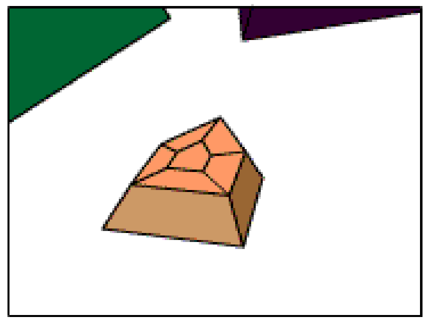
Robot Motion Planning

Philippe Cheng
MIT
Geometric Computation (6.838J)
11.1.01



Overview

Introduction
Motion Planning Problem, motivation

Work & Configuration Space
Defining robot placement and configuration

Point Robot Motion Planning
Map decomposition and path computation for the simple case

Minkowski Sums
Config-space obstacles, definition, examples, theorems, and algorithm

Polygonal Robot Motion Planning
Putting it together, algorithm analysis

Motion Planning with Rotations
Modifications to config-space to handle rotations

Summary
Recap of solution to problem

Introduction

Goal in robotics

Autonomous robots that can plan their own motions

Motion planning problem

Move from start to destination

No collisions with walls (using floorplans)

No collisions with people (using sensors)

Examples

Robots in a warehouse or factory

AI

Simplifying Assumptions

2D Planar Region

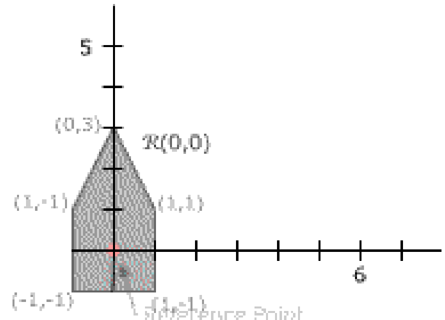
Polygonal obstacles

Polygonal robot

Static (no people)

Constraint-free movement (translations/rotations, no car-like movements)

Work Space & Configuration Space



R(x,y) - Robot reference point at coordinates (x,y)
R(0,0) - Robot at origin

Robot vertices are defined relative to the reference point

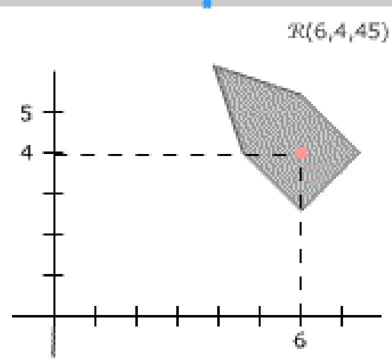
For example, suppose at R(0,0) the vertices lie at:
(1,-1), (1,1), (0,3), (1,-1), (-1,-1)

Then at R(6,4), the robot's vertices are now at:
(7,3), (5,5), (6,7), (5,5), (5,3)

For rotations, let the reference point = the pivot point
R(x,y,Φ) - Added Φ parameter defines robot orientation

Continue

Work Space & Configuration Space



R(x,y) - Robot reference point at coordinates (x,y)
R(0,0) - Robot at origin

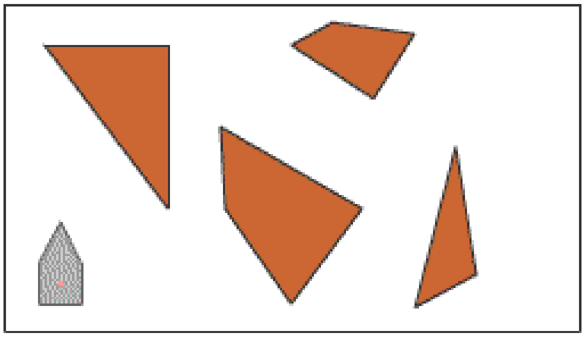
Robot vertices are defined relative to the reference point

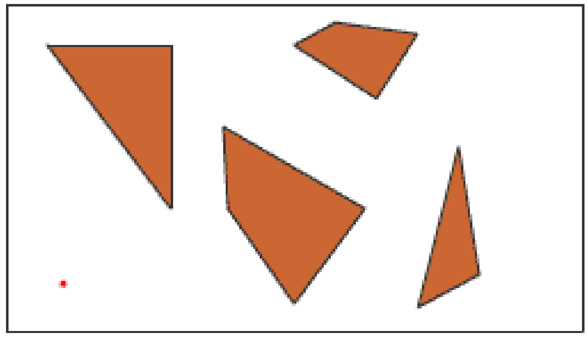
For example, suppose at R(0,0) the vertices lie at:
(1,-1), (1,1), (0,3), (1,-1), (-1,-1)

Then at R(6,4), the robot's vertices are now at:
(7,3), (5,5), (6,7), (5,5), (5,3)

For rotations, let the reference point = the pivot point
R(x,y,Φ) - Added Φ parameter defines robot orientation

Continue





Work Space

2D environment with set **S** of obstacles
The "real world" where the robot moves around

Configuration Space

Parameter space for the robot
Defined as **C(R)**

A polygonal robot in the work space is represented by a dot in the configuration space

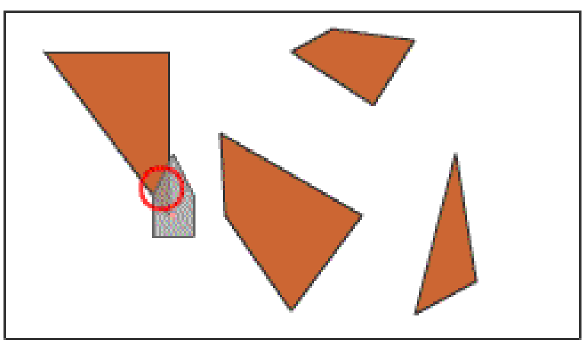
Not all points in the config space are possible

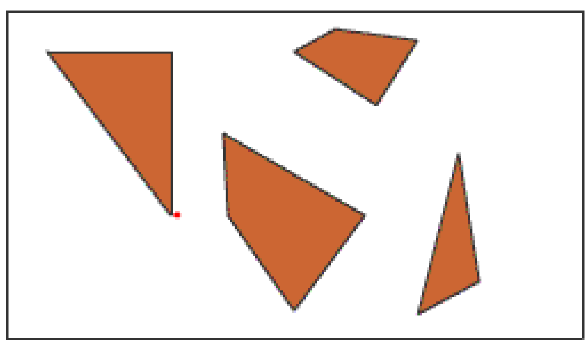
- **Forbidden Config Space C_{forb}(R,S)** - Set of points in the config space that correspond to placements in the work space where the robot intersects an obstacle (forms **C-Obstacles**)
- **Free Config Space C_{free}(R,S)** - The rest of the config space

unobstructed path in the work space -> Path for the robot in the config space

Continue

Robot Motion Planning > Work & Configuration Space 06/43





Work Space

2D environment with set **S** of obstacles
The "real world" where the robot moves around

Configuration Space

Parameter space for the robot
Defined as **C(R)**

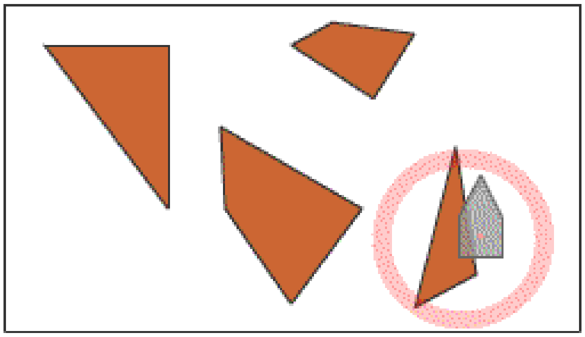
A polygonal robot in the work space is represented by a dot in the configuration space

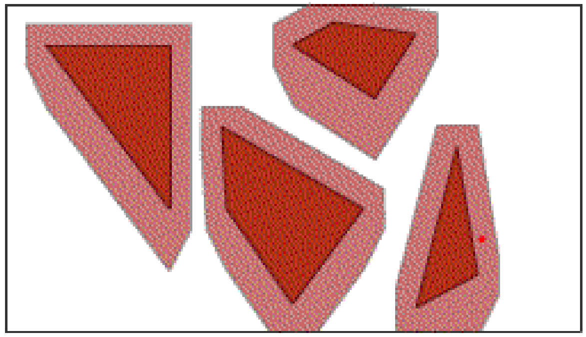
Not all points in the config space are possible

