Robot Motion Planning

Outline:

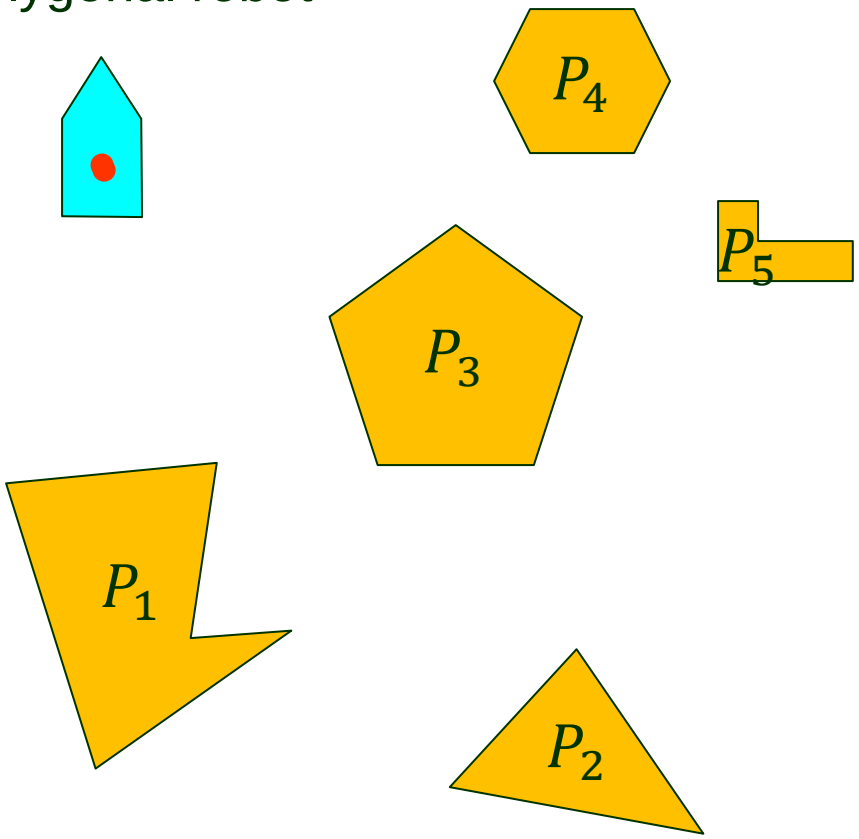
I. Configuration and degrees of freedom

II. Configuration space (C-space)

III. Planning of a collision-free path

I. The Planning Problem

Polygonal robot

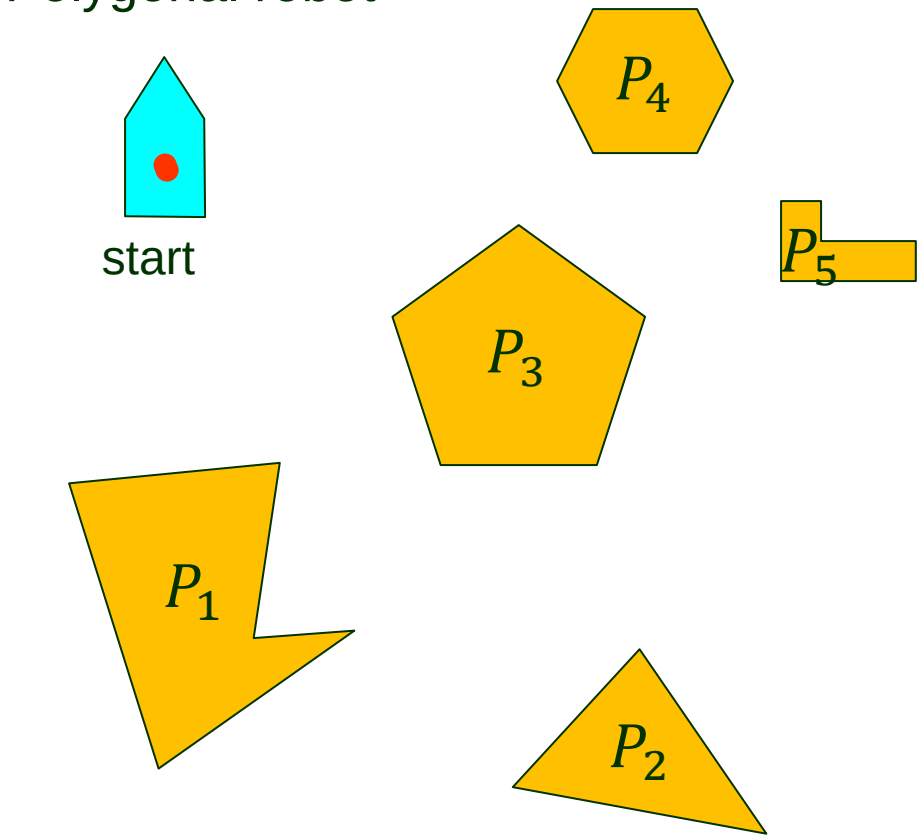


Static environment:

- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

I. The Planning Problem

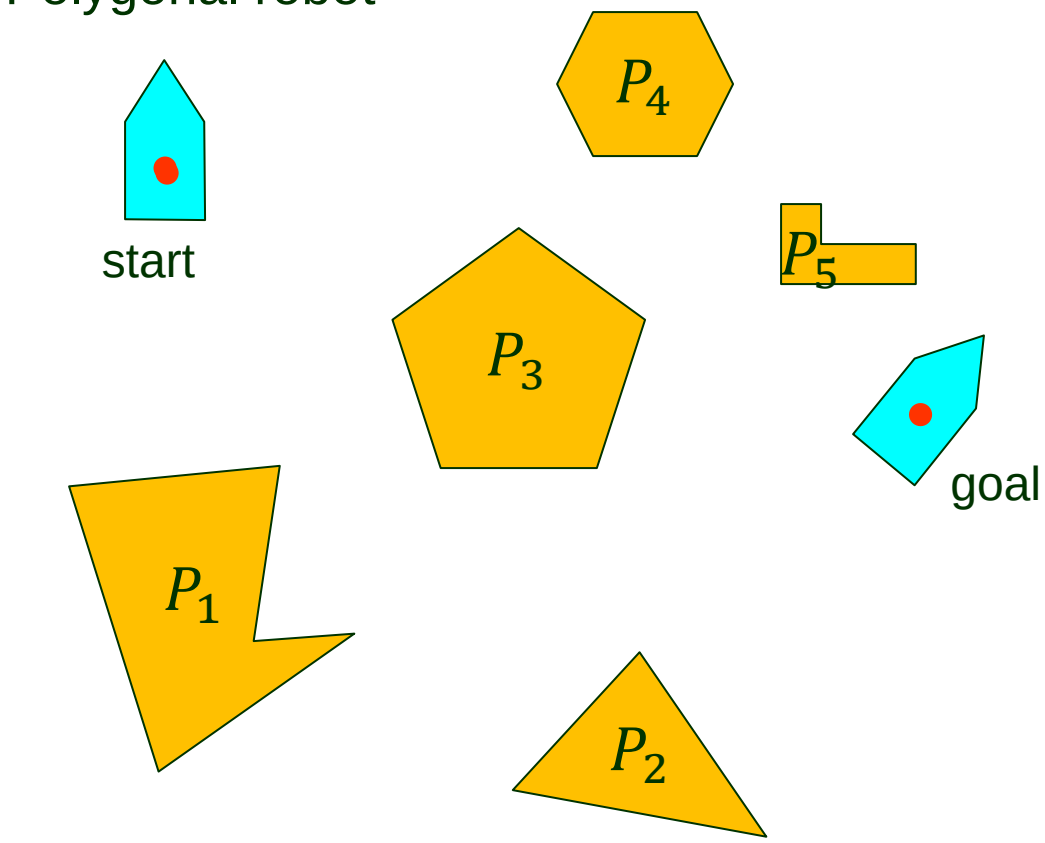


Static environment:

- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

I. The Planning Problem

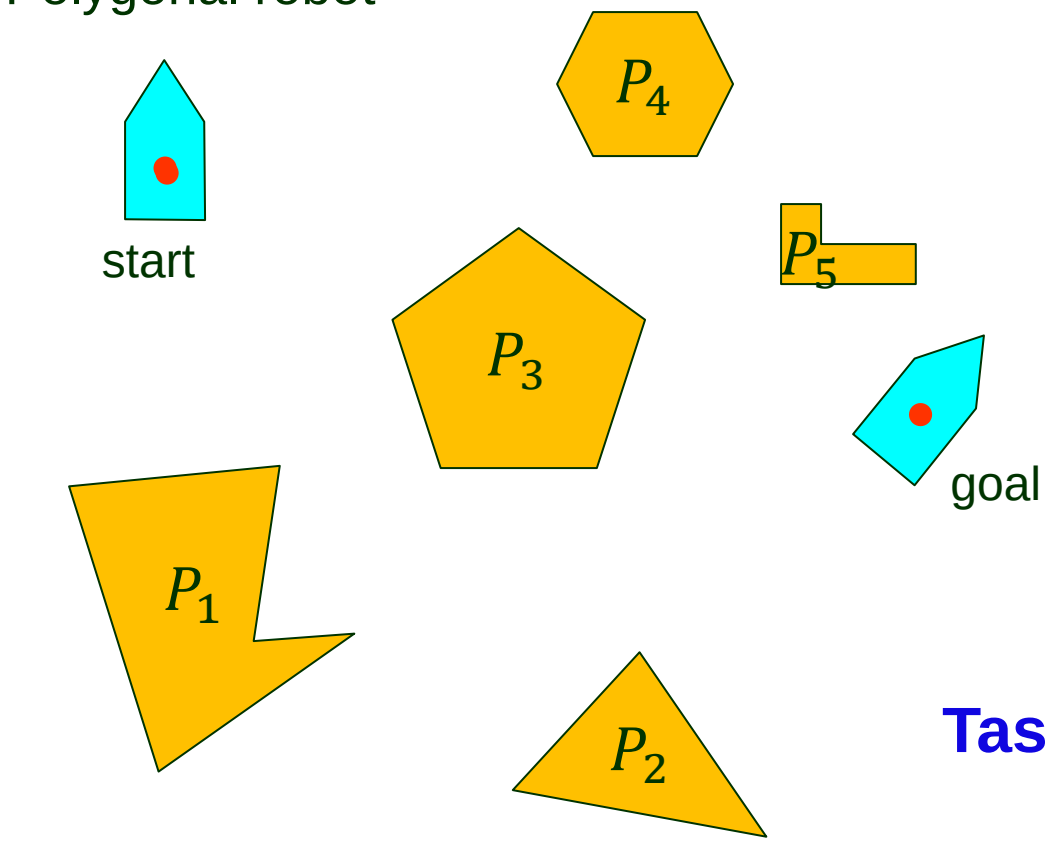


Static environment:

- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

I. The Planning Problem



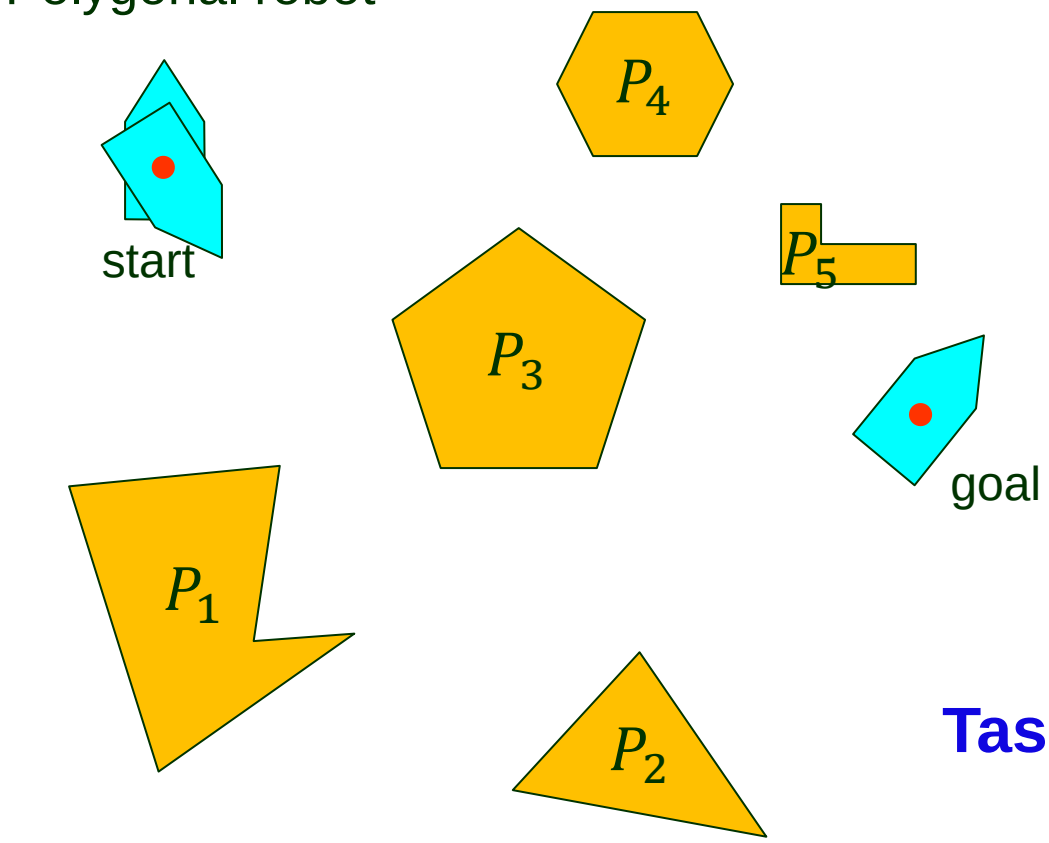
Static environment:

- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

Task: Move the robot from the start placement to the goal placement *without colliding* with any obstacles.

I. The Planning Problem



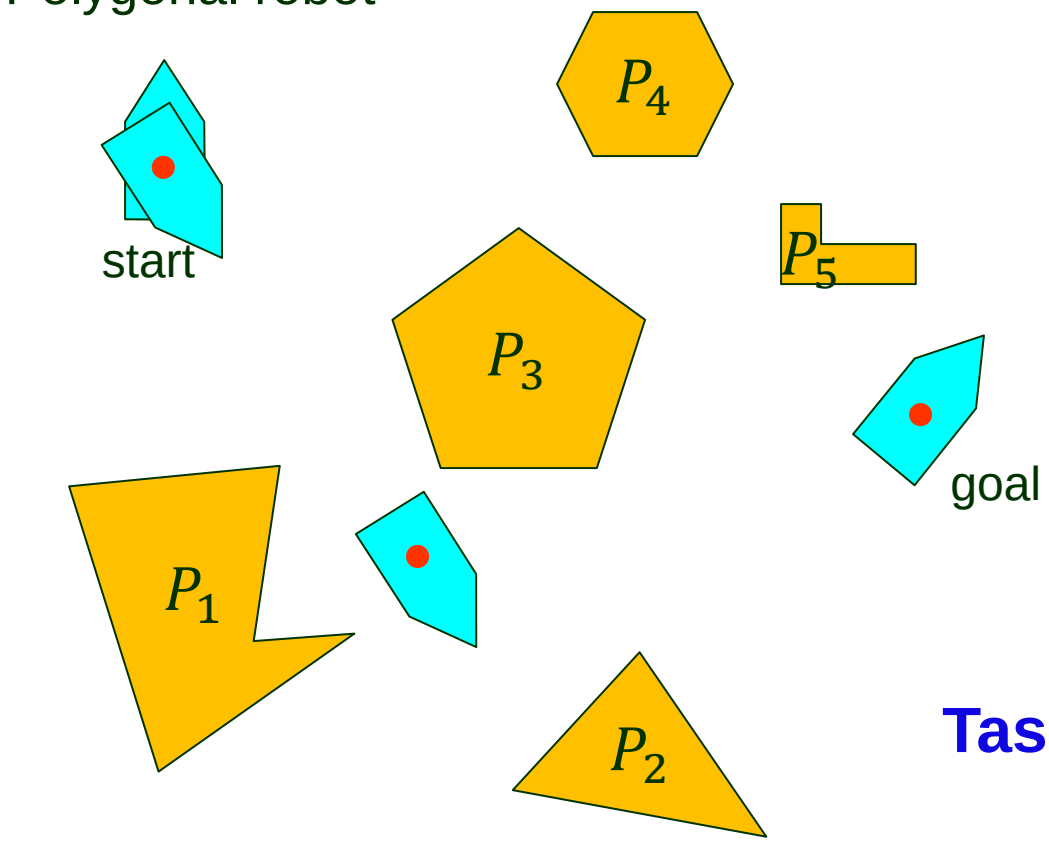
Static environment:

- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

Task: Move the robot from the start placement to the goal placement *without colliding* with any obstacles.

I. The Planning Problem



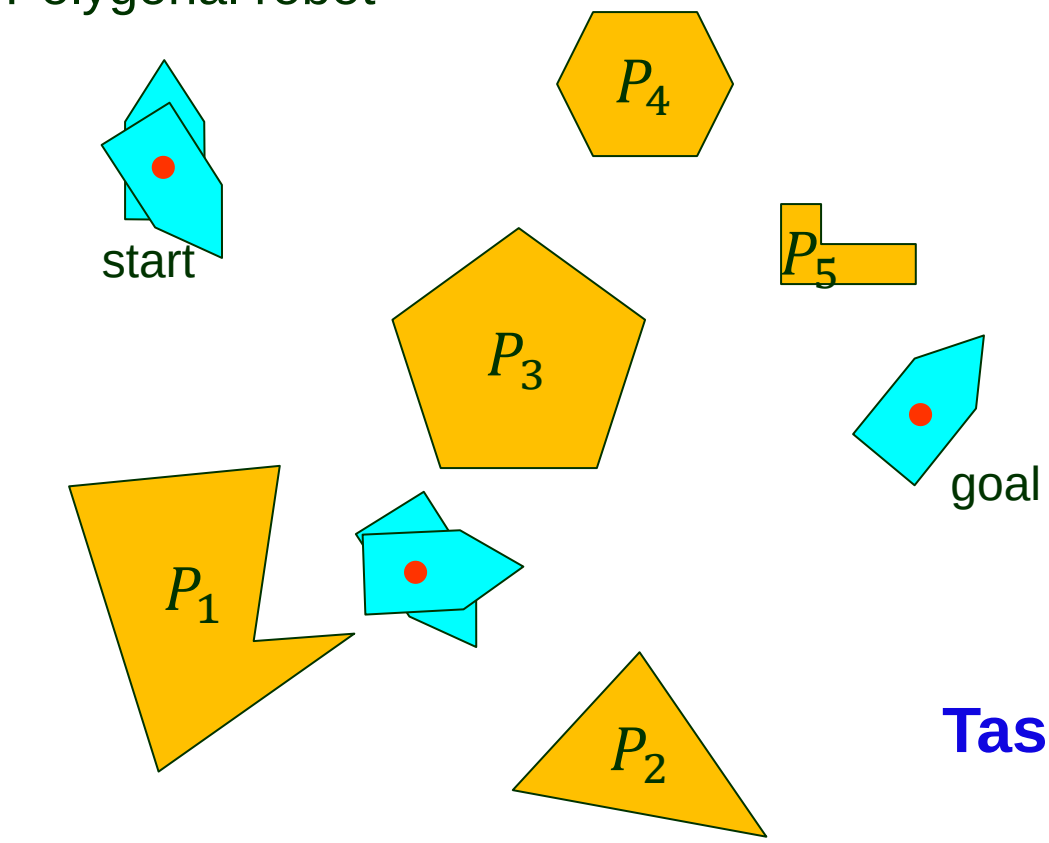
Static environment:

- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

Task: Move the robot from the start placement to the goal placement *without colliding* with any obstacles.

I. The Planning Problem



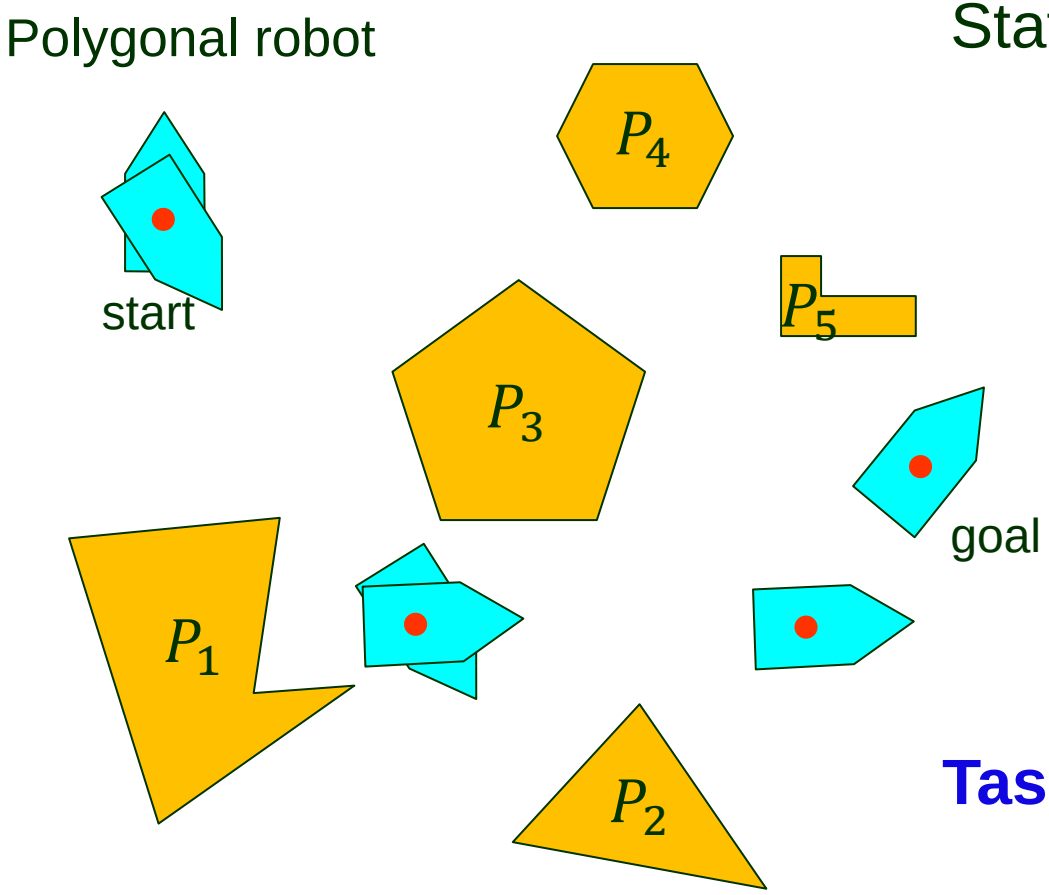
Static environment:

- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

Task: Move the robot from the start placement to the goal placement *without colliding* with any obstacles.

I. The Planning Problem



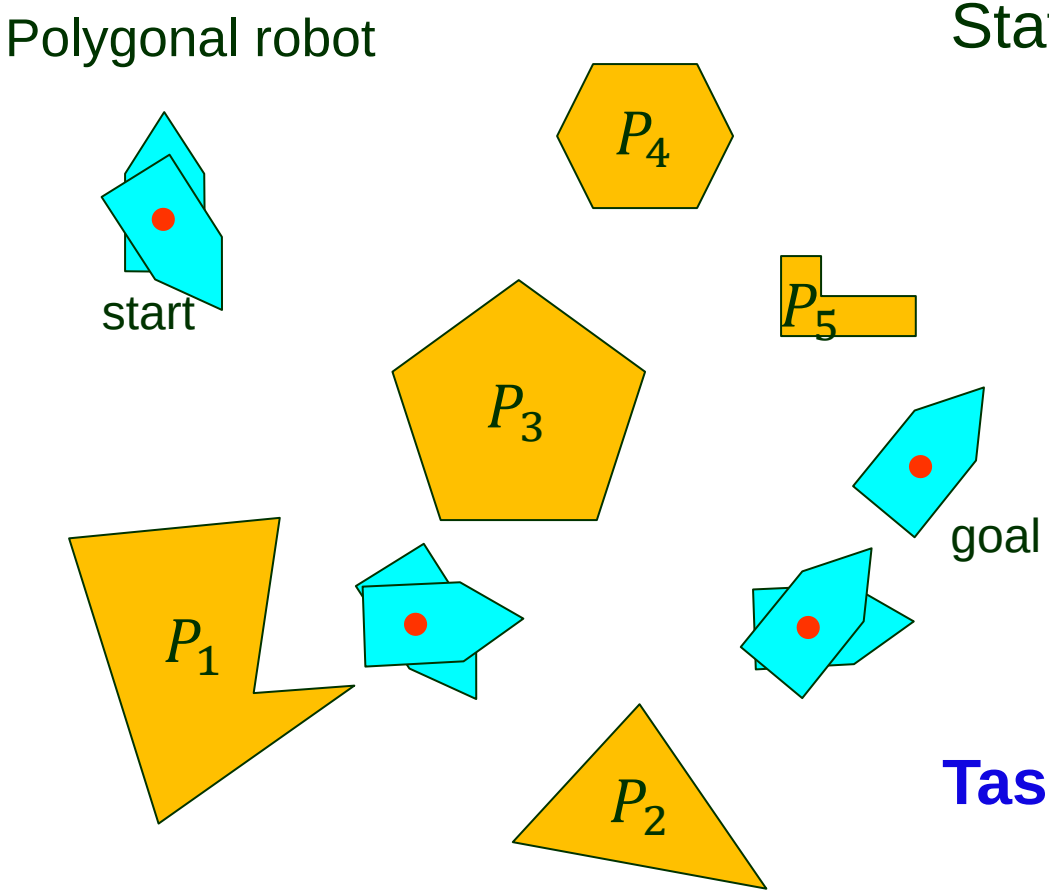
Static environment:

- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

Task: Move the robot from the start placement to the goal placement *without colliding* with any obstacles.

