UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Diseño e implementación de una interfaz gráfica de usuario para mapeado de entornos y navegación en ROS

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jose Manuel Rapado García

Tutor: Miguel Ángel Mateo Pla

Co-Tutor: Román Navarro García

Curso 2015-2016

Resumen

En el presente trabajo se diseña e implementa una interfaz gráfica web para que usuarios sin experiencia en ROS puedan utilizar los algoritmos de mapeado y navegación autónoma y tener una visualización del mapa y del robot al mismo tiempo. Previamente, se introducen algunos conceptos fundamentales de Robot Operating System (ROS) y revisan trabajos anteriores basados en ROS sobre la misma materia ROS. La interfaz web se ha desarrollado usando tecnologías web (e.g. HTML, CSS, Javascript) debido a que con ellas se pueden obtener herramientas fáciles de usar. También se han usado herramientas y paquetes de las bibliotecas de ROS. Posteriormente se ha hecho una revisión del trabajo realizado y se han sugerido futuras modificaciones.

Palabras clave: robótica, robot móvil, ROS, navegación autonoma, mapeado de entorno, websockets, rosbridge, web, aplicación grafica de usuario, slam-gmapping, amcl, move-base

Abstract

In this project, a graphical user web-interface that allows non-experienced users to use the ROS mapping and autonomous navigation algorithms and to have a map and a robot visualized is designed and implemented. Previously, some main concepts about ROS were introduced and the previous work in this area using ROS were reviewed.The web interface has been developed using web technologies (e.g. HTML, CSS, Javascript) due to they permit to provide easy working tools. It have also been used tools and packages from ROS libraries. Subsequently, we made a final work review and suggested future approaches.

Key words: robotics, mobile robot, ROS, autonomous navigation, mapping, websockets, rosbridge, web, graphical user interface, slam-gmapping, amcl, move-base

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

El presente Trabajo Fin de Grado (TFG) se enmarca dentro de la robótica de servicios que es la rama de la robótica que se vale de robots móviles para realizar servicios útiles para el bienestar de los humanos y del equipamiento, excluyendo operaciones de manufactura. Según la "International Federation of Robotics" (IFR), un robot móvil se define como: "Un mecanismo actuado programable en dos o más ejes con un grado de autonomía, moviéndose en su entorno para realizar tareas planificadas. Autonomía en este contexto se entiende como la habilidad de realizar tareas planificadas basándose en el estado actual o en sensorización, sin intervención humana."

Una de las aplicaciones de los robots de servicio es la navegación autónoma en entornos controlados (navegación indoor). Esta navegación se caracteriza por la utilización de un mapa del entorno que el robot móvil utiliza para, con la ayuda de los algoritmos de navegación adecuados, desplazarse de un lugar a otro del espacio conociendo de antemano donde están los obstáculos. La información del mapa se puede actualizar con sensores del propio robot (e.g. láseres, cámaras,...), sensores que también se utilizan para ajustar la información de odometría del propio robot.

Los principales algoritmos de navegación y localización que se utilizan son, por un lado, el algoritmo "Simultaneous Localization and Mapping" (SLAM), el cual se basa en métodos estadísticos de muestreo para generar un mapa y localizarse en él, y por otro, el algoritmo "Analysis Monte Carlo Localization" (AMCL), basado en el análisis de Monte Carlo, que sirve para localizar el robot dentro de un mapa previamente generado.

Actualmente varios middleware de robótica integran todos estos algoritmos. Estos middleware se encargan de ofrecer abstracción de hardware, controladores para dispositivos y mecanismos de comunicación. Uno de los middleware más utilizados por la comunidad de robótica es Robot Operating System (ROS). ROS cuenta con un amplio catálogo de herramientas y de interfaces gráficas de usuario que permiten visualizar a la información del robot. Así mismo, nos permite extender la visualización de datos del robot en la web a través de varias bibliotecas e interfaces.

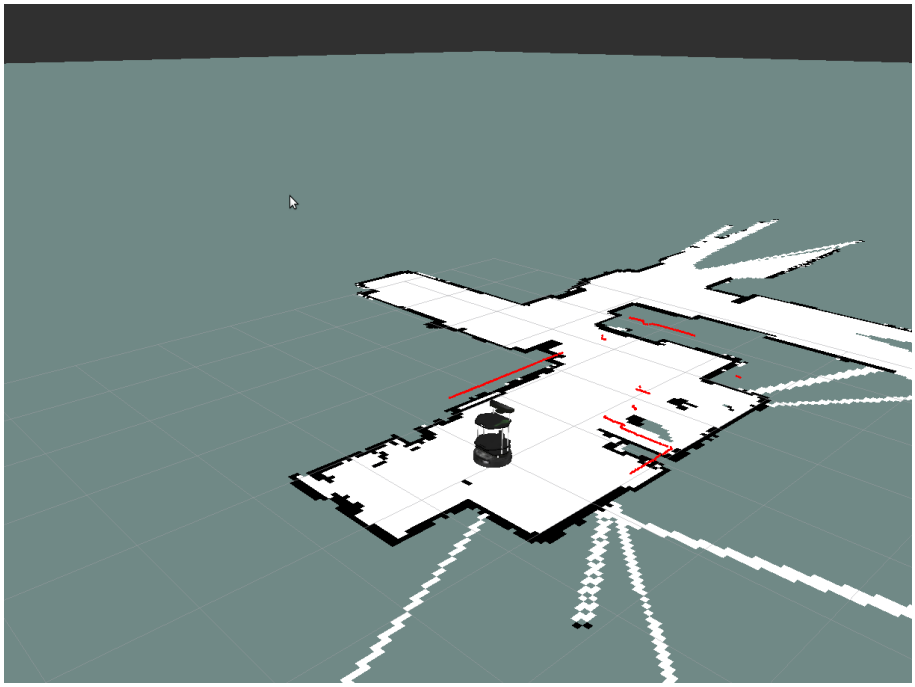

Figura 1.1: Robot generando un mapa con un láser 2D usando SLAM-gmapping

A pesar de contar con herramientas para visualizar la información del robot y con algoritmos de navegación eficientes, estos no son fáciles de aprender y su integración en la aplicación requiere de unos conocimientos de robótica y programación de bajo y alto nivel que sólo disponen los usuarios de robótica expertos.

Por otra parte, las interfaces web ofrecen a los usuarios un acceso universal e independiente a entornos interactivos, amigables y fáciles de usar. De esta manera usuarios no expertos pueden acceder y utilizar herramientas que ya están preconfiguradas y que funcionan con solo realizar un simple proceso.

En vista de lo anteriormente expuesto, en este proyecto se presenta el diseño e implementación de una interfaz gráfica de usuario web, mediante la cual el usuario de robótica no experto puede crear un mapa del entorno del robot y después realizar navegación en base a dicho mapa utilizando las bibliotecas de navegación de ROS y el estándar web.

Los principales objetivos de este TFG son:

- Dotar a la comunidad ROS de una interfaz de usuario web para la creación y navegación de mapas.
- Obtener una interfaz de usuario de fácil uso que permita visualizar el robot y el mapa durante el proceso de generación y navegación de mapas.
- Simplificar el proceso de creación de mapas para un usuario no experto.
- Simplificar el proceso de navegación autónoma basado en un mapa para un usuario no experto.
- Acceder a este tipo de herramientas desde cualquier navegador web.

En los siguientes capítulos se va a documentar el trabajo realizado en el proyecto. En el capítulo 2 se describe el marco teórico del proyecto. En el capítulo 3 se aborda la descripción de la solución adoptada. En primer lugar se realizar el análisis de los requisitos de la solución. En segundo lugar se realiza la descripción de su diseño. Después se aborda la implementación de la solución. Por último se realiza un ejemplo de funcionamiento de la aplicación a través de distintas capturas de pantalla. Finalmente en el capítulo 4 se realizará una revisión final del trabajo y se pondrán de manifiesto las conclusiones obtenidas.

CAPÍTULO 2

Marco teórico

2.1 Robot Operating System (ROS)

Gran cantidad de sistemas *middleware* [1] han propuesto la posibilidad de compartir bibliotecas entre los desarrolladores de software para robótica. Entre ellos podemos citar Player/Stage, Carnegie Melon Navigation Toolkig (CARMEN), Microsoft Robotics Studio, YARP, Lightweight Communication and Marshalling (LCM), y Robot Operating System (ROS)[2]. Estos sistemas ofrecen interfaces comunes que permiten reutilización de código y bibliotecas compartidas. Entre estos sistemas ROS se ha convertido en un estándar de facto como *middleware* capaz de ofrecer sistemas de control robótico a través de una red de área local.

ROS compone un marco de trabajo muy flexible para desarrollar software en robótica. Está formado por una colección de herramientas, bibliotecas y convenciones con el objetivo de simplificar al máximo la difícil tarea de programar comportamientos robóticos complejos y robustos dentro de un amplio catálogo de plataformas robóticas [21]. A pesar de ser referido como un sistema operativo, este no lo es como tal (es más bien un meta sistema operativo), debido a que no ofrece ni gestión ni planificación de procesos tal y como lo haría un sistema operativo al uso [4]. ROS permite la abstracción del hardware, ofrece controladores para dispositivos, acceso a útiles bibliotecas y potentes herramientas.

ROS surge a partir de los trabajos de investigación realizados en la universidad de Stanford. Estos trabajos tenían el objetivo de crear sistemas dinámicos de software que involucraban inteligencia artificial (IA) como el proyecto *Stanford Artificial Inteligence Robot System* (STAIRS) o el proyecto *Personal Robot* (PR). Fue en 2007, cuando en Willow Garage (grupo de visionarios y talentos de la robótica), empezó a proveer de importantes recursos para extender todos esos conceptos y desarrollar software para ello. Su esfuerzo inicial pronto se vió respaldado e impulsado por un gran número de investigadores que contribuyeron con su tiempo y experiencia a desarrollar las ideas principales de ROS y los paquetes de software de su núcleo. [20] El software fue desarrollado en código abierto utilizando la licencia BSD, y poco a poco se ha convertido en una plataforma ampliamente

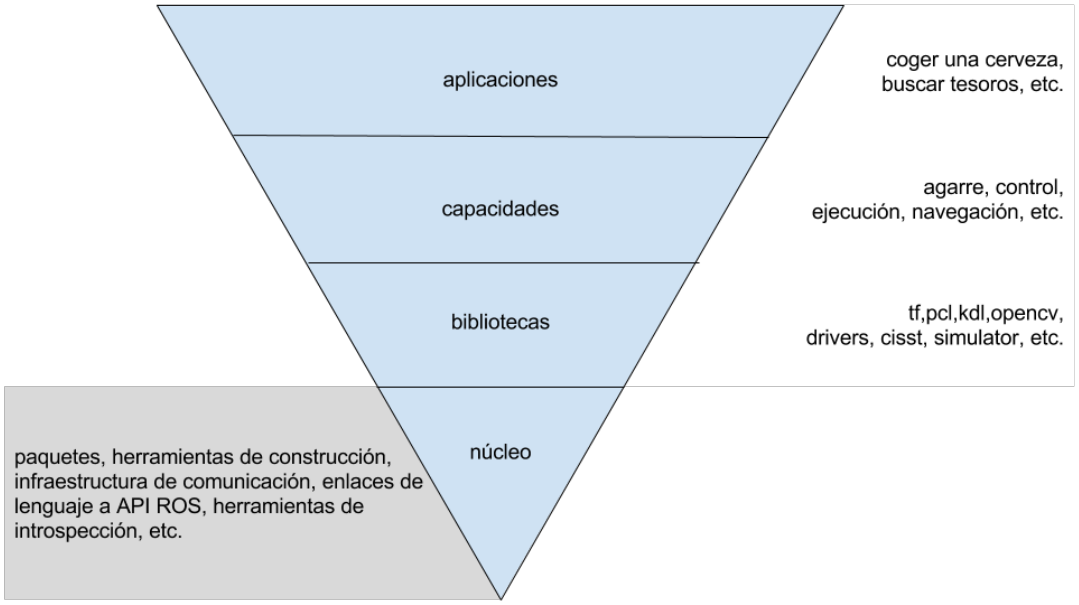

Figura 2.1: Capas de abstracción de ROS

utilizada en la comunidad de investigación robótica [22]. ROS fue diseñado para resolver un conjunto de problemas encontrados al desarrollar grandes proyectos de robótica de servicios. Sin embargo la arquitectura de ROS va más allá del propósito de la robótica de servicios o la manipulación de objetos del entorno con robots. ROS permite al usuario final de una plataforma robótica utilizarla para cualquier fin, centrándose exclusivamente en desarrollar la parte del problema al que quiere dar solución.

2.1.1. Características de ROS

Las características más destacables de este *middleware* de robótica son:

Peer to Peer

ROS ofrece una arquitectura distribuida descentralizada, en el que diferentes procesos locales o remotos ubicados en la misma red de área local, se comunican entre sí utilizando una topología punto a punto. Para conseguir descentralización ROS combina la topología punto a punto con tecnología de *buffers* y un patrón de mensaje en difusión, por la que cada mensaje se envía a un punto o se difunde a todos los puntos. De esta manera se evita la necesidad de un servidor central para intercomunicar todos los procesos.

Multilenguaje

ROS es independiente del lenguaje de programación, pero da soporte nativo en lenguajes como C++, Python, Octave/Matlab, LISP y a otros lenguajes como JAVA. La especificación de ROS está hecha a nivel de la capa de mensajería. La

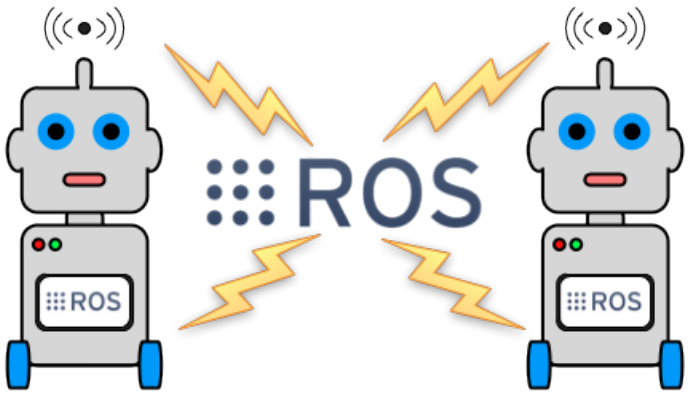

Figura 2.2: Diferentes sistemas robóticos conectados entre sí gracias a ROS

negociación y configuración punto a punto utiliza el protocolo de llamadas de procedimiento remoto (XML-RPC), de manera que las llamadas las codifica utilizando tecnología XML y HTTP como mecanismo de transporte. En la mayoría los lenguajes de programación existen bibliotecas que implementan este mecanismo. Por ello, las implementaciones de ROS se hacen en nativo para cada lenguaje con el objetivo de dar un mejor uso a las convenciones de programación propias de cada uno de ellos. Otras veces la implementación nativa no es posible por lo que se utiliza envoltorios para dar soporte a ese nuevo lenguaje.

Para soportar desarrollo con entre diferentes lenguajes, ROS utiliza un estándar de definición de interfaces (IDL). De ese modo describe los mensajes enviados entre varios procesos del sistema. IDL utiliza ficheros de texto cortos donde se definen los campos de cada mensaje, y permite componer mensajes a partir de otros mensajes ya definidos.

Además contiene generadores de código para cada lenguaje soportado que generan implementaciones en lenguaje nativo de los mensajes. Cada mensaje es una estructura de datos, y son automáticamente serializados y deserializados por ROS conforme se envían y se reciben. Debido a esto se ahorra tiempo y errores. Además es muy fácil definir nuevos mensajes.

Basado en herramientas

En un esfuerzo claro por minimizar la complejidad de ROS, se ha seguido un diseño con micronúcleo, en vez de desarrollar un entorno de desarrollo monolítico de ejecución. Esto significa que el núcleo de ROS implementa la mínima funcionalidad para poder lanzar procesos y manejar la comunicación entre procesos, simulando a los sistemas operativos de los dispositivos móviles. Complementando a esta característica, alrededor del núcleo existe un gran número de pequeñas herramientas que posibilitan la compilación y ejecución de los diversos componentes de ROS. Estas herramientas realizan diferentes funcionalidades: navegación de ficheros, obtención y modificación de parámetros de configuración, etc.

2.1.2. Sistema de archivos ROS

ROS no es un sistema operativo, pero como tal tiene su sistema de archivos organizado de una determinada manera. Se puede distinguir entre:

- **Paquetes:** Son la mínima unidad de software de ROS. Contienen los procesos de ejecución de ROS (nodos), bibliotecas, archivos de configuración, y mucho más, que se organizan como una unidad independiente. Los paquetes son un elemento constructivo atómico y los elementos de lanzamiento de software ROS.
- **Archivos package.xml:** Estos archivos contienen información sobre autor, licencia, dependencias, *flags* de compilación, etc, de un paquete. Cuando está dentro de un meta-paquete debe contener la lista de paquetes como dependencias y tener una etiqueta de exportación.
- **Meta-paquetes:** Este termino agrupa un grupo de paquetes que tienen un propósito único o especial.
- **Archivos de Mensajes (.msg):** Los archivos de mensaje de ROS definen la estructura de los mensajes que intercambian los nodos entre sí. Cuando se define un mensaje propio para un paquete se guarda en el directorio msg del propio paquete.
- **Archivos de Servicio (.srv):** Los archivos de servicio de ROS definen la estructura de los mensajes de servicio que van a intercambiar los nodos. Cuando se define un mensaje propio para un paquete, se guarda en el directorio srv del propio paquete.
- **Repositorios:** La mayoría de los paquetes se mantienen usando un sistema de control de versiones (SCV) como git, subversion (svn), mercurial (hg). El conjunto de paquetes que comparten el mismo SCV se llama repositorio.

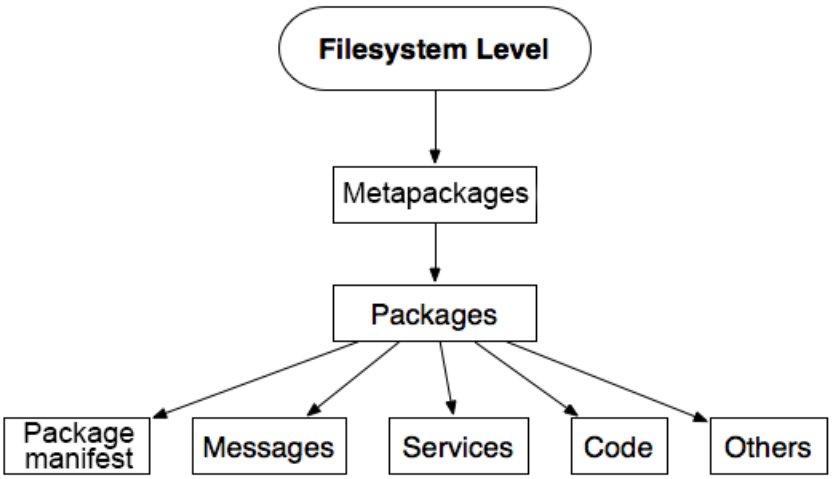

Figura 2.3: Sistema de archivos de ROS

2.1.3. Grafo de computación de ROS

El grafo de computación de ROS está formado por diversas entidades que se relacionan entre sí para hacer funcionar el sistema. A continuación se realiza una breve explicación de las mismas.

