UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Herramienta para la creación y edición de rutas en un Sistema de Gestión de Flotas para entornos industriales

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: José Gómez Gadea

Tutor: Vicente Javier Julián Inglada
Román Navarro García

Curso 2018-2019

Resumen

Un Sistema de Gestión de Flotas es un conjunto de herramientas software que permiten el control y coordinación de una flota de robots móviles para el desempeño de una serie de tareas automáticas, tales como el transporte autónomo de carros de mercancía. Este sistema es necesario para que se lleve a cabo una utilización óptima de la flota y de sus recursos compartidos (ascensores, cruces, puertas, pasillos de un solo carril, posiciones de carga y descarga, cargadores de baterías, etc.), pero también para que no se produzcan interbloqueos o inanición.

Una de las partes más importantes de este sistema es la herramienta de creación y edición de rutas y recursos compartidos, que facilite la puesta en marcha y mantenimiento del gestor en cualquier tipo de instalación.

Actualmente la configuración de la ruta se define en forma de grafo, en el que se establecen los puntos de interés así como la interconexión entre ellos, dotando, tanto a los nodos como a sus arcos con información necesaria para la correcta gestión de la flota.

Palabras clave: automatización, flota, coordinación, optimización, gestión, ROS, web

Resum

Un Sistema de Gestió de Flotes es un conjunt de ferramentes software que permeten el control i coordinació de una flota de robots mòbils per a l'acompliment d'una serie de tasques de transport automàtic, tals com el transport autònom de carros de mercaderia. Aquest sistema es necessari per a que es porte a terme una utilització òptima de la flota i dels seus recursos compartits (ascensors, encreuaments, portes, carregadors de bateríes, etc.), però també per a que no es produeixquen interbloquejos o inanició.

Una de les parts més importants d'aquest sistema es la ferramenta de creació i edició de rutes i recursos compartits, que facilite la posada en marxa i manteniment del sistema en qualsevol tipus d'instal·lació.

Actualment la configuració de la ruta es defineix en forma de graf, on s'estableixen els punts d'interés així com la interconexió entre ells, dotant, tant als punts com als seus arcs d'informació necessaria per a la correcta gestió de la flota.

Paraules clau: automatització, flota, coordinació, optimització, gestió, ROS, web

Abstract

A Fleet Management System is a set of software tools that allows the control and coordination of a fleet of mobile robots to perform a group of tasks automatically, such as the autonomous transport of carts full of materials. This system is required to make an optimum utilization of the fleet and its resources (elevators, doors, load and unload positions, battery chargers, etc.), but also to avoid deadlocks or starvation.

One of the most important part of this system is the tool for creating and modifying routes and shared resources, which helps with the start-up and maintenance of the system in any kind of installation.

Currently, the route configuration is defined as a graph, where the points of interest as well as the interconnection between them are stablished, providing to the points and their arcs necessary information for the correct management of the fleet.

Key words: automation, fleet, coordination, optimization, management, ROS, web

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

El gran crecimiento de las empresas tecnológicas ha generado la necesidad de utilizar flotas de robots que trabajen de forma conjunta para obtener un mismo propósito. En estos entornos controlados se puede observar cómo, a medida que pasa el tiempo, se va incrementando la cantidad de autómatas que se encargan de tareas cada vez más complejas.

Hoy en día, en la robótica móvil existen algoritmos muy robustos para la planificación de rutas y tareas de forma óptima. La aplicación de estos en un entorno industrial cambiante no es tan simple y lleva mucho esfuerzo. Además, cuando esta aplicación se realiza sobre sistemas que envuelven un conjunto de distintos robots autónomos, con una infinidad de datos que ofrecen sus sensores, aumenta todavía más la complejidad [1].

1.1 Motivación

En el siglo XIX se produce un gran auge de los autómatas y se desarrollan importantes avances en todas las ramas de la ingeniería. Desde que la automatización ha sido estandarizada en la industria a lo largo del siglo XX [4], se ha trabajado constantemente en el desarrollo de nuevas tecnologías que permitan su implementación de forma sencilla y de una manera más rápida. Todo esto, junto con las mejoras en el propio funcionamiento del robot, se ha convertido en algo imprescindible para mantenerse a la delantera en el complejo mundo de la robótica.

Una de las áreas más demandadas junto con la automatización de grandes compañías ha sido la de la robótica móvil. En este campo, se requiere de complejos sistemas que sean capaces de administrar el funcionamiento de forma eficiente de una gran flota de robots. Aunque los actuales Sistemas de Gestión de Flotas han evolucionado a un ritmo sorprendente, se ha podido observar el alto coste temporal que conlleva su adaptación a las distintas instalaciones. Para ello se propone, entre otras mejoras, la creación de una herramienta capaz de modelar de forma gráfica e intuitiva las rutas y acciones a seguir por la flota de robots (incluso mejorando la eficiencia de dichas acciones para optimizar el funcionamiento de los robots).

Para ello, se cuenta además con el apoyo de Robotnik[1], una compañía valenciana que ofrece servicios de ingeniería e I+D de alta calidad dentro del área de la robótica en el ámbito nacional e internacional.

[1] La empresa Robotnik https://www.robotnik.es

1.2 Objetivos

El objetivo principal del proyecto, llevado a cabo en Robotnik, reside en el diseño de una interfaz gráfica que se pueda acoplar a la web de control del sistema de gestión de flotas, actualmente en desarrollo. En dicha interfaz, el usuario debe ser capaz de ver el grafo del sistema, que contiene la actual configuración de misiones y rutas. También debe ser posible añadir nuevas rutas y editarlas, así como añadir nuevas misiones o eliminar alguna ya existente. Más adelante se profundiza en la función del grafo y su estructura.

Para poder desacoplar esta interfaz del resto del sistema de gestión de flotas, se pretende modificar uno de los nodos del propio gestor (llamado *fms_routes*[2]) para que sea el único encargado de comunicarse con la interfaz (y así que pueda hacer de intermediario). Con la modificación de este nodo también se pretende minimizar el número de nodos necesarios para el funcionamiento del sistema.

Además, se pretende que la interfaz no sea la única forma de modificar el grafo. Para ello, se ha establecido también como objetivo que todas las funciones que se puedan realizar desde la interfaz se puedan realizar también mediante comandos de ROS[3].

Como otra posible forma de editar el grafo, se puede modificar el archivo que el *fms_routes* carga al iniciar. Para ello se pretende investigar sobre los diversos formatos compatibles con este propósito, y utilizar aquel que sea lo más eficiente y comprensible por el usuario.

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

1. Comprobar la viabilidad de las diversas opciones para el diseño de la web.
2. Utilizar una estructura de datos óptima para almacenar el grafo en memoria.
3. Programar una comunicación entre la interfaz gráfica y el sistema de gestión de flotas.
4. Desarrollar las partes web que nos permitan de forma sencilla visualizar y modificar el grafo.
5. Integrar en el sistema de gestión de flotas la nueva metodología y validar todas las funciones desarrolladas.

1.3 Estructura de la memoria

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

A continuación, en el segundo capítulo se presenta el panorama actual de la empresa para la que va dirigida este proyecto y como se procederá a mejorar esta situación. Se presenta además otra tecnología con la cual evitar la costosa generación de grafos y los motivos por los cuales es descartada.

[2]Nodo *fms_routes*: Parte del sistema de gestión de flotas que se encarga de cargar el grafo, modificarlo y calcular las rutas mediante una variación del algoritmo Dijkstra.

[3]*Robot Operating System*: Framework para el desarrollo de software para robots, para más detalle se puede consultar el apartado 4.3.1.

En el tercer capítulo se analiza la problemática actual y se descartan las opciones menos válidas tanto por temas de eficiencia como por complejidad. También se analizará la solución final propuesta.

Seguidamente, en el capítulo cuatro se comentan los diversos formatos utilizados. Además, se habla acerca de la estructura que sigue el proyecto y las comunicaciones entre los distintos nodos.

En el quinto capítulo se describe la forma que tienen de trabajar cada uno de los nodos de ROS y las acciones que realizan. También se hace una descripción de la estructura de la base de datos. Se comentan además funcionalidades útiles añadidas.

El capítulo seis describe las partes software tanto del servidor como del propio robot necesarias para comprender el completo funcionamiento del sistema de gestión de flotas. Se comenta el proceso de instalación y configuración de cada una.

Como capítulo siete se muestran las distintas pruebas realizadas con el nuevo sistema de creación de grafos y el *feedback* aportado por los dos usuarios que han probado la interfaz gráfica.

Para finalizar, concluimos el proyecto en el capítulo ocho y tratamos los trabajos futuros que permitirán ampliar y complementar las funcionalidades de la aplicación en el capítulo nueve.

CAPÍTULO 2
Estado del arte

La demanda de la robótica móvil crece de forma exponencial gracias sobre todo a la utilización de los robots en un entorno de trabajo donde se requiere de un constante transporte de materiales y productos. En estos entornos, raramente nos encontramos con un único robot [1]. Las agrupaciones o flotas de robots son las distribuciones más utilizadas en estos casos.

2.1 Sistemas de Gestión de Flotas

La gestión de flotas es la administración y logística de un conjunto de vehículos de una organización. En ésta se pueden incluir diversas funciones; como la gestión de los vehículos, el control de sistemas pertenecientes al entorno e incluso la prioridad de las acciones a realizar por los robots [2].

Un Sistema de Gestión de Flotas (SGF), en inglés *Fleet Management System* (FMS), consiste en un servicio informático que nos permite realizar dicha gestión y, en lo referente al trabajo actual, la gestión de los robots y su entorno.

En nuestro caso de uso se utiliza un sistema centralizado, que consiste en un ordenador que realiza la función de servidor y un router al cual se conectan todos los robots el cual realiza la comunicación con dicho servidor.

El funcionamiento del sistema de gestión de flotas en el caso de Robotnik comienza con la petición de las tareas a realizar, bien mediante una web a la cual el cliente debe tener acceso o bien mediante un sistema automatizado de elección de tareas en función a diversos sensores. Tanto el *feedback* de estos sensores como los distintos recursos de los robots son especificados dentro de un grafo. Cada nodo de dicho grafo representa una posición en el mapa sobre el cual se mueven los robots, por lo que si se envía una tarea al robot, éste es capaz de llegar recorriendo el grafo desde su posición actual hasta el nodo destino en el que se encuentra la tarea a realizar. Los puntos del grafo contienen información sobre qué tipo de nodo es. En función de que tipo de

nodo sea y de la configuración aplicada a nivel de programación se definen las acciones que se realizarán en una situación de paso por parte del robot (que pueden consistir en la apertura de puertas, limitar su velocidad, etc.).

La programación de este grafo con las características del nodo y las acciones que puede o debe realizar consiste en la edición de un archivo en formato XML[1]. Este archivo es leído cada vez que se inicializa el sistema.

A continuación podemos observar un ejemplo de archivo en formato XML utilizado, en el cual hay definidos 2 nodos. Ambos tienen un arco que apunta al otro nodo respectivamente (lo cual quiere decir que hay una ruta para ir de uno al otro):

```
<CGraph> <!-- Inicio del grafo -->
   <m_NodeList> <!-- Inicio de lista de nodos -->
      <CNode> <!-- Inicio nodo 0 -->
         <m_iNumber>0</m_iNumber>
         <!-- Definimos su posición en el mapa -->
         <m_Position>
            <X>1.2</X>
            <Y>4.3</Y>
            <Z>0.0</Z>
            <!-- Orientación -->
            <Theta>0.0</Theta>
            <!-- Mapa al cual pertenece el nodo -->
            <Frame>map</Frame>
         </m_Position>
         <!-- Nodo de carga de batería -->
         <m_bCharge>1</m_bCharge>
         <!-- Nombre del nodo -->
         <m_sName>Node docking 0</m_sName>
         <m_ArcList> <!-- Inicio lista de nodos destino -->
            <CArc> <!-- Arista hacia nodo 1 -->
               <m_iNodeDst>1</m_iNodeDst>
            </CArc>
         </m_ArcList>
      </CNode> <!-- Fin nodo -->
      <CNode>
         <m_iNumber>1</m_iNumber>
         <m_Position>
            <X>3.0</X>
            <Y>4.3</Y>
            <Z>0.0</Z>
            <Theta>1.57</Theta>
            <Frame>map</Frame>
         </m_Position>
         <!-- No es un nodo de paso (es nodo hoja)-->
         <m_bStop>1</m_bStop>
         <!-- Es un nodo de carga de carro -->
         <m_bLoad>1</m_bLoad>
         <m_sName>Node pick 1</m_sName>
         <m_ArcList>
         <CArc>
            <m_iNodeDst>0</m_iNodeDst>
         </CArc>
         </m_ArcList>
      </CNode>
   </m_NodeList>
</CGraph>
```

Ejemplo 2.1: Ejemplo de dos nodos en XML

[1]Lenguaje de etiquetado XML https://en.wikipedia.org/wiki/XML

Se ha realizado un estudio sobre el estado del arte en otras empresas así como a nivel general. Aunque actualmente existen sistemas de gestión de flotas muy avanzados, la mayoría de ellos no están integrados con ROS. Además, de los pocos que sí tienen integración con ROS, ninguno es de software libre ni se trata de software que esté a la venta. Sí que hay diversos algoritmos que se pueden utilizar, tales como el algoritmo de *Dijkstra*[2], que permiten gestionar el funcionamiento de los robots de forma óptima y eficiente [2].

