Instituto Tecnológico de Estudios Superiores de Monterrey

Fundamentación de robótica & Manchester Robotics

Manchester Robotics: Challenge 1

Profesores:

Rigoberto Cerino Jiménez

Dr. Mario Martinez

Integrantes

Daniel Castillo López A01737357

Emmanuel Lechuga Arreola A01736241

Paola Rojas Domínguez A01737136

20 de Febrero de 2025

Contenido

Resumen

En el presente documento se presentarán conceptos introductorios y se hará uso de algunos comandos aprendidos para hacer uso del programa llamado ROS2 en un entorno de linux, y para eso ocuparemos paquetes, tópicos, nodos y un rqt_plot para poder observar los resultados, además de haber realizado una pequeña investigación con el fin de conocer más el programa ROS2, de lo cual presentamos que es un programa para el desarrollo de software en robótica, su estructura básica de funcionamiento y los elementos principales para una estructura básica de comunicación en este espacio de trabajo; y una vez realizado esta investigación se presentará más a detalle sobre la realización del challenge, en el que se nos pide la creación de 2 nodos, en el cual uno actuará como un actuador de señales específicamente una señal sinusoidal, el segundo como un proceso que toma la señal del primero y la modifica, generando una señal procesada y ambas señales se trazaran utilizando rqt_plot.

Objetivos

Los siguientes objetivos buscan una comprensión integral de cómo desarrollar sistemas básicos utilizando ROS 2 y sus herramientas.

- ➔ Revisar conceptos básicos de ROS.
- ➔ Reforzar conceptos básicos de ROS.
- ➔ Implementar un sistema distribuido.
- ➔ Automatizar la ejecución (Visualizar datos en tiempo real).
- ➔ Describir la estructura y funcionamiento de ROS.
- ➔ Desarrollar habilidades de depuración

Introducción

Ros o mejor conocido como Robot Operating System, es un framework (entorno de trabajo) abierto para el desarrollo de software en robótica, lo cual lo convierte en un conjunto de herramientas bibliotecas y convenciones que facilitan la creación de sistemas robóticos complejos en los cuales se pueden crear redes de comunicación multiplataforma. (ROS Wiki 2023).

La estructura básica de funcionamiento que tiene ROS se divide en cuatro puntos clave como lo son los nodos, el máster, los métodos de comunicación y los sistemas de archivos. Los nodos son procesos independientes que realizan una tarea específica (ej: control de sensores y procesamiento de datos). estos se comunican entre sí mediante mensajes a través de un sistema de publicación/suscripción o servicios (Quigley et al 2009). Por otro lado tenemos el Master o coordinador central que gestiona el registro de nodos, tópicos, servicios y parámetros, este se encarga de facilitar la conexión entre los nodos. Los métodos de comunicación cuenta con por así decirlo 3 secciones que son los topics, servicios y acciones.

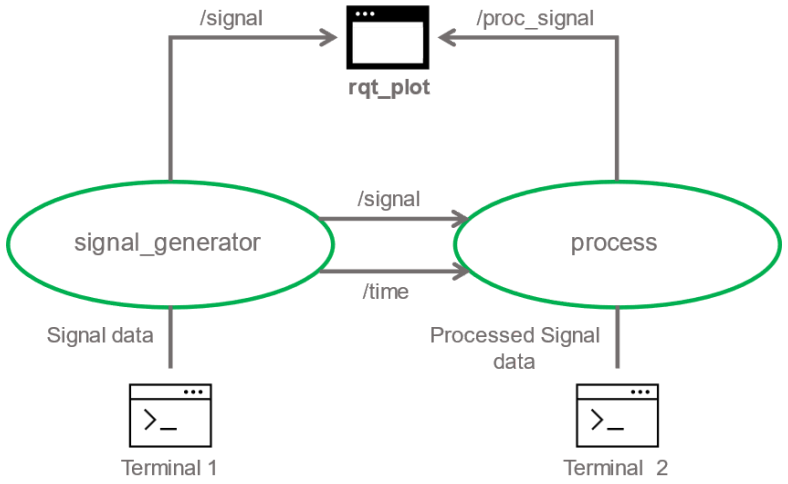
Los topics, son canales asíncronos para la comunicación unidireccional mediante mensajes, el cual usa el método publish/subscribe; en cuanto a los servicios es una comunicación síncrona para tareas puntuales; por último las acciones, similar a los servicios pero permiten tener una retroalimentación (feedback) y cancelación. Por último tenemos el sistema de archivos el cual está compuesto por paquetes y por espacios de trabajo, los primeros son unidades básicas de organización, que contienen código, dependencias, configuraciones y mensajes personalizados. Los segundos son entornos para compilar y desarrollar paquetes en nuestro caso sería con el uso de Visual Studio Code.

