Lab #7: Artificial Potential Field Path Planning

In this experiment, you will implement the Artificial Potential Field (APF) path planning algorithm on a TurtleBot3 robot to navigate towards a goal while autonomously avoiding obstacles. You will utilize the **LiDAR** for obstacle detection and **odometry** for robot localization.

The robot is modeled as a particle moving in a potential field (Figure 1). The goal acts as an attractive source, pulling the robot toward it, while obstacles act as repulsive sources, pushing the robot away to prevent collisions.

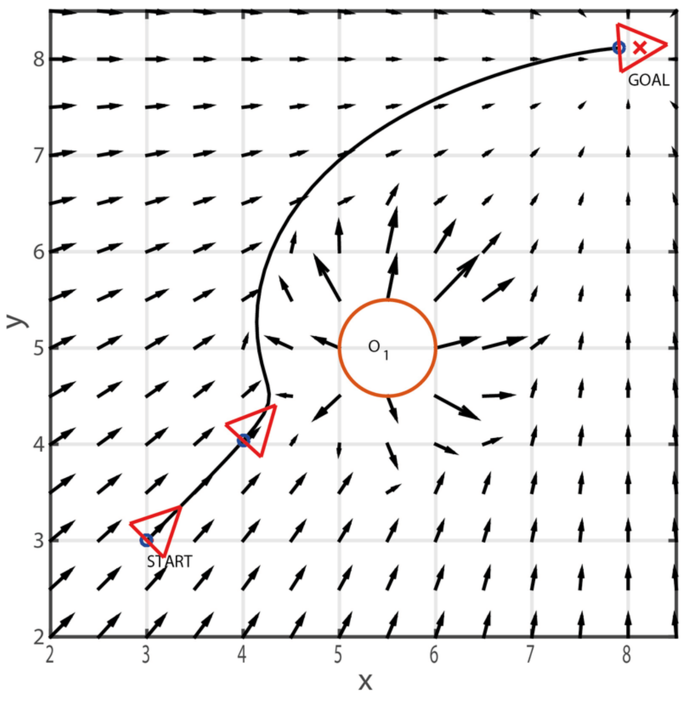

Figure 1: Artificial potential field path planning .

Recall that the attractive force F_{att} is calculated based on the error between the robot's current position **vector** q and the goal position vector q_{goal}

$$F_{att} = -k_{att}(q - q_{goal}) \tag{1}$$

where k_{att} is the attractive gain.

The repulsive force F_{rep} is only active when the robot is within a critical distance ρ_0 of an obstacle. For each obstacle detected by the LiDAR at position q_{obst}, the repulsive force is calculated as

$$F_{rep} = \begin{cases} k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\frac{q-q_{obst}}{\rho^3(q)}, & \text{if } \rho(q) \le \rho_0 \\ 0, & \text{if } \rho(q) > \rho_0 \end{cases} \tag{2}$$

where q_{obst} is the position vector of the obstacle (derived from LiDAR scan points), ρ_0 is the critical distance between the robot and the obstacle, $\rho(q)$ is the Euclidean distance to the obstacle which can be calculated as

$$\rho(q) = \sqrt{(q(1) - q_{obst}(1))^2 + (q(2) - q_{obst}(2))^2} \tag{3}$$

The total force $F_{tot} = F_{att} + \sum F_{rep}$ is converted into velocity commands as follows

$$V = [V_x, V_y] = \alpha F_{tot} \tag{4}$$

$$v = \sqrt{V_x^2 + V_y^2} \tag{5}$$

$$\theta_d = \text{atan2d}(V_y, V_x) \tag{6}$$

where the desired heading θ_d is the angle of the force vector. The angular velocity ω is then controlled by the heading error: $\omega = k_\theta(\theta_d - \theta)$.

Implementation Details

- **Localization:** Use the `/odom` topic to receive the robot's current x, y position and orientation θ. Note that orientation is provided as a quaternion; convert it to Euler angles.
- **Obstacle Detection:** The `/scan` topic provides range data from the LiDAR. Convert these polar coordinates into the robot's local Cartesian frame to identify q_{obst}.
- The overall implementation should follow the structure presented in the **Appendix**.

Environment Set-up

Refer to the Lab #0 document to connect to the **TurtleBot3** and prepare the MATLAB environment in order to start the algorithm implementation.

Things to do:

Implement the APF algorithm and validate it by performing the tests below. For each test case, you are required to **record data and generate a plot** showing the robot's trajectory (from odometry), the goal position, and the detected obstacles (from LiDAR).

- **Test 1 - No Obstacle:** Place your robot in an open space and define a goal position (e.g., $x = 2.0, y = 0.0$) in front of the robot. Ensure there are no obstacles within the sensor range. Try different goals.

- **Test 2 - Single Obstacle:** Place your robot in an open space and place an obstacle in front of the robot. Try different goals and obstacles.

- **Test 3 - Multiple/Dynamic Obstacles:** Place your robot in an open space and select a goal. Place multiple obstacles in front of the robot. Try different goals and obstacles.

- Submit your report **via SUCourse** until the report submission deadline.

Post-Lab Report Deadline: 7 January 2026, 23:59 via **SUCourse**

- Make sure that your report includes **introduction**, **procedure**, **results**, **conclusion**, **discussion** and **appendix**. Provide your **MATLAB** codes in Appendix section appropriately

Answer the following questions in the Discussion section of your Post-lab report:

1. How did you decide the gains ($k_{att}, k_{rep}, k_\theta$)? How do they affect the motion of the robot?
2. How did you decide the critical distance ρ_0?
3. What happens if an obstacle is placed directly on the line between the robot and the goal? Comment on your results.
4. What happens when you select a target behind the robot?

Appendix

Pseudo code structure:

```
%% GET INITIAL POSE & HEADING
odomSub = ros2subscriber(node,"/odom","nav_msgs/Odometry");

startMsg = receive(odomSub, 10);
pose_start = startMsg.pose.pose.position;
quat_start = startMsg.pose.pose.orientation;
x_start = pose_start.x;
y_start = pose_start.y;
angles_start = quat2eul([quat_start.w, quat_start.x, quat_start.y
    , quat_start.z]);
yaw_start = angles_start(1);

%% INITIALIZATIONS
%% MAIN LOOP
while NOT reached goal

    %% --- 1. ROBOT LOCALIZATION ---

    % Normalize Odometry Pose and Heading
    dx = current x position - x_start;
    dy = current y position - y_start;

    x_calibrated = dx * cos(yaw_start) + dy * sin(yaw_start);
    y_calibrated = -dx * sin(yaw_start) + dy * cos(yaw_start);
    q_robot = [x_calibrated, y_calibrated];
    theta = current heading - yaw_start;

    %% --- 2. CALCULATE ATTRACTIVE FORCE ---

    %% --- 3. CALCULATE REPULSIVE FORCE ---

    scanMsg = receive(scanSub, 10);
    ranges = scanMsg.ranges;
    angles_scan = scanMsg.angle_min + (0:length(ranges)-1)' *
       scanMsg.angle_increment;
    % Filter valid obstacles (Within range AND in the front)
    % Iterate over valid_idx to Calculate Total Repulsive Force

    %% --- 4. MOTION CONTROL USING TOTAL FORCE ---

end
```