Exercise Session 1

Theory

- ROS architecture
- ROS master, nodes, and topics
- Console commands
- Catkin workspace and build system
- Launch-files

Exercise

Get to know ROS by inspecting the simulation of a Super Mega Bot (SMB) robot.

1. Setup the SMB simulation:
   Download the smb_common zipped folder on the course website. Unzip it and place it in the ~/git folder. Navigate into ~/Workspaces/smb_ws/src and make a symlink. Compile the smb_gazebo package with catkin.

2. Launch the simulation with roslaunch and inspect the created nodes and their topics using (Lecture 1 Slides 11/12):
   ```
   rosnode list
   rostopic list
   rostopic echo [TOPIC]
   rostopic hz [TOPIC]
   rqt_graph
   ```

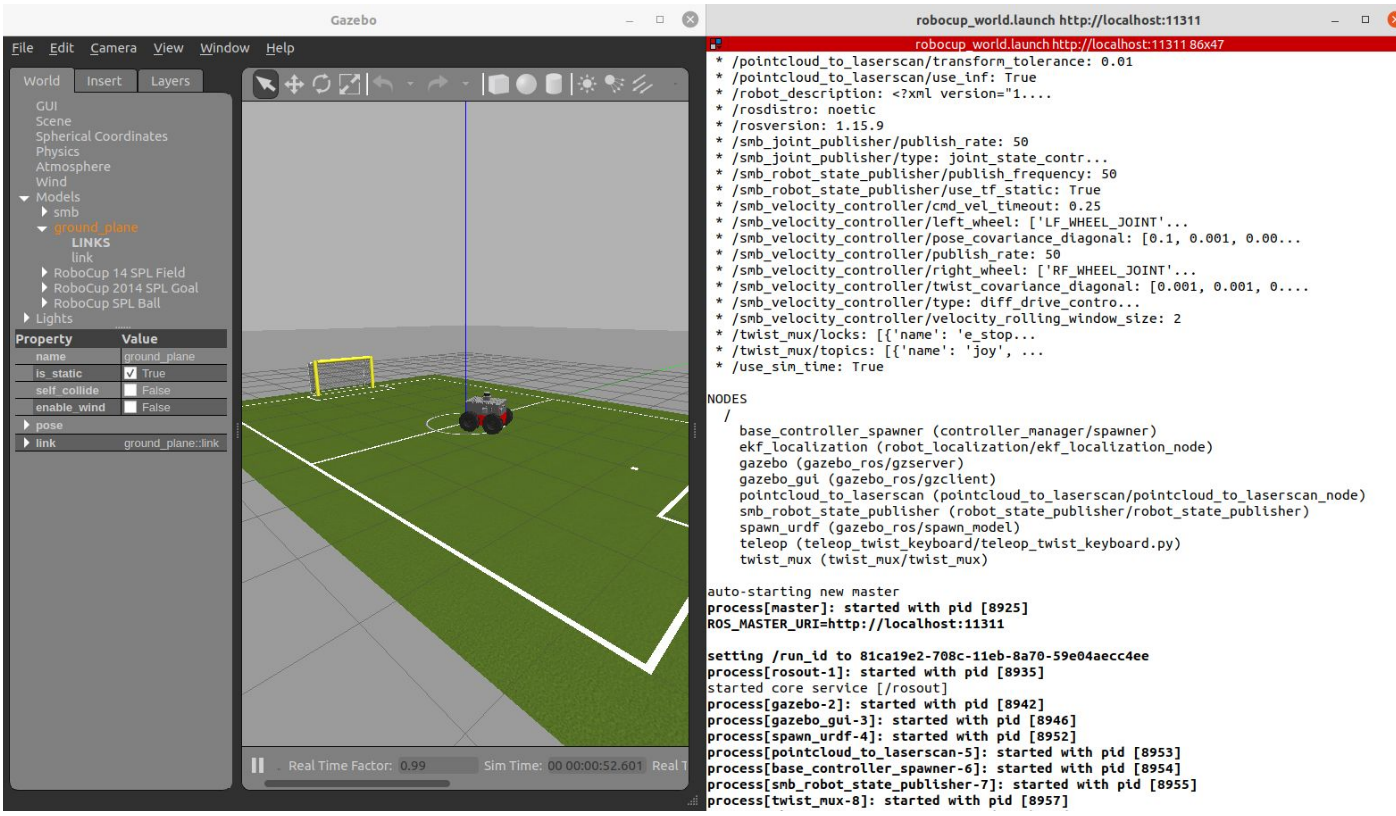
   For more information take a look at the slides or:
   http://wiki.ros.org/rostopic
   http://wiki.ros.org/rosnode

