Universidad Zaragoza
1542

Trabajo Fin de Grado

Navegación y Control de Sistema Multirobot

Navigation and Control of Multirobot Systems

Autor

Jorge Jarne Brun

Directores

Eduardo Montijano Muñoz

Carlos Sagüés Blázquiz

Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza

2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

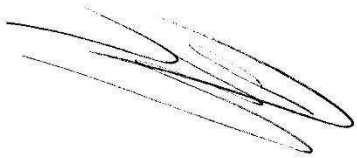
D./Dª. JORGE JARNE BRUN,

con nº de DNI 18174779H en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) GRADO, (Título del Trabajo) NAVEGACIÓN Y CONTROL DE SISTEMA MULTIROBOT,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 20 de junio de 2017



Fdo: JORGE JARNE BRUN

AGRADECIMIENTOS

Agradecer a Eduardo Montijano toda la atención y tiempo que me ha dedicado a lo largo de estos meses y que tanto me ha ayudado en la elaboración de este trabajo. También agradecer a Carlos Sagüés todo su trabajo y dedicación, como director de este trabajo y como profesor de la carrera.

RESUMEN

NAVEGACIÓN Y CONTROL DE SISTEMA MULTIROBOT

El uso de robots cada vez es más común en tareas de navegación y exploración, debido a su eficiencia y seguridad. La idea de usar un conjunto de robots en lugar de uno solo mejora notablemente estas tareas, debido a un gran ahorro de tiempo. Sin embargo, la utilización de múltiples robots complica considerablemente el problema de la navegación. Por un lado, la cantidad de variables a controlar aumenta linealmente con el número de robots. Por otro lado, es necesario calcular las trayectorias que deben seguir todas estas variables. Todo esto sin descuidar los problemas relacionados con la localización de cada robot dentro del entorno. El objetivo de este trabajo es proponer una solución conjunta a todos estos problemas, de tal manera que un equipo de robots pueda navegar de forma segura en entornos reales con obstáculos.

Para resolver el problema de planificación, en el trabajo se ha hecho uso de una abstracción que englobe al conjunto de robots. De esta manera se ha conseguido mantener el número de variables en la planificación constante e independiente del número de robots. En el TFG se ha evaluado esta técnica utilizando mapas sintéticos y mapas reales calculados utilizando un algoritmo de SLAM (Simultaneous Localization and Mapping).

Para el control individual de cada robot dentro de la abstracción se ha utilizado un algoritmo de cobertura basado en particiones de Voronoi. Este control requiere las ubicaciones de cada robot, por lo que se han analizado diferentes técnicas de localización, utilizando un método basado en filtros de partículas.

Todos estos métodos se han integrado en el software standard de robótica llamado ROS (Robot Operating System), que ha facilitado la comunicación con los robots y el control de los dispositivos a bajo nivel de éstos. La integración de todos los algoritmos dentro de este framework ha requerido el desarrollo de elementos de comunicación entre diferentes aplicaciones.

Por último, se han realizado experimentos donde se han probado diferentes trayectorias tanto en entornos simulados realistas como con robots reales en un entorno con obstáculos. En ambos casos se ha logrado que los robots sigan la trayectoria planificada con un movimiento uniforme en formación y evitando colisiones entre sí y con el entorno.

ÍNDICE

CAPITULO 1. INTRODUCCIÓN

1.1 Motivación

Actualmente, los robots son usados en gran cantidad de aplicaciones. Su uso va desde la exploración en zonas inaccesibles para el ser humano, como podría ser la superficie de otro planeta, hasta aplicaciones más cotidianas como tareas agrícolas, donde el uso de robots permite aumentar la productividad y abaratar los costes de producción. En estas aplicaciones, la utilización de un conjunto de robots coordinados, en lugar de uno solo, permite mejorar la eficiencia de la operación, sobre todo en los casos en los que se trata de entornos grandes y extensos.

Aunque el uso de un grupo de robots presenta grandes ventajas, el control del movimiento de éstos no es tarea fácil, y aún más si el grupo de robots es grande. Existen ya diferentes formas de realizar este control, pero muchas están sujetas a algunas limitaciones. Por ejemplo, un método muy común es el de mantener una formación geométrica entre los robots e ir cambiándola a lo largo del tiempo. Sin embargo, conforme se aumenta el número de robots a mover, más difíciles son las formaciones geométricas. Además, en tareas de navegación, la planificación de la trayectoria está totalmente limitada por el tipo de formación adoptada por los robots. Otro método muy usado es el que se conoce como “líder-seguidor”, en éste, un robot lidera el grupo y es el que en la navegación sigue la trayectoria calculada. El resto de robots, los seguidores, mantienen unas posiciones relativas al líder. Este método tiene un gran problema y es que, si el líder falla, los seguidores fallan también. Además, el líder tiene los movimientos y las velocidades limitados para que los seguidores puedan seguir su trayectoria.

Una solución interesante es el uso de una abstracción [6] para trabajar con el grupo de robots como si de un solo robot se tratase. Esto se debe a que simplifica el control multirobot a un número fijo de variables, independiente de la cantidad de robots. Además, se planifica la trayectoria de la abstracción [1], en lugar de calcular una trayectoria para cada robot. Gracias a un control a bajo nivel basado en las particiones de Voronoi [3], se puede hacer mover a cada robot dentro de la abstracción. El funcionamiento de las particiones se basa en crear una abstracción que rodee a los robots, por ejemplo, una circunferencia, y en ésta hacer tantas divisiones como numero de robots haya. Cada división tiene un punto llamado centroide, que es el punto óptimo de cada región, y el que asegura el recubrimiento óptimo de la circunferencia de la abstracción. El control de bajo nivel depende de este centroide y de la posición del robot. Por ello hay que estudiar técnicas de localización de los robots, como por ejemplo, el uso de una cámara. Sin embargo, este método tiene una gran limitación y es que la localización de los robots depende del campo de visión de la cámara.

La gran ventaja que presenta el sistema de navegación desarrollado en este trabajo respecto a otros, es que funciona tanto en interiores como en exteriores, gracias al uso de un algoritmo de localización basado en la información procedente de los sensores que tiene cada robot. La ausencia de sensores externos para la localización de cada robot es lo que hace a este sistema ser tan versátil.

Para poder realizar la localización del robot hay que obtener previamente un mapa del entorno. Sin embargo, obtener éste sin usar ningún sensor externo al robot es un

problema ya que para la construcción del mapa es necesario conocer la localización del robot. Esta necesidad de tener que conocer a la vez la posición y el mapa se ha solucionado en este trabajo mediante la aplicación de algoritmos de SLAM.

1.2 Objetivos

La meta principal de este proyecto es realizar, en un entorno real, el movimiento en formación de un conjunto de robots que siguen una trayectoria planificada en un mapa, teniendo en cuenta ciertas restricciones como la existencia de obstáculos. Para ello se proponen los siguientes objetivos:

- Búsqueda y uso de una plataforma que permita tratar a los robots a alto nivel y conseguir así una abstracción del hardware y de los controladores de dispositivos, además de simplificar la comunicación entre los robots.

- Creación de un mapa 2D mediante la navegación teleoperada de un único robot en un entorno desconocido.

- Realización de la planificación de trayectorias para ir de un punto a otro del mapa evitando obstáculos y usando la mejor ruta posible.

- Control de los robots mediante el uso de una abstracción que mueve de forma ordenada a los robots.

1.3 Alcance

Este trabajo parte de un proyecto fin de carrera [1] y de un trabajo fin de grado [2]. Del primero [1] se ha usado el planificador de trayectorias y del segundo [2] se ha utilizado el control de bajo nivel basado en particiones de Voronoi, ambos realizados en Matlab. En este proyecto se realiza la integración de estas implementaciones y se adapta para hacerlo funcionar con robots reales y únicamente con sensores integrados en los mismos.

Para ello, se ha realizado la puesta en marcha de la plataforma ROS en un ordenador y se ha estudiado su estructura interna para entender bien su funcionamiento. Además, para hacer la comunicación entre este software y las implementaciones anteriores desarrolladas en Matlab, se han tenido que hacer modificaciones en los archivos que se ejecutan de ambos programas.

