# Report: ROS Lab Sessions 1 & 2 CAZAUBON Lorenz - DESUE Léo

## Table of Contents



## Lab 1

### First Hello World package

*For our very first package we decided to follow this tutorial from start to finish.*

- Once all files were correctly created, we built the package with this command:

```
lorenz@Legion:~/Documents/ros2_ws$ colcon build --packages-select
cpp_pubsub
Starting >>> cpp_pubsub
Finished <<< cpp_pubsub [0.13s]
Summary: 1 package finished [0.30s]
```

- The build was succesful, our package is ready to be used. To do so we simply had to run the talker and listener in two distinct terminals.

```
#First terminal
lorenz@Legion:~/Documents/ros2_ws$ . install/setup.bash
lorenz@Legion:~/Documents/ros2_ws$ ros2 run cpp_pubsub talker
[INFO] [1731851398.103893880] [minimal_publisher]: Publishing: 'Hello,
world! 0'
[INFO] [1731851398.603926237] [minimal_publisher]: Publishing: 'Hello,
world! 1'
[INFO] [1731851399.104009651] [minimal_publisher]: Publishing: 'Hello,
world! 2'
[INFO] [1731851399.604020636] [minimal_publisher]: Publishing: 'Hello,
world! 3'
[INFO] [1731851400.104067241] [minimal_publisher]: Publishing: 'Hello,
world! 4'
```

```
[INFO] [1731851400.604100785] [minimal_publisher]: Publishing: 'Hello,
world! 5'
```

```
#Second terminal
lorenz@Legion:~/Documents/ros2_ws$ . install/setup.bash
lorenz@Legion:~/Documents/ros2_ws$ ros2 run cpp_pubsub listener
[INFO] [1731851398.104100483] [minimal_subscriber]: I heard: 'Hello,
world! 0'
[INFO] [1731851398.604107905] [minimal_subscriber]: I heard: 'Hello,
world! 1'
[INFO] [1731851399.104287497] [minimal_subscriber]: I heard: 'Hello,
world! 2'
[INFO] [1731851399.604303230] [minimal_subscriber]: I heard: 'Hello,
world! 3'
[INFO] [1731851400.104361709] [minimal_subscriber]: I heard: 'Hello,
world! 4'
[INFO] [1731851400.604389317] [minimal_subscriber]: I heard: 'Hello,
world! 5'
```

### Turtlesim Publisher

- We launched both the turtlesim_node and turtle_teleop_key nodes in two terminals :

```
#First terminal
lorenz@Legion:~/Documents/ros2_ws$ ros2 run turtlesim turtlesim_node
qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in ""
[INFO] [1731852837.026445475] [turtlesim]: Starting turtlesim with
node name /turtlesim
[INFO] [1731852837.029203595] [turtlesim]: Spawning turtle [turtle1]
at x=[5,544445], y=[5,544445], theta=[0,000000]
```

```
#Second terminal
lorenz@Legion:~/Documents/ros2_ws$ ros2 run turtlesim
turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to
cancel a rotation.
'Q' to quit.
```

- Then we used the following commands to find on which topic and what type of data we need to send to move the turtle.

```
lorenz@Legion:~/Documents/ros2_ws$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose

lorenz@Legion:~/Documents/ros2_ws$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 1
```

**Topic:** `/turtle1/cmd_vel`

**Data:** `geometry_msgs/msg/Twist`

- Now that we know on wich topic and what data we need to publish we built a new package named `turtle_control` that used most of the code from our very first package `cpp_pubsub`.

  We only had a few lines to modify for this to work :

  **turtle_publisher.cpp**

```cpp
#include <chrono>
#include <functional>
#include <memory>

#include "rclcpp/rclcpp.hpp"
#include "geometry_msgs/msg/twist.hpp"

using namespace std::chrono_literals;

class TurtlePublisher : public rclcpp::Node
{
public:
    TurtlePublisher() : Node("turtle_publisher")
    {
        publisher_ = this->create_publisher<geometry_msgs::msg::Twist>
("/turtle1/cmd_vel", 10);
        timer_ = this->create_wall_timer(
            500ms, std::bind(&TurtlePublisher::publish_message,
this));
    }

private:
    void publish_message()
    {
        auto message = geometry_msgs::msg::Twist();
        message.linear.x = 1.0;  // Vitesse linéaire
        message.angular.z = 0.5; // Vitesse angulaire
        publisher_->publish(message);
```

```
    }

    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr
publisher_;
    rclcpp::TimerBase::SharedPtr timer_;
};

int main(int argc, char *argv[])
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<TurtlePublisher>());
    rclcpp::shutdown();
    return 0;
}
```

