industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Puesta a punto de un robot real autónomo inteligente usando ROS.

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: Adrián Sánchez Granero
Directora: Dra. Nieves Pavón Pulido
Codirector: Dr. Juan Antonio López Riquelme

Cartagena, a 2 de Junio de 2021

Universidad Politécnica de Cartagena

Agradecimientos

A mis padres y mi hermana por aguantarme pese a estar todo el día trasteando con el robot, unas veces con mejores resultados que otras. Agradecer también a la Doctora Nieves Pavón por darme la oportunidad de cursar su optativa Robótica Móvil pese a no corresponder con mi grado, además de brindarme la oportunidad de hacer un Trabajo Fin de Grado de carácter práctico, algo que en estos tiempos de pandemia se agradece.

RESUMEN

Este trabajo aborda la puesta a punto de un robot móvil autónomo real, tanto a nivel de hardware como a nivel de software y firmware. Para ello se empleará una plataforma ya existente que se actualizará, mejorará y configurará para que se pueda implementar una arquitectura de software distribuida basada en ROS, que permita al robot móvil navegar de forma autónoma, además de proporcionarle habilidades de visión artificial basadas en técnicas de Machine Learning. El sistema se ha probado tanto en simulación como en un entorno real, demostrándose, así, que es posible crear y validar diferentes componentes software en el ecosistema de ROS, de forma muy sencilla, generando una arquitectura de software fuertemente desacoplada.

Además, de validar la propia plataforma real desarrollada, se han integrado componentes de interacción natural con el robot realizados en otros trabajos, que sólo se habían probado en un contexto de simulación.

Para mejorar la capacidad de interacción natural del robot con las personas, se ha mejorado un reconocedor de personas, mediante la identificación automática de la cara de dicha persona. El robot puede aprender a reconocer a cada usuario a través de un modelo específico que permite una clasificación binaria (es o no es). Esta metodología permite que el robot pueda aprender a reconocer nuevas personas, simplemente, aprendiendo un nuevo modelo a partir de un conjunto de imágenes tomadas por la cámara instalada en el robot. El proceso de aprendizaje se lleva a cabo en un elemento de proceso fuera de la plataforma, y el modelo es incluido manualmente para que el robot pueda realizar el proceso de inferencia de manera local.

Todos los componentes desarrollados e incluidos se han validado adecuadamente mediante un conjunto de tests que se presentan de forma detallada en este trabajo.

Índice

Figuras.

1 Introducción.

En el presente capítulo se describen las motivaciones que han llevado a la realización de este trabajo enfocado en la construcción, desarrollo y validación de un robot móvil inteligente autónomo en el ámbito doméstico con capacidad para interaccionar con humanos de forma natural. Se describen los principales problemas existentes relacionados con los objetivos a tratar para el montaje, puesta en marcha y funcionamiento de dicho robot, permitiendo la interacción usuario-robot de una forma cómoda y accesible. También se resumen los contenidos de los distintos capítulos, para mayor facilidad de lectura del trabajo si se quisiera emplear como guía.

1.1 Motivación y objetivos.

Normalmente, cuando se habla de Robótica Móvil, se suele pensar en robots de carácter militar, industrial, coches autónomos o drones de medio/gran tamaño. Esto es debido a que esta rama de la industria parece tener un carácter esencialmente técnico, lo cual hace difícil el imaginarlo en un contexto cotidiano. Sin embargo, ya existen diversos robots inteligentes autónomos en el contexto de la Robótica de Servicios, concretamente en lo referente a los escenarios domésticos. Los conocidos como robots domésticos están destinados a realizar, o al menos facilitar, las tareas del hogar, así como proporcionar apoyo o ayuda al usuario. Debido a que el presente trabajo está destinado a relacionar la Robótica Móvil con el ámbito doméstico, dejaremos de lado los robots de hogar con carácter estático como los robots de cocina. Hablamos pues de cortadoras de césped autónomas, robots limpiadores de piscinas, friegasuelos y los más conocidos, los robots aspiradora. No parece atrevido decir que, cuando se habla con alguien sobre tener un robot en casa hoy en día, lo primero en lo que se piensa es en un electrodoméstico tipo *Roomba*. En la Figura 1 se muestra la tendencia de la evolución de los robots de servicio, que tendrían como meta un robot humanoide, pero esta meta no tiene por qué ser la más acertada o aceptada.

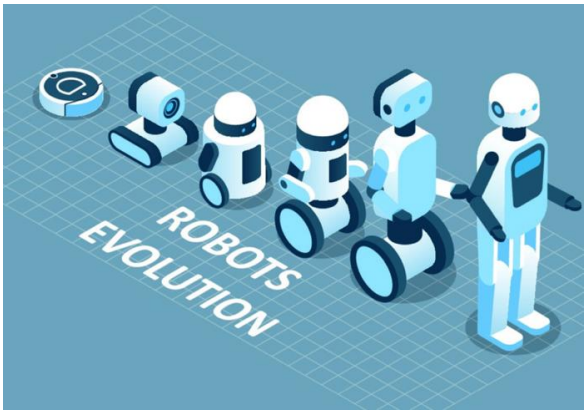


Figura 1. Evolución de los robots de servicio según AER Automation.

Sin embargo, la reciente situación de pandemia global ha demostrado que estos robots mantienen de manera muy eficiente el hogar, pero descuidan el tema humano. Durante la cuarentena, aquellos que no tuvieron la suerte de confinarse con familia o compañeros de piso, sufrieron un gran aislamiento. Este hecho fue más grave en las personas mayores, ya que quedaron en una posición más vulnerable al no poder recibir visitas, no sólo familiares sino de ayuda en su día a día, por ejemplo, de un asistente. Y no sólo las personas mayores se ven afectadas, la soledad y el aislamiento afectan a todos en mayor o menor medida. Esto nos hizo considerar el potencial de un robot doméstico que no cuide del hogar, sino de los integrantes del mismo.

El primer problema era la movilidad. El interior de una casa está repleto de esquinas, recovecos y muebles, lo que dificulta la movilidad del robot. Para abordar este problema es posible basarse en los robots aspiradora.

Por otra parte, un gran problema es el tema de la comunicación robot-usuario. Para la gente joven no resulta difícil dar órdenes o recibir información del robot a través de una aplicación para teléfono o Tablet, pero dado que se busca que llegue al mayor número de gente posible, hay que considerar también a la gente mayor. Por ello sería deseable encontrar una forma más “natural” de interaccionar con el robot, como los comandos de voz o el reconocimiento facial.

Con este proyecto se quiere demostrar que, con la tecnología existente, se puede construir un prototipo de robot doméstico capaz de interactuar de una forma cómoda y natural con personas. Este trabajo tratará la adaptación de tecnologías y metodologías existentes, así como el montaje, puesta en marcha y funcionamiento de un prototipo.

El objetivo principal, por tanto, es la puesta a punto de un robot real autónomo inteligente a nivel de hardware, software y firmware. Para logarlo se requiere cumplir una serie de subobjetivos específicos:

- Montaje del chasis e instalación de sensores y actuadores. El tamaño del robot debe adecuarse al espacio libre para la circulación de que dispone una vivienda y los sensores deben colocarse y orientarse para su funcionamiento en interiores.
- Desarrollo de una arquitectura de software de alto nivel para permitir la navegación autónoma del robot. El robot ha de ser capaz de calcular una trayectoria para llegar a un objetivo y ser capaz de alcanzarlo sin incidencias.
- Desarrollo de pruebas específicas para implementar y evaluar los sensores de visión. De esta forma se comprueba como robot y usuario van a interactuar.
- Incorporación de algoritmos de reconocimiento y síntesis de voz, diseñados en otro trabajo, pero que fueron solamente probados en un entorno de simulación.
- Mejora de un algoritmo de detección de caras, para que el robot pueda reconocer personas.

1.2 Resumen de capítulos.

Introducción. Se realiza una aproximación al tema de los robots domésticos y se analizan los problemas existentes, que llevan a pensar en una posible solución. También se presenta el objetivo principal de construir un prototipo y los subobjetivos necesarios para llevarlo a cabo.

Estado del arte. Se describen los robots reales que se han implantado en entornos domésticos y de interacción con humanos en el ámbito de la Robótica de Servicio. También se describe ROS como marco de trabajo (framework) empleado y las tecnologías disponibles útiles en Robótica de Servicio a nivel de interacción humano-robot.

Diseño del sistema. Se describe el robot en su conjunto y cómo los componentes interactúan para conseguir los objetivos propuestos. Se presenta la arquitectura tanto de hardware y montaje, como de software. Además, se describen los componentes adicionales que se han integrado para realizar el reconocimiento de personas y el reconocimiento y síntesis de voz.

Análisis de resultados. Se explica el conjunto de pruebas que se han diseñado para evaluar cada módulo por separado y la integración de todos los módulos. Asimismo, se comentan los resultados obtenidos junto con las ventajas e inconvenientes del sistema resultante.

Conclusiones y trabajo futuro. Se describen las conclusiones obtenidas del estudio y las pruebas realizadas, así como las principales líneas de estudio futuras que se podrían abordar.

2 Estado del arte.

En el presente capítulo se presenta una introducción a la Robótica de Servicios, así como a los principales robots reales que se han implantado en entornos domésticos y a los métodos actuales usados para interacción robot-humano. Además, se presentará ROS como la herramienta que permitirá desarrollar e implementar el software del robot, comentando qué robots reales en el ámbito de servicios lo emplean.

Por otra parte, se describen las tecnologías disponibles útiles en Robótica de Servicios a nivel de interacción humano-robot, como son el reconocimiento y síntesis de voz y el reconocimiento facial.

2.1 La Robótica de Servicios.

*La **Federación Internacional de Robótica (IFR)** define el concepto de robot de servicio como "**un robot o equipamiento que realiza tareas útiles para personas o equipos, excluyendo aplicaciones de automatización Industrial**"* [1].

Por lo tanto, los robots que se incluyen en esta categoría estarían destinados a facilitar la vida de las personas, ya sea realizando tareas de carácter más arriesgado como la supervisión de túneles o torres de alta tensión, o tareas cotidianas como aspirar el suelo o cortar el césped.

Como se ha comentado con anterioridad, un gran desafío a la hora de desarrollar este tipo de robots está relacionado con el proceso de interacción usuario-robot dado que, si no se poseen ciertos conocimientos de programación o del lenguaje que use el robot en cuestión, será casi imposible para alguien inexperto interactuar con el robot. Precisamente por esto, a continuación, se resumen las principales características de una serie de robots reales implantados en entornos domésticos y se analiza cómo interactúan con usuarios humanos, no necesariamente familiarizados con la Tecnología, en general, y la Robótica, en particular.

2.1.1 Roomba de iRobot.

El dispositivo *Roomba* (ver Figura 2), se puede considerar como el máximo representante de los robots domésticos. Este robot aspirador está equipado con un sensor infrarrojo de proximidad para detectar paredes y obstáculos, un sensor de caída que comprueba constantemente la distancia al suelo para evitar caídas en desniveles, algunos modelos incluso tienen una cámara para que se pueda comprobar si está pasando algo inusual en la estancia en la que se encuentra.

Figura 2. Roomba i3+.

Lo que hace a estos robots tan conocidos es que no se limitan a moverse sin control, evitando obstáculos mientras aspiran el suelo, sino que algunos modelos avanzados crean un mapa de la vivienda. Con este mapa no solo saben cuántas estancias tiene que limpiar, también sabe cuáles ha limpiado.

En cuanto a la interacción con el usuario, mediante la aplicación *iRobot Home*, se puede ver el mapa que ha creado pudiendo editarlo nombrando las diferentes estancias o especificando zonas dentro de las mismas. Con la aplicación, se pueden marcar zonas a las que no se requiere que acceda, como los baños o la terraza. También se pueden establecer rutinas de limpieza, marcando el orden de las habitaciones por las que ha de pasar. Algunos modelos son compatibles con *Alexa*, por lo que mediante comandos de voz se puede indicar a la aspiradora que ruta deseamos que realice, haciendo más cómodo y natural la comunicación con *Roomba* como si de una persona se tratara.

Otros fabricantes han creado robots similares con prestaciones parecidas, por lo que la oferta disponible es, a día de hoy, bastante amplia y bien aceptada por los clientes en todo el mundo.

2.1.2 Alexa de Amazon.

Alexa es el asistente virtual controlado por voz creado por Amazon, y lanzado en noviembre de 2014 junto a su línea de altavoces inteligentes Echo [2]. Es cierto que no se trata de un robot móvil, pero su inteligencia artificial suple esta carencia con una interacción tan natural para las personas como es el lenguaje.

Alexa (ver Figura 3) funciona mediante comandos de voz. Es tan sencillo como decir su nombre y luego especificar lo que se quiere. Gracias a su inteligencia artificial y conectividad con la red, es capaz de informarte

sobre noticias de actualidad, sobre meteorología, poner música de ambiente e incluso es posible hacer compras por *Amazon* usando dicho asistente.

Figura 3. Alexa de Amazon.

Posee una SDK abierta, por lo que ya es compatible con multitud de dispositivos como pueden ser bombillas inteligentes, lavadoras inteligentes, diferentes dispositivos domóticos y como ya se ha mencionado, con robots aspiradora como *Roomba*. Es verdad que un altavoz capaz de decir el tiempo o encender unas bombillas no es precisamente el robot doméstico en el que se podría pensar, sin embargo, su capacidad para comunicarse con el usuario de una forma tan fluida e intuitiva como son los comandos de voz, lo convierten en el ejemplo a seguir en el campo de la interacción humano-robot.

Además de Alexa, otros proveedores de Cloud Computing, como Google, han desarrollado modelos similares, con una funcionalidad y un grado de compatibilidad parecidos.

2.1.3 Robot humanoide Pepper.

"Pepper es el primer humanoide diseñado específicamente para convivir y relacionarse con seres humanos. Puede ***interpretar el estado de ánimo de las personas*** *de su entorno y modificar su comportamiento en base a su programación. Además, su forma de interactuar es completamente intuitiva y amigable puesto que* ***reconoce gestos, sonidos, expresiones y tacto****. La interactividad es el concepto principal en el diseño de Pepper. Sus múltiples sensores táctiles y de sonido, junto a su revolucionaria cámara 3D, le permiten registrar e interpretar de forma detallada el entorno. Además, su pantalla táctil transmite y recibe información y posee conexión on-*

line. Este potente hardware ofrece múltiples opciones a la hora de programar y configurar el robot para que interactúe con su entorno" [3].

Este robot social con aspecto humanoide está ideado para realizar tareas como recepcionista, promotor, punto de información o analista. Su elevado precio, a partir de quince mil euros, así como su sofisticado hardware hacen pensar que no es un robot doméstico, pero si a su capacidad de mantener una conversación le añadiéramos la habilidad de manipular objetos o realizar tareas del hogar, tendríamos el robot doméstico por excelencia. En la Figura 4 se aprecian las diferentes partes que componen a *Pepper*.

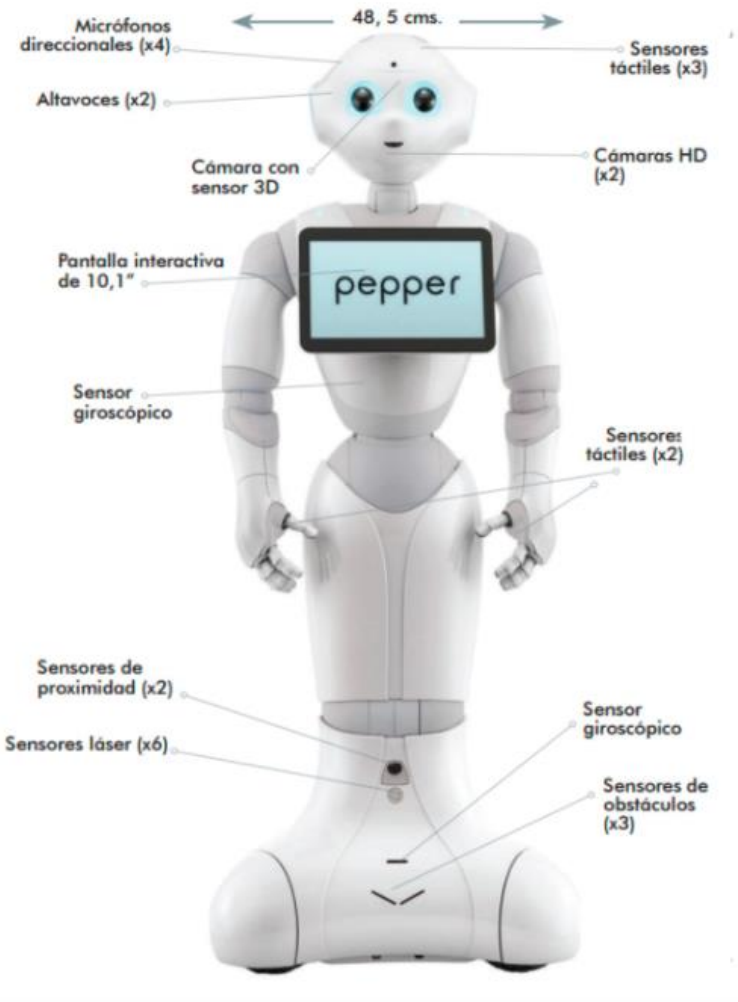


Figura 4. Robot humanoide *Pepper*.

En resumen, para nuestro prototipo tomaremos como ejemplos y meta a estos tres robots, ya que se desea que el robot desarrollado tenga la movilidad y planificación de un *Roomba*, la capacidad de interacción de *Alexa* y la habilidad para comprender e interpretar de *Pepper*, todo esto adaptado a un prototipo que se ajuste a la economía doméstica.

2.2 ROS (Robot Operating System).

"ROS es un entorno de desarrollo de software de código abierto para robots. Provee de servicios que se esperarían de un sistema operativo, incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comunes, pasaje de mensajes entre procesos y manejo de paquetes. También brinda herramientas y librerías para obtener, construir, escribir y correr código a través y mediante varias computadoras" [4].

Pese a no ser un sistema operativo como tal, ROS permite integrar los diferentes componentes del robot, programarlos, comunicarlos entre sí y monitorizarlos. Gracias a su funcionalidad basada en nodos y paquetes, se puede programar cada módulo por separado (un paquete con nodos para la navegación, otro para la visión, etc.) y lanzarlos todos con un mismo archivo que llama a los nodos implementados en cada paquete, pudiendo usar un visualizador o la terminal de comandos para ver los resultados o acciones que se están desarrollando, además de poder dar órdenes nuevas.

ROS está soportado en plataformas basadas en UNIX, por lo que Linux sería la opción más adecuada para usarlo. Al ser tanto Linux como ROS de software libre, permite a sus desarrolladores el innovar, crear e implementar paquetes y librerías, que se ponen al servicio de la comunidad de forma libre. Actualmente ROS es compatible con Python y C++, entre otros lenguajes, lo que significa que si se poseen conocimientos de programación es fácil adaptarse a cualquiera de estos lenguajes.

ROS se emplea sobre todo en robots con carácter educativo o de investigación, sin embargo, hay diversos robots que se comercializan y que emplean ROS.

2.2.1 Pioneer 3DX.

Pioneer 3DX (ver Figura 5) es un robot con configuración diferencial de dos ruedas motrices y una rueda loca (tipo castor). El robot cuenta con un array de dispositivos de ultrasonido (sonar) frontal, batería, encoders para las ruedas, un microcontrolador con el firmware ARCOS y el paquete de desarrollo de software de robots móviles Pioneer SDK (Software Development Kit).

Figura 5. Pioneer 3DX.

Es totalmente compatible con ROS e incluso ya existen diversos paquetes de ROS para este modelo. Es un robot de carácter genérico destinado a educación e investigación, por lo que no tiene un propósito definido. Se le pueden añadir diversos componentes y programarlo libremente para convertirlo en un robot de servicios totalmente funcional.

2.2.2 Turtlebot 2.

Es el robot de ROS más famoso, casi todos los tutoriales y paquetes de navegación y movilidad están basados en este dispositivo. *Turtlebot 2* (ver Figura 6) es un robot de bajo coste destinado a la investigación, con una cámara para visión 3D y una base móvil que recuerda mucho a *Roomba*.

Figura 6. Turtlebot 2 de ROS.

Su SDK y librerías están totalmente disponibles en ROS. Su propósito es de prueba y aprendizaje, pero se podría modificar y programar para que realizara tareas más específicas.

2.2.3 ROSbot.

ROSbot (ver Figura 7) es el más parecido al que se diseñará y construirá en este trabajo, su versión completa cuenta con un robot móvil dotado de una cámara para visión y un dispositivo láser tipo LIDAR para percibir la posición de los objetos en el escenario, situados a una determinada altura, en un plano específico.

Figura 7. ROSbot.

2.3 Interacción humano-robot.

La interacción humano-robot es el estudio de los distintos campos donde los robots se relacionan, interactúan o comparten espacio de trabajo con humanos [5]. Por tanto, el robot no sólo ha de ser capaz de entender las órdenes que se le dan, sino que también debe ser capaz de comunicarse con el humano en cuestión.

La interacción más simple entre humano y robot es mediante una interfaz gráfica. El usuario escribe los comandos o utiliza ciertas combinaciones de teclas para enviar una acción al robot y éste la realiza, pudiendo ser monitoreada por la interfaz gráfica, así el robot puede informar de lo que está haciendo en todo momento. Pero esta clase de interacción tan básica sólo sirve si humano y robot están en espacios separados, ya que sin esa interfaz gráfica ninguno de los dos puede saber el estado del otro.

Cuando se habla de interacción robot-humano, la mayoría de las personas no relacionadas con el ámbito de la investigación en Robótica, consideran las *Tres Leyes de la Robótica*, enunciadas por Isaac Asimov:

"Primera Ley: un robot no puede dañar a un ser humano ni, por inacción, permitir que un ser humano sufra daño.

Segunda Ley: un robot debe cumplir las órdenes de los seres humanos, excepto si dichas órdenes entran en conflicto con la Primera Ley.

Tercera Ley: un robot debe proteger su propia existencia en la medida en que ello no entre en conflicto con la Primera o Segunda Ley." [6]

Estas leyes consideran a los robots como elementos dotados de inteligencia real e, incluso de conciencia, características reservadas para los humanos, que hoy en día no son inherentes a los sistemas automáticos o robóticos existentes. De hecho, la discusión sobre estos aspectos se sale del contexto de la Tecnología, para

entrar en el ámbito de la Filosofía. En cualquier caso, un robot, actualmente, es un dispositivo que actuará según el programa que un ser humano ha escrito, o de un modelo que un ser humano ha entrenado (en el caso de que se utilicen sistemas basados en Aprendizaje Automático). Aun así, es importante considerar dos aspectos en el proceso de interacción humano-robot: físico y social.

La **interacción física** es la que está más orientada hacia un entorno industrial, en el que el robot comparte espacio de trabajo con operadores humanos. En esta situación es importante mantener la integridad del operario y del robot, evitando que se pongan en peligro mutuamente. Es por ello que estos robots están equipados con un gran número de sensores destinados a percibir la presencia de personas para garantizar su seguridad. Por poner un ejemplo, si un operario entra en la zona en la que se encuentra el robot y éste lo detecta mediante sus sensores y en base a ciertos cálculos llega a la conclusión de que la maniobra que va a realizar a continuación pondría en peligro a dicho operario, cesará la maniobra de inmediato y no continuará con su labor hasta que la zona sea segura.

Por otra parte, con la **interacción social** se persigue conseguir que el robot se comporte de una forma más natural, comprendiendo lo que se le dice o los gestos que los usuarios hacen. De esta forma, el robot será capaz de distinguir entre diferentes personas o distinguir entre una conversación y una orden.

Las máquinas dotadas con inteligencia artificial cada vez disponen de algoritmos más complejos y de sensores capaces de captar el significado de las reacciones humanas. Algunos conceptos, por ejemplo, Machine Learning (ML), también conocido como Aprendizaje Automático, Deep Learning (DL) o Aprendizaje Profundo, Natural Language Processing (NLP) o Procesamiento de Lenguaje Natural, Sentimental Analysis, Computación Afectiva o incluso el Big Data han contribuido a que los robots nos conozcan cada vez mejor, aunque sea a través del procesamiento de datos. [7]

Como tecnología a destacar de las anteriormente mencionadas, tenemos el paradigma ML, que está presente en una gran variedad de aplicaciones que se usan de forma cotidiana.

2.3.1 Machine Learning.

'Machine Learning' –aprendizaje automático– es una rama de la inteligencia artificial que permite que las máquinas aprendan sin ser expresamente programadas para ello. Una habilidad indispensable para hacer sistemas capaces de identificar patrones entre los datos para hacer predicciones. Esta tecnología está presente en un sinfín de aplicaciones como las recomendaciones de Netflix o Spotify, las respuestas inteligentes de Gmail o el habla de Siri y Alexa. [8]

Esta tecnología se emplea en muy diversos ámbitos, desde los robots que resuelven el cubo de Rubik, a vehículos autónomos, asistentes virtuales o asistentes para la ayuda al diagnóstico médico. Gracias a las técnicas de ML,

las máquinas tienen la habilidad de aprender de los datos que se les proporcionan, por lo que la estadística es parte crucial de esta tecnología. Actualmente existen tres tipos de ML:

Aprendizaje por refuerzo: a la máquina se le designa una tarea y ésta trata de resolverla mediante ensayo y error, hasta encontrar la mejor forma de completar la tarea.

Aprendizaje supervisado: se entrena a la máquina con una serie de datos etiquetados. Así, por ejemplo, si se le muestra la foto de un salón y se dice que en un salón siempre hay un sofá, un televisor y ventanas; al enseñarle más tarde otra foto en la que se encuentran esos tres objetos, será capaz de inferir que se trata de un salón.

Aprendizaje no supervisado: aquí ya no hay etiquetas, sino que la máquina busca patrones que se parezcan y se puedan agrupar.

Su flexibilidad y adaptabilidad a los datos que va recibiendo y su capacidad de aprender de las acciones que ya ha tomado, es lo que hace esta tecnología tan útil.

2.3.2 Reconocimiento de voz.

"El reconocimiento de voz es un tipo de inteligencia artificial que trata de establecer una comunicación entre el hombre y los ordenadores o dispositivos inteligentes, a través del lenguaje humano" [9]. El reconocimiento de voz es esencial para la implementación de comandos de voz en un robot o dispositivo, ya que sin esta capacidad sería incapaz de relacionar lo que le estamos diciendo con los comandos que tiene programados. El motor de voz debe ser capaz de identificar las sílabas del conjunto de fonemas que está recibiendo, para combinarlas y formar las palabras que ha entonado el interlocutor. Existen dos tipos de reconocimiento de voz:

- **Reconocimiento automático del locutor:** estos sistemas permiten a la máquina comprobar que la persona que habla es realmente a la que debe obedecer. Cada persona tiene unos parámetros de habla diferentes (tono, acento, deje, etc.) que la hacen única y fácil de identificar para el sistema. Para entrenar o calibrar al sistema, el usuario debe recitar una serie de muestras de voz que le solicita la máquina, para así crear una serie de patrones que lo identifiquen. De este modo funcionan los asistentes de voz como *Siri* para reconocer a los usuarios.
- **Reconocimiento automático del habla:** básicamente es la transcripción de voz a texto, para simplemente mostrar ese texto o convertirlo en tareas que debe realizar la máquina.

