

GLO-4001/7021

INTRODUCTION À LA ROBOTIQUE

MOBILE

Cartes +
Planification

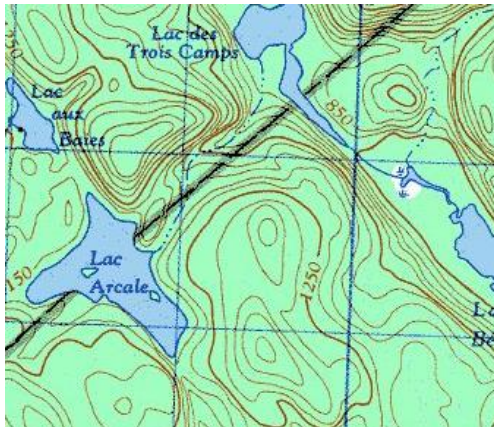
Cartes

Représentation du monde : carte

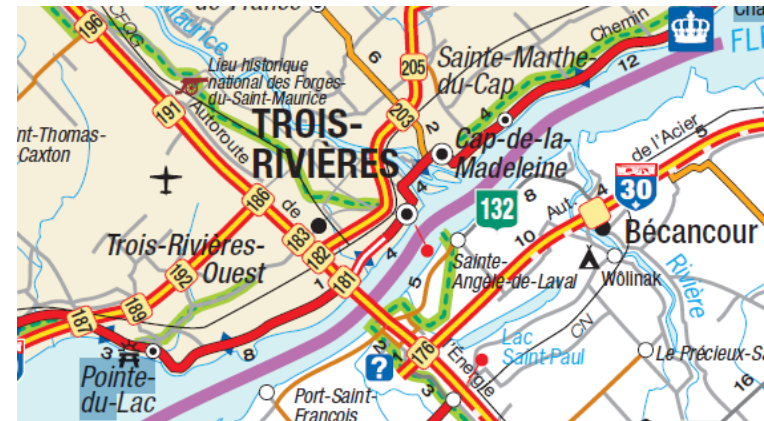
- Notre robot accumule de l'information sur le monde : **connaissance**
- Une carte permettra de ^(planifier) raisonner sur le monde
- Plusieurs types possibles de représentation de cette **connaissance**
- Problème fondamental en intelligence artificielle
 - représentation des connaissances
- Compromis entre taille stockage, facilité d'usage, objectifs à accomplir

Représentation universelle

- Pas de représentation universelle possible
- Argument pour les architectures réactives :
 - « le monde est son meilleur modèle »
 - aucune carte utilisée
 - exemple : algorithmes *bug* (~~à venir~~)
- Type de la carte dépendra de la tâche/capteurs



carte topographique

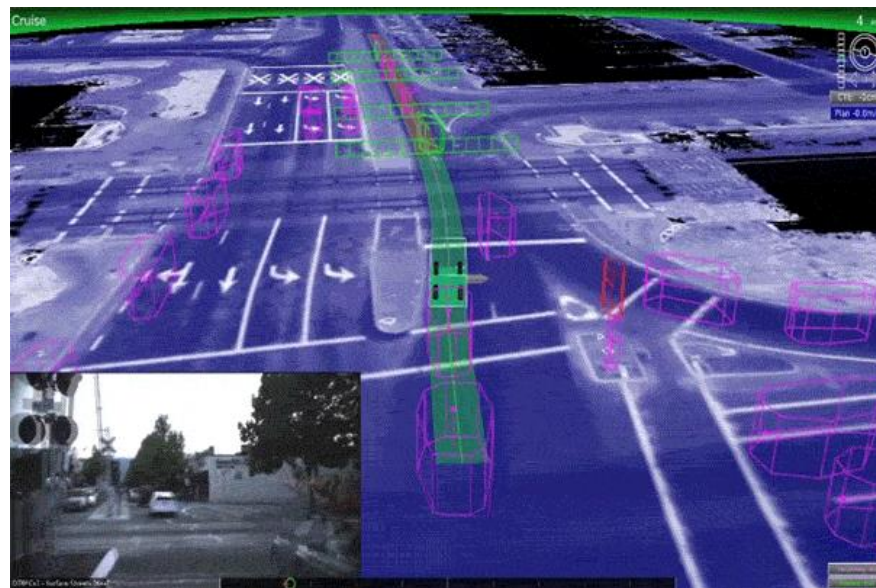


carte routière

Cartes pour Google car

Utilise plusieurs cartes :

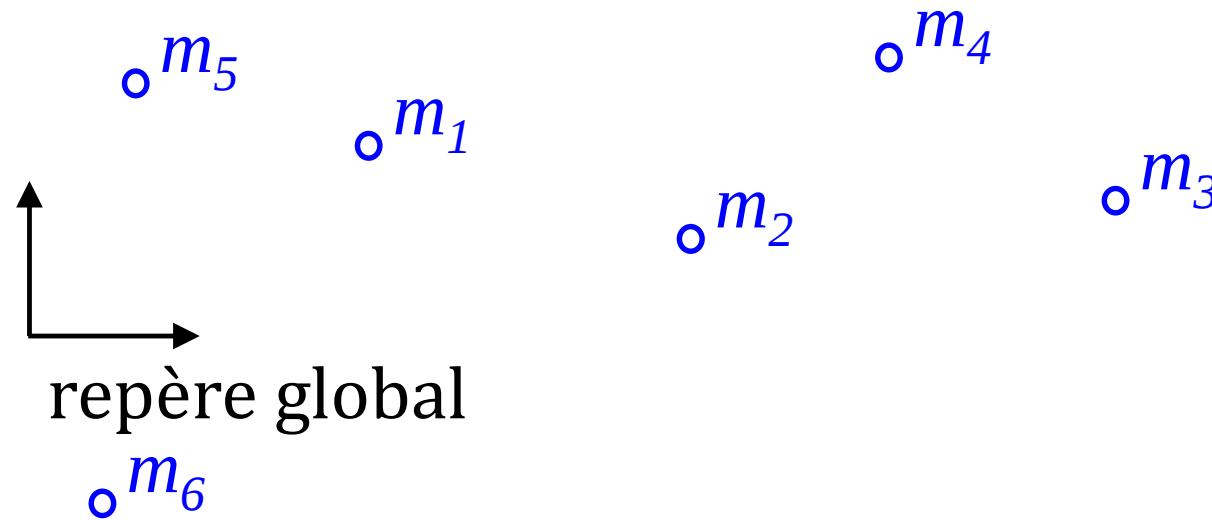
- 1 carte de la **réflectance laser** des surface (*texture model*) pour localisation par filtre à particule via corrélation d'image infrarouge (précision 11 *cm*)
- 1 carte **d'élévation** (altitude) pour la forme du monde (sert à identifier les obstacles dynamiques qui « ressortent » du terrain)
- 1 carte **d'information tactique** (intersection, feux de circulation, voies sur l'autoroute, passage piéton, etc..)



Types de cartes en robotique mobile

- **Métrique**

- décrire un environnement avec un système de coordonnées absolu
- notion de distance calculable entre tous les points



point de repère $m_i = [x \ y \ signature]^T$

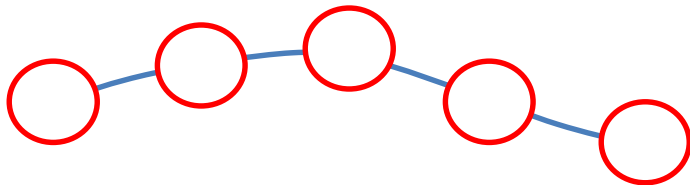
Types de cartes en robotique mobile

- **Métrique**

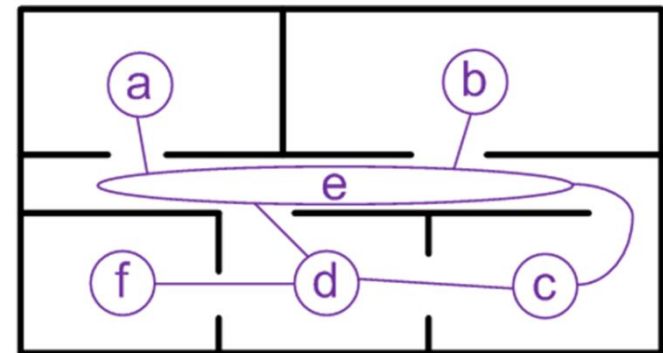
- décrire un environnement avec un système de coordonnées absolu
- notion de distance calculable entre tous les points

- **Topologique**

- graphe (nœud=endroit, arêtes=connexion)
- évacue beaucoup d'information (~~distance~~)
- représentation des relations locales entre les endroits



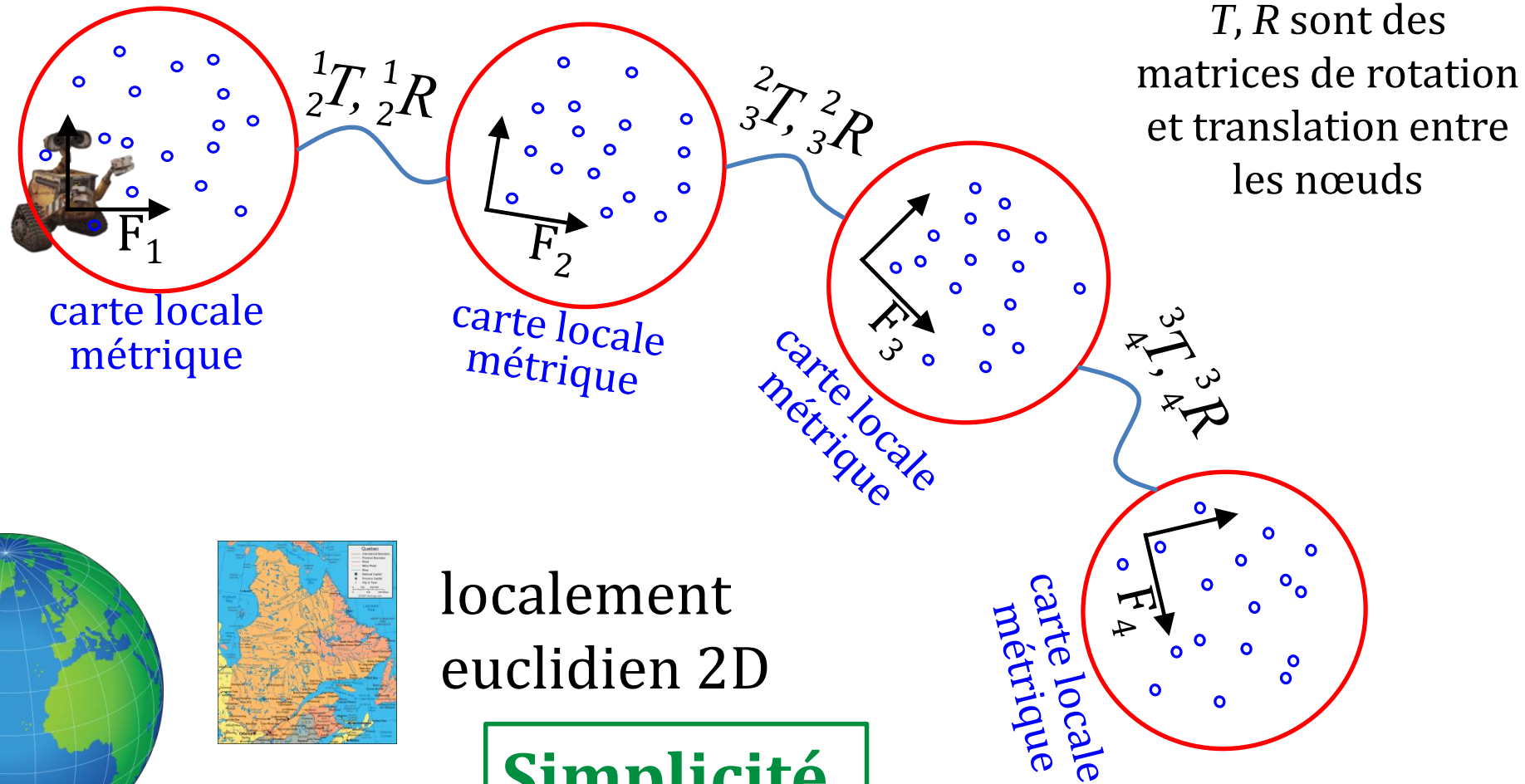
ou



Types de cartes en robotique mobile

- **Métrique**
 - décrire un environnement avec un système de coordonnées absolu
 - notion de distance calculable entre tous les points
- **Topologique**
 - graphe (nœud=endroit, arêtes=connexion)
 - évacue beaucoup d'information (~~distance~~)
 - représentation des relations locales entre les endroits
- **Topométrique**
 - graphe, carte métrique à chaque nœud
 - arêtes peuvent contenir une notion de distance + orientation relatives

Exemple carte topométrique



(manifold
dans 3D)



localement
euclidien 2D

Simplicité

Local Metrical and Global Topological Maps in the Hybrid Spatial Semantic Hierarchy, B. Kuipers, et al., *ICRA* 2004. 11

Caractéristiques des cartes (2)

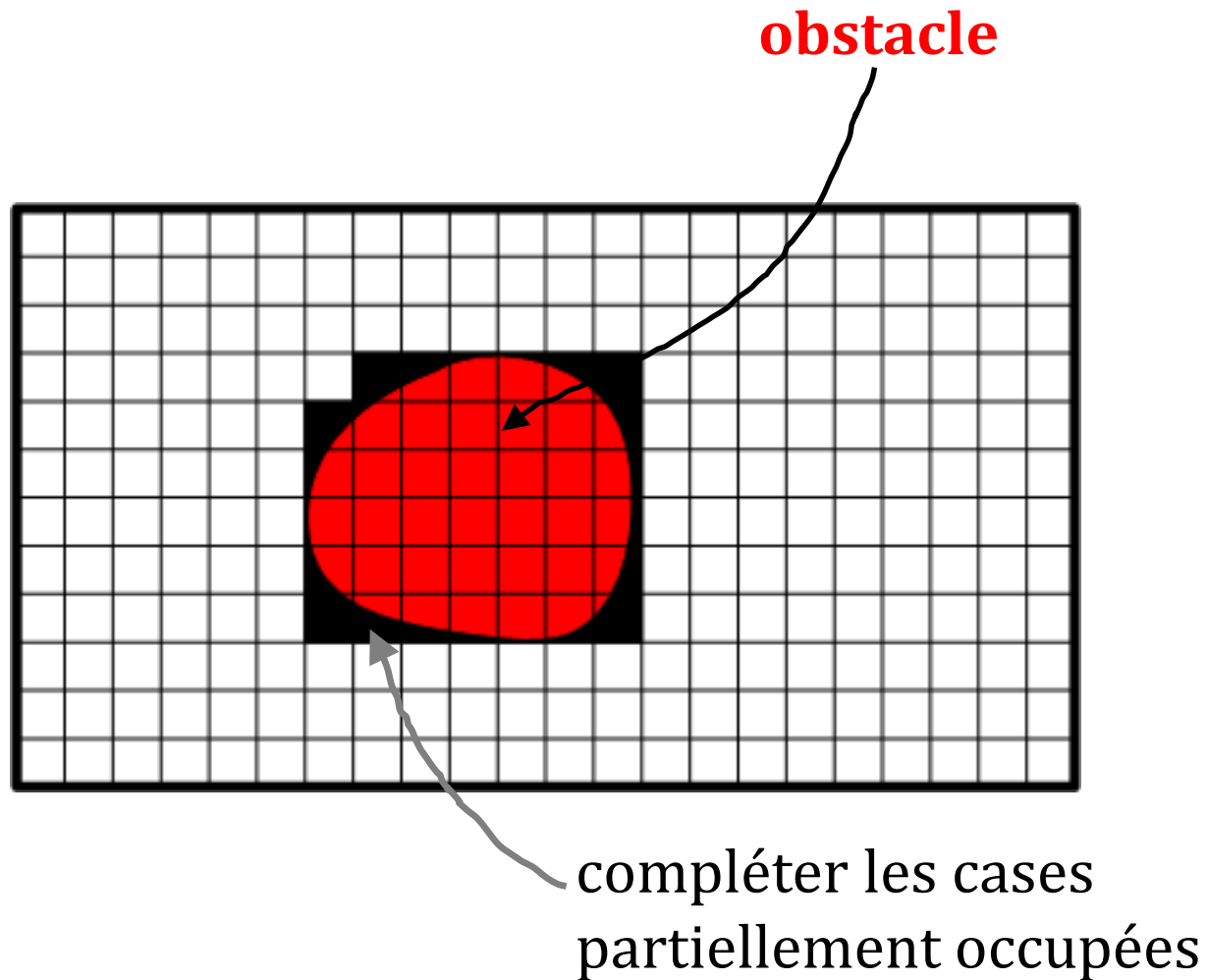
- **Continue**

- positions sont des réels
 - une lampe à $(-1.45534, +5.899485)$
 - impact laser sur mur à $(+2.323, +1.234)$

- **Discrète**

- discrétiser le monde en case de taille fixe
- valeur binaire : 0= libre 1= obstacle
- valeur continue
 - $[0, 1]$ (probabilité obstacle)
 - $[0, \infty]$ (cote obstacle)
 - $[-\infty, \infty]$ $\log(\text{cote})$
- *e.g.* grilles d'occupation (*occupancy grids*)

Cartes : grillage uniforme

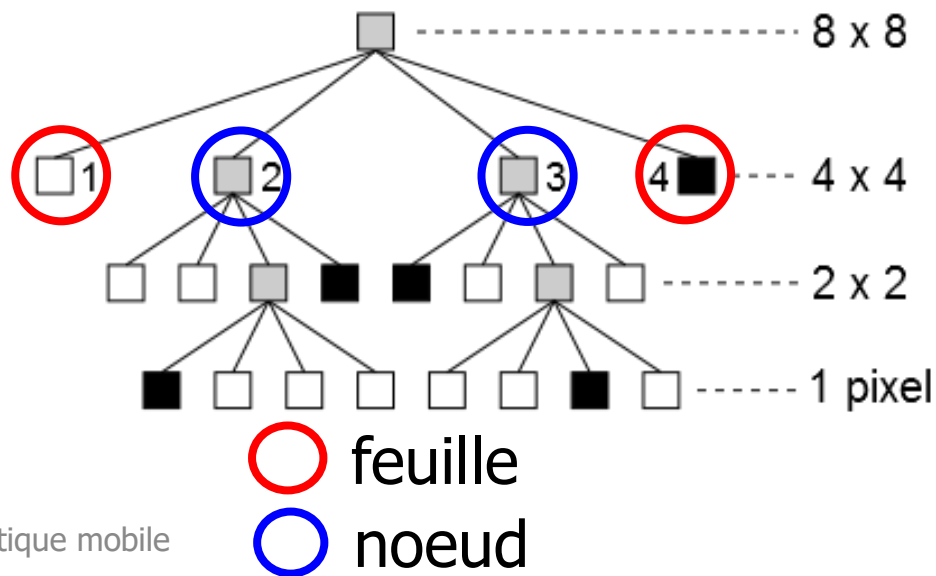


Cartes : grille uniforme

- Décide de l'intervalle échantillonnage, *e.g.* 1 *cm*
 - 2D : pièce de 10 *m* x 10 *m*
 - $1000 \times 1000 = 1\,000\,000$ « pixels »
 - 3D : pièce de 10 *m* x 10 *m* x 3 *m*
 - $1000 \times 1000 \times 300 = 300\,000\,000$ « voxels »
- (-) Espace de stockage
- (-) Erreur de discrétisation
- (+) Temps d'accès rapide

Cartes : échantillonnage Quadtree

- Structure de données en arbre
- Compression de l'information
- Par contre, structure de l'arbre change beaucoup pour petits changements d'image



Quadtree

feuille

nœud

feuille



vide



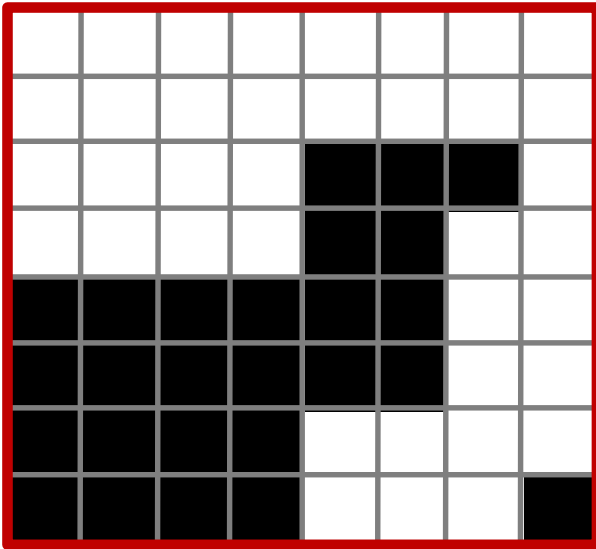
en partie plein



plein



..... 8 x 8

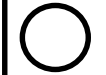


Quadtree

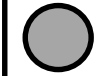
feuille

nœud

feuille



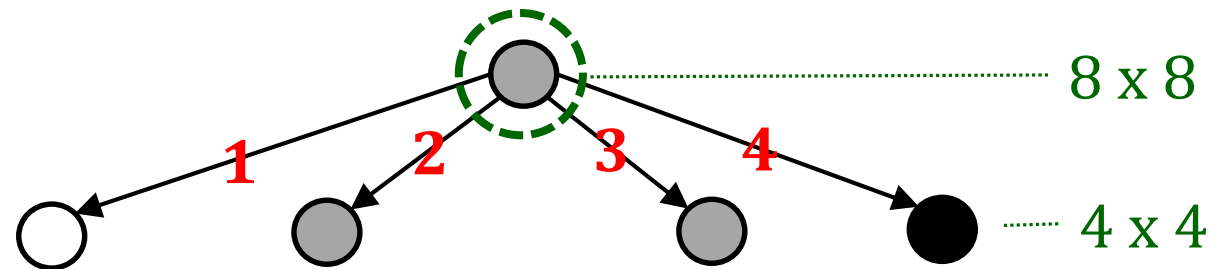
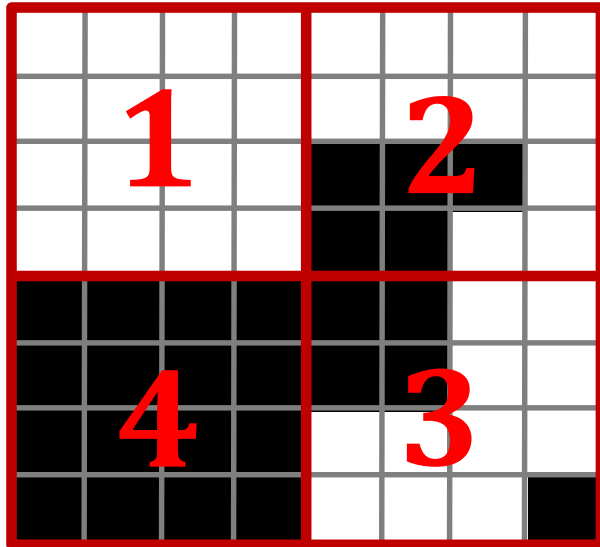
vide



en partie plein



plein

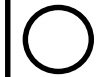


Quadtree

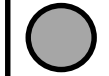
feuille

nœud

feuille



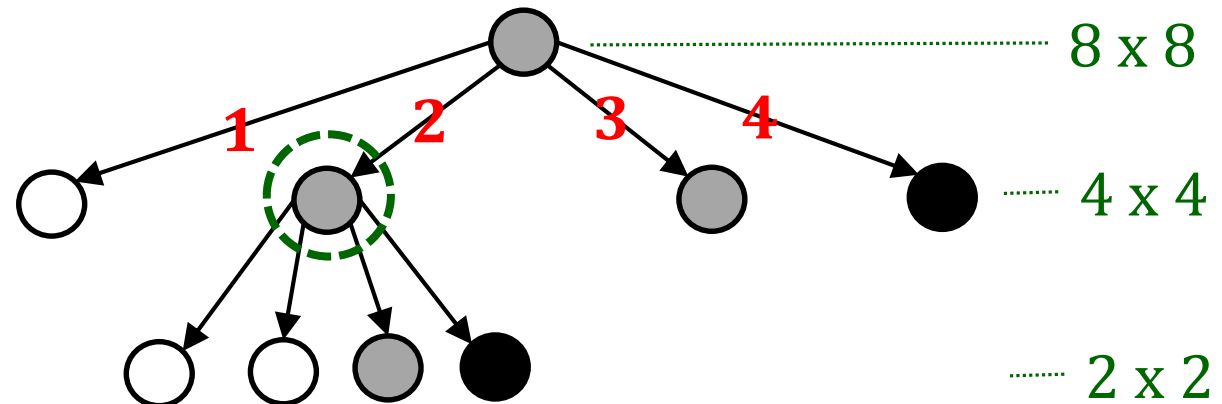
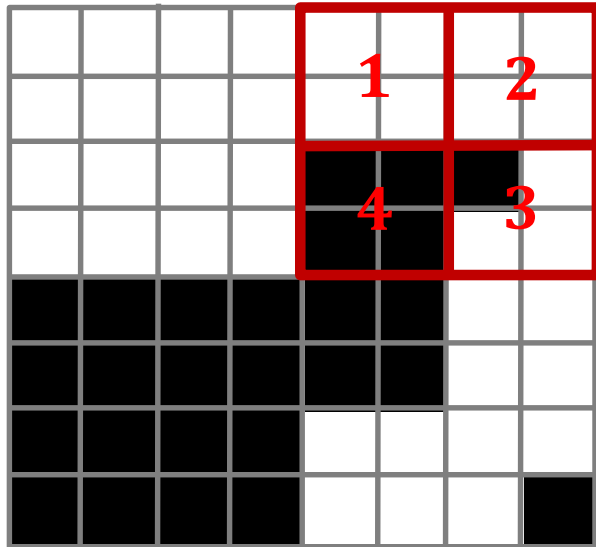
vide



