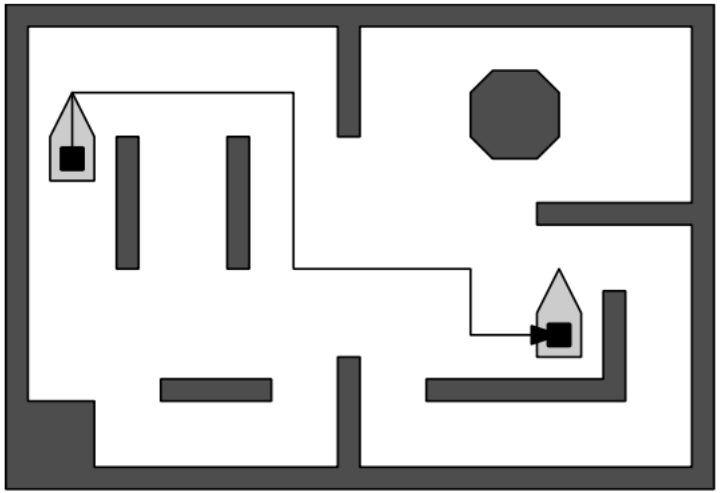
Technische Universität Braunschweig

Einführung in algorithmische Geometrie

Arne Schmidt

Kapitel 3 – Robot Motion Planning



Neues Kapitel

Wie kann ein Roboter durch ein Gebiet fahren?

Was passiert, wenn sich mehrere Roboter gleichzeitig bewegen?

Kapitel 3.1 – Work- und Configurationspace

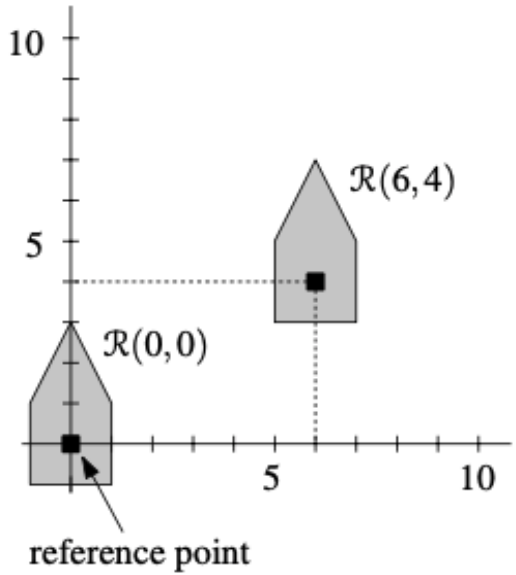
Workspace

Definition 3.1
Sei $\mathcal{R}$ ein Roboter, der sich einer zwei-dimensionalen Ebene (**workspace**) S bewegt. Dabei ist

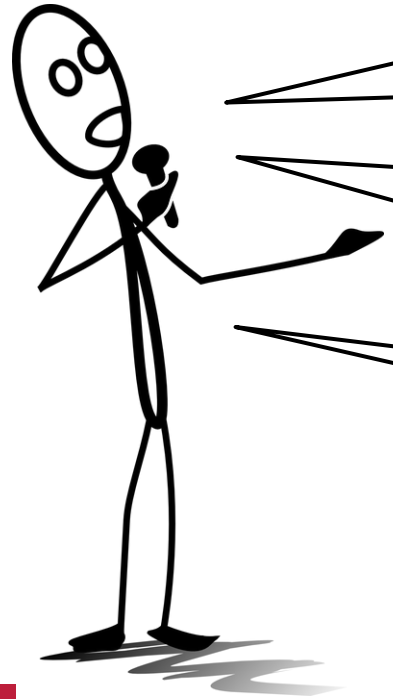
- $\mathcal{R}$ ein einfaches Polygon
- S eine Menge $\{P_1, \dots, P_t\}$ von polygonalen Hindernissen.

Eine **Konfiguration** (oder **Platzierung**) ist der Vektor, um den der Roboter vom Ursprung verschoben wird.
Der um (x, y) bewegte Roboter wird druch $\mathcal{R}(x, y)$ beschrieben.
$\mathcal{R}(0,0)$ beschreibt den Roboter selbst!

Der Vektor $(0, 0)$ wird auch **Referenzpunkt** von $\mathcal{R}$ genannt.
Die Konfiguration kann damit als Verschiebung des Referenzpunktes beschrieben werden.



Degree of Freedom



Kann der Roboter sich drehen, dann entspricht die Konfiguration $\mathcal{R}(x, y, \phi)$ dem Roboter mit Referenzpunkt an der Stelle (x, y) und Drehung um ϕ.

Für jede weitere Bewegungsmöglichkeit, kommt ein weiterer Parameter hinzu. Man spricht auch von den **Degrees of Freedom (DoF)** des Roboters.

Wir betrachten hier einen DoF von 2.

Frage: Wie viel DoF besitzt ein Roboter im $\mathbb{R}^3$, der sich beliebig drehen kann?

Dimensionen

Mit jedem Freiheitsgrad erhöht sich der Configuration Space um eine Dimension

Das wird schnell unübersichtlich.

Was ist für ein zwei-dimensionalen Configuration Space erlaubt?

Configuration Space

Definition 3.2
Der Parameterraum eines Roboters $\mathcal{R}$ wird auch **Konfigurationsraum** (**configuration space**) $\mathcal{C}(\mathcal{R})$ genannt.