Se emplean diferentes métodos para el reconocimiento de voz:

Modelos ocultos de Markov (HMM en inglés) (ver Figura 8): "*es un proceso estocástico que consta de un proceso de Markov no observado (oculto) y un proceso observado O, cuyos estados son dependientes estocásticamente de los estados ocultos. La tarea fundamental consiste en determinar los parámetros ocultos a partir de los*

parámetros observados" [10]. En este caso se habla de procesos estadísticos complejos que manejan una gran cantidad de variables relacionando los parámetros observados con los que se desconocen.

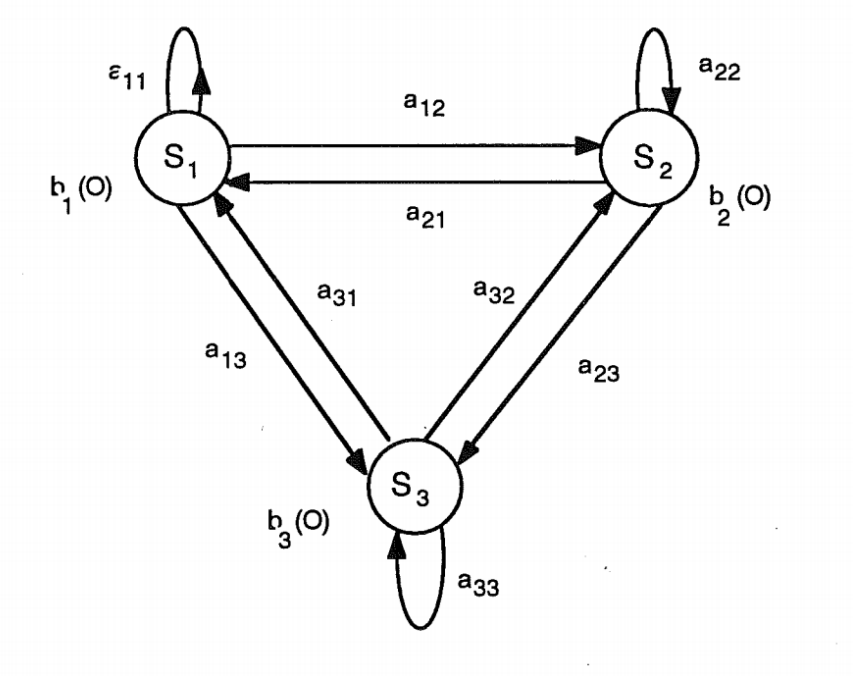


Figura 8. Representación de un HMM.

Técnicas de programación dinámica (DTW, del inglés Dynamic Time Warping): son unos algoritmos que se basan en medir el porcentaje de semejanza existente entre dos secuencias que pueden variar en tiempo o velocidad. Por tanto, si un dato puede linealizarse, podrá analizarse mediante DTW (ver Figura 9).

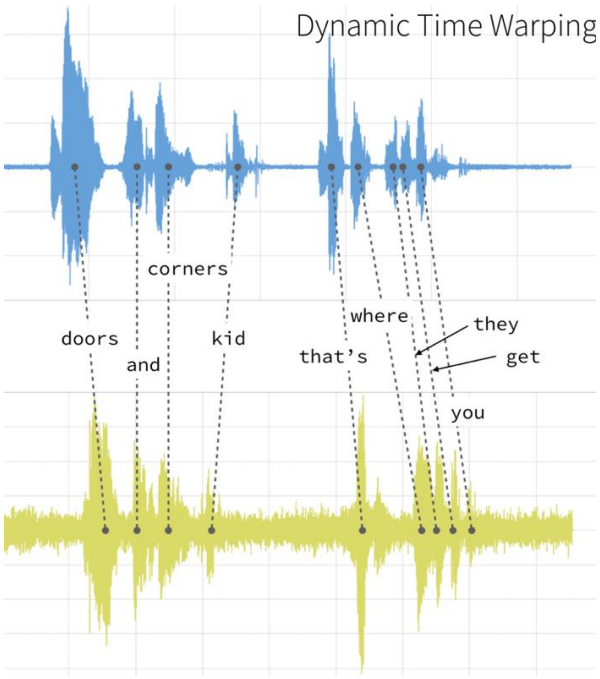


Figura 9. Ejemplo de comparación de dos ondas de audio, que pronuncian la misma oración, con DTW.

Reconocimiento de voz automático de extremo a extremo (ver Figura 10): este método se basa en el uso de redes neuronales. Los sistemas más tradicionales centrados en HMM requieren de un modelo para cada aspecto del habla: acústica, lenguaje y pronunciación. Estos modelos son tratados por separado y luego puestos en común para dar una respuesta. Sin embargo, los modelos de reconocimiento de voz de extremo a extremo integran todos estos modelos de un modo más simple en su línea de entrenamiento, reduciendo así el tiempo de decodificación de la pista de audio.

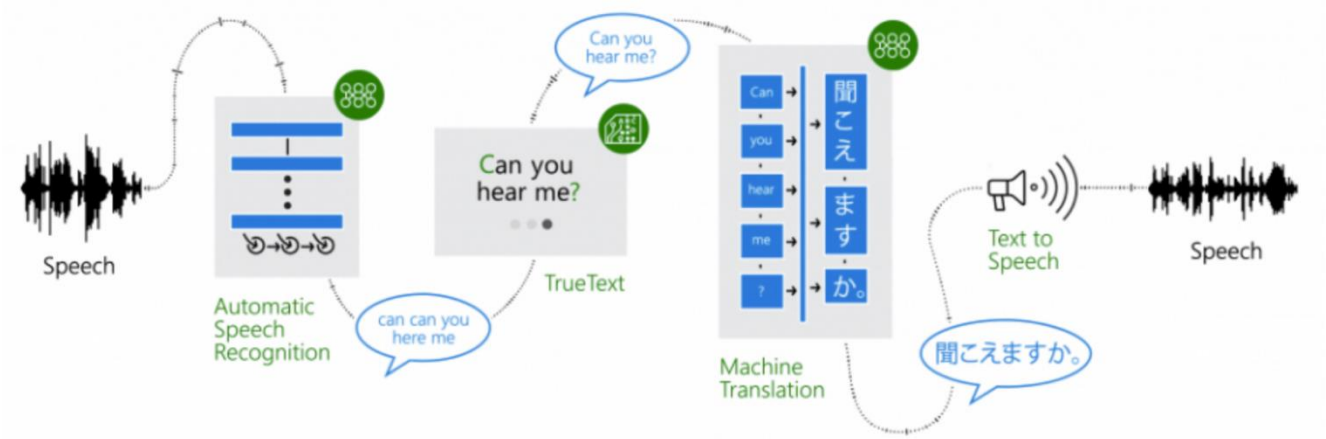


Figura 10. Implementación de un modelo extremo a extremo en Microsoft Translator.

Este método es el que utilizan empresas tan conocidas como Apple, Google, Amazon o Microsoft en sus dispositivos o programas que integran reconocimiento de voz entre sus funcionalidades.

2.3.3 Síntesis de voz.

"La Síntesis de Voz, también conocida como Conversión de Texto a Voz (CTV), consiste en dotar al sistema de la capacidad de convertir un texto dado en voz. Esto se puede hacer mediante grabaciones realizadas anteriormente por personas" [11]. En la Figura 11 podemos observar el diagrama de bloques que sigue un sistema de Síntesis de Voz (TTS, del inglés Text to Speech).

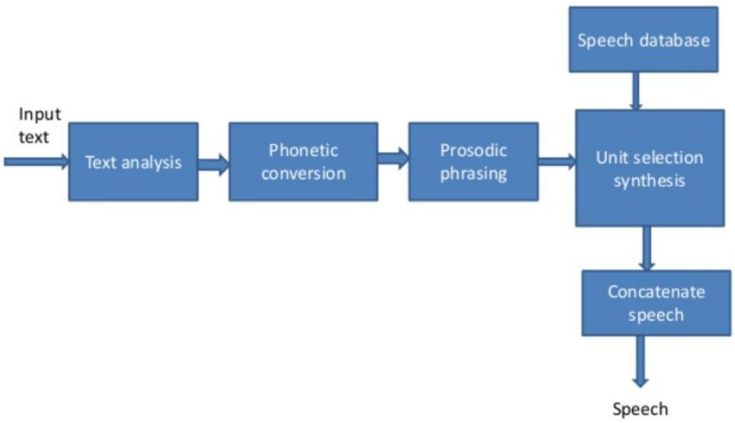


Figura 11. Diagrama de bloques de un sistema TTS (Text to Speech).

El sistema empleará una serie de grabaciones de voz de palabras completas, frases comunes o fonemas para generar el audio de salida, buscando que el sonido sea lo más inteligible y natural posible. Este sistema ha de ser totalmente capaz de convertir cualquier texto a voz. Para conseguir esta síntesis podemos optar por dos métodos:

- **Síntesis concatenativa:** está basado en la conexión de segmentos de audio pregrabados. El método más usado para lograr esta síntesis es la selección de unidades, que emplea una base de datos que contiene grabaciones de voz de fonemas, sílabas, palabras completas, frases y oraciones comunes del idioma correspondiente. Así es capaz de conseguir un sonido natural, pero el gran tamaño que alcanzan estas bases de datos las hace difíciles de manejar, sobretodo si se dispone de varios idiomas.
- **Síntesis de formantes:** este método no usa grabaciones de voz para la respuesta del sistema, sino que crea una onda de sonido artificial, creando una voz robótica. Es cierto que su tono robótico no podrá acercarlo a la naturalidad del método anterior, pero este método produce programas más pequeños que no requieren usar una base de datos con muestras de voz.

Actualmente, multitud de empresas ofrecen servicios online de esta categoría. Esta clase de servicios en la Nube no son gratuitos y requieren de una conexión de forma ininterrumpida a la red. Entre ellas podemos destacar:

Google Cloud: esta plataforma incluye un servicio de síntesis de voz con una AIP basadas en las tecnologías de IA de Google. La API crea voces muy humanas gracias a su desarrollo basado en conocimientos de síntesis de voz mediante DeepMind. Posee una gama de más de 220 voces para más de 40 idiomas. También consta de una versión beta que permite crear una voz personalizada basada en grabaciones de audio.

Amazon Web Services: esta plataforma en la nube de la famosa empresa de compras online cuenta con *Amazon Polly*, que además de la conversión de texto a voz que se ha mencionado anteriormente también dispone de redes neuronales para hacer la transformación, lo que mejora la calidad del habla.

Existen también aplicaciones gratuitas para desempeñar esta tarea, que no requieren de una conexión a internet. Estos sintetizadores son de una menor calidad, produciendo una voz robótica y algún que otro problema de dicción. Algunos ejemplos son *eSpeak*, *Festival* y *pyttsx3*.

2.3.4 Reconocimiento del lenguaje natural.

*"El **Procesamiento del Lenguaje Natural** (PNL), en inglés Natural Language Processing (NLP), es un campo de conocimiento que combina las tecnologías de la ciencia computacional con la lingüística aplicada y que, básicamente, pretende conseguir que una máquina comprenda lo que expresa una persona mediante el uso de una lengua natural, sea cual sea la lengua materna de la persona"* [12]. Esta rama de la inteligencia artificial se emplea en diversas áreas de actuación, como se muestra en la Figura 12.

Figura 12. Aplicaciones del NLP, cortesía de DAIL.

- **Extracción de información**: para extraer información automáticamente de manera estructurada o semiestructurada a partir de un documento de texto legible por la máquina.
- **Respuesta a preguntas**: son buscadores de respuestas que reconocen clases de preguntas del tipo "cómo", "cuándo", "dónde", "por qué", listas, definiciones, etc., y recupera respuestas planteadas en lenguaje natural.
- **Traducción automática**: traducción de texto o voz de un lenguaje natural a otro.
- **Síntesis de voz**: creación de voz artificial.
- **Recuperación de información**: procesamiento de documentos de texto digitales para su recuperación en base a unas palabras clave establecidas por el usuario.
- **Comprensión del lenguaje:** lectura, interpretación, contextualización, establecimiento de significado e intención por parte de una máquina sobre un mensaje dado.
- **Reconocimiento del habla:** se procesa la voz del usuario para reconocer la información que transmite y transcribirla a texto o ejecutar órdenes.
- **Generación de lenguaje natural:** convertir datos a texto o voz.

El NLP se basa en la separación de las piezas elementales del lenguaje, para su interpretación mediante métodos estadísticos, algoritmos, reglas y aprendizaje basado en *Deep Learning* y *Machine Learning.* Las funciones que integran los NLP son las siguientes:

- **Segmentación de palabras:** separa las palabras de un texto dado.
- **Segmentación de oraciones:** separa las oraciones que contiene el texto.
- **Lematización:** obtención de la forma base de una palabra (por ejemplo, la base de un verbo conjugado es su infinitivo).
- **Segmentación morfológica:** obtención de los morfemas que conforman la palabra.

- **Análisis sintáctico:** determinar las relaciones que mantiene las palabras de una frase.
- **Etiquetado gramatical:** determinar la función de la palabra dentro de la frase.

La aplicación de técnicas de NLP en el ámbito empresarial aporta una gran cantidad de beneficios. Se pueden destacar:

Agilizar y automatizar trabajo: mayor rapidez para el procesado de documentos de una empresa, rebajando la carga de trabajo del personal y permitiendo que se dediquen a tareas más apremiantes o que requieran de un carácter más humano.

Análisis de sentimientos: mediante NLP es posible ser alertados de comportamientos de la comunidad de usuarios como son los comentarios positivos/negativos, tendencias, problemas o estado de satisfacción general de nuestro público.

Traducción automática: gracias al NLP es posible romper la barrera de idioma y mantener una conversación con una persona de diferente nacionalidad sin la necesidad de que uno de los implicados hable en una lengua extranjera o la necesidad de un intérprete.

Chatbots y sistemas de conversación: mediante el NLP, estos bots son capaces de comprender no sólo la información que se está recibiendo, sino su contexto, proporcionando respuestas adecuadas y adaptadas al usuario.

Extracción de contenido importante: el NLP agiliza las tareas de rescate de información útil que deben de realizarse diariamente en trabajos como recursos humanos o en la abogacía, rebajando la carga de trabajo del personal y permitiendo la concentración de esfuerzos en tareas más importantes.

2.3.5 Redes Neuronales Artificiales.

Las redes neuronales se emplean en métodos de *Machine Learning*. Las **redes neuronales** son modelos simples del procesamiento de información que realiza el sistema nervioso humano. Las unidades básicas de procesamiento son las neuronas, que se organizan en capas. Una red neuronal está compuesta por tres capas, como se muestra en la Figura 13 [13]:

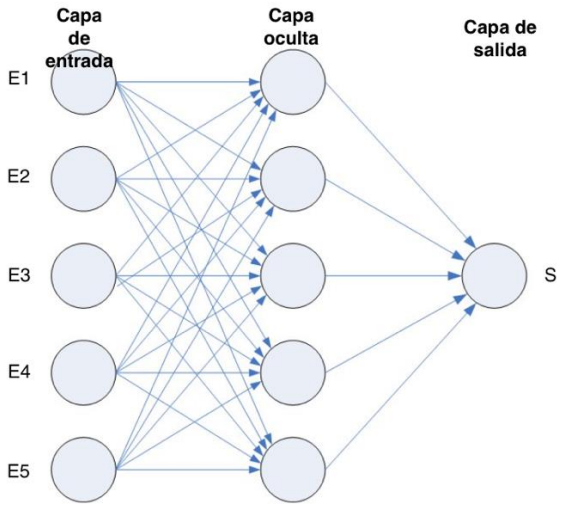


Figura 13. Estructura de una red neuronal artificial.

- **Capa de entrada:** se encuentra al inicio de la red y solo puede existir una. Aquí se muestran las unidades que representan los datos de entrada. La cantidad de neuronas dependerá de la dimensión de esas entradas.
- **Capas ocultas:** como mínimo debe existir una. Se encuentran entre la capa de entrada y la de salida. Su función es procesar los datos de entrada, multiplicándolos por sus correspondientes pesos o ponderaciones en el proceso, dando como resultado un valor que proporcionará el dato de salida.
- **Capa de salida:** se encuentra al final de la red y solo puede existir una. Aquí se encuentra la unidad o unidades del campo o los campos de destino.

La red neuronal examina cada registro de forma individual, generando una predicción para cada uno y ajustando las ponderaciones cunado la predicción realizada resulta ser errónea. Este proceso se repite numerosas veces y así es como la red aprende y mejora las predicciones hasta alcanzar un nivel marcado como óptimo.

En un principio las ponderaciones o pesos de cada neurona son aleatorios y dan unos resultados poco acertados. El **entrenamiento** es lo que permite a la red neuronal aprender y ser más precisa. Este entrenamiento se basa en la continuada presentación de ejemplos con resultados conocidos a la red, de forma que las respuestas que generan se comparan con los resultados, esta comparación da lugar a cambios en las ponderaciones de forma continuada hacia atrás. Al repetir este proceso una gran cantidad de veces, la red se vuelve cada vez más precisa en su predicción de resultados. Una vez entrenada la red, cuando alcanza un porcentaje de fiabilidad que el usuario considere óptimo (mayor a un 80% por ejemplo), la red está lista para emplearse en casos en los que se desconoce el resultado.

2.3.6 TensorFlow.

Tensorflow es una plataforma de código abierto de extremo a extremo para el aprendizaje automático (AA). Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que permite desarrollar e implementar con facilidad aplicaciones con tecnología de AA [26].

TensorFlow es una plataforma desarrollada por Google que permite construir y entrenar redes neuronales. Su arquitectura flexible le permite implementar el cálculo en equipos de escritorio, servidores o dispositivos móviles con una sola API.

Ofrece varios niveles de abstracción para adaptarse a las necesidades del usuario. La API (Interfaz de Programación de Aplicaciones) de alto nivel de Keras permite compilar y entrenar modelos. En el apartado 2.3.7 se describe la biblioteca Keras. TensorFlow permite el entrenamiento y la implementación de modelos en servidores, la web o dispositivos perimetrales sin importar el lenguaje que utilice la plataforma. Dependiendo de la plataforma existen distribuciones de TensorFlow específicas, como puede ser TensorFlow Lite para dispositivos móviles o TensorFlow.js para entornos de JavaScript.

TensorFlow está instalado como herramienta en Google Colab, un entorno inteactivo gratuito que permite escribir y ejecutar código.

2.3.7 Keras.

Keras es una biblioteca de código abierto, escrita en Python, para la creación de redes neuronales [14]. Esta biblioteca funciona a nivel de modelo, proporcionando una selección de bloques modulares que se emplean para desarrollar modelos complejos de *Deep Learning*. De esta forma, las capas de la red neuronal a configurar se relacionan unas con otras, siguiendo un principio modular. Esto reduce la carga de trabajo del usuario, que no tiene la necesidad de entender o controlar directamente el *framework*.

Keras se encuentra actualmente implementado en herramientas como TensorFlow. Esta biblioteca simplifica el proceso de creación de una red neuronal, la interacción del usuario es mínima y si se desarrollan errores, se proporciona la información necesaria para poder resolver la incidencia. Las principales características de Keras son:

- **Alta compatibilidad con las plataformas** de modelos desarrollados: los modelos de Keras son compatible con iOS, Android, Google Cloud y Raspberry Pi.
- **Compatibilidad entre *frameworks***: Keras permite combinar varios marcos de trabajo o transferir los modelos creados de un marco a otro.

- **Soporte múltiple para GPU:** los recursos para el desarrollo de los módulos de aprendizaje o entrenamiento de las redes neuronales se pueden distribuir entre diferentes chips o tarjetas gráficas.
- **Desarrollado por grandes empresas:** Keras está en mantenimiento y desarrollo con el apoyo de empresas tan grandes e importantes como Google, Apple, Amazon AWS y Nvidia entre otros.

3 Diseño del sistema.

En el presente capítulo se expondrán las diferentes arquitecturas, dispositivos y softwares empleados en la construcción del prototipo. Se describe los módulos hardware del robot y como se ha ensamblado. Se especifica el diseño de arquitectura de software empleado en la puesta a punto del robot. Se explica que es el ecosistema Gazebo/ROS y la importancia de la simulación en labores de prueba del prototipo. También, se detalla el proceso de integración de todos los elementos del robot real dentro del sistema ROS. Finalmente, se describe como se ha usado la simulación mediante RVIZ para la monitorización tanto del robot real como de la simulación.

Por otra parte, también se especifican los métodos y componentes adicionales que se han integrado para realizar el reconocimiento de personas, así como el reconocimiento y síntesis de voz.

Por último, se describe cómo se han integrado todos los componentes que dan forma al prototipo, además de las interfaces de usuario desarrolladas para establecer la comunicación usuario-robot.

3.1 Arquitectura del sistema.

El prototipo construido es un robot de tres ruedas, dos motoras y una rueda loca de tipo castor. Las ruedas motoras son independientes la una de la otra, lo que permite al robot poder girar sobre sí mismo. En lo referente a sensores propioceptivos cada rueda cuenta con un encoder, encargado de informar sobre el estado de ésta (cuánto ha rotado), lo que permite al sistema calcular la odometría. Para controlar las ruedas motoras, se conectan al circuito Devantech MD25, que por medio de un driver nos permitirá programar su funcionamiento. Los sensores exteroceptivos que se han empleado son un dispositivo de barrido láser (tipo LIDAR) destinado a labores de navegación, y una cámara tipo RGB para la realización del reconocimiento de personas y la síntesis y reconocimiento de voz. Como nexo de unión y programación de todos los componentes del robot se dispone de una SBC (Single Board Computer), Raspberry Pi 4, que actúa como elemento central de procesamiento.

En lo referente al software, en la Raspberry Pi 4 se ha instalado el sistema operativo Ubuntu 18.04 para poder hacer uso de ROS Melodic, el ecosistema que permite la programación, seguimiento y monitorización del robot. En el espacio de trabajo de ROS se incluirán los drivers de la cámara y el láser, además de un driver ROS para el control del sistema de locomoción mencionado (MD25), cedido por la Dra. Nieves Pavón para su inclusión y modificación en este proyecto. En este espacio de trabajo se crearán los archivos que configurarán el robot para realizar tareas de navegación e interacción humano-robot.

Por lo tanto, podemos distinguir entre dos partes bien diferenciadas de la arquitectura del sistema, el hardware y el software. En los apartados siguientes se explicará cada parte de forma detallada.

3.2 Arquitectura hardware.

Debido a la situación originada como consecuencia de la pandemia Covid-19, era muy complicado usar el prototipo de la UPCT a causa de su gran tamaño, la difícil disponibilidad de espacios de trabajo en la ETSII y la siempre presente amenaza de contagio aun cumpliendo todas las restricciones y normas para la prevención del virus. Para no renunciar al enfoque práctico del trabajo y pasar a trabajar con simulaciones, se apostó por un enfoque doméstico, la construcción desde cero de un prototipo cuyo tamaño fuese adecuado al espacio de libre movilidad que posee una vivienda, con un chasis compuesto por materiales que se pueden encontrar en cualquier ferretería y tienda de electrónica. Esto permitió que, salvo componentes más específicos como los sensores, se pudieran conseguir la mayoría de los materiales en un periodo reducido de tiempo. Este enfoque también ha permitido centrar el trabajo en casa, por lo que se ha reducido la posibilidad de contagio, además de brindar la oportunidad de probar el robot en el entorno para el que se ha diseñado, el interior de una vivienda.

Los componentes de la arquitectura hardware del robot son los siguientes:

Raspberry Pi 4 (ver Figura 14, Figura 15, Figura 16)**:** es un SBC de escritorio de doble pantalla, que ouede usarse como unidad de procesamiento de un robot, el hub de un hogar inteligente, un centro multimedia, un núcleo de IA en red, un controlador de fábrica y mucho más [13]. En este caso se empleará como elemento de procesamiento local del robot.

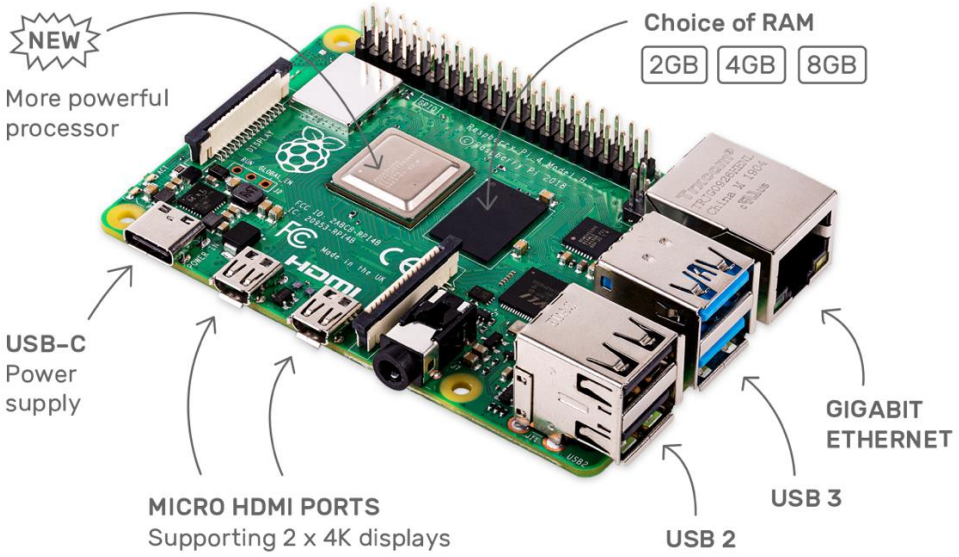


Figura 14. Descripción general de la Raspberry Pi 4.

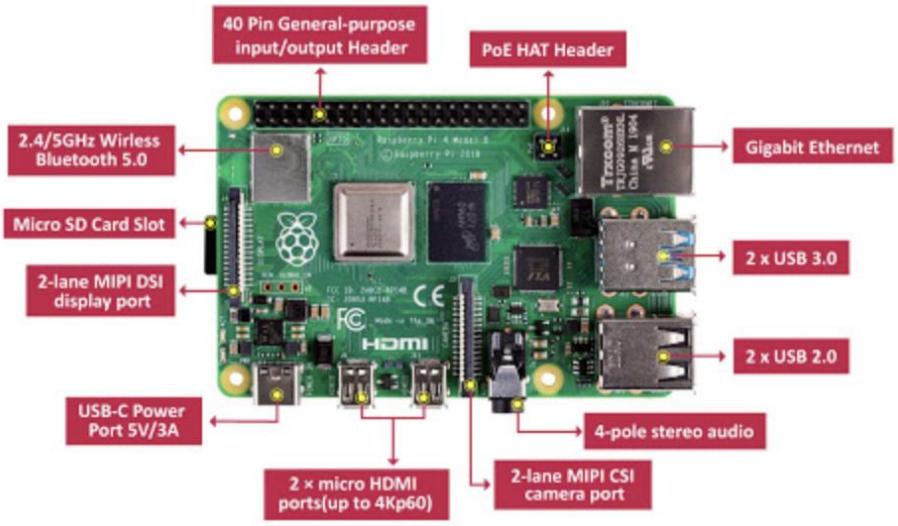


Figura 15. Componentes de la Raspberry Pi 4, aportadas por MSRobotics.

Processor:	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memory:	1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC
Connectivity:	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE Gigabit Ethernet 2 × USB 3.0 ports 2 × USB 2.0 ports.
GPIO:	Standard 40-pin GPIO header (fully backwards-compatible with previous boards)
Video & sound:	2 × micro HDMI ports (up to 4Kp60 supported) 2-lane MIPI DSI display port 2-lane MIPI CSI camera port 4-pole stereo audio and composite video port
Multimedia:	H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
SD card support:	Micro SD card slot for loading operating system and data storage
Input power:	5V DC via USB-C connector (minimum 3A[1]) 5V DC via GPIO header (minimum 3A[1]) Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature 0–50ºC
Compliance:	For a full list of local and regional product approvals, please visit https://www.raspberrypi.org/documentation/hardware/raspberrypi/conformity.md
Production lifetime:	The Raspberry Pi 4 Model B will remain in production until at least January 2026.

