Sampling-Based Motion Planning: From PRMs to RRTs

Instructor: Chris Mavrogiannis

TAs: Kay Ke, Gilwoo Lee, Matt Schmittle

*Slides based on or adapted from Sanjiban Choudhury, Steve LaValle, Peter Allen, Pieter Abbeel
Logistics

- Lab 3
  - Deadline Friday March 6th
  - Demo Thursday March 5th (recitation slots)
  - Extra Credit important for final project
- Final Project
  - Out today!
  - Demo Thursday March 12th
  - Short writeup due Monday 16th
- Special Topics
Explicit Geometry based planners impractical in high dimensional spaces.
Exact solutions with complex geometries provably exponential
Sampling-based planners can often create plans in high-dimensional spaces efficiently
Rather than Compute the C-Space explicitly, we Sample it
Completeness in Motion Planning

- **Complete Planner**: always answers a path planning query correctly in bounded time (return a path if one exists, otherwise report failure)
- **Probabilistically Complete Planner**: Probability of returning a path approaches 1 as more samples are generated.
- **Resolution-Complete Planner**: Probability of finding a path approaches 1 as the resolution becomes finer.
Do not attempt to explicitly construct the C-Space and its boundaries.

Simply need to know if a single robot configuration is in collision

Exploits simple tests for collision with full knowledge of the space

Collision detection is a separate module—can be tailored to the application

As collision detection improves, so do these algorithms

Different approaches for single-query and multi-query requests
This is the PRM Algorithm!

PRM = Probabilistic Roadmap

1. Sample vertices randomly and collision check

2. Try to connect vertices within radius (or k NN)
   - collision check potential edges

3. Search graph to find a solution
   - can reuse graph across multiple queries

Probabilistically Complete

What is the optimal radius?

What happens if radius too large? too small?
What is the optimal radius?

Set the radius to $r = \gamma \left( \frac{\log|V|}{|V|} \right)^{1/d}$ where magic constant!

$\gamma \geq 2(1 + 1/d)^{1/d} \frac{\mu(C_{\text{free}})}{\zeta_d}$

Also known as a Random Geometric Graph (RGG)
This is the PRM* Algorithm!

1. Sample vertices randomly

2. Use optimal radius formula to connect vertices

3. Search graph to find a solution

**Theorem:** Probabilistically complete AND Asymptotically optimal

“Sampling-based Algorithms for Optimal Motion Planning”
Sertac Karaman and Emilio Frazzoli, IJRR 2011
Can we do better than random?

Uniform random sampling tends to clump

Ideally we would want points to be spread out evenly

**Question:** How do we do this without discretization?
Halton Sequence

Generalization of Van de Corput Sequence

**Intuition:** Create a sequence using prime numbers that uniformly densify space

Link for exact algorithm: https://observablehq.com/@jrus/halton
How do we connect vertices?

Halton sequences have much better coverage
(i.e. they are low dispersion)

Connect vertices that are within a radius of

\[ r = \gamma \left( \frac{1}{|V|} \right)^{1/d} \]  

(as opposed to:)

\[ r = \gamma \left( \frac{\log |V|}{|V|} \right)^{1/d} \]
This is the gPRM Algorithm!

1. Sample vertices randomly

2. Use **optimal radius formula** to connect vertices

3. Search graph to find a solution

**Theorem:** Probabilistically complete AND Asymptotically optimal AND Asymptotic rate of convergence

“Deterministic Sampling-Based Motion Planning: Optimality, Complexity, and Performance”
Lucas Janson, Brian Ichter, Marco Pavone, IJRR 2017
What makes a good graph?

1. A good graph must be **sparse** (both in vertices and edges)

2. A good graph must have **good coverage**

3. A good graph must have the **same connectivity** of free space
The Narrow Passage: Planning’s boogie man!

Mathematical Question: How many samples do we need to connect the space (with high probability)?

Why is narrow passage mathematically hard to plan in?
How many samples do we need?

**Theorem** [Hsu et al., 1999] Let $2n$ vertices be sampled from $X_{\text{free}}$. Then the roadmap is connected with probability at least $1 - \gamma$ if:

$$n \geq \left\lceil 8 \frac{\log(\frac{8}{\varepsilon\alpha\gamma})}{\varepsilon\alpha} + \frac{3}{\beta} + 2 \right\rceil$$

The shape of free C-space is dictated by $\alpha$, $\beta$, $\varepsilon \in [0, 1]$

Visibility of free space ($\varepsilon$)  
Expansion of visibility ($\alpha$, $\beta$)

Narrow passage has small values of $\alpha$, $\beta$, $\varepsilon$

Hence, needs more samples to find a path
How do we bias sampling?

