RETO SEMANAL II
Manchester Robotics

Abraham Ortiz Castro
Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
A01736196@tec.mx

Alan Iván Flores Juárez
Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
a01736001@tec.mx

Jesús Alejandro Gómez Bautista
Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
A01736171@tec.mx

Ulises Hernández Hernández
Dpto. de Ingenierías Tec de Monterrey
Tecnológico de Monterrey
Puebla, México
A01735823@tec.mx

I. Resumen

ROS 2 puede resultar un tanto compleja vista como un todo, sin embargo, al profundizar un poco en algunas de sus características más significativas en el campo de la transmisión y definición de nodos, veremos elementos que le otorgan versatilidad y que en un todo, permiten expandir aún más el nivel de posibilidades que el entorno ofrece al usuario, dejando en muchos casos la posibilidad de adaptarlo a sus necesidades. En este ejercicio hemos notado lo importante y útiles que son los parámetros, debido a que se vuelve muy sencilla la manipulación de las respuestas, de la misma manera los config file llegaron a ser útiles para tener las definiciones iniciales estabelicidas y no gastar tiempo innecesario. Por último, los mensajes personalizados son de gran ayuda para la comunicación entre nodos de manera más sencilla, debido a que es más fácil adaptarlos a tus necesidades.

II. Objetivo.

- Crear un config file para utilizarlo con launch.
- Usar parámetros para modificar señales en tiempo real desde la terminal.
- Crear mensajes personalizados para el tráfico sencillo de datos.

III. Introducción.

El reto pasado sirvió como una actividad introductoria y vinculatoria para comprender el funcionamiento de ROS 2, no por nada, el inicio de la misma así como los propósitos de la misma estuvo dedicada completamente al entendimiento de ROS, en esta ocasión profundizaremos un poco más en esto, particularmente, nos centraremos en elementos propios del entorno de trabajo, elementos que son necesarios para lograr completar con éxito el reto de esta semana, el cual continua por la misma línea de generador y recosntructor de señales, con la diferencia de introducir mensajes personalizado.

A. ¿Qué es Namespaces?

Namespace es un mecanismo que nos permitirá iniciar dos nodos similares sin incurrir en conflictos de nombre de nodo o de tema, con la condición de compartir un mismo prefijo, por ejemplo, para dos nodos, sensor1 y sensor2, si creamos un namespace ”robot”, el nombre de estos nodos se escribiría tal que *robot1/sensor1* y *robot1/sensor2* respectivamente. [1].

1) Uso de namespaces: En general, como ya se mencionó anteriormente, esta funcionalidad es de gran utilidad para evitar conflictos de nombre entre los nodos, asi mismo, puede emplearse para organizar nodos con relaciones lógicas. [1].

2) Declaración de Namespaces: Para especificar el namespace, se puede crear desde el nodo en python de la siguiente forma:

```
class MyNode(Node):
    def __init__(self):
        super().__init__('my_node', namespace='my_namespace')
```

[1].

3) Visualizarlos: Con el comando en terminal *ros 2 node list* para ver los nodos enlazados en el mismo. [1].

4) Namespaces y múltiples nodos: Si bien definir namespaces para un pequeña cantidad de nodos puede que no implique una gran tarea, para grandes cantidades existe una acción que nos permite definir un namespaces global para cada descripción de archivo, donde cada nodo anidado heredará estas características. [1].

B. ¿Qué es Parameters?

Un parameter puede verse como un valores de configuración de nodos los cuales pueden almacenar parámetros como *bool, int64, float64, string, byte[], bool[], int64[], float64[] o string[]*. En general, son usados al inicio para la configuración de nodos y durante el tiempo de ejecución sin modificar el código, en este sentido, el parámetro tiene la misma vida útil que el nodo. Su estructura general consta de una clave, un valor y un descriptor. [2].