Figura 16. Especificaciones de la Raspberry Pi 4, cortesía de *raspberrypi.org*.

MD25 (ver Figura 17, Figura 18): es un circuito de tipo puente en H capaz de controlar dos motores de corriente continua de hasta 2,8 amperios y 12V. El circuito se controla externamente mediante un bus serie o mediante bus I2C, lo que permite su comunicación con cualquier microcontrolador moderno. El circuito cuenta con numerosos registros que controlan cada motor en términos de aceleración, velocidad o corriente, entre otros, y los contadores de los encoders de cada motor [14].

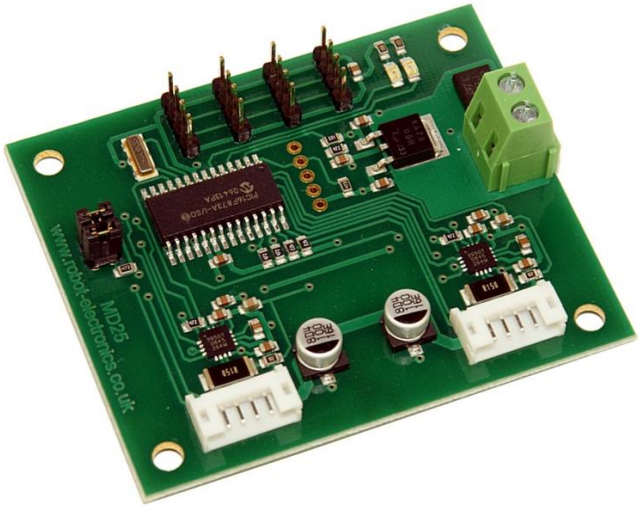


Figura 17. Vista general de la placa MD25, aportado por *superrobotica.com*.

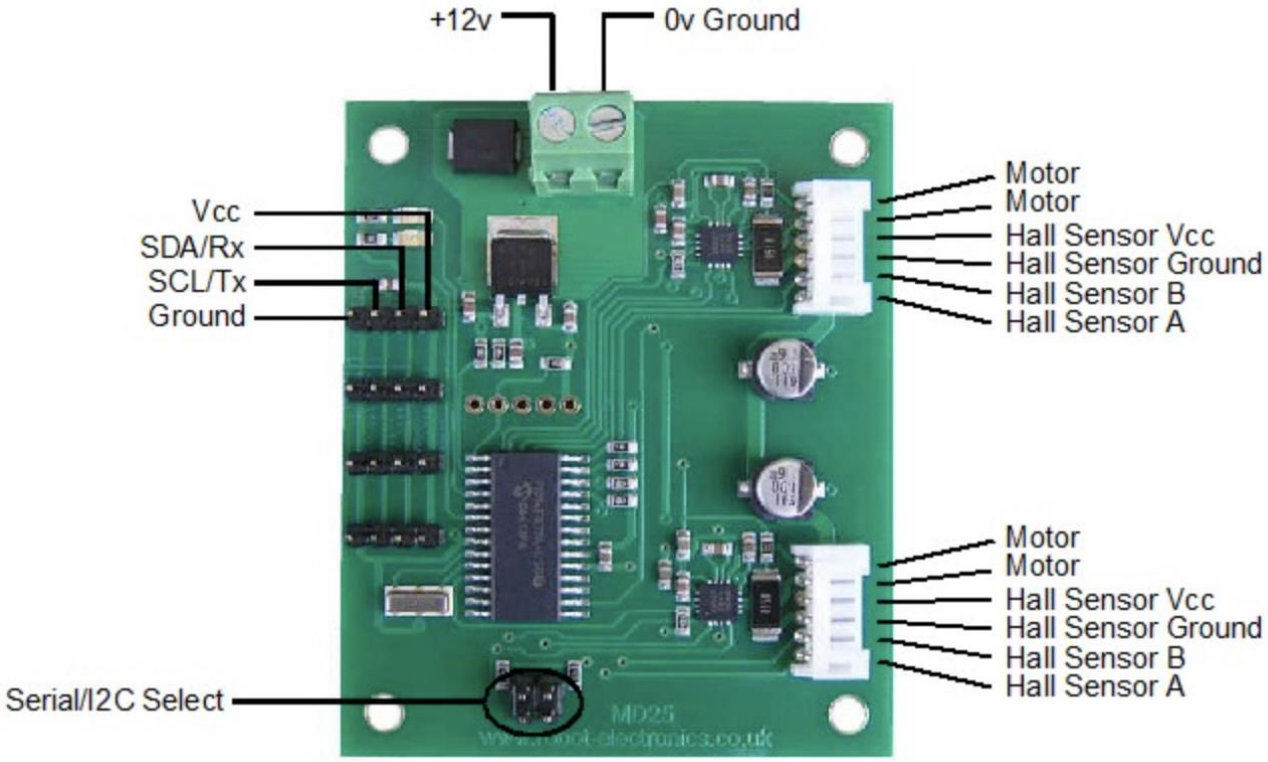


Figura 18. Esquema de conexiones de la placa MD25, aportado por *superrobotica.com*.

RPLidar A1 (ver Figura 19, Figura 20, Figura 21, Figura 22): es un escáner láser 2D de 360 grados de bajo coste desarrollado por SLAMTEC. El sistema puede realizar un escaneado de 360 grados con un rango o distancia máxima de 12 metros. La nube de puntos 2D producida puede emplearse en mapeado, localización y modelado de objetos/entorno [15]. Las aplicaciones para las que este dispositivo de barrido láser puede usarse son:

- Navegación y localización de un robot de servicio doméstico o de limpieza.
- Navegación y localización de un robot de uso general.
- Localización y evitación de obstáculos de un juguete inteligente.
- Escáner de entorno y modelado 3D.
- Mapeado y localización simultánea (SLAM).

Figura 19. Vista general del RPLidar A1, cortesía de SLAMTEC.

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter(m)	TBD	0.15 - 12	TBD	White objects
Angular Range	Degree	n/a	0-360	n/a	
Distance Resolution	mm	n/a	<0.5 <1% of the distance	n/a	<1.5 meters All distance range*
Angular Resolution	Degree	n/a	≤1	n/a	5.5Hz scan rate
Sample Duration	Millisecond(ms)	n/a	0.5	n/a	
Sample Frequency	Hz	n/a	4000	8000	
Scan Rate	Hz	1	5.5	10	Typical value is measured when RPLIDAR A1 takes 360 samples per scan

Figura 20. Especificaciones de medición del RPLidar A1 de SLAMTEC.

Item	Unit	Min	Typical	Max	Comments
Laser wavelength	Nanometer(nm)	775	785	795	Infrared Light Band
Laser power	Milliwatt (mW)	TBD	3	5	Peak power
Pulse length	Microsecond(us)	TBD	110	300	

Figura 21. Especificaciones ópticas del RPLidar A1 de SLAMTEC.

Item	Unit	Min	Typical	Max	Comments
Scanner system voltage	Volt (V)	4.9	5	5.5	If the voltage exceeds the max value, it may damage the core.
Scanner system voltage ripple	Millivolt(mV)		20	50	High ripple may cause the core working failure.
Scanner system start current	Milliampere (mA)	TBD	500	600	Underpower may cause the startup failure.
Scanner system current	Milliampere (mA)	TBD	80	100	Sleep mode, 5V input
		TBD	300	350	Work mode, 5V input
Motor system voltage	Volt (V)	5	5	10	Adjust voltage according to speed
Motor system current	Milliampere (mA)	TBD	100	TBD	5V input

Figura 22. Especificaciones de potencia del RPLidar A1 de SLAMTEC.

Orbbec Astra Pro (ver Figura 23, Figura 24, Figura 25): es una cámara 3D de tipo RGB-D desarrollada para ser compatible con aplicaciones desarrolladas con OpenNI [16]. Se puede aplicar principalmente en:

- Control por gestos.
- Robótica.
- Digitalización 3D.
- Desarrollo de nubes de puntos.
- Sistemas interactivos.
- VR/AR (Virtual Reality/Augmented Reality).

Figura 23. Orbbec Astra Pro, cámara RGB de Orbbec.

Figura 24. Componentes de la Orbbec Astra Pro, según ROS Components.

N° Modelo	ORBBEC ASTRA PRO
Dimensiones	160 x 30 x 40 (mm)
Peso	300 g
Rango	0.4 -8 m
Profundidad de Imagen	720p@30FPS
Campo de Visión	60° horiz. x 49.5 ° vert. (73° diagonal)
Micrófonos	2
Sistema Operativo	Windows, Linux, Android

Figura 25. Especificaciones de la cámara Orbbec Astra Pro, cortesía de ROS Components.

Devantech RD02 (ver Figura 26, Figura 27): es un sistema de controlador diferencial de motores a 12 voltios para un robot. Consta de dos motores EMG30 con encoders, soporte de montaje, dos ruedas y conexión preparada para MD25 [17]. El sistema de locomoción del robot se ha construido usando este sistema.

Figura 26. Sistema motriz RD02 de Devantech.

Rated voltage	12v
Rated torque	1.5kg/cm
Rated speed	170rpm
Rated current	530mA
No load speed	216
No load current	150mA
Stall Current	2.5A
Rated output	4.22W
Encoder counts per output shaft turn	360

Figura 27. Especificaciones del sistema RD02 de Devantech, cortesía de RobotShop.

Batería de plomo ácido de 12V 5Ah (ver Figura 28): esta batería se emplea para suministrar corriente tanto al MD25 y el sistema de motorización como a la Raspberry Pi 4.

Figura 28. Batería de plomo ácido de 12V 5Ah.

UBEC (ver Figura 29) [18]: es una fuente de alimentación conmutada externa a su regulador que sirve para garantizar la alimentación de la Raspberry Pi 4. Esto es debido a que la batería que se emplea para alimentar al robot es de 12V, mientras que la Raspberry Pi 4 soporta 5V. El UBEC se encarga de adaptar la corriente de la batería a las especificaciones de la Rasoberry Pi 4.

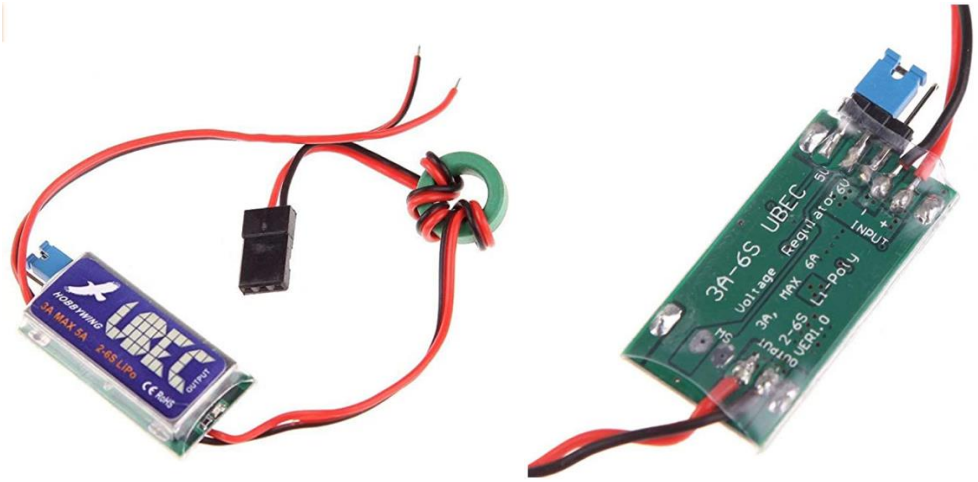


Figura 29. UBEC de 5V 3A vendido por QC EU en Amazon.

Adaptador serie TTL a USB (ver Figura 30): con este adaptador es posible conectar y comunicar la placa MD25 con la Raspberry Pi 4, mediante la conexión serie disponible, lo que facilita el uso del driver de ROS utilizado.

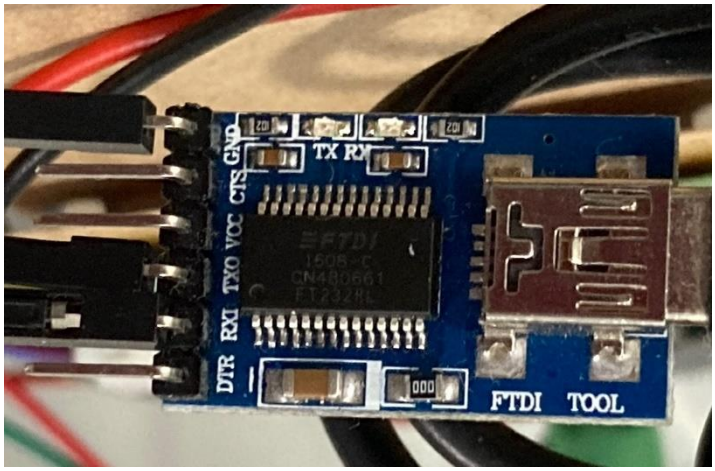


Figura 30. Adaptador serie TTL a USB.

El **chasis** del robot se ha construido a partir de elementos que se pueden encontrar en cualquier ferretería y tienda de electrónica. Los componentes del chasis son:

- (x3) Tablero de fibra de madera de 300x200x3mm.
- (x4) Barras roscadas de 6mm de diámetro por 200mm de largo.
- (x33) Tuercas de 6mm de diámetro interno.
- (x5) Tornillos de 6mm de diámetro y 15mm de largo.

- (x16) Arandelas de 6mm de diámetro interno y 1mm de espesor.
- (x1) Placa de baquelita de 300x200x3mm.
- (x8) Tornillos de electrónica con tuerca de 3mm de diámetro.
- (x1) 1m de velcro de doble cara.
- (x1) Rueda loca de 25mm de radio.

El chasis toma la forma de un robot de dos plantas, en la planta inferior se colocan la Raspberry Pi 4, el UBEC y el MD25 con su adaptador, la batería y se atornillan por debajo las ruedas. Para reforzar, la planta inferior se ha puesto doble tablero de fibra de madera, pero con una separación para poder pasar cableado. En la planta superior se coloca solo el láser, ya que al ser de 360 grados no debe encontrase obstáculos que generen falsos valores a la hora de mapear. La cámara se ha colocado en una posición intermedia entre ambas plantas, para un mejor ángulo de visión.

Las conexiones del láser y la cámara a la Raspberry Pi 4 se realizan mediante USB, mientras que la alimentación y transmisión de datos con la placa MD25 se muestra en la Figura 31.

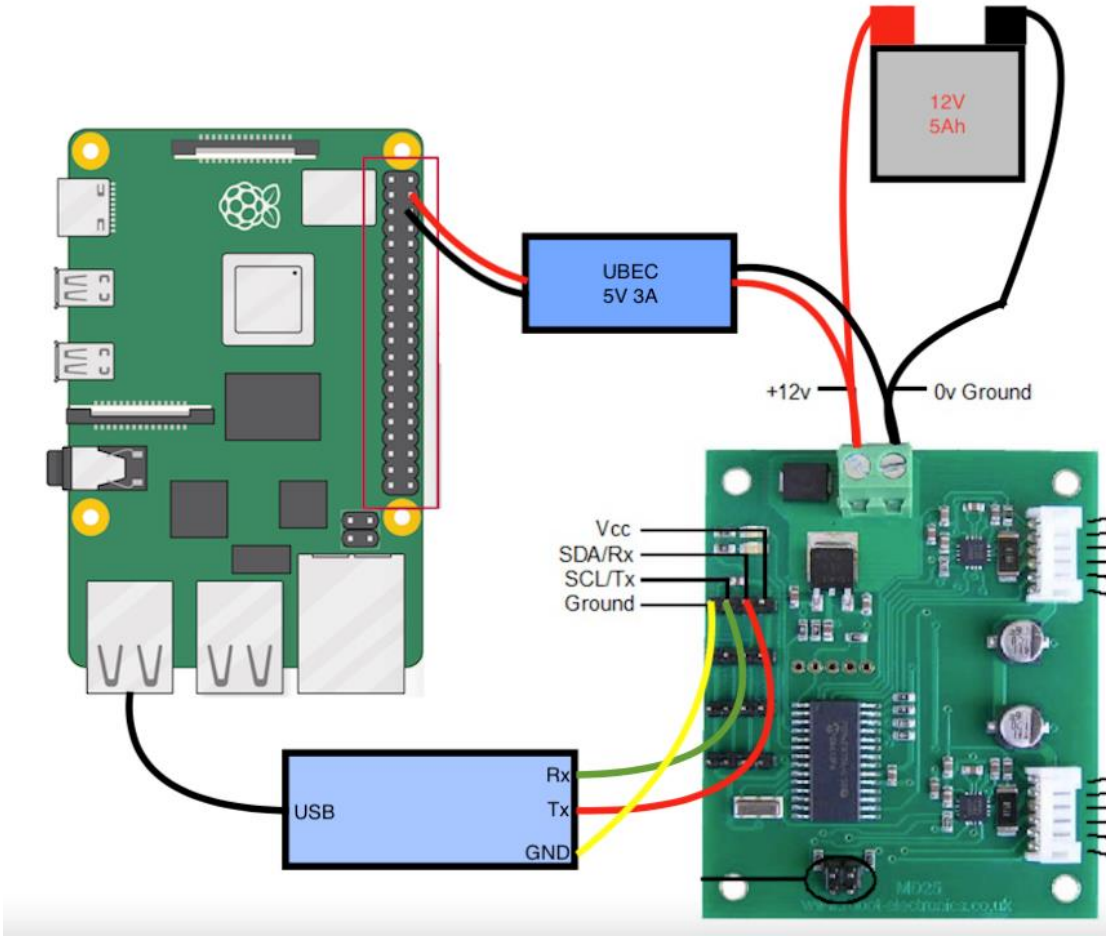


Figura 31. Esquema de interconexionado entre batería, MD25 y Raspberry Pi 4 (elaboración propia).

En la Figura 32 se muestra la vista frontal del robot. Se pueden distinguir claramente los dos niveles del robot y como se ha colocado la cámara Orbbec Astra Pro en una posición intermedia, que permitirá ajustar el grado de inclinación de la misma, para centrar la visión en lo que tiene justamente delante o en zonas de mayor altura. La rueda loca (de tipo castor) es la rueda delantera y el dispositivo SBC Raspberry Pi se coloca justo encima de ella para un acceso cómodo.

Figura 32. Vista frontal.

En la Figura 33 se presenta la vista lateral izquierda. Se observa el doble refuerzo que tiene la base del robot, así como que las ruedas motoras están instaladas en la parte posterior del prototipo, siendo esta la zona motriz. Encima de las ruedas motoras, en la base del robot se ha colocado la placa MD25, encargada de controlar las ruedas. La batería que alimenta al robot está colocada entre las dos barras de la parte posterior del prototipo, donde hay un mayor refuerzo gracias al anclaje de las barras y las ruedas a la base, lo que favorece el reparto del peso de la batería y que el robot la pueda soportar sin problemas.

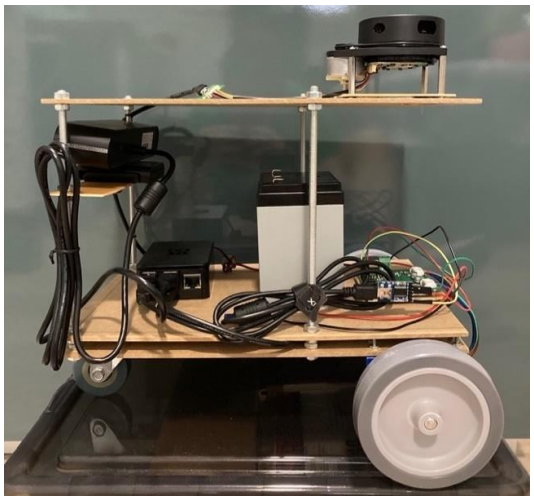


Figura 33. Vista lateral izquierda.

En la Figura 34 se muestra la vista trasera. Se puede observar la conexión de los motores de las ruedas, el adaptador serie TTL a USB y el dispositivo UBEC a la placa MD25, que a su vez se conecta a la batería para alimentar todo el sistema. El láser RPLidar A1 está colocado en la parte superior del robot.

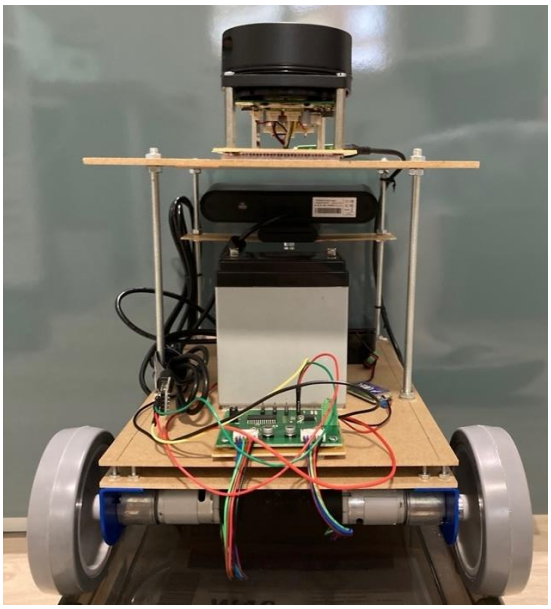


Figura 34. Vista trasera.

En la Figura 35 se presenta la vista lateral derecha. En ella se puede ver cómo se aprovecha la separación entre las dos planchas de fibra de madera que conforman la base del robot, para pasar cableado como el conector USB del láser.

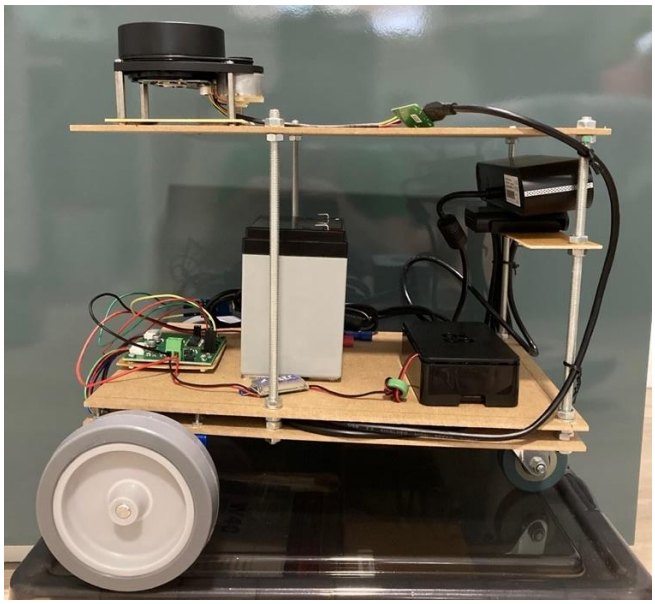


Figura 35. Vista lateral derecha.

En la Figura 36 se muestra la vista superior del prototipo. Esta figura ilustra cómo se ha colocado el láser, de forma que su centro se corresponde con el del eje que une las ruedas motoras. Esta forma de colocar el láser hace que, a la hora de hacer la programación, los cálculos de transformadas sean más sencillos y se produzcan menos errores.

Figura 36. Vista superior.

3.3 Arquitectura software.

La finalidad de este trabajo es el montaje, puesta en marcha y funcionamiento de un robot real autónomo con funciones de navegación y localización, además de visión artificial para la interacción humano-robot mediante métodos de ML y DL. Para integrar todos los componentes del sistema se ha utilizado ROS, dado que posee herramientas para la programación, simulación y monitorización del prototipo. En la asignatura Robótica Móvil, previamente cursada en la Universidad Politécnica de Cartagena se sentaron las bases necesarias para entender y emplear este ecosistema de programación, aunque sólo en simulación debido al contexto de pandemia COVID19 vivido. Así, la curva de aprendizaje para adaptar ROS al sistema planteado ha sido poco pronunciada.

Para la navegación autónoma del robot se emplearán los paquetes de navegación que hay disponibles para ROS, junto con los drivers ROS ya existentes para el láser RPLidar A1 y el driver ROS del sistema de locomoción basado en MD25, modificado para incluir mejoras en la maniobrabilidad del robot.

Para que el robot sea capaz de desplazarse por el interior de la vivienda de forma autónoma, se tiene que realizar previamente tareas de mapeado, localización, trazado de trayectoria, teleoperación y, opcionalmente, exploración autónoma.

Por otra parte, la cámara RGB-D, que también incorpora un array de micrófonos se usará para labores de interacción humano-robot tales como:

- Reconocimiento de voz.
- Reconocimiento de personas.
- Body tracking.

Muchas de estas tareas, que utilizan el sensor mencionado para adquirir los datos de entrada, necesitan que dichos datos se procesen siguiendo el paradigma de DL. Para facilitar la implementación de modelos de DL se ha utilizado el marco de desarrollo (framework) *TensorFlow*.

Pero antes de instalar estos elementos software y comenzar la programación hay que tener un sistema operativo (OS del inglés) de base. Se ha optado por Ubuntu 18.04, ya que es compatible con ROS Melodic (la versión. De ROS utilizada en este proyecto) y está concebida como una distribución de software libre. Este OS nos permitirá instalar e implementar todos los componentes del sistema en la Raspberry Pi 4 de una forma más cómoda, ya que se proporciona una interfaz gráfica como es el desktop o escritorio de este OS.

Primero es necesario buscar la imagen en el sitio web oficial de Ubuntu, este archivo incluye todo lo necesario para instalar el OS (ver Figura 37).

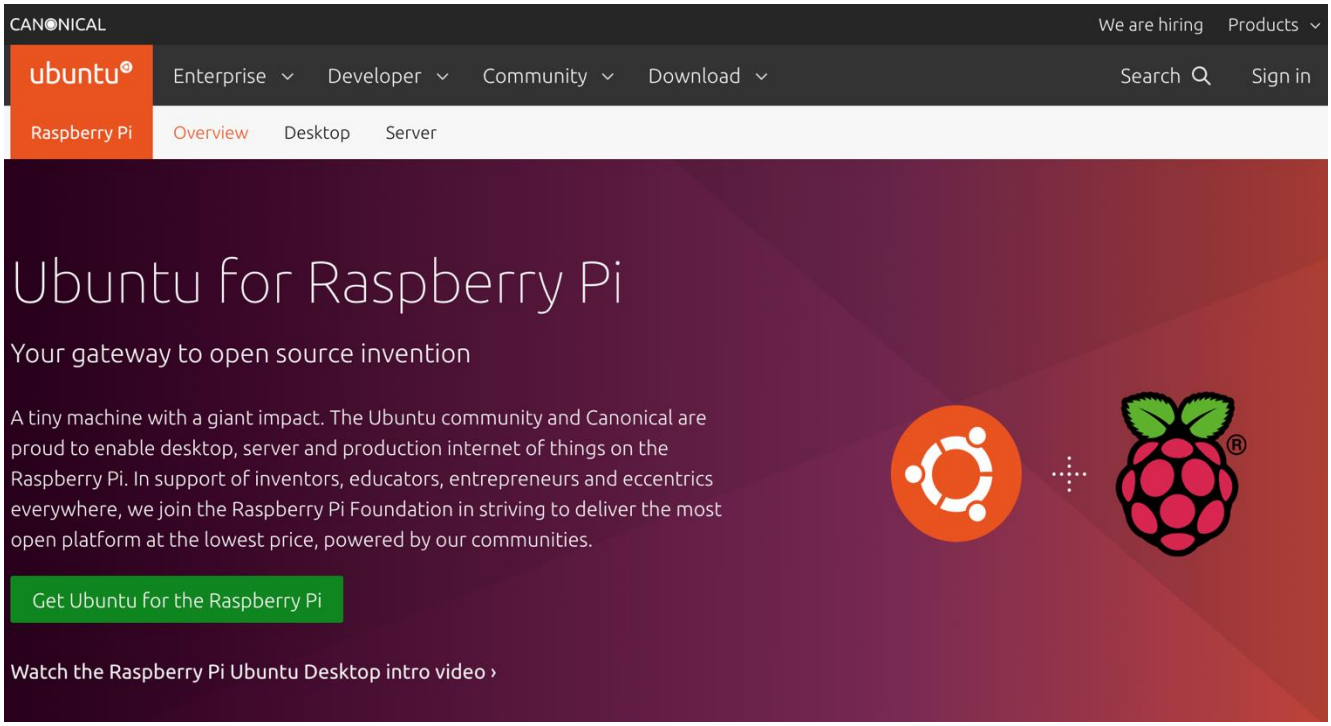


Figura 37. Sitio web de Ubuntu para la descarga de las imágenes que contienen los sistemas operativos disponibles.

