Robot Motion Planing
Getting Where You Want to Be

Fatemeh Dehghani
F.dehghani@stu.yazduni.ac.ir

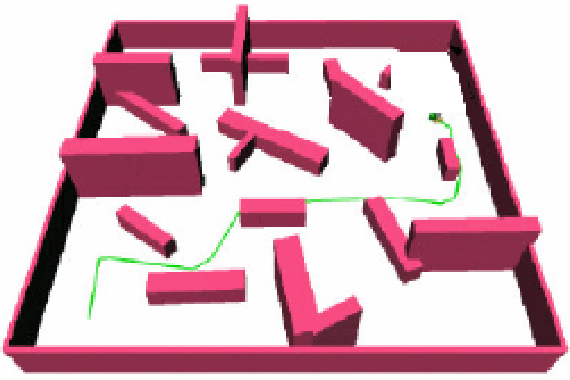
Yazd University

Computer Science 2010

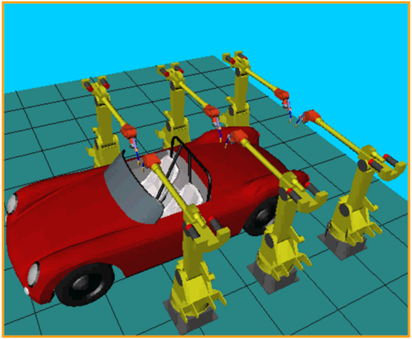
Outline

Outline

What is Motion Planning?

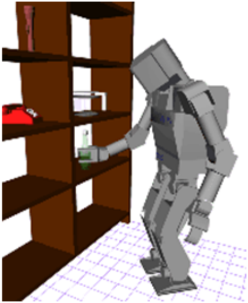


Type of Robot



Robot arm

Type of Robot



Mobile robot

Definition

What is Motion Planning?

- Determining where to go without hit obstacles

Definition

What is Motion Planning?

- Determining where to go without hit obstacles
- Given:
  - a robot R, with start and goal position

Definition

What is Motion Planning?

- Determining where to go without hit obstacles
- Given:
  - a robot R, with start and goal position
  - set S of obstacles

Definition

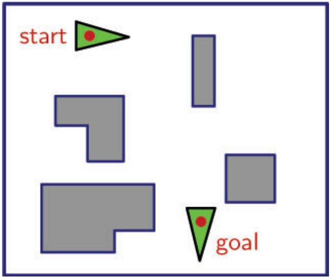
What is Motion Planning?

- Determining where to go without hit obstacles
- Given:
  - a robot R, with start and goal position
  - set S of obstacles
- find collision-free path for the robot.

Definition

What is Motion Planning?

- Determining where to go without hit obstacles
- Given:
  - a robot R, with start and goal position
  - set S of obstacles
- find collision-free path for the robot.



Simplification

Assumptions

- Look at a 2-dimensional motion planning problem

Simplification

Assumptions

- Look at a 2-dimensional motion planning problem
- The environment is a planar region

Simplification

Assumptions

- Look at a 2-dimensional motion planning problem
- The environment is a planar region
- Obstacles and robot are polygons

Simplification

Assumptions

- Look at a 2-dimensional motion planning problem
- The environment is a planar region
- Obstacles and robot are polygons
- There are no mobile obstacles

Simplification

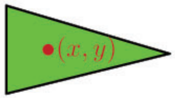
Assumptions

- Look at a 2-dimensional motion planning problem
- The environment is a planar region
- Obstacles and robot are polygons
- There are no mobile obstacles
- Robot can move in arbitrary directions

Outline

Specifying a Robot placement

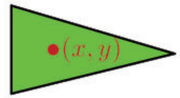
rigid object in 2D,
translation only



x- and *y*-coord of
reference point
2 degrees of freedom

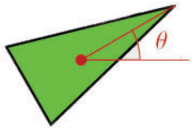
Specifying a Robot placement

rigid object in 2D, translation only



x- and y-coord of reference point
2 degrees of freedom

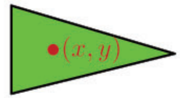
rigid object in 2D, translations and rotations



x- and y-coord of reference
point, angle of rotation
3 degrees of freedom

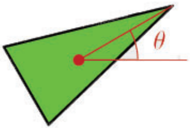
Specifying a Robot placement

rigid object in 2D, translation only



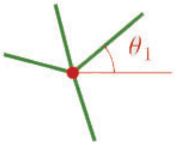
x- and y-coord of reference point
2 degrees of freedom

rigid object in 2D, translations and rotations



x- and y-coord of reference
point, angle of rotation
3 degrees of freedom

point robot with k arms,
translations and movement of arms



x- and y-coord of reference
point, k rotation angles
$k + 2$ degrees of freedom

Configuration spaces

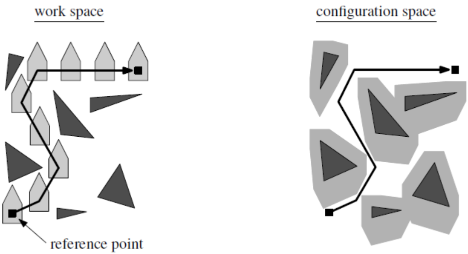
Placement

- Placement of robot with f degrees of freedom can be specified with f parameters

Configuration spaces

Placement

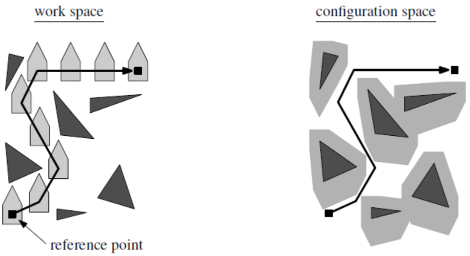
- Placement of robot with f degrees of freedom can be specified with f parameters
- The parameter space of a robot R is usually called its configuration space that denote by $C(R)$



Configuration spaces

Placement

- Placement of robot with f degrees of freedom can be specified with f parameters
- The parameter space of a robot R is usually called its configuration space that denote by $C(R)$
- a placement in work space $\Leftrightarrow$ a point in f-dimensional configuration space



Configuration spaces

example

- Robot in 2D, translations only $\Longrightarrow$ configuration space: $\mathbb{R}^2$

Configuration spaces

example

- Robot in 2D, translations only $\Longrightarrow$ configuration space: $\mathbb{R}^2$
- **For robat in 2D, translations and rotations:**

  A point (x, y, ϕ) in configuration space corresponds to the placement $R(x, y, \phi)$ in the work space

  That is:

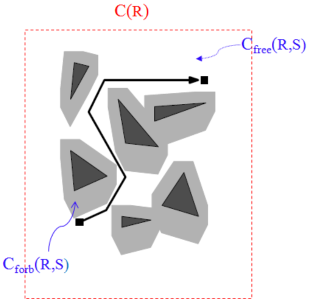
$$\textit{configuration space} : \mathbb{R}^2 \times [0 : 2\pi]$$

Free and Forbidden spaces

Free Space

Points in configuration space corresponding to collision-free placements

Denote by $C_{free}(R, S)$



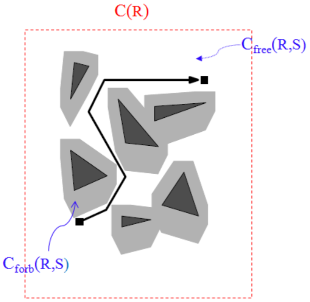
Free and Forbidden spaces

Free Space

Points in configuration space corresponding to collision-free placements
Denote by $C_{free}(R, S)$

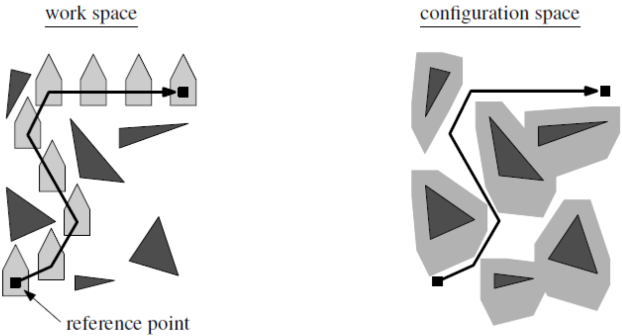
Forbidden Space

Points in configuration space corresponding to colliding placements
Denote by $C_{forb}(R, S)$



To map placement and path

To map placement and path



How to map obstacles to configuration space?

Configuration space obstacles

- An obstacle P is mapped to the set of points p in configuration space such that $R(p)$ intersects P.
  Denote by C-obstacle

How to map obstacles to configuration space?

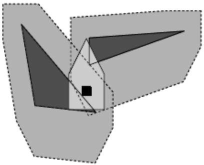
Configuration space obstacles

- An obstacle P is mapped to the set of points p in configuration space such that $R(p)$ intersects P.
  Denote by C-obstacle
- obstacles are open sets, so that the robot is allowed to touch them

How to map obstacles to configuration space?

Configuration space obstacles

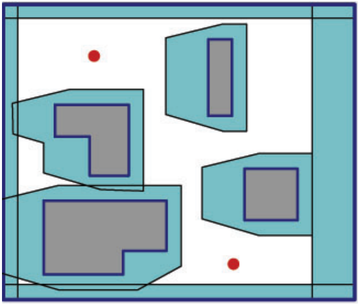
- An obstacle P is mapped to the set of points p in configuration space such that $R(p)$ intersects P.
  Denote by C-obstacle
- obstacles are open sets, so that the robot is allowed to touch them
- C-obstacles may overlap even when the obstacles in the work space are disjoint. This happens when there are placements of the robot where it intersects more than one obstacle at the same time.



