Simulación de un algoritmo de navegación sobre un modelo y uso de los resultados en un robot real

Pablo Cabrera Pérez

Resumen —

Este proyecto consistirá en hacer una simulación de un algoritmo de navegación y evasión de obstáculos sobre un modelo en el entorno "Gazebo" y uso de los resultados en un robot "*turtlebot*". Se implementará el algoritmo en un modelo de uso gracias al entorno ROS, en el cual se llevarán a cabo todas las simulaciones previas a su uso en el robot real. Dicho algoritmo, consistirá en una simulación de una navegación en un mapa determinado o por explorar; y la detección, y por consiguiente, la evasión de los obstáculos en el mismo si la situación lo requiere.

Abstract —

This project will be about make a simulation of a navigation algorithm and avoidance of obstacles in a model of use in the Gazebo environment. Then we will use these results in a "turtlebot" robot. We will implement the algorithm in a model of use with the given help of the ROS environment, in where we are going to make all of ours simulation before they use in real robot. Said algorithm, will consist in navigation in a determinate map or one that we should explore, and the detection, and therefore the avoidance of obstacles if the situation requires it.

Index Terms — ROS, GAZEBO, MoveIt, Joint, Links, Robot, ROS Navigation stack, ROS plugins, nodelets, controllers, goal position, local map, global map, laser scan.

1 Introducción

ESTE proyecto trata sobre una simulación de un algoritmo de navegación en un robot. Además también se implementará para que el robot pueda evitar los obstáculos que se encuentre por el camino.

El principal objetivo es poder construir el algoritmo para que el robot pueda navegar de forma autónoma dándole nosotros un punto determinado al que ir. Él, teniendo el punto de partida (su posición) y el punto de llegada trazará una ruta para llegar hasta el final del recorrido.

A su vez demostraremos que podemos dotar de una inteligencia bastante elevada a una simple máquina como puede ser un pequeño robot con ruedas, para que pueda navegar desde un punto hasta otro, y evitando las obstáculos que encuentre por el camino.

Se realizará un estudio completo de las herramientas necesarias para poder llevar a cabo este proyecto. Y poder cumplir el objetivo final que no será otro que, que el robot logre moverse de forma autónoma a un punto que le indiquemos, sin tener mayores problemas para poder llegar hasta el objetivo.

Durante este documento abordaremos primero que nada los objetivos del trabajo, se explicará lo que queremos conseguir. Seguiremos con el estado del arte donde se expondrán los conceptos e ideas más importantes. Abordaremos también la metodología seguida para una buena realización del proyecto así como los fallos encontrados, para a continuación seguir con una exposición de los resultados y acabando con unas conclusiones tras la realización del proyecto.

Se cerrará el artículo con unos agradecimientos y la bibliografía utilizada durante la realización del trabajo.

- *E-mail de contacto: pablozizou_131@hotmail.com*
- *Mención realizada: Ingeniería de Computación.*
- *Trabajo tutorizado por: Ricardo Toledo (Visión para Computadores)*
- *Curso 2016/17*

2 OBJETIVOS

Este proyecto trata sobre la navegación autónoma y evasión de obstáculos. Para una buena práctica se recomienda utilizar Linux, Ubuntu 16.04 LTS en mi caso, e instalar ROS full desktop, ROS Kinetic en mi sistema operativo, siendo esta la versión compatible con Ubuntu 16.04.

El objetivo principal de este proyecto es poder hacer un algoritmo para que nuestro robot pueda navegar de forma autónoma y evada obstáculos.

Pero primero, para poder llegar a nuestro gran objetivo final, hay que cumplir unos ciertos requisitos, ir adquiriendo ciertos conocimientos e ir haciendo otros objetivos más pequeños.

Comprender el entorno con el que se trabaja, ROS, es fundamental para poder hacer una buena práctica del proyecto, así como sus herramientas, Gazebo, MoveIt, etc. Por ello para empezar había que instalar estos programas e iniciarse en su mundo. Haciendo pequeños tutoriales explicativos, mirando ejemplos, etc.

Más adelante se fue añadiendo complejidad a las tareas a realizar, cada vez añadiendo nuevas paquetes y nuevos métodos para acabar entrelazando unas herramientas con otras. Y poder así conseguir el objetivo final.

3 ESTADO DEL ARTE

Durante nuestro proyecto podemos destacar algunas ideas y conceptos muy importantes para su realización.

Para empezar hay que tener los medios y recursos adecuados para su realización, esto implica un ordenador medianamente potente en el cual podamos montar una máquina virtual o bien como es mi caso un sistema operativo aparte, que sea Linux – Ubuntu.