Una vez descargado el archivo, es necesario copiar (o “quemar”), la tarjeta Micro SD que funciona como disco duro de la Raspberry Pi. Se ha usado *balenaEtcher* (ver Figura 38), una aplicación gratuita disponible para

multitud de sistemas operativos como Windows o Mac OS. Esta aplicación es muy simple de utilizar, solo hay que abrirla, seleccionar el archivo a "quemar" y en que dispositivo, y realizar la acción de comenzar.

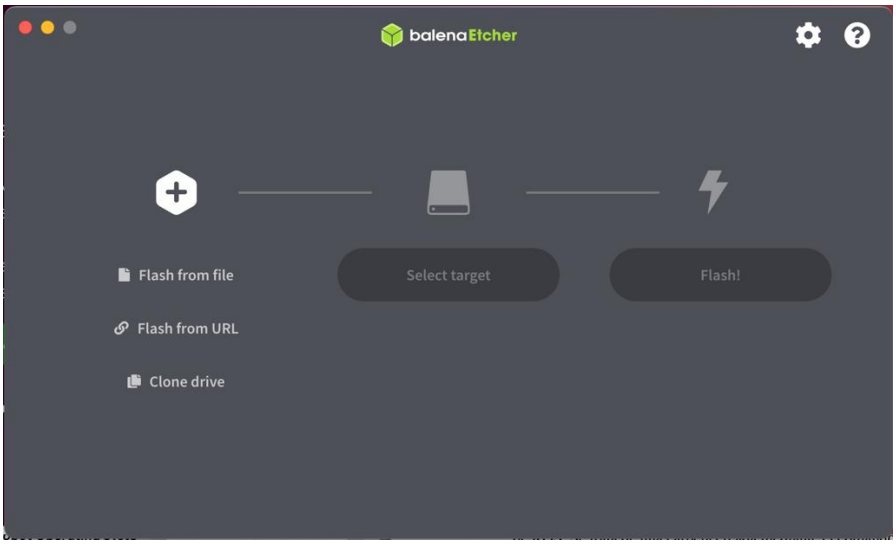


Figura 38. Captura de pantalla de la ventana de la aplicación balenaEtcher.

Tras este paso, la Micro SD ya está lista, pero para inicializar la Raspberry Pi es necesario:

- Conectar un teclado y un ratón.
- Conectarla a un monitor por medio de un cable HDMI.
- Conectarla a la red por medio de un cable Ethernet.
- Insertar la tarjeta Micro SD.

Una vez hecho esto, es posible conectarla a la corriente y comenzará un proceso de inicialización que puede verse en pantalla. Cuando éste haya concluido, se mostrará en pantalla los siguientes mensajes:

```
Ubuntu 18.04.4 LTS Ubuntu tty1
ubuntu login:
```

El usuario (login) y clave establecidas por defecto son:

```
login: ubuntu
password: ubuntu
```

Es necesario introducir una nueva contraseña y, una vez establecida, aparecerá un mensaje indicando que se está en Ubuntu 18.04 (ver Figura 39).

```
Last login: Thu Mar 3 13:00:00 UTC 2020 on ***.***.***.***
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.3.0-1017-raspi2 aarch64)
* Documentation:  https://help.ubuntu.com
* Management:     https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

  System information as of Thu Mar 5 15:41:49 UTC 2020

  System load:  0.89                  Processes:        126
  Usage of /:   1.3% of 117.11GB    Users logged in:  0
  Memory usage: 6%                  IP addresss for wlan0:***.***.***.***
  Swape usage:  0%


0 packages can be updated.
0 updates are security updates.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
```

Figura 39. Ejemplo de mensaje que puede aparecer en pantalla, indicando que nos encontramos usando ya Ubuntu 18.04.

Para evitar problemas con la configuración wifi, se recomienda seguir usando el cable Ethernet hasta tener el escritorio habilitado. Gracias a este cable, la Raspberry Pi tiene acceso a internet sin necesidad de configurar la wifi. Para asegurar que la última versión del software está disponible se usan los comandos siguientes:

```
sudo apt-get update
sudo apt-get upgrade
```

Tras esto ya se puede instalar el escritorio de Ubuntu, por lo que se introduce:

```
sudo apt-get install ubuntu-desktop
```

Una vez terminado el proceso, se reinicia la Raspberry Pi mediante el comando:

```
sudo reboot
```

Esta vez la Raspberry Pi se iniciará como cualquier ordenador, pidiendo que se escriba la contraseña para acceder ya al entorno del escritorio (ver Figura 40). Ahora en la configuración de red, es posible conectarse a cualquier red wifi disponible, para poder dejar de usar el cable Ethernet y que así el robot se pueda mover libremente. También es posible establecer el idioma y demás configuraciones que el usuario considere importantes para un funcionamiento cómodo del sistema.

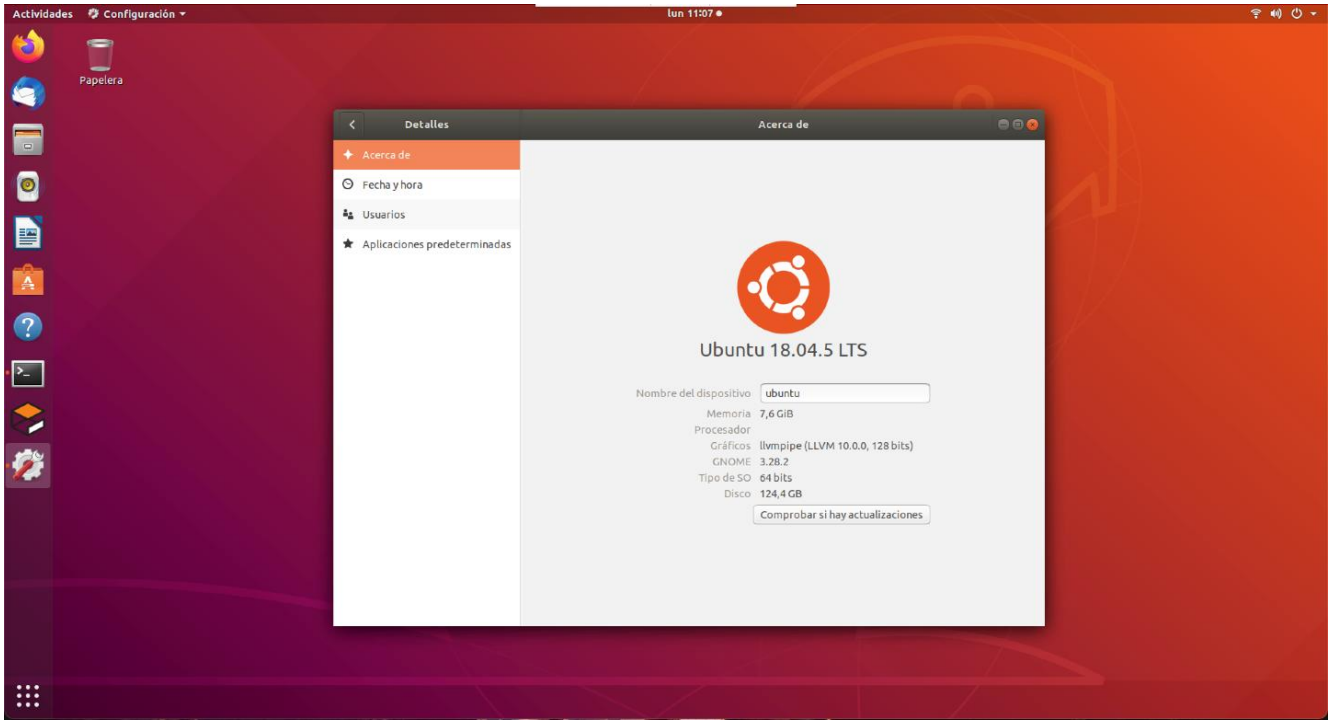


Figura 40. Captura de pantalla del escritorio de Ubuntu y ventana de configuración.

3.3.1 Ecosistema Gazebo/ROS.

Como ya se ha comentado con anterioridad, ROS puede considerarse como una pila de software (un framework de desarrollo), que permite desarrollar e implementar software para robots tanto reales como simulados. Aunque no es un sistema operativo como tal, ROS es capaz de proveer servicios típicos de un sistema operativo estándar como:

- Abstracción de hardware.
- Controladores de dispositivos.
- Bibliotecas (Libraries).
- Herramientas de visualización.
- Comunicación por mensajes.
- Administración de paquetes.

ROS se distribuye bajo la licencia *open source* BSD, lo que lo hace altamente compatible con Ubuntu, como ya se ha mencionado. ROS puede considerarse atendiendo a tres niveles de conceptos, que se detallan a continuación [19]:

Los conceptos de **nivel de sistema de archivos** cubren recursos que se encuentran en el disco, tales como:

- **Paquetes:** son la unidad principal para organizar el software en ROS. Pueden controlar nodos, bibliotecas, conjuntos de datos, archivos de configuración o cualquier cosa que se pueda organizar en conjunto. Los paquetes son el elemento de construcción y lanzamiento más básico en ROS.
- **Metapaquetes:** son paquetes especializados en representar un grupo de paquetes relacionados, son como una especie de marcador.
- **Manifiestos de paquetes:** los *package.xml* proporcionan datos sobre un paquete como su nombre, versión, descripción, información de licencia, dependencias y demás información que pueda contener.
- **Repositorios:** son una colección de paquetes que comparten un sistema VCS común. Los paquetes que comparten un VCS comparten a su vez la misma versión y se pueden lanzar juntos.
- **Tipos de mensajes (msg):** las descripciones de los mensajes incluidas en los paquetes definen las estructuras de datos para los mensajes que envía ROS.
- **Tipos de servicio (srv):** las descripciones de servicio incluidas en los paquetes definen los datos de solicitud y respuesta para servicios en ROS.

El **nivel de gráfico de cálculo** de ROS es la red punto a punto (peer-to-peer), de procesos de ROS que procesan datos de forma conjunta, los conceptos básicos de este nivel son:

- **Nodos:** son procesos que realizan cálculos. Por ejemplo, en un robot móvil habrá un nodo para controlar las ruedas motoras, otro para procesar los datos que aporta el láser, otro para la visión a través de cámara, etc.
- **Maestro:** proporciona el registro de nombres y búsqueda del resto del gráfico de cálculo, por lo que, sin él los nodos no podrían localizarse ni intercambiar mensajes.
- **Servidor de parámetros:** permite el almacenamiento de datos por clave en una ubicación central.
- **Mensajes:** es una estructura de datos que comprende campos escritos. Esto es lo que usan los nodos para comunicarse.
- **Topics:** los mensajes se enrutan a través de un sistema de transporte publicación/suscripción. Un nodo manda un mensaje publicándolo en un tema determinado, este tema es el nombre que utiliza para identificar el contenido del mensaje. El nodo que busque cierto tipo de mensaje se tendrá que suscribir al tema adecuado.
- **Services:** la solicitud/respuesta se hace por medio de los servicios, que se definen mediante un par de estructuras de mensajes, una para solicitud y otra para la respuesta.
- **Bags:** son un formato para guardar y reproducir datos de mensajes, como por ejemplo los datos que aportan sensores como un láser.

En la Figura 41 podemos ver el esquema de comunicación entre dos nodos.

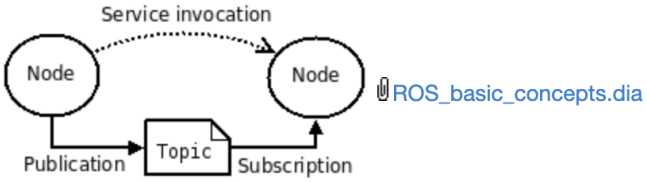


Figura 41. Esquema de comunicación publicación/suscripción entre dos nodos en ROS (wiki.ros.org).

Los conceptos del **nivel de la comunidad** de ROS son recursos que permiten a comunidades separadas de usuarios el intercambio de software y conocimiento. Estos recursos incluyen:

- **Distribuciones:** las distribuciones de ROS son colecciones versionadas para la instalación del software en un sistema operativo, como puede ser Linux.
- **Repositorios:** donde se almacenan los componentes de software de robot desarrollados por la comunidad o empresas que desarrollen software de tipo abierto para sus productos.
- **Wiki ROS:** es el sitio web donde la comunidad puede documentarse sobre la funcionalidad de ROS, estudiar los conceptos de ROS o seguir tutoriales para practicar o entender esos conceptos.
- **ROS Answers:** sitio web de preguntas y repuestas donde la comunidad puede exponer sus casos y se discuten posibles soluciones.

El proceso de desarrollo requiere crear un espacio de trabajo, donde se colocarán los paquetes a utilizar durante la fase de desarrollo y funcionamiento de nuestro robot, ya sea real o simulado. Para crear el espacio de trabajo se deben seguir los siguientes pasos:

Primero hay que instalar la distribución de ROS para el sistema operativo Ubuntu 18.04, a la que le corresponde ROS Melodic. Se debe escribir en la terminal lo siguiente:

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

Después hay que configurar las claves para que el origen sea real:

```
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Actualizamos el conjunto de paquetes:

```
$ sudo apt update
```

Ahora ya es posible instalar la distribución completa de ROS:

```
$ sudo apt install ros-melodic-desktop-full
```

Lo siguiente es establecer el entorno, para ello se escriben los comandos siguientes:

```
$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
$ cat .bashrc
$ source .bashrc
```

Ahora se comprueba que el comando roscd lleva a la carpeta donde se ha instalado ROS:

```
$ roscd
/opt/ros/melodic$
```

A continuación, se instalan las dependencias para la construcción de paquetes de ROS:

```
$ sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential
```

Una vez instalado, se puede inicializar rosdep:

```
$ sudo apt install rosdep
$ sudo rosdep init
$ rosdep update
```

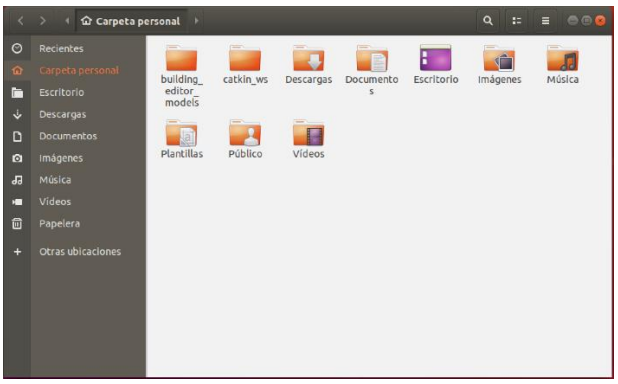
Ya estaría completa la instalación de ROS, pero hay que establecer una serie de configuraciones, como el espacio de trabajo:

```
$ cd /home/ubuntu
```

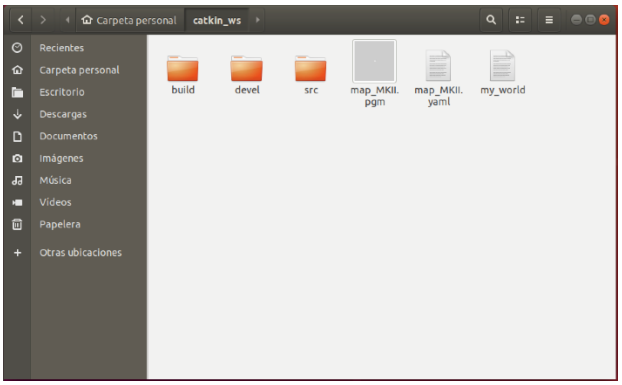
De este modo, la carpeta actual es el directorio de trabajo (ubuntu es el nombre de nuestro usuario). Se prosigue creando la carpeta catkin_ws con:

```
$ mkdir -p catkin_ws/src
$ cd catkin_ws
$ catkin_make
$ echo “source /home/ubuntu/catkin_ws/devl/setup.bash” >> /home/ubuntu/.bashrc
$ source /home/ubuntu/.bashrc
```

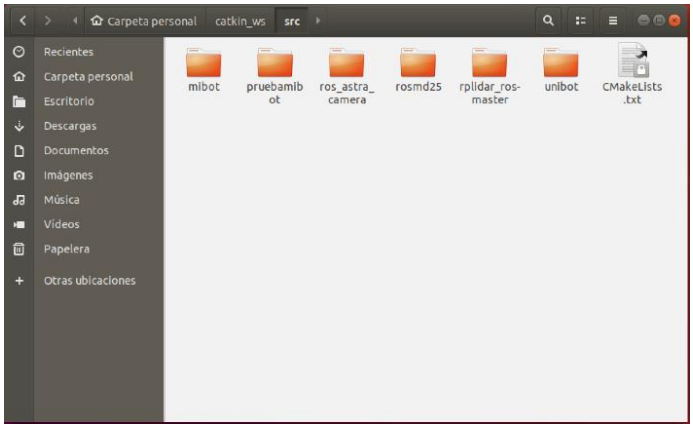
Con esto ya se ha creado el espacio de trabajo, por el que se puede acceder a través de la terminal escribiendo roscd o por medio del escritorio como aparece en la Figura 42.



(a)



(b)



(c)

Figura 42. Captura de pantalla del acceso mediante escritorio al espacio de trabajo y las dependencias donde se ubican los paquetes del robot. a) Carpeta personal del usuario. b) Espacio de trabajo. c) Dependencias de los diferentes paquetes.

Gazebo es un simulador de entornos en 3D que se instala con la versión completa de ROS. Gazebo permite realizar simulaciones de robots y sincronizarlas con ROS para operar como si de un robot real se tratara. Esto es bastante interesante, ya que resulta de gran ayuda poder probar el robot en entornos simulados antes de hacerlo en escenarios reales. En la Figura 43 se puede ver una captura de pantalla del simulador Gazebo.

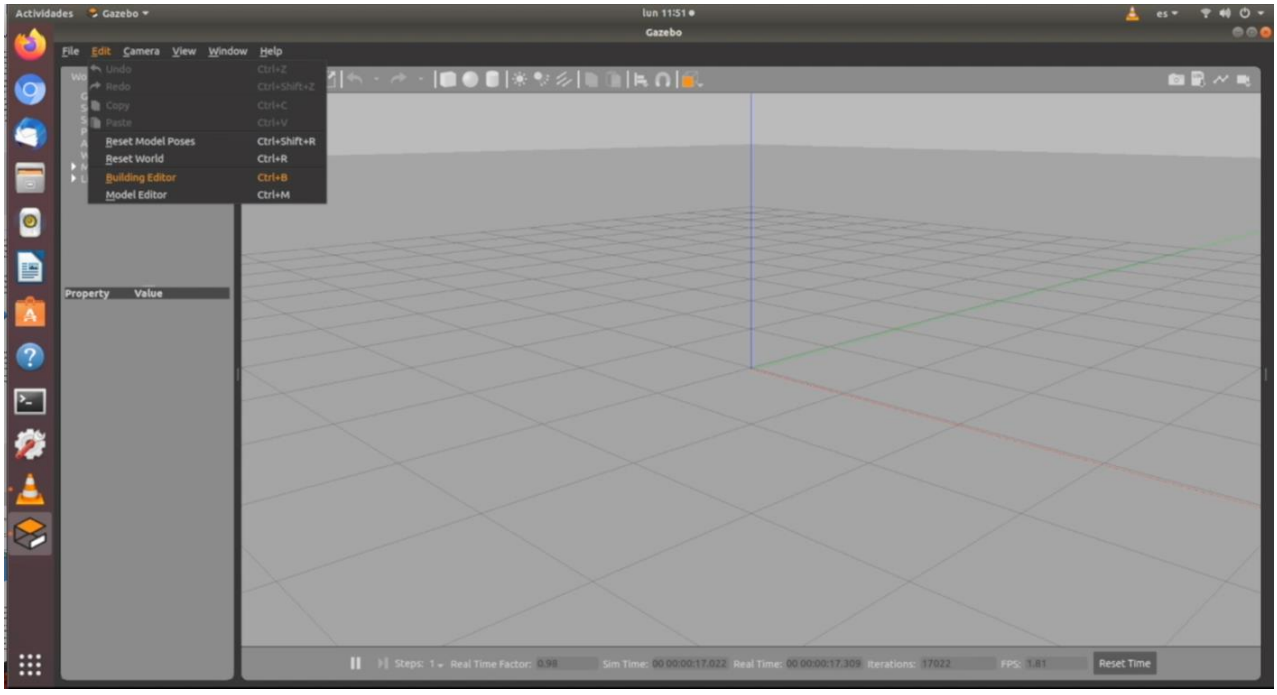


Figura 43. Captura de pantalla que muestra la pantalla de inicio del simulador de entornos 3D Gazebo.

3.3.2 Diseño de componentes para el robot real.

En esta sección se describe el proceso de integración de todos los elementos que componen el robot real, a nivel de software, dentro de ROS.

El sistema de locomoción se controla mediante la placa MD25. El driver ROS de esta placa se denomina ***rosmd25*** y ha sido desarrollado y proporcionado por la Dra. Nieves Pavón. Se han realizado, además, las modificaciones necesarias para permitir que el robot esté dotado de una mejor maniobrabilidad. El driver contiene un nodo, que se debe incluir en el archivo *launch* que permitirá la ejecución de todos los nodos necesarios para que el sistema de navegación completo funcione adecuadamente. Este nodo se encarga de los cálculos y la publicación de la odometría en el *topic odom*, y también es el encargado de establecer los comandos de velocidad para que el robot se mueva.

Para la navegación del robot, ROS dispone de un paquete llamado *Navigation* [20]. Este paquete cubre:

- Publicación de datos de sensores.
- Publicación de información de odometría.
- Configuración de las transformadas.
- Construcción de un mapa.
- Localización.
- Planificación global y local de caminos.

En realidad, los procesos de construcción de mapas, localización y planificación están realmente implementados mediante paquetes independientes que se combinan para dar el soporte necesario al paquete *Navigation.*

Para la construcción del mapa se emplea ***slam_gmapping***, un nodo de ROS del paquete *Gmapping*. Este nodo toma los datos publicados en el *topic scan* por el láser, que junto con la información aportada por los *frames baselink* y *odom* (un *frame* es el sistema de coordenadas de un objeto o del conjunto global del sistema), permite la construcción de un mapa en 2D de la estancia en la que se encuentra el robot. Para que el mapa sea lo más completo posible, el robot debe recorrer la habitación para que el láser recoja el mayor número de datos de la geometría de la habitación. Para ello se usa el paquete de ROS ***teleop_twist_keyboard*** [21]. Este paquete permite teleoperar el robot mediante las teclas del teclado del ordenador. Para usarlo se tiene que escribir en una terminal el siguiente comando:

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Lo que inicia el paquete. Cuando se termine de iniciar, aparecerá en la terminal el mensaje que se muestra en la Figura 44, indicando la función de cada tecla y que ya está operativa la teleoperación del robot.

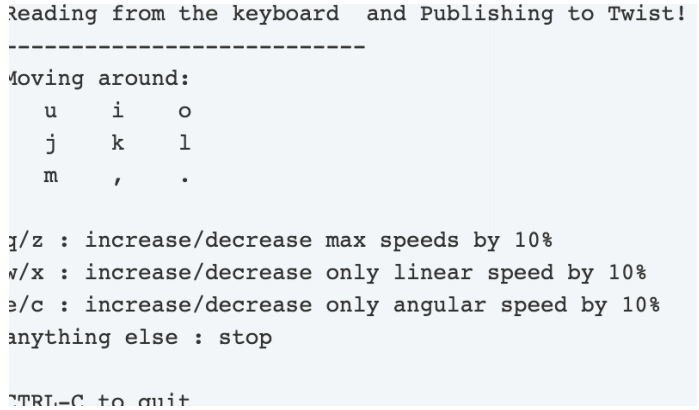


Figura 44. Mensaje por terminal de los controles para teleoperar el robot mediante *teleop_twist_keyboard*.

Una vez hecho el mapa, se guarda usando el paquete ***map_server***. Para guardar el mapa, se escribe lo siguiente en una terminal:

```
$ rosrun map_server map_saver -f map
```

Este comando genera y guarda los archivos *map.pgm* y *map.yaml* en la carpeta personal del usuario. El archivo de tipo *pgm* contiene la imagen del mapa, mientras que el archivo tipo *yaml* contiene la información necesaria para la navegación por ese mapa.

El paquete AMCL de ROS es un sistema probabilístico de localización para la movilidad de un robot en 2D [22]. Este paquete usa un filtro de partículas para rastrear la pose del robot por un mapa conocido. Para incluir estas funciones en el robot se deben crear dos archivos tipo *launch*, que se incluirán en el *launch* final del robot junto con el nodo *move_base*. El archivo ***amcl_only.launch*** tiene la siguiente forma:

```
<launch>
  <node pkg="amcl" type="amcl" name="amcl" output="screen">
  <remap from="scan" to="scan"/>
  <param name="odom_frame_id" value="odom"/>
  <param name="odom_model_type" value="diff-corrected"/>
  <param name="base_frame_id" value="baselink"/>
  <param name="update_min_d" value="0.05"/>
  <param name="update_min_a" value="0.05"/>
  <param name="min_particles" value="1000"/>
  <param name="global_frame_id" value="map"/>
  <param name="tf_broadcast" value="true" />
  <param name="initial_pose_x" value="0.0"/>
  <param name="initial_pose_y" value="0.0"/>
  <param name="initial_pose_a" value="0.0"/>
 </node>
</launch>
```

Este archivo contiene todos los parámetros que AMCL necesita. Para el mapa se crea el archivo ***map_server.launch*** que tiene la forma:

```
<launch>
  <arg name="map_file" default="/home/ubuntu/catkin_ws/map.yaml"/>      <!-- path of map file -->
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" respawn="true" />
</launch>
```

Para la configuración del nodo *move_base* se tiene que crear una carpeta llamada *config*, donde se crean cuatro archivos que contienen la configuración de las diferentes partes que componen la navegación autónoma del robot. En el archivo ***costmap_common_params.yaml*** se establecen los parámetros comunes del mapa global y local en la forma:

```
obstacle_range: 6.0
raytrace_range: 8.5
footprint: [[0.12, 0.14], [0.12, -0.14], [-0.12, -0.14], [-0.12, 0.14]]
map_topic: /map
subscribe_to_updates: true
observation_sources: laser_scan_sensor
laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan, topic: scan, marking: true, clearing: true}
global_frame: map
robot_base_frame: baselink
always_send_full_costmap: true
```

Los parámetros para la localización global se introducen en el archivo ***global_costmap_params.yaml*** en la forma:

```
global_costmap:
        update_frequency: 2.5
        publish_frequency: 2.5
        transform_tolerance: 0.5
        static_map: true
        rolling_window: true
        width: 15
        height: 15
        origin_x: -7.5
        origin_y: -7.5
        resolution: 0.1
        infilation_radius: 2.5
```

Los parámetros para la localización local se introducen en el archivo ***local_costmap_params.yaml*** en la forma:

```
local_costmap:
        update_frequency: 5
```

```
        publish_frequency: 5
        transform_tolerance: 0.25
        static_map: false
        rolling_window: true
        width: 3
        height: 3
        origin_x: -1.5
        origin_y: -1.5
        resolution: 0.1
        infilation_radius: 0.6
```

Por último, en el archivo ***trayectory_planner.yaml***, se establecen los parámetros necesarios para el cálculo de trayectorias para la navegación autónoma en la forma:

```
TrayectoryPlannerROS:
        max_vel_x: 0.2
        min_vel_x: 0.1
        max_vel_theta: 0.35
        min_vel_theta: -0.35
        min_in_place_vel_theta: 0.25
        acc_lin_theta: 0.25
        acc_lin_x: 2.5
        acc_lin_y: 2.5
        holonomic_robot: true
        meter_scoring: true
        xy_global_tolerance: 0.15
        yaw_global_tolerance: 0.25
```

El nodo ***move_base*** se debe incluir, junto con los otros dos archivos tipo *launch* en el archivo *launch* final del robot. Se añaden en la forma:

```
<launch>
…
<node pkg=”move_base” type=”move_base” name=”move_base” output=”screen”>
        <param name=”controller_frequency” value=”20.0”/>
        <rosparam file=”$(find robot_gazebo)/config/costmap_common_params.yaml” command=”load” ns=”global_costmap” />
        <rosparam file=”$(find robot_gazebo)/config/costmap_common_params.yaml” command=”load” ns=”local_costmap” />
        <rosparam file=”$(find robot_gazebo)/config/local_costmap_params.yaml” command=”load” />
        <rosparam file=”$(find robot_gazebo)/config/global_costmap_params.yaml” command=”load” />
        <rosparam file=”$(find robot_gazebo)/config/trayectory_planner.yaml” command=”load” />
</node>
<include file=”$(find robot_gazebo)/launch/map_server.launch” />
<include file=”$(find robot_gazebo)/launch/amcl_only.launch” />
```

```
</launch>
```

Los paquetes para el láser RPLidar A1 y la cámara Orbbec Astra Pro para ROS están disponibles en GitHub, la página que utiliza ROS.org y la comunidad de usuarios para subir los paquetes de ROS de dispositivos compatibles y proyectos que se han desarrollado. Para instalar estos paquetes basta con descargar el archivo comprimido de la web y descomprimirlo en el espacio de trabajo que se haya creado para ROS, *catkin_ws*.

3.3.3 Diseño de componentes para simulación.

Gazebo se ha empleado para la construcción del modelo del robot, para realizar pruebas, comprobar su correcto funcionamiento en una simulación y poder visualizarlo correctamente en RViz.

RViz es una herramienta gráfica que permite visualizar los diferentes mensajes de los *topics* que se están generando por los nodos en ejecución, tanto de un robot real como de una simulación. En RViz se puede visualizar el funcionamiento completo del robot: los datos del láser, el modelo del robot, los datos de la cámara, los datos de odometría, el mapa, etc. Esta herramienta es especialmente útil para la comprobación del correcto funcionamiento del robot. En la Figura 45 se muestra la visualización del modelo del robot a través de RViz.

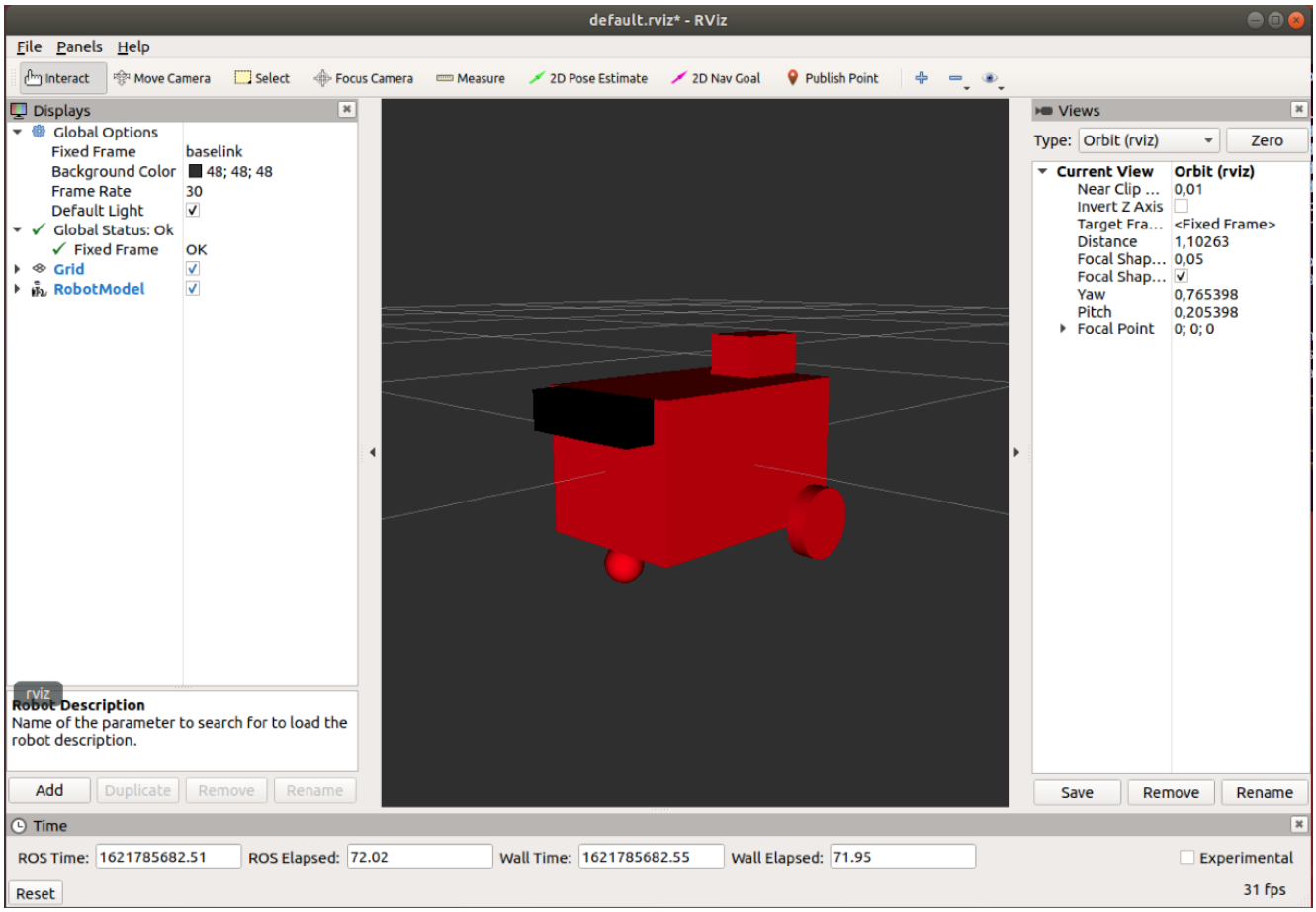


Figura 45. Visualización del modelo del robot a través de RViz.

El modelo del robot que muestra la Figura 45 es la visualización del archivo tipo *urdf* (formato de lenguaje para describir robots) del paquete del robot que se ha creado. En este archivo se encuentra la siguiente información:

- Definición de parámetros geométricos de las diferentes partes que conforman el modelo: base, rueda izquierda, rueda derecha, rueda castor, láser, cámara.
- Estimaciones de los momentos de inercia de las diferentes partes del modelo.
- Definición de los *links*, las partes del modelo. Para cada *link* se define su geometría, inercia y límites de colisión.
- Definición de los diferentes *joints*. Los *joints* son el tipo de unión que presenta cada parte del modelo.

Si hay algún *topic* en el que se esté publicando información por parte de algún nodo, RViz lo puede visualizar. Por ejemplo, en la Figura 46 se muestra los datos que proporciona el láser cuando está barriendo la habitación. Estos datos se encuentran en el *topic scan*, al que se suscribe *LaserSan* en RViz para poder visualizarlos.

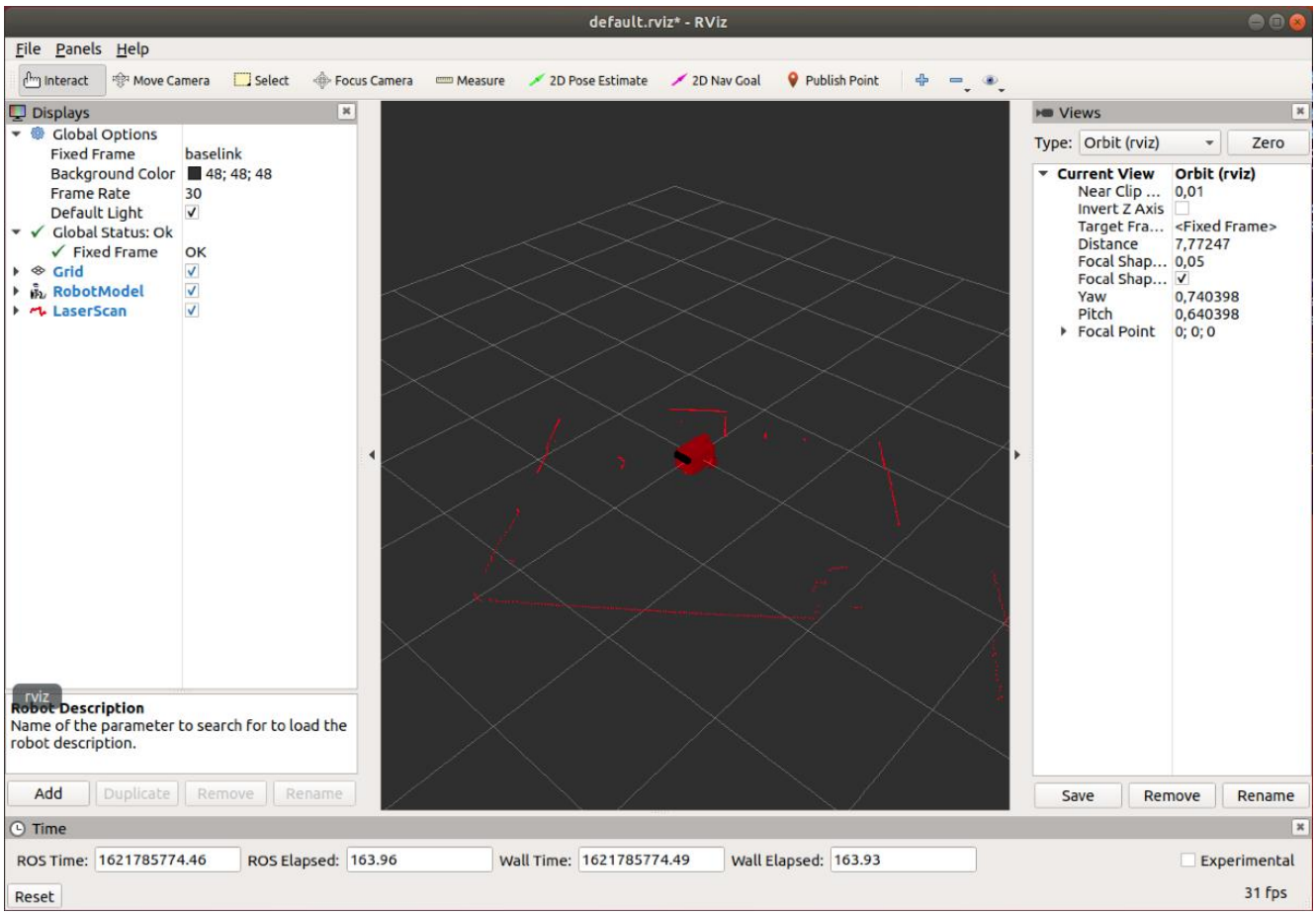


Figura 46. Visualización de los datos del láser a través de RViz.

3.3.4 Diseño de componentes para interacción humano-robot.

En esta sección se describen los componentes adicionales, a nivel de software, que se han integrado para realizar las tareas de reconocimiento de personas y reconocimiento y síntesis de voz.

1) Métodos de Deep Learning.

DL o aprendizaje profundo se define como un algoritmo automático estructurado o jerárquico que emula el aprendizaje humano con el fin de obtener ciertos conocimientos. Destaca porque no requiere de reglas programadas previamente, sino que el propio sistema es capaz de aprender por sí mismo para efectuar una tarea a través de una fase previa de entrenamiento [25].

El DL es un método de ML que se basa en el uso de **redes neuronales artificiales** profundas altamente conectadas para el procesamiento de información, principalmente para la automatización de análisis predictivos, clasificación o regresión, entre otros. En el apartado 2.3.5 ya se explicó el funcionamiento y composición de las redes neuronales. Las redes neuronales pueden ser virtuales, creadas como un software dentro de un ordenador como TensorFlow o Keras, pero también pueden construirse con silicio de manera física. En este proyecto, se utilizarán redes neuronales programadas mediante Keras sobre el framework de TensorFlow en Python.

Para el entrenamiento de la máquina se combina un **aprendizaje supervisado**, en el que el usuario humano etiqueta los datos, con un **aprendizaje no supervisado**, en el que la máquina usa los datos aportados para establecer relaciones y crear patrones.

Como se ha comentado, los métodos de DL están presentes en multitud de sistemas como:

- **Traductores inteligentes**: como el servicio Google Translate, que emplea esta tecnología para que el sistema aprenda de las traducciones corregidas.
- **Lenguaje natural hablado y escrito**: los asistentes virtuales como Siri de Apple.
- **Reconocimiento de voz**: como Alexa de Amazon, que te permite realizar compras a la plataforma por medio de comandos de voz.
- **Interpretación semántica**: como los chatbots, que pueden contestar de forma automática a los mensajes.
- **Reconocimiento facial**: como la tecnología que emplea Apple en los modelos más recientes de iPhone para seguridad y desbloqueo del teléfono.
- **Visión computacional**: el buscador de imágenes de Google emplea esta tecnología para encontrar las imágenes que concuerdan con la búsqueda que se realiza.

2) Tensorflow.

TensorFlow es una plataforma que permite construir y entrenar redes neuronales, como se describió en el apartado 2.3.6. TensorFlow se empleará junto con la biblioteca Keras (descrita en el apartado 2.3.7), para la creación y entrenamiento de modelos de redes neuronales para el reconocimiento y síntesis de voz y el reconocimiento facial. Para el reconocimiento y síntesis de voz se empleará el modelo desarrollado por el compañero Roberto Oterino Bono [27].

3) Reconocimiento y síntesis de voz.

Para el **reconocimiento de voz** se ha optado por emplear directamente la librería *DeepSpeech*, que permite la implementación de modelos de TensorFlow Lite. El modelo que viene incorporado está desarrollado para el inglés, pero existe un repositorio denominado *DeepSeech-Polyglot* que cuenta con un modelo en español con más de 600 horas de entrenamiento [27].

La librería *DeepSpeech* cuenta con las siguientes funciones:

- *Model (model_path)* se emplea para cargar el modelo.
- *createStream( )* permite comenzar la inferencia del audio.
- *feedAudioContent(audio_buffer)* proporciona la señal de audio, que debe ser de 16 bits mono raw.
- *finishStream( )* que calcula la decodificación final de una inferencia de transmisión continua y devulve el resultado.

Para transformar el audio en texto, primero se debe grabar y procesar el audio. Para ello se emplea la librería *PyAudio*, que cuenta con:

- *PyAudio ( )* para crear un objeto de clase *PyAudio*.
- *open ( )* que permite configurar la captura de audio en base a una serie de parámetros.
- *start_stream ( )* que inicia la captura de audio.
- *is_active ( )* que indica si la captura de audio está activa.
- *stop_stream ( )* que para la captura de audio.
- *close ( )* que cierra la captura de audio.
- *terminate ( )* que se usa para eliminar un objeto creado de clase *PyAudio*.

Conforme se captura el audio, se realiza la inferencia y la transcripción se produce de forma continua hasta que la captura de audio termina. El código para la transcripción de voz a texto quedaría:

```
self.__model = deepspeech.Model(model_path)

def process_audio(self,in_data, frame_count, time_info, status): data16 = np.frombuffer(in_data, dtype=np.int16) self.__context.feedAudioContent(data16)
```

```
    return (in_data, pyaudio.paContinue)
def run_inference(self):
    while self.__stream.is_active():
        time.sleep(0.1)
        a = input()
        if a == 's':
            self.__stream.stop_stream()
            self.__stream.close()
            self.__audio.terminate()
            print('Finished recording.')
            text = self.__context.finishStream()
            print('Final text = {}'.format(text))
            break
return text
```

Para la **síntesis de voz** se emplea la librería *pyttsx3*, que utiliza los paquetes de idioma instalados en el sistema operativo del dispositivo, lo que le otorga un sonido natural, a diferencia de otros sintetizadores con voz robótica. Cuenta con la cumplimentación de la librería *talkey*, que reduce considerablemente las líneas de código a escribir. El código para la síntesis de voz a partir de texto quedaría:

```
tts = talkey.Talkey()
tts.say("No le reconozco, pruebe de nuevo", 'es')
```

Una vez transcrita la voz a texto, se puede iniciar el proceso de **reconocimiento del lenguaje natural**, que es lo que consigue que la máquina comprenda ese texto. La librería *spacy-udpipe* junto con el modelo *spanish-gsd-ud-2.5-191206.udpipe*, permite analizar frases en español. Esta librería cuenta con las funciones:

- *spacy_udpipe.load(model name)* para crear un objeto de clase spacy en del idioma que se haya seleccionado como parámetro.
- *token.pos_* que permite acceder al etiquetado gramatical de una palabra.
- *token.lemma_* que permite obtener la raíz semántica de una palabra determinada.

Para analizar una frase, hay que crear un objeto *spacy* e introducirle el texto a interpretar. Este objeto descompone la frase en *tokens* (palabras), que tienen una serie de atributos que se usarán para definir las reglas lógicas para su interpretación. Al establecer las reglas lógicas se debe tener en cuenta que el robot va a recibir órdenes del tipo “ve a un lugar” o “trae algo de tal sitio”. Por lo que debe distinguir entre “ir” e “ir y volver”. El código para el reconocimiento del lenguaje natural de un mensaje quedaría:

```
self.__model = spacy_udpipe.load("es-gsd")
def analyze_text(self, text):
    verbo = objeto = lugar = ""
    dic = {}
```

```
    doc = self.__model(text)
    for token in doc:
        dic[token] = token.pos_
    for i in range (0, len(dic)):
        if i == 0:
            if list(dic.values())[i] == 'ADV': donde = True
            else: donde = False
        if list(dic.values())[i] == 'VERB':
            verbo = str(list(dic.keys())[i].lemma_)
        if list(dic.values())[i] == 'PROPN':
            lugar += str(list(dic.keys())[i])
        if list(dic.values())[i] == 'NOUN' and donde == True:
            lugar += str(list(dic.keys())[i])
        elif list(dic.values())[i] == 'NOUN' and donde == False:
            if list(dic.values())[i-2] == 'ADP' or list(dic.values())[i-1] == 'ADP':
                    lugar = str(list(dic.keys())[i])
            else: objeto = str(list(dic.keys())[i])
    return verbo, objeto, lugar
```

4) Reconocimiento de personas.

Para crear un modelo de red neuronal de TensorFlow para el **reconocimiento facial**, se usará como base el algoritmo y esquema de red existente en el tutorial avanzado de TensorFlow para la clasificación de imágenes [28]. Este modelo muestra cómo clasificar imágenes de flores, usando un modelo de Keras, obteniendo:

- Una carga eficiente de datos fuera del disco.
- Poder identificar el sobreajuste y aplicar técnicas para mitigarlo.

Este ejemplo sigue un flujo de trabajo básico de aprendizaje automático:

i) Examinar y comprender el conjunto de datos.
ii) Construir una canalización de entrada.
iii) Construir el modelo.
iv) Entrenar el modelo.
v) Probar el modelo.
vi) Mejorar el modelo.

En el Anexo VIII se detalla el desarrollo de este ejemplo de modelo de TensorFlow. El modelo para el reconocimiento facial se crea en un nuevo archivo de Python. Se comienza importando TensorFlow y otras bibliotecas necesarias, como Keras.

```
import matplotlib.pyplot as plt
```

```
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import pathlib
```

Se emplea un conjunto de datos con dos categorías: *adrian* y *noadrian*. En *adrian* se encuentran las imágenes con las que el sistema será capaz de identificar al usuario (en este caso a Adrián), mientras que en la carpeta *noadrian* se encuentran imágenes de otras personas, para que el sistema sea capaz de identificar que, efectivamente, esas personas no son el usuario. Para importar el conjunto de datos se escribe:

```
data_dir = pathlib.Path('/Users/adriansanchezgranero/Documents/TFG/Faces')
print(data_dir)
```

Para la creación del conjunto de datos se ha empleado un algoritmo de MATLAB para recortar la cara que aparece en la imagen. El algoritmo de Matlab se muestra a continuación:

```
% Initialize the detector
faceDetector = vision.CascadeObjectDetector;
shapeInserter = vision.ShapeInserter('BorderColor','Custom','CustomBorderColor',[0 255 0]);
% Read the image
k = 100;
numero = 101;
for j=1:7
   nombrenumero = numero+j;
   nombre = ".\" + nombrenumero + ".jpg";
   fprintf("%s\n", nombre);
   I = imread(nombre); % Read your image here
   imshow(I);
   bbox = step(faceDetector, I);
   % Draw boxes around detected faces and display results
   I_faces = step(shapeInserter, I, int32(bbox));
   figure;imshow(I_faces), title('Detected faces');
   % Cropping individual faces
   for i=1:size(bbox,1)
      face = imresize(imcrop(I,bbox(i,:)),[128 128]);
      figure;imshow(face,[]);
      k = k + 1;
      nombrefoto = "c"+k+".jpg";
```

```
      imwrite(face, nombrefoto);
   end
end
```

Para ampliar el conjunto de datos hasta una cantidad óptima para el entrenamiento se ha empleado un algoritmo de MATLAB para rotar las imágenes resultantes anteriores, obteniendo así a partir de una imagen todas sus rotaciones posibles, aumentando considerablemente el conjunto de datos de que dispondrá el modelo. El algoritmo de MATLAB para rotar las imágenes es el siguiente:

```
nombre = "./noadrian/c107"
extension = ".jpg";
im = imread(nombre+extension);
for i=1:359
  imrotada = imrotate(im, i);
  nombrefoto = nombre + "girada" + i + extension;
  imwrite(imrotada, nombrefoto);
end
```

Con el conjunto de datos resultante se generarán unas imágenes de 100x100 para que el modelo las emplee en el entrenamiento y validación. Se escribe:

```
batch_size = 32
img_height = 100
img_width = 100
```

El 60% de las imágenes va destinado a entrenamiento, mientras que el 40% restante se empleará para validar el modelo. Para crear el conjunto de entrenamiento y el conjunto de validación se escribe lo siguiente:

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.4,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.4,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

Se deben estandarizar los datos de entrada, ya que los valores del canal RGB de las imágenes están en el rango [0, 255] y una red neuronal debe de tener unos valores de entrada pequeños para un mayor rendimiento. Por ello, se estandarizan los valores para el rango [0, 1].

```
from tensorflow.keras import layers
normalization_layer = tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
```

A continuación, se define el número de clases y el modelo. El modelo consta de una capa entrada, cuatro capas ocultas y una capa de salida. En las capas ocultas se realiza el paso de tamaño de imagen a 32x32 y la aplicación de filtros.

```
num_classes = 2
model = tf.keras.Sequential([
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(64, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(60, activation='relu'),
  layers.Dense(num_classes)
])
```

El algoritmo de aprendizaje empleado es *adam*, para ajustar automáticamente el parámetro de aprendizaje que indica si se está aprendiendo demasiado rápido o a una velocidad óptima. La métrica que se utiliza es la de precisión.

```
model.compile(
  optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
```

Para entrenar el modelo de escribe lo siguiente:

```
epochs = 2


history = model.fit(
  train_ds,
  validation_data=val_ds,
```

```
  epochs=epochs
)

model.summary()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)
```

Para guardar el modelo se escribe:

```
print("Guardando el modelo")
model.save('/Users/adriansanchezgranero/Documents/TFG/model/mimodeloguardado')
```

Para comprobar el funcionamiento del modelo, se carga el modelo junto con una imagen que no estuviera incluida en el conjunto de datos de entrenamiento o de validación.

```
adrian_path = "/Users/adriansanchezgranero/Documents/TFG/Muestras foto cara/adrian1.png"

img = keras.preprocessing.image.load_img(
    adrian_path, target_size=(img_height, img_width)
)

print("Voy por aquí")

img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

El modelo contestará imprimiendo por pantalla la clase a la que pertenece la imagen (*adrian* o *noadrian*) junto con el porcentaje de confianza. En el Anexo IX se muestra el código completo del modelo.

