.init();
    // This is a good place to register your POJO entity classes.
    ObjectifyService.register(Modelo.class);
    ObjectifyService.register(Mapa.class);
  }
```

Fragmento de código 18 Registrar entidades POJO.

En este punto ya se puede compilar la aplicación. Para ello se realiza un *Maven Build* usando la meta *compile* y posteriormente probar la aplicación en modo local (localhost) como se verá en el apartado 4.1.4.

Para desplegar la aplicación en Google en primer lugar hay que crear un nuevo proyecto desde *Google Cloud* tal y como muestra la Fig. 40, asignándole una cuenta de facturación para poder utilizar los servicios de la Nube aunque el gasto sea nulo. Además, hay que activar los servicios que va utilizar la aplicación, en este caso *App Engine, Endpoints, Datastore y Pub/Sub.*

Fig. 40 Creación proyecto Google Cloud

Una vez creado hay que asegurarse de que el proyecto está seleccionado, para ello desde una terminal de Windows lanzamos la siguiente instrucción: *gcloud config set project tfmrobot.* Además, hay que iniciar con la cuenta de Google en el entorno de desarrollo de Eclipse y seleccionar el proyecto, se creará una variable de entorno automáticamente como muestra la Fig. 41.

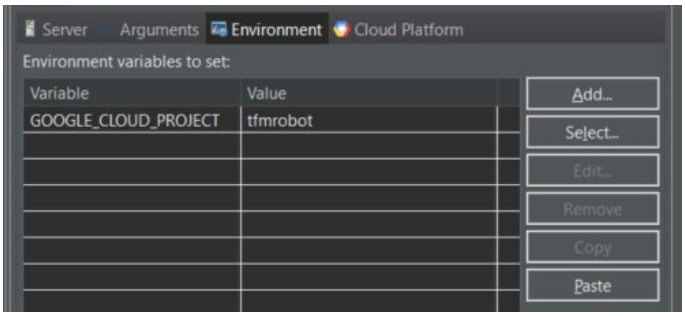

Fig. 41 Configuración proyecto Google Cloud en Eclipse.

Finalmente, ya se puede desplegar la aplicación utilizando *Deploy App Engine Standard.* Cuando se despliega se crea automáticamente un espacio de *Cloud Storage* donde se almacenan todos los ficheros de la aplicación.

3.5.2 FrontEnd.

Para diseñar la página web hay que modificar el archivo *index.html* que se Encuentra en la carpeta *webapp* (ver Fig. 39). En la cabecera hay que añadir los recursos que proporciona *Bootstrap*, el diseño responsive (ver Fragmento de código 19) y los *scripts* que se van a utilizar.

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css
">
    <script
src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.slim.min.js"></scr
ipt>
    <!-- Popper JS -->
    <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></
script>
    <!-- Latest compiled JavaScript -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.mi
n.js"></script>
```

Fragmento de código 19 Recursos Bootstrap 5 y diseño responsive.

El contenido de la página web variará según el rol de la persona que se autentifique. Para ello se ha utilizado Google *Sign-In* que administra el flujo de *OAuth 2.0* y el ciclo de vida del *token*, lo que simplifica su integración con las *APIs* de *Google*. Cualquier aplicación que use *OAuth 2.0* para acceder a las *APIs* de Google debe tener credenciales de autorización que identifiquen la aplicación en el servidor *OAuth 2.0* [65]. Para ello hay que ir a la consola de Google y crear las credenciales de ID de cliente OAuth dentro de API y servicios como muestra la Fig. 42.

Fig. 42 Creación credenciales Oauth

Ahora en la cabecera hay que incluir la biblioteca de la plataforma de Google de inicio de sesión, y el ID que se ha creado anteriormente como muestra el Fragmento de código 20. Además, es necesario añadir el botón de inicio, en este caso se ha utilizado la configuración predeterminada de Google.

```
<script src="https://apis.google.com/js/platform.js" async
defer></script>
<meta name="google-signin-client_id"
content="YOUR_CLIENT_ID.apps.googleusercontent.com">
<div class="g-signin2" data-onsuccess="onSignIn"></div>
```

Fragmento de código 20 Biblioteca inicio sesión Google desatendido.

Una vez que se inicie sesión con un usuario de Google se puede obtener la información ligada a dicho usuario y el *token* para poder usar los servicios del *BackEnd* (ver Fragmento de código 21). Está información es la que se usa para mostrar un contenido u otro de la página web.

```
<script>
      function onSignIn(googleUser) {
        // Useful data for your client-side scripts:
        var profile = googleUser.getBasicProfile();
        var id_token = googleUser.getAuthResponse().id_token;
      }
</script>
```

Fragmento de código 21 Información inicio de sesión.

Por otro lado, se puede agregar un botón de cierre de sesión de Google tal y como muestra Fragmento de código 22.

```
<button type="button" class="btn btn-danger" onclick="signOut();">Sign
Out</button>


function signOut() {
    var auth2 = gapi.auth2.getAuthInstance();
    auth2.signOut().then(function () {
      console.log('User signed out.');
    });
}
```

Fragmento de código 22 Cierre de sesión Google.

El problema con este método es que ha quedado descontinuada la biblioteca de la plataforma de JavaScript de inicio de sesión de Google para la web. Por ello se ha migrado a la biblioteca de Servicios de Identidad de Google [66], [67]. Para ello se ha utilizado el Fragmento de código 23.

```
<script src="https://accounts.google.com/gsi/client" async
defer></script>


<div id="g_id_onload"
         data-client_id=" YOUR_CLIENT_ID.apps.googleusercontent.com "
         data-context="signin"
         data-ux_mode="popup"
         data-callback="handleCredentialResponse">
    </div>
    <div class="g_id_signin" data-type="standard"></div>


function handleCredentialResponse(response) {
    responsePayload = decodeJwtResponse(response.credential);
}
```

Fragmento de código 23 Inicio sesión en Google.

Por otro lado, para poder usar la funcionalidad multihilo, se ha utilizado el concepto de *worker*. Las tareas se comunican entre sí mediante mensajes. Cuando una tarea hace un *Post* de un mensaje se comunica con el hilo principal donde se creó el *worker* y ese hilo es el que procesa que le ha llegado un mensaje y le asigna una función. Para crear los *workers* se ha utilizado el Fragmento de código 24, reutilizando el código para realizar una petición con JavaScript – jQuery que proporciona Postman como se verá en el apartado 4.1.4, y se ha incluido el *script* en la cabecera como *Worker_xhr.js*.

```
onmessage = function (e) {
    var worker_response = {
        response: null,
        id_worker: e.data.id_worker,
    };
    try {
        var xhr = new XMLHttpRequest();
        xhr.withCredentials = e.data.data_withCredentials;
        xhr.open(e.data.data_method, e.data.data_url, false);

        xhr.setRequestHeader("Content-Type", "application/json");
        xhr.send(e.data.data_body);
        switch (xhr.status) {
            case 200:
                        worker_response.response = xhr.response;
                        break;
            default:
                        worker_response.response = "Fallo cobsulta";
        }
    } catch(e) {
        console.log(e);
        worker_response.response = e;
    }
    this.postMessage(worker_response);
    this.self.close();
}
```

Fragmento de código 24 Creación workers.

Por último, ya sólo queda desarrollar todas las funcionalidades y el diseño de la propia página web. Para el diseño se han utilizado los componentes de *Bootstrap*; las funciones se han realizado en el script *peticionNube.js* que se ha incluido en la cabecera. A continuación, se describen los diferentes componentes de la página web y su funcionalidad:

Registrar modelo

Esta función permite guardar el modelo de IA en el *Datastore*. Para ello hay que realizar una petición al método de almacenamiento implementado en el *BackEnd*. El funcionamiento es sencillo, tan sólo hay que hacer una petición *Post* al método correspondiente del *Endpoint* rellenando el cuerpo del mensaje con las variables obtenidas de los elementos del diseño web. El Fragmento de código 25 muestra de forma resumida la implementación.

```
var reader = new FileReader();
        reader.onload = function (evt) {
            modelo = evt.target.result;
            w = crear_worker();
            var r = {
                id_worker: 1,
                data_method: metodo,
                data_withCredentials: true,
                data_url: url+servicio,
                data_body: JSON.stringify(datos_body)
            }
            w.postMessage(r);
            reader.readAsText(objmodelo, "UTF-8");
```

Fragmento de código 25 Función registrar modelo.

El diseño gráfico de este módulo se puede ver en la Fig. 43 y la implementación del mismo en el Fragmento de código 26. Los componentes del diseño web son campos para introducir texto, lista desplegable, selector para subir un archivo y un botón para lanzar la petición asociada. Estos elementos van a ser comunes para al resto de módulos del diseño web.

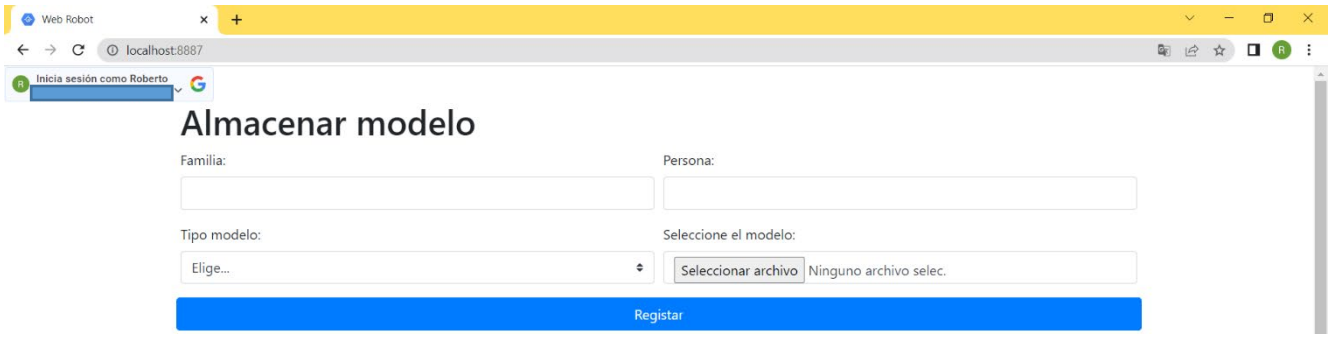

Fig. 43 Diseño módulo almacenamiento modelo.

```
<div class="form-group col-md-6">
    <label for="caja1">Familia:</label>
    <input type="text" class="form-control" id="caja1">
</div>
<div class="form-group col">
    <label class="mr-sm-2" for="caja3">Tipo modelo:</label>
    <select class="custom-select mr-sm-2" id="caja3">
    <option selected>Elige...</option>
    <option value="1">XGB</option>
    <option value="2">DNN</option>
    <option value="3">SVM</option>
    <option value="4">KNN</option>
    </select>
</div>
<div class="form-group col">
    <label class="mr-sm-2" for="customFile">Seleccione el modelo:</label>
    <input type="file" class="form-control" id="customFile">
</div>
<div class="form-row">
    <button type="button" class="btn btn-primary btn-block"
onclick="registrar_modelos()">Registar</button>
</div>
```

Fragmento de código 26 Componentes utilizados Html y Bootstrap 5.

Listar Modelos

Esta función recupera los modelos en el *Datastore* filtrando la consulta por los atributos familia y tipo. Para ello hay que realizar una petición al método listar implementado en el *BackEnd*. El funcionamiento es similar a la función anterior. Para rellenar la tabla se ha desarrollado la función del Fragmento de código 27 que consiste en rellenar las celdas de la tabla utilizando los atributos de los objetos de la lista de modelos cuando se recibe la respuesta de la petición a través de *w.onmessage = function(respuesta).*

```
function rellenartabla(datos){
    console.log(datos);
    if (datos.l) {
        console.log(datos.l);
        if (datos.l.length > 0) {
            var tabla = document.getElementById("tablamodelos");
            for (var i = 0; i < datos.l.length; i++) {
                var row = tabla.insertRow(i+1);
                var c1 = row.insertCell(0);
                var c2 = row.insertCell(1);
                var c3 = row.insertCell(2);
                c1.innerHTML = datos.l[i].familia;
                c2.innerHTML = datos.l[i].persona;
                c3.innerHTML = datos.l[i].tipo;
            } } } }
```

Fragmento de código 27 Rellenar tabla.

El aspecto gráfico de este módulo se puede ver en la Fig. 44.

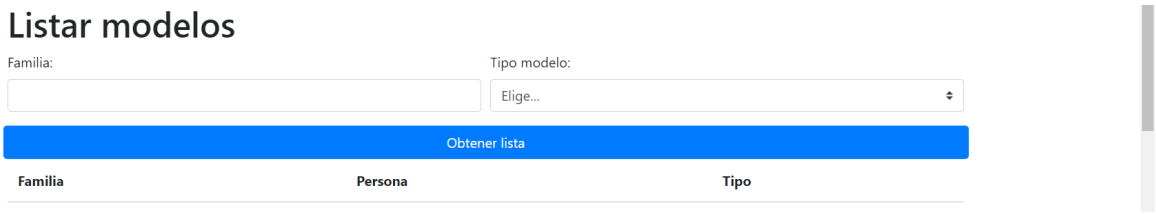

Fig. 44 Diseño módulo listado de modelos.

Descargar Modelo

Esta función permite descargar un modelo filtrando según los campos rellenados. El funcionamiento es similar al anterior pero en este caso se ha utilizado el script FileSaver.js del repositorio github [68]. El diseño gráfico de este módulo se puede ver en la Fig. 45.

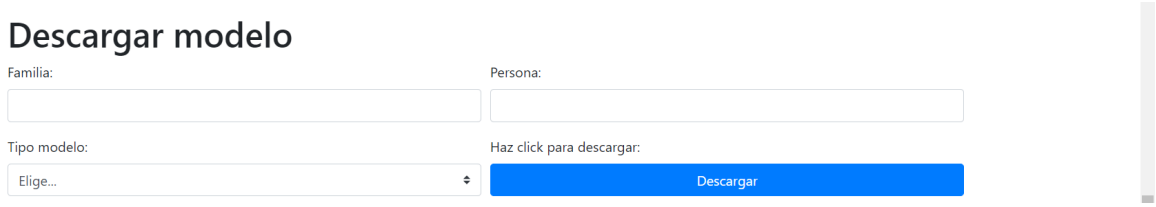

Fig. 45 Diseño módulo descarga modelo.

Publicar modelo

Esta función tiene como objetivo publicar un mensaje para disparar la función de descargar el modelo en el robot cuando el suscriptor reciba ese mensaje. En primer lugar hay que crear un tema desde *Pub/Sub* de la consola de Google tal y como muestra la Fig. 46.

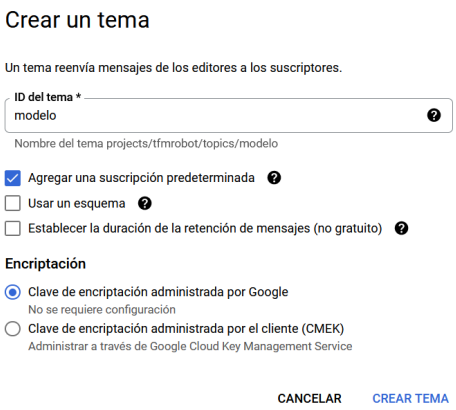

Fig. 46 Creación tema Pub/Sub.

Para poder utilizar el publicador de Google se ha utilizado la biblioteca de la plataforma de Google de inicio de sesión porque hay que estar autenticado. Para publicar hay que realizar una petición de tipo *Post* a la *url* de la *API Pub/Sub* con el ID del proyecto y el tema creado anteriormente. En el cuerpo de la petición hay que codificar el mensaje en *base64* para que no se produzca un error y opcionalmente se pueden añadir atributos. El procedimiento se puede ver de forma resumida en el Fragmento de código 28.

```
const attrs = {
      'sent_by': 'client'
    };
let body = { 'messages': [] }; {
            console.log('publishing', messages);
            body['messages'].push({
            'data': btoa(messages),
            'attributes': attrs
            });
        }
return await gapi.client.request({
            'path':'https://pubsub.googleapis.com/v1/projects/' +
                   PROJECT_ID + '/topics/' + topic + ':publish',
            'method':'POST',
            'headers': {},
            'body': body,
        });
        }
```

Fragmento de código 28 Publicador desatendido.

Sin embargo, este método ya no funciona por el mismo motivo que sucedía en la autentificación, la biblioteca ha quedado obsoleta. Como alternativa, se ha proporcionado permisos de publicación al usuario autenticado tal y como muestra la Fig. 47. De este modo se puede utilizar este servicio directamente con una petición Post usando JavaScript – jQuery.

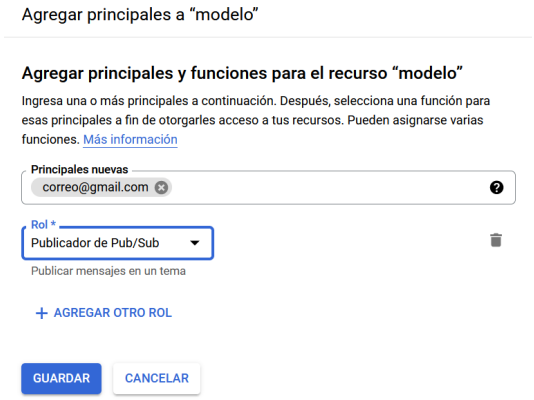

Fig. 47 Permisos de publicación.

El diseño gráfico se puede ver en la Fig. 48.

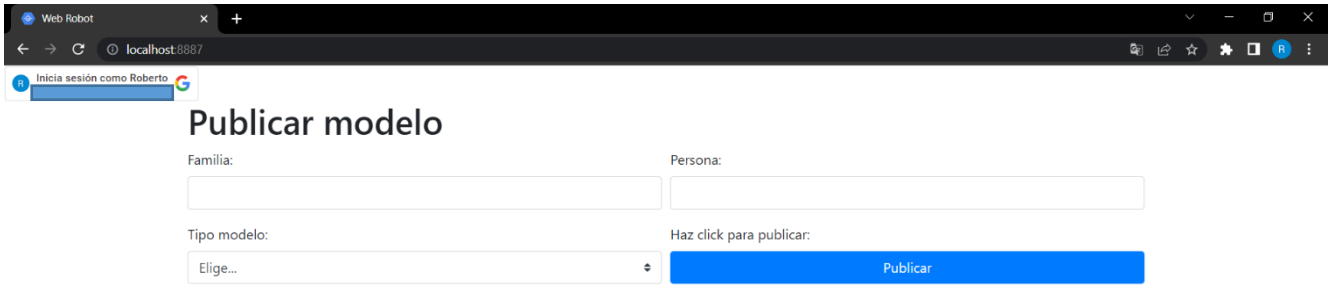

Fig. 48 Diseño módulo publicación de modelo.

Registrar mapa

Esta función permite guardar el mapa en el *Datastore*. Su funcionamiento es igual a la función registrar modelo, tan solo cambia la dirección de la petición *Post* y el cuerpo del mensaje. El aspecto gráfico se puede ver en la Fig. 49.

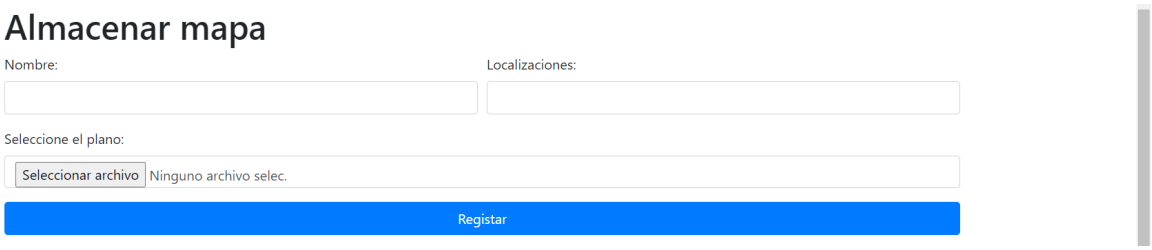

Fig. 49 Diseño módulo almacenamiento mapa.

Listar Mapas

Esta función es similar a la de "listar modelos" pero filtrando la consulta por el atributo nombre. El diseño gráfico se puede ver en la Fig. 50.

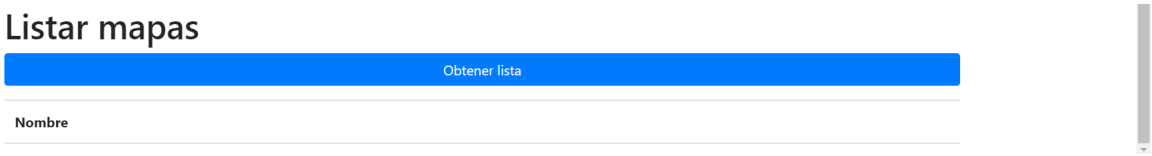

Fig. 50 Diseño módulo listado de mapas.

Publicar mapa

Esta función es similar a la de publicar modelo, tan solo cambiar el tema de publicación y el mensaje. Es aspecto gráfico se puede ver en la Fig. 51

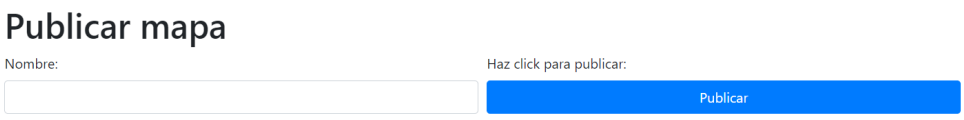

Fig. 51 Diseño módulo publicación de mapa.

Por último, se van describir las aplicaciones de los suscriptores que hay en el robot para descargar el modelo de Inteligencia Artificial y el mapa cuando se realiza una publicación. Se va a utilizar la librería de *Python* que proporciona Google [69]. Para poder usarla hay que crear una Clave *API* desde la consola tal y como muestra la Fig. 52 y usarla de variable de entorno en el dispositivo donde se quieran utilizar.

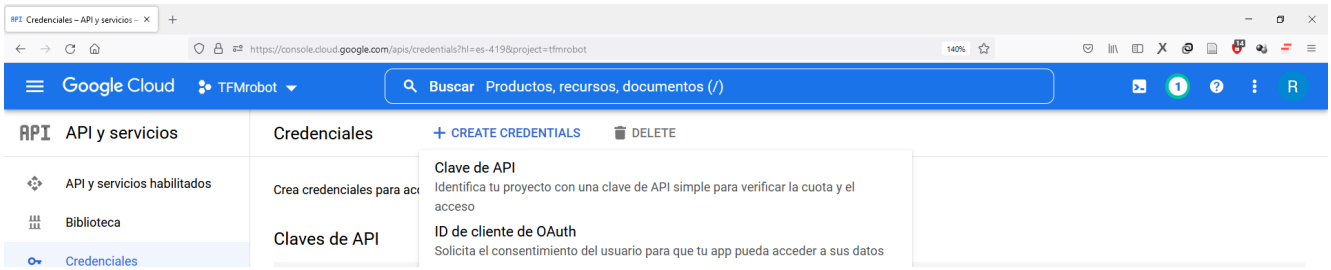

Fig. 52 Creación clave API.

