



**Tecnológico  
de Monterrey**

## **Fundamentación y robótica**

### **Challenge 1**

*Fundamentación de robótica (Gpo 101)*

#### **Alumnos:**

**Nombre**

**Matrícula**

Carlos Adrián Delgado Vázquez

Alfredo Diaz López A01737250

Juan Paulo Salgado Arvizu A01737223

# Resumen

En el presente reporte se presenta la implementación de un sistema en ROS compuesto por dos nodos: uno generador de una señal sinusoidal y otro procesador de dicha señal. Utilizando herramientas como `rqt_plot`, se visualizaron ambas señales (original y procesada) en una misma ventana para verificar su correcto funcionamiento. Además, se desarrolló un archivo de lanzamiento (launch file) que automatiza la ejecución de los nodos y la visualización de las señales. Este proyecto permitió aplicar conceptos clave de ROS y explorar el uso de herramientas de visualización y automatización en entornos robóticos.

## Objetivo

### Objetivo General:

El objetivo principal de este reto es aplicar los conceptos vistos en clase sobre ROS (Robot Operating System), específicamente en la creación y manejo de nodos. Para ello, debemos desarrollar dos nodos: uno que genere una señal sinusoidal y otro que procese esa señal. Además, tenemos que visualizar ambas señales usando herramientas de ROS como `rqt_plot` y crear un archivo de lanzamiento (launch file) que ejecute todo de manera automática.

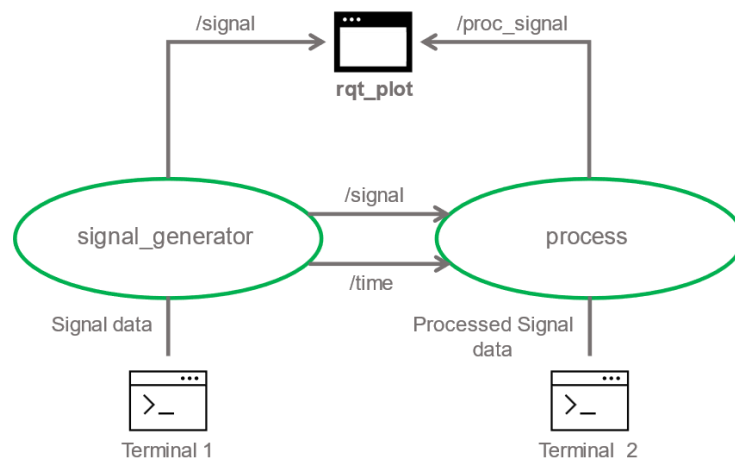


Gráfico 1 Nodos y terminales

## Objetivos Particulares:

1. **Crear un nodo generador de señales:** El primer nodo debe generar una señal sinusoidal, que será la base para el resto del reto.
2. **Crear un nodo procesador de señales:** El segundo nodo tomará la señal generada por el primer nodo y la modificará para producir una señal procesada.
3. **Graficar las señales:** Usar `rqt_plot` para visualizar tanto la señal original como la procesada en una misma ventana, lo que nos ayudará a verificar que todo funciona correctamente.
4. **Depurar y monitorear:** Mostrar la información de las señales en diferentes terminales para poder depurar y asegurarnos de que los datos se están generando y procesando como esperamos.
5. **Crear un launch file:** Hacer un archivo de lanzamiento que ejecute ambos nodos, abra `rqt_plot` y muestre las señales automáticamente, sin necesidad de hacerlo manualmente.

```
rqt_node = Node(name='rqt_plot',  
                package='rqt_plot',  
                executable='rqt_plot',  
                arguments=['/signal/data', '/proc_signal/data']  
                )
```

Gráfico 2 Uso de `rqt_node`

En resumen, este reto nos ayuda a practicar la creación de nodos en ROS, la manipulación de señales y la visualización de datos, además de fomentar nuestra capacidad para investigar y mejorar nuestros propios códigos.

## Introducción

El sistema ROS (Robot Operating System) es un meta sistema operativo de código abierto diseñado para facilitar el desarrollo de software para robots. Proporciona servicios esenciales como la abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comunes, comunicación entre procesos y gestión de paquetes. Además, ofrece herramientas y librerías que permiten obtener, construir, escribir y ejecutar código en múltiples computadoras. Por lo que ROS comparte similitudes con otros frameworks de robótica como Player, YARP, Orocos, CARMEN, Orca, MOOS y Microsoft Robotics Studio, pero se destaca por su flexibilidad y amplia adopción en la comunidad de robótica.

