ROS2 Tutorial Course Summary

This notebook summarizes the commands and concepts covered in the Robotics Back-End ROS2 tutorial series.

Episode 1: Intro: Install and Setup ROS2 Humble

This episode focuses on installing ROS2 Humble on Ubuntu 22.04.

Commands:

- `sudo apt update` : Updates the package lists for upgrades and new package installations.
- `sudo apt install --no-install-recommends ros-humble-desktop` : Installs the ROS 2 Humble desktop package without recommended packages.
- `sudo apt install ros-humble-desktop` : Standard command to install ROS 2 Humble with essential tools, demos, and tutorials.
- `sudo apt upgrade` : Upgrades all upgradable packages on the system. Recommended before installing ROS 2.
- `source /opt/ros/humble/setup.bash` : Sources the setup file for ROS 2 Humble installation to set up environment variables.
- `echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc` : Adds the ROS 2 setup command to your `.bashrc` file for automatic setup.
- `source ~/.bashrc` : Executes the `.bashrc` file, applying changes to the current terminal.
- `ros2 --version` : Checks the installed ROS 2 version and verifies if ROS 2 has been installed correctly.

Episode 2: Start Your First ROS2 Node

This episode introduces ROS 2 nodes and demonstrates running pre-built example nodes.

Commands:

- `ros2 run demo_nodes_cpp talker` : Runs the `talker` node that publishes "Hello World" messages.
- `ros2 run demo_nodes_cpp listener` : Runs the `listener` node that subscribes to the "Hello World" messages.
- `rqt_graph` : Provides a visual representation of the ROS 2 graph, showing running nodes and connections.
- `ros2 run turtlesim turtlesim_node` : Runs the `turtlesim_node` that simulates a turtle in a 2D environment.

- `ros2 run turtlesim turtle_teleop_key` : Runs the node that allows keyboard control of the turtle.

Episode 3: Create and Set Up a ROS2 Workspace

This episode explains creating and setting up a ROS 2 workspace to organize your own code.

Commands:

- `mkdir ros2_ws` : Creates a new directory for your ROS 2 workspace.
- `cd ros2_ws` : Changes directory to your workspace.
- `mkdir src` : Creates a source directory inside your workspace for ROS 2 packages.
- `sudo apt install python3-colcon-common-extensions` : Installs `colcon` , the build tool for ROS 2.
- `cd /usr/share/colcon_argcomplete/hook` : Navigates to the directory with the `colcon` autocompletion script.
- `source /usr/share/colcon_argcomplete/hook/colcon_argcomplete.bash` : Sources the `colcon` autocompletion script.
- `gedit ~/.bashrc` or `nano ~/.bashrc` : Opens the `.bashrc` file for editing.
- `echo "source /usr/share/colcon_argcomplete/hook/colcon_argcomplete.bash" >> ~/.bashrc` : Adds `colcon` autocompletion to `.bashrc` .
- `cd ~/ros2_ws` : Returns to your workspace directory.
- `colcon build` : Builds packages in the `src` directory and creates `build` , `install` , and `log` directories.
- `source install/setup.bash` : Sources the workspace setup file to make custom packages discoverable.
- `echo "source ~/ros2_ws/install/setup.bash" >> ~/.bashrc` : Adds workspace sourcing to `.bashrc` .

Episode 4: Create a ROS2 Python Package

This episode guides you through creating your first ROS 2 package.

Commands:

- `cd ~/ros2_ws/src` : Navigates to the source directory.
- `ros2 pkg create --build-type ament_python my_robot_controller --dependencies rclpy` : Creates a new Python package.
- `sudo snap install code --classic` : Installs Visual Studio Code using snap.
- `cd ~/ros2_ws` : Returns to workspace root.

- `colcon build` : Builds packages including the new `my_robot_controller` package.
- `pip3 list | grep setuptools` : Checks the installed `setuptools` version.
- `pip3 install setuptools==58.2.0` : Installs a specific version of `setuptools` (for fixing build errors).
- `source ~/.bashrc` : Sources the `.bashrc` file to update environment.

Episode 5: Create a ROS2 Node with Python and OOP

This episode demonstrates creating a basic ROS 2 node using object-oriented programming.

Commands:

- `cd ~/ros2_ws/src/my_robot_controller/my_robot_controller` : Navigates to the Python package directory.
- `touch my_first_node.py` : Creates an empty Python file.
- `chmod +x my_first_node.py` : Makes the file executable.
- `code .` : Opens the current directory in VS Code.
- `colcon build` : Builds the workspace after creating/modifying nodes.
- `source install/setup.bash` : Sources the workspace to make nodes discoverable.
- `ros2 run my_robot_controller my_first_node` : Runs the new node.
- `ros2 node list` : Lists all running ROS 2 nodes.
- `ros2 node info /first_node` : Displays information about a specific node.
- `colcon build --symlink-install` : Builds using symbolic links for faster Python development.

Episode 6: What is a ROS2 Topic?

This episode explains ROS 2 topics for communication between nodes.

