Lab 3: Trajectory Planning for the Lynx

MEAM 520, University of Pennsylvania

October 9, 2020

This lab consists of two portions, with a pre-lab due on **Friday, October 16, by midnight (11:59 p.m.)** and a lab (code+report) due on **Friday, October 23, by midnight (11:59 p.m.)**. Late submissions will be accepted until midnight on Saturday 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 50 points.

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. To help you actually learn the material, what you submit must be your own work, not copied from any other individual or team. Any submissions suspected of violating Penn's Code of Academic Integrity will be reported to the Office of Student Conduct. When you get stuck, post a question on Piazza or go to office hours!

Individual vs. Pair Programming

Work closely with your partner throughout the lab, following these guidelines, which were adapted from "All I really needed to know about pair programming I learned in kindergarten," by Williams and Kessler, *Communications of the ACM*, May 2000. This article is available on Canvas under Files / Resources.

- Start with a good attitude, setting aside any skepticism, and expect to jell with your partner.
- Don't start alone. Arrange a meeting with your partner as soon as you can.
- Use just one setup, and sit side by side. For a programming component, a desktop computer with a large monitor is better than a laptop. Make sure both partners can see the screen.
- At each instant, one partner should be driving (writing, using the mouse/keyboard, moving the robot) while the other is continuously reviewing the work (thinking and making suggestions).
- Change driving/reviewing roles at least every 30 minutes, *even if one partner is much more experienced than the other*. You may want to set a timer to help you remember to switch.
- If you notice an error in the equation or code that your partner is writing, wait until they finish the line to correct them.
- Stay focused and on-task the whole time you are working together.
- Take a break periodically to refresh your perspective.
- Share responsibility for your project; avoid blaming either partner for challenges you run into.
- Recognize that working in pairs usually takes more time than working alone, but it produces better work, deeper learning, and a more positive experience for the participants.

1 Pre-lab Tasks (due October 16, 5 pts)

Directions: For the pre-lab component of this lab, you turn in your own **individual** work. Show your work to receive full credit. These calculations should be typed or written legibly. Submit a **pdf** on Gradescope containing your work.

This lab is about planning collision-free trajectories through a cluttered environment using the Lynx robot. Consider the Lynx robot trying to move from an initial configuration q_{start} to another configuration q_{end}. The environment has rectangular block obstacles in it, which can be described by their extreme coordinates $[x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}]$. The goal is for the robot to move from the start point to the end point without colliding with any of the obstacles.

You will implement an RRT planner and then examine its effectiveness compared to A* (which we will provide) in different environments. To plan out your approach for the lab, answer the following questions:

- Will you plan in the workspace or the configuration space?
- How will you represent the space that you are planning in?
- How will you check for collisions (with the end effector? with the rest of the robot?)
- How will you account for the geometry (volume) of the robot?
- How will you compute waypoints?
- How will you interpolate between waypoints?

Explain your choices.

2 Lab (due October 23, 45 pts)

The remainder of the lab should be done with a partner. You may work with anyone you choose, but you must work with them for all parts of this assignment. You will both turn in the same report and code (see Submission Instructions), for which you are jointly responsible and you will both receive the same grade.

2.1 Methods

- Summarize your planning strategy. If you are making simplifying assumptions, justify them.

  (Note: you should have already described your initial plan in your pre-lab. The purpose of this section of the report is to describe the planner you ended up actually implementing, and in particular to make note of any changes from your initial plan. The description should stand alone, since the TA reading your report may not be the same one who read your pre-lab. If you decide that your pre-lab description was sufficient, you can simply copy it here.)

- Include pseudocode in your report.

2.1.1 Robot Dimension

The robot dimensions are the same as in Lab 1 and 2. Additionally. the gripper's fingers are 12.5 mm long from center to tip.

2.2 Coding

2.2.1 Simulation Environment Updates

We've made a few tweaks to the simulation environment, and you must take the following steps to have an up to date simulator. **These steps are needed for all students!**

1. **Update the Gazebo simulator:** On the Virtual Machine, open a terminal and run the command

   `cd ~/meam520_ws/src/meam520_sim && git pull`

   This will update the code on your machine to match the current version.

2. **Update the Core:** From Canvas, redownload the `Core.zip` file for your respective language and extract it in the same location as for Lab0.

3. **(Gazebo Pro Tip:)** Right-click on the robot and select 'Move To' to automatically zoom your view into the robot. Hold shift, click the robot's base, and drag to reorient the view.

2.2.2 Your Tasks

- Download the file `lab3.zip` attached to this assignment.
- Implement your planner in the file `RRT`. The function takes in a map struct (see below) as well as two configurations q_{start} and q_{goal}, which are the starting and ending configurations of the robot, respectively. It should return an $N \times 6$ array containing the sequence of joint variable values along your trajectory that you send to the Lynx robot.

