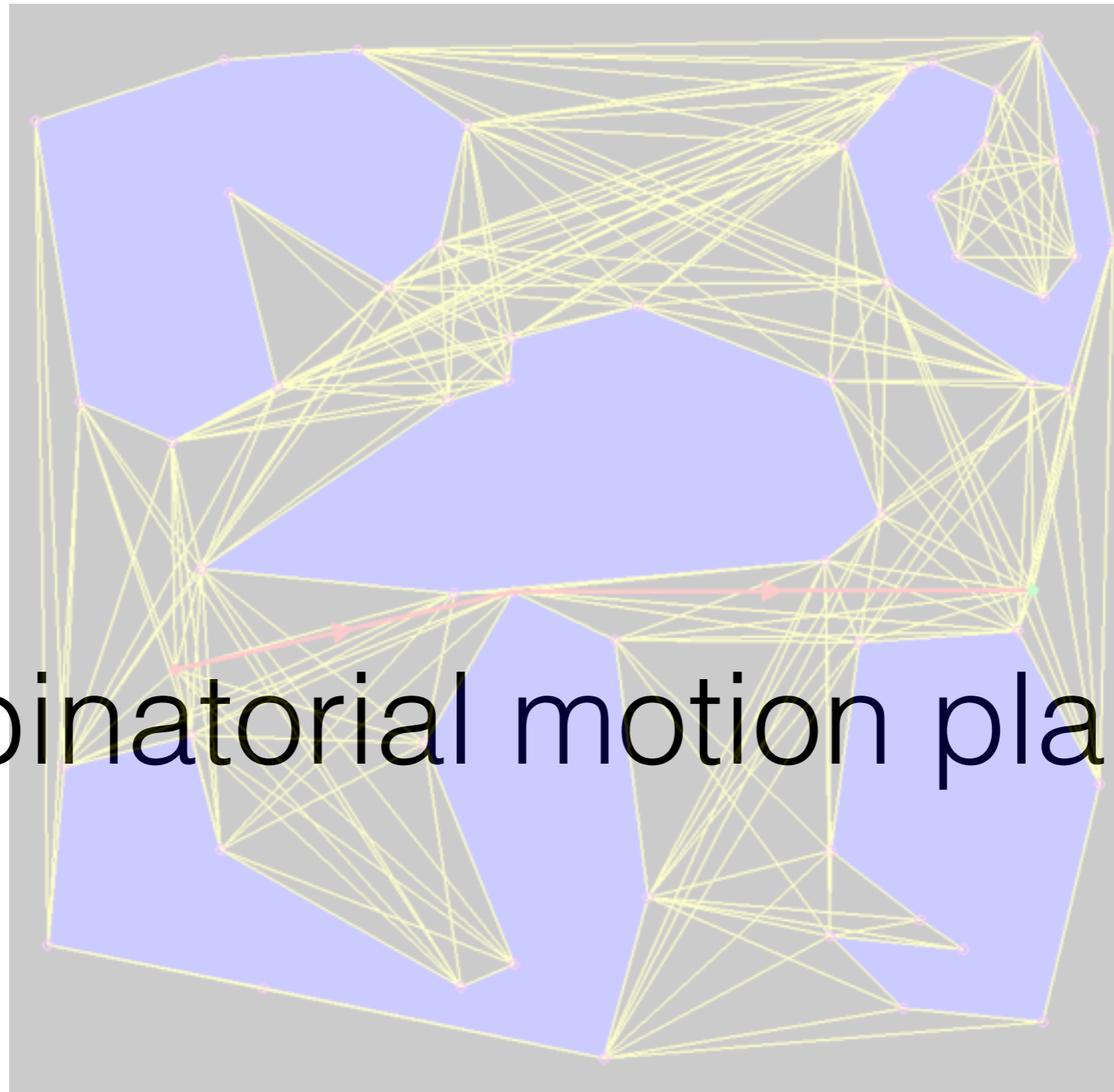


Combinatorial motion planning



Computational Geometry
csci3250
Laura Toma
Bowdoin College

Motion Planning

Input:

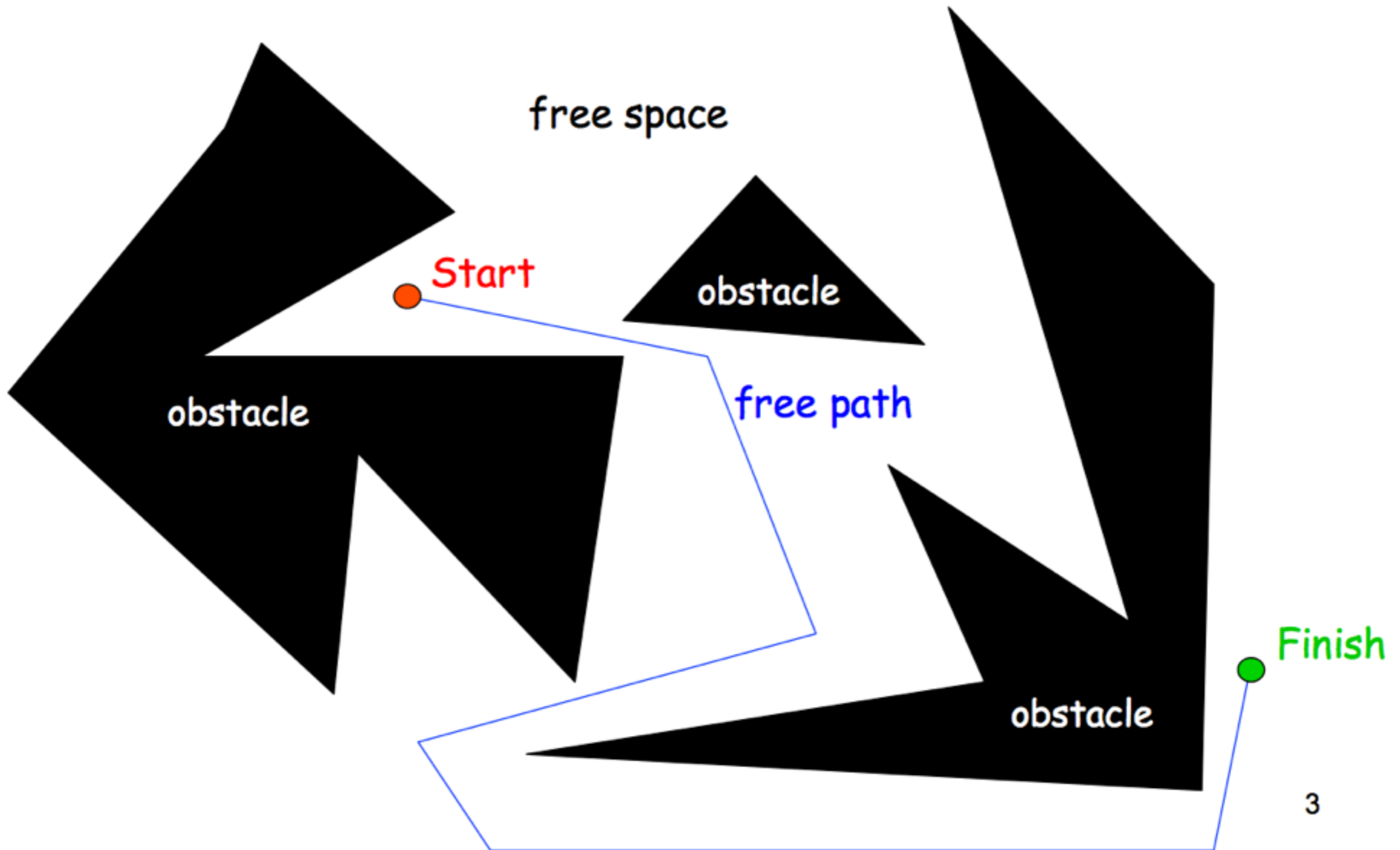
- a robot R
- start and end position
- a set of obstacles $S = \{O_1, O_2, \dots\}$

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

Problem



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 (wrt objective function)

Approaches

- **Combinatorial (exact)**

- Compute an exact representation of free space
- Roadmap: a graph that represents the free space
- Find a path using the roadmap



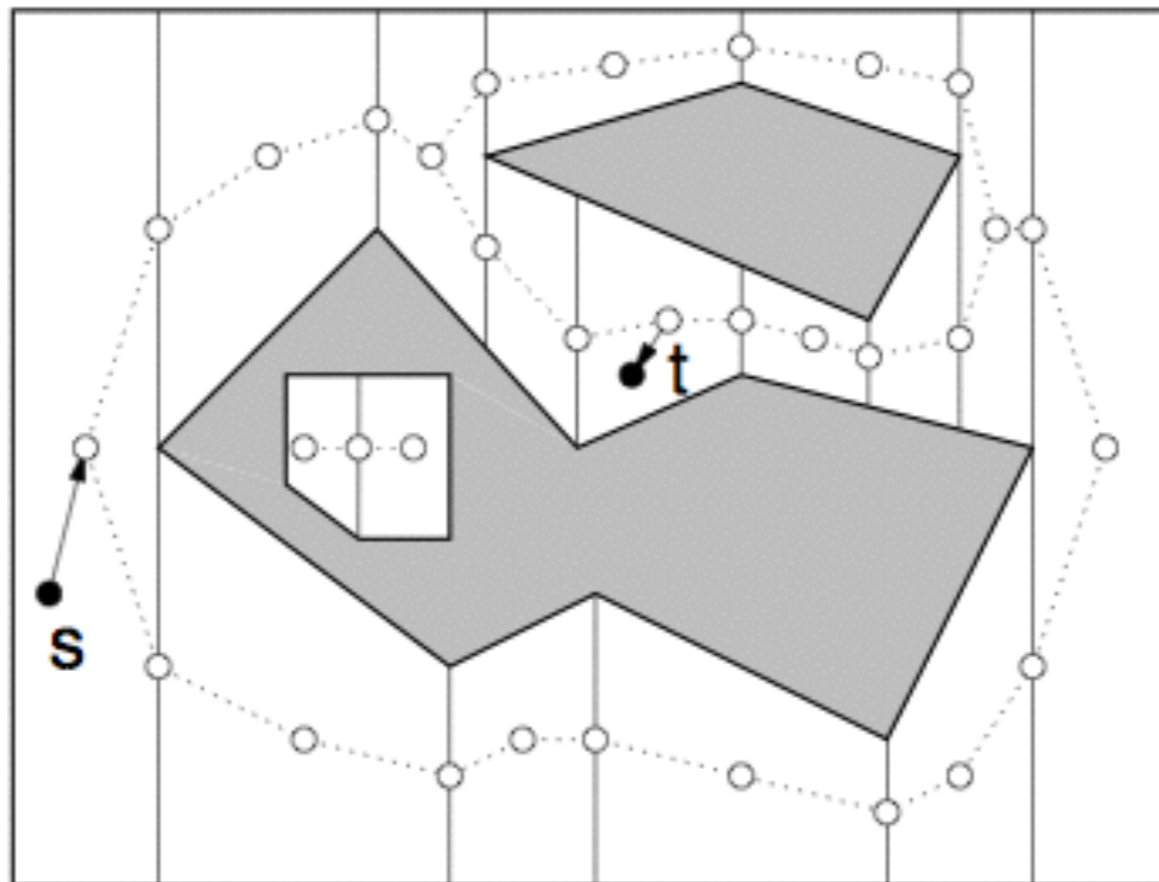
today

- **Approximate**

- roadmaps, sampling-based, search based, space decomposition, probabilistic, potential functions

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

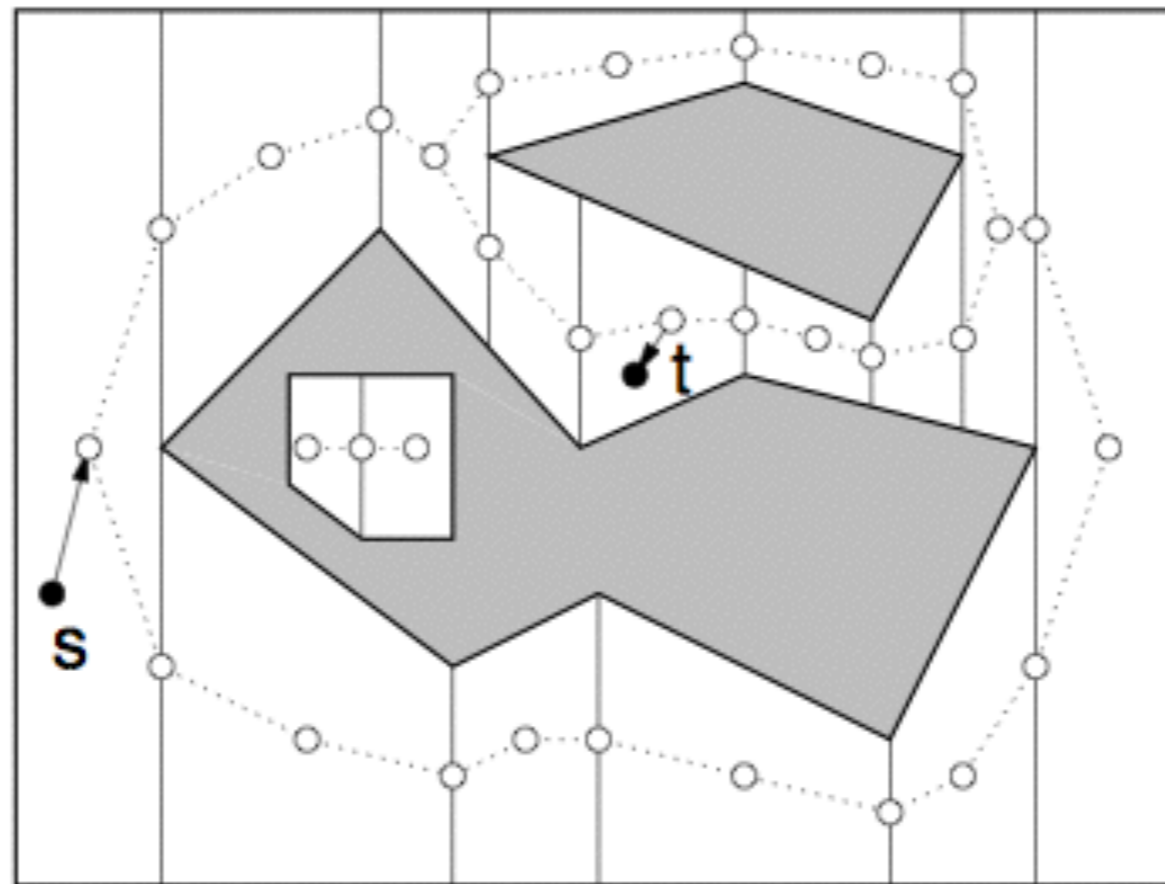


n = complexity of obstacles
(total number of edges)

Point robot in 2D

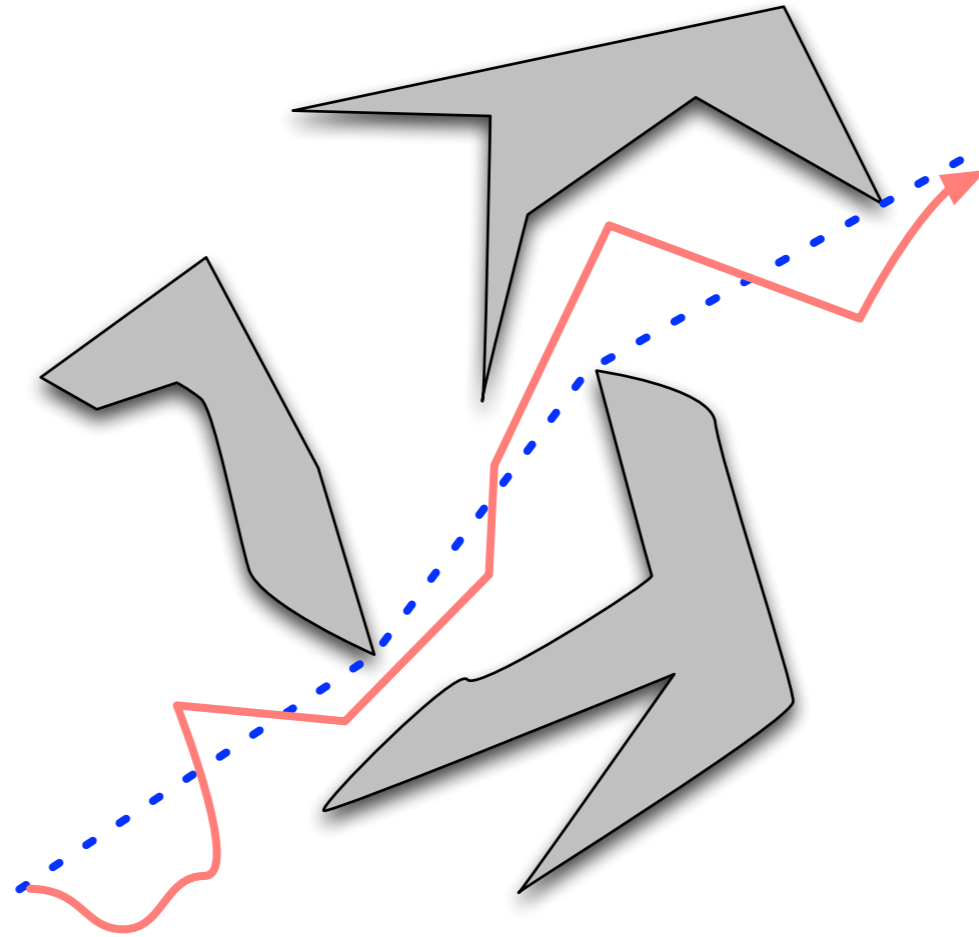
Result:

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 $O(n \lg n)$ 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.



