Reto semanal

Semana 2

Manchester Robotics

Integrantes:

A017314050 | Eleazar Olivas Gaspar

A01735696 | Azul Nahomi Machorro Arreola

A01732584 | Angel Estrada Centeno

A01735692 | Arick Morelos del Campo

Profesores:

Rigoberto Cerino Jimenez

Alfredo Garcia Suarez

Juan Manuel Ahuactzin Larios

Febrero 2024

Resumen:

La actividad consiste en potenciar el reto anterior, el primer nodo actuará como generador de señales de alta frecuencia que generarán señales senoidales, cuadradas o de diente de sierra, el segundo nodo actuará como nodo reconstructor de señales. Se debe generar un archivo launch para iniciar ambos nodos y rqt_plot al mismo tiempo.

Objetivos:

Desarrollar las capacidades de manejo del framework ROS,e incrementar el grado de interacción y la implementación de funciones disponibles en ROS. Este reto nos permitirá comprender el funcionamiento de un archivo tipo "launch" para ejecutar todos los nodos desde un mismo lugar, así como la interacción con parámetros, permitiendo al usuario elegir distintas acciones a ejectuar a través de la consola.

Introducción:

ROS es una plataforma de desarrollo de software versátil y potente que ha ganado popularidad en la comunidad robótica debido a su arquitectura modular, su robusto sistema de comunicación entre nodos y su amplia gama de herramientas y bibliotecas de código abierto. Para familiarizarnos más a fondo con ROS explicaremos algunos conceptos importantes que serán utilizados para realizar este mini reto.

Namespaces:

Los Namespaces son una característica importante en ROS que proporciona un método para organizar y estructurar los nodos, tópicos, servicios y otros recursos dentro de un entorno robótico. Permiten evitar conflictos de nombres y facilitan la modularidad y la reutilización del código, entre las características de esta herramienta tenemos que permiten agrupar los nodos relacionados y los recursos asociados bajo un nombre común, también ayudan a evitar colisiones de nombres entre diferentes componentes de un sistema robótico, pueden ser anidados para crear una jerarquía de nombres que refleje la estructura del sistema robótico y proporcionan flexibilidad en la gestión y configuración de los componentes del sistema.

Parameters:

Los parámetros son valores configurables que se utilizan para ajustar el comportamiento y la configuración de los nodos y componentes en un sistema ROS. Estos valores pueden ser modificados durante la ejecución del sistema sin necesidad de recompilar el código, lo que facilita la personalización y la adaptabilidad del sistema robótico, entre las características de esta herramienta tenemos que los parámetros pueden ser ajustados dinámicamente durante la ejecución del sistema, lo que permite adaptar el comportamiento del robot según sea necesario, también pueden ser accedidos y modificados desde diferentes nodos dentro del sistema, lo que facilita la comunicación y la coordinación entre los componentes, asi como tambien pueden ser guardados y cargados desde archivos de configuración, lo que permite mantener la configuración del sistema entre sesiones de ejecución y por ultimo pueden ser validados para asegurar que cumplan con ciertos criterios antes de ser utilizados, lo que ayuda a garantizar la integridad y la consistencia de los datos.

Custom Messages:

Los Custom Messages son definiciones de estructuras de datos utilizadas para la comunicación entre los nodos en un sistema ROS. Permiten definir tipos de datos específicos para representar información relevante para una aplicación robótica particular, que puede no estar cubierta por los mensajes estándar proporcionados por ROS, entre las características de esta herramienta tenemos que permiten definir estructuras de datos adaptadas a las necesidades específicas de una aplicación robótica, lo que facilita la representación de información compleja y especializada, pueden ser compartidos y reutilizados en diferentes proyectos y sistemas robóticos, lo que promueve la interoperabilidad y la colaboración, al definir mensajes personalizados específicos, se mejora la claridad y la legibilidad del código, ya que los mensajes reflejan directamente los datos relevantes para la aplicación, también los mensajes personalizados pueden ser utilizados de manera similar a los mensajes estándar de ROS, lo que facilita su integración en los sistemas existentes y la comunicación entre los nodos.

