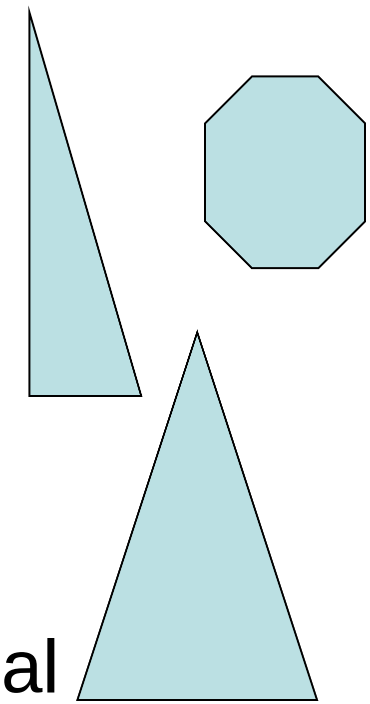
Motion Planning

Thanks to

Piotr Indyk

Piano Mover's Problem

- Given:
  - A set of obstacles
  - The initial position of a robot
  - The final position of a robot
- Goal: find a path that
  - Moves the robot from the initial to final position
  - Avoids the obstacles (at all times)

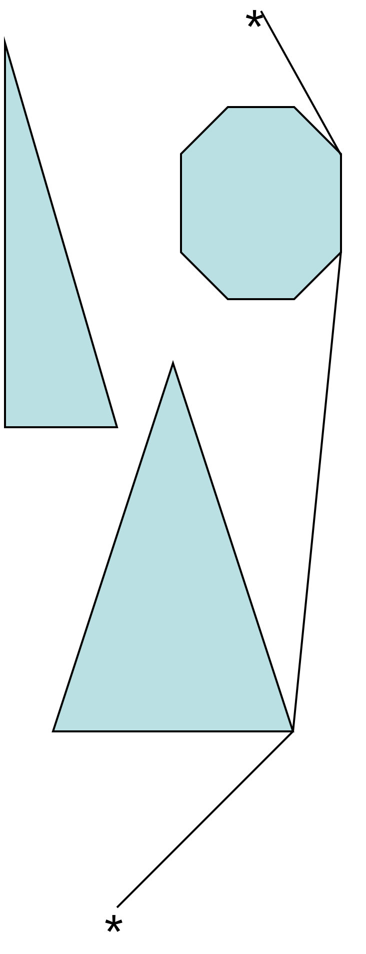


Basic notions

- Work space – the space with obstacles
- Configuration space:
  - Describes the robot’s position
- Forbidden space = positions in which robot collides with an obstacle
- Free space: the rest
- Collision-free path = path in the free part of configuration space

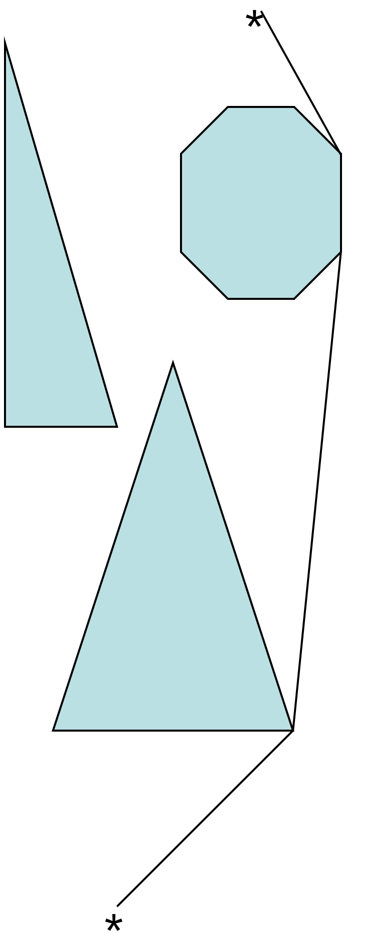
Point case

- Assume robot is a point
- Then work space=configuration space
- Free space = Work Space minus the obstacles



Point case – General Algorithm

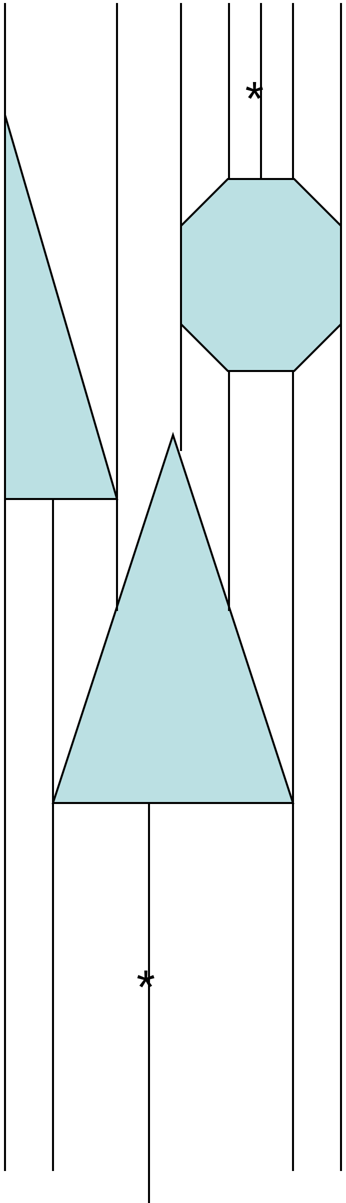
- Construct a data structure ROADMAP to represent the free space
- Given any start and goal positions use ROADMAP to decide whether collision free path is possible