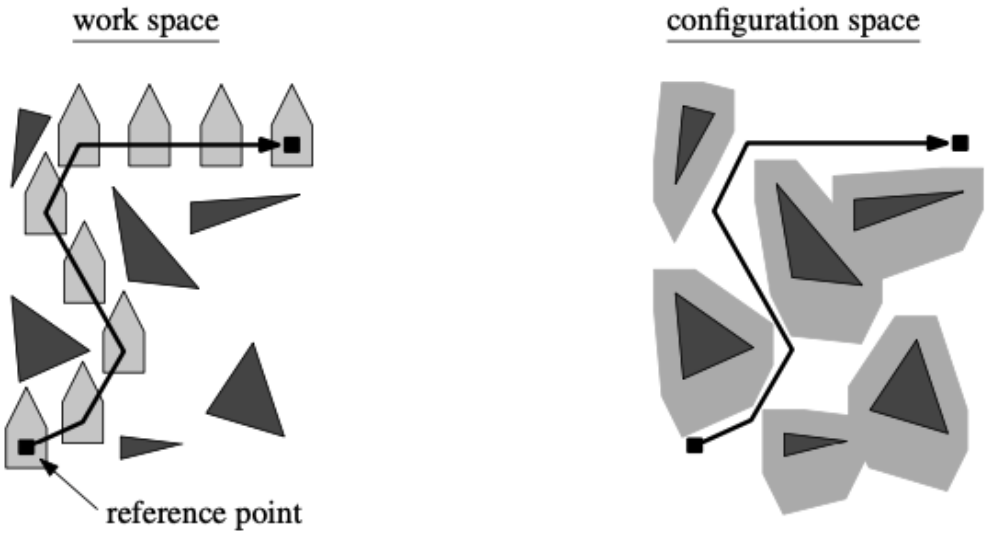
Ein Punkt p im Konfigurationsraum entspricht einer Konfiguration $\mathcal{R}(p)$ im Workspace.

Eine Konfiguration p heißt **verboten** (forbidden), wenn $\mathcal{R}(p)$ sich mit Hindernissen überlappt. Andernfalls heißt die Konfiguration **erlaubt** (free).

Der Konfigurationsraum $\mathcal{C}_{forb}(\mathcal{R})$ bestehend aus verbotenen Konfigurationen heißt **verbotener Konfigurationsraum** (forbidden space).
Analog ist $\mathcal{C}_{free}(\mathcal{R})$ bestehend aus erlaubten Konfigurationen der **freie Konfigurationsraum** (free space).

Bewegungen

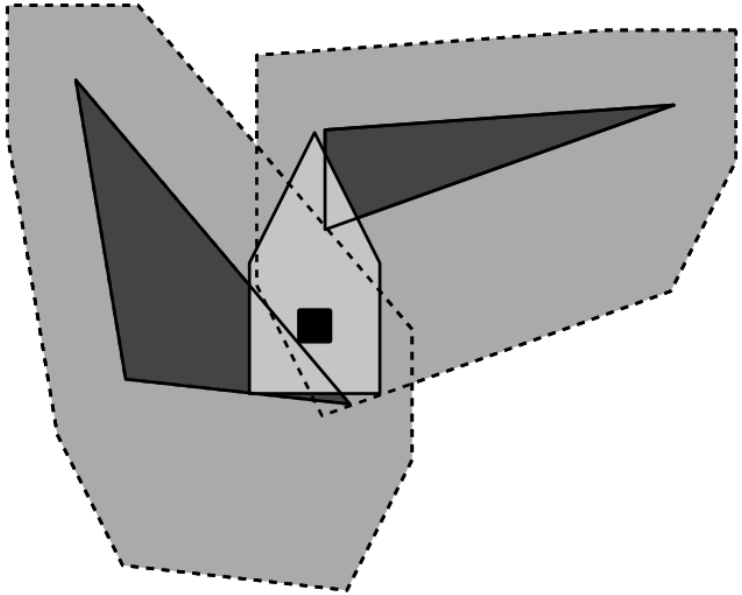
Lemma 3.3
Ein Pfad im Workspace entspricht einer Kurve im Konfigurationsraum.
Ein kollisionsfreier Pfad entspricht einer Kurve im freien Konfigurationsraum.



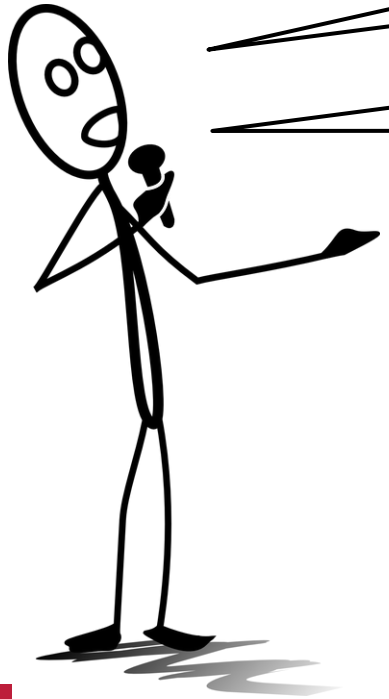
Hindernisse

Definition 3.4
Der Hinderniss-Konfigurationsraum (configuration space obstacle, $\mathcal{C}$-obstacle) eines Hindernisses P ist die Menge an Punkten p, sodass $\mathcal{R}(p) \cap P \neq \emptyset$.

Wir betrachten P dabei als **offene Menge** (open set), d.h. $\mathcal{R}(p)$ und P dürfen sich berühren.



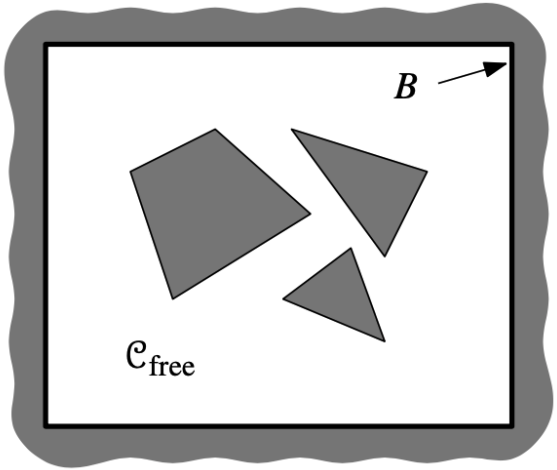
Vereinfachungen



Für den Start schauen wir uns Punktroboter an.

Zusätzlich benutzen wir ein eine Bounding Box B, in der wir uns bewegen und alle Hindernisse enthält.

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



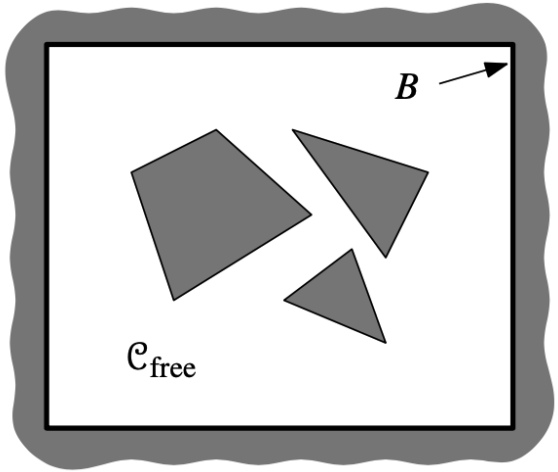
Datenstrukturen

Wie speichern wir diese Informationen?

