BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

UN FRAMEWORK BASADO EN ROS PARA LA NAVEGACIÓN DE ROBOTS MÓVILES AUTÓNOMOS

TESIS PROFESIONAL

Para obtener el título de:

Ingeniero en Tecnologías de la Información

Presenta:

Rosa Laura Villarreal Onofre

Asesor:

Dr. Abraham Sánchez López

México, Puebla Noviembre, 2021

Agradecimientos

A mis padres, quienes durante todos mis años de formación me han apoyado incondicionalmente para cumplir mis metas, brindándome amor y cariño en cada etapa de mi vida. Sepan que todo el esfuerzo y sacrificios que han realizado son la base de la persona que aspiro a ser. Este logro es compartido, y jamás encontraré las palabras que describan el infinito agradecimiento y respecto que siento hacia ustedes. Papá, gracias por ser el guía de mi vida, el maestro del que nunca dejaré de aprender. Mamá, gracias por tus cuidados, por tu amor y más importante, por ser mi amiga y confidente.

A mi hermana Fernanda, te agradezco por siempre estar para mí, por los momentos buenos y malos que hemos pasado, gracias por siempre creer en mí y apoyarme incondicionalmente en mis sueños y metas.

A mi asesor de tesis, el Dr. Abraham, por todo el conocimiento que recibí de su parte, por acompañarme durante toda la carrera. Agradezco el que me haya invitado a formar parte del laboratorio MOVIS en donde no sólo aprendí demasiado, pero también conocí a personas invaluables para mí. Siempre tendrá mi respeto y admiración.

Al MC. Juan Carlos Conde, al que considero un amigo, y al Dr. Alfredo Toriz. Gracias por su guía, por aquellas clases que renovaron mi espíritu para seguir aprendido, son unos excelentes docentes que me inspiran confianza y admiración.

A mi mejor amigo Luis, así como a mis queridos amigos Alberto y Esteban, gracias por su paciencia, apoyo y cuidado que tuve la fortuna de recibir, por siempre animarme. A

mis amigos Jiuber y Pablo por todas las aventuras y vivencias que tuvimos, aprendí un poco de la vida en cada una de ellas. Todos ustedes son mi segunda familia.

A todos los profesores y amigos que tuve la dicha de conocer durante los últimos años, siempre serán parte de los recuerdos más preciados de esta etapa tan importante de mi vida.

A la Benemérita Universidad Autónoma de Puebla y la Facultad de Ciencias de Computación, me voy con la certeza de que escogí el lugar correcto para completar mi formación profesional.

Índice General

Índice de Figuras

Índice de Tablas

Introducción

Posiblemente suene difícil de imaginar, pero se cree que desde el año 300 a. C. la robótica dio sus primeros pasos cuando Aquitias de Tarentum construyó un pájaro mecánico capaz de agitar sus alas y sobrevolar a una altura por encima de los 200 metros, movido por una corriente viento. Desde entonces, y a través de los años, el campo de la robótica ha crecido de gran manera: robots móviles autónomos, humanoides, brazos mecánicos, etc. por lo que las áreas para la investigación han aumentado en consecuencia. Una de estas áreas es precisamente la navegación: ¿Dónde se encuentra el robot?, ¿Hacia dónde va?, ¿Cómo llegará ahí? Las respuestas a estas preguntas han sido un tema de discusión desde su planteamiento, y en la actualidad, existen diversas técnicas para dar solución a estas interrogantes.

Es por ello que este trabajo de tesis se enfoca en la investigación y propuesta de un framework para la navegación autónoma de robots móviles, basado en el Sistema Operativo Robótico (ROS), que permita implementar las diversas tareas relacionadas en una serie de pasos lógicos y sencillos para integrar el sistema de navegación en un robot móvil. Mediante la construcción de un solo ecosistema, se busca reducir el tiempo empleado para configurar las actividades y requisitos para la navegación, al mismo tiempo que se simplifica el proceso.

El motivo principal que impulsa esta investigación, es realizar una contribución a la comunidad inmersa en esta área, en especial a aquellos que utilizan ROS para el desarrollo de distintas aplicaciones robóticas.

El objetivo general de este trabajo es establecer una herramienta de utilidad para su uso dentro de la comunidad de ROS. De manera específica, se espera:

1. Implementar las actividades de la pila de navegación en una sola ventana.
2. Contar con distintos prototipos de robots debidamente configurados, según su tipo, para hacer funcionar el sistema de navegación.
3. Incorporar varios ambientes y mapas predefinidos para su uso.
4. Mostrar en una visualización 3D la agregación continua de diversos elementos gráficos, resultantes en cada paso de la pila.
5. Disminuir el tiempo requerido para implementar un sistema de navegación en distintos robots.
6. Contribuir a la comunidad con un proyecto abierto para su futura colaboración y mejora.

Dicho lo anterior, el presente documento se encuentra estructurado de la siguiente manera:

El Capítulo 1 presenta el estado del arte referente a la navegación, indagando en cada una de las actividades que la componen y explicando su estructura y formulación. Se habla sobre la navegación, la localización y la planificación de movimiento.

En el Capítulo 2 se discuten algunos de los algoritmos más importantes para la planificación, así como sus variantes y diferencias: Dijstra, A*, D*, RTT. Esta lista está basada en los algoritmos mayormente utilizados por ROS.

El Capítulo 3 brinda una introducción al Sistema Operativo Robótico, a sus conceptos y paquetes más importantes utilizados para el desarrollo de la propuesta de esta tesis.

En el Capítulo 4 se describe la forma tradicional de implementar un sistema de navegación con la pila de navegación de ROS y se describe la propuesta del framework a desarrollar como resultado de la investigación realizada.

El Capítulo 5 muestra el resultado de la propuesta realizada, describe el uso y resultados obtenidos a través de cada paso que incluye la solución.

Finalmente, en el Capítulo 6 se discuten las conclusiones obtenidas de la elaboración de la propuesta presentada, así como los posibles trabajos futuros a realizar a partir de la investigación y trabajo documentado a lo largo de esta tesis.

Capítulo 1

1 Estado del arte

En este primer capítulo se realiza una introducción a los temas relacionados a la navegación de robots móviles, base de este trabajo de tesis. Si bien el enfoque principal se centra en la navegación basada en un mapa dado y no en su construcción y actualización continua, es necesario conocer los principios que la componen.

1.1 Navegación

La necesidad de incrementar la autonomía en las aplicaciones robóticas ha motivado la creación y desarrollo de robots móviles. El propósito es limitar en todo lo posible la intervención humana. Para ello, el robot debe poseer la suficiente inteligencia para reaccionar y tomar decisiones basándose en observaciones de entorno. La autonomía de un robot móvil se basa principalmente en el sistema de navegación autónoma [1].

La Enciclopedia Británica define la navegación como *"ciencia de dirigir una nave determinando su posición, rumbo y distancia recorrida. La navegación se ocupa de encontrar el camino hacia el destino deseado, evitando colisiones, ahorrando combustible y cumpliendo con los horarios"*. En este sentido, la navegación de un robot móvil es entonces, la habilidad de un robot móvil de ir de un punto a otro de una manera ordenada [2].

Las tareas involucradas en la navegación de un robot móvil son: la percepción del entorno a través de sus sensores, de modo que le permita crear una abstracción del mundo; la planificación de una trayectoria libre de obstáculos, para alcanzar el punto destino seleccionado; y el guiado del vehículo a través de la referencia construida.

Realizar una tarea de navegación para un robot móvil significa recorrer un camino que lo conduzca desde una posición inicial hasta otra final, pasando por ciertas posiciones intermedias o submetas. El problema de la navegación se divide en las siguientes cuatro etapas:

- *Percepción del mundo*: Mediante el uso de sensores externos, creación de un mapa o modelo del entorno donde se desarrollará la tarea de navegación (González, 1993).
- *Planificación de la ruta*: Crear una secuencia ordenada de objetivos o submetas que deben ser alcanzadas por el vehículo. Esta secuencia se calcula utilizando el modelo o mapa de entorno, la descripción de la tarea que debe realizar y algún tipo de procedimiento estratégico.
- *Generación del camino*: En primer lugar, define una función continua que interpola la secuencia de objetivos construida por el planificador. Posteriormente procede a la discretización de la misma a fin de generar el camino.
- *Seguimiento del camino*: Efectúa el desplazamiento del vehículo, de acuerdo al camino generado mediante el control adecuado de los actuadores del vehículo (Martínez, 1994).

Estas tareas pueden llevarse a cabo de forma separada, aunque en el orden especificado. La interrelación existente entre cada una de estas tareas conforma la estructura de control de navegación básica en un robot móvil y se muestra en la Figura 1.1.

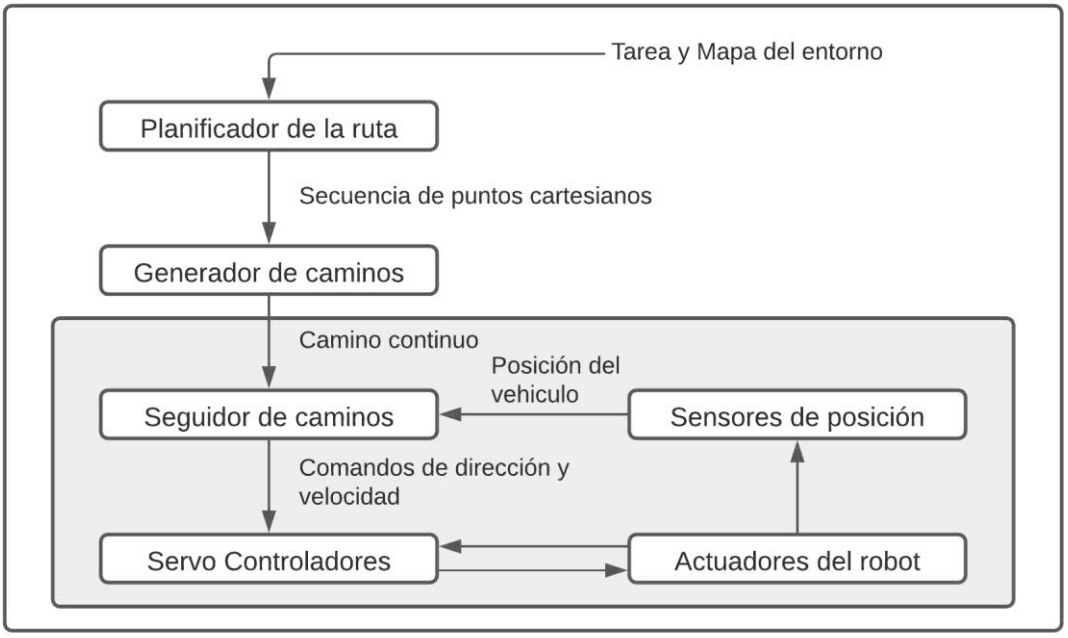

Figura 1.1 Estructura de control de navegación básica para un robot móvil

En el esquema anterior, se parte de un mapa de entorno y de las especificaciones de la tarea de navegación. De estos datos se realiza la planificación de un conjunto de objetivos representados como una secuencia de puntos cartesianos dispersos que definen la ruta. Dicho conjunto cumple los requisitos de la tarea impuesta asegurándose de que la ruta asociada está libre de obstáculos. Mediante el uso del generador del camino se construye la referencia que utilizará el seguidor de caminos para generar los comandos de direccionamiento y velocidad que actuarán sobre los servo controladores del vehículo. Por último, mediante el uso de los sensores internos del vehículo (sensores de posición) en conjunción con técnicas odométricas, se produce una estimación de la posición actual (Cox, 1991), la cual será realimentada al seguidor de caminos.

La complejidad del sistema necesario para desarrollar esta tarea depende principalmente del conocimiento que se posee del entorno de trabajo. Así, la Figura 1.1

considera que se cuenta con un mapa del entorno que responde de forma fiel a la realidad. Mediante el uso adecuado del mismo se puede construir un camino que cumpla los requisitos impuestos por la tarea de navegación, sin que el vehículo colisione con algún elemento del entorno.

Sin embargo, es posible que el modelo del entorno del que dispone el robot adolezca de ciertas imperfecciones al omitir algunos detalles del mismo. El esquema presentado en la Figura 1.1 resulta ineficaz, al no asegurar la construcción de un camino libre de obstáculos. Por ello se necesita introducir en la estructura de control básica nuevos elementos que palien este defecto. Un esquema de navegador utilizado en aplicaciones, donde la información acerca del entorno de trabajo varía desde un perfecto conocimiento del mismo hasta poseer un cierto grado de incertidumbre, es el mostrado en la Figura 1.2. Dicho sistema corresponde al implantado en el robot móvil Blanche de los laboratorios AT&T (Nelson y Cox, 1990).

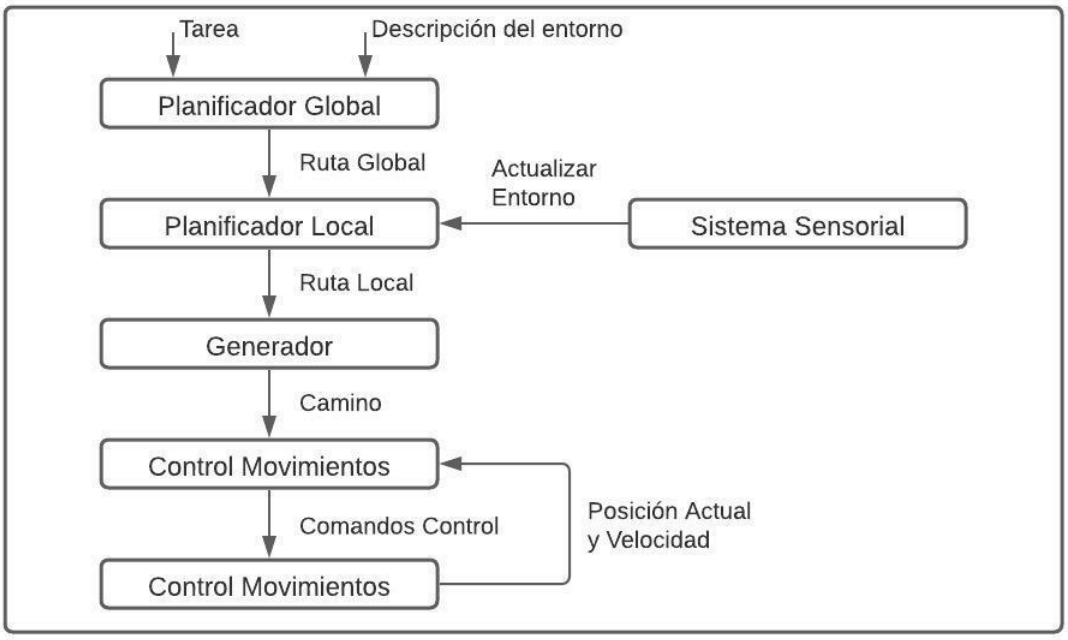

Figura 1.2 Navegador implantado en el robot móvil Blanche de AT&T

El esquema presentado contiene en líneas generales, el funcionamiento de la estructura de navegación básica. Lo novedoso reside en el desdoblamiento de la tarea de planificación en dos subtareas: planificación global y local. La primera de estas subtareas es análoga al módulo de planificación de la Figura 1.1 y construye una ruta sobre la cual se puede definir un camino libre de obstáculos según la información que a priori se posee del entorno. Si la descripción del entorno introducida fuese perfecta, la ruta calculada sería de forma directa la entrada de la tarea generador. Sin embargo, al no serlo, puede dar lugar a la construcción de un camino que no esté libre de obstáculos, con el consiguiente peligro de que el vehículo impacte con algún elemento del entorno. La tarea de planificación local recibe información del sistema sensorial sobre el entorno local del robot, según el radio de alcance de los sensores externos de a bordo. Mediante el análisis de estos datos actualiza el modelo preliminar del entorno y decide si se precisa replanificar la ruta local del robot.

La clave del esquema presentado en la Figura 1.2 para adaptarse a diversos entornos, aunque no se posea un conocimiento exhaustivo del mismo, reside en la distinción efectuada entre planificación global y local. Ambos conceptos se pueden definir con mayor precisión de la forma que sigue (Levi, 1987):

- **Planificación global:** Construir o planificar la ruta que lleve al robot a cada una de las submetas determinadas por el control de misión, según las especificaciones del problema que debe resolverse. Esta planificación es una aproximación al camino final que se va a seguir, ya que en la realización de esta acción no se consideran los detalles del entorno local al vehículo.
- **Planificador local:** Resolver las obstrucciones sobre la ruta global en el entorno local al robot para determinar la ruta real que será seguida. El modelo del entorno local se

construye mediante la fusión de la información proporcionada por los sensores externos del robot móvil.

La construcción de la ruta global puede realizarse antes de que el vehículo comience a ejecutar la tarea, mientras que la planificación local se lleva a cabo en tiempo de ejecución. En el caso de realizar una navegación sobre entornos totalmente conocidos es obvio que resulta innecesario proceder a una planificación local, pero a medida que disminuye el conocimiento de la zona por la cual el robot móvil realiza su tarea, aumenta la relevancia de la misma.

En aplicaciones de navegación en exteriores o de campo (Daily, 1988; Brunitt y otros, 1992) el conocimiento que se posee del entorno es pobre y por tanto se necesita hacer un uso intensivo de la planificación local. En el robot móvil Navlab II de CMU (Goto y Stentz, 1987) la navegación está confiada totalmente al planificador local, de suerte que el camino que debe seguir se construye de forma dinámica a medida que se navega. El esquema empleado recurre a un uso más intenso del sistema sensorial que en el caso del Blanche, y se le responsabiliza de la coordinación de la percepción, planificación y control del vehículo para guiarlo por el camino especificado, mientras verifica el entorno, y realizar el sorteo de obstáculos. La interacción de cada uno de los módulos en este esquema de navegación local queda representada en la Figura 1.3.

El funcionamiento de este navegador local está basado en la realización de un ciclo de construcción del mapa local del entorno inmediato al robot móvil, la elección de una ruta segura por la cual puede pasar el vehículo. De acuerdo con la información suministrada, se procede a construir el camino, y, por último, realizar el seguimiento. La iteración de este

ciclo ocurre cada vez que el control de movimientos termina de seguir el camino actual, pasándose en ese momento a la construcción del próximo mapa local de entorno.

La diferencia entre los esquemas de navegador presentados radica en la adaptación que debe poseer el robot móvil para moverse por su entorno de trabajo según el conocimiento del cual disponga sobre la estructura del mismo, dándose mayor ponderación a la planificación global o local. Sin embargo, mantienen un nexo común en la realización de forma secuencial y continua de las operaciones de percepción, planificación-generación y seguimiento. Este grupo de acciones permite el desarrollo de la navegación minimizando cierto índice de coste, como puede ser la distancia recorrida o el consumo energético del vehículo. Se habla en este caso de una descomposición jerárquica en módulos funcionales que encadenados en forma de ciclo realizan la denominada navegación estratégica (Brooks, 1986). Además, este tipo de navegación se caracteriza por la necesidad de conocer, con el mínimo error, posible la posición actual del vehículo, ya que la realimentación de esta información es la base para el cálculo de la próxima acción de control. Mediante el uso de la odometría del vehículo se puede realizar esta acción, pero debido a la naturaleza del método y a las características de los sensores utilizados, la estimación efectuada se ve afectada por errores acumulativos (Watanabe y Yuta, 1990). Cuando dichos errores alcanzan niveles indeseables se hace necesario eliminarlos mediante el uso de algoritmos de estimación de la posición basados en referencias externas (González J., 1993).

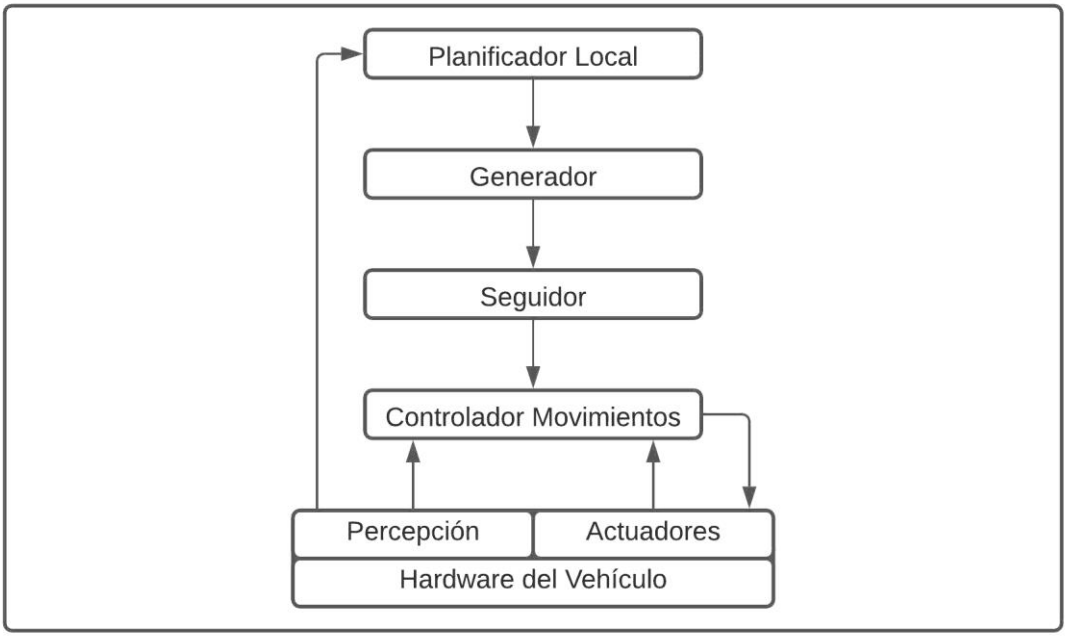

Figura 1.3 Navegador local implantado en el Navlab II de CMU

La navegación estratégica tiene sus limitaciones en entornos dinámicos no conocidos, ya que requiere un completo conocimiento de la dinámica de los posibles obstáculos móviles, además de una adecuada actualización del mapa de entorno [3].

En general, para resolver el problema de la navegación de robots, se necesita encontrar respuesta a las siguientes preguntas: ¿Dónde está el robot?, ¿A dónde se dirige?, ¿Cómo llegará ahí?

1.2 Localización

La localización de robots proporciona una respuesta a la pregunta: *¿Dónde está el robot?* Se requiere una solución confiable a esta pregunta que permita realizar tareas útiles, ya que el conocimiento de la ubicación actual es esencial para decidir qué hacer a continuación [4, 5]. Esta sección se centra en las soluciones al problema de la localización

de robots cuando el mapa de su entorno está disponible. El problema se convierte entonces en una estimación de la pose del robot (posición y orientación) en relación con el marco de coordenadas en el que se define el mapa.

Un robot móvil equipado con sensores para monitorear su propio movimiento puede calcular un estimado de su localización en relación con el lugar donde comenzó si se dispone de un modelo matemático del movimiento. Esto se conoce como odometría o *dead reckoning*. Los errores presentes en las mediciones del sensor y el modelo de movimiento hacen que las estimaciones de ubicación del robot obtenidas del *dead recknoning* sean cada vez menos fiables a medida que el robot navega en su entorno. Los errores en las estimaciones de *dead recknoning* pueden ser corregidos cuando el robot observa su entorno usando sensores y es capaz de correlacionar la información contenida en un mapa.

La formulación del problema de la localización en robots depende del tipo de mapa disponible, así como de las características de los sensores utilizados para observar su entorno. En una posible formulación, el mapa contiene ubicaciones de algunos puntos de referencia o características prominentes presentes en el entorno y el robot puede medir el alcance y/o el rumbo de estas características en relación con el robot. Alternativamente, el mapa podría tener la forma de una cuadrícula de ocupación (Occupancy grid) que proporcione las regiones ocupadas y libres de un entorno y así los sensores a bordo del robot pueden medir la distancia a la región ocupada más cercana en una dirección determinada. Dado que la información de los sensores suele estar alterada por el ruido, es necesario estimar no sólo la ubicación del robot, sino también la medida de la incertidumbre asociada con la estimación de la ubicación. El conocimiento de la confiablidad de la estimación de ubicación juega un papel importante en los procesos de toma de decisiones utilizados en robots móviles, ya que pueden producirse

consecuencias catastróficas si se toman decisiones asumiendo que las estimaciones de ubicación son perfectas cuando no lo son. El filtrado bayesiano [6] es una técnica poderosa que puede ser aplicada para obtener una estimación de la localización de un robot y la incertidumbre asociada. Tanto el filtro de Kalman extendido (EKF) como el filtro de partículas proporcionan aproximaciones manejables al filtrado bayesiano.

1.2.1 Modelo de vehículo y modelos del sensor

Los modelos matemáticos que describen el comportamiento del robot y los sensores montados en él son los componentes más importantes en la formulación del problema de localización del robot. El modelo cinemático del vehículo describe las ecuaciones que gobiernan el movimiento del robot en respuesta a las acciones de control. La Figura 1.4 ilustra un robot de accionamiento diferencial que opera en un plano bidimensional donde la velocidad de avance y la velocidad angular del cuerpo del robot se pueden controlar utilizando dos motores que accionan las dos ruedas. La ecuación diferencial que describe cómo la posición y la orientación del robot evolucionan con el tiempo, en función de su velocidad angular y de avance, se conoce como modelo de movimiento del robot.

La relación entre las observaciones de los sensores y la ubicación del robot en el mapa se conoce como modelo de sensor. El modelo de sensor depende de las características del sensor montado en el robot, así como de la forma en que se representa el mapa del entorno. El mapa del entorno generalmente se define usando coordenadas de puntos de referencia o características conocidas, o en forma de una cuadrícula de ocupación donde el estado de cada celda de la cuadrícula define si el área representada por la celda está ocupada o no. La Figura

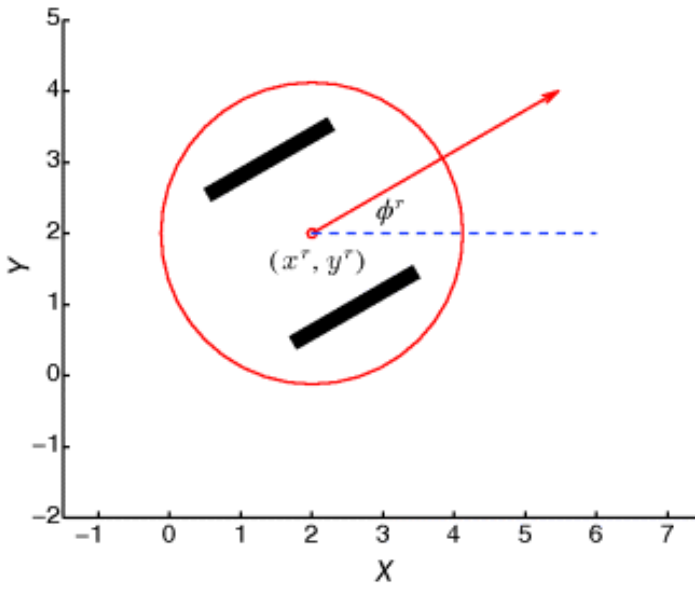

Figura 1.4 Robot diferencial operando en un plano de dos dimensiones

1.5 ilustra un mapa con cuatro puntos de referencia (puntos negros) que se encuentran ubicados en el entorno en (2, 5), (2, - 2), (4, 5), (4, - 2), respectivamente. El robot (el círculo rojo que muestra la posición del centro del robot, la flecha roja que muestra la orientación) comienza desde (0, 0, 0) T en el tiempo 0. En los pasos de tiempo 1 y 2, observa los puntos de referencia 1 y 2; en el paso de tiempo 3, observa los puntos de referencia 1 y 3, en el paso de tiempo 4, observa los puntos de referencia 3 y 4. Las observaciones de robot a punto de referencia se indican mediante líneas azules.

Por otro lado, la Figura 1.6 muestra un mapa de cuadrícula de ocupación donde las áreas ocupadas están sombreadas y el espacio libre está representado por el área blanca. El robot se mueve unos pasos (la pose de un robot se muestra como un círculo más una flecha); las lecturas del telémetro láser en la primera pose (rojo) se representan como líneas rojas.

