Universidad Zaragoza

Trabajo Fin de Máster

Asignación distribuida de tareas dinámicas en sistemas multi-robot

Distributed dynamic tasks assignment of multi-robot systems

Autor

Iñigo Etayo Gil

Directores

Eduardo Montijano

Danilo Tardioli

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2020

Agradecimientos

Antes de empezar con la redacción de este Trabajo Fin de Máster quiero agradecer a toda la gente que ha contribuido para que se pudiera llevar a cabo.

En primer lugar me gustaría agradecer a Eduardo Montijano y a Danilo Tardioli que como directores de este Trabajo han formado parte directa de su realización, mostrando interés y ofreciendo su ayuda en todo momento. Sus conocimientos y experiencia han sido parte fundamental del resultado que se ha conseguido, más aún teniendo en cuenta en las situaciones en las que se ha tenido que llevar a cabo este trabajo.

Por otra parte, agradecer al i3A por otorgarme la oportunidad de trabajar como investigador en sus laboratorios. Y en especial a los compañeros del grupo RoPeRT, quienes han compartido largas horas de trabajo en el laboratorio y han estado ahí para ayudarme en todas las complicaciones encontradas a lo largo de la realización de este trabajo.

Agradecer también a todos mis amigos y compañeros que durante esta etapa han aportado su granito de arena tanto a nivel personal como académico.

Por último, agradecer a toda mi familia por el apoyo aportado durante esta etapa y durante todos los años que hay detrás. Sin su ayuda y dedicación constante no podría haber llegado hasta donde estoy hoy.

A todos ellos, muchas gracias.

Resumen

Asignación distribuida de tareas dinámicas en sistemas multi-robot

Los sistemas multi-robot han experimentado en los últimos años un vasto desarrollo a nivel de investigación debido a sus cualidades para mejorar la realización de trabajos como la exploración y la vigilancia. Estos sistemas hacen posible un incremento de la eficiencia y la seguridad, al permitir el uso de conjuntos de robots para la realización de una o varias tareas en concurrencia en lugar del uso de un solo robot. Sin embargo, para que estos sistemas funcionen correctamente hay que solventar problemas relacionados con el control de los robots y su posicionamiento en el entorno, así como con la asignación de las tareas que cada uno de los robots tiene que desempeñar. Es en este último problema donde se centra el desarrollo de este trabajo.

En el presente Trabajo Fin de Máster (TFM) se desarrolla un algoritmo capaz de asignar las tareas a realizar por cada uno de los robots de una manera distribuida, es decir, cada uno de los robots toma decisiones de una manera independiente que dan lugar a un comportamiento óptimo colectivo sin la necesidad de una unidad central que lo dirija todo. El algoritmo empleado hace uso del método símplex, método matemático empleado para la resolución de problemas de optimización en los que se quiere minimizar o maximizar un coste o un beneficio.

Dentro de los problemas que existen al trabajar con sistemas distribuidos, este trabajo se centra en analizar los aspectos relacionados con la transmisión de información entre los robots, estudiando como la cantidad de información mandada y su selección influye en los resultados de la asignación. En el TFM se proponen diferentes políticas de envío de información, estudiando las ventajas e inconvenientes que supone el utilizar un mayor ancho de banda y el emplear diferentes criterios de selección de la información.

Por otra parte, en el TFM se ha prestado especial atención a los aspectos de implementación real del método, típicamente relegados a un segundo plano en este tipo de soluciones. El TFM incluye una implementación totalmente distribuida del algoritmo haciendo uso de la plataforma de desarrollo ROS. Esta implementación se ha evaluado tanto en un entorno de simulación realista como en el laboratorio del grupo de robótica utilizando tres robots para monitorizar dinámicamente a tres personas.

Índice General

Capítulo 1

Introducción

En la actualidad el uso de equipos multi-robot se está convirtiendo en una alternativa real para la realización de tareas tales como exploración, búsqueda y rescate, vigilancia, etc. Un ejemplo de esto son los grupos de robots utilizados por el ejército para la detección de minas antipersona o los robots utilizados por las agencias espaciales para la exploración espacial, como podría ser el vehículo motorizado que se desplaza por la superficie de Marte conocido como *rover* (Figura 1.1).

Durante la realización de estas actividades cada robot se encarga de una tarea individual que le ha sido asignada previamente para la consecución de la tarea global, teniendo en cuenta diferentes propiedades como puede ser su ubicación. Es en la necesidad de asignar estas tareas adecuadamente donde se enmarca este Trabajo Fin de Máster.

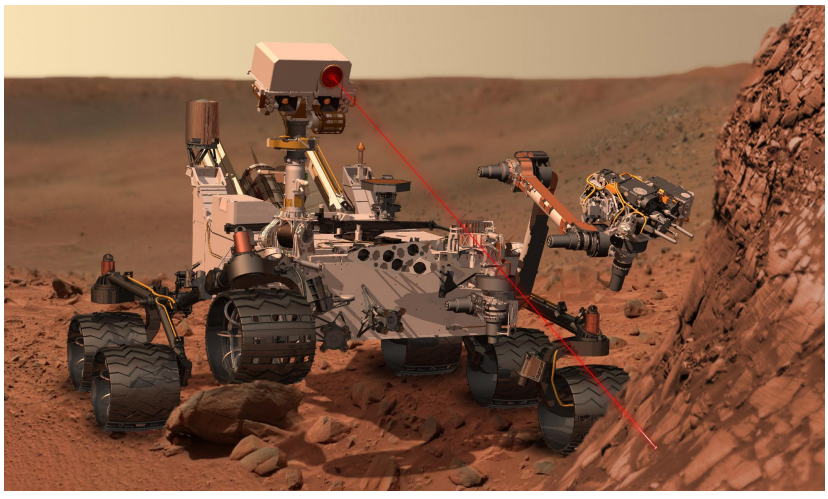

Figura 1.1: Ejemplo de robot de exploración en la superficie de Marte. *(https://es.wikipedia.org/wiki/Mars_rover)*

Para concretar el enfoque de este TFM, se considera a modo de ejemplo la retransmisión de un partido de baloncesto. Durante el tiempo de partido se han

instalado una serie de cámaras móviles alrededor del campo para analizar los movimientos de cada jugador. Estas cámaras pueden interpretarse como sistemas robóticos móviles con capacidad para tomar decisiones de observación. El número de cámaras es suficientemente elevado como para que una persona supervise esta vigilancia, por lo que se requiere que las cámaras sean capaces de decidir de forma autónoma a qué jugadores observar en cada momento para obtener el mayor detalle de la imagen y poder analizar mejor sus movimientos. En este TFM se quiere desarrollar ese algoritmo capaz de realizar esa asignación de objetivos a cada una de las cámaras sin la supervisión de una persona ni de un ordenador central.

1.1. Estado del arte

La asignación de tareas en entornos multi-robot se asocia normalmente a la resolución de un problema de optimización, en el que se busca una solución que minimice (o maximice) un determinado criterio de calidad global sobre el equipo en conjunto. Aplicando esto al ejemplo expuesto anteriormente sobre el partido de baloncesto, obsérvese la Figura 1.2 donde se han escogido tres cámaras que deben de vigilar a tres jugadores. La minimización de los costes en este problema iría asociada a la minimización global de la suma de las distancias entre cada cámara y el jugador asignado, así pues para la asignación de objetivos el resultado final se encontraría en la imagen de la derecha, consiguiendo así optimizar el encuadre del jugador en la cámara.

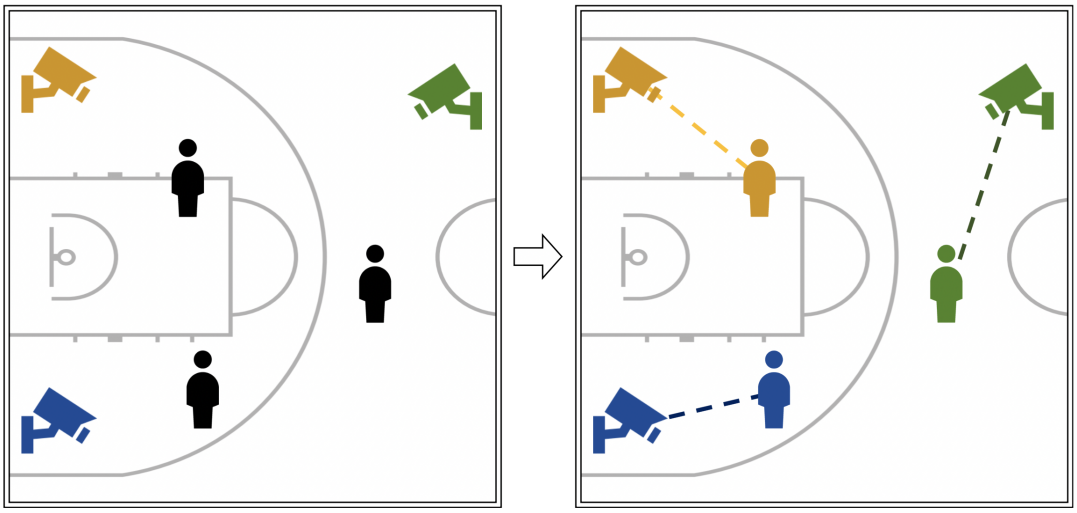

Figura 1.2: Ejemplo de asignación de objetivos según posición.

Durante años la resolución de este problema ha sido llevada a cabo de un modo centralizado, es decir, el proceso de cálculo se desarrolla en una unidad central, con conocimiento total de la información, a la que están conectados todos los robots y del

que reciben toda la información necesaria para realizar la tarea asignada, ya sea de manera física o remota. Debido a su interés más allá de la robótica, en la literatura se pueden encontrar algoritmos clásicos capaces de resolver este problema de manera óptima y eficiente, como pueden ser el algoritmo húngaro (Kuhn, 1955), el método símplex (Luenberger and Ye, 2008) o el algoritmo de Gross (Garfinkel, 1971).

Cuando el número de robots aumenta y se precisa más cantidad de información a trasmitir para el correcto funcionamiento del algoritmo, los sistemas centralizados se empiezan a encontrar con problemas de latencia y mayores requerimientos de procesamiento de datos así como de gestión de envío y recepción de la información. Este mismo problema se encuentra también en sistemas descentralizados en los que sigue existiendo una unidad superior que procesa la información para otros robots. Por ello es necesario buscar otras alternativas que van ligadas al uso de sistemas distribuidos, en los que cada uno de los robots es capaz de tomar decisiones como la asignación de su propia tarea a realizar sin la necesidad de conocer toda la información disponible (Figura 1.3). Además, en este tipo de configuraciones, la caída de uno de los nodos no impide el funcionamiento del resto de ellos. Ejemplos de ello se pueden encontrar en Chopra et al. (2017) donde se aplica el algoritmo húngaro, o en Burger et al. (2012) donde se hace uso del método símplex.

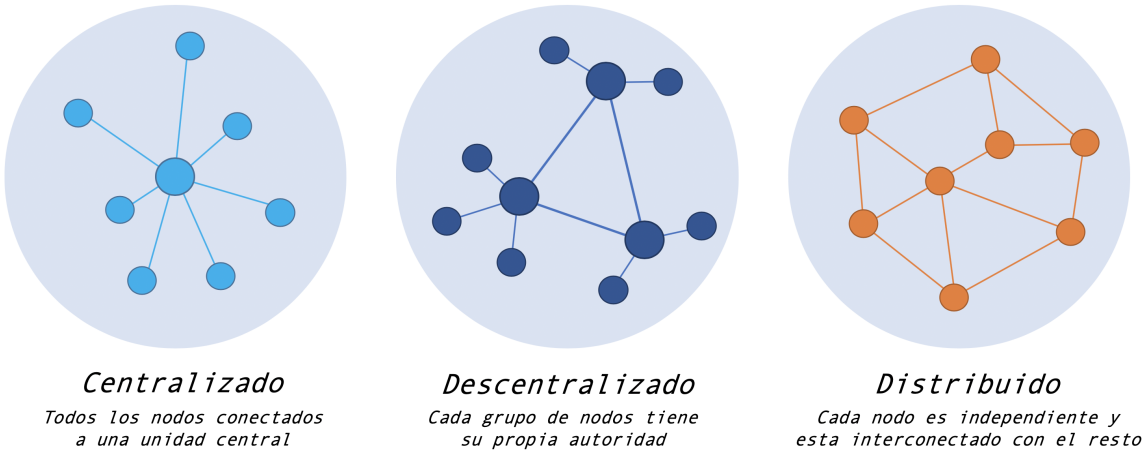

Figura 1.3: Comparación diferentes tipos de redes.

Actualmente la gran mayoría de algoritmos existentes en la literatura tratan de resolver el problema de manera distribuida, pero de un modo estático, es decir, la asignación de tareas se calcula una sola vez, por lo que para obtener una nueva asignación, si se ha originado un cambio en las condiciones del problema, es necesaria la reevaluación de la asignación y la ejecución del algoritmo completo. Debido a que la mayoría de los sistemas naturales tienen un componente dinámico en el tiempo, las últimas líneas de investigación se centran en solventar este problema sin la necesidad de

reinicializar el algoritmo; es aquí donde se encuentra un artículo que trata esta línea de investigación y se va a utilizar como referencia a lo largo de este TFM, (Montijano et al., 2019). En este artículo se presenta un algoritmo de asignación distribuida y dinámica basado en el algoritmo símplex, el cual permite la formulación del problema haciendo uso de costes variables en el tiempo sin necesidad de reinicialización. Adicionalmente, el algoritmo propuesto es capaz de obtener resultados cercanos a los que se obtendrían a través de un método centralizado, pero con la particularidad de que cada uno de los robots solo es conocedor de la información que recibe a través de los robots que se encuentran dentro de su radio de comunicación, sus vecinos. En este artículo se hace uso del algoritmo para la asignación de objetivos humanos en grupos de robots equipados con cámaras de vigilancia, un ejemplo similar al de la retransmisión del partido.

En el trabajo mencionado la información intercambiada entre los robots se escoge en base a la solución propuesta por el símplex. Esta información no se escoge de manera intuitiva, sino que respeta la selección tomada por el algoritmo, pero sin saber que hay detrás de esa elección. Se cree que para determinados costes y situaciones, la información elegida por el símplex no es la mejor y podría ser seleccionada de otras maneras con mejores resultados. Por esta razón se ha escogido como primera línea de investigación de este TFM el estudio de diferentes estrategias de comunicación de la información con el fin de, además de establecer criterios conocidos, mejorar los resultados obtenidos en Montijano et al. (2019). Adicionalmente, a vista de la falta de una experimentación distribuida real, ya que en Montijano et al. (2019) la experimentación se desarrolla de un modo distribuido simulado utilizando el software Matlab, se procederá a la implementación de un sistema distribuido en un laboratorio real. Este aspecto, el cual no suele considerarse primordial en la literatura de estos temas, supone un salto importante para llevar los sistemas distribuidos de la teoría a la realidad.

Es en este contexto en el cual se desarrolla este Trabajo Fin de Máster: “Asignación distribuida de tareas dinámicas en sistemas multi-robot”, dentro de un proyecto de investigación en colaboración con el departamento de Ingeniería de Sistemas y Automática de la Universidad de Zaragoza y el Instituto Universitario de Investigación en Ingeniería de Aragón (i3A), lugar donde se ha concedido al autor de este trabajo un contrato de investigación para la realización de prácticas extracurriculares para alumnos con alto rendimiento académico. Dentro de este instituto se ha trabajado concretamente con el grupo de investigación de Robótica, Percepción y Tiempo Real (RoPeRT) en el que están involucrados ambos directores de este trabajo y que tiene como líneas de investigación la robótica móvil, la localización y mapeado y la visión por computador.

1.2. Objetivos y Alcance

Este proyecto tiene como objetivo principal desarrollar un algoritmo de carácter distribuido capaz de realizar la asignación de tareas dinámicas en un entorno multi-robot. Los objetivos específicos que se han planteado para llegar a desarrollar este algoritmo son los siguientes:

- Estudio y comprensión del estado del arte relacionado con sistemas multi-robot, problemas de optimización mediante el uso del algoritmo símplex y con profundización en sistemas distribuidos.
- Estudio de diferentes estrategias de comunicación de la información entre robots teniendo en cuenta su posibilidad de implementarlas en el algoritmo.
- Implementación del algoritmo en una plataforma distribuida real.
- Realización de simulaciones a nivel computacional para la validación del funcionamiento del algoritmo. Posteriormente, experimentación de la plataforma diseñada tanto a nivel virtual, como a nivel real.