Por último, se han realizado experimentos tanto en entornos simulados como con robots reales. En éstos se han probado diferentes trayectorias planificadas para comprobar cómo se comporta el sistema de navegación desarrollado y cómo influye el método de localización usado.

Las pruebas simuladas requieren además el uso de otro programa, Gazebo. La creación de escenarios en él y la simulación de los robots supone de nuevo un estudio de su funcionamiento y de la comunicación con el resto de programas.

1.4. Contenido de la memoria

El resto de la memoria se estructura de la siguiente manera:

- Capítulo 2: Este capítulo explica como mediante un planificador de trayectorias, se calcula un recorrido sin colisiones en un entorno con obstáculos, que junto con la aplicación de un control a bajo nivel, permite calcular las acciones que se deben aplicar a los robots para que sigan la trayectoria planificada.

- Capítulo 3: Aquí se estudian diferentes técnicas para la obtención de un mapa, debido a que el planificador anterior lo necesita para calcular la trayectoria. Por otro lado, se analizan algunos algoritmos de localización ya que el control de bajo nivel de cada robot depende de su posición en el mapa.

- Capítulo 4: Las técnicas anteriores se integran en una plataforma de robótica, lo que permite que se puedan aplicar en entornos reales. Por ello, en este capítulo se hace un estudio de la arquitectura del software que hace esta implementación posible, en concreto, el framework ROS, el simulador Gazebo y el entorno gráfico RVIZ.

- Capítulo 5: Se analizan los experimentos que se han llevado a cabo, tanto en entornos de simulación como con robots reales.

- Capítulo 6: Se muestran las conclusiones del proyecto junto a las dificultades que se han encontrado durante su desarrollo. Por último, se proponen las líneas de trabajo futuras que podrían mejorar los resultados obtenidos en este trabajo.

CAPÍTULO 2. ALGORITMO DE NAVEGACIÓN

El objetivo de este capítulo es describir el funcionamiento del algoritmo de navegación multirobot, que permite mover los robots en un entorno con obstáculos, siguiendo un recorrido previamente calculado por un planificador de trayectorias. El cual, se explica también en el capítulo y ha sido adaptado para trabajar con sistemas multirobot.

2.1 Descripción del robot

Se trata de un robot TurtleBot modelo 2 (2012) como el de la Figura 1, que está formado por una base móvil con dos ruedas y una estructura superior donde se encuentra el ordenador y los sensores. La base móvil tiene un movimiento no holónomo, esto significa que no puede desplazarse hacia los lados por el deslizamiento de las ruedas.

Figura 1. Robot TurtleBot modelo 2

Su estado se puede representar con tres variables, la posición (x_i e y_i) y la orientación (θ_i). Por tanto, el movimiento de robot se explica con las siguientes ecuaciones:

$$x_i\,(t+1) = x_i\,(t) - v_i\,(t) sen(\theta_i(t))\Delta t, \tag{2.1}$$

$$y_i(t+1) = y_i(t) + v_i(t) cos\big(\theta_i(t)\big)\Delta t, \tag{2.2}$$

$$\theta_i(t+1) = \theta_i(t) + \omega_i(\mathrm{t})\Delta t, \tag{2.3}$$

Donde el índice i hace referencia al número de robot, Δt al tiempo de muestreo, v_i representa la velocidad lineal del robot y ω_i su velocidad angular.

Respecto a su equipamiento, cuenta con un ordenador de la marca Intel llamado NUC, en el que se encuentra el sistema operativo Ubuntu (Linux) con el software ROS instalado. El robot está equipado con un láser Hokuyo como el de la Figura 2. Éste pesa cerca de 160 gramos y tiene un alcance de aproximadamente 5 metros. Cuenta con un ángulo de visión de 240º y con una resolución de 0, 36º. Su funcionamiento requiere una

alimentación de 5V de tensión y 2.5 W de potencia.

Figura 2. Laser Hokuyo

El robot también cuenta con una cámara RGB llamada Kinect que tiene sensor de profundidad y que en este proyecto se usa para grabar lo que ven los robots en primera persona durante los experimentos.

Para poder alimentar el ordenador que está en la estructura del robot, se usa una batería portátil. Este ordenador intercambia información con el láser y con el robot mediante el uso de dos puertos USB. El propio robot cuenta con una batería interna con la que se alimenta los motores que hacen mover las ruedas y el láser Hokuyo.

2.2 Planificación de trayectorias

Como cada robot tiene 3 variables que lo representan $\theta, x\ e\ y$, con el aumento del número de robots, las variables a controlar aumentan linealmente. De este problema nace la necesidad de usar una abstracción cuyo número de variables sea independiente de la cantidad de robots que se usen. En este caso, se ha usado un círculo como área que cubre a todos los robots, entonces esta abstracción se puede representar con solo 3 variables, centro y radio del círculo.

Por otro lado, en la tarea de navegación, un planificador de trayectorias es el que calcula el camino que debe seguir cada robot para moverse entre dos puntos del mapa evitando choques con obstáculos. Para adaptar este planificador al uso de la abstracción se ha usado la variación hecha en [1], la cual, se basa en el algoritmo de búsqueda A* (Anexo B) y permite obtener, la trayectoria que debe seguir la abstracción y también el tamaño de su radio. Por ejemplo, para pasar por sitios estrechos el radio de la circunferencia disminuye para que los robots puedan pasar.

El algoritmo original A* considera como casillas vecinas aquellas que están alrededor de ésta. La variación del código considera, además, como casillas vecinas, aquellas con radios directamente inferior y superior. De esta manera las trayectorias planificadas no tienen grandes cambios en sus radios.

Otra cosa a tener en cuenta es que si cambia el radio de la abstracción hay que cambiar también el tamaño de los obstáculos y así evitar colisiones. Un ejemplo de esto se puede ver en la Figura 3, para los tamaños 0.65, 0.85, 1.05 y 1.35 m.

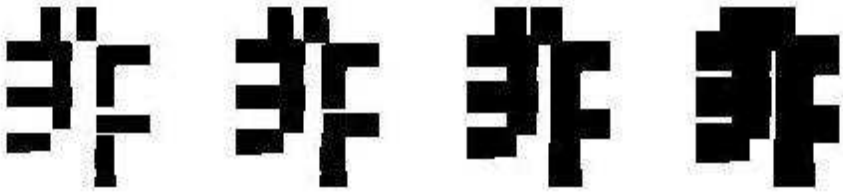


Figura 3. Mapa con variación del tamaño de sus obstáculos

El algoritmo original solo tiene en cuenta el coste de posición, el cual depende únicamente de la distancia euclídea entre una casilla y otra, sin embargo, en el algoritmo que se ha usado se añade un nuevo coste de modificar el tamaño de la abstracción, es decir, el radio de la circunferencia.

Este nuevo coste asociado al tamaño de la abstracción se centra en dar prioridad al uso de radios de mayor tamaño con la formula siguiente:

$$\mathrm{coste}_{cambioRadio} = \mathrm{k}(\mathrm{n}_R - \mathrm{R}), \tag{2.4}$$

El número de radios usados para el cálculo de la trayectoria está discretizado, por lo que la variable n_R es el número de radios posibles y R es el índice del radio que se está probando, los radios están ordenados de menor a mayor tamaño.

Su aplicación se puede ver en la Figura 4 donde el radio de la abstracción va modificándose a lo largo del recorrido adoptando en todo momento el radio máximo cuando los obstáculos se lo permiten

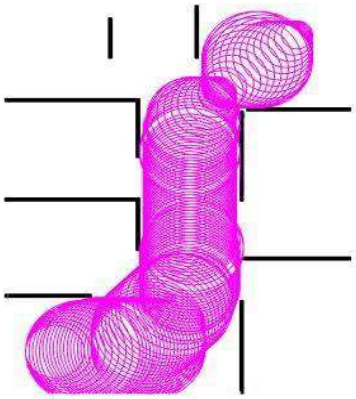


Figura 4. Planificación de trayectoria con variación del radio

2.3 Partición de Voronoi

Los robots tienen que ser capaces de moverse ordenadamente dentro de las abstracciones para así poder seguir la trayectoria calculada anteriormente. Para ello, se usa el control de bajo nivel realizado en el proyecto [2], que se basa en las particiones de Voronoi para calcular las acciones a aplicar a los robots. La partición de Voronoi divide el círculo en diferentes regiones por las que se mueve cada robot. La teoría que hay detrás del cálculo de estas particiones se encuentra en el Anexo A.

