Reto semanal

Semana 1

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

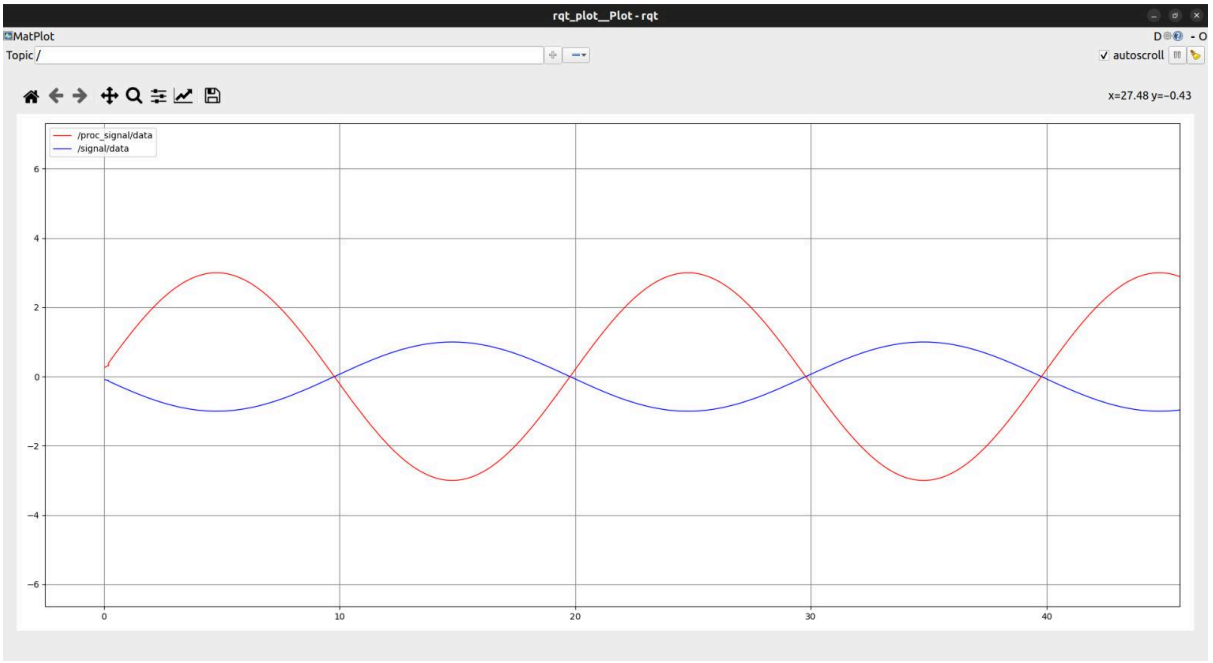
Alfredo Garcia Suarez

Juan Manuel Ahuactzin Larios

Febrero 2024

Resumen:

La actividad consiste en crear dos nodos, el primero actuará como un generador de señales, lo cual generará una señal senoidal. El segundo nodo actuará como un proceso, que tomará la señal generada por el nodo anterior y la modificará generando una nueva señal procesada. Ambas señales son trazadas utilizando la herramienta rqt_plot.



Objetivos:

Desarrollar las capacidades de manejo del framework ROS 2, familiarizarse con la metodología de trabajo de este así como reconocer la arquitectura básica de su funcionamiento. Este primer reto permitirá generar un primer acercamiento con los agentes que incorporan una red de trabajo en ROS, desde cómo organizar los proyectos en un espacio de trabajo, generar los primeros paquetes, hasta entender la metodología mediante la cual ROS permite la comunicación entre diferentes nodos.

Introducción:

ROS (Robot Operating System) es un conjunto de herramientas de código abierto y bibliotecas que ayudan a los desarrolladores de robots a crear software, ROS no es como exactamente su nombre lo dice, un sistema operativo de manera tradicional, es mejor dicho

un marco de trabajo flexible que proporciona herramientas para la robótica, como abstracción del hardware, el control de dispositivos de bajo nivel, la administración de paquetes y la comunicación entre procesos.

La estructura básica de funcionamiento de ROS está dividida en estos principales agentes que integran su red de comunicación, nodos, tópicos, mensajes, servicios y parámetros. Los nodos son procesos computacionales que realizan tareas específicas, estos se comunican entre sí mediante el intercambio de mensajes a través de tópicos. Los tópicos son canales de comunicación asincrónica utilizados para la transmisión de mensajes entre los nodos, los cuales pueden publicar mensajes en un tópico o suscribirse a un tópico para recibir mensajes. Los mensajes son estructuras de datos definidas en ROS que se intercambian entre nodos a través de los tópicos, los mensajes contienen información relevante para las operaciones que se requieren como datos de sensores, comandos de actuadores, etc. Los servicios proporcionan comunicaciones sincrónicas uno a uno entre nodos y permiten que un nodo solicite una operación a otro nodo y espere una respuesta. Los parámetros son valores utilizados para configurar el comportamiento de los nodos y permiten ajustar dichos parámetros durante la ejecución del sistema.