Overview

Total Procedure

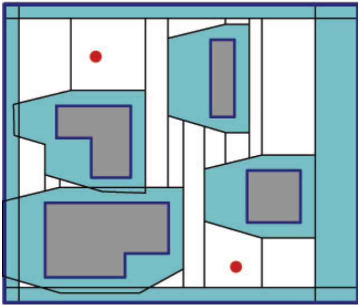
1. Decompose free space into constant-complexity cells
2. Construct dual graph of decomposition
3. Find path in graph
4. Transform path in graph to path in configuration space



Overview

Total Procedure

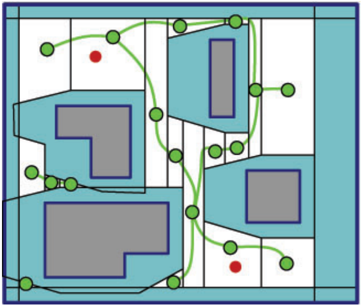
1. Decompose free space into constant-complexity cells
2. Construct dual graph of decomposition
3. Find path in graph
4. Transform path in graph to path in configuration space



Overview

Total Procedure

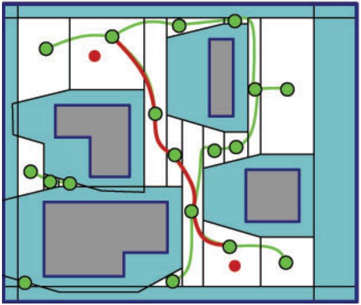
1. Decompose free space into constant-complexity cells
2. Construct dual graph of decomposition
3. Find path in graph
4. Transform path in graph to path in configuration space



Overview

Total Procedure

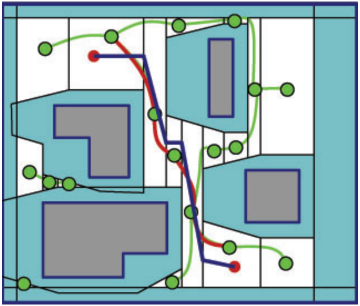
1. Decompose free space into constant-complexity cells
2. Construct dual graph of decomposition
3. Find path in graph
4. Transform path in graph to path in configuration space



Overview

Total Procedure

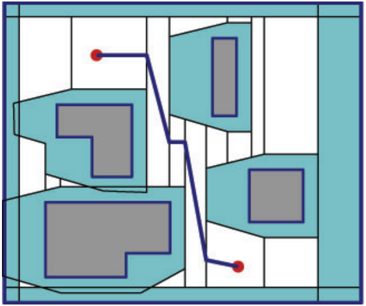
1. Decompose free space into constant-complexity cells
2. Construct dual graph of decomposition
3. Find path in graph
4. Transform path in graph to path in configuration space



Overview

Total Procedure

1. Decompose free space into constant-complexity cells
2. Construct dual graph of decomposition
3. Find path in graph
4. Transform path in graph to path in configuration space



Outline

A Point Robot

Start with a simple case

- problem reduces to motion planning for point robot in configuration space

A Point Robot

Start with a simple case

- problem reduces to motion planning for point robot in configuration space
- For a point Robot, the work space and the configuration space are identical

A Point Robot

Start with a simple case

- problem reduces to motion planning for point robot in configuration space
- For a point Robot, the work space and the configuration space are identical
- Marking:
    - denote the robot by R

A Point Robot

Start with a simple case

- problem reduces to motion planning for point robot in configuration space
- For a point Robot, the work space and the configuration space are identical
- Marking:
  - denote the robot by R
  - denote the obstacle by $P_1, \cdots, P_t$

A Point Robot

Start with a simple case

- problem reduces to motion planning for point robot in configuration space
- For a point Robot, the work space and the configuration space are identical
- Marking:
  - denote the robot by R
  - denote the obstacle by $P_1, \cdots, P_t$
- The obstacles are polygons with disjoint interiors, whose total number of vertices is denoted by n.

A Point Robot

Start with a simple case

- problem reduces to motion planning for point robot in configuration space
- For a point Robot, the work space and the configuration space are identical
- Marking:
  - denote the robot by R
  - denote the obstacle by $P_1, \cdots, P_t$
- The obstacles are polygons with disjoint interiors, whose total number of vertices is denoted by n.

We will construct a data structure storing a representation of the free space

Simplification

Assumptions

- To simplify we restrict the motion of the robot to a large bounding box B that contains the set of polygons

Simplification

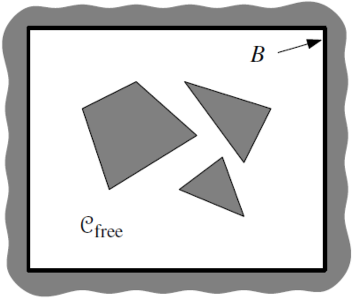
Assumptions

- To simplify we restrict the motion of the robot to a large bounding box B that contains the set of polygons
- $C_{free} = B \setminus \bigcup_{i=1}^{T} P_i$

Simplification

Assumptions

- To simplify we restrict the motion of the robot to a large bounding box B that contains the set of polygons
- $C_{free} = B \setminus \bigcup_{i=1}^{T} P_i$



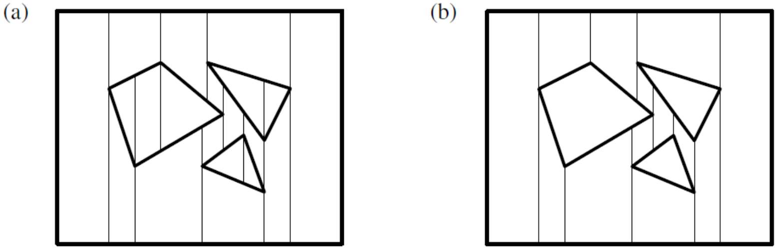
Algorithm ComputeFreeSpace(S)

Algorithm COMPUTEFREESPACE(S)
Input. A set S of disjoint polygons.
Output. A trapezoidal map of $\mathcal{C}_{\text{free}}(\mathcal{R}, S)$ for a point robot $\mathcal{R}$.

1. Let E be the set of edges of the polygons in S.
2. Compute the trapezoidal map $\mathcal{T}(E)$ with algorithm TRAPEZOIDALMAP described in Chapter 6.
3. Remove the trapezoids that lie inside one of the polygons from $\mathcal{T}(E)$ and return the resulting subdivision.

Trapezoidal map

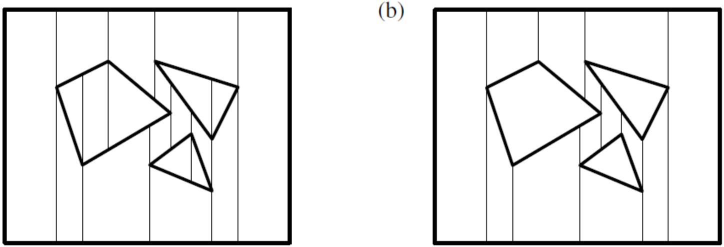
$T(C_{free})$



Trapezoidal map

$T(C_{free})$

(a) (b)



How do we find the trapezoids inside the obstacles?

we know for each trapezoid the edge that bounds it from the top,therefore it suffices to check whether the edge bounds the obstacle from above or from below

Complexity of Algorithm ComputeFreeSpace

Algorithm COMPUTEFREESPACE(S)
Input. A set S of disjoint polygons.
Output. A trapezoidal map of $\mathcal{C}_{\text{free}}(\mathcal{R}, S)$ for a point robot $\mathcal{R}$.

1. Let E be the set of edges of the polygons in S.
2. Compute the trapezoidal map $\mathcal{T}(E)$ with algorithm TRAPEZOIDALMAP described in Chapter 6.
3. Remove the trapezoids that lie inside one of the polygons from $\mathcal{T}(E)$ and return the resulting subdivision.

O(n logn)

Complexity of Algorithm ComputeFreeSpace

Algorithm COMPUTEFREESPACE(S)
Input. A set S of disjoint polygons.
Output. A trapezoidal map of $\mathcal{C}_{\text{free}}(\mathcal{R}, S)$ for a point robot $\mathcal{R}$.
1. Let E be the set of edges of the polygons in S.
2. Compute the trapezoidal map $\mathcal{T}(E)$ with algorithm TRAPEZOIDALMAP described in Chapter 6.
3. Remove the trapezoids that lie inside one of the polygons from $\mathcal{T}(E)$ and return the resulting subdivision.

O(1) for any trapezoid and there is at most 3n +1 trapezoid $\Rightarrow$ O(n)

Complexity of Algorithm ComputeFreeSpace

Algorithm COMPUTEFREESPACE(S)
Input. A set S of disjoint polygons.
Output. A trapezoidal map of $\mathcal{C}_{\text{free}}(\mathcal{R}, S)$ for a point robot $\mathcal{R}$.

1. Let E be the set of edges of the polygons in S.
2. Compute the trapezoidal map $\mathcal{T}(E)$ with algorithm TRAPEZOIDALMAP described in Chapter 6.
3. Remove the trapezoids that lie inside one of the polygons from $\mathcal{T}(E)$ and return the resulting subdivision.

O(1) for any trapezoid and there is at most 3n +1 trapezoid $\Rightarrow$ O(n)

Lemma 13.1

A trapezoidal map of the free configuration space for a point robot moving among a set of disjoint polygonal obstacles with n edges in total can be computed by a randomized algorithm in $O(n \log n)$ expected time.

Finding a path

How do we use $T(C_{free})$ to find a path from P_{start} to P_{goal}?

Finding a path

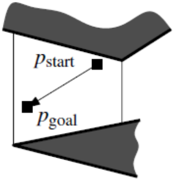
How do we use $T(C_{free})$ to find a path from P_{start} to P_{goal}?

1. If P_{start} and P_{goal} are in the same trapezoid of the map: the path is a straight line.

Finding a path

How do we use $T(C_{free})$ to find a path from P_{start} to P_{goal}?

1. If P_{start} and P_{goal} are in the same trapezoid of the map: the path is a straight line.



Finding a path

How do we use $T(C_{free})$ to find a path from P_{start} to P_{goal}?

1. If P_{start} and P_{goal} are in the same trapezoid of the map: the path is a straight line.
2. If the start and goal position are in different trapezoids: To guide the motion across trapezoids we construct a road map through the free space.

How do to construct the road map?

Step

1. Place a node in the center of each trapezoid

How do to construct the road map?

Step

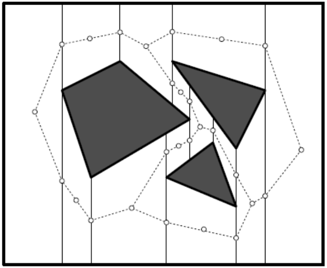
1. Place a node in the center of each trapezoid
2. Place a node in the middle of each vertical extension

How do to construct the road map?

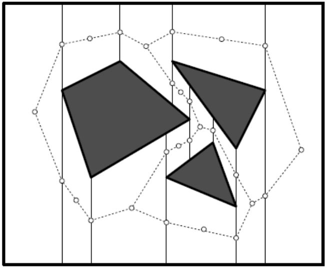
Step

1. Place a node in the center of each trapezoid
2. Place a node in the middle of each vertical extension
3. There is an arc between two nodes if and only if one node is in the center of a trapezoid and the other node is on the boundary of that same trapezoid

Road Map



Road Map



Constructed Time

The road map g_{road} can be constructed in $O(n)$ time by traversing the doubly-connected edge list of $T(C_{free})$

To Use the Road Map

Finding a path

We can use the road map, together with the trapezoidal map, to plan a motion from a start to a goal position.
To this end:

- Determine the trapezoids $\triangle_{start}$ and $\triangle_{goal}$ containing these points.

