



**Tecnológico
de Monterrey**

**Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Puebla**

Materia: Fundamentación de robótica

Clave: TE3001B

Grupo: 101

Profesores:

Alfredo García Suárez
Rigoberto Cerino Jiménez
Juan Manuel Ahuactzin Larios

Challenge 1

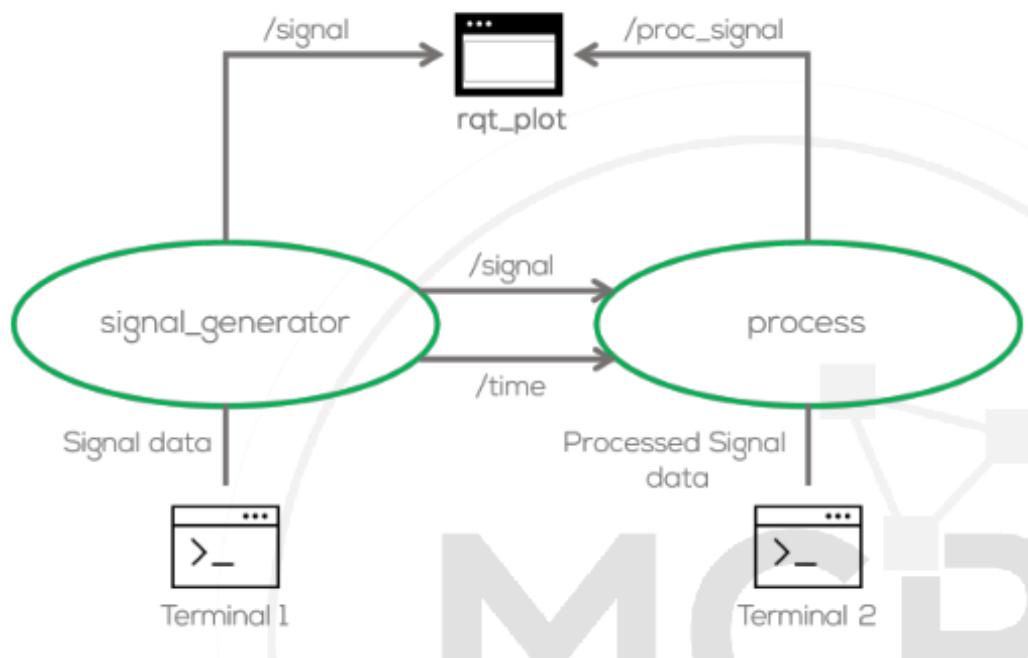
Alumnos

Omar Perez Dominguez | A01738306
Ezequiel Luna Trejo | A01738153
Antonio Méndez Rodríguez | A01738269

17 de Febrero De 2026

Resumen

Este proyecto consistió en el desarrollo de un sistema de procesamiento de señales en tiempo real utilizando ROS 2. Se implementó un paquete llamado ***signal_processing*** compuesto por dos nodos fundamentales : un generador, encargado de publicar una señal senoidal y el tiempo a 2 Hz , y un procesador, que se suscribe a estos datos para transformar la señal. La manipulación principal realizada fue el aumento de amplitud, junto con la aplicación de un *offset* y un desfase para asegurar una salida controlada. El sistema fue validado mediante un archivo Launch que integra la visualización comparativa en ***rqt_plot*** y el monitoreo en terminales de diagnóstico.



Objetivo General

Desarrollar e integrar un sistema de procesamiento de señales distribuido mediante el uso de nodos en ROS 2, garantizando la correcta sincronización y transformación de datos en tiempo real.

Objetivos Específicos

- Implementación de Comunicación Asíncrona: Crear un paquete llamado ***signal_processing*** que gestione la interacción entre nodos productores y consumidores utilizando los tipos de mensajes estándar de ROS (***std_msgs***).
- Generación de Telemetría: Diseñar un nodo generador capaz de publicar una señal sinusoidal y una referencia temporal a una frecuencia constante de 2 Hz a través de tópicos independientes (***/signal*** y ***/time***).
- Procesamiento y Acondicionamiento de Señales: Desarrollar un nodo de procesamiento que suscriba y transforme la señal original aplicando operaciones matemáticas de aumento de amplitud.
- Automatización: Configurar un archivo *Launch* que permita la ejecución simultánea de los nodos, la apertura de terminales de diagnóstico y la inicialización de herramientas gráficas de monitoreo.
- Validación y Análisis Visual: Utilizar la herramienta ***rqt_plot*** para comparar gráficamente la señal cruda frente a la señal procesada, asegurando la integridad del algoritmo de transformación.