Note: complete, but not optimal

Point robot in 2D

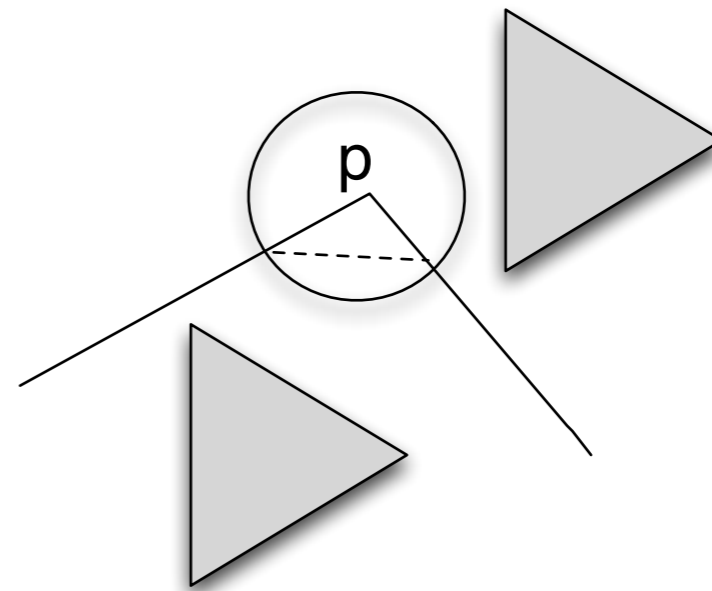
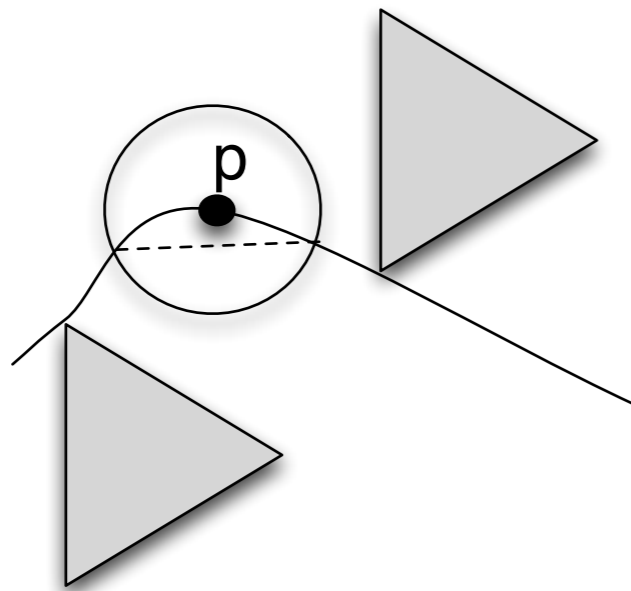


What if we wanted the shortest path?

Shortest paths for point robot in 2D

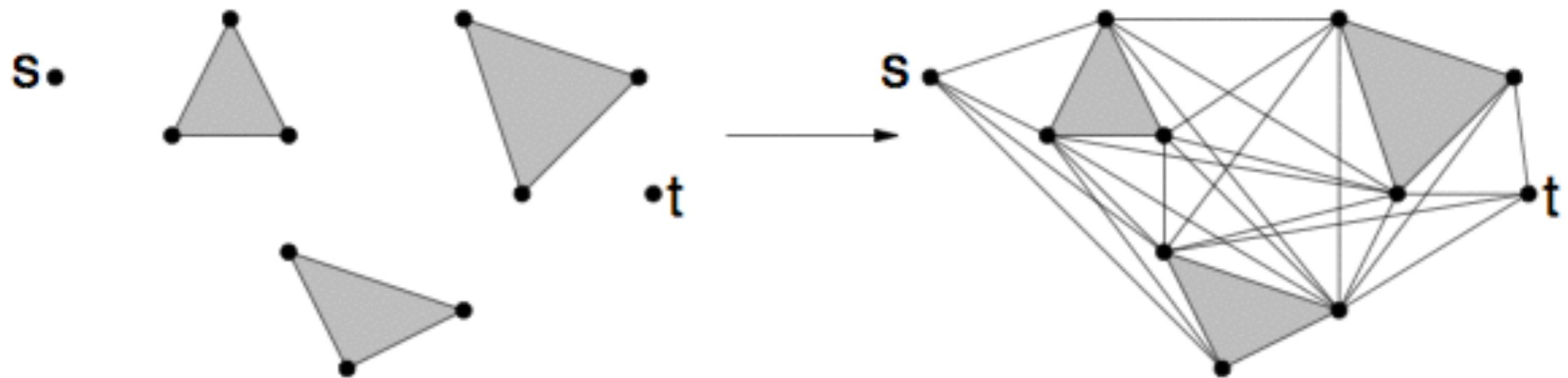
Claim: Any shortest path among a set S of disjoint polygonal obstacles

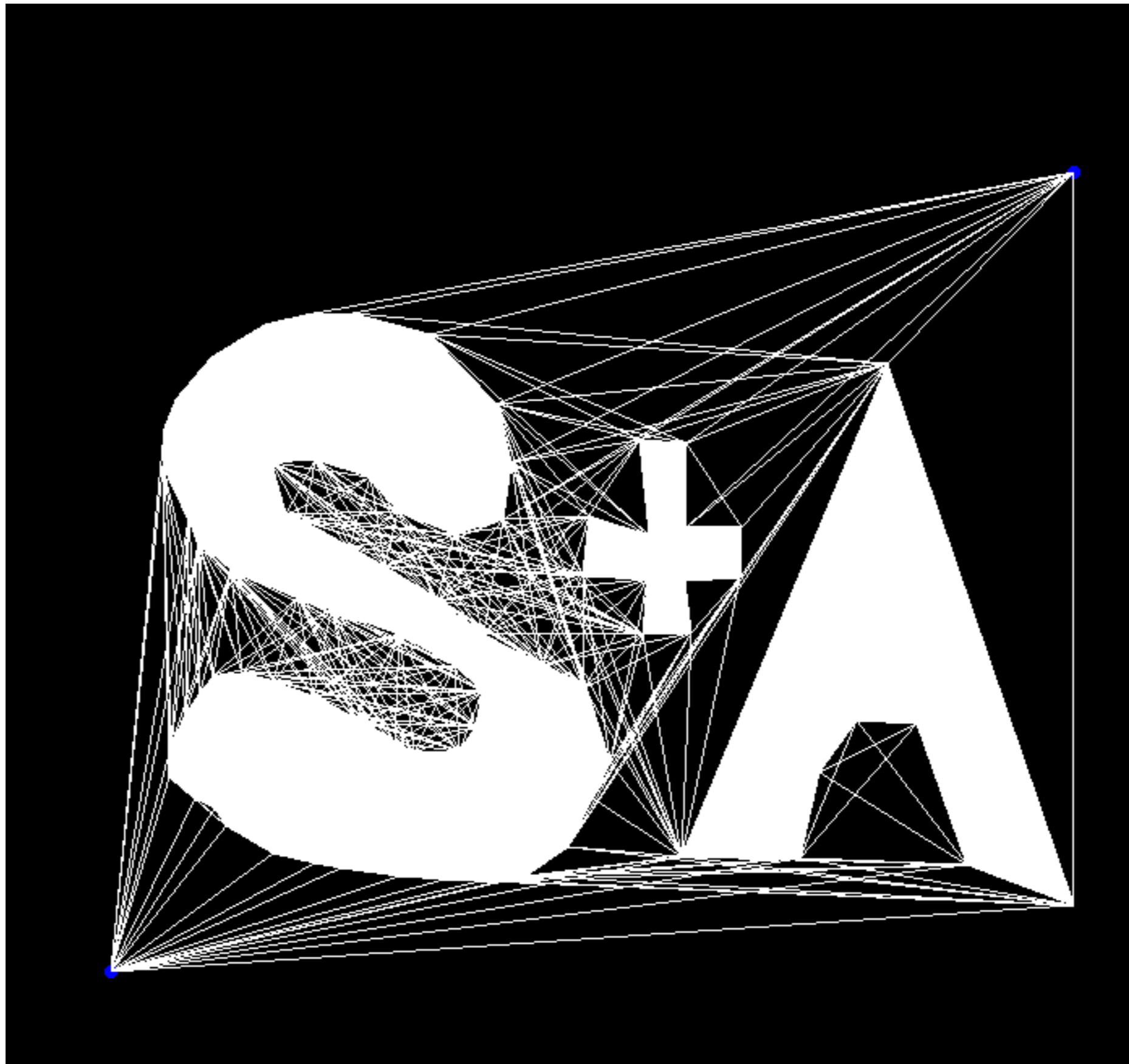
- is a polygonal path (that is, not curved)
- its vertices are the vertices of S .

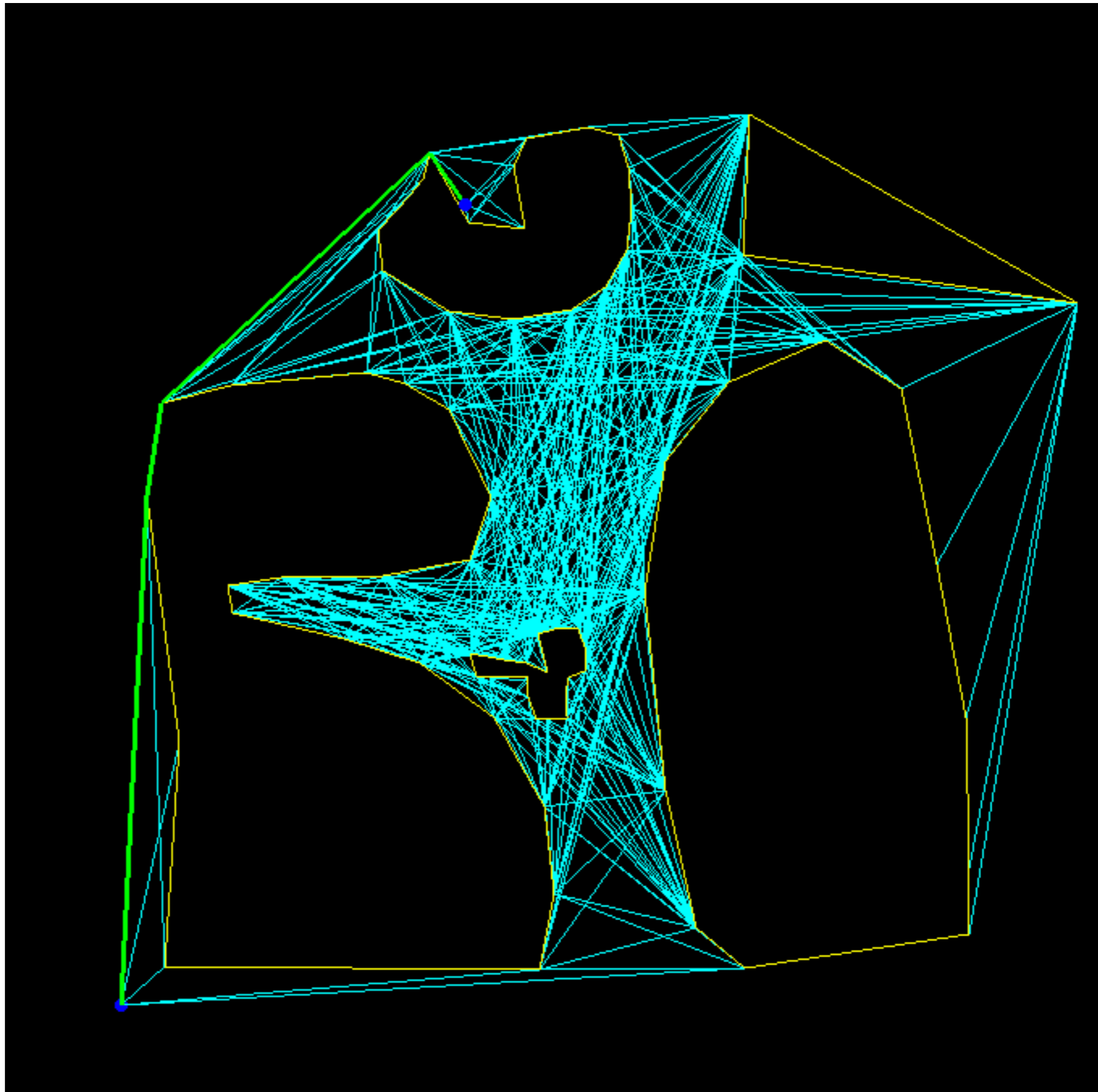


Shortest paths for point robot in 2D

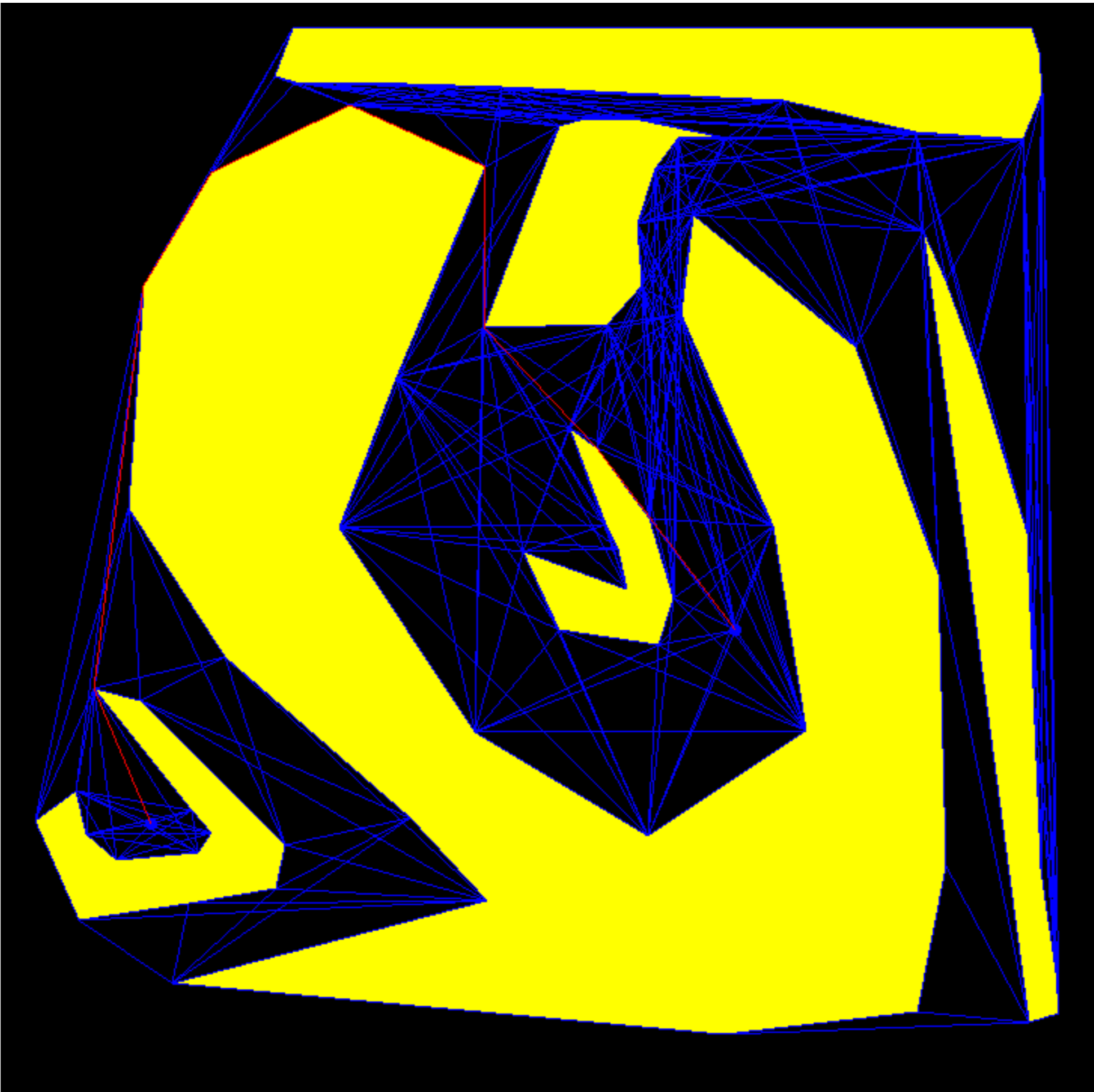
- Idea: Build the visibility graph
 - all possible ways to travel between the vertices of the obstacles
- Claim: any shortest path must be a path in the VG

