16-843 – Manipulation Algorithms
Motion Planning

Motion Planning I
Configuration Space Obstacles

**Star Algorithm**

Convex robot, translation

$C_{obs}$: convex polygon

Determine the order of these edges!

Key observation: every edge of $C_{obs}$ is a translated edge from either $A$ or $O$
Configuration Space Obstacles

**Star Algorithm**
Convex robot, translation

$C_{obs}$: convex polygon

**Solid representation of $C_{obs}$**

**Idea:** Create convex obstacles by intersecting half-planes

Two types of contacts:

- Vertex of $A$ and Edge of $O$
- Edge of $A$ and Vertex of $O$
**Configuration Space Obstacles**

**Star Algorithm**
Convex robot, translation

$C_{obs}$: convex polygon

**Solid representation of $C_{obs}$**

**Idea**: Create convex obstacles by intersecting half-planes

Contact occurs when $n$ and $v$ are perpendicular

\[ n \cdot v = 0 \]

\[ H = \{(x_t, y_t) \in C \mid n \cdot v(x_t, y_t) \leq 0\} \]

Analysis of contacts:

Function of kinematics

\[ f(q) \leq 0 \]
Configuration Space Obstacle

What happens when the robot translates *and* rotates in the plane?

Need to consider orientation, i.e., need to consider all possible contact interactions between all pairs of object and robot features!

**Problem: Exponential run times!**
Real Time Configuration Space Transforms for Obstacle Avoidance
Newman and Branicky, 1991

Objective:
Fast \( C_{obs} \) construction for manipulator with revolute joints

Motivation:
\( C_{obs} \) construction is exponential in time. Not suitable for real time planning.
Real Time Configuration Space Transforms for Obstacle Avoidance

Newman and Branicky, 1991

**Approach:**

- Define transformation primitives that allow to transformation workspace obstacles to c-Space obstacles
- Algorithm: lookup + analytical transformations to quickly generate the c-space for complex environments from simple primitives
Real Time Configuration Space
Transforms for Obstacle Avoidance
Newman and Branicky, 1991

General Transformation Properties:

• Set Union Property: \( C_{obs}(B_1 \cup B_2) \equiv C_{obs}(B_1) \cup C_{obs}(B_2) \)

Relevance: \( C_{obs}(E) = \bigcup_{i=1}^{n} B_i \)  \( E = \bigcup_{i=1}^{n} B_i \)
Can decompose complex objects into sets for which we have efficient transformations

• Set Containment Property:
  • A physical object that is fully enclosed by a larger one has a transform that is fully contained in the transform of the larger one
  • Only need transformations of boundaries

If \( B_1 \supseteq B_2 \) :
\( C_{obs}(B_1) = C_{obs}(B_1) \cup C_{obs}(B_2) \)
\( C_{obs}(B_1) \supseteq C_{obs}(B_2) \)
Real Time Configuration Space Transforms for Obstacle Avoidance

Newman and Branicky, 1991

General Transformation Primitives:

• Point Primitives
• Line Primitives
• Circle Primitives
Real Time Configuration Space Transforms for Obstacle Avoidance
Newman and Branicky, 1991

General Transformation Primitives:

- Point Primitives

\[ \theta_1 = \text{elbow} \cdot \cos \left( \frac{l_1^2 + d^2 + s^2}{2l_1d} \right) = f_1(d, \text{elbow}, s) \quad (9) \]

\[ \theta_2 = \text{elbow} \cdot \left( \cos \left( \frac{l_1^2 + s^2 - d^2}{2l_1s} \right) - \pi \right) = f_2(d, \text{elbow}, s) \quad (10) \]
Advantages:
Can quickly generate complex configuration space obstacle from precomputed and stored primitives.

Disadvantages:
A lot of work! Only approximation. Robot arm dependent.
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

Objective
Simple and efficient motion planning algorithm for \textit{general} manipulator moving in a cluttered environment

What is it about:
Configuration space representation that is suitable for motion planning (easy to implement).
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

Motivation

Existing motion planning algorithms are either:

• Not simple enough to implement
• Not general enough

Approach:

Discretize joint angles and look at ranges of joint angle values
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

Key Idea:
Slice Projections

\[ q \in C_{obs} = \mathbb{R}^n \]
Configuration for manipulator with n degrees of freedom

Approximate \( C_{obs} \) by an union of \( n-1 \) dimensional slices

\[ \pi_j(q) = \left( q_1, \ldots, q_{j-1}, q_{j+1}, \ldots, q_n \right) \]

\[ \Pi_{[a_j,b_j]}(C_{obs}) = \left\{ \pi_j(q) \mid q \in C_{obs}, q_j \in [a_j,b_j] \right\} \]

Slice projection of \( C_{obs} \) for values of \( q_j \) in the range of \( a_j-b_j \)
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

**Key Idea:**
Slice Projections

Example:

\[ a = 0 \]

\[ b = \frac{\pi}{4} \]

\[ q \in [a, b] \]
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

**Key Idea:**

Slice Projections

Example
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

Key Idea:
Slice Projections

Example:
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

Key Idea:
Slice Projections
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