- **Forbidden Config Space C_{forb}(R,S)** - Set of points in the config space that correspond to placements in the work space where the robot intersects an obstacle (forms **C-Obstacles**)
- **Free Config Space C_{free}(R,S)** - The rest of the config space

unobstructed path in the work space -> Path for the robot in the config space

Continue





Work Space

2D environment with set **S** of obstacles

The "real world" where the robot moves around

Configuration Space

Parameter space for the robot

Defined as **C(R)**

A polygonal robot in the work space is represented by a dot in the configuration space

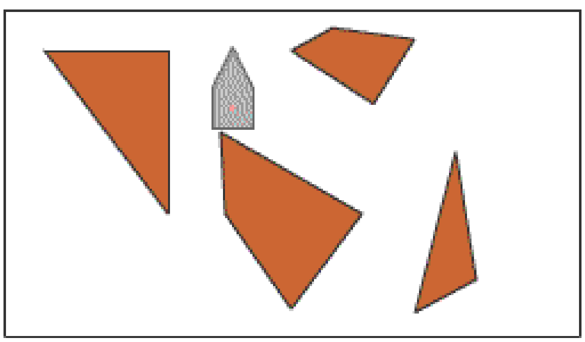
Not all points in the config space are possible

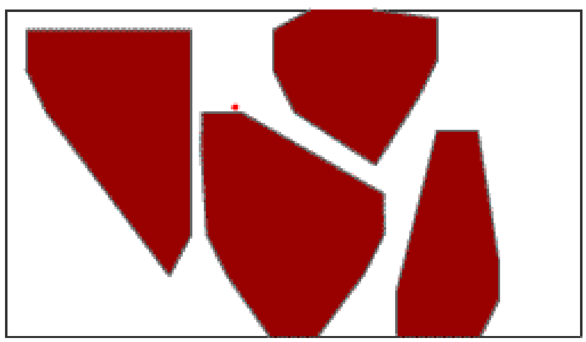
- **Forbidden Config Space C_{forb}(R,S)** - Set of points in the config space that correspond to placements in the work space where the robot intersects an obstacle (forms **C-Obstacles**)
- **Free Config Space C_{free}(R,S)** - The rest of the config space

unobstructed path in the work space -> Path for the robot in the config space

Continue

Robot Motion Planning > Work & Configuration Space 06/43





Work Space

2D environment with set **S** of obstacles

The "real world" where the robot moves around

Configuration Space

Parameter space for the robot

Defined as **C(R)**

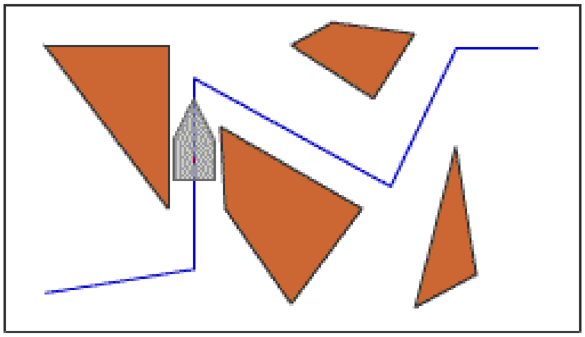
A polygonal robot in the work space is represented by a dot in the configuration space

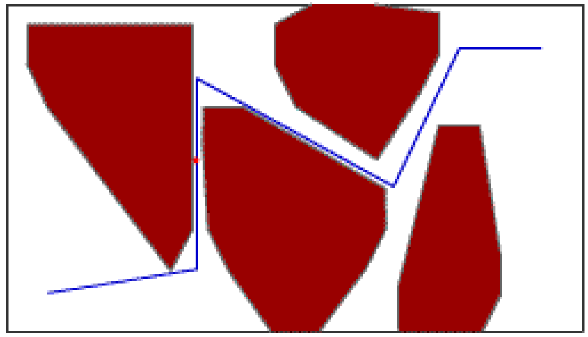
Not all points in the config space are possible

- **Forbidden Config Space C_{forb}(R,S)** - Set of points in the config space that correspond to placements in the work space where the robot intersects an obstacle (forms **C-Obstacles**)
- **Free Config Space C_{free}(R,S)** - The rest of the config space

unobstructed path in the work space -> Path for the robot in the config space

Continue





Work Space

2D environment with set **S** of obstacles
The "real world" where the robot moves around

Configuration Space

Parameter space for the robot
Defined as **C(R)**

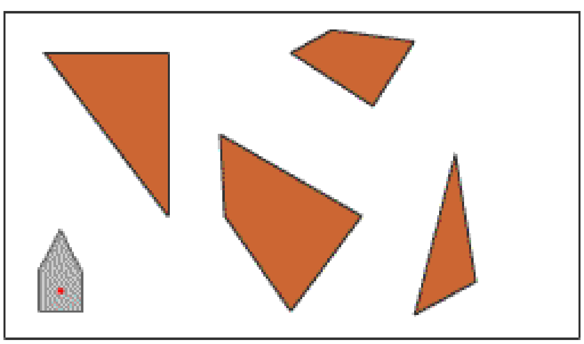
A polygonal robot in the work space is represented by a dot in the configuration space

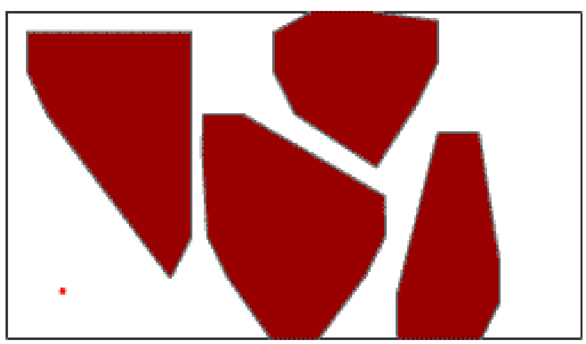
Not all points in the config space are possible

- **Forbidden Config Space $C_{forb}(R,S)$** - Set of points in the config space that correspond to placements in the work space where the robot intersects an obstacle (forms **C-Obstacles**)
- **Free Config Space $C_{free}(R,S)$** - The rest of the config space

unobstructed path in the work space -> Path for the robot in the config space

Point Robot Motion Planning





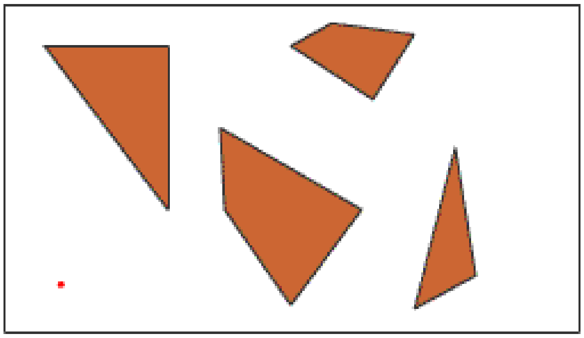
Work Space **Configuration Space**

First, let's analyze the simple case - a point robot

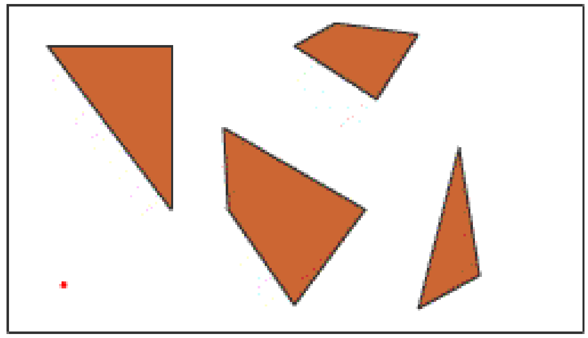
What should the configuration space look like?