La funcionalidad de visualizar datos georreferenciados es bastante común en muchos entornos gráficos que operan sobre sistemas de gestión de flotas. Esto se debe a que una gran parte del proceso de gestión consiste en modificar acciones del sistema que se encuentran en unas coordenadas concretas. Existen diversas librerías de software libre que permiten el desarrollo de una interfaz capaz de modificar grafos con nodos en posiciones concretas. También las hay para visualizar mapas y localizaciones en las cuales observar sus acciones asignadas [3].

2.2 Problemática

Prácticamente desde los orígenes de la empresa en 2002, se ha estado trabajando en el desarrollo de un sistema de gestión de flotas capaz de optimizar el funcionamiento de diversos robots. Las primeras versiones fueron pensadas como una solución primeriza al problema.

El problema se debe principalmente a la ampliación del sistema realizada a lo largo de estos años que, dada su estructura inicial, suma complejidad a la ya de por sí farragosa programación del grafo. Por tanto, se trata de una estructura que, además de confusa, no es nada eficiente a la hora de implementar la comunicación con alguna herramienta de gestión de grafos interactiva.

Además, cada nuevo tipo de nodo añadido requiere que se modifiquen muchas secciones del código, lo cual es poco práctico dado que constantemente se están añadiendo tipos que cumplan las posibles nuevas acciones.

Esta gran variedad de tipos de nodos también suman complejidad a la definición del grafo en el XML, que se ha convertido en todo un quebradero de cabeza debido al diseño poco intuitivo y visual que éste ofrece. Además, la lectura del fichero es ineficiente, llegando a ralentizar notablemente la carga del sistema de gestión de robots en instalaciones con una gran cantidad de nodos.

Sumada a la poca optimización de carga del grafo, se da la situación de tener muchos nodos distintos en una misma ubicación debido a la forma en la que se definen estos nodos y sus acciones acopladas.

2.3 Propuesta

Como medida principal se busca desarrollar una herramienta interactiva con la que se pueda programar un grafo de forma sencilla hasta el punto de que cualquier usuario, mediante una pequeña explicación, sea capaz de utilizarla.

Se pretende realizar la implementación de diversos servicios del *Framework* ROS en el nodo *fms_routes* para que, mediante ellos, sea posible la completa edición del grafo.

[2] Algoritmo de Dijkstra: Consiste en un algoritmo cuya finalidad es la de determinar cual es el camino más corto en un grafo que tiene pesos en cada arista (en nuestro caso de uso, estos pesos son las distancias entre los nodos de dichas aristas).

La herramienta gráfica debe ser capaz de funcionar únicamente haciendo uso de estos servicios.

Se busca realizar una modificación de la estructura de datos para evitar el mínimo número de cambios necesarios en el código al agregar una nueva acción.

El paso de XML a otro formato es otra propuesta bastante deseada. Se pretende buscar un formato que sea menos confuso, más eficiente, y que permita que una ampliación de las acciones de los nodos no requiera de una modificación de su serializador[3].

En último lugar y como posible modificación para aumentar la eficiencia, se propone preparar el nodo *fms_routes* para almacenar todas las acciones del grafo (para que configurarlas no requiera de grandes conocimientos de programación) y tratarlas desde fuera, permitiendo aplicar varias acciones a un mismo nodo en función al contexto actual.

[3]Serialización: La serialización consiste en un proceso de codificación de un objeto en un medio de almacenamiento. Estos datos almacenados pueden utilizarse para crear un nuevo objeto que es idéntico en todo al original.

CAPÍTULO 3
Análisis y propuesta

En Robotnik nos encontramos casi desde sus comienzos con un sistema funcional de gestión de flotas pero ineficiente, que causa la pérdida de muchas horas y dinero en su configuración debido a su complejidad. En este capítulo se comentan los presentes problemas y sus posibles soluciones, así como la propuesta final y una justificación acerca de por qué ha sido escogida.

3.1 Análisis del problema

Como se ha comentado anteriormente en el apartado 2.1, la definición de un grafo es bastante compleja dado que solo se puede programar mediante la edición del XML, comprobando las coordenadas de la ubicación donde se desee ubicar cada nodo desde RViz[1]. Podemos visualizar este proceso a continuación:

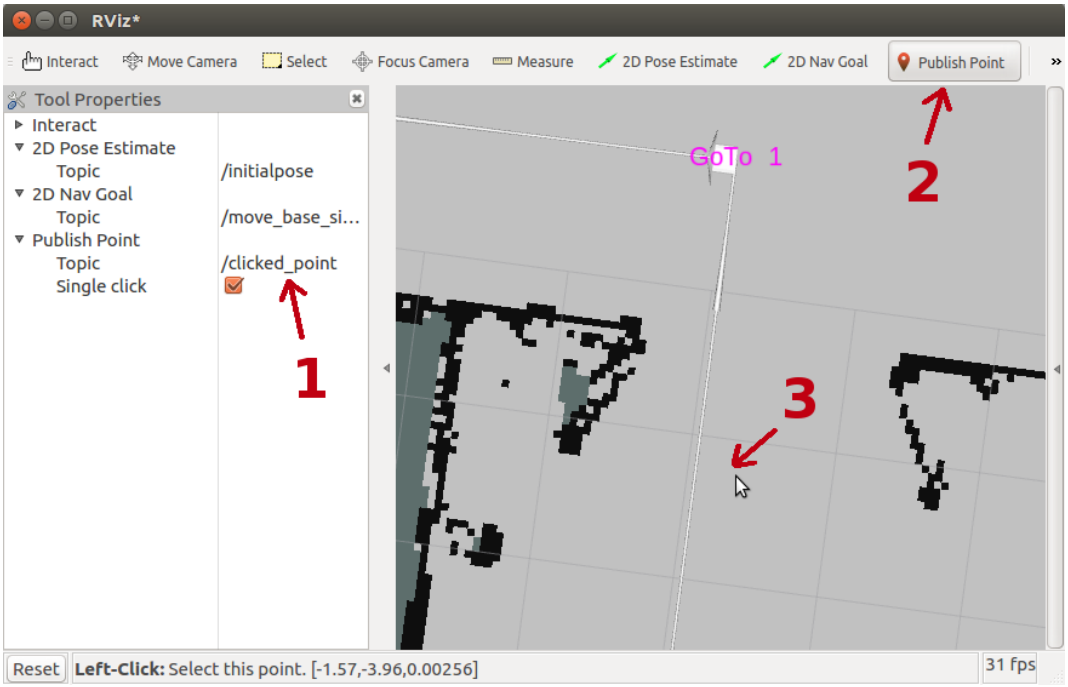

Figura 3.1: Selección de punto en RViz

[1]RViz: Se trata de un programa diseñado para desarrolladores de ROS que permite la visualización de todos los datos recibidos por los sensores del robot e incluso permite la publicación de comandos mediante el uso de sus interfaces.

Es necesario abrir el programa RViz (figura 3.1) habiendo cargado previamente el mapa en ROS. A partir de aquí se han de seguir los siguientes pasos:

1. Escribir el nombre del topic[2] en el cual se quiere recibir la posición seleccionada.
2. Seleccionar la herramienta "*Publish Point*", que viene por defecto ya instalada en el programa. También se puede utilizar la herramienta "*2D Pose Estimate*", que además nos retorna la orientación.
3. Seleccionar sobre el punto deseado. En caso de haber seleccionado la herramienta que retorna la orientación, habrá que mover el ratón hacia la posición a la que se desea apuntar.

Una vez seleccionado el punto, se puede ver la posición de éste mediante la lectura de los datos del topic a través de un terminal, como se puede observar en la figura 3.2. Estos datos de posición deben ser especificados en el XML para el nodo correspondiente, que dada la confusa estructura vista en el ejemplo 2.1, es muy fácil confundir.

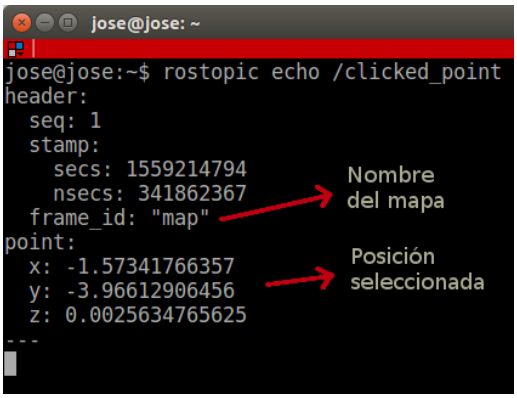

Figura 3.2: Captura del punto desde el terminal.

Además, aunque en el ejemplo solo se muestran un par de tipos de nodo, hay decenas de tipos según la instalación en la cual se quiera configurar. Esto se une a la problemática actual, debido a la forma en la cual funciona el gestor de rutas, que fuerza la creación de diversos nodos para una misma acción (la mayoría de los cuales son nodos lógicos, que no requieren ubicación). En la figura 3.3 podemos observar como ejemplo de esta problemática un grafo utilizado para una instalación en la que hay 49 carros que se tienen que transportar entre 37 líneas. Los nodos que se encuentran a distinto nivel del mapa 2D son nodos lógicos, sobre los cuales no importa la posición.

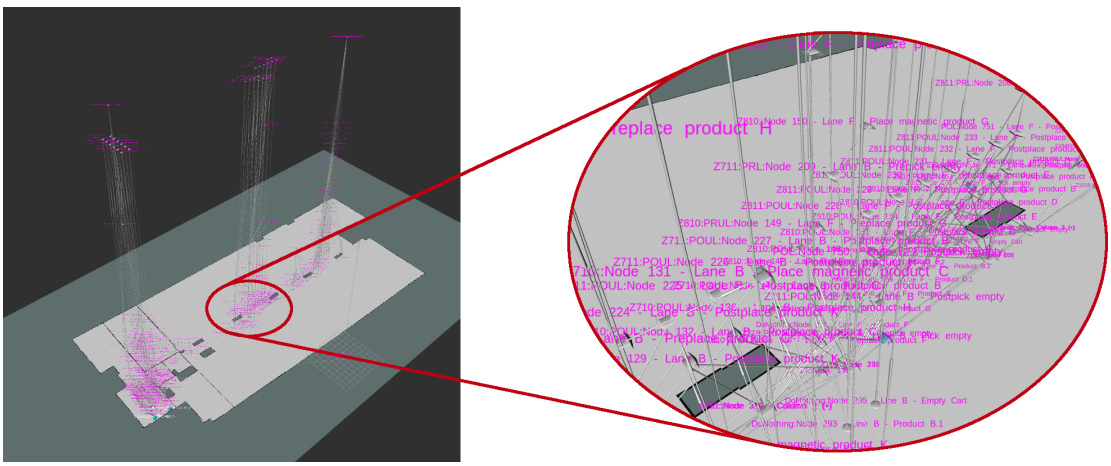

Figura 3.3: Grafo de ejemplo del proyecto RB2

Como podemos observar, la cantidad de nodos es tal que los nombres asignados a estos impiden prácticamente verlos.

[2]Topic: Bus de datos sobre el cual los nodos de ROS intercambian mensajes.

Esta complejidad a la hora de programar el grafo provoca que sea fácil cometer errores muy difíciles de depurar. Esto a largo plazo acaba repercutiendo en una gran pérdida económica para la empresa, ya que es habitual tener que pasar varios días extra en la empresa del cliente como derivado de los problemas con el grafo. Cada uno de estos días pueden suponer fácilmente un coste mínimo de 200 € debido al precio de la estancia o el transporte a la empresa del cliente, dietas de trabajadores y otros gastos derivados.

Es algo muy común que, al empezar una preparación en una nueva instalación industrial, se requiera añadir nuevos tipos de nodos que realicen acciones que anteriormente no estaban disponibles debido a que en estos proyectos previos no eran necesarias. Por cómo está programado el código es necesaria la modificación de 8 ficheros distintos, modificando la programación de varias líneas de código para añadir la compatibilidad a dicha acción. Esto, aparte del tiempo que requiere, hace que sea necesario estar modificando o ampliando el núcleo básico del gestor de rutas constantemente. En la figura 3.4 podemos observar todas las clases que hay que modificar y sus métodos afectados.

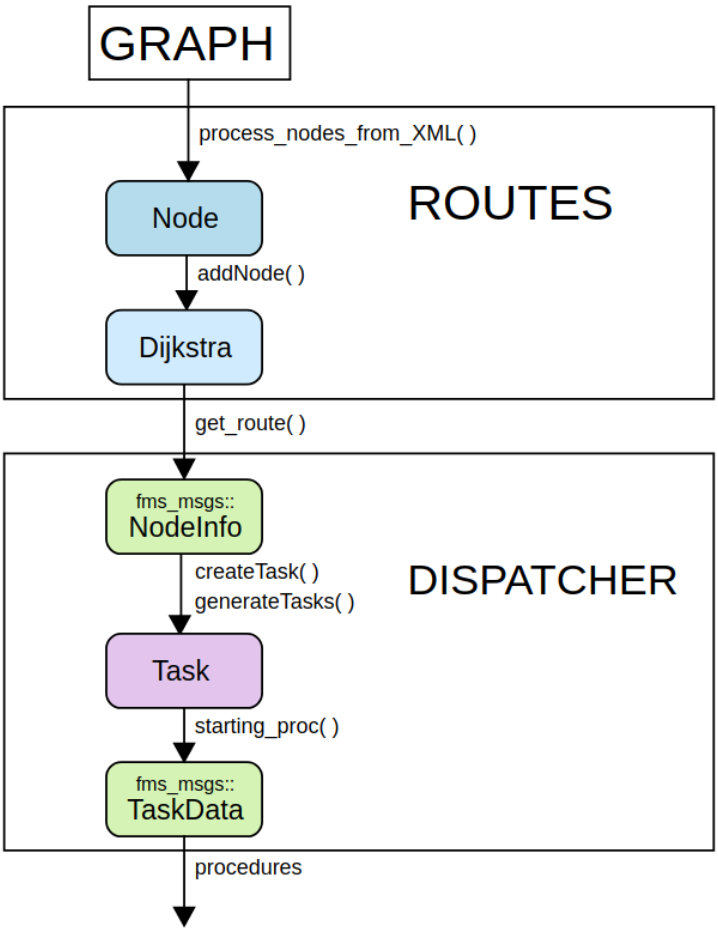

Figura 3.4: Proceso de creación de nuevo tipo de nodo.