Elementos principales de la red de comunicaciones de ROS.

- Nodos (nodes).
    - Pueden publicar mensajes, suscribirse a temas, ofrecer servicios o ejecutar acciones.
- Temas (topics).
    - Canales nombrados donde los nodos publican o suscriben mensajes.
- Mensajes (Messages).
    - Estructuras de datos que definen el formato de la información intercambiada. (ej. std_msgs/string std_msgs/float32), aunque también se pueden definir los propios mensajes.
- Servicios (services).
    - Comunicación de tipo petición-respuesta (request-response) entre 2 nodos.
- Acciones (Actions).
    - Similar a los servicios, solo que estas son de larga duración y permite tener retroalimentación (feedback) cada cierto tiempo mientras se ejecuta.
- Parámetros (Parameters).
    - Configuraciones que se pueden pasar a un nodo para cambiar su comportamiento sin modificar su código y se guardan en un servidor central.
- Sistema DDS (Data distribution Service).
    - Se utiliza para gestionar la comunicación entre nodos, proporcionando facilidad de emparejamiento, calidad en el servicio y comunicación en general.
- Calidad de Servicio (Qos).
    - Configuraciones que determinan cómo es que los mensajes son manejados.

Solución del problema

La actividad consiste en crear dos nodos, el primero de estos (signal generator) se desarrollara como un generador de señal que comunicará los datos necesarios para una señal senoidal y una variable del tiempo, el segundo (process), tendrá que recibir los datos del primer nodo para modificarlos y generar una nueva señal con valores distintos. Ambos nodos tendrán que mostrar sus resultados usando rqt_plot, para comparar las diferencias de estas señales, como también mostrar sus resultados en diferentes terminales. Por último se debe de crear un launch para los nodos y mostrar la terminal rqt_plot al mismo tiempo.

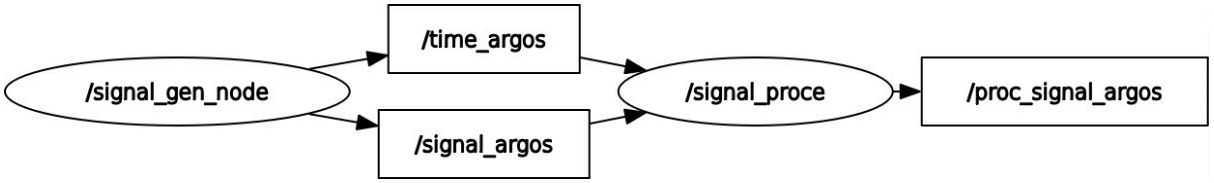


1. Crear un paquete
    - Manteniendo los nodos en un paquete, establecimos el lenguaje de python para la actividad desarrollando estos nodos dentro del paquete “signal_processing _argos”
2. Nodo “Signal_generator”
    - El primer nodo “signal_generator”, debe de tener una comunicación con el otro nodo, creamos dos mensajes que servirán como la comunicación entre estos, dichos mensajes van con los nombres /time_argos y /signal_argos.
    - Al publicar estos mensajes usaremos un formato Float32, siendo un formato de números que usaremos para ambos mensajes, siendo estos los valores que estarán comunicando, como agregado crearemos valores de “i” como el tiempo y “w” el valor de la curva senoidal.
    - El tiempo del periodo le daremos un valor de 0.1, que establece que el valor de 10 Hz para la función timer_cb
    - Haremos una función dentro del nodo, de nombre “timer_cb” que actuará como una rutina de ejecución periódica que publicará los mensajes anteriormente definidos.
    - Primero creamos el valor del mensaje que en Float32, asignando data el valor actual de i que representa el tiempo, posteriormente se publicará el mensaje para el canal de comunicación /time_argos.