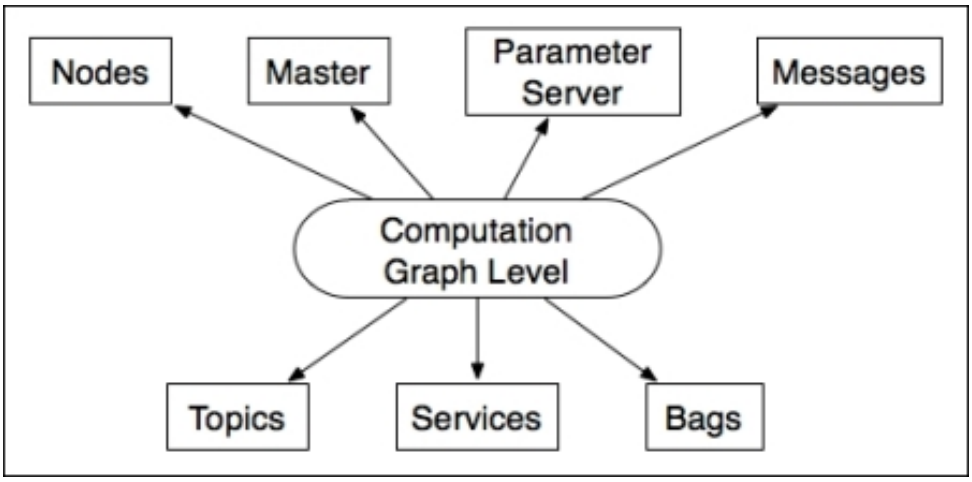

Figura 2.4: Grafo de computación de ROS

- **Nodos:** Los nodos son las unidades de proceso principales de un sistema basado en ROS. Cada nodo realiza un trabajo de computación. Esto confiere a ROS modularidad. En un sistema róbotico basado en ROS, coexisten un gran numero de nodos, cada uno responsable de una parte del sistema (velocidad de las ruedas, localización del robot, planificación de trayectorias,...). ROS proporciona bibliotecas cliente con una amplia documentación asociada que contienen todas las funciones necesarias para crear un nodo de ROS. Podemos destacar roscpp, una biblioteca de funciones ROS programada en C++, y rospy para Python.

- ***Master*:** ROS proporciona una entidad denominada *Master* y es la encargada de gestionar la ejecución de los nodos y de posibilitar la comunicación de los mismos dentro del sistema. Proporciona un servicio de registro de nombres para que cada nodo registre su identificador en el momento de entrar a formar parte del sistema. De esta manera el *Master*, lleva un control de aquellos nodos que están activos en el sistema y otros nodos pueden realizar consultas para obtener información sobre nodos ajenos a él. Además capacita a los nodos de poder establecer comunicación entre ellos, intercambiar mensajes, o de invocar servicios.

- **Servidor de parámetros:** El servidor de parámetros forma parte del *Master*. Permite mantener un diccionario de pares clave-valor, de manera que los nodos pueden consultar información por clave.

- **Mensajes:** Los nodos se comunican entre sí mediante el intercambio de mensajes. Un mensaje es simplemente una estructura de datos, que contiene campos con tipos de datos conocidos (e.g. string, float, int, etc.).

- **Topics:** Los topics exponen un punto de conexión del sistema para que los nodos puedan intercambiar información mediante mensajes. La comunicación a través de topics es unidireccional y asíncrona. Cada *topic* tiene asociado un identificador del contenido del mensaje y un tipo de mensaje. El modelo de intercambio de mensajes es publicador/subscriptor. De esta manera se consigue transferencia de información uno a uno, uno a muchos, muchos a uno y muchos a muchos. Un nodo o nodos envían un mensaje mediante la publicación de un mensaje en un *topic* determinado. El nodo o nodos que necesitan información de ese tipo de datos se subscribe al *topic* correspondiente y pueden leer el contenido. En general, los productores de información son los publicadores y los suscriptores son los consumidores. De esta manera se permite separar la producción del consumo de mensajes. Cada publicador/subscriptor tiene un *buffer* de tamaño programable que utiliza para el intercambio de información.

- **Servicios:** Los servicios surgen para completar el modelo de comunicación entre procesos. Proveen al sistema de un mecanismo de comunicación síncrono. El modelo de comunicación publicador/subscriptor es un paradigma de comunicación muy flexible, pero no es apropiado para una interacción petición/respuesta que en muchas ocasiones es requerido en un sistema distribuido. La petición/respuesta se realiza a través de servicios, los cuales se definen mediante mensajes estructurados. Los nodos ofrecen un servicio con un identificador para que otros nodos clientes los utilicen. Así un nodo cliente enviará un mensaje de petición al servicio con identificador deseado y el nodo propietario de ese servicio responderá con un mensaje de respuesta.

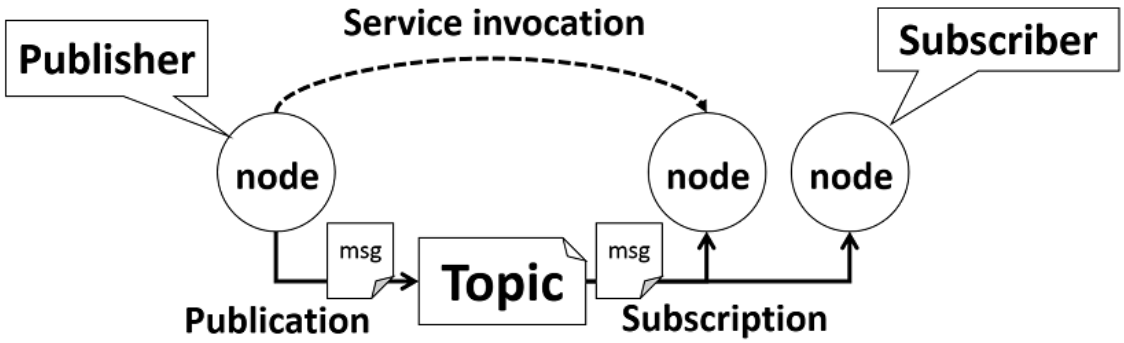

Figura 2.5: Comunicación entre nodos en ROS

- **Bags:** Estas entidades permiten guardar datos de mensajes ROS en un formato especial y reproducirlos más tarde. Son un importante mecanismo para el almacenamiento de datos, como datos de sensores, que pueden ser difíciles de recoger pero necesarios para desarrollar y probar algoritmos.

2.1.4. Herramientas de ROS en línea de órdenes

A continuación se presentan algunas de las herramientas que se utilizan para el desarrollo de software con ROS.

- **roscore:** Permite poner en funcionamiento al *Master* de ROS. Internamente lanza el servidor de parámetros y el nodo rosout que se encarga de la monitorización del sistema.
- **rosrun:** Permite al sistema lanzar un nodo indicando el nombre de su paquete y el nombre del nodo a ejecutar.
- **roslaunch:** Permite lanzar archivos con extensión launch. Estos archivos tienen un formato XML y usan el lenguaje de etiquetas para describir lanzamiento de nodos, configuración de parámetros, inclusión de ficheros y otras opciones. Con roslaunch es posible lanzar múltiples nodos de ROS en local o remotamente a través de SSH.
- **rosnode:** Permite mostrar información de depuración de los nodos que se están ejecutando en el sistema ROS, incluyendo subscriptores, publicadores y conexiones.
- **rostopic:** Permite obtener información sobre los topics de ROS, incluyendo publicadores y subscriptores de cada *topic*, su frecuencia de actualización o su tipo de mensaje asociado.
- **rosservice:** Permite mostrar los servicios que están disponibles en todo el sistema ROS y hacer peticiones sobre cualquiera de ellos.
- **rosparam:** Permite obtener y establecer parámetros del servidor de parámetros de ROS. También se le puede dar valor mediante el uso de un archivo de extensión yaml.
- **catkin:** Es el sistema de compilación de paquetes de ROS. Combina macros de CMAKE y scripts de Python para dotar de funcionalidad sobre el flujo de trabajo de CMAKE.

2.1.5. Herramientas gráficas de ROS

- **RViz:** Es un paquete de ROS que contiene una potente herramienta de visualización 3D. El interés de esta herramienta radica en su capacidad para mostrar *topics* tales como nubes de puntos, trayectorias calculadas o transformadas junto con el modelo 3D del robot.
- **rqt:** Es un conjunto de herramientas software que implementan diversas herramientas GUI en forma de *pluggins*. Estos se pueden añadir en una ventana de rqt o pueden ejecutarse individualmente. Los usuarios pueden desarrollar sus propios *pluggins* para rqt utilizando Python o C++.

- **Gazebo:** El simulador más ampliamente utilizado por los usuarios de ROS es Gazebo. Gazebo es un simulador bien diseñado que permite probar rápidamente algoritmos, diseños de robots, y realizar pruebas de regresión utilizando escenarios realistas. Gazebo ofrece la posibilidad de simular con precisión y eficiencia poblaciones de robots en entornos interiores y exteriores complejos. Integra un robusto motor de físicas, gráficos de alta calidad, y las interfaces de programación y gráficos más convenientes. ROS se integra en Gazebo a través de su paquete Gazebo-ROS y ofrece generación de datos de sensores (con o sin ruido), como láseres 2D, cámaras 2D/3D, cámaras RGB-D, sensores de contacto, fuerza o par y así como la posibilidad de usar modelos creados por el usuario.

2.1.6. Otros paquetes de ROS

ROS ha asimilado muchas herramientas, algoritmos y sistemas para servir de base a complejos controles robóticos. Una completa colección de paquetes ofrecen algoritmos de procesado de visión, interpretado de nubes de puntos y localización simultanea y mapeado (SLAM), entre otros.

Rosbridge

Rosbridge es un paquete de ROS que surgió para poder extender toda la funcionalidad de ROS a la web. Esta herramienta aporta una capa de abstracción sobre ROS. Rosbridge convierte toda la arquitectura ROS en un *backend*, por lo que no es necesario que los desarrolladores de aplicaciones web deban tener conocimientos de bajo nivel de interfaces de control, sistemas basados en *middleware* y complicados algoritmos de medida y control. El protocolo Rosbridge proporciona acceso a los mensajes y servicios de ROS como objetos de JavaScript Object Notation (JSON), y permite controlar ejecución de nodos y parámetros de entorno. Rosbridge usa el protocolo websocket, que está construido sobre HTML, lo que hace que cualquier navegador web apto para utilizar esta tecnología, sin necesidad de instalación.

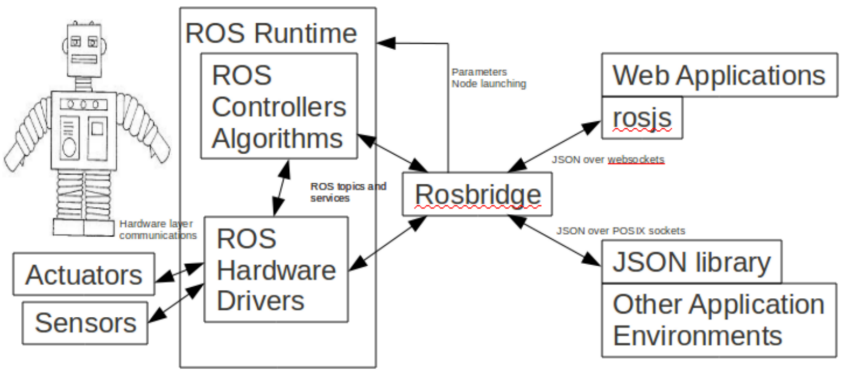

Figura 2.6: Relaciones de Rosbridge con otros componentes de sistema

Robot Web Tools (RWT)

A partir de Rosbridge, surge en 2012 el proyecto Robot Web Tools (RWT)[5]. El proyecto perseguía dotar de mayores niveles de interoperabilidad y portabilidad a un amplio espectro de sistemas robóticos, dispositivos e interfaces de usuario. Los objetivos de ese proyecto eran: 1) dotar de bibliotecas de visualización web para robots de manera que se puedan desarrollar interfaces robot-humano más usables e intuitivas; 2) posibilitar el uso de robótica en la nube, reduciendo el ancho de banda de los mecanismos de transporte. El mayor aporte de este proyecto ha sido la versión 2.0 del protocolo rosbridge y varias bibliotecas Javascript que facilitan la interacción robot-humano en la web. Destacan la biblioteca roslibjs (biblioteca cliente para comunicar con topics, servicios y acciones con ROS), ros2djs y ros3djs (que permiten introducir elementos en el navegador para visualizar robots, mapas, etc.).

Paquete SLAM-gmapping

Este paquete contiene un envoltorio de ROS para el paquete externo OpenSlam Gmapping. El paquete proporciona localización simultánea y mapeado basándose en la información de un láser 2D montado sobre el robot. Esto permite crear un mapa de rejilla de ocupación 2D (parecido a un plano de la planta de un edificio) a partir de los datos que provienen el láser y la información de posición recogida por el robot móvil (odometría).

Pila de navegación ROS

La pila de navegación es bastante sencilla conceptualmente. Básicamente recoge información de la odometría y del sensor láser y envía órdenes de velocidad para mover un robot móvil. Los paquetes más importantes que se utilizan dentro de la pila de navegación son, por una parte, el paquete de AMCL, que proporciona la localización del robot dentro de un mapa en base a información probabilística, y por otro lado, el paquete move_base, que integra toda la información de posición, velocidad, obstáculos detectados, obstáculos estáticos, para mover un robot móvil de un punto a otro.

2.2 Estado del Arte

Muchas aplicaciones de usuario web han permitido interactuar con el entorno a los usuarios no expertos de una manera amigable, independiente de la plataforma y de acceso universal[10]. Las aplicaciones web se han empleado en diversas ocasiones en el campo de la robótica para operar robots a distancia, recolectar masivamente datos, sincronizar robots durante experimentos y estudiar nuevas formas de Interacción Robot-Humano (IRH). Los primeros experimentos llevados a cabo en el año 95, permitían controlar un brazo robótico para manipular bloques de colores con la ayuda de una interfaz de usuario en un navegador web[9]. Goldberg utilizó una aplicación web para la monitorización y mantenimiento de jardines a distancia, desde donde además se podía ordenar al robot

que cultivara y regara las plantas [17]. El trabajo de Crick et al. [6] permitía a 132 usuarios conectarse desde diversas localizaciones para teleoperar un robot por un laberinto utilizando una interfaz web y demostró que era posible realizar experimentos con usuarios a gran escala a través de interfaces web sencillas. Osentoski et al, fue pionero en utilizar rosbridge[7] y rosjs[8] para interactuar y visualizar datos de una plataforma robótica compleja. Recientemente, y gracias a la evolución de las tecnologías web, surge la idea de preparar laboratorios de robótica remotos, en los cuáles un robot real está disponible en una dirección web para realizar pruebas y medir resultados. La primeras pruebas de estos laboratorios fueron realizadas en los laboratorios de Willow Garage con el robot PR2. En ellas se ofrecia una interfaz web para poder trabajar remotamente con el robot PR2.

Con la reciente llegada de la computación en la nube y el llamado Internet de las Cosas (Internet of the things, IoT), también han aparecido nuevos modelos de servicios. En 2010 aparece el concepto de Robot as a service (RaaS), que consiste en exponer servicios y aplicaciones de robótica a través de servicios web en la nube [12]. Kato et al.[13], propusieron el Rsi Research cloud, el cual ofrecía servicios robóticos a través de Internet. Los autores explotaron el protocolo Robot Service Network Protocol (RSNP) para ocultar complejidades y exponer servicios robóticos a usuarios no expertos, en concreto, mediante la puesta en marcha de un servicio de vídeo vigilancia. Koubâa et al. [14], con su trabajo RoboWeb, propusieron una arquitectura orientada a servicio basado en protocolo SOAP con el objetivo de virtualizar recursos de hardware y software robótico y exponerlos como servicios a través de la web. La finalidad de este trabajo era ofrecer un laboratorio de robótica remoto, que funcionara con ROS, para que investigadores y estudiantes pudieran acceder a las plataformas robóticas soportadas por ROS. El protocolo REST también se han utilizado para llevar a cabo trabajos en este sentido. Souza et al. [15] presentó una aproximación para adoptar los servicios web REST y superar las limitaciones de las comunicaciones basadas en el protocolo RPC. Otros autores [16] han propuesto una arquitectura RESTful, con una capa de aplicación y otra de servicios REST, para incrementar la flexibilidad y mantenibilidad de los sistemas. De esta arquitectura ha surgido la propuesta de interfaces y recursos para definir servicios robóticos. Recientemente, se ha desarrollado ROStfull [23], una implementación de código abierto de un servidor de servicios web ligero que utiliza rosbridge para dar soporte a servicios web REST. El servidor expone topics, servicios y acciones a través de RESTful.

Este trabajo pretende aportar una herramienta que permita visualizar información del robot a través de una interfaz web. En el caso que nos ocupa esta herramienta permite visualizar el mapa y el robot durante el proceso de generación del mapa, y permite visualizar el mapa y enviar puntos de navegación del mapa al robot para que navegue autónomamente.

2.3 Marco del proyecto

El proyecto se ha realizado durante el periodo de prácticas de empresa en Robotnik Automation SLL. La empresa se dedica a la robótica de servicios desde hace más de 10 años y cuenta con una media de 20 empleados, formado por un equipo humano especializado en diferentes ingenierías. Desde su inicio han desarrollado e introducido en el mercado mundial varias plataformas móviles, con un gran impacto en el campo de la investigación. Colaboran además con diversos proyectos europeos del programa Horizon2020, aportando en algunos casos soporte para robots o participando en el desarrollo del software necesario para llevar a cabo el proyecto.

El proyecto RADIO

El desarrollo de este proyecto también se enmarca dentro de uno de los proyectos europeos del programa antes citado. El proyecto tiene por titulo "Robots in Assisted Living Environments (RADIO): Unobtrusive, Efficient, Reliable and Modular Solutions for Independent Ageing" (http://www.radio-project.eu/).El proyecto trata de encontrar un modo de monitorizar y detectar patrones de envejecimiento en personas mayores a través de un robot de servicio personal y un conjunto de sensores domóticos situados en el hogar del anciano. De este modo, la actuación del personal sanitario puede ser más rápida y eficiente ante una situación de riesgo.

La herramienta desarrollada, viene a ser una ayuda para que personal no técnico (e.g. médicos, enfermeros, cuidadores, etc.) pueda generar mapas del entorno donde va a estar el robot y después mediante otras herramientas pueda navegar autónomamente a ciertos puntos del hogar del anciano.

CAPÍTULO 3
Solución adoptada

En el marco teórico se presentaron algunos trabajos en los que los autores habían utilizado previamente la combinación de ROS con interfaces gráficas de usuario web para visualizar información del robot.

Siguiendo ese mismo enfoque, la solución que se ha adoptado para el problema expuesto en la introducción, ha sido diseñar e implementar una interfaz web de usuario, en la cual se le permite a un usuario no experto generar un mapa y navegar por dicho mapa utilizando los algoritmos de generación de mapas y navegación de ROS. Esta solución requiere de un mecanismo que permita gestionar la ejecución y la parada de nodos necesarios para dicho propósito ejecutándose en el *backend* de la aplicación. Ambas partes se comunican a través de *websockets* utilizando rosbridge.

3.1 Análisis

En esta parte de la memoria, se va a llevar a cabo un análisis de los requisitos de la aplicación que se ha desarrollado. En primer lugar, se va a especificar los requisitos de la aplicación tanto funcionales, como no funcionales y de implementación. Por último se elabora un pequeño diagrama de casos de uso donde quedan especificados que operaciones puede realizar el usuario no experto dentro de la interfaz gráfica web.

3.1.1. Requisitos

Requisitos funcionales

A continuación se detallan todos los requisitos funcionales que debe cumplir la aplicación que se está desarrollando:

- El usuario podrá acceder a una interfaz de generación de mapa y otra de navegación.
- El usuario podrá generar mapas. Para ello tendrá que presionar un botón en la aplicación para que de esta forma se pongan en marcha ciertos nodos de ROS encargados del proceso.
- El usuario debe poder visualizar el mapa y el robot mientras realiza el proceso de generación de mapa.
- El usuario podrá salvar los mapas generados. Para ello tendrá que introducir un nombre deseado y presionar un botón en la aplicación y de esta forma se ponga en marcha el nodo responsable del proceso de guardado.
- El usuario debe poder recuperar los mapas generados, para ello tendrá que introducir el nombre el mapa deseado y presionar un botón para que se pongan en marcha los nodos de ROS responsables de la carga de mapa en el sistema.
- El usuario debe poder visualizar un mapa utilizando la interfaz de navegación y los nodos de ROS implicados en el proceso.
- El usuario debe poder navegar en el mapa de navegación dando puntos a los que debe navegar el robot.

