Project 2: Sampling Based Planning

CS 6370 / ME EN 6225 Motion Planning
University of Utah

Due: 15 Nov 2015 by 11:55pm

Instructions

All solutions should be uploaded via the assignment on Canvas.

Naming: Your upload should be named in the format *<uid>*.zip where *<uid>* is your Utah uid. The first level of the zip should contain a single folder with your uid as the name *<uid>*.

Code Organization: You should have a text file called README.txt in the root directory of your submitted archive which explains how to run the code submitted. All necessary files, like map or environment files should be included in your submission.

Written Answers: Place all written answers to questions as well as any images or other output data used to explain your responses in a single PDF document. This should be clearly named Proj*<number>*-answers.pdf, where number is the Project number (i.e. 2 for this assignment). Please make sure to write your name at the top of this document!

Sampling Based Planning (85 total points)

We will focus this project and implementing and analyzing some of the sample based planning methods discussed in class.

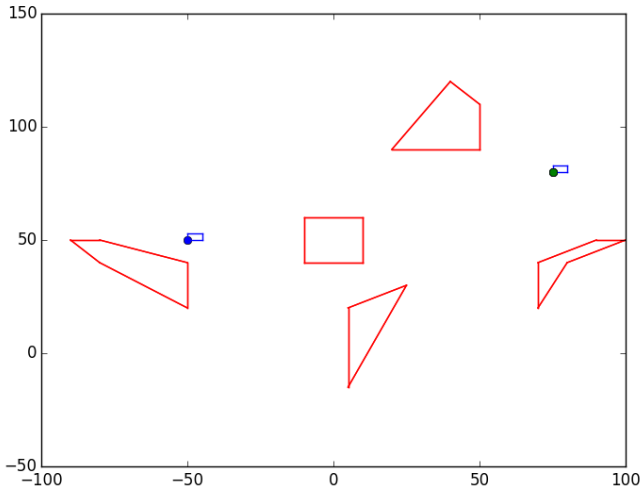

Figure 1: Environment 0 with start and goal configurations drawn.

To help you with getting started there is an included python file `collisions.py` which has some classes defined to help you focus on the algorithm implementation. There is an additional file

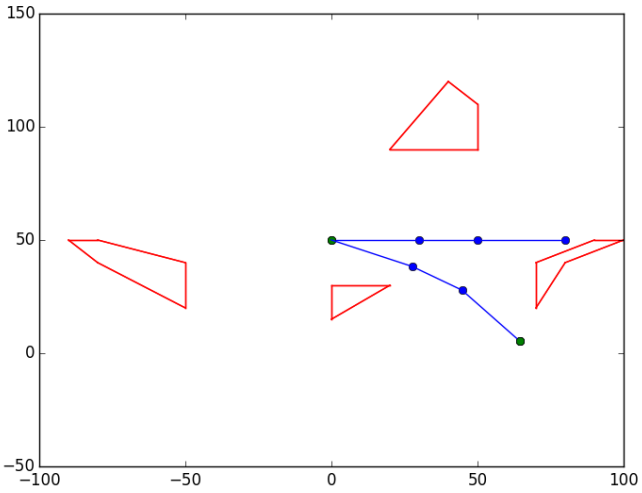

Figure 2: Environment 1 with start and goal configurations drawn.

named `PythonPointers.pdf` on Canvas which gives some links and simple examples to help people who are new to python. The code also contains function headers for the search methods you need to implement. Feel free to change the function arguments necessary to make it easier for you to solve the problems. There are two environments provided in the files `env0.txt` and `env1.txt`, these define the robot parameters the robot location and a set of obstacles. They also provided a default start and goal position, but your implementation should be able to change these values when necessary.

