UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Interfaz de usuario y modelado del entorno 3d para robot manipulador móvil de intervención

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Alejandro Arnal Espinola

Tutor: María Pilar Tormos Juan
Román Navarro García

Curso 2017-2018

Resum

En el present treball es dissenya una interfície web estàndard a totes les plataformes mòbils ofertes per l'empresa de robòtica Robotnik Automation. Per a això, s'ha partit d'una interfície específica ja existent en la qual a més del redisseny del web, s'han implementat noves funcionalitats que permet conèixer l'estat del robot de forma remota en el temps real. També s'han implementat algunes funcions d'interacció amb el robot com la teleoperació a través de la pàgina web o la possibilitat de mapejar en tres dimensions mitjançant l'ús de càmeres amb sensor de profunditat.

Paraules clau: ROS, interfície web, robòtica, mapejat 3D, robot, Django

Resumen

En el presente trabajo se diseña una interfaz web estándar a todas las plataformas móviles ofertadas por la empresa de robótica Robotnik Automation. Para ello, se ha partido de una interfaz específica ya existente en la que además del rediseño de la web, se han implementado nuevas funcionalidades que permiten conocer el estado del robot de forma remota en tiempo real. También se han implementado algunas funciones de interacción con el robot como la teleoperación a través de la página web o la posibilidad de mapear en tres dimensiones mediante el uso de cámaras con sensor de profundidad.

Palabras clave: ROS, interfaz web, robótica, mapeado 3D, robot, Django

Abstract

In the present work, a standard web interface is designed for all mobile platforms offered by robotics company Robotnik Automation. To do this, it has started with an already existing specific interface in which, in addition to redesigning the web, new features have been implemented that allow knowing the status of the robot remotely in real time. Some functions of interaction with the robot have also been implemented, such as teleoperation through the website or the possibility of mapping in three dimensions by using cameras with depth sensor.

Key words: ROS, web interface, robotics, 3D mapping, robot, Django

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

La creciente necesidad de automatizar tareas que resultan pesadas, repetitivas o incluso peligrosas ha provocado que a medida que pasa el tiempo, sea más común ver dispositivos robotizados en nuestro entorno cotidiano realizando dichas tareas. De esta forma, nos podemos encontramos desde robots que automatizan una cadena de montaje hasta robots que nos ahorran la tarea de aspirar en casa. En el contexto de la robótica, más concretamente en la robótica de servicio, se enmarca el presente Trabajo Fin de Grado (TFG).

Según la Organización Internacional de la Normalización (ISO), se define como robot de servicio en la norma ISO 8373 [1] a cualquier robot que realiza una tarea útil para un ser humano y que no es considerado como robot industrial. En dicha norma, un robot de ámbito industrial es un manipulador multifuncional, reprogramable en tres o más ejes que se mueve de forma autónoma y puede estar fijo o ser móvil cuyo tarea está englobada en el ámbito industrial. De igual forma, la Federación Internacional de Robótica (IFR) categoriza como robot de servicio a todo aquel que, independientemente de su autonomía, realiza una tarea (no industrial) para una persona [2].

Este trabajo pretende facilitar la interacción del usuario con una plataforma móvil robótica de servicio cuya finalidad es dar soporte en misiones EOD (*Explosive Ordnance Disposal*). Para ello se plantea el desarrollo de una aplicación web a través de la cual el usuario será capaz de visualizar datos de interés del robot como la batería, la localización dentro de un mapa y el estado de los motores entre otros, además de permitir comandar y teleoperar al robot desde dicha interfaz web.

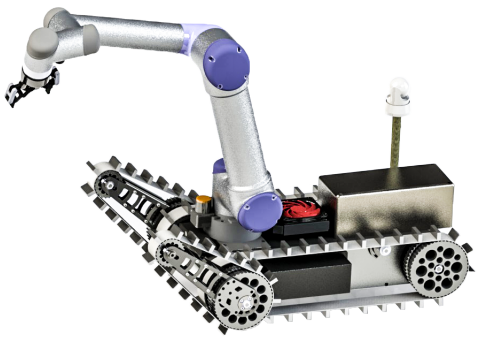

Figura 1.1: Plataforma Rising.

1.1 Objetivos

El objetivo principal de este trabajo reside en el diseño de una interfaz web, para la plataforma móvil Rising[1], en la que el usuario sea capaz de visualizar información útil aportada por el software y hardware del robot.

Con la finalidad de poder reutilizar dicha aplicación web en el futuro, se pretende realizar un desarrollo lo más fiel posible a la configuración común a todas las plataformas móviles ofertadas por la empresa Robotnik Automation SLL.

Debido a las características específicas del robot Rising y la finalidad del mismo, que se presenta en el próximo capítulo, se ha establecido también como objetivo, el estudio de la compatibilidad de software basado en ROS para el mapeado en tres dimensiones del entorno con Intel Realsense D435, el sensor RGBD (*Red Green Blue Depth*) que tiene equipado la plataforma Rising.

Se han fijado una serie de subobjetivos que permitirán cumplir los requisitos del proyecto:

- Desarrollar una interfaz estándar para todos los robots de Robotnik a partir de una aplicación específica ya existente.
- Modularizar la página web en función de la información mostrada para permitir que el cliente pueda configurar qué información desea visualizar.
- Probar los paquetes RTAB-Map[2] y RGBDSLAM[3] de ROS para el dispositivo Intel Realsense D435.
- Añadir la funcionalidad de mapeado en tres dimensiones a la página web.

1.2 Estructura

La presente memoria está estructurada a lo largo de nueve capítulos. En el actual, y primer capítulo , se describe el proyecto así como sus objetivos principales.

Seguidamente, en segundo capítulo, se presenta la empresa para la que va dirigida este proyecto y se contextualiza dentro de dicha empresa. Igualmente, se analiza un trabajo de temática similar realizado por un antiguos alumno de la Escuela Técnica Superior de Ingeniería Infromática (ETSINF) de la Universidad Politécnica de Valencia (UPV) y se realiza una comparación entre la solución aportada por dicho trabajo y la desarrollada en este proyecto.

El capítulo tres expone el problema que debe solventar este TFG. También se presentan la solución al problema escogida, así como otras posibles soluciones disponibles en el mercado.

A continuación, en el cuarto capítulo, se describe detalladamente la solución elegida para la aplicación. Se muestra la arquitectura del sistema y los requisitos mínimos del mismo además de presentar y explicar las tecnologías que se han utilizado para la ejecución de la aplicación.

En el capítulo cinco se describen cada una de las etapas del desarrollo de la aplicación web. Explicando las adaptaciones realizadas en la web específia ya desarrollada, ade-

[1] Plataforma móvil RISING - https://www.robotnik.es/manipuladores-roboticos-moviles/rissing
[2] RTAB-Map - http://introlab.github.io/rtabmap
[3] RGBDSLAM - https://felixendres.github.io/rgbdslam_v2

más de explicar las nuevas funcionalidades que se han implementado en la interfaz de usuario.

En el sexto capítulo se desarrolla el proceso de implantación de la aplicación en el espacio de trabajo real. Seguidamente, en el capítulo capítulo siete, se describen las pruebas realizadas con usuarios no familiarizados con el sistema. También se ha hecho un análisis de los resultados obtenidos a partir de las pruebas con la finalidad de verificar la usabilidad de la aplicación web.

El octavo capítulo consiste en las conclusiones obtenidas a lo largo del desarrollo de la aplicación web. En este capítulo se plasman los conocimientos obtenidos por el alumno además de hacer un análisis de las tecnologías utilizadas en relación con el temario estudiado a lo largo del Grado en Ingeniería Informática en la ETSINF.

En el último capítulo se han planteado algunas posibles mejoras de la aplicación que podrían desarrollarse en el futuro.

CAPÍTULO 2
Estado del arte

Desde apretar botones y ver bombillas enciendiéndose y apagándose a ordenadores que muestran gráficamente toda la información del sistema[3], en pocos años las interfaces de interacción entre máquinas y humanos han evolucionado exponencialmente.

Para la robótica industrial tradicional bastaba con un par de botones que permitiesen realizar acciones repetitivas y un par de señales luminosas que mostrasen el estado de la máquina. Sin embargo, con la aparición de los robots colaborativos esto ha cambiado. Los robots colaborativos tienen algunas diferencias con respecto a los robots tradicionales: son fácilmente integrables en diferentes entornos, interactúan con trabajadores e incluyen sensores de seguridad que les permiten compartir espacio con humanos [4].

Figura 2.1: RB2-Base. Robot colaborativo de Robotnik.

Es por esto que aparece la necesidad de crear nuevas formas de interactuar con un robot de una forma más sencilla.

En el ámbito de la robótica colaborativa se encuentra Robotnik Automation SLL. Robotnik es una empresa dedicada al desarrollo de producto y prestación de servicios de ingeniería e I+D en el área de la robótica. La actividad profesional de la empresa se centra en la producción de robots autónomos de transporte interior y aplicaciones de robótica de servicio así como la participación en proyectos de I+D en robótica y diseño de sistemas robotizados a medida.

Robotnik lleva desde 2004 participando en proyectos de I+D dentro del Programa Marco[1]de la Unión Europea (UE) además de en numerosos proyectos de investigación a nivel nacional. En este contexto, uno de los proyectos en los que actualmente está participando la empresa es el proyecto RISING [2].

[1] Programa Marco - https://eshorizonte2020.es/mas-europa/71-programa-marco
[2] Proyecto RISING - https://www.robotnik.es/rising

Mediante el proyecto RISING se pretende el desarrollo de una nueva plataforma robótica multipropósito con manipuladores, sistemas de comunicaciones y capacidad de procesamiento avanzadas. El objetivo principal del proyecto consiste en el desarrollo de un sistema robotizado que asista en operaciones de cuerpos de nacionales de seguridad en entornos difíciles. Entre los requisitos establecidos en el proyecto se encuentra el desarrollo de una interfaz de usuario que permita la visualización de datos útiles del robot en tiempo real. El presente TFG pretende abarcar dicho requisito.

Robotnik Automation desarrolla todos sus robots basándose en *Robot Operating System* (ROS). En el contexto de este *framework* de trabajo existen muchas interfaces de usuario desarrolladas previamente [5]. La empresa, en concreto, dispone de varias webs de uso específico de proyectos en los que ha trabajado anteriormente. Para el desarrollo de la aplicación web que se presenta en este TFG, se partirá de una de las páginas webs desarrolladas anteriormente por Robotnik. La página web de la que se va a partir, es a su vez, es el TFG de un antiguo alumno de la ETSINF, por lo que en el próximo apartado, se presentará dicha web y se hará un análisis sobre las mejoras que se pretenden aplicar.

2.1 Crítica al estado del arte

Como se ha comentado anteriormente, el trabajo *Diseño e implementación de una interfaz gráfica de usuario para mapeado de entornos y navegación en ROS*[3] de José Manuel Rapado García, servirá como punto de inicio para el desarrollo de la nueva interfaz web.