El modelo de suscripción utilizado es la extracción asíncrona porque proporciona una mayor capacidad de procesamiento, no se bloquean los mensajes nuevos y permite la confirmación de un mensaje a la vez. A continuación, se describen las dos aplicaciones desarrolladas:

Suscriptor modelo

Esta aplicación está suscrita al tema donde se realizan las publicaciones del modelo tal y como muestra el Fragmento de código 29. En el momento que aparece un nuevo mensaje en el tema, el suscriptor lo procesa y envía una confirmación de recepción *ack* (ver Fragmento de código 30). En este caso el procesamiento consiste en decodificar el mensaje cuya información permite realizar una petición *Post* al servicio que se creó en el *EndPoint* para descargar modelos del *Datastore*.

```
def suscribe():
    streaming_pull_future = subscriber.subscribe(subscription_path,
callback=callback)
    with subscriber:
        try:
            streaming_pull_future.result()
        except TimeoutError:
            streaming_pull_future.cancel()
            streaming_pull_future.result()
```

Fragmento de código 29 Suscripción.

```
def callback(message: pubsub_v1.subscriber.message.Message) -> None:
    topic_respuesta = message.data.decode('ascii')
    message.ack()
    body = topic_respuesta.split("_")
    if (body[0] != None) and (body[1] != None) and (body[2] != None):
        save_model(body[0], body[1], body[2])
    else:
        print("Esta incompleta la respuesta")
```

Fragmento de código 30 Procesamiento mensaje suscripción.

Suscriptor mapa

Esta aplicación está suscrita al tema donde se realizan las publicaciones del mapa. Es funcionamiento es similar al anterior, tan solo cambia el post procesamiento del mensaje. Después de realiza la petición para obtener la instancia mapa del *Datastore* filtrado según la información del mensaje se realizan dos funciones. La primera es muy sencilla, tan solo consiste en guardar en un fichero JSON el atributo coordenadas del objeto mapa solicitado.

La segunda transforma el mapa almacenado para que pueda ser utilizado para la navegación en *ROS*. La navegación utiliza el nodo *map_server* [70] y necesitas un archivo *YAML* con los siguientes campos:

- *image*: ruta del archivo que contiene la imagen del mapa
- *resolution*: resolución del mapa en metros/píxel
- *origin*: coordenadas del píxel inferior del mapa (x, y, *yaw*)
- occupied_*thress*: píxeles con una probabilidad de ocupación mayor que este umbral se consideran completamente ocupados.
- *free*_tresh: píxeles con una probabilidad de ocupación inferior a este umbral se consideran completamente libres
- *negate*: para invertir los umbrales de ocupación.

El Fragmento de código 31 muestra la implementación de la función para transformar el mapa y obtener el archivo YAML.

```
image = readb64(img_data)
altura_px, anchura_px = image.shape
scale = float(parametros[1].split('/')[0])/float(parametros[1].split('/')[1])
altura_m = float(parametros[2])/1000
anchura_m = float(parametros[3])/1000
resolucion = 0.05

sx = anchura_m / (scale*anchura_px*resolucion)
sy = altura_m / (scale*altura_px*resolucion)

res = cv2.resize(image, None, fx=sx, fy=sy , interpolation = cv2.INTER_CUBIC)
mapLocation  = '/home/roberto/catkin_ws/src/robotjuno_navigation/maps/'
mapName = parametros[0] + '_plano'

cv2.imwrite(mapLocation + mapName +  '.pgm', res)
with open(mapLocation + mapName + '.yaml', "w") as f:
    f.write("image: " + mapLocation + mapName + ".pgm\n")
    f.write("resolution: " + str(resolucion) + "\n")
    f.write("origin: [" + str(-1) + "," +  str(-1) + ", 0.000000]\n")
    f.write("negate: 0\n")
    f.write("occupied_thresh: 0.65\n")
    f.write("free_thresh: 0.196")
```

Fragmento de código 31 Procesamiento mapa para uso en ROS.

3.6 Integración de todos los componentes.

Por un lado, se dispone de una aplicación que proporciona una serie de funcionalidades para alojar, listar y descargas las diferentes configuraciones (modelos y mapas) del robot. Para desarrolla esta aplicación se ha utilizado *AppEngine*, *Endpoints* y *Datastore* tal y como se ha visto en el apartado 3.5.1.

También, se ha diseñado una página web para proporcionar una interfaz amigable a la hora de consumir dichas funcionalidades. Se ha tenido en cuenta que el contenido de la web variará según el rol de usuario que acceda. Además, se ha añadido el servicio Pub/Sub de Google para poder descargar de forma remota las configuraciones en el robot siempre y cuando tenga las credenciales necesarias. Todo este desarrollo se puede consultar en el apartado 3.5.2.

Además, se ha partido de la integración mediante ROS realizada en [1]. En el apartado 3.3.1 se ha desarrollado una solución de reconocimiento facial más sólida, por tanto hay que modificar el nodo servidor de reconocimiento facial que se desarrolló. La nueva solución consiste en un nodo servidor que obtiene el tensor de características de 512 que proporciona el modelo FaceNet de la persona detectada y un nodo servidor que realiza la inferencia a partir de dicho tensor para identificar a la persona. En cuanto al nodo servidor de voz a texto y el nodo de procesamiento natural el cambio es sencillo, tan solo hay que intercambiar las librerías utilizadas en dichos nodos por las desarrolladas en los apartados 3.3.2 y 3.3.3.

Por último, se ha utilizado el paquete de navegación configurado en el apartado 3.3.5.

La arquitectura software final del sistema se puede ver en la Fig. 53 y el funcionamiento del sistema dentro del ecosistema de ROS se puede ver en la Fig. 54.

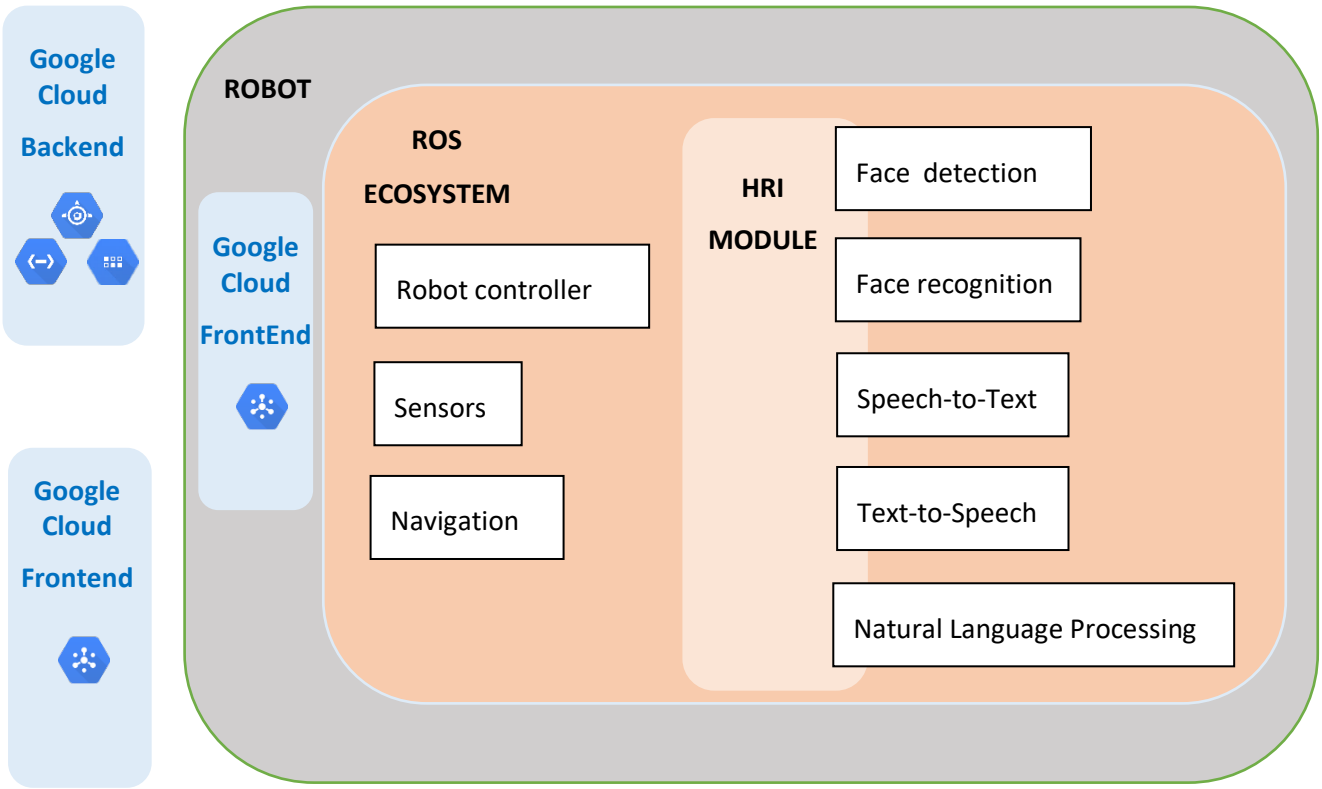

Fig. 53 Arquitectura software del sistema.

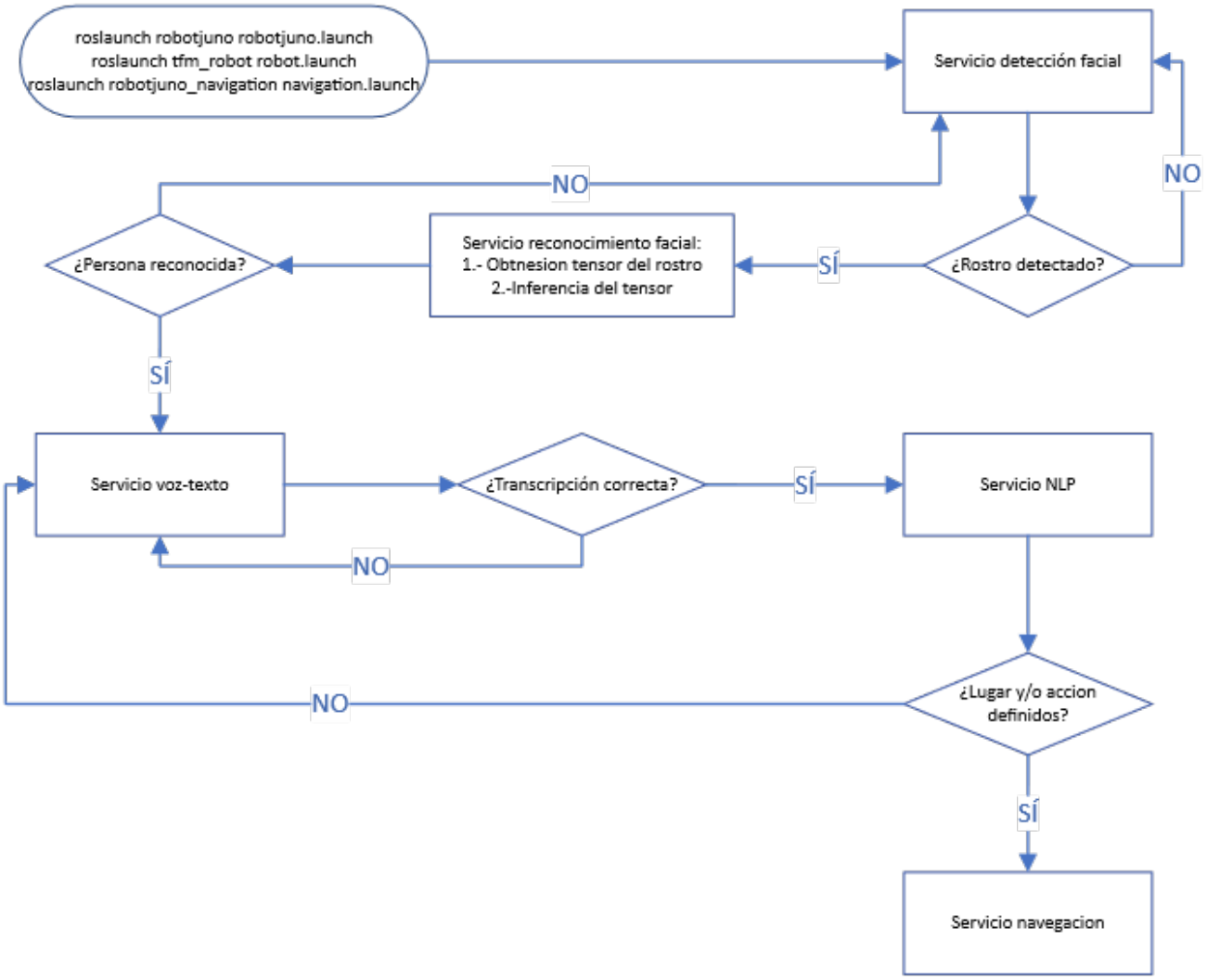

Fig. 54 Flujograma funcionamiento sistema en ROS.

Al principio, se intentó desplegar todo el sistema software sobre un único dispositivo, una Nvidia Jetson. Sin embargo, debido a que no ha podido adquirirse por causas de fuerza mayor (crisis de suministrors), se ha utilizado el ordenador personal cuyas características se pueden ver en la Tabla 5.

Tabla 5 Características ordenador personal.

Model No.	AORUS 15G
Procesador	Intel Core i7 10875H 2.30GHz
Memoria RAM	32.00 GB DDR4 2666MHz
Tarjeta Gráfica	NVIDIA RTX 2070 Super (No se ha podido utilizar en el SO)
Cámara Web	Logitech Carl Zeiss Tessar

Sistema Operativo	Ubuntu 18.04

ROS está diseñado pensando en la computación distribuida luego es posible utilizar varios dispositivos siempre y cuando se cumpla que solo un dispositivo actúe como maestro y que todos los dispositivos estén configurados para utilizarlo [71]. Para cumplir esta configuración hay que definir las variables de entorno tal cual indica la Fig. 55.

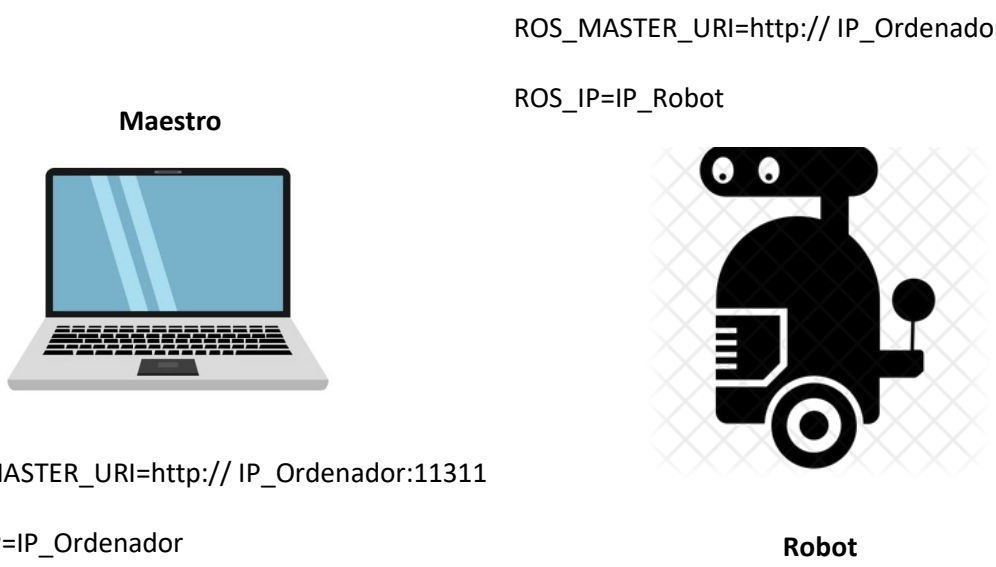

Fig. 55 Computación distribuida ROS.

En el ordenador se han desplegado los módulos de interacción humano robot y el paquete de navegación y en el robot se ha lanzado el controlador MD49, el paquete del sensor láser Hokuyo y el paquete de la cámara RGB-D Orbbec.

4 Análisis de resultados.

En el desarrollo de este capítulo se mostrará en qué consisten las pruebas que se han utilizado y los resultados que se han obtenido.

4.1 Pruebas diseñadas y resultados.

Para realizar las pruebas se ha seguido utilizando el entorno virtual para *Python* creado en [1] ya que permite instalar librerías en dicho entorno sin inferir en el sistema operativo principal. De esta forma, se pueden realizar los ensayos y una vez que se han obtenido buenos resultados, proceder con la integración en el nuevo sistema desarrollado.

En esta ocasión para el entrenamiento de los modelos se ha utilizado *Visual Studio Code* [72], con la extensión *Jupyter* [73] y así poder trabajar con el *Dataset* de forma local.

4.1.1 Pruebas modelos reconocimiento facial y resultados.

A continuación se van a obtener las métricas de los modelos de reconocimiento facial descrito en el apartado 0:

DNN

Se ha variado la estructura de la red y el optimizador. La precisión de los modelos se pueden ver en la Tabla 6.

Tabla 6 Precisión modelos red neuronal.

Roberto			
Red 5-3-1		Red 10-5-1	
Adam	SGD	Adam	SGD
99,15%	97,45%	99,15%	97,45%

Nieves			
Red 5-3-1		Red 10-5-1	
Adam	SGD	Adam	SGD
98,27%	98,27%	98,27%	98,27%

Se observa que para ambos casos la estructura de la red neuronal y la elección del optimizador no llegan a ser influyentes.

KNN

Se ha variado el parámetro k. Las diferentes métricas se pueden ver en la Tabla 7, Tabla 8, Tabla 9 y Tabla 10 .

Tabla 7 Roberto K = 2.

36	0

precision	recall	f1-score	support	accuracy

3	79

	precision	recall	f1-score	support	accuracy
0	0,9231	1	0,96	36	0,9746
1	1	0,9634	0,9814	82	

Tabla 8 Roberto K =4.

36	0
3	79

	precision	recall	f1-score	support	accuracy
0	0,9231	1	0,96	36	0,9746
1	1	0,9634	0,9814	82	

Tabla 9 Nieves K =2.

31	0
3	82

	precision	recall	f1-score	support	accuracy
0	0,9118	1	0,9538	31	0,9741
1	1	0,9647	0,982	85	

Tabla 10 Nieves K = 4.

31	0
3	82

	precision	recall	f1-score	support	accuracy
0	0,9118	1	0,9538	31	0,9741
1	1	0,9647	0,982	85	

Se llega a la conclusión que variar vecindad de 2 a 4 no afecta al entrenamiento del modelo.

SVM

Se ha variado el *kernel*. Las diferentes métricas se pueden ver en la Tabla 11, Tabla 12, Tabla 13 y Tabla 14.

Tabla 11 Roberto *kernel* lineal.

36	0
0	82

	precision	recall	f1-score	support	accuracy
0	1	1	1	36	1
1	1	1	1	82	

Tabla 12 Roberto *kernel* polinomial.

36	0
0	82

	precision	recall	f1-score	support	accuracy
0	1	1	1	36	1
1	1	1	1	82	

Tabla 13 Nieves kernel lineal.

31	0

	precision	recall	f1-score	support	accuracy

0	85

0	1	1	1	31	0,9741
1	1	1	1	85	

Tabla 14 Nieves *kernel* polinomial.

31	0
0	85

	precision	recall	f1-score	support	accuracy
0	1	1	1	31	0,9741
1	1	1	1	85	

En este caso la elección del *kernel* tampoco afecta al entrenamiento del modelo

XGBoost

Se ha utilizado un modelo con los parámetros estándar y otro con los parámetros optimizados. Las métricas se pueden ver en la Tabla 15, Tabla 16, Tabla 17 y Tabla 18.

Tabla 15 Roberto parámetros estándar.

36	0
0	82

	precision	recall	f1-score	support	accuracy
0	1	1	1	36	1
1	1	1	1	82	

Tabla 16 Roberto parámetros optimizados.

36	0
0	82

	precision	recall	f1-score	support	accuracy
0	1	1	1	36	1
1	1	1	1	82	

Tabla 17 Nieves parámetros estándar.

31	0
0	85

	precision	recall	f1-score	support	accuracy
0	1	1	1	31	0,9741
1	1	1	1	85	

Tabla 18 Nieves parámetros optimizados.

31	0
0	85

	precision	recall	f1-score	support	accuracy
0	1	1	1	31	0,9741
1	1	1	1	85	

Se observa que la optimización de los hiper parámetros no afectan al entrenamiento del modelo.

Una vez obtenidas todas las métricas de todos los parámetros se llega a la conclusión que los modelos con mejores resultados son SVM y XGBoost.

4.1.2 Pruebas de reconocimiento de voz y resultados.

Se han realizado las pruebas de transcripción de voz a texto con las siguientes frases:

1. ¿Dónde está el despacho?
2. Acércate a la cocina y tráeme un vaso de leche.
3. Quiero ir a la habitación de mis padres porque quiero coger el libro azul.
4. ¿Dónde está el cuarto de baño? Quiero ir a lavarme la cara.
5. Ve al cuarto de baño y trae el cepillo de dientes.

Los resultados se pueden ver en las Fig. 57, Fig. 58, Fig. 59, Fig. 60,Fig. 61 Fig. 62, Fig. 63, Fig. 64 y Fig. 65.

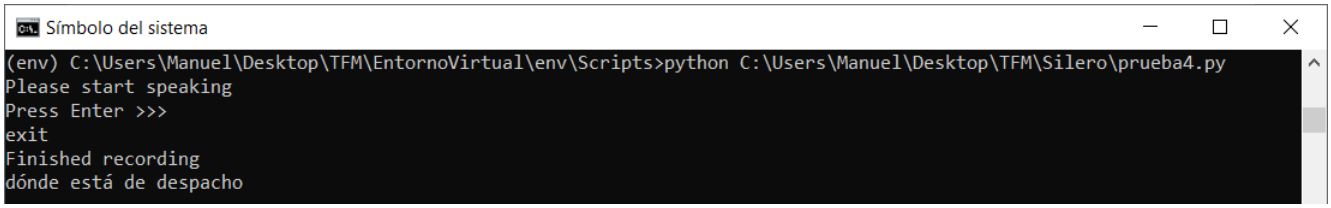

Fig. 56 Prueba frase 1 STT *Silero*.

