Hints for the POTENTIAL FIELD assignment

In order to use a potential field to guide a robot to a user selected destination, you need to create a ROS node that continuously execute the following steps:

1) compute a repulsive force vector pushing the robot away from each obstacle it can perceive, find the total repulsive force as the sum of all the repulsive forces caused by all the obstacles, $F_r = \Sigma F_r^i$
2) compute an attractive force vector force (F_a) to attract the robot toward a specific target goal (X_{goal}).
3) find the total force $F_T = F_r + F_a$, and steer the robot to lineup to this total sum of all forces, by setting its translational and rotational velocity at each step of the behaviour's loop.

You have two options for step 1, respectively:
a) access the local or the global maps that are already updated in ROS by the components included in the navigation stack. In this way you will be able to have the x,y absolute coordinates (in the coordinate system of the map) of each obstacle detected by the robots' sensors.
b) Subscribe directly to the laser readings and have a callback method to examine them and compute the resulting forces. If you use this option, and access the laser readings directly, you can easily find the polar coordinates (distance and bearing) of each obstacle detected by the laser, and from these, the x,y coordinates of each obstacle in the robot-centric coordinate system. You can then compute and sum all the repulsive forces caused by each obstacle in this robot-centric coordinate system. However, <u>it is important that before you sum the resulting repulsive force with the attractive force caused by the target goal, you express both forces in the same XY reference frame</u>. You have two options to do this last step:

- option 1: use the robot-centric reference system, by expressing the attractive force in that reference frame.
- option 2: use the absolute (map) reference system, by expressing the repulsive force in that reference frame.

In order to drive the robot, you can publish twist messages either in the same laser callback or in a separate loop (however, if you do the latter, you need to make sure that your loop calls the right spin function to let ROS handle incoming messages. The other option is to use threads (although it is generally more difficult, as you need to synchronize your code).

Computing the sum of all the repulsive force vectors in the robot's frame of reference is easy, as the origin of all vectors is the centre of the robot and you can easily find the angle of the i-th vector in the robot-centric frame (where angle = 0 points toward the current heading of the robot), by using the information in the `sensor_msgs::LaserScan` message, as in the following example:

```
// Process the incoming laser scan message
void commandCallback(const sensor_msgs::LaserScan::ConstPtr& msg) {
    // examine all readings
    for (unsigned int i=0; i< msg->ranges.size(); i++) {
      double currAngle = msg->angle_min + msg->angle_increment*i;
      // TODO: compute the i-th repulsive force vector
      // TODO: find the sum of all forces
    }
}
```

ROS has utility math functions to handle coordinate transformations and vector operations and you can use the tf sub-system in ROS to find the position of the robot in the global frame.

In addition to the procedures reported in the papers in Moodle, you can experiment with the following functions to find the **intensity of each force** (or come up with your own version):

a) The attractive force F_a must increase with the distance to the goal, try:

$$\|F_a\| = \gamma \cdot \|X_{goal} - X_{robot}\|^2$$

b) Repulsive force (per obstacle / sensor reading) must decrease with the distance to the obstacle. Remember that the force vector must point in the opposite direction to the obstacle (it must "push" the robot away from the corresponding obstacle.

$$\|F_r^i\| = \begin{cases} \dfrac{\alpha}{(d_i - d_{safe})^2} & \text{if} \quad d_{safe} + \varepsilon < d_i < \beta \\ \dfrac{\alpha}{\varepsilon^2} & \text{if} \quad d_i < d_{safe} + \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

c) Use vector summation for the forces of all sensor readings as well the attractive force:

$$F = \sum_i \vec{F}_r^i + \vec{F}_a$$

where d_{safe} is a short safety distance, and the other parameters are defined by you. For example, $d_{safe} = 0.5$, $\gamma=\alpha=1$, $\beta=4m$, $\varepsilon=0.05m$

From forces to velocity commands:

- **Angular velocity**: w=K_w (θ_F - θ_r) , where K_w is a scaling constant, θ_F is the orientation of the cumulative force, and θ_r is the orientation of the robot. Hint, use the atan2 function for your calculations, during subtraction take into account the 0, 2 discontinuity.

- **Linear velocity**: you can project the cumulative force along the orientation of the robot and use the projected force or add a dumping factor by reducing the linear velocity when the rotational one increases. Remember to truncate accordingly, to account for the robot's maximum velocity (this should be set to 0.5 for robot's safety). <u>If the vector points behind the robot, set the linear velocity to 0</u>.

Local Minima: In order for the robot to move out of local minima, you could introduce a random force. You can monitor the robots position and increase the magnitude of the random force the more the robot stays near the same place.

Development Hints

- Use either stageros or gazebo to test and experiment with your behaviour.
- Start by writing the structure of the ROS node, with publishers, subscribers, callback(s) and stub functions.
- Develop each component separately, e..g. starting from the repulsive force or from the attractive force, before combining them.
- At each stage, experiment with linear velocity always set to 0, and move the robot with your mouse to make sure that it orients itself in the right direction. When you are happy with the direction, include the linear velocity.
- Don't use magic numbers, your code must be modular, simple, efficient, easy to read, re-use and extend.
- The goal can be hard-coded in the robot (i.e. you will need to re-build it if the goal changes) or (better) it may be passed through a service or a topic and (even better) via rviz.

Test

On the day of the test, you will be asked to:

1) show a simulated run of your behaviour on your laptop using either stageros or gazebo
2) drive a real turtlebot to a place positioned 5 meters ahead of its starting positions, on a course with a number or randomly placed little obstacles.

You program will be graded based on performance (speed, number of collisions) and quality of the code.