Lab 4: Potential Field Planning

MEAM 520, University of Pennsylvania

June 30, 2020

Before starting, read the General Lab Instructions carefully!

This lab is due on **Monday, August 3, by midnight (11:59 p.m.)**. Submit your lab report as a pdf file and your code on Gradescope. Late submissions will be accepted until midnight on **Thursday, August 6**, but they will be penalized by 25% for each partial or full day late. After the late deadline, no further assignments may be submitted; post a private message on Piazza to request an extension if you need one due to a special situation such as illness. This assignment is worth 30 points.

You may do the exercise in pairs and add your partner on Gradescope to ensure both students receive credit. You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. When you get stuck, post a question on Piazza or go to office hours to ask for help!

1 Overview

In this lab, you will investigate reactive planning using potential fields for the mobile robot that you worked with in the previous labs. The power of potential fields over the off-board planners is that they are naturally reactive, so plans can change as the environment changes.

For the purpose of this lab, we are using the robot model with the LIDAR sensors. These are sensors that measure the distance to the objects around the robot. In Gazebo simulation, the sensor is attached on top of the mobile base so it can scan for obstacles in 360 degrees at that height. See the figure below (the white cylinder) .

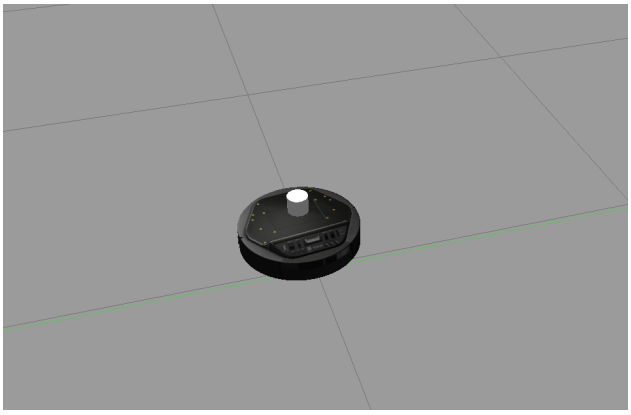

Figure 1: Mobile Robot in Gazebo Simulation

You will write a potential field planner based on this robot model and the sensor data. You will test this planner on virtual static and dynamic environments.

2 Concepts

- Write a mathematical description of a potential field controller for the mobile robot. You should report this part with equations and explain your choices.
- Include pseudocode for the relevant components of your planning algorithm (only required for the planner functions, not the main simulation loop).

3 Coding Assignment

- Download `lab4.zip` from Canvas and extract it in your directory. Go to the `code` folder which contains the following python script:
  - `runsim.py`: this file is the main simulation file that you run for your tests. It connects each part of the code, defining APIs with the mobile robot class, the planner updates, and the lidar scan processor. It initializes parameters for the tests and then runs in loops.
  - `mobile_robot.py`: this file abstracts away all communication with ROS/Gazebo. It has been given to you in previous labs but you should use the newest one.
  - `lidar_scan.py`: this file gets raw sensor data from the LIDAR scan and did some data preprocessing. It outputs the [x, y] positions of the point clouds which represent the obstacles.
  - `potential_field.py`: this file contains the skeleton for the potential field planner that you will need to implement.

  In order for the planner to work, you will need to fill in the functions in `potential_field.py`. The `mobile_robot.py` and `lidar_scan.py` are given to you and shall NOT be modified. You may meed to modify `runsim.py` for some simulation parameters in your tests. Please refer to the documentation provided in the code for more information.
- After inspecting the files in the folder, open `potential_field.py`. You will find two functions that need to be completed inside:
  - `Planner_update`: the function updates one step of the planner using potential field. The main simulation calls the function in the loop until end position is reached. The function has the following inputs/outputs (described in more detail in the function header):
    - Inputs:
      - `x` - the robot's current state, 1×3 numpy array
      - `start` - start position for the test, 1×3 numpy array
      - `goal` - goal position for the test, 1×3 numpy array
      - `robot` - the mobile robot class containing intrinsic parameters
      - `obstacles` - a group of n obstacle positions measured by the LIDAR, $n \times 2$ numpy array
    - Outputs:
      - `u` - desired wheel velocities, 1×2 numpy array
  - `select_wheel_vel`: the function receives the net force calculated by `Planner_update` and computes the desired wheel velocities. The function has:
    - Inputs:
      - `F` - a vector corresponding to the net force in the potential field, 1×2 numpy array
      - `x` - the robot's current state, 1×3 numpy array
    - Outputs:
      - `u` - desired wheel velocities, 1×2 numpy array

