This project implements a ROS 2 package named turtle_control, built using Python and the turtlesim simulator. It demonstrates essential ROS 2 concepts including publishers, subscribers, services, parameters, and launch files. The package includes two nodes: figure8_driver and trace_toggle, both launched together using a single launch file.

The figure8_driver node publishes velocity commands to the topic /turtle1/cmd_vel to make the turtle move in a figure-eight pattern. This is achieved by using a constant linear velocity and a sinusoidal angular velocity (angular.z = sin(t)), where t is derived from the ROS clock. The node also subscribes to /turtle1/pose to log the turtle's (x, y, θ) position at 1 Hz. A ROS 2 parameter pattern_speed controls the speed of the pattern, allowing it to be adjusted without modifying the code.

The second node, trace_toggle, creates a ROS 2 service on /toggle_trace using the standard std_srvs/srv/SetBool type. When called, this service toggles the turtle's pen by sending a request to the built-in /turtle1/set_pen service. If the data is true, the pen is turned on to draw; if false, the turtle moves without leaving a trail.

All nodes are launched together using the bringup.launch.py file, which starts turtlesim_node, figure8_driver, and trace_toggle. This allows for convenient testing and demonstration using one launch command.

Some challenges encountered included figuring out why the launch file wasn't being found (fixed by correcting setup.py to install the file) and ensuring that the pen toggle service was running before sending service calls. Additionally, using real-time (get_clock().now().nanoseconds) instead of hardcoded time steps was crucial to generate a smooth and continuous figure-eight pattern.

Overall, this project gave hands-on experience with core ROS 2 features. I really enjoyed doing this project. I learned new things and new way to solve problems.