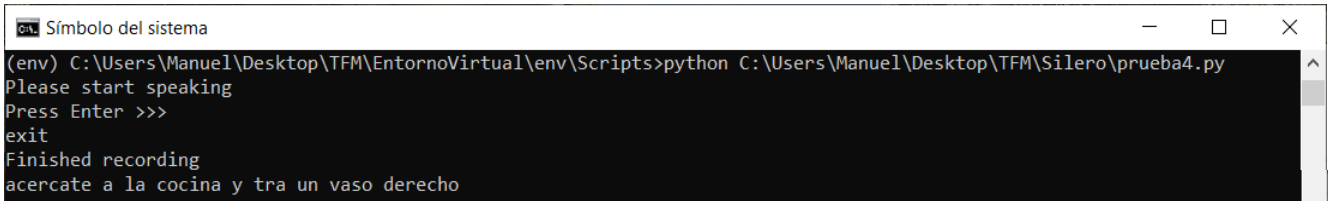

Fig. 57 Prueba frase 2 STT *Silero.*

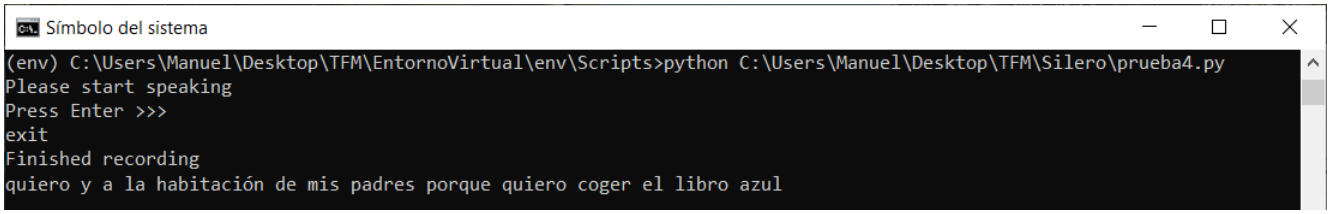

Fig. 58 Prueba frase 3 STT *Silero.*

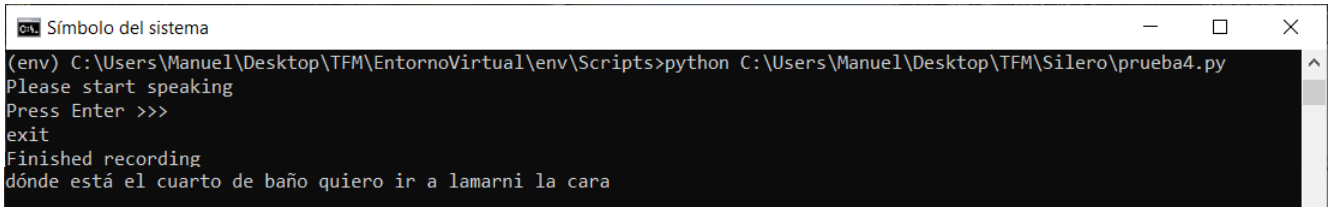

Fig. 59 Prueba frase 4 STT *Silero.*

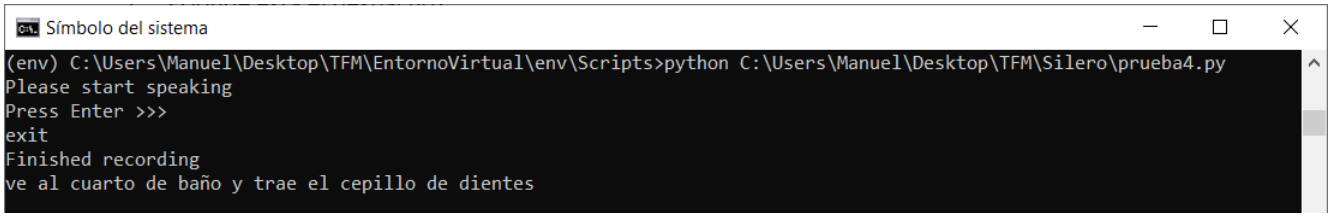

Fig. 60 Prueba frase 5 STT *Silero.*

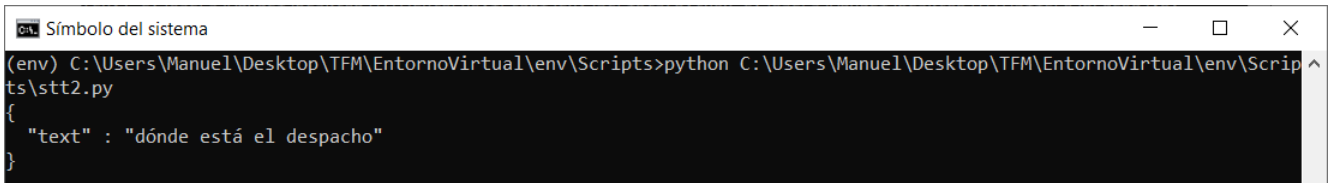

Fig. 61 Prueba frase 1 STT *Vosk.*

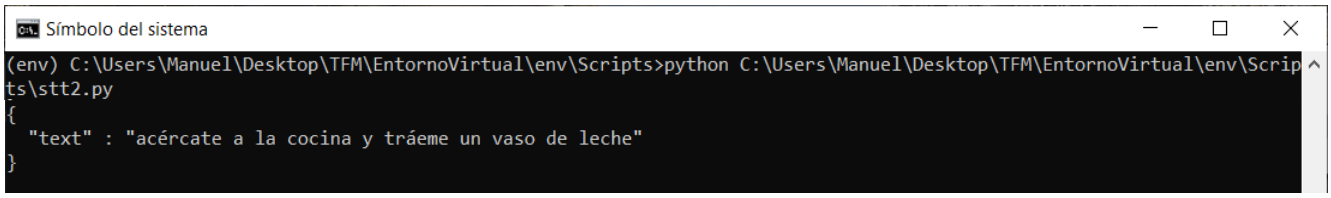

Fig. 62 Prueba frase 2 STT *Vosk.*

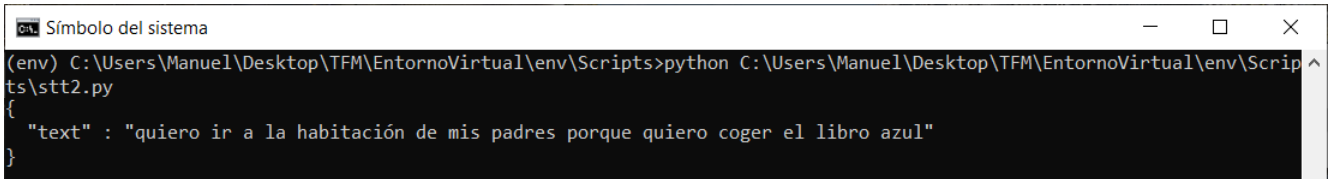

Fig. 63 Prueba frase 3 STT *Vosk.*

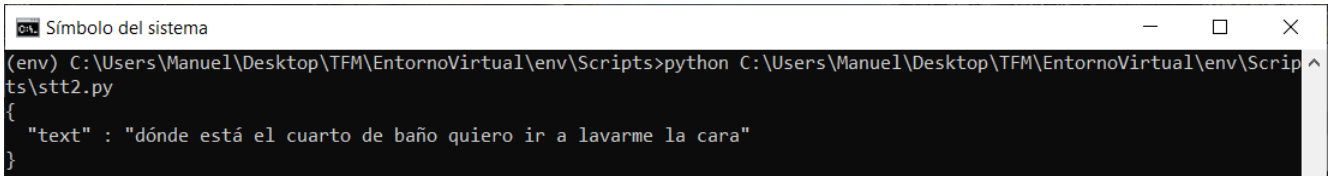

Fig. 64 Prueba frase 4 STT *Vosk.*

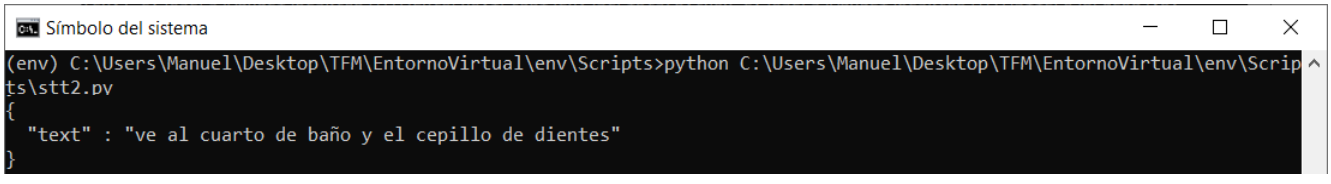

Fig. 65 Prueba frase 5 STT *Vosk.*

Se observa que el modelo de reconocimiento de voz que proporciona *Vosk* es más preciso que el modelo de *Silero*. Además, como se explicó en el apartado 3.3.2, está optimizado para dispositivos embebidos como la *Raspberry Pi*.

4.1.3 Pruebas de reconocimiento de lenguaje natural y resultados.

En el caso de *NLP* se comprueba el funcionamiento utilizando las mismas frases que el apartado 4.1.2.

Como se puede ver en las Fig. 66, Fig. 67, Fig. 68, Fig. 69 y Fig. 70 con la clase clase *Matcher* y el sistema experto desarrollados en el apartado 3.3.3, es posible obtener los verbos, los lugares y los objetos de las frases. Además, se verifica que una vez obtenida dicha información es posible asignar las coordenadas definidas a dichos lugares y objetos.

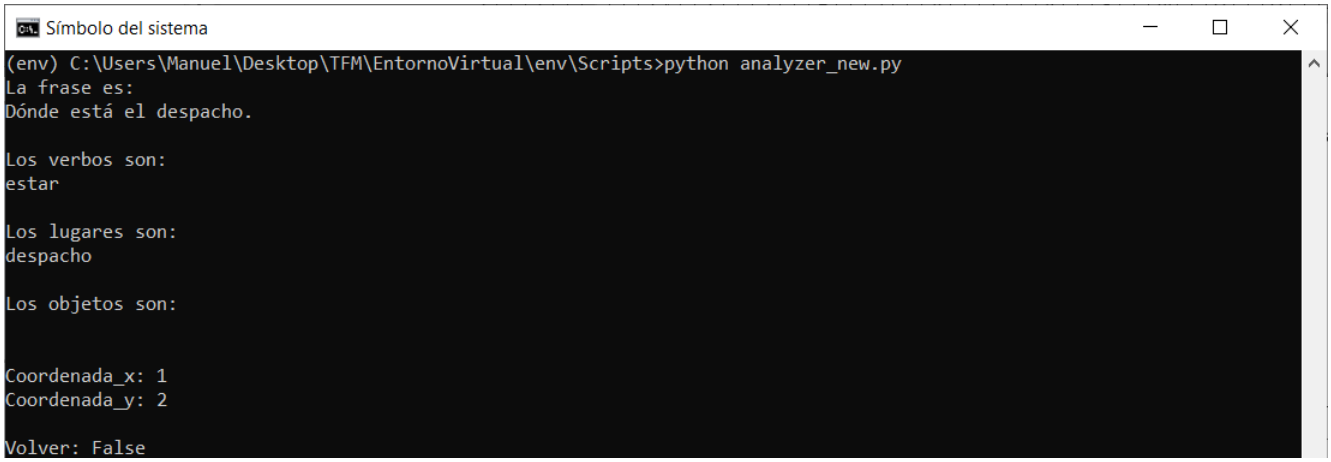

Fig. 66 Prueba frase 1 NLP.

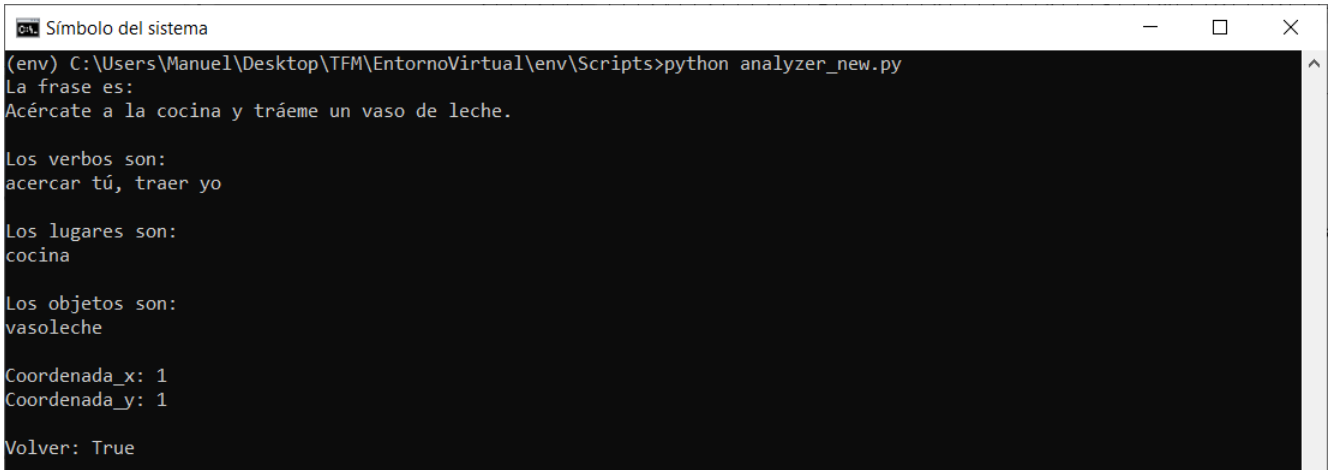

Fig. 67 Prueba frase 2 NLP.

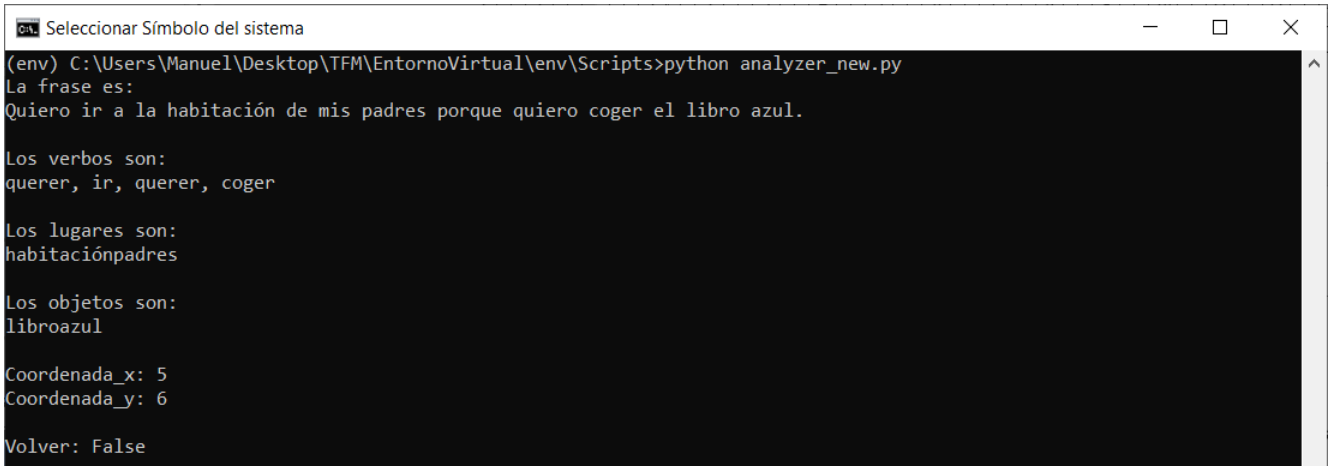

Fig. 68 Prueba frase 3 NLP.

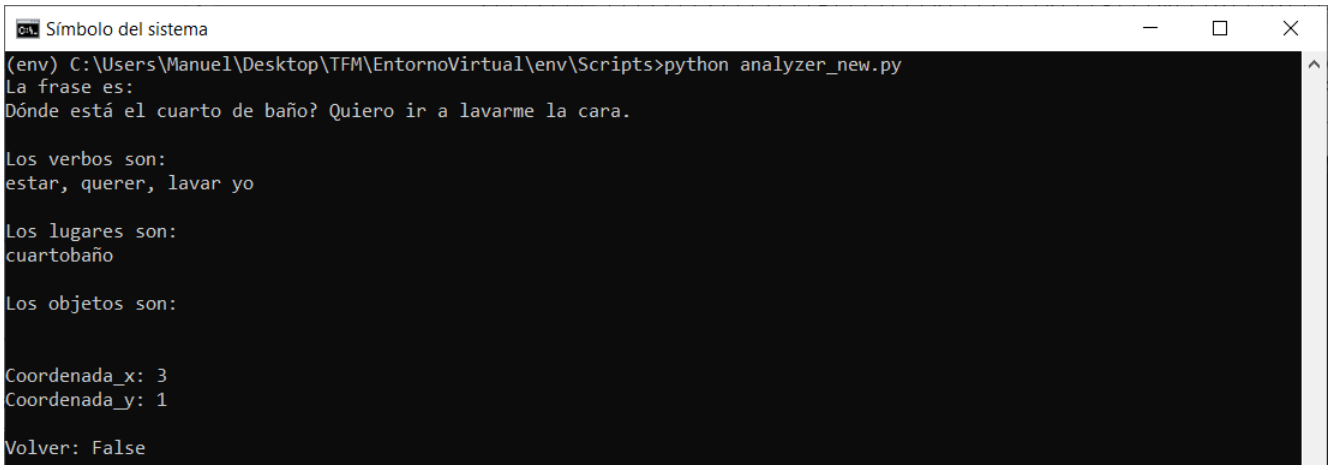

Fig. 69 Prueba frase 4 NLP

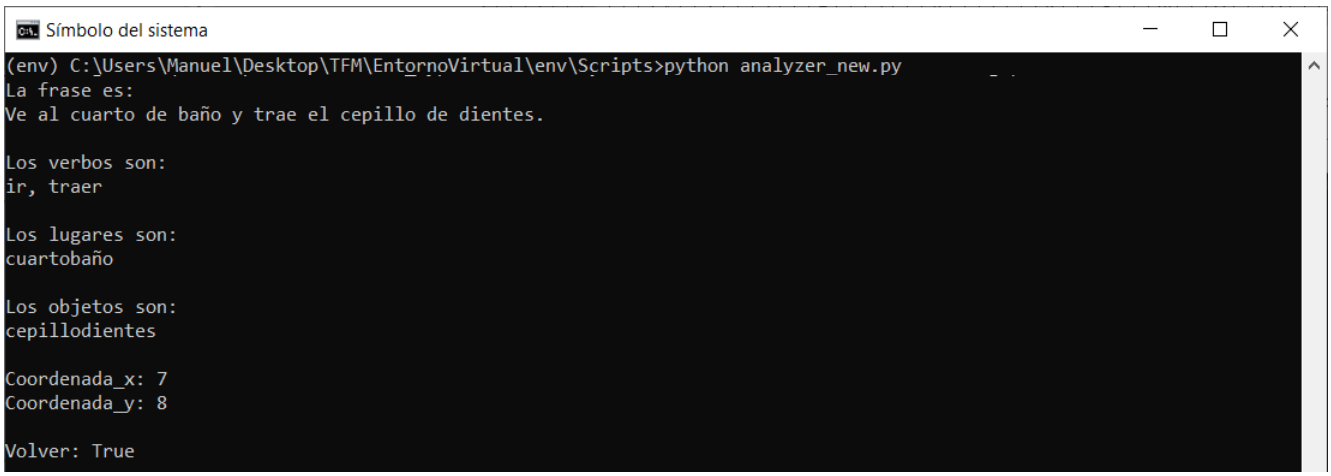

Fig. 70 Prueba frase 5 NLP

4.1.4 Pruebas de funcionamiento *BackEnd* y resultados.

Previo al despliegue de la aplicación a la Nube se puede lanzar de forma local simulando el servidor de *Google* desde la propia máquina (*localhost*) usando el botón de *Run As AppEngine* en Eclipse. Además, también se puede simular el *Datastore* de *Google* usando el comando *gcloud beta emulators datastore start* en la terminal de *Windows* y utilizando como variable de entorno en *Run configuration* de Eclipse la dirección *url* que proporciona tal y como se ve en la Fig. 71

```
[datastore] If you are using a library that supports the DATASTORE_EMULATOR_HOST environment variable, run:
[datastore]
[datastore]   export DATASTORE_EMULATOR_HOST=localhost:8081
```

Fig. 71 Url simulación Datastore.

Una vez desplegado en local, se pueden probar los servicios utilizando la aplicación *Postman* [74] que permite realizar peticiones *REST*. Una característica importante que tiene es que muestra cómo se realizarían esas peticiones en varios lenguajes de programación. Se ha usado JavaScript – jQuery como plantilla en el apartado 3.5.1.

Postman se ha utilizado para comprobar el funcionamiento de los diferentes métodos implementados en el Endpoint tal y como se ve en las Fig. 72, Fig. 73, Fig. 74, Fig. 75, Fig. 76 y Fig. 77.

