Combinatorial path planning

2. Polygonal robot among obstacles in 2D

csci3250: Computational Geometry
Laura Toma
Bowdoin College

Path planning in 2D

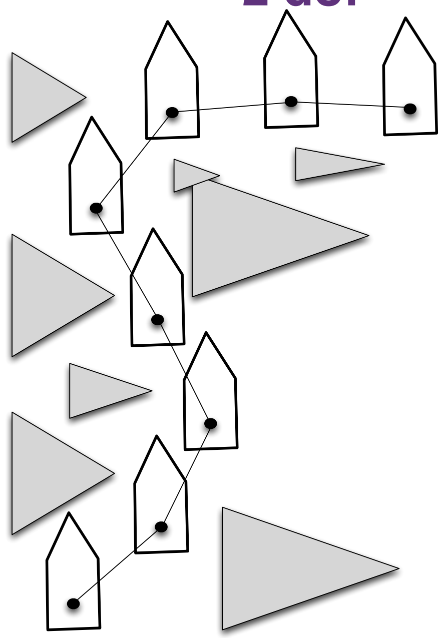
- point robot moving among arbitrary polygons in 2D

today

- polygonal robot moving among arbitrary polygons in 2D

translation only

2 dof



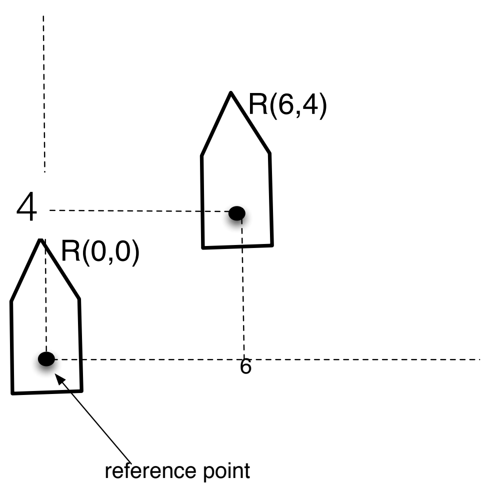
translation+rotation

3 dof

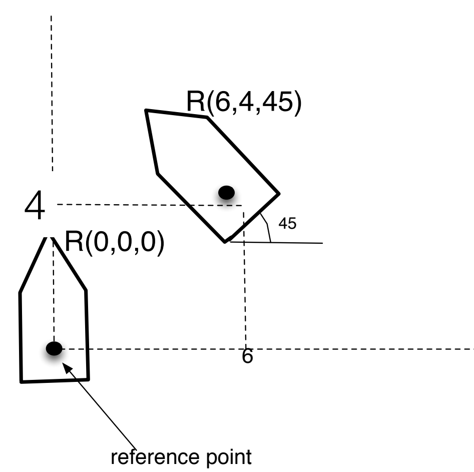
screenshot from internet

Physical space vs Degrees of freedom

- Physical space: Space where robot moves around
  - e.g. we are in 2D
- Degrees of freedom: How many independent ways can the robot move?
  - translation X and Y => 2 dof
  - translation+ rotation => 3 dof



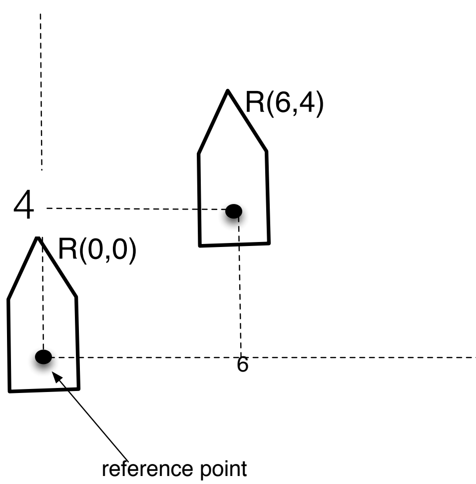
translation only



translation + rotation

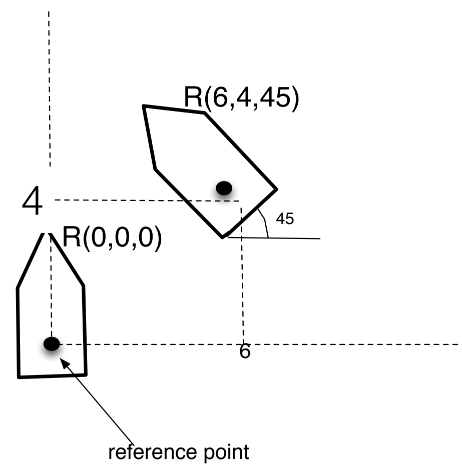
Placement

- A placement of a robot is a set of coordinates that specify where the robot is in space. One coordinate per degree of freedom (dof)
    - translation only: a placement of the robot is specified by (x, y)
    - translation+ rotation: a placement of the robot is specified by (x, y, θ)



translation only

$$R(x, y)$$

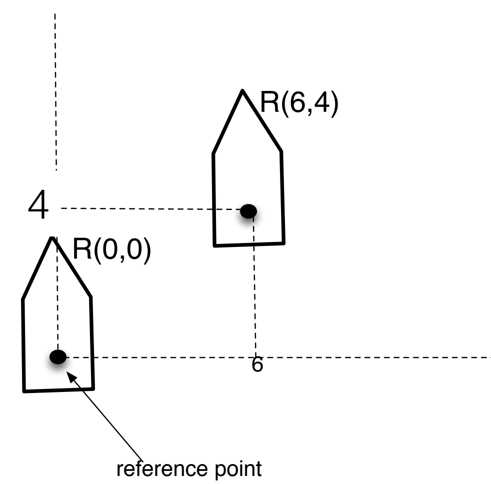


translation + rotation

$$R(x, y, \theta)$$

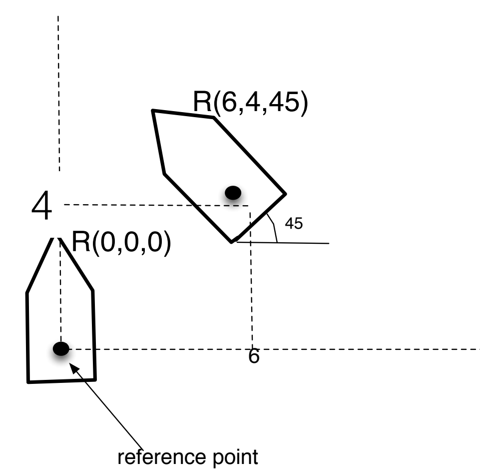
Configuration space (C-space)

- The parametric space of the robot = space of all possible placements of the robot
- A point in C-space corresponds to placement of the robot in physical space



translation only

C-space = all placements (x, y)



translation + rotation

C-space = all placements (x, y, θ)

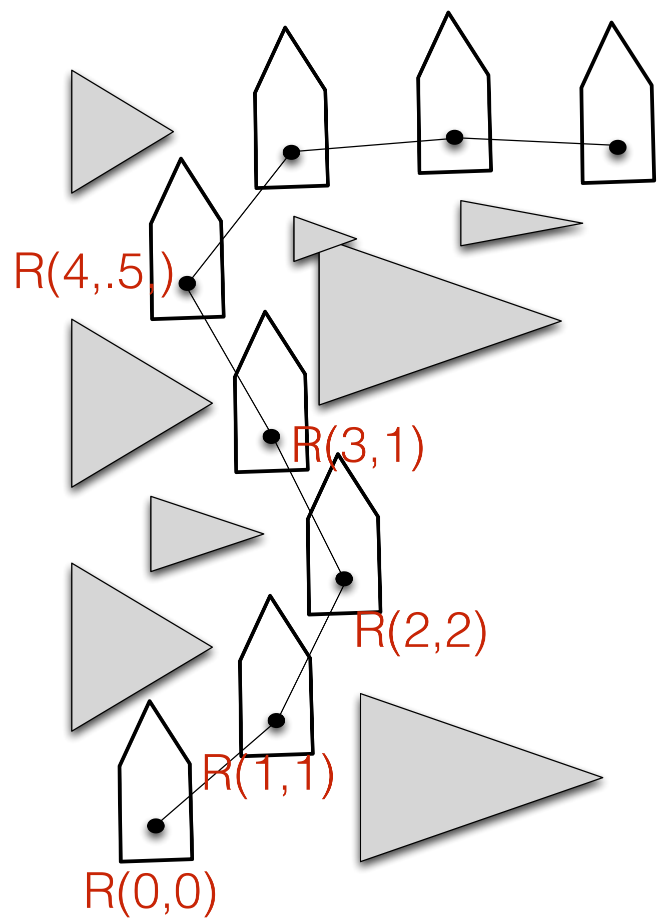
Physical Space and C-space

physical space	robot	C-space
2D	polygon, translation only $R(x, y)$	2D
2D	polygon, translation + rotation $R(x, y, \theta)$	3D
3D	polygon, translation + rotation $R(x, y, z, \theta_x, \theta_y, \theta_z)$	6D
3D	robot with arms and joints	many dof

Path planning in C-space

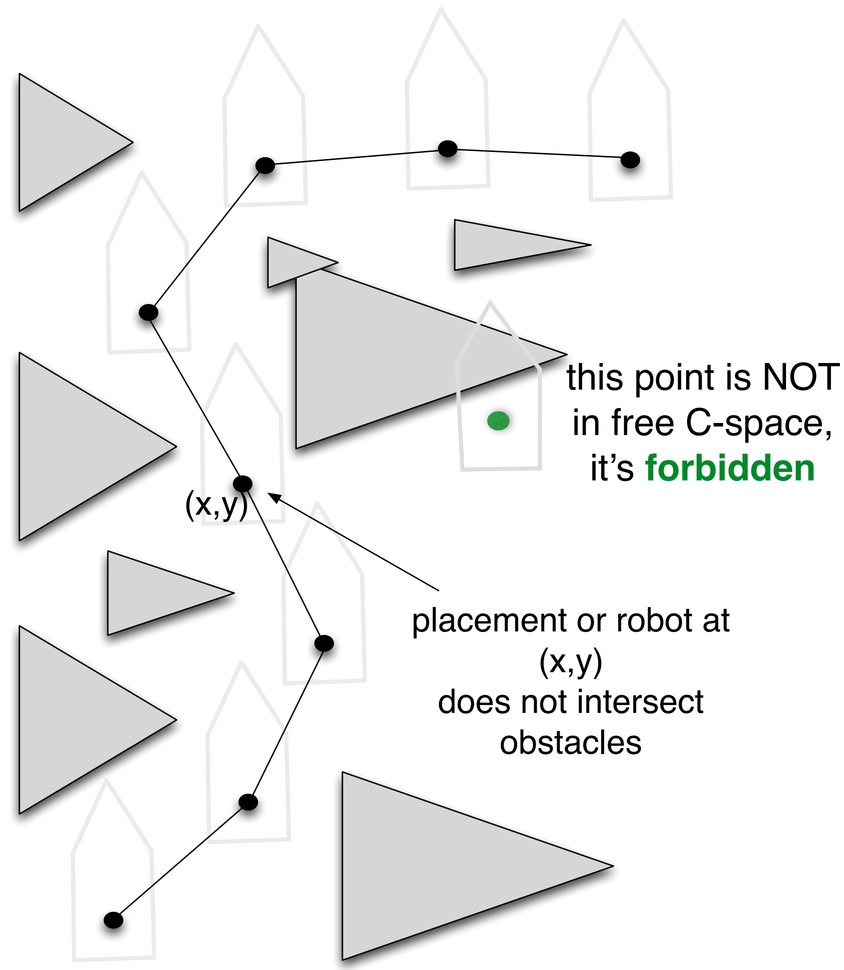
- The robot moves in physical space. Any path for robot in physical space corresponds to a a set of placements in C-space ==> a path in C-space
- Path in physical space <==> path in C-space

Path planning is done in C-space because it captures the dof of the robot



Free and forbidden points in C-space

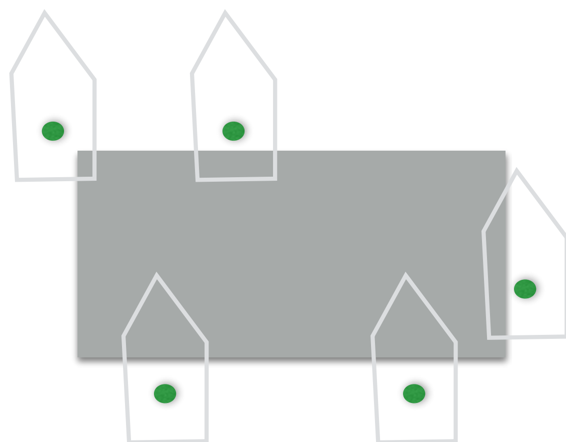
- A configuration (x,y, …) in C-space is **free** if placing R(x,y, …) does not intersect any obstacle, and **forbidden** otherwise



- In general if we have a k-dimensional C-space: a configuration $(x_1, x_2, ..x_k)$ is **free** if placing $R(x_1, x_2, ..., x_k)$ does not intersect any obstacle, and **forbidden** otherwise

Physical Obstacles ==> "Extended" C-obstacles

Not every placement R(x, y) "outside" the obstacle is free of collisions.

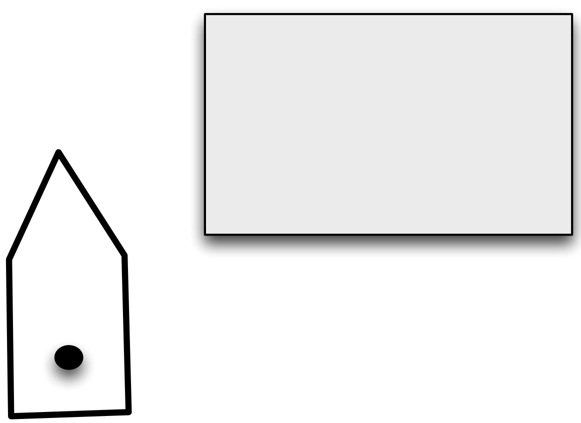


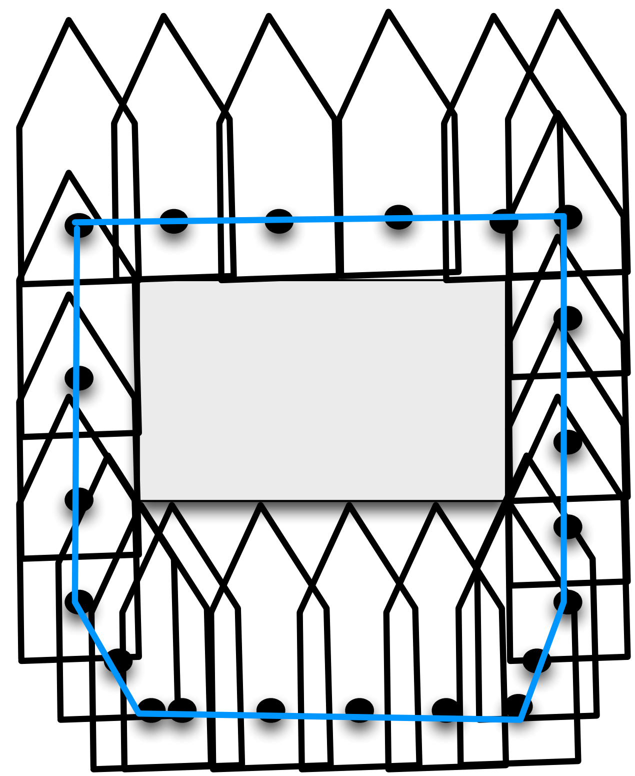
Extended obstacle in C-space:

the set of placements (x,y) so that R(x,y) intersects that obstacle

Extended obstacles or C-obstacles

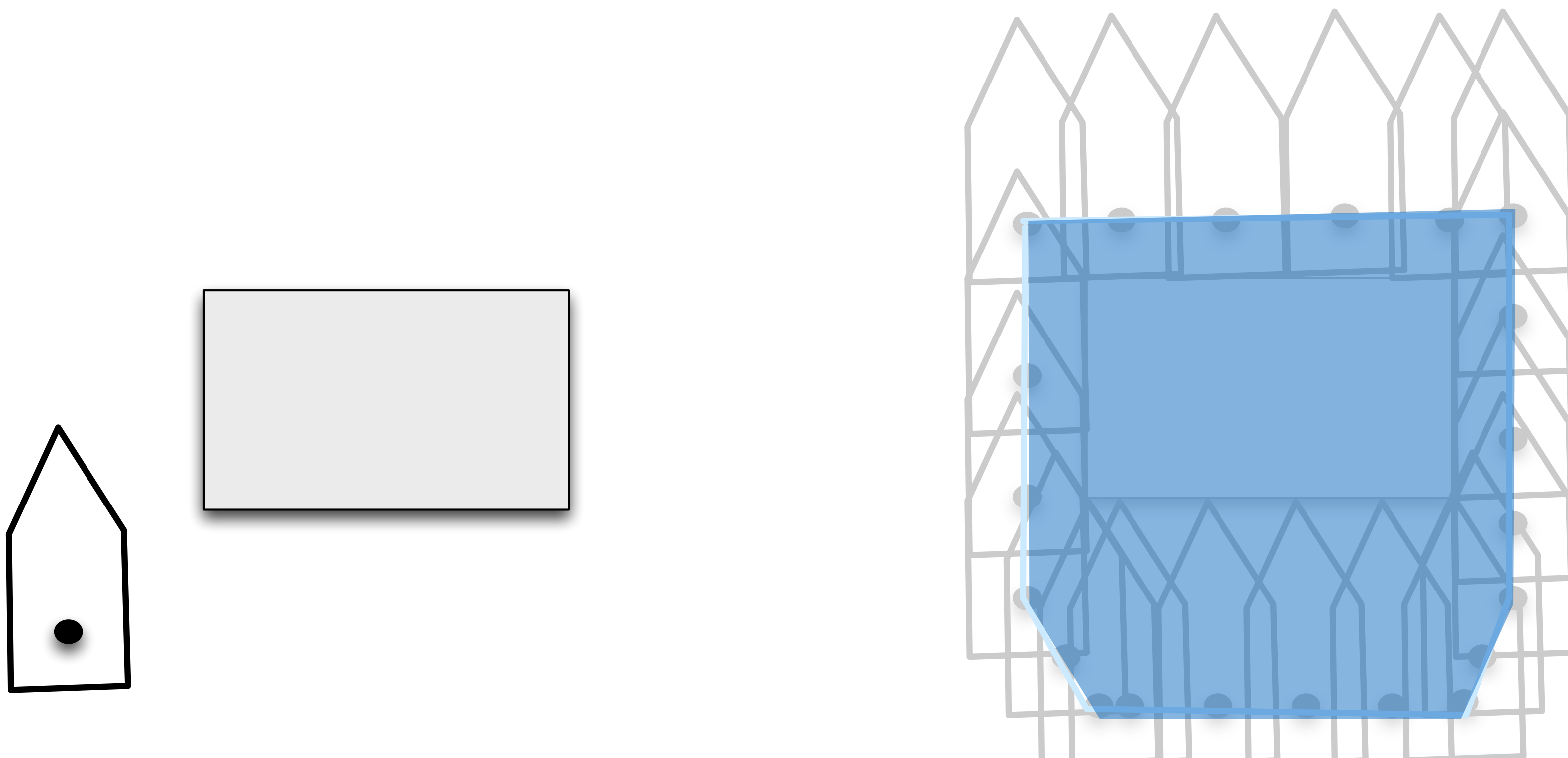
- Given obstacle O and robot R
    - C-obstacle = the placements of R that cause intersection with O





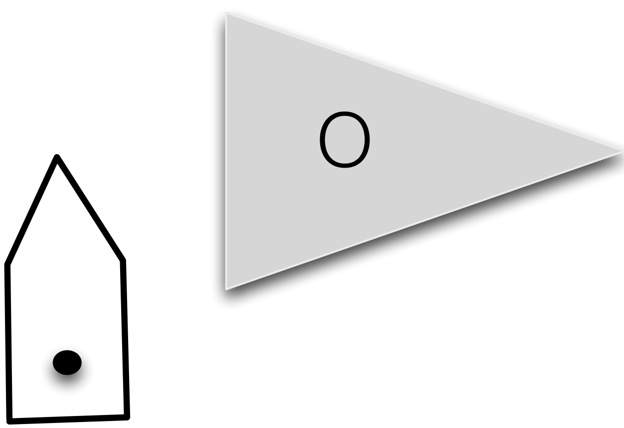
Extended obstacles or C-obstacles

- Given obstacle O and robot R
    - C-obstacle = the placements of R that cause intersection with O



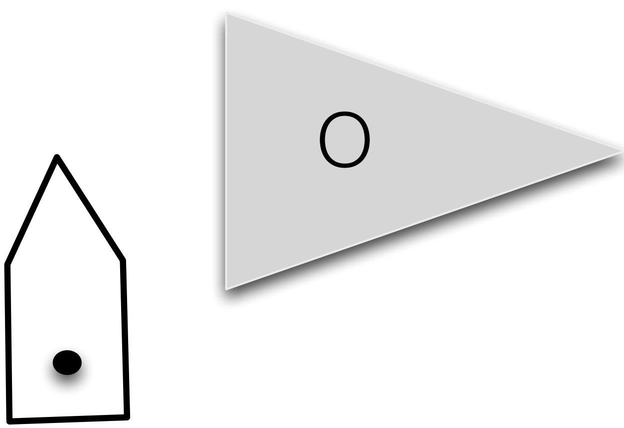
Extended obstacles or C-obstacles

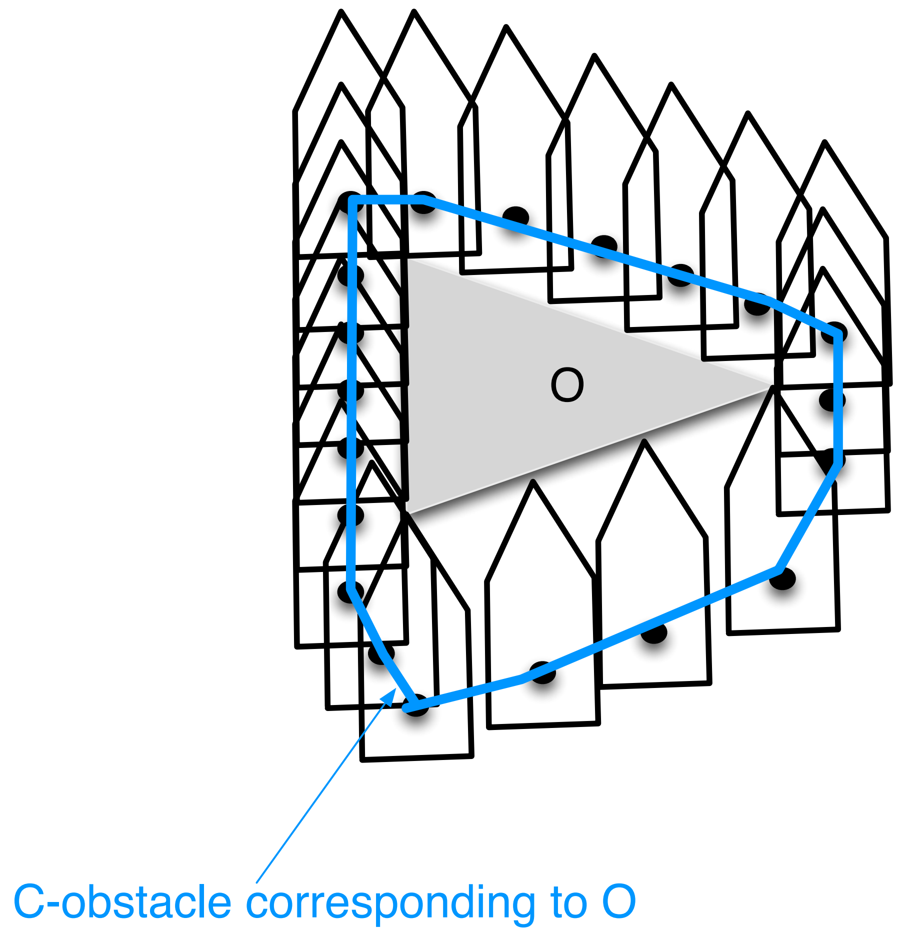
- Given obstacle O and robot R
    - C-obstacle = the placements of R that cause intersection with O



Extended obstacles or C-obstacles

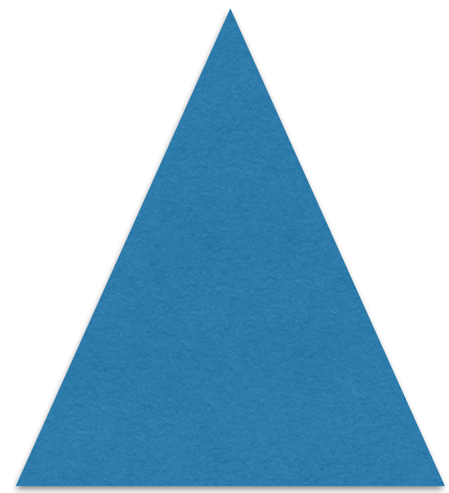
- Given obstacle O and robot R
    - C-obstacle = the placements of R that cause intersection with O





Class work

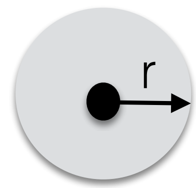
Find the corresponding C-obstacles



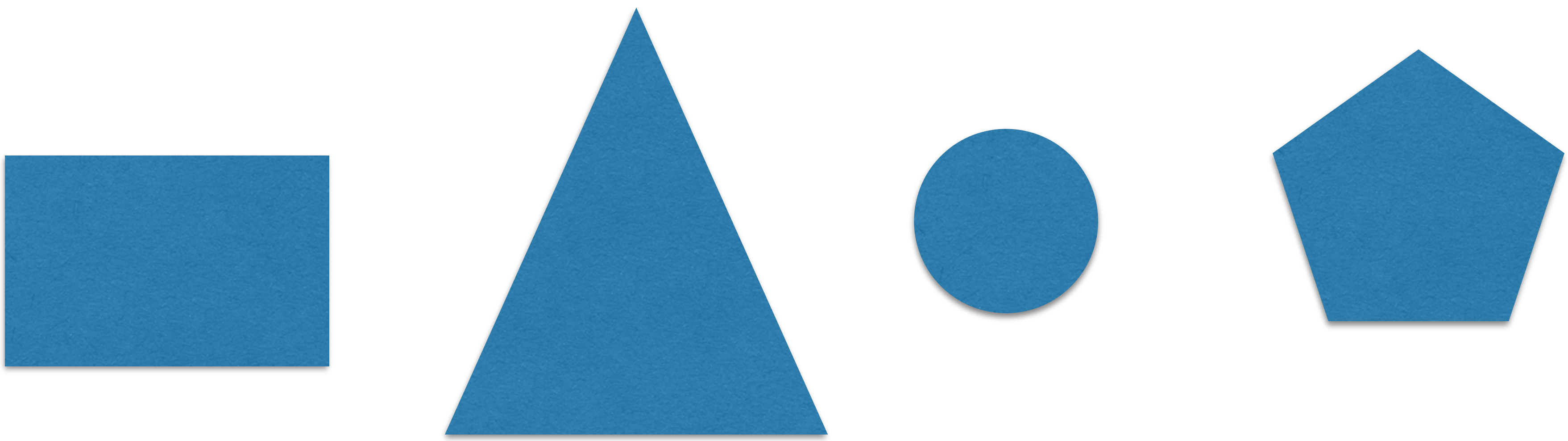
- Draw a small set of obstacles such that their C-obstacles overlap.
- Draw a scene of obstacles such that free physical space is not disconnected, but the the free C-space is disconnected.