El gráfico de procesos de ROS es una red de procesos peer-to-peer que pueden estar distribuidos en varias máquinas, conectados mediante la infraestructura de comunicación de ROS. Este sistema soporta diversos estilos de comunicación, incluyendo llamadas

RPC sincrónicas a través de servicios, transmisión asincrónica de datos mediante topics y almacenamiento de configuraciones en un servidor de parámetros. Aunque ROS no es un framework de tiempo real, puede integrarse con sistemas que sí lo son, permitiendo la transmisión de mensajes ROS en entornos de tiempo real. Además, ROS cuenta con una integración efectiva con herramientas como Orocos Real-time, ampliando sus capacidades en aplicaciones que requieren respuestas en tiempo real.

La diferencia de ROS respecto a otras plataformas de robótica radica principalmente en soportar la reutilización de código en la investigación y desarrollo dentro de la robótica. Al ser una estructura distribuida de procesos llamados *Nodes* (Nodos), permite el diseño individualizado, pero a su vez es fácilmente adaptable a otros procesos. Una característica muy importante en ROS la agrupación de procesos en *paquetes* y *pilas*, que fácilmente pueden ser intercambiados, compartidos y distribuidos.

Existen 2 versiones de ROS, ROS 1 y ROS 2, pero presentan diferencias importantes. ROS 1 se basa en una arquitectura centralizada que depende de un "master" para coordinar la comunicación entre nodos, lo que puede convertirse en un punto vulnerable, y no está preparado para manejar aplicaciones en tiempo real. Por otro lado, ROS 2 prescinde del "master" y adopta DDS (Data Distribution Service), lo que lo vuelve más resistente, escalable y adecuado para entornos distribuidos y aplicaciones que requieren tiempo real. Además, ROS 2 es compatible con múltiples plataformas (Linux, Windows y macOS), incorpora mejoras en seguridad y utiliza un sistema de compilación más avanzado. En definitiva, ROS 2 representa una evolución de ROS 1, orientada a ser más versátil, segura y apta para aplicaciones industriales y de alta criticidad.

En ROS 2, la red de comunicación está diseñada para ser más robusta, descentralizada y adaptable a entornos distribuidos. Los nodos son los componentes básicos que ejecutan tareas específicas, como controlar sensores o procesar datos, y se comunican entre sí sin depender de un proceso centralizado. La comunicación se realiza principalmente a través de topics, canales asíncronos donde los nodos publican y reciben mensajes (estructuras de datos que definen el formato de la información). Para interacciones síncronas, ROS 2 utiliza servicios, donde un nodo solicita una acción o información a otro y recibe una respuesta. Además, el Parameter Server permite almacenar y acceder a configuraciones y parámetros durante la ejecución. Usa DDS (Data Distribution Service), que gestiona la comunicación de manera descentralizada, mejorando la escalabilidad y fiabilidad. También incluye acciones, que permiten operaciones asíncronas de larga duración con retroalimentación, y bags para almacenar y reproducir datos de topics. Herramientas como RVIZ y rqt facilitan la visualización y monitoreo de la red, haciendo de ROS 2 un sistema moderno y adecuado para aplicaciones en tiempo real y entornos industriales.

## Solución del problema

Se implementó una solución basada en ROS 2, utilizando dos nodos principales: `signal_generator` y `process`. A continuación, se describe la metodología seguida, los

elementos utilizados, las funciones implementadas y el seguimiento del código desarrollado.

### 1. Creación del paquete y nodos:

- Se creó un nuevo paquete llamado **signal\_processing** utilizando las dependencias `std_msgs` y `roscpp`.
- Dentro del paquete, se desarrollaron dos nodos: **signal\_generator** y **process**, cada uno con responsabilidades específicas.