I. The Planning Problem



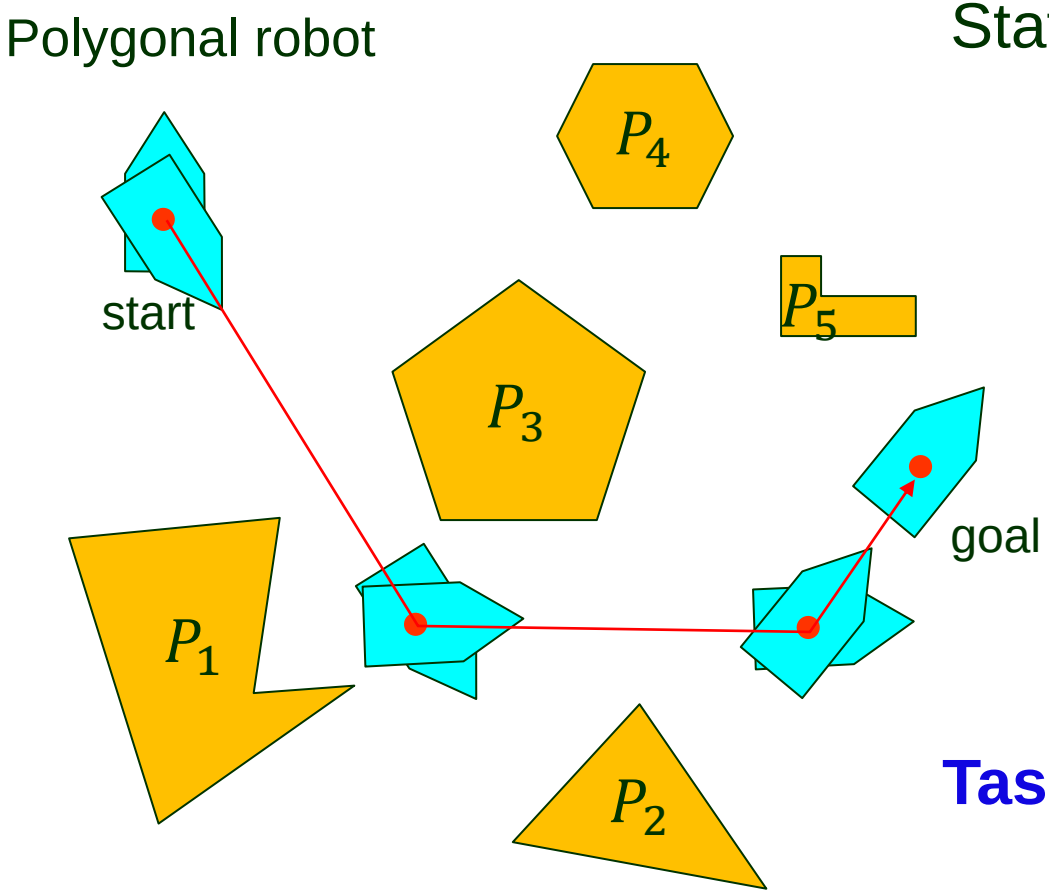
Static environment:

- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

Task: Move the robot from the start placement to the goal placement *without colliding* with any obstacles.

I. The Planning Problem



Static environment:

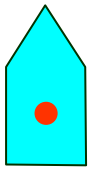
- 2-dimensional (2D)
- with polygonal obstacles

$$S = \{P_1, P_2, \ldots, P_t\}$$

Task: Move the robot from the start placement to the goal placement *without colliding* with any obstacles.

Reference Point

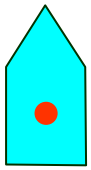
The robot as a polygon R.



Reference Point

The robot as a polygon R.

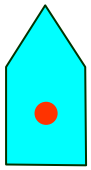
How to represent its placement, or *configuration*?



Reference Point

The robot as a polygon R.

How to represent its placement, or *configuration*?

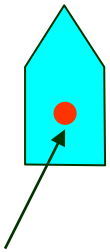


Pick a *reference point* (usually in the interior of R).

Reference Point

The robot as a polygon R.

How to represent its placement, or *configuration*?



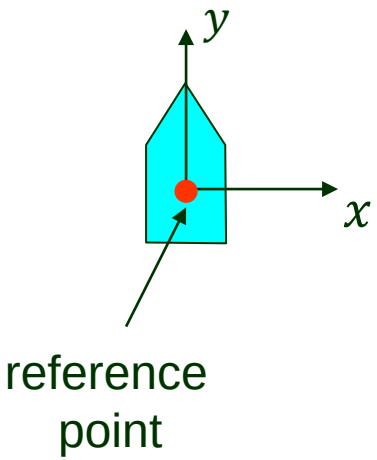
reference point

Pick a *reference point* (usually in the interior of R).

Reference Point

The robot as a polygon R.

How to represent its placement, or *configuration*?

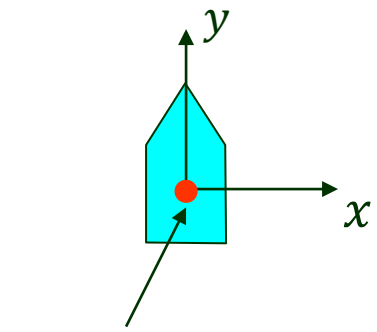


Pick a *reference point* (usually in the interior of R).

Reference Point

The robot as a polygon R.

How to represent its placement, or *configuration*?



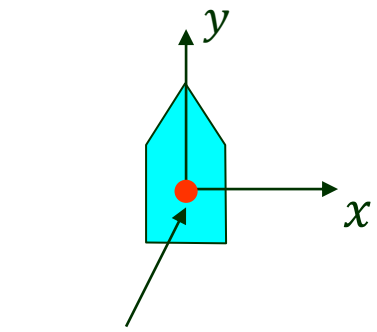
Pick a *reference point* (usually in the interior of R).

- No rotation

Reference Point

The robot as a polygon R.

How to represent its placement, or *configuration*?



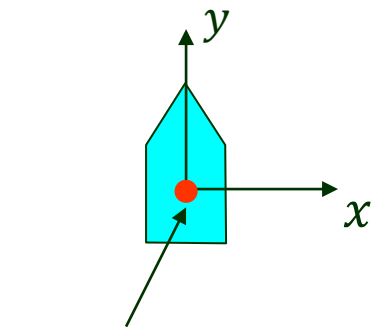
Pick a *reference point* (usually in the interior of R).

- No rotation
  - The robot can only translate.

Reference Point

The robot as a polygon R.

How to represent its placement, or *configuration*?



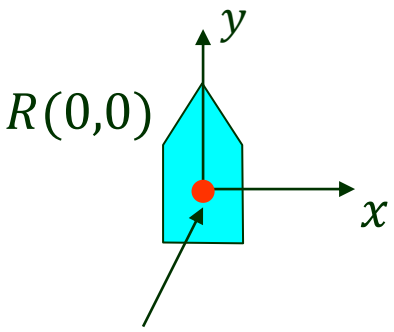
Pick a *reference point* (usually in the interior of R).

- No rotation
  - The robot can only translate.
  - $R(x, y)$: the robot placement with its reference point at (x, y).

Reference Point

The robot as a polygon R.

How to represent its placement, or *configuration*?



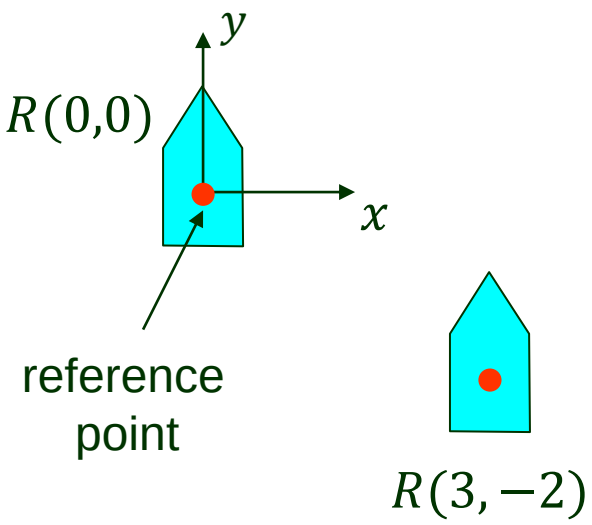
Pick a *reference point* (usually in the interior of R).

- No rotation
  - The robot can only translate.
  - $R(x, y)$: the robot placement with its reference point at (x, y).

Reference Point

The robot as a polygon R.

How to represent its placement, or *configuration*?



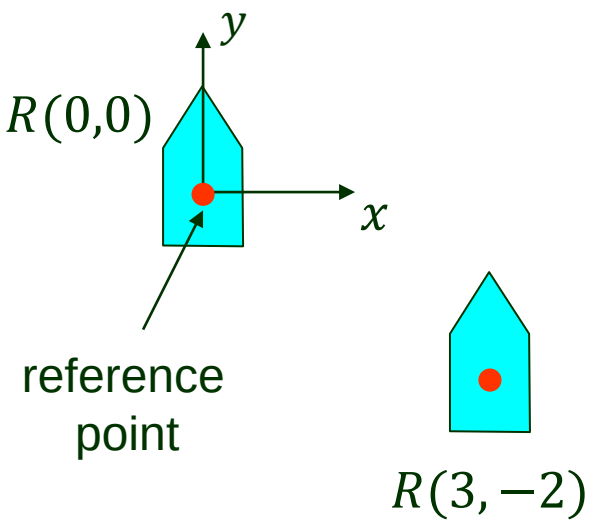
Pick a *reference point* (usually in the interior of R).

- No rotation
  - The robot can only translate.
  - $R(x, y)$: the robot placement with its reference point at (x, y).

Reference Point

The robot as a polygon R.

How to represent its placement, or *configuration*?

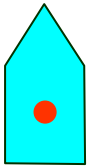


Pick a *reference point* (usually in the interior of R).

- No rotation
  - The robot can only translate.
  - $R(x, y)$: the robot placement with its reference point at (x, y).
- It has only two *degrees of freedom* (2 DOFs): x and y.

Configuration

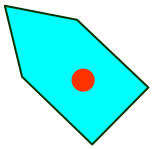
- With rotation
  - The robot can additionally change its orientation around the reference point.

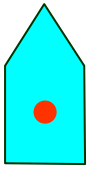


$R(0,0,0)$

Configuration

- With rotation
  - The robot can additionally change its orientation around the reference point.

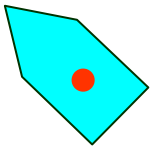


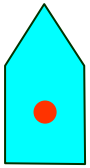


$R(0,0,0)$

Configuration

- With rotation
  - The robot can additionally change its orientation around the reference point.
  - Need to specify the rotation angle θ.

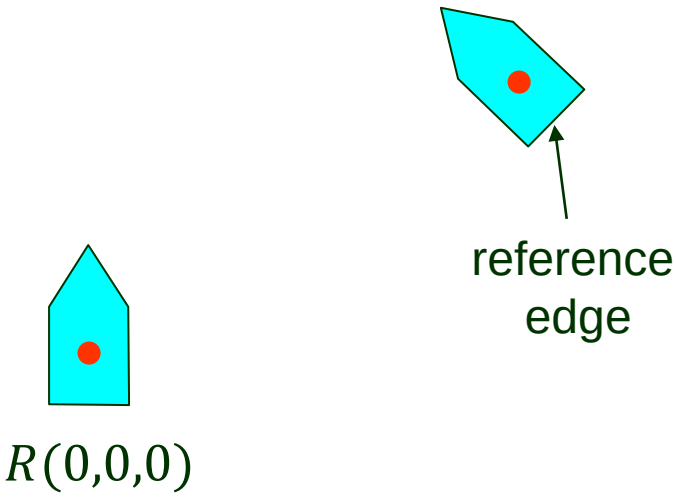




$R(0,0,0)$

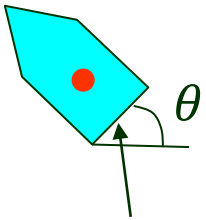
Configuration

- With rotation
  - The robot can additionally change its orientation around the reference point.
  - Need to specify the rotation angle θ.

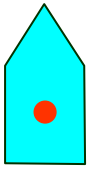


Configuration

- With rotation
  - The robot can additionally change its orientation around the reference point.
  - Need to specify the rotation angle θ.

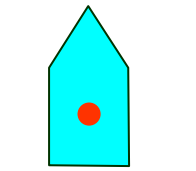


reference edge

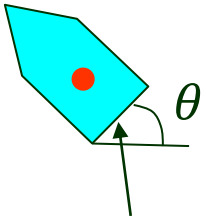


$R(0,0,0)$

Configuration



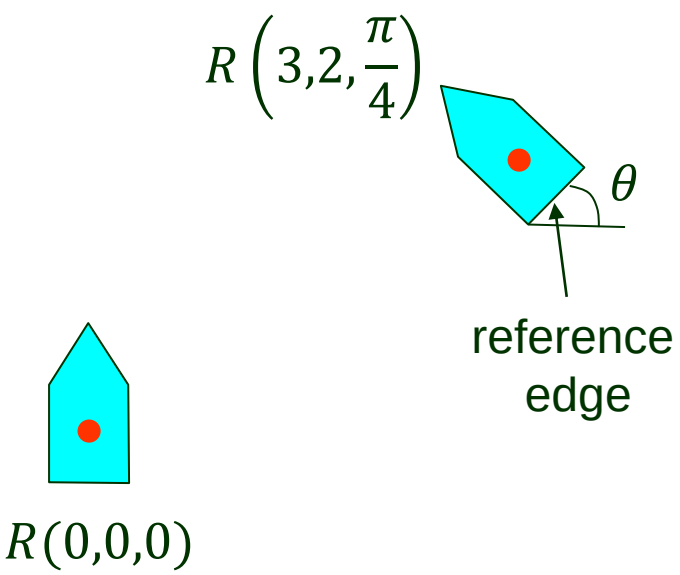
$R(0,0,0)$



reference edge

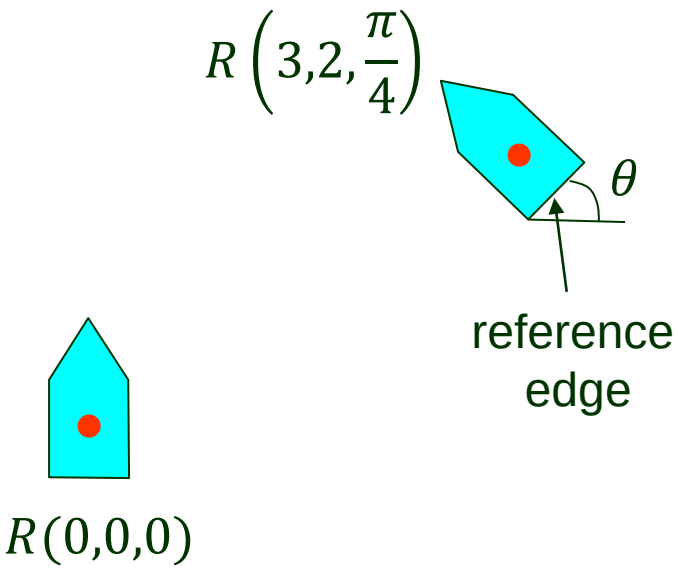
- With rotation
  - The robot can additionally change its orientation around the reference point.
  - Need to specify the rotation angle θ.
  - $R(x, y, \theta)$: the robot placed at (x, y) with orientation θ.

Configuration



- With rotation
  - The robot can additionally change its orientation around the reference point.
  - Need to specify the rotation angle θ.
  - $R(x, y, \theta)$: the robot placed at (x, y) with orientation θ.

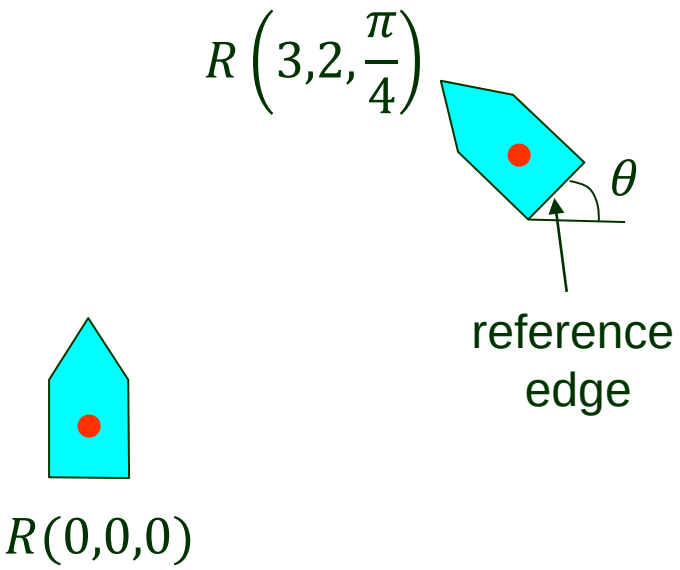
Configuration



- With rotation
  - The robot can additionally change its orientation around the reference point.
  - Need to specify the rotation angle θ.
  - $R(x, y, \theta)$: the robot placed at (x, y) with orientation θ.

3 DOFs: x, y, θ

Configuration



- With rotation
  - The robot can additionally change its orientation around the reference point.
  - Need to specify the rotation angle θ.
  - $R(x, y, \theta)$: the robot placed at (x, y) with orientation θ.

3 DOFs: x, y, θ

Configuration could also include the robot's velocity and angular velocity.

Pose refers to position and orientation only.

Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point 

Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point  2 DOFs: (x, y)

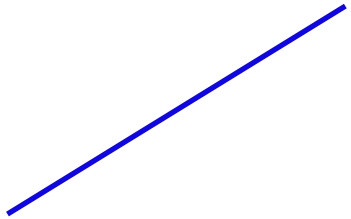
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point

2 DOFs: (x, y)

line



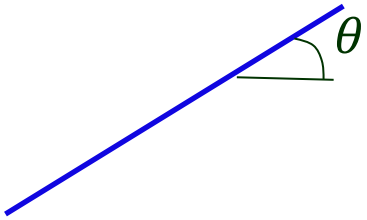
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point

2 DOFs: (x, y)

line



- θ: orientation

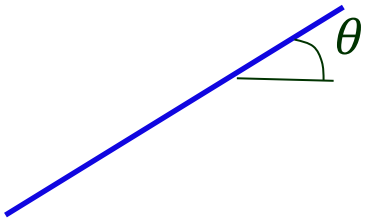
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point •

2 DOFs: (x, y)

line



- θ: orientation
- translation in the direction of the line does not change its configuration.

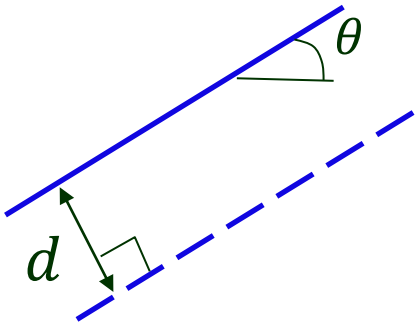
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point •

2 DOFs: (x, y)

line



- θ: orientation
- translation in the direction of the line does not change its configuration.

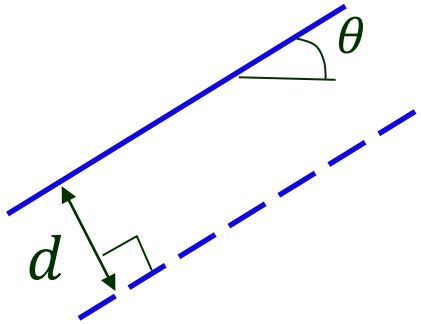
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point

2 DOFs: (x, y)

line



- θ: orientation
- translation in the direction of the line does not change its configuration.
- d: signed distance of translation in the perpendicular direction

Degrees of Freedom

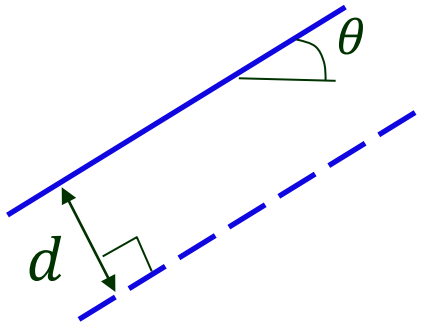
Number of *independent parameters* that define the configuration of an object.

point •

2 DOFs: (x, y)

line

2 DOFs: (θ, d)



- θ: orientation
- translation in the direction of the line does not change its configuration.
- d: signed distance of translation in the perpendicular direction

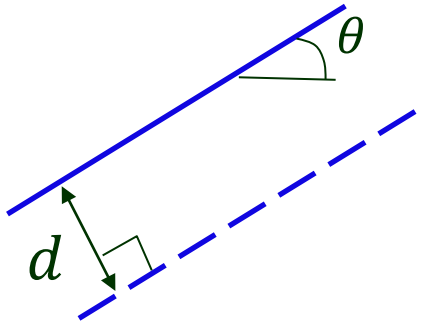
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point

2 DOFs: (x, y)

line



2 DOFs: (θ, d)

- θ: orientation
- translation in the direction of the line does not change its configuration.
- d: signed distance of translation in the perpendicular direction

line segment

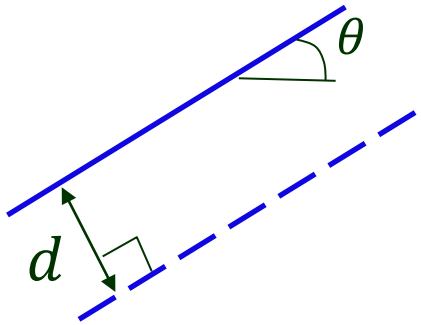
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point

2 DOFs: (x, y)

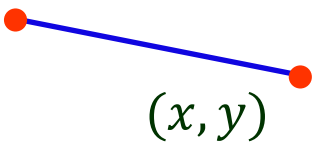
line



2 DOFs: (θ, d)

- θ: orientation
- translation in the direction of the line does not change its configuration.
- d: signed distance of translation in the perpendicular direction

line segment



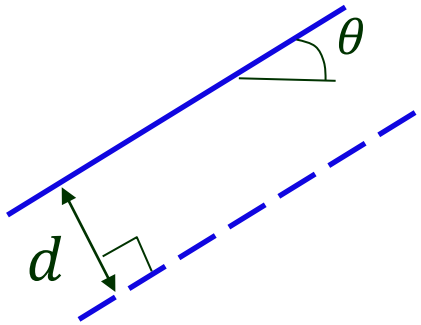
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point

2 DOFs: (x, y)

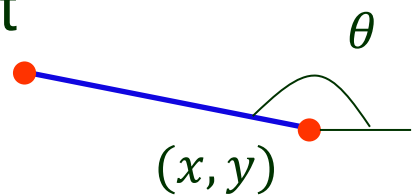
line



2 DOFs: (θ, d)

- θ: orientation
- translation in the direction of the line does not change its configuration.
- d: signed distance of translation in the perpendicular direction

line segment



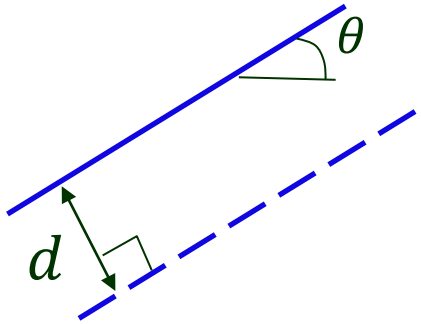
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point

2 DOFs: (x, y)

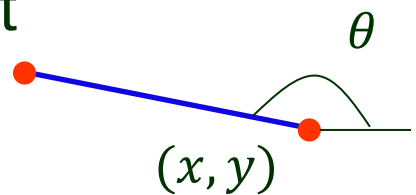
line



2 DOFs: (θ, d)

- θ: orientation
- translation in the direction of the line does not change its configuration.
- d: signed distance of translation in the perpendicular direction

line segment



- translation along the segment changes its configuration.