en partie plein



plein

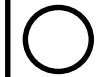


Quadtree

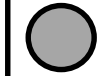
feuille

nœud

feuille



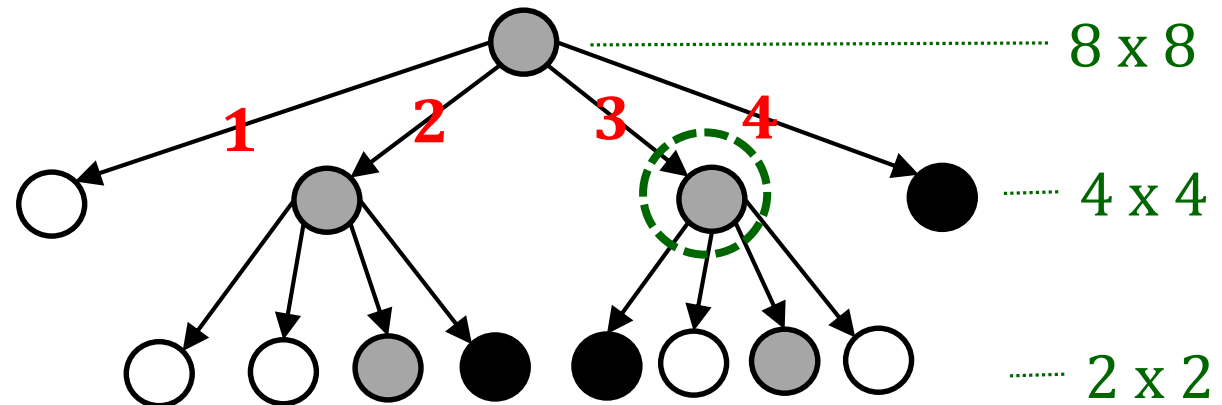
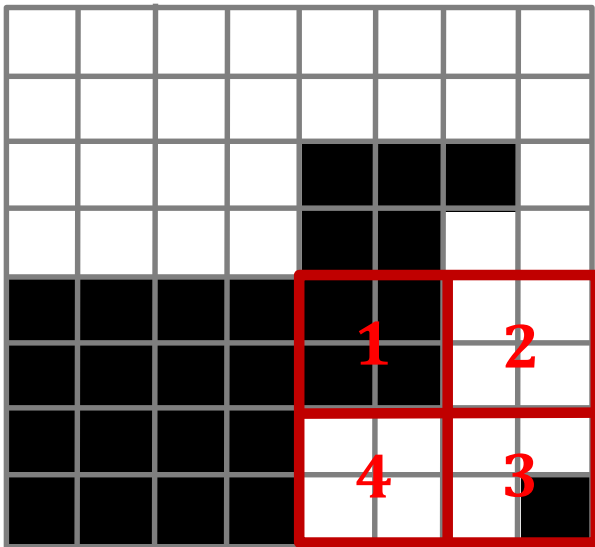
vide



en partie plein



plein

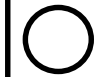


Quadtree

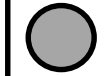
feuille

nœud

feuille



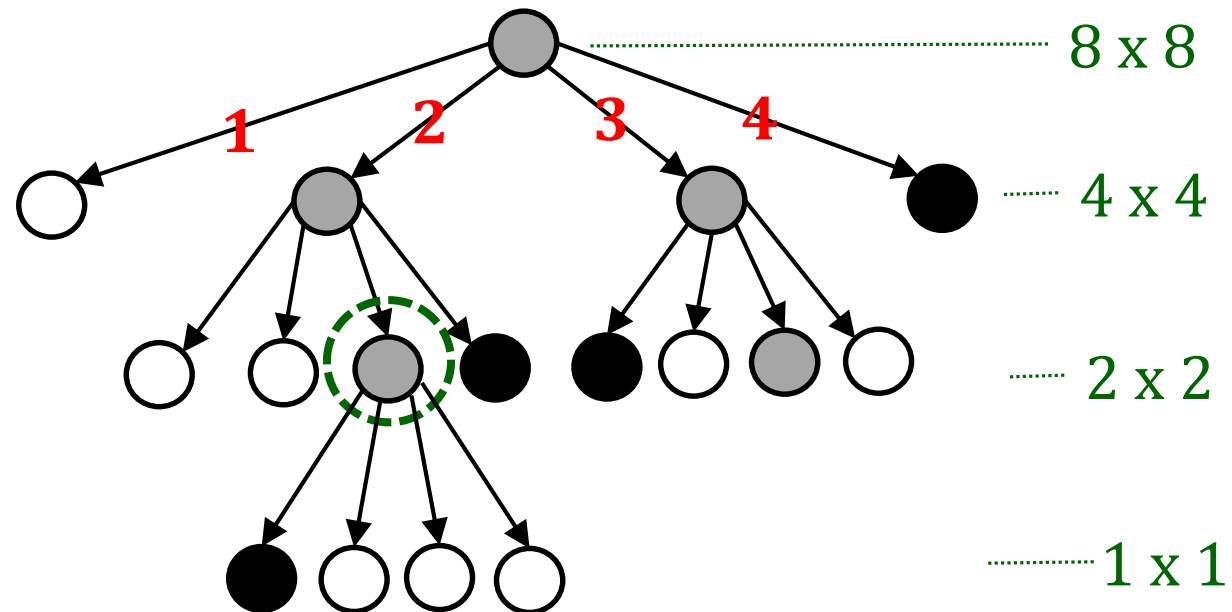
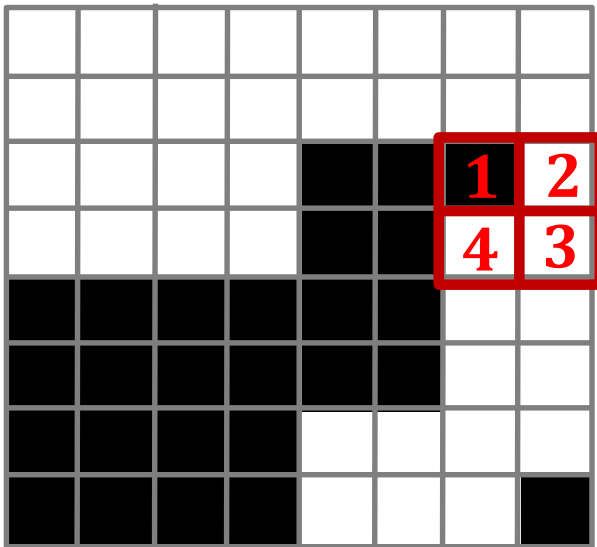
vide



en partie plein



plein

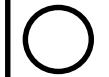


Quadtree

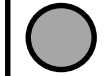
feuille

nœud

feuille



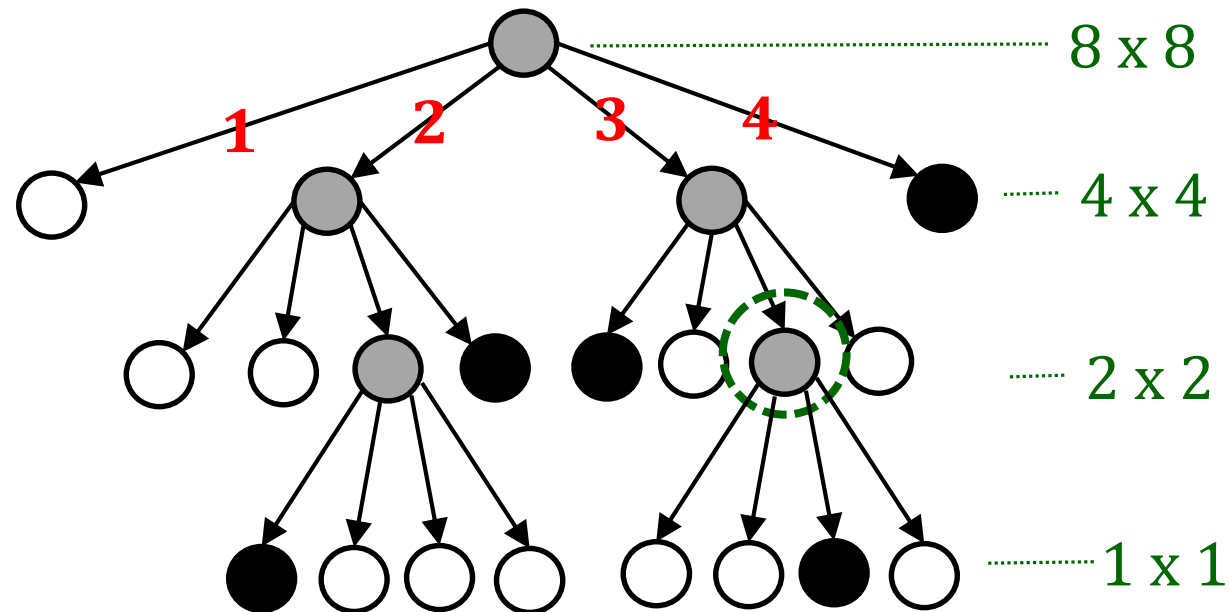
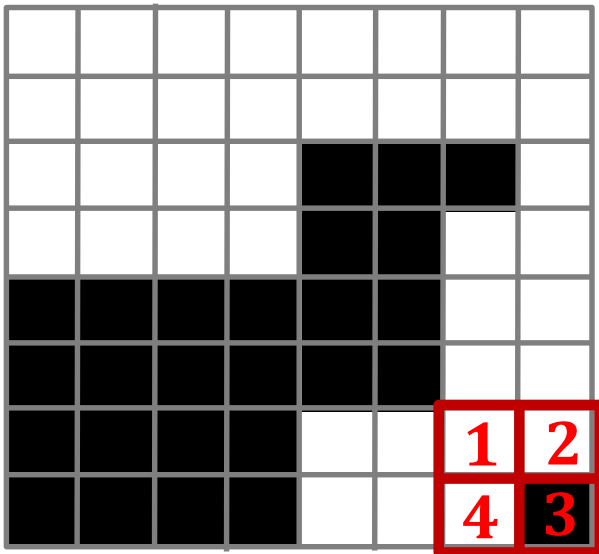
vide



en partie plein

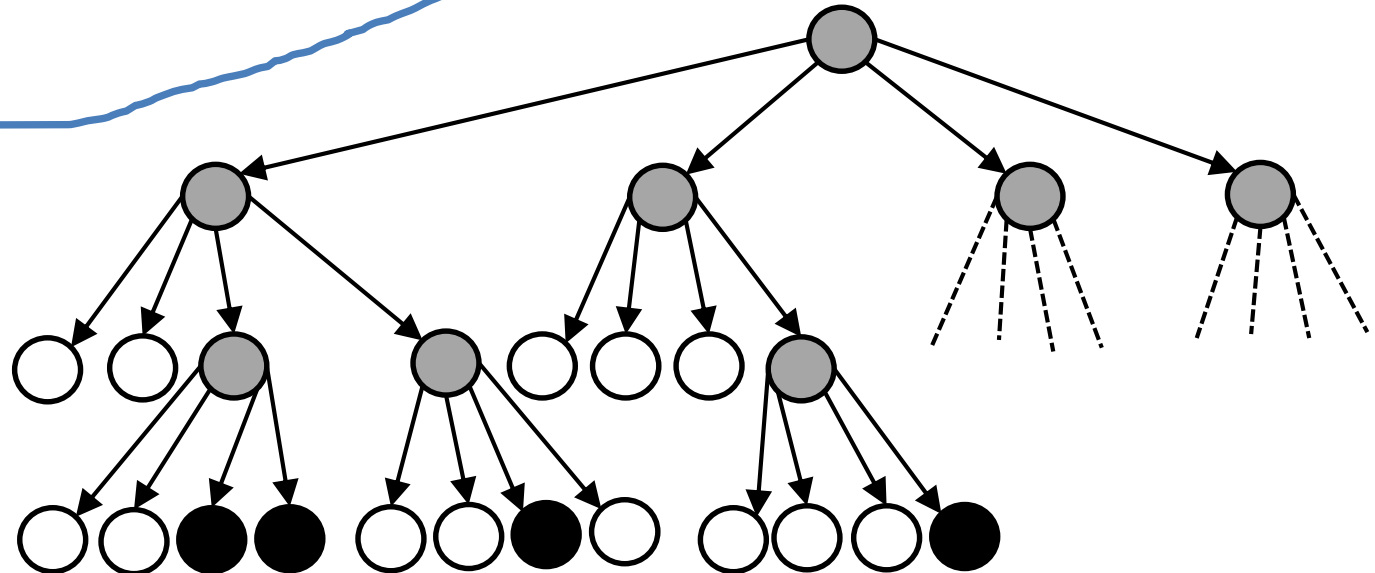
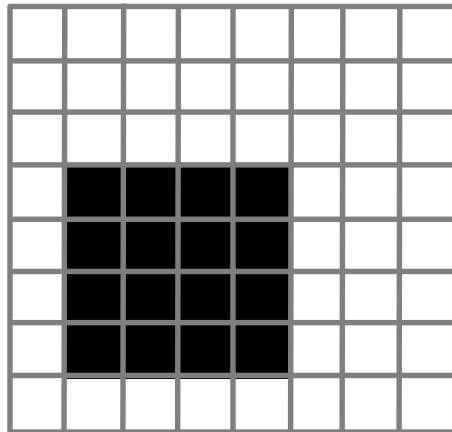
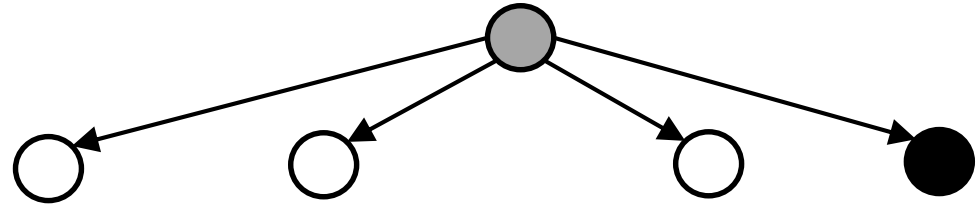
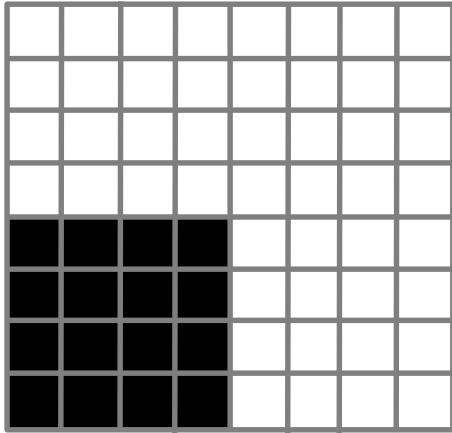


plein



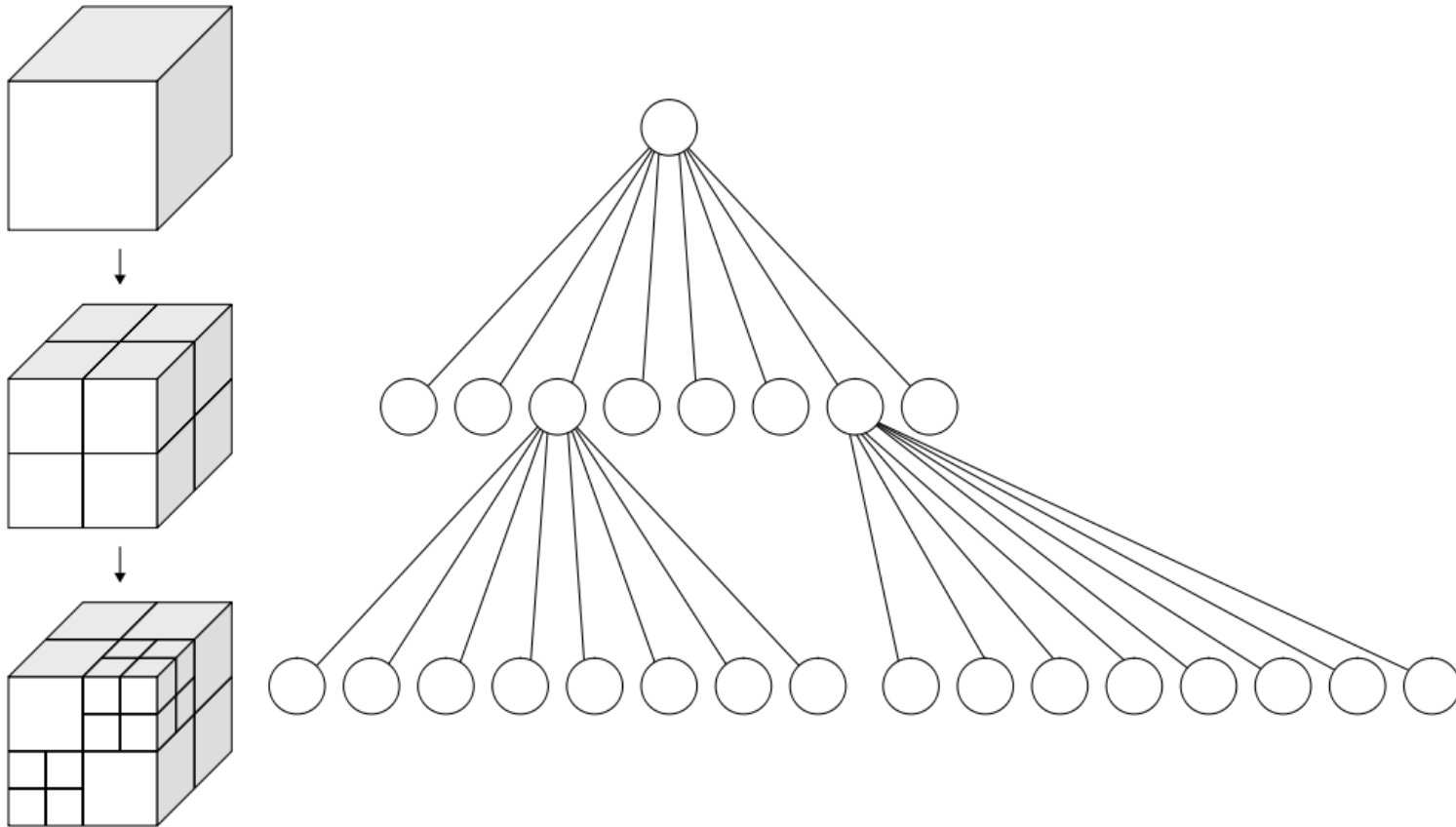
Quadtree : instabilité

- Petit changement à l'entrée $\neq$ petit changement à la sortie



Octree

- Pour les représentations en 3D

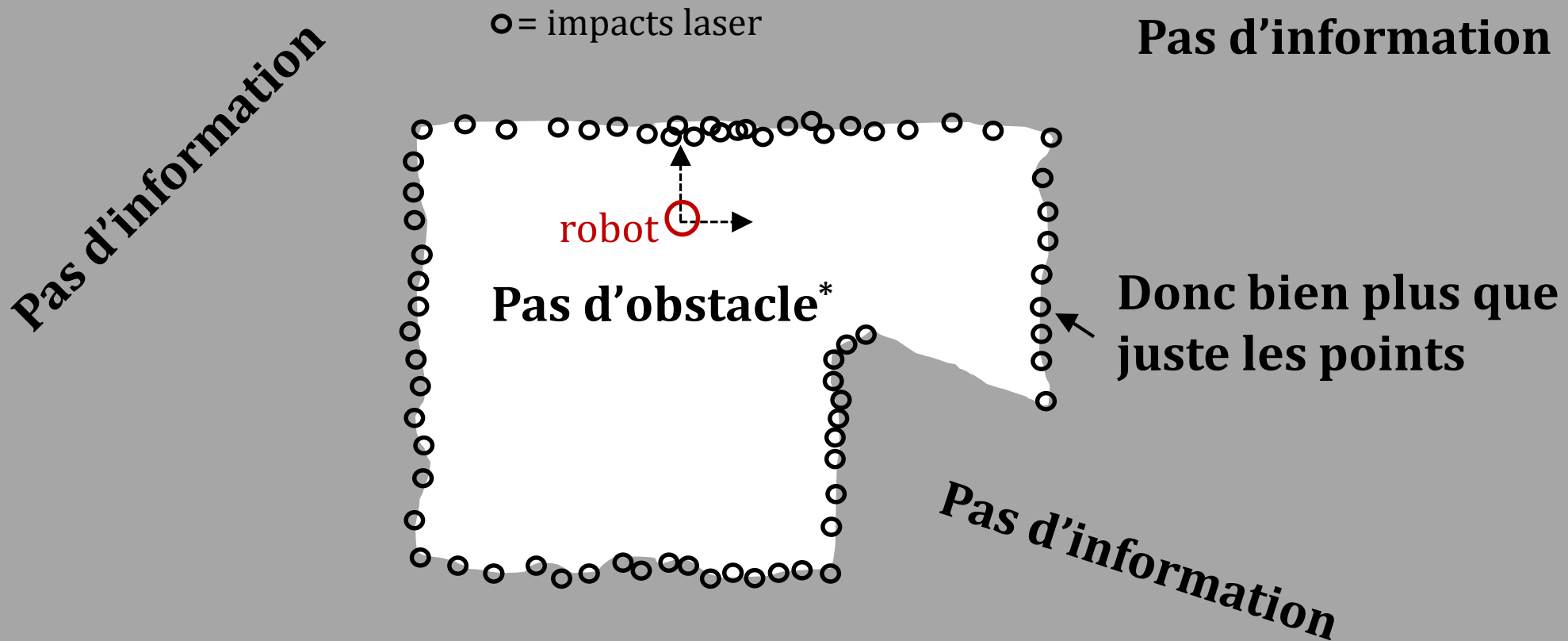


tiré de wikipédia

Construction d'une carte de type grille d'occupation (*occupancy grid*)

Grille d'occupation : intuition

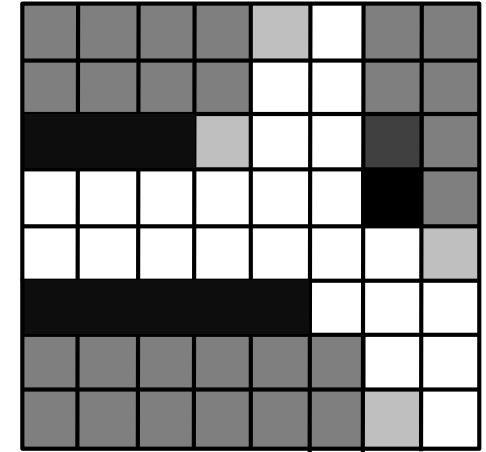
- Quelles informations vous avez, avec ces mesures laser?