Continue

Point Robot Motion Planning



Work Space

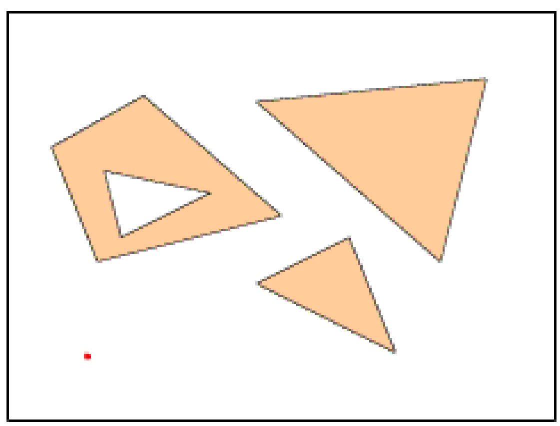


Configuration Space

First, let's analyze the simple case - a point robot

- Work space and config space look identical
- C-Obstacles = Obstacles

Representing the Environment



Create bounding box with enclosing area B

So now you have:

- Point robot **R**
- Set of Obstacles **S** = {P1,... ,Pt}
- $\mathbf{C}_{free} = B \setminus \bigcup_{i=1}^{t} P_i$

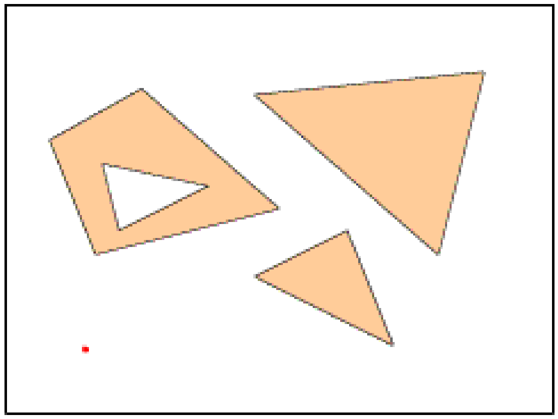
Representing the Environment

Use a trapezoidal map (Chapter 6) to represent the free space

Extend vertical lines up and down from every vertex until they hit something

Remove trapezoids inside obstacles

O(n log n) expected time



Continue

Robot Motion Planning > Point Robot Motion Planning 09/43

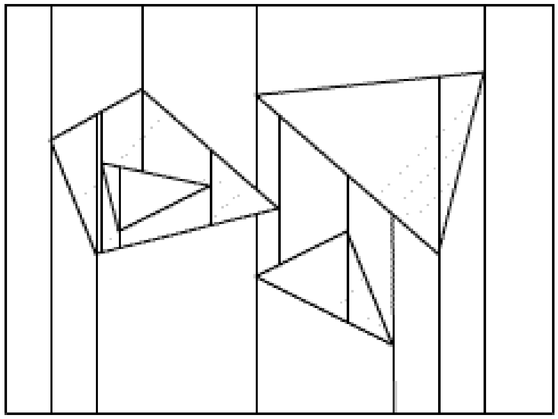
Representing the Environment

Use a trapezoidal map (Chapter 6) to represent the free space

Extend vertical lines up and down from every vertex until they hit something

Remove trapezoids inside obstacles

O(n log n) expected time



Continue

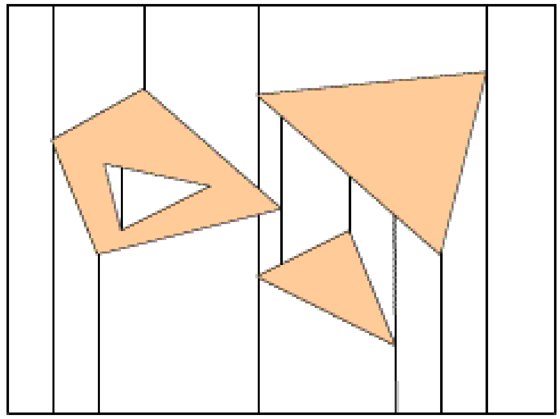
Finding a Path

Create graph G_{road} to represent the possible paths

Place a node at the center of each trapezoid and the middle of each vertical extension

For each trapezoid, draw an edge from the center node to each vertical extension node

Constructed in O(n) time by traversing the doubly connected edge list of the trapezoidal map $T(C_{free})$



Continue

Finding a Path

Finding a path from p_{start} to p_{goal}

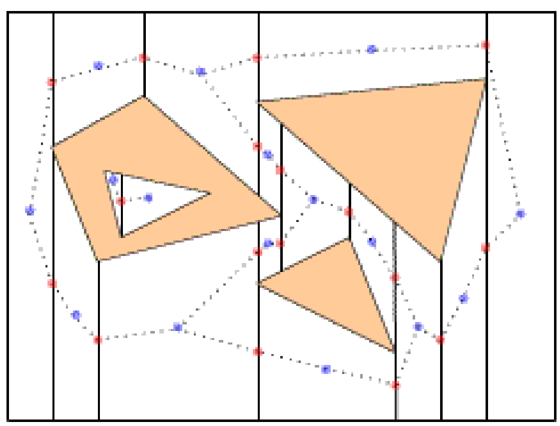
Find trapezoid Δ_{start} containing p_{start} and trapezoid Δ_{goal} containing p_{goal}

If $\Delta_{start} = \Delta_{goal}$, just go directly from p_{start} to p_{goal}

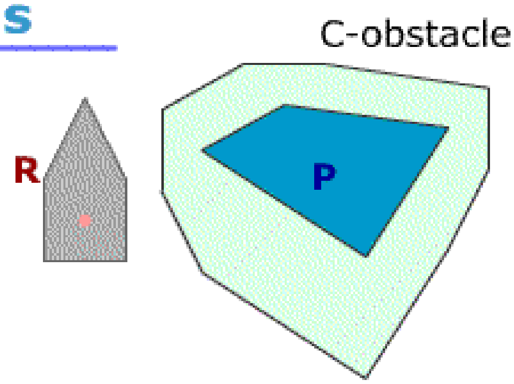
else use breadth-first search to find a path from Δ_{start} to Δ_{goal}

Path = p_{start} -> path from Δ_{start} to Δ_{goal} -> p_{goal}

Overall Running Time:
Preprocessing - O(n log n)
Path Query - O(log n + n)



Minkowski Sums



To begin studying polygonal robot motion planning, we need to cover C-obstacles and Minkowski sums.

C-obstacles help define the free config space to contruct our road graphs.

C-obstacle

$CP := \{(x,y) : R(x,y) \cap P \neq 0\}$

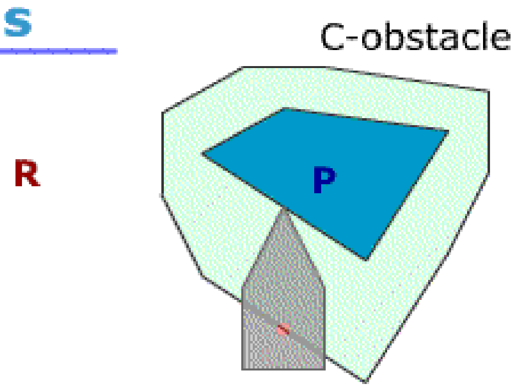
This means a C-obstacle is the set of points in config space that map to placements of R where R intersects an obstacle P

It turns out C-obstacles can be easily calculated using Minkowski Sums...

Continue

Robot Motion Planning > Minkowski Sums 12/43

Minkowski Sums



To begin studying polygonal robot motion planning, we need to cover C-obstacles and Minkowski sums.

C-obstacles help define the free config space to contruct our road graphs.

C-obstacle

$CP := \{(x,y) : R(x,y) \cap P \neq 0\}$

This means a C-obstacle is the set of points in config space that map to placements of R where R intersects an obstacle P

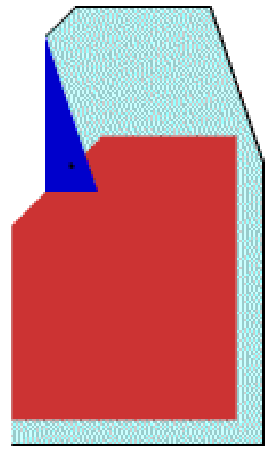
It turns out C-obstacles can be easily calculated using Minkowski Sums...

Minkowski Sums

Minkowski Sum of two sets S_1, S_2, denoted by $S_1 \oplus S_2$, is
$S_1 \oplus S_2 := \{p + q : p \in S_1, q \in S_2\}$, where
$p + q := (p_x + q_x, p_y + q_y)$

Minkowski sum of two sets of numbers is the vector sum of all pairs.

For this reason the Minkowski Sum polygon can be carved by moving the S_1 around the border of S_2 as shown in the demo.



Continue

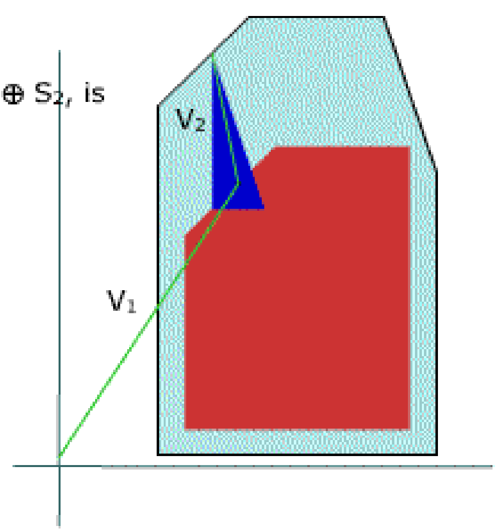
Robot Motion Planning > Minkowski Sums 13/43

Minkowski Sums

Minkowski Sum of two sets S_1, S_2, denoted by $S_1 \oplus S_2$, is
$S_1 \oplus S_2 := \{p + q : p \in S_1, q \in S_2\}$, where
$p + q := (p_x + q_x, p_y + q_y)$

Minkowski sum of two sets of numbers is the vector sum of all pairs.

