[Lab 3]

SLAM configuration and Navigation

Resources

[1] GitHub of the lab: https://github.com/emarche/Mobile-robotics-4S009023-UniVR

Requirements

- Setup of L1 and L2

SLAM methods

There are different algorithms/packages for the SLAM:

- Gmapping
- Cartographer
- Hector
- Karto

While Gmapping is installed by default, the other SLAM methods require additional dependent packages and are out of scope for the course. You can find additional information on the wiki.

Gmapping

We will use Gmapping as it is the standard method used by the built in SLAM node.

It has several parameters to change performances in different environments. An exhaustive list is available in the wiki.

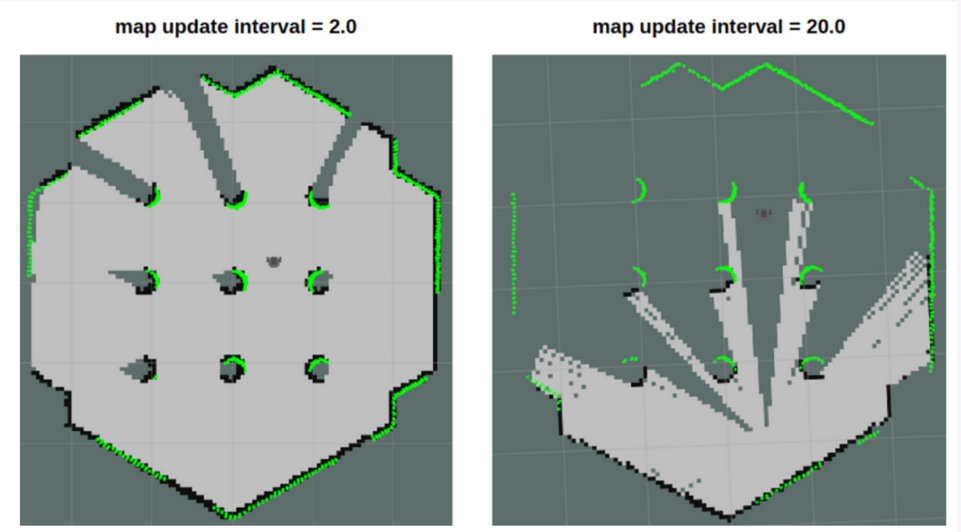
Here we will give a brief overview on the most important ones.

Gmapping tuning

These parameters are defined in the turtlebot3 package:

`turtlebot3_slam/config/gmapping_params.yaml`

- maxUrange: maximum usable range of the lidar (measured in meters)
- max_update_interval: time between updating the map



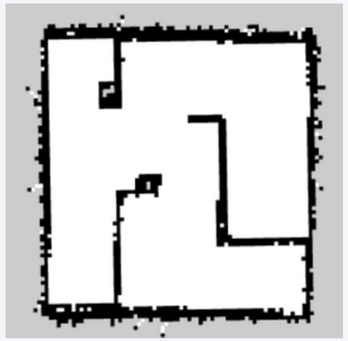
Gmapping tuning

- linearUpdate: scan after translating longer than this value.
- angularUpdate: similar to the above one, but for the angular velocity.
- temporalUpdate: process the scan if the last one is older than this value (in seconds)

The map is then based on the robot's odometry, tf, and scan information and it is drawn in RViz while the robot is travelling. Remember that we can save the resultant map:

```
$ rosrun map_server map_saver -f ~/map
```

The map is stored as a 2d Occupancy Grid Map, which is commonly used in ROS.



Navigation

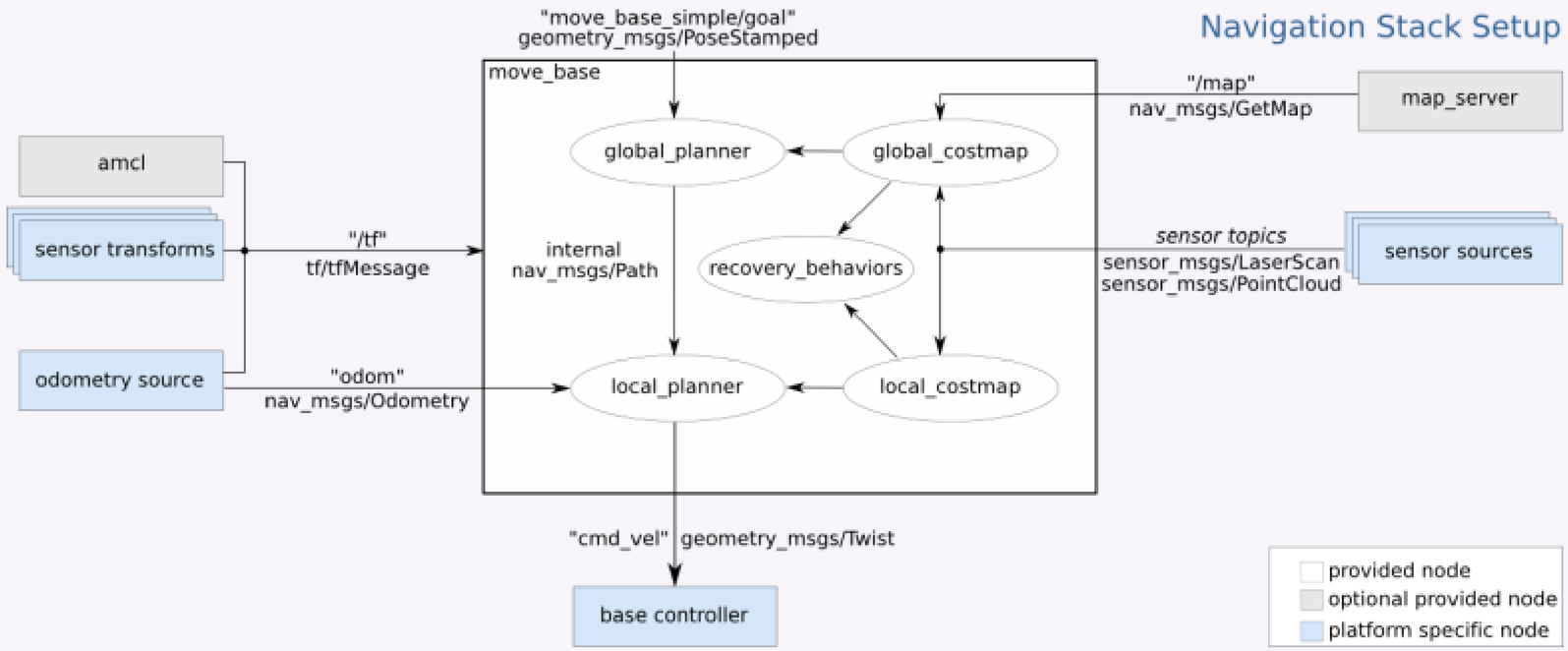
The Navigation Stack takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base.

