Modelado y Simulación de Robots - GIRS

Práctica 3: Simulación de Robots usando middleware

Anexo: Lanzador de la simulación y del robot

En este documento se muestran fragmentos de código para definir el fichero `*.launch.py` para lanzar el simulador con el mundo adecuado así como el robot, RViz, transformaciones de los topics, etc.

Lanzador principal

Se implementa la función principal que gestiona el lanzamiento de los nodos, incluyendo la carga de sus configuraciones y su instanciación correspondiente.

Configuración del entorno

Primero se define la variable `moveit_config` que almacena la configuración del robot. Esta información se suele utilizar para configurar nodos como `move_group` o `rviz2`. No obstante, en este caso no se utiliza y solo se comparte para tener diferentes alternativas al construir el *launch file*.

A continuación se declara el parámetro `use_sim_time` con valor por defecto `true`. Esto será utilizado para configurar los nodos lanzados. Es importante en este caso tener este valor activado para asegurar una correcta sincronía.

Además se especifican las rutas de los modelos. En este ejemplo, se muestra cómo añadir las variables de entorno del sistema `GZ_SIM_RESOURCE_PATH` y `GZ_SIM_MODEL_PATH` con las rutas correspondientes mediante la función definida al final del documento `get_model_paths`.

```
def generate_launch_description():
    moveit_config = MoveItConfigsBuilder("rover",
    package_name="rover_moveit_config").to_moveit_configs()
    declare_sim_time = DeclareLaunchArgument(
        'use_sim_time', default_value='true',
        description="use_sim_time simulation parameter"
    )
    model_path = ''
    resource_path = ''

    pkg_path = get_package_share_directory('msr_robot')
    model_path += join(pkg_path, 'models')
    resource_path += pkg_path + model_path

    if 'GZ_SIM_MODEL_PATH' in environ:
        model_path += pathsep+environ['GZ_SIM_MODEL_PATH']
    if 'GZ_SIM_RESOURCE_PATH' in environ:
        resource_path += pathsep+environ['GZ_SIM_RESOURCE_PATH']

    model_path = get_model_paths(['msr_robot'])
```

Lanzar simulación, colocar al robot en el mundo y publicadores de estado del robot

En este ejemplo, se lanza el simulador utilizando la función `start_gzserver`, definida al final del archivo. El uso del método `OpaqueFunction` garantiza que esta acción se ejecute durante la evaluación del lanzamiento, en el momento en que se llama, asegurando que tareas previas como la configuración de rutas ya se hayan completado correctamente.

A continuación, se inserta el robot en la simulación (proceso conocido como *spawning the robot*). Para ello, se utiliza el nodo `create` del paquete `ros_gz_sim`, donde es posible configurar parámetros como el nombre del modelo, el tópico en el que se publica la descripción del robot (`/robot_description`) y la opción de utilizar el tiempo de simulación.

Finalmente, se lanza el nodo encargado de publicar el estado del robot. Esto se realiza mediante el `launch rsp.launch` .py, generado previamente con el *MoveIt Setup Assistant*.

```
start_gazebo_server_cmd = OpaqueFunction(function=start_gzserver)

   # Spawning robot
gazebo_spawn_robot = Node(
    package="ros_gz_sim",
    executable="create",
    output="screen",
    arguments=[
        "-model", "rover",
        "-topic", "robot_description",
        "-use_sim_time", "True",
    ],
)


robot_description_launcher = IncludeLaunchDescription(
   PathJoinSubstitution(
       [FindPackageShare("rover_moveit_config"), "launch",
        "rsp.launch.py"]
   ),
)
```

Otros nodos: RViz y adaptadores de interfaz

A continuación, se lanza RViz para visualizar la información generada durante la simulación, y se configuran las interfaces de los mensajes correspondientes. Para ello, se invoca el nodo `rviz2`, definido en el paquete del mismo nombre. Para mostrar la información relevante, previamente hemos creado un archivo de configuración `robot.rviz`, que contiene los detalles de visualización del sistema, tales como las cámaras, el sensor láser, el modelo del robot, entre otros elementos.

El nodo `parameter_bridge` del paquete `ros_gz_bridge` se utiliza para mapear los tópicos definidos en Gazebo, permitiendo que estos sean accesibles desde ROS 2. Para facilitar la modularidad y el mantenimiento, la configuración necesaria para realizar esta transformación se encuentra almacenada en un archivo de configuración (consultar el documento **Ejemplo configuración bridge**).

Para la conversión de los mensajes de las cámaras, se recomienda utilizar el paquete `ros_gz_image`, que incluye optimizaciones específicas para este tipo de mensajes. Este

nodo se configura con parámetros como los nombres de los tópicos a transformar, así como detalles adicionales sobre el tipo de salida, la calidad de la imagen comprimida en formato JPEG, etc.

