ROS-I Basic Training "Mobility"
ROS Navigation Tutorial

Instructor: Nicolas Limpert

August 9, 2017

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Contents

1 Introduction

This tutorial is aimed to provide an approach to the ROS Navigation-Stack to be used on the KUKA YouBot. It introduces the very basic concepts of the ROS Navigation-Stack by focusing on the move_base as well as the most interesting configuration parameters for the costmap (occupancy grid) and the teb_local_planner.
The configuration samples and files will only cover essential values. Although the configuration of ROS Navigation serves many different parameters which can lead to a better performance. As usual, please refer to the ROS Wiki for further documentation:

- http://wiki.ros.org/move_base
- http://wiki.ros.org/costmap_2d
- http://wiki.ros.org/teb_local_planner

Further notes:

- Lines beginning with $ are terminal commands
- Lines beginning with # indicate the syntax of the commands
- The symbol ↪ represents a line break.

2 Requirements

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

To fullfill the given task the move_base node needs:

- Map (`nav_msgs/GetMap`):
  Anything that provides a map (e.g. **gmapping** or **hector_slam** or a static map served by the `map_server`).

- Odometry (`nav_msgs/Odometry`):
  To calculate the next motion commands according to the difference of the robot to the current global waypoint (**hector_slam** works fine for that purpose too, although a real odometry would be desired).

- Localization (`tf/tfMessage`):
  To be aware of the robot's position. A package like AMCL provides a robust approach to localization. In particular, we need a relation between a robot frame (e.g. base_footprint) and a reference frame (e.g. odom). These frames have to be configured in the costmap parameters too.

- Sensory (`sensor_msgs/LaserScan`):
  In order to fill the costmaps with information we want to make use of sensory information gathered by a laser scanner. The user can choose different sensor types to incorporate into the particular scenario (e.g. PointCloud data or even heat information).

- Mobile Base (`geometry_msgs/Twist`):
  Interface of the Navigation-Stack to send motion commands. This is done by sending messages that provide linear and angular velocities.
  Roughly speaking you need a base which executes the motion commands.

3 Setup

This section aims at creating a new package to enable navigation for the KUKA YouBot.

3.1 Depending packages

Make sure to checkout the following:

```
$ cd <your_ws>/src
$ git clone https://github.com/rst-tu-dortmund/teb_local_planner.
  ↪ git
```

```
$ git clone https://github.com/rst-tu-dortmund/costmap_converter.
  ↪ git
$ git clone https://github.com/nlimpert/teb_local_planner_youbot.
  ↪ git
$ rosdep install teb_local_planner
$ rosdep install costmap_converter
$ cd ..
$ catkin_make
```

These packages are required as they serve the motion planning used for the Youbot within the move_base.

3.2 Launchfile

Have a look into the package called `teb_local_planner_youbot`. In there you will find a set of configuration parameters that need to be set up.
Within this tutorial you should look into the following files. Note that everything you need to set up is marked in < > symbols. An explaination for the different files can be found in the following section.
The files of interest are the following ones:

- `costmap_common_params.yaml`
- `global_costmap_params.yaml`
- `local_costmap_params.yaml`
- `move_base_params.yaml`
- `teb_local_planner_params.yaml`

3.3 Configuration

3.3.1 costmaps

The local and global costmaps require configurations. These are stored in three config files. Keep in mind that you can freely exchange the following configuration values between the global and local costmap settings.

1. **costmap_common_params.yaml**

   - footprint:
     This is a set of points defining the footprint of the robot.
   - transform_tolerance:
     Stop the robot for safety reasons in case transform information is not accurate.
   - observation_sources:
     Depending on the sources one would like to incorporate in setting the occupancy grid, one has to set this option. For example, we would like to incorporate the data gathered by a laser scanner. Beneath the laser scanner, we might make use of a PointCloud as we can't detect chairs just with a laser scanner. Make sure that you set up the correct frame for the laser and the correct topic name.
   - inflation_radius:
     Max. distance from an obstacle at which costs are incurred for planning paths.
   - cost_scaling_factor:
     Exponential rate at which the obstacle cost drops off. You might want to try different values in order to get different distances kept by the global planner from obstacles.
   - resolution:
     Cell size in meters. This value has a high influence in the computational load on either updating the costmap information or generating local or global paths. The lower this value the higher the computational load. Although setting this value too high probably results in a waste of space (e.g. a resolution of 0.05 results in a loss of space of up to 0.1m in the worst case).

2. **{global,local}_costmap_params.yaml**

   - global_frame:
     Fixed frame in order to track motion to the robot_base_frame. This will either be /map or /odom, based on your mapping setup.
   - robot_base_frame:
     Frame on the robot in order to track motion to the global_frame. This is usually base_footprint.
   - {width,height}:
     The size of the costmap. Like the resolution, these value have a high influence in computational load as depending on your sensory setup the costmap has to perform accurate cell updating including raytracing of laser scan and / or pointcloud information.