El concepto del láser del robot: este láser va incorporado en la parte superior del robot. Esta posición elevada le permite un mejor escáner sobre su entorno. Esto nos permitirá crear un mapa en 3D, que nos ayudará a evitar obstáculos dinámicos además de los objetos estáticos ya definidos. Podemos ver una muestra de su utilidad en la siguiente imagen:

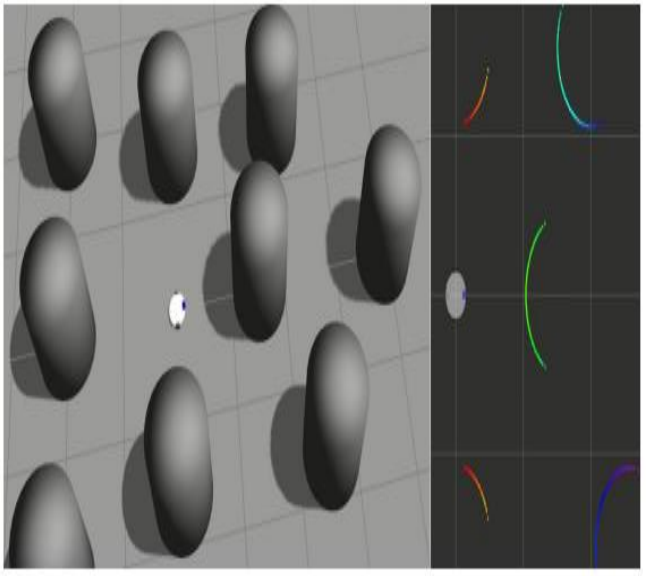

Figura 1: En esta imagen podemos ver la funcionalidad del láser, como detecta los objetos que tiene a su alrededor.

El concepto de los mapas de costes: construidos gracias al láser nombrado anteriormente. Estos mapas nos servirán para poder movernos por el entorno en el cual nos encontremos. Nos darán una visión de los límites y obstáculos dinámicos que puedan ir apareciendo así como los objetos estáticos. Existen dos tipos de mapas. El mapa de coste local y el mapa de coste global. Los mapas de coste son construidos usando los obstáculos presentes entorno al robot.

El primero usado para navegación local, este monitorizará los obstáculos alrededor del robot. Se irá actualizando según la configuración que nosotros le otorguemos.

El segundo es utilizado para la navegación global. Este tendrá una idea general de todo el mapa en el que se encuentra el robot. De igual forma que el mapa de costes locales este también deberá irse actualizando según nuestra configuración. Este mapa se irá alimentando del servidor de mapa e irá enviando información al mapa de costes locales. Como vemos ambos mapas están interconectados, por lo que ambos mapas son muy importantes, sobre todo en el momento de evadir objetos y en el caso de que el robot se quede estancado en algún punto del mapa. En ese caso habrá que recurrir a ambos mapas para poder reconducirlo a un punto válido.

La idea de definir un buen láser y que este pueda crear unos buenos mapas de costes es fundamental para nuestro proyecto. Son los pilares sobre los cuales se asientan nuestros objetivos.

A continuación mostramos una imagen con ambos mapas, el mapa de costes locales y el mapa de costes globales.

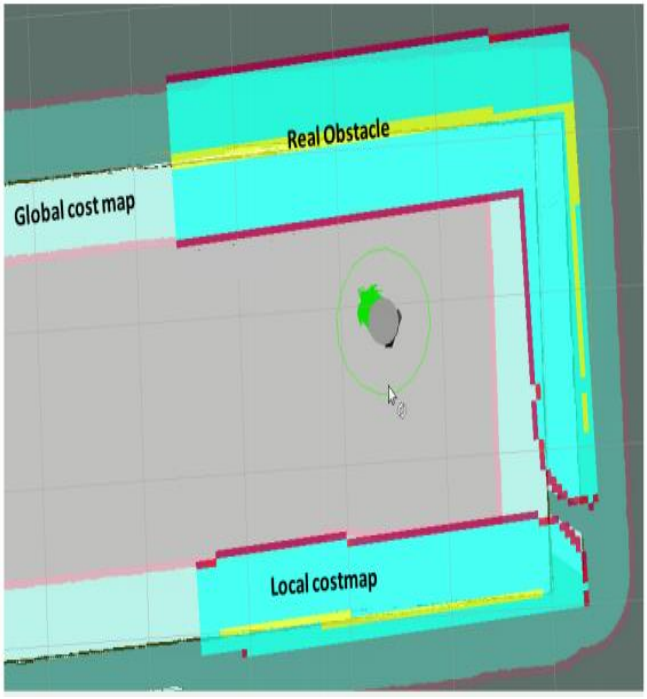

Figura 2: Aquí podemos ver el mapa de costes locales y globales que utilizará el robot para moverse.