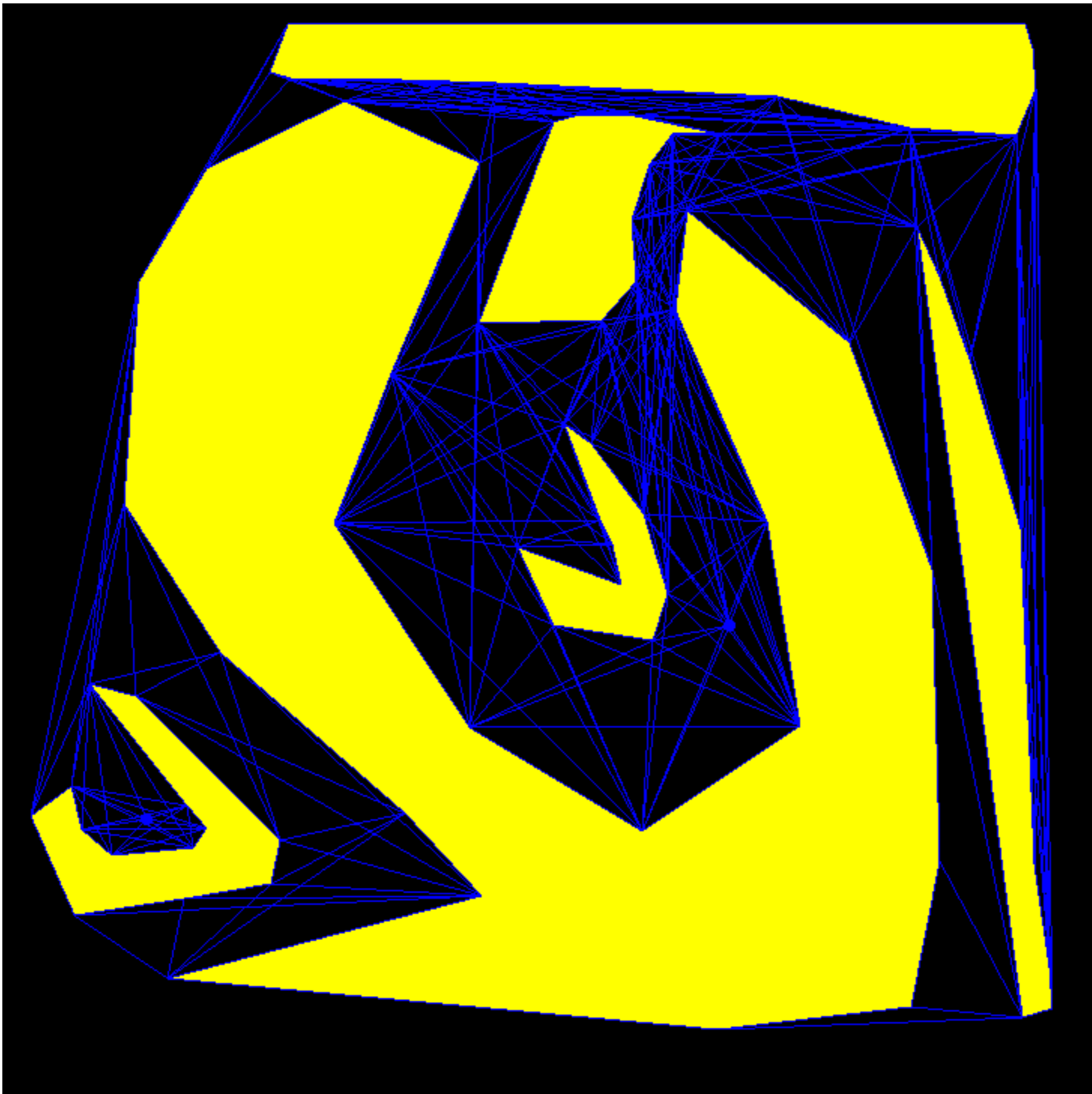


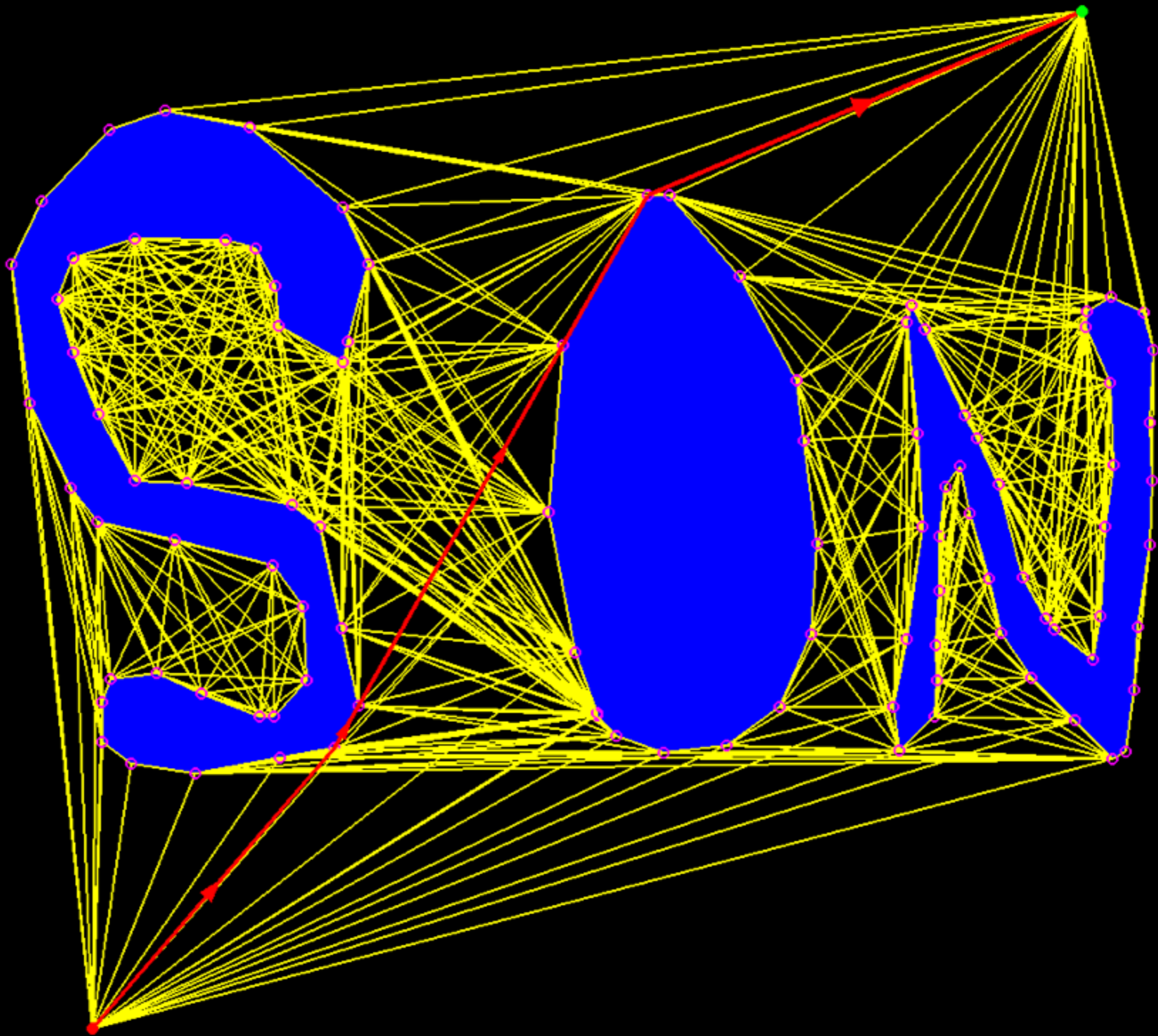












n = complexity of obstacles
(total number of edges)

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

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
 - $V_{VG} = O(n), E_{VG} = O(n^2)$ <----- 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: $O(E_{VG} \lg n) = O(n^2 \lg n)$

n = complexity of obstacles
(total number of edges)

Shortest paths for point robot in 2D

Algorithm

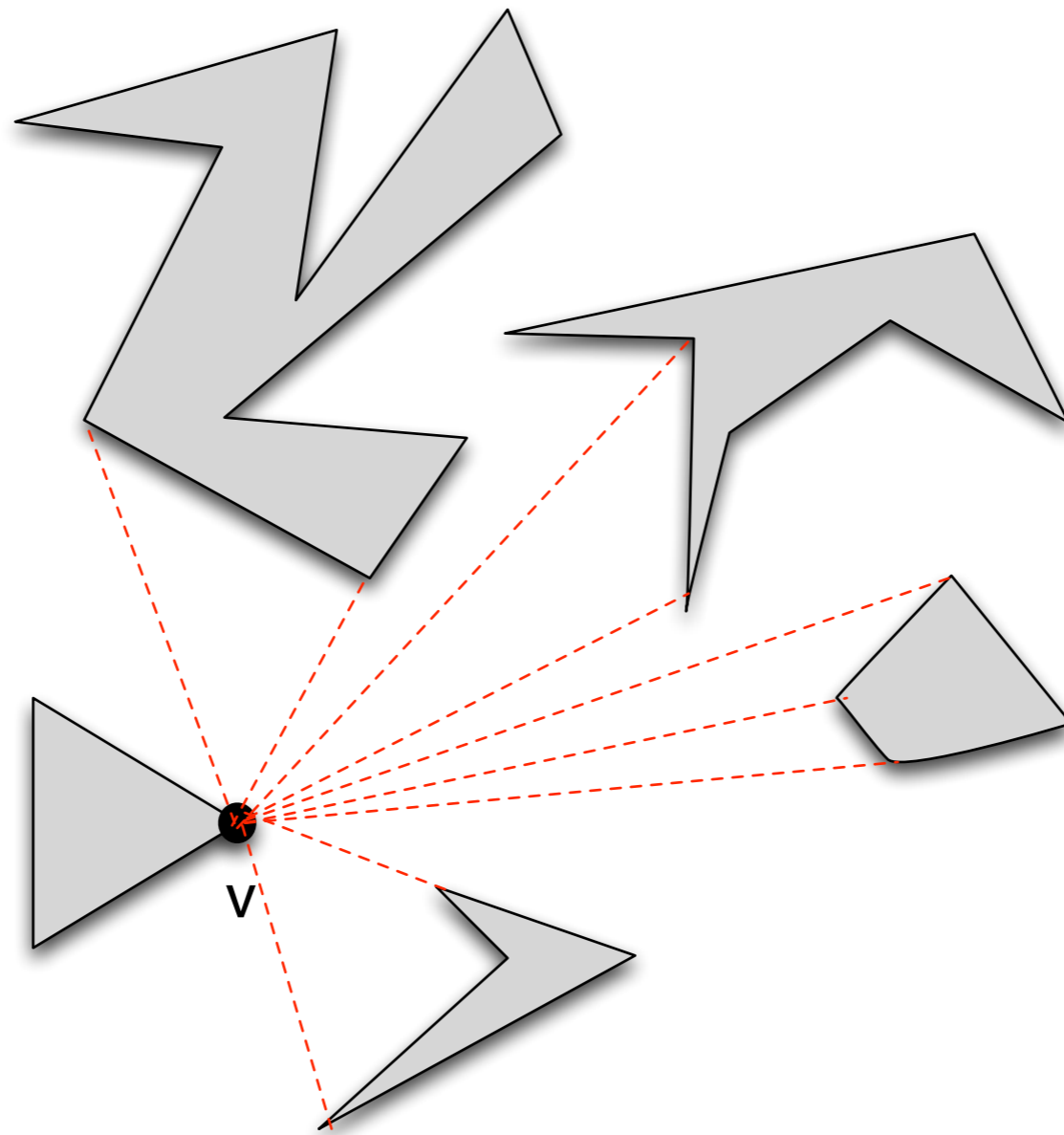
- Compute visibility graph $\leftarrow O(n^2 \lg n)$
- SSSP (Dijkstra) in VG $\leftarrow O(E_{VG} \lg n)$

Theorem:

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

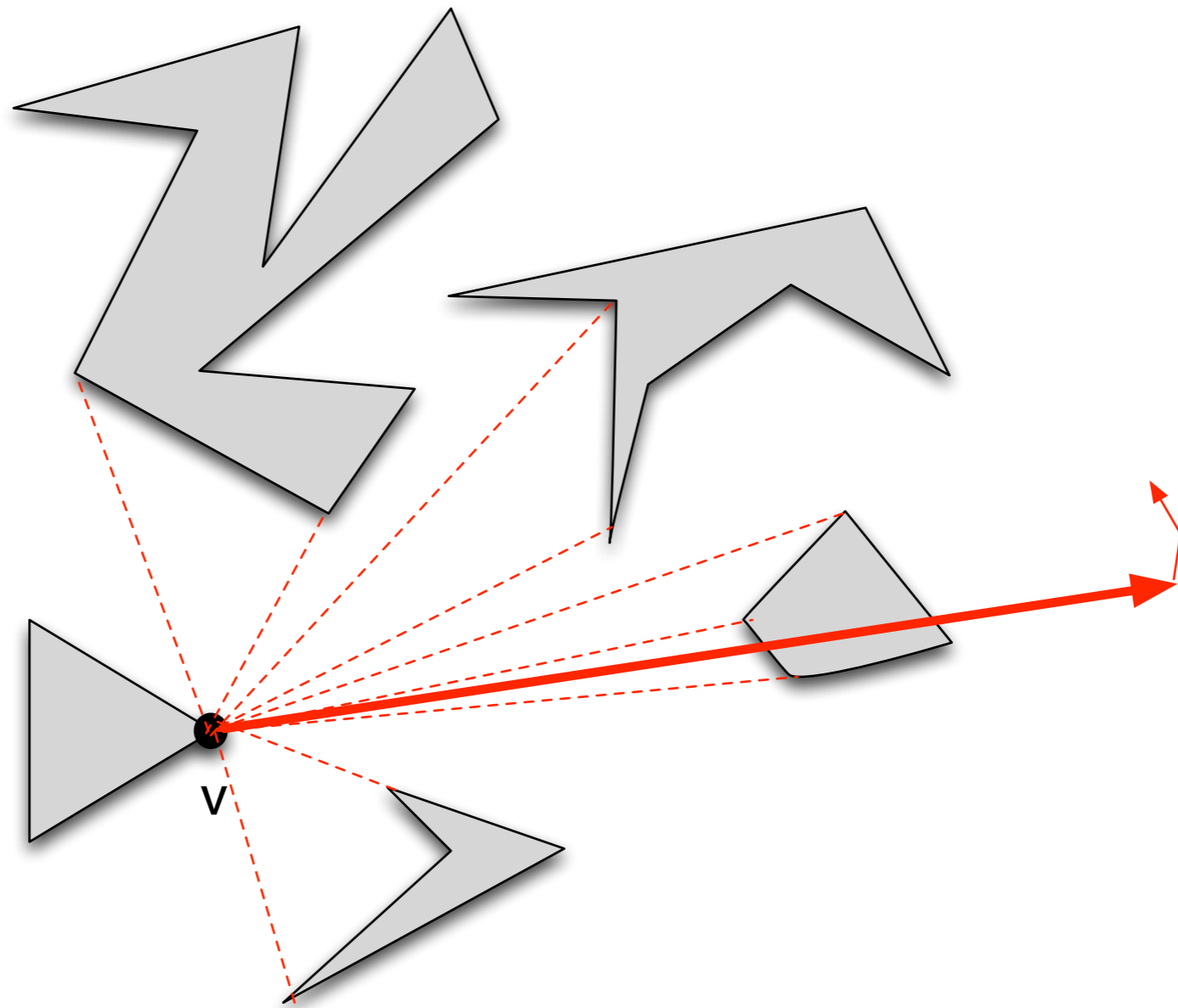
Improved computation of VG

- **Idea**
 - for every vertex v : compute all vertices visible from v in $O(n \lg n)$



Improved computation of VG

- **Idea**
 - for every vertex v : compute all vertices visible from v in $O(n \lg n)$



Improved computation of VG

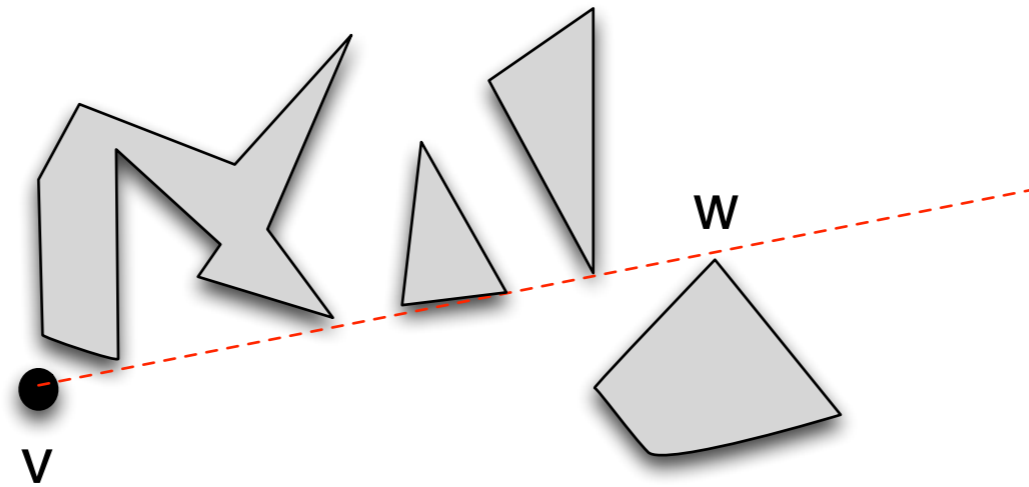
- **Idea**
 - for every vertex v : compute all vertices visible from v in $O(n \lg n)$



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 in $O(n \lg n)$



w visible if vw does not intersect the interior of any obstacle

Motion planning with VG

Comments

- Optimal and complete
- VG needs to be computed only once, so we can think of it as pre-processing
- VG may be large \implies path planning doomed to quadratic complexity

Some history:

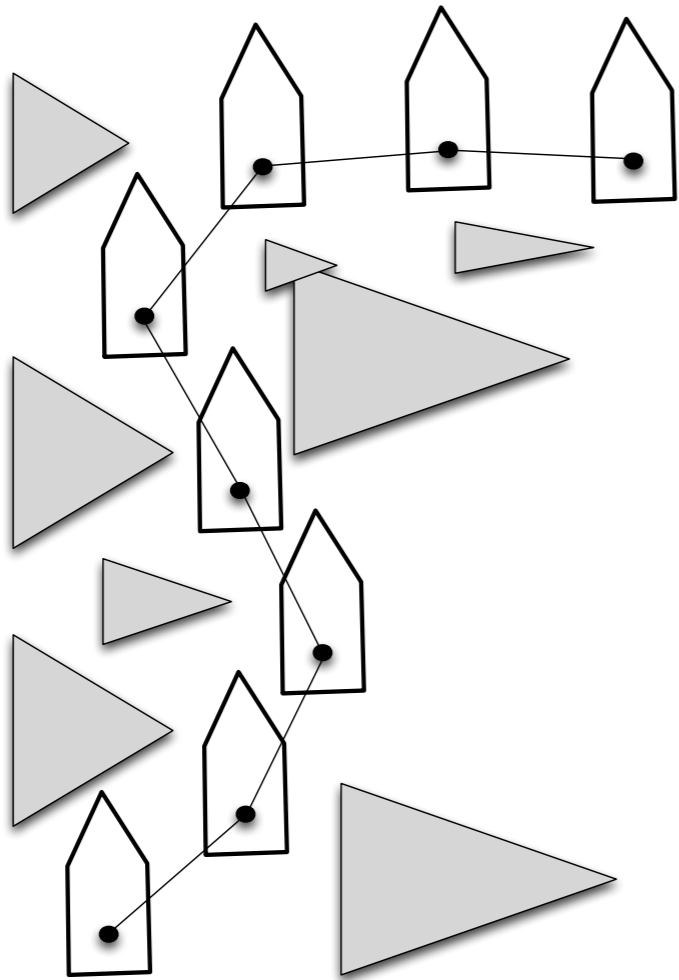
- Quadratic barrier broken by Joe Mitchell: SP of a point robot moving in 2D can be computed in $O(n^{5/3 + \epsilon})$
- Hershberger and Suri [1993]: SP of a point robot moving in 2D can be computed in $O(n \lg n)$ (“continuous Dijkstra” approach)
- Special cases can be solved faster:
 - e.g. SP inside a simple polygon w/o holes: $O(n)$ time

VG in 3D

- VG does not generalize to 3D
- Shortest paths 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 and optimal planning in 3D is hopeless