3.4 teb_local_planner

As mentioned before, the teb_local_planner is the package of choice in this navigation-tutorial. Sidenote: Please also have a look at http://wiki.ros.org/teb_local_planner/Tutorials in order to get a complete set of tutorials for robot navigation using *teb_local_planner*.
In combination to the global_planner we can make use of a fast globally planning approach and an approach that locally incorporates kinematic constraints. The global_planner does not require further configuration and should work out-of-the-box. The local_planner needs a proper configuration.

1. **teb_local_params.yaml**

   - TebLocalPlannerROS/dt_ref:
     This value determines the metric difference between the states of the planned path. Increase this to get a better performance.
   - TebLocalPlannerROS/max_global_plan_lookahead_dist:
     This value tells the local planner what point on the global path is to be taken as the reference point for the local planner.

- TebLocalPlannerROS/feasibility_check_no_poses:
  This value takes the given number of poses from the robots position and checks them for feasibility w.r.t the defined footprint.
- TebLocalPlannerROS/max_vel*:
  These values determine the particular velocities. They are important to let the scan matcher / mapping process keep up.
- TebLocalPlannerROS/acc_lim*:
  These values determine the particular acceleration limits. They are only meant for optimization purposes and do not directly smoothen velocities.
- TebLocalPlannerROS/footprint_model:
  Next to the footprint defined in the costmap the teb_local_planner makes use of an own definition of a footprint. This is done to lower the computational effort required to perform planning. In our case a line fits, that describes the metric difference from rear to front axle. You might also try a setup with two_circles. It might allow the planner to obtain better paths.
- TebLocalPlannerROS/*goal_tolerance:
  The goal tolerances set in meters and rad. One can tune this value to prevent the robot from oscillating but keeping a as high as possible precision.
- TebLocalPlannerROS/min_obstacle_dist:
  The minimum distance to be kept from obstacles. This helps in order to get a smoother overall behaviour but it can also lead to complications in narrow environments.
  Note: This value is quite important because the YouBot base sometimes cannot achieve the desired steering angles / rotational velocity that the navigation stack wants to be executed.
  You might want to tweak this value in order to get a good performance in narrow environments.
- TebLocalPlannerROS/weight*:
  It is possible to tweak weights in order to tune the overall optimization behaviour of

the teb_local_planner.

4 Start the navigation

Important: Make sure you have a working localization running as described in the localization tutorial.
In order to start the navigation simply launch the move_base.launch.

4.1 Sending simple navigation goals

To send a simple navigation goal to the move_base you have two options:

1. Use the button "2D Nav Goal" in RViz
2. Publish a ROS message of type geometry_msgs/PoseStamped via the topic move_base_simple/goal.

4.2 Reloading of parameters

Once you set up your new parameters you have to delete the former values from the parameter server by entering the following command:

```
$ rosparam delete /move_base_node
```

Afterwards you have to restart the move_base, so quit the running instance of move_base by pressing CTRL+C in the terminal where move_base is running and simply repeat its execution afterwards.

5 Navigation tuning

After successfull setup of the navigation stack it makes sense to try out the performance of your current setup. This can be done by sending simple navigation goals in RViz.

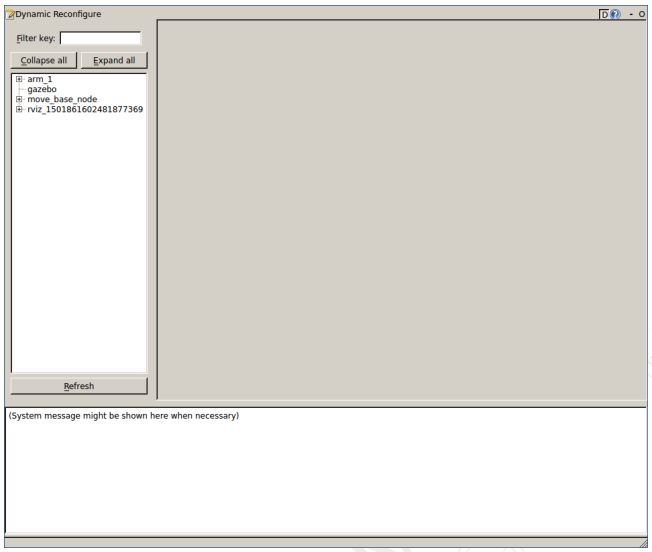

Figure 1: rqt_reconfigure after startup

The Navigation Stack makes heavy use of dynamic_reconfigure which allows to dynamically setup certain parameters.
The `rqt`-tools provide a client for this:

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

A window should popup which looks similar like shown in figure 1. On the left you'll notice the several nodes providing an instance of DynamicReconfigureServer (and are thus dynamically reconfigurable). Try to find out which parameters can be setup by exploring the rqt_reconfigure tool.