Práctica de Laboratorio 2

Adrián Lozano González
Ingeniería en Robótica y Sistemas Digitales
Instituto Tecnológico y de
Estudios Superiores de Monterrey
CDMX, México
A01661437@tec.mx

Israel Macías Santana
Ingeniería en Robótica y Sistemas Digitales
Instituto Tecnológico y de
Estudios Superiores de Monterrey
CDMX, México
A01027029@tec.mx

***Abstract*—En esta práctica se aborda la programación y ejecución de un programa elaborado en ROS2 para observar las funcionalidades básicas a través de la creación de paquetes, nodos, tópicos y mensajes. Para ejemplificar y mostrar cada uno de ellos, se creó un paquete con dos nodos, un *publisher* y un *subscriber*, de forma que el *publisher* mande una señal sinusoidal al *subscriber* y este, posteriormente sea capaz de modificar la señal y publicarla de nuevo. El resultado se tendría que ver en una interfaz que grafique ambas señales. El objetivo final es generar un archivo *launch* que despliegue el paquete creado con ambos nodos y la interfaz que muestre la gráfica de ambass señales.**

***Index Terms*—ROS2, nodos, tópicos, mensajes, señal**

I. Introducción

En el marco de esta práctica se explora todo lo referente a los fundamentos ROS2 mediante comunicacijón básica entre dos nodos. Al ser una herramienta esencial y con gran crecimiento en los últimos años, ROS plantea una excelente forma de gestionar sistemas robóticos a cualquier nivel, es por ello que resulta necesario conocer la forma de implementación de cualquier sistema de robótica embebida, esto significa conocer los fundamentos de ROS, en particular ROS2 dada su reciente y conveniente escalabilidad. A lo largo de la presente práctica se discutirán temas y conceptos sumamente relevantes a la hora de trabajar con un $middleware$ como lo es ROS, todo lo referente a la comunicación, interpretación y objetivo de este intercambio de datos. Será necesario comprender y aprender a implementar un espacio de trabajo y paquete para realizar comunicación entre diversos elementos con diferentes tópicos y mensajes. Todo lo anterior con el fin de realizar un sistema básico de procesamiento de señales entre dos nodos.

II. Marco Teórico

A. ROS2

ROS2 son las siglas para referirse a *Robot Operating System* el cuál es un conjunto de librerias y herramientas de software para construir aplicaciónes robóticas. ROS2 es un $middleware$, es decir, un software que permite la interacción o conectividad entre distintas aplicaciones en una red que se basa en un mecanismo de publicar/suscribir para gestionar mensajes y comunicación entre diversos componentes. A diferencia de ROS1, ROS2 tiene una librería base escrita en C con librerías montadas sobre ella, es decir, permite mayor diversidad de lenguajes y se vuelve mayormente escalable y adaptable. Los lenguajes que tienen soporte en ROS2 ya no solo serían $Python$ o $C++$, si no que se puede incorporar $Java$ o $C\#$. Los programas de ROS2 se construyen en base a nodos, tópicos, servicios y mensajes. Todo ello forma el $ROS\ graph$, es decir, toda la red de nodos en un sistema de $ROS2$, sus conexiones y cómo se comunican.

B. Nodos

Un nodo es un componente del $ROS\ graph$ y debe de ser responsable de un único propósito modular. Los nodos pueden mandar y recibir información por medio de tópicos, servicios, parámetros o acciones. Un nodo puede publicar mensajes por diferentes tópicos y simultaneamente suscribirse a distintos tópicos de otros nodos. Para mantener la comunicación, los nodos sufren un proceso de **descubrimiento**. Cuando un nodo es inicializado, anuncia la presencia a todos los nodos en la red con el mismo dominio ROS, con base en esto se determina si los nodos se pueden comunicar. Los nodos alertan de su presencia y de su inactividad para que la comunicación tenga buena calidad de servicio.

C. Tópicos

Los tópicos son uno de los principales medios por los cuales se transmite la información entre nodos y a las diferentes partes del sistema. Es una manera de estandarizar los mensajes y de comunicación entre nodos, de esta manera la infoermación y datos se pueden interpretar de acuerdo a los tópicos que un nodo este suscrito o publicando.