Requisitos no funcionales

- La interfaz debe ser sencilla, atractiva e intuitiva, de tal forma que no suponga un esfuerzo al usuario a la hora de hace uso de ella.
- Se debe poder visualizar en cualquier navegador con HTML5 y Javascript.
- La aplicación debe consumir poco ancho de banda.
- La aplicación no disminuirá el rendimiento del robot.

Requisitos de implementación

- La interfaz de usuario estará desarrollada con tecnologías web (HTML, CSS y Javascript).
- La aplicación deberá implementar la manera de comunicar acciones del usuario y gestionar la ejecución y parada nodos.
- La aplicación podrá adaptarse al tamaño de la pantalla del dispositivo donde se esté ejecutando.

3.1.2. Casos de uso

El actor principal de la aplicación desarrollada es el usuario no experto y de él parten todas las acciones que se inician en el sistema.

En la figura 3.1, se hallan representados los casos de uso más importantes que se han extraído para la aplicación.

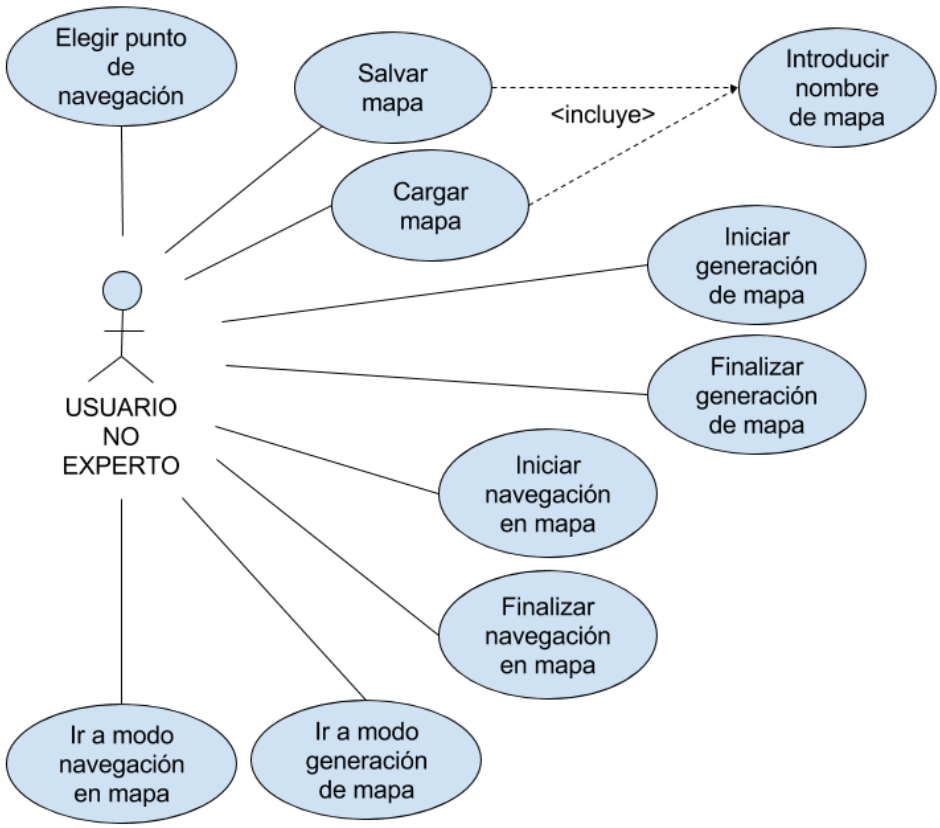

Figura 3.1: Casos de uso de la interfaz de usuario

Los casos de uso los que se van a especificar a continuación son:

- Caso de uso: Salvar mapa
- Caso de uso: Cargar mapa
- Caso de uso: Iniciar mapa
- Caso de uso: Finalizar mapa
- Caso de uso: Iniciar navegación
- Caso de uso: Finalizar navegación
- Caso de uso: Elegir punto de navegación

Caso de Uso	Salvar mapa			SAMA1		
Actores	Usuario no experto					
Tipo	Primario					
Referencias						
Precondición	El usuario se encuentra en la interfaz de generación de mapa y el proceso está iniciado. El usuario tiene que haber introducido un nombre en el mapa					
Autor	Jose Rapado	**Fecha**	01/06/2016	**Versión**	1.0	
Propósito						
Iniciar el proceso de guardado del mapa generado.						
Resumen						
El usuario inicia el proceso de guardado del mapa en el sistema mediante una petición al servicio correspondiente de la aplicación						
Curso normal (Básico)						
1	El usuario presiona el botón guardar mapa					
			2	Se lanza la petición al servicio.		
			3	Se muestra un mensaje al usuario		
Curso Alternos						
3b	Si el proceso de generación de mapa no esta iniciado se avisa al usuario de ello.					
3c	Si el usuario no ha introducido nombre de mapa se avisa al usuario					

Tabla 3.1: Caso de uso SAMA1: Salvar mapa

Caso de Uso	Cargar mapa		CAMA1		
Actores	Usuario no experto				
Tipo	Primario				
Referencias					
Precondición	El usuario se encuentra en la interfaz de navegación y la navegación en mapa no está iniciada. El usuario tiene que haber introducido un nombre en el mapa.				
Autor	Jose Rapado	**Fecha**	01/06/2016	**Versión**	1.0
Propósito					
Iniciar el proceso de carga del mapa para poder utilizarlo.					
Resumen					
El usuario inicia el proceso de carga del mapa en el sistema mediante una petición al servicio correspondiente de la aplicación.					
Curso normal (Básico)					
1	El usuario presiona el botón cargar mapa				
			2	Se lanza la petición al servicio.	
			3	Se muestra un mensaje al usuario.	
Cursos Alternos					
3b	Si el proceso de navegación está iniciado se avisa al usuario de ello.				
3c	Si el usuario no ha introducido nombre de mapa se avisa al usuario.				

Tabla 3.2: Caso de uso CAMA1: Cargar mapa

Caso de Uso	Iniciar mapa			INMA1		
Actores	Usuario no experto					
Tipo	Primario					
Referencias						
Precondición	El usuario se encuentra en la interfaz de generación de mapa y el proceso no está iniciado.					
Autor	Jose Rapado	**Fecha**	01/06/2016		**Versión**	1.0
Propósito						
Iniciar el proceso de generación de mapa para obtener un mapa del entorno.						
Resumen						
El usuario inicia el proceso de generación del mapa en el sistema mediante una petición al servicio correspondiente de la aplicación. El sistema lanza los procesos necesarios para que el robot genere el mapa y el usuario visualice el mapa en la rejilla junto con el robot.						
Curso normal (Básico)						
1	El usuario presiona el botón iniciar mapa.					
			2	Se lanza la petición al servicio.		
			3	Se muestra un mensaje al usuario.		
Cursos Alternos						
3b	Si el proceso de generacíon está iniciado se avisa al usuario de ello.					

Tabla 3.3: Caso de uso INMA1: Iniciar mapa

Caso de Uso	Finalizar mapa			FIMA1		
Actores	Usuario no experto					
Tipo	Primario					
Referencias						
Precondición	El usuario se encuentra en la interfaz de generación de mapa y el proceso está iniciado.					
Autor	Jose Rapado	**Fecha**	01/06/2016	**Versión**	1.0	
Propósito						
Finalizar el proceso de generación de mapa después de haber generado el mapa.						
Resumen						
El usuario finaliza el proceso de generación del mapa en el sistema mediante una petición al servicio correspondiente de la aplicación. El sistema para los procesos que se estaban ejecutando para la generación de mapas.						
Curso normal (Básico)						
1	El usuario presiona el botón finalizar mapa.					
			2	Se lanza la petición al servicio.		
			3	Se muestra un mensaje al usuario.		
Cursos Alternos						
3b	Si el proceso de generacíon no está iniciado se avisa al usuario de ello.					

Tabla 3.4: Caso de uso FIMA1: Finalizar mapa

Caso de Uso	Iniciar navegación			INNA1		
Actores	Usuario no experto					
Tipo	Primario					
Referencias						
Precondición	El usuario se encuentra en la interfaz de navegación basada en mapa y el proceso no está iniciado.					
Autor	Jose Rapado	**Fecha**	01/06/2016		**Versión**	1.0
Propósito						
Iniciar el proceso de navegación basado en mapa después de haber cargado el mapa.						
Resumen						
El usuario inicia el proceso de navegación en mapa en el sistema mediante una petición al servicio correspondiente de la aplicación. El sistema ejecuta los procesos necesarios para que el robot navegue autonómamente.						
Curso normal (Básico)						
1	El usuario presiona el botón de iniciar navegación.					
			2	Se lanza la petición al servicio.		
			3	Se muestra un mensaje al usuario.		
Cursos Alternos						
3b	Si el proceso de navegación está iniciado se avisa al usuario de ello.					

Tabla 3.5: Caso de uso INNA1: Iniciar navegación

Caso de Uso	Finalizar navegación		FINA1		
Actores	Usuario no experto				
Tipo	Primario				
Referencias					
Precondición	El usuario se encuentra en la interfaz de navegación basada en mapa y el proceso está iniciado.				
Autor	Jose Rapado	**Fecha**	01/06/2016	**Versión**	1.0
Propósito					
Finalizar el proceso de navegación basado en mapa después navegar por el mapa.					
Resumen					
El usuario finalizar el proceso de navegación de mapa en el sistema mediante una petición al servicio correspondiente de la aplicación. El sistema para los procesos necesarios para que el robot navegue autonómamente.					
Curso normal (Básico)					
1	El usuario presiona el botón finalizar navegación.				
			2	Se lanza la petición al servicio.	
			3	Se muestra un mensaje al usuario.	
Cursos Alternos					
3b	Si el proceso de navegación no está iniciado se avisa al usuario de ello.				

Tabla 3.6: Caso de uso FINA1: Finalizar navegación

Caso de Uso	Elegir punto de navegación		ENMA1		
Actores	Usuario no experto				
Tipo	Primario				
Referencias					
Precondición	El usuario se encuentra en la interfaz de navegación basada en mapa y el proceso está iniciado.				
Autor	Jose Rapado	**Fecha**	01/06/2016	**Versión**	1.0
Propósito					
Enviar un punto de navegación en el mapa, y que el robot navegue autónomamente.					
Resumen					
El usuario presiona sobre un punto del mapa con el objetivo de que el robot navegue autónomamente hasta ese punto.					
Curso normal (Básico)					
1	El usuario presiona sobre un punto del mapa de navegación				
			2	Se envía el punto al sistema.	
			3	Se muestra una marca sobre el mapa de navegación.	
Cursos Alternos					
1b	Si el usuario presiona otro punto de navegación mientras el robot se dirige al objetivo, se va al punto 2.				

Tabla 3.7: Caso de uso ENMA1: Elegir punto de navegación

Caso de Uso	Ir al modo de navegación basada en mapa			NAMA1	
Actores	Usuario no experto				
Tipo	Primario				
Referencias					
Precondición	El usuario se encuentra en el menú de selección de modo en la portada de la interfaz web. El usuario se encuentra en la interfaz de generación de mapa.				
Autor	Jose Rapado	**Fecha**	01/06/2016	**Versión**	1.0
Propósito					
Acceder a la interfaz web de usuario de navegación basada en mapa.					
Resumen					
El usuario desea acceder a la interfaz de navegación basada en mapa, bien desde la interfaz de generación de mapa o bien desde el menu de selección de la portada de la interfaz web.					
Curso normal (Básico)					
1	El usuario presiona sobre el botón navegar mapa.				
			2	Se muestra la interfaz de navegación basada en mapa.	
Cursos Alternos					

Tabla 3.8: Caso de uso NAMA1: Ir al modo navegación basada en mapa

Caso de Uso	Ir al modo de generación de mapa			GEMA1	
Actores	Usuario no experto				
Tipo	Primario				
Referencias					
Precondición	El usuario se encuentra en la portada de la interfaz web de usuario, en el menú de selección de modo. El usuario se encuentra en la interfaz de navegación basado en mapa.				
Autor	Jose Rapado	**Fecha**	01/06/2016	**Versión**	1.0
Propósito					
Acceder a la interfaz web de usuario de generación de mapa.					
Resumen					
El usuario desea acceder a la interfaz de generación de mapa, bien desde la interfaz de navegación basada en mapa o bien desde el menú de selección de la portada de la interfaz web.					
Curso normal (Básico)					
1	El usuario presiona sobre el botón generar mapa.				
			2	Se muestra la interfaz de generación de mapa.	
Cursos Alternos					

Tabla 3.9: Caso de uso GEMA1: Ir al modo generación de mapa

3.2 Diseño

En esta sección se va a describir el diseño que se va a utilizar para la aplicación. En primer lugar se describirá la arquitectura de la aplicación, el diseño de la interfaz web y el diseño del nodo de ROS que se ha desarrollado.

3.2.1. Arquitectura de la aplicación

La arquitectura de la aplicación se puede diferenciar en dos partes. Por una parte está el *frontend*, que está formado por la interfaz web de usuario. Por otra parte está el *backend*, formado por una serie de nodos e interfaces ROS (controladores del robot, algoritmos de generación de mapa y navegación, rosbridge, nodo mánager, etc.). El *backend* ofrecerá las interfaces web a través de un servidor web.

En la figura 3.2 se puede observar la arquitectura en la que va a funcionar la aplicación. En la parte ROS se diseñara un nodo que se encargado de relacionar la funcionalidad del la parte visual y trasera.

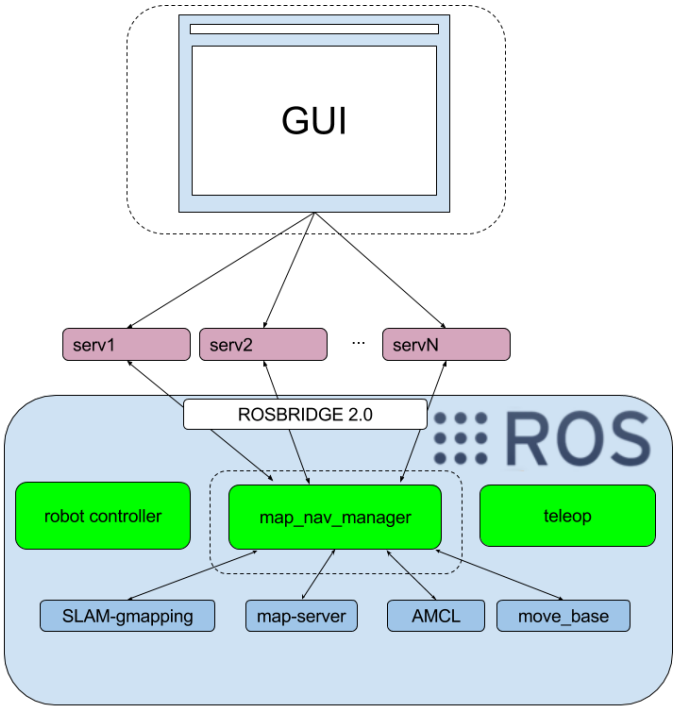

Figura 3.2: Arquitectura de la solución adoptada

3.2.2. Diagramas de secuencia

En el diagrama de secuencia se pueden ver como las diferentes partes del sistema van a interactuar para llevar a cabo los objetivos deseados.

A continuación, en las siguientes figuras, se van a presentar los diagramas de secuencia para los siguientes casos de uso más relevantes, y son:

- SAMA1: Salvar mapa
- CAMA1: Cargar mapa
- INMA1: Iniciar mapa
- FIMA1: Finalizar mapa
- INNA1: Iniciar navegación
- FINA1: Finalizar navegación
- ENMA1: Elegir punto de navegación