La acción a aplicar en cada robot depende de la distancia de la localización actual del robot p_i al centroide de su región de Voronoi c_{vi}.

$$\mathrm{u}_i = \mathrm{k}_r(\mathrm{c}_{vi} - p_i), \tag{2.5}$$

Con esta acción se puede calcular la velocidad lineal y angular necesaria en cada robot.

$$\begin{pmatrix} \mathrm{v}_i \\ \mathrm{w}_i \end{pmatrix} = \begin{pmatrix} cos(\theta_i) & sin(\theta_i) \\ \frac{-sin(\theta_i)}{l} & \frac{cos(\theta_i)}{l} \end{pmatrix} \begin{pmatrix} \mathrm{u}_{xi} \\ \mathrm{u}_{yi} \end{pmatrix}, \tag{2.6}$$

La aplicación de esta ley de control a una abstracción estática hace que cada robot llegue al centroide de su región. Sin embargo, con el uso del planificador de trayectorias del apartado anterior, la abstracción se va desplazando, lo que genera también el movimiento de los centroides.

El movimiento de la abstracción (radio y centro) a lo largo de la trayectoria se calcula así:

$$\mathrm{centro}(t+1) = \mathrm{centro}(t) + \mathrm{u}_{centro}\Delta t, \tag{2.7}$$

$$\mathrm{radio}(t+1) = \mathrm{radio}(t) + \mathrm{u}_{radio}\Delta t, \tag{2.8}$$

Este cálculo depende las variables u_{centro} y u_{radio}. Éstas representan la acción y se calculan con un regulador proporcional donde el error es la diferencia entre el centro o radio actual y el que se quiere alcanzar.

$$\mathrm{u}_{centro} = \mathrm{k}_c(centro' - centro), \tag{2.9}$$

$$\mathrm{u}_{radio} = \mathrm{k}_r(\mathrm{radio}' - \mathrm{radio}), \tag{2.10}$$

CAPÍTULO 3. NAVEGACIÓN EN ENTORNOS REALES

El algoritmo de planificación anterior necesita un mapa para poder calcular la trayectoria de la abstracción, por lo que en este capítulo se estudian las técnicas de construcción del mapa basadas en el uso de sensores del robot. También se analizan los diferentes algoritmos de localización existentes ya que, las Ecuaciones 2.2 y 2.3 del control de los robots, dependen de la posición del robot en el mapa para calcular sus velocidades.

3.1 Creación del mapa

A continuación, se explica el proceso de creación de un mapa 2D mediante el uso de un robot y la información que aportan sus sensores.

El proceso empieza con la navegación teleoperada de uno de los robots. Durante este proceso es necesario guardar los registros del láser y de la odometría. Es importante que a lo largo de la navegación se cumplan ciertas reglas para que el mapa obtenido sea lo más preciso posible:

- Evitar rotaciones rápidas.

- Intentar que durante la navegación no haya objetos dinámicos cerca del láser ya que podrían aparecer como obstáculos en el mapa 2D.

- Dar varias vueltas sobre el entorno para que detecte cuando el robot ha vuelto a pasar por una localización que ya había estado y así mejorar la estimación de la posición del robot. Este procedimiento es conocido como la detección del cerrado del bucle.

Los registros guardados junto a la aplicación de un algoritmo de SLAM es lo que permite la obtención del mapa. Con esto se soluciona la necesidad de obtener el mapa del entorno y a la vez localizarse en él.

Una de las posibles soluciones al SLAM es el uso de algoritmos basados en filtros de partículas, éstos son filtros no paramétricos y usan un numero finito de partículas para representar la solución, si el número de partículas fuera infinito, la solución sería la óptima. El filtro de partículas tiene en general un alto tiempo de cálculo, pero éste depende del número de partículas que se usen para el cálculo de la solución, al usar muchas partículas se está más cerca de la solución óptima. La ventaja de este algoritmo está en que se puede buscar el equilibro entre coste computacional y exactitud de la solución. En concreto, el filtro usado es el RaoBlackwellized, éste está explicado en [4] y consiste en la factorización del problema en dos más simples:

$$p(\mathrm{x}_{1:t}, \mathrm{m}|\mathrm{z}_{1:t}, \mathrm{u}_{1:t-1}) = p(\mathrm{m}|\mathrm{x}_{1:t}, \mathrm{z}_{1:t}) \cdot p(\mathrm{x}_{1:t}|\mathrm{z}_{1:t}, \mathrm{u}_{1:t-1}), \tag{3.1}$$

Siendo m el mapa, x la trayectoria del robot a lo largo del tiempo, z las medidas del sensor láser y u las medidas de odometría. El problema se divide en el cálculo del mapa conociendo la posición y, por otro lado, la estimación de la posición con los datos del láser

y de la odometría.

El proceso comienza con el reparto aleatorio de muchas partículas, cada una de éstas representa una trayectoria del robot y cada una tiene un peso que indica como de parecida es la partícula a la realidad, es decir, la probabilidad de que la partícula esté cerca de la solución. El filtro de partículas realiza 4 pasos:

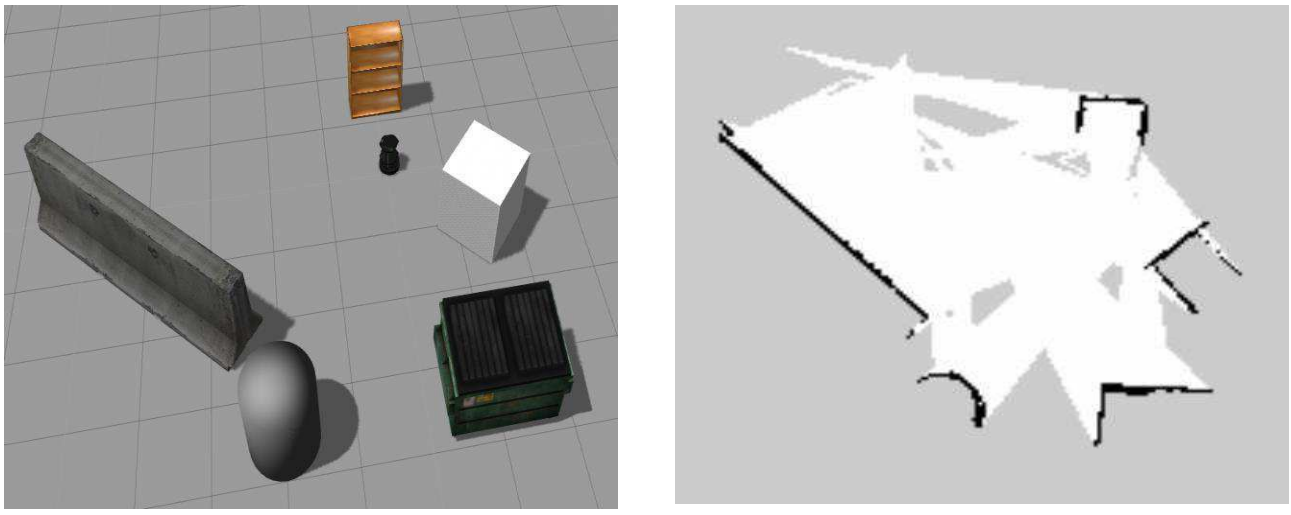
- Se obtiene una nueva generación de partículas proveniente de la aplicación de las ecuaciones de movimiento del robot a las partículas de la generación anterior. Esta fase se conoce también como la de predicción.

- Calculo del peso que tiene cada partícula, este cálculo depende del error entre la medida predicha y la lectura de los sensores, en este caso del láser y de la odometría.

- Resampleo en las zonas donde las partículas tengan un mayor peso, pero esta vez se les asocia a todas las partículas el mismo peso. Sin embargo, al hacer este paso es posible que se elimine la partícula correcta y que, además, esto lleve a la perdida de diversidad de partículas. La solución que se propone en [4] es la de realizar este resampleo solo cuando se cumpla que la variable llamada “numero efectivo de partículas” sea menor que $N/2$,siendo N el número de partículas. Esta variable mide como de bien se aproximan las partículas al objetivo siguiente.