**En supposant que les obstacles soient plus larges que l'espace entre les faisceaux*

Gérer l'information : *grille d'occupation*

- Carte du monde divisée en carrés de tailles égales ($10 \times 10 \text{ cm}$)
- Chaque case de la grille encode la probabilité qu'un obstacle y soit présent :
 - $p(c)=0$  si libre (certain)
 - $p(c)=0.5$  si on ne connaît rien
 - $p(c)=1.0$  si obstacle (certain)
- On connaît parfaitement la pose x_t du robot¹
- Au début, toutes les cases sont à $p(c)=0.5$ (environnement inconnu)
- Mesure z_t du capteur indique présence/absence d'obstacle, en probabilité, via l'inverse de la fonction du capteur



Création de la carte d'occupation m

- L'on cherche carte m composée de cases c_i contenant un obstacle (notez l'absence des commandes $u_{1:t}$)

$$p(m \mid x_{1:t}, z_{1:t}) \text{ où } m = \{c_1, \dots, c_N\}$$

- En considérant les cases comme indépendantes¹, on peut simplifier le problème en factorisant selon c_i : $p(c_1, c_2 \mid A) = p(c_1 \mid A) p(c_2 \mid A)$

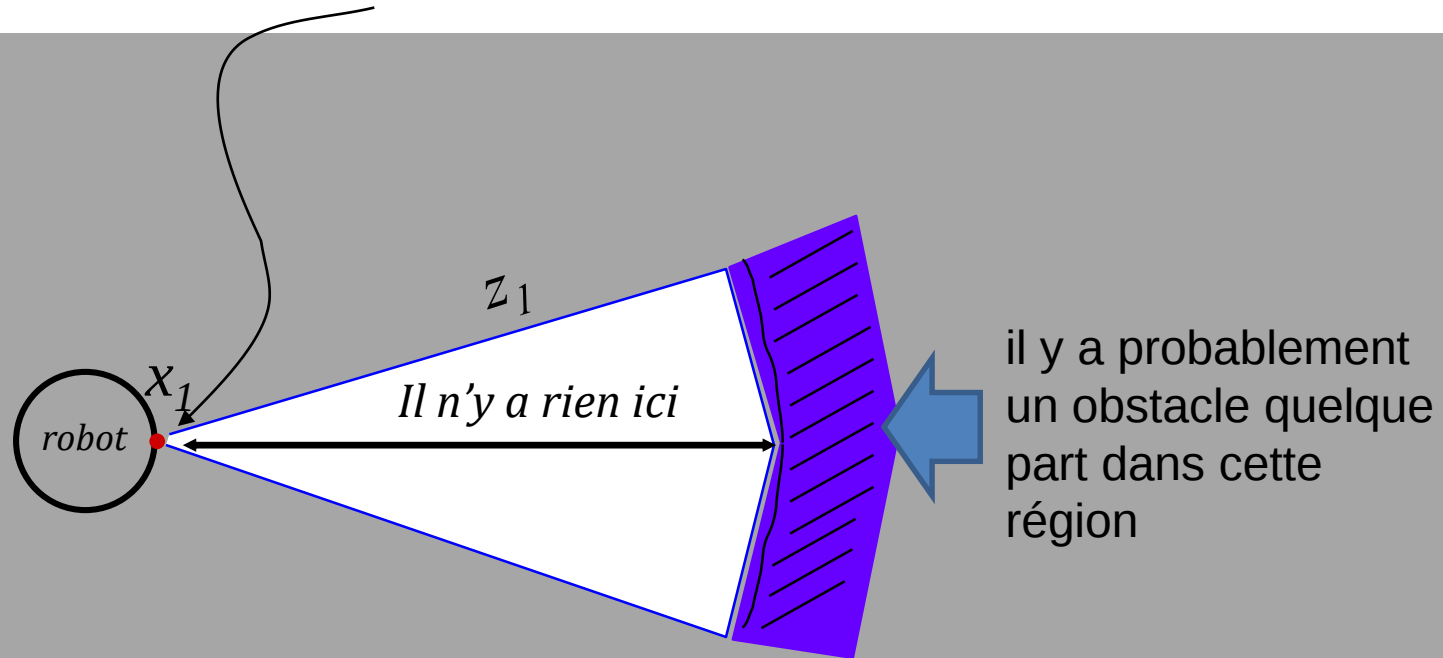
$$p(m \mid x_{1:t}, z_{1:t}) = \prod_{i=1}^N p(c_i \mid x_{1:t}, z_{1:t})$$

Création de la carte d'occupation m

- Comme les cases c_i sont indépendantes, incorporer une nouvelle donnée $\{x_t, z_t\}$ consistera à mettre à jour de façon individuelle les cases c_i affectées
- Simplifie/accélère grandement le problème...
- ... mais au prix d'une certaine perte d'exactitude (corrélation spatiale non-respectée)
- Quelles sont les cases c_i affectées?
 - dépend (du modèle inverse) du capteur

Grille d'occupation avec sonar

Si un sonar indique obstacle à 4 mètres?

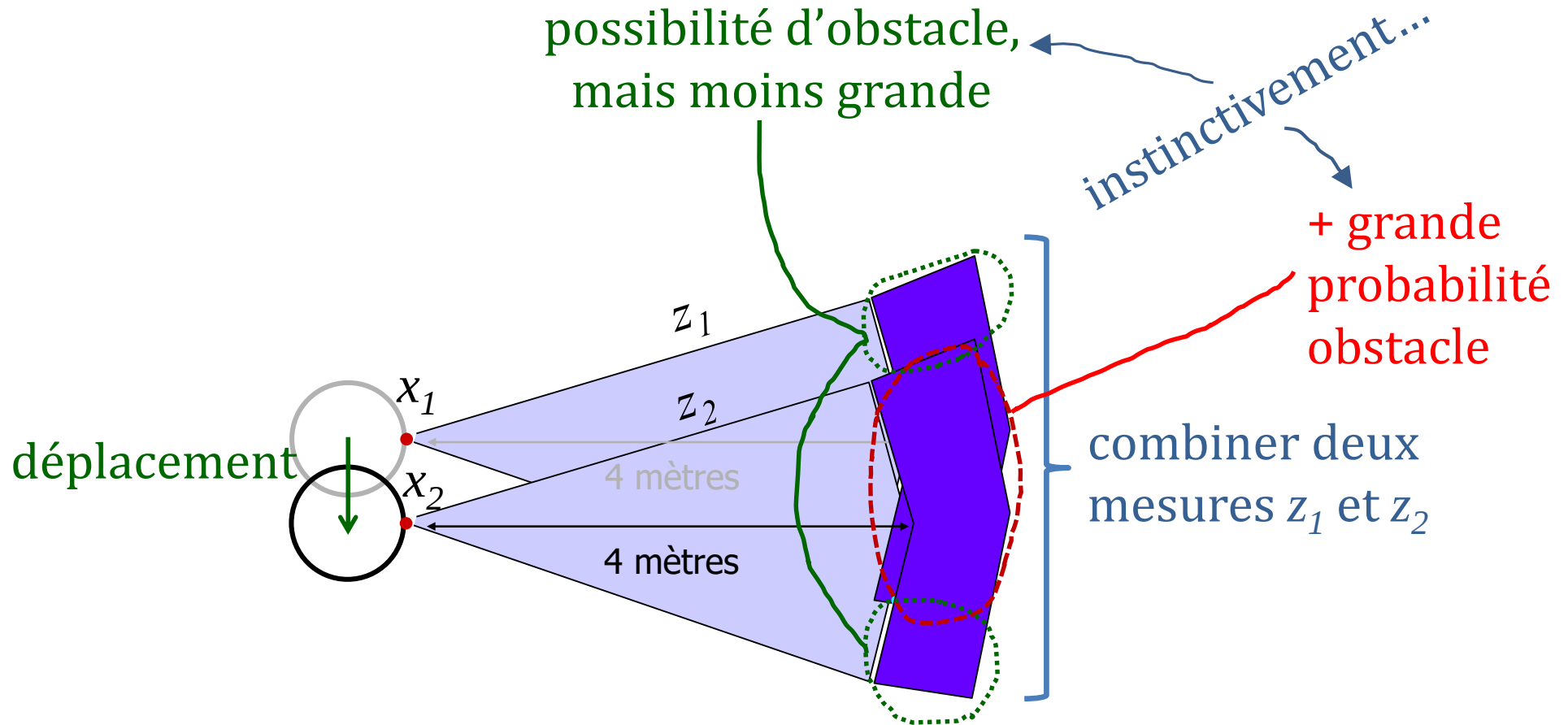


*Rappel : sonar émet en
forme de cône*

Carte locale

-  vide
-  pas d'information
-  obstacle

Grille d'occupation : accumulation d'évidences



Comment combiner ces informations de manière fondée et efficace?

Combinaison d'évidences sonar

- Combiner les **cotes**: $\text{cote}(p) = \frac{p}{(1-p)} = \frac{p}{\bar{p}}$ p ne s'est pas produit

- Si $p=75\%$ chance de gagner, la **cote** sera :

$$\frac{0.75}{(1-0.75)} = \frac{0.75}{0.25} = 3$$

- Chacune des cases c_i contiendra la **cote** (ou $\log(\text{cote})$ qui encode la probabilité de présence d'un obstacle.
- Pourquoi les **cotes**? pour faciliter les calculs¹...

Algorithme grille d'occupation

```
occupancy_grid_mapping( $\{C_{i,t-1}\}$ ,  $x_t$ ,  $z_t$ )
```

```
  for all cells  $m_i$  in map do
```

```
    if  $m_i$  is in perceptual field of  $z_t$  then
```

```
       $C_{i,t} = C_{i,t-1} \times$  cote_sensor( $m_i$ ,  $x_t$ ,  $z_t$ )
```

```
    else
```

```
       $C_{i,t} = C_{i,t-1}$ 
```

```
    endif
```

```
  endfor
```

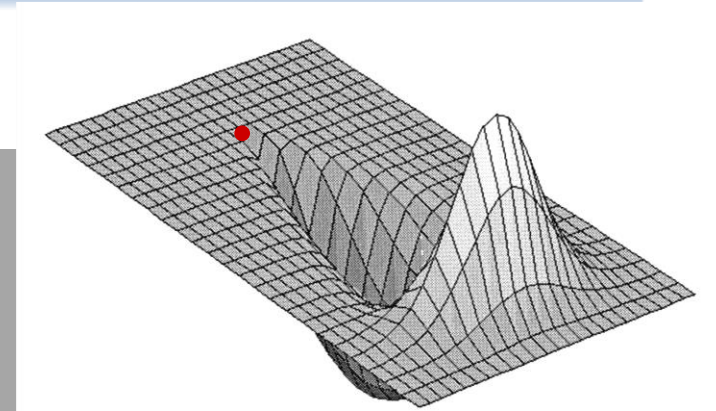
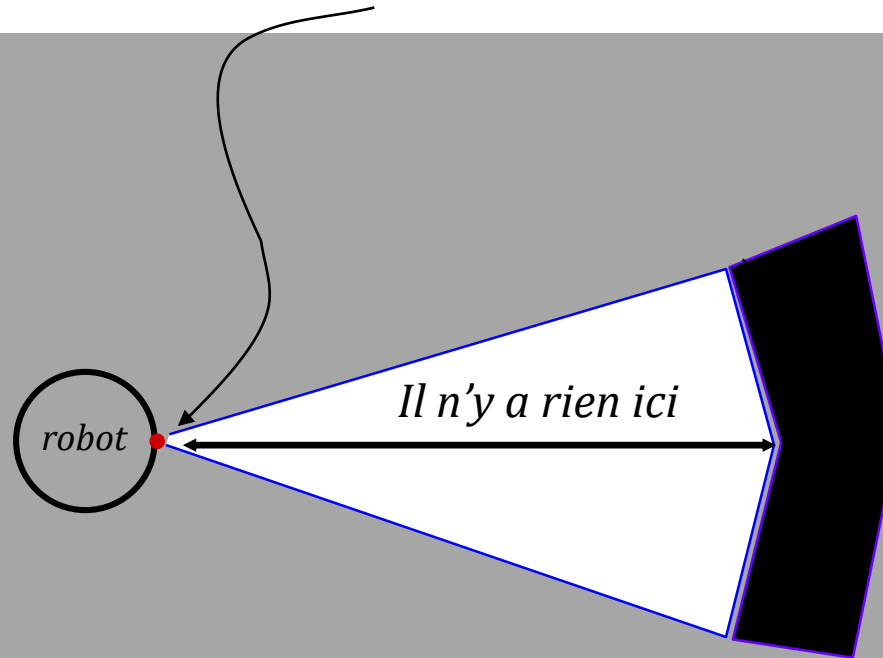
```
  return  $\{C_{i,t}\}$ 
```

Inverse du modèle du capteur

Carte de départ est le prior $C_{i,0} = \frac{p(\text{libre})}{p(\text{occupé})}$ (souvent 1)

Modèle inverse du capteur pour c : $p(c|z,x)$

Si un sonar indique obstacle à 4 mètres?



il y a probablement
un obstacle quelque
part dans cette
région

*Rappel : sonar émet en
forme de cône*

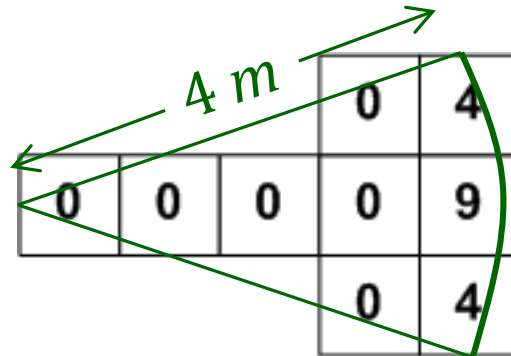
Cote du modèle inverse

-  cote < 1
-  cote = 1 (pas d'info)
-  cote > 1

Exemple

Fonction du capteur sonar pour $z = 4$ m

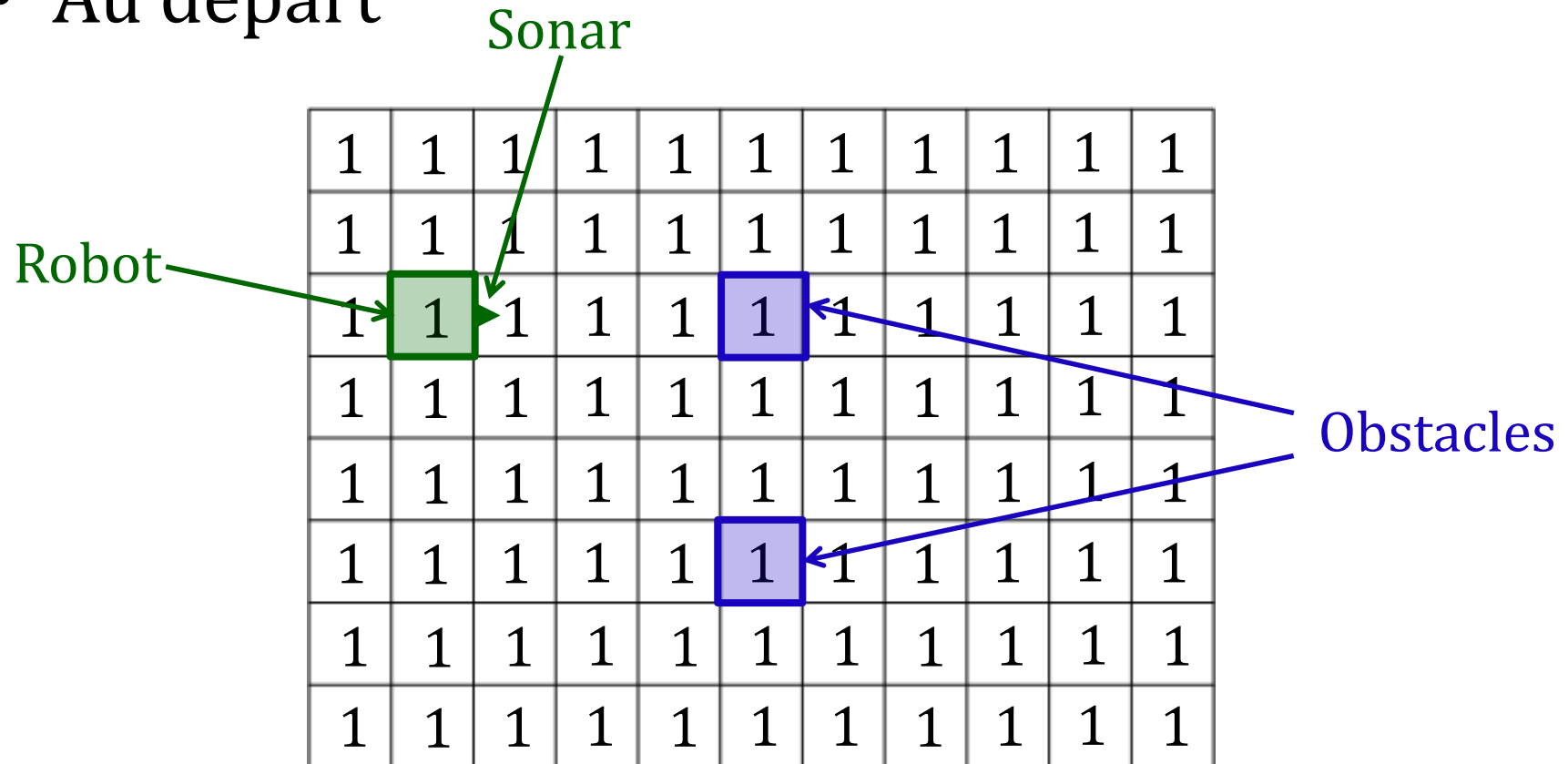
`cote_sensor(...)` =



À l'intérieur du
cône de sonar

Exemple

- Au départ



Exemple

- Première mesure $Z_1 = 4\ m$

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0	0	9	
			0	4

Exemple

- Première mesure $Z_1 = 4\ m$

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	9	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0	0	9	
			0	4

Exemple

- Déplacement

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	9	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0	0	9	
			0	4

Exemple

- Measure $Z_2 = 4\ m$

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	9	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0	0	9	
			0	4

Exemple

- Measure $Z_2 = 4\ m$

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0	0	9	
			0	4

Exemple

- Déplacement

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0	0	9	
			0	4

Exemple

- Measure $Z_3 = 4 \text{ m}$

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0	0	9	
			0	4

Exemple

- Measure $Z_3 = 4 \text{ m}$

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	0	0	0	0	144	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0		0	9
			0	4

Exemple

- Long déplacement...

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	0	0	0	0	144	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0		0	9
			0	4

Exemple

- Measure $Z_4=3m$

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	0	0	0	0	144	1	1	1	1	1
1	0	0	0	0	36	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

			0	4
0	0	0	9	
			0	4

Exemple

- Measure $Z_4=3m$

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	0	0	0	0	0	144	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1
1	0	0	0	0	0	36	1	1	1	1
1	1	1	1	0	4	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

cote_sensor(z=4)

			0	4
0	0	0	0	9
			0	4

cote_sensor(z=3)

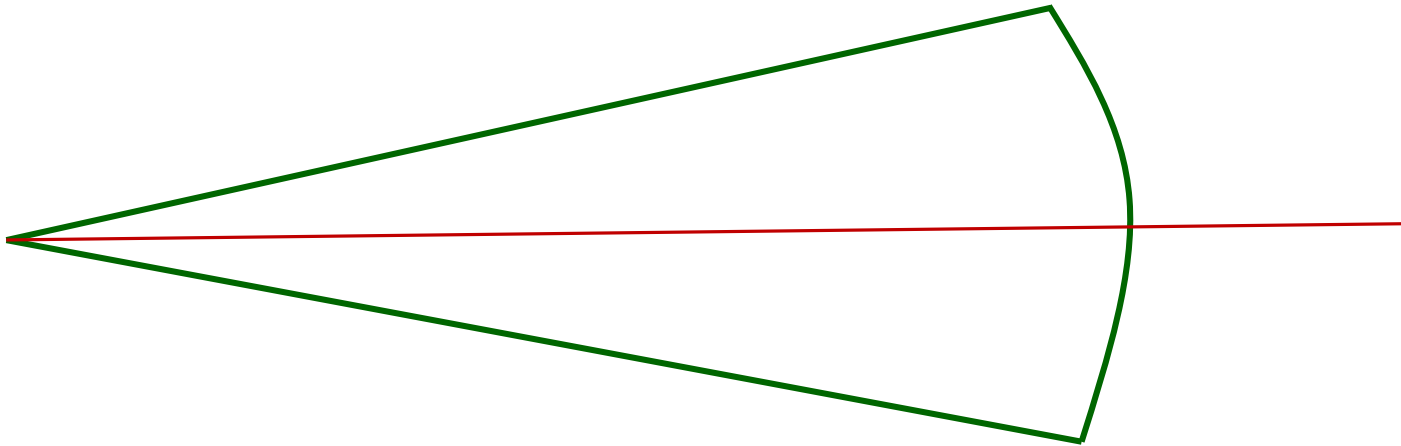
			0	4
0	0	0	9	
			0	4

Notes sur l'exemple précédent

- Mauvaise idée d'avoir des cotes de 0 pour le capteur. Préférable d'avoir des valeurs faibles mais non nulles pour tenir compte des erreurs possibles
- Importance de la trajectoire dans la construction
- Peut utiliser le log des cotes :
 - passe de multiplications à des additions
 - va aller de $-\infty$ à $+\infty$ (0 = aucune connaissance)
 - meilleure stabilité numérique

Avec laser 2D?

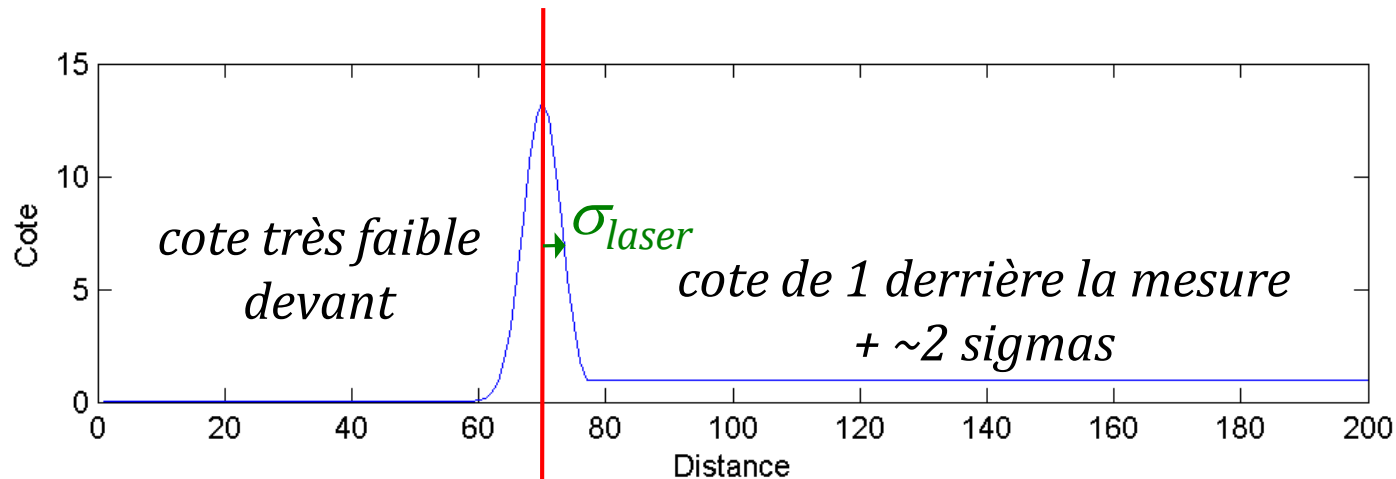
- Un peu plus compliqué, car le « cône » du **laser** est beaucoup plus étroit...



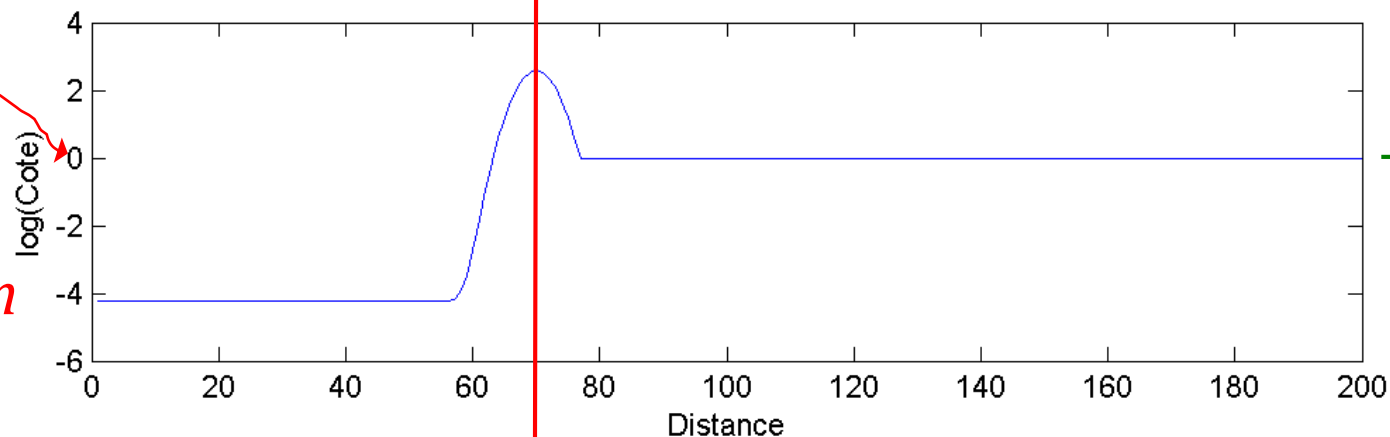
- Il ne mettra pas à jour beaucoup de cases...