Algorithm to create C-Space approximation:

For $i=1$ to $i=n$:

Compute $C$-Space($i$):

- Ignore links $> i$
- Find range of legal values range($q_i$) by rotating joint $i$ given the current joint ranges $C$-Space($i-1$)
- If $i<n$: sample range($q_i$) at specified resolution
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

Algorithm to create C-Space approximation:

For \( i = 1 \) to \( i = n \):

Compute C-Space(\( i \)):

- Ignore links > \( i \)
- Find range of legal values \( \text{range}(q_i) \) by rotating joint \( i \) given the current joint ranges C-Space(\( i-1 \))
- If \( i < n \): sample \( \text{range}(q_i) \) at specified resolution
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

How to compute \( \text{range}(q_i) \): 

Assume: \( k < i \) are fixed and safe

Two types of collision:
- Edge link with vertex object
- Vertex link with edge object
How to compute range($q_i$):

1. Determine if vertex and infinite support line of edge intersect

\[ n \cdot v(q_i) + d = 0 \]
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

How to compute $\text{range}(q_i)$:

1. Determine if vertex and infinite support line of edge intersect
2. In edge constraint: Does it indeed intersect with edge
3. Orientation constraint: Ensure polygon edges do not intersect object at contact point
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

How to compute range($q_i$):

1. Determine if vertex and infinite support line of edge intersect
2. In edge constraint:
   compute the coordinates
3. Orientation constraint:
   Ensure polygon edges do not intersect object at contact point
4. Reachability constraint:
   Can it reach point without collision?
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

How to compute $\text{range}(q_i)$:

1. Determine if vertex and infinite support line of edge intersect
2. In edge constraint: compute the coordinates
3. Orientation constraint: Ensure polygon edges do not intersect object at contact point
4. Reachability constraint: Can it reach point without collision?
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

The effect of ranges of joint angles:

Idea: Grow the shape of the links to accommodate the range of the joint angles!
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

Free-Space Representation:

**Regions**: set of adjacent slices with overlapping ranges

Build **region graph** to search for path between points in different regions
A Simple Motion Planning Algorithm for General Robot Manipulation

Thomas Lozano-Perez, AAAI 1986

Advantages:

Simpler and more efficient than complete C-space construction.

Disadvantages:

• Approximation: may miss legal paths
• Rapid grow in execution time and memory with growing DOF
Motion Planning
Motion Planning

Let $W = \mathbb{R}^m$ being the work space, $O \subset W$ the set of obstacles, $A(q)$ the robot in configuration $q \in C$

$$C_{obs} = \{q \in C \mid A(q) \cap O \neq \emptyset\}$$

$$C_{free} = C \setminus C_{obs}$$

**Motion planning**: find a continuous path

$$\tau : [0,1] \rightarrow C_{free}$$

With $\tau(0) = q_I$, $\tau(1) = q_G$

Planning with robot being point in C-Space
Motion Planning

Overview of all motion planning we will handle:

- Combinatorial Planning
- Sampling based Planning
- Potential Functions
- Jacobian Transpose
- Decision Theory
Motion Planning

Continuous terrain needs to be discretized for path planning!

Two general approaches:

Combinatorial Planning
- Explicit characterization of $C_{\text{free}}$
- Capture connectivity of $C_{\text{free}}$ into graph and do graph search to find solutions
- Often complete

Sampling based planning
- Sample in C-space
- Use collision detection to probe for collisions with obstacles
- Probabilistic complete
Completeness

Complete:
A motion planner always returns a solution if there exist one and returns a failure otherwise in bounded time.

Probabilistic Complete:
The probability tends to zero as the number of sampled points increases. No time bound.
Road maps

Road map $R$, is a union of 1-D curves s.t. for all $q_I \in C_{free}$ and $q_G \in C_{free}$ there exist a path between $q_I$ and $q_G$ if:

1) Accessibility: There is a path to $q_I \in C_{free}$ to a $q' \in R$
2) Departability: There is a path from a $q_G \in C_{free}$ to a $q'' \in R$
3) Connectivity: There exist a path in $R$ between $q'$ and $q''$

Building $R$:

- Nodes: $q \in C_{free}$ (or its boundary)
- Edge between nodes: if there is a free path between the two nodes

Path Planning:

1) Connect $q_I$ and $q_G$ to road map to $q'$ and $q''$
2) Find path from $q'$ and $q''$ in $R$
Combinatorial Planning

Main combinatorial planning techniques:

- Visibility graphs
- Voronoi diagrams
- Exact cell decomposition
- Approximate cell decomposition

Result: Graph, often road map

Require: Explicit $C_{obs}$ representation
Visibility graph

Idea:
Construct a path as polygonal line connecting $q_i$ and $q_G$ through vertices of $C_{obs}$

Why?
Any collision-free path can be transformed into a piece-wise linear path that bends only at the obstacle vertices.
Visibility graph