We somehow need more samples here

1. Sample near obstacle surface?
   V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. 1999

2. Add samples that are in between two obstacles?

3. Train a learner to detect the narrow passages?
   B. Ichter, J. Harrison, M. Pavone. Learning Sampling Distributions for Robot Motion Planning, 2018
# Summary of ways to create graphs

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>How to sample vertices?</th>
<th>How to connect vertices?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lattice</td>
<td>Discretize</td>
<td>connectivity rule</td>
</tr>
<tr>
<td>PRM</td>
<td>Uniform random</td>
<td>r-disc, k-nn</td>
</tr>
<tr>
<td>PRM*</td>
<td>Uniform random</td>
<td>optimal r-disc, k-nn</td>
</tr>
<tr>
<td>gPRM</td>
<td>Halton sequence</td>
<td>optimal r-disc, k-nn</td>
</tr>
<tr>
<td>Bridge</td>
<td>Sample with bridge test</td>
<td>any visible points</td>
</tr>
<tr>
<td>Gaussian</td>
<td>Sample near obstacles</td>
<td>r-disc, k-nn</td>
</tr>
<tr>
<td>MAPRM</td>
<td>Sample along medial axis</td>
<td>r-disc, k-nn</td>
</tr>
<tr>
<td>Approx. Visibility</td>
<td>Sample on surface of obstacles</td>
<td>any visible points</td>
</tr>
<tr>
<td>Graph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learnt Sampler</td>
<td>Use CVAE to approximate free space</td>
<td>optimal r-disc, k-nn</td>
</tr>
</tbody>
</table>
What graph should I use?

Low dim (2-3): Discretize evenly

Higher dim(>=4): Halton sequence

Narrow passage: Bias sampling
So far we have looked at an Explicit Evaluated Graph

Is it a good idea to evaluate every edge that we discover?

Is it a good idea to explicitly store the entire graph in memory?
Do we need to store this whole graph?

DIMACS dataset
1.5 million vertices, 3.67 million edges

A* search touches a small fraction

Bidirectional A*
Key Idea:
Use implicit unevaluated graphs
Implicit graph

1. Initialize with at least one seed vertex

   E.g. The start state, the goal state, both (for bi-directional search)

2. Provide a **generative** function for **producing successors**

\[
\text{succ}(u) = \{(u, v) | (u, v) \in E\}
\]

   function returns both
   - new vertices to add to V
   - new edges to add to E
Searching with implicit graph

1. Start the search with initial vertex (start, goal, or both)

2. Whenever search chooses to “expand a vertex”
   - call \texttt{succ}(u)
   - get successor vertices
   - evaluate the edges
   - add these vertices to the search queue
Explicit vs Implicit Graphs

- Explicit Graph:
  - All vertices are identified individually and represented separately.
  - All edges are identified individually and represented separately.

- Implicit Graph:
  - Only a subset, possibly only one, of the vertices is given an explicit representation. (The others are implied.)
  - Only a subset, and possibly zero, of the edges is given an explicit representation.
  - A set of "operators" is provided that can be used to construct "new" edges and vertices.
Planning with differential constraints
Differential constraints

So far we assumed only kinematic constraints

\[ q \notin C_{obs} \]

When is this assumption true?

- when controller can track any path

- when robots move very slowly (stop and turn)
Differential constraints

We now introduce differential constraints

\[ \dot{q} = f(q, u) \]

Two new terms:

1. Introduction of control space
2. Introduction of an equality constraint
Differential constraints make things **even harder**

**Holonomic constraints:** Constraints that can be integrated, i.e.

\[
\dot{q} = f(q, u) \quad \rightarrow \quad g(q, u) = 0
\]

**Non-holonomic constraints:** Constraints that can’t be integrated, i.e. i.e. the system is trapped in some sub-manifold of the config space
Differential constraints make things even harder

Emergency landing where UAV can only turn left

“Left-turning-car”

Non-holonomic constraints: Constraints that can’t be integrated, i.e.

i.e. the system is trapped in some sub-manifold of the config space
How do we incorporate differential constraints in our framework?

Only change the STEER function!
Recap: STEER function for geometric

\[ \text{steer}(q_1, q_2) \]

A steer function tries to join two configurations with a feasible path

\[ (1 - \alpha)q_1 + \alpha q_2 \]

Example: Connect them with a straight line and check for feasibility
STEER function incorporate dynamics!

\textbf{steer}(q_1, q_2)

A steer function tries to join two configurations with a \textit{dynamically} feasible path
STEER function incorporate dynamics!

Formally called the boundary value problem (BVP)

Find a control trajectory $u(t) \in U$

Such that $q(0) = q_1$, $q(t_f) = q_2$

$$\dot{q}(t) = f(q(t), u(t))$$
How do we come up with STEER function?

