🧠 Objective

The objective of this project was to gain hands-on experience with ROS 2 concepts by building a package that demonstrates the use of:

- **Publishers** – to send movement commands
- **Subscribers** – to read the turtle's pose
- **Services** – to toggle the turtle's pen on and off
- **Timers** – to implement time-based control
- **Parameters** – to easily tune behavior
- **State Machines** – to control the turtle's motion

The goal behavior is to make the turtle move in a continuous **figure-eight** pattern, while allowing dynamic control over whether the turtle draws or not.

📂 Node Structure

➤ `Figure8Driver` (Main Node)

Component	Purpose
`Publisher`	`/turtle1/cmd_vel` – Sends `Twist` messages to control turtle movement.
`Subscriber`	`/turtle1/pose` – Receives `Pose` messages to track current position.
`Timer`	Controls the update rate of movement and logs pose periodically.
`Service Server`	`/toggle_trace` – A custom `std_srvs/srv/Empty` service to toggle the pen.
`Service Client`	`/turtle1/set_pen` – Sends a request to change the pen state (on/off).

`State Machine`	Governs the motion sequence: `turn_left` → `turn_right` → repeat.

🔄 Movement Logic (Figure-8 Pattern)

- The turtle alternates between turning left and right in circular arcs.
- The angular velocity is calculated to complete each loop approximately in `2π / angular_speed`.
- Once a full figure-eight is drawn, the pattern restarts, creating a **continuous loop**.

🖊️ Pen Toggle Feature

- The node includes a ROS 2 service `/toggle_trace` that switches the drawing pen on or off.
- When called, it sends a `SetPen` request with the `off` flag toggled.
- This allows visual control — you can pause drawing while keeping the turtle moving.

⚙️ Key Parameters

Parameter	Default	Description
`pattern_speed`	2.0	Linear forward speed of the turtle
`angular_speed_multiplier`	0.8	Multiplier to compute turning rate

These are declared as ROS 2 parameters, allowing easy tuning via command-line or launch files.

⚠️ Challenges Faced

1. **ROS 2 Installation & Setup:**
   Getting `colcon`, ROS 2 dependencies, and the workspace properly built required fixing broken packages and sourcing setup files correctly.

2. **Turtle Drawing Reset:**
   Initially, after finishing the figure-eight, the turtle would stop. We refactored the state machine to restart the pattern endlessly.

3. **Pen Toggle Logic:**
   Integrating the `SetPen` service required carefully coordinating service clients and handling the `off` logic dynamically.

4. **Synchronizing Timers:**
   We adjusted the loop rate and pose logging timers to balance smooth motion with meaningful logs.

✅ Outcomes

- The turtle continuously draws a figure-eight path on the screen.
- The movement is parameterized and reusable.
- A service-based interface allows dynamic control over drawing state.
- All components are built using proper ROS 2 architecture.
