Introduction to Mobile Robotics

Practical Work 9: Nav2 Behavior Tree

Zhi Yan

ENSTA, Institut Polytechnique de Paris

1 Objective

In the last practical, you used the Nav2 stack as a "black box" that the exploration node sent commands to. In this session, we will open that black box. The goal is to learn how to configure and modify Nav2. You will:

1. Launch the Nav2 stack to navigate a **known map**.
2. Change the **global planner** (e.g., from NavFn (for Navigation Function) to SmacPlanner (for Steve Macenski)).
3. Create a simple custom **Behavior Tree** (BT) to make the robot perform a "stop-and-spin" action.

2 Background: The Nav2 Stack

Nav2 is not one node. It's a collection of **servers**, each with a job:

- `map_server`: Loads your saved map file and serves it.
- `AMCL`: Localizes the robot on the map (*attention: localization, not SLAM*).
- `planner_server`: Generates the global path (e.g., Dijkstra, A*, etc.).
- `controller_server`: Generates velocity commands to follow the path (local planner).
- `bt_navigator`[1]: Coordinates all the others using a Behavior Tree (the "brain").

We will configure the `planner_server` and the `bt_navigator`.

3 Experiment A: Navigation on a Known Map

First, let's get the robot navigating on the map you saved in Practical Work 7 or 8.

1. **Copy your map**: Find your `my_map.yaml` and `my_map.pgm` files (or `exploration_map.yaml` and `exploration_map.pgm`) and copy them into a new directory, e.g., `~/ros2_ws/src/my_map_pkg/maps/`.
2. **T1 - Launch Gazebo:** We just need the robot and the world.

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

[1]Note: The name of the `bt_navigator` node does not have the `_server` suffix like other nodes.

3. **T2 - Launch AMCL:** We use the `localization_launch.py` launch file in the `nav2_bringup` ROS package, which starts `map_server` and `AMCL`. We need to pass it the path to our map file.

```
# Replace the path with the location of YOUR map file
ros2 launch nav2_bringup localization_launch.py use_sim_time:=True map
    :=/home/[USER]/ros2_ws/src/my_map_pkg/maps/my_map.yaml
```

4. **T3 - Launch navigation:** We use the `navigation_launch.py` launch file, which starts `planner_server`, `controller_server` and `bt_navigator`.

```
ros2 launch nav2_bringup navigation_launch.py use_sim_time:=True
```

5. **T4 - Launch RViz2:**

```
ros2 launch nav2_bringup rviz_launch.py
```

6. **T5 (OPTIONAL) - Launch `twist_converter`:** For some of you, there may be a communication mismatch between Nav2 and the Gazebo simulator as experienced in Practical Work 8. Don't worry, just launch the amazing `twist_converter`.

```
ros2 run ros2_exploration twist_converter --ros-args -p use_sim_time:=
    true
```

7. **Localize the robot:**
   - In Gazebo, the robot starts at (0,0), but AMCL doesn't know this.
   - In RViz2, you might only see a map and nothing else (you won't see those red `/scan` readings, robot TF, etc.).
   - **You must tell the robot where it is.**
   - Click the "2D Pose Estimate" button in the top toolbar.
   - Click and drag an arrow at the robot's true location on the map, pointing in its correct direction.
   - The red scan data should show up to align with the map. The robot is now localized!
8. **Send a goal:**
   - Now, click the "Nav2 Goal" button in the top toolbar.
   - Click and drag an arrow to set a goal pose.
   - The robot should plan a path (red line) and begin to follow it.

4 Experiment B: Changing the Global Planner

By default, Nav2 uses `NavFn` (a Dijkstra-based planner). Let's change it to the modern `SmacPlanner` (an A*-based planner).

1. **Find the params file:** The Nav2 launch file gets its settings from a `.yaml` file. Let's find the default one:

```
# This will show you the path, e.g., /opt/ros/humble
ros2 pkg prefix nav2_bringup

# The file is at:
# [PATH]/share/nav2_bringup/params/nav2_params.yaml
```

2. **Copy and edit:**

   - Copy this `nav2_params.yaml` file into your own package, e.g., `~/ros2_ws/src/my_map_pkg/params/`.
   - Open **your copy** of the file.
   - Search for `planner_server:`.
   - Under this, find `GridBased:`, change the parameters from

```
plugin: "nav2_navfn_planner/NavfnPlanner"
tolerance: 0.5
use_astar: false
allow_unknown: true
```

     to

```
plugin: "nav2_smac_planner/SmacPlanner2D"
tolerance: 0.125
allow_unknown: true
```

   - **Note that** in Jazzy, use "::" instead of "/". For more details, please refer to `https://docs.nav2.org/configuration/packages/smac/configuring-smac-2d.html`.

