Introduction to Mobile Robotics | Assignment #2 | Due (see course website)
University of Ottawa | Dr. W. Gueaieb

**Total** (20 points)

## Instructions

- Individual work only. ABSOLUTELY no collaboration or communication of any kind 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).
- Submissions are handled electronically through an automatic submission system on the course website, which shuts down automatically, as per the flexible-deadline submission policy explained on the course website. All other forms of submissions (eg., by email or hand-to-hand) will be rejected. To avoid last-minute hassles, it is recommended that you submit your updated work as you progress. You can make as many submissions as you want. Only the last submission will be marked.
- The source code will neither be marked nor debugged. It will be executed to ensure proper functionality with different input/parameters. Only the program result is graded. It is your responsibility to make sure that it runs as described. Otherwise, if you have a different (incompatible) software version for instance, it may not be possible to test it.
- The source code must be very well commented so that it is easy for the marker to understand your logic when needed.
- You have to write your own programs. Downloading 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 plagiarism.

## Deliverables

- Submit your work online through the course website in exactly two files as follows:
  1. Your source code. Name it `firstname_lastname_ID` (with the proper file extension). Do NOT archive it in a zip file.
  2. A “00Readme.txt” file explaining how created and run the ROS package.

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 position. Despite its conceptual simplicity, finding a solution to this problem is more algorithmically complex than it may sound. 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 in the previous assignment.

(20$^{\text{pts}}$) **1.** Write a Python program to implement a ROS node which takes the $(x, y)$ coordinates of a destination position and then drives the robot Husky from its initial position $(0, 0)$ to that destination. The program must:

- insist that $0 \leq x, y \leq 10$. Note that these coordinates are given with respect to the world coordinate system.
- read the robot's pose from the `/odometry/filtered` topic. The pose must not be calculated by integrating the robot's velocity over time.
- drive the robot by publishing its linear and angular velocity on the `/cmd_vel` topic with a frequency of 10 Hz.
- use the robot's LiDAR to sense and avoid a single stationary (non-moving) obstacle, placed on the robot's way, while reaching for its target location.
- consider the target position to be reached if the robot gets within 10 cm from it.
- constantly report the following signals on the screen:
  - The robot's current and target positions, and the distance (error) between them (in m).
  - The robot's current and target orientations, and the difference (error) between them (in degrees).

The program will be tested as follows:

1. Husky simulation is first launched by running the command:
   `roslaunch husky_gazebo husky_empty_world.launch`
2. Then, your node is run as described in your readme file.
3. While the robot is navigating towards its target position, an obstacle (construction cone in Gazebo) will be placed and fixed on the way of the robot. The latter must avoid collision with the obstacle by bypassing it before eventually reaching the destination.

Note that to calculate the robot's orientation (yaw) from a quaternion, you can use 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