Motion Planning for Dynamic Environments

Part II: Motion Planning: Finding the Path

Steven M. LaValle
University of Illinois
A car driving on a gigantic sphere:

The C-space is:
A car driving on a gigantic sphere:

The C-space is: $SO(3) = \mathbb{R}P^3$

To see it, imagine car is painted on $S^2$ and rotate $S^2$ about its center.

It is not $S^2 \times S^1$: Cartesian product vs. fiber bundle (Hopf fibration)
Given robot $\mathcal{A}$ and obstacle $\mathcal{O}$ models, C-space $\mathcal{C}$, and $q_I, q_G \in \mathcal{C}_{free}$.

Automatically compute a path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ so that $\tau(0) = q_I$ and $\tau(1) = q_G$. 
Combinatorial planning
(exact planning)

Sampling-based planning
(probabilistic planning, randomized planning)

The methods differ in the philosophy they use to discretize the problem.

Also: Approximate cell decompositions
A planning algorithm may be:

- **Complete**: If a solution exists, it finds one; otherwise, it reports failure.

- **Semi-complete**: If a solution exists, it finds one; otherwise, it may run forever.

- **Resolution complete**: If a solution exists, it finds one; otherwise, it terminates and reports that no solution within a specified resolution exists.

- **Probabilistically complete**: If a solution exists, the probability that it will be found tends to one as the number of iterations tends to infinity.
Combinatorial Planning
Mostly developed in the 1980s

Influence from computational geometry and computational real algebraic geometry

All algorithms are complete

Usually produce a roadmap in $C_{free}$

Extremely efficient for low-dimensional problems

Some are difficult to implement (numerical issues)
Methods produce a topological graph $G$:

- Each vertex is a configuration $q \in C_{free}$.
- Each edge is a path $\tau : [0, 1] \rightarrow C_{free}$ for which $\tau(0)$ and $\tau(1)$ are vertices.

Sometimes, $C_{free}$ may be replaced by $cl(C_{free})$ (include the boundary of $C_{free}$).

This allows the robot to “scrape” the obstacles.
A roadmap is a topological graph $G$ with two properties:

1. **Accessibility**: From anywhere in $C_{free}$ it is trivial to compute a path that reaches at least one point along any edge in $G$.

2. **Connectivity-preserving**: If there exists a path through $C_{free}$ from $q_I$ to $q_G$, then there must also exist one that travels through $G$. 
Assume that $C_{obs}$ (and $C_{free}$) are piecewise linear. Could be a point robot among polygonal obstacles. Could be a polygonal, translating robot among polygonal obstacles. The methods tend to extend well to a disc robot.

Use clever data structures to encode vertices, edges, regions. Example: Doubly connected edge list.
We consider four methods:

- Trapezoidal decomposition
- Triangulation
- Maximum-clearance roadmap (retraction method)
- Shortest-path roadmap (reduced visibility graph)
Try to extend a ray above or below every vertex.

There are four cases:
Use the plane sweep principle to efficiently determine where the rays terminate.

Sort vertices by $x$ coordinate.

Handle extensions from left to right, while maintaining a vertically sorted list of edges.

Leads to $O(n \log n)$ running time. Easy to implement.
The resulting roadmap $G$: 

[Diagram of a roadmap $G$ showing a trapezoidal decomposition with obstacles and connected nodes]
Solving a query: Get from \( q_I \) to \( q_G \).
Compute triangulation: 
\(O(n^2)\) time naive, \(O(n)\) optimal, \(O(n \log n)\) a good tradeoff.

Build easy roadmap from the triangulation:
Imagine obtaining a skeleton by gradually thinning $C_{free}$.
Based on *deformation retract* from topology.
Also is a kind of generalized Voronoi diagram.

O’Dunlaing, Yap, 1983
Three cases contribute to the roadmap:

- **Edge-Edge**: \( O(n^4) \) time naive, \( O(n \log n) \) optimal.

