Combinatorial Motion Planning Algorithms

EN.530.663: Robot Motion Planning

Jin Seob Kim, Ph.D

Senior Lecturer, LCSR, ME dept., JHU

Spring 2024

Introduction

Combinatorial MP algorithms

- Consider C_{free} as it is.
- All algorithms are "complete".
- Typically "low dimensionality" and complex models $\rightarrow$ can provide an elegant and practical solution.
  - e.g, $\mathcal{W} = \mathbb{R}^2$ and the robot is only translating with fixed orientation.
- Can provide theoretical upper bounds.
- Warning: practicality issue (hard to implement)
- Application: coverage planning problem

Roadmaps

- Let $\mathcal{G}$ be a topological graph that maps into C_{free}. Let $S \subset C_{free}$ be the set of all points reached by $\mathcal{G}$.
- The graph $\mathcal{G}$ is called a **roadmap** if it satisfies
  1. **Accessibility** From any $q \in C_{free}$, it is simple and efficient to compute a path $\tau : [0, 1] \rightarrow C_{free}$ such that $\tau(0) = q$ and $\tau(1) = s$, $\forall s \in S$.
  2. **Connectivity-preserving** If $\exists \tau : [0, 1] \rightarrow C_{free}$ s.t. $\tau(0) = q_I$ and $\tau(1) = q_G$, then $\exists \tau' : [0, 1] \rightarrow S$ s.t. $\tau(0) = s_1$ and $\tau(1) = s_2$, $(s_1, s_2 \in S)$.
- Due to these properties, a roadmap provides a discrete representation of the continuous motion planning problem.
- A query (q_I, q_G) is solved by connecting each query point to the roadmap and then performing a graph search on $\mathcal{G}$.

Exact Cell Decomposition

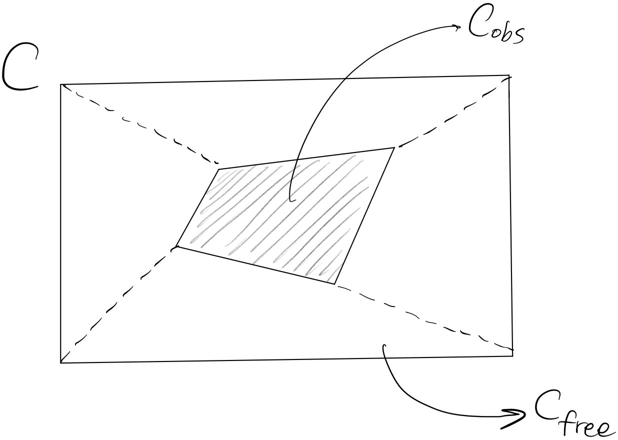
Exact Cell Decomposition

- General Procedures
  - Decompose the robot's C_{free} into a collection of non-overlapping regions (called cells) whose union is exactly C_{free}. $\rightarrow$ need a finite data structure.
  - cells $\rightarrow$ k-cell : k-dimensional cell
  - Construct and search the connectivity graph representing the adjacency relation among cells.
  - Extract a path (by using Dijkstra or A^*).

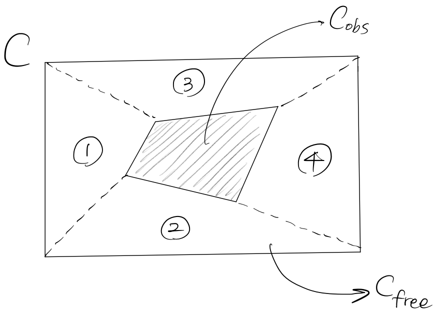
Exact Cell Decomposition

- Three Properties of Cell Decomposition
  1. Computing a path from one point to another inside a cell must be trivially easy.
     - ex: every cell is convex $\rightarrow$ any pair of points in a cell is connected by a line segment.
  2. Adjacency information for the cells can be easily extracted to build a roadmap.
     - ex: edge-sharing $\rightarrow$ then maybe choose a center point in that edge for the roadmap.
  3. For a given q_I and q_G, it should be efficient to determine which cells contain them.
     - ex: again, convex cells are useful: in Matlab, `inpolygon` .