Para el estudio del problema se ha partido de la lectura de diferentes artículos de investigación relacionados con el algoritmo símplex, así como varios libros relacionados con la programación lineal y con la teoría de grafos. Entre los artículos tratados se ha centrado el estudio en Montijano et al. (2019), el cual se ha usado como idea inicial del algoritmo.

Se ha utilizado el código implementado en Montijano et al. (2019) en el software Matlab para comprender la secuencia del algoritmo y entender mejor el problema. A partir de ahí se han planteado una serie de propuestas de comunicación diferentes a las originales y se han implementado en el algoritmo. Para analizar el comportamiento de las propuestas se ha requerido el uso del Cluster Hermes debido a la magnitud de los experimentos, mucho más alta que los planteados en el artículo original. Con la gran cantidad de información recopilada se han estudiado diferentes métricas y se han evaluado los resultados de acuerdo a ellas.

Una vez estudiadas las diferentes propuestas de comunicación se ha ideado una arquitectura distribuida para poder utilizar el algoritmo desarrollado en robots reales utilizando la plataforma de desarrollo ROS (Robot Operating System) (ros.org). Esta arquitectura se ha diseñado desde cero ya que la arquitectura presente en Montijano et al. (2019) no era una arquitectura distribuida, sino una centralizada estructurada para que simulara la componente distribuida del algoritmo.

Para poder llevar a cabo la implementación de esta arquitectura se ha requerido el aprendizaje de los comandos de ROS, así como el lenguaje de programación *C++* junto con la librería externa *Eigen* para poder realizar cálculos matriciales. Adicionalmente, para que el sistema permitiera una cómoda y rápida visualización del estado del sistema, se ha estudiado el uso de *Stage* y de *RViz*, dos entornos utilizados junto con ROS para la simulación y visualización de los robots y de las asignaciones de las tareas.

Una vez desarrollada la arquitectura distribuida con las propuestas de comunicación implementadas en ella, se ha utilizado el software de Matlab para el análisis de los resultados y su posterior representación gráfica.

Por último, con los datos obtenidos se ha realizado una evaluación del algoritmo y se han comentado las conclusiones obtenidas tras la realización de este Trabajo Final de Máster.

1.3. Organización de la memoria

La memoria de este TFM se ha estructurado en base a 6 capítulos y 2 anexos. En el capítulo 1 se ha realizado una breve introducción al contexto en el que se desarrolla el trabajo, así como cual ha sido la motivación y los objetivos planteados. En el capítulo 2 se ha introducido el algoritmo planteado y se han explicado los fundamentos teóricos en los que se basa. En el capítulo 3 se han discutido diferentes estrategias de comunicación entre los robots para implementar en el algoritmo. El capítulo 4 se ha centrado en definir la arquitectura software encargada de calcular el algoritmo en los diferentes robots de manera distribuida y su posterior implementación en la plataforma ROS. En el capítulo 5 se han mostrado los resultados de las simulaciones y experimentos llevados a cabo para validar el modelo, y, por último, en el capítulo 6 se han comentado las principales conclusiones alcanzadas tras la realización de este Trabajo Fin de Máster y las posibles líneas futuras que se abren tras él. Adicionalmente se han incluido 2 anexos: en el anexo A se explican una serie de aspectos teóricos relacionados con la teoría de grafos y con el algoritmo símplex, y en el anexo B se explica de una manera más completa el funcionamiento de la plataforma de desarrollo ROS.

Capítulo 2

Asignación distribuida de tareas dinámicas

En este capítulo se define el problema de asignación de tareas distribuido y dinámico utilizando un grupo de robots móviles. Para la comprensión del capítulo, debido al uso de la teoría de grafos y las relaciones con el algoritmo símplex, se recomienda la lectura previa del Anexo A.

2.1. Formulación del problema

En este trabajo se va considerar un grupo de N robots modelado por un grafo conectado y no dirigido, $\mathcal{G} = (V, E)$. Cada uno de los robots se considerará un vértice del grafo y las aristas del grafo indicarán la posibilidad de comunicación entre cada par de robots. La finalidad de los robots es observar a un conjunto de N objetivos móviles a lo largo del tiempo. Asemejando este problema con un problema de programación lineal, se puede expresar matemáticamente de la forma

$$\begin{aligned} \max_{x_{ij}(t)} \quad & \sum_{t} \sum_{i=1}^{N} \sum_{j=1}^{N} x_{ij}(t) r_{ij}(t), \\ \text{subject to} \quad & \sum_{j=1}^{N} x_{ij}(t) = 1, \qquad \forall i \in \{1, \ldots, N\}, \\ & \sum_{i=1}^{N} x_{ij}(t) = 1, \qquad \forall j \in \{1, \ldots, N\}, \\ \text{and} \quad & x_{ij}(t) = \{0, 1\}, \qquad \forall i, j \in \{1, \ldots, N\}, \end{aligned} \quad , \tag{2.1}$$

donde la asignación de cada uno de los robot con el objetivo a vigilar se define según $x_{ij}(t)$ $(i, j \in \{1, \ldots, N\})$, donde $x_{ij}(t) = 1$ indica que el objetivo j está siendo observado por el robot i en el instante de tiempo t. Las restricciones mostradas en (2.1) establecen que cada objetivo solo puede ser observado por un robot y que cada robot solo puede observar a un objetivo en cada instante de tiempo.

La ecuación a maximizar en el problema está definida en función de los parámetros $r_{ij}(t)$, que indican la recompensa que recibe cada robot por observar a cada uno de los objetivos a lo largo del tiempo, la cual puede ser definida de acuerdo a multitud de parámetros. En este trabajo ha sido definida según la posición relativa entre robot y objetivo de forma exponencial

$$r_{ij}(t) = e^{-\|\boldsymbol{p}_i(t) - \boldsymbol{q}_j(t)\|}, \tag{2.2}$$

pero podría haberse utilizado cualquier otro tipo de variable para recompensar a los robots. Conforme la distancia relativa entre un objetivo y un robot disminuye, también lo hace el exponente de la ecuación y, por lo tanto, se expresa un incremento de la recompensa de dicho robot para observar a ese objetivo. Cada robot es capaz de calcular esta recompensa para los valores asociados a su posición ($r_{ij}(t) \quad \forall j$), pero no para las de los otros robots, valores que solo podrá conocer a través de la información que los robots pueden ir mandando a lo largo del tiempo.

El problema también se puede formular con notación vectorial,

$$\begin{array}{ll} \text{maximize} & \sum_t \mathbf{r}(t)^T \mathbf{x}(t), \\ \text{subject to} & \mathbf{A}\mathbf{x}(t) = \mathbf{b}, \\ \text{and} & \mathbf{x}(t) \in \{0,1\}^{N^2}, \end{array} \tag{2.3}$$

donde $\mathbf{A}$ y $\mathbf{b}$ expresan las restricciones del problema y se observa que la matriz $\mathbf{A}$ está formada por restricciones linealmente dependientes, por lo que se ha cogido una partición de *2N-1* filas de $\mathbf{A}$ para delimitar el problema y permitir la independencia entre las restricciones, dando así lugar a las matrices

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{I}_N \bigotimes \mathbf{1}_N^T \\ \mathbf{1}_N^T \bigotimes \boldsymbol{I}_{N-1} \end{pmatrix} \in \mathbb{R}^{2N-1 \times N^2}, \quad \text{and} \quad \boldsymbol{b} = \mathbf{1}_{2N-1} \tag{2.4}$$

conocidas por todos los robots ya que todos ellos conocen la identificación de los robots y de los objetivos a lo largo de la red gracias a una primera inicialización.

Dado que estamos ante un problema de programación lineal, la primera idea sería pasar a resolverlo a través del método símplex ya explicado, pero en la formulación del problema encontramos dos problemas que nos impiden la utilización de este algoritmo en su manera primitiva, estos son:

– Se quiere implementar la solución de este problema de una manera distribuida para mejorar su alcanzabilidad, pero en este problema la información conocida por cada uno de los robots (recompensas, $r_{ij}(t)$), no es completa y por tanto la resolución del algoritmo símplex en su manera tradicional requeriría conocer todo el problema por cada uno de los robots.

– Las recompensas asociadas a las asignaciones de los robots ($r_{ij}(t)$) son variables en el tiempo ya que los objetivos se van desplazando y por lo tanto estos valores se ven afectados por ese desplazamiento según su definición (eq. 2.5). Así pues se requiere un algoritmo capaz de tener en cuenta la dinámica de este problema, cosa que el símplex tradicional no tiene en cuenta y daría lugar a soluciones desfasadas en el tiempo acordes a recompensas obsoletas.

2.2. Símplex distribuido dinámico

Para resolver el primer problema se va a hacer uso de una variante de símplex distribuido encontrada en Burger et al. (2012). Este algoritmo se basa en el método matricial explicado en el Anexo A.

Partiendo del problema de la ecuación 2.3 se transforma este a su forma canónica, es decir, se convierte el problema en un problema de minimización y se tratará a partir de ahora las recompensas de la forma

$$r_{ij}(t) = 1 - e^{-\|\boldsymbol{p}_i(t) - \boldsymbol{q}_j(t)\|}, \tag{2.5}$$

a continuación se obtiene una base inicial, $\mathbf{B}^{[\mathbf{k}]}$, para cada robot asociada a las variables no básicas definidas según el método big-M. Este superíndice k hace referencia a cada uno de los robots, donde se utiliza el símplex de manera independiente. Para que estas variables *slack* de las columnas artificiales añadidas no influyan en la solución se les asigna una recompensa de valor lo suficientemente alto como para que no alteren la solución.

Esta base inicial, $\mathbf{B}^{[\mathbf{k}]}$, y sus recompensas asociadas, $\mathbf{r}^{[\mathbf{k}]}_{\mathbf{B}^{[\mathbf{k}]}}$, serán los valores que transmitirá cada uno de los robots a sus vecinos, para que éstos sean capaces de actualizar las recompensas del problema conforme el entorno cambia. Cabe destacar que tanto la matriz $\mathbf{A}$, como el vector $\mathbf{b}$, no es necesario mandarlos dentro del mensaje ya que son idénticas para todos los robots una vez se haya realizado una inicialización sabiendo cada uno el índice de cada robot y de cada objetivo.

Una vez obtenida una base inicial, y establecido el criterio de envío de datos entre los robots para asegurar la distribución de la información, el algoritmo simplemente necesita calcular el método símplex de manera iterativa mientras actualiza la información del problema con los datos recibidos y con los conocidos. De este modo el algoritmo será capaz de converger a una solución óptima igual para todos los robots en un tiempo finito.

En cuanto a las variaciones en el tiempo de $r_{ij}(t)$, cada robot es capaz de actualizar las recompensas asociados a su observación, pero el resto tienen que ser renovadas

periódicamente a través de la información intercambiada por los vecinos. Si solo se tuviera en cuenta el valor de las recompensas tal cual se van recibiendo por cada robot, podría dar lugar a soluciones subóptimas si se diera, por ejemplo, el caso de recibir el mismo valor de $r_{ij}(t)$ a través de dos vecinos, pero recorriendo dos caminos de propagación de la información distintos y por consiguiente tener valores de $r_{ij}(t)$ desactualizados si el último valor recibido corresponde al camino más largo. Para solventar esta cuestión se hace uso de un valor de antigüedad asociado a cada una de las recompensas conocidas por cada robot de la forma:

$$a_{ij}(t) = \begin{cases} 0 & \forall (i,j) \in \mathbf{A}^{[k]} \\ a_{ij}(t-1)+1 & \text{otherwise} \end{cases} . \tag{2.6}$$

Utilizando este parámetro el algoritmo es capaz no solo de actualizar únicamente las recompensas más recientes, sino también de despreciar los valores con una antigüedad suficiente como para que no sean de utilidad a la hora de obtener una solución óptima,

$$r_{ij}(t) = \begin{cases} r_{ij}(t) & \forall (i,j) \in \mathbf{B}^{[k]} \\ \hat{r}_{ij}(t) & (i,j) \notin \mathbf{B}^{[k]} \wedge \\ & \min a_{ij}^{[\mathcal{N}_k \cup k]}(t) \leq N \\ -1 & \min a_{ij}^{[\mathcal{N}_k \cup k]}(t) > N \end{cases} . \tag{2.7}$$

En la ecuación anterior se observa que para los valores no pertenecientes a la base óptima, se hace uso de una predicción de las recompensas ($\hat{r}_{ij}(t)$). Esta predicción se utiliza para dar valor a las recompensas asociadas a robots no vecinos y de los cuales se pueden dar iteraciones sin conocer su valor. Asumiendo que las recompensas siguen una evolución suave en el tiempo, la predicción de estas recompensas se calcula siguiendo una extrapolación lineal según:

$$\hat{r}_{ij}(t) = r_{ij}(t_{ij}) + a_{ij}(t)(r_{ij}(t_{ij}) - r_{ij}(t_{ij}-1)) \tag{2.8}$$

A continuación se muestra el algoritmo completo tal y como se describe en Montijano et al. (2019). Este algoritmo al ser distribuido sería necesario ejecutarlo de manera indefinida en cada uno de los robots, de este modo cada uno de ellos sería capaz de recalcular la asignación óptima conforme los objetivos se vayan desplazando y éste reciba información de sus robots vecinos.

Algorithm 1 Asignación Distribuida Dinámica (Robot k)

1: Usar símplex para obtener la base inicial, $\mathbf{B}^{[k]}$, usando $\mathbf{A}^{[k]}$
2: **while** true **do**
3: Calcular recompensas propias $r_{kj}, j = 1, \ldots, N$
4: Actualizar edad de $r_{ij}(t)$
5: Mandar índices de $\mathbf{B}^{[k]}, \mathbf{r}^{[k]}_{\mathbf{B}^{[k]}}$ y $\mathbf{a}^{[k]}_{\mathbf{B}^{[k]}}$
6: Recibir información de robots vecinos
7: Actualizar recompensas $r_{ij}(t)$
8: Calcular una nueva base, $\mathbf{B}^{[k]}$, usando LexSimplex
9: Asignar j tal que $x_{ij} \in \mathbf{x}_{\mathbf{B}^{[k]}}$ y $x_{ij} = 1$
10: **end while**

Capítulo 3

Estrategias de comunicación de la información

El envío de datos entre los robots puede parecer una tarea trivial y sin influencia en la asignación cuando se trabaja con grupos de pocos robots, pero cuando se empieza a operar en entornos donde el número de robots crece considerablemente, la cantidad de datos crece de manera lineal para cada robot y de manera cuadrática para la red completa y puede llegar a convertirse en uno de los puntos más vulnerables del sistema debido a las limitaciones del ancho de banda de la red utilizada.

Al tener tanta información es muy importante establecer un criterio de selección apropiado para conseguir los mejores resultados con la mínima cantidad de información transmitida. El criterio elegido va a repercutir de manera sustancial en el desarrollo del modelo y, por lo tanto, es un aspecto clave a analizar.

Es por esta razón que en este capítulo se van a proponer estrategias de comunicación de la información diferentes a las utilizadas en Montijano et al. (2019). Para ello se va a tener en cuenta tanto la cantidad de datos intercambiado entre los robots, como la selección de estos datos para mejorar el algoritmo inicial en lo máximo posible.

3.1. Estrategia original

En el Capítulo 2.2 ya se hacía mención de la política de envío de datos que se seguía en Montijano et al. (2019), donde se había definido un mensaje que cada uno de los robots iba mandando a sus vecinos en cada una de las iteraciones del algoritmo. Este mensaje estaba constituido por los índices de la matriz $\mathbf{B}^{[k]}$, es decir, los índices de la matriz $\mathbf{A}$ que hacen referencia a sus columnas que forman la matriz $\mathbf{B}^{[k]}$, así como las recompensas asociadas a esos índices ($\mathbf{r}^{[k]}_{\mathbf{B}^{[k]}}$) y sus respectivas antigüedades ($\mathbf{a}^{[k]}_{\mathbf{B}^{[k]}}$).

Hasta ahora solo se ha hecho referencia a la tipología de este mensaje y los datos que engloba, pero conocido el problema también se puede identificar tanto el tamaño

que tendrá el mensaje en función del número de robots, como la composición de este. Según la tipología del problema se sabe que el número de restricciones a cumplir es igual a *2N-1*, que coincide con el número de filas de la matriz $\mathbf{A}$. Conociendo este dato se fija el número de variables básicas en *2N-1*, y por consiguiente, el número de valores que compondrán $\mathbf{B}^{[k]}$, $\mathbf{r}^{[k]}_{\mathbf{B}^{[k]}}$ y $\mathbf{a}^{[k]}_{\mathbf{B}^{[k]}}$ será también de *2N-1*.