Como se puede observar en el esquema, debido al serializado y deserializado de XML utilizado (Xerces[3]), al modificar la estructura del nodo se debe modificar también el método que se encarga de cargar y guardar el grafo. Al tratarse de una estructura tan extensa debido a su mal diseño, no es una modificación sencilla. Además, por esta misma complejidad, tenemos el problema ya comentado de que la carga del fichero es ineficiente, llegando a ralentizar la carga del sistema un tiempo considerable.

[3]Xerces: Se trata de una colección de bibliotecas para el análisis sintáctico, validación, serialización y manipulación de documentos XML de la Apache Software Foundation [10].

3.2 Identificación y análisis de posibles soluciones

Como posibles opciones a la hora de mostrar el grafo se han analizado diversas alternativas. El objetivo es el de mostrar una interfaz lo más accesible posible y que no requiera su instalación en un ordenador ajeno al sistema. Como medida de seguridad, dicho ordenador deberá estar conectado a la misma red Wi-Fi.

Dado que no se busca realizar ningún tipo de preparación sobre el sistema cliente, se ha descartado inmediatamente todo aquello que se trate de instalables. Además, se ha descartado código ejecutable como Java y derivados debido a que también requiere una instalación previa de la máquina virtual, así como de descargar previamente el archivo a ejecutar. Otra opción, ya propuesta anteriormente, es la posibilidad de crear una herramienta de RViz. Esta última opción se llegó a comenzar y existe un creador de nodos para RViz diseñado en Robotnik por el propio autor de este trabajo. Esta herramienta añade una plantilla del nodo al fichero XML que define el grafo, con su posición ya definida en función a lo que hayamos seleccionado desde RViz. El proyecto se detuvo dada la limitada API que ofrece RViz y la complicación que esto suponía.

Por estas causas, sumado a la comodidad de mantener todo el control del sistema de gestión de flotas unificado en un mismo sitio, se ha llegado a la conclusión de que la mejor opción es diseñar una herramienta gráfica que se pueda acoplar a la web del sistema de gestión de flotas en desarrollo. Para ello se ha decidido utilizar HTML 5 con *Canvas*, que permite dibujar gráficos mediante JavaScript (comentado en el apartado 4.3.2).

El HTML 5 es la última versión del popular lenguaje de etiquetado. El *Canvas* es una parte de HTML 5 que nos permite redibujar constantemente un mapa de bits controlado mediante JavaScript a través de las llamadas que ofrece su API [5]. Las ventajas que ofrece HTML 5 con *Canvas* respecto a otros sistemas con la misma finalidad como flash[4] es que necesita muchos menos recursos para ejecutarse y no requiere de un *plug-in* en el navegador. Además, es libre y abierto, por lo que no se requiere de ninguna licencia.

Se han tenido en cuenta diversas librerías JavaScript para el diseño de la interfaz, las cuales se describen a continuación:

- Librería *ros3djs*: La librería *ros3djs* de JavaScript es la que se utiliza actualmente para mostrar el grafo desde la web del FMS. Ofrece una interfaz 3D que permite visualizar un mapa desde cualquier ángulo. Esta interfaz 3D para la finalidad buscada es innecesaria, ya que dificulta la visualización y empeora el rendimiento de forma bastante acusada. En la figura 3.6 podemos observar el uso continuo que hace de la CPU esta librería.
- Librería *ros2djs*: Esta librería de JavaScript es más eficiente dado que simplemente carga un mapa en 2D. Aunque tiene una API bastante reducida, permite cargar el mapa de ROS en un *Canvas* de forma óptima. No tiene soporte para mostrar los puntos de un grafo con el formato deseado.
- Librería *visjs*: Es una librería de visualización de grandes cantidades de datos dinámicos, que permite la manipulación de estos. Entre otras formas de visualización, hay una que permite mostrar grafos. Esto puede ser útil ya que, además, nos permite establecer los nodos del grafo en una posición concreta en el *Canvas* (por lo que el nodo puede ser ubicado en la localización a la que pertenece en caso de poder cargar de fondo una representación del mapa).

[4]Flash: Plataforma software multimedia utilizada para producir animaciones, aplicaciones web interactivas, juegos, etc. Prácticamente se encuentra en desuso debido a la aparición de otros estándares más modernos. [9]

Figura 3.5: *Canvas* con librería *ros3djs*.

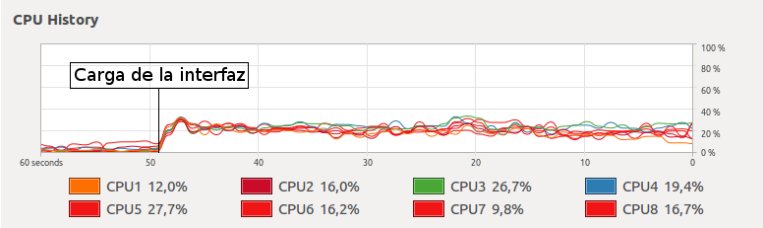

Figura 3.6: Rendimiento de la librería *ros3djs*.

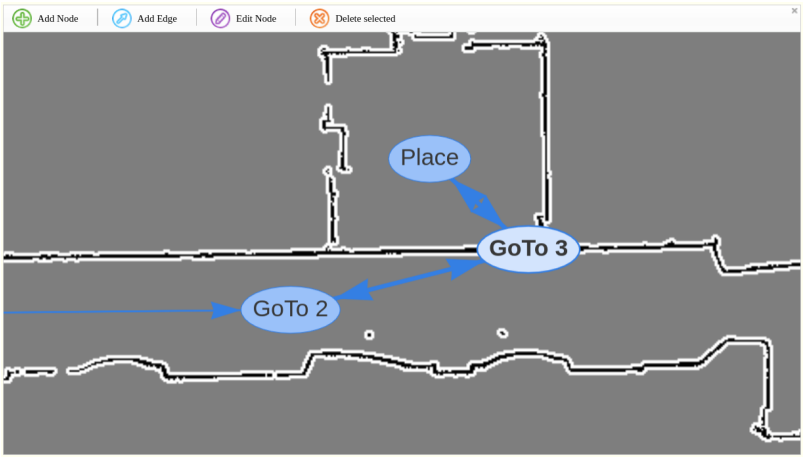

Figura 3.7: *Canvas* con librerías *ros2djs* y *visjs* juntas.

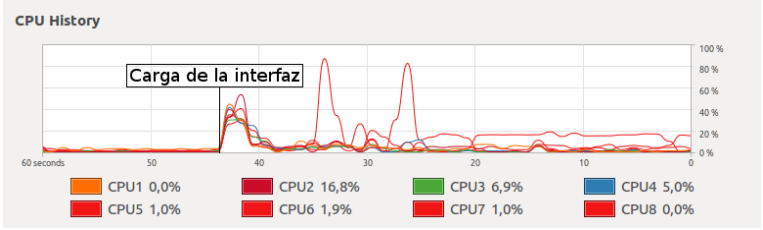

Figura 3.8: Rendimiento de las librería *ros2djs* y *visjs* juntas.

Ambas opciones comentadas anteriormente (figura 3.5 y figura 3.7) son válidas como posible interfaz de visualización y creación de grafos. Como se puede observar en las dos gráficas de rendimiento de las figuras 3.6 y 3.8, la librería *ros2djs* junto con *visjs* ofrece un mejor rendimiento. Analizando la segunda gráfica se puede observar cómo se da un pico de uso de CPU con la carga del grafo, pero una vez cargado mantiene un consumo muy reducido. En la primera gráfica en cambio se mantiene siempre un consumo demasiado elevado, lo cual es problemático sobre todo en entornos donde la cantidad de nodos es muy elevada o el mapa es muy grande (por lo que debe cargar un entorno 3D mayor).

Anteriormente se han comentado varios de los problemas que se dan debido a la forma en la que se carga el grafo, así como por el formato utilizado para guardar los datos a cargar. Por esta razón se han tenido que barajar diversas opciones como alternativas para almacenar el grafo. Las más idóneas han sido:

- Seguir utilizando XML: Una posible opción es dejar de utilizar la librería Xerces (comentada anteriormente) y utilizar otra librería más eficiente. Se han encontrado varias opciones como *RapidXML* o *PlugiXML*, por lo que es una opción viable. En el ejemplo 2.1 ya se ha podido observar cómo sería la definición de un grafo de dos nodos.

- Utilizar JSON: A diferencia de XML, JSON ofrece compatibilidad nativa con vectores y es más sencillo de leer y modificar. No soporta diversos tipos de datos como imágenes, gráficos, etc. En el ejemplo 3.1 se puede observar cómo sería la definición del mismo grafo en JSON.

- Utilizar conversión a bytes: Esta opción es la más eficiente de todas, ya que en un mismo grafo es la que menos espacio ocupa en memoria y realiza una carga más eficiente. El problema de esta opción es que es inviable la lectura del grafo por un humano solo con un editor de ficheros (y su edición es menos viable todavía).

3.3 Solución propuesta

Finalmente se toma como mejor opción la utilización de la librería *visjs* junto con *ros2djs* debido a su eficiencia, sencillez de utilización y facilidad de programación comentadas anteriormente. Es necesario modificar la librería *ros2djs* para poder utilizarla junto con *visjs*. Al tratarse de una librería muy sencilla y limitada, no es muy problemática esta modificación.

Para evitar la necesidad de utilizar tantos nodos en un mismo punto, se pretende modificar la forma en la que se trabaja con el grafo. La problemática actual se debe a que las acciones se definen a nivel de *fms_routes*, que se configura para actuar según el tipo de nodo en que se encuentre actualmente el robot. En la figura 7.9 podemos observar un ejemplo del número de nodos necesarios para poner dos líneas con tres carros que el robot sea capaz de coger y dejar según sus necesidades.

La propuesta actual consiste en poder implementar todas las posibles acciones de modo que si se ubican en la misma posición pertenezcan al mismo nodo. Para ello se desea incorporar en la base de datos del nodo un vector con todas las acciones que puede ejecutar, y las condiciones que se han de cumplir para ejecutarlas. Con esta modificación se espera reducir significativamente el número de nodos necesarios en el grafo, como se puede observar en la figura 7.10.

También se quiere cambiar el tipo de la base de datos, junto con su respectiva modificación en el código, para que en lugar de trabajar con un tipo de dato de C++ compartido

entre los módulos del programa se pueda trabajar con mensajes de ROS. Estos mensajes consisten en un tipo de dato que se genera al compilar con catkin, en caso de estar definidos en el archivo correspondiente (más información en el apartado 4.3.1) [8]. Esta es la mejor alternativa dado que simplemente con la adición de nuevos campos en el mensaje (externo al código), se podría añadir una nueva acción a los nodos.

Como protocolo de comunicación se pretende utilizar *rosbridge* [11], que ofrece un servidor *WebSocket* para que las interfaces web puedan comunicarse con ROS. La intención con la interfaz es que sea complementaria, pero no necesaria. Por este motivo se quiere que toda la edición del grafo se pueda realizar mediante comandos de ROS. La decisión de escoger *rosbridge* es esta misma, que la web utilice los servicios que ROS ofrece para modificar el grafo.

Finalmente se ha decidido escoger JSON como formato para almacenar el grafo por la existencia de una librería desarrollada íntegramente en Robotnik, llamada *jsonization*[5]. Esta librería tiene la función de, al ser compilada, modificar el método de serialización aplicado a cada uno de los mensajes de ROS. Así, si añadimos nuevos tipos de acciones o campos a los nodos, solo hay que modificar la propia estructura definida en los mensajes y recompilar. Con esto nos ahorramos la problemática mostrada en la figura 3.4. Además, como ya se ha comentado anteriormente, un fichero JSON es más fácil de leer y editar por un humano que el resto de alternativas ofrecidas, dada su estructura fácilmente comprensible. También tiene soporte nativo de vectores (a diferencia de XML).

[5] *jsonization*: Librería de ROS desarrollada por Robotnik Automation que permite serializar y deserializar mensajes de ROS a un fichero JSON.

```
{
    "nodes": [
        {
            "id": "0",
            "name": "Node docking 0",
            "pose": {
                "x": 1.2,
                "y": 4.3,
                "z": 0.0,
                "theta": 0.0,
                "frame_id": "map"
            },
            "action": "charge",
            "arc_list": [
                {
                    "node_dest": "1"
                }
            ]
        },
        {
            "id": "1",
            "name": "Node pick 1",
            "pose": {
                "x": 3.0,
                "y": 4.3,
                "z": 0.0,
                "theta": 1.57,
                "frame_id": "map"
            },
            "action": "pick_cart",
            "arc_list": [
                {
                    "node_dest": "0"
                }
            ]
        }
    ]
}
```

Ejemplo 3.1: Ejemplo de dos nodos en JSON

CAPÍTULO 4
Diseño de la solución

La solución propuesta ha sido llevada a cabo en función de la arquitectura y configuración ya aplicada anteriormente en el sistema de gestión de rutas. Esta configuración junto con los cambios realizados es lo que se pretende analizar en este capítulo.