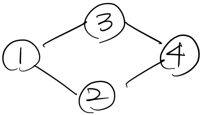
Exact Cell Decomposition: Example



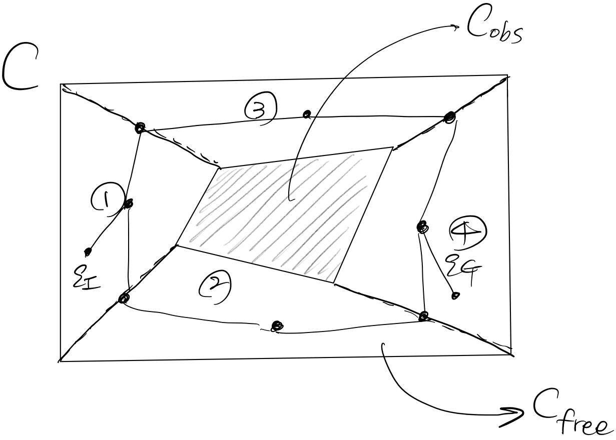
Exact Cell Decomposition: Example



Connectivity graph



Exact Cell Decomposition: Example



$q_I \in$ cell 1

$q_G \in$ cell 4

Exact Cell Decomposition

- Other methods
  - Voronoi diagram
  - triangulation
  - Maximum-clearance roadmaps (Sec.6.2.3)
  - and more...

Vertical/Cylindrical Cell Decomposition

Vertical Cell Decomposition

- Cells are usually trapezoids or triangles. $\rightarrow$ also called “trapezoidal decomposition”.
- One of the easiest methods.
- Consider only x-coordinates of all vertices to form a “vertical line”.
- See Fig. 6.3 – 6.5.

Vertical Cell Decomposition

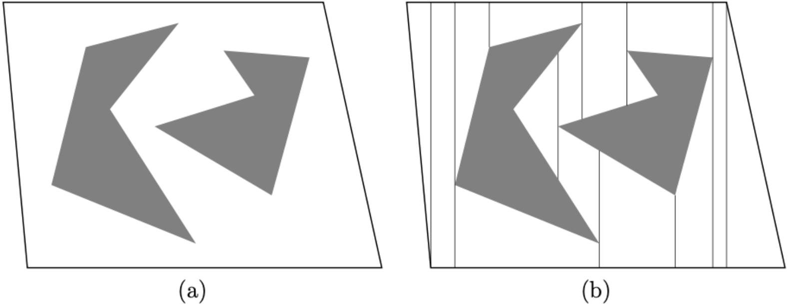


Figure 6.3: The vertical cell decomposition method uses the cells to construct a roadmap, which is searched to yield a solution to a query.

Vertical Cell Decomposition

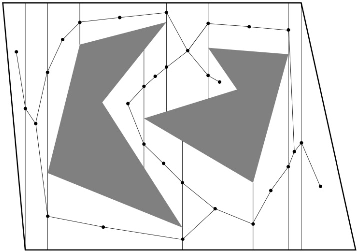


Figure 6.4: The roadmap derived from the vertical cell decomposition.

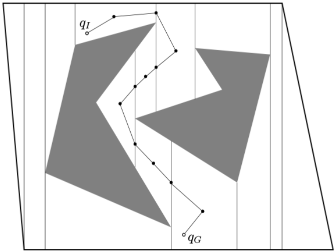
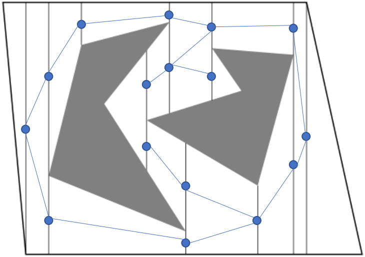
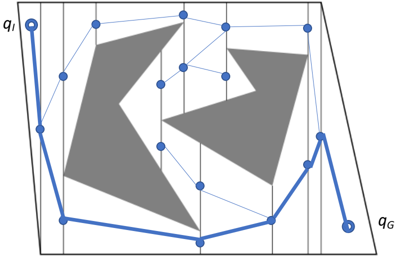


Figure 6.5: An example solution path.

Vertical Cell Decomposition





Vertical Cell Decomposition

- Vertical Cell Decomposition Detail

  1. Move the sweep-line from left to right.
  2. For each vertex it touches, extend from the vertex along the line.
  3. Stop when the line hit the features of the environment (e.g., end of the environment box).
  4. Compute the centers and/or boundary centers and connect them.