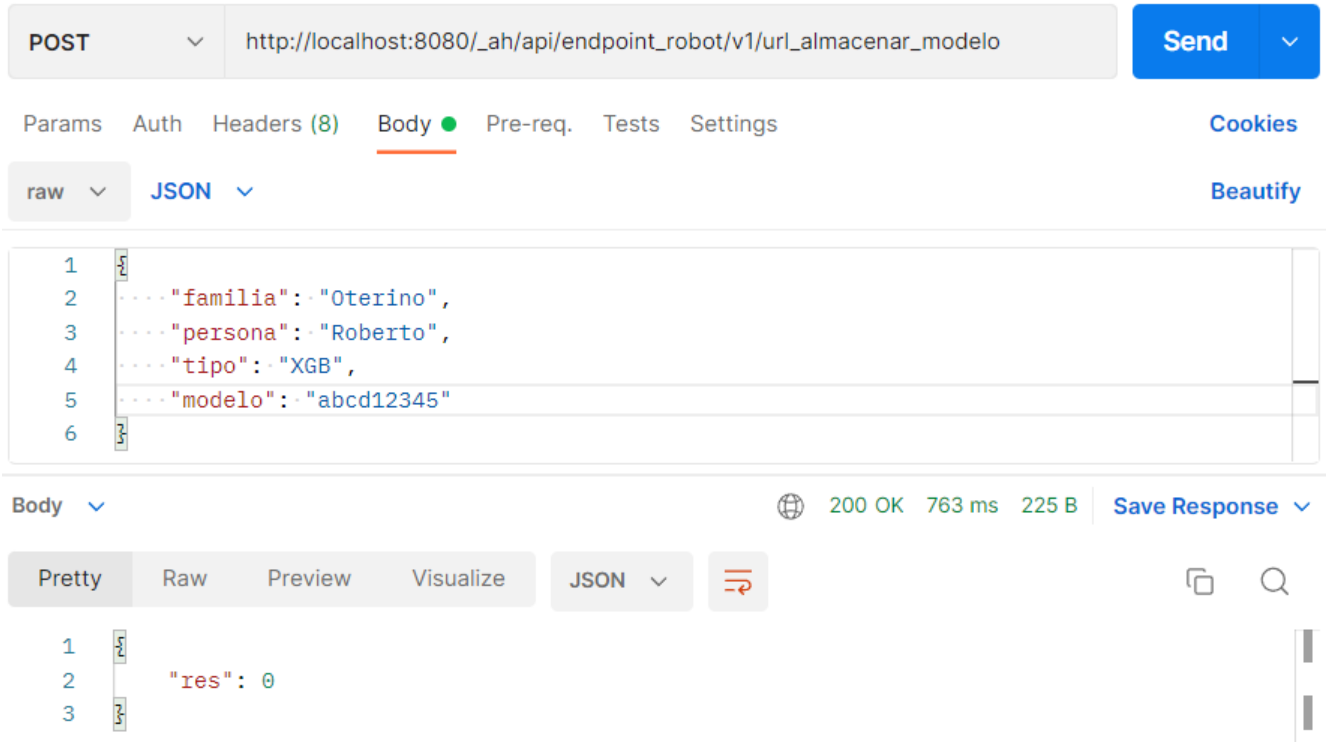

Fig. 72 Prueba local almacenamiento modelo

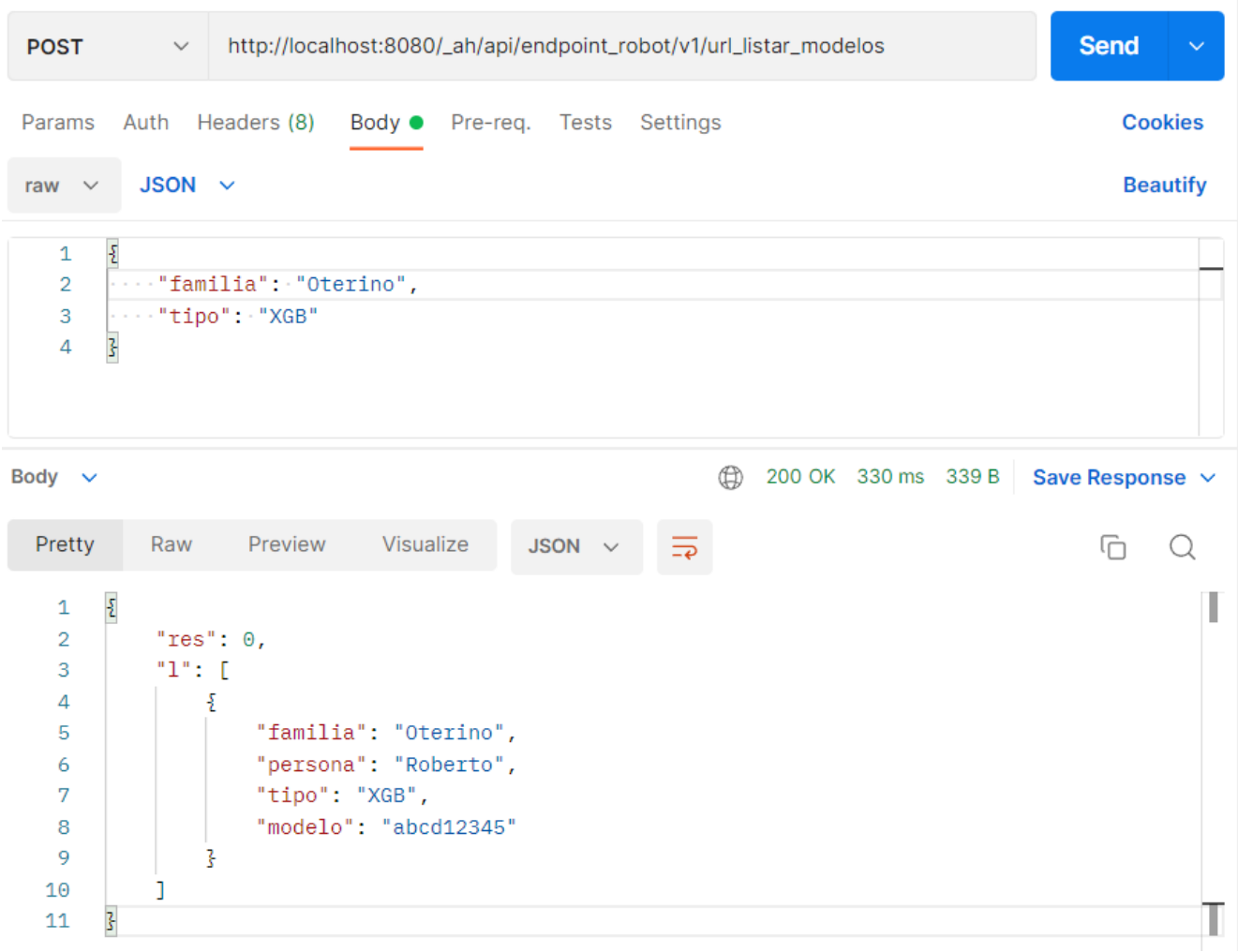

Fig. 73 Prueba local listado modelos.

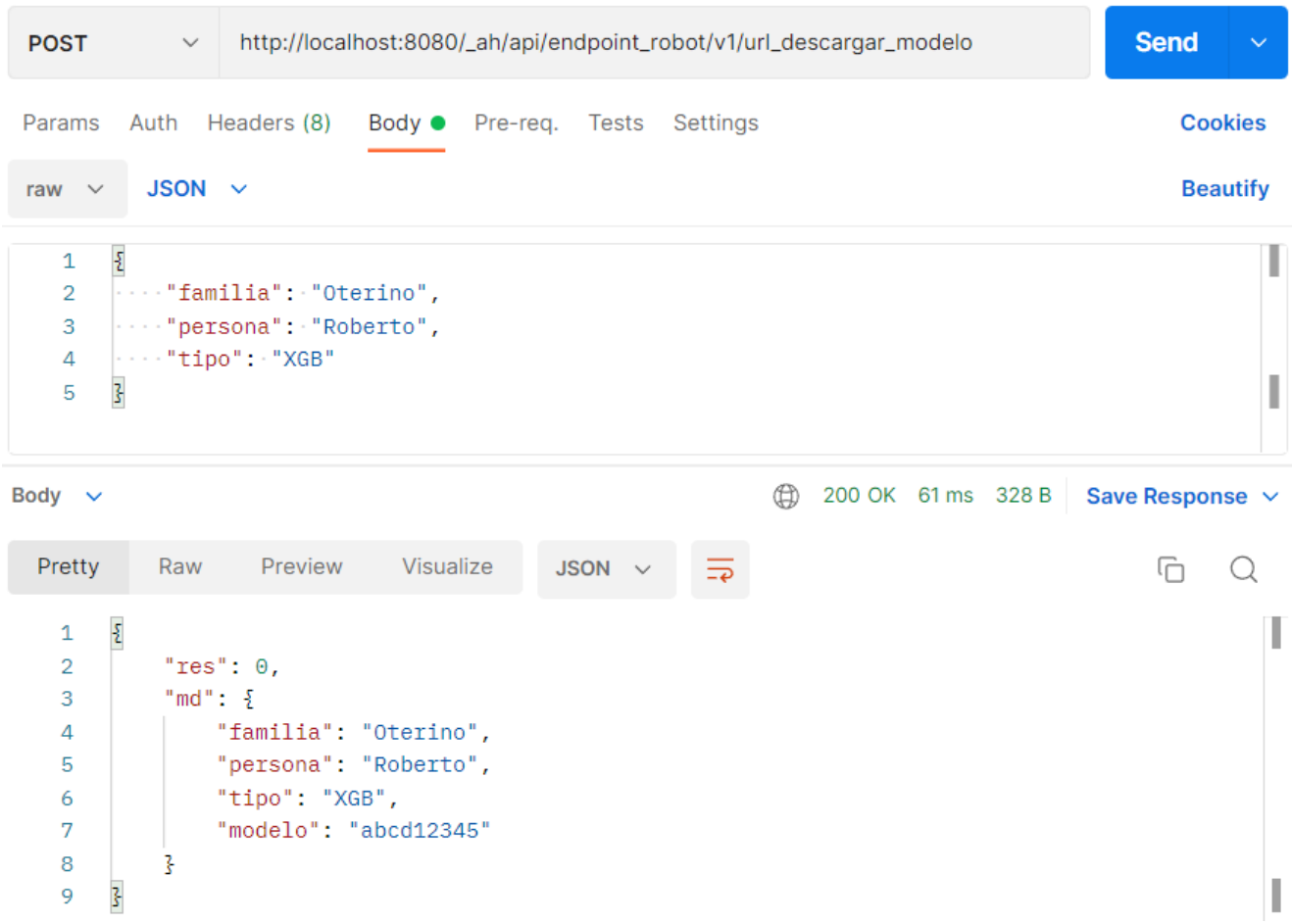

Fig. 74 Prueba local descarga modelo.

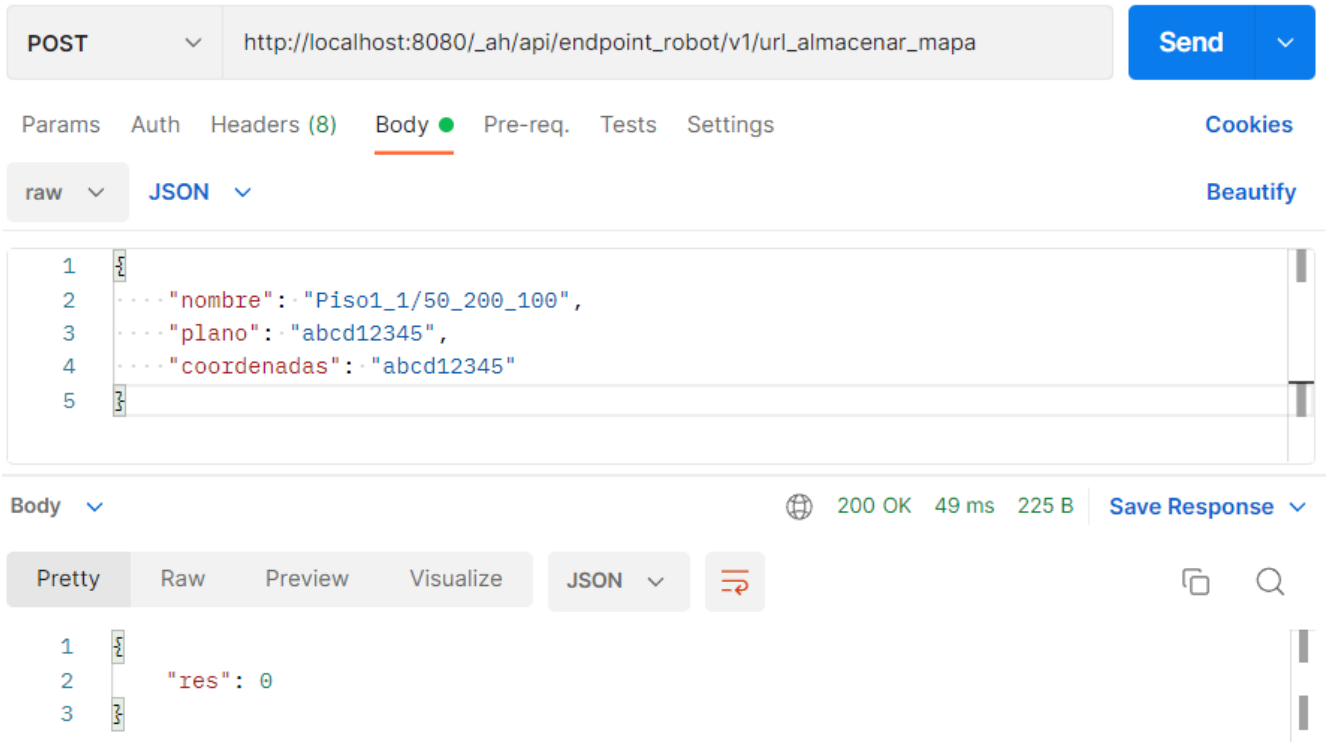

Fig. 75 Prueba local almacenar mapa.

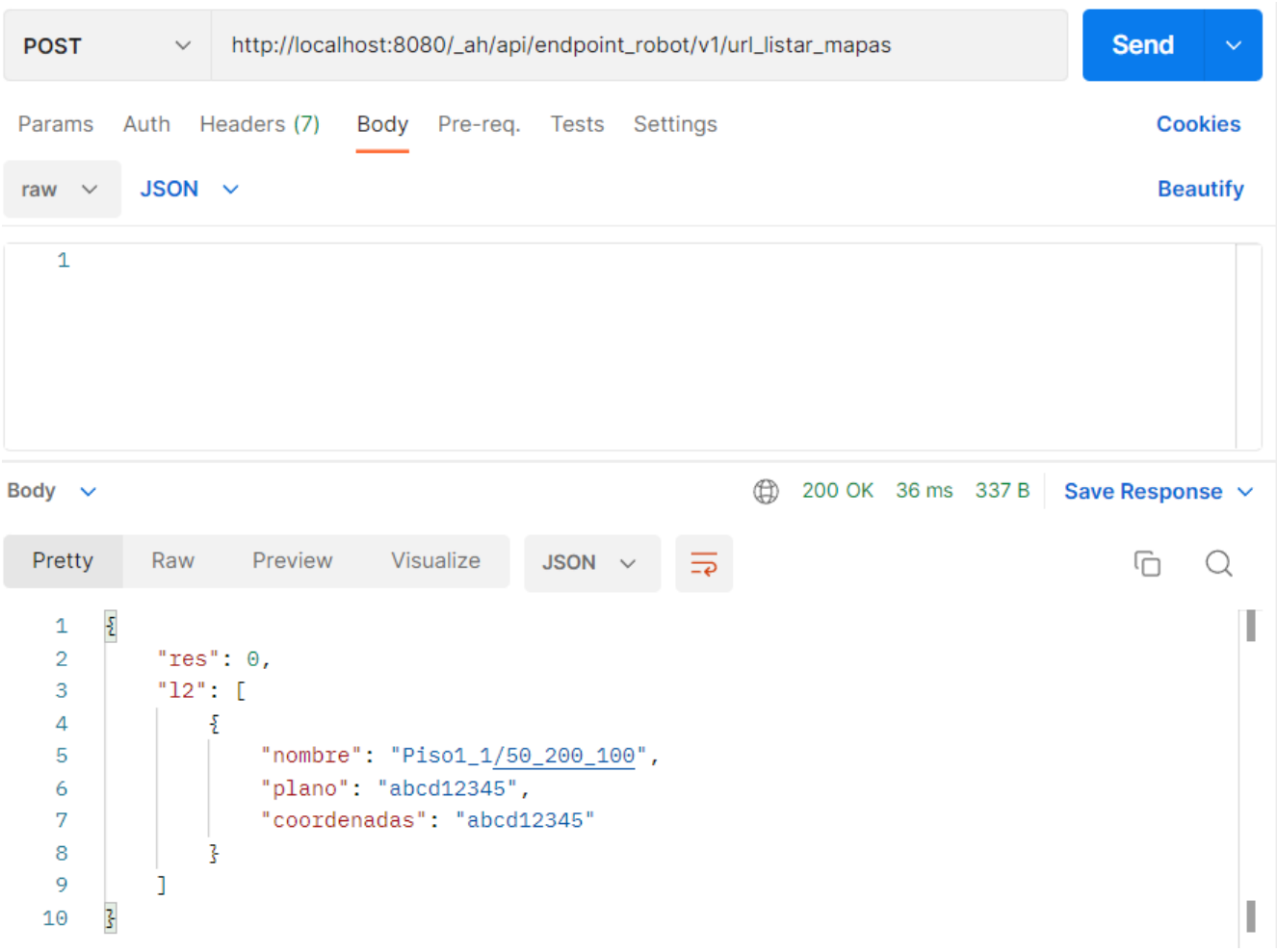

Fig. 76 Prueba local listar mapas.

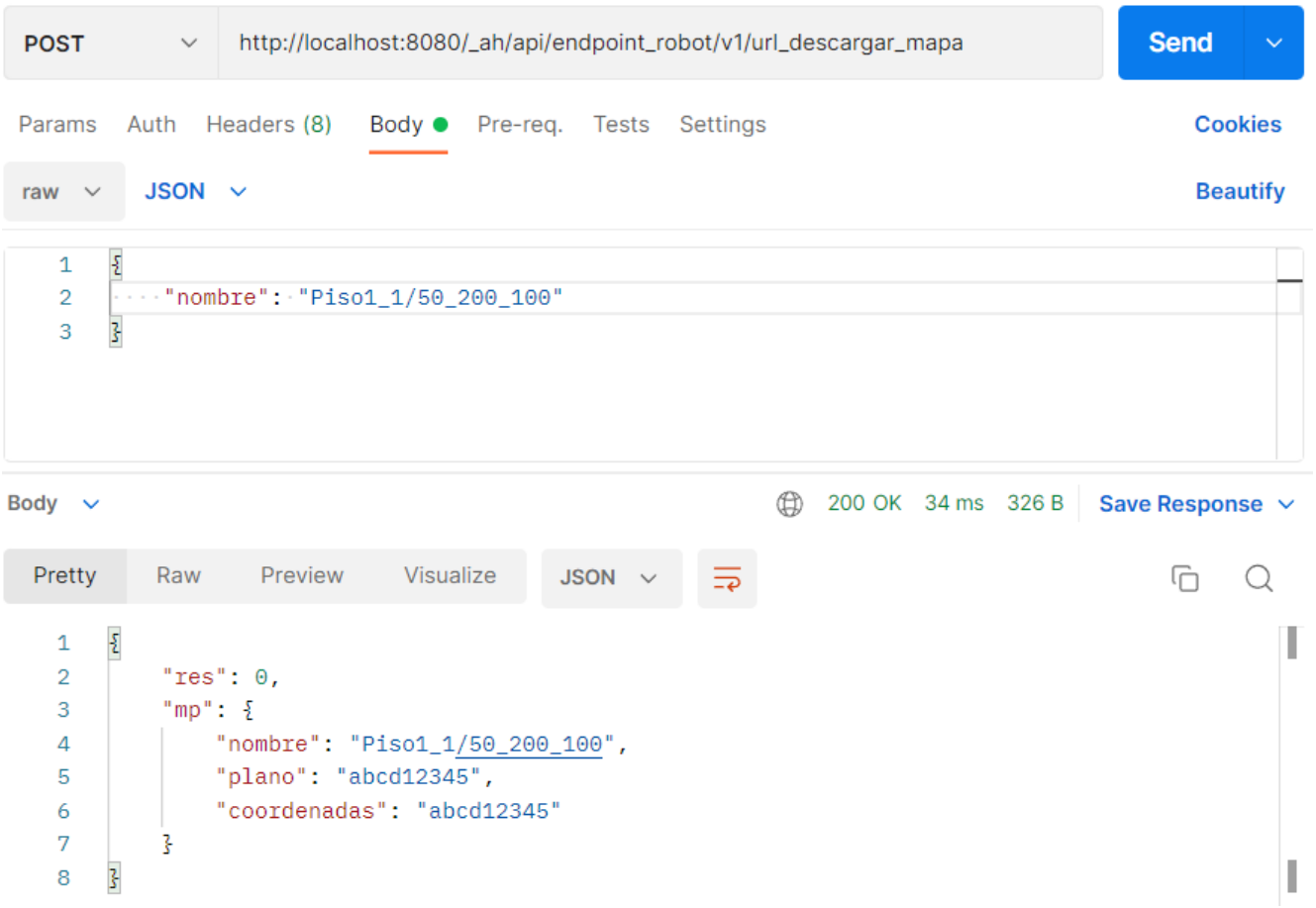

Fig. 77 Prueba local descargar mapa.

Se observa que en todas las respuestas de las peticiones el atributo *res* tiene el valor 0. Esto indica que se han realizado correctamente, en caso contrario indicaría que se ha producido un error. Esto permite obtener una trazabilidad tal y como se ha explicado en el apartado 3.5.1.

Por otro lado Eclipse también permite lanzar la aplicación en modo depuración, luego una vez localizado el error se pueden utilizar puntos de ruptura en el código para conocer el estado de las variables y averiguar por qué se está produciendo.

Una vez comprobado el correcto funcionamiento de forma local, ya se puede desplegar en la Nube de *Google* y realizar las mismas pruebas. De nuevo se ha utilizado *Postman* para verificar que los métodos funcional tal y como muestran las Fig. 78, Fig. 79, Fig. 80, Fig. 81, Fig. 82 y Fig. 83.

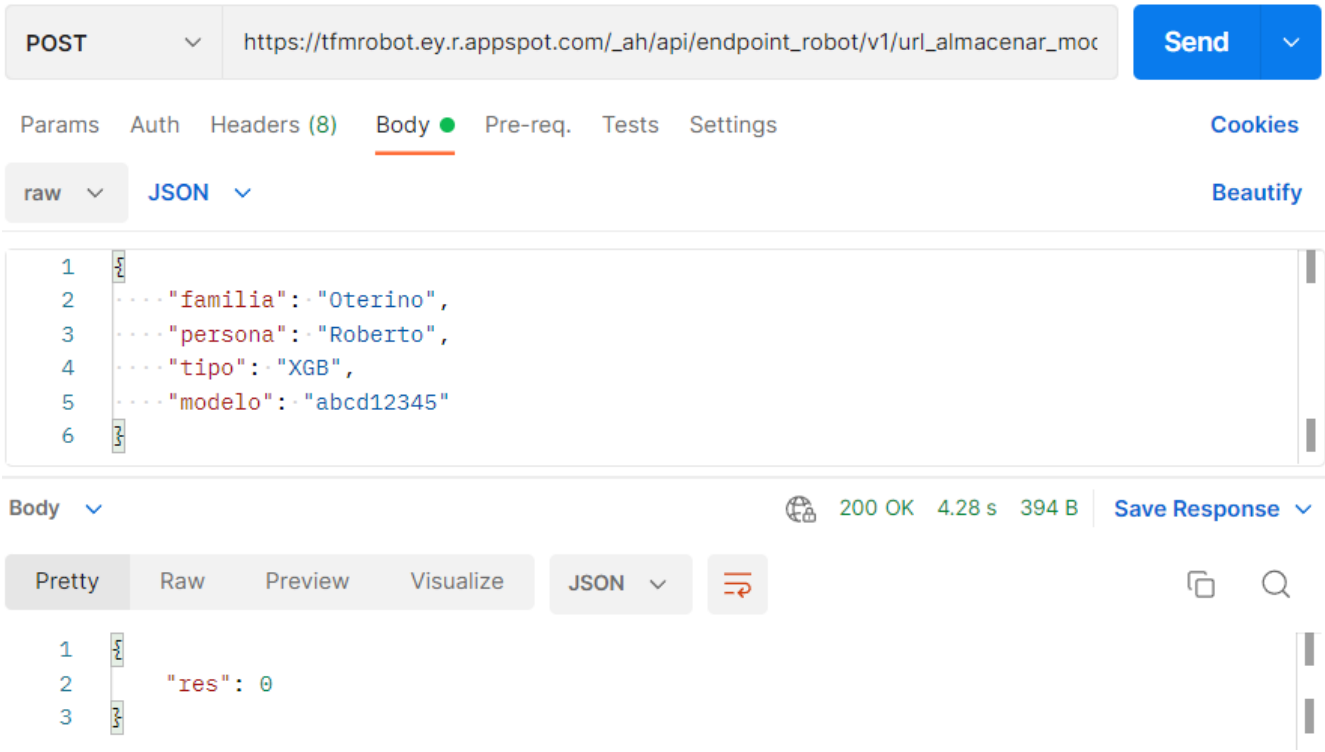

Fig. 78 Prueba on cloud almacenar modelo.

