Autonomous Robotics
Winter 2024
Abhishek Gupta
TAs: Karthikeya Vemuri, Arnav Thareja
Marius Memmel, Yunchu Zhang

Slides borrowed from many sources – Sidd Srinivasa, Sanjiban Choudhury, Dieter Fox
Class Outline

- **State Estimation**
  - Robotic System Design
  - Filtering
    - Localization
    - SLAM

- **Control**
  - Feedback Control
  - PID Control
    - MPC
    - LQR

- **Planning**
  - Search
  - Heuristic Search
    - Motion Planning
    - Lazy Search

- **Learning**
  - Imitation Learning
  - Policy Gradient
    - Actor-Critic
    - Model-Based RL
Logistics

- HW 3 due Feb 14
- Paper commentaries due Wednesday 2/14
Lecture Outline

Why is the problem hard?

A recipe for solving motion planning problems

Graph Construction Techniques

Planning via Explicit Search
Geometric Path Planning Problem

Also known as

Piano Mover’s Problem (Reif 79)

Given:

1. A workspace $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An obstacle region $\mathcal{O} \subseteq \mathcal{W}$.
3. A robot defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of $m$ links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The configuration space $C$ ($C_{\text{obs}}$ and $C_{\text{free}}$ are then defined).
5. An initial configuration $q_I \in C_{\text{free}}$.
6. A goal configuration $q_G \in C_{\text{free}}$. The initial and goal configuration are often called a query $(q_I, q_G)$.

Compute a (continuous) path, $\tau : [0,1] \rightarrow C_{\text{free}}$, such that $\tau(0) = q_I$ and $\tau(1) = q_G$.

Also may want to minimize cost $C(\tau)$.
Differential constraints

In geometric path planning, we were only dealing with C-space

\[ q \in \mathcal{C} \]

We now introduce differential constraints

\[
\begin{bmatrix}
\dot{q} \\
\ddot{q}
\end{bmatrix} = f\left(\begin{bmatrix}
q \\
q
\end{bmatrix}, u\right)
\]

Let the state space \( x \) be the following augmented C-space

\[ x = (q, \dot{q}) \quad \dot{x} = f(x, u) \]
Differential constraints make things even harder.

These are examples of non-holonomic system.

Non-holonomic differential constraints are not completely integrable.

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”

These are examples of non-holonomic system

non-holonomic differential constraints are not completely integrable

i.e. the system is trapped in some sub-manifold of the config space
Motion planning under differential constraints

1. Given world, obstacles, C-space, robot geometry (same)

2. Introduce state space $X$. Compute free and obstacle state space.

3. Given an action space $U$

4. Given a state transition equations  $\dot{x} = f(x, u)$

5. Given initial and final state, cost function  $J(x(t), u(t)) = \int c(x(t), u(t))dt$

6. Compute action trajectory that satisfies boundary conditions, stays in free state space and minimizes cost.
Challenges in Motion Planning

Computing configuration-space obstacles

Planning in continuous high-dimensional space

Underactuated dynamics/constrained system does not allow direct teleportation

Goal: tractable approximations with provable guarantees!

EXAMPLE FROM HOWIE CHOSET
Lecture Outline

Why is the problem hard?

A recipe for solving motion planning problems

Graph Construction Techniques

Planning via Explicit Search
How might we tackle this problem?

Given:

1. A workspace $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An obstacle region $\mathcal{O} \subset \mathcal{W}$.
3. A robot defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of $m$ links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The configuration space $\mathcal{C}$ ($\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$ are then defined).
5. An initial configuration $q_I \in \mathcal{C}_{\text{free}}$.
6. A goal configuration $q_G \in \mathcal{C}_{\text{free}}$. The initial and goal configuration are often called a query $(q_I, q_G)$.

Compute a (continuous) path, $\tau : [0, 1] \rightarrow \mathcal{C}_{\text{free}}$, such that $\tau(0) = q_I$ and $\tau(1) = q_G$. 

Lets use ideas from search!
How might we tackle this problem?

Given:
1. A workspace $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An obstacle region $\mathcal{O} \subseteq \mathcal{W}$.
3. A robot defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of $m$ links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The configuration space $\mathcal{C}$ ($C_{\text{obs}}$ and $C_{\text{free}}$ are then defined).
5. An initial configuration $q_I \in C_{\text{free}}$.
6. A goal configuration $q_G \in C_{\text{free}}$. The initial and goal configuration are often called a query $(q_I, q_G)$.

Compute a (continuous) path, $\tau : [0, 1] \to C_{\text{free}}$, such that $\tau(0) = q_I$ and $\tau(1) = q_G$.  

Continuous space

Hard to characterize obstacles
Sampling-Based Motion Planning

Computing configuration-space obstacles is hard
- Use a collision checker instead!

