CSE 468/568 Lab 1: Robot Control and Using `tf` in ROS

The objective of this assignment is to get started with robot simulation in ROS, and learn to control the car with a LiDAR.
Extra credit: use `tf` to relate the coordinate system of one robot with the other.

Prep Work

- Complete ROS tutorials 1 through 12 (in Section 1.1 Beginner) and be able to write a simple publisher and subscriber in ROS
- You should now know which language you will be programming in. Most students typically use Python or C++. Depending on this choice, go through tutorial 11 or 12 in detail on how you can program ROS nodes with publishers and subscribers.
- Use the provided VM or install F1tenth simulator on your own Linux installation by following the given instructions.
- Familiarize yourself with basic Linux use including terminal commands. Here are two tutorials that could help in this regard
  - The Linux command line for beginners
  - Basic Linux commands for beginners

Learning Outcomes

- Getting familiar with programming in ROS
- Getting familiar with the F1tenth simulator
- Learning `ROS tf`

Getting Familiar with ROS

We hope you are now familiar with the ROS package structure. If you are using the class provided VM, your workspace should be in `~\f1tenth`. Make sure you have set up your environment as described in Section 2 and 3 here.

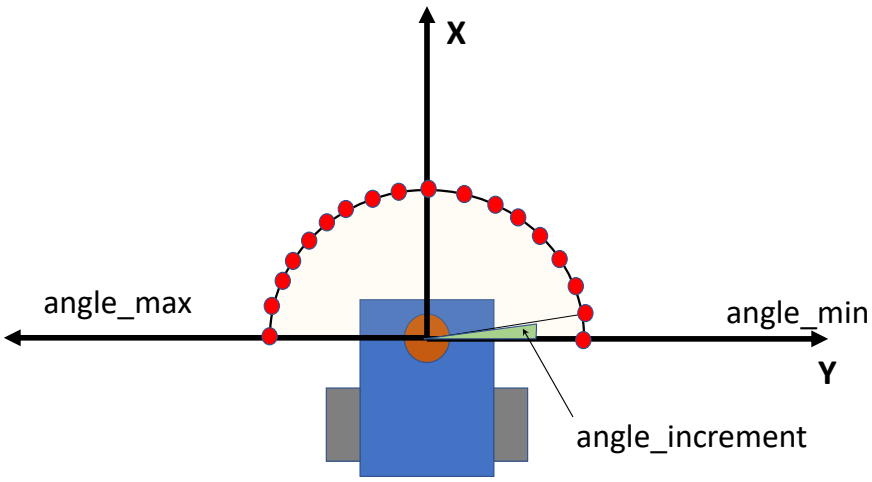

Figure 1: Explanation of fields in LaserScan Message. `angle_min` is the orientation of the start of the scan readings wrt sensor x-axis. `angle_max` is the orientation of the end of the scan readings wrt sensor x-axis.

Evader Controller (7 points)

In this section, you will write your own controller. Familiarize yourself with the given robot by checking the information it publishes and subscribes. The robot also has a laser range finder attached to it. Move the robot by dragging it with your mouse pointer to face a wall. Monitor the output of the laser by using `rostopic echo` in the command line. You should be able to understand the output of the sensor from the stage world file and its output you observed.

Write a controller node that drives the robot straight at a constant speed of 2m/s. When the robot is close to an obstacle, turn in a random new direction (add a non-zero `steering_angle`), and continue driving. Create a new launch file `evader.launch` that adds the execution of this node to what was in the earlier launch file.

LaserScan Message

NOTE: All angles are in radians. You can convert any of those readings to degrees by multiplying it by $\frac{180}{\pi}$ where π is 3.14158.

Shown in Figure 1 is a top-view of a robot with a LiDAR. The sensor coordinate frame is assumed to be the same as the robot coordinate frame. The LiDAR shown has a 180 degree field of view. Since the sensor axis is aligned with the robot axis, the x-axes of

sensor and robot are aligned (i.e., sensor is facing forward wrt the robot). The red circles are the orientations of the points that the LiDAR can measure range to. If you count, there are 18 such points. Therefore, the shown sensor can measure range at points every 10°. All angles are wrt sensor x-axis, which in this case is also the robot x-axis. As described earlier, this webpage shows the `LaserScan` data structure. In this case, if you did a `rostopic echo \scan` you would see the following values

```
angle_min=-1.57079
angle_max=1.57079
angle_increment=.174532
```

This should tell you what you see from Figure 1 that the sensor has a viewing range from -90°(-1.57079 radians) to +90°(+1.57079 radians) wrt sensor x-axis. It should also tell you that the sensor can measure ranges to points at increments of 10°(.174532 radians). The size of `ranges` array should be 18 with `ranges[0]` corresponding to the range measurement to a point at -90°wrt sensor x-axis, `ranges[1]` corresponding to the range measurement to a point at -80°wrt sensor x-axis, `ranges[2]` corresponding to the range measurement to a point at -70°wrt x-axis of the sensor and so on.

NOTE: This is NOT how the LiDAR on the F1tenth car is configured. However, if you look at the corresponding values of `angle_min`, `angle_max` and `angle_increment`, you should be able to figure out what the corresponding `ranges` array readings correspond to. This is a simple example just to illustrate the meaning of these values.

Pursuer-Evader (Extra Credit: 2 points)

Read through the `tf` tutorial. Publish the coordinate frame of the robot wrt the global frame. Create a new world file with a second robot called `pursuer`, and drop it into the world close to the first robot. Write a controller node for the pursuer that subscribes to the `tf` messages from the evader, and follows the evader by going to the spot it was at from one second before. Create a third launch file `pursuer-evader.launch` to run the new world with the two robots, the evader controller, and the pursuer controller as separate nodes.

Submission Instructions

Lab submissions will be through Autograder. You'll need to VPN into the UB network or be on campus to submit. Please see here for instructions to do this. We will add you to the roster on the Autograder portal. You should zip the f1tenth simulator package and submit that package after naming it lab1.zip. The assignment is due Saturday, Oct 2 before midnight.

Your programming assignment will be worth 7 points with an additional 2 points for extra credit. We'll have a separate multiple choice questionnaire on autograder that will account for the remaining 3 points for a total of 10 points and 2 extra credit points.