ISSN 1870-4069

Un framework basado en ROS para la navegación de robots móviles autónomos

Rosa-L. Villarreal-O., Abraham Sánchez-L., Martin Estrada-A., Rogelio González-V.

Benemérita Universidad Autónoma de Puebla,
Facultad de Ciencias de la Computación,
México

{rlauravillarreal, rogelio.gzzvzz}@gmail.com
{abraham.sanchez, martin.estrada}@correo.buap.mx

Resumen. 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. La autonomía de un robot móvil se basa principalmente en el sistema de navegación autónoma. La propuesta de este trabajo es un framework para la navegación autónoma de robots móviles basado en el sistema operativo robótico (ROS), lo que permitirá 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. Se presentan ejemplos prácticos simulados para validar la propuesta.

Palabras clave: Robot móvil, navegación, ROS, arquitectura, simulación.

A ROS-Based Framework for Autonomous Mobile Robot Navigation

Abstract. The need to increase autonomy in robotic applications has motivated the creation and development of mobile robots. The purpose is to limit human intervention as much as possible. The autonomy of a mobile robot is mainly based on the autonomous navigation system. The proposal of this work is a framework for autonomous navigation of mobile robots based on the robotic operating system (ROS), which allows implementing the various related tasks in a series of logical and simple steps to integrate the navigation system in a mobile robot. By building a single ecosystem, it seeks to reduce the time spent configuring the activities and requirements for navigation, while simplifying the process. Practical simulated examples are presented to validate the proposal.

Keywords: Mobile robot, navigation, ROS, architecture, simulation.

1. Introducción

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 [1, 9]. 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 [6, 9]:

- Percepción del mundo: Mediante el uso de sensores externos, creación de un mapa o modelo del entorno donde se desarrollará la tarea de navegación [2],
- 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 [6],
- 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 función, con el 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 [2].

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. La complejidad del sistema necesario para desarrollar esta tarea de navegación, depende principalmente del conocimiento que se posee del entorno de trabajo. En general, para resolver el problema de navegación de robots, se necesita encontrar respuesta a las preguntas: ¿Dónde está el robot?, ¿A dónde se dirige?, ¿Cómo llegará ahí? [9, 10]. La principal complejidad en la resolución de tareas propias de la robótica, es justamente la implementación de algoritmos para diferentes funciones que proveen en mayor medida el grado de autonomía al robot móvil.

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 [7]. 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.

Este framework pone a disposición una serie de herramientas y librerías para obtener, construir, escribir, y ejecutar código a través de múltiples computadoras [8]. Como sabemos, 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. Efectivamente hay libros que detallan aspectos de ROS, pero estos no son fáciles de entender y menos de llevar a cabo, sino se tiene experiencia en la robótica móvil, es una de nuestras aportaciones importantes.

Se presentan al final del presente trabajo, una serie de simulaciones realizadas con este framework, aclarando que no fue posible probar con el robot real, un robot Pioneer 3DX con el que cuenta el grupo de trabajo, debido a las restricciones impuestas por la pandemia de Covid19. Si bien, no se presentan resultados realizados con el robot real, ello no significa que no se realizarán; de hecho, antes de liberar el código del framework, realizaremos experimentos con nuestro robot Pionner P3DX.

2. Sistema operativo robótico

Robot Operating System (ROS), es un conjunto de herramientas informáticas en forma de software libre (el sistema está bajo licencia BSD) de código abierto, que permite el desarrollo de software para robótica. Fue desarrollado originalmente en 2007 por la empresa estadounidense Willow Garage, para su robot PR2 (Personal Robot 2) [7]. Las principales ventajas de utilizar el conjunto de herramientas y librerías que proporciona ROS se pueden resumir a continuación:

- 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 usados 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.

Para utilizar la propuesta del framework, se utilizaron algunas herramientas adicionales, las cuales se mencionan a continuación:

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.

Gmapping: Es un paquete que incluye un contenedor ROS para Gmapping de OpenSlam. Proporciona SLAM basado en laser que permite crear mapas de rejilla de ocupación 2D a partir de los datos recopilados por el láser y de pose del robot móvil.

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.

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. 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.

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.

