


ROS2 LAB #1: GETTING STARTED GUIDE

Abdelbacet Mhamdi

Senior-lecturer, Dept. of EE

ISSET Bizerte — Tunisia

 a-mhamdi

Abstract — This guide introduces the fundamentals of ROS2 (Robot Operating System 2) using the TurtleSim simulator. TurtleSim is a simple 2D simulator that helps beginners understand core ROS2 concepts without the complexity of real hardware.

I. ENVIRONMENT SETUP

A. Checking Your Shell

First, identify which shell you're currently using:

```
which $SHELL
```

This command returns the path to your current shell executable (bash, zsh, etc.). Different shells may require different setup procedures, so this information is important for proper configuration.

B. Sourcing ROS2 Installation

Add ROS2 commands and environment variables to the current terminal session:

```
source /opt/ros/humble/setup.bash
```

This step is essential as **ROS2** commands won't be available without sourcing. The sourcing must be done in every new terminal session, or you can add this line to your shell's configuration file (.bashrc, .zshrc) to make it automatic.

II. RUNNING YOUR FIRST ROS2 NODE

A **node** is a fundamental unit of computation. Each node is a separate process that performs a specific task. Nodes communicate with each other through topics, services, and actions. The command structure follows the pattern:

```
ros2 run <package_name> <executable_name>
```

To list all executables in a package:

```
ros2 pkg executables <package_name>
```

The following command lists all executables in the turtlesim package:

```
ros2 pkg executables turtlesim
```

Let's launch the **TurtleSim** simulator:

```
ros2 run turtlesim turtlesim_node
```

This command opens a blue window with a turtle in the center.

III. UNDERSTANDING TOPICS AND COMMUNICATION

A. Listing Active Topics

Topics are named communication channels where nodes can publish (send) or subscribe (receive) messages.

Display all currently active topics with their message types:

```
ros2 topic list -t
```

The **-t** flag shows the message type for each topic. Expected output includes topics such as:

- /turtle1/cmd_vel [geometry_msgs/msg/Twist]
- /turtle1/color_sensor [turtlesim/msg/Color]
- /turtle1/pose [turtlesim/msg/Pose]

B. Inspecting Message Structure

To understand the data structure of messages used in topics, you can inspect a specific message type. This command shows the Twist message structure:

```
ros2 interface show geometry_msgs/msg/Twist
```

The Twist message contains linear and angular velocity components for 3D space.

```
geometry_msgs/Vector3 linear
float64 x
float64 y
float64 z
geometry_msgs/Vector3 angular
float64 x
float64 y
float64 z
```

For the 2D turtle, we primarily use `linear.x` (forward/backward) and `angular.z` (rotation).

IV. PUBLISHING MESSAGES

A. Single Movement Command

If we want to move the turtle forward by 1 unit, we can publish a single message to the `/turtle1/cmd_vel` topic:

```
ros2 topic pub --once /turtle1/cmd_vel
geometry_msgs/msg/Twist "{linear: {x: 1.0}}"
```

Let's breakdown the structure of the previous instruction:

- `ros2 topic pub`: Command to publish to a topic
- `--once`: Publish only one message then exit
- `/turtle1/cmd_vel`: The target topic name
- `geometry_msgs/msg/Twist`: The message type
- `"{linear: {x: 1.0}}"`: Message data in YAML format

B. Continuous Movement Command

Now, we want to publish velocity commands continuously to create circular motion:

```
ros2 topic pub -r 1 /turtle1/cmd_vel
geometry_msgs/msg/Twist "{linear: {x: 1.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: .7}}"
```

The key parameters are:

- `-r 1`: Publish at a 1 Hz (i.e., 1 message per sec)
- `linear: {x: 1.0}`: Move forward at 1 unit/sec
- `angular: {z: .7}`: Rotate at 0.7 radians/sec

As it can be observed, the combination of forward motion and rotation creates circular movement.

V. MONITORING COMMUNICATION

In **ROS2**, it is crucial to monitor messages being published on a topic in real-time. This can be done using the `ros2 topic echo` command:

```
ros2 topic echo /turtle1/cmd_vel
```

This debugging tool displays all messages on the specified topic. When used alongside the continuous movement command, we'll see the velocity messages being printed continuously.

VI. INTERACTIVE CONTROL

A. Direct Turtle Teleop

Let's use TurtleSim's built-in keyboard control to teleoperate the turtle:

```
ros2 run turtlesim turtle_teleop_key
```

This executable is specifically designed for TurtleSim and publishes directly to `/turtle1/cmd_vel`, eliminating the need for topic remapping.

B. Keyboard Teleop with Remapping

In general, it is possible to control the turtle using keyboard input with topic remapping:

```
ros2 run teleop_twist_keyboard
teleop_twist_keyboard --ros-args --remap
cmd_vel:=/turtle1/cmd_vel
```

The command components are:

- `teleop_twist_keyboard`: Package and executable name
- `--ros-args --remap`: ROS2 argument for topic remapping
- `cmd_vel:=/turtle1/cmd_vel`: Maps the default `cmd_vel` topic to `/turtle1/cmd_vel`

VII. SYSTEM VISUALIZATION

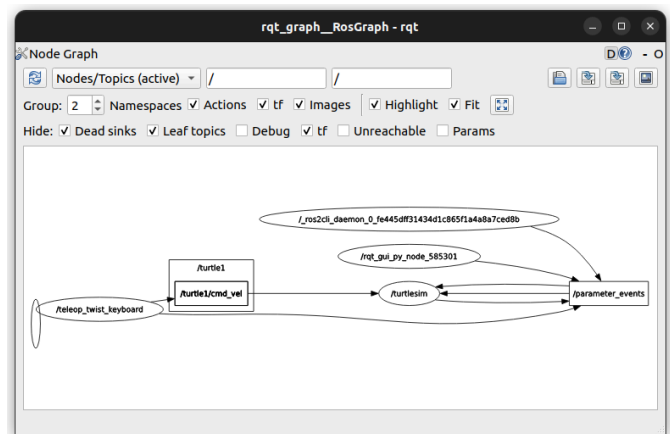
There is a tool to visualize the relationships between nodes and topics, called `rqt_graph`:

```
rqt_graph
```

This opens a graphical tool displaying:

- Nodes as ovals
- Topics as rectangles
- Arrows showing communication direction

The visualization helps understand data flow in **ROS2** systems and is invaluable for debugging communication issues in complex robotic applications.



VIII. SUMMARY

A. Terminology

Concept	Description
Nodes	Independent processes performing specific tasks
Topics	Named communication channels for message passing
Messages	Structured data types (e.g., Twist for velocity)
Publishing	Sending messages to topics
Subscribing	Receiving messages from topics
Packages	Collections of related nodes and resources
Remapping	Redirecting topic names for system flexibility

B. Command Reference

Command	Purpose
which \$SHELL	Check current shell
source /opt/ros/humble/setup.bash	Source ROS2 environment
ros2 run <package> <executable>	Run a ROS2 node
ros2 topic list -t	List active topics with types
ros2 interface show <msg_type>	Show message structure
ros2 topic pub <topic> <type> <data>	Publish to topic
ros2 topic echo <topic>	Monitor topic messages
rqt_graph	Visualize system graph