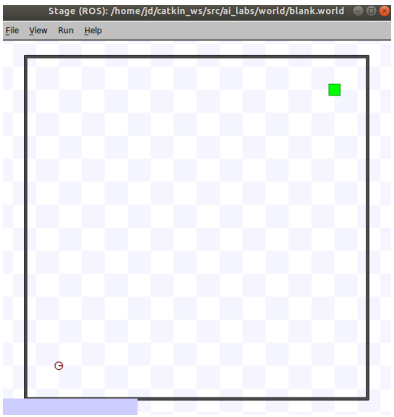
Lab 4 – Navigating with Potential Fields
100 Points

Introduction

In this lab we will use Potential Fields to navigate our robot on a map. To complete the lab you must submit via Canvas a zip file containing your source code and results of your program execution.

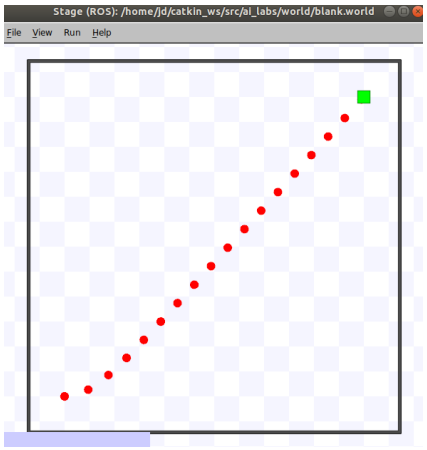
Setup Procedure

1. Download the .zip file from Canvas for Lab 4.
2. First, we will move the code to our ai_labs directory
    - Unzip files from blackboard into the ai_labs folder (you should have created this for lab 1)
    - Then make a directory (if it doesn't exist already) within ai_labs for our python scripts by running (from within the ai_labs folder)
        - mkdir scripts
    - 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 necessary files (images and world files) into the world folder
        - mv *.world world/
        - mv *.png world/
3. Install the stage_ros package using the following command (or its equivalent for other ROS versions):
    - sudo apt install ros-noetic-stage-ros
4. 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) sitting there.
    - roslaunch ai_labs lab4a.launch
    - In the Stage ROS window, go to View->Footprints to enable footprints
    - (If it doesn't work, you can try to run catkin_make from the catkin_ws folder or to run source catkin_ws/devel/setup.bash)
5. Now you can modify the code in potential_fields.py to complete the rest of the lab.



Part A [25 points]

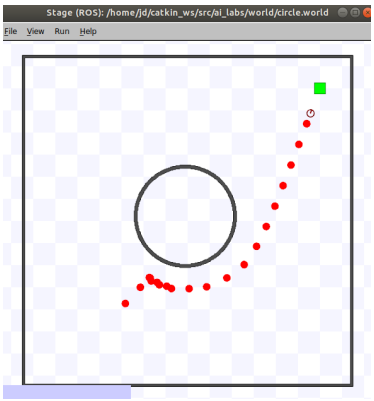
1. Your first task will be to write the code that will compute the attractive force to help the robot move toward the goal. This code should be written in the goal_force function where it says "PART A CODE HERE". The force_to_goal variable is where the goal force you determine should be stored, as that is returned
2. Once you have written this code you can test it by running the code (step 3 above). Your robot should now drive toward to the goal (since there is nothing in its way), which is shown on the map as a green box. (Tip: Use the View→Footprints menu option to view the robot's trajectory). Once there it should be able to stay there. You can play around with the parameters in the goal_force and drive_from_force functions a bit now to get this to work to your liking. We will deal with avoiding the obstacle in the next task.



Part B [30 points]

1. Now run roslaunch ai_labs lab4b.launch Your robot probably won't reach the goal, due to the large obstacle in the way. Your next task is to write the obstacle force code, which should be done in the obstacle_force function, at the spot that says "PART B CODE HERE". For each laser reading (of which there are many per scan), we will add in an associated obstacle avoidance force. Once you have written the necessary code, you can rerun your robot on to test it. Modify the various parameters in each function (goal_force,

drive_from_force, obstacle_force) until the robot avoids the obstacle like you want and then reaches the goal.



Part C [20 points]

1. You will now implement a repulsive force magnitude function that decreases linearly from the obstacle until it is 0 at the appropriate distance. The function you will modify is called get_pf_magnitude_linear and your code goes where it says "PART C CODE HERE". Once you have implemented this, modify the code in the obstacle_force function so that it calls your new repulsive force magnitude function instead of get_pf_magnitude_constant. Modify parameters in the code until the robot reaches the goal effectively.
2. Now that your robot can navigate the map from part b with your new obstacle avoidance force magnitude function, try it on a new map by running

   roslaunch ai_labs lab4c.launch

   Keep modifying the parameters and experimenting with the code until your robot can navigate to the goal on this map too.