4 METODOLOGÍA

4.1 Procedimiento

En un primer momento, se instalará todo lo necesario para llevar a cabo el proyecto, es decir: el sistema operativo, el programa a usar (ROS), etc. Posteriormente, se comenzará a investigar (mediante la lectura de libros, manuales, tutoriales, foros u otros recursos) cómo usar dicho programa, las librerías que pueden servirnos para la implementación tanto del robot como del algoritmo, las mejores herramientas que acompañan al programa, los paquetes que pueden ser interesantes para la visualización del robot y nuestro entorno, los diferentes tipos de robots, los comandos para iniciar los programas, la compatibilidad de versiones, su estructura general, los diferentes apartados en los que se divide, entre otros.

Todo el proyecto será implementado con la ayuda del entorno ROS, además de sus complementos como lo son Gazebo, MoveIt, sus librerías, etc., el sistema Linux, concretamente Ubuntu 16.04 LTS, los lenguajes de programación Python, C++, XML, así como ficheros de texto, y diversas fuentes de internet que se citarán por consiguiente en la bibliografía.

La metodología optada es, el seguimiento del libro "Mastering Ros for Robotic Programming" en el cual nos explican cómo proceder para poder implementar nuestro algoritmo. Este comenzará con cosas básicas para ir conociendo el entorno sobre el cual trabajaremos durante todo nuestro proyecto. Tutoriales de inicialización que nos servirá de gran ayuda para poder entender su funcionamiento así como su estructura y método de trabajo. Poco a poco iremos aprendiendo nuevas funcionalidades que nos acercarán a nuestro deseado algoritmo de navegación. Así como su wiki en la cual encontramos más detalles sobre algunos temas y varias webs de información y resolución de errores.

Una vez comenzada la inicialización de estos programas, ponemos en marcha la visualización de varios robots, un brazo mecánico y un pequeño "turtlebot".

Observamos como nuestros modelos de robots son visualizados por las herramientas del programa. Primero obtenemos un ejemplo de vista en 2D y después podemos obtener una en 3D. En dichas visualizaciones podemos ver en detalle los links y joints del robot entre otros, sus ejes, la rotación que puede tener, su movilidad, etc. Llamamos joints a las uniones de varias partes de un robot, y decimos links a dichas partes del robot; podemos decir que el joint es la articulación y los links son las partes que une una articulación. Un punto a tener en cuenta es que podemos interactuar directamente con los robots, gracias sus simulaciones, moverlos, girarlos, etc., lo que permite un mejor entendimiento del trabajo que se está llevando a cabo.

Se continuó con la construcción de los mapas globales y locales, esto gracias al laser. Dichos mapas son los que no servirán para poder mover el robot por todo nuestro entorno.

Una vez consignados el robot, los mapas, el entorno y todo lo necesario para su ejecución, se comenzó a probar la navegación del robot. Primero guiándolo nosotros por el mapa, es decir con el mismo teclado le indicábamos hacia donde queríamos que fuese. Podemos controlar su velocidad, su giro, etc. Comprobábamos que los mapas se iban recalculando y fuesen acorde a su entorno, esto es que el láser cumpla su función y tengamos un mapa idéntico al entorno que nos rodea para poder movernos correctamente en el sin la necesidad de mirar la imagen de nuestro entorno, simplemente gracias a los mapas que calcula el propio robot.

A continuación se muestra una imagen con el recorrido que seguiría el robot sin nada que obstruyese su paso.

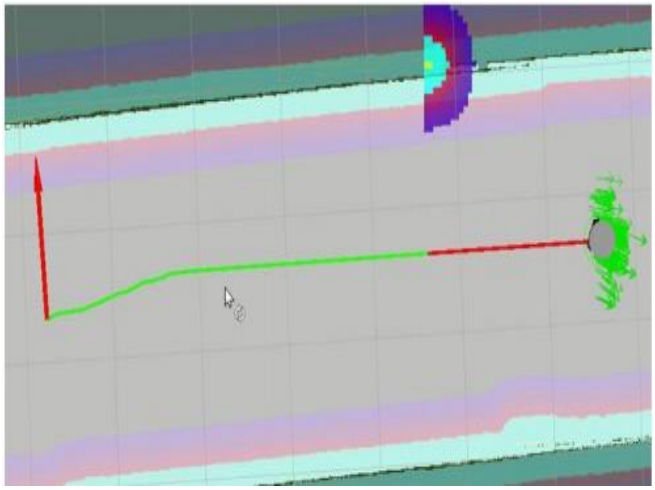

Figura 3: En esta imagen podemos ver el recorrido que realizará el robot para ir a un determinado punto sin que haya obstáculos.

Tras poder realizar una navegación manual con el robot, se procedió a implantar la navegación autónoma, que el robot vaya solo a un lugar determinado, sin la necesidad de que una persona lo tenga que controlar y mover, simplemente dándole unas coordenadas y que él se dirija hacia ellas. Igualmente tras la realización de este algoritmo para la navegación autónoma, se lanzó el robot y su entorno y se le indicó varios puntos a los que ir para comprobar que el robot era capaz de ir al punto que nosotros le enviábamos.