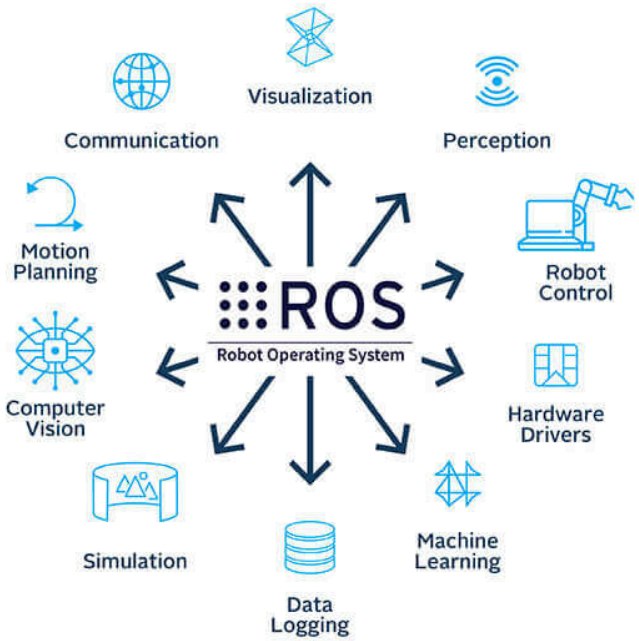


Figura 1: Se presenta diagrama de las herramientas a las que podemos usar dentro del ROS.

Teniendo esta visión general de lo que es ROS y su funcionamiento, para esta actividad realizaremos dos nodos, uno que genere una señal senoidal y otro que actuará como un proceso para tomar la señal generada y la modificará generando una señal procesada.

Solución del problema:

El código implementado cuenta con dos archivos principales donde se definen los nodos a utilizar y se les dan las funciones a realizar. El primer archivo es "signal_generator":

```
1  import rclpy
2  import numpy as np
3  from rclpy.node import Node
4  from std_msgs.msg import String
5  from std_msgs.msg import Float32
6
7  class Singnal_generator(Node):
8
9      def __init__(self):
10         super().__init__('signal_generator')
11         self.publisher = self.create_publisher(Float32, 'signal', 10)
12         timer_period = 0.01
13         self.timer = self.create_timer(timer_period, self.timer_callback2)
14         self.get_logger().info('signal node successfully initialized!!!')
15         self.msg = Float32()
16         self.i = 0
17
18     def timer_callback2(self):
19         self.msg.data = np.sin(np.pi*self.i)
20         self.publisher.publish(self.msg)
21         self.i = self.i + 0.001
22
23
24 def main(args=None):
25     rclpy.init(args=args)
26     m_p = Singnal_generator()
27     rclpy.spin(m_p)
28     m_p.destroy_node()
29     rclpy.shutdown()
30
31
32 if __name__ == 'main':
33     main()
```

Este código primero crea un publicador al tema "signal" de tipo float (línea 11). En su callback, el dato que mandará será una onda senoidal (líneas 19 a 21).