4.1 Arquitectura del sistema

La arquitectura básica del sistema de creación, edición y visualización del grafo consiste en la mostrada en el esquema de la figura 4.1, que se describe con mayor profundidad en la próxima sección 4.2 [6].

La arquitectura básica del sistema que se pretende desarrollar para la edición de grafos se debe conocer para comprender el funcionamiento general del gestor de flotas. Esta estructura funciona de forma que un ordenador, ajeno a toda la instalación, se pueda conectar al router central al cual está conectado el servidor y los robots. Esta conexión, obviamente, solo es posible mediante las credenciales de acceso al sistema. Una vez conectado al router, desde cualquier ordenador compatible con HTML 5 (las últimas versiones de los principales navegadores web lo son), se accede a una dirección IP establecida. Esta dirección IP lleva al servidor web Apache, que ofrece la web del sistema de gestión de flotas. La web, mediante un *WebSocket* de JavaScript creado desde el ordenador cliente, se conecta con los servicios de ROS previamente inicializados en el servidor. Mediante las funciones que proporciona la interfaz del grafo (que hacen uso de ROS a través del *WebSocket*) se realiza la modificación del grafo. El servidor con ROS es el que se encarga, a través de la información recogida en dicho grafo y el feedback de la flota, de controlar a los robots. Mediante la web del sistema de gestión de flotas también se pueden modificar otros apartados del

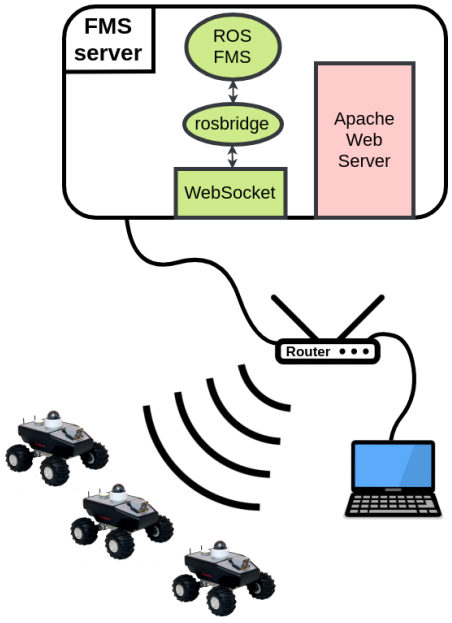

Figura 4.1: Esquema con la arquitectura básica del sistema de gestión de flotas.

funcionamiento de la flota de robots e incluso configuraciones individuales de estos, pero esta posibilidad es ajena al caso de uso actual.

4.2 Diseño detallado

En el esquema mostrado en la figura 4.2 podemos observar el diseño de las partes que conciernen a la modificación a realizar. La interfaz gráfica es el principal objetivo del proyecto, que como ya se ha comentado anteriormente, se pretende integrar dentro de la web del sistema de gestión de flotas. Se comunicará con ROS mediante un *WebSocket* y, mediante los servicios de ROS que deberán ser proporcionados por el *fms_routes*, se controlará la edición del grafo.

Además, se pretende que el *fms_routes* tenga un servicio que al ser llamado realice el guardado o la carga del grafo en el fichero JSON, como se muestra en la figura 4.2.

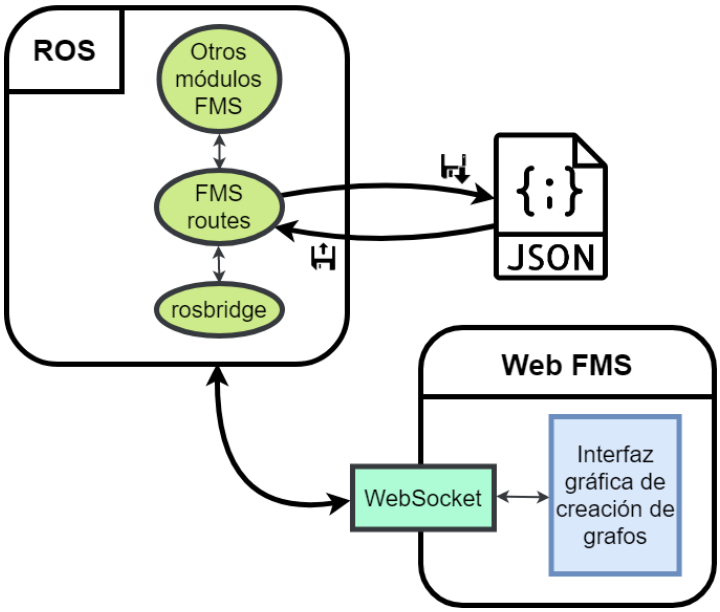

Figura 4.2: Arquitectura enfocada a la modificación a realizar.

4.3 Tecnología usada

4.3.1. ROS (Robot Operating System)

ROS es un *framework*[1] flexible que aporta la función de programar software para robots mediante el uso de sus herramientas y librerías. Gracias a sus sistemas de comunicación, permite el control de los robots dentro de un entorno centralizado. Se le llama Sistema Operativo por los servicios que aporta, tales como la abstracción del hardware, la gestión de paquetes, el paso de mensajes entre procesos, etc [12].

Este framework tiene diversos elementos, de los cuales los más importantes para nuestro caso de uso son:

[1]Framework: Su traducción al español es "entorno de trabajo" y consiste en un conjunto de herramientas software que permiten organizar y desarrollar programas y aplicaciones software.

- El compilador catkin: Actualmente se trata del principal sistema de compilación[2] de ROS, que combina los compiladores de C++ y Python permitiendo una mejor distribución de los paquetes y mejor compilación entre diversas versiones de los lenguajes, entre otras ventajas [8]. Este sistema es el que se utiliza para instalar el código desarrollado en los robots y en el servidor del sistema de gestión de flotas.

- Nodos de ROS: Es la forma en la cual ROS llama a todos los distintos programas que se están ejecutando sobre el sistema. En nuestro caso de uso, el *fms_routes* es uno de estos nodos, que se conecta mediante las herramientas de comunicación de ROS con el resto de nodos del sistema de gestión de rutas. Estos nodos pueden estar programados en C++ o en Python. Todo el sistema de gestión de rutas utiliza C++ por su mayor eficiencia y la compilación que ofrece. Mientras que Python es un lenguaje interpretado, C++ es un lenguaje compilado que nos permite borrar el código fuente una vez realizada la compilación. Esto permite que el cliente no pueda acceder al código base.

- Mensajes de ROS: ROS implementa un sistema de mensajes gracias al cual, una vez compilados, se gestionan los detalles de la comunicación entre los nodos mediante mecanismos de publicación y suscripción. Gracias a esto no hay que implementar ninguna interfaz entre nodos.

- *Topics*: Un topic consiste en un sistema de comunicación basado en la publicación de mensajes por parte de uno o más nodos que pueden ser leídos por todos los nodos que estén suscritos a ellos. En nuestro caso de uso, es muy útil para la publicación del grafo y el mapa (a la cual se suscribe la interfaz gráfica de creación de grafos).

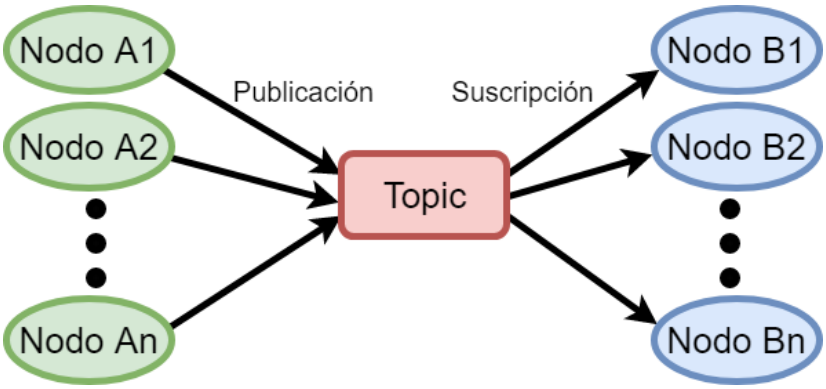

Figura 4.3: Comunicación mediante topics.

- Servicios: También son una herramienta de comunicación, la cual establece una relación cliente/servidor entre los nodos que se conectan y el nodo que ofrece el servicio. Cuando el servidor recibe una petición de un cliente, éste ejecuta una acción configurada previamente y retorna una respuesta al cliente que ha hecho la petición. En nuestro caso de uso, estos servicios son especialmente útiles para la edición del grafo. En las peticiones se puede informar al *fms_routes* acerca del cambio deseado, y las respuestas nos indican si se ha realizado correctamente o si ha habido algún error.

[2]Un sistema de compilación es aquel responsable de generar las dependencias y ejecutables a partir de código escrito por el usuario.

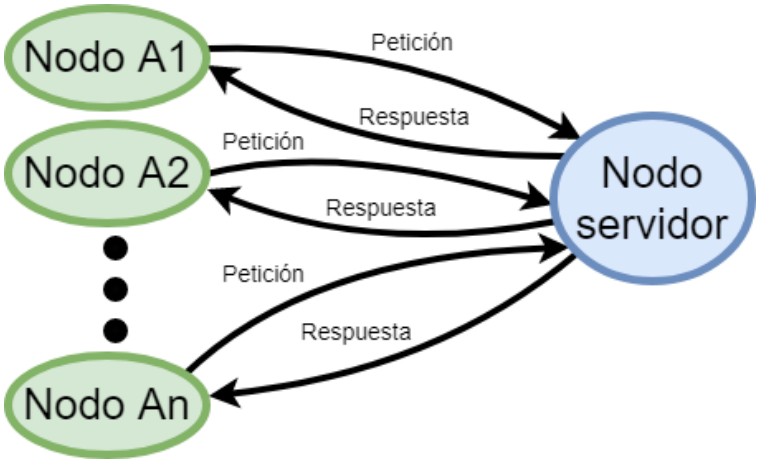

Figura 4.4: Comunicación mediante servicios.

4.3.2. JavaScript

JavaScript es un lenguaje de programación orientado a objetos. Se utiliza junto con HTML 5 para programar el comportamiento de las páginas web dinámicas. Dado que es un lenguaje interpretado, ofrece la ventaja de poder ser probado y depurado mediante cualquier navegador sin la necesidad de procesos intermedios como compilar o ejecutar (la simple carga del navegador es la que se encarga de ejecutar el código). En nuestro caso de uso, JavaScript es necesario para la herramienta interactiva de edición y visualización de grafos por las opciones que nos ofrece.

Ha habido tres librerías en concreto para JavaScript que han simplificado enormemente el desarrollo de esta herramienta. Aunque dos de ellas ya se han comentado anteriormente al valorar las diversas opciones disponibles, se pretende describirlas más exhaustivamente:

- La librería de código abierto *roslibjs* proporcionada por ROS aporta un *WebSocket* mediante el cual es posible conectarse con ROS, leer el topic a través del cual lee la información del grafo y llamar a los servicios de edición del grafo.

- La librería *visjs* se utiliza por su eficiente visualización de grandes cantidades de datos. En este caso, estos datos se muestran a través de un grafo. Como prueba de rendimiento, se ha intentado cargar un grafo con 9200 nodos para ver si puede gestionarlos y es capaz de ordenarlos en zonas (definidas por colores). Se puede observar el resultado (que se ha cargado en cuestión de décimas de segundo) en la figura 4.5.

- Aunque también se ha hecho una mención a *ros2djs* anteriormente, comentar que la única función que se está utilizando de ella actualmente es la visualización del mapa de ROS sobre el cual se muestra el grafo. La librería además se está utilizando de forma ineficiente, dado que no es compatible con la librería de *visjs* con la que se usa conjuntamente (para mostrar el mapa y grafo a la vez) y se ha hecho una modificación temporal sobre *ros2djs* a modo de parche. Se pretende en una futura versión de la interfaz eliminar esta librería, e incorporar sobre el código otra forma de visualizar el mapa.

4.3.3. JSON

El estándar JSON (*JavaScript Object Notation*) es un formato de intercambio de datos claro y fácilmente comprensible y modificable por los humanos. A la hora de serializar y deserializar en este tipo de archivos, es muy rápido y sencillo de convertir. En el ejemplo 3.1 comentado anteriormente se puede observar la sencillez del formato [13].

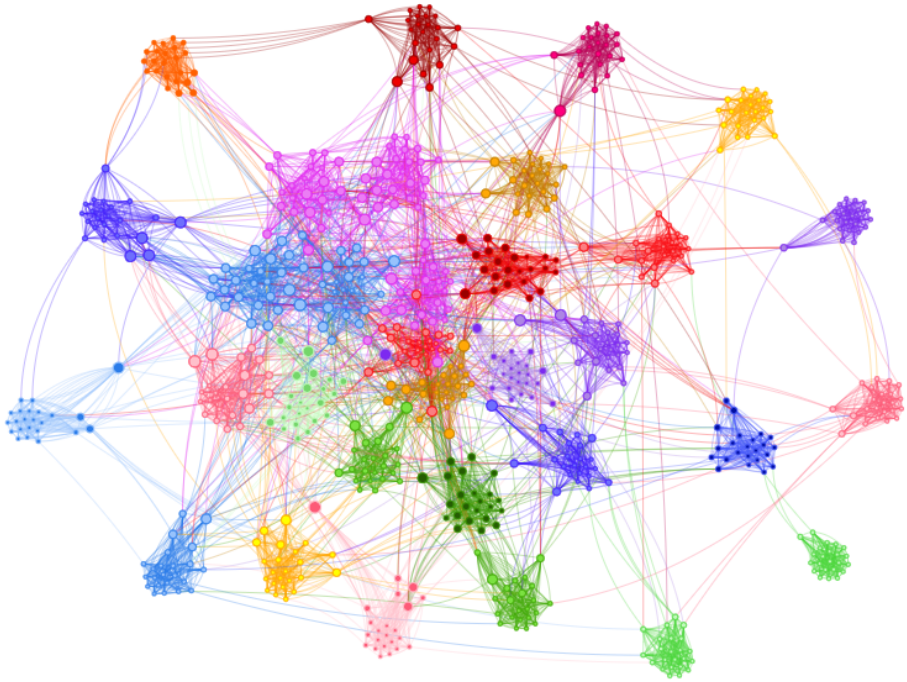

Figura 4.5: Grafo dividido por zonas (colores) con 9200 nodos.

