Computational Geometry
csci3250

Laura Toma

Bowdoin College
Motion Planning

Input:

- a robot R and a set of obstacles $S = \{O_1, O_2, \ldots\}$
- start position $p_{\text{start}}$
- end position $p_{\text{end}}$

Find a path from start to end (that optimizes some objective function).

Parameters:

- geometry of obstacles (polygons, disks, convex, non-convex, etc)
- geometry of robot (point, polygon, disc)
- robot movement (dof)
- objective function to minimize (euclidian distance, nb turns, etc)
- 2d, 3d
- static vs dynamic environment
- exact vs approximate algorithm
- known vs unknown map
Motion Planning

Ideally we want a planner to be complete and optimal.

- A planner is complete:
  - it always finds a path when a path exists

- A planner is optimal:
  - it finds an optimal path
A Preview of Approaches

**Exact (geometric/combinatorial)**

- Roadmaps
- from trapezoidal decomposition
- Visibility graphs

**Approximate**

- sampling-based
- approximate space decomposition
Today

**Exact (geometric/combinatorial)**

- Point robot in 2D
  - Paths: Roadmaps via trapezoid decomposition
  - Shortest paths: Visibility graph

- Polygon robot in 2D
  - Translation only
  - Handling rotations
Point robot in 2D

- General idea
  - Compute free space
  - Compute a representation of free space
  - Build a graph of free space
  - Search graph to find path ←---------- Reduce motion planning to graph search
Point robot in 2D

• General idea
  
  • Compute free space
  
  • Compute a representation of free space: trapezoidal decomposition, size $O(n)$, $O(n \log n)$ time
  
  • Build a graph of free space: size $O(n)$, $O(n)$ time
  
  • Search graph to find path: BFS in $O(n)$ time
Point robot in 2D

Result:

Let R be a point robot moving among a set of polygonal obstacles in 2D with n edges in total. We can pre-process S in $O(n \log n)$ expected time such that, between any start and goal position, a collision-free path for R can be computed in $O(n)$ time, if it exists.

\[ n = \text{complexity of obstacles} \]
\[ \text{(total number of edges)} \]
Point robot in 2D

What if we wanted the shortest path $p_{\text{start}}$ to $p_{\text{goal}}$?
Any shortest path among a set $S$ of disjoint polygonal obstacles:

1. is a polygonal path (that is, not curved)
2. its vertices are the vertices of $S$. 
Shortest paths for point robot in 2D

- Idea: Build the visibility graph
  - all possible ways to travel between the vertices of the obstacles
- Claim: any shortest path must be a path in the VG

![Diagram of a visibility graph and obstacles]
Shortest paths for point robot in 2D
Shortest paths for point robot in 2D

Algorithm

- Compute visibility graph
  - $V = \{\text{set of vertices of obstacles} + p_{\text{start}} + p_{\text{end}}\}$
- SSSP (Dijkstra) in VG

- How big is VG and how long does it take to compute it?

$n = \text{complexity of obstacles (total number of edges)}$
Shortest paths for point robot in 2D

Algorithm

- Compute visibility graph
  - \( V = \{\text{set of vertices of obstacles } + p_{\text{start}} + p_{\text{end}}\} \)
- SSSP (Dijkstra) in VG

\[ n = \text{complexity of obstacles} \]
\[ (\text{total number of edges}) \]
Shortest paths for point robot in 2D

Algorithm

- Compute visibility graph
  - \( V = \{ \text{set of vertices of obstacles} + p_{\text{start}} + p_{\text{end}} \} \)
- SSSP (Dijkstra) in VG

- Complexity of VG

\( n = \text{complexity of obstacles (total number of edges)} \)
Shortest paths for point robot in 2D

Algorithm

- Compute visibility graph
  - \( V = \{\text{set of vertices of obstacles} + p_{\text{start}} + p_{\text{end}}\} \)
- SSSP (Dijkstra) in VG

- Complexity of VG
  - \( V_{VG} = O(n), \ E_{VG}=O(n^2) \) \( \langle \cdots \rangle \) can have quadratic size

\( n \) = complexity of obstacles
(total number of edges)
Shortest paths for point robot in 2D

