UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMPUS APUCARANA

Engenharia da Computação

Navegação Autônoma de Robôs na Plataforma ROS (Robot Operating System)

Código Múltiplos robôs autônomos com Action client e server

Autores:

Leandro Martins Tosta (Bolsista PIBIT 2023-2024)

Lucio Agostinho Rocha (Orientador)

Introdução

Implementação de múltiplos robôs autônomos utilizando a simulação de robôs no Turtlesim. Este ambiente nos possibilita criar, controlar e posicionar virtualmente várias tartarugas, permitindo a experimentação prática com conceitos de robótica autônoma. O objetivo final é explorar e aprimorar estratégias de coordenação e interação entre os robôs, contribuindo para o desenvolvimento de soluções mais complexas e robustas em um contexto de sistemas multi-robôs.

Requisitos

- Possuir um workspace ROS
- Configuração Node de cliente e servidor em Python

Metodologia

Passo 1: inserir arquivo em /ros2_ws/src/beswarm_actions

- beswarm_action_client.py

```
import sys
import rclpy
from rclpy.action import ActionClient
from rclpy.node import Node
from beswarm_action_interfaces.action import Beswarm

class BeswarmActionClient(Node):
    def __init__(self):
        super().__init__('beswarm_action_client')
        self._action_client = ActionClient(self, Beswarm, 'beswarm')

    def send_goal(self, client_id):
        goal_msg = Beswarm.Goal()
        goal_msg.client_id = client_id

        self._action_client.wait_for_server()

        self._send_goal_future =
self._action_client.send_goal_async(goal_msg,
feedback_callback=self.feedback_callback)

self._send_goal_future.add_done_callback(self.goal_response_callback)

    def goal_response_callback(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info('Goal rejected :(')
            return
```

```
        self.get_logger().info('Goal accepted :)')
        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)

    def get_result_callback(self, future):
        result = future.result().result
        self.get_logger().info('Result: {0}'.format(result.x))

    def feedback_callback(self, feedback_msg):
        feedback = feedback_msg.feedback
        self.get_logger().info('Received feedback:
{0}'.format(feedback.partial_sequence))

def main(argv):
    rclpy.init(args=None)

    if len(argv) < 1:
        print('beswarm_action_client <NUM_PARTICLES>')
        return

    i = 0

    NUM_PARTICLES = int(argv[i])
    action_client = BeswarmActionClient()
    action_client.send_goal(NUM_PARTICLES)
    rclpy.spin(action_client)

if __name__ == '__main__':
    main(sys.argv[1:])
```

Passo 2: inserir arquivo em /ros2_ws/src/beswarm_actions

- beswarm_action_server.py

```
import time
import rclpy
import time

from rclpy.action import ActionServer
from rclpy.node import Node
from beswarm_action_interfaces.action import Beswarm
from std_msgs.msg import String
from geometry_msgs.msg import Twist
from random import seed
from random import random
```

```
class BeswarmActionServer(Node):



    def __init__(self):

        super().__init__('beswarm_action_server')
        # seed random number generator
        seed(1)

        print('Action server waiting for connections...')
        self._action_server = ActionServer(
            self,
            Beswarm,
            'beswarm',
            self.execute_callback
        )

    def execute_callback(self, goal_handle):
        self.get_logger().info('Executing goal...')
        feedback_msg = Beswarm.Feedback()
        feedback_msg.partial_sequence = 0
        node = rclpy.create_node('minimal_publisher')

        for k in range(10):
            for i in range(1, goal_handle.request.client_id+1):
                value = '/turtle'
                value += str(i)
                value += '/cmd_vel'
                publisher = node.create_publisher(Twist, value, 10)

                # Position updates
                msg = Twist()
                i = 0

                msg.linear.x += random()
                msg.linear.y += random()
                msg.angular.z = 0.0

                i += 1

                node.get_logger().info('Publishing: %s"' % msg.linear.x)
                publisher.publish(msg)

                # ---
                feedback_msg.partial_sequence = msg.linear.x
                self.get_logger().info('Feedback:
{0}'.format(feedback_msg.partial_sequence))
                goal_handle.publish_feedback(feedback_msg)
```

```
            time.sleep(0.1)

        goal_handle.succeed()
        result = Beswarm.Result()
        result.x = feedback_msg.partial_sequence   # action interface
response field
        return result

def main(args=None):
    rclpy.init(args=args)
    beswarm_action_server = BeswarmActionServer()
    rclpy.spin(beswarm_action_server)

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

Passo 3: inserir arquivo em /ros2_ws/src/beswarm_actions

- multirobot_start.sh

```
#!/bin/bash

# Inicie o nó Turtlesim em segundo plano
ros2 run turtlesim turtlesim_node &

sleep 3

# Para limpar testes anteriores
ros2 service call /clear std_srvs/srv/Empty

# Aguarde um curto período para garantir que o nó Turtlesim esteja pronto
sleep 2

i=1

# Posicao inicial
x=1
y=1
z=0

# Retirar a marca de rastro da turtle1
ros2 service call /turtle1/set_pen turtlesim/srv/SetPen "{'r': 0, 'g': 0, 'b': 0,
'width': 0, 'off': 1}"

