Programación de comportamientos de un robot autónomo.

Gerard Vivet Moral

Resumen—El propósito de este proyecto, es lograr un mayor grado de autonomía de los comportamientos de un robot. Para incrementar la autonomía, necesitamos supervisar las baterías del robot, parar su estado actual, volver a la estación de carga y acoplarse cuando sea necesario. Asimismo, el robot se desplazará por su entorno hacia destinos dados por el usuario, a la vez evitando objetos que interrumpan su trayectoria planificada de manera dinámica. Todo ello adherido en un solo script Python y ROS (Sistema operativo robótico) que permitirá controlar los componentes de un robot desde uno o varios ordenadores. En este documento se detallarán los métodos, resultados, además de la utilización del script entre otros.

Palabras clave—Robot, ROS, sistema operativo robótico, auto acoplamiento, navegación autónoma, comportamientos, evasión obstáculos dinámicos, rospy.

Abstract—The purpose of this Project is to achieve a greater degree of automation of robot behaviours. In order to increase the autonomy, we need to supervise the battery power, stop its actual state, back to the charging station and docking to it when necessary. Likewise, the robot moves in its environment, towards user targets, both avoiding objects that interrupt its path dynamically. Everything is embedded in Python script and ROS (Robot Operating System) which allow us to control the robot components in one or several computers. In this document the methods shall be detailed, results, as well as the use of the script along with others.

Index Terms—Robot, ROS, robot operating system, auto docking, autonomous navigation, behaviours, dynamic obstacle avoidance, rospy.

1 Introducción

Actualmente y cada vez más, los humanos tendemos a ser más cómodos haciendo uso de la tecnología. Quien no ha imaginado un robot que pueda traer tu refresco preferido al lado de tu escritorio mientras trabajas, además de las ventajas de navegar con total libertad evitando a las personas y tener la inteligencia suficiente para irse a cargar cuando sea necesario, pues basta de imaginar porque aquí está la solución.

Este proyecto se basa en la idea de comportamientos básicos de un robot de manera automática, todo ello ejecutado en un solo script y dejar actuar el robot en base a las peticiones de un usuario.

El desarrollo se realizará gracias al robot autónomo TurtleBot[1], capacitado con una estructura de movilidad propia, capaz de elaborar sus propios mapas con la captura de profundidades de Kinect[2] y una base de carga fija. TurtleBot no está capacitado de una unidad de procesamiento por lo que también es necesario el uso de dos ordenadores bajo ROS.
El documento detallará qué objetivos se van a abarcar, el estado del arte, qué metodologías se han utilizado, el porqué de estas, los objetivos que se han solucionado, y finalmente se expondrán las conclusiones.

1.1 Objetivos

Es posible fraccionar el proyecto en los siguientes objetivos.

- Funcionamiento del robot de forma remota.
- Interpretación y navegación del mapa global, va a tener que desplazarse en el entorno por coordenadas dadas por el usuario.
- Incorporar evasión de obstáculos dinámicos.
- Comprobación de la capacidad de las baterías, retornar a la estación de carga y proceder al acople.
- Diseño simple de un sistema de control para el usuario.

- *E-mail de contacto: gerard.vivet@e-campus.uab.cat*
- *Mención realizada: Engañaría de Computación.*
- *Trabajo tutorizado por: Ricardo Toledo (CVC)*
- *Curso 2015/16*

2 Estado Del Arte

Actualmente existen muchos tipos de robots, androides, móviles, industriales, médicos, teleoperadores, poliarticulados. Este proyecto se va a centrar en el desarrollo del robot móvil.
Estos cuentan con patas o ruedas para desplazarse y sistemas de sensores, que reciben la información del exterior. Usualmente, estos robots son utilizados en almacenes para el traslado de mercaderías u objetos. También son útiles para la investigación de lugares de difícil acceso o incluso para tareas más domesticas como el barrido del suelo. Existen muchas versiones de robots móvil, pero uno de los temas a abarcar es el posicionamiento dinámico, actualmente están saliendo nuevas tecnologías tales como el coche de Google con la colaboración de Alphabet Inc[3] que empiezan a estandarizar sistemas de posicionamiento.

