Lab08

NAV2

Objectives

- Practice with the Nav2 framework
- Set up and launch a complete navigation task in ROS 2
- Assign goal to the robot by the Rviz2 panel, and visualize the computed trajectory
- Use a map of the environment to navigate through it
- Understand the role of navigation sub-systems: planners, controllers, map server, AMCL localization module, costmap

Exercise

The template for this laboratory is available at the following link.

Task 1

In this task, you will use NAV2 without a map.
As a first thing, launch Gazebo with the Turtlebot3 world:

```
$ export TURTLEBOT3_MODEL=burger
$ ros2 launch turtlebot3_bringup gazebo.launch.py
```

Then, you should launch NAV2 without the map using the `navigation.launch.py` launch file:

```
$ ros2 launch lab08_pkg navigation.launch.py
params_file:=<path_to_lab08_pkg>/lab08_pkg/config/nav2_params_odo
m.yaml
```

Then, start RViz with the `rviz.launch.py` launch file:

```
$ ros2 launch lab08_pkg rviz.launch.py
rviz_config:=<path_to_lab08_pkg>/lab08_pkg/rviz/nav2_odom_view.rv
iz
```

In this case, you will use some custom launch files which do not activate the map_server NAV2 plugin. Therefore, the odometry frame `odom` will be considered as your fixed origin (no transformation from `map` → `odom`).

Notice that in the parameters file you are using for this task (at `lab08_pkg/config/nav2_params_odom.yaml`) you are setting odom as global frame in the relevant fields:

```
bt_navigator/global_costmap/behavior_server:
  ros__parameters:
    global_frame: odom
```

Now, assign some goals to the robot through the RViz interface. In the upper bar, select "NAV2 goal" and then drag the arrow directly where you want to assign the goal to the robot in the scene (you can also set a goal orientation by giving the arrow a direction).

! You will notice the local costmap (squared around the robot) and the global costmap, expanded with lidar observations!

Task 2

In this task, you will launch the robot navigation again, but using the Turtlebot3's world **map**. Loading the map of the environment allows you to plan a path from your starting position to the goal, and use Adaptive Monte Carlo Localization (ACML) as the global localization system.

As a first thing, launch Gazebo with the Turtlebot3 world:

```
$ export TURTLEBOT3_MODEL=burger
$ ros2 launch turtlebot3_bringup gazebo.launch.py
```

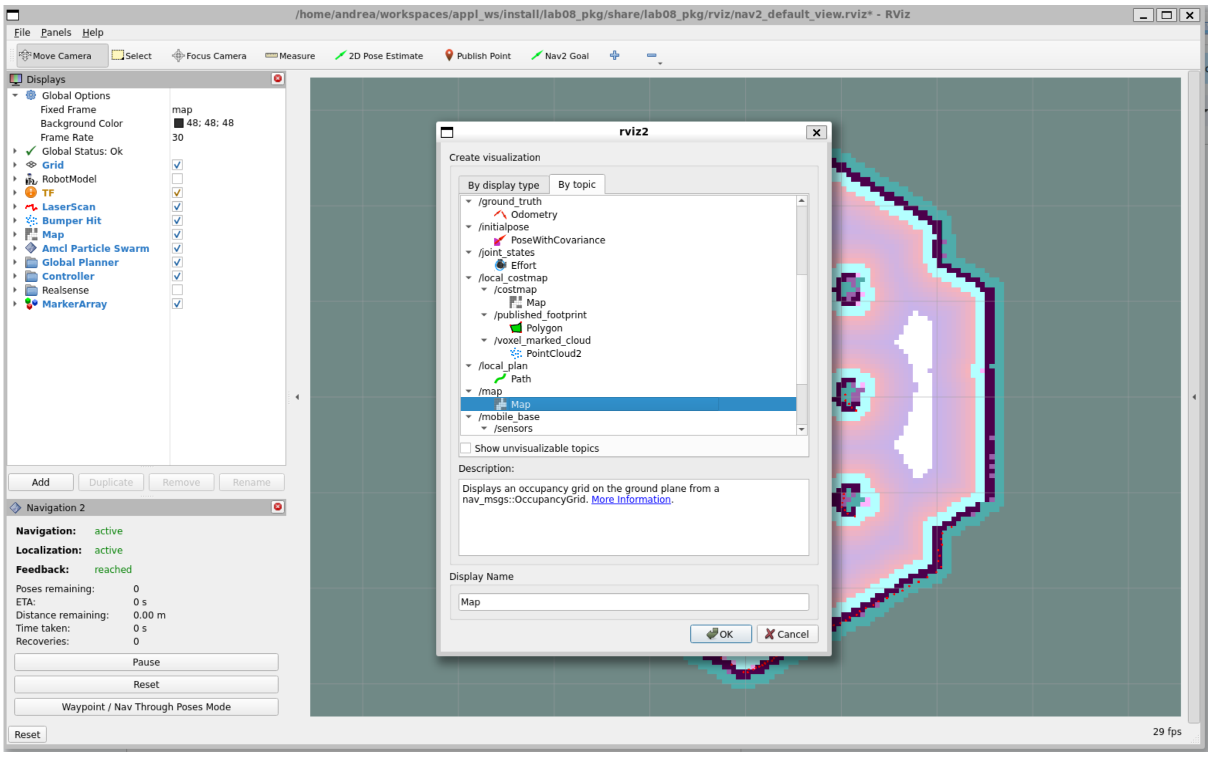
Then, you should launch NAV2 (loading the map) using the **`bringup_launch.py`** launch file:

```
$ ros2 launch lab08_pkg bringup_launch.py
```