D. Servicios

Los servicios son una forma alternativa de comunicación entre nodos donde en vez de basarse en publicar/suscribir, se basa en un modelo de solicitud/respuesta. A diferencia de la comunicación por tópicos donde los nodos se suscriben a un flujo de datos, los servicios solo devuelven información cuando es solicitada por un cliente. Esto permite controlar el flujo de mensajes.

E. Mensajes

Los mensajes son el tipo de dato que utilizan los tópicos para comunicarse entre ellos. Son un tipo de estructura que recibe tipos de datos, enteros, *floats*, booleanos, chars, *double*, strings, uint16, uint64, entre otros.

F. Librerías de cliente

Las librerías de cliente de ROS son elementos sumamente relevantes para la escalabilidad y para la interpretación del software. Estas librerías permiten a los nodos escritos en distintos lenguajes comunicarse entre sí. La librería base es $ROS\ client\ library\ (RCL)$ que implementa los requerimientos para las $APIs$ de distintos lenguajes, esto permite mayor facilidad de escribir ciertas librerías específicas y estandarizar el comportamiento. Las librerías cuales ROS2 da mantenimiento son las de $Python$ **(rclpy)** y $C++$ **(rclcpp)**

G. Launches

El sistema de launches permite configurar un sistema y ejecutarlo como se describa. La configuración de estos lanzamientos determina los programas a correr, dónde correrlos, argumentos, entre otros parámetros. Los archivos de $launch$ son escritos en $Python$, XML o $YAML$ y pueden comenzar distintos nodos y actuar en distintos eventos. Usar $launches$ facilita la ejecución de programas grandes.

III. Metodología

Los materiales requeridos para el desarrollo de la práctica son:

- Una computadora con Linux
- ROS2 Humble

En esta sección se aborda la explicación metodológica de la práctica, desde la configuración del ambiente de trabajo hasta desplegar el archivo *launch* que contiene todos los archivos ejecutables.

Primeramente se tiene que preparar un espacio de trabajo donde guardar el paquete que se vaya a crear, después, este espacio de trabajo se tiene que compilar y generar una carpeta interna dónde guardar los archivos ejecutables y posteriormente se crea el paquete de ROS2 dentro de la carpeta. Para esto, se puede usar el comando *ros2 pkg create* desde la terminal de LINUX para indicar el tipo de lenguaje que se usa (Python o C++), el nombre del primer nodo, la ubicación, las dependencias, licencias entre otros parámetros. Una vez hecho esto, el espacio de trabajo está listo para que se creen los nodos y cualquier archivo ejecutable.

A. Nodo: Signal Generator

El nodo de *Signal Generator* tiene la función de *pusblisher* dentro del paquete, por lo que es a través de él por donde la información va a ser transmitida al resto de suscriptores. Dentro del nodo se tiene que especificar el tópico por el que se van a mandar los mensajes, así como el tipo de mensaje que se va a transmitir.

Este nodo tiene la función de generar una señal sinusoidal dentro de un *timer* que publique los datos a través del tópico con una frecuencia de $10Hz$. Para este nodo se utilizaron dos tópicos diferentes, el tópico *signal* y el tópico *time*. En ellos se publican los datos del tiempo y de la amplitud de la onda respectivamente. En el nodo se tiene que programar el $publisher$ usando la función ***create_publisher*** dónde se especifica el tipo de dato que va a tener el tópico, y el nombre. En este caso, para ambos tópicos *signal* y *time*, el tipo de dato que se va a compartir es un $float$ de 32 bits.