To Use the Road Map

Finding a path

We can use the road map, together with the trapezoidal map, to plan a motion from a start to a goal position.
To this end:

- Determine the trapezoids $\triangle_{start}$ and $\triangle_{goal}$ containing these points.
- if they are the same trapezoid: motion is only in a straight line

To Use the Road Map

Finding a path

We can use the road map, together with the trapezoidal map, to plan a motion from a start to a goal position.
To this end:

- Determine the trapezoids $\triangle_{start}$ and $\triangle_{goal}$ containing these points.
- if they are the same trapezoid: motion is only in a straight line
- Otherwise, let v_{start} and v_{goal} be the nodes of g_{road} that have been placed in the center of these trapezoids.
  The path will construct now consists of three parts

Summarize

The path will construct now consists of three parts:

Summarize

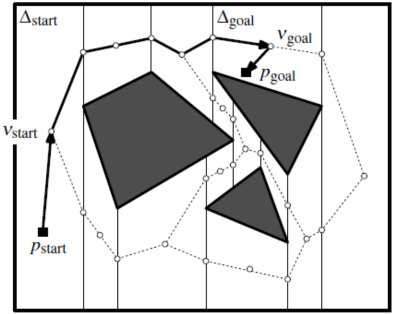
The path will construct now consists of three parts:

1. a straight-line motion from P_{start} to v_{start}

Summarize

The path will construct now consists of three parts:

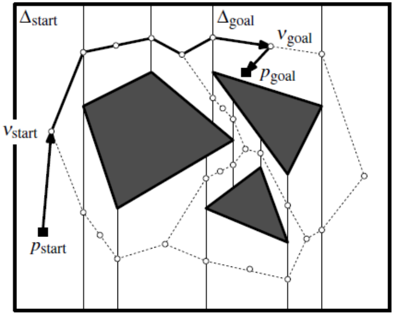
1. a straight-line motion from P_{start} to v_{start}



Summarize

The path will construct now consists of three parts:

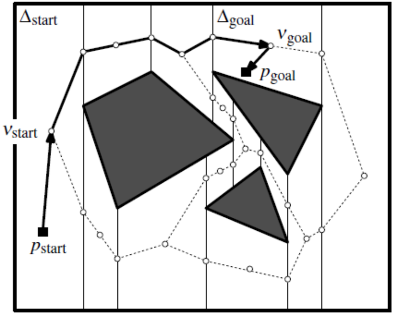
1. a straight-line motion from P_{start} to v_{start}
2. a path from v_{start} to v_{goal} along the arcs of the road map (with breadth-first search)



Summarize

The path will construct now consists of three parts:

1. a straight-line motion from P_{start} to v_{start}
2. a path from v_{start} to v_{goal} along the arcs of the road map (with breadth-first search)
3. a straight-line motion from v_{goal} to P_{goal}



Algorithm ComputePath

Algorithm COMPUTEPATH($\mathcal{T}(\mathcal{C}_{free}), \mathcal{G}_{road}, p_{start}, p_{goal}$)

Input. The trapezoidal map $\mathcal{T}(\mathcal{C}_{free})$ of the free space, the road map $\mathcal{G}_{road}$, a start position p_{start}, and goal position p_{goal}.

Output. A path from p_{start} to p_{goal} if it exists. If a path does not exist, this fact is reported.

1. Find the trapezoid Δ_{start} containing p_{start} and the trapezoid Δ_{goal} containing p_{goal}.
2. **if** Δ_{start} or Δ_{goal} does not exist
3. **then** Report that the start or goal position is in the forbidden space.
4. **else** Let v_{start} be the node of $\mathcal{G}_{road}$ in the center of Δ_{start}.
5. Let v_{goal} be the node of $\mathcal{G}_{road}$ in the center of Δ_{goal}.
6. Compute a path in $\mathcal{G}_{road}$ from v_{start} to v_{goal} using breadth-first search.
7. **if** there is no such path
8. **then** Report that there is no path from p_{start} to p_{goal}.
9. **else** Report the path consisting of a straight-line motion from p_{start} to v_{start}, the path found in $\mathcal{G}_{road}$, and a straight-line motion from v_{goal} to p_{goal}.

Algorithm ComputePath

Algorithm ComputePath($\mathcal{T}(\mathcal{C}_{\text{free}}), \mathcal{G}_{\text{road}}, p_{\text{start}}, p_{\text{goal}}$)

Input. The trapezoidal map $\mathcal{T}(\mathcal{C}_{\text{free}})$ of the free space, the road map $\mathcal{G}_{\text{road}}$, a start position p_{start}, and goal position p_{goal}.

Output. A path from p_{start} to p_{goal} if it exists. If a path does not exist, this fact is reported.

1. Find the trapezoid Δ_{start} containing p_{start} and the trapezoid Δ_{goal} containing p_{goal}.
2. **if** Δ_{start} or Δ_{goal} does not exist
3. **then** Report that the start or goal position is in the forbidden space.
4. **else** Let v_{start} be the node of $\mathcal{G}_{\text{road}}$ in the center of Δ_{start}.
5. Let v_{goal} be the node of $\mathcal{G}_{\text{road}}$ in the center of Δ_{goal}.
6. Compute a path in $\mathcal{G}_{\text{road}}$ from v_{start} to v_{goal} using breadth-first search.
7. **if** there is no such path
8. **then** Report that there is no path from p_{start} to p_{goal}.
9. **else** Report the path consisting of a straight-line motion from p_{start} to v_{start}, the path found in $\mathcal{G}_{\text{road}}$, and a straight-line motion from v_{goal} to p_{goal}.

O(log n)

Algorithm ComputePath

Algorithm COMPUTEPATH($\mathcal{T}(\mathcal{C}_{\text{free}}), \mathcal{G}_{\text{road}}, p_{\text{start}}, p_{\text{goal}}$)

Input. The trapezoidal map $\mathcal{T}(\mathcal{C}_{\text{free}})$ of the free space, the road map $\mathcal{G}_{\text{road}}$, a start position p_{start}, and goal position p_{goal}.

Output. A path from p_{start} to p_{goal} if it exists. If a path does not exist, this fact is reported.

1. Find the trapezoid Δ_{start} containing p_{start} and the trapezoid Δ_{goal} containing p_{goal}.
2. **if** Δ_{start} or Δ_{goal} does not exist
3.   **then** Report that the start or goal position is in the forbidden space.
4.   **else** Let v_{start} be the node of $\mathcal{G}_{\text{road}}$ in the center of Δ_{start}.
5.   Let v_{goal} be the node of $\mathcal{G}_{\text{road}}$ in the center of Δ_{goal}.
6.   Compute a path in $\mathcal{G}_{\text{road}}$ from v_{start} to v_{goal} using breadth-first search. → **O(n)**
7.   **if** there is no such path
8.   **then** Report that there is no path from p_{start} to p_{goal}.
9.   **else** Report the path consisting of a straight-line motion from p_{start} to v_{start}, the path found in $\mathcal{G}_{\text{road}}$, and a straight-line motion from v_{goal} to p_{goal}.

Algorithm ComputePath

Algorithm COMPUTEPATH($\mathcal{T}(\mathcal{C}_{\text{free}}), \mathcal{G}_{\text{road}}, p_{\text{start}}, p_{\text{goal}}$)

Input. The trapezoidal map $\mathcal{T}(\mathcal{C}_{\text{free}})$ of the free space, the road map $\mathcal{G}_{\text{road}}$, a start position p_{start}, and goal position p_{goal}.

Output. A path from p_{start} to p_{goal} if it exists. If a path does not exist, this fact is reported.

1. Find the trapezoid Δ_{start} containing p_{start} and the trapezoid Δ_{goal} containing p_{goal}.
2. **if** Δ_{start} or Δ_{goal} does not exist
3.   **then** Report that the start or goal position is in the forbidden space.
4.   **else** Let v_{start} be the node of $\mathcal{G}_{\text{road}}$ in the center of Δ_{start}.
5.   Let v_{goal} be the node of $\mathcal{G}_{\text{road}}$ in the center of Δ_{goal}.
6.   Compute a path in $\mathcal{G}_{\text{road}}$ from v_{start} to v_{goal} using breadth-first search.
7.   **if** there is no such path
8.   **then** Report that there is no path from p_{start} to p_{goal}.
9.   **else** Report the path consisting of a straight-line motion from p_{start} to v_{start}, the path found in $\mathcal{G}_{\text{road}}$, and a straight-line motion from v_{goal} to p_{goal}.

O(n)

Correctness of Algorithm

Are the reported path always collision-free?

Any path we report must be collisionfree, since it consists of segments inside trapezoids and all trapezoids are in the free space.

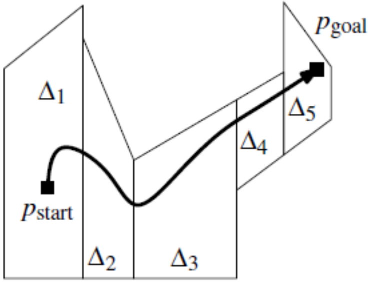
Correctness of Algorithm

Are the reported path always collision-free?

Any path we report must be collisionfree, since it consists of segments inside trapezoids and all trapezoids are in the free space.

Do we always find a collision-free path if one exists?

- suppose that there is a collision-free path from p_{start} to p_{goal}
- The path from p_{start} to p_{goal} must cross a sequence of trapezoids $\triangle_1, ..., \triangle_k$
- Let v_i be the node of g_{road} that is in the center of $\triangle_i$



Conclusion

Theorem 13.2

Let R be a point robot moving among a set S of polygonal obstacles with n edges in total. We can preprocess S in $O(nlogn)$ 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

Conclusion

Theorem 13.2

Let R be a point robot moving among a set S of polygonal obstacles with n edges in total. We can preprocess S in $O(nlogn)$ 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

The path computed by the algorithm is collision-free, but we can give no guarantee that the path does not make large detours

Outline

Motion Planning problem for a Polygon Robot

Recall

- We assume that the robot R is convex, and for the moment we also assume that the obstacles are convex

Motion Planning problem for a Polygon Robot

Recall

- We assume that the robot R is convex, and for the moment we also assume that the obstacles are convex
- $R(x, y)$ to denote the placement of R with its reference point at (x, y).

Motion Planning problem for a Polygon Robot

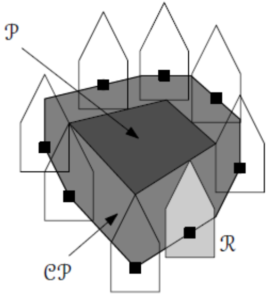
Recall

- We assume that the robot R is convex, and for the moment we also assume that the obstacles are convex
- $R(x, y)$ to denote the placement of R with its reference point at (x, y).
- C -obstacle, of an obstacle P and the robot R is defined as the set of points in configuration space such that the corresponding placement of R intersects P.