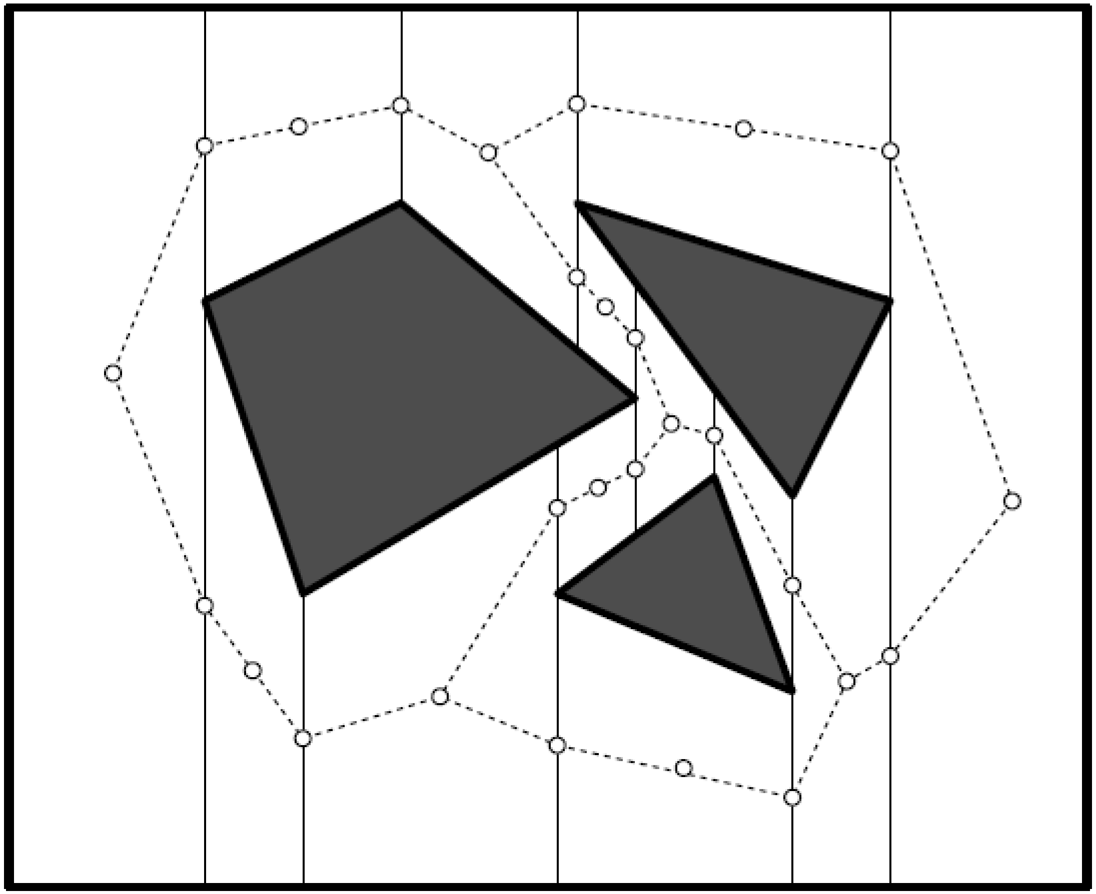
Finding a path

- ROADMAP:
- Compute the trapezoidal map to represent the free space
- Place a node
  - At center of each trapezoid
  - Of each edge of the trapezoid
- Put edges between the vertices in the same trapezoids.
- Path finding=BFS in the roadmap

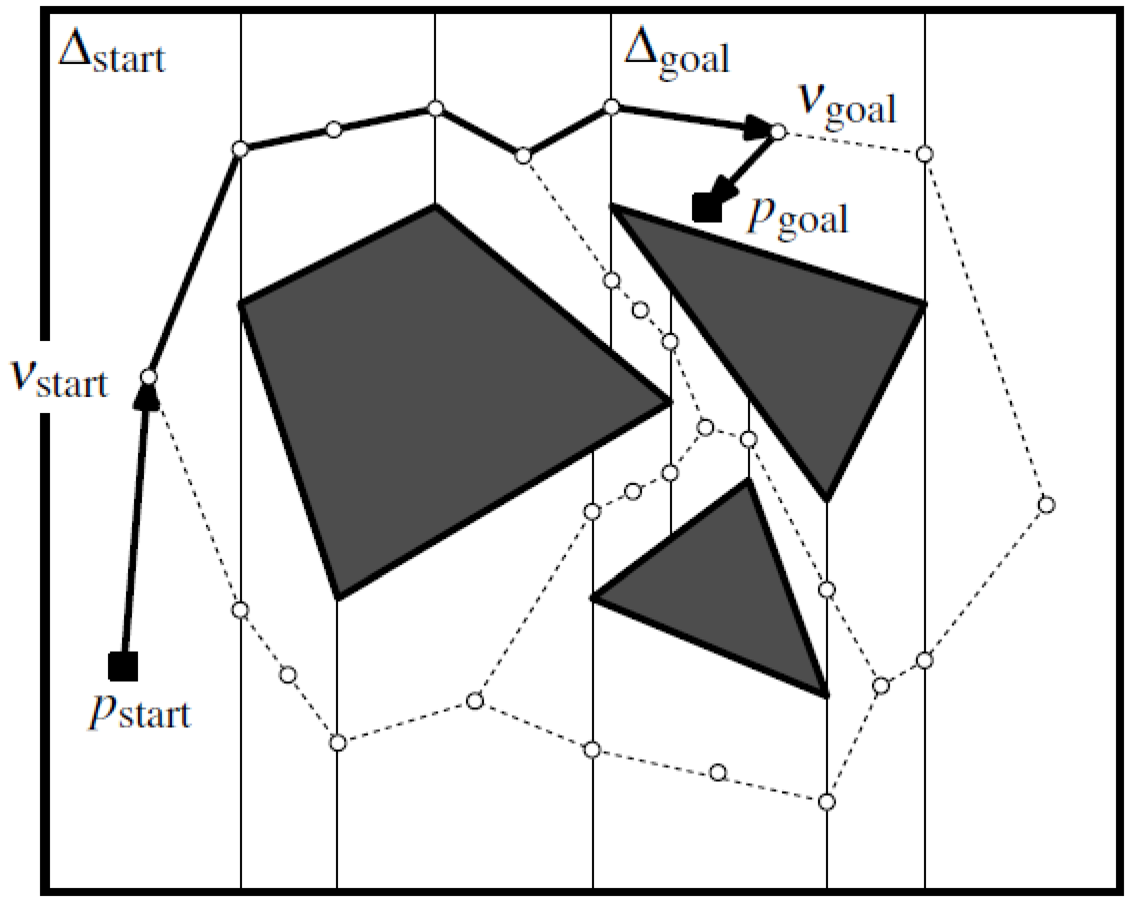
Note – the size of the roadmap is linear, but the path is probably not the shortest.