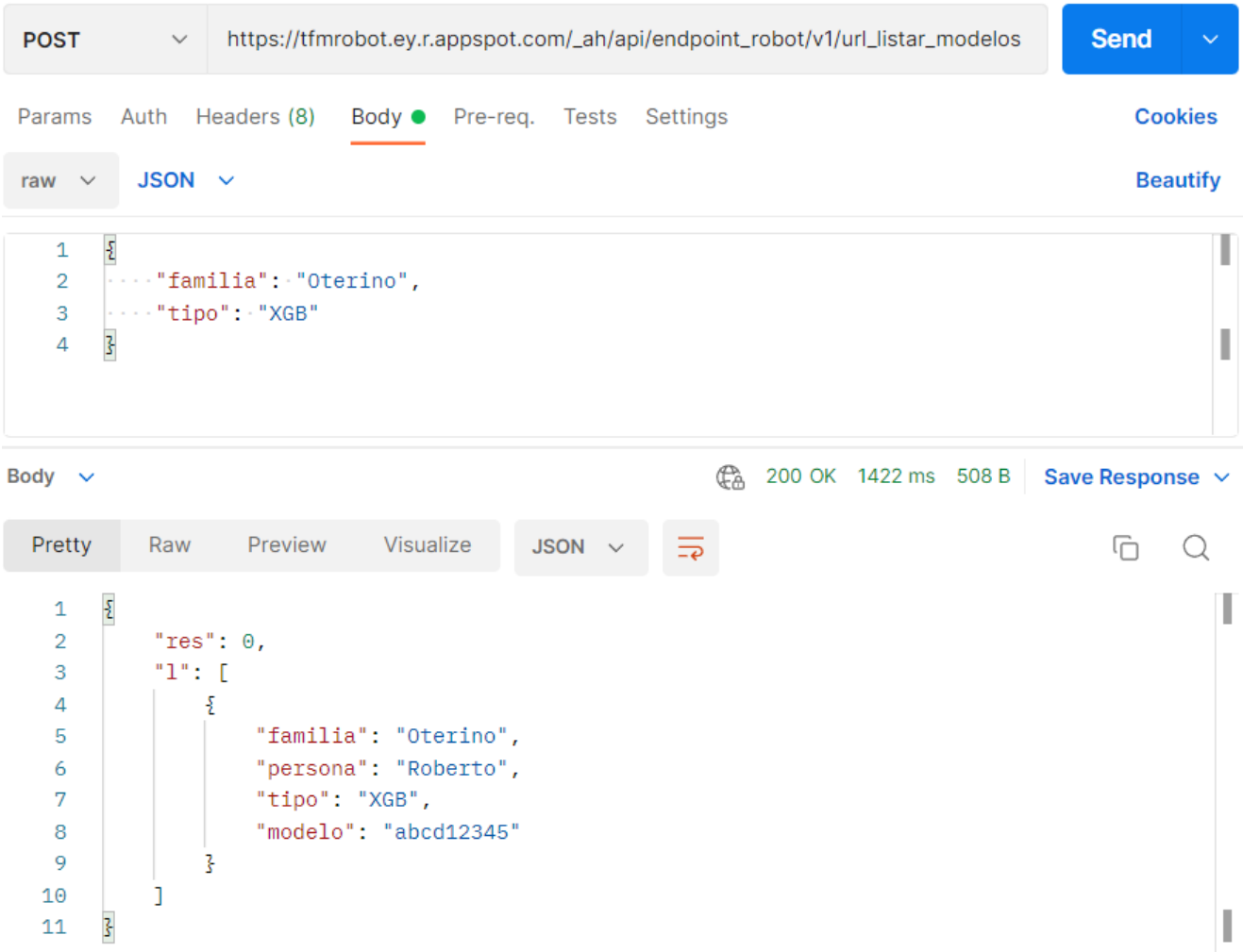

Fig. 79 Prueba on Cloud listar modelos.

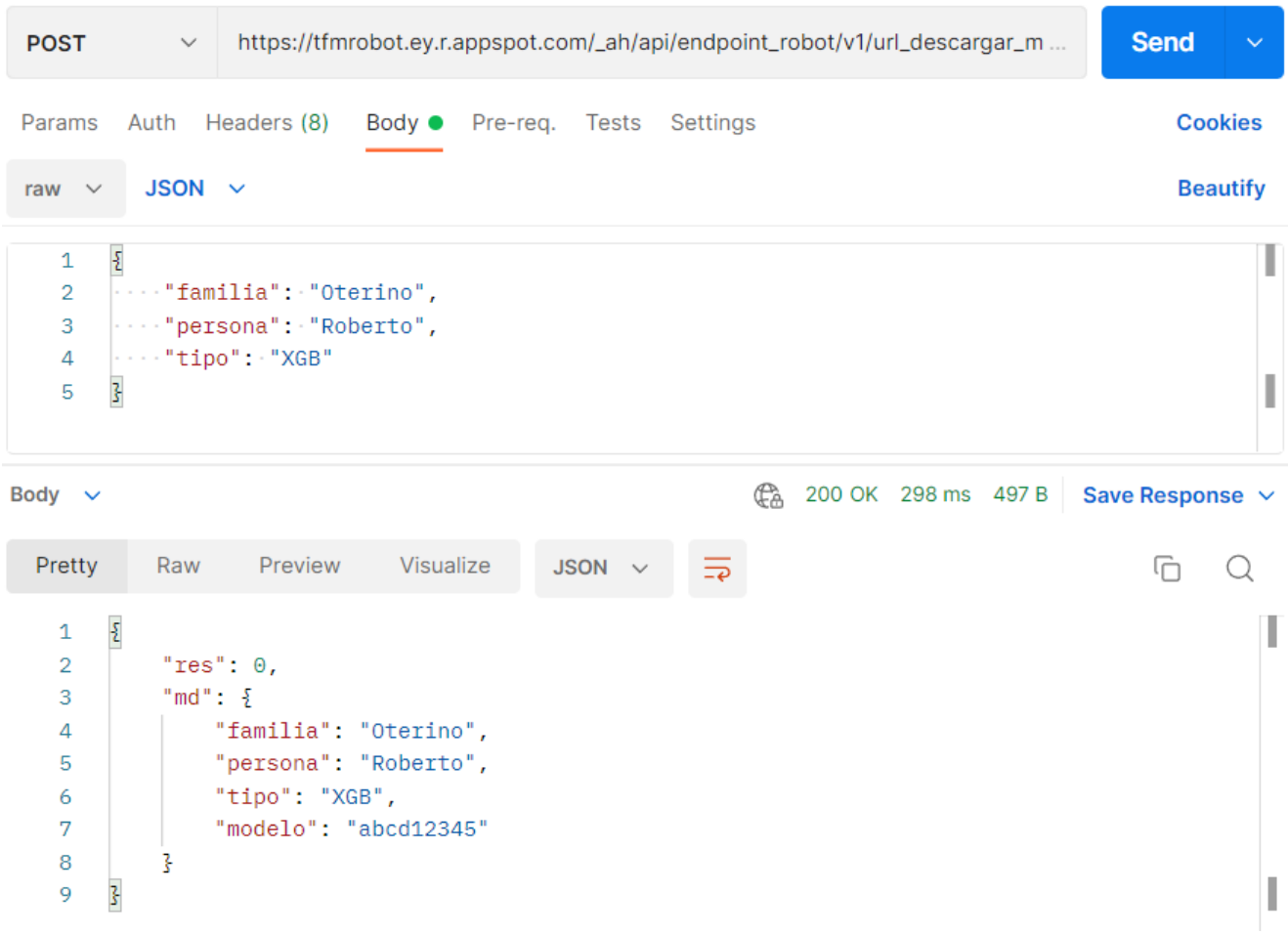

Fig. 80 Prueba on Cloud descargar modelo.

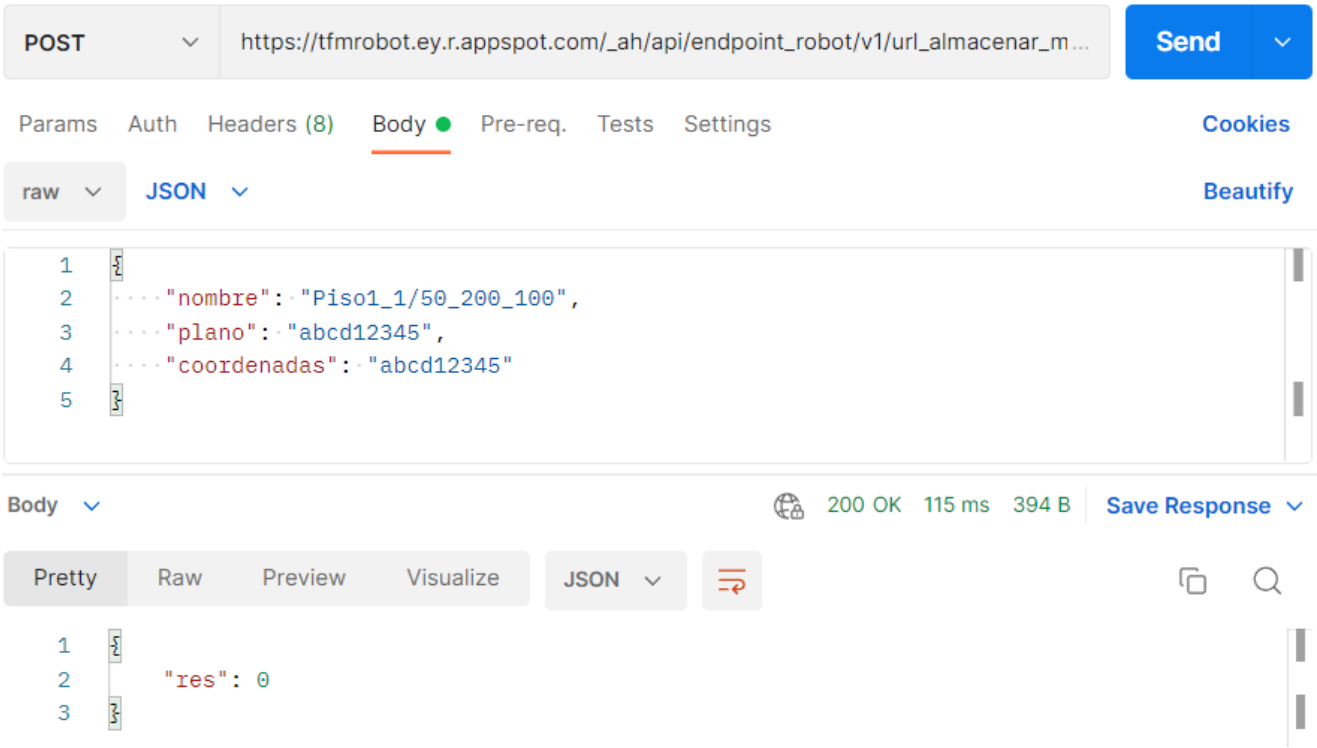

Fig. 81 Prueba on Cloud almacenar mapa.

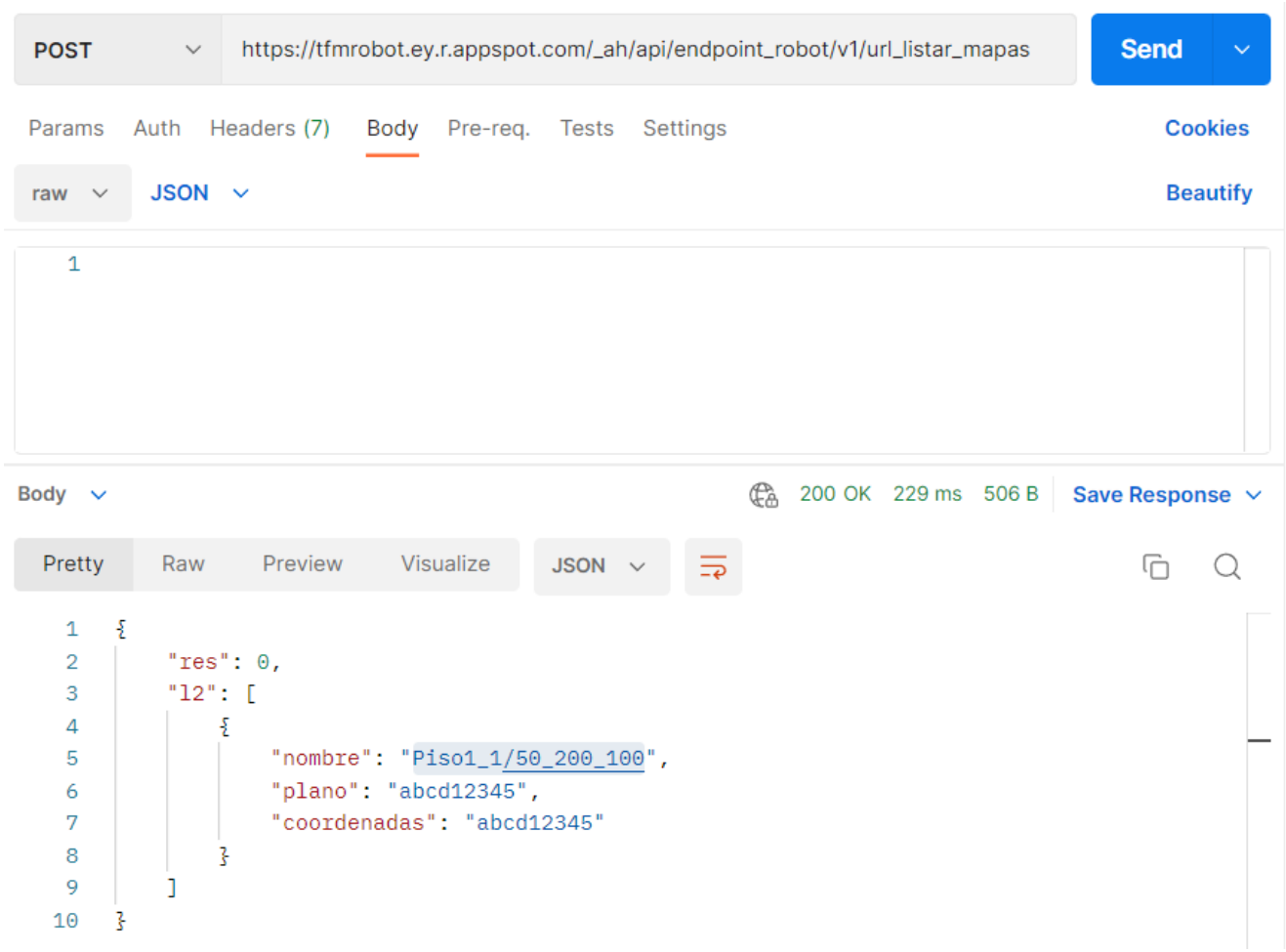

Fig. 82 Prueba on Cloud listar mapas.

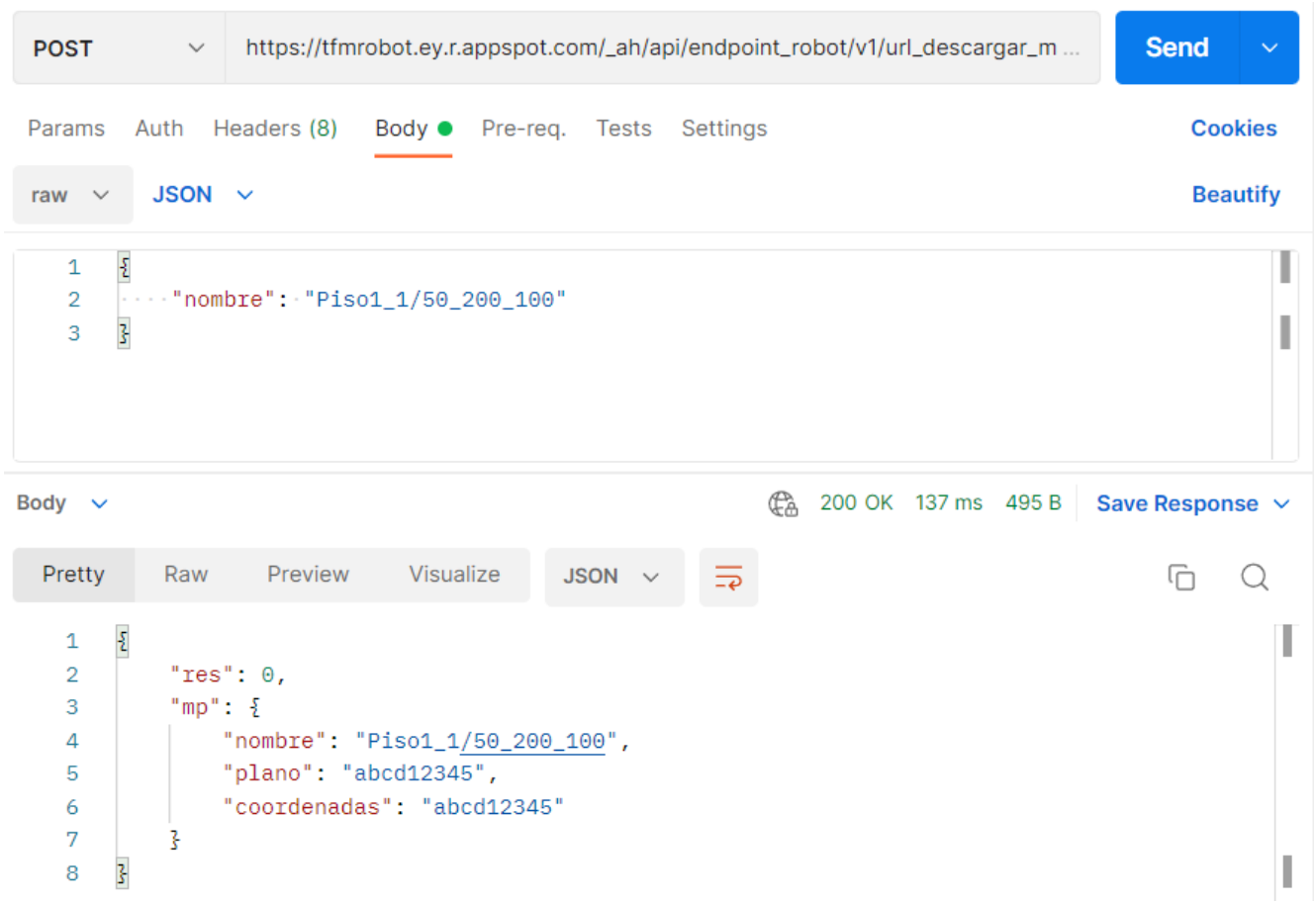

Fig. 83 Prueba on Cloud descargar mapa.

En la Fig. 78 se puede ver que la latencia de la petición es mayor que el resto, esto es debido a que la instancia de la aplicación estaba recién levantada. Además, también se puede observar que la latencia media en las peticiones que se hacen sobre la aplicación desplegada en la Nube es mayor que la local.

Por otro lado desde el *Dashbpard* de *App Engine* de la consola de *Google* también se puede visualizar información sobre las peticiones que se han realizado y la traza de las mismas tal y como muestran las Fig. 84, Fig. 85 y Fig. 86.

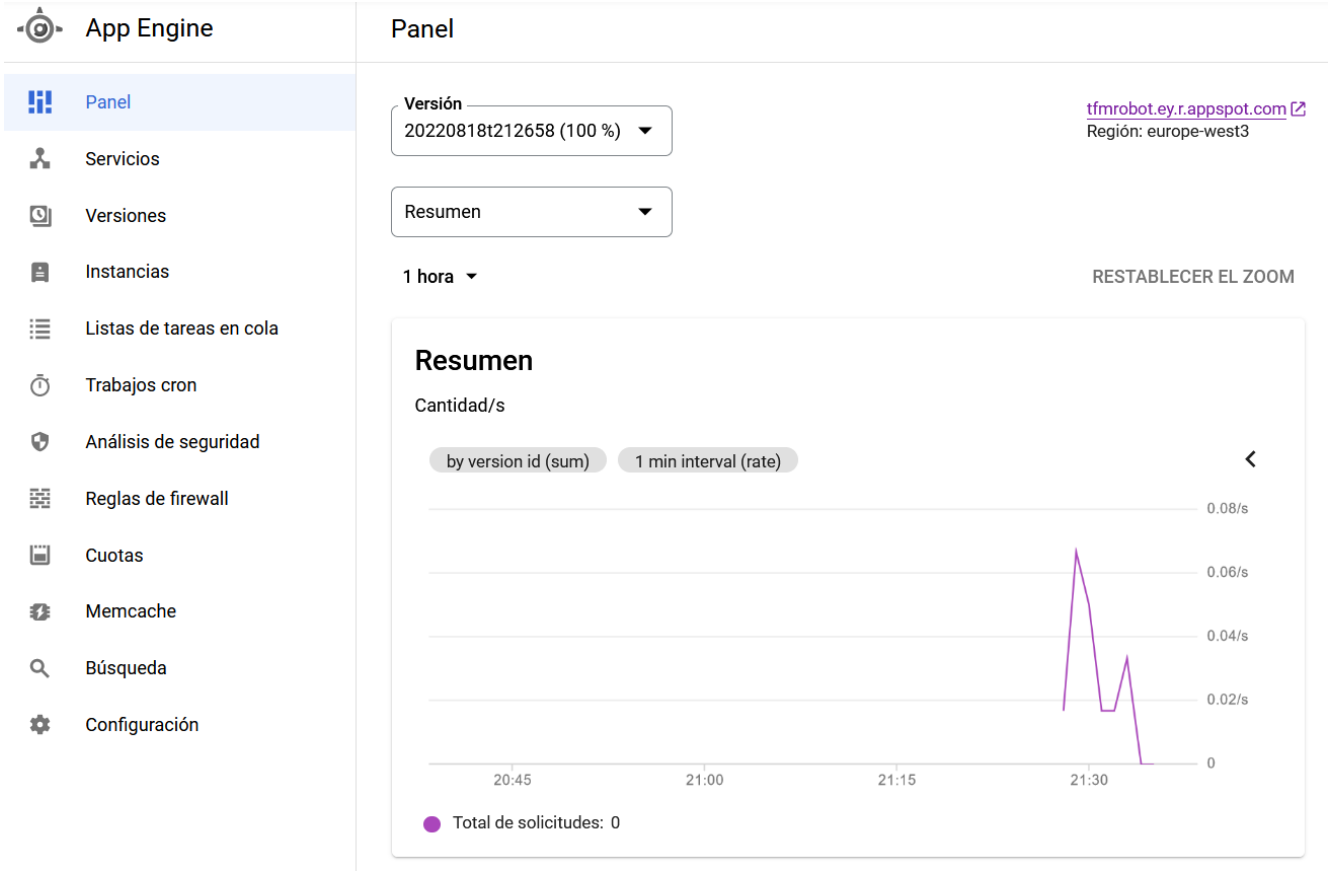

Fig. 84 Dashboard App Engine.