- Estimación del mapa para cada partícula basándose en la trayectoria y en las observaciones.

El resultado de la aplicación de este algoritmo es el mapa 2D de la Figura 5 (b), obtenido en el escenario de la Figura 5 (a). En el mapa se ven las imperfecciones debidas al alcance del láser, lo que produce zonas sin determinar y contornos de los objetos incompletos.



(a) *(b)*

Figura 5. (a) Escenario simulado en Gazebo. (b) Mapa 2D obtenido con el laser

3.2 Localización de los robots en el mapa

Existen diversidad de métodos para localizar un robot, el más conocido probablemente sea el GPS, que permiten conocer, con gran precisión y mediante triangulación, la localización de un objeto. Tiene la gran limitación que solo funciona en sitios a cielo abierto.

Otra manera de estimar la posición es mediante la odometría, que permite conocer cómo se va desplazando un robot respecto a su posición inicial. Su funcionamiento se basa en el uso de unos encorders que suelen estar situados en las ruedas. Estos encoders permiten conocer la posición de la rueda mediante la emisión de pulsos digitales. Sin embargo, la odometría tiene dos tipos de errores típicos, sistemáticos, por ejemplo, la diferencia entre los diámetros de las dos ruedas del robot, y no sistemáticos, cuando la rueda patina sobre el suelo. La odometría solo es útil para pequeños recorridos, puesto que los errores sistemáticos que se producen se van acumulando. Los errores no sistemáticos aparecen eventualmente, pero crean errores muy grandes.

También es común, localizar los robots usando una cámara fija, pero estas técnicas suelen tener un tiempo de cómputo alto, debido a que usan algoritmos de visión por computador, y además el campo de localización de los robots está limitado al rango de visión de la cámara.

Resulta interesante entonces, lograr una localización que no haga uso de ningún sensor externo al robot y que sea lo más precisa posible, tanto en sitios cerrados como abiertos. Por eso, el sistema de localización que se ha implantado es capaz de obtener la posición de cada robot respecto a una referencia, usando solo sus propios sensores. Este sistema usa un algoritmo basado en técnicas de Montecarlo y al igual que la solución al SLAM, usa filtros de partículas.

La explicación de su funcionamiento se ha extraído de [5] y es la siguiente, se parte de un numero N de partículas repartidas por todo el mapa, se aplica a cada partícula, la ecuación del movimiento del robot con la información de los encoders. Se llega así a una nueva hipotética posición en el mapa para cada partícula. Luego se calcula la probabilidad de que cada partícula corresponda a la posición real del robot, comparando las medidas del láser con las que vería en las hipotéticas posiciones de las partículas. Con todas las probabilidades calculadas, se define un límite, y todas aquellas partículas que están por debajo de éste se suprimen. De nuevo se reparte un numero N de partículas en el mapa, pero esta vez en las posiciones de las partículas que están por encima del límite definido. La posición que devuelve el algoritmo es la media ponderada de todas estas partículas.

El resultado del proceso de localización se puede observar bien en la Figura 6, las flechas verdes representan las partículas, cada una con distinta posición y orientación. La línea roja es la lectura del láser del robot, la cual prácticamente se superpone con los obstáculos del mapa, esto significa que el robot se ha localizado correctamente.

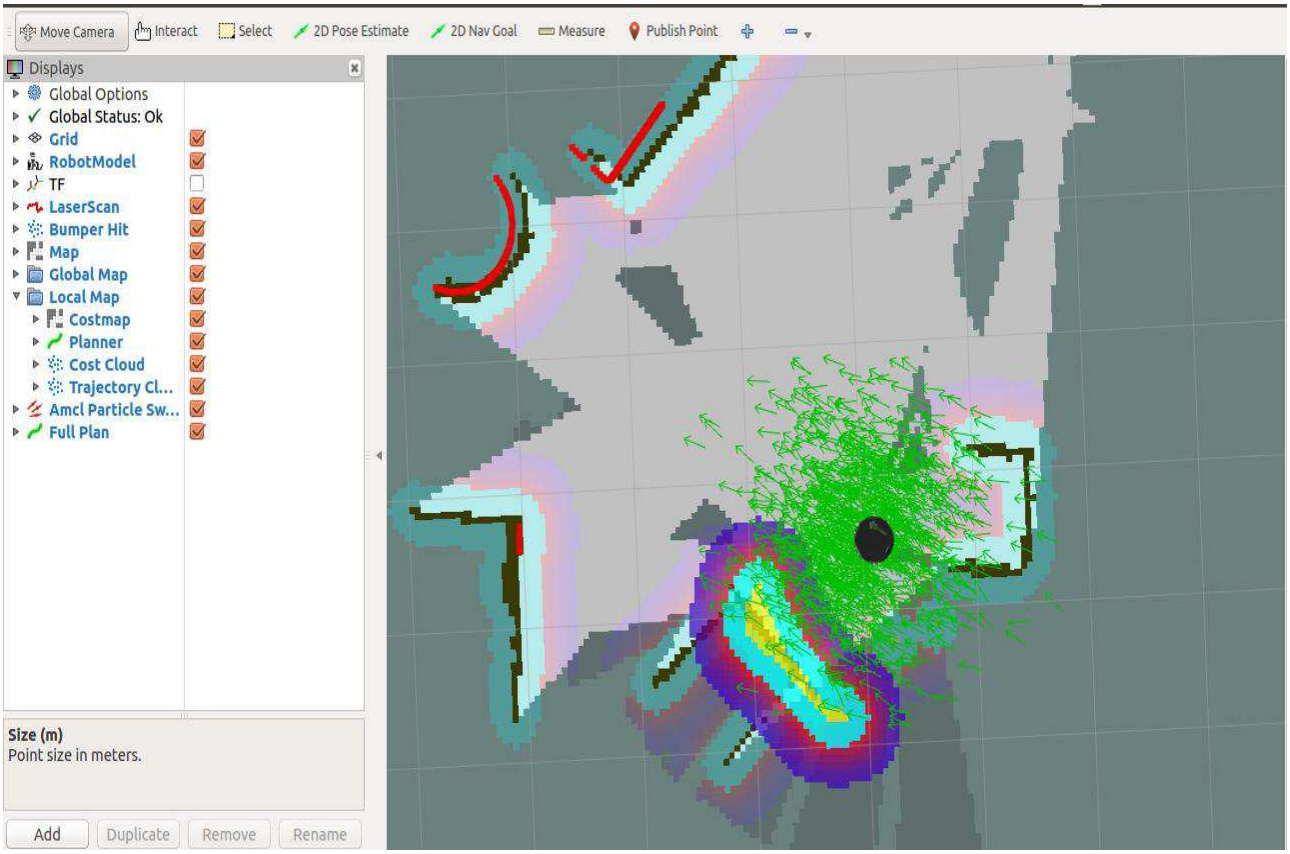


Figura 6. Visualización en RVIZ del funcionamiento de la localización

CAPÍTULO 4. ARQUITECTURA SOFTWARE

Este capítulo describe las piezas del software necesarias para poder aplicar los fundamentos teóricos, explicados en capítulos anteriores, a entornos reales. Hay que entender cómo funciona este software y como es su estructura interna para poder así realizar las implementaciones correspondientes.

4.1 Simulación/Gazebo/ROS (software)

ROS es un sistema operativo standard en el mundo de la robótica. Su uso facilita toda tarea relacionada con robots, esto se debe a la gran cantidad de librerías y herramientas que tiene este software. ROS se encarga de diversas tareas, por ejemplo, control de dispositivos de bajo nivel, abstracción del hardware, paso de mensajes entre procesos, creación de mapas, uso y control de actuadores, etcétera.

Su arquitectura interna está formada por:

- Nodo: es un proceso que realiza cálculos y que gestiona la entrada y salida de información de él. Por ejemplo, un nodo es el encargado de controlar los motores de las ruedas del robot.