Introducción: Procesamiento de Señales en ROS 2

¿Qué es ROS?

El Robot Operating System (ROS), específicamente en su versión ROS 2 Humble, no es un sistema operativo tradicional, sino un middleware o conjunto de herramientas y librerías de software diseñado para ayudar a construir sistemas robóticos. Su objetivo principal es facilitar la creación de comportamientos robóticos complejos mediante la división del hardware y la gestión de la comunicación entre diferentes procesos, esto lo facilita ya que trabaja cada parte del robot de manera individual.

Estructura básica de funcionamiento

El funcionamiento de ROS se basa en un sistema que modula todos los procesos. En este proyecto, la estructura se organiza dentro de un Workspace (espacio de trabajo), el cual contiene paquetes específicos como el creado para este reto: **signal_processing**.

- Paquetes: Son la unidad fundamental de organización de software en ROS.
- Nodos: Son los procesos individuales que realizan cómputo. Un sistema de ROS suele consistir en muchos nodos que trabajan en conjunto.

Principales agentes y Red de comunicación

La comunicación en este sistema se basa en el modelo Publicador/Suscriptor:

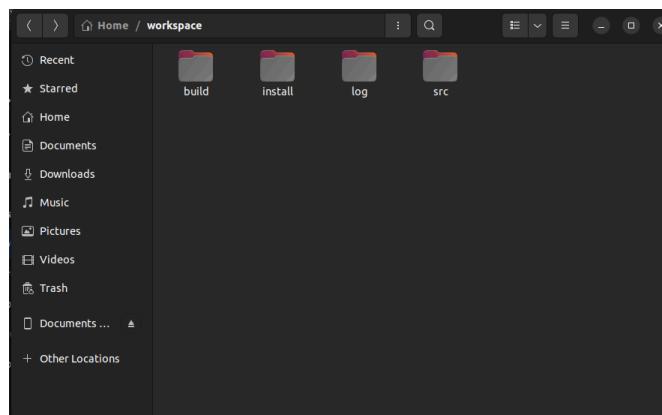
- Tópicos (Topics): Son los canales de comunicación. Los nodos intercambian mensajes a través de ellos.
- Mensajes: Son estructuras de datos (como std_msgs/Float64) que viajan por los tópicos.
- Agentes del Reto: Generador: Actúa como un publicador que envía una señal senoidal pura.
- Procesador: Actúa simultáneamente como suscriptor (recibe la señal original) y publicador (envía la señal modificada con mayor amplitud).
- rqt_plot: Funciona como un suscriptor visual que permite monitorear el comportamiento de las señales en tiempo real.

Solución del problema

Metodología

Para el cumplimiento de los objetivos, se siguió una metodología de desarrollo modular basada en los estándares de ROS 2 Humble sobre Ubuntu 24.04. El proceso se dividió en las siguientes etapas:

- Configuración del Entorno: Se creó un espacio de trabajo denominado ***workspace*** con una estructura de carpetas src para alojar los paquetes de desarrollo.



- Creación del Paquete: Se generó el paquete ***signal_processing*** utilizando la herramienta ***ros2 pkg create***, definiendo dependencias clave como ***rclpy*** y ***std_msgs***.
- Desarrollo de Nodos: Se programaron dos nodos independientes en Python para separar las tareas de generación y procesamiento de datos.
- **Integración y Automatización:** Se configuró un archivo ***Launch*** para permitir la ejecución simultánea de los nodos y las herramientas de visualización.

