Lesson 6 Lab sheet- IMAT5233 Intelligent mobile robotics
Assignment 1
Creating a 2D Map of an Unknown Environment by tutulebot3

Aims

- Visualising a map using rviz
- Update grid cells on a map
- Subscribe to the /odom topic and get the co-ordinate and heading of Tutlebot3
- Subscribe to the topic related to the laser scan and find the position of an obstacle in the global coordinate system

- Open the Tourtlebote3 gazebo simulator project that we used in the previous Lesson, and Launch the Turtlebot in a non-empty environment (Install the Turtlebot3 packages if you don't installed it – see previous lab sheets for mor information):

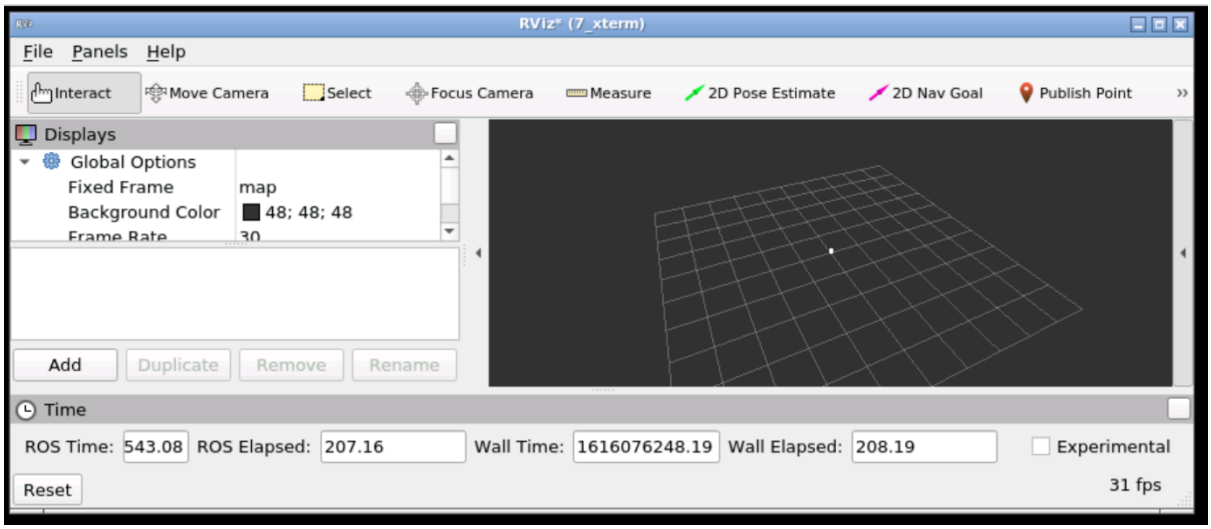
  $ export TURTLEBOT3_MODEL=burger

  $ roslaunch turtlebot3_gazebo turtlebot3_world.launch

- open the Gazibo simulator

1. Visualising a map using rviz:

- Rviz, abbreviation for ROS visualization, is a powerful 3D visualization tool for ROS
- Run rviz using the following instruction:
- $rosrun rviz rviz
- By clicking on the 'Graphical tools' tab in the bottom bar of ROS development studio.
- You should see the following figure:



- Create an empty package called 'mapping'