1) Declaración de parámetros: El nodo necesitará declarar todos los parámetros que aceptará desde el inicio, esto para reducir posibles errores más adelante. Esto dependerá del tipo de nodo, pues en algunos casos se pueden crear intancias que permiten configurar y obtener parámetros. [2]

2) Comportamiento de los parámetros: Cada parámetro puede comportarse de diferente forma, de forma predefinida necesitará declararse desde el inicio del nodo, sin aceptar modificaciones durante la ejecución, esto con el fín de evitar el ingreso de un valor diferente al tipo de valores que puede recibir el parámetro. Sin embargo, si el código lo permite y es necesario, podemos declarar el parámetro como uno dinámico, siendo este el segundo caso. [2]

3) Establecer valores:

- **Al ejecutar un nodo:** Se puede hacer mediante la línea de comandos individuales o mediante archivos YAML.
- **Al lanzar un nodo:** Se puede modificar a través de la función de lanzamiento de ROS 2. [2]

4) Interactuar con parameters:

- `/node_name/describe_parameters`: devuelve una lista de descriptores asociadas a los parámetros. [2]
- `/node_name/get_parameter_types`: a partir de una lista de nombres de parámetros, devuelve una lista con los tipos de parámetros asociados a estos. [2]
- `/node_name/get_parameters`: a partir de una lista opcional de prefijos, devuelve los parámetros disponibles con ese prefijo. [2]
- `/node_name/set_parameters`: dada una lista con nombres y valores de parámetros, intenta establecer parámetros en el nodo. Devuelve una lista del resultado de esta acción (puede ser exitosa o no). [2]
- `/node_name/set_parameters_atomically`: dada una lista con nombres y valores de parámetros, intenta establecer parámetros en el nodo. Devuelve un único resultado, por lo que si uno falla, todo falla. [2]
- `ros2 param`: es el comando que comunmente se utiliza para interactuar con los nodos que se encuentran en ejecución [2]

5) Devoluciones de llamada:

- **Establecer parámetro:** Su objetivo es brindar al usuario la posibilidad de inspeccionar el próximo cambio de parámetros y rechazarlo. [2]
- **Evento de parámetro:** Su objetivo principal es birndarle al usuario la capacidad de reacción a los parámetros que han sido aceptados. [2]

C. ¿Qué son los Custom messages?

Los mensajes personalizados son mensajes que el usuario define, los cuales le permiten extender el conjunto de tipos de datos admitidos por ROS 2, permitiendo la transmisión de información específica entre nodos fuera de los tipos de mensaje admitidos. [3]

1) Mensajes incorporados: Los custom messages no solo permiten crear nuevas definiciones de mensaje, también pueden modificar los incorporados con nuevas definiciones.

2) Recomendaciones para la creación de mensajes personalizados: Siempre es importante verificar si existe un mensaje incorporado que puedas emplear antes de crear un mensjae personalizado, y en caso de existir coincidencia, considera si es posible emplearlo de forma directa o si es necesario adaptarlo. [3]

IV. Solución del problema

Para dar solución al problema se comienza creando un paquete dentro de nuestra carpeta de proyecto, en este caso RoboInges-Manchester creada previamente con la función `mkdir <nombre_carpeta>`, de igual manera se crea la subcarpeta "Reto2", posteriormente se procede a crear el paquete con el siguiente comando en la terminal

```
ros2 pkg create --build-type ament_python
--node-name signal_gener params_reto
--dependencies rclpy std_msgs
```

Una vez echo esto, se utiliza el comando colcon build para construir el paquete, de la siguiente manera:

```
colcon build
source install/setup.bash
```

Hecho esto abrimos Visual Studio con el comando `code .`, una vez abierto el programa ingresamos a nuestra carpeta `'src/params_reto'` donde encontraremos nuestro programa .py con el nombre asignado previamente, en neustro caso `signal_gener` aquí es donde se escribe el código necesario para la creación de nuestras cinco señales y la creación de los dos topicos de 'ri type', 'ri amplitude', 'ri freq', ri offset' y 'ri phase shift', esto con el fin de que el otro nodo pueda recibirla y así procesarla. El código se muestra a continuación:

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32
from mensaje_personal.msg import Num
import math

class My_Talker_Params(Node):
    def __init__(self):
        super().__init__('ri_signal_generator')
        self.declare_parameters(
            namespace='',
            parameters=[
                ('ri_type', 1),
                ('ri_amplitude', 1.5),
                ('ri_freq', 2.0),
                ('ri_offset', 1.0),
                ('ri_phase_shift', 1.0)
            ])
        self.pub = self.create_publisher(Float32,
        'ri_signal', 1000)  # Frecuencia de publicación de 1kHz
        timer_period = 0.001  # Período de
        temporizador para 1kHz
        self.timer = self.create_timer(timer_period,
        self.timer_callback)

        self.pub_info = self.create_publisher(Num,
        'ri_signal_params', 1000)  # Frecuencia de publicación de 1kHz

        self.get_logger().info('Talker params node initialized')

    def timer_callback(self):
        ri_type = self.get_parameter('ri_type').get_parameter_value().
        integer_value
        ri_amplitude = self.get_parameter('ri_amplitude').
```