- Other important notes:
  - The way that the robot state is measured is shown in the following figure. Pay attention that θ in the state is the angle between X axis (the red line in Gazebo environment) and the robot's front axis.

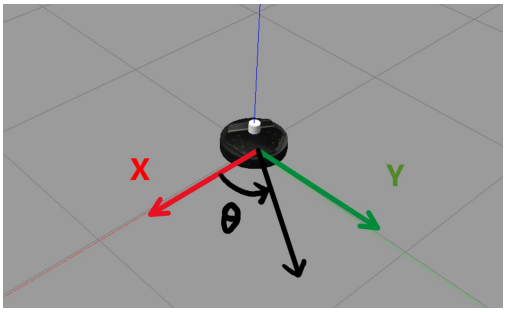

Figure 2: Robot State in Gazebo

  - When you compute wheel velocities, be sure to check the output values. The appropriate range of the wheel speed is set to be [-20, 20] (r/s). Tune your parameters if the output is too large/small. Also, the robot model represents the left/right wheels with respect to people's view rather than the robot's view. That is to say, for example, when right wheel speed is larger than left wheel speed, it would turn RIGHT (clockwise) rather than left. Knowing the above information is very helpful when you run your tests and debug code.
  - Although the function format and skeleton are provided to you, you could choose whatever way to implement as long as it works with the main simulation. As always, remember to keep the code clean and well-documented when you submit.

4 Simulation

Similar to previous labs, in order to run your code in simulation, you will first need to open a terminal, and run `meam520_update` which will update the necessary files in the system. Note that if you receive the error "Your local changes to the following files will be overwritten by merge", you should run `git stash` from `/meam520_ws/src`. After that, run `meam520_mobile` to initialize the Gazebo simulation environment as shown below.

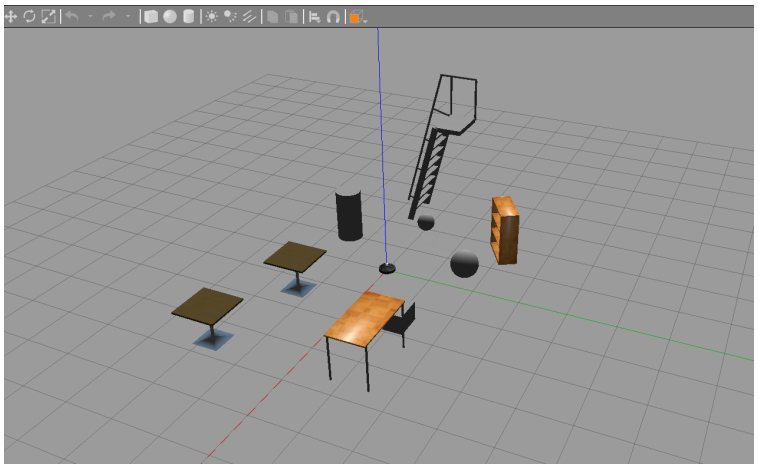

Figure 3: Simulation Environment

Inside the environment there are several objects that have been set up, such as the bookshelf, the tables, and some basic object models (box, cube, cylinder). It is the default environment that you will test your code in throughout the lab. However, to make it more interesting, you can also modify your own environment by inserting models (find "Insert" tab on the left) or deleting them (select a model, right click, find "Delete").