Convex polygon moving in 2D

- How can the robot move?
 - Translation only
 - Translation + rotation



screenshot from internet

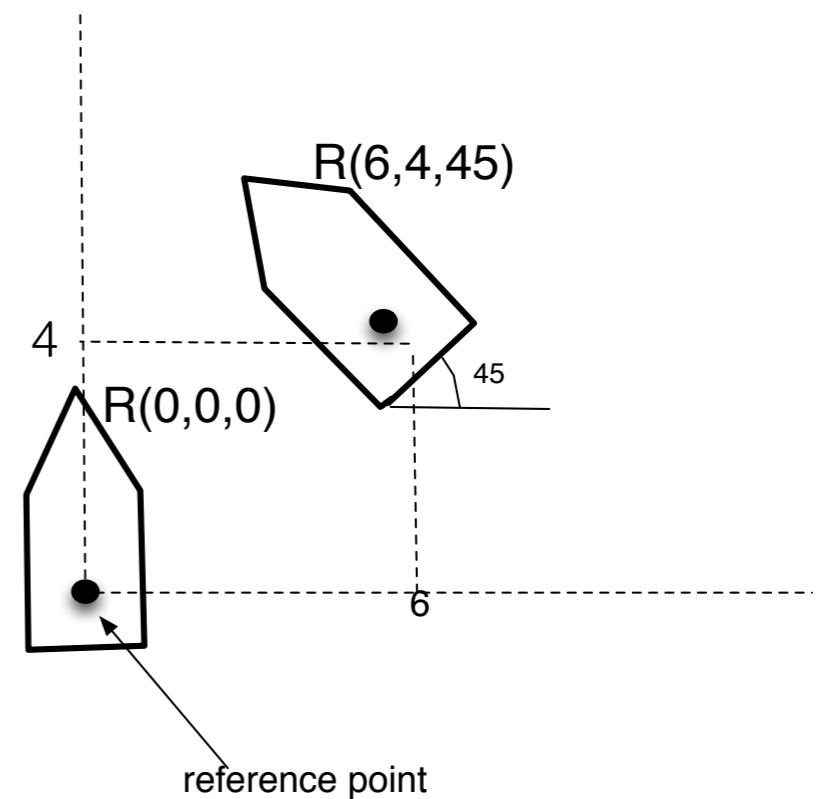
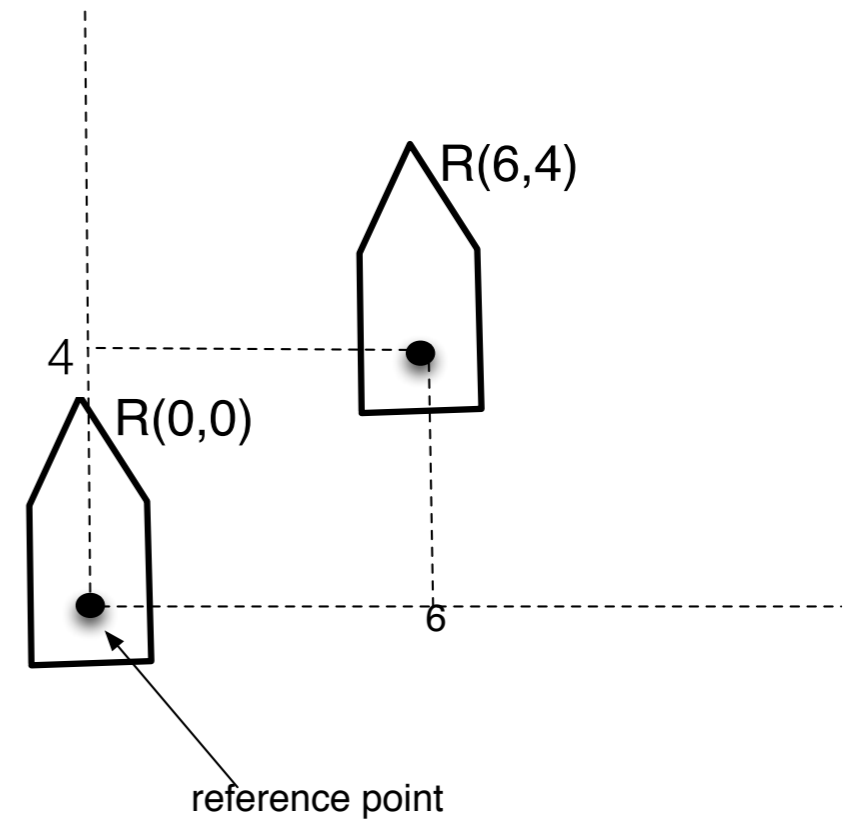
Work/physical space

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

- Example:

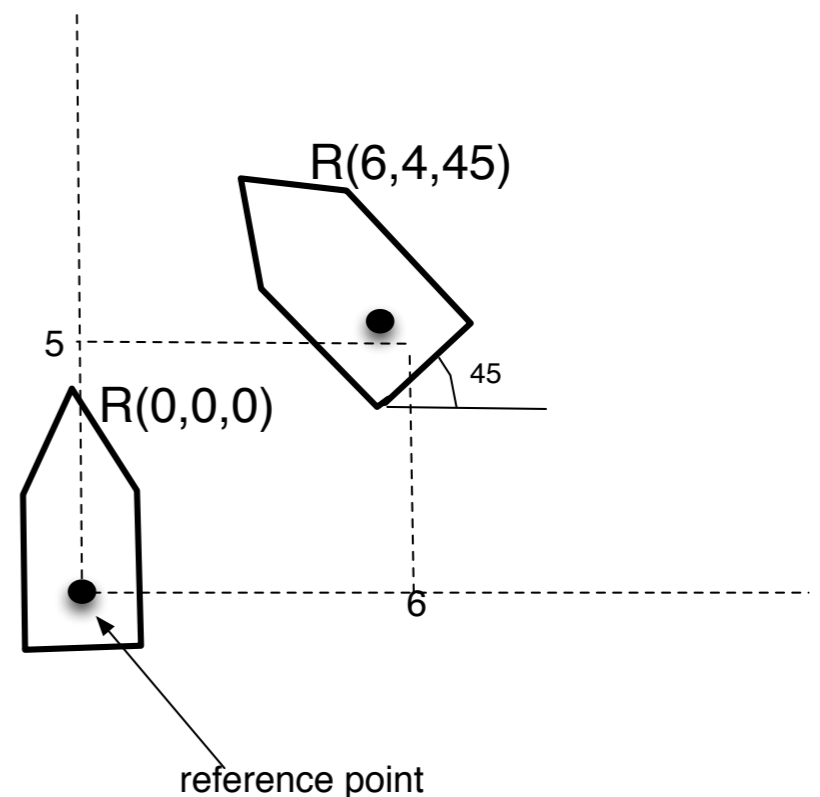
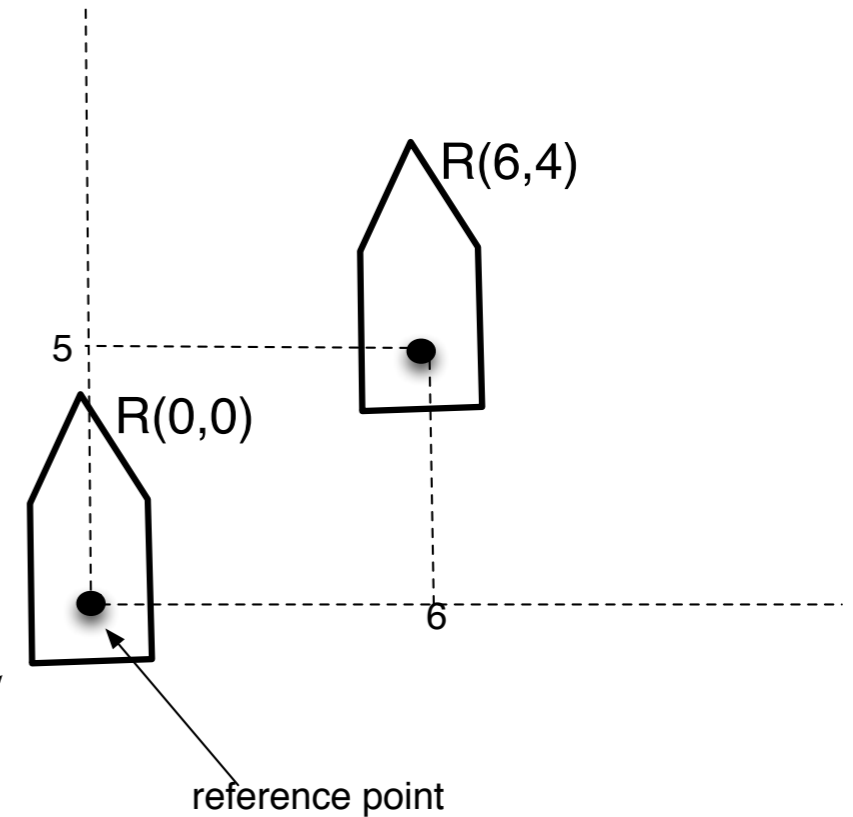
$R(x,y)$

$R(x,y,\theta)$



Configuration space (C-space)

- A point in C-space corresponds to placement of the robot in physical space
- C-space: The parametric space of the robot = space of all possible placements of the robot
- A path corresponds to a path in C-space
- **Examples:**
 - 2D, translation only $\leftrightarrow R(x,y)$
 - 2D, transl.+ rot. $\leftrightarrow R(x,y, \theta)$

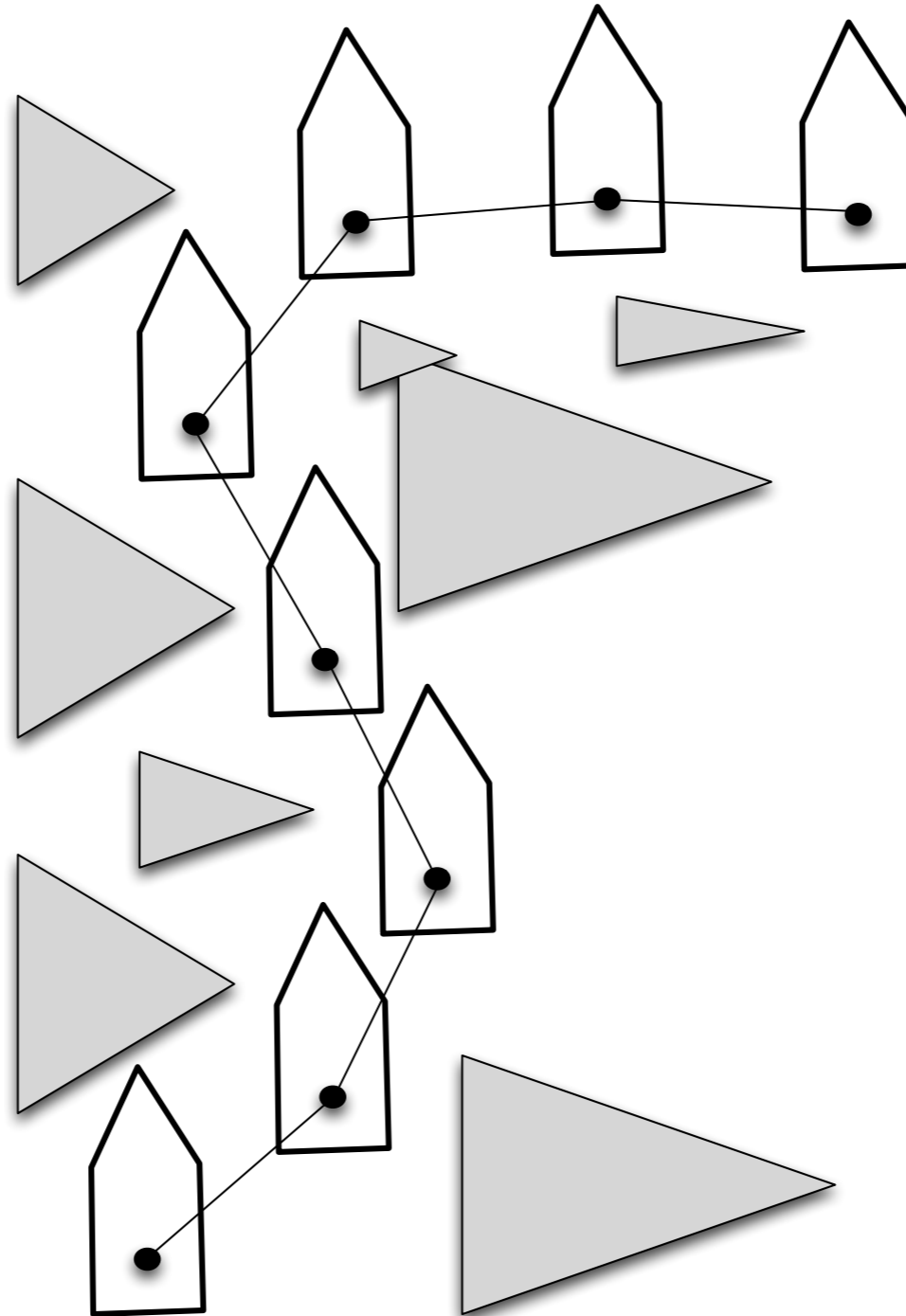


Physical Space and C-space

robot	physical space	C-space
polygon, (translation only)	2D	2D
polygon, (translation + rotations)	2D	3D
polygon (translation, rotations)	3D	6D
Robot arm with joints	3D	#DOF