Requirements:

- It is meant for both differential drive and holonomic wheeled robots only.
- It Requires a planar laser which is used for map building and localization.

It was originally developed on a square robot, so its performance will be best on robots with a similar shape. It also works with arbitrary shapes, but it may have difficulty with large rectangular robots in narrow spaces.

Navigation Setup



The navigation stack assumes that the robot is configured as in the above diagram:

- White components are mandatory and are already implemented
- Gray components are optional and are already implemented
- Blue components must be created for each robot platform.

Navigation

In our scenario, the blue components are provided by sensor readings, which are already published in the corresponding topics.

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml
```

In order to launch the navigation node, remember to specify the map of the environment (i.e., the one we saved with SLAM)

Launch Navigation

If we open the navigation launch file, we can analyze the dependencies of the stack, i.e., the different components of the previous diagram.

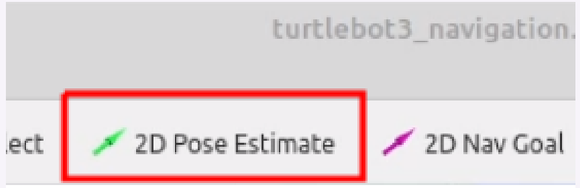
```
<!-- AMCL -->
<include file="$(find turtlebot3_navigation)/launch/amcl.launch"/>

<!-- move_base -->
<include file="$(find turtlebot3_navigation)/launch/move_base.launch">
  <arg name="model" value="$(arg model)" />
  <arg name="move_forward_only" value="$(arg move_forward_only)"/>
</include>
```

In particular, AMCL is used for localization, and move_base for the planning and to provide the actual motor velocities.

Estimate the Pose

First of all, we have to initialize the AMCL parameters by estimating the initial robot position in RViz. In particular, the TurtleBot3 has to be correctly located on the map and the scan data should overlaps the displayed map.



- Click on the 2D Pose Estimate button
- Click on the map where the actual robot is located and drag the large green arrow toward the direction where the robot is facing.
- Launch the teleoperation node to precisely locate the robot by moving back and forth a bit to collect the surrounding environment information.

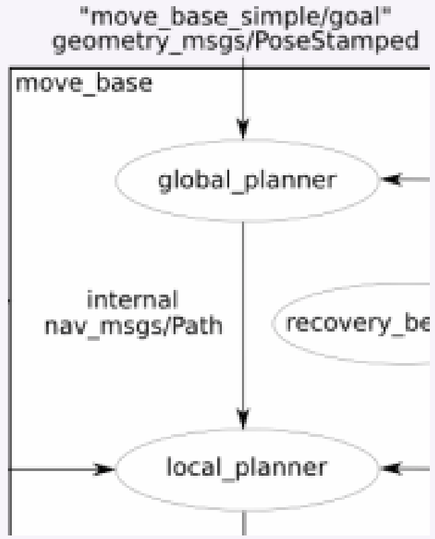
Set the Goal

We can now specify the desired target position for the robot.

- Click on the 2D Nav Goal button
- Click on the map to set the destination of the robot and drag the green arrow toward the direction where the robot will be facing.

Planners

Both the AMCL and the move_base node have different parameters to tune the navigation and perception of the robot.



Today we focus on the move_base node, which is responsible for the planning of the robot.

The move_base package provides an implementation of an action that, given a goal in the world, will attempt to reach it. The move_base node links together a global and local planner to accomplish its global navigation task.

Planners

We can specify different local/global planners by setting the corresponding parameter in the move_base.launch file (which is inside the navigation package).

For example, here we specify to use the dwa_local_planner node that implements the Dynamic Window Approach to local TurtleBot3 navigation on a plane.

```
<!-- move_base -->
<node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
  <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
```

For more informations about global and local planners, we refer to the official wiki.

Exercise

Practice with the navigation node both in the turtlebot3_world environment and in your custom environment.

In the next lesson, we will see the tuning of different parameters and how they influence the actual navigation, and implement a simple local planner with rospy.