

ROS2: A IRMANDADE DO CÓDIGO

UMA JORNADA ÉPICA PELA ROBÓTICA

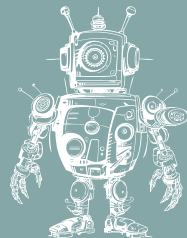


MATHEUS SILVA

SUMARIO



Introdução	03
Capítulo I: O Despertar dos Autômatos	04
Capítulo II: Forjando a Aliança dos Nós	06
Capítulo III: A Dança das Máquinas	07
Capítulo IV: Explorando Terras Desconhecidas com SLAM	09
Conclusão	11



INTRODUÇÃO

Bem-vindo a "ROS 2: A Irmandade do Código - Uma Jornada Épica pela Robótica". Este e-book é uma aventura guiada pelo mundo do ROS 2 (Robot Operating System 2), uma das ferramentas mais poderosas e versáteis para o desenvolvimento de sistemas robóticos modernos. Vamos explorar, de forma prática e direta, como utilizar ROS 2 para criar, gerenciar e integrar componentes robóticos, transformando ideias em realidade.

Nossa jornada será dividida em quatro capítulos principais, cada um focado em aspectos fundamentais do ROS 2, acompanhados de exemplos de código reais. Vamos começar com uma introdução ao ROS 2, aprender a criar e comunicar entre nós, integrar sensores e atuadores, e finalmente, mergulhar nas técnicas de SLAM (Simultaneous Localization and Mapping) para navegação em ambientes desconhecidos.

Prepare-se para uma aventura emocionante, onde cada linha de código aproxima você do domínio da robótica com ROS 2.

CAPITULO I



O DESPERTAR DOS AUTÔMATOS

O que é ROS 2?

ROS 2 (Robot Operating System 2) é um conjunto de bibliotecas e ferramentas que ajudam a construir sistemas robóticos. Ele facilita a criação de software modular, onde diferentes partes de um robô podem ser desenvolvidas e testadas independentemente.

Instalando ROS 2

Para começar, precisamos instalar o ROS 2. No Ubuntu, podemos usar os seguintes comandos:

Não se esqueça de configurar seu ambiente:

```
Bash
sudo apt update
sudo apt install ros-foxy-desktop
```

```
Bash
source /opt/ros/foxy/setup.bash
```

Capítulo I



Primeiro Nó: Olá Mundo

Vamos criar um nó simples que publica uma mensagem "Olá, Mundo!".

```
Python

import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class OlamundoPublisher(Node):
    def __init__(self):
        super().__init__('olamundo_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        self.timer = self.create_timer(0.5, self.timer_callback)

    def timer_callback(self):
        msg = String()
        msg.data = 'Olá, Mundo!'
        self.publisher_.publish(msg)
        self.get_logger().info(f'Publicando: {msg.data}')

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

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

Um nó é essencialmente um programa que realiza uma função específica dentro do sistema robótico. Este exemplo simples demonstra como iniciar com ROS 2 criando um nó que envia mensagens. É o primeiro passo para entender como os componentes de um sistema robótico se comunicam e trabalham juntos.

CAPITULO II



FORJANDO A ALIANÇA DOS NÓS

Comunicando-se entre Nós

Para comunicação, ROS 2 utiliza tópicos. Vamos criar um nó assinante que recebe a mensagem "Olá, Mundo!".

```
Python

# olamundo_sub.py
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class OlamundoSubscriber(Node):
    def __init__(self):
        super().__init__('olamundo_subscriber')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            10)
        self.subscription

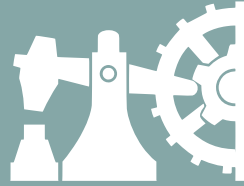
    def listener_callback(self, msg):
        self.get_logger().info(f'Recebido: {msg.data}')

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

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

Este código cria um nó publicador simples em ROS 2, chamado OlamundoPublisher, que envia a mensagem "Olá, Mundo!" a cada 0.5 segundos para um tópico chamado 'topic'.