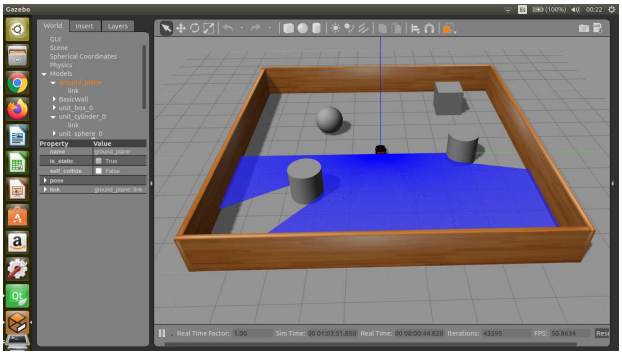

Fig. 1. Ambiente 3D construido en el simulador de Gazebo.

A continuación, se presenta la propuesta del framework, el punto central de nuestra aportación es el manejo eficiente y novedoso de la pila de navegación, que se resaltará en la sección de los experimentos realizados con el framework propuesto.

3. Propuesta del framework

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 con el 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.

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 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 cuando se usa 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 Fig. 1, 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.

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. 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, el cual se desarrolló como parte del trabajo.

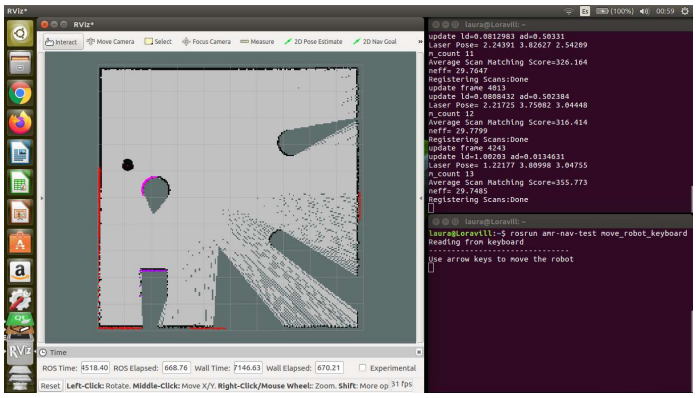

Fig. 2. Construcción de un mapa de rejillas 2D.

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 Fig. 2 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 va coloreando, las áreas grises que 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, 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.

Lo anterior genera dos archivos, uno con extensión pgm, la imagen del mapa y un archivo yaml relacionado que contiene metadatos del mapa. Una vez obtenida la imagen del mapa, los procesos que se encontraban ejecutando pueden ser detenidos, su propósito ya ha sido cumplido.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 órdenes 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, entonces se determinan cuáles serán los planificadores global y local para utilizar. Estos archivos son de extensión yaml y contienen los parámetros configurados más adecuados al robot en uso.

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.

3.1. 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 rejilla 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.

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. 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 dentro del mapa, lo que derivará en errores de cálculo para la navegación. La Fig. 3 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, 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. 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 Fig. 4 muestra la interacción de la ventana de comandos que se encuentra ejecutando el archivo launch y la ventana Rviz con sus diversos elementos.

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 Fig. 5 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.

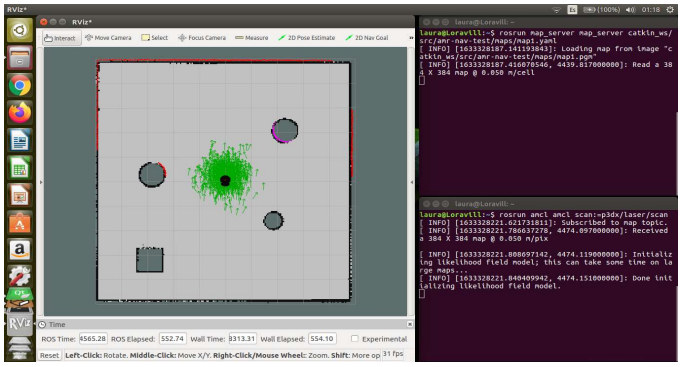

Fig. 3. Ejecución del nodo map_server y amcl.

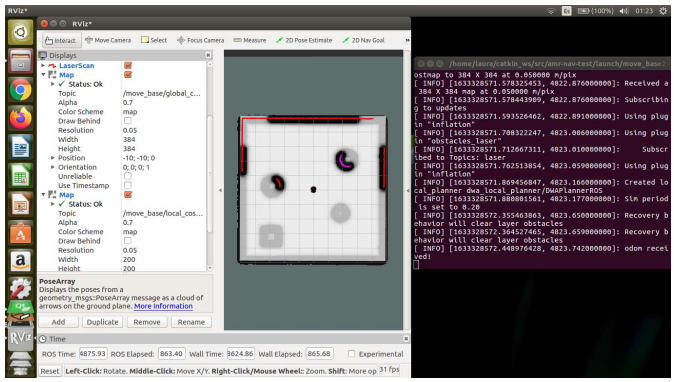

Fig. 4. Interacción del modo move_base con Rviz.