Además, para poder mandar los datos, se tiene que generar un *timer* cada 100 milisegundos con la función ***create_timer***, la cual recibe la frecuencia con la que se va a lanzar el *timer* y la función que va a estar dentro. En la función de $timer\_callback$ se le va a asignar a un atributo llamado $msg\_signal.data$ los datos de la señal sinusoidal que se va a transmitir por el tópico, mientras que a otro atributo llamado ***msg_time.data***, se le va a asignar el valor del tiempo que se definió anteriormemete, el mismo que se usó para que la función del *callback* se mande llamar. Esta asginación permite que la onda sinusoidal se genere con el mismo valor de tiempo que se está trasnmitiendo por el tópico. Finalmente, para que ambos mensajes se publiquen, se tienen que mandar los atributos de ***msg_signal.data*** y ***msg_time.data*** por medio de la función $publish$ para que el mensaje se envíe por su tópico respectivo.

B. Nodo: Processed Signal

El nodo de Processed Signal es un nodo que cuenta con ambas funcionalidades de *pusblisher* y de *subscriber*, pues está suscrito a los dos tópicos que publica el nodo Signal Generator y además publica por medio de otro tópico la señal modificada. Al igual que el nodo anterior, éste publica y recibe la información por medio de un *timer* cada 100 milisegundos. Éste segundo nodo modifica la fase, la amplitud y el desplazamiento de la onda original y la publica por medio del tópico llamaddo *proc signal*. La onda modificada recibe un desplazamiento de 1.5, un adelanto de fase de π, y una reducción de amplitud a la mitad. En el nodo se tiene que programar, al igual que el nodo anterior, el tópico por dónde se va a transmitir la información que contienen los datos de la señal modificada por medio de la función ***create_publisher***. En este caso, el tópico se llama *proc_signal*, y el tipo de dato es un $float$. Además, este nodo va a tener dos suscripciones a los tópicos que generó el nodo anterior, para esto se usa la función ***create_subscription*** donde se describe el tópico al que se suscribe, el tipo de dato que recibe el suscriptor, y la función de *callback* con la que se van a estar recibiendo los mensajes.

Dentro de los *callback* de cada suscripción se tienen que guardar los datos que recibe el nodo dentro de un atributo. En el caso del tópico de la señal, se guarda el valor de la amplitud de la onda y en el tópico del tiempo se guarda el dato del tiempo que envía el otro nodo. Aprovechando que los *callback* tienen una frecuencia de lanzamiento de 10Hz, se puede generar la señal modifcada a partir de los datos que se están recibiendo y, de una vez publicarla por medio del tópico *proc_signal*.

Para generar la señal modificada, cómo se describe en la ecuación 1, se definen tres variables que guardan los valores del desplazamiento, el adelanto de fase y la amplitud. En el caso del desplazamiento y el desfase, los valores se definieron manualmente, mientras que para la amplitud, se definió un atributo de media amplitud dónde se modificó el valor orginal que recibe el nodo por medio del tópico de *signal* multiplicándolo por 0.5 para tener una nueva onda con la mitad de la amplitud original, con un adelanto de *pi rads* o $90°$, y un desplazamiento vertical de 1.5 para que toda la onda sea positiva.

$$g(t) = Asin(t + p) + k \tag{1}$$

C. Archivo Launch

Este archivo de Python será el encargado de lanzar el nodo de $signalgenerator$ y $process$ de manera simultánea, además de dos paquetes relevantes para la visualización como lo son rqt_graph y rqt_plot. Este archivo se encuentra dentro de una carpeta $launch$ dentro del paquete y será necesario estructurarlo de esta manera con el fin de poder lanzar todos los archivos especificados. Al tener el control de la gestión de archivos, el archivo de launch debe de tener dependencias del sistema operativo (os), de descripción del lanzado y creación de nodo para ser corrido de la misma manera.

Cada nodo será creado a partir de la clase $Node$ proveniente del paquete $launch\_ros.actions$ y tendrán el atributo de paquete asignado como el nombre del paquete actual ($courseworks$), el archivo que será lanzado correspondientemente y la salida, es decir, donde se visualizará la información del nodo, por ello será $screen$. Finalmente, dentro de una lista descriptiva siendo parámetro de $LaunchDescription$, se enlistarán todos los nodos generados, en este caso serán el nodo de $signal\_generator$, $process$, rqt_graph y rqt_plot. Por parte del archivo $launch$ el propósito será devolver esta lista descriptiva.