- Topic: es un canal que permite conectar dos o más nodos y por los que se produce un intercambio de información. En cada nodo hay que indicar a que topic va a estar conectado y que tipo de mensajes van a intercambiar.

- Mensajes: son estructuras de datos que se envían los nodos mediante los topics. Los mensajes pueden contener datos de tipo integer, float point, arrays, etcétera. ROS cuenta con un conjunto de tipos de mensajes ya definidos.

En los primeros experimentos también se usa Gazebo, que sirve para simular robots y escenarios en un entorno 3D. En estos experimentos, Gazebo es el punto de entrada de información para otros programas.

Gazebo permite simular el robot TurtleBot con el que luego se hacen las pruebas reales. Esto es posible gracias a una librería que viene con ROS, donde hay un archivo URDF (Unified Robot Description Format) en el que está la geometría del robot descrita, como un conjunto de bloques o “links”. Estos bloques pueden tener diferentes colores y formas geométricas, además, sus uniones tienen propiedades físicas como la fricción, de tal manera que se puede obtener un comportamiento del robot simulado muy parecido al de la realidad. Para poder lanzar el robot en el simulador Gazebo, se ejecuta un nodo llamado “spawn_turtlebot_model” y al que se pasa como parámetro el archivo URDF del robot.

Otro elemento importante es la representación del escenario en Gazebo. De nuevo, con las librerías vienen diferentes escenarios. Sin embargo, para realizar experimentos resulta interesante crear uno propio. Su creación se puede hacer desde el entorno grafico de Gazebo o desde Matlab con la Robotics System Toolbox. Esta toolbox facilita la

conectividad de hardware y proporciona algoritmos para el desarrollo de aplicaciones de robótica. Además, ha permitido la automatización de la creación del escenario y, como en Matlab se almacenan las posiciones y dimensiones de cada uno de los obstáculos, se puede obtener la imagen del plano sin necesidad de usar SLAM, con la gran ventaja de que el mapa no presenta imperfecciones debidas a las limitaciones del láser.

Otro software que se ha usado es RVIZ, un visualizador 3D que permite representar visualmente los datos recogidos por los sensores y mostrar información sobre el estado de los robots, tanto en entornos simulados como con robots reales. Para recolectar esta información, se suscribe a los topics que crean los robots. El uso de este programa ha sido muy útil para representar el estado de la abstracción a la vez que se ve como se mueven los robots sobre el mapa.

Todos estos programas se comunican entre ellos con la estructura interna de ROS, es decir, mediante nodos y topics. Para aclarar mejor este funcionamiento se va a analizar un ejemplo de comunicación entre todos los programas, en concreto el de la Figura 7, que ha sido obtenida con la herramienta de ROS “rqt_graph”.

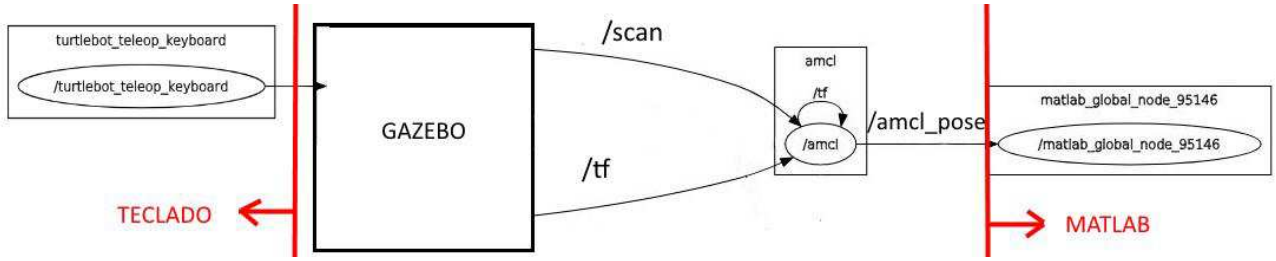


Figura 7. Ejemplo comunicación ROS/Gazebo/Matlab

Ahora se va a describir que hace cada bloque en orden de izquierda a derecha:

- El bloque de teleoperacion se encarga de transformar la pulsación de unas teclas concretas del teclado en movimiento del robot en el simulador. Para ello este bloque envía mensajes con información del movimiento a un nodo interno del bloque de Gazebo llamado “cmd_vel_mux”, el cual se encarga de hacer la conversión de velocidades y enviarlas al robot.

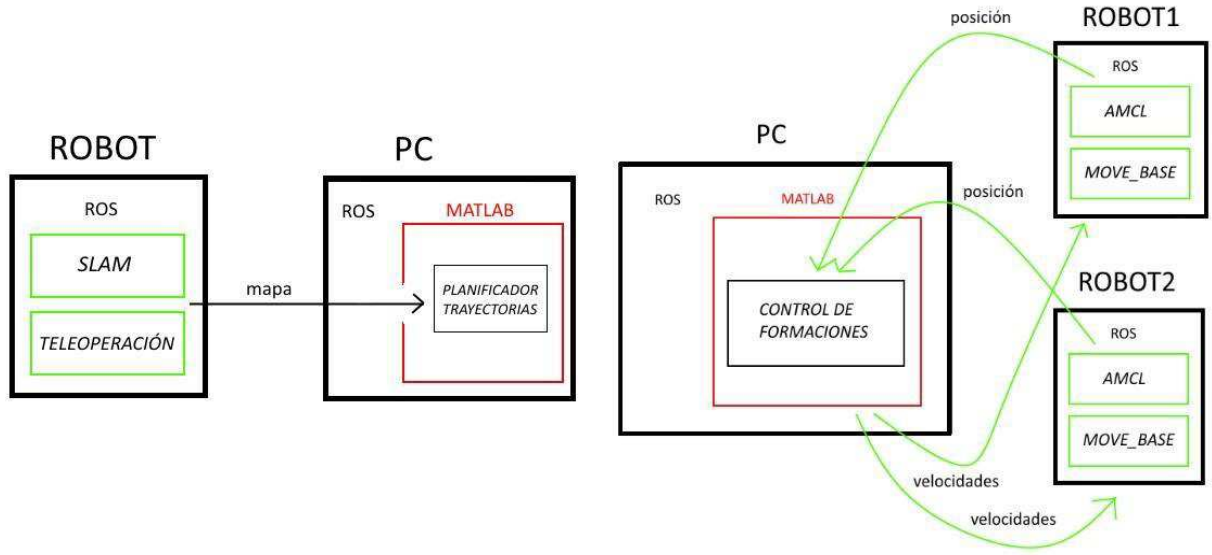
- En el bloque de Gazebo se encuentran muchos nodos ejecutándose cuya misión es que el simulador vaya correctamente, calcular las medidas del láser, realizar la comunicación ROS-Gazebo, etcétera.

- El bloque AMCL es el que ejecuta el algoritmo de localización para calcular las posiciones de los robots. Se puede ver como este bloque recibe las medidas del láser y la relación o transformada del robot respecto a una referencia. Esta información proviene del simulador. La salida del bloque es la estimación de la posición.

- El último bloque se ejecuta dentro de Matlab. Para poder recibir la localización de los

robots desde los códigos que se ejecutan en Matlab es necesario subscribirse al topic “amcl_pose” procedente del bloque anterior.

Realmente, durante los experimentos de navegación, el número de nodos y de conexiones entre ellos es mucho más grande que el que aparece en la Figura 7. Además, antes de ejecutar el bucle de control que mueve los robots (proceso online), hay que obtener el mapa y la planificación de la trayectoria sobre éste (procesos offline). Se han representado en la Figura 8 dos esquemas para ver más claro cómo se produce la secuencia de operaciones y como se conectan los diferentes programas. En el esquema aparecen los nodos y los topics en color verde y los bloques de código de Matlab en rojo.



(a) *(b)*

Figura 8. (a) Procesos offline. (b) Procesos online.

Los procesos offline de la Figura 8 (a) comienzan con el lanzamiento de dos nodos de ROS, que se encargan de controlar el robot por teleoperacion y de ejecutar el algoritmo de SLAM. Con estos nodos se obtiene una imagen del mapa en 2D que es transferida al ordenador. En éste, se usa el planificador de trayectorias de Matlab junto al mapa obtenido para calcular la trayectoria. Conociendo ésta y el mapa, ya se puede ejecutar el bucle de control de la Figura 8 (b). Este control es lanzado en Matlab y su funcionamiento requiere la lectura de las posiciones, las cuales, son proporcionadas por el nodo AMCL de cada robot. El algoritmo de control de Matlab, tras la lectura de las posiciones, calcula las velocidades de cada robot y se las envía al nodo MOVE_BASE. Este nodo traduce los comandos de velocidad en movimiento de las ruedas.