Roadmap



Path in the roadmap via BFS

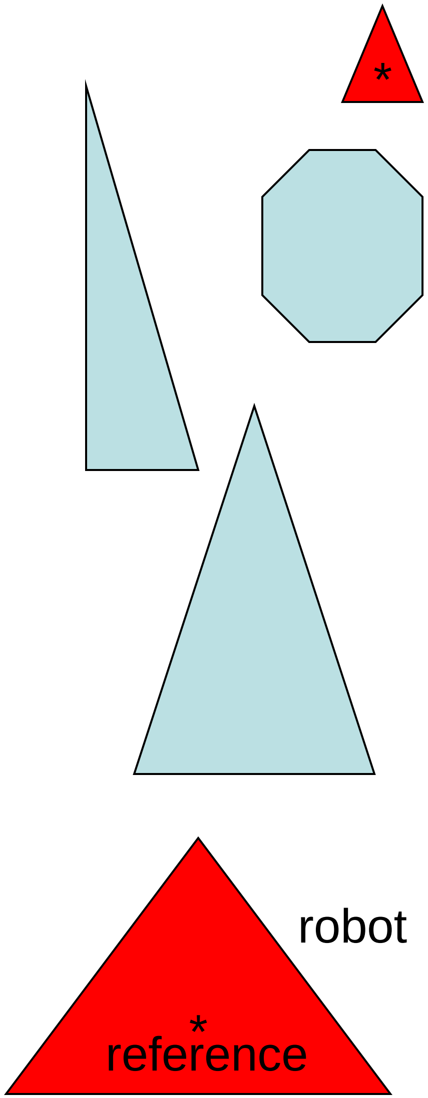


Complexity

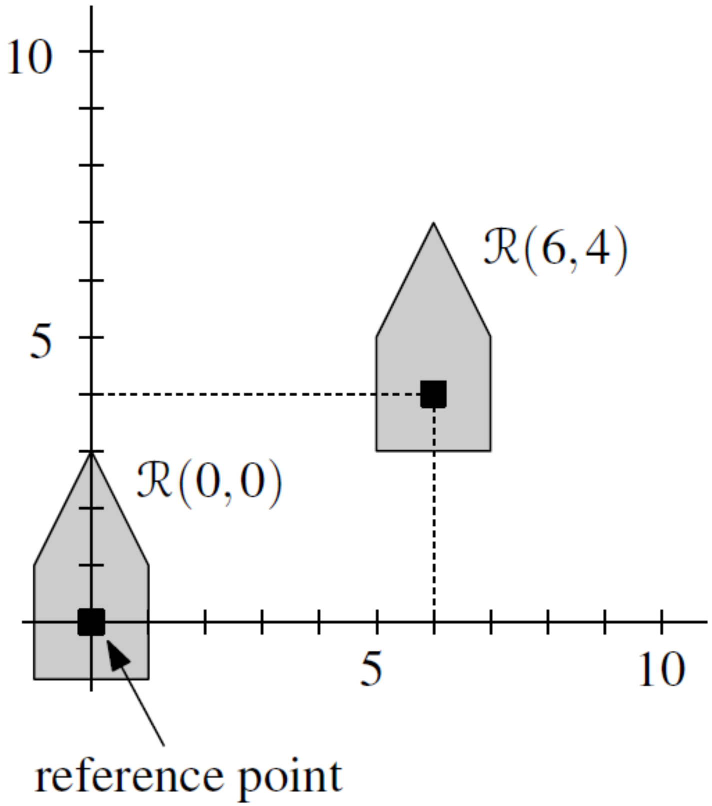
- Build Road Map: O(n logn) time
  - Trapezoidal Map of n segments: O(n log n) time
  - O(n) trapezoids, O(n) vertices
  - Add edges to roadmap takes O(n) time
- Collision Free path: O(n) time
  - Find start and goal trapezoids O(log n)
  - BFS takes O(n) time

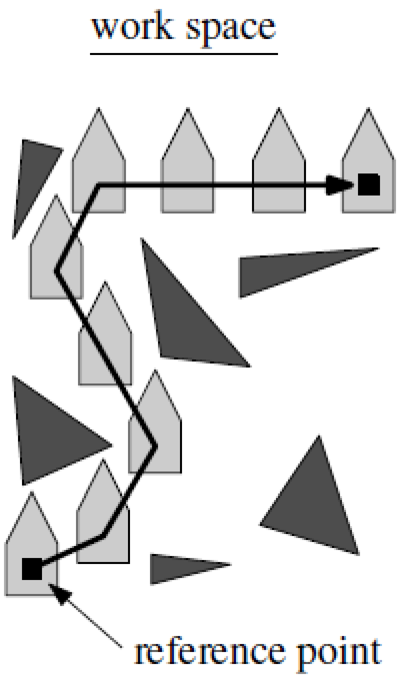
Non-point robots

- Assume a convex robot
- Assume each obstacle is convex (by triangulating the obstacles)
- We specify a point on the robot, called its **reference.**
- We specify the position of the robot by specifying the location of the reference

