PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
IIC-2685 ROBÓTICA MÓVIL

PROYECTO FINAL

Fecha de Presentación: lunes 11 de julio de 2022

Descripción del proyecto

Ha sido un largo semestre!. A través de él, hemos logrado programar los TurtleBots virtuales para darles instrucciones precisas de movimiento (control de bajo nivel), efectuar localización en un mapa conocido (robótica probabilística) y hacer que su robot explore el ambiente (comportamientos reactivos). Ahora que ya comprendemos la matemática subyacente a los sistemas de navegación, en esta oportunidad utilizaremos esta tecnología desde una perspectiva de usuario. Para ello, haremos uso de uno de los paquetes más importantes de ROS, el cual se conoce comúnmente como Stack de Navegación.

Actividad 1: Configuración del Stack de Navegación de ROS

Para configurar el stack de navegación de ROS, comenzaremos creando un nuevo paquete denominado *proyecto_grupo_x_2022*, donde "x", deberá ser reemplazado por su número de grupo.

Dentro del paquete recién creado, ustede deberá construir un archivo launch que inicialice los siguientes componentes:

Simulador de Turtlebot

Para simular las dinámicas de movimiento del TurtleBot, usted deberá levantar el nodo *kobuki_simulator* proporcionado por el paquete *very_simple_robot_simulator* [1].

Simulador de LIDAR

De manera similar al caso anterior, para simular las lecturas realizadas por un LIDAR, usted deberá levantar el nodo *lidar_simulator* incluido dentro del paquete *very_simple_robot_simulator* [1]. Por razones de rendimiento, para el presente proyecto usted deberá utilizar la versión compilada de este nodo, es decir, la que se inicializa e la siguinte forma:

<node pkg="very_simple_robot_simulator" name="lidar_simulator" type="**lidar_simulator**" />

Adicionalmente, deberá configurar los siguientes parámetros utilizando el tag <param> dentro del tag <node>:

effective_hfov = 181
view_depth = 20.0

Para mayor información sobre como utilizar el tag <param>, véase [2].

Servidor de Mapa

Para poder publicar un mapa a partir de un archivo, utilizaremos el nodo *map_server* [3]. Este nodo se encuentra dentro de la paquetería base de ROS, y soporta el formato estándar de archivos que ya conocemos desde nuestras experiencias de laboratorio (yaml y pgm).

Para el caso particular del presente proyecto, usted deberá cargar el mapa *mapa_bodega.yaml* que será proporcionado junto al presente enunciado.

```
image: mapa_bodega.pgm
resolution: 0.050000
origin: [-1.900000, -1.500000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Transformación de Coordenadas

En general, un robot consta de una base móvil más una serie de sensores que lo ayudan a percibir su entorno. Estos sensores pueden ser ubicados en posiciones y ángulos arbitrarios con respecto a la base del robot, y por lo tanto, para conocer la posición del centro del robot dentro de un ambiente, debemos primero *trasladar* las mediciones de los sensores al centro del robot.

Para poder realizar esta *traslación*, es necesario contar con un sistema de coordenadas para cada entidad (base móvil, lidar, odometría, mapa, etc), y además, conocer como se relacionan cada uno de estos sistemas entre sí. A modo de ejemplo, en la figura 1 se muestra un robot móvil junto a los sistemas de coordenadas más comunes que podremos encontrar en navegación 2D.

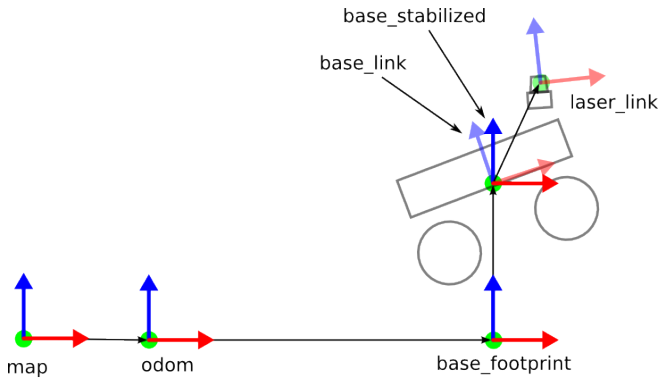

Figure 1: Definición de sistemas de coordenadas para un robot móvil.

Según lo anterior, para poder deducir la pose del centro del robot a partir de las mediciones del Lidar, tendremos que informar acerca de la diferencia de distancia y ángulo que existe entre el sistema de coordenadas de la base móvil y el sistema de coordenadas del Lidar. Para ello, utilizaremos el nodo de trasnformación *tf* de ROS [4]. Éste recibirá como parámetro de entrada la relación que existe entre la base móvil y nuestro Lidar mediante un vector del tipo: (x, y, z, yaw, pitch, roll).

Pregunta: En el caso particular del simulador, consideraremos que el Lidar se encuentra ubicado justo en el centro del robot. Según esto, ¿ Qué valor debería tomar el vector de transfomación a publicar por tf ?.

Visualizador de Información

Cuando trabajamos con robots móviles, ya sea en etapa de configuración o simplemente en operación normal, es de suma utilidad visualizar como van evolucionando las varibales del sistema mientras el robot se va desplazando en el ambiente. Para el caso de sistemas de navegación, en general necesitaremos visualizar:

mapa, robot dentro del mapa, partículas generadas por el localizador, información entregada por los sensores, etc.

Para poder realizar esta tarea de manera sencilla, ROS provee el visualizador *rviz* [5]. Este visualizador permite leer directamente de los tópicos del sistema y convertir los datos recibidos en información visual.

Localizador Monte Carlo Adaptivo (AMCL)

En el Laboratorio 3, usted programó un algoritmo de localización basado en la técnica *Monte-Carlo*, el cual permitía localizar al robot dentro de un mapa. En esta oportunidad, utilizaremos el sistema de localización de ROS, el cual implenta una versión optimizada el *filtro de partículas* denominada *Adaptive Monte Carlo Localization* (AMCL) [6].

Para utilizar este algoritmo, incluiremos el nodo *amcl* y lo configuraremos a través del tag <param> con los siguientes parámetros:

```
odom_model_type: "diff"
use_map_topic: true
initial_pose_x: 5.325
initial_pose_y: 0.814
initial_pose_a: 1.570
laser_max_beams: 181
min_particles: 250
```

"Pregunta": Investigue cuál es la utilidad de estos parámetros y explíquelos brevemente en su presentación.

Stack de Navegación de ROS

Una vez que tenemos información de los sensores, mapa del entorno, transformación de coordenadas y localización del robot, solo nos falta calcular la mejor ruta hasta un destino, y luego, ejecutar su seguimiento evadiendo obstáculos estáticos y dinámicos.

Para todas estas tareas (y más!) utilizaremos el stack de navegación de ROS que se encuentra implementado por el nodo *move_base* [7]. Debido a su complejidad, este nodo posee un gran número de parámetros de configuración que no alcanzaremos a analizar con detalle en el presente proyecto. Sin embargo, junto a este documento usted encontrará un archivo comprimido que contiene un conjunto de archivos de configuración que definirán los parámetros que necesitamos para nuestra actividad.

El archivo en cuestión se denomina *param.tar.gz*. Al descomprimirlo, se generará un directorio de nombre *param*, el cual deberá copiar dentro del directorio base de su paquete.

Una vez realizada esta copia, deberá agregar el nodo *move_base* en su archivo launch, con el siguiente parámetro de configuración:

```
controller_frequency: 5.0
```

Además, dentro del mismo tag *node* utilizado para levantar *move_base*, deberá cargar los archivos de configuración contenidos en el directorio param mediate el tag <rosparam>. Para ello, considere la siguiente lista de atributos por cada tag:

```
file: path_to/costmap_common_params.yaml command="load" ns="global_costmap"
file: path_to/costmap_common_params.yaml command="load" ns="local_costmap"
file: path_to/local_costmap_params.yaml command="load"
file: path_to/global_costmap_params.yaml command="load"
file: path_to/dwa_local_planner_params.yaml command="load"
file: path_to/move_base_params.yaml command="load"
```

Pregunta: ¿ En qué se diferencia el tag <param> del tag <rosparam> ?

Demostración

Para la demostración, usted deberá:

1. Levantar todos los nodos antes señalados mediante el lanzamiento de un archivo launch confeccionado por usted.
2. Una vez levantado el conjunto de nodos, ustede deberá visualizar:
    - Mapa
    - Pose del robot proporcionada por AMCL
    - Partículas utilizadas por AMCL
    - Lecturas del Lidar

Pregunta: ¿ Las lecturas del Lidar visualizadas son coherentes con lo esperado ?. En caso de que no lo sean, ¿ por qué ocurre esto ?.

Actividad 2: Moviendo nuestro TurtleBot virtual

Una vez configurados los parámetros e inicializados los nodos, el sigiuente paso es enviar al TurtleBot hasta una pose deseada. Para ellos deberá ejecutar dos pasos:

1. Establecer la pose inicial del robot mediante interfaz gráfica rviz.
2. Enviar el robot hasta alguna pose de destino de su elección, utilizando la interfaz gráfica rviz.

Antes de ejecutar los pasos anteriores, agregue al visualizador el tópico que permite recibir el plan de navegación global.

Pregunta: ¿ Cómo describiría la navegación observada ? (fluida, errática, etc.). ¿ Qué podría estar causando los comportamientos observados ?.

Actividad 3: Interactuando con el stack mediante Python

Al llegar a este punto, usted ya debería ser capaz de visualizar los principales tópicos de su sistema de navegación, así como también, de mover el TurtleBot utilizando rviz.

Ahora, es momento de aprender a integrar el stack de navegación de ROS a sus scripts de Python. Para ello, deberá programar un código que sea capaz de mover al robot hasta alcanzar una secuencia de 3 poses a su elección.

Para hacer más fácil su desarrollo, junto al presente enunciado se le entregará un ejemplo sencillo de utilización de la API actionlib (*nav_stack_example.py*), el cual le permitirá enviar poses objetivo al robot.

References

[1] https://github.com/gasevi/very_simple_robot_simulator

[2] http://wiki.ros.org/roslaunch/XML/param

[3] http://wiki.ros.org/map_server

[4] http://wiki.ros.org/tf

[5] http://wiki.ros.org/rviz

[6] http://wiki.ros.org/amcl

[7] http://wiki.ros.org/move_base