Lab 3: Trajectory Planning for Mobile Robot

MEAM 520, University of Pennsylvania

July 8, 2020

Before starting, read the General Lab Instructions carefully!

This lab consists of a lab report due by midnight on **Monday, July 20, by midnight (11:59 p.m.)**. Late submissions will be accepted until midnight on **Thursday, July 23** following the deadline, 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. 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!

1 Overview

In this lab we will be exploring trajectory planning using both graph and sampling-based approaches. You will implement one of these approaches yourself and and then generate a trajectory based on your way-points. You mobile robot would move following the path and you could use your controller implemented in Lab2. The purpose is to examine the effectiveness of both trajectory planning algorithms in different environments. You will think about the strengths and weaknesses of each approach.

For this lab, you may choose to implement **either** A* or RRT. We have provided you `.pyc` files to run simulation before you implementing your own planner. You **do not** need to implement both. To help guide your decision, it may be useful to know that MEAM620 (Advanced Robotics) covers graph-based planning and requires students to implement Dijkstra's Algorithm/A* for a quadrotor. If you are planning to take MEAM620, consider taking this opportunity to implement RRT. If you are not planning to take MEAM620, feel free to implement either planner, but please consider that implementing A* can be a really good exercise.

2 Concepts

Based on the path planning algorithms covered in lecture, devise a strategy that you plan to implement for this lab.

- Summarize your planning strategy. Which algorithm are you going to implement? Will you plan in the workspace or in configuration space?
- What algorithm will you use to plan a trajectory without colliding with any of the obstacles? How will you detect collision (for implementation we have given you `CheckCollisionOnce.pyc`, but how might you replicate this functionality?) What kind of trajectory generating strategy are you going to use? Will your trajectory converge to your destination in dynamic situation? What simplifying assumptions are you making, if any? If you are making simplifying assumptions, justify them.
- Write pseudocode for your planner before beginning the implementation. Include this pseudocode in your report.

3 Coding Assignment

Implement your planner. You will be filling in **either** `Astar.py` **or** `RRT.py`. You will also need to fill in **functions** in `mobile_path.py` to be able to generate your trajectory.

- Download the file `lab3.zip` attached to this assignment.
- Write your code in either `Astar.py` or `RRT.py`. Each function takes in a address of a map, configurations q_{start} and q_{goal} and the value of the robot radius. It should return an $N \times 2$ array containing the sequence of position values along your trajectory that you send to the mobile robot. Assume a constant time step between the waypoints in this list.
  - If you decide to write `astar.py`:
    - Use 8-connected grid for your searching algorithm.
    - You are welcome to use the functions implemented in this class in your planner if needed.
    - You may also use `CheckCollisionOnce.pyc`, if your code needs it. This function takes in two points **linePt1**, **linePt2** as `nparray` and one obstacle **box** as `list` and return boolean value representing whether collision happens or not.
  - If you decide to write `rrt.py`:
    - We **suggest** you call `CheckCollisionOnce.pyc` to detect collisions for a robot configuration q. This function takes in two points **linePt1**, **linePt2** as `nparray` and one obstacle **box** as `list` and return boolean value representing whether collision happens or not.
    - You may vectorize this function for multiple input configurations, if you so choose.
- We **provide** you an `OccupancyMap.py` for the planner to use. The inputs of `OccupancyMap.py` is the address of the map and the radius of the robot. It returns a boolean grid map showing the occupancy status of the grid. Minkowski sum is implemented in this class. You should read through the class before you start implementing your planner so you have a glance on what would be useful for you.
- Write your code in `mobile_path.py` to implement your trajectory generator.
  - Read through the documentation for the solution implementation of the Path class from Lab 2. You may either use this code or the code you wrote for your own implementation. Note that the provided implementation of the Path class suggests a trajectory that consists of linear paths between points. If you are planning to use a modified version of this class be sure to document the differences in your lab report
  - Complete the implementation of the `Trajectory` class by writing the `update` function. This function takes as an argument the current robot state, evaluates if it has reached a the next waypoint, and then constructs the path that it should be following to reach its next destination.
- Using the controller you wrote in Lab 2, as well as the planner and trajectory generator you just implemented run `mobile_sandbox` to run a simulation.
- The map for your code is a `.json` file and for the simulation is a `.world` file. You could simply load the `.json` file as a `dict` and access values inside it if needed. We have provided four maps for you to test. The parameters are listed as following:
  - `bounds`: [`X_min`, `X_max`, `Y_min`, `Y_max`]. A list representing the boundary of the world for the simulation. These denote the lower left corner and the upper right corner of the world.
  - `blocks`: [`X_min`, `X_max`, `Y_min`, `Y_max`]. A vector containing multiple obstacles in the environment. The notation is the same as that for `bounds`.