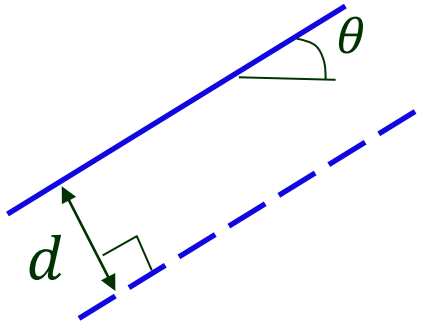
Degrees of Freedom

Number of *independent parameters* that define the configuration of an object.

point

2 DOFs: (x, y)

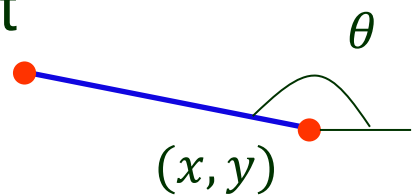
line



2 DOFs: (θ, d)

- θ: orientation
- translation in the direction of the line does not change its configuration.
- d: signed distance of translation in the perpendicular direction

line segment



3 DOFs: (x, y, θ)

- translation along the segment changes its configuration.

DOF (cont'd)

DOFs for a robot as one rigid body:

	2D	3D
translation	(x, y)	(x, y, z)
translation & rotation	(x, y, θ)	$(x, y, z, \theta, \phi, \psi)$

DOF (cont'd)

DOFs for a robot as one rigid body:

	2D	3D
translation	(x, y)	(x, y, z)
translation & rotation	(x, y, θ)	$(x, y, z, \theta, \phi, \psi)$

rotation angles about stationary x, y, z axes,

DOF (cont'd)

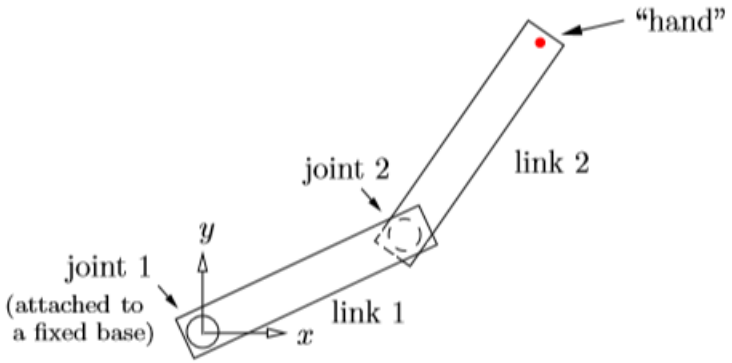
DOFs for a robot as one rigid body:

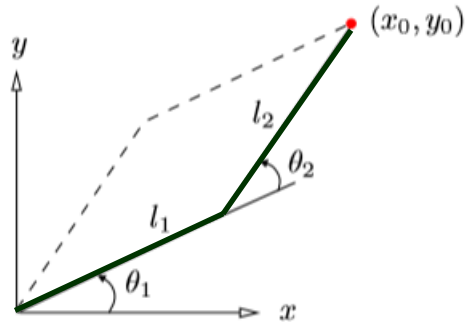
	2D	3D
translation	(x, y)	(x, y, z)
translation & rotation	(x, y, θ)	$(x, y, z, \theta, \phi, \psi)$

rotation angles about stationary x, y, z axes, or three Euler angles: roll, pitch, and yaw

Serial Manipulators

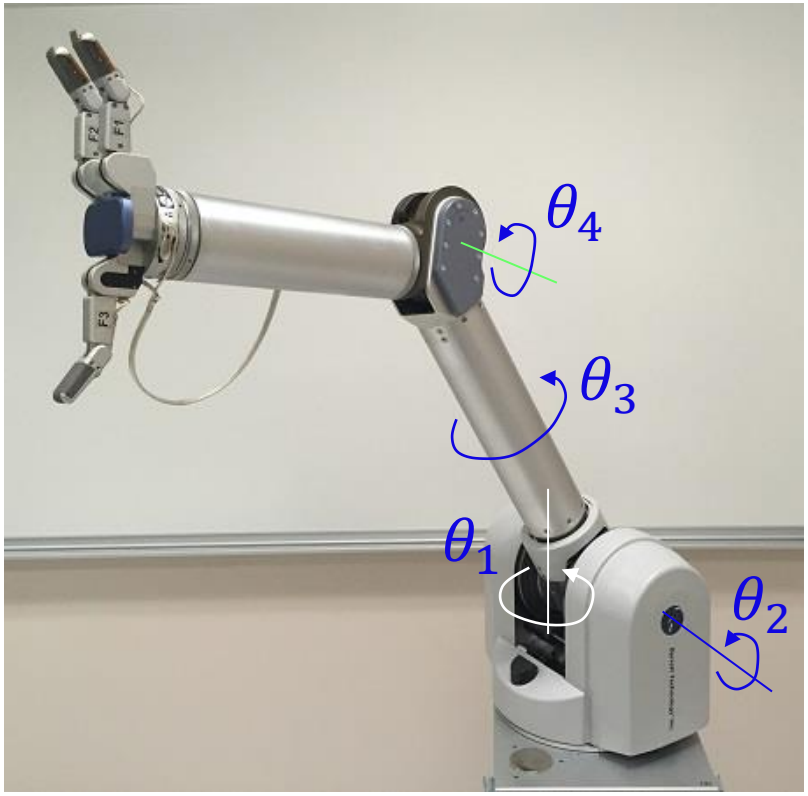
2-DOF manipulator: (θ_1, θ_2)





Serial Manipulators

4-DOF WAM Arm + 3-DOF BarretHand
(both from Barret Technology, Inc.)



Shadow Hand

- Anthropomorphic
- 24 joints.

20 DOFs almost like the human hand!

Image: Shadow Robot Company

II. Configuration Space (C-space)

Work space W: 2D environment

Configuration space $C(R)$: parameter space of the robot R in W

II. Configuration Space (C-space)

Work space W: 2D environment

Configuration space $C(R)$: parameter space of the robot R in W

translation + rotation

II. Configuration Space (C-space)

Work space W: 2D environment

Configuration space $C(R)$: parameter space of the robot R in W

translation + rotation ⇨ C-space: $\mathbb{R}^2 \times [0,2\pi)$

II. Configuration Space (C-space)

Work space W: 2D environment

Configuration space $C(R)$: parameter space of the robot R in W

translation + rotation $\Rightarrow$ C-space: $\mathbb{R}^2 \times [0,2\pi)$

From the C-space to the workspace:

$$C(R) \rightarrow W$$

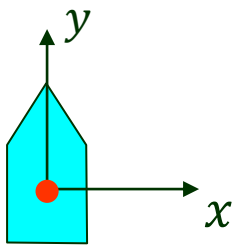
$$(x, y, \theta) \mapsto R(c)$$

placement

Point in the C-space

A polygonal robot is *represented as a point* in the C-space.

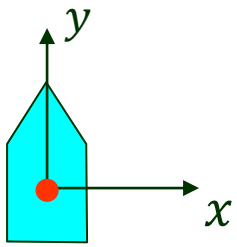
$(0, 0, 0)$ $\mapsto$



Point in the C-space

A polygonal robot is *represented as a point* in the C-space.

$(0, 0, 0)$ $\mapsto$

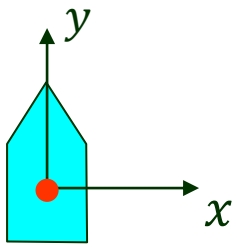


$\left(0, -2, -\frac{\pi}{4}\right)$ $\mapsto$

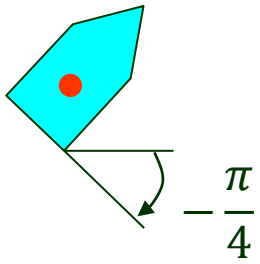
Point in the C-space

A polygonal robot is *represented as a point* in the C-space.

$(0, 0, 0)$ $\mapsto$

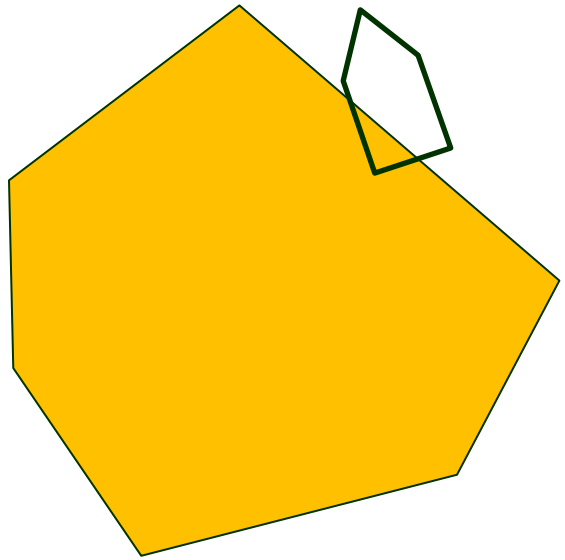


$\left(0, -2, -\frac{\pi}{4}\right)$ $\mapsto$



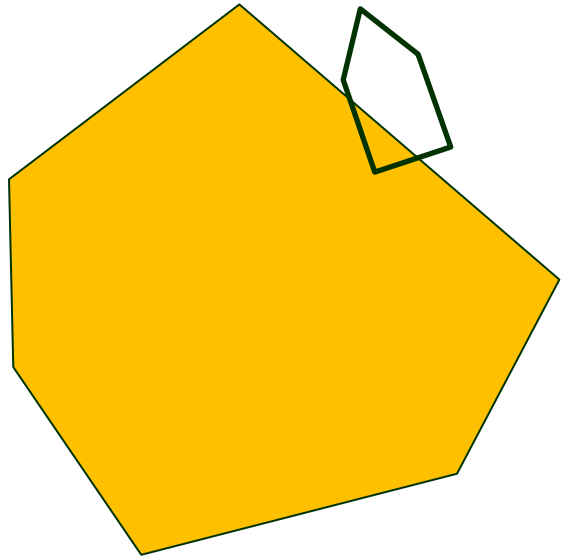
Collision and Forbidden Configuration

A configuration (x, y, θ) is *forbidden* if the placement $R(x, y, \theta)$ collides with an obstacle. Otherwise, it is free.



Collision and Forbidden Configuration

A configuration (x, y, θ) is *forbidden* if the placement $R(x, y, \theta)$ collides with an obstacle. Otherwise, it is free.



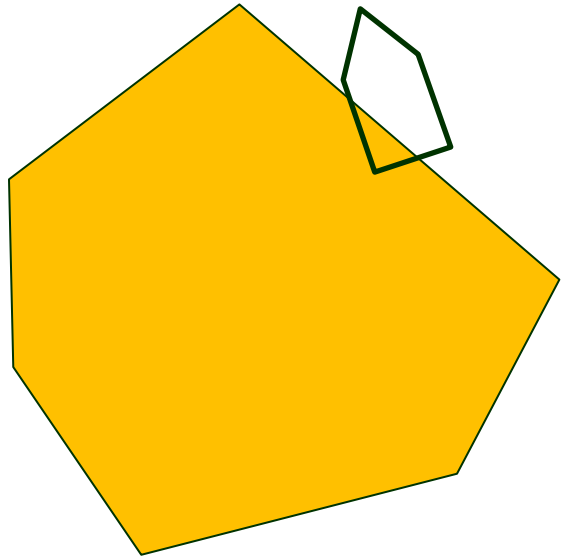
set of obstacles

↓

Forbidden C-space $C_{forb}(R, S)$:
the set of forbidden configurations

Collision and Forbidden Configuration

A configuration (x, y, θ) is *forbidden* if the placement $R(x, y, \theta)$ collides with an obstacle. Otherwise, it is free.



set of obstacles

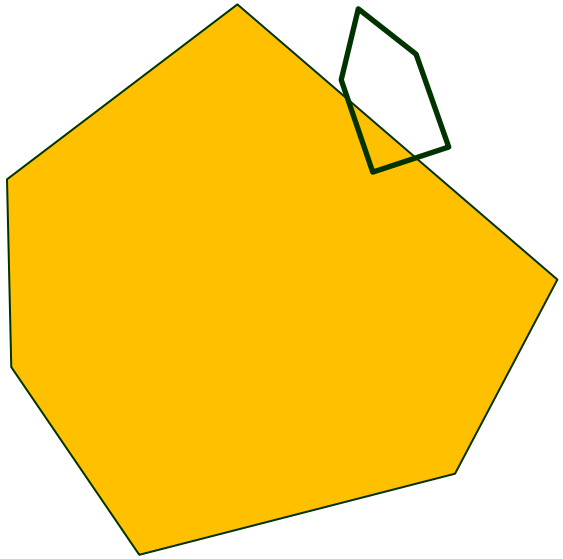
↓

Forbidden C-space $C_{forb}(R, S)$: the set of forbidden configurations

Free C-space $C_{forb}(R, S)$: the set of free configurations

Collision and Forbidden Configuration

A configuration (x, y, θ) is *forbidden* if the placement $R(x, y, \theta)$ collides with an obstacle. Otherwise, it is free.



set of obstacles ↓

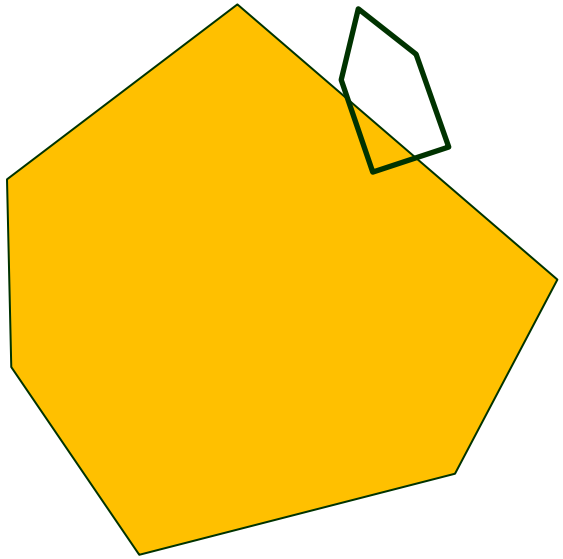
Forbidden C-space $C_{forb}(R, S)$: the set of forbidden configurations

Free C-space $C_{forb}(R, S)$: the set of free configurations

$$C_{forb}(R, S) \cup C_{free}(R, S) = C(R)$$

Collision and Forbidden Configuration

A configuration (x, y, θ) is *forbidden* if the placement $R(x, y, \theta)$ collides with an obstacle. Otherwise, it is free.



set of obstacles ↓

Forbidden C-space $C_{forb}(R, S)$: the set of forbidden configurations

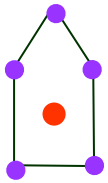
Free C-space $C_{forb}(R, S)$: the set of free configurations

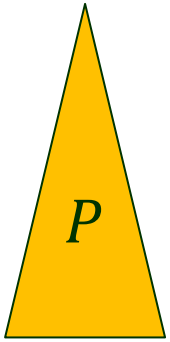
$$C_{forb}(R, S) \cup C_{free}(R, S) = C(R)$$

$$C_{forb}(R, S) \cap C_{free}(R, S) = \emptyset$$

C-obstacle for a Translating Robot

Robot Obstacle

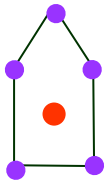


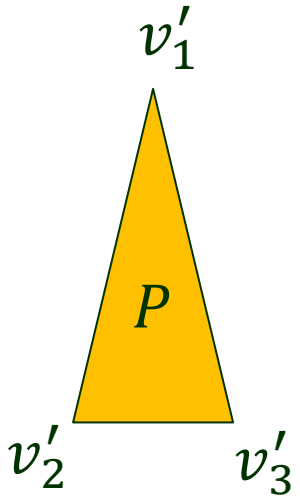


$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

Robot Obstacle

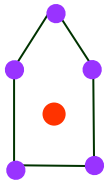




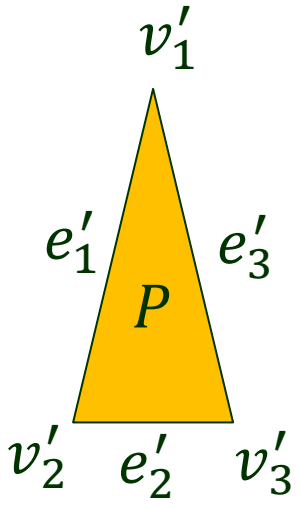
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

Robot



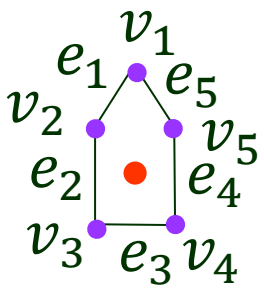
Obstacle



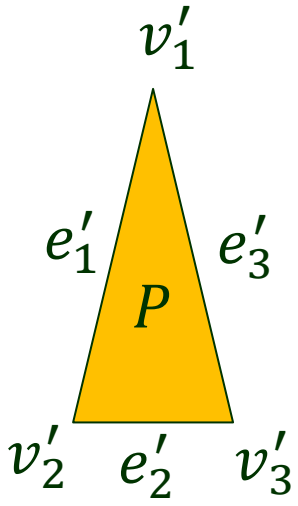
$P \mapsto$ *C-obstacle* $O(P)$: $\{\, c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

Robot



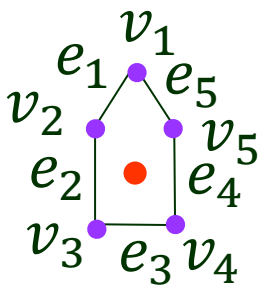
Obstacle



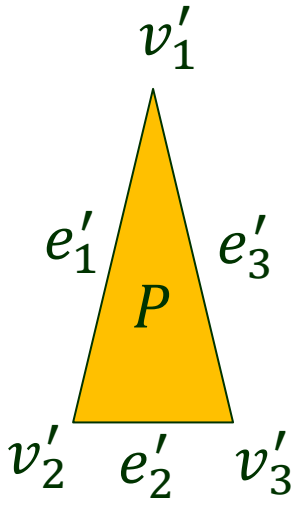
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

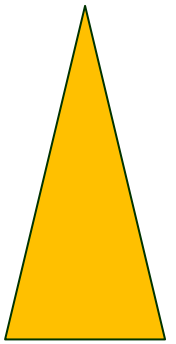
Robot



Obstacle



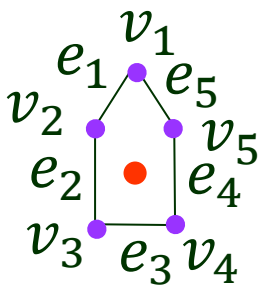
C-obstacle

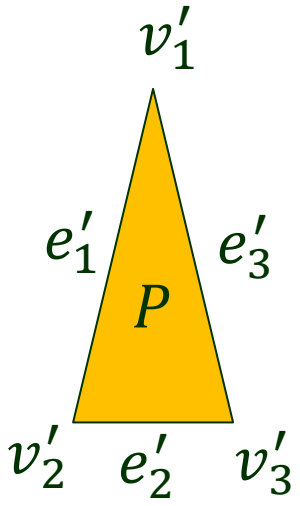


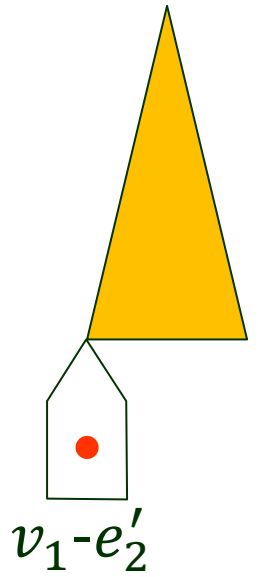
$P \mapsto$ *C-obstacle* $O(P)$: $\{\, c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}}\}$

C-obstacle for a Translating Robot

Robot Obstacle C-obstacle



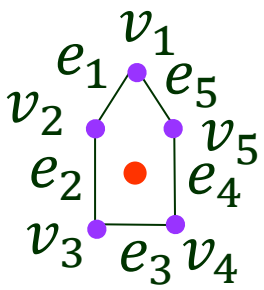


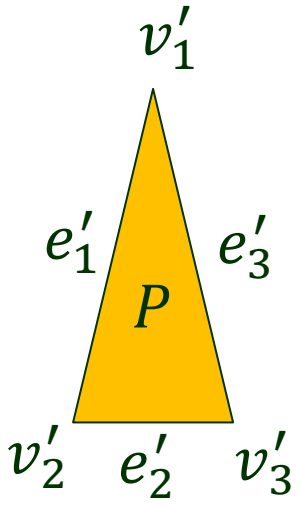


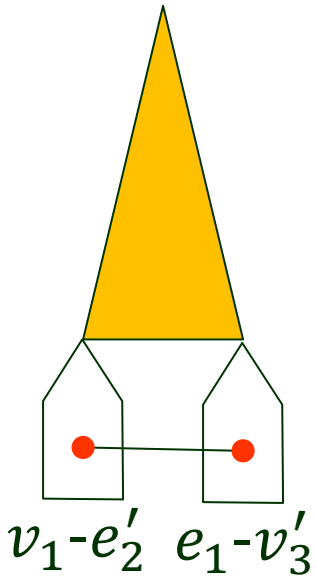
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

Robot Obstacle C-obstacle



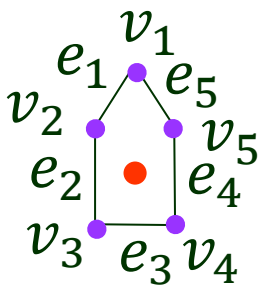




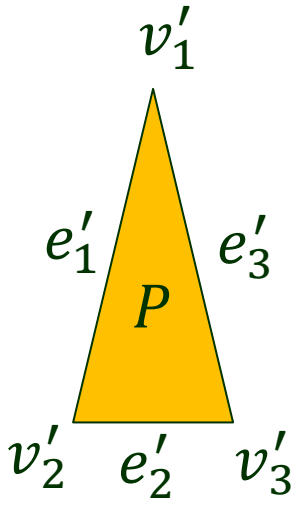
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

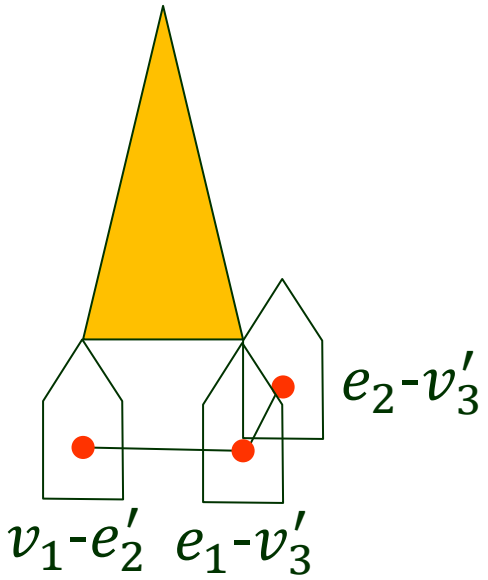
Robot



Obstacle



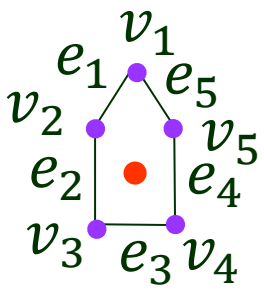
C-obstacle



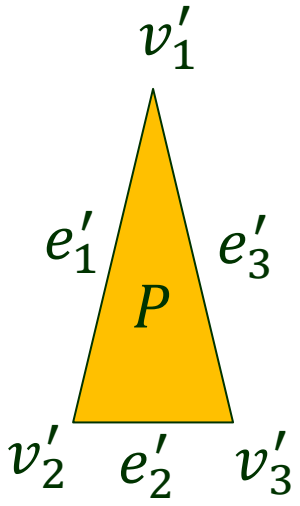
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

