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.

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 10Hz.
- 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 10cm 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