Lab exercise Nr. 2

This exercise will cover running navigation stack to navigate your robot in an unknown environment. We will add a lidar sensor to our robot to help it understand its environment.

1. First lets change our simplebot_description to help the robot see the environment. Lets add Hokuyo lidar to robot description file. Add the following code to `simplebot.urdf.xacro` file:

```
  <!-- Macro defining hokuyo lidar units -->
  <xacro:macro name="hokuyo_lidar" params="side x y z yaw">
    <!-- Lidar link -->
    <link name="${side}_laser_link">
      <collision>
        <origin xyz="0 0 -0.0225" rpy="0 0 0"/>
        <geometry>
          <box size="0.05 0.05 0.07"/>
        </geometry>
      </collision>
      <visual>
        <origin xyz="0 0 -0.0225" rpy="0 0 0"/>
        <geometry>
          <box size="0.05 0.05 0.07"/>
        </geometry>
        <material name="white"/>
      </visual>
    </link>
    <!-- Joint connecting lidar to base_link -->
    <joint name="${side}_laser_joint" type="fixed">
      <parent link="base_link"/>
      <child link="${side}_laser_link"/>
      <origin xyz="${x} ${y} ${z}" rpy="0 0 ${yaw}"/>
    </joint>
    <!-- Gazebo tag describing how the lidar is simulated in 
Gazebo -->
    <gazebo reference="${side}_laser_link">
      <sensor name="${side}_laser" type="ray">
        <pose>0 0 0 0 0 0</pose>
        <ray>
          <scan>
            <horizontal>
              <!-- The URG-04LX-UG01  has  683 steps with 
0.35139 Degree resolution -->
              <resolution>1</resolution>
              <max_angle>2.0944</max_angle>
```

```
               <!-- 120 Degree -->
               <min_angle>-2.0944</min_angle>
               <!-- -120 Degree -->
               <samples>683</samples>
             </horizontal>
           </scan>
           <range>
             <min>0.2</min>
             <max>5.6</max>
             <resolution>0.01</resolution>
           </range>
         </ray>

         <plugin name="${side}_laser"
filename="libgazebo_ros_laser.so">
           <topicName>${side}_laser/${side}_scan</topicName>
           <frameName>${side}_laser_link</frameName>
         </plugin>

         <always_on>1</always_on>
         <update_rate>10</update_rate>
         <!-- You can change this property to true to
visualize lidar rays in Gazebo -->
         <visualize>false</visualize>
       </sensor>
     </gazebo>
   </xacro:macro>

   <!-- Now we actually create two lidar instances that are
added to our robot -->
   <xacro:hokuyo_lidar side="front" x="0.35" y="0" z="0.15"
yaw="0"/>
   <xacro:hokuyo_lidar side="back" x="-0.35" y="0" z="0.15"
yaw="${M_PI}"/>
```

Note: As always the code should be added before the final `</robot>` flag of the xacro file!

2. Now that your robot has lidar sensors we can start navigating in the world and avoiding obstacles. First create new package named:

```
simplebot_navigation
```

3. Second step is to create launch folder in this package. In this folder we will create two launch files, namely `move_base_local.launch` and

`navigation.launch`. `move_base_local.launch` launch file will start the main navigation stack node called `move_base`. This node executes all navigation related code. Internally it contains many plugins such as cost_maps for representing the map of the robot surroundings and planners that are used to search for paths that the robot is expected to execute. We will discuss this in details later. Now lets create `move_base_local.launch` file with the following content:

```
<launch>
  <!-- The main node that starts the navigation stack -->
  <node pkg="move_base" type="move_base" respawn="false"
name="move_base" output="screen">
    <!-- common configuration for both global and local
costmaps -->
    <rosparam file="$(find
simplebot_navigation)/config/costmap_common.yaml"
command="load" ns="global_costmap"/>
    <rosparam file="$(find
simplebot_navigation)/config/costmap_common.yaml"
command="load" ns="local_costmap"/>
    <!-- the configuration that is separate for the global
and local cost maps -->
    <rosparam file="$(find
simplebot_navigation)/config/costmap_global_odom.yaml"
command="load"/>
    <rosparam file="$(find
simplebot_navigation)/config/costmap_local.yaml"
command="load"/>

    <!-- the configuration of the planners -->
    <rosparam file="$(find
simplebot_navigation)/config/teb_local_planner.yaml"
command="load"/>
    <rosparam file="$(find
simplebot_navigation)/config/global_planner.yaml"
command="load"/>

    <!-- The velocity command that is send to the robot
controller -->
    <remap from="cmd_vel"
to="/simplebot_controller/cmd_vel"/>
  </node>
</launch>
```

And a separate file `navigation.launch`. This file is used to both start our robot simulator and the `move_base_local.launch.` The content of this file are as follows:

```
<launch>
  <!-- Run the robot simulation file that we previously
created -->
  <include file="$(find
simplebot_control)/launch/simplebot_controllers.launch"/>

  <!-- Start the move_base_local file that runs the
navigation stack -->
  <include file="$(find
simplebot_navigation)/launch/move_base_local.launch"/>
</launch>
```

