MAE 4180/5180, CS 4758/5758, ECE 4180/5772

Autonomous Mobile Robots: Lab #4

Motion Planning

Instructor:
Dr. Hadas Kress-Gazit

Objectives

In this lab, students will experiment with planning collision-free trajectories in a known map using potential functions and rapidly-exploring random trees (RRTs). Potential functions, known as "multi-query" planners, plan globally over a workspace while RRTs are known as "single-query" planners because they aim to find a path from a *specific* start point to the goal point. The purpose of this lab is to give an understanding of how different motion planners work in practice when the robot is not an idealized holonomic point robot.

Prelab - Tuning the potential function

To run this lab, you are provided with two types of maps corresponding to the same physical setup. We have divided the workspace into two sides—yellow and orange— and you will be assigned one color before the lab starts.

- The files `labSphereMap_wall_yellow.mat` and `labSphereMap_wall_orange.mat` contain:
  - `map`: the definition for a sphere world. Column 1 and 2 contain the X and Y coordinates of the centers of the spheres, and column 3 contain the radius. The first line describes the boundary of the workspace while all other lines describe the obstacles.
  - `goali`: (x,y) position of the goal where i is the goal number.
- The files `labBoxMap_wall_yellow.mat` and `labBoxMap_wall_orange.mat` contain:
  - `map`: a matrix of the coordinates for all walls in the environment $[x_1, y_1, x_2, y_2]$. The first four rows of `map` define the walls of the environment outer boundary. The remaining rows define the walls of the interior obstacles.
  - `goali`: (x,y) position of the goal where i is the goal number.

In preparation for the lab, please download the lab maps, and tune the potential function as described below (one set of parameters that do not create local minima and one set that does). In the lab roster spreadsheet, we will indicate which color and which goal you should be using. Once the potential function is tuned, generate the plots listed below.

Required Code

- `potentialPoint.m`
- `potentialPlot.m`
- `potentialPlanner.m`
- `rrtPlanner.m`
- Any functions required to run the above (e.g. `limitCmds.m`, `robot2global.m`, etc...)

Required plots (to be shown to the TAs at the beginning of the lab)

In the following we omit the color of the map in the map file.

- A plot of a potential field **with no** local minima for the **lab sphere map** (`labSphereMap_wall.mat`).
- A plot of a potential field **with** local minima for the **lab sphere map** (`labSphereMap_wall.mat`).

- A plot of the robot trajectory in the simulator using potential function controller for the **lab sphere map** (`labSphereMap_wall.mat`). You may use the file `labBoxMap_wall.txt` for the simulator.
- A plot of the robot trajectory in the simulator using an RRT for the **lab polygon map** (`labBoxMap_wall.mat`).

---

1 Lab Manual

In the following we omit the color of the map in the map file name.

1.1 Station Set-up

(a) Unplug your iRobot Create. Make sure to only pick up the robot from the *bottom* and not the sensor platform. **Be careful not to hit the markers on the top of the robot.** Put it on the floor next to your lab station.

(b) Take note of the name of your robot.

(c) Turn on the robot by pressing the power button. The Raspberry Pi takes about 20-30 seconds to boot.

(d) In your MATLAB Command Window, run `Robot = CreatePiInit('robotName')` where `robotName` is the robot name from step (b). The initialization process creates the variable `Robot`, which contains the port configurations for interfacing with the Create and the added sensors and the robot name; it has five elements:

   - `Robot.Name` contains the robot name.
   - `Robot.OL_Client` used for getting the robot pose ground truth from the Optitrack system.
   - `Robot.CreatePort` used for sending commands and reading sensors from the robot.
   - `Robot.DistPort` used for getting depth information from the realsense camera.
   - `Robot.TagPort` used for getting tag information from the realsense camera.

(e) Check that you have connected to the robot properly by running `BeepRoombaCreate(Robot)`. You should hear your robot beep.

(f) Put your robot on the field. Make sure you can access the robot's localization information by running `[x,y,theta] = OverheadLocalizationCreate(Robot)`.

(g) If any of steps e-f fail, disconnect from the robot (run `CreatePiShutdown(Robot)`), shut it down, close Matlab, restart Matlab, turn the robot on and go back to step d.

   Important:

   - If the control function exits with an error, make sure to stop the robot by typing in the command window: `SetFwdVelAngVelCreate(Robot, 0, 0)`
   - When you are done working with the robot, or you wish to restart Matlab or the connection to the robot, first run `CreatePiShutdown(Robot)` to disconnect properly from the robot.

1.2 Potential Functions

(a) Place your robot on the field; make sure the overhead localization can see it. In your code, **make sure the max velocity is 0.1 m/sec**.

(b) For each group member, using the parameters you came up with for the lab map that created a potential function with no local minima, run your control program `potentialPlanner.m` from the homework assignment. Your robot should navigate to the goal without hitting any obstacles. Move your robot to approximately the same start location for each run. **For each group member's code**, save the trajectory data. Make sure you save or write down the values of c_{att}, c_{rep} and Q used to create each trajectory.

