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_{start}
- end position p_{end}

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

- Ideally we would like a planner that's 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

Applications

Motion Planning

Combinatorial motion planning

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

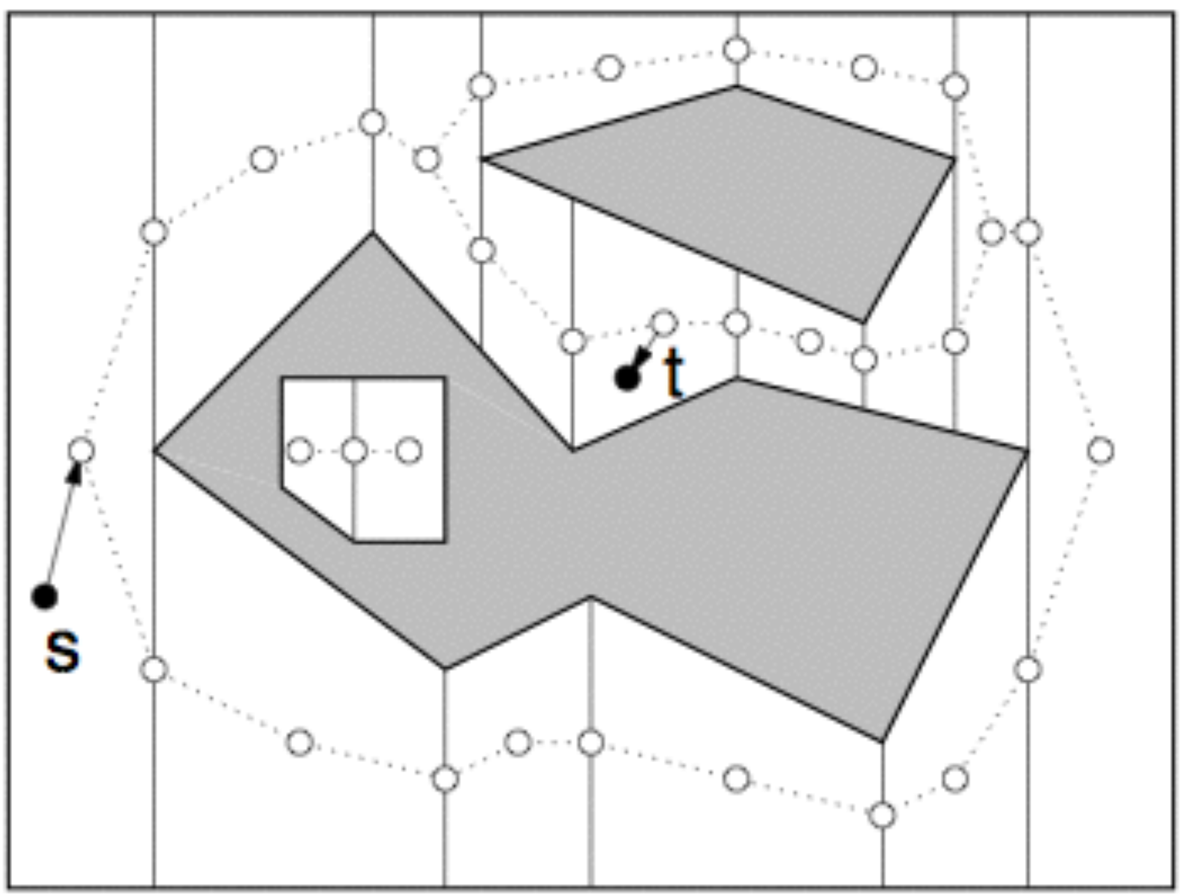
- Polygon robot in 2D
    - Translation only
    - Handling rotation

today

Approximate motion planning

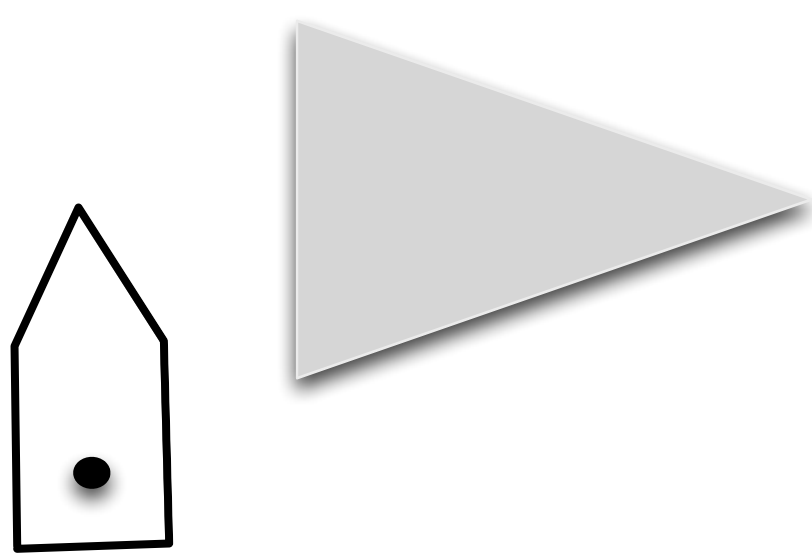
Point robot in 2D

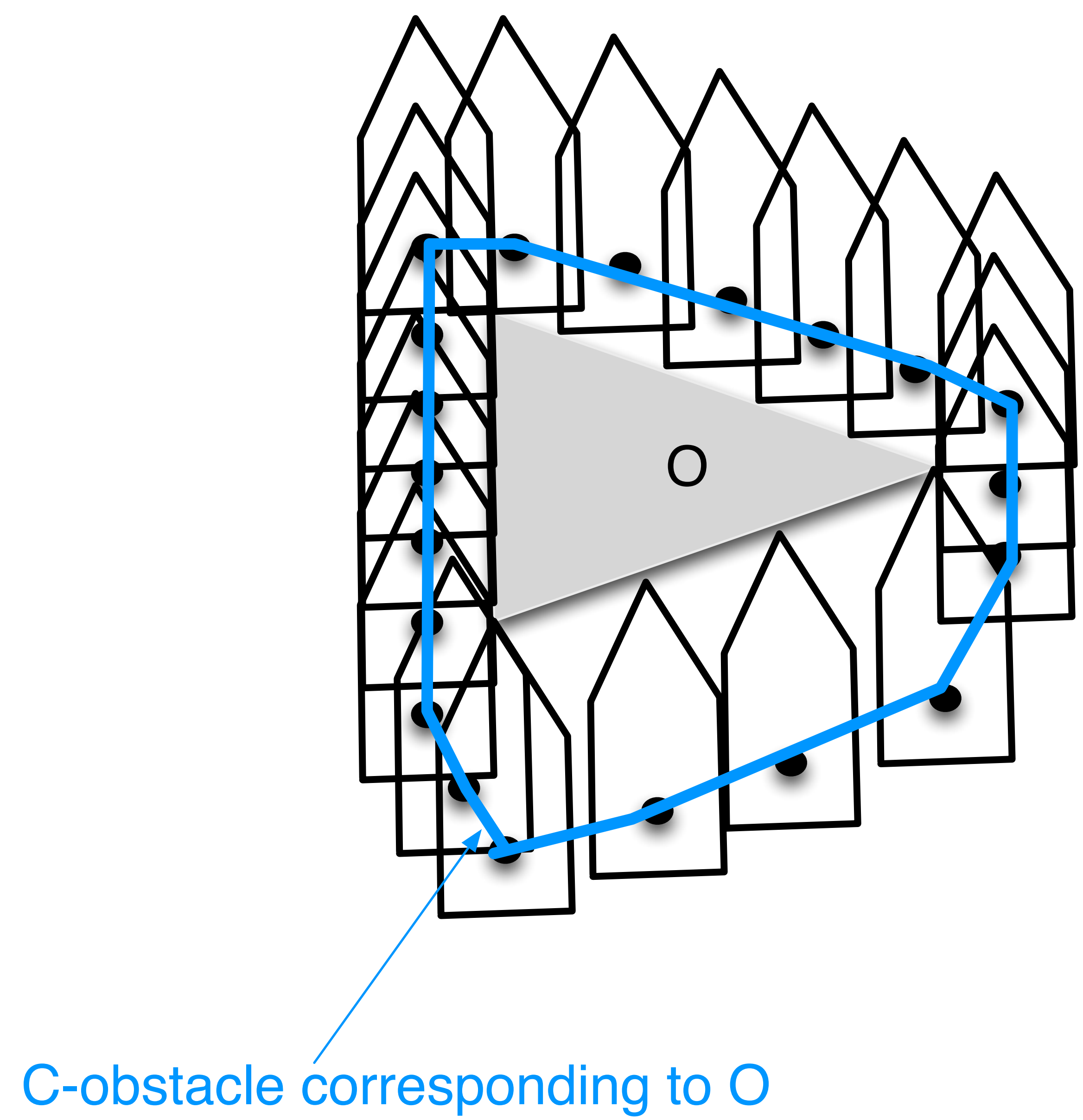
- General idea
    - Compute a trapezoid decomposition of free space
    - Build a graph (roadmap) of free space
    - Search graph to find path <---------- Reduce motion planning to graph search



Polygonal robot in 2D

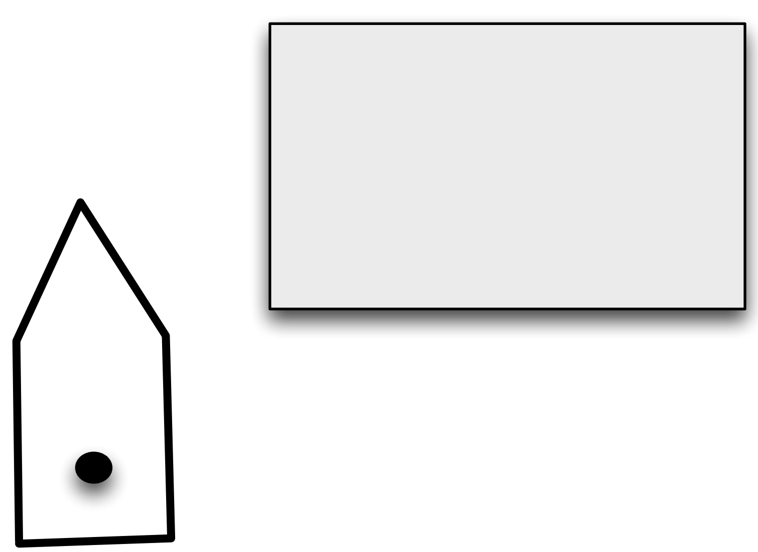
- Robot R(x,y)
- The C-obstacle corresponding to obstacle O represents the set of all placements that cause intersection with O.

