Potential Fields and Control
In class programming exercise #2
February 1, 2017

This is what we will work on in class. If you are trying this on your own, after you complete step 3 you should be caught up with the class. If you run into trouble please email Dr. Archibald (ca861@msstate.edu). These instructions assume that you have completed the in-lecture coding from before and have the necessary `ai_lectures` package setup.

Note: For this exercise, the forces will be in global coordinates NOT robot coordinates (which is how lab 2 will probably be).

1. First, we will get the code in the correct place to use for this lab
    - Open a new terminal in Ubuntu and ROS-change directory (`roscd`) to `ai_lectures`
        - `roscd ai_lectures`
    - Unzip programming exercise (PE2) files from blackboard into `ai_lectures` folder
    - Then copy the python script into the `scripts` folder by running
        - `mv *.py scripts/`
    - We need to make the script executable, so now type (and don't forget the *)
        - `chmod +x scripts/*`
    - And we need to copy over the world file into into the right folder
        - `mv *.world world/`
2. Now we will test that this is all working. Run the following command from the terminal. You should see the simulator window pop up and a robot (circle shape) sitting there.
    1. `roslaunch ai_lectures pfield.launch`
3. Now you can modify the code in `pfield.py` to complete the rest of the exercise.
    A. First, implement the code to determine a drive command from the force that is input. (Look for the `#PE2 A: DETERMINE DRIVE COMMAND HERE` comment, in the drive_from_force function). For this exercise, the forces will be in global coordinates NOT robot coordinates (which is how lab 2 is). So this function will differ slightly from what is in lab 2.
    B. The code has a uniform force built in (look in the main loop #1.) This force should push the robot to the top right at 45 degrees if your function is working properly. Test it by rerunning
        - `roslaunch ai_lectures pfield.launch`
    C. Now implement the goal force, in the `goal_force` function. (Look for the `#PE2 B: DETERMINE GOAL FORCE HERE (IN GLOBAL COORDINATES)` comment)
    D. Rerun and play with parameters until your robot can get successfully to the goal
    E. Change the direction and strength of the uniform force, does your robot still get to the goal?

- Tips:
    - To drive the robot forward/backward, set `twist.linear.x` to +/- values.
    - To spin the robot, set `twist.angular.z` to +/- values
    - The `atan2(y,x)` function (in the math module) will take an x and y value and return the angle to get to that (x,y) coordinate from (0,0). To get the angle from point `a(a_x, a_y)` to point `b(b_x,b_y)` call
        - `atan2( (b_y - a_y), (b_x - a_x) )`
    - If you have trouble getting ROS to `roscd` to your directory, try running the following command from the `catkin_ws` directory
        - `source devel/setup.bash`
        - Add the line "`source <path to your catkin>/catkin_ws/devel/setup.bash`" to your `.bashrc` file to have this happen automatically when you log into a terminal. (make sure that `<path to your catkin>` is replaced by the path on your system to the `catkin_ws` folder. You can get this path by typing `pwd` from the terminal when you are in the directory.