Algunos de los robots soportados por ROS[4] se pueden encontrar en WillowGarage PR2 un robot personal desarrollado por Willow Garage[5].
Otros de los proyectos es Robotnik FREIGHT[6], una empresa Española, especializados en robots móviles en arquitecturas sobre ROS.
FREIGHT es una plataforma diseñada para hacer frente a la industria de la logística con estación de carga, sin embargo la accesibilidad para este tipo de robots solo es para grandes industrias.
Gracias a Kobuki será posible desarrollar un sistema de comportamientos para un robot móvil, útil para cualquier robot que funcione bajo ROS, mejorando la accesibilidad para el hogar.

3 Metodología

El proyecto se ha desarrollado según lo previsto adaptadas a las fechas de entrega [7]. El desarrollo se realiza bajo el sistema operativo GNU/Linux[8], con el framework ROS(Robotic Operating System), con el lenguaje de programación Python[9] ya que es posible utilizar la librería rospy[10] ,que permite interactuar con ROS. El informe se ha desarrollado bajo las rubricas de evaluación impuestas por la Universidad Autónoma de Barcelona [11], siguiendo los estándares del IEEE [12].

3.1. Que es ROS

ROS (Robotic Operating System), es un sistema Licencia-BSD [13], para controlar los componentes robóticos desde un PC. Un sistema ROS es similar a un grafo, se compone de un número de nodos independientes, cada uno de los cuales se comunica con los otros utilizando un modelo de publicación y subscripción de mensajería. Contiene una colección de herramientas, librerías, y modelos que permite simplificar la creación de tascas de comportamientos complejos de un robot. Por ejemplo, un driver del sensor puede ser implementado como nodo mientras publica los datos en una cadena de mensajes, y esos mensajes pueden ser leídos por otros nodos, como sistemas de alto nivel, búsqueda de caminos, filtros y direccionamientos.

3.2 Porque ROS

ROS permite que los nodos no tengan que estar en un mismo sistema o incluso no tienen que ser de la misma arquitectura. Esto hace que ROS sea realmente flexible y adaptable a las necesidades finales del usuario.
Fue construido desde cero para fomentar el desarrollo de la robótica colaborativa [14], teniendo una mejora continua, con una amplia comunidad [15].
Algunos de los componentes centrales son:

- **Infraestructura de comunicaciones**: El nivel más bajo de ROS ofrece una interface de comunicación de paso de mensajes [16] y petición de respuestas de llamadas remotas [17].
- **Características especificas de un robot**: Provee librerías para utilizar rápidamente con el robot tales como, Geometría [18], descripción del lenguaje [19], estimación de pose [20], localización [21], mapeo [22], y navegación [23].
- **Herramientas**: Tiene un sistema de herramienta que permite, la introspección, depuración, y visualización del estado del sistema que se está ejecutando. Rviz, rqt mostrados en la siguiente imagen 1.

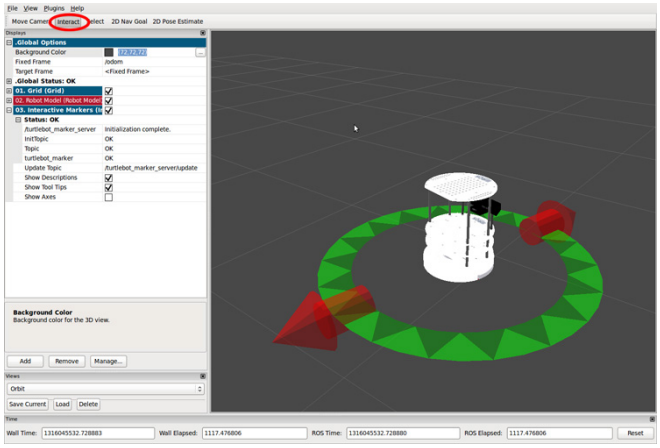


Fig. 1. *Rviz será la herramienta más utilizada de este proyecto, permita la visualización 3D de la posición de los sensores y su posición global.*

3.3 Conceptos Generales

Es un pequeño sistema operativo que corre por encima de sistemas UNIX permitiendo la abstracción como la comunicación de diferentes hardware, control y paso de mensajes entre los procesos.