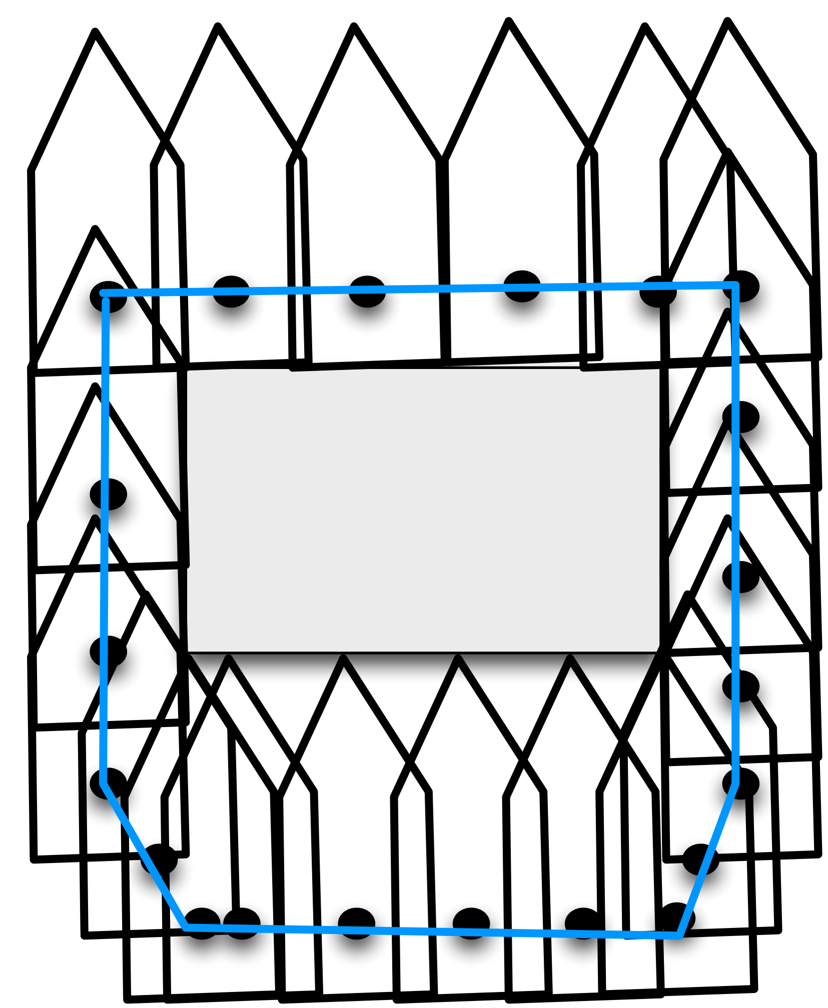




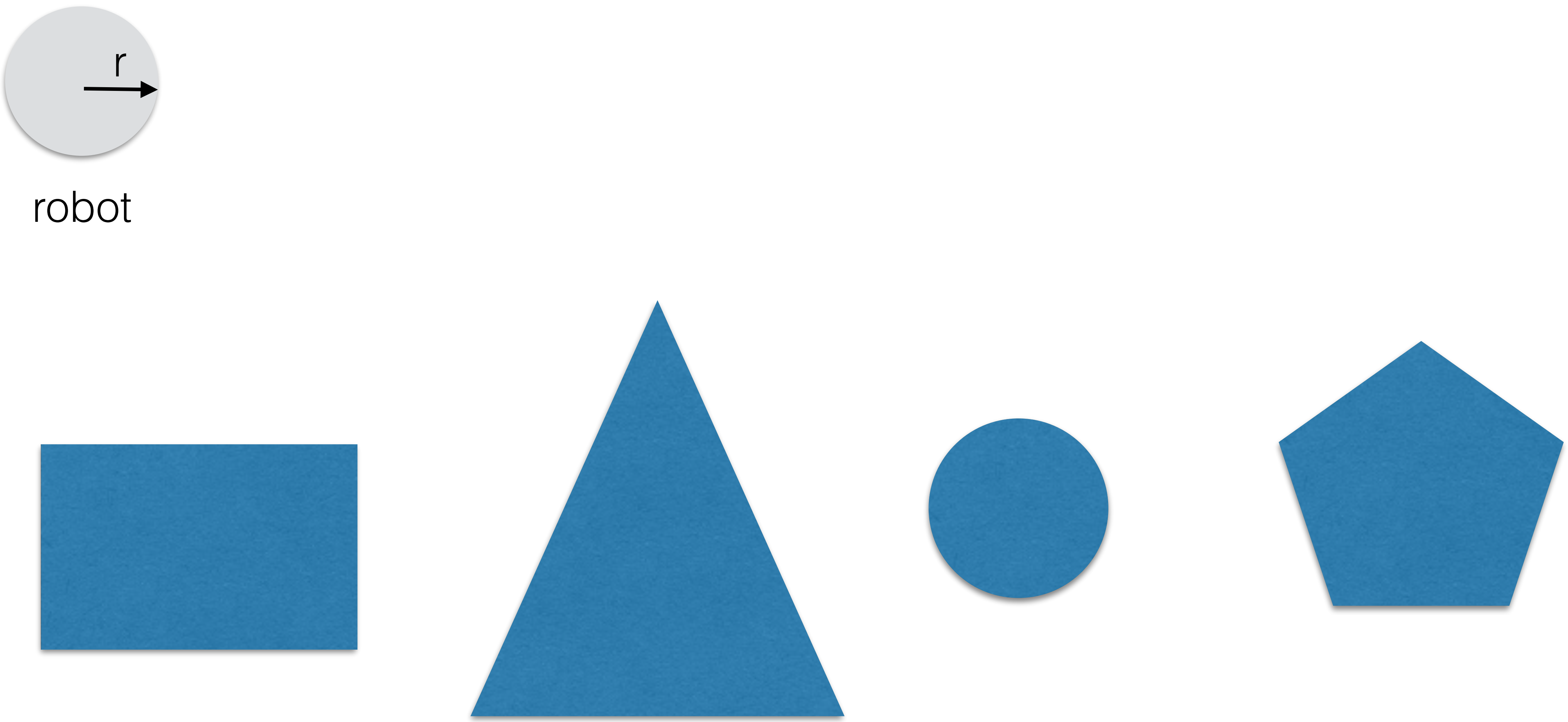
Polygonal robot in 2D

- Robot R(x,y)
- The C-obstacle corresponding to obstacle O represents the set of all placements that cause intersection with O.





Exercise



Show the corresponding C-obstacles for a disc robot.

Exercise

Show the corresponding C-obstacle.

Polygonal robot translating in 2D

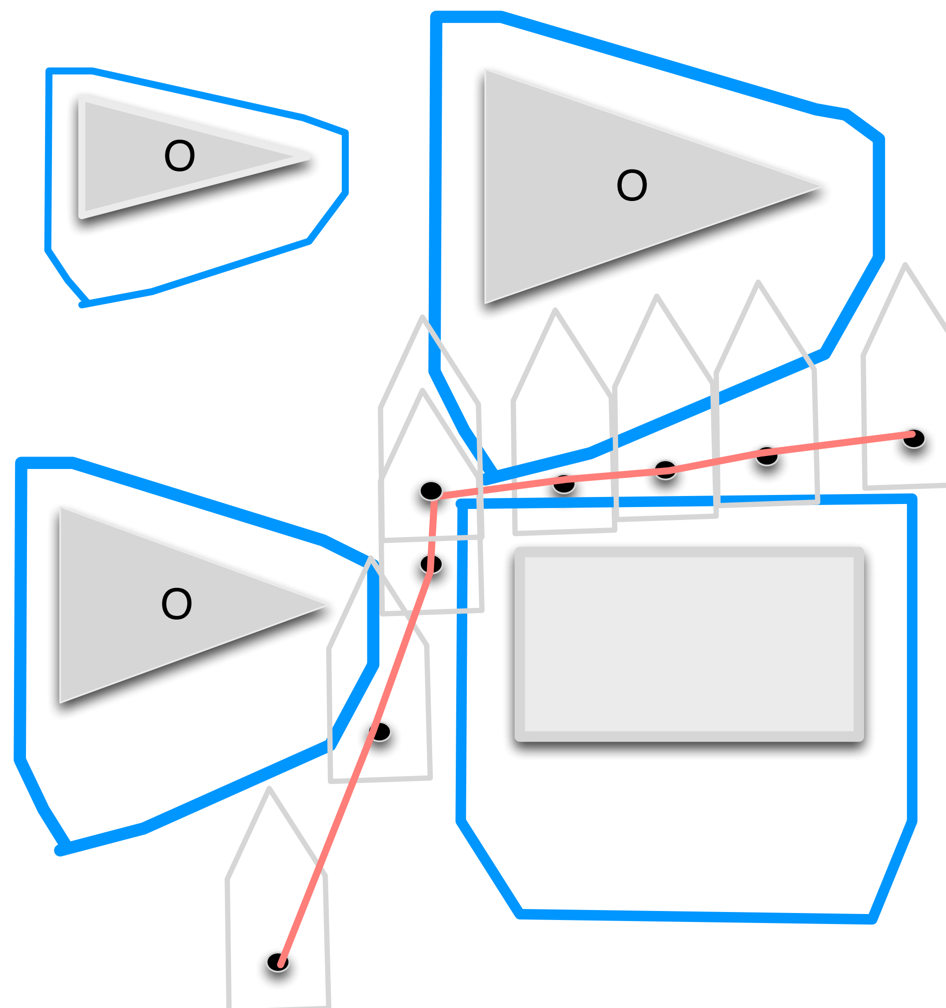
Algorithm

- 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 point

//robot moving in free C-space

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



How fast can we do this?

