ROS2 Apuntes

¿Qué es ROS2, cuándo usarlo y por qué?

Ros significa "sistema operativo de robots". Es algo entre un middleware y un marco construido especialmente para aplicaciones de robótica. El objetivo de Ros2 es proporcionar un estándar para el software de robótica que los desarrolladores pueden usar y reutilizar en cualquier robot.

¿Qué tal si pudieras construir sobre una base de software sólida para programar directamente funcionalidades de alto nivel y trabajar en casos de uso en lugar de cuestiones técnicas de bajo nivel que ya existen de todos modos? Este es el por qué detrás de Ros.

Ros2 está aquí para ayudarte a desarrollar aplicaciones robóticas potentes y escalables. Puedes usarlo siempre que desees crear un software para un robot que requiere mucha comunicación entre sus subprogramas o tiene funcionalidades que van más allá de un caso de uso muy simple.

Estos son los dos puntos principales que describen a ROS2:

1. En primer lugar, Ros2 te proporciona una forma de separar tu código en bloques reutilizables, junto con un conjunto de herramientas de comunicación para comunicarte fácilmente entre todos tus subprogramas.
2. El segundo punto principal es que Ros2 te proporciona muchas herramientas y bibliotecas "plug and play" que te ahorrarán una gran cantidad de tiempo y, lo más importante, te evitarán reinventar la rueda.

Se dice que Ros2 es agnóstico al lenguaje, lo que significa que puedes programar algunas partes de tu aplicación en un lenguaje de programación y otra parte en otro lenguaje de programación simplemente porque las herramientas de comunicación no dependen de un lenguaje específico.

Terminal Terminator

Los atajos de teclado para dividir el terminal son "`Ctrl+Shift+O`" para dividir horizontalmente (arriba y abajo) y "`Ctrl+Shift+E`" para dividir verticalmente (izquierda y derecha).

Instalación de ROS2

La instalación de ROS2 no tiene mucho misterio, simplemente hay que seguir las instrucciones que aparecen en la siguiente web.

Instalación de Ignition gazebo

La instalación de ignition gazebo se hace siguiendo los pasos de esta página.
Los tutoriales para empezar a utilizar gazebo también se encuentran disponibles desde la misma web.

Para iniciar gazebo con interfaz gráfica: `gz sim` luego aparece un menú, pero entrando directamente desde él me da error y no entra. Para entrar bien hay que entrar desde consola con: `gz sim nombreSim.sdf`

Creando un paquete Python

Para crear un paquete Python desde la terminal vamos al directorio que queramos (de ahora en adelante siempre **/src**) y escribimos:

```
ros2 pkg create nombrePaquete --build-type ament_python
--dependencies nombreLibrería
```

Con esa línea creamos un paquete con el nombre que queramos, luego le decimos que será un paquete Python y le indicamos las dependencias necesarias (son opcionales, las puedes añadir luego). (rclpy)

Cada paquete de ROS2, ya sea en C++ o en Python, contendrá un archivo llamado "package.xml".

En ese archivo, básicamente encontrarás dos elementos:

1. El primero es una serie de información que incluye el nombre, la versión de tu paquete, una breve descripción y una lista de datos de mantenimiento seguida de la licencia. Esto está vacío por ahora. Si alguna vez deseas publicar tu paquete, compartirlo con la comunidad de código abierto o incluso agregar una licencia comercial, deberás editar estas etiquetas.

2. Luego, tenemos las dependencias que especificamos al crear el paquete. Si necesitas agregar una nueva dependencia, simplemente añadirás otra etiqueta de dependencia debajo de esa. Justo después de las dependencias puedes ver el tipo de construcción en Python para el paquete.

Para compilar usando colcon desde el terminal simplemente escribimos `colcon build` (donde lo hayamos instalado), pero esto compila todo. Para compilar solo un paquete usamos: `colcon build --package-select nombrePaquete`

Creando un paquete C++

Crear un paquete C++ es pŕacticamente igual que si lo hacemos en Python, lo único que cambia es el ament, que utiliza cmake. El nombre y las librerías también depende de las necesidades. (rclcpp)

```
ros2 pkg create nombrePaquete --build-type ament_cmake
--dependencies nombreLibrería
```

La arquitectura del paquete de C++ es bastante diferente a la del paquete de Python, y eso se debe al cambio en el tipo de construcción.

