CS 294-115 Algorithmic Human-Robot Interaction **Fall 2016**

Lecture 4: Motion Planning, Part III

Scribes: Xinlei Pan, Daniel Seita

4.1 Administration and Review

We will set up Piazza for this class. In addition, we will eventually have PRO/CON presentations on research papers, and we will also talk about final project ideas. But for now, we will hold off on those and finish motion planning in this lecture.

In our last lecture, we talked about *visibility graphs* and *grid search*. Recall the pros and cons of these algorithms. Visibility graphs are optimal and complete, but make the overly restrictive assumptions of polygonal obstacles and an explicit representation of C_{obs}. Grid search does not need to make those two assumptions, as it represents the configuration space with a grid and tries to connect points corresponding to grid elements. While grid search is *resolution* complete and *resolution* optimal, it technically does not satisfy the stronger qualities of being *complete* or *optimal*.

While these two algorithms are well-known in motion planning, their research importance nowadays may be due to historical reasons rather than prevalence. Currently, about 90 to 95 percent of research labs in this country use a third category of motion planning algorithms, based on *sampling*.

4.2 Sampling-Based Planners

These represent C_{free} from sampled configurations.

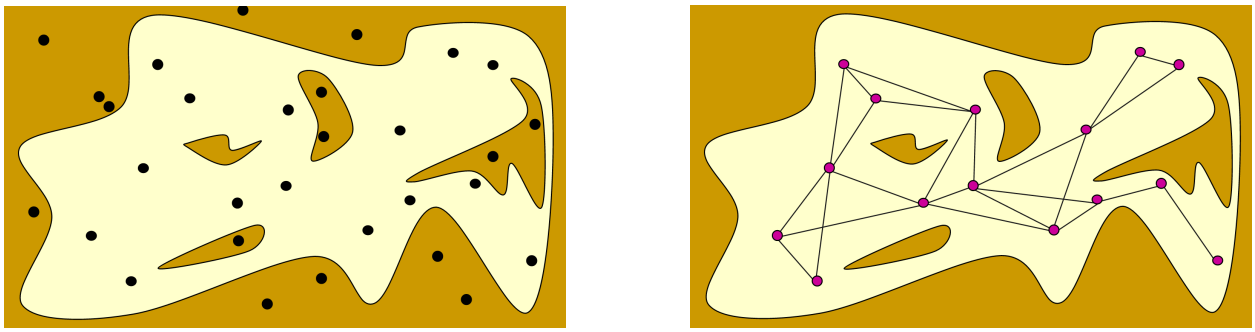
4.2.1 Probabilistic Roadmap (PRM)

The PRM algorithm [due to Kavraki et al] is a well-known sampling-based motion panning algorithm. It makes the following assumptions:

1. That there exists a collision checker similar to the function we had in grid search. That is, a function which takes in a configuration q as input and returns 1 if $q \in C_{\text{obs}}$, and 0 otherwise.
2. That we have a simple planner $B_s(q_1, q_2)$, which quickly returns some (not necessarily "optimal") path from q_1 to q_2, or reports failure. A typical case (in 2-D, at least) would be to return the straight line between q_1 and q_2, if possible.

Here are the steps of PRM:

1. Start with the start and goal configurations: q_s and q_g.
2. Sample M "milestones" in C_{free}, using rejection sampling to avoid points in C_{obs}.



(a) Sampled points.

(b) Connected Points.

Figure 4.1: An example of the PRM algorithm in action. Our configuration space consists of C_{free}, the yellow region, and C_{obs}, the brown region. In (a), the algorithm has sampled for candidate milestone points, but has not yet rejected the points in C_{obs}. In (b), the algorithm has only kept the points in C_{free} and has connected the resulting points with some of their nearest neighbors. (Note that for the sake of simplicity, there is no q_s or q_g indicated.) Image credit: CS 287, Fall 2015 Lecture Slides.

3. Try to connect all the milestones using B_s. Three options for doing so are:
   (a) Connect all points together. This is the most expensive of the three methods.
   (b) Only connect points with their k nearest neighbor points. This is called k-PRM.
   (c) Only connect points that are within distance R of each other. With respect to each point, this is like considering a disc of radius R centered at that point.