Carga actual

URI	Solicitudes por minuto actual	Solicitudes últimas 24 horas	Ciclos de mega CPU de tiempo de ejecución última hora	Promedio de latencia última hora	Seguimiento últimas 24 horas
/_ah/api/endpoint_robot /v1/url_almacenar_modelo	0	2	4,400	2,142 ms	Ver seguimiento
/_ah/api/endpoint_robot /v1/url_descargar_modelo	0.2	1	0	65 ms	Ver seguimiento
/_ah/warmup	0	1	4,399	3,728 ms	Ver seguimiento
/_ah/api/endpoint_robot /v1/url_listar_mapas	0.2	1	2,199	183 ms	Ver seguimiento
/_ah/api/endpoint_robot /v1/url_listar_modelos	0	1	4,399	1,373 ms	Ver seguimiento
/_ah/api/endpoint_robot /v1/url_descargar_mapa	0.2	1	2,199	89 ms	Ver seguimiento
/_ah/api/endpoint_robot /v1/url_almacenar_mapa	0.2	1	4,400	64 ms	Ver seguimiento

Fig. 85 Peticiones App Engine.

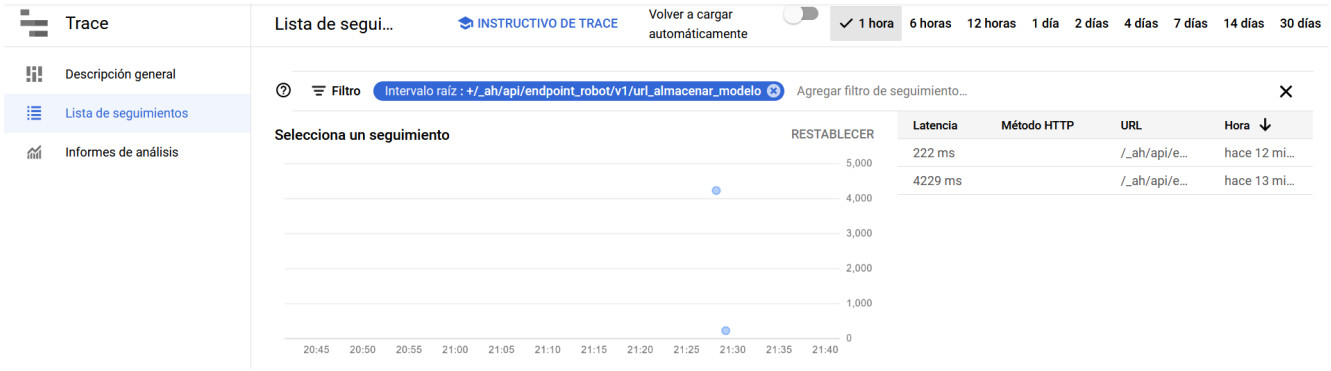

Fig. 86 Traza petición almacenar modelo.

Por último, se puede comprobar que se han guardado correctamente las entidades en el *Datastore* desde la consola de Google. En las Fig. 87 y Fig. 88 se ven los dos tipos de entidades almacenadas con sus atributos.

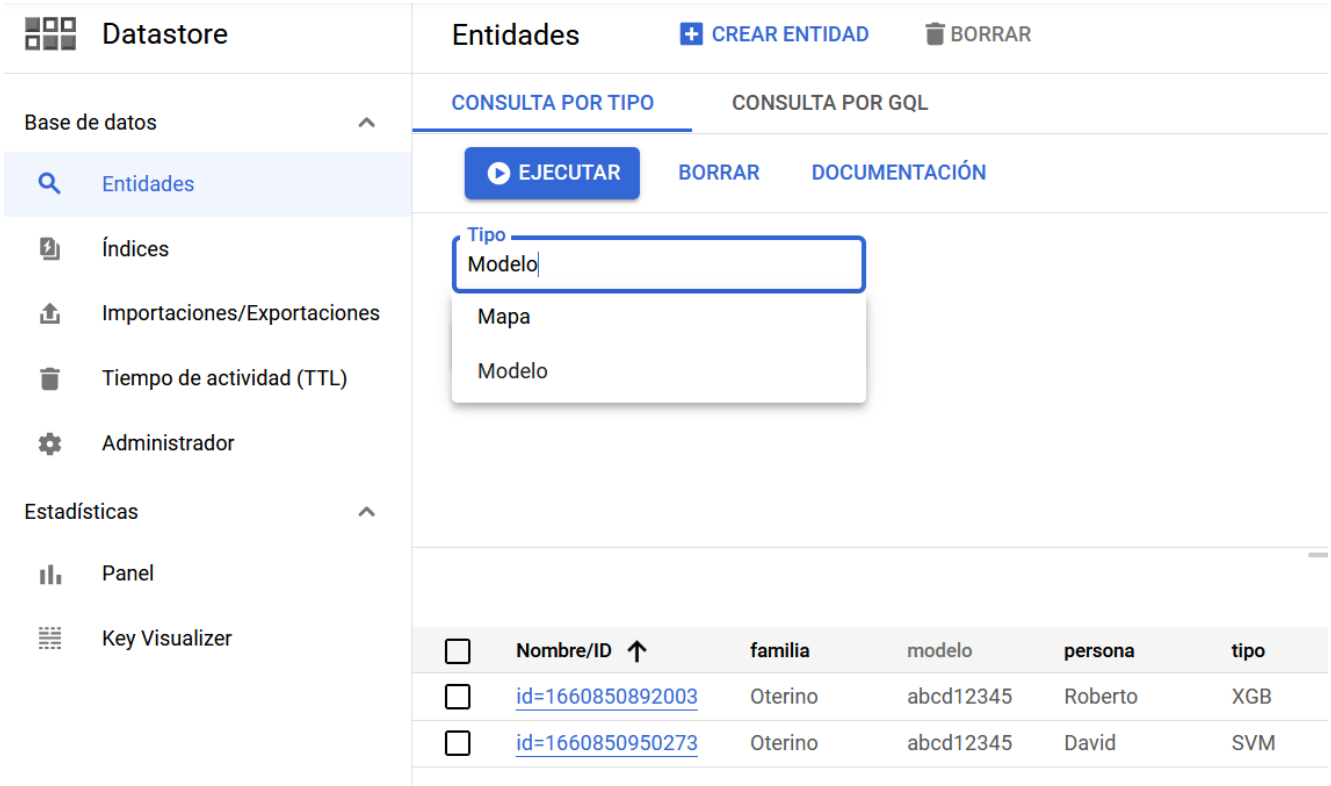

Fig. 87 Entidad modelo Datastore.

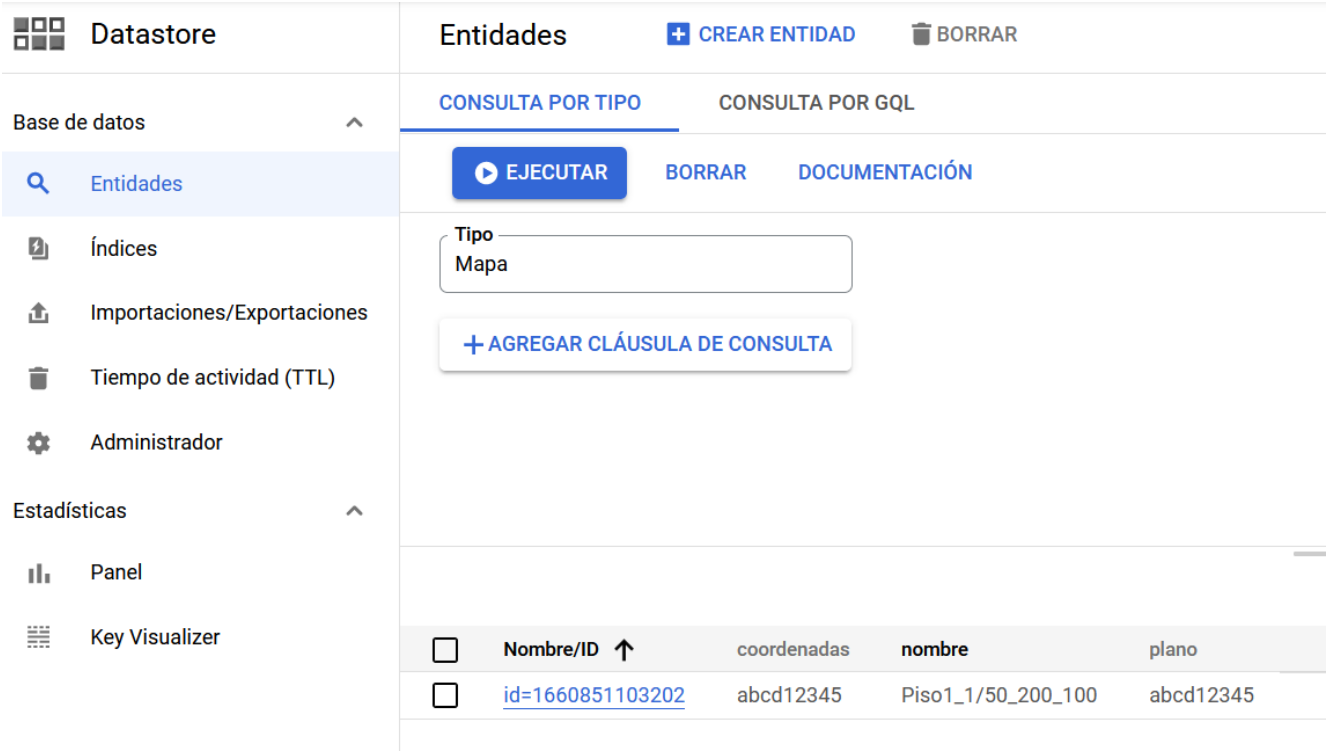

Fig. 88 Entidad mapa Datastore.

4.1.5 Pruebas de funcionamiento FrontEnd y resultados.

Para realizar las pruebas del *FrontEnd* de la aplicación se ha utilizado *Web Server for Chrome* [75] que permite lanzar páginas web desde una carpeta local a través de la red utilizando *http* tal y como se muestra en la Fig. 89. Además, se ha usado la consola que proporciona el propio navegador de modo que se puede visualizar los mensajes registrados con la función *console.log()*. En la Fig. 90 se ve la información al iniciar sesión en la página. Por otro lado, desde la pestaña *Sources* se pueden poner puntos de ruptura para depurar el código.

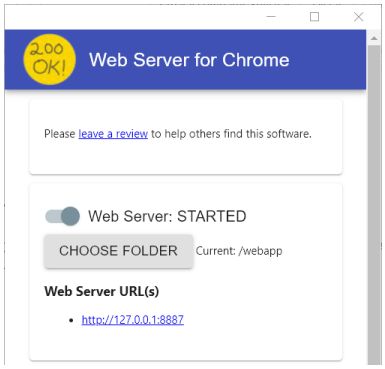

Fig. 89 Web Server for Chrome.

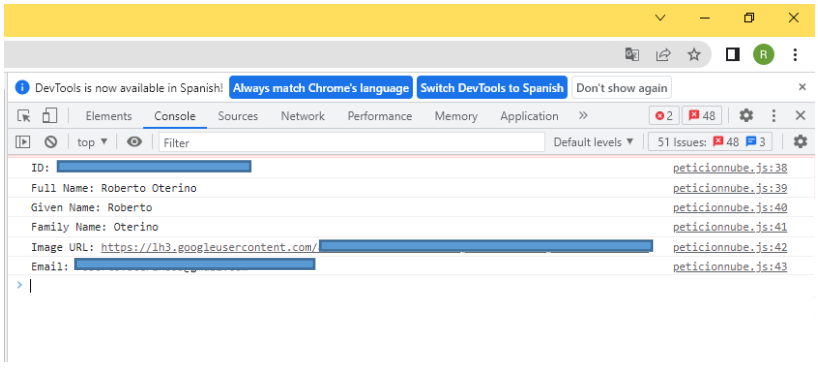

Fig. 90 Consola Chrome.

Como en el apartado 4.1.4 se ha comprobado el correcto funcionamiento de la aplicación desplegada en la Nube, las pruebas de la página web se han realizado simulando el servidor de *Google*. En la Fig. 91 y la Fig. 92 se muestran la consola de *Chrome*, las respuestas de los diferentes métodos implementados en el *BackEnd*. Además, se observa la interfaz web para uno de los roles de usuario.

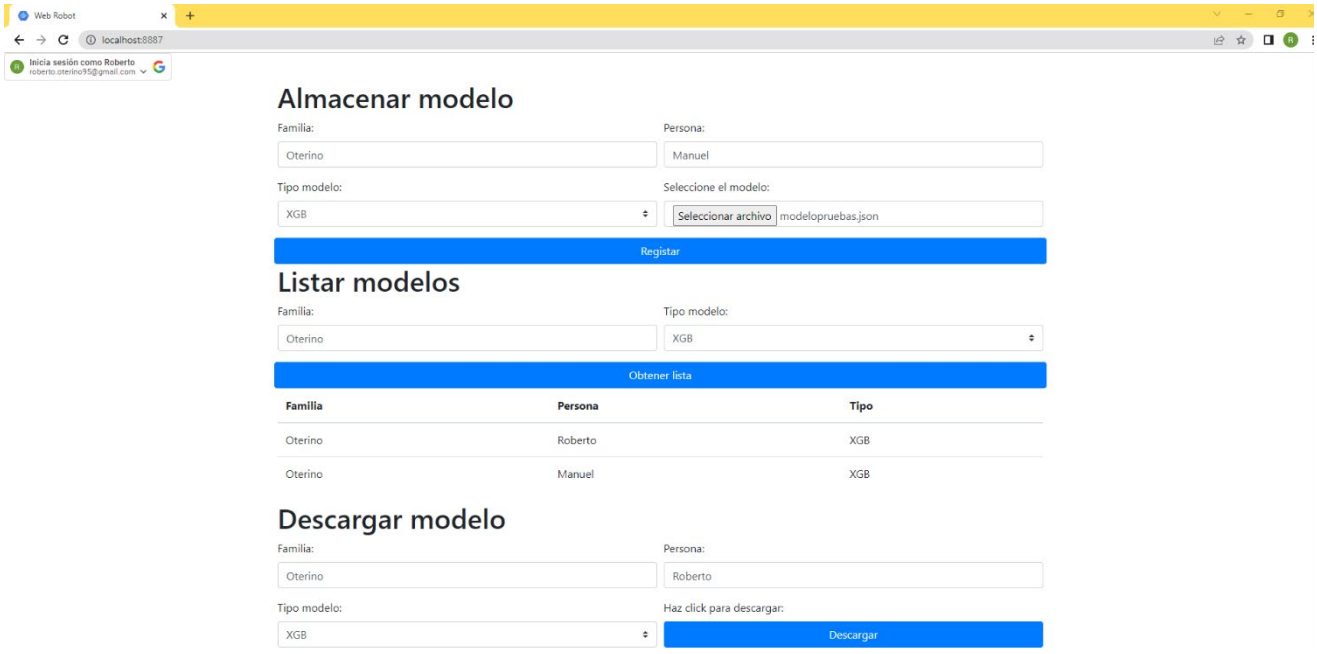

Fig. 91 Prueba métodos modelo FrontEnd.

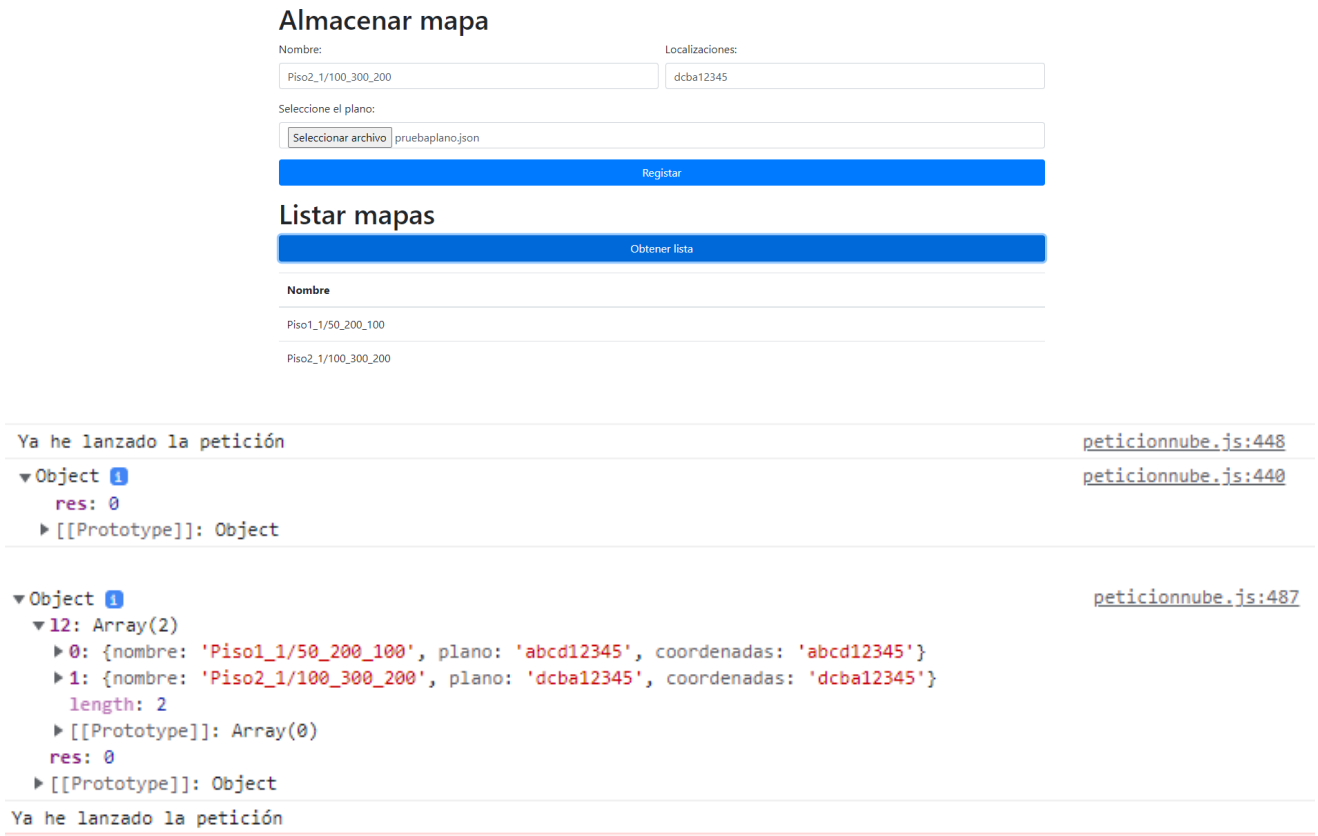

Fig. 92 Prueba métodos mapa FrontEnd.

En la Fig. 93 se observa la interfaz web para el otro rol de usuario y el funcionamiento de la *API Pub/Sub* de *Google*. Se observa que se ha recibido el código de respuesta de estado satisfactorio *HTTP 200 OK* que indica que la solicitud ha tenido éxito.

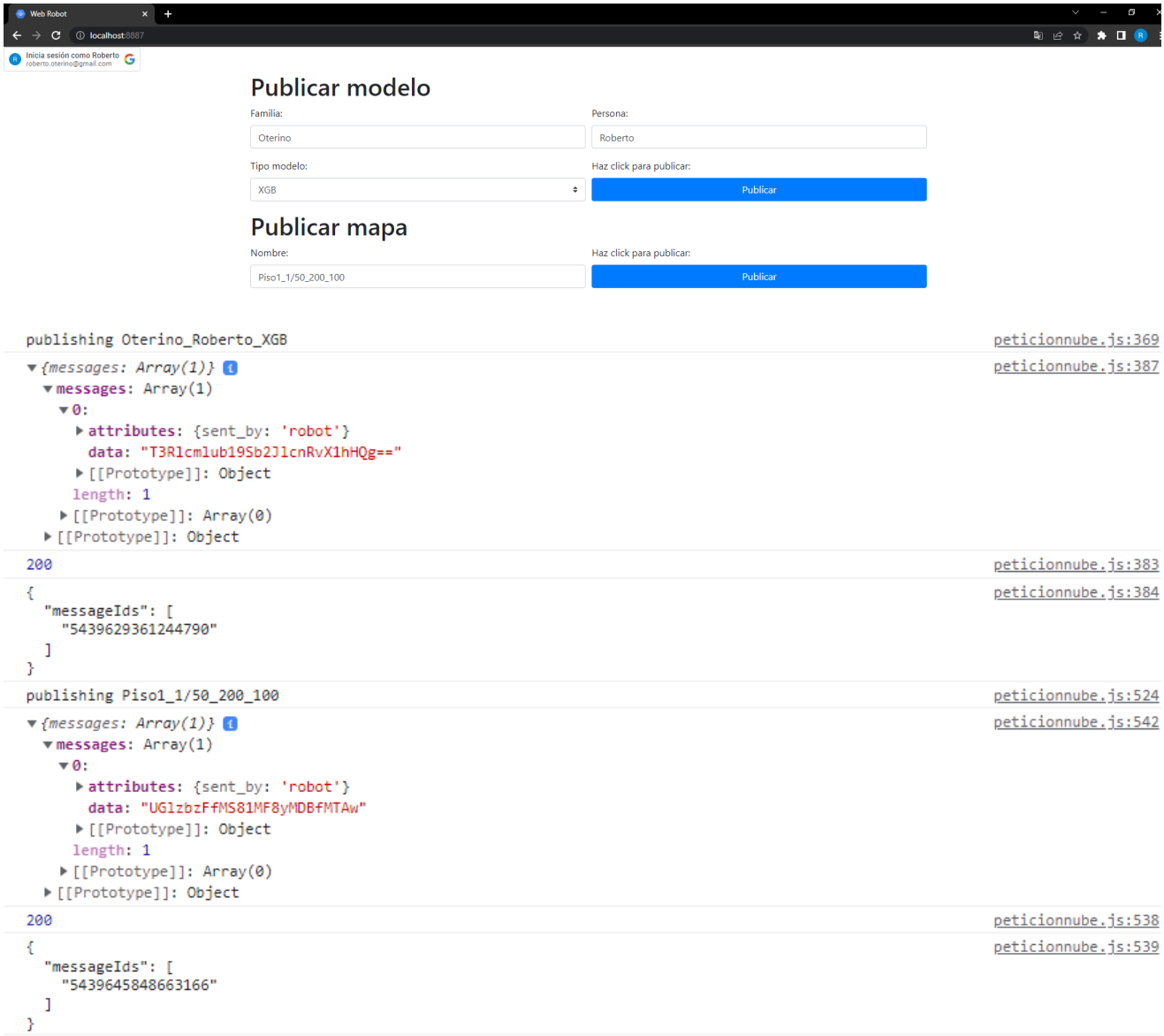

Fig. 93 Prueba publicadores FrontEnd.