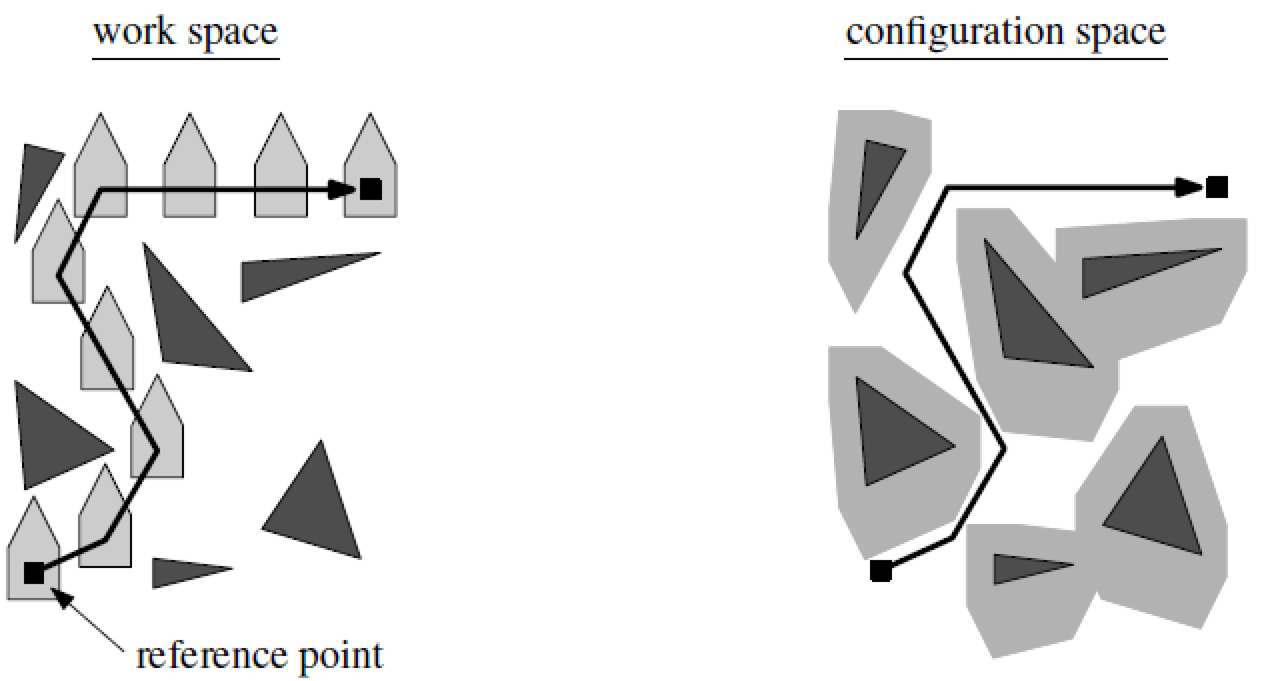


Specifying location of robot



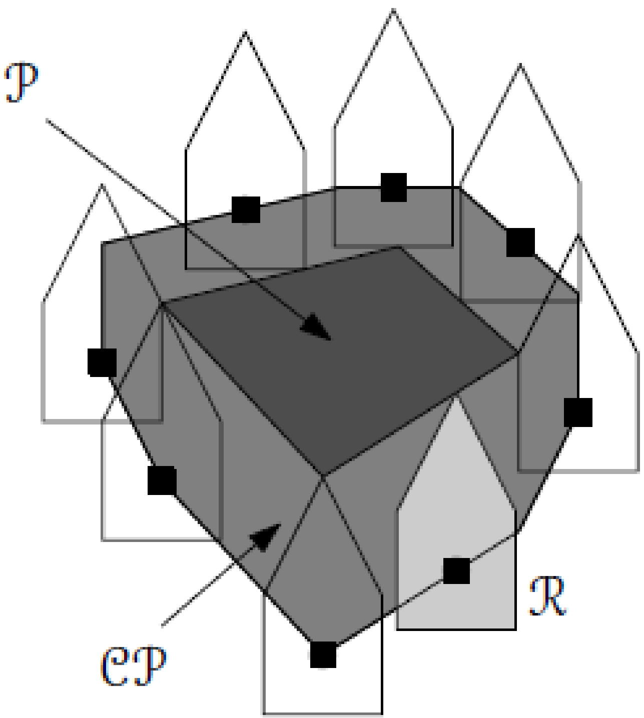


Collision Free Path



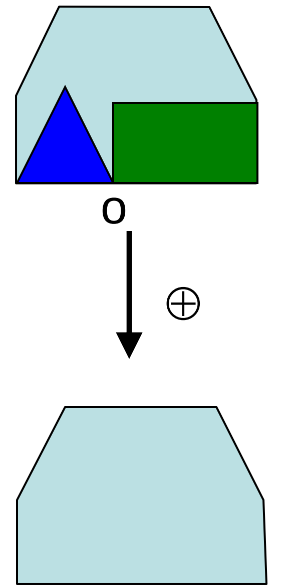
Non-point robots - cont

- C-obstacle = the set of robot positions which overlap an obstacle
- Free space: workspace minus C-obstacles
- Given a robot and obstacles, how to calculate C-obstacles ?

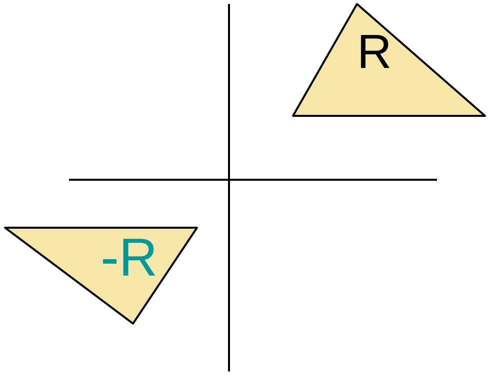


Minkowski Sum

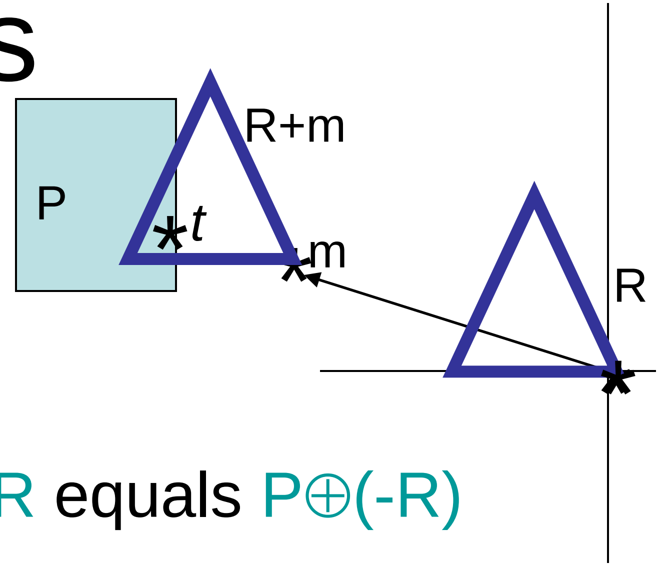
- Minkowski Sum of two sets P and Q is defined as $P \oplus Q = \{p+q: p \in P, q \in Q\}$



R vs (-R)