- Create a python code in the package and call it ‘mapper.py’. Use the python file available on blackboard. Make ‘mapper.py’ executable (chmode +x …)
- In ‘mapper.py’ there is a class called ‘Map()’. The Map class stores an occupancy grid as a two-dimensional numpy array.
- The second class in the ‘mapper.py’ code is ‘Mapper()’ that will create a map from laser scan data. Have a close look at the subscriber that subscribes to ‘scan’ topic, i.e. rospy.Subscriber('scan', LaserScan, self.scan_callback, queue_size=1) and its call back function: ‘self.scan_callback’
- The Mapper() class has a member function called ‘scan_callback()’ which is a call back function of the node that is subscribed to ‘scan’ topic. You could write your code in this function to plot on the map shown in rviz.
- Adjust the values of the grid cells, self._map.grid[i, j], and see what will happen on rviz. Note that the hight and the width of the grid have been initialised in the ‘Map()’ class to 50.
- If you couldn’t see the map in the rviz, you might need to add topic map to your rviz.
- use ‘rosrun’ command to run ‘mapper.py’.
- Therefore the code, i.e. mapper.py, will create the initial grid for mapping and you can change the probability of occupancy of a cell at i^{th} hight and j^{th} width by ‘self._map.grid[i, j]=pij’ in ‘scan_callback()’ function.
- Initialise the probability related to all cells in the grid to 0. Then move the robot and find the barriers.
- Use the sensor reading from ‘scan’ topic to find the position of an obstacle, and find the corresponding cell on the grid related to each obstacle. (Note that the resolution of the map was set to 0.1 in ‘Map()’ class. ‘resolution=.1’ means width and height of each grid cell is 0.1 meter.) Then Change the probability of the cell on the grid map corresponding to the obstacle to 1. This procedure will repeat for different readings/obstacles to create a scattered map.
  Use the following steps as a guideline to do the mentioned procedures.

2. Subscribe to the /odom topic and get the co-ordinate and heading of Tutlebot3:

- Write python code to subscribe to ‘/odom’ topic and print the **quaternions** data for the robot based on what you learnt in previous weeks. Use the code in Appendix A as a guideline to convert the quaternions data, (x,y,z,w), to Euler angles, i.e. (roll, pitch, yaw). Note that you should change the code based on your requirement. As the turtlebot3 could not fly, only the yaw changes when the robot turns around. Use the turtle bot keyboard teleoperation to turn the bot and see if the yaw is changing?
- Print the position of the robot in the global coordinate system using ‘msg.pose.pose.position’ along with the yaw angle. Move the robot around in the 2D space and see which component of the position will change. Then create a triplet that contains the global position of the robot in the 2-D coordinate system and heading (yaw) of the robot, i.e. (x_r, y_r, θ_r).

3. Subscribe to the topic related to the laser scan and find the position of an obstacle in the global coordinate system

- Launch the keyboard teleoperation and move the turtulebot3 close to a barrier in a position where there is a barrier in front of the robot.
- write appropriate code based on what you have learnt from week 6 to subscribe to the topics that have the laser scan data. As you remember we should subscribe to '/scan' topic:

```
#! /usr/bin/env python

import rospy
from sensor_msgs.msg import LaserScan

def callback(msg):
    print (len(msg.ranges))
rospy.init_node('scan_values')
sub = rospy.Subscriber('/scan', LaserScan, callback)
rospy.spin()
```

- Change the 'callback (msg)' function in the python code to print the range for the reading in the degree of 0. Use 'msg.ranges[0]' to read the range (distance) of barrier which is in 0 degree, i.e. print (msg.ranges[0]). So you have the direction, 0 degree, and the distance of the barrier, i.e. (r,θ_s). See the lecture note for week 4 (Scattergram mapping) for more information.
- Find the local position of the barrier at (r,θ_s), i.e. (x_s, y_s), using the trigonometry described in the lecture note for week 4.
- Map the local coordinate of the barrier, i.e. (x_s, y_s), to the global coordinate system using the robot position in the global system acquired in the previous stage, i.e. (x_r, y_r, θ_r). As you remember from the lecture note, you need a rotation and translation:
- Multiply in the rotation matrix to get $(\mathrm{x}', \mathrm{y}')$:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta_r) & -\sin(\theta_r) \\ \sin(\theta_r) & \cos(\theta_r) \end{bmatrix} \mathrm{x} \begin{bmatrix} x_s \\ y_s \end{bmatrix}$$

$$\begin{cases} x' = \cos(\theta_r)x_s - sin(\theta_r)y_s \\ y' = \sin(\theta_r)x_s + cos(\theta_r)y_s \end{cases}$$

Translate $(\mathrm{x}', \mathrm{y}')$to get the position of the barrier in the global system, i.e. (x",y"):

$$\begin{cases} x" = x' + x_r \\ y" = y' + y_r \end{cases}$$

- Find the corresponding cell in the map and put its occupancy probability to 1 (see the following Note 1).
- This procedure will be continued while the robot moving until all the scattered map is created. You could use more reading in different degrees instead of only one reading at 0 degree, i.e. msg.ranges[i], to speed up the map creating and improve the accuracy of the map.