Se trata de una interfaz compuesta por tres vistas (inicio, mapeado y navegación) desarrollada exclusivamente para un caso de uso de un proyecto en el que participó Robotnik anteriormente. Con el presente proyecto se pretende reutilizar las funcionalidades ya implementadas para crear una interfaz con nuevas utilidades que sea útil para el robot Rising y otras plataformas móviles de Robotnik.

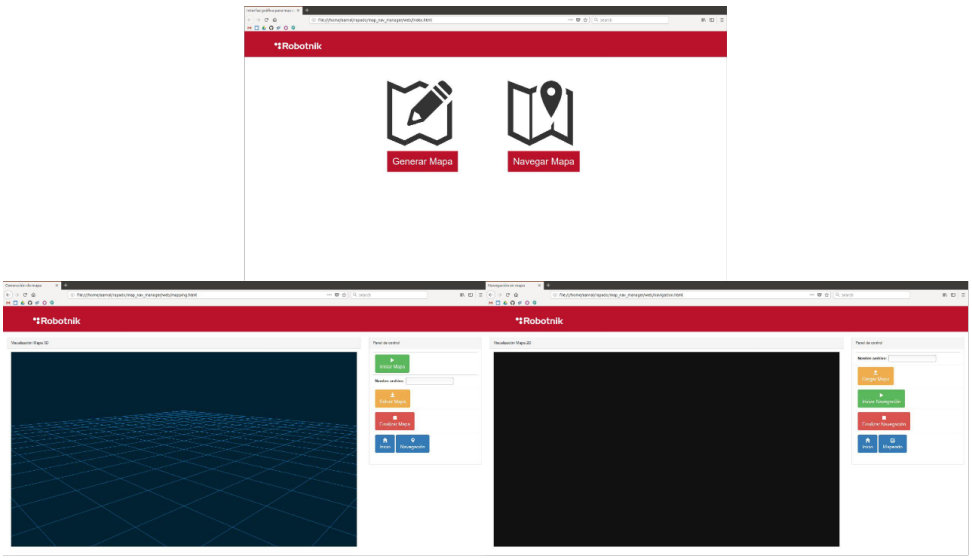

Figura 2.2: Antigua interfaz de usuario.

[3] Diseño e implementación de una interfaz gráfica de usuario para mapeado de entornos y navegación en ROS https://riunet.upv.es/handle/10251/69347

2.2 Propuesta

Este trabajo tiene como finalidad el desarrollo de una aplicación web que permita al usuario conocer el estado de la plataforma Rising en cualquier momento. Debido a las caraterísticas y el propósito de este robot, se ha establecido la posibilidad de poder realizar, a través de interfaz web, mapeado en tres dimensiones mediante el uso de un sensor RGBD.

Como se ha comentado anteriormente en la sección de objetivos, se pretende crear una aplicación que cumpla los requisitos específicos del robot Rising pero que de igual forma sea lo más estándar posible al resto de plataformas móviles que oferta la empresa Robotnik con la finalidad de hacer que esta aplicación sea fácilmente escalable o reutilizada por la empresa en proyectos futuros.

Para ello, se va a partir del proyecto presenteda en la sección previa a la presente. Se pretende unificar todas las vistas de dicha aplicación en una única vista más compacta y sencilla, que a parte de proporcionar las funcionalidades de mapeado y navegación que ya estaban implementadas en la versión inicial de la web, también permita ver el estado del robot, como por ejemplo: el estado en el que se encuentran los drivers de los motores, la posición y orientación del robot con respecto al punto de arranque del mismo o información sobre los sensores que lleva equipado.

CAPÍTULO 3
Análisis del problema

Como paso previo al desarrollo del proyecto, se necesita especificar cuál va a ser el alcance del mismo. Para ello, en este capítulo se van a definir cuáles son las necesidades que la aplicación pretende cubrir.

Además de la definición de las funcionalidades que se buscan implementar en la aplicación, en este capítulo se va a analizar algunas soluciones posibles al problema. Por otra parte, se expone cuál ha sido la solución elegida finalmente para el desarrollo de este trabajo.

Las funcionalidades que se pretenden implementar en la aplicación pueden ser reflejadas a través de un diagrama de casos de usos como se expone en la siguiente figura.

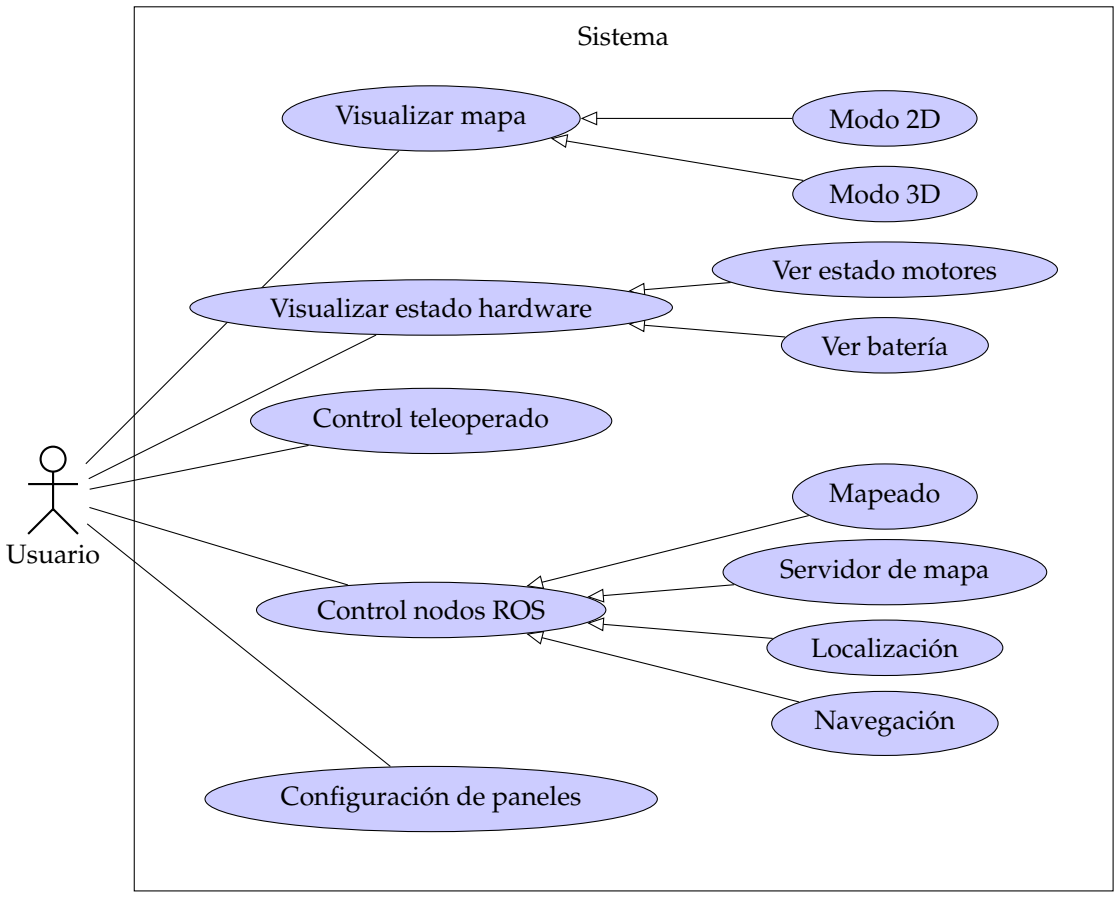

Figura 3.1: Diagrama de casos de uso de la aplicación.

Como se puede observar, a parte de permitir al usuario visualizar un mapa en dos dimensiones, como ya permitía la versión previa de la página web, ahora también se puede visualizar un mapa en tres dimensiones.

El usuario también tendrá la posibilidad de visualizar el estado de los motores, teniendo acceso los distintos *flags* que proporcionan los *drivers* de los motores. Además podrá saber cuánta batería le queda al robot y teleoperarlo desde la interfaz web.

Para hacer más personalizable la vista de la aplicación, el usuario también puede modificar la visibilidad de la misma a través de la pestaña de configuración.

De entre los casos de uso expuestos, cabe destacar aquellos relacionados con el control de nodos de ROS, por lo que en la siguiente figura se muestra un diagrama de secuencia de todo el proceso de inicio o parada de un nodo de ROS a través de la página web. Esto permite ver todo el proceso de llamadas internas que realiza la aplicación cada vez que se quiere inicializar o terminar un nodo de ROS.

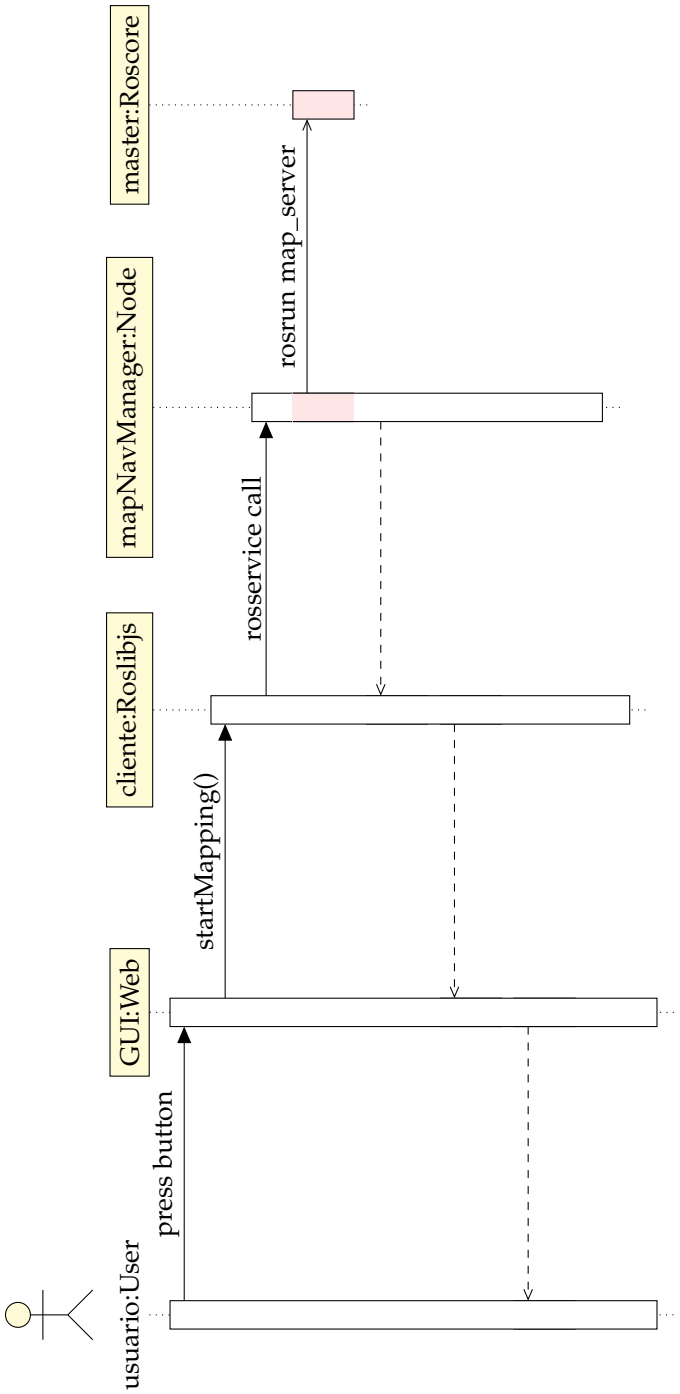

Figura 3.2: Diagrama de secuencia. Control de nodos ROS

3.1 Identificación y análisis de soluciones posibles

Una vez identificado el problema y haber evaluado los requisitos de la aplicación, se puede hacer un análisis sobre las posibles soluciones que solventarían las necesidades del cliente de igual forma que lo hace la solución propuesta en el presente trabajo.