Para hacerse una idea de cómo estaría formado este mensaje, en la Figura 3.1 se muestra un ejemplo de un sistema con $N = 3$. En las primeras líneas se presenta la asignación, x_{ij}, y las recompensas y antigüedades conocidas por el robot 1. Esto se podría deducir al ver que el valor de antigüedad de los tres primeros términos de a_{ij}, asociados al robot 1, es igual a 0. Esta asignación declara que el robot 1 observa al objetivo 1, y asume que el robot 2 observa al objetivo 3 y el robot 3 al objetivo 2 y, por consiguiente, en las siguientes líneas que determinan la composición del mensaje que mandaría el robot 1 a sus vecinos, los primeros N términos harían referencia a las posiciones de la asignación $(1, 6\, y\, 8)$ y los otros *N-1* términos, determinados por el propio algoritmo, completarían la base del mensaje.

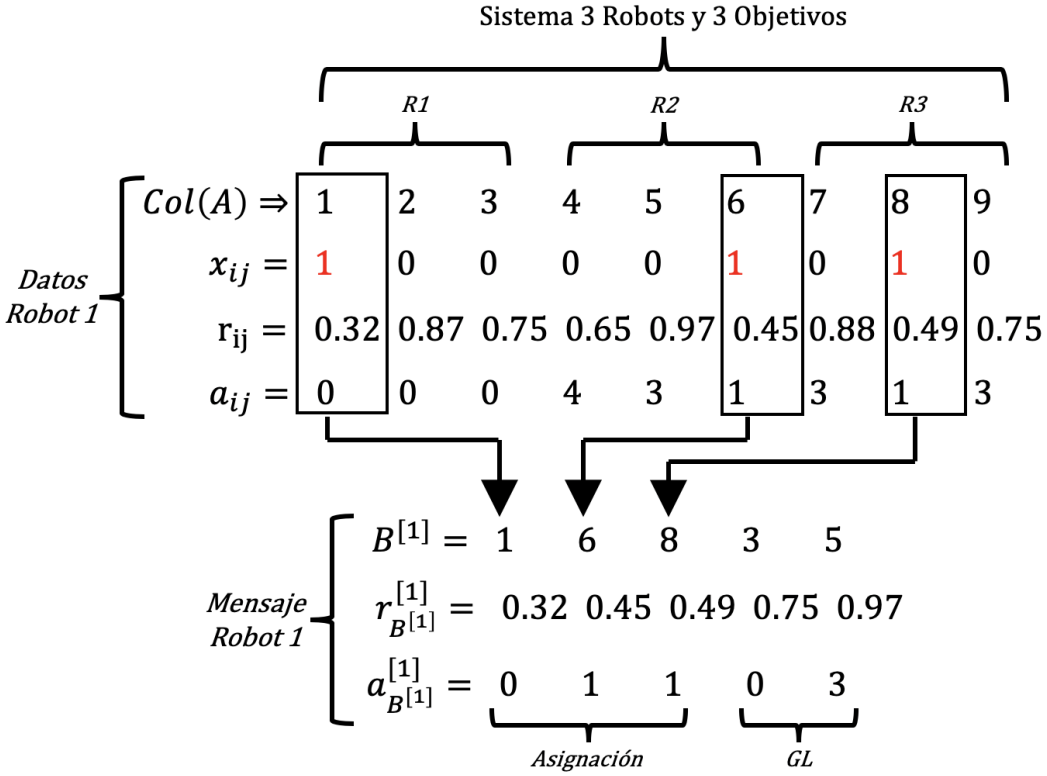

Figura 3.1: Ejemplo de asignación y composición del mensaje para $N = 3$.

En un primer análisis se podría deducir que si aumentáramos el tamaño del mensaje, es decir, en vez de mandar *2N-1* valores de r_{ij}, se mandara un número mayor, cada uno de los robots sería capaz de acceder a más información del entorno y ésta a su vez más reciente. De esta forma se conseguiría un modelo más próximo a un sistema

centralizado con información completa, el cual sería capaz de alcanzar una solución óptima más rápido. El inconveniente, por otro lado, vendría de la mano de lo que se ha hablado al inicio de este capítulo: la dificultad de tratar grandes cantidades de información y, por consiguiente, la necesidad de establecer un punto de equilibrio entre la cantidad de información mandada para aproximarse a un modelo centralizado y el ancho de banda que esta ocupa para no saturarlo.

Otra vía de mejora del algoritmo se hace evidente una vez detallada la composición de $\mathbf{r}^{[k]}_{\mathbf{B}^{[k]}}$. Por definición del problema, cada objetivo solo puede ser observado por un robot y cada robot solo puede observar a un objetivo; esto impone las restricciones del problema y convierte a la solución de asignación en un vector de ceros y unos en el que solo puede haber una cantidad de unos igual al número de robots. Dado que el mensaje tiene un tamaño de *2N-1*, esto da lugar a un total de *N-1* valores del mensaje que no hacen referencia a la asignación del problema y por tanto se podrían asimilar a grados de libertad del mensaje. En el ejemplo de la Figura 3.1 serían los valores de las dos últimas columnas del mensaje.

En la propuesta original de Montijano et al. (2019) la selección de estos valores libres se hace en base a la solución propuesta por el símplex, es decir, el símplex selecciona los valores libres que componen la base $\mathbf{B}^{[k]}$ con el fin de optimizar el resultado de la asignación, pero sin seguir un criterio fijo de selección. Si por el contrario hiciéramos uso de esos grados de libertad de acuerdo a un criterio definido, por ejemplo seleccionando los valores con una mayor recompensa (r_{ij}) o una menor antigüedad (a_{ij}), se podrían conseguir variaciones en los resultados, sabiendo además concretamente la procedencia del cambio.

3.2. Estrategias propuestas

Se van a proponer una serie de estrategias haciendo uso de manera conjunta de las dos vías de mejora explicadas con anterioridad. Para ello primero para cada estrategia propuesta se va a establecer el número de datos que se van a mandar, para así estimar un límite máximo de ancho de banda, y, a continuación, se seleccionará un criterio para la selección de los valores libres a mandar, si se diera el caso.

Propuesta 1: Solo mandar asignación

En esta primera propuesta no va a ser necesario definir un criterio de selección de los grados de libertad ya que solo se van a mandar N valores entre los robots. Estos valores corresponden a los de la asignación y por lo tanto no habrá grados de libertad en el mensaje (ver Figura 3.2).

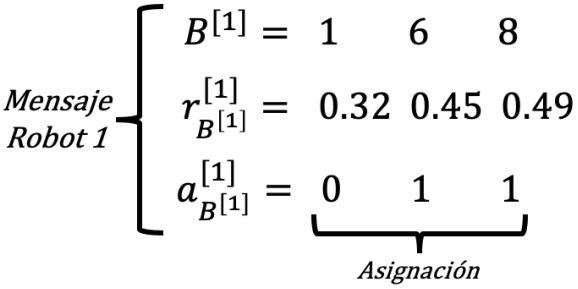

Figura 3.2: Ejemplo de composición del mensaje para $N = 3$ usando la propuesta 1, solo mandar asignación.

Con esta propuesta se quiere comprobar si, como sugiere la intuición, el tamaño del mensaje es relevante para la convergencia del modelo, y que su reducción empeora gravemente los resultados obtenidos al limitar la información a la que pueden acceder cada uno de los robots y alejándose así el modelo de un sistema centralizado con información total.

Propuesta 2: Asignación y datos propios

Para la segunda propuesta se ha vuelto a establecer *2N-1* como el tamaño del mensaje, pero se ha establecido un criterio de selección para los grados de libertad diferente del propuesto en Montijano et al. (2019),. El criterio elegido define estos *N-1* valores como los valores asociados al robot que manda el mensaje, es decir, cada robot además de la asignación va a mandar las recompensas asociadas a su propia posición, las cuales tienen una antigüedad de *0*. En la Figura 3.3 se muestra un ejemplo con la misma asignación que en la Figura 3.1, pero aplicando esta nueva propuesta.

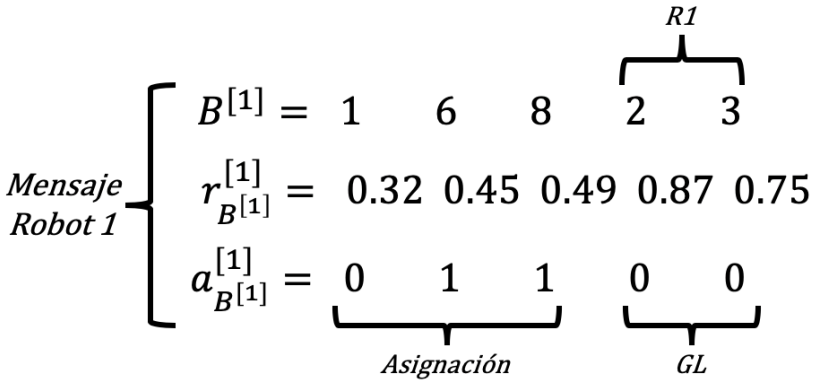

Figura 3.3: Ejemplo de composición del mensaje para $N = 3$ usando la propuesta 2, mandar asignación y datos propios.

La idea de esta propuesta pretende mantener en la red de comunicación los datos más recientes posibles, es decir, si a lo largo de las transmisiones se manda información

no relevante seleccionada por el propio algoritmo símplex, se puede llegar a mandar información antigua. Por el contrario, si en cada iteración, cada robot manda su propia información que acaba de calcular, se priorizan los datos más recientes y de este modo la información recibida por los robots difícilmente será descartada por ser anticuada.

Propuesta 3 y 4: Aumentar tamaño del mensaje

La propuesta 3 y 4 alteran el número de valores a enviar que se incrementa hasta *3N-1* para examinar las mejoras de los resultados al mandar más información, pero sin llegar a tener una comunicación completa entre robots.

Para la propuesta 3 se ha seleccionado un criterio de mínima recompensa para la selección de los grados de libertad del mensaje. En la Figura 3.4 se aplica esta propuesta al ejemplo anterior.

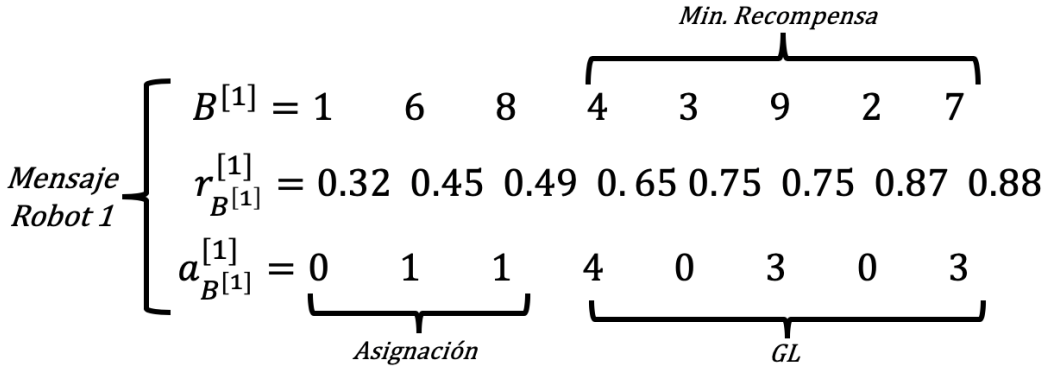

Figura 3.4: Ejemplo de composición del mensaje para $N = 3$ usando la propuesta 3, mandar más información con un criterio de mínima recompensa.

Por el contrario, en la propuesta 4 el criterio se basa en mandar los valores más recientes, es decir, los de mínima antigüedad entre los datos de cada robot. Se puede deducir que entre estos valores, se encontrarán los asociados a cada robot, debido a que la antigüedad de estos es siempre igual a *0*. Un ejemplo de esta propuesta es mostrado en la Figura 3.5

Propuesta 5: Mandar toda la información conocida

Como última propuesta se van a mandar todos los datos conocidos por cada robot para analizar los resultados que se podrían alcanzar si no se tuviera en cuenta el ancho de banda a ocupar por el mensaje y sus posibles inconvenientes en el sistema (ver Figura 3.6).

En la Tabla 3.1 se muestra a modo de resumen la comparativa entre las diferentes estrategias propuestas, tanto en función de la cantidad de datos transmitidos, como del criterio elegido para seleccionar estos datos.

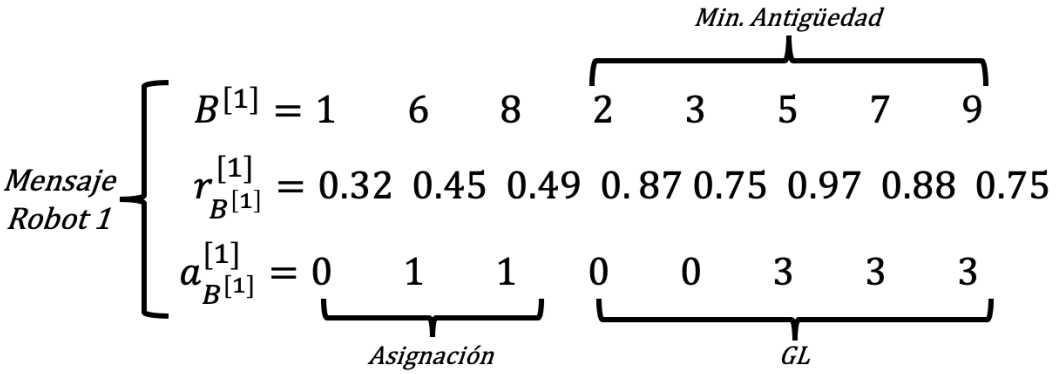

Figura 3.5: Ejemplo de composición del mensaje para $N = 3$ usando la propuesta 4, mandar más información con un criterio de mínima antigüedad.

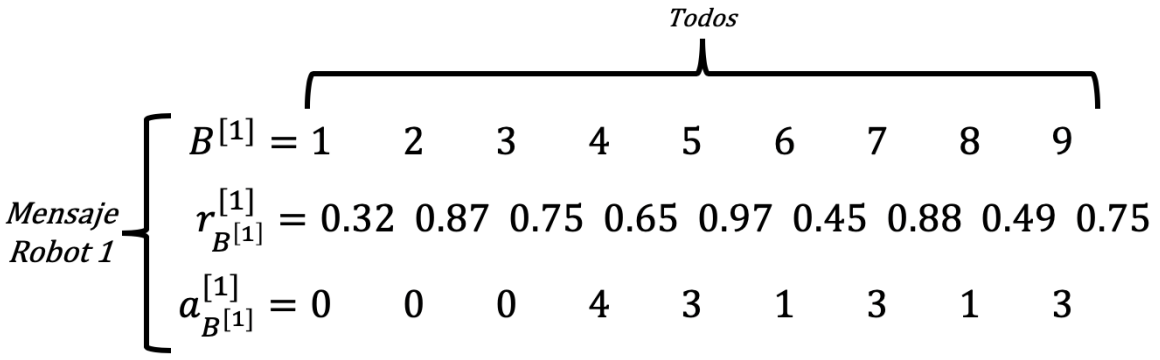

Figura 3.6: Ejemplo de composición del mensaje para $N = 3$ usando la propuesta 5, mandar toda la información conocida.

Nº	Estrategia	Tamaño del mensaje	Criterio
0	Original	2N - 1	Óptimo Simplex
1	Asignación	N	Solo Asignación
2	Asig. + Propios	2N - 1	Asignación + Datos propios
3	Extra coste	3N - 1	Min. recompensa
4	Extra edad	3N - 1	Min. antigüedad
5	All	N^2	Toda la información

Tabla 3.1: Resumen comparativo de las diferentes estrategias de comunicación propuestas.

Capítulo 4

Implementación en plataforma real

Hasta el momento solo se han mencionado aspectos teóricos del problema de asignación, pero no se ha tratado ningún aspecto relativo a su implementación en una plataforma real. Uno de los hitos principales de este trabajo es el diseño de una arquitectura software capaz no solo de realizar la asignación de las tareas a través del método símplex de un modo distribuido, sino también de la administración de todo el tráfico de información entre los robots.

En este capítulo se va a tratar de explicar e ilustrar la arquitectura diseñada. Desde los requerimientos previos que esta debe cumplir, hasta su confección y su funcionamiento dentro de la plataforma ROS debidamente en profundidad en el Anexo B.

4.1. Plataforma ROS

Para implementar esta arquitectura se va a hacer uso de la plataforma de desarrollo ROS (Robot Operating System) que está concebida para el desarrollo de software de robots a través de un sistema compuesto por una serie de herramientas, bibliotecas y convenciones ordenadas de un modo jerárquico.