  - `resolution`: [`X_res`, `Y_res`]. A list containing the resolution in horizontal direction and vertical direction. These usually could be the same. You could play with these numbers to see how resolutions changing would affect your planning.

- The code will not be graded on its speed, but it will need to run fast enough for the graders to verify that it works. As a rule of thumb, running the planner on a map the size of our simulation world should not take more than 2 minutes on a regular laptop.

4 Simulation

We **strongly** recommend that you write and test your A^* or RRT first before attempting to run the simulation. We have provided you with obfuscated versions of RRT, A^*, and the controller. For the purposes of the comparison questions, you may use the solution version of whichever algorithm you did not write. If you are unable to get your planner or controller working, you may use the solution version for partial credit. Any attempt to de-obfuscate the solution code will be considered a violation of Penn's academic integrity policy and action will be taken as such.

- In the terminal, run the command `meam520_update`.
- Go to `/home/meam520_ws/src/turtlebot/turtlebot_simulation/turtlebot_gazebo/worlds`. Change the map you are using `playground_i.world` to `playground.world`. These are one-to-one correspondances with the `.json` maps.
- Run the command `meam520_mobile` in terminal to open gazebo.
- Use `spyder3` to open `mobile_sandbox`, hit `run` and you could see simulation live in Gazebo.
- To use the bytecode solution files, replace the relevant `.py` file with the appropriate `.pyc` file in the `pyc_code` folder.

5 Questions to Think About

Design a few tests (environments, start and end positions) to check whether your planning algorithms work. Explain why you have chosen those tests. Run each test several times to evaluate the performance of your planner. For each test run A* and RRT (you will only have written one, for the other use the encrypted solution code). Consider and answer the following questions for both planners.

- How often does your path planner succeed?
- How long does it take for your planner to find a path (running time)?
- When your planner finds a path, is the path the same over multiple runs?
- What kinds of environments/situations did your planner work well for? What kinds of environments/situations was it bad at?
- What would be the advantages and disadvantages for both planners in higher degree of freedom such as planning with the Lynx robot?

Test how your trajectory generator and controller work in conjunction with your planner by examining the results from the Gazebo simuation for the tests you run.

- How often does you software suite succeed in reaching the goal?
- What type of plans were difficult for your trajectory generator/controller to follow?

- How does your planned trajectory differ from the simulation results? What is the error like between the two?
- How do you expect this system would work if it were deployed on a physical system?
- What changes would you make to your implementation if you had more time with the lab?

You should plan to run tests for at least four maps (you can use the provided map files if you wish), plus any additional ones you want to consider. Include your maps in your submission zip. In order to create you `.world` map for gazebo, you could follow the instructions as below:

- One block of $< modelname >$ / $< /model >$ corresponds to one obstacle. The units for numerical value to change are in meters.
- Change the `pose` value in the block to the desired value. These are [X_o, Y_o, Z_o, yaw, pitch, roll]. The origin is the mass center.
- In the block $< collisionname >$ / $< /collision >$, change the geometrical box size to the desired values. The order are [X, Y, Z].
- In the block $< visualname >$ / $< /visual >$, change the geometrical box size to the desired values.
- Save the changes and relaunch gazebo by Running the command `meam520_mobile` in terminal.

6 Submission Instructions

You should submit a **pdf report** on Canvas containing your solutions and analysis for these tasks, as well as a **zip file containing all of your code**. These should be 2 separate files. **Do not include your pdf in the zip.**

6.1 Report

Your report should include:

- Your description and pseudocode for your planner. Please indicate which planner you chose.
- A brief description or list of function names to help us relate the pseudocode to your code.
- Your 2-pg analysis of your experimental results and the answers to the questions asked in the section Questions to Think About

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

6.2 Code

Your submission should be a zip file containing all the files needed to run the project. **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.