Algoritmo de creación mapas mediante el seguimiento autónomo de la pared

José Manuel Bueno Carmona
Department of Systems Engineering and Automation
Universidad Carlos III de Madrid
Madrid, Spain
jobuenoc@ing.uc3m.es

***Abstract*—El objetivo de este documento es comentar las herramientas utilizadas para crear un algoritmo capaz de seguir la pared y a la vez crear el mapa en dos dimensiones de la zona que recorre.**

***Index Terms*—Turtlebot, SLAM, gmapping, LaserScan, Odometría**

I. Introduction

De manera previa a la navegación de cualquier robot por un espacio interior, es necesario que el sistema sepa donde se encuentra. Si no se puede decir el punto en el que comienza el movimiento, no se podrá planificar un camino. Además si, una vez hemos planificado un camino, confiamos sólo en los datos de odometría, es muy probable que el robot se pierda después de recorrer una distancia grande. Es por ello que tener un buen mapa para poder comparar los datos que ve el equipo sensorial con los datos que se tienen previamente guardados ayudará a recolocar al robot y así corregir los errores de odometría. Este proceso suele ser largo y tedioso. Es por ello que los algoritmos que generan el mapa sin necesidad de una persona teleoperando la máquina son muy útiles.

A. Entorno del robot

El robot utilizado es el Turtlebot [1]. Este robot es famoso por ser un producto de bajo coste y además de tener software de código abierto. Es un robot usado en interiores y pensado para que pueda moverse por libertad en zonas preparadas para los seres humanos, como puede ser un casa. Es un equipo perfecto para la creación de aplicaciones y los propios creadores animan a la comunidad científica a compartir los programas y códigos que otras personas han configurado. Cuenta con una cámara Kinect 360, también conocida como Kinect 1.0. Con ella, puede realizar distintas aplicaciones de navegación o detección de objetos gracias a las imágenes que capta, tanto imágenes en color como imagenes de profundidad.

Para las pruebas se van a utilizar mapas de diversos tamaños y distinta disposición de las paredes para así poder comprobar el funcionamiento correcto en varias circunstancias. Estos mapas se componen unicamente de paredes y no tendrán otros componentes ni fijos ni móviles en él. Esta característica puede influir negativamente en la localización del robot dentro del mapa que se está creando.

B. Software utilizado

El desarrollo de los algoritmos se va a realizar apoyándose en la plataforma ROS (*Robotic Operating System*) [2]. Fue creada por la misma compañía que desarrolló el robot Turtlebot, Willow Garage. Su objetivo es simplificar los algoritmos para crear un robot con un comportamiento robusto. Entre los muchos simuladores en los que se puede aplicar esta plataforma está Gazebo, que es el programa escogido para simular los habitáculos que se quieren mapear. Además, el código del programa va a usar como base los siguientes paquetes de código abierto para ROS:

1) *gmapping*
2) *turtlebot_navigation*

Fig. 1. Robot Turtlebot [1]

II. Algoritmo de movimiento

Se va a plantear una solución de seguimiento del robot a lo largo de la pared que realice dos movimientos distintos: Primero, dar una vuelta completa sobre sí mismo para observar los alrededores de la zona en la que se encuentre y, si es necesario, relocalizarse correctamente. Y segundo,seguir la pared para descubrir nuevas zonas del mapa. Ambos movimientos se irán intercalando para crear un mapa lo más fiel posible.

A. Datos necesarios del proceso simulado

Para mover el robot como se requiere, se van a necesitar una serie de datos provenientes del simulador. Estos datos seguirán

la estructura predefinida en ROS, que es la manera de la cual se publica desde Gazebo. Los datos son:

- *nav_msgs/Odometry*. En este tipo de mensaje envía la posición actual del robot según se ha calculado por los componentes mecánicos del mismo. Si solo se utiliza esta variable para determinar la posición del Turtlebot, a la larga se puede tener un error muy grande, pues los datos recibidos no son tan precisos y los errores que tiene se acumulan.
- *sensor_msgs/LaserScan*. Los datos que recoge la cámara Kinect se muestran aquí. Este mensaje es el primordial para la creación del mapa, pues estos datos determinarán la posición de las paredes y ayudarán al robot a relocalizarse, si es necesario. Hay que destacar que este *topic* no lo otorga directamente el simulador. En realidad se utiliza la imagen de profundidad de la Kinect para determinar crear un laser falso. Esto último se realiza con el paquete *depthimage_to_laserscan*.

Por último, hay que destacar que, aparte de los datos que se reciben del simulador, el programa mandará una orden para que se produzca el movimiento usando el mensaje *geometry_msgs/Twist*, el cual necesita datos de como va a ser el movimiento lineal y angular en cada momento.

