Introduction to Mobile Robotics

Practical Work 7: `slam_toolbox`

Zhi Yan

ENSTA, Institut Polytechnique de Paris

1 Objective

The goal of this session is to build a 2D map of a simulated environment and experimentally investigate the core principles of SLAM. We will use `slam_toolbox`, a popular graph-based SLAM implementation in ROS 2, as our tool to answer three fundamental questions:

1. **The problem:** How bad is odometry drift, and how can we visualize it?
2. **The front-end:** How does a SLAM system continuously correct for this drift?
3. **The back-end:** What does a "loop closure" event look like, and what is its effect on the map?

You will also learn to launch the necessary nodes, visualize the mapping process, and save the generated map for future use.

2 Prerequisites

- A working ROS 2 Humble installation with Gazebo and TurtleBot3 packages (from Practical work 4).
- Familiarity with launching Gazebo simulations and using RViz2.
- The SLAM tools we will install below.

3 Installing SLAM Tools

Before we start, let's open a terminal and install the required packages:

```
sudo apt update
sudo apt install ros-humble-slam-toolbox
sudo apt install ros-humble-nav2-map-server
```

The first package contains the SLAM algorithm, and the second provides us with the `map_saver_cli` tool to save our map.

4 The 4-Terminal Setup

We need four terminals running simultaneously:

- **Terminal 1 (T1):** Gazebo simulation, which provides the environment and robot physics (publishes `/tf` and sensor data).

- **Terminal 2 (T2):** The SLAM node, which subscribes to `/scan` and `/tf`, and publishes the map (`/map`) and the `map` → `odom` transform correction.
- **Terminal 3 (T3):** The visualization (RViz2), to see the map as it is being built and monitor the robot's status.
- **Terminal 4 (T4):** The teleoperation node, to drive the robot around and explore the environment.

We will launch them in stages for our experiments. Let's set up our first one.

5 Experiment A: The Problem (Visualizing Drift)

First, let's prove why we need SLAM. We will run the robot **without** the SLAM node to see odometry drift in real-time.

1. **Induce drift (IMPORTANT):** By default, Gazebo's simulation is too perfect and its odometry (`/odom`) shows almost no drift. To simulate a more realistic scenario, we must manually edit the robot's model file to introduce wheel slip.
   - Open the `model.sdf` file for the TurtleBot3 Burger. 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/models/
        turtlebot3_burger/model.sdf
     ```

   - Open this `model.sdf` file with a text editor.
   - Search for the `wheel_left_link` and `wheel_right_link` sections. Inside each, find the `<friction>` tags.
   - Change `<mu>100000.0</mu>` to `<mu>10.0</mu>`.
   - Change `<mu2>100000.0</mu2>` to `<mu2>10.0</mu2>`.
   - You must do this for **both** wheels. Save the file. This reduced friction will now cause the wheels to slip, creating the odometry drift we want to observe.
2. **T1 - Launch Gazebo:** Launch the TurtleBot3 world.

   ```
   # Don't forget to set the robot model!
   export TURTLEBOT3_MODEL=burger
   ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
   ```

3. **T3 - Launch RViz2:**

   ```
   rviz2
   ```

4. **T4 - Launch teleop:**

   ```
   export TURTLEBOT3_MODEL=burger
   ros2 run turtlebot3_teleop teleop_keyboard
   ```

5. **RViz configuration for experiment A:**

- In the "Displays" panel, set the **Fixed Frame** to `odom`. This is crucial. We want to see the world from the perspective of the drifting odometry.
- Click "Add" and add a **LaserScan** display. Set the topic to `/scan`.
- Add a **TF** display.

6. **Run the test:**
   - Make T4 (teleop) the active window.
   - Look at your Gazebo window (this is the "ground truth"). Drive the robot out of its starting area, take a large loop around the central obstacles, and drive back to exactly where you started.
   - Now, look at your RViz window.
7. **Observations (answer in your report):**
   - In RViz (Fixed Frame: `odom`), where is the robot relative to its start? Did it return to the origin?
   - Look at the LaserScan data. Are the walls from the start of your run and the end of your run aligned?
   - You have just visualized **odometry drift**.