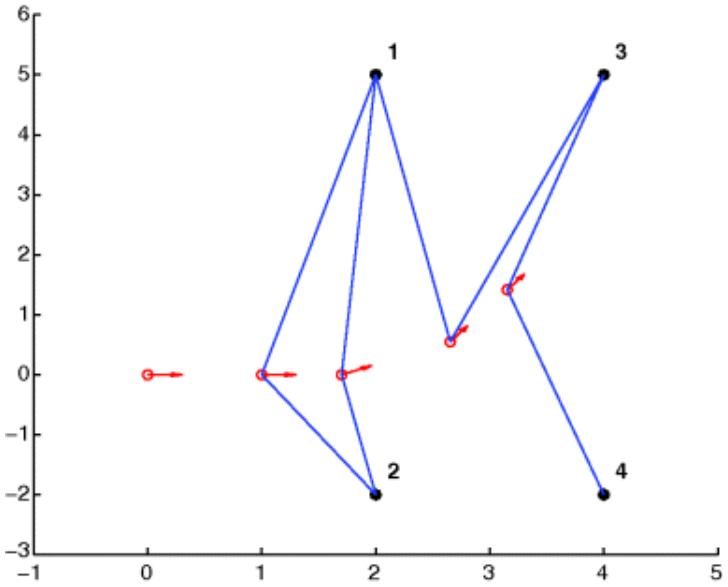

Figura 1.5Problema de localización con un mapa basado en puntos de referencia

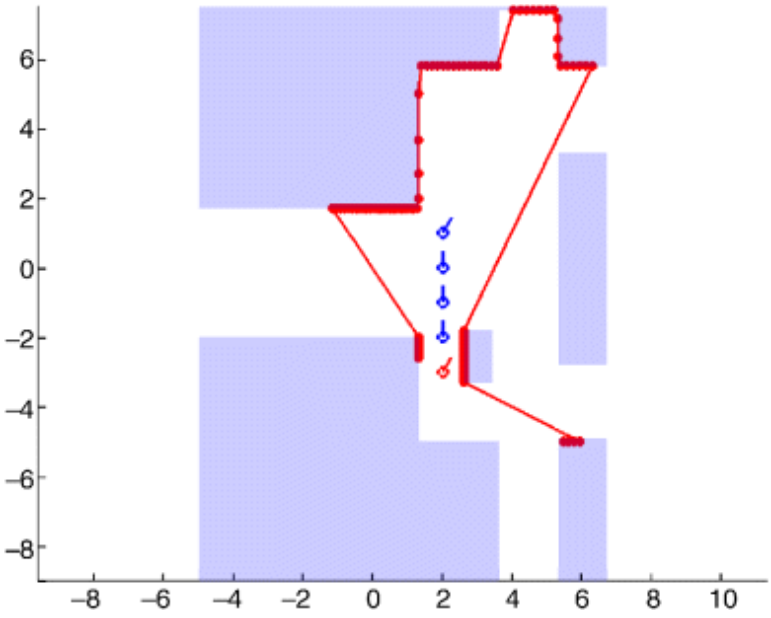

Figura 1.6 Problema de localización con un mapa de cuadrícula de ocupación

En las siguientes secciones, se describen un modelo de vehículo y los modelos de sensor que se utilizan normalmente para la localización de robots. Por simplicidad y claridad de la notación, se asume que el robot tiene tres grados de libertad y se mueve en un plano bidimensional.

Modelo de vehículo

Las ecuaciones cinemáticas que gobiernan el movimiento del robot de accionamiento diferencial ilustrado en la Figura 1.4 están dadas por

$$\begin{aligned}\dot{x}^r(t) &= (\upsilon(t) + \delta\upsilon(t))\cos(\varphi^r(t)) \\ \dot{y}^r(t) &= (\upsilon(t) + \delta\upsilon(t))\sin(\varphi^r(t)) \\ \dot{\varphi}^r(t) &= \omega(t) + \delta\omega(t)\end{aligned}$$

Ecuación 1.1

donde las coordenadas $x^r(t)$ y $y^r(t)$ describen la posición del centro del robot móvil en el tiempo t, la orientación $\varphi^r(t)$ es el ángulo entre el rumbo del robot y el eje x del marco de coordenadas global. $\dot{x}^r(t)$ denota la derivada de $x^r(t)$ con respecto al tiempo t. La velocidad de avance $\upsilon(t)$ y la velocidad angular $\omega(t)$ son las entradas de control del robot. $\delta\upsilon(t)$ y $\delta\omega(t)$ son las diferencias entre el valor de control pretendido y los valores de control reales (ruidos de control) y se supone que son gaussianos de media cero.

Discretizar la Ecuación 1.1 del modelo de movimiento en tiempo continuo con un tiempo de muestreo ΔT y el método de Euler da como resultado

$$x_{k+1}^r = x_k^r + (\upsilon_k + \delta\upsilon_k)\Delta T \cos(\varphi_k^r)$$
$$y_{k+1}^r = y_k^r + (\upsilon_k + \delta\upsilon_k)\Delta T \sin(\varphi_k^r)$$
$$\varphi_{k+1}^r = \varphi_k^r + (\omega_k + \delta\omega_k)\Delta T$$

Ecuación 1.2

donde $(x_k^r, y_k^r, \varphi_k^r)$ es la ubicación del robot en el paso de tiempo k, υ_k es la velocidad en el tiempo k, y ω_k es la velocidad angular en el tiempo k ; $\delta\upsilon_k$ y $\delta\omega_k$ son los ruidos de velocidad de tiempo discreto y los ruidos de velocidad angular, respectivamente.

Modelo de sensor para mapas basados en puntos de referencia

Considerando un entorno que contiene N_0 puntos de referencia en posiciones conocidas $(x_L^i, y_L^i),\ i = 1, ..., N_0$. Para simplificar, se supone que las incertidumbres asociadas con las ubicaciones de los puntos de referencia son cero, aunque es relativamente sencillo ampliar el análisis si este no es el caso. En cada paso de tiempo mientras está en movimiento, el robot observa el rango (distancia) y/o el rumbo (ángulo relativo) a uno o más puntos de referencia. El modelo de observación proporciona un mecanismo para calcular los valores esperados de las observaciones de los sensores, dado el conocimiento del mapa y una estimación de la ubicación del robot. Si el sensor montado en el robot observa tanto el rango como el rumbo hasta el punto de referencia i en el paso de tiempo $k+1$, entonces el modelo de observación viene dado por

$$r_{k+1}^i = \sqrt{(x_L^i - x_{k+1}^r)^2 + (y_L^i - y_{k+1}^r)^2} + \omega_r$$
$$\theta_{k+1}^i = a\tan\left(\frac{y_L^i - y_{k+1}^r}{x_L^i - x_{k+1}^r}\right) - \varphi_{k+1}^r + \omega_\theta$$

Ecuación 1.3

donde ω_r y ω_θ son ruidos de observación gaussianos de media cero.

Los telémetros láser y los sensores ultrasónicos son los sensores más comunes que se utilizan para obtener mediciones de distancia y rumbo a puntos de referencia. En el caso de un sensor que sólo puede observar el rumbo, por ejemplo, una cámara, la ecuación para θ_{k+1}^{i} se convierte en el modelo del sensor. En la Figura 1.5 se ilustra un problema simple de localización de robots con un mapa basado en puntos de referencia.

Modelo de sensor para mapas de cuadrícula de ocupación

Los mapas de cuadrícula de ocupación proporcionan una representación discretizada de un entorno en el que cada una de las celdas de la cuadrícula se clasifica en dos categorías: ocupada o libre. Considérese el escenario en el que un sensor en el robot puede determinar la distancia a la celda de cuadrícula ocupada más cercana a lo largo de una dirección determinada. Un telémetro láser es un sensor de este tipo. Estos sensores consisten en un rayo láser que gira a una velocidad relativamente alta del orden de decenas de revoluciones por segundo y mide la distancia al obstáculo que lo refleja. Si no hay ningún obstáculo dentro del rango del sensor, no se recibe una reflexión y el sensor normalmente informa una distancia máxima nominal $d_{\max}$. Aunque las mediciones de rango obtenidas dependen del entorno y la ubicación del robot, no es factible encontrar un modelo de observación analítica de la forma de la Ecuación 1.3 en este escenario. Sin embargo, dada una estimación de la ubicación del robot y el mapa de cuadrícula, el valor esperado de una medición de rango puede obtenerse numéricamente usando rayos de fundición [7]. Esto hace posible evaluar la probabilidad de una pose determinada, que es suficiente en algunos enfoques de localización, como un filtro de partículas.

La Figura 1.6 muestra un ejemplo simple de un problema de localización de un robot en el que un telémetro láser observa un entorno descrito utilizando una cuadrícula de ocupación. El robot se mueve unos pasos en el entorno. El escaneo proporcionado por el sensor en la primera pose se muestra en rojo.

1.2.2 Filtro de Kalman Extendido

El problema de localización en un mapa basado en puntos de referencia es encontrar la pose del robot en el tiempo $k+1$ como

$$x_{k+1} = (x^r_{k+1}, y^r_{k+1}, \varphi^r_{k+1})^T$$

Ecuación 1.4

dado el mapa, la secuencia de acciones del robot $\upsilon_i, \omega_i (i = 0,...,k)$ y las observaciones del sensor desde el tiempo 1 hasta el tiempo $k+1$.

En su forma más fundamental, el problema es estimar las poses del robot $x_i (i = 0,...,k+1)$ que mejor concuerdan con todas las acciones del robot y todas las observaciones de los sensores. Esto se puede formular como un problema de mínimos cuadrados no lineal utilizando los modelos de movimiento y observación derivados en el apartado que describe el modelo de sensor para mapas basados en puntos de referencia**.** La solución al problema de optimización resultante se puede calcular utilizando un esquema iterativo como Gauss-Newton para obtener la trayectoria del robot y como una consecuencia de la pose actual del robot.

Si los ruidos asociados con las mediciones del sensor se pueden aproximar usando distribuciones gaussianas, y una estimación inicial para la ubicación del robot en el tiempo

0, descrita usando una distribución gaussiana $x_0 \sim N(\hat{x}_0, P_0)$ con $\hat{x}_0$ conocido, P_0 está disponible ($\hat{x}$ se usa para denotar el valor estimado de x), se puede obtener una solución aproximada a este problema de mínimos cuadrados no lineales usando un Filtro de Kalman Extendido (EKF). EKF resume eficazmente todas las medidas obtenidas en el pasado en la estimación de la ubicación actual del robot y su matriz de covarianza. Cuando una nueva observación del sensor está disponible, la estimación de la ubicación actual del robot y su covarianza se actualizan para reflejar la nueva información recopilada. Los pasos esenciales del algoritmo de localización basado en EKF se describen a continuación:

Se denota

$$\mathbf{u}_k = (\upsilon_k, \omega_k)^T$$
$$\mathbf{w}_k = (\delta\upsilon, \delta\omega)^T$$

Ecuación 1.5

Entonces, el modelo de proceso no lineal (desde el tiempo k hasta el tiempo $k+1$) como se indica en la Ecuación 1.2 se puede escribir en forma compacta como

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k)$$

Ecuación 1.6

donde $\mathbf{f}$ es la función de transición del sistema, $\mathbf{u}_k$ es el control y $\mathbf{w}_k$ es el ruido de proceso gaussiano de media cero $\mathbf{w}_k \sim N(0, Q)$.

Considerando el caso general en el que se observa más de un punto de referencia. Representando todas las observaciones $r_{k+1}^i, \theta_{k+1}^i$ juntas como un solo vector $\mathbf{z}_{k+1}$, y todos los

ruidos ω_r, ω_θ juntos como un solo vector $\mathbf{v}_{k+1}$, el modelo de observación en el tiempo $k+1$ como se indica en la Ecuación 1.3 puede también estar escrito en forma compacta como

$$\mathbf{z}_{k+1} = \mathbf{h}(\mathbf{x}_{k+1}) + \mathbf{v}_{k+1}$$

Ecuación 1.7

donde $\mathbf{h}$ es la función de observación obtenida de la Ecuación 1.3 y $\mathbf{v}_{k+1}$ es el ruido de observación gaussiano de media cero $\mathbf{v}_{k+1} \sim N(0, R)$.

Sea la mejor estimación de $\mathbf{x}_k$ en el tiempo k

$$\mathbf{x}_k \sim N(\hat{\mathrm{x}}_k, P_k)$$

Ecuación 1.8

Entonces, el problema de localización se convierte en uno de estimar $\mathbf{x}_{k+1}$ en el tiempo $k+1$:

$$\mathbf{x}_{k+1} \sim N(\hat{\mathrm{x}}_{k+1}, P_{k+1})$$

Ecuación 1.9

donde $\hat{\mathbf{x}}_{k+1}, P_{k+1}$ se actualizan utilizando la información recopilada mediante los sensores. EKF logra esto de la siguiente manera.

Se predice usando el modelo de proceso:

$$\bar{\mathbf{x}}_{k+1} = \mathbf{f}(\hat{\mathbf{x}}_k, \mathbf{u}_k, 0)$$

Ecuación 1.10

$$\bar{P}_{k+1} = J_{\mathrm{f}_x}(\hat{\mathbf{x}}_k, \mathbf{u}_k, 0) P_k J_{\mathrm{f}_x}^T(\hat{\mathbf{x}}_k, \mathbf{u}_k, 0) + J_{\mathrm{f}_\omega}(\hat{\mathbf{x}}_k, \mathbf{u}_k, 0) Q J_{\mathrm{f}_\omega}^T(\hat{\mathbf{x}}_k, \mathbf{u}_k, 0)$$

Ecuación 1.11

donde $J_{\mathrm{f}_x}(\hat{\mathbf{x}}_k, \mathbf{u}_k, 0)$ es el jacobiano de la función $\mathbf{f}$ con respecto a $\mathbf{x}$, $J_{\mathrm{f}_\omega}(\hat{\mathbf{x}}_k, \mathbf{u}_k, 0)$ es el jacobiano de la función $\mathbf{f}$ con respecto a $\mathbf{w}$, ambos evaluados en $(\hat{\mathbf{x}}_k, \mathbf{u}_k, 0)$.

Actualizar usando la observación:

$$\hat{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_{k+1} + K(\mathbf{z}_{k+1} - \mathbf{h}(\bar{\mathbf{x}}_{k+1}))$$

Ecuación 1.12

$$P_{k+1} = \bar{P}_{k+1} - KSK^T$$

Ecuación 1.13

donde la covarianza de innovación S (aquí $\mathbf{z}_{k+1} - \mathbf{h}(\bar{\mathbf{x}}_{k+1})$ se llama innovación) y la ganancia de Kalman K están dadas por

$$S = J_{\mathrm{h}}(\bar{\mathbf{x}}_{k+1}) \bar{P}_{k+1} J_{\mathrm{h}}^T(\bar{\mathbf{x}}_{k+1}) + R$$

Ecuación 1.14

$$K = \bar{P}_{k+1} J_{\mathrm{h}}^T(\bar{\mathbf{x}}_{k+1}) S^{-1}$$

Ecuación 1.15

donde $J_{\mathrm{h}}(\bar{\mathbf{x}}_{k+1})$ es el jacobiano de la función $\mathbf{h}$ con respecto a $\mathbf{x}$ evaluado en $\bar{\mathbf{x}}_{k+1}$.

La aplicación recursiva de las ecuaciones anteriores durante cada instante en que se recopila una nueva observación produce una estimación actualizada de la ubicación actual del robot y su incertidumbre. Esta naturaleza recursiva hace que EKF sea el algoritmo más eficiente computacionalmente disponible para la localización de robots.

Un requisito previo importante para la localización basada en EKF es la capacidad de asociar las mediciones obtenidas con puntos de referencia específicos presentes en el medio ambiente. Los puntos de referencia pueden ser artificiales, por ejemplo, reflectores láser o características geométricas naturales presentes en el entorno, como segmentos de línea, esquinas o planos [8, 9]. En muchos casos, la observación en sí no contiene ninguna información sobre qué punto de referencia en particular se está observando. La asociación de datos es el proceso en el que se toma una decisión en cuanto a la correspondencia entre una observación del sensor y un punto de referencia en particular. La asociación de datos es fundamental para el funcionamiento de un localizador basado en EKF, ya que pueden producirse fallas catastróficas si las decisiones de asociación de datos son incorrectas.

EKF se basa en la aproximación del movimiento no lineal y los modelos de observación mediante ecuaciones lineales de modo que los ruidos del sensor se pueden aproximar mediante distribuciones gaussianas. Estos son supuestos razonables en muchas condiciones prácticas y, por lo tanto, EKF es la opción obvia para resolver el problema de localización del robot cuando el mapa del entorno consta de puntos de referencia claramente identificables.

La Figura 1.7 muestra el resultado de la localización EKF para el problema simple que se muestra en la Figura 1.5. La realidad en el terreno de las poses del robot y las poses estimadas del robot se muestran en rojo y azul, respectivamente. Las elipses de confianza del 95% obtenidas de las matrices de covarianza en el proceso de estimación de EKF también se muestran en la figura.

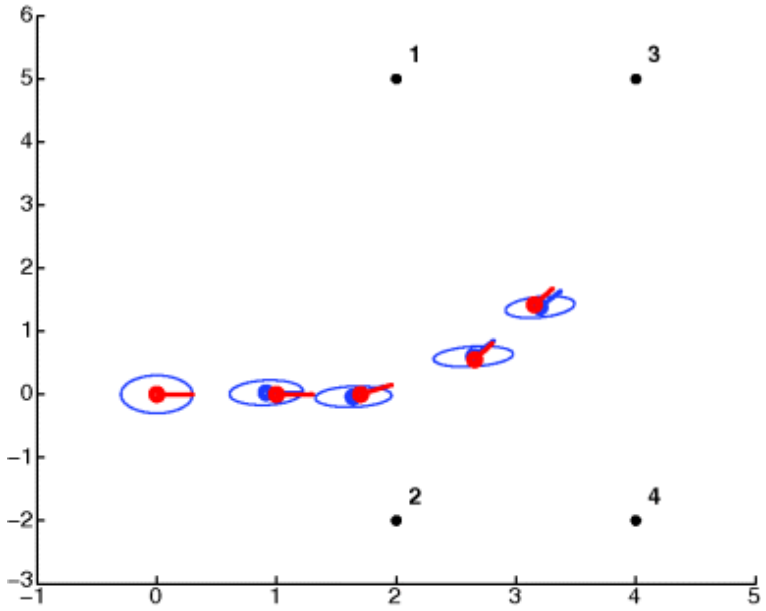

Figura 1.7 Resultado de la localización EKF

1.2.3 Filtro de partículas

Hay dos situaciones importantes en las que EKF no es el método de elección para la localización de robots. La primera es cuando el entorno está representado por una cuadrícula de ocupación. El modelo de sensor para mapas de cuadrícula de ocupación, descrito anteriormente, no es un modelo analítico, sino que se basa en el proceso numérico de proyección de rayos y, como tal, no es adecuado para su uso con un EKF. La otra situación es cuando la ubicación inicial del robot es completamente desconocida, lo que generalmente se conoce como el problema de localización global. En este caso, la ubicación del robot debe describirse utilizando una distribución de probabilidad arbitraria; por lo tanto, se viola la suposición de Gauss que es la base de la formulación EKF. En general, manipular distribuciones de probabilidad arbitrarias es computacionalmente intratable. Una posible estrategia es discretizar el espacio de posibles ubicaciones de los robots y, por lo tanto, tratar con la manipulación de la distribución de probabilidad discreta. Este método se conoce como localización de Markov. La carga de cálculo asociada con la localización de Markov es

proporcional al tamaño del entorno y la resolución de la discretización, lo que hace que esta estrategia sea inadecuada en muchas situaciones.

En la localización del filtro de partículas (también conocida como localización de Monte Carlo) [10], en lugar de discretizar el espacio de las ubicaciones de los robots, se utiliza un conjunto ponderado de estimaciones de la ubicación del robot, denominado partículas, para describir la distribución de probabilidad de la ubicación del robot. A medida que los cálculos se centran en las partículas y se colocan más partículas en ubicaciones de robot más probables, los filtros de partículas proporcionan una alternativa más eficiente a la localización de Markov. El número de partículas utilizadas determina la precisión de la representación. Sin embargo, aumentar el número de partículas para obtener una mayor precisión conduce a un proceso de estimación más costoso.

En el filtro de partículas, cada partícula en efecto proporciona una suposición sobre la ubicación del robot. Por lo tanto, cada partícula está representada por tres variables (x, y (posición), φ (orientación)) para un robot que opera en un plano bidimensional. Cada partícula i tiene un peso ω_i que indica la contribución de esa partícula a la distribución de probabilidad. La suma de los pesos de todas las partículas se establece en 1, es decir, $\sum_{i=1}^{n} \omega_i = 1$, donde n es el número total de partículas utilizadas. Una colección de tales conjeturas describe el mejor conocimiento disponible, generalmente denominado creencia, en cuanto a la verdadera ubicación del robot. En el caso de la localización global, la ubicación inicial del robot es completamente desconocida; por lo tanto, es igualmente probable que todas las ubicaciones del entorno contengan al robot. Por tanto, se utiliza un conjunto de partículas igualmente ponderadas distribuidas uniformemente en el entorno para representar

la creencia de la ubicación del robot. Durante el proceso de localización, esta creencia se actualiza a medida que se adquiere más y más información de los sensores.

En el filtro de partículas, cada vez que se recopila información de los sensores, se actualiza la creencia actual. El proceso es el siguiente [11, 12]:

1. **Predicción:** Cuando se le ordena al robot que se mueva, la nueva creencia se obtiene moviendo cada partícula de acuerdo con la Ecuación 1.2 del modelo de movimiento con $\delta\upsilon_k, \delta\omega_k$ generados aleatoriamente.
2. **Actualización:** Cuando se recibe una nueva observación de sensor, la creencia se actualiza utilizando un modelo de observación. En este paso, los pesos de las partículas se cambian para reflejar la probabilidad de que la verdadera ubicación del robot coincida con la partícula correspondiente. En el caso de la j-ésima observación de un telémetro láser, se utiliza la proyección de rayos de cada partícula para obtener una medida esperada $\hat{d}_j$. Si la medición real viene dada por d_j y si se supone que el ruido del sensor es una media cero con una varianza σ_d^2, la probabilidad se puede calcular usando una distribución gaussiana basada en

$$\frac{1}{\sigma_d\sqrt{2\pi}}\exp\{-\frac{(\hat{d}_j - d_j)^2}{2\sigma_d^2}\}$$

Ecuación 1.16

Como en un caso dado se adquieren múltiples observaciones de rango independientes del sensor, la probabilidad de obtener una secuencia de observaciones se calcula multiplicando todas las probabilidades. Una vez calculadas las probabilidades de

todas las partículas, estas se normalizan para obtener el peso de cada una de las partículas.

3. **Remuestreo:** Se realiza para evitar la situación en la que un pequeño número de partículas con grandes pesos dominan la representación de la creencia. Una estrategia común utilizada para el remuestreo [13] es la siguiente:
    a. Calcular una estimación del número efectivo de partículas como

$$n_{\text{eff}} = \frac{1}{\sum_{i=1}^{n} \omega_i^2}$$

Ecuación 1.17

    b. Si n_{eff} es menor que un umbral, extraiga n partículas del conjunto de partículas actual con probabilidades proporcionales a sus pesos. Reemplace el conjunto de partículas actual con este nuevo. Establezca los pesos de cada partícula en $1/n$.
4. El conjunto de partículas resultante representa la creencia actualizada de la ubicación del robot.

Este proceso se repite a medida que se toman nuevas acciones de control y se encuentran disponibles nuevas observaciones. Se puede utilizar la media o la moda de la distribución de probabilidad correspondiente si se desea un valor numérico para la mejor estimación de la ubicación del robot.

La Figura 1.8 muestra el resultado de la localización del filtro de partículas para el problema simple dado en la Figura 1.6. Las partículas en el último paso se muestran en verde, la verdad básica de la última pose del robot está en rojo, la mejor estimación de la última pose del robot está en azul.

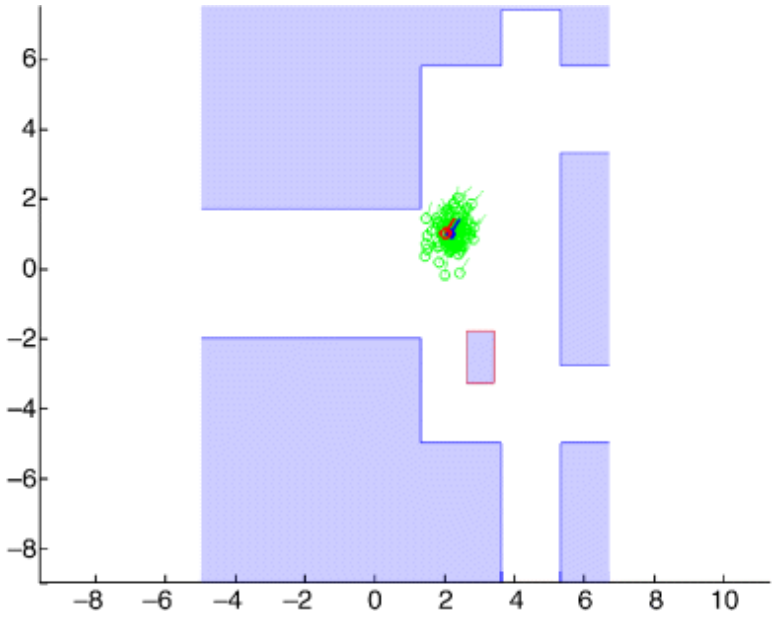

Figura 1.8 Resultado de la localización del filtro de partículas

1.2.4 Técnicas de localización alternativas

Cuando no se dispone de un mapa del entorno, el problema de localización del robot se vuelve significativamente más desafiante. En este caso, el problema se convierte en estimar simultáneamente la ubicación del robot y las ubicaciones de los puntos de referencia. En los últimos 15 años, han surgido métodos robustos y eficientes para lidiar con la localización de robots en entornos desconocidos, también conocido como el problema de localización y mapeo simultáneos (SLAM) [6]. SLAM en un entorno basado en características ha sido bien estudiado y se ha demostrado que tanto la optimización de EKF como la de mínimos cuadrados podrían usarse para resolver SLAM de manera confiable en la mayoría de los casos [14, 15]. La presencia de solucionadores eficientes para problemas de optimización de mínimos cuadrados no lineales a gran escala [16] ha dado como resultado soluciones en tiempo real para la estimación de la ubicación del robot en entornos desconocidos. Debido a su bajo costo y la presencia de algoritmos que extraen información

rica, las cámaras se han convertido en un sensor importante en muchas aplicaciones de navegación de robots.

El problema de estimar la ubicación del robot dado un mapa ahora se considera un problema resuelto, aunque los entornos altamente dinámicos poblados de personas plantean desafíos importantes en las implementaciones prácticas. El problema más complejo de estimar continuamente la ubicación de un robot dentro de un entorno desconocido durante un largo período sigue siendo objeto de mucha investigación.

1.3 Planificación de movimiento

La planificación de movimiento [18] implica hacer que un robot determine automáticamente cómo moverse mientras evita colisiones con obstáculos. La planificación es un aspecto obvio de la navegación que responde a la pregunta *¿Cuál es el mejor camino para llegar ahí?*

Su formulación original, llamada *The Piano Mover's Problem,* piensa en determinar cómo mover un mueble complicado a través de una casa desordenada. Este problema básico de planificación de movimiento es definido como sigue.

1. Un *espacio de trabajo* W, donde $W = \mathbb{R}^2$ o $W = \mathbb{R}^3$.
2. Una *región de obstáculos* $O \subset W$.
3. Un *robot* definido en W. Ya sea un cuerpo rígido A o una colección de m enlaces: $A_1, A_2, ..., A_m$.
4. El *espacio de configuración* C (C_{obs} y C_{free} son definidas en consecuencia).

5. Una *configuración inicial* $q_I \in C_{free}$.

6. Una *configuración meta* $q_G \in C_{free}$. La configuración inicial y meta son comúnmente llamadas como *query* (q_I, q_G).

Calcular un camino (*continuo*), $T:[0,1] \to C_{free}$ de tal manera que $T(0) = q_I$ y $T(1) = q_G$.

Hay varias cuestiones que deben tenerse en cuenta en la planificación de la ruta de los robots móviles debido a los diversos propósitos y funciones del propio robot. El grado de dificultad de la planificación de movimiento en robots varía mucho dependiendo de un par de factores: si toda la información sobre los obstáculos (por ejemplo, tamaños, ubicaciones, movimientos, etc.) se conoce antes de que el robot se mueva y si estos obstáculos se mueven o permanecen en su lugar a medida que el robot avanza.

1.3.1 Categorías

Los diferentes problemas relacionados a la planificación de movimiento de robots móviles se pueden dividir en tres categorías de acuerdo al conocimiento que tiene el robot sobre su entorno, la naturaleza del mismo y el enfoque utilizado para resolver el problema, La Figura 1.9 denota lo antes mencionado.

Naturaleza del entorno: El problema de la planificación de movimiento se puede realizar tanto en entornos estáticos como dinámicos: un entorno estático no varía, las posiciones de origen y destino son fijas y los obstáculos no varían de ubicación con el tiempo. Sin embargo, en un entorno dinámico, la ubicación de los obstáculos y la posición de la meta pueden variar durante el proceso de búsqueda. Normalmente, la planificación de movimiento en entornos dinámicos es más compleja que en entornos estáticos debido a la incertidumbre del entorno.