La cualidad principal de ROS, y por la que se ha elegido para realizar la implementación, es la facilidad de desarrollo de entornos multi-robot mediante la utilización de herramientas y entornos ya diseñados por otros usuarios. En la Figura 4.1 se observa el papel de ROS en estos entornos, en los que su función principal es regir y administrar todo el intercambio de información entre los diferentes elementos que componen el sistema completo. Entre estos elementos se destacan:

– Nodos: Procesos que realizan cálculos dentro de ROS y están asociados normalmente a la realización de funciones de cálculo o de control de los robots.

– Topics: Canales de información para intercambio de datos entre nodos. Se basan en un sistema editor-suscriptor (publisher-subscriber en inglés) en los que los nodos pueden definirse como editores de ese topic, es decir, mandan información a través del topic, como suscriptores, leyendo la información publicada por el editor, o funcionar tanto como editor y suscriptor. Estos canales se crean a través de conexiones TCP/IP generalmente.

– Mensajes: Tipos de datos de ROS que utilizan los nodos al subscribirse o publicar en un topic. Definen la estructura de datos de los topics.

– Herramientas de simulación, visualización de resultados, control de robots, etc., formadas por conjuntos de nodos.

– Elementos externos: Extensiones del entorno de ROS para aumentar su funcionalidad, por ejemplo, mediante un vínculo con Matlab y Simulink.

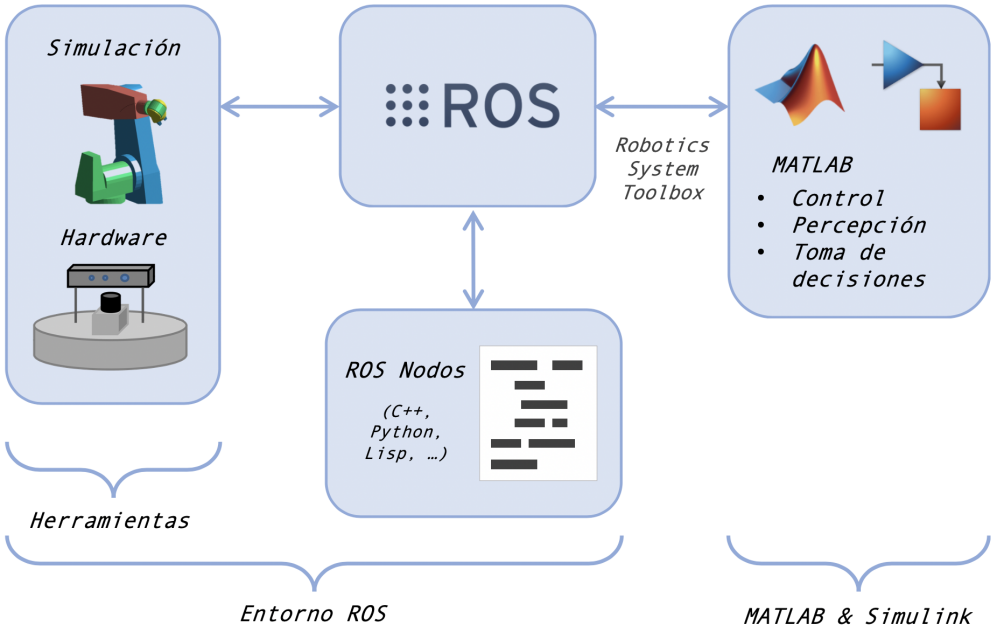

Figura 4.1: Esquema de los principales componentes y herramientas de la plataforma ROS.

El papel de ROS también es fundamental para facilitar las tareas relacionadas con el funcionamiento de los robots, ya que se encarga del control de bajo nivel de los robots, la abstracción del hardware, el flujo de mensajes entre los diferentes componentes, el control de los actuadores, etc. Todo esto permite que para la creación de un entorno sea necesario la diferenciación y el desarrollo de los diferentes nodos que se encargan de cada una de las tares que se quieran conseguir por parte de los robots. Para su creación

se utilizan principalmente los lenguajes de programación *C++* y *Python*. Por último, una vez definidos los robots a utilizar y sus funcionalidades en el entorno, se pueden añadir diferentes herramientas para, por ejemplo, la realización de simulaciones o su control. Estas herramientas pueden ser creadas por el propio usuario o utilizar las ya preexistentes creadas por otros usuarios como pueden ser:

- Herramientas de simulación de entornos de robots, como las herramientas *Stage* y *Gazebo*, capaces de crear un mundo virtual con una serie de robots con sus correspondientes sensores y actuadores.
- Herramientas de visualización de los robots, como *RViz* que permite la visualización en 3D de los los datos generados en los nodos y de los resultados recogidos por los sensores de los robots.
- Herramientas para el control de robots, de cámaras, etc., normalmente desarrolladas por los propios fabricantes de los robots para facilitar el desarrollo y uso de sus productos.

4.2. Diseño de la arquitectura del sistema

Partiendo de que la arquitectura a desarrollar es de tipo distribuido, hay que diferenciar por un lado los diferentes elementos del sistema, es decir, los diferentes robots independientes, y por otro lado que información se distribuye a cada uno de estos robots. En la Figura 4.2 se muestra una idea inicial de esta arquitectura donde se muestra en verde la información que va a circular por la red, la posición de los objetivos (necesaria debido a que en este trabajo no se han considerado los problemas de percepción de los objetivos), que tienen que conocer todos los robots, y los mensajes necesarios para realizar la asignación. La figura también muestra cuáles serán las principales tareas a desempeñar por cada uno de los robots: el posicionamiento para observar al objetivo y el cálculo de la asignación.

4.2.1. Requisitos de la arquitectura

Para identificar las herramientas y los nodos que se van a tener que desarrollar para crear la arquitectura se va a centrar la vista en el nodo principal de asignación, el cual será el encargado de realizar el cálculo de la asignación para cada uno de los robots. En la Figura 4.3 se muestra cómo debería de ser la información de la que parte este nodo y cual sería la información que es capaz de generar.

Si se identifican los diferentes componentes del esquema, tomando como nodo de asignación el correspondiente al robot k, se tendrá en primer lugar como dato de entrada

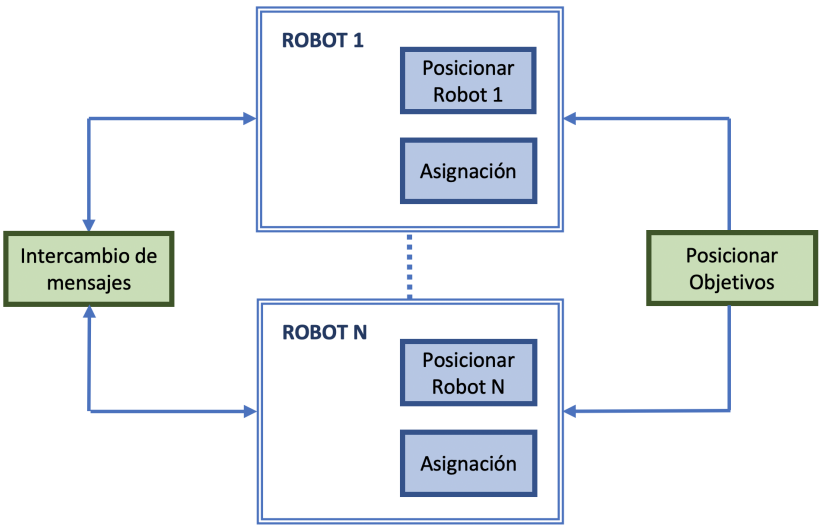

Figura 4.2: Arquitectura de alto nivel del sistema distribuido.

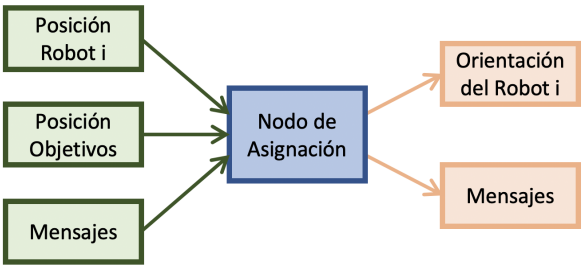

Figura 4.3: Entradas y salidas del nodo de asignación.

la posición de este robot y la posición de los diferentes objetivos. Con esta información el nodo será capaz de calcular las recompensas del robot k ($r_{kj}\ \forall\ j$) para, a continuación, actualizar esas recompensas con la información proveniente de los otros robots a través de los Mensajes. Con estos datos el nodo tendrá suficiente información como para calcular la asignación del robot k y ordenarle al robot que cambie su posición para orientarse respecto al objetivo asignado. Por último, con la información generada en el cálculo del símplex, el nodo mandará a los demás robots la información adecuada a través del Mensaje.

Tras reconocer estos componentes y marcando la necesidad de crear un entorno de simulación y visualización de los robots, se pueden establecer los requerimientos de nodos y de herramientas para la arquitectura a diseñar. Algunos de estos ya se tenían en cuenta en la arquitectura empleada en el artículo original (ver Figura 4.4) por lo que los requerimientos finales serán:

– Creación de un entorno de simulación multi-robot. Se hará uso de la herramienta *Stage* partiendo de un script ya existente.

– Visualización de los resultados. Tanto a nivel de posicionamiento de los robots en

un mapa preestablecido como en la visualización de la orientación y asignación de estos. Para esta tarea se hará uso de la herramienta *RViz*.

- Obtención de las recompensas, $r_{ij}(t)$. En este caso se relacionarán las recompensas con la posición relativa entre robot y objetivo.

  - Posicionamiento de los objetivos. Al igual que en Montijano et al. (2019) se utilizará un sistema de localización interior inalámbrico (*Decawave*) basado en una localización por triangulación tag-anchor similar a como funciona un sistema de posicionamiento GPS. En este aspecto cabe destacar que podrían implementarse diferentes tipos de técnicas de localización de los objetivos, incluso algún sistema de detección basado, por ejemplo, en técnicas de *machine learning*, pero todo estos aspectos quedan fuera del alcance de este trabajo.

  - Posicionamiento de cada robot. Partiendo de un mapa pre-establecido, cada uno de los robots se localizará mediante un algoritmo AMCL (Adaptive Monte Carlo Location) (Thrun et al., 2000) que estima la posición y orientación del robot utilizando un filtro de partículas gracias a un sensor LiDAR incorporado en el robot (*Hokuyo*) conforme este se va moviendo por el mapa conocido.

- Cambio de la orientación de los robots. Para ello se hará uso del nodo *Kobuki* diseñado por el fabricante de los robots como capa de abstracción de hardware capaz de realizar movimientos en el robot a partir de comandos de velocidad lineal y angular basados en odometría.

- Envío y recepción de mensajes entre los robots. Uso de un topic, *Mensajes*, que será enviado y recibido por cada robot.

- Nodo de asignación, implementado en lenguaje *C++* junto con la librería *Eigen* para cálculo matricial.

- Visualización de las cámaras de los robots.

4.2.2. Diseño final

Conocidos estos requisitos se ha desarrollado la arquitectura mostrada en la Figura 4.5. A simple vista se podría decir que se asemeja a la mostrada en la Figura 4.4, pero eso solo se debe a que varias partes de la arquitectura, relacionadas con el propio robot y con el posicionamiento de estos, son las mismas. Por contra, toda la parte relacionada

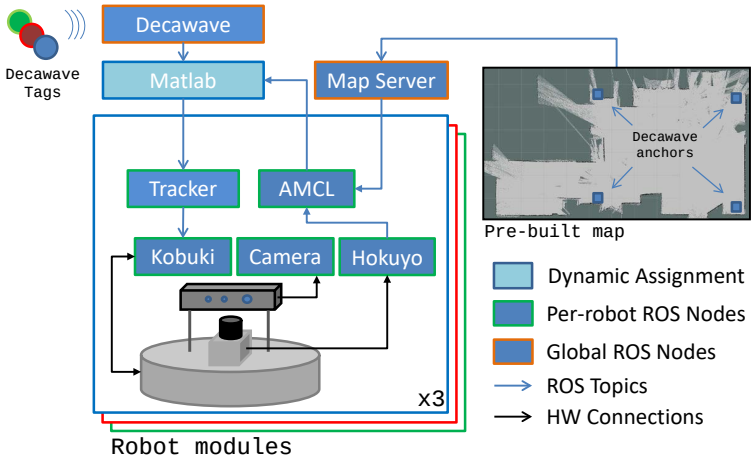

Figura 4.4: Arquitectura utilizada en el artículo original.

con la asignación de tareas es completamente diferente. En la idea original se hacía uso del software Matlab, que era quien, conocida toda la información, realizaba el cálculo del símplex utilizando parte de la información simulando un proceso distribuido. Una vez realizado este cálculo, mandaba la información correspondiente a los diferentes robots. Por el contrario, en la arquitectura diseñada, el proceso de asignación se realiza dentro de cada uno de los robots, en el nodo *Asignación*, siendo así un proceso distribuido real y no necesitando el uso de un software externo centralizado para este cálculo, sino que cada robot es capaz de realizar la asignación ejecutando ese nodo independientemente.

4.2.3. Funcionamiento dentro de ROS

Para entender el funcionamiento de la arquitectura primero se tiene que hacer una diferenciación a nivel global y a nivel individual de cada robot. A nivel global se identifica el nodo encargado del posicionamiento de los objetivos, *Decawave*, que calcula la posición de todos los objetivos y se la remite a los robots. También a nivel global se dispone del topic *Mensajes*, un mensaje bidireccional con un formato propio diseñado para almacenar la información relacionada con el símplex de acuerdo a lo tratado en el Capítulo 3. A nivel individual cada robot dispone de una serie de nodos propios de entre los que destacan:

- *Hokuyo*, nodo encargado del funcionamiento del LiDAR incorporado en el robot.
- *AMCL Pose*, con la información recibida de *Hokuyo* y conocido el mapa pre-establecido, será capaz de inferir la posición del robot.
- *Asignación*, nodo central del cálculo de la asignación. Recibirá la posición del robot (*AMCL Pose*) y la de los objetivos (*Decawave*), para que, junto con la

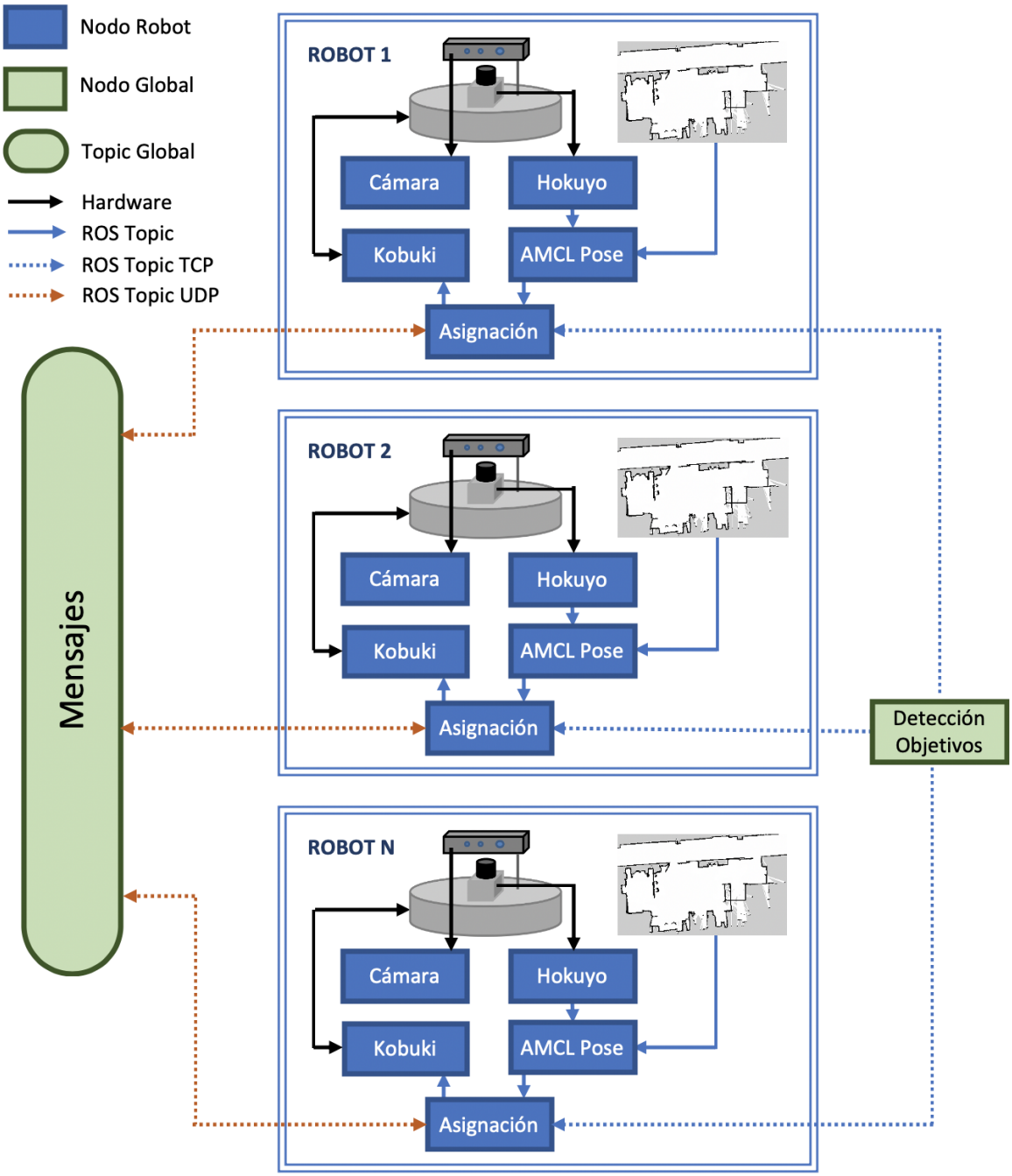

Figura 4.5: Arquitectura diseñada para el nodo de asignación.

información recibida del topic *Mensajes*, se calcule la asignación de dicho robot y se envíe la orientación necesaria a *Kobuki*. Adicionalmente se actualiza la información del topic *Mensajes* y se transmite al resto de robots.

