Práctica 5 - Navegación con Nav2 Simple Commander

Robótica de Servicios

Máster en Robótica e Inteligencia Artificial

Resumen

Esta práctica está asociada con la instalación, configuración y uso de herramientas de navegación en ROS 2, específicamente el paquete Navigation 2 [1] y Nav2 Simple Commander para el control de tareas de navegación mediante scripts en Python. Los objetivos planteados son:

- Familiarizarse con la instalación y configuración de los paquetes de navegación en ROS 2.
- Configurar un entorno de simulación con Turtlebot3 en Gazebo.
- Enviar objetivos de navegación al robot mediante Nav2 Simple Commander.
- Modificar parámetros en los archivos de configuración de Nav2 para personalizar el comportamiento de navegación.

Requisitos

Será necesario disponer de un entorno de trabajo base. La recomendación por simplicidad es la siguiente:

- Robot Operating System 2 (ROS 2)
  - Ubuntu 22.04
  - ROS 2 Humble y Gazebo

Workspace en ROS

Utilizando el repositorio de ejemplos: `https://github.com/fjrodl/navigation2_ignition_gazebo_turtlebot3` y la guía proporcionada, sigue los pasos indicados para completar esta práctica.

Ejercicio: Uso de Nav2 Simple Commander

Este ejercicio guiará al estudiante en el uso de `Nav2 Simple Commander` y en la modificación de los parámetros de navegación para controlar el comportamiento de navegación del robot.

Ejemplo Práctico

A continuación, se detallan los pasos para realizar un ejemplo práctico usando `Nav2 Simple Commander` en Python, en el cual el robot se dirigirá a una posición especificada y luego seguirá una serie de puntos de ruta:

1. Asegúrate de tener los paquetes de `Nav2` y `Nav2 Simple Commander` instalados. Si es necesario, instala los paquetes correspondientes:

```
    sudo apt install ros-humble-navigation2 ros-humble-nav2-simple-commander

```

2. Inicia el entorno de simulación y los paquetes de navegación:

```
    ros2 launch nav2_bringup navigation_launch.py use_sim_time:=True

```

3. Crea un script en Python para controlar el robot con `Nav2 Simple Commander`. Guarda el archivo como `simple_commander_example.py`:

```python
from nav2_simple_commander.robot_navigator import BasicNavigator, NavigationResult
from geometry_msgs.msg import PoseStamped
import rclpy
import time

def main():
    \# Inicializa el sistema ROS 2
    rclpy.init()


    navigator = BasicNavigator()


    goal_pose = PoseStamped()
    goal_pose.header.frame_id = 'map'
    goal_pose.pose.position.x = 2.0
    goal_pose.pose.position.y = 3.0
    goal_pose.pose.orientation.w = 1.0


    navigator.goToPose(goal_pose)


    while not navigator.isTaskComplete():
        feedback = navigator.getFeedback()
        if feedback:
            print("Distancia restante al objetivo: ", feedback.distance_remaining)
        time.sleep(1)


    result = navigator.getResult()
    if result == NavigationResult.SUCCEEDED:
        print("El robot alcanzo el objetivo")
    elif result == NavigationResult.CANCELED:
        print("La navegacion fue cancelada.")
    else:
        print("La navegacion fallo.")


    rclpy.shutdown()

if __name__ == '__main__':
    main()


```

4. Ejecuta el script:

```
python3 simple_commander_example.py

```

Archivos Importantes de Nav2 y Modificación de Parámetros

Para ajustar el comportamiento de navegación en Nav2, puedes modificar varios archivos de configuración en `nav2_bringup`. Algunos de los archivos clave son:

- **costmap_parameters.yaml**: Este archivo controla los parámetros del costmap (mapa de costos) global y local. Para cambiar el valor de inflación de obstáculos, busca el siguiente bloque de código dentro de `costmap_parameters.yaml`:

```
inflation_layer:
  plugin: "nav2_costmap_2d::InflationLayer"
  inflation_radius: 0.5  >>> Cambia este valor segun el tamanyo deseado
  cost_scaling_factor: 10.0

```

  Cambia el valor de `inflation_radius` para ajustar la zona de inflación de los obstáculos. Un valor mayor ampliará el área de evitación de obstáculos.

- **planner_server.yaml**: Contiene configuraciones relacionadas con el planificador global. Aquí puedes modificar parámetros como la tolerancia al objetivo o el tipo de planificador (por ejemplo, cambiar entre `Dijkstra` y `A*`).

- **controller_server.yaml**: Este archivo contiene configuraciones del controlador local que gestiona cómo se ajusta el robot al camino. Puedes ajustar parámetros como la velocidad máxima del robot o el algoritmo de control utilizado (por ejemplo, `DWB`).

- **behavior_tree_navigator.xml**: Define la estructura del árbol de comportamiento utilizado en Nav2 para la toma de decisiones durante la navegación. Puedes modificarlo para añadir o quitar comportamientos específicos según tus necesidades.