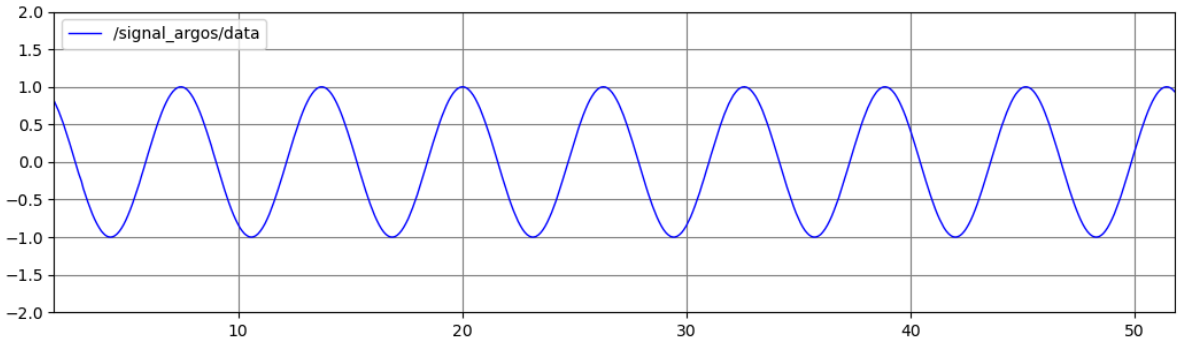
- Igualmente crearemos un data el valor actual a w para ser el mensaje que estará relacionado con la operación senoidal $y = sin(t)$ que en esta manera será *np.sin(self.i)*, posteriormente se publicará el mensaje para el canal de comunicación /signal_argos.
- Todo esto estará desarrollado en una función main se encargara de ejecutar el código, llenar el mensaje, publicarlo y detener el nodo si se detiene el programa.

3. Nodo "Signal_proce"
    - El segundo nodo "signal_proce"debe de inscribirse para poder recibir los mensajes de /time_argos y /signal_argos, como también crearemos un mensaje /proc_signal_argos que mostrará el procedimiento de modificación de la señal original
    - Al publicar estos mensajes usaremos un formato Float32, siendo un formato de números que usaremos para el mensaje, siendo estos los valores que estará comunicando al final, al igual que crearemos un valor que estará recibiendo los últimos valores de i (time) y w (signal) para nuestra señal, como también creamos un valor que será nuestro valor de desfase.
    - El tiempo del periodo le daremos un valor de 0.05, que establece que el valor de 5 Hz para la función timer_cb
    - Creamos dos funcione time:callback y signal_callback, que recibe el mensaje de sus respectivos nombres y guarda el último valor de ambos
    - En la función timer_cb, se realizan el proceso que modifica nuestra señal final *np.sin(self.last_i - self.phase_shift)* hará que haya un retroceso en nuestra señal, restando el valor original con nuestra variable
    - Dentro de la misma función también modificamos la amplitud de la onda, primero sumando 1 para estar siempre en positivo y luego dividiendo la onda a la mitad
    - Asignamos el valor del mensaje que en Float32, asignando data el valor actual de proc_w que representa el valor de la onda , posteriormente se publicará el mensaje para el canal de comunicación /proc_signal_argos
4. Launcher
    - Crearemos un launch que pueda ejecutar ambos nodos y su distribución de prioridad se establecerá dentro de un .py
    - En el .py de nombre signal_processing_argos, establecerá 2 dos nodos signal_generator y Signal_prece en ese respectivo orden,
    - Agregamos un código para poder abrir una comparativa de los datos de /signal_argos y /proc_signal_argos