Como tal, los algoritmos deben adaptarse a cualquier cambio inesperado, como la llegada de nuevos obstáculos en movimiento en la ruta planificada previamente o cuando el objetivo se

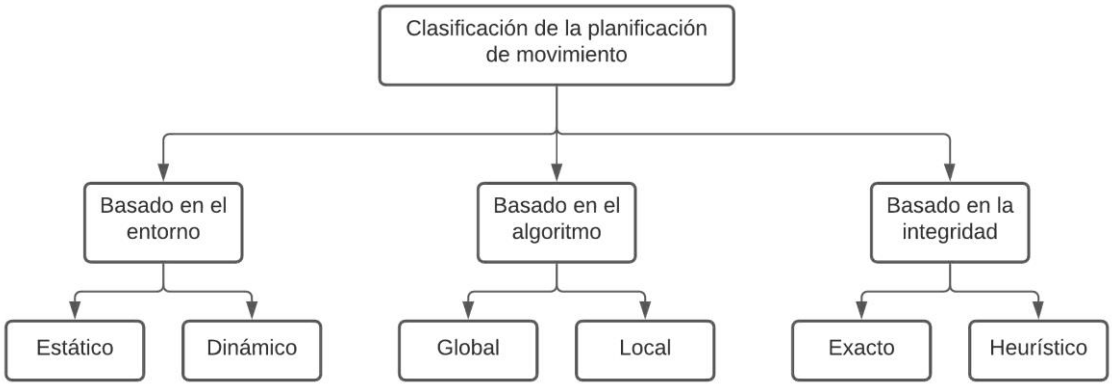

Figura 1.9 Clasificación de la planificación de movimiento

está moviendo continuamente. Cuando tanto los obstáculos como los objetivos se mueven, el problema de planificación se vuelve aún más crítico, ya que debe reaccionar de manera efectiva en tiempo real tanto a los movimientos de los objetivos como de los obstáculos. Los enfoques de planificación aplicados en entornos estáticos no son apropiados para el problema dinámico.

Conocimiento del mapa: La planificación de la ruta de los robots móviles se basa básicamente en un mapa existente como referencia para identificar la ubicación inicial y objetivo y el vínculo entre ellos. La cantidad de conocimiento del mapa juega un papel importante para el diseño del algoritmo de planificación. Según el conocimiento del robot sobre el entorno, la planificación de la ruta se puede dividir en dos clases: en la primera clase, el robot tiene un conocimiento a priori sobre el entorno modelado como un mapa. Esta categoría se conoce como *planificación global* o *planificación deliberativa*. La segunda clase de planificación asume que el robot no tiene información a priori sobre su entorno (es decir,

entorno incierto). En consecuencia, debe detectar la ubicación de los obstáculos y construir un mapa estimado del entorno en tiempo real durante el proceso de búsqueda para evitar obstáculos y adquirir un camino adecuado hacia el estado de meta. Este tipo de planificación de ruta se conoce como *planificación local* o *navegación reactiva*. La Tabla 1.1 muestra las diferencias entre estas dos clases.

Integridad: Dependiendo de su integridad, el algoritmo de planificación de rutas se clasifica como exacto o heurístico. Un algoritmo exacto encuentra una solución óptima si existe o prueba que no existe una solución factible. Los algoritmos heurísticos buscan una solución de buena calidad en menos tiempo.

En este capítulo se abarcaron los temas más importantes relacionados al problema de la navegación, necesarios para comprenderlo de manera general, así como de forma individual según las tareas involucradas: ya sea localizar el robot o planificar una ruta, estas actividades en conjunto componen la solución al problema de la navegación. En el capítulo 2 se introducen los algoritmos de planificación más comunes como parte de la literatura.

Tabla 1.1 Diferencias entre la planificación local y la planificación global

Planificación local	Planificación global
Basado en sensores	Basado en mapas
Navegación reactiva	Navegación deliberativa
Respuesta rápida	Respuesta relativamente lenta
Supone que el espacio de trabajo está incompleto o parcialmente incompleto	El espacio de trabajo es conocido
Genera el camino y avanza hacia el objetivo mientras evitas obstáculos u objetos	Genera una ruta factible antes de avanzar hacia la posición de la meta
Realizado en línea (online)	Realizado sin conexión (offline)

Capítulo 2

2 Algoritmos de planificación

A lo largo de la historia se han propuesto diversos algoritmos para la planificación de movimiento, algunos con mejor desempeño que otros. En realidad, la cantidad de algoritmos citados en la literatura es extenso y puede ser confuso para personas que recién incursionan en el mundo de la robótica. En este capítulo se revisarán algunos de los algoritmos de planificación comúnmente usados en vehículos autónomos y robótica, esto busca servir como punto de partida al campo de la autonomía que se encuentra en rápida evolución. Cabe aclarar que los siguientes algoritmos son representativos y no cubren la totalidad de las técnicas y algoritmos destinados a la planificación.

2.1 Algoritmo Dijkstra

El algoritmo de Dijkstra es una forma bastante genérica de encontrar la ruta más corta entre dos vértices que están conectados por aristas. Dijkstra funciona resolviendo subproblemas, calculando el camino más corto desde la arista inicial a los vértices y entre los vértices más cercanos a la fuente [20]. Encuentra el siguiente vértice más cercano manteniendo los nuevos vértices en una cola de prioridad mínima y almacena sólo un nodo intermedio para que únicamente se pueda encontrar una ruta más corta.

Dijkstra encuentra el camino más corto en un entorno acíclico y puede calcular el camino más corto desde el punto de partida hasta los demás. El algoritmo tradicional de Dijkstra se basa en una estrategia codiciosa para la planificación de rutas. Se utiliza para encontrar el camino más corto en un grafo. Busca la solución del camino más corto sin una atención formal al pragmatismo de la misma.

El algoritmo de Dijkstra modificado tiene como objetivo encontrar rutas alternativas en situaciones donde los costos de generar las rutas más cortas plausibles son significativos. Este algoritmo modificado introduce otro componente del algoritmo clásico en forma de probabilidades que definen el estado de libertad a lo largo de cada borde del grafo [21]. Esta técnica ayuda a superar las deficiencias computacionales del algoritmo de referencia, lo que respalda su uso en aplicaciones novedosas.

Otro algoritmo mejorado de Dijkstra reserva todos los nodos con la misma distancia tanto el nodo de origen como los nodos intermedios, y luego busca de nuevo desde todos los nodos intermedios hasta atravesar con éxito y alcanzar el nodo de destino.

El algoritmo de Dijkstra no puede almacenar datos aparte de los nodos que se han atravesado previamente. Para superar esta desventaja, se introduce un esquema de almacenamiento que implementa un diccionario multicapa [22] para el almacenamiento de datos, que consta de dos diccionarios y una lista de estructura de datos organizada en orden jerárquico. El primer diccionario asigna todos y cada uno de los nodos a sus nodos vecinos. El segundo diccionario almacena la información de la ruta de cada ruta vecina [22].

El algoritmo de Floyd es un algoritmo de grafos popular para encontrar la ruta más corta en un grafo ponderado positivo o negativo [23], mientras que Dijkstra funciona mejor para encontrar el camino más corto en un grafo ponderado positivo.

Dijkstra es un algoritmo confiable para la planificación de rutas. También tiene mucha memoria ya que tiene que calcular todos los resultados posibles para determinar la ruta más corta, y no puede manejar bordes negativos. Su complejidad de cálculo es $O(n2)$ siendo n el número de nodos en la red [24]. Debido a sus limitaciones, surgieron muchas variantes mejoradas basadas en sus aplicaciones.

En general, el algoritmo de Dijkstra es más adecuado para un entorno estático y/o planificación de ruta global, ya que la mayoría de los datos necesarios están predefinidos para el cálculo de la ruta más corta; sin embargo, hay aplicaciones en las que se ha utilizado el algoritmo de Dijkstra para entornos dinámicos. La Tabla 2.1 muestra la clasificación para el algoritmo Dijstra y sus variantes.

Tabla 2.1 Idoneidad del algoritmo de Dijkstra y sus variantes según la clasificación basada en restricciones del entorno estáticas y dinámicas.

Algoritmo	Restricciones estáticas	Restricciones dinámicas
Dijkstra	x	
Dijkstra mejorado	x	
Diccionario multicapas	x	x
Floyd y Dijkstra	x	

2.2 Algoritmo A*

A* es un algoritmo popular de planificación de rutas de recorrido de grafos. A* funciona de manera similar al algoritmo Dijkstra, excepto que guía su búsqueda hacia los

estados más prometedores, lo que puede ahorrar una cantidad significativa de tiempo de cálculo [25]. A* es el más utilizado para abordar una solución casi óptima con el conjunto de datos/nodos disponibles.

Es muy utilizado en entornos estáticos, sin embargo, hay casos en los que este algoritmo se utiliza en entornos dinámicos. La función básica se puede adaptar a una aplicación o entorno específicos en función de las necesidades específicas. A* es similar a Dijkstra en que funciona basándose en el árbol de ruta de menor costo desde el punto inicial hasta el punto objetivo final. El algoritmo base usa la ruta menos costosa y la expande usando la función que se muestra a continuación

$$f(n) = g(n) + h(n)$$

Ecuación 2.1

donde $g(n)$ es el costo real desde el nodo n hasta el nodo inicial, y $h(n)$ es el costo de la ruta óptima desde el nodo objetivo hasta n.

Es más simple y menos pesado computacionalmente que muchos otros algoritmos de planificación de rutas, y su eficiencia se presta a la operación en sistemas restringidos e integrados. El algoritmo A* es un algoritmo heurístico que utiliza información heurística para encontrar la ruta óptima. Necesita buscar nodos en el mapa y establecer funciones heurísticas apropiadas para orientación, como la distancia euclidiana, la distancia de Manhattan o la distancia diagonal.

Existe una compensación entre la velocidad y la precisión cuando se utiliza el algoritmo A*. Se puede disminuir la complejidad temporal del algoritmo a cambio de mayor

memoria, o consumir menos memoria a cambio de ejecuciones más lentas. En ambos casos, se encuentra el camino más corto.

En el algoritmo jerárquico A*, la búsqueda de ruta se divide en pocos procesos, se encuentra la solución óptima para cada proceso y luego se obtiene la solución óptima global, que es la ruta más corta al lugar vacío. Se realiza otra mejora en este algoritmo, con el algoritmo jerárquico A* mejorado que usa el tiempo óptimo como índice de evaluación e, introduciendo una variable, se usa una suma ponderada junto con la heurística [26]:

$$f_w(n) = (1-w)g(n) + wh(n)$$

Ecuación 2.2

donde w es un coeficiente ponderado.

El coeficiente ponderado se define con respecto a cada aplicación y, en la mayoría de los casos, podría usarse para representar la distancia lineal (por ejemplo, la distancia euclidiana). Se afirma que la A* jerárquico mejorado tiene una eficiencia y precisión mejoradas [27]. Otra variación de este algoritmo se utiliza en el ***valet parking*** donde el vehículo tiene una dirección de movimiento que depende de la velocidad, la marcha, el ángulo de dirección y otros parámetros de la cinemática del vehículo.

Este algoritmo es llamado A* híbrido. El algoritmo A* híbrido mejora el algoritmo A* normal cuando se implementa en un robot no holonómico. Este algoritmo utiliza la cinemática de un vehículo para predecir el movimiento del vehículo dependiendo de la velocidad, marcha y ángulo de dirección, y otros parámetros del vehículo, estos agregarán costos a la función heurística. Se introdujo una versión mejorada de A* híbrido en la misma aplicación con la ayuda de un planificador de rutas de diagramas de visibilidad.

El diagrama de visibilidad fue uno de los primeros algoritmos de búsqueda basados en grafos utilizados en la planificación de rutas en robótica. Este método garantiza encontrar el camino más corto desde el inicio hasta la posición final. Las posiciones de inicio, final y obstáculo se introducen como entrada al algoritmo A* híbrido donde A* se ejecuta en los resultados del diagrama de visibilidad, que luego proporciona la distancia óptima. Esto se denomina algoritmo A* híbrido guiado [28]. Algunas funciones heurísticas comunes se muestran en la Tabla 3.

El algoritmo A* es computacionalmente simple en relación con otros algoritmos de planificación de rutas. Con ajustes para la cinemática del vehículo, el ángulo de dirección y el tamaño del vehículo, A* es adecuado para aplicaciones automotrices. Se pueden crear nuevas funciones de costo con respecto al muestreo de pasos iguales, como en el caso de la Ecuación 2.2. La función de costo que se muestra incluye los costos de distancia y de

Tabla 2.2 Tipos más comunes de funciones heurísticas usadas en los algoritmos de planificación de caminos

Función	Ecuación
Distancia Euclidiana	$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
Distancia de Manhattan	$\lvert(x_1 - x_2)\rvert + \lvert(y_1 - y_2)\rvert$
Distancia de Octil	$\max \lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert$

dirección, penalizando cada paso utilizando el costo en cada movimiento. Las consideraciones anteriores evitan giros bruscos y de esta manera mejoran la suavidad del camino [29].

$$f(n) = K_1 g(n) + K_2 h(n) + K_3 p(n)$$

Ecuación 2.3

donde K_1, K_2, K_3 son pesos con valor positivo, $g(n)$ y $h(n)$ son los mismos que la función base A*, y $p(n)$ es el factor de penalización basado en el costo de giro.

*Lifelong Planning A** (LPA*) [30] combina las mejores técnicas de métodos de búsqueda tanto incrementales como heurísticos. Promete encontrar los caminos más cortos mientras los costos de borde de un grafo cambian o los vértices se agregan o eliminan debido a la adición o eliminación de obstáculos. El algoritmo inicializa los valores g (distancias iniciales) de los vértices encontrados hasta el infinito y sus valores rhs (valores anticipados de un paso) de acuerdo con la Ecuación 2.4 y realiza una búsqueda A* al principio.

En las etapas posteriores, actualiza los valores rhs y las claves, Ecuación 2.5, de los vértices afectados por los costos de borde modificados, así mismo decide su membresía en función de la consistencia local en la cola de prioridad y, en última instancia, vuelve a calcular el camino más corto desde el principio hasta el objetivo. Sin embargo, cuando se usa el algoritmo LPA*, se debe volver a planificar la ruta desde el punto de partida hasta el punto de destino [31].

Así, este algoritmo es más rápido que los dos métodos de búsqueda individuales porque reutiliza partes del árbol de búsqueda anterior que son idénticas al árbol nuevo y aplica conocimiento heurístico para enfocar la búsqueda.

$$rhs(s) = \begin{cases} 0 & \text{si } s = s_{start} \\ \min_{s' \in pred(s)} (g(s') + c(s, s')) & \text{de otro modo} \end{cases}$$

Ecuación 2.4

$$K(s) = \left[\min(g(s), rhs(s)) + h(s); \min(g(s), rhs(s))\right]$$

Ecuación 2.5

Para resumir, la Tabla 2.3 muestra una lista de algoritmos A* y sus variantes utilizadas en la planificación de rutas en contextos estáticos y dinámicos.

Tabla 2.3 Clasificación del algoritmo A y sus variantes basada en restricciones del entorno estáticas y dinámicas*

Algoritmo	Restricciones estáticas	Restricciones dinámicas
A*	x	
A* jerárquico	x	
A* jerárquico mejorado	x	
A* híbrido		x
A* híbrido guiado		x
A* con muestreo por pasos iguales	x	
A* diagonal	x	
A* con heurísticas inteligentes		x
Lifelong Planning A*		x

Es importante destacar que el algoritmo A* es computacionalmente eficiente. Esto lo hace adecuado para aplicaciones implementadas en entornos estáticos. La velocidad y la eficiencia computacionales de A* y sus variantes dependen de la precisión de la función heurística.

2.3 Algoritmo D*

La planificación de caminos en entornos dinámicos y parcialmente conocidos de una manera eficiente es cada vez más crítica, por ejemplo, para vehículos automatizados. Para

resolver este problema, se utiliza el algoritmo D* (o Dynamic A*) para generar una trayectoria libre de colisiones en medio de obstáculos en movimiento. D* es un algoritmo de búsqueda incremental informado que repara parcialmente el mapa de costos y el mapa de costos calculado previamente.

El algoritmo D* procesa el estado de un robot hasta que es eliminado de la lista abierta y, al mismo tiempo, la secuencia de estados es calculada junto con los indicadores hacia atrás para dirigir al robot a la posición objetivo o actualizar el costo debido a un obstáculo detectado y colocar los estados afectados en la lista abierta. Los estados en la lista abierta se procesan hasta que el costo de la ruta desde el estado actual al objetivo es menor que un umbral mínimo, los cambios de costo se propagan al siguiente estado y el robot continúa siguiendo los indicadores hacia atrás en la nueva secuencia hacia el objetivo [32]. D* es más de 200 veces más rápido que un re-planificador óptimo.

El algoritmo D* Lite [33] se basa en LPA* intercambiando el vértice inicial y final e invirtiendo todos los bordes algorítmicamente. El algoritmo encuentra la ruta más corta desde el nodo objetivo al nodo de inicio en un entorno dinámico desconocido minimizando los valores de rhs calculados mediante la Ecuación 2.4. Los valores clave de los vértices se calculan y actualizan mediante la Ecuación 2.5 cuando una conexión cambia con la variación en los pesos de sus bordes de conexión.

Dado que el valor heurístico disminuye en un valor, se agrega el mismo valor a todas las nuevas claves calculadas. Este método evita atravesar la cola de prioridad cada vez que cambian las conexiones. Por lo tanto, el algoritmo D* Lite tiene un costo computacional más bajo en comparación con LPA* ya que evita el reordenamiento de la cola de prioridad. Este algoritmo es adecuado para la planificación de rutas de vehículos autónomos en entornos

abarrotados, donde el algoritmo puede alcanzar un resultado de reprogramación rápido cuando se encuentran obstáculos inesperados [34].

El algoritmo D* Lite mejorado [35] es una mejora del algoritmo D* Lite, de hecho, mantiene el mismo principio de búsqueda de una ruta, pero supera sus problemas a través de mejoras que enfatizan la evasión de obstáculos complicados, previniendo que el robot atraviese entre dos obstáculos y a través de las esquinas de los mismos, creando paredes virtuales de ser preciso y eliminando caminos innecesarios que producen el camino más corto desde la posición objetivo hasta la posición inicial.

La idea de utilizar la interpolación para producir mejores funciones de valor para muestras discretas en un espacio de estado continuo no es nueva. Este enfoque se ha utilizado en la programación dinámica para calcular el valor de los sucesores que no están en el conjunto de muestras [36, 37].

El algoritmo D* Field [38] es una extensión de la familia de algoritmos D* ampliamente utilizada que emplea la interpolación lineal para producir rutas globalmente suaves y de bajo costo.

El algoritmo D* y sus variantes se pueden emplear para cualquier problema de optimización de costos, donde el costo de la ruta cambia durante la búsqueda de la ruta óptima hacia la meta. D* es más eficiente cuando estos cambios se detectan más cerca del nodo actual en el espacio de búsquedas. La Tabla 2.4 diferencia a este algoritmo y sus variantes.

Tabla 2.4 Resumen del algoritmo D y sus variantes*

Algoritmo	Restricciones estáticas	Restricciones dinámicas
D*		x
D* Lite		x
D* Lite mejorado		x
D* Field		x

2.4 Árboles de exploración rápida

Los algoritmos mencionados hasta antes de este punto, son algoritmos de naturaleza estática y que requieren de una ruta especificada por adelantado. En contraste, los algoritmos RTT por sus siglas en inglés, Rapidly-Exploring Random Trees, son algoritmos dinámicos y en línea que no requieren de esta característica. Más bien, se expanden en todas las regiones y, según los pesos asignados a cada nodo, crean una ruta desde el principio hasta el objetivo. Los RRT se introdujeron para manejar amplias clases de problemas de planificación de rutas. Fueron diseñados específicamente para manejar restricciones no holonómicas (restricciones que no son integrables en restricciones posicionales).

Los RRT y los mapas de ruta probabilísticos (PRM) comparten las mismas propiedades deseables, y ambos fueron diseñados con pocas heurísticas y parámetros arbitrarios. Esto da un mejor rendimiento y consistencia en los resultados. Los PRM pueden requerir conexiones de miles de configuraciones o estados para encontrar una solución, mientras que los RRT no requieren que se establezcan conexiones entre estados para encontrar una solución. Esto ayuda a aplicar RRT a no holonómicos y planificación *kynodinamic* [39].

Los RRT se expanden rápidamente en el espacio, crecen desde el punto de partida y se expanden hasta que el árbol está lo suficientemente cerca del punto objetivo. En cada iteración, el árbol se expande hasta el vértice más cercano del vértice generado aleatoriamente. Este vértice más cercano se selecciona en términos de una métrica de distancia. Puede ser euclidiana, Manhattan o cualquier otra métrica de distancia.

RRT se expande fuertemente en porciones inexploradas del espacio de configuración del robot en comparación con árboles aleatorios ingenuos (naive), que tienden a expandirse fuertemente en lugares que ya están explorados, ver Figuras 2.1 y 2.2. Por lo tanto, se induce que RRT está sesgado hacia regiones inexploradas. Los vértices de RRT siguen una distribución uniforme. El algoritmo es relativamente simple y los RRT siempre permanecen conectados, aunque el número de bordes es mínimo.

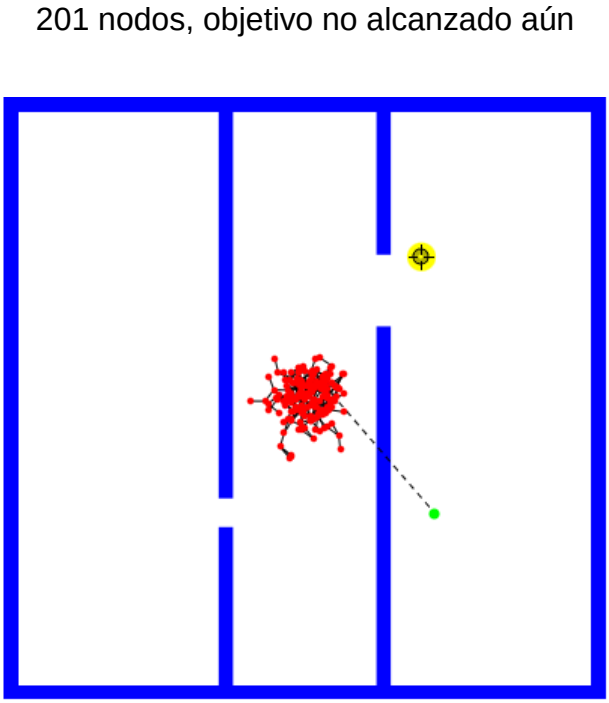

Figura 2.1 Exploración de un árbol aleatorio con 201 nodos

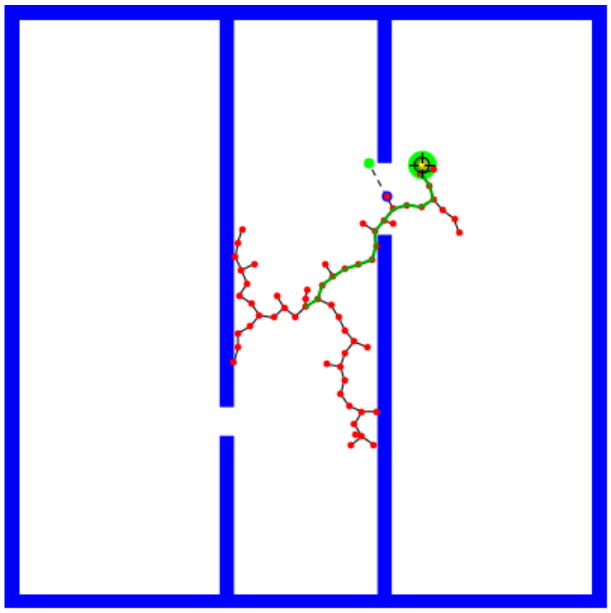

Figura 2.2 Exploración de un árbol aleatorio con 61 nodos

Dado que los algoritmos RRT pueden hacer frente a restricciones no holonómicas, se pueden aplicar a casi cualquier sistema de ruedas. Dependiendo de las necesidades del usuario, se puede escoger una variante de RRT [41].

RRT Connect o RRT bidireccional combina dos RRT, uno en la posición inicial y un segundo en la posición objetivo, conectándolos mediante una heurística. Este enfoque es adecuado para problemas que no involucran restricciones diferenciales. Durante cada iteración, un árbol se extiende y el nuevo vértice se conecta al vértice más cercano del otro árbol. Luego, el papel de cada árbol se invierte y ambos árboles exploran el espacio de configuración libre. Este algoritmo es adecuado para planificar los movimientos de un brazo robótico con altos grados de libertad [42].

En los árboles aleatorios basados en sensores (SRT), se crea un *road map* del área explorada con una región segura (SR) asociada [43, 44]. Los sensores detectan el área segura

local (LSR). Cada nodo del SRT consta de una configuración libre con una región segura local asociada. La región segura consta de todas las regiones seguras locales. Es una estimación del espacio libre que rodea al robot en una configuración determinada. La forma del LSR depende de las características del sensor (por ejemplo, la resolución angular) del robot. La forma LSR puede ser un circulo o una estrella.

El árbol se expande hacia direcciones seleccionadas aleatoriamente de modo que la nueva configuración y la ruta que llega al robot de prueba estén contenidas en la región local segura del nodo. Durante cada iteración, los datos del sensor se recopilan y analizan para generar una región plausible que estima el espacio libre que rodea al robot en la configuración actual. Luego, se agrega al árbol un nuevo nodo que contiene la configuración actual y el LSR. Dependiendo de los sensores del robot, se pueden utilizar diferentes técnicas de percepción. La estrategia de exploración LSR en forma de estrella es más precisa, como se ha demostrado experimentalmente [43].

RRT* extiende RRT para encontrar la ruta óptima desde un inicio hasta un nodo objetivo utilizando la igualdad de triángulos, ver Figura 2.3. A medida que aumenta el número de nodos, se descubren rutas de menor costo (más óptimas). RRT Connect es particularmente útil para brazos robóticos. La heurística de conexión funciona con mayor eficacia cuando se pueden esperar espacios relativamente abiertos para la mayoría de las consultas de planificación. La heurística de conexión se desarrolló originalmente con este tipo de problema en mente [45]. La Tabla 2.5 muestra clasifica el algoritmo RRT y sus variantes.

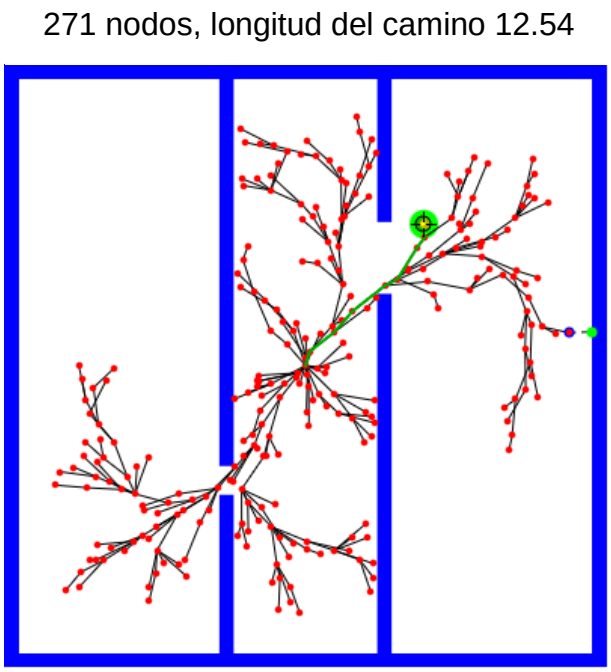

Figura 2.3 Exploración de RRT con 271 nodos*

Tabla 2.5 Clasificación del algoritmo RRT y sus variantes en base a restricciones estáticas y dinámicas

Algoritmo	Restricciones estáticas	Restricciones dinámicas
RRT		x
RRT Connect		x
SRT		x
RRT*		x

Como ya se mencionó, la lista de algoritmos existentes para la planificación de caminos es extensa y no se limita a la descrita en este capítulo. Sin embargo, considerando la naturaleza de este trabajo de tesis es importante fundamentar las bases de al menos los algoritmos descritos arriba.

Capítulo 3

3 Sistema Operativo Robótico

ROS, por sus siglas Robotic Operating System, es un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo [47]. Proporciona servicios estándar como la abstracción del hardware, el control de dispositivos de bajo nivel, la implementación de funcionalidad de uso común, el paso de mensajes entre procesos y el mantenimiento de paquetes. Además, es de código abierto lo que permite realizar colaboraciones de mejora a través de su comunidad, la cual cuenta con poco más de 68,000 usuarios registrados en su página oficial ROS Answers.

Este framework pone a disposición una serie de herramientas y librerías para obtener, construir, escribir, y ejecutar código a través de múltiples computadoras. Permite crear código que después será usado en uno o varios robots, las limitaciones dependen del tipo de robot para el cual se escriben los programas, en especial las relacionadas a las características físicas del mismo. Cabe mencionar que ROS permite integrar el código generado tanto en robots del mundo real como en robots virtuales a través de un sistema de simulación.

Las principales ventajas de utilizar el conjunto de herramientas y librerías que proporciona ROS se puede resumir en las siguientes:

- Mecanismo de comunicación entre programas

Es un estándar que permite la comunicación entre diversos programas de un mismo sistema, ya sea en una computadora o en varias.