```
        get_parameter_value().double_value
        # Convertir a segundos
        time = self.get_clock().now().to_msg().nanosec / 1e9
        ri_freq = self.get_parameter('ri_freq').get_parameter_value().
        double_value
        ri_offset = self.get_parameter('ri_offset').
        get_parameter_value().double_value
        ri_phase_shift =  self.get_parameter('ri_phase_shift').
        get_parameter_value().double_value
        if ri_type == 1:
            # Sen
            signal_value = math.sin(2 * math.pi * time)
        elif ri_type == 2:
            # Generar un pulso cuadrado
            signal_value = 1.0 if (math.sin(2 * math.pi * time ) > 0)
            else 0.0
        elif ri_type == 3:
            # sen mas cos
            signal_value = math.sin(2 * math.pi * time)+math.cos(time)
        elif ri_type == 4:
            # Generar una señal coseno
            signal_value = math.cos(2 * math.pi * time )
        elif ri_type == 5:
            # Generar una señal tangente
            signal_value = math.tan(time)
        else:
            # Sen en cualquier otro caso
            signal_value = math.sin(2 * math.pi * time)

        # Publicar el valor de la señal
        msg = Float32()
        msg.data = signal_value
        self.pub.publish(msg)
        msgDato = Num()
        msgDato.type = ri_type
        msgDato.amplitude = ri_amplitude
        msgDato.freq = ri_freq
        msgDato.offset = ri_offset
        msgDato.phase_shift = ri_phase_shift
        msgDato.time = time
        self.pub_info.publish(msgDato)
        # amp = Float32()
        # amp.data = ri_amplitude
        # self.pub_datos.publish([amp,amp])


def main(args=None):
    rclpy.init(args=args)
    m_t_p = My_Talker_Params()
    rclpy.spin(m_t_p)
    m_t_p.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Posteriormente se procede a crear el nodo resposanble de reconstruir nuestra señal, que en nuestro caso se llama `signal_reconstruction` dentro de nuestra carpeta `'src/params_reto'`, este nodo será el responsable de reconstruir nuestra señal recibida a través de los dos tópicos previamente descritos. El código de este nodo se muestra a continuación:

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32
from mensaje_personal.msg import Num
import math


class MySignalReconstructor(Node):

    def __init__(self):
        super().__init__('signal_reconstructor')
        self.subscription_signal = self.create_subscription(
            Float32,
            'ri_signal',
            self.signal_callback,
            1000)
        self.subscription_params = self.create_subscription(
            Num,
            'ri_signal_params',
            self.params_callback,
            1000)
        self.publisher = self.create_publisher(Float32,
        'ri_signal_reconstructed', 1000)

        self.get_logger().info('Constructor
        params node initialized')

        self.type = 0
        self.amplitude = 0.0
        self.freq = 0.0
        self.offset = 0.0
        self.phase_shift = 0.0
        self.time = 0.0


    def params_callback(self, msg):
        self.type = msg.type
        self.amplitude = msg.amplitude
        self.freq = msg.freq
        self.offset = msg.offset
        self.phase_shift = msg.phase_shift
        self.time = msg.time

    def signal_callback(self, msg):
        reconstructed_signal = 0.0
        if self.type == 1:
            reconstructed_signal = self.amplitude
            * math.sin(2 * math.pi * self.time *
            self.freq + self.phase_shift) + self.offset
        elif self.type == 2:
            # Generar un pulso cuadrado con
            frecuencia, offset, amplitud y desplazamiento de
            fase variables
            # Calcula la señal del pulso cuadrado
            signal_value = 1.0 if (math.sin(2 *
            math.pi * self.freq * self.time + self.phase_shift)
            > 0) else 0.0
            # Ajusta la amplitud y el offset
            reconstructed_signal = signal_value *
            self.amplitude + self.offset