3. Command a desired velocity to the robot from the terminal (`rostopic pub [TOPIC]`) (Lecture 1 Slide 13)

4. Use **teleop_twist_keyboard** to control your robot using the keyboard. Find it online and compile it from source! Use `git clone` to clone the repository to the folder ~/git. (Lecture 1 Slides 22-26)
   For a short git overview see:
   http://rogerdudler.github.io/git-guide/files/git_cheat_sheet.pdf

5. Write a launch file with the following content (Lecture 1 Slides 27-30):
   - smb simulation with a different world:
   Include `smb_gazebo.launch` file and change the `world_file` argument to a world from the directory `/usr/share/gazebo-11/worlds` (e.g. `worlds/robocup14_spl_field.world`). This might take a little while to load

the first time. Note that the world_name is with respect to `/usr/share/gazebo-11/`



Left: Gazebo with Robocup14 World, Right: First lines of output when starting the launch file you have to set up

Evaluation

- ❑ Check if teleop_twist_keyboard is compiled from source (`roscd teleop_twist_keyboard` should show the `smb_ws` folder) [40%]
- ❑ Start the launch file. This should bring everything up that's needed to drive SMB with the keyboard as shown in the above image. [60%]

Hints

- If the robot stops again after sending the velocity command, specify the rate of the publisher. Check out `rostopic pub --help`.

Exercise Session 2

Theory

- ROS package structure
- Integration and programming with Eclipse
- ROS C++ client library (roscpp)
- ROS subscribers and publishers
- ROS parameter server
- RViz visualization

Exercise

In this exercise, you will create your first ROS package. The package should be able to subscribe to a laser scan message from the SMB robot and process the incoming data. This node will be the basis for the next exercises. Use Eclipse to edit your package (Lecture 2 Slides 9-13).

Make sure to look at the ROS template for reference https://github.com/leggedrobotics/ros_best_practices. It will help you a lot for the implementation, as it has a similar node to what you have to do in this exercise!