El segundo archivo se llama "process":

```python
import rclpy
import numpy as np
from std_msgs.msg import Float32
from rclpy.node import Node
from std_msgs.msg import String

class Process(Node):
    def __init__(self):
        super().__init__('process')
        self.sub = self.create_subscription(Float32, 'signal', self.listener_callback, 10)
        self.get_logger().info('process node initialized!!!')
        self.publisher = self.create_publisher(Float32, 'proc_signal', 10)

    def listener_callback(self, msg):
        self.msg2 = Float32()
        self.msg1 = msg.data
        self.msg2.data = self.msg1 * -3
        self.publisher.publish(self.msg2)

def main(args=None):
    rclpy.init(args=args)
    m_s = Process()
    rclpy.spin(m_s)
    m_s.destroy_node()
    rclpy.shutdown()

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

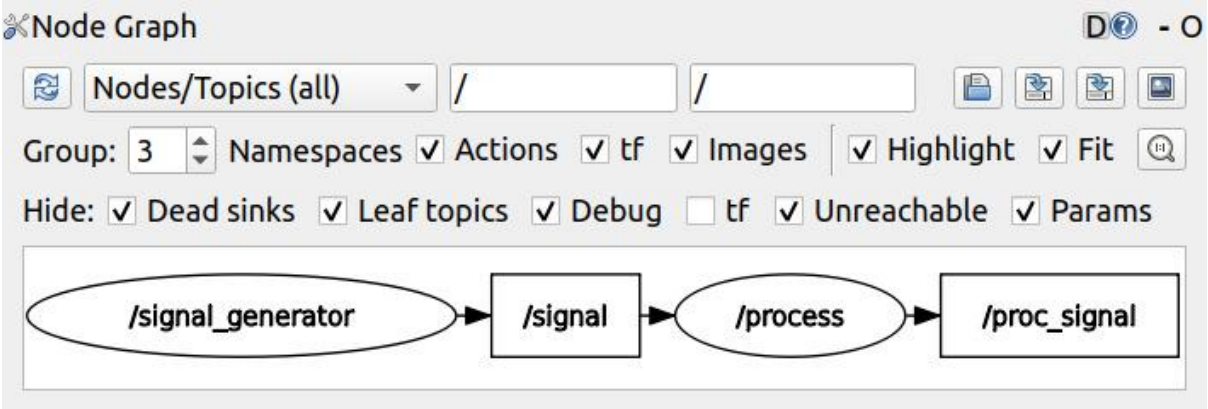
Este programa lee a través de una suscripción a “signal” el dato que el publicador está mandando, y posteriormente mediante un nuevo publicador vuelve a mandar la señal leída, pero transformada a un nuevo canal “proc_signal”.

En su función callback se lee define el nuevo dato a mandar como un float; también se lee el dato del publicador anterior y posteriormente se procesa multiplicándolo por menos tres (líneas 15 a 18).

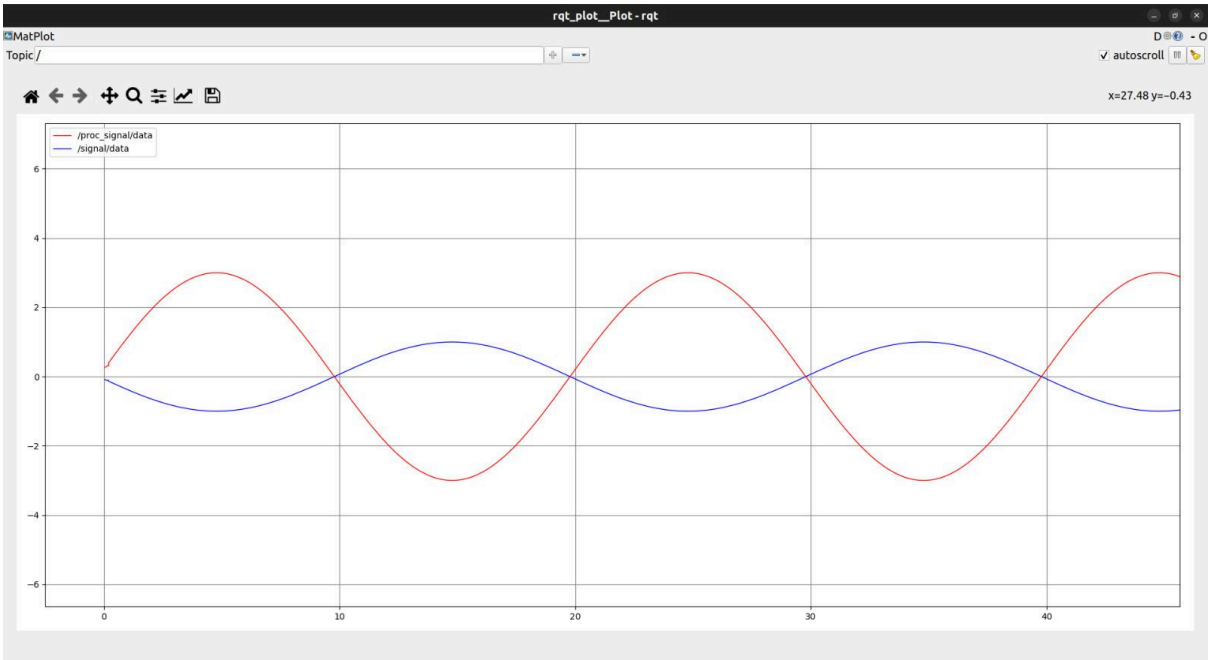
Resultados:

Con la herramienta rqt_graph podemos analizar nuestros procesos en general, podemos observar que como primer nodo tenemos al generador de la señal senoidal y a través del tópico signal se comunica con el segundo nodo que procesa la señal para mostrarnos la señal

procesada en un nuevo tópico llamado proc_signal.



Aquí se observan las dos señales, en color azul tenemos la señal senoidal original y en color rojo se nos muestra la señal ya procesada.



Conclusión:

Gracias a este mini reto logramos familiarizarnos con el manejo de ROS 2 en cuanto a su metodología de trabajo y su arquitectura básica, al momento del desarrollo de este trabajo aprendimos más sobre la comunicación entre nodos a través de los tópicos con los que ROS 2 trabaja. Uno de los problemas o preguntas que nos surgieron en el proceso fue que al momento de probar nuestro código en diferentes computadores veíamos que se captan ciertos datos que generaban ruido en cada una de nuestras máquinas en las que se estaba trabajando, ahora sabemos que la razón de esto es que al trabajar con ROS 2 y tener nuestras máquinas

conectadas a la misma red, estas se comunican entre sí por lo que si alguno mandaba datos que el otro no tenía, igual los recibimos, en este reto fue algo que nos causó problemas pero en un ambiente en el que se requiera analizar datos de diferentes medios, esto es algo sumamente útil.

Bibliografia:

Installation — ROS 2 Documentation: Humble documentation. (s. f.). https://docs.ros.org/en/humble/Installation.html

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

ROS/concepts. (s. f.). https://wiki.ros.org/ROS/Concepts