CSC_5RO14_TA – Session 3 - Ex1

Exercise

The goal of this exercise is to close the control loop for the Husky robot. You will extract the position of a pillar from the laser scan and then control the robot to drive into the pillar.

1. Let's start with your `5ro14_husky_controller` package from the session 2. Or the correction provided on the CSC_5RO14_TA website:
   https://perso.ensta-paris.fr/~battesti/website/teachings/rob314/

   In your *~/.bashrc* file, at the end, you should have the line :
   ```
   export HUSKY_LMS1XX_ENABLED=1
   ```

2. Download the *world files*, which contain a description of the environment for the Gazebo simulator, and place them in a new folder called `worlds` in your `5ro14_husky_controller` package: https://perso.ensta-paris.fr/~battesti/rob314_download/5ro14_session3_worlds.zip

3. Modify the launch file from the last exercise such that:
   - The *teleop_twist_keyboard* node is removed, because we want the robot to control its own movement . (You can also comment lines in the launch file by framing the text with `<!--     -->`)
   - `singlePillar.world` file should be loaded as the world, instead of the `robocup14_spl_field.world` file. Be careful, the path is not the same : you can use the `find` command in the launch file.
     This file comes from the `5ro14_session3_worlds.zip` zip file. It is a world with only one cylindrical shape, a "pillar".

   If all goes well, the launch file should start without errors, but also without any action on the robot.

The goal of the following steps is to send the robot a command to the pillar. To do this, we calculate the distance and angle of the pillar as seen by the lidar. We'll use the "cmd_vel" topic to send the "Twist" command message. First we need to add a dependency to "geometry_msgs". Next, we need to configure a "publisher" that will be used to send this message to the robot. Finally, we need to fill this message with the direction to the pillar that we've already calculated.

4. By modifying the `laserCallBack` function, extract the distance and angle of the pillar from the laser scan with respect to the robot. (check the data on https://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html)
   If all goes well, your code should compile without a problem.

5. In the idea of using the `/cmd_vel` topic to send a `geometry_msgs::Twist` message to Husky, you need to add `geometry_msgs` as a dependency to your package.
   Do this by modifying your `CMakeLists.txt` and `package.xml` (same structure as with `sensor_msgs`) (Session 2).

If all goes well, your code should still compile without any problems.

6. Create a publisher (named `m_commandVelocityPublisher`, for example) in the `MyController` class, on the `/cmd_vel` topic to be able to send a `geometry_msgs::Twist` message to Husky.
   Nothing will happen until this publisher publishes the message, but we can still check that the program compiles correctly.

7. In the callback method of the laser scan topic, write some code that moves Husky toward the pillar, by using the angle of the pillar from the laser scan with respect to the robot (see question 4). This can be a simple P (proportional) controller.
   In the callback method, we will fill in the message and send it with the correct publisher function.

8. Add a new ROS parameter for your controller gain in your node and use it in the callback method (Session 2).

9. Modify your launch file to add the controller gain parameter.
   Run your launch file with `roslaunch`.
   Try different values of your gain to find a good value. What happens if the value is too high or too low?

10. Start your launch file with `roslaunch`.
    - If necessary, add a “RobotModel” plugin to RViz to visualize the Husky robot. (Session 2)
    - Make sure that *odom* is set as the *Fixed Frame* (under Global Options) and adjust the size of the laser scan points, if necessary.
    - Display the laser scan in RViz using the “LaserScan” plugin (click on “*add*” in the bottom left of the window if necessary).
    - Add a “TF display” plugin to RViz,
    - Visualize all the TFs on the robot. You may need to use the “Focus Camera” button.
    - Save the RViz configuration in a *.rviz file in your package.

11. In Gazebo, you can translate the pillar. If you place the pillar behind the robot, the robot cannot see the pillar. Find a (simple) algorithm and code it to solve this case.

12. We want to visualize the estimated position of the pillar in RViz. For this, we will use a special message : `visualization_msgs::Marker.`
    Have a look at this page : https://wiki.ros.org/rviz/DisplayTypes/Marker
    Publish a visualization marker for RViz, for example a sphere, that shows the estimated position of the pillar. For example, you can use a topic called `/pillar_marker.`
    We want to publish every time we detect the pillar, so inside the `laserCallBack` callback function, you can publish the marker at a position in the *frame* of the laser.
    So you will use a relative position according to the position of the laser. To make it work, you need to specify the correct `header.frame_id` for the message. It should be the same as the received `LaserScan`. RViz will automatically transform the marker into the odom frame.

13. Start your launch file with `roslaunch`. Add a “Marker” plugin to RViz to visualize your marker message. Make sure to set `/pillar_marker` as “Marker Topic” in RViz.

14. You have more time ?

    In Gazebo, you can use the `driveThrough.world`.
    Code a node for the robot to drive between the two pillars.