ROB314 – Session 2 - Exo1

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.

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

   In you file ~/.bashrc, at the end, add the line :
   ```
   export HUSKY_LMS1XX_ENABLED=1
   ```

   Download the Zip archive containing prepared files of the package rob314_husky_controller at this adress:
   https://perso.ensta-paris.fr/~battesti/rob314_download/session_2/rob314_husky_controller.zip

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

   You need the package `teleop_twist_keyboard`, as in the exercise of the session 1.
   If needed, install it in the folder ~/catkin_ws/src, 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.
   Inspect the source code.
   For the moment, the node doesn't do anything.

MyNode.cpp create a node called ***rob314_husky_controller_node***, then create an instance of ***MyController***, then use the blocking function ***ros::spin()***.
MyController.cpp will contain:

- the subscription to a topic
- the callback to the topic
- the access to parameters

4. **Create a subscriber** object to the `/scan` topic (or with Noetic, `/front/scan`), this topic contains the laser scan message from the Husky robot.
   For the moment, set the queue size at 10.
   The message type of `/scan` (or with Noetic, `/front/scan`) is LaserScan: http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html
   Find inspiration in the *ros_best_pratices* package, but keep the subscriber's callback empty.
   The package should compile without errors.

5. Take a look in your *controller.launch* file (same as in "Session 1 - Exercise 3") and add the commands needed to **launch the `rob314_husky_controller_node` node**.

6. In the call to this subscriber, now replace the queue size and topic name by variables ***scanTopicName*** and ***scanTopicQueueSize***.
   Load parameters named *scan_topic_name* and *scan_topic_queue_size* into these two variables. For that, use the m_nodeHandle->getParam() function.

7. In your *controller.launch* file, add the commands needed to **load the two parameters** (*scan_topic_name* and *scan_topic_queue_size)* with *param* tag

8. Complete the callback method for your subscriber. It should **compute the smallest distance** measurement from the vector *ranges* in the message of the laser scanner, and output it to the terminal.
   Inspect the message type here :
   http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/LaserScan.html
   Tips: To filter some "NaN" value, you can use the function std::isnormal()

9. Show the laser scan in **RViz** with the plugins "LaserScan" (click *add* at the bottom left of the window) .

10. 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.

11. Add RViz to your launch file. And load the RViz configuration you saved earlier.