Part D [10 points]

1. Now that your robot can navigate the map for parts a, b, and c, try it on part d by running

   roslaunch ai_labs lab4d.launch

   Keep modifying the parameters and experimenting with the code until your robot can navigate to the goal on all four maps.

Part E: Discussion [15 points]

1. Describe the process you used to determine what values to set the parameters in the code to. Do you think there is a way to do this automatically? Why or why not?

2. Describe and explain (give reasons for) the behavior of the robot when it could sense one or more obstacles and a goal was present.

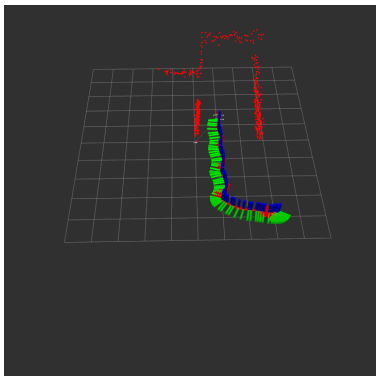
3. Describe how the different repulsive force magnitude profile (linear) that you implemented changed the behavior of the robot, and why. If it didn't change anything, explain why this is.

4. List some advantages of the potential fields approach as demonstrated in this lab.

5. List some disadvantages of the potential fields approach as demonstrated in this lab.

Part F: Extra Credit

1. [5 points] Once your robot navigates safely through the 3 provided maps, design a map that the robot cannot reach the goal on (without modifying the parameters) but for which there exists a valid path to the goal. Show this map and describe what happens. Include a screen capture of the robot's path. Discuss this and any ideas you would have to get the robot to reach the goal.
2. [5 points] If you did Extra Credit option 1, use one of the methods discussed in class to help the robot reach the goal on the map you created. Include a screen capture of the robot's new path.
3. [5 points] Implement another repulsive force magnitude function, the exponential decay method presented in the book and lecture slides. Use this to navigate the maps from this lab. Include screen captures of the robot's new paths.
4. [5 points] Instead of a repulsive obstacle avoidance force, try to get your robot to navigate through all maps using a tangential field. Do you need to be intelligent about which way your field directs the robot around each target? Include screen captures of the robot's new paths.
5. [5 points] Use rviz to visualize the robot's trajectory as well as the laser scans.



6. [max 20 pts] Open-ended. Implement any visualization or algorithmic improvements to the program. In your report, justify the change and explain how it is advantageous compared to the original program.

Tips

Make sure that the *force_to_goal* is calculated in terms of the robot's local reference frame, not the global frame. The *force_to_goal* can be calculated as follows:

```
target_angle = angle from robot [x,y] to goal [x,y]
current_angle = robot[2]
angle_to_goal = difference between target_angle and current_angle
force_to_goal = [strength * math.cos(angle_to_goal),strength *
math.sin(angle_to_goal)]
```

On the other hand, the *obstacle_force* is computed based on the distance readings from the 2D laser scanner of the robot. The laser scanner readings are in the **local** reference frame of the robot, so you may directly use them to calculate the repulsive force in the **local** reference frame of the robot without doing any reference frame conversions. The laser scanner emits 481 laser beams in total and each return from a laser beam is considered an obstacle. The total repulsive force is the sum of individual repulsive forces, one from each obstacle.

The *cur_angle* variable describes the angle in radians between the obstacle and the robot's heading in the local reference frame. Thus, *cur_angle* of 0 means that the obstacle direction is in front of the robot (the local x-axis) whereas *cur_angle* of pi / 2 means that the obstacle direction is to the left of the robot (the local y-axis). The **angle** of each individual repulsive force should be such that the resulting vector points in a direction opposite to the vector described by cur_angle. Thus, if an obstacle is directly in front of the robot, the repulsive force vector from that obstacle should point in the negative x direction.

There are two different ways to determine the **magnitude** of each individual repulsive force: constant (used in Part B) and linearly decreasing with distance (used in Part C). Thus, for Part C, you will need to call the function *get_pf_magnitude_linear* with the distance reading from laser_scan.ranges[i]. The magnitude calculation can be implemented as follows:

```
if distance < distance_threshold:
    percentage = number depending on distance and distance_threshold
    return percentage * max_strength
else
    return 0
```

where percentage is 1.0 when distance is equal to 0 and decreases linearly to 0.0 when distance is equal to distance_threshold.

Frequently-asked Questions
Q: Why is my robot stuck / oscillating / experiencing jerky motion?
A: These are expected and known issues with the potential fields algorithm. You do not need to address them directly but feel free to mention it in Part E: Discussion of your writeup.

Deliverables

You should turn in on Canvas a .zip file with your potential_fields.py file, as well as a write-up with your results. For each map, include a screen capture (as a .png or .jpg image, with a natural file name) of the simulator with the robot's trail/footprint showing the path it took to get to the goal. You may do this in multiple screen shots for a single map if the robot doesn't move fast enough to get the whole path at the same time (the footprints disappear after a little while). If you decide to do extra credit, include the necessary files and screen captures to demonstrate this.