- Reusabilidad de código

  Los paquetes estándar proporcionados en las distribuciones de ROS implementan muchos de los algoritmos comúnmente usado en robótica que ya han sido depurados y usados de forma estable. Es decir, no es necesario reinventar la rueda, en muchos de los casos ROS ya cuenta con un paquete que proporciona la o las funcionalidades que se buscan para un robot.

- Testeado rápido

  Puesto que ROS implementa un diseño de comunicación por paso de mensajes, se pueden realizar simulaciones para aislar la funcionalidad del sistema y así crear conjuntos de prueba.

3.1 Conceptos

Nodo

Un nodo es un proceso que realiza algún tipo de computación en el sistema. Los nodos se combinan dentro de un grafo, compartiendo información entre ellos, para crear ejecuciones complejas. Un nodo puede controlar un sensor láser, otro los motores de un robot y otro la construcción de mapas. Los nodos pueden utilizar o proporcionar algún servicio.

Tema

Un tema es un canal que actúa como una tubería, donde otros nodos ROS pueden publicar o leer información, de esta forma se brinda un sistema de comunicación a través de

distintas clases de mensajes entre los diversos nodos que se encuentren ejecutando en una computadora.

Mensaje

Tipo de dato de ROS que es utilizando durante la suscripción y publicación de un tema.

Paquete

Los paquetes son el principal sistema de organización de los programas ROS, como tal, todo programa ROS que se quiera ejecutar debe estar organizado dentro de un paquete. Un paquete puede contener un nodo, una librería, conjunto de datos, o cualquier cosa que pueda constituir un módulo.

Pila

Una pila es un conjunto de nodos que proporciona alguna funcionalidad.

Master

El ROS Master proporciona servicios de nomenclatura y registro al resto de nodos del sistema ROS. Realiza un seguimiento de los publicadores y subscritores de temas y servicios. El papel de Master es permitir que los nodos ROS individuales se ubiquen entre sí. Una vez que estos nodos se han localizado, se comunican de igual a igual.

ROS Core

Es una colección de nodos y programas que son requisitos previos de un sistema basado en ROS. Antes de esperar que algo funcione dentro de ROS se debe ejecutar el comando roscore para iniciar el Master, la salida de mensajes y el servidor de parámetros.

3.2 Rviz

Es una herramienta de visualización 3D para ROS. Proporciona una vista del modelo del robot que se carga en el servidor de parámetros de ROS, y que es provisto como un archivo urdf con la descripción exacta de la constitución del robot. Captura la información de sus sensores, si existen, y reproduce estos datos para su visualización. Es importante señalar que rviz no es una herramienta o programa de simulación, para este caso existen otras opciones como Gazebo.

Rviz es ampliamente utilizado ya que le permite al usuario visualizar diversos elementos como el modelo del robot, sus ejes, el láser del escáner, entre muchos otros que pueden ser agregados al panel principal. La Figura 3.1 presenta una configuración básica en Rviz que contiene el modelo del famoso robot mira.

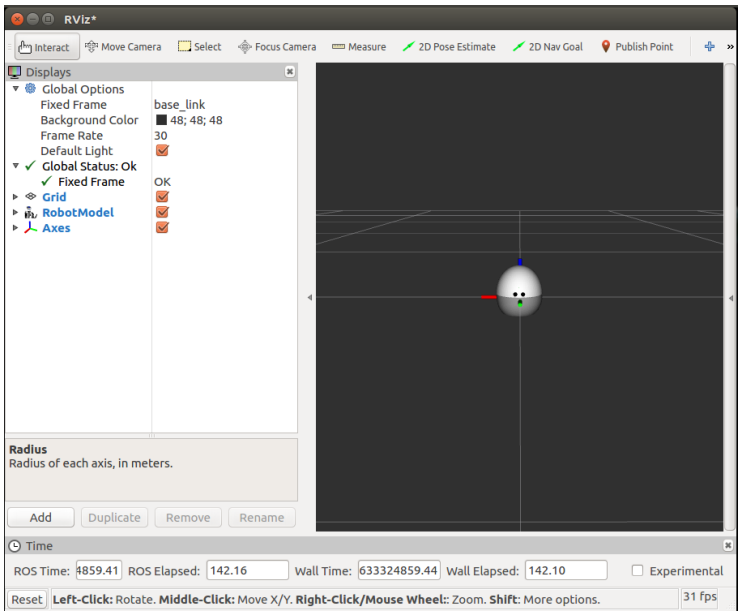

Figura 3.1 Configuración en Rviz para mostrar el modelo del robot mira

Independientemente de si se tiene un robot real o se realiza una simulación, Rviz es comúnmente usado para monitorear la interacción que genera el código que se instala en un robot según su propósito.

3.3 Navigation Stack

La pila de navegación 2D que ROS pone a disposición toma la información de la odometría, flujos de sensores y posición meta con el objetivo de generar comandos de velocidad que son enviados a una base móvil. Como requisito previo para el uso de la pila de navegación, el robot debe ejecutar ROS, tener un árbol de transformación tf en su lugar y publicar los datos del sensor utilizando los tipos de mensajes ROS correctos. Además, la pila de navegación debe configurarse para que la forma y la dinámica de un robot se desempeñe a un alto nivel.

Si bien esta pila está diseñada para ser de propósito general, es cierto que existen ciertos requerimientos referentes al hardware que deben cumplirse para funcionar de forma óptima:

1. Está diseñado para robots con ruedas de accionamiento diferencial y holonómicos únicamente. Se asume que la base móvil se controla enviando los comandos de velocidad deseados para lograrlo en forma de: velocidad x, velocidad y, velocidad theta.
2. Requiere un láser plano montado en algún lugar de la base móvil. Este láser se utiliza para la construcción y localización de mapas.
3. La pila de navegación se desarrolló en un robot cuadrado, por lo que su rendimiento será mejor en robots que sean casi cuadrados o circulares. Funciona en robots de formas y tamaños arbitrarios, pero puede tener dificultades con robots rectangulares grandes en espacios estrechos como puertas.

Dada la naturaleza de la solución implementada, esta pila de navegación es perfecta dado que el robot principal que será utilizado para llevar a cabo las pruebas es un robot de tipo diferencial, holonómico y de forma cuadrada.

Si bien la pila de navegación significa un fuerte punto de partida, aún se deben configurar diversos aspectos, afortunadamente la wiki oficial de ROS cuenta con varios tutoriales que brindan los pasos necesarios para establecer esta pila en un robot que cumpla con las restricciones ya mencionadas. De forma general, para que el navigation stack de ROS funcione se necesita de la siguiente información.

- Configuración de transformación

La pila de navegación requiere que el robot publique información sobre las relaciones entre los marcos de coordenadas usando tf, que es el árbol de transformación (tf) del robot. Esto se debe a que la referencia del robot, que debe ser el punto central del mismo, normalmente no se encuentra en la misma posición que el sensor laser que percibe la información de los rayos, por ello, es necesario transformar las distancias obtenidas desde el sensor para hacerlas coincidir con el marco (frame) base del robot.

- Información del sensor

  Para poder evitar obstáculos es necesario contar con la lectura del sensor que indica la presencia de los mismos, esto se realiza a través del envió constante de un tipo de mensajes específico de ROS.

- Información de la odometría

  La odometría está estrictamente relacionada con el árbol de transformación tf y esta información también se recibe a través de un tipo de mensaje específico de ROS.

- Controlador base

  Se asume que se pueden enviar comandos de velocidad al marco de coordenadas base, lo que permitirá que el robot pueda moverse.

- Mapas

  Si bien la pila de navegación no requiere explícitamente un mapa, dado que la solución implementada corresponde a navegación basada en mapas, es importante contar con uno o más de ellos.

3.4 Gmapping

Es un paquete que incluye un contenedor ROS para Gmapping de OpenSlam. Proporciona SLAM basado en laser que permite crear mapas de cuadrícula de ocupación 2D a partir de los datos recopilados por el láser y de pose del robot móvil. En la sección 1.2 se hace referencia a estos conceptos.

Este paquete contiene un nodo llamado slam_gmapping que es comúnmente usado para construir mapas. El requisito básico de hardware para hacer SLAM es un escáner láser que esté montado horizontalmente en la parte superior del robot, así como los datos de odometría del mismo. El mapa que se crea se publica en el tema /map durante todo el proceso de mapeo.

3.5 AMCL

Es un sistema de localización probabilística para un robot móvil en 2D. Implementa el enfoque de localización de Monte Carlo adaptativo (o muestreo KLD), como lo describe Dieter Fox, que utiliza un filtro de partículas para rastrear la pose de un robot en un mapa conocido. En la sección 1.2 se describen los aspectos y técnicas conocidas para resolver el problema de localización.

Toma un mapa basado en láser, escaneos láser y mensajes de transformación y genera estimaciones de pose. Al inicio, amcl inicializa su filtro de partículas de acuerdo con los parámetros proporcionados, en caso de no ingresar ningún parámetro, toma los valores por defecto definidos en su documentación.

Es importante señalar que amcl tiene muchas opciones de configuración, lo que podría afectar el desempeño de la localización. Convierte los escaneos láser de entrada al

frame de odometría (~odom_frame_id). Por lo tanto, debe existir una ruta a través del árbol tf desde el frame en el que se publican los escaneos láser hasta el frame de odometría.

Durante la operación, amcl estima la transformación del frame base (~base_frame_id) con respecto al frame global (~global_frame_id) pero sólo publica la transformación entre el frame global y el frame de odometría (~ odom_frame_id). Básicamente, esta transformación explica el desvio que se produce al utilizar *Dead Reckoning*. Las transformaciones publicadas tienen fecha futura. Las Figuras 3.2 y 3.3 muestran la diferencia que existe al realizar la localización con odometría y con amcl.

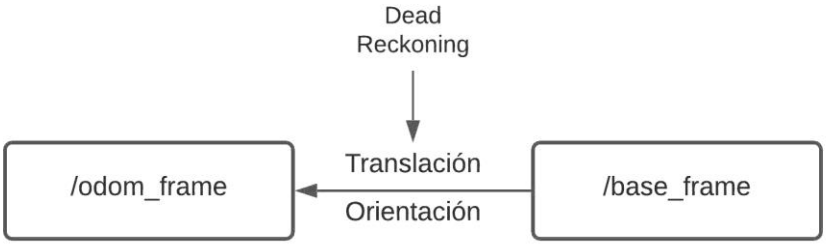

Figura 3.2 Localización usando odometría

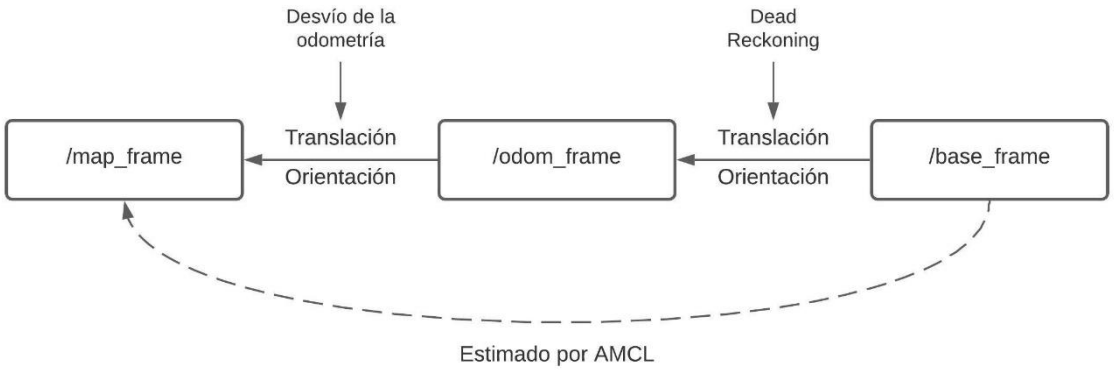

Figura 3.3 Localización usando AMCL

3.6 Move Base

El paquete move_base proporciona una implementación de una acción que, dado un objetivo en el mundo, intentará alcanzarlo con una base móvil. El nodo move_base vincula un planificador global y local para realizar su tarea de navegación global. Admite cualquier planificador global que se adhiera a la interfaz nav_core::BaseGlobalPlanner especificada en el paquete nav_core y cualquier planificador local que se adhiera a la interfaz nav_core::BaseLocalPlanner especificada en el paquete nav_core.

Haciendo un paréntesis necesario, el paquete nav_core contiene las interfaces clave para la pila de navegación. Todos los planificadores y comportamientos de recuperación que deseen utilizarse como complementos en el nodo move_base deben adherirse a estas interfaces.

La interfaz nav_core::BaseGlobalPlanner provee los distintos planificadores globales utilizados en la tarea de navegación. Actualmente existen tres planificadores dentro de esta interfaz:

- global_planner: Un planificador global rápido e interpolado creado como un reemplazo más flexible de navfn. Utiliza Dijkstra y A* como sus algoritmos para la navegación.
- navfn: Un planificador global basado en cuadrículas que utiliza una función de navegación para calcular la ruta de un robot. La función de navegación se calcula con el algoritmo de Dijkstra.

- carrot_planner: Un planificador global simple que toma un punto de objetivo especificado por el usuario e intenta mover el robot lo más cerca posible de él, incluso cuando ese punto de objetivo está en un obstáculo.

Vale la pena señalar que la lista no se reduce a estos planificadores, por el contrario, es posible crear otros basados en distintos algoritmos como RRT, a través de la creación de plugins en el nodo move_base. En el capítulo 2 se detallan con mayor profundidad algunos de los algoritmos que son utilizados en la interfaz para los planificadores globales.

Por otro lado, nav_core::BaseLocalPlanner proporciona una interfaz para los planificadores locales. Los planificadores locales que actualmente utilizan esta interfaz son:

- base_local_planner: Proporciona implementaciones del enfoque de ventana dinámica (DWA) y enfoques de despliegue de trayectoria para el control local.
- dwa_local_planner: Implementación de DWA modular con una interfaz mucho más limpia y fácil de entender, así como variables de eje más flexibles para robots holonómicos, en comparación con DWA de base_local_planner
- eband_local_planner: Implementa el método Elastic Band en el colector SE2.
- teb_local_planner: Implementa el método Timed-Elastic-Band para la optimización de la trayectoria en línea.
- mpc_local_planner: Proporciona varios enfoques de control predictivo de modelos integrados en el colector SE2.

Ahora bien, conviene enfatizar los enfoques mencionados en los planificadores locales a fin de comprender como operan. La idea básica de los algoritmos de despliegue de trayectoria (Trajectory Rollout) y enfoque de ventana dinámica (DWA) es la siguiente:

1. Muestrear discretamente en el espacio de control del robot (dx, dy, dtheta).
2. Para cada velocidad muestreada, se realiza una simulación hacia adelante desde el estado actual del robot para predecir lo que sucedería si la velocidad muestreada se aplicara durante un período de tiempo (corto).
3. Se evalúa cada trayectoria resultante de la simulación de avance, utilizando una métrica que incorpore características tales como: proximidad a obstáculos, proximidad a la meta, proximidad a la ruta global y velocidad. Se descartan trayectorias ilegales (aquellas que chocan con obstáculos).
4. Se elige la trayectoria de mayor puntuación y se envía la velocidad asociada a la base móvil.
5. Limpiar y repetir.

DWA se diferencia de Trajectory Rollout en cómo se muestrea el espacio de control del robot. El despliegue de trayectoria toma muestras del conjunto de velocidades alcanzables durante todo el período de simulación hacia adelante dados los límites de aceleración del robot, mientras que DWA toma muestras del conjunto de velocidades alcanzables para un solo paso de simulación dados los límites de aceleración del robot. Esto significa que DWA es un algoritmo más eficiente porque muestrea un espacio más pequeño, pero Trajectory Rollout puede superarlo en robots con límites de aceleración bajos porque DWA no simula aceleraciones constantes hacia adelante. Sin embargo, en la práctica, se encuentra que DWA

y Trayectoria Rollout funcionan de manera comparable, así que se puede recomendar el uso de DWA por sus ganancias de eficiencia.

El nodo move_base también mantiene dos mapas de costos, uno para el planificador global y otro para el planificador local que se utilizan para realizar tareas de navegación. El paquete costmap_2d proporciona una implementación de un mapa de costos 2D que toma datos de sensores del mundo, construye una cuadrícula de ocupación 2D o 3D de los datos (dependiendo de si se utiliza una implementación basada en vóxeles) e infla los costos en un mapa de costos 2D basado en la cuadrícula de ocupación y un radio de inflación.

Finalmente, es importante decir que el nodo move_base proporciona una interfaz ROS para configurar, ejecutar e interactuar con la pila de navegación en un robot. La Figura 3.4 muestra una vista de alto nivel de este nodo y su interacción con otros componentes, varios de los cuales ya se han discutido en secciones anteriores. Los nodos en azul varían según la plataforma del robot, en gris es opcional, pero se proporciona para todos los sistemas, y los nodos blancos son obligatorios, pero también se proporcionan para todos los sistemas.

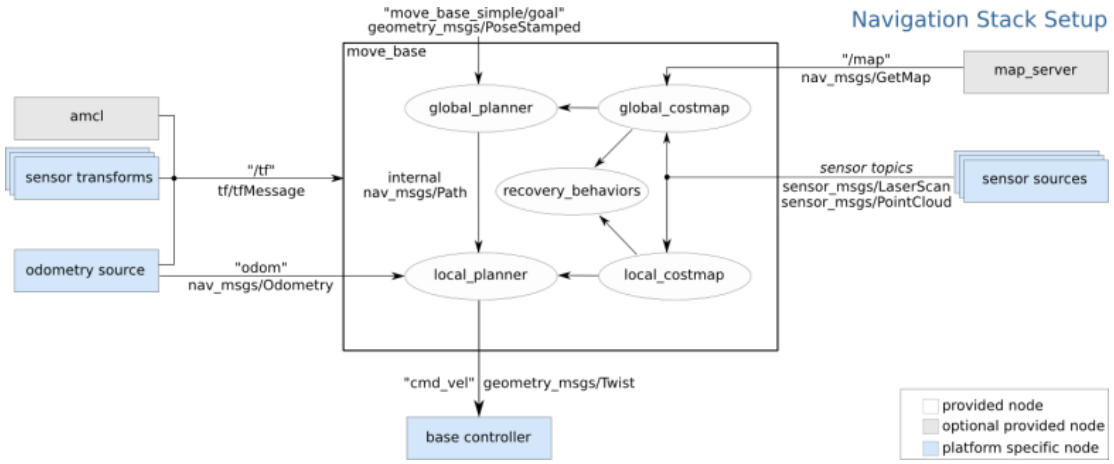

Figura 3.4 Vista de alto nivel del nodo move_base

Capítulo 4

4 Propuesta del framework

Como se menciona en el capítulo 1, la navegación de robots móviles conlleva una serie de pasos para funcionar, de hecho, implementar un sistema de navegación para un robot desde cero definitivamente consumiría mucho tiempo y esfuerzo. Afortunadamente, ROS provee esta funcionalidad a través del paquete Navigation Stack lo que facilita la tarea principal, sin embargo, esta pila de trabajo es descentralizada, lo que significa que cada paso de la navegación se debe realizar de forma separada y como un proceso independiente, lo que puede dificultar la labor sobre todo a personas con poca experiencia en el área de la robótica o en el uso de ROS.

La propuesta que se realiza entonces, es construir un framework que incluya todas las funcionalidades de los nodos de la pila de navegación en una única ventana, con la posibilidad de escoger distintos mapas y configurar parámetros. Para entender porque esto significaría una mejora y contribución a la comunidad de ROS, en la siguiente sección se describe la forma tradicional de poner a punto la pila de navegación en el robot Pioneer 3DX.

4.1 Forma tradicional

Para poder implementar la pila de navegación en un robot con ROS se suele seguir una serie de pasos generales haciendo uso de diversos nodos y paquetes y que en varios casos requieren configuraciones específicas de parámetros de acuerdo al modelo del robot al que se le desea integrar el sistema de navegación. Se asume que el modelo del robot ya se encuentra en el servidor de parámetros de ROS.

4.1.1 Iniciar ROS Core

El primer paso y el más fundamental antes de querer hacer cualquier cosa, es iniciar el proceso principal que administra el sistema ROS. Siempre se debe estar ejecutando el comando roscore en una consola independiente. La Figura 4.1 muestra la ejecución de dicho comando.

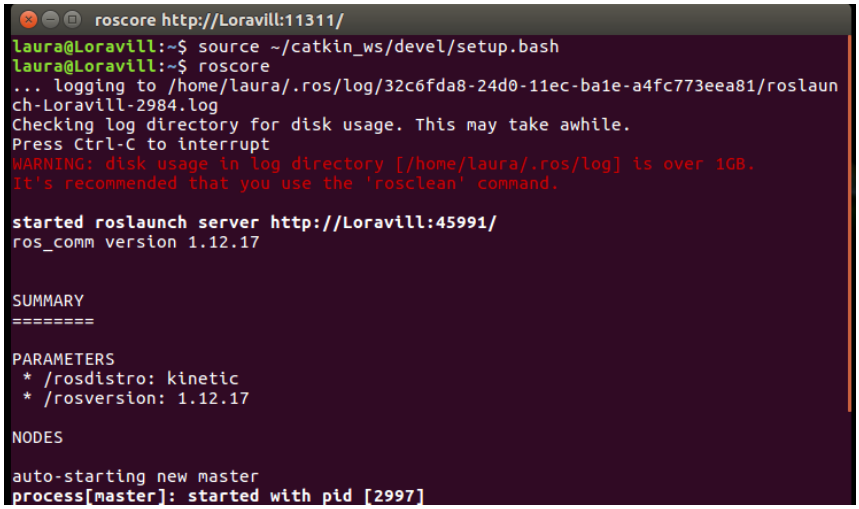

Figura 4.1 Iniciando el proceso principal de ROS mediante el comando roscore

4.1.2 Construir un mapa

Como ya se mencionó en el apartado correspondiente, la pila de navegación funciona con o sin un mapa, sin embargo, la propuesta del framework requiere de un mapa a priori

para realizar la tarea de navegación. Es aquí que entra el paquete Gmapping y que a través del nodo slam_gmapping permite crear un mapa basado en las lecturas recibidas desde un sensor láser.

Suponiendo que se tiene el ambiente simulado mostrado en la Figura 4.2, se requiere obtener un mapa de cuadrículas de ocupación 2D a partir de este, dicho mapa contendrá la información necesaria para realizar las tareas consecuentes de la navegación.

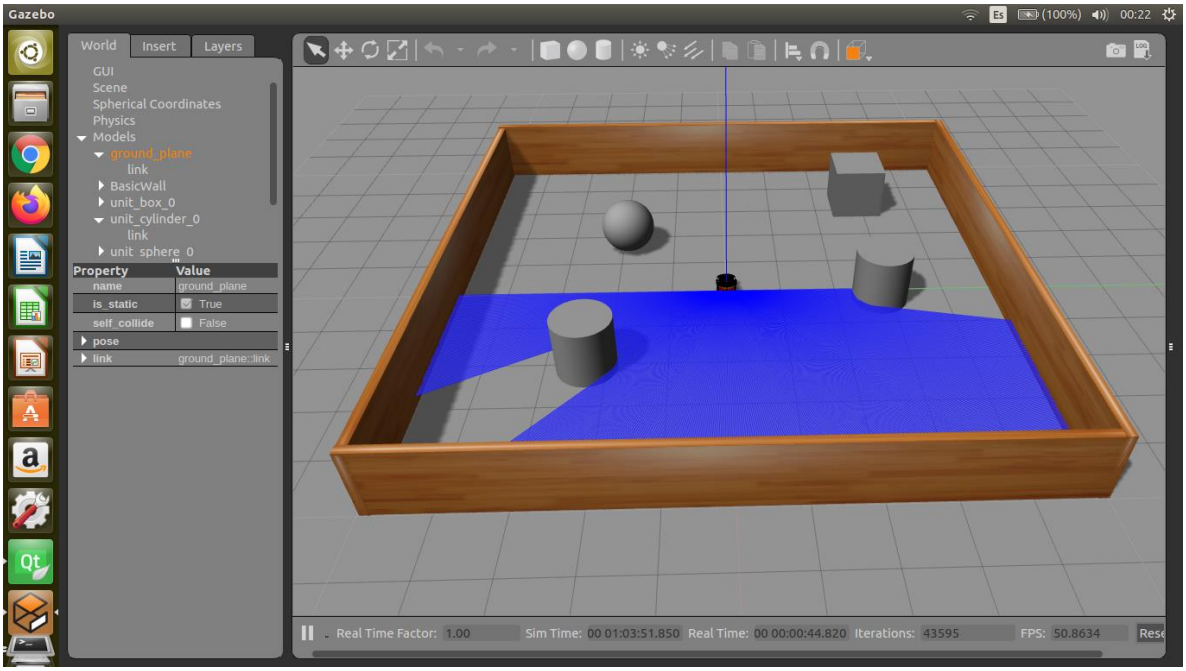

Figura 4.2 Ambiente 3D construido en Gazebo Simulator

Para lograr la construcción del mapa, se debe contar con 3 terminales: la primera debe ejecutar el nodo slam_gmapping pasándole como parámetro el nombre del sensor laser montado en el robot, lo que permitirá empezar con el registro de los datos que se perciben con el sensor, ver Figura 4.3. Este nodo también pone a disposición el tema /map, lo que permite visualizar cómo es que se va creando el mapa mientras el robot se mueve.

La segunda terminal debe ejecutar un nodo que permita mover el robot a través del ambiente, suele ocuparse código personalizado para lograr esta tarea. En este caso específico, se cuenta con un programa que permite controlar el movimiento del robot con las flechas del teclado. La Figura 4.4 muestra la terminal con el programa ejecutado.

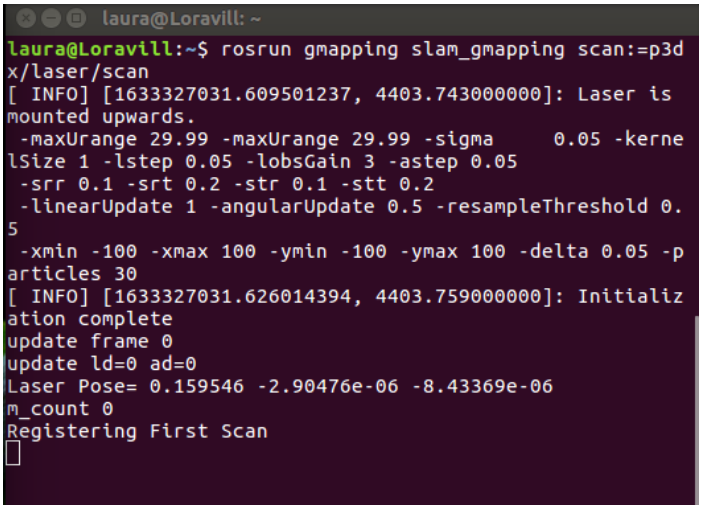

Figura 4.3 Ejecución del nodo slam_gmapping del paquete Gmapping

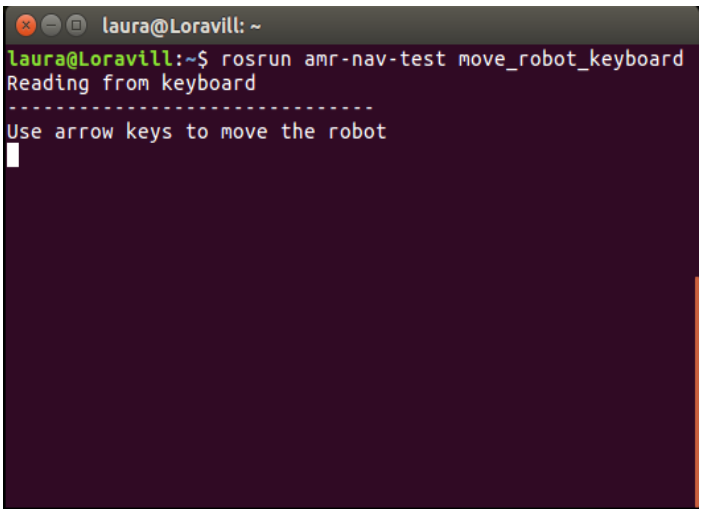

Figura 4.4 Ejecución del nodo para mover un robot

Antes de continuar con la tercera terminal, hasta este punto y con las dos consolas ejecutando sus nodos correspondientes se puede empezar a construir el mapa, la Figura 4.5 muestra el panorama completo y la interacción entre Rviz y las consolas ya descritas. A medida que se mueve el robot a lo largo del mapa, este se va coloreando, las áreas grises representan el espacio ya explorado.

Cuando el área completa se encuentre en color gris se debe guardar el mapa generado. Aquí entra otro nodo que deberá ser ejecutado en una nueva terminal, siguiendo la numeración, esta es la tercera. Para poder guardar el mapa, se debe ejecutar el nodo map_saver del paquete map_server, pasándole como parámetro la ubicación y nombre del nuevo mapa.