Cuando se realizan las simulaciones sin robots físicos, los nodos que se ejecutan en el robot se sustituyen por nodos de Gazebo que ejercen la misma función. Ademas, Gazebo también publica por su cuenta las localizaciones de los robots. Esta información la publica en el topic “/model_states”, sin embargo, esta localización no proviene de ninguna estimación como el AMCL, simplemente el programa devuelve la posición verdadera de cada objeto que está simulando.

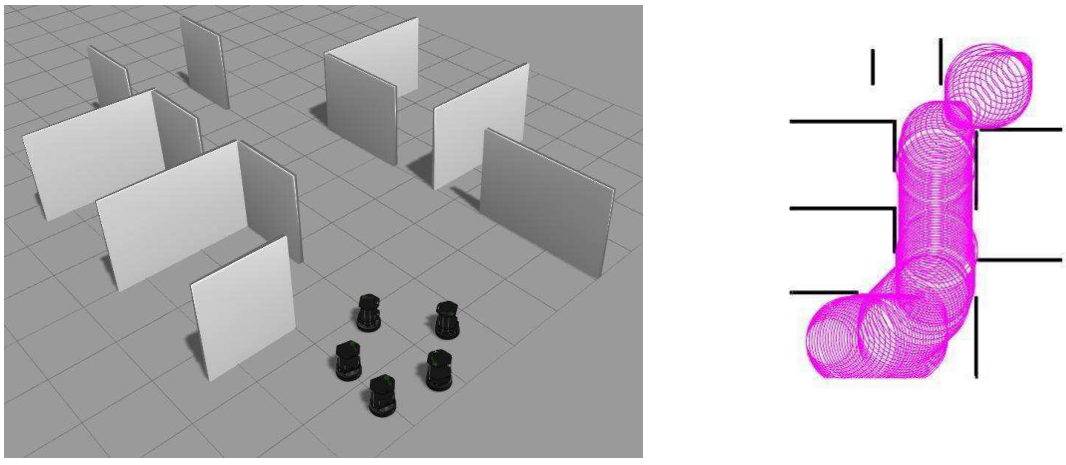
CAPÍTULO 5. RESULTADOS

Se han realizado experimentos tanto en entornos simulados como con robot reales. El objetivo es aplicar los algoritmos explicados y ver cómo se comportan los robots durante el proceso de la navegación en ambos entornos.

5.1 Simulaciones

En el primer experimento se ha realizado la navegación autónoma con un grupo de 5 robots simulados en Gazebo. Éstos parten de una formación en pentágono como se observa en la Figura 9 (a). Para simplificar este experimento, se ha usado la localización que ofrece Gazebo en lugar de usar el algoritmo de localización, esto supone la gran ventaja de que no existe error a la hora de conocer la posición de cada robot.

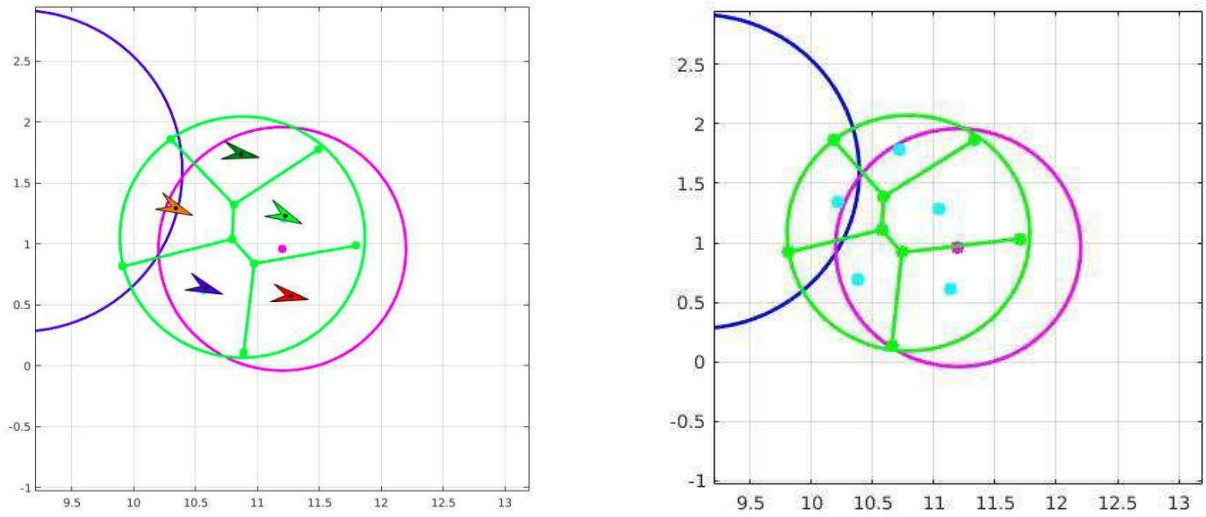
Para este experimento se ha creado un nuevo escenario de lo que podría ser una vivienda o una oficina. Tras la obtención del mapa 2D, se ha calculado con Matlab la trayectoria que se ve en la Figura 9 (b), la cual tiene como punto de inicio uno cercano al centro del pentágono que forman los robots. Además, se ve como el tamaño de la abstracción va variando a lo largo del recorrido con unos diámetros que van desde 0.85 m hasta 1.65 m. El cambio de tamaño más notable es en el último tramo, donde hay una zona estrecha y para ser atravesada por todos los robots, la abstracción se reduce.



(a) *(b)*

Figura 9. (a) Mapa simulado en Gazebo. (b) Planificación de la trayectoria con unos diámetros de abstracción de 0.85 m – 1.65 m.

Durante la ejecución del control en Matlab, se muestra en una imagen como varia el estado de las particiones de Voronoi a lo largo de toda la trayectoria. Un ejemplo de esto se puede ver en la Figura 10, son circunferencias de distintos colores, en azul se muestra la posición de la que parten los robots, en rosa el siguiente destino y en verde las particiones que se calculan en cada iteración. En la circunferencia también aparecen dibujados los centroides con puntos azules como se puede observar en la Figura 10 (b).



(a) *(b)*

Figura 10. (a) Robot en partición Voronoi. (b) Centroides en partición Voronoi

Los centroides son los puntos a los que tienen que desplazarse los robots. En la Figura 10 (a) apenas se parecían los centroides debido a que en cada iteración se calcula una nueva abstracción muy cerca de la actual, por eso en la foto se solapan el dibujo de los robots con los centroides.

Por último, en la Figura 11 aparecen la sucesión de posiciones de cada robot durante la ejecución del experimento. Se observa que siguen perfectamente la trayectoria planificada de la Figura 9 (b).

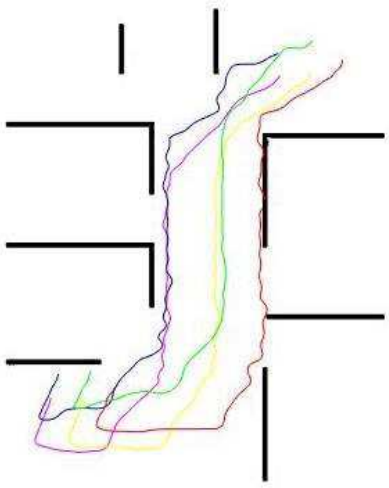


Figura 11. Recorrido seguido por los robots

5.2 Experimentos con robots reales

Se han realizado dos experimentos en el Instituto de Investigación de Ingeniería de Aragón con dos robots TurtleBot.

Para realizar los experimentos con los robots, antes se ha tenido que hacer una preparación de todo el hardware y software, tanto de los robots como de otros componentes. A continuación, se va a explicar esta preparación previa a los experimentos.

