Resposta da Questão 2 – Pergunta 2:

Alexandre Nuernberg

Diante do que foi exposto anteriormente, crie uma imagem Docker para a utilização dos pacotes ROS2 versão Humble. A imagem deve incluir as dependências necessárias para construir pacotes ROS, suas bibliotecas de comunicação, pacotes de mensagens, e ferramentas de linha de comando. O arquivo Dockerfile deve ser entregue juntamente com instruções claras para construir a imagem Docker e executar containers a partir dela.

Crie pacotes ROS2 (C++ ou Python3) que forneçam as seguintes funcionalidades:

2. Pacote "2" deve simular a leitura de um sensor com uma taxa de amostragem de 1 Hz. Os dados do sensor devem passar por um filtro de média móvel considerando os últimos 5 valores adquiridos pelo sensor. Esse pacote deve prover duas interfaces de serviço, a primeira deve retornar os últimos 64 resultados gerados pelo filtro, e a segunda deve zerar os dados gerados pelo filtro;

Segue o procedimento para criar o simulador do sensor com uma taxa de amostragem de 1Hz.

Passo 1: Criar o Pacote ROS2 para Simulação do Sensor com Filtro de Média Móvel

1. Criar o pacote ROS2:
   No terminal do container Docker:

```
   cd ~/ros2_ws/src
   ros2 pkg create sensor_sim --build-type ament_python --dependencies rclpy std_msgs
example_interfaces
```

2. Criar o script do nó:
   No diretório do pacote, crie o arquivo sensor_node.py:

```
   cd ~/ros2_ws/src/sensor_sim/sensor_sim
   touch sensor_node.py
```

   Edite sensor_node.py com o seguinte conteúdo:

```
# Arquivo sensor_node.py comentado:

import rclpy                      # Importa a biblioteca ROS2 para Python
from rclpy.node import Node       # Importa a classe Node para criar nós ROS2
from std_msgs.msg import Float32  # Importa a mensagem Float32 para publicar dados do sensor
from std_srvs.srv import Trigger  # Importa a mensagem Trigger para o serviço de reset
from std_msgs.msg import Float32MultiArray  # Importa a mensagem Float32MultiArray para o serviço de
dados

import random                     # Importa a biblioteca para gerar números aleatórios

class SensorNode(Node):           # Define a classe SensorNode que herda de Node
    def __init__(self):           # Construtor da classe
        super().__init__('sensor_node')  # Inicializa o nó com o nome 'sensor_node'
        self.publisher_ = self.create_publisher(Float32, 'sensor_data', 10)  # Cria um publisher para
o tópico 'sensor_data'
        self.timer = self.create_timer(1.0, self.publish_sensor_data)  # Cria um timer para publicar
dados a cada 1 segundo
        self.data = []            # Lista para armazenar os dados brutos do sensor
        self.filtered_data = []   # Lista para armazenar os dados filtrados

        # Cria os serviços
        self.get_data_service = self.create_service(Trigger, 'get_filtered_data',
self.get_filtered_data)           # Serviço para obter os dados filtrados
```

```
        self.reset_data_service = self.create_service(Trigger, 'reset_filtered_data',
self.reset_filtered_data)         # Serviço para resetar os dados

    def publish_sensor_data(self):  # Função para publicar os dados do sensor
        # Simula a leitura do sensor
        sensor_value = random.uniform(0.0, 10.0)  # Gera um valor aleatório entre 0 e 10
        self.data.append(sensor_value)  # Adiciona o valor à lista de dados brutos

        # Aplica o filtro de média móvel
        if len(self.data) > 5:    # Verifica se há pelo menos 5 valores na lista
            self.data.pop(0)      # Remove o valor mais antigo da lista
        moving_average = sum(self.data) / len(self.data)  # Calcula a média móvel dos últimos 5
valores
        self.filtered_data.append(moving_average)  # Adiciona a média móvel à lista de dados
filtrados
        if len(self.filtered_data) > 64:  # Limita o tamanho da lista de dados filtrados a 64 valores
            self.filtered_data.pop(0)  # Remove o valor mais antigo da lista

        # Publica os dados filtrados
        msg = Float32()  # Cria uma mensagem Float32
        msg.data = moving_average # Atribui a média móvel à mensagem
        self.publisher_.publish(msg)  # Publica a mensagem no tópico 'sensor_data'
        self.get_logger().info(f'Published filtered sensor data: {moving_average}')  # Exibe uma
mensagem no console

    def get_filtered_data(self, request, response):  # Função para lidar com o serviço de obter dados
filtrados
        # Retorna os dados filtrados
        msg = Float32MultiArray() # Cria uma mensagem Float32MultiArray
        msg.data = self.filtered_data  # Atribui a lista de dados filtrados à mensagem
        return response  # Retorna a mensagem com os dados filtrados

    def reset_filtered_data(self, request, response):  # Função para lidar com o serviço de resetar
dados
        # Reseta os dados filtrados
        self.filtered_data.clear()  # Limpa a lista de dados filtrados
        response.success = True   # Define o status da resposta como sucesso
        response.message = "Filtered data has been reset."  # Define a mensagem da resposta
        return response           # Retorna a resposta

def main(args=None):              # Função principal do nó
    rclpy.init(args=args)         # Inicializa a biblioteca ROS2
    sensor_node = SensorNode()    # Cria uma instância da classe SensorNode
    rclpy.spin(sensor_node)       # Executa o nó até que seja interrompido
    sensor_node.destroy_node()    # Destrói o nó
    rclpy.shutdown()              # Finaliza a biblioteca ROS2

if __name__ == '__main__':        # Verifica se o script está sendo executado como principal
    main()  # Executa a função main
```

3. Modificar o arquivo setup.py para incluir o novo script no campo entry_points:

```
   entry_points={
        'console_scripts': [
            'sensor_node = sensor_sim.sensor_node:main',
        ],
   },
```

4. Compilar o pacote:
   No diretório do workspace, compile com colcon:

```
  cd ~/ros2_ws
   colcon build
```

5. Faça o source do workspace:

```
  source install/setup.bash
```

6. Executar o nó:

```
   ros2 run sensor_sim sensor_node
```

Para testar o nó do simulador de sensor com a publicação dos dados do sensor:

```
rosuser@95974631fb73:~/ros2_ws$ pwd
/home/rosuser/ros2_ws
rosuser@95974631fb73:~/ros2_ws$ source install/setup.bash
rosuser@95974631fb73:~/ros2_ws$ ros2 run sensor_sim sensor_node
[INFO] [1730644014.236062361] [sensor_node]: Published filtered sensor data: 1.9457367975042617
```

```
[INFO] [1730644015.218182240] [sensor_node]: Published filtered sensor data: 3.0278630776986355
[INFO] [1730644016.218151504] [sensor_node]: Published filtered sensor data: 3.1017283107528004
[INFO] [1730644017.217980901] [sensor_node]: Published filtered sensor data: 4.821496080933019
```

Obs: A parte da questão: “Esse pacote deve prover duas interfaces de serviço, a primeira deve retornar os últimos 64 resultados gerados pelo filtro, e a segunda deve zerar os dados gerados pelo filtro” não está operacional.