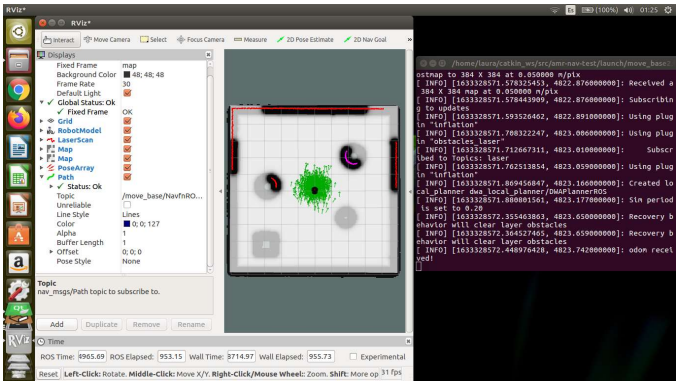

Fig. 5. Adición del elemento Path en Rviz.

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.

Lo que ocurre a continuación es la generación del camino, y el envío de los comandos de velocidad a la base móvil hasta que el robot llega a la posición meta.

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! Esto es parte de nuestra contribución.

Es por ello, que la propuesta que se presenta en este trabajo 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 él envió de distintas posiciones meta.

4. Resultados experimentales

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.

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 anteriomente y la simulación de Gazebo.

De forma general, la Fig. 6 muestra la arquitectura de alto nivel de AMR NAV y su interacción con los sistemas antes mencionados. Como se puede observar, la propuesta realizada provee un sistema para controlar los procesos detrás de cada paso para la navegación a través de una interfaz de usuario.

Las Fig. 7 y Fig. 8 muestran el entorno de la ventana principal de este framework. En la primera se muestran los dos primeros pasos del navigation stack, en la otra los dos faltantes. En esta parte 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.

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.

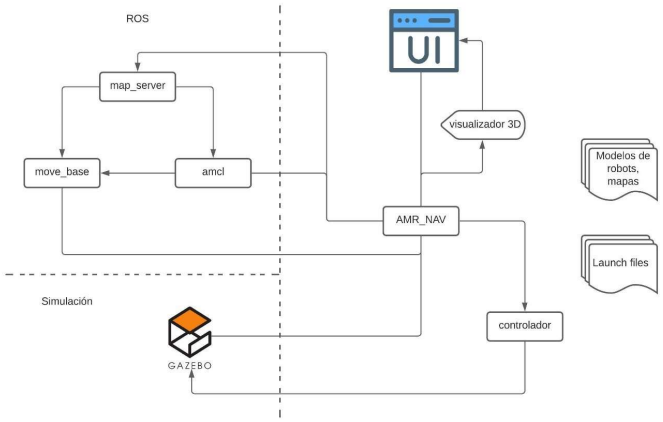

Fig. 6. Arquitectura de alto nivel de AMR NAV.

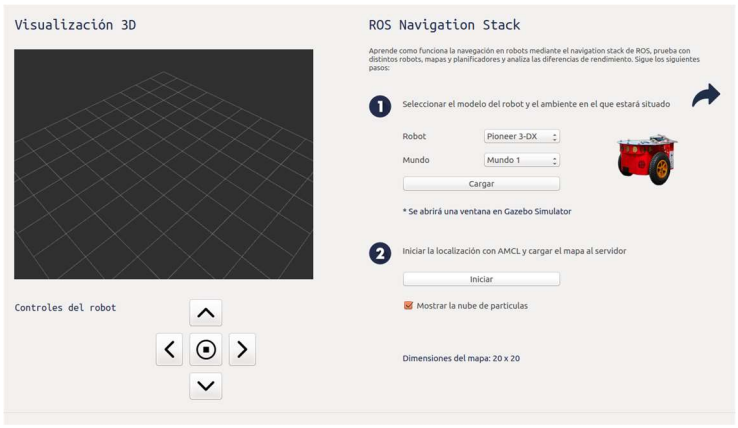

Fig. 7. Ventana principal de AMR NAV, parte 1.

