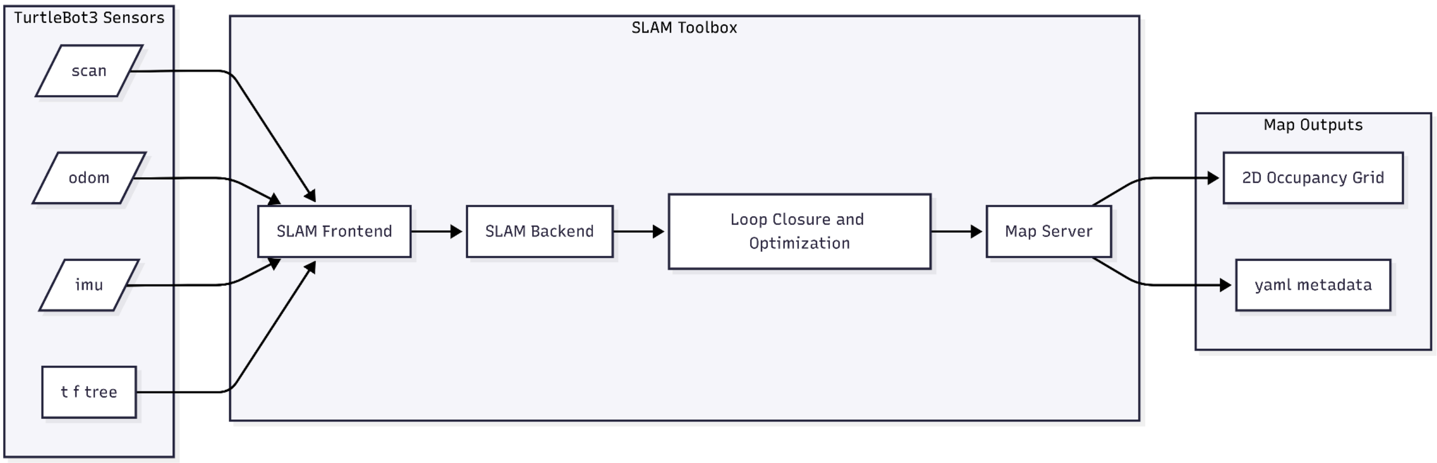
FULL SOLUTION : PART 1: MAPPING (TurtleBot3 + SLAM Toolbox)

Exercise: Create a Custom ROS2 Package for Mapping and Navigation



1. Create the Package

Open a terminal inside your ROS2 workspace:

```
cd ~/ros2_ws/src
```

Create the package:

```
ros2 pkg create tb3_mapping_navigation --build-type ament_python
```

This generates:

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

2 . Create the Launch Folder

```
mkdir ~/ros2_ws/src/tb3_mapping_navigation/launch
```

3. Add SLAM Parameters

Create a config folder:

```
mkdir ~/ros2_ws/src/tb3_mapping_navigation/config
```

Create the file:

```
nano ~/ros2_ws/src/tb3_mapping_navigation/config/slam_params.yaml
```

Paste this inside:

```
slam_toolbox:
  ros__parameters:
    use_sim_time: true
    slam_params_file: "mapper_params_online_sync.yaml"
    mode: "mapping"
```

Save and exit.