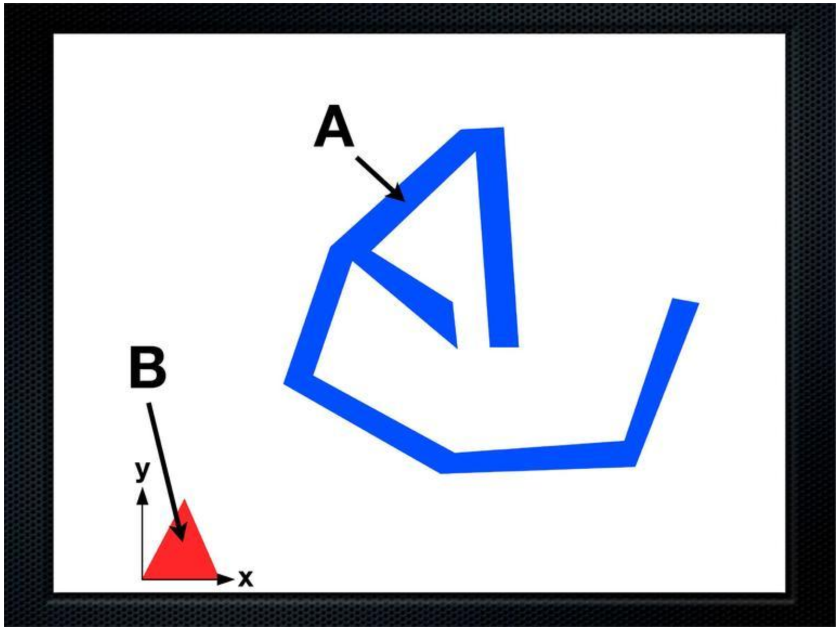
C-obstacles

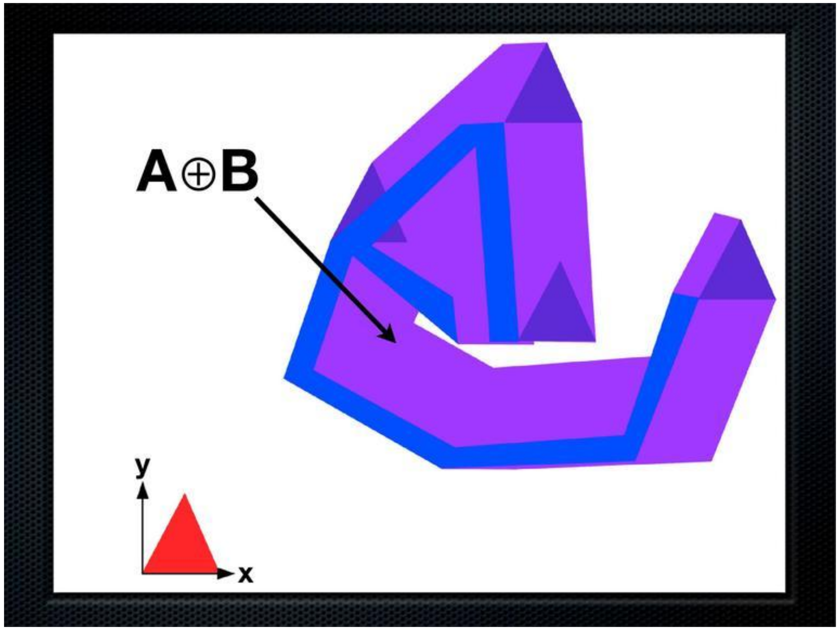


- Thm: The C-obstacle of P and robot R equals $P \oplus (-R)$
- Proof:
  - Assume R collides with P at position m. We want to show that $m \in P \oplus (-R)$
  - Consider $t \in (R+m) \cap P$
  - Then $t - m \in R \rightarrow -t+m \in -R$
  - Since $t \in P$, we have $m \in P \oplus (-R)$
- Reverse direction is similar