Los $launches$ pueden ejecutar cualquier archivo ya sea de $Python$, XML o $YAML$. Es por ello que en el archivo de configuración $setup.py$ es necesario especificar que esos archivos son los que podrán ser lanzados especificando además el nombre del paquete y todo el $path$ para encontrar tales archivos. Completando este paso y añadiendo la dependencia necesaria de $ros2launch$, todos los nodos estarán listos para ser lanzados.

IV. Resultados

En los resultados del ejercicio se puede observar el lanzamiento de los archivos ejecutables que se describieron en la lista descriptiva del archivo *launch*. Entre ellos se encuentran los mensajes de *log in* de cada uno de los nodos en la terminal y las interfaces gráficas de NodeGraph y el rqt_plot. Para saber que los nodos fueros exitosamente ejecutados se tiene que ver el mensaje *logger* en la terminal que se programó en cada uno de los nodos. En la figura 1 y en la figura 2 del apéndice se pueden ver los mensajes de inicialización de los nodos, además de la información que cada uno de los nodos transmiten respectivamente por sus tópicos.

En la figura 4 del apendice se aprecia la interfaz de *rqt_plot* con la señal via el tópico de *signal*, pues es la información que publica el nodo de *Signal Generator*. Para verificar que la señal modificada es correcta y tiene el comportamiento esperado, se tiene que añadir el dato que se quiere observar dentro de la interfaz de *rqt_plot* sobre la gráfica orginal. Para esto se describe el tópico al que se quiere acceder y el dato que se desea ver en la barra superior de la interfaz. En la figura 5 del apendice se observan ambas gráficas correspondientes a dos tópicos distintos.

Finalmente, para comprobar que el lanzamiento del archivo *launch* fue correcto, al ejecutar el comando

```
ros2 launch basic_comms launch.py
```

Todos los archivos ejecutables que se describieron en la lista descriptiva van a desplegarse en la pantalla. En la figura 6 del apéndice se muestran ambas interfaces gráficas, *NodeGraph* y *rqt_plot*.

V. Análisis de resultados

Con cada una de las pantallas de resultados disponibles se pueden analizar y observar los comportamientos de las señales, y la creación de los nodos y tópicos. Dentro de la interfaz de *NodeGraph* aparecen dos burbujas de nodos, lo cuál indica que los dos nodos, *Signal Generator* y *Processed Signal*, se crearon satisfactoriamente. Además, del nodo de *Signal Generator* salen dos flechas con los nombres de $\backslash signal$ y $\backslash time$, los cuales indican los tópicos que tiene ese nodo. Seguidamente, de los tópicos se observan dos flechas con dirección al nodo de *Processed Signal*, indicando que dicho nodo tiene dos suscripciones a ambos tópicos. Finalmente, se aprecia cómo de este nodo sale otra flecha con el nombre de $\backslash proc\_signal$ la cuál pertenece al tópico por dónde se manda la información de la señal procesada. Esta interfaz gráfica permite observar que nodos y tópicos se encuentran activos, así como los *publishers* y *subscribers* que tienen los nodos.

Para observar que las señales se estén generando de forma correcta, la interfaz *rqt_plot* ayuda a corroborar los datos de ambos tópicos. Si se selecciona la opción de mostrar la información del tópico $\backslash signal$, se puede apreciar en azul la gráfica sinusoidal que se genera en el primer nodo, la cual tiene un rango de -1 a 1, sin desplazamientos ni desfases. Mientras que, si añada y se selecciona la opción de ver los datos del tópico $\backslash proc\_signal$, se aprecia en rojo la señal orginal modificada, la cuál tiene un desfase de $90°$, un desplazamiento vertical de 1.5 para mantener la señal postiva, y una reducción de la amplitud a la mitad.

VI. Conclusiones

- Conclusión Israel Macías: A lo largo de la práctica pude observar y entender los distintos componentes básicos del software ROS2 para crear paquetes y comunicar

información desde distintos puntos en la zona de trabajo. Crear nodos y tópicos ayuda mucho a entender como la información fluye y se distribuye dependiendo las publicaciones y suscripciones. Creo que esto es fundamental para entender la creación de ambientes de trabajo más complejas con aplicaciones a futuro.