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

C -obstacle

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



Minkowski Sum

Definition

The Minkowski sum of two sets $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$ is:

$$S_1 \oplus S_2 := \{p + q : p \in S_1, q \in S_2\}$$

$$p + q := (p_x + q_x, p_y + q_y)$$

Minkowski Sum

Definition

The Minkowski sum of two sets $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$ is:

$$S_1 \oplus S_2 := \{p + q : p \in S_1, q \in S_2\}$$

$$p + q := (p_x + q_x, p_y + q_y)$$

Notation

For a point $p = (p_x, p_y)$ we define $-p := (-p_x, -p_y)$, and for a set S we define $-S := \{-p : p \in S\}$

Minkowski Sum

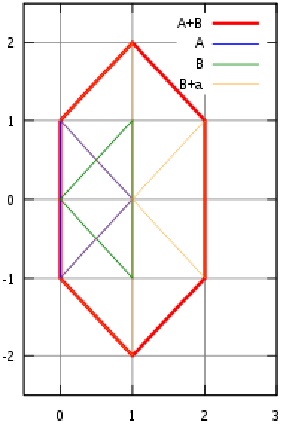
Example

$$A = \{(1,0),(0,1),(0,1)\}$$

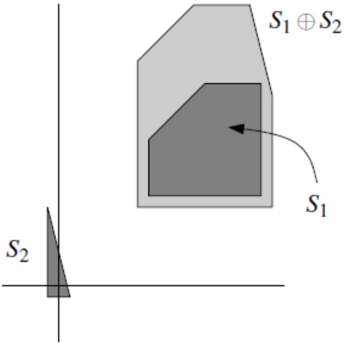
$$B = \{(0,0),(1,1),(1,1)\}$$

then the Minkowski sum is:

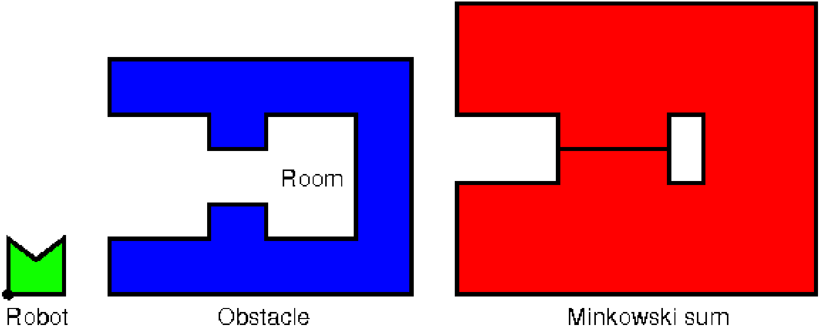
$$A \oplus B = \{(1,0),(2,1),(2,1),(0,1),(1,2),$$
$$(1,0),(0,1),(1,0),(1,2)\}$$



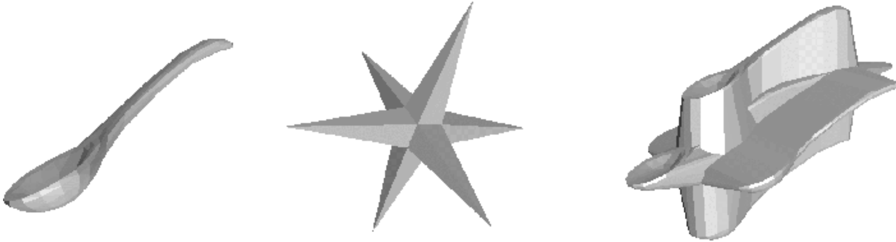
Example for Minkowski Sum



Example for Minkowski Sum



Example for Minkowski Sum



Theorem 13.3

Theorem 13.3

Let R be a planar, translating robot and let P be an obstacle. Then the C -obstacle of P is $P \oplus (-R(0,0))$.

Theorem 13.3

Theorem 13.3

Let R be a planar, translating robot and let P be an obstacle. Then the C -obstacle of P is $P \oplus (-R(0,0))$.

Proof

$R(x,y)$ Intersect $P \Longleftrightarrow (x,y) \in P \oplus (-R(0,0))$

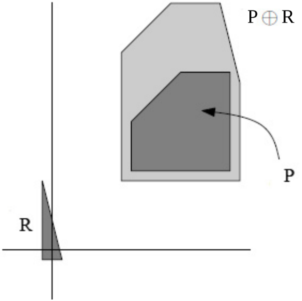
Theorem 13.3

Theorem 13.3

Let R be a planar, translating robot and let P be an obstacle. Then the C -obstacle of P is $P \oplus (-R(0,0))$.

Proof

$R(x,y)$ Intersect $P \iff (x,y) \in P \oplus (-R(0,0))$



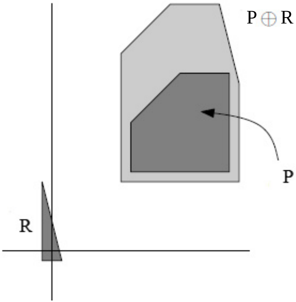
Theorem 13.3

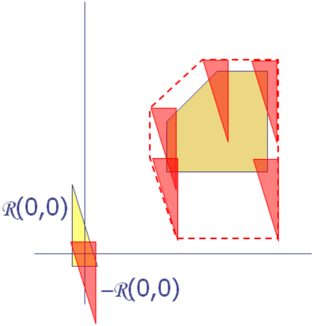
Theorem 13.3

Let R be a planar, translating robot and let P be an obstacle. Then the C -obstacle of P is $P \oplus (-R(0,0))$.

Proof

$R(x,y)$ Intersect $P \iff (x,y) \in P \oplus (-R(0,0))$





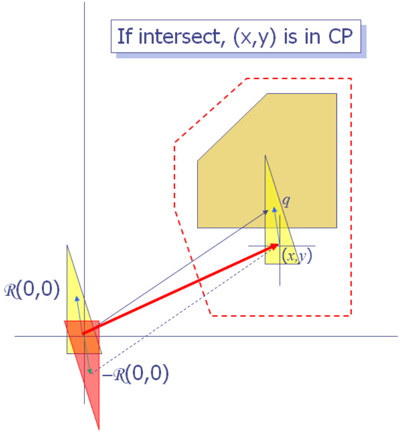
Proof Of Theorem 13.3

If intersect,(x, y) is in CP

A Point q in intersection $\Rightarrow$
$q \in R(x, y) \wedge q \in P$

$\because q \in R(x, y) \Rightarrow$
$(q_x - x, q_y - y) \in R(0, 0)$
$(-q_x + x, -q_y + y) \in -R(0, 0)$

$\because q \in P \Rightarrow$
$(q_x, q_y) + (-q_x + x, -q_y + y) =$
$(x, y) \in P \oplus (-R(0, 0))$



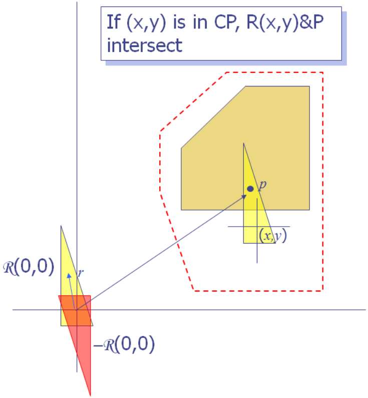
Proof Of Theorem 13.3

If (x, y) is in CP, $R(x, y)$ and P intersect

$(x, y) \in P \oplus (-R(0,0)) \Longrightarrow$

$\exists (r_x, r_y) \in R(0,0) \wedge \exists (p_x, p_y) \in P$
Such that: $(x, y) = (p_x - r_x, p_y - r_y))$

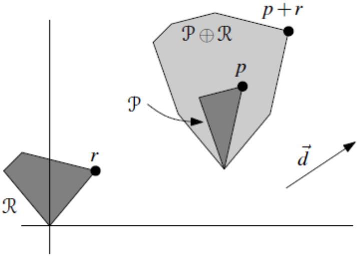
That is:
$p_x = x + r_x$
$p_y = y + r_y$



Observation

Observation 13.4

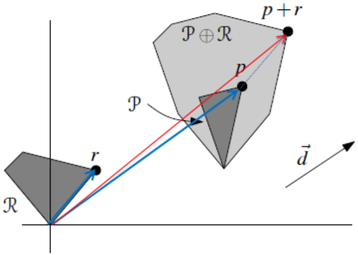
Let P and R be two objects in the plane, and let $CP := P \oplus R$.
An extreme point in direction $\vec{d}$ on CP is the sum of extreme points in direction $\vec{d}$ on P and R.



Observation

Observation 13.4

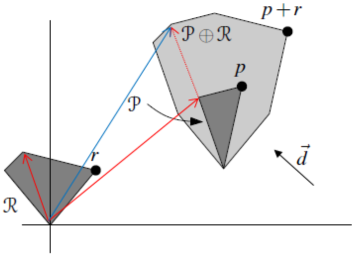
Let P and R be two objects in the plane, and let $CP := P \oplus R$.
An extreme point in direction $\vec{d}$ on CP is the sum of extreme points in direction $\vec{d}$ on P and R.



Observation

Observation 13.4

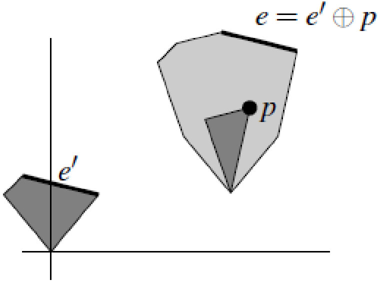
Let P and R be two objects in the plane, and let $CP := P \oplus R$.
An extreme point in direction $\vec{d}$ on CP is the sum of extreme points in direction $\vec{d}$ on P and R.



Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

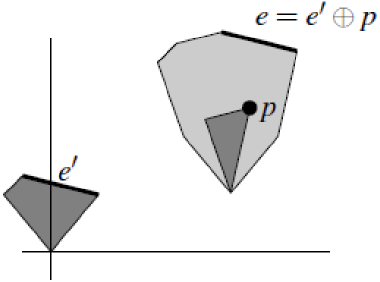


Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

Proof



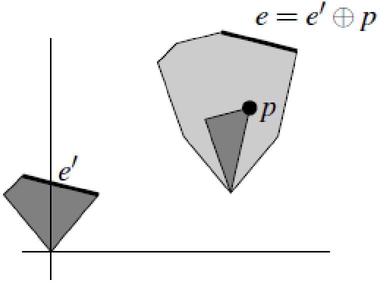
Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

Proof

- First part follows directly from the definition.(sliding argument)



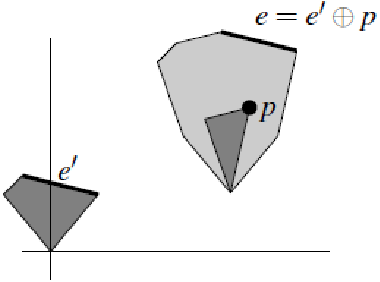
Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

Proof

- First part follows directly from the definition.(sliding argument)
- consider an edge e that is extreme in the direction of its outer normal.



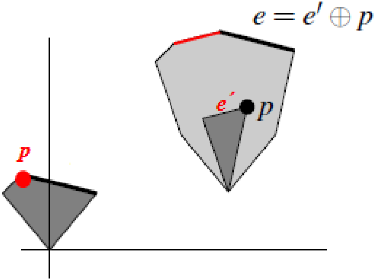
Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

Proof

- First part follows directly from the definition.(sliding argument)
- consider an edge e that is extreme in the direction of its outer normal.



