Practica 0

Rocio Fabiola Romero Bernal

1. Introducción

La Práctica 0 introduce los conceptos fundamentales de ROS 2 necesarios para estructurar y comunicar un sistema robótico, en particular los nodos, los tópicos y las interfaces que permiten el intercambio de información entre procesos distribuidos. Esta práctica busca que el estudiante comprenda cómo se organiza un proyecto en ROS 2 mediante paquetes y cómo estos elementos se relacionan para simular el control de un actuador, como el motor de un robot móvil.

1.1. Contexto de ROS 2

En ROS 2, un nodo es una unidad de procesamiento que realiza una tarea específica, mientras que los tópicos constituyen canales de comunicación donde los nodos publican y se suscriben a mensajes de forma asíncrona. Esta arquitectura distribuida permite dividir el sistema en componentes modulares que pueden desarrollarse, probarse y reutilizarse de manera independiente, aspecto que resulta esencial para la ingeniería mecatrónica aplicada a robots reales. Dentro de esta arquitectura, un nodo publicador es aquel que genera datos y los envía periódicamente a un tópico, sin necesidad de conocer qué otros nodos los recibirán. Por su parte, un nodo suscriptor se conecta a un tópico para escuchar los mensajes que se publican en él y ejecutar una función cada vez que llega nueva información. En el contexto de manipuladores robóticos, como los ejemplos del SCARA y el Dofbot, los nodos de control actúan como publicadores de trayectorias articulares, mientras que los controladores de bajo nivel del robot se comportan como suscriptores que interpretan esas trayectorias y las transforman en movimiento real. Este esquema desacoplado facilita el diseño modular del software, permitiendo que diferentes partes del sistema se desarrollen, prueben y sustituyan de forma independiente.

1.2. Paquetes e interfaces

La práctica también exige revisar cómo se documenta y organiza la creación de paquetes en ROS 2, considerando la estructura típica de directorios y archivos como `package.xml`, `CMakeLists.txt` o `setup.py`, que describen dependencias, nodos y scripts ejecutables. Comprender este "árbol" de archivos permite, integrar correctamente nuevos nodos y facilitar el mantenimiento del código, siguiendo las recomendaciones de documentación de paquetes.

1.3. Tipos de interfaces en ROS 2

Finalmente, se estudian las tres interfaces básicas de ROS 2: mensajes, servicios y acciones, empleadas respectivamente para flujos de datos continuos, solicitudes síncronas y tareas de mayor duración con retroalimentación. El conocimiento de estas interfaces prepara al alumno para diseñar sistemas de comunicación más complejos, en los que un mismo robot puede transmitir datos de sensores por tópicos, atender peticiones mediante servicios y ejecutar movimientos de largo plazo usando acciones.

2. Desarrollo de la práctica

El desarrollo de la práctica, se centra en la creación de un entorno de trabajo en ROS 2 donde se configure un paquete y se programe los nodos necesarios para establecer comunicación mediante tópicos. A partir de los ejemplos mostrados en los videos, primero se genera un paquete dentro del *workspace* y se añaden los archivos fuente en Python, definiendo un nodo publicador que envía mensajes relacionados con la velocidad de un motor y un nodo suscriptor encargado de recibirlos e interpretarlos.

1. En el primer video se muestra cómo preparar el entorno de trabajo de ROS 2 creando un *workspace* y dentro de él la carpeta `src`, donde se almacenarán los paquetes de la práctica. A partir de ahí se genera un paquete nuevo con el comando `ros2 pkg create`, lo que produce la estructura básica de archivos (`CMakeLists.txt`, `package.xml` y directorios de código) sobre la cual se implementarán los nodos relacionados con el control de velocidad del motor.

2. El siguiente paso del desarrollo consiste en escribir el código del nodo publicador, definiendo un tipo de mensaje apropiado y una tasa de publicación que envíe de manera periódica comandos de velocidad al tópico seleccionado. En el mismo paquete se implementa el nodo suscriptor, que se suscribe al mismo tópico y procesa los valores recibidos, asociándolos con una variable que representa la velocidad del motor o el estado del sistema.

3. Una vez creados ambos nodos, se procede a compilar el paquete desde la raíz del *workspace* utilizando `colcon build` y posteriormente se actualiza el entorno con el comando `source install/setup.bash` para poder ejecutar los programas. Después, en terminales separadas, se lanzan el publicador y el suscriptor, verificando que los mensajes se transmiten correctamente mediante herramientas como `ros2 topic list`, `ros2 topic echo` y `ros2 node list`.

4. Finalmente, los videos relacionan estos nodos con la simulación del comportamiento de un motor, mostrando cómo los valores publicados en el tópico pueden usarse para modificar la velocidad de un actuador virtual en un entorno de simulación o en un ejemplo gráfico. En esta etapa el estudiante observa la respuesta del sistema ante distintos comandos de velocidad y reflexiona sobre cómo la estructura de nodos y tópicos podría escalarse hacia un control más completo mediante servicios, acciones u otros componentes de ROS 2.