Vertical Cell Decomposition

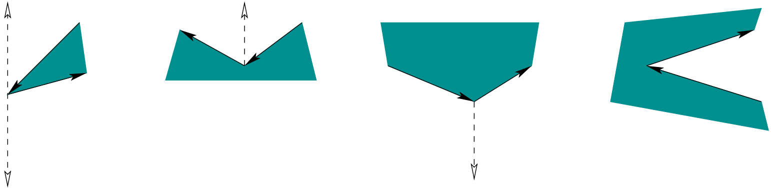
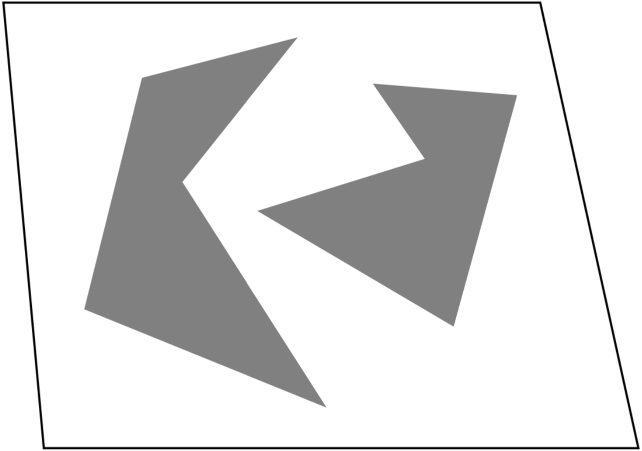


Figure 6.2: There are four general cases: 1) extending upward and downward, 2) upward only, 3) downward only, and 4) no possible extension.

Vertical Cell Decomposition: Algorithm

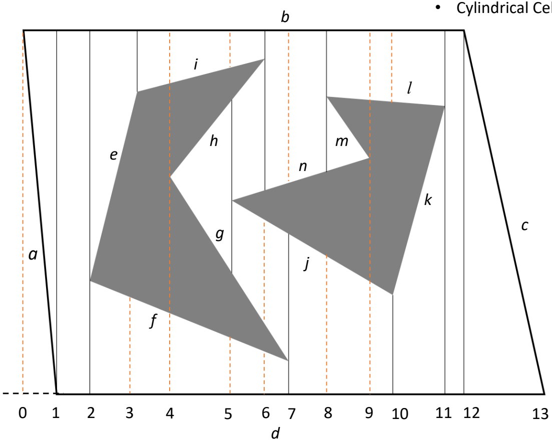
- Simple Algorithm
  1. Sort the x-coordinates of input points (vertices of C_{obs}) $\rightarrow O(n \log n)$.
  2. For each point, check whether the vertical scan line intersects each line segment of the environment/obstacle, then pick the closest intersection points. In other words, given x_i, consider all line segments.
  3. Overall, computation time: $O(n^2)$.
- Efficient Algorithm
  - Use “plane-sweep” (or “line-sweep”) $\rightarrow O(n \log n)$.
  - Main idea: “keep the List”.

Vertical Cell Decomposition: Algorithm Execution

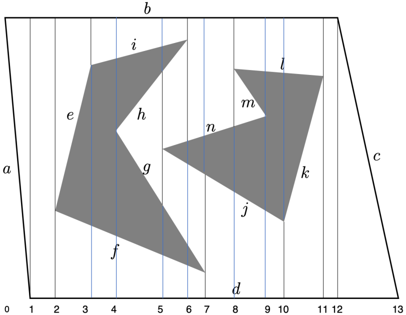


Vertical Cell Decomposition: Algorithm Execution

- Vertical Cell Decomposition
- Cylindrical Cell Decomposition

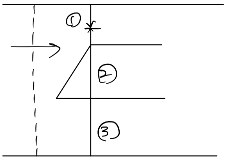


Vertical Cell Decomposition: Algorithm Execution



Event	Sorted Edges in L	Event	Sorted Edges in L
0	$\{a, b\}$	7	$\{d, j, n, b\}$
1	$\{d, b\}$	8	$\{d, j, n, m, l, b\}$
2	$\{d, f, e, b\}$	9	$\{d, j, l, b\}$
3	$\{d, f, i, b\}$	10	$\{d, k, l, b\}$
4	$\{d, f, g, h, i, b\}$	11	$\{d, b\}$
5	$\{d, f, g, j, n, h, i, b\}$	12	$\{d, c\}$
6	$\{d, f, g, j, n, b\}$	13	$\{\}$

Vertical Cell Decomposition: Algorithm Execution



→ split into 3 segments

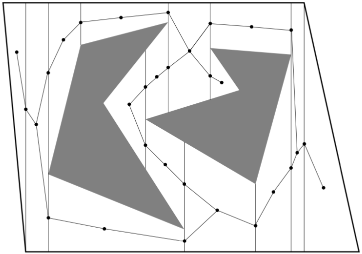
For ①: keep it (∗ : center point)