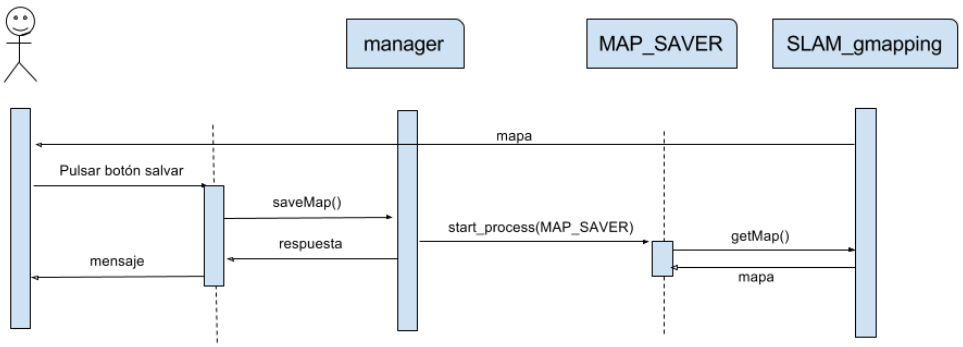

Figura 3.3: Diagrama de secuencia para SAMA1: Salvar mapa

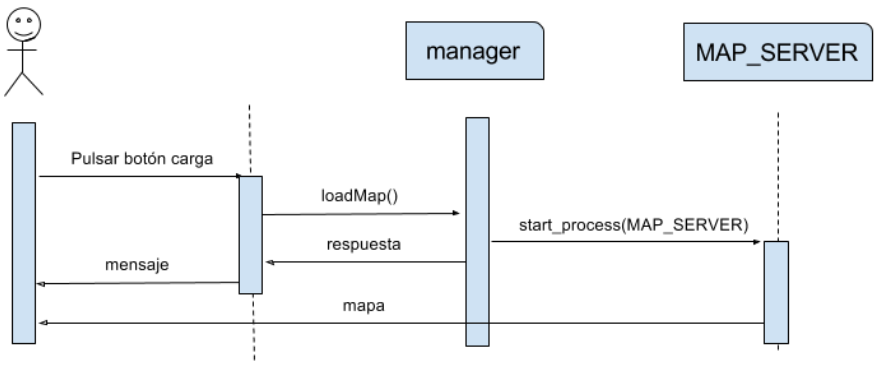

Figura 3.4: Diagrama de secuencia para CAMA1: Cargar mapa

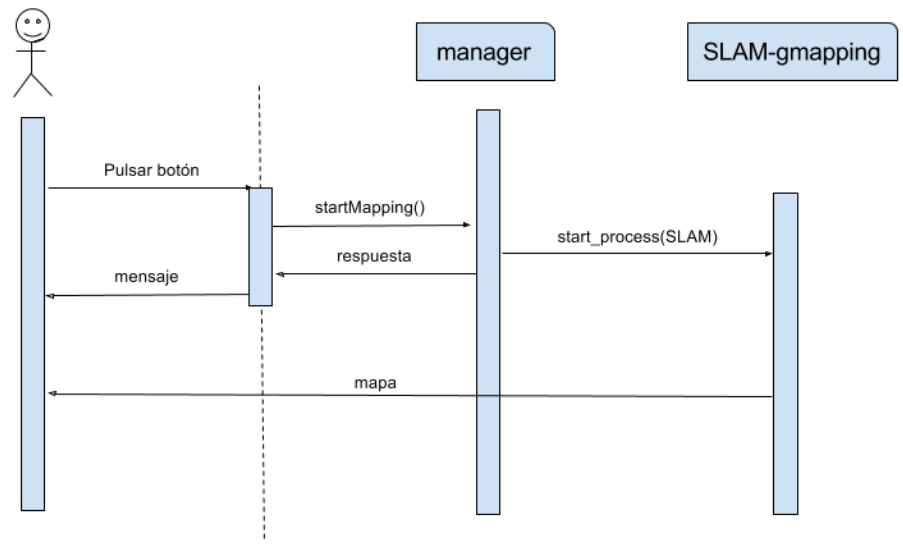

Figura 3.5: Diagrama de secuencia para INMA1: Iniciar mapa

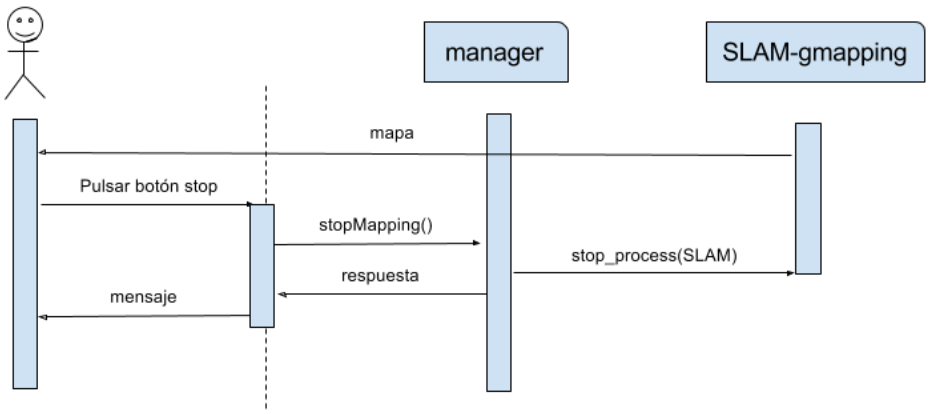

Figura 3.6: Diagrama de secuencia para FIMA1: Finalizar mapa

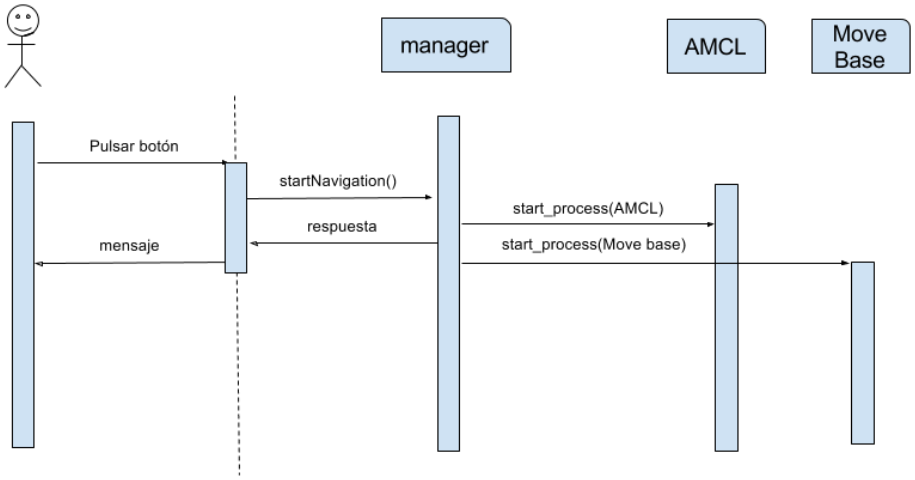

Figura 3.7: Diagrama de secuencia para INNA1: Iniciar navegación

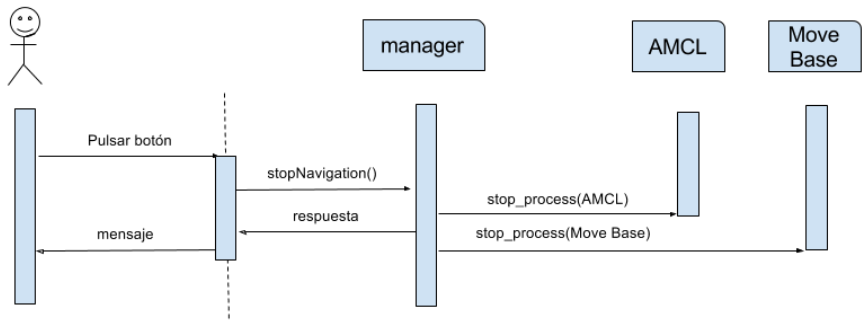

Figura 3.8: Diagrama de secuencia para FINA1: Finalizar navegación

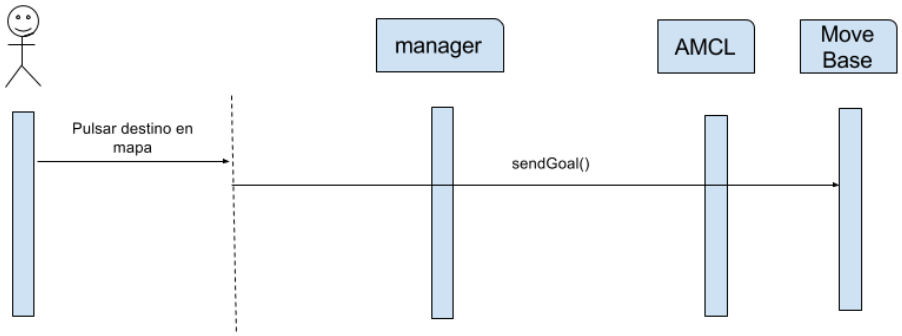

Figura 3.9: Diagrama de secuencia para ENMA1: Elegir punto de navegación

3.2.3. Diseño de la interfaz web

Las interfaces gráficas utilizadas en la aplicación deben recoger y dar funcionalidad a los casos de uso representados anteriormente. Para ello se va a utilizar los bocetos que realizados antes de desarrollar la aplicación y se va a describir el funcionamiento de cada una de sus partes y el flujo entre cada pantalla. Vamos a numerar las pantallas de modo que sea más fácil identificarlas. En la tabla 3.10 podemos encontrar dicha numeración

Número	Documento
1	index.html
2	mapping.html
3	navigation.html

Tabla 3.10: Pantallas de la interfaz gráfica

index.html: Este documento es la portada de la aplicación. Estará compuesta por dos imágenes que permitirán la navegación a otras pantallas. En la figura 3.10 se puede ver el boceto.

El flujo de control de esta pantalla es el siguiente:

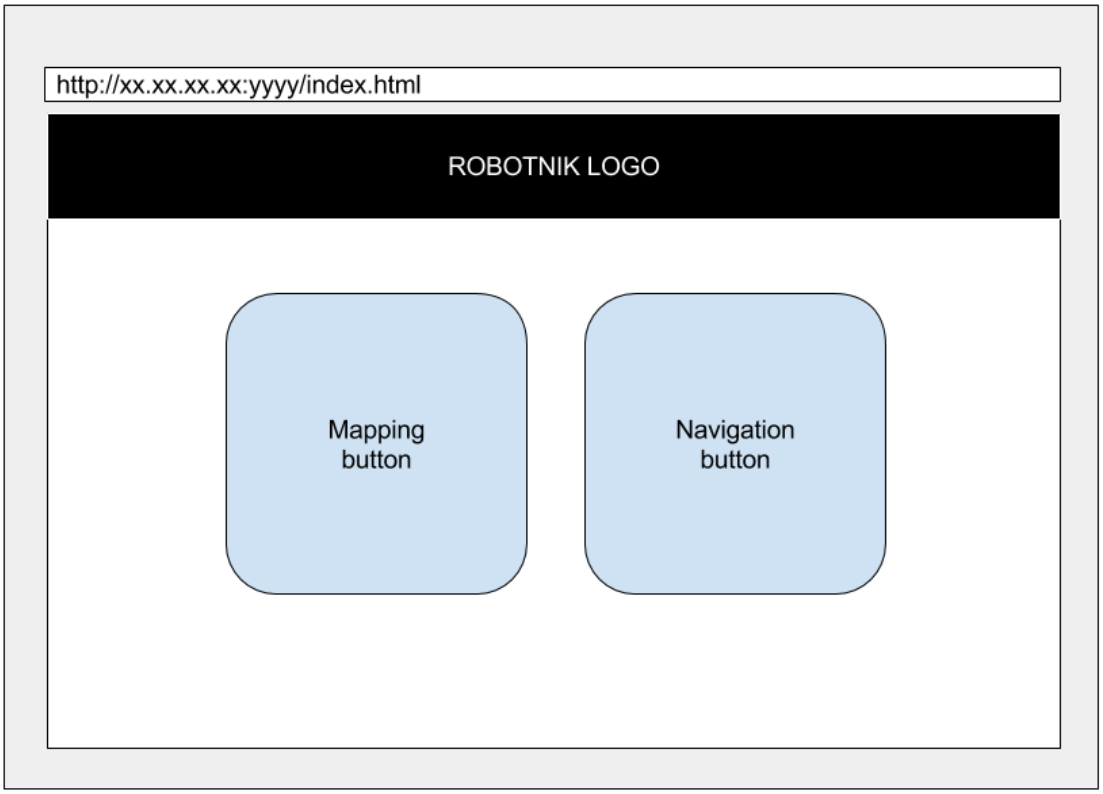

Figura 3.10: Boceto portada de la aplicación

- Si se presiona sobre el botón mapping, el usuario navegará hasta la ventana de generación de mapas (mapping.html).

- Si se presiona sobre el botón navigation, el usuario navegará hasta la ventana de navegación basada en mapa (navigation.html).

mapping.html: En este documento se recoge la visualización del mapa que se va a generar en el proceso y la interacción del usuario con el sistema para llevarlo a cabo. Estará compuesta por un visualizador 3D a la izquierda y un panel de control a la derecha, con botones que permitirán iniciar y finalizar el mapa, guardarlo y navegar hacia el resto de los documentos de la aplicación. En la figura 3.11 se puede ver el boceto.

El flujo de control de esta pantalla es el siguiente:

- Si se presiona el botón "Start", el sistema ejecutará los nodos ROS necesarios para realizar el proceso de generación de mapa e informará al usuario con un mensaje.

- Si se presiona sobre el botón "Stop", el sistema parará la ejecución de los nodos necesarios para realizar el proceso de generación de mapa e informará al usuario.

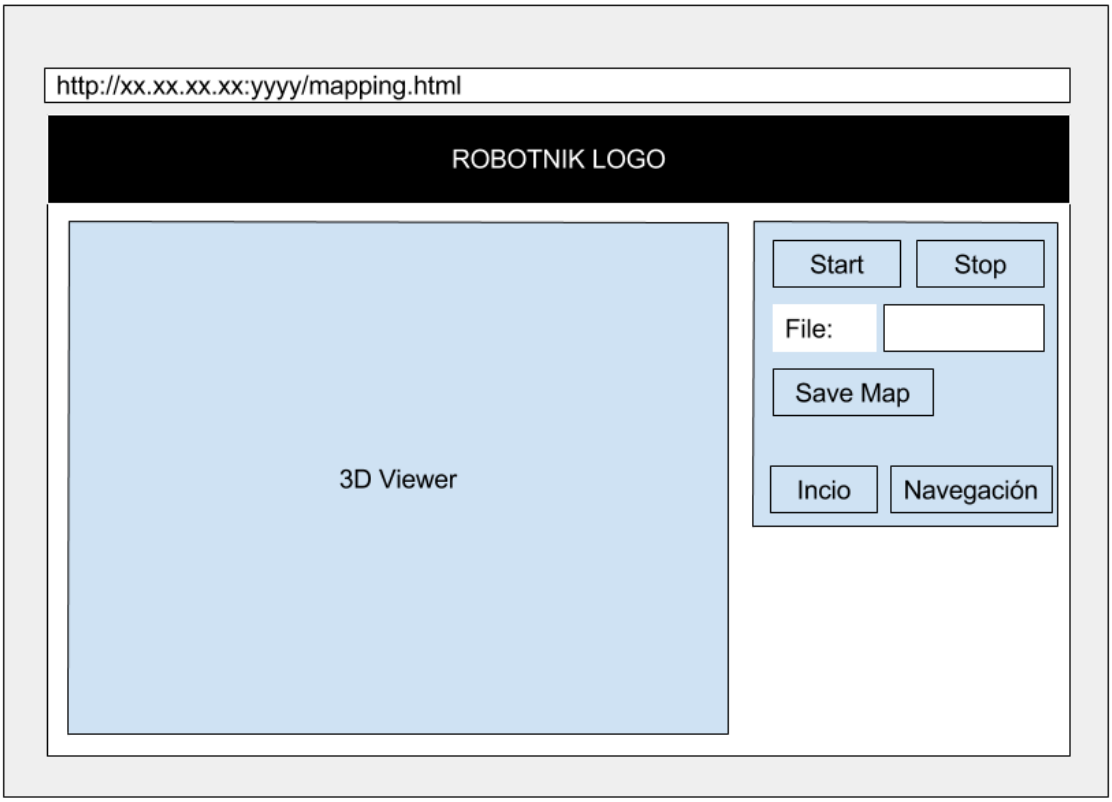

Figura 3.11: Boceto interfaz de generación de mapa

- Si se presiona sobre el botón "Save Map", el sistema recogerá el nombre del mapa recogido en el campo de texto "File" y ejecutará los nodos necesarios para guardar el mapa con ese nombre.
- Si se presiona sobre "Inicio", se volverá al documento inicial.
- Si se presiona sobre "Navegación", el sistema parará los nodos ROS implicados en la generación de mapa y el usuario navegará a la pantalla de navegación basada en mapa.

navigation.html: En este documento se recoge la visualización del mapa que se va a utilizar para mandar puntos de navegación y la interacción del usuario con el sistema para llevarlo a cabo. Estará compuesta por un visualizador 2D a la izquierda y un panel de control a la derecha con botones que permitirán iniciar y finalizar la navegación, cargar el mapa y navegar hacia el resto de los documentos de la aplicación. En la figura 3.12 se puede ver el boceto.

El flujo de control de esta pantalla es el siguiente:

- Si se presiona sobre el botón "Load Map", el sistema recogerá el nombre del mapa recogido en el campo de texto "File" y ejecutará los nodos necesarios para recuperar el mapa con ese nombre.
- Si se presiona el botón "Start", el sistema ejecutará los nodos ROS necesarios para realizar el proceso de navegación en mapa e informará al usuario con un mensaje.

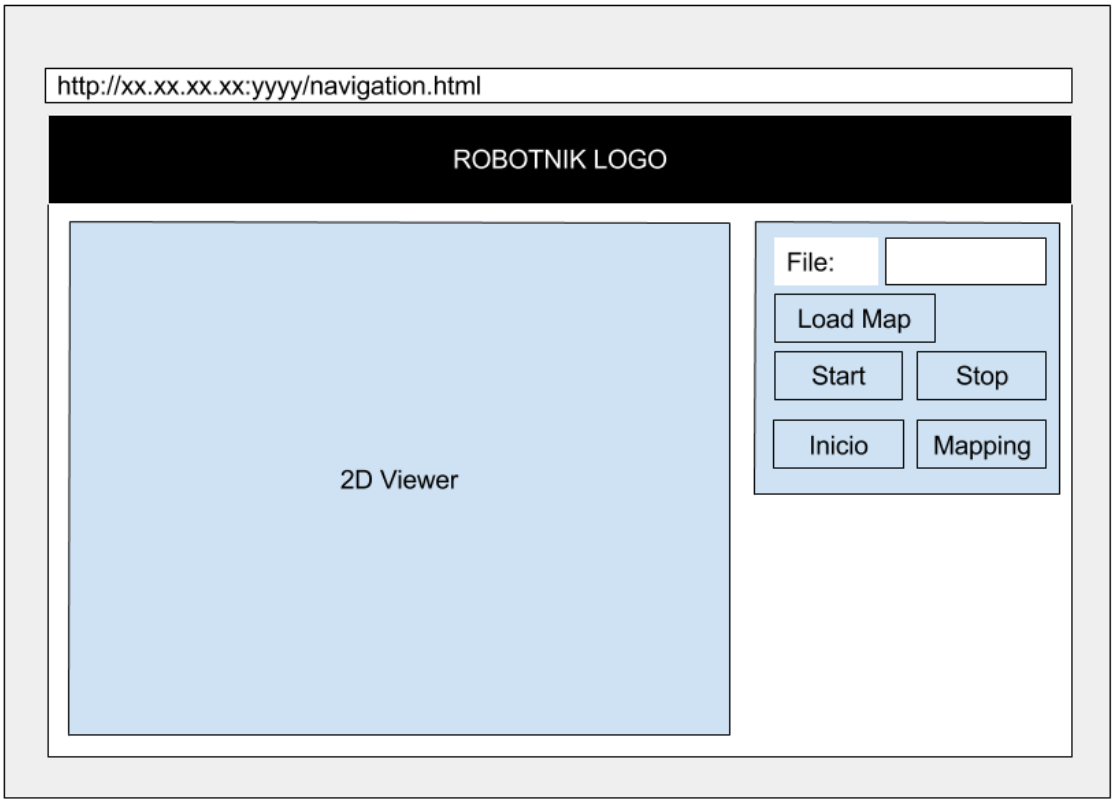

Figura 3.12: Boceto interfaz de navegación en mapa

- Si se presiona sobre el botón "Stop", el sistema parará la ejecución de los nodos necesarios para realizar el proceso de navegación en mapa e informará al usuario.
- Si se presiona sobre "Inicio", se volverá al documento inicial.
- Si se presiona sobre "Mapping", el sistema parará los nodos ROS implicados en la navegación en mapa y el usuario navegará a la pantalla de generación de mapas.
- Si se presiona un punto sobre el mapa el robot irá a ese punto.

3.2.4. Diseño del paquete map_nav_manager

El paquete de ROS **map_nav_manager** que se ha desarrollado contiene la siguiente arquitectura de directorios representado en la figura 3.13, siguiendo la arquitectura característica de un paquete de ROS.

- Directorio launch: Contiene los ficheros .launch que lanzan por una parte el servidor web, el servidor de rosbridge, así como los que lanzan los procesos de mapeado y navegación.
- Directorio map: Contiene los mapas generados mediante el uso de la interfaz de usuario web.

- Directorio scripts: Contiene scripts necesarios para la ejecución del servidor web mini-httpd.
- Directorio src: Contendrá la implementación del nodo de ROS.
- Directorio srv: Contiene la definición servicios propios del paquete y que se utilizar en el nodo map_nav_manager.
- Directorio web: Contiene los ficheros necesarios para visualizar la interfaz de usuario web desde cualquier navegador. En su interior podemos encontrar subcarpetas que incluyen librerias Javascript (js), estilos CSS (css) y otras carpetas necesarias.
- package.xml: En este fichero se describe las dependencias que puede tener el paquete con otros paquetes de ROS.
- CMakeLists.txt: En este fichero se recoge toda la configuración de CMake, necesaria para compilar el paquete con catkin.