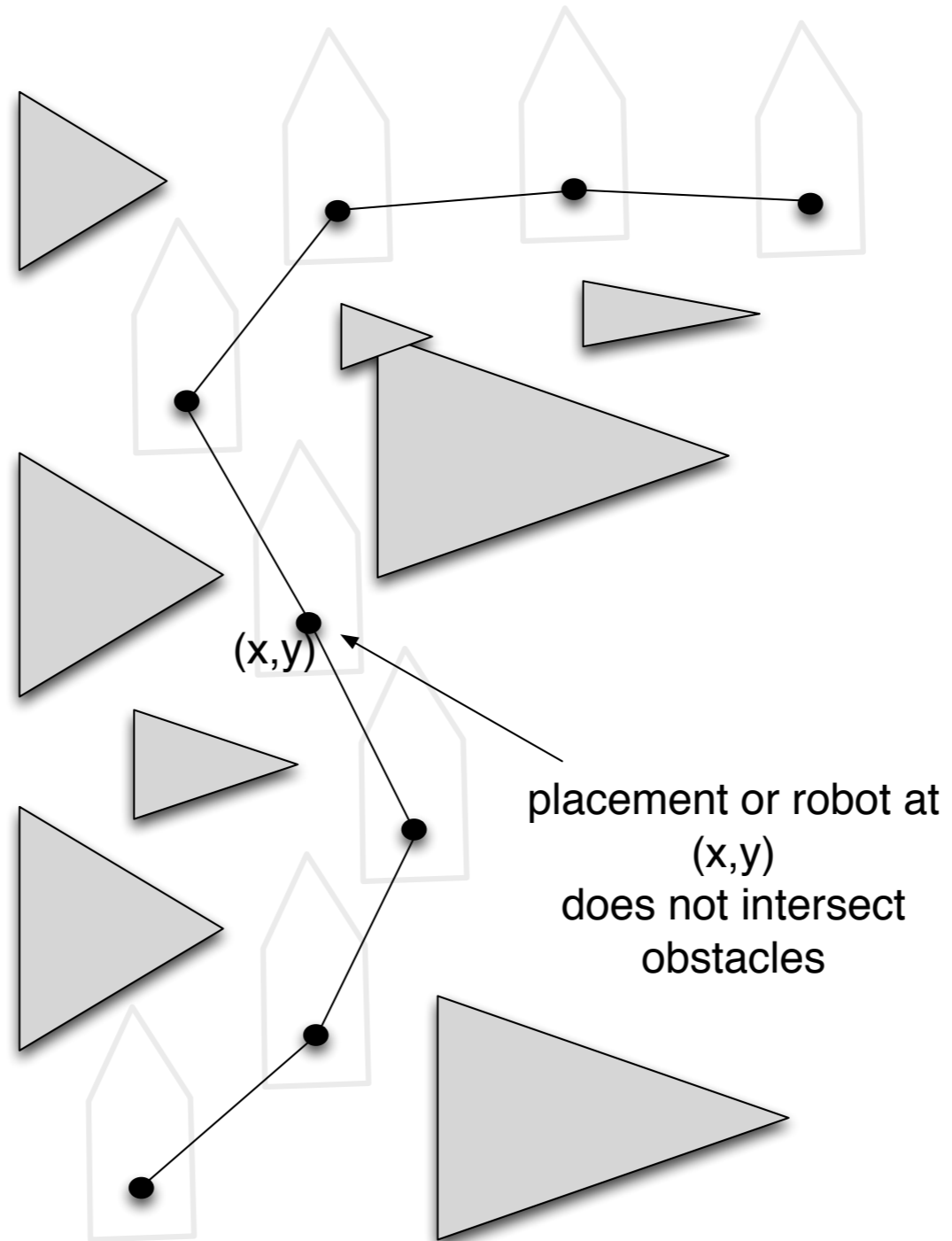
Motion planning in C-space

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



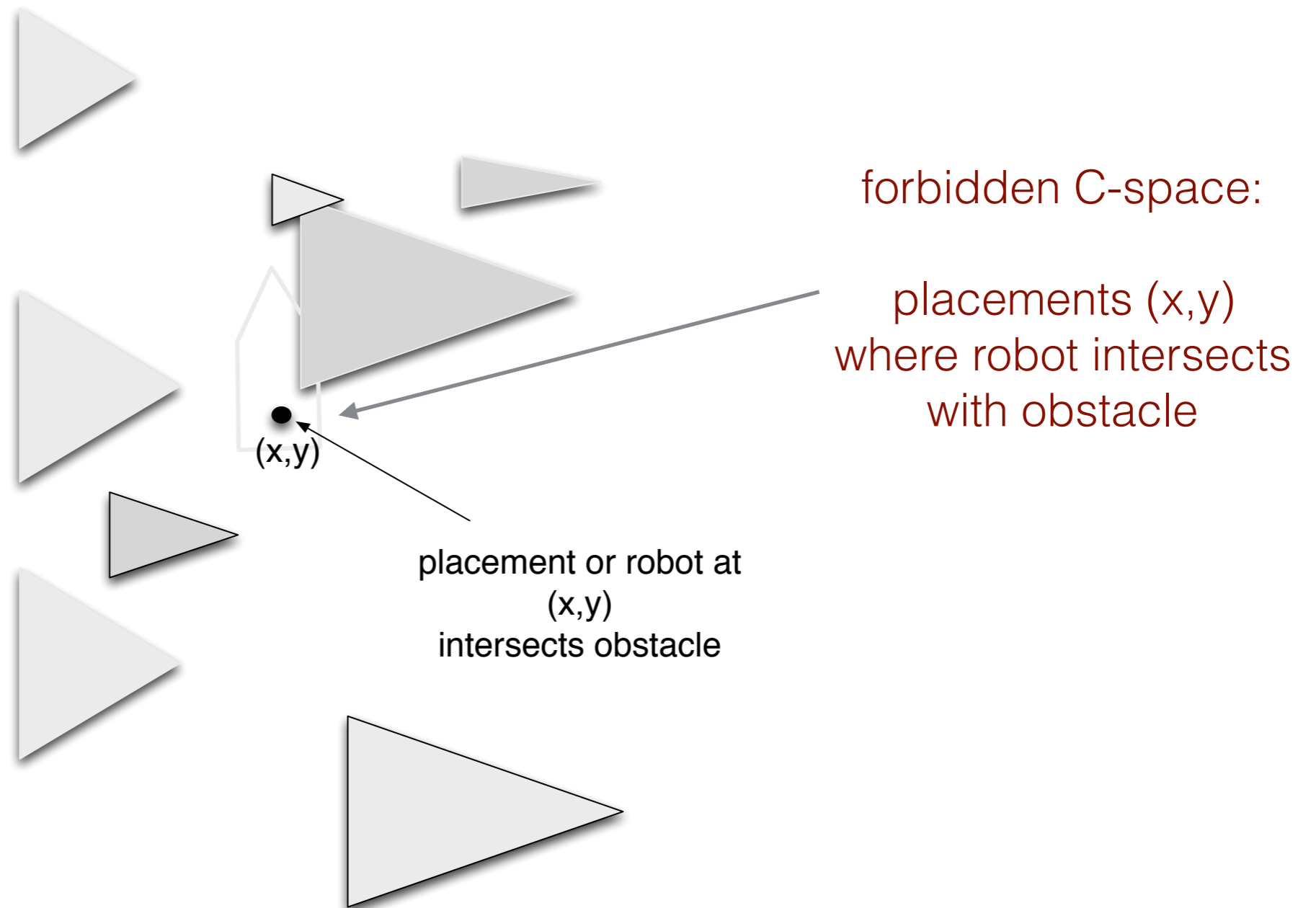
Motion planning in C-space

- Free C-space



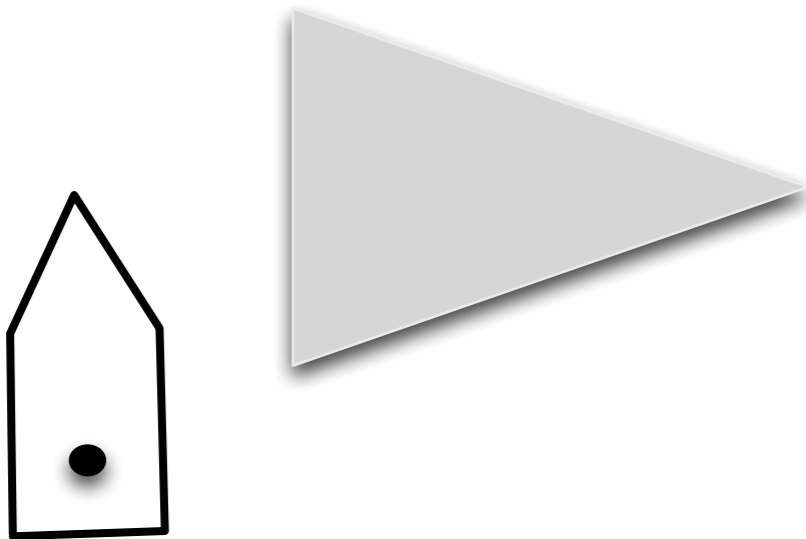
Motion planning in C-space

- Forbidden C-space



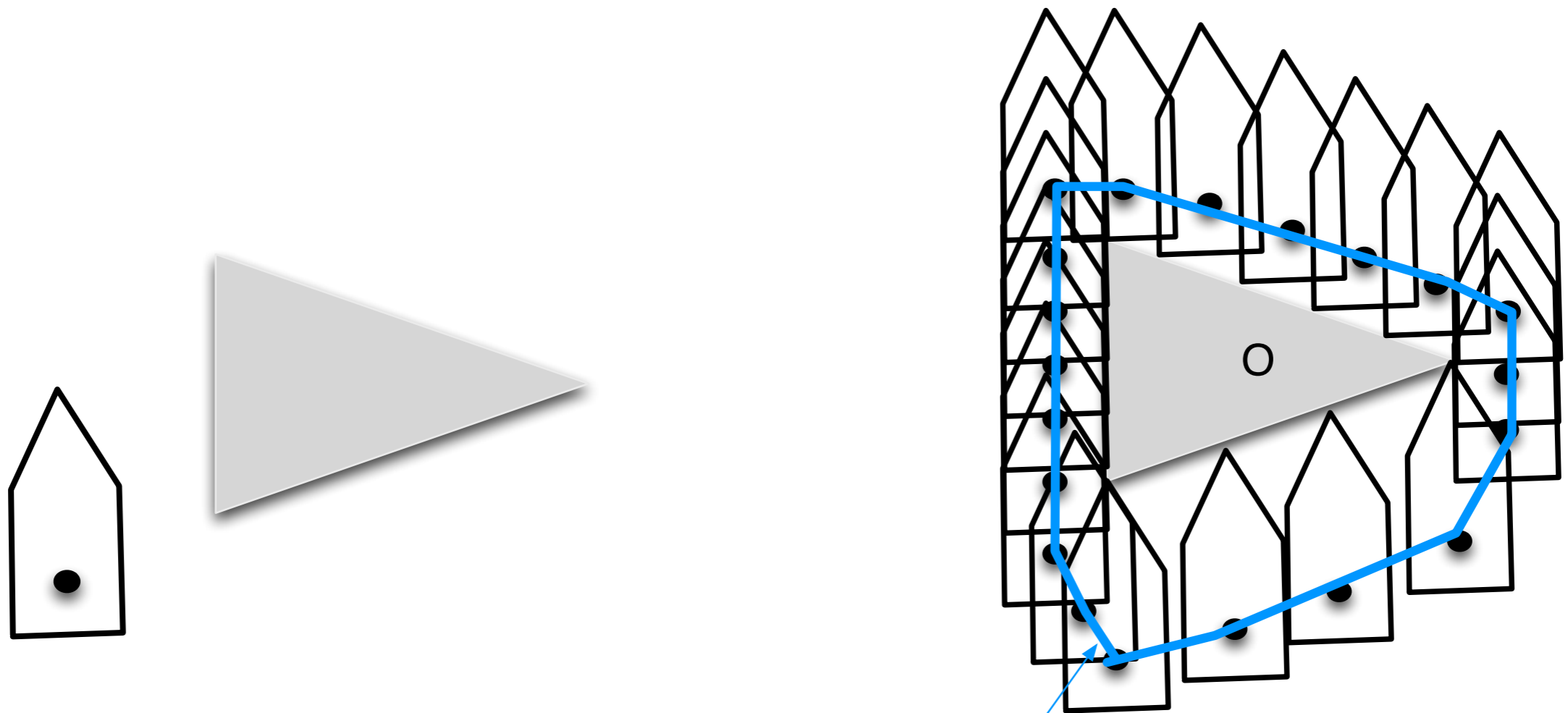
C-obstacles

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



C-obstacles

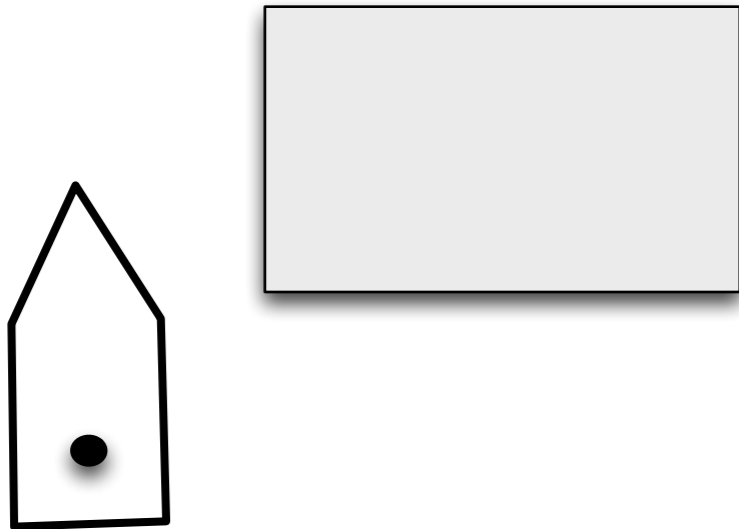
- **Extended obstacle, or C-obstacle**
 - Given obstacle O , robot $R(x,y)$: what placements cause intersection with O ?



C-obstacle corresponding to O

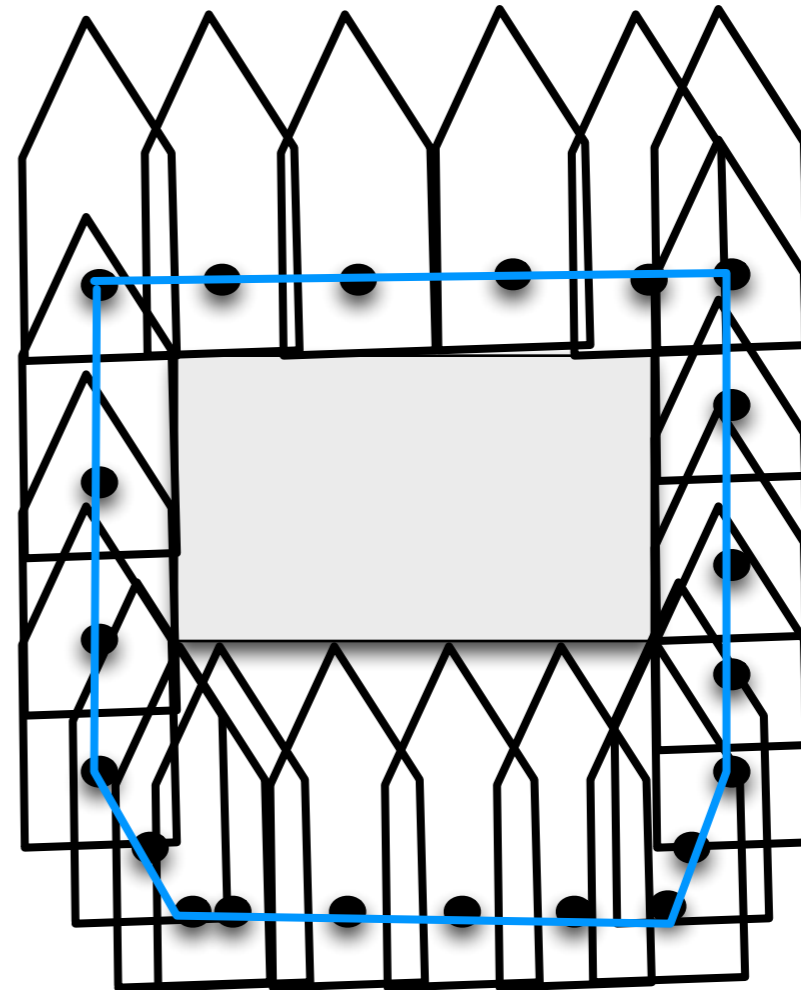
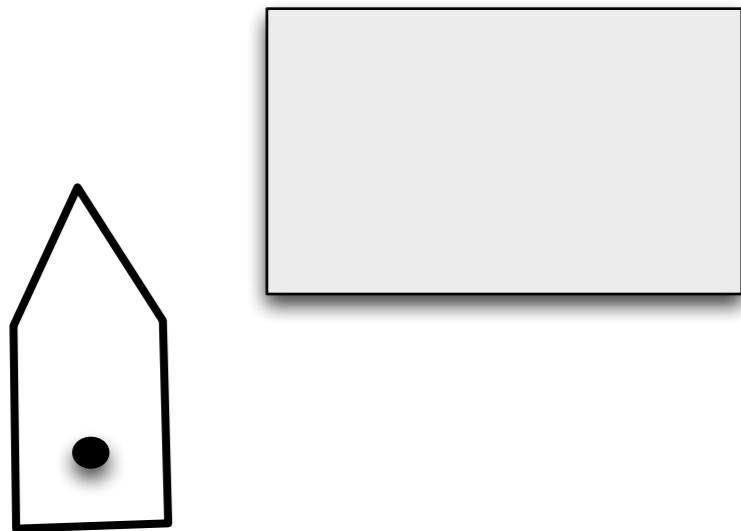
C-obstacles

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

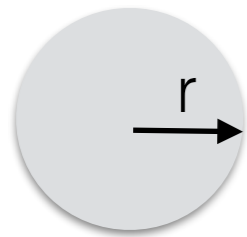


C-obstacles

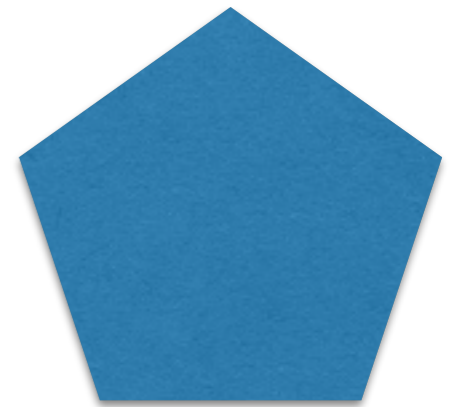
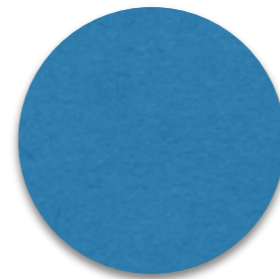
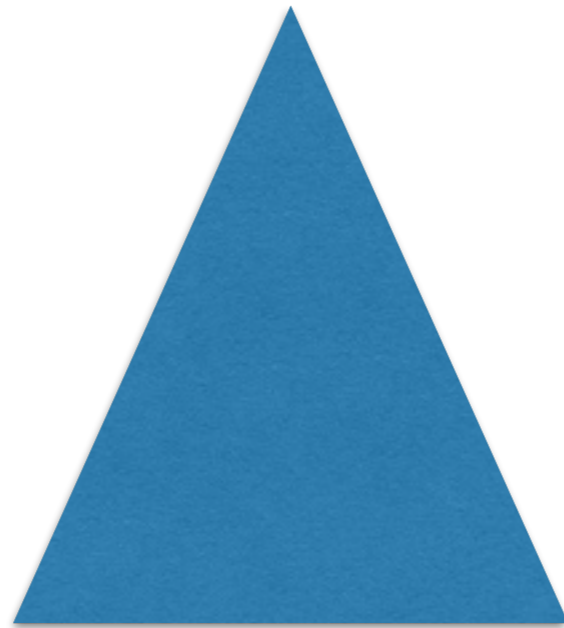
- Extended obstacle, or C-obstacle
 - Given obstacle O , robot $R(x,y)$: what placements cause intersection with O ?



Exercise

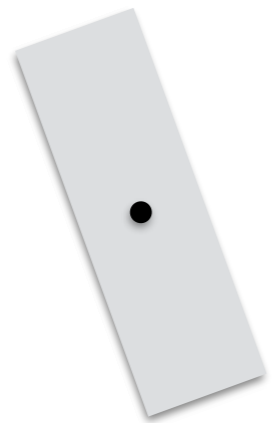


robot

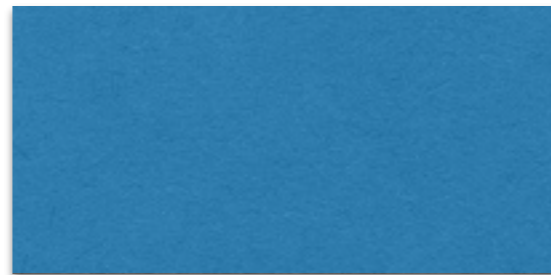


Show the corresponding C-obstacles for a disc robot.

Exercise



robot



Show the corresponding C-obstacle.

Polygonal robot translating in 2D

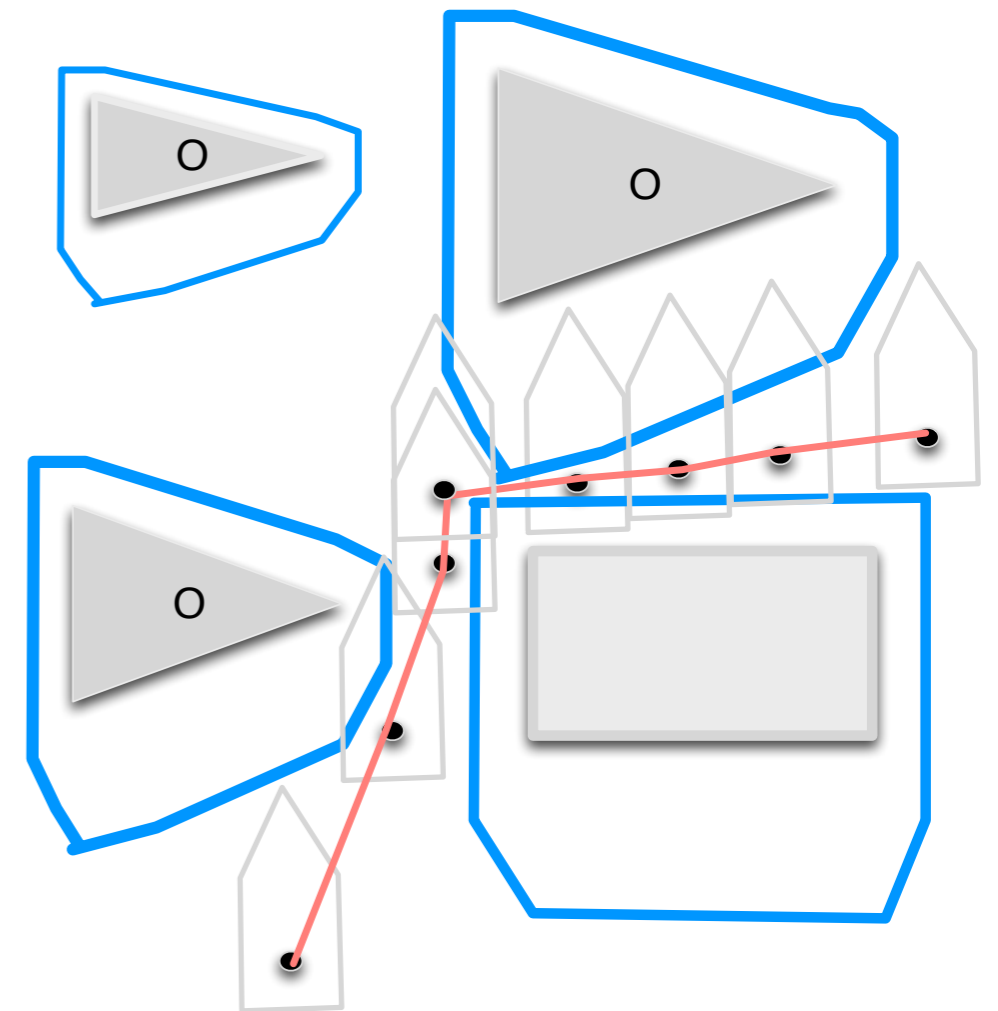
Algorithm

- 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 a point

//robot moving in free C-space

- Compute a trapezoidal map of free C-space
- Compute a roadmap



How fast can we do this?

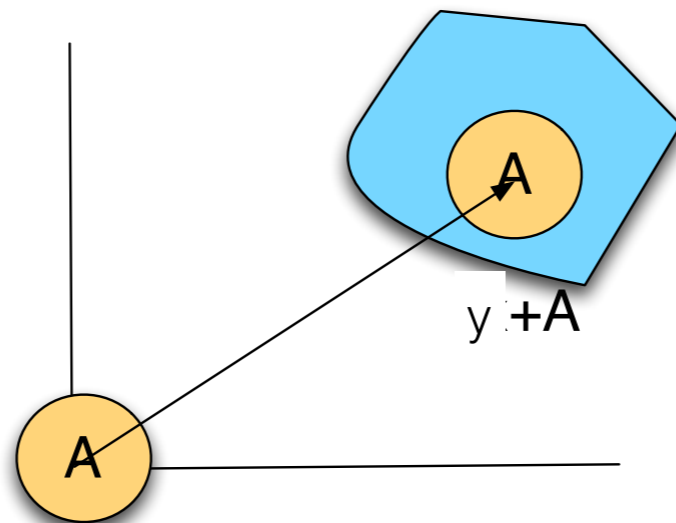
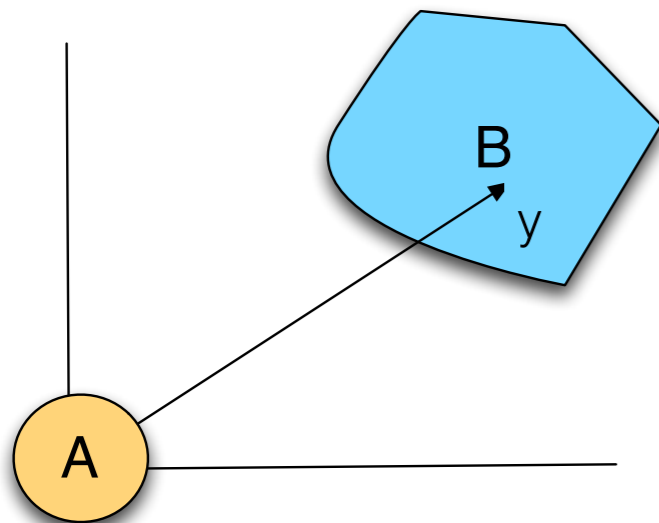
How do we compute C-obstacles?