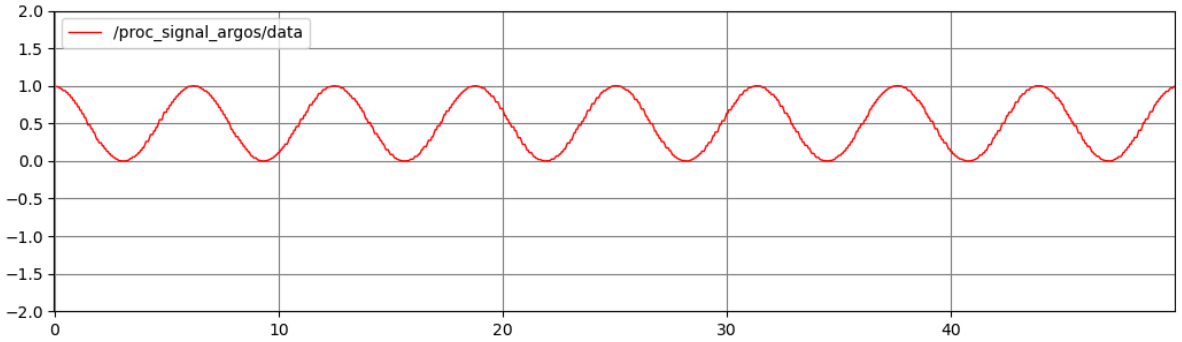
Resultados



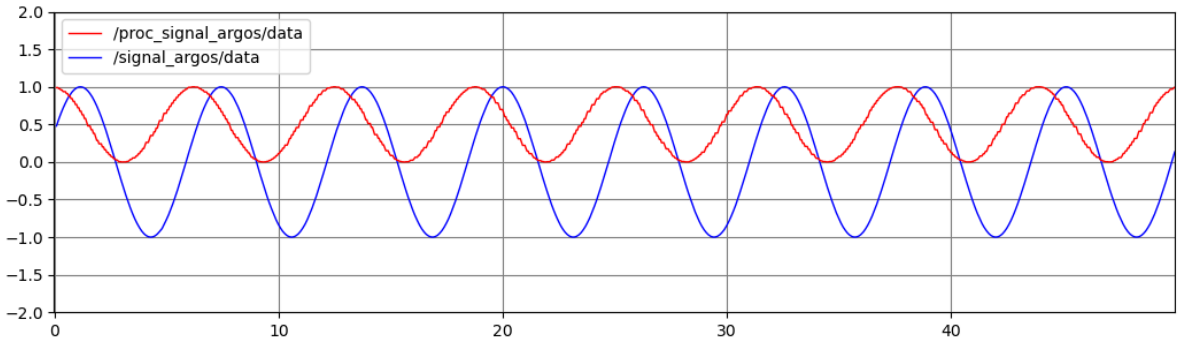
Al tener 2 nodos se muestran con los nombres/signal_gen_node que genera /time_argos y /signal_argos que los lleva al siguiente nodo /signal_proce que al final libera un valor de /proc_signal_argos.



El valor de /signal_argos es el mensaje que libera el primer nodo que es nuestra onda senoidal original, esta onda va en 10 Hz.



El valor de /proc_signal_argos es el mensaje que libera el segundo nodo que es nuestra onda senoidal que es procesada por los cambios de amplitud y retroceso, como podemos ver la onda tiene un cambio más quebrado en la línea que lo forma por estar en 5 Hz.



Ahora el launch forma nuestra gráfica comparativa que muestra los cambios de nuestra onda senoidal original con la onda procesada que se establece siempre en positivo y a la mitad de nuestra onda.

Conclusiones

En esta actividad se consiguió implementar un sistema distribuido básico utilizando ROS2, logrando cumplir los objetivos planteados anteriormente. Se reforzaron los conceptos fundamentales, tales como la estructura de comunicación basada en nodos, tópicos y mensajes.

El reto de crear dos nodos que se comunicaran de forma efectiva permitió comprender la importancia de la arquitectura distribuida en ROS2. El nodo signal_gen_node generó correctamente una señal senoidal, y el nodo signal_proce recibi, modificó y publicó la señal procesada con cambios en desfase y amplitud. La implementación del archivo launch facilitó la ejecución de ambos nodos y la visualización de resultados.

Una posible mejora sería implementar un control dinámico del tiempo de muestreo para ajustar la frecuencia de publicación en función de las necesidades del sistema. Otra mejora podría ser explorar el uso de parámetros dinámicos para modificar el desfase y la amplitud de la señal en tiempo de ejecución sin necesidad de modificar el código.

En conclusión, el desarrollo del reto permitió consolidar conocimientos en ROS2 y su aplicación en la comunicación entre nodos, reforzando la comprensión de los principios de un sistema distribuido en robótica.

Referencias

Open Robotics. (2023a). ROS 2 documentation: Architecture. ROS 2. https://docs.ros.org/en/rolling/Concepts.html

Open Robotics. (2023b). ROS launch files. ROS Wiki. http://wiki.ros.org/roslaunch

Open Robotics. (2023c). ROS tutorials: Writing a simple publisher and subscriber. ROS Wiki. http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber

Open Robotics. (2023d). ROS wiki: Conceptual overview. ROS Wiki. http://wiki.ros.org/ROS/Concepts

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: An open-source robot operating system. ICRA Workshop on Open Source Software. https://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf

Open Robotics. (n.d.). Understanding ROS 2 Services. ROS Documentation. Recuperado el [fecha de recuperación], de https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html