En resumen, se realizo un nodo publicador denominado velocidad pub node, que publica velocidades que van cambiando en rpm; un nodo publicador subscriptor llamado velocidad pub sub node, ademas de un nodo subscritor llamado plot velocidad que publica en graficas la velocidad contra el tiempo, para esto ultimo se tubo que generar un entorno virtual

Listing 1: Código del nodo velocidad_pub_node

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Int64

class VelocidadPub(Node):
    def __init__(self):
```

```
        super().__init__("velocidad_pub_node")
        self.number_ = 1
        self.number_publisher_ = self.create_publisher(
            Int64, "velocidad_topic_pub", 10)
        self.timer_ = self.create_timer(1.0, self.publish_number)
        self.get_logger().info(
            "El nodo velocidad_pub_node esta activo")

    def publish_number(self):
        msg = Int64()
        msg.data = self.number_
        self.number_publisher_.publish(msg)
        self.number_ += 4

def main(args=None):
    rclpy.init(args=args)
    node = VelocidadPub()
    rclpy.spin(node)
    rclpy.shutdown()

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

Listing 2: Código del nodo velocidad_sub_pub_node

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Int64, Float64
import time

class VelocidadPubSub(Node):

    def __init__(self):
        super().__init__("velocidad_pub_sub_node")

        self.rad_pub_ = self.create_publisher(Float64, "
           velocidad_rad_topic",10)
        self.number_subscriber_ = self.create_subscription(Int64, "
           velocidad_topic_pub", self.callback_number, 10)
        self.t0 = time.time()
        self.get_logger().info("El nodo velocidad_pub_sub_node esta
            activo")



    def callback_number(self,msg):
        rpm= msg.data
        rad_s=rpm*(2*3.141592)/60
        t_now = time.time() - self.t0
        out_msg = Float64()
        out_msg.data = rad_s
        self.rad_pub_.publish(out_msg)
```

```
        self.get_logger().info(f"t = {t_now:.2f} s, Velocidad recibida: {rpm} rpm, convertida:{rad_s:.2f} rad/s")
def main(args=None):
    rclpy.init(args=args)
    node = VelocidadPubSub()
    rclpy.spin(node)
    rclpy.shutdown()

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

Listing 3: Nodo plot_velocidad

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Int64
import matplotlib.pyplot as plt
import time


class PlotVelocidad(Node):

    def __init__(self):
        super().__init__('plot_velocidad')
        # t pico de rpm
        self.sub_rpm = self.create_subscription(
            Int64,
            '/velocidad_topic_pub',
            self.callback_rpm,
            10
        )
        # t pico de rad/s
        self.sub_rad = self.create_subscription(
            Int64,
            '/velocidad_rad_topic',
            self.callback_rad,
            10
        )

        self.get_logger().info('El nodo plot_velocidad est  activo')

        self.t_list = []
        self.rpm_list = []
        self.rad_list = []

        self.t0 = time.time()
        self.last_rpm = 0.0

    def callback_rpm(self, msg):
        self.last_rpm = float(msg.data)

```

```
39      def callback_rad(self, msg):
40          rad_s = float(msg.data)
41          t_now = time.time() - self.t0
42
43          self.t_list.append(t_now)
44          self.rpm_list.append(self.last_rpm)
45          self.rad_list.append(rad_s)
46
47          self.get_logger().info(
48              f't={t_now:.2f} s, Velocidad recibida: {self.last_rpm
                    :.2f} rpm, '
49              f'convertida: {rad_s:.2f} rad/s'
50          )
51
52
53  def main(args=None):
54      rclpy.init(args=args)
55      node = PlotVelocidad()
56
57      try:
58          rclpy.spin(node)
59      except KeyboardInterrupt:
60          pass
61
62      # Gr fica rad/s vs tiempo
63      plt.figure()
64      plt.plot(node.t_list, node.rad_list)
65      plt.xlabel('Tiempo [s]')
66      plt.ylabel('rad/s')
67      plt.title('Velocidad en rad/s vs tiempo')
68      plt.grid(True)
69      plt.xlim(0, 30)
70      plt.ylim(0, 1000)
71
72      # Gr fica rpm vs tiempo
73      plt.figure()
74      plt.plot(node.t_list, node.rpm_list)
75      plt.xlabel('Tiempo [s]')
76      plt.ylabel('rpm')
77      plt.title('Velocidad en rpm vs tiempo')
78      plt.grid(True)
79      plt.xlim(0, 30)
80      plt.ylim(0, 1000)
81
82      plt.show()
83
84      node.destroy_node()
85      rclpy.shutdown()
86
87
88  if __name__ == '__main__':
```

```
89      main()
```

Lamentablemente no logre hacer que graficaran solo consegui las graficas

Conclusión

En esta práctica se logró comprender la estructura fundamental de un sistema robótico basado en ROS 2, comenzando por la definición y funcionamiento de los nodos y tópicos como unidades básicas de procesamiento y canales de comunicación. A partir de estos conceptos fue posible implementar nodos en `rclpy` capaces de suscribirse a señales de velocidad, procesar la información recibida y registrarla de forma ordenada en listas de tiempo y velocidad.

En conjunto, la práctica sentó las bases para futuras actividades donde se requerirá un control más avanzado del robot y una interpretación rigurosa de las variables de estado.

Referencias

[1] Open Robotics, *ROS 2 Documentation*. Disponible en: https://docs.ros.org

[2] E. Peña Medina, *Nodos Publicadores y Suscriptores C++*. Video en YouTube, Facultad de Ingeniería, UNAM, 2025. Disponible en: https://youtu.be/PBtGcoQul6w

[3] Autor del canal, *Nodos en ROS 2 programados en Python*. Video en YouTube. Disponible en: https://youtu.be/5j4I0yB1YqI