There are three possible cases

Case 1: We can analytically solve the BVP :) 😎

Case 2: We need to numerically solve the BVP. 🤔

Case 3: We can’t even solve the BVP! 😳
Case 1: We can analytically solve the BVP

Consider the dynamics of your racecar

\[
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{bmatrix} = \begin{bmatrix}
V \cos \theta \\
V \sin \theta \\
C \tan \delta
\end{bmatrix}
\]

\[|\delta| \leq \delta_{max}\]

Can we solve this \textit{analytically}?
Case 1: We can analytically solve the BVP

Yes! The solution is called the Dubins path 🧁

![Diagram showing the trajectories for two words with labels $R_\alpha$, $S_d$, and $L_\gamma$. The path consists of $R_\alpha S_d L_\gamma$.

Right Straight Left

$R_\alpha S_d L_\gamma$
Case 1: We can analytically solve the BVP

Dubins showed that ALL solutions had to be one of 6 classes

\{LRL, RLR, LSL, LSR, RSL, RSR\}.

Hence, given a query, evaluate ALL 6 options, and pick the shortest one!
Random sampling with Dubins steering
Case 2: We need to numerically solve the BVP

What if you were changing the steering rate?

\[
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta} \\
\ddot{\theta}
\end{bmatrix} = \begin{bmatrix}
V \cos \theta \\
V \sin \theta \\
\dot{\theta} \\
u
\end{bmatrix}
\]

\(|\dot{\theta}| \leq C_1 \quad |u| \leq C_2\)

\(q_1 = (x_1, y_1, \theta_1, \dot{\theta}_1)\)

\(q_2 = (x_2, y_2, \theta_2, \dot{\theta}_2)\)

No longer called Dubins path, called Clothoid path

Can’t solve analytically ... need numerical solutions 😐
Case 2: We need to numerically solve the BVP

Create a non-linear optimization problem

Represent the path as a curvature polynomial

Enforce end point constraints

https://nlopt.readthedocs.io/en/latest/
Figure 2. A 3D search space, consisting of position and heading ($x$, $y$, $\theta$). The Reeds–Shepp car can move forward and backward. It can drive straight or turn left or right at a fixed curvature. Left: The designed control set precisely hits vertices in a rectangular grid. It was derived from the car’s basic motions by carefully choosing their length. Center: The reachability tree to depth 2. Right: The reachability tree (search space) obtained by copying the control set at every vertex in a $\mathcal{C}$ space with four headings. Each dot represents four distinct vertices overlaid on each other, each representing different values of heading. Although this search space will not generate a turn of less than the chosen curvature, and although heading is continuous across vertices, the instantaneous transitions of curvature at the vertices do not respect steering rate limitations. Moreover, considering only four different heading values typically is impractical.

Figure 3. An example state lattice. A repeated and regular pattern of vertices and edges comprises the state lattice. The inset shows the control set, the motions leading to some nearby neighbors of a vertex. The overall motion plan (thick black curve) is simply a sequence of such edges. Here, a greater number of headings was used than in Figure 2. Reverse motions were omitted for clarity.

Enabling a planner to be well positioned with respect to properties 1 and 2 is related to the problem of sampling in the space of motions, and it remains an active area of our related work (Green & Kelly, 2007; Pivtoraiko, Knepper, & Kelly, 2007). One of the benefits of the state lattice approach is that it performs planning strictly in state space, which is easy to sample effectively. Regular lattice sampling features minimum dispersion and discrepancy, which allows the search to proceed effectively. Conversely, achieving effective sampling in control space is hard in general. However, the state lattice induces an convenient sampling, as motions that fit the lattice are found a posteriori. Thus, motion sampling inherits sampling effectiveness from the state lattice. An approach to satisfying properties 1 and 2 in state lattice design is presented in Pivtoraiko and Kelly (2005b). A simplified state lattice design, described in Section 4.2, can also be used as a departing point in evaluating the state lattice concept with a particular motion planner.

The general principle to address property 3 is to reduce the number of motions in the control set as far as possible. In the case of deterministic planning, the size of the control set defines the branching factor of the search space and, thus, significantly affects planning complexity.

3. MOTION PLANNING USING STATE LATTICES

This section is devoted to a discussion of constrained motion planning using state lattices. Here we utilize the search space, developed in the preceding sections, and discuss the algorithmic details of enabling planning and efficient replanning under differential constraints.

3.1. Search Algorithm

Because the state lattice is a directed graph, any systematic graph search algorithm is appropriate for finding a path in it. It is typically desired that a planner return optimal paths with respect to the desired cost criterion (e.g., time, energy, or path length) and that it be efficient. The A* (Hart, Nilsson, & Raphael, 1968) and D*Lite (Koenig & Likhachev, 2002) heuristic search algorithms were used in this work because they satisfy these requirements.