CAPÍTULO 5
Desarrollo de la solución propuesta

Este proyecto se compone de diversas secciones, que se pretenden desarrollar a lo largo de distintas fases. En este capítulo se comenta en qué consisten estas partes que forman el proceso de desarrollo.

5.1 Interfaz para la web

Para el desarrollo de la interfaz interactiva se ha propuesto hacer dos vistas distintas; una primera en la que se pueda ver el grafo sin posición establecida para los nodos y otra en la que se vea el mapa generado por *ros2djs* con los nodos en sus respectivas condiciones.

- Vista de grafo: La intención con esta vista es que los nodos no estén asociados a una posición en el mapa, si no que la propia librería se encargue de posicionar los nodos aleatoriamente de modo que no se superpongan. Esto ayuda a que, si hay una gran cantidad de nodos muy juntos entre ellos, sea posible modificarlos de forma más cómoda. En la figura 5.1 podemos observar un grafo en el cual todos los nodos están prácticamente sobre las mismas coordenadas, pero *visjs* nos ayuda a visualizar de forma más cómoda las relaciones entre ellos.

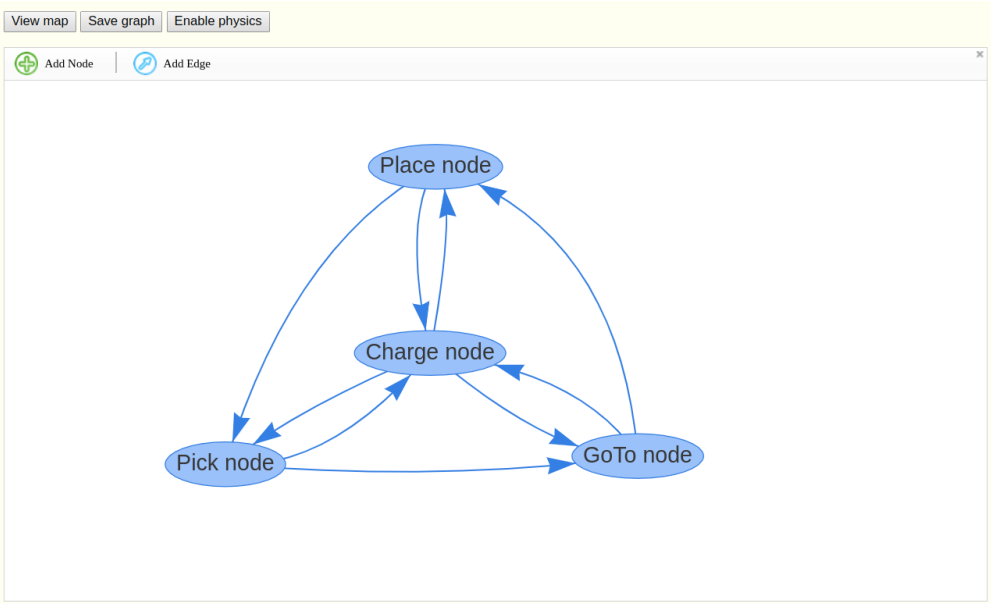

Figura 5.1: Vista de grafo en la interfaz web.

Hay que tener en cuenta que se trata de una versión inicial, donde no importa el diseño sino la utilidad. Se pretende, una vez esté totalmente terminado e integrado con el resto del sistema de gestión de rutas, modificar el diseño para que todos los apartados sean coherentes a una misma especificación.

- Vista de mapa: La intención con esta vista es que los nodos se posicionen en función de un mapa mostrado en el *Canvas*, con la posición relativa a la indicada sobre ROS. Esta visualización permite posicionar de forma interactiva los nodos sin tener que conocer las coordenadas precisas en el mapa. De todos modos, si es necesario, se ofrece la función de modificar las coordenadas de numéricamente y ver la nueva posición seleccionada en tiempo real. En esta vista se planteó la opción de utilizar *ros2djs* sin *visjs* por su complicada integración pero se llegó a la conclusión de que era más costoso diseñar un sistema de visualización y edición de puntos desde cero que integrar ambas librerías en un mismo *Canvas*.

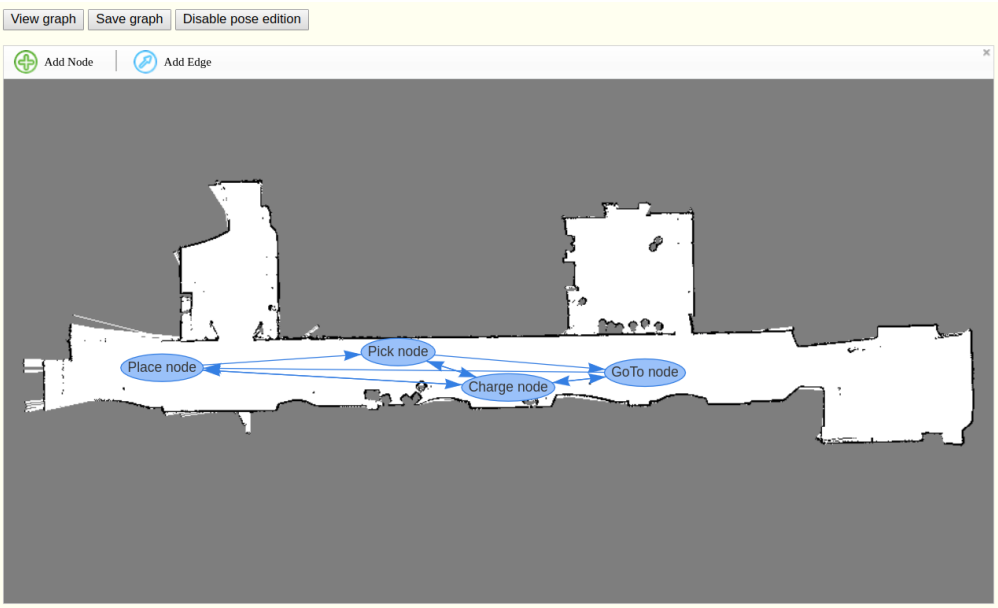

Figura 5.2: Vista de mapa en la interfaz web.

5.1.1. Web FMS

La finalidad de esta web es la gestión de la flota de robots. Mediante ella se les pueden mandar nuevas tareas, detenerlos o incluso visualizar sus procesos actuales. Aunque se está trabajando en una nueva web (la cual integrará la interfaz mostrada en las figuras 5.1 y 5.2), todavía se hace uso de la antigua, que se puede observar en la figura 5.3. Actualmente también tiene una vista del mapa con los nodos, pero es muy rudimentaria e implementa el diseño 3D que queremos evitar por su mal rendimiento.

5.2 Estructura de datos

Aparte del fichero JSON comentado anteriormente para guardar el grafo, se utiliza una jerarquía de mensajes (figura 5.4) que también se usa a nivel interno en las clases de C++ (que se comentan posteriormente en la sección 5.3). Esta jerarquía es interesante mostrarla ya que es sobre la que se trabaja a la hora de implementar la funcionalidad interna del grafo (por ejemplo, el algoritmo de *Dijkstra*).

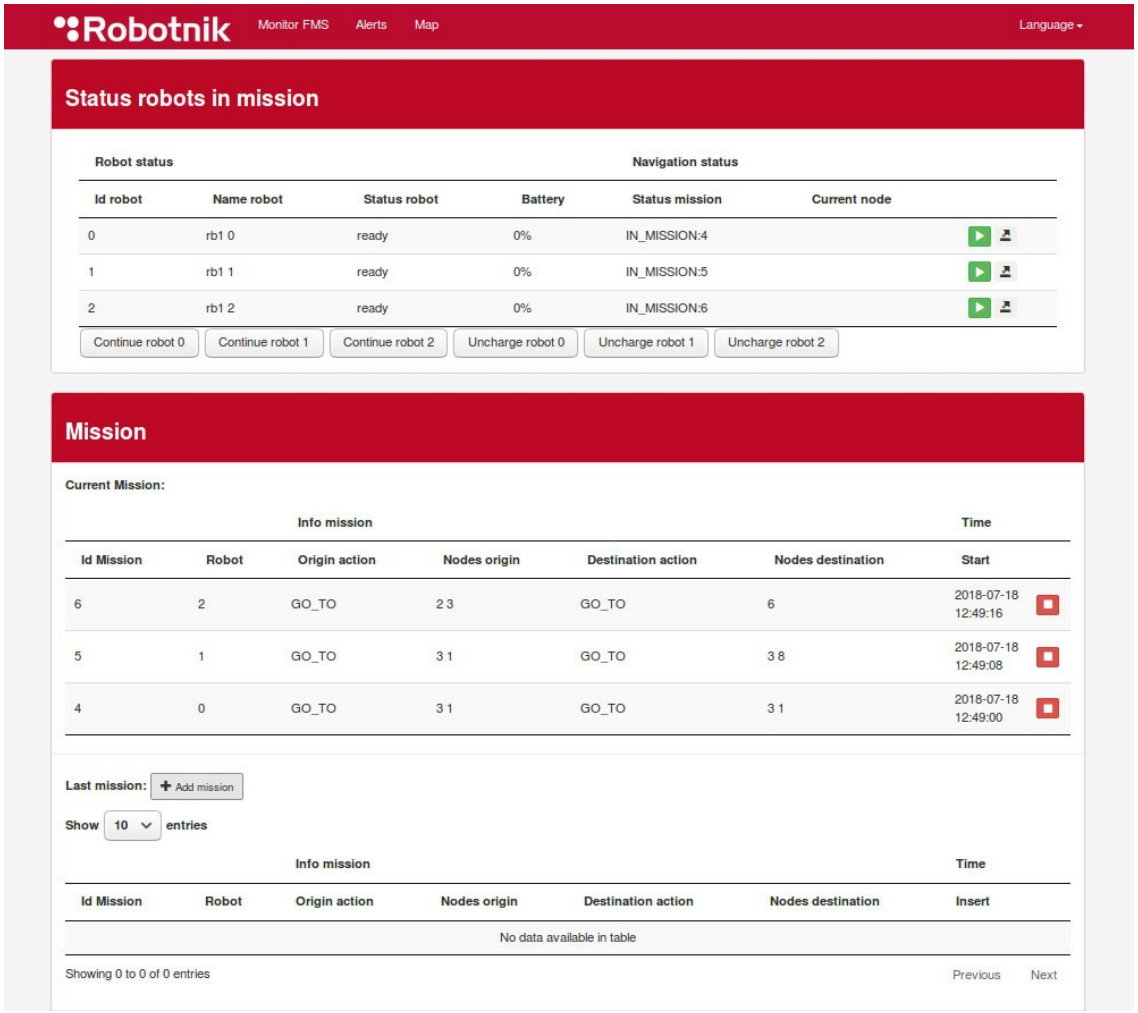

Figura 5.3: Vista de la interfaz de la antigua web del sistema de gestión de flotas.

5.3 Ficheros C++

Aunque la modificación de estos ficheros no es el principal objetivo del proyecto, es el que más tiempo ha llevado dado que se ha rediseñado totalmente la estructura interna, simplificándola y haciendo que funcione de forma más eficiente. Además, como se ha comentado anteriormente, se ha modificado la carga del grafo y la gestión de éste.

5.3.1. Dijkstra

La clase Dijkstra tiene este nombre porque contiene el algoritmo de Dijkstra, mediante el cual se calcula la mejor ruta para llegar de un nodo a otro. En esta clase es donde se mantiene el grafo cargado durante la ejecución y se implementan los métodos para su edición, tales como la adición de nuevos nodos y aristas, edición de sus características, etc.

También se encarga de validar, tras la carga del grafo o su edición, que éste cumpla ciertas características. Las características principales que debe cumplir son:

- De cualquier nodo se puede acceder a todos los nodos.
- Todos los arcos tienen un nodo destino.

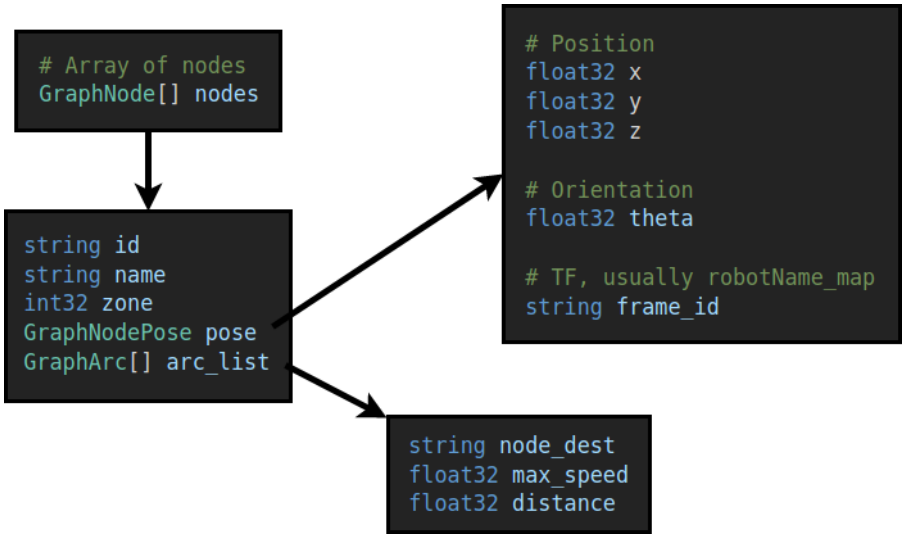

Figura 5.4: Estructura del grafo.