Algorithm

- Compute visibility graph
  - \( V = \{\text{set of vertices of obstacles} + p_{\text{start}} + p_{\text{end}}\} \)
- SSSP (Dijkstra) in VG

- Complexity of VG
  - \( V_{\text{VG}} = O(n), E_{\text{VG}} = O(n^2) \) <-------- can have quadratic size
- Computing VG
Shortest paths for point robot in 2D

Algorithm

- Compute visibility graph
  - $V = \{\text{set of vertices of obstacles } + p_{\text{start}} + p_{\text{end}}\}$
- SSSP (Dijkstra) in VG

- Complexity of VG
  - $V_{\text{VG}} = O(n)$, $E_{\text{VG}} = O(n^2)$ <-------- can have quadratic size

- Computing VG
  - naive: for each edge, check if intersects any obstacle. $O(n^3)$

$n$ = complexity of obstacles (total number of edges)
Shortest paths for point robot in 2D

Algorithm

- Compute visibility graph
  - $V = \{\text{set of vertices of obstacles} + p_{\text{start}} + p_{\text{end}}\}$
- SSSP (Dijkstra) in VG

- Complexity of VG
  - $V_{VG} = O(n), E_{VG} = O(n^2)$ \textbullet can have quadratic size

- Computing VG
  - naive: for each edge, check if intersects any obstacle. $O(n^3)$
  - improved: $O(n \lg n)$ per vertex, $O(n^2 \lg n)$ total
Shortest paths for point robot in 2D

Algorithm

- Compute visibility graph
  - \( V = \{ \text{set of vertices of obstacles} + p_{\text{start}} + p_{\text{end}} \} \)
- SSSP (Dijkstra) in VG

- Complexity of VG
  - \( V_{\text{VG}} = O(n), \ E_{\text{VG}}=O(n^2) \) \(<------ \text{can have quadratic size}\)

- Computing VG
  - naive: for each edge, check if intersects any obstacle. \( O(n^3) \)
  - improved: \( O(n \lg n) \) per vertex, \( O(n^2 \lg n) \) total

- Dijkstra on VG: \( O(E_{\text{VG}} \lg n) = O(n^2 \lg n) \)
Shortest paths for point robot in 2D

Algorithm

- Compute visibility graph \( \mathcal{O}(n^2 \lg n) \)
- SSSP (Dijkstra) in VG \( \mathcal{O}(E_{VG} \lg n) \)

Theorem:

A shortest path of two points among a set of polygonal obstacles with \( n \) edges in total can be computed in \( \mathcal{O}(E_{VG} \lg n) = \mathcal{O}(n^2 \lg n) \) time.
Improved computation of VG

- Idea
  - for every vertex $v$: compute all vertices visible from $v$
Improved computation of VG

• Idea
  • for every vertex v: compute all vertices visible from v
Improved computation of VG

• Idea
  • for every vertex v: compute all vertices visible from v

active structure stores all edges intersected by sweep line, ordered by distance from v
Improved computation of VG

- Idea
  - for every vertex $v$: compute all vertices visible from $v$

$w$ visible if $vw$ does not intersect the interior of any obstacle
Improved computation of VG

- Idea

  - for every vertex $v$: compute all vertices visible from $v$
  - Overall: $n \times n \log n = O(n^2 \log n)$
Motion planning via visibility graph

Comments

• Is it optimal? Is it complete?

• VG needs to be computed only once, so we can think of it as pre-processing

• VG may be large ==> doomed to quadratic complexity

History/overview:

• Quadratic barrier broken by Joe Mitchell: SP of a point robot moving in 2D can be computed in $O(n^{5/3} + \varepsilon)$

• Hershberger and Suri [1993]: SP of a point robot moving in 2D can be computed in $O(n \lg n)$ (“continuous Dijkstra” approach)

• Special cases can be solved faster:
  • e.g. SP inside a simple polygon w/o holes: $O(n)$ time
Shortest paths in 3D

- VG does not generalize to 3D
- SP in 3D much harder: no easy way to discretize the problem
  - inflection points of SP are not restricted to vertices of S, can be inside edges
  - 3D shortest paths among polyhedral obstacles is NP-complete
- Complete planning in 3D is hopeless
Today