        elif self.type == 3:
            reconstructed_signal = self.amplitude *
            math.sin(2 * math.pi * self.time *
            self.freq + self.phase_shift) +
            self.offset + math.cos(self.time)
        elif self.type == 4:
            # Generar una señal coseno con frecuencia
            y desplazamiento de fase variables
            reconstructed_signal = self.amplitude *
            math.cos(2 * math.pi * self.time *
            self.freq + self.phase_shift) + self.offset
        elif self.type == 5:
            # Generar una señal tangente con
            frecuencia y desplazamiento de fase variables
            reconstructed_signal = self.amplitude *
            math.tan(2 * math.pi * self.time *
            self.freq + self.phase_shift) + self.offset
        else:
            # Si el tipo no es reconocido, generar
            una señal senoidal de 2 Hz y amplitud de 0.5
            por defecto
            reconstructed_signal = self.amplitude *
            math.sin(2 * math.pi * self.time *
            self.freq + self.phase_shift) +
            self.offset

        reconstructed_msg = Float32()
        reconstructed_msg.data = reconstructed_signal
        self.publisher.publish(reconstructed_msg)


def main(args=None):
    rclpy.init(args=args)
    signal_reconstructor = MySignalReconstructor()
    rclpy.spin(signal_reconstructor)
    signal_reconstructor.destroy_node()
    rclpy.shutdown()
```

```
if __name__ == '__main__':
    main()
```

Dado que este nodo fue creado directamente en Visual Studio dentro de la carpeta 'src/params_reto', es necesario agregar la siguiente línea de código dentro del archivo setup.py dentro de entry_points={} resultando de la siguiente manera:

```
entry_points={
        'console_scripts': [
            'signal_generator = courseworks.signal_generator:main',
            'signal_reconstruction = params_reto.signal_reconstruction:
            main',
        ],
    },
```

Dado que a su vez requerimos de la creación de un launcher para ejecutar tanto los nodos, así como 'rqt_plot' y 'rqt_graph' modificamos el archivo params_reto/package.xml para incluir las siguientes líneas de código:

```
<exec_depend>ros2launch</exec_depend>
<exec_depend>mensaje_personal</exec_depend>
```

De igual forma se añadieron las siguientes líneas de código al archivo 'setup.py' dentro de entry_points={}:

```
'signal_gener = params_reto.signal_gener:main',
'signal_reconstruction = params_reto.signal_reconstruction:
main',
```

De igual manera se creó una carpeta con el nombre mensaje_personal dentro de la carpeta 'src', en esta se agregó el archivo 'msg/Num.msg', que incluye lo siguiente:

```
int64 type
float32 amplitude
float32 freq
float32 offset
float32 phase_shift
float32 time
```

Este archivo nos servirá para crear otro tipo de dato, los cuales son utilies para una transmisión más sencilla de los datos por medio de los tópicos.

De igual manera se creó un archivo 'params.yaml' dentro de la carpeta src/params_reto/config que incluye los siguientes datos, para la inicialización de los parametros de las señales, dicho código se muestra a continuación:

```
ri_signal_generator:
  ros__parameters:
    ri_type: 2
    ri_amplitude: 5.0
    ri_freq: 10.0
    ri_offset: 7.0
    ri_phase_shift: 3.0
```

Por último se creo el archivo params_launch.py dentro de config/launch en este archivo se generan las descripciones de lanzamiento para los diferentes nodos, así como de 'rqt_plot' y 'rqt_graph'.

Después de ello se tienen que escribir las siguientes líneas de código en la terminal para lanzar el launcher:

```
colcon build
source install/setup.bash
ros2 launch params_reto params_launch.py
```

Posterior a esto se deben ejecutar ambos nodos, así como rqt graph y plot.

Para modificar la señal a desplegar podemos escribir la siguiente línea de comando, con el número a poner correspondiendo a cada una de las señales enviadas:

```
ros2 param set /ri_signal_generator
ri_type <1-5>
```

De igual manera podemos modificar los otros parámetros de las señales para alterar su apariencia.

V. Resultados

Como resultado obtenemos dos nodos activos que por medio de un archivo launch se ejecutan los codigos para inicializar los nodos y mostrar la grafica con las señales, en total hay 5 señales que se pueden graficar, así como modificar su amplitud, tipo y frecuencia con algunos comandos desde terminal. A continuación se muestran las imagenes del resultado, en la cual se puede apreciar los comandos realizados en la terminal, así como una grafica con dos señales que se grafican al mismo tiempo, una es de la señal original que se genera en el nodo de "ri signal generator" y la otra señal es de la señal reconstruida con los datos de incio o ya con los datos modificados desde la terminal

A. Ejecución del programa

En la figura 1 se puede observar el comando que se corrio en la terminal para el archivo launch, de esta forma se inicializaron los nodos y las graficas solo con correr ese comando y con estos dos comando previos.