– *Kobuki*, nodo encargado del movimiento del robot. Recibirá comandos de velocidad angular por parte del nodo *Asignación* para orientar el robot hacia el objetivo asignado.

– *Cámara*, nodo encargado de la visualización proveniente de la cámara incorporada

en el robot.

Dentro del esquema mostrado cabe también diferenciar los tipos de comunicaciones que existen. Entre los nodos propios de cada robot, dado que se encuentran dentro de la misma unidad, la información se transmite a través de topics locales. El resto de conexiones entre los nodos y topics globales con los robots se realizan de modo inalámbrico. Cabe destacar que estas conexiones inalámbricas de manera genérica se realizan a través de conexiones TCP en las que se requiere tanto de un emisor como de un receptor para entablar el canal de comunicación. Por el contrario, el topic *Mensajes* en el que leen y escriben los robots, se ha definido para que se transmita a través del protocolo UDP. Este tipo de protocolo permite el envío de información entre un emisor y un receptor, pero sin la necesidad de confirmación de llegada del mensaje. De esta forma el emisor manda la información y el receptor, si puede, la recibirá, pero siempre se mandará la información por parte del emisor aunque no se sepa si esta ha llegado a su destino o no. Se ha definido este protocolo ya que en un entorno real, conforme los robots se van moviendo puede que la comunicación entre un par de robots no se de, por lo que así no es necesario confirmar el tránsito de información en cada instante, sino que cada robot manda su información y el robot que pueda la leerá y el que no no.

Para el caso en el que se quiera simular un entorno de este tipo en un único ordenador, al no haber problemas en las comunicaciones internas, se daría una comunicación completa entre los robots. Para que la simulación se asemeje más a la realidad, se ha añadido una capa adicional de software que sirve para bloquear las comunicaciones entre robots que se encuentren a una distancia superior a un umbral fijo. De este modo se puede simular un entorno real modificando las comunicaciones a nuestro parecer.

Capítulo 5

Análisis de resultados

Planteadas las diferentes estrategias de comunicación este capítulo se centra, en primer lugar, en realizar un análisis empírico de las diferentes estrategias de comunicación en el entorno simulado de Matlab. A continuación, en el capítulo se describen los experimentos realizados con la implementación de ROS tanto a nivel virtual dentro de un mismo ordenador, como a nivel de laboratorio utilizando robots y objetivos reales.

5.1. Simulación - Matlab

En las simulaciones realizadas se ha partido del modelo de Matlab utilizado en el artículo original. Este modelo es capaz de posicionar aleatoriamente grupos de N robots y N objetivos, moverlos aleatoriamente a lo largo del recinto delimitado y realizar el cálculo del símplex y la transmisión de los mensajes de cada uno de los robots de manera distribuida como si de un entorno real se tratase. En la Figura 5.1 se muestra un ejemplo visual de una simulación llevada a cabo con este modelo, en ella se observa un grupo de 7 robots (puntos azules) y 7 objetivos (cruces verdes). Las líneas grises que unen algunos robots indican la existencia de comunicación entre esos pares de robots, y las líneas verdes y rojas indican las asignaciones entre robot y objetivo, siendo estas de color verde si coinciden con la asignación óptima centralizada y de color rojo si no.

Este modelo se ha modificado en todo lo relativo a la transmisión de mensajes, incorporando así las diferentes variantes de las estrategias propuestas en el Capítulo 3. Adicionalmente, también se han modificado las mediciones tomadas durante la simulación, para evaluar parámetros que nos sirvan a la hora de comparar las diferentes propuestas.

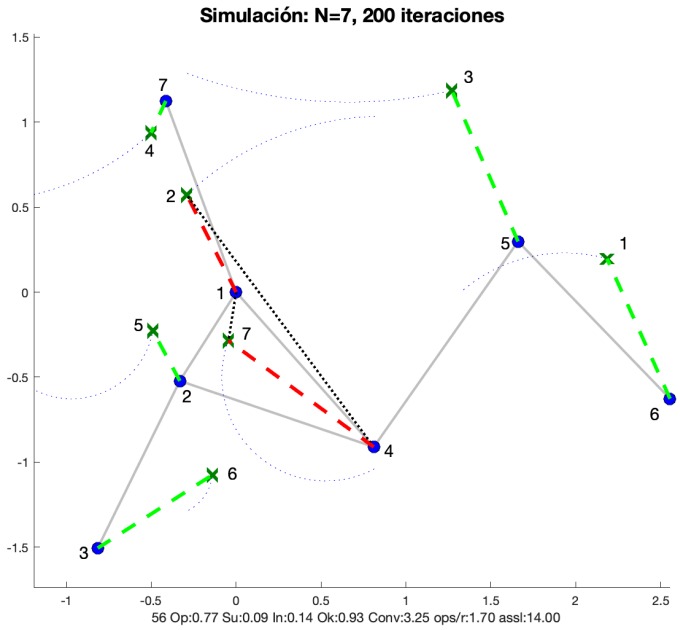

Figura 5.1: Ejemplo de simulación en Matlab para $N = 7$.

5.1.1. Parámetros de la simulación

Para evaluar las diferentes propuestas se han realizado una serie de simulaciones basadas en métodos de Monte Carlo, iguales para cada propuesta, con los siguientes parámetros. El número de robots y de objetivos ha sido variado desde $N = 3$ hasta $N = 20$ en incrementos de *1*, y desde $N = 20$ hasta $N = 40$ en incrementos de *5*. Cada simulación se ha realizado con 200 escenarios aleatorios distintos de posiciones de los robots y de los objetivos, manteniendo estos constantes entre las propuestas, a su vez en cada una de las simulaciones se ha ejecutado el algoritmo 200 veces para cada robot. Los robots han mantenido su posición inicial durante la simulación y los objetivos se han ido moviendo siguiendo una trayectoria circular aleatoria.

En la Tabla 5.1 se muestran a modo resumen las diferentes simulaciones realizadas. Cabe destacar que debido a los costes computacionales del modelo, los cuales se incrementan considerablemente conforme aumenta el número N, estas simulaciones se han realizado utilizando el Cluster Hermes del i3A. Las últimas simulaciones ($N = 35, 40$) se han llevado a cabo solo para 20 escenarios distintos en lugar de 200, debido al mismo problema de los tiempos de computo, el cual alcanzó un total de 32 días en el caso de $N = 30$.

N	ΔN	Propuestas	Escenarios	Iteraciones
$3 \rightarrow 20$	1	$0 \rightarrow 5$	200	200
$20 \rightarrow 30$	5	$0 \rightarrow 5$	200	200
$35 \rightarrow 40$	5	$0 \rightarrow 5$	20	200

Tabla 5.1: Resumen de las diferentes simulaciones llevadas a cabo en Matlab.

5.1.2. Métricas de evaluación

Para comparar las diferentes propuestas se han establecido una serie de parámetros a medir separables en dos grupos. En primer lugar se ha medido la distribución de las asignaciones a lo largo de las 200 iteraciones. Para ello se ha registrado el número de:

- Asignaciones óptimas, en las que la asignación de todos los robots coincide con la asignación óptima obtenida mediante el algoritmo húngaro centralizado.
- Asignaciones subóptimas, en las que algunas de las asignaciones difieren con las de la asignación óptima, pero se cumplen las restricciones de que cada robot solo ve a un objetivo y cada objetivo es visto por un solo robot.
- Asignaciones incompletas, en las que alguno de los objetivos no está siendo observado y por lo tanto otro está siendo visto por más de un robot.

A continuación se ha evaluado el sobrecoste medio de las asignaciones subóptimas referido al coste óptimo, es decir, el incremento de coste que se produce al tener la asignación subóptima en lugar de la óptima. Esta sobrecoste solo se ha medido para las asignaciones subóptimas ya que en el caso de las asignaciones incompletas este valor puede llegar a ser negativo al poder ser su coste menor que para una asignación óptima. Este hecho se puede dar cuando cada robot observa al objetivo que tiene más próximo sin importar que este objetivo ya esté siendo observado por otro robot y, por lo tanto, el coste total sería menor que el coste óptimo si se cumpliera con las restricciones del problema.

5.1.3. Resultados de la simulación

En la Figura 5.2 se observa la distribución de estas asignaciones para las diferentes propuestas a distintos valores de N. En una primera visión general fijándose principalmente en el número de robots, se observa que, conforme este número se incrementa, el número de asignaciones incompletas también aumenta. Es decir, cuanto mayor es el grupo de robots, peor son las asignaciones y hay muchos instantes en los que hay objetivos sin observar. Por otro lado, si nos fijamos dentro de cada valor de N en

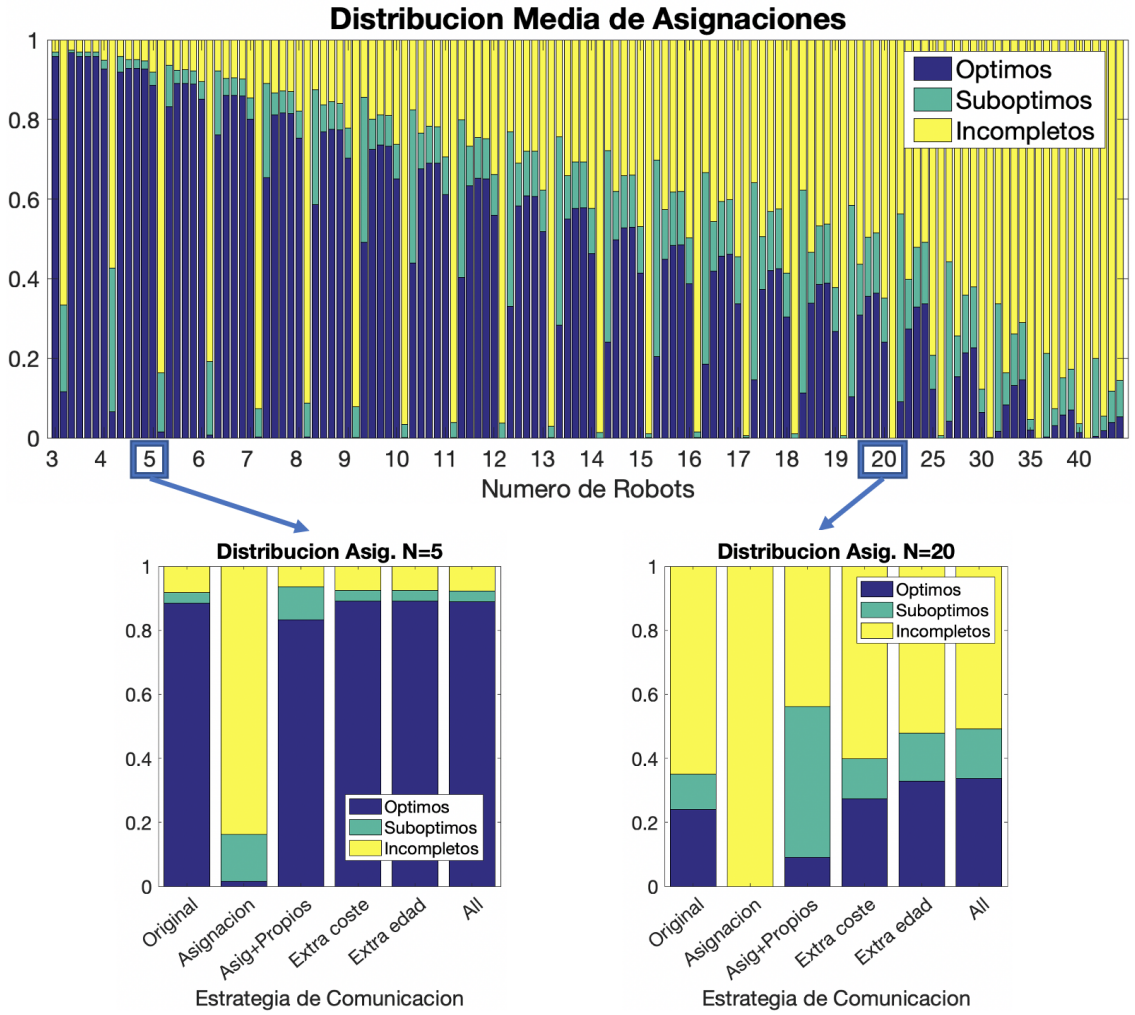

Figura 5.2: Gráficas de resultados de las simulaciones en Matlab correspondientes a la distribución de asignaciones para distintos valores de N.

la variación de las asignaciones en función de la propuesta elegida, se comprueba que el uso de la propuesta 1, es decir, mandar solo la asignación, obtiene una distribución de asignaciones muy alejada del resto, con valores de más del 80 % de incompletos incluso para $N = 5$. En cuanto al resto de propuestas, se percibe un moderado incremento en la optimalidad del modelo conforme se incrementa el tamaño del mensaje. Esto se da comparando la propuesta inicial con las propuestas 3, 4 y 5. Por último, en los resultados de la propuesta 2 (mandar la asignación más los valores propios), se consigue una disminución considerable de asignaciones incompletas para todos los valores de N, pero manteniendo el tamaño del mensaje en *2N-1*. Esta propuesta confirma la hipótesis que se tenía de que se podían conseguir mejores resultados que los originales modificando el criterio de selección de los datos de los mensajes; la única pega observada por el momento, es que al reducirse el número de incompletos, ha aumentado el número de subóptimos, por lo que para comprobar si este cambio perjudica las asignaciones

óptimas, se va a proceder a analizar los sobrecostes obtenidos por estas asignaciones subóptimas a lo largo de las 200 iteraciones.

Estos sobrecostes se evalúan en la Figura 5.3. Aquí se muestra el sobrecoste medio de las asignaciones subóptimas referido al coste óptimo. Para las propuestas 0, 3, 4 y 5, es decir, las propuestas que van incrementando el tamaño del mensaje, el sobrecoste es prácticamente el mismo ya que la distribución de asignaciones no varía en exceso. Para la propuesta 2, y para la cual se ha querido mostrar estos resultados, sí que se evidencia un incremento en el sobrecoste, pero este no llega a superar valores del $3,5\,\%$, por lo que la calidad de las observaciones de los objetivos, aunque no es óptima, está lo suficientemente cerca como para suponer que es buena.

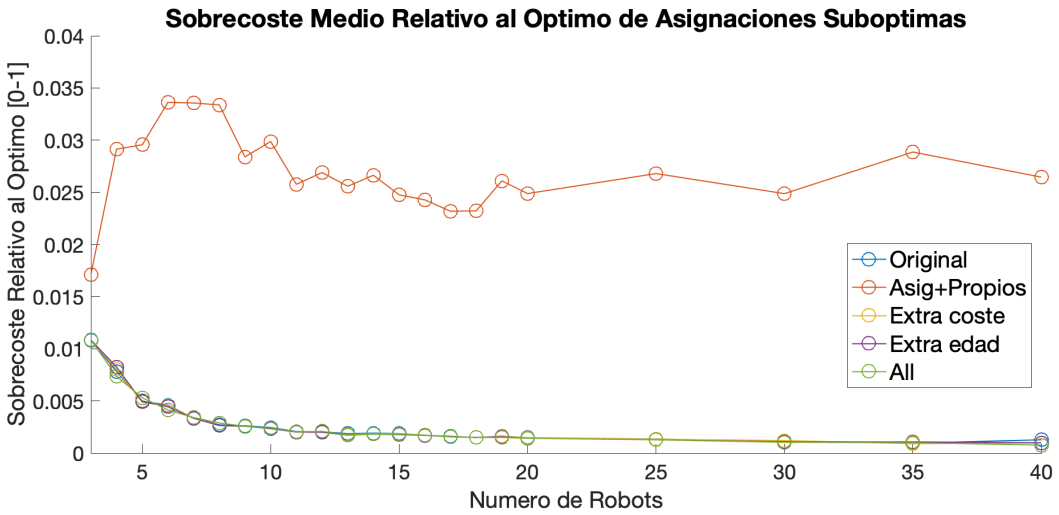

Figura 5.3: Sobrecoste medio relativo al óptimo de las asignaciones subóptimas para distintos valores de N.