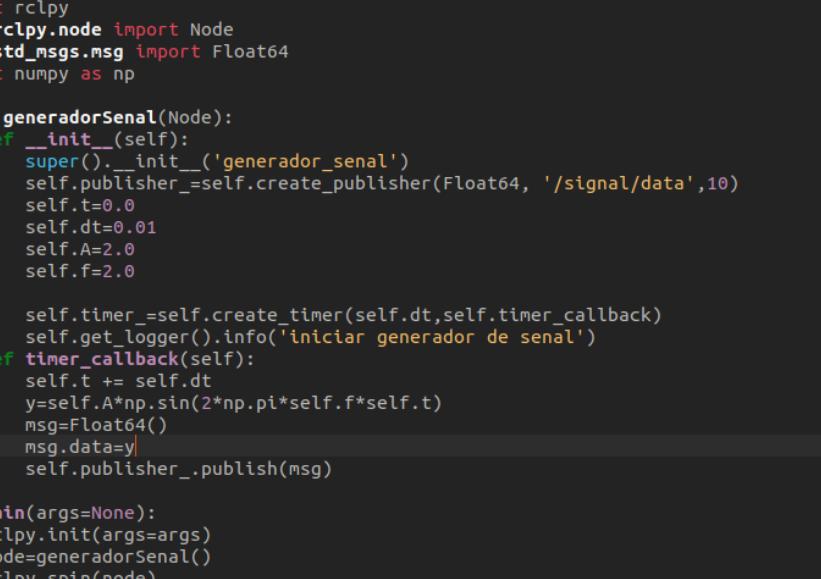
Descripción de los Elementos y Funciones

El sistema se compone de tres elementos principales que interactúan en la red de comunicación:

A. Nodo Generador (*generador_senal.py*)

Su función es crear una señal senoidal pura.

El código implementa una clase que hereda de *Node*. Se inicializa un publicador en el tópico */signal/data* y se configura un temporizador que se dispara cada 0.01 segundos. En cada iteración, se utiliza la función seno de NumPy para calcular el valor actual de la señal basándose en el tiempo transcurrido, encapsulando este resultado en un mensaje de tipo *Float64* para su transmisión.

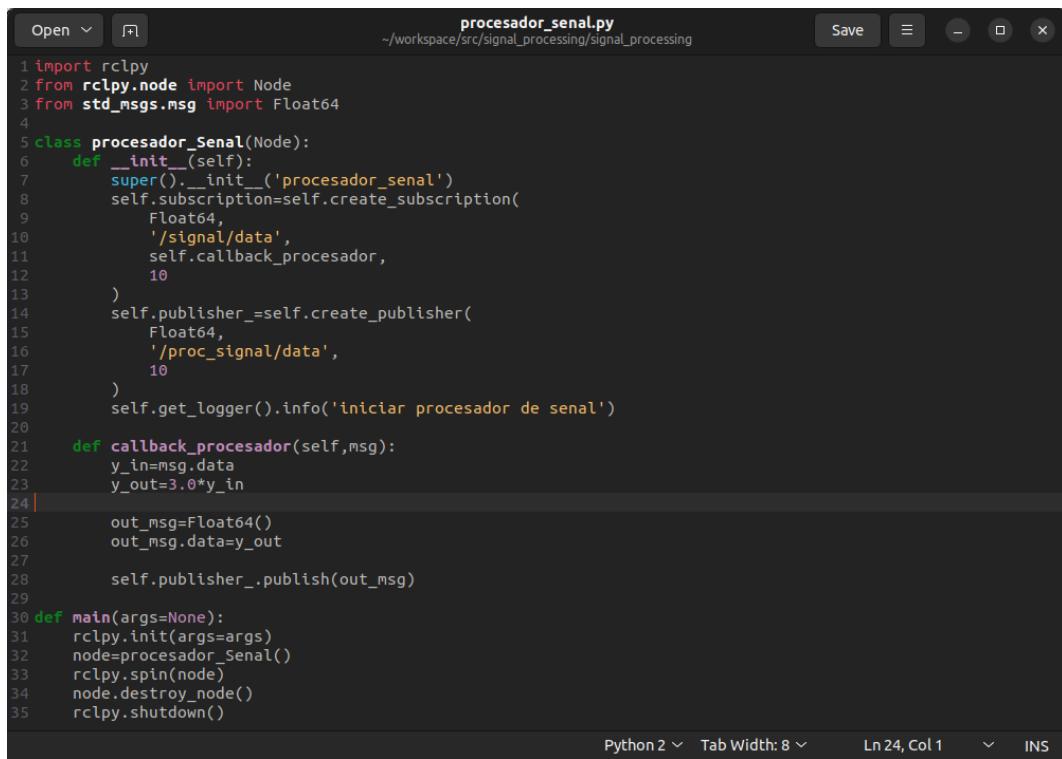


```
Open ▾ + generador_señal.py ~/workspace/src/signal_processing/signal_processing Save ⌂ ⌄ ×
1 import rclpy
2 from rclpy.node import Node
3 from std_msgs.msg import Float64
4 import numpy as np
5
6 class generadorSenal(Node):
7     def __init__(self):
8         super().__init__('generador_señal')
9         self.publisher_=self.create_publisher(Float64, '/signal/data',10)
10        self.t=0.0
11        self.dt=0.01
12        self.A=2.0
13        self.f=2.0
14
15        self.timer_=self.create_timer(self.dt,self.timer_callback)
16        self.get_logger().info('iniciar generador de señales')
17    def timer_callback(self):
18        self.t += self.dt
19        y=self.A*np.sin(2*np.pi*self.f*self.t)
20        msg=Float64()
21        msg.data=y
22        self.publisher_.publish(msg)
23
24 def main(args=None):
25     rclpy.init(args=args)
26     node=generadorSenal()
27     rclpy.spin(node)
28     node.destroy_node()
29     rclpy.shutdown()
```