Solución del problema:

- **Parámetros desde el archivo launch**

```
signal_generator_node = Node(
    package='courseworks',
    executable='signal_generator',
    output='screen',
    parameters=[{'type': 1,
                 'frequency': 2.0,
                 'offset': 0.0,
                 'amplitude': 0.5,
                 }]#mandar parametros
)

process_node = Node(
    package='courseworks',
    executable='process',
    output='screen',
    parameters=[{'phase_shift': 0.0,
                 'offset': 0.0,
                 'amplitude': -3.0,
                 }]#mandar parametros
)
```

- **Generador de señal**

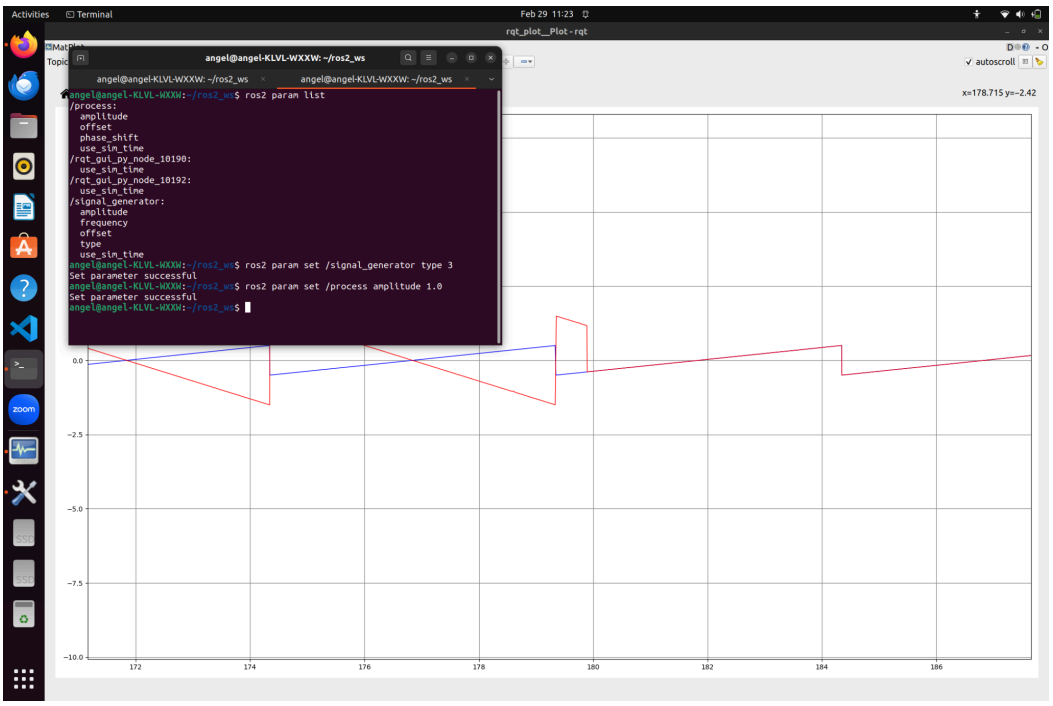
```
def __init__(self):
    self.i = 0
    super().__init__('signal_generator')
    self.publisher = self.create_publisher(Float32, 'signal', 10)
    self.publisher2 = self.create_publisher(Float32, 'signal_params', 10)

    self.declare_parameter('type', 1) # default signal
    self.declare_parameter('frequency', 2.0)  # default frequency
    self.declare_parameter('offset', 0.0)  # default offset
    self.declare_parameter('amplitude', 0.5)  # default amplitude

    timer_period = 0.01
    self.timer = self.create_timer(timer_period, self.timer_callback2)
    self.get_logger().info('signal node successfully initialized!!!')
    self.msg = Float32()


def timer_callback2(self):

    waveform = self.get_parameter('type').get_parameter_value().integer_value
    frequency = self.get_parameter('frequency').get_parameter_value().double_value
    frequency2 = 10
    offset = self.get_parameter('offset').get_parameter_value().double_value
    amplitude = self.get_parameter('amplitude').get_parameter_value().double_value

    if waveform == 1:
        self.msg.data = amplitude * np.sin(2 * np.pi * frequency * self.i + offset)
    elif waveform == 2:
        self.msg.data = amplitude * np.cos(2 * np.pi * frequency * self.i + offset)
    elif waveform == 3:
        self.msg.data = float(amplitude * signal.sawtooth(2 * np.pi * frequency * self.i + offset))
    elif waveform == 4:
        self.msg.data = float(amplitude * signal.square(2 * np.pi * frequency * self.i + offset))

    self.publisher.publish(self.msg)
    self.publisher2.publish(self.msg)
    self.i = self.i + 0.001
```