Wie können wir schnell Pfade bestimmen?

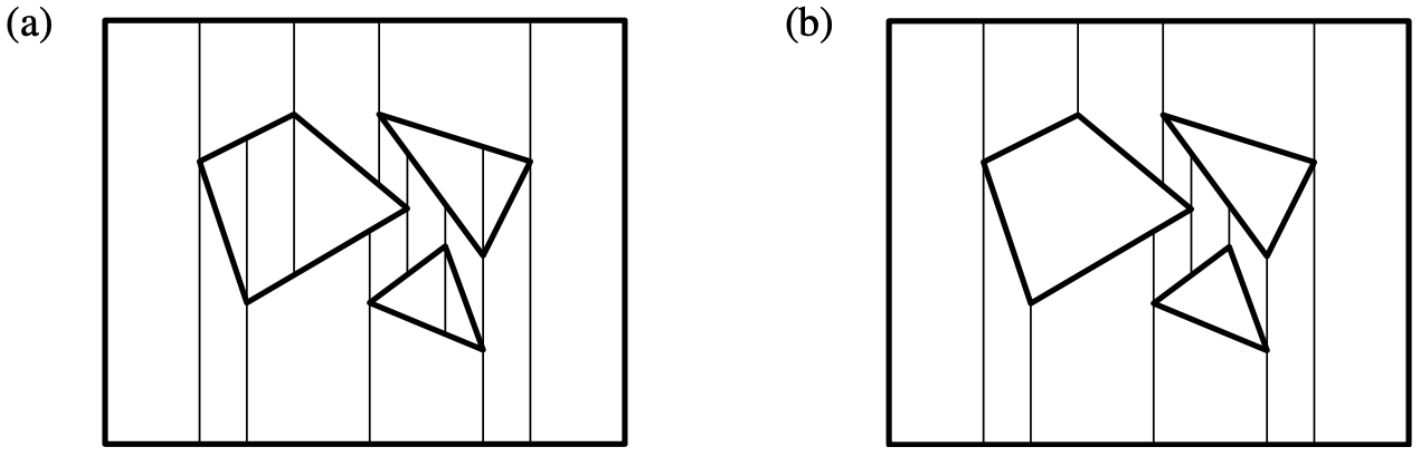
Kapitel 3.2 – Roadmap

Trapezoidal Maps



Können wir nicht eine Art Trapezoidal Map nutzen?

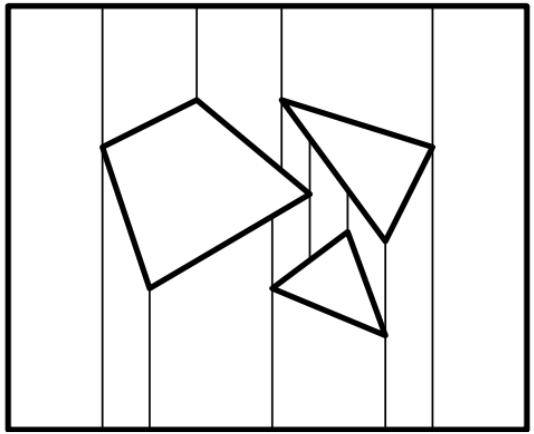
Trapezoidal Maps



Die Bereiche im Inneren von Hindernissen müssen wir nicht betrachten.

Lemma 3.5
Eine Trapoezoidal Map des freien Konfigurationsraums für einen Punktroboter und disjunkten polygonalen Hindernissen bestehend aus n Segmenten kann in erwarteter Zeit $O(n \log n)$ berechnet werdenl

Finden valider Pfade



Wie finden wir nun Pfad zwischen zwei Punkten im freien Raum?

Zunächst:
Bestimme die Flächen in der sich Start und Ziel befinden.

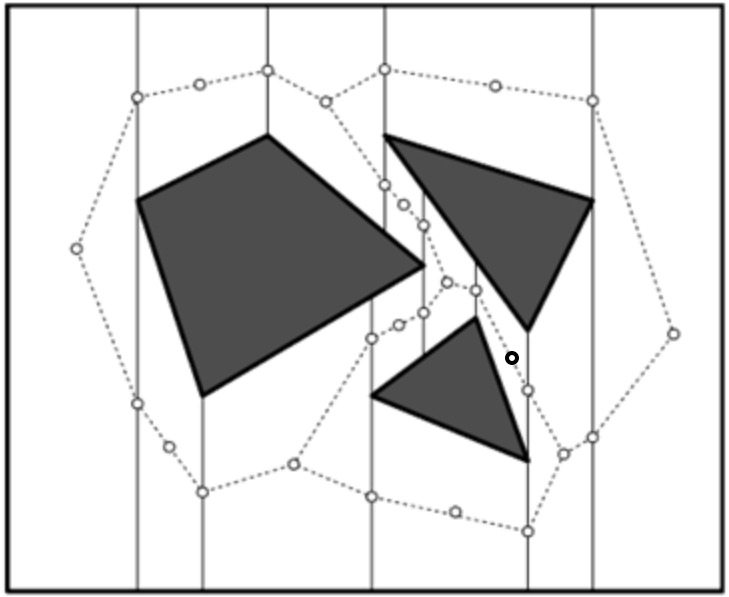
Liegen sie in der gleichen Fläche, ist es einfache eine gerade Linie!

Für unterschiedliche Flächen wird es interessant.
Wir müssen wissen durch welche Trapeze wir traversieren müssen.

