Node Structure in Both Packages

This project contains two main Python nodes implemented in the `turtle_control` package:

1. figure8_driver.py

This node is responsible for publishing velocity commands to make the turtle trace a figure-8 pattern.

- Publishes to `/turtle1/cmd_vel` using geometry_msgs/Twist.
- Subscribes to `/turtle1/pose` using turtlesim/Pose and logs position (x, y, theta) every second.
- Uses a ROS 2 parameter `pattern_speed` to adjust the movement speed dynamically.
- Includes two timers: one for publishing movement commands (10 Hz), and another for logging pose (1 Hz).

2. trace_toggle.py

This node provides a ROS 2 service `/toggle_trace` using std_srvs/SetBool.

- When the service is called with `data: true`, it enables the turtle's pen via `/turtle1/set_pen`.
- When called with `data: false`, it disables the pen.
- It creates a client for the built-in `/turtle1/set_pen` service (turtlesim/srv/SetPen).
- The node waits for the service to be available before declaring readiness.

Challenges Encountered During Development

As a beginner to ROS 2 and Python nodes, I faced several challenges during this project:

- Forgot to source the workspace (`source install/setup.bash`) after building caused command errors.
- The launch file did not initially work due to missing `launch/` and `resource/` folders.
- Service call errors occurred because `/turtle1/set_pen` wasn't available immediately; problem occurred while looping.
- Incorrect parameter retrieval led to the turtle not moving; proper use of `declare_parameter()` and `get_parameter()` was needed.
- Naming mismatches between declared services and client calls caused runtime issues.