(c) **Increase the max velocity to 0.3 m/sec**. Run the control program with one of the parameters from part b. Save the trajectory data. Make sure you save or write down the values of c_{att}, c_{rep} and Q used to create the trajectory. You only have to do this once per group.

(d) **Change the max velocity back to 0.1 m/sec.** Using one set of the tuning parameters (c_{att}, c_{rep}, Q) that create local minima, run your control program `potentialPlanner.m`. Start the robot from a location that will cause it to get stuck in the local minimum. You only have to do this once per group. Save the robot's trajectory and the values of c_{att}, c_{rep} and Q.

1.3 Rapidly-Exploring Random Trees

(a) Place the robot on the field. Your robot will be moving to the same goal point as in the previous section.

(b) Edit your control program `rrtPlanner.m` to load `labBoxMap_wall.mat` and to build the tree towards the goal you've chosen. Have your program call `OverheadLocalizationCreate.m` to find the robot's current pose, and make that the RRT 'start' position. Make sure your program accounts for the radius of the robot.

(c) Run `rrtPlanner.m` and stop when the robot reaches its goal. Save the following: start and goal locations, the full search tree, the final solution path (waypoints), and the robot's trajectory from `datastore.truthPose`. Repeat for all group members.

(d) To make sure your algorithm ran as expected, plot the start and goal locations, the map walls, the full search tree, the final solution path (waypoints), and the robot's trajectory. (You may want to save this figure so you don't need to re-create it for the post-lab assignment.)

(e) Repeat (c)–(d) using a larger step size for your RRT (you only have to do this once per group).

(f) Repeat (c)–(d) using a smaller step size for your RRT (you only have to do this once per group).

(g) Repeat (c)–(d) for **each** member of your group, but choose different start points.

---

2 Post-Lab Assignment

Remember to submit your assignment as a group on Gradescope. To form a group:

1. **One** individual from the group submits the PDF on Gradescope.
2. When you click on your submitted assignment, there will be a section denoted by "Group" below the assignment name. Click on "View or edit group."
3. Add student(s) by selecting their name(s) from the drop-down menu under "Add Student."

2.1 Potential Fields (50 Points)

(a) For each group member, plot the potential field using the values of c_{att}, c_{rep} and Q chosen in part 1.2(b) to avoid local minima. On the same figure, plot the map and the corresponding trajectories from part 1.2(a). Did the robot behave as you expected? Why or why not?

(b) Comment on the differences between each members' trajectories. Which combination of c_{att}, c_{rep} and Q seemed to perform the "best"? Why?

(c) Did the robot hit a wall? What are a few reasons why the robot might hit a wall (even if yours didn't), when the potential field planner is specifically designed to avoid obstacles?

(d) For each group member, choose one of the trajectories from 2.1(a) to re-create with the simulator, as you did in HW6. You should use your control program `potentialPlanner.m` and the same values of c_{att}, c_{rep} and Q. (Try to start the robot as close as possible to the same starting location!) Plot both trajectories (the one from lab and the simulated one) on the same plot. Comment on the differences between the two trajectories, and discuss the possible causes.

(e) Plot the potential field using the values of c_{att}, c_{rep} and Q chosen in part 1.2(c) (changing the max velocity in the control function). On the same figure, plot the map and the corresponding trajectory. What effect did increasing the velocity have on the trajectory?

(f) Plot the potential field using the values of c_{att}, c_{rep} and Q chosen in part 1.2(d) to create a local minimum. On the same figure, plot the map and the trajectory from part 1.2(d). Did the robot behave as you expected? Why or why not?

2.2 Rapidly-Exploring Random Trees (50 Points)

(a) Choose a run from Section 1.3(c), plot the map walls from `labBoxMap_wall.mat`, the full search tree, the final solution path (waypoints), and the robot's trajectory. Also plot the start and goal positions.

(b) Plot the robot's trajectory and the map `labBoxMap_wall.mat` but now with a robot that has a radius of 0.16 m (instead of a point robot). Did the robot ever cross a boundary? If so, why? If not, what was the closest the robot came to an obstacle?

(c) How closely did your robot follow the solution path? Explain any deviations.

(d) Explain how changing the step size in parts 1.3(e)–(f) affected the RRT and the robot's trajectory.

(e) Describe any differences between group members' RRT code. Was there a noticeable difference in speed? Why?

(f) Compare the trajectory created using one of the potential function to one of the RRT solutions.

2.3 Lab preparation (mandatory, not graded)

(a) For the code you wrote as part of the pre-lab, were there any coding errors that were not discovered when using the simulator?

(b) How much time did you spend debugging your code in the lab?

(c) Did you observe any behaviors in the lab that you did not expect, or that did not match the simulated behaviors?

(d) What was the contribution of each group member to the lab and the report?