- **Vertex-Vertex**

- **Vertex-Edge**

Picture from Latombe, 1991
Optimal planning is easy in polygonal environments.

The shortest-path roadmap contains all vertices and edges that optimal paths follow when obstructed.

Imagine pulling a string tight between $q_I$ and $q_G$. 
Every reflex vertex (interior angle $> \pi$) is a roadmap vertex.

Edges in the roadmap correspond to two cases:

1. Consecutive reflex vertices
2. Bitangent edges

A bitangent edge is needed when this is true:

$$\left( f_l(p_1, p_2, p_5) \oplus f_l(p_3, p_2, p_5) \right) \lor \left( f_l(p_4, p_5, p_2) \oplus f_l(p_6, p_5, p_2) \right),$$

in which $f_l$ is a left-turn predicate.
To solve a query, connect $q_I$ and $q_G$ to the roadmap:
Use Dijkstra’s algorithm to search for a shortest path.
If $C$ is 3 or more dimensions, most methods do not extend. Optimal path planning for 3D polyhedra is NP-hard. Maximal clearance roadmaps become disconnected in 3D.

Trapezoidal decomposition extends:
Higher Dimensions

- Specialized decompositions: ladder, rigid planar robot, discs
- Cylindrical algebraic decomposition (Schwartz, Sharir, 1983)
- Canny’s roadmap algorithm (1987)

Rearranging a bunch of rectangles is PSPACE-hard:
Decomposition for a Ladder

$O(n^5)$ time and space
Schwartz, Sharir, 1983
Cylindrical Algebraic Decomposition

Developed by Collins to decide Tarski sentences

Doubly exponential time and space.
Schwartz, Sharir, 1983
Singly exponential time.
Canny, 1987

Sampling-Based Planning
Sampling-Based Planning: Philosophy

- Use collision detector to separate planning from input geometry
- Systematically sample (random vs. deterministic) the free space
- Single-query: Incremental sampling and searching
- Multiple-query: Precompute a sampling-based roadmap
In topology, a set $U$ is called *dense in* $V$ if $cl(U) = V$.

Implication: Every open subset of $V$ contains at least one point in $U$.

Example: The rational numbers $\mathbb{Q}$ are dense in $\mathbb{R}$
(every open interval contains some fractions)

If $U$ is dense and countable, then a dense *sequence* can be formed:

$$\alpha : \mathbb{N} \rightarrow U$$

This imposes a linear ordering on $U$: $\alpha(1), \alpha(2), \ldots$

Example: A random sequence is dense with probability one.
Uniform random samples seem easy to produce. Statistical independence makes it easy to combine sample sets.

In reality, note that pseudo-random sequences are generated.
Be careful in curved spaces!

To generate a random point on $S^n$: Generate $n$ Guassian iid samples and normalize.

Uniform random rotation in $SO(3)$:
Choose three points $u_1, u_2, u_3 \in [0, 1]$ uniformly at random.

$$(a, b, c, d) = (\sqrt{1 - u_1 \sin 2\pi u_2}, \sqrt{1 - u_1 \cos 2\pi u_2}, \sqrt{u_1 \sin 2\pi u_3}, \sqrt{u_1 \cos 2\pi u_3}).$$
### Deterministic Alternative: 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>[0, 1]</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>
Let $P$ be a finite set of points in metric space $(X, \rho)$. The dispersion of $P$ is:

$$
\delta(P) = \sup_{x \in X} \left\{ \min_{p \in P} \{\rho(x, p)\} \right\}.
$$

In a bounded space, a dense sequence drives the dispersion to zero.
van der Corput is asymptotically optimal in terms of dispersion

Halton: Generalize van der Corput by using relatively prime bases (2, 3, 5, 7, 11, ...) for each coordinate.

More uniform than random (which needs $O((\log n)^{1/d})$ times as many samples needed to produce the same expected dispersion).

