Proyecto: Navegación Avanzada con Nav2, Voz y Múltiples Planificadores

Robótica de Servicios

Máster en Robótica e Inteligencia Artificial

En este proyecto, se propone desarrollar una aplicación de navegación avanzada utilizando ROS 2 y Nav2, donde el robot debe seguir una serie de puntos de ruta (waypoints) leídos desde un archivo. Además, durante el trayecto, el robot interactuará mediante voz, ya sea proporcionando información o solicitando input del usuario. Se explorará también el uso de múltiples planificadores locales (Local Planners) para optimizar la navegación en diferentes situaciones.

Repositorio General: https://classroom.github.com/a/nuVnVQv3

Objetivos

- Implementar la lectura de una lista de puntos de ruta desde un archivo y enviar esos objetivos al robot utilizando Nav2 Simple Commander.
- Integrar capacidades de síntesis y reconocimiento de voz para que el robot pueda hablar y recibir comandos de voz durante la navegación.
- (Opcional) Configurar y utilizar múltiples planificadores locales en Nav2, cambiando entre ellos dinámicamente según las necesidades de la navegación.

Instrucciones

1. **Lectura de Waypoints desde un Archivo**
   - Crea un archivo de texto (por ejemplo, `waypoints.yaml`) que contenga al menos 5 posiciones objetivo (waypoints) en formato YAML.
   - Cada waypoint debe incluir las coordenadas `x`, `y` y la orientación `theta`.
   - Implementa un script en Python que lea este archivo y almacene los waypoints en una lista.
   - Utiliza Nav2 Simple Commander para enviar secuencialmente cada waypoint al robot y monitorear el progreso.
   - Utiliza Nav2 Simple Commander para enviar aleatoriamente a cada waypoint al robot y monitorear el progreso.
2. **Integración de Voz**
   - Utiliza paquetes como `ros_speech_commands` o `speech_recognition` para el reconocimiento de voz, y `sound_play` o `festival_tts` para la síntesis de voz en ROS 2.
   - **Extra:** Puedes utilizar el chatbot https://github.com/mgonzs13/chatbot_ros
   - Programa el robot para que, al llegar a cada waypoint, reproduzca un mensaje de voz informando su estado o preguntando al usuario si debe proceder al siguiente punto.

- Implementa la capacidad de recibir comandos de voz del usuario para decidir si continuar, repetir el punto actual o detener la navegación.

3. **Uso de Múltiples Planificadores Locales** (Opcional)

   - Investiga los diferentes planificadores locales disponibles en Nav2, como `DWB`, `TEB` y `RPP`.
   - Configura el sistema para permitir cambiar entre diferentes planificadores locales durante la ejecución. Esto implicará modificar los archivos de configuración y/o utilizar plugins.
   - Define criterios para cambiar de planificador durante la navegación (por ejemplo, cambiar a un planificador específico en áreas congestionadas o al acercarse a obstáculos).
   - Implementa la lógica en el script para cambiar dinámicamente el planificador local según los criterios definidos.

Implementación

Ejemplo de Script en Python

A continuación, se presenta un esquema básico del script en Python que puede servir de punto de partida:

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

def read_waypoints(file_path):
    with open(file_path, 'r') as f:
        waypoints = yaml.safe_load(f)
    return waypoints

def main():
    rclpy.init()

    navigator = BasicNavigator()

    # Espera hasta que el sistema de navegacion este listo
    navigator.waitUntilNav2Active()

    # Leer waypoints desde el archivo
    waypoints = read_waypoints('waypoints.yaml')

    for idx, wp in enumerate(waypoints):
        goal_pose = PoseStamped()
        goal_pose.header.frame_id = 'map'
        goal_pose.pose.position.x = wp['x']
        goal_pose.pose.position.y = wp['y']
        goal_pose.pose.orientation.z = wp['z']
        goal_pose.pose.orientation.w = wp['w']

        # Enviar objetivo al robot
        navigator.goToPose(goal_pose)

        while not navigator.isTaskComplete():
            feedback = navigator.getFeedback()
            if feedback:
                print(f"Dist. restante al WP {idx+1}: {feedback.distance_remaining:.2f} m")
```

```
            time.sleep(1)

        result = navigator.getResult()
        if result == NavigationResult.SUCCEEDED:
            print(f"Objetivo {idx+1} alcanzado.")
            # Aqui puedes anadir codigo para que el robot hable o solicite input
        else:
            print(f"Fallo al alcanzar el objetivo {idx+1}.")
            break

        # Logica para cambiar de planificador local si es necesario
        # changePlanner('nombre_del_planificador')

    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Configuración de Múltiples Planificadores Locales

Para permitir el cambio de planificadores locales, debes:

- Modificar el archivo de parámetros del controlador local (`controller_server.yaml`) para incluir múltiples planificadores.
- Asegurarte de que los plugins de los planificadores están correctamente configurados.
- Implementar una función en tu script que pueda cambiar dinámicamente el planificador activo utilizando servicios o acciones proporcionados por Nav2.