Road Map

Definition 3.6
Die Road Map $\mathcal{G}_{road}$ ist ein Graph, welcher im freien Raum eingebettet wird.
Definiert wird der Graph wie folgt:

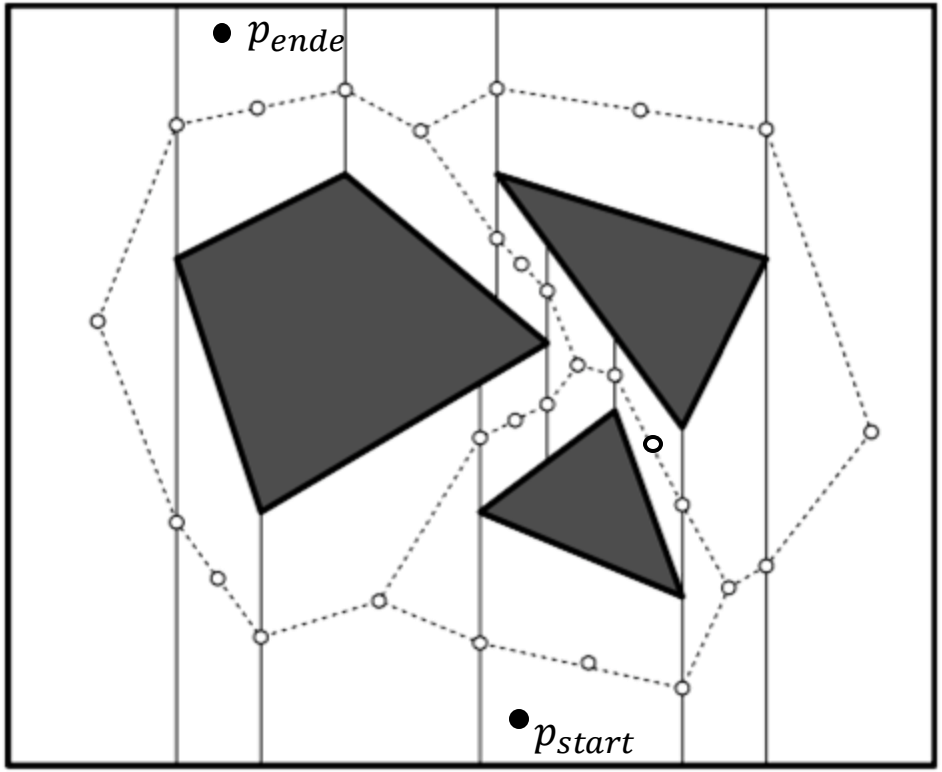
- Für jedes Trapez T der Traperzoidal Map existiert ein Knoten in der Mitte von T.
- Für jedes Paar benachbarter Trapeze T, T' existiert ein Knoten in der Mitte der gemeinsamen vertikalen Linie von T und T'.
- Zwei Knoten T, ℓ werden verbunden, wenn T ein Trapezknoten ist und ℓ eine vertikale Linie von T ist.



Lemma 3.7
Die Road Map kann aus der Trapezoidal Map in Zeit $O(n)$ konstruiert werden.

Beispiel

Schritt 1:
Bestimme Δ_{start} und Δ_{ende}, sowie v_{start} und v_{ende} der Road Map

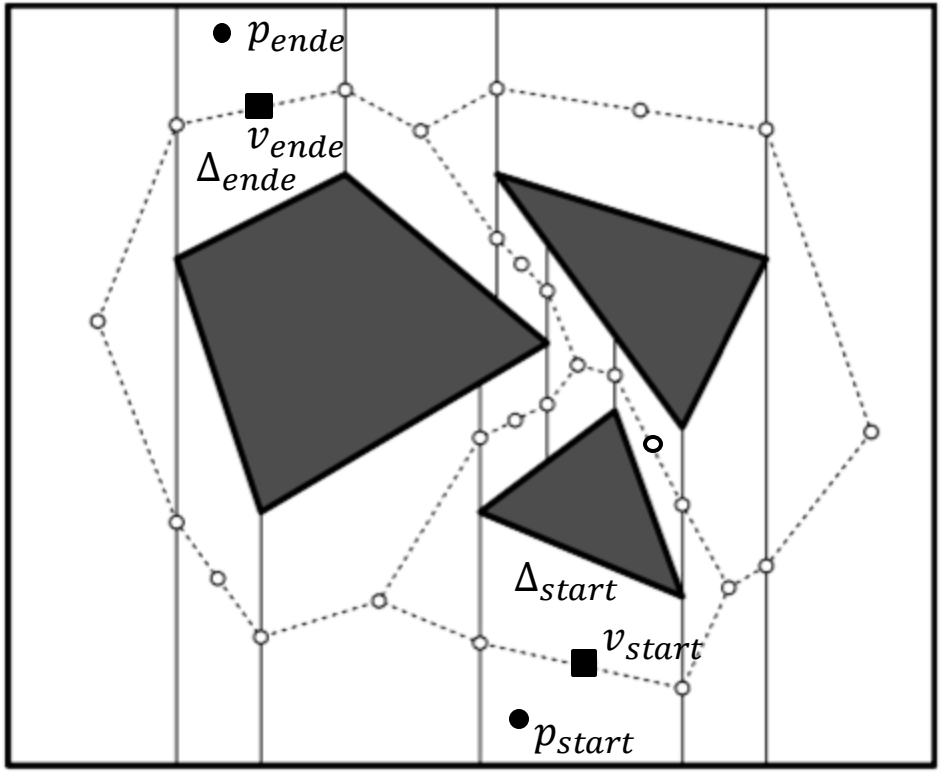


Beispiel

Schritt 1:
Bestimme Δ_{start} und Δ_{ende}, sowie v_{start} und v_{ende} der Road Map

Schritt 2:
Bestimme Pfad von v_{start} nach v_{ende} über die Road Map.