- Conclusión Adrián Lozano: Tras haber concluído la implementación del procesamiento básico de una señal sinusoidal mediante la comunicación de dos nodos, se pudieron reforzar los fundamentos básicos de ROS2. Se crearon dos nodos con distintos publishers y tipos de mensaje. Se pudo lograr comunicación por medio de tópicos comprendiendo los tipos de datos y un procesamiento básico de un seno. Se comprendió la facilidad de crear archivos launches para facilitar la ejecución de diversos nodos.

REFERENCES

[1] Venkatadri A. (2023) ROS 1 vs ROS 2 What are the Biggest Differences?, Model-prime, Recuoerado de: https://www.model-prime.com/blog/ros-1-vs-ros-2-what-are-the-biggest-differences.

[2] De basqueGeek, V. T. L. E. (2018, 11 enero). ROS, parte II – Servicios, mensajes y topics. Geek Gasteiz. https://geekgasteiz.wordpress.com/2018/01/10/ros-parte-ii-servicios-mensajes-y-topics/

[3] Understanding services — ROS 2 Documentation: Foxy documentation. (s.f.). https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

[4] Creating a launch file — ROS 2 Documentation: Foxy documentation. (s.f.). https://docs.ros.org/en/foxy/Tutorials/Intermediate/Launch/Creating-Launch-Files.html

[5] M, G. (2022, 4 octubre). ROS2: hacia la nueva generación de robótica. QNV Solutions. https://qnv.com/2019/11/20/ros2-hacia-la-nueva-generacion-de-robotica/#:~:text=ROS%202%20(Robot%20OperatingSystem%202,prototipos%20hasta%20desarrollo%20y%20producci%C3%B3n.

[6] ¿Qué es el middleware? — IBM. (s.f.). https://www.ibm.com/mx-es/topics/middleware#:~:text=el%20siguiente%20paso-,%C2%BFQu%C3%A9%20es%20el%20middleware%3F,aplicaciones%20en%20una%20red%20distribuida.

[7] Creating a package — ROS 2 Documentation: Foxy documentation. (s.f.). https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html

[8] Concepts — ROS 2 Documentation: Foxy documentation. (s.f.). https://docs.ros.org/en/foxy/Concepts.html#nodes

Enlace al video: https://www.youtube.com/watch?v=qcYtkMOhCvI

Apéndice

```
alogo@alogo-Lenovo-IdeaPad-S145-15IWL:~/ros2_ws$ ros2 run courseworks signal_generator
[INFO] [1708900313.471592923] [signal_generator_node]: Signal Generator node successfully initialized!
[INFO] [1708900313.550196492] [signal_generator_node]: Tiempo: 0.1 Senial: 0.09983341664682815
[INFO] [1708900313.650017681] [signal_generator_node]: Tiempo: 0.2 Senial: 0.19866933079506122
[INFO] [1708900313.750503981] [signal_generator_node]: Tiempo: 0.30000000000000004 Senial: 0.2955202066613396
[INFO] [1708900313.850554783] [signal_generator_node]: Tiempo: 0.4 Senial: 0.3894183423086505
[INFO] [1708900313.950280955] [signal_generator_node]: Tiempo: 0.5 Senial: 0.479425538604203
[INFO] [1708900314.050355584] [signal_generator_node]: Tiempo: 0.6 Senial: 0.5646424733950354
[INFO] [1708900314.150866423] [signal_generator_node]: Tiempo: 0.7 Senial: 0.644217687237691
[INFO] [1708900314.250498304] [signal_generator_node]: Tiempo: 0.7999999999999999 Senial: 0.7173560908995227
[INFO] [1708900314.350858584] [signal_generator_node]: Tiempo: 0.8999999999999999 Senial: 0.7833269096274833
[INFO] [1708900314.450895396] [signal_generator_node]: Tiempo: 0.9999999999999999 Senial: 0.8414709848078964
[INFO] [1708900314.550756016] [signal_generator_node]: Tiempo: 1.0999999999999999 Senial: 0.8912073600614353
[INFO] [1708900314.650572022] [signal_generator_node]: Tiempo: 1.2 Senial: 0.9320390859672263
[INFO] [1708900314.750679395] [signal_generator_node]: Tiempo: 1.3 Senial: 0.963558185417193
[INFO] [1708900314.850459518] [signal_generator_node]: Tiempo: 1.4000000000000001 Senial: 0.9854497299884603
[INFO] [1708900314.950901562] [signal_generator_node]: Tiempo: 1.5000000000000002 Senial: 0.9974949866040544
[INFO] [1708900315.050886333] [signal_generator_node]: Tiempo: 1.6000000000000003 Senial: 0.9995736030415051
[INFO] [1708900315.151137806] [signal_generator_node]: Tiempo: 1.7000000000000004 Senial: 0.9916648104524686
[INFO] [1708900315.251196847] [signal_generator_node]: Tiempo: 1.8000000000000005 Senial: 0.973847630878195
[INFO] [1708900315.350932398] [signal_generator_node]: Tiempo: 1.9000000000000006 Senial: 0.9463000876874142
[INFO] [1708900315.450820430] [signal_generator_node]: Tiempo: 2.0000000000000004 Senial: 0.9092974268256815
[INFO] [1708900315.550827449] [signal_generator_node]: Tiempo: 2.1000000000000005 Senial: 0.8632093666488735
[INFO] [1708900315.650830440] [signal_generator_node]: Tiempo: 2.2000000000000006 Senial: 0.8084964038195899
[INFO] [1708900315.750600750] [signal_generator_node]: Tiempo: 2.3000000000000007 Senial: 0.7457052121767197
```

