Introduction to Mobile Robotics | Assignment #3 | Name: _____
University of Ottawa | Dr. W. Gueaieb

Total (80 points)

Instructions

- Individual work only. ABSOLUTELY no collaboration of any kind with anybody concerning any part of the assignment is allowed during the assignment period. Questions must be addressed to the professor during office hours (not by email).
- Source codes will neither be marked nor debugged. They will be executed to ensure proper functionality with different input/parameters. Only the program results are graded. It is the student's responsibility to make sure that their programs run as described.
- Source codes must be very well commented so that they are easy for the marker to understand when needed.
- You have to write your own programs. Copying a source code from the Internet or any other source is not acceptable and will be considered as plagiarism.
- All students must be aware of the rules and regulations posted on the course website, especially those on assignment submission and plagiarism.

Deliverables

- Submit your work online through the course website in exactly 3 files as follows:
  1. The source code of each node in a separate file (with a proper file name and extension).
  2. A "00Readme.txt" text file explaining how to create the package and run any of the nodes.
- Do NOT archive the files in a zip file.
- Do not forget to attach all the files BEFORE you click the "Submit" button.

Problems

Path planning is an integral process of almost any robotic application. It is concerned with the calculation of a viable collision-free path from a source to a target robot. Despite its conceptual simplicity, finding a solution to this problem is more algorithmically complex than it may seem. Before formally studying this topic in class (in the few coming weeks), in this assignment you are given the opportunity to heuristically study a simple version of the problem by capitalizing on the knowledge acquired so far in class and in the previous assignment.

(80^{pts}) **1.** Some classes of path planning algorithms drive the robot to its destination by continuously alternating between two behaviors:

1. Heading the robot straight to the target in a straight line
2. Circumventing the obstacle by maintaining a fixed distance between the robot and the closest point to the detected obstacle

In this exercise, you will create a ROS package to implement this behavior on Husky (with the LiDAR enabled). The package will consist of 2 nodes, as described below. Write a Python program for each node.

(a) (20 pts) Node `sensed_object` uses the LiDAR to calculate the distance d and heading α to the closest point on a detected object (with respect to the LiDAR's frame), and publish them over topic `sensed_object` at a rate of 2 Hz. If nothing is detected within the LiDAR's nominal sensing range, then assign 'NaN' to both quantities. The messages sent over this topic are of type `Pose2D` which is defined in package `geometry_msgs`. Use message variables `x` and `theta` to store the values of d and α, respectively (store `theta` in degrees).
Hint: refer to the example seen in class about reading data from a LiDAR.

(b) (60 pts) Node `navigate_robot` drives the robot as follows:

1. Navigate the robot in a straight line towards the closest point on a detected object as along as $d \geq e$, for some safety distance $e = 0.7$ m. Note that during this phase, the object is allowed to be displaced while keeping $d \geq e$.
2. When d reaches e, the robot should fully circumvent the obstacle (either to the left or to the right, as you wish) as shown in Fig. 1, while trying to maintain that same distance e from the obstacle. At the end of this phase, the robot must stop when it is within $\epsilon = 10$ cm from its destination.
3. If d and α are 'NaN' at any instant in time, then the robot must stop.

For simplicity, you may assume that the scene may contain no more than one object, and that $d \geq e$ when an object is initially detected. Note that the object can be of any shape.
Husky's pose can be read from the `/odometry/filtered` topic. The robot's orientation (yaw) can be calculated from a quaternion using the function `euler_from_quaternion`, as explained at:
https://www.theconstructsim.com/ros-qa-how-to-convert-quaternions-to-euler-angles/
http://docs.ros.org/jade/api/tf/html/python/transformations.html
After enabling its LiDAR, Husky can be spawned in an empty environment in Gazebo by running the command: `roslaunch husky_gazebo husky_empty_world.launch`
Objects can be added by dragging them from the "Insert" tab in Gazebo to the scene. You may want to test your package with different objects. You can also add primitive objects (e.g., cuboid, sphere, cylinder) by clicking on their icons in Gazebo's top bar.

Make sure to clearly define your package parameters (e and ϵ) at the top of your code so that the Corrector can easily test your nodes with different values.

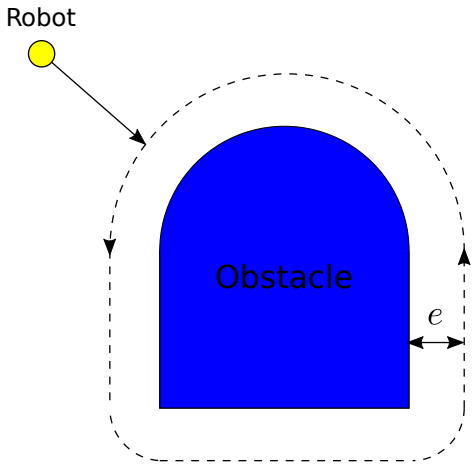

Figure 1: Setup example