Das funktioniert bspw. mit Breitensuche



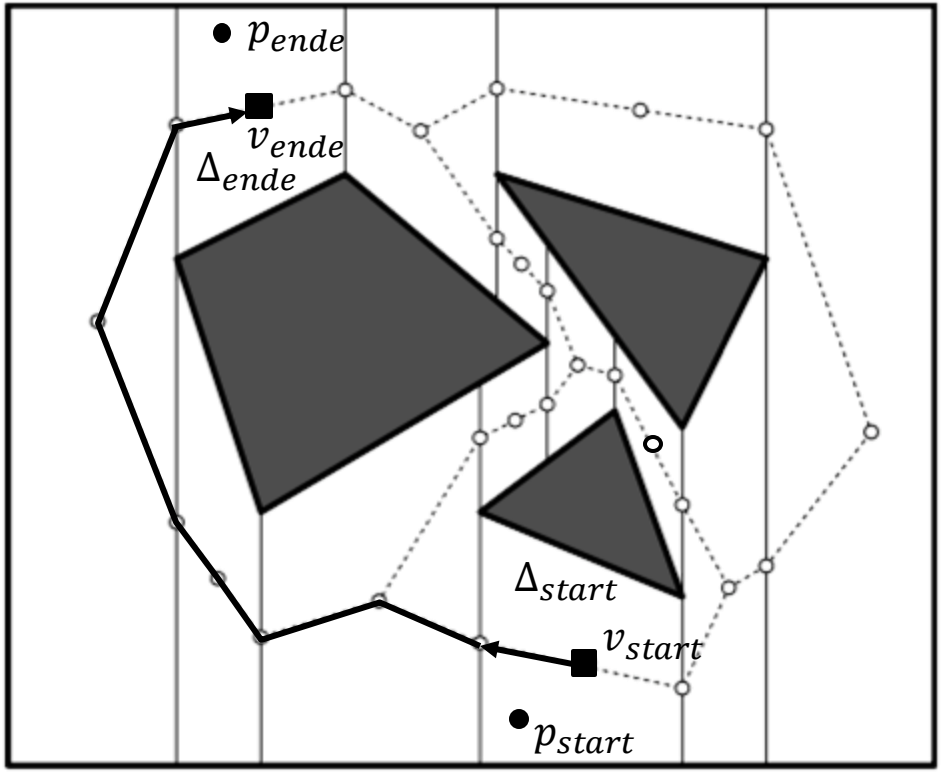
Beispiel

Schritt 1:
Bestimme Δ_{start} und Δ_{ende}, sowie v_{start} und v_{ende} der Road Map

Schritt 2:
Bestimme Pfad von v_{start} nach v_{ende} über die Road Map.

Das funktioniert bspw. mit Breitensuche

Schritt 3:
Verbinde p_{start} mit v_{start} sowie v_{ende} mit p_{ende}



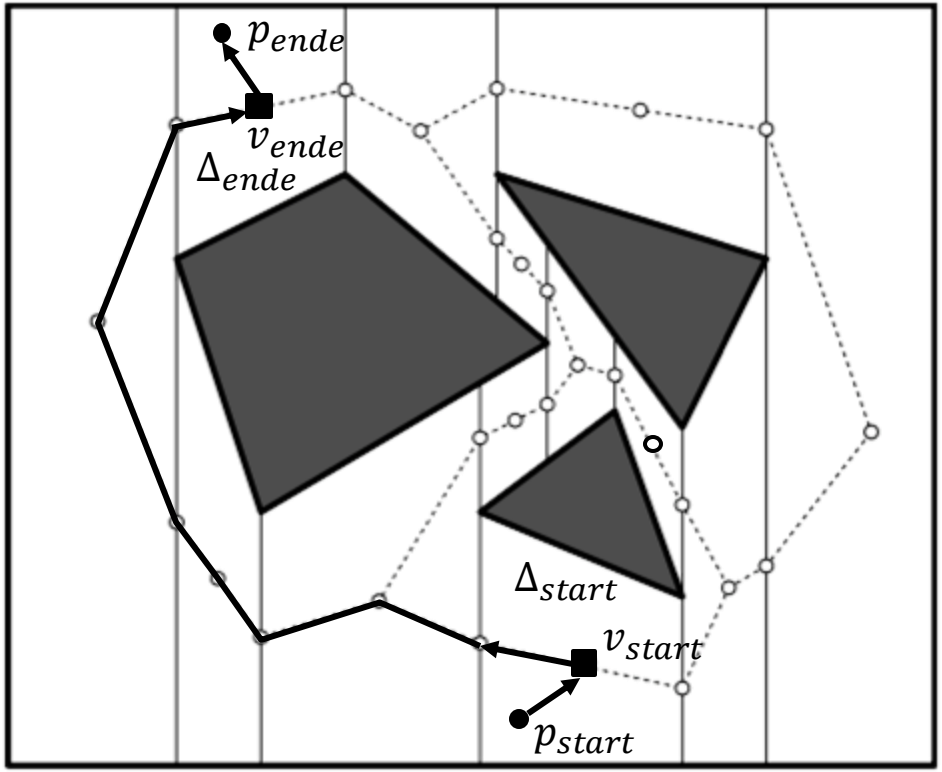
Beispiel

Schritt 1:
Bestimme Δ_{start} und Δ_{ende}, sowie v_{start} und v_{ende} der Road Map

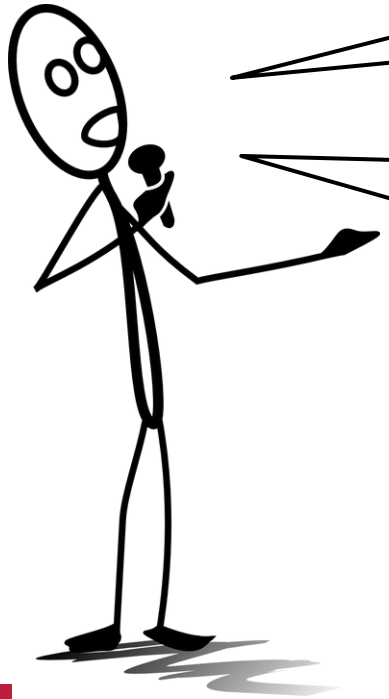
Schritt 2:
Bestimme Pfad von v_{start} nach v_{ende} über die Road Map.

Das funktioniert bspw. mit Breitensuche

Schritt 3:
Verbinde p_{start} mit v_{start} sowie v_{ende} mit p_{ende}



Achtung



Dieses Verfahren über Breitensuche minimiert nur die Anzahl an „Hops“ von Trapez zu Trapez.

Den geometrisch kürzesten Weg zu bestimmen ist aufwändiger und oft auch nicht interessant: Wir müssten ganz nah an die Hindernisse heran.

Algorithmus

Algorithmus 3.7 (ComputePath)
Eingabe: Trapezoidal Map des freien Raumes $\mathcal{T}(\mathcal{C}_{free})$, Road Map $\mathcal{G}_{road}$, Start- p_{start} und Zielpositionen p_{ende}
Ausgabe: Ein Pfad von p_{start} zu p_{ende}, falls existent. Falls nicht existent, wird dies ausgegeben.