Una vez nuestro robot es capaz de navegar autónomamente es hora de poner un poco más de complejidad a su navegación. Hablamos de los obstáculos. Introducimos en el mapa varios de ellos, cualquier tipo es válido, sea un muro, una simple pelota, un cubo, etc. Ahora tenemos que ser capaces de poder evitarlos para poder llegar a al destino. De la misma forma que se probó su navegación autónoma, se probó su navegación con la evasión de obstáculos, al enviarle unas coordenadas a las que ir, era capaz, gracias a su laser y sus mapas locales y globales, detectar los objetos que obstruían su paso para evitarlos y llegar a su punto de destino.

Con la prueba de esto, podemos decir que se consiguió el objetivo establecido al principio del proyecto que no era otro que poder conseguir que el robot navegase de forma autónoma sin problemas, y que evitase cualquier impedimento que le surgiese a su paso, llegando así a su destino final.

4.2 Fallos

No obstante, durante todo este procedimientos nos topamos con varios tipos de errores, alguno de ellos bastante complicados de solucionar. Estos fallos se producían durante varias fases del proyecto. Los primeros fallos ocurrían a la hora de compilar los ficheros, es decir de construir todas las dependencias, crear los paquetes y las librerías. Otros llegaban a la hora de buscar los caminos de las rutas de los ficheros, otros al ejecutar los archivos y montar las imágenes de visualización del robot y del entorno. También a la hora de interactuar con el robot para moverlo. Como podemos ver estos fallos se producían en diferentes áreas de nuestro trabajo.

4.2.1 Fallos menores

Errores comunes como pueden ser la falta de librerías o paquetes, que requiera de su instalación para poder arrancar una determinada herramienta. La ausencia de ficheros, como puede ser la imagen de los mapas a cargar, es decir el entorno sobre el cual queremos que trabaje y se mueve nuestro robot. La falta de la exportación de algún fichero, la mala declaración de algunas variables y error al indicar el camino de los ficheros.

4.2.2 Fallos complejos

Y otros errores más graves a los que se le dedicó mucho esfuerzo y trabajo para poder solventarlos, como podía ser la incompatibilidad de versiones, que requería añadir algunas líneas de código suplementarias, que desconocíamos, instalar algunos parches, para poder compilar el código, la instalación de nuevas librerías, y demás soluciones encontradas buscando en diferentes webs. Todo ello era consultado en páginas de internet, en foros donde algunos ya habían tenido el mismo fallo y lo comentaban, gracias a ello podía seguir avanzando y solventando fallos. También encontramos que una vez inicializado el programa, no éramos capaces de comunicarnos con el robot, no podíamos moverlo manualmente, no recibía la instrucciones que le enviábamos, para ello se tuvo que hacer una revisión del código para comprobar si faltaba algo como por ejemplo algún método de conexión o sincronización, comprobar que se cargaba correctamente, que no hubiesen fallos a la hora de sincronizar el robot con el mando de control, llamado también "keyboard", que los canales de envío y recepción de mensajes estuviesen creados correctamente e interconectados unos con otros, etc., hasta poder solventarlo. Y de esta manera iban surgiendo errores a los cuales debíamos enfrentarnos y solventar para poder cumplir nuestros objetivos.

Todo esto ha requerido un laborioso trabajo de investigación por diferentes portales de internet, algunos bastantes desconocidos pero que servían y ayudaban a corregir ciertos puntos del trabajo. .

Las fuentes, tales como las páginas webs han sido de gran ayuda y utilidad en todo momento, ya que, durante varias fases del proyecto nos hemos topado con muchos errores. Errores que no sabía a qué se debían ni de dónde provenían y en los cuales pasé bastante tiempo para in-

tentar solventarlo, analizando fichero por fichero para ver de dónde podía proceder, hasta hallar con él, de esta forma me aseguraba que los demás ficheros estaban funcionando correctamente y podía encontrar el que me estaba causando los problemas de funcionamiento.

La propia página del programa ROS, tenía en foro en el cual se discutía y comentaban los fallos que iban surgiendo durante la ejecución del programa. Los fallos eran expuestos en el foro, comentando el error generado por el programa, es decir la salida que obtenía, así como una copia de los ficheros en los cuales podía estar el problema. Estos ficheros eran leídos y analizados, y servían para un mejor entendimiento del caso y poder corregirlos. Cabe mencionar que el libro, antes mencionado, de por sí ya venía con algunos fallos de serie, lo que complicó todavía más las tareas. Fallos como que no incluyen un fichero necesario, no exportaban los paquetes y métodos que hacían falta para otros ficheros, no instalaban las librerías necesarias para la compilación del programa.