For this reason the Minkowski Sum polygon can be carved by moving the S_1 around the border of S_2 as shown in the demo.



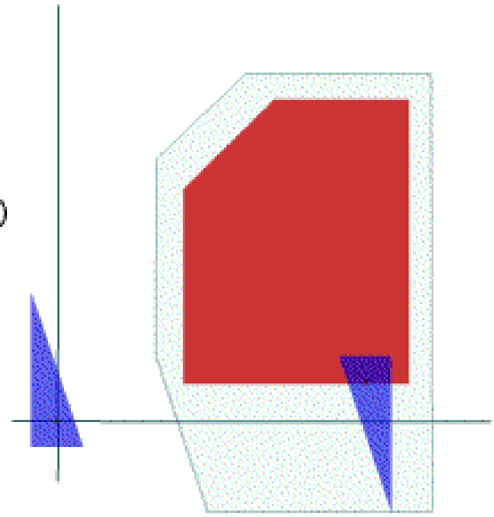
Note

For a point p = (p_x,p_y), define -p := (-p_x,-p_y) and for a set S, define -S := {-p : p ∈ S}

So for polygon A, -A is simply polygon A flipped

Theorem 1 - The C-obstacle of P is P ⊕ (-R(0,0))
where P is an obstacle and R is the robot

What this means is you can get the C-obstacle by computing the Minkowski Sum of the obstacle and the robot flipped.



Continue

Robot Motion Planning > Minkowski Sums 14/43

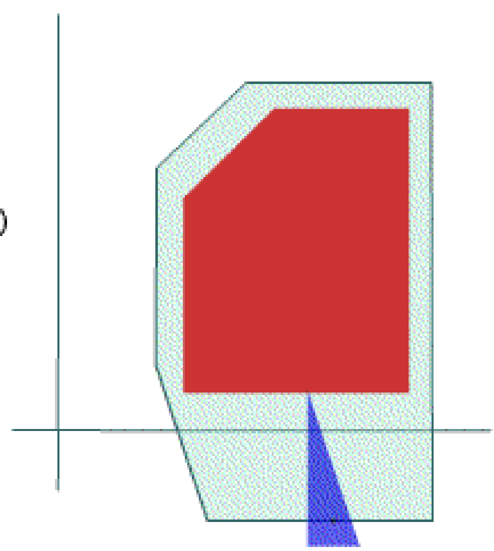
Note

For a point p = (p_x,p_y), define -p := (-p_x,-p_y) and for a set S, define -S := {-p : p ∈ S}

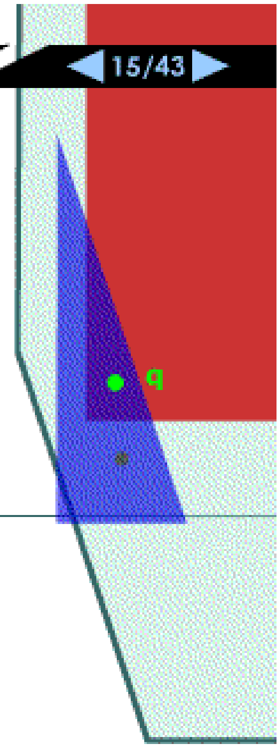
So for polygon A, -A is simply polygon A flipped

Theorem 1 - The C-obstacle of P is P ⊕ (-R(0,0))
where P is an obstacle and R is the robot

What this means is you can get the C-obstacle by computing the Minkowski Sum of the obstacle and the robot flipped.



Theorem 1 - The C-obstacle of P is $P \oplus (-R(0,0))$
where P is an obstacle and R is the robot

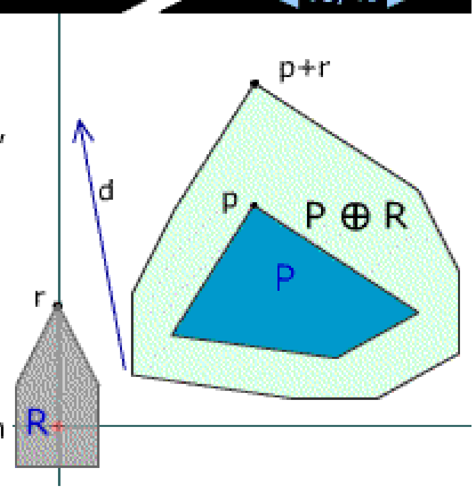


Proof
Prove that $R(x,y)$ intersects P iff $(x,y) \in P \oplus (-R(0,0))$

Step 1: Suppose $R(x,y)$ intersects P
Let q be a point in the intersection.
Since $q \in R(x,y)$, $(q_x - x, q_y - y) \in R(0,0)$
Rearrange to get $(-q_x + x, -q_y + y) \in -R(0,0)$
Since $q \in P$
this implies $(x,y) \in P \oplus (-R(0,0))$

Step 2: Let $(x,y) \in P \oplus (-R(0,0))$, this means there are points $(r_x,r_y) \in R(0,0)$ and $(p_x,p_y) \in P$ s.t. $(x,y) = (p_x-r_x, p_y-r_y)$
Rearrange to get $p_x = r_x + x$, $p_y = r_y + y$
this implies $R(x,y)$ intersects P

Our eventual algorithm for Minkowski Sum computation requires understanding of extreme points, pseudodiscs and directions.



Extreme Points

Note that the extreme point in direction d of $P \oplus R$ is the sum of the extreme points of P and R

Theorem 2 - Let P and R be two convex polygons with n and m edges, then $P \oplus R$ is a convex polygon with at most n + m edges.

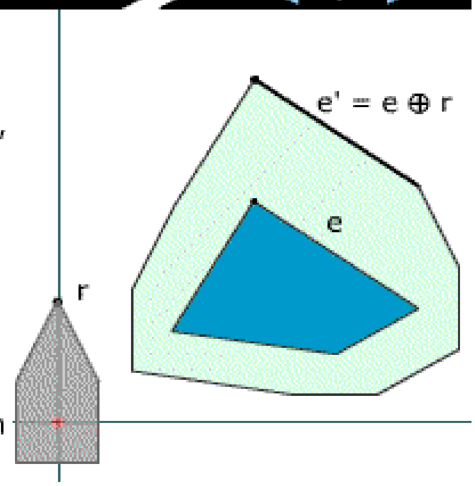
Intuition - an edge e of $P \oplus R$ must come from an edge in P or R. An edge in P or R cannot contribute more than once.

Continue

Our eventual algorithm for Minkowski Sum computation requires understanding of extreme points, pseudodiscs and directions.

Extreme Points

Note that the extreme point in direction d of P ⊕ R is the sum of the extreme points of P and R



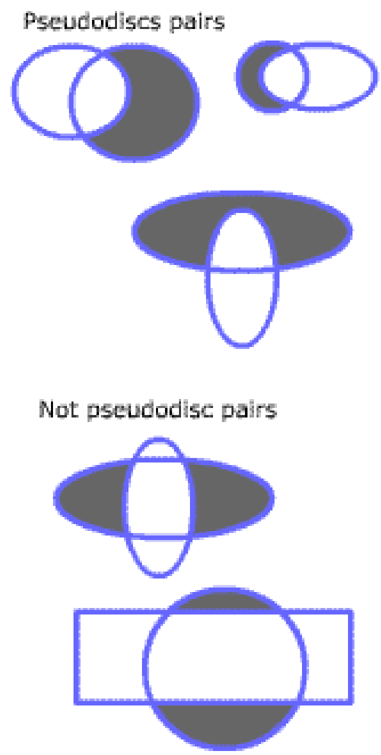
Theorem 2 - Let P and R be two convex polygons with n and m edges, then P ⊕ R is a convex polygon with at most n + m edges.

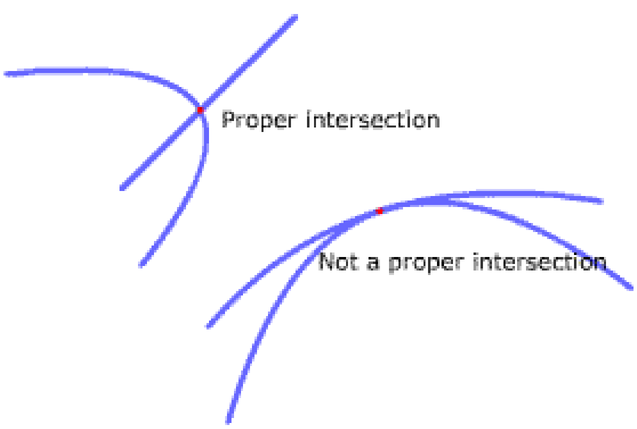
Intuition - an edge e of P ⊕ R must come from an edge in P or R. An edge in P or R cannot contribute more than once.