  To keep your code organized,

  - We **suggest** you separately fill out and call a function `isRobotCollided` to detect collisions for a robot configuration q.
  - We have provided a function `detectCollision` (see below) to help you with this check. You may change the function `detectCollision` if it does not serve your purposes in its basic form. Include a copy of the `detectCollision` file in your submission.

- 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 our sample map should not take more than 15 minutes on a regular laptop.
- `lab3.zip` additionally contains the following additional files you may use:

  - `Astar`: Our implementation of the A* planner for the Lynx. The function operates in the same way at your `RRT`, taking as input the map struct and two configurations q_{start} and q_{goal}, and outputting a $N \times 6$ array of waypoints.

    (Note A* is a complete planner and searches the entire configuration space. It may take a while depending on your computer's capabilities. If needed, you can change the resolution of the search in (Python: astar.py, line 28 (replace `[0.1,0.1,0.1,0.3]` with your desired resolutions for joints 1-4); MATLAB: astar.m, line 34 (replace `[0.1,0.1,0.1]` with your desired resolutions for joints 1-3))
  - `loadmap`: loads in a map from a text file and stores it as a map struct. Usage: `map = loadmap(filename)`, where `filename` is a string indicating the location of the text file.

    The map struct contains two entries: obstacles and boundaries. Both of these are stored as `[xmin,ymin,zmin,xmax,ymax,zmax]` arrays.
  - A folder `maps`: Containing various test maps

- `detectCollision`: Detects collision between a line segment and a rectangular prism. Usage: `iscollided = detectCollision(linePt1, linePt2, box)`, where `linePt1` and `linePt2` are $N \times 3$ arrays, with each row $[x, y, z]$ being the start/end locations of the line segment, and `box` is an array of the form `[xmin,ymin,zmin,xmax,ymax,zmax]`. Returns a Boolean `iscollided`, which is true if the line segment intersects with the box and false otherwise. Keep in mind that this assumes a **line of zero thickness**.
  - `detectCollision` is vectorized to accept any number of line segments, but only one box per call. If you call it with multiple lines, `iscollided` will be row-aligned with the line segments.
- FK code for the Lynx

2.2.3 Testing your code

We have given you sample test code `TestPath_Sim` that will call either the A* or RRT planner and send the outputted waypoints to the simulator.

In order to run your code first place the maps folder into `/meam520_ws/src/python_code/Lab3/` on the virtual machine (Note: if you followed the MATLAB setup instructions, you will need to create this folder. ROS will pull the maps from this specific location).

Modify `TestPath_Runsim` so that it loads in the correct map file, for example `mapFoo.txt`. Then launch the simulation using:

```
$roslaunch al5d_gazebo lab3.launch map_name:= mapFoo.txt
```

This will automatically process the map file `mapFoo.txt` and load it into Gazebo.

You will need to modify the loop in `TestPath_Sim` ($\sim$ line 45 in MATLAB and line 49 in Python) to decide when to send the next waypoint in the path. Currently it is coded to immediately send the next waypoint. The timing will need to depend on the distance between waypoints and the time needed to move between them.

2.3 Evaluation

Design a few tests (environments, start and end positions) to check whether your planner work. Explain why you have chosen those tests. Run each test several times to evaluate the performance of your planner and compare it against the A* planner we have provided. Consider questions such as:

- 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?
- How do the answers to these questions change for different environments or variations in your implementation?

Include your maps in your submission.

2.4 Analysis

Discuss any conclusions on your planner based on the evaluations. For example,

- What kinds of environments/situations did your planner work well for? What kinds of environments/situations was it bad at?
- What issues did you run into or what differences were there between your expected and experimental performance?
- What changes would you make to your implementation if you had more time with the lab?

What differences do you notice between the grid based planner (A*) and the random search (RRT)?

3 Submission Instructions

Submit the assignment. One person from each pair should submit code and a pdf copy of the report to the Gradescope assignments for Lab 3. As with Lab 2 please submit the report to Lab 3 Report and code to Lab 3 Code. After selecting the files and uploading them, the website will take you to the next page, where in the top right corner you should add your group members. If you do not add your group members they will not get credit for the assignment.

3.1 Report

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

1. Your answers to the conceptual questions in the Methods section, typed or legibly hand-written.
2. A short 1-pg description of how the concepts are incorporated into your code. Include pointers to important line numbers of subfunctions in your code. This will help the graders understand and provide feedback on your work. (This description can be bulleted. No need to use full sentences.)
3. Your experimental results, including a description of your experimental setup (i.e., what were your inputs) and collected data.
4. Your analysis comparing expectations against reality and extrapolations to general conclusions you would make from this lab.

3.2 Code Submission

Your code should be cleaned up so that it is easy to follow. Remove any commented-out commands that you are not using, and add comments to explain the tricky steps. Clearly indicate which parts of the code correspond to which parts of the lab. Your code submission should include:

1. Your `RRT`.
2. Any additional functions needed to run your code not included in the original code.
3. Your sample maps.

Each file should be attached separately to Gradescope. Do **not** zip them into a single file attachment.