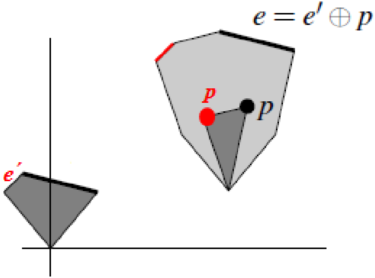
Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

Proof

- First part follows directly from the definition.(sliding argument)
- consider an edge e that is extreme in the direction of its outer normal.



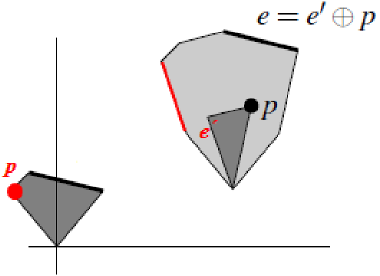
Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

Proof

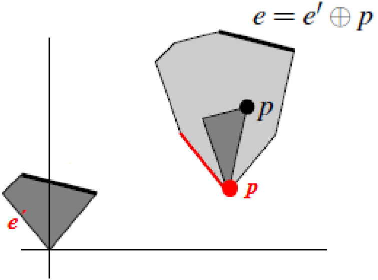
- First part follows directly from the definition.(sliding argument)
- consider an edge e that is extreme in the direction of its outer normal.



Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.



Proof

- First part follows directly from the definition.(sliding argument)
- consider an edge e that is extreme in the direction of its outer normal.

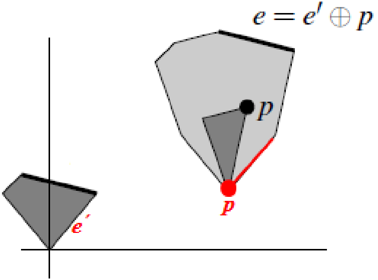
Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

Proof

- First part follows directly from the definition.(sliding argument)
- consider an edge e that is extreme in the direction of its outer normal.



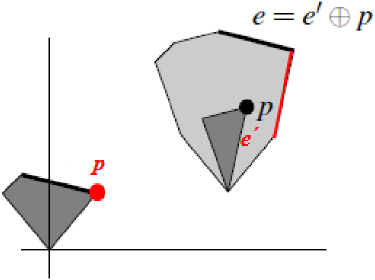
Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

Proof

- First part follows directly from the definition.(sliding argument)
- consider an edge e that is extreme in the direction of its outer normal.



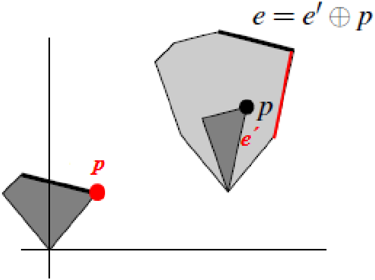
Theorem

Theorem 13.5

Let P and R be two convex polygons with n and m edges, respectively. Then the Minkowski sum $P \oplus R$ is a convex polygon with at most $n + m$ edges.

Proof

- First part follows directly from the definition.(sliding argument)
- consider an edge e that is extreme in the direction of its outer normal.



Question

Why we say **at most**?

A pair of Pseudodiscs

Definition

Consider two planar objects o_1 and o_2, each bounded by a simple closed curve.
the pair o_1, o_2 is called a pair of pseudodiscs:

A pair of Pseudodiscs

Definition

Consider two planar objects o_1 and o_2, each bounded by a simple closed curve.
the pair o_1, o_2 is called a pair of pseudodiscs:

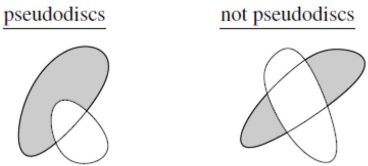
- **Definition 1:** if their boundaries ∂o_1 and ∂o_2 intersect in at most two points

A pair of Pseudodiscs

Definition

Consider two planar objects o_1 and o_2, each bounded by a simple closed curve.
the pair o_1, o_2 is called a pair of pseudodiscs:

- **Definition 1:** if their boundaries ∂o_1 and ∂o_2 intersect in at most two points



A pair of Pseudodiscs

Definition

Consider two planar objects o_1 and o_2, each bounded by a simple closed curve.
the pair o_1, o_2 is called a pair of pseudodiscs:

- **Definition 1:** if their boundaries ∂o_1 and ∂o_2 intersect in at most two points

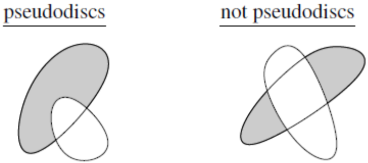
Degenerate Case

A pair of Pseudodiscs

Definition

Consider two planar objects o_1 and o_2, each bounded by a simple closed curve.
the pair o_1, o_2 is called a pair of pseudodiscs:

- **Definition 1:** if their boundaries ∂o_1 and ∂o_2 intersect in at most two points
- **Definition 2:** if $\partial o_1 \cap int(o_2)$ is connected and $\partial o_2 \cap int(o_1)$ is connected

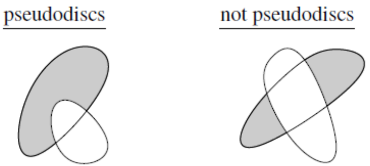


A pair of Pseudodiscs

Definition

Consider two planar objects o_1 and o_2, each bounded by a simple closed curve.
the pair o_1, o_2 is called a pair of pseudodiscs:

- **Definition 1:** if their boundaries ∂o_1 and ∂o_2 intersect in at most two points
- **Definition 2:** if $\partial o_1 \cap int(o_2)$ is connected and $\partial o_2 \cap int(o_1)$ is connected
- **Definition 3:** if $o_1 - o_2$ is connected and $o_2 - o_1$ is connected



Collection of pseudodiscs

Definition

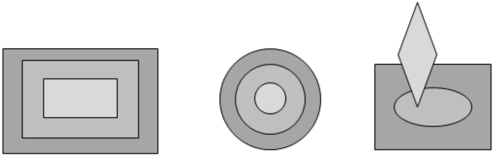
A collection of objects, each bounded by a simple closed curve, is called a collection of pseudodiscs if every pair of objects in the collection is a pair of pseudodiscs

Collection of pseudodiscs

Definition

A collection of objects, each bounded by a simple closed curve, is called a collection of pseudodiscs if every pair of objects in the collection is a pair of pseudodiscs

Collection of Pseudodiscs



Collection of pseudodiscs

Definition

A collection of objects, each bounded by a simple closed curve, is called a collection of pseudodiscs if every pair of objects in the collection is a pair of pseudodiscs

Note

pseudodisc property is about the way in which (the boundaries of) two objects can interact. It does not make sense to say of a single object that it is a pseudodisc.

Boundary crossing

Definition

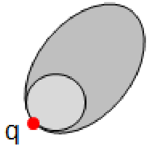
consider two polygons P and $P^{'}$.
an intersection point $p \in \partial P \cap \partial P^{'}$ is a boundary crossing if ∂P crosses from the interior of $P^{'}$ to the exterior of $P^{'}$ at p

Boundary crossing

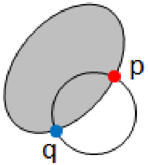
Definition

consider two polygons P and $P^{'}$.
an intersection point $p \in \partial P \cap \partial P^{'}$ is a boundary crossing if ∂P crosses from the interior of $P^{'}$ to the exterior of $P^{'}$ at p

p is not boundary crossing



p and q is boundary crossing



An important property of polygonal pseudodiscs

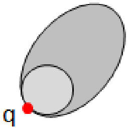
Observation 13.6

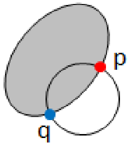
A pair of polygonal pseudodiscs P, P' defines at most two boundary crossings.

An important property of polygonal pseudodiscs

Observation 13.6

A pair of polygonal pseudodiscs P, P' defines at most two boundary crossings.





The more extreme of polygon

Definition

The more extreme of polygon

Definition

- Consider the pairs of convex polygons with disjoint interiors

The more extreme of polygon

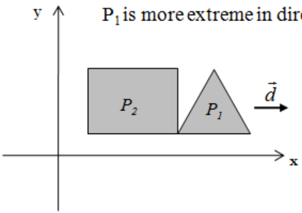
Definition

- Consider the pairs of convex polygons with disjoint interiors
- one polygon is more extreme in a direction $\vec{d}$ than another polygon if its extreme points lie further in that direction than the extreme points of the other polygon

The more extreme of polygon

Definition

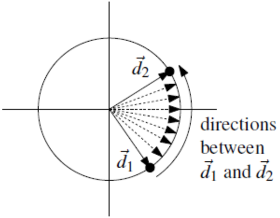
- Consider the pairs of convex polygons with disjoint interiors
- one polygon is more extreme in a direction $\vec{d}$ than another polygon if its extreme points lie further in that direction than the extreme points of the other polygon



extreme points for various directions

Definition

The range from a direction $\vec{d_1}$ to a direction $\vec{d_2}$ is defined as the directions corresponding to points in the counterclockwise circle segment from the point representing $\vec{d_1}$ to the point representing $\vec{d_2}$.



Observation

Observation 13.7

Let P_1 and P_2 be convex polygons with disjoint interiors.
Let $\vec{d}_1$ and $\vec{d}_2$ be directions in which P_1 is more extreme than P_2.

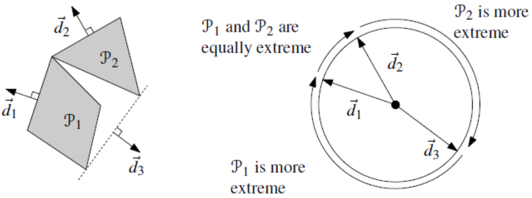
Then either P_1 is more extreme than P_2 in all directions in the range from $\vec{d}_1$ to $\vec{d}_2$, or it is more extreme in all directions in the range from $\vec{d}_2$ to $\vec{d}_1$.

Observation

Observation 13.7

Let P_1 and P_2 be convex polygons with disjoint interiors.
Let $\vec{d_1}$ and $\vec{d_2}$ be directions in which P_1 is more extreme than P_2.

Then either P_1 is more extreme than P_2 in all directions in the range from $\vec{d_1}$ to $\vec{d_2}$, or it is more extreme in all directions in the range from $\vec{d_2}$ to $\vec{d_1}$.



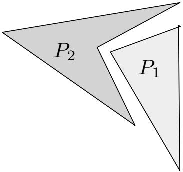
Observation

Observation 13.7

Let P_1 and P_2 be convex polygons with disjoint interiors.
Let $\vec{d_1}$ and $\vec{d_2}$ be directions in which P_1 is more extreme than P_2.

Then either P_1 is more extreme than P_2 in all directions in the range from $\vec{d_1}$ to $\vec{d_2}$, or it is more extreme in all directions in the range from $\vec{d_2}$ to $\vec{d_1}$.