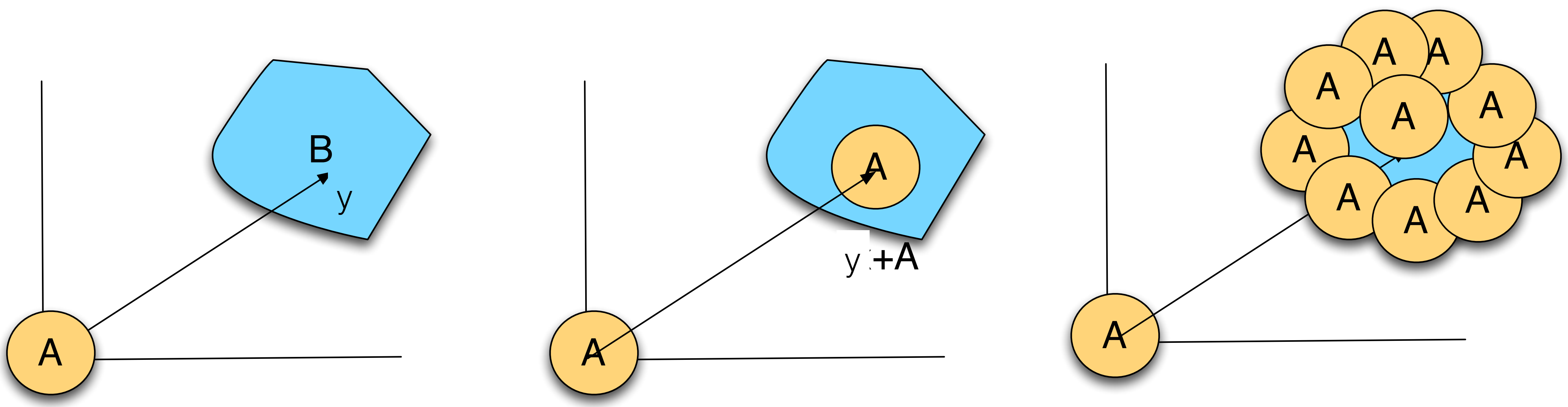
How do we compute C-obstacles?

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

  vector sum   x, y vectors

- Interpretation: consider set A to be centered at the origin. Then A + 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.