- No hay dos nodos con una misma ID.
- No hay dos arcos dirigidos con los mismos nodos de origen y destino.

También se encarga de tareas como el bloqueo de nodos o zonas. Una zona es una lista de nodos, en la cual hay asignada una cantidad máxima de robots. Si esta cantidad máxima se sobrepasa, el algoritmo de Dijkstra deja de tener en cuenta todos los nodos que pertenecen a esa zona a la hora de calcular una ruta.

Además, se ha realizado una modificación importante en lo que respecta al cálculo de la posición actual de los robots en el grafo. En anteriores versiones, la posición del robot dependía de la posición del nodo más cercano. Esto podía causar que en situaciones como la de la figura 5.5 el sistema calculase que un robot se encuentra en un nodo que en realidad ni siquiera está en esa misma habitación. La solución que se aplicaba anteriormente era aumentar el número de nodos. Esto se ha solucionando teniendo en cuenta también la posición de los arcos, que también pasan a representar una posible posición del robot.

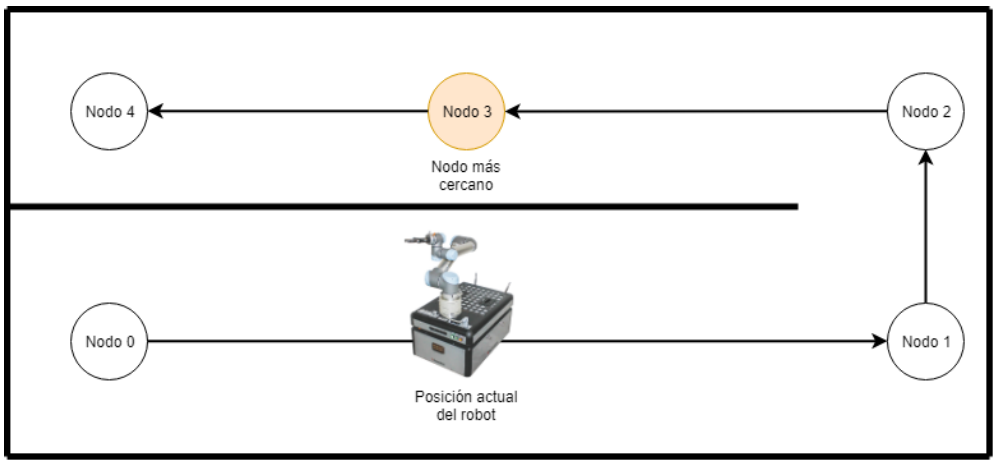

Figura 5.5: Situación problemática de posicionamiento del robot sobre el grafo en anteriores versiones.

5.3.2. Graph

La clase Graph es la encargada de interactuar con la clase Dijkstra. Se encarga de sobrescribir los métodos de Dijkstra, añadiéndoles una gestión de errores para que sea posible utilizar sus métodos sin fallos en caso de no utilizar la interfaz gráfica, que también limita las funciones a las estrictamente válidas.

Su función principal es la de cargar el grafo desde el archivo JSON y guardarlo posteriormente. Una vez terminada la carga del JSON, le manda los datos del grafo a Dijkstra, que como hemos comentado antes es la clase encargada de mantenerlos en tiempo de ejecución.

5.3.3. Routes

El fichero Routes es el encargado de inicializar una instancia de Graph e implementar diversos topics y servicios para poder modificar el grafo a través de ellos, mediante llamadas a los métodos de Graph. Por tanto, es el que realiza la conexión con ROS y permite, a través de dicha conexión, el funcionamiento de la web.

Las modificaciones realizadas en este fichero han consistido en la implementación de dichos topics y servicios de edición.

También se encarga de responder las peticiones del resto del sistema de gestión de flotas, proporcionando un servicio en cuya petición se indica la ID de un nodo destino y de un nodo final, y retorna la lista de nodos por los cuales hay que pasar para llegar de uno a otro.

5.3.4. Acciones

Aunque las acciones forman parte de la segunda fase del proyecto, se tiene ya una idea base sobre la cual partir. Se pretende que todas las acciones se puedan configurar a nivel de grafo desde la interfaz gráfica. Para ello se ha pensado en una estructura de datos que permita configurar dichas acciones de forma simple y lo más genérica posible.

Se añadirá un nuevo campo a la estructura de los arcos, que consistirá en un vector de acciones. La estructura de dichas acciones será la mostrada en la figura 5.6.

Figura 5.6: Estructura de las acciones.

La intención con esta estructura es que cada vez que se vaya a pasar a un nuevo nodo, se analice el vector de acciones que tiene el arco por el cual se va a ir. Por cada acción podemos tener varias condiciones (o ninguna en caso de que queramos que se ejecute sí o sí dicha acción). En cada una de esas condiciones se indicará el nombre de la condición (por ejemplo, "*elevator_raised*" o "*cart_free*"). En la variable *condition_enabled* se indicará si se pretende que dicha condición debe ser verdadera o falsa para que se cumpla la acción. Además, se establece una variable auxiliar para el caso en el que se requieran más datos, como el nombre de la base de carga a la que se quiera ir en caso de estar disponible. Deberán cumplirse todas las condiciones para que se ejecute la acción.

Hay que tener en cuenta que las acciones mencionadas en la estructura de datos y sus respectivas condiciones deberán estar definidas y soportadas por el código que se encargue de gestionarlas.

CAPÍTULO 6
Implantación

Para mantener el sistema de gestión de flotas trabajando continuamente en una flota de robots, controlada por un servidor, primero es necesario entender los componentes del software del gestor que se instalan en el proceso de implantación. Además, de este modo, es posible solucionar los diversos errores que puedan surgir con la llegada de actualizaciones del sistema.

6.1 Robot

El robot se compone de varias partes software para funcionar. La instalación de estos componentes es bastante sencilla gracias al script desarrollado por Robotnik que permite, mediante una simple guía de instalación que se proporciona al cliente, instalar todo el sistema en pocos pasos.

Normalmente este software se proporciona ya instalado en los robots. Aun así, es útil entender algunos de sus componentes.

6.1.1. Robot Local Control

Consiste en un servidor de acciones que se conecta con un conjunto de componentes del robot para realizarlas. Tiene ya programado un gran conjunto de funcionalidades (como puede ser coger un carro o ir a la base de carga), por lo que, a través de dicho servidor, el sistema puede ejecutar todos los procedimientos deseados y, por tanto, gestionar la flota en su totalidad. Se instala junto con las librerías de Robotnik.

6.1.2. Servicios REST

Toda la comunicación entre los robots y el sistema de gestión de flotas se realiza mediante REST, que consiste en un estándar que en el caso de Robotnik se utiliza para la creación de APIs. En dichas APIs se utiliza JSON para solicitar acciones mediante peticiones por parte del servidor, y para recibir las respuestas enviadas por los robots. Se instala junto con las librerías de ROS modificadas por Robotnik.

6.1.3. Mapeado de zona

Es necesario realizar un buen mapeado de la zona en la cual va a trabajar el sistema de gestión de flotas mediante los sensores láser del robot, dado que el grafo marcará las posiciones a las cuales va a ir el robot sobre el mapa que se genere. Por tanto, cuando

más fiel a la realidad sea el mapa, más real será el posicionamiento. El mapeado de la zona se realiza en la web de gestión particular del robot desde el cual se esté realizando el proceso, mostrada en la figura 6.1.

El robot se ubica sobre el mapa gracias a sus sensores láser, los cuales detectan la forma de las paredes que tiene enfrente y las relacionan con el mapa. También se utilizan marcas en el entorno, como códigos QR[1], para aumentar su precisión o corregir posibles pérdidas de ubicación.

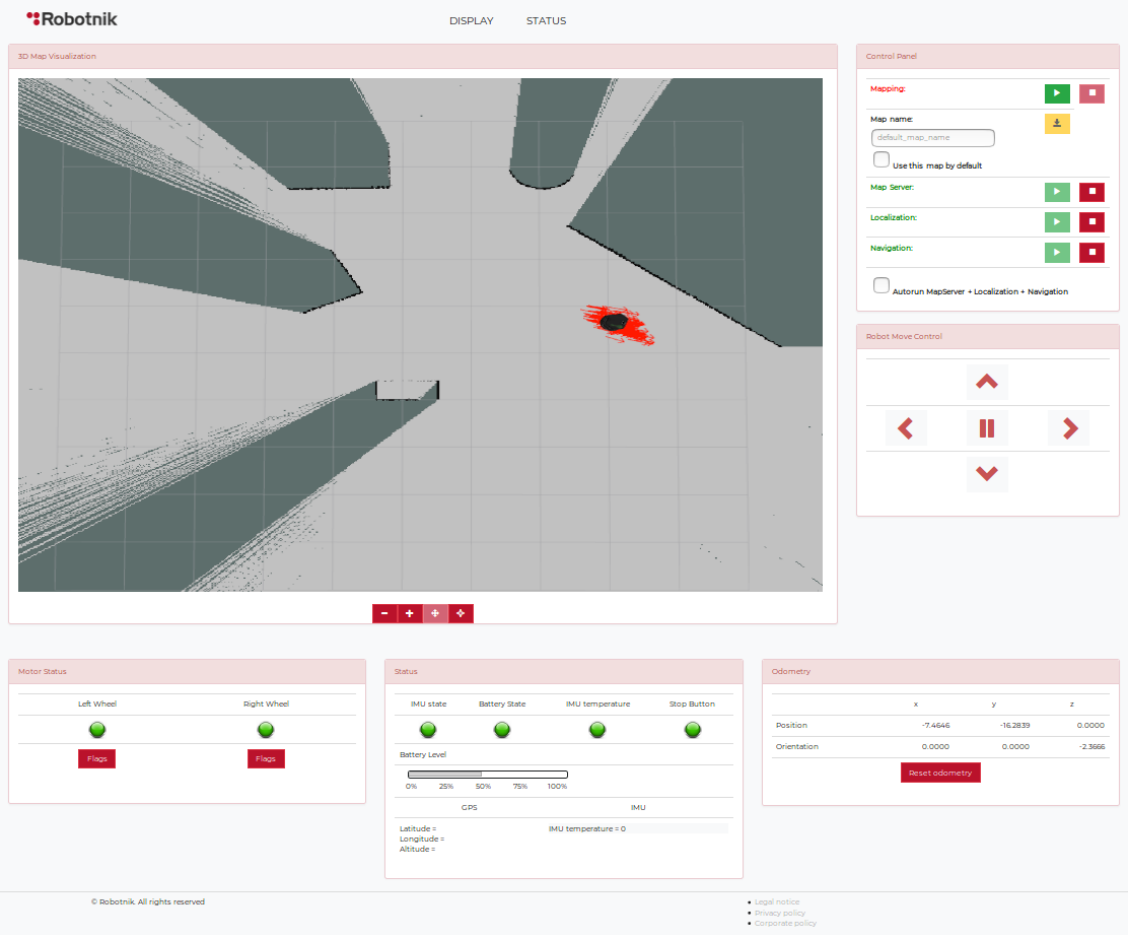

Figura 6.1: Web para el mapeado.

6.2 Servidor

El servidor también consta de varias aplicaciones que se deben instalar y conocer para su funcionamiento. Se comentan las más importantes y su proceso de instalación de una forma muy resumida, dado que en ninguna situación se realiza dicha instalación fuera de Robotnik y sus actualizaciones vienen acompañadas de una guía muy completa.

6.2.1. Base de Datos FMS

Mediante el programa *tasksel*, que se puede encontrar en los repositorios de Ubuntu, instalamos LAMP, que consiste en un paquete de software que permite operar com-

[1]Código QR: Código de barras bidimensional cuadrado que almacena datos que pueden ser leídos mediante el uso de una cámara. En caso de utilizarlos para la localización, el dato que se indica en el QR es la posición del mismo.

partiendo el máximo número de librerías para ser lo más eficiente posible. En este caso LAMP se utiliza para la instalación de Apache, MySQL y PHP.

Una vez configurado el software, para el caso de MySQL simplemente habrá que importar la base de datos que se proporciona. Esta base de datos es necesaria para el funcionamiento del sistema de gestión de rutas.

El servidor Apache se utiliza para la web del sistema de gestión de flotas. Esta web se lanzará junto con el resto de nodos de ROS, por lo que no hace falta ninguna configuración previa.

6.2.2. Repositorios ROS

Se trata de una carpeta que contiene el sistema de gestión de rutas y todas las librerías requeridas, así como la web en la que se integra la herramienta gráfica desarrollada en este proyecto. No es necesario conocer el funcionamiento ni contenido de estos repositorios, ya que no afecta a nuestro caso de uso (más allá de las estructuras comentadas anteriormente).

La instalación se realiza de forma bastante sencilla ejecutando los comandos indicados en una guía de instalación que se ofrece al cliente.

6.2.3. Configuración misiones

Aunque la parte de las misiones todavía se encuentra en proceso de desarrollo, se pretende que su configuración sea tan sencilla como la propia definición del grafo. Esto es, como se ha comentado anteriormente en el apartado 5.1, a través de la web.