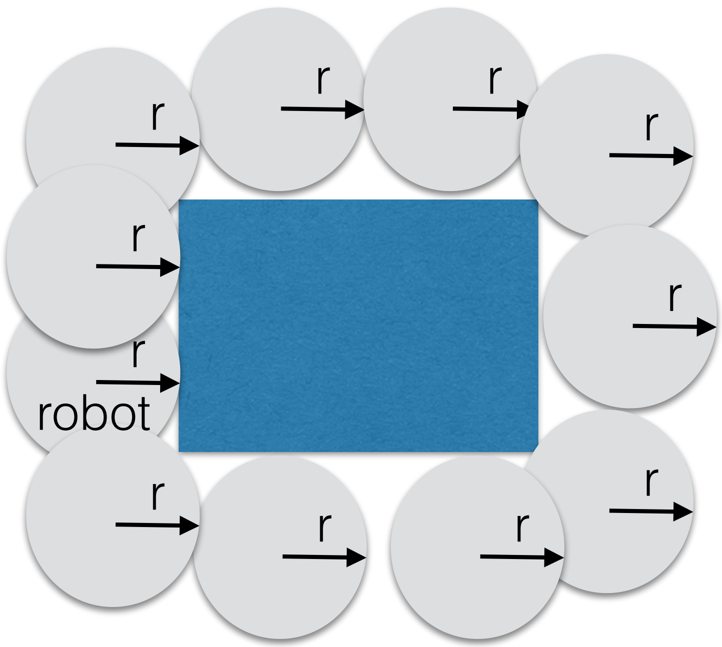
Class work

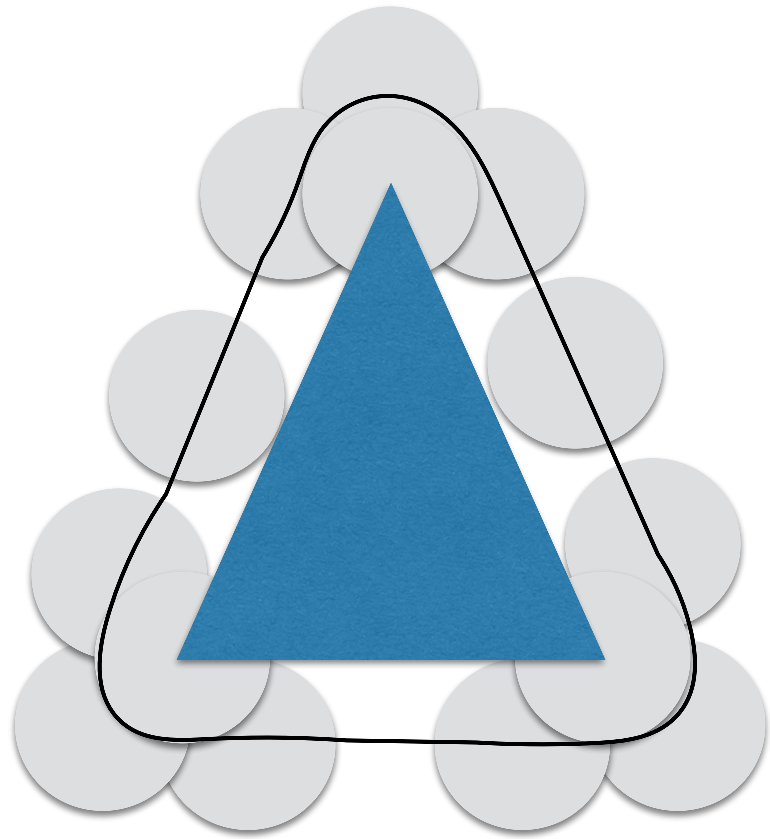
Find the corresponding C-obstacles



robot



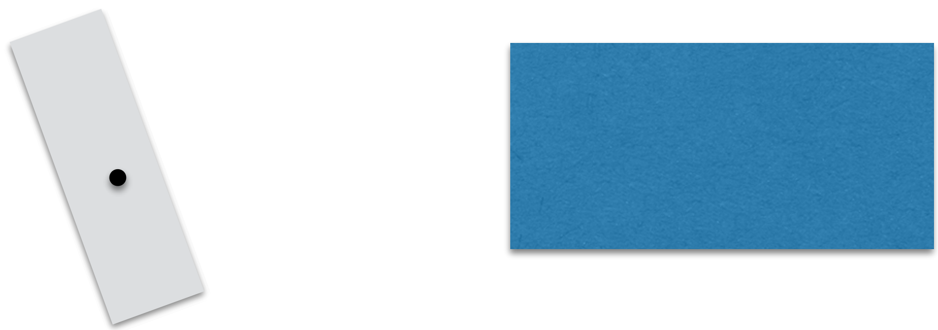

r
r
r
r
r
r
r
robot
r
r
r
r



Class work

Find the corresponding C-obstacles

translation only

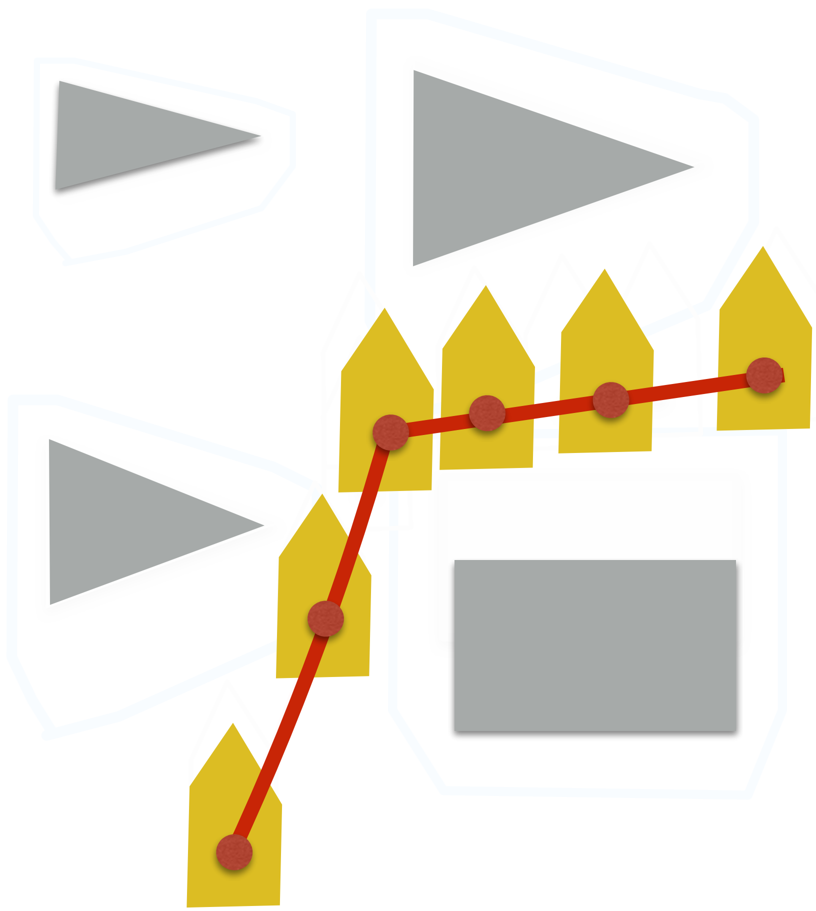


robot

Polygonal robot translating in 2D

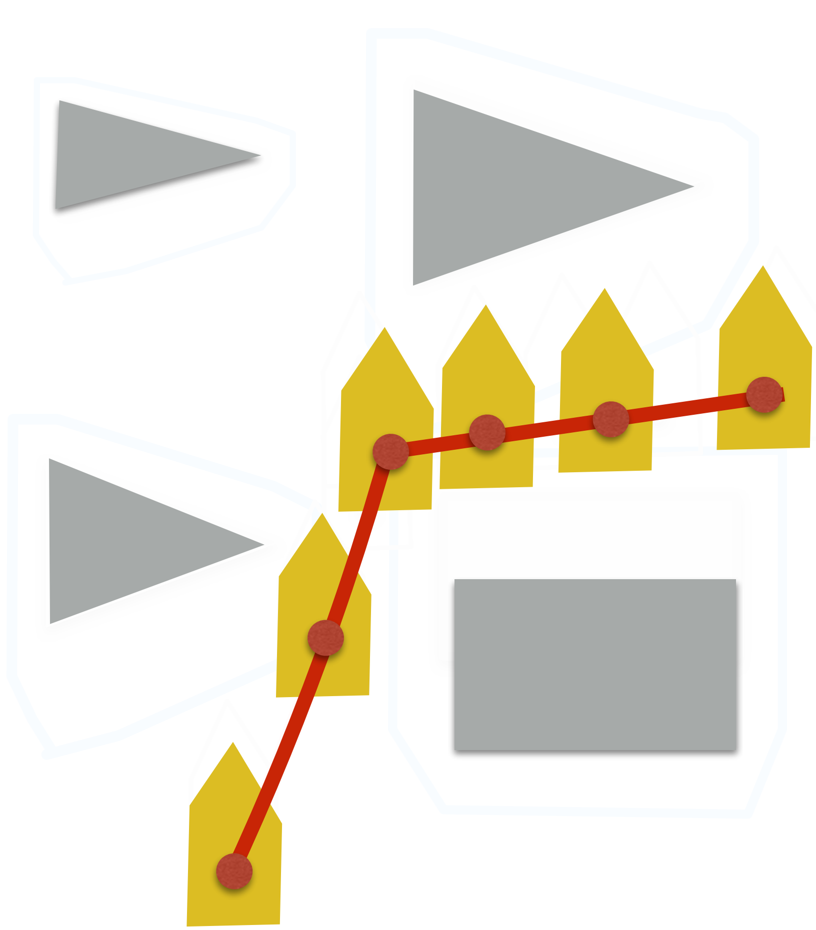
We want a collision free path for the robot from *start* to *end*

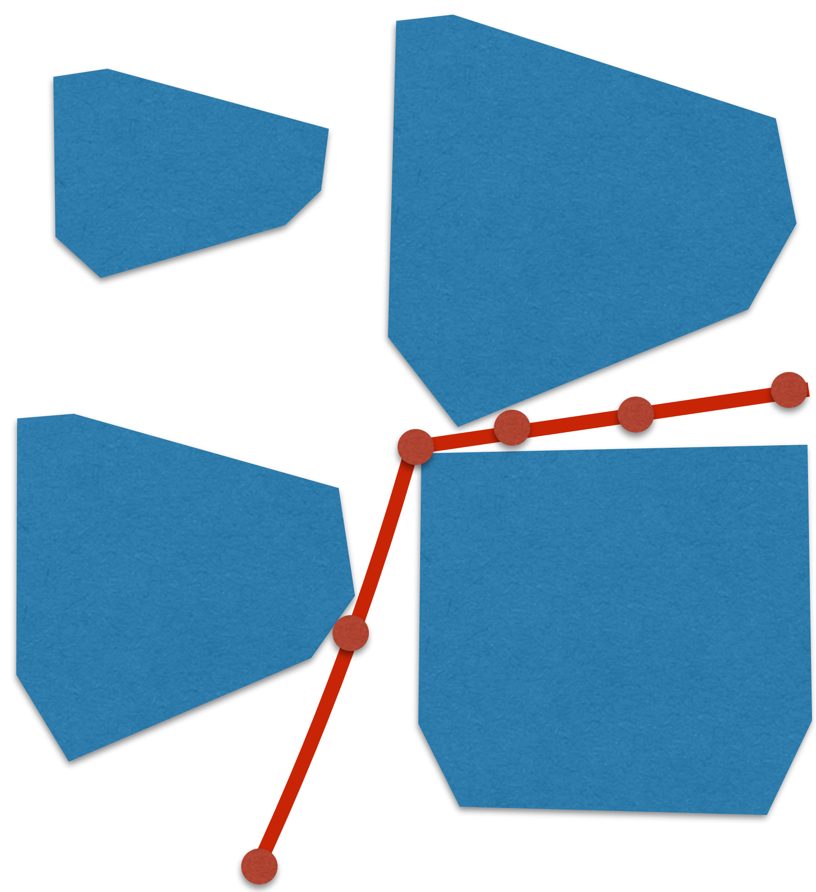
Any placement R(x, y) along the path is in free C-space and thus outside the C-obstacles



Polygonal robot translating in 2D

polygonal robot R among obstacles → **point** robot among C-obstacles



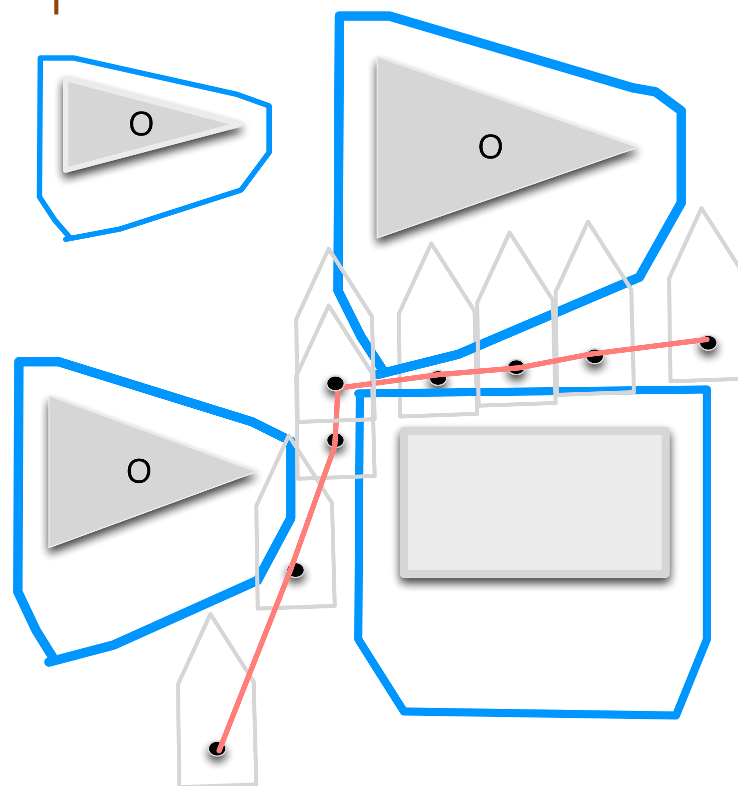


Polygonal robot translating in 2D

Algorithm (list of obstacles, robot R)

- For each obstacle O, compute the corresponding C-obstacle
- Compute the union of C-obstacles, then compute its complement. That's the free C-space

  //planning for R reduces to planning for point-robot moving in free C-space
- Compute a roadmap of free C-space
  - a trapezoidal decomposition graph + BFS
  - or, a visibility graph + Dijkstra



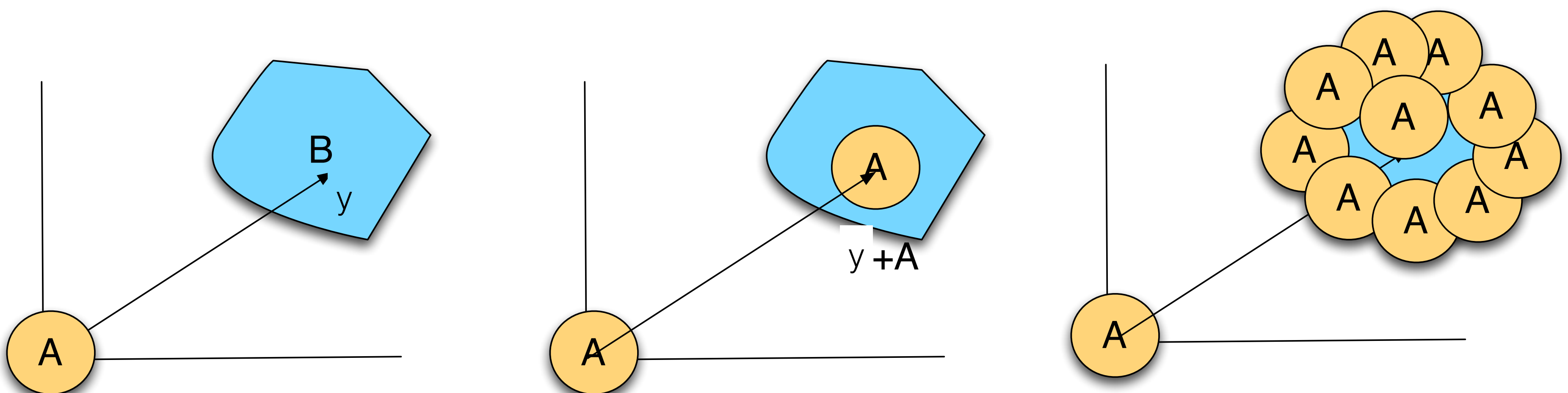
How to compute C-obstacles?

Minkowski sum

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

vector sum   x, y vectors

- Interpretation: consider set A to be centered at the origin. Then $B \oplus A$ 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.

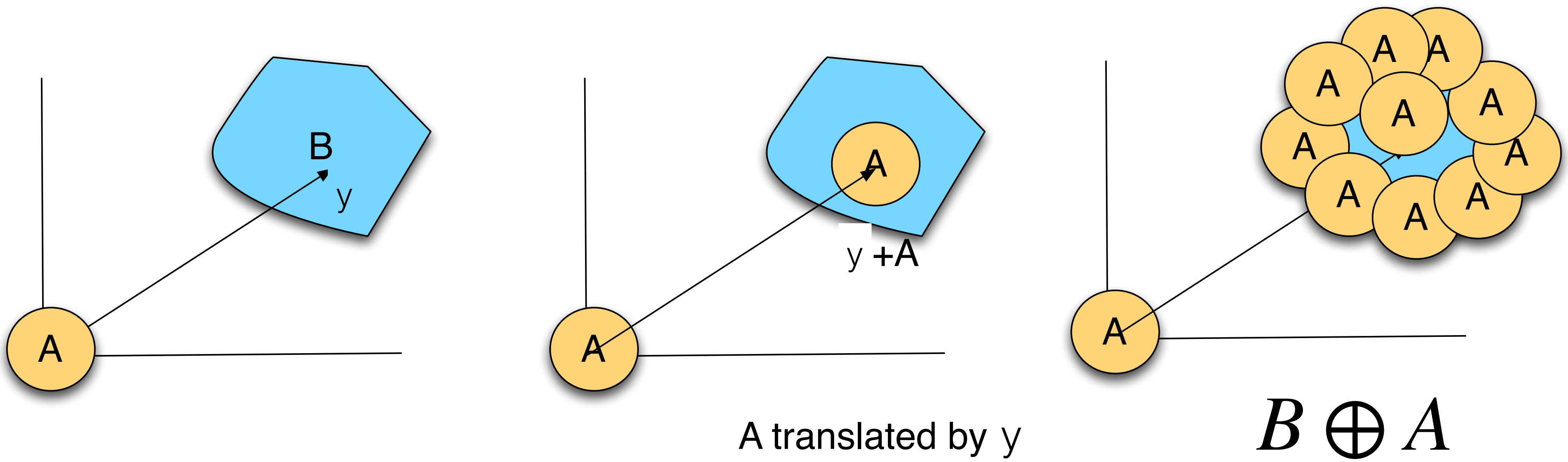


A translated by y

$B \oplus A$

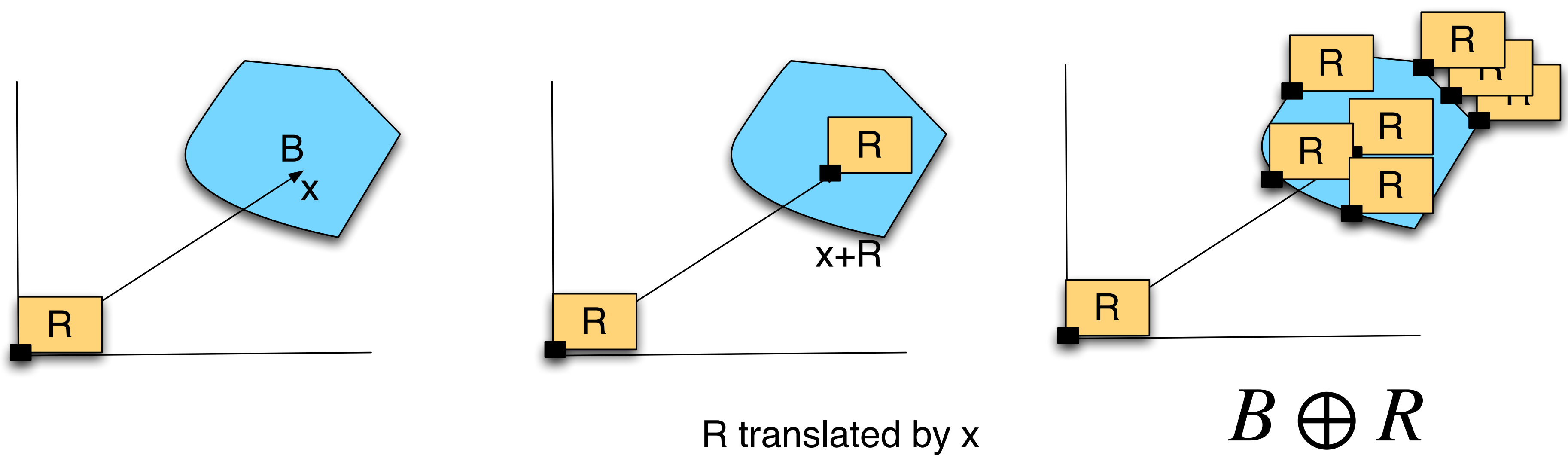
Minkowski sum

- What is the boundary of $B \oplus A$?
  - Slide A so that the center of A traces the boundary of B