6 Experiment B: The Front-end (The `map` → `odom` Correction)

Now, let's turn on the SLAM node and see how it continuously fights this drift.

1. **T2 - Launch `slam_toolbox`:**
   - `slam_toolbox` provides a ready-to-use launch file.
   - **VERY IMPORTANT:** We must tell it to use the simulation's clock by passing the argument `use_sim_time:=True`. Otherwise, it may lead to inconsistent timestamps for various messages (e.g., sensor data, odometry, TF, etc.), which would be disastrous for timestamp-sensitive algorithms such as SLAM.

```
# We MUST use sim time, or the timestamps will be inconsistent
ros2 launch slam_toolbox online_async_launch.py use_sim_time:=True
```

2. **RViz re-configuration for experiment B:**
   - In "Displays", change the **Fixed Frame** from `odom` to `map`. This is the most important step. We now want to see the world from the stable, globally consistent `map` frame.
   - Click "Add" and add a **Map** display. It should subscribe to the `/map` topic.
3. **Observe the TF tree:**
   - In the TF display, you will now see three main frames: `map` (from SLAM), `odom` (from odometry), and `base_link` (the robot).
   - Drive the robot around slowly. Pay close attention to the relationship between the frames.
4. **Observations (answer in your report):**
   - Describe the transform `odom` → `base_link`. Is it smooth or jerky?

- Describe the transform `map` → `odom`. Is it smooth or jerky?
- This "jumping" `map` → `odom` transform is the **front-end's correction** (scan-matching). It is constantly publishing the difference between the "drifty" odometry and the "corrected" map pose.

7 Experiment C: The Back-End (Triggering Loop Closure)

The front-end provides local correction, but drift can still accumulate over long distances. Only the back-end can fix this by performing a **loop closure**. Let's trigger one.

1. **Build up drift:**
   - Start driving the robot on a long, exploratory path. Go down a hallway, turn, go down another hallway... explore a large area of the map.
   - **IMPORTANT:** Do not return to a previously visited area yet. As you drive, small front-end errors are accumulating.
2. **Trigger the event:**
   - After exploring a large new area, slowly drive the robot back towards your starting point (or any other area you have already mapped).
   - Watch RViz very closely as your robot's current LaserScan begins to overlap with the old map data.
3. **Observations (answer in your report):**
   - Describe exactly what happens the moment the robot "recognizes" the old area.
   - Did just the local area get fixed, or did the entire map (even the far-away hallways you just mapped) "snap" or "shift" into a more consistent alignment?
   - You have just witnessed a **back-end loop closure**! The graph optimizer just corrected the entire pose-graph based on this new, high-confidence constraint.

8 Saving the Map

Once you are satisfied with your map (i.e., you have explored all areas and all loop closures are resolved), you can save it to the disk.

1. Stop the teleoperation node (Ctrl+C in T4).
2. In that same terminal (or a new, sourced terminal), use the `map_saver_cli` tool we installed.

```
# 'ros2 run <package> <executable> -f <your_map_name>'
ros2 run nav2_map_server map_saver_cli -f my_map
```

3. This command calls a service to fetch the map being published on the `/map` topic and save it.
4. Check your current directory, and you should now have two files:
   - `my_map.pgm`: An image file (Portable GrayMap) showing the map. Black pixels are obstacles, white is free space, and gray is unknown.

- `my_map.yaml`: A metadata file that describes the map's resolution, origin, etc., like this one:

```
image: my_map.pgm
mode: trinary
resolution: 0.050000
origin: [-1.000000, -1.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.25
```

Congratulations! You have not just made a map, but also verified the core components of SLAM: the problem of drift, the continuous correction of the front-end, and the global optimization of the back-end. The you have generated and saved is the cornerstone of autonomous navigation, which we will cover in the next sessions.