El modelo queda entrenado y funcional para futuras aplicaciones, pero limitado al uso desde el ordenador. Para poder implementar el modelo en un dispositivo móvil o un SBC Raspberry Pi, se debe hacer una conversión del modelo a TensorFlow Lite, que es un formato optimizado. Para convertir un modelo guardado se emplean las siguientes líneas de código en un archivo Python [29]:

```
import tensorflow as tf
# Convert the model
converter = tf.lite.TFLiteConverter.from_saved_model(/Users/adriansanchezgranero/Documents/TFG/model/mimodeloguardado)
tflite_model = converter.convert()
# Save the model.
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

Esto genera un archivo TensorFlow Lite que contiene la conversión del modelo guardado anteriormente. Ahora el modelo se podría implementar en un dispositivo móvil o una SBC por medio del archivo generado y un "intérprete" para la plataforma específica en la que se va a implementar [30]. Para el ejemplo de una SBC Raspberry, el archivo TensorFlow Lite se podría ejecutar con Python.

3.4 Integración de todos los componentes.

Una vez descrito el funcionamiento de los diferentes sistemas que conforman el robot, es necesario comentar como se integran unos con otros.

Para la parte de ROS, se dispone de diferentes nodos, cada uno enfocado a una tarea específica:

- El nodo r**osmd25** se encarga de controlar las ruedas motoras.
- El nodo **move_base** se encarga de la navegación autónoma del robot.
- El nodo **rplidar** se encarga de publicar la información que registra el láser en el *topic scan*.
- El nodo **astra_camera** se encarga de publicar la información que registra la cámara Orbbec Astra Pro en los *topics* correspondientes.
- El nodo **map_server** carga el mapa.
- El nodo **amcl** se encarga de localizar al robot en el mapa.

Todos estos nodos se incluyen dentro del archivo tipo *launch* que se encarga de lanzar el robot, cargando toda la información del robot e iniciando todos los nodos. Para visualizar los diferentes *topics* y la información que aportan, se utiliza RViz. Este visualizador permite ver lo que detecta el robot y la información que se está cargando, como el mapa, la imagen que recibe la cámara o lo que el láser está escaneando. También permite dar órdenes de navegación, marcando puntos de destino o una trayectoria concreta.

Para la parte de inteligencia artificial y DL se emplea TensorFlow. TensorFlow se encargará del reconocimiento facial y del reconocimiento y síntesis de voz. Al igual que ROS, los modelos ligeros de TensorFlow se ejecutan en el SBC Raspberry Pi. El entrenamiento de los modelos, sin embargo, se ha realizado mediante un programa escrito en Python usando un ordenador convencional. El proceso de entrenamiento también se puede llevar a cabo en Google Colab, que es un entorno gratuito que permite escribir y ejecutar código en la nube. Cuenta con herramientas como TensorFlow o Keras entre otras.

Como todos los sistemas que componen al robot se han integrado dentro de la Raspberry Pi y ésta está incorporada al robot, es necesaria alguna forma de poder acceder a esta placa. Para ello se emplea el escritorio remoto. Esto permite visualizar el entorno del escritorio de la Raspberry Pi y todas sus aplicaciones desde otro ordenador. Para instalar el escritorio remoto se ha usado VNC, que crea una relación servidor-cliente entre la Raspberry Pi y el ordenador desde el que se la quiere controlar. Primero se debe instalar el servidor VNC en la Raspberry Pi, se escribe lo siguiente en la terminal de comandos de la placa:

```
$ sudo apt install x11vnc
```

Una vez completada la instalación, hay que establecer la contraseña de acceso al servidor:

```
$ x11vnc -storepasswd
Enter VNC password:
Verify password:
Write password to /home/ubuntu/.vnc/password? [y]/n
```

Tras escribir la contraseña y verificarla escribiéndola otra vez, al escribir “y” habrá establecido la contraseña para el servidor de la Raspberry Pi. Para activar el servidor se escribe lo siguiente en una terminal:

```
$ x11vnc -usepw
```

En la Figura 47 se muestra la respuesta al comando anterior. En la terminal se puede ver que el escritorio de VNC que se ha activado se encuentra en el puerto 5901. Esta información es crucial para enlazar correctamente el escritorio remoto.

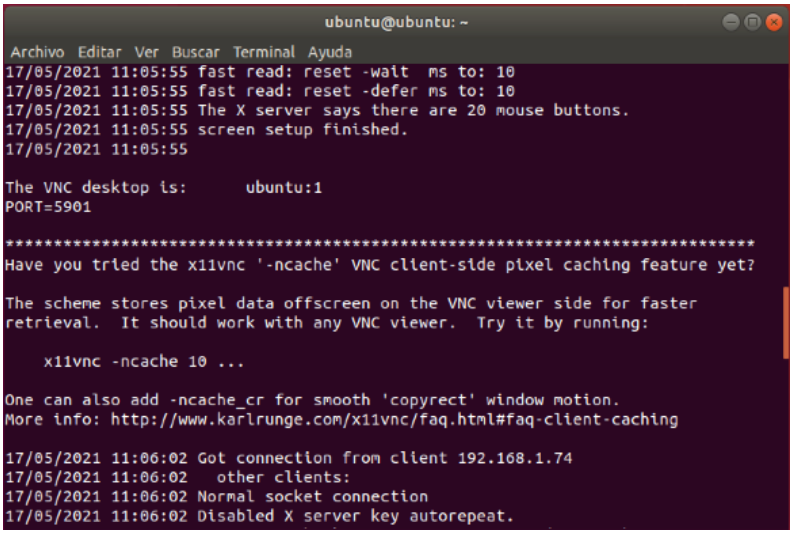


Figura 47. Activación del escritorio remoto VNC a través de la terminal de la Raspberry Pi.

Ya está activo el escritorio remoto en la Raspberry Pi, solo falta conectar el ordenador con el que se quiera usar a este escritorio. Si x11vnc es el servidor y el retransmite los datos, se necesita un visualizador para usar el escritorio remoto en el ordenador. El visualizador empleado es VNC Viewer, un visualizador gratuito que se puede instalar de forma sencilla y está disponible para multitud de sistemas operativos. Una vez instalado, se inicia y aparecerá una ventana como la que muestra la Figura 48.

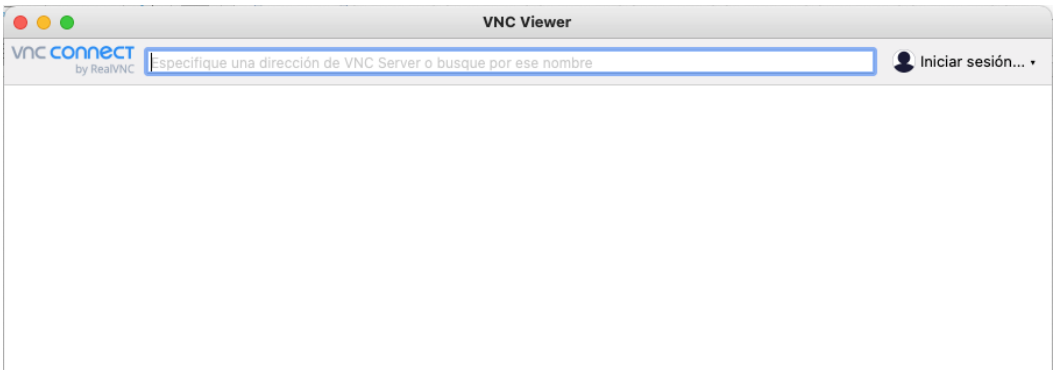


Figura 48. Ventana de inicio de VNC Viewer.

Para conectar el escritorio remoto, se escribe la dirección IP de la Raspberry Pi y el puerto que utiliza x11vnc para retransmitir, en la forma IP:PORT. Una vez escrito, se pulsa la tecla intro y debería aparecer algo similar a la Figura 49.

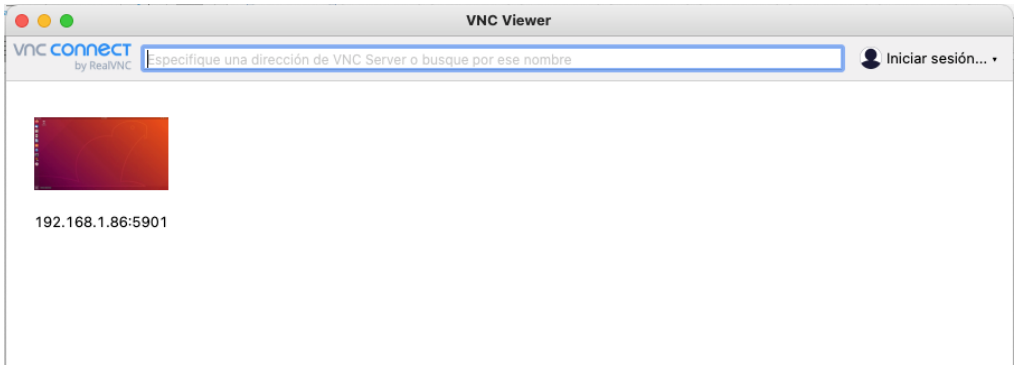


Figura 49. Selección del escritorio remoto.

Haciendo doble click en el icono, aparecerá una ventana en la que solicita la contraseña para conectar con el servidor x11vnc. Una vez escrita la contraseña, se conectará a la Raspberry Pi y en una ventana nueva aparecerá el escritorio, permitiendo interaccionar con él de la misma manera que si se hiciera directamente con la Raspberry Pi conectada a un monitor. Mientras no se cierre VNC Viewer o la terminal en la que se está ejecutando x11vnc en la Raspberry Pi, el escritorio remoto estará activo y funcionando, como se muestra en la Figura 50.

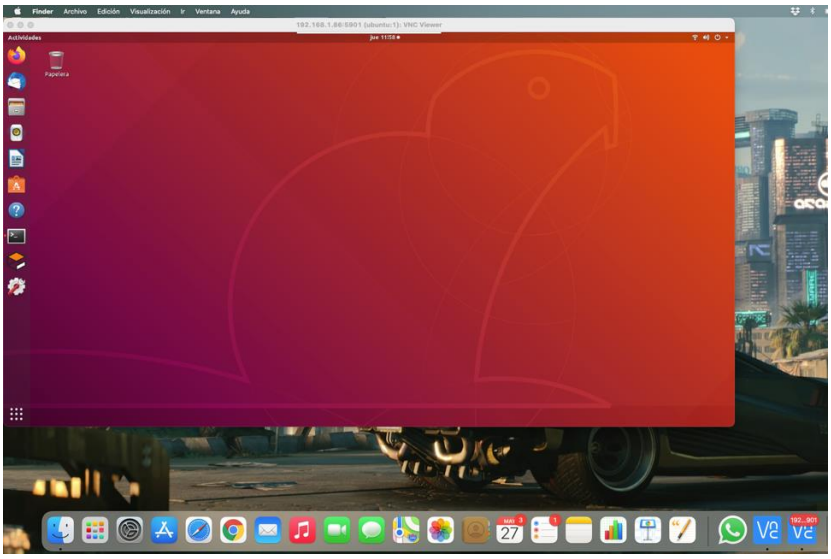


Figura 50. Captura de pantalla que muestra la ventana en la que se encuentra el escritorio remoto.

De esta forma se puede controlar la Raspberry Pi y por tanto al robot desde otro ordenador, para que no haya cables que impidan que se mueva libremente.

4 Análisis de resultados.

En el presente capítulo se presentan y describen las diferentes pruebas que se han llevado a cabo para comprobar el funcionamiento de todas las características que presenta el robot. Se puede distinguir entre pruebas de navegación y pruebas de interacción humano-robot. También se comentan los resultados obtenidos y se discute la importancia y repercusión de los mismos.

4.1 Pruebas diseñadas.

Para realizar las pruebas de navegación se ha empleado el entorno de ROS, que permite el control y monitorización tanto de la simulación como del robot real. Dentro de las pruebas de navegación se puede distinguir entre las pruebas en el entorno de simulación de Gazebo y las pruebas realizadas con el robot real.

Para las pruebas de interacción humano-robot se emplean modelos de TensorFlow. Para entrenar estos modelos se emplea Google Colab, un servicio de Google Drive que permite ejecutar y programar código Python con las ventajas: no requiere configuración, es gratuito y permite compartir y acceder a contenido de forma fácil.

4.1.1 Pruebas de navegación.

Con estas pruebas se busca la navegación autónoma del robot por un mapa conocido. Se puede distinguir entre pruebas con la simulación y pruebas con el robot real.

Pruebas con el entorno de simulación de Gazebo.

Para las pruebas de navegación en la simulación se crea un paquete en el espacio de trabajo de ROS *catkin_ws*. En este paquete se encuentran dos carpetas:

- robot_description: aquí se encuentra el modelo del robot con sus propiedades físicas y parámetros para su simulación.
- robot_gazebo: aquí se encuentran los diferentes *launch* y configuraciones para lanzar la simulación del robot.

Dentro de la carpeta *robot_description* en la carpeta *urdf* hay dos archivos: *xacro* y *gazebo*. En el archivo *xacro* se encuentran todos los aspectos físicos del modelo del robot. En el Anexo I se muestra el contenido del archivo. En el archivo *gazebo* se encuentra la información que se necesita para simular el robot y se definen las propiedades de los dos sensores: la cámara y el láser. En el Anexo II se muestra el contenido de este archivo.

Una vez definido el modelo del robot para la simulación, hay que crear un mundo para probar el robot. Gazebo cuenta con un editor de mundos, con el que se pueden crear estancias o circuitos de prueba para el robot. En la Figura 51 se muestra el uso del editor de mundos de Gazebo.

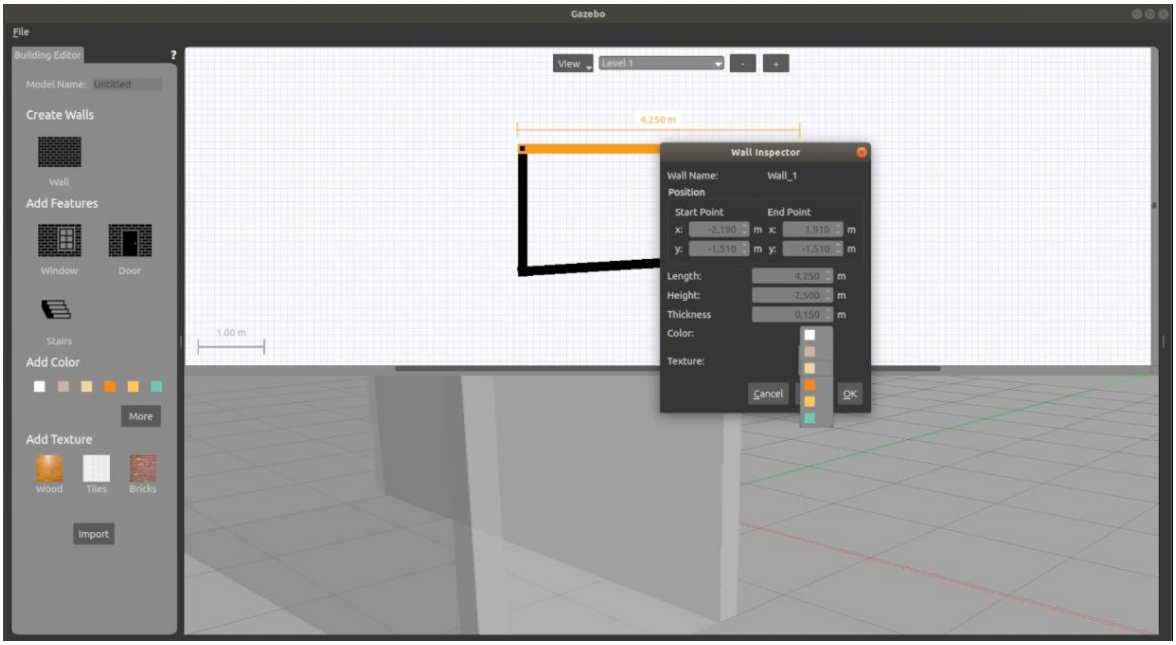


Figura 51. Editor de mundos de Gazebo.

Una vez creada la estancia, se guarda el mundo. Ahora ya se puede lanzar el robot en Gazebo directamente en el escenario creado previamente. En la Figura 52 se muestra el escenario creado para las pruebas de simulación del robot.

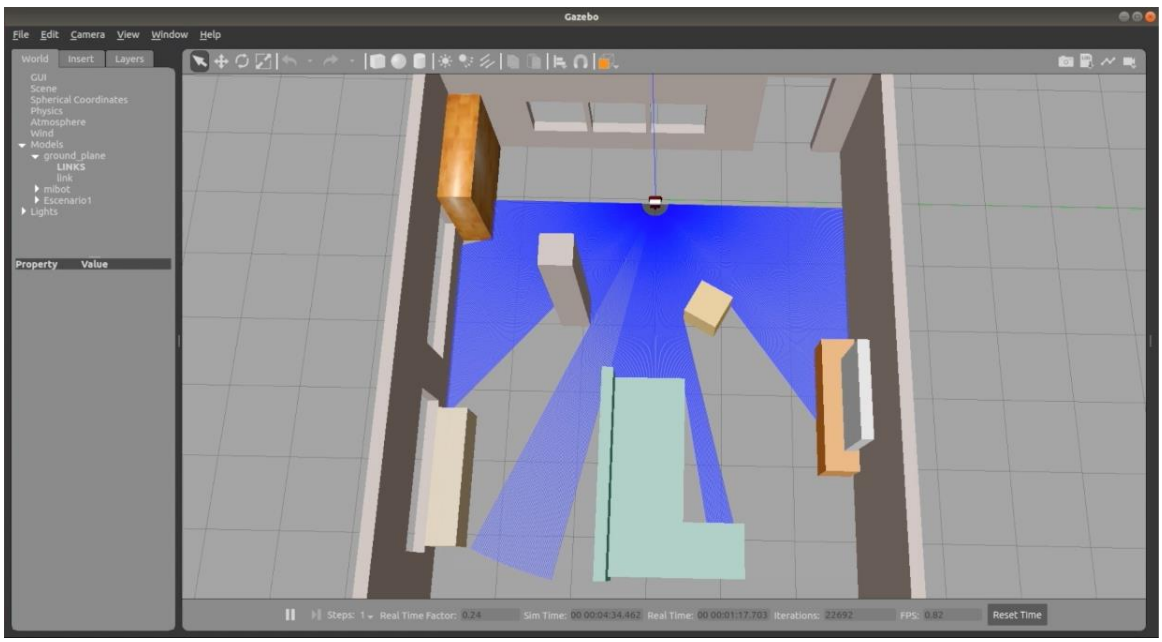


Figura 52. Escenario creado en Gazebo para pruebas de simulación del robot.

Para mapear este escenario hay que iniciar RViz, para ello se escribe lo siguiente en una terminal:

```
$ rosrun rviz rviz
```

Una vez abierto RViz, se establece *odom* como *fixed frame* y se indican los *topics* que se desean visualizar:

- RobotModel
- LaserScan: /scan
- Camera: /raw
- Map: /map

Para iniciar la creación del mapa se escribe en otra terminal:

```
$ rosrun gmapping slam_gmapping scan:=scan _base_frame:=odom
```

Este comando inicia el mapeo de la estancia. Para completar el mapa se debe teleoperar el robot para que recorra toda la estancia y se genere el mapa correctamente. Para teleoperar el robot se escribe el siguiente comando en otra terminal:

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

En la Figura 53 se muestra como se va generando el mapa conforme el robot recorre la estancia. En el Anexo III se muestra el contenido del archivo *launch* para realizar esta tarea de mapeo.

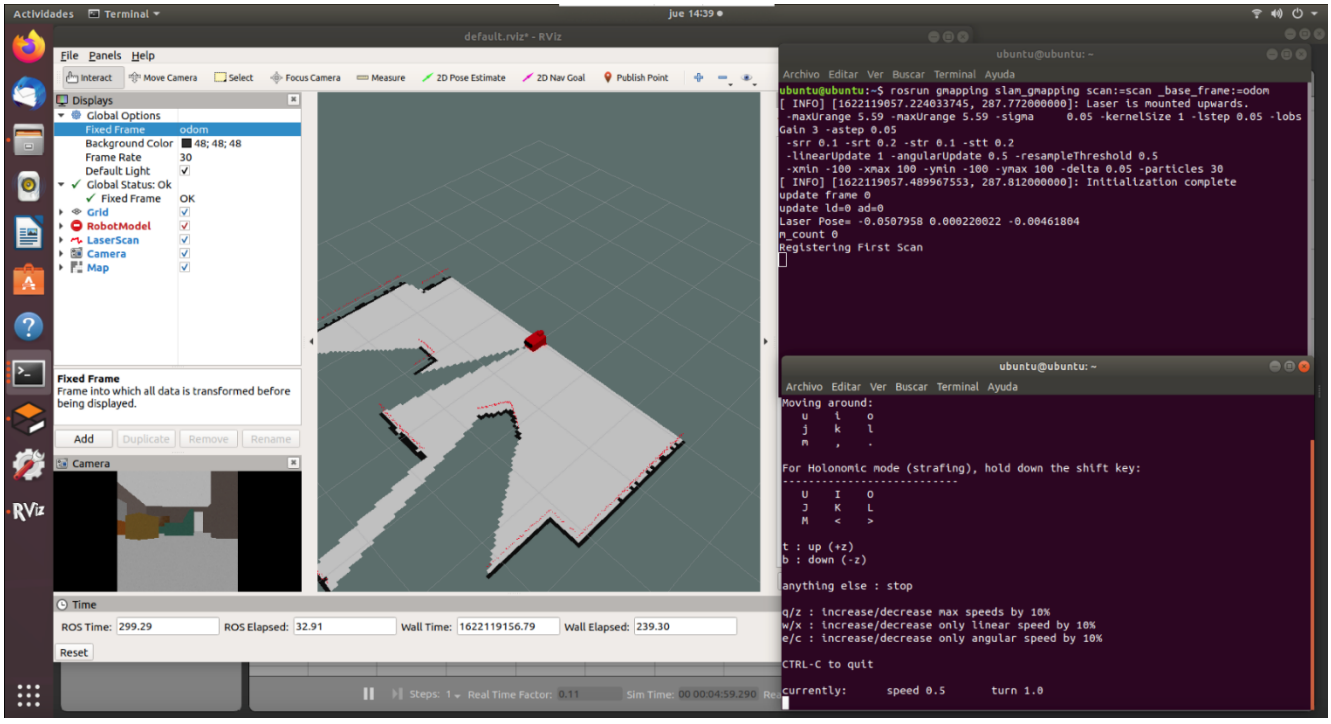


Figura 53. Mapeo de la simulación mediante slam_gmapping y la teleoperación del robot.

Una vez hecho el mapa, se guarda usando el paquete *map_server*. Para guardar el mapa, se escribe lo siguiente en una terminal:

```
$ rosrun map_server map_saver -f map_MKII
```

Este comando genera y guarda los archivos *map_MKII.pgm* y *map_MKII.yaml* en la carpeta personal del usuario. El archivo de tipo *pgm* contiene la imagen del mapa (Figura 54), mientras que el archivo tipo *yaml* contiene la información necesaria para la navegación por ese mapa (en el Anexo IV se muestra el contenido de este archivo).

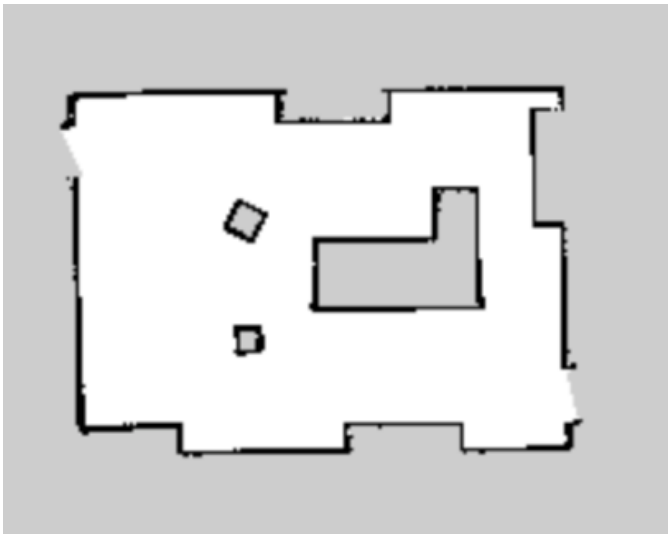


Figura 54. Mapa resultante de la estancia.

Para la navegación autónoma del robot en la simulación se deben crear una serie de archivos de configuración y de tipo *launch* como se describió en el apartado 3.3.2. Los archivos que se deben crear son:

- map_server.launch
- amcl_only.launch
- costmap_common_params.yaml dentro de la carpeta *config*.
- global_costmap_params.yaml dentro de la carpeta *config*.
- local_costmap_params.yaml dentro de la carpeta *config*.
- trayectory_planner.yaml dentro de la carpeta *config*.

En el Anexo V se muestra el contenido del archivo *launch* final para la navegación autónoma del robot en la simulación de Gazebo. La Figura 55 muestra la visualización mediante RViz del robot en el mapa creado.

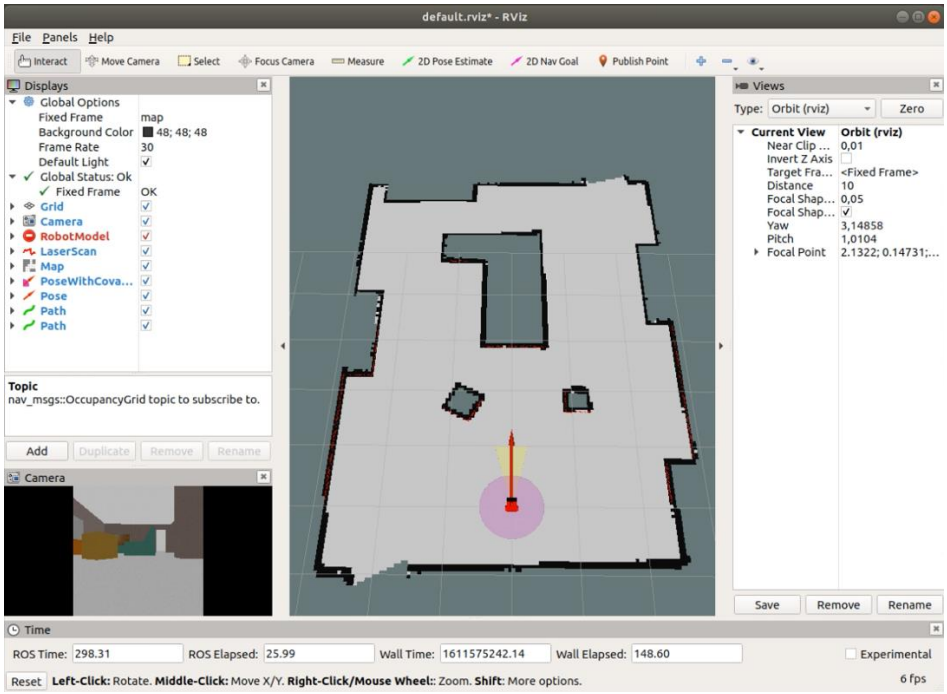


Figura 55. Visualización mediante RViz del robot en el mapa.

Con la herramienta *2D Nav Goal* de RViz se puede establecer el punto al que se quiere que el robot acuda. RViz también permite ver los mapas tipo *costmap* y la trayectoria calculada. En la Figura 56 se puede ver cómo se visualizan estos parámetros por RViz.

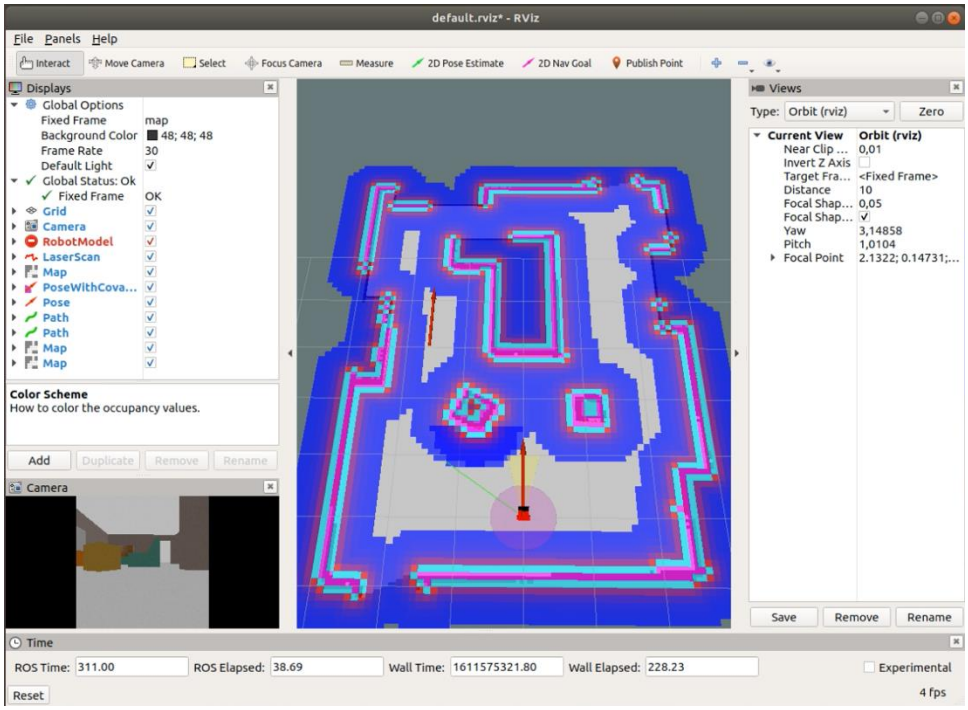


Figura 56. Establecimiento en RViz del punto de destino.

El círculo rosado en el que se encuentra el robot es la representación gráfica de la estimación de la pose de este en el mapa (ver Figura 56). Este círculo representa la probabilidad de situación del robot, siendo el centro de este círculo el lugar con más probabilidad de presencia del robot. Al principio este circulo es bastante grande, pero conforme el robot se mueve, se va haciendo más certera su localización, con lo que el círculo se reduce. Esto se puede observar en la Figura 57, donde la estimación de la pose del robot es tan exacta que es un punto que coincide con el centro del robot.

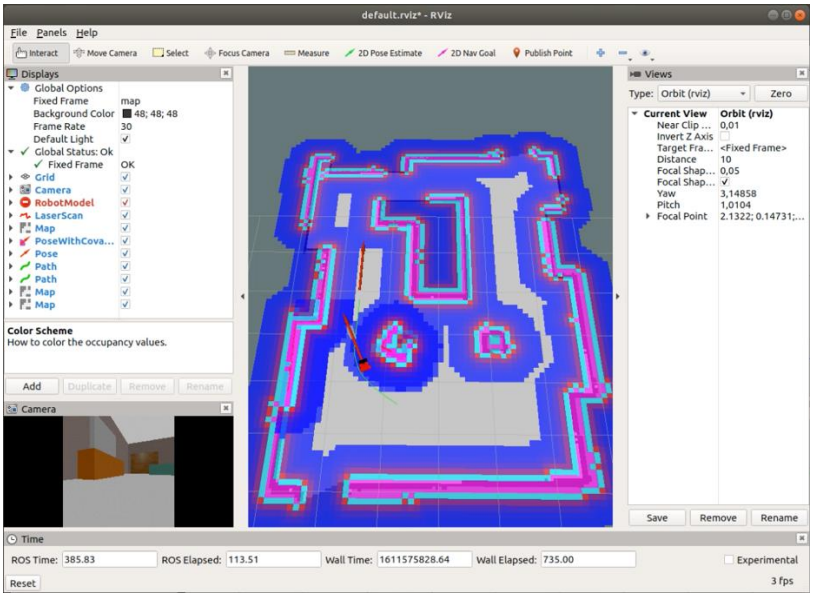


Figura 57. Trazado de trayectoria y estimación de la pose del robot.

En la Figura 58 se puede observar como el robot alcanza el destino que se le había fijado.

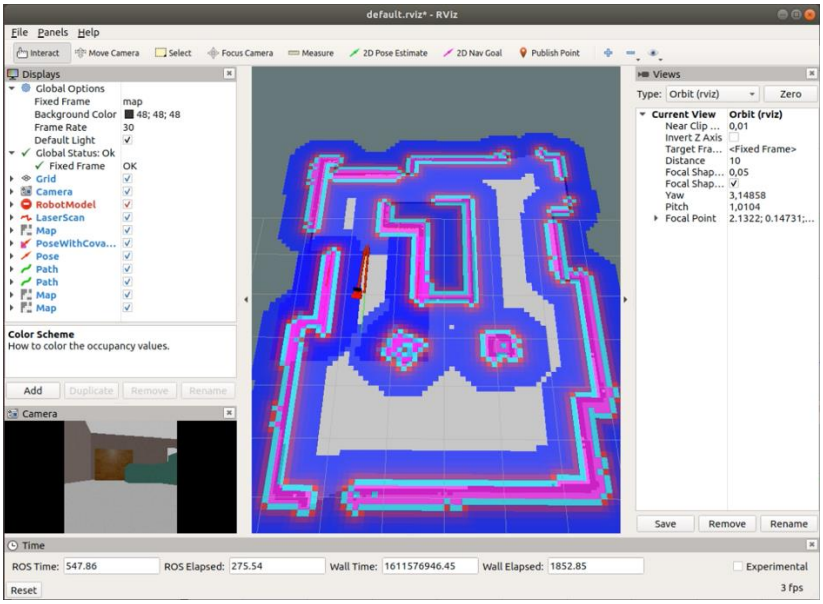


Figura 58. El robot alcanza su destino.

Pruebas con el robot real.

El procedimiento para la navegación autónoma del robot real es muy similar. La mayor diferencia es que no se requiere de Gazebo. El archivo *xacro* se mantiene para el robot real, ya que es el modelo que se ha creado para que aparezca en RViz y se pueda monitorizar. Como el robot ya no se encuentra en el entorno virtual de Gazebo, en su *launch* hay que suscribirlo tanto al nodo del láser RPLidar, como al nodo del controlador del sistema de locomoción rosmd25. En el Anexo VI se muestra el *launch* para que el robot real genere el mapa. Para usar gmapping, en vez de iniciarlo en una terminal, se ha optado por crear un archivo que lo inicie con los parámetros del RPLidar ya establecidos y se incluye en el *launch* general del robot. En el Anexo VII se muestra el interior del archivo que configura e inicia gmapping. En la Figura 59 se muestra como se puede monitorizar al robot real en RViz mientras genera un mapa de la estancia en la que se encuentra.

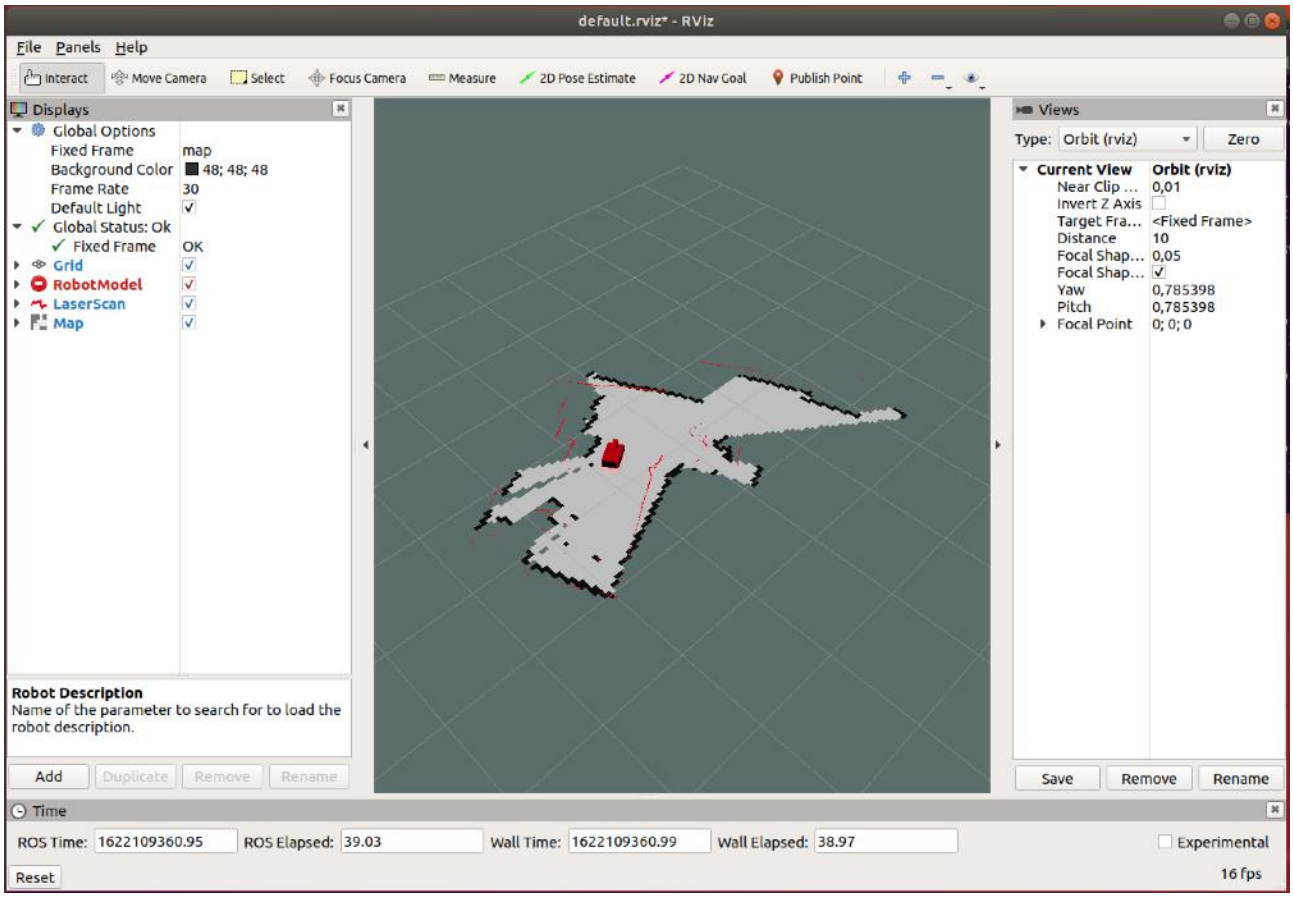


Figura 59. Visualización mediante RViz del mapa que está generando el robot real.

A partir de este punto el procedimiento continúa de la misma forma que en la simulación.

4.1.2 Pruebas de reconocimiento visual.

Para las pruebas de reconocimiento facial, se emplearán imágenes que el sistema no conoce, puesto que no están incluidas en el conjunto de datos con el que se entrena y valida al modelo.

En primer lugar, se le aporta la imagen mostrada en la Figura 60. A esta imagen le corresponde la clasificación *adrian*. En la Figura 61 se muestra la respuesta del modelo frente a esta imagen.

Figura 60. adrian5.png.

```
= RESTART: /Users/adriansanchezgranero/Documents/TFG/Modelos/reconocimientocaras2.
py
/Users/adriansanchezgranero/Documents/TFG/Faces
0
Found 80 files belonging to 2 classes.
Using 48 files for training.
Found 80 files belonging to 2 classes.
Using 32 files for validation.
['adrian', 'noadrian']
0.0 1.0
Epoch 1/2

1/2 [==============>.............] - ETA: 0s - loss: 0.6864 - accuracy: 0.6250

2/2 [==============================] - ETA: 0s - loss: 0.6691 - accuracy: 0.6458

2/2 [==============================] - 1s 353ms/step - loss: 0.6691 - accuracy: 0.
6458 - val_loss: 0.6943 - val_accuracy: 0.5000
Epoch 2/2

1/2 [==============>.............] - ETA: 0s - loss: 0.6166 - accuracy: 0.4688

2/2 [==============================] - ETA: 0s - loss: 0.6016 - accuracy: 0.5833

2/2 [==============================] - 0s 246ms/step - loss: 0.6016 - accuracy: 0.
5833 - val_loss: 0.5427 - val_accuracy: 0.7812
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 100, 100, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 98, 98, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 32)        18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 21, 21, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 3200)              0
_________________________________________________________________
dense (Dense)                (None, 60)                192060
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 122
=================================================================
Total params: 221,686
Trainable params: 221,686
Non-trainable params: 0
_________________________________________________________________
Guardando el modelo
Voy por aquí
This image most likely belongs to adrian with a 57.65 percent confidence.
```

Figura 61. Resultado del modelo de reconocimiento facial para la imagen adrian5.png.

El modelo clasifica la imagen adrian5.png (ver Figura 60) como *adrian* con un 58% de confianza (ver Figura 61), lo que es correcto.

En segundo lugar, se le aporta al modelo la imagen mostrada en la Figura 62. A esta imagen le corresponde la clasificación *noadrian*. En la Figura 63 se muestra la respuesta del modelo frente a esta imagen.

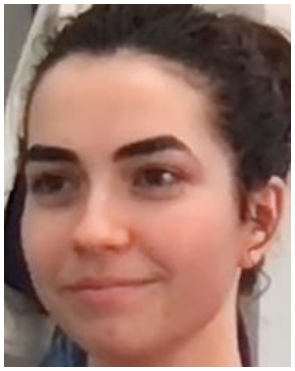


Figura 62. irene1.png.

```
= RESTART: /Users/adriansanchezgranero/Documents/TFG/Modelos/reconocimientocaras2.
py
/Users/adriansanchezgranero/Documents/TFG/Faces
0
Found 80 files belonging to 2 classes.
Using 48 files for training.
Found 80 files belonging to 2 classes.
Using 32 files for validation.
['adrian', 'noadrian']
0.0 1.0
Epoch 1/2

1/2 [==============>..............] - ETA: 0s - loss: 0.6866 - accuracy: 0.4688

2/2 [==============================] - ETA: 0s - loss: 0.6690 - accuracy: 0.5625

2/2 [==============================] - 1s 355ms/step - loss: 0.6690 - accuracy: 0.
5625 - val_loss: 0.6324 - val_accuracy: 0.6250
Epoch 2/2

1/2 [==============>..............] - ETA: 0s - loss: 0.5759 - accuracy: 0.7500

2/2 [==============================] - ETA: 0s - loss: 0.5576 - accuracy: 0.7917

2/2 [==============================] - 0s 247ms/step - loss: 0.5576 - accuracy: 0.
7917 - val_loss: 0.5343 - val_accuracy: 0.6250
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 100, 100, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 98, 98, 64)        1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 32)        18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 21, 21, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 3200)              0
_________________________________________________________________
dense (Dense)                (None, 60)                192060
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 122
=================================================================
Total params: 221,686
Trainable params: 221,686
Non-trainable params: 0
_________________________________________________________________
Guardando el modelo
Voy por aquí
This image most likely belongs to noadrian with a 68.22 percent confidence.
```

Figura 63. Resultado del modelo de reconocimiento facial para la imagen irene1.png.

El modelo clasifica la imagen irene1.png (ver Figura 62) como *noadrian* con un 68% de confianza (ver Figura 63), lo que es correcto.

4.1.3 Pruebas de reconocimiento y síntesis de voz.

Para la prueba de reconocimiento de voz, se le aporta al sistema una pista de audio para que el modelo la convierta a texto.

Para la prueba de síntesis de voz, se le aporta al sistema un texto para que el modelo lo convierta en audio.

Para la prueba de reconocimiento de lenguaje natural, sele aporta al sistema un texto para que el modelo realice un análisis gramatical.

4.2 Resultados.

En lo referente a las **pruebas de navegación**, la simulación en Gazebo responde adecuadamente, generando correctamente un mapa del escenario y pudiendo navegar de forma autónoma por dicho escenario mediante el uso de un mapa conocido. Por otra parte, el robot real presenta un problema con la publicación de la odometría, lo que dificulta el mapeado y por consiguiente genera una navegación autónoma errática.

Las **pruebas de reconocimiento facial** han sido un éxito, pero presentando una confianza entre el 55% y el 70%. Como se puede observar en la Figura 61 y Figura 63, atendiendo a la última época, se pueden observar unos valores de precisión de entrenamiento y de validación con una diferencia de un 20%, lo que indica la posibilidad de un sobreajuste. La presencia de sobreajuste alerta de que el conjunto de datos no es lo suficientemente amplio y por ello da como resultado niveles de confianza reducidos.

4.3 Discusión.

Una vez expuestos los resultados de las diferentes pruebas que se han llevado acabo, se puede afirmar que se ha logrado el objetivo de crear un robot real con capacidades de navegación autónoma y de interacción con humanos.

En lo referente a la navegación, la simulación ha respondido correctamente. Sin embargo, las pruebas con el robot real presentaban un reto adicional, dado que la movilidad de éste es reducida: sólo puede avanzar hacia delante en línea recta o describiendo arcos. Esto hace que sea complicado el acceso a lugares con giros pronunciados. Para poder solucionar esto, se ha propuesto modificar el driver del controlador del sistema de locomoción, para permitir al robot real que gire sobre sí mismo. Este añadido aporta una mayor maniobrabilidad. En una simulación, los parámetros del entorno son proporcionados por Gazebo, lo que reduce

la posibilidad de incidencias en gran medida. Al usar un robot real, pueden darse lugar a incidencias como que los sensores registren una gran cantidad de "ruido" (datos no válidos, que no se corresponden con las medidas registradas y que dificultan o distorsionan el conjunto de datos), lo que produce valores falsos que podrían dar lugar a un desajuste de la odometría. Esto repercutiría en que el robot real está trabajando con valores reales y falsos al mismo tiempo. Para comprobar y solucionar este asunto, se debería calibrar la odometría que publica el robot.

La red neuronal para el reconocimiento de personas es capaz de distinguir entre el usuario y un desconocido. El echo de diseñar el modelo de la red neuronal de forma binaria (*adrian*, *noadrian*), limita al sistema a poder reconocer sólo a una persona. Para que el sistema sea capaz de reconocer a más de un usuario, se debería crear un modelo por persona que se desea reconocer, lo que aumenta en gran medida el conjunto de datos y la carga de trabajo del sistema. El modelo diseñado ha respondido correctamente, pero como se ha comentado en sus resultados, podría existir cierto sobreajuste. La presencia de sobreajuste implica que el conjunto de datos no es lo suficientemente grande para un óptimo entrenamiento del sistema, resultando en clasificaciones erróneas para situaciones nuevas (distintas de con las que se ha entrenado).

La implementación del software de reconocimiento y síntesis de voz, aportado por el compañero Roberto Oterino Bono, supone la dificultad de trabajar con diversos lenguajes de programaión. El software anteriormente mencionado está desarrollado en Python con TensorFlow, que se debe convertir a Python para la versión de TensorFlow Lite para poder implementarlo en la SBC Raspberry Pi, que a su vez se debe ajustar para poder emplearlo con ROS, que está desarrollado en C++. Estos cambios de lenguaje dificultan la comunicación entre los diversos softwares desarrollados, repercutiendo en el correcto funcionamiento del sistema.

5 Conclusiones y trabajo futuro.

Para finalizar el trabajo, se realiza un resumen de los objetivos que se han logrado llevar a cabo, junto con la repercusión de las metodologías descritas en apartados anteriores en los resultados obtenidos. Por último, se describen las limitaciones que presenta el sistema desarrollado, junto con posibles proyectos que podrían solucionarlas o añadir nuevas funciones.

5.1 Conclusiones.

Se ha construido un robot real, con capacidades de navegación autónoma limitadas pero funcionales. Únicamente con el uso de un dispositivo de barrido láser, un dispositivo diferencial de locomoción y una SBC Raspberry Pi, se ha conseguido mapear una estancia y que el robot sea capaz de navegar por ella de forma autónoma.

Aunque se han dado ciertos problemas con las pruebas para el robot real, relacionados con la calibración de la odometría, las pruebas demuestran que todos los componentes funcionan apropiadamente. Para resolver el problema de la calibración, se debe mejorar el funcionamiento del propio controlador del sistema de locomoción.

Se ha desarrollado con éxito un módulo de red neuronal destinado al reconocimiento de personas, distinguiendo entre el usuario y una persona desconocida para el sistema. El uso de TensorFlow junto con Keras ha reducido en gran medida la complejidad de la creación del software. Aunque la confianza del modelo no supera el 80% en la mayoría de los casos y la posibilidad de presencia de sobreajuste puedan afectar al correcto funcionamiento del modelo, se podría solucionar aumentando el conjunto de datos de que se dispone. Simplemente habría que tomar más imágenes del usuario a reconocer y de otras personas presentes en el entorno en el que se empleará el robot, para que el modelo mejore. Con esta técnica, se pueden crear modelos para diferentes usuarios, con lo que es posible que el robot pueda aprender fácilmente a distinguir varios usuarios, de una forma más escalable.

Dado que se partía de conocimientos de ROS, el lenguaje base que se poseía era C++, y puesto que los modelos de red neuronal de TensorFlow utilizan Python, esto requirió del estudio del nuevo lenguaje de programación y de métodos para la conversión entre lenguajes.

Se ha logrado implementar el software desarrollado con anterioridad por otro usuario, en el sistema creado para el presente trabajo.

En conclusión, se ha logrado cumplir el objetivo principal de este trabajo, la construcción de un robot real con capacidades de navegación autónoma y de interacción con personas.

5.1.1 Trabajo futuro.

En lo referente a trabajos futuros, se plantea la posibilidad de integrar el modelo de red neuronal para el reconocimiento de personas en el robot real. Para ello se debería hacer una conversión de dicho modelo a una versión de TensorFlow Lite, como se explicó en el apartado 3.3.4.

Relacionado con el reconocimiento de personas, se propone el implementar más modelos, para que el sistema sea capaz de reconocer a más usuarios. Del mismo modo se recomienda ampliar el conjunto de datos a emplear por el modelo en las tareas de entrenamiento y validación para evitar el sobreajuste.

Por otra parte, relacionado con el tema de la navegación autónoma del robot real, se propone realizar una calibración de la odometría para que se pueda crear el mapa sin desajustes que dificulten las tareas de navegación.

Por último, como trabajo futuro se plantea el crear un modelo para el seguimiento de personas (body tracking) mediante el uso de la cámara Orbbec Astra Pro, para proyectar el punto medio de la nube de puntos que se correspondería al rostro del usuario al nivel del suelo, convirtiendo ese punto en una consigna de destino para la navegación del robot. Esto dotaría al robot con la habilidad de seguir al usuario de forma indefinida, hasta que el usuario se detenga o se le ordene al robot que cese el seguimiento.

6 Bibliografía

[1] «Revista de Robots,» 3 Abril 2021. [En línea]. Available: https://revistaderobots.com/robots-y-robotica/que-es-la-robotica/. [Último acceso: 10 Mayo 2021].

[2] Y. Fernández, «Xataka,» 15 Enero 2021. [En línea]. Available: https://www.xataka.com/basics/que-alexa-que-puedes-hacer-que-dispositivos-compatibles. [Último acceso: 10 Mayo 2021].

[3] S. R. Europe, «AliveRobots,» Robotrónica, 3 Junio 2016. [En línea]. Available: https://aliverobots.com/robot-pepper/. [Último acceso: 10 Mayo 2021].

[4] J. E. Riva, «ROS.org,» Open Robotics, 13 Marzo 2021. [En línea]. Available: http://wiki.ros.org/es/ROS/Introduccion. [Último acceso: 10 Mayo 2021].

[5] R. O. Bono, «Diseño de un robot móvil de servicio con capacidades de interacción natural con humanos,» UPCT, Cartagena, 2021.

[6] C. Frabetti, «El País,» PRISA, 25 Agosto 2017. [En línea]. Available: https://elpais.com/elpais/2017/08/24/ciencia/1503574908_187790.html. [Último acceso: 11 Mayo 2021].

[7] «IAT,» 13 Octubre 2020. [En línea]. Available: https://iat.es/tecnologias/robotica/social/. [Último acceso: 11 Mayo 2021].

[8] «BBVA,» Banco Bilbao Vizcaya Argenyaria S.A., 8 Noviembre 2019. [En línea]. Available: https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/#:~:text=El%20'machine%20learning'%20%E2%80%93aprendizaje,los%20datos%20para%20hacer%20predicciones.. [Último acceso: 11 Mayo 2021].

[9] «INVOX Medical,» [En línea]. Available: https://invoxmedical.com/noticias/la-inteligencia-artificial-en-el-reconocimiento-de-voz/. [Último acceso: 11 Mayo 2021].

[10] C. M. G. Vélez, «Modelos ocultos de Markov para el desarrollo de un sistema de ayuda al habla para personas que sufren de disartia,» Universidad Nacional Pedro Ruiz Gallo, Lambayeque, Perú, 2018.

[11] C. Villoria, «Observatorio Tecnológico,» Gobierno de España, Ministerio de Educación, Cultura y Deporte, 31 Enero 2009. [En línea]. Available: http://recursostic.educacion.es/observatorio/web/eu/software/software-general/689-reconocimiento-y-sintesis-de-voz. [Último acceso: 12 Mayo 2021].

[12] J. Abad, «DAIL,» DAIL Software, 8 Enero 2019. [En línea]. Available: https://www.dail.es/aplicaciones-del-procesamiento-del-lenguaje-natural/. [Último acceso: 13 Mayo 2021].

[13] «IBM,» IBM Corporation, 2020. [En línea]. Available: https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model. [Último acceso: 25 Mayo 2021].

[14] TensorFlow, «TensorFlow,» Google, [En línea]. Available: https://www.tensorflow.org/?hl=es-419. [Último acceso: 26 Mayo 2021].

[15] «IONOS,» 1&1 IONOS España S.L.U., 8 Octubre 2020. [En línea]. Available: https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-busqueda/que-es-keras/. [Último acceso: 25 Mayo 2021].

[16] «raspberrypi.org,» Raspberry Pi, [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. [Último acceso: 14 Mayo 2021].

[17] «SuperRobotica,» 14 Mayo 2021. [En línea]. Available: http://www.superrobotica.com/S310114.htm. [Último acceso: 15 Mayo 2021].

[18] R. A. I. a. Datasheet, «rbotshop.com,» 23 Marzo 2018. [En línea]. Available: https://www.robotshop.com/media/files/pdf2/ld108_slamtec_rplidar_datasheet_a1m8_v1.1_en_2_.pdf. [Último acceso: 15 Mayo 2021].

[19] «ROS Components,» [En línea]. Available: https://www.roscomponents.com/es/camaras/248-orbbec.html. [Último acceso: 15 Mayo 2021].

[20] «RobotShop,» [En línea]. Available: https://www.robotshop.com/uk/drive-system-12-volt.html. [Último acceso: 15 Mayo 2021].

[21] «Amazon,» [En línea]. Available: https://www.amazon.es/ICQUANZX-conmutaci%C3%B3n-helic%C3%B3ptero-Avi%C3%B3n-Quadcopter/dp/B081CV3D2B/ref=sr_1_2?dchild=1&keywords=ubec&qid=1621097536&s=toys&sr=1-2. [Último acceso: 15 Mayo 2021].

[22] J. E. Riva, «Wiki ROS,» 10 Febrero 2021. [En línea]. Available: http://wiki.ros.org/es/ROS/Conceptos. [Último acceso: 17 Mayo 2021].

[23] F. Larribe, «wiki.ros.org,» Open Robotics, 14 Septiembre 2020. [En línea]. Available: http://wiki.ros.org/navigation. [Último acceso: 23 Mayo 2021].

[24] M. Purvis, «wiki.ros.org,» Open Robotics, 22 Enero 2015. [En línea]. Available: http://wiki.ros.org/teleop_twist_keyboard. [Último acceso: 23 Mayo 2021].

[25] «wiki.ros.org,» Open Robotics, 27 Agosto 2020. [En línea]. Available: http://wiki.ros.org/amcl. [Último acceso: 23 Mayo 2021].

[26] SmartPanel, «SmartPanel,» Andalucía Open Future, 10 Abril 2018. [En línea]. Available: https://www.smartpanel.com/que-es-deep-learning/. [Último acceso: 26 Mayo 2021].

[27] R. O. Bono, «Diseño de un robot móvil de servicio con capacidades de interacción con humanos.,» UPCT, Cartagena, 2021.

[28] tensorflow.org, «TensorFlow,» Google, 20 Mayo 2021. [En línea]. Available: https://www.tensorflow.org/tutorials/images/classification. [Último acceso: 28 Mayo 2021].

[29] TensorFlow, «TensorFlow,» Google, 5 Febrero 2021. [En línea]. Available: https://www.tensorflow.org/lite/convert/index?hl=es_419. [Último acceso: 30 Mayo 2021].

[30] TensorFlow, «TensorFlow,» Google, 24 Marzo 2021. [En línea]. Available: https://www.tensorflow.org/lite/guide/inference?hl=es_419. [Último acceso: 30 Mayo 2021].

[31] «SAS,» SAS Institute Inc., [En línea]. Available: https://www.sas.com/es_es/insights/analytics/what-is-natural-language-processing-nlp.html. [Último acceso: 13 Mayo 2021].

[32] D. O. Delgado, «OpenWebinars,» 21 Septiembre 2017. [En línea]. Available: https://openwebinars.net/blog/que-es-ros/. [Último acceso: 17 Mayo 2021].

7 Anexos.

7.1 Anexo I

mibot.xacro