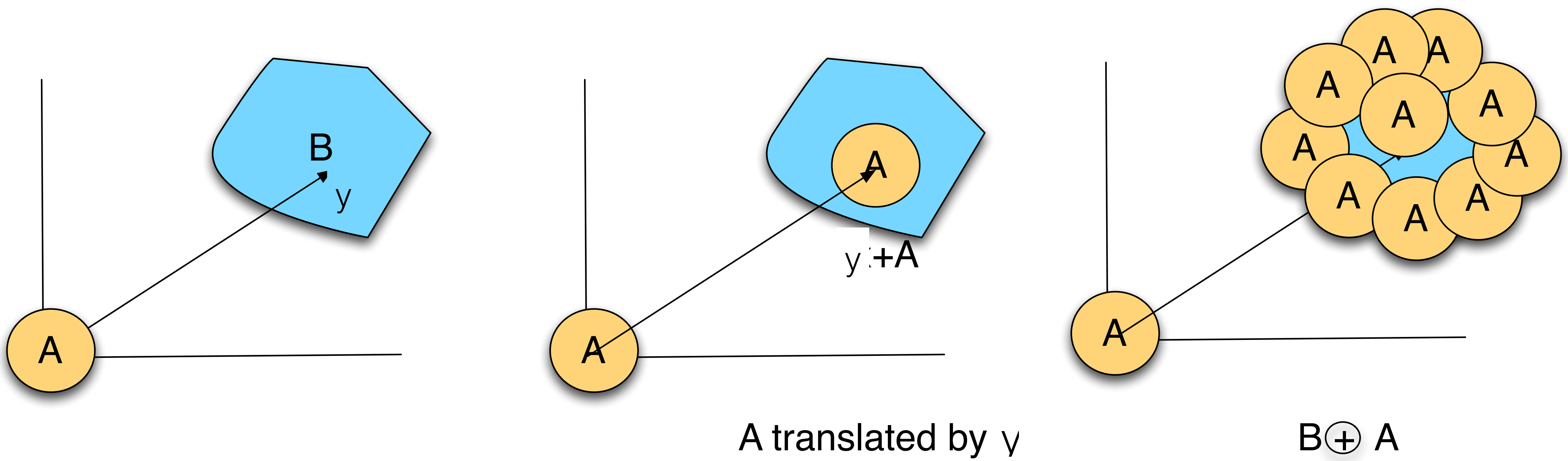


A translated by y

$B \oplus A$

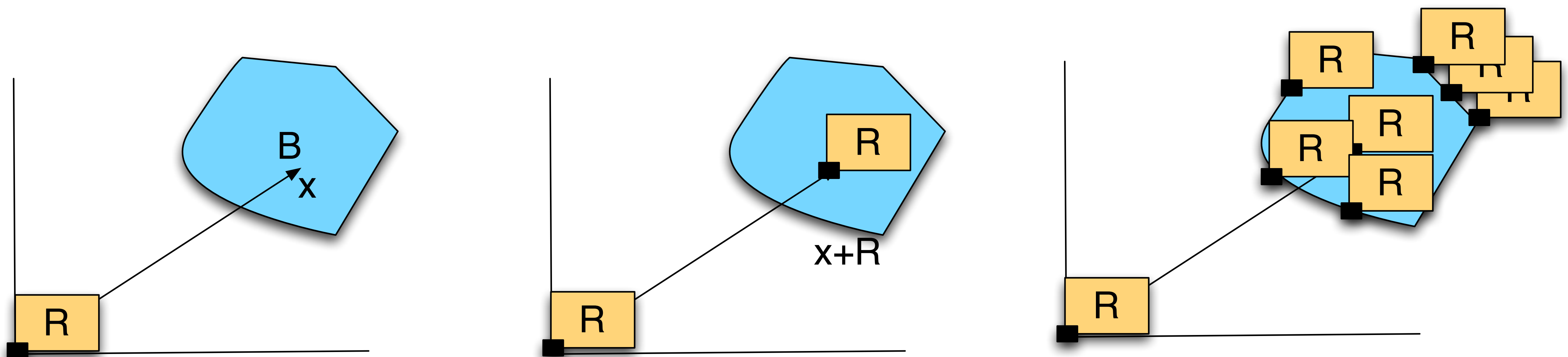
Minkowski sum

- A ⊕ B: Slide A so that the center of A traces the edges of B



C-obstacles as Minkowski sums

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

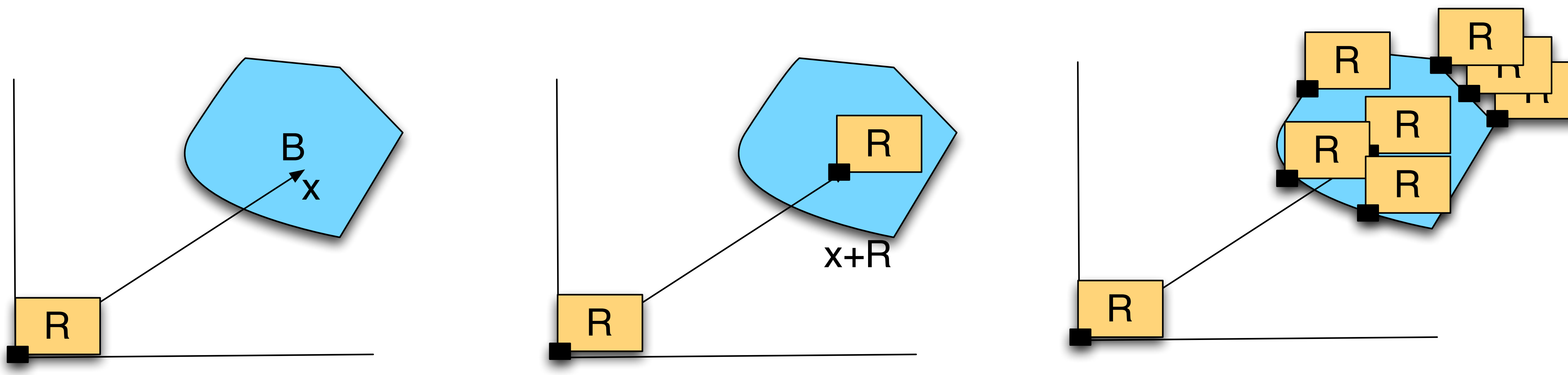


R translated by x

$B \oplus R$

C-obstacles as Minkowski sums

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

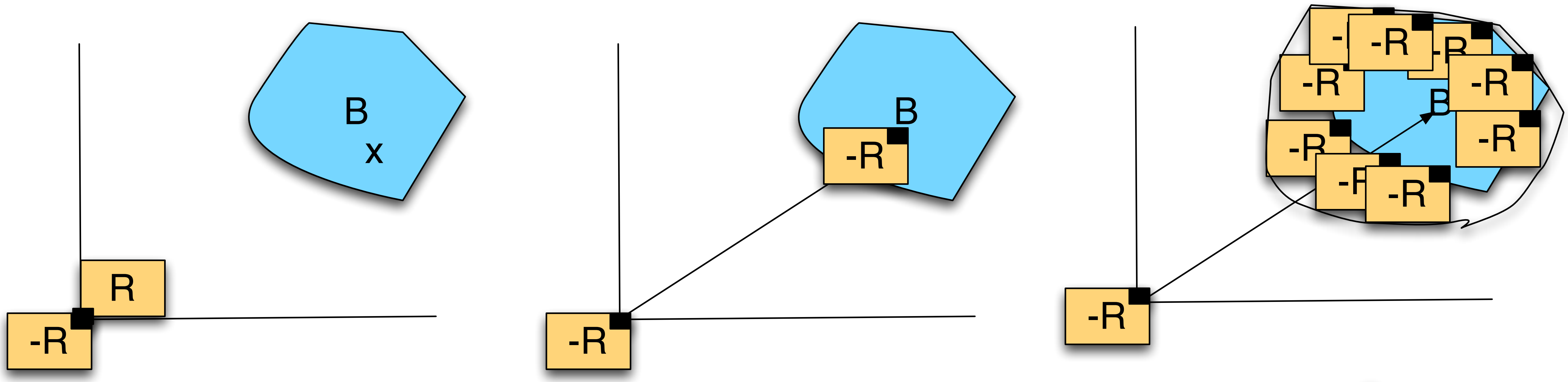


R translated by x

B $\oplus$ R

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

C-obstacles as Minkowski sums



-R: R reflected by origin

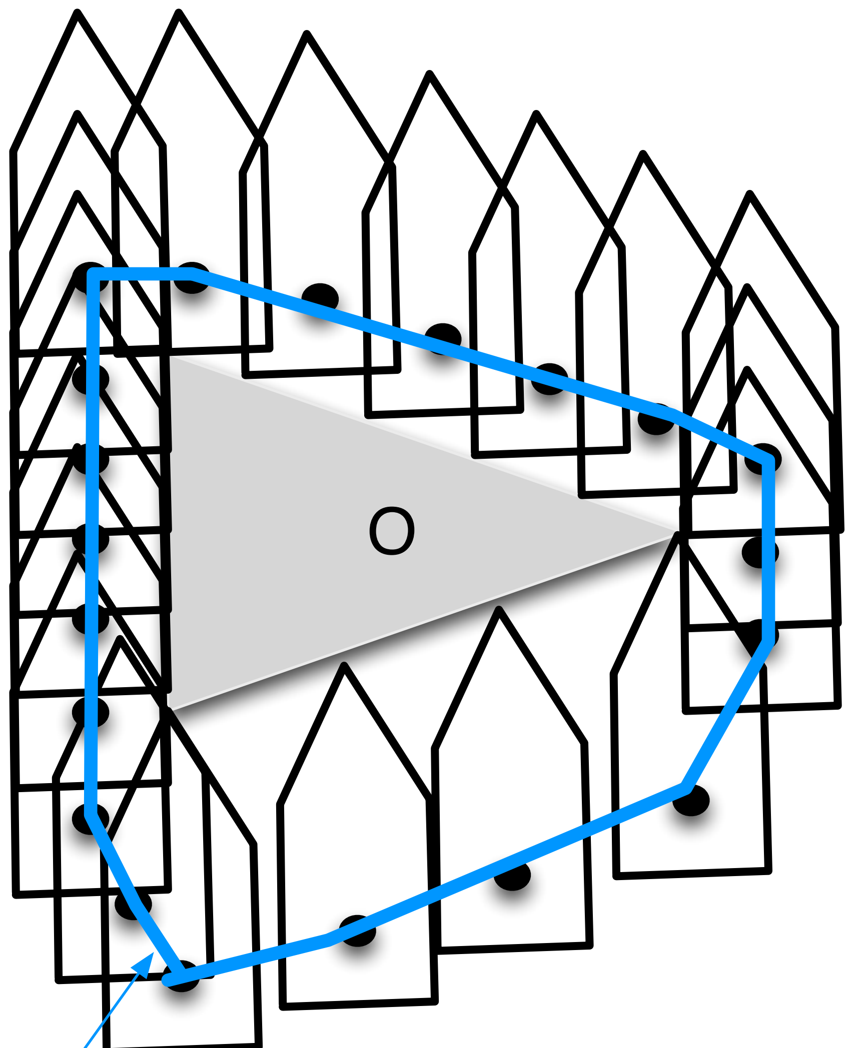
-R translated by x

$B \oplus -R$

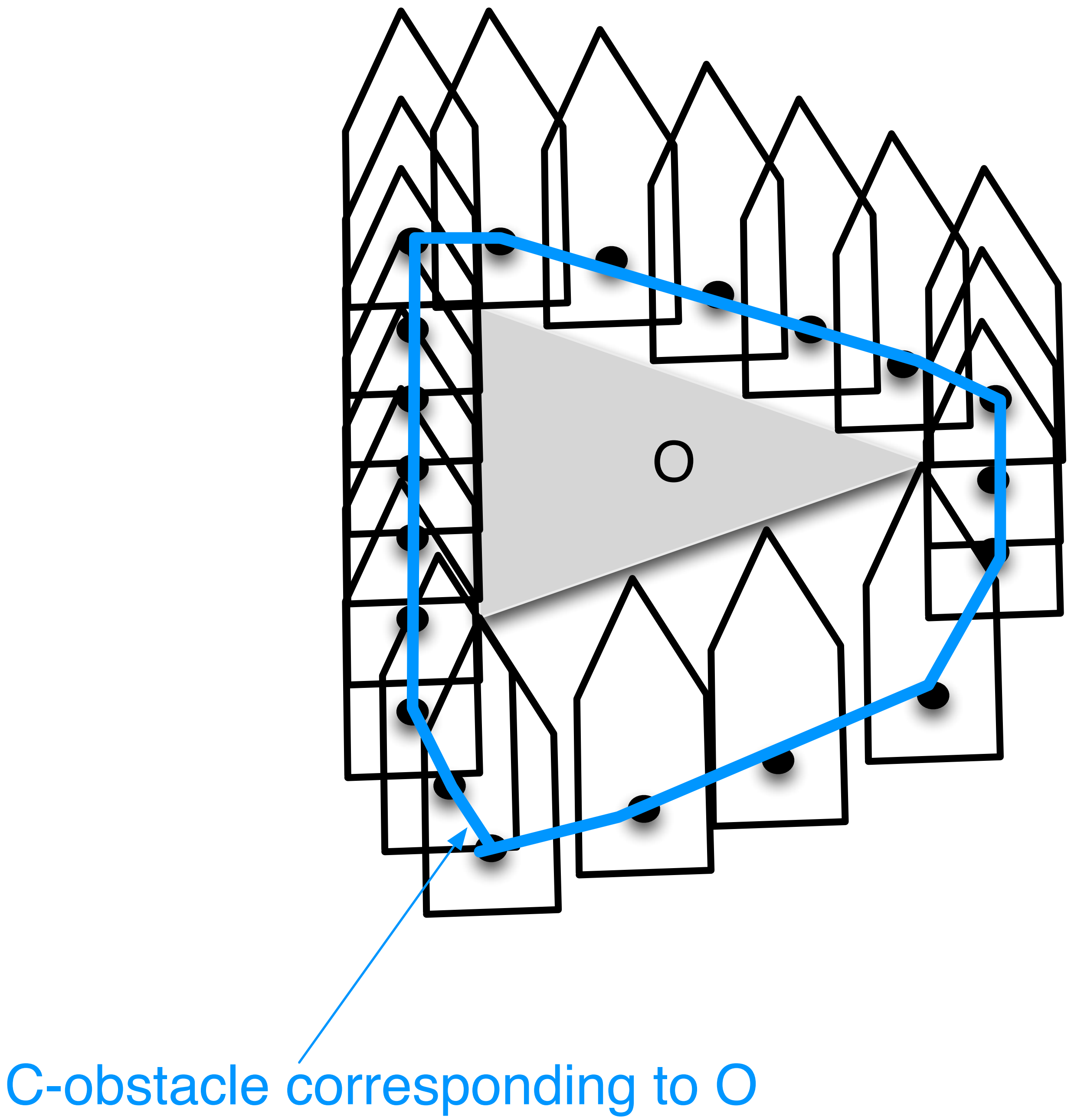
The C-obstacle of B is $B \oplus (-R(0,0))$.