Sukharev theorem:
For any set $P$ of $k$ samples in $[0, 1]^d$:

$$\delta(P) \geq \frac{1}{2 \left\lfloor k^{\frac{1}{d}} \right\rfloor},$$

(1)

in which $\delta$ is the $L_\infty$ dispersion.

The best possible placement of $k$ points:

Think: “points per axes” for any sample set
Holding the dispersion fixed requires exponentially many points in dimension.
Johnson-Lindenstrauss Lemma

Maybe you didn’t need all of those dimensions anyway...

Pick any positive $\epsilon < 1$ and any set $P$ of $k$ points in $\mathbb{R}^n$, there exists a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ so that for all $x, y \in P$,

$$(1 - \epsilon)\|x - y\|^2 \leq \|f(x) - f(y)\|^2 \leq (1 + \epsilon)\|x - y\|^2,$$

and $m = 4 \ln n/(\epsilon^2/2 - \epsilon^3/3)$.

In other words, a low-distortion, low-dimensional embedding exists.

The basis of many dimensionality reduction methods, in machine learning, compressed sensing, computational geometry, ...
A Spectrum of Sample Sequences

1. Random sample sequence – can these really be generated?
2. Pseudo-random sequence
3. Low-dispersion sequence
4. Multiresolution grid
Irregularity Does Not Help

196 points in each square region:

- Pseudo-random points
- Pseudo-random points
- Halton points
- Hammersley points
- Lattice points
- Sukharev grid
Maintain a hierarchy of bounding regions

Two opposing criteria:

1. The region should fit the intended body points as tightly as possible.

2. The intersection test for two regions should be as efficient as possible.

Popular packages from UNC: PQP, I-Collide, ...
**Vertex-Vertex** Each point of the closest pair is a vertex of a polygon.

**Edge-Vertex** One point of the closest pair lies on an edge, and the other lies on a vertex.

**Edge-Edge** Each point of the closest pair lies on an edge. In this case, the edges must be parallel.
How many collision checks should be performed along an edge?

Using workspace distance information, may be able to guarantee collision-free segments.

Let \( a(q) \in A(q) \) denote a point on the robot. Find a constant \( c > 0 \) so that

\[
\|a(q) - a(q')\| < c\|q - q'\| \tag{3}
\]

over robot points and configuration pairs \( q, q' \).
Given a single query: \( q_I, q_G \in C_{\text{free}} \)

1. **Initialization:** Form \( G(V, E) \) with vertices \( q_I, q_G \) and no edges.

2. **Vertex Selection Method (VSM):** Choose a vertex \( q_{\text{cur}} \in V \) for expansion.

3. **Local Planning Method (LPM):** For some \( q_{\text{new}} \in C_{\text{free}} \), attempt to construct a path \( \tau_s : [0, 1] \rightarrow C_{\text{free}} \) such that \( \tau(0) = q_{\text{cur}} \) and \( \tau(1) = q_{\text{new}} \).

4. **Insert an Edge in the Graph:** Insert \( \tau_s \) into \( E \), as an edge from \( q_{\text{cur}} \) to \( q_{\text{new}} \). If \( q_{\text{new}} \) is not already in \( V \), then it is inserted.

5. **Check for a Solution:** Determine whether \( G \) encodes a solution path.

6. **Return to Step 2:** Iterate unless a solution has been found or the algorithm reports failure.
A convenient way to express the challenges of incremental sampling and searching
Barraquand, Latombe, 1989

Use BFS on an implicit, high resolution grid. Use random walks to escape local minima.