Una posible solución al problema planteado sería el desarrollo de una aplicación de escritorio a través de *ROS Qt Creator Plug-in* [1], un entorno de desarrollo integrado (IDE) que facilita la creación de aplicaciones, compatibles con ROS, basadas en Qt [2]. A pesar de permitir al usuario la visualización de datos y la interacción con el robot a través de una interfaz gráfica, esta solución presenta algunas desventajas:

- El usuario necesitaría tener instalado ROS en su máquina remota para poder visualizar los datos provenientes del sistema robotizado.
- La aplicación desarrollada debe ser instalada en el dispositivo desde el que el usuario quiera controlar el robot de forma remota.

Otra planteamiento de la solución sería usar la herramienta RVIZ [3]. Se trata de una herramienta basada en Qt, que permite la visualización de datos de sistema que usa ROS, en el próximo capítulo se detallan algunas características de dicha herramienta. De la misma forma que la solución anterior, el usuario tiene que tener instalado ROS en su máquina para poder hacer uso de esta utilidad. Presenta, además, algunas desventajas adicionales:

- Algunas estructuras de datos necesitan de *plugins* de terceros para ser visualizados en RVIZ o incluso es necesario desarrollar dichos *plugins*.
- No todos los datos son visualizables en RVIZ.

En la siguiente figura se muestra el uso de la herramienta RVIZ para la plataforma móvil Rising.

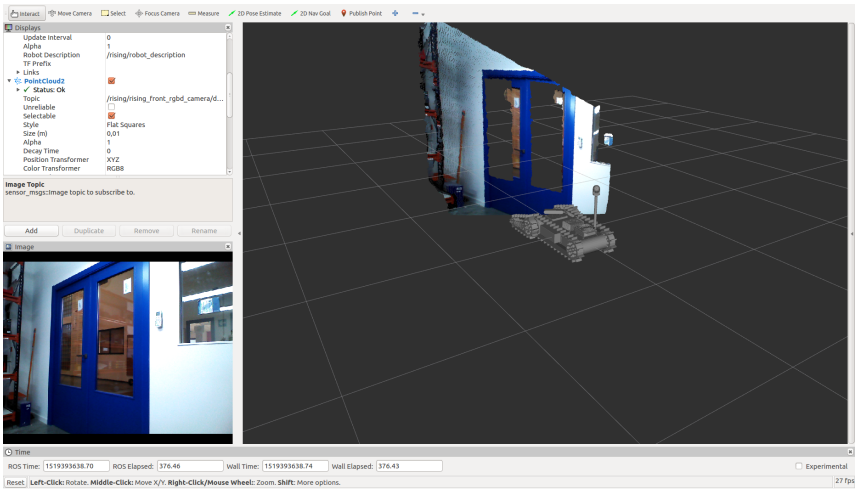

Figura 3.3: Visualización de la plataforma Rising en RVIZ

Como se puede observar, a pesar de tratarse de soluciones que cumplen con los requerimientos del problema, no pueden ser consideradas como soluciones factibles ya que

[1] ROS Qt Creator - https://github.com/ros-industrial/ros_qtc_plugin
[2] Qt - https://www.qt.io
[3] RVIZ - http://wiki.ros.org/rviz

obliga al usuario a tener instalado ROS, y por consiguiente un sistema operativo compatible. Además de la instalación de ROS, la configuración del *framework* para trabajar con un robot desde un equipo remoto, puede ser tediosa para un usuario no familiarizado con el uso de la terminal del sistema.

3.2 Solución propuesta

Debido a las restricciones que presentan las soluciones alternativas comentadas anteriormente y teniendo en cuenta las necesidades de la aplicación final, se ha elegido como solución el desarrollo de una interfaz web.

Entre los aspectos más interesantes de desarrollar una aplicación web está la posibilidad de permitir al usuario poder visualizar datos provenientes del robot, de igual forma que lo podría hacer con RVIZ o una aplicación Qt, sin necesidad de instalar ningún tipo de software o requerir de algún tipo de conocimiento específico para el uso de la aplicación.

Para poder implementar esta aplicación, se hace uso de *roslibjs* [4], una librería de JavaScript que permite la interacción con ROS a través de un navegador. La solución propuesta se trata de un aplicación web, desarrollada en Django[5], que es alojada localmente en el robot, haciéndola únicamente accesible desde la red local proporcionada por el sistema robótico. De esta forma se permite tener acceso al estado del robot de una forma relativamente sencilla, restringiendo el acceso únicamente a aquellos usuarios autorizados para acceder a la red del robot.

Para ello, se va a partir de una aplicación web ya existente en la que es posible mapear en dos dimensiones y navegar a través de un mapa. Se pretende unificar las funcionalidades ya existentes con nuevas caracterísitcas (como poder ver el estado de los motores, teleoperarlo o poder visualizar información de los sensores con los que va equipado el robot) en una única vista completamente personalizable por el usuario.

[4] Roslibjs - https://github.com/RobotWebTools/roslibjs
[5] Django - https://www.djangoproject.com/start/overview

CAPÍTULO 4
Diseño de la solución

Una vez se tiene claro los requisitos de la aplicación, se puede ahondar en qué estructura va a tener nuestro sistema y cuáles son las herramientas más adecuadas que se adapatan a las necesidades del mismo.

4.1 Arquitectura del sistema

Los robots de la empresa Robotnik Automation, poseen una red local que permite a los usuarios acceder remotamente a ellos. En la siguiente figura se presenta la estructura de la red del sistema robotizado para su fácil comprensión.

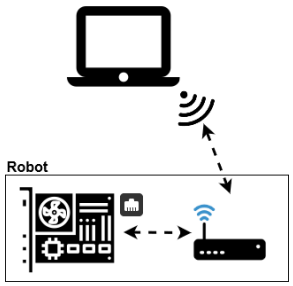

Figura 4.1: Acceso al robot a través de la red local.

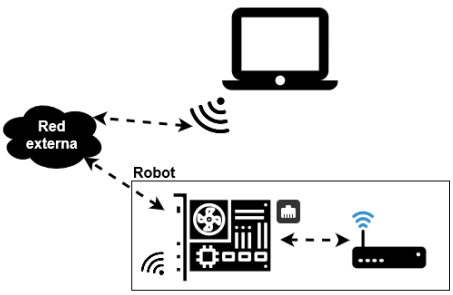

Figura 4.2: Acceso al robot a través de una red externa.

El sistema robotizado tiene una red de área local servida por un *router*. Comunmente, este *router* está configurado como un punto de acceso a través del cual el ordenador del robot está conectado a la red local usando una conexión *ethernet*. En algunos casos, la placa base del robot posee *Wi-Fi*, que permite al robot estar conectado al mismo tiempo a

su red de área local y a otra red distinta. En este último caso, el cliente puede acceder al robot de dos formas: uniéndose a la red local del robot a través de la red *Wi-Fi* que crea el *router* robot (Figura 4.1) o a través de la red a la que se conecta el robot usando el *Wi-Fi* de su placa base (Figura 4.2).

La aplicación web que se desarrolla en este proyecto será servida por el ordenador del robot, por lo que será elección del cliente a través de qué red acceder a ella. De esta manera, si el usuario se encuentra en un espacio donde hay diversas redes disponibles, puede elegir si acceder al robot a través de una red externa, habiendo conectado previamente el robot a dicha red, o a través de la local. De igual forma, si se encuentra en un espacio en el que no existen otras redes, siempre puede acceder desde la red local.

El único requisito que presenta la aplicación web con respecto a la configuración estándar de los robots que oferta la empresa Robotnik, es la instalación del *framework* Django.

4.2 Diseño detallado

La aplicación web tendrá en todo momento una vista personalizada a las necesidades del cliente. Esto se consigue gracias a un archivo de configuración en el que se definen toda la información que se puede visualizar dentro de la página web. A partir de este archivo de configuración, Django es capaz de mostrar en la interfaz todas aquellas características que están implementadas y generar una vista personalizada a las necesidades del usuario. Desde la propia aplicación web, el ususario puede modificar la visibilidad de las utilidades que proporciona la web y guardar dicha configuración.

Una vez generado la vista personalizada con algunos campos ya rellenados gracias a la configuración establecida, es necesario rellenar la información que es proporcionada por el robot a través de ROS. Para ello, se ha utilizado *rosbrdige*, que se explicará más adelante.

4.3 Tecnologías utilizadas

4.3.1. ROS

Robot Operating System es considerado un metasistema operativo ya que provee al usuario tanto de servicios estándares propios de un sistema operativo (abstracción del *hardware*, paso de mensajes entre procesos, gestión de paquetes, etc.), como de herramientas y librerías para la creación de sistemas robóticos descentralizados.

Los elementos fundamentales de un sistema basado en ROS son:

- **Nodos**: Los nodos son procesos que tienen como objetivo realizar una función específica dentro del sistema robótico. Pueden coexistir infinidad de nodos al mismo tiempo dentro de un robot basado en ROS.
- **Master**: Es la entidad encargada de gestionar a todos los nodos. Entre sus funciones se encuentra el registro de nombres de los nodos y el control de la interacción entre los nodos a través de la gestión de publicaciones y subscripciones a los *topics*, que se explicarán más adelante en esta sección.
- **Servidor de parámetros**: Es un diccionario accesible por todos los nodos. Es creado al iniciar el sistema a partir de los parámetros que se han usado para la ejecución

de todos los nodos. A pesar de que está pensado para ser estático, es decir, mantenerse en el mismo estado durante toda la ejecución, permite la reconfiguración de parámetros en tiempo de ejecución.

Entre todas las características y herramientas que ofrece ROS, se pueden destacar tres:

- **Infraestructura de comunicación**: También referida a ella como el *middleware* de ROS, provee al sistema de una interfaz de comunicación que permite a los diferentes procesos compartir información de una forma bastante sencilla. Algunas características de esta infraestructura son:

  - **Mensajes**: ROS tiene unos tipos de mensajes básicos para encapsular información. A partir de estos mensajes básicos se puede definir fácilmente mensajes más complejos que se ajusten a los requisitos del robot.
  - **Topics**: Son buses tipados con mensajes de ROS. Los nodos del sistema pueden publicar información en estos buses para que sea accesible al resto de nodos o puede leer mensajes de estos *topics* para el posterior tratamiento de la información. En un *topics* puede haber un número indefinido de publicadores, que generan el mensaje y lo mandan, y de suscriptores, que leen mensajes y tratan la información.

