Road Map Methods

Including material from Howie Choset
The Basic Idea

- Capture the connectivity of $Q_{\text{free}}$ by a graph or network of paths.
RoadMap Definition

- A roadmap, RM, is a set of trajectories (i.e. $f(t, q_A, q_B)$) such that for all $q_{\text{start}} \in Q_{\text{free}}$ and $q_{\text{goal}} \in Q_{\text{free}}$ can be connected by a path:

- The three ingredients of a roadmap
  - **Accessibility:** There is a path from $q_{\text{start}} \in Q_{\text{free}}$ to some $q' \in RM$
  - **Departability:** There is a path from some $q'' \in RM$ to $q_{\text{goal}} \in Q_{\text{free}}$
  - **Connectivity:** there exists a path in RM between $q'$ and $q''$
RoadMap Path Planning

1. Build the roadmap
   a) nodes are points in $Q_{\text{free}}$ or its boundary
   b) two nodes are connected by an edge if there is a free path between them (i.e. $f(\ t, q_A, q_B )$)

2. Connect $q_{\text{start}}$ and $q_{\text{goal}}$ points to the road map at point $q'$ and $q''$, respectively

3. Find a path on the roadmap between $q'$ and $q''$. The result is a path in $Q_{\text{free}}$ from start to goal
Overview

• Deterministic methods
  ✓ Some need to represent $Q_{\text{free}}$, and some don’t.
  ✓ are complete
  ✓ are complexity-limited to simple (e.g. low-dimensional) problems
    • example: Canny’s Silhouette method (5.5)
      – applies to general problems
      – is singly exponential in dimension of the problem
Visibility Graph methods

- Defined for polygonal obstacles
- Nodes correspond to vertices of obstacles
- Nodes are connected if
  - they are already connected by an edge on an obstacle
  - the line segment joining them is in free space
- Not only is there a path on this roadmap, but it is the *shortest* path
- If we include the start and goal nodes, they are automatically connected
- Algorithms for constructing them can be efficient
  - $O(n^3)$ brute force
Visibility Graph in Action

1. Draw lines of sight from the start and goal to all “visible” vertices and corners of the world.

\[ e_{ij} \neq \emptyset \iff sv_i + (1 - s)v_j \in \text{cl}(Q_{\text{free}}) \quad \forall s \in (0, 1) \]
Visibility Graph in Action

2. Second, draw lines of sight from every vertex of every obstacle like before. Remember lines along edges are also lines of sight.

\[ e_{ij} \neq \emptyset \iff sv_i + (1 - s)v_j \in \text{cl}(Q_{\text{free}}) \quad \forall s \in (0, 1) \]
Visibility Graph in Action
The Visibility Graph in Action (Part 4)
Visibility Graph in Action

• Repeat until you’re done.
Visibility Graphs

• Find a path in the graph.
  – Bread first, depth first, Dijkstra’s, etc.
The Sweepline Algorithm

1: For each vertex $v_i$, calculate $\alpha_i$, the angle from the horizontal axis to the $O(n)$ line segment $\overline{vv_i}$.
2: Create the vertex list $\mathcal{E}$, containing the $\alpha_i$'s sorted in increasing order. $O(n \log n)$.
3: Create the active list $\mathcal{S}$, containing the sorted list of edges that intersect the horizontal half-line emanating from $v$.
4: for all $\alpha_i$ do $n$ times (once for each vertex)
5: if $v_i$ is visible to $v$ then
6: Add the edge $(v, v_i)$ to the visibility graph.
7: end if
8: if $v_i$ is the beginning of an edge, $E$, not in $\mathcal{S}$ then
9: Insert the $E$ into $\mathcal{S}$.
10: end if
11: if $v_i$ is the end of an edge in $\mathcal{S}$ then
12: Delete the edge from $\mathcal{S}$.
13: end if
14: end for

Analysis: For a vertex, $n \log n$ to create initial list, $\log n$ for each $\alpha_i$
Overall: $n \log (n)$ (or $n^2 \log (n)$ for all $n$ vertices)
Algorithm:
Initially:
1. calculate the angle $\alpha_i$ of segment $\overline{vv_i}$ and sort vertices by this creating list $\mathcal{E}$
2. create a list $S$ of edges that intersect the horizontal from $v$ sorted by intersection distance
For each $\alpha_i$
   - if $v_i$ is visible to $v$ then add $\overline{vv_i}$ to graph
   - if $v_i$ is the “beginning” of an edge $E$, insert $E$ in $S$
   - if $v_i$ is the “end” of and edge $E$, remove $E$ from $S$