```
<?xml version="1.0"?>
<!-- Nombre del robot y definicion del espacio de nombres xacro -->
<robot name="mibot" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- Importamos archivo .gazebo con información visual y posibles plugins -->
  <xacro:include filename = "$(find unibot_description)/urdf/mibot.gazebo" />
  <!-- Propiedades o parametros que utilizaremos -->
  <xacro:property name="ancho_base" value="0.20"/>
  <xacro:property name="largo_base" value="0.30"/>
  <xacro:property name="grosor_base" value="0.187"/>
  <xacro:property name="radio_rueda" value="0.048"/>
  <xacro:property name="radio_ruedacastor" value="0.024"/>
  <xacro:property name="grosor_rueda" value="0.025"/>
  <xacro:property name="distancia_a_suelo_rueda" value="0.0935"/>
  <xacro:property name="distancia_a_suelo_ruedacastor" value="0.1175"/>
  <xacro:property name="masa_base" value="2.787"/>
  <xacro:property name="masa_rueda" value="0.152"/>
  <xacro:property name="masa_ruedacastor" value="0.076"/>

  <xacro:property name="ixx_base" value = "0.02902405025"/>
  <xacro:property name="iyy_base" value = "0.0301925"/>
  <xacro:property name="izz_base" value = "0.01741155025"/>

  <xacro:property name="ixx_rueda" value = "0.00009546866"/>
  <xacro:property name="iyy_rueda" value = "0.000175104"/>
  <xacro:property name="izz_rueda" value = "0.00009546866"/>

  <xacro:property name="ixx_ruedacastor" value = "0.0000175104"/>
  <xacro:property name="iyy_ruedacastor" value = "0.0000175104"/>
  <xacro:property name="izz_ruedacastor" value = "0.0000175104"/>

  <xacro:property name="alturalaser" value="0.053"/>
  <xacro:property name="anchuralaser" value="0.096"/>
```

```
<xacro:property name="proflaser" value="0.069"/>
<xacro:property name="masalaser" value="0.193"/>
<xacro:property name="ixxlaser" value = "0.00012175083"/>
<xacro:property name="iyylaser" value = "0.00022479675"/>
<xacro:property name="izzlaser" value = "0.00019340208"/>
<xacro:property name="desplzlaser" value = "0.12"/>

<xacro:property name="altura_rgb" value="0.05"/>
<xacro:property name="anchura_rgb" value="0.04"/>
<xacro:property name="prof_rgb" value="0.164"/>
<xacro:property name="masa_rgb" value="0.310"/>
<xacro:property name="ixx_rgb" value = "0.00075939666"/>
<xacro:property name="iyy_rgb" value = "0.00073614666"/>
<xacro:property name="izz_rgb" value = "0.00010591666"/>
<xacro:property name="desplz_rgb" value = "0.0685"/>

<!-- Evitamos warning en Gazebo sobre baselink como root y la inercia -->
<link name = "none">
</link>
<!-- Definimos el link baselink que es el objeto que representa el chasis -->
<link name="baselink">
  <!-- Definimos los limites de colision del chasis -->
  <collision>
          <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${largo_base} ${ancho_base} ${grosor_base}" />
    </geometry>
  </collision>
  <!-- Definimos parametros relacionados con el aspecto visual -->
  <visual>
          <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${largo_base} ${ancho_base} ${grosor_base}" />
    </geometry>
  </visual>
       <!-- Definimos algunos parametros relacionados con la dinamica que tendra el chasis en la simulacion -->
  <inertial>
          <origin xyz="0 0 0" rpy="0 0 0"/>
          <mass value="${masa_base}"/>
          <inertia
```

```
                ixx="${ixx_base}"
        ixy="0"
                        ixz="0"
                        iyy="${iyy_base}"
                        iyz="0"
                        izz="${izz_base}"/>
  </inertial>
</link>

<!-- Definimos la rueda izquierda -->
<link name="ri">
        <!-- Definimos los limites de colision de la rueda izquierda -->
  <collision>
            <origin xyz="0 0 0" rpy="-1.5708 0 0"/>
    <geometry>
      <cylinder radius="${radio_rueda}" length="${grosor_rueda}" />
    </geometry>
  </collision>
  <visual>
            <origin xyz="0 0 0" rpy="-1.5708 0 0"/>
    <geometry>
      <cylinder radius="${radio_rueda}" length="${grosor_rueda}" />
    </geometry>
  </visual>
        <inertial>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <mass value="${masa_rueda}"/>
            <inertia
                ixx="${ixx_rueda}"
        ixy="0"
                        ixz="0"
                        iyy="${iyy_rueda}"
                        iyz="0"
                        izz="${izz_rueda}"/>
        </inertial>
</link>

<!-- Definimos la rueda derecha -->
<link name="rd">
        <!-- Definimos los limites de colision de la rueda derecha -->
```

```
    <collision>
            <origin xyz="0 0 0" rpy="-1.5708 0 0"/>
      <geometry>
        <cylinder radius="${radio_rueda}" length="${grosor_rueda}" />
      </geometry>
    </collision>
    <visual>
            <origin xyz="0 0 0" rpy="-1.5708 0 0"/>
      <geometry>
        <cylinder radius="${radio_rueda}" length="${grosor_rueda}" />
      </geometry>
    </visual>
          <inertial>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <mass value="${masa_rueda}"/>
            <inertia
              ixx="${ixx_rueda}"
              ixy="0"
              ixz="0"
              iyy="${iyy_rueda}"
              iyz="0"
              izz="${izz_rueda}"/>
          </inertial>
</link>

<!-- Definimos la rueda castor -->
<link name="castor">
          <!-- Definimos los limites de colision de la rueda castor -->
    <collision>
            <origin xyz="0 0 0" rpy="-1.5708 0 0"/>
      <geometry>
        <sphere radius="${radio_ruedacastor}" />
      </geometry>
    </collision>
    <visual>
            <origin xyz="0 0 0" rpy="-1.5708 0 0"/>
      <geometry>
        <sphere radius="${radio_ruedacastor}" />
      </geometry>
    </visual>
```

```
        <inertial>
          <origin xyz="0 0 0" rpy="0 0 0"/>
          <mass value="${masa_ruedacastor}"/>
          <inertia
            ixx="${ixx_ruedacastor}"
            ixy="0"
            ixz="0"
            iyy="${iyy_ruedacastor}"
            iyz="0"
            izz="${izz_ruedacastor}"/>
        </inertial>
</link>

<!-- Link para el dispositivo de barrido laser -->
<link name="laser">
  <!-- Definimos los limites de colision del laser -->
  <collision>
          <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${anchuralaser} ${proflaser} ${alturalaser}" />
    </geometry>
  </collision>
  <!-- Definimos parametros relacionados con el aspecto visual -->
  <visual>
          <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${anchuralaser} ${proflaser} ${alturalaser}" />
    </geometry>
  </visual>
       <!-- Definimos algunos parametros relacionados con la dinamica que tendra el laser en la simulacion -->
  <inertial>
          <origin xyz="0 0 0" rpy="0 0 0"/>
          <mass value="${masalaser}"/>
          <inertia
            ixx="${ixxlaser}"
            ixy="0"
            ixz="0"
            iyy="${iyylaser}"
            iyz="0"
            izz="${izzlaser}"/>
```

```
  </inertial>
</link>

<!-- Link para el dispositivo camara RGB -->
<link name="rgb">
  <!-- Definimos los limites de colision de la camara RGB -->
  <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${anchura_rgb} ${prof_rgb} ${altura_rgb}" />
    </geometry>
  </collision>
  <!-- Definimos parametros relacionados con el aspecto visual -->
  <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="${anchura_rgb} ${prof_rgb} ${altura_rgb}" />
    </geometry>
    <material name="blue"/>
  </visual>
      <!-- Definimos algunos parametros relacionados con la dinamica que tendra la camara RGB en la simulacion -->
  <inertial>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <mass value="${masa_rgb}"/>
        <inertia
          ixx="${ixx_rgb}"
          ixy="0"
          ixz="0"
          iyy="${iyy_rgb}"
          iyz="0"
          izz="${izz_rgb}"/>
  </inertial>
</link>

<!-- Definicion de los Joints -->

<joint name="none_base" type = "fixed">
      <parent link = "none"/>
      <child link = "baselink"/>
</joint>
```

```
  <joint name="joint_base_ri" type = "continuous">
          <axis xyz = "0 1 0" rpy = "0 0 0"/>
          <parent link = "baselink"/>
          <child link = "ri"/>
          <origin xyz = "-0.115 -0.1125 -${distancia_a_suelo_rueda}" rpy = "0 0 0"/>
  </joint>

  <joint name="joint_base_rd" type = "continuous">
          <axis xyz = "0 1 0" rpy = "0 0 0"/>
          <parent link = "baselink"/>
          <child link = "rd"/>
          <origin xyz = "-0.115 0.1125 -${distancia_a_suelo_rueda}" rpy = "0 0 0"/>
  </joint>

  <joint name="joint_base_castor" type = "continuous">
          <axis xyz = "0 1 0" rpy = "0 0 0"/>
          <parent link = "baselink"/>
          <child link = "castor"/>
          <origin xyz = "0.124 0 -${distancia_a_suelo_ruedacastor}" rpy = "0 0 0"/>
  </joint>

  <joint name="joint_base_laser" type = "fixed">
     <axis xyz = "0 1 0" rpy = "0 0 0"/>
     <parent link="baselink"/>
     <child link="laser" />
     <origin xyz="-0.115 0 ${desplzlaser}" rpy = "0 0 0"/>
  </joint>

  <joint name="joint_base_rgb" type = "fixed">
     <axis xyz = "0 1 0" rpy = "0 0 0"/>
     <parent link="baselink"/>
     <child link="rgb" />
     <origin xyz="0.17 0 ${desplz_rgb}" rpy = "0 0 0"/>
  </joint>

</robot>
```

7.2 Anexo II

mibot.gazebo

```
<?xml version="1.0"?>
<robot>
  <gazebo reference = "baselink">
        <visual>
          <material>
                <ambient>1.0 0.0 0.0 1.0</ambient>
                <diffuse>0.0 0.0 0.0 1.0</diffuse>
                <specular>0.0 0.0 0.0 1.0</specular>
                <emissive>0.0 0.0 0.0 1.0</emissive>
          </material>
        </visual>
  </gazebo>
  <gazebo reference = "ri">
        <visual>
          <material>
                <ambient>0.0 0.0 0.0 1.0</ambient>
                <diffuse>0.0 0.0 0.0 1.0</diffuse>
                <specular>0.0 0.0 0.0 1.0</specular>
                <emissive>0.0 0.0 0.0 1.0</emissive>
          </material>
        </visual>
  </gazebo>
  <gazebo reference = "rd">
        <visual>
          <material>
                <ambient>0.0 0.0 0.0 1.0</ambient>
                <diffuse>0.0 0.0 0.0 1.0</diffuse>
                <specular>0.0 0.0 0.0 1.0</specular>
                <emissive>0.0 0.0 0.0 1.0</emissive>
          </material>
        </visual>
  </gazebo>
  <gazebo reference = "castor">
        <visual>
          <material>
                <ambient>0.0 0.0 0.0 1.0</ambient>
```

```
                    <diffuse>0.0 0.0 0.0 1.0</diffuse>
                    <specular>0.0 0.0 0.0 1.0</specular>
                    <emissive>0.0 0.0 0.0 1.0</emissive>
                </material>
            </visual>
</gazebo>
<gazebo reference="laser">
  <visual>
            <material>
                    <ambient>0.3 0.1 0.8 1.0</ambient>
                    <diffuse>0.0 0.0 0.0 1.0</diffuse>
                    <specular>0.0 0.0 0.0 1.0</specular>
                    <emissive>0.0 0.0 0.0 1.0</emissive>
            </material>
          </visual>
  <sensor type="ray" name="head_hokuyo_sensor">
             <pose>0 0 0 0 0 0</pose>
      <visualize>true</visualize>
      <update_rate>100</update_rate>
      <ray>
        <scan>
          <horizontal>
            <samples>681</samples>
            <resolution>1</resolution>
            <min_angle>-1.570796</min_angle>
            <max_angle>1.570796</max_angle>
          </horizontal>
        </scan>
        <range>
          <min>0.20</min>
          <max>5.6</max>
          <resolution>0.01</resolution>
        </range>
        <noise>
          <type>gaussian</type>
          <mean>0.0</mean>
          <stddev>0.01</stddev>
        </noise>
      </ray>
      <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
```

```
            <topicName>/scan</topicName>
            <frameName>laser</frameName>
          </plugin>
    </sensor>
  </gazebo>
<gazebo reference="rgb">
  <sensor type="camera" name="cameraRGB">
   <update_rate>30.0</update_rate>
   <camera name="head">
    <horizontal_fov>1.3962634</horizontal_fov>
    <image>
     <width>800</width>
     <height>800</height>
     <format>R8G8B8</format>
    </image>
    <clip>
     <near>0.02</near>
     <far>300</far>
    </clip>
    <noise>
     <type>gaussian</type>
     <!-- Noise is sampled independently per pixel on each frame.
          That pixel's noise value is added to each of its color
          channels, which at that point lie in the range [0,1]. -->
     <mean>0.0</mean>
     <stddev>0.007</stddev>
    </noise>
   </camera>
   <plugin name="gazebo_ros_camera_controller" filename="libgazebo_ros_camera.so">
    <alwaysOn>true</alwaysOn>
    <updateRate>0.0</updateRate>
    <cameraName>mibot/cameraRGB</cameraName>
    <imageTopicName>image_raw</imageTopicName>
    <cameraInfoTopicName>camera_info</cameraInfoTopicName>
    <frameName>rgb</frameName>
    <hackBaseline>0.07</hackBaseline>
    <distortionK1>0.0</distortionK1>
    <distortionK2>0.0</distortionK2>
    <distortionK3>0.0</distortionK3>
    <distortionT1>0.0</distortionT1>
```

```
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
 </gazebo>
  <gazebo>
            <plugin name = "differential_drive_controller" filename = "libgazebo_ros_diff_drive.so">
               <alwaysOn>true</alwaysOn>
               <updateRate>100</updateRate>
               <leftJoint>joint_base_ri</leftJoint>
               <rightJoint>joint_base_rd</rightJoint>
               <wheelSeparation>0.16</wheelSeparation>
               <wheelDiameter>0.08</wheelDiameter>
               <torque>1</torque>
               <legacyMode>false</legacyMode>
               <commandTopic>/cmd_vel</commandTopic>
        <odometryTopic>/odom</odometryTopic>
               <odometryFrame>odom</odometryFrame>
               <robotBaseFrame>baselink</robotBaseFrame>
            </plugin>
  </gazebo>
</robot>
```

7.3 Anexo III

prueba.launch

```
<launch>
  <arg name="paused" default="false" />
  <arg name="use_sim_time" default="true" />
  <arg name="gui" default="true" />
  <arg name="headless" default="false" />
  <arg name="debug" default="false" />

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(find robot_gazebo)/worlds/my_world.world" />
        <arg name="debug" value="$(arg debug)" />
        <arg name="gui" value="$(arg gui)" />
        <arg name="paused" value="$(arg paused)" />
        <arg name="use_sim_time" value="$(arg use_sim_time)" />
        <arg name="headless" value="$(arg headless)" />
  </include>

  <param name="robot_description"
        command="$(find xacro)/xacro '$(find robot_description)/urdf/mibot.xacro'" />

  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
        respawn="false" output="screen" args="-urdf -model mibot -param robot_description" />

  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" ></node>
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />

</launch>
```

7.4 Anexo IV

map_MKII.yaml

```
image: map_MKII.pgm
resolution: 0.050000
origin: [-100.000000, -100.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

7.5 Anexo V

mibot.launch

```
<launch>
  <arg name="paused" default="false" />
  <arg name="use_sim_time" default="true" />
  <arg name="gui" default="true" />
  <arg name="headless" default="false" />
  <arg name="debug" default="false" />

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(find robot_gazebo)/worlds/my_world.world" />
        <arg name="debug" value="$(arg debug)" />
        <arg name="gui" value="$(arg gui)" />
        <arg name="paused" value="$(arg paused)" />
        <arg name="use_sim_time" value="$(arg use_sim_time)" />
        <arg name="headless" value="$(arg headless)" />
  </include>

  <param name="robot_description"
        command="$(find xacro)/xacro '$(find robot_description)/urdf/mibot.xacro'" />

  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
        respawn="false" output="screen" args="-urdf -model mibot -param robot_description" />

  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" ></node>
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />

  <arg name="use_rosbot" default="true"/>

  <node if="$(arg use_rosbot)" pkg="tf" type="static_transform_publisher"
    name="head_hokuyo_sensor" args="0.05 0 0.045 0 0 0 baselink laser 100" />

  <node pkg="move_base" type="move_base" name="move_base" output="screen">
        <param name="controller_frequency" value="20.0"/>
        <rosparam file="$(find robot_gazebo)/config/costmap_common_params.yaml" command="load"
                ns="global_costmap" />
        <rosparam file="$(find robot_gazebo)/config/costmap_common_params.yaml" command="load"
                ns="local_costmap" />
```

```
        <rosparam file="$(find robot_gazebo)/config/local_costmap_params.yaml" command="load" />
        <rosparam file="$(find robot_gazebo)/config/global_costmap_params.yaml" command="load" />
        <rosparam file="$(find robot_gazebo)/config/trayectory_planner.yaml" command="load" />
  </node>

  <include file="$(find robot_gazebo)/launch/map_server.launch"/>
  <include file="$(find robot_gazebo)/launch/amcl_only.launch"/>

</launch>
```

7.6 Anexo VI

mki_unibot.launch

```
<launch>
  <arg name="paused" default="false" />
  <arg name="use_sim_time" default="true" />
  <arg name="gui" default="true" />
  <arg name="headless" default="false" />
  <arg name="debug" default="false" />

  <param name="unibot_description"
         command="$(find xacro)/xacro '$(find unibot_description)/urdf/unibot.xacro'" />
  <node pkg="tf" type="static_transform_publisher" name="base_to_ri" args="-0.115 -0.1125 -0.0935 0 0.0 0.0 baselink ri 100"/>
  <node pkg="tf" type="static_transform_publisher" name="base_to_rd" args="-0.115 0.1125 -0.0935 0 0.0 0.0 baselink rd 100"/>
  <node pkg="tf" type="static_transform_publisher" name="base_to_castor" args="0.124 0.0 -0.1175 0 0.0 0.0 baselink castor 100"/>
  <node pkg="tf" type="static_transform_publisher" name="base_to_rgb" args="0.17 0.0 0.0685 0 0.0 0.0 baselink rgb 100"/>
  <node pkg="tf" type="static_transform_publisher" name="base_to_laser" args="-0.115 0.0 0.12 0 0.0 3.14 baselink laser 100"/>
 <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" >
    <remap from="robot_description" to="unibot_description" />
  </node>
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" >
    <remap from="robot_description" to="unibot_description" />
  </node>

  <node pkg="rosmd25" type ="rosmd25_node" name="rosmd25_node" />

  <node name="rplidarNode"        pkg="rplidar_ros"  type="rplidarNode" output="screen">
  <param name="serial_port"       type="string" value="/dev/ttyUSB0"/>
  <param name="serial_baudrate"    type="int"    value="115200"/><!--A1/A2 -->
  <!--param name="serial_baudrate"    type="int"    value="256000"--><!--A3 -->
  <param name="frame_id"          type="string" value="laser"/>
  <param name="inverted"          type="bool"   value="true"/>
  <param name="angle_compensate"    type="bool"   value="true"/>
  </node>

  <node pkg="unibot_gazebo" type="odom_msgs" name="odom_msgs" />
  <include file="$(find unibot_gazebo)/launch/rplidar_gmapping.launch" />

</launch>
```

7.7 Anexo VII

rplidar_gmapping.launch

```
<launch>
 <arg name="scan_topic" default="scan" />
 <arg name="base_frame" default="baselink" />
 <arg name="odom_frame" default="odom" />

 <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen" >
   <param name="base_frame" value="$(arg base_frame)"/>
   <param name="odom_frame" value="$(arg odom_frame)"/>
   <param name="map_update_interval" value="0.01"/>
   <param name="maxUrange" value="4.0"/>
   <param name="maxRange" value="5.0"/>
   <param name="sigma" value="0.05"/>
   <param name="kernelSize" value="3"/>
   <param name="lstep" value="0.05"/>
   <param name="astep" value="0.05"/>
   <param name="iterations" value="5"/>
   <param name="lsigma" value="0.075"/>
   <param name="ogain" value="3.0"/>
   <param name="lskip" value="0"/>
   <param name="minimumScore" value="30"/>
   <param name="srr" value="0.01"/>
   <param name="srt" value="0.02"/>
   <param name="str" value="0.01"/>
   <param name="stt" value="0.02"/>
   <param name="linearUpdate" value="0.05"/>
   <param name="angularUpdate" value="0.0436"/>
   <param name="temporalUpdate" value="-1.0"/>
   <param name="resampleThreshold" value="0.5"/>
   <param name="particles" value="8"/>

   <param name="xmin" value="-1.0"/>
   <param name="ymin" value="-1.0"/>
   <param name="xmax" value="1.0"/>
   <param name="ymax" value="1.0"/>

   <param name="delta" value="0.05"/>
```

```
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>

    <remap from="scan" to="$(arg scan_topic)"/>

  </node>

</launch>
```

7.8 Anexo VIII

Modelo de Clasificación de Imágenes de TensorFlow.

Desde Google Drive se crea un nuevo archivo de Google Colab. Se comienza importando TensorFlow y otras bibliotecas necesarias, como Keras.

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

Se descarga y explora el conjunto de datos. Este ejemplo utiliza un conjunto de datos de unas 3700 imágenes de flores, con un subdirectorio por clase de flor, cinco subdirectorios en total.

```
flower_photo/
  daisy/
  dandelion/
  roses/
  sunflowers/
  tulips/
```

```
import pathlib
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)
```

Después de la descarga, el archivo Colab debe tener una copia del conjunto de datos disponible, con 3670 imágenes en total.

```
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

Las imágenes se cargan fuera del disco usando *image_dataset_from_directory*. Esto lleva el directorio de imágenes en el disco a un *tf.data.Dataset*. Para crear el conjunto de datos, primero se definen los parámetros para la carga de los mismos.

```
batch_size = 32
img_height = 180
img_width = 180
```

Se empleará el 80% de las imágenes para entrenar el modelo y un 20% para la validación del modelo.

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

Los nombres de cada clase están almacenados en el atributo *class_names*, que se corresponden a los nombres de los directorios (la clase de flor que aparece en la foto).

```
class_names = train_ds.class_names
print(class_names)
```

Para visualizar los datos se pueden emplear las siguientes líneas de código que, a modo de ejemplo, visualizan las nueve primeras imágenes del conjunto de datos de entrenamiento.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

Para configurar el conjunto de datos para el rendimiento, se deben de haber cargado los datos correctamente. Hay dos métodos importantes para la carga de datos:

- *Dataset.cache( )* mantiene las imágenes en la memoria después de que se carguen fuera del disco durante la primera época. Esto asegura que el conjunto de datos no sature al modelo mientras se entrena.
- *Dataset.prefetch( )* superpone el preprocesamiento de datos y la ejecución del modelo mientras se entrena.

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Se deben estandarizar los datos, ya que los valores del canal RGB de las imágenes están en el rango [0, 255] y una red neuronal debe de tener unos valores de entrada pequeños para su correcto funcionamiento. Por ello, se estandarizan los valores para el rango [0, 1].

```
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
```

Tras preparar los datos, se debe crear el modelo. Este modelo consta de tres bloques de convolución con una capa de grupo máxima en cada uno.

```
num_classes = 5
model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

Para compilar el modelo se escribe lo siguiente:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Si se quiere visualizar un resumen de las capas de la red neuronal, se puede escribir:

```
model.summary()
```

Para entrenar el modelo se escribe:

```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

En la época diez se puede observar una precisión de entrenamiento del 99% y una precisión de validación del 64%. Esto significa que el modelo funciona con un 99% de efectividad con las imágenes de entrenamiento, pero con un 64% de efectividad con las imágenes de testeo. Esta diferencia de precisión es una señal de la presencia de sobreajuste. Cuando la cantidad de ejemplos para entrenamiento es pequeña, el modelo puede aprender ruidos o detalles que no son necesarios de los ejemplos de entrenamiento, lo que puede impactar negativamente en el desempeño del modelo con ejemplos nuevos. La presencia de sobreajuste implica que el modelo tendrá dificultades para funcionar con un nuevo conjunto de datos.

Una solución para este problema es el aumento de datos, que se basa en la generación de datos de entrenamiento adicionales a partir de los ejemplos ya existentes. Para implementar el aumento de datos se puede usar las capas de preprocesamiento de Keras.

```
data_augmentation = keras.Sequential(
 [
  layers.experimental.preprocessing.RandomFlip("horizontal",
                                 input_shape=(img_height,
                                        img_width,
                                        3)),
  layers.experimental.preprocessing.RandomRotation(0.1),
  layers.experimental.preprocessing.RandomZoom(0.1),
 ]
)
```

Una vez se corrige el sobreajuste, se vuelve a compilar y entrenar al modelo.

```
model.compile(optimizer='adam',
          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          metrics=['accuracy'])
epochs = 15
history = model.fit(
 train_ds,
 validation_data=val_ds,
 epochs=epochs
)
```

Para la época quince se puede observar una precisión de entrenamiento del 75% y una precisión de validación del 72%. Hay un menor sobreajuste y ahora las precisiones están más alineadas.

Finalmente, se puede emplear este modelo ya entrenado para clasificar una imagen que no conoce.

```
sunflower_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg"
sunflower_path = tf.keras.utils.get_file('Red_sunflower', origin=sunflower_url)
img = keras.preprocessing.image.load_img(
```

```
    sunflower_path, target_size=(img_height, img_width)
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch
predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

El modelo responde, con un 99,88% de confianza, que la flor de la imagen es un girasol (sunflower), lo que es totalmente correcto.

Este modelo ya está entrenado y es totalmente funcional, por lo que se puede implementar en otros dispositivos. Este ejemplo sería muy parecido a un modelo para el reconocimiento facial, solo hay que cargarle el nuevo conjunto de datos con las caras que debe reconocer y entrenarlo.

7.9 Anexo IX

reconocimientocaras.py

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL

import tensorflow as tf

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

import pathlib
##dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
##data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
##print(data_dir)
data_dir = pathlib.Path('/Users/adriansanchezgranero/Documents/TFG/Faces')
print(data_dir)

image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)

adrian = list(data_dir.glob('adrian/*'))
PIL.Image.open(str(adrian[0]))

batch_size = 32
img_height = 100
img_width = 100

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.4,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.4,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

class_names = train_ds.class_names
print(class_names)

'''plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")

for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
'''
from tensorflow.keras import layers

##Reescalamos y normalizamos para un mejor rendimiento
normalization_layer = tf.keras.layers.experimental.preprocessing.Rescaling(1./255)

normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixels values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

```
num_classes = 2

model = tf.keras.Sequential([
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(64, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(60, activation='relu'),
  layers.Dense(num_classes)
])

model.compile(
  optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])

epochs = 2

history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)

model.summary()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

'''plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
```

```
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()'''



print("Guardando el modelo")

model.save('/Users/adriansanchezgranero/Documents/TFG/model/mimodeloguardado')

adrian_path = "/Users/adriansanchezgranero/Documents/TFG/Muestras foto cara/irene1.png"

img = keras.preprocessing.image.load_img(
    adrian_path, target_size=(img_height, img_width)
)

print("Voy por aquí")

img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```