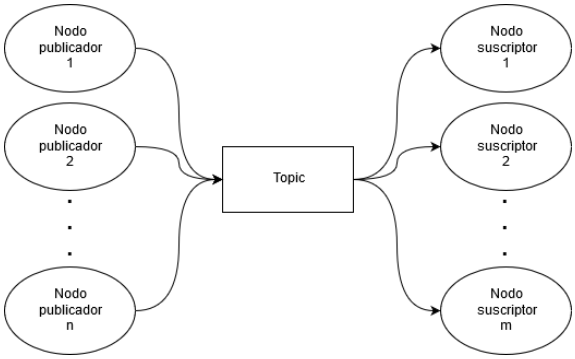

Figura 4.3: Flujo de mensajes a través de *topics*

  - **Servicios**: Son un tipo de comunicación entre dos nodos entre los que se establece una relación cliente/servidor. Es una interacción RPC (*Remote Procedure Call*) en la que el nodo servidor ejecutará una acción concreta cuando reciba una petición (*request*) de un cliente. Una vez acabado el cómputo, el servidor responde *(reply)* al cliente con el resultado de la llamada al servicio. Los servicios tienen un tipo de mensajes especiales compuestos por la petición y la respuesta. Estos mensajes se pueden definir de forma similar a la de los *topics*.
  - **Rosbag**: Permite guardar, para su posterior reproducción, el flujo de mensajes que se produce a través de los *topics*.

- **Software específico de robótica**: ROS provee al usuario de diversas librerías y herramientas específicas en el campo de la robótica que permiten una configuración fácil de un sistema robotizado. Entre otros, se pueden destacar los paquetes de navegación, de localización y de diagnóstico del sistema.

- **Herramientas**: ROS ofrece gran cantidad de heramientas que permiten al usuario visualizar los datos que se están generando en su sistema robotizado, lo que facilita la tarea de depuración dicho sistema. Se pueden destacar:

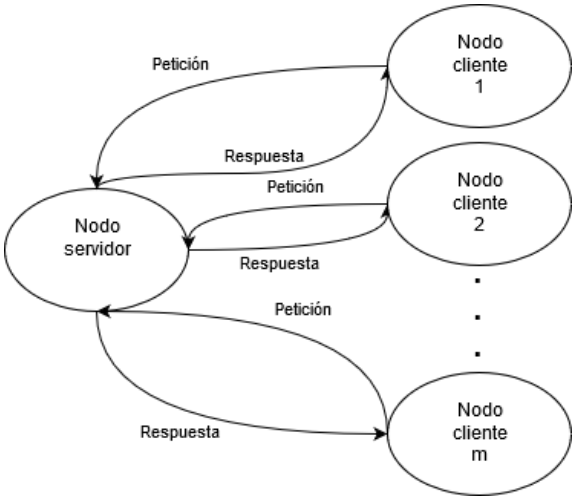

Figura 4.4: Flujo de mensajes a través de servicios de ROS

- **RVIZ**: Es una herramienta para la visualización en tres dimensiones de la información publicada en los *topics* del robot. Permite al usuario ver desde las lecturas de los láseres del robot hasta la imagen capturada por una cámara. De igual forma, también es posible visualizar el modelo tridimensional del robot para facilitar la compresión de los datos en el entorno virtual.

- **Rqt**[1]: Proporciona un *framework* basado en Qt para la creación de interfaces personalizadas que permiten: la visualización de los datos en tiempo real y la interacción con el sistema robótico a través de publicaciones en *topics* o llamadas a servicios de ROS. Rqt proporciona algunos *plugins* básicos para el tratamientos de los datos del robot sin necesidad de crear una interfaz específica.

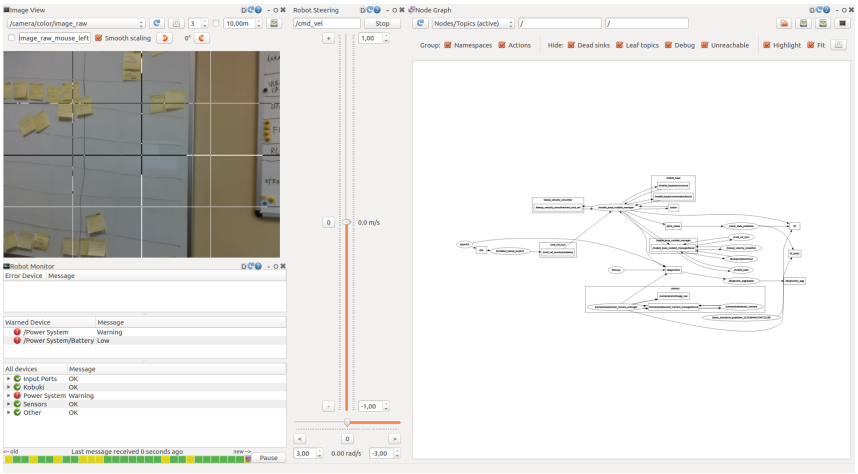

Figura 4.5: Configuración de rqt con diversos plugins

También cabe destacar *rosbridge*, una API JSON que permite la interacción de sistemas basados en ROS con otros sistemas no necesariamente basados en ROS.

[1]Rqt - http://wiki.ros.org/rqt

4.3.2. Django

Django es un *framework* de desarrollo web basado en *Python*. Se pueden destacar tres elementos característicos:

- *Model*: Un modelo es una clase que representa algo con lo que se va a trabajar en la aplciación. Suelen representar tablas de una base de datos, por lo que es común que existan atributos en los modelos y relaciones con otros. Al definir un modelo, Django sabrá como crear una tabla dentro de la base de datos que represente toda la información asociada, además de proveer al desarrollador de algunos métodos útiles para el manejo de datos.
- *Template*: Representa el formato que se le quiere dar a la información. Son archivos *HyperText Markup Language* (HTML) en los que no está toda la información. Los *templates* son útiles para definir un formato que se va a repetir varias veces en la web pero con contenidos distintos. Usando parámetros dentro de *templates*, Django es capaz de completar ese archivo con información.
- *Views*: Es una función encargada de completar la información de los *templates* a través del uso de los *models*.

En la siguiente figura se muestra la relación que existe entre estos elementos.

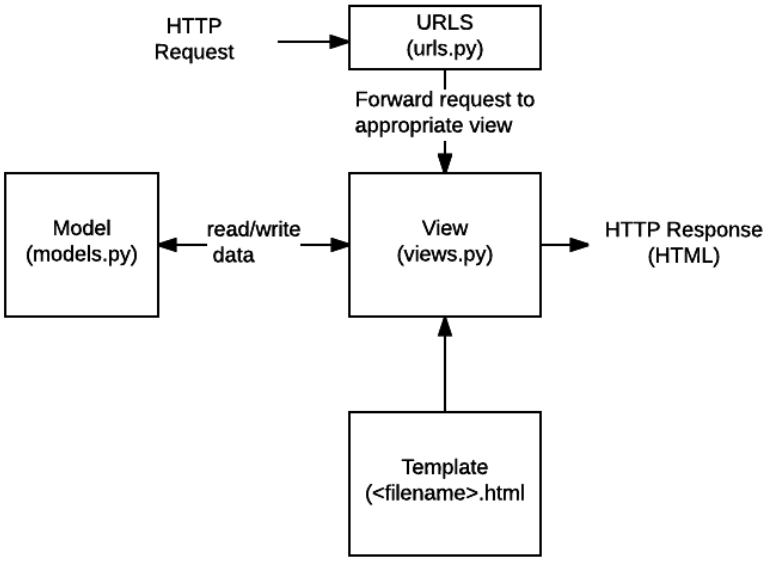

Figura 4.6: Esquema de arquitectura Django.

CAPÍTULO 5

Desarrollo de la solución propuesta

Para el desarrollo de este proyecto, por una parte se ha adaptado y aumentado las funcionalidades de la web y por otra parte se han ajustado los paquetes de mapeado en tres dimensiones para su funcionamiento con la cámara Intel Realsense D435.

5.1 Aplicación web

Para el desarrollo de la página web se ha utilizado Django como entorno de trabajo. Como se ha comentado anteriormente, se parte de una aplicación ya desarrollada. Para dicha aplicación, el servidor era instanciado haciendo uso de la librería estándar *SimpleHTTPServer* [1] de Python.

Los recursos de la página web original se estructuraban como se refleja en la siguiente figura.

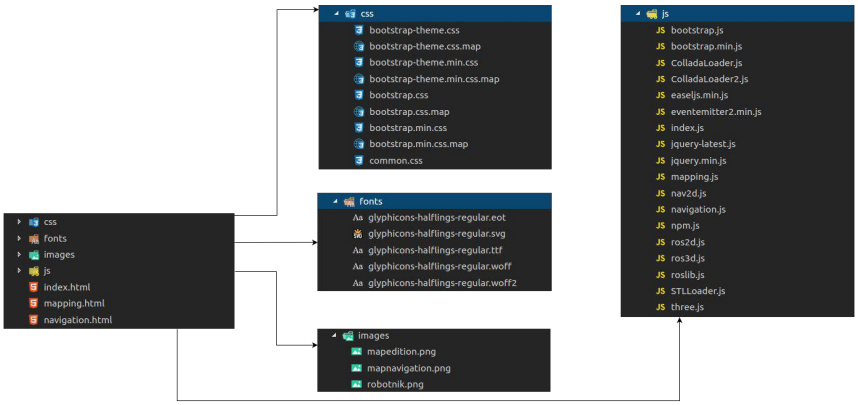

Figura 5.1: Estructura de carpetas original

Como se puede observar, la página de la que se parte esta compuesta por tres vistas: *index*, *mapping* y *navigation*. Además tiene un directorio que contiene todos los archivos de estilos, otro que contiene los archivos de las fuentes que utiliza la página web y por último, una carpeta en la que se encuentran todos los archivos JavaScript.

Para el desarrollo de la nueva interfaz web, en primer lugar se han repartido los recursos de la antigua página siguiendo la estructura de carpetas que establece Django.

Debido a que uno de los objetivos que se pretende conseguir con la aplicación es la posibilidad de hacer que la vista de la misma sea personalizable a gusto del cliente, se

[1] SimpleHTTPServer - https://docs.python.org/2/library/simplehttpserver.html

ha apostado por la modularización de la interfaz principal en vistas parciales. De igual forma, toda la lógica que había implementada anteriormente en la página web, así como las nuevas funcionalidades desarrolladas, han sido modularizadas. De esta forma, en vez de tener todas las funcionalidades centralizadas en un único archivo, se dispone de varios archivos que implementan las características específicas de cada parte de la página.

Una vez realizada dicha reestructuración, se obtiene una jerarquía de directorios como la que se muestra en la siguiente figura.

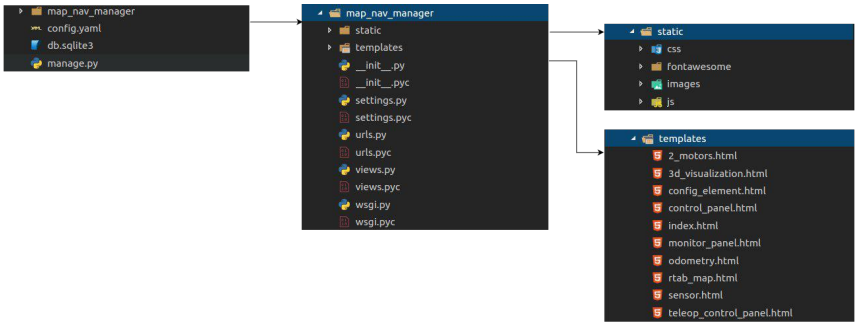

Figura 5.2: Estructura de carpetas actual.