Funciona mediante un sistema interconectado compuesto por:

- **Nodos:** Son los que realizan los cálculos. Un sistema diseñado en ROS se compone de muchos nodos, hechos para ser modular. El nodo controlará telemetro, odómetro, motores, planificación de trayectoria entre otros.

- **Máster**: El Ros máster proporciona el registro de nombres y búsqueda para el sistema de nodos. Este otorga las direcciones a los demás así como las conexiones que hay actualmente.

- **Parámetros del servidor**: Permite que los parámetros de los datos sean almacenados al máster.

- **Mensajes**: Los nodos se comunican mediante un mensaje. Simplemente contiene la estructura de comunicación para entenderse entre sí.

- **Tema o tópico**: Los mensajes se transportan a través de un sistema de publicaciones/subscripciones semántico. Un nodo envía un mensaje a un tema, que este es útil para identificar el contenido del mensaje. Un nodo que esté interesado a un tema, puede subscribirse siempre que quiera.

- **Servicios**: Es un concepto importante ya que se hace uso del paquete move_base[24]. En el modelo de publicación/subscripción permite un sistema de muchos a muchos que a veces no es útil cuando es necesario un sistema de petición respuesta, esto se logra gracias a los servicios que se definen con los mensajes de solicitud/respuesta.

En resumen el ROS máster contiene temas y servicios de información para los nodos ROS. Los nodos tienen una comunicación con el máster de manera fija o dinámicamente.

3.4 El Robot

Para este proyecto ha sido necesario el uso del robot Kobuki figura 2 o Turtlebot, está configurado por defecto para correr bajo ROS con una velocidad de translación de 70cm/s, una velocidad de rotación de 180 grados/s, posibilidad de carga de hasta 5 Kg, un tiempo de operación estimado de 3 horas. También dispone de una estación de carga con 3 sensores Infrarrojos [25], de la cual serán necesarias 1,5 horas de recarga.

Fig. 2. *El robot móvil Kobuki, una plataforma de arquitectura ROS, utilizado en este proyecto.*

3.5 Pasos previos

Antes de trabajar con ROS y el robot, es necesario tener dos computadoras con ROS instalado. Una de las máquinas tendrá que controlar los procesos del robot, llamada máster, y la otra será la estación de trabajo por donde serán dadas las órdenes necesarias al máster, tal y como se muestra en la figura 3.

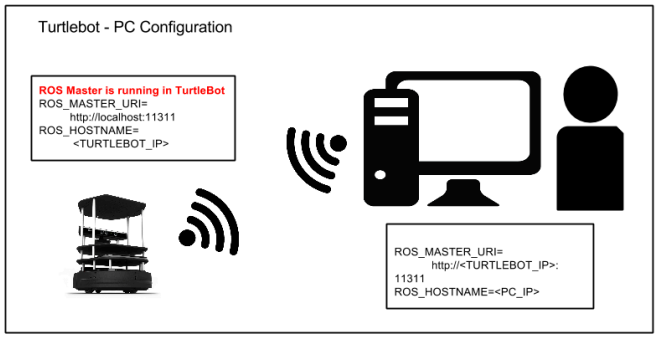


Fig. 3. *Roles de los sistemas para la comunicación online. Imagen extraída de http://wiki.ros.org.*

Esto ofrece la posibilidad en la estación de trabajo de ejecutar el máster mediante SSH(Secure Shell)[26] y los procesos necesarios. Mientras que en el mismo terminal se actúa como cliente monitorizando el sistema por Rviz. Previamente, es necesario tener el mapa global en el máster, TurtleBot tiene que elaborar un mapa mediante Gmapping, para poder elaborar trayectorias. Con la estación de trabajo se irá recorriendo el espacio con las comandas de teleoperación [27] y visualizando el mapa con Rviz. Como resultado se obtendría el mapa de la estancia. Gmapping es un eficiente RBPF(Rao –Blackwellized particle filter) [28], para desarrollar mapas a partir de los datos de un láser. Una vez guardado el mapa visualizado en la figura 4, se podrá interactuar con el robot dando

órdenes de posicionamiento.