3. **Relaunch T3 (navigation):** This time, we tell the launch file to use **our** params file instead of the default.

```
# STOP your old T3 first!
ros2 launch nav2_bringup navigation_launch.py use_sim_time:=True \
params_file:=/home/[USER]/ros2_ws/src/my_map_pkg/params/nav2_params.
    yaml
```

4. **Observations (answer in your report):** Send a new goal. Does the path and robot behavior look different? If so, please describe the difference and give the reasons.

5 Experiment C: Designing a Custom Behavior Tree

Now for the most exciting part. We will change the robot's "brain" (i.e., the Behavior Tree) from "go to a goal" to "go to a goal and spin".

1. **Find the BT file:** The default BT is defined in the `nav2_params.yaml` file. Look for `bt_navigator:`, below which you will see:

```
# 'default_nav_through_poses_bt_xml' and 'default_nav_to_pose_bt_xml'
    are use defaults:
# nav2_bt_navigator/navigate_to_pose_w_replanning_and_recovery.xml
# nav2_bt_navigator/navigate_through_poses_w_replanning_and_recovery.
    xml
# They can be set here or via a RewrittenYaml remap from a parent
    launch file to Nav2.
```

   We are going to modify `navigate_to_pose`'s BT. Let's first find `navigate_to_pose_w_replanning_and_recovery.xml` in `[PATH]/share/nav2_bt_navigator/behavior_trees`.
   Question: Can we complete the experiment by modifying `navigate_through_poses`'s BT? If so, how? If not, why?

2. **Copy the BT file:**

- Copy the `.xml` file into your own package, e.g., `~/ros2_ws/src/my_map_pkg/behavior_trees/`.
- Rename it `spin.xml`.

3. **For Humble users:** Open `spin.xml`. Add a `Sequence` of "wait, spin, wait" actions into the "MainTree", as shown below.

```
<root main_tree_to_execute="MainTree">
  <BehaviorTree ID="MainTree">
    <KeepRunningUntilFailure>
      <Sequence>
        <RecoveryNode number_of_retries="6" name="NavigateRecovery">
        ... (leave the original tree here) ...
        </RecoveryNode>

        <Wait wait_duration="2.0"/>
        <Spin spin_dist="3.14"/>
        <Wait wait_duration="2.0"/>
      </Sequence>
    </KeepRunningUntilFailure>
  </BehaviorTree>
</root>
```

4. **For Jazzy users:** Open `spin.xml`. The current main tree is at the top, called `<BehaviorTree ID="MainTree">`. Change the value of ID from `"MainTree"` to `"Navigate"`. Add a new tree including a `Sequence` of "wait, spin, wait" actions at the top of the file, inside `<root>`, as shown below.

```
<!-- THIS IS OUR NEW TREE -->
<BehaviorTree ID="StopSpin">
  <KeepRunningUntilFailure>
    <Sequence>
      <SubTree ID="Navigate" goal="{goal}" _autoremap="true"/>
      <Wait wait_duration="2.0"/>
      <Spin spin_dist="3.14"/>
      <Wait wait_duration="2.0"/>
    </Sequence>
  </KeepRunningUntilFailure>
</BehaviorTree>

<!-- THE ORIGINAL TREE IS STILL BELOW -->
<BehaviorTree ID="Navigate">
  ... (leave the original tree here) ...
</BehaviorTree>
```

   At the very top of the XML file,
   change `main_tree_to_execute="MainTree"` to `main_tree_to_execute="StopSpin"`.

5. **Update `nav2_params.yaml`:** In your `nav2_params.yaml` file, find
   `# They can be set here or via a RewrittenYaml remap from a parent launch file to Nav2.` and add the following line below it:

```
default_nav_to_pose_bt_xml: "/home/[USER]/ros2_ws/src/my_map_pkg/
   behavior_trees/spin.xml"
```

6. **Relaunch T3 (navigation):** Same command as Experiment B.

7. **Observations (answer in your report):** Send a new goal. What is your robot's behavior now that it has stopped? What happens if you send a new goal to the robot again? Can you draw the robot's current behavior tree structure?

Congratulations! You have now configured planners and programmed a custom autonomous behavior by modifying the core Behavior Tree of Nav2.