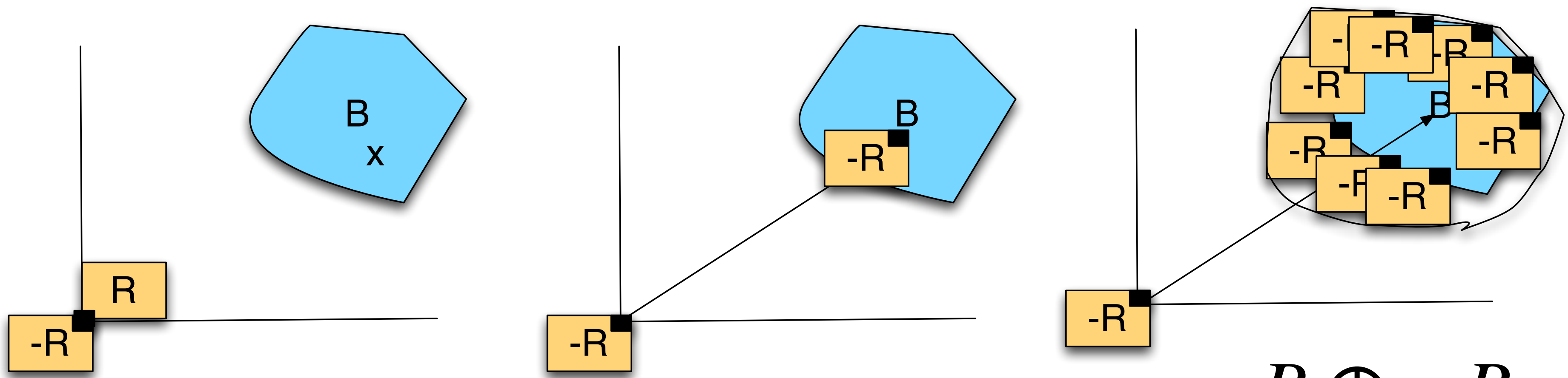
C-obstacles as Minkowski sums

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



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

C-obstacles as Minkowski sums

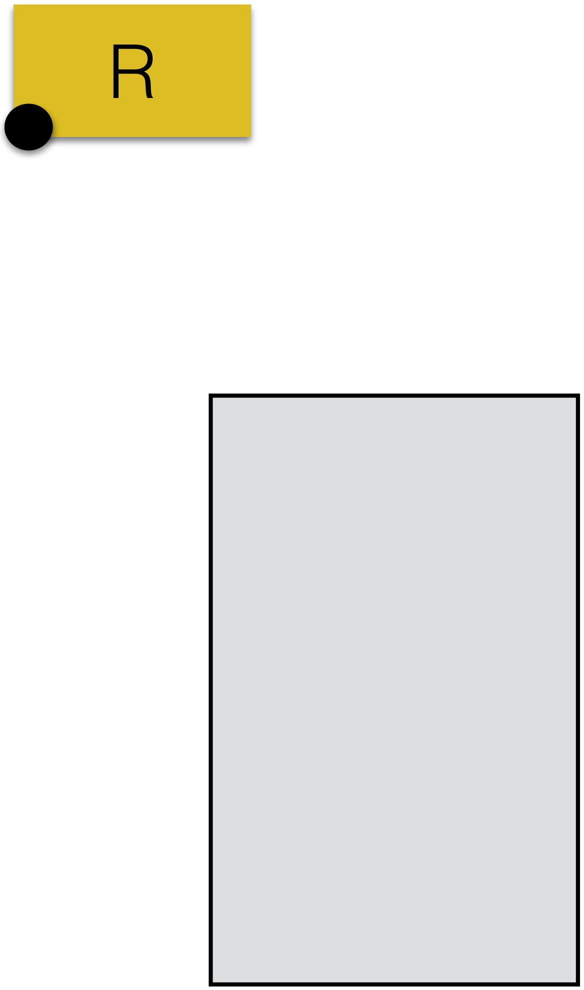


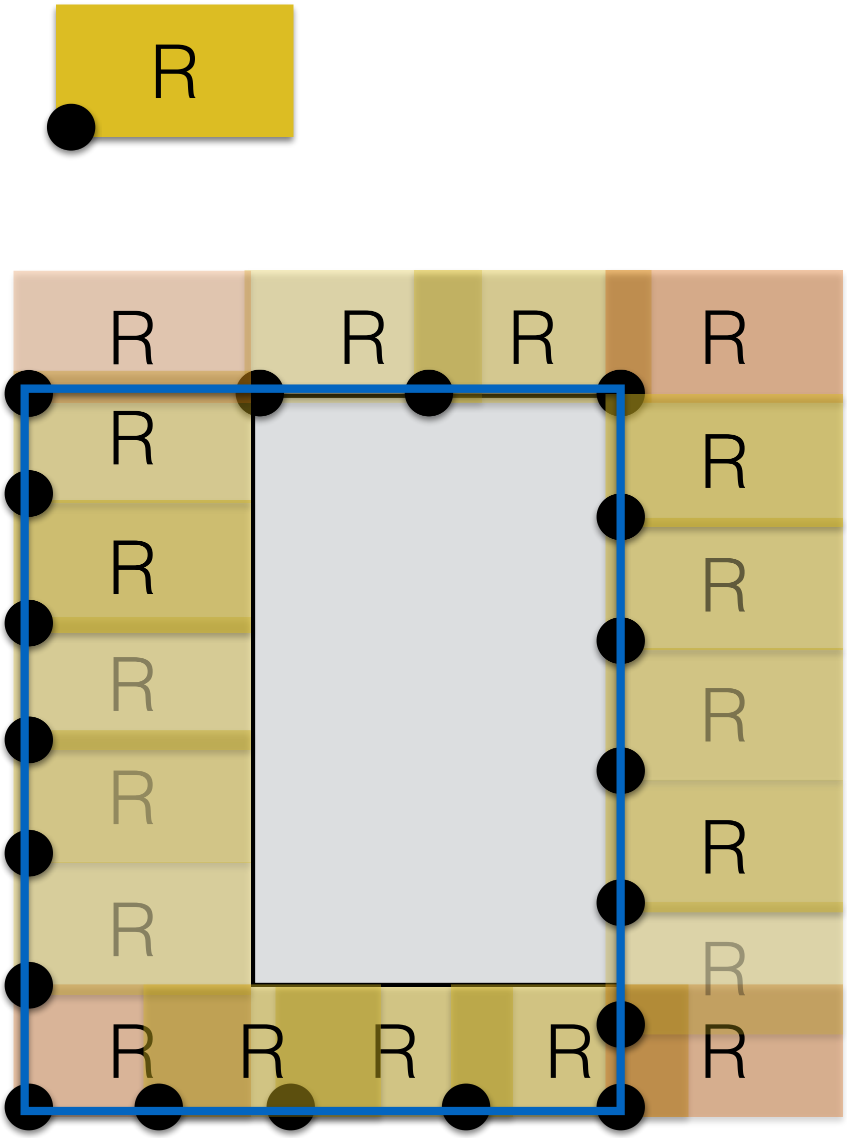
-R translated by x

$B \oplus -R$

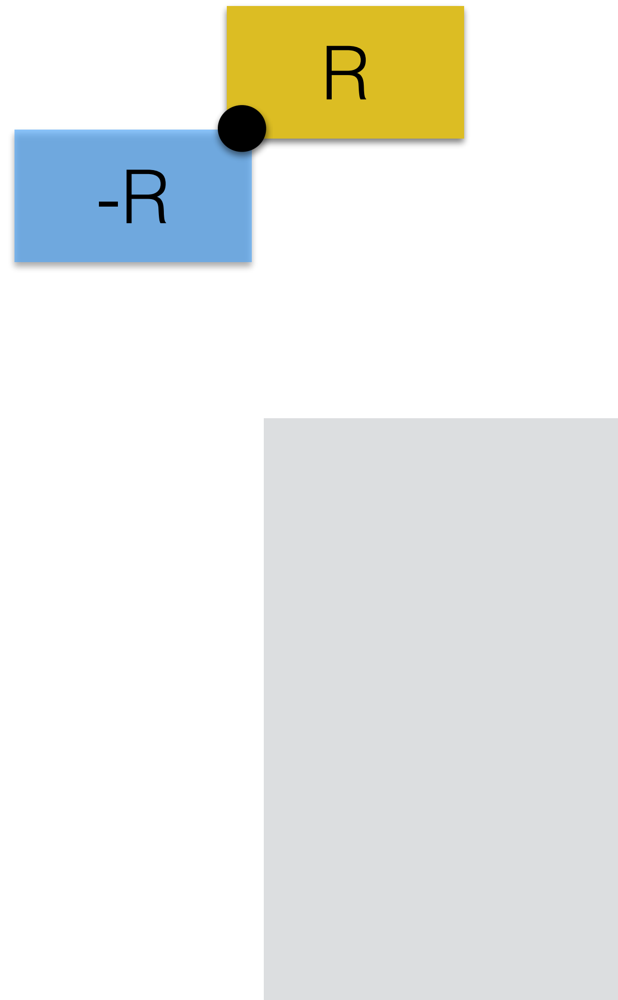
-R: R reflected by origin

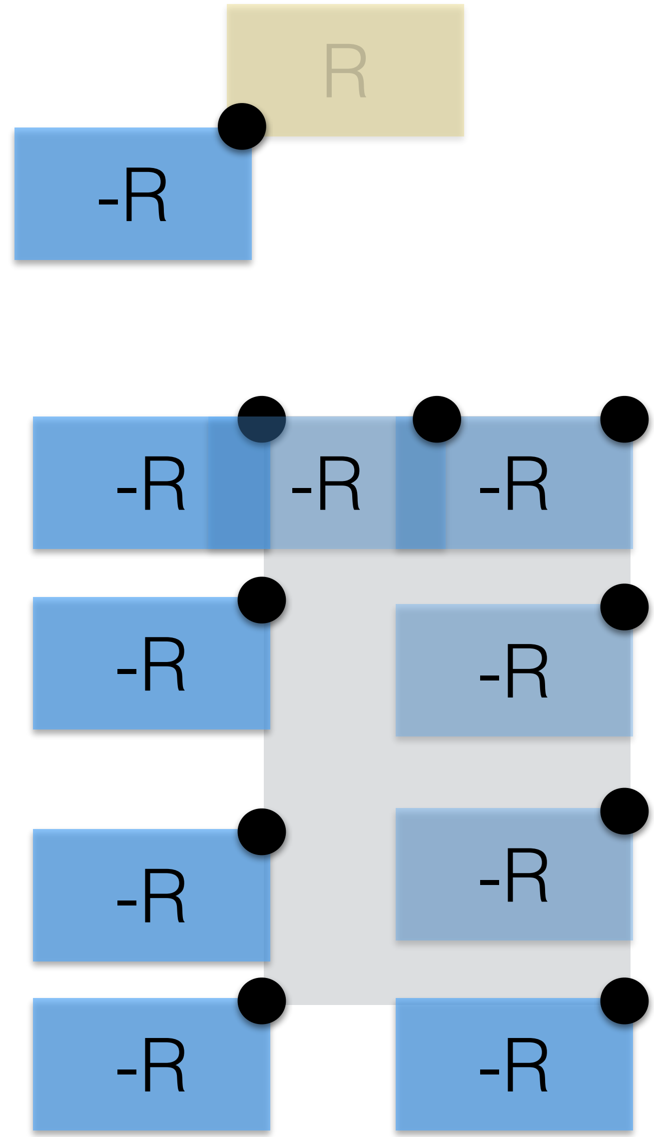
The C-obstacle of B is $B \oplus -R(0,0)$