Minkowski sum

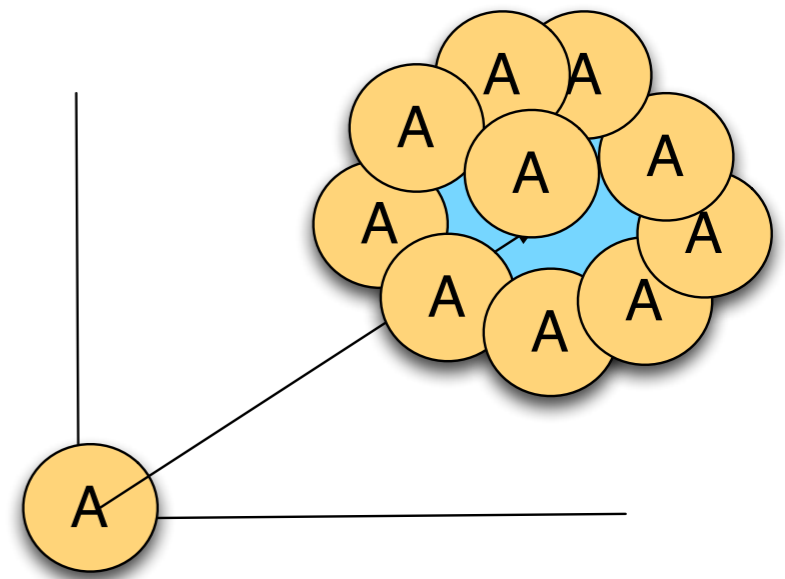
- Let A, B two sets of points in the plane
- Define $A \oplus B = \{x + y \mid x \text{ in } A, y \text{ in } B\}$ ← Minkowski sum



- Interpretation: consider set A to be centered at the origin. Then $A \oplus B$ represents many copies of A , translated by y , for all y in B ; i.e. place a copy of A centered at each point of B .



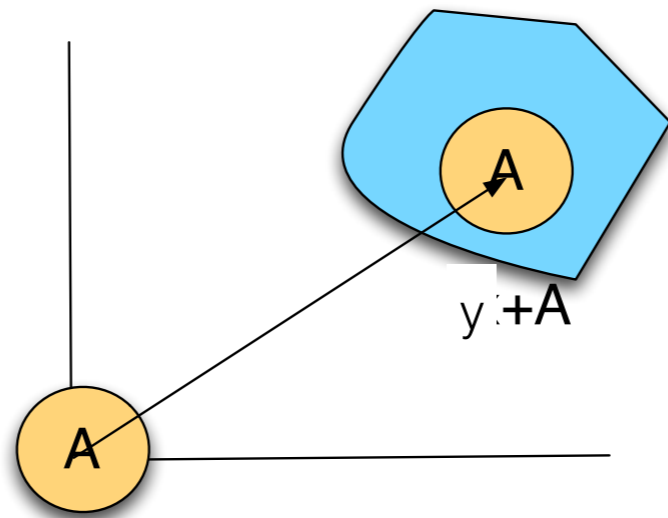
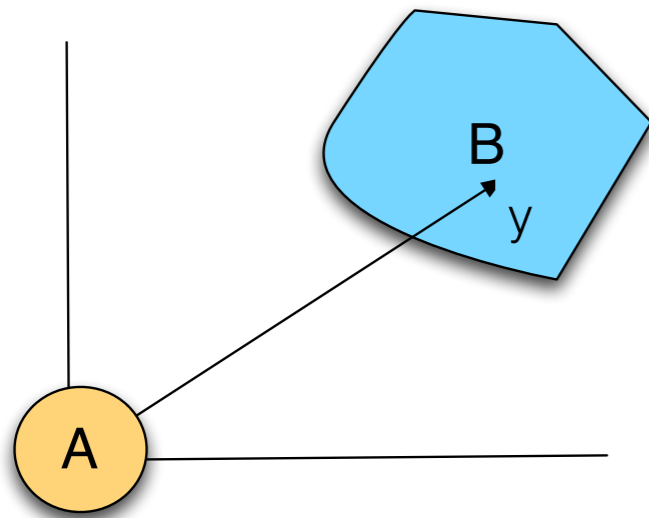
A translated by y



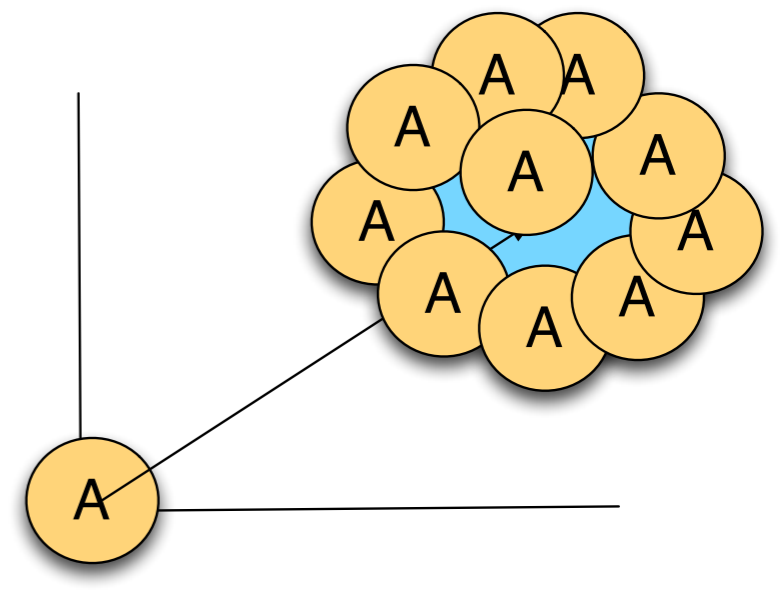
$B \oplus A$

Minkowski sum

- $A \oplus B$: Slide A so that the center of A traces the edges of B



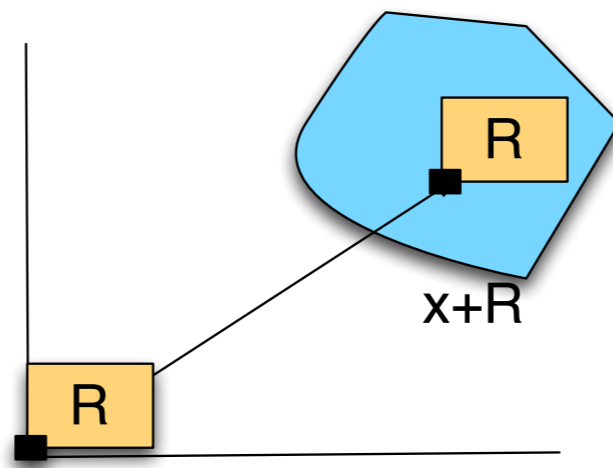
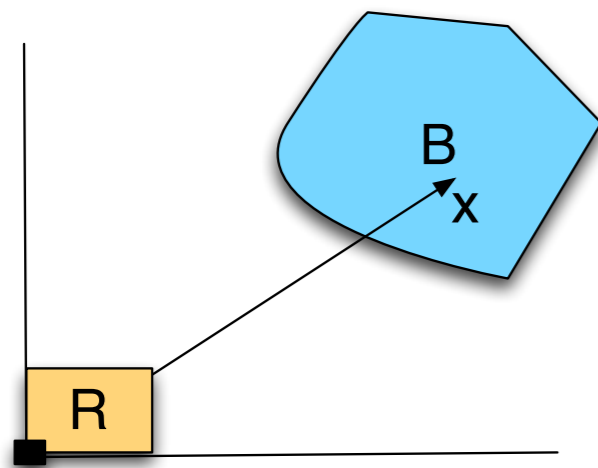
A translated by y



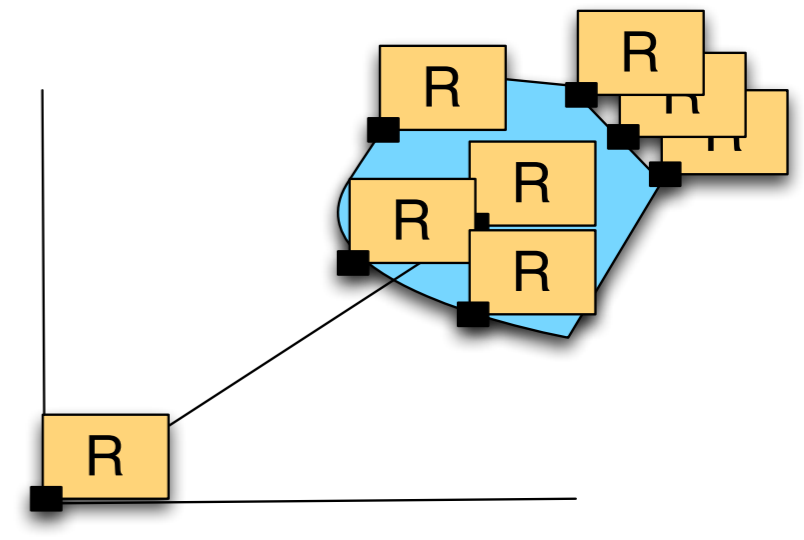
$B \oplus A$

C-obstacles as Minkowski sums

- Consider a robot R with the center in the lower left corner



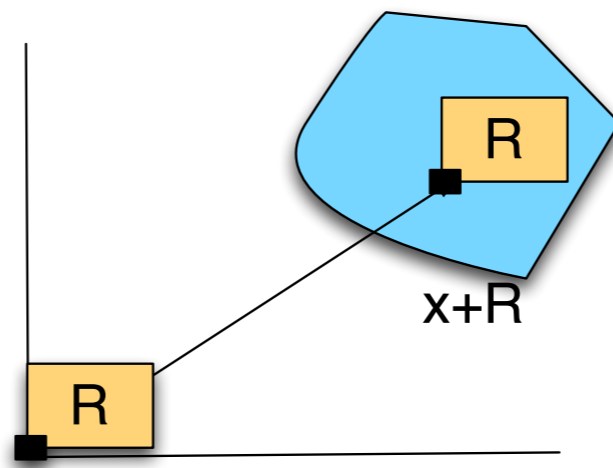
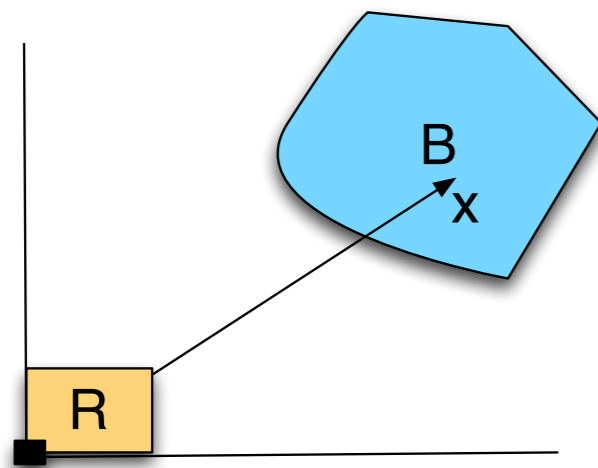
R translated by x



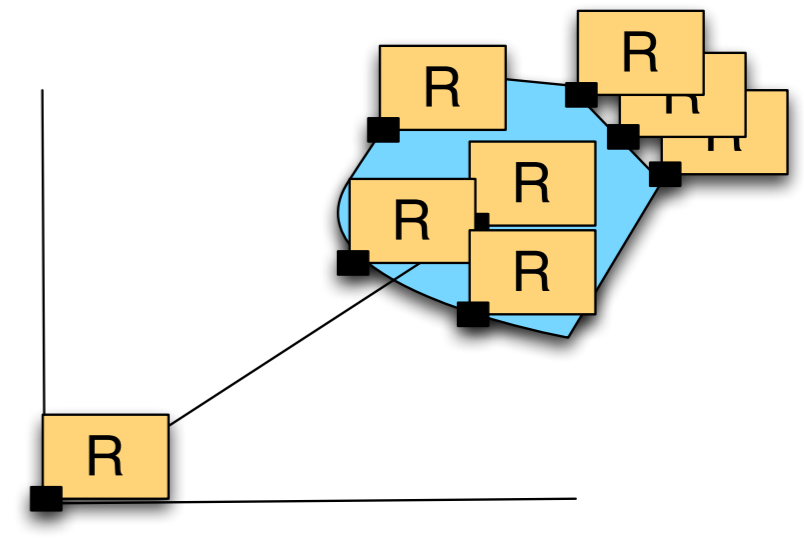
$B \oplus R$

C-obstacles as Minkowski sums

- Consider a robot R with the center in the lower left corner



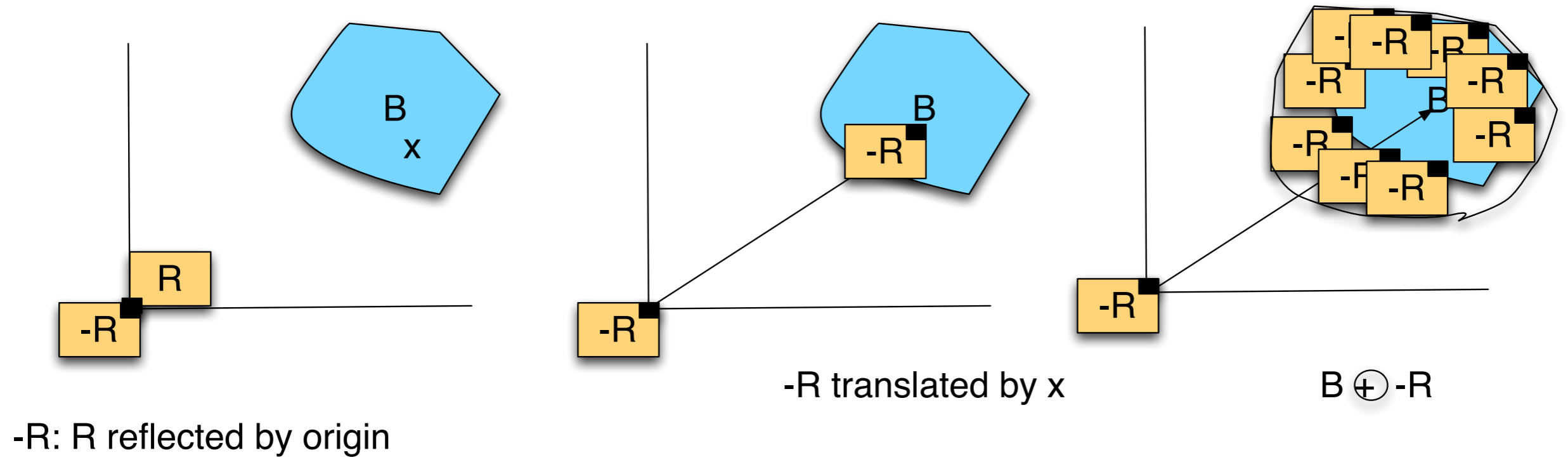
R translated by x



$B \oplus R$

$B \oplus R$ is not quite the C-obstacle of B

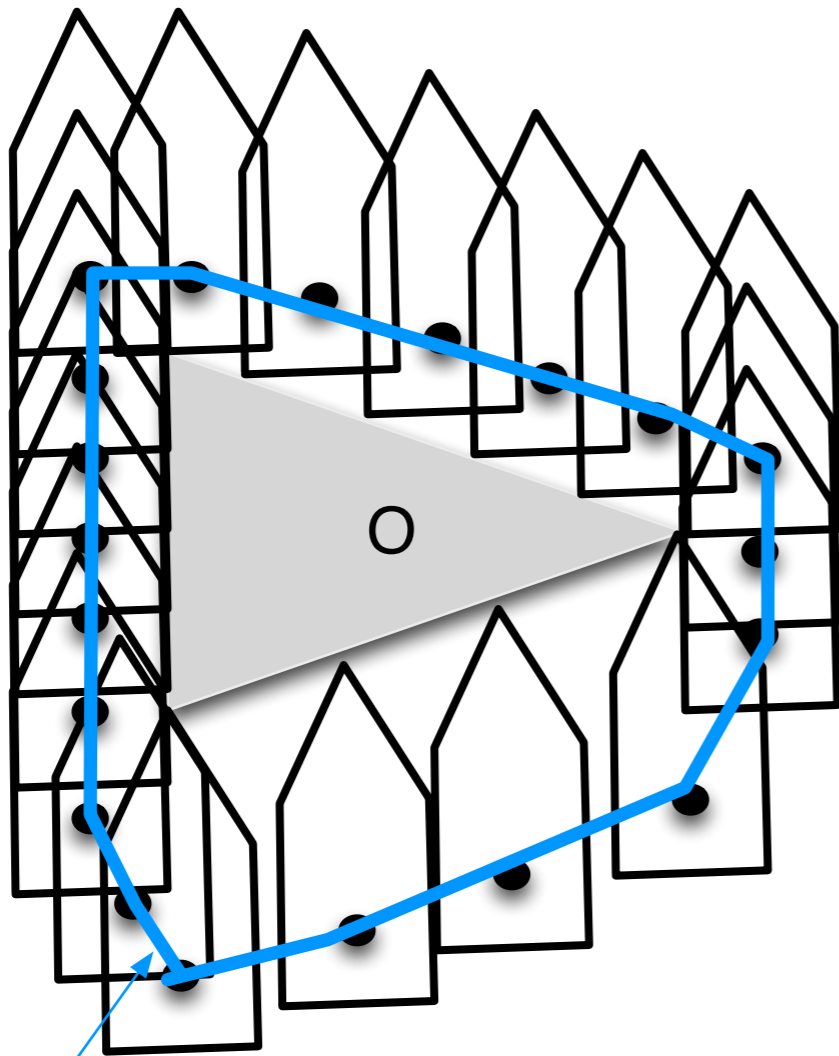
C-obstacles as Minkowski sums



The C-obstacle of B is $B \oplus (-R(0,0))$.