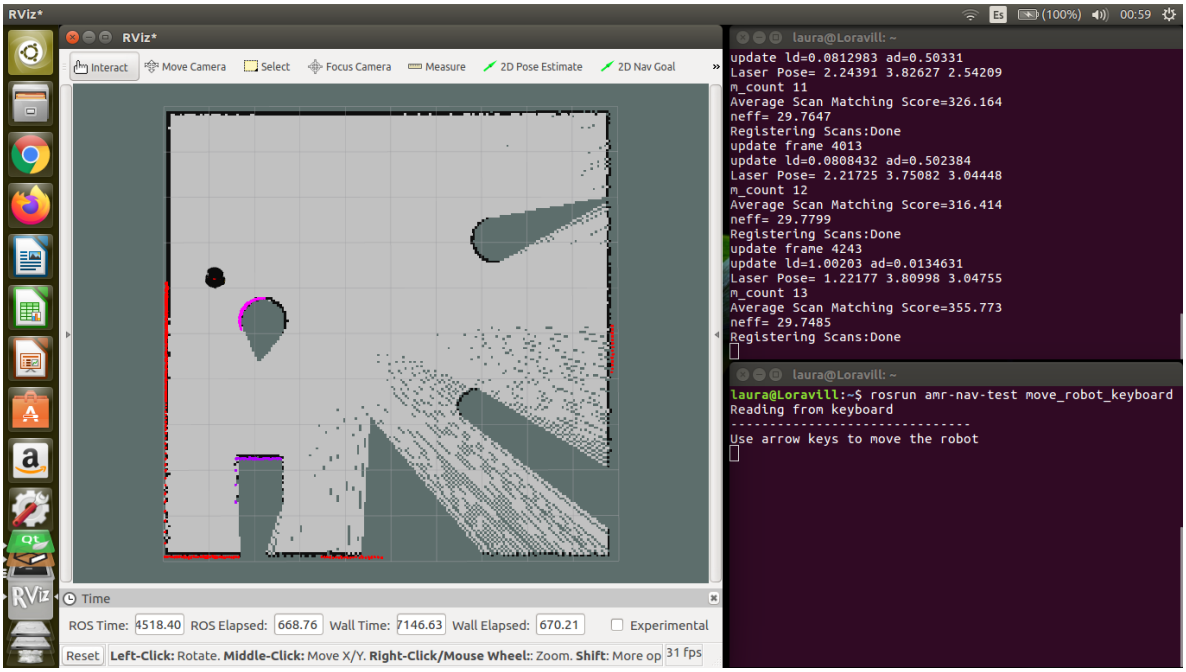

Figura 4.5 Construcción de un mapa de cuadriculas 2D

Lo anterior genera dos archivos, uno con extensión pgm que muestra la imagen del mapa y un archivo yaml relacionado que contiene metadatos del mapa. La Figura 4.6 muestra el resultado obtenido después de explorar el mapa con gmapping.

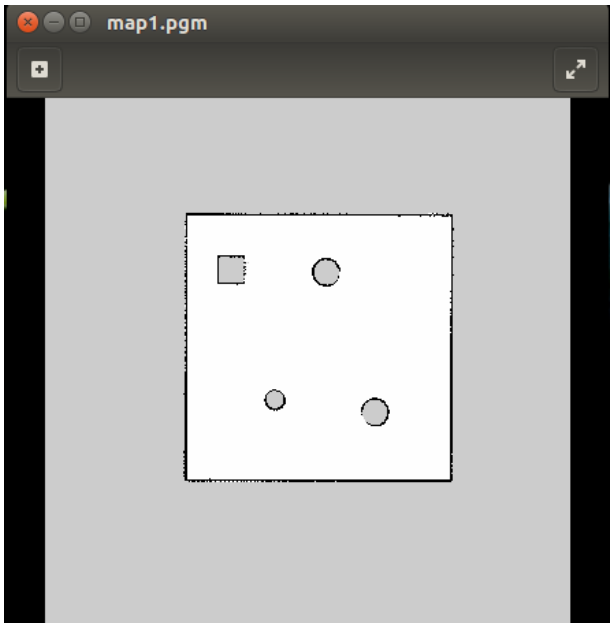

Figura 4.6 Mapa generado con Gmapping

Una vez obtenida la imagen del mapa los procesos que se encontraban ejecutando pueden ser detenidos, su propósito ya ha sido cumplido.

4.1.3 Creación de un paquete

Habiendo cumplido con los requisitos básicos, el primer paso para implementar el sistema de navegación en el robot es crear un paquete que contenga todas las configuraciones y archivos launch. Los archivos launch son, en esencia, archivos XML con las ordenes y parámetros para ejecutar uno o varios nodos en un solo paso, evitando así escribir los comandos directamente en las ventanas de línea de consola. El primer archivo launch deberá

lanzar los nodos necesarios para obtener los datos del sensor, la odometría del robot y la configuración de transformación del árbol tf, ya que, sin estos, el sistema de navegación simplemente no funcionará.

Dentro de este paquete, también es necesario contar con los archivos de configuración de mapa de costos, es decir, se debe determinar cuáles serán los planificadores global y local que serán utilizados para llevar a cabo la navegación. Estos archivos deben ser de extensión yaml y contar con los parámetros configurados que mejor se adecuen al robot que se está utilizando. De forma general se ocupan un total de 3 archivos, el primero con las configuraciones comunes de los mapas de costos, los otros corresponden a la configuración de costos global y local.

Además, se tiene también un archivo con la configuración del base_local_planner, que es responsable de calcular los comandos de velocidad para enviar a la base móvil del robot dado un plan de alto nivel.

4.1.4 Ejecutar la pila de navegación

Este paso es el resultado de todas las acciones realizadas con anterioridad. Recapitulando, se cuenta con un mapa de ocupación de cuadrícula 2D con las características del ambiente en el que se encuentra el robot y se cuenta con los archivos con las configuraciones requeridas para el nodo move_base. Lo que sigue es cargar el mapa al servidor, ejecutar el nodo amcl para la localización y finalmente ejecutar el nodo move_base.

Entonces, se debe cargar el mapa del ambiente en el que se desea navegar, esto se realiza mediante el nodo map_server del paquete map_server, la ubicación del mapa a

mostrar es un parámetro que debe ser establecido antes de ejecutar el nodo. La Figura 4.7 muestra de qué manera se habilita el mapa en el servidor de mapas de ROS.

Lo siguiente es iniciar la localización a través del nodo amcl, al hacerlo también se habilitará el frame map, que es la referencia en la que se basa la odometría del robot con respecto a un mapa. Este nodo recibe las transformaciones necesarias desde el frame base para poder mantenerse localizado correctamente, ver Figura 4.8. En este sentido, amcl se mantiene publicando en el tema /particlecloud el conjunto de estimaciones de pose que mantiene el filtro.

Al haber realizado la anterior, es posible agregar un elemento PoseArray en Rviz que permitirá subscribirse al tema /particlecloud, con esto se habilitarán flechas direccionales. Para saber si la localización se está realizando de forma correcta, basta con ver la dispersión de las flechas, si se concentran en un lugar cerca del robot quiere decir que la precisión es bastante alta, de lo contrario es una señal de que el robot no se encuentra bien localizado

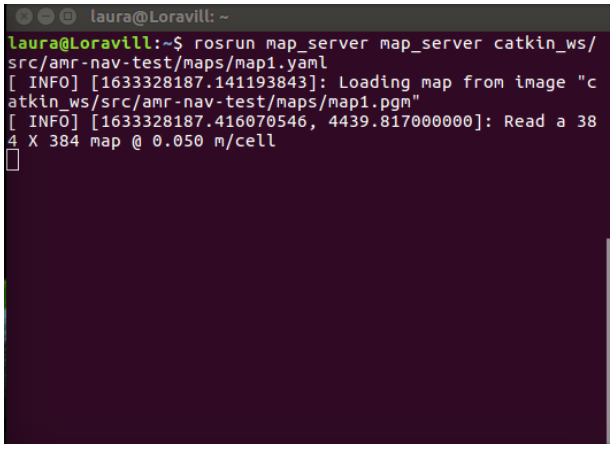

Figura 4.7 Proceso para cargar un mapa al servidor

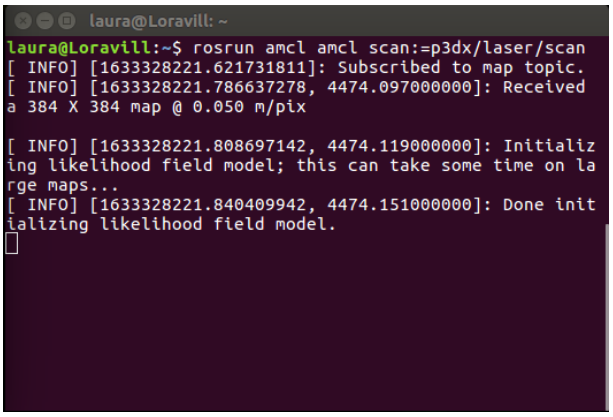

Figura 4.8 Ejecución del nodo amcl

dentro del mapa, lo que derivará en errores de cálculo para la navegación. La Figura 4.9 muestra la interacción de la ventana Rviz con las ventanas de comandos que ejecutan los diversos nodos.

Para finalizar, el último nodo que se debe ejecutar es el nodo move_base, a decir, es el más importante ya que carga con las configuraciones de los mapas de costos y los parámetros del planificador base local. Dado que move_base requiere de diversos parámetros, los cuales están declarados en los archivos yaml, lo mejor es ejecutar este nodo desde un archivo launch. La Figura 4.10 muestra la estructura para lanzar la ejecución del nodo contenido en el archivo especificado.

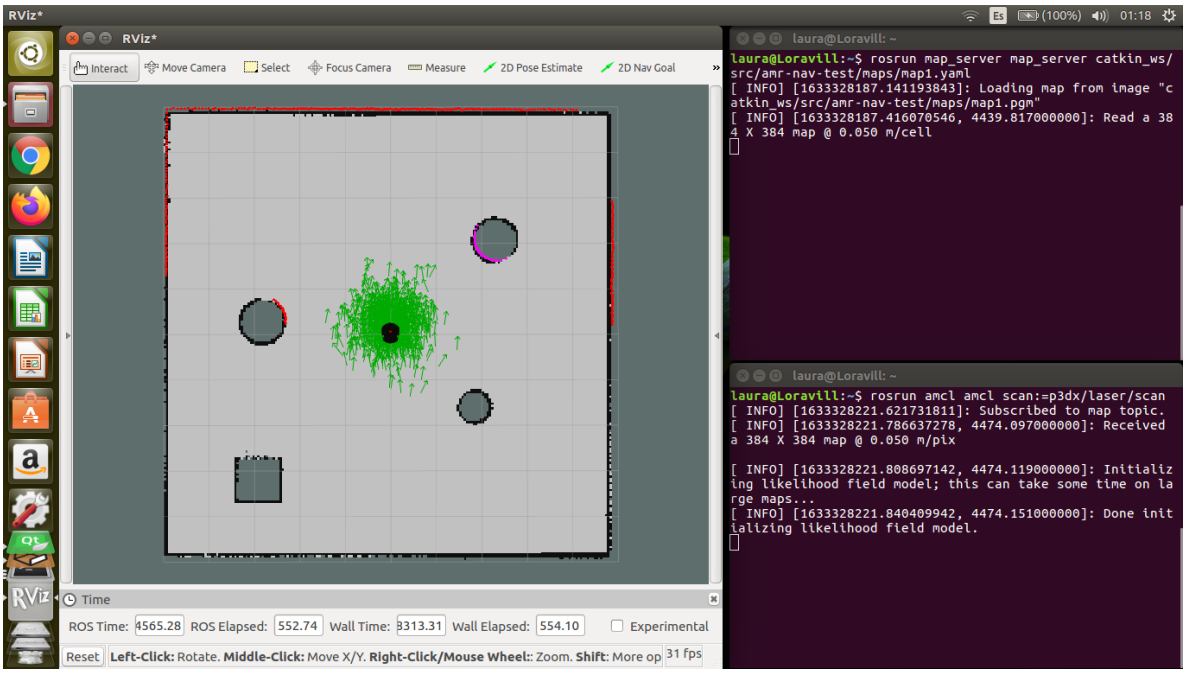

Figura 4.9 Ejecución del nodo map_server y amcl

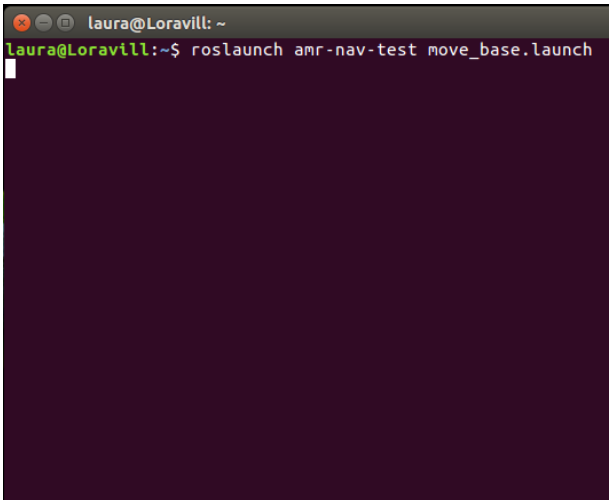

Figura 4.10 Ejecución del nodo move_base

Una vez iniciado el nodo move_base, nuevos temas se ponen a disposición, por lo tanto, es posible configurar la vista de dos elementos de tipo mapa en Rviz. El primero será un mapa estático definido dentro del mapa de costos global. El segundo mapa enfatiza las

lecturas del sensor y se basa sobre el mapa de costos local, de hecho, es posible visualizar de manera más robusta los distintos obstáculos en el mapa. La Figura 4.11 muestra la interacción de la ventana de comandos que se encuentra ejecutando el archivo launch y la venta Rviz con sus diversos elementos.

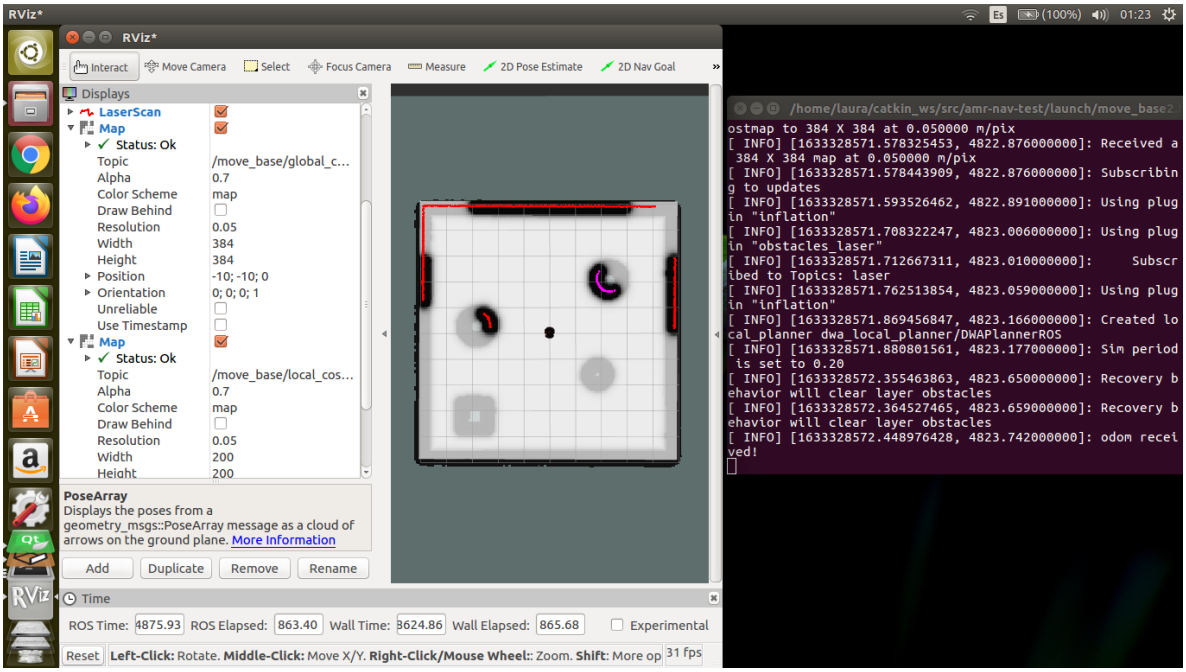

Figura 4.11 Interacción del nodo move_base con Rviz

A partir de este momento, ya es posible definir puntos en el mapa como metas para que la pila de navegación calcule un camino y envíe los comandos de velocidad a la base para realizar el recorrido hacia el objetivo. Para visualizar el camino calculado se debe agregar un elemento de tipo Camino (Path) que se suscribe al tema con el nombre del planificador base local. En la Figura 4.12 se observa la adición de este elemento al panel de Rvix, como se nota, cuando se señale un punto meta y se genere un camino, este se mostrará como una línea de color azul.

Rviz incluye la funcionalidad de establecer un punto meta en el mapa, lo que envía la señal para calcular el camino que conecta los puntos de partida y final, la flecha verde grande que se muestra en la Figura 4.13 designa el punto y orientación de la posición meta.

Lo que ocurre a continuación es la generación del camino, ver Figura 4.14, y el envío de los comandos de velocidad a la base móvil hasta que el robot llega a la posición meta, la Figura 4.15 presenta la posición del robot dentro del mapa después de alcanzar la meta.

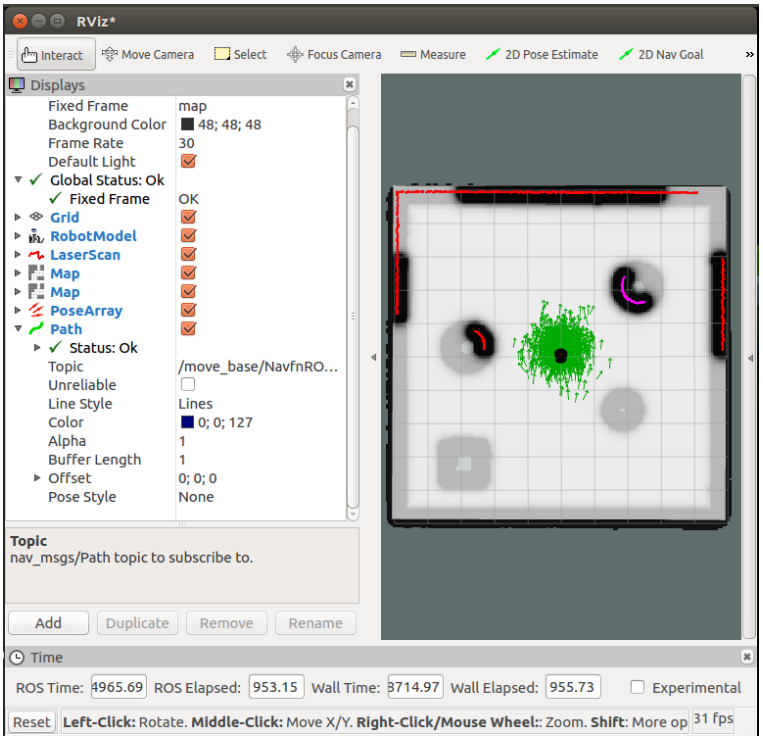

Figura 4.12 Adición del elemento Path en Rviz

Figura 4.13 Selección de la posición meta en el mapa

Figura 4.14 Generación del camino entre un punto inicial y un punto final

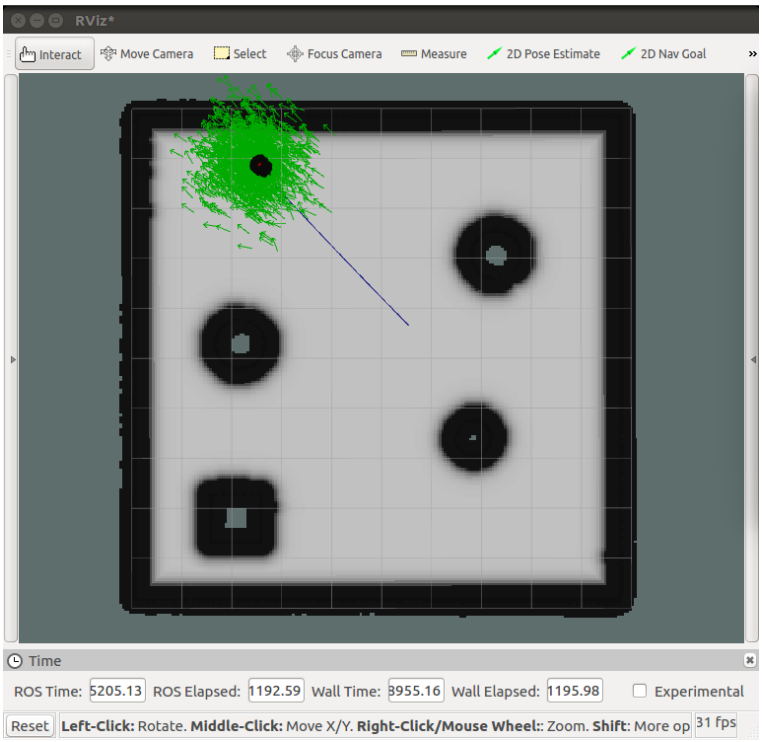

Figura 4.15 Robot que ha alcanzado la posición objetivo

Como se puede observar se ha implementado satisfactoriamente la pila de navegación en el robot Pioneer 3DX. Sin embargo, es claro que para lograrlo se tuvieron que ejecutar diversos nodos de forma separada en un total de 7 terminales, sin mencionar la instancia de Rviz presente a la cual se le tuvieron que ir agregando diversos elementos conforme se fue avanzado en el proceso. Pese a que ROS provee un sistema de navegación para robots diferenciales, es un hecho que reina el caos cuando se trata de implementar una pila de navegación desde cero, más aún, para personas con poco conocimiento en ROS, ¡es todo un reto!

Es por ello que la propuesta que se presenta en este trabajo de tesis toma todos los procesos, ejecuciones, y demás, y las embebe en una sola ventana, intuitiva y con

configuraciones y modelos precargados para evitar todo el proceso que conlleva la pila de navegación y centrarse únicamente en observar el funcionamiento mediante el envió de distintas posiciones meta.

4.2 AMR NAV

4.2.1 Descripción

Se propone la creación de un framework, AMR NAV (Autonomous Mobile Robot Navigation), basado en ROS para la navegación autónoma de robots móviles, tomando como punto de partida el paquete navigation stack. Este framework busca reunir todas las actividades que se deben llevar a cabo, como parte de la pila de navegación, en un solo ecosistema que minimice el tiempo necesario para configurar un sistema de este tipo en un robot, al mismo tiempo que muestra la relación directa de estos pasos en una visualización 3D. Además. Busca ser una herramienta de aprendizaje para los nuevos y antiguos interesados en el mundo de la robótica y de ROS.

4.2.2 Requisitos

Como parte de los requisitos de AMR NAV, se tienen los siguientes:

- Debe separar y presentar cada paso de la pila de navegación de forma clara y sencilla en una sola ventana.
- Debe permitir seleccionar el modelo del robot y el mapa estático sobre los cuales se realizará la navegación.
- Debe presentar una descripción o vista previa de los modelos antes de cargarlos al servidor.
- Debe permitir controlar al robot para su posicionamiento en el mapa.

- Debe configurar el ambiente simulado en Gazebo y abrir una instancia de esta simulación como referencia.
- Debe mostrar una visualización en 3D como se haría en Rviz, con la diferencia de que cada elemento que se utilice será agregado de forma automática cuando así se necesite.
- Debe permitir seleccionar entre los diferentes planificadores globales y locales que utiliza ROS.
- Debe permitir seleccionar un punto dentro del mapa como punto objetivo para la tarea de navegación.
- Debe gestionar los nodos, procesos y mensajes involucrados entre ROS y Gazebo
- AMR NAV debe acortar el tiempo requerido para habilitar un sistema de navegación en un robot utilizando ROS.

4.2.3 Suposiciones

Este framework debe ser simple y sencillo de usar, sin embargo, se recomienda estar en sincronía con las definiciones y conceptos básicos de ROS, ya que no se profundiza en cada paso de la navegación, más bien se describe la implicación y resultados esperados en cada uno de ellos. Aún para una persona que recién está centrando su atención en esta área, es preferible conocer, al menos, de forma general lo que implica la programación de navegación en robots autónomos.

4.2.4 Arquitectura

AMR NAV interactúa con los nodos que provee ROS como parte de su pila de navegación, los nodos propios que le permiten gestionar los procesos descritos en el apartado 4.1 y la simulación de Gazebo. De forma general, la Figura 4.16 muestra la arquitectura de alto nivel de AMR NAV y su interacción con los sistemas antes mencionados.

Como se observa, la propuesta realizada provee un sistema para controlar los procesos detrás de cada paso para la navegación a través de una interfaz de usuario.

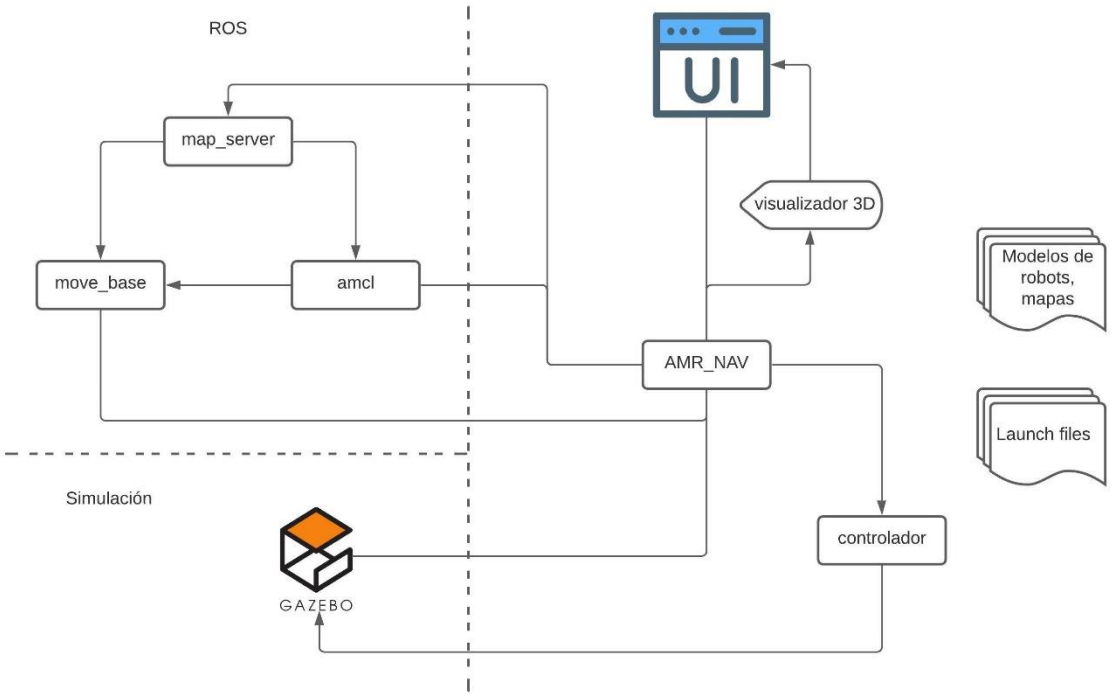

Figura 4.16 Arquitectura de alto nivel de AMR NAV

Del lado de ROS, existen tres nodos esenciales que se comunican entre ellos para cumplir las tareas involucradas en la navegación, estos nodos a su vez, mandan y reciben mensajes a un nodo principal AMR_NAV que, de hecho, es parte del trabajo realizado.

AMR NAV se compone de las siguientes características:

- Un nodo AMR_NAV que gestiona de forma interna cada proceso involucrado en la pila de navegación. Se encarga de lanzar la ejecución de cada nodo provisto por ROS mediante el uso de hilos. A su vez, se ocupa de configurar y crear una instancia del ambiente simulado en Gazebo.
- Un nodo controlador que permite mover al robot a través del mapa, este nodo envía comandos de velocidad a la base del robot móvil.
- Una interfaz gráfica que permite acoplar y presentar cada paso necesario para instaurar la pila de navegación en un robot. Así mismo, permite seleccionar entre diversas configuraciones y modelos predefinidos.
- Un visualizador 3D que presenta el comportamiento de rviz de forma personalizada. Este visualizador se encuentra incrustado en la ventana principal de la interfaz gráfica, pero de hecho es un proceso independiente que se comunica directamente con AMR_NAV.
- Una serie de archivos, modelos y mapas ya probados para ser utilizados como punto de referencia para la ejecución de los nodos ROS y que pueden ser configurados para ocuparse con distintos robots, generalizando así algunos parámetros básicos.

A fin de complementar el esquema de la Figura 4.16 se presenta un diagrama que muestra de forma más detallada las distintas actividades involucradas en esta propuesta, suponiendo un seguimiento lineal de los pasos que se llevan a cabo para ejecutar la navegación. La Figura 4.17 detalla lo antes mencionado.