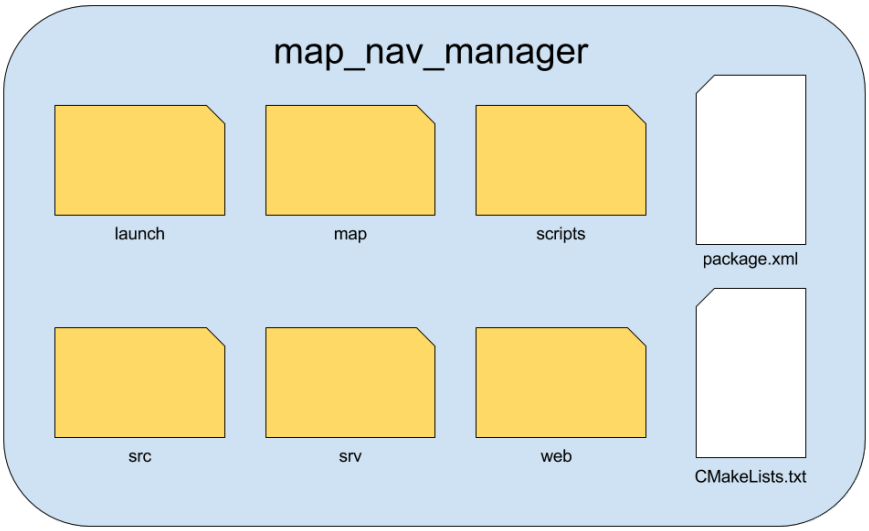

Figura 3.13: Estructura del paquete map_nav_manager

El nodo map_nav_manager deberá ofrecer servicios para poder comunicarse con la interfaz gráfica web. En la figura 3.14 podemos ver un esquema de las comunicaciones del nodo. Este estará registrado en el *Master* de ROS y ofrecerá los servicios siguientes:

- Iniciar mapeado
- Parar mapeado
- Iniciar navegacíón
- Parar navegación
- Cargar mapa
- Salvar mapa

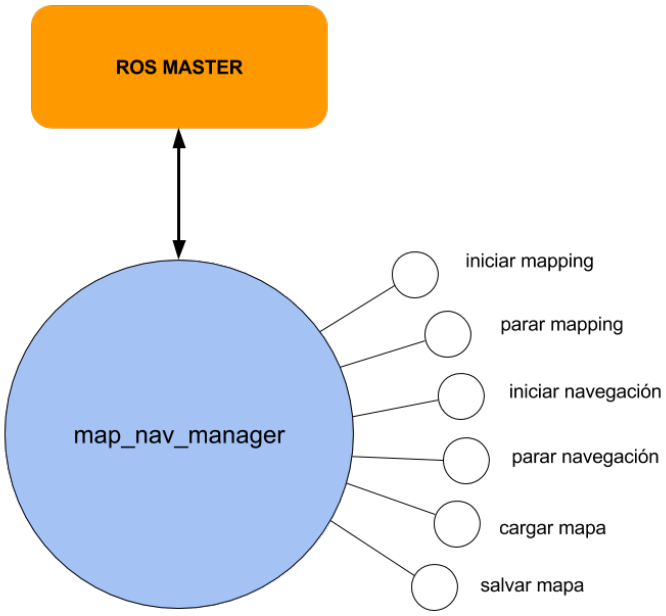

Figura 3.14: Esquema comunicaciones del nodo map_nav_manager

3.3 Implementación de la solución

3.3.1. Interfaz gráfica

El primer paso ha sido el desarrollo de la interfaz gráfica, para estar seguro de la viabilidad de la aplicación a través de tecnologías web.

Para implementar la interfaz de usuario se ha utilizado Bootstrap. Bootstrap combina una hoja de estilos con Javascript para poder dar estilo visual a la estructura del documento HTML. De esta manera se obtienen interfaces web visualmente atractivas, menos complicadas y que se ajustan al tamaño de la pantalla.

También se han incluido bibliotecas de Robot Web Tools que también están escritas en Javascript para añadir widgets de visualización de objetos 3D y rejillas de navegación 2D.

Los gráficos utilizados para los botones de generación y navegación de mapa se han obtenido de `https://icons8.com/`. Se ha respetado la licencia al incluir el enlace a la fuente.

Como editor de archivos se ha utilizado **Sublimetext2**, ya que permite multitud de configuraciones especiales y añadidos para facilitar la producción de código.

Portada de la aplicación

La portada, que es el punto de entrada a la aplicación, se encuentra en el documento **index.html**. Desde este documento se puede navegar a otros dos documentos: el documento de generación de mapa, **mapping.html**, y el documento de navegación basado en mapa, **navigation.html**, que se describen posteriormente. El documento **index.html** está compuesta por una cabecera. Debajo aparece otro espacio para el contenido, en el que pueden encontrarse dos botones. A la izquierda (en 1) un botón de un mapa con un lápiz, que al pulsarlo nos lleva a la página de generación de mapa. A la derecha (en 2) otro botón con forma de mapa con localizador que nos lleva al documento de navegación basado en mapa.

La implementación de la funcionalidad de la portada es bastante sencillo (ver figura 3.15). La funcionalidad del botón marcado como 1, es un simple *hiperlink* asociado a la imagen, que al presionarlo nos lleva a la interfaz de generación de mapa. Es similar para la funcionalidad del botón 2.

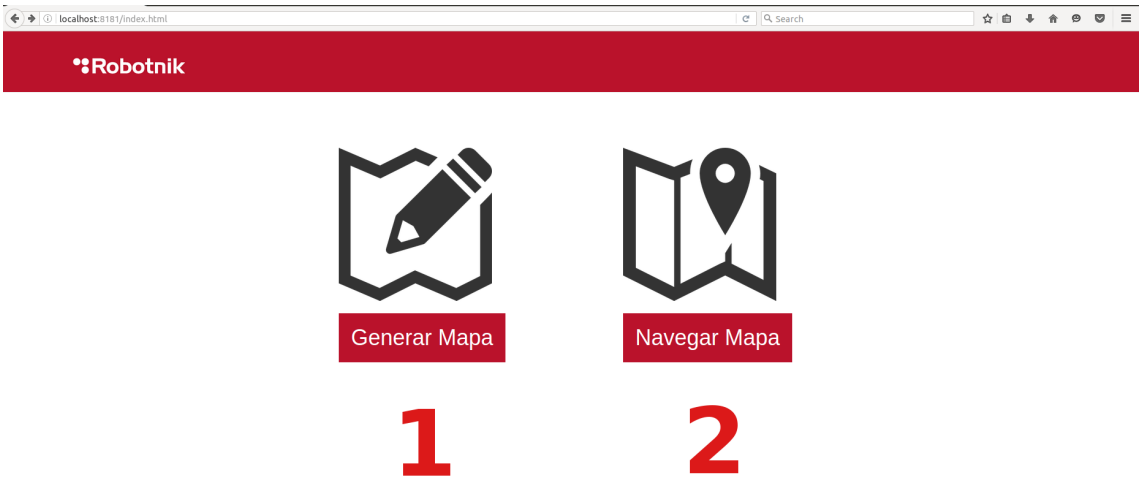

Figura 3.15: Interfaz visual entrada aplicación

Interfaz gráfica de generación de mapa

En la figura 3.16 se puede visualizar la interfaz gráfica desarrollada para generar mapas de entorno.

Para comunicar la interfaz con el sistema ROS se ha utilizado la biblioteca **roslib.js** que permite crear una comunicación mediante websockets con Rosbridge. Además es la biblioteca cliente que implementa los mecanismos de comunicación con ROS.

Para introducir el visualizador (en 1) se han utilizado bibliotecas **ros3d.js** de Robot Web Tools, que permite insertar un *widget* en el documento web.

Para dotar de funcionalidad al panel de botones de la parte derecha del documento se ha creado un fichero Javascript con funciones.

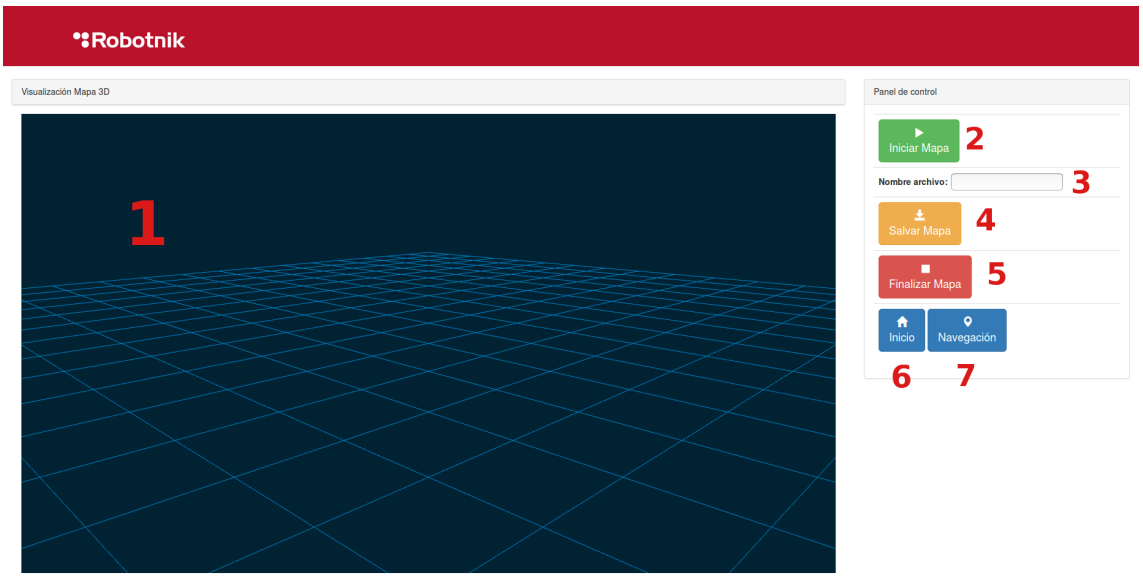

Figura 3.16: Interfaz visual de generación de mapa

A continuación se enumeran los elementos que aparecen en el panel de control y cómo se ha implementado su funcionalidad:

- El botón **Iniciar Mapa** (en 2) permite iniciar el proceso de generación de mapa. Este botón tiene asociada la función "startMapping". La función comprueba que la generación de mapa no está llevándose a cabo en ese momento y si es así, crea un cliente para el servicio ofrecido por el mánager de ROS start_mapping_srv, y envía una petición utilizando un mensaje de tipo Trigger. Además recoge la respuesta del servicio y la muestra por pantalla en una ventana de advertencia.

- El campo de texto (en 3) permite al usuario introducir una cadena de texto, que indica el nombre que le quiere dar al mapa.

- El botón **Salvar Mapa** (en 4) permite salvar el mapa generado. Tiene asociada la función "saveMap". Esta función comprueba primero que la generación de mapa se está llevando a cabo y si es así comprueba que el campo de texto (en 3) no esté vacío. En caso de que esté vacío, envía un mensaje al usuario avisando que es necesario rellenar el campo de texto. Si todo está correcto esta función crea un cliente para el servicio de ROS save_map_srv y envía una petición utilizando un mensaje de tipo SetFilename. Esta función rellena el campo de petición del servicio con el nombre que haya especificado en el cuadro de texto y lo envía al mánager.

- El botón **Finalizar Mapa** (en 5) permite finalizar el proceso de generación de mapa. Este botón tiene asociada la función "stopMapping". Esta función comprueba si la generación de mapa esta activa. Si no es asi manda un mensaje al usuario informando de que el proceso de generación de mapa no se

ha iniciado. Por el contrario, si el proceso está activo crea un cliente para el servicio de ROS stop_mapping_srv que ofrece el nodo mánager y envía una petición utilizando un mensaje de tipo Trigger. Además recoge la respuesta del servicio y la muestra por pantalla en una ventana de advertencia.

- El botón **Inicio** (en 6) permite al usuario navegar al documento inicial. Este botón tiene asociada la función "goIndex". Esta función comprueba si el proceso de generación de mapa está activo y si es así finaliza el proceso y vuelve al documento inicial.

- El botón **Navegación** (en 7) da la funcionalidad de navegar hacia la interfaz de navegación basada en mapa. Tiene asociado la función "goNavigation". Esta función comprueba si se está realizando la generación de mapa y si es así finaliza el proceso y se dirige al documento de navegación.

Interfaz gráfica de navegación en mapa

La interfaz gráfica para navegación en mapa se encuentra en la figura 3.17. Sigue un diseño similar al mostrado en la interfaz anterior.

Como anteriormente, esta interfaz necesita utilizar la biblioteca **roslib.js** para comunicarse con el sistema ROS.

Para visualizar el mapa de navegación, se ha utilizado un *widget* de la biblioteca de Robot Web Tools **nav2d.js**.

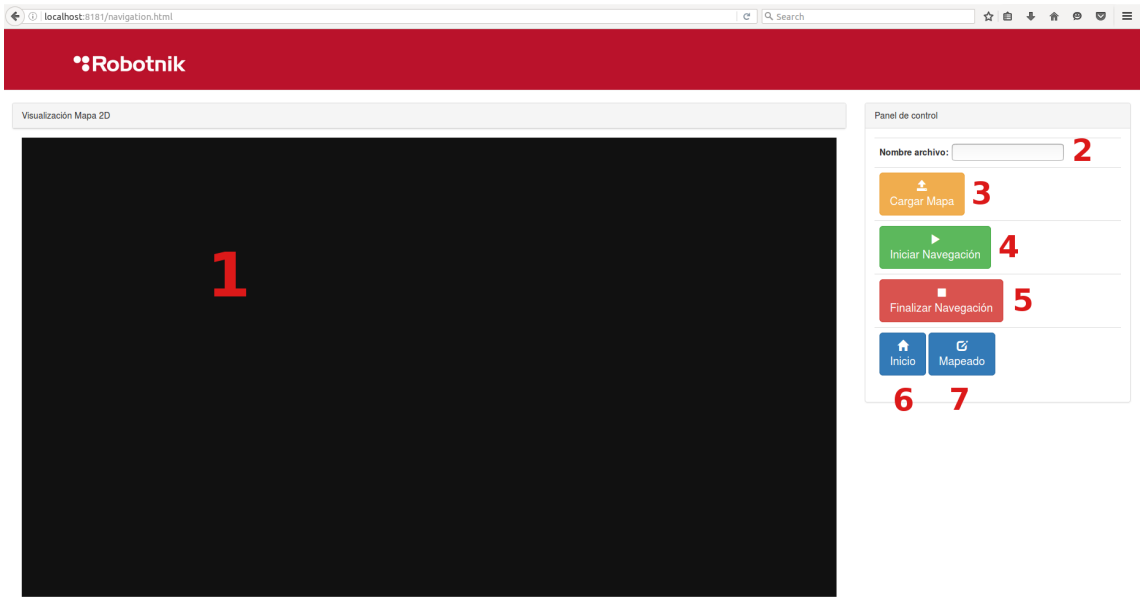

Figura 3.17: Interfaz visual de navegación de mapa

A continuación se enumeran los elementos que aparecen en el panel de control y cómo se ha implementado su funcionalidad:

- El campo de texto (en 2) permite al usuario introducir una cadena de texto, que indica el nombre que le quiere tiene el mapa a recuperar.
- El botón **Cargar Mapa** (en 3) permite cargar el mapa generado. Su funcionalidad la implementa la función "loadMap". Esta función comprueba primero que la generación de mapa no se está llevando a cabo y si es así comprueba que el campo de texto (en 2) no esté vacío. En caso de que esté vacío, envía un mensaje al usuario avisando que es necesario rellenar el campo de texto. Si todo está correcto esta función crea un cliente para el servicio de ROS load_map_srv y envía una petición utilizando un mensaje de tipo SetFilename. Esta función rellena el campo de petición del servicio con el nombre que haya especificado en el cuadro de texto y lo envía al mánager.
- El botón **Iniciar Navegación** (en 4) permite iniciar el proceso de navegación basado en el mapa cargado previamente. Este botón tiene asociada la función "startNavigation". La función comprueba que la navegación de mapa no está llevándose a cabo en ese momento y si es así, crea un cliente para el servicio ofrecido por el mánager de ROS start_navigation_srv, y envía una petición utilizando un mensaje de tipo Trigger. Además recoge la respuesta del servicio y la muestra por pantalla en una ventana de advertencia.
- El botón **Finalizar Navegación** (en 5) permite finalizar el proceso de navegación basada en mapa. Su funcionalidad esta implementada en la función "stopNavigation". Esta función comprueba si la navegación en mapa esta activa. Si no es así manda un mensaje al usuario informando de que el proceso de generación de mapa no se ha iniciado. Por el contrario, si el proceso está activo crea un cliente para el servicio de ROS stop_navigation_srv que ofrece el nodo mánager y envía una petición utilizando un mensaje de tipo Trigger, para finalizar el proceso. Además recoge la respuesta del servicio y la muestra por pantalla en una ventana de advertencia.
- El botón **Inicio** (en 6) permite al usuario navegar al documento inicial. Este botón tiene asociada la función "goIndex". Esta función comprueba si el proceso de navegación en mapa está activo y si es así finaliza el proceso y vuelve al documento inicial.
- El botón **Navegación** (en 7) da la funcionalidad de navegar hacia la interfaz de generación de mapa. Tiene asociado la función "goMapping". Esta función comprueba si se está realizando la navegación en mapa y si es así finaliza el proceso y se dirige al documento de generación de mapa.

3.3.2. Nodo map_nav_manager

Para la programación del nodo de ROS encargado de gestionar la ejecución y parada de otros nodos ROS a demanda desde la interfaz web, se ha utilizado el lenguaje de programación Python, debido a que ROS tiene soporte para este lenguaje con su biblioteca **rospy**.

La idea inicial fue implementar un nodo que pudiera funcionar de manera similar a como lo hace la herramienta de ROS **roslaunch**. Esto era leer los archivos

con extensión roslaunch y sacar la información de los nodos que se lanzaban en ellos, así como sus parámetros específicos de configuración para esa ejecución. Se consultó la documentación de la biblioteca pero finalmente no fue posible sacar mucho de esa idea porque la documentación era imprecisa.

Finalmente se optó por otra solución más sencilla de implementar. Esta pasaba por utilizar la bibliotecas de Python **shlex** y **subprocess**. La biblioteca shlex permite hacer análisis léxico de órdenes tipo UNIX y separarlas en lexemas mas sencillos para después utilizarlas con subprocess.

La biblioteca subprocess permite ejecutar procesos, conectarse a sus tuberías de entrada/salida/error, y obtener sus códigos de retorno. Combinando estas dos bibliotecas se consiguió poder ejecutar y parar nodos programáticamente.

También era necesario utilizar la biblioteca de funciones de **rosparam** para borrar parámetros del servidor de parámetros de ROS cuando se paran los procesos.

La lógica del nodo, es pues bastante sencilla. Se ha implementando un nodo de ROS que es un servidor que expone seis servicios. Cuando la interfaz web hace una llamada a un determinado servicio, una función de *callback* se encarga de procesar la petición y lanzar el proceso requerido.

Los servicios que expone el nodo y su funcionalidad se comentan a continuación:

- **start_mapping_srv**. Este servicio es de tipo std_srvs/Trigger y tiene la **callback** asociada startMappingServiceCb. Esta función se encarga de recoger la petición y de lanzar el nodo de ROS SLAM-gmapping que interviene en la creación del mapa.
- **stop_mapping_srv**. Este servicio es de tipo std_srvs/Trigger y tiene la **callback** asociada stopMappingServiceCb. Esta función se encarga de recoger la petición y de parar la ejecución del nodo SLAM-gmapping, a la vez que borra cualquier parámetro relacionado con el nodo en el servidor de parámetros.
- **start_navigation_srv**. Este servicio es de tipo std_srvs/Trigger y tiene la **callback** asociada startNavigationServiceCb. Esta función se encarga de recoger la petición y de lanzar la ejecución del nodo ROS para AMCL y el nodo de Move Base, que permiten la navegación sobre un mapa.
- **stop_navigation_srv**. Este servicio es de tipo std_srvs/Trigger y tiene la **callback** asociada stopNavigationServiceCb. Esta función se encarga de recoger la petición y de finalizar la ejecución del nodo ROS para AMCL y el nodo de Move Base, a la vez que borra cualquier parámetro relacionado con ellos en el servidor de parámetros.
- **load_map_srv**. Este servicio es de tipo map_nav_manager/SetFilename y tiene la *callback* asociada loadMapServiceCb. Esta función se encarga de recoger la petición y el nombre del mapa a cargar y lanza la ejecución del nodo map_server de ROS, que permite cargar mapas desde un fichero.

- **save_map_srv**. Este servicio es de tipo map_nav_manager/SetFilename y tiene la **callback** asociada saveMapServiceCb. Esta función se encarga de recoger la petición y el nombre del mapa a salvar y lanza la ejecución del nodo map_saver de ROS, que permite guardar mapas generados.

La definición de los servicios Tigger y SetFilename se pueden consultar en el anexo A.1 y A.2

3.4 Ejemplo de funcionamiento

Para realizar este ejemplo se ha utilizado la simulación del robot movil Turtlebot 2, que ha sido el robot utilizado en el proyecto RADIO, mencionado anteriormente en el marco del proyecto.