```
colcon build
source install/setup.bash
```

B. Graficación señales

Para la graficación de señales se utiliza la herramienta rqt plot, esto con la intención de gráficar las señales, tanto la producida por el nodo 'signal generator', así como la reconstruida por el nodo 'signal reconstructor'. Para esto se utiliza la siguiente linea de comando, la cual no es necesario escribir en la terminal, porque con el archivo launch previamente creado solito se ejecuta.

```
ros2 run rqt_plot rqt_plot
```

En la figura 2 se muestran las señales como van cambiando, dependiendo de la seleccionada desde la terminal como se puede ver en la figura 4. La señal número 5 se muestra en otra la figura 3 debido a que es muy grande su amplitud.

```
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 launch params_reto params_launch.py
[INFO] [launch]: All log files can be found below /home/ulises/.ros/log/2024-02-
28-21-09-20-838065-ainse-HP-EliteDesk-705-G2-SFF-10554
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [signal_gener-1]: process started with pid [10555]
[INFO] [signal_reconstruction-2]: process started with pid [10557]
[INFO] [rqt_graph-3]: process started with pid [10559]
[INFO] [rqt_plot-4]: process started with pid [10561]
[signal_reconstruction-2] [INFO] [1709176161.200138981] [signal_reconstructor]:
Constructor params node initialized
[signal_gener-1] [INFO] [1709176161.200676070] [ri_signal_generator]: Talker par
ams node initialized
```

Fig. 1. Comando de inicialización

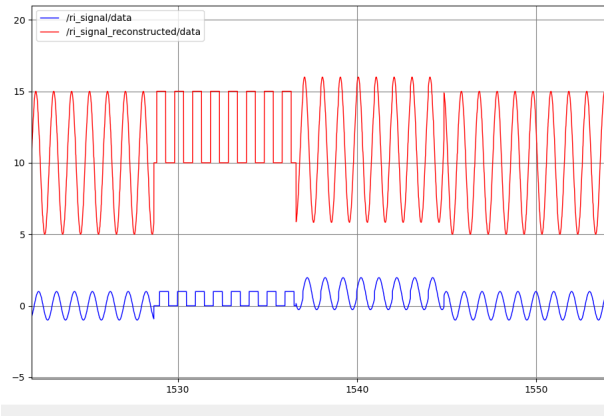

Fig. 2. Graficación de señales

C. Señales reconstruidas

Dentro de la graficación de las señales se mostro la reconstrucción con los datos enviados desde la terminal, para modificar el tipo de señal, la amplitud, frecuencia, etc. En la figura 5 se puede ver la señal generada y en la figura 6 se muestra la señal ya reconstruida con los siguientes comandos que estan en la figura 7

De la misma manera se trabajo con la señal de tipo 2, que es cuadrada a comparacion del tipo 1 que es senoidal. Primero se puede observar la señal generada en la figura 8, luego la señal reconstruida en la figura 9 y por ultimo se muestran los comandos con los cuales se modificarons los atributos de la

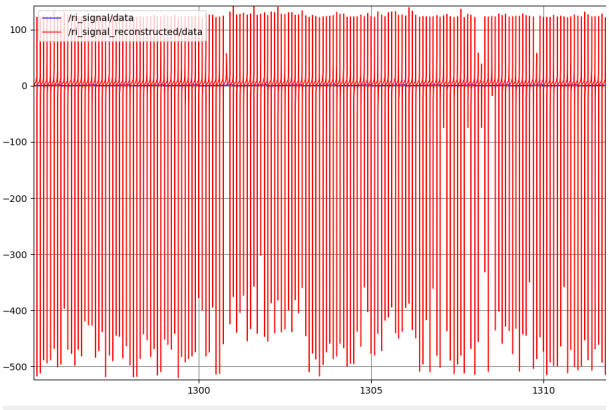

Fig. 3. Graficación de señal tipo 5