R

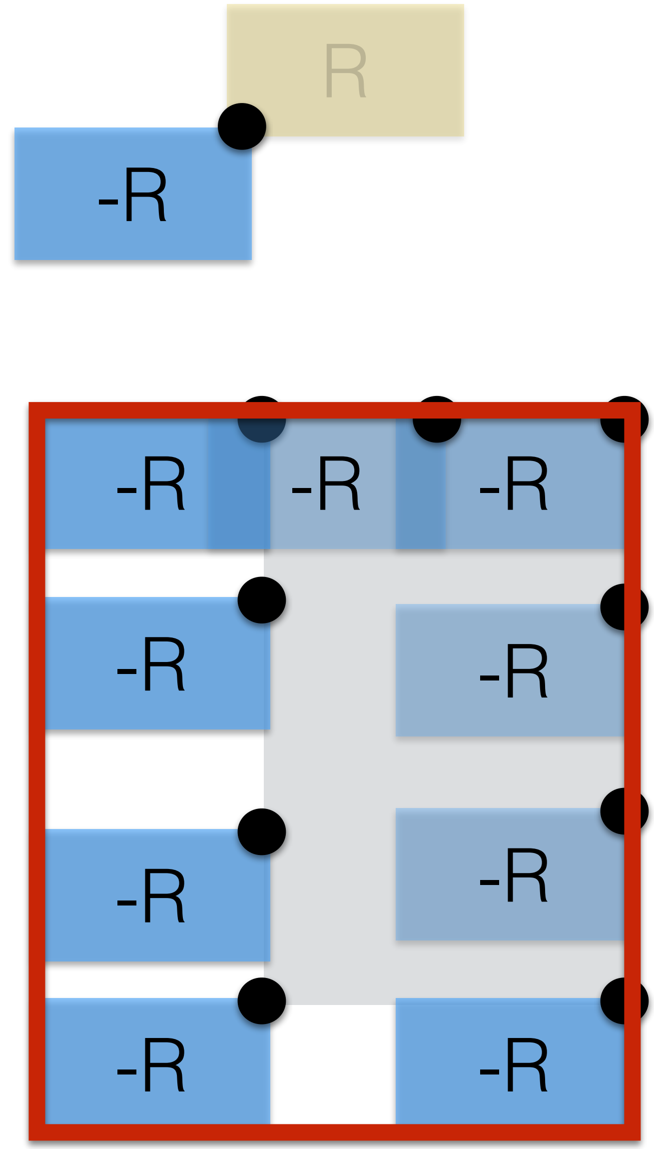


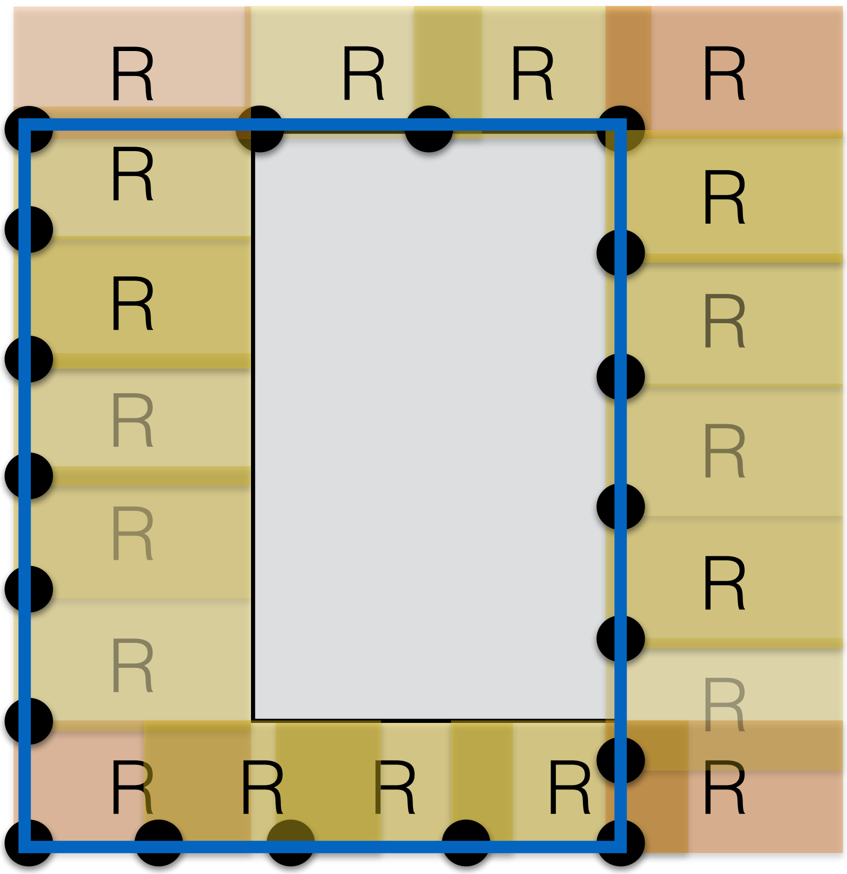
extended obstacle


R
-R

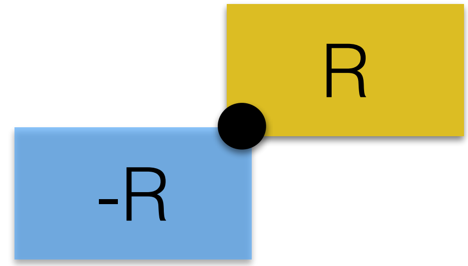


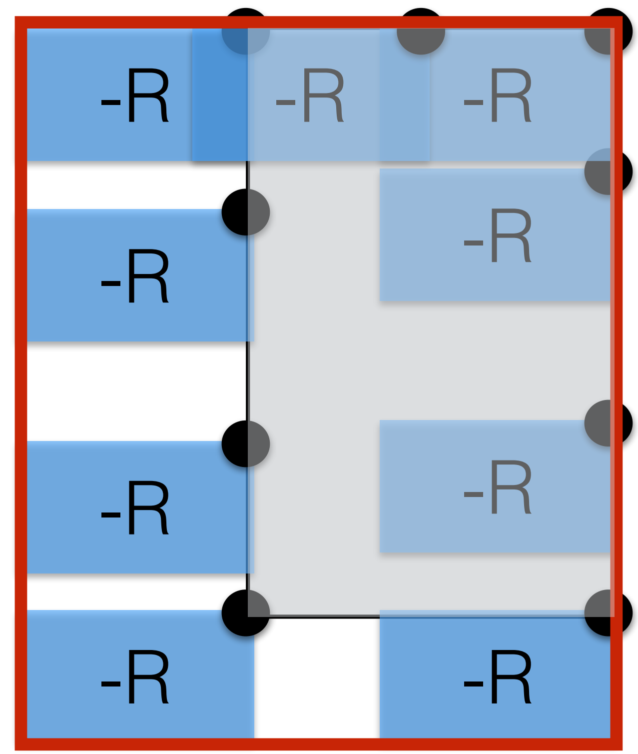
$O \oplus -R(0,0)$


R
-R
-R
-R
-R
-R
-R
-R
-R
-R
-R

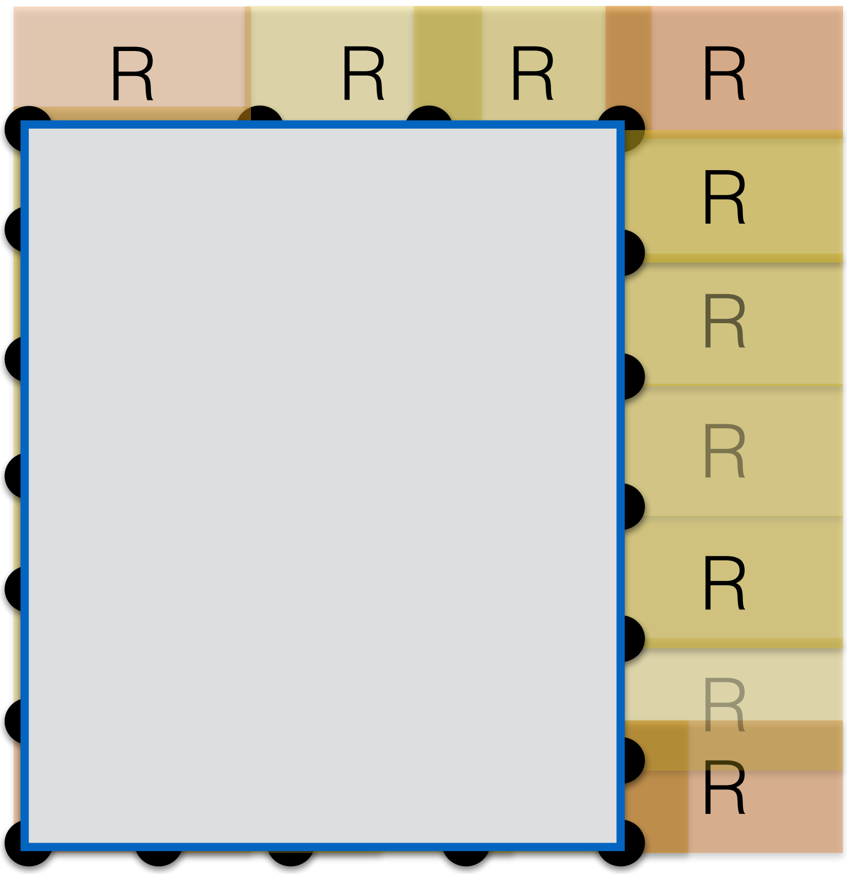


extended obstacle

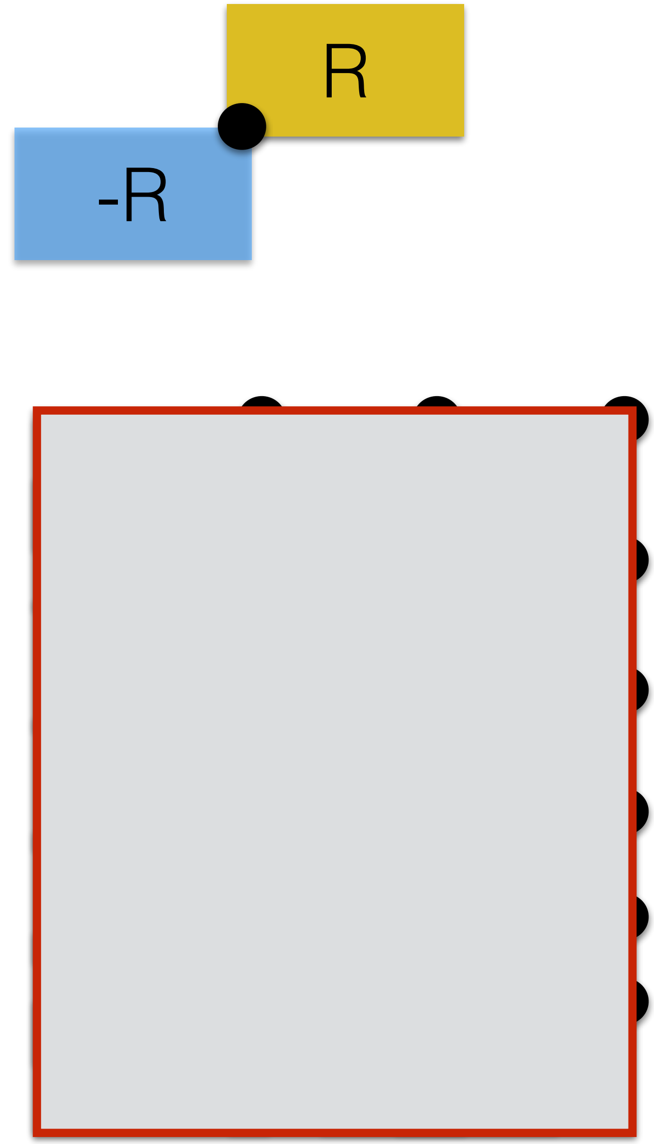




$O \oplus -R(0,0)$

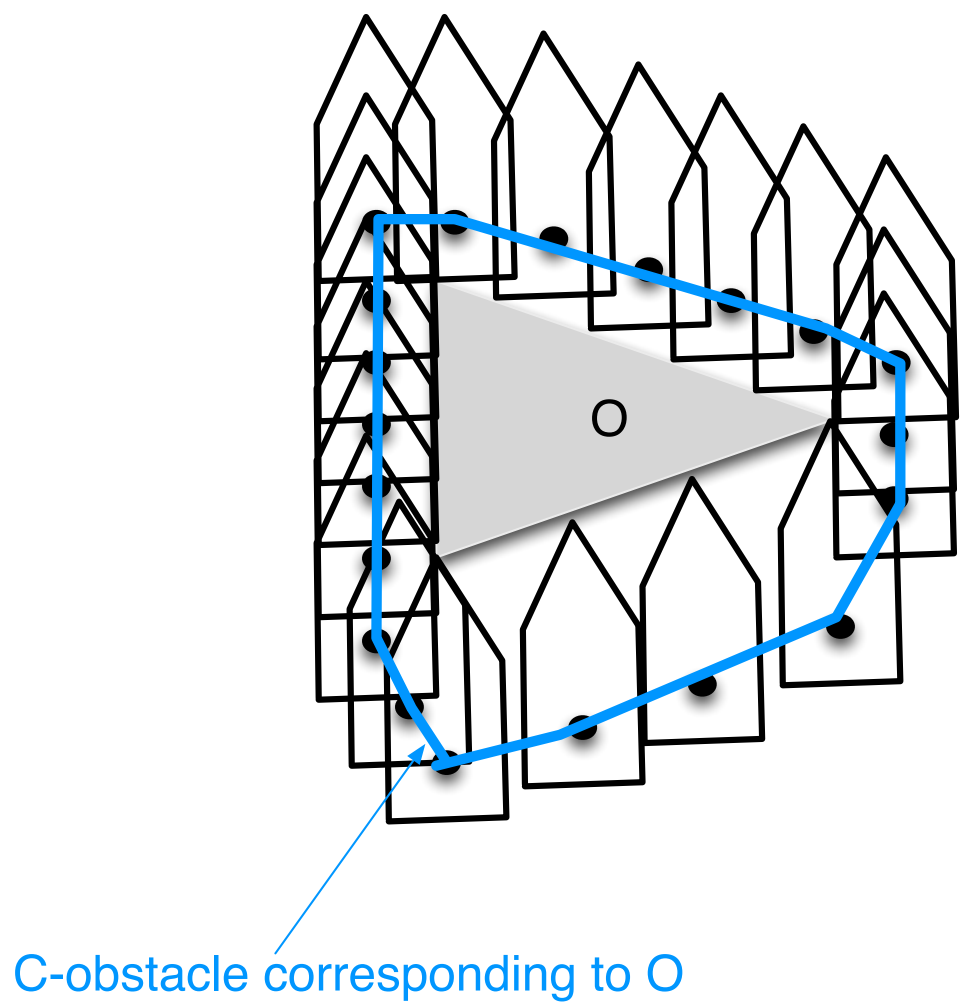


extended obstacle



$O \oplus -R(0,0)$

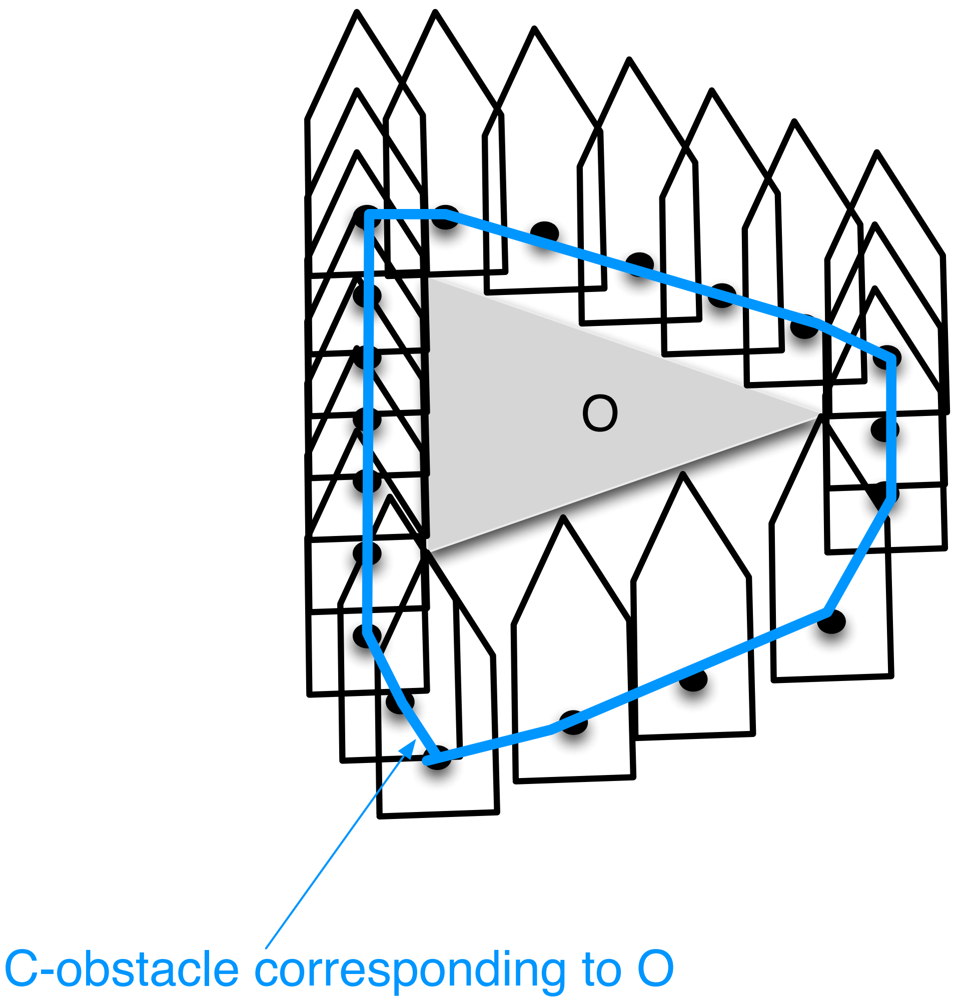
Slide so that R touches the obstacle

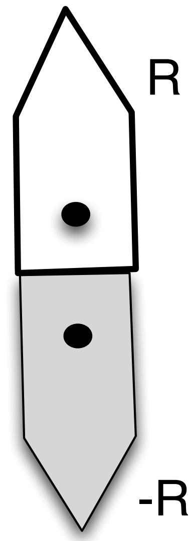


Slide so that R touches the obstacle

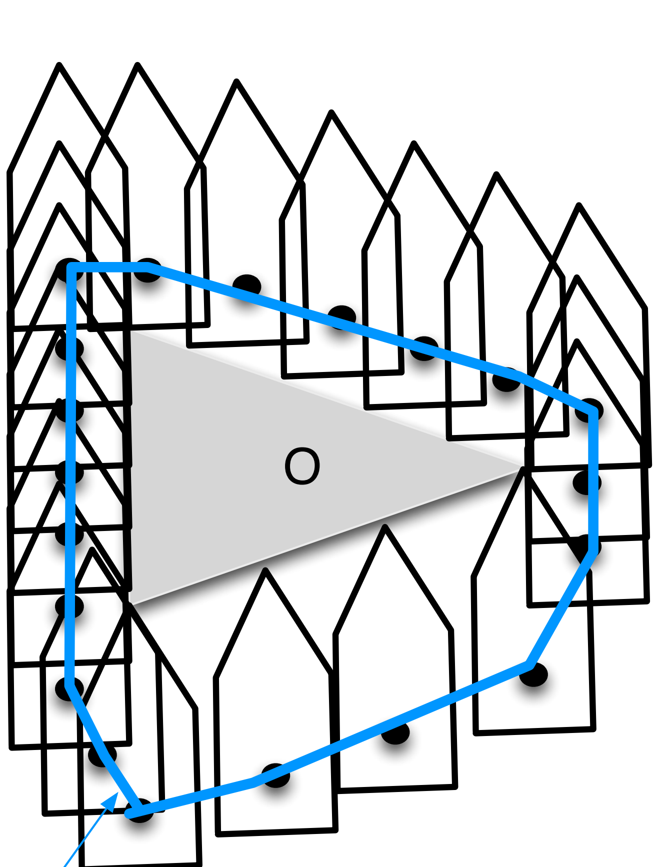
Find O + (-R)