Parámetros Clave para la Configuración de Nav2

Al configurar la navegación en `Nav2`, además de los anteriores, existen otros parámetros importantes en el sistema de mapas de costos (*costmap*) y en los servidores de planificación y control. A continuación se presentan algunos de los más relevantes:

1. Parámetros del Costmap

- **resolution**: Define el tamaño de cada celda en el mapa de costos, en metros. Una resolución más baja permite un mapa más detallado, pero puede aumentar el consumo de recursos.

- **robot_radius**: Especifica el radio del robot en metros, utilizado para definir la zona alrededor de él que se debe mantener libre de obstáculos.

- **cost_scaling_factor**: Factor de escala que determina la rapidez con la que aumenta el costo cerca de los obstáculos. Un valor alto hace que el robot sea más conservador al acercarse a obstáculos.

- **obstacle_range**: Define la distancia máxima desde el robot en la que los obstáculos se registrarán en el costmap. Cualquier obstáculo fuera de este rango será ignorado.
- **raytrace_range**: Rango en el que se realiza la detección de obstáculos mediante ray tracing, útil para detectar paredes o límites en áreas grandes.
- **observation_sources**: Define las fuentes de datos de sensores (como `scan` o `pointcloud`) que se utilizan para identificar obstáculos. Cada sensor tiene sus propios parámetros específicos, como `obstacle_range` y `raytrace_range`.

2. Parámetros del Planner Server

- **planner_plugins**: Lista de plugins de planificación disponibles, como `nav2_navfn_planner/NavfnPlanner` o `smac_planner/SmacPlannerHybrid`. La selección del planificador afecta la eficiencia y la suavidad de las rutas.
- **tolerance**: Tolerancia en metros para alcanzar el objetivo. Establecer una mayor tolerancia puede hacer que la planificación sea más rápida, aunque menos precisa.
- **downsample_costmap**: Opción para reducir la resolución del costmap durante la planificación. Esto puede aumentar la velocidad del planificador en mapas grandes.
- **allow_unknown**: Permite o impide la planificación en áreas desconocidas del mapa (áreas sin exploración previa). Este parámetro es importante si el robot necesita explorar nuevas áreas.

3. Parámetros del Controller Server

- **controller_plugins**: Define los controladores utilizados, como `dwb_controller/DWBLocalPlanner` o `teb_local_planner/TEBPlanner`. Diferentes controladores tienen distintas capacidades de evasión y seguimiento de trayectorias.
- **goal_checker_plugins**: Plugins que verifican si el objetivo se ha alcanzado. Configurar esto puede ayudar a definir condiciones de llegada más flexibles o estrictas.
- **max_velocity** y **min_velocity**: Controlan las velocidades máxima y mínima en las direcciones lineal y angular, lo cual afecta la agilidad y seguridad del robot.
- **acc_lim_x**, **acc_lim_y** y **acc_lim_theta**: Límites de aceleración en los ejes `x`, `y` y `theta` (angular). Estos valores son importantes para asegurar que el robot acelere y desacelere de manera segura.
- **min_obstacle_distance**: Especifica la distancia mínima a un obstáculo en la que el robot debe detenerse, útil para evitar colisiones en espacios ajustados.

4. Parámetros de Recuperación y Comportamientos Especiales

- **recovery_behaviors**: Configura comportamientos de recuperación automática, como giros en su lugar o retroceso, cuando el robot se queda atascado.
- **cycle_frequency**: Frecuencia de actualización del ciclo de recuperación. Un valor más alto permite que el robot recupere más rápidamente, pero consume más recursos.
- **recovery_plugin**: Define el tipo de comportamiento de recuperación que se debe activar, como el retroceso o giro en áreas bloqueadas.

Ejemplo de Configuración en `costmap_parameters.yaml`

```yaml
inflation_layer:
  plugin: "nav2_costmap_2d::InflationLayer"
  inflation_radius: 0.5
  cost_scaling_factor: 10.0

obstacle_layer:
  plugin: "nav2_costmap_2d::ObstacleLayer"
  observation_sources: scan
  scan:
    topic: /scan
    max_obstacle_height: 2.0
    obstacle_range: 2.5
    raytrace_range: 3.0
```

Cada parámetro afecta directamente cómo el robot percibe y navega por su entorno. Configurarlos correctamente permite mejorar la precisión de la navegación, reducir el consumo de recursos y adaptar el comportamiento del robot a entornos específicos.

Preguntas

1. ¿Cuál es la función de `Nav2 Simple Commander` y cómo ayuda en la automatización de tareas de navegación?
2. ¿Qué efecto tiene el parámetro `inflation_radius` en el archivo `costmap_parameters.yaml` y cómo afecta la navegación del robot?
3. ¿Cómo podrías utilizar un script en `Nav2 Simple Commander` para automatizar una prueba de navegación en un entorno simulado?
4. Describe un caso en el que modificar el archivo `planner_server.yaml` podría mejorar el rendimiento de navegación.

.

Referencias

[1] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2718–2725, 2020.