5 RESULTADOS

Una vez finalizado el proyecto, podemos evaluar los resultados obtenidos que nos demuestran que el trabajo realizado ha sido bueno ya que hemos alcanzado los objetivos propuestos. Se propuso y se consiguió una correcta simulación de navegación autónoma del proyecto de robot en un espacio determinado por la configuración de determinados mapas dinámicos en los que se tiene que mover interactuando con ellos y con la elección de rutas posibles y certeras que permitan desplazarse sin tener problemas en la navegación por encontrar obstáculos entre origen y destino. Analizar estos resultados es tan sencillo como ver que el robot navega autónomamente por el mapa y que llegaba al punto que nosotros le habíamos puesto como objetivo final.

Estos mapas pueden ser mapas ya establecidos cuyas imágenes cargaremos en el entorno, importándolas desde archivos alternativos o bien podemos iniciar un mundo vacío y nosotros ir creando un mapa y añadir obstáculos a nuestro criterio, el mapa que creamos más conveniente para nuestra simulación y que más se aproxime a lo que buscamos, o necesitemos plasmar por exigencias de la propia realidad. Esto es, dado un entorno de navegación, que incluso puede ser creado por nosotros mismos, instalado el robot en dicho mapa, este es capaz de interpretar la información proporcionada como punto de origen, un punto A, y luego la orden de encontrar una ruta posible para llegar a un punto de destino que también se le proporciona y que con sus características instaladas, laser para identificar los obstáculos y algoritmos para identificar y crear las rutas posibles.

Una vez recibida la posición final a la cual debe ir, hará todos los cálculos necesarios para poder llegar a ella: cálculo de costes locales, costes globales, escáner de obstáculos y evasión de estos, el trazo de la ruta, posibles imprevistos u obstáculos durante la navegación, etc. Quizás esta posición final, llamada "goal position" es una de las informaciones más importantes que necesite el robot, ya que gracias a ella analizará primero su entorno y después la ruta para poder llegar al objetivo, analizando su entorno en todo momento, por si tuviese que evitar algún obstáculo.

Hemos comprobado que el robot es capaz de analizar lo que tiene alrededor, es decir detectar los obstáculos, los sitios a los que puede llegar y como puede llegar a ellos. Además es conveniente hacer una ruta con el robot por todo el mapa y de esta manera, podrá crear un mapa global de su entorno llamado "global map". Este mapa global es como una imagen que guarda de todo el escenario. Por el que se puede mover con sus límites, restricciones, y demás informaciones necesarias para una correcta navegación. También tiene un mapa de costes locales, es decir lo que más próximo tiene a su alrededor, que se irá recalculando cada vez que el robot se mueva, para poder ir actualizando sus rutas y mapas. Este mapa llamado "local map" viene a ser lo mismo que el global map pero a una menor escala, con lo más próximo a él. Para poder construir ambos mapas, el robot consta de un láser el cual determina si hay algo próximo a su entorno y detecta cuando hay algo que obstruye el camino. Para poder moverse el robot tiene dos ruedas principales las cuales le dan la fuerza para desplazarse, funcionan como motores. Y dos ruedas más pequeñas adicionales en los otros costados que permiten una mejor estabilidad al robot.

Estamos satisfechos con estos resultados ya que hemos podido observar que con un algoritmo simple se puede conseguir resultados interesantes parar aplicar en temas como aeronáutica o cosmonáutica, por supuesto con programación más compleja pero nos quedamos con que se puede ver la opción de elección de esos mapas creados y esas rutas alternativas para llegar a cumplir el objetivo propuesto. Así logramos que el robot se pueda mover de forma autónoma a un punto determinado. Además si se interactúa con los mapas dado que la realidad puede cambiar en segundos se puede observar que añadiendo algún obstáculo en la ruta no se presenta ningún problema ya que este será capaz de crear en segundos el estudio de esos mapas para crearlos y con ello crear esas rutas alternativas que nos den la opción de evitarlos y llegar a su destino.

A continuación se muestra una imagen con el recorrido que haría el robot si hubiese un obstáculo de por medio.

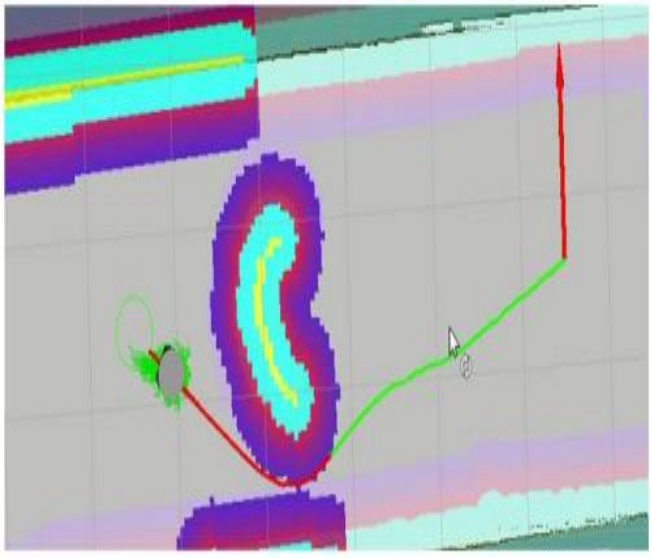

Figura 4: En esta imagen podemos ver el recorrido que realizará el robot para ir a un determinado punto cuando hay obstáculos en su camino.