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





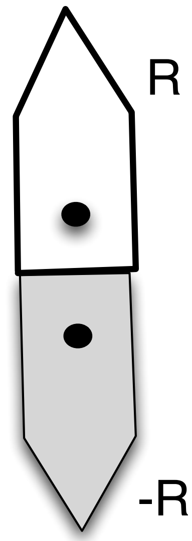
Slide so that R touches the obstacle

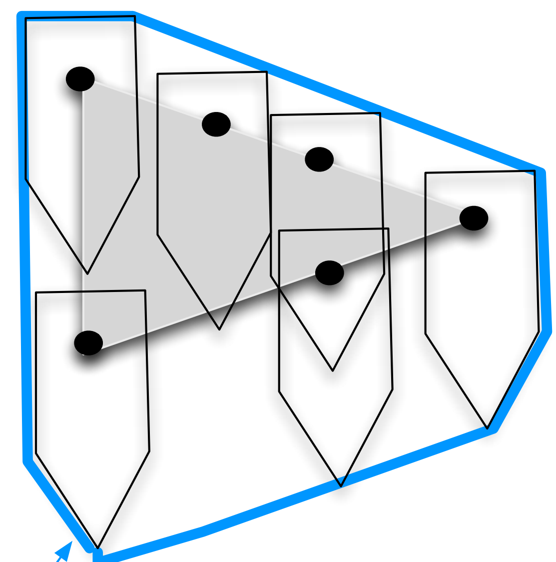


C-obstacle corresponding to O

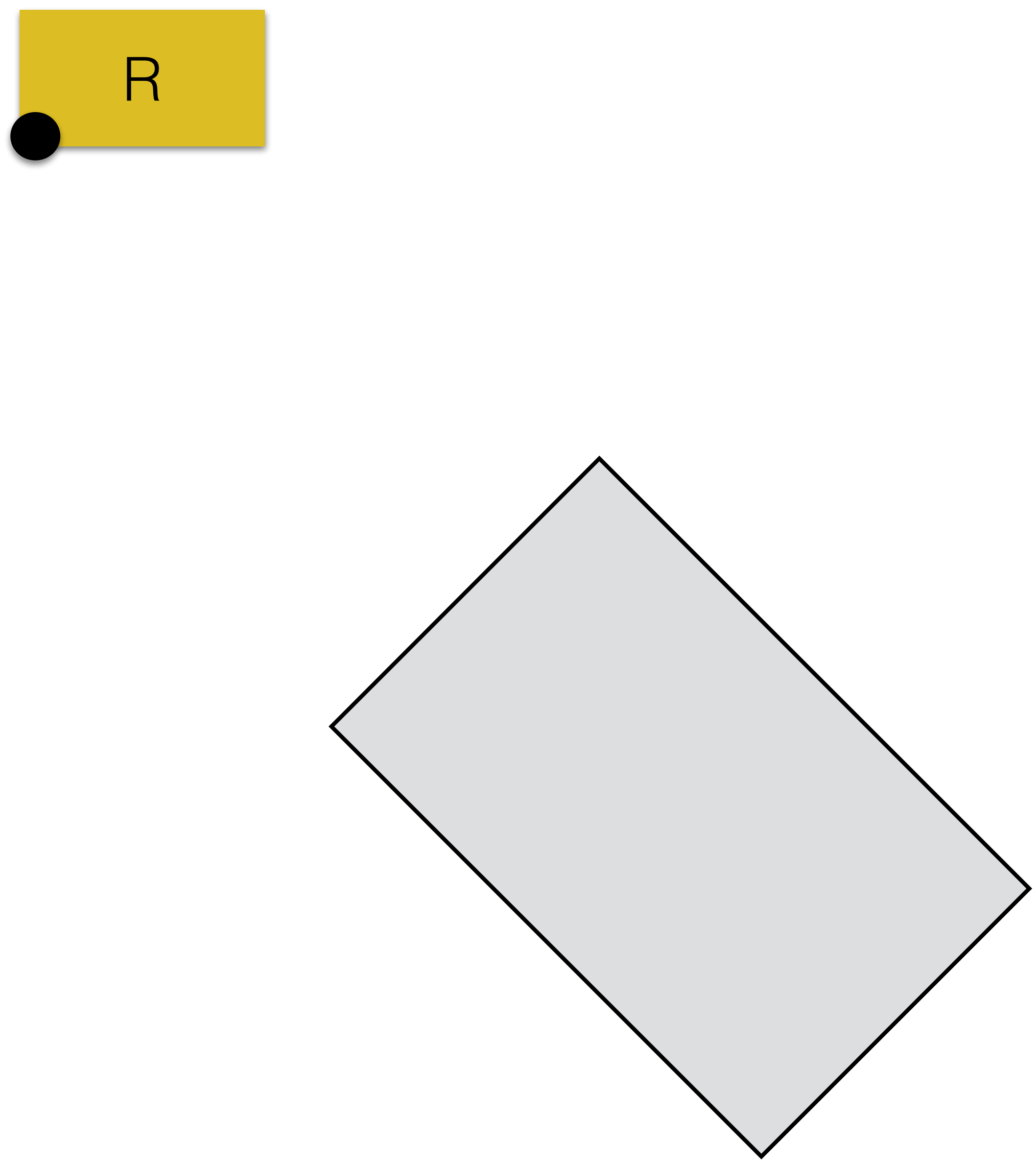
Slide so that centerpoint of -R traces the edges of obstacle