Exemple matlab GridMap . zip

- Pour les cotes du capteur, j'ai utilisé un modèle approximatif



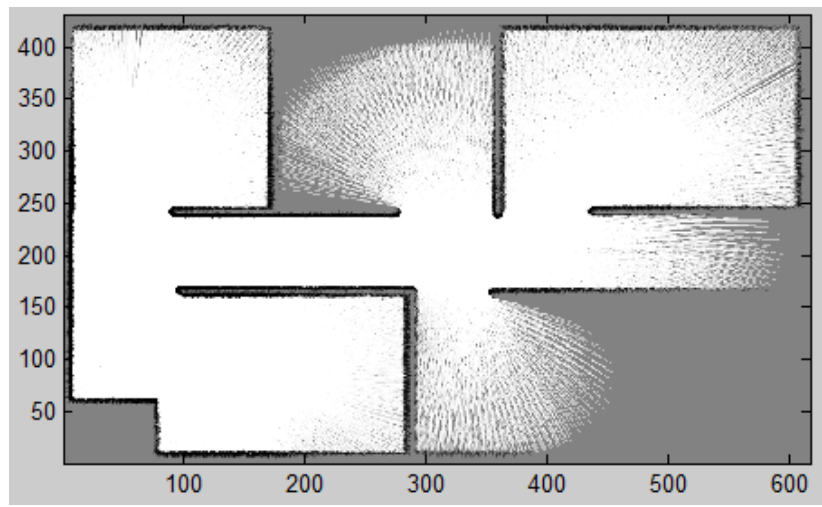
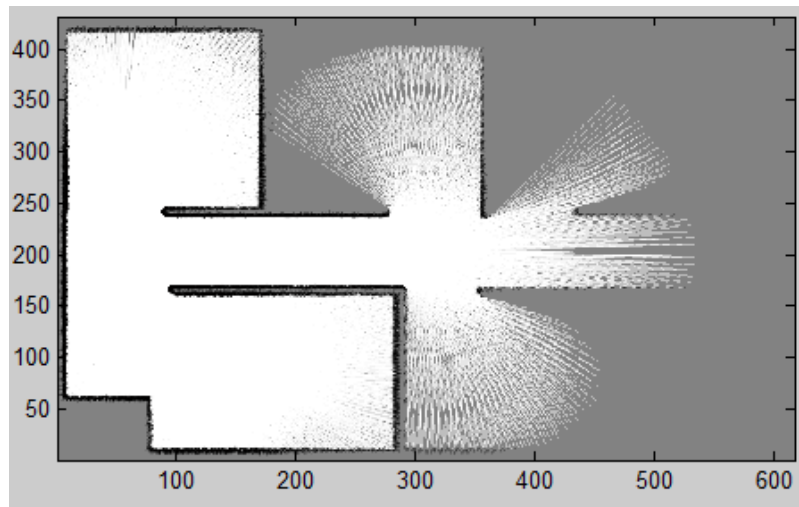
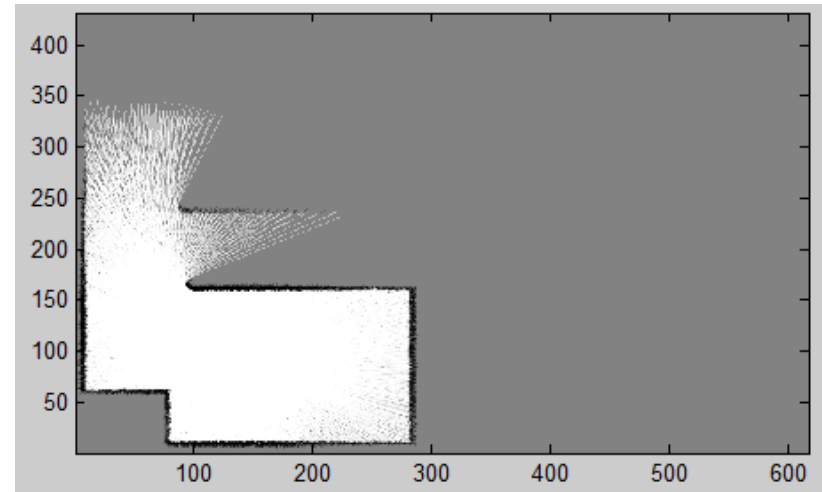
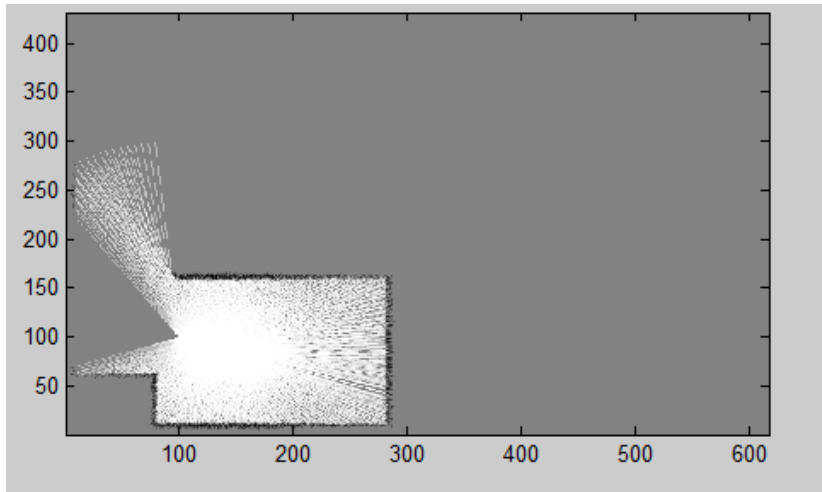
*log(1)=0
indique
absence
d'information*



occupé

libre

Exemple matlab GridMap .zip



Pourquoi faire grille occupation?

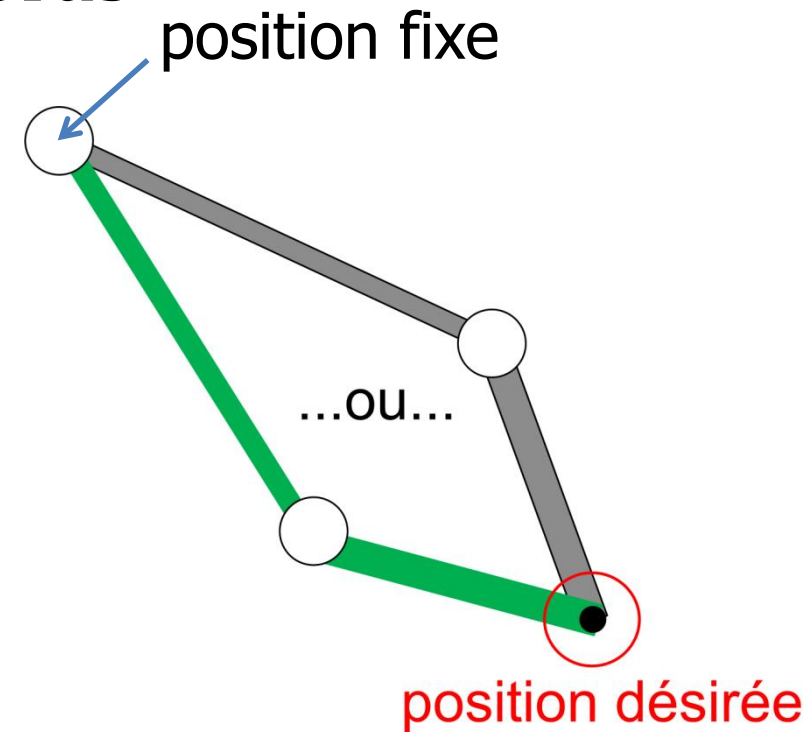
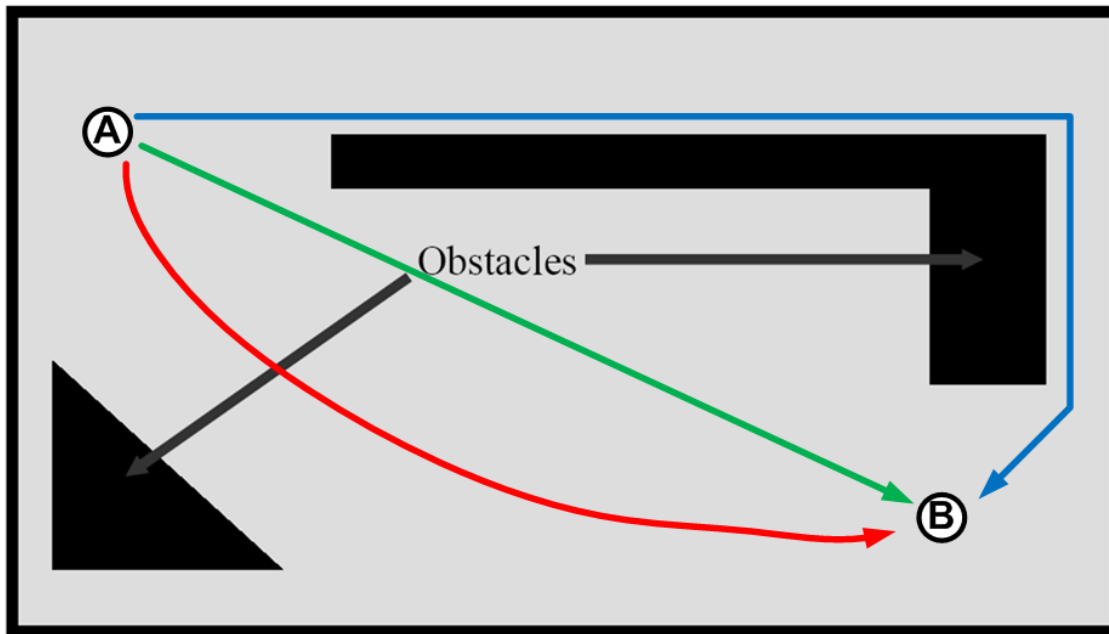
- Pour planifier!
 - explorer les endroits inconnus
 - savoir où sont les endroits libres sur la carte
- Pour la localisation!
 - possible, si la carte a été construite de manière fiable
 - avec des scan laser :
<http://wiki.ros.org/gmapping>

G. Grisetti, C. Stachniss, and W. Burgard: Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters, *IEEE Transactions on Robotics*, Volume 23, pp. 34-46, 2007.

Planification

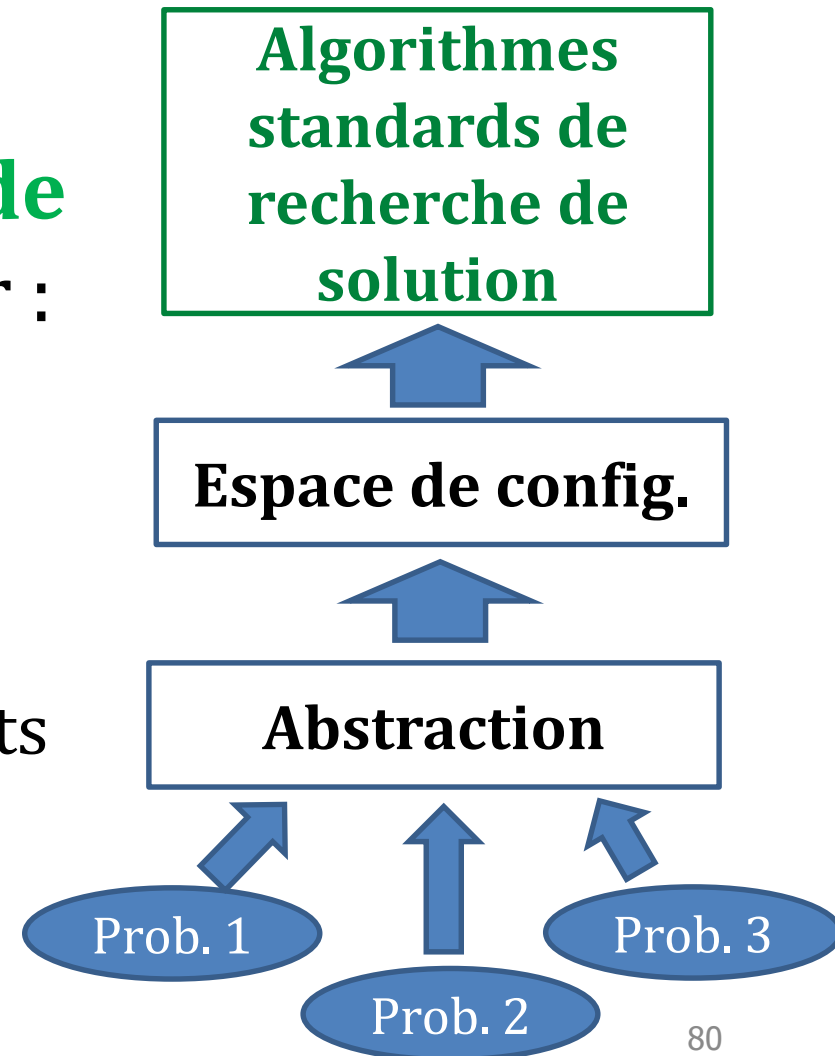
Planification : mal posé

- Déplacer un robot de A vers B
- Positionner l'extrémité d'un bras

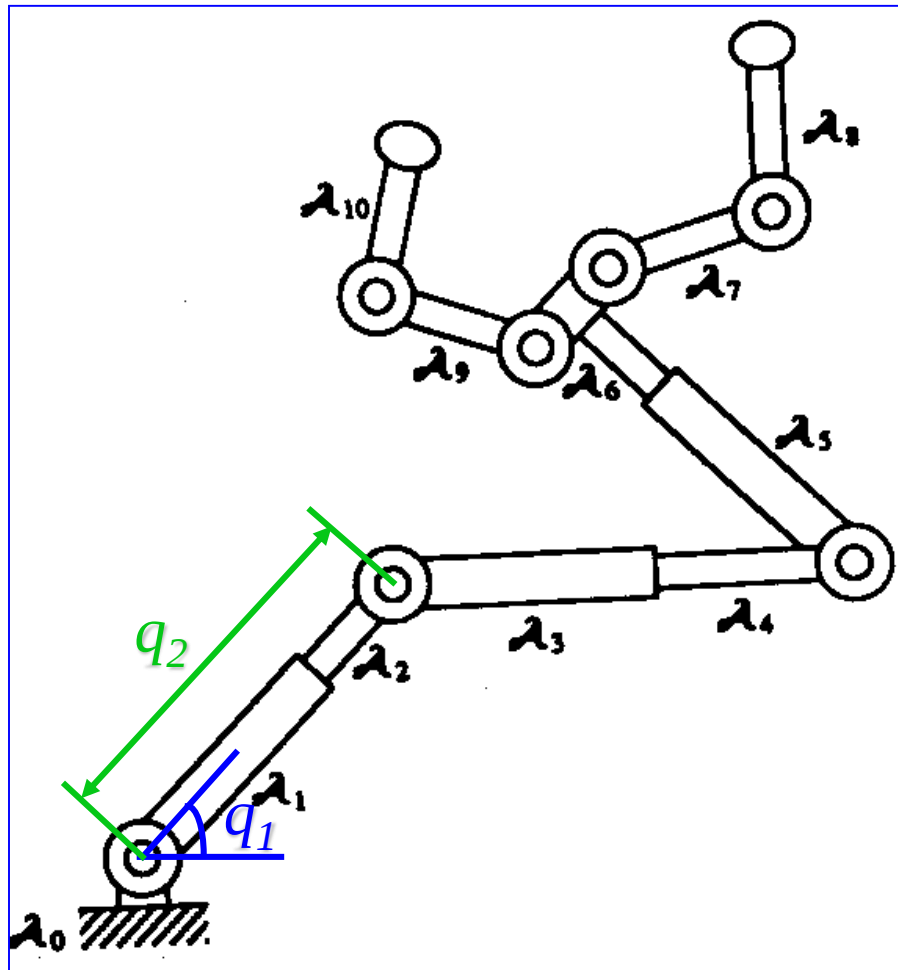


Espace de configuration

- Représentation abstraite d'un problème de planification
- Employer des **algorithmes de recherches** génériques pour :
 - robot 2D glisse en x, y
 - robot conduite différentielle
 - bras articulé
 - robot humanoïde avec 30 joints
 - etc.



Exemple configuration (bras articulé)

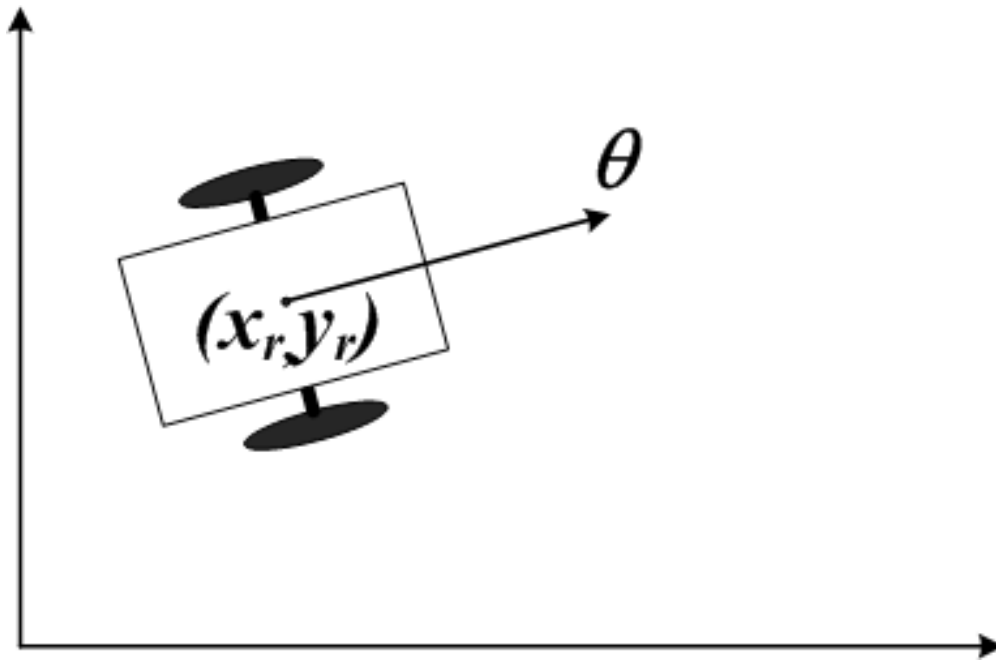


- La configuration d'un bras est l'ensemble des angles θ_i et étirements d_j .
- Pour un robot articulé complexe de 10 actionneurs :

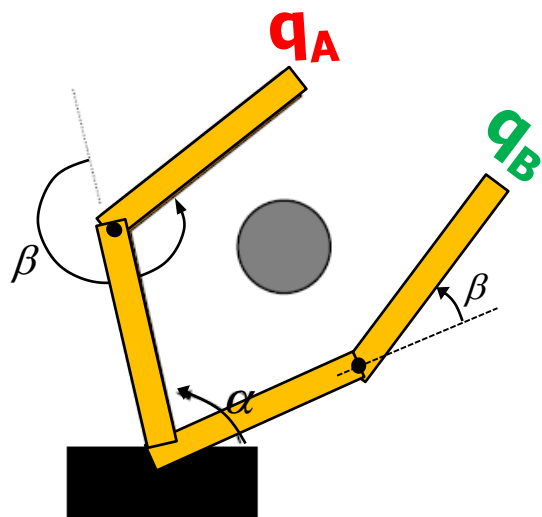
$$\vec{q} = (q_1, q_2, \dots, q_{10})$$

Exemple configuration (robot mobile)

- Espace de travail en 2D
- Configuration : $\vec{q} = (x_r, y_r, \theta)$

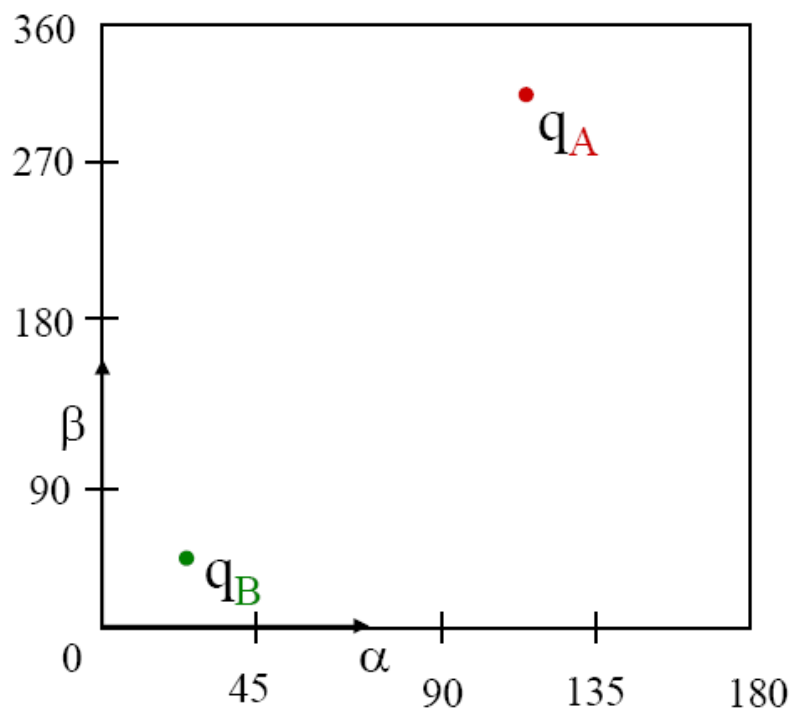


Obstacle dans l'espace de travail



Un obstacle dans l'espace de travail

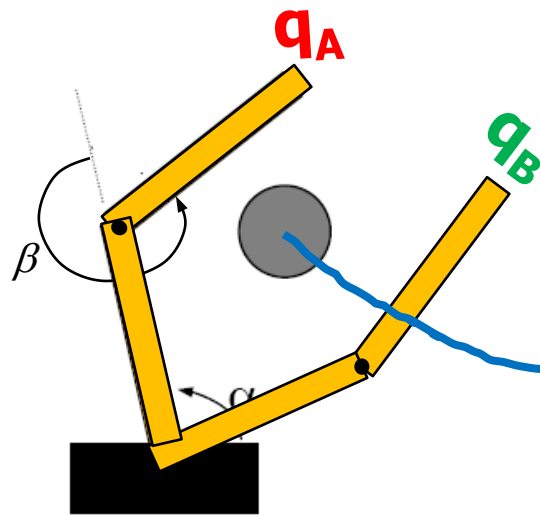
Où placer  ?



Espace de configuration

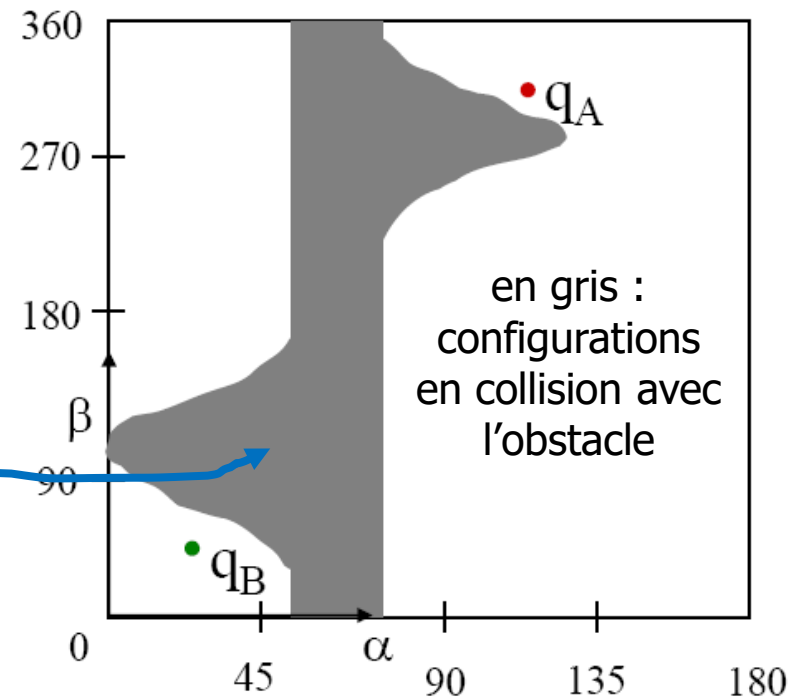
(en 2D, car 2 joints)

Obstacle dans l'espace des configurations



Un obstacle dans l'espace de travail

Comment aller de q_A vers q_B ?