Exact (geometric/combinatorial)

- Point robot in 2D
  - Paths: Roadmaps via trapezoid decomposition
  - Shortest paths: Visibility graph

- Polygon robot in 2D
  - Translation only
  - Handling rotations
Convex polygon moving in 2D

- How can the robot move?
  - Translation only
  - Translation + rotation
Work/physical space

- Space where robot moves around
- A placement of robot is specified by the degrees of freedom (dof) of the robot

- Examples:
  
  2D: \( R(x,y) \)
  
  \( R(x,y,\theta) \)

  2D: \( R(x,y,\theta_1,\theta_2) \)

  3D: \( R(x,y,z) \)
Configuration space (C-space)

- A point in C-space corresponds to placement of the robot in physical space.
- The parametric space of the robot = space of all possible placements of the robot.
  - Examples:
    - 2D, translation only $\iff R(x,y)$
    - 2D, transl. + rot. $\iff R(x,y, \theta)$
- A path for R corresponds to a path in C-space.
<table>
<thead>
<tr>
<th>robot</th>
<th>physical space</th>
<th>C-space</th>
</tr>
</thead>
<tbody>
<tr>
<td>polygon, (translation only)</td>
<td>2D</td>
<td>2D</td>
</tr>
<tr>
<td>polygon, (translation + rotations)</td>
<td>2D</td>
<td>3D</td>
</tr>
<tr>
<td>polygon (translation, rotations)</td>
<td>3D</td>
<td>6D</td>
</tr>
<tr>
<td>Robot arm with joints</td>
<td>3D</td>
<td>#DOF</td>
</tr>
</tbody>
</table>
Motion planning in C-space

- Any path for R corresponds to a path for R in C-space
- We want a path that does not intersect obstacles
Motion planning in C-space

- Free C-space

placement or robot at (x,y) does not intersect obstacles
Motion planning in C-space

- Forbidden C-space

forbidden C-space: placements \((x,y)\) where robot intersects with obstacle

placement or robot at \((x,y)\) intersects obstacle
C-obstacles

- Extended obstacle, or C-obstacle
  - Given obstacle $O$, robot $R(x,y)$: what placements cause intersection with $O$?
C-obstacles

- Extended obstacle, or C-obstacle
- Given obstacle $O$, robot $R(x,y)$: what placements cause intersection with $O$?
C-obstacles

- Extended obstacle, or C-obstacle
  - Given obstacle $O$, robot $R(x,y)$: what placements cause intersection with $O$?
C-obstacles

- Extended obstacle, or C-obstacle
  - Given obstacle $O$, robot $R(x,y)$: what placements cause intersection with $O$?
Translational motion planning

- Algorithm for polygonal robot translating in 2D
  - For each obstacle $O$, compute the corresponding C-obstacle
  - Compute the union of C-obstacles
  - Compute its complement. That’s the free C-space.

  //now the problem is reduced to point robot moving in free C-space

- Compute a trapezoidal map of free C-space
- Compute a roadmap
Translational motion planning

- Algorithm for polygonal robot translating in 2D
  - For each obstacle $O$, compute the corresponding C-obstacle
  - Compute the union of C-obstacles
  - Compute its complement. That's the free C-space.

  //now the problem is reduced to point robot moving in free C-space
  
- Compute a trapezoidal map of free C-space
- Compute a roadmap

How long?  
Is it complete? Optimal?
Path planning: A Preview of Approaches

**Geometric/combinatorial**

- Roadmaps
  - via trapezoidal decomposition of free space
  - Shortest path-roadmaps (Visibility graph)
  - max-clearance roadmaps (Voronoi diagrams)

**Pros:**
- complete, optimal
- elegant

**Cons:**
- practical only in simple instance (disks, convex shapes, <= 3 DOF)

**Approximate/Heuristical**

- approximate cell decomposition
- potential methods
- incremental sampling
- probabilistic roadmaps

**Pros:**
- work in many realistic scenarios and finds paths

**Cons:**
- not complete, not optimal
- often no efficiency guarantees