La idea es que se puedan completar las acciones y sus condiciones desde la interfaz gráfica. Por como está desarrollado el actual proyecto, a nivel de la interfaz gráfica sería un cambio muy sencillo (que consiste en añadir campos para la adición y edición de las acciones de cada arco).

6.3 Resultado

En la figura 6.2 podemos ver un ejemplo de una flota de robots ya configurada para encargarse del transporte de materiales. Con un mínimo conocimiento acerca del proceso de implantación, es posible que el cliente siga actualizando el sistema con las nuevas versiones proporcionadas sin apenas soporte por parte de Robotnik.

Figura 6.2: Flota de robots en funcionamiento

CAPÍTULO 7
Pruebas

Dado que el nuevo sistema de gestión de flotas compatible con el presente proyecto todavía se encuentra en fase de desarrollo, se ha optado por diseñar un *script* capaz de llamar al nuevo *fms_routes* para ponerlo a prueba.

Se le han incluido varios parámetros para poder realizar diversas pruebas sin modificar el código. El *script* está programado en ROS y se encarga de obtener la ruta hasta cierto nodo desde el nodo actual a través de la llamada al nuevo *fms_routes*. Además, están programadas las acciones que se desea que realice el robot tras su paso por cada nodo (en función al nombre del nodo).

La principal llamada utilizada por este script, modificando los campos *"from_node"* y *"to_node"* en función al ID de los nodos origen y destino, es la siguiente:

```
$ rosservice call /fms_routes_node/get_route
                    "from_node: 'pick_0000' to_node: 'place_0005'"
```

Esta llamada, en el caso de la figura 7.1, retorna la siguiente respuesta:

```
$ ret: True
  msg: "Route found from pick_0000 to place_0005"
  nodes: [pick_0000, node1_0001, node2_0002, node3_0003, node4_0004,
          place_0005]
```

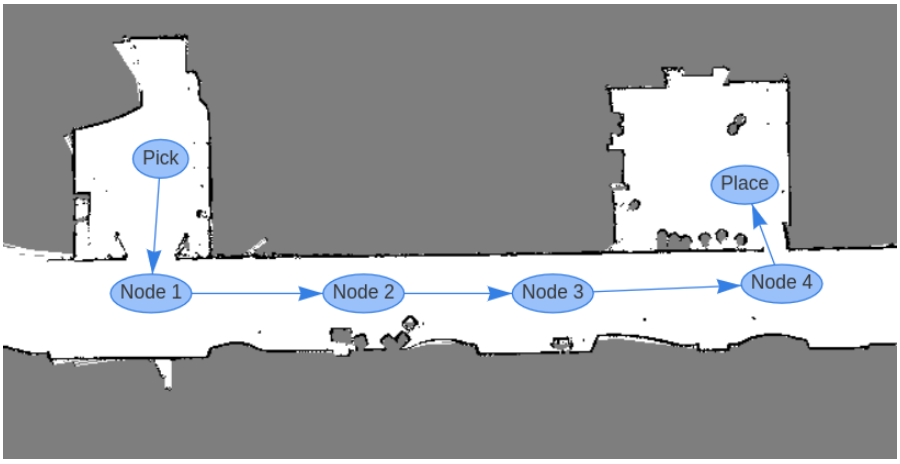

Figura 7.1: Grafo utilizado para uno de los test.

El *script* permite una reproducción bastante fiel a la secuencia de acciones que va a realizar realmente el nuevo *fms_routes*. Además, para probar también la interfaz gráfica

de creación y edición de grafos, se ha realizado una encuesta a un grupo de personas de Robotnik con conocimientos muy básicos sobre el sistema de gestión de flotas. Para realizar esta encuesta se ha pedido que diseñen una ruta. Previamente se les ha mostrado un borrador del grafo que debían diseñar que se trata del de la figura 7.1.

Las funciones de la interfaz gráfica que se han permitido utilizar al usuario son las siguientes:

- Añadir nodo o modificar sus parámetros: Esta opción permite añadir nuevos nodos sobre la posición seleccionada en el mapa. Una vez hecho clic sobre la posición deseada, aparece un menú con diversos campos a rellenar que son los distintos parámetros del nodo. También se ofrece la opción de modificar estos campos en un nodo ya creado previamente. Los parámetros a modificar se pueden ver en la figura 7.2.

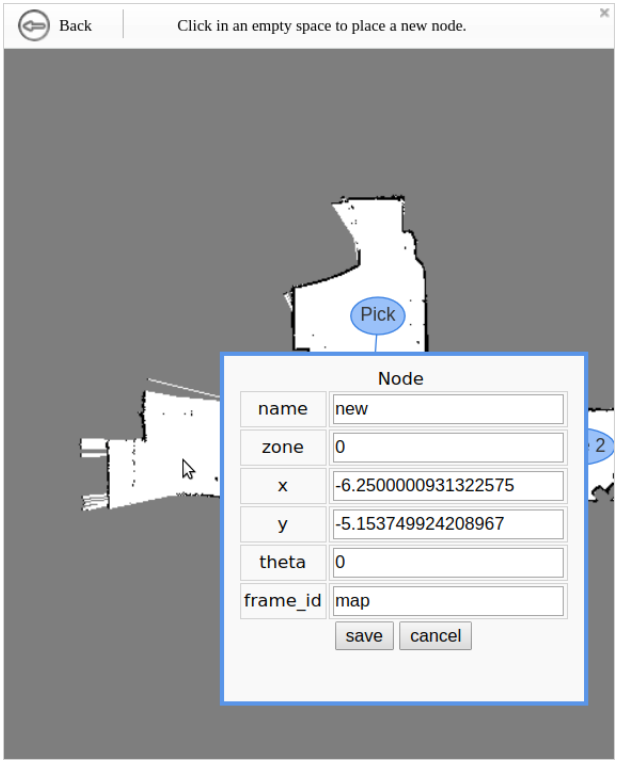

Figura 7.2: Adición de un nodo y modificación de sus parámetros.

- Añadir un arco: En este caso, una vez pulsado el botón correspondiente, la interfaz nos hace elegir un nodo al cual añadir el arco. Una vez seleccionado dicho nodo, arrastramos hacia el nodo al que queremos dirigir el arco. Los parámetros, que en este caso no se suelen modificar en todos los arcos, se editarán posteriormente. Este proceso se puede ver en la figura 7.3
- Editar parámetros de un arco: Como hemos comentado anteriormente, los parámetros del arco se modifican posteriormente a su adición. Para ello, hay que hacer doble clic sobre el arco que se desee modificar. Se pueden ver estos parámetros en la figura 7.4.
- Modificar posición de un arco: Puede que se desee modificar el nodo origen o destino de alguno de los arcos. Para ello habrá que seleccionar el arco, pulsar el botón de editar, y arrastrar la esquina del arco que queramos cambiar de nodo. Se puede observar un ejemplo de esto en la figura 7.5.

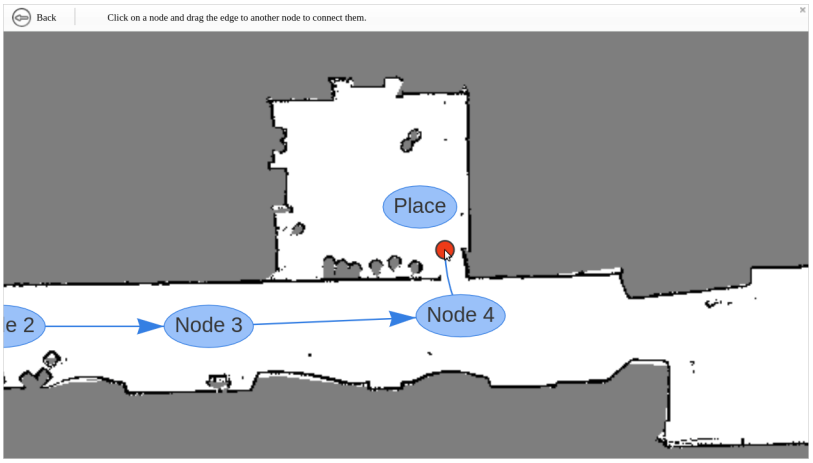

Figura 7.3: Adición de un nuevo arco.

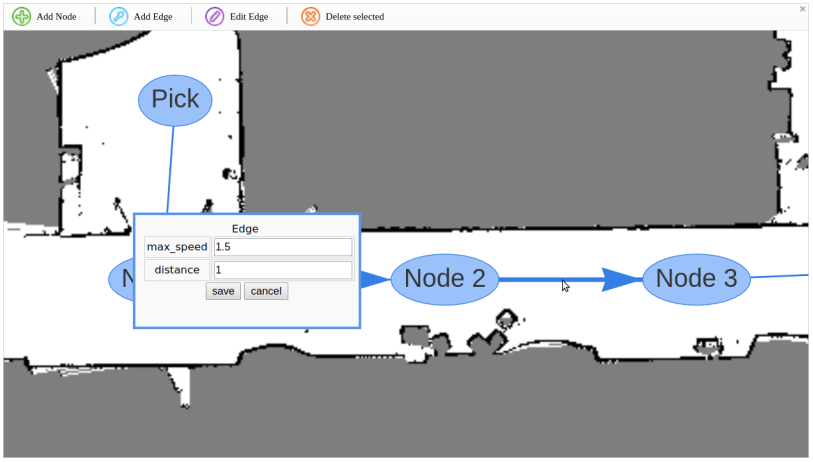

Figura 7.4: Edición de los parámetros de un arco.

- Eliminar un nodo o arco: Se puede seleccionar un nodo o un arco y eliminarlos. En caso de borrar un nodo, se borrarán todos los arcos que salen o terminan en él.
- Activar físicas: Como podemos observar en la figura 7.7, hay un botón para activar las físicas. Este botón solo aparece en la vista del grafo sin el mapa, y permite que las físicas programadas en la librería VIS se activen y posicionen los nodos automáticamente para que ninguno se superponga y la vista sea más clara en grafos con muchos nodos.
- Guardar grafo: Otro de los botones que podemos observar en la figura 7.7 es el de guardar grafo, que se encarga de llamar al servicio de ROS correspondiente para que guarde el grafo actual en un fichero JSON. De este modo, si se reinicia el sistema, se mantendrá el grafo con las nuevas modificaciones.
- Seleccionar varios nodos: Si al hacer clic sobre un nodo o arco, mantenemos el botón "ctrl" pulsado, se añadirá dicho nodo o arco a la selección actual. Esto sirve para poder mover al mismo tiempo varios nodos a la vez, o eliminar al mismo tiempo varios nodos y arcos a la vez. Se puede observar la selección al mismo tiempo de medio grafo en la figura 7.8.

Se ha realizado una encuesta a todos los usuarios que han utilizado la aplicación para comprobar que se han cumplido los objetivos deseados. Para ello se han valorado tres apartados:

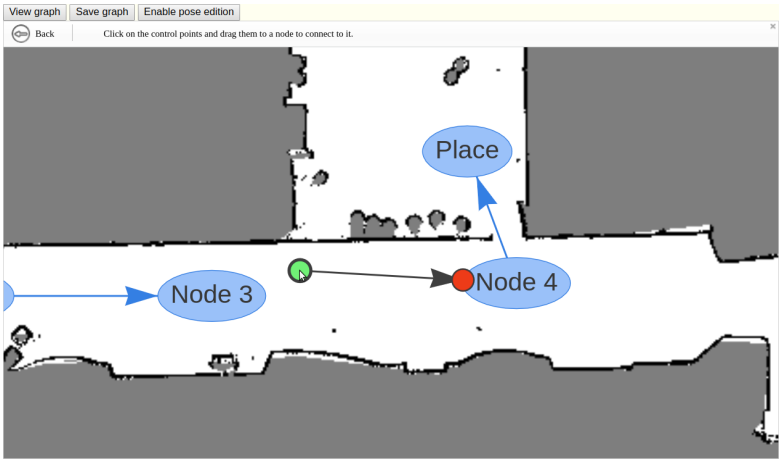

Figura 7.5: Modificación de la posición de un arco.

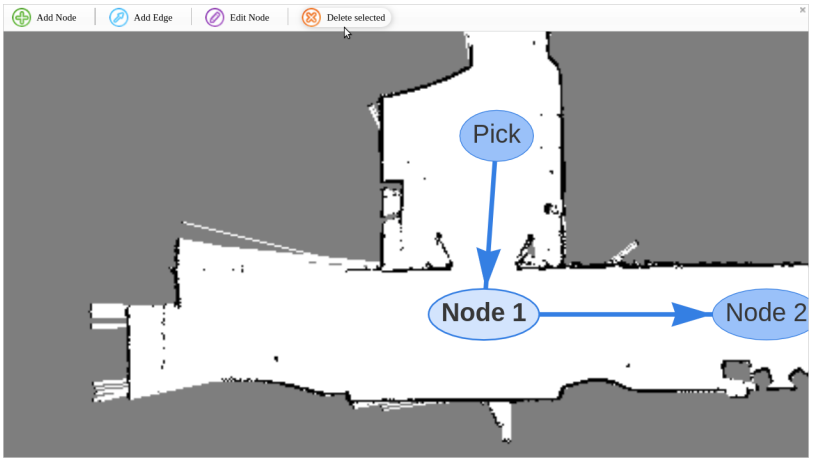

Figura 7.6: Eliminación de un nodo o un arco.

- Facilidad de aprendizaje y uso. Se ha obtenido un 10/10, sobre el cual no ha habido ningún comentario negativo.