4. Create the Mapping Launch File

```
nano ~/ros2_ws/src/tb3_mapping_navigation/launch/mapping.launch.py
```

Paste the **full solution code**:

```
#!/usr/bin/env python3
"""
TurtleBot3 Mapping Launch File - SLAM with Gazebo + Teleop
```

```
"""

import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import (
    DeclareLaunchArgument,
    IncludeLaunchDescription,
    ExecuteProcess,
    TimerAction,
    LogInfo,
    OpaqueFunction,
)
from launch.launch_description_sources import
PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node

def launch_setup(context, *args, **kwargs):

    use_sim_time = context.launch_configurations.get('use_sim_time',
'true')
    with_teleop = context.launch_configurations.get('with_teleop',
'true')

    # Paths
    turtlebot3_gazebo_pkg =
get_package_share_directory('turtlebot3_gazebo')
    tb3_mapping_pkg =
get_package_share_directory('tb3_mapping_navigation')

    world_path = os.path.join(
        turtlebot3_gazebo_pkg, 'worlds', 'turtlebot3_world.world'
    )

    # ------------------------------
    # 1. Gazebo + TurtleBot3
    # ------------------------------
    gazebo_launch = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([
            os.path.join(turtlebot3_gazebo_pkg, 'launch',
'turtlebot3_world.launch.py')
        ]),
        launch_arguments={
            'use_sim_time': use_sim_time,
            'world': world_path,
        }.items()
    )

    actions = [gazebo_launch]

    # -----------------------------
    # 2. SLAM Toolbox (Delayed)
    # -----------------------------
    slam_params = os.path.join(tb3_mapping_pkg, 'config',
'slam_params.yaml')

    slam_node = Node(
        package='slam_toolbox',
        executable='async_slam_toolbox_node',
        name='slam_toolbox',
        output='screen',
        parameters=[
            slam_params,
            {'use_sim_time': use_sim_time == 'true'}
        ],
    )

    actions.append(TimerAction(period=8.0, actions=[slam_node]))

    # -----------------------------
    # 3. Teleoperation (Optional)
    # -----------------------------
    if with_teleop.lower() == 'true':
        teleop = ExecuteProcess(
            cmd=[
                'xterm', '-e',
                'ros2', 'run', 'teleop_twist_keyboard',
'teleop_twist_keyboard'
            ],
            output='log',
        )

        actions.append(TimerAction(period=10.0, actions=[teleop]))
        actions.append(LogInfo(msg="Teleop starts in a separate
xterm window..."))

    return actions


def generate_launch_description():

    return LaunchDescription([
        DeclareLaunchArgument(
            'use_sim_time',
            default_value='true',
            description='Use simulation time'
        ),
        DeclareLaunchArgument(
            'with_teleop',
            default_value='true',
            description='Launch teleop keyboard'
        ),
        LogInfo(msg='Starting TurtleBot3 Mapping Pipeline...'),
        OpaqueFunction(function=launch_setup),
    ])
```

Save and exit.

5. Update setup.py

Open:

```
nano ~/ros2_ws/src/tb3_mapping_navigation/setup.py
```

Edit entry points:

```
entry_points={
    'console_scripts': [],
    'launch': [
        'mapping =
tb3_mapping_navigation.mapping:generate_launch_description',
    ],
},
```

Save.

6. Build the Package

```
cd ~/ros2_ws
colcon build --packages-select tb3_mapping_navigation
```

Source workspace:

```
source install/setup.bash
```

7. Run the Mapping Pipeline

```
ros2 launch tb3_mapping_navigation mapping.launch.py
```

What will happen:

✔ Gazebo opens
✔ TurtleBot3 loads
✔ LiDAR publishes `/scan`
✔ Odometry publishes `/odom`
✔ After 8 seconds → SLAM Toolbox starts
✔ After 10 seconds → teleop keyboard window opens

8. Drive the Robot to Build the Map

In the teleop window:

```
W = forward
```

```
A = rotate left
D = rotate right
S = stop
```

As you move:

✔ `/map` topic updates
✔ Occupancy grid appears in RViz

9. Save the Map

Run:

```
ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap \

  "{name: {data: '/home/houssem/ros2_ws/src/tb3_mapping_navigation/maps/tb3_map'}}"
```

The following files appear:

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

These will be used later in **Part 2 (Navigation)**.