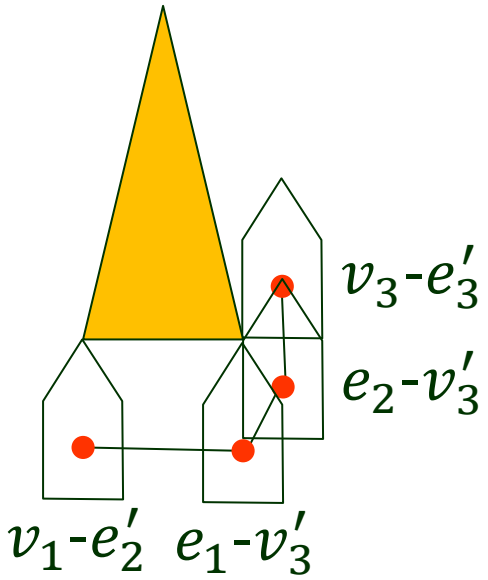
Robot



Obstacle



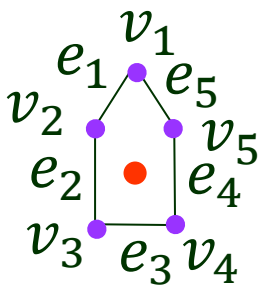
C-obstacle



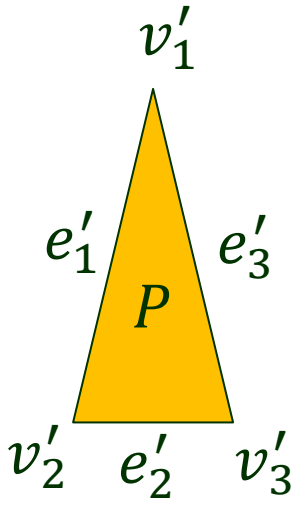
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

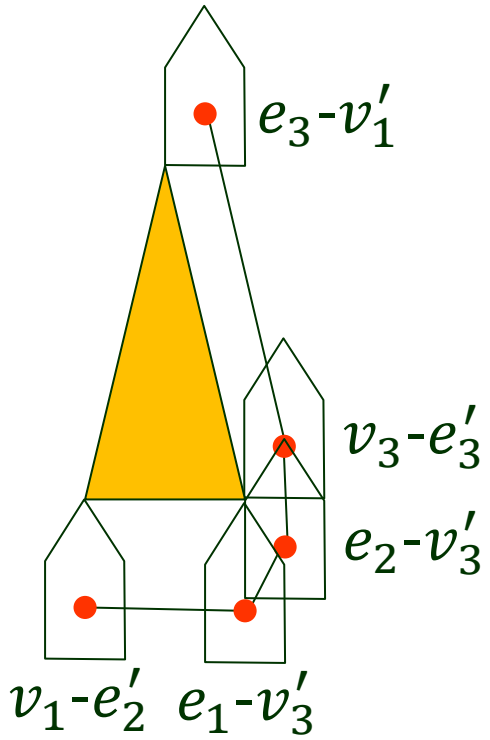
Robot



Obstacle



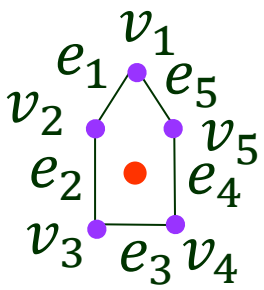
C-obstacle



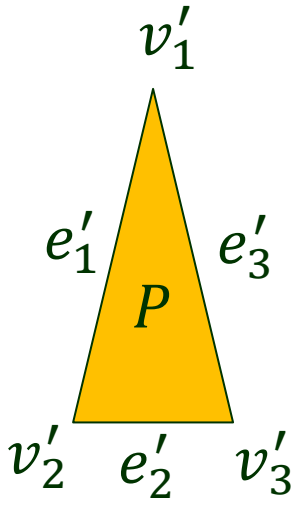
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

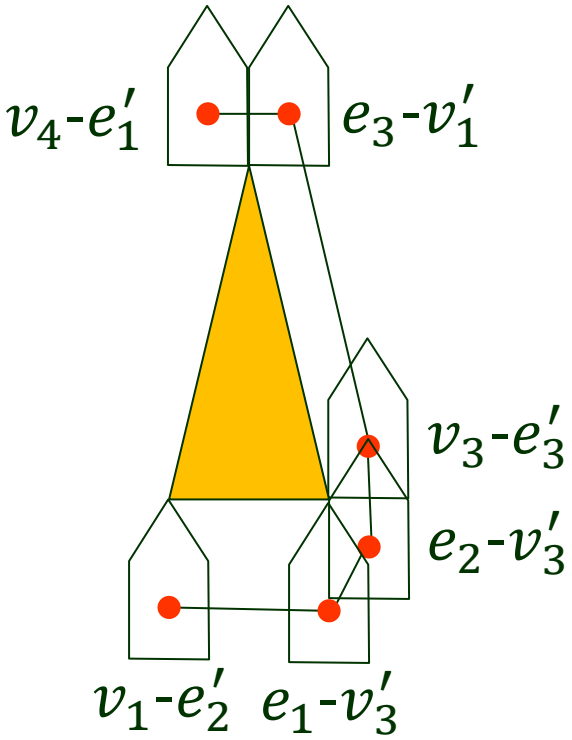
Robot



Obstacle



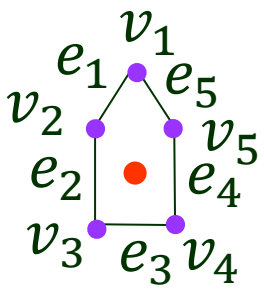
C-obstacle



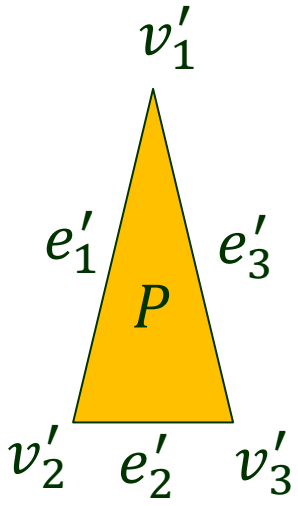
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

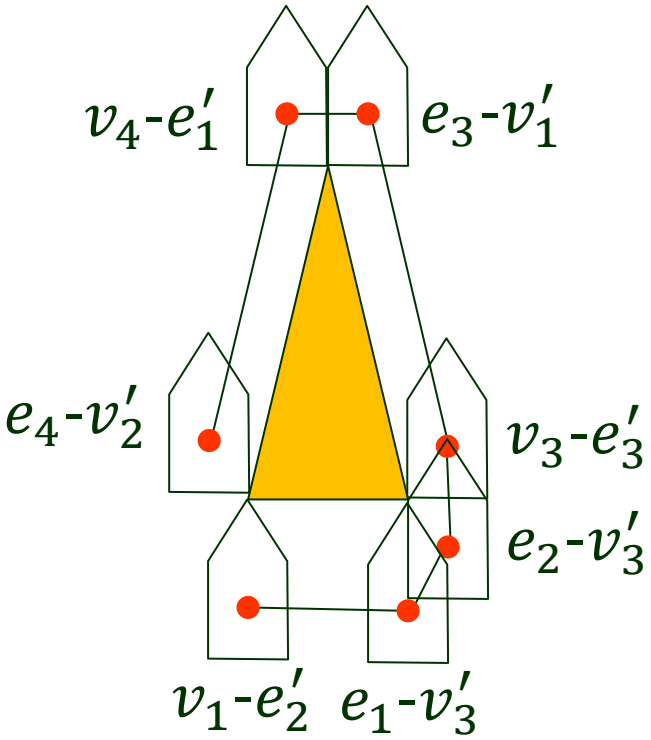
Robot



Obstacle



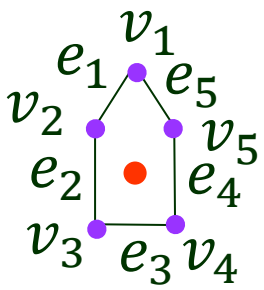
C-obstacle



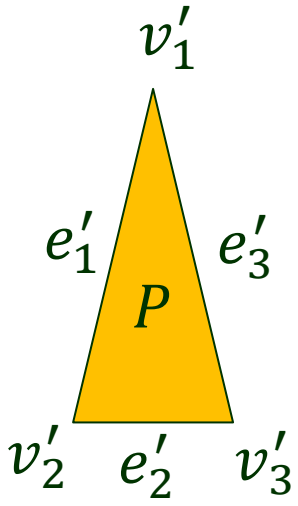
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

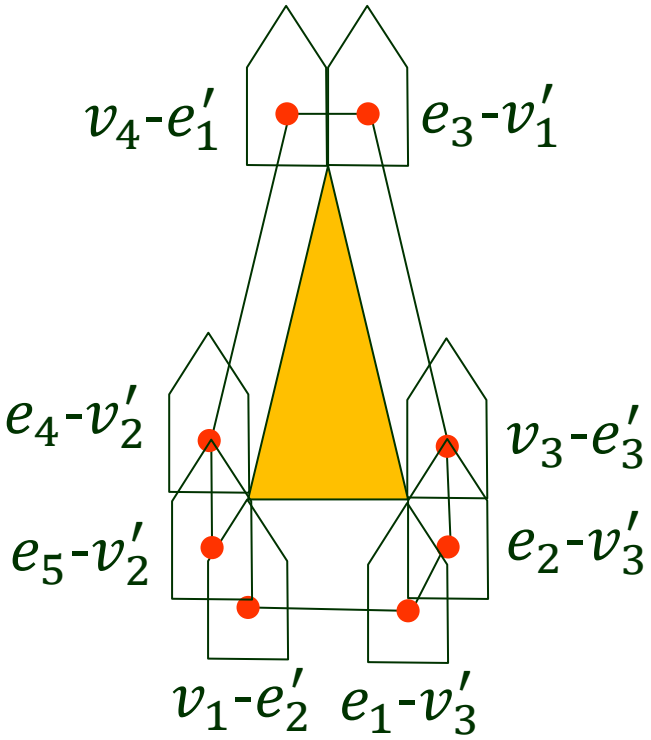
Robot



Obstacle



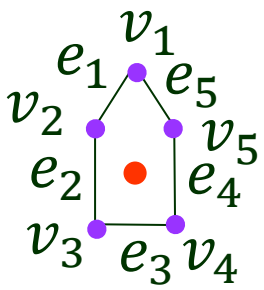
C-obstacle



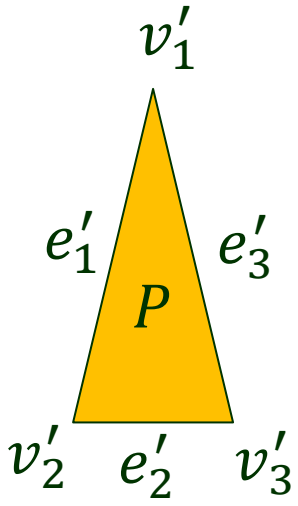
$P \mapsto$ *C-obstacle* $O(P)$: $\{\, c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

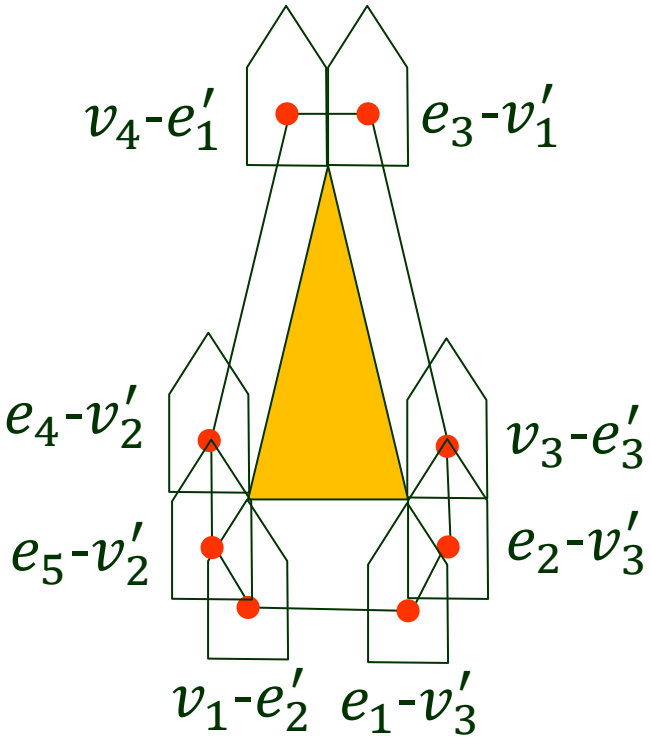
Robot



Obstacle



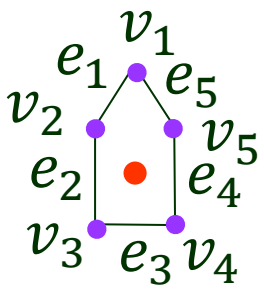
C-obstacle



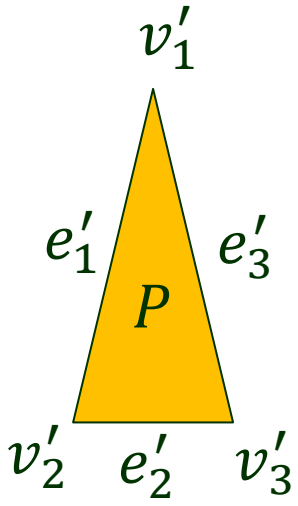
$P \mapsto$ *C-obstacle* $O(P)$: $\{\, c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

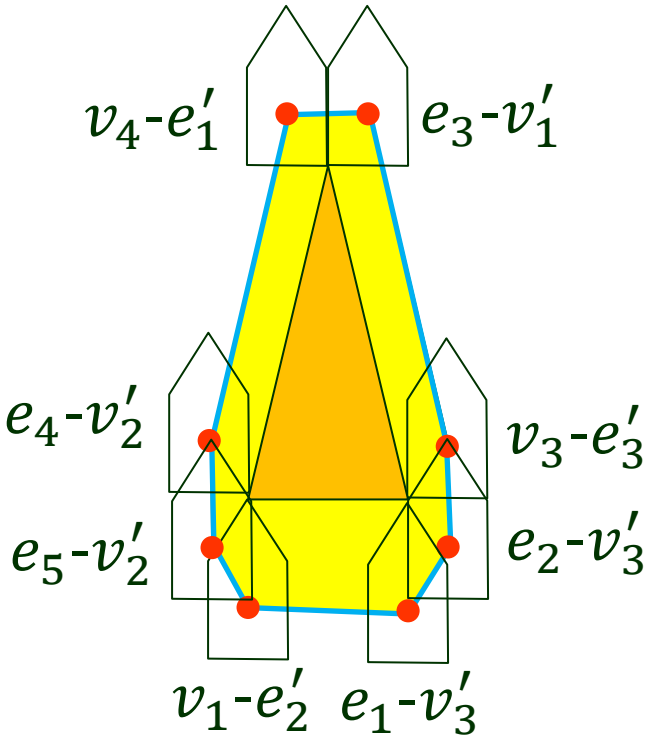
Robot



Obstacle



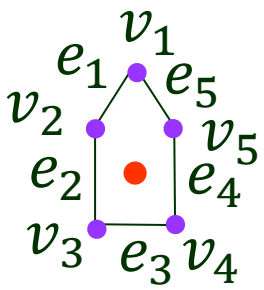
C-obstacle



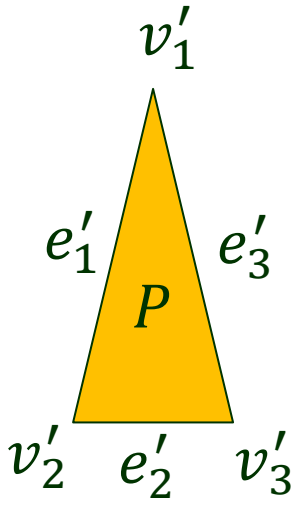
$P \mapsto$ *C-obstacle* $O(P)$: $\{ c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

C-obstacle for a Translating Robot

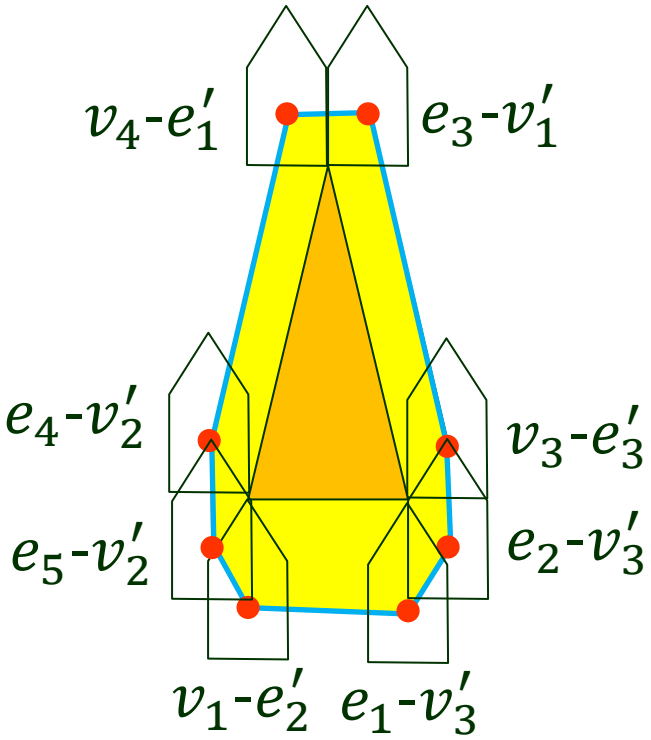
Robot



Obstacle



C-obstacle

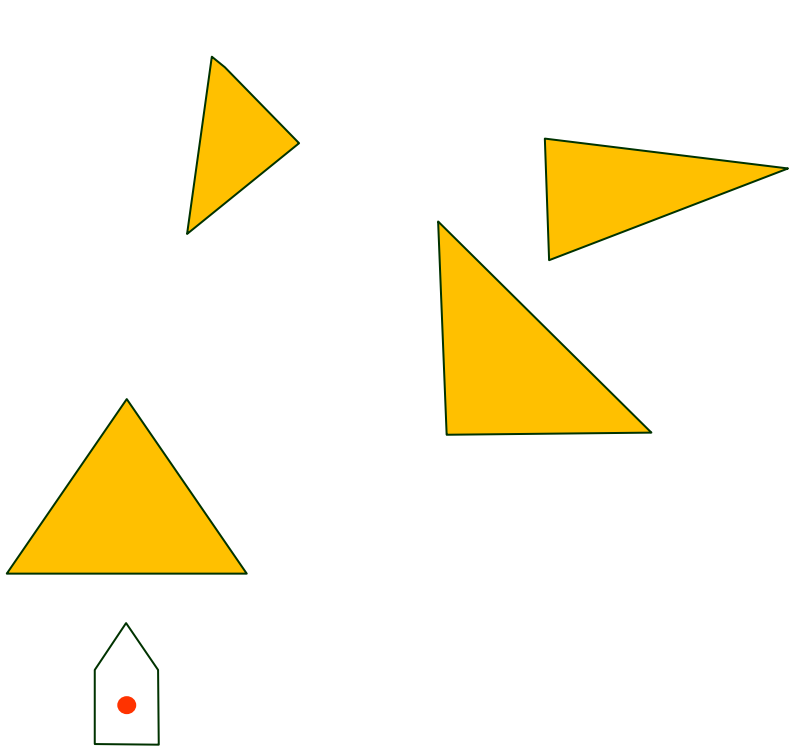


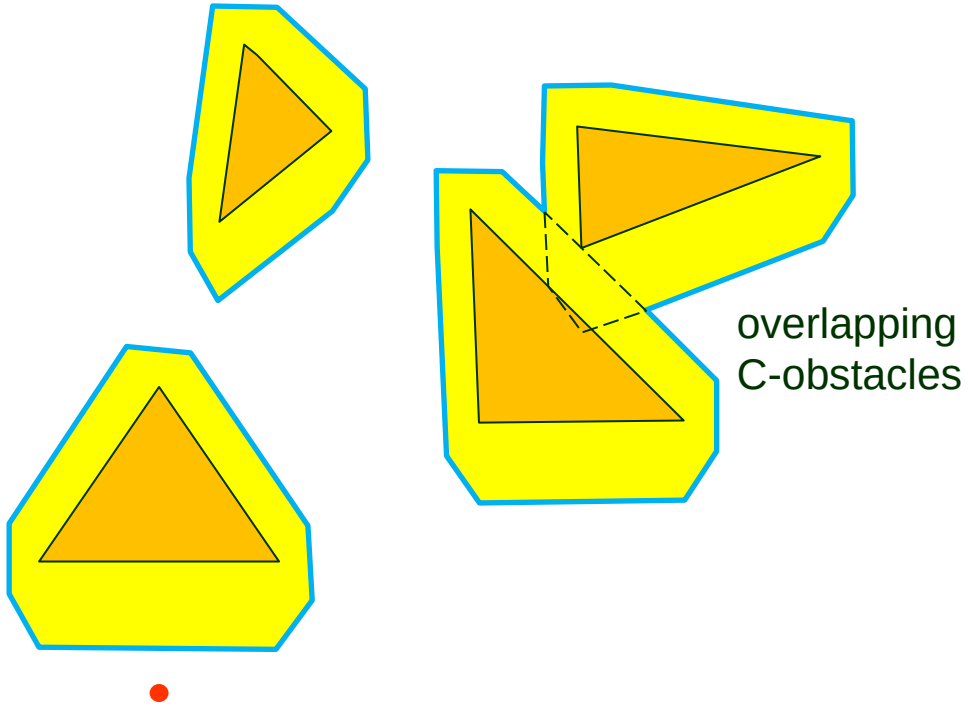
$P \mapsto$ *C-obstacle* $O(P)$: $\{\, c \mid c \in C(R)$ and $\underbrace{R(c) \cap P \neq \emptyset}_{\text{collision}} \}$

$$C_{forb}(R,S) = \bigcup_{P \in S} O(P)$$

Path

Work space:		C-space:
robot	↦	point
obstacle	↦	C-obstacles (possibly joined)
path	↦	curve





