ROS2 Omni-Wheel Robot: Odometry Logging During Motion

This notebook demonstrates how to control an omni-wheel robot using ROS2 and collect real-time odometry data **while the robot is moving**.

We will:

1. Initialize ROS2 and the robot control node
2. Start the robot moving forward for 5 seconds
3. Log odometry data (position and angular velocity) during that movement
4. Display the collected data in a pandas DataFrame

In [ ]:
```
import rclpy
from omni_robot_controller import OmniWheelControlNode
from get_odom_variable_final import get_odom_variable
import time
import pandas as pd
import threading
```

Step 1: Initialize ROS2 and the Robot Node

We begin by initializing the ROS2 Python environment (`rclpy`) and creating an instance of our custom omni-wheel robot control node.

In [ ]:
```
rclpy.init()
node = OmniWheelControlNode()
```

Step 2: Start the Robot Moving in a Background Thread

Since the robot's movement function (`move_forward`) is blocking (i.e. it pauses code execution while running), we use a background thread to run the movement so we can **log odometry data simultaneously** in the main thread.

In [ ]:
```
def move_bot():
    node.move_forward(0.3, 5.0)  # Move forward at 0.3 RPS for 5 seconds
```

```
# Start the movement in a separate thread
threading.Thread(target=move_bot).start()
```

Step 3: Log Odometry Data While Moving

We now collect odometry data every second for 5 seconds **while the robot is moving**. We extract:

- `pose.pose.position.x` : the forward position of the robot
- `twist.twist.angular.z` : the angular velocity (rotation) around the z-axis

This shows how the robot's position and orientation change over time.

In [ ]:
```
log_data = []

for i in range(5):  # Collect one sample per second for 5 seconds
    pos_x = get_odom_variable('pose.pose.position.x')
    ang_z = get_odom_variable('twist.twist.angular.z')
    log_data.append({'time_s': i+1, 'pos_x_m': pos_x, 'ang_z_rad_s': ang_z})
    print(f"[{i+1}s] pos_x = {pos_x:.3f}, ang_z = {ang_z:.3f}")
    time.sleep(1.0)
```

Step 4: Display the Collected Data

Now we convert our odometry log into a pandas DataFrame for easier analysis and display.

In [ ]:
```
df = pd.DataFrame(log_data)
df
```

Step 5: Clean Shutdown

Finally, we shut down the ROS2 node and environment cleanly.

In [ ]:
```
node.destroy_node()
rclpy.shutdown()
```

Appendix: Alternate Method Using `time.time()` for Logging Control

This section shows a slightly more flexible version of odometry logging where we use `time.time()` to manage how long the robot moves and logs, instead of a fixed loop with `range()`.

This is useful when you want finer control over how long the bot runs or want to change sampling frequency easily.

In [ ]:
```python
# Start movement in a thread as before
threading.Thread(target=lambda: node.move_forward(0.3, 5.0)).start()

# Log odometry for 5 seconds using timestamps
start_time = time.time()
log_data2 = []

while time.time() - start_time < 5.0:
    pos_x = get_odom_variable('pose.pose.position.x')
    ang_z = get_odom_variable('twist.twist.angular.z')
    seconds_elapsed = int(time.time() - start_time)
    log_data2.append({'time_s': seconds_elapsed, 'pos_x_m': pos_x, 'ang_z_ra
    print(f"[{seconds_elapsed}s] pos_x = {pos_x:.3f}, ang_z = {ang_z:.3f}")
    time.sleep(1.0)
```

Appendix: Simple One-Time Odometry Variable Printing

This section demonstrates how to use `get_odom_variable()` to retrieve and print specific odometry values manually — useful for quick debugging or spot checks without logging or looping.

In [ ]:
```python
# Get and print individual odometry values
x = get_odom_variable('pose.pose.position.x')
y = get_odom_variable('pose.pose.position.y')
z = get_odom_variable('pose.pose.position.z')
yaw_rate = get_odom_variable('twist.twist.angular.z')

print(f"x = {x:.3f} m")
print(f"y = {y:.3f} m")
print(f"z = {z:.3f} m")
print(f"angular z (yaw rate) = {yaw_rate:.3f} rad/s")
```