En la carpeta raíz del proyecto se encuentra el archivo de configuracion (*config.yaml*) que es utilizado para definir todas las funcionalidades que tiene la página web.

Dentro de la carpeta *map_nav_manager* se encuentran todos los recursos de la web. Por una parte está la carpeta *templates*, donde están todos los archivos *.html*. El archivo *index.html* define la vista principal, la lógica y estilos básicos de la aplicación web. A partir de la configuración cargada desde el archivo de configuración mencionado anteriormente, Django estructura la vista principal con las vistas parciales correspondientes. La lógica y estilos específicos de cada componente de la página están incluidos en los *templates* que les corresponde, por lo que únicamente serán cargados si realmente son utilizados.

Comúnmente, el usuario usará una única configuración que se adapte a sus necesidades, por lo que con la modularización de las funcionalidades que se ha planteado, la página únciamente cargará aquellos recursos que realmente necesita.

Tras la adaptación a la nueva estructura de carpetas y el rediseño, la vista principal de la web tiene el siguiente aspecto:

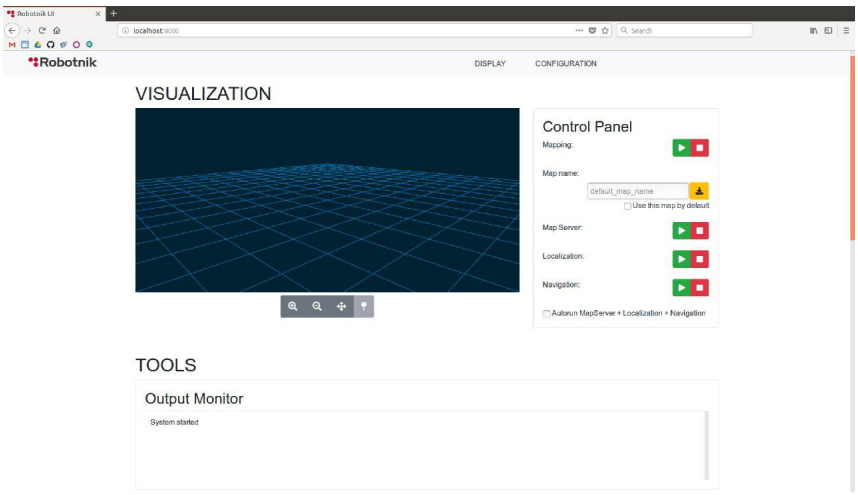

Figura 5.3: Nueva interfaz web.

Ante la posibilidad de utilizar la página web desde dispositivos de diferentes resoluciones, se ha aprovechado el rediseño de la web para hacer que la nueva aplicación sea *responsive* y se adapte a cualquier dispositivo.

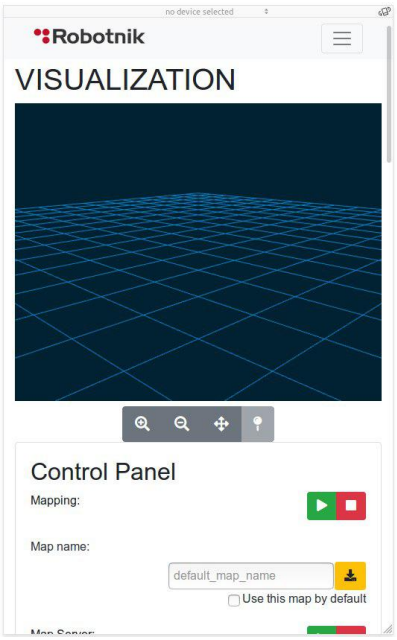

Figura 5.4: Nueva interfaz web en dispositivo móvil.

Como se ha expuesto anteriormente, a parte de integrar las funcionalidades de las que disponía la aplicación original, se han añadido nuevas características como:

- **Monitor**: Un panel que permite mostrar información sobre qué está sucediendo en el robot. El resto de secciones de la página publican información en este monitor si el usuario lo tiene disponible reflejando información sobre los comandos que se le están mandado al robot o la información que está generando.
- **Panel de teleoperación**: Se trata de cuatro flechas que permiten mover el robot desde la página web. Para ello, cada vez que se pulsa en una de ellas, se manda un comando de velocidad al *topic* correspondiente.
- **Estado de los motores**: Permite visualizar información sobre los motores. Para conseguir esto, se lee del *topic* de ROS de los motores y se formatea dichos mensajes en información legible para el usuario.
- **Odometría**: Muestra información de la posición y la orientación del robot con la posición inicial del mismo cuando es encendido. Dispone de un botón que permite resetear este sistema de referencia.
- **Estado de los sensores**: En estos paneles se muestra el tipo del sensor, el modelo del mismo y además se permite al usuario comprobar si están funcionando correctamente a través de un botón.

5.2 Paquetes de mapeado

Para el uso de la cámara, en primer lugar ha sido necesario situar el modelo de la cámara con la orientación y posición correspondiente con respecto al modelo del robot.

Esto se debe a que ambos paquetes usan un eje de coordenadas base que se corresponde con el de la base del robot y otro que se corresponde con el de la base de la cámara. Estableciendo una relación de orientación y posición entre estos dos ejes y conociendo los movimientos que realiza el robot, ambos algoritmos son capaces de reconstruir mapas a través de los datos leídos de la cámara Intel Realsense D435.

Para el uso con ROS, se ha consultado qué datos necesitan los paquetes de para poder mapear en tres dimensiones. Los tipos de mensaje ROS que requieren son: *CameraInfo* y *Image*. El mensaje del tipo *CameraInfo* tiene información específica del dispositivo que se está utilizando, mientras que el de tipo *Image* tiene codificada la imagen que se está viendo con el sensor. Ambos paquetes necesitan obtener información sobre la cámara que se está usando así como información sobre lo que se está viendo, tanto de profundidad como de color, por lo que los dos paquetes necesitan como mínimo suscribirse a tres *topics* de ROS que les proporcione toda esta información. Además de esta información, ambos paquetes requieren conocer los dos sistemas de referencia comentados anteriormente (el de la cámara y el del robot).

5.2.1. RTAB-Map

RTAB-Map (Real-Time Appearance-Based Mapping) es un *RGB-D SLAM* basado en un detector incremental de bucle cerrado que compara cada lectura con las anteriroes. El detector incremental de bucle cerrado determina si una imagen proviene de una localización ya conocida o una nueva [6].

Además de suscrbirse a los *topics* y de usar los sistemas de referencia, el paquete de RTAB-Map permite usar un laser bidimensional para ayudar al mapeado. También puede coger información sobre la posición y orientación del robot con respecto al inicio de todo el proceso para mejorar la calidad del mapeado. A pesar de ofrecer estas características, se ha optado por no usarlas, para comparar ambos paquetes en las mismas condiciones. De esta forma, la información de posicón y orientación del robot durante el proceso de mapeado es calcualdo en función de la información recibida a través de la cámara.

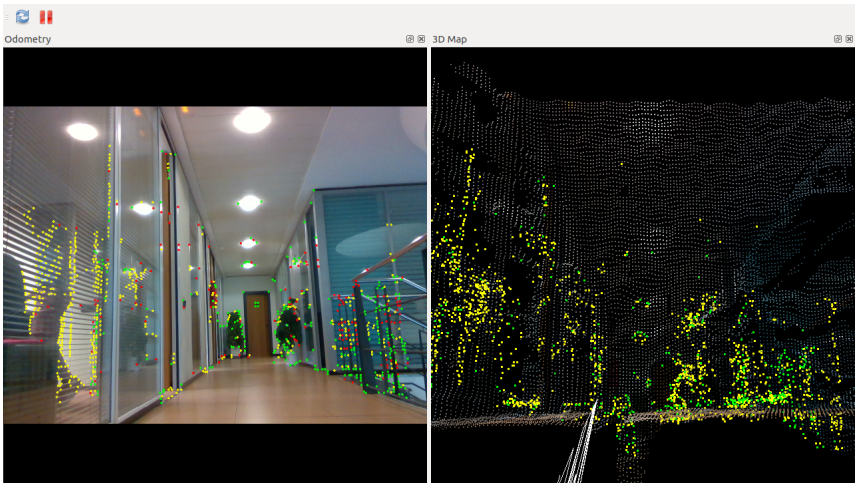

Figura 5.5: Interfaz de RTAB-Map con Intel Realsense D435.

5.2.2. RGBDSLAM

RGBDSLAM es un paquete de SLAM destinado al uso con cámaras con sensor de profundidad. Puede ser usado para la genereación de nubes de puntos y mapas de alta precisión [7].

A diferencia de RTAB-Map, este paquete únicamente puede saber la posición del robot a través de la información recogida a través de la cámara, es decir, no permite suscribirse a un *topic* de ROS que provea esa información.

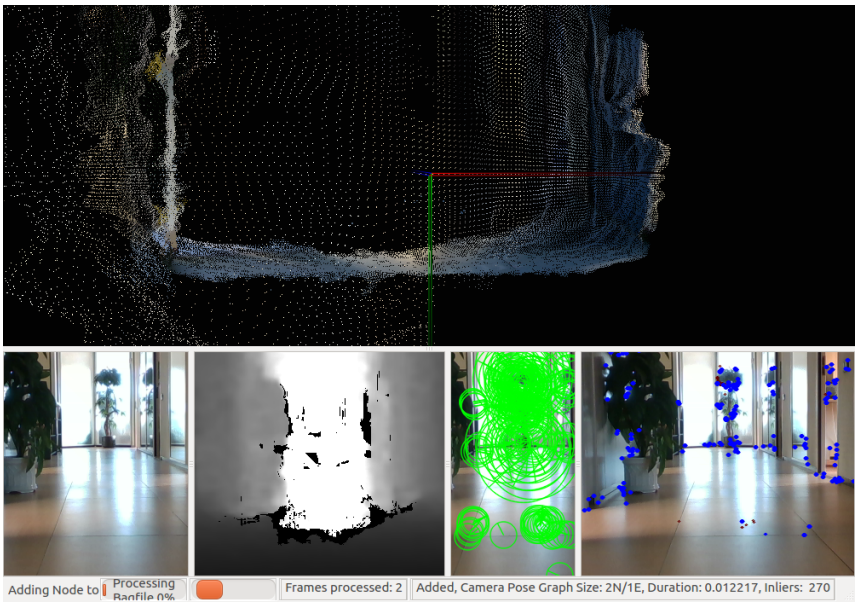

Figura 5.6: Interfaz de RGBDSLAM con Intel Realsense D435.