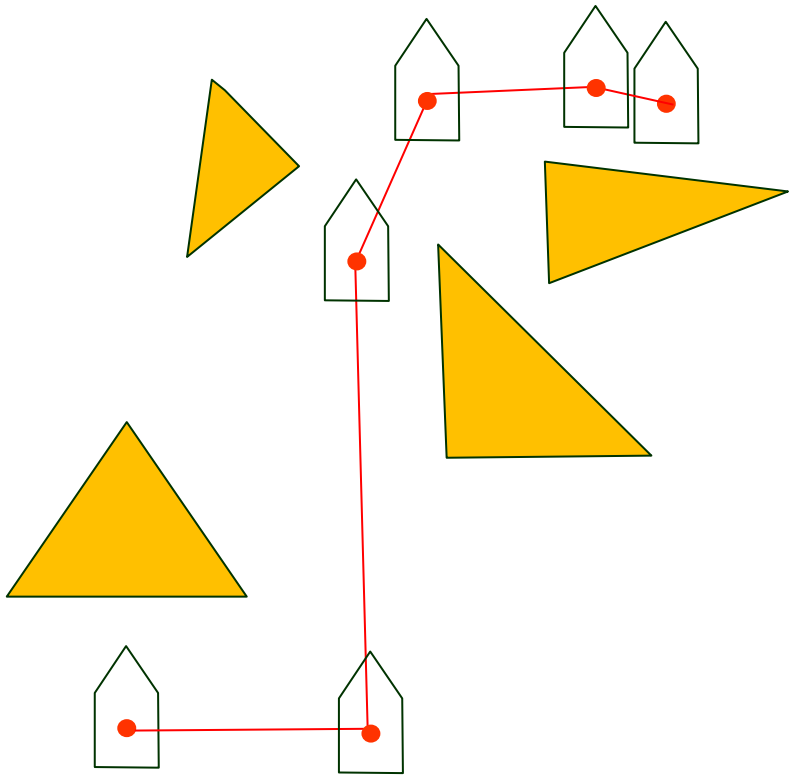
Path

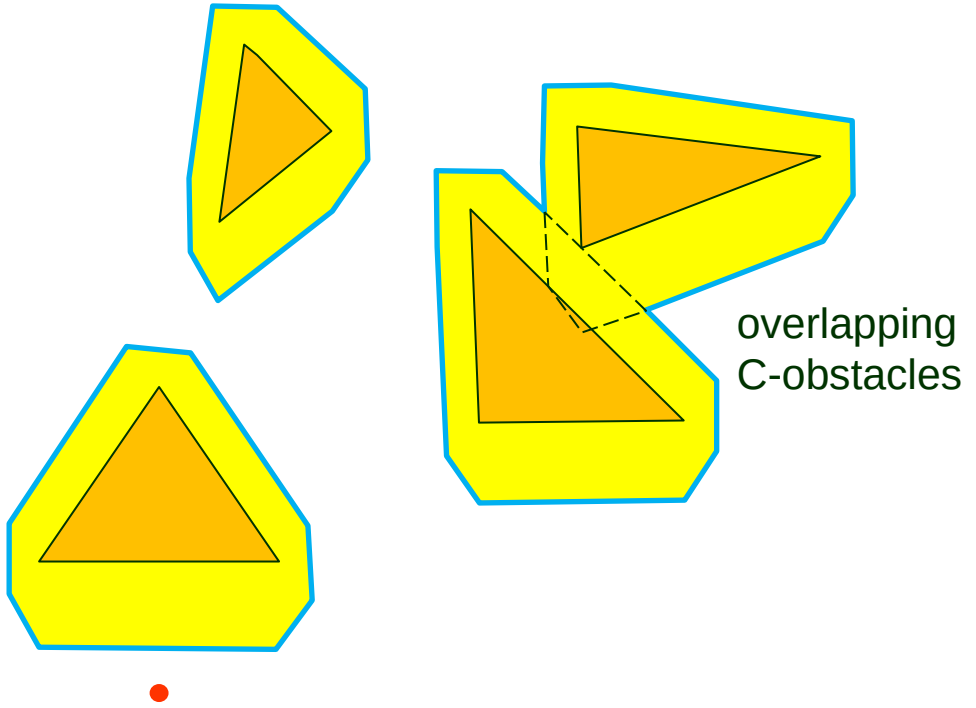
Work space: C-space:

robot ↦ point

obstacle ↦ C-obstacles (possibly joined)

path ↦ curve





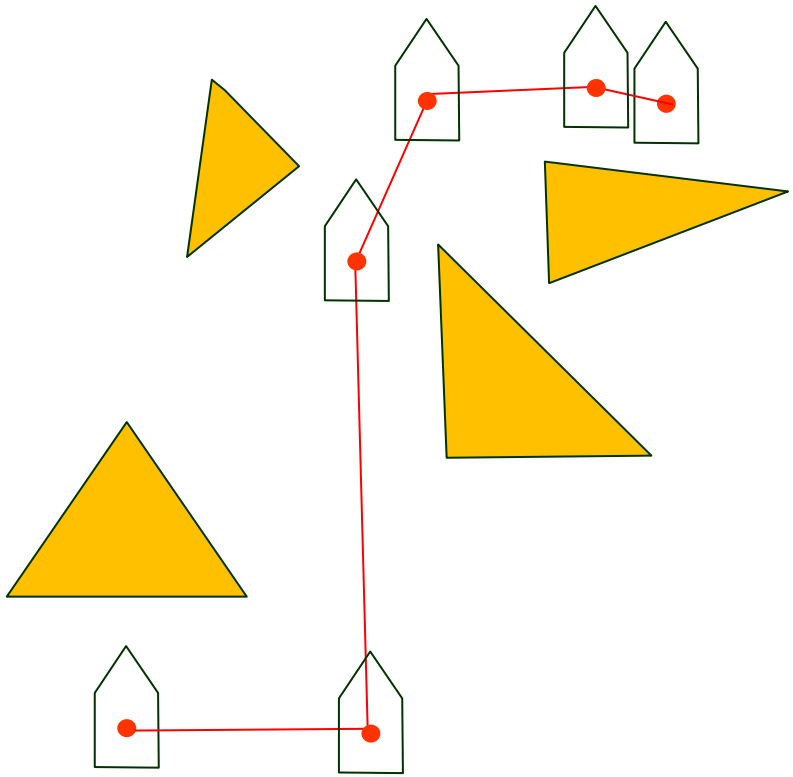
Path

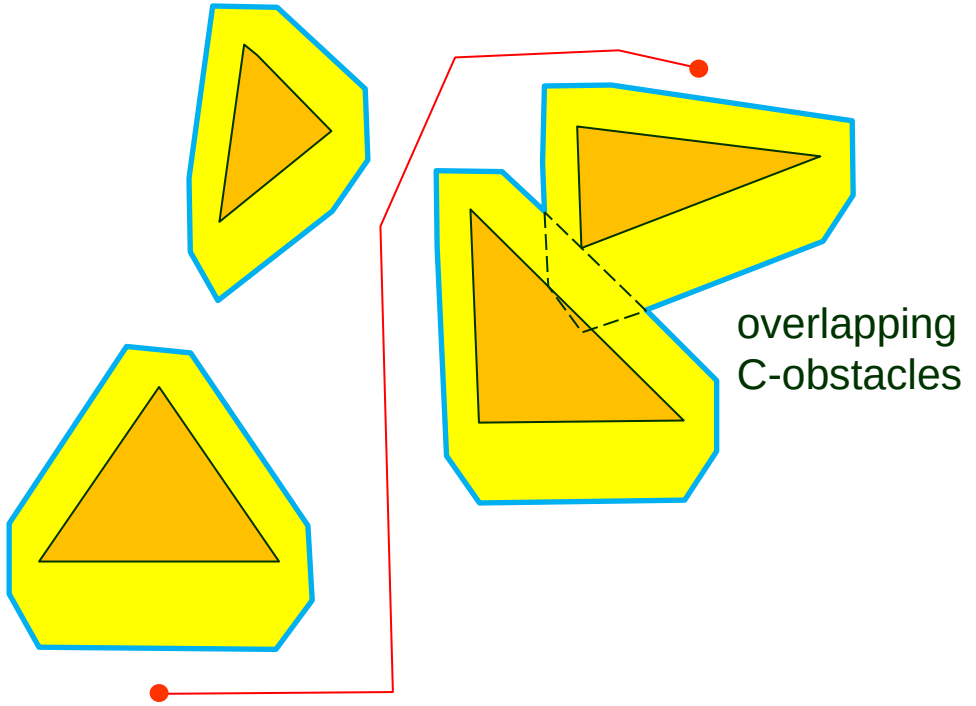
Work space: C-space:

robot ↦ point

obstacle ↦ C-obstacles (possibly joined)

path ↦ curve





Point Robot

Work space = C-space & every obstacle = its C-obstacle

Point Robot

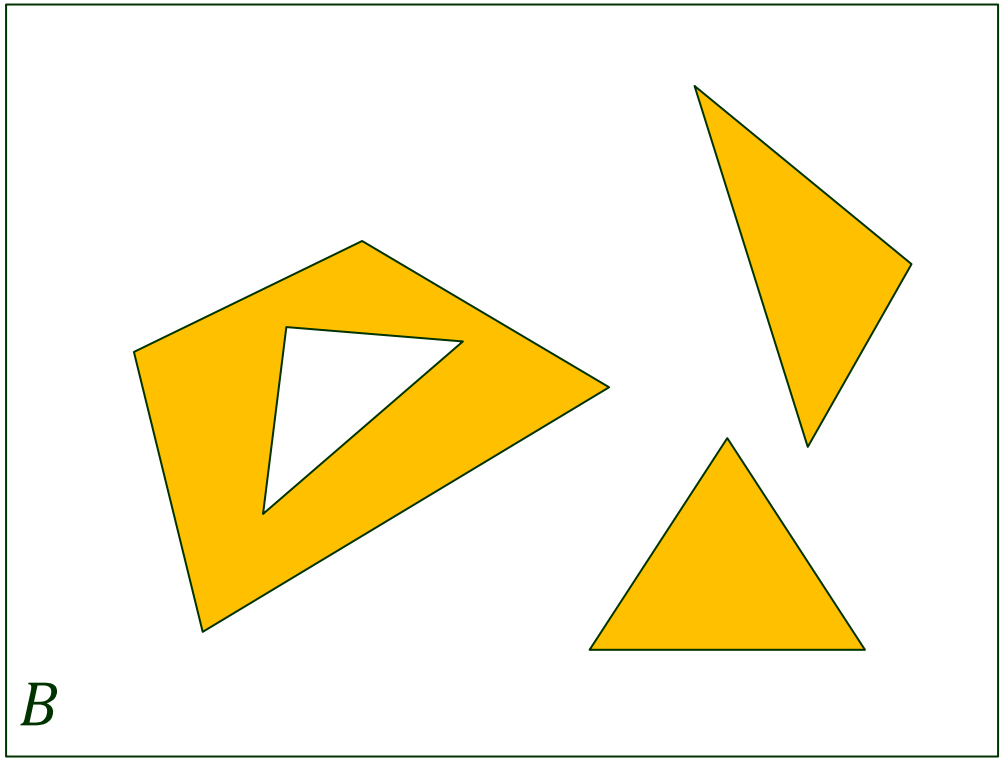
Work space = C-space & every obstacle = its C-obstacle

- Obstacles $P_1, P_2, \ldots, P_t$ have disjoint interiors.

Point Robot

Work space = C-space & every obstacle = its C-obstacle

- Obstacles $P_1, P_2, \ldots, P_t$ have disjoint interiors.
- Bounding box B contains all obstacles.

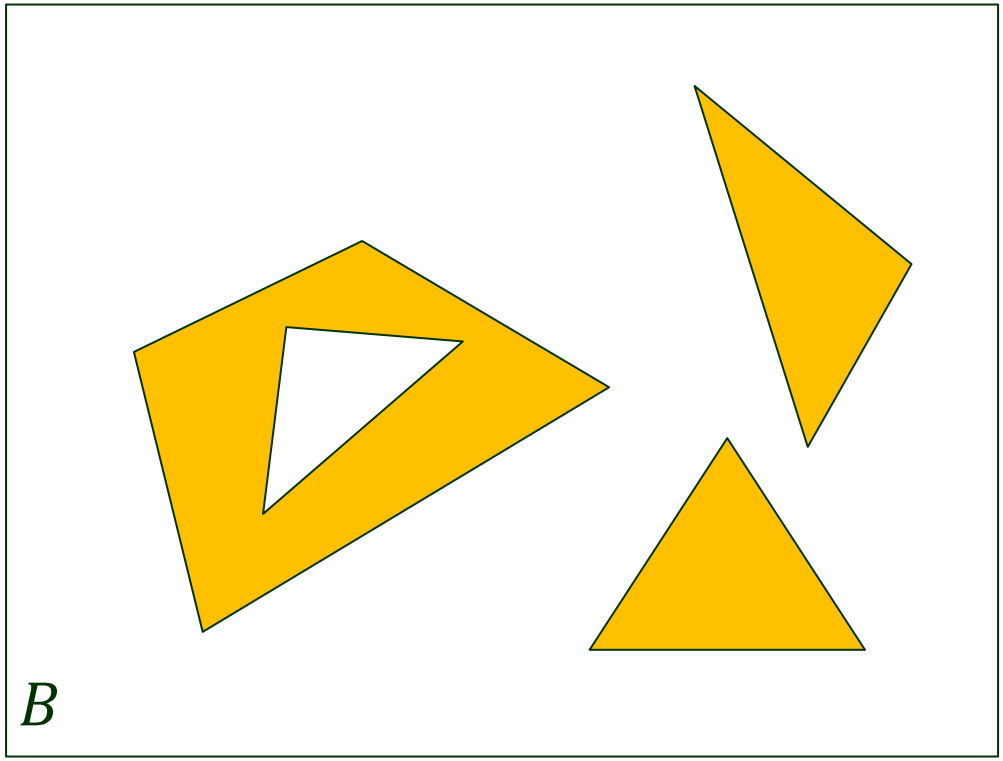


Point Robot

Work space = C-space & every obstacle = its C-obstacle

- Obstacles $P_1, P_2, \ldots, P_t$ have disjoint interiors.
- Bounding box B contains all obstacles.

$$C_{free} = B \setminus \bigcup_{i=1}^{t} P_i$$



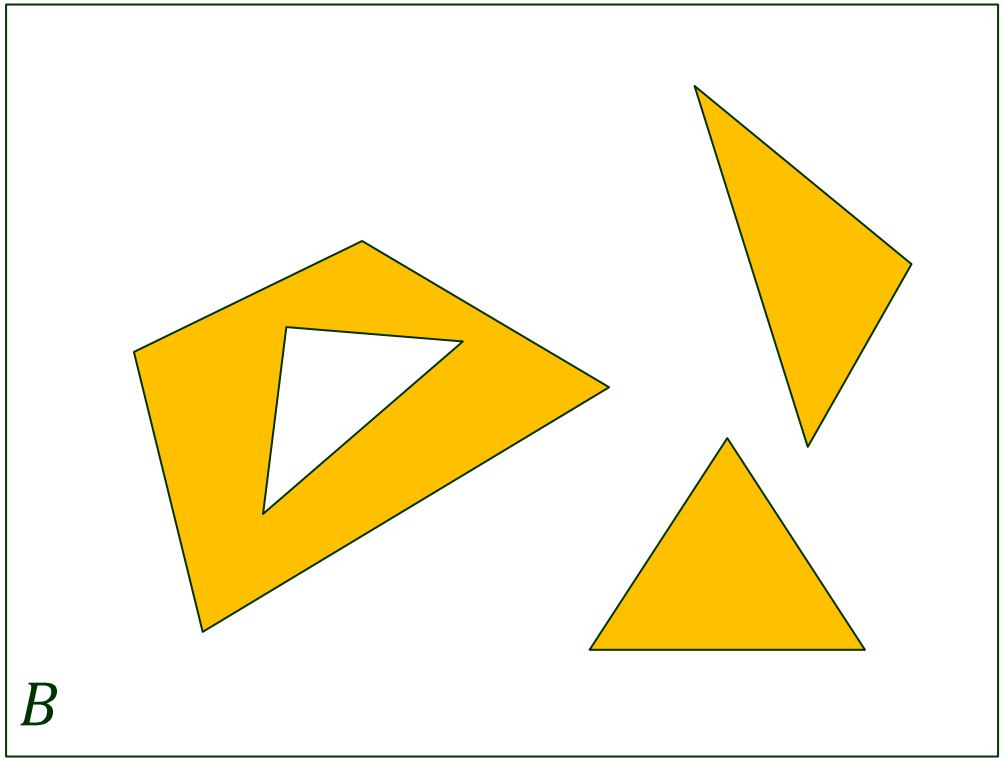
Point Robot

Work space = C-space & every obstacle = its C-obstacle

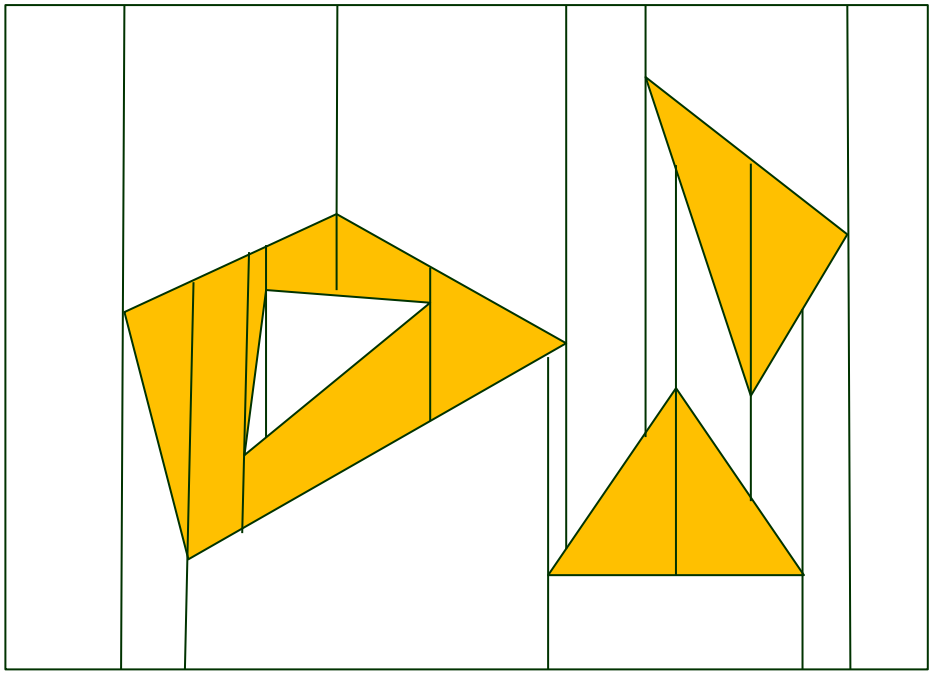
- Obstacles $P_1, P_2, \ldots, P_t$ have disjoint interiors.
- Bounding box B contains all obstacles.

$$C_{free} = B \setminus \bigcup_{i=1}^{t} P_i$$

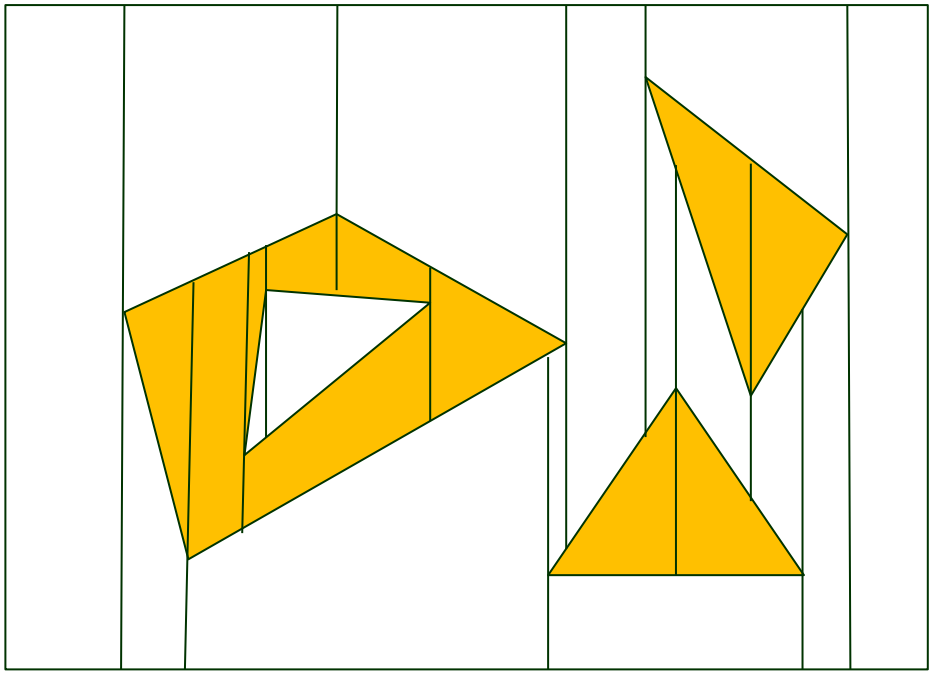
  - Also the free work space.
  - Possibly disconnected.



C_{free} as a Trapezoidal Map

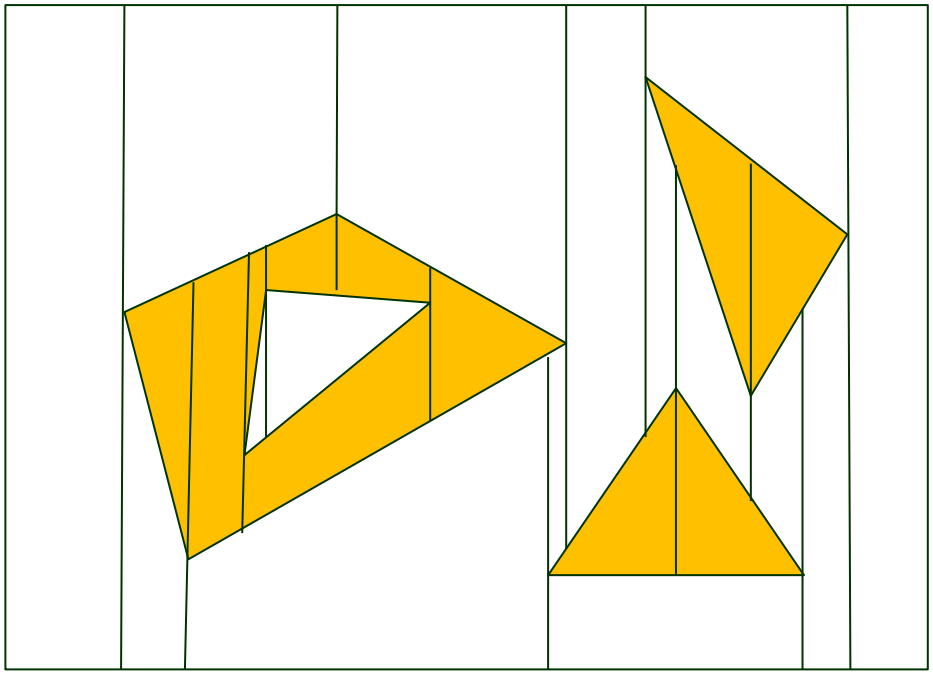


C_{free} as a Trapezoidal Map



Remove trapezoids lying inside the obstacles

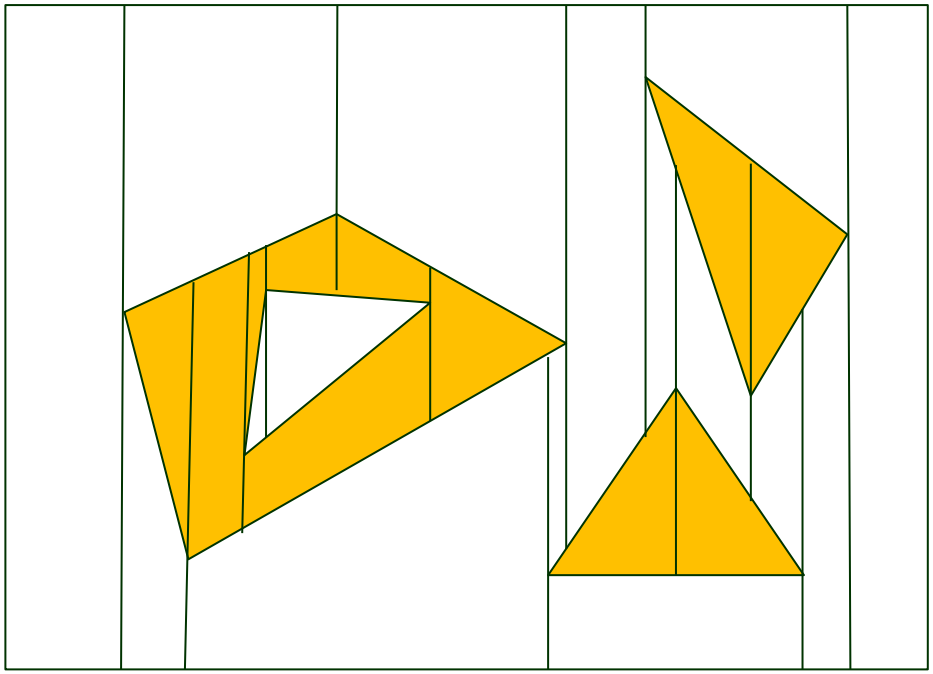
C_{free} as a Trapezoidal Map



Remove trapezoids lying inside the obstacles

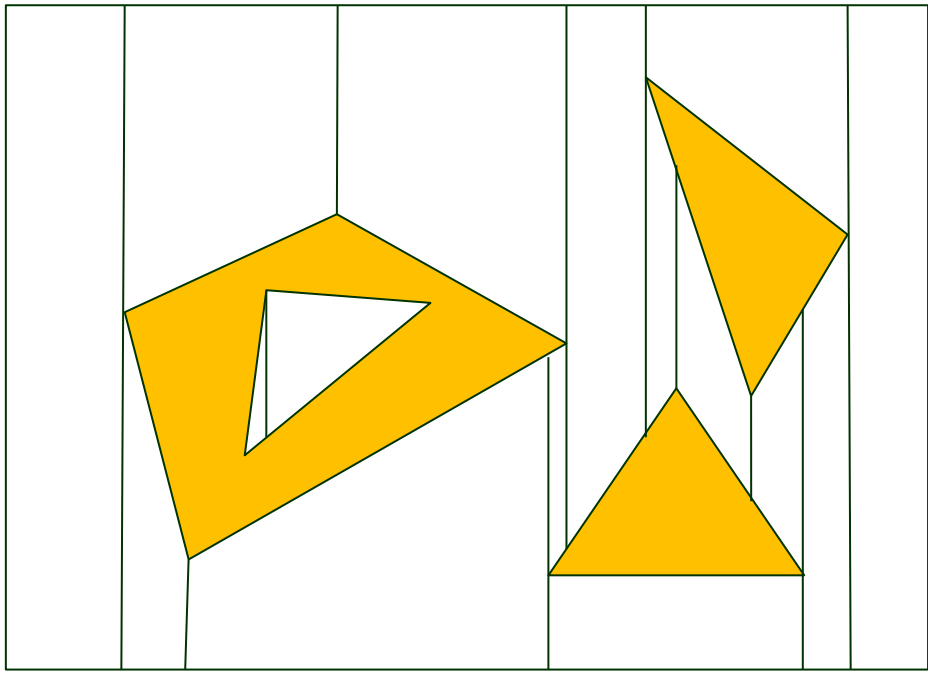
A trapezoid is removed if its top edge bounds an obstacle from above.