Además, desde *la API Pub/Sub* de la consola de *Google* también se puede ver que en los temas creados se reciben los mensajes de los publicadores tal y como muestran las Fig. 94 y Fig. 95.

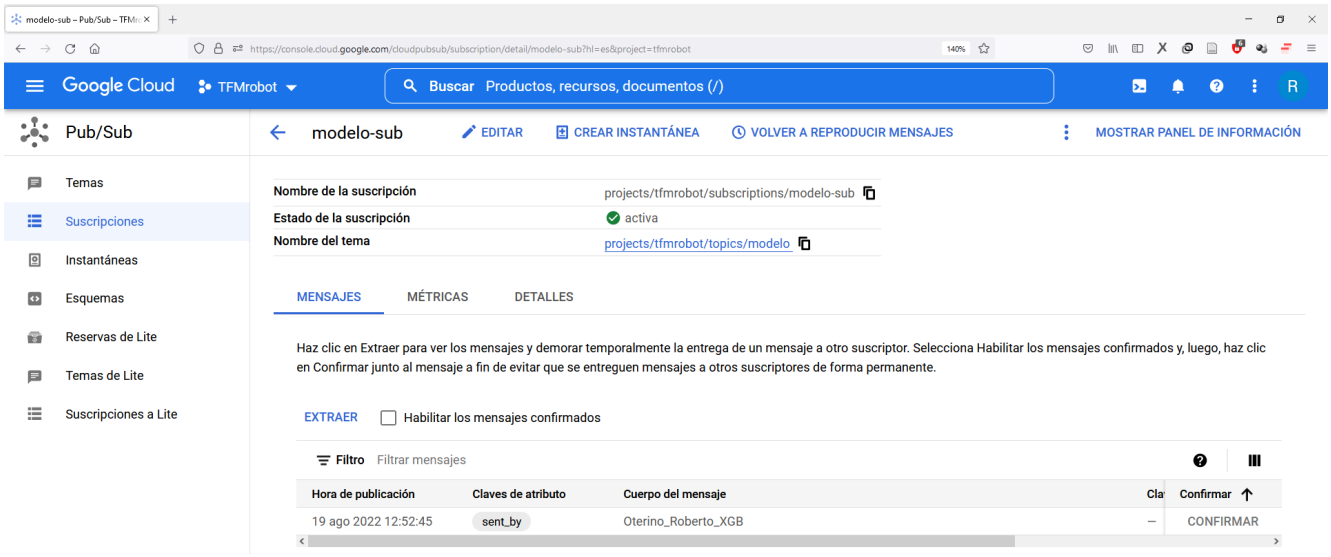

Fig. 94 Prueba mensajes publicados tema modelo.

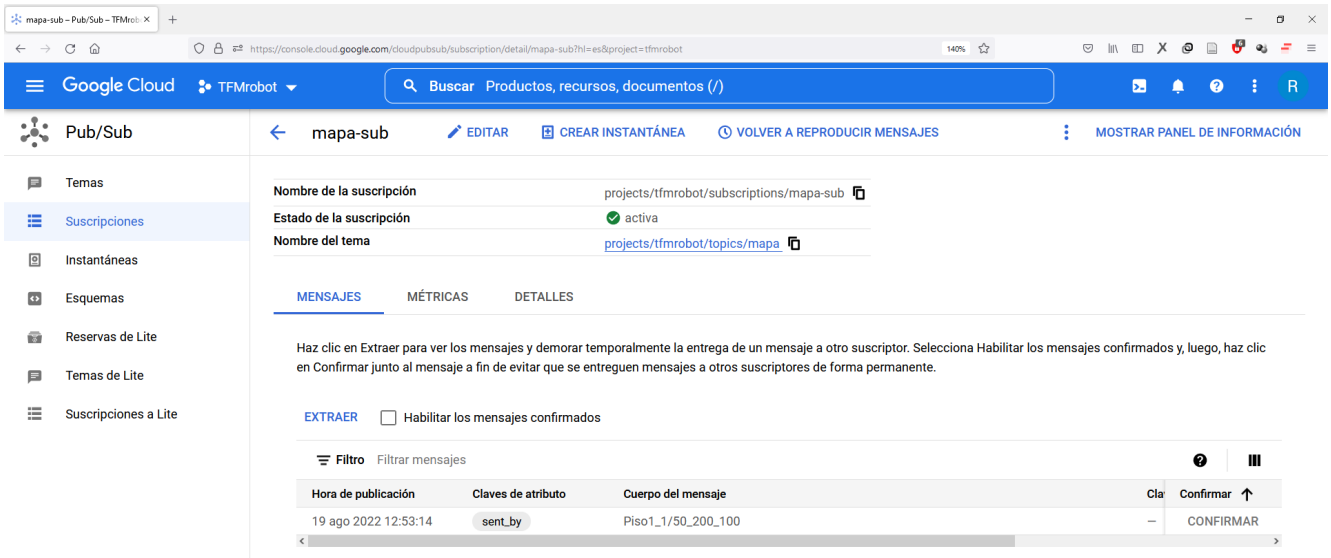

Fig. 95 Prueba mensajes publicados tema mapa.

Por último, se comprueba que los suscriptores reciben el mensaje publicado en los temas y los procesan tal y como se ve en las Fig. 96 y Fig. 97.

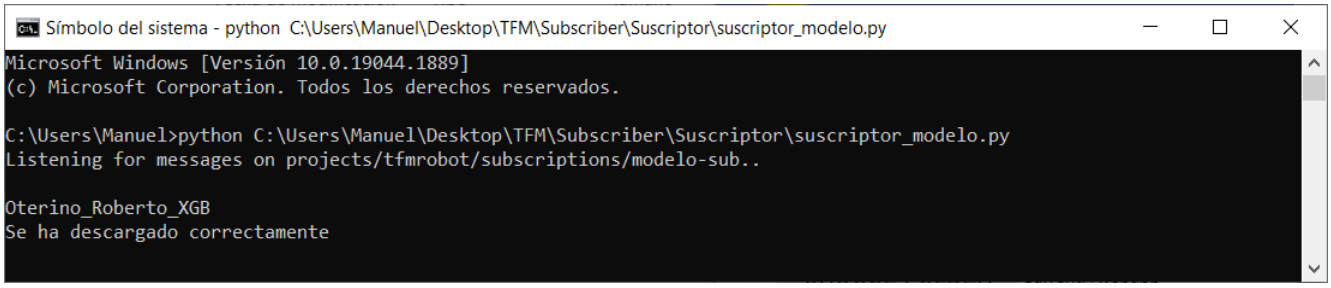

Fig. 96 Prueba suscripción modelo.

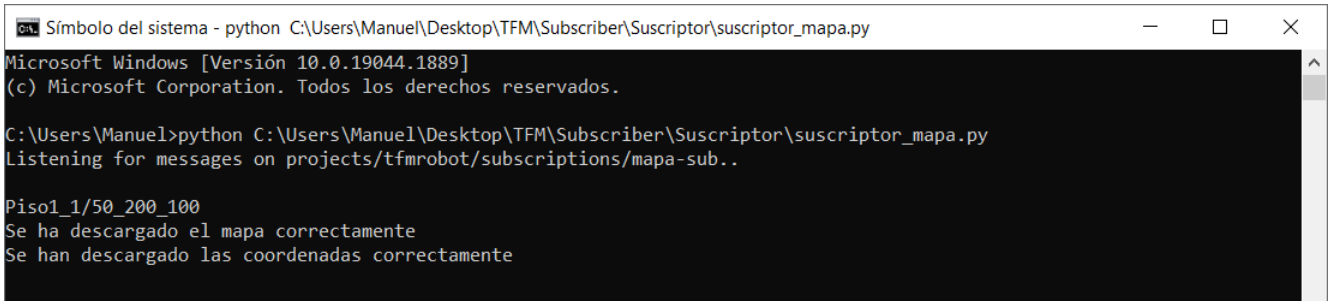

Fig. 97 Prueba suscripción mapa.

4.1.6 Pruebas de navegación y resultados.

Primero hay que comprobar cómo se comporta el controlador MD49. Se ha realizado una rotación del robot sobre sí mismo. Como se puede ver en la Fig. 98 la odometría que se visualiza en rviz es opuesta a la pose del robot. Esto indica que el sistema de tracción del robot está puesto al revés. La solución más sencilla que se ha aplicado es invertir las variables de la lectura del encóder derecho e izquierdo que se utilizan para calcular la odometría del robot en el controlador MD49.

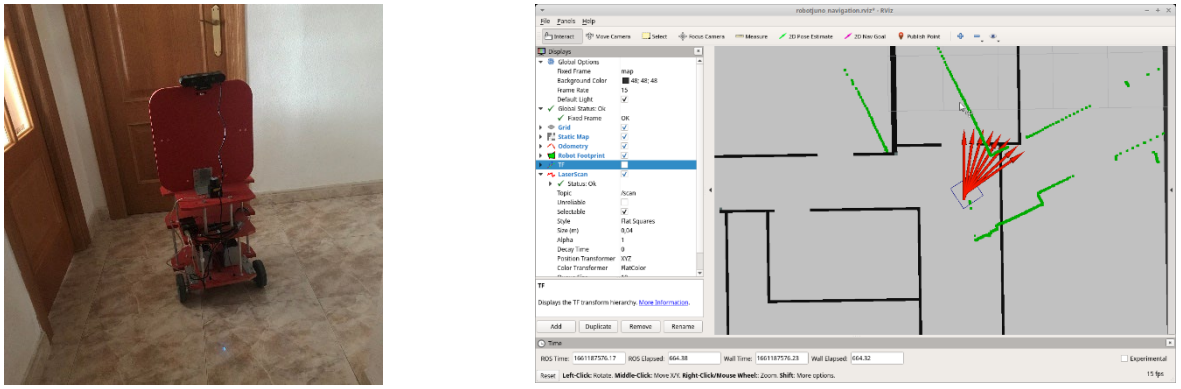

Fig. 98 Prueba rotación robot.

En segundo lugar, partiendo de una posición, se ha iniciado el movimiento en línea recta.

Fig. 99 Prueba movimiento línea recta vista robot.

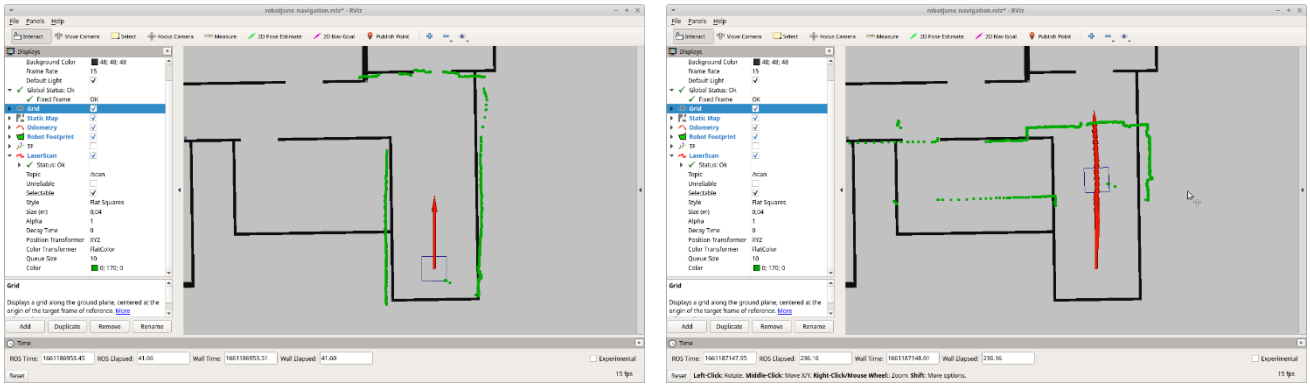

Fig. 100 Prueba movimiento línea recta vista rviz.

Como se observa en las Fig. 99 y Fig. 100 el robot real avanza más, lo que indica que los encóders no son capaces de ofrecer una buena lectura por lo que ofrecen un cálculo incorrecto de la odometría.

Para comprobar que se produce ese problema se ha realizado también un giro del robot sobre sí mismo.

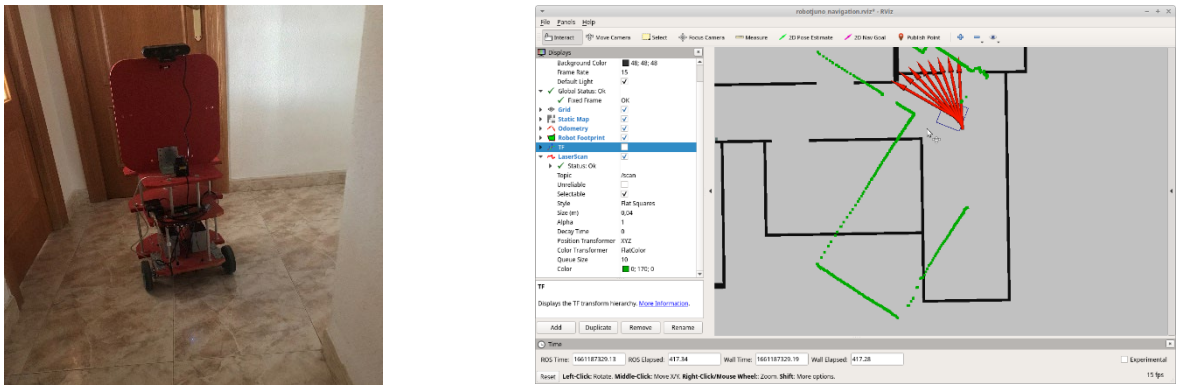

Fig. 101 Prueba 2 rotación robot.

De nuevo se comprueba que el giro del robot es mayor que la información que proporciona la odometría tal y como se ve en la Fig. 101.

La solución que se aplicado en esta ocasión es aumentar el radio de la rueda en el controlador MD49. Con ello se consigue que la longitud que interpreta el robot por vuelta sea mayor traduciéndose en una mejora de la odometría como se puede ver en las Fig. 102 y Fig. 103.

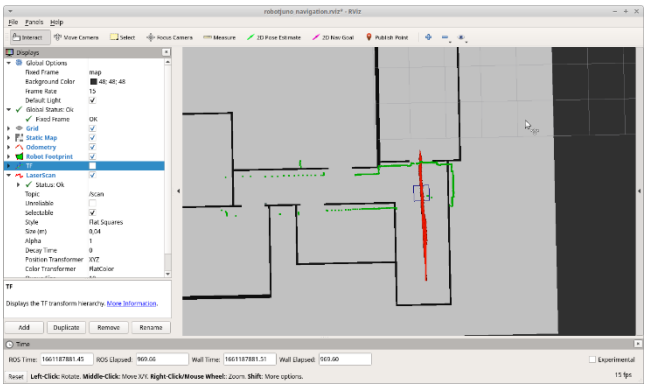

Fig. 102 Prueba movimiento línea recta odometría corregida.

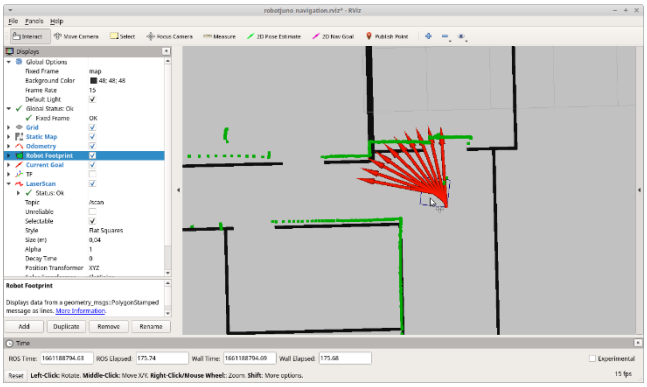

Fig. 103 Prueba rotación odometría corregida.

Ahora ya se pueden realizar las pruebas de navegación. Como se observa en la Fig. 104, partiendo de una posición iniciar, el robot es capaz de alcanzar la meta objetivo. Sin embargo, el sistema de navegación es deficiente y en muchas ocasiones no es capaz de trazar una ruta, sobre todo cuando existen muchos obstáculos o avanza por lugares estrechos.

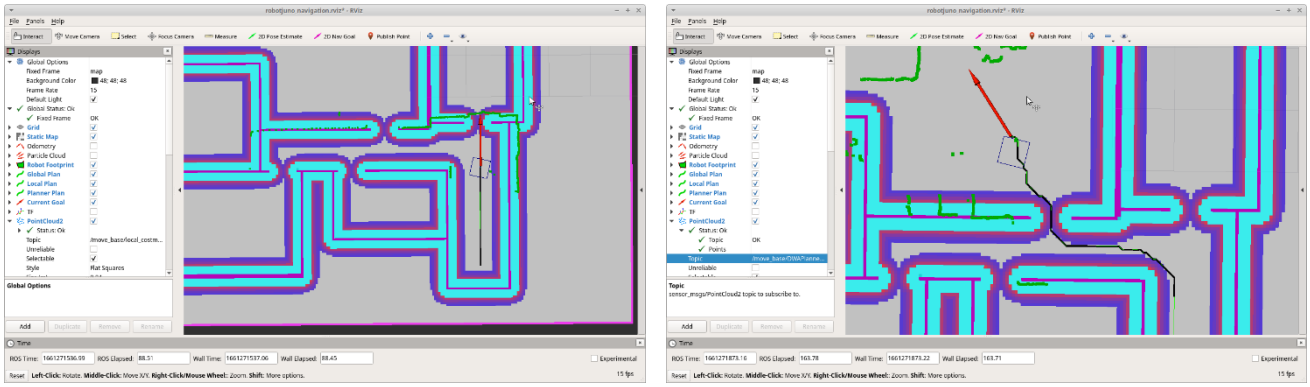

Fig. 104 Pruebas navegación robot.

4.1.7 Pruebas de integración y resultados.

Previamente, en el apartado 4.1.5 se ha mostrado el diseño de la página web y cómo es posible descargar las configuraciones en el robot.

En cuanto al ecosistema ROS se ha realizado dos pruebas. La primera consiste en comprobar que todos los elementos del robot están relacionados, es decir, que existan las transformaciones de coordenadas de los diferentes *frames*. La segunda consiste en visualizar las relaciones de los nodos del despliegue completo del sistema.

El resultado de ambas pruebas se pueden ver en las Fig. 105 y Fig. 106.

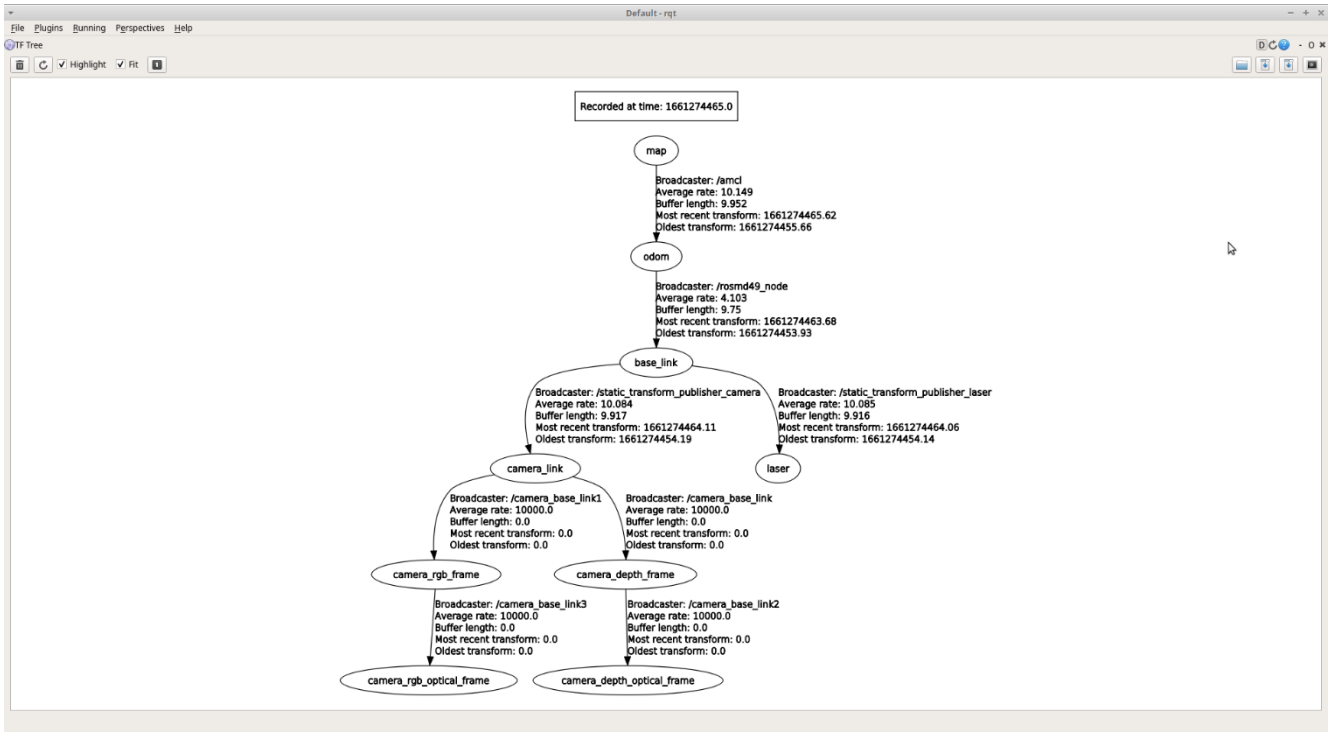

Fig. 105 Esquema gráfico de las transformaciones de los sistemas de referencia del robot.