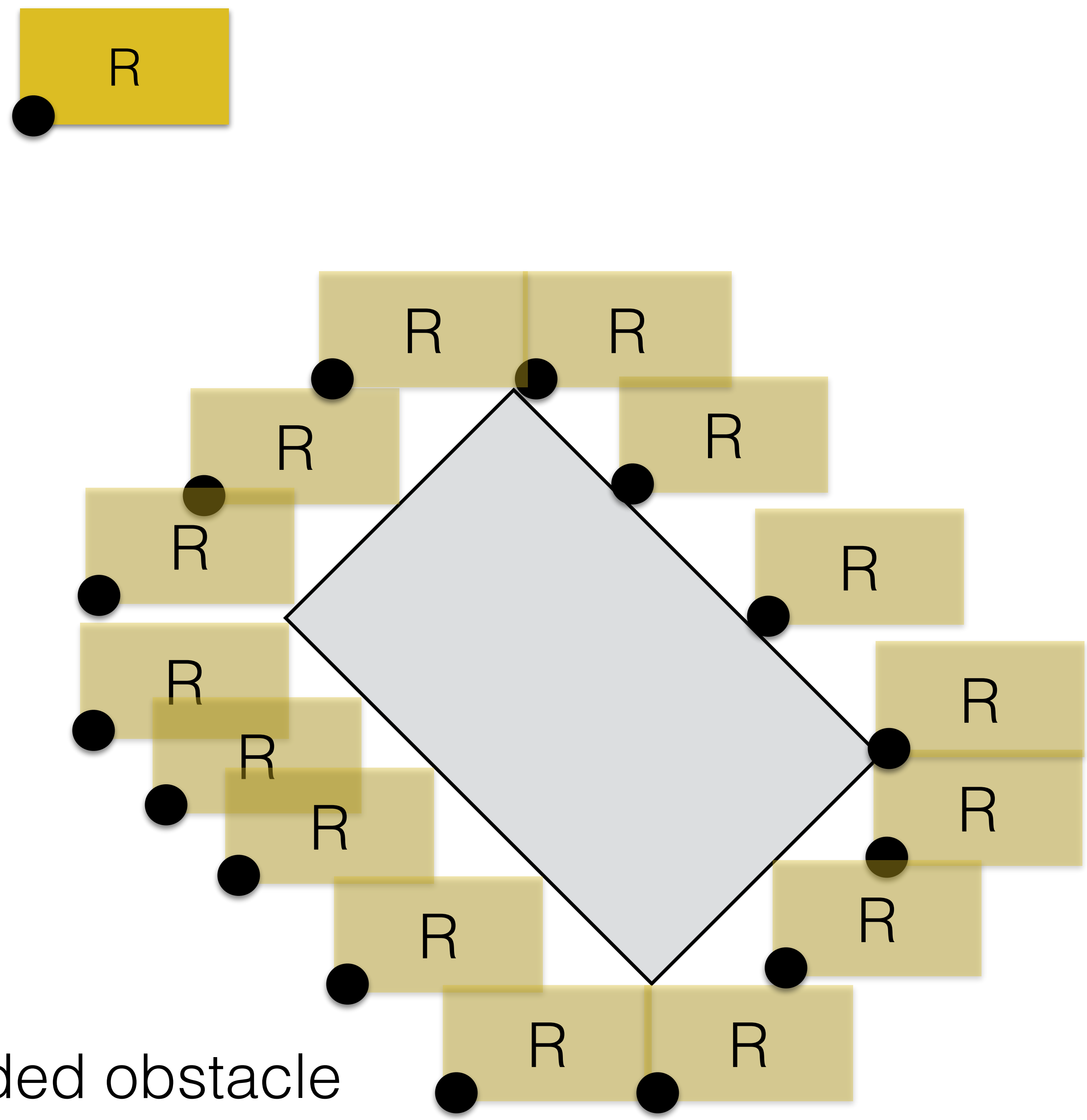
Find $O \oplus -R$



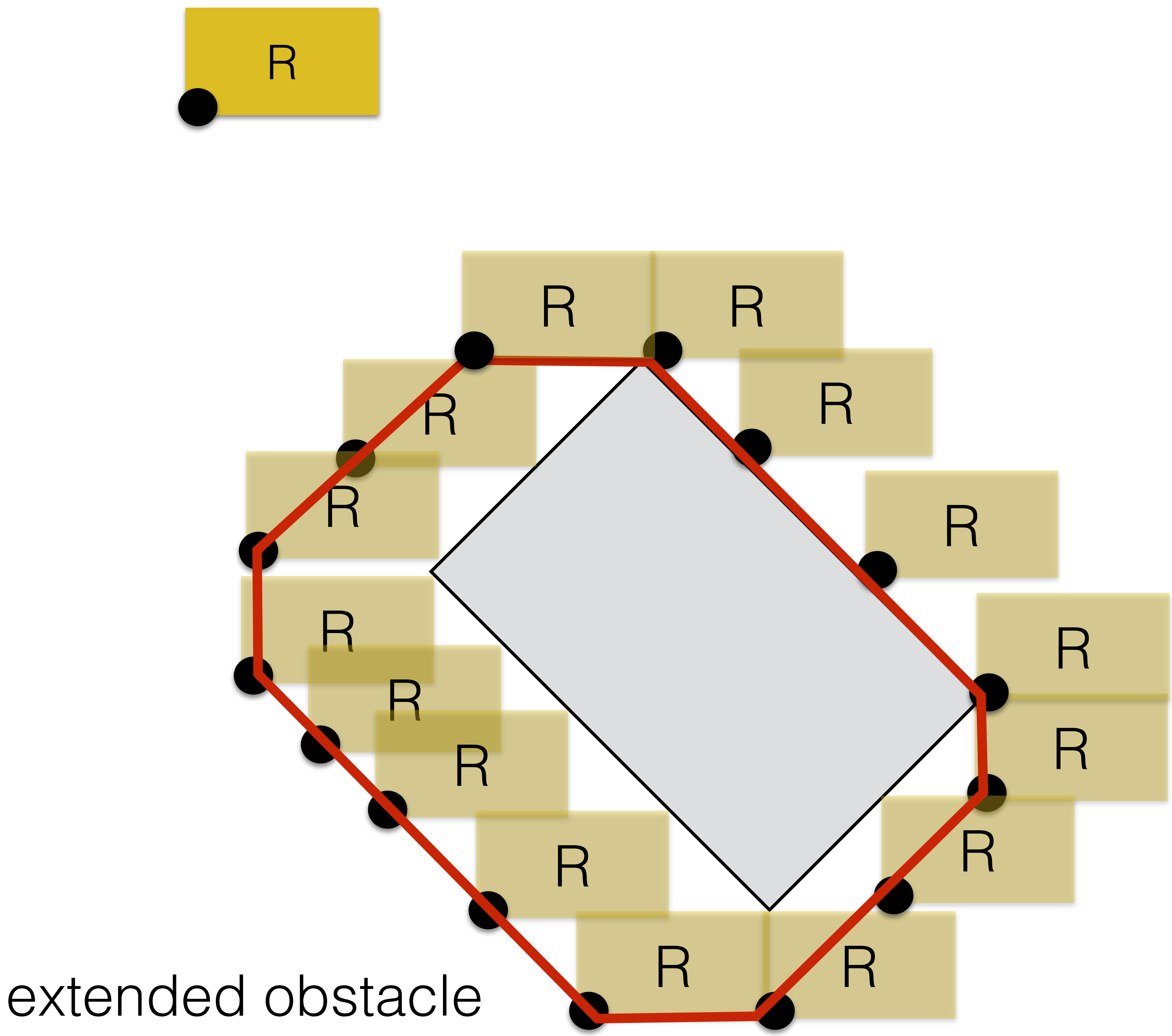


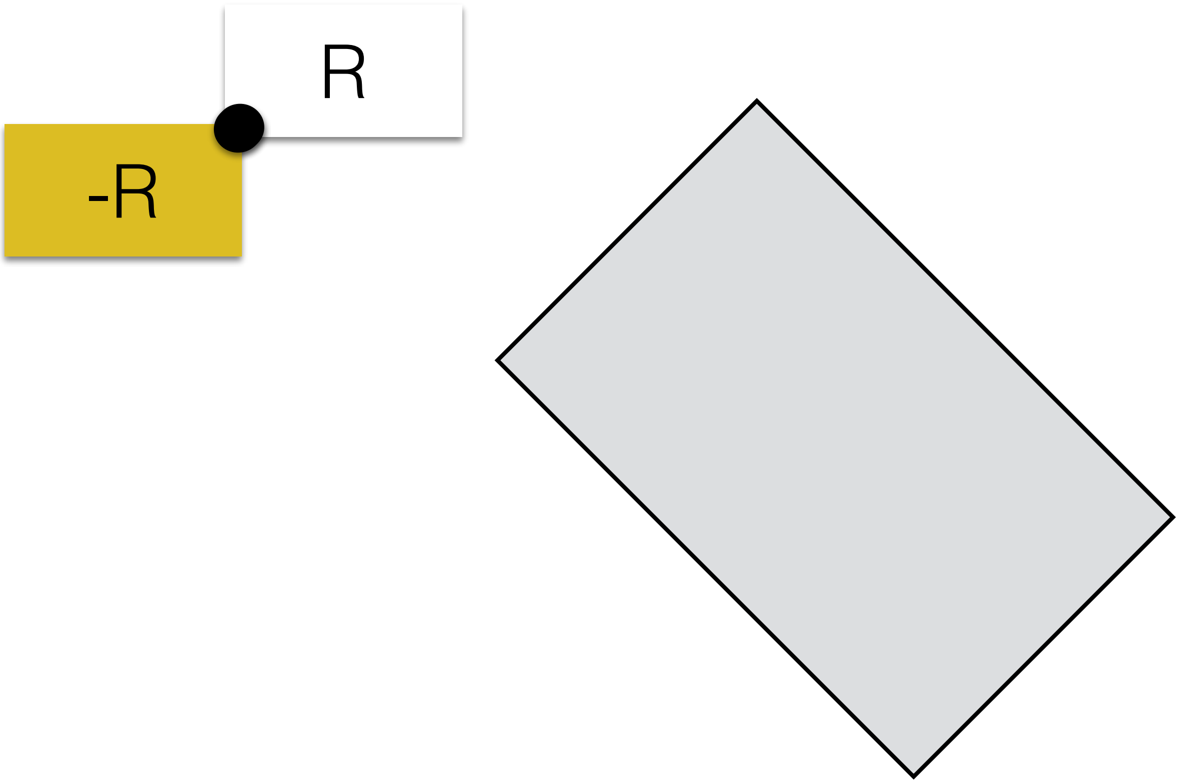
C-obstacle corresponding to O

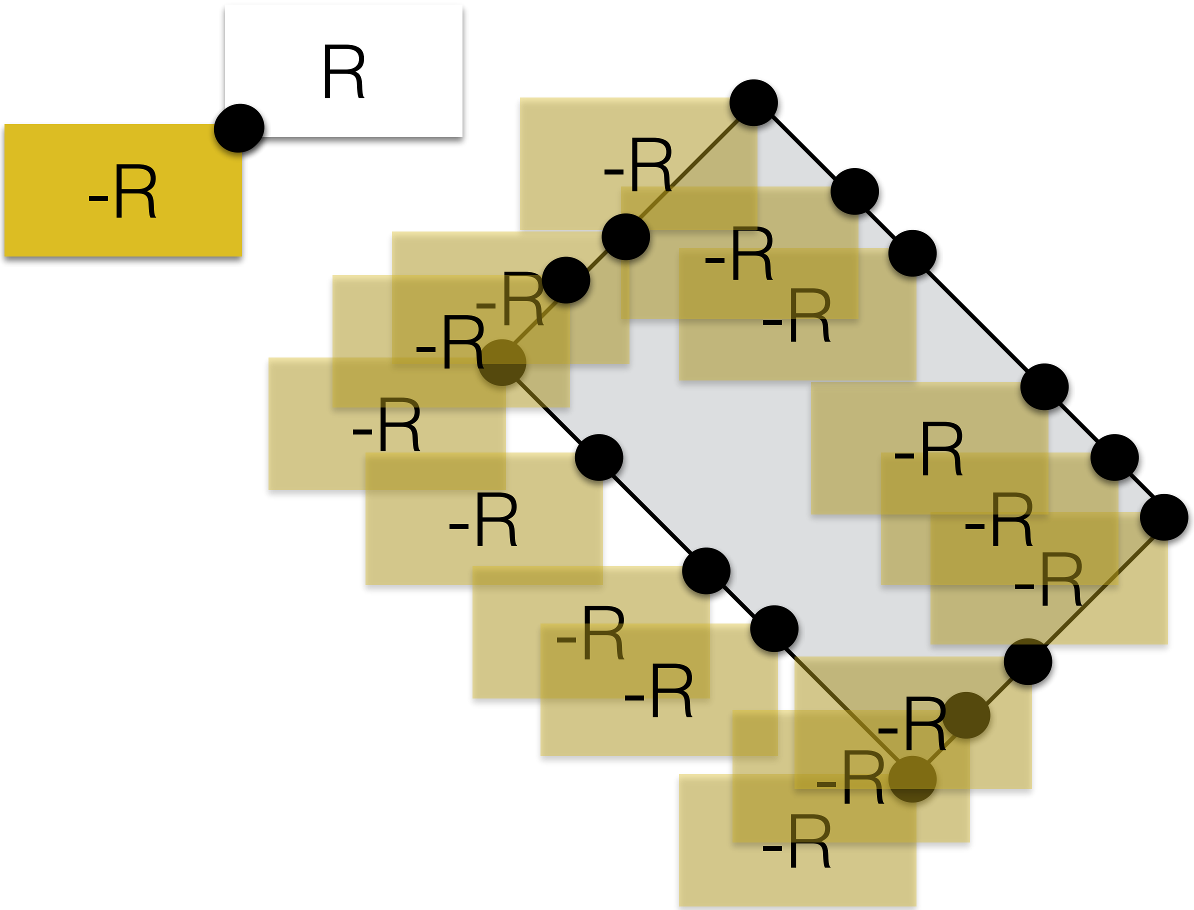

R



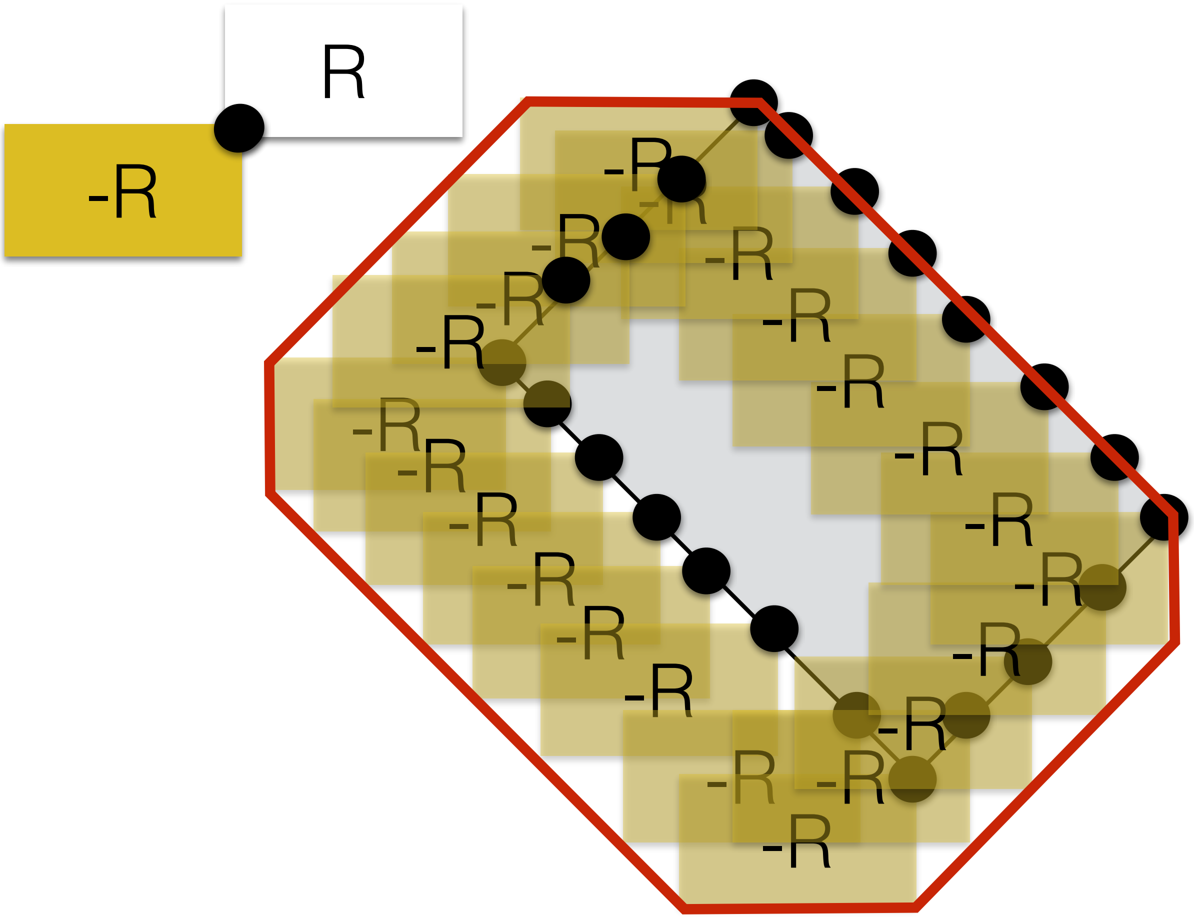
extended obstacle


R
R
R
R
R
R
R
R
R
R
R
R
R
R
R
R
R
extended obstacle

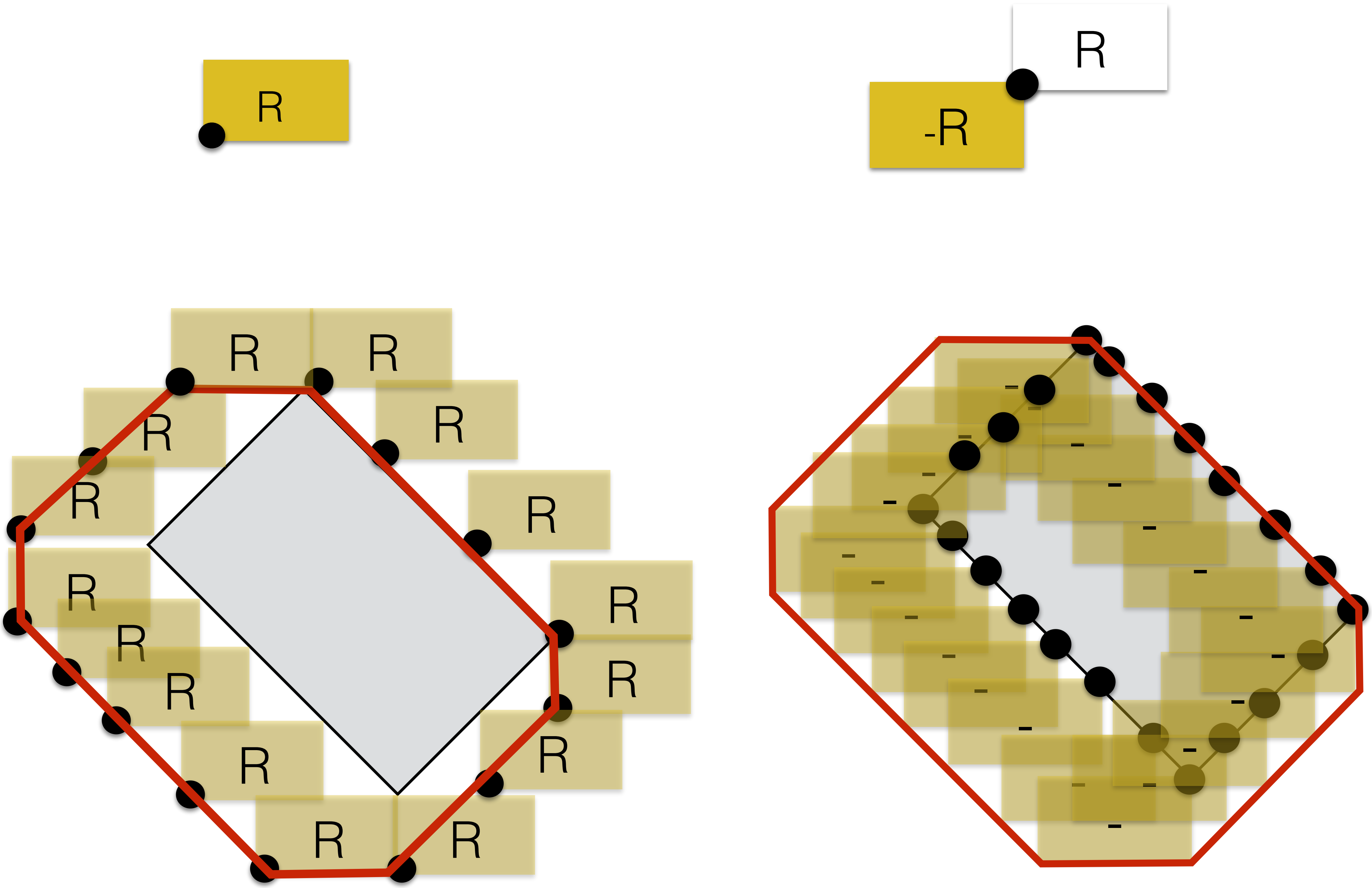

R
-R



R
-R
-R
-R
-R
-R
-R
-R
-R
-R
-R
-R
-R
-R
-R
-R
-R
-R


R
-R

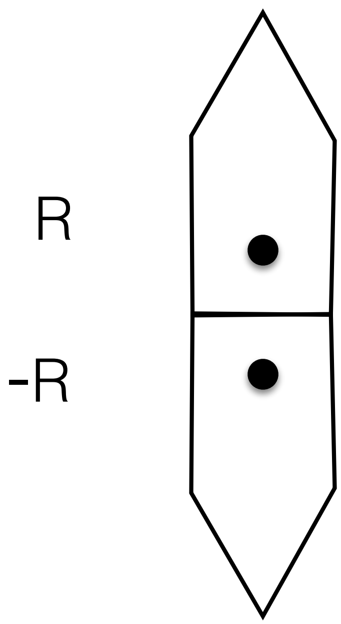
Same!

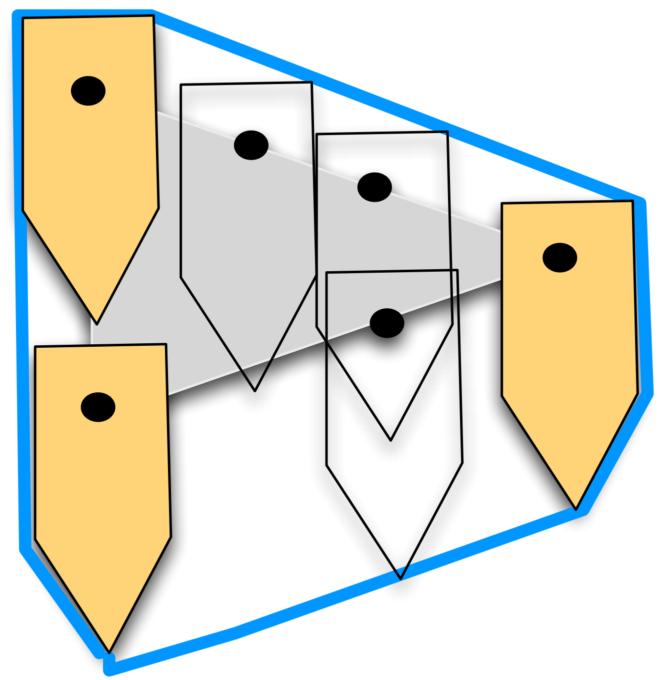


Recap

- We want to compute extended obstacles
- We expressed extended obstacles as Minkowski sum

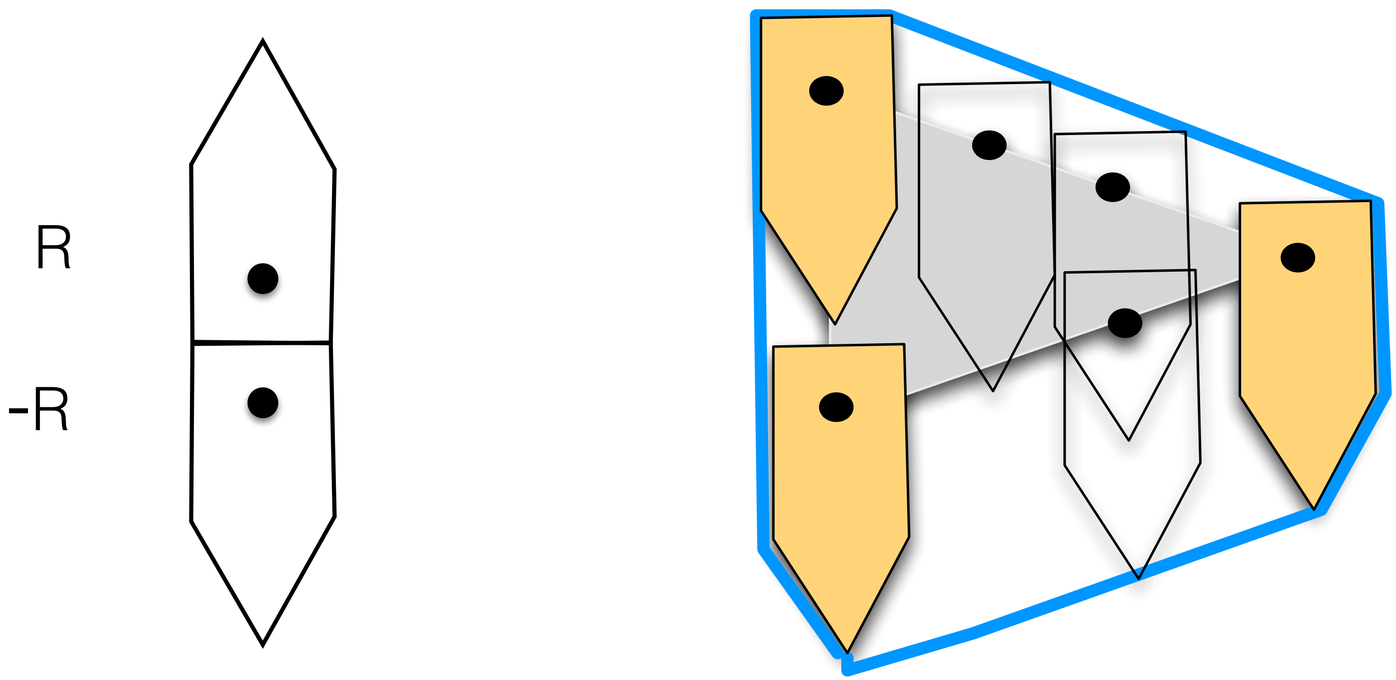
- How do we compute Minkowski sums?





Convex robot with **convex** polygon

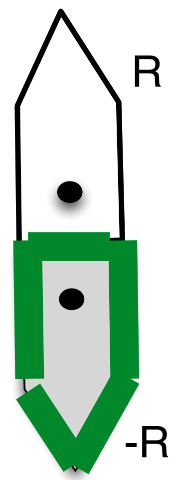
- To compute: Place -R at all vertices of O and compute convex hull

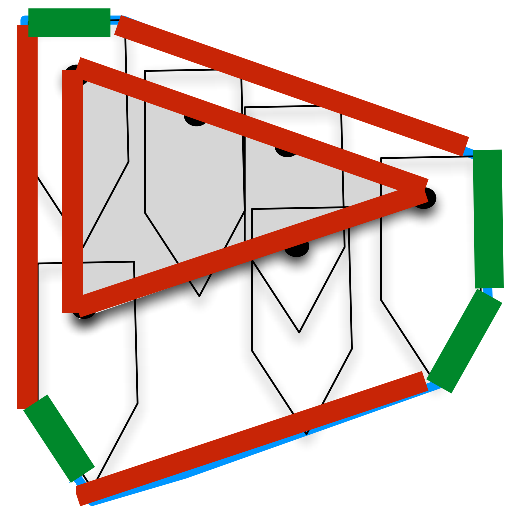


Convex robot with **convex** polygon

- Even better, it is possible to compute in $O(m+n)$ time by walking along the boundaries of R and O

- Each edge in R, O will cause an edge in $O \oplus -R$
- $O \oplus -R$ has O(m+n) edges





parallel edges will cause same edge

Computing extended obstacles: What's known

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 $O(m + n)$ 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 => $O(m \cdot n)$
- non-convex + non-convex polygons
  - Size of Minkowski sum: $O(n^2 \cdot m^2)$

3D

- it gets worse . . .

So far we've considered only translation

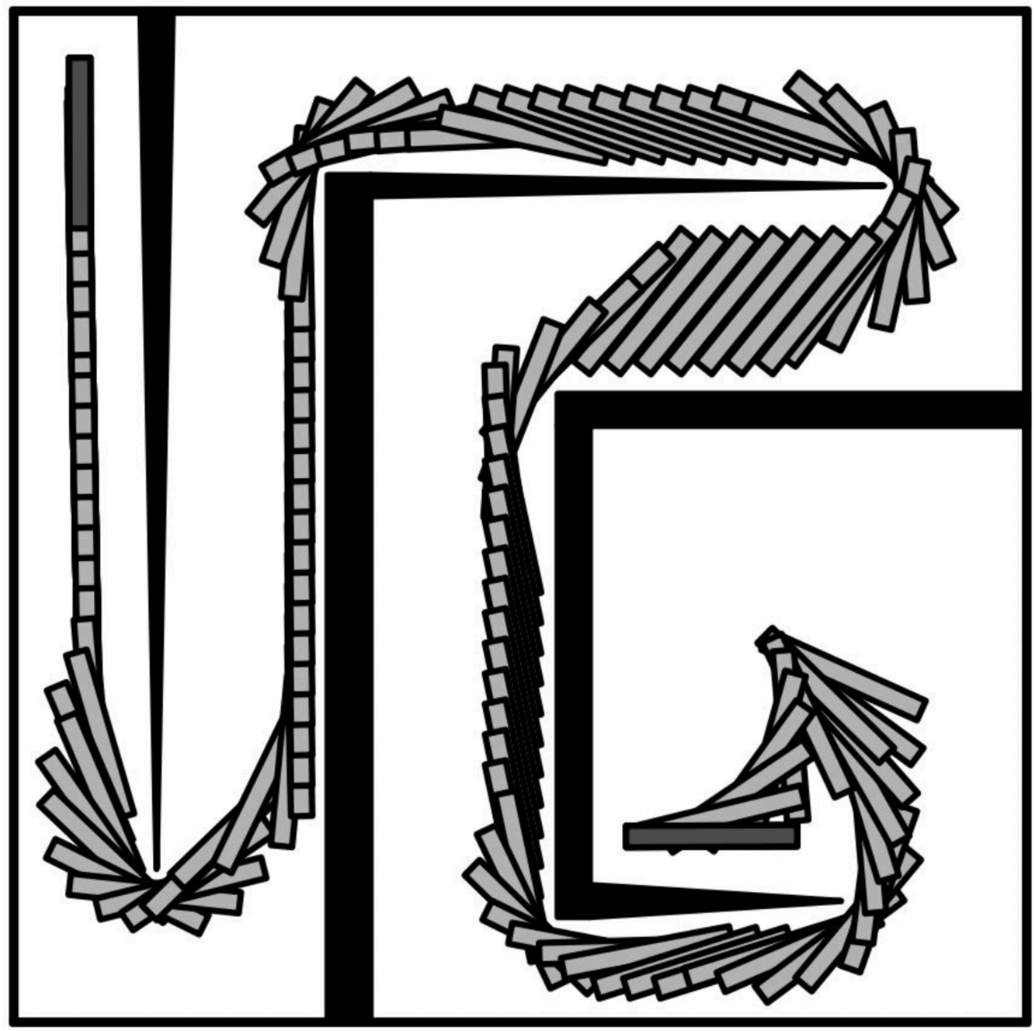
Next: Translation + Rotation

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.