Not holed for non- convex polygons



Minkowski sums forms a collection of pseudodiscs

Theorem 13.8

Let P_1 and P_2 be two convex polygons with disjoint interiors, and let R be another convex polygon. Then the two Minkowski sums $P_1 \oplus R$ and $P_2 \oplus R$ are pseudodiscs.

Minkowski sums forms a collection of pseudodiscs

Theorem 13.8

Let P_1 and P_2 be two convex polygons with disjoint interiors, and let R be another convex polygon. Then the two Minkowski sums $P_1 \oplus R$ and $P_2 \oplus R$ are pseudodiscs.

Proof

- Define:

$$CP_1 := P_1 \oplus R$$

$$CP_2 := P_2 \oplus R$$

Minkowski sums forms a collection of pseudodiscs

Theorem 13.8

Let P_1 and P_2 be two convex polygons with disjoint interiors, and let R be another convex polygon. Then the two Minkowski sums $P_1 \oplus R$ and $P_2 \oplus R$ are pseudodiscs.

Proof

- Define:

$$CP_1 := P_1 \oplus R$$

$$CP_2 := P_2 \oplus R$$

- By symmetry, it suffices to show that $\partial CP_1 \cap int(CP_2)$ is connected

Minkowski sums forms a collection of pseudodiscs

Theorem 13.8

Let P_1 and P_2 be two convex polygons with disjoint interiors, and let R be another convex polygon. Then the two Minkowski sums $P_1 \oplus R$ and $P_2 \oplus R$ are pseudodiscs.

Proof

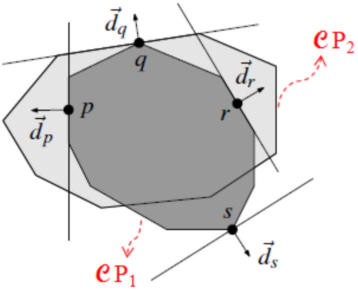
- Define:

$$CP_1 := P_1 \oplus R$$

$$CP_2 := P_2 \oplus R$$

- By symmetry, it suffices to show that $\partial CP_1 \cap int(CP_2)$ is connected
- Proof with Contradiction

$\partial \mathcal{C}P_1 \cap \mathrm{int}(\mathcal{C}P_2)$ is not connect



complexity of union

Theorem 13.9

Let S be a collection of convex polygonal pseudodiscs with n edges in total. Then the complexity of their union is at most $2n$

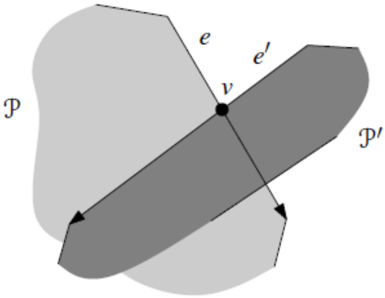
complexity of union

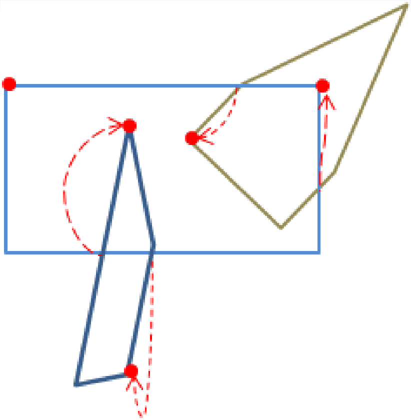
Theorem 13.9

Let S be a collection of convex polygonal pseudodiscs with n edges in total. Then the complexity of their union is at most $2n$

Proof

We prove the bound by charging every vertex of the union to a pseudodisc vertex in such a way that any pseudodisc vertex is charged at most two times


e
e'
v
$\mathcal{P}$
$\mathcal{P}'$



Compute the Minkowski sum

A very simple algorithm

Compute the Minkowski sum

A very simple algorithm

1. For each pair v, w of vertices, with $v \in P$ and $w \in R$,compute $v + w$

Compute the Minkowski sum

A very simple algorithm

1. For each pair v, w of vertices, with $v \in P$ and $w \in R$,compute $v + w$
2. compute the convex hull of all these sums

Compute the Minkowski sum

A very simple algorithm

1. For each pair v, w of vertices, with $v \in P$ and $w \in R$,compute $v + w$
2. compute the convex hull of all these sums

Problems

this algorithm inefficient when the polygons have many vertices, because it looks at every pair of vertices.

Compute the Minkowski sum

an alternative algorithm and easy to implement

Compute the Minkowski sum

an alternative algorithm and easy to implement

- only looks at pairs of vertices that are extreme in the same direction

Compute the Minkowski sum

an alternative algorithm and easy to implement

- only looks at pairs of vertices that are extreme in the same direction
- it run in linear time

angle

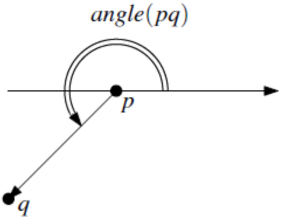
Definition

In the algorithm we use the notation angle (pq) to denote the angle that the vector $\vec{pq}$ makes with the positive x-axis.

angle

Definition

In the algorithm we use the notation angle (pq) to denote the angle that the vector $\vec{pq}$ makes with the positive x-axis.



Algorithm MinkowskiSum

Algorithm MinkowskiSum($\mathcal{P}, \mathcal{R}$)

Input. A convex polygon $\mathcal{P}$ with vertices $v_1, \ldots, v_n$, and a convex polygon $\mathcal{R}$ with vertices $w_1, \ldots, w_m$. The lists of vertices are assumed to be in counterclockwise order, with v_1 and w_1 being the vertices with smallest y-coordinate (and smallest x-coordinate in case of ties).

Output. The Minkowski sum $\mathcal{P} \oplus \mathcal{R}$.

$i \leftarrow 1$; $j \leftarrow 1$

$v_{n+1} \leftarrow v_1$; $v_{n+2} \leftarrow v_2$; $w_{m+1} \leftarrow w_1$; $w_{m+2} \leftarrow w_2$

repeat

  Add $v_i + w_j$ as a vertex to $\mathcal{P} \oplus \mathcal{R}$.

  if $angle(v_i v_{i+1}) < angle(w_j w_{j+1})$

   then $i \leftarrow (i+1)$

   else if $angle(v_i v_{i+1}) > angle(w_j w_{j+1})$

     then $j \leftarrow (j+1)$

     else $i \leftarrow (i+1)$; $j \leftarrow (j+1)$

until $i = n+1$ **and** $j = m+1$

Algorithm MinkowskiSum

Algorithm MINKOWSKISUM($\mathcal{P},\mathcal{R}$)

Input. A convex polygon $\mathcal{P}$ with vertices $v_1,\ldots,v_n$, and a convex polygon $\mathcal{R}$ with vertices $w_1,\ldots,w_m$. The lists of vertices are assumed to be in counterclockwise order, with v_1 and w_1 being the vertices with smallest y-coordinate (and smallest x-coordinate in case of ties).

Output. The Minkowski sum $\mathcal{P}\oplus\mathcal{R}$.

$i \leftarrow 1$; $j \leftarrow 1$

$v_{n+1} \leftarrow v_1$; $v_{n+2} \leftarrow v_2$; $w_{m+1} \leftarrow w_1$; $w_{m+2} \leftarrow w_2$

repeat

  Add $v_i + w_j$ as a vertex to $\mathcal{P}\oplus\mathcal{R}$.

  if $angle(v_iv_{i+1}) < angle(w_jw_{j+1})$

   then $i \leftarrow (i+1)$

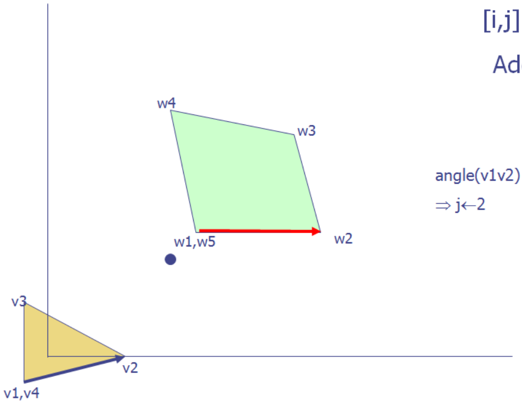
   else **if** $angle(v_iv_{i+1}) > angle(w_jw_{j+1})$ → O(n+m)

    then $j \leftarrow (j+1)$

    else $i \leftarrow (i+1)$; $j \leftarrow (j+1)$

until $i = n+1$ **and** $j = m+1$

A sample of implement



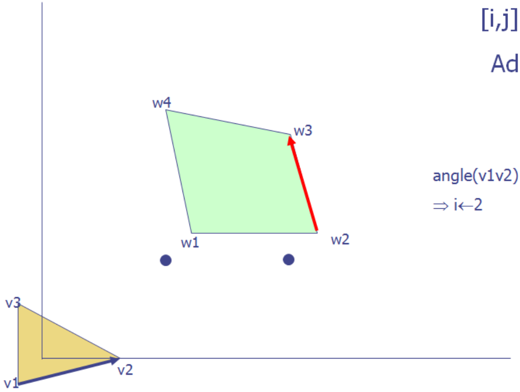
$[i,j] = (1,1)$

Add v_1+w_1

angle(v1v2) > angle(w1w2)

$\Rightarrow j \leftarrow 2$

A sample of implement



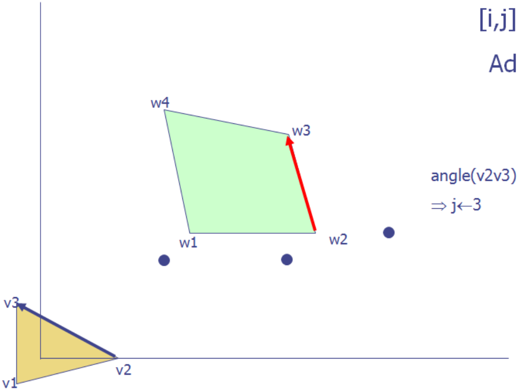
$[i,j] = (1,2)$

Add v_1+w_2

angle(v1v2) < angle(w2w3)

$\Rightarrow i \leftarrow 2$

A sample of implement



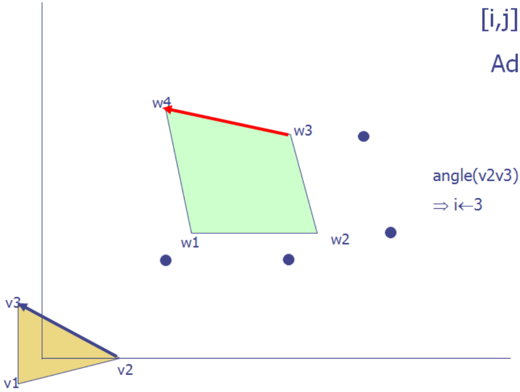
$[i,j] = (2,2)$

Add v_2+w_2

angle(v2v3) > angle(w2w3)