1. Finde Δ_{start} bzw. Δ_{ende}, die p_{start} bzw p_{ende} beinhalten.
2. Falls Δ_{start} oder Δ_{ende} nicht existent, gib aus, dass sich Start oder Ziel im verbotenen Raum befinden.
3. Sei v_{start} bzw. v_{ende} der Knoten von $\mathcal{G}_{road}$ zu Δ_{start} bzw. Δ_{ende}
4. Berechne Pfad P von v_{start} nach v_{ende} in $\mathcal{G}_{road}$.
5. Falls kein Pfad existiert, gib an, dass v_{ende} nicht erreicht werden kann.
6. Andernfalls: Gib folgenden Pfad aus: $(v_{start} P v_{ende})$

Korrektheit und Laufzeit

Lemma 3.8
Algorithmus 3.7 berechnet einen kollisionsfreien Pfad, falls dieser existiert, in Zeit $O(n)$.

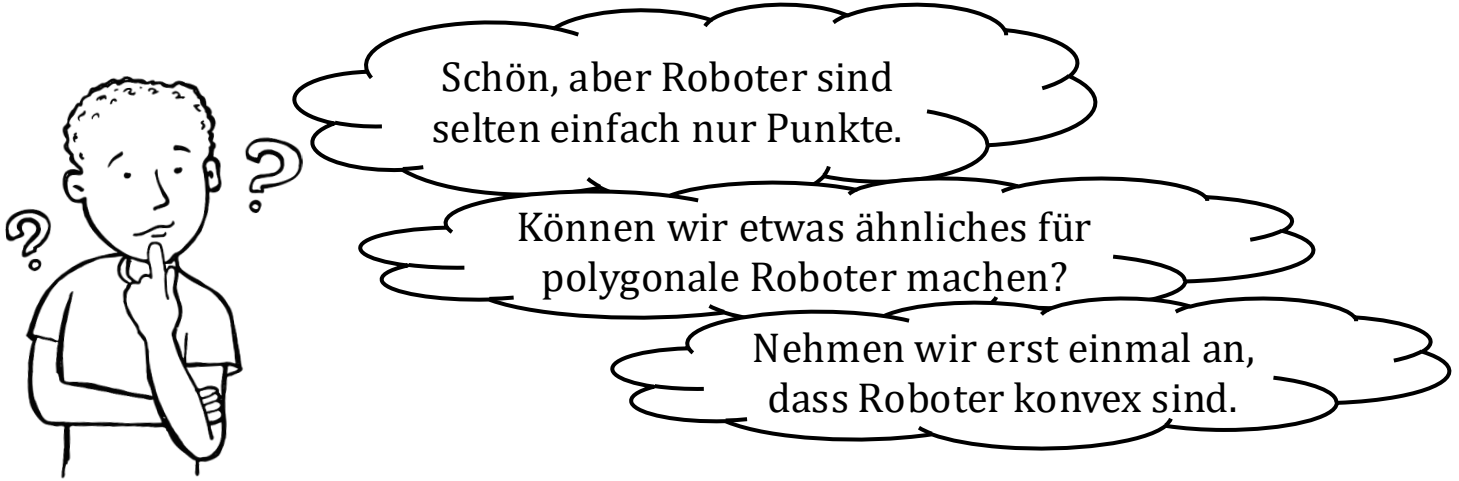
Korrektheit:
Ein existierender Pfad bewegt sich nur von Trapezmitte zu einer vertikalen Erweiterung und anderherum.
Diese Segmente liegen innerhalb eines Trapez und überlappen sich nicht mit Hindernissen!
→ Vollständiger Pfad ist kollisionsfrei.

Zeit:
Suche nach Start- und Endtrapez dauert $O(\log n)$ Zeit.
Pfadberechnung mit Breitensuche benötigt $O(n)$ Zeit.
→ Gesamter Algorithmus benötigt $O(n)$ Zeit.

Bewegungsplan

Theorem 3.9
Sei $\mathcal{R}$ ein Punktroboter, der sich unter einer Menge S von polygonalen Hindernissen bewegt.

Wir können S in erwarteter Zeit $O(n \log n)$ so vorbearbeiten, sodass ein kollisionsfreier Pfad zwischen zwei beliebigen Position für $\mathcal{R}$ in Zeit $O(n)$ bestimmt werden kann.



Kapitel 3.3 – Minkowski Summen

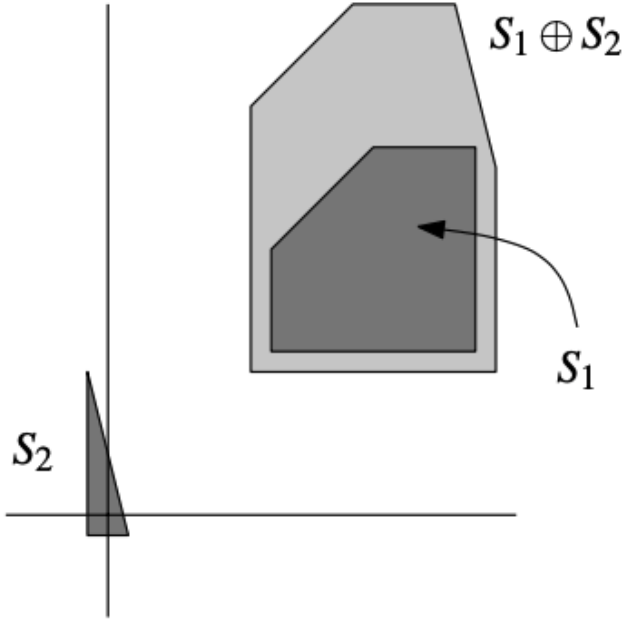
Minkowski Summe

Definition 3.10
Die **Minkowski Summe** von zwei mengen $S_1, S_2 \subset \mathbb{R}^2$ ist definiert als

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

wobei $p + q$ eine Vektoraddition ist.

Die Hindernisse und den Roboter können wir als solche planaren Mengen beschreiben!



Minkowski Summe

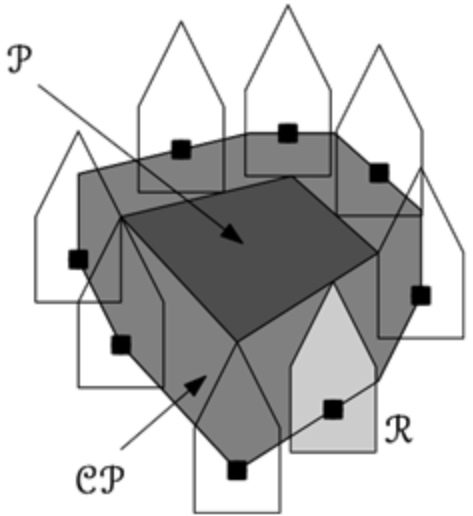
Theorem 3.11
Sei $\mathcal{R}$ ein planarer, sich verschiebener Roboter und sei P ein polygonales Hinderniss. Dann ist $\mathcal{C}$-obstacle von P definiert durch $P \oplus \left(-\mathcal{R}(0,0)\right)$.