### 2. Nodo **signal\_generator**:

- Este nodo genera una señal sinusoidal en función del tiempo, es decir,  $y=f(t)=\sin(t)$ .
- La señal generada se publica en un tópico llamado **/signal** utilizando un mensaje de tipo `Float32`.
- El tiempo  $t$  se publica en otro tópico llamado **/time**, también con un mensaje `Float32`.
- El nodo opera a una frecuencia de 10 Hz y utiliza la función `get_logger().info` para imprimir los resultados en la terminal.

### 3. Nodo **process**:

- Este nodo se suscribe a los topics **/signal** y **/time** para recibir la señal sinusoidal y el tiempo.
- Procesa la señal recibida aplicando las siguientes transformaciones:
  - *Offset: Se añade un valor constante  $\alpha$  a la señal para garantizar que sea positiva para todo  $t \geq 0$ .*
  - *Reducción de amplitud: La amplitud de la señal se reduce a la mitad.*
  - *Desplazamiento de fase: Se añade un desplazamiento de fase a la señal original. Este valor se puede definir como un parámetro o variable (en este caso, se `hardcodeó`).*
- El resultado se publica en un nuevo topic llamado **/proc\_signal** utilizando un mensaje `Float32`.
- El nodo también opera a una frecuencia de 10 Hz e imprime los resultados en la terminal.

### 4. Implementación matemática:

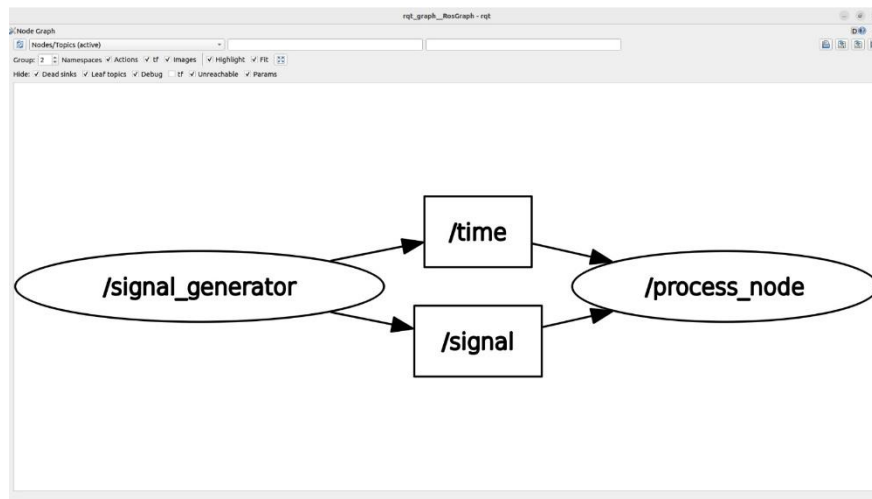
- Para el procesamiento de la señal, se utilizaron funciones trigonométricas y operaciones matemáticas básicas. Se consideraron identidades trigonométricas.
- Se empleó la biblioteca **NumPy** para facilitar el manejo de operaciones matemáticas y funciones trigonométricas.

## 5. Pruebas y ajustes:

- Se verificó el correcto funcionamiento de ambos nodos, asegurando que la señal generada y procesada se publicara en los topics correspondientes.
- Se ajustaron parámetros como la frecuencia de operación y el desplazamiento de fase para validar el comportamiento del sistema.
- Se verificó el correcto funcionamiento de graph y de la visualización de la señal.

# Resultados

Obtenemos el grafo de nodos generado con `rqt_graph`



*Gráfico 3 Estructura del nodo y comunicación*

## Descripción:

**/signal\_generator:** Este nodo actúa como un generador de señales, específicamente una señal sinusoidal, como se describe en el reto.