4. Now create the `config` folder where we will store the config files that are loaded by navigation stack.
5. Create the costmap_common.yaml configuration file with this content:

```
# describes robot boundaries as a polygon
footprint: [[0.45, 0.3], [0.45, -0.3], [-0.45, -0.3], [-0.45,
0.3]]
# describes how much the footprint should b increased when
avoiding obstacles
footprint_padding: 0.03
# the name of robot base_link frame
robot_base_frame: base_link

# how often to send constmap to rviz for visualization
publish_frequency: 1.0
# the maximal time between transform messages
transform_tolerance: 0.5

# the parameters of the costmap inflation layer plugin
inflation_layer:
  # how fast cost decreases after inflation radius
  cost_scaling_factor: 8.0
  # the radius that is used when inflating obstacles
  inflation_radius: 1.5

# parameters of the obstacles costmap
obstacles:
  # list of sensors used for obstacle avoidance
  observation_sources:  front_laser back_laser
  # indicates that unknown space should be tracked
  track_unknown_space: true
  # the parameters of first observation source
  front_laser: {
    # data source type (type of sensor)
    data_type: LaserScan,
    # sensor message frame id
    sensor_frame: /front_laser_link,
    # sensor message topic
    topic: /front_laser/front_scan,
    # indicates that sensor can add new obstacles
    marking: true,
    # indicates that this sensor can remove obstacles
    clearing: true,
    # how often we expect to receive sensor messages (shoud
increase in slower computers)
    expected_update_rate: 0.1,
    # indicates that infinity measurements are valid
    inf_is_valid: true,
    # how far can the furthes obstacle be
    obstacle_range: 15.0,
    # how far to raytrace when removing obstacles
    raytrace_range: 20.0
    }
  # the parameters of second sensor (parameter meaning is the
same)
  back_laser: {
    data_type: LaserScan,
    sensor_frame: /back_laser_link,
    topic: /back_laser/back_scan,
    marking: true,
    clearing: true,
    expected_update_rate: 0.1,
    inf_is_valid: true,
    obstacle_range: 15.0,
    raytrace_range: 20.0
    }
```

You can read more about all of these parameters here. Note that parameters here are specified according to the new specification (this can be different in old ROS tutorials)!

6. Now create parameters that are only valid for global cost map (`costmap_global_odom.yaml`). In this exercise we only want the navigation to be performed in odom frame without using the global map. We still have to create a global_costmap with an empty map. This file looks like this:

```
global_costmap:
  # indicates that map is not changing its position with
robot
  static_map: true
  # should be opposite of the above
  rolling_window: false

  # the global frame id of this costmap, we are only
navigating in odom frame with
  # no map therfore this is set to odom. It should be set to
/map otherwise
  global_frame: /odom

  # update rate of this costmap
  update_frequency: 3.0

  # patameters that determine the size of this map
  width: 30.0
  height: 30.0
  resolution: 0.2
  origin_x: -15.0
  origin_y: -15.0

  # the plugins that are used by this costmap. Note that
parameters of these plugins
  # can be found in costmap_common.yaml file!
  plugins:
    - {name: obstacles,       type:
"costmap_2d::ObstacleLayer"}
    - {name: inflation_layer, type:
"costmap_2d::InflationLayer"}
```

7. Now we need similar configuration file for the local costmap. The file name should be `costmap_local.yaml` with the following content:

```
local_costmap:
  # global frame of this costmap
  global_frame: /odom

  # plugins used by this costmap
  plugins:
    - {name: obstacles,       type:
"costmap_2d::ObstacleLayer"}
    - {name: inflation_layer, type:
"costmap_2d::InflationLayer"}
```

```
  # We'll configure this costmap to be a rolling window...
meaning it is always
  # centered at the robot
  static_map: false
  rolling_window: true

  # update frequency of this costmap
  update_frequency: 10

  # the size and resolution of this costmap
  width: 5.0
  height: 5.0
  resolution: 0.05
  origin_x: 0.0
  origin_y: 0.0
```

Note that `costmap_local.yaml` and `costmap_global_odom.yaml` only contain the parameters that are different between the costmaps. The config that is the same for both costmaps are located in `costmap_common.yaml`.

8. Now that we have our local and global costmaps we need planners that will find paths in these costmaps. The parameters of global planner are added to `global_planner.yaml` and it looks like this:

```
base_global_planner: global_planner/GlobalPlanner

GlobalPlanner:
  publish_potential: false
```

To make sure that you can start this planner the following package has to be installed:

```
sudo apt install ros-kinetic-global-planner
```

9. Now lets create the config for the local planner (`teb_local_planner.yaml`). We are using the teb planner that is installed using this command:

```
sudo apt install ros-kinetic-teb-local-planner
```

The configuration file content is as follows:

```
base_local_planner: teb_local_planner/TebLocalPlannerROS

TebLocalPlannerROS:

 odom_topic: /odom
 map_frame: /odom

 # footprint_model that is used by local planner. This is
different from costmap
 # footprint!
 footprint_model:
   type: "two_circles"
   front_offset: 0.2 # for type "two_circles"
   front_radius: 0.2 # for type "two_circles"
   rear_offset: 0.2 # for type "two_circles"
   rear_radius: 0.2 # for type "two_circles"
```

You can read more about the configuration of this planner here.

10. The last step is to open `rviz` and visualize all the things that we have created with `move_base` node. First change the fixed frame to `odom`. Now visualize the following topics/information:

```
RobotModel
TF
/back_laser/back_scan
/front_laser/front_scan
/move_base/GlobalPlanner/plan
/move_base/TebLocalPlannerROS/local_plan
/move_base/TebLocalPlannerROS/teb_poses
/move_base/current_goal
/global_costmap/costmap
/local_costmap/costmap
/simplebot_controller/odom
```

11. Last try to issue navigation goals for you robot. This can be done with `2D Nav Goal` tool from the top panel of the `rviz` window.

This is the end of the L2 exercize. As always full code is available here. Our simple robot will not be navigating very well in all situations. This can be fixed by making the robot model more maneuverable and of circular shape. Students that want maximal mark during defense should adjust the robot geometry and parameters so that it is able to navigate through doors that are 1.5x larger than robot radius. Good luck and feel free to contact me if you have questions.