Simulando um ambiente robótico integrado no ROS2

Aluna: Tainara Rodrigues Teixeira
Turma: Engenharia da Computação
Professor: Rodrigo Nicola

Orientações

Para a construção da realização da atividade será necessário seguir os seguintes critérios de avaliação:

Espera-se a capacidade demonstrável de interagir com o sistema operacional de robôs, criando um nó de comunicação com uma solução pré-existente. A entrega deve ser um vídeo demonstrando o funcionamento do projeto, um texto conciso descrevendo como foi feita a implementação e o link para o repositório público no github onde foi feita a implementação. Tendo o seguinte padrão de qualidade:

Ao terminar esta atividade, espera-se a validação das suas instalações locais do ambiente de desenvolvimento que será utilizado ao longo do semestre. Com sua finalização, os estudantes deverão adicionar um link para um vídeo não listado no Youtube com a simulação funcionando.

a) Configuração adequada do ambiente de desenvolvimento ROS; (peso 2)
b) Funcionamento correto do script de interação com o turtlesim; (peso 3)
c) Explicação coerente e concisa da implementação; (peso 3)
d) Congruência entre o que foi escrito e o código disposto no repositório do github; (peso 2)

Detalhamento dos componentes da solução

Conforme foi compreendido, a solução terá os seguintes componentes utilizados:

Linux no Windows com o WSL2
Com o uso do WSL2 (Subsistema do Windows para Linux) é possível instalar várias distribuições como a que vamos utilizar, o Ubuntu. Assim, usuários que não utilizam o Linux como sistema operacional padrão podem executar aplicações e ferramentas de linha de comando, scripts ou ambientes de desenvolvimento específicos do Linux.

Segue o link de como seguir com mais informações sobre o manual de instalação do WSL2: https://learn.microsoft.com/pt-br/windows/wsl/install

Ubuntu
Desenvolvido pela empresa Canonical Ltd, e sendo um dos sistemas operacionais de código aberto e mais utilizado no mundo todo, o Ubuntu será essencial para a solução, pois, usaremos uma das ferramentas que ele oferece para a simulação em 2D de robôs, o TurtleSim.

Para a instalação do Ubuntu no WSL2, confira o manual a seguir:
https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-11-with-gui-support#3-download-ubuntu

ROS2
O ROS2 (Robot Operating System 2) que oferece um conjunto de ferramentas e bibliotecas e nos permitirá integrar todo o sistema operacional para termos um ambiente robótico integrado e fundamentalmente composto pelo o Ubuntu e o TurtleSim.

Para a instalação do framework o manual a seguir evidencia como instalar o ROS2 no Ubuntu Linux, lembre-se de colocar cada linha de comando individualmente por vez:
https://docs.ros.org/en/dashing/Installation/Ubuntu-Install-Binary.html

TurtleSim

O TurtleSim é uma ferramenta de simulação que faz parte do ROS2 já instalado anteriormente, ele permite a criação de robôs virtuais em um ambiente gráfico, nele é possível definir o deslocamento necessário do robô.

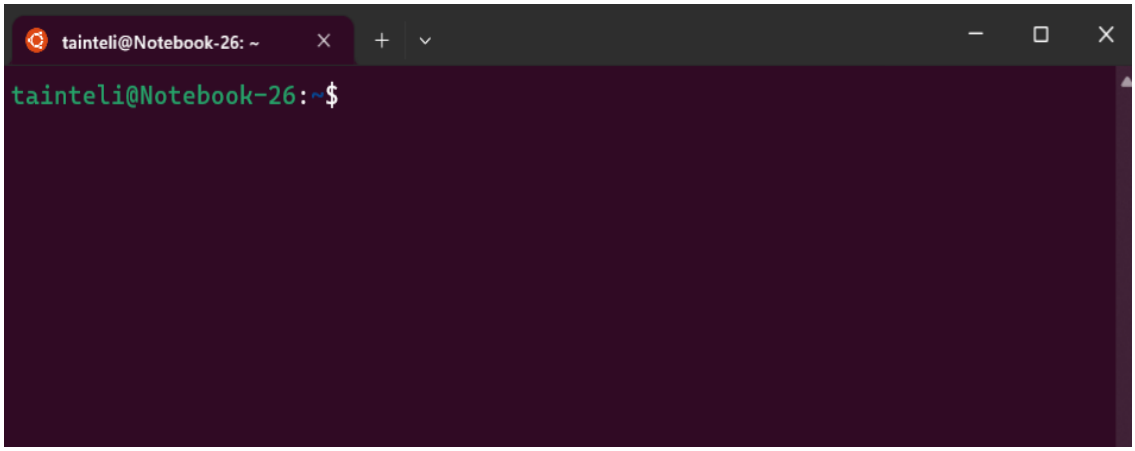
Código em python

O cliente desenvolvido em python3 define e publica as requisições, ou seja, ele é responsável por enviar as requisições para TurtleSim (Servidor) que vai ouvi-las e mover a tartaruga com os dados recebidos do cliente.