6 CONCLUSIÓN

Tras la realización del proyecto, lo primero que quiero destacar es la gran capacidad de las herramientas utilizadas durante el mismo, como puede ser Gazebo, MoveIt entre otros. Herramientas muy potentes que nos permiten realizar cualquier tipo de tarea que queramos llevar a cabo.

Hoy en día ya se utilizan estos métodos en muchos casos, como por ejemplo el enviar un satélite al espacio de forma autónoma en el que le decimos que vaya a un determinado lugar,"goal position", también tendrá un algoritmo parecido, pero mucho más complejo obviamente, y por supuesto un local y global map para poder saber por dónde se mueve. Que puede depender e ir cambiando en función de las ondas térmicas y ondas solares entre otros. Otro caso puede ser el de los GPS, también tiene un algoritmo de navegación y es el claro ejemplo en donde desde un punto de partida le decimos adonde queremos llegar y este nos traza la mejor ruta, o varias dependiendo de las condiciones, evitando los obstáculos, siendo considerados estos como edificios, monumentos, e incluso calles cortadas, etc.

Como podemos ver casi todo depende de un buen cálculo de los mapas tanto locales como globales, que junto a la posición final son las cosas más importantes para que el robot pueda navegar autónomamente. Sin estos mapas el robot no podría realizar una ruta segura hasta su objetivo. Estos mapas son alterados constantemente por el láser, principal valedor de estos mapas. Un radar potente y de buena calidad asegura un buen planning de los mapas. Consideramos siempre los mapas globales y locales para interactuar con el medio y así poder elegir la ruta más acertada para llegar al punto de destino.

Todas estas herramientas pueden tener una gran utilidad y facilitarnos mucho la vida. Se puede aplicar en ámbitos de videojuegos, de creación de robots de limpiezas, de explorar territorios, de distribuir cosas, de fabricación, etc. En ellas podemos hacer las tareas o proyectos que queramos realizar. Así es, si queremos probar un algoritmo con un determinado robot en un mapa establecido, para ver y analizar su comportamiento así como mejorarlo y corregir los errores que puedan surgir, podremos simular todo esto en nuestro entorno y además sin coste alguno.
Gracias a este tipo de programas podríamos mejorar en muchos aspectos cotidianos de nuestra vida.
Es también una buena herramienta de diversión y entretenimiento, en la cual podemos pasar la tarde inventado entornos probando nuestros robots, simulando escenas, etc.

Me siento muy afortunado de haber podido realizar este proyecto. Gracias a su realización me ha enseñado mucho más de un mundo que ya veía muy interesante de por sí y con muchísimo futuro en nuestras vidas, y he aprendido muchas cosas nuevas e innovadoras, para mí. Seguiré investigando y aprendiendo nuevas tecnologías y nuevos métodos de un mundo apasionante. Y porque no, dedicarme a ello.

AGRADECIMIENTOS

Han sido cuatro años y medio de carrera universitaria bastante duros, unos más que otros, pero de los cuales en cada uno de ellos hemos madurado y aprendido cosas nuevas. Quiero agradecer primero que nada a mis padres por hacer posible mis estudios fuera de casa y que nunca me haya faltado de nada. Siempre mostrándome todo su apoyo y ayudándome en todo. Eternamente agradecidos, sin ellos no hubiese llegado adonde estoy ahora mismo.
Mi hermana también se cuela en esta primera posición, por estar ahí para todo. Ambos animándome para seguir adelante.
Agradecer también al resto mi familia por todo el apoyo que me han dado, la preocupación y ayuda que me han mostrado. Y estar pendiente por si me pudiese faltar cualquier cosa.
Por su puesto a todos los profesores que he tenido la suerte de ser su alumno y aprender nuevas cosas. Que sin ellos todas estas enseñanzas no habrían sido posibles.
También a todas esas personas que han estado conmigo en las buenas y malas, en la cercanía o lejanía, llámense amigos, compañeros de universidad, compañeros de piso, y demás personas que me han ayudado a seguir creciendo como persona.

Destaco también la posibilidad y oportunidad de intercambio y ese buen año de Erasmus, lleno de nuevas experiencias que me sirvieron para crecer tanto a nivel personal como profesional.

BIBLIOGRAFÍA

[1] Wiki.ros.org. (2016). es - ROS Wiki. [online] Available at: http://wiki.ros.org/es.