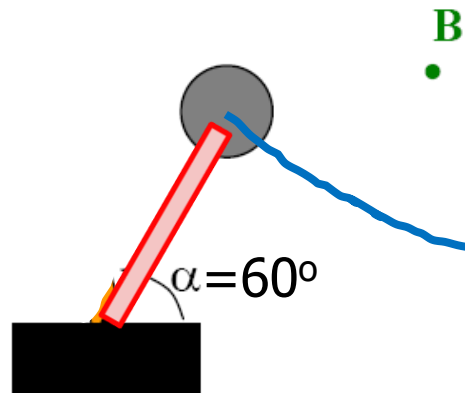


Espace de configuration

(en 2D, car 2 joints)

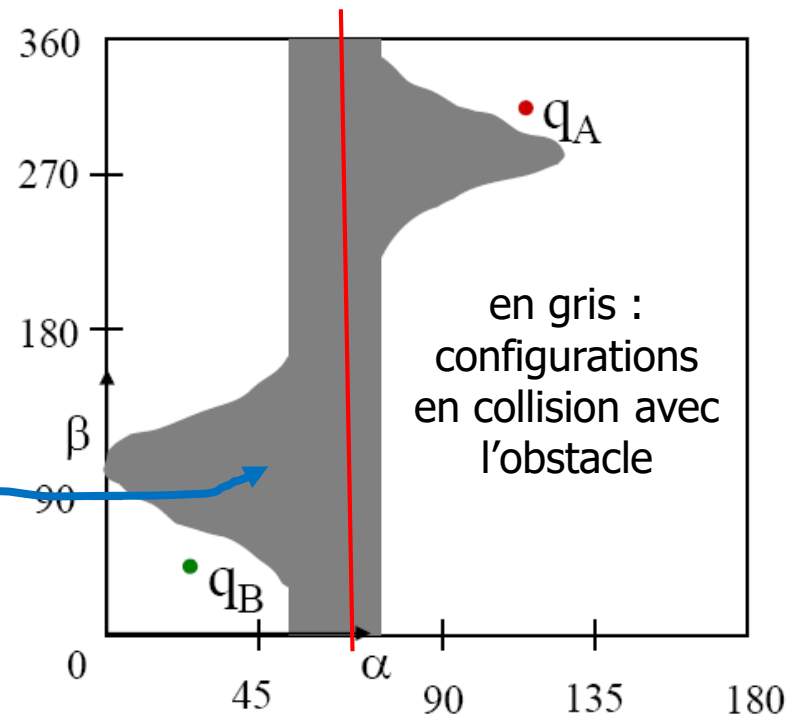
Obstacle dans l'espace des configurations

dès que $\alpha = 60^\circ$, on a collision, peu importe β



Un obstacle dans l'espace de travail

Comment aller de q_A vers q_B ?



Espace de configuration

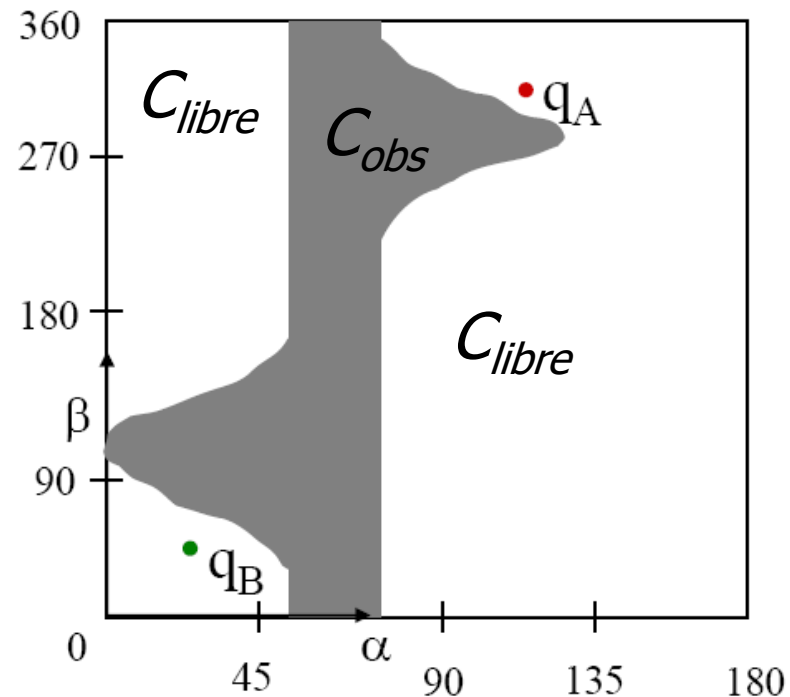
(en 2D, car 2 joints)

Espace des configurations : terminologie

L'espace des configurations C est composé :

- espace libre C_{libre}
- espace des obstacles C_{obs}

$$C = C_{libre} + C_{obs}$$



Espace de configuration
(en 2D, car 2 joints)

Obstacle dans l'espace des configurations

construction de l'espace des configurations avec obstacles

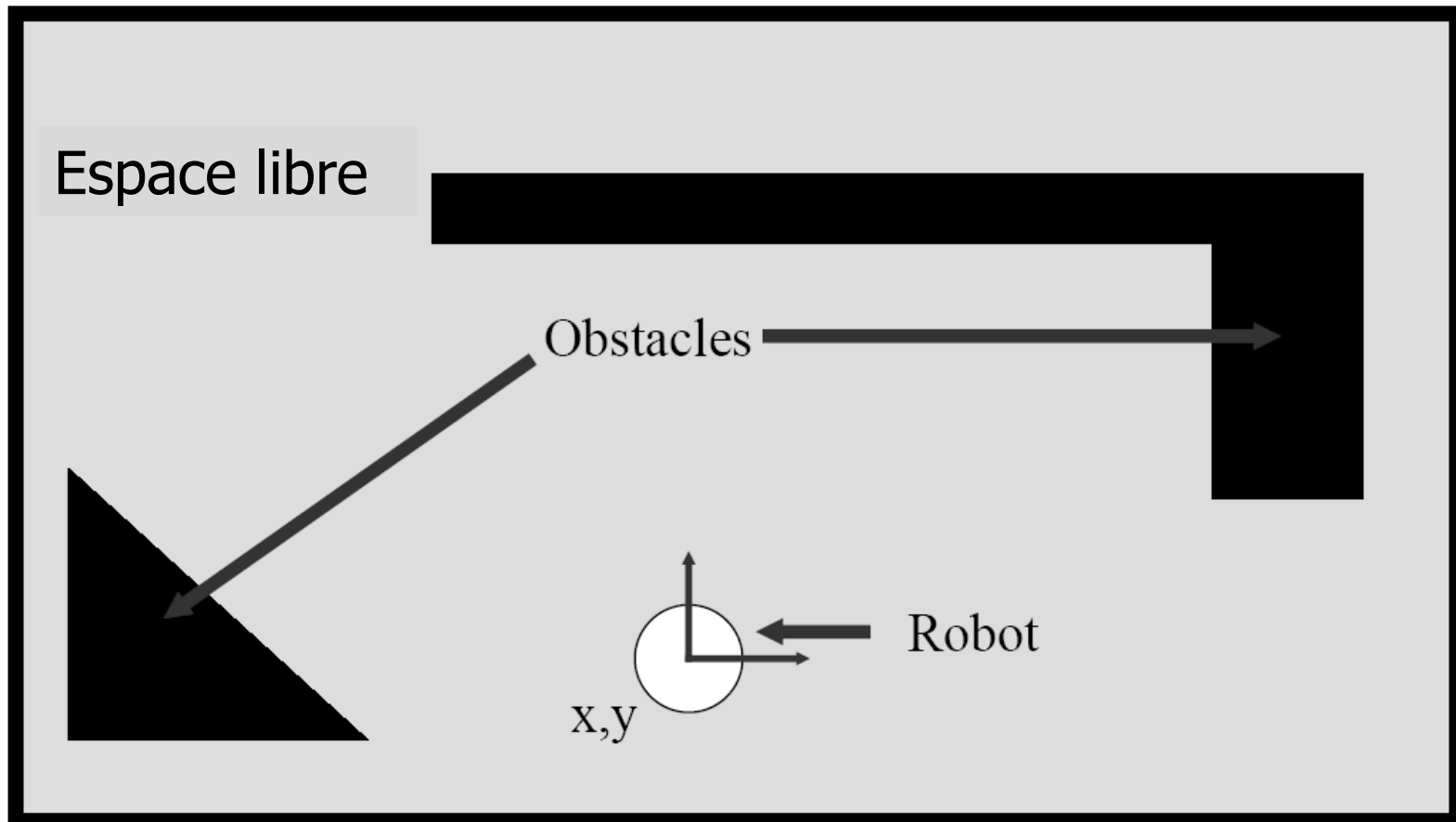


□ C_{libre}

■ C_{obs}

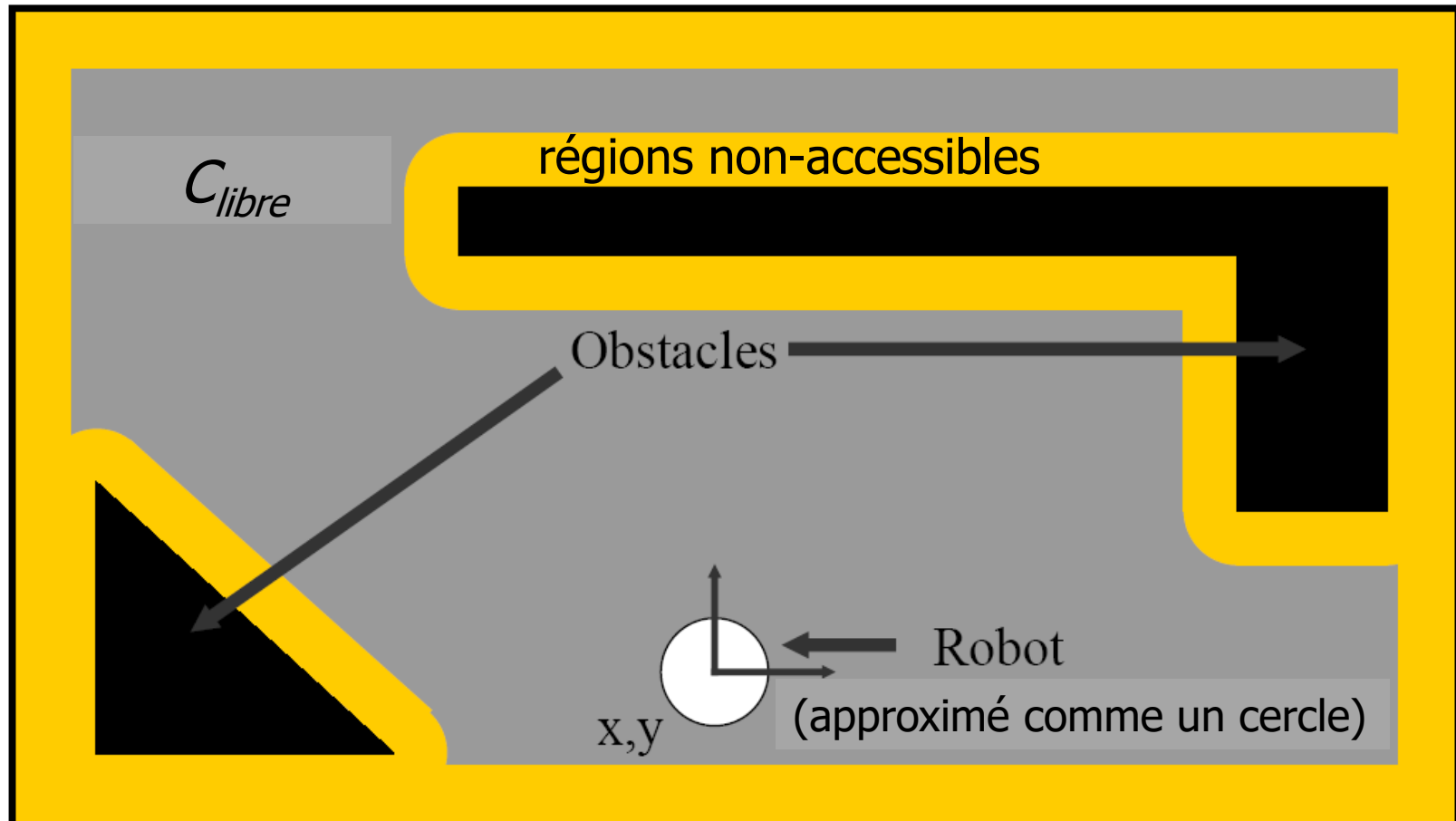
Espace de configuration C

- Pour un robot circulaire se déplaçant en x-y



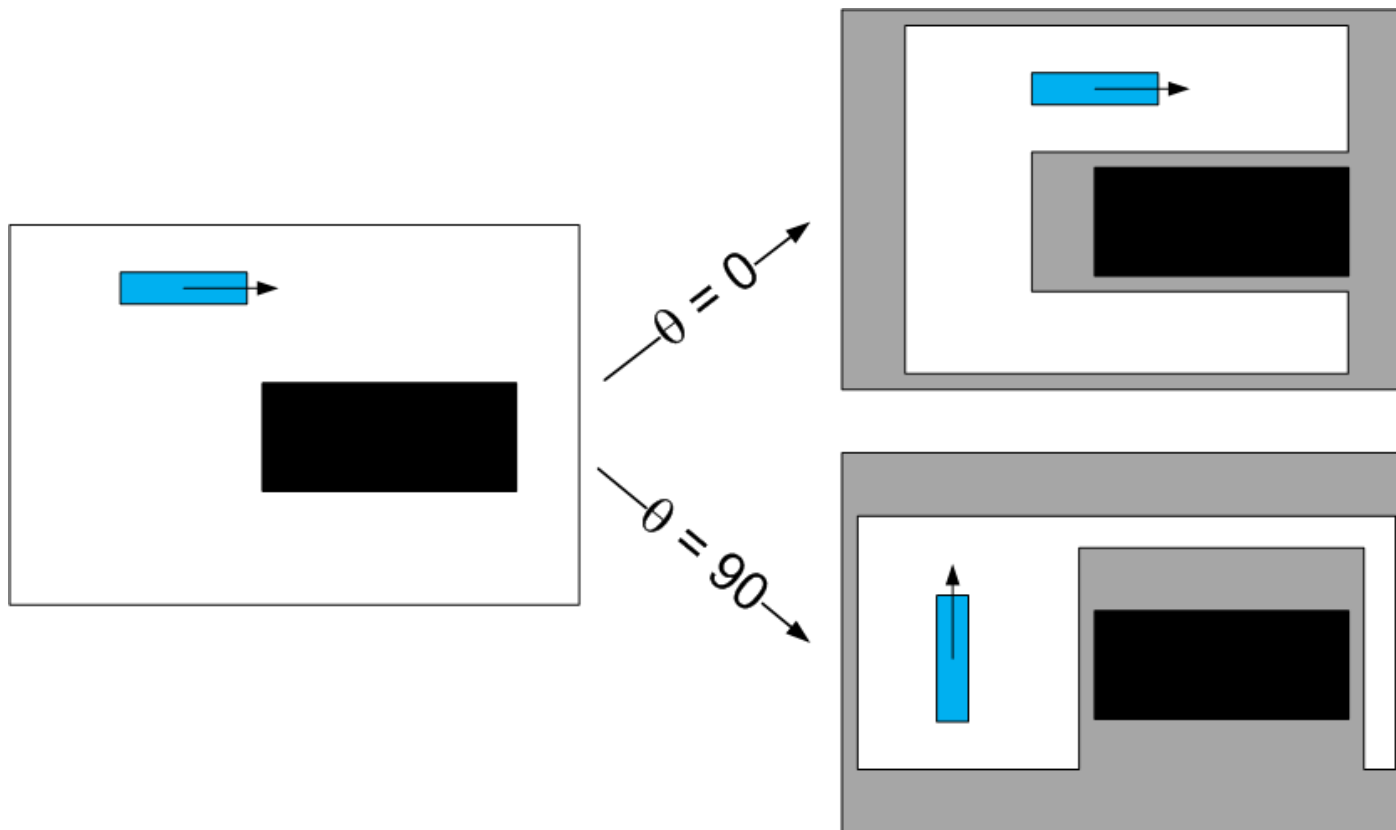
Espace de configuration C

- Le centre du robot ne peut être que dans le gris



Espace de configuration C

- Pour un robot se déplaçant en conduite différentielle : (x, y, θ) (3 dimensions)



Planification de trajectoire : abstraction

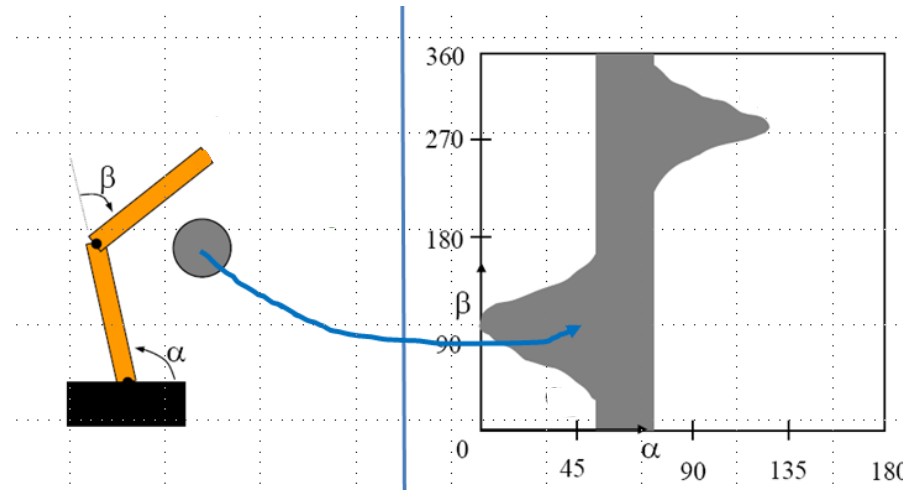
- ...aussi appelé « *piano mover's problem* »
- On a :
 - espace de travail W en $\mathbb{R}^2$ ou $\mathbb{R}^3$
 - régions d'obstacles O
 - un robot R avec géométrie dans W} « monde physique »
 - espace configuration C (avec $C = C_{libre} + C_{obs}$)
 - configuration initiale $q_1 \in C_{libre}$
 - configuration finale $q_2 \in C_{libre}$
- Trouver une trajectoire τ continue dans C_{libre} entre q_1 et q_2 .

Planification de trajectoire : abstraction

- ...aussi appelé « *piano mover's problem* »
- On a :

- espace de travail W en $\mathbb{R}^2$ ou $\mathbb{R}^3$
- régions d'obstacles O
- un robot R avec géométrie dans W
- espace configuration C (avec $C = C_{libre} + C_{obs}$)

calcul de C_{obs}
n'est pas facile



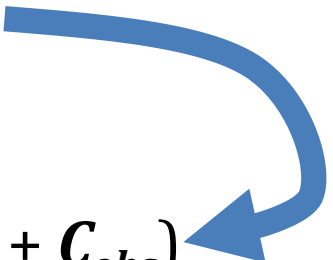
Planification de trajectoire : abstraction

- ...aussi appelé « *piano mover's problem* »

- On a :

- espace de travail W en $\mathbb{R}^2$ ou $\mathbb{R}^3$
- régions d'obstacles O
- un robot R avec géométrie dans W
- espace configuration C (avec $C = C_{libre} + C_{obs}$)
- configuration initiale $q_1 \in C_{libre}$
- configuration finale $q_2 \in C_{libre}$

calcul de C_{obs}
n'est pas facile

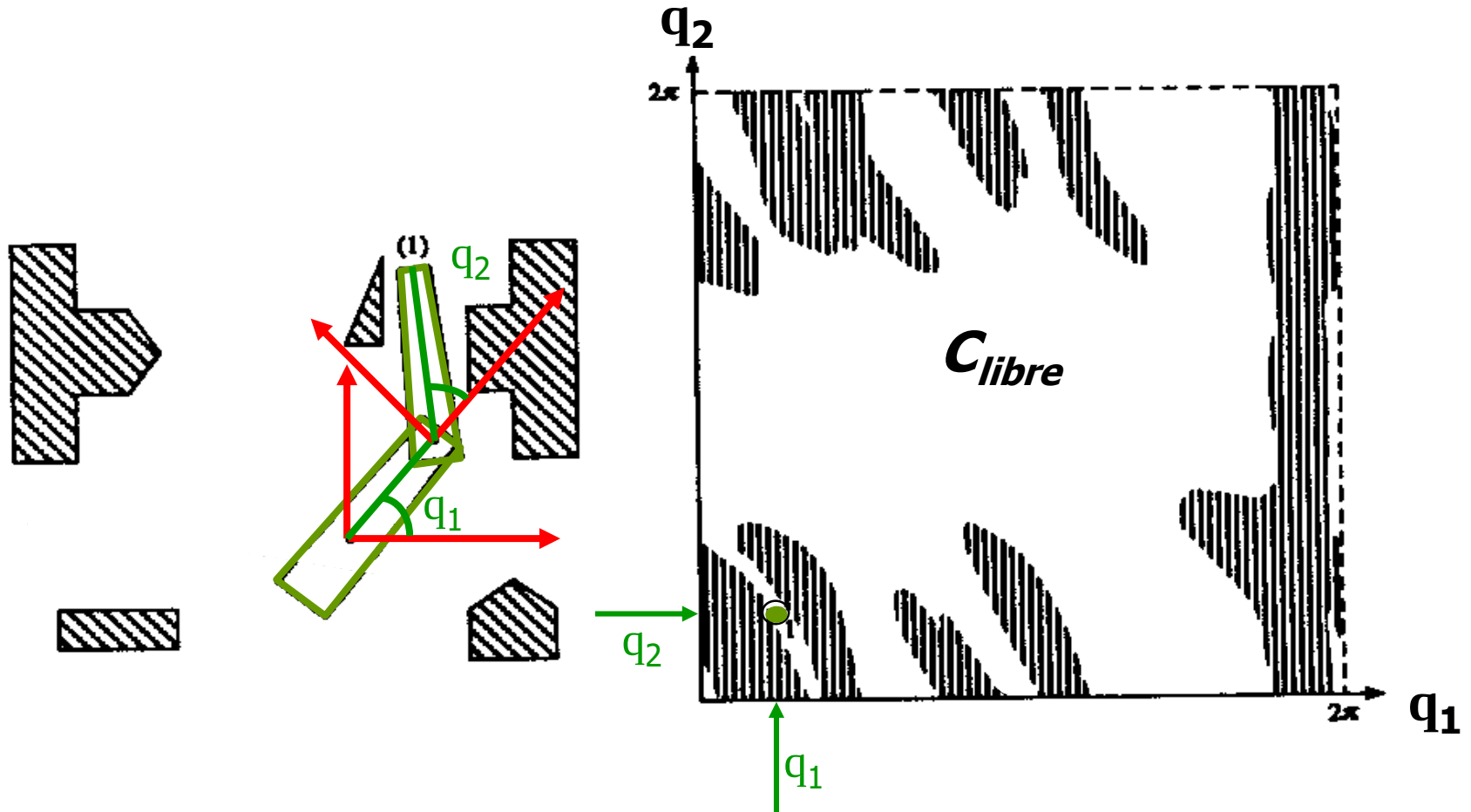


- Trouver une trajectoire τ continue dans C_{libre} entre q_1 et q_2 .