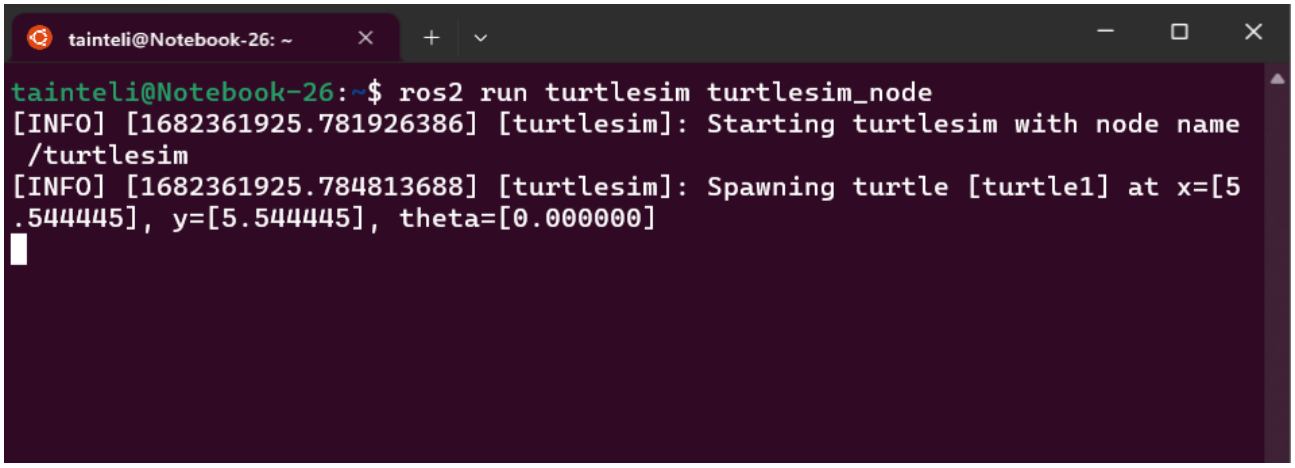
Passo a passo da integração

Após todas as configurações e instalações feitas de acordo com os manuais citados, será necessário seguir os próximo passos:

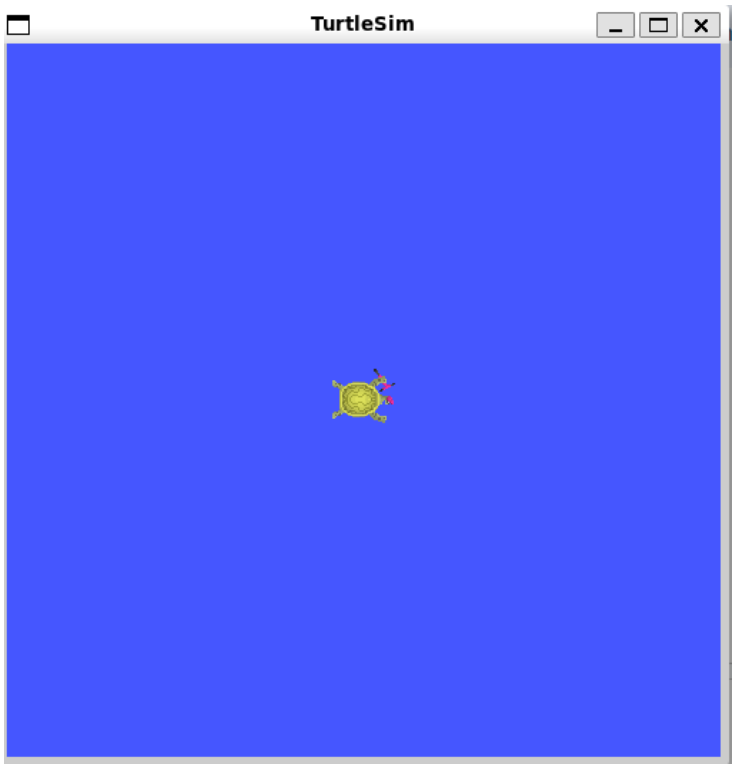
1. **Abrir um terminal Ubuntu:**



2. **Iniciar o TurtleSim (Servidor) com o seguinte comando:**

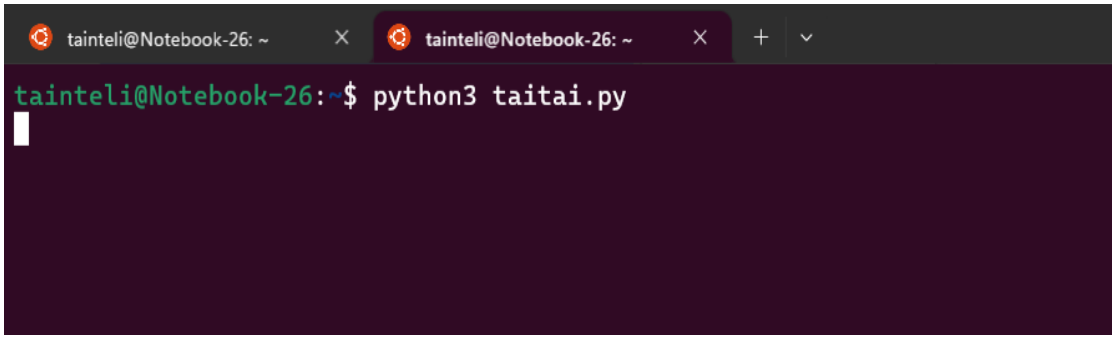


Essa linha de comando é responsável por iniciar e integrar o TurtleSim com o terminal Ubuntu, sendo o TurtleSim que começará a publicar e receber mensagens do ROS2, que por sua vez, atuam como canais de comunicação. O “turtlesim_node” é responsável pela criação da simulação gráfica do robô virtual no ambiente bidimensional. Perceba que após o comando ser excutado, a janela gráfica do TurtleSim é aberta, como podemos observar na seguinte imagem:

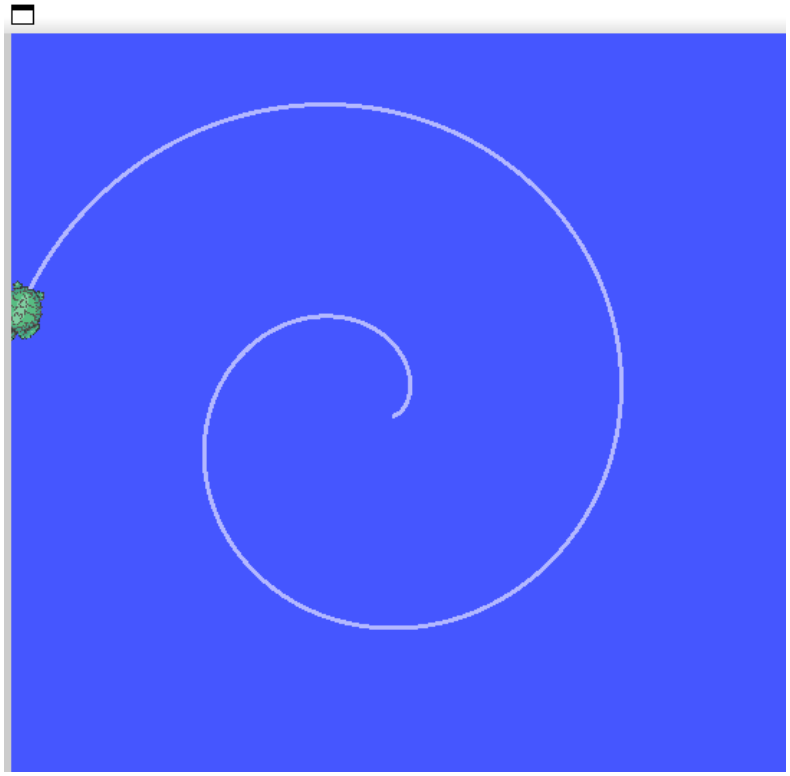


Como o previsto, ainda não estamos com o código integrado, assim a tartaruga não recebe nenhuma informação para se locomover e permanece estática.

3. **Abrir um segundo terminal para integrar com o código:**



Ao colocar “python3” adicione o nome do arquivo para ser integrado com o TurtleSim. Após executar, deixe o primeiro terminal Ubuntu que iniciamos anteriormente e visualize como a simulação gráfica irá alterar, a tartaruga receberá as novas informações e fará o caminho solicitado. Neste caso, ela mostrará uma espiral.



Processo de criação do script

Com base no exemplo dado durante a instrução, foi possível implementar o primeiro teste com o código abaixo:

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist

class TurtleController(Node):
def __init__(self):
super().__init__('turtle_controller')
self.publisher_ = self.create_publisher(Twist, 'turtle1/cmd_vel', 10)
self.timer_ = self.create_timer(0.1, self.move_turtle)
self.twist_msg_ = Twist()

def move_turtle(self):
self.twist_msg_.linear.x = 1.0
self.twist_msg_.angular.z = 0.5
self.publisher_.publish(self.twist_msg_)

def main(args=None):
rclpy.init(args=args)
turtle_controller = TurtleController()1
rclpy.spin(turtle_controller)
turtle_controller.destroy_node()
rclpy.shutdown()

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

Sendo um ótimo exemplo de nó que irá controlar a tartaruga para o ROS2. Ele começa importando a biblioteca “rclpy” e a classe Node do pacote “rclpy.node”. Logo, importa a mensagem Twist do pacote “geometry_msgs” para executar sua principal função de controle sobre o deslocamento do robô.

