Exercise : Create a Custom ROS2 Package for Mapping and Navigation (TurtleBot3)

Goal of the Exercise

In this exercise, you will **design and implement your own ROS2 package** that launches a **complete SLAM and Navigation pipeline** for a TurtleBot3 robot.

You will not use the default launch commands directly. Instead, you must **create custom launch files** that orchestrate mapping and navigation using existing ROS2 packages (`slam_toolbox`, `nav2_bringup`, TurtleBot3 drivers).

This exercise evaluates your understanding of:

- ROS2 package structure
- Launch files
- SLAM → Navigation workflow
- Inputs, outputs, and data flow
- Proper use of parameters and arguments

Learning Objectives

By the end of this exercise, the student should be able to:

- Create a ROS2 package using `ament_python`
- Write ROS2 launch files
- Launch SLAM for mapping using `slam_toolbox`
- Save and load maps
- Launch Navigation2 (Nav2) using a saved map

- Understand which nodes publish and consume `/scan`, `/odom`, `/map`, `/amcl_pose`, `/cmd_vel`

Context

You are working with a **TurtleBot3 robot (real or simulated)** running ROS2 Humble.

The robot has:

- 2D LiDAR publishing `/scan`
- Odometry publishing `/odom`
- IMU publishing `/imu`

Your task is to prepare a reusable launch setup so that:

- One command starts **mapping**
- Another command starts **navigation using a saved map**

Exercise Requirements

1. Create a New ROS2 Package

Create a new package named:

`tb3_mapping_navigation`

Constraints:

- Language: `ament_python`
- The package must contain a `launch/` directory
- All launch files must be written in Python

2. Mapping Launch File

Create a launch file named:

`mapping.launch.py`

This launch file must:

1. Launch the TurtleBot3 simulation or robot driver
2. Launch `slam_toolbox` in **online mapping mode**
3. Launch a keyboard teleoperation node
4. Allow the user to drive the robot to build a map

Mapping Inputs

- `/scan`
- `/odom`
- TF frames

Mapping Outputs

- `/map`
- `map → odom` transform

Expected result:

- A real-time occupancy grid map visible in RViz

3. Save the Map (Manual Step)

After mapping, the student must save the map using:

```
ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap
"{name: {data: '/home/<user>/tb3_map'}}"
```

Deliverable:

- `tb3_map.yaml`
- `tb3_map.pgm`

4. Navigation Launch File

Create a second launch file named:

`navigation.launch.py`

This launch file must:

1. Start the TurtleBot3 robot drivers (or simulation)
2. Start the Navigation2 stack
3. Load the previously saved map using `map_server`
4. Start the `amcl` node for localization
5. Allow goal-based navigation

Navigation Inputs

- `/scan`
- `/odom`
- Map file (`.yaml`)

Navigation Outputs

- `/amcl_pose`
- `/cmd_vel`

Expected result:

- The robot localizes on the map
- The robot navigates autonomously to target positions sent from RViz

5. Launch Arguments (Mandatory)

Your launch files must support launch arguments:

Mapping launch arguments

- `use_sim_time` (true / false)

Navigation launch arguments

- `use_sim_time`
- `map_file` (path to the `.yaml` map)

Package Structure (Expected)

```
tb3_mapping_navigation/
├── package.xml
├── setup.py
├── resource/
│   └── tb3_mapping_navigation
├── tb3_mapping_navigation/
│   └── __init__.py
└── launch/
    ├── mapping.launch.py
    └── navigation.launch.py
```

Validation Checklist

To validate the exercise, the student must demonstrate:

Mapping

- `/map` topic is published
- Robot movement updates the map
- SLAM runs without TF errors

Navigation

- `/amcl_pose` topic is present
- Robot appears correctly localized in RViz
- Robot reaches navigation goals autonomously
- `/cmd_vel` is published by Nav2