[2] Es.wikipedia.org. (2016). Sistema Operativo Robótico. [online] Available at: https://es.wikipedia.org/wiki/Sistema_Operativo_Rob%C3%B3tico.

[3] Ee.surrey.ac.uk. (2001). UNIX / Linux Tutorial for Beginners. [online] Available at: http://www.ee.surrey.ac.uk/Teaching/Unix/.

[4] Wiki.ros.org. (2016). ROS/Tutorials - ROS Wiki. [online] Available at: http://wiki.ros.org/ROS/Tutorials.

[5] Wiki.ros.org. (2016). ROS/StartGuide - ROS Wiki. [online] Available at: http://wiki.ros.org/ROS/StartGuide#Learning_ROS.

[6] Answers.ros.org. (2017). [test.launch] is neither a launch file in package - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/216217/testlaunch-is-neither-a-launch-file-in-package/

[7] Answers.ros.org. (2017). Book: Mastering ROS for Robotics Programming, 2nd Edn; Chapter 2 - Working with 3D Robot Modeling in ROS - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/242044/book-mastering-ros-for-robotics-programming-2nd-edn-chapter-2-working-with-3d-robot-modeling-in-ros/

[8] Answers.ros.org. (2017). Book: Mastering ROS for Robotics Programming, 2nd Edn; Chapter 3 - Simulating Robot using ROS & Gazebo - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/242778/book-mastering-ros-for-robotics-programming-2nd-edn-chapter-3-simulating-robot-using-ros-gazebo/

[9] GitHub. (2017). Could not load controller 'joint_state_controller' because controller type 'joint_state_controller/JointStateController' does not exist · Issue #7 · qboticslabs/mastering_ros. [online] Available at: https://github.com/qboticslabs/mastering_ros/issues/7

[10] GitHub. (2017). Could not load controller 'joint_state_controller' because controller type 'joint_state_controller/JointStateController' does not exist. · Issue #10 · qboticslabs/mastering_ros. [online] Available at: https://github.com/qboticslabs/mastering_ros/issues/10

[11] Gazebosim.org. (2017). Gazebo : Tutorial : Installing gazebo_ros_pkgs. [online] Available at: http://gazebosim.org/tutorials?tut=ros_installing&cat=connect_ros

[12] Gazebosim.org. (2017). Gazebo : Tutorial : Which combination of ROS/Gazebo versions to use. [online] Available at: http://gazebosim.org/tutorials?tut=ros_wrapper_versions&cat=connect_ros#InstallingGazebo

[13] Wiki.ros.org. (2017). joy - ROS Wiki. [online] Available at: http://wiki.ros.org/joy#joy_node.py

[14] Wiki.ros.org. (2017). joy/Tutorials/ConfiguringALinuxJoystick - ROS Wiki. [online] Available at: http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick

[15] Wiki.ros.org. (2017). xacro - ROS Wiki. [online] Available at: http://wiki.ros.org/xacro#Deprecated_Syntax

[16] GitHub. (2017). [chapter 3, Adding the ROS teleop node] roslaunch diff_wheeled_robot_control keyboard_teleop.launch · Issue #2 · qboticslabs/mastering_ros. [online] Available at: https://github.com/qboticslabs/mastering_ros/issues/2

[17] Answers.ros.org. (2017). can't locate node in package - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/145801/cant-locate-node-in-package/

[18] Wiki.ros.org. (2017). catkin/CMakeLists.txt - ROS Wiki. [online] Available at: http://wiki.ros.org/catkin/CMakeLists.txt

[19] Answers.ros.org. (2017). Complaint from openni_launch on already advertised services - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/12244/complaint-from-openni_launch-on-already-advertised-services/

[20] Moveit.ros.org. (2017). Concepts | MoveIt!. [online] Available at: http://moveit.ros.org/documentation/concepts/

[21] Wiki.ros.org. (2017). Distributions - ROS Wiki. [online] Available at: http://wiki.ros.org/Distributions

[22] Answers.ros.org. (2017). How can I speed up the shutdown of gazebo with pr2_controler_manager? [closed] - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/11111/how-can-i-speed-up-the-shutdown-of-gazebo-with-pr2_controler_manager/

[23] Answers.ros.org. (2017). Multiple errors launching openni.launch: "Tried to advertise a service that is already advertised in this node" - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/11313/multiple-errors-launching-opennilaunch-tried-to-advertise-a-service-that-is-already-advertised-in-this-node/

[24] Answers.ros.org. (2017). ROS-Gazebo Failed to load joint_state_controller - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/154166/ros-gazebo-failed-to-load-joint_state_controller/

[25] Answers.ros.org. (2017). System requirements for rviz and gazebo - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/11191/system-requirements-for-rviz-and-gazebo/

[26] M.blog.csdn.net. (2017). #error This file requires compiler and library support for the ISO C++ 2011错误解决办法. [online] Available at: http://m.blog.csdn.net/article/details?id=50759490