```
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 param set /ri_signal_generator ri_type 1
Set parameter successful
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 param set /ri_signal_generator ri_type 2
Set parameter successful
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 param set /ri_signal_generator ri_type 3
Set parameter successful
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 param set /ri_signal_generator ri_type 4
Set parameter successful
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 param set /ri_signal_generator ri_type 5
Set parameter successful
```

Fig. 4. Cambio de tipo de señal

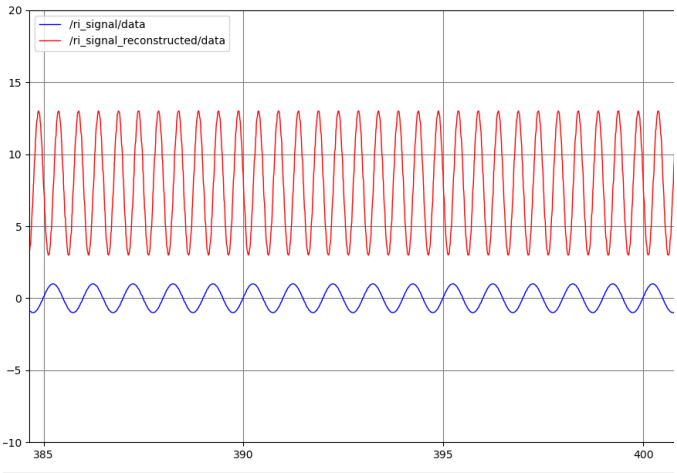

Fig. 5. Señal 1 generada

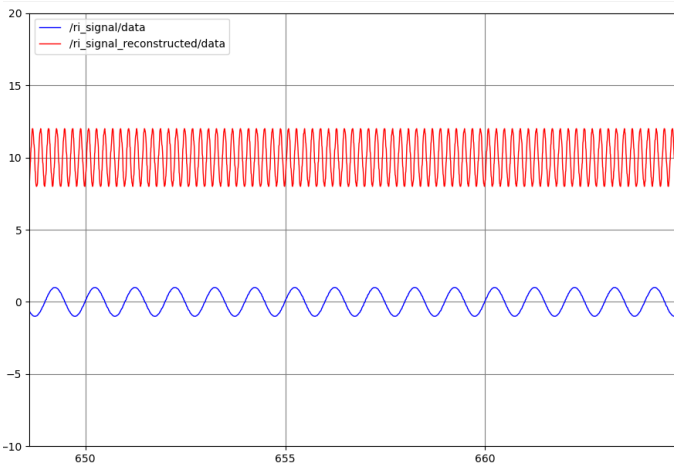

Fig. 6. Señal 1 reconstruida

```
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 param set /ri_signal_generator ri_offset 10.0
Set parameter successful
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 param set /ri_signal_generator ri_freq 5.0
Set parameter successful
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 param set /ri_signal_generator ri_amplitude 2.0
Set parameter successful
ulises@ainse-HP-EliteDesk-705-G2-SFF:~/Documents/Sexto_semestre/RoboInges-Manche
ster/Reto2$ ros2 param set /ri_signal_generator ri_phase_shift 5.0
Set parameter successful
```

Fig. 7. Comandos para modificar la señal 1

señal, como se puede ver en la figura 10

D. Visualización de nodos y tópicos

Para la graficación de los nodos y tópicos se utilizó la herramienta 'rqt graph', aquí es donde se muestra la conexión y los mensajes que se transmiten entre los diferentes nodos, es una gran herramienta para hacer depuración y visualzar las difetentes conexiones de nuestro paquete en ROS. Para esto se utiliza la siguiente línea de comando que se corre con el archivo launch:

```
ros2 run rqt_graph rqt_graph
```

Y el resultado se puede observar en la figura 11.

E. Interpretación de resultados

Como podemos observar en la figura 2, se despliegan las señales, aquella generada por el nodo 'signal generator' mostrada en azul, así como aquella reconstruida por el nodo 'signal reconstructed' mostrado en rojo, En esa figura se puede apreciar como fueron cambiando las señales al cambiar el tipo de señal desde la terminal, y despues en la sección de señales reconstruidas se observa como se modifica la señal seleccionada con los parametros dados por el usuario. De igual forma podemos observar en la figura 11 ambos nodos, el generador así como el reconstruidor de las señales, en conjunto con el tópico '/signal' y '/signal params' que es utilizado para enviar nuestra señal y los parametros entre ambos nodos.

VI. Conclusiones

Como conclusión obtenemos el desarrollo de un proyecto basado en ROS 2, donde se implementaron nodos y topicos para transmitir una señal y su posterior modificación, en equipo se logró cumplir los objetivos establecidos, principalmente gracias al uso de herramientas de ROS 2 para compilación, ejecución y obtención de información. A lo largo del proceso se identificó un área de mejora en el uso de graficación de ros 2, debido a que llegaba a mandar un error extraño de vez en cuando y no encontramos la razón

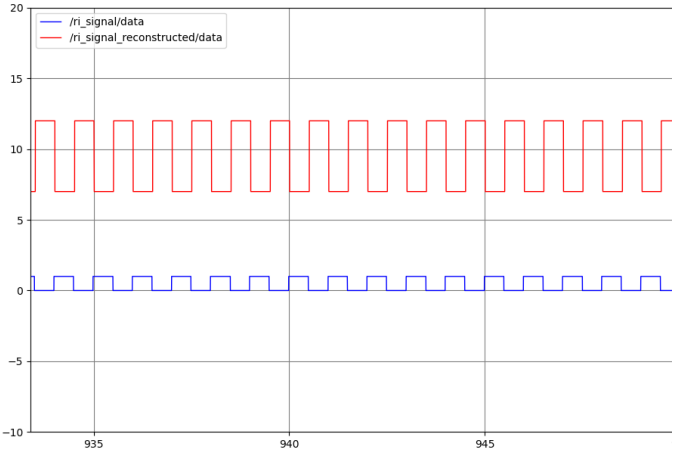

Fig. 8. Señal 2 generada

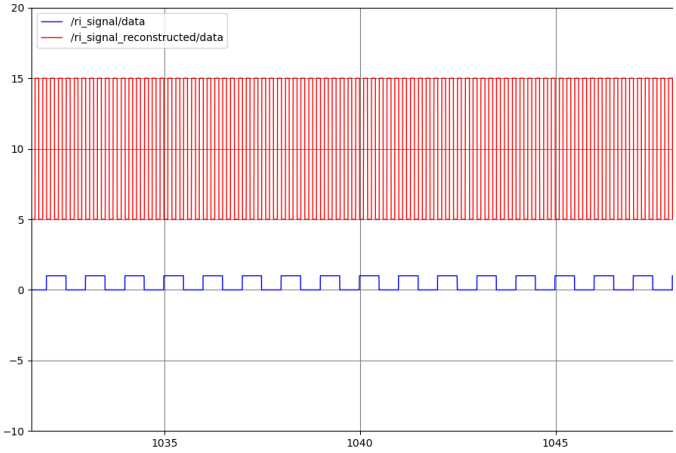

Fig. 9. Señal 2 reconstruida

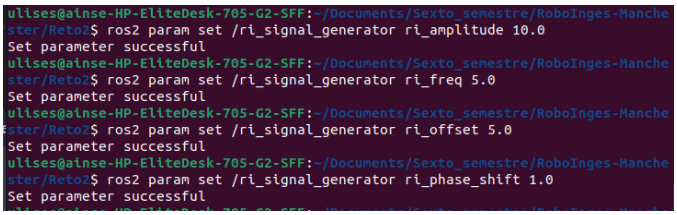

Fig. 10. Comandos para modificar la señal 2

de ese error. Finalmente, se pudo obbtener la gráfica con ambas señales, la original y las 5 señales modificadas, lo que demostraba el éxito del reto propuesto.

References

[1] Managing large projects — ROS 2 Documentation: Foxy documentation. (s.f.). link.

[2] About parameters in ROS 2 — ROS 2 Documentation: Foxy documentation. (s.f.). link.

[3] ROS Custom Message Support - MATLAB y Simulink - MathWorks América Latina. (s.f.). link.

[4] rqt plot - ROS Wiki. (s. f.). link.

[5] rqt graph - ROS Wiki. (s. f.). link.

[6] ManchesterRoboticsLtd. (s. f.). GitHub - ManchesterRoboticsLtd/TE3001B-Robotics-Foundation-2024. GitHub. link.

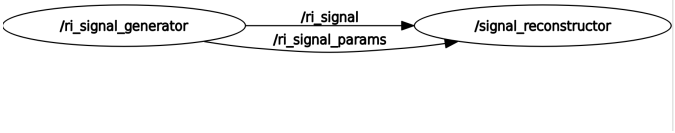

Fig. 11. Conexiones