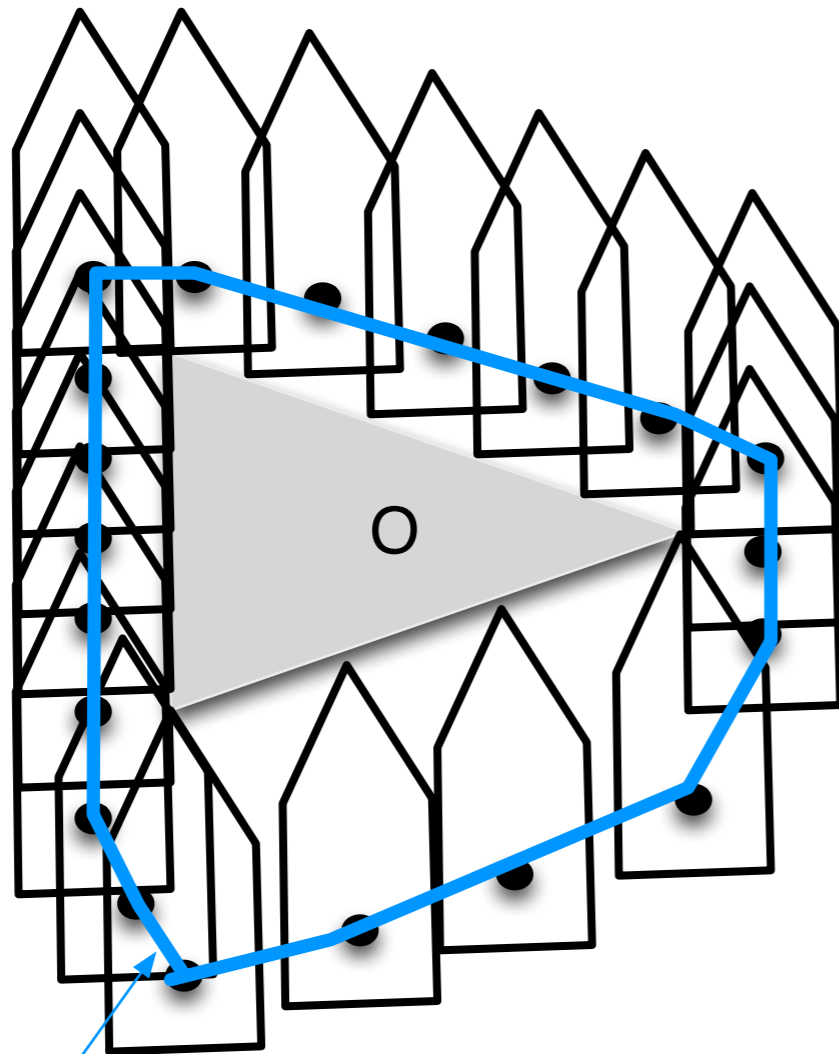
Slide so that R touches the obstacle

Find $O + (-R)$



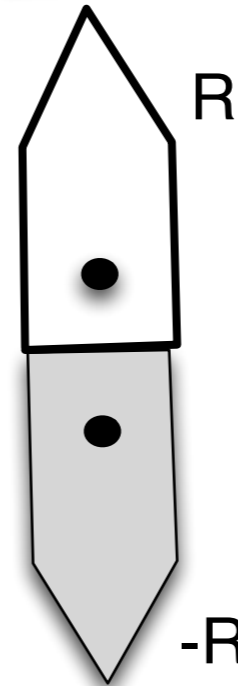
C-obstacle corresponding to O

Slide so that R touches the obstacle

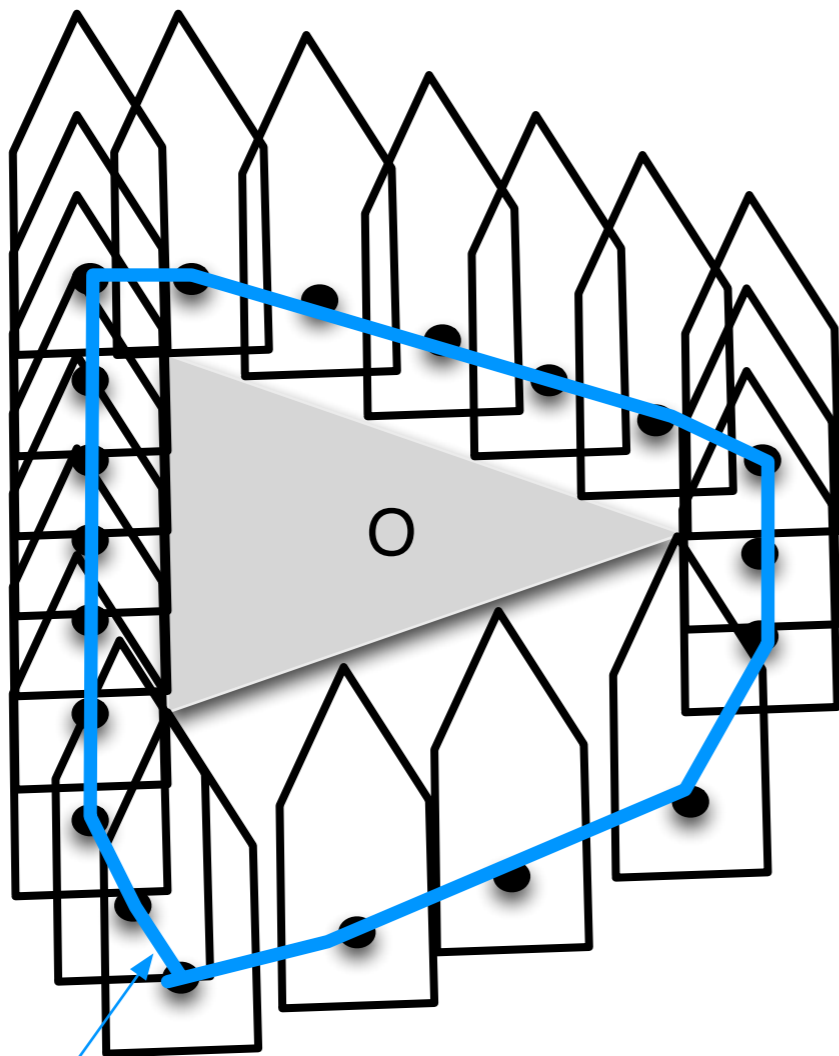


C-obstacle corresponding to O

Slide so that centerpoint of $-R$ traces the edges of obstacle

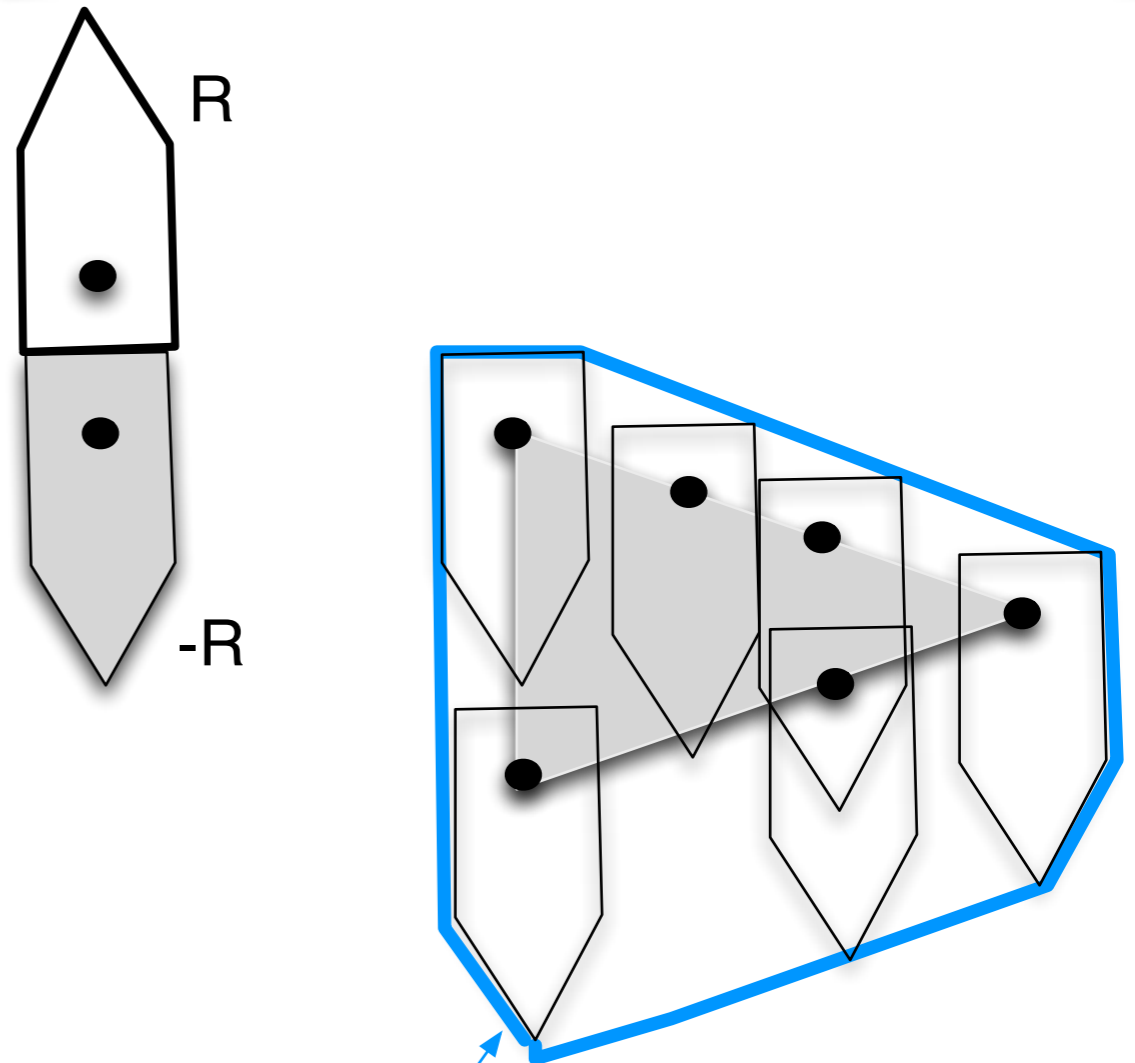


Slide so that R touches the obstacle



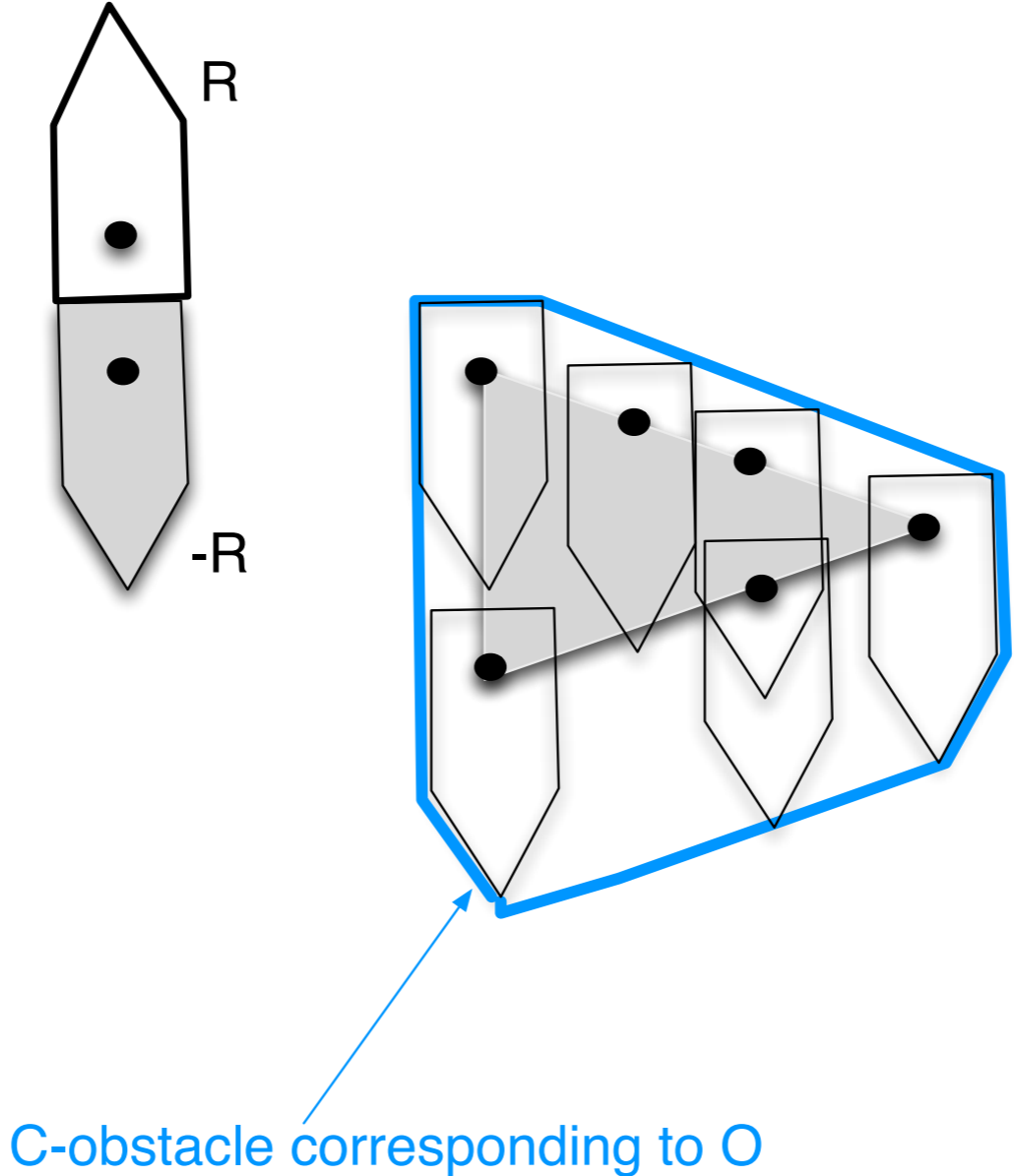
C-obstacle corresponding to O

Slide so that centerpoint of $-R$ traces the edges of obstacle

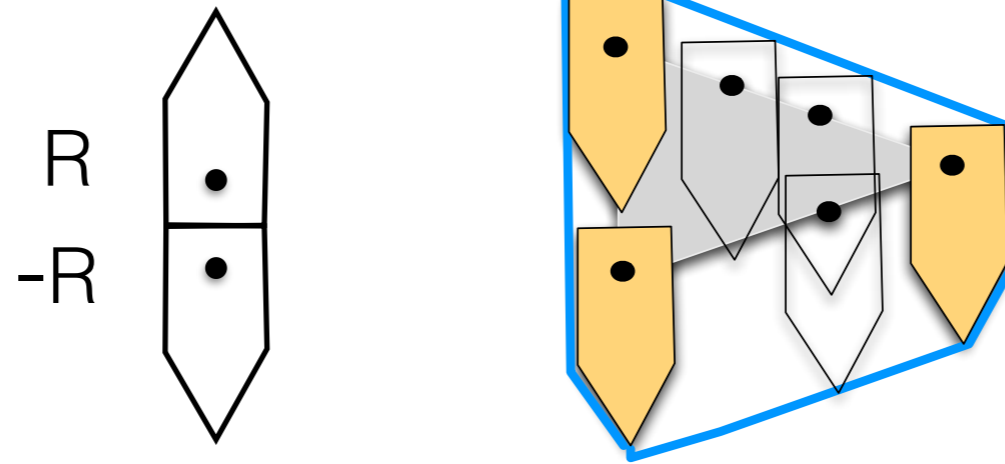


C-obstacle corresponding to O

How do we compute Minkowski sums?



Computing Minkowski sums



CASE 1: Convex robot with convex polygon

Observations:

- Each edge in R , O will cause an edge in $R+O$
- $R+O$ has $m+n$ edges (unless there are parallel edges)
- To compute: Place $-R$ at all vertices of O and compute convex hull
- Possible to compute in $O(m+n)$ time by walking along the boundaries of R and O

Computing Minkowski sums

2D

- **convex + convex polygons**
 - The Minkowski sum of two convex polygons with n , and m edges respectively, is a convex polygon with $n+m$ edges and can be computed in $O(n+m)$ time.
- **convex + non-convex polygons**
 - Triangulate and compute Minkowski sums for each pair [convex polygon, triangle], and take their union
 - Size of Minkowski sum: $O(m+3)$ for each triangle $\Rightarrow O(mn)$
- **non-convex + non-convex polygons:**
 - size of Minkowski sum: $O(n^2m^2)$

3D

- it gets worse . . .

Polygonal robot translating in 2D

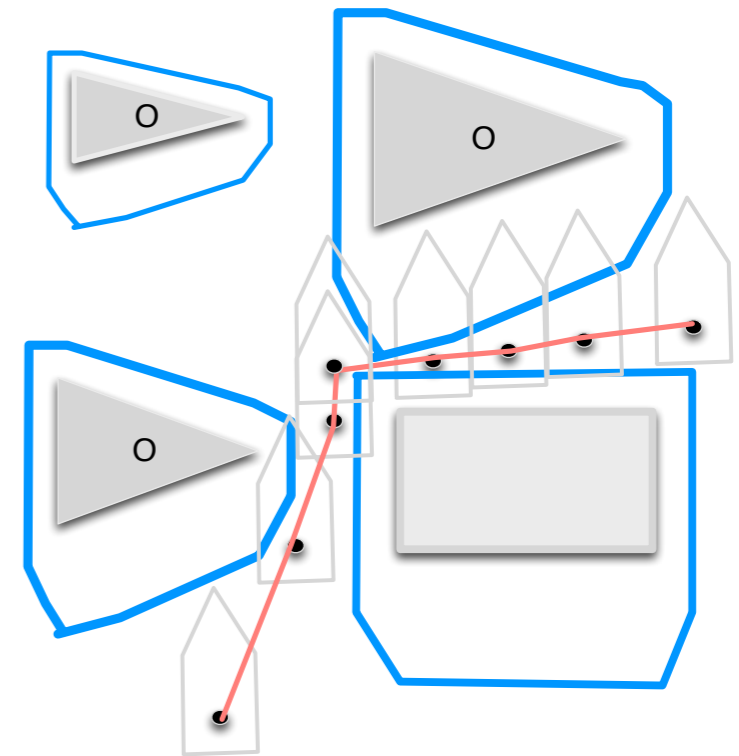
Algorithm

- 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



For a **convex** robot of **$O(1)$ size**

- Free C-space can be computed in $O(n \lg^2 n)$ time.
==> With $O(n \lg^2 n)$ time pre-processing, a collision-free path can be found for any start and end in $O(n)$ time.

Complete, non optimal.

Polygonal robot in 2D with rotations

- Physical space is 2D
- A placement is specified by 3 parameters: $R(x, y, \theta) \implies$ C-space is 3D.