5.2. Experimentación - ROS virtual

Para el primer experimento, ejecutado a nivel virtual, se han utilizado 4 robots y 4 objetivos posicionados de manera aleatoria a lo largo de un mapa del laboratorio, el cual coincide con el que se empleará con robots reales más adelante. En la Figura 5.4 se expone este mapa junto con la posición tomada por los robots, rodeados en azul, y la de los objetivos, rodeados en rojo.

Este experimento va a suponer una prueba para verificar el funcionamiento de la arquitectura propuesta. Para ello los objetivos van a recorrer trayectorias circulares a una velocidad constante, mientras en cada uno de los robots se ejecuta el algoritmo símplex. Para que el algoritmo se comporte de modo similar al real, se va a hacer uso de la capa adicional explicada al final del capítulo 4 para bloquear las comunicaciones entre pares de robots en función de la distancia entre estos. Durante este proceso se van a ir

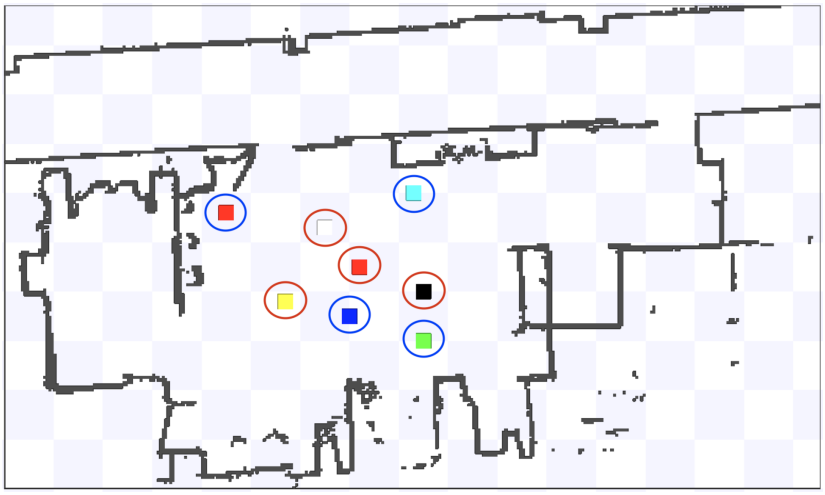

Figura 5.4: Captura de un experimento para $N = 4$ usando la herramienta Stage. Robots rodeados en azul y objetivos en rojo.

reuniendo los valores de las asignaciones tomadas por cada robot, así como la posición de cada robot y objetivo en cada iteración de cálculo, para, posteriormente, poder evaluar la asignación óptima y compararla con la obtenida durante el experimento. Para poder ir visualizando durante el experimento el funcionamiento se ha utilizado la herramienta RViz, visualizando en el mismo mapa no solo la posición de robots y objetivos, sino también la asignación tomada por estos y el rango de comunicación alcanzado por cada uno de los robots (ver Figura 5.5). Adicionalmente se ha medido la frecuencia con la que se realizan las iteraciones.

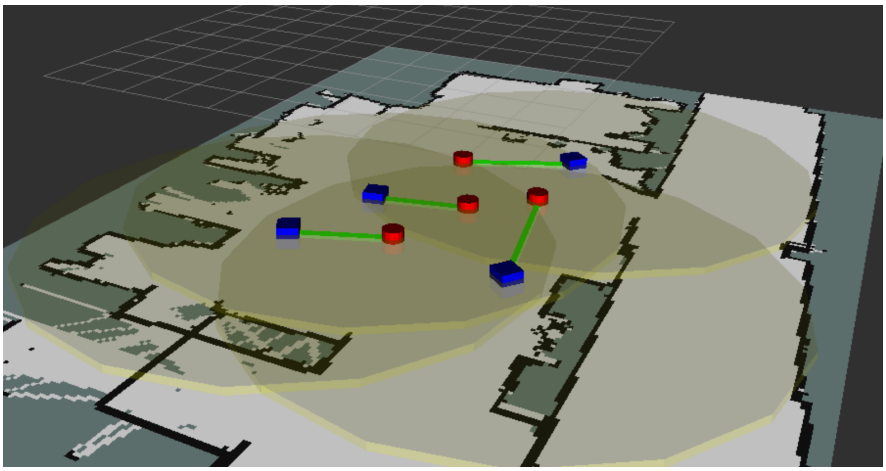

Figura 5.5: Captura de un experimento para $N = 4$ usando la herramienta RViz. Se muestran asignaciones y alcance de las comunicaciones además de robots y objetivos.

En la Tabla 5.2 se muestran los resultados obtenidos para grupos de 4, 6 y 8 robots. Todos los ensayos se han llevado a cabo durante un total de 60 segundos y se han tomado dos muestras para cada una de las combinaciones de número de robots y

propuesta utilizada. Al analizar la distribución de las asignaciones se puede advertir que la tendencia observada en las simulaciones en Matlab se sigue teniendo en estos ensayos con algunas pequeñas modificaciones, en este caso sustituir la propuesta original por la opción de mandar la asignación más los valores propios no siempre supone una reducción del número de asignaciones incompletas. Por el contrario, si se mandan todos los datos, si se consigue para todos los ensayos un aumento de la asignaciones óptimas y un descenso de las incompletas. Por último, destacar que para un tiempo de 60 segundos de ensayo, el número de iteraciones realizadas ha ido disminuyendo al aumentar el número de robots utilizados, dando a notar de este modo el incremento del coste computacional requerido.

N	Propuesta	Iteraciones	$f.(Hz)$	Óptimos	Subóptimos	Incompletos
4	Original	2000	40.82	89,21 %	1,90 %	8,89 %
4	Asig.+Propios	2000	40.78	89,55 %	1,06 %	9,39 %
4	All	2000	40.67	91,68 %	2,65 %	5,67 %
6	Original	1600	40.24	72,23 %	15,56 %	12,21 %
6	Asig.+Propios	1600	39.96	75,99 %	6,64 %	17,37 %
6	All	1600	39.98	88,87 %	3,96 %	7,17 %
8	Original	800	39.12	75,66 %	4,46 %	19,88 %
8	Asig.+Propios	800	39.11	71,87 %	4,41 %	23,72 %
8	All	800	38.97	81,21 %	3,98 %	14,81 %

Tabla 5.2: Distribución de asignaciones obtenidas tras experimentos en ROS a nivel virtual para $N = 4, 6, 8$.

5.3. Experimentación - ROS real laboratorio

En este último experimento, ya realizado en el laboratorio, se han empleado 3 robots TurtleBot equipados con un ordenador capaz de ejecutar ROS internamente para controlar cada uno de los nodos requeridos: Kobuki, Hokuyo, Cámara y Asignación. Para la preparación del ensayo ha sido necesaria la implementación del nodo de asignación en cada uno de los robots junto con una primera inicialización asociando a cada uno de los robots un identificador que será necesario para el tráfico de información y la asignación. A continuación, se ha establecido una red local, a la cual se han conectado todos los robots y que se empleará para el tráfico de datos entre los robots y el flujo de información proveniente del nodo Decawave, instalado en un ordenador que también servirá para inicializar los robots y recibir los resultados de los experimentos, pudiendo así representarlos mediante RViz y compararlos con las asignaciones óptimas como en el caso del ensayo virtual.

En las Figuras 5.6 y 5.7 se muestra una imagen inicial de como estaría montado el setup al iniciar el ensayo. Se puede identificar en la primera de ellas no solo los tres robots utilizados, sino también las tres personas que harán de objetivos y que se irán moviendo a lo largo de todo el laboratorio para ir obteniendo diferentes asignaciones. En la misma figura se puede apreciar una primera asignación en la que cada uno de los robots está orientado hacia la persona con el recuadro de su mismo color. En la segunda se examina la representación del setup a través de RViz en tiempo real, donde al igual que en los ensayos a nivel virtual se señalan los robots, los objetivos y las asignaciones, pero, en este caso, también se distinguen las imágenes producidas por las cámaras de los distintos robots en las que se aprecian los objetivos observados. Al ser un sistema formado por solo 3 robots y estar reunidos en un entorno pequeño, la comunicación se ha dado entre cada par de robots.

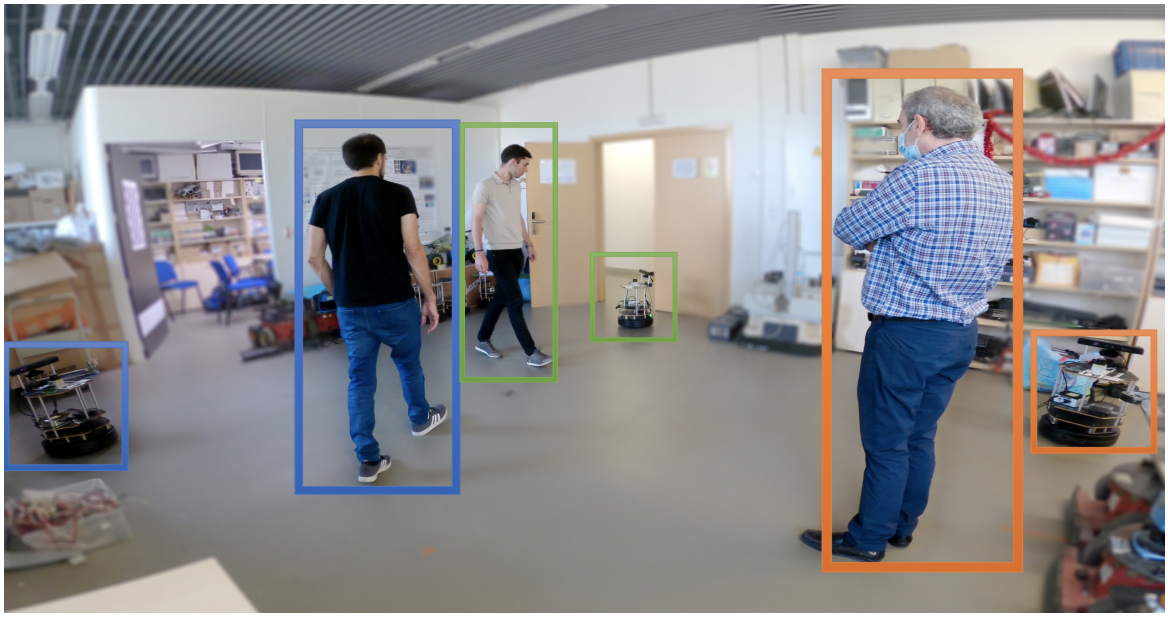

Figura 5.6: Visión del conjunto utilizado para los experimentos en el laboratorio. Se visualizan robots y objetivos utilizados con sus respectivas asignaciones identificada cada una con el mismo color que el robot que los observa.

En la Tabla 5.3 se muestran los resultados obtenidos para el experimento real llevado a cabo en el laboratorio. Para este ensayo solo se ha analizado una estrategia de comunicación, mandar la asignación y los valores propios de cada robot, la cual se había valorado positivamente tras las simulaciones en Matlab. Los resultados alcanzados, a pesar de ser algo inferiores a los obtenidos en la simulación para tres robots, siguen la misma distribución de asignaciones y apuntan al correcto funcionamiento de la implementación, pudiendo así asegurar que se ha logrado confeccionar un sistema distribuido totalmente funcional.

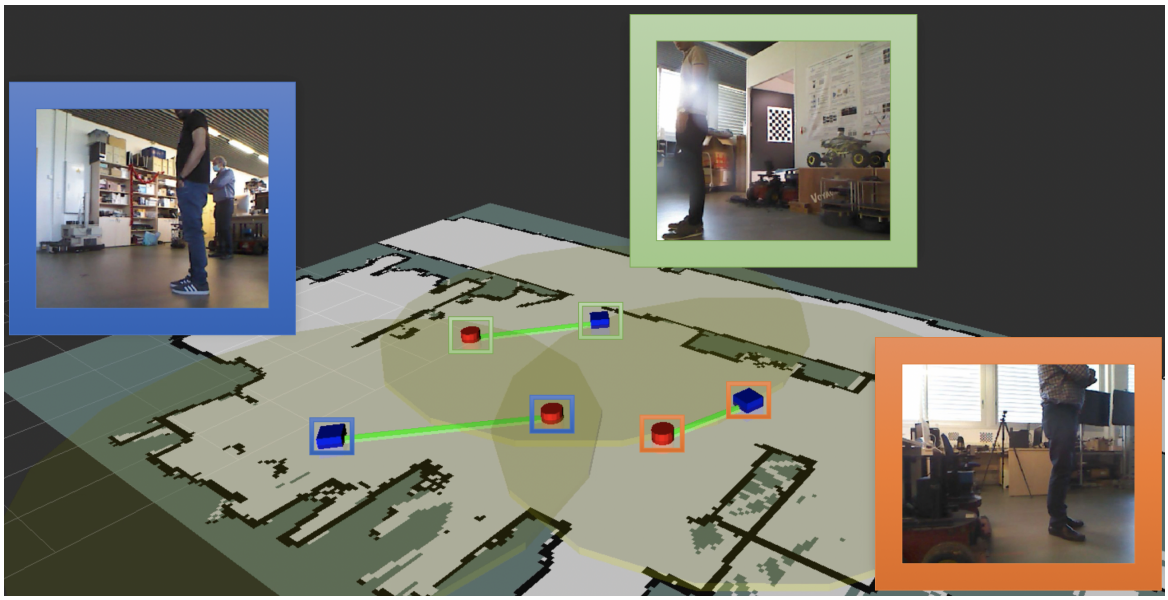

Figura 5.7: Representación en RViz en tiempo real del setup utilizado en el laboratorio. Muestra de los robots, los objetivos, las asignaciones y las imágenes de las cámaras incorporadas en los robots.

N	Propuesta	Iteraciones	$f.(Hz)$	Óptimos	Subóptimos	Incompletos
3	Asig.+Propios	800	39.12	75,77 %	3,76 %	20,47 %

Tabla 5.3: Distribución de asignaciones obtenidas tras experimentos en ROS a nivel real para $N = 3$.

Capítulo 6

Conclusiones y líneas futuras

En este capítulo se exponen las principales conclusiones obtenidas de este estudio y se plantean las líneas futuras de investigación como continuación a este Trabajo Fin de Máster.

6.1. Conclusiones

La utilización de sistemas multi-robot distribuidos ha ganado prominencia en los últimos años especialmente en campos como la exploración y la vigilancia, en los que su utilización para realizar tareas conjuntas permite un aumento de la seguridad y la eficiencia. Sin embargo, dada la gran complejidad de estos sistemas y los diferentes problemas que acarrean, como, por ejemplo, el control de los robots o su posicionamiento, en este proyecto se ha centrado la vista en el problema de asignación de las tareas a desarrollar por cada uno de los robots.

En este proyecto dicha cuestión se ha resuelto haciendo uso del algoritmo de optimización símplex del que se ha comprendido el funcionamiento así como su aplicación en sistemas distribuidos, donde se quiere evitar la utilización de una unidad central de gestión. Sucesivamente, se han estudiado diferentes estrategias de comunicación entre los robots y se ha validado experimentalmente que es posible una mejora en las asignaciones de las tareas, sin alterar el volumen de datos intercambiado, pero usando otros criterios para la selección de la información.

Además, se ha logrado implementar dentro de la plataforma ROS —una plataforma para desarrollo de software para el control de robots muy utilizada en ámbito académico e industrial— un algoritmo distribuido capaz de asignar tareas dinámicas en grupos multi-robot. Esta implementación se ha llevado a cabo cumpliendo con los todos los requisitos necesarios para que cada uno de los robots funcione de manera independiente, ejecutándose en cada uno de ellos el proceso encargado de realizar la asignación de

tareas y la comunicación entre los robots.