Note 1:

In the initial code, i.e. mapper.py, the grid size is 50 by 50. There are 50*50 cells in the grid. The probability for the 50*50 cells are hold in an array called ‘self._map.grid’. As it is shown in ‘scan_callback(self, scan)’ function in ‘mapper.py’, we should pass the appropriate ‘i’ and ‘j’ indices to set the probability for the cell: self._map.grid[i, j] = .9. The indices ‘i’ and ‘j’ should be an integer in [0,50] for this grid. You could change the grid size to have a larger grid.

So you need to find the corresponding ‘i’ and ‘j’ based on x and y value. Add the following function to ‘class Mapper(object)’ and use it to get the indices corresponding to appropriate cell in self._map.grid[i, j] array by passing x and y.

```
def get_indix(self, x, y):
    x=x-self._map.origin_x
    y=y-self._map.origin_y

    i=int(round(x/self._map.resolution))
    j=int(round(y/self._map.resolution))
    return i, j
```

Note 2:

Note that the laser sensor has reliable reading if the distance of a barrier is higher than ‘range_min’ and also the distance is lower than ‘range.max’. So, it could be a good idea to use an ‘if’ to only consider each reading that is between the min and max value:

```
if scan.ranges[i] > scan.range_min and scan.ranges[i] < scan.range_max:
```

Note 3:

Note that if you use 'i' in scan.ranges[i] to find the distance of an obstacle then the theta_s for that obstacle will be ‘i’ degree. If you use ‘math.sin()’, you should convert the degree 'i' in scan.ranges[i] to radians using 'math.radians(i)'.

Appendices:

Appendix A: **Convert quaternions to Euler angles [1]**

```
#!/usr/bin/env python
import rospy
from nav_msgs.msg import Odometry
from tf.transformations import euler_from_quaternion, quaternion_from_euler

def get_rotation (msg):
    global roll, pitch, yaw
    orientation_q = msg.pose.pose.orientation
    #print(msg.pose.pose.position)#xyz position
    orientation_list = [orientation_q.x, orientation_q.y, orientation_q.z, orientation_q.w]
    (roll, pitch, yaw) = euler_from_quaternion (orientation_list)
    print yaw

rospy.init_node('my_quaternion_to_euler')

sub = rospy.Subscriber ('/odom', Odometry, get_rotation)

r = rospy.Rate(1)
```

```
while not rospy.is_shutdown():
    r.sleep()
```

Reference:

[1] How to convert quaternions to Euler angles
https://www.theconstructsim.com/ros-qa-how-to-convert-quaternions-to-euler-angles/
[2] nav_msgs/OccupancyGrid Message http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/OccupancyGrid.html
[3] [RDS] 007 - ROS Development Studio #Howto use RViz and other ROS Graphical Tools in RDS
https://www.youtube.com/watch?v=xkS_OrMN2ag
[4] ROS Navigation
https://risc.readthedocs.io/1-ros-navigation.html#mapping
[5] tf tutorial
http://wiki.ros.org/tf/Tutorials
[6]Random moving
http://www2.ece.ohio-state.edu/~zhang/RoboticsClass/docs/ECE5463_ROSTutorialLecture3.pdf
[7] rviz User Guide
https://wiki.ros.org/rviz/UserGuide
[8]How to Output Odometry Data
https://www.theconstructsim.com/ros-qa-196-how-to-output-odometry-data/