Then, start RViz with the `rviz.launch.py` launch file:

```
$ ros2 launch lab08_pkg rviz.launch.py
```

In the RViz window, you will see the world map, or you can add it in this way .. to the scene.



Fixed Frame: map
Background Color: 48; 48; 48
Frame Rate: 30

Navigation: active
Localization: active
Feedback: reached
Poses remaining: 0
ETA: 0 s
Distance remaining: 0.00 m
Time taken: 0 s
Recoveries: 0

Description:
Displays an occupancy grid on the ground plane from a nav_msgs::OccupancyGrid. More Information.

Display Name: Map

In this case, you will have to specify an approximative initial position of the robot to initialize the AMCL system (on the upper task bar, select "2D pose estimation and drag the green arrow at the estimated initial position of the robot). The AMCL will provide an estimate of the global localization transformation `map → odom)`. You will see the particles of the filter around the scene. Try to move the robot assigning it a goal, do you notice any difference?

Task 3

In this Task you will use the Simple Commander API to utilize the NAV2 framework from a simple Python script. You will find the Simple Commander documentation here.

In addition you will find the `example_nav_to_pose.py` and `example_nav_through_poses.py` in the package provided.

The objective is to navigate through the following poses:
[-0.5718, -0.56, 0.0]
[1.803, -1.66, pi/2]
[-0.0, 1.99, pi]
[-1.715, 1.031, -pi/6]

You should use first the **`goToPose()`** function, for each intermediate goal.
Then, use the **`goThoughPoses()`** function and pass to it the full list of poses.

To use the Simple Commander interfaces, you should launch first the NAV2 launch file, then start RViz as you did in the previous tasks and set the initial position.

Finally, you can use the script you created by running it from the terminal:

```
$ ros2 run lab08_pkg example_nav_to_pose
```

Or for the **`goThoughPoses()`** version:

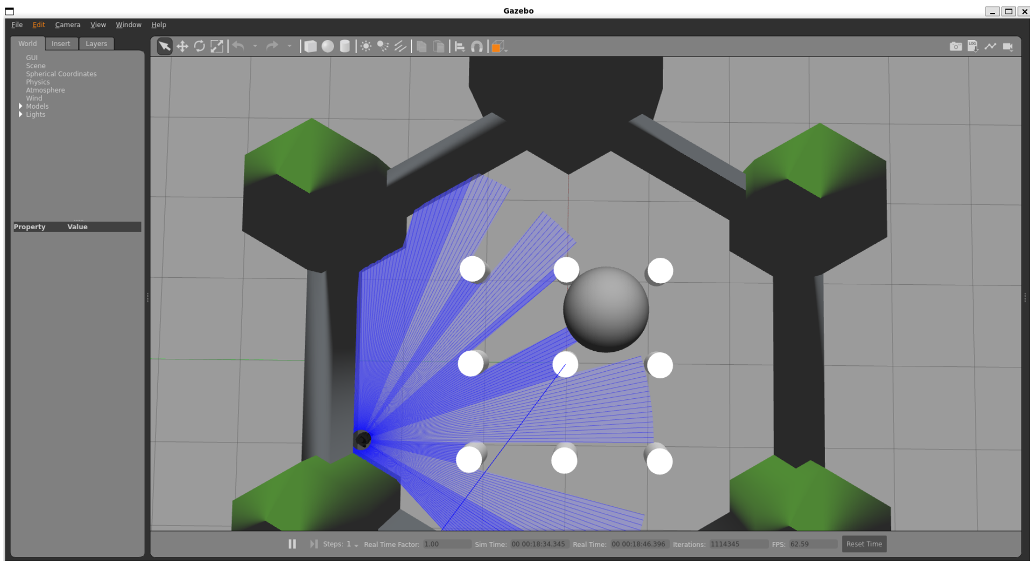
```
$ ros2 run lab08_pkg example_nav_to_pose
```