Planning in continuous high-dimensional space is hard
- Construct a discrete graph approximation of the continuous space!

(EXAMPLE FROM HOWIE CHOSET)
Planning as Search

Convert into a search problem

Search the graph for a least-cost path from $s_{\text{start}}$ to $s_{\text{goal}}$

Can use efficient techniques for \textbf{discrete} graph search

Explicit graph search

Implicit sampling-based search
Recasting Planning as Search

Can use efficient techniques for **discrete** graph search

**Which ones?**

**How?**

Convert into a search problem

**planning map**

search the graph for a least-cost path from $s_{\text{start}}$ to $s_{\text{goal}}$
Recasting Planning as Search

Convert into a search problem

How? = Sampling

Can use efficient techniques for *discrete* graph search

Which ones? = *Best-first explicit search* or *Implicit sampling-based graph search*
Sampling-Based Motion Planning

CREATE GRAPH

SEARCH GRAPH

INTERLEAVE
Sampling-Based Motion Planning

NEW PLANNING ALGORITHM = GRAPH CONSTRUCTION × FANCY SEARCH ALGORITHM × ++ for efficiency
Lecture Outline

Why is the problem hard?

A recipe for solving motion planning problems

Graph Construction Techniques

Planning via Explicit Search
Creating a Graph

\[ G = (V, E) \]

1. Sample collision-free configurations as vertices (including start and goal)
2. Connect neighboring vertices with simple movements as edges
Creating a Graph

\[ G = (V, E) \]

1. Sample collision-free configurations as vertices (including start and goal)
2. Connect neighboring vertices with simple movements as edges
Strategy 1: Lattice Sampling / Discretization

Main idea: create a grid, and connect neighboring points (4-conn, 8-conn, ...)

Pros/Cons?
Strategy 2: Uniform Random Sampling

Main idea: sample uniformly between each dimension’s lower/upper bounds
Connect vertices within radius (r-disc) or k nearest neighbors

Pros/Cons?
Probabilistic Roadmap (PRM)

When should we collision-check edges?
What is the optimal radius? (PRM with optimal radius = PRM*)

KAVRAKI ET AL., 1996
Alternatives to Random Sampling
Strategy 3: Low-Dispersion Sampling

Main idea: Halton sequence uniformly densifies the space.
Detour: Van der Corput sequence

<table>
<thead>
<tr>
<th>$i$</th>
<th>Naive Sequence</th>
<th>Binary</th>
<th>Reverse Binary</th>
<th>Van der Corput</th>
<th>Points in $[0, 1]$/ ~</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1/16</td>
<td>.0001</td>
<td>.1000</td>
<td>1/2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1/8</td>
<td>.0010</td>
<td>.0100</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3/16</td>
<td>.0011</td>
<td>.1100</td>
<td>3/4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1/4</td>
<td>.0100</td>
<td>.0010</td>
<td>1/8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5/16</td>
<td>.0101</td>
<td>.1010</td>
<td>5/8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3/8</td>
<td>.0110</td>
<td>.0110</td>
<td>3/8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7/16</td>
<td>.0111</td>
<td>.1110</td>
<td>7/8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1/2</td>
<td>.1000</td>
<td>.0001</td>
<td>1/16</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9/16</td>
<td>.1001</td>
<td>.1001</td>
<td>9/16</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5/8</td>
<td>.1010</td>
<td>.0101</td>
<td>5/16</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>11/16</td>
<td>.1011</td>
<td>.1101</td>
<td>13/16</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3/4</td>
<td>.1100</td>
<td>.0011</td>
<td>3/16</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>13/16</td>
<td>.1101</td>
<td>.1011</td>
<td>11/16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7/8</td>
<td>.1110</td>
<td>.0111</td>
<td>7/16</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15/16</td>
<td>.1111</td>
<td>.1111</td>
<td>15/16</td>
<td></td>
</tr>
</tbody>
</table>
### Detour: Van der Corput sequence

The \( b \)-ary representation of the positive integer \( n \geq 1 \) is

\[
n = \sum_{k=0}^{L-1} d_k(n) b^k = d_0(n) b^0 + \cdots + d_{L-1}(n) b^{L-1},
\]

where \( b \) is the base in which the number \( n \) is represented, and \( 0 \leq d_k(n) < b \); that is, the \( k \)-th digit in the \( b \)-ary expansion of \( n \). The \( n \)-th number in the van der Corput sequence is