# Serviço para teletransportar a turtle1
ros2 service call /turtle1/teleport_absolute turtlesim/srv/TeleportAbsolute
"{'x': $x, 'y': $y, 'theta': 0.0}"

# Certificar que a proxima nao sobrepor
y=$((y + 1))
```

```
# Fazer o spawn de das outras tartarugas

while [ $i -le $1 ]; do
    echo "Creating Node turtle$i..."



    ros2 topic pub --once /turtle$i/cmd_vel geometry_msgs/msg/Twist
"{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"

    ros2 service call /turtle$i/set_pen turtlesim/srv/SetPen "{'r': 0, 'g': 0, 'b': 0,
'width': 0, 'off': 1}"
    a=$((i + 1))  # Use $((...)) para expressões aritméticas

    if [ $a -le $1 ]; then
        echo "Spawn turtle$a..."
        ros2 service call spawn turtlesim/srv/Spawn "{'x': $x, 'y': $y, 'theta': 0.0,
'name': 'turtle$a'}"
        #x=$((x + 1))
        y=$((y + 1))
    fi

    i=$((i + 1))
    sleep 1
done
```

Passo 4: inserir arquivo em /ros2_ws/src/beswarm_actions

- multirobat_finish.sh

```
#!/bin/bash
source /opt/ros/iron/setup.bash

echo 'Finishing /usr/bin/python3 processes...'
killall /usr/bin/python3
killall multirobot_start.sh

# Adicionando comando para fechar a janela do Turtlesim
pkill -f turtlesim_node
```

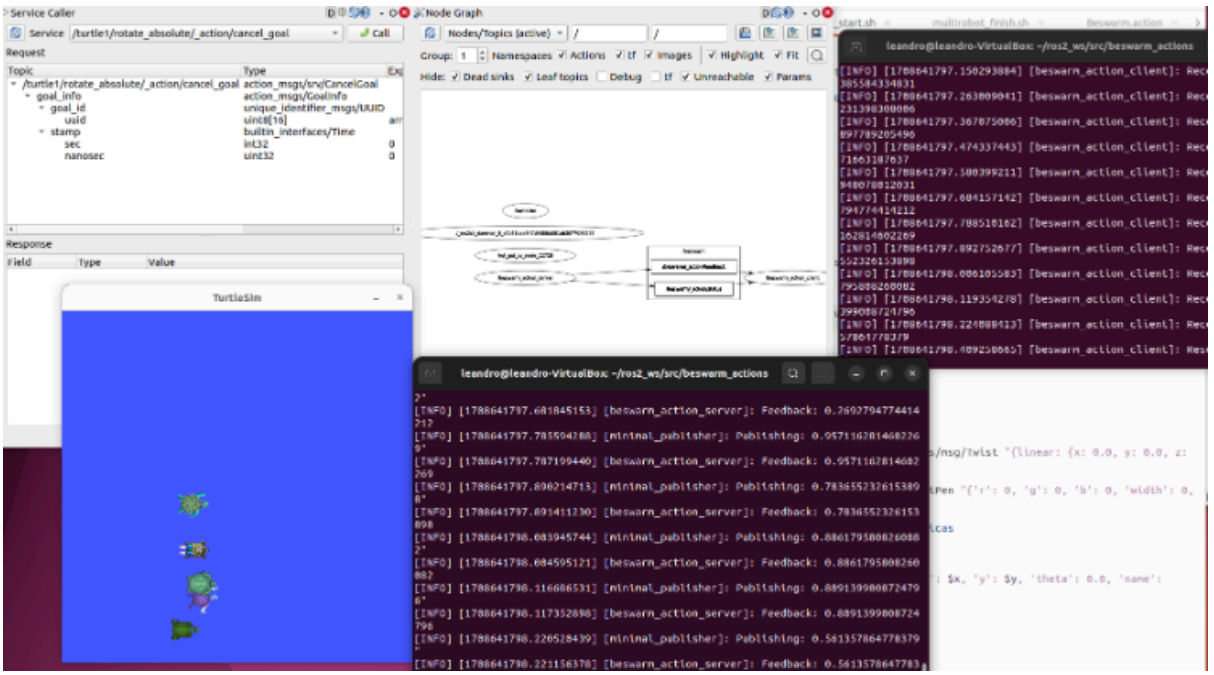

Figura 1: Múltiplos robôs autônomos com Action client e server. Fonte: Autoria Própria.

Materiais adicionais

- O middleware padrão usado pelo ROS 2 é o (RMW). Consulte o guia sobre como trabalhar com vários RMWs. Working with multiple ROS 2 middleware implementations.
- Tutoriais para desenvolver habilidades em ROS 2. Tutorials — ROS 2 Documentation: Iron documentation.