Your planner should be general to work between both the environments. The first one uses a simple box robot that can only translate, while the second environment has an RRR robot with a fixed base. We will focus on planning in the 2D plane were we have polygonal objects, which makes collision checking easy to implement. I've provided a simple robot class for an the box robot and RRR robot, which we will be using for the experiments. There is also a class that defines the environments and provides collision checking and simple visualization functions in `collisions.py`. I've also provided `rrt.py` which has a tree structure you can use in building your RRT, as well as some empty functions to fill out. You're on your own for setting up the PRM structures.

1 Rapidly-Exploring Random Trees (50 Points)

Implement RRT We'll start with implementing a traditional RRT and then build upon that with a few extensions. I should be able to run your code using a map file as described above and it should be clear how to change the transition function and actions used. You will need to be able to visualize the RRT tree and resulting plan/path for evaluation. For the three link robot, you can show a tree of the end-effector positions, to make visualization easier to understand. These functions are provided in the `PolygonalEnvironment` class. It may be

helpful to first plan and visualize these methods using a point in 2D space instead of with the full robot.

1.1 (20 pts) Implement an RRT. using the three-link robot in the plane. Use the two environments provided to plan a path from init to goal. Sample towards the goal instead of a random sample with 5% probability. Provide an image of both your tree and the resulting plan your algorithm finds for running on env0. Supply images of the resulting plan execution for env1.

For help debugging, I got a plan on env0 in about 1.8 seconds with a step size of 2. For env1 I used a step size of 0.15 and needed at least 5000 samples to get a plan, but sometimes more samples. This would sometimes take over 2 minutes to run.

Run your RRT for both environments again with three, significantly different start and goal positions for each and again provide images of the tree and plan as before.

1.2 (10 pts) Compare the trees and plans produced by sampling towards the goal with 0%, 5%, 10%, and 50% probability for env0.

1.3 (10 pts) Extend your RRT implementation to create an RRT connect, replacing the standard `build_rrt()` function with `build_rrt_connect()` which continues to extend towards a given sample instead of only making a single ϵ length step until a collision occurs. Compare the resulting tree from running RRT-Connect to running the standard RRT for both env0 and evn1. Use a 5% goal sampling bias for both versions. Again provide the images of resulting tree and plan found by your algorithm.

1.4 (10 pts) Now using your RRT connect implementation build a bi-directional RRT where you build two RRT-connects alternatively growing each one. Replace growing towards the goal with growing towards the other tree. Compare the resulting graph / trees and plan found to those found in the other versions above.

2 Probabilistic Roadmaps (30 Pts)

Implement PRM You now need to implement a PRM using the same robot and map models as above. Use a simple straight-line planner that checks collisions at a step size of ϵ for a local planner (hint use epsilon equal to worked well for your RRT).

2.1 (20 pts) Implement the standard PRM using the same robots and environment as above. Build the PRM on the same maps, but now show results for planning with multiple start and goal locations, while building only a single roadmap. Show the roadmap built before attaching goals and initial positions, as well as the plans found for each of the different test situations. How does the resulting map change with different number of samples? How about changing the number of nearest neighbors used in building your roadmap.

2.2 (10 pts) Implement the Gaussian sample based PRM, where you sample a second point from a Gaussian distribution centered at the first sample. Only keep the resulting configuration if one sample is in an obstacle and one is outside the obstacle. Compare the resulting roadmaps you get on the two maps to using the standard PRM.

It could be helpful to use the `numpy.random.normal()` function to sample form a normal distribution.

2.3 BONUS 15 pts Now use your standard RRT implementation from problem 1 as the local planner in your PRM. Provide an image of your resulting roadmap. How does changing the number of samples change the performance of the PRM with an RRT local planner? How does the number of nearest neighbors affect performance?

3 Self Analysis (5 pts)

3.1 (1 pt) What was the hardest part of the assignment for you?

3.2 (1 pt) What was the easiest part of the assignment for you?

3.3 (1 pt) What problem(s) helped further your understanding of the course material?

3.4 (1 pt) Did you feel any problems were tedious and not helpful to your understanding of the material?

3.5 (1 pt) What other feedback do you have about this homework?