Se ha montado una red local con un router para así realizar la comunicación de cada robot con el ordenador que realiza el cálculo de las acciones de éstos. Se habilita un servidor DHCP en este router que permite asociar una IP fija a cada robot. Este servidor usa la MAC que tiene cada ordenador para identificarlos, la MAC es número alfanumérico de 12 dígitos.

Por otro lado, se ha usado el protocolo SSH, que permite acceder desde un ordenador a la terminal de otro, aunque éstos no estén conectados a la misma red local. De esta manera, se pueden ejecutar bloques del software ROS en cada robot desde un ordenador externo.

ROS facilita mucho la comunicación red entre los robots y el ordenador ya que hace transparente el intercambio de mensajes por los topics. Se ha realizado la automatización de la asignación de nombres a los topics, lo que evita fallos en las comunicaciones. Por ejemplo, si el láser de cada robot envía su información por el topic “/scan”, al haber dos robots, habría dos topics “/scan” y crearía solapamientos. Tras la automatización, se asocia el nombre del robot a cada topic, de esta manera, los topics del ejemplo pasan a ser “divcore_NUC_1/scan” y “divcore_NUC_2/scan”.

El experimento comienza de nuevo con la creación del mapa del entorno con un único robot. El resultado se puede ver en la Figura 12.

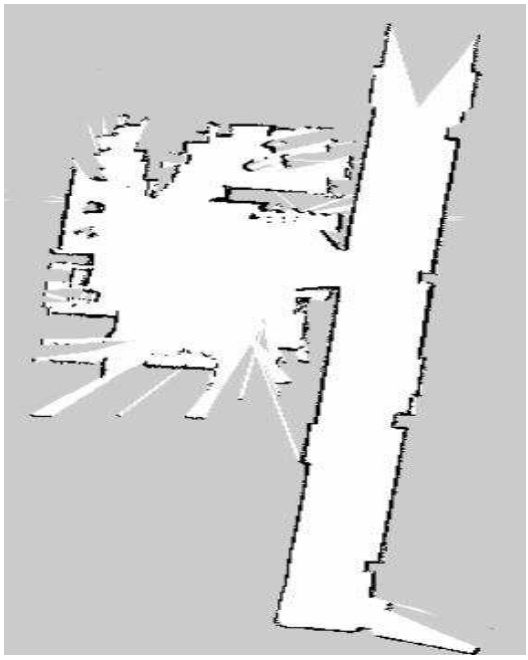


Figura 12. Mapa 2D real obtenido con el laser

Este mapa es guardado en el ordenador que funciona como servidor y mediante el lanzamiento de un nodo de ROS llamado “map_server”, el algoritmo de localización que se ejecuta en el ordenador de cada robot puede acceder a la información del mapa sin necesidad de tenerlo guardado en él. A continuación, se describen las dos pruebas realizadas.

5.2.1 Habitación

Los robots en este experimento se mueven en una gran habitación, la idea de la prueba es mover los robots por la zona de la habitación que está libre de obstáculos para ver como el planificador de trayectorias usa el radio de abstracción más grande que le permite el entorno. Para ver más claro este cambio del tamaño de los radios en la trayectoria, se le ha indicado al planificador que comience y acabe el recorrido con el radio más pequeño de todos los definidos.

La trayectoria planificada se puede ver en la Figura 13 (a) y usa un rango de diámetros que van desde 0.75 m hasta 1.75 m. El resultado de la prueba aparece en la Figura 13 (b), donde se dibuja el recorrido que ha seguido cada robot. A la vista de las imágenes, se puede decir que los robots han seguido la trayectoria planificada correctamente.

Gracias a que el sistema de navegación desarrollado es eficiente, el control está funcionando en tiempo real. Por esta razón, los robots durante los experimentos se han movido sin interrupciones y de forma constante. Además, el algoritmo de localización es robusto frente a pequeños cambios en el entorno, de tal manera, que sigue funcionando aunque las lecturas del láser no concuerden con exactitud con el mapa 2D. Por tanto, solo hace falta crear de nuevo el mapa de una zona cuando se produzcan grandes cambios en ella. Una característica importante es que el funcionamiento del algoritmo no se ve influido por el movimiento de personas alrededor del robot.

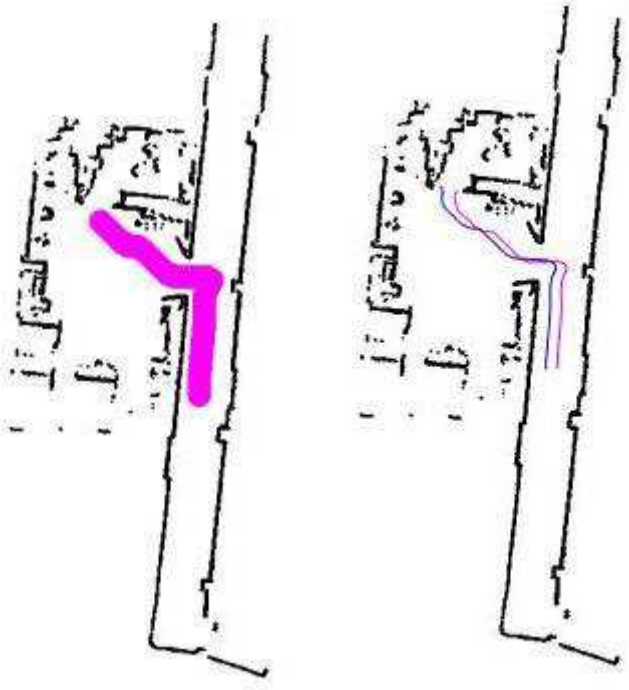
(a) *(b)*

Figura 13. (a) Planificación de la trayectoria para diámetros de 0.75 m – 1.75 m (b) Recorrido trazado por los robots

5.2.1 Habitación-Pasillo

El experimento consiste en que los robots partiendo del interior de la habitación lleguen a un punto medio del pasillo. El mayor reto está en que atraviese los dos robots la puerta debido que es una zona estrecha.

Se ha planificado la trayectoria de la figura 14 (a) para un diámetro de la abstracción de 0.75 m. A su lado (b) está el recorrido que han seguido los robots en la prueba.



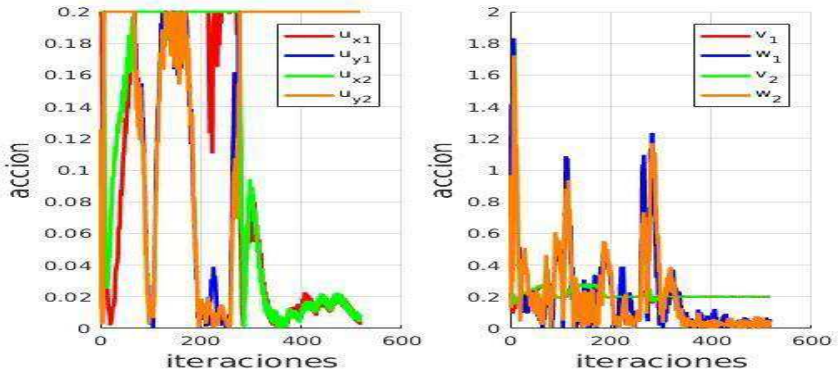
(a) *(b)*

Figura 14. (a) Planificación de la trayectoria para un diámetro de 0.75 m. (b) Recorrido de los robots.

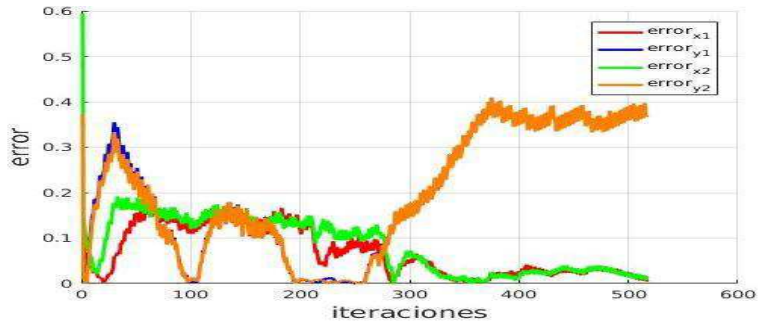
Para ver mejor el comportamiento de los robots durante el experimento, se han graficado los datos obtenidos en el control que ejecuta Matlab. En la Figura 15 (a) se observa las acciones referidas a ejes absolutos. Ambos robots tienen unas acciones que se saturan en varios tramos, alcanzado el valor máximo de la acción. Las acciones son proporcionales al error, cuando éste sobrepasa un valor, la acción se satura.