C_{free} as a Trapezoidal Map



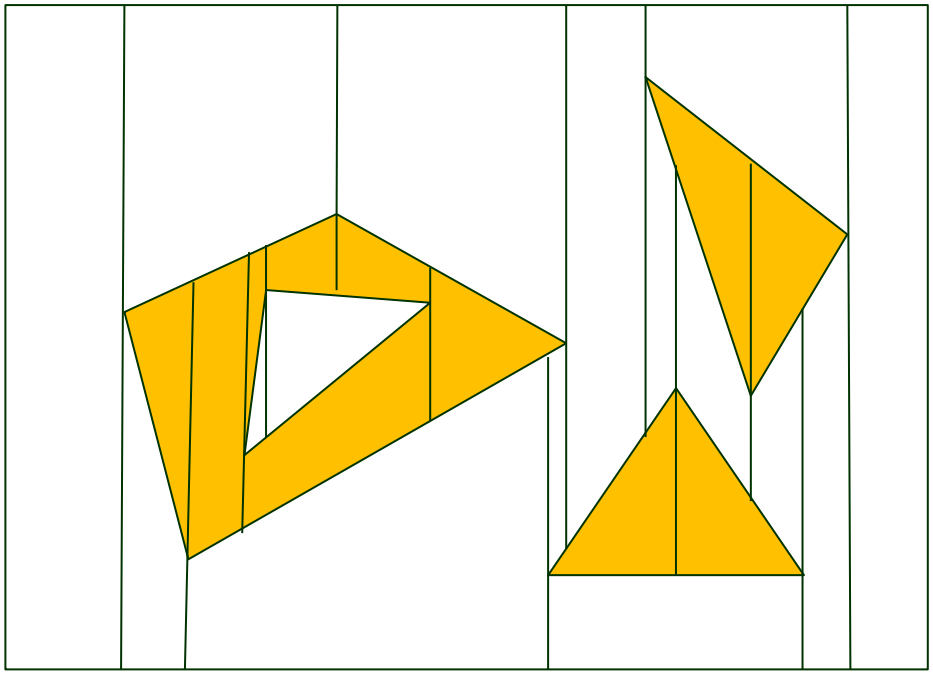
Remove trapezoids lying inside the obstacles

$T(C_{free})$



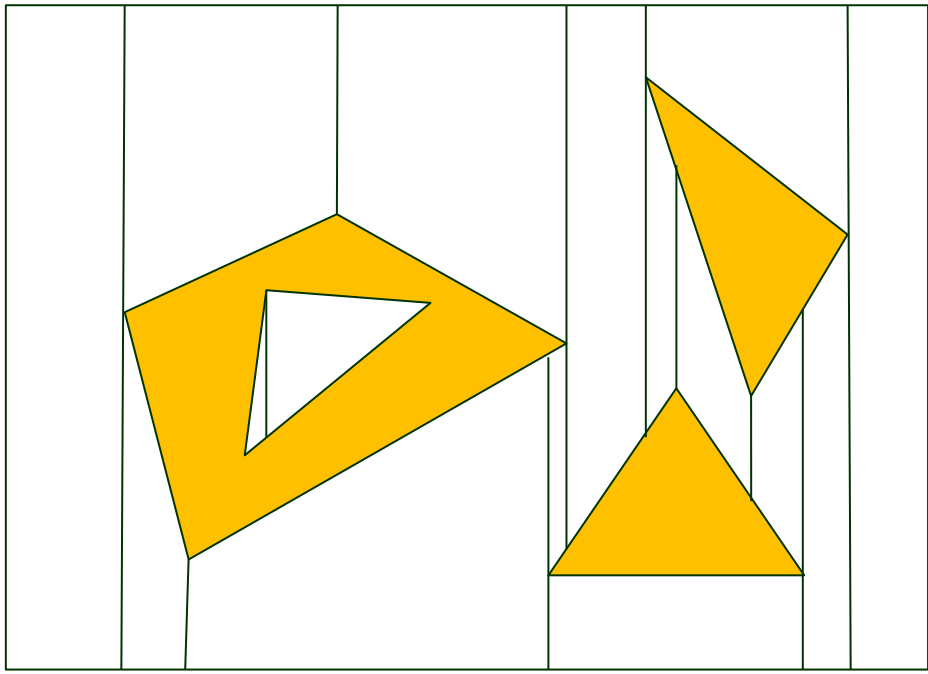
A trapezoid is removed if its top edge bounds an obstacle from above.

C_{free} as a Trapezoidal Map



Remove trapezoids lying inside the obstacles

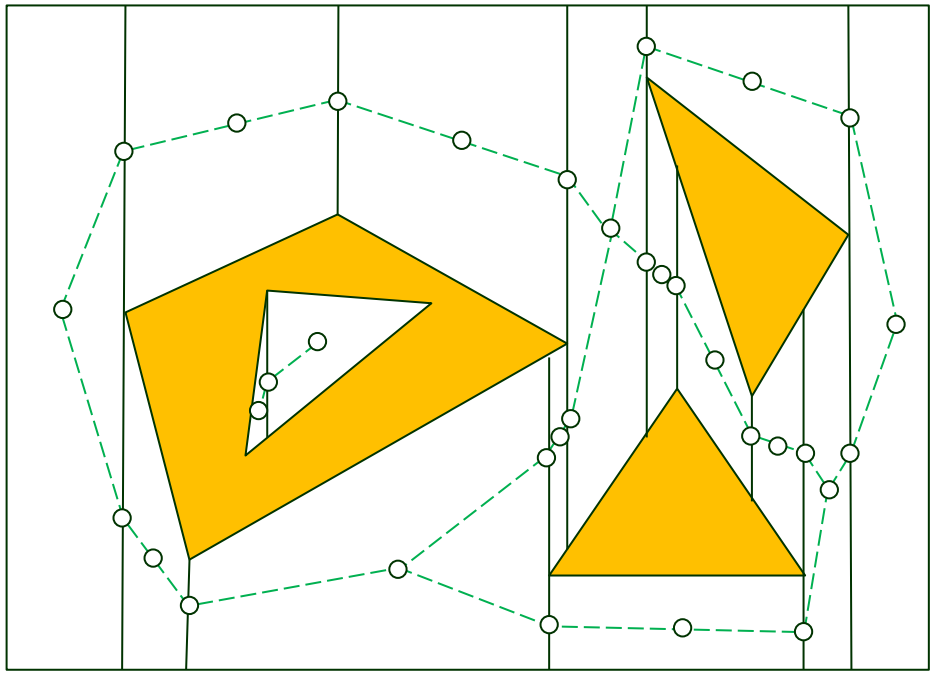
$T(C_{free})$



A trapezoid is removed if its top edge bounds an obstacle from above.

$O(n \log n)$ expected time.

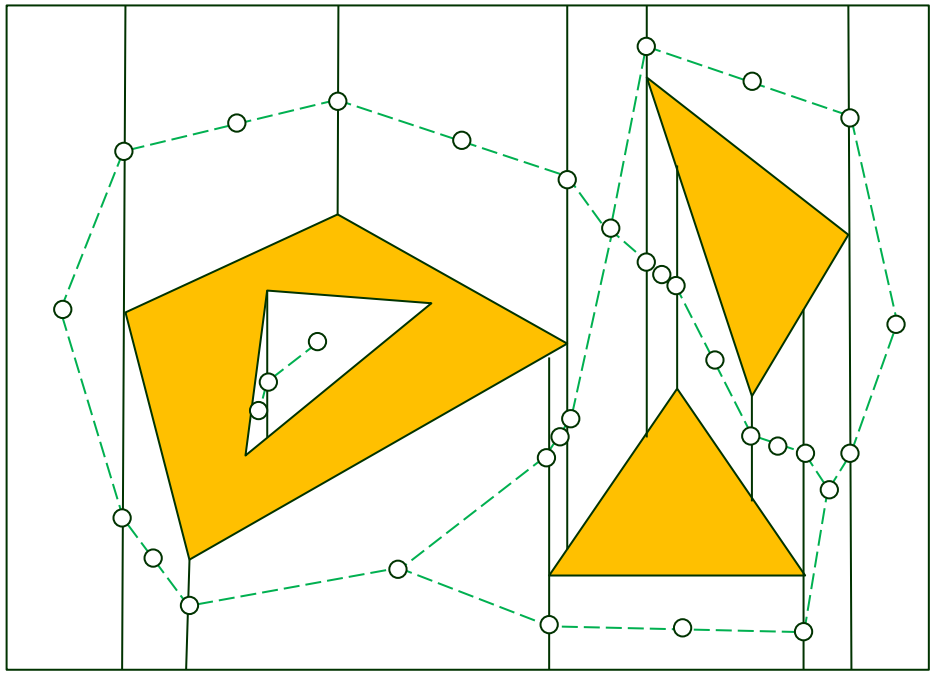
III. Roadmap



A *planar graph* embedded in C_{free} and formed by adding

- one node in the center of each trapezoid;
- one node in the middle of each extension;
- an arc between two nodes if one is in the center of a trapezoid while the other is on the boundary.

III. Roadmap

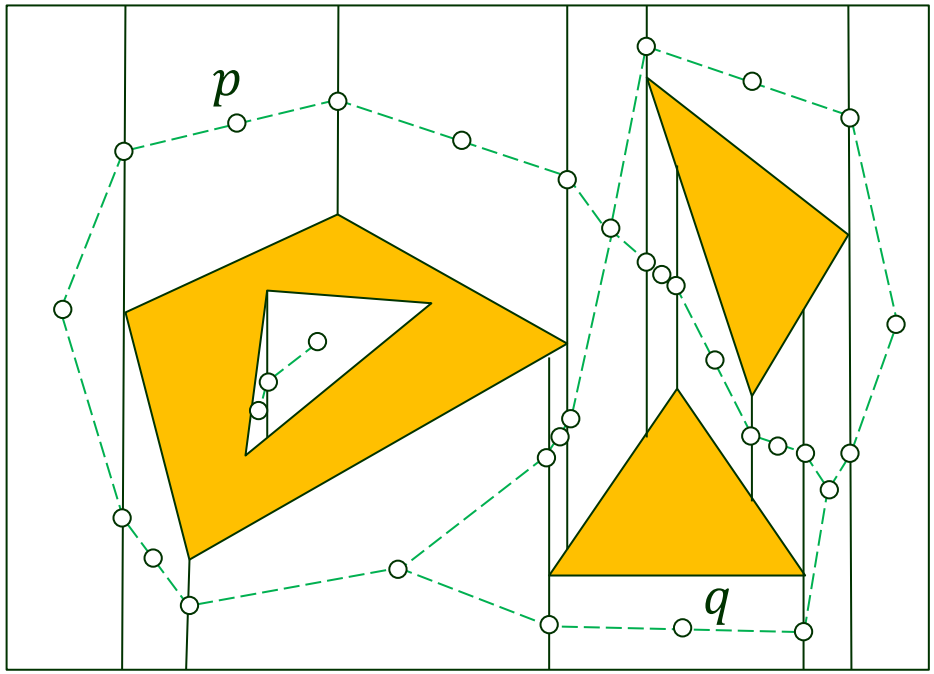


A *planar graph* embedded in C_{free} and formed by adding

- one node in the center of each trapezoid;
- one node in the middle of each extension;
- an arc between two nodes if one is in the center of a trapezoid while the other is on the boundary.

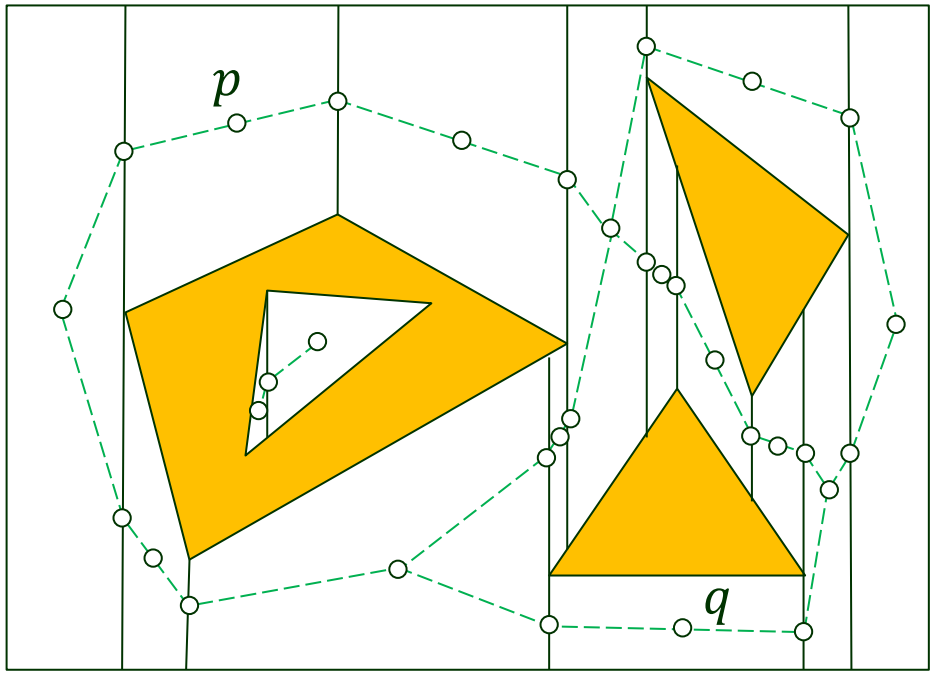
The roadmap is constructed by traversing the DCEL of $T(C_{free})$.

Roadmap Construction



Traverse the DCEL of the trapezoidal map $T(C_{free})$.

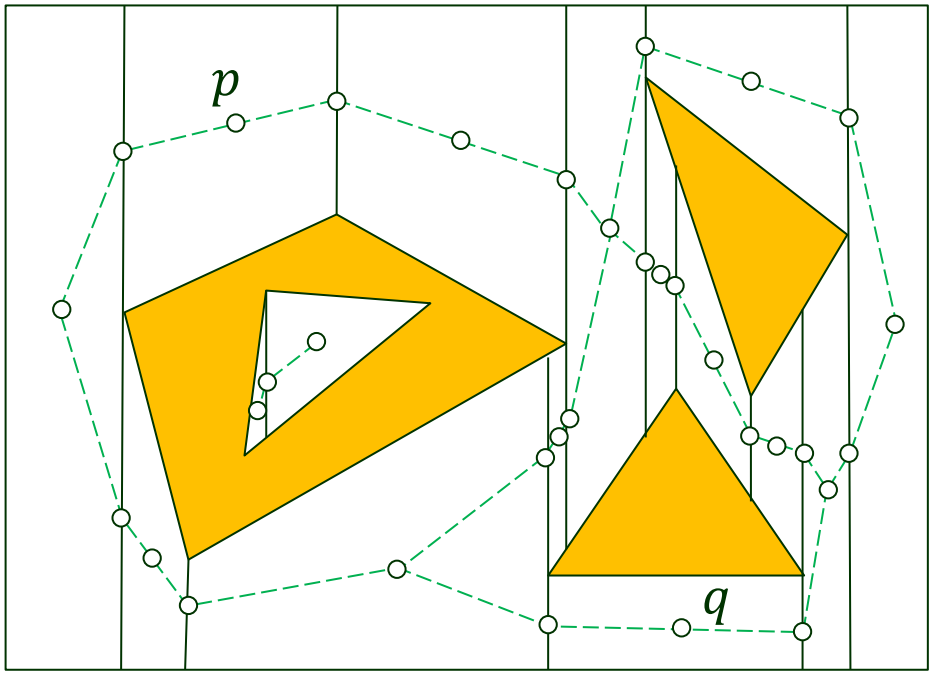
Roadmap Construction



Traverse the DCEL of the trapezoidal map $T(C_{free})$.

$O(n)$

Roadmap Construction

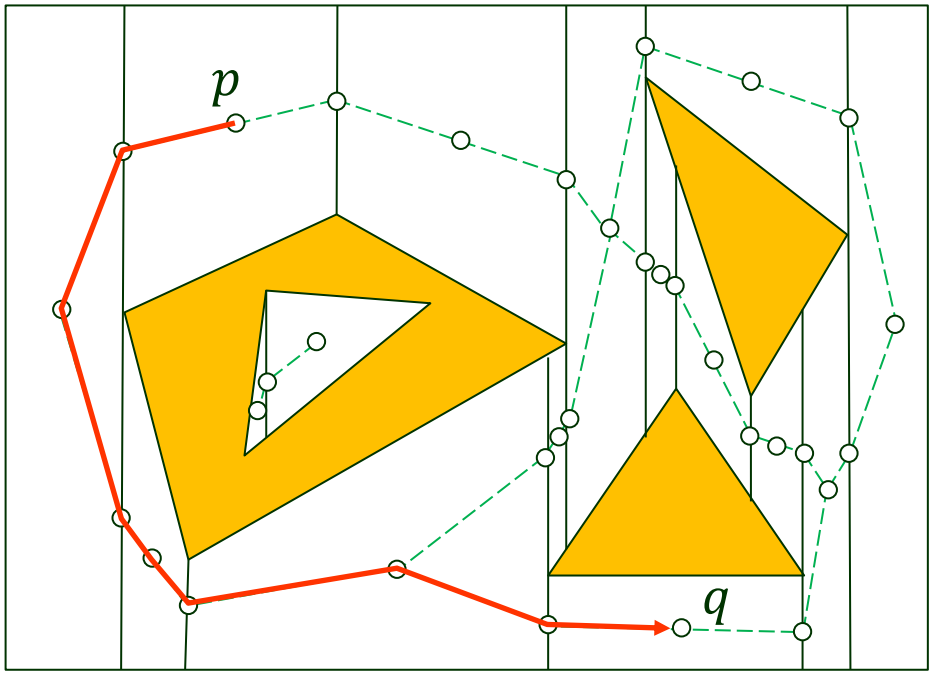


Traverse the DCEL of the trapezoidal map $T(C_{free})$.

$O(n)$

We can navigate from the node representing the center of one trapezoid to that representing the center of any other reachable trapezoid.

Roadmap Construction

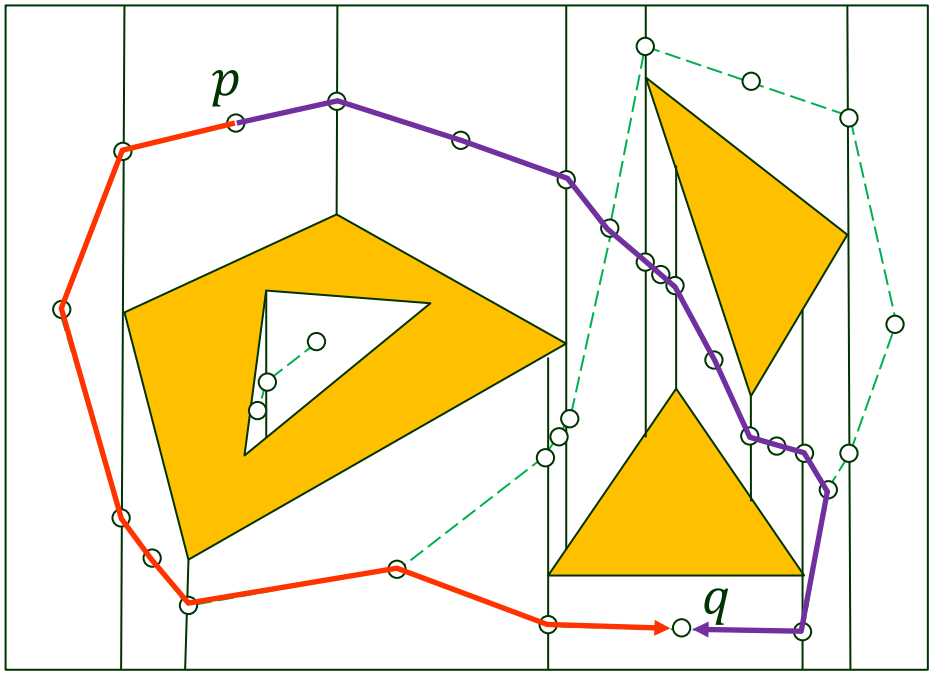


Traverse the DCEL of the trapezoidal map $T(C_{free})$.

$O(n)$

We can navigate from the node representing the center of one trapezoid to that representing the center of any other reachable trapezoid.

Roadmap Construction

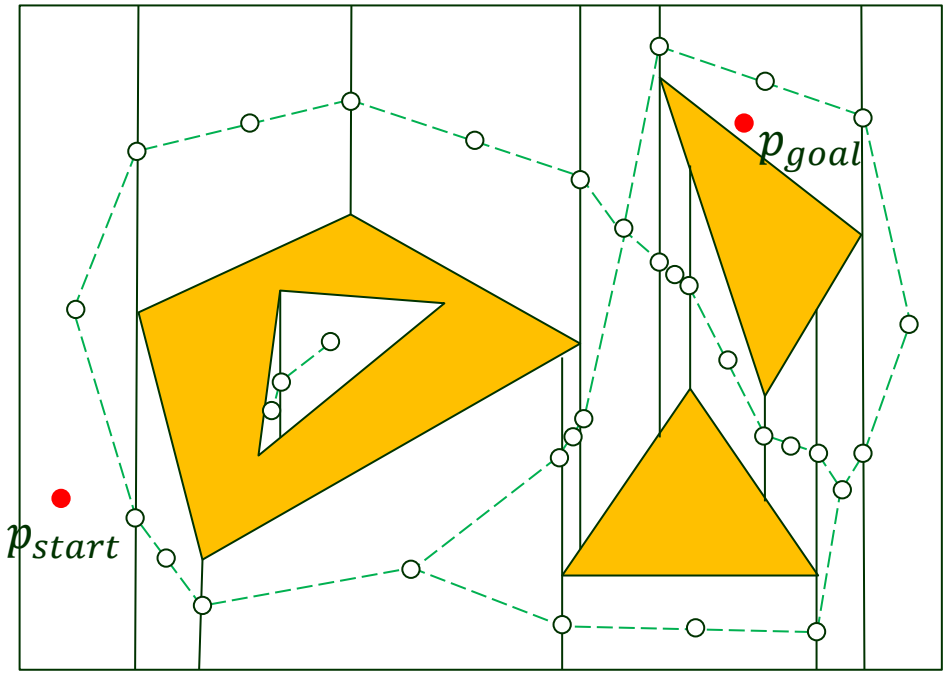


Traverse the DCEL of the trapezoidal map $T(C_{free})$.

$O(n)$

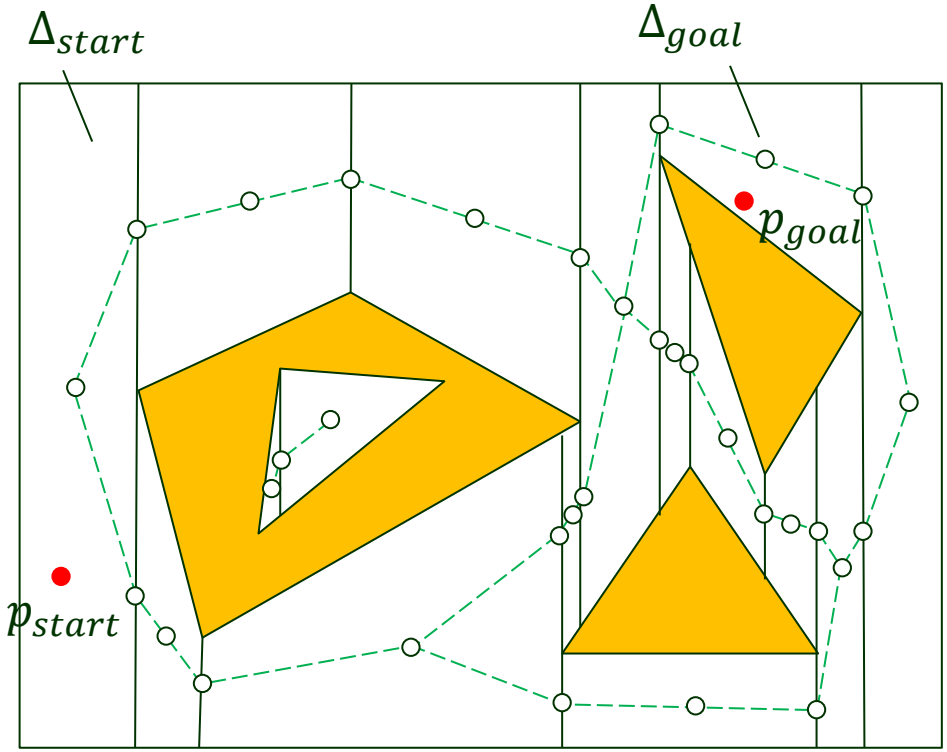
We can navigate from the node representing the center of one trapezoid to that representing the center of any other reachable trapezoid.

Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

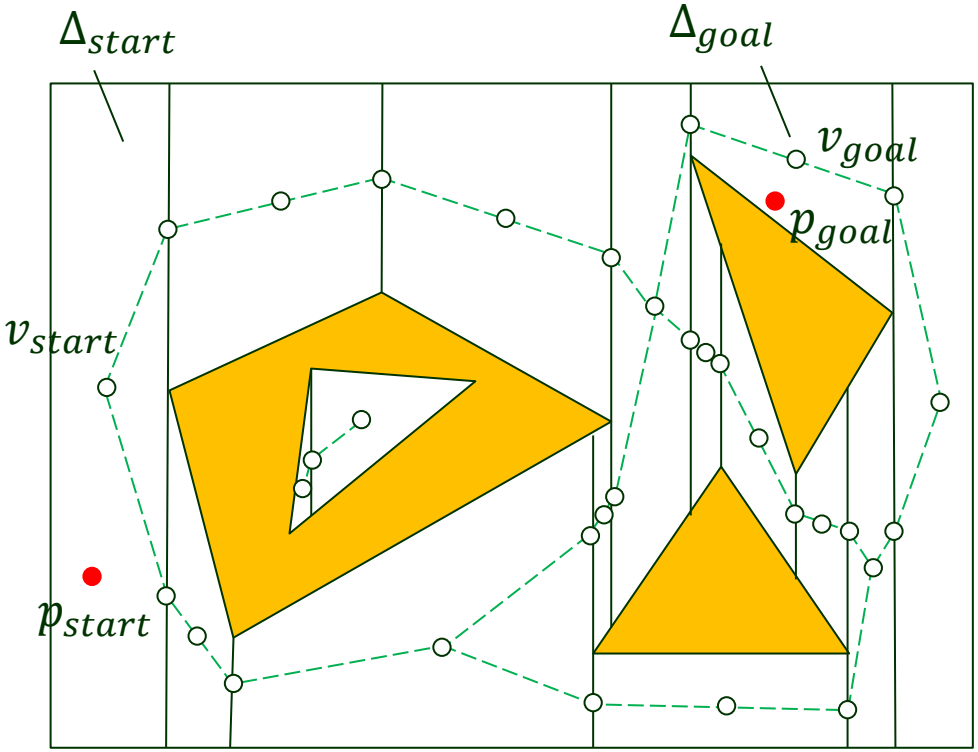
Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.

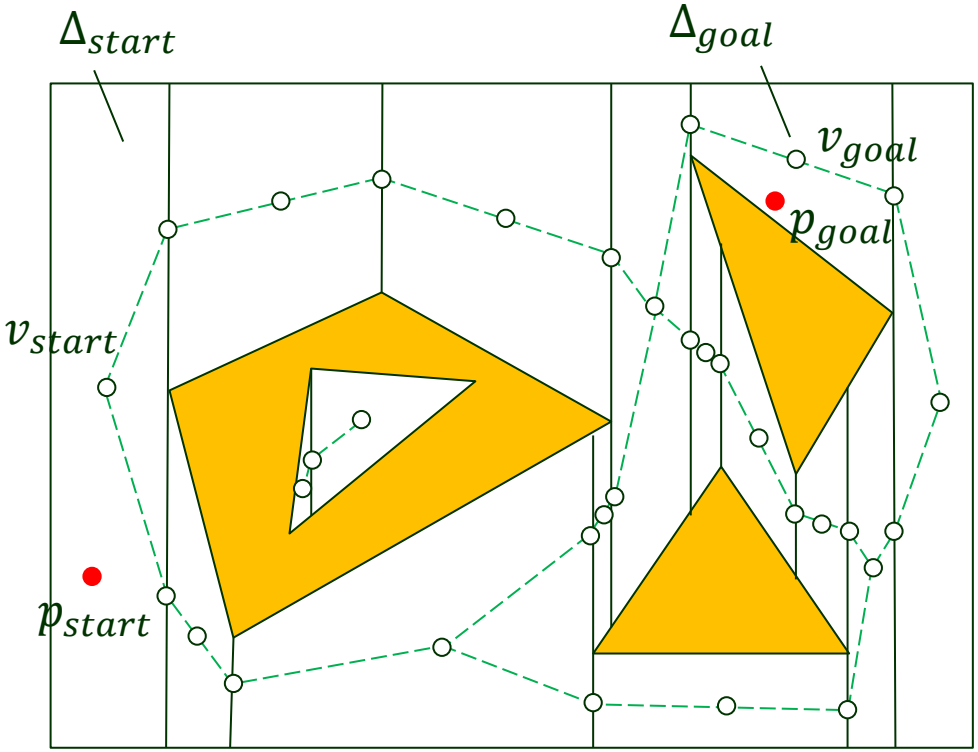
Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.
- Let v_{start} and v_{goal} be the nodes at the centers of Δ_{start} and Δ_{goal}.

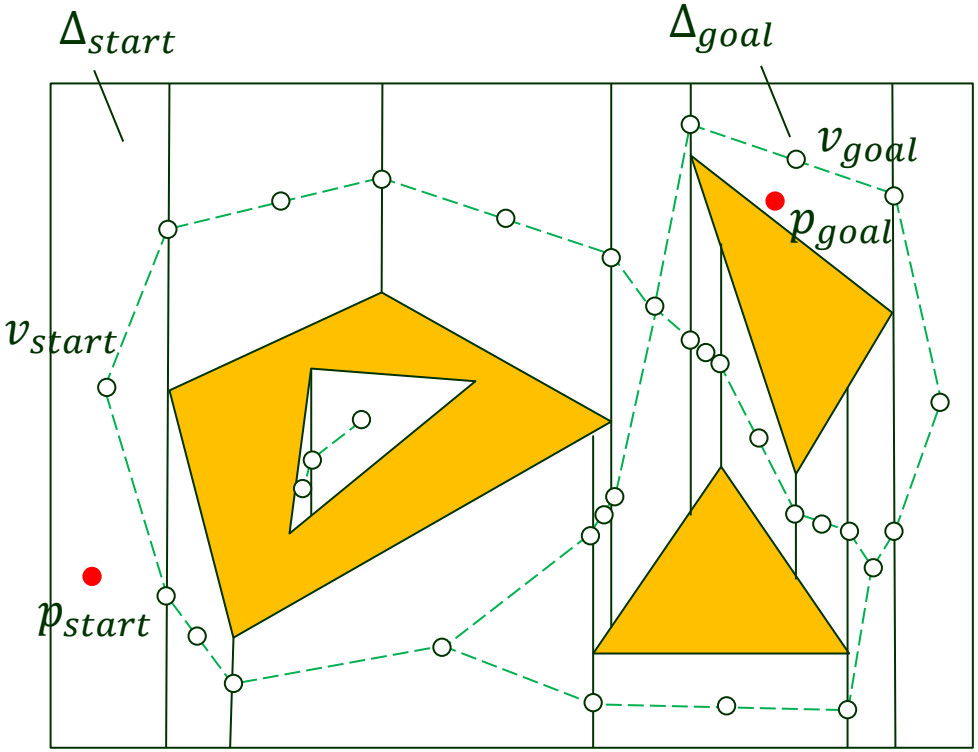
Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.
- Let v_{start} and v_{goal} be the nodes at the centers of Δ_{start} and Δ_{goal}.
- Three parts of the path $p_{start} \rightsquigarrow p_{goal}$:

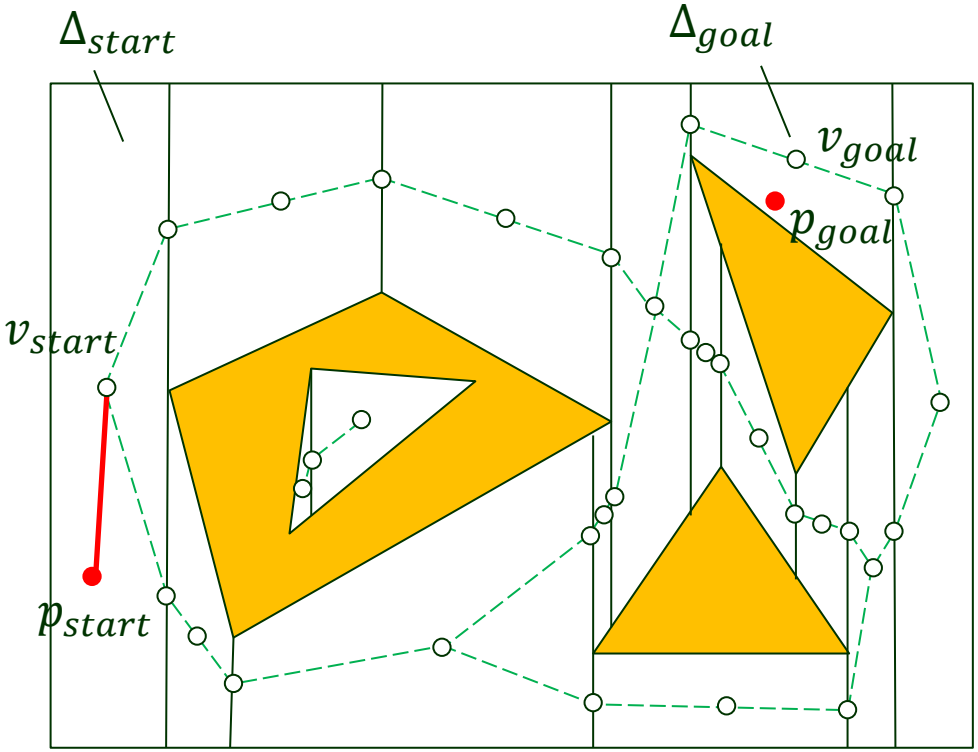
Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.
- Let v_{start} and v_{goal} be the nodes at the centers of Δ_{start} and Δ_{goal}.
- Three parts of the path $p_{start} \rightsquigarrow p_{goal}$:
  - straight line segment $\overline{p_{start}v_{start}}$;

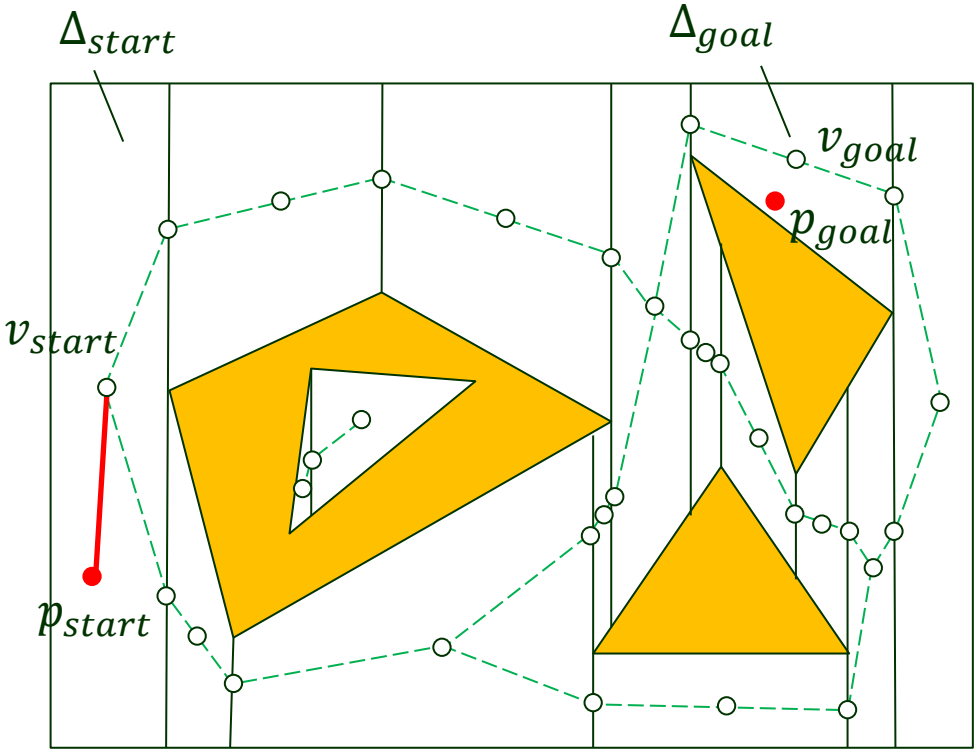
Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.
- Let v_{start} and v_{goal} be the nodes at the centers of Δ_{start} and Δ_{goal}.
- Three parts of the path $p_{start} \rightsquigarrow p_{goal}$:
  - straight line segment $\overline{p_{start}v_{start}}$;

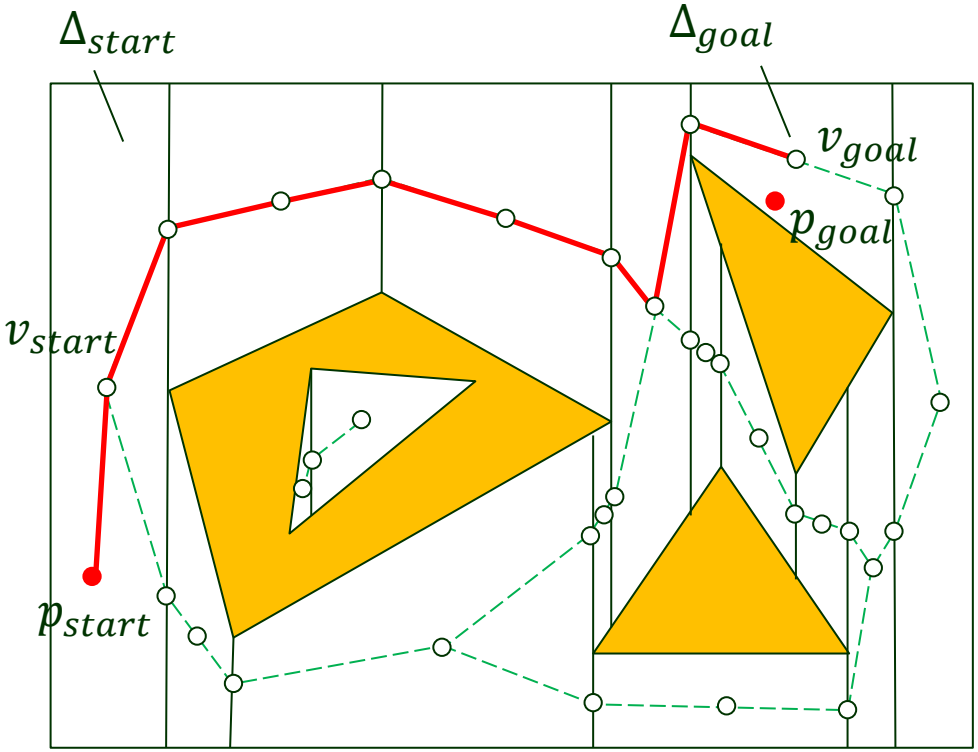
Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.
- Let v_{start} and v_{goal} be the nodes at the centers of Δ_{start} and Δ_{goal}.
- Three parts of the path $p_{start} \rightsquigarrow p_{goal}$:
  - straight line segment $\overline{p_{start}v_{start}}$;
  - path $v_{start} \rightsquigarrow v_{goal}$ (via BFS);

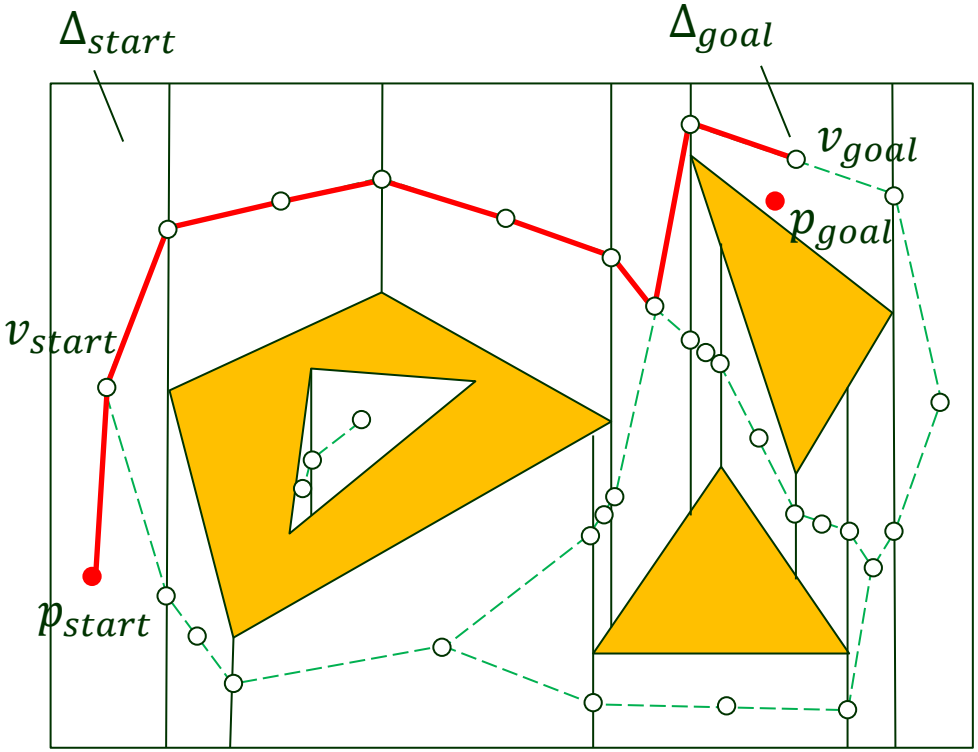
Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.
- Let v_{start} and v_{goal} be the nodes at the centers of Δ_{start} and Δ_{goal}.
- Three parts of the path $p_{start} \rightsquigarrow p_{goal}$:
  - straight line segment $\overline{p_{start}v_{start}}$;
  - path $v_{start} \rightsquigarrow v_{goal}$ (via BFS);

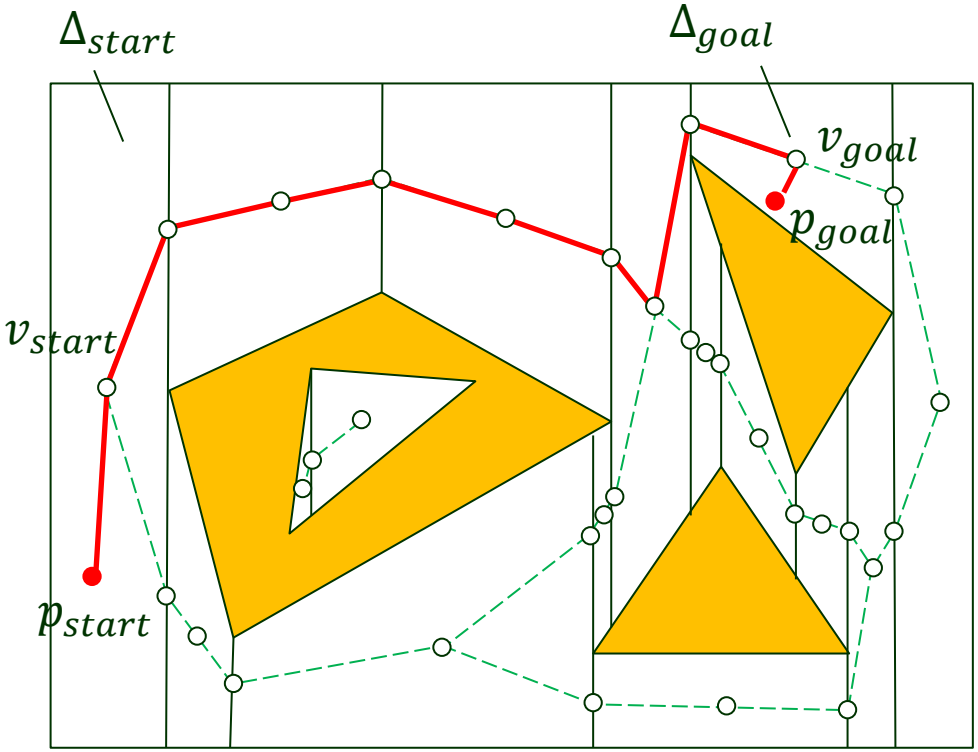
Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.
- Let v_{start} and v_{goal} be the nodes at the centers of Δ_{start} and Δ_{goal}.
- Three parts of the path $p_{start} \rightsquigarrow p_{goal}$:
  - straight line segment $\overline{p_{start}v_{start}}$;
  - path $v_{start} \rightsquigarrow v_{goal}$ (via BFS);
  - straight line segment $\overline{v_{goal}p_{goal}}$.

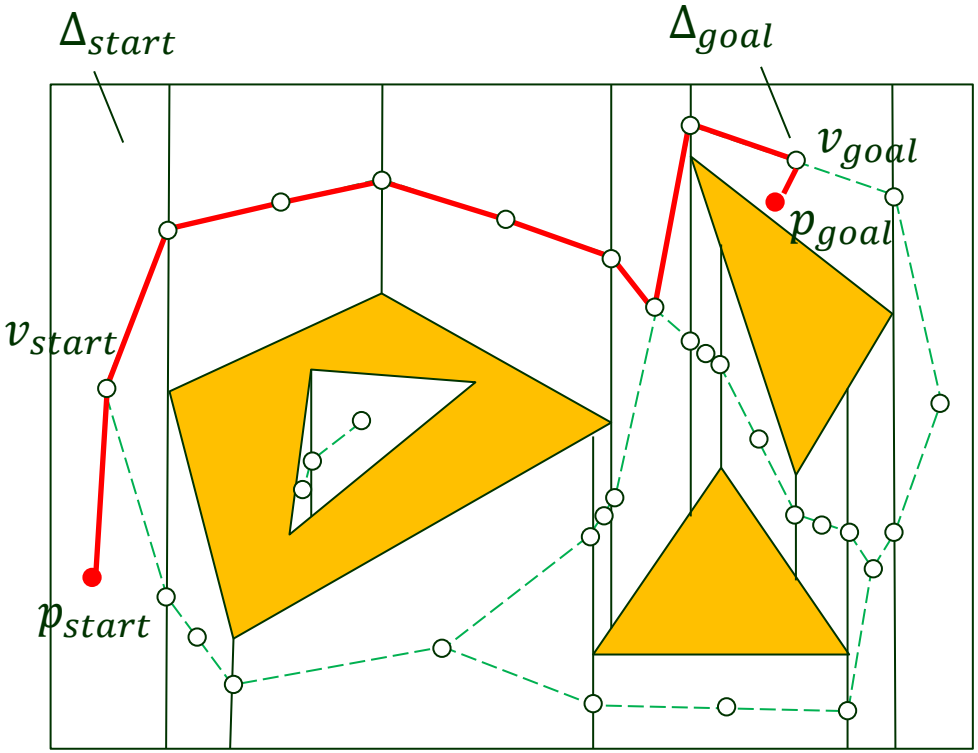
Motion Planning for a Point Robot



Input: starting position p_{start} and goal position p_{goal}.