4.1 Test for Static Environment

- Run `runsim.py` in Spyder or by the terminal. Design a few tests, change start and end positions to check whether your algorithm works in the static environments. Here are a few examples:

Test	Start ([x, y, θ])	Goal ([x, y, θ])
1	[0, 0, 0]	[-2, 1, 0]
2	[-2, 1, 0]	[-5, -2, 0]
3	[-5, -2, 0]	[3, 2, 0]
4	[3, 2, 0]	[4, -3, 0]
5	[4, -3, 0]	[0, 0, 0]

- You can edit other test conditions in `runsim.py` if needed, and you may want to tune the planner parameters to improve performance. Report your test results (with tables and/or figures) in the report. In the meantime, we suggest that you take a look at section 4.3 to begin thinking about the questions.

  If the above simulation goes on well, proceed to simulation for dynamic environments.

4.2 Test for Dynamic Environment

- We have provided a model plugin to the Gazebo world to make the obstacles move in the environment. A plugin in Gazebo is a chunk of code that is inserted into the simulation and it allows us a complete access to the physical properties of the model.

- To run the simulation with the plugin code, you will need to add it to the Gazebo plugin path. First, you simply find the path for the `build` folder inside `Lab4/plugin`. Then, open a terminal and type the command (for the following example the path is `~/Lab4/plugin/build`, you should replace that with your own):

```
export GAZEBO_PLUGIN_PATH=${GAZEBO_PLUGIN_PATH}:~/Lab4/plugin/build
```

After including the path, run `meam520_mobile` and you will see the two sphere models in Fig 3 now moving around. Note that every time you start the simulation in a new terminal, you should include the path once again. Otherwise, the default simulation would be static.

- Similar to running tests in static environments, you run `runsim.py` and change start/goal positions to see how your planner works. Try to design some tough tests where the robot is likely to hit the moving obstacles. Report your results and think about the questions.
- **Optional:** If you are interested, take a look at `animated_model_1.cc` and `animated_model_2.cc` under `plugin` folder. In the code we provided two ways to move the Gazebo model: one is to make it follow a pre-defined trajectory, the other is to set a linear velocity.

  You may edit the code to modify the motion of the dynamic obstacles. But before you do that, make sure you fully understand what the code is doing. The **official tutorials** might be a good reference for you to look at.

4.3 Questions to Think About

For tests in static environment, consider questions as follows:

- How often does your path planner succeed? Cite your test results as a support to your answer.
- What kinds of tests/situations did your planner work well for? What was it bad at?
- Did you use an approach to escape local minima? If you did, describe your method and how well it worked. If not, think of some ways to tackle the problem (you are not required to implement this).
- When might you choose to use a potential field over your Lab 3 planner, or vice versa? In your answer, compare their advantages and disadvantages and provide detailed analysis.

Include dynamic obstacles in the simulation. Consider the following questions:

- How well did the planner work when you introduced dynamic obstacles?
- Were there limits to how the robot and the obstacles encountered so that the robot could avoid collision?
- What parameters did you change in your potential field planner to suit tests in the dynamic environments? Explain why and how it affected your planner performance.

5 Submission Instructions

You should submit a **pdf report** on Gradescope containing your solutions and analysis for these tasks, as well as a **zip file containing all of your code**. These should be 2 separate submissions.

5.1 Report

Your report should include:

- Your answers to the conceptual questions (method description, pseudocode, etc).
- Your up to 4-pg analysis of your experimental results and the answers to the Questions to Think About in the Coding Assignments section.

The format of the report is up to you, but you should make sure that it is clear, organized, and readable.

5.2 Code

Clean up your code. The TAs will need to examine your code to ensure it is correct, part of your grade for this lab will be based on coding style. Make your code easy to follow by including comments in your code using meaningful function and variable names. Delete lines of code which have been commented out.

Your submission should include **all of your code** in the `code` folder. You do not have to submit the plugin files.