10. Expected Mapping Result

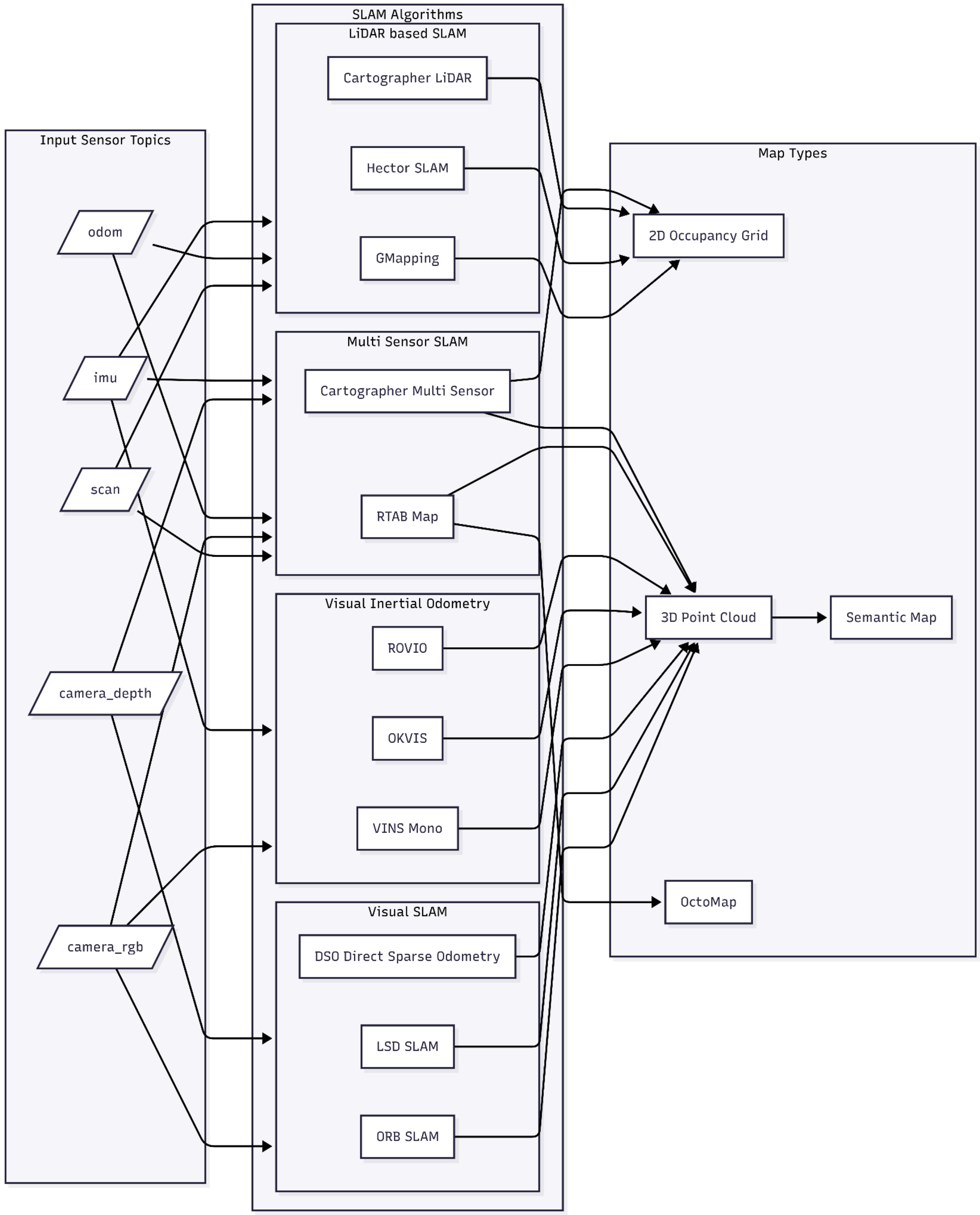
A correct solution will show:

✔ `/map` is being published
✔ Map grows as the robot moves
✔ No TF errors appear
✔ Student can save the map
✔ SLAM Toolbox outputs:

```
[INFO] Map saved!
```

11. Final File Tree (Correct Structure)

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


SLAM Algorithms
LiDAR based SLAM
Cartographer LiDAR
Hector SLAM
GMapping
Input Sensor Topics
odom
imu
scan
camera_depth
camera_rgb
Multi Sensor SLAM
Cartographer Multi Sensor
RTAB Map
Visual Inertial Odometry
ROVIO
OKVIS
VINS Mono
Visual SLAM
DSO Direct Sparse Odometry
LSD SLAM
ORB SLAM
Map Types
2D Occupancy Grid
3D Point Cloud
Semantic Map
OctoMap