A continuación se va a realizar un ejemplo completo de ejecución del proceso de generación de mapa a través la interfaz de usuario. Se va a generar el mapa de tres habitaciones de la planta del hospital donde se lleva a cabo el proyecto RADIO. En la figura 3.18 se puede ver la planta y el robot simulados en el simulador Gazebo. Aunque simulado el proceso no difiere del realizado con un robot real. Para mover el robot es necesario teleoperarlo mediante un mando de PlayStation 3 o mediante el teclado.

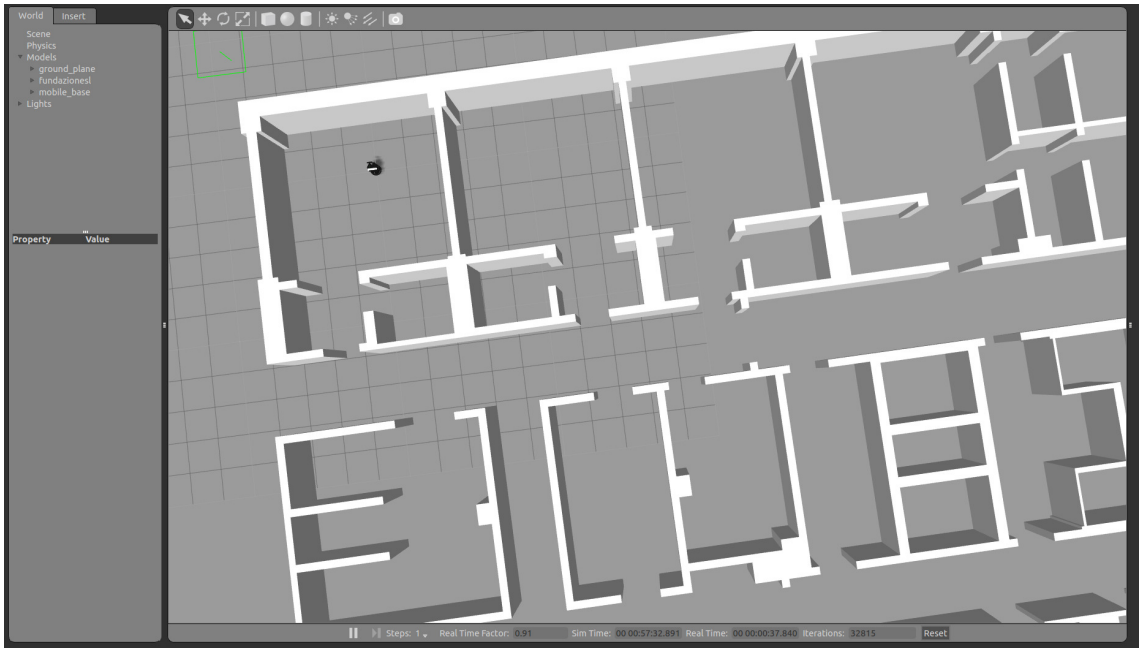

Figura 3.18: Robot utilizado para realizar la demostración de generación de mapa visto en el simulador Gazebo

Antes de lanzar la ejecución del nodo map_nav_manager el robot debe estar con todos los nodos de ROS que lo controlan funcionando y publicando información en el topic de odometria (/odom) y publicando sus transformadas en el topic de transformadas (/tf). Una vez se ha lanzado el nodo map_nav_manager, tendremos el servidor web, el servidor de rosbridge y todo lo necesario funcionando.

3.4.1. Generación de un mapa

Una vez realizado lo anterior basta con abrir un navegador e introducir la dirección IP del robot y el puerto 8181. El servidor web de este ejemplo está en local y está disponible en la dirección web http://localhost:8181. En la figura 3.19, se puede observar el documento portada de la interfaz de usuario.

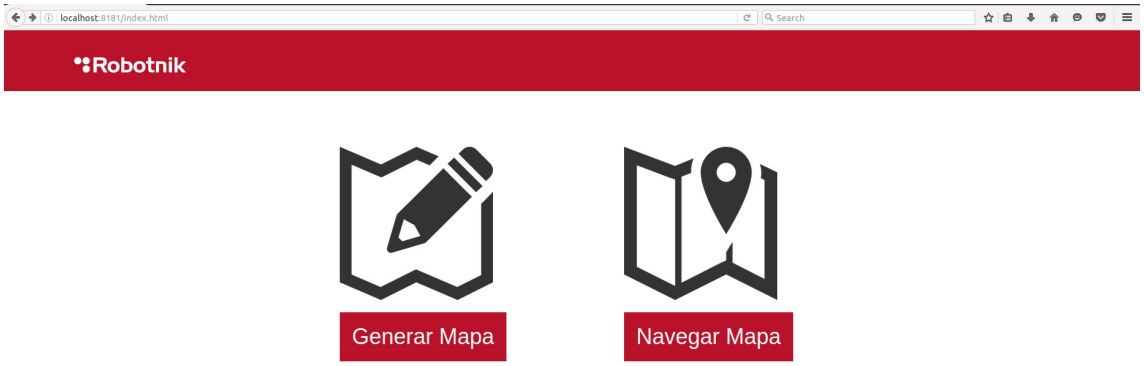

Figura 3.19: Documento inicial de la interfaz de usuario web

A continuación se presiona en el botón "Generar Mapa"para ir a la interfaz de generación de mapa. En el visualizador 3.20 aparece el modelo del robot con todas las partes amontonadas. Esto es normal ya que el visualizador 3D está tomando como referencia el topic /map, y no existe hasta que el nodo de SLAM-gmapping esté lanzado.

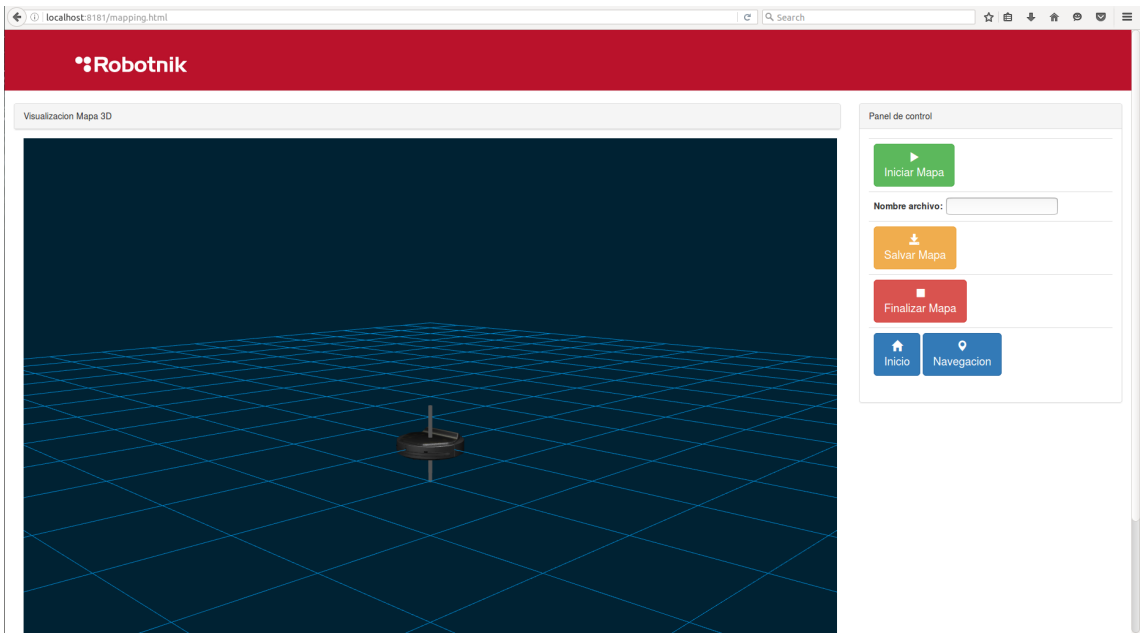

Figura 3.20: Vista de la rejilla 3D

Para iniciar el proceso de generación de mapa se presiona el botón **Iniciar Mapa** de la interfaz web. De este modo se lanza la petición a través de un servicio desde la interfaz web hacia el nodo map_nav_manager para ejecutar el nodo SLAM-gmapping. La aplicación nos informa de que el nodo se ha lanzado correc-

tamente 3.21. Una vez lanzado el nodo se empieza a visualizar el robot y el mapa correctamente 3.22.

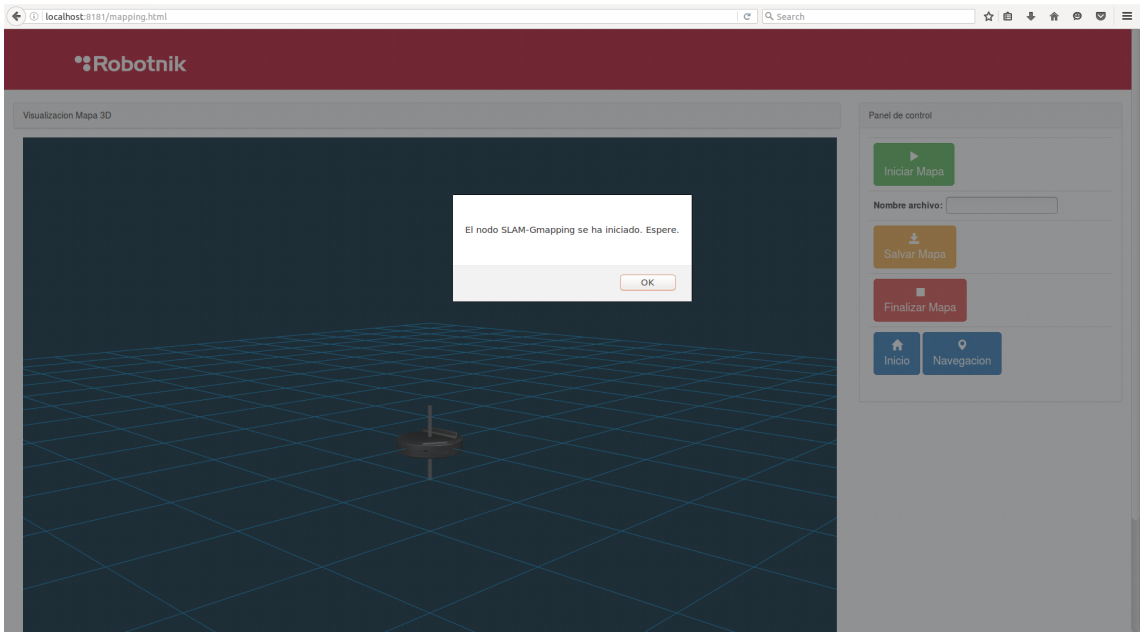

Figura 3.21: Mensaje de advertencia de que el proceso SLAM-gmapping ha sido lanzado

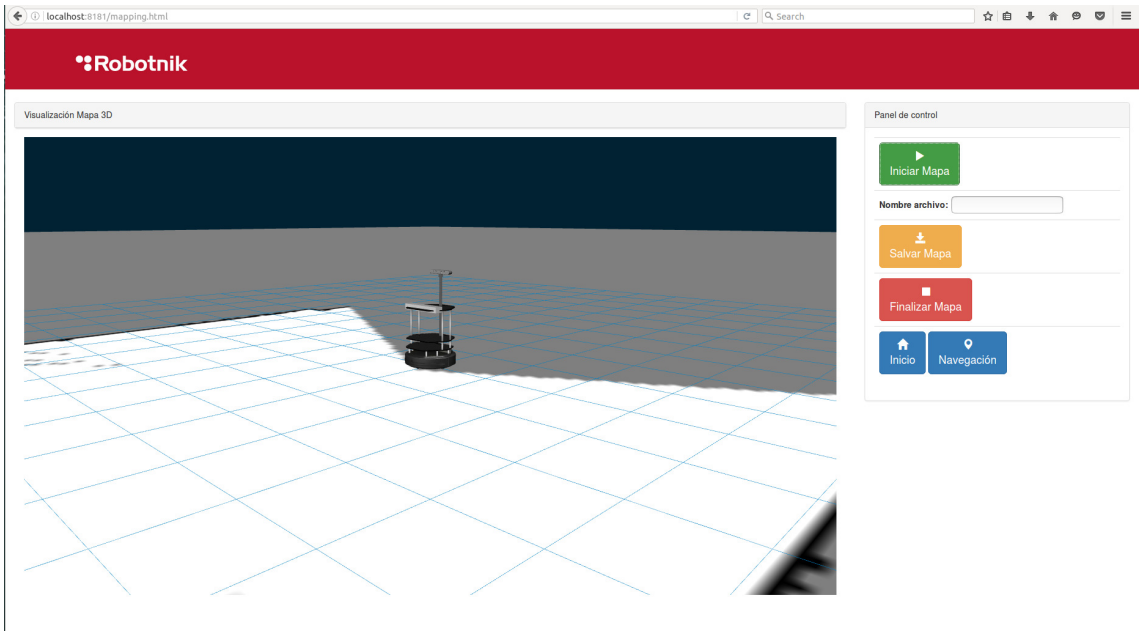

Figura 3.22: Visualización del robot y el mapa

A partir de ese momento, se ha de teleoperar el robot para que vaya obteniendo el mapa del entorno. En la figura 3.23 se observa que el robot se ha ido moviendo por el entorno y ha ido obteniendo el mapa de de algunas habitaciones.

En la figura 3.24 se observa al robot llegando a la tercera habitación, que también se va a explorar para que se incluya en el mapa. Una vez realizado el proceso y fuera de la habitación, se introduce el nombre que se desee para el mapa y se pulsa en el botón **Guardar Mapa**. Para este ejemplo se ha usado el nombre *tresha-*

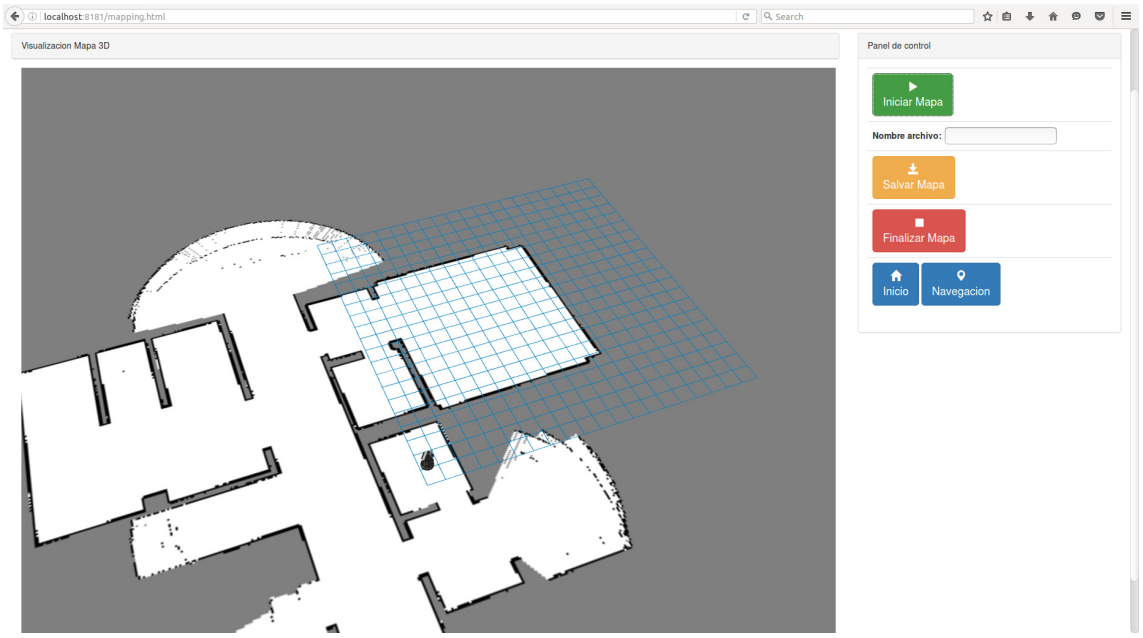

Figura 3.23: Visualización del mapa en la segunda habitación

bitaciones. En la figura 3.25 se puede ver como la interfaz advierte de que el mapa se ha guardado correctamente.

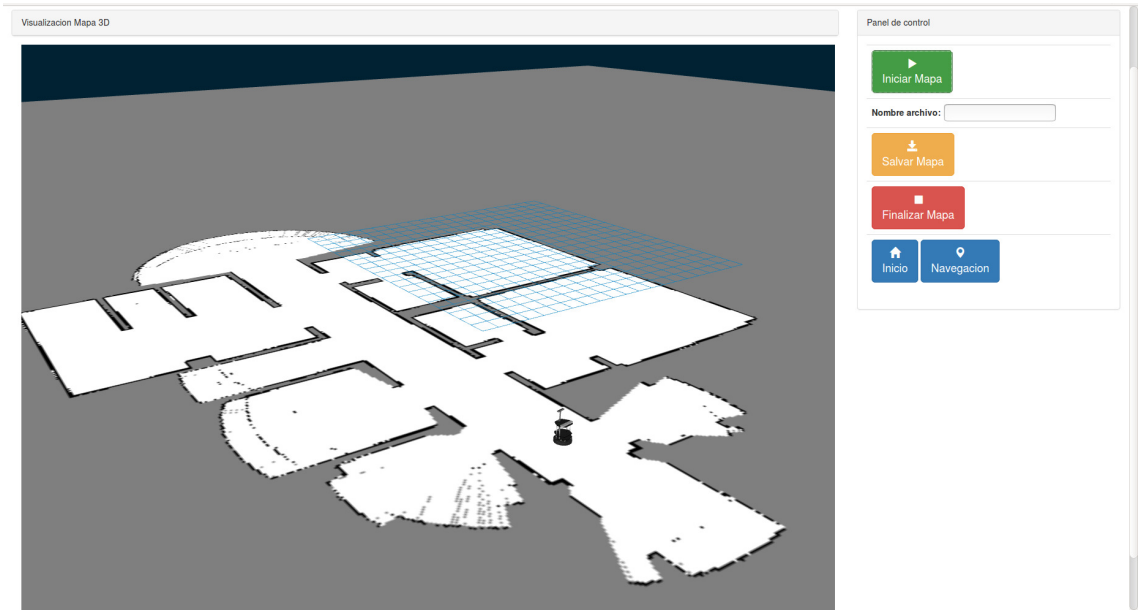

Figura 3.24: Visualización del robot y el mapa en una rejilla 3D

Para concluir el proceso de generación de mapa, se presiona el botón **Finalizar Mapa**, para finalizar el nodo SLAM-gmapping de ROS. La aplicación muestra el aviso de que el proceso ha finalizado con éxito. 3.26

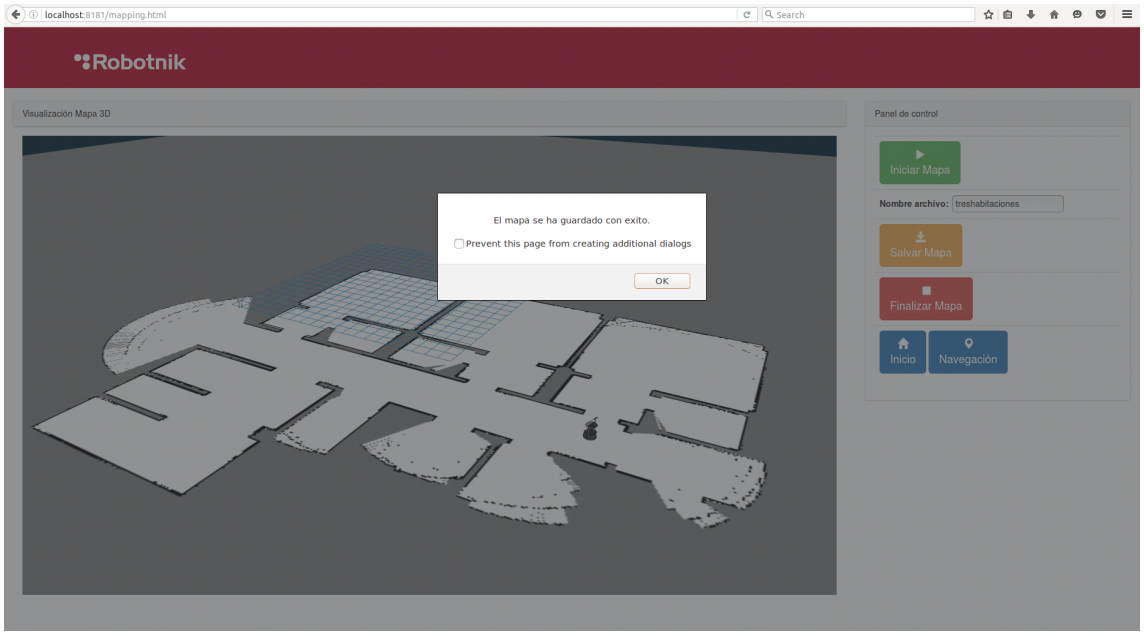

Figura 3.25: Mensaje de advertencia informando al usuario que el mapa se ha guardado con éxito