Slide so that R touches the obstacle

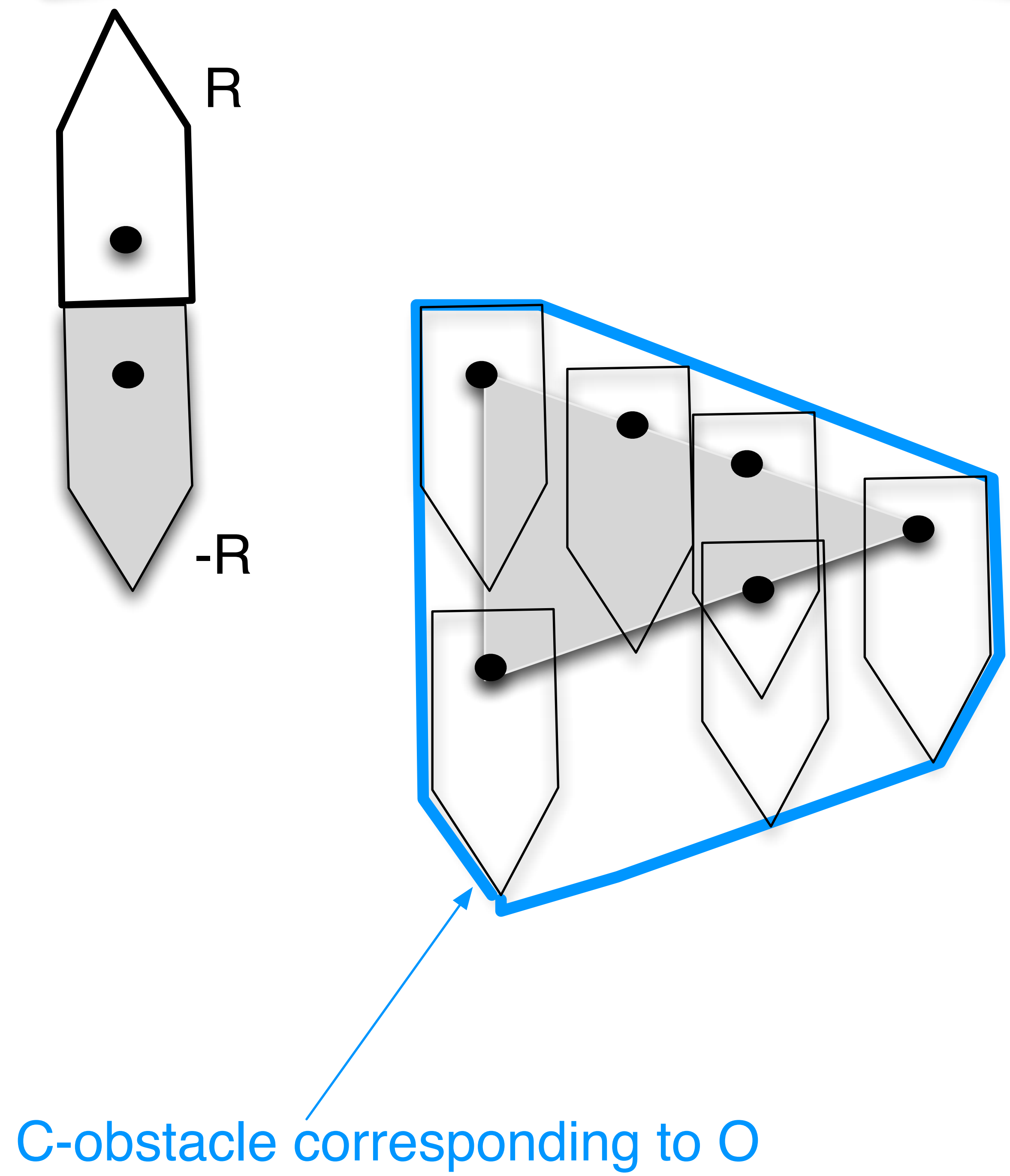
Find O + (-R)



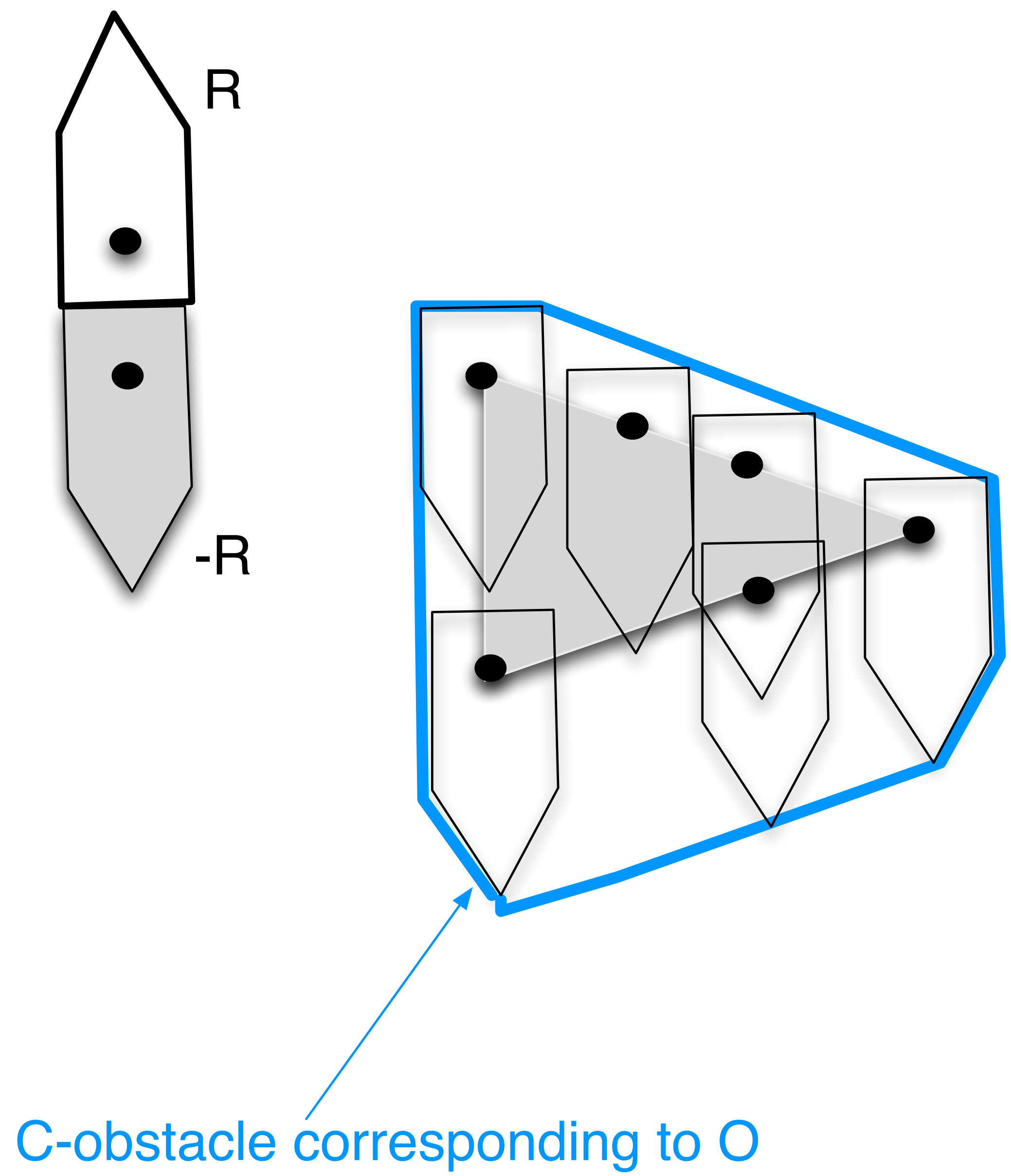
Slide so that R touches the obstacle



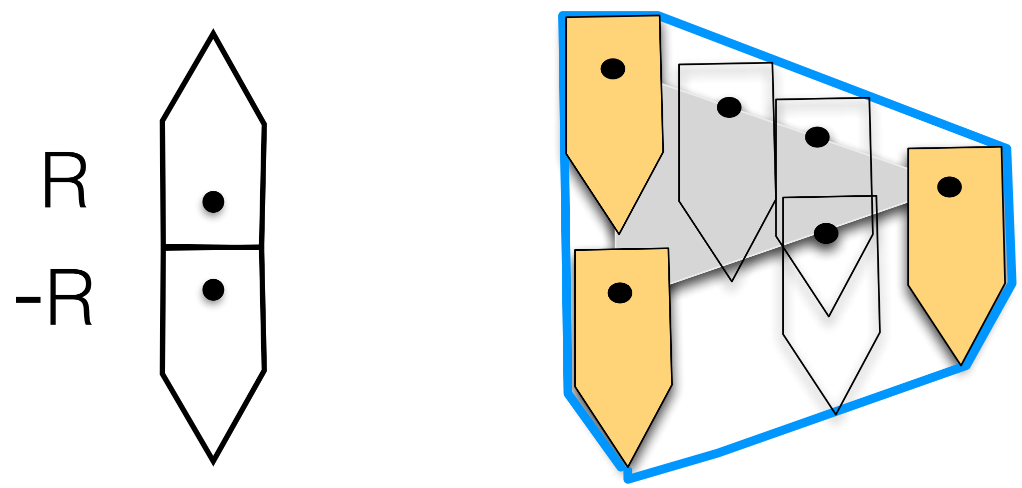
Slide so that centerpoint of -R traces the edges of obstacle



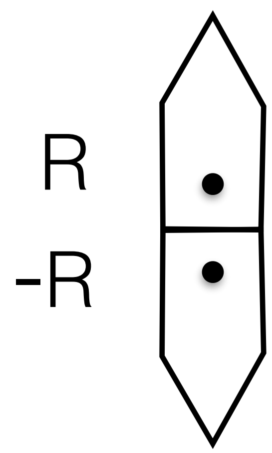
How do we compute Minkowski sums?