Figura 1. Terminal con mensaje *logger* y datos de los tópicos *signal* y *time*

```
alogo@alogo-Lenovo-IdeaPad-S145-15IWL:~/ros2_ws$ ros2 run courseworks process
[INFO] [1708900368.801254172] [process]: Process node successfully initialized!
[INFO] [1708900371.424102020] [process]: Señal Alterada: 1.476861695401443
[INFO] [1708900371.462713072] [process]: Señal Alterada: 1.451471005834396
[INFO] [1708900371.560292405] [process]: Señal Alterada: 1.4180483069857577
[INFO] [1708900371.660610571] [process]: Señal Alterada: 1.3779259559751003
[INFO] [1708900371.760563178] [process]: Señal Alterada: 1.332703694748994
[INFO] [1708900371.860275161] [process]: Señal Alterada: 1.2841843060363063
[INFO] [1708900371.960352532] [process]: Señal Alterada: 1.234302094008924
[INFO] [1708900372.062036453] [process]: Señal Alterada: 1.1850457113462167
[INFO] [1708900372.161530822] [process]: Señal Alterada: 1.1383787314029759
[INFO] [1708900372.260505508] [process]: Señal Alterada: 1.0961618562998754
[INFO] [1708900372.362049179] [process]: Señal Alterada: 1.0600780278890307
[INFO] [1708900372.461407880] [process]: Señal Alterada: 1.0315657776224232
[INFO] [1708900372.559760559] [process]: Señal Alterada: 1.0117617824075888
[INFO] [1708900372.659772680] [process]: Señal Alterada: 1.0014556001931347
[INFO] [1708900372.761327612] [process]: Señal Alterada: 1.0010580899513448
[INFO] [1708900372.863988288] [process]: Señal Alterada: 1.0058445628534713
[INFO] [1708900372.960634059] [process]: Señal Alterada: 1.0155685221319481
[INFO] [1708900373.060225946] [process]: Señal Alterada: 1.0301326359526533
[INFO] [1708900373.160347924] [process]: Señal Alterada: 1.0493916609537228
[INFO] [1708900373.259800313] [process]: Señal Alterada: 1.0731528024946841
[INFO] [1708900373.360051712] [process]: Señal Alterada: 1.1011790981264913
[INFO] [1708900373.460567047] [process]: Señal Alterada: 1.1331903107541466
[INFO] [1708900373.559921333] [process]: Señal Alterada: 1.1688662358796624
```