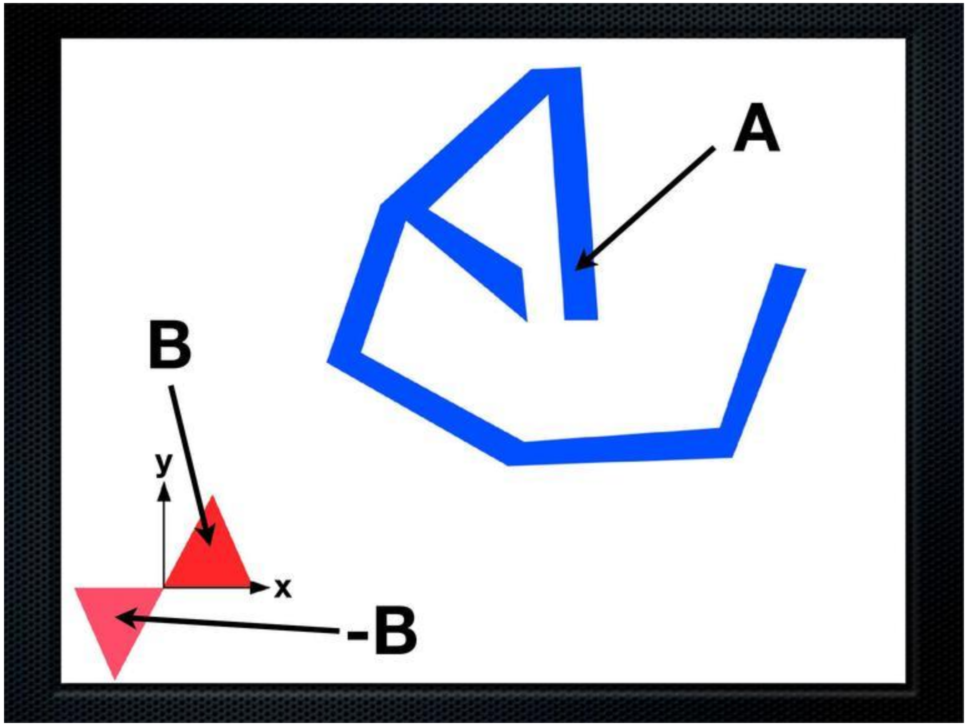
Minkowski Sum

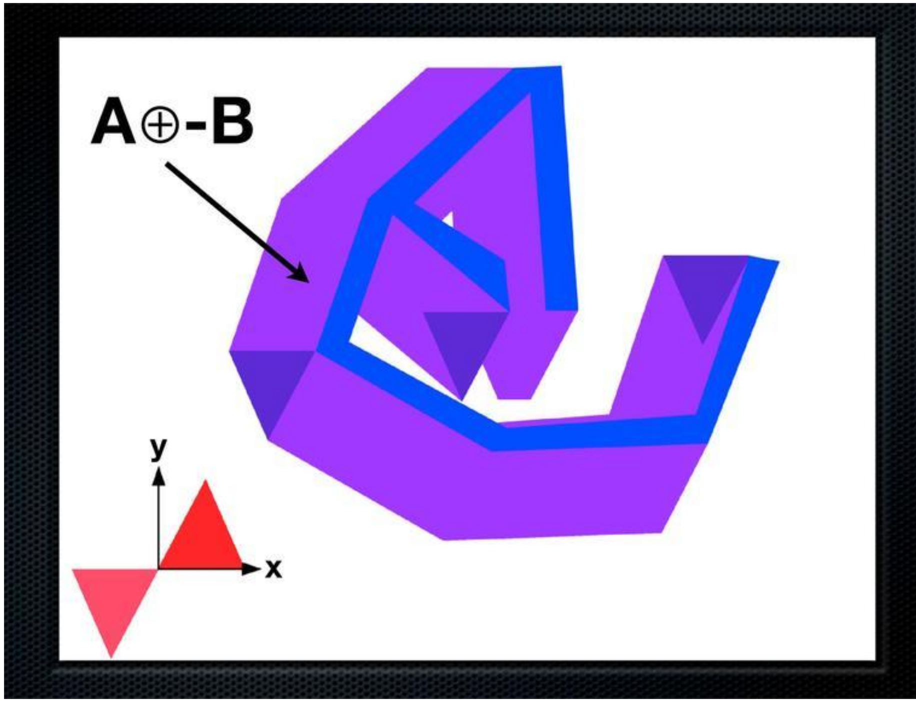
- $A \oplus B = \{a+b : a \in A,\ b \in B\}$