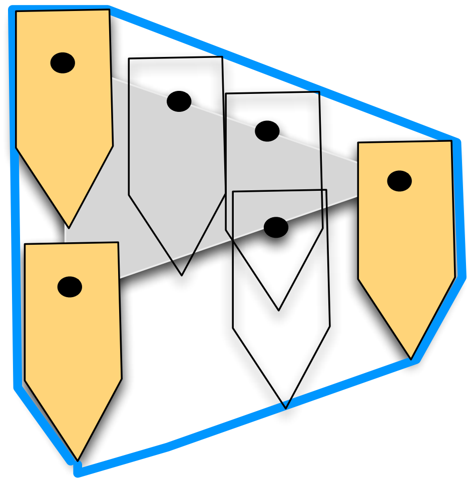


Computing Minkowski sums



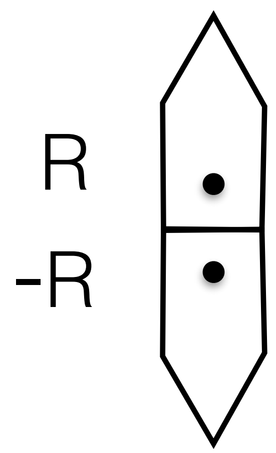
Computing Minkowski sums

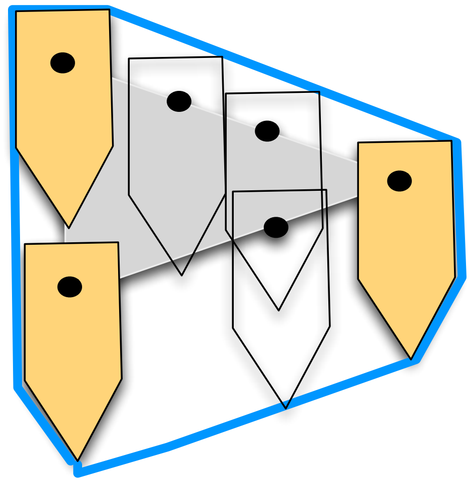




CASE 1: Convex robot with convex polygon

Computing Minkowski sums

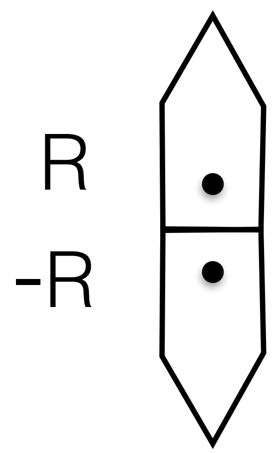


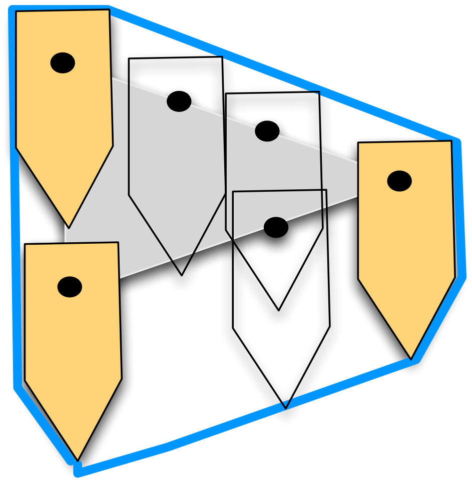


CASE 1: Convex robot with convex polygon

Observations:

Computing Minkowski sums



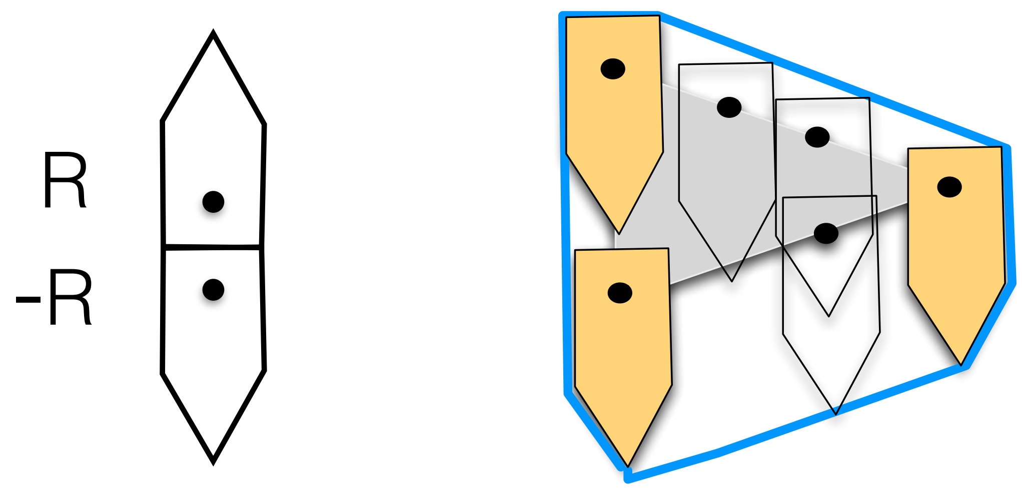


CASE 1: Convex robot with convex polygon

Observations:

- Each edge in R, O will cause an edge in R+O

Computing Minkowski sums

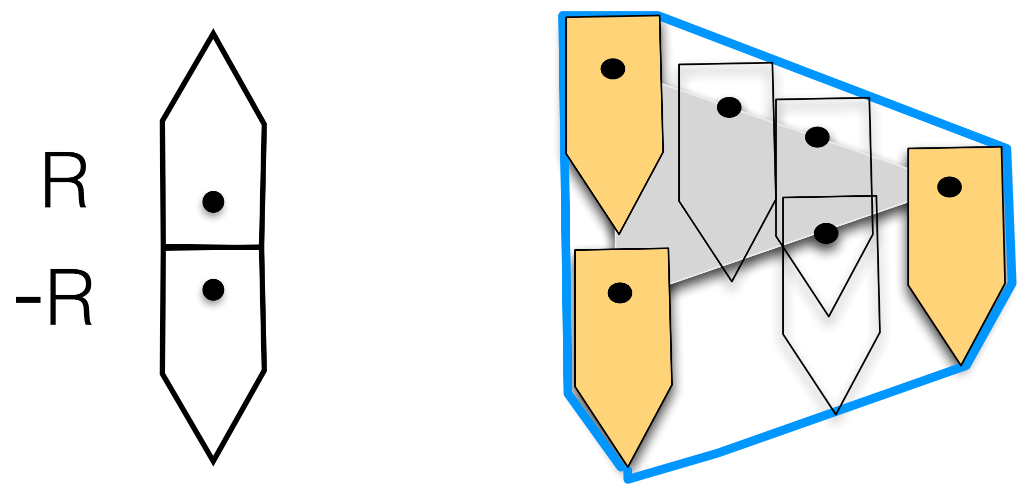


CASE 1: Convex robot with convex polygon

Observations:

- Each edge in R, O will cause an edge in R+O
- R+O has m+n edges unless there are parallel edges

Computing Minkowski sums

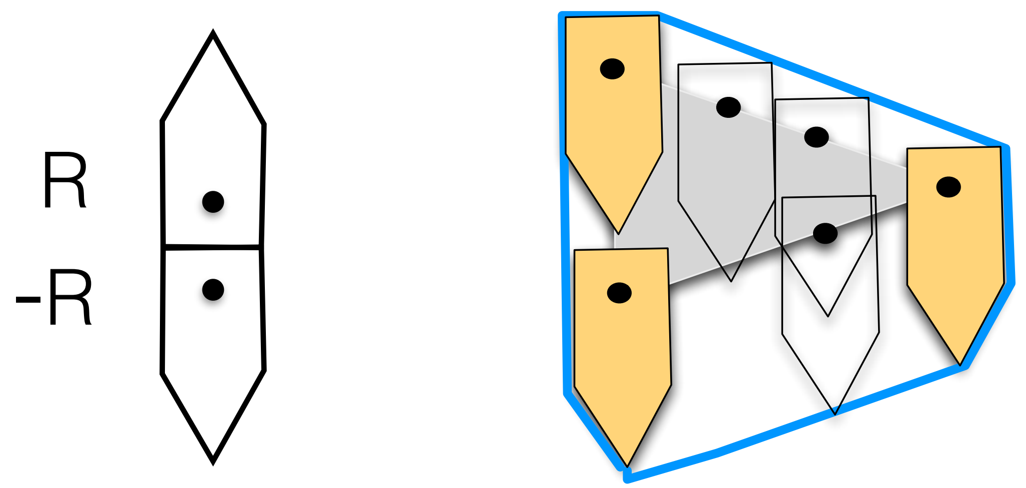


CASE 1: Convex robot with convex polygon

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

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

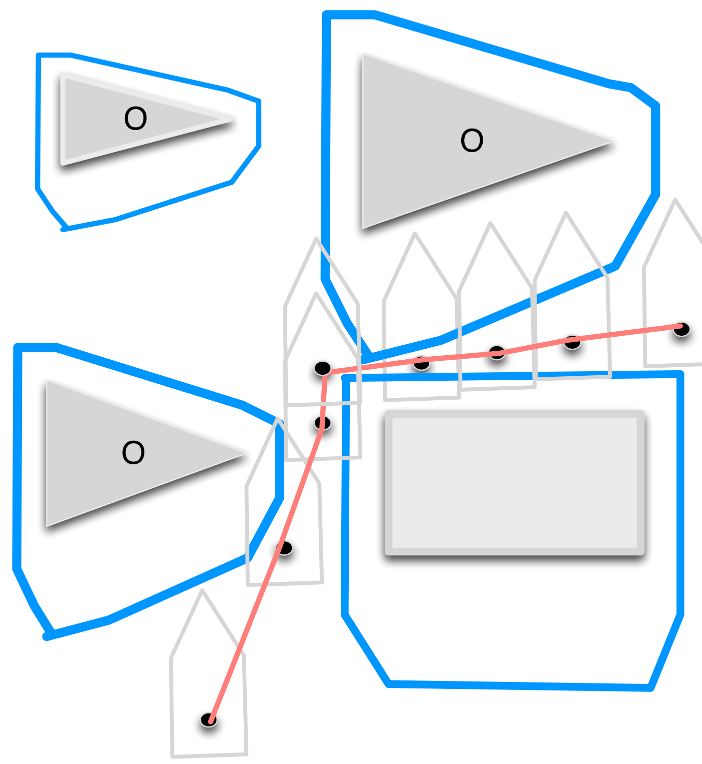
2D

- 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 them, and compute Minkowski sums for each pair of triangles, and take their union
  - size of Minkowski sum: 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 specifies by 3 parameters: R(x,y, theta) ==> C-space is 3D.