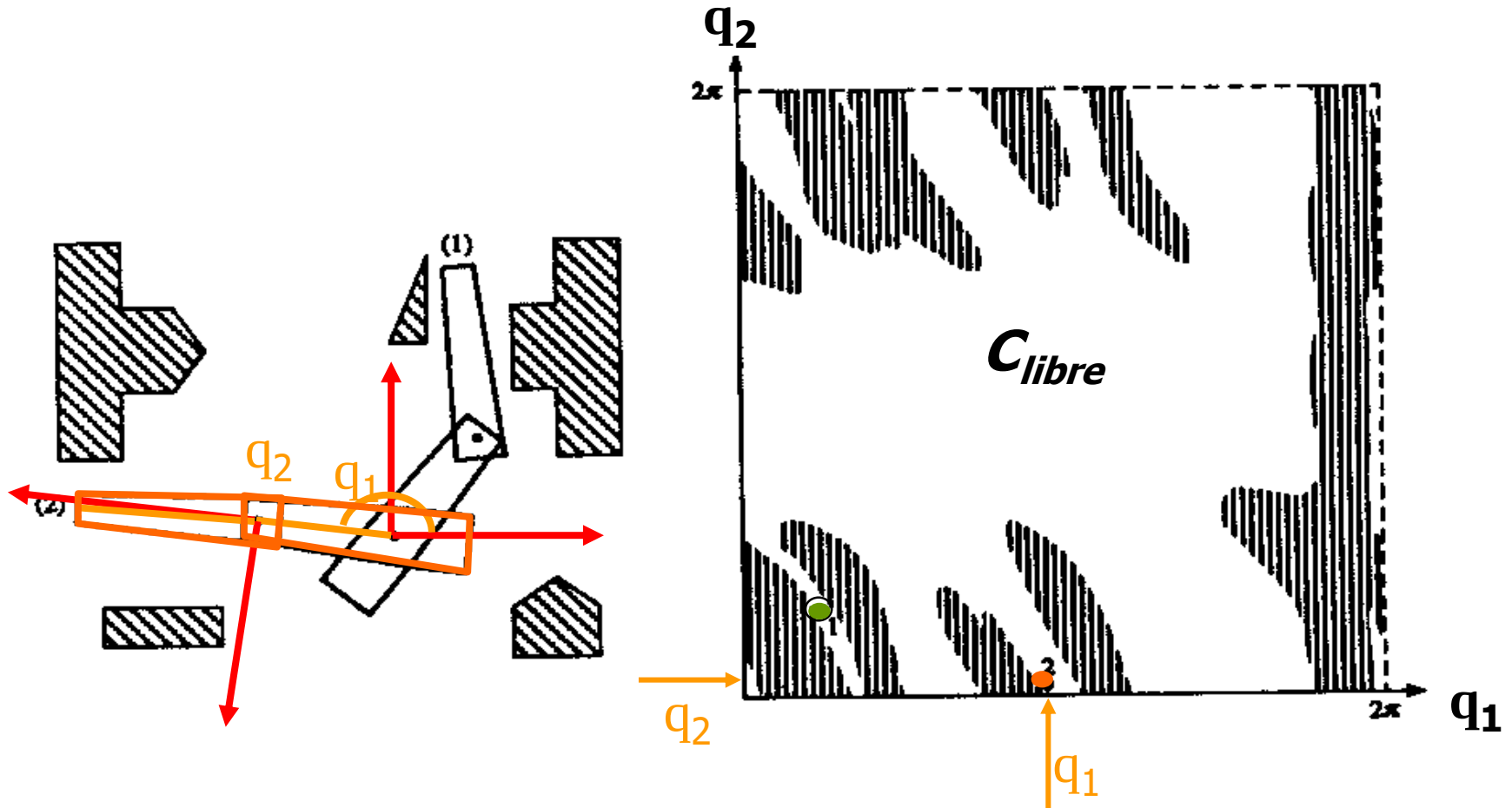
calcul exact approximativement
exponentiel en dimension de q

Espace de configuration : position 1

- Pour un bras avec 2 joints rotatifs q_1 et q_2

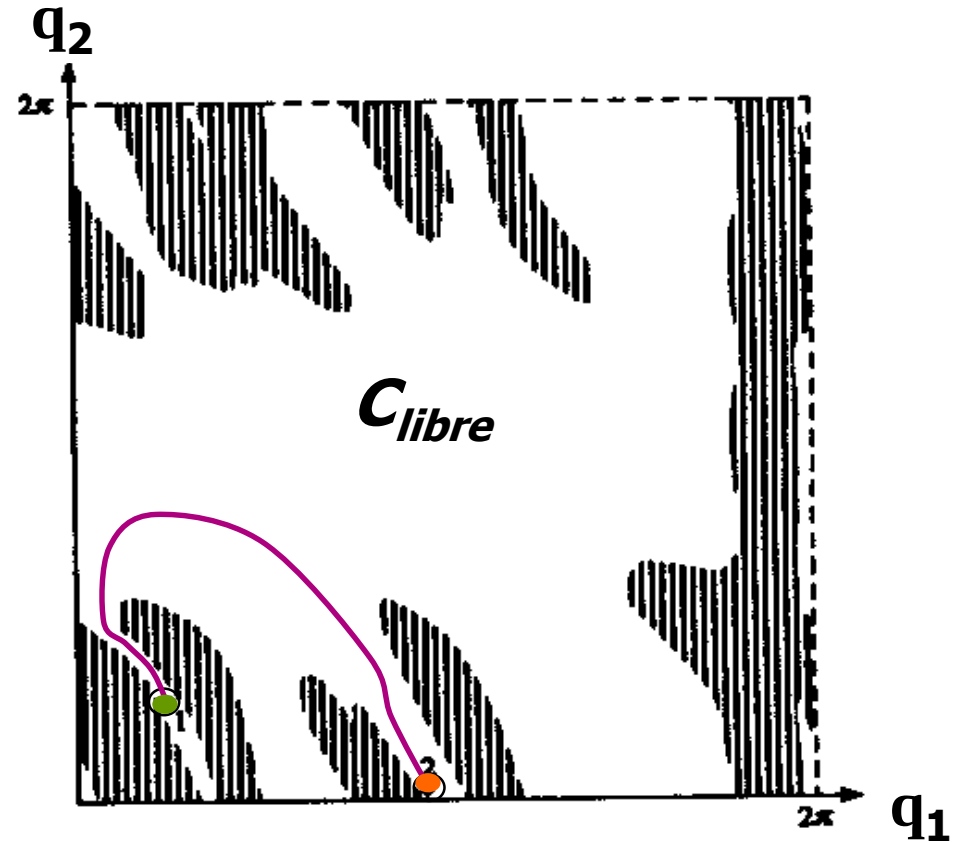
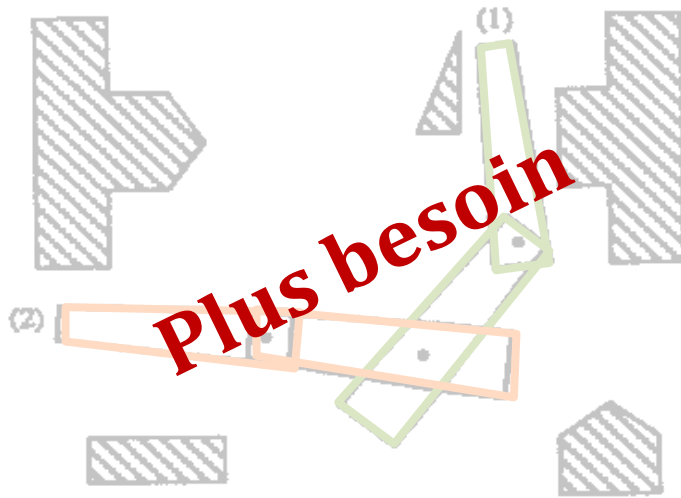


Espace de configuration : position 2



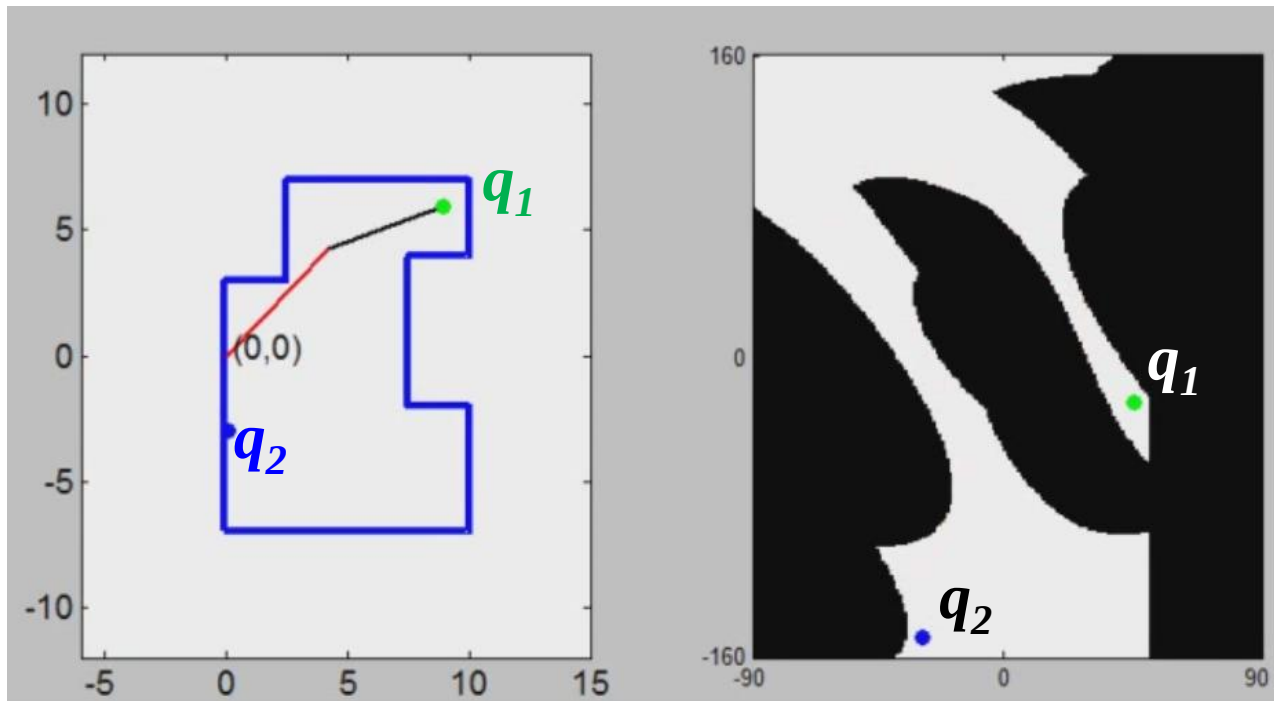
Espace de configuration

- Trajectoire possible entre q_1 et q_2 ? **oui!**

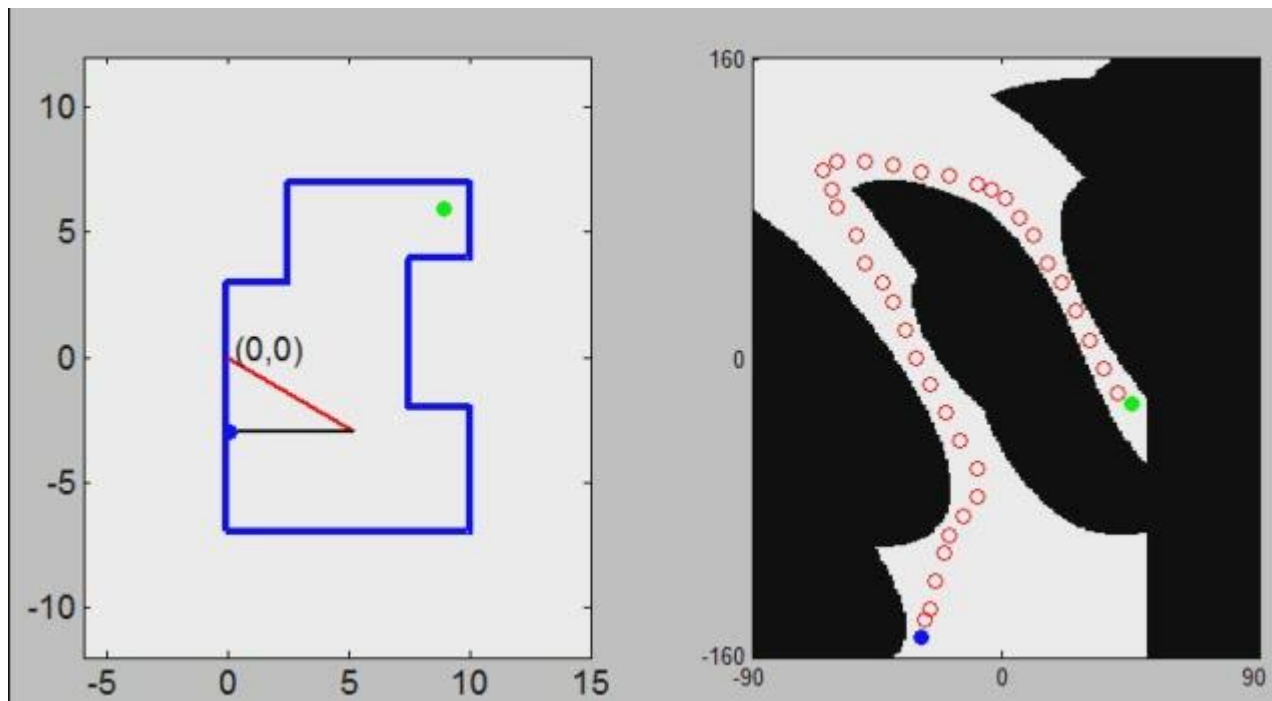


Exemple : bras robotisé

On veut déplacer le bout du bras (*end effector*) de q_1 à q_2 .



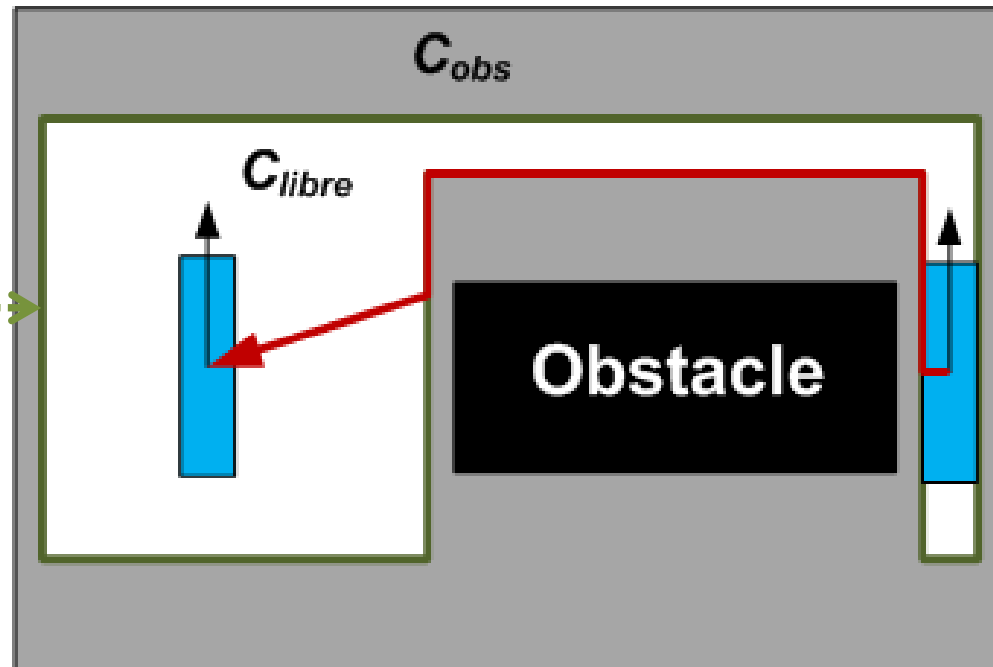
Exemple : bras robotisé



Configuration semi-libre

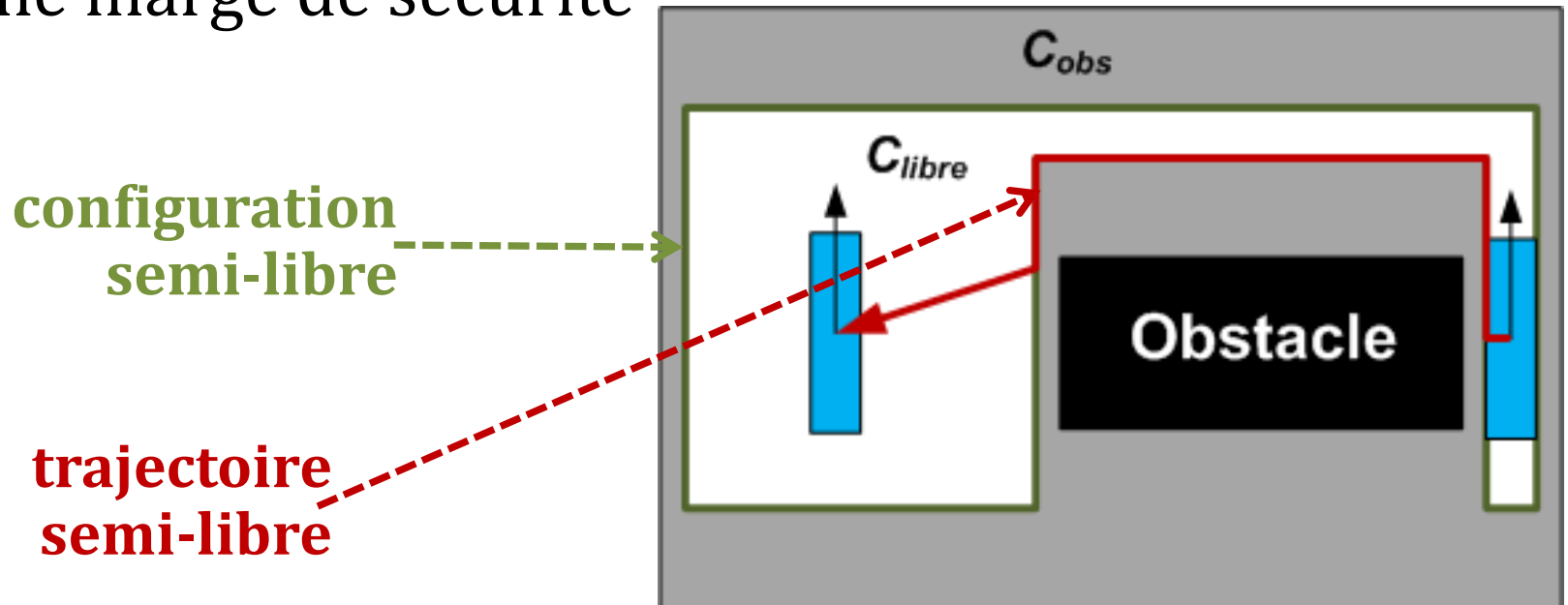
- Une configuration est semi-libre si le robot touche un obstacle, sans le pénétrer.
- Les configurations semi-libres sont à la bordure entre C_{obs} et C_{libre} .

configuration
semi-libre



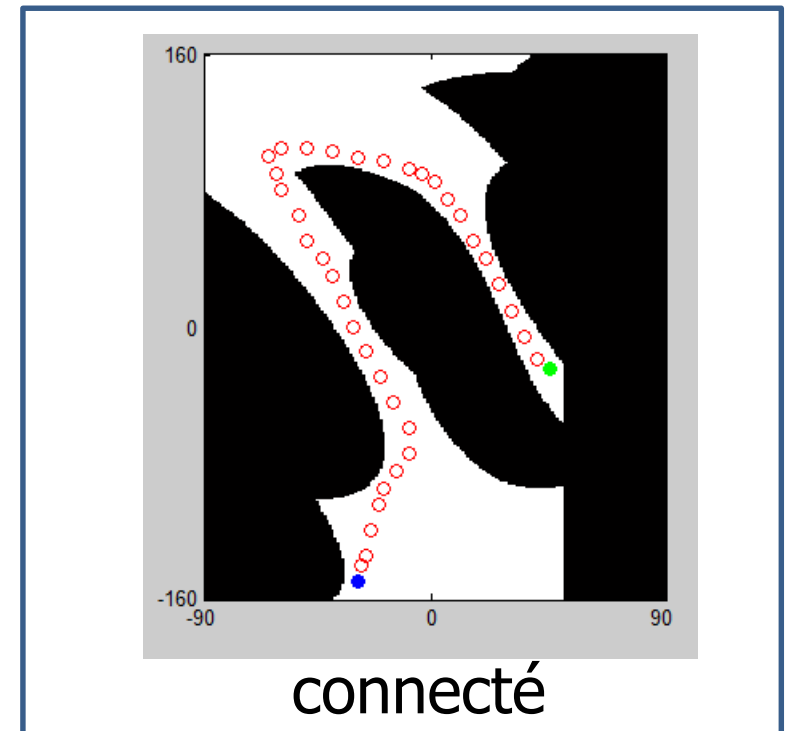
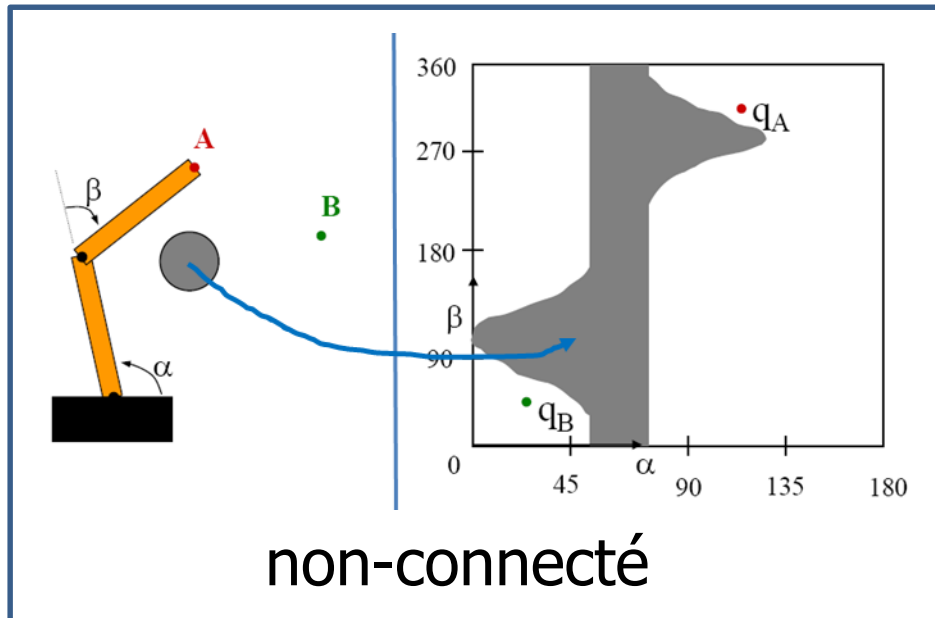
Trajectoire semi-libre

- Trajectoire est libre si entièrement dans C_{libre} .
- Trajectoire est semi-libre si elle contient au moins une configuration semi-libre.
- On cherche, en général, à éviter ces configurations : aucune marge de sécurité



Connectivité de l'espace-C

- L'espace de configuration $\mathcal{C}$ est connecté s'il existe une trajectoire entre n'importe quelle paire de configuration q_1 et q_2 dans $\mathcal{C}_{libre}$



Planification de mouvements

- Recherche à trouver une trajectoire dans l'espace des configurations, sans collision $\rightarrow C_{libre}$
- **Méthodes déterministes**
 - à chaque fois, l'algorithme retourne la même réponse
- **Probabilistes**
 - fonctionne par échantillonnage au hasard
 - ne trouve pas toujours la même trajectoire

Méthodes Déterministes

- Par discrétisation :
 - Graphe des visibilités
 - Décomposition cellulaire verticale
 - ~~Diagrammes de Voronoï~~
- ~~Algorithmes *bug0*, *bug1* et *bug2*~~
- Champs de potentiels
- Propagation de front d'ondes

Solutions exactes vs approximatives

- Un algorithme **exact** :
 - trouvera une solution si elle existe
 - **algorithme complet**
 - temps de calcul trop important pour des problèmes intéressants (e.g. humanoïdes à 30 joints).
- Un algorithme **approximatif** :
 - n'assure pas de trouver la solution...
 - ...mais est plus rapide.
 - *e.g.* grille discrète des configurations
 - (angle multiples de 5°, par exemple)

Discrétisation de l'espace

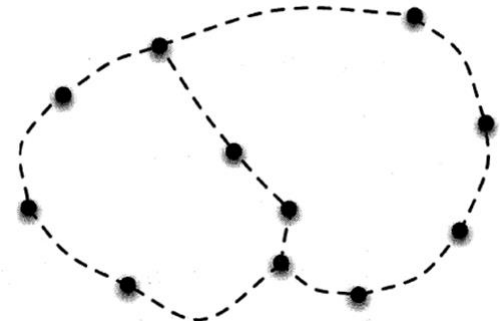
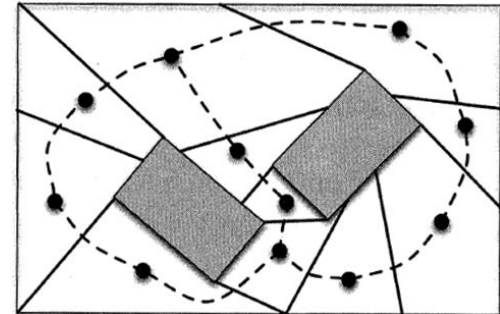
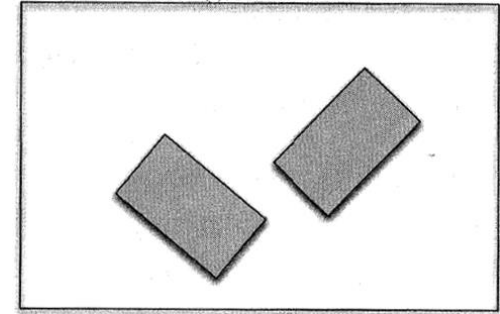
Représentation continue
(espace des configurations)



Discrétisation
+ graphe

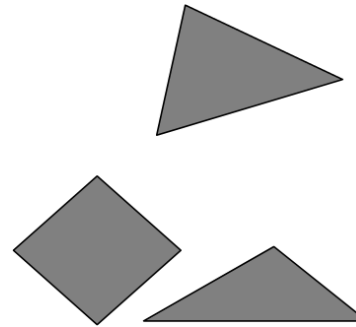


Recherche dans graphe
(blind, best-first, A*)