Let the term fidelity refer to the resolution of both the state samples and its connecting controls. If, hypothetically, the fidelity of the state lattice were...
Case 3: We can’t even solve the BVP

Typically rules out roadmap methods because we cannot exactly connect 2 states

Why?

1. Dynamics too complicated - nonlinear optimization doesn’t converge

2. Even if it did, too expensive to run online
Case 3: We can’t even solve the BVP

If we are working with implicit graphs, we don’t need to exactly connect two states.

Pretend this was the state you wanted all along!
Incremental Sampling-Based Planners

- Multiple-query methods such as PRM valuable in highly structured environments.
- Online planning does not require multiple queries
- Computing roadmap a priori computationally challenging/infeasible.
- Incremental sampling-based planning algorithms single-query counterpart to PRMs.
- Key idea:
  Avoid necessity to set number of samples \textit{a priori};
  return solution as soon as set of trajectories is rich enough.

Rapidly Exploring Random Trees (RRTs)

RRTs

• Build a tree by iteratively connecting next states via the execution of random controls
• Carefully sample controls to ensure good coverage
The RRT Algorithm

**GENERATE_RRT**(x\textit{init}, K, Δt)

1. \(\mathcal{T}.\text{init}(x_{init});\)
2. for \(k = 1\) to \(K\) do
3. \(x_{rand} \leftarrow \text{RANDOM\_STATE}();\)
4. \(x_{near} \leftarrow \text{NEAREST\_NEIGHBOR}(x_{rand}, \mathcal{T});\)
5. \(u \leftarrow \text{SELECT\_INPUT}(x_{rand}, x_{near});\)
6. \(x_{new} \leftarrow \text{NEW\_STATE}(x_{near}, u, \Delta t);\)
7. \(\mathcal{T}.\text{add\_vertex}(x_{new});\)
8. \(\mathcal{T}.\text{add\_edge}(x_{near}, x_{new}, u);\)
9. Return \(\mathcal{T}\)

**Probabilistically Complete**

**Exponential rate of decay for the probability of failure**
Extend Function

- No obstacles, holonomic:

- With obstacles, holonomic:

- Non-holonomic: approximately (sometimes as approximate as picking best of a few random control sequences) solve two-point boundary value problem

Slide credit: Pieter Abbeel
Figure 5.18: If the nearest point in $S$ lies in an edge, then the edge is split into two, and a new vertex is inserted into $G$.

Figure 5.19: In the early iterations, the RRT quickly reaches unexplored parts. However, the RRT is dense in the limit (with probability one), which means that it gets arbitrarily close to any point in the space.

RRT Expansion

45 iterations

2345 iterations
Notable variation: Bidirectional RRT

Key idea:
Grow 2 trees, 1 starting from start, 1 from goal, connect them, done.

RRT*

- Asymptotically optimal
- Probabilistically complete
- Computationally efficient

Algorithm 6: RRT*

1. $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset$
2. for $i = 1, \ldots, n$ do
3.     $x_{\text{rand}} \leftarrow \text{SampleFree}_i$
4.     $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}})$
5.     $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}})$
6.     if $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ then
7.         $X_{\text{near}} \leftarrow \text{Near}(G = (V, E), x_{\text{new}}, \min \{\gamma_{\text{RRT}}, (\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$
8.         $V \leftarrow V \cup \{x_{\text{new}}\}$
9.         $x_{\min} \leftarrow x_{\text{nearest}}; c_{\min} \leftarrow \text{Cost}(x_{\text{nearest}}) + c(\text{Line}(x_{\text{nearest}}, x_{\text{new}}))$
10.    foreach $x_{\text{near}} \in X_{\text{near}}$ do
11.        if $\text{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \land \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ then
12.            $x_{\min} \leftarrow x_{\text{near}}; c_{\min} \leftarrow \text{Cost}(x_{\text{near}}) + c(\text{Line}(x_{\text{near}}, x_{\text{new}}))$
13.    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$
14.    foreach $x_{\text{near}} \in X_{\text{near}}$ do
15.        if $\text{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \land \text{Cost}(x_{\text{new}}) + c(\text{Line}(x_{\text{new}}, x_{\text{near}})) < \text{Cost}(x_{\text{near}})$ then
16.            $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}})$
17.            $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
18.    return $G = (V, E)$


Slide credit: Pieter Abbeel
Smoothing

Paths extracted from sampling-based motion planners tend to be jerky.

Remedies:

1. Shortcutting

   Along the found path, pick two vertices $x_{t1}$, $x_{t2}$ and try to connect them directly (skipping over all intermediate vertices)

2. Nonlinear optimization for optimal control

   Allows to specify an objective function that includes smoothness in state, control, small control inputs, etc.

Slide credit: Pieter Abbeel