Alexandre Nuernberg
Resposta da Questão 2: (COM GIT)

Atenção!
O código de resolução encontra-se disponível em:
https://github.com/alexandreberg/Prova_SENAI_02302_2024_Questao2

ou com:

```
git clone https://github.com/alexandreberg/Prova_SENAI_02302_2024_Questao2.git
```

O desenvolvimento de sistemas robóticos vem se tornando cada vez mais abordado em soluções industriais, comerciais e de serviços. Essa busca é acompanhada pela complexidade dos processos e problemas encontrados.

No âmbito de sistemas embarcados, a aplicação de ROS (Robot Operating System), agrega funcionalidades, bibliotecas e ferramentas para construção e programação de sistemas robóticos [2]. Para auxiliar no desenvolvimento de soluções com o ROS, muitos desenvolvedores de software embarcado aproveitam a versatilidade da plataforma Docker, projetada para facilitar compilações, execuções e compartilhamento de aplicações [3].

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:

1. Pacote "1" deve publicar mensagens a cada 1 (um) segundo com informações sobre a quantidade total de memória, o uso de memória RAM em Gigabyte e o percentual do uso;

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;

3. Pacote "3" deve encontrar, via requisição de ação, o décimo número primo, gerando respostas intermediárias pela interface de ação e também o resultado final.

Referências

[1] Git. Distributed even if your workflow isnt. Disponível em: <https://git-scm.com/>.
[2] ROS. Robot Operating System. Disponível em: <https://www.ros.org/>.
[3] Docker. Accelerate how you build, share and run applications. Disponível em: <https://www.docker.com/>.
[4] Molcut, A. I.; Lica, S.; Lie, Ion. Cybersecurity for Embedded Systems: A review. 2022 International Symposium on Electronics and Telecommunication. DOI: 10.1109/ISETC56213.2022.10009944.
[5] Krishnan, P.; Suman, V. Effectiveness of Random Testing of Embedded Systems. 2012 45th Hawaii International Conference on System Sciences. DOI 10.1109/HICSS.2012.233.

Boa Prova!

Passo-a-passo para instalar o Docker no Ubuntu Linux
Rodar esse passo na máquina host que hospedará os containers Docker

Segue a resposta da parte: 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.

Atenção!

Antes de rodar a imagem do ros2, é necessário que o Docker esteja instalado na máquina host. Seguem os passos de instalação.
Obs: comandos abaixo devem ser executados num terminal com shell por exemplo (*bash*) pode ser usado o *xterm*.
Toda a instalação foi efetuada num sistema: Ubuntu 22.04.

Passo 1: Atualizar o sistema

```
sudo apt update
```

Atualiza a lista de pacotes disponíveis nos repositórios do Ubuntu.

Passo 2: Instalar pacotes necessários

```
sudo apt install apt-transport-https ca-certificates curl gnupg lsb-release
```

Instala os pacotes necessários para adicionar o repositório do Docker e baixar sua chave GPG.

Passo 3: Adicionar a chave GPG do Docker

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
```

Baixa a chave GPG do repositório do Docker e a adiciona ao seu sistema.

Passo 4: Adicionar o repositório do Docker

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

Adiciona o repositório oficial do Docker à lista de repositórios do seu sistema.

Passo 5: Atualizar a lista de pacotes

```
sudo apt update
```

Atualiza a lista de pacotes para incluir os pacotes do repositório do Docker.

Passo 6: Instalar o Docker Engine

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Instala o Docker Engine, a CLI do Docker, o containerd (runtime do container) e o plugin do Docker Compose.

Passo 7: Adicionar seu usuário ao grupo docker

```
sudo usermod -aG docker $USER
```
Adiciona seu usuário ao grupo "docker", permitindo que se execute comandos do Docker sem sudo. Será necessáriofazer logout e login novamente para que essa alteração tenha efeito.

Passo 8: Iniciar o Docker

```
sudo systemctl start docker
```
Inicia o serviço do Docker.

Passo 9: Verificar a instalação

```
sudo docker run hello-world
```
Baixa e executa uma imagem de teste do Docker, verificando se a instalação foi bem-sucedida.

O Docker agora está instalado no seu sistema Ubuntu. Você pode começar a usar o dockercomando para baixar imagens, executar containers e gerenciar suas aplicações.

Passo-a-passo para criar a imagem Docker à partir de um Dockerfile

1. Salve o Dockerfile:
Crie um arquivo chamado *Dockerfile* com o conteúdo fornecido.

