ECE 383/MEMS 442/ECE 590 Lab 3: Motion Planning

Due date: 10/18/2015 at 5pm

Instructions: This assignment is to be completed *individually*. You are allowed to discuss the problems with your classmates, but all work must be your own. You are not permitted to copy code, take written notes, or look up online solutions.

All coding will be done on the Klamp't Docker container on http://vm-manage.oit.duke.edu/. You will need to study the Klamp't motion planning API at http://motion.pratt.duke.edu/klampt/pyklampt_docs/classklampt_1_1cspace_1_1CSpace.html and http://motion.pratt.duke.edu/klampt/pyklampt_docs/classklampt_1_1cspace_1_1MotionPlan.html.

To submit, copy your code from the browser to files lab3a.py, lab3b.py, and lab3d.py. You will also be creating a lab3.pdf file. Submit all of these on Sakai.

Problem 1: Potential Fields

Lab 3A displays a point robot (blue sphere) in a field of obstacles (dark grey cylinders). The goal position is the red sphere. On every iteration, it computes a step of a potential field method, and the trace will be displayed on screen.

Implement an obstacle avoiding potential field. To calculate the virtual force for the repulsive field, you must calculate the direction and distance to each obstacle. Tune the strength of the attractive potential field, the parameters of the repulsive potential field, and the time step to get "reasonable" performance for a large sampling of random start points, as returned by the start() function.

Now make an adjustment to the field of obstacles that causes your tuned method to fail to reach the goal due to a local minimum. To do so, comment out the existing return statement in the obstacles() function, and adjust the obstacles appropriately.

Problem 2: Sampling-based Motion Planning

Lab 3B shows a cylindrical robot moving in a square planar domain amongst cylindrical obstacles. On each iteration, it performs some number of steps of a sampling-based motion planner. The roadmap is drawn in yellow. When a plan is found, it is displayed in blue.

The feasibility test in CircleObstacleCSpace is incorrect because it only considers collisions between obstacles and the center of the robot. Implement a correct feasibility test so the path does not cause the robot to actually collide with the boundaries of the region and the obstacles. [When you are convinced that your feasibility test is correct, copy this code to lab3b.py and continue to the next problem.]

Problem 3: Empirical Performance Testing

This problem still uses Lab3B, but here you will investigate how performance differs between different motion planners. These settings are specified in the MotionPlanning.setOptions calls in the makePlanner() function.

A. Investigate the difference between the 'prm' and 'rrt' settings, which switch between the PRM planner and RRT planner. Tune the parameters knn (between 1 and 100), connectionThreshold (between 0.01 and 1.0), and perturbationThreshold (between 0.01 and 1.0) and investigate how they affect performance (i.e., planning time) and the shape of the solution path. Change the radius of the obstacle down to 0.2 and up to 0.35 and do the same investigation. [For each parameter, 3-5 values or so should give you a good sense of performance. For better response times, you will want to set drawRoadmap=False for most of these experiments.]

   Plot and describe the results of your testing in lab3.pdf (include figures or snapshots if appropriate):

   - Run the planner 20 times. How much variability do you observe between runs?
   - What is the performance difference between PRM and RRT? Qualitatively, how do the computed paths and roadmaps differ?
   - How do knn and connectionThreshold affect performance of PRM? How do they affect the shape of the solution path?
   - How does perturbationThreshold affect performance of RRT? How does it affect the shape of the solution path?
   - Suggest high-quality values for the planner type and parameters.

B. The second commented out block of MotionPlanning.setOptions calls will switch to an *optimizing* planner that does not halt after the first path is found. Returning this planner, along with optimizing=True, tells the server to continue planning to shorten path length.

   With the first line uncommented, the optimizing planner uses a "shortcutting" method that first calls a sampling-based planner, then randomly attempts to draw feasible straight line segments between points on the existing path. Devise a new start/goal and set of obstacles such that shortcutting does not always converge to an optimal path. Take a snapshot of the results and include it in lab3.pdf.

   Now, enable one of the other planning methods suggested in the comments.

   - The "rrt*" method continues growing a roadmap after the first path is found.
   - The "fmm*" method uses a modified grid method called the Fast Marching Method. It also progressively refines the grid so that the solution gets closer to optimal as more iterations are spent planning.
   - The random-restart + shortcutting RRT method repeatedly generates paths via RRT, and performs shortcutting.

Take snapshots to show that your chosen method does indeed converge toward an optimal path with your new problem.

Problem 4: Planning in Non-Cartesian Spaces

Lab 3D shows a translating and rotating mobile robot in a larger environment (10m x 10m). The SE2ObstacleCSpace implements a configuration space which is in SE(2). It uses existing Klamp't functionality for performing collision checking, and path planning "works" in that it is able to compute feasible paths. However, these paths are not optimal, even when using optimizing planners, because they are unable to interpolate through a rotation of 0. (Pay attention to the robot's orientation).

Implement modified interpolation and distance functions in SE2ObstacleCSpace that will allow your method to account for the topology of SE(2) and correctly optimize paths.

(Note that FMM* will not work, since it is not set up for non-Euclidean spaces)