Tytus Felbor
Andrea Infanti
ROS & Gazebo : Lab 1 & 2 Report

Lab 1 : Intro to ROS 2

1. Create your own program in your own package

To start off my lab partner created a package named *justin* () using the suggested `ament_cmake` as the build type. Then we proceeded to update the `package.xml` to include dependencies for `rclcpp`, `std_msgs`, `geometry_msgs`, and `sensor_msgs`. Lastly, we modified the `CMakeLists.txt` file to find these packages and compile two executables: `talker` - `MinimalPublisher.cpp` & `listener` - `subscriber_member_function.cpp`.

2. Send command to the turtlesim

In our `MinimalPublisher.cpp`, we initialized the publisher using `create_publisher<geometry_msgs::msg::Twist>`. Its goal is to send messages to the topic `/turtle1/cmd_vel`, which then enables us to send velocity commands directly to the turtlesim node.

3. Reading joystick data

Next, we implemented a subscription to the `/joy` topic in the same node. This subscriber listens for `sensor_msgs::msg::Joy` messages, which contain the data about the axes and buttons from the connected joystick.

4. Control the turtle

Inside the implemented `joy_callback` function, we mapped the joystick axes to the turtle's velocity. To be specific, we assigned `axes[1]` (the vertical axis) to the linear X velocity and `axes[0]` (the horizontal axis) to the angular Z velocity. Both values are then multiplied by a factor of 2.0 before being published to the turtle, which allows the joystick to steer the robot in a little quicker fashion.

Lab 2 : Intro to ROS 2

1. Create a launch file

The next step was implementing a Python launch file named *lab2_launch.py*. This script enables the startup of our entire system by launching these three specific nodes:

- *turtlesim_node*: responsible for the simulation of our small environment.
- *teleop_twist_keyboard*: a keyboard controller (since I didn't have a joystick available) was running in a new terminal and its output acts as an input to our controller.
- *turtle_controller* (our talker executable): this node loads the configuration file and processes the inputs.

2. Parameters

Lastly, we created a configuration file in *config/params.yaml* which defines two parameters: *linear_gain_x* (set to 2.0) and *angular_gain_z* (set to 3.0). In the C++ code, *MinimalPublisher.cpp*, we updated the node to:

- Declare and retrieve the above mentioned parameters.
- Subscribe to */cmd_vel_in*.
- Apply the gains to the incoming linear and angular velocities.
- Publish the modified velocities to */turtle1/cmd_vel* to control the turtle with the adjusted sensitivity.