Por último, dado que el controlador de las ruedas necesita información temporal asociada a la velocidad, se utiliza el paquete `twist_stamper` para añadir esta información a la velocidad publicada. Además, se aprovecha la funcionalidad de remapeo de tópicos para añadir el *namespace* del robot, lo que permite una mejor organización y evita conflictos con otros robots que puedan estar presentes en la misma simulación.

```
    # Rviz config and launching
    rviz_config_file = PathJoinSubstitution(
        [FindPackageShare("msr_robot"), "rviz", "robot.rviz"]
    )

    rviz_node = Node(
        package="rviz2",
        executable="rviz2",
        name="rviz2",
        output="log",
        arguments=["-d", rviz_config_file],
    )

# Bridge
    bridge = Node(
        package='ros_gz_bridge',
        executable='parameter_bridge',
        name='bridge_ros_gz',
        parameters=[
            {
                'config_file': join(
                    pkg_path, 'config', 'rover_bridge.yaml'
                ),
                'use_sim_time': True,
            }
        ],
        output='screen',
    )

    # Image bridge
    gz_image_bridge_node = Node(
        package="ros_gz_image",
        executable="image_bridge",
        arguments=[
            "/front_camera/image",
            "/arm_camera/image"
        ],
        output="screen",
        parameters=[
            {'use_sim_time': True,
             'camera.image.compressed.jpeg_quality': 75},
        ],
    )
```

```
# Twist stamped
twist_stamped = Node(
    package="twist_stamper",
    executable="twist_stamper",
    name="twist_stamper",
    output="screen",
    parameters=[
        {
            "use_sim_time": True,
        }],
    remappings={('cmd_vel_out', '/rover_base_control/cmd_vel'),
                ('cmd_vel_in', '/cmd_vel')},
)
```

Finalizar función principal

Finalmente se crea un objeto de tipo `LaunchDescription` que reúne todas las acciones previamente definidas para su ejecución dentro del lanzador.

```
# Create the launch description
ld = LaunchDescription()
ld.add_action(SetEnvironmentVariable('GZ_SIM_RESOURCE_PATH',
 model_path))
ld.add_action(SetEnvironmentVariable('GZ_SIM_MODEL_PATH',
 model_path))
ld.add_action(robot_description_launcher)
ld.add_action(declare_sim_time)
ld.add_action(bridge)
ld.add_action(gz_image_bridge_node)
ld.add_action(start_gazebo_server_cmd)
ld.add_action(rviz_node)
ld.add_action(gazebo_spawn_robot)
ld.add_action(twist_stamped)
return ld
```

Utilidades

Antes de la especificación del lanzador principal debes declarar las librerías a usar y, si así lo has definido, las funciones adicionales.

Librerías

```
from os.path import join
from os import environ, pathsep
from ament_index_python.packages import get_package_share_directory,
get_package_prefix
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, SetEnvironmentVariable,
IncludeLaunchDescription, OpaqueFunction
from launch.substitutions import PathJoinSubstitution
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare
from moveit_configs_utils import MoveItConfigsBuilder
```

Función para lanzar Gazebo server y client

Se define en una función el lanzamiento del servidor de Gazebo (`gzserver`) y su cliente (`gzclient`) utilizando el launch oficial de Gazebo (`gz_sim.launch.py de ros_gz_sim`).

```
# Function to start the Gazebo server and client
def start_gzserver(context, *args, **kwargs):
    pkg_path = get_package_share_directory('urjc_excavation_world')
    # world_name = 'small_house'
    world_name = LaunchConfiguration('world_name').perform(context)
    world = join(pkg_path, 'worlds', world_name + '.world')
    # world = 'empty.sdf'

    # Launch Gazebo server
    start_gazebo_server_cmd = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            join(get_package_share_directory('ros_gz_sim'), 'launch',
                         'gz_sim.launch.py')),
        launch_arguments={'gz_args': ['-r -s -v 4 ', world]}.items()
    )

    # Launch Gazebo client
    start_gazebo_client_cmd = IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            join(get_package_share_directory('ros_gz_sim'),
                         'launch',
                         'gz_sim.launch.py')
        ),
        launch_arguments={'gz_args': [' -g ']}.items(),
    )

    return [start_gazebo_server_cmd, start_gazebo_client_cmd]
```

Función para configurar las rutas de los modelos

Para que Gazebo encuentre los modelos del robot (.sdf, .urdf), construye las rutas a los modelos (`model_paths`) y añade lo que ya exista en la variable de entorno `GZ_SIM_RESOURCE_PATH` .

```
# Function to get the model paths
def get_model_paths(packages_names):
    model_paths = ""
    for package_name in packages_names:
        if model_paths != "":
            model_paths += pathsep

        package_path = get_package_prefix(package_name)
        model_path = join(package_path, "share")

        model_paths += model_path

    if 'GZ_SIM_RESOURCE_PATH' in environ:
        model_paths += pathsep + environ['GZ_SIM_RESOURCE_PATH']

    return model_paths
```