B. Nodo Procesador (*procesador_señal.py*)

Este nodo actúa como un filtro de procesamiento en tiempo real.

El nodo procesador actúa como un sistema de lazo abierto que recibe la señal del generador mediante una suscripción al tópico **/signal/data**. La lógica principal reside en la función callback, la cual multiplica de forma asíncrona cada valor de entrada por un factor de 3.0. Finalmente, el nodo publica el resultado en el tópico **/proc_signal/data**, permitiendo que herramientas como *rqt_plot* comparan ambas señales y verifiquen el aumento de la amplitud en tiempo real.

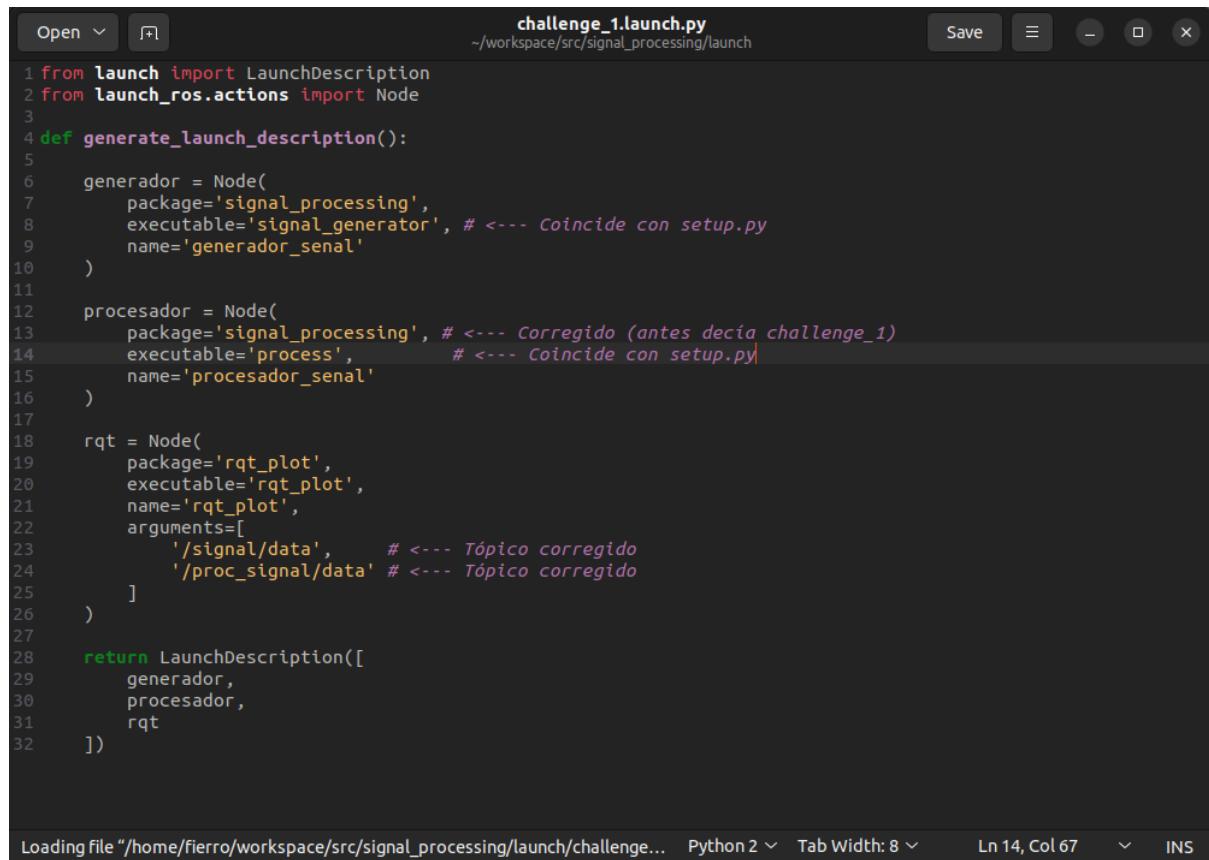


```
1 import rclpy
2 from rclpy.node import Node
3 from std_msgs.msg import Float64
4
5 class procesador_Senal(Node):
6     def __init__(self):
7         super().__init__('procesador_señal')
8         self.subscription=self.create_subscription(
9             Float64,
10             '/signal/data',
11             self.callback_procesador,
12             10
13         )
14         self.publisher_=self.create_publisher(
15             Float64,
16             '/proc_signal/data',
17             10
18         )
19         self.get_logger().info('iniciar procesador de señal')
20
21     def callback_procesador(self,msg):
22         y_in=msg.data
23         y_out=3.0*y_in
24
25         out_msg=Float64()
26         out_msg.data=y_out
27
28         self.publisher_.publish(out_msg)
29
30 def main(args=None):
31     rclpy.init(args=args)
32     node=procesador_Senal()
33     rclpy.spin(node)
34     node.destroy_node()
35     rclpy.shutdown()
```