Figura 2. Terminal con mensaje *logger* y datos de los tópicos *proc_signal*

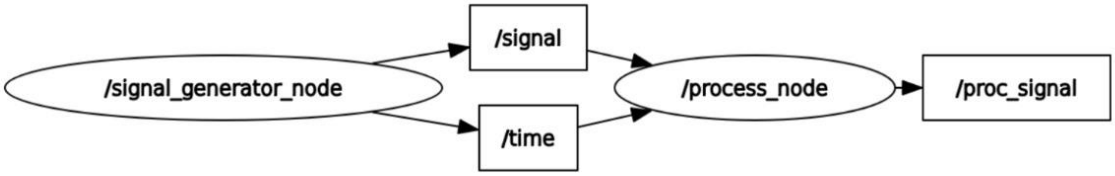

Figura 3. NodeGraph con los nodos y tópicos del paquete

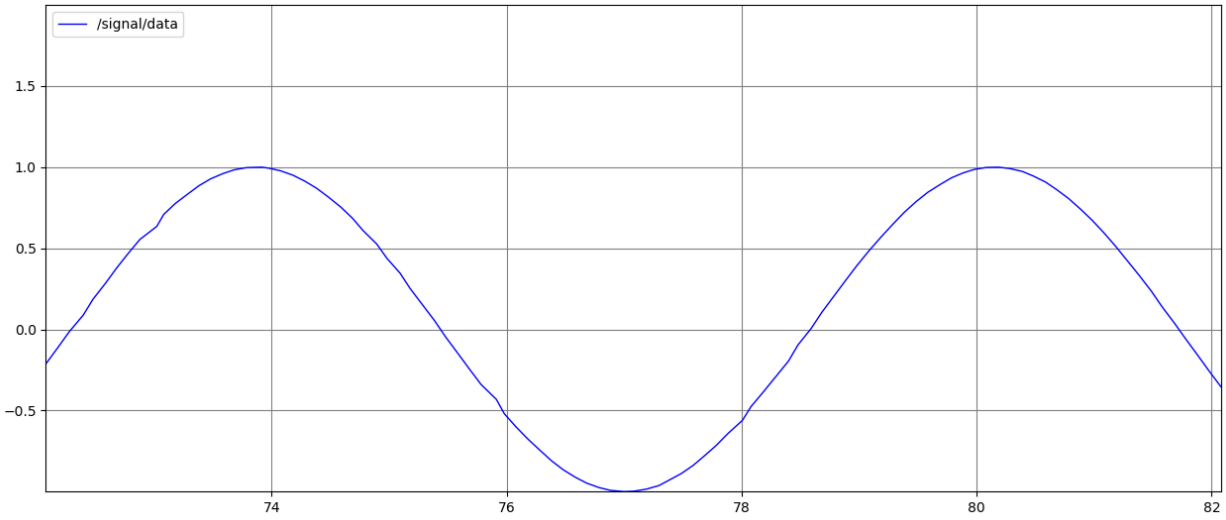

Figura 4. Señal sinusoidal original en *rqt_graph*