B. Movimiento de giro

La acción de realizar una vuelta sobre si mismo es crucial, pues si el robot solamente realiza un seguimiento por la pared, es muy probable que el mapa no tenga datos suficientes sobre las zonas en el interior de los mundos creados en el simulador. Para llevar a cabo este movimiento se va a depender de los datos de la odometría. Primero, se va a calcular el ángulo al que se encuentra el robot antes de comenzar el giro, teniendo en cuenta que los cero grados se corresponden con la orientación inicial al abrir el simulador. Sabiendo este ángulo, se puede declarar una orientación objetivo que se corresponda a un ángulo levemente inferior. El giro terminará cuando el ángulo real sobrepase angularmente el objetivo. La velocidad de giro será bastante lenta para prevenir errores de odometría que pueden surgir cuando se usan velocidades elevadas. Durante el algoritmo no se va a priorizar el tiempo con la intención de conseguir un mapa fiable.

C. Movimiento del seguimiento de la pared

El seguimiento de la pared es un comportamiento reactivo, es decir, no se va a mover con un planificador o pensando las zonas a las que debe ir para completar el mapa. Para conseguir este objetivo, van a ser necesarios los datos recabados del *LaserScan*. Para simplificar el código, no serán necesarios todos los puntos calculados por el laser, pues estos superan los 600. En realidad, se realizará una selección de ellos para que se tenga una noción de cómo se debe realizar el movimiento. Se han elegido simplemente cinco valores que se corresponden a lo que la cámara Kinect tiene justo delante de ella y otros cinco correspondientes a uno de los laterales. La elección de que lateral elegir determinará también hacia que lado se va a realizar el seguimiento de la pared. El robot se mueve siguiente unas normas básicas:

1) Si los datos centrales no detectan ningún objeto próximo, se programa el robot para que avance.
2) Si los datos centrales detectan un objeto, el robot no avanza.
3) Si los datos laterales no detectan ningún objeto próximo, se realiza un movimiento angular para acercarse más al lateral correspondiente y, por lo tanto, para encontrar la pared que tiene que seguir.
4) Si los datos laterales detectan un objeto, se intentará alejar levemente de la pared para evitar colisiones.

Con estas normas, se consigue que mientras se recorre el mapa, se mantenga una distancia prudencial con la pared en todo momento y se produzca una parada al llegar a un cambio de pared. Además en esquinas salientes, se producirá un movimiento angular hasta llegar a reencontrar una pared. Durante este proceso se van a volver a utilizar velocidades bajas para prevenir errores innecesarios de odometría por tener velocidades altas. Además, la velocidad lineal debe ser bastante más elevada que la angular para prevenir que el robot colisione en vertices salientes. Es necesario encontrar la relación óptima de velocidades para que no existan choques entre las esquinas, pero que el robot pueda encontrar la pared rápidamente. Por último, se debe determinar cual va a ser el momento de tener que volver a dar una vuelta. En este programa se ha decidido que se va a determinar según la distancia entre el lugar actual y el lugar donde se realizó la última vuelta. Por lo tanto vuelven a necesitar los datos de la odometría, pero en esta ocasión serán necesarias las posiciónes x e y, mientras que la orientación no es necesaria. La posición z no se tiene en cuenta porque los mapas utilizados no tienen diferencia de nivel. Se calculará una distancia euclidiana para determinar si está o no lo suficientemente lejos.

III. ALGORITMO DE *gampping*

Para la creación del mapa se va a usar el algoritmo *gmapping* de ROS [3]. Se trata de un algoritmo SLAM (*Simultaneous Localization and Mapping*), que como su propio nombre indica, es un algoritmo que crea un mapa a la vez que se intenta encontrar en el mismo para corregir los errores de odometría. El resultado es un mapa de dos dimensiones que se va creando a medida que el robot avanza. Con el paquete *gmapping* de ROS, ya se puede usar este algoritmo.

A. Control de parámetros

Dentro del paquete *turtlebot_navigation*, se pueden obtener los parámetros predeterminados para el Turtlebot con distintos sensores, entre ellos la cámara Kinect. No obstante, estos valores no han funcionado correctamente, por lo que se ha decidido variar algunos de ellos para tener un mejor resultado. Estos problemas surgen porque el robot no reconoce dónde se encuentra y realiza de manera erronea la relocalización. Para evitarlo se hace lo siguiente:

- Reducción de los errores de odometría. Se avisa al algoritmo de que los errores de odometría no son muy

altos y por lo tanto se puede fiar más de la posición en la que se encuentra. Esto produce que el proceso de relocalización se produzca en menor medida. Aunque así se consigue un mapa más robusto, es destacable que los últimos trazos de mapa parecen menos adecuados, por lo que estos parámetros no serían aceptables en mapas exageradamente grandes.