CAPITULO III



A DANÇA DAS MÁQUINAS

Trabalhando com Sensores

Vamos integrar um sensor de distância simples (simulado) ao nosso sistema ROS 2.

```
Python

# sensor_pub.py
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32
import random

class SensorPublisher(Node):
    def __init__(self):
        super().__init__('sensor_publisher')
        self.publisher_ = self.create_publisher(Float32, 'sensor_data', 10)
        self.timer = self.create_timer(1.0, self.timer_callback)

    def timer_callback(self):
        distance = random.uniform(0.0, 10.0) # Simulando um sensor de distância
        msg = Float32()
        msg.data = distance
        self.publisher_.publish(msg)
        self.get_logger().info(f'Distância: {msg.data:.2f} m')

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

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

O código cria um nó chamado `SensorPublisher` que simula um sensor de distância, publicando valores aleatórios entre 0.0 e 10.0 metros a cada segundo no tópico `'sensor_data'`. Esse nó ajuda a entender como integrar sensores ao ROS 2 e enviar dados sensoriais para outros nós no sistema.

CAPÍTULO III



Atuadores em Ação

Agora, vamos criar um nó que controla um atuador (simulado) com base nos dados do sensor.

```
Python

# atuador_sub.py
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32

class AtuadorSubscriber(Node):
    def __init__(self):
        super().__init__('atuador_subscriber')
        self.subscription = self.create_subscription(
            Float32,
            'sensor_data',
            self.listener_callback,
            10)
        self.subscription

    def listener_callback(self, msg):
        if msg.data < 2.0:
            self.get_logger().info('Alerta: Objeto muito próximo! Ativando atuador.')

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

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

O código cria um nó chamado `AtuadorSubscriber` que recebe dados de distância do tópico `'sensor_data'`. Se a distância recebida for menor que 2.0 metros, ele registra um alerta indicando que um objeto está muito próximo, simulando a ativação de um atuador em resposta aos dados do sensor.

CAPITULO IV

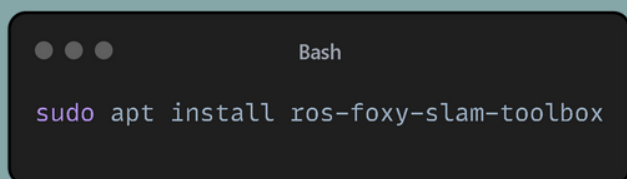
Explorando Terras Desconhecidas com “SLAM”

Introdução ao SLAM

SLAM (Simultaneous Localization and Mapping) permite que robôs mapeiem ambientes desconhecidos enquanto se localizam dentro desses mapas. Vamos usar a biblioteca `slam_toolbox` do ROS 2.

Configurando o SLAM Toolbox

Primeiro, instale o `slam_toolbox`:

A terminal window with a dark background and rounded corners. At the top, there are three small gray circles on the left and the word "Bash" on the right. The command `sudo apt install ros-foxy-slam-toolbox` is entered in a light blue monospace font.

```
Bash
sudo apt install ros-foxy-slam-toolbox
```

Navegando com SLAM

Após mapear, podemos usar o mapa para navegação. Para isso, podemos integrar outros pacotes como `nav2`, configurando um sistema de navegação completo.

CAPITULO IV



Executando SLAM

Vamos configurar um lançamento básico para o SLAM. Crie um arquivo de lançamento em Python para configurar o slam_toolbox.

```
Python

# slam_toolbox.launch.py
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='slam_toolbox',
            executable='sync_slam_toolbox_node',
            name='slam_toolbox',
            output='screen',
            parameters=[
                {'use_sim_time': False}
            ],
            remappings=[
                ('/slam_toolbox/scan', '/scan')
            ]
        )
    ])
]
```

Para executar o lançamento, utilize o seguinte comando:

```
Bash

ros2 launch slam_toolbox slam_toolbox_launch.py
```

CONCLUSÃO



Com esses exemplos, você deu os primeiros passos para dominar ROS 2 e suas aplicações na robótica. Iniciamos com a criação de um nó publicador simples, "Olá, Mundo!", que nos introduziu aos conceitos básicos de nós e tópicos. Em seguida, exploramos a comunicação entre nós, mostrando como enviar e receber mensagens.

A integração de sensores e atuadores nos mostrou como obter dados do ambiente e reagir a eles, essencial para qualquer sistema robótico. Finalmente, entramos no mundo do SLAM (Simultaneous Localization and Mapping), aprendendo a mapear e navegar em ambientes desconhecidos, uma habilidade crucial para robôs móveis.

A jornada com ROS 2 é contínua e cheia de oportunidades para aprender e inovar. Continue experimentando, criando novos nós e integrando diferentes componentes. A prática constante e a curiosidade são seus maiores aliados nessa aventura épica pela robótica. Boa sorte e divirta-se explorando o vasto universo de ROS 2!