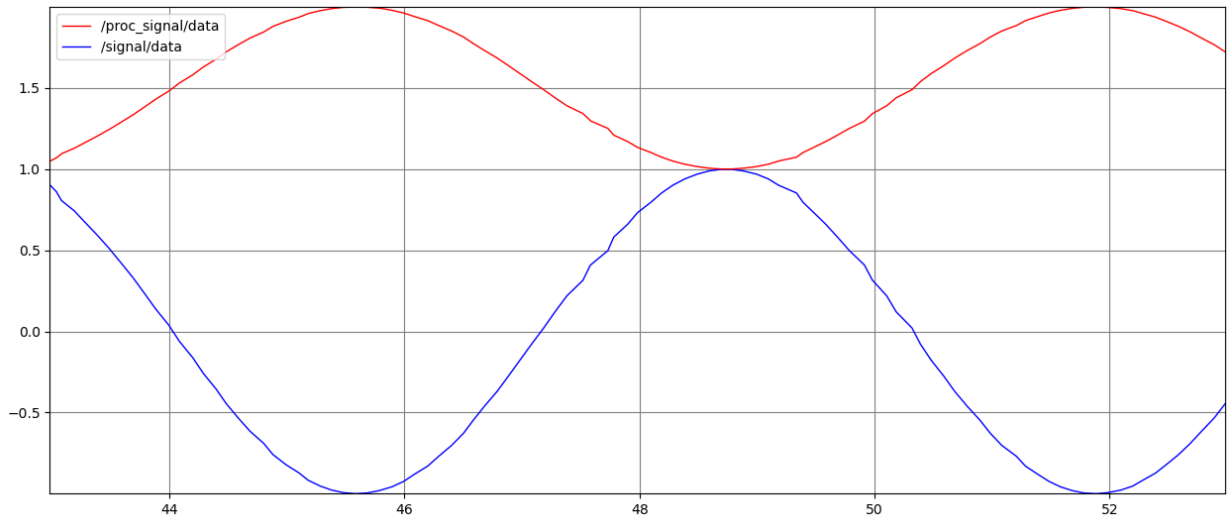

Figura 5. Señal sinusoidal original y modificada en *rqt_graph*

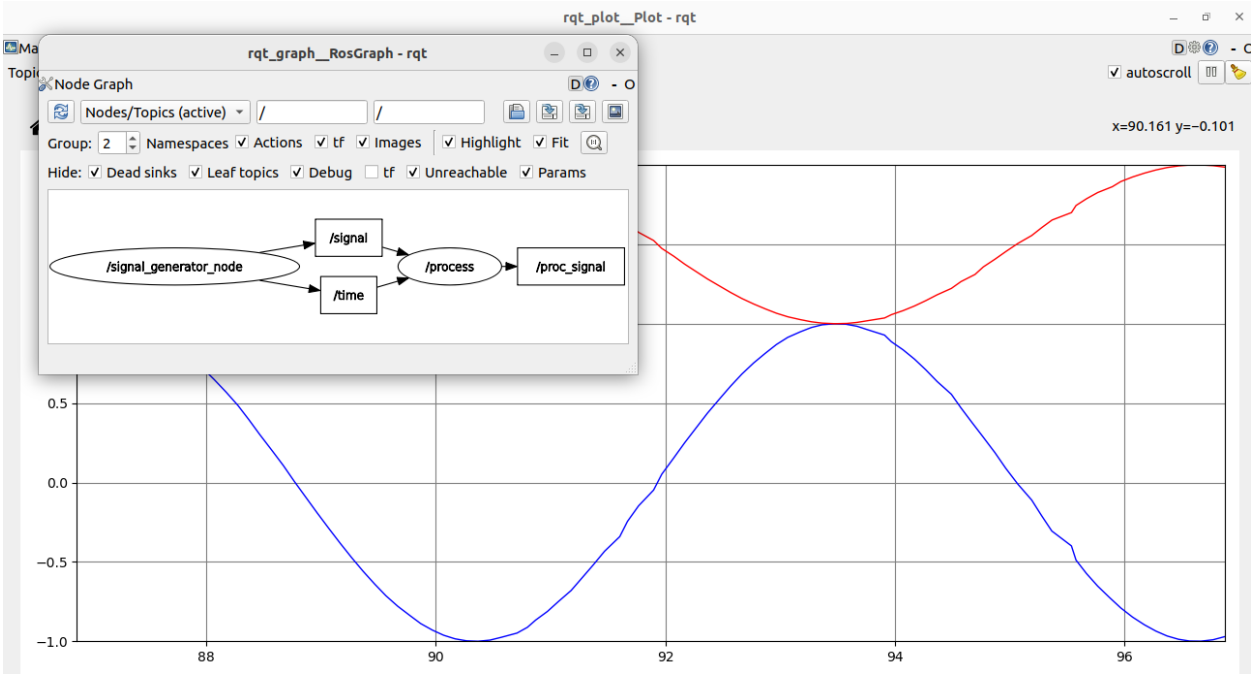

Figura 6. Ventanas de las interfaces gráficas después de ser desplegadas con el launch