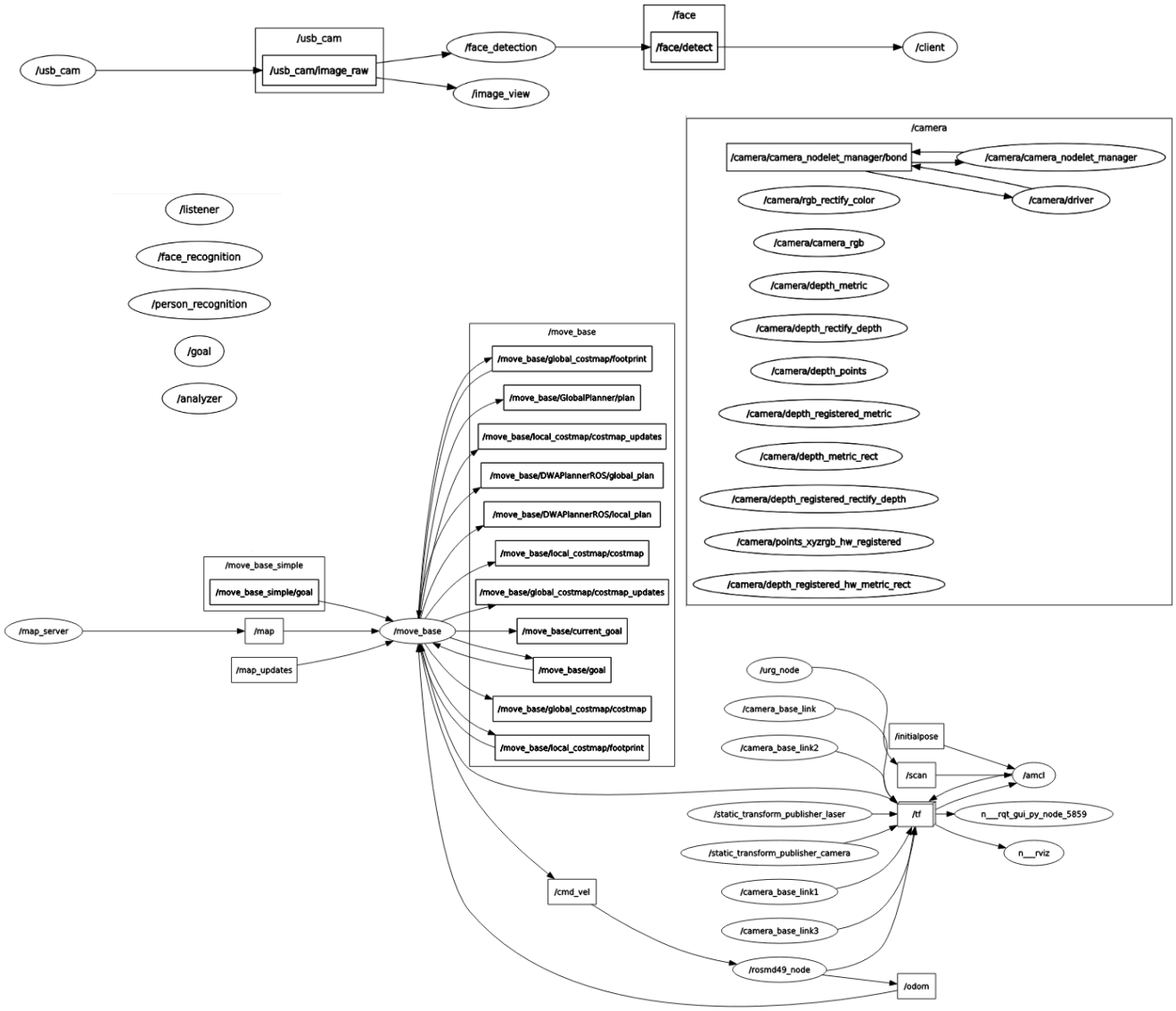

Fig. 106 Esquema gráfico de los nodos y topics activos.

4.2 Discusión.

Una vez comprobado todos los resultados, se puede afirmar que se ha conseguido logar el objetivo de aumentar la robustez de los módulos que permiten la interacción humano robot y de poder desplegar configuraciones en el robot de forma remota.

A continuación, se describen los principales beneficios y limitaciones del sistema:

Beneficios:

- El nuevo sistema de reconocimiento facial se puede reentrenar de una forma sencilla. Además, conforme aumente el número de personas a identificar, se pueden estudiar cuál de las soluciones propuestas es óptima.

- El modelo de reconocimiento de voz ha superado con creces las expectativas. Se ha conseguido un sistema mucho más robusto y mucho más ligero.
- El nuevo sistema de procesamiento de lenguaje natural permite que se puedan ampliar las instrucciones y las localizaciones que entiende el robot de una forma sencilla.
- La aplicación en la Nube abre un abanico de oportunidades para desarrollar funcionalidades bidireccionales con el robot.
- De nuevo se confirma que ROS permite integrar todos los componentes aun estando en dispositivos diferentes y con versiones distintas.

Limitaciones:

- La odometría que proporciona el robot tiene sus limitaciones por lo que dificulta las tareas de navegación.
- El paquete de navegación de ROS no está pensado para lugares estrechos y/o que tengan muchos obstáculos por lo que no se ha logrado que la navegación autónoma sea 100% efectiva.

5 Conclusiones y trabajo futuro.

Para concluir el trabajo, se realizará en este capítulo un pequeño resumen de los objetivos alcanzados tras el desarrollo del mismo. Por último, se describirán nuevas líneas de trabajo que permiten mejorar las limitaciones y desarrollar nuevas funcionalidades.

5.1 Conclusiones.

Partiendo del trabajo realizado anteriormente [1] se ha desarrollado un nuevo sistema de interacción humano robot que elimina las limitaciones que presentaba. En cuanto al sistema de reconocimiento facial se han propuesta varias alternativas para evitar que se realizase de forma empírica. Las alternativas hacen uso de métodos de aprendizaje profundo (DNN) y aprendizaje supervisado (KNN, SVM y XGBoost) cuya inferencia permite que la clasificación se produzca de forma objetiva. Se han alcanzado niveles de confianza cercanos al 100%.

Respecto al sistema de reconocimiento de voz se han validado dos nuevas soluciones. Con una de ellas (*Silero*) se han obtenido resultados similares. Sin embargo, con el segundo (*Vosk*) se ha aumentado la fiabilidad total del sistema tras comprobar su buen funcionamiento, ya que era el mayor punto débil. También se ha desarrollado un sistema de procesamiento de lenguaje natural escalable y fácil de usar gracias a los métodos de búsqueda de patrones que proporciona la nueva librería utilizada (*Spacy*).

Por otro lado, gracias al desarrollo de la aplicación en la Nube de Google, se ha demostrado la gran capacidad y funcionalidades que disponen este tipo de soluciones. Además, al haber diseñado una página web se consigue que cualquier usuario pueda consumir los servicios de forma sencilla e intuitiva. También se ha mostrado cómo es posible comunicarse de forma remota con el robot y que consuma los servicios desarrollados.

En cuando a la navegación se han sufrido problemas derivados de la odometría que proporcionaba el propio controlador motriz del robot y del paquete de navegación de ROS. Sin embargo, se han solventado parcialmente para conseguir que se desplazase el robot de forma autónoma.

Finalmente, no se ha podido desplegar todo el sistema en único dispositivo, sin embargo ha permitido enseñar la capacidad de un ecosistema distribuido como es ROS. Se ha integrado todo el sistema en dos dispositivos y enseñando el funcionamiento completo del mismo.

En definitiva, se ha alcanzado el principal objetivo de este trabajo: desplegar todo un sistema de interacción de una persona con un robot que dispone de navegación autónoma y proporcionarle funcionalidades en la Nube.

5.1.1 Trabajo futuro.

Se proponen varias líneas de investigación en cuanto a trabajos futuros, como primera opción se plantea aumentar las funcionalidades desplegadas en la Nube. Por ejemplo, se puede desarrollar un sistema que permita ver el estado de carga del robot desde una interfaz web. Otra línea de trabajo es migrar la aplicación desarrollada a Android.

Por otro lado, los nuevos sistemas de reconocimiento de voz y procesamiento de lenguaje natural pueden ser reentrenados. Por ello se propone desarrollar una estructura que almacene los datos correctos durante el uso del robot. Cada cierto período de tiempo que se considere oportuno se pueden entrenar automáticamente los modelos y sustituir en el sistema. De este modo, se puede conseguir aumentar la robustez de la aplicación de una forma automática.

Además, sería interesante trasladar el entrenamiento de reconocimiento facial a la Nube. Se recomienda almacenar los tensores que se obtienen aplicando el algoritmo de detección facial *FaceNet* sobre las imágenes para evitar problemas relacionados con la ley de protección de datos en el caso de que se almacenasen las imágenes directamente.

Por último, se propone mejorar la navegación autónoma. En este caso una posible solución, además de mejorar el controlador del robot por ejemplo incorporando una IMU, es generar un mapa semántico para que el robot pueda ubicarse identificando elementos característicos.

6 Bibliografía.

[1] «tfg-ote-dis.pdf». Accedido: 24 de septiembre de 2021. [En línea]. Disponible en: https://repositorio.upct.es/bitstream/handle/10317/9278/tfg-ote-dis.pdf?sequence=1&isAllowed=y

[2] «Professional Service Robots Resources & Education | RIA», *Robotics Online*. https://www.robotics.org/service-robots (accedido 7 de enero de 2021).

[3] K. Li, J. Wu, X. Zhao, y M. Tan, «Real-Time Human-Robot Interaction for a Service Robot Based on 3D Human Activity Recognition and Human-Mimicking Decision Mechanism», en *2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, Tianjin, China, jul. 2018, pp. 498-503. doi: 10.1109/CYBER.2018.8688272.

[4] «Mell y Grance - The NIST Definition of Cloud Computing.pdf». Accedido: 20 de agosto de 2022. [En línea]. Disponible en: http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

[5] P. P, D. K. G., Yaazhlene. P, M. Venkata Ganesh, y V. B, «Fog Computing: Issues, Challenges and Future Directions», *Int. J. Electr. Comput. Eng. IJECE*, vol. 7, n.º 6, p. 3669, dic. 2017, doi: 10.11591/ijece.v7i6.pp3669-3673.

[6] «Shi et al. - 2016 - Edge Computing Vision and Challenges.pdf».

[7] «Intel® Neural Compute Stick 2 Especificaciones de productos», *Intel*. https://www.intel.es/content/www/es/es/products/sku/140109/intel-neural-compute-stick-2/specifications.html (accedido 20 de agosto de 2022).

[8] «USB Accelerator», *Coral*. https://coral.ai/products/accelerator/ (accedido 20 de agosto de 2022).

[9] «Sistemas integrados NVIDIA para las máquinas autónomas de la próxima generación», *NVIDIA*. https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/ (accedido 20 de agosto de 2022).

[10] «All Robots - ROBOTS: Your Guide to the World of Robotics». https://robots.ieee.org/robots/ (accedido 20 de agosto de 2022).

[11] «Home page», *Misty Robotics.* https://www.mistyrobotics.com/ (accedido 20 de agosto de 2022).

[12] «NAO le robot humanoïde et programmable | SoftBank Robotics». https://www.softbankrobotics.com/emea/es/nao (accedido 20 de agosto de 2022).

[13] «Asimo - Honda.mx», *HONDA de México*. http://www.honda.mx/asimo (accedido 20 de agosto de 2022).

[14] H. Ltd, «EMIEW3 Development – Robots aiming to Live with Humans : Research & Development : Hitachi». https://www.hitachi.com/rd/sc/story/emiew3/index.html (accedido 20 de agosto de 2022).

[15] «tfg-san-pue.pdf». Accedido: 10 de septiembre de 2022. [En línea]. Disponible en: https://repositorio.upct.es/bitstream/handle/10317/9454/tfg-san-pue.pdf?sequence=1

[16] «EMG49 data». https://www.robot-electronics.co.uk/htm/emg49.htm (accedido 23 de agosto de 2022).

[17] «MD49 technical documentation». Accedido: 23 de agosto de 2022. [En línea]. Disponible en: https://www.robot-electronics.co.uk/htm/md49tech.htm

[18] «URG-04LX-UG01 | Products List | Discontinued | Scanning Rangefinder | Distance Data Output | URG-04LX-UG01», *HOKUYO AUTOMATIC CO., LTD.* https://www.hokuyo-aut.jp/search/single.php?serial=166 (accedido 23 de agosto de 2022).

[19] «hokuyo_node - ROS Wiki». Accedido: 23 de agosto de 2022. [En línea]. Disponible en: http://wiki.ros.org/hokuyo_node

[20] «Snapshot». Accedido: 23 de agosto de 2022. [En línea]. Disponible en: https://shop.orbbec3d.com/Astra-Pro-Plus

[21] *astra_camera*. Orbbec, 2022. Accedido: 23 de agosto de 2022. [En línea]. Disponible en: https://github.com/orbbec/ros_astra_camera

[22] «TPS5430EVM-173 Evaluation board | TI.com». https://www.ti.com/tool/TPS5430EVM-173 (accedido 23 de agosto de 2022).

[23] «imgaug — imgaug 0.4.0 documentation». https://imgaug.readthedocs.io/en/latest/ (accedido 11 de agosto de 2022).

[24] «pandas - Python Data Analysis Library». https://pandas.pydata.org/ (accedido 11 de agosto de 2022).

[25] K. Team, «Keras documentation: Layer activation functions». https://keras.io/api/layers/activations/#relu-function (accedido 10 de agosto de 2022).

[26] K. Team, «Keras documentation: SGD». https://keras.io/api/optimizers/sgd/ (accedido 10 de agosto de 2022).

[27] K. Team, «Keras documentation: Adam». https://keras.io/api/optimizers/adam/ (accedido 10 de agosto de 2022).

[28] «Snapshot». Accedido: 10 de agosto de 2022. [En línea]. Disponible en: https://www.nvidia.com/en-us/glossary/data-science/xgboost/

[29] J. B. M. Vega, «Tutorial: XGBoost en Python», *Medium*, 24 de enero de 2022. https://medium.com/@jboscomendoza/tutorial-xgboost-en-python-53e48fc58f73 (accedido 10 de agosto de 2022).

[30] «User guide: contents», *scikit-learn*. https://scikit-learn/stable/user_guide.html (accedido 11 de agosto de 2022).

[31] «Classification Using Nearest Neighbors - MATLAB & Simulink - MathWorks España». https://es.mathworks.com/help/stats/classification-using-nearest-neighbors.html (accedido 10 de agosto de 2022).

[32] «Support Vector Machine (SVM) - MATLAB & Simulink». https://es.mathworks.com/discovery/support-vector-machine.html (accedido 10 de agosto de 2022).

[33] R. Taylor, «What is WER? What Does Word Error Rate Mean?», *Rev*. https://www.rev.com/blog/resources/what-is-wer-what-does-word-error-rate-mean (accedido 9 de agosto de 2022).

[34] «Ali y Renals - 2018 - Word Error Rate Estimation for Speech Recognition.pdf». Accedido: 9 de agosto de 2022. [En línea]. Disponible en: https://www.pure.ed.ac.uk/ws/portalfiles/portal/63902419/ewer_acl2018.pdf

[35] «Papers with Code - MediaSpeech Benchmark (Speech Recognition)». https://paperswithcode.com/sota/speech-recognition-on-mediaspeech (accedido 9 de agosto de 2022).

[36] «PyTorch». https://www.pytorch.org (accedido 10 de agosto de 2022).

[37] A. Veysov, *Silero Models*. 2021. Accedido: 24 de septiembre de 2021. [En línea]. Disponible en: https://github.com/snakers4/silero-models

[38] «PyAudio: Cross-platform audio I/O for Python, with PortAudio». https://people.csail.mit.edu/hubert/pyaudio/ (accedido 11 de agosto de 2022).

[39] «VOSK Offline Speech Recognition API», *VOSK Offline Speech Recognition API*. https://alphacephei.com/vosk/ (accedido 24 de septiembre de 2021).

[40] «Model adaptation for VOSK», *VOSK Offline Speech Recognition API*. https://alphacephei.com/vosk/adaptation (accedido 10 de agosto de 2022).

[41] «VOSK Models», *VOSK Offline Speech Recognition API*. https://alphacephei.com/vosk/models (accedido 11 de agosto de 2022).

[42] «spaCy 101: Everything you need to know · spaCy Usage Documentation», *spaCy 101: Everything you need to know*. https://spacy.io/usage/spacy-101#whats-spacy (accedido 10 de agosto de 2022).

[43] «spaCy 101: Everything you need to know · spaCy Usage Documentation», *spaCy 101: Everything you need to know*. https://spacy.io/usage/spacy-101#features (accedido 10 de agosto de 2022).

[44] «Universal Dependencies». https://universaldependencies.org/ (accedido 12 de agosto de 2022).

[45] «spaCy 101: Everything you need to know · spaCy Usage Documentation», *spaCy 101: Everything you need to know*. https://spacy.io/usage/spacy-101#training (accedido 10 de agosto de 2022).

[46] «spaCy 101: Everything you need to know · spaCy Usage Documentation», *spaCy 101: Everything you need to know*. https://spacy.io/usage/spacy-101#pipelines (accedido 10 de agosto de 2022).

[47] «Spanish · spaCy Models Documentation», *Spanish*. https://spacy.io/models/es (accedido 12 de agosto de 2022).

[48] «Matcher · spaCy API Documentation», *Matcher*. https://spacy.io/api/matcher (accedido 12 de agosto de 2022).

[49] «tf - ROS Wiki». Accedido: 24 de agosto de 2022. [En línea]. Disponible en: http://wiki.ros.org/tf#static_transform_publisher

[50] S. Macenski, *Spatio-Temporal Voxel Layer*. 2022. Accedido: 24 de agosto de 2022. [En línea]. Disponible en: https://github.com/SteveMacenski/spatio_temporal_voxel_layer

[51] S. Macenski, D. Tsai, y M. Feinberg, «Spatio-temporal voxel layer: A view on robot perception for the dynamic world», *Int. J. Adv. Robot. Syst.*, vol. 17, n.º 2, p. 172988142091053, mar. 2020, doi: 10.1177/1729881420910530.

[52] «¿Qué es IaaS (infraestructura como servicio)?», *Google Cloud*. https://cloud.google.com/learn/what-is-iaas?hl=es (accedido 13 de agosto de 2022).

[53] «¿Qué es PaaS? | Google Cloud», *Google Cloud*. https://cloud.google.com/learn/what-is-paas?hl=es-419 (accedido 13 de agosto de 2022).

[54] «Software como servicio (SaaS)», *Google Cloud*. https://cloud.google.com/saas?hl=es (accedido 13 de agosto de 2022).

[55] «Documentación de Cloud○Endpoints | Cloud Endpoints | Google Cloud». https://cloud.google.com/endpoints/docs?hl=es-419 (accedido 13 de agosto de 2022).

[56] S. M. Talim, «Go, Cloud Endpoints and App Engine», *Google Cloud - Community*, 21 de septiembre de 2015. https://medium.com/google-cloud/go-cloud-endpoints-and-app-engine-19d290dafda3 (accedido 13 de agosto de 2022).

[57] «Datastore», *Google Cloud*. https://cloud.google.com/datastore?hl=es-419 (accedido 13 de agosto de 2022).

[58] «Cloud SQL para PostgreSQL, MySQL y SQL Server | Cloud SQL: Servicio de bases de datos relacionales | Google Cloud». https://cloud.google.com/sql?hl=es-419 (accedido 13 de agosto de 2022).

[59] «Cloud Storage | Google Cloud». https://cloud.google.com/storage?hl=es-419 (accedido 13 de agosto de 2022).

[60] «Home · objectify/objectify Wiki · GitHub». https://github.com/objectify/objectify/wiki (accedido 13 de agosto de 2022).

[61] «¿Qué es Pub/Sub? | Documentación de Cloud Pub/Sub | Google Cloud». https://cloud.google.com/pubsub/docs/overview?hl=es-419 (accedido 13 de agosto de 2022).

[62] «Guía de inicio rápido | Cloud Tools for Eclipse | Google Cloud». https://cloud.google.com/eclipse/docs/quickstart?hl=es-419 (accedido 16 de agosto de 2022).

[63] M. O. contributors Jacob Thornton, and Bootstrap, «Bootstrap». https://getbootstrap.com/ (accedido 13 de agosto de 2022).

[64] «Bootstrap: ¿qué es, para qué sirve y cómo instalarlo?», *Rock Content - ES*, 12 de abril de 2020. https://rockcontent.com/es/blog/bootstrap/ (accedido 13 de agosto de 2022).

[65] «Integración del inicio de sesión de Google en su aplicación web | Google Sign-In for Websites», *Google Developers*. https://developers.google.com/identity/sign-in/web/sign-in?hl=es-419 (accedido 16 de agosto de 2022).

[66] «Migra a los Servicios de identidad de Google | Google Identity Services JavaScript SDK», *Google Developers*. https://developers.google.com/identity/oauth2/web/guides/migration-to-gis?hl=es-419 (accedido 16 de agosto de 2022).

[67] «Migrar desde Acceso con Google | Sign In With Google», *Google Developers*. https://developers.google.com/identity/gsi/web/guides/migration?hl=es-419 (accedido 16 de agosto de 2022).

[68] «FileSaver.js/LICENSE.md at master · eligrey/FileSaver.js», *GitHub*. https://github.com/eligrey/FileSaver.js (accedido 16 de agosto de 2022).

[69] «Pub/Sub client libraries | Cloud Pub/Sub Documentation», *Google Cloud*. https://cloud.google.com/pubsub/docs/reference/libraries (accedido 17 de agosto de 2022).

[70] «map_server - ROS Wiki». http://wiki.ros.org/map_server (accedido 17 de agosto de 2022).

[71] «ROS/Tutorials/MultipleMachines - ROS Wiki». Accedido: 24 de agosto de 2022. [En línea]. Disponible en: http://wiki.ros.org/ROS/Tutorials/MultipleMachines

[72] «Visual Studio Code - Code Editing. Redefined». https://code.visualstudio.com/ (accedido 24 de agosto de 2022).

[73] «Jupyter Notebooks in Visual Studio Code». https://code.visualstudio.com/learn/educators/notebooks (accedido 24 de agosto de 2022).

[74] «Postman API Platform | Sign Up for Free», *Postman*. https://www.postman.com/ (accedido 18 de agosto de 2022).

[75] «Web Server for Chrome». https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb (accedido 19 de agosto de 2022).