Pseudodiscs Pairs

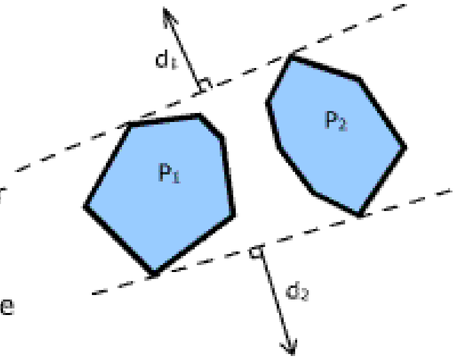
A pair o_1, o_2 of planar objects are a pair of pseudodiscs if it satisfies the pseudodisc property that $o_1 \setminus o_2$ and $o_2 \setminus o_1$ are connected.

Pseudodisc pairs have the property of having at most two proper intersections at their boundaries.



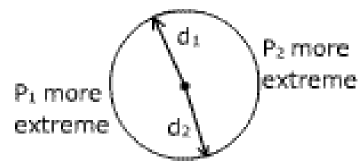


Directions



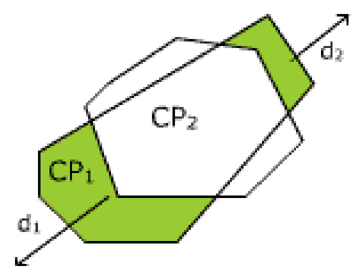
Given two convex polygons, one is more extreme in direction d if its extreme point in direction d is further than that of the other polygon.

The range in which polygons are more extreme can be modelled by a unit circle.



Observation
Let P_1 and P_2 be convex polygons with disjoint interiors. Let d_a and d_b be directions in which P_1 is more extreme, then P_1 is more extreme in all directions from either d_a to d_b or d_b to d_a.

Theorem 3 - Let P_1 and P_2 be convex polygons with disjoint interiors, and let R be another convex polygon. Then $P_1 \oplus R$ and $P_2 \oplus R$ are pseudodisc pairs.



Proof by Contradiction
Define $CP_1 := P_1 \oplus R$ and $CP_2 := P_2 \oplus R$. By symmetry it suffices to show that $CP_1 \setminus CP_2$ is connected.

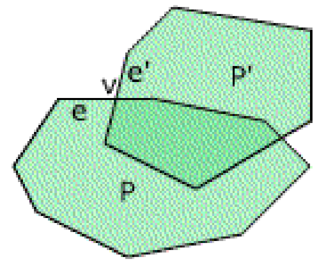
Suppose $CP_1 \setminus CP_2$ is not a pseudodisc pair because it forms two unconnected components.

Then there are two different directions d_1 and d_2 such that CP_1 is more extreme than CP_2.

Since CP_1 and CP_2 are both Minkowski Sums involving R, P_1 must also be more extreme than P_2 in directions d_1 and d_2.

This means P_1 is more extreme for all directions in the range d_1 to d_2 or d_2 to d_1, implying the two components are connected which is a contradiction to our assumption.

Theorem 4 - Let S be a collection of polygonal pseudodiscs with n edges total. Then the complexity of their union is O(n).



Proof
A bound of 2n on the maximal complexity of the union can be found by charging every vertex of the union to a pseudodisc vertex s.t. any pseudodisc vertex is charged at most twice.

Two types of vertices in the union boundary - original boundary pseudodisc vertices and vertices formed from intersections.

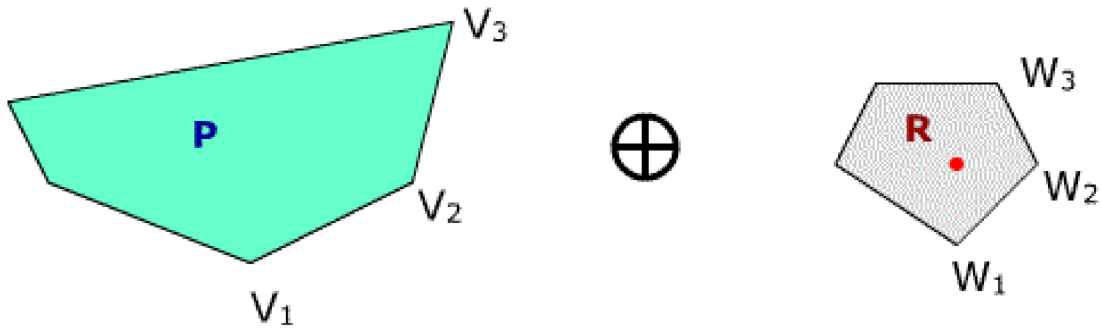
Pseudodisc vertices are charged to themselves whereas intersection vertices are charged to an interior vertex found by following one of the edges that formed it.

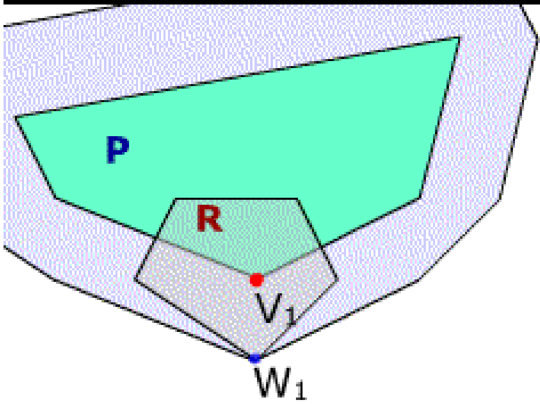
Now each original pseudodisc vertex was charged at most twice.

Finding the points of P ⊕ R

For each polygon, sort vertices in counter-clockwise order, starting with the bottom-most one.

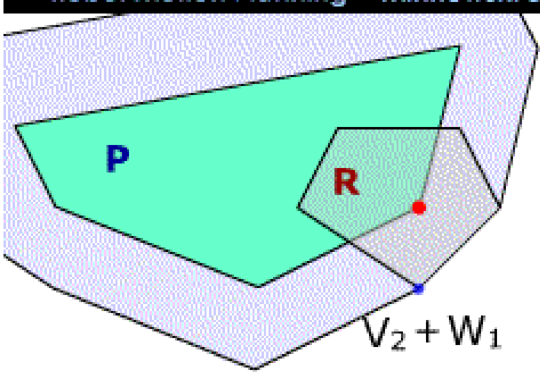
Now we will compute the points in the Minkowski Sum in counter-clockwise order.

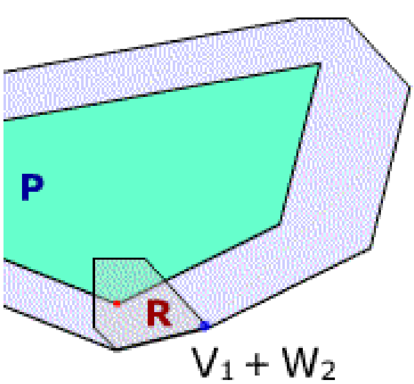




Start by adding point $V_1 + W_1$, the lowest point on the Minkowski Sum

Since the Minkowski Sum is carved by sliding R around the border,
should $V_2 + W_1$ be the next point?

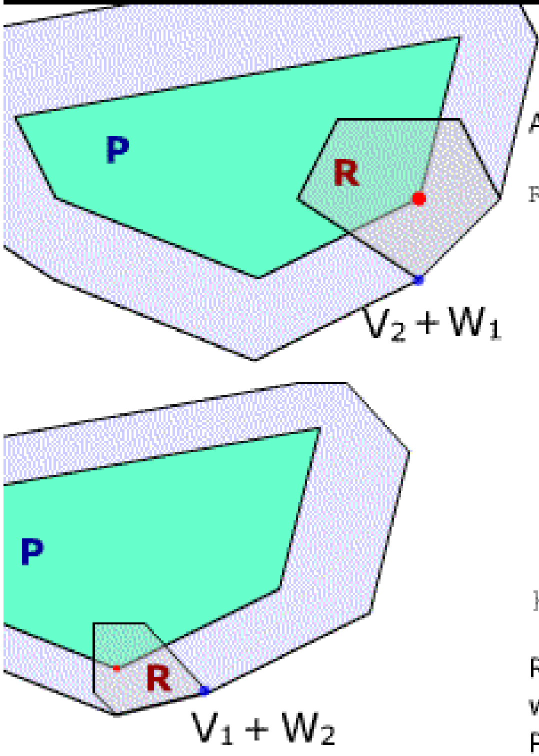




Not necessarily, in the first case the next point was indeed $V_2 + W_1$ but in the second case R did not move.