Zunächst, betrachte $q = \left(q_x, q_y\right) \in \mathcal{R}(x,y) \cap P \neq \emptyset$.

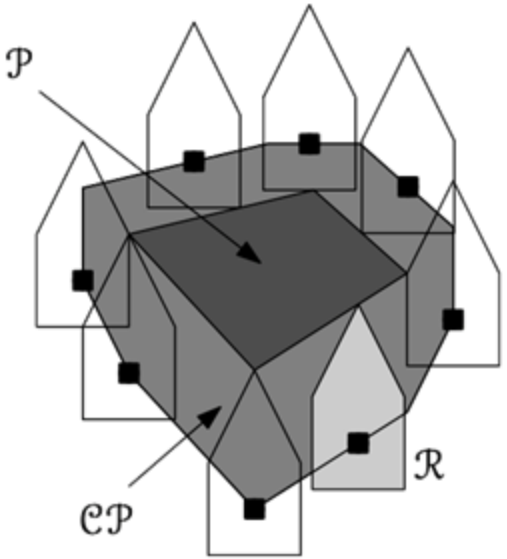
- Da $q \in \mathcal{R}(x,y)$ ist $\left(q_x - x, q_y - y\right) \in \mathcal{R}(0,0)$.
- Also $\left(-q_x + x, -q_y + y\right) \in -\mathcal{R}(0,0)$
- Mit $q \in P$ folgt $(x,y) \in P \oplus \left(-\mathcal{R}(0,0)\right)$

Andersherum, betrachte $(x,y) \in P \oplus \left(-\mathcal{R}(0,0)\right)$.

- Dann existieren $\left(r_x, r_y\right) \in \mathcal{R}(0,0)$ und $\left(p_x, p_y\right) \in P$ mit $(x,y) = \left(p_x - r_x, p_y - r_y\right)$, bzw. $\left(p_x, p_y\right) = \left(r_x + x, r_y + y\right)$.
- Also ist $\mathcal{R}(x,y) \cap P \neq \emptyset$.



Minkowski Summe



Beobachtungen, wenn $\mathcal{R}$ und P konvex sind:

- Entweder laufen wir entlang einer Kante von $\mathcal{R}$ oder P.
- Das machen wir nur einmal!
- Wir können uns auf die Ränder von $\mathcal{R}$ und P konzentrieren.

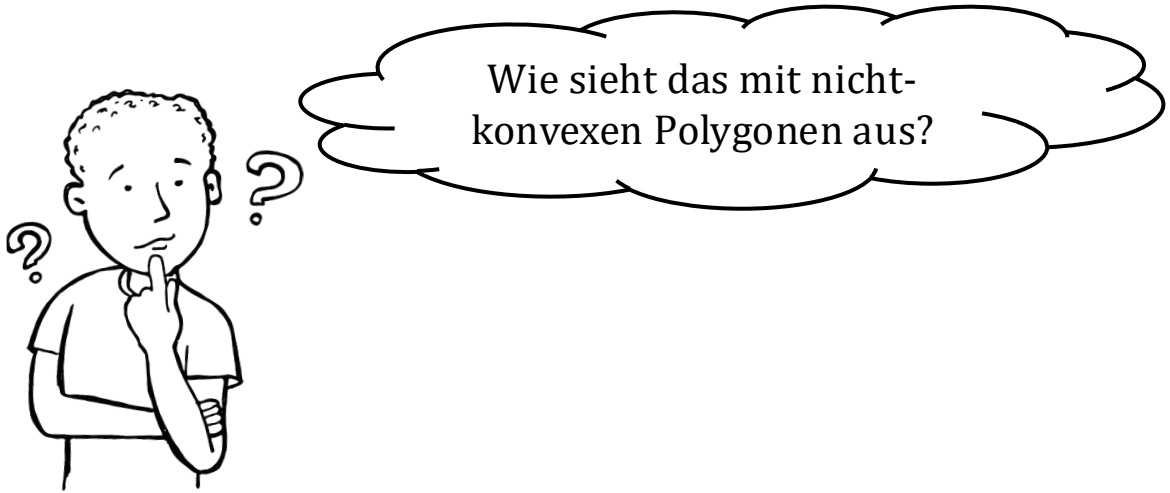
Algorithmus für die Minkowski Summe

Algorithmus 3.12 (MinkowskiSum)
Eingabe: Konvexe Polygone $P, \mathcal{R}$ mit Knoten $v_1, \dots, v_n$ bzw. $w_1, \dots, w_m$ (geordnet entgegen Uhrzeigersinn)
Ausgabe: Die Minkowski Summe $P \oplus \mathcal{R}$.

1. $i = 1, j = 1$
2. *Wiederhole*
   a) Füge $v_i + w_j$ als Knoten zu $P \oplus \mathcal{R}$ hinzu.
   b) Falls Winkel$(v_i v_{i+1})$ < Winkel$(w_j w_{j+1})$ dann $i = i+1$
   c) Ansonsten, falls Winkel$(v_i v_{i+1})$ > Winkel$(w_j w_{j+1})$ dann $j = j+1$
   d) Ansonsten $i = i+1$ und $j = j+1$
3. Bis $i = n+1$ und $j = n+1$

Laufzeit

Lemma 3.13
Algorithmus 3.12 berechnet in Zeit $O(n + m)$ die Minkowski Summe von zwei konvexen Polygonen mit n bzw. m Knoten.