En ese paquete de C++, tenemos un directorio "include" y uno "source". En C++, es común poner los archivos header en el directorio "include" y los archivos CPP en la carpeta "source". Ahora, tienes el archivo `package.xml`, que básicamente es lo mismo que en el paquete de Python: primero, la información y luego la biblioteca de C++.

Si necesitas otras dependencias, simplemente agregarás otra etiqueta `<depend>` debajo de esa. También hay un archivo ‘CMakeLists.txt` y cuando usamos C++,

necesitamos compilar nuestro código, y aquí es donde lo harás. También indicarás dónde deseas instalar el código, etc.

Como puedes ver, también tenemos aquí la dependencia para C++. Si necesitas agregar una nueva dependencia para tu paquete de C++, deberás agregarla aquí en el `package.xml` y también en el `CMakeLists.txt`.

¿Qué es un nodo de ROS2?

Un nodo es una subparte de tu aplicación con un propósito único. Tu aplicación contendrá muchos nodos agrupados en paquetes, que se comunicarán entre sí.

Para entenderlo mejor, consideremos un ejemplo simplificado. Empezamos con un paquete de cámara, que manejará una cámara como una unidad independiente. Creamos nodos dentro del paquete, como un controlador para programar la cámara y obtener fotogramas, y un programa de procesamiento de imágenes.

Cada nodo puede lanzarse por separado. Los nodos se comunican mediante las funcionalidades de comunicación de ROS.

En otro ejemplo, tenemos un paquete de planificación de movimiento con nodos que calculan la planificación de movimiento y corrigen trayectorias según factores externos.

Un tercer paquete, control de hardware, controla el hardware del robot con nodos para controlar motores y publicar el estado del hardware.

Los nodos se comunican entre paquetes, vinculando nodos de procesamiento de imágenes a nodos de corrección de trayectoria, por ejemplo.

Los nodos reducen la complejidad del código al separar la aplicación en paquetes y proporcionan tolerancia a fallos al ejecutarse en procesos independientes. ROS2 es independiente del lenguaje, permitiendo nodos en Python y C++ que se comuniquen sin problemas.

Ten en cuenta que dos nodos no pueden tener el mismo nombre, así que en tus programas y en tu código, tendrás que asegurarte de que todos los nombres de los nodos sean únicos.

Nodos en C++

Para crear nuestro primer nodo, sigue estos pasos:

1. Crea un archivo .cpp en la carpeta "src" del paquete que hemos creado previamente. Por ejemplo, "**my_first_node.cpp**".
2. Incluye la biblioteca de ROS2 para C++ con `**#include "rclcpp/rclcpp.hpp"**`.

VS Code puede generar advertencias sobre la ruta de inclusión. Para solucionarlo, agrega la ruta de ROS al archivo `c_cpp_properties.json`.

```
"includePath": [
    "${workspaceFolder}/**",
    "/opt/ros/humble/include/**"
],
```

Ejemplo del Nodo Mínimo en C++:

```
#include "rclcpp/rclcpp.hpp"

int main(int argc, char **argv) {
    rclcpp::init(argc, argv); // Inicializa las comunicaciones
en ROS2
    auto node = std::make_shared<rclcpp::Node>("cpp_test"); //
Crea el nodo DENTRO del archivo, no es el archivo el nodo
    RCLCPP_INFO(node->get_logger(), "Hello cpp Node"); //
Imprime
    rclcpp::spin(node); // Mantiene el nodo activo y en
ejecución
    rclcpp::shutdown(); // Terminan las comunicaciones

    return 0;
}
```

Para compilar e instalar el nodo, configura el archivo `CMakeLists.txt` del mismo paquete.

```
# Crea el ejecutable e instala el nodo
add_executable(cpp_node src/my_first_node.cpp)
ament_target_dependencies(cpp_node rclcpp)

install(TARGETS
```

```
  cpp_node
  DESTINATION lib/${PROJECT_NAME}
)
```

Explicación:

- La primera línea crea el ejecutable. El primer argumento es el nombre del ejecutable, y el segundo es el archivo donde se encuentra el nodo.
- Se añaden las dependencias al ejecutable.
- Por último, se instala el nodo para poder ejecutarlo directamente con comandos de ROS2 en la consola.