A classe TurtleController após ser definida como um nó, também herda a classe Node. A função “inti()” do construtor é responsável por configurar o nó, estabelecer uma conexão com o publicador que envia mensagens Twist. Ademais, a função também executa um temporizador que executa a próxima função “move_turtle” que a cada um décimo de segundo cria um objeto Twist para ser atualizado pelo programa. Dados as medidas o círculo é formado.

Aliás, a função “move_turtle” é responsável além de ser chamada pelo temporizador e definir a velocidade linear e angular da tartaruga publica a mensagem Twist. Já a “main” inicializa o ROS2, executa o nó, aguardar as mensagens, e depois finaliza o nó para encerrar o ROS2.

Com base no exemplo, a lógica da espiral foi construída, utilizei as mesmas bibliotecas listadas acima, adicionando a “time” e a “math”, a primeira serve basicamente para calcular o tempo para entre a anterior e próxima posição do robô e a segunda é aquela que me permite fazer as operações matemáticas que irão calcular as posições da tartaruga. A lógica primordial foi utilizar o exemplo e aprimorar o código para que ao invés de um círculo, o trajeto continue sendo circular porém aumente gradativamente o seu raio conforme um determinado período e frequência de tempo, fazendo assim a tartaruga se movimentar em um padrão de espiral. Abaixo podemos conferir o cálculo que foi construído para chegar neste resultado de espiral, tendo em vista essa ideia de reciclar a lógica do círculo e acrescentar as equações paramétricas de uma espiral bidimensional, dada as seguintes expressões de x e y:

$$x = r * \cos(w * t) * t$$
$$y = r * \sin(w * t) * t$$

onde:

- r é o raio da espiral;
- w é a velocidade angular da tartaruga em radianos por segundo;

- t é o tempo decorrido desde o início da espiral;

Assim, calculando as coordenadas de x e y da tartaruga em cada iteração do loop. A seguir, é possível visualizar o código:

```
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
import time
import math

class TurtleController(Node):
  def __init__(self):
    super().__init__('turtle_controller')
    self.publisher_ = self.create_publisher(Twist, 'turtle1/cmd_vel', 10)
    self.timer_ = self.create_timer(0.1, self.move_turtle)
    self.twist_msg_ = Twist()

  def move_turtle(self):
    radius = 0.1  # valor do raio da espiral
    revolutions = 2  # vezes que irá fazer a espiral
    angular_velocity = 1.0  # valor da velocidade angular medida em
radianos/segundos da tartaruga
    frequency = 10  # frequência da publicação das mensagens
    period = 1.0 / frequency  # intervalo de segundo sobre cada mensagem
    time_increment = period / 2  # incremento de tempo com a finalidade de calcular
a próxima posição

    t = 0.0  # tempo inicial
    for i in range(int(revolutions * 2 * math.pi / time_increment)):
      # cálculo para a próxima posição da tartaruga com base no tempo inicial
      x = radius * math.cos(angular_velocity * t) * t
      y = radius * math.sin(angular_velocity * t) * t
```

```
        # cálculo da velocidade angular da tartaruga com base a posição e velocidade atual
        linear_velocity = math.sqrt(x**2 + y**2) / period  # valor da velocidade angular constante
        angular_velocity = angular_velocity  # velocidade angular constante

        # criar e publicar uma mensagem Twist para a posição e velocidade atual
        twist_msg = Twist()
        twist_msg.linear.x = linear_velocity
        twist_msg.angular.z = angular_velocity
        self.publisher_.publish(twist_msg)

        # aguarda a duração do período atual
        time.sleep(period)

        # atualiza o tempo que foi decorrido
        t += time_increment


def main(args=None):
  rclpy.init(args=args)
  turtle_controller = TurtleController()
  rclpy.spin(turtle_controller)
  turtle_controller.destroy_node()
  rclpy.shutdown()


if __name__ == '__main__':
```

Após o término de criação da lógica, o código ao ser conectado com o terminal Ubuntu e o TurtleSim como o indicado no item **Passo a passo da integração** e executará como previsto, o movimento em espiral da tartaruga.

Demonstração prática do ambiente robótico integrado no ROS2

Link do vídeo no postado como não listado no youtube:
https://youtu.be/Kho2fSXd0xo

Link do GitHub:
https://github.com/taiinteli/Simulando-Ambiente-Robotico