CSC_5RO14_TA – Session 2 - Exo1

ROS Theory

- ROS package structure
- Integration and programming
- 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, in the end, be able to subscribe to a laser scan message from the Husky robot and process the incoming data. This node will be the basis for the next exercises.

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

1. **Install** the husky-desktop package:
   ```
   sudo apt update
   sudo apt install ros-noetic-husky-desktop
   ```

   In your *~/.bashrc* file, at the end, add the line to add a lidar on the Husky robot:
   export HUSKY_LMS1XX_ENABLED=1

   This line will be taken into account if you launch a new terminal or if you run:
   `source ~/.bashrc`

   Download the zip archive containing the prepared files of the `5ro14_husky_controller` package from the following address:
   https://perso.ensta-paris.fr/~battesti/5ro14_download/session_2/5ro14_husky_controller.zip

2. **Install** the package into your catkin workspace:
   unzip the folder in the *catkin_ws/src* folder
   cd ~/catkin_ws
   catkin_make
   source ~/catkin_ws/devel/setup.bash

   You will need the `teleop_twist_keyboard` package, as in the exercise of the session 1.
   If necessary, install it in the *~/catkin_ws/src* folder, with the command :
   `git clone https://github.com/ros-teleop/teleop_twist_keyboard.git`

The package should compile without errors.

3. **Inspect** the `CMakelists.txt` and `package.xml` files of the `5ro14_husky_controller` package.
   Inspect the source code.
   For the moment, the node doesn't do anything.
   MyNode.cpp contains the 'main' function which:
   - creates the node named ***5ro14_husky_controller_node***,
   - then creates an instance of ***MyController***,
   - then uses the blocking function ***ros::spin()***.

   MyController.cpp will contain later the details of the node. We'll add to it in the following questions:
   - the subscription to a topic
   - the callback to the topic
   - the access to the parameters

4. **Create a subscriber** object to the `/front/scan` topic, which contains the laser scan message from the Husky robot.
   For the moment, set the queue size to 10.
   The message type of `/front/scan` is LaserScan:
   http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html
   Take inspiration from the *ros_best_pratices* package, but leave the subscriber's callback empty.
   The package should compile without errors.

5. Take a look at your *controller.launch* file (same as in "Session 1 - Exercise 3").
   For the moment, in the launch file, there is no call to the package `ro14_husky_controller` (and its node *5ro14_husky_controller_node*) in which it is located!…
   You should try it:
   roslaunch 5ro14_husky_controller controller.launch

   It will run Gazebo and the Husky simulation in a soccer field.

   Add the lines needed in *controller.launch* file to **start the `5ro14_husky_controller_node` node**.
   Try again this launch file: now with "rostopic info", you can see that the `/front/scan` topic is subscribed by your node.

6. In the call to this subscriber ("nodeHandle->subscribe..."), now replace the queue size and topic name parameters with the ***scanTopicName*** and ***scanTopicQueueSize*** variables.
   Load parameters named *scan_topic_name* and *scan_topic_queue_size* into these two variables. Use the m_nodeHandle->getParam() function to do this.
   Take inspiration from the *ros_best_pratices* package
   If all goes well, your code should compile without any problems.

7. In your *controller.launch* file, add the commands needed to **load the two parameters** (*scan_topic_name* and *scan_topic_queue_size)* with the *param* tag
   If all goes well, your launch file should start without crashing.

8. Complete the callback method for your subscriber. It should **compute the smallest distance** measurement from the vector *ranges* in the laser scanner message, and output it to the terminal with ROS_INFO_STREAM function.
   Check the message type here :
   http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html
   Tips: To filter some “NaN” values, you can use the function std::isnormal()
   If all goes well, your code should compile and your launch file should start without crashing.

   It is difficult to see properly into the terminal because of the output from other nodes. The solution is to use an ROS tool that allows messages to be filtered: ***rqt_console***.
   Try it.

   To test the distance, you can place, in Gazebo, a geometric volume in front of the Husky robot.

9. Display the laser scan in **RViz** with the “LaserScan” plugin (click on “*add”* at the bottom left of the window).

10. Make sure you **set *odom* as the *Fixed Frame*** (under Global Options) and adjust the size of the laser scan points. You can save your current RViz configuration as the default configuration by pressing ctrl+s.

11. In your launch file, use the RViz package to load the RViz configuration file you saved earlier. This will automatically launch RViz when you run your launch file.