Polygonal robot in 2D with rotations

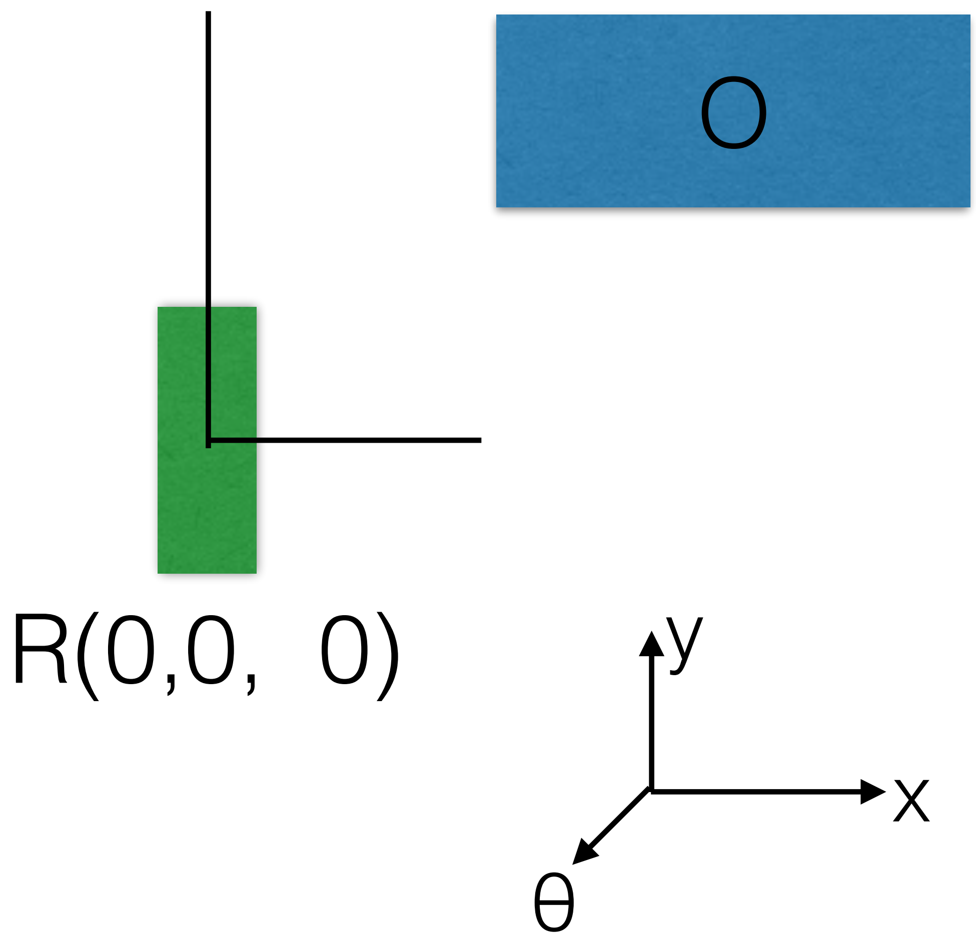
- We'd like to extend the same approach:

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

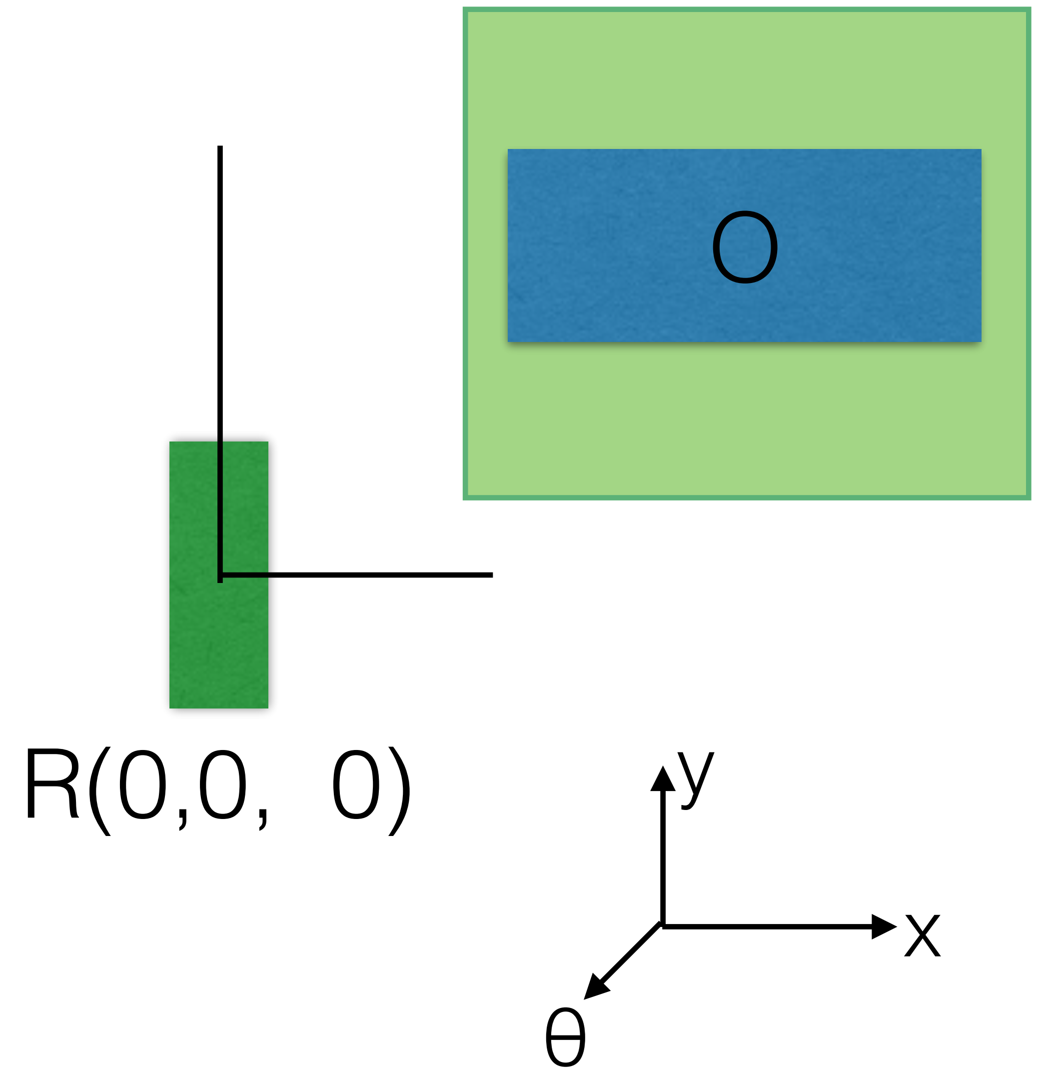
Polygonal robot in 2D with rotations

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



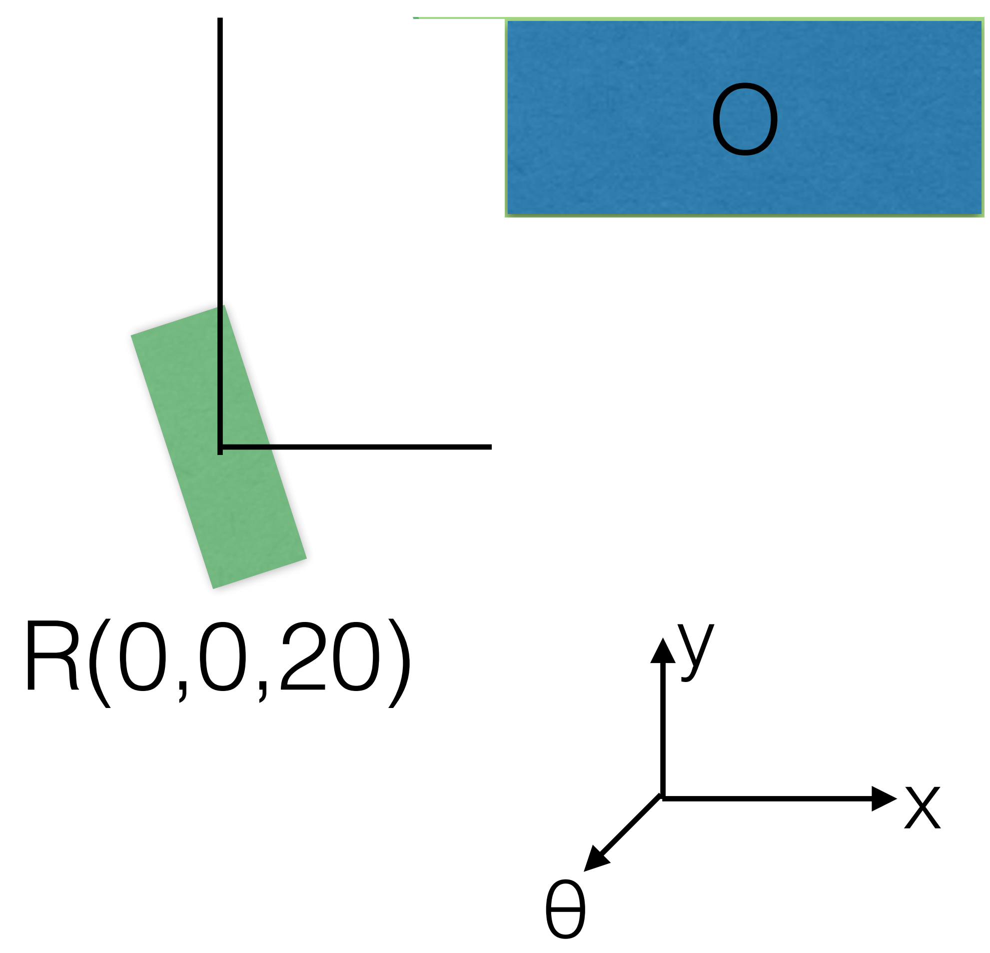
Polygonal robot in 2D with rotations

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



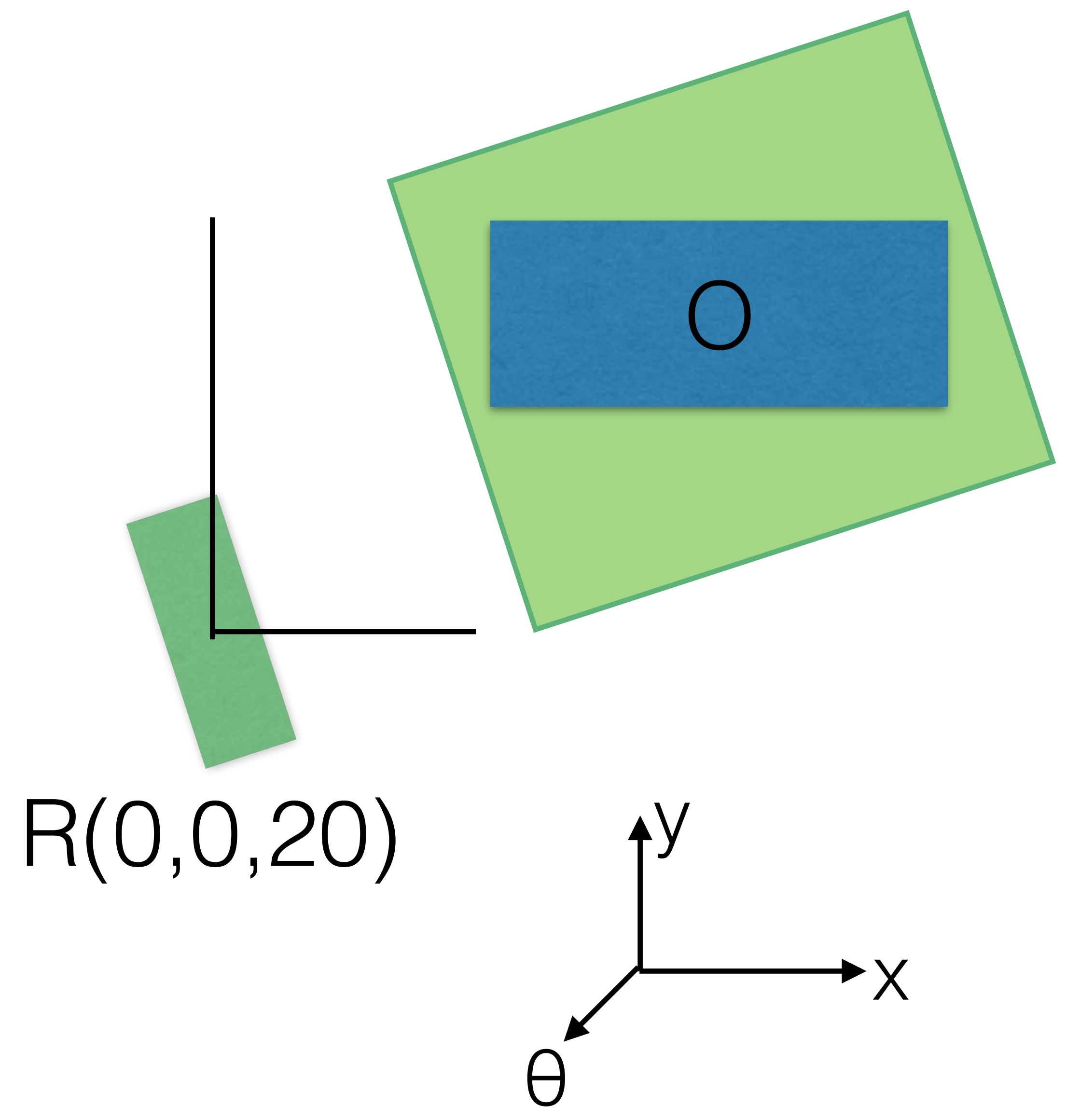
Polygonal robot in 2D with rotations

- 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?



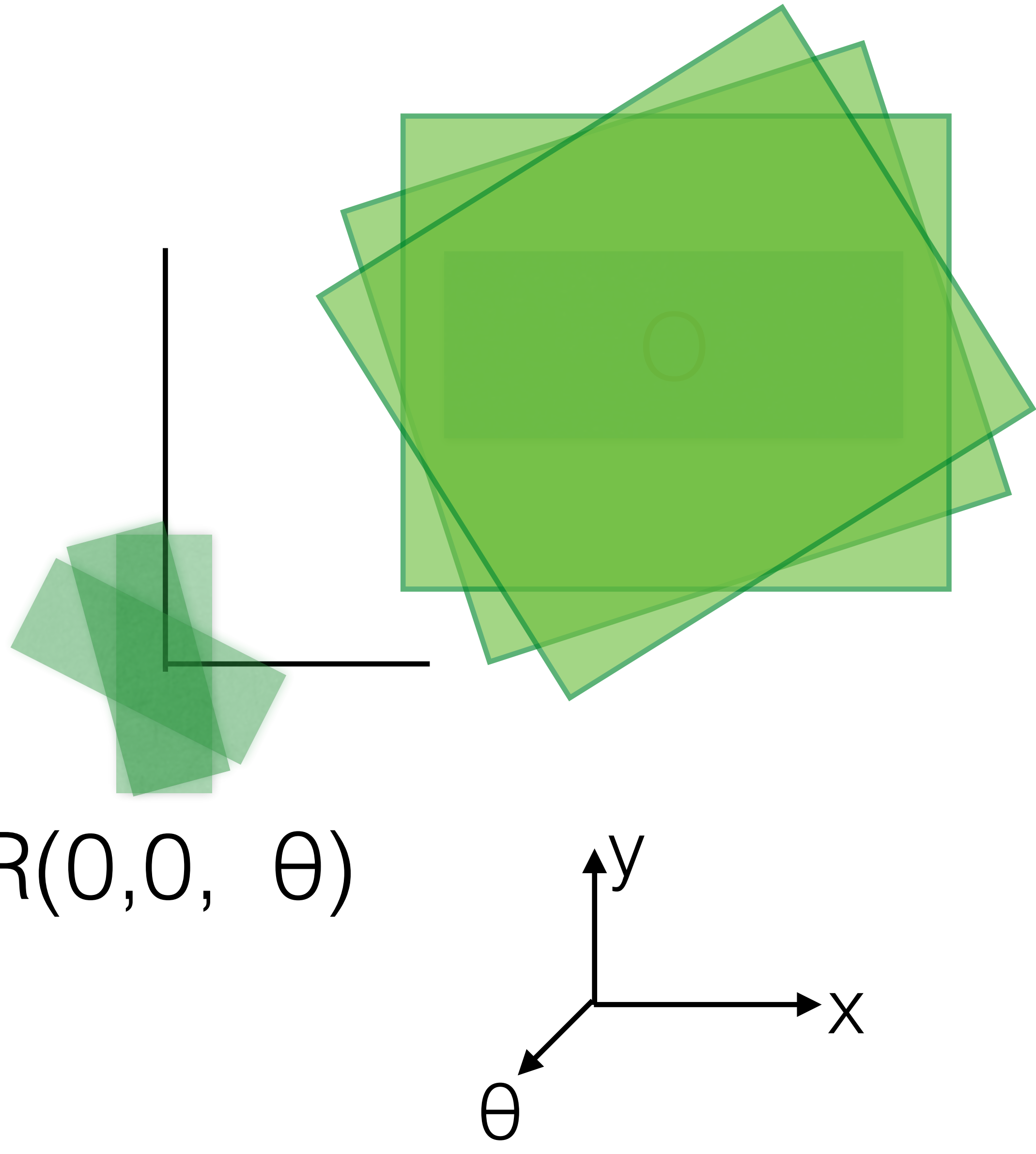
Polygonal robot in 2D with rotations

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 ← space is 3D
  2. compute a decomposition of free space into simple cells
  3. construct a roadmap
  4. BFS on roadmap

Polygonal robot in 2D with rotations

- Difficult to construct a good cell decomposition for curved 3D space
- A simpler approach:
  - For a fixed angle 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
  - Add edges between slices to allow robot to move up/down between slices (this correspond to rotational moves)

  => 3D graph

Is this complete?

Heuristical/approximate motion planning

- Approximate cell decomposition
    - grid
    - quadtrees
- Potential field
- Roadmaps
    - Incremental sampling
    - Probabilistic roadmaps
- Hybrid

> search/explore free space:
>
> AI search heuristics

Issues:
huge C-space, local minima
performance guarantees? completeness? optimality?

Potential field methods

- Idea:
  - Define a potential field
  - Robot moves in the direction of steepest descent on potential function

- Ideally potential function has global minimum at the goal, has no local minima, and is very large around obstacles
- Algorithm outline:
  - place a regular grid over C-space
  - search over the grid with potential function as heuristic

- Con: can get stuck in local minima