In the first case, R moves along the border of P and carves a parallel line. But if angle(W_1,W_2) is smaller than angle(V_1,V_2) then a side of R extends further.

For the algorithm, repeat these comparisons until you've circled around both polygons.



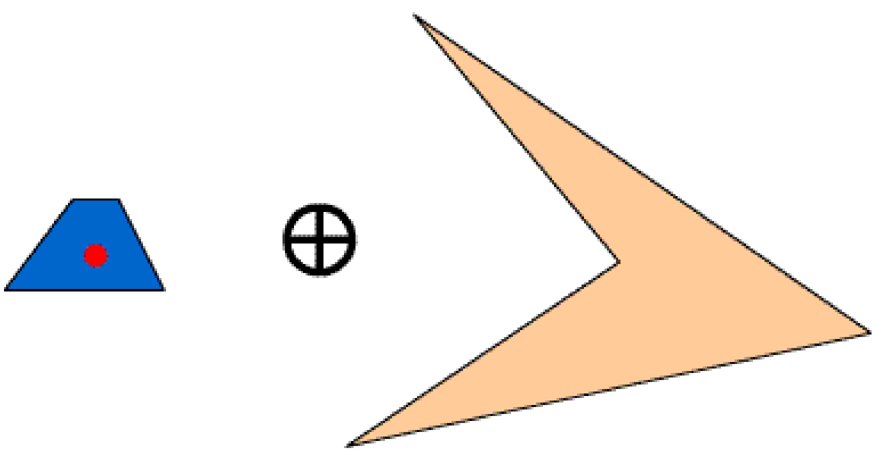
Algorithm MinkowskiSum(P,R)

```
Repeat until iterated around both polygons {
   Add Vi + Wj
   if angle(ViVi+1) < angle (WjWj+1) {
      //Next robot edge falls inside
      j++
   } else if angle(ViVi+1) > angle(WjWj+1) {
      //Next robot edge extends out
      i++
   } else {
      //Both edges parallel
      i++, j++
   }
}
```

Running Time: O(n + m)
where n and m are the number of edges in
P and R

Ok, so how do we find the Minkowski Sum of non-convex polygons?

A: By triangulating the polygon, finding the Minkowski Sums, and then taking the union of the results.

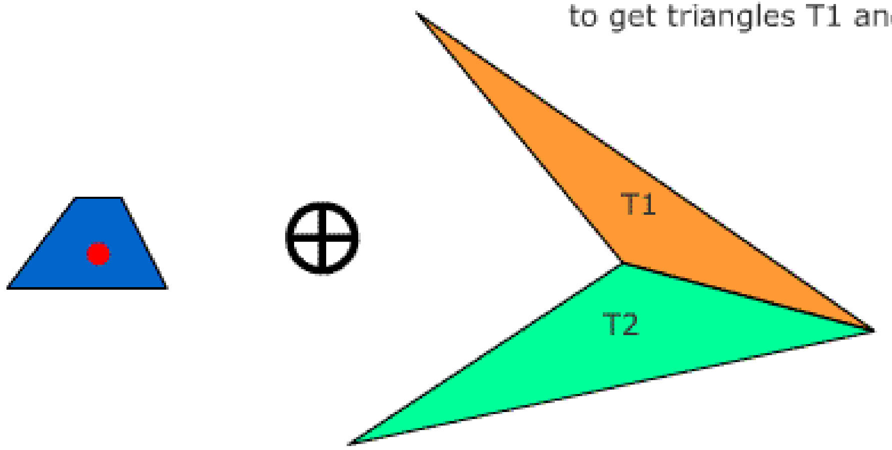


Continue

Ok, so how do we find the Minkowski Sum of non-convex polygons?

A: By triangulating the polygon, finding the Minkowski Sums, and then taking the union of the results.

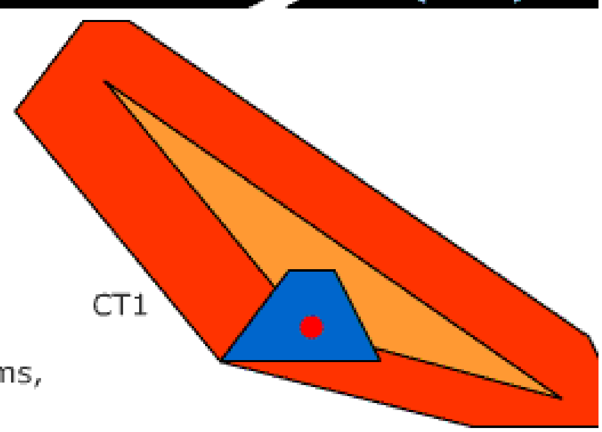
Triangulate Obstacle P
to get triangles T1 and T2



Robot Motion Planning > Minkowski Sums 25/43

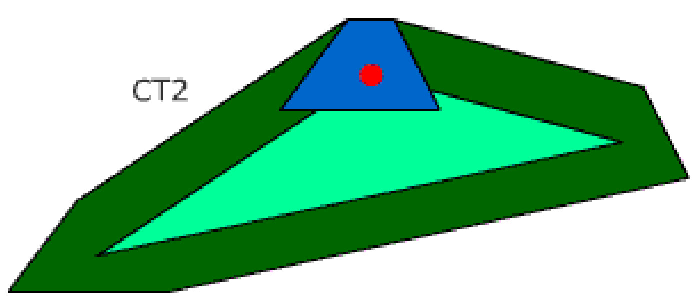
Ok, so how do we find the Minkowski Sum of non-convex polygons?

A: By triangulating the polygon, finding the Minkowski Sums, and then taking the union of the results.



Compute the two Minkowski Sums,
CT1 and CT2

CT1 = T1 ⊕ R
CT2 = T2 ⊕ R

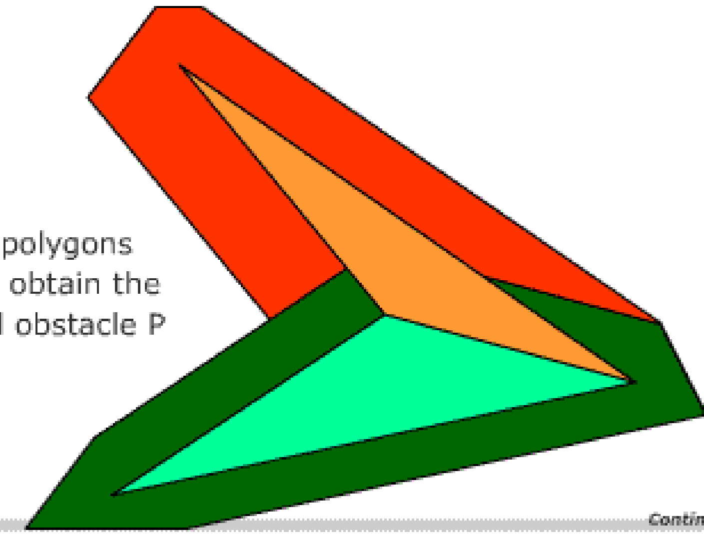


Continue

Ok, so how do we find the Minkowski Sum of non-convex polygons?

A: By triangulating the polygon, finding the Minkowski Sums, and then taking the union of the results.

Now just merge the two polygons CT1 and CT2 together to obtain the C-obstacle of the original obstacle P



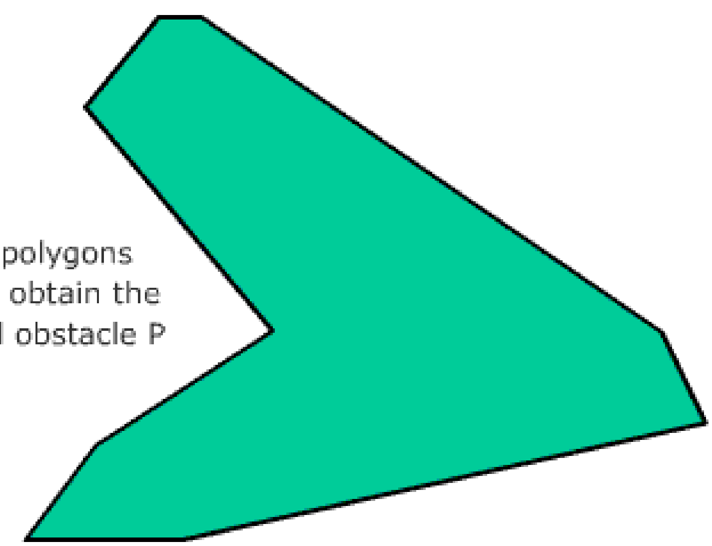
Continue

Robot Motion Planning > Minkowski Sums 25/43

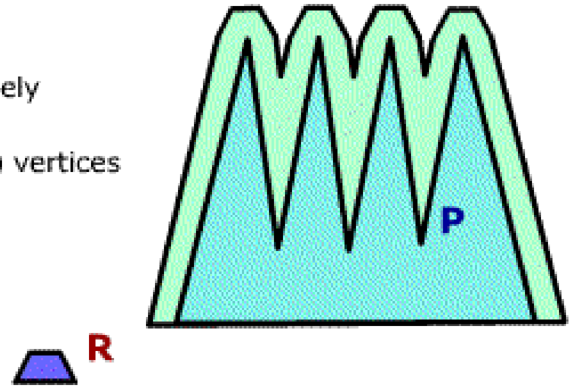
Ok, so how do we find the Minkowski Sum of non-convex polygons?