Para ejecutarlo, usa el siguiente comando en la consola (se recomienda usar `source ~/.bashrc` antes para refrescar las configuraciones):

```
source ~/.bashrc
ros2 run nombre_paquete nombre_ejecutable
```

En este caso concreto:

```
ros2 run my_cpp_pkg cpp_node
```

En el primer paquete he dejado una plantilla para usarla y evitar tener que escribir el código repetitivo en cada archivo. Incluye una clase creada.

ROS2 Librerías para diferentes lenguajes

En primer lugar, es fundamental comprender la variedad de bibliotecas de cliente empleadas para interactuar con estas arquitecturas. Hasta ahora, hemos utilizado rclpy para Python y rlccpp para C++. ¿Pero de dónde provienen y por qué comparten el prefijo "rcl"?

"rcl" no es solo un prefijo; es una biblioteca de ROS2, que significa "ros client library". Esta biblioteca, escrita en C puro, aglutina las funcionalidades centrales bajo el nombre de rcl. Así que, en esencia, estas bibliotecas parten de la misma raíz.

Ahora, hablemos del middleware de ROS2 que se apoya en DDS (Servicio de Distribución de Datos). Este middleware gestiona todas las comunicaciones en la aplicación, proporcionando una capa esencial para el intercambio de información.

rcl, como biblioteca base para ROS sirve como puente hacia el middleware de ROS2. No obstante, en la práctica, no utilizamos rcl directamente. En su lugar, se opta por otras bibliotecas de cliente construidas sobre rcl.

Lo fascinante aquí es que, ya sea que programes en C++ o Python, estás trabajando con la misma biblioteca base, rcl. Esta coherencia es clave y simplifica el desarrollo en ambos lenguajes.

Además, aunque C++ y Python son los lenguajes más respaldados, la flexibilidad de la biblioteca permite la extensión a otros lenguajes. Ya existen implementaciones para Node.js (rclnodejs) y Java (rcljava), lo que subraya la versatilidad de estas herramientas.

Monitorear nodos

Vamos a explorar la herramienta de línea de comandos "`ros2`". Cuando presionas la tecla Tab dos veces, se despliegan todos los comandos disponibles, y hay varios. Ya hemos visto "`ros2 run`" y "`ros2 package`" para la creación de paquetes; sin embargo, en este curso, conoceremos otros comandos.

Si escribes "`ros2 run`", este comando espera que le proporciones un paquete y un ejecutable. Esto te permite lanzar cualquier ejecutable desde la carpeta de instalación global de ROS2 o desde el espacio de trabajo de ROS2. Para obtener ayuda sobre el comando, puedes escribir "`ros2 run -h`" y recibirás un mensaje de ayuda, una práctica que puedes aplicar a cualquier otro comando para obtener documentación.

Otra herramienta útil es "`ros2 node`". Al escribir "`ros2 node`" y presionar Tab dos veces, obtienes una segunda acción. Por ejemplo, puedes utilizar "`ros2 node list`" para visualizar todos los nodos en ejecución en tu entorno. Esto facilita la supervisión de la actividad de los nodos.

Además, "`ros2 node info`" te proporciona información detallada sobre un nodo específico. Al ingresar el nombre del nodo, puedes obtener datos sobre sus subscriptores, publicadores, y más. Los publicadores y servidores de servicios que visualizas están presentes para todos los nodos. Por ejemplo, "rosout" actúa como un publicador general, recopilando registros de todas las aplicaciones. También hay publicadores y servidores de servicios destinados a los parámetros, y cada nodo gestiona sus propios parámetros.

En resumen, hemos explorado el comando "`ros2 pkg create`" para la creación de un nuevo paquete. También hemos visto cómo utilizar "`ros2 run`" con el nombre del paquete y el ejecutable asociado. Además, los comandos "`ros2 node list`" y "`ros2 node info`" permiten listar nodos y obtener información detallada sobre ellos, respectivamente.

Y es muy importante notar que no deberías tener dos nodos con el mismo nombre en tu gráfico.

Renombrar nodos en tiempo de ejecución

En tu aplicación de ROS2, es posible que desees iniciar el mismo nodo varias veces, pero con una configuración diferente.

Supongamos que cuentas con un nodo para un sensor de temperatura y necesitas manejar cinco sensores en total. En este caso, tendrás que lanzar el mismo nodo cinco veces, asignando un nombre único a cada sensor para su identificación.