<table>
<thead>
<tr>
<th>Vertex</th>
<th>New $S$</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>${E_4, E_2, E_8, E_6}$</td>
<td>Sort edges intersecting horizontal half-line</td>
</tr>
<tr>
<td>$\alpha_3$</td>
<td>${E_4, E_3, E_8, E_6}$</td>
<td>Delete $E_2$ from $S$. Add $E_3$ to $S$.</td>
</tr>
<tr>
<td>$\alpha_7$</td>
<td>${E_4, E_3, E_8, E_7}$</td>
<td>Delete $E_6$ from $S$. Add $E_7$ to $S$.</td>
</tr>
<tr>
<td>$\alpha_4$</td>
<td>${E_8, E_7}$</td>
<td>Delete $E_3$ from $S$. Delete $E_4$ from $S$. ADD $(v, v_4)$ to visibility graph</td>
</tr>
<tr>
<td>$\alpha_8$</td>
<td>${}$</td>
<td>Delete $E_7$ from $S$. Delete $E_8$ from $S$. ADD $(v, v_8)$ to visibility graph</td>
</tr>
<tr>
<td>$\alpha_1$</td>
<td>${E_1, E_4}$</td>
<td>Add $E_4$ to $S$. Add $E_1$ to $S$. ADD $(v, v_1)$ to visibility graph</td>
</tr>
<tr>
<td>$\alpha_5$</td>
<td>${E_4, E_1, E_8, E_5}$</td>
<td>Add $E_8$ to $S$. Add $E_5$ to $S$.</td>
</tr>
<tr>
<td>$\alpha_2$</td>
<td>${E_4, E_2, E_8, E_5}$</td>
<td>Delete $E_1$ from $S$. Add $E_2$ to $S$.</td>
</tr>
<tr>
<td>$\alpha_6$</td>
<td>${E_4, E_2, E_8, E_6}$</td>
<td>Delete $E_5$ from $S$. Add $E_6$ to $S$.</td>
</tr>
<tr>
<td>Termination</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reduced Visibility Graphs

- The current graph as too many lines
  - lines to concave vertices
  - lines that “head into” the object
- A reduced visibility graph consists of
  - Edges that are “separating”
  - Edges that are “supporting” (i.e. do not head into the object at either endpoint)

![Diagram of Visibility Graph and Reduced Visibility Graph](image)
Voronoi Diagrams

- Given a set of points (called sites), divide the space in Voronoi regions
  - The region of a site is defined by all the points closer the site
  - Edges are equally distant points from sites

There’s an emergency, where is the nearest station?
Generalized Voronoi Diagram

• Extends the Voronoi diagram to non-point sites
  – A region is defined by all the points that are closer to an obstacle

• The boundaries (composed of points between **two** equidistant obstacles) are used to plan the path

• Topological skeleton of a simple polygon (no holes)
GVD Construction

Polygonal Space Construction

1. The set of points equidistant to two points is a line
2. The set of points equidistant to two segments is a line
3. The set of points equidistant to a line and a point is a parabola
GVD Construction

Grid Construction

- Recall the brushfire algorithm produces a set of points that are equidistance from all obstacles.

Almost by definition, this set of points (the Voronoi diagram) is a roadmap since each “watershed” can be connected to the backbone, then the backbone connects to all other “watersheds.”
Brushfire example

Principles of Robot Motion
Brushfire example

Principles of Robot Motion
Brushfire example

Principles of Robot Motion
Is This A Roadmap?

• How do we know the roadmap is connected?
  
  – **Accessibility**: follow gradient uphill to diagram – guaranteed to be free as you are going away from obstacles
  
  – **Departability**: go downhill from closest point to goal
  
  – **Connectivity**: The book talks about deformation retracts as a way of showing this; we will not cover this
Robotic Motion Planning: Cell Decompositions
(with some discussion on coverage)
Types of Decompositions

- Trapezoidal Decomposition
- Morse Cell Decomposition
  - Boustrophedon decomposition
  - Morse decomposition definition
  - Sensor-based coverage
  - Examples of Morse decomposition
- Visibility-based Decomposition
Adjacency Graph

- Node correspond to a cell
- Edge connects nodes of adjacent cells
- Two cells are adjacent if they share a common boundary
Trapezoidal Decomposition
Path Planning

- Path Planning in three steps:
  - A trapezoid is convex. Any two points on its boundary can be connected by a straight line
  - Planner determines cells that contain the start and goal
  - Planner searches for a path within adjacency graph
  - Find the exact path in the configuration space.
Trapezoidal Decomposition
Trapezoidal Decomposition
Trapezoidal Decomposition
Trapezoidal Decomposition

Thanks to H. Choset, N. Amato
Trapezoidal Decomposition

Thanks to H. Choset, N. Amato
Trapezoidal Decomposition

- The robot crosses the boundary between adjacent cells at the midpoints.
Implementation

• Input is vertices and edges
• Sort n vertices $O(n \log(n))$
• Determine vertical extensions
  – For each vertex, intersect vertical line with each edge – $O(n)$ time
  – Total $O(n^2)$ time
Sweep Line Approach

- Sweep a line through the space stopping at vertices \( v_i \)
- Maintain a list \( L \) of the current edges the slice intersects
- Determining the intersection of slice with \( L \) requires \( O(n) \) time but with an efficient data structure like a balanced tree, perhaps \( O(\log n) \)
- Determine between which two edges the vertex \( v \) lies (\( e_{\text{LOWER}} \) and \( e_{\text{UPPER}} \))
- Maintaining \( L \) takes \( O(n \log n) – \log n \) for insertions, \( n \) for vertices
- For each vertex \( v \), let \( e_{\text{lower}} \) and \( e_{\text{upper}} \) be the two edges that contain \( v \).
Coverage

Planner determines an exhaustive walk through the adjacency graph

Planner computes explicit robot motions within each cell

Problems

1. Polygonal representation
2. Quantization
3. Position uncertainty
4. Full information
Trapezoidal vs Boustrophedon Decomposition

Coverage Path in a Cell.

Coverage Path in a Cell.
Complete Coverage

Exhaustive walk 1–2–4–2–3–5–6–5–7–5–8–1