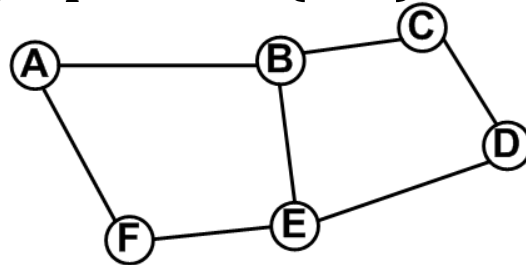


Graphe des visibilités

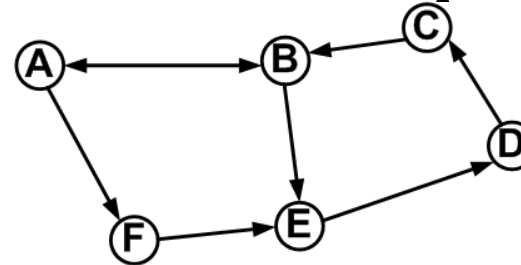
- Vous avez une carte géométrique connue
- Obstacles sont des **polygones**



- Utilise graphe $G = (E, V)$ non-orienté comme représentation



graphe non-orienté



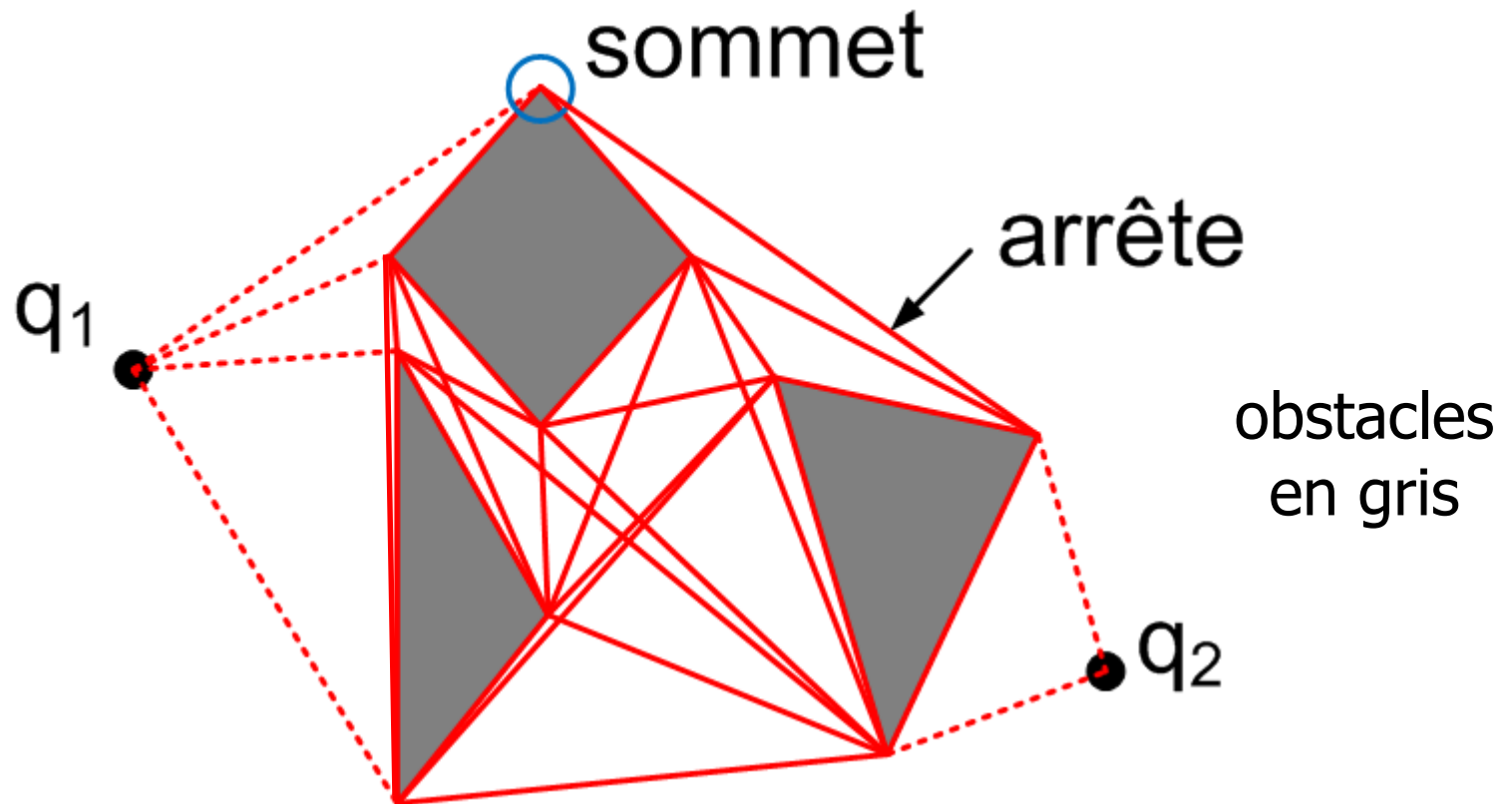
graphe orienté

- **Algorithme complet***
- Approprié pour monde 2D
- Approche par « carte routière » : *roadmap*

*complet : qui retourne une solution, si elle existe

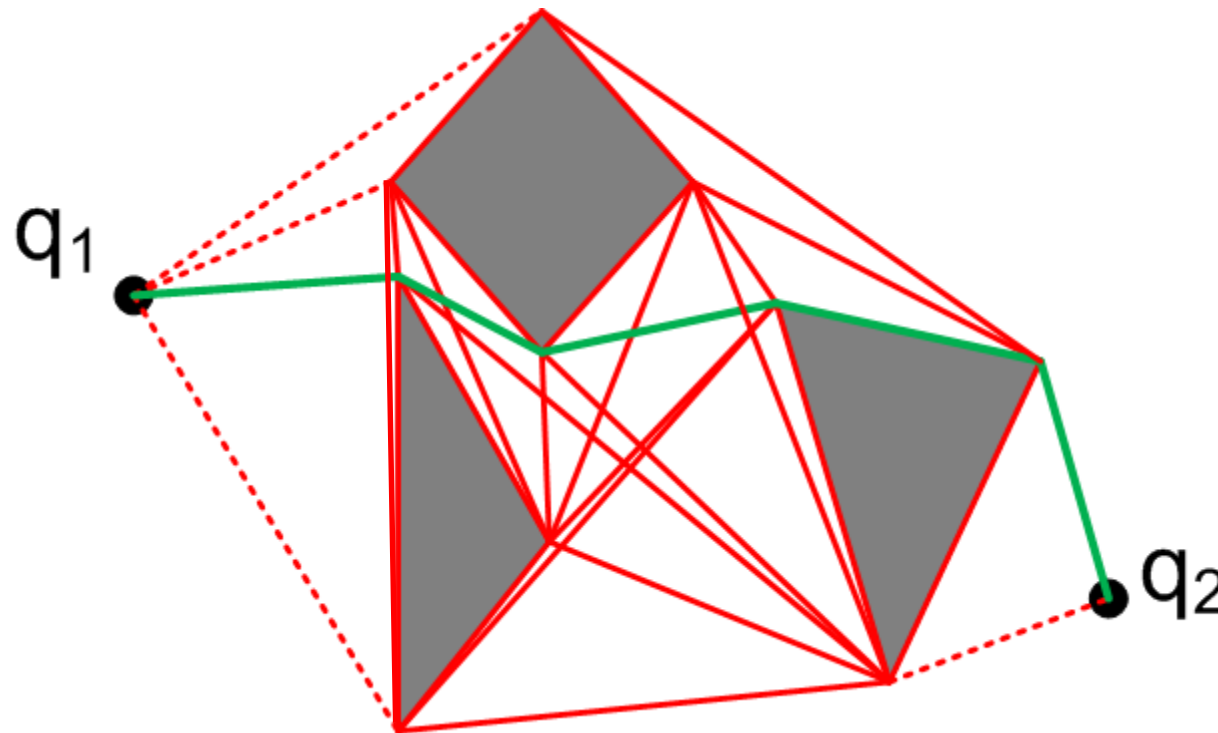
Discrétisation : graphe des visibilités

- Les sommets V des obstacles qui sont mutuellement visibles sont connectés ensemble par une arête E .



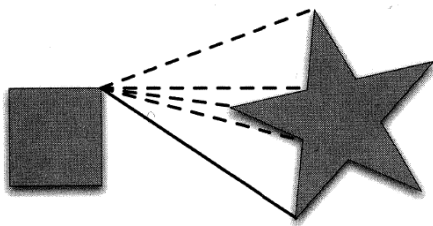
Planification : graphe des visibilité

- Planification : recherche le chemin le plus court dans graphe G entre q_1 et q_2 .
- Algorithme de *Dijkstra* : complexité $O(|E| + |V| \log |V|)$

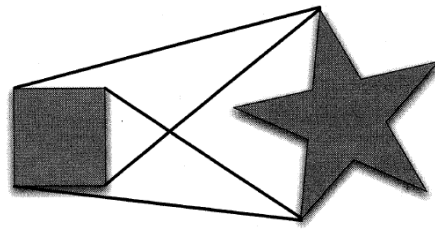


Graphe des visibilité réduites

- Une ligne passant par deux sommets $V_1 \in \mathbf{A}$ et $V_2 \in \mathbf{B}$ est dite tangente si elle ne passe pas à l'intérieur d'un des deux obstacles $\mathbf{A}$ ou $\mathbf{B}$.
- Vient réduire le nombre d'arrête E dans graphe G .

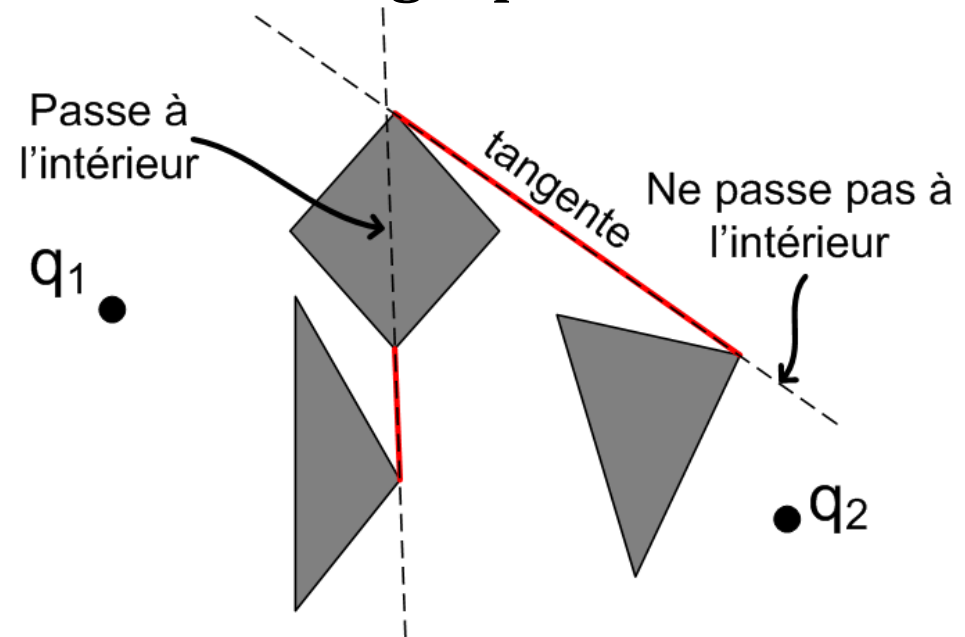


(a) Co-tangent lines from one vertex



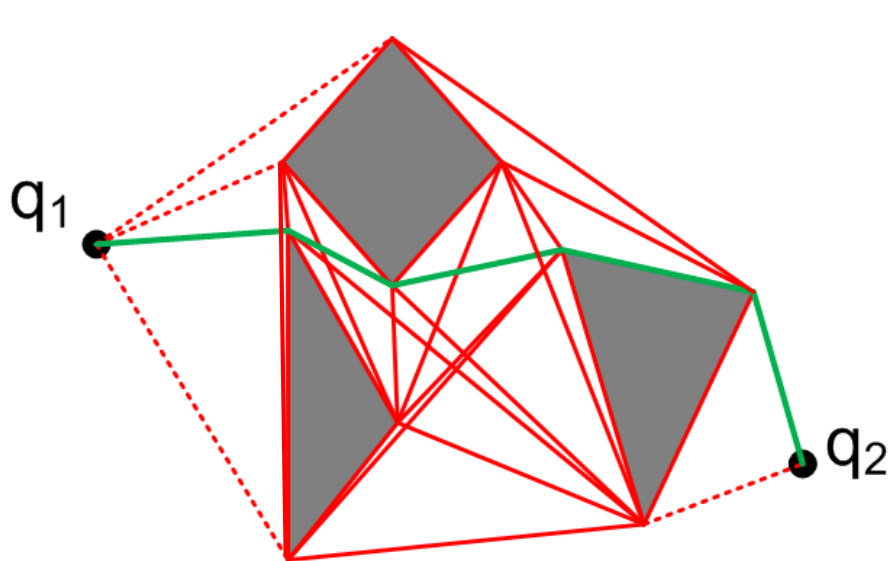
Co-tangent lines between two objects

Au maximum, 4 tangentes par
paire d'obstacles

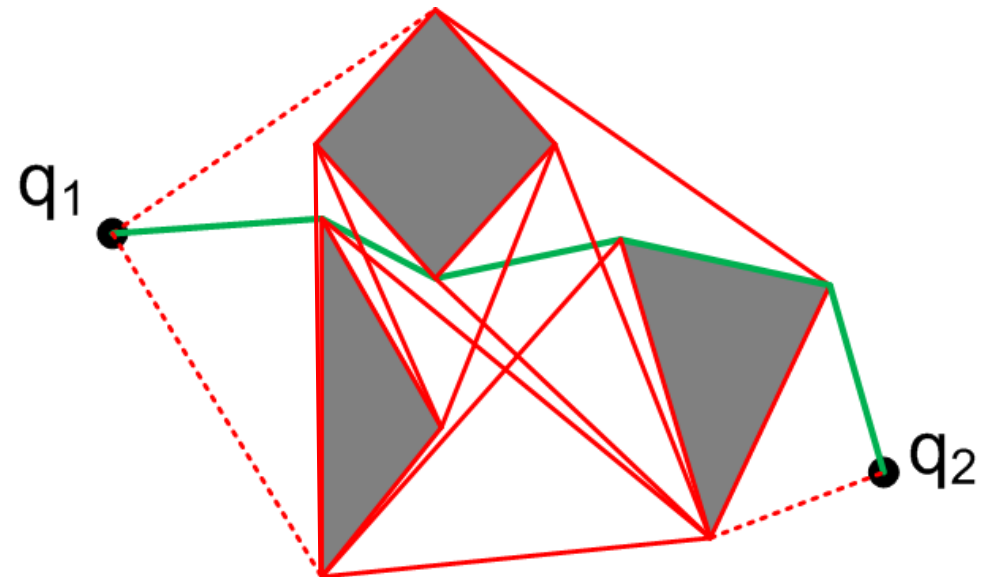


Graphe des visibilité réduites

- Graphe plus simple = recherche plus rapide
- Donne toujours la solution la plus courte



graphe des visibilités



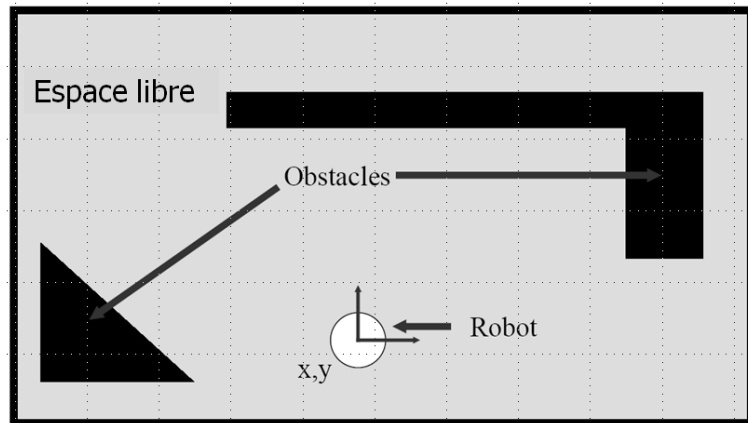
graphe des visibilités réduites

Graphe des visibilités

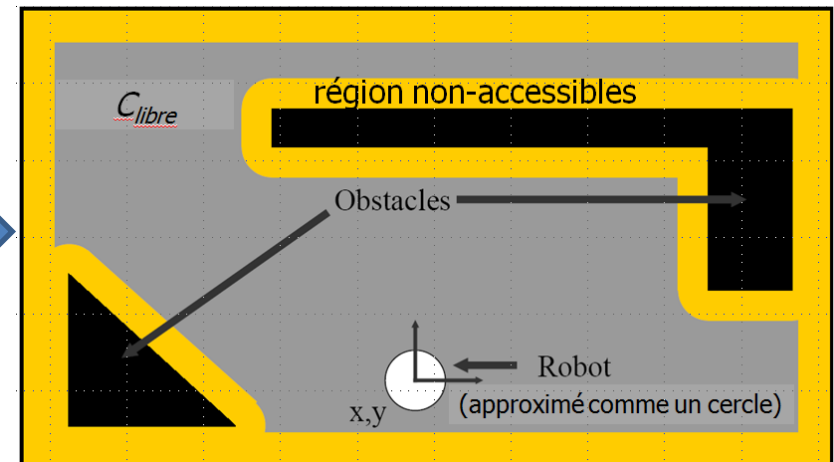
- Produit nécessairement la trajectoire la plus courte entre q_1 et q_2 ...
- ...mais avec possiblement des trajectoires semi-libres...
- ...et pour un robot de la taille d'un point.

Somme de Minkowski

- Somme de Minkowski :
 - « gonfle » les obstacles pour tenir compte taille robot
 - robot redevient un point
- Algo complet seulement si le robot est circulaire



minkowski

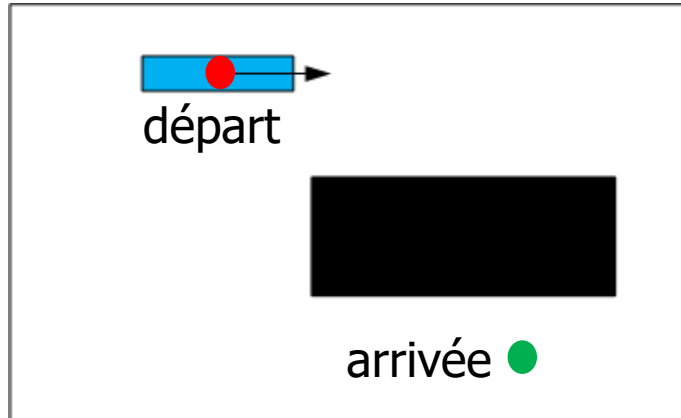


Somme de Minkowski

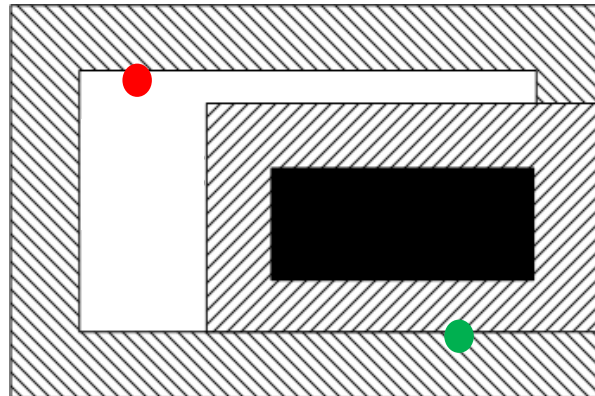
- Aucune garantie d'être complet si non-circulaire



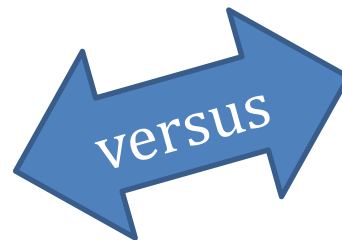
Somme de Minkowski



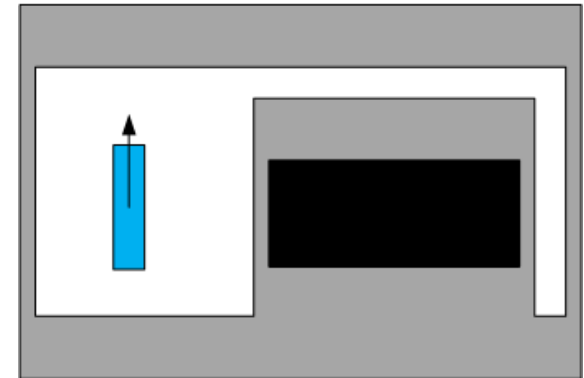
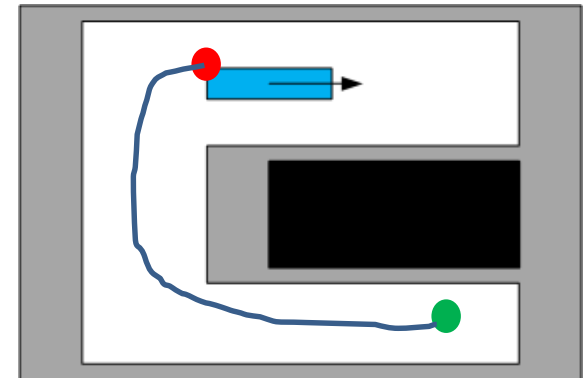
NON-COMPLET!