Es crucial tener en cuenta que no puedes lanzar el mismo nodo con el mismo nombre más de una vez. Aunque no recibiras un error al hacerlo, veamos las posibles consecuencias.

Al utilizar el comando "`ros2 node list`" para listar los nodos, surge una advertencia indicando que algunos nodos comparten un nombre idéntico en el gráfico, lo cual podría ocasionar efectos secundarios no deseados. Al revisar la información de un nodo específico mediante "`ros2 node info`", también se emite una advertencia sobre la existencia de dos nodos con el mismo nombre, y solo se muestra información acerca de uno de ellos.

A diferencia de ROS1, en ROS2 ahora es posible iniciar dos nodos con el mismo nombre. Sin embargo, aunque sea técnicamente posible, no se aconseja hacerlo debido a los posibles problemas que podría generar en el gráfico y las comunicaciones.

Entonces, ¿cómo puedes abordar esta situación cuando necesitas iniciar el mismo nodo múltiples veces, como en el caso del sensor de temperatura? La solución es sencilla: simplemente debes cambiar el nombre del nodo al iniciarlo.

Para lograr esto, puedes emplear el comando "`ros2 run`" y agregar argumentos específicos después de "`--ros-args`". Un ejemplo práctico sería utilizar "`--remap`" seguido de "`__node:=miNodo`" para asignar el nombre "`miNodo`" al nodo. Es esencial destacar que este nombre no está fijado de manera permanente y puede modificarse dinámicamente al iniciar el nodo con "`ros2 run`".

Esta característica resulta especialmente útil cuando necesitas duplicar cierto comportamiento y deseas contar con nodos que tengan configuraciones únicas sin tener que realizar modificaciones en el código en cada ocasión.

Ejemplo:

```
ros2 run my_cpp_pkg cpp_node --ros-args --remap __node:=miNodo
ros2 run my_cpp_pkg cpp_node --ros-args -r __node:=miNodo
```

Rqt y rq_graph

Veamos cómo funciona rqt, una herramienta para resolver problemas en tu gráfico y nodos de ROS2.

Para empezar, simplemente escribe "`rqt`" en la consola para abrir esta herramienta de depuración. Aunque podrías usar "`ros2 run`", la buena noticia es que rqt tiene su propio ejecutable, haciéndolo más fácil de usar.

Cuando entras a rqt, te enfrentas a una pantalla vacía, pero no te preocupes, es normal. rqt es como una caja de herramientas, y el elemento que nos interesa aquí es el "gráfico de nodos" (node graph).

El "gráfico activo" muestra el estado actual del gráfico. Como aún no hemos iniciado ningún nodo, está vacío. Vamos a probar iniciando un nodo para ver qué pasa. Al actualizar la pantalla en rqt, verás el nodo recién iniciado. Si lanzamos el mismo nodo con otro nombre y actualizamos la pantalla, ahora vemos dos nodos.

Es clave tener en cuenta que la ventana de rqt en sí misma es un nodo, como se refleja en la lista de nodos. Si quitas la opción de depuración y desactivas "dead sink", podrás observar el tópico "rosout" al cual cada nodo publica. Aunque los nodos operan de manera independiente, todos se comunican mediante este tópico.

Cuando generas registros (logs), estos se envían tanto a "rosout" como a la interfaz de línea de comandos de ROS2 (ROS2 CLI). Además, hay un "daemon" que se inicia al arrancar un nodo, facilitando la interconexión entre nodos, aunque por ahora no es necesario preocuparse por ello.

También puedes arrancar directamente el gráfico con el ejecutable "rqt_graph", una opción más rápida pero equivalente. El uso de rqt graph resulta muy útil para obtener una visión global de lo que sucede en tu gráfico de ROS2 y para identificar fácilmente posibles errores.

Primer contacto con turtle-sim

Vamos a experimentar con un paquete ya existente que proporciona una simulación simplificada de un robot. Este paquete se llama "turtlesim".

Si no tienes el paquete de turtlesim instalado, puedes hacerlo escribiendo simplemente `` `sudo apt install ros-distro-turtlesim` ``, donde "distro" es el nombre de la distribución que estás utilizando (por ejemplo, "humble").

Una vez instalado, puedes abrir una nueva terminal para depurar, o simplemente actualizar la terminal actual usando `` `source ~/.bashrc` ``, ya que el paquete está instalado, pero la terminal aún no lo reconoce.