- **Procesador de señal**

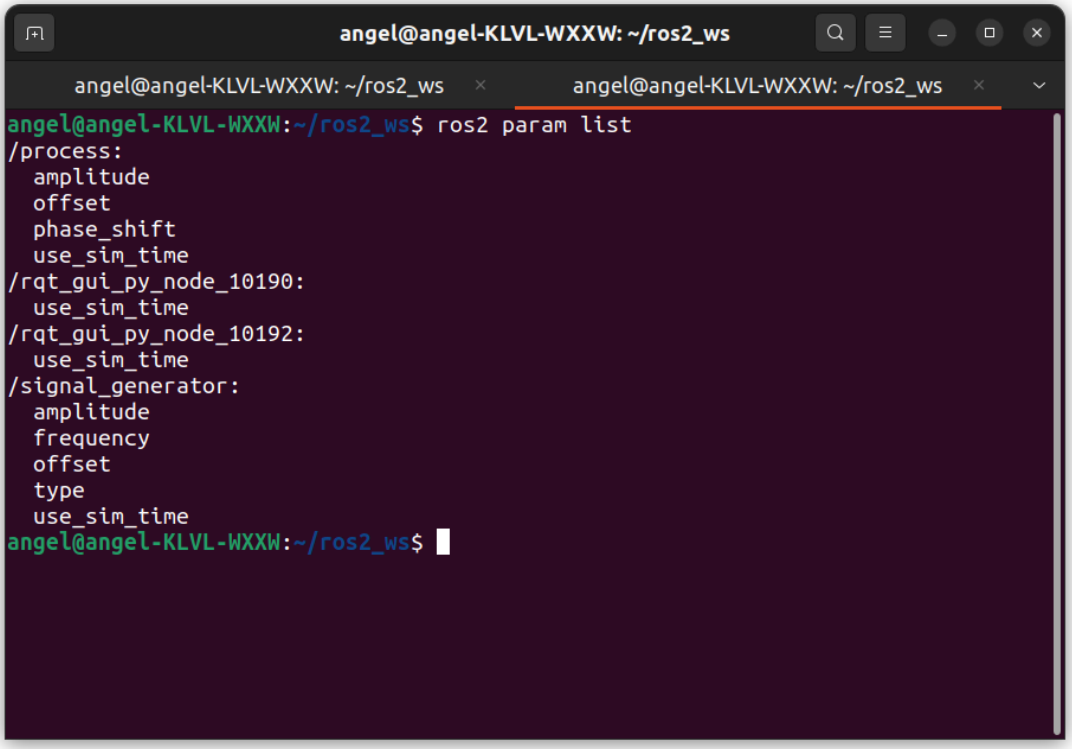
```python
def __init__(self):
    super().__init__('process')
    self.sub = self.create_subscription(Float32, 'signal_params', self.listener_callback, 10)
    self.get_logger().info('process node initialized!!!')
    self.publisher = self.create_publisher(Float32, 'signal_reconstructed', 10)

    self.declare_parameter('phase_shift', 0.0)  # default PS
    self.declare_parameter('offset', 0.0)  # default offset
    self.declare_parameter('amplitude', -3.0)  # default amplitude

def listener_callback(self, msg):

    offset = self.get_parameter('offset').get_parameter_value().double_value
    amplitude = self.get_parameter('amplitude').get_parameter_value().double_value
    phase_shift = self.get_parameter('phase_shift').get_parameter_value().double_value

    self.msg2 = Float32()
    self.msg1 = msg.data
    if not self.msg1:
        self.msg2.data = 0.0
    else:
        self.msg2.data = (self.msg1* amplitude ) + offset
    self.publisher.publish(self.msg2)
```

Resultados:

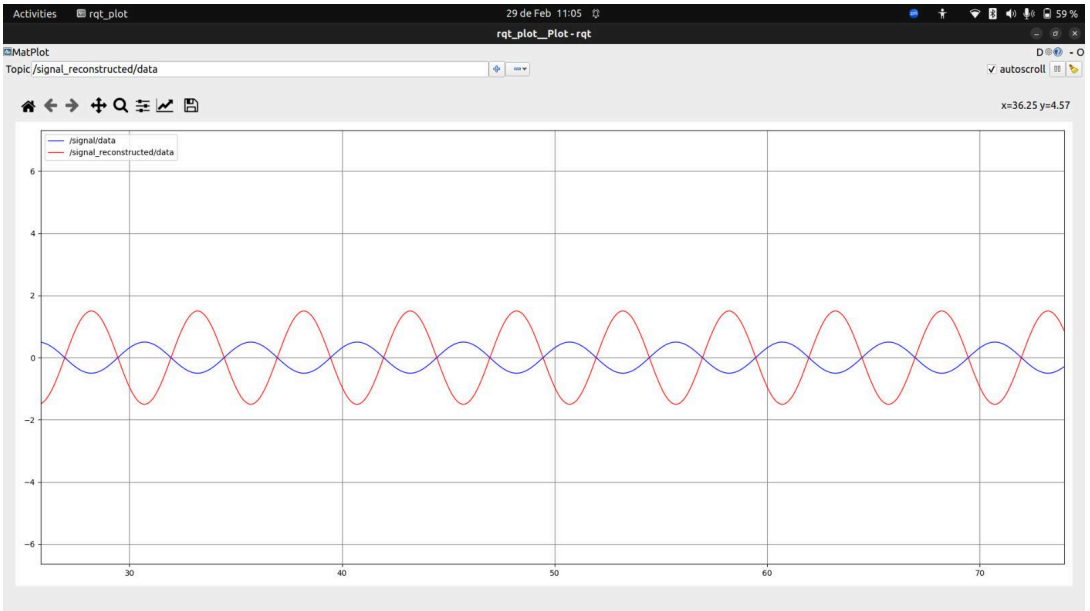
La amplitud de la señal reconstruida ha sido modificada por un valor de -3 con respecto a la señal original para poder ser apreciada y diferenciada de manera visual en las imágenes. Todos los parámetros de ambas señales han sido modificados por medio de la terminal. En la siguiente imagen se muestra la señal original y la señal reconstruida bajo los mismo parámetros:



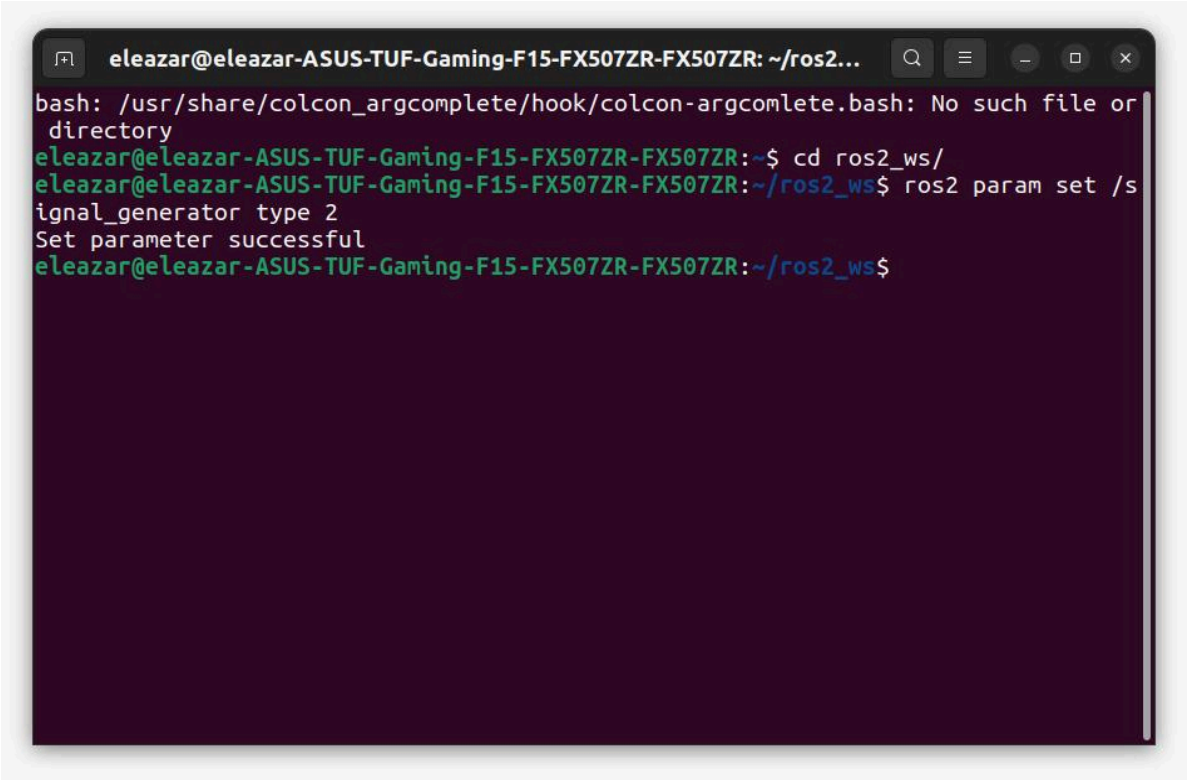
- **Parámetros**

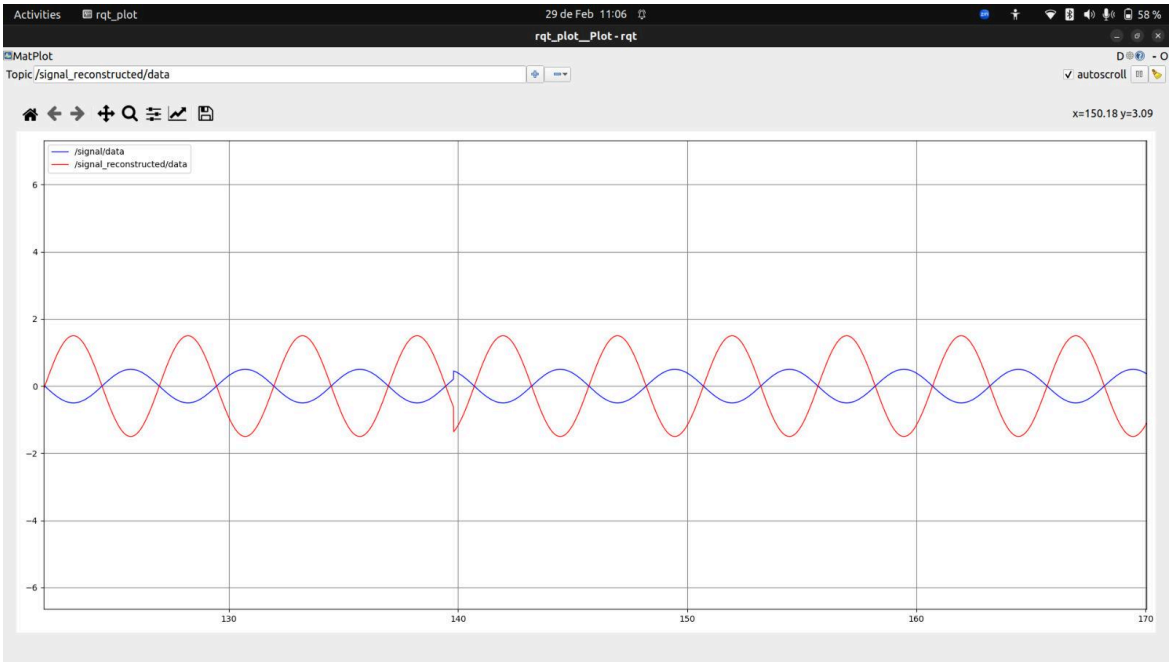


- **Señal senoidal**

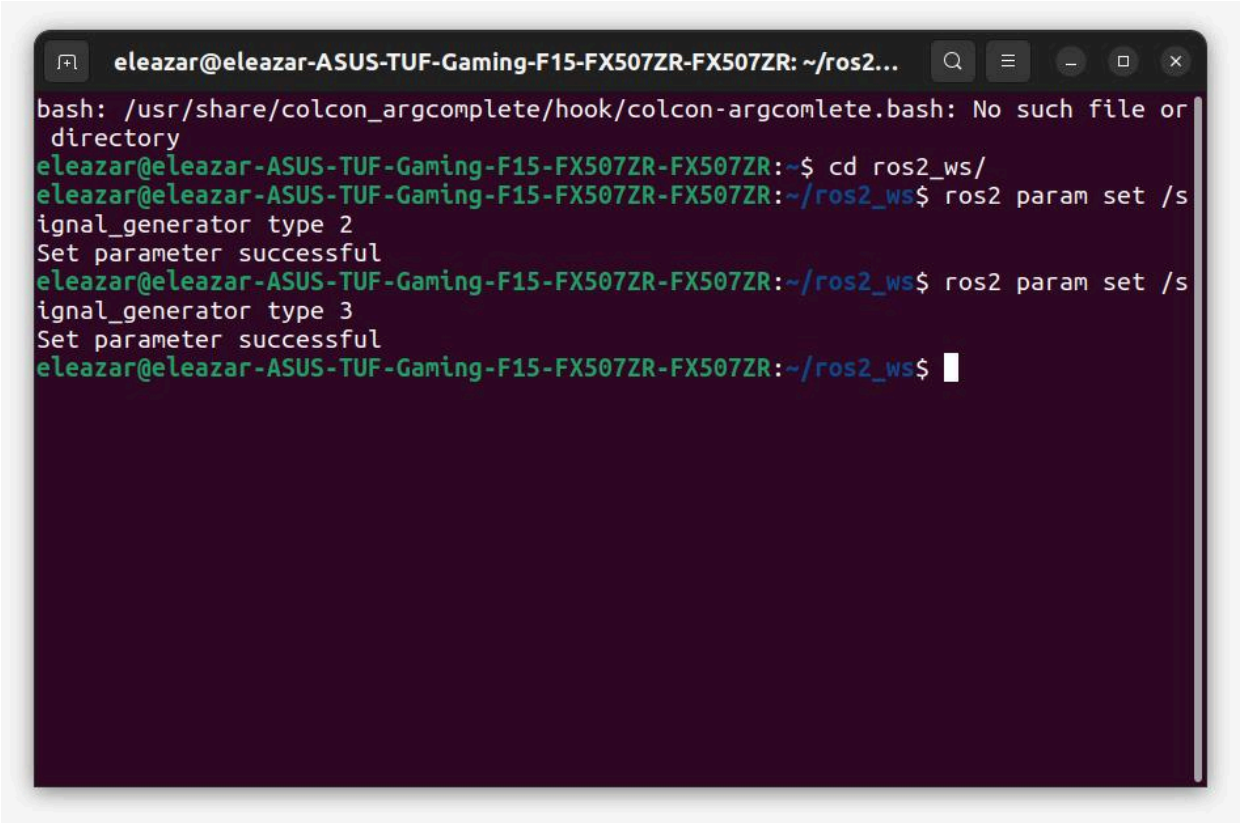


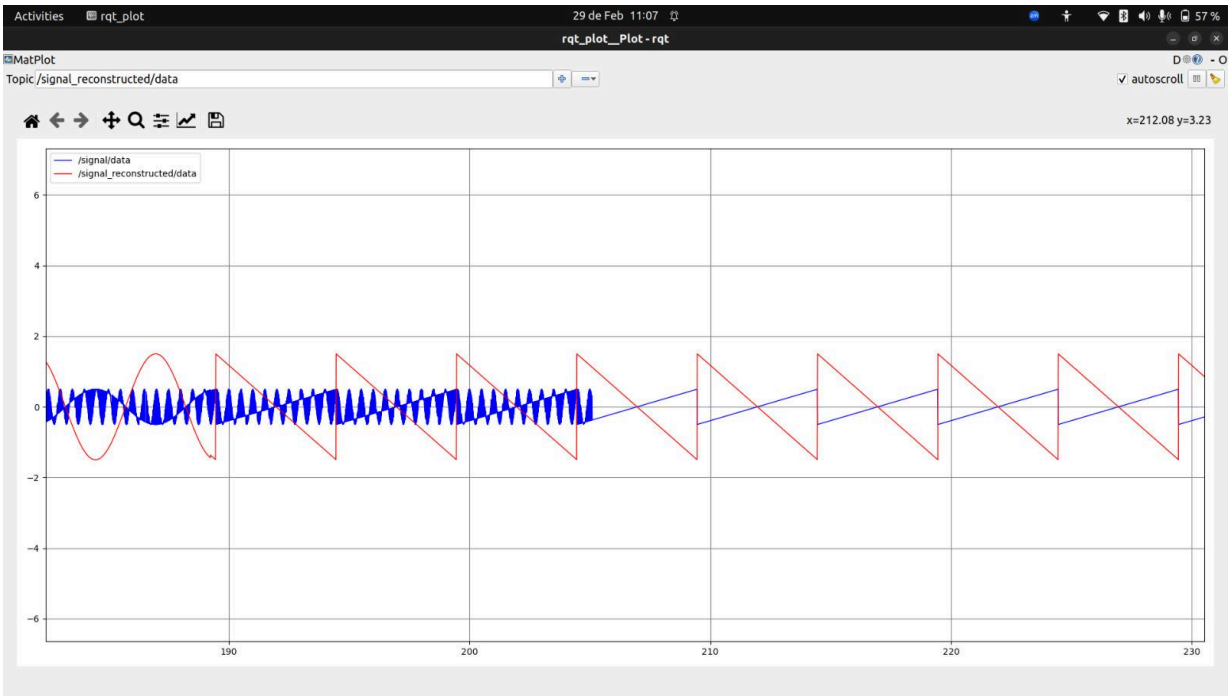
- **Senal cosenoidal**



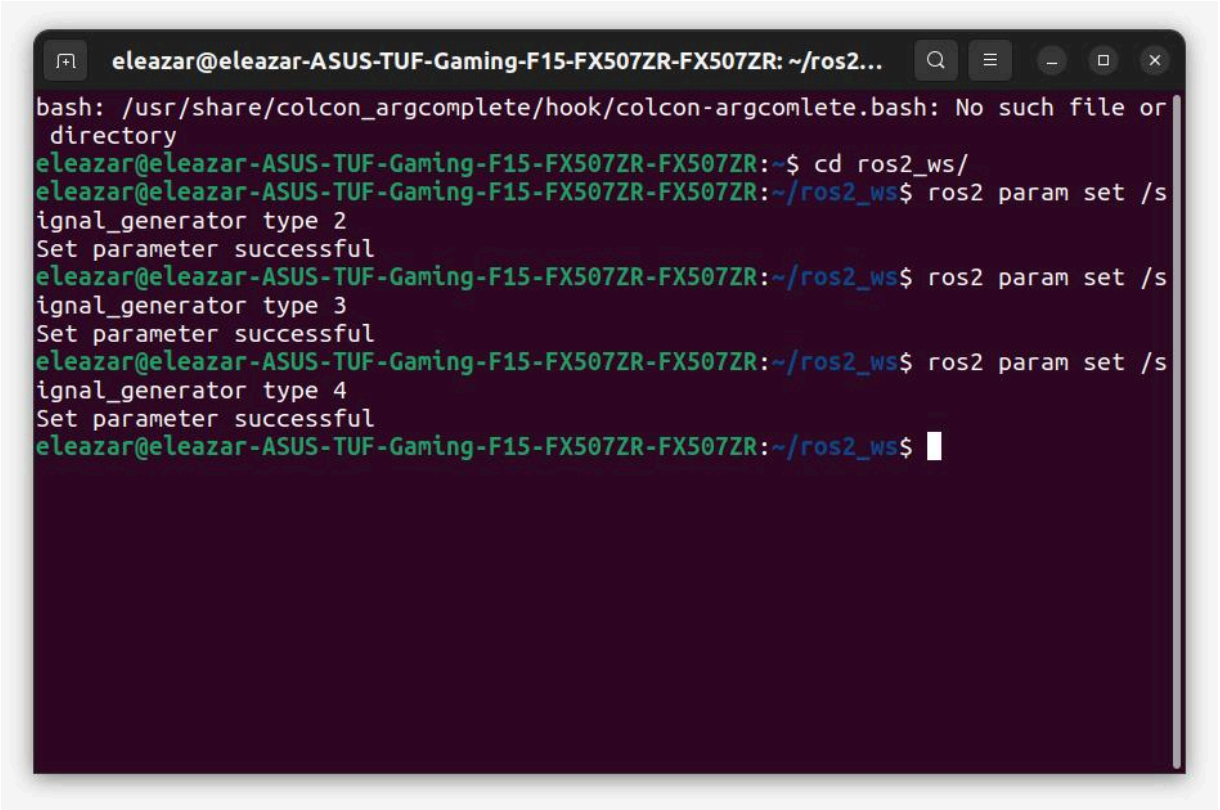


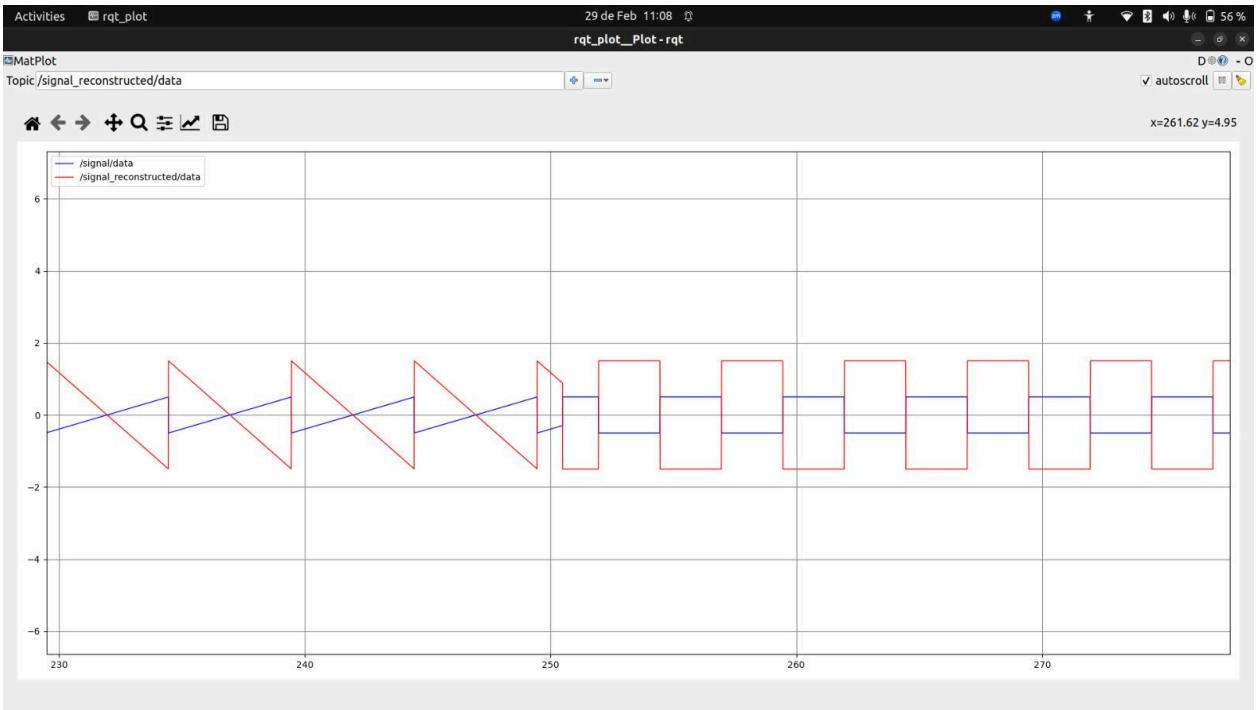