Como resultado final se ha obtenido un modelo completamente operativo dentro de la plataforma ROS. Este modelo es capaz de asignar un objetivo a cada robot, posicionar este último respecto a esta asignación y tramitar todo el tráfico de información entre los robots. El modelo, capaz de funcionar con diferentes estrategias de comunicación, ha demostrado ser operativo en entornos reales, habiéndose probado de manera virtual en grupos de hasta 8 robots, y de un entorno real con 3 robots. En ambas experimentaciones los resultados obtenidos han seguido la tendencia observada en las simulaciones, donde a mayor cantidad de robots peores son las asignaciones y a mayor cantidad de información intercambiada entre los robots, más se acerca el modelo a un sistema centralizado, es decir, mayor es la cantidad de asignaciones óptimas. Por todo esto, se han confirmado las observaciones iniciales y se ha demostrado su aplicabilidad en una implementación real.

6.2. Líneas futuras

Como posibles líneas de investigación que se han abierto tras la comprobación del funcionamiento de la arquitectura desarrollada y debido a los aspectos que se quedan fuera del alcance de este trabajo, se propone:

- Estudiar alternativas del algoritmo que permitieran la asignación de un número de tareas superior al número de robots, aspecto ya visto en otros artículos de investigación, pero sin el componente distribuido ni dinámico de las asignaciones.

- Sustituir las técnicas de posicionamiento de los objetivos por otras que no requieran el uso de sistemas de posicionamiento centralizado. Idea de esto sería, por ejemplo, el uso de técnicas de visión por computador.

Capítulo 7

Bibliografía

H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer Science, 2008.

Robert S. Garfinkel. An improved algorithm for the bottleneck assignment problem. *Operations research*, 19(7):1747–1751, 1971.

S. Chopra, G. Notarstefano, M. Rice, and M. Egerstedt. A distributed version of the hungarian method for multirobot assignment. *IEEE Transactions on Robotics*, 33 (4):932–947, Aug 2017.

M. Burger, G. Notarstefano, F. Allgower, and F. Bullo. A distributed simplex algorithm for degenerate linear programs and multi-agent assignments. *Automatica*, 48(9): 2298–2304, 2012.

E. Montijano, D. Tardioli, and A. R. Mosteo. Distributed dynamic sensor assignment of multiple mobile targets. 2019.

About ros. `https://www.ros.org/about-ros/`.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*, volume 1. MIT press Cambridge, 2000.

Mehran Mesbahi and Magnus Egerstedt. *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press, 2010.

Alvaro García. *ROS: Robot Operating System*. PhD thesis, Universidad Politécnica de Cartagena, 2013.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Fundamentos teóricos

En este anexo se van a describir fundamentos teóricos relacionados con la teoría de grafos y con el algoritmo símplex que serán de utilidad para poder describir la tipología del problema a abordar explicada en el Capítulo 2 y a la cual se hace referencia a lo largo de todo este trabajo.

A.1. Teoría de grafos

La teoría de grafos es una rama de las matemáticas y las ciencias de la computación que estudia las propiedades de los grafos (Mesbahi and Egerstedt, 2010). Se define un grafo finito, no dirigido y sencillo ($\mathcal{G} = (V, E)$) como un conjunto finito de elementos llamados vértices (V) que están conectados entre sí mediante un conjunto de aristas (E). El conjunto de vértices esta formado por n elementos definidos matemáticamente como

$$V = \{v_1, v_2, ..., v_n\}. \tag{A.1}$$

Estos, a su vez, definen el conjunto de aristas,

$$E = \{v_1v_2, v_2v_3, v_3v_4, v_3v_5, v_2v_5, v_4v_5\}, \tag{A.2}$$

definido como un subconjunto de dos elementos de V, denotado por $V \times V$, de la forma $\{v_iv_j\}$ donde $i, j = 1, 2, ..., n \quad \text{y} \quad i \neq j$ (ver ecuación A.2). Al tratarse de un grafo no dirigido, si existe una conexión entre el vértice v_i y el vértice v_j, también existirá entre el v_j y el v_i.

Los grafos pueden representarse tanto de manera matemática, como se ha visto hasta ahora con el par *(V, E)*, como también se pueden definir de manera gráfica, como se muestra en la Figura A.1, donde se representa el grafo correspondiente a los valores de la ecuación A.2. En la representación gráfica de un grafo, los puntos representan los vértices, y las líneas que unen los vértices representan las aristas.

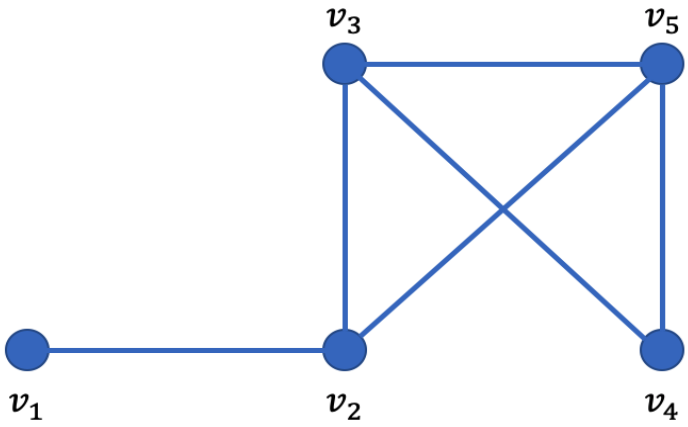

Figura A.1: Ejemplo gráfico de un grafo no dirigido y conectado.

Para la correcta interpretación del problema se van a explicar una serie de términos específicos de grafos que se usarán en el desarrollo del problema. Entre ellos hay que definir lo que es un grafo conectado, que se define como un grafo en el que para cada par de vértices en V, existe un camino a través de las aristas que conecta ese par de vértices. Si no existe tal camino, se dice que el grafo es inconexo. El ejemplo mostrado en la Figura A.1 describe un grafo conectado.

Debido a su capacidad no solo gráfica, sino también matemática, la teoría de grafos se ayuda de la representación matricial y es de gran uso en áreas como las ciencias de la computación y las telecomunicaciones. Entre estas matrices se encuentran:

– Matriz de grado ($\Delta(\mathcal{G})$): Matriz diagonal donde $[\Delta(\mathcal{G})]_{ii}$ muestra el grado de dicho vértice ($d(v_i)$), es decir, el número de vértices adyacentes a este.

$$\Delta(\mathcal{G}) = \begin{pmatrix} d(v_1) & 0 & \dots & 0 \\ 0 & d(v_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d(v_n) \end{pmatrix} \tag{A.3}$$

– Matriz de adyacencia ($A(\mathcal{G})$): Matriz simétrica $n \times n$ que muestra las relaciones de adyacencia del grafo según la forma:

$$[A(\mathcal{G})]_{ij} = \begin{cases} 1 & \text{si } v_i v_j \in E \\ 0 & \text{si no} \end{cases} \tag{A.4}$$

– Matriz laplaciana ($L(\mathcal{G})$): Representación completa del grafo en la que la suma de todas las filas debe de ser igual a cero. Para un grafo no dirigido se define:

$$L(\mathcal{G}) = \Delta(\mathcal{G}) - A(\mathcal{G}) \tag{A.5}$$

Para el ejemplo de la Figura A.1 tendríamos la matriz de grado y la matriz de adyacencia

$$\Delta(\mathcal{G}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix} \quad \text{y} \quad A(\mathcal{G}) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \tag{A.6}$$

y del mismo modo la matriz laplaciana como unión de las dos anteriores

$$L(\mathcal{G}) = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 \\ 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{pmatrix} \tag{A.7}$$

La matriz laplaciana se caracteriza por ser una matriz simétrica semi definida positiva, es decir, se pueden obtener sus valores propios y ordenar sus valores reales de la forma:

$$\lambda_1(\mathcal{G}) \leq \lambda_2(\mathcal{G}) \leq \cdots \leq \lambda_n(\mathcal{G}) \tag{A.8}$$

con $\lambda_1(\mathcal{G}) = 0$

Una vez obtenidos estos valores, si partimos de un grafo conectado, es decir, si $\lambda_2(\mathcal{G}) > 0$, se puede continuar analizando la conectividad del grafo, definiéndose esta con dos términos ($\kappa_0(\mathcal{G})$, $\kappa_1(\mathcal{G})$) y pudiendo deducir sus valores según la inecuación

$$\lambda_2(\mathcal{G}) \leq \kappa_0(\mathcal{G}) \leq \kappa_1(\mathcal{G}) \leq d_{min}(\mathcal{G}), \tag{A.9}$$

donde d_{min} es el mínimo grado de entre todos los vértices del grafo, es decir, el valor más pequeño de los términos de $\Delta(\mathcal{G})$.

- Conectividad del grafo $\kappa_0(\mathcal{G})$: Mínimo número de vértices que habría que eliminar de $\mathcal{G}$ para que el grafo fuera inconexo.

- Conectividad del grafo $\kappa_1(\mathcal{G})$: Mínimo número de aristas que con su eliminación incrementaría el número de componentes conectados de $\mathcal{G}$.

A.2. Algoritmo símplex

La programación lineal es el campo de la programación matemática dedicado a optimizar, ya sea maximizando o minimizando, una función objetivo de tipo lineal de varias variables reales mayores o iguales a cero. Esta función esta sujeta a restricciones

expresadas mediante una serie de ecuaciones e inecuaciones lineales (Luenberger and Ye, 2008).

Su representación matemática suele estar expresada según su forma canónica

$$\begin{array}{lll} \text{minimize} & c_1x_1 + c_2x_2 + \cdots + c_nx_n & \\ \text{subject to} & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n & = b_1 \\ & a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n & = b_2 \\ & \vdots & \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n & = b_m \\ \text{and} & x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0, & \end{array} , \tag{A.10}$$

o según su forma compactada vectorialmente

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c^T x}, \\ \text{subject to} & \boldsymbol{Ax = b},. \\ \text{and} & \boldsymbol{x \geq 0}, \end{array} \tag{A.11}$$

En esta última forma $\mathbf{x}$ es un vector columna de dimensión n, $\mathbf{c^T}$ es un vector fila también de dimensión n, $\mathbf{A}$ es una matriz $m \times n$ y $\mathbf{b}$ un vector columna de tamaño m.

Para la resolución de este tipo de problemas se ha usado tradicionalmente el método símplex. La idea de este método es pasar de una solución cualquiera del problema a otra, disminuyendo constantemente el valor objetivo hasta alcanzar la solución mínima. Para la realización de este procedimiento, el algoritmo se basa en un método de pivotaje que se explica paso a paso a continuación en su forma tabular, y al final del capítulo en su forma matricial. Esta última será la utilizada en el algoritmo debido a su simplicidad a la hora de implementar.

En primer lugar se ha de realizar la conversión de un problema específico a su forma estándar (eq. A.10) y así adaptarlo al método símplex. Para que el problema sea adecuado para su resolución debe cumplir las siguientes condiciones:

- El objetivo consistirá en minimizar el valor de la función objetivo.
- Los términos de $\mathbf{b}$ deben ser no negativos.
- Todas las restricciones deben ser ecuaciones de igualdad.
- Todas las variables (x_i) deben tener valor positivo o nulo.

Si el sistema a resolver en su forma inicial no cumple algunas de estas condiciones, será necesario realizar alguna transformación para que las cumpla. De este modo si el sistema inicial requiere una maximización, hay que transformarlo a un sistema de minimización multiplicando por -1 la función objetivo. Esta regla se aplicará del mismo modo si algún término de $\mathbf{b}$ es negativo, teniendo en cuenta que al multiplicar cada

expresión por *-1* la restricción de la inecuación variará. Por último hay que transformar las restricciones en forma de inecuación a su forma de igualdad. Para ello se va a considerar el caso sencillo en el que las inecuaciones del problema son del tipo $\leq$. En este caso basta con crear una serie de variables objetivos llamadas *slacks* no negativas ($x_s \geq 0$) que aparecen con coeficiente cero en la función objetivo y sumadas en la ecuación correspondiente convirtiendo la inecuación de tipo $\leq$ en una igualdad.

En el ejemplo

$$\begin{array}{lll} \text{maximize} & 3x_1 + x_2 + 2x_3 & \\ \text{subject to} & 2x_1 + x_2 + x_3 & \leq 2 \\ & x_1 + 2x_2 + 3x_3 & \leq 5 \\ & 2x_1 + 2x_2 + x_3 & \leq 6 \\ \text{and} & x_i \geq 0, i = 1, \ldots, 3, & \end{array}$$

$$\Downarrow \tag{A.12}$$

$$\begin{array}{lll} \text{minimize} & -3x_1 - x_2 - 2x_3 & \\ \text{subject to} & 2x_1 + x_2 + x_3 + x_4 & = 2 \\ & x_1 + 2x_2 + 3x_3 + x_5 & = 5 \\ & 2x_1 + 2x_2 + x_3 + x_6 & = 6 \\ \text{and} & x_i \geq 0, i = 1, \ldots, 6, & \end{array}$$

se observa la transformación mencionada pasando de un sistema de maximización a uno de minimización, junto con la creación de tres variables *slack* (x_4, x_5, x_6) dadas tres inecuaciones a cumplir.

Una vez estandarizado el problema se pasa a aplicar el método símplex y con ello la construcción de la primera tabla del modelo. En la Tabla A.1 se muestra como sería esta tabla de manera genérica y su construcción para el ejemplo presentado anteriormente. Se puede observar durante la construcción de la tabla que todos los términos son conocidos excepto los definidos en la última fila de la tabla donde se encuentra el término z_j definido de la forma $z_j = \sum_{i=1}^{m}(C_{bi}P_j)$. Los términos $z_j - c_j$ se pasan a denominar coste reducido.

Tras construir la tabla se comprueba si el método ha alcanzado ya la solución óptima verificando si se cumple la condición de parada. Esta condición, para el caso que tratamos en un problema de minimización, consiste en no encontrar ningún valor positivo en la última fila de la tabla ($z_j - c_j$). Si la condición de parada se cumple, en las primeras dos columnas de la tabla se encontraría el valor asociado a cada una de las variables distintas de cero, y el valor objetivo correspondería al último valor de esa misma columna (z_0). Si por el contrario la condición de parada no se cumple, es necesario realizar una nueva iteración del algoritmo, es decir, determinar cual va a ser el elemento pivote, actualizar la tabla tras el pivotaje y volver a comprobar la condición de parada, y así comprobar si se ha alcanzado el valor mínimo alcanzable

c		c_1	...	c_n	0	...	0
Variables	P_0	P_1	...	P_n	P_{n+1}	...	P_{n+m}
x_{n+1}	b_1	A_{11}	...	A_{1n}	1	...	0
$\vdots$		$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n_{n+m}	b_m	A_{m1}	...	A_{mn}	0	...	1
$z_j - c_j$	z_0	$z_1 - c_1$	...	$z_n - c_n$	0	...	0

$\Downarrow$

c		-3	-1	-2	0	0	0
Variables	P_0	$\mathbf{P_1}$	P_2	P_3	P_4	P_5	P_6
$\mathbf{x_4}$	**2**	**2**	1	1	1	0	0
x_5	5	1	2	3	0	1	0
x_6	6	2	2	1	0	0	1
$z_j - c_j$	0	**3**	1	2	0	0	0

Tabla A.1: Estructura genérica de la primera tabla del modelo y su evaluación para el ejemplo de la eq. A.12. Marcado en rojo el elemento pivote E_{IJ} para la primera iteración.

para la función objetivo.

Para escoger la variable pivote (x_J), se observa el coste reducido de las diferentes variables y se selecciona la variable con el mayor coste reducido entre los valores positivos. A continuación, se distingue la variable que va a salir en lugar de la variable de pivotaje, como aquella que se encuentra en la fila cuyo cociente P_0/P_j sea el menor de los estrictamente positivos. Para el caso propuesto la variable de entrada sería x_1, ya que su coste reducido tiene un valor de *3*, y la variable de salida sería x_4 ($2/2 < 6/2 < 5/1$). Definidas las variables de entrada y salida se marca el elemento pivote (E_{IJ}) donde se cortan la columna de la variable de entrada (E_{iJ}) y la fila de la variable de salida (E_{Ij}) y se procede a actualizar la tabla modificando los elementos del siguiente modo, de forma análoga al método de Gauss-Jordan, consiguiendo todos los valores nulos en la columna de pivotaje excepto el elemento pivote convertido a valor *1*.