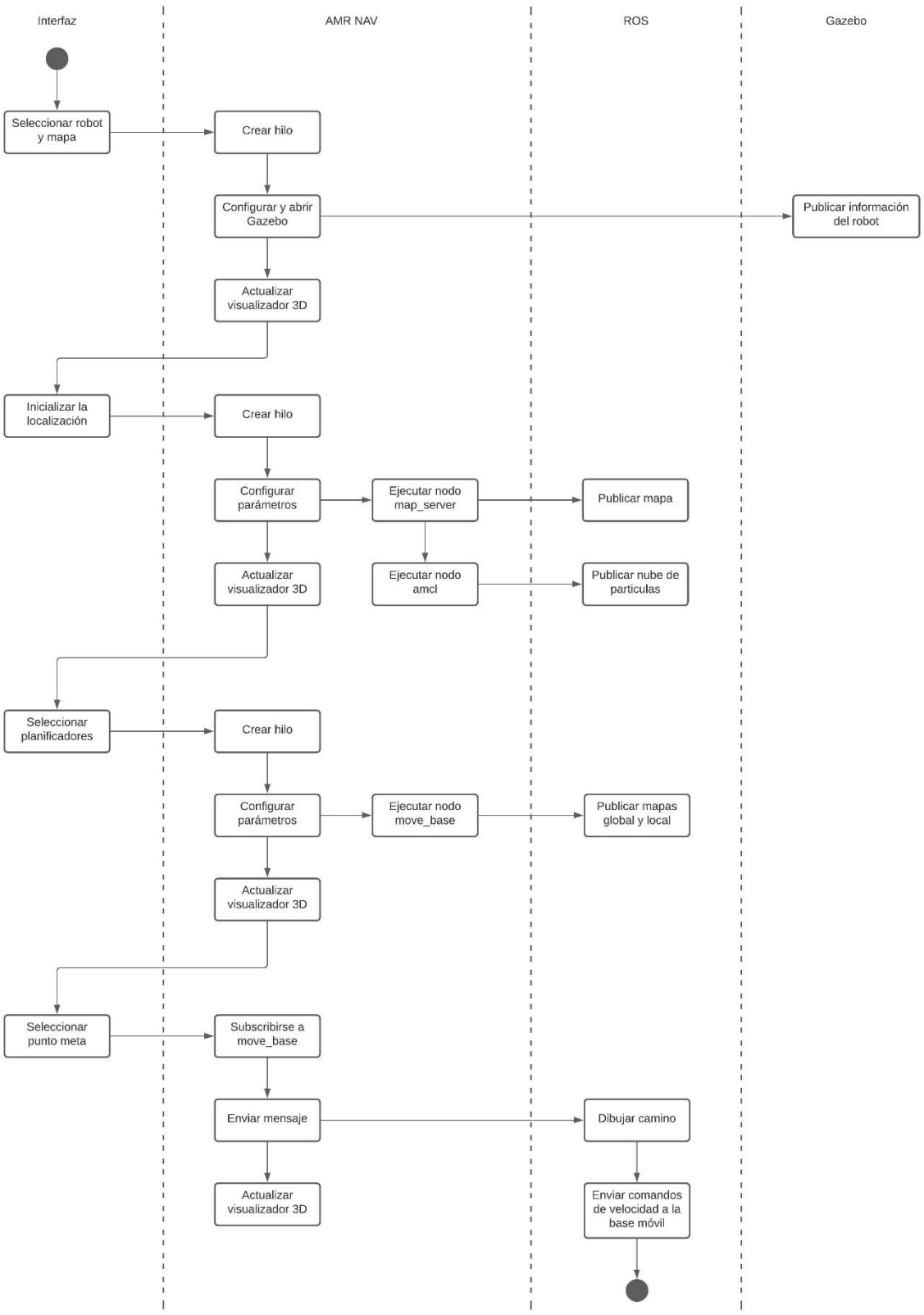

Figura 4.17 Diagrama de actividades de AMR NAV

4.2.5 Diseño

Habiendo defino los requisitos y descrito el comportamiento interno de AMR NAV, el diseño de la interfaz gráfica es lo que sigue. Dado que uno de los objetivos de este framework es acoplar los distintos pasos que se incluyen en la pila de navegación de ROS en una sola ventana, la interfaz de usuario debe ser simple, pero al mismo tiempo explicativa, es decir, debe ser capaz de reflejar de forma intuitiva su uso. Por ello, se proponen los diseños mostrados en las Figuras 4.18 y 4.19.

Como se observa, AMR NAV consta de una sola ventana que abarca la totalidad de la pantalla, de lado izquierdo se tiene un visualizador 3D que es de hecho un rviz personalizado, inmediatamente debajo de este, se encuentran los botones que permiten controlar al robot para moverlo a voluntad en todo el mapa. Como nota importante, estos controles deben deshabilitarse al ejecutar el paso número 4, pues se espera que el nodo

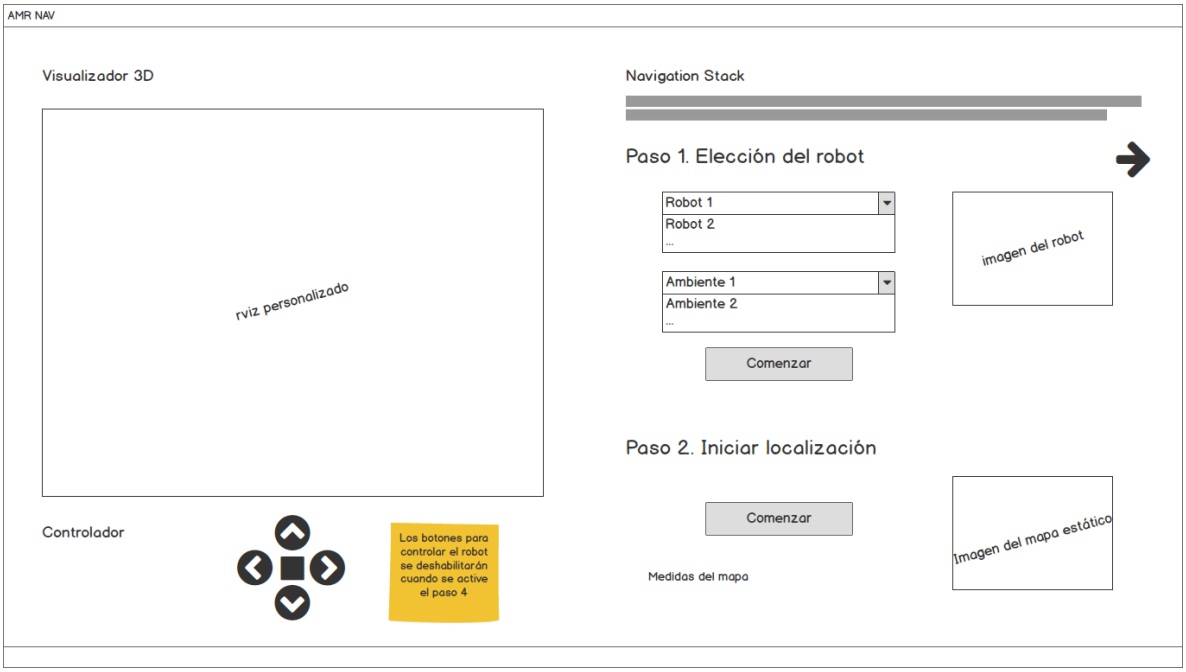

Figura 4.18 Diseño propuesto para AMR NAV (Pasos 1 y 2)

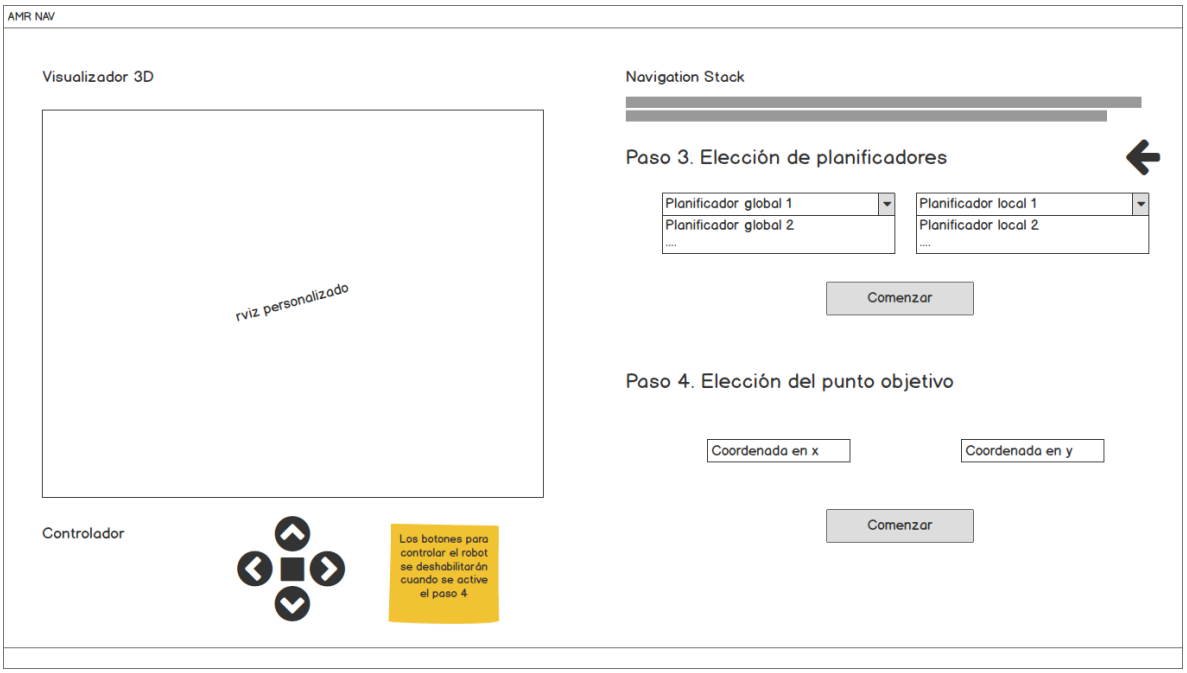

Figura 4.19 Diseño propuesto para AMR NAV (Pasos 3 y 4)

move_base envíe comandos de movimiento a la base móvil del robot, por lo que se podrían cruzar las instrucciones y provocar una falla en el paso final hacia un punto meta. Por otro lado, en la parte derecha de la pantalla se tienen los pasos específicos que son necesarios para inicializar la pila de navegación en un robot seleccionado. Al ser 4 los pasos, se optó por mostrarlos de dos en dos para una mejor distribución, de este modo se tiene un botón en la parte superior que permite cambiar entre los pasos 1-2 a 3-4 y viceversa.

Además, como regla básica, se cuenta con dos ventanas emergentes (diálogos) que muestran la información sobre AMR NAV y la ayuda. El diseño es genérico para ambos, por lo que la Figura 4.20 muestra esta estructura que ambas ventanas presentan.

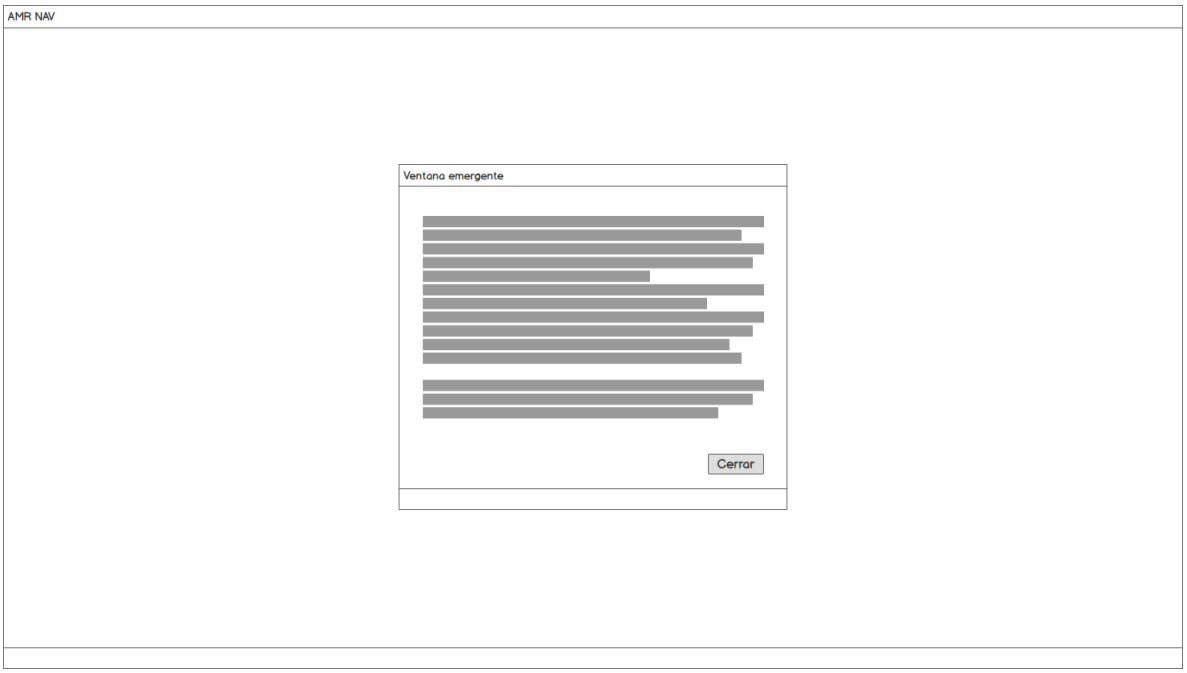

Figura 4.20 Diseño propuesto para AMR NAV (Ventana flotante)

4.2.6 Recursos

Para el desarrollo y prueba del framework AMR NAV se utilizaron los siguientes recursos y herramientas:

Tabla 4.1 Recursos y herramientas utilizadas en el desarrollo de la propuesta

Recurso/Herramienta	Descripción	Justificación
CPU	Procesador AMD Ryzen™ 5 2500U	Requerido debido a la carga de distintos procesos simultáneos.
GPU	Gráficos AMD Radeon™ Vega 8	Dado que se incluye un sistema de visualización y simulación 3D, es necesario un buen rendimiento gráfico.
Sistema Operativo	Ubuntu 14.04 LTS, ROS	ROS es la base del desarrollo de este framework ya que se hace uso de sus distintos paquetes y librerías.

Simulador	Gazebo Simulator	Necesario para la establecer la simulación de un robot real en diversos entornos.
IDE	Qt Creator	Utilizado para el desarrollo del framework. Permite crear aplicaciones con interfaz de usuario.
Línea de Comandos	Consola Ubuntu	Requerida para la ejecución de los paquetes y nodos en modo de prueba.
Lenguaje de Programación	C++	Por su capacidad de administrar y ejecutar diversos procesos es que este lenguaje es utilizado.

4.2.7 Implementación

Habiendo definido lo anterior, en este apartado se muestra el resultado obtenido durante la etapa de desarrollo. De este modo, siguiendo la lista de requisitos y el diseño planteado, se obtiene una interfaz de usuario simple pero concisa.

Lo primero que se observa al ejecutar AMR NAV es una ventana que centraliza diversas tareas: un panel de visualización 3D, un sistema de controles y la sección correspondiente a la pila de navegación con sus diferentes pasos.

Como se menciona en el capítulo 3, ROS cuenta con una herramienta de visualización 3D llamada Rviz, y como se constató en la sección 4.1, esta instancia (ventana) normalmente se encuentra a la vista para poder llevar un mejor control de los elementos y modelos. Sin embargo, también se observa que, al tener que abrir diversas consolas para ejecutar los distintos nodos, la ventana Rviz debe ser ajustada en su tamaño para poder observar ambos:

visualizador y consolas, al mismo tiempo. Esto implica una mala distribución gráfica de estas herramientas y que pueden entorpecer el monitoreo de los mensajes recibidos y publicados a través de los diversos temas en ROS. Por ello, AMR NAV incluye un solo panel, sin la necesidad de agregar, conforme se avanza en los pasos de la pila de navegación, nuevos elementos. A medida que se ejecutan los pasos en AMR NAV de forma automática muestra los modelos, lecturas de sensor, mapas, etc. requeridos, ahorrando espacio y enfocándose solamente en los elementos gráficos más importantes.

Las Figuras 4.21 y 4.22 muestran el entorno de la ventana principal de este framework. En la primera se muestran los 2 primeros pasos del navigation stack, en la otra los dos faltantes.

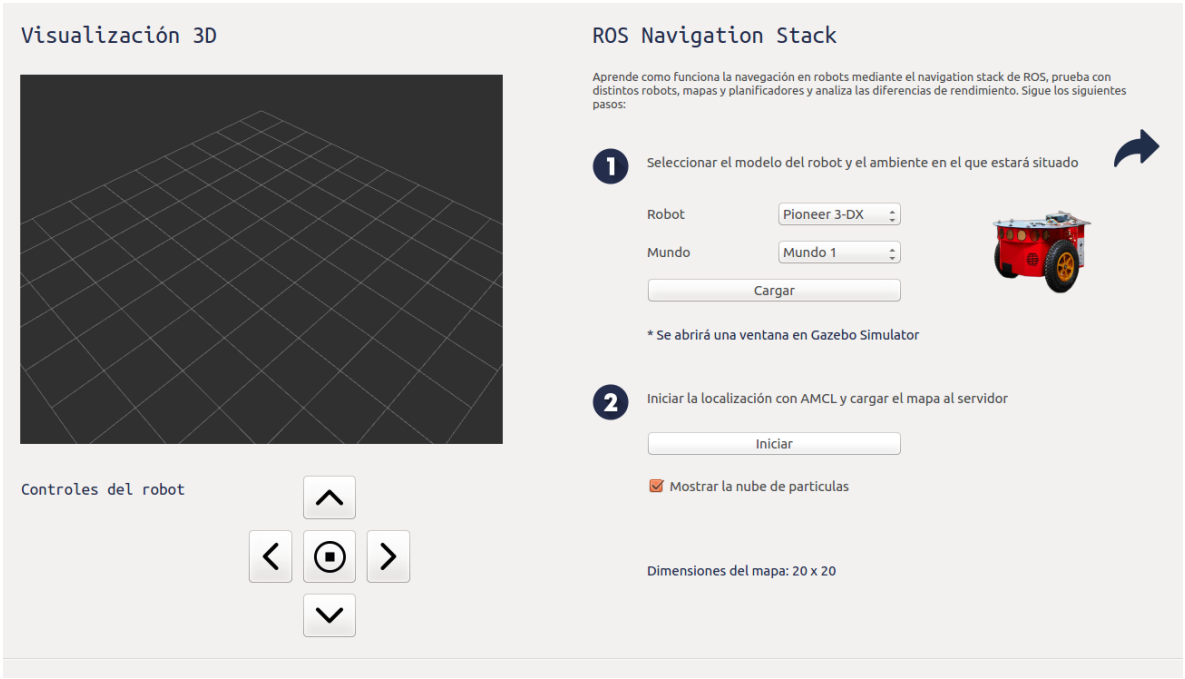

Figura 4.21 Ventana principal de AMR NAV, parte 1

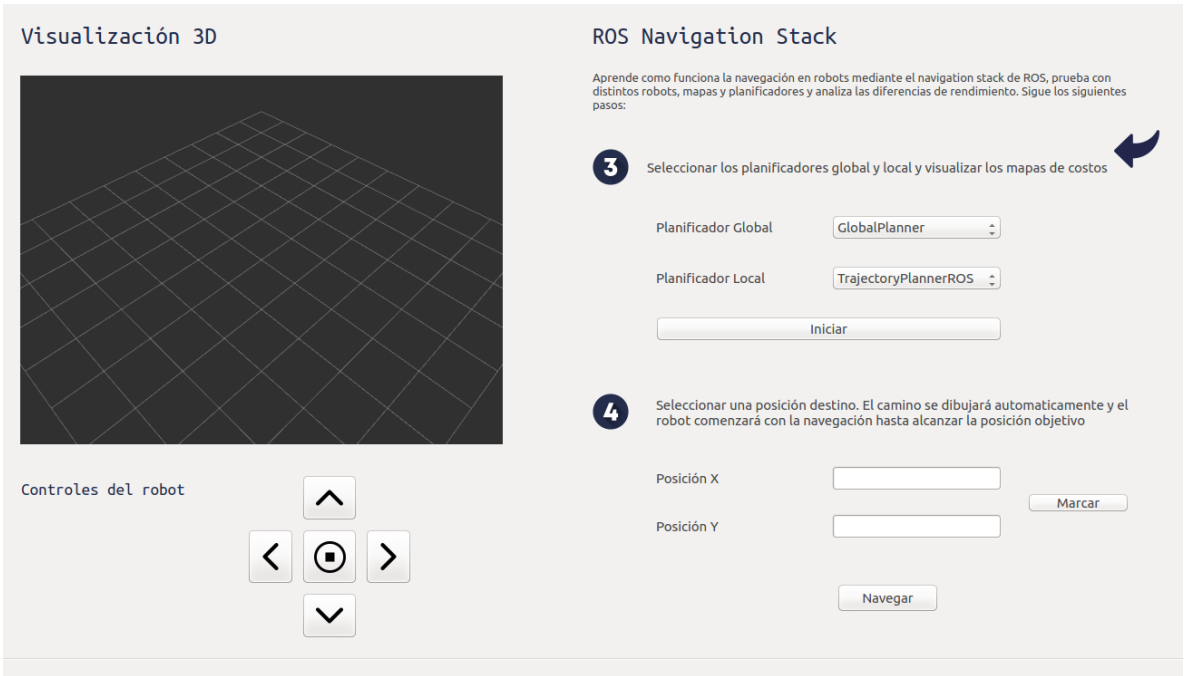

Figura 4.22 Ventana principal de AMR NAV, parte 2

Capítulo 5

5 Resultados

En este capítulo se presentan los resultados obtenidos al utilizar AMR NAV para ejecutar una pila de navegación en un robot, las características que se buscan comparar con respecto al modo tradicional son principalmente el tiempo para completar los pasos y la facilidad de estos. También, se probará la efectividad en la obtención del camino desde un punto inicial y hasta un punto final, y si los comandos de velocidad generados logran completar el recorrido.

5.1 Robot 1: TurtleBot

TurtleBot es un kit de robot personal de bajo costo con software de código abierto. Fue creado en Willow Garage por Melonee Wise y Tully Foote en noviembre de 2010. El kit TurtleBot consta de una base móvil, un sensor 3D, una computadora portátil y el kit de hardware de montaje TurtleBot. La Figura 5.1 muestra el modelo de este robot en el Gazebo. AMR NAV pone a disposición el uso de este robot para la aplicación de la pila robótica.

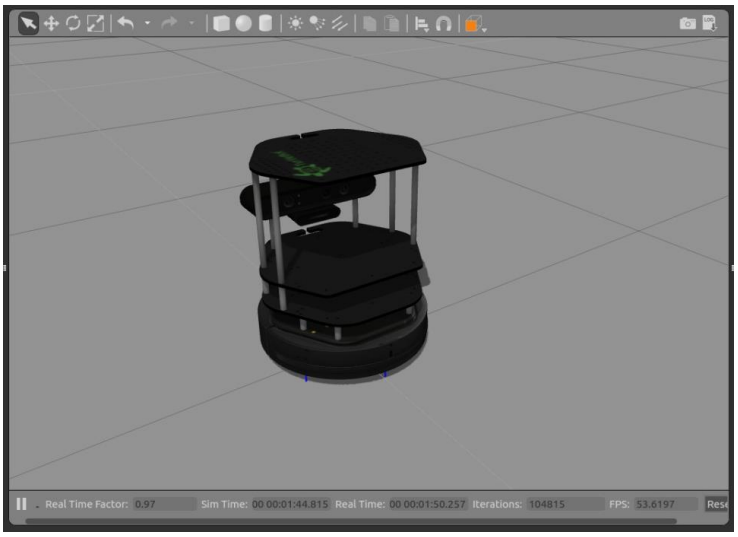

Figura 5.1 Modelo del robot TurtleBot en Gazebo

Para este robot se diseñaron 4 casos de prueba con distintas configuraciones, las cuales se describen a continuación.

5.1.1 Caso de prueba 1

En este primer caso, se ocuparon las configuraciones mostradas en la Tabla 5.1, estas configuraciones se deben acoplar al modelo del robot y mapa escogido para realizar la tarea de navegación.

Tabla 5.1 Resumen caso de prueba 1 usando el robot TurtleBot

Caso 1	
Modelo del robot	TurtleBot
Mapa	Número 2: Sencillo con algunos obstáculos dispersos
Planificador global	NavfnROS
Planificador local	DWAPlannerROS

Estas configuraciones son fácilmente seleccionadas en AMR NAV, lo que agiliza el proceso completo. El mapa número 2 seleccionado se muestra en la Figura 5.2, su tamaño es de 10x10 celdas.

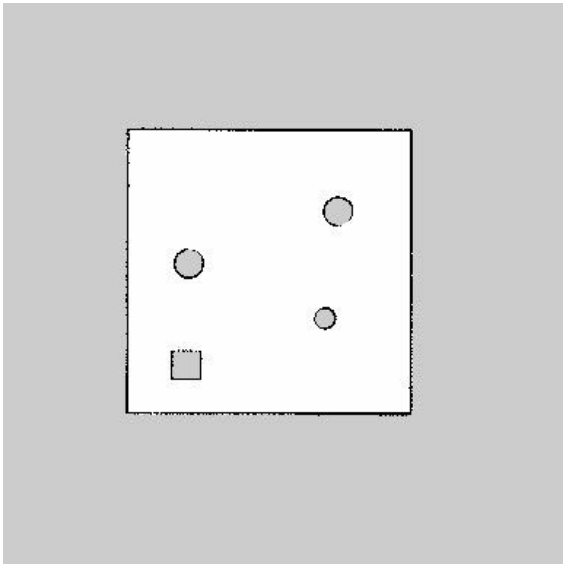

Figura 5.2 Mapa número 2 disponible en AMR NAV

Ahora bien, para comprobar el correcto funcionamiento de la pila de navegación en este caso de prueba, se eligieron tres configuraciones para la generación del camino correspondiente. Además, se comprueba que, el camino generado conecte los puntos señalados y que el robot de hecho siga este camino y llegue a la meta.

Configuración 1

Tabla 5.2 Configuración 1 para el caso de prueba 1 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	Si
(0,0)	(4,4)		

En esta configuración se generó el camino para conectar las configuraciones dadas y los comandos hacia la base móvil dieron como resultado el posicionamiento del robot en el punto meta señalado. La Figura 5.3 muestra este resultado utilizando AMR NAV.

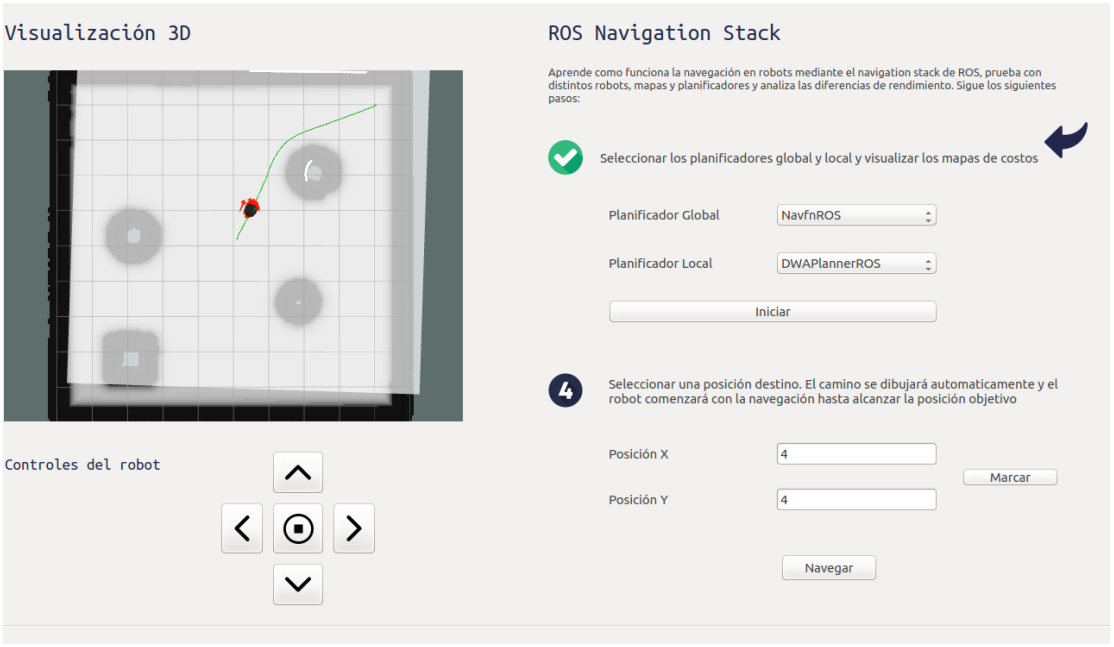

Figura 5.3 Resultados obtenidos en AMR NAV (1)

Configuración 2

Tabla 5.3 Configuración 1 para el caso de prueba 1 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	Si
(4,4)	(-2,-3)		

Partiendo del punto al que llegó el robot siguiendo la configuración anterior, se seleccionó otro punto en el mapa para continuar con el proceso de la navegación. Los resultados fueron exitosos, la Figura 5.4 muestra lo anterior.