Ejemplo de código en Python para cambiar de planificador

Para cambiar el planificador dinámicamente en Nav2, puedes usar la API de servicios de ROS 2 en Python. A continuación, se muestra un ejemplo donde se cambia el planificador configurando el parámetro `planner_id` en el nodo `planner_server` para alternar entre diferentes plugins de planeación.

Este ejemplo utiliza el servicio `set_parameters` de ROS 2 para cambiar el parámetro dinámico del planificador en Nav2.

Este código asume que tienes al menos dos plugins de planificación configurados en `planner_server`, por ejemplo, `GridBased` y `SmacPlanner`.

```
import rclpy
from rclpy.node import Node
from rcl_interfaces.srv import SetParameters
from rcl_interfaces.msg import Parameter, ParameterType, ParameterValue

class ChangePlanner(Node):
    def __init__(self):
        super().__init__('change_planner_client')
        self.cli = self.create_client(SetParameters, '/planner_server/
    set_parameters')
        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Esperando a que el servicio de set_parameters
    este disponible...')

    def set_planner(self, planner_name):
        req = SetParameters.Request()
```

```
        # Definimos el nuevo parametro del planificador
        param = Parameter()
        param.name = 'planner_id'
        param.value = ParameterValue(type=ParameterType.PARAMETER_STRING,
    string_value=planner_name)

        # Agregamos el parametro a la solicitud
        req.parameters = [param]

        # Llamamos al servicio para cambiar el planificador
        future = self.cli.call_async(req)
        rclpy.spin_until_future_complete(self, future)

        if future.result() is not None:
            self.get_logger().info(f'Planificador cambiado a: {planner_name}')
        else:
            self.get_logger().error('Error al cambiar el planificador')

def main(args=None):
    rclpy.init(args=args)

    # Creamos el nodo y el cliente de servicio
    change_planner_client = ChangePlanner()

    # Ejemplo: cambiar al planificador GridBased
    change_planner_client.set_planner('GridBased')

    # Cambiar despues al planificador SmacPlanner (puedes alternar como necesites)
    change_planner_client.set_planner('SmacPlanner')

    change_planner_client.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Listing 1: Código Python para cambiar el planificador

Explicación

- **Cliente de servicio**: `change_planner_client` crea un cliente para el servicio `/planner_server/set_parameters`, que permite cambiar parámetros en tiempo de ejecución.
- **Parámetro dinámico**: Cambiamos el valor de `planner_id` a través de `SetParameters` para indicar qué plugin de planificador usar.
- **Uso del servicio**: Llamamos al servicio para establecer el nuevo planificador, y el cambio se reflejará inmediatamente si el planificador está bien configurado en `planner_server`.

Configuración adicional

En el archivo de configuración YAML de `planner_server`, asegúrate de tener definidos múltiples plugins de planificación. Por ejemplo:

```
planner_server:
```

```yaml
  ros__parameters:
    planner_plugins: ["GridBased", "SmacPlanner"]
    GridBased:
      plugin: "nav2_navfn_planner/NavfnPlanner"
    SmacPlanner:
      plugin: "nav2_smac_planner/SmacPlanner"
```

Listing 2: Ejemplo de configuración YAML para múltiples planificadores

Este ejemplo proporciona una forma básica de cambiar el planificador en tiempo de ejecución.

Preguntas

1. ¿Cómo se implementó la lectura de waypoints desde el archivo y cómo se integró con Nav2 Simple Commander?
2. ¿Qué paquetes o herramientas se utilizaron para la síntesis y reconocimiento de voz en ROS 2?
3. Describe el proceso para configurar y cambiar entre múltiples planificadores locales en Nav2 antes o durante la ejecución. ¿Cómo se realizaría este proceso desde fichero de configuración?
4. (Opcional) ¿Qué criterios se utilizaron para cambiar de planificador local? Si lo has cambiado cómo afectó esto al comportamiento del robot cuando navega de forma normal entre Waypoints. ¿Y si se encuentra un objeto en su ruta? ¿Y si se queda atascado entre varios objetos que le impiden pasar?

Entrega

En tu repositorio se espera que entregues:

- El código fuente completo, incluyendo scripts y archivos de configuración modificados.
- Un informe breve en el README.md explicando:
  - El diseño del sistema y cómo se implementaron las funcionalidades solicitadas.
  - Las instrucciones para ejecutar el proyecto.
  - Los desafíos encontrados y cómo se resolvieron.
  - Respuestas a las praguntas anteriores.
- (Opcional) Un video demostrativo mostrando el robot navegando entre los waypoints, interactuando mediante voz y cambiando de planificador local.

Sugerencias

- Asegúrate de manejar adecuadamente las excepciones y errores en el script, especialmente al interactuar con el sistema de voz y al cambiar de planificador.
- Si no te da tiempo a hacerlo todo, evita interacción por voz y realiza un input desde teclado.
- Prueba el sistema en diferentes escenarios para evaluar el rendimiento de los distintos planificadores locales. Esto es, mete algún objeto en tu mapa que evite que el comportamiento sea sistemático en todas las iteraciones.

- Recuerda visitar la web de Nav2 para saber más: https://docs.nav2.org/commander_api/index.html
- Documenta cualquier ajuste o problema encontrado durante el desarrollo para incluirlo en el informe.