Bonus Worksheet: Robot Navigation Using ROS2

Universität Bremen
FB 3 – Mathematik und Informatik
Arbeitsgruppe Robotik
Prof. Dr. Dr. h.c. Frank Kirchner
Dr. rer. nat. Teena Hassan
M. Sc. Mihaela Popescu

Deadline: 22.12.2022
Robot Design Lab
03-IBGA-FI-RDL
WiSe 22-23

1 Quiz on ROS 2 Navigation Stack [5%]

The ROS 2 Navigation stack is a highly useful bundle of modules needed for autonomous navigation in mobile robots. It contains several packages and each package contains multiple nodes with many configurable parameters. In this task, you will use the documentation of the ROS 2 Navigation Stack[1] to understand some of the important parameters that are needed for path planning and obstacle avoidance.

1. **Path Planner:** Which path planning algorithms does the `Navfn Planner plugin` support? How can you select which of these algorithms should be used? [1%]
   Hint: Check the documentation here:
   https://navigation.ros.org/configuration/packages/configuring-navfn.html.

2. **Path Planner:** Why does the `Navfn Planner plugin` include a parameter called `tolerance`? [1%]
   Hint: Check the documentation here:
   https://navigation.ros.org/configuration/packages/configuring-navfn.html.

3. **Dynamic Window Approach:**

   (a) How would you set the following `DWB plugin` parameters for your TurtleBot3 Burger robot? Justify your answer. [1%]

   - `max_speed_xy`
   - `max_vel_theta`

   Hint: Check the following link:
   https://navigation.ros.org/configuration/packages/dwb-params/kinematic.html.

   (b) Plot the velocity space based on the values you set for the above parameters. [1%]

4. **Dynamic Window Approach:** In the lecture, we learned about three criteria that are used by the objective function to evaluate each candidate trajectory. In ROS 2 Navigation stack, these criteria are referred to as `Critics` and there are several types of `Critics`. Explain how `TwirlingCritic` and `PreferForwardCritic` would influence the choice of trajectories by the dynamic window based approach. [1%]
   Hint: You can see the list of critics here:
   https://navigation.ros.org/configuration/packages/configuring-dwb-controller.html.

2 Robot Navigates to Goal [5%]

In this task, you will let the robot plan a path from its current position to a chosen goal position in a pre-defined map using the ROS 2 Navigation stack.

2.1 Setup, Build, and Launch Navigation Nodes

Please follow the instructions given below, in order to set up your ROS2 workspace for navigation:

1. Download and unzip the `worksheet04-path-planning.zip` file uploaded in Stud.IP. Use the `.zip` file corresponding to the ROS 2 version that you are using on your machine.

2. Copy the folder `worksheet04-path-planning` and its contents to `~/rdl_ws/src/`.

[1] https://navigation.ros.org/

3. Now you have a ROS 2 package named `worksheet04-path-planning` in your workspace. Inside this package, there is a folder named `map` containing the files `map.pgm` and `map.yaml`. Replace these files with the map you created for your arena in Worksheet03.
4. Open the `burger.yaml` file in the folder named `param` and adapt the path for the parameter `default_nav_to_pose_bt_xml`.
5. Build the package using `colcon`:

   ```
   cd ~/rdl_ws
   colcon build --packages-select worksheet04-path-planning
   ```

6. Source the local and global `setup.bash` files.

   ```
   source /opt/ros/humble/setup.bash
   source install/setup.bash
   ```

7. Now launch the ROS 2 navigation nodes:

   ```
   ros2 launch worksheet04-path-planning navigation2.launch.py
   ```

8. If the launch was successful, you should now see `rviz2` loaded with a pre-defined map.
9. If the launch failed, then you might need to install the ROS2 navigation packages on your machine:
   - For ROS2 Humble on Ubuntu 22.04.:
     `sudo apt install ros-humble-navigation2 ros-humble-nav2-bringup ros-humble-turtlebot3*`
   - For ROS2 Foxy on Ubuntu 20.04.:
     `sudo apt install ros-foxy-navigation2 ros-foxy-nav2-bringup '~ros-foxy-turtlebot3-.*'`
10. Connect to your robot and launch the `bring_up` nodes:

    ```
    ssh ubuntu@192.168.1.[30 + turtlebot_ID]
    # enter password to log in (Password: robotics)
    ```

11. On the robot, add the following two lines to `~/.bashrc`:

    ```
    export ROS_DOMAIN_ID=30 + turtlebot_ID
    export TURTLEBOT3_MODEL='burger'
    ```

12. After this, launch the bring up nodes on the robot:
    `ros2 launch turtlebot3_bringup robot.launch.py`

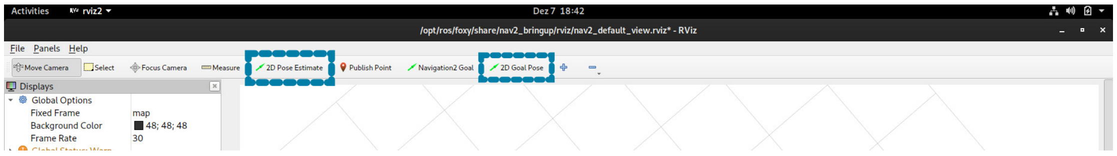
2.2 Set Initial Pose

To set the **initial pose** of the robot via `rviz2`:

1. Place the robot at a location of your choice.
2. Look for the button `2D Pose Estimate` at the top of `rviz2` GUI. If not found, then add it as follows:
   - Press the '+' button at the top of `rviz2` GUI.
   - In the window that opens, click on `SetInitialPose` and then click `OK`.
   - Now, the button named `2D Pose Estimate` will appear at the top (see Figure 1).
3. Click on `2D Pose Estimate` and then click and drag on the map to indicate the approximate current position and orientation of the robot.
4. The selected initial pose will be published on the topic `/initialpose`. If the navigation nodes are running, then you will see the global and local costmaps and the localization particles around the current location of the robot.
5. If you pick up and move the robot manually to a new location, then you should reinitialise the current pose of the robot. For this you should first do:

   ```
   ros2 service call /reinitialize_global_localization std_srvs/srv/Empty
   ```

   After this, you can use the `2D Pose Estimate` button in `rviz2` to specify the current pose of the robot.



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

Abbildung 1: rviz2 header after adding buttons to set initial and goal poses (highlighted in blue).

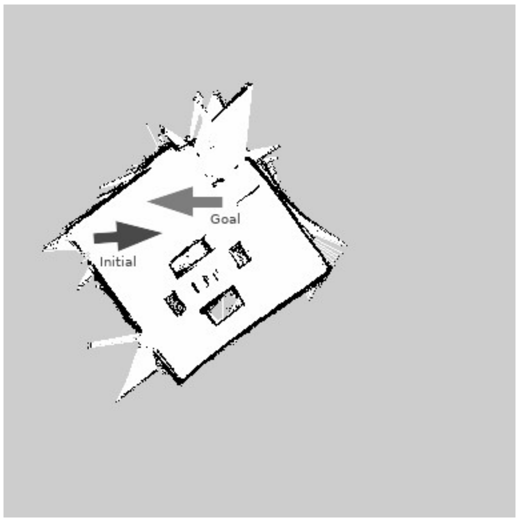


Abbildung 2: Example of initial and goal poses.

2.3 Set Goal Pose

To set the **goal pose** of the robot via `rviz2`:

1. Decide on a goal pose to which you would like the robot to navigate.
2. Press the '+' button at the top of `rviz2` GUI. In the window that opens, click on `SetGoal` and then click `OK`. Now, a button named `2D Goal Pose` will appear at the top (see Figure 1).
3. Click on `2D Goal Pose` and then click and drag on the map to indicate the desired goal position and orientation.
4. The selected goal pose will be published on the topic `/goal_pose`. If the navigation nodes are running, then you will see the planned path from initial pose to goal pose. If the connection is reliable, the robot would start driving to the goal pose along the planned path.
5. After the robot has driven to the goal, you will see an output message from the `controller_server` that says `[controller_server]: Reached the goal!` Now you can choose another goal pose for the robot by clicking the `2D Goal Pose` button in `rviz2`.

2.4 Plan a Path to Goal [5%]

1. Choose an initial pose and a goal pose as described in Sections 2.2 and 2.3.
2. **Take a screenshot** of `rviz2` showing the global and local costmaps as well as the planned path from initial to goal pose. Attach the screenshot in your solution. [0.5%]
3. On the screenshot that you took, **label** (i) the local costmap, (ii) the global costmap, (iii) the current location, (iv) the goal location, and (v) the planned path. [2.5%]
4. What are global and local costmaps used for? [2%]

2.5 Drive to Goal

As soon as the planned path appears on `rviz2` GUI, the robot should drive automatically to the goal pose. However, due to communication delays as well as unreliabilities in the navigation stack, the robot might fail to navigate. If it fails, you can try to set the goal pose once again, or relaunch `navigation2.launch.py`. If your robot navigates, then observe the following in `rviz2` GUI:

- Local costmap moves as the robot moves.
- Current position of the robot (shown by the moving 3D axes) changes as the robot moves.
- Actual path of the robot is close to the planned path, but does not always overlap.
- Robot slows down and attempts to navigate around a new obstacle that is placed on its planned path.
- Such new obstacles are now visible in the local costmap.
- If the robot cannot progress to the goal after multiple attempts, the plan is aborted and pre-defined recovery behavior is initiated.
- The robot rotates to the goal orientation only when it is close to the goal position. This effect is caused by the `RotateToGoalCritic` of the `DWBLocalPlanner`. Take a look at the `burger.yaml` file in the `param` folder of the given `worksheet04-path-planning` package, to understand how these critics are set.

3 Submission Procedure

- Please use the LaTeX template provided in *StudIP/Wiki* to write your solutions. Upload the PDF file as a .zip file in StudIP.
- The naming style of your submission should follow the pattern **Gxx_Bonus_lastname1_lastname2_lastname3.zip**, where *xx* stands for the group number and 'Bonus' stands for the bonus worksheet.