$\Rightarrow j \leftarrow 3$

A sample of implement



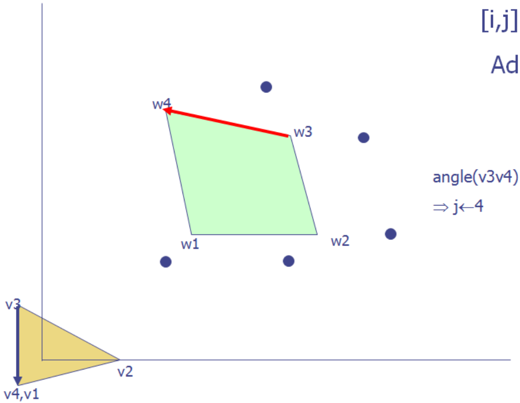
[i,j] = (2,3)

Add v_2+w_3

angle(v2v3) < angle(w3w4)

$\Rightarrow$ i←3

A sample of implement



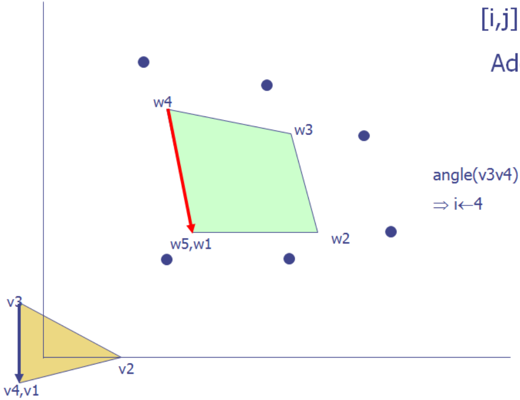
[i,j] = (3,3)

Add v_3+w_3

angle(v3v4) > angle(w3w4)

$\Rightarrow$ j$\leftarrow$4

A sample of implement



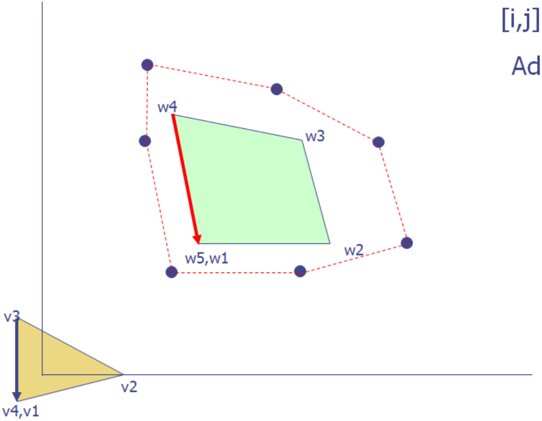
[i,j] = (3,4)

Add v_3+w_4

angle(v3v4) < angle(w4w5)

$\Rightarrow$ i$\leftarrow$4

A sample of implement



[i,j] = (4,4)

Add v_4+w_4

Theorem

Theorem 13.10

The Minkowski sum of two convex polygons with n and m vertices, respectively, can be computed in $O(n + m)$ time.

The Minkowski Sum for non-convex polygons

Compute the convex $R \oplus$ Non-convex P

The Minkowski Sum for non-convex polygons

Compute the convex $R \oplus$ Non-convex P

- the following equality holds for any sets S_1, S_2, and S_3:

$$S_1 \oplus (S_2 \cup S_3) = (S_1 \oplus S_2) \cup (S_1 \oplus S_3)$$

The Minkowski Sum for non-convex polygons

Compute the convex $R \oplus$ Non-convex P

- the following equality holds for any sets S_1, S_2, and S_3:

$$S_1 \oplus (S_2 \cup S_3) = (S_1 \oplus S_2) \cup (S_1 \oplus S_3)$$

- Therefor,we can do the following steps:

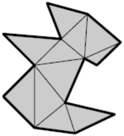
The Minkowski Sum for non-convex polygons

Compute the convex $R \oplus$ Non-convex P

- the following equality holds for any sets S_1, S_2, and S_3:

$$S_1 \oplus (S_2 \cup S_3) = (S_1 \oplus S_2) \cup (S_1 \oplus S_3)$$

- Therefor,we can do the following steps:
  1. Triangulate P into $t_1, ..., t_{n-2}$



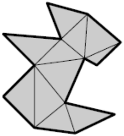
The Minkowski Sum for non-convex polygons

Compute the convex $R \oplus$ Non-convex P

- the following equality holds for any sets S_1, S_2, and S_3:

$$S_1 \oplus (S_2 \cup S_3) = (S_1 \oplus S_2) \cup (S_1 \oplus S_3)$$

- Therefor,we can do the following steps:
  1. Triangulate P into $t_1, \ldots, t_{n-2}$
  2. Compute $R \oplus t_1, \ldots, R \oplus t_{n-2}$



The Minkowski Sum for non-convex polygons

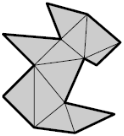
Compute the convex $R \oplus$ Non-convex P

- the following equality holds for any sets S_1, S_2, and S_3:

$$S_1 \oplus (S_2 \cup S_3) = (S_1 \oplus S_2) \cup (S_1 \oplus S_3)$$

- Therefor,we can do the following steps:
  1. Triangulate P into $t_1, \ldots, t_{n-2}$
  2. Compute $R \oplus t_1, \ldots, R \oplus t_{n-2}$
  3. Compute their union

$$P \oplus R = \bigcup_{i=1}^{n-2} t_i \oplus R$$

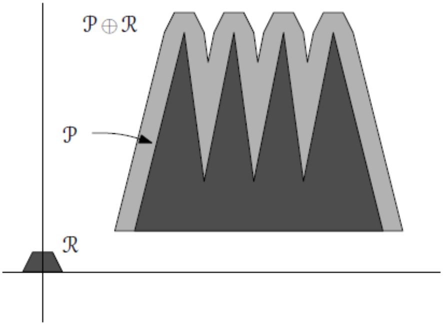


The Minkowski Sum for non-convex polygons

complexity of $P \oplus R$

- $t_i \oplus R$ is a convex polygon with at most $m + 3$ vertices(according to Theorem 13.5)
- the triangles have disjoint interiors, so the collection of Minkowski sums is a collection of pseudodiscs(according to Theorem 13.8)
- Hence, the complexity of their union is linear in the sum of their complexities(according to Theorem 13.9)
- This implies that the complexity of $P \oplus R$ is $2\left[(n-2)(m+3)\right] \in O(nm)$

The worst case



The Minkowski Sum for non-convex polygons

Compute the Non-Convex $R \oplus$ Non-convex P

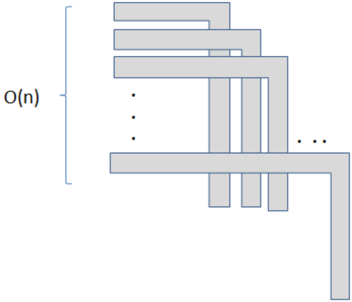
1. triangulate both polygons and get a collection of $n-2$ triangles t_i, and a collection of $m-2$ triangles u_j
2. Minkowski sum of P and R is now the union of the Minkowski sums of the pairs t_i, u_j
   Each sum $t_i \oplus u_j$ has constant complexity.
3. $P \oplus R$ is the union of $(n-2)(m-2)$ constant-complexity polygons

$$P \oplus R = \bigcup_{i=1}^{n-2} \bigcup_{j=1}^{m-2} t_i \oplus u_j$$

This implies that the total complexity of $P \oplus R$ is $O(n^2 m^2)$

The worst case

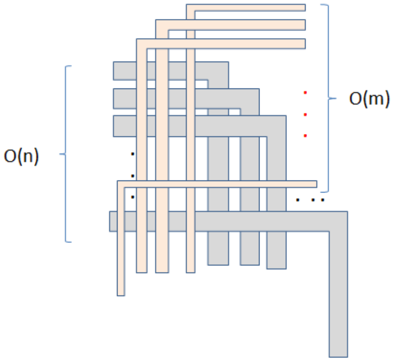
Complexity union: $o(n^2)$



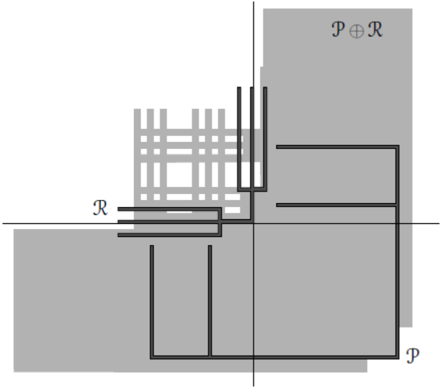
The worst case

Complexity : $o(n^2)$
Complexity : $o(m^2)$

$\Rightarrow$ Complexity = $O(n^2m^2)$



The worst case



summarize

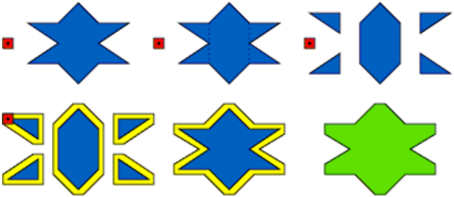
Theorem 13.11

Let P and R be polygons with n and m vertices, respectively.
The complexity of the Minkowski sum $P \oplus R$ is bounded as follows:

- it is $O(n+m)$ if both polygons are convex
- it is $O(nm)$ if one of the polygons is convex and one is non-convex
- it is $O(n^2m^2)$ if both polygons are non-convex

These bounds are tight in the worst case.

summarize



Outline

1. Introduction
2. Work Space and Configuration Space
3. A Point Robot
4. Minkowski Sums
5. **Translational Motion Planning**

Return to the planar motion planning problem

Theorem 13.12

Let R be a convex robot of constant complexity, translating among a set S of non-intersecting polygonal obstacles with a total of n edges. Then the complexity of the free configuration space $C_{free}(R, S)$ is $O(n)$

Return to the planar motion planning problem

Theorem 13.12

Let R be a convex robot of constant complexity, translating among a set S of non-intersecting polygonal obstacles with a total of n edges. Then the complexity of the free configuration space $C_{free}(R, S)$ is $O(n)$

Proof

Return to the planar motion planning problem

Theorem 13.12

Let R be a convex robot of constant complexity, translating among a set S of non-intersecting polygonal obstacles with a total of n edges. Then the complexity of the free configuration space $C_{free}(R, S)$ is $O(n)$

Proof

- Any triangle is convex, with disjoint interiors
- The free configuration space is the complement of the union of the C-obstacles of these triangles
- **Therefor:**

Return to the planar motion planning problem

Theorem 13.12

Let R be a convex robot of constant complexity, translating among a set S of non-intersecting polygonal obstacles with a total of n edges. Then the complexity of the free configuration space $C_{free}(R, S)$ is $O(n)$