Luego, puedes ejecutar el comando `` `ros2 run turtlesim turtlesim_node` `` en la terminal para iniciar un nodo que abrirá una ventana con una tortuga en el medio.

Un nodo puede hacer mucho más que simplemente mostrar una salida en la terminal. Puedes realizar visualizaciones, trabajar en red y mucho más. Después, puedes cerrar la comunicación del nodo cuando hayas terminado.

Si ejecutas `` `ros2 node list` ``, verás el nodo llamado "/turtlesim". Ahora, cambiemos el diseño y arranquemos otro nodo para mover la tortuga, para ello escribimos: ‘`ros2 run turtlesim turtle_teleop_key`’. Puedes utilizar las teclas de flecha para mover la tortuga en la pantalla. Este nodo recibe tus inputs, los envía al nodo "/turtlesim" y muestra el resultado en la pantalla.

Si ejecutas `rqt_graph` y actualizas, podrás ver la entrada de los nodos "/turtlesim" y “/teleop_turtle” y cómo están comunicándose entre sí.

Topics

En robótica y automatización, ROS2 (Robot Operating System 2) emerge como una plataforma esencial que facilita la interconexión y comunicación entre los diversos nodos de un sistema robótico. En el corazón de esta comunicación eficiente se encuentran los "topics" (temas) de ROS2, un concepto que se puede entender mejor a través de una analogía con un transmisor y receptor de radio.

Imaginemos un transmisor de radio como un nodo que publica datos en un tema específico, representado por un número o frecuencia, por ejemplo, 98.7. Este número actúa como un identificador, permitiendo que los nodos receptores, como un teléfono o un automóvil, se suscriban a ese tema para recibir la información transmitida.

En esta analogía, el transmisor de radio se convierte en un editor (publisher) que envía datos sobre el tema 98.7, mientras que los dispositivos receptores, como teléfonos o automóviles, actúan como suscriptores (subscribers) que reciben y decodifican la información. Es crucial destacar que tanto los editores como los suscriptores deben utilizar la misma estructura de datos para asegurar una comunicación efectiva.

La flexibilidad de este sistema se revela cuando se consideran múltiples nodos y su capacidad para publicar en y suscribirse a varios temas. Un nodo puede actuar como un editor en un tema específico, como el transmisor de radio que publica tanto señales FM como AM en diferentes temas, y simultáneamente puede ser un suscriptor en otro tema, como un automóvil que recibe datos de ubicación mientras escucha la radio.

La independencia de los nodos es un aspecto fundamental. Cada nodo, ya sea un editor o un suscriptor, opera de manera autónoma, sin conocimiento de los demás participantes en el tema. Esto permite diversas combinaciones, como nodos que solo publican, solo suscriben o realizan ambas funciones.

El uso de temas se destaca en situaciones donde se requiere el envío de flujos de datos unidireccionales. Los temas proporcionan un canal de comunicación efectivo donde los editores publican información y los suscriptores la reciben, sin la necesidad de respuestas desde el suscriptor hasta el editor. Además, tanto los editores como los suscriptores son anónimos, enfocándose únicamente en la interacción a través del tema en cuestión.

En términos prácticos, la documentación de ROS2 define un tema como un bus nombrado a través del cual los nodos intercambian mensajes. Es esencial notar que,

aunque en la analogía se usaron números con puntos como nombres de temas, en la implementación real, un nombre de tema debe cumplir con ciertos requisitos, como comenzar con una letra y permitir letras, números, guiones bajos, guiones y barras inclinadas.

Consideraciones sobre los “topics”

- ➢ El flujo de datos es unidireccional, por lo que los nodos pueden publicar en el tema y algunos nodos pueden suscribirse al tema.
- ➢ No hay respuesta de un suscriptor a un editor.
- ➢ Los datos van en una sola dirección.
- ➢ Los editores y suscriptores son anónimos.
- ➢ Un editor solo sabe que está publicando en un tema y un suscriptor solo sabe que está suscribiéndose a un tema, nada más.
- ➢ Un tema tiene un tipo de mensaje.
- ➢ Todos los editores y suscriptores en el tema deben usar el mismo tipo de mensaje asociado con el tema.

