Lecture 13
Sampling Based Methods (II)

Instructor: Jingjin Yu
Course Logistics Update (I)

Update on 2nd home work and midterm

⇒ Good news - they will be combined
⇒ Focus on practical aspects
  ⇒ The goal is to give you better understanding of basic motion planning algorithms
  ⇒ You should know how to implement them at a basic level
⇒ Two implementation tasks
  ⇒ One on vertical cell decomposition or shortest path roadmap
  ⇒ One on probabilistic roadmap (PRM) or rapidly-exploring random trees (RRT)
  ⇒ You will implement using either Python or Java
    ⇒ You may use math libraries (e.g., numpy) but not existing solutions
⇒ You need to implement the pieces
  ⇒ Combinatorial methods
    ⇒ Build the roadmap
    ⇒ Graph search
  ⇒ Sampling based methods
    ⇒ Sampling
    ⇒ Collision detection
    ⇒ Graph search
Course Logistics Update (II)

For both problems, assume you are given a rectangular region with polygonal obstacles, with (point) start \((x_I)\) and goal \((x_G)\), e.g.,

You are to provide (1) the roadmap and (2) the solution path

⇒ Both are a set of vertices and edges
⇒ More details to come later in the week
Today’s Lecture

Quick review of basics in sampling based planning methods
The general incremental sampling and search
Rapidly-Exploring Random Trees (RRT)
  ➞ Basic RRT planner
  ➞ Kino-dynamic

Optimal sampling based planning methods

Can be high dimensional
The Working of PRM (Kavraki et al.)

$C_{\text{free}}$, generally high dimensional
Generating Random Samples

Random sample
Rejecting Samples Outside $C_{free}$
Collecting Enough Samples in $C_{\text{free}}$
Connect to $k$ Nearest Neighbors (If Possible)
Connect to $k$ Nearest Neighbors (If Possible)
Query Phase
Path Smoothing
Key Components of Sampling Based Planners

PRM requires several important subroutines to work

- **An efficient sampling routine** is needed to generate the samples to $C_{\text{free}}$
  - Sample distribution is measured with **dispersion**
  - Sampling can be random or deterministic

- **Collision checking**
  - Uses collision detector as black boxes
  - Many choices, e.g., BVH

- A **metric** for determining how “close” two samples are to each other
- **Efficient nearest neighbor search** for building the roadmap graph
  - Based on the given metric
Incremental sampling and search often only needs to find a way to connect a single pair of start ($x_I$) and goal ($x_G$) for an environment. It has the following template:

- **Initialization**: create an empty graph $G = (V, E)$
  - Add $x_I$ or $x_G$ or both to the vertex set $V$
- **Vertex selection**: choose $x_{cur} \in V$ for expansion
- **Local planning method**: sample some $x_{new}$ and attempt to connect $x_{cur}$ to $x_{new}$ via a collision-free path
- **Edge insertion**: if a collision-free path $e(x_{cur}, x_{new})$ is found, insert it into $G$
  - Note that $x_{new}$ is also added to $V$
- **Check for a solution**: determine whether a solution exists given the current $G$
  - Return to the vertex selection step

Many different methods using similar schemes:

- PRM (what we already covered)
- RRT
Drawbacks of Multi-Query Methods

PRM is known as a “multi-query” sampling-based method because after initial roadmap is built, multiple queries can be executed on the same roadmap.

⇒ But, this also means that the roadmap is likely to have a lot of useless information stored if we want to run a single query.
⇒ People developed **single-query** methods to handle such situations.
⇒ One popular method is the rapidly-exploring random trees (LaValle & Kuffner).
Rapidly-Exploring Random Trees w/o Obstacle

RRT without obstacle simply grows a tree from a point

⇒ Basically, tries to connect new points to the closest part of the existing tree
RRT with Obstacles

When there are obstacles, try to extend the tree as much as possible.

Same procedure if sample falls inside an obstacle.
Tree Building Example
Improving Efficiency with Bidirectional Search

Recall in discrete search, e.g., BFS, when both $x_I$ and $x_G$ are known, we can run bidirectional search:

- Grow two trees, one from $x_I$, one from $x_G$.
- Basically, allocate similar effort to grow both trees.
- Keep growing until the two trees meet.

- Significant savings in high dimensions.
- Can also do multiple trees.
Recall that sampling based methods work well when the problems are relatively easy to solve. Sometimes, problems are just hard.

Difficult Cases

(a) $q_I$ $q_G$

(b) $q_I$ $q_G$

(c) $q_I$ $q_G$ $q_C$

(d) $q_I$ $q_G$
Solving Kinodynamic Problems

Besides solving single-query problems faster, RRT is suitable for solving problems for systems with **differential constraints**

⇒ For examples, a normal car is such a system

⇒ Standard PRM and RRT cannot be applied!
Kinodynamic RRT

We can grow an RRT respecting the differential constraints

- Standard PRM and RRT cannot be applied!
- Need to compute path more carefully (requires solving BVP)
- Example w/o obstacles
Kinodynamic RRT Examples (I)

RRT after 500, 1000, 1582 nodes
Kinodynamic RRT Examples (II)
Non-Optimality of PRM and RRT

PRM and RRTs are not optimal

⇒ It is possible construct instances to make PRM/RRT produce long paths

⇒ Can we do better?

⇒ Need to keep “re-wiring” the graph structure
Consider growing an RRT without obstacles

- For each new sample $x_n$, check its log $n$ neighborhood
- If there are better paths from $x_I$ to $x_n$, pick that path
- The resulting algorithm is called RRT* (Karaman & Frazzoli)

RRT* is an asymptotically optimal sampling based algorithm
- As the number of samples goes to infinity, an optimal path from $x_I$ to $x_G$ is obtained

Other optimal sampling based planners are available as well
- PRM* (Karaman & Frazzoli)
- Stable Sparse RRT (SST) (Bekris)
RRT* Example with Obstacles

RRT* after 500, 1500, 2500, 5000, 10000, 15000 Iterations
RRT* Example with Cost Variance

High cost region

Low cost region
RRT* Compared to RRT

RRT [top] and RRT*[bottom] after 250, 500, 2500, and 10000 vertices are added