**CMakeLists.txt**

```
//add the following
find_package(geometry_msgs REQUIRED)

add_executable(turtle_publisher src/turtle_publisher.cpp)
ament_target_dependencies(turtle_publisher rclcpp geometry_msgs)

install(TARGETS turtle_publisher
        DESTINATION lib/${PROJECT_NAME})
```

**package.xml**

```
<!--add the following-->
<depend>geometry_msgs</depend>
```

- Perfect ! Now we simply need to build our package, run it and turtlesim to enjoy a turtle making circles. xD

```
lorenz@Legion:~/Documents/ros2_ws$ ros2 run turtle_control
turtle_publisher
```

 Turtle Making Circles

## Turtlesim Joy Control

- For this part of the lab session, we created a new **.cpp** file called `turtle_joy.cpp` to control the turtle with the joystick.
- Now we need to find out how the joy node works to get the data sent by the joystick and publish it to the turtlesim node. To do so, we will use the same process as before :

```
lorenz@Legion:~/Documents/ros2_ws$ ros2 topic list
/joy
/joy/set_feedback
/parameter_events
/rosout

lorenz@Legion:~/Documents/ros2_ws$ ros2 topic info /joy
Type: sensor_msgs/msg/Joy
Publisher count: 1
Subscription count: 0
```

**Topic:** `/joy`

**Data:** `sensor_msgs/msg/Joy`

- Next setp is to modify our code to subscribe to the joy_node and publish the commands to the turtlesim_node.

  Here's the code :

  **turtle_joy.cpp**

```
#include "rclcpp/rclcpp.hpp"
#include "geometry_msgs/msg/twist.hpp"
#include "sensor_msgs/msg/joy.hpp"

class TurtleJoy : public rclcpp::Node {
public:
    TurtleJoy() : Node("turtle_joy") {

        publisher_ = this->create_publisher<geometry_msgs::msg::Twist>
("/turtle1/cmd_vel", 10);

        subscription_ = this-
>create_subscription<sensor_msgs::msg::Joy>("/joy", 10,
std::bind(&TurtleJoy::joy_callback, this, std::placeholders::_1));
    }

private:
    void joy_callback(const sensor_msgs::msg::Joy::SharedPtr msg) {

        auto twist = geometry_msgs::msg::Twist();

        twist.linear.x = msg->axes[1];
        twist.linear.y = msg->axes[0];
        twist.angular.z = msg->axes[3];

        RCLCPP_INFO(this->get_logger(), "Values - v_x: %f, v_y: %f,
a_z: %f", msg->axes[1], msg->axes[0], msg->axes[3]);

        publisher_->publish(twist);
```

```
    }

    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr
publisher_;
    rclcpp::Subscription<sensor_msgs::msg::Joy>::SharedPtr
subscription_;
};

int main(int argc, char * argv[]) {
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<TurtleJoy>());
    rclcpp::shutdown();
    return 0;
}
```

**CMakeLists.txt**

```
// Add the following
find_package(sensor_msgs REQUIRED)

add_executable(turtle_joy src/turtle_joy.cpp)
ament_target_dependencies(turtle_joy rclcpp geometry_msgs sensor_msgs)

install(TARGETS turtle_joy
        DESTINATION lib/${PROJECT_NAME})
```

**package.xml**

```
<!--add the following-->
<depend>sensor_msgs</depend>
```

- To test our new node. Same as before we need to launch both nodes in two terminals.

```
#First terminal
lorenz@Legion:~/Documents/ros2_ws$ ros2 run turtlesim turtlesim_node
qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in ""
[INFO] [1731852837.026445475] [turtlesim]: Starting turtlesim with
node name /turtlesim
[INFO] [1731852837.029203595] [turtlesim]: Spawning turtle [turtle1]
at x=[5,544445], y=[5,544445], theta=[0,000000]
```

```
#Second terminal
lorenz@Legion:~/Documents/ros2_ws$ ros2 run turtle_control turtle_joy
```


- Now we can control the turtle with the joystick !

# Lab 2

## Launch

- It's a bit annoying to launch in different terminals each node to use what we created in the previous lab, one solution is to use launch files.

  Once executed all the nodes will be automatically launched, saving us some time.

- Inside our package we created a folder called `launch`, and wrote some **python** code.