Commands:

- `ros2 run demo_nodes_cpp talker` : Runs the node that publishes to `/chatter`.
- `ros2 run demo_nodes_cpp listener` : Runs the node that subscribes to `/chatter`.
- `rqt_graph` : Visualizes nodes and topics connections.
- `ros2 topic list` : Lists all active ROS 2 topics.
- `ros2 topic info /chatter` : Shows information about a specific topic.
- `ros2 interface show std_msgs/msg/String` : Displays the message type definition.
- `ros2 topic echo /chatter` : Displays received messages from a topic.
- `ros2 run turtlesim turtlesim_node` : Runs the turtle simulation.

- `ros2 run turtlesim turtle_teleop_key` : Runs keyboard control for the turtle.
- `ros2 topic info /turtle1/cmd_vel` : Shows information about the velocity command topic.
- `ros2 interface show geometry_msgs/msg/Twist` : Shows the velocity message type definition.

Episode 7: Write a ROS2 Publisher with Python

This episode guides you through creating a node that publishes messages.

Commands:

- `cd ~/ros2_ws/src/my_robot_controller/my_robot_controller` : Navigates to package directory.
- `touch draw_cycle.py` : Creates a new publisher file.
- `chmod +x draw_cycle.py` : Makes the script executable.
- `code .` : Opens VS Code.
- `ros2 run turtlesim turtlesim_node` : Runs turtlesim to test your publisher.
- `ros2 topic list` : Lists active topics to verify your publisher's topic.
- `ros2 topic info /turtle1/cmd_vel` : Checks information about the command velocity topic.
- `cd ~/ros2_ws` : Returns to workspace root.
- `colcon build --symlink-install` : Builds the workspace with the new publisher.
- `source install/setup.bash` : Sources the workspace.
- `ros2 run my_robot_controller draw_cycle` : Runs your publisher node.
- `rqt_graph` : Verifies connections between your publisher and turtlesim.
- `ros2 topic echo /turtle1/cmd_vel` : Monitors messages from your publisher.

Episode 8: Write a ROS2 Subscriber with Python

This episode explains creating a node that subscribes to a topic.

Commands:

- `cd ~/ros2_ws/src/my_robot_controller/my_robot_controller` : Navigates to package directory.
- `touch pose_subscriber.py` : Creates a new subscriber file.
- `chmod +x pose_subscriber.py` : Makes the script executable.
- `code .` : Opens VS Code.
- `ros2 topic list` : Lists topics to identify which to subscribe to.
- `rqt_graph` : Visualizes existing publishers on the target topic.
- `ros2 topic info /turtle1/pose` : Checks information about the pose topic.

- `ros2 interface show turtlesim/msg/Pose` : Shows the pose message definition.
- `ros2 topic echo /turtle1/pose` : Monitors messages on the pose topic.
- `cd ~/ros2_ws` : Returns to workspace root.
- `colcon build --symlink-install` : Builds the workspace with the new subscriber.
- `source install/setup.bash` : Sources the workspace.
- `ros2 run my_robot_controller pose_subscriber` : Runs your subscriber node.

Episode 9: Create a Closed Loop System with a Publisher and a Subscriber

This episode focuses on creating a closed-loop control system within a single node.

Key Concepts:

- **Closed-loop control:** System where output is fed back to adjust input.
- **Single node with subscriber and publisher:** Combining receiving data and sending commands.
- **Topics used:** `/turtle1/pose` (subscribed) and `/turtle1/cmd_vel` (published).

Commands:

- `ros2 run turtlesim turtlesim_node` : Runs the turtle simulator.
- `rqt_graph` : Visualizes the connections in the closed-loop system.
- `ros2 run my_robot_controller turtle_controller` : Runs the custom controller node.

Episode 10: What is a ROS2 Service?

This episode introduces ROS 2 services for request-response communication.

Commands:

- `ros2 run demo_nodes_cpp add_two_ints_server` : Runs a service server node.
- `ros2 service list` : Lists all active ROS 2 services.
- `ros2 node list` : Lists active nodes that may host services.
- `ros2 service type /add_two_ints` : Shows the type of a service.
- `ros2 interface show example_interfaces/srv/AddTwoInts` : Displays the service definition.
- `ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 2, b: 5}"` : Calls a service.
- `ros2 run turtlesim turtlesim_node` : Runs turtlesim with several services.

- **`ros2 service type /turtle1/set_pen`** : Shows the type of the set_pen service.
- **`ros2 interface show turtlesim/srv/SetPen`** : Displays the SetPen service definition.
- **`ros2 service call /turtle1/set_pen turtlesim/srv/SetPen "{r: 255, g: 0, b: 0, width: 3, off: 0}"`** : Changes the pen color.

Episode 11: Write a ROS2 Service Client with Python

This episode demonstrates creating a node that calls a ROS 2 service.

Key Concepts:

- Creating a service client with `self.create_client(srv_type, srv_name)`
- Importing service types from appropriate modules
- Waiting for service availability with `client.wait_for_service()`
- Creating and populating request objects
- Calling services asynchronously with `client.call_async(request)`
- Handling service responses with callback functions
- Implementing error handling for service calls

Commands:

- **`ros2 topic hz /turtle1/pose`** : Measures the publishing rate of the pose topic.