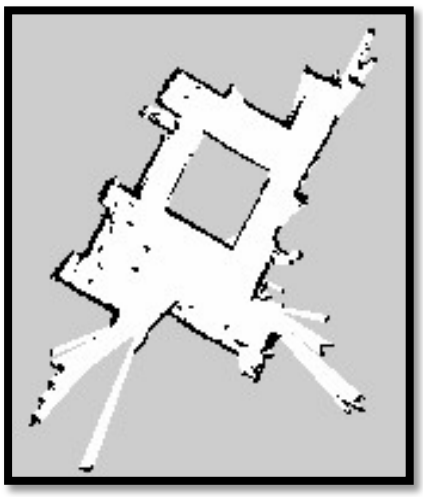


fig. 4*. En esta figura se muestra como Rviz interpreta el mapa elaborado mediante RBPF. El resultado es una vista en planta de la estancia*

3.6 Planificador Global

ROS incorpora un paquete llamado nav_core [29], es una implementación rápida que interpola entre el planificador global y la navegación. Su funcionamiento es muy simple consta de los siguientes parámetros de configuración;

- Búsqueda del camino mínimo a través del sistema de celdas, seguirá los límites de las celdas.
- Utiliza el cálculo de potencial cuadrático para aproximarse a su destino mejorando y suavizando las trayectorias.
- Puede utilizar A*Path[30] o Dijkstra[31] . Esencialmente A* es más rápido ya que trata de buscar el mejor camino según su heurística mientras que Dijkstra explorará todos los caminos posibles ofreciendo el camino mínimo.

El planificador Global está incorporado en la pila de navegación que utiliza move_base que a continuación se justificará su uso

3.7 Planificador Local

Proporciona un control que impulsa la base móvil en el plano. Este controlador sirve para conectar el planificador a la ruta del robot. Dado un mapa, el planificador crea una trayectoria cinemática para llegar desde un punto hasta un lugar objetivo. En el camino planificado crea localmente a su alrededor una función de costes de todas sus posibles trayectorias descartando las más elevadas. Su trabajo será utilizar esta función para controlar las velocidades y una vez evitado el obstáculo retornar a su planificador global. Los pasos son:

- Muestrea de forma discreta el espacio de control del robot en (dx,dy,dtheta).
- Por cada velocidad simula el estado actual del robot para predecir lo que sucedería en un corto periodo de tiempo.
- Evalúa cada trayectoria resultante hacia adelante utilizando una métrica como la proximidad de obstáculos, la proximidad a la ruta global, y la velocidad. Descartando las trayectorias ilegales como colisiones.
- Elije la mejor trayectoria y envía la velocidad a move_base.
- Y repite de nuevo.

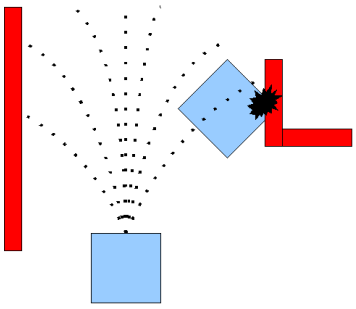


Fig. 5. *Comportamiento del robot en el planificador local.*

3.8 Navegación

Una vez entendidos los conceptos de planificación, el sistema necesita un método para la navegación.
Para ello incorpora move_base que provee una implementación de las acciones [32], que dado un objetivo en el mapa global, intentará alcanzarlo con la base móvil. El nodo move_base encaja con el planificador global y el planificador local para lograr su objetivo. Tal y como se muestra en la figura 6 una vista de alto nivel de sus interacciones.

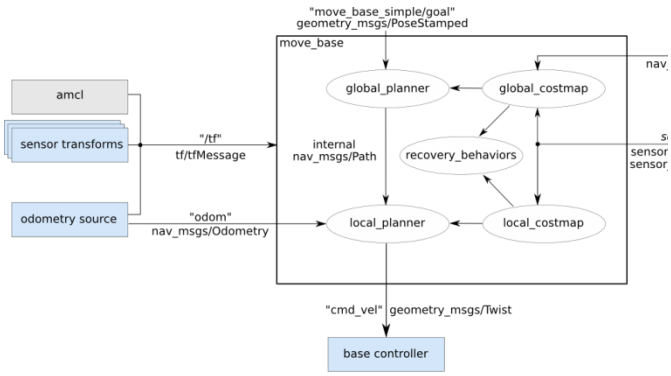


Fig. 6. Pila de navegación del paquete move_base. Imagen extraída de http://wiki.ros.org.