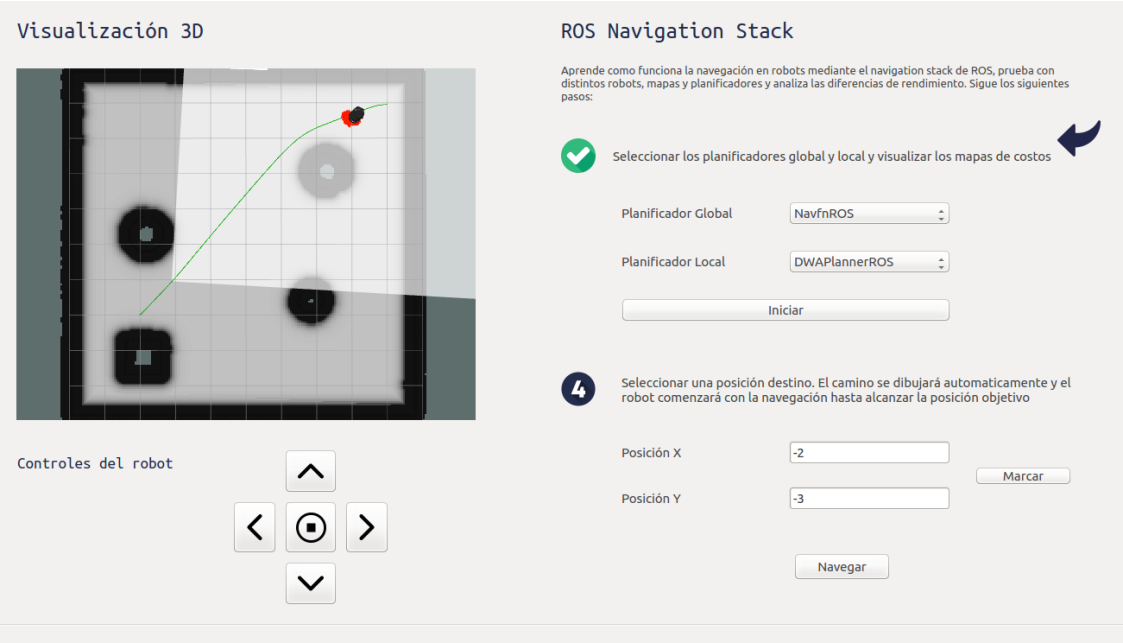

Figura 5.4 Resultados obtenidos en AMR NAV (2)

Configuración 3

Tabla 5.4 Configuración 3 para el caso de prueba 1 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	Si
(-2, -3)	(2,-3)		

Al igual que en la configuración 2, se parte del último punto alcanzado y hacia una nueva meta. El camino se generó satisfactoriamente y el robot llegó al objetivo. La Figura 5.5 muestra estos resultados.

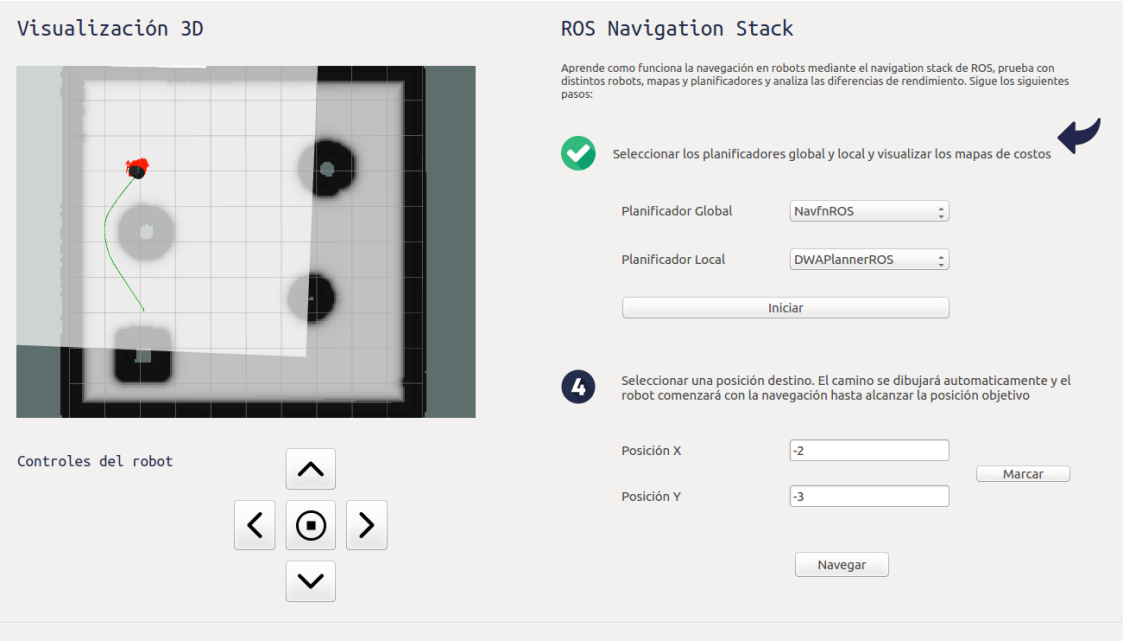

Figura 5.5 Resultados obtenidos en AMR NAV (3)

5.1.2 Caso de prueba 2

En este segundo caso de prueba se sigue ocupando el mismo mapa, pero los planificadores son reemplazados por otros disponibles en AMR NAV. La Tabla 5.5 muestra los detalles del caso de prueba.

Tabla 5.5 Resumen caso de prueba 2 usando el robot TurtleBot

Caso 2	
Modelo del robot	TurtleBot
Mapa	Número 2: Sencillo con algunos obstáculos dispersos
Planificador global	GlobalPlanner
Planificador local	TrajectoryPlannerROS

Al igual que en el caso de prueba 1, se tienen tres configuraciones para la generación de un camino y su seguimiento. Dado que el mapa y el robot no cambian, vale la pena utilizar

las mismas configuraciones presentadas en el caso anterior, esto con el objetivo de observar cómo afecta la elección de planificadores al momento de realizar la navegación en un robot.

Configuración 1

Tabla 5.6 Configuración 1 para el caso de prueba 2 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	Si
(0,0)	(4,4)		

En apariencia los resultados fueron los mismos, incluso el camino generado es bastante similar al obtenido en el caso y configuración anterior. En este punto no se encuentran diferencias significativas entre los planificadores seleccionados.

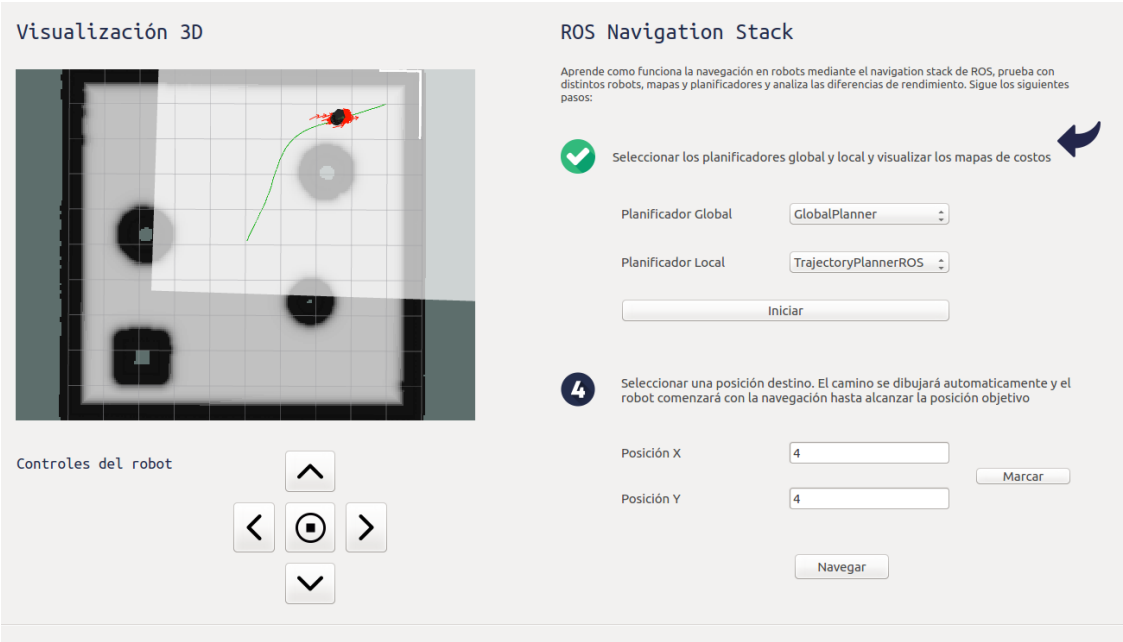

Figura 5.6 Resultados obtenidos en AMR NAV (5)

Configuración 2

Tabla 5.7 Configuración 2 para el caso de prueba 2 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	Si
(4,4)	(-2,-3)		

Los resultados fueron satisfactorios y el camino generado es similar al construido en el caso de prueba 1 con otros planificadores. La Figura 5.7 muestra al robot en camino al punto objetivo siguiendo la ruta trazada.

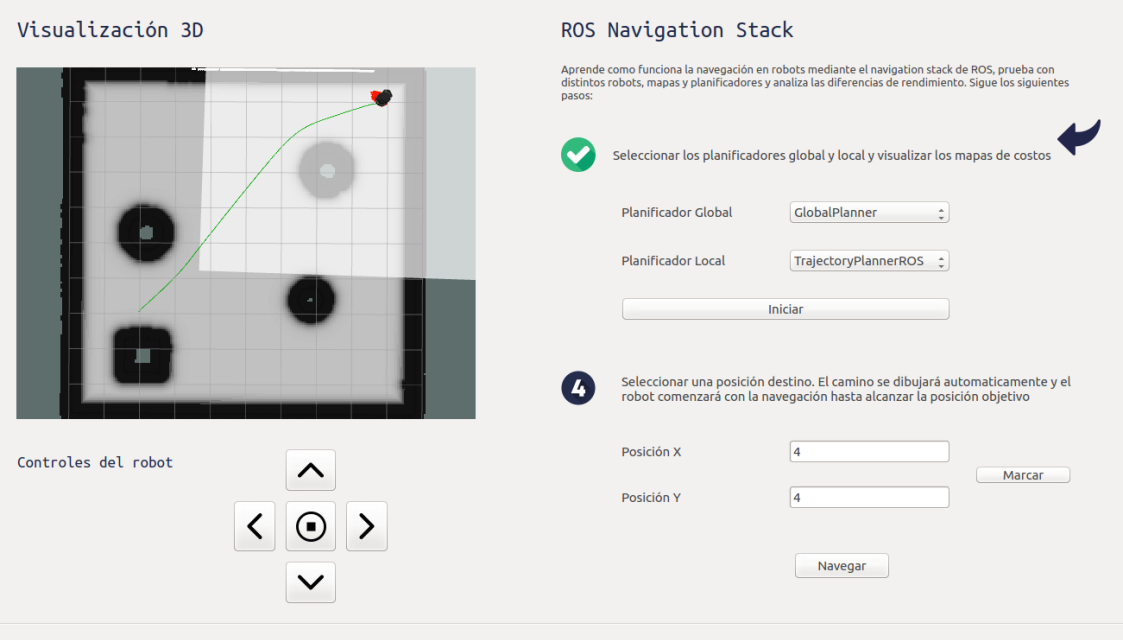

Figura 5.7 Resultados obtenidos en AMR NAV (6)

Configuración 3

Tabla 5.8 Configuración 3 para el caso de prueba 2 usando el robot TurtleBot

Configuraciones		¿Se generó un camino que conecte ambos puntos?	¿El robot llegó al punto objetivo?
Inicial	Final	Si	No
(-2, -3)	(2,-3)		

En esta tercera configuración se nota un cambio significativo al modificar los planificadores, pues a pesar de que se genera un camino que conecta ambos puntos, en medio de su recorrido hacia el punto meta, el robot colisiona con el obstáculo a su derecha y detiene su movimiento, evitando así que complete la navegación. Este tipo de problemas son comúnmente falla del planificador local. La Figura 5.8 muestra el resultado antes descrito.

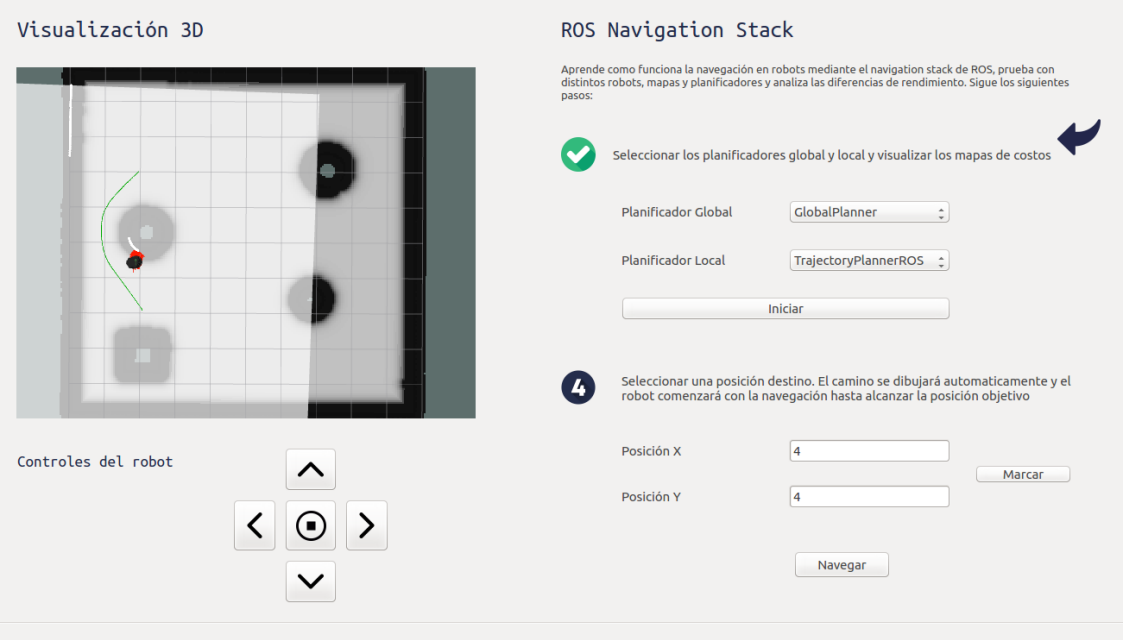

Figura 5.8 Resultados obtenidos en AMR NAV (7)

5.1.3 Caso de prueba 3

En el tercer caso de prueba, el mapa seleccionado es un tanto más complejo que el anterior pues presenta una división por cuartos, en los cuales hay diversos obstáculos de diferentes formas, ver Figura 5.9. Este mapa tiene un tamaño de 20x20 celdas, es decir, es el doble de grande que el mapa utilizado en el caso de prueba 1. La Tabla 5.9 se muestra un resumen de las configuraciones seleccionadas en AMR NAV.

Tabla 5.9 Resumen caso de prueba 3 usando el robot TurtleBot

Caso 3	
Modelo del robot	TurtleBot
Mapa	Número 5: Complejo con diversos obstáculos y divisiones
Planificador global	NavfnROS
Planificador local	DWAPlannerROS

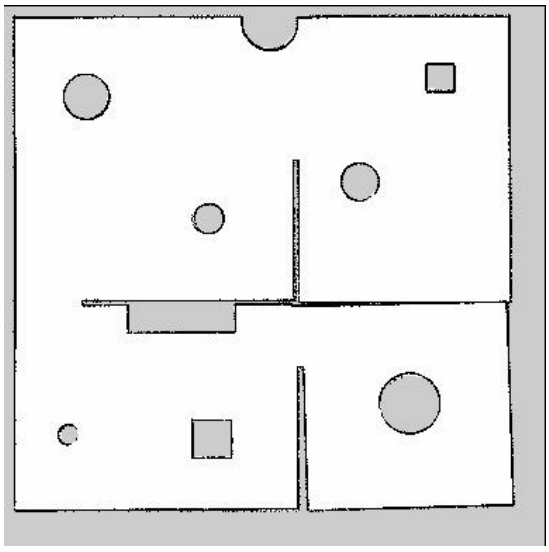

Figura 5.9 Mapa número 5 disponible en AMR NAV

En total, se utilizaron tres diferentes configuraciones de las posiciones inicial y final para la generación de un camino, las cuales se detallan en seguida.

Configuración 1

Tabla 5.10 Configuración 1 para el caso de prueba 3 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	Si
(8,-8)	(7,1)		

El resultado de esta configuración fue satisfactorio y se alcanzó la configuración final señalada. La Figura 5.10 muestra el resultado obtenido en AMR NAV. Como se observa, el robot TurtleBot se encuentra avanzando siguiendo el camino dibujado en el mapa.

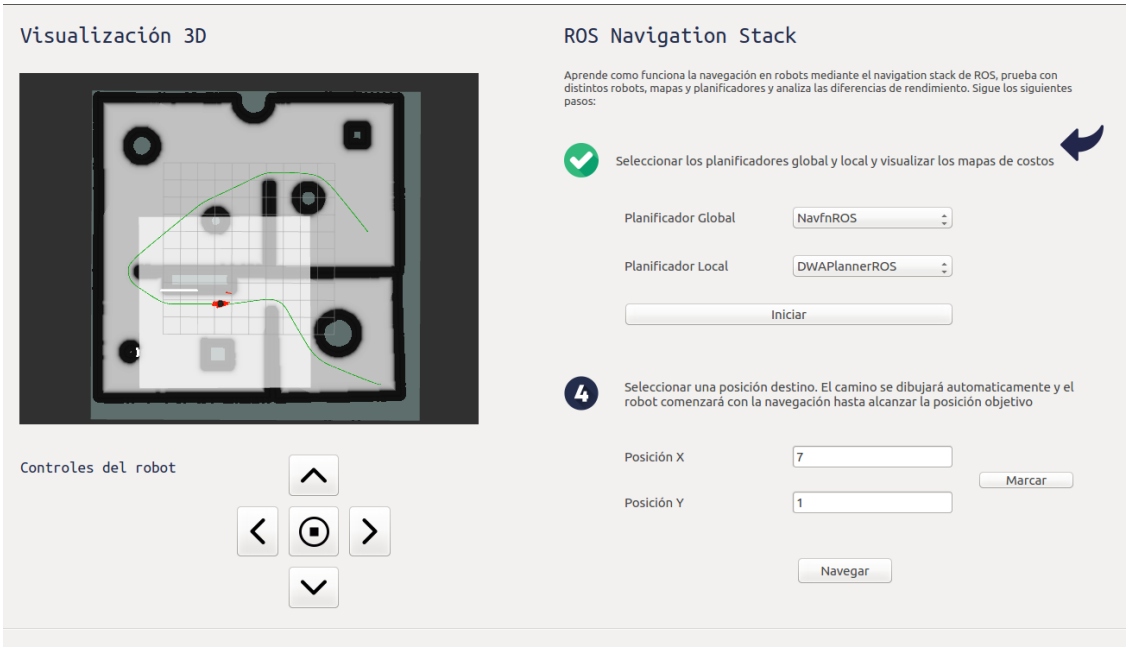

Figura 5.10 Resultados obtenidos en AMR NAV (8)

Configuración 2

Tabla 5.11 Configuración 2 para el caso de prueba 3 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	Si
(7,1)	(0,0)		

Partiendo del punto alcanzado en la configuración 1, se definió una nueva meta justo al centro del mapa. La Figura 5.11 muestra que de hecho el camino se generó y el robot llegó a la posición indicada.

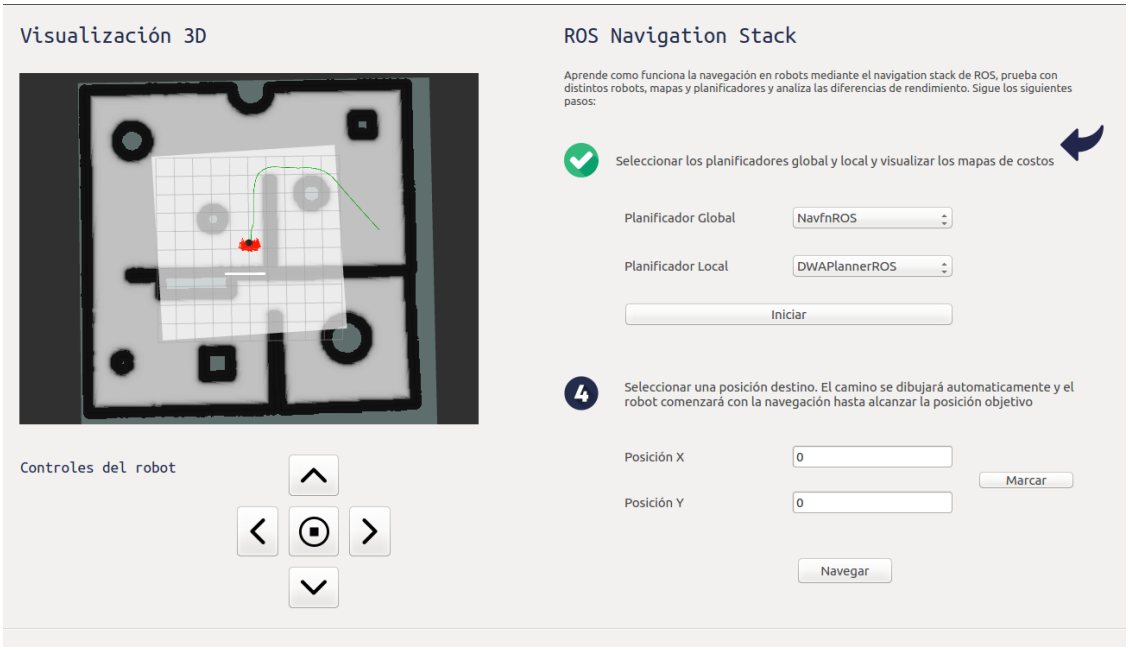

Figura 5.11Resultados obtenidos en AMR NAV (9)

Configuración 3

Tabla 5.12 Configuración 3 para el caso de prueba 3 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	No
(0,0)	(-2,-8)		

Partiendo del origen, es decir, coordenadas (0,0) en el mapa, se estableció el punto meta en (-2,-8). El camino se generó de forma satisfactoria, sin embargo, el recorrido del robot se interrumpió a medio camino dado que la lectura del sensor indicó la presencia de un obstáculo frente al robot, que de hecho no existe en el mapa. Esto indica un error en las lecturas enviadas al planificador local y que por lo tanto genera comandos de velocidad erróneos. La Figura 5.12 muestra el punto exacto en el que el robot se detuvo.

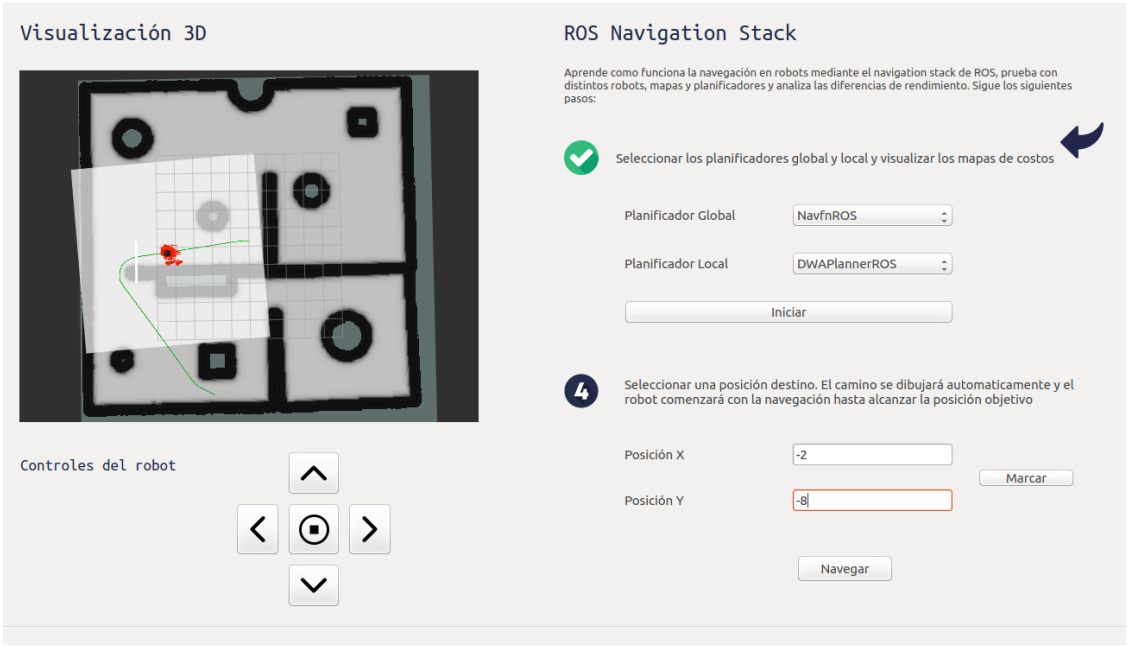

Figura 5.12 Resultados obtenidos en AMR NAV (10)

5.1.4 Caso de prueba 4

En este último caso de prueba se cambiaron los planificadores global y local, el mapa y modelo del robot se mantienen. Los detalles de este caso se muestran en la Tabla 5.13.

Tabla 5.13 Resumen caso de prueba 4 usando el robot TurtleBot

Caso 4	
Modelo del robot	TurtleBot
Mapa	Número 5: Complejo con diversos obstáculos y divisiones
Planificador global	GlobalPlanner
Planificador local	TrajectoryPlannerROS

En el caso de las configuraciones utilizadas, se realizaron diversas pruebas, sin embargo, la mayoría de ellas falló para alcanzar la meta, por lo que sólo se presentarán dos configuraciones de las cuales se pudieron sacar algunas conclusiones.

Configuración 1

Tabla 5.14 Configuración 1 para el caso de prueba 4 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	No
(8,-8)	(7,1)		

Esta configuración es igual a la configuración 1 del caso de prueba 3. Se construyó el camino de manera satisfactoria pero no se alcanzó el punto final, de hecho, sólo unos segundos después de que el robot comenzó a recibir los comandos de velocidad, este se dirigió de frente al obstáculo más cercano, por lo que es notable que la configuración por

defecto del planificador local no fue la adecuada para este caso y configuración. La Figura 5.13 muestra el inicio de la navegación del robot.

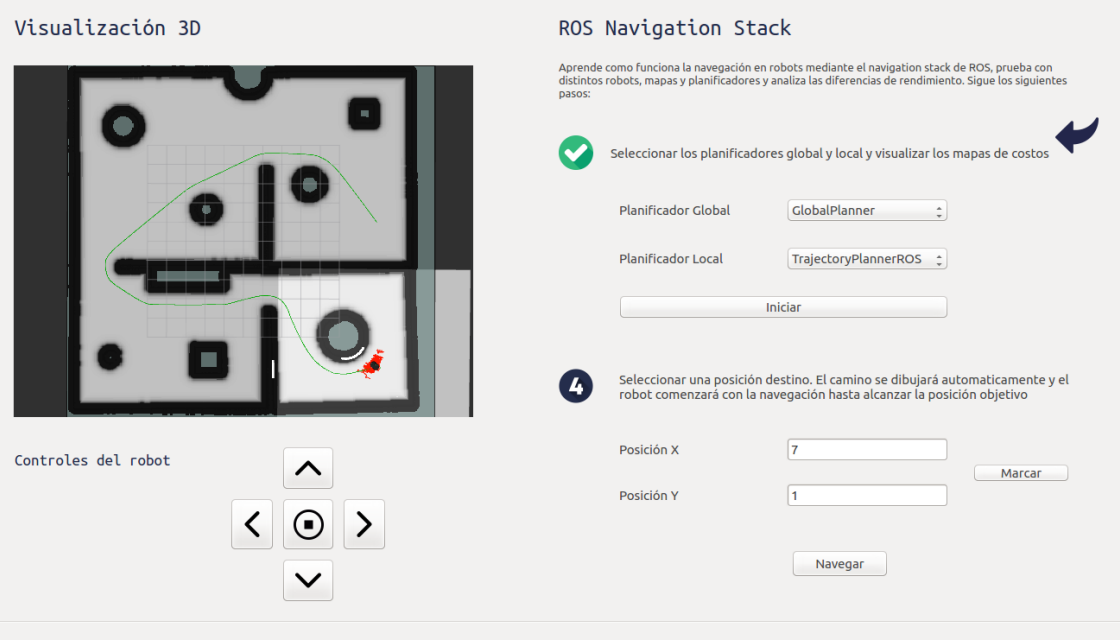

Figura 5.13 Resultados obtenidos en AMR NAV (11)

Configuración 2

Tabla 5.15 Configuración 2 para el caso de prueba 4 usando el robot TurtleBot

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	Si
(8,-8)	(7,-3)		

De entre las configuraciones probadas, se destacó que aquellas más cercanas entre sí dieron resultados positivos tanto para generar el mapa como para alcanzar la meta. Por ello, se tiene esta configuración como caso representativo. La Figura 5.14 muestra al robot en la configuración final especificada.