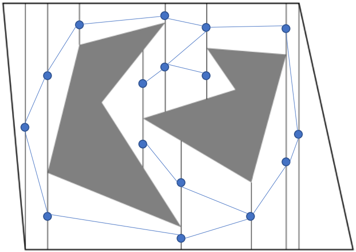
For ②: ignore (NOT consider)

For ③: keep it ⇒ cylindrical cell decompostion
ignore ⇒ vertical cell decomposition

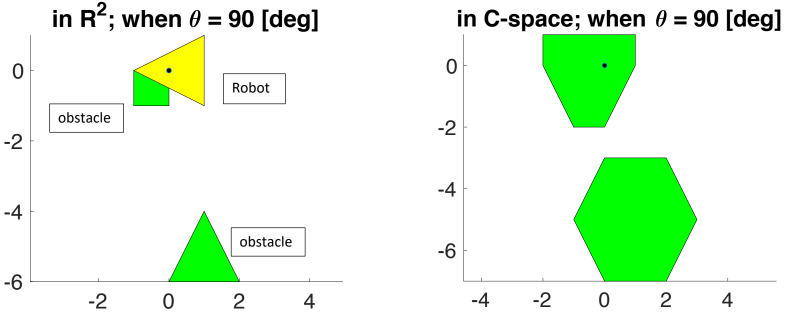
Vertical Cell Decomposition: Algorithm Execution

- Line segment split and collect center points
- Then generate a graph.

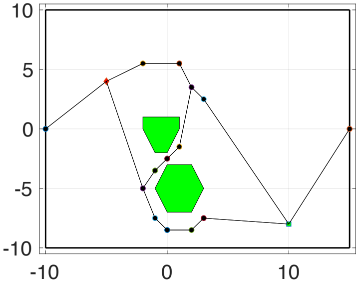


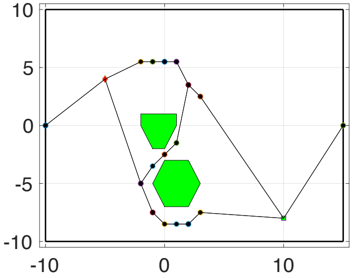


Vertical Cell Decomposition: Example



Vertical Cell Decomposition: Example





Vertical Cell Decomposition: 3D example

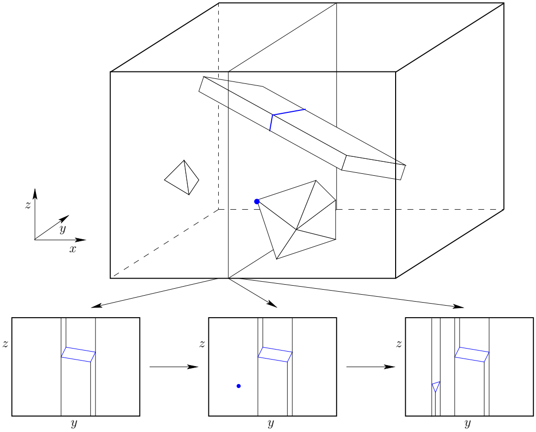
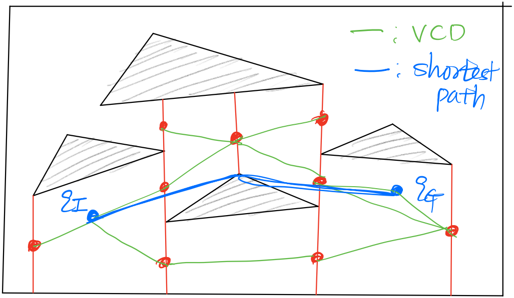


Figure 6.20: In higher dimensions, the sweeping idea can be applied recursively.

Visibility Graph

Visibility Graph



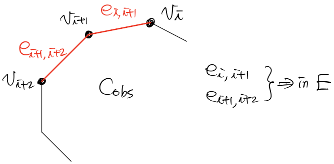
Visibility Graph

- Visibility graph $G = (V, E)$
  - $V = \{v_i\}$ node set: includes q_I, q_G, and all the vertices of the configuration space obstacles.
  - $E = \{e_{ij}\}$ where e_{ij} denotes a straight line segment that connects v_i and v_j.
  - $e_{ij} \neq \emptyset \Leftrightarrow tv_i + (1 - t)v_j \in \mathcal{C}l(C_{free}),\ \forall t \in [0, 1]$