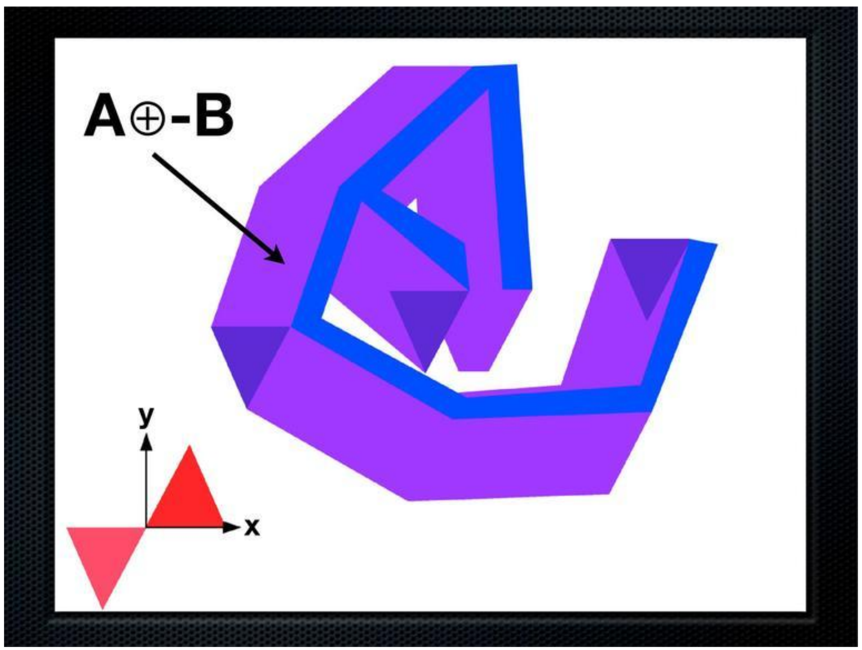


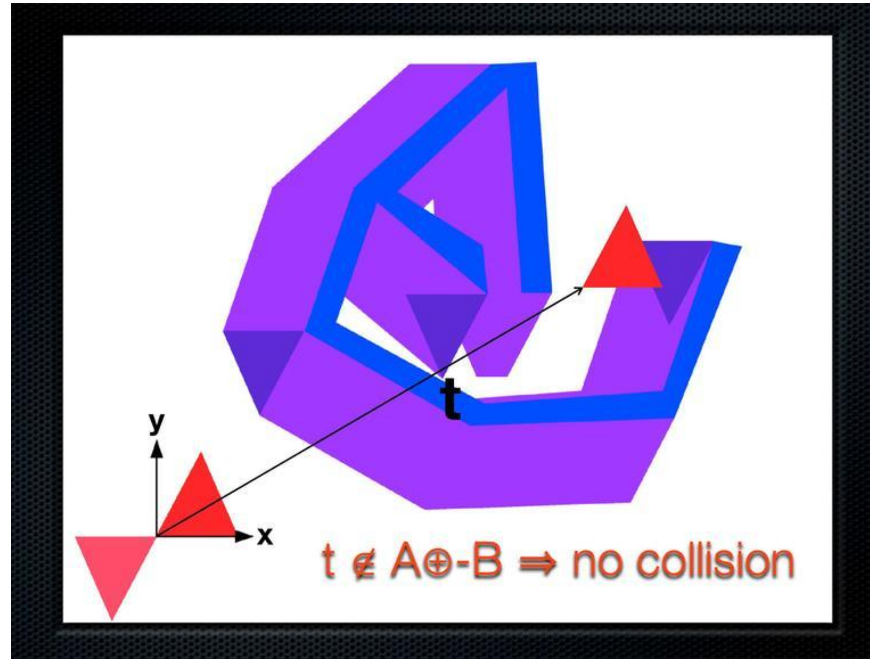
A⊕-B

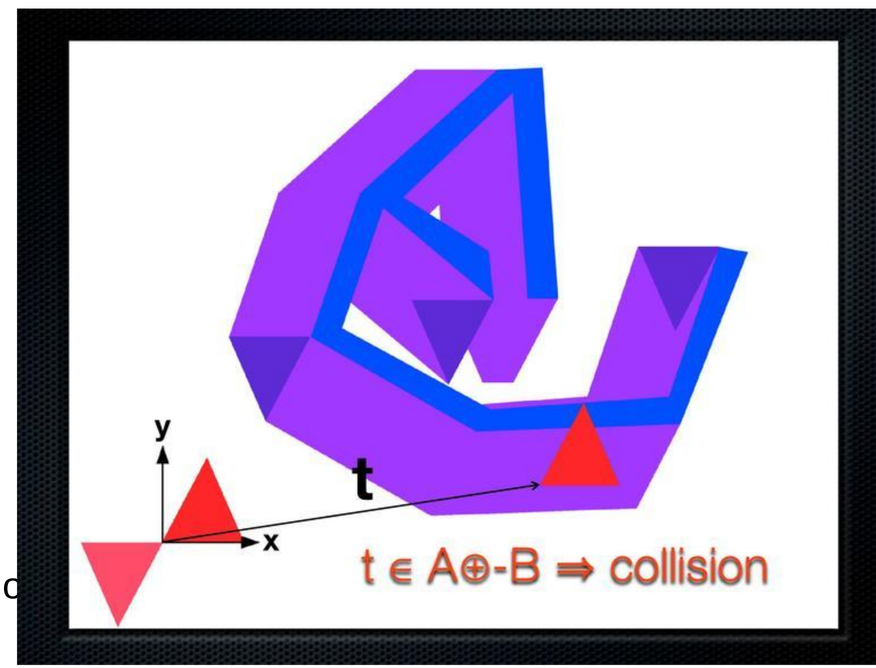




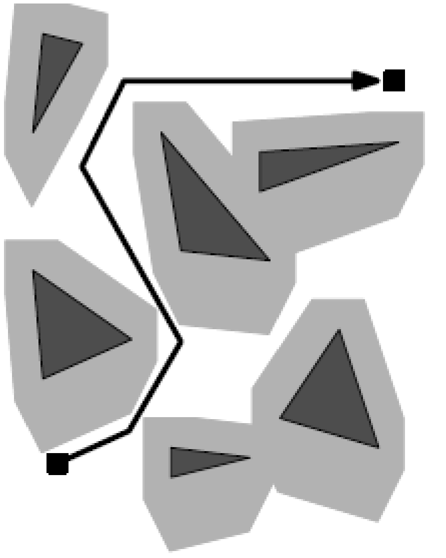
C-obstacle of A and robot B equals A⊕(-B)







Algorithm outline



- Find C-Obstacles
- Create trapezoid map for union of all C-obstacles

- Efficiency depends:
  - Computation time of C-Obstacles
  - Computation of trapezoidal map: O(n log n) where is n is complexity of union of all C-obstacles (number of edges)

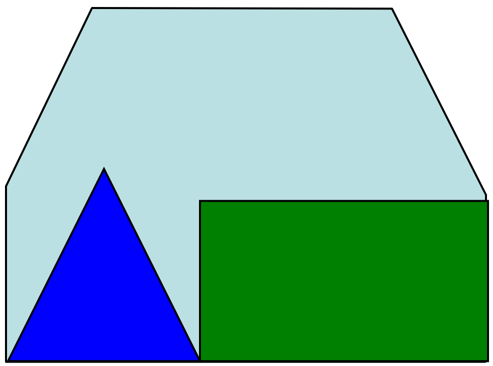
I. Properties of $P \oplus R$

- Thm: If P,R convex, then $P \oplus R$ is convex:
- Proof:
  - Consider $t_1,t_2 \in P \oplus R$. We know $t_i=p_i+r_i$ for $p_i \in P,\ r_i \in R$
  - P,Q convex: $\lambda p_1+(1-\lambda)p_2 \in P,\ \lambda r_1+(1-\lambda)r_2 \in R$
  - Therefore:

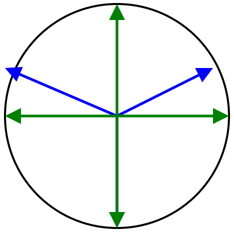
$$\lambda t_1+(1-\lambda)t_2 = \lambda(p_1+r_1) + (1-\lambda)(p_2+r_2) \in P \oplus R$$

II. Properties of P⊕R

A point $p \in$ Q is **extreme** (I.e. corner of *Q*) if there is some vector (direction) *d* such that $p*d = max \{ q*d \mid q \in Q\}$



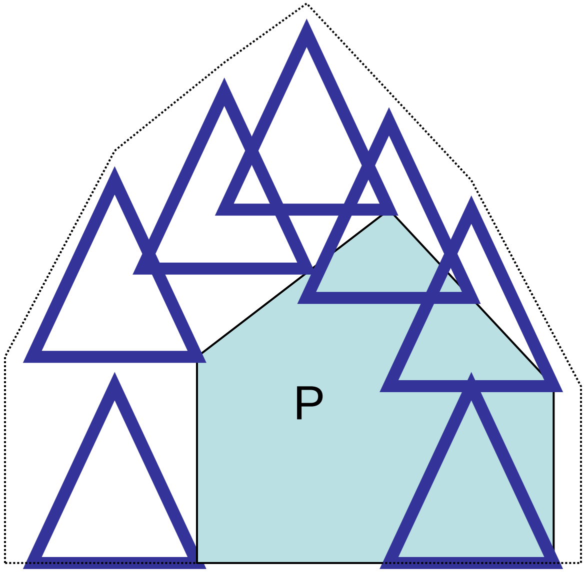
- Observation: an extreme point of P⊕R in direction d is a sum of extreme points of P and R in direction d



- Simple algorithm – convex hull

III. Properties of P⊕R

- Theorem: If P, R convex and has m and n edges then P⊕R has at most n+m edges.
- Intuition: Each edge of P⊕R is parallel to either an edge of P or an edge of R. No edge of P,R contributes more than once.