It was able to solve high dimensional problems, but required too much parameter tuning.
Connect $q_I$ and $q_G$ to the grid.
Apply classical grid search: BFS, DFS, Dijkstra, $A^*$.
Some Other Incremental Planners

- Ariadne’s clew algorithm (Mazer et al. 1992)
- Expansive space planner (Hsu et al., 1997)
- Rapidly exploring Random Trees (LaValle, Kuffner, 1998)
- SBL planning (Sanchez, Latombe, 2001)
- Adaptive random walk planner (Carpin, Pillonetto, 2005)
Suppose $X = [-25, 25]^2$ and $q_I = (0, 0)$. 
Pick a vertex a random, extend one unit in a random direction repeat, ... 
What happens?
Suppose $X = [-25, 25]^2$ and $q_I = (0, 0)$. Pick a vertex at random, extend one unit in a random direction repeat, ... What happens?
By changing the vertex selection method, we obtain this:

Rather than pick a *vertex* at random, pick a *configuration* at random.
SIMPLE_RRT($q_0$)

1. $G$.init($q_0$);
2. for $i = 1$ to $k$ do
   3. $q_r \leftarrow$ RandomConf($i$);
   3. $G$.add_vertex($q_r$);
   4. $q_n \leftarrow$ NEAREST($S(G), q_r$);
   5. $G$.add_edge($q_n, q_r$);

To bias toward the goal, $q_G$ can be substituted for RandomConf($i$) in some (e.g., every 100) iterations. RandomConf($i$) can be replaced by any dense sequence $\alpha(i)$ to obtain Rapidly exploring Dense Trees (RDTs).
Extend the nearest vertex (using the metric!) to the random point:

If there is an obstacle, then stop short:
If the nearest RRT point lies in an edge, it is better to extend from there:

\[ q_0, \alpha(i), q_n \]

45 iterations 2345 iterations
An approximate solution: Insert intermediate vertices.
For large RRTs (thousands of nodes), nearest-neighbor requests dominate.

In some settings, Kd-trees can dramatically improve performance.
To solve a $q_I, q_G$ query with RRTs.
Grow two trees: 1) $T_i$ from $q_I$ and 2) $T_g$ from $q_G$.

Repeat the following four steps:

1. Extend $T_i$ using $\alpha(i)$, making $q_{new}$.
2. Extend $T_g$ using $q_{new}$. If connected, then solution found.
3. Extend $T_g$ using $\alpha(i + 1)$, making $q_{new}$.
4. Extend $T_i$ using $q_{new}$. If connected, then solution found.
If there are multiple queries in the same $C_{\text{free}}$, then precomputing a roadmap may pay off.

BUILD_ROADMAP
1 $G$.init(); $i \leftarrow 0$;
2 while $i < N$
3     if $\alpha(i) \in C_{\text{free}}$ then
4         $G$.add_vertex($\alpha(i)$); $i \leftarrow i + 1$;
5 for each $q \in \text{NEIGHBORHOOD}(\alpha(i), G)$
6     if (not $G$.same_component($\alpha(i), q$)) and $\text{CONNECT}(\alpha(i), q)$ then
7         $G$.add_edge($\alpha(i), q$);
Connection rules:

- **Nearest K:** The $K$ closest points to $\alpha(i)$ are considered. This requires setting the parameter $K$ (a typical value is 15).

- **Component K:** Try to obtain up to $K$ nearest samples from each connected component of $G$.

- **Radius:** Take all points within a ball of radius $r$ centered at $\alpha(i)$.

- **Visibility:** Try connecting $\alpha$ to all vertices in $G$.

Sampling strategies: Gaussian, medial axis, bridge-test, ...
See Karaman, Frazzoli, IJRR 2011 for PRM connection theory.
Simeon, Laumond, Nissoux, 2000

Define two different kinds of vertices in $G$:

**Guards:** To become a *guard*, a vertex, $q$ must not be able to see other guards.

**Connectors:** To become a *connector*, a vertex, $q$, must see at least two guards.
Differential Constraints
Due to robot kinematics and dynamics, most systems are *locally* constrained, in addition to *global* obstacles.

Let $\dot{q}$ represent the C-space velocity.

In ordinary planning, any “direction” is allowed and the magnitude does not matter.

Thus, we could say

$$\dot{q} = u$$ (4)

and $u \in \mathbb{R}^n$ may be any velocity vector so that $\|u\| \leq 1$. 
More generally, a control system (or state transition equation) constrains the velocity:

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

and \( u \) belongs to some set \( U \) (usually bounded).