[27] Gazebosim.org. (2017). Gazebo : Tutorial : Plugins 101. [online] Available at: http://gazebosim.org/tutorials?tut=plugins_hello_world&cat=write_plugin

[28] Gazebosim.org. (2017). Gazebo : Tutorial : Which combination of ROS/Gazebo versions to use. [online] Available at: http://gazebosim.org/tutorials?tut=ros_wrapper_versions

[29] Answers.gazebosim.org. (2017). gazebo default directory and can not insert model - Gazebo: Q&A Forum. [online] Available at: http://answers.gazebosim.org/question/177/gazebo-default-directory-and-can-not-insert-model/

[30] Wiki.ros.org. (2017). kinetic/Migration - ROS Wiki. [online] Available at: http://wiki.ros.org/kinetic/Migration

[31] Wiki.ros.org. (2017). kinetic/Planning/Maintenance - ROS Wiki. [online] Available at: http://wiki.ros.org/kinetic/Planning/Maintenance

[32] Discourse.ros.org. (2017). New Packages for Jade and Kinetic 2016-11-15. [online] Available at: https://discourse.ros.org/t/new-packages-for-jade-and-kinetic-2016-11-15/824

[33] Wiki.ros.org. (2017). pr2_description - ROS Wiki. [online] Available at: http://wiki.ros.org/pr2_description
[34] Answers.ros.org. (2017). problem with plugin in gazebo - ROS Answers: Open Source Q&A Forum. [online] Available at: http://answers.ros.org/question/43578/problem-with-plugin-in-gazebo/
[35] Ros.org. (2017). ROS Kinetic Kame Released - ROS robotics news. [online] Available at: http://www.ros.org/news/2016/05/ros-kinetic-kame-released.html
[36] Repositories.ros.org. (2017). ROS packages in Indigo Jade Kinetic - 2017-01-21 02:36:42 -0800. [online] Available at: http://repositories.ros.org/status_page/compare_indigo_jade_kinetic.html
[37] “BOOST_JOIN”, Q. (2017). Qt4 + CGAL - Parse error at "BOOST_JOIN". [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/15455178/qt4-cgal-parse-error-at-boost-join
[38] Wiki.ros.org. (2017). catkin/CMakeLists.txt - ROS Wiki. [online] Available at: http://wiki.ros.org/catkin/CMakeLists.txt#Overall_Structure_and_Ordering
[39] Wiki.ros.org. (2017). catkin/commands/catkin_make - ROS Wiki. [online] Available at: http://wiki.ros.org/catkin/commands/catkin_make
[40] GitHub. (2017). Fix: Parse error at "BOOST_JOIN" by bchretien · Pull Request #399 · ros-planning/moveit_ros. [online] Available at: https://github.com/ros-planning/moveit_ros/pull/399
[41] Wiki.ros.org. (2017). gazebo_ros_pkgs - ROS Wiki. [online] Available at: http://wiki.ros.org/gazebo_ros_pkgs
[42] Docs.ros.org. (2017). ImuDisplay — rviz_plugin_tutorials documentation. [online] Available at: http://docs.ros.org/jade/api/rviz_plugin_tutorials/html/display_plugin_tutorial.html
[43] Bloom.readthedocs.io. (2017). Bloom — bloom 0.5.10 documentation. [online] Available at: http://bloom.readthedocs.io/en/0.5.10
[44] GitHub. (2017). Parse error at "BOOST_JOIN" with Boost 1.58 in moveit_ros/visalization package · Issue #653 · ros-planning/moveit_ros. [online] Available at: https://github.com/ros-planning/moveit_ros/issues/653
[45] Docs.ros.org. (2017). PlantFlagTool — rviz_plugin_tutorials documentation. [online] Available at: http://docs.ros.org/jade/api/rviz_plugin_tutorials/html/tool_plugin_tutorial.html
[46] Wiki.ros.org. (2017). pr2_simulator/Tutorials/StartingPR2Simulation - ROS Wiki. [online] Available at: http://wiki.ros.org/pr2_simulator/Tutorials/StartingPR2Simulation
[47] Wiki.ros.org. (2017). ros_control - ROS Wiki. [online] Available at: http://wiki.ros.org/ros_control
[48] Bugreports.qt.io. (2017). [QTBUG-22829] boost 1.48, Qt and [Parse error at "BOOST_JOIN"] error - Qt Bug Tracker. [online] Available at: https://bugreports.qt.io/browse/QTBUG-22829
[49] Wiki.ros.org. (2017). urdf/XML - ROS Wiki. [online] Available at: http://wiki.ros.org/urdf/XML
[50] Ioan A. Şucan, L. (2017). The Open Motion Planning Library. [online] Ompl.kavrakilab.org. Available at: http://ompl.kavrakilab.org/