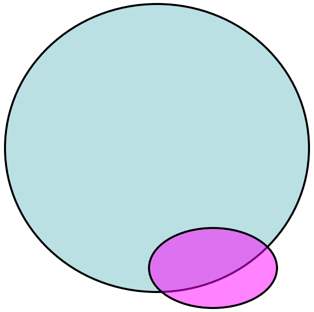


- Implications:
- Compute a C-obstacle in O(n+m) time
- Each C-obstacle has complexity O(n+m)
  - Is this enough?

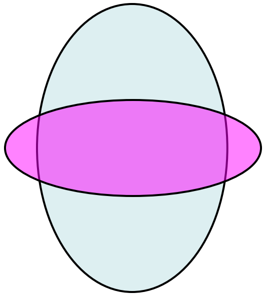
Pseudodisc Pairs

- O_1 and O_2 are Pseudodiscs if both O_1-O_2 and O_2-O_1 are connected
- I,e, at most two proper intersections of boundaries

Yes

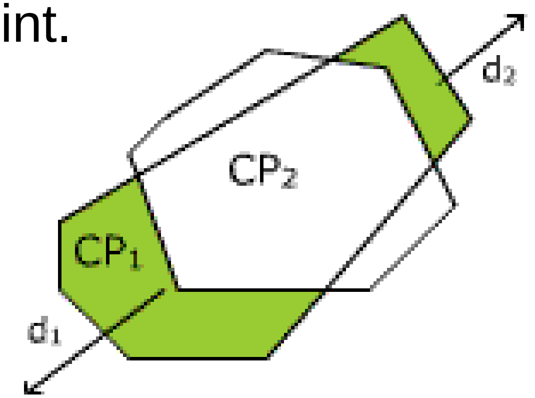


No



- Note: Pseudodiscs describes how **TWO** objects interact. Not used to describe one object.

Minkowski sums are pseudodiscs

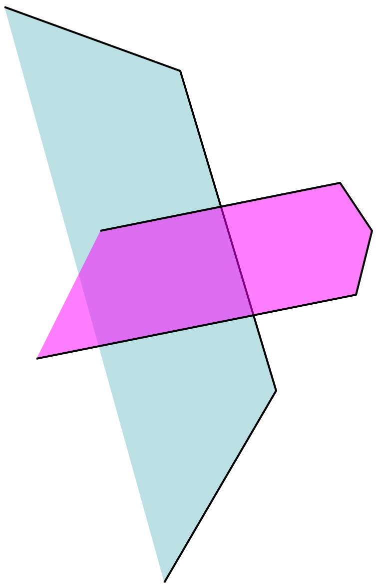


- Thm: If P_1, P_2,R, are convex and P_1 and P_2 are disjoint. Then $CP_1=P_1\oplus R$ and $CP_2=P_2\oplus R$ are pseudo-discs.

- Proof by contradiction:
- Suppose CP_1-CP_2 is has 2 connected components
  - CP_1 is more extreme than CP_2 in two directions d_1 and d_2
  - CP_2 is more extreme than CP_1 in a direction between d_1 and d_2 and in a direction between d_2 and d_1
- By properties of $\oplus$:
  - P_1 is more extreme than P_2 in directions d_1 and d_2
  - P_2 is more extreme than P_1 in a direction between d_1 and d_2 and in a direction between d_2 and d_1
- Configuration impossible for disjoint, convex P_1 ,P_2

Union of pseudo-discs

- Thm: Let $P_1,\ldots,P_k$ be polygons in pseudo-disk positions. Then their union has complexity $|P_1| +\ldots+ |P_k|$



- Proof:
  - Suffices to bound the number of vertices
  - Each vertex either original or induced by intersection
  - Charge each intersection vertex to the next original vertex in the interior of the union
  - Each vertex charged at most twice

Ananlysis: Convex Robot, Convex Obstacles

- Given: Total #edges in *Obstacles=n, Robot=m*
- Compute all C-obstacles in *O(m + n) time*
- Computation time for Trapezoidal Map:
  - If *k* obstacles total complexity of C-obstacles *O(n+mk)*
  - Union of all C-Obstacles has complexity *O(n+mk)*
  - Trapezoidal map computed *O(n+mk* log(n+*mk*))

Analysis:
Convex Robot and Non-convex Obstacles

- Given *complexity of all obstacles=n, robot=m*
- Triangulate obstacles into $T_1,\ldots,T_n$. Time $O(n \log n)$
- Compute $R \oplus T_1, \ldots, R \oplus T_n$ Time $O(n(m+3))=O(nm)$
- Complexity of union of all C-obstacles $O(nm)$
  - Trapezoidation computed in time $O(mn \log(mn))$

- Compute their union $O(mn \log^2(mn))$:
  - divide-and-conquer + line sweep,
  - similar to computing the union of squares from hw
  - (can be done faster)

Compute the Union

- Divide and Conquer:

  ComputeUnion $R \oplus T_1, \ldots, R \oplus T_n$

  1. Let C_1 = ComputeUnion($R \oplus T_1, \ldots, R \oplus T_{n/2}$)
  2. Let C_2 = ComputeUnion($R \oplus T_{n/2+1}, \ldots, R \oplus T_n$)
  3. Return $C_1 \cup C_2$ can compute using line sweep

- Complexity of C_1 , C_2 is $O(mn)$ → line sweep takes $O(mn \log(mn))$ time
- Recurrence $T(n) = 2T(n/2) + O(mn \log(mn))$
- Solves to $O(mn \log^2 (mn))$

Result Summary

- Given:
  - Robot R of complexity m, translating among
  - Disjoint polygonal obstacles with total complexity n
- We can:
  - Preprocess workspace (i.e. build Roadmap) in $O(nm \log^2(nm))$ time
  - Answer if there is a collision free path from any start to any goal in $O(mn)$ time

Higher dim – randomized planner

- Usually the complexity of the free space for a robot with *d* degrees of freedom in an environment of complexity n is $\Theta(n^d)$
- It is not practical to construct the free space.
- Instead, we (very roughly) do
  - create a sample *S* of positions of *R*
  - For each position, check if is free. If yes, it is a node of the graph.
  - For every pair of free positions, chech if the segment connecting them is free. If yes connect them by an edge.
  - Find a path from *s* to *t* in this graph.

- Works well in practice
- Problem: narrow passage.
- Application (one of many): protein docking.