Polygonal robot in 2D with rotations

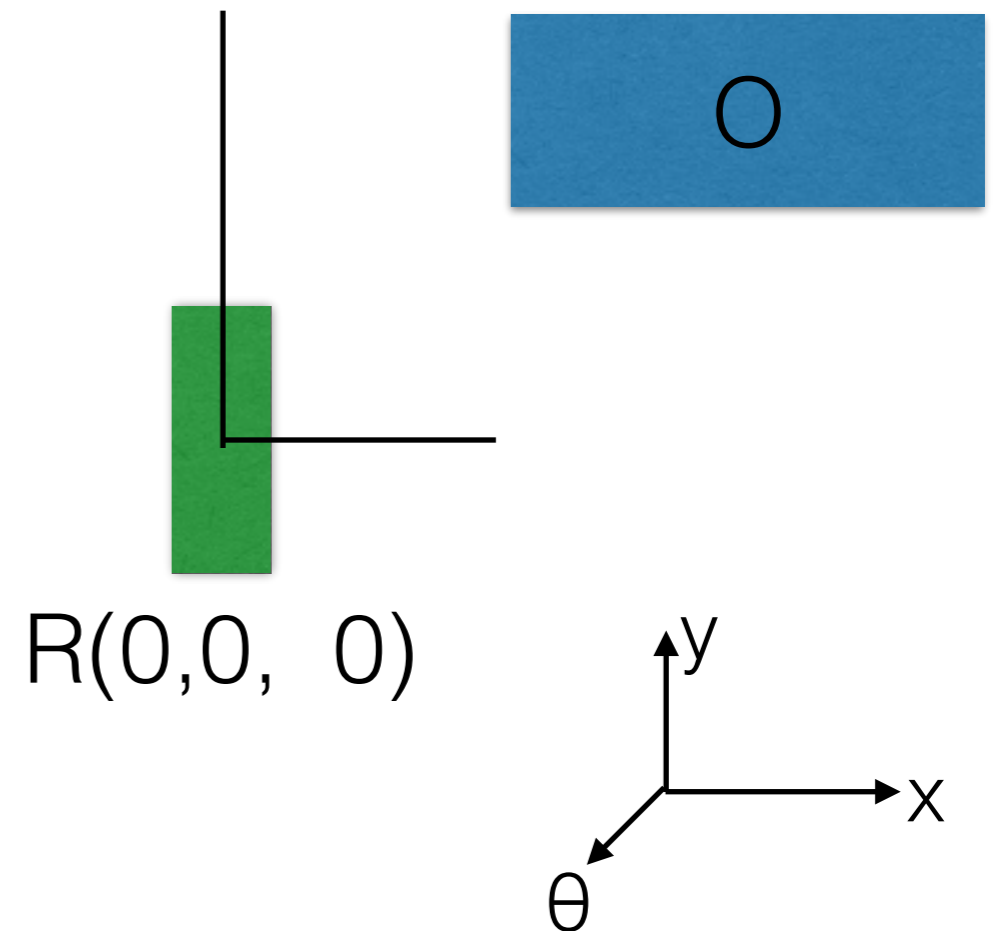
- We'd like to extend the same approach:

Reduce to point robot moving among C-obstacles in C-space.

- Compute C-obstacles
- Compute free space as complement of union of C-obstacles
- Decompose free space into simple cells
- Construct a roadmap
- BFS on roadmap

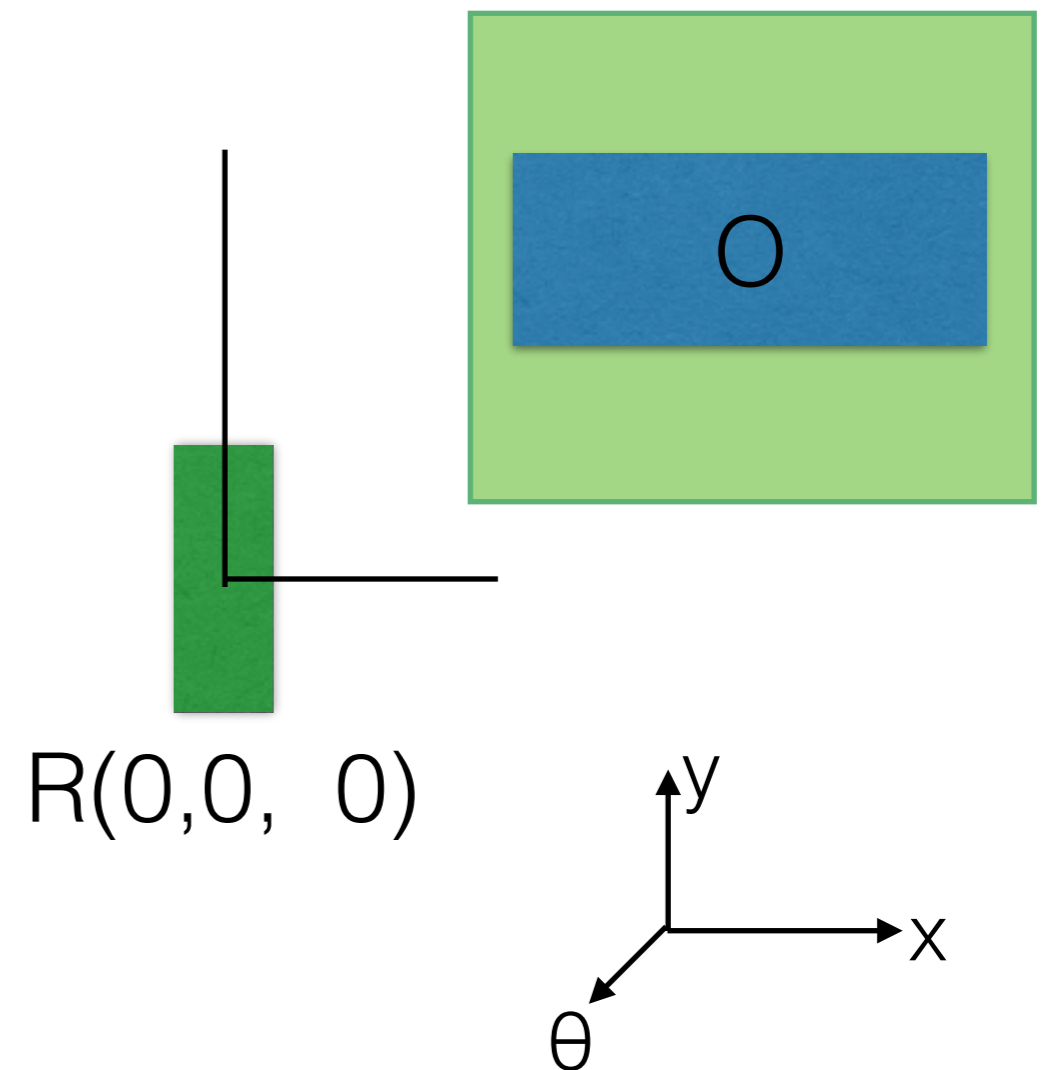
Polygonal robot in 2D with rotations

- What does a C-obstacle look like when rotations are allowed?



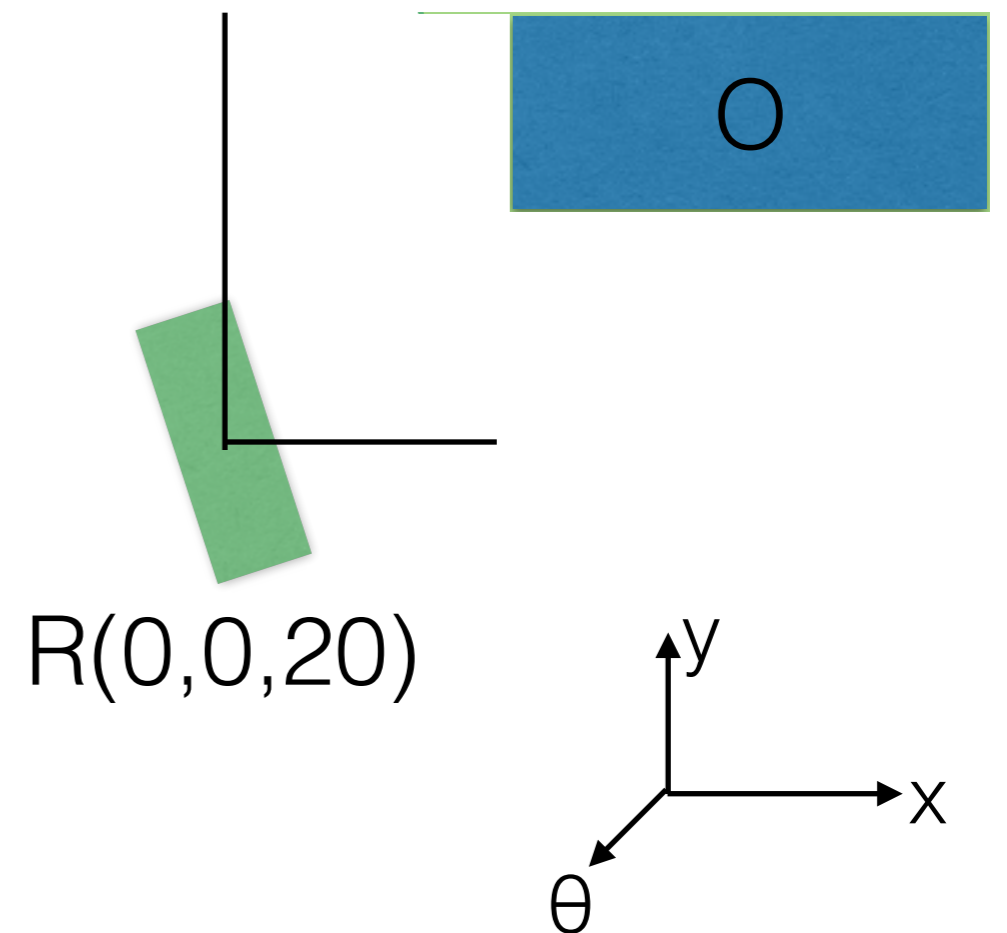
Polygonal robot in 2D with rotations

- What does a C-obstacle look like when rotations are allowed?



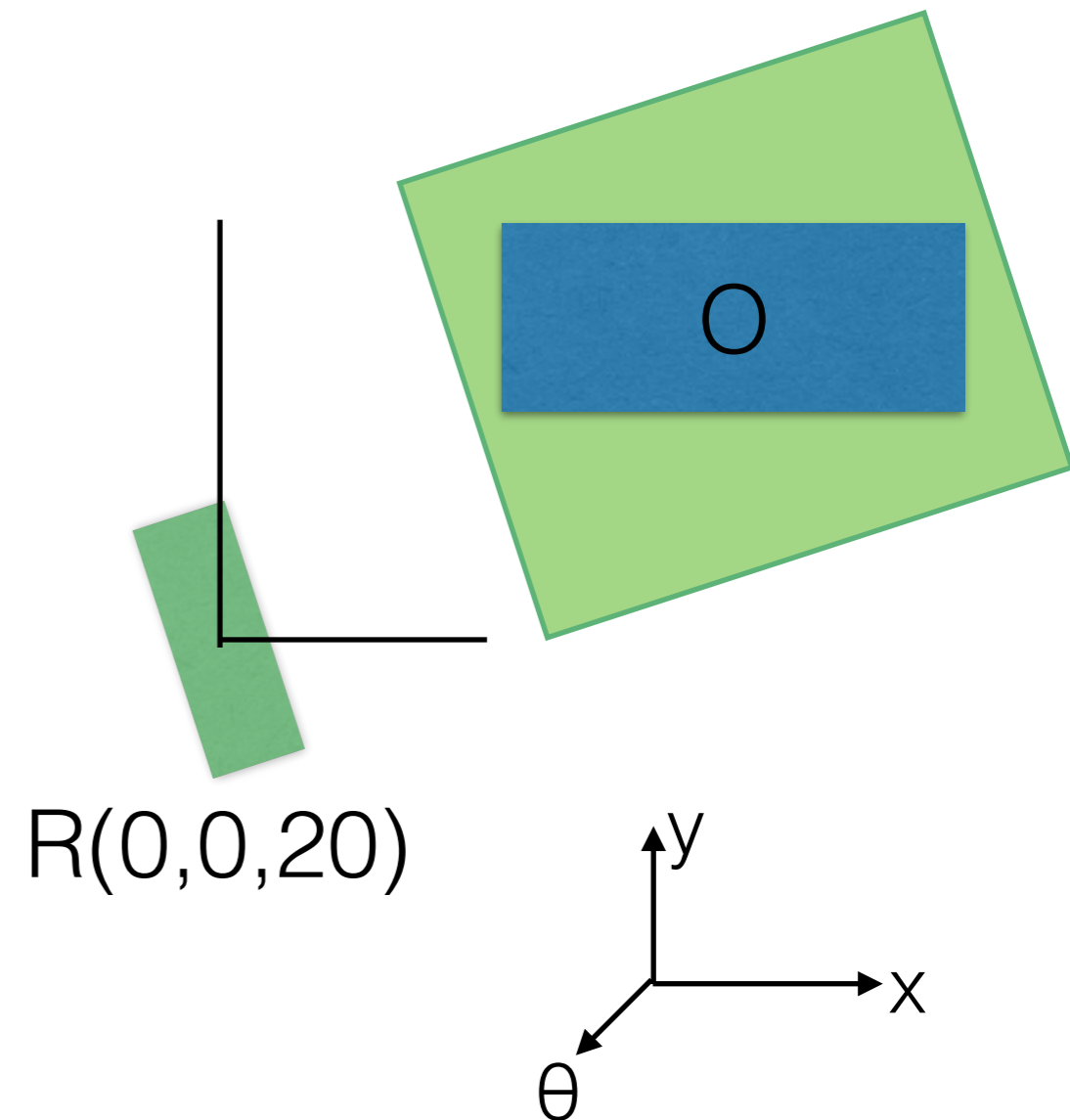
Polygonal robot in 2D with rotations

- What does a C-obstacle look like when rotations are allowed?



Polygonal robot in 2D with rotations

- What does a C-obstacle look like when rotations are allowed?



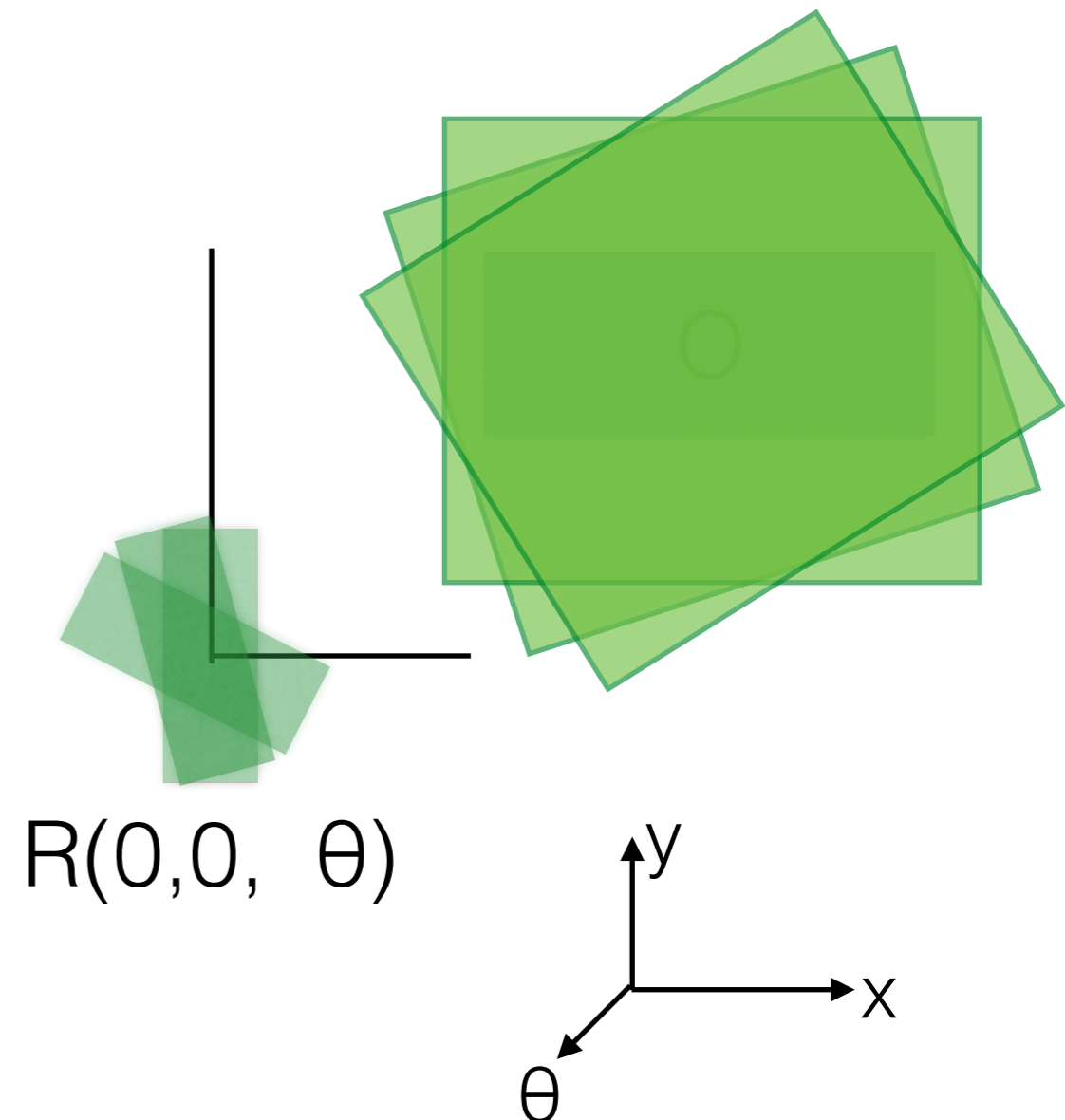
Polygonal robot in 2D with rotations

A C-obstacle is a 3D shape.

Imagine moving a horizontal plane vertically through C-space.

Each cross-section of the C-obstacle is a Minkowski sum $O \oplus -R(0,0,\theta)$

=> twisted pillar



Polygonal robot in 2D with rotations

What's known:

- C-space is 3D
- Boundary of free space is curved, not polygonal.
- Combinatorial complexity of free space is $O(n^2)$ for convex, $O(n^3)$ for non-convex robot

Polygonal robot in 2D with rotations

What's known:

- C-space is 3D
 - Boundary of free space is curved, not polygonal.
 - Combinatorial complexity of free space is $O(n^2)$ for convex, $O(n^3)$ for non-convex robot
-
- Extend same approach:

1. Compute C-obstacles and C-free
2. Compute a decomposition of free space into simple cells
3. Construct a roadmap
4. BFS on roadmap

space is 3D



Difficult to construct a good cell decomposition for curved 3D space



Polygonal robot in 2D with rotations

- Difficult to construct a good cell decomposition for curved 3D space
- A possible approach:
 - If angle is fixed: you got translational motion planning
 - Discretize rotation angle and compute a finite number of slices, one for each angle
 - Construct a trapezoidal decomposition for each slice and its roadmap
 - Link them into a 3D roadmap
 - Add “vertical” edges between slices to allow robot to move up/down between slices; these correspond to rotational moves.
- Example: Consider two angles a and b . If placement (x,y) is in free space in slice a , and (x,y) is in free space in slice b , then the 3D roadmap should contain a vertical edge between slice a and b at that position

Is this complete ?

Combinatorial path planning: Summary

- **Idea: Compute free C-space combinatorially (= exact)**
- **Approach**
 - Reduce (robot, obstacles) => (point robot, C-obstacles)
 - Compute roadmap of free C-space
 - any path: trapezoidal decomposition or triangulation
 - shortest path: visibility graph
- **Comments**
 - Complete
 - Works beautifully in 2D and for some cases in 3D
 - Worst-case bound for combinatorial complexity of C-objects in 3D is high
 - Unfeasible/intractable for high #DOF
 - A complete planner in 3D runs in $O(2^{n^{\#DOF}})$