5.2.3. Comparativa

A pesar de que ambos paquetes cumplen el requisito de poder realizar mapeado en tres dimensiones con la cámara Intel Realsense D435, se ha observado que en las mismas condiciones y con resultados de mapeado muy similares, el consumo de recursos de RGBDSLAM es superior al de RTAB-Map.

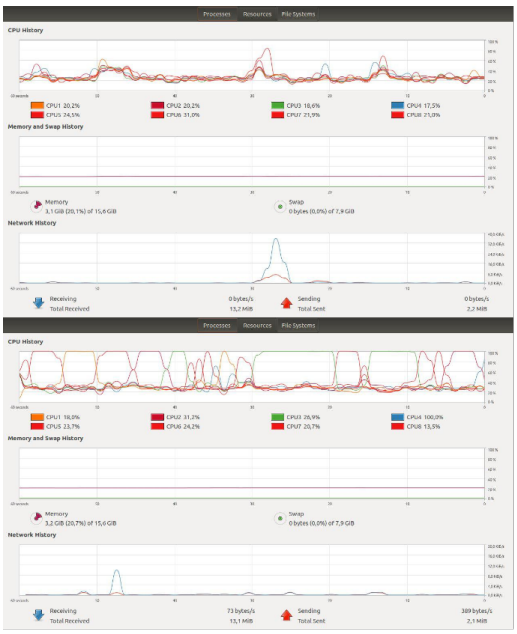

Figura 5.7: Consumo de recursos. Arriba RTAB-Map. Abajo RGBDSLAM.

Por otra parte, RTAB-Map permite usar la lectura a través de láser y el cálculo de la posición del robot a través del software que ya lleva equipado el robot, lo que permite obtener mejores resultados a la hora de mapear.

Por lo que se ha elegido RTAB-Map como paquete para mapear en tres dimensiones para este proyecto por considerar que ofrece mayores prestaciones con un consumo inferior a la otra alternativa.

CAPÍTULO 6
Implantación

Para la puesta en marcha de la aplicación en el robot, es necesaria una instalación y configuración previa de los paquetes de los que depende este proyecto. Por una parte se explicará el proceso de instalación de la página web y por otra parte, la instalación de los paquetes que son requeridos para el mapeado en tres dimensiones. Se detalla la instalación de todos los paquetes para la versión 16.04 del sistema operativo Ubuntu y la versión Kinetic de ROS.

6.1 Django

Django es el *framework* de desarrollo web que se ha elegido para este proyecto. La posibilidad de crear una vista principal a partir de vistas parciales y se implmentación en Python, han sido, entre otras, algunas de las razones por las que se ha elegido este *framework*.

Para instalar Django, es necesario instalar *pip*[1] previamente.

```
1 sudo apt install python-pip
```

Una vez instalado pip, ya se puede instalar Django:

```
1 pip install Django
```

Tras haber instalado Django, ya se puede crear un proyecto con la siguiente línea de comando:

```
django-admin startproject <mysite>
```

Django deberá haber creado un directiorio con el nombre <*mysite*>, con la estructura básica del espacio de trabajo. Para comprobar que el proyecto se ha iniciado correctamente, hay que situarse en la carpeta raíz del proyecto y ejecutar el siguiente comando:

```
python manage.py runserver
```

Lo que debería sacar por pantalla algo parecido a lo que refleja la siguiente figura.

[1] pip - https://pypi.org/project/pip/

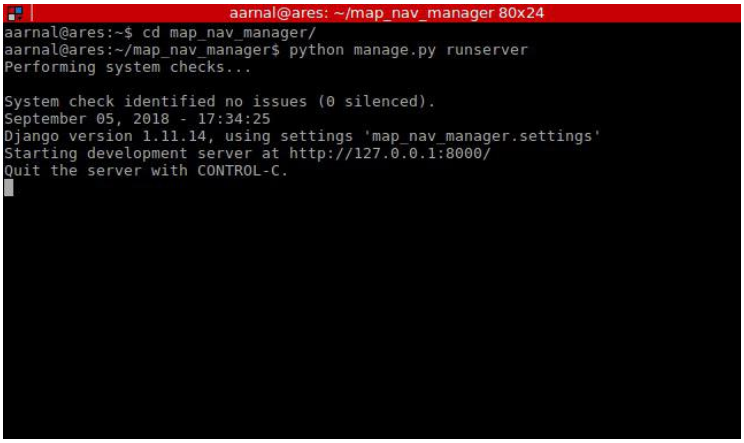

Figura 6.1: Terminal de comandos corriendo el servidor del proyecto.

6.2 Mapeado 3D

Para el proceso de mapeado, en primer lugar se ha instalado en el robot el software necesario para que el dispositivo Intel RealSense D435 pueda mandar datos a través de ROS. Para ello, primero ha sido necesario instalar el *Intel RealSense SDK 2.0* [2]

En primer lugar se registra la clave pública del servidor:

```
sudo apt-key adv --keyserver keys.gnupg.net --recv-key C8B3A55A6F3EFCDE ||
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-key
    C8B3A55A6F3EFCDE
```

A continuación se añade el servidor a la lista de repositorios:

```
sudo add-apt-repository "deb http://realsense-hw-public.s3.amazonaws.com/
    Debian/apt-repo xenial main" -u
```

Previamente a la instalación de los paquetes, es necesario actualizar la lista de paquetes disponibles con los nuevos servidores.

```
sudo apt-get update
```

Una vez actualizada la lista de paquetes, se instalan los paquetes básicos que permiten la ejecución de demos y herramientas de *librealsense*.

```
sudo apt-get install librealsense2-dkms librealsense2-utils
```

Por último, es preciso instalar los paquetes de desarrolladores para permitir la compilación de aplicaciones que usan *librealsense* a través de ROS

```
sudo apt-get install librealsense2-dev librealsense2-dbg
```

Una vez instalado el *Intel RealSense SDK 2.0*, hay que instalar el paquete de *realsense2_camera*[3]. Debido a que la cámara Intel RealSense D435 ha sido anteriormente utilizada en Robotnik para otros proyectos, se ha optado por instalar la versión modificada existente en el repositorio de la empresa. En primer lugar, copiamos el repositorio en el espacio de trabajo:

```
git clone https://github.com/RobotnikAutomation/realsense
```

[2] Intel RealSense SDK 2.0 - https://github.com/IntelRealSense/librealsense
[3] Realsense ROS - http://wiki.ros.org/realsense2_camera

Una vez copiado en el espacio de trabajo, se ha de compilar el paquete:

```
catkin_make
```

Por último, se debe instalar el paquete que se acaba de compilar:

```
catkin_make install
```

En este punto, el sistema tiene todo lo necesario para trabajar con la Intel RealSense D435. Para comprobar su funcionamiento se puede ejecutar el siguiente comando, que lanza el nodo ROS que se comunica con la cámara:

```
roslaunch realsense2_camera rs_camera.launch
```

Este nodo, sacará por pantalla la configuración que va aplicando a medida que se ejecuta. Para verificar el funcionamiento de la cámara, se puede comprobar a través de RVIZ, visualizando uno de los *topics* por los que publica el nodo de la cámara o visualizando uno de estos tópics a través del *plugin* específico de Rqt para la visualización de imagen a través de ROS.

6.2.1. RTAB-Map

Se ha optado por una instalación desde las fuentes de los paquetes de RTAB-Map. En primer lugar se ha instalado el paquete de *rtabmap* y *rtabmap_ros* desde los repositorios de ROS para asegurarse que todas las dependencias necesarias se instalan correctamente en el sistema. Una vez instalados los paquetes, estos han sido desinstalados para su posterior reinstalación, esta vez desde las fuentes. Aunque una vez instalados los paquetes a través de los repositorios de la distribución de ROS, todos los nodos relacionados a estos paquetes son accesibles y pueden ejecutarse fácilmente, se ha optado por una instalación desde las fuentes para tener acceso a todos los archivos relacionados con el paquete así como a las configuraciones por defecto de los nodos de *rtabmap*.

En primer lugar, se clona el repositorio de *rtabmap*:

```
git clone https://github.com/introlab/rtabmap.git rtabmap
```

Después de clonar el repositorio, hay que compilar e instalar el paquete:

```
cd rtabmap/build
cmake ..
make
sudo make install
```

Una vez instalado rtabmap, se ha porcedido a la instalación del paquete de ROS de *rtabmap*. En primer lugar, se clona el repositorio en la carpeta src del espacio de trabajo de ROS:

```
git clone https://github.com/introlab/rtabmap_ros.git src/rtabmap_ros
```

Tras haberlo clonado, es necesario compilar el paquete:

```
catkin_make
```

Una vez instalado, se ha realizado una prueba para comprobar que el paquete se ha instalado correctamente. Para ello basta con lanzar cualquiera de los archivos *.launch* de los que dispone el paquete. Acto seguido, aparece una interfaz como la que se muestra en la siguiente figura.

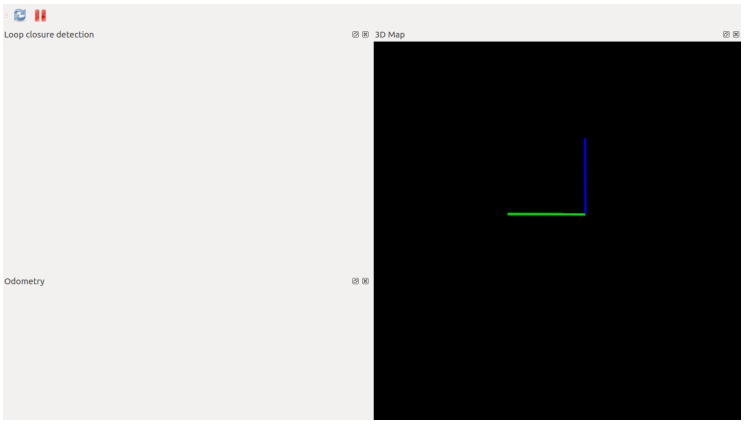

Figura 6.2: Interfaz de RTAB-Map.

6.2.2. RGBDSLAM

Para usar RGBDSLAM es necesario instalar el paquete específico de ROS además de *g2o*[4], un *framework* para optimizar funciones de error no lineales basadas en gráficos del que depende RGBDSLAM.

En primer lugar se tienen que instalar las dependencias de g2o:

```
sudo apt-get install libsuitesparse-dev libeigen3-dev
```

Una vez se tienen las dependencias instaladas, se descarga la versión recomendad del paquete g2o:

```
G2O_REPO_DIR=g2ofork
git clone -b c++03 https://github.com/felixendres/g2o.git $G2O_REPO_DIR
```

Tras descargarlo, hay que compilarlo e instalarlo:

```
mkdir $G2O_REPO_DIR/build
cd $G2O_REPO_DIR/build
cmake .. -DCMAKE_INSTALL_PREFIX=$G2O_REPO_DIR/install -DG2O_BUILD_EXAMPLES=OFF
nice make -j8 install
```

Después de haber instalado g20, se debe descargar el paquete de rgbdslam:

```
WORKSPACE=$SUBDIR/rgbdslam_catkin_ws
mkdir -p $WORKSPACE/src
git clone -b kinetic https://github.com/felixendres/rgbdslam_v2.git $WORKSPACE/
    src/rgbdslam
```

Una vez clonado el paquete, se tiene que compilar. Para ello, primero es necesario exportar la carpeta donde se instaló g2o para que pueda ser localizable durante la compilación:

```
export G2O_DIR=$G2O_REPO_DIR/install
catkin_make -C $WORKSPACE -j2
```

De igual forma que con RTAB-Map, basta con lanzar uno de los archivos *.launch* de ejemplo para comprobar que funciona correctamente.