En la Figura 15 (b) aparecen las acciones lineales y angulares, las oscilaciones en las acciones angulares significan cambios de orientaciones y las acciones lineales, dibujadas en verde y rojo, toman valores prácticamente constantes en las zonas donde la trayectoria es casi lineal

El error se puede observar en la Figura 15 (c) y se calcula como la diferencia de la posición del robot al centroide de la abstracción. El problema está, en que algunas veces la lectura de posiciones no es precisa del todo, entonces, aunque el robot realmente esté cerca del centroide, si el algoritmo de localización devuelve una posición lejana al centroide, la acción que se aplica al robot es muy grande.



(a) *(b)*



(c)

Figura 15. (a) Acción de cada robot referido en ejes absolutos. (b) Acciones lineales y angulares. (c) Errores de posición respecto al centroide.

En las últimas iteraciones de la Figura 15 (c), aparece un error pequeño respecto el eje X en ambos robots, sin embargo, respecto al eje Y, el error es mayor, cerca de 40 cm. Esta ocurrencia se puede explicar con la Figura 16, en ella se ve que cuando los robots están ya en el pasillo, se encuentran mal localizados ya que los haces de sus láseres, dibujados con color azul y verde, no concuerdan con el mapa, sino que están desplazados en el eje Y.

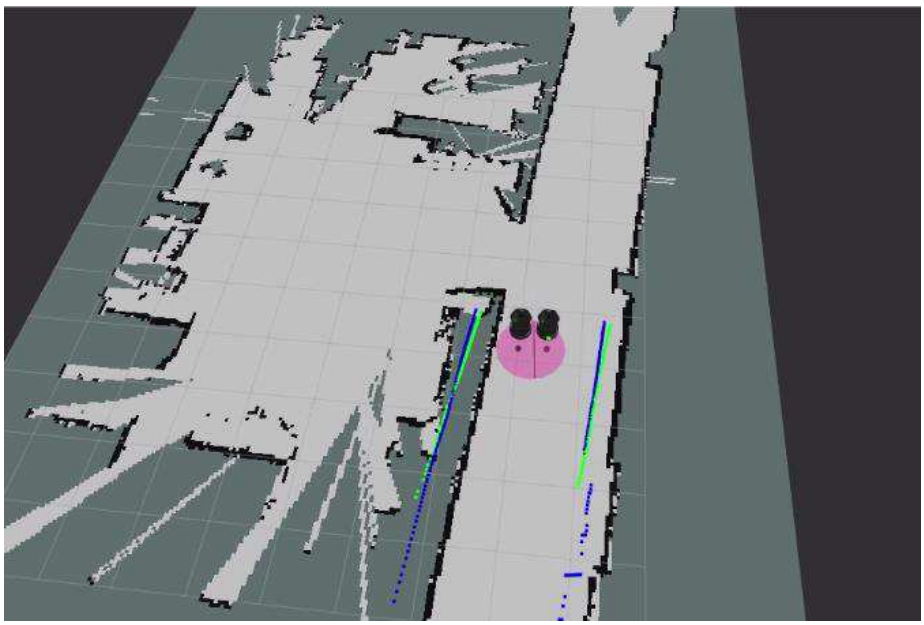


Figura 16. Robots visualizados en RVIZ mal localizados en el mapa

Este error de ubicación en la zona del pasillo se debe a que el algoritmo de localización no trabaja bien en zonas simétricas, ya que no es capaz de resolver la ambigüedad sobre la posición usando información geométrica de su entorno. La solución es añadir otra técnica de localización que permita eliminar la ambigüedad en la posición. Usar una nueva técnica supone, incorporar nuevos sensores al robot o al entorno, desarrollar el nuevo algoritmo de localización y adaptar ambas técnicas para que trabajen juntas. Por ello, solucionar los errores de posición en zonas simétricas queda fuera del alcance de este trabajo.

Sin embargo, a pesar de este error en zonas simétricas, los robots han podido moverse bien por el pasillo. En el resto de zonas, el algoritmo ha funcionado bastante bien, por lo que atravesar la zona estrecha de la puerta no ha supuesto ningún problema. Además, los robots de nuevo han mantenido una velocidad uniforme a lo largo de todo el recorrido y en ningún momento se han salido de su región de la abstracción, lo que asegura la ausencia de colisiones entre ellos

CAPÍTULO 6. CONCLUSIONES FINALES

6.1 Conclusiones

En este proyecto se ha puesto en funcionamiento un sistema de navegación y control de sistemas multirobot en entornos reales con obstáculos. Para ello, se ha realizado un estudio de métodos de localización, planificación de trayectorias y control multirobot.

Se han adaptado las implementaciones en Matlab del trabajo de fin de grado [2] y el proyecto fin de carrera [1] para que funcionen juntos y también para que se puedan comunicar con la plataforma ROS.

Como este control se basa en la posición de cada robot, se han estudiado diferentes métodos para la localización de los robots. Este estudio ha concluido con el uso de un algoritmo que se basa en un plano 2D de la zona donde se va a realizar la navegación y en la información recogida por sensores del robot.

Con todo lo anterior, se han llevado a cabo experimentos tanto en un entorno de simulación realista como con robots reales.

El resultado final en los experimentos reales es que los robots han seguido la trayectoria planificada bastante bien en muchos casos y otras veces con pequeñas desviaciones debidas a los errores de posición. En ambos casos, en ningún momento los robots han estado fuera de su región en la abstracción y siempre han mantenido movimientos sin cambios bruscos y con una velocidad moderada. Por lo que se puede decir que se han cumplido los objetivos propuestos de manera satisfactoria.

6.2 Líneas futuras

Hay varios puntos, que están fuera del alcance de este trabajo, pero que sería interesante realizar un estudio más detallado ya que podrían mejorar los resultados obtenidos

La navegación en entornos sencillos se ha realizado bastante bien. Sin embargo, cuanto más complejo se hace el entorno, más acumulación de errores hay por parte de cada bloque que compone la navegación. La solución podría consistir en añadir nuevos sensores a los robots o también utilizar algoritmos más sofisticados que combinen la información de varios robots.

Por otro lado, podría ser interesante usar varias formas de la abstracción durante la navegación. Algunas de estas formas podrían ser circunferencias, elipses o rectángulos. De tal manera, que cuando haya pasos estrechos, se use aquella formación que mejor se adapte al paso y así dar versatilidad al movimiento de los robots.

BIBLIOGRAFÍA

[1] Vera, D. (2011). Planificación y control con restricciones de formaciones de robots. [Proyecto fin de carrera]. Universidad de Zaragoza. Recuperado el 10 de febrero de 2017, de https://zaguan.unizar.es/record/6479?ln=es

[2] Pascual, B. (2016). Control de formaciones multirobot con visión. [Proyecto fin de carrera]. Universidad de Zaragoza. Recuperado el 10 de febrero de 2017, de https://zaguan.unizar.es/record/60807?ln=es

[3] Cortés, J; Martínez, S; Karatas, T; Bullo, F. Coverage control for mobile sensing networks. IEEE Transactions on Robotics and Automation (ISSN 1042-296X), DOI: 10.1109/TRA.2004.824698, Vol. 20,2004

[4] Grisetti, G; Stachniss, C; Burgard, W. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. IEEE Transactions on Robotics (ISSN 1552-3098), DOI: 10.1109/TRO.2006.889486, Vol. 23,2007.

[5] Gómez, J; Soriano, A; Vallés, M; Valera, A; Martínez, M. Implantación de un algoritmo de localización basado en un método de Montecarlo para un robot móvil omnidireccional. XXXIII Jornadas de Automática, Vigo, 5 al 7 de septiembre de 2012

[6] Michael, N; Kumar, V. Planning and Control of Ensembles of Robots with Nonholonomic Constraints. The Int. Journal of Robotics Research, 2009, Vol. 28. 962–975.