```
alerta@china:~$ cd
alerta@china:~$ mkdir docker
alerta@china:~$ cd docker
alerta@china:~/docker$ vi Dockerfile
```

Dockerfile com comentários:
Dockerfile

```
# Define a imagem base como Ubuntu 22.04
FROM ubuntu:22.04

# Configura o frontend do apt para não interativo (modo silencioso)
ENV DEBIAN_FRONTEND=noninteractive
# Define o fuso horário para São Paulo
ENV TZ=America/Sao_Paulo

# Instala pacotes necessários
RUN apt update && \
    apt install -y --no-install-recommends \
    software-properties-common \  # Permite adicionar repositórios PPA
    build-essential \            # Ferramentas essenciais para compilar software
    cmake \                      # Sistema de build para o ROS
    git \                        # Sistema de controle de versão
    python3-pip \                # Gerenciador de pacotes Python
    wget \                       # Utilitário para download de arquivos da web
    curl \                       # Utilitário para download de arquivos da web
    gnupg2 \                     # Ferramenta para verificação de assinaturas digitais
    lsb-release \                # Utilitário para obter informações sobre a
distribuição Linux
    locales \                    # Suporte a localização
    sudo \                       # Utilitário para executar comandos como superusuário
    tzdata && \                  # Dados de fuso horário
    rm -rf /var/lib/apt/lists/*  # Limpa o cache do apt para reduzir o tamanho da imagem

# Configura o locale para inglês americano
RUN locale-gen en_US en_US.UTF-8 && \
    update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8

# Adiciona o repositório ROS 2
RUN curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
/usr/share/keyrings/ros-archive-keyring.gpg && \
    echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-
archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(lsb_release -cs) main" | tee
/etc/apt/sources.list.d/ros2.list > /dev/null

# Instala o ROS 2 Humble versão Desktop
RUN apt update && \
    apt install -y --no-install-recommends ros-humble-desktop && \
    rm -rf /var/lib/apt/lists/*

# Configura o ambiente ROS 2
ENV ROS_DISTRO humble
RUN echo "source /opt/ros/$ROS_DISTRO/setup.bash" >> /root/.bashrc

# Cria um diretório para o workspace ROS 2
RUN mkdir -p /ws_ros2/src
WORKDIR /ws_ros2

# Comando para iniciar o container (bash)
CMD ["/bin/bash"]
```

O arquivo Dockerfile é necessário para a criação da imagem. Ele contém as instruções para o Docker sobre como construir a imagem, incluindo a instalação do sistema operacional, pacotes, dependências e configurações.
Este Dockerfile instala o pacote ros-humble-desktop, que inclui ferramentas como RViz e Gazebo.

2. Construa a imagem:
Abra o terminal na pasta onde você salvou o Dockerfile e execute o seguinte comando:

```
docker build -t ros2-humble .
```

Constroi a imagem Docker com base nas instruções do Dockerfile.

- -t ros2-humble: Define o nome da imagem como "ros2-humble".
- .: Indica que o Dockerfile está no diretório atual.

3. Execute um container a partir da imagem:
Após a imagem ser criada, você pode executar um container com o seguinte comando:

```
docker run -it ros2-humble
```

Inicia um container interativo (-it) com base na imagem "ros2-humble".

Rodando o container

Verifique os containers que estão disponíveis e rodando

```
alerta@china:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED         STATUS          PORTS     NAMES
95974631fb73   ros2-humble    "/bin/bash"   10 hours ago    Up 2 seconds              
ros2_02
alerta@china:~$
```

Caso não esteja, inicie o container do ros2:

```
alerta@china:~$ docker start ros2_02
```

Logar no container e criar um usuário *rosuser, ajusta a senha e adiciona ao grupo sudo*:

```
alerta@china:~$  docker exec -it ros2_02 bash
root@95974631fb73:/ws_ros2#
root@95974631fb73:/ws_ros2# useradd -m -d /home/rosuser -s /bin/bash rosuser
root@95974631fb73:/ws_ros2# passwd rosuser
root@95974631fb73:/ws_ros2# usermod -aG sudo rosuser
root@95974631fb73:/ws_ros2# su - rosuser
```

Resposta da Questão 2 - Pergunta 1: Publicar mensagens:

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

1. Pacote "1" deve publicar mensagens a cada 1 (um) segundo com informações sobre a quantidade total de memória, o uso de memória RAM em Gigabyte e o percentual do uso;

Obs: Para os seguintes passos, está se supondo que o container com o ROS2 já está ativo e você já está logado nele conforme descrito nos procedimentos anteriores.
Para criar um pacote ROS2 que publique informações de memória em um container Docker, siga os passos abaixo:
Pré-requisitos:
Instalar o *colcon*.
Para instalar o *colcon*, execute o seguinte comando no seu container:

```
sudo apt update
sudo apt install python3-colcon-common-extensions -y
```

1. Verifique o ambiente: Confirme que você está no ambiente correto do ROS2. No terminal do container, faça o *source* da shell:

```
source /opt/ros/humble/setup.bash
```

Crie o workspace:
Crie a estrutura do workspace ROS2:

```
mkdir -p /home/rosuser/ros2_ws/src
```

2. Crie o pacote: Crie um novo pacote
ROS2 chamado "**mem_info**", que será responsável por publicar as informações de memória.
No diretório de trabalho do seu workspace recém criado (/ros2_ws/src), execute:

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

3. Edite o nó de publicação:
- Navegue até a pasta do pacote e crie o arquivo Python que publicará as informações de memória.

```
cd mem_info/mem_info
touch memory_publisher.py
```

- Edite *memory_publisher.py* com o seguinte conteúdo:

```
import rclpy  # Importa a biblioteca principal do ROS2
from rclpy.node import Node  # Importa a classe Node para criar nós ROS2
from std_msgs.msg import String  # Importa o tipo de mensagem String para publicar os dados
import psutil  # Importa a biblioteca psutil para obter informações sobre a memória
import time  # Importa a biblioteca time para debugging

class MemoryPublisher(Node):  # Define a classe MemoryPublisher que herda da classe Node
    def __init__(self):
        super().__init__('memory_publisher')  # Inicializa o nó com o nome 'memory_publisher'
        self.publisher_ = self.create_publisher(String, 'memory_info', 10)  # Cria um publisher que
publica mensagens do tipo String no tópico 'memory_info' com um tamanho de buffer de 10
        self.timer = self.create_timer(1.0, self.publish_memory_info)  # Cria um timer que chama a
função publish_memory_info a cada 1 segundo

    def publish_memory_info(self):  # Define a função que publica as informações de memória
        mem = psutil.virtual_memory()  # Obtém as informações da memória virtual usando a biblioteca
psutil
        total_memory_gb = mem.total / (1024 ** 3)  # Calcula a memória total em GB
        used_memory_gb = mem.used / (1024 ** 3)  # Calcula a memória usada em GB
        memory_percent = mem.percent  # Obtém o percentual de uso da memória

        msg = String()  # Cria uma mensagem do tipo String
        msg.data = f'Total Memory: {total_memory_gb:.2f} GB, Used Memory: {used_memory_gb:.2f} GB,
Usage: {memory_percent}%'  # Formata a mensagem com as informações de memória
        self.publisher_.publish(msg)  # Publica a mensagem no tópico 'memory_info'
        self.get_logger().info(f'Publishing: "{msg.data}"')  # Exibe a mensagem publicada no console

def main(args=None):  # Função principal do script
```

```
    rclpy.init(args=args)  # Inicializa o ROS2
    memory_publisher = MemoryPublisher()  # Cria uma instância da classe MemoryPublisher
    rclpy.spin(memory_publisher)  # Executa o nó até que seja interrompido
    memory_publisher.destroy_node()  # Destrói o nó após a interrupção
    rclpy.shutdown()  # Desliga o ROS2

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

4. Instale dependências: Instale a biblioteca *psutil*, que é necessária para obter os dados de memória, executando:

```
pip install psutil
```

