Introduction to Mobile Robotics

Practical Work 8: Frontier-based Exploration

Zhi Yan

ENSTA, Institut Polytechnique de Paris

1 Objective

The goal of this session is to implement a fully autonomous frontier-based exploration system in ROS 2. You will *not* be writing the algorithm from scratch. Instead, we will use the powerful, existing ROS 2 packages designed for this. The goal is to learn how to **launch, configure, and observe** a fully autonomous exploration system, relying on the layered architecture of ROS 2.

2 The Tool: `ros2_exploration`

`ros2_exploration` is a package that implements the frontier-based exploration, exactly as described in the lecture.

- It works by subscribing to the `/map` topic (generated by SLAM).
- It detects frontiers, selects a target, and sends that target as a *goal* to the Nav2 navigation stack.
- When the robot reaches the goal (or fails), it requests the next goal, and the cycle repeats.

This is also a perfect example of system integration (cf. Lecture 3): the `ros2_exploration` package is the "brain" sending commands to the "body" Nav2 (which we will cover in the next lesson).

3 Prerequisites

3.1 Install Nav2

First, install the main Nav2 packages which we will use for navigation:

```
sudo apt update
sudo apt install ros-humble-nav2-bringup
```

3.2 Install `ros2_exploration`

The `ros2_exploration` package is not available as a Debian, so we must build it from source.

```
# 1. Enter your workspace
cd ~/ros2_ws/src

# 2. Clone the repository
git clone https://github.com/yzrobot/ros2_exploration.git
```

```
# 3. Go back to the workspace root
cd ~/ros2_ws

# 4. Install dependencies
rosdep install -i --from-path src --rosdistro humble -y

# 5. Build the workspace
colcon build --symlink-install
```

3.3 Understanding the velocity command mismatch

For some of you, there may be a communication mismatch between Nav2 and the Gazebo simulator:

- **Nav2** publishes velocity commands on `/cmd_vel` as a `geometry_msgs/msg/Twist` message.
- **Gazebo** (via `ros_gz_bridge`) expects velocity commands on `/cmd_vel` as a `geometry_msgs/msg/TwistStamped` message.

The good news is that the `ros2_exploration` package provides a node to fix this! The `twist_converter.py` node subscribes to a `Twist` topic (`/cmd_vel`) and republishes it as a `TwistStamped` topic (`/cmd_vel_stamped`), and the `exploration_launch.py` file automatically starts this converter for you. Now, you must tell `ros_gz_bridge` to listen to the `/cmd_vel_stamped` topic that `twist_converter` will publish to (instead of its default `/cmd_vel`). To do so:

1. Edit the `spawn_turtlebot3.launch.py` file. You can find its location with this command:

```
# This will show you the path, e.g., /opt/ros/humble
ros2 pkg prefix turtlebot3_gazebo

# The file you need to edit is:
# [PATH_FROM_ABOVE]/share/turtlebot3_gazebo/launch/spawn_turtlebot3.
    launch.py
```

2. Search for the `start_gazebo_ros_bridge_cmd` node.
3. Add the `remappings` argument to this node:

```
start_gazebo_ros_bridge_cmd = Node(
    package='ros_gz_bridge',
    executable='parameter_bridge',
    arguments=[
        '--ros-args',
        '-p',
        f'config_file:={bridge_params}',
    ],
    output='screen',
    # ADD THESE LINES:
    remappings=[
        ('/cmd_vel', '/cmd_vel_stamped'),
    ]
)
```

4. Save and exit the file.

3.4 Fix TurtleBot3 friction

IMPORTANT: You must undo the changes you made to the TurtleBot3 `model.sdf` file in Practical Work 7. Change the friction values `<mu>` and `<mu2>` back to `100000.0`. The navigation stack performs poorly with wheel slip.

4 The 5-terminal Setup

For this practical work, we need 5 terminals:

- **Terminal 1 (T1): Simulation**: Gazebo.
- **Terminal 2 (T2): SLAM**: The `slam_toolbox` to build the map.
- **Terminal 4 (T3): Visualization**: RViz2.
- **Terminal 3 (T4): Navigation**: Nav2.
- **Terminal 3 (T5): Exploration**: The `ros2_exploration` to send the goal.

5 Experiment: Autonomous Exploration

1. **T1 - Launch Gazebo:**

```
export TURTLEBOT3_MODEL=burger
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```

2. **T2 - Launch `slam_toolbox`:**

```
# Don't forget use_sim_time!
ros2 launch slam_toolbox online_async_launch.py use_sim_time:=True
```

3. **T3 - Launch RViz2:**

```
# This launchfile loads a perfect RViz config for navigation
ros2 launch nav2_bringup rviz_launch.py
```

4. **T4 - Launch Nav2:** Before automating, ensure the navigation system loads correctly.

```
# 'map_subscribe_transient_local:=true' tells Nav2 to use the live map
    from SLAM
ros2 launch nav2_bringup navigation_launch.py use_sim_time:=True
   map_subscribe_transient_local:=True
```

5. **T5 - Launch `ros2_exploration`:** This is the main step.

   REMINDER: Before running this command, make sure the workspace you just built is sourced so that ROS 2 can find the `ros2_exploration` package.

```
# Don't forget use_sim_time!
ros2 launch ros2_exploration exploration_launch.py use_sim_time:=True
```

6 Analysis and Observation

Watch RViz2. Almost immediately, the robot should begin to move on its own.

- The robot receives a goal (a point on a frontier) from the exploration node (T5).
- The navigation stack (T4) plans a global path (in red) to that goal.
- The local controller (T4) follows that path, avoiding obstacles.
- Once the goal is reached, the robot stops, "looks" around (updating its SLAM map (T2)), and the exploration node (T5) calculates the *next* best frontier and sends a new goal.

Things to observe:

- In RViz2, click first "Add" then "By topic" and add the `/exploration_frontiers` topic (type `visualization_msgs/msg/Marker`). You should see the detected frontiers in real-time.
- Watch the robot explore the world. How does it decide where to go?
- What happens when the entire map is explored (there are no accessible frontiers left)?
- How long did the entire exploration take?
- Do you have any ideas on how to improve the exploration performance?

7 Saving Your Autonomous Map

Once the robot has explored the entire environment, you can save your map.

1. **Before stopping anything**, open a new terminal (T6) and use the `slam_toolbox` service to save the map:

```
# While T1, T2, T3, T4, T5 are still running:
ros2 run nav2_map_server map_saver_cli -f exploration_map
```

2. Now you can stop all your terminals (Ctrl+C).
3. Compare this `exploration_map.pgm` with the one you made manually in Practical Work 7. Is the quality better or worse? Why?

Congratulations! You have just launched a fully autonomous mobile robot system. This system integrates: 1) **simulation (Gazebo),** 2) **SLAM (slam_toolbox),** 3) **navigation (Nav2), and** 4) **exploration (ros2_exploration). This layered architecture is fundamental to mobile robotics.**