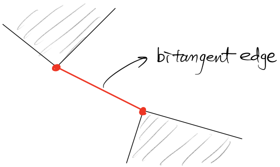
Visibility Graph

Specifically:

1. Consecutive reflex vertices
2. Bitangent edges: touches two reflex vertices that are mutually visible.



(a) Consecutive edge

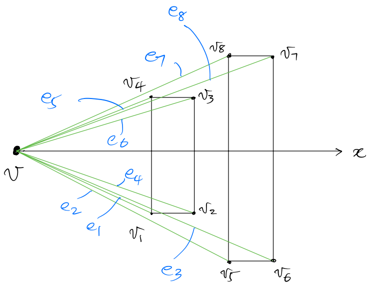


(b) Bitangent edge

Visibility Graph: Radial Sweep

- Rotational Plane Sweep Algorithm:
  1. Sort the obstacle vertices according to the CCW angle that $\overline{v_i v_j}$ makes with the positive x-axis.
  2. Check if $v_i, v_j \in \text{same } C_{obs}$.
  3. If so, keep only "boundary" edges to insert into G.
  4. If NOT, insert the edges that are "visible" to G.
- Improvements: read p.218 (Sec.6.2.4)

Visibility Graph: Radial Sweep: Visible



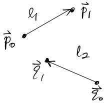
- Sorted edges: $\{e_6, e_8, e_5, e_7, e_2, e_1, e_3, e_4\}$
- Edges to keep (visible edges from v): $\{e_5, e_7, e_2, e_1\}$

Visibility Graph

- Needs an algorithm to check the intersection of line segments.
- Regarding the “piano mover’s problem”, it may not provide a best solution.
- For higher dimensional problems (e.g., 3D), it does not provide a shortest-path solution.
  - $\rightarrow$ The shortest path in 3D does not need to pass through vertices.

Visibility Graph: Line Segments Intersection

□ Computing two line segments intersection



$l_1 : \vec{p}_0 + s(\vec{p}_1 - \vec{p}_0)$, $s \in [0, 1]$

$l_2 : \vec{q}_0 + t(\vec{q}_1 - \vec{q}_0)$, $t \in [0, 1]$

Let $\begin{cases} \vec{u} = \vec{p}_0 - \vec{p}_1 \\ \vec{v} = \vec{q}_1 - \vec{q}_0 \end{cases}$

- $\left| \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \, \|\vec{v}\|} \right| = 1 \rightarrow l_1, l_2$ are parallel.

- Let $\left.\begin{array}{l} A = [\vec{u}_1 \;\; \vec{u}_2] \in \mathbb{R}^{2\times2} \\ \vec{b} = \vec{p}_0 - \vec{q}_0 \end{array}\right\} \Rightarrow$ solve $A\vec{x} = \vec{b}$, $\vec{x} = \begin{pmatrix} s \\ t \end{pmatrix}$

$\because \; \underline{l_1 = l_2}$

$\vec{p}_0 + s(\vec{p}_1 - \vec{p}_0) = \vec{q}_0 + t(\vec{q}_1 - \vec{q}_0)$

$(\vec{p}_0 - \vec{p}_1)s + (\vec{q}_1 - \vec{q}_0)t = \vec{p}_0 - \vec{q}_0$

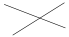
$$\underbrace{[\vec{p}_0 - \vec{p}_1 \quad \vec{q}_1 - \vec{q}_0]}_{A} \underbrace{\begin{pmatrix} s \\ t \end{pmatrix}}_{\vec{x}} = \underbrace{\vec{p}_0 - \vec{q}_0}_{\vec{b}}$$

First, check <u>parallel</u>

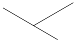
$A = [\vec{a}_1 \;\; \vec{a}_2] \Rightarrow \left| \frac{\vec{a}_1 \cdot \vec{a}_2}{\|\vec{a}_1\| \, \|\vec{a}_2\|} \right| = 1$

If not, compute for s and t

if • $0<s<1$ and $0<t<1 \Rightarrow$ intersect

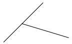


else if • $(s=0$ or $s=1)$ and $(0<t<1)$



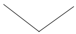
$\Rightarrow$ intersect

else if • $(0<s<1)$ and $(t=0$ or $t=1)$



$\Rightarrow$ intersect

else if • $(s=0$ or $s=1)$ and $(t=0$ or $t=1)$



$\Rightarrow$ <u>NOT</u> intersect.

else NOT intersect

Visibility Graph: Example