A graph defined by:

- Nodes: $q_I, q_G$, obstacle vertex
- Edges: either obstacle edge or edge between nodes that does not intersect with any obstacle

Runtime: $O(n^2 \log n)$

Shortest Path roadmaps
Voronoi Diagram

**Idea:** Divide the space into regions closest to a particular ‘site’
Generalized Voronoi Diagram

**Idea:** Set of points that have maximal clearance from all nearest objects
**Generalized Voronoi Diagram**

**Idea:** Set of points that have maximal clearance from all nearest objects

**Voronoi diagram**

\[ V(C_{\text{free}}) = \left\{ q \in C_{\text{free}} \mid |\text{near}(q)| > 1 \right\} \]

where

\[ \text{near}(q) = \left\{ p \in B_{\text{free}} \mid d(p,q) = \min(p,q) \right\} \]

The set of configurations with more than one nearest base point p
Generalized Voronoi Diagram

**Idea:** Set of points that have maximal clearance from all nearest objects

For polygonal $C_{obs}$ consists of $n$ lines and parabolic segments:

Naïve: $O(n^4)$
Best: $O(n \log n)$

Pros:
- maximize clearance good when uncertainty involved (what is always the case)

Cons:
- Attraction to open space
- Suboptimal paths
- Real time for low-dimensional C-space

Figure: Howie Choset
Cell Decomposition

Idea:
Decompose $C_{\text{free}}$ into regions (cells) and create connectivity graph that represents adjacencies

Two types of approach:

- **Exact Cell Decomposition:** Union of cells corresponds to $C_{\text{free}}$
- **Approximate Cell Decomposition:** Decomposes $C_{\text{free}}$ into cells with pre-defined shape; union of cells typically only approximates $C_{\text{free}}$ space
Exact Cell Decomposition

**Example**: Trapezoid Decomposition

Each cell is either triangle or trapezoid and is formed by:

1) Draw vertical line segments by shooting rays upward and downward from each polygonal vertex
Exact Cell Decomposition

Example: Trapezoid Decomposition
Each cell is either triangle or trapezoid and is formed by:

1) Draw vertical line segments by shooting rays upward and downward from each polygonal vertex
2) Each cell corresponds to one node in connectivity graph
Exact Cell Decomposition

**Example:** Trapezoid Decomposition

Each cell is either triangle or trapezoid and is formed by:

1) Draw vertical line segments by shooting rays upward and downward from each polygonal vertex
2) Each cell corresponds to one node in connectivity graph
3) Nodes are connected if the corresponding cells share a common boundary

Nodes: Cells
Edges: between all adjacent cells

Run time best: $O(n \log n)$
N number of cells
Approximate Cell Decomposition

Exact cell decomposition can be inefficient for complex objects and problems

Idea: Use cells with the same simple predefined shape, e.g., rectangle, square

• Hierarchical space decomposition: Quadtree or Octree decomposition

• The union of those non-overlapping cells is contained in $C_{free}$
Approximate Cell Decomposition

Example: Quadtree decomposition

Pros:
- Iteration of repetitive and simple computation
- Simple to implement
- Can be made complete
Combinatorial Planning

Summary

Pros

Completeness

In finite time:
• Report failure if no solution exist
• Report solution, otherwise

Cons

Become intractable with increasing C-space dimensionality and/or object complexity

Combinatorial explosion in representing robot, obstacles and $C_{obs}$, especially with manipulators

How can we do better?
Sampling Based Planning

Sampling based planning algorithms:

• Probabilistic Road Maps (PRM)
• Rapidly Exploring Random Trees
• Derivates of above

More efficient in most practical problems but weaker guarantees:

• Probabilistic Complete: The probability tends to 0 that no solution exist if not found yet

Main planning approaches found in manipulation!
Discrete Motion Planning

Given a graph, how do we search?

Search Algorithms:
- Breadth-First Search
- Depth-First Search
- Dijkstra’s algorithm
- Greedy best-first Search
- A*
- D*
## Summary

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Combinatorial Planning</th>
<th>Sampling Based Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Visibility Graphs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Generalized Voronoi Diagrams</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Exact Cell Decomposition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Approximate Cell Decomposition</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concept</th>
<th>Combinatorial Planning</th>
<th>Sampling Based Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Explicit representation of $C_{obs}$</td>
<td>• Sample in C-Space</td>
</tr>
<tr>
<td></td>
<td>• Captures connectivity of $C_{free}$ in graph and finds path with graph search</td>
<td>• Use collision detection to probe for collisions with obstacles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pros and Cons</th>
<th>Combinatorial Planning</th>
<th>Sampling Based Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pros</td>
<td>complete</td>
<td>Cons</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Becomes intractable with increasing C-Space and object/robot complexity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Completeness</th>
<th>Combinatorial Planning</th>
<th>Sampling Based Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td></td>
<td>Probabilistic Complete</td>
</tr>
</tbody>
</table>
Summary

Concepts:

• Completeness
• Probabilistic Complete
• Roadmap