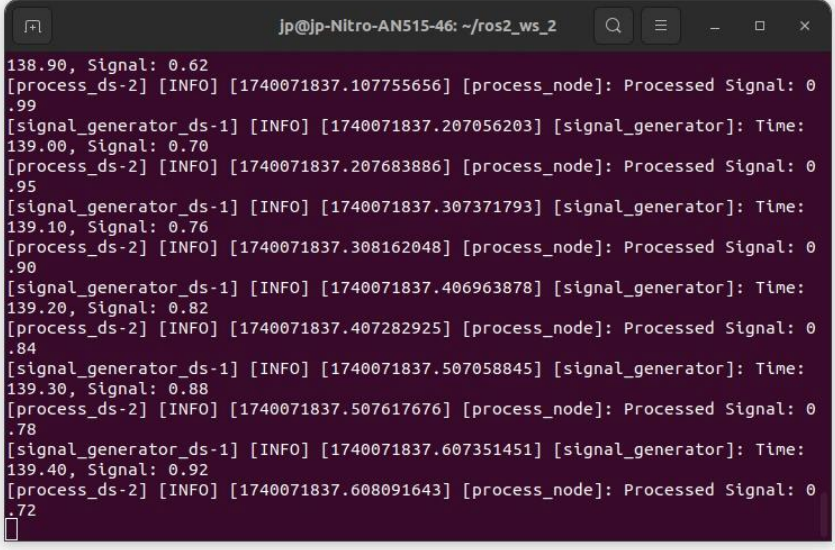
**/time:** Este comando o etiqueta probablemente se refiere a la gestión del tiempo para sincronizar la generación de señales.

**/process\_node:** Este nodo toma la señal generada por `/signal_generator` y la modifica, generando una "señal procesada".

**/signal:** Este comando o etiqueta probablemente se refiere a la señal generada o procesada.

## Análisis:

Este grafo indica una arquitectura de publicación-suscripción en ROS donde un nodo generador de señales está transmitiendo datos que son posteriormente procesados por otro nodo. La estructura parece bien definida y funcional.



```
jp@jp-Nitro-AN515-46: ~/ros2_ws_2
138.90, Signal: 0.62
[process_ds-2] [INFO] [1740071837.107755656] [process_node]: Processed Signal: 0.99
[signal_generator_ds-1] [INFO] [1740071837.207056203] [signal_generator]: Time: 139.00, Signal: 0.70
[process_ds-2] [INFO] [1740071837.207683886] [process_node]: Processed Signal: 0.95
[signal_generator_ds-1] [INFO] [1740071837.307371793] [signal_generator]: Time: 139.10, Signal: 0.76
[process_ds-2] [INFO] [1740071837.308162048] [process_node]: Processed Signal: 0.90
[signal_generator_ds-1] [INFO] [1740071837.406963878] [signal_generator]: Time: 139.20, Signal: 0.82
[process_ds-2] [INFO] [1740071837.407282925] [process_node]: Processed Signal: 0.84
[signal_generator_ds-1] [INFO] [1740071837.507058845] [signal_generator]: Time: 139.30, Signal: 0.88
[process_ds-2] [INFO] [1740071837.507617676] [process_node]: Processed Signal: 0.78
[signal_generator_ds-1] [INFO] [1740071837.607351451] [signal_generator]: Time: 139.40, Signal: 0.92
[process_ds-2] [INFO] [1740071837.608091643] [process_node]: Processed Signal: 0.72
```

*Gráfico 4 Registros en consola*

### Descripción:

**Time:** Representa el tiempo en el que se generó o procesó la señal.

**Signal:** El valor de la señal generada o procesada en ese momento.

**[process\_ds-2]:** Identificador del nodo de procesamiento que está manejando la señal.

**[signal\_generator\_ds-1]:** Identificador del generador de señales.

**[INFO]:** Indica que los mensajes son informativos.

**Processed Signal:** El valor de la señal después de ser procesada por el nodo de procesamiento.

### Análisis:

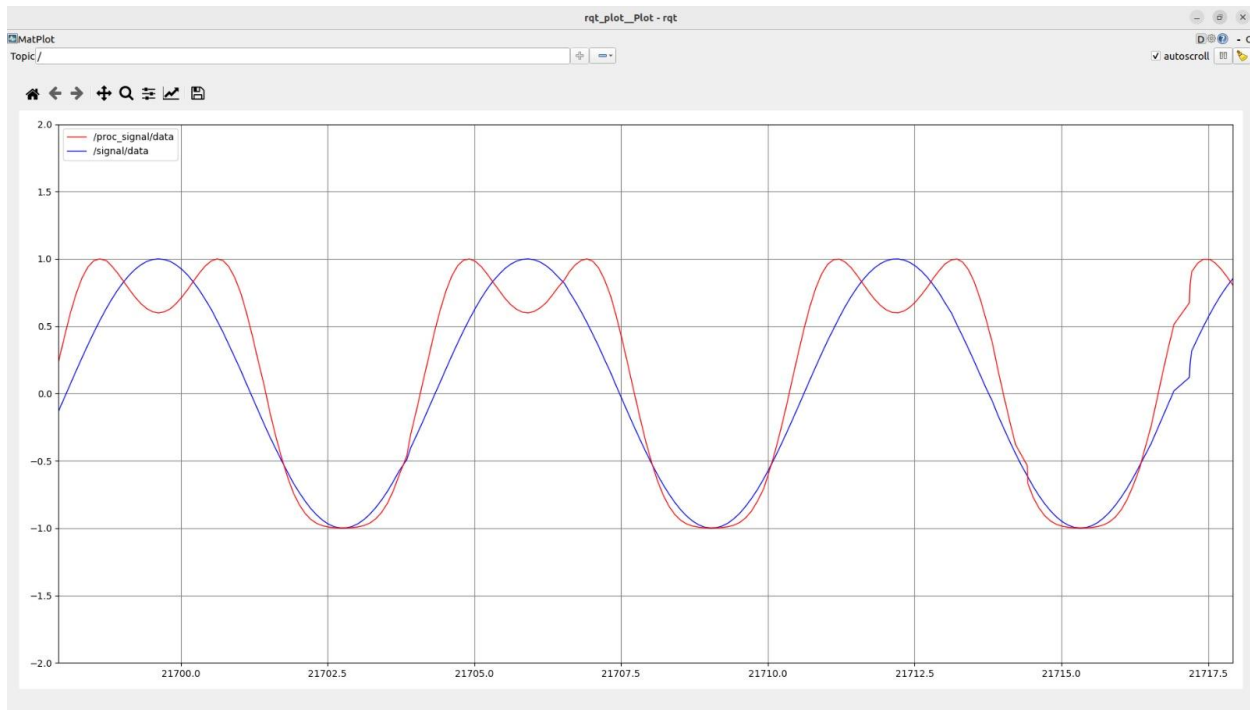
Se observa que:

El generador de señales (signal\_generator\_ds-1) produce señales que aumentan con el tiempo.

El nodo de procesamiento (process\_ds-2) recibe estas señales y las procesa, resultando en valores que disminuyen con el tiempo.

Finalmente, tenemos la visualización de los resultados con la gráfica generada con `rqt_plot`. Donde la línea azul representa la señal generada por el nodo `/signal_generator` y la línea roja muestra la señal después de ser procesada por el nodo `/process_node`.

A partir de la gráfica, se evidencia una modificación en la señal procesada con respecto a la original, lo que indica que el nodo de procesamiento está aplicando una transformación sobre los datos.



*Gráfico 5 Visualización de señales*

## Descripción:

Esta imagen representa una gráfica de señales obtenida con `rqt_plot`, que permite visualizar los datos publicados en diferentes tópicos. Se observan dos señales:

**Línea azul (`/signal/data`):** Representa la señal original generada.

**Línea roja (`/proc_signal/data`):** Representa la señal procesada por el nodo `/process_node`.

## Análisis:

Se observa que la señal procesada (línea roja) mantiene la misma frecuencia que la señal original (línea azul), pero con una leve modificación en la amplitud y forma de onda. Esto sugiere que el procesamiento aplicado introduce un cambio en la señal, posiblemente un filtro, una transformación matemática o una modulación.

La relación entre ambas curvas sugiere que el nodo de procesamiento no introduce una distorsión severa, lo cual podría indicar que el sistema está funcionando correctamente.



# Conclusión

El proyecto demostró el funcionamiento eficiente de ROS como plataforma para la creación y gestión de sistemas distribuidos. A través de la implementación de dos nodos interconectados, se comprobó la capacidad de ROS para facilitar la comunicación y sincronización entre componentes. La integración de herramientas para graficación en ROS, permitió visualizar las señales en tiempo real, validando el flujo de datos entre los nodos. Además, el uso de un archivo de lanzamiento resaltó la flexibilidad de ROS para automatizar procesos complejos, reduciendo la intervención manual. En conjunto, el sistema desarrollado refleja la robustez y escalabilidad de ROS, consolidándolo como una herramienta esencial para aplicaciones en robótica y automatización.