   For all these methods, the only connections should be those that pass the collision checker.
4. If q_s and q_g are in the same connected component, find the path using graph search. Otherwise, return to step 2 to sample more milestones and repeat.

See Figures 4.1a and 4.1b for a visual example of PRM.

What can we say about PRM?

1. (CON) It is not complete nor optimal.
2. (weak PRO) It is *probabilistically* complete, meaning that if the algorithm is run long enough, the graph will contain a solution (with probability one) if a path exists. More precisely, $\lim_{n\to\infty} \Pr(sol \mid \exists sol) \to 1$ where n is the number of iterations. But this property is weak.
3. (PRO) It is fast, scalable, does not make overly restrictive assumptions (made by algorithms such as visibility graphs), and as evident by its popularity nowadays, it works well in practice.

The PRM algorithm presented here is technically a newer version. The original PRM algorithm dealt with multiple queries, which does not use Step 1 of PRM (as listed earlier). The multi-query algorithm can, as suggested by the name, deal with multiple queries $\{(q_s, q_g)^{(i)}\}_{i=1}^{N}$ rather than a single (q_s, q_g), which can be handled by one pre-processing step. As an analogy suggested by Anca: we don't have our entire lives planned out ahead of us.[1] Why do we need fixed start and goal states?

[1] Well, with the exception of certain graduate students, but never mind.

There are many variations of PRM, and many ways to improve the algorithm for specific domains. For instance, PRM implicitly constructs a random graph, but we may wish to have a *weighted* graph to focus on more critical regions of the configuration space. In addition, it is unclear if the goal of connecting all our nearest points is the best one; certain connections may be redundant. Think of three points q_1, q_2, q_3 in a near straight line. It may or may not make sense for PRM to focus on connecting q_1 and q_3 if there already exists connections with $q_1 - q_2$ and $q_2 - q_3$. Anca wisely suggests implementing these algorithms in 2-D to understand them better and to test variations.

4.2.2 Rapidly Exploring Random Trees (RRT)

RRTs [due to LaValle and Kuffner] are another well-known sampling algorithm for motion planning. The steps for the most popular variant, *bidirectional* RRT, are:

1. Like PRM, we have a start configuration q_s and a goal configuration q_g.
2. Each iteration, sample with $M = 1$ milestones.[2]
3. Keep track of the connected components that contain q_s and q_g. Call these graphs G_s and G_q. Connect the newest milestone with the closest point in G_s and the closest point in G_g. (This step is similar to what one would perform in k-PRM with $k = 1$.) Notice that because $k = 1$, G_s and G_g are *trees*.
4. If G_s and G_g can be connected, we are done. Otherwise, we continue. We can sample one point and keep connecting them to the nearest neighbors in G_s and G_g, or alternatively, we can try to sample two points, one for G_s and one for G_g. The algorithm can also bias the sampling selection into unexplored regions.

The "original RRT" was unidirectional, with some minor sampling near the goal state. Under some circumstances, Bidirectional RRT can result in two trees that collectively consists of a substantially smaller portion of the configuration space than unidirectional RRT (and thus is more efficient).

There are *many* variations of RRTs, but Bidirectional RRTs are probably a good choice. In terms of their properties, their PROs/CONs are similar to PRMs:

4.2.2.1 "Original Original" RRTs: Kinodynamics

Kinodynamic motion planning is the problem of finding the optimal path for a robot with motion constraints (velocities and accelerations, etc.) from a starting configuration to a goal configuration, and the path has to be collision free. Kinodynamic RRT's goal is to satisfy both global obstacle constraints which is from RRT and local differential constraints (from kinodynamic motion planning). The variant we briefly mentioned in class assumes that we can run several trajectories of the robot for some number of steps, and we can keep the trajectory that results in the closest point to our goal and which passes the collision checker.

[2] If this seems worrisome to you, think about how this relates to Stochastic Gradient Descent (SGD), possibly the most common gradient-based learning procedure in machine learning nowadays.