\[
g_b(n) = \sum_{k=0}^{L-1} d_k(n) b^{-k-1} = d_0(n) b^{-1} + \cdots + d_{L-1}(n) b^{-L}.
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>Naive Sequence</th>
<th>Binary</th>
<th>Reverse Binary</th>
<th>Van der Corput</th>
<th>Points in ([0,1]/\sim)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1/16</td>
<td>.0010</td>
<td>.1000</td>
<td>1/2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1/8</td>
<td>.0110</td>
<td>.0100</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3/16</td>
<td>.0011</td>
<td>.1100</td>
<td>3/4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1/4</td>
<td>.0100</td>
<td>.0010</td>
<td>1/8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5/16</td>
<td>.0101</td>
<td>.1010</td>
<td>5/8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3/8</td>
<td>.0110</td>
<td>.0110</td>
<td>3/8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7/16</td>
<td>.0111</td>
<td>.1110</td>
<td>7/8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1/2</td>
<td>.1000</td>
<td>.0001</td>
<td>1/16</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9/16</td>
<td>.1001</td>
<td>.1001</td>
<td>9/16</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5/8</td>
<td>.1010</td>
<td>.0101</td>
<td>5/16</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>11/16</td>
<td>.1011</td>
<td>.1101</td>
<td>13/16</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3/4</td>
<td>.1100</td>
<td>.0011</td>
<td>3/16</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>13/16</td>
<td>.1101</td>
<td>.1011</td>
<td>11/16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7/8</td>
<td>.1110</td>
<td>.0111</td>
<td>7/16</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15/16</td>
<td>.1111</td>
<td>.1111</td>
<td>15/16</td>
<td></td>
</tr>
</tbody>
</table>
Strategy 3: Low-Dispersion Sampling

Halton sequence – multi-dimensional van der corput sequence, co-prime bases

position(1234, 10) → [1, 2, 3, 4]
halton(1234, 10) → \frac{4}{10} + \frac{3}{100} + \frac{2}{1000} + \frac{1}{10000}

position(1234, 2) → [1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0]
halton(1234, 2) → \frac{1}{4} + \frac{1}{32} + \frac{1}{128} + \frac{1}{256} + \frac{1}{2048}

position(1234, 3) → [1, 2, 0, 0, 2, 0, 1]
halton(1234, 3) → \frac{1}{3} + \frac{2}{27} + \frac{2}{729} + \frac{1}{2187}

position(0x4d2, 16) → [4, 13, 2]
halton(0x4d2, 16) → \frac{2}{16} + \frac{13}{256} + \frac{4}{4096}

HTTPS://OBSERVABLEHQ.COM/@JRUS/HALTON
What Graphs Are Good?

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

A good graph must have good free-space coverage
  For every configuration in the free space, there’s a vertex in the graph that can be connected to it.

A good graph must have good free-space connectivity
  For every connected pair of points in the free space, there’s a path on the graph between them.
Creating a Graph

\[ G = (V, E) \]

1. Sample collision-free configurations as vertices (including start and goal)
2. Connect neighboring vertices with simple movements as edges
Creating a Graph

\[ G = (V, E) \]

Connect collision free edges
API for Graph Construction

Input

1. A collision checker
   \textit{coll}(q)

2. Steering method
   \textit{steer}(q_1, q_2)

Graph Construction

Output

Connected graph for search
Let’s take a look at the inputs

We need to give the planner a collision checker

\[
\text{coll}(q) = \begin{cases} 
0 & \text{in collision, i.e. } q \in C_{\text{obs}} \\
1 & \text{free, i.e. } q \in C_{\text{free}}
\end{cases}
\]

What work does this function have to do?

Collision checking is expensive!
Let’s take a look at the inputs

We need to give the planner a steer function

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

A steer function tries to join two configurations with a feasible path
Computes simple path, calls \(\text{coll}(q)\), and returns success if path is free

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

Example: Connect them with a straight line and check for feasibility
Can steer be smart about collision checking?

steer\((q_1, q_2)\) has to assure us line is collision free (upto a resolution)

Things we can try:

1. Step forward along the line and check each point

2. Step backwards along the line and check each point

......
Can steer be smart about collision checking?

Say we chunk the line into 16 parts

Any collision checking strategy corresponds to sequence

(Naive) \[ \alpha = 0, \frac{1}{16}, \frac{2}{16}, \frac{3}{16}, \ldots, \frac{15}{16} \]

(Bisection) \[ \alpha = 0, \frac{8}{16}, \frac{4}{16}, \frac{12}{16}, \ldots, \frac{15}{16} \]
### Ans: Van der Corput sequence