  **turtlesim_joy_launch.py**

```
from launch import LaunchDescription
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory

def generate_launch_description():
return LaunchDescription([
    Node(
        package='turtlesim',
        executable='turtlesim_node',
        name='turtlesim_node'
    ),
    Node(
        package='joy',
        executable='joy_node',
        name='joy_node'
    ),
    Node(
        package='turtle_control',
        executable='turtle_joy',
        name='turtle_joy'
    ),
])
```

- Time to test our launch file :

```
lorenz@Legion:~/Documents/ros2_ws$ ros2 launch turtle_control
turtlesim_joy_launch.py
[INFO] [launch]: All log files can be found below
/home/lorenz/.ros/log/2024-11-17-17-15-04-969365-Legion-19508
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [turtlesim_node-1]: process started with pid [19509]
[INFO] [joy_node-2]: process started with pid [19511]
[INFO] [turtle_joy-3]: process started with pid [19513]
```

```
[turtlesim_node-1] qt.qpa.plugin: Could not find the Qt platform
plugin "wayland" in ""
[turtlesim_node-1] [INFO] [1731860105.060040919] [turtlesim_node]:
Starting turtlesim with node name /turtlesim_node
[turtlesim_node-1] [INFO] [1731860105.062914182] [turtlesim_node]:
Spawning turtle [turtle1] at x=[5,544445], y=[5,544445], theta=
[0,000000]
```

## Parameters

- Now we'd like to be able to dynamically change some values in our code. To do so we are going to create a `config` folder and `params.yaml` file.

**params.yaml**

```
node_name:
    ros__parameters:
        linear_gain_x: 1.0
        linear_gain_y: 1.0
        angular_gain_z: 1.0
```

**launch.py**

```
#add the following
import os
param_file = os.path.join(
get_package_share_directory('turtle_control'),
'config',
'params.yaml'
)
```

**turtle_joy.cpp**

```
// Add the following in public:
this->declare_parameter("linear_gain_x", 1.0);
this->declare_parameter("linear_gain_y", 1.0);
this->declare_parameter("angular_gain", 1.0);
```

```
// Add the following in private:
auto linear_x = get_parameter("linear_gain_x").as_double();
auto linear_y = get_parameter("linear_gain_y").as_double();
auto angular_z = get_parameter("angular_gain").as_double();


twist.linear.x = msg->axes[1] * linear_x;
```

```
twist.linear.y = msg->axes[0] * linear_y;
twist.angular.z = msg->axes[3] * angular_z;


RCLCPP_INFO(this->get_logger(), "Gains - linear_x: %f, linear_y: %f,
angular_z: %f", linear_x, linear_y, angular_z);
```

- Let's try to modify the values directly from the terminal :

```
#First Terminal
lorenz@Legion:~/Documents/ros2_ws$ ros2 launch turtle_control
turtlesim_joy_launch.py
[INFO] [launch]: All log files can be found below
/home/lorenz/.ros/log/2024-11-17-17-39-08-112650-Legion-21021
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [turtlesim_node-1]: process started with pid [21022]
[INFO] [joy_node-2]: process started with pid [21024]
[INFO] [turtle_joy-3]: process started with pid [21026]
[turtlesim_node-1] qt.qpa.plugin: Could not find the Qt platform
plugin "wayland" in ""
[turtlesim_node-1] [INFO] [1731861548.203783985] [turtlesim_node]:
Starting turtlesim with node name /turtlesim_node
[turtlesim_node-1] [INFO] [1731861548.206426936] [turtlesim_node]:
Spawning turtle [turtle1] at x=[5,544445], y=[5,544445], theta=
[0,000000]
```

```
#Second Terminal
lorenz@Legion:~/Documents/ros2_ws$ ros2 param list
/joy_node:
    autorepeat_rate
    coalesce_interval_ms
    deadzone
    device_id
    device_name
    qos_overrides./parameter_events.publisher.depth
    qos_overrides./parameter_events.publisher.durability
    qos_overrides./parameter_events.publisher.history
    qos_overrides./parameter_events.publisher.reliability
    sticky_buttons
    use_sim_time
/turtle_joy:
    angular_gain
    linear_gain_x
    linear_gain_y
    qos_overrides./parameter_events.publisher.depth
    qos_overrides./parameter_events.publisher.durability
    qos_overrides./parameter_events.publisher.history
    qos_overrides./parameter_events.publisher.reliability
    use_sim_time
```

```
/turtlesim_node:
    background_b
    background_g
    background_r
    qos_overrides./parameter_events.publisher.depth
    qos_overrides./parameter_events.publisher.durability
    qos_overrides./parameter_events.publisher.history
    qos_overrides./parameter_events.publisher.reliability
    use_sim_time

lorenz@Legion:~/Documents/ros2_ws$ ros2 param set turtle_joy
angular_gain 10.0
Set parameter successful
```