- **Señal sawtooth**



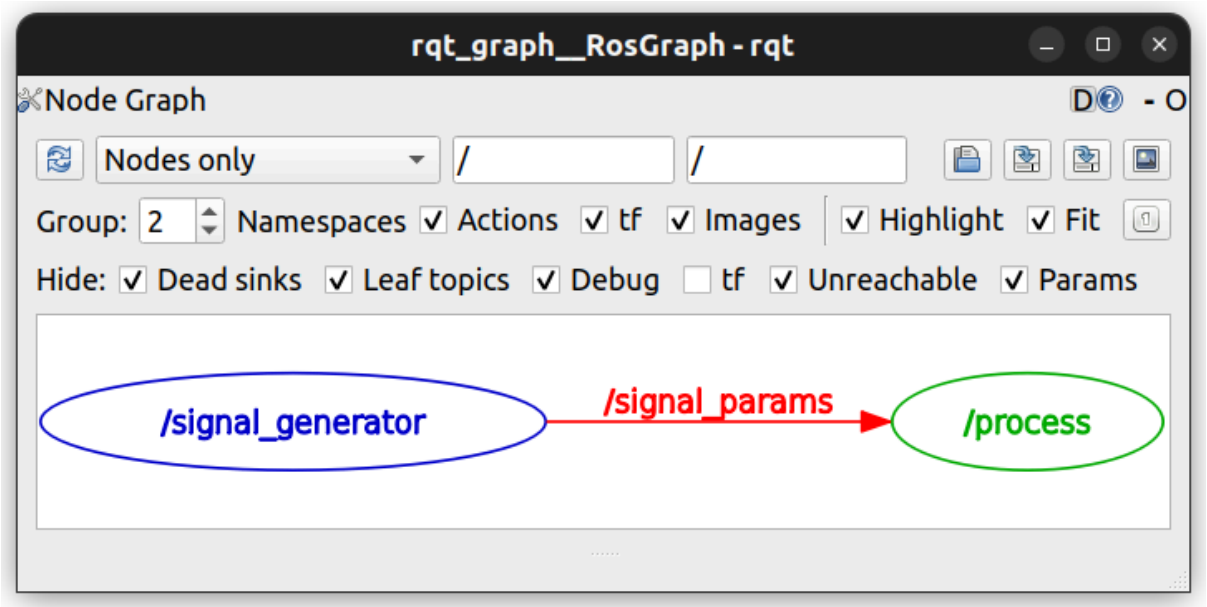


- **Señal pulsos**





- **rqt_graph**



Conclusión:

Gracias a este segundo mini reto logramos adentrarnos mucho más con las herramientas de ROS 2, trabajamos principalmente con los namespaces y los parameters, al momento del desarrollo de este trabajo aprendimos más sobre la comunicación entre nodos a partir del funcionamiento de los custom messages para una mejor representación de información, los parameters fueron de las cosas más importantes en este reto ya que con estos ajustamos el comportamiento y la configuración de las señales creadas en el nodo y estos son de gran ayuda ya que con estos podemos hacer cambios durante la ejecución del sistema sin tener que recompilar el código, simplemente con una instrucción en la terminal hacemos la modificación de las señales que se están proyectando.

Bibliografía:

ROS - Robot Operating System. (s. f.). https://www.ros.org/

ROS Documentation. (s.f.). https://docs.ros.org/