CS685 - Homework 2 Part 1, September 29, *Jana Košecká*

1. **Moving to a point.** Consider a differential drive model of the mobile robot discussed in the class with a configuration space $[x, y, \theta]^T$, where we can control linear and angular velocity of the robot v and ω. The kinematic model describing the motion of the mobile robot is as follows:

$$\begin{aligned} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= \omega \end{aligned}$$

   Implement the closed loop feed-back control law for the robot to move a desired goal $[x_d, y_d]^T$. The robot's velocity should be proportional to:

$$v = k_v\sqrt{(x - x_d)^2 + (y - y_d)^2}$$

   and the angular velocity

$$\omega = k_\omega(\theta_d - \theta)$$

   where θ_d is the relative angle between the vehicle and the goal:

$$\theta_d = \tan^{-1}\frac{y_d - y}{x_d - x}.$$

   Write a function `[x,y,theta] = goToPoint(`x_0`,` x_d`)` which will take the initial pose, the goal coordinates and returns the trajectory of the robot. Hand in the code and plots of trajectories going from at least 5 different positions and headings to the desired goal.

2. **Moving to a pose.** Consider steering control between two poses for a differential drive robot. The control law is designed by transforming the problem to polar coordinates and designing the control law there. The resulting control strategy is:

$$v = k_\rho\rho \quad (1)$$
$$\omega = k_\alpha\alpha + k_\beta\beta \quad (2)$$

   where ρ, α, β is the configuration of the robot expressed in polar coordinates (with respect to the goal). You can experiment with the values around $k_r = 3; k_b = -1.5; k_a = 8$.

   (a) Write a function `[x,y,theta] = goToPose(`x_0`,` x_g`)` which will take the initial pose, the desired pose and returns the trajectory of the robot. Inside you can use the `diffDrive` function written for hw1 to simulate the robot for one time step.

   (b) Test the control law for variety of initial/goal positions. You can assume that $x_0 = [0, 0, 0]$ and $x_g = [x_d, y_d, \theta_d]$ The examples of use of that function can look like:
   ```
   [x,y,theta] = goToPose([0,0,0], [50,50,pi/2])
   [x,y,theta] = goToPose([0,0,0], [-50,-50,pi/2])
   [x,y,theta] = goToPose([0,0,0], [50,-50,-pi/2])
   ```
   Submit the plots of $x(t), y(t), \theta(t)$, the printout of the code and any observations you may have.

3. **Potential Field Based Control**. Consider a point like robot in the workspace with the area $[0, 100] \times [0, 100]$. Represent the obstacles in the environment as circles with centers at $[40, 30]$ and $[70, 40]$ each with radius 5. Assume that the initial position of the robot is $x_0 = [50, 50]$. Write down a function `goToObst`(x_g, y_g), which takes as input arbitrary goal position in the workspace and returns the trajectory which the robot followed to get to the goal using potential field based method. The parameters of the potential functions can be set as variables inside of `goToObst`. Submit the code and plots demonstrating the capability of the robot to avoid obstacles and reach the goal.

4. **Motion planning:** Probabilistic roadmap and shortest path on the grid.

   - Implement function `isCollision` in the probabilistic roadmap planner `prm.py` and experiment with the number of samples to find the path between start point [50,100] and goal point [350,80]. Visualize the the final path that is found using Dijstra's algorithm on the roadmap.
     The starter code and the map is `http://cs.gmu.edu/~kosecka/cs682/hw2/`
   - Modify the part of the code in `prm.py` for generating roadmap by assuming grid like representation of the environment where each pixel is a grid cell, assuming the 4-connected neighborhood. Given this new graph structure, run the Dijstra's algorithm on the same start and goal nodes as above.

   For both techniques submit the modified code (`prm.py, grimap.py`), images of 4 paths and roadmaps and any observations you may have. You can run the `python prm.py` just to check if the visualization works.