En el caso que el robot sea obstaculizado incluso con su planificación local, tiene un sistema de recuperación de comportamientos para limpiar su espacio de trabajo.
En primer lugar los obstáculos fuera de una región específica se borrarán del mapa del robot. A continuación, si es posible, el robot procederá a rotar y limpiar su espacio, sino lo logra, pasará a un modo más agresivo.

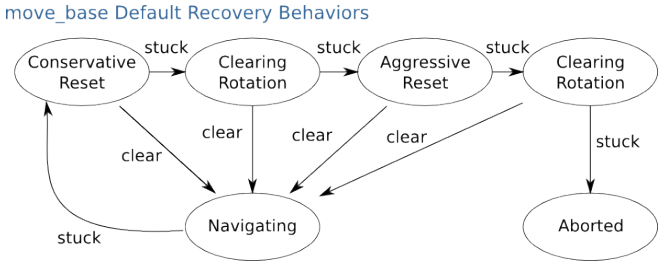


Fig. 8. Recuperación de comportamientos para el paquete move_base. Imagen extraída de http://wiki.ros.org.

3.9 Acoplamiento a base

Previamente la estación de carga debe estar conectada a la corriente al menos en un espacio abierto de 2 metros de ancho y 5 metros de largo. No tiene que haber ningún tipo de obstáculo entre el robot y la estación de carga en el momento acople. Finalmente la estación debe estar lo suficientemente sujeta para que el robot no la arrastre.

El funcionamiento es posible gracias a 3 emisores infrarrojos [33] situados en la estación de carga y 3 receptores infrarrojos en el robot. En la estación, las luces infrarrojas cubren tres regiones en el frente, izquierda, centro y derecha además de dividirlas en dos sub-campos cercanos y lejanos. Cada celda codifica esta información por lo que el robot sabe en todo momento donde se sitúa respecto a su estación.

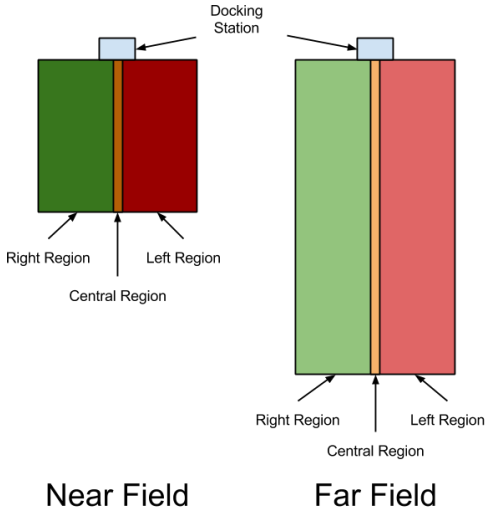


Fig. 7. Región de los sensores infrarrojos para la estación de recarga. Imagen extraída de http://wiki.ros.org.

Una vez visto los emisores falta la parte de los infrarrojos receptores que contiene el robot Kobuki, cuando se coloca dentro del campo de visión de la estación de acoplamiento al menos un sensor recibe el señal IR este esta es capturada y enviada al ordenador mediante el tópico */dock_ir*.

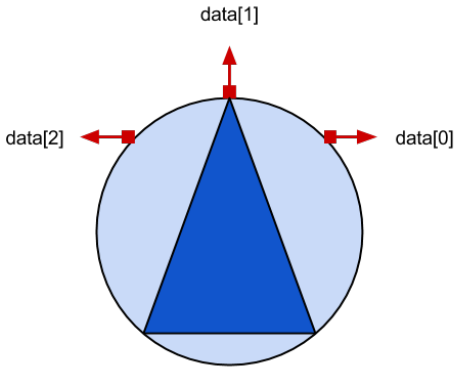


Fig. 9. Posición de los tres receptores en el robot.

Por lo que ya se dispone de la información suficiente para proceder al acoplamiento automático del robot.