A: By triangulating the polygon, finding the Minkowski Sums, and then taking the union of the results.

Now just merge the two polygons CT1 and CT2 together to obtain the C-obstacle of the original obstacle P



If P is non-convex and R is convex
Say P and R have n and m vertices respectively
Triangulate P into n-2 triangles (Chapter 3)
For each triangle t, find t ⊕ R (at most m+3) vertices
Find the union of all the Minkowski Sums



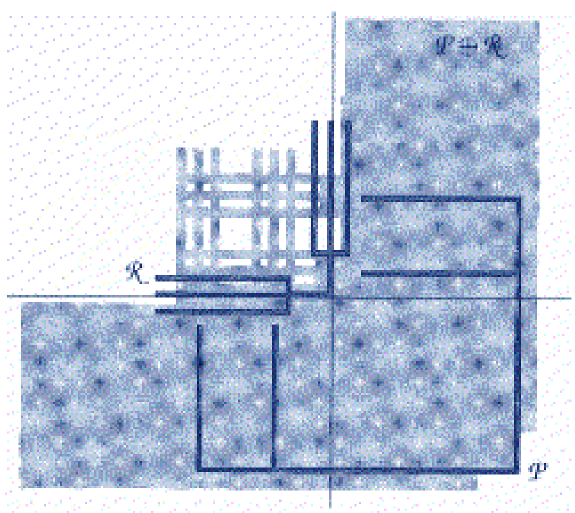
Complexity

All triangles are disjoint, thus the Minkowski Sums of each triangle with R form a collection of pseudodiscs. (Theorem 3) The Complexity of their union is linear to the sum of their complexities. There are n-2 polygons each with m+3 complexity so the total complexity is **O(nm)**.

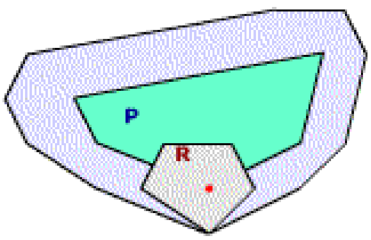
If both P and R are non-convex

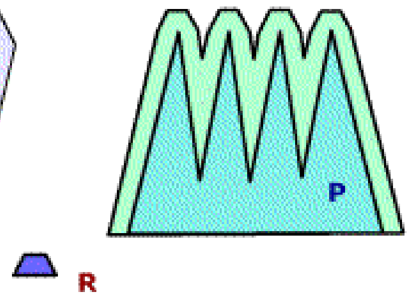
Triangulate them into n-2 and m-2 triangles respectively. Find the Minkowski Sum of all pairs, this creates (n-2)(m-2) polygons of constant complexity.

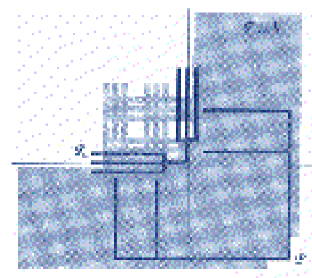
The union of all these polygons is thus of $O(n^2m^2)$ complexity. One such example is shown here.



In summary, the complexity of a Minkowski Sum is as follows
O(n+m) if both polygons are convex
O(nm) if only one of the polygons is convex
O(n^2m^2) if both polygons are non-convex







Polygonal Robot Motion Planning

Now that we know how to calculate C-obstacles we can solve the polygonal robot motion planning problem.

With a convex robot of constant complexity, the complexity of the free config space is O(n)

Triangulated obstacles
-> O(n) triangles
-> O(n) C-obstacles each with constant complexity
-> O(n) set of pseudodiscs

Since the complexity of the union of pseudodiscs is linear in the sum of their complexities, the resulting union has linear complexity

How do we find the free configuration space?

The forbidden config space is the union of all the C-obstacles
Use divide-and-conquer to compute all the unions
The free config space is then the complement of this

The free configuration space can be computed in $O(n \log^2 n)$ time where n is the total number of edges

Triangulation - O(n log n)
If obstacle P_i has m_i complexity, it can be triangulated in O(m log m) time
Total time to triangulate all obstacles is proportional to

$$\sum_{i=1}^{t} m_i \log m_i \leq \sum_{i=1}^{t} m_i \log n = n \log n$$

Computing C-obstacles - O(n)
Minkowski sum of O(n) triangles with a robot of constant complexity takes O(n) time

Merging - $O(n \log^2 n)$
One merge step can be done in $O((n_1+n_2+k) \log (n_1+n_2))$ (Chapter 2)
where n_1, n_2, and k are the complexities of C_{forb1}, C_{forb2}, and $C_{forb1} \cup C_{forb2}$
The complexity of the fordidden space is O(n) so the merge step is O(n log n)
With divide-and-conquer we get this recurrence
$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n \log n)$ which is $O(n \log^2 n)$

Total Time = $O(n \log n) + O(n) + O(n \log^2 n) = O(n \log^2 n)$

Polygonal Robot Motion Planning Solution Summary

Let R be a convex robot of constant complexity translating among a set S of disjoint polygonal obstacles with n edges in total. We can preprocess S in $O(n \log^2 n)$ 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.

Motion Planning with Rotations

Restricting robot motion to only translations is a major disadvantage since some robots may need to change their orientation to pass through a narrow passage or corner.

The configuration space we have seen so far represents only one angle

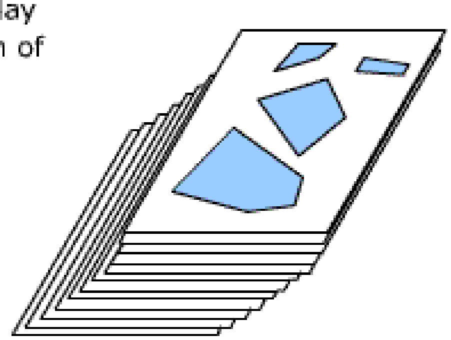
As a robot changes orientation, the C-obstacles alter to represent the new Minkowski sum.

Each possible orientation creates a different configuration space - resulting in multiple levels of configuration spaces.

Each level corresponds to a given angle Φ and contains its own roadmap G_i. With multiple levels of roadmaps, we need some way to connect them together to form one G_{road} for the entire configuration space.

Movements across a particular level still represents a translation at a fixed angle. Moving from one level to another (adjacent upward or downward) represent a rotation in the object.

For each adjacent levels, compute the overlay (chapter 2) to find the common intersection of their configuration spaces. These enable passage from one level to another.



Since arbitrary orientations would require an infinite number of levels, we need to sample the possible angles.

So to find a path, change orientation to the closest level then follow the computed path that may go across levels. At the goal change to the closest level and do a final rotation.

Problems

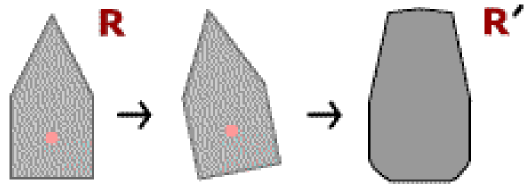
A start orientation may be in free space but if the closest level is not then a false negative will be given.

Moving from one slice to another could be an undetected collision if the discretization wasn't enough.

Both problems can be reduced by increasing the number of levels.

One solution is to modify robot R in the following way.

Rotate R according to the interval between levels. If there is a level every 10 degrees, rotate R left and right by 5 degrees. R' is the new robot formed by the sweeping.