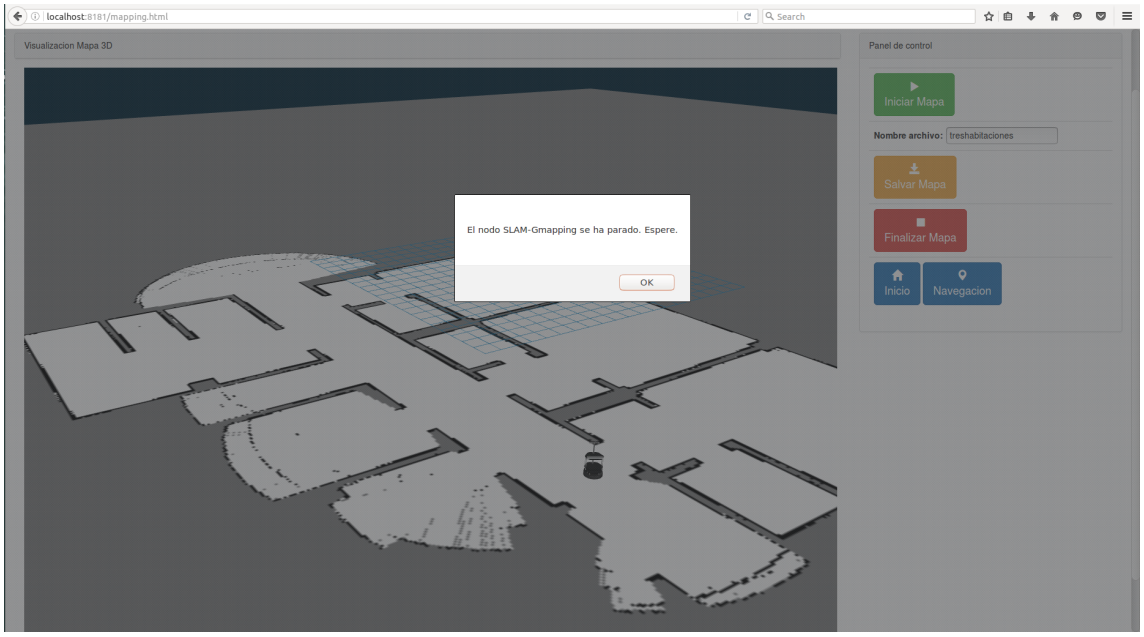

Figura 3.26: Mensaje de advertencia informando al usuario que el proceso SLAM-gmapping se ha detenido

3.4.2. Navegación basada en mapa

A continuación se va a realizar un ejemplo completo de ejecución del proceso de navegación basado en el mapa que hemos generado en el punto anterior. Para acceder a la interfaz web de navegación en mapa se puede hacer desde el documento inicial de la aplicación pulsando sobre el botón **Navegar Mapa**. Seguidamente se carga la interfaz gráfica de navegación (figura 3.27).

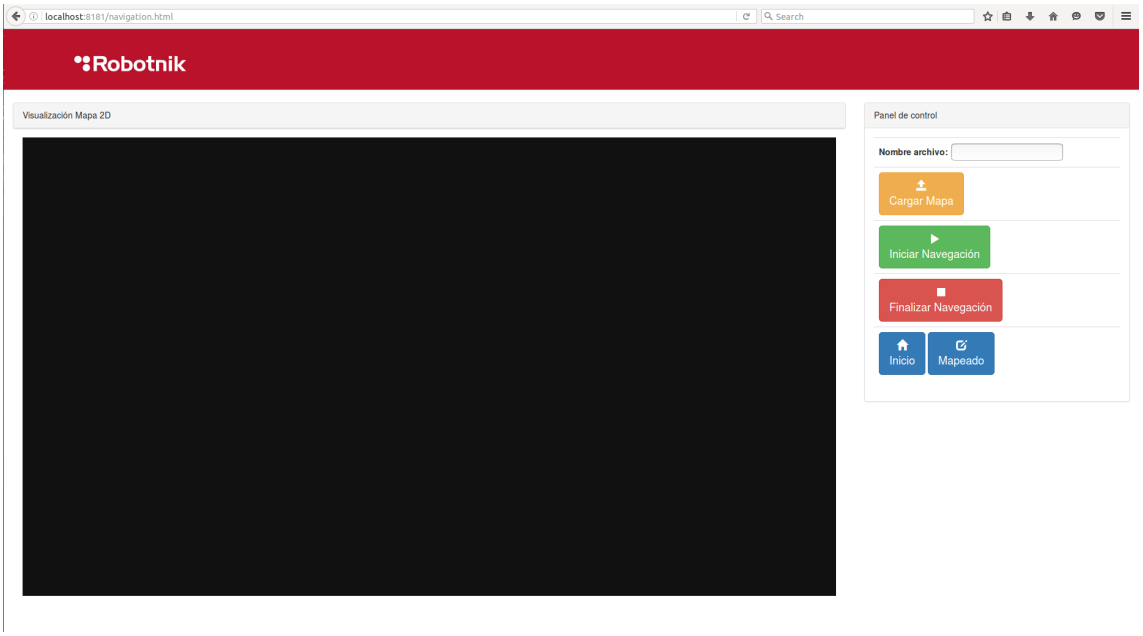

Figura 3.27: Interfaz gráfica de navegación basada en mapa

Después se introduce el nombre del mapa que se desea cargar. En este caso se va a usar el generado anteriormente, asi que se especifica *treshabitaciones* en el nombre del archivo. En la figura 3.28 se puede ver como la aplicación ha cargado el mapa y ha mostrado un mensaje al usuario de que el proceso se ha completado con éxito. Tras aceptar la advertencia se muestra el mapa en la rejilla de visualización 2D, como se ve en la figura 3.29.

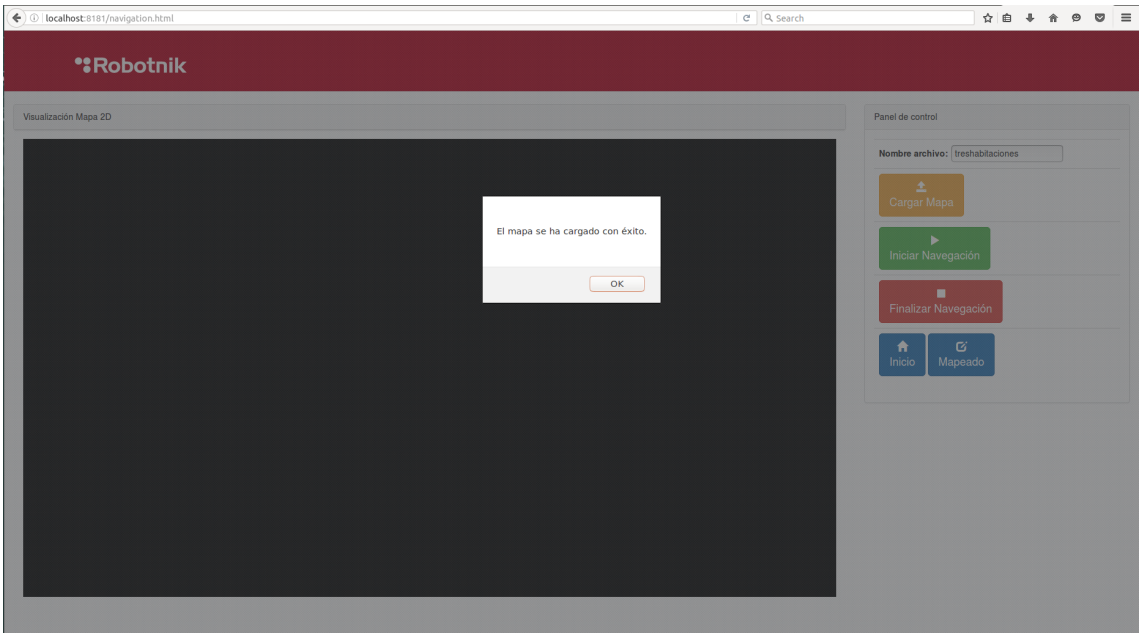

Figura 3.28: Mensaje de advertencia informando al usuario que el mapa se ha cargado con éxito

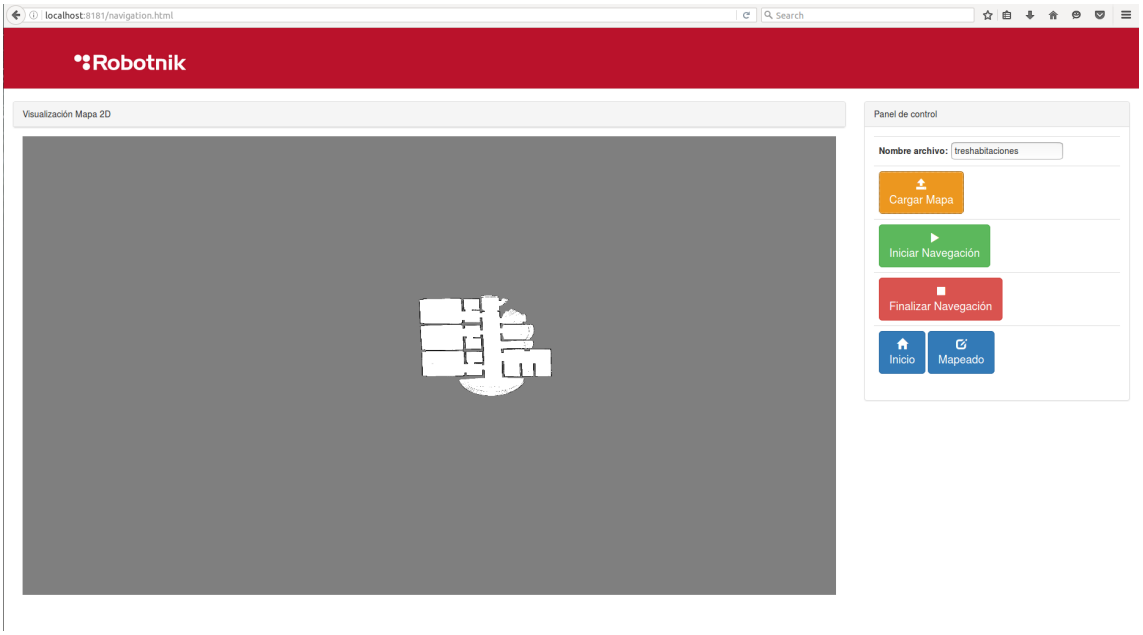

Figura 3.29: Mapa visualizado en la rejilla 2D

Seguidamente se presiona el botón **Iniciar Navegación** para lanzar los nodos de ROS necesarios para poder efectuar la navegación por el mapa cargado. En la figura 3.30 se observa el mensaje indicando al usuario que el proceso se ha iniciado correctamente.Tras aceptar el mensaje se puede ver en la figura 3.31, un triangulo naranja sobre el mapa que representa al robot. El vértice apuntando hacia adelante indica la orientación del robot.

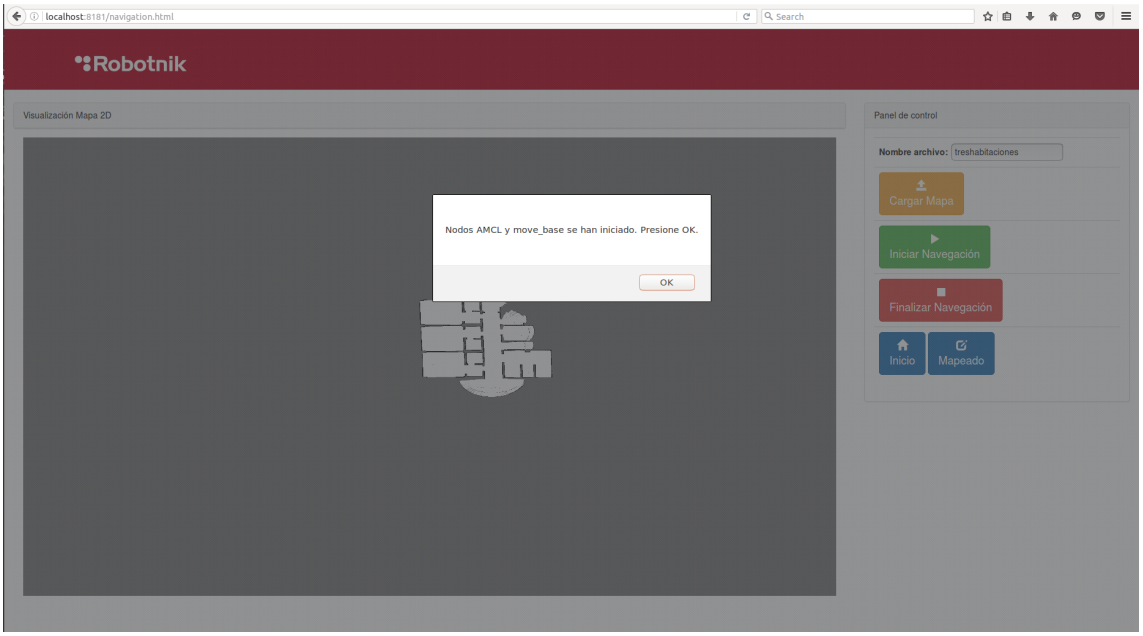

Figura 3.30: Mensaje de advertencia informando al usuario de los nodos de ROS, AMCL y Move Base, se han lanzado

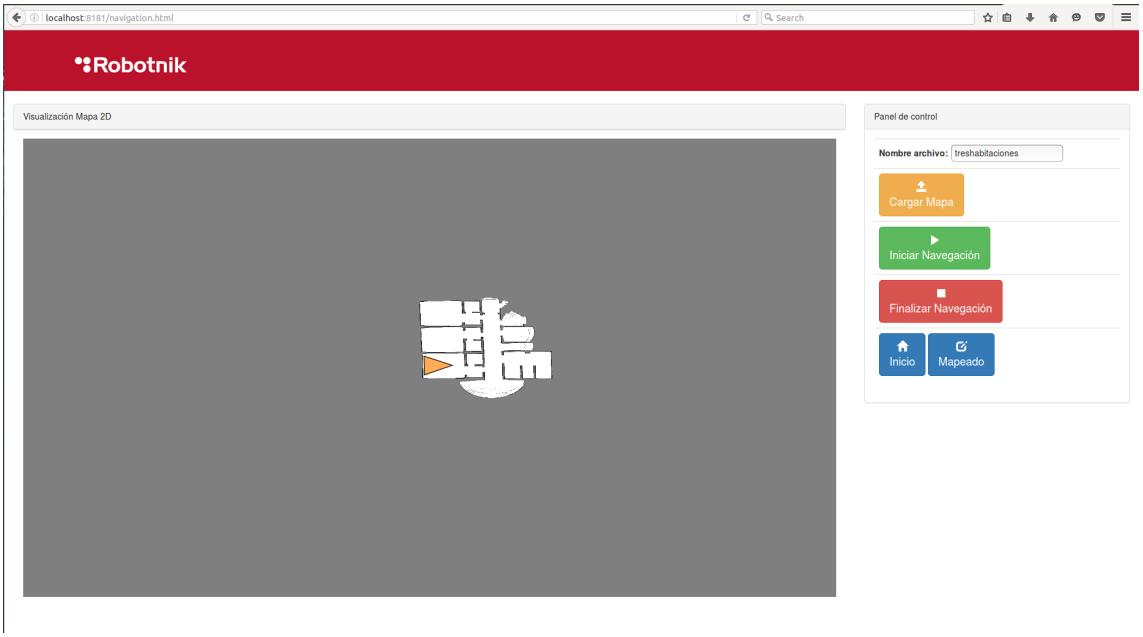

Figura 3.31: Representación del robot mediante un triángulo naranja en la rejilla de navegación

Para especificar un punto de navegación basta con hacer click sobre cualquier punto blanco del mapa. En el mapa aparecerá un triangulo de color rosa que indica el destino al que debe dirigirse el robot, como se puede ver en la figura 3.32. Es posible determinar la orientación del robot en el punto de destino haciendo click y manteniéndolo mientras se mueve el ratón.

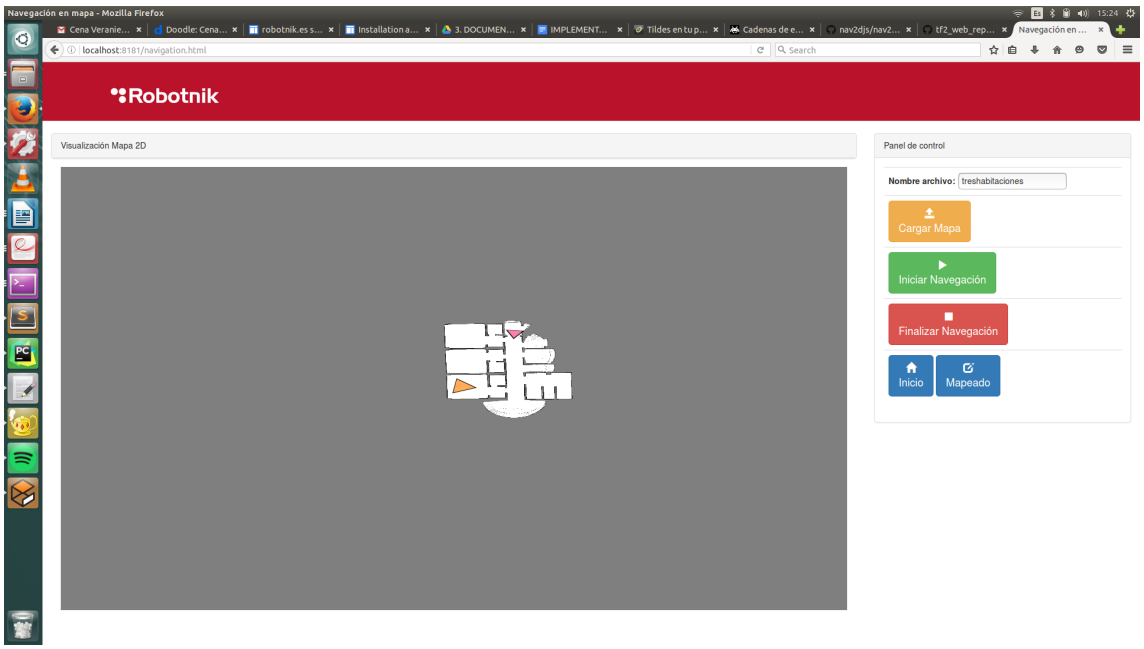

Figura 3.32: Representación del punto de destino mediante un triángulo rosa

Automáticamente el robot inicia la navegación hacia el destino especificado. En la figura 3.34 y 3.33 se puede observar el robot dirigiéndose hacia el objetivo.