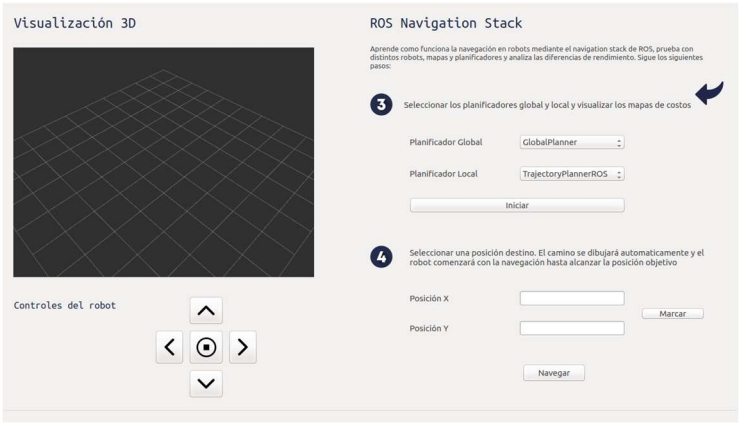

Fig. 8. Ventana principal de AMR NAV, parte 2.

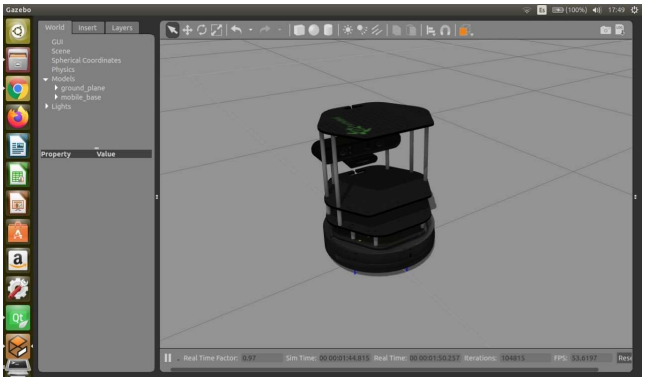

Fig. 9. Modelo del robot TurtleBot en Gazebo.

La Fig. 9 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. Para este robot se diseñaron 4 casos de prueba con distintas configuraciones, por motivos de brevedad se presentarán solo los resultados mediante imágenes. Lo importante a destacar, son los diferentes tipos de mapas utilizados, desde los fáciles (con pocos obstáculos) hasta los complicados (con obstáculos estrechos, por ejemplo); igualmente otro punto a resaltar fue el cambio de los planificadores tanto local como global, esto con la finalidad de evaluar el desempeño del robot en relación con su ambiente de trabajo.

Para los casos de prueba propuestos, usamos los siguientes planificadores local y global. NavfnROS (global) y DWAPlannerROS (local) para el primer caso de prueba. GlobalPlanner (global) y TrajectoryPlannerROS (local) para el segundo caso de prueba. NavfnROS (global) y DWAPlannerROS (local) para el tercer caso de prueba. GlobalPlanner (global) y TrajectoryPlannerROS (local) para el cuarto caso de prueba. Ver la Fig. 10 para los resultados obtenidos con dos mapas diferentes.

El robot 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 Fig. 11 muestra el modelo del 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.

Para los casos de prueba propuestos, usamos los siguientes planificadores local y global. NavfnROS (global) y DWAPlannerROS (local) para el primer caso de prueba. GlobalPlanner (global) y TrajectoryPlannerROS (local) para el segundo caso de prueba. En la Fig. 11 se muestran los resultados de esta segunda prueba con el AMR NAV.

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.