What about Rotating Robots?

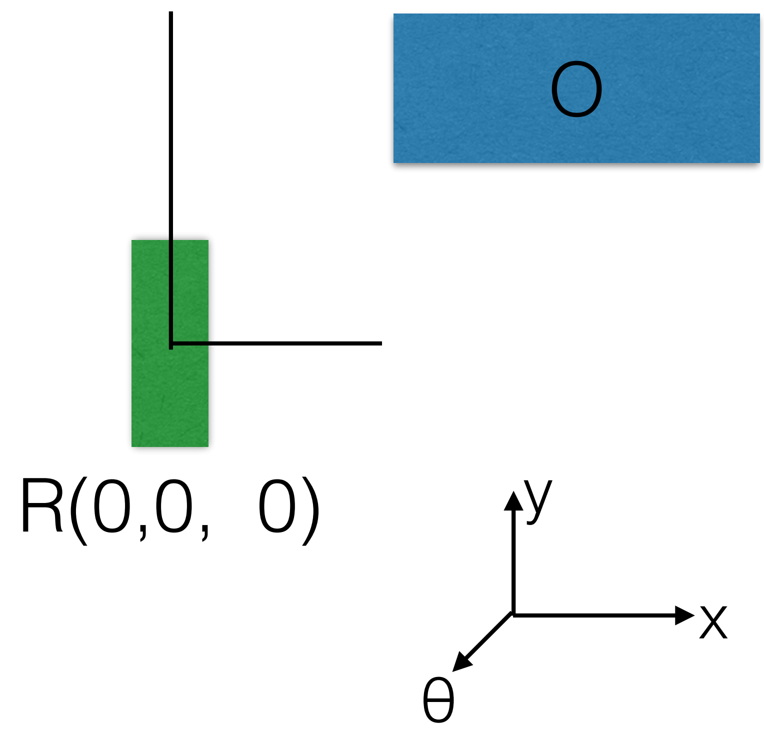
- Rotation may be necessary to complete the task



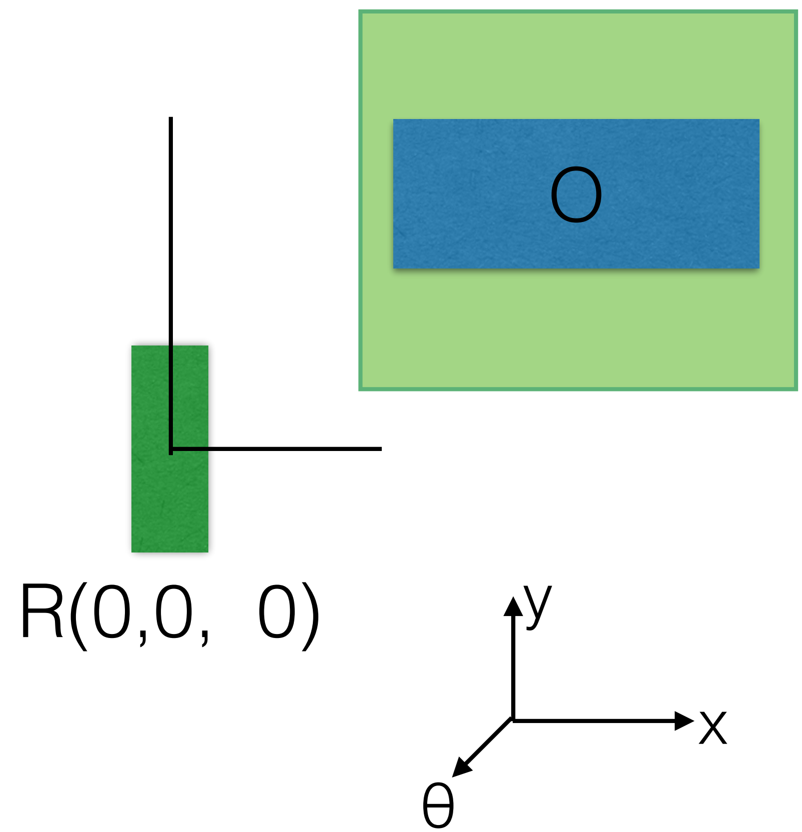
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 13

Polygonal robot in 2D with rotations

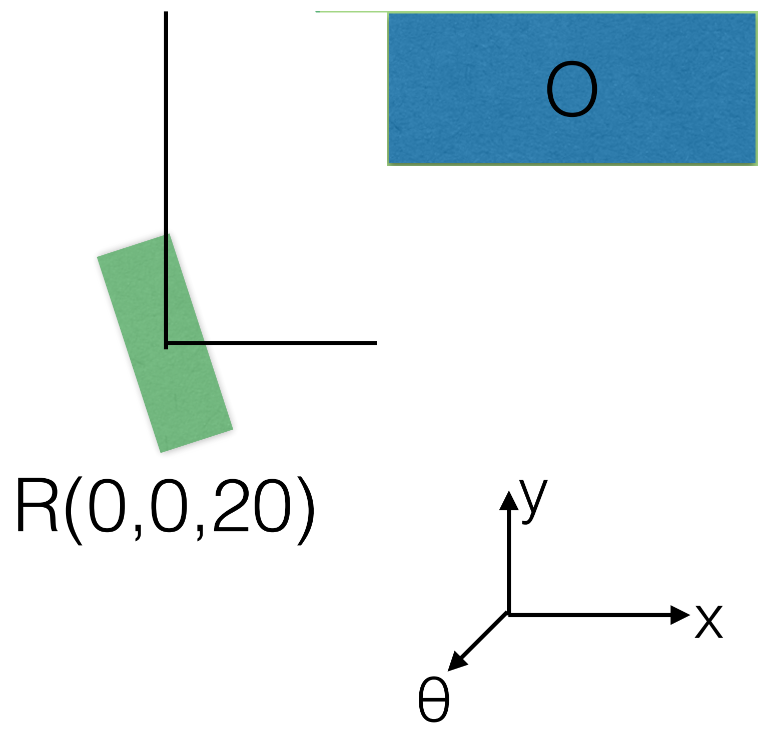
- What does a C-obstacle look like when rotations are allowed?



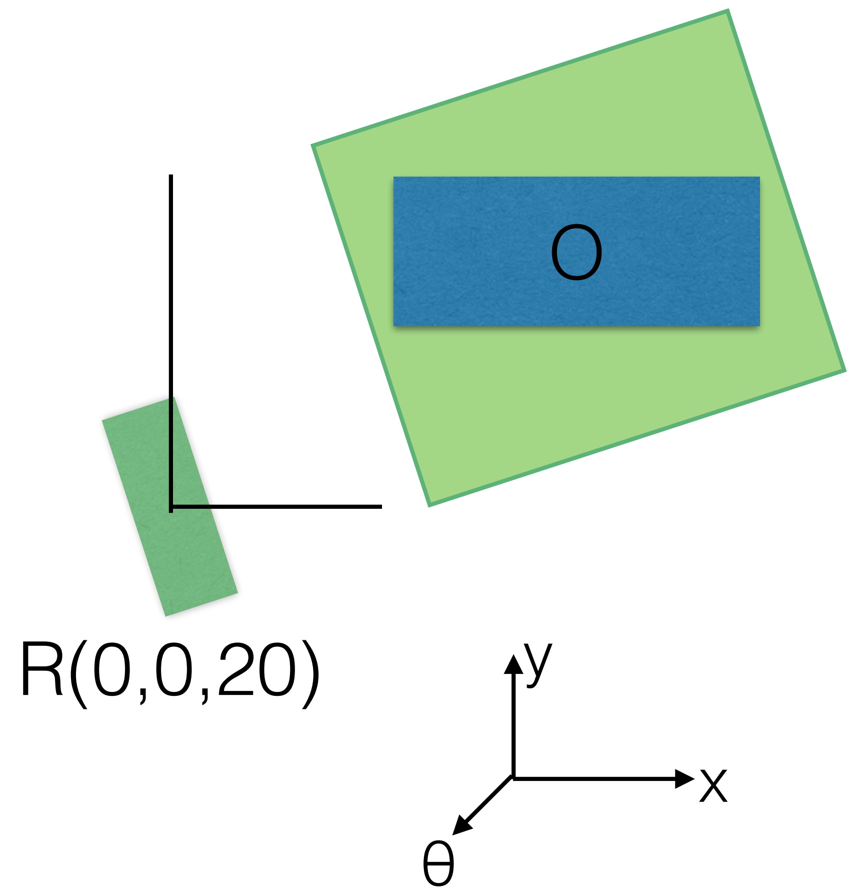
- What does a C-obstacle look like when rotations are allowed?



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

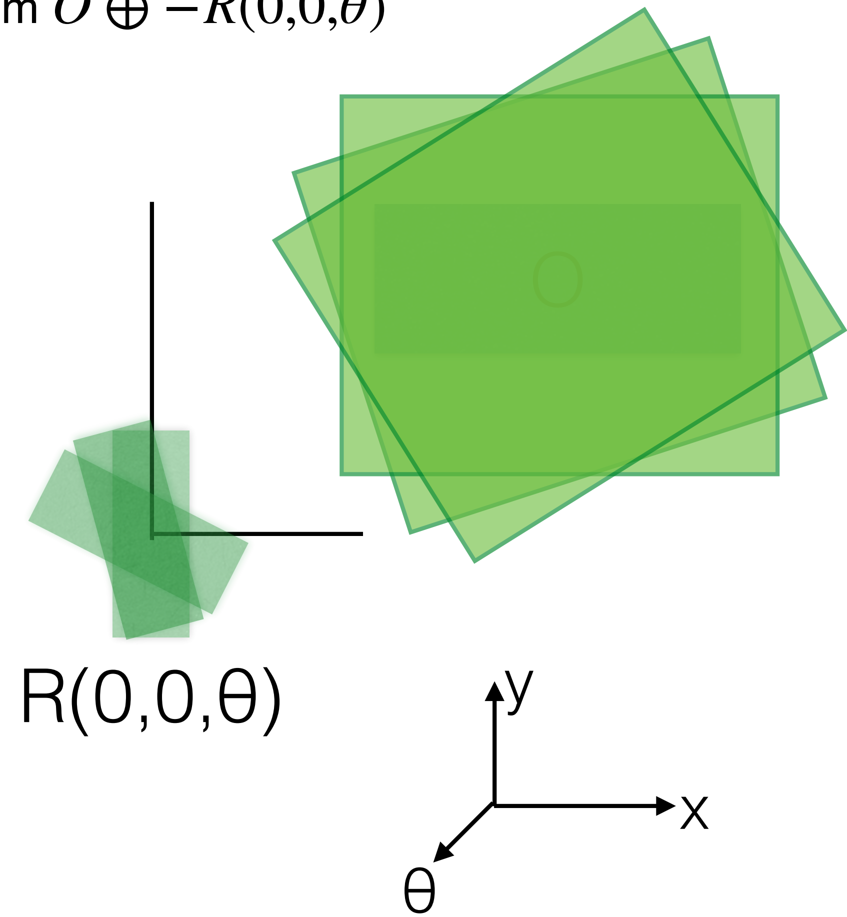


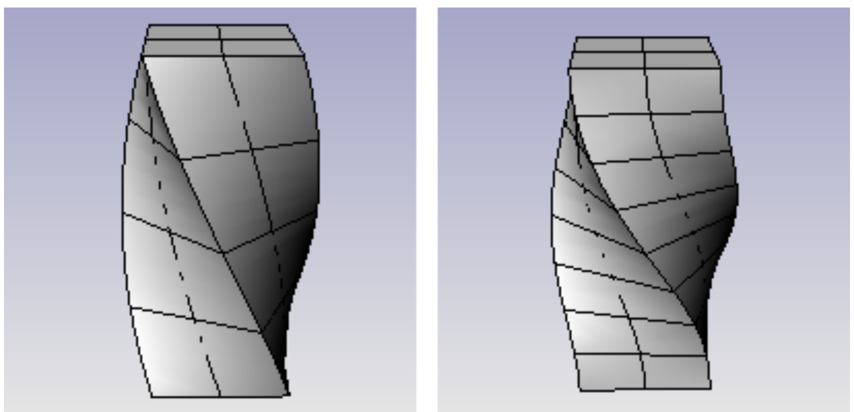
- What does a C-obstacle look like when rotations are allowed?



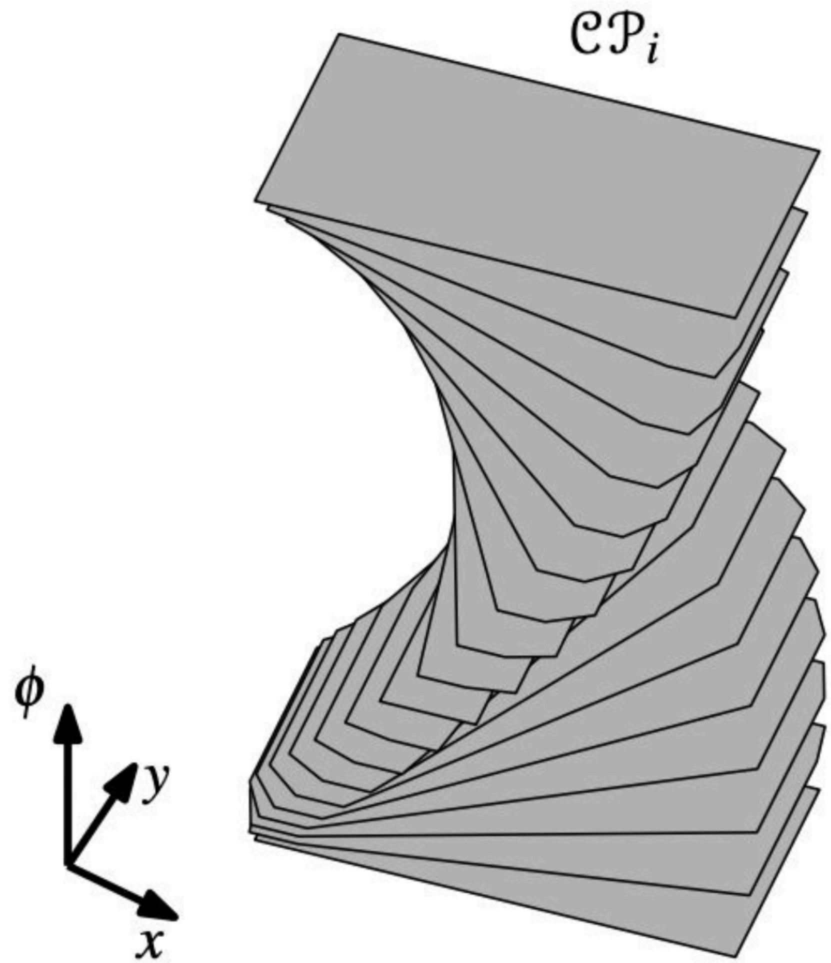
A C-obstacle is a 3D shape, with curved boundaries

- Imagine moving a vertical plane through C-space. Each position of the plane will correspond to a fixed θ.
- Each cross-section of a C-obstacle is a Minkowski sum $O \oplus -R(0{,}0{,}\theta)$
- => twisted pillar





configuration space



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

- Planner:
  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

An idea to approximate this

One possible approximate 3d roadmap

- Discretize rotation angle and compute a finite number of slices, one for each angle
- For a fixed angle: you got translational motion planning
    - 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 consecutive 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?
    - No, it’s an approximation.

Combinatorial/geometric path planning: Summary

- Compute the free C-space geometrically (= exactly)
- **A geometric planner**
    - Compute extended obstacles and free C-space
    - Compute roadmap of free C-space: trapezoidal decomposition or 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 #dif
        - A complete planner in 3D runs in $O(2^{n^{\#dof}})$