Proof

- Any triangle is convex, with disjoint interiors
- The free configuration space is the complement of the union of the C-obstacles of these triangles
- **Therefor:**
  - Triangulate each obstacle polygon $\Longrightarrow$ get a set of $O(n)$ triangular,

Return to the planar motion planning problem

Theorem 13.12

Let R be a convex robot of constant complexity, translating among a set S of non-intersecting polygonal obstacles with a total of n edges. Then the complexity of the free configuration space $C_{free}(R, S)$ is $O(n)$

Proof

- Any triangle is convex, with disjoint interiors
- The free configuration space is the complement of the union of the C-obstacles of these triangles
- **Therefor:**
  - Triangulate each obstacle polygon $\Longrightarrow$ get a set of $O(n)$ triangular,
  - robot has constant complexity $\Longrightarrow$ any C-obstacle have constant complexity($m+3$ is constant)

Return to the planar motion planning problem

Theorem 13.12

Let R be a convex robot of constant complexity, translating among a set S of non-intersecting polygonal obstacles with a total of n edges. Then the complexity of the free configuration space $C_{free}(R, S)$ is $O(n)$

Proof

- Any triangle is convex, with disjoint interiors
- The free configuration space is the complement of the union of the C-obstacles of these triangles
- **Therefor:**
  - Triangulate each obstacle polygon $\Longrightarrow$ get a set of $O(n)$ triangular,
  - robot has constant complexity $\Longrightarrow$ any C-obstacle have constant complexity($m + 3$ is constant)
  - C-obstacles form a set of pseudodiscs $\Longrightarrow$ the union has linear complexity(total number of C-obstacle is n)

Algorithm to construct the free space

$C_{free} = B - C_{forb}$

- computing the free space C_{free}, we shall compute the forbidden space C_{forb}
- the free space is simply its complement.

Algorithm to construct the free space

$C_{free} = B - C_{forb}$

- computing the free space C_{free}, we shall compute the forbidden space C_{forb}
- the free space is simply its complement.
- Let $P_1, \ldots, P_n$ denote the triangles that we get when we triangulate the obstacles:

$$C_{forb} = \bigcup_{i=1}^{n} CP_i = \bigcup_{i=1}^{n} P_i \oplus (-R(0,0))$$

Therefor the problem convert to computing forbidden space

Algorithm Forbidden Space

Algorithm FORBIDDENSPACE($\mathcal{CP}_1, \dots, \mathcal{CP}_n$)
Input. A collection of $\mathcal{C}$-obstacles.
Output. The forbidden space $\mathcal{C}_{\text{forb}} = \bigcup_{i=1}^{n} \mathcal{CP}_i$.

1. **if** $n = 1$
2. **then return** $\mathcal{CP}_1$
3. **else** $\mathcal{C}^1_{\text{forb}} \leftarrow$ FORBIDDENSPACE($\mathcal{P}_1, \dots, \mathcal{P}_{\lceil n/2 \rceil}$)
4. $\mathcal{C}^2_{\text{forb}} \leftarrow$ FORBIDDENSPACE($\mathcal{P}_{\lceil n/2 \rceil + 1}, \dots, \mathcal{P}_n$)
5. Compute $\mathcal{C}_{\text{forb}} = \mathcal{C}^1_{\text{forb}} \cup \mathcal{C}^2_{\text{forb}}$.
6. **return** $\mathcal{C}_{\text{forb}}$

Algorithm Forbidden Space

Algorithm FORBIDDENSPACE($\mathcal{CP}_1, \ldots, \mathcal{CP}_n$)
Input. A collection of $\mathcal{C}$-obstacles.
Output. The forbidden space $\mathcal{C}_{\text{forb}} = \bigcup_{i=1}^{n} \mathcal{CP}_i$.
1. **if** $n = 1$
2. **then return** $\mathcal{CP}_1$
3. **else** $\mathcal{C}^1_{\text{forb}} \leftarrow$ FORBIDDENSPACE($\mathcal{P}_1, \ldots, \mathcal{P}_{\lceil n/2 \rceil}$)
4. $\mathcal{C}^2_{\text{forb}} \leftarrow$ FORBIDDENSPACE($\mathcal{P}_{\lceil n/2 \rceil + 1}, \ldots, \mathcal{P}_n$)
5. $\boxed{\text{Compute } \mathcal{C}_{\text{forb}} = \mathcal{C}^1_{\text{forb}} \cup \mathcal{C}^2_{\text{forb}}.}$ → The heart of this algorithm!
6. **return** $\mathcal{C}_{\text{forb}}$

Running Time

Lemma 13.13

The free configuration space C_{free} of a convex robot of constant complexity translating among a set of polygons with n edges in total can be computed in $O(n log^2 n)$ time.

Lemma 13.13

Proof

- $m_i \implies$ complexity of obstacle P_i
- triangulating all the obstacles takes time:

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

Lemma 13.13

Proof

- $m_i \implies$ complexity of obstacle P_i
- triangulating all the obstacles takes time:

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

- Computing the C-obstacles of each of the resulting triangles takes linear time in total

Lemma 13.13

Proof

- $m_i \implies$ complexity of obstacle P_i
- triangulating all the obstacles takes time:

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

- Computing the C-obstacles of each of the resulting triangles takes linear time in total
- the merge step (line 5) can be done in:

$$O((n_1 + n_2 + k) log(n_1 + n_2))$$

$n_1 \Rightarrow$ the complexity of C^1_{forb}
$n_2 \Rightarrow$ the complexity of C^2_{forb}
$k \Rightarrow$ the complexity of $C^1_{forb} \cup C^2_{forb}$

Lemma 13.13

Proof

- $m_i \implies$ complexity of obstacle P_i
- triangulating all the obstacles takes time:

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

- Computing the C-obstacles of each of the resulting triangles takes linear time in total
- the merge step (line 5) can be done in:

$$O((n_1 + n_2 + k) log(n_1 + n_2))$$

$n_1 \Rightarrow$ the complexity of C^1_{forb}
$n_2 \Rightarrow$ the complexity of C^2_{forb}
$k \Rightarrow$ the complexity of $C^1_{forb} \cup C^2_{forb}$

- n_1, n_2 and k are all $O(n)$, so the time for the merge step is $O(n log n)$

Lemma 13.13

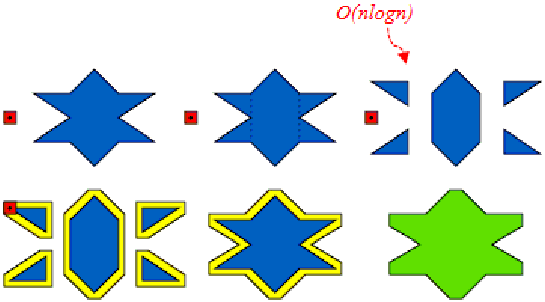
Time the algorithm

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(nlogn)$$

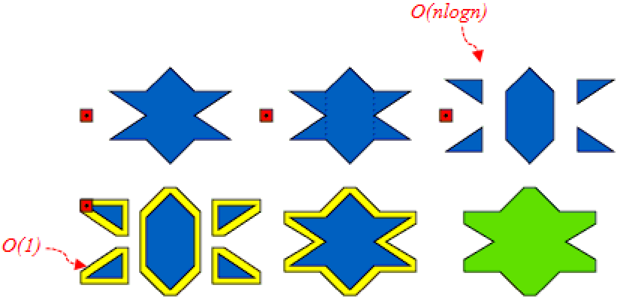
$$\Downarrow$$

$$O(nlog^2 n)$$

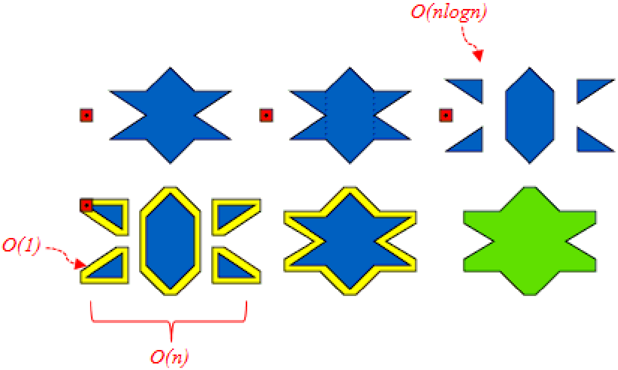
Time the algorithm



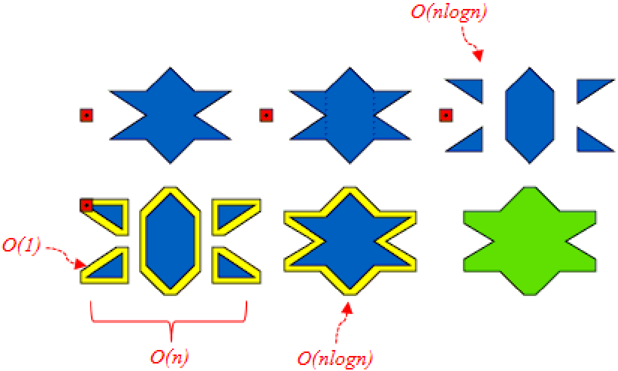
Time the algorithm



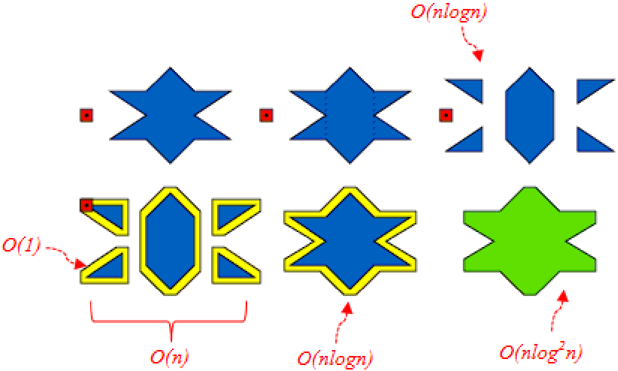
Time the algorithm



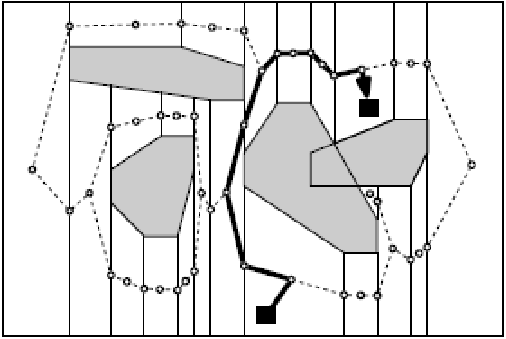
Time the algorithm



Time the algorithm



Continuance is same way as point Robot



Running Time

Lemma 13.13

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 $O(nlog^2n)$ expected time, such that between any start and goal position a collision-free path for R, if it exists, can be computed in $O(n)$ time.

END


Keep working hard
but also
make sure you enjoy your work
Mark de Berg