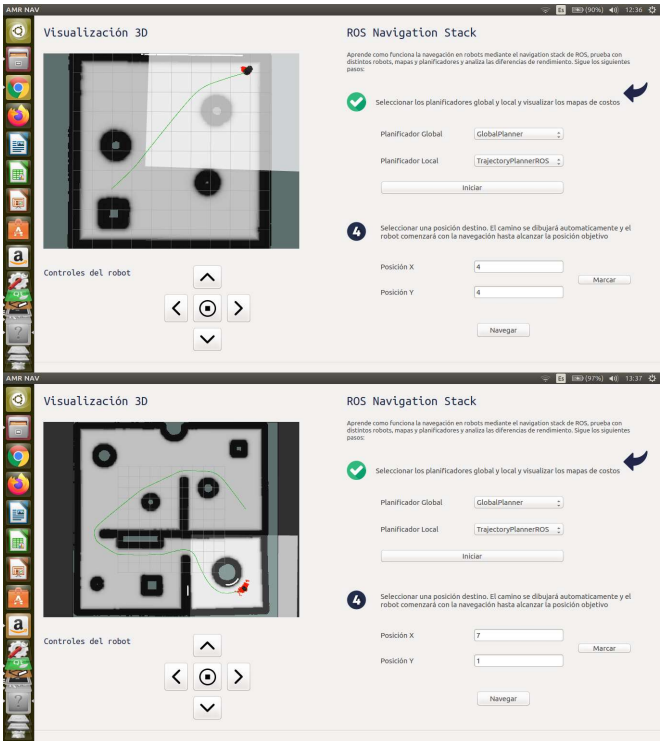

Fig. 10. Resultados obtenidos en AMR NAV con el robot TurtleBot.

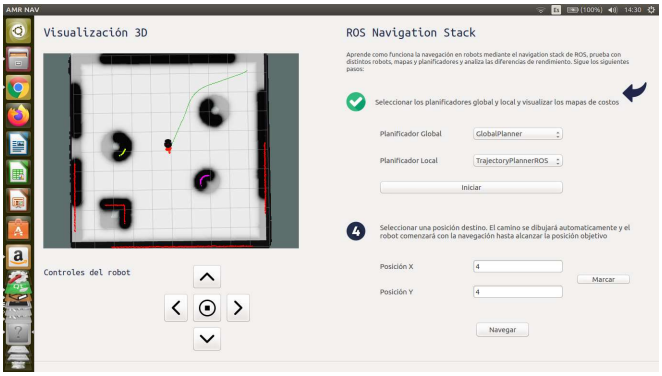

Fig. 11. Resultados obtenidos en AMR NAV con el robot Pioneer 3DX.

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.

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.

Para el desarrollo y pruebas del framework propuesto en este trabajo, se utilize una computadora PC basada en el procesador AMD Ryzen 5 2500 U, con tarjeta gráfica AMD Radeon Vega 8, utilizando el Sistema operative Linux, distribución Ubuntu 14.04. Es claro que, como parte de un siguiente trabajo, se presentarían las ejecuciones en tiempo real, utilizando un robot real. Y es ahí cuando valdrá la pena realizar un estudio comparativo más detallado.

5. Conclusiones y trabajo futuro

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.

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. Estamos en la depuración del código y otros detalles para su posterior liberación.

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.

Referencias

1. Dellaert, F., Fox, D., Birgard, W., Thrun, S.: Monte Carlo localization for mobile robots. In: Proceedings 1999 IEEE International Conference on Robotics and Automation, vol.2, pp. 1322–1328 (1999) doi: 10.1109/ROBOT.1999.772544
2. Dissanayake, G., Newman, P., Clark, S., Durrant-Whyte, H., Csorba, M.: A solution to the simultaneous localization and map building problem. IEEE Transactions on Robotics and Automation, vol. 17, no. 3, pp. 229–241 (2001) doi: 10.1109/70.938381
3. Karur, K., Sharma, N., Dharmatti. C., Siegel, J.: A survey of path planning algorithms for mobile robots. Vehicles (www.mdpi.com/journal/vehicles). vol. 3, no. 3, pp. 448–468 (2021) doi: 10.3390/vehicles3030027
4. Lentin, J., Cacace, J.: Mastering ROS for robotics programming. Packt Publishing (2021)
5. Leonard, J. J., Durrant-Whyte, H. F.: Mobile robot localization by tracking geometric beacons. IEEE Trans. Rob. Autom., vol. 7, no. 3, pp. 376–382 (1991)
6. Muñoz M. V.: Planificación de trayectorias para robots móviles. Tesis del Departamento de Ingeniería de Sistemas y Automática, Universidad de Malaga – España (1997)
7. Robot operating system (https://www.ros.org/) (2022)
8. Ronquillo, J. B.: Hands-on ROS for robotics programming. Pack Publishing (2020)
9. Siegwart, R., Rez, N. I., Scaramuzza, D., Arkin, R. C.: Introduction to autonomous mobile robots. MIT Press, 2nd Edition (2011)
10. Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics. MIT Press (2005)