Finalmente si el robot se coloca en la región central de la estación de acoplamiento solo es necesario ir hacia adelante.
Simplemente hay que seguir la región central para que el robot pueda atracar. Sin embargo pueden ocurrir 3 situaciones generales.

- El robot se encuentra con un sensor data[2] o data[0] en una región externa pero el otro sensor no se encuentra en ninguna región. En este estado el

robot tiene que pivotar en el sentido del sensor detectado hasta que ambos sensores se encuentren en la misma región, en este punto el robot se encuentra con la perpendicular de la región central y solo tendrá que seguir hacia delante hasta encontrar el data[1] con la región central una vez alcanzado el centro tendrá que girar 45° y seguir hacia adelante para llegar hasta la estación de carga.

- El robot se encuentra con ambos sensores dentro de una misma región. Simplemente tiene que pivotar hasta encontrarse el sensor data[1] con la región central.

- El robot se encuentra con el sensor central en una región no central y los sensores data[2] y data[0] en regiones distintas. Prácticamente el funcionamiento será parecido al del estado anterior, solo tiene que pivotar en contra de la región que se encuentra el sensor data[1] hasta encontrarse con la región central.

3.10 Baterías

Para tener un control de las baterías Turtlebot dispone de dos, la suya interna que dará energía a sus controladores y la batería del ordenador.
Para ello será necesario suscribirse al tópico */mobile_base/sensor/core* y al *laptop_charge*. Gracias a que rospy implementa en el suscriptor una función de retorno, permitirá ejecutar susodicha función una vez le llegué el parámetro, se dispondrá en todo momento del estado de las baterías y se podrán ejecutar las acciones que sean necesarias.

Mediante la función de retorno o dicho como *callback* y el estado actual de las baterías se ha creado un procedimiento. Cuando las baterías refluyan de un cierto parámetro limite (que se podrá modificar siempre cuando se quiera), el robot va a parar su objetivo actual y va a generar un nuevo objetivo para volver cerca de la base, ya que para poder posicionase en la estación de carga tiene que estar alrededor de tres metros de distancia, limite de los sensores IR de la estación.

En el momento que el robot llegue a la estación de carga será el momento de posicionarse y acoplarse, mediante los mensajes kobuki_msgs [34] se podrá saber el estado de proximidad referente a la estación de carga y sus sensores, nombrados en al apartado anterior.

Cabe recalcar que en la estación de carga solo se cargará la batería del robot, no la del ordenador. Una vez llegado a un límite máximo de carga de la batería del robot, retornará su antiguo objetivo pendiente re calculando otra vez su objetivo, ya que la estación de carga puede estar en distintos lugares.

3.11 Sistema de Interfaz de Usuario

Finalmente, para completar la totalidad del proyecto se ha decidido crear una interfaz gráfica, un sistema de control de usuario mediante Pygame y un sistema visual del robot para poder modificar los rangos de carga e introducir los objetivos deseados en tiempo real, también se dispondrá del estado actual del robot.
En esta parte, se ha adjuntado todas las funcionalidades en un solo script Python con rospy, para poder ejecutar el sistema de carga del robot y facilitar el uso al usuario final. También se ha desarrollado un sistema de bloqueo cuando el robot este en estado pendiente de carga, y inversamente, retornará al estado libre cuando esté totalmente cargado, además de un sistema de simulación de carga completa engañando al robot entendiendo que las baterías están totalmente cargadas.

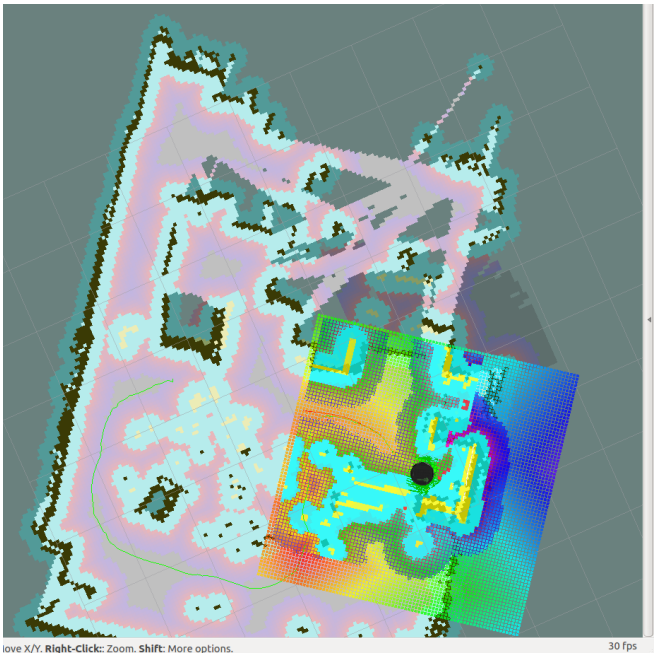
Fig. 10. *Pequeña interfaz de usuario para poder controlar el robot y darle posiciones. También se puede observar el estado actual de las baterías.*