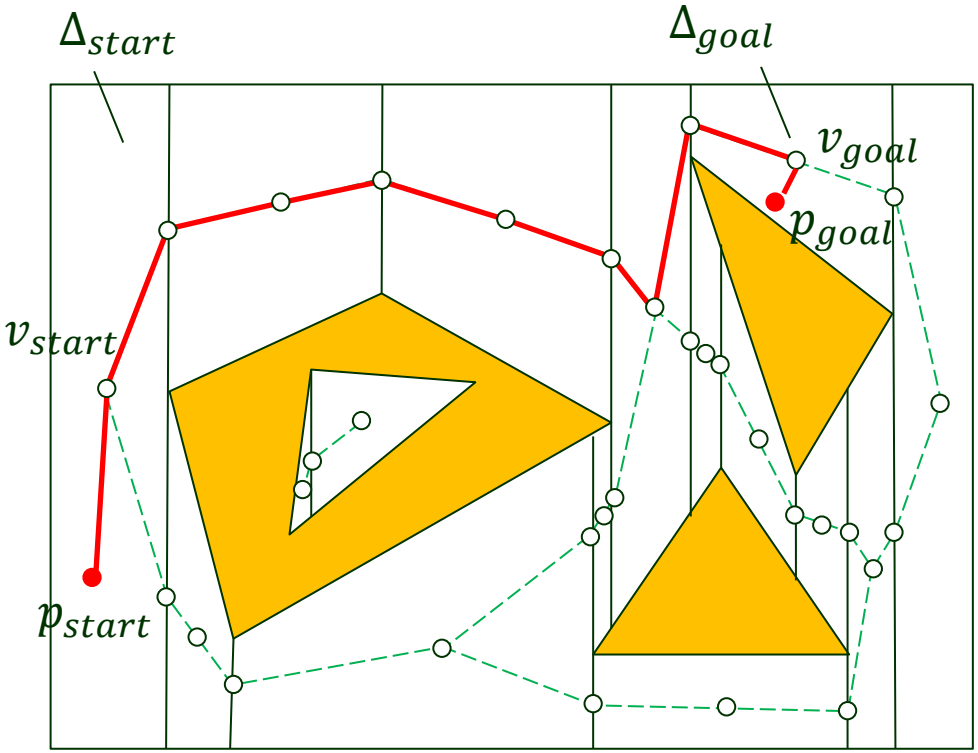
- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.
- Let v_{start} and v_{goal} be the nodes at the centers of Δ_{start} and Δ_{goal}.
- Three parts of the path $p_{start} \rightsquigarrow p_{goal}$:
  - straight line segment $\overline{p_{start}v_{start}}$;
  - path $v_{start} \rightsquigarrow v_{goal}$ (via BFS);
  - straight line segment $\overline{v_{goal}p_{goal}}$.

Collison-Free Path



Correctness hinges on two questions:

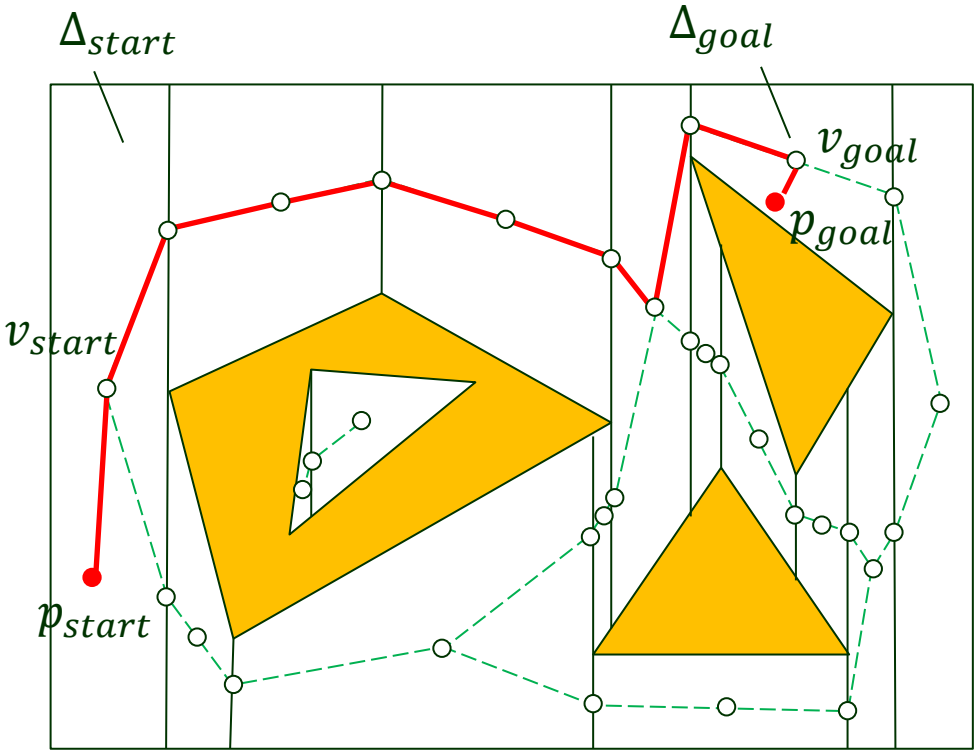
Collison-Free Path



Correctness hinges on two questions:

- Is the found path collision free?

Collison-Free Path



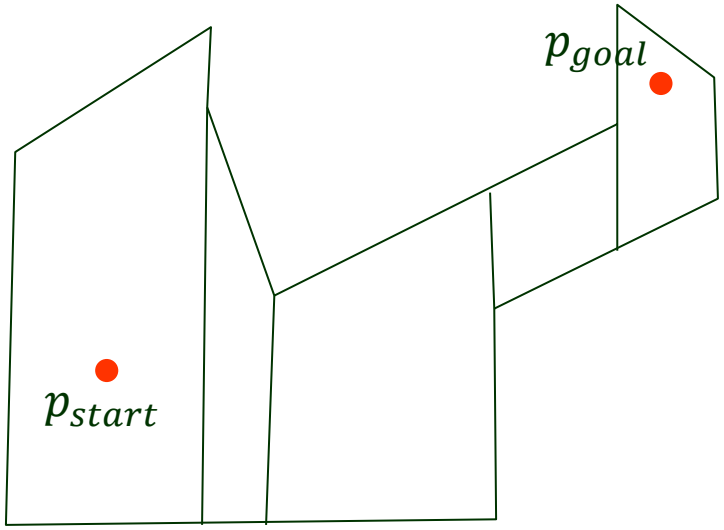
Correctness hinges on two questions:

- Is the found path collision free?

Yes, because it consists of segments inside trapezoids (which are in C_{free}).

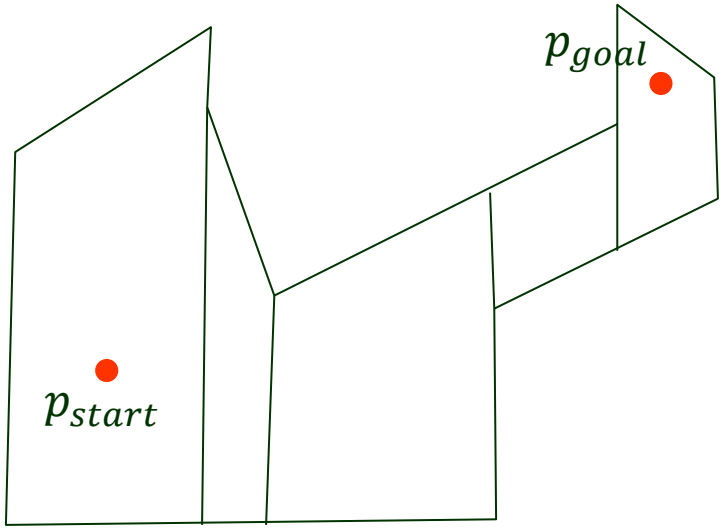
Completeness

- Does the algorithm always find a collision-free path whenever one exists?



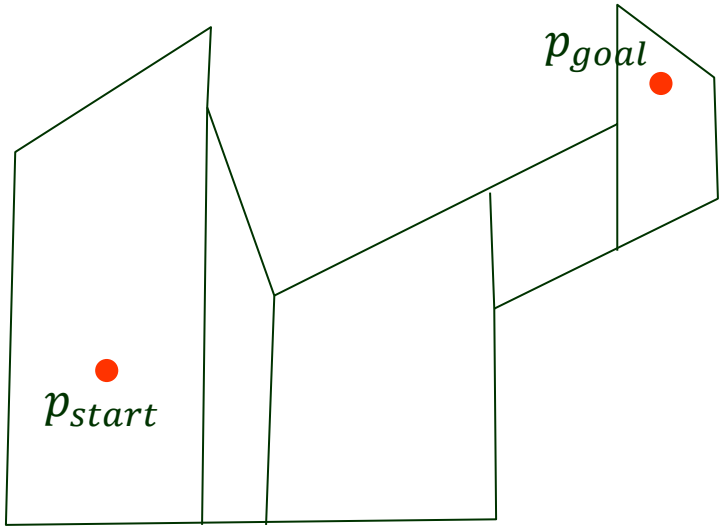
Completeness

- Does the algorithm always find a collision-free path whenever one exists?
  - Suppose there is a collision-free path $p_{start} \rightsquigarrow p_{goal}$.



Completeness

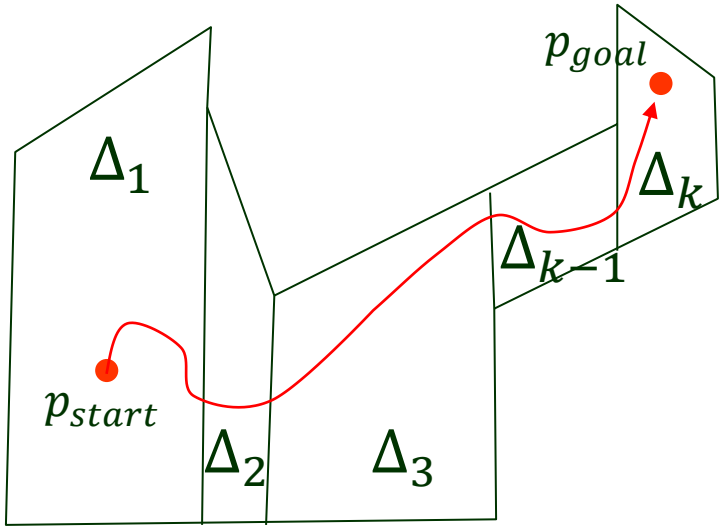
- Does the algorithm always find a collision-free path whenever one exists?



- Suppose there is a collision-free path $p_{start} \rightsquigarrow p_{goal}$.
- Δ_{start} and Δ_{goal} are trapezoids $\subseteq C_{free}$.

Completeness

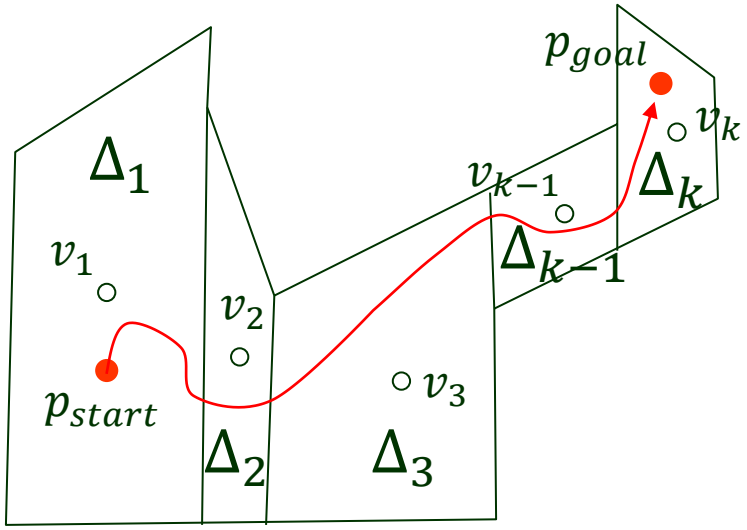
- Does the algorithm always find a collision-free path whenever one exists?



- Suppose there is a collision-free path $p_{start} \rightsquigarrow p_{goal}$.
- Δ_{start} and Δ_{goal} are trapezoids $\subseteq C_{free}$.
- Path $p_{start} \rightsquigarrow p_{goal}$ crosses trapezoids $\Delta_1 = \Delta_{start}, \Delta_2, \ldots, \Delta_k = \Delta_{goal}$.

Completeness

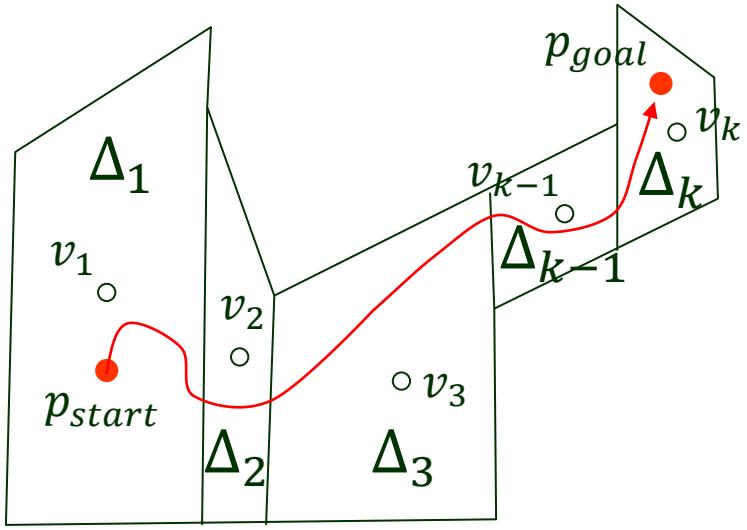
- Does the algorithm always find a collision-free path whenever one exists?



- Suppose there is a collision-free path $p_{start} \rightsquigarrow p_{goal}$.
- Δ_{start} and Δ_{goal} are trapezoids $\subseteq C_{free}$.
- Path $p_{start} \rightsquigarrow p_{goal}$ crosses trapezoids $\Delta_1 = \Delta_{start}, \Delta_2, \ldots, \Delta_k = \Delta_{goal}$.

Completeness

- Does the algorithm always find a collision-free path whenever one exists?



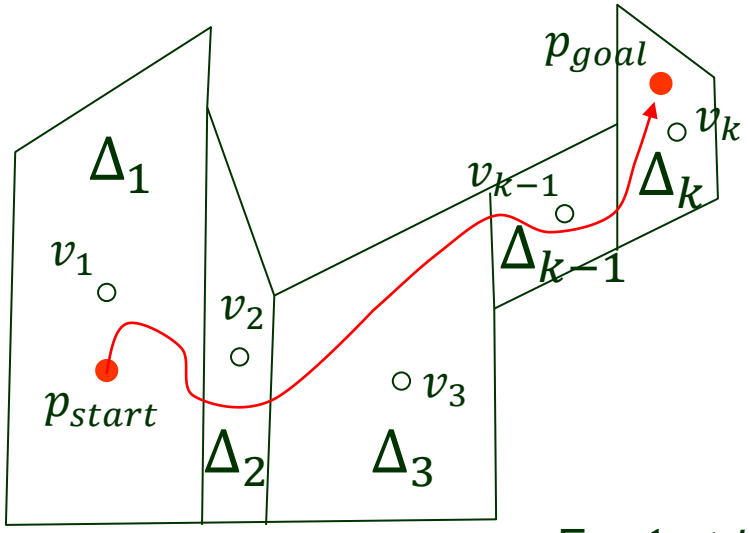
- Suppose there is a collision-free path $p_{start} \rightsquigarrow p_{goal}$.
- Δ_{start} and Δ_{goal} are trapezoids $\subseteq C_{free}$.
- Path $p_{start} \rightsquigarrow p_{goal}$ crosses trapezoids $\Delta_1 = \Delta_{start}, \Delta_2, \ldots, \Delta_k = \Delta_{goal}$.

  ⇩ Node v_i at the center of Δ_i

  It suffices to show that a collision-free path $v_1 \rightsquigarrow v_k$ can be found.

Completeness

- Does the algorithm always find a collision-free path whenever one exists?



- Suppose there is a collision-free path $p_{start} \rightsquigarrow p_{goal}$.
- Δ_{start} and Δ_{goal} are trapezoids $\subseteq C_{free}$.
- Path $p_{start} \rightsquigarrow p_{goal}$ crosses trapezoids $\Delta_1 = \Delta_{start}, \Delta_2, \ldots, \Delta_k = \Delta_{goal}$.

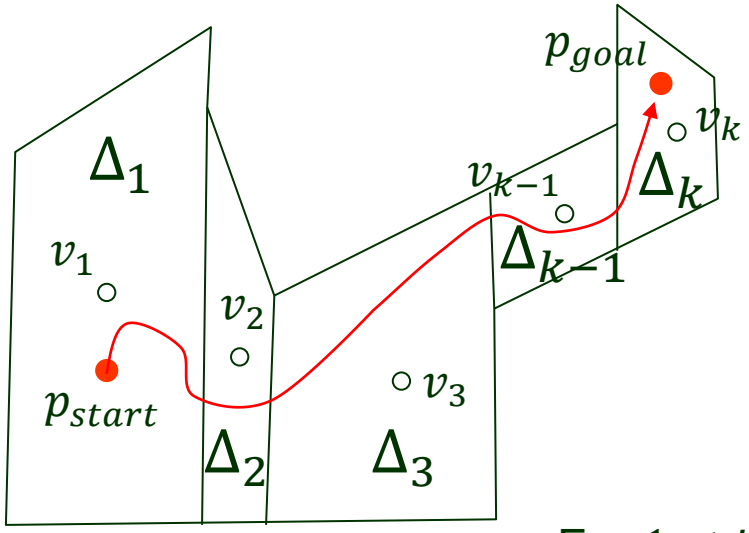
  ⇩ Node v_i at the center of Δ_i

  It suffices to show that a collision-free path $v_1 \rightsquigarrow v_k$ can be found.

- For $1 \leq i \leq k-1$, Δ_i borders Δ_{i+1} (at a vertical extension)

Completeness

- Does the algorithm always find a collision-free path whenever one exists?



- Suppose there is a collision-free path $p_{start} \rightsquigarrow p_{goal}$.
- Δ_{start} and Δ_{goal} are trapezoids $\subseteq C_{free}$.
- Path $p_{start} \rightsquigarrow p_{goal}$ crosses trapezoids $\Delta_1 = \Delta_{start}, \Delta_2, \ldots, \Delta_k = \Delta_{goal}$.

⇩ Node v_i at the center of Δ_i

It suffices to show that a collision-free path $v_1 \rightsquigarrow v_k$ can be found.

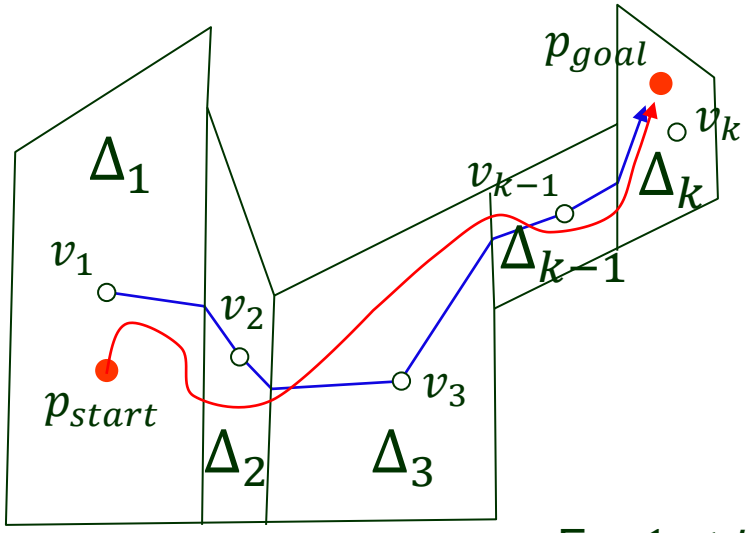
- For $1 \leq i \leq k-1$, Δ_i borders Δ_{i+1} (at a vertical extension)

⇩

Collision free paths $v_i \rightsquigarrow v_{i+1}$ exist for $1 \leq i \leq k-1$.

Completeness

- Does the algorithm always find a collision-free path whenever one exists?



- Suppose there is a collision-free path $p_{start} \rightsquigarrow p_{goal}$.
- Δ_{start} and Δ_{goal} are trapezoids $\subseteq C_{free}$.
- Path $p_{start} \rightsquigarrow p_{goal}$ crosses trapezoids $\Delta_1 = \Delta_{start}, \Delta_2, \ldots, \Delta_k = \Delta_{goal}$.

⇩ Node v_i at the center of Δ_i

It suffices to show that a collision-free path $v_1 \rightsquigarrow v_k$ can be found.

- For $1 \leq i \leq k-1$, Δ_i borders Δ_{i+1} (at a vertical extension)

⇩

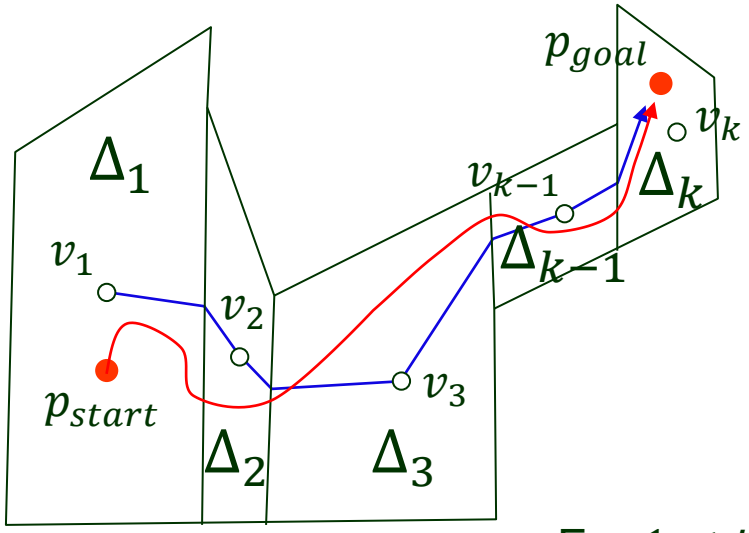
Collision free paths $v_i \rightsquigarrow v_{i+1}$ exist for $1 \leq i \leq k-1$.

⇩

A path $v_1 \rightsquigarrow v_k$ exists (and will be found by BFS).

Completeness

- Does the algorithm always find a collision-free path whenever one exists?



- Suppose there is a collision-free path $p_{start} \rightsquigarrow p_{goal}$.
- Δ_{start} and Δ_{goal} are trapezoids $\subseteq C_{free}$.
- Path $p_{start} \rightsquigarrow p_{goal}$ crosses trapezoids $\Delta_1 = \Delta_{start}, \Delta_2, \ldots, \Delta_k = \Delta_{goal}$.

⇩ Node v_i at the center of Δ_i

It suffices to show that a collision-free path $v_1 \rightsquigarrow v_k$ can be found.

- For $1 \leq i \leq k-1$, Δ_i borders Δ_{i+1} (at a vertical extension)

⇩

Collision free paths $v_i \rightsquigarrow v_{i+1}$ exist for $1 \leq i \leq k-1$.

⇩

A path $v_1 \rightsquigarrow v_k$ exists (and will be found by BFS). □

Planning Time

Environment complexity: total number n of vertices of the obstacles.

Planning Time

Environment complexity: total number n of vertices of the obstacles.

#trapezoids $= O(n)$

Planning Time

Environment complexity: total number n of vertices of the obstacles.

#trapezoids $= O(n)$

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.

Planning Time

Environment complexity: total number n of vertices of the obstacles.

#trapezoids $= O(n)$

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.

  Use the point location structure of the trapezoidal map.

Planning Time

Environment complexity: total number n of vertices of the obstacles.

#trapezoids $= O(n)$

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.

  Use the point location structure of the trapezoidal map. $O(\log n)$

Planning Time

Environment complexity: total number n of vertices of the obstacles.

#trapezoids $= O(n)$

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.

  Use the point location structure of the trapezoidal map. $O(\log n)$

- Construct the path $v_{start} \rightsquigarrow v_{goal}$ (via BFS).

Planning Time

Environment complexity: total number n of vertices of the obstacles.

#trapezoids $= O(n)$

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.

  Use the point location structure of the trapezoidal map. $O(\log n)$

- Construct the path $v_{start} \rightsquigarrow v_{goal}$ (via BFS). $O(n)$

Planning Time

Environment complexity: total number n of vertices of the obstacles.

#trapezoids $= O(n)$

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.

  Use the point location structure of the trapezoidal map. $O(\log n)$

- Construct the path $v_{start} \rightsquigarrow v_{goal}$ (via BFS). $O(n)$

- Report the path $p_{start} \rightsquigarrow p_{goal}$.

Planning Time

Environment complexity: total number n of vertices of the obstacles.

#trapezoids $= O(n)$

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.

  Use the point location structure of the trapezoidal map. $O(\log n)$

- Construct the path $v_{start} \rightsquigarrow v_{goal}$ (via BFS). $O(n)$

- Report the path $p_{start} \rightsquigarrow p_{goal}$. $O(n)$

Planning Time

Environment complexity: total number n of vertices of the obstacles.

#trapezoids $= O(n)$

- Determine the trapezoids Δ_{start} and Δ_{goal} containing p_{start} and p_{goal}, respectively.

  Use the point location structure of the trapezoidal map. $O(\log n)$

- Construct the path $v_{start} \rightsquigarrow v_{goal}$ (via BFS). $O(n)$

- Report the path $p_{start} \rightsquigarrow p_{goal}$. $O(n)$

Theorem A set S of polygonal obstacles with n edges can be preprocessed in $O(n \log n)$ expected time such that a collision-free path for a point robot between any start and goal positions can be computed in $O(n)$ time if it exists.