Minkowski : pas de solution

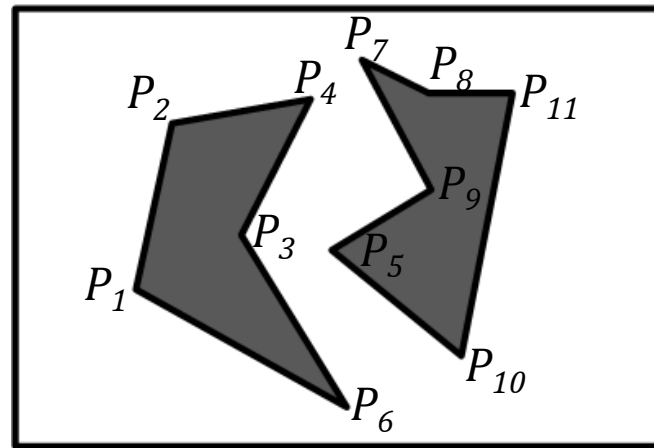


Espace de configuration
en 3 D: solution possible



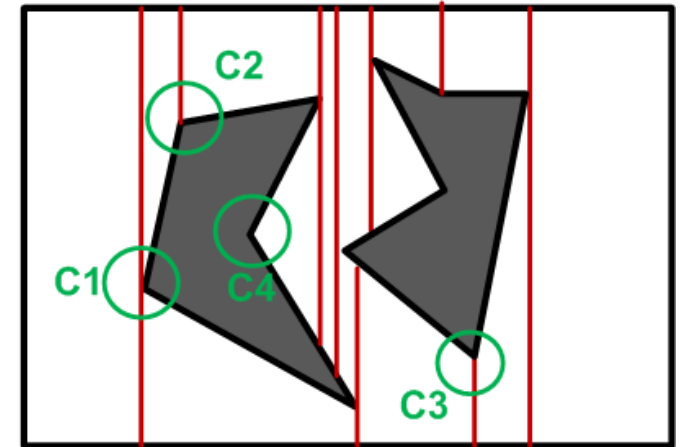
Décomposition cellulaire verticale

- Décomposer une carte 2D en trapèzes et triangles (éléments convexes),
- Pour chaque sommet P_i des obstacles:
 - étend **ligne verticale** en haut et en bas, jusqu'à un obstacle
- 4 cas possibles:
 - **C1**:(haut,bas)
 - **C2**:(haut),
 - **C3**:(bas),
 - **C4**:(aucun)

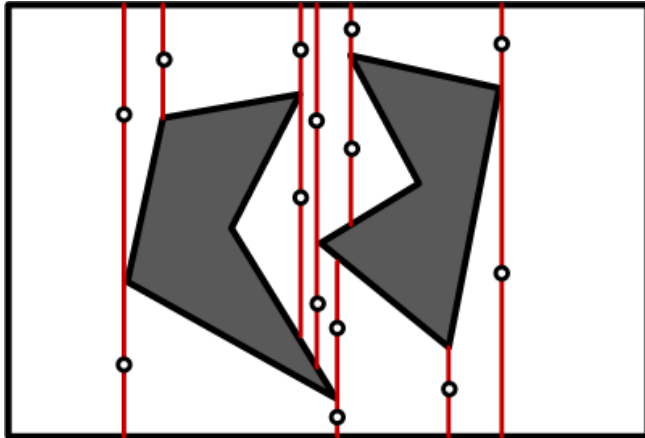


—balayage→

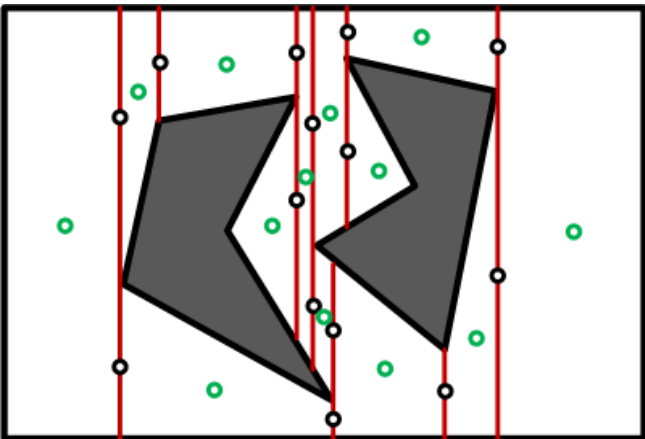
Étape 1



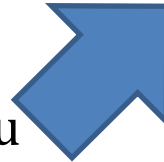
Décomposition cellulaire verticale



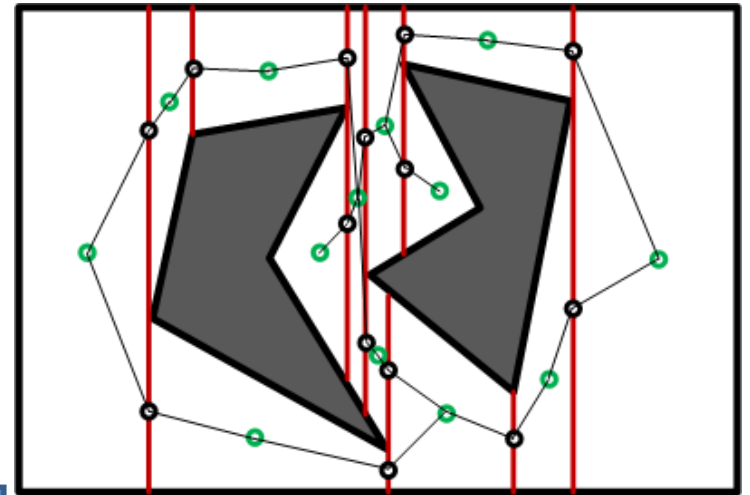
Étape 2:
place 1 point au
milieu frontière
entre cellules



Étape 3:
place 1 **point** au
milieu de la
cellules



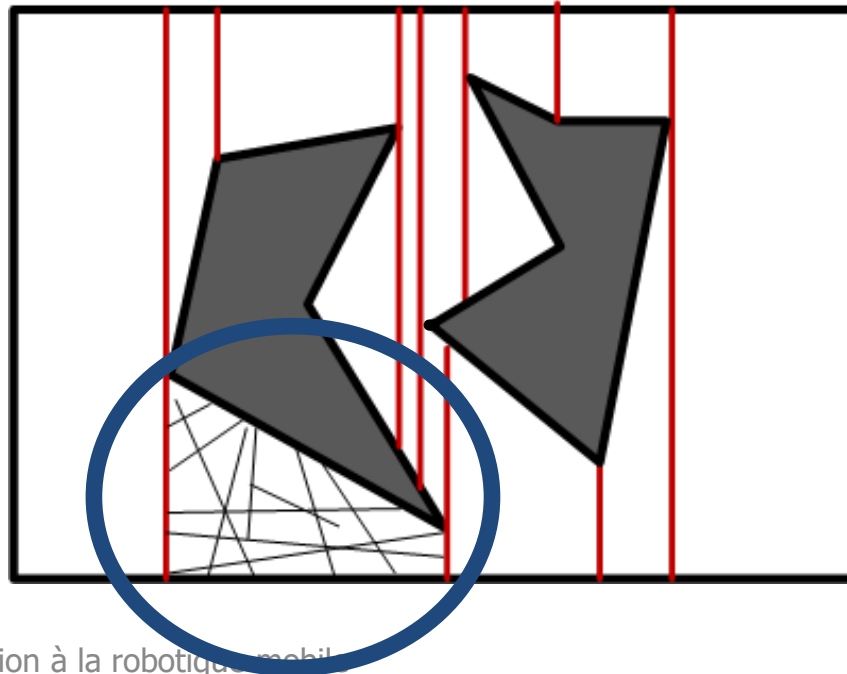
- Créer une
carte routière



Étape 4:
relie les **centres**
aux points
frontières

Décomposition cellulaire verticale

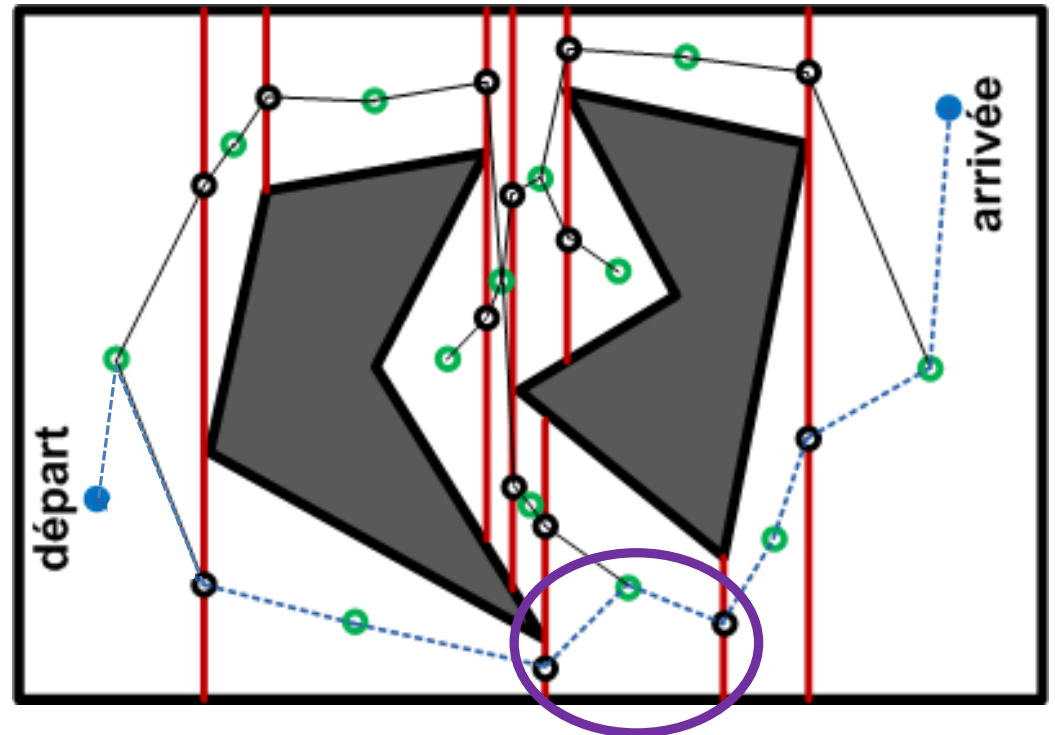
- Chaque cellule est convexe
 - ce qui veut dire que dans une cellule, on peut rejoindre n'importe quel autre point à l'intérieur, en ligne droite.



Décomposition cellulaire verticale

- Planification :
recherche du chemin
le plus court dans le
graphe
- Donne des trajectoires
qui sont en général
loin des obstacles
- Création de la carte :
complexité $O(n \log n)$

approche roadmap



**pas optimal en
distance**

Champs de potentiels

- Imaginez que :
 - **robot** est une particule **+**
 - **obstacles** sont aussi chargés **+** (répulsion)
 - **but** est chargé **-** (attraction)
- Trajectoire est le chemin parcouru par la particule (robot) dans le champs de potentiel

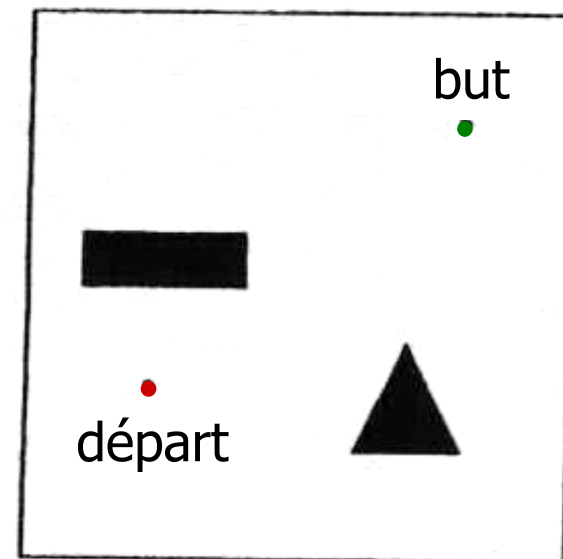
Champs de potentiels

Calcul des forces de répulsions et d'attraction

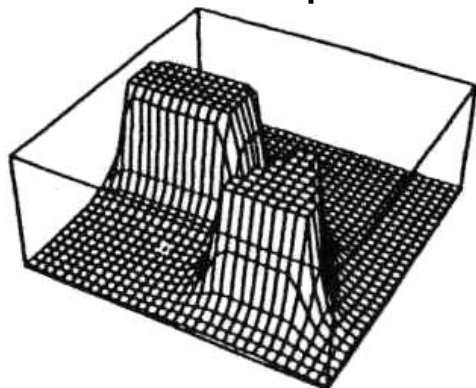
$$U(q) = U_{but}(q) + \sum U_{obstacles}(q)$$

$$F = -\nabla U(q) = \begin{bmatrix} \frac{\partial}{\partial x} U(q) \\ \frac{\partial}{\partial y} U(q) \end{bmatrix}$$

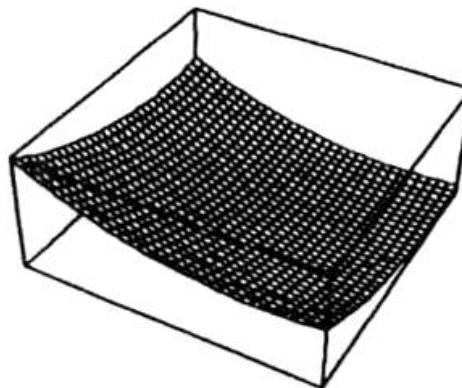
gradient ∇ indique la direction maximale de croissance d'un champ scalaire



potentiel de répulsion

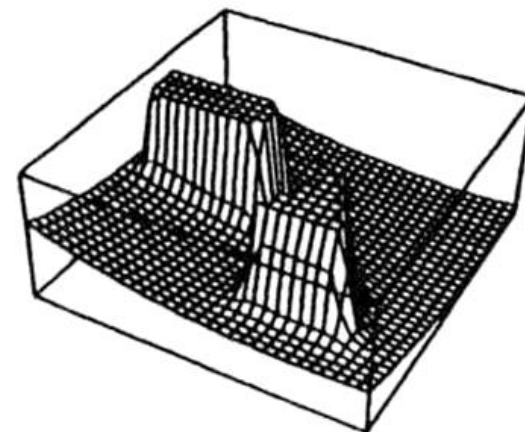


potentiel d'attraction



+

=



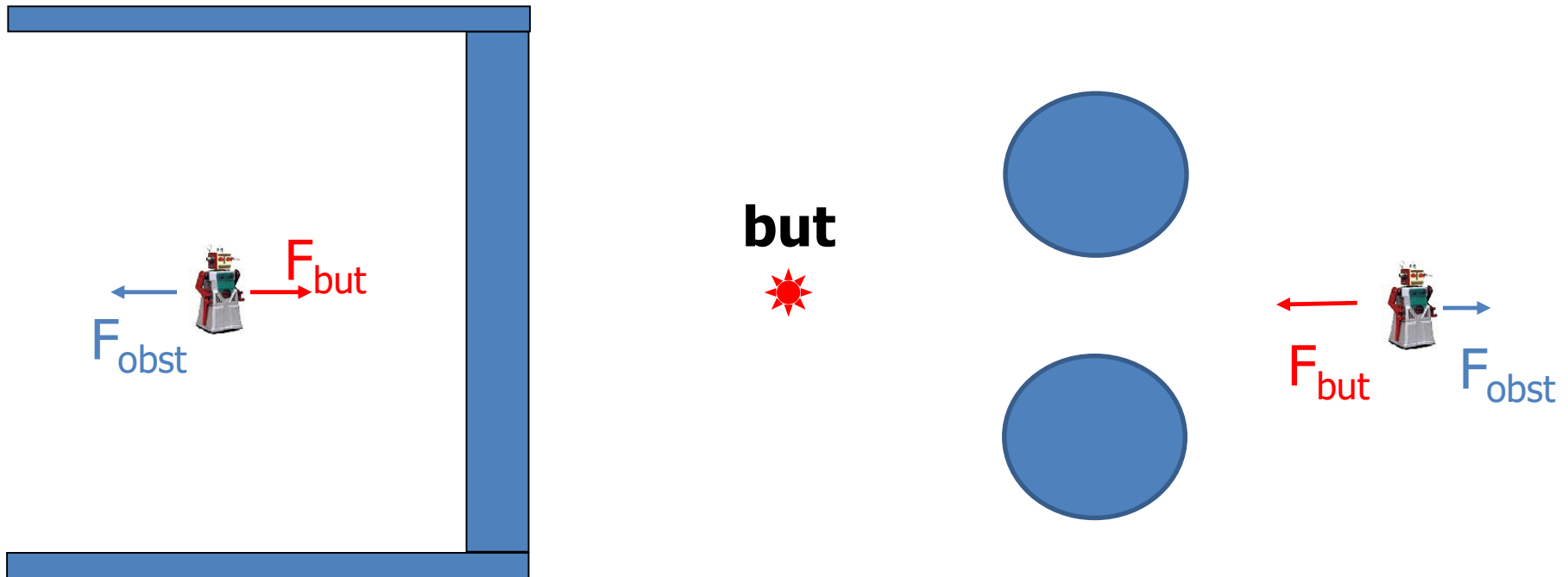
$$\sum U_{obstacles}(q)$$

$$U_{but}(q)$$

$$U(q)$$

Champs de potentiels

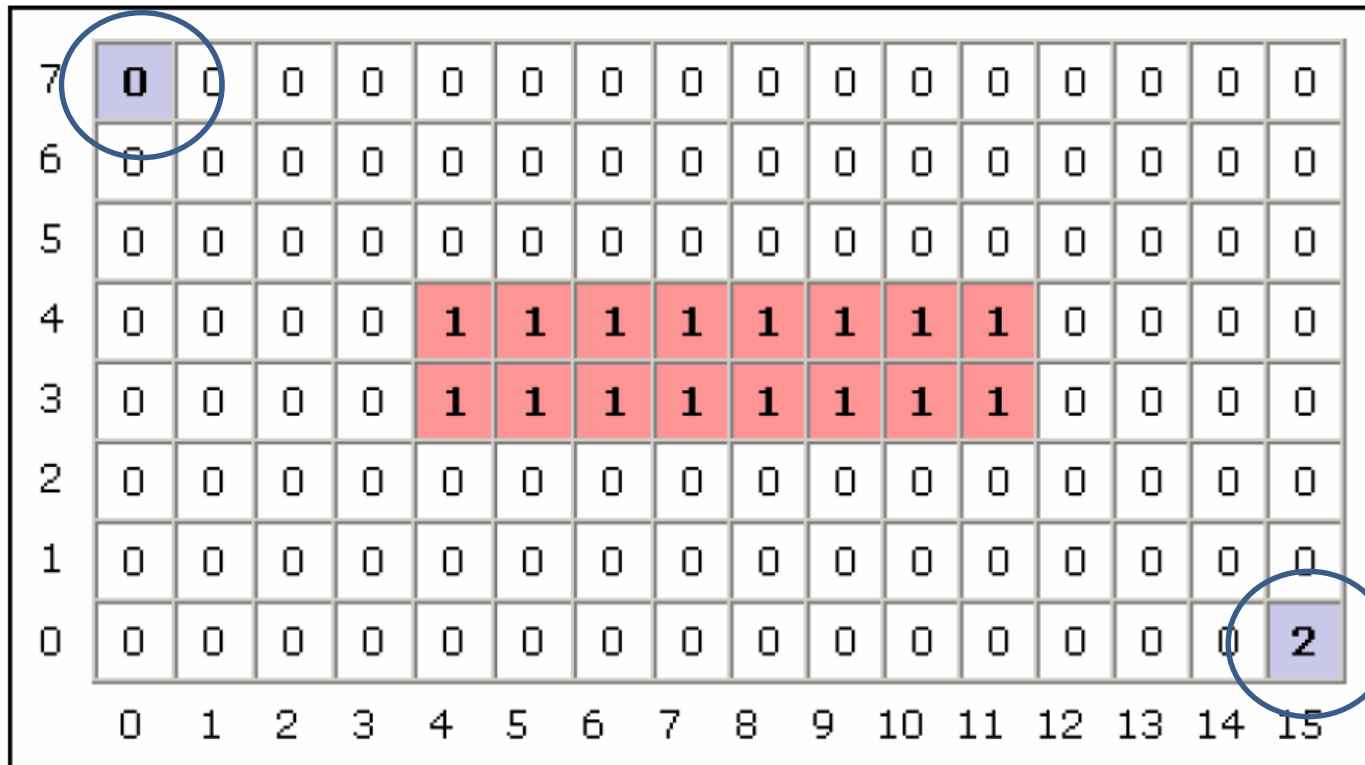
- Tombé en partie en désuétude parce que sensible aux minimum locaux.



Planification par front d'onde

- *Wavefront planner*
- 0: non-visité 1: obstacle 2: indique le départ

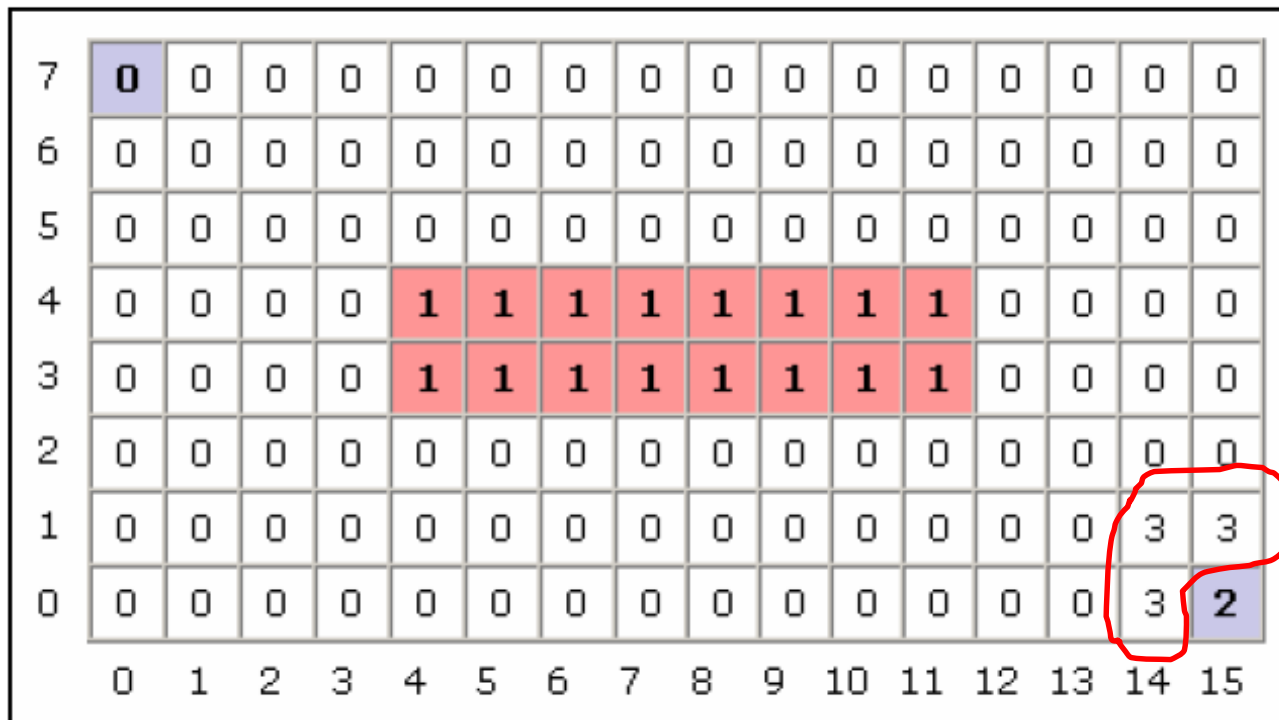
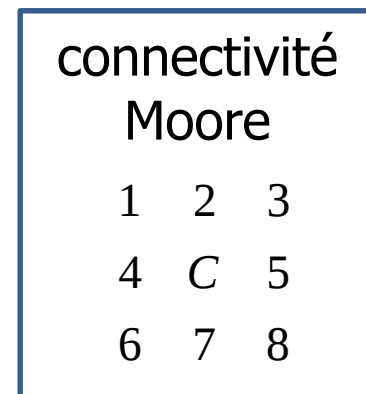
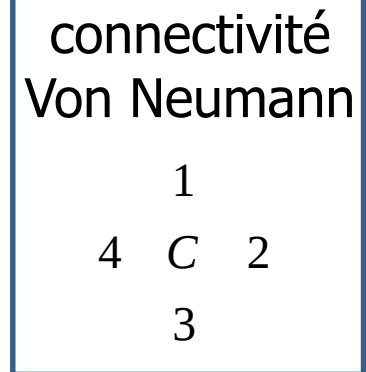
but



départ

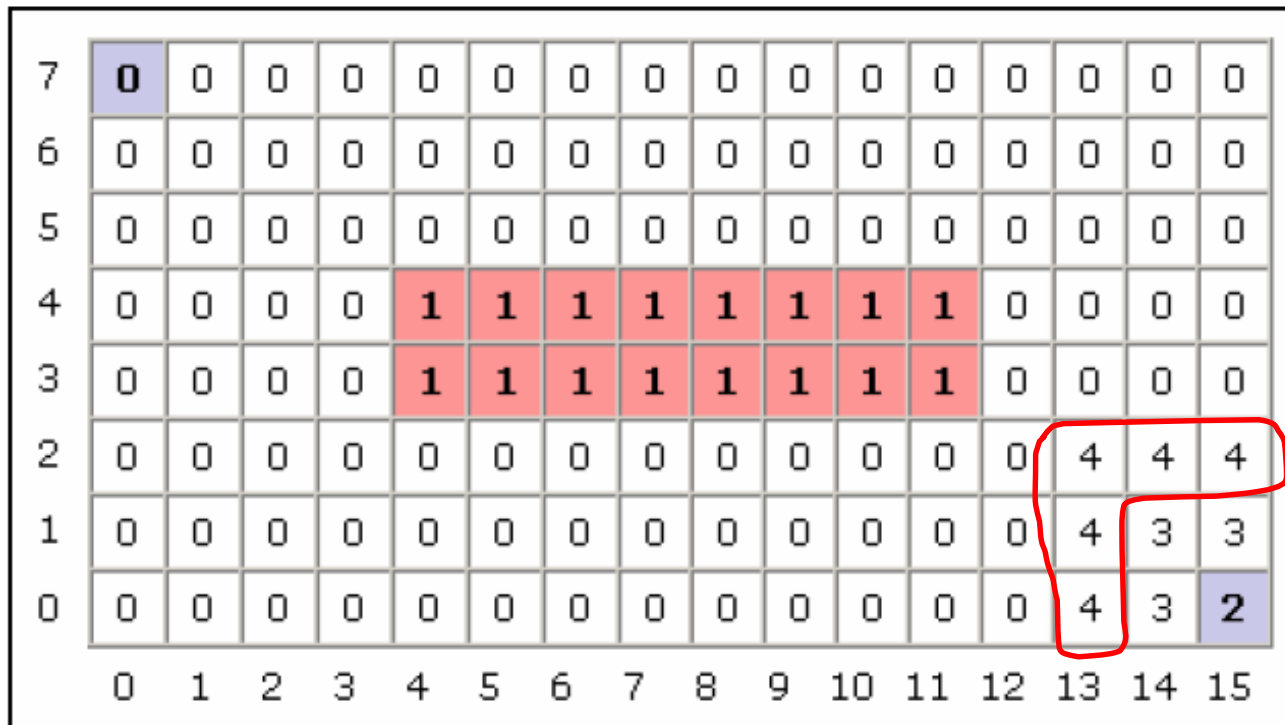
Planification par front d'onde

- *Wavefront planner*
- 0: non-visité 1: obstacle 2: indique le départ