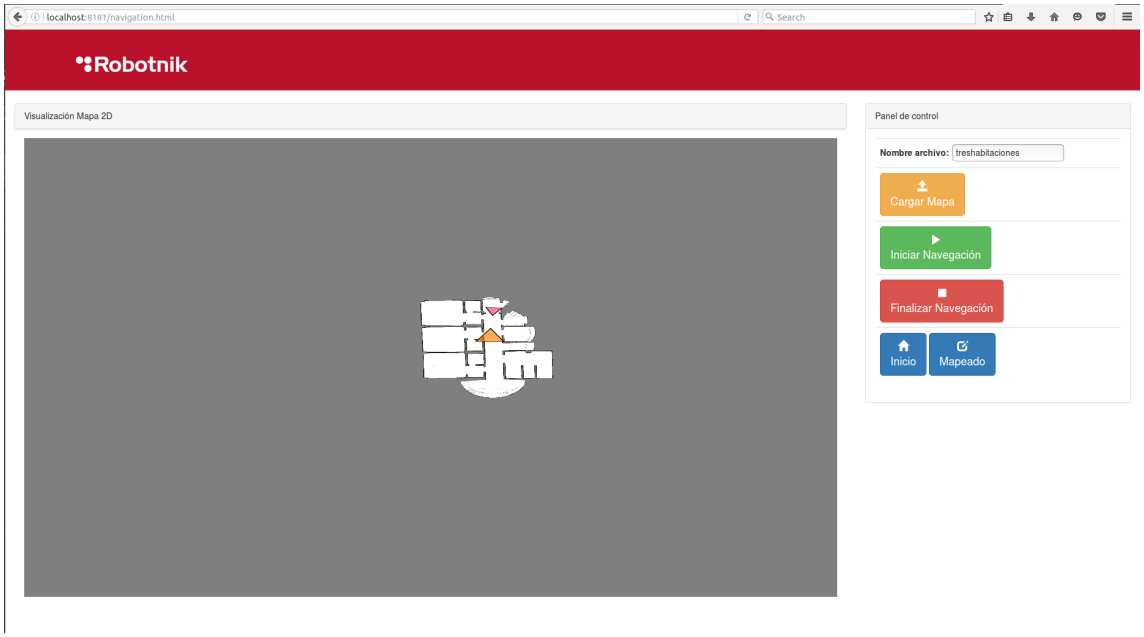

Figura 3.33: Visualización en la interfaz del robot dirigiéndose al destino

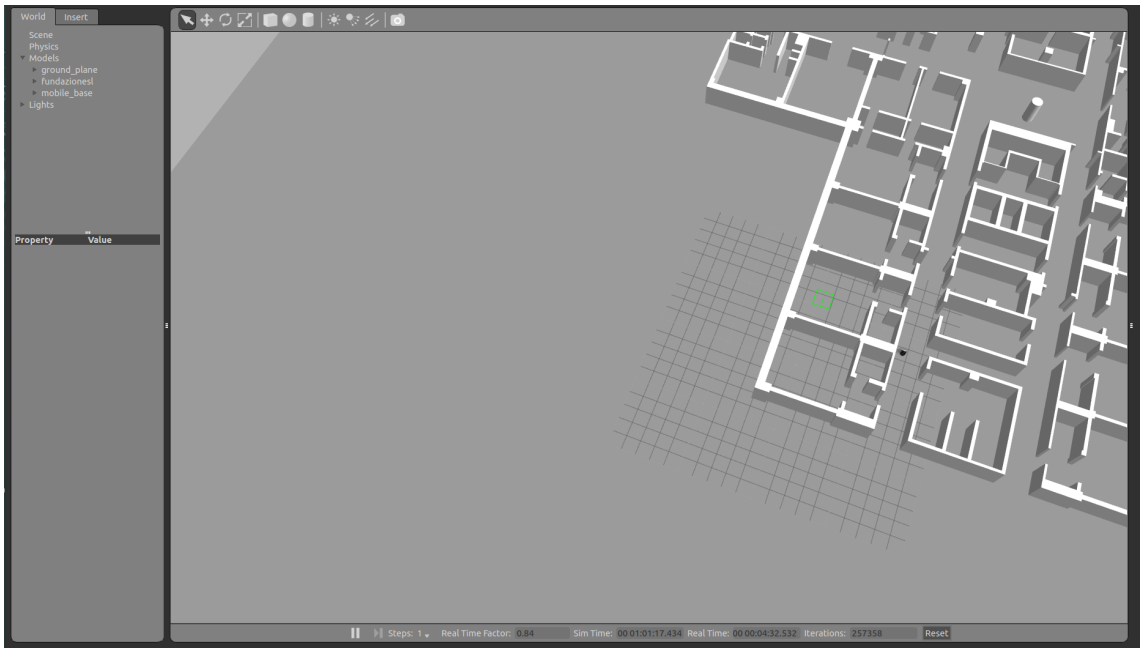

Figura 3.34: Vista en el simulador Gazebo del robot dirigiéndose al destino

Finalmente para terminar la navegación basada en mapa, se pulsa el botón de la interfaz **Finalizar Navegación**. La interfaz informará al usuario que el proceso se ha detenido, como se puede observar en la figura 3.35

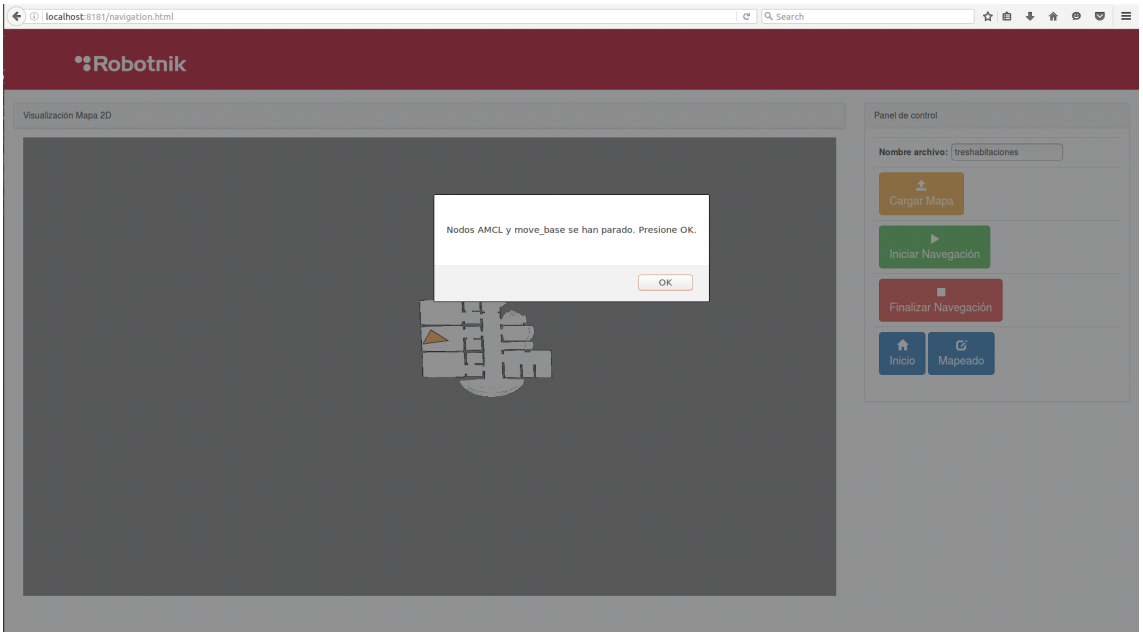

Figura 3.35: Mensaje de advertencia informando al usuario la navegación se ha detenido con éxito

CAPÍTULO 4
Conclusiones

La idea inicial del TFG desarrollado era diseñar e implementar una interfaz gráfica que permita a usuarios no expertos en robótica generar mapas del entorno con un robot y que además permita a los mismos realizar navegación autónoma con el mismo, dentro del ecosistema que ofrece ROS.

El software desarrollado ofrece la solución proporcionando una interfaz gráfica web que permite a los usuarios, por un lado, poner en marcha los algoritmos necesarios para generar un mapa de entorno con un robot, a la vez que les muestra la visualización del mapa y el robot durante el proceso. Por otro lado la interfaz permite al usuario recuperar un mapa y poner en marcha los algoritmos de navegación, ofreciendo en la interfaz una representación del mapa en la cual el usuario puede marcar puntos de navegación. De ese modo se el ofrece al usuario sin conocimientos de ROS, una interfaz sencilla, limpia e intuitiva en la que el usuario no tiene porque conocer los detalles de funcionamiento de los algoritmos que subyacen.

En primer lugar se quería dotar a la comunidad ROS de una interfaz de usuario para la generación de mapas y navegación basada en dicho mapa. Para que una herramienta pueda ser validada por la comunidad ROS tiene que cumplir una serie de estándares y recomendaciones. Esta herramienta necesita ser probada en profundidad y refinada en algunos aspectos.

Otro objetivo que perseguía el trabajo consistía en obtener una herramienta que permitiera al usuario visualizar el robot y el mapa durante el proceso de generación de mapa y durante la navegación. Este objetivo se ha cumplido pues como se ha comentado es exactamente lo que hace la aplicación.

En cuanto a los siguientes objetivos, simplificar el proceso de creación de mapas y simplificar el proceso de navegación basado en mapas para el usuario no experto, también se han conseguido. En este caso, simplificar el proceso ha implicado que el usuario no tiene que conocer con precisión cómo ejecutan los algoritmos que ofrecen la funcionalidad deseada. Aunque la interfaz y el proceso se han intentado implementar lo más simple posibles no tenemos la retroalimentación de pruebas objetivas que lo atestigüen.

El último objetivo del trabajo persigue proporcionar una herramienta que fuera accesible desde cualquier navegador web. Este objetivo se ha podido conseguir sólo en parte, porque depende de que el navegador web ofrezca soporte para HTML5. En el caso de navegadores de escritorio como Firefox o Chrome, la interfaz se visualizaba correctamente, pero en navegadores de algunos dispositivos móviles (tabletas, móviles,etc.) depende del navegador instalado.

A pesar de que ROS, ofrece gran cantidad de atajos y herramientas para desarrollar software para robots, me he encontrado con muchos problemas sobretodo a nivel teórico ya que la robótica es una ciencia difícil que requiere muchas horas de entrenamiento y estudio. Otro problema lo he encontrado a la hora de consultar ciertas bibliotecas de ROS. Algunas de ellas están poco documentadas o la documentación está incompleta, lo cual dificulta el uso de las mismas. Debido a ello gran parte del tiempo empleado en el desarrollo del trabajo ha sido para documentarme y aprender cómo solucionar problemas. Ello ha hecho que no haya podido dedicarle más tiempo a implementar algunas mejoras.

Una vez terminado el proyecto y después de redactar y revisar la memoria escrita he podido darme cuenta de que decisiones tomadas durante las fases previas a la ejecución del proyecto son algunas correctas y otras no tanto, y se podrían tomar diferentes decisiones para realizar nuevas ampliaciones y mejoras del mismo.

Una posible mejora para el futuro es transformar toda la parte de navegación basada en mapa de una visualización 2D a una visualización 3D, de manera que fuera posible poder enviar puntos en el mapa y navegar, similar a como se hace en la herramienta visual de ROS.

Otro aspecto que se puede mejorar de la interfaz es cambiar la manera de guardar y recuperar los mapas. Actualmente se hace mediante un campo de texto que el usuario rellena con el nombre del fichero, lo cual no es intuitivo. Este sistema podría reemplazarse por un sistema de navegación de ficheros que permite seleccionar el mapa del mismo modo que se hace en el explorador de archivos de un sistema operativo.

Futuros trabajos pueden también ir encaminados a mejorar el nodo mánager de ROS que se ha desarrollado para esta aplicación y que gestiona los procesos involucrados en la generación de mapas y navegación. Sería interesante desarrollar uno que fuera más robusto y que pudiera monitorizar el ciclo de vida de los procesos (parado, ejecución, espera,etc.).

Una extensión de más envergadura de este trabajo puede ser implementar un planificador de rutas de manera que desde la misma interfaz web se puedan marcar puntos en el mapa que configuren una ruta y que el robot los pueda seguir.

La realización del presente TFG me ha supuesto un esfuerzo para completar con éxito un proyecto en el que he trabajado durante los meses previos, poniendo en relieve todo los conocimientos que he ido adquiriendo durante los cuatro años académicos y, al mismo tiempo, adquiriendo competencias, conocimientos y habilidades que antes no poseía. Además, en mi caso particular, poder realizar

este proyecto dentro del ambiente de una empresa me ha permitido desarrollar una aplicación que es posible que se aplique en el mundo profesional.

Para finalizar, me siento satisfecho con el trabajo realizado. Este trabajo me ha permitido tomar contacto con el mundo de la robótica y me ha permitido empezar a aprender conceptos y técnicas que se utilizan frecuentemente en este campo. Además me ha permitido adquirir conocimientos de tecnologías web, que hasta este momento no tenía. Estoy seguro de que gracias a este proyecto, el aprendizaje de nuevos conceptos va a ser mucho más rápido y me va a permitir desarrollar un mejor trabajo en el futuro.Trabajar con LaTeX también ha supuesto una complicación añadida, pero he disfrutado mucho ya tiene muchas órdenes útiles y es muy amigable para un informático.

Bibliografía

[1] A. Koubâa. ROS as a service: web service for robot operating system, *Journal of Software Engineering Robotics*, 6(1), 2015.

[2] J. Kramer, M. Scheutz. Development environments for autonomous mobile robots: A survey, *Autonomous Robots*, 22(2), 101-132, 2007.

[3] F. Ellouze, A. Koubâa, H. Youssef. ROS Web Services: A tutorial, *Robot Operating System: A complete reference*, 10, 463-490, 2015.

[4] M. Quigley, K. Conley, B.P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng. ROS: an open-source Robot Operating System, *ICRA Workshop on Open Source Software*, 2009.

[5] R. Toris, J. Kammerl, D. Lu, J. Lee, O.C. Jenkins, S. Osentoski, M. Wills, S. Chernova. Robot Web Tools: Efficient Messaging for Cloud Robotics. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[6] C. Crick, S. Osentoski, G. Jay, O.C. Jenkins. Human and robot perception in large-scale learning from demonstration, *Proceedings of the 6th ACM/IEEE International Conference on Human-Robot Interaction*, 2011.

[7] C. Crick, G.T. Jay, S. Osentoski, B. Pitzer, O.C. Jenkins. Rosbridge: Ros for non-ros users, *International Symposium on Robotics Research (ISRR 2011)*, 2011.

[8] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. DuHadway, O. Jenkins Robots as web services: Reproducible experimentation and application development using rosjs, *International Conference on Robotics and Automation (ICRA 2011)*, 2011.

[9] K. Taylor, J. Trevelyan. A Telerobot On The World Wide Web, *National Conference of the Australian Robot Association*, 1995.

[10] J. Lee. Web Applications for Robots using rosbridge.

[11] D. Schulz, W. Burgard. Robust Visualization for Web-based Control of Mobile Robots, *Robust Visualization for Web-based Control of Mobile Robots*, 2001.

[12] C. Yinong, D. Zhihui, and G.A. Marcos. Robot as a service in cloud computing, *Service Oriented System Engineering (SOSE)*, 1, 2, 2010.

[13] K. Yuka, I. Toru, M. Yoshihiko, O. Keiju, U. Miwa, T. Yosuke, N. Masahiko. Research and development environments for robot services and its implementation *System Integration (SII)*, 2011.

[14] A. Koubaa, E. Maehle, K. Rmer, W. Karl, and E. Tovar. A service-oriented architecture for virtualizing robots in robot-as-a-service clouds *Architecture of Computing Systems ARCS 2014*, 8350, 196–208, 2014.

[15] R. Souza, F. Pinho, L. Olivi, and E. Cardozo. A restful platform for networked robotics, *Ubiquitous Robots and Ambient Intelligence (URAI)*, 423–428. 2, 5.4, 2013.

[16] R. Safaripour, F. Khendek, R. Glitho, and F. Belqasmi. A restfull architecture for enabling rapid development and deployment of companion robot applications, *Computing, Networking and Communications (ICNC)*, 2, 971–976, 2014.

[17] K. Goldberg. *The Robot in the Garden: Telerobotics and Telepistemology in the Age of the Internet*, MIT press, Cambridge, MA, USA, 2001.

[18] L. Joseph. *Mastering ROS for Robotics Programming*, Pack Publishing, Birmingham, UK, 2015.

[19] R. Patrick Goebel. *ROS by example*, Lulu, 2014.

[20] ROS.org History, Consultado en http://www.ros.org/history/.

[21] ROS.org About, Consultado en http://www.ros.org/about-ros/.

[22] ROS.org Metrics 2015, Consultado en http://download.ros.org/downloads/metrics/metrics-report-2015-07.pdf

[23] Introducing rostful: Ros over restful web services, Consultado en http://www.ros.org/news/2014/02/introducing-rostful-ros-over-restful-web-services.html

[24] ROS wiki Consultado en http://wiki.ros.org

APÉNDICE A

Ficheros utilizados en la aplicación

A.1 Definición del servicio Trigger

```
---
bool success
string message
```

Listing A.1: Definición del servicio std_srvs/Trigger

A.2 Definición del servicio SetFilename

```
string name
---
bool ret
string msg
```

Listing A.2: Definición del servicio map_nav_manager/SetFilename

A.3 Fichero servers.launch

```
<?xml version="1.0"?>
<launch>

  <!-- iniciar servidor web -->

  <node name="mini_httpd" pkg="map_nav_manager" type="mini-httpd.sh"
      output="screen" />

  <!-- iniciar rosbridge server -->

  <include file="$(find rosbridge_server)/launch/rosbridge_websocket.
      launch" />

  <!-- iniciar tf2_web_republisher node -->

  <node name="tf2_web_republisher" pkg="tf2_web_republisher" type="
      tf2_web_republisher" output="screen" />

  <!-- iniciar robot_pose_publisher node -->

  <node name="robot_pose_publisher" pkg="robot_pose_publisher" type="
      robot_pose_publisher" output="screen" />

  <!-- iniciar manager ROS -->

  <node name="map_nav_manager" pkg="map_nav_manager" type="
      map_nav_manager_node.py" output="screen" />


</launch>
```

Listing A.3: Fichero server.launch

APÉNDICE B

Puesta en marcha de la aplicación

B.1 Dependencias de paquetes de ROS

- Rosbridge

```
sudo apt-get install ros-indigo-rosbridge-suite
sudo apt-get install ros-indigo-rosbridge-server
sudo apt-get install ros-indigo-rosbridge-library
```

- SLAM-gmapping

```
sudo apt-get install ros-indigo-slam-gmapping
```

- AMCL

```
sudo apt-get install ros-indigo-amcl
```

- MoveBase

```
sudo apt-get install ros-indigo-move-base
sudo apt-get install ros-indigo-move-base-msgs
```

- Robot Pose Publisher:

```
sudo apt-get install ros-indigo-robot-pose-publisher
```

- TF2 Web Republisher

```
sudo apt-get install ros-indigo-tf2-web-republisher
```

B.2 Lanzamiento de la aplicación

- El robot esté ejecutando los controladores en ROS y publicando su información de odometría en el topic /odom.
- El robot tiene que estar equipado con un láser 2D y su nodo controlador debe estar publicando datos en /scan (o namespace/scan).
- Descargar el paquete mav_nap_manager desde el repositorio de Github.

```
https://github.com/JoseRobotnik/map_nav_manager.git
git@github.com:JoseRobotnik/map_nav_manager.git
```

Pasos:

1. Añadir el paquete al directorio de catkin y compilar.
2. Se ejecuta en un terminal el archivo server.launch con la orden:

```
roslaunch map_nav_manager servers.launch
```

3. Se ejecuta en otro terminal el nodo map_nav_manager con la orden:

```
rosrun map_nav_manager map_nav_manager_node.py
```

4. Abrir un navegador con la dirección IP del robot y el puerto 8181 para tener acceso a la interfaz.