[4] g2o - https://github.com/RainerKuemmerle/g2o

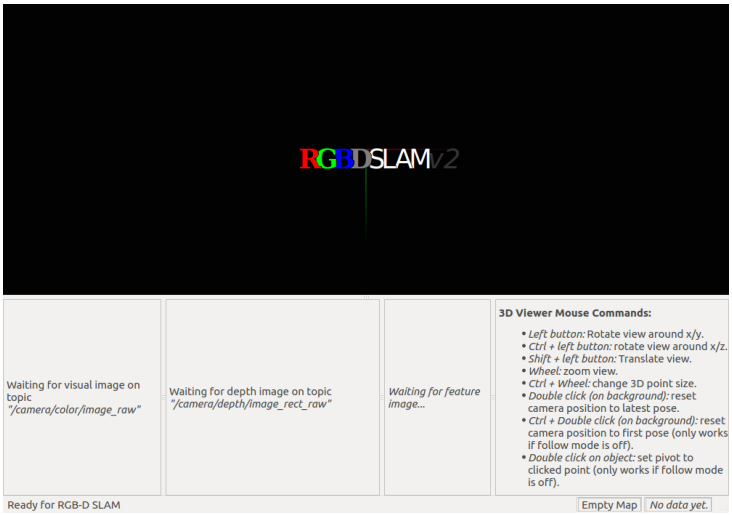

Figura 6.3: Interfaz de RGBDSLAM.

CAPÍTULO 7
Pruebas

En el presente capítulo se explican las pruebas que se han realizado de la aplicación web con usuarios. Al tratarse de una funcionalidad que forma parte de un producto final que todavía está en desarrollo, la empresa ha decidido no hacer pruebas de usabilidad de la interfaz web con el usuario final. Sin embargo, con la finalidad de tener información sobre la interfaz web útil para posibles mejoras de la misma, se ha optado por realizar pruebas con trabajadores de Robotnik ajenos al departamento de informática.

Se ha escogido a diez trabajadores de la empresa y se han establecido tres casos de uso en los que sin ningún tipo de información ni manual, los usuarios han de cumplir los requisitos establecidos.

El primer caso es iniciar el mapeado en tres dimensiones.

1. Dirigirse al panel de mapeado en 3D.
2. Apretar el boton *Play*.

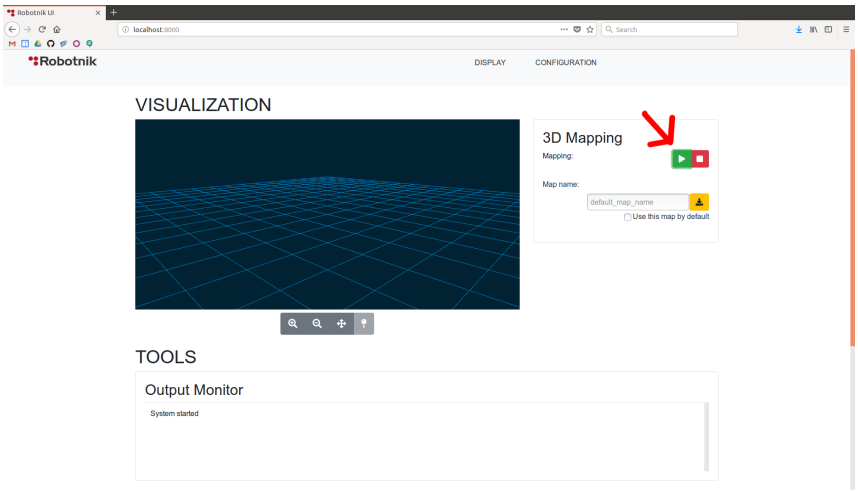

Figura 7.1: Interfaz de ususario. Botón para empezar mapeado.

El total de los usuarios supieron como empezar a mapear desde la configuración inicial de la interfaz de usuario.

El segundo caso es configurar la aplicación para que únicamente se visualice el estado de los motores y de la cámara.

1. Ir a la pestaña de *Configuración*.

2. Desmarcar todos los *switches* de visualización de módulos, excepto el del estado del motor y el de la cámara.

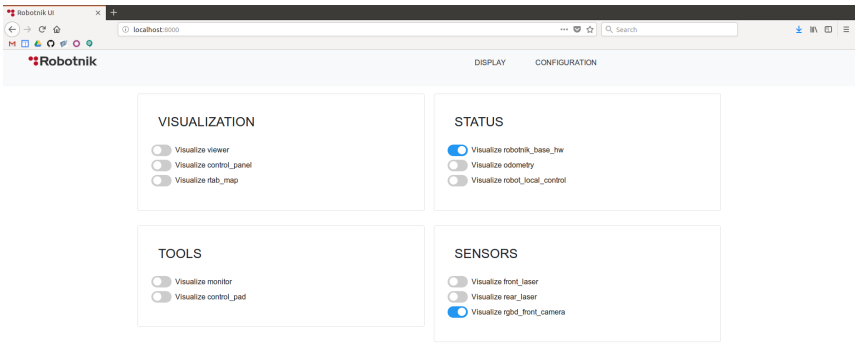

Figura 7.2: Interfaz de usuario. Pestaña de configuración.

3. Volver a la pestaña de *Display* para ver el resultado.

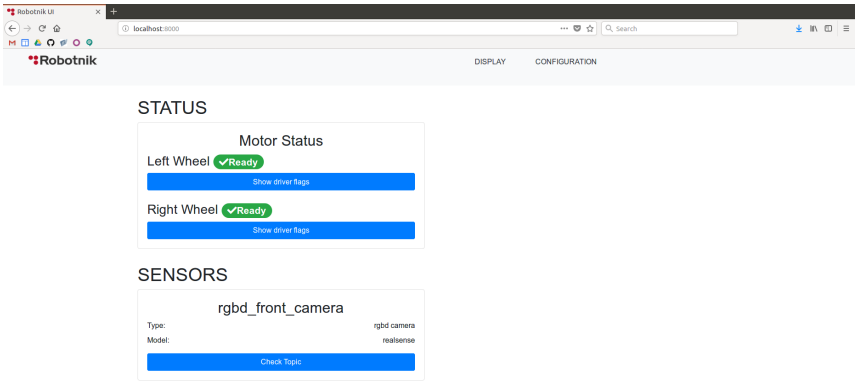

Figura 7.3: Interfaz de usuario. Configuración con módulos de motores y cámara.

El ochenta por ciento de los usuarios supieron configurar la página web para los requisitos presentados. Los dos ususarios restantes, no supieron cuál era la opción que había que dejar marcada para visualizar la tarjeta correspondiente al estado de los motores.

Cargar un mapa 2D partiendo de una configuración de la página en la que no se visualiza por defecto el panel que permite ejecutar dicha funcionalidad

1. Ir a la pestaña de *Configuración*.

2. Marcar el módulo de mapeado en 3D.

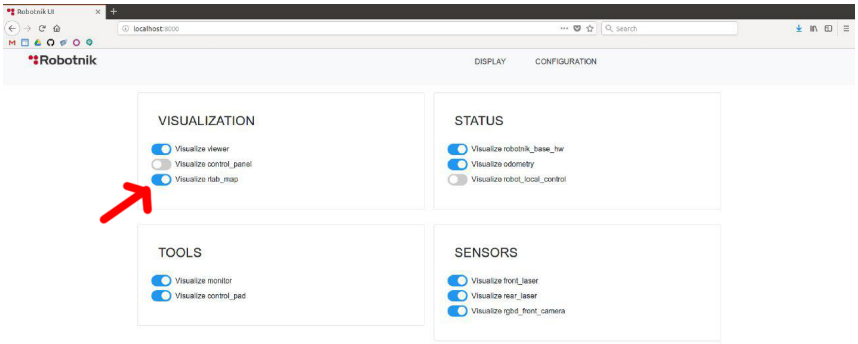

Figura 7.4: Interfaz de usuario. Pestaña de configuración, mostrar mapeado 3D

3. Ir a la pestaña de inicio.
4. Apretar el botón *Play* del mapeado 3D.
5. Apretar el botón de guardar cuando se haya acabado de mapear.

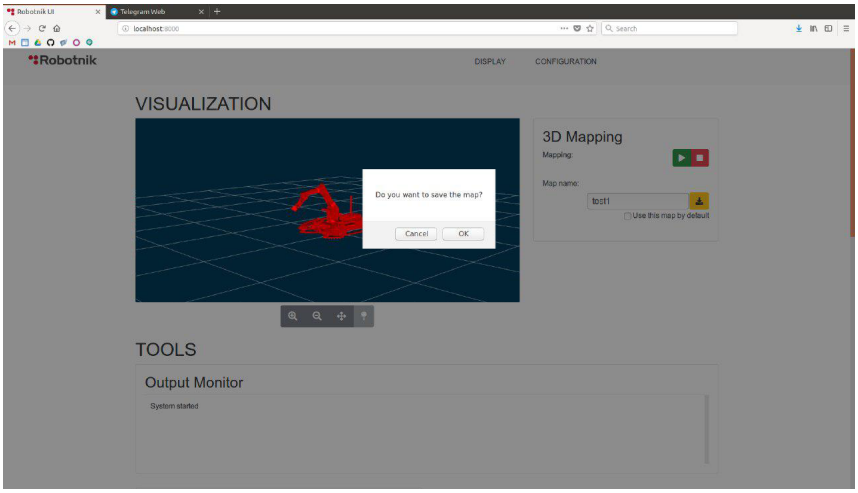

Figura 7.5: Interfaz de usuario. Botón para guardar mapa.

Siete de los ususarios logararon realizar la prueba con éxito. Tres de los usuarios no supieron cuál era la opción correspondiente a hacer visible la tarjeta de mapeado 3D porque el nombre asignado no era lo suficientemente claro.

En vistas a los resultados obtenidos, no se plantea el rediseño de la página y la versión actual de la aplicación es aceptada hasta una nueva etapa de pruebas con el usuario real de la aplicación. Habiendo detectado algunos aspectos de la interfaz que resultan poco intuitivos para algunos usuarios, se ha desarrollado un breve manual de usuario, que permita el fácil uso de la aplicación. Dicho manual está incluido como apéndice de este trabajo.

CAPÍTULO 8
Conclusiones

Se ha desarrollado una página web que cumple los requisitos específicos de la plataforma móvil Rising y que además es compatible con cualquier otro robot que actualmente se encuentre en producción por la empresa Robotnik.

Se ha conseguido una estructura estándar para la representación de la información dentro de la página web. De esta forma, a través del uso del archivo de configuración en el que se especifican los componentes *hardware* y los paquetes *software* de los que dispone el robot, resulta inmediata la adapatación de la página a cualquier robot, independientemente del número de componentes del que disponga. Por lo que se considera, que el desarrollo de este proyecto, no solo cumple con las necesidades de la plataforma Rising, si no que además aporta una valor añadido al resto de productos de la empresa.