4 RESULTADOS

Kobuki logra completar todos los objetivos del proyecto. Permite alcanzar cualquier coordenada dentro de su mapa global evitando obstáculos dinámicos.

Una vez refluyan ambas baterías el robot pasará a un estado pendiente de carga y ejecutará las acciones pertinentes para alcanzar la estación más cercana. Al llegar cerca de la estación de carga empieza el proceso de posicionamiento.

Fig. 11. *Mapa interno del robot con su ventana local, y el recorrido a ejecutar con el Global planner en el laboratorio de la Escuela de ingeniería de la Universidad Autónoma de Barcelona.*



Al detectar el estado actual de la batería del robot a su máxima carga se desajuntará retrocediendo unos centímetros hacia atrás y dando una vuelta de 180º, para encararse de nuevo a su objetivo pendiente en el caso de haberlo cancelado.

Al llegar al objetivo, actionlib nos responde con sus llamadas de retorno del estado completado o no logrado. Teniendo un feedback continuo gracias a la interfaz de usuario.

5 CONCLUSIONES

En este proyecto se ha logrado entender las funcionalidades básicas del sistema ROS así como el sistema de paso de mensajes. Además se ha estudiado la librería rospy que permite programar e interconectarse rápidamente con Python. Finalmente se ha creado un algoritmo de posicionamiento y paro en tiempo real gracias al paquete actionlib que permitirá editar y obtener el estado de una acción, también se ha usado subscripciones y publicaciones de un tópico para poder mover el robot y saber el estado de los sensores.

Todo ello logrado en su mayoría gracias la implementación de la acción move_base que provee un nivel alto de abstracción del sistema de navegación. No es necesario reinventar la rueda y ROS tiene las soluciones pertinentes para la utilización de acciones, la idea fundamental es que cualquier algoritmo creado en ROS pueda ser escalable, modular y este proyecto dota de todas estas ventajas.

Las posibles extensiones a incorporar en el proyecto son muchas, entre ellas sería tener el gasto estimado en función del recorrido pendiente que le queda hasta llegar a un objetivo, permitiendo la prevención de una posible insuficiencia de energía y así logre llegar a la estación de carga primero, si este no se ve capaz de llegar a su objetivo con el estado actual de las baterías. Otra idea para incorporar sería poder tener un punto de carga dinámico, el robot tendría que lograr buscar la estación o mediante un sistema de posicionamiento en interiores (IPS).

AGRADECIMIENTOS

Quiero mostrar mi profundo agradecimiento al Sr. Ricardo Toledo mi tutor del trabajo, por la ayuda, los recursos necesarios para la elaboración del proyecto, el entorno de trabajo, así como la cantidad de consejos que me ha otorgado.

BIBLIOGRAFÍA

[1] Kit de desarrollo del robot de código abierto para aplicaciones con ruedas. Fecha de la última visita 13/01/2016. http://www.turtlebot.com/.

[2] Kinect, dispositivo de código abierto inicialmente para diseño en videojuegos, pero también es utilizado para mapeo de profundidades. Fecha de la última visita 13/01/2016. https://es.wikipedia.org/wiki/Kinect.

[3] Coche Self-Driving Google. Fecha de la última visita 13/01/2015. https://www.google.com/selfdrivingcar/.

[4] ROS, sistema abierto para robots, provee herramientas, librerías y convenciones. Fecha de la última visita 13/01/2016. http://www.ros.org/.