- Claridad de la visualización. Se ha obtenido un 7/10. Los comentarios recibidos sobre esta bajada de puntuación están relacionados con el tamaño de los nodos y su poca claridad si se ponen muy cerca unos de otros. Gracias a estos comentarios, una de las próximas mejoras planteadas es un ajuste del tamaño de los nodos en función a la ampliación del mapa. Otra mejora pendiente es la de añadir un botón que modifique la forma de los nodos, de modo que al pulsarlo muestre los nodos como simples puntos más pequeños que permitan ver su ubicación de forma más precisa.

- Rapidez de edición: Se ha obtenido un 9/10. La valoración de la rapidez de edición del grafo solo la han realizado los usuarios que ya estaban habituados a modificar el grafo mediante la edición del XML, ya que son los únicos con una experiencia sobre la cual comparar. Ha habido sugerencias para aumentar todavía más la velocidad, como la implementación de comandos, que también se pretende implementar próximamente dado que no es un añadido costoso.

Esta encuesta la han realizado 11 usuarios en total, que son los miembros del departamento de software que han tenido la oportunidad de probar la interfaz. De estos

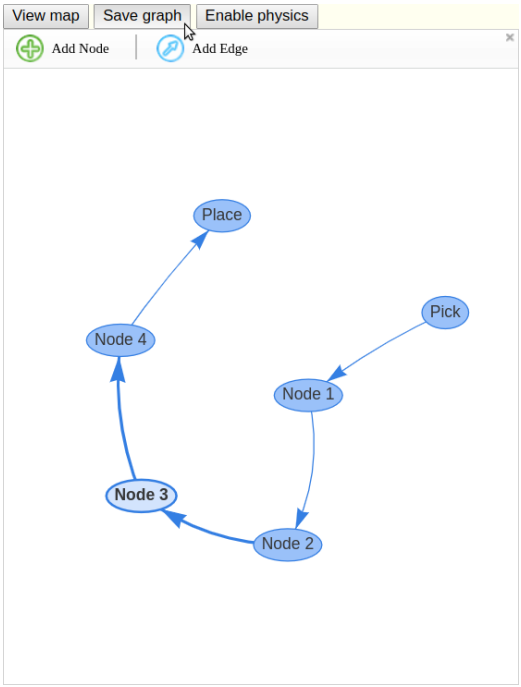

Figura 7.7: Botones externos al *Canvas*.

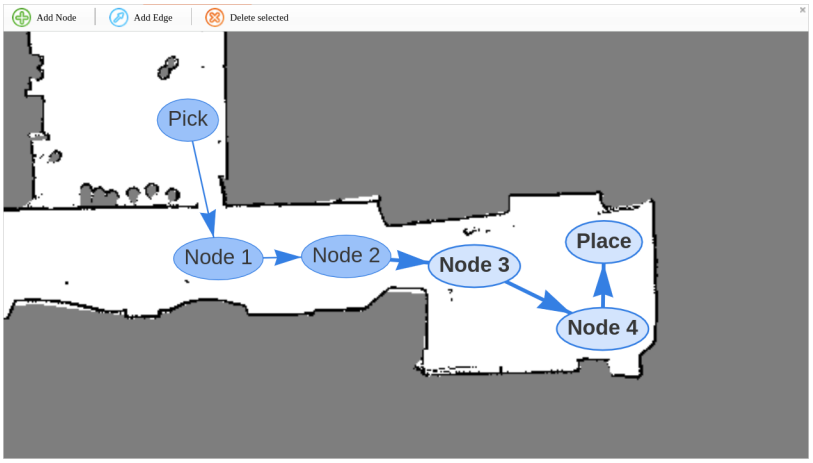

Figura 7.8: Selección simultánea de varios nodos y arcos.

miembros, solo 4 habían configurado previamente una ruta en el viejo sistema de gestión de flotas.

Se ha podido comprobar por el *feedback* recibido en la encuesta que la creación de grafos funciona correctamente. Aunque se trata de un diseño muy rudimentario y que no sigue un mismo patrón, no es una prioridad de este proyecto el tener un diseño consistente (aunque se prevé que en un futuro lo tenga). La finalidad es ser práctico. Podemos concluir, por tanto, que esta nueva interfaz, además de ser cómoda de utilizar, ahorrará mucho tiempo y dinero en las futuras puestas en marcha del sistema de gestión de flotas en nuevas instalaciones.

También se han realizado varias pruebas de configuración para ver el número de nodos que hacían falta anteriormente para definir una zona y compararlo con el número de nodos que hacen falta con el nuevo sistema para definir la misma zona. El más representativo es el caso en el que se define una zona de dejada y recogida de carros. Podemos

ver un ejemplo de este caso con espacio para tres carros en las figuras 7.9 (antes) y 7.10 (después). Se puede observar una disminución del número de nodos de hasta un 83 %.

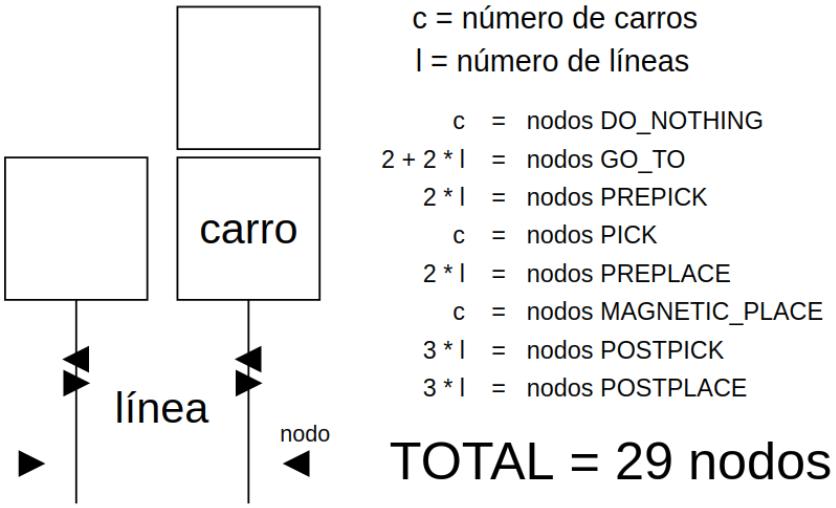

Figura 7.9: Número de nodos necesarios en el **viejo grafo**.

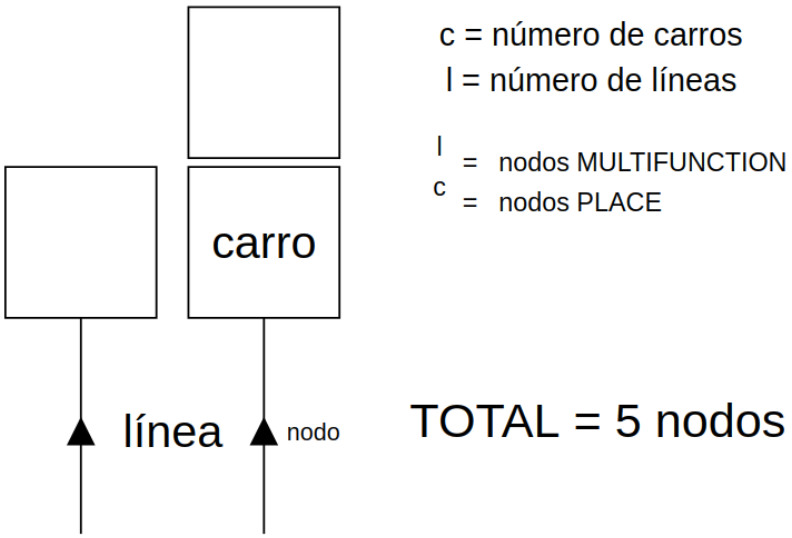

Figura 7.10: Número de nodos necesarios en el **nuevo grafo**.

CAPÍTULO 8
Conclusiones

A lo largo de este trabajo se ha podido observar el proceso de desarrollo de una herramienta gráfica web capaz de establecer rutas mediante el grafo de un sistema de gestión de flotas de robots. Esta interfaz, que se pretende acoplar a la web de control del sistema de gestión de flotas, ha permitido lograr una eficiencia mucho mayor que el anterior método de creación de dichos grafos.

También se ha mejorado una parte del propio sistema, que en un futuro se integrará con un renovado gestor de flotas más avanzado y eficiente. Concretamente se ha modificado el nodo *fms_routes* para que actúe como *backend* de la interfaz gráfica. Se ha optimizado el funcionamiento de dicho nodo y, gracias al nuevo planteamiento del sistema de gestión de flotas, se ha reducido el número de nodos necesarios para realizar las mismas acciones. Además, se ha programado de forma que solo con el uso de ROS (sin utilizar la interfaz ni modificar manualmente la base de datos) sea posible crear y modificar el grafo.

Se ha establecido una metodología como estándar para editar los mensajes tanto a nivel de ROS como a nivel de la interfaz. Esto permitirá que toda modificación de dichos mensajes para añadir nuevas acciones no aumente su complejidad.

Se ha pasado a JSON como formato para almacenar el grafo en lugar de XML. Esta modificación ha aumentado considerablemente la eficiencia a la hora de cargar el grafo gracias al cambio de librerías, además de haber simplificado la base de datos a ojos de un humano.

El desarrollo de este proyecto, por tanto, se puede considerar un éxito ya que finalmente ha quedado demostrado que es una solución que va a permitir un gran avance en la instalación de nuevas flotas de robots e incluso en sistemas de un único robot en el cual se necesite un grafo de acciones para gestionarlo.

8.1 Relación del trabajo desarrollado con los estudios cursados

Pese a que para una gran parte del proyecto se han utilizado tecnologías que no se han estudiado en el Grado de Ingeniería Informática, se han podido observar muchas similitudes con otras que si que se han cursado y que han permitido que el aprendizaje sea relativamente sencillo.

Las asignaturas que más han ayudado para este proyecto han sido Sistemas Gráficos Interactivos y Tecnologías de Sistemas de Información en Red. Han sido muy útiles para el desarrollo de la parte con JavaScript. Además, la segunda también ha sido muy útil para conocer los patrones de comunicación, como pub/sub y req/rep, que se utilizan para

los *topics* y servicios de ROS. Por otra parte, el aprendizaje de C y su uso de los punteros, las estructuras de datos vistas a lo largo de la carrera y las distintas arquitecturas han sido también muy importantes para el proyecto.

8.2 Trabajos futuros

El próximo paso consiste en la implementación de las acciones para que pueda funcionar el sistema de gestión de flotas. Además, hay diversas funciones que en el antiguo sistema de gestión de flotas ya están implementadas y se pretenden hacer funcionar en el nuevo, pero la interfaz no es todavía compatible con ellas.

8.2.1. Soporte para las acciones

Esta es una implementación relativamente sencilla de realizar a nivel de la interfaz gráfica, dado que se limita a modificar la estructura de los mensajes.

A nivel de ROS tiene un poco más de complicación dado que hay que implementar el soporte a estas acciones y condiciones, comentadas en el apartado 5.3.4.

8.2.2. Soporte para varias plantas

Esta función, integrada a nivel de sistema de gestión de flotas, requiere que se puedan soportar varios mapas sobre cada uno de los cuales estarán parte de los nodos del grafo. Lo que se utiliza en este caso es el parámetro *frame_id* para indicar a qué mapa pertenece cada nodo.

A nivel de la interfaz gráfica, habrá que implementar un sistema que permita seleccionar qué mapa visualizar. Según el mapa que esté seleccionado, se mostraran unos u otros nodos en función del *frame_id* que tengan.

Bibliografía

[1] Adrián Jiménez Cámara. Centralized control for robot fleets. *Universidad Carlos III de Madrid*, septiembre, 2012.

[2] Alejandro Rodriguez Villalobos. Integración de un SIG con modelos de cálculo y optimización de rutas de vehículos CVRP y software de gestión de flotas. *Escuela Politécnica Superior de Alcoy*, septiembre, 2007.

[3] J. A. Bañares, P. Álvarez, R. López y P.R. Muro-Medrano. Incorporación de componentes de visualización SIG en entornos distribuidos con tecnologías COM/CORBA, aplicación a un sistema de monitorización de flotas . *Departamento de Informática e Ingeniería de Sistemas. Centro Politécnico Superior, Universidad de Zaragoza*, 2001.

[4] F. M. Sánchez Martín, F. Millán Rodríguez, J. Salvador Bayarri, J. Palou Redorta, F. Rodríguez Escovar, S. Esquena Fernández, H. Villavicencio Mavrich. *Historia de la robótica: de Arquitas de Tarento al robot Da Vinci (Parte I)*. Fundació Puivert, Barcelona, primera edición, 2007.

[5] Steve Fulton y Jeff Fulton. *HTML5 Canvas*. O'REILLY, Beijing, segunda edición, 2013.

[6] Christopher Crick. *Rosbridge: ROS for NON-ROS Users*. Primera edición, 2017.

[7] J. Flores Rubio. *El SAE de TransMilenio: Sistema de Gestión de Flotas para el transporte masivo de Bogota*. Primera edición, 2005.

[8] Panorama conceptual sobre catkin, emitido el 20 de abril de 2017. Consultado en http://wiki.ros.org/catkin/conceptual_overview.

[9] Flash & The Future of Interactive Content. Consultado el 20 de junio en https://theblog.adobe.com/adobe-flash-update/.

[10] El proyecto Apache Xerces, emitido el 17 de agosto de 2016. Consultado en https://xerces.apache.org/charter.html.

[11] Wiki oficial de *rosbridge*. Consultado el 12 de junio en http://wiki.ros.org/rosbridge_suite.

[12] Web oficial de ROS. Consultado el 13 de junio en https://www.ros.org/.

[13] Web oficial de JSON. Consultado el 19 de junio en https://www.json.org/.