Task 4

Now, compare different planners, controllers and costmap parameters.
You should change the file `nav2_params_amcl.yaml` modifying the parameters.

1. Understanding **Costmap**
    a. **Local costmap** handles local planning and control costs, therefore it is composed of two layers: `["voxel_layer", "inflation_layer"]`. "voxel_layer" places obstacles costs not present in the map (3D extension of obstacle layer). **Global costmap** also presents the static layer derived from the loaded map: `["static_layer", "obstacle_layer", "inflation_layer"]`. Perform the following modification, and relaunch the nav2 system to see the changes in RViz.

    b. Try to insert a new obstacle in Gazebo. You will find that a new cost will be added to the costmap as soon as it will be visible from the lidar of the robot. The **"obstacle_layer"** is the one responsible for adding costs of obstacles not present in the loaded static map.



    c. First, try to change the size and resolution of the **local costmap** to:
        i. width: 9 (increasing the size augment the local planning horizon)
        ii. height: 9
        iii. resolution: 0.1 (this makes the costmap more coarse but fast)
    d. change the resolution of the **global costmap** to 0.2:

   e. then, try different values of the **inflation radius** in both local and global costmap, in the inflation layer:
      i. inflation radius: 0.3, 0.6, 1.0
2. Practice with **controller**:
   a. try the **DWB**. Parameter tuning is a delicate and critical issue of DWA, the default values should be the optimized ones:
      i. The dynamic window and the granularity of the velocity grid are defined by the kinematic parameters:
         1. max_vel_x: 0.26
         2. max_vel_theta: 1.0
         3. acc_lim_x: 2.5
         4. acc_lim_theta: 3.2
         5. vx_samples: 20 (number of vx samples to evaluate)
         6. vtheta_samples: 20 (number of vtheta samples to evaluate)
         7. sim_time: 1.7 (max sim time to evaluate a velocity set)

      ii. Desired behaviors are expressed in terms of critics: ["RotateToGoal", "Oscillation", "BaseObstacle", "GoalAlign", "PathAlign", "PathDist", "GoalDist"] and significant scalar costs:
         1. BaseObstacle.scale: 0.02
         2. PathAlign.scale: 0.0
         3. GoalAlign.scale: 0.0
         4. PathDist.scale: 32.0
         5. GoalDist.scale: 24.0
         6. RotateToGoal.scale: 32.0

   b. try the **Regulated Pure Pursuit** to follow the global path. Comment the DWB controller server params and substitute it with the controller server parameters of the Regulated Pure Pursuit (you find the code snippet here)
      i. you may try to change some parameters:
         1. desired_linear_vel: 0.26
         2. lookahead_dist: 0.3, 1.2
   c. **Planner** server:
      i. the **Navfn Planner** plugin implements a wavefront Dijkstra or A* expanded holonomic planner. Enable A* by changing:
         1. use_astar: true
      ii. You can try other variants of the A* planner that take into account kinematics constraints of the robot by changing the plugin with:
         1. 2D Smac Planner (2D A*) (code snippet at the page bottom)
         2. Hybrid A*
         3. Theta* (Any-angle planner, already commented in the params file)

Hint (optional): you can use this Nav2 package in your Localization Project to move the TurtleBot3 in the map (instead of using the teleop keyboard!).