[5] Willow Garage PR2 un robot para aplicaciones personales, logró abrir puertas, ubicar enchufes y conectarse. Fecha de la última visita 13/01/2016. https://www.willowgarage.com/pages/pr2/overview.

[6] Robotnik FREIGHT, empresa española que utiliza en sus robots el sistema operativo robótico. Fecha de la última visita 13/01/2016. http://www.robotnik.es/robots-moviles/freight/

[7] Esquemas de desarrollo para Universidad Autónoma de Barce-

lona. Fecha de la última visita 13/01/2016. http://www.uab.cat/guiesdocents/2015-16/g102748a2015-16iCAT.pdf.
[8] Sistema operativo abierto GNU/Linux. Fecha de la última visita 13/01/2016. http://www.gnu.org/.
[9] Lenguaje de programación Google, Python. Fecha de la última visita 13/01/2016. https://www.python.org/.
[10] Librería Python para la comunicación con ROS. Fecha de la última visita 13/01/2016. http://wiki.ros.org/rospy.
[11] Universidad Autónoma de Barcelona. Fecha de la última visita 13/01/2016. http://www.uab.cat/
[12] Articulo estándar del IEEE. Fecha de la última visita 13/01/2016. https://www.ieee.org/documents/transactions_journals.pdf
[13] Licencias BSD. Fecha de la última visita 13/01/2016. Https://es.wikipedia.org/wiki/Licencia_BSD
[14] Colaboraciones ROS. Fecha de la última visita 13/01/2016. http://www.ros.org/contribute/
[15] Comunidad Ros. ROS. Fecha de la última visita 13/01/2016. http://answers.ros.org/questions/
[16] Rosbag, sistema de almacenamiento de los mensajes en tópicos. Fecha de la última visita 13/01/2016. http://wiki.ros.org/Bags
[17] Servicios ROS, permite cambiar el modelo a petición respuesta. Fecha de la última visita 13/01/2016. http://wiki.ros.org/Services.
[18] Paquete coordenadas ROS. Fecha de la última visita 13/01/2016. http://wiki.ros.org/tf
[19] Modelado ROS. Fecha de la última visita 13/01/2016. http://wiki.ros.org/robot_model.
[20] Estimación 3D de la posición del robot. Fecha de la última visita 13/01/2016. http://wiki.ros.org/robot_pose_ekf
[21] Estimador probabilístico de localización. Fecha de la última visita 13/01/2016. http://wiki.ros.org/amcl.
[22] Gmapping estimador. Fecha de la última visita 13/01/2016. http://wiki.ros.org/gmapping
[23] Pila de navegación ROS . Fecha de la última visita 13/01/2016. http://wiki.ros.org/navigation
[24] Move_Base, proporciona la implementación de paquetes de acciones ROS. Fecha de la última visita 13/01/2016. http://wiki.ros.org/move_base
[25] Especificaciones Kobuki. Fecha de la última visita 13/01/2016. http://docs.google.com/document/export?format=pdf&id=15k7UBnYY_GPmKzQCjzRGCW-4dIP7zl_R_7tWPLM0zKI.
[26] SSH. Fecha de la última visita 13/01/2016. https://es.wikipedia.org/wiki/Secure_Shell.
[27] Teleoperación ROS. Fecha de la última visita 13/01/2016. http://wiki.ros.org/teleop_twist_keyboard.
[28] Algoritmo filtro de partículas RAO. Fecha de la última visita 13/01/2016. https://en.wikipedia.org/wiki/Particle_filter.
[29] Interface de navegación ROS. Fecha de la última visita 13/01/2016. http://wiki.ros.org/nav_core.
[30] Algoritmo de búsqueda A*. Fecha de la última visita 13/01/2016.https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*.
[31] Algoritmo de caminos mínimos Dijkstra. Fecha de la última visita 13/01/2016. https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra.
[32] Action Lib, librería de acciones Ros. Fecha de la última visita 13/01/2016. http://wiki.ros.org/actionlib.
[33] Funcionamiento Sensor Infrarrojo. Fecha de la última visita 13/01/2016. https://es.wikipedia.org/wiki/Sensor_infrarrojo
[34] Mensajes Kobuki. Fecha de la última visita 13/01/2016. http://wiki.ros.org/kobuki_msgs