A function \( \tilde{u} : T \to U \) is applied over a time interval \( T = [0, t_f] \) and the configuration \( q(t) \) at time \( t \) is given by the state at time \( t \) is given by

\[ q(t) = q(0) + \int_0^t f(q(t'), \tilde{u}(t')) \, dt'. \]

in which \( q(0) \) is the initial configuration.
This car drives forward only:

\[ C = \mathbb{R}^2 \times S^1. \]

Let \( u = (u_s, u_\phi) \) and \( U = [0, 1] \times [-\phi_{max}, \phi_{max}] \).

Control system of the form \( \dot{q} = f(q, u) \):

\[
\begin{align*}
\dot{x} &= \cos \theta \\
\dot{y} &= \sin \theta \\
\dot{\theta} &= \frac{u_s}{L} \tan u_\phi.
\end{align*}
\]
Stepping forward in the Dubins car

Two stages

Four stages
Handling higher order derivatives on $C$ allows dynamical system models. This includes accelerations, momentum, drift.

Let $X$ be a state space (or phase space). Typically, $X = C \times \mathbb{R}^n$, in which $n$ is the dimension of $C$. Each $x \in X$ represents a $2n$ dimensional vector $x = (q, \dot{q})$.

A control system then becomes

$$\dot{x} = f(x, u)$$

Note that $\dot{x}$ includes $\ddot{q}$ components (hence, acceleration constraints).
The obstacle region in $X$ is usually:

$$X_{\text{obs}} = \{ x \in X \mid \kappa(x) \in C_{\text{obs}} \},$$

This has cylindrical structure:
Discretize time and action space:

A trajectory in $\mathcal{U}$

A trajectory in $\mathcal{U}_d$

Using countability to try all sequences:
Local Planning

Combinatorial Planning
Sampling-Based Planning
Differential Constraints

System Simulator

\[ x(0) \rightarrow \tilde{u}_t \rightarrow t \rightarrow \text{System Simulator} \rightarrow \tilde{x}_t \]

Easy

Two-Point Boundary-Value Solver

\[ x_I \rightarrow x_G \rightarrow \tilde{u}_t \]

Usually Hard
Boundary Value Problems

Combinatorial Planning
Sampling-Based Planning
Differential Constraints

No BVP

One BVP

One BVP

One BVP
Four stages for Dubins

Limiting one vertex per cell

Barraquand, Latombe, 1993
For an RRT, just replace the “straight line” connection with a local planner.

```
SIMPLE_RRT_WITH_DIFFERENTIAL_CONSTRAINTS(x_0)
1 \( G.\text{init}(x_0); \)
2 \( \textbf{for } i = 1 \textbf{ to } k \textbf{ do} \)
3 \( x_n \leftarrow \text{NEAREST}(S(G), \alpha(i)); \)
4 \( (\tilde{u}^P, x_r) \leftarrow \text{LOCAL_PLANNER}(x_n, \alpha(i)); \)
5 \( G.\text{add\_vertex}(x_r); \)
6 \( G.\text{add\_edge}(\tilde{u}^P); \)
```

Problems: Need good metrics and primitives
Combinatorial Planning

Sampling-Based Planning

Differential Constraints

- Combinatorial vs. sampling based.
- For some problems, combinatorial is far superior.
- For most “industrial problems” sampling-based works well.
- Weaker notions of completeness are tolerated.
- Dimensionality always an issue (Sukharev).

For an infinite sample sequence $\alpha : \mathbb{N} \to X$, let $\alpha_k$ denote the first $k$ samples.

Find a metric space $X \subseteq \mathbb{R}^n$ and $\alpha$ so that:

1. The dispersion of $\alpha_k$ is $\infty$ for all $k$.

2. The dispersion of $\alpha$ is 0.

Hint: Do not make it too complicated.