Minkowski Summe beliebiger Polygone

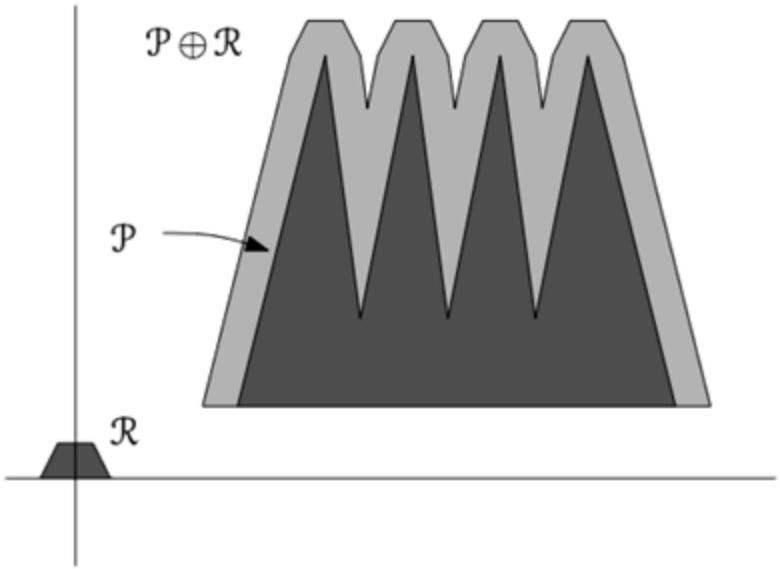
Theorem 3.15
Die Minkowski Summe von zwei Polygonen P_1, P_2 mit n bzw. m Knoten benötigt Zeit

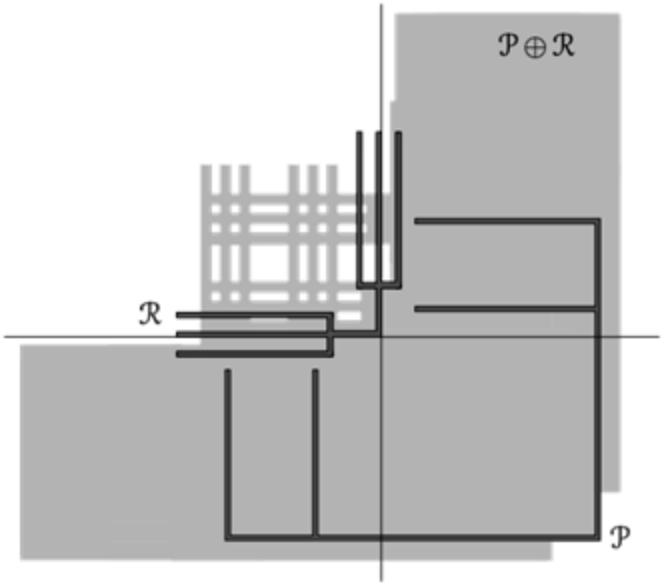
- $O(n+m)$, wenn beide Polygone konvex sind.
- $O(nm)$, wenn eines der beiden Polygone konvex ist.
- $O(n^2m^2)$, wenn beide Polygone nicht konvex sind.

Die Minkowski Summe kann die angegebene Komplexität erreichen.

Beweisskizze:
Teile nicht-konvexe Polygone in konvexe Teile, berechne Minkowski Summe aller konvexen Paare beider Polygone und füge Resultate zusammen.

Worst-Cases


$\mathcal{P} \oplus \mathcal{R}$
$\mathcal{P}$
$\mathcal{R}$


$\mathcal{P} \oplus \mathcal{R}$
$\mathcal{R}$
$\mathcal{P}$

Freespace

Okay, wir wissen nun...

Wie können wir einen Punktroboter bewegen.

Wir können auf Punktroboter reduzieren.

Wir müssen dazu nur den Freespace berechnen!

Kapitel 3.4 – Motion Planning mit Verschieben

Robot Motion Planning mit Verschiebung

Theorem 3.16
Sei $\mathcal{R}$ ein konvexer Roboter konstanter Komplexität, der sich unter einer Menge S von polygonalen Hindernissen mit insgesamt n Kanten bewegt.

a) Die Komplexität des freien Raumes $\mathcal{C}(\mathcal{R}, S)$ ist dann $O(n)$.
b) Der freie Raum kann in Zeit $O(n \log^2 n)$ Zeit berechnet werden.

Beweisskizze zu a)
Zeige, dass die Vereinigung der Minkowski-Summen höchstens um einen konstanten Faktor wachsen kann. Bestimme dazu wie oft die Ränder zweier Minkowski-Summen sich schneiden können.

Robot Motion Planning mit Verschiebung

Theorem 3.16
Sei $\mathcal{R}$ ein konvexer Roboter konstanter Komplexität, der sich unter einer Menge S von polygonalen Hindernissen mit insgesamt n Kanten bewegt.

a) Die Komplexität des freien Raumes $\mathcal{C}(\mathcal{R}, S)$ ist dann $O(n)$.
b) Der freie Raum kann in Zeit $O(n \log^2 n)$ Zeit berechnet werden.

Beweisskizze zu b)
Es muss die Vereinigung aller Minkowski-Summen der Obstacles berechnet werden.
Nutze dazu Divide-and-Conquer, sowie das Berechnen von Overlays.

Algorithmus: Forbidden-Space

Algorithmus 3.17 (FORBIDDEN-SPACE)
Eingabe: Eine Menge von Hindernissen $\mathcal{P}_1, \dots, \mathcal{P}_n$.
Ausgabe: $\mathcal{C}_{forb} = \bigcup_{i=1}^{n} \mathcal{CP}_i = \bigcup_{i=1}^{n} \mathcal{P}_i \oplus -\mathcal{R}(0,0)$

1. If $n = 1$, return $\mathcal{CP}_1$
2. Else
   a) $\mathcal{C}_{forb}^1 = \text{FORBIDDEN-SPACE}\left(\mathcal{P}_1, \dots, \mathcal{P}_{\left\lceil \frac{n}{2} \right\rceil}\right)$
   b) $\mathcal{C}_{forb}^2 = \text{FORBIDDEN-SPACE}\left(\mathcal{P}_{\left\lceil \frac{n}{2} \right\rceil + 1}, \dots, \mathcal{P}_n\right)$
   c) $\mathcal{C}_{forb} = \mathcal{C}_{forb}^1 \cup \mathcal{C}_{forb}^2$
   d) *return* $\mathcal{C}_{forb}$

Laufzeit: $T(n) = 2\,T\left(\frac{n}{2}\right) + O(n \log n)$, also $T(n) \in O(n \log^2 n)$

Nächstes Mal

Mit einem (konvexen) Roboter ohne Drehung ist das Problem gut lösbar.

Was passiert mit mehreren Robotern?

Welche Frage-stellungen gibt es?