Universidad de San Andrés, Depto. de Ingeniería

Trabájo Práctico Final

Parte A

I-402 - Principios de la Robótica Autónoma

Prof. Ignacio Mas, Tadeo Casiraghi y Bautista Chasco

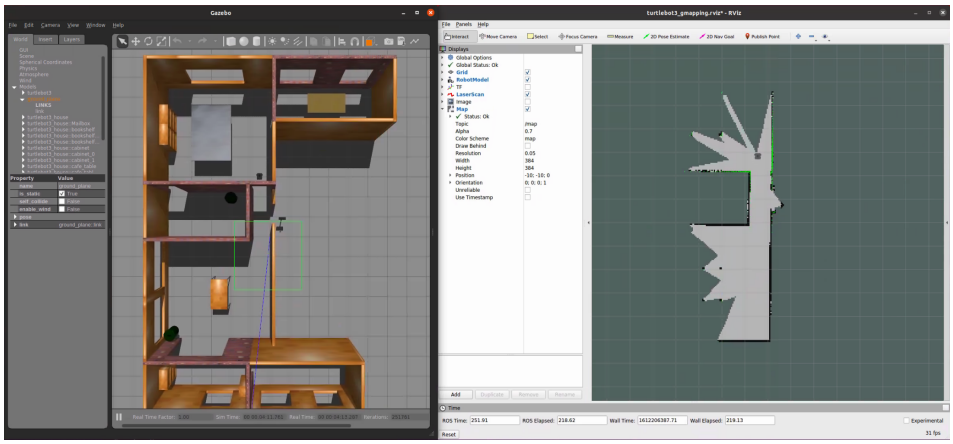
11 de junio de 2025

Fecha límite de entrega: 29/06/25, 23.59hs.
Modo de entrega: Enviar por el Aula Virtual del Campus **en un solo archivo comprimido** el paquete de ROS con código comentado y el informe pdf.

En este trabajo práctico final, los alumnos deberán integrar los principales conceptos abordados a lo largo de la materia de Principios de la Robótica Probabilística mediante la implementación de un sistema autónomo de localización y mapeo (SLAM). Utilizando un robot TurtleBot3 simulado en el entorno de Gazebo, el objetivo en esta primera parte (Parte A) será que el robot explore un entorno desconocido tipo laberinto y construya un mapa del mismo mientras estima su propia posición.

Esta etapa representa un caso de aplicación realista donde convergen múltiples herramientas vistas durante la cursada, como la estimación de estado en presencia de ruido, el uso de sensores inexactos (como el LIDAR) y la fusión de información sensorial. La correcta implementación de esta fase es fundamental, ya que el mapa generado y la precisión de localización serán la base sobre la cual se desarrollará la Parte B, centrada en la navegación autónoma.

1. SLAM - Generación del mapa



En esta primera etapa, los alumnos deberán implementar un sistema de SLAM utilizando el robot TurtleBot3 en un entorno simulado en Gazebo. El objetivo principal es que el robot explore de forma manual o autónoma un entorno tipo laberinto y construya un mapa de ocupación del mismo empleando los datos del sensor LIDAR y la odometría.

Esta etapa permite aplicar de forma práctica los algoritmos de estimación de estado y mapeo estudiados en la materia, ya sea mediante técnicas como SLAM con Filtro de Partículas (FastSLAM) o SLAM con Filtro de Kalman Extendido (EKF SLAM), utilizando herramientas provistas por ROS 2. Al finalizar esta sección, los alumnos deberán exportar el mapa generado y verificar su calidad, ya que servirá como base para la navegación en la siguiente fase del trabajo.

1.1. Preparación del entorno

Antes de comenzar con la implementación de SLAM, es necesario configurar correctamente el entorno de simulación. Para esto se debe tener ya instalado ROS2 Humble, Gazebo y los paquetes de Turtlebot3 ya sea nativamente en una computadora con Ubuntu 22.04 (o cualquier otro OS que lo permita) o mediante Robostack. Recuerden que también es una opción hacer el trabajo en las computadoras del laboratorio de informática del edificio Sullair, pero si se desea hacer eso se deberá tener mucho cuidado de no dejar el código desarrollado en la computadora.

1.2. Lanzamiento del robot y teleoperación

Una vez que el entorno de simulación está correctamente configurado, el siguiente paso es lanzar el robot TurtleBot3 dentro de un escenario tipo laberinto y permitir que explore el entorno para que más adelante pueda construirse el mapa.

1.2.1. Control manual del robot (teleoperación)

Para facilitar la exploración inicial del entorno, se recomienda comenzar con control manual del robot. ROS 2 ofrece un paquete de teleoperación por teclado que puede usarse de la siguiente forma (Recuerde setear el tipo de turtlebot BURGER y el source en cada terminal):

En una terminal ubicada en el directorio del workspace, lanzar el mundo con el robot (si no se hizo previamente):

```
source install/setup.bash
export TURTLEBOT3_MODEL=burger
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

En otra terminal, lanzar la teleoperación:

```
source install/setup.bash
export TURTLEBOT3_MODEL=burger
ros2 run turtlebot3_teleop teleop_keyboard
```

1.2.2. Uso de un entorno personalizado

Junto con la consigna se provee el esqueleto del paquete necesario para correr el trabajo práctico. Además se incluye el modelo png, el launch file del maze, y un maze world de ejemplo. Una vez formados los grupos recibirán por mail su maze world específico a su grupo. Para la correcta instalación y ejecución del código podrán referirse al video que se subirá en las próximas fechas.

1.3. SLAM en ROS 2

En esta etapa, deberán implementar su propio algoritmo de SLAM utilizando los datos publicados por el TurtleBot3 simulado. El objetivo es construir un mapa del entorno mientras se estima simultáneamente la pose del robot. Deben implementar su propio algoritmo de filtros de partículas (FastSLAM), como se trabajó durante la cursada.

1.3.1. Transformaciones de ternas

Las transformaciones pertinentes a los problemas a resolver, como el de Ray Casting, quedan a merced del alumno, sea usando tf2 o transformando manualmente de coordenadas polares a euclideanas, el alumno debe resolverlo como considere. Respecto a los cambios de coordenadas mas complejos como los de Odom a Map, se manejan internamente con TransformBroadcaster, donde se ajusta la transformacion relativa. RViz2 lo recibe de su subscripcion a tf, gracias a esto el mapa no se deforma con el slip de Odom. Esta funcionalidad se entrega ya programada pero se contempla que el alumno entienda del concepto.

1.3.2. Tópico del LIDAR: /scan

El sensor LIDAR del TurtleBot3 publica sus datos en el tópico:

`/scan`

Este tópico es de tipo sensor_msgs/msg/LaserScan e incluye un conjunto de distancias medidas en diferentes ángulos en un plano 2D. Las principales variables que les serán útiles son:

- ranges: lista de distancias medidas en metros
- angle_min, angle_max: rango angular de las mediciones
- angle_increment: resolución angular entre cada lectura

Deben convertir estas lecturas al marco del mundo utilizando la pose actual estimada del robot. Las coordenadas de los obstáculos pueden luego usarse para actualizar un mapa probabilístico, por ejemplo, un mapa de ocupación (gridmap).

1.3.3. Tópico de odometría: /odom

El robot también publica su odometría estimada en:

`/odom`

Este tópico es de tipo nav_msgs/msg/Odometry y proporciona la posición y orientación estimadas del robot. El mensaje contiene:

- pose.pose.position.x, y
- pose.pose.orientation (en cuaterniones)

Para usar esta información como entrada en el filtro de movimiento, deben convertir la orientación a ángulo (yaw) y calcular el desplazamiento relativo entre dos lecturas consecutivas:

```
delta_x = x_{t} - x_{t-1}
delta_y = y_{t} - y_{t-1}
delta_theta = theta_t - theta_{t-1}
```

Esto proporciona la odometría diferencial que se usará para actualizar la estimación de la pose del robot. El alumno deberá guardar los valores de posición t-1.

1.3.4. Movimiento

Para recorrer el mapa puede moverlo manualmente con el teclado.

Sin embargo, como opcional (suma puntos al momento de evaluar la nota final), puede implementar un código que haga que el robot recorra de manera automática todo el laberinto asegurándose que cubrió todo el espacio.

1.3.5. Guardado y publicación del mapa

Una vez satisfecho con el mapa hecho en RViz2, se guarda el mismo con el comando:

```
ros2 run nav2_map_server map_saver_cli -f ~/map
```

Donde *map* es el nombre del archivo como se guardara la imagen y el metadato (.pgm y .yaml). Ese es el output objetivo de la parte 1 del trabajo final, y es el input de la parte 2 donde según el mapa realizado el robot se podrá localizar de forma autónoma y llegar a los objetivos programados por el alumno.

1.3.6. Visualización con RViz

Durante la ejecución del SLAM, es altamente recomendable usar RViz para visualizar los datos de entrada y la salida de su sistema:

Elementos útiles a visualizar:

- /scan: visualización de los rayos del LIDAR.
- /odom: trayectoria del robot estimada por odometría.
- /tf: relaciones entre los marcos (por ejemplo, base_link y odom).
- /map: mapa generado por su algoritmo.

Pueden cargar un archivo .rviz preconfigurado, o configurar la vista manualmente.

1.4. Evaluación del mapa

Para evaluar la calidad y corrección del mapa generado por su algoritmo de SLAM, se tendrán en cuenta los siguientes aspectos:

1.4.1. Coherencia con el entorno simulado

El mapa debe reflejar correctamente la estructura del entorno tipo laberinto utilizado en Gazebo. Las paredes, obstáculos y pasillos deben estar representados de forma clara y precisa en el mapa. No debe haber grandes espacios libres donde en realidad hay paredes, ni paredes que no existen en el entorno.

1.4.2. Resolución y nivel de detalle

La resolución del mapa debe ser suficiente para distinguir los elementos importantes del entorno (puertas, esquinas, obstáculos). El mapa no debe contener ruido excesivo, ni saltos abruptos que dificulten su uso para navegación.

1.4.3. Consistencia temporal

El mapa debe ser estable a lo largo del tiempo: actualizaciones constantes no deben generar cambios erráticos o inconsistencias. El robot debe ser capaz de construir el mapa completo sin perder la ubicación o generar grandes desviaciones.

1.4.4. Uso para navegación

El mapa generado debe permitir la planificación de rutas entre puntos arbitrarios dentro del laberinto. Se probará la capacidad del robot para localizarse y desplazarse correctamente usando el mapa generado.

1.5. Entregables

Los paquetes a resolver se entregarán de forma modular. Serán dos, uno para cada parte del final. Se publicará un videotutorial de la instalación y el lanzamiento de los paquetes de forma nativa en Ubuntu y en Robostack. Con esto se espera que, una vez setteado correctamente el ambiente, el alumno pueda darle launch al paquete e interactuar con Gazebo - RViz2 - la terminal, para poder ir construyendo y debuggeando el nodo a resolver. Únicamente se debe completar el código de python que hace al nodo principal de cada paquete, no se contemplan otros cambios.