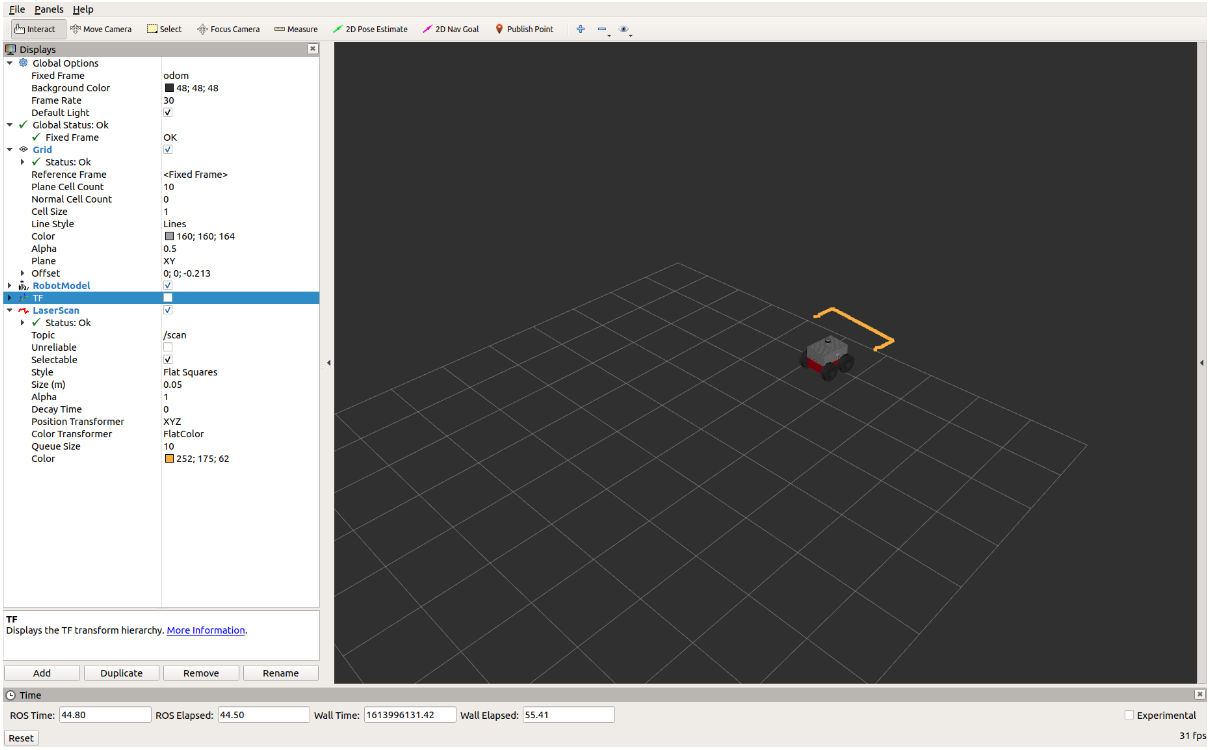
1. **OPTIONAL** (more difficult): Create the package `smb_highlevel_controller` from scratch. You can use the command `catkin_create_pkg` to create a new package with the dependencies `roscpp` and `sensor_msgs`.

2. **OR** (easy): Download the Zip archive containing prepared files of the package `smb_highlevel_controller` from the course website.

3. Inspect the `CMakelists.txt` and `package.xml` files. (Lecture 2 Slides 5-7)
4. Create a subscriber to the `/scan` topic. (Lecture 2 Slides 19-21)
5. Add a parameter file with topic name and queue size for the subscriber of the topic `/scan`. (Lecture 2 Slides 22-23)
6. Create a callback method for that subscriber which outputs the smallest distance measurement from the vector ranges in the message of the laser scanner to the terminal. Inspect the message type here http://docs.ros.org/en/api/sensor_msgs/html/msg/LaserScan.html
7. Add your launch file from Exercise 1 to this package and modify it to:
    - run the `smb_highlevel_controller` node.
    - load the parameter file.
8. Pass the argument `laser_enabled` from your launch file to the `smb_gazebo.launch` file with value `true`.
9. Show the laser scan in RViz and add RViz to your launch file. Make sure to set *odom* as the *Fixed Frame* (under *Global Options*) and adapt the size of the laser scan

points. You can save your current RViz configuration as the default configuration by pressing ctrl+s. (Lecture 2 Slides 24-26)

10. [OPTIONAL] Check the *pointcloud_to_laserscan* node, find out what it is doing. Which topic is it publishing on and which is it subscribing on? Visualize the 3D point cloud and the laser scan in Rviz.
11. [OPTIONAL] Create an additional subscriber to the 3D point cloud and print how many points it has.



RViz visualization of a single laser scan. Multiple obstacles are placed around the robot. Note the changed “Fixed Frame” as well as “Size (m)”.

Evaluation

- ❏ Start the launch file and drive around with SMB. There should be changing output from the laser scanner in the terminal. [40%]
- ❏ Check if the node is implemented as the template suggests. [30%]
- ❏ Is a parameter file used? [15%]
- ❏ Is the laser scan visualized in RViz as shown in the image? [15%]

OPTIONAL

- ❏ Correctly explain what *pointcloud_to_laserscan* node is doing. Is the 3D point cloud changing as the robot moves? [10% bonus]
- ❏ Is the number of points inside the cloud shown in the terminal? Is it the callback implemented correctly? [10% bonus]