<table>
<thead>
<tr>
<th>$i$</th>
<th>Naive Sequence</th>
<th>Binary</th>
<th>Reverse Binary</th>
<th>Van der Corput</th>
<th>Points in $[0, 1]/\sim$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.0000</td>
<td>.0000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1/16</td>
<td>.0001</td>
<td>.1000</td>
<td>1/2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1/8</td>
<td>.0010</td>
<td>.0100</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3/16</td>
<td>.0011</td>
<td>.1100</td>
<td>3/4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1/4</td>
<td>.0100</td>
<td>.0010</td>
<td>1/8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5/16</td>
<td>.0101</td>
<td>.1010</td>
<td>5/8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3/8</td>
<td>.0110</td>
<td>.0110</td>
<td>3/8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7/16</td>
<td>.0111</td>
<td>.1110</td>
<td>7/8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1/2</td>
<td>.1000</td>
<td>.0001</td>
<td>1/16</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9/16</td>
<td>.1001</td>
<td>.1001</td>
<td>9/16</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5/8</td>
<td>.1010</td>
<td>.0101</td>
<td>5/16</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>11/16</td>
<td>.1011</td>
<td>.1101</td>
<td>13/16</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3/4</td>
<td>.1100</td>
<td>.0011</td>
<td>3/16</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>13/16</td>
<td>.1101</td>
<td>.1011</td>
<td>11/16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7/8</td>
<td>.1110</td>
<td>.0111</td>
<td>7/16</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15/16</td>
<td>.1111</td>
<td>.1111</td>
<td>15/16</td>
<td></td>
</tr>
</tbody>
</table>
How can we move from one configuration to another?
→ Hard in general!

Define a steering function that is tasked with connecting two configurations
→ Previously, steering function was trivial (straight line)
To construct a graph under differential constraints:
1. Sample collision free configuration states (check with collision checker)
2. Solve boundary-value problem to see if states can be connected
3. If connectable, add an edge, otherwise no edge
4. Benefit!
Solving the Boundary Value Problem

\[ q_1 = (x_1, y_1, \theta_1) \]

\[ q_2 = (x_2, y_2, \theta_2) \]

\[
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{bmatrix}
= 
\begin{bmatrix}
v \cos \theta \\
v \sin \theta \\
v \tan \delta/L
\end{bmatrix}
\]

\[ 0 \leq v \leq v_{\text{max}}, \quad |\delta| \leq \delta_{\text{max}} \]
Dubins showed that all solutions had to be one of six classes
\{LRL, RLR, LSL, LSR, RSL, RSR\}

Given two configurations to connect, evaluate all six options, return shortest one

Car has fixed forward velocity; Reeds-Shepp curves may include backward velocity
What Environments Are Hard?

Sampling-based methods struggle with narrow passages. Probability of sampling an edge in the passage is very small, so with a finite number of samples, the two halves of the roadmap may not be connected.

**Practical solutions:** sample near obstacle surface, bridge test to add samples between two obstacles, train ML algorithm to detect narrow passages.
Creating a Graph

\[ G = (V, E) \]

1. Sample collision-free configurations as vertices (including start and goal)
2. Connect neighboring vertices with simple movements as edges
Sampling-Based Motion Planning

CREATE GRAPH

SEARCH GRAPH

INTERLEAVE
Lecture Outline

Why is the problem hard?

A recipe for solving motion planning problems

Graph Construction Techniques

Planning via Explicit Search
Minimal Cost Path on a Graph

START, GOAL

COST (E.G. LENGTH)
Minimal Cost Path on a Graph

START, GOAL

COST (E.G. LENGTH)

GRAPH (VERTICES, EDGES)
Minimal Cost Path on a Graph

START, GOAL

COST (E.G. LENGTH)

GRAPH (VERTICES, EDGES)
Best-First Search Meta-Algorithm
Best-First Search Meta-Algorithm

Key insight: maintain a priority queue of promising nodes, ranked by $f(s)$

- Initialize queue with start node
- While goal isn’t reached
  - Pop the most promising node from the queue
  - If it’s not the goal, enqueue its neighbors
- When goal is reached, compute path by backtracking to the start
Best-First Search Meta-Algorithm

DIJKSTRA

A*
Best-First Search Implementation

Inputs: graph $G = (V, E)$; cost $c(s, s') = c(e)$; start and goal
Data structures maintained

- OPEN: priority queue of nodes that may be expanded (with priority $f$)
- CLOSED: set of nodes that have been expanded
- $g(s)$: estimated minimum cost from start to node $s$ ("cost-to-come")
Best-First Search Implementation

Initialize $g(\text{start}) = 0$ and all other $g$-values to infinity
Insert start into OPEN
While goal not in CLOSED
    Remove $s$ with smallest $f(s)$ from OPEN
    Add $s$ to CLOSED
    For every neighbor $s'$
        If $g(s) + c(s, s') < g(s')$, update $g(s')$ and add $s'$ to OPEN (with parent $s$)
Dijkstra’s Shortest Path Algorithm

Best-first search with $f(s) = g(s)$
Only expands nodes with lower cost-to-come than goal!