Configuration spaces based on new C-obstacles

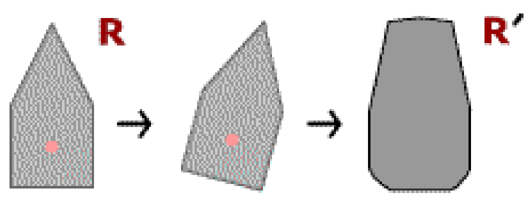
Levels now represent intervals (ie thicker levels)

Robot Motion Planning > Motion Planning with Rotations 36/43

Both problems can be reduced by increasing the number of levels.

One solution is to modify robot R in the following way.

Rotate R according to the interval between levels. If there is a level every 10 degrees, rotate R left and right by 5 degrees. R' is the new robot formed by the sweeping.

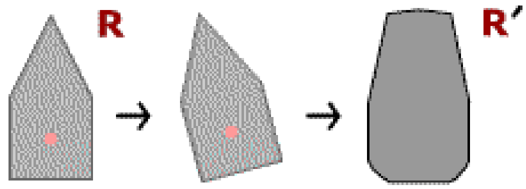


Configuration spaces based on new C-obstacles

Levels now represent intervals (ie thicker levels)

Since R' is larger, the free configuration space at each level is smaller.

Moving from one level to another with R' will no longer result in R (the actual robot) colliding with a level midway.

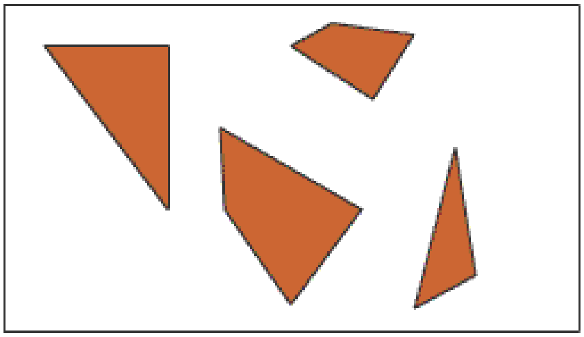


This might make some previously possible paths impossible.

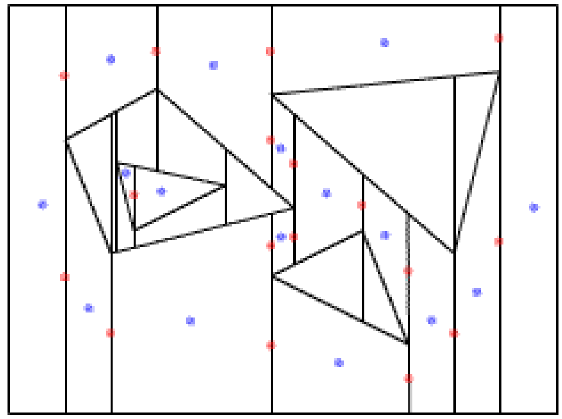
In practice these issues are avoided by using sufficient levels in conjunction with the R' trick.

Summary

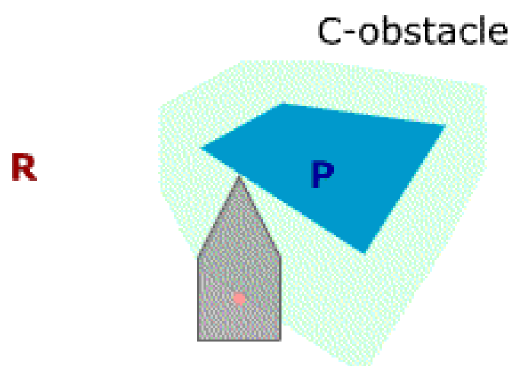
The first step to solving the robot motion planning problem was to create the configuration space as our representation.



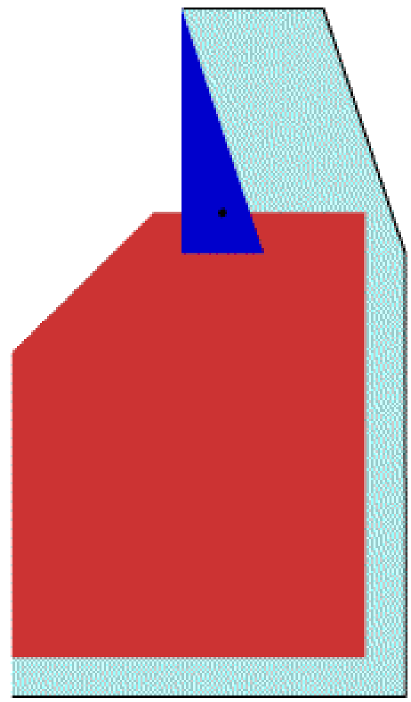
Use trapezoidal decomposition to create graphs for path queries



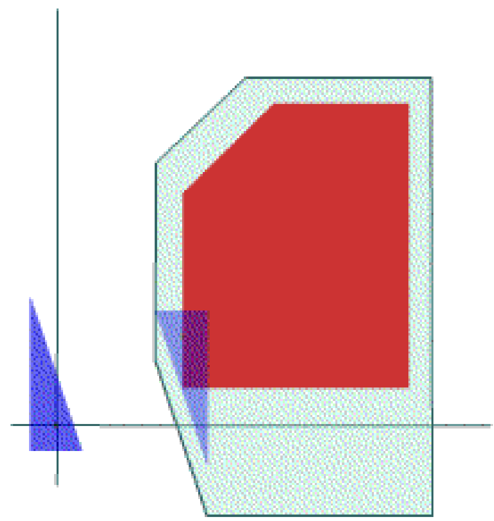
Studied C-obstacles to enable polygon robots



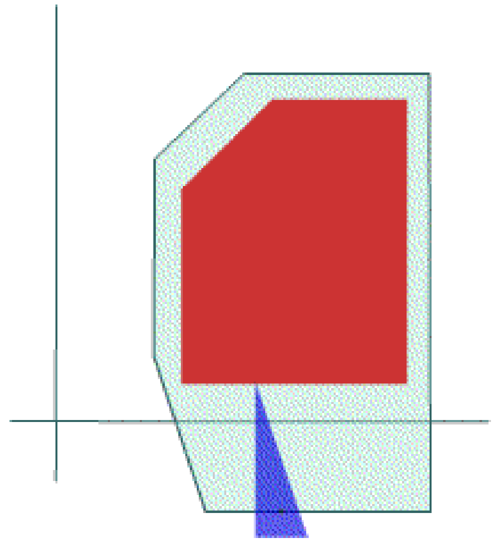
Used Minkowski Sums
to create C-obstacles



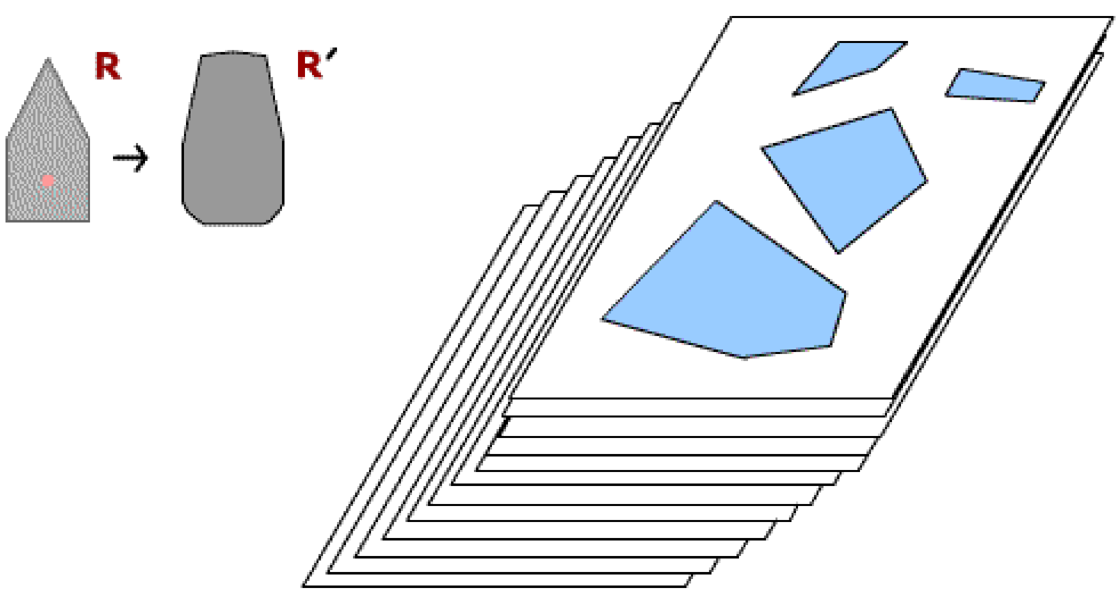
Used Minkowski Sums
to create C-obstacles



Used Minkowski Sums
to create C-obstacles



Rotation creating multiple levels of configurations spaces



Questions?