- Modificación del borde del mapa. La resolución del mapa es muy importante, pues cuando el tamaño de un pixel es muy pequeño afecta en la relocalización. Cuando el borde del mapa es más fino, es menos probable que el laser coincida con la pared, pues siempre habrá un pequeño error. Cuando esto ocurre, el algoritmo tiende a intentar que el robot se relocalice, pero como esto pasa en un mapa cuyas paredes son todas iguales, ocurre que no suele realizarse una recolocación correcta. Además, si esto ocurre al principio del programa, se estarían perdiendo muchos datos porque es necesario construir mapa en vez de relocalizarse.

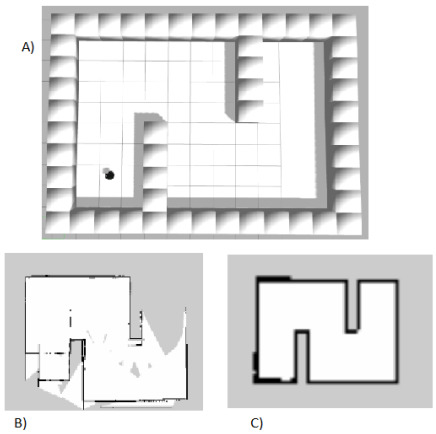

Fig. 2. Ejemplo de mapas creados con distintos parámetros. A) mapa original. B) Mapa con parámetros predeterminados para la cámara Kinect en el paquete *turtlebot_navigation*. C) Mapa con los parámetros elegidos

En la figura 2 podemos observar un ejemplo donde se pueden ver los errores que existen si se utilizan los datos predeterminados y la solución final con estas variable cambiadas.

IV. Resultados

En esta sección se van a destacar los resultados más interesantes al realizar pruebas en distintos mapas. Es notable que el algoritmo funciona mucho mejor en mapas pequeños, como se esperaba. El mapa de la figura 2 es relativamente pequeño, pero se pueden observar unos picos en el borde izquierdo del mapa que, aunque no afecten en gran medida sobre el resultado final, aparecen debido a los errores de odometría ya comentados con anterioridad. Aunque el algoritmo sea lento, se puede crear el mapa con una sola vuelta del robot, es decir, no es normal que necesite repasar zonas del mapa para hacerlo más fiable. Esto es un punto a favor porque si fuese necesario seguir el proceso de *gmapping*, los errores de odometría seguirían aumentando.

Fig. 3. Ejemplo de mapa con zonas interiores vacías

Por otro lado, cuando se quiere realizar el proceso en un mapa más grande, como es el caso de la figura 3, va a surgir un problema cuando existan espacios vacíos muy grandes. Esto es porque los datos del laser de la cámara Kinect no proporciona valores que estén muy alejados, ya que realmente este equipo no es capaz de medir distancias lejanas. En este ejemplo en cuestión, se puede deducir que la zona no tratada es zona vacía. Esto es porque en el caso de que hubiese algún obstáculo, se hubiera detectado en el proceso de dar la vuelta. Además, justo al lado de la zona no detectada, se puede ver que hay un objeto alejado de la pared que el robot sí ha sido capaz de detectar. Siempre que el caso sea el descrito, se puede hacer un tratamiento posterior de la imagen final completando las zonas vacías. No obstante, en mapas incluso más grandes es probable que se creen espacios vacíos cuando no lo son y no se pueda realizar este tratado posterior.

V. Conclusiones

El resultado final es un algoritmo capaz de realizar por su cuenta mapas de tamaño pequeño y medio que, aun siendo un proceso lento, genera un mapa bastante fiel al modelo real. Si fuese necesario generar resultados en lugares de mayor área, es recomendable usar, junto a este algoritmo, un método de planificación de trayectorias que acerque al robot a aquellos lugares que aún no ha podido observar correctamente, ya sea porque se encuentra muy alejado de la pared comprobada o porque, debido a la disposición del mapa, el movimiento de seguimiento de la pared solo se realiza por una pequeña parte del lugar completo. Por último me gustaría destacar de nuevo que si los mapas con los que se están realizando las pruebas tuviesen objetos distintos a los muros o formas de muros más diferenciables, se podrían volver a variar los parámetros para obtener un mejor resultado.

References

[1] Open Source Robotics Foundation, Inc (s.f.). "Turtlebot2". Disponible en: https://www.turtlebot.com/turtlebot2/
[2] ROS (s.f.). About ROS. Disponible en: http://www.ros.org/about-ros/
[3] ROS (s.f.). gmapping. Disponible en: http://wiki.ros.org/gmapping