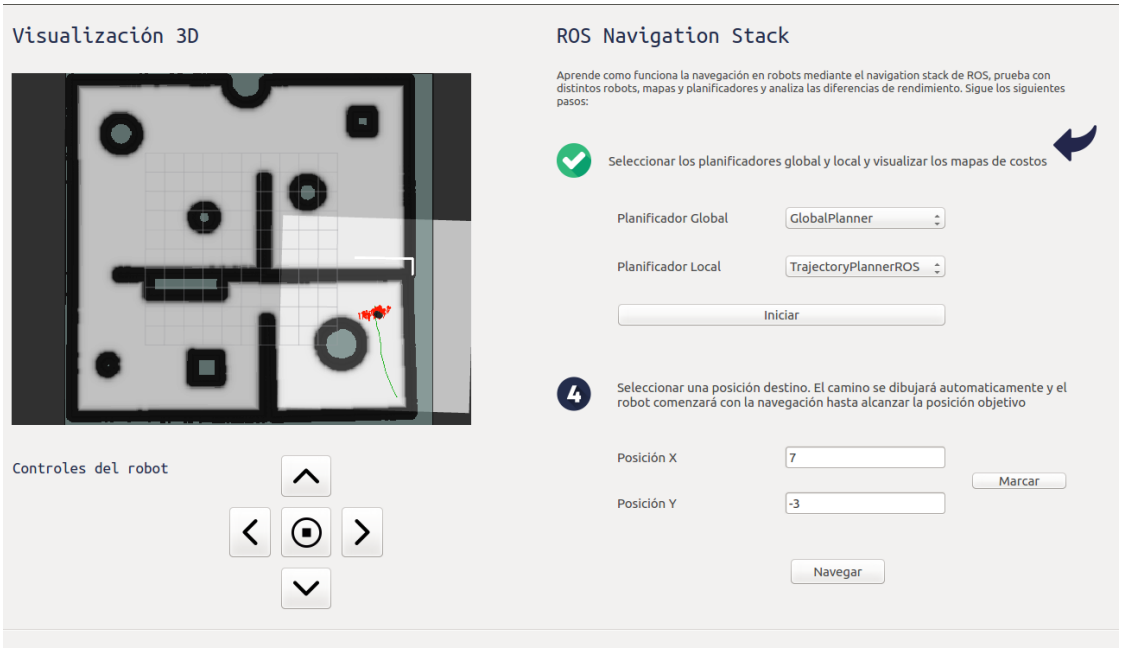

Figura 5.14 Resultados obtenidos en AMR NAV (12)

5.2 Robot 2: Pioneer 3DX

Es un robot pequeño y ligero de tipo diferencial con dos ruedas y dos motores. Este robot es de los más populares para su uso en el área educativa y de investigación en laboratorios. La Figura 5.15 muestra el modelo del Pioneer 3DX en Gazebo.

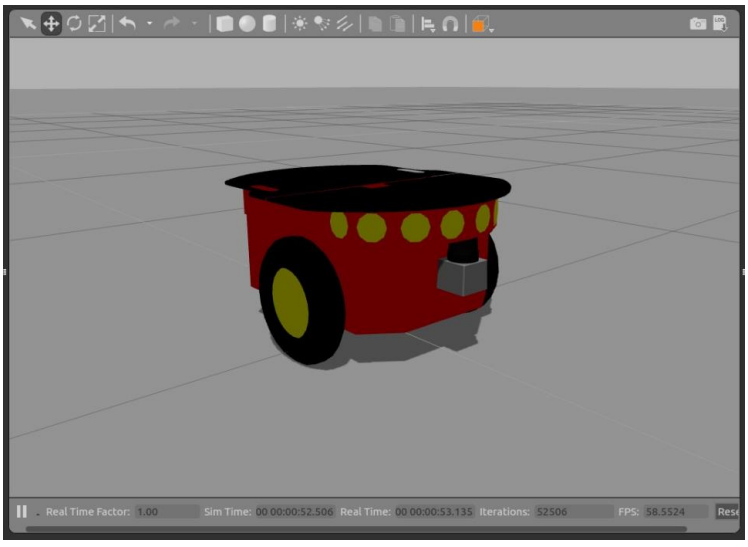

Figura 5.15 Modelo del robot Pioneer 3DX en Gazebo

Dado que AMR NAV permite estandarizar lo más posible alguna de las configuraciones básicas de la pila de navegación, es interesante ver cómo estas configuraciones pueden interferir en los resultados al utilizar un robot diferente. Por ello, se crearon dos casos de prueba que permitan ilustrar las diferencias e identificar las posibles causas de una falla.

5.2.1 Caso de prueba 1

En este caso de prueba se utiliza el mapa mostrado en la Figura 5.2. Los detalles específicos del caso se muestran en la Tabla 5.16.

Tabla 5.16 Resumen caso de prueba 1 usando el robot Pioneer 3DX

Caso 1	
Modelo del robot	Pioneer 3DX
Mapa	Número 2: Sencillo con algunos obstáculos dispersos
Planificador global	NavfnROS
Planificador local	DWAPlannerROS

De manera general, se ocuparon dos configuraciones con respecto a los puntos iniciales y finales, se describen a continuación.

Configuración 1

Tabla 5.17 Configuración 1 para el caso de prueba 1 usando el robot Pioneer 3DX

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	Si
(0,0)	(4,4)		

Esta configuración es idéntica a la presentada en el caso de prueba 1 con el robot TurtleBot, y tal como en esos resultados, se generó una ruta que conecta ambos puntos y el robot logró llegar al objetivo señalado en el mapa. La Figura 5.16 muestra lo anterior.

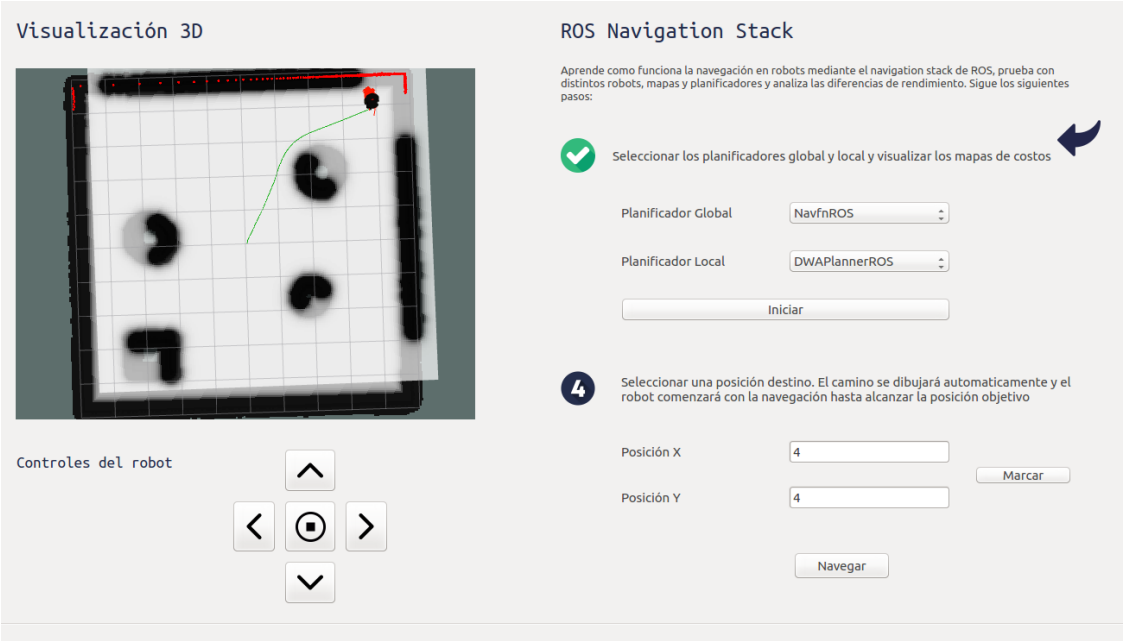

Figura 5.16 Resultados obtenidos en AMR NAV (13)

Configuración 2

Tabla 5.18 Configuración 2 para el caso de prueba 1 usando el robot Pioneer 3DX

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	No
(4,4)	(-2,-3)		

Para esta configuración, se encontró que el camino se generó de manera correcta, sin embargo, el robot no logró llegar al objetivo, ya que en medio de su recorrido y al acercarse al primer obstáculo cercano, los comandos enviados a la base móvil del robot provocaron

que esté se detuviera. La Figura 5.17 muestra al robot Pioneer 3DX en el punto exacto en el que detuvo su movimiento.

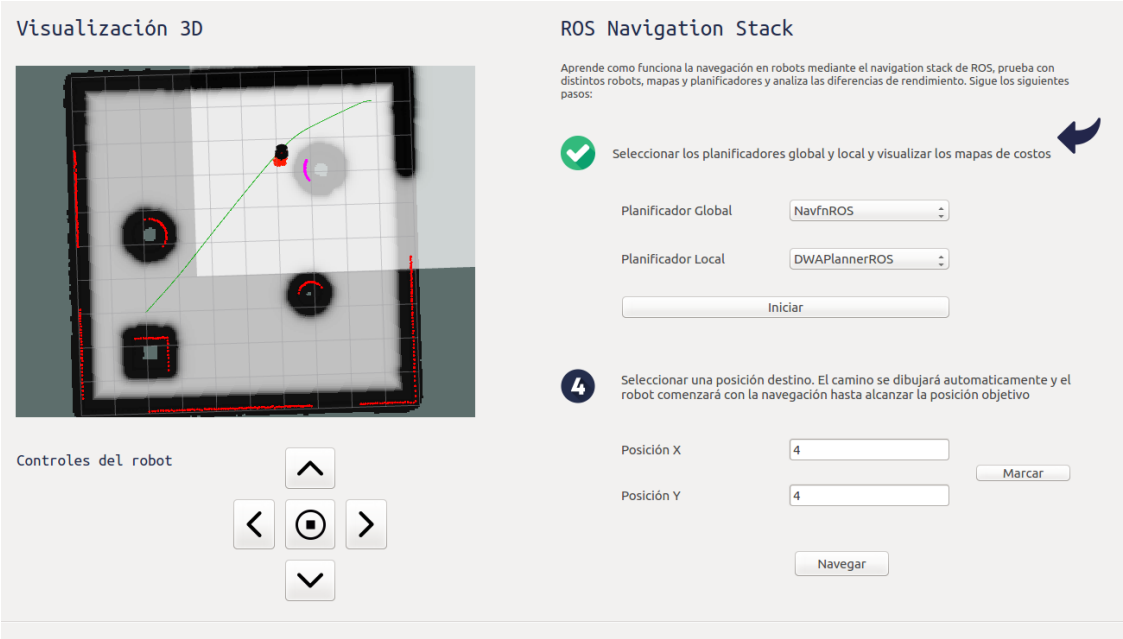

Figura 5.17 Resultados obtenidos en AMR NAV (14)

5.2.2 Caso de prueba 2

En este segundo, y último caso de prueba, se mantuvo el mapa utilizado en el caso anterior, lo que cambia son los planificadores utilizados para llevar a cabo parte de la tarea de la navegación. La Tabla 2.19 resume los parámetros utilizados.

Tabla 5.19 Resumen caso de prueba 1 usando el robot Pioneer 3DX

Caso 2	
Modelo del robot	Pioneer 3DX
Mapa	Número 2: Sencillo con algunos obstáculos dispersos
Planificador global	GlobalPlanner
Planificador local	TrajectoryPlannerROS

Del mismo modo, se probaron diversas configuraciones, sin embargo, todas resultaron con problemas para que el robot llegará al punto final de la ruta trazada, es por ello que, en este apartado, se presenta solo una configuración pues es la representativa de la mayoría de los casos.

Configuración 1

Tabla 5.20 Configuración 1 para el caso de prueba 2 usando el robot Pioneer 3DX

Configuraciones		**¿Se generó un camino que conecte ambos puntos?**	**¿El robot llegó al punto objetivo?**
Inicial	Final	Si	No
(0,0)	(4,4)		

Como se menciona en el párrafo anterior, en esta configuración se logró generar un camino desde un punto inicial y uno final, sin embargo, el robot jamás completo el recorrido del camino, de hecho, tan solo iniciar, el robot comenzó a girar en 360 grados y se mantuvo así por unos segundos antes de informar que no se pudo alcanzar la meta. La Figura 5.18 muestra el resultado obtenido a través de AMR NAV.

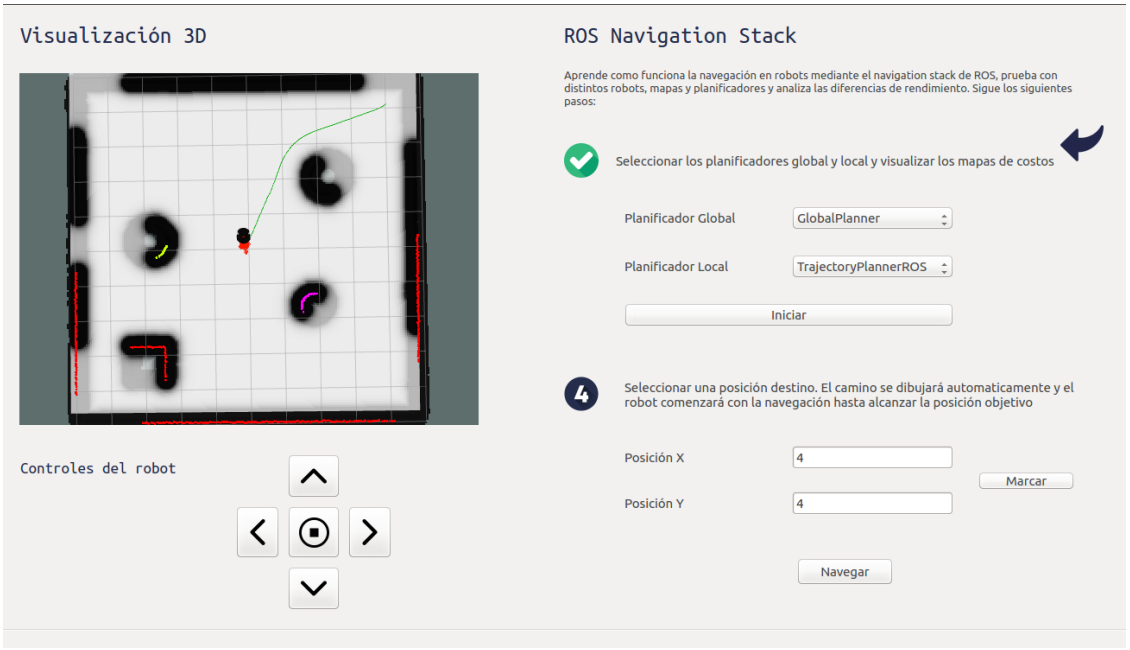

Figura 5.18 Resultados obtenidos en AMR NAV (15)

Los resultados obtenidos sugieren que, al menos para los dos modelos de robots utilizados, las configuraciones incorporadas en AMR NAV permiten ejecutar una pila de navegación sin mayor problema. Esto refuerza la viabilidad de la propuesta presentada, pues en efecto, logró simplificar y generalizar parámetros básicos requeridos por los distintos nodos ejecutados.

De lo anterior, también se demuestra que el planificador global NavfnROS y el planificador local DWAPlannerROS tienen un mejor rendimiento comparado con los otros planificadores utilizados. Es importante señalar que las configuraciones en los mapas de costos pudieron haber interferido notablemente en el desempeño del planificador local, lo cual provocó errores en algunos casos. Dado que en todos los escenarios de prueba se logró trazar la ruta desde un punto de inicio y hasta un punto final, se infiere que las configuraciones predefinidas para los planificadores globales dieron resultado, al menos en los dos robots

utilizados. No así las configuraciones del planificador local, por lo que podría ser necesario abstraer los parámetros de mayor peso y solicitarlos al usuario, de esta manera se tendría que agregar un apartado en AMR NAV que permita configurar los parámetros más importantes de los planificadores locales.

Los resultados son de hecho favorables, pues comprueban que AMR NAV logró centralizar y extender los procesos necesarios para utilizar una pila de navegación. Además, el hecho de contar con los modelos de los robots, mapas y archivos de configuración para su uso inmediato, logran reducir notablemente el esfuerzo y tiempos que serían requeridos de hacerlo de forma tradicional.

Capítulo 6

6 Conclusiones y trabajo futuro

A continuación, se describen las conclusiones obtenidas a lo largo de este proceso, así como algunos posibles trabajos futuros tomando como base esta tesis.

6.1 Conclusiones

En este trabajo se propuso un framework para la navegación de robots autónomos basado en ROS, considerando que el proceso tradicional puede ser complicado de lograr, en especial para aquellos que son nuevos en el uso del Sistema Operativo Robótico. Y es que, a pesar de contar con una comunidad con suficientes miembros, la verdad es que muchos de los foros de discusión para aclarar dudas se encuentran con temas o preguntas sin resolver.

Como resultado se desarrolló AMR NAV que integra una pila de navegación basada en mapas que ROS provee como parte de sus paquetes y librerías. A decir verdad, el tiempo que se invirtió en completar las actividades que requiere el navigation stack, según el tutorial disponible en la página oficial de ROS, fue de varias semanas, inclusos meses, pues como ya se comentó, muchas dudas no pudieron ser resueltas mediante la investigación en foros de la comunidad, por lo que se optó por aprender a través de prueba y error para resolver mediante la práctica. Después de considerarlo, es un hecho que la pila de navegación en si misma ya

ahorra muchas otras actividades, sin embargo, aún fue complicado llegar al resultado deseado.

De este modo, AMR NAV tomó todas esas horas de investigación y práctica y las convirtió en un proceso de sólo 4 pasos para establecer el sistema de navegación en un robot. En contraste con la forma tradicional, permitió solucionar este problema en menos de 5 minutos.

Finalmente, y aunque no fue liberado para su uso y prueba, se espera que este trabajo signifique una buena contribución para todos aquellos han tenido dificultades para utilizar la pila de navegación, al igual que muchos otros.

6.2 Trabajo futuro

En este punto, AMR NAV es un prototipo que puede y debe ser mejorado mediante la contribución de la comunidad. A continuación, se presentan algunos posibles trabajos futuros que pueden desarrollarse a partir de la investigación y trabajo realizados, o que, por exceder el alcance de esta tesis no han sido considerados en primera instancia, sean:

- Ampliar el número de modelos y mundos disponibles actualmente.
- Agregar algoritmos propios para los planificadores mediante la creación de plugins compatibles con ROS, de este modo, la lista actual no debería ser limitada, pudiendo ser agregados otros tipos de algoritmos comúnmente utilizados en la literatura como RTT.
- Examinar otras formas de localización para obtener un mejor rendimiento en los cálculos realizados por el planificador local.

- Implementar la navegación en robots sin tener el conocimiento del mapa con anticipación, claro que esto incluiría investigación más especializada sobre SLAM.
- Explorar otras áreas de la robótica e incluirlas en este framework como una extensión del propósito original.

BIBLIOGRAFÍA

[1] Ibarra Bonilla, M. N. (2009). *Navegación autónoma de un robot con técnicas de localización y ruteo.* [Tesis de Maestría, Instituto Nacional de Astrofísica, Óptica y Electrónica]. Repositorio Institucional del Instituto Nacional de Astrofísica, Óptica y Electrónica. http://inaoe.repositorioinstitucional.mx/jspui/handle/1009/394

[2] Andersen, J. C. (2007). *Mobile Robot Navigation.*

[3] Muñoz Martínez, V. F. (1997). *Planificación de trayectorias para robots móviles*. Málaga, España: Departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga. ISBN 84-8498-970-4

[4] Giralt, G., Sobek, R., Chatila, R. (1979). *A Multi-Level Planning and Navigation System for a Mobile Robot: A First Approach to Hilare*. 6th International Joint Conference on Artificial Intelligence, Vol. 1, Morgan Kaufmann Publishers Inc.

[5] Durrant-Whyte, H.F. (1994). *Inertial navigation systems for mobile robots.* IEEE Trans. Robot. Autom, 21, 11–16.

[6] Thrun, S., Burgard, W., y Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

[7] Blume, H., Heimann, B. *A Laser Range Scanner Simulation for Probabilistic Object Tracking*. (2007). ICRA 2007 Workshop: Planning, Perception and Navigation for Intelligent Vehicles.

[8] Drumheller, M. (1987). *Mobile robot localization using sonar*. IEEE Trans. Pattern Anal. Mach. Intell, 2, 325–332.

[9] Leonard, J.J., Durrant-Whyte, H. F. (1991). *Mobile robot localization by tracking geometric beacons.* IEEE Trans. Rob. Autom., 7, 376–382.

[10] Rekleitis, I. M. (2004). *A Particle Filter Tutorial for Mobile Robot Localization.* Centre for Intelligent Machines, Technical Report. McGill University.

[11] Dellaert, F., Fox, D., Burgard W., Thrun, S. (1999). *Monte Carlo Localization for Mobile Robots.* Proc. IEEE International Conference on Robotics and Automation.

[12] Fox, D., Burgard, W., Dellaert, F., Thrun, S. (1999). *Monte Carlo Localization: Efficient Position Estimation for Mobile Robots.* Proc. Sixteenth National Conference on Artificial Intelligence. John Wiley & Sons Ltd.

[13] Kitagawa, G., Comput, J. (1996). *Monte-Carlo filter and smoother for non-Gaussian nonlinear state space models.* Graph Stat, 5, 1–25.

[14] Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H., Csorba, M. (2001). A *Solution to the Simultaneous Localization and Map Building (SLAM) Problem.* IEEE Trans. Rob. Autom., 17, 229–241.

[15] Dellaert, F., Kaess, M. (2006). *Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing.* Int. J. Rob. Res. 25(12), 1181–1203.

[16] Kummerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W. (2011). *A General Framework for Graph Optimization.* In Proc. IEEE International Conference on Robotics and Automation (ICRA), 3607–3613.

[17] Shoudong, H., Gamini, D. (2016). *Robot Localization: An Introduction.* 10.1002/047134608X.W8318.

[18] Lavalle, S. M. (2011). *Motion planning: Part I: The essentials.* IEEE Robotics and Automation Magazine, 18(1), 79-89. [5751929]. https://doi.org/10.1109/MRA.2011.940276

[19] Koubaa, A., Bennaceur, H., Chaari, I., Trigui, S., Ammar, A., Sriti, M., Alajlan, M., Cheikhrouhou, O., Javed, Y. (2018). *Introduction to Mobile Robot Path Planning.* 10.1007/978-3-319-77042-0_1.

[20] Dijkstra, E. (1959). *A note on two problems in connexion with graphs.* Numer. Math. 1, 269–271.

[21] Gbadamosi, O.A., Aremu, D.R. (2020). *Design of a Modified Dijkstra's Algorithm for finding alternate routes for shortest-path problems with huge costs.* In Proceedings of the 2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS), 1–6.

[22] Syed Abdullah, F., Iyal, S., Makhtar, M., Jamal, A.A. (2015). *Robotic Indoor Path Planning Using Dijkstra's Algorithm with Multi-Layer Dictionaries.* In Proceedings of the 2015 2nd International Conference on Information Science and Security (ICISS), Seoul, Korea.

[23] Kang, H., Lee, B.h., Kim, K. (2009). *Path Planning Algorithm Using the Particle Swarm Optimization and the Improved Dijkstra Algorithm.* In Proceedings of the 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, Wuhan, China, 18, 1002–1004.

[24] Gass, S.I., Harris, C.M. (2001). *Near-optimal solution. In Encyclopedia of Operations Research and Management Science*. Springer: New York, NY, 555.

[25] Ferguson, D., Likhachev, M., Stentz, A. (2005). *A Guide to Heuristic-based Path Planning.* In Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems. International Conference on Automated Planning and Scheduling (ICAPS), Monterey, CA, 9–18.

[26] Cheng, L., Liu, C., Yan, B. (2014). *Improved hierarchical A-star algorithm for optimal parking path planning of the large parking lot.* In Proceedings of the 2014 IEEE International Conference on Information and Automation (ICIA), Hailar, Hulun Buir, China, 695–698.

[27] Goto, T., Kosaka, T., Noborio, H. (2003). *On the heuristics of A* or A algorithm in ITS and robot path-planning.* In Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453), Las Vegas, NV, 2, 1159–1166.

[28] Sedighi, S., Nguyen, D.V., Kuhnert, K.D. (2019). *Guided Hybrid A-star Path Planning Algorithm for Valet Parking Applications.* In Proceedings of the 2019 5th International Conference on Control, Automation and Robotics (ICCAR), Beijing, China, 570–575.

[29] Yijing, W., Zhengxuan, L., Zhiqiang, Z., Zheng, L. (2018). *Local Path Planning of Autonomous Vehicles Based on A* Algorithm with Equal-Step Sampling.* In Proceedings of the 2018 37th Chinese Control Conference (CCC), Wuhan, China, 7828–7833.

[30] Koenig, S., Likhachev, M., Furcy, D. (2004). *Lifelong Planning A*.* Artif. Intell., 155, 93–146.

[31] Xie, K., Qiang, J., Yang, H. (Consultado en agosto 2021*). Research and Optimization of D-Start Lite Algorithm in Track Planning.* IEEE 8, 161920–161928.

[32] Stentz, A. (2003). *Optimal and Efficient Path Planning for Unknown and Dynamic Environments;* The Robotics Instilute Carnegie Mellon University: Pittsburge, PA.

[33] Koenig, S., Likhachev, M. (2002). *D* Lite.* In Eighteenth National Conference on Artificial Intelligence. American Association for Artificial Intelligence: Menlo Park, CA, 476–483.

[34] Wang, J., Garratt, M.A., Anavatti, S.G. (2016). *Dynamic path planning algorithm for autonomous vehicles in cluttered environments.* In Proceedings of the 2016 IEEE International Conference on Mechatronics and Automation, Harbin, China, 1006–1011.

[35] Yun, S.C. (2011). *Enhanced D * Lite Algorithm for Autonomous Mobile Robot Velappa Ganapathy*. 2011. Disponible en: https://www.researchgate.net/publication/251979681_Enhanced_D_Lite_Algorithm_for_mobile_robot_navigation.

[36] Larson, R.E., Casti, J.L. (1987). *Principles of Dynamic Programming.* M. Dekker: New York, NY, USA, 2.

[37] Larson, R. (1967). *A survey of dynamic programming computational procedures.* IEEE Trans. Autom. Control, 12, 767–774. [Referencia Cruzada]

[38] Ferguson, D., Stentz, A. (2005). *Field D*: An Interpolation-Based Path Planner and Replanner*. In Robotics Research; Springer: Berlin/Heidelberg, Germany, 28, 239–253. [Referencia Cruzada]

[39] LaValle, S.M., Kuffner, J.J., Jr. (2001). *Randomized kinodynamic planning*. Int. J. Robot. Res. 20, 378–400. [Referencia Cruzada]

[40] Huang, L. (2018). *Rapidly Exploring Random Tree (RRT) and RRT*.* Disponible en: http://wolfram.com/knowledgebase/source-information (Consultado en agosto 2021).

[41] Souissi, O., Benatitallah, R., Duvivier, D., Artiba, A., Belanger, N., Feyzeau, P. (2013). *Path planning: A 2013 survey.* In Proceedings of the 2013 International Conference on Industrial Engineering and Systems Management (IESM), Rabat, Morocco, 1–8.

[42] Kuffner, J.J., LaValle, S.M. (2000). *RRT-connect: An efficient approach to single-query path planning.* In Proceedings of the 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), San Francisco, CA, 2, 995–1001.

[43] Oriolo, G., Vendittelli, M., Freda, L., Troso, G. (2004). *The SRT method: Randomized strategies for exploration.* In Proceedings of the IEEE International Conference on Robotics and Automation. Proceedings. ICRA 04, New Orleans, LA, 5, 4688–4694. [Referencia Cruzada]

[44] Yiping, Z., Jian, G., Ruilei, Z., Qingwei, C. (2014). *A SRT-based path planning algorithm in unknown complex environment.* In Proceedings of the 26th Chinese Control and Decision Conference (2014 CCDC), Changsha, China, 3857–3862.

[45] Kuner, J., Latombe, J.c., Guibas, L., Bregler, C. (2000). *Autonomous Agents for Real-Time Animation.* [Ph.D. Thesis, Stanford University]. Stanford, CA, USA.

[46] Karur, K., Sharma, N., Dharmatti, C., Siegel, J. (2021). *A Survey of Path Planning Algorithms for Mobile Robots.* Vehicles 2021, 3, 448–468. https://doi.org/10.3390/vehicles3030027

[47] Robot Operating System. (2021, 6 de agosto). Wikipedia, La enciclopedia libre. Fecha de consulta: 10:00, octubre 4, 2021 desde

https://es.wikipedia.org/w/index.php?title=Robot_Operating_System&oldid=137501462.