A parte del rediseño de la página, se han implementado algunas funcionalidades que no existían en la aplicación de origen como son: la posibilidad de mapear en tres dimensiones, visualizar información de los sensores con los que va equipado el robot y comprobar si están publicando información a través del *topic* de ROS, configurar la vista de la página según las necesidades y poder grabar durante un periodo de tiempo toda la información que es publicada por los *topics* del robot.

El desarrollo de este proyecto ha permitido que el alumno entienda la metodología de trabajo así como la arquitectura a seguir a la hora de desarrollar una aplicación web dependiente de un sistema ya existente.

8.1 Relación del trabajo desarrollado con los estudios cursados

A pesar de haber utilizado tecnologías que no se han presentado en el Grado de Ingeniería Informática, los conocimientos adquiridos en el mismo han ayudado al alumno al desarrollo de este proyecto y a la comprensión de la arquitectura de estas tecnologías.

Entre las asignaturas a destacar que han ayudado al alumno a la comprensión de la arquitectura de las tecnologías utilizadas se encuentra: Tecnología de sistemas de información en la red (TSR), donde se estudiaron los patrones de comunicación (como pub/-sub y req/rep) de los que hacen uso los *topics* de ROS; por otra parte se ha hecho uso de Django, que sigue una arquitectura modelo-vista-controlador, que guarda estrecha relación con los conocimientos estudiados en Ingeniería del software (ISW). Por otra parte, el aprendizaje de Python y JavaScript durante el grado, ha facilitado el desarrollo de algunas funcionalidades de la aplicación.

CAPÍTULO 9
Trabajos futuros

En el presente capítulo se presentan algunos objetivos o posibles mejoras del trabajo para un futuro.

Se observó que la página web podría ser utilizada por un usuario de menor autoridad que el cliente que compra el producto, por lo que se considera como una posible mejora para la aplicación web el desarrollo de un sistema de autenticación. Lo que conllevaría:

- Implantación de la gestión de usuarios.
- Aplicar un sistema para gestionar de forma segura las claves de usuarios.
- Establecer el sistema jerárquico no solo para acceso a vistas completas de la página web, si no también para vistas parciales ya existentes.

Otra posible mejora sería la reducción de consumo de recursos que tiene la aplicación. Para ello habría que realizar un estudio del consumo de recursos que tiene la página, tanto para el cliente como para el servidor. En base a este estudio, actuar en busca de la optimización del consumo para permitir que la experiencia a través de la aplicación web sea lo más fluida posible en cualquier caso.

Finalmente, también se plantea la redacción de un breve manual para el desarrollador que explique cómo está estructurada la aplicación e indique los pasos a seguir en caso de querer ampliarla con nuevas funcionalidades o modificar la configuración de la misma. Enseñando al ususario desarrollador que en caso de no tener que desarrollar ninguna funcionalidad nueva, para adapatar la web a un robot nuevo, bastaría con modificar el archivo de configuración. Aunque esta última posible mejora no se trata de una optimización o ampliación del software desarrollado, puede ser de gran utilidad para desarrollos futuros que partan de esta aplicación web.

Bibliografía

[1] Norma ISO 8373:2012 Consultado en https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en.

[2] Robótica de servicio segun la Federación Internacional de Robótica. Consultado en https://ifr.org/service-robots.

[3] Jeanine Katzel Information systems: The evolution of the HMI *Control Engineering*, 2012.

[4] Robótica tradicional y robótica colaborativa, diferencias. Consultado en http://cfzcobots.com/robotica-tradicional-y-robotica-colaborativa-diferencias

[5] Jihoon Lee. Web Applications for Robots using rosbridge Consultado en http://cs.brown.edu/research/pubs/theses/masters/2012/lee.pdf

[6] RTAB-Map Consultado en http://introlab.github.io/rtabmap

[7] RTABSLAM Consultado en https://felixendres.github.io/rgbdslam_v2

APÉNDICE A
Manual de usuario

El paquete de map_nav_manager consta de una aplicación web que permite al usuario conocer en tiempo real información sobre el estado del robot y interacutar con él. La página está dividida en dos vistas:

A.1 Display

Esta es la vista principal de la aplicación, en esta es donde se puede interactuar con el robot y obtener información del mismo. Esta vista está dividida en cuatro subsecciones, que marcan los tipos de las diferentes funcionalidades.

A.1.1. Visualization

En esta sección se muestra todo aquello relacionado con el mapeado, la navegación y la localización,

A.1.2. Tools

En este apartado, se encuentran algunas herramientas útiles que permiten la interacción con el robot:

- **Monitor**: saca información sobre los procesos que está realizando el robot.

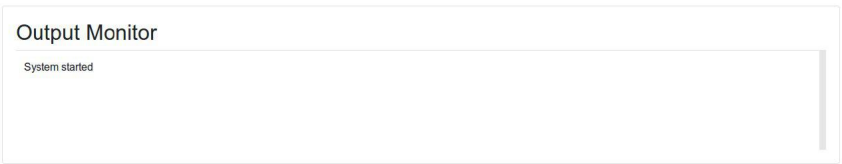

Figura A.1: Interfaz de usuario. Monitor de procesos

- **Robot Move Control**: Permite teleopetar al robot desde la página web.

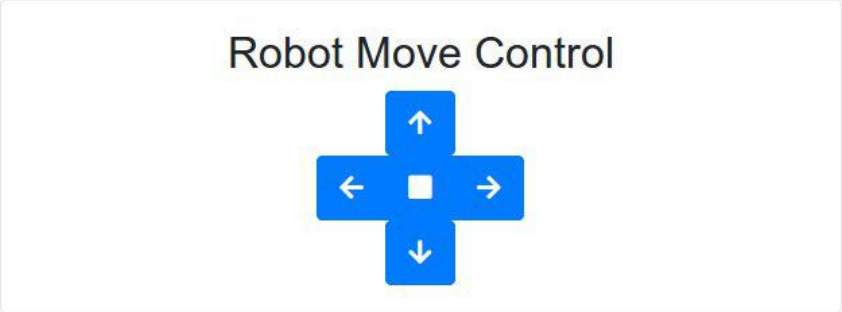

Figura A.2: Interfaz de usuario. Herramienta de teleopetación

A.1.3. Status

En este apratado se puede ver el estado del robot:

- **Motor Status**: Muestra el estado de los motores. Puede tener una etiqueta roja o una verde asociada en función de si los motores están funcionando correctamente. Apretando el botón asociado a cada motor, se pueden ver información sobre los *flags* que proporcionan los *drivers*.

Figura A.3: Interfaz de usuario. Panel de motores

- **Odometry**: Permite conocer la posición y orientación del robot con respecto al punto en el que se puso en marcha. Apretando el botón se puede reiniciar el sistema de coordenadas.

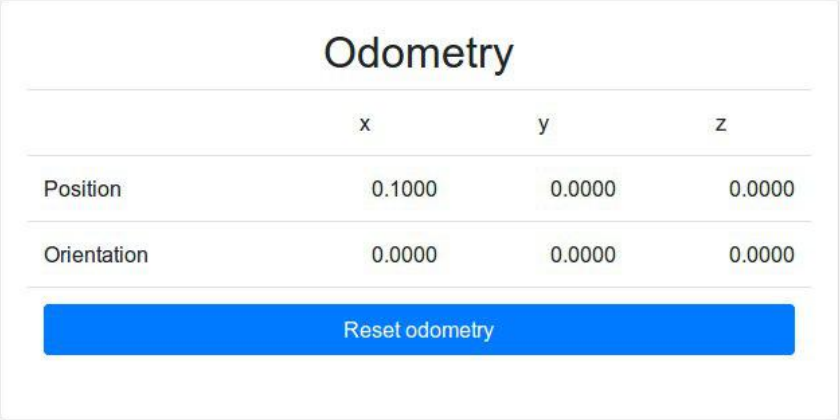

Figura A.4: Interfaz de usuario. Tabla de odometría

A.1.4. Sensors

En este apartado se puede ver el estado e información de los sensores con los que va equipado el robot. Permite conocer los tipos de sensores y el modelo de los mismos. Cada sensor lleva asociado un botón que permite hacer una consulta que muestra si el sensor está mandado información a ROS o no.

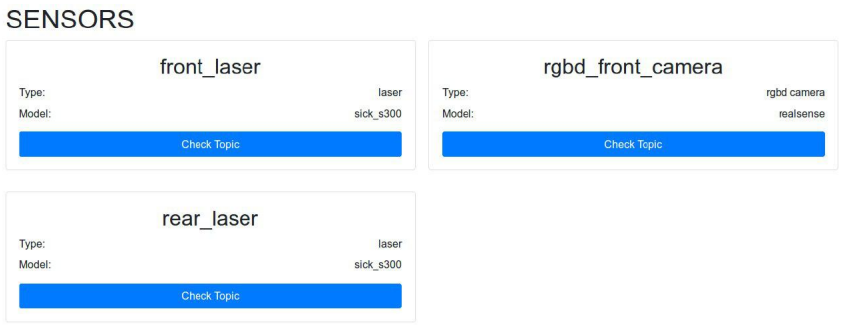

Figura A.5: Interfaz de usuario. Sección de sensores

A.2 Configuración

En esta vista el usuario puede ver todas las funcionalidades que tiene disponible la página web. Las funcionalidades están están separadas según el tipo en el que están clasificadas. Para dejar de visualizar alguna de las funcionalidades, simplemente hay que dirigirse al *switch* asociado a dicha funcionalidad en la pestaña de configuración. Descmar el switch provocará que esta funcionalidad ya no sea visible en la página web hasta que no se vuelva a habilitar de nuevo.

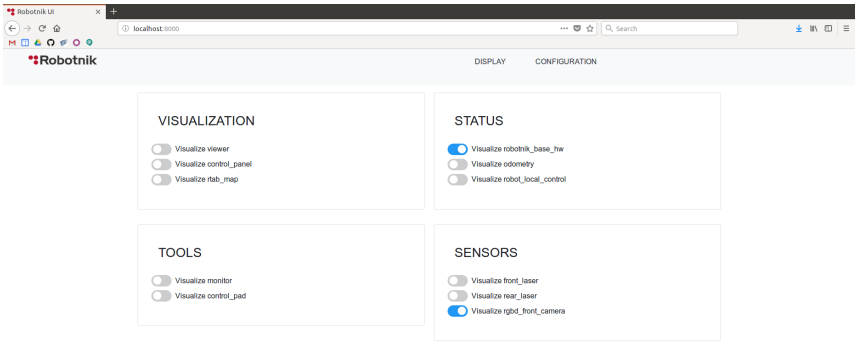

Figura A.6: Interfaz de usuario. Tabla de odometría

Para la configuración mostrada en la figura previa, únciamente se verá el panel que marca el estado de los motores y el panel asociado a la cámara.

Si se desmarcan todas las funcionalidades de un tipo, la sección asociada a ese tipo desaparecerá completamente de la página web.