Publishers en C++

Un publicador (publisher) es un nodo que crea y envía mensajes a un tema específico. Los suscriptores que escuchan en ese tema pueden recibir y procesar estos mensajes. Aquí hay algunos detalles sobre los publicadores en ROS2:

1. Creación de un publicador: En C++, un publicador se crea utilizando el método `create_publisher` del nodo. Necesitas especificar el tipo de mensaje que el publicador va a enviar y el nombre del tema al que va a publicar. También puedes especificar el tamaño de la cola de mensajes para el publicador.

```
publisher_ =
this->create_publisher<example_interfaces::msg::String>("topic
_name", 10);
```

2. Publicación de mensajes: Una vez que tienes un publicador, puedes usarlo para publicar mensajes. Primero, creas un mensaje del tipo apropiado, luego lo llenas con datos y finalmente lo publicas con el método `publish` del publicador.

```
auto message = example_interfaces::msg::String();
message.data = "Hello, world!";
publisher_->publish(message);
```

3. Ciclo de vida del publicador: El publicador permanece activo y puede publicar mensajes mientras el nodo que lo creó esté activo. Cuando el nodo se destruye, también se destruye el publicador.

4. Punteros compartidos: En C++, los publicadores se manejan a menudo como punteros compartidos (`std::shared_ptr`). Esto significa que el publicador se destruirá automáticamente cuando ya no haya referencias a él.

5. QoS (Quality of Service): Al crear un publicador, puedes especificar una política de QoS que controla cómo se entregan los mensajes. Esto puede incluir cosas como la confiabilidad de la entrega de mensajes, la duración de los mensajes y la política de historia.

Subscribers en C++

En ROS2, un suscriptor (subscriber) es un nodo que se suscribe a un tema específico para recibir y procesar mensajes enviados a ese tema por los publicadores. Aquí hay algunos detalles sobre los suscriptores en ROS2:

1. Creación de un suscriptor: En C++, un suscriptor se crea utilizando el método `create_subscription` del nodo. Necesitas especificar el tipo de mensaje que el suscriptor va a recibir y el nombre del tema al que se va a suscribir. También debes proporcionar una función de devolución de llamada que se invocará cada vez que se reciba un mensaje. Además, puedes especificar el tamaño de la cola de mensajes para el suscriptor.

```
subscriber_ =
this->create_subscription<example_interfaces::msg::String>("to
pic_name", 10, std::bind(&ClassName::callbackFunction, this,
std::placeholders::_1));
```

2. Función de devolución de llamada: La función de devolución de llamada es una función que se invoca cada vez que se recibe un mensaje. Esta función recibe el mensaje como argumento y puede procesarlo como sea necesario.

```
void callbackFunction(const
example_interfaces::msg::String::SharedPtr msg){
	// Procesar el mensaje
}
```

3. Ciclo de vida del suscriptor: El suscriptor permanece activo y puede recibir mensajes mientras el nodo que lo creó esté activo. Cuando el nodo se destruye, también se destruye el suscriptor.

4. Punteros compartidos: En C++, los suscriptores se manejan a menudo como punteros compartidos (`std::shared_ptr`). Esto significa que el suscriptor se destruirá automáticamente cuando ya no haya referencias a él.

5. QoS (Quality of Service): Al crear un suscriptor, puedes especificar una política de QoS que controla cómo se reciben los mensajes. Esto puede incluir cosas como la confiabilidad de la entrega de mensajes, la duración de los mensajes y la política de historia.

Debug ROS2 Topics usando CLI

Si deseamos obtener más información sobre un tema, escribiremos `'ros2 topic info'` seguido del nombre del tema. Este comando nos proporciona detalles cruciales, como el tipo de datos manejado, así como el número de publicadores y suscriptores asociados. Es útil para comprender la dinámica de intercambio de información en nuestro sistema.

'`ros2 topic echo`' permite "escuchar" los mensajes en tiempo real. Además, con '`ros2 interface show`' se puede visualizar la definición precisa del mensaje, aportando claridad sobre los datos que debemos enviar al tema.

Con '`ros2 topic hz /nombre_topic`', podemos obtener la frecuencia de publicación real del topic.

'`ros2 topic bw /nombre_topic`', nos da información sobre el ancho de banda utilizado por el nodo. Esto puede ser crucial para optimizar el rendimiento, especialmente cuando lidiamos con tasas de publicación elevadas.

Se puede publicar directamente desde la terminal con '`ros2 topic pub`'. Aunque este comando puede no usarse a menudo, es valioso conocerlo para situaciones específicas.