5. Atualize o arquivo *setup.py*:

- No diretório do pacote *mem_info*, abra o arquivo *setup.py* e adicione o novo script ao campo *entry_points*:

```
    entry_points={
        'console_scripts': [
            'memory_publisher = mem_info.memory_publisher:main',
        ],
    },
```

O arquivo *setup.py* na íntegra ficou assim:

```
rosuser@95974631fb73:~/ros2_ws/src/mem_info$ pwd
/home/rosuser/ros2_ws/src/mem_info
rosuser@95974631fb73:~/ros2_ws/src/mem_info$ more setup.py
#Arquivo setup.py comentado

from setuptools import find_packages, setup  # Importa as funções find_packages e setup do setuptools

package_name = 'mem_info'                    # Define o nome do pacote como 'mem_info'

setup(  # Chama a função setup para configurar o pacote
    name=package_name,  # Define o nome do pacote
    version='0.0.0',  # Define a versão do pacote
    packages=find_packages(exclude=['test']),  # Encontra todos os pacotes Python no diretório atual,
exceto o diretório 'test'
    data_files=[  # Define os arquivos de dados a serem incluídos no pacote
        ('share/ament_index/resource_index/packages',  # Diretório onde o arquivo de índice de
recursos será instalado
            ['resource/' + package_name]),  # Arquivo de índice de recursos para o pacote
        ('share/' + package_name, ['package.xml']),  # Diretório onde o arquivo package.xml será
instalado
    ],
    install_requires=['setuptools'],  # Define os pacotes necessários para instalar este pacote
    zip_safe=True,  # Indica se o pacote pode ser instalado a partir de um arquivo zip
    maintainer='rosuser',  # Define o nome do mantenedor do pacote
    maintainer_email='rosuser@todo.todo',  # Define o email do mantenedor do pacote
    description='TODO: Package description',  # Define a descrição do pacote (ainda a ser preenchida)
    license='TODO: License declaration',  # Define a licença do pacote (ainda a ser preenchida)
    tests_require=['pytest'],  # Define os pacotes necessários para executar os testes do pacote
    entry_points={  # Define os pontos de entrada do pacote
        'console_scripts': [  # Define os scripts de console que podem ser executados
            'memory_publisher = mem_info.memory_publisher:main',  # Define o script
'memory_publisher' que chama a função main do módulo memory_publisher
        ],
    },
)

rosuser@95974631fb73:~/ros2_ws/src/mem_info$
```

6. Compile o pacote: No diretório do workspace, compile o pacote com:

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

7. Execute o source do workspace:

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

8. Execute o nó: para publicar as informações de memória:
O nó deve publicar as informações sobre a memória a cada segundo.
Obs: Para parar o nó que está rodando, use *Ctrl+C* no terminal onde o nó está em execução. Isso enviará um sinal de interrupção (SIGINT) para o processo, encerrando-o.

Log do Memory Publisher em execução:

```
rosuser@95974631fb73:~/ros2_ws/src/mem_info/mem_info$ cd ~/ros2_ws
rosuser@95974631fb73:~/ros2_ws$ source install/setup.bash
rosuser@95974631fb73:~/ros2_ws$ ros2 run mem_info memory_publisher
[INFO] [1730599819.033372280] [memory_publisher]: Publishing: "Total Memory: 7.67 GB, Used Memory: 5.57 GB, Usage: 79.1%"
[INFO] [1730599820.013866939] [memory_publisher]: Publishing: "Total Memory: 7.67 GB, Used Memory: 5.57 GB, Usage: 79.1%"
[INFO] [1730599821.013863384] [memory_publisher]: Publishing: "Total Memory: 7.67 GB, Used Memory: 5.57 GB, Usage: 79.1%"
[INFO] [1730599822.014015446] [memory_publisher]: Publishing: "Total Memory: 7.67 GB, Used Memory: 5.57 GB, Usage: 79.1%"
[INFO] [1730599823.013195038] [memory_publisher]: Publishing: "Total Memory: 7.67 GB, Used Memory: 5.57 GB, Usage: 79.1%"
```

Resposta da Questão 2 - Pergunta 2: sensor_node

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.

Obs a Pergunta 3 não foi respondida por falta de tempo.