– Elementos de la fila del elemento pivote:

$$E_{ij_{k+1}} = E_{ij_k}/E_{IJ} \tag{A.13}$$

– Elementos de las filas restantes:

$$E_{ij_{k+1}} = E_{ij_k} - (E_{iJ_k} \cdot E_{Ij_{k+1}}) \tag{A.14}$$

Tras actualizar la tabla solo quedaría volver a comprobar la condición de parada y continuar —o no— con otra iteración. En la Figura A.2 se muestra a modo de gráfico

c		-3	-1	-2	0	0	0
Variables	P_0	P_1	P_2	$\mathbf{P_3}$	P_4	P_5	P_6
x_1	1	1	1/2	1/2	1/2	0	0
$\mathbf{x_5}$	**4**	0	3/2	**5/2**	-1/2	1	0
x_6	4	0	1	0	-1	0	1
$z_j - c_j$	-3	0	-1/2	**1/2**	-3/2	0	0

Tabla A.2: Tabla actualizada del ejemplo de la eq. A.12 tras el primer pivotaje partiendo de la Tabla A.1. Marcado en rojo el elemento pivote E_{IJ} para la segunda iteración.

el proceso que se sigue para la aplicación del método símplex y en la Tabla A.2 como sería la tabla actualizada del ejemplo propuesto tras la primera iteración, se puede observar que se no se cumple la condición de parada y por tanto habría que iterar de nuevo con el elemento E_{23} como pivote.

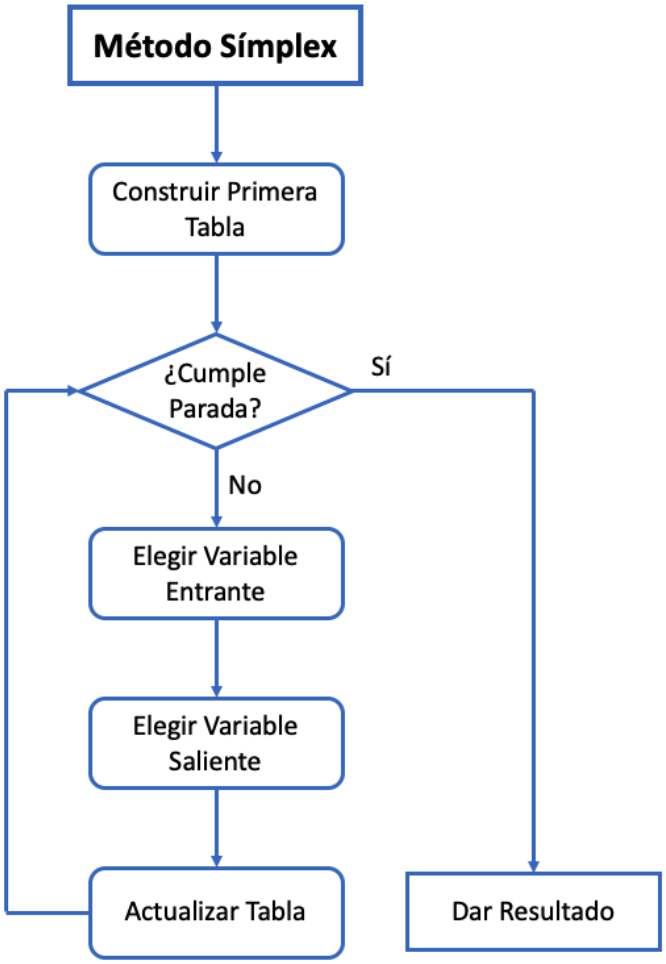

Figura A.2: Esquema gráfico de resolución del método símplex.

Partiendo de la resolución de manera tabular vista hasta ahora se podría realizar el cálculo matricial del algoritmo símplex. Para ello, tomando inicialmente la forma de la

ecuación A.11 y habiendo realizado las conversiones necesarias para su transformación a la forma estándar, se reescribe a la forma:

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c_B x_B + c_N x_N}, \\ \text{subject to} & \boldsymbol{B x_B + N x_N = b}, \\ \text{and} & \boldsymbol{x \geq 0}, \end{array} \tag{A.15}$$

Donde aparecen los términos B y N que hacen referencia a las variables básicas y a las no básicas, así llamadas respectivamente, siendo éstas primeras las variables asociadas a las variables *slacks* en la primera iteración. Para el ejemplo tratado anteriormente, las nuevas matrices serían:

$$\begin{array}{l} \mathbf{x_B} = \begin{pmatrix} x_4 \\ x_5 \\ x_6 \end{pmatrix}, \quad \mathbf{x_N} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad \mathbf{c_B} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{c_N} = \begin{pmatrix} -3 \\ -1 \\ 2 \end{pmatrix} \\ \mathbf{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{N} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 3 \\ 2 & 2 & 1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 2 \\ 5 \\ 6 \end{pmatrix} \end{array} \tag{A.16}$$

Una vez definidas las matrices del sistema, el proceso de resolución matricial del algoritmo sigue las mismas consideraciones que el método tabular y por lo tanto se podría aplicar el mismo esquema de resolución que se muestra en la Figura A.2 intercambiando "Tabla"por "Matrices", pero para una mejor interpretación del método y para mostrar las operaciones que habría que seguir, se presentan a continuación los pasos a seguir.

1. Determinar una solución inicial alcanzable ($\mathbf{\Omega}$) partiendo de $\mathbf{x_B}$, ($\mathbf{\Omega} = \mathbf{x_B}^T$).

2. Partiendo de la matriz cuadrada $\mathbf{B}$, evaluar su inversa ,$\mathbf{B^{-1}}$.

3. Calcular los costes reducidos ($z_j - c_j$) para todas las variables no básicas ($\mathbf{x_N}$), según:

$$z_j - c_j = \boldsymbol{c_B B^{-1} N_j} - c_j \tag{A.17}$$

   Si $z_j - c_j \leq 0$, se ha alcanzado la condición de parada y el valor objetivo es $z = \mathbf{c_B} \cdot \mathbf{x_B}$, para $\mathbf{x_B} = \mathbf{B^{-1}} * \mathbf{b}$. Si no, se determina la variable de entrada, x_k, según

$$z_k - c_k = \min\{z_j - c_j \mid j \text{ es no básica y } z_j - c_j > 0\} \tag{A.18}$$

   y se pasa al siguiente paso.

4. Calcular $\mathbf{B^{-1}N_k}$. Si todos sus elementos son ≥ 0, condición de parada, la solución es no acotada. Si no, calcular $\mathbf{B^{-1}b}$ y determinar la variable de salida, x_r, tal que:

$$\frac{(\mathbf{B^{-1}b})_\mathbf{r}}{(\mathbf{B^{-1}N_k})_\mathbf{r}} = \min\left\{ \frac{(\mathbf{B^{-1}b})_\mathbf{i}}{(\mathbf{B^{-1}N_k})_\mathbf{i}} \;\middle|\; i \text{ es básica y } (\mathbf{B^{-1}N_j})_i < 0 \right\} \tag{A.19}$$

5. Determinar la nueva base, $\mathbf{\Omega_{nueva}} = \mathbf{\Omega} \cup \{x_k\} - \{x_r\}$, definir $\mathbf{x_B} = \mathbf{\Omega}^T$ y volver al paso 2 con las matrices actualizadas según las nuevas variables básicas.

Anexos B

ROS

Sistema Operativo Robótico (siglas ROS en inglés) es un marco flexible para escritura de software de robots compuesto por una serie de herramientas, bibliotecas y convenciones que tienen como objetivo simplificar la tarea de crear una plataforma robótica compleja y robusta (ros.org). La principal ventaja de esta plataforma respecto a otras existentes en la actualidad es su facilidad de intercambio de software al ser una plataforma libre y esto le ha permitido la construcción de una gran comunidad de colaboradores en el mundo.

Esta plataforma posee gran versatilidad al poderse ejecutar sobre maquinas tipo Unix, principalmente Ubuntu y Mac OS X, siendo esta primera la utilizada en el desarrollo de este TFM. Adicionalmente a la hora de desarrollo de código permite el uso de distintos lenguajes como son Python y C++, siendo este último el que se utilizará a lo largo de todo esta explicación.

B.1. Estructura/Arquitectura interna

La estructura interna de ROS está organizada de una manera jerárquica en la que a la cabeza se encuentran los repositorios como ficheros compuestos por diferentes paquetes, y que permiten una sencilla descarga de entornos de desarrollo (García, 2013). Un escalón por debajo se encuentran los paquetes. Los paquetes son las unidades principales de organización dentro de ROS, en ellos se almacena todo lo necesario para que este sea funcional. Los paquetes a su vez están compuestos por nodos, los cuales son procesos que realizan cálculos dentro de ROS y gestionan la entrada y salida de información (ver Figura B.1). Esta información que entra y sale de los nodos se intercambia con otros nodos a través de canales de información llamados *topics* y formados por estructuras de datos definidas previamente conocidas como *mensajes*.

Como se ha mencionado, un nodo es un proceso en el que se realizan cálculos

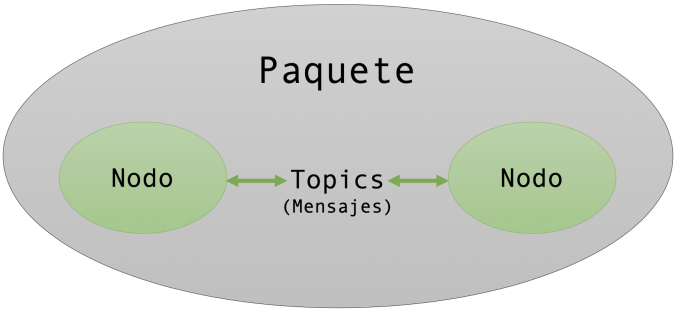

Figura B.1: Arquitectura del paquete.

asociados normalmente a una función, como podrían ser el nodo dedicado al giro de las ruedas de un robot o el encargado de su posicionamiento GPS. Estos nodos se combinan en un grafo y se comunican mediante *topics* formando entre todos ellos lo que se podría deducir como el sistema de control de un robot, el cual normalmente está formado por varios nodos. El envío de información entre los nodos de un robot o entre otros nodos de la red se realiza de un modo publisher-subscriber, es decir, para que exista un canal de comunicación debe de haber un nodo que emita un mensaje y otro nodo que lo reciba, si existen al menos un receptor y un publicador el canal se formará.

En la Figura B.2 se muestra un ejemplo formado por tres nodos marcados en verde: el nodo que controla el robot, /*Robot*_1, el nodo empleado para la visualización del robot en un mapa, /*robot_visual*, y el nodo /*teleop_robot* que permite mover el robot con un joystick. Entre estos nodos existen dos canales de comunicación establecidos, el primero de ellos conecta el robot con el visualizador a través del intercambio de información usando el *topic* /*Robot*_1/*pose* que consiste en un mensaje de tipo posición, que indica la coordenadas y orientación del robot en cada instante. Como la flecha indica, el visualizador es el que accede a esta información para poder posicionar el robot en el mapa. El segundo canal parte desde el nodo /*teleop_robot*, que publica información en el topic /*Robot*_1/*cmd_vel* de tipo comando de velocidad, que permite modificar la posición del robot mediante consignas de velocidad.

Como se advierte en este ejemplo, tanto cada uno de los nodos que forman el sistema, como los topics que se utilizan, deben tener un nombre distinto al resto para poder realizar los vínculos entre ellos sin ambigüedad. Si se diera el caso de tener un sistema con, por ejemplo, 3 robots, sería necesario que los nodos encargados de estos robots tuvieran nombres distintos (/*Robot*_1, /*Robot*_2, /*Robot*_3) al igual que los topics utilizados para definir su posición (/*Robot*_1/*pose*, /*Robot*_2/*pose*, ...).

Respecto a los topics usados en el ejemplo, se ha referido al topic /*Robot*_1/*pose* como a un mensaje de tipo posición, esto hace alusión a la estructura y tipo de datos

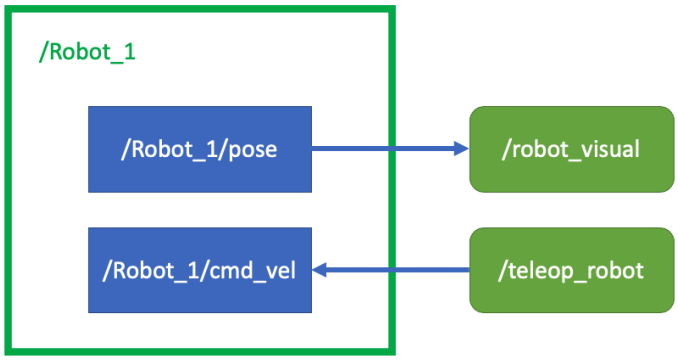

Figura B.2: Ejemplo nodo-topic.

que forman el mensaje. Dentro de ROS existen una serie de mensajes predefinidos, como pueden ser los mensajes tipo posición y orientación, comando de velocidad, secuencia de caracteres, etc. Por el contrario, si se requiere el envió de un mensaje característico diferente a los ya predefinidos, ROS permite la creación de otros tipos de mensajes mediante definición de unos simples archivos de texto.

B.2. Visualización y simulación

ROS está diseñado para el desarrollo de entornos de robots, por lo que entre las herramientas de las que se dispone existen una serie de visualizadores como es *RViz*, que permite la visualización de los robots en entornos dinámicos creados por el usuario (ver Figura B.3). Estos visualizadores no solo permiten posicionar el robot en un mapa, sino que también son capaces de mostrar información añadida como puede ser los datos recogidos por sensores de los robots o sobre el propio estado de los robots.

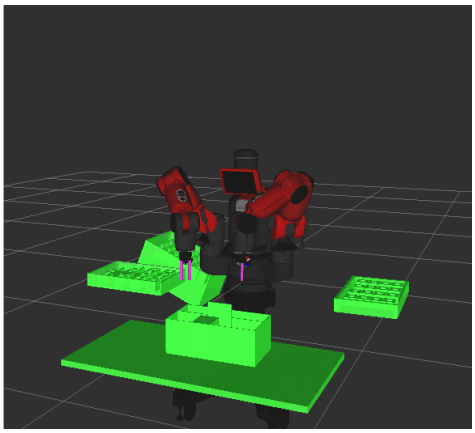

Figura B.3: Ejemplo de visualización en ROS utilizando la herramienta RViz.

Dentro de ROS también existen diferentes herramientas como *Stage* y *Gazebo* que permiten la creación de entornos virtuales y la inicialización de los diferentes robots

que en el se vayan a utilizar para poder crear simulaciones de entornos reales. En el caso de la herramienta *Stage*, es posible diseñar mapas, posicionar los robots e incluso establecer comportamientos y movimientos a los robots con el uso de nodos que se intercomunican con el simulador. Este intercambio de información con el simulador también se utiliza para comunicarse con los visualizadores ya mencionados, ya que la representación creada por *Stage* no es muy detallada para algunos casos (ver Figura B.4). De aquí que en la mayoría de casos se haga un uso simultáneo de *Stage* y, por ejemplo, *RViz* para crear la simulación de los robots y a continuación mostrarlos en el visualizador. El otro simulador mencionado, *Gazebo*, suele ser utilizado en entornos donde se requiera un nivel de detalle más alto, ya que permite la creación de entornos mucho más complejos, pero con una dificultad añadida en su desarrollo.

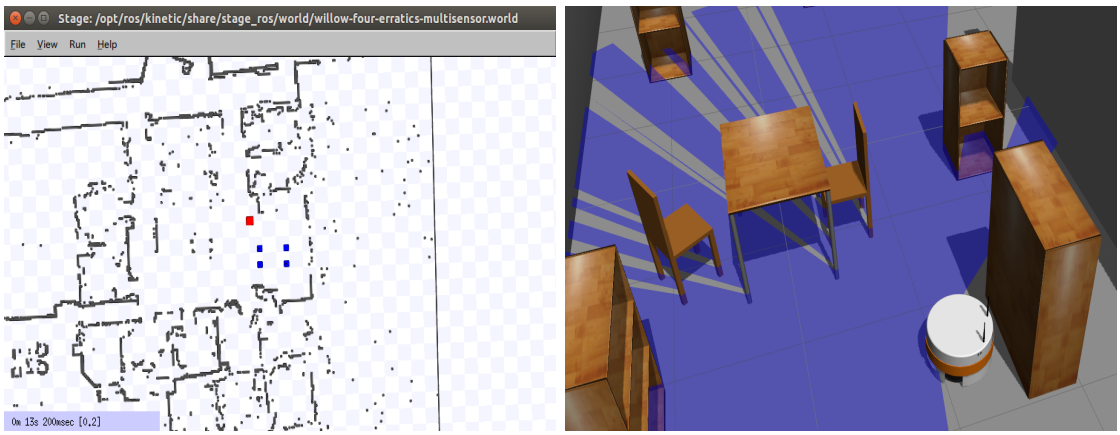

Figura B.4: Ejemplo de simulación en ROS utilizando Stage a la izquierda y Gazebo a la derecha.