Python 2 ▾ Tab Width: 8 ▾ Ln 24, Col 1 ▾ INS

C. Archivo Launch (*challenge_1.launch.py*)

Es el director de orquesta del proyecto. Su función es instanciar los nodos anteriores y lanzar automáticamente la interfaz de *rqt_plot* con los tópicos ya configurados para su visualización inmediata.



The screenshot shows a code editor window with the file `challenge_1.launch.py` open. The code defines a launch description for a ROS project. It includes three nodes: `generador`, `procesador`, and `rqt`. The `generador` node is from the `signal_processing` package, `executable` is `signal_generator`, and its `name` is `'generador_señal'`. The `procesador` node is also from the `signal_processing` package, `executable` is `process`, and its `name` is `'procesador_señal'`. The `rqt` node is from the `rqt_plot` package, `executable` is `rqt_plot`, and its `name` is `'rqt_plot'`. It has arguments specifying topics: `/signal/data` and `/proc_signal/data`. The code returns a `LaunchDescription` object containing these three nodes.

```
1 from launch import LaunchDescription
2 from launch_ros.actions import Node
3
4 def generate_launch_description():
5
6     generador = Node(
7         package='signal_processing',
8         executable='signal_generator', # <--- Coincide con setup.py
9         name='generador_señal'
10    )
11
12     procesador = Node(
13         package='signal_processing', # <--- Corregido (antes decía challenge_1)
14         executable='process', # <--- Coincide con setup.py
15         name='procesador_señal'
16    )
17
18     rqt = Node(
19         package='rqt_plot',
20         executable='rqt_plot',
21         name='rqt_plot',
22         arguments=[
23             '/signal/data', # <--- Tópico corregido
24             '/proc_signal/data' # <--- Tópico corregido
25         ]
26    )
27
28     return LaunchDescription([
29         generador,
30         procesador,
31         rqt
32     ])
```

Loading file "/home/fierro/workspace/src/signal_processing/launch/challenge... Python 2 Tab Width: 8 Ln 14, Col 67 INS

Resultados

En esta sección se presentan las evidencias obtenidas tras la ejecución del sistema de procesamiento de señales en el entorno ROS 2 Humble.

1. Evidencia Gráfica (*rqt_plot*)

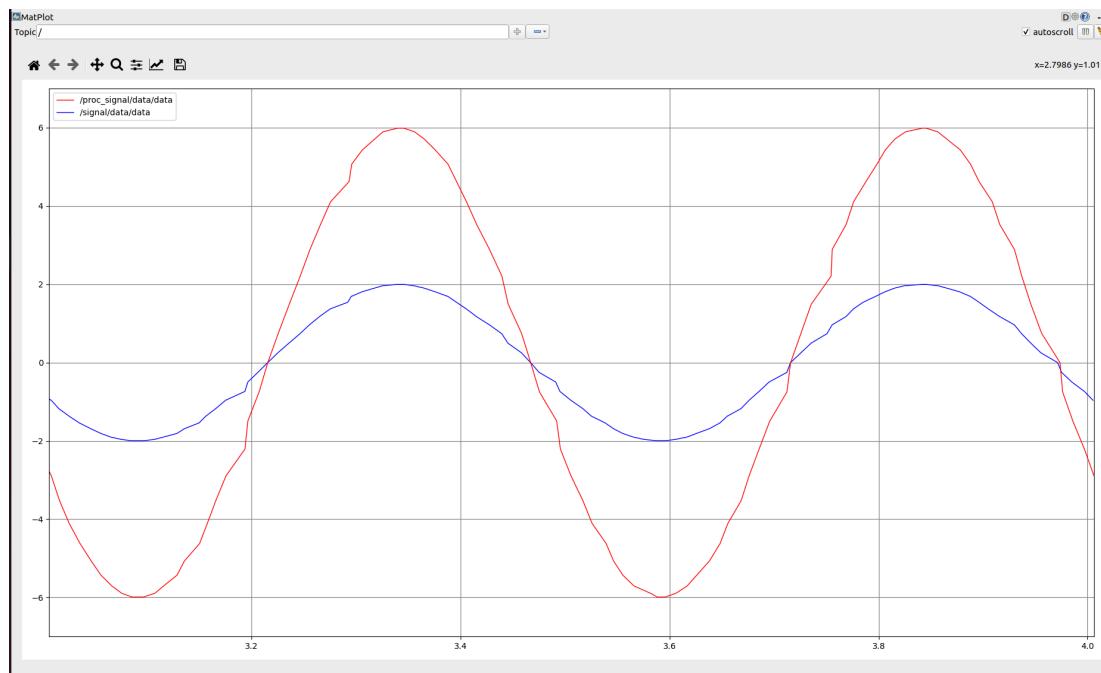
Se utilizó la herramienta *rqt_plot* para visualizar en tiempo real el comportamiento de los nodos. En la gráfica obtenida, se pueden observar dos señales senoidales:

- Señal Original (Azul): Representa los datos provenientes del nodo *generador_senal*, con una amplitud de 2.0 y una frecuencia de 2.0 Hz.
- Señal Procesada (Rojal): Representa los datos del nodo *procesador_senal*, los cuales muestran la misma frecuencia pero con una amplitud de 6.0.

2. Análisis de Resultados

El análisis de la gráfica permite confirmar el correcto funcionamiento de la red de comunicación:

- Sincronización: Ambas señales mantienen la misma fase y frecuencia, lo que demuestra que el procesamiento de datos se realiza en tiempo real sin retrasos significativos (latencia) apreciables en la gráfica.
- Verificación Matemática: Se confirma que la ganancia aplicada de 3.0 en el nodo procesador es exacta, ya que el pico de la señal original en 2.0 se transforma en un pico de 6.0 en la señal de salida.



Conclusiones

Los objetivos planteados para este reto se lograron satisfactoriamente debido a la correcta implementación de una arquitectura modular que permitió la comunicación asíncrona y eficiente entre los nodos de generación y procesamiento de señales. El cumplimiento total de la actividad se evidencia en la capacidad del sistema para transformar datos en tiempo real sin pérdida de sincronía, validando así la robustez del middleware ROS 2 Humble ejecutado sobre Ubuntu 24.04. Como propuesta de mejora a la metodología implementada, se sugiere la integración de Parámetros dinámicos de ROS 2, lo cual permitiría ajustar la frecuencia o la ganancia de la señal desde la terminal sin necesidad de modificar el código fuente y recompilar el paquete, optimizando así la flexibilidad del sistema para futuras aplicaciones de control robótico.

Referencias Bibliográficas (APA)

- Fundación Open Source Robotics. (s.f.). *ROS 2 Documentation: Humble Hawksbill*. Recuperado de <https://docs.ros.org/en/humble/>
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). *Array programming with NumPy*. *Nature*, 585(7825), 357-362.
- Python Software Foundation. (2026). *Python Language Reference, version 3.12*. Recuperado de <https://www.python.org>
- Quigley, M., Gerkey, B., & Smart, W. D. (2015). *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media.
- Ubuntu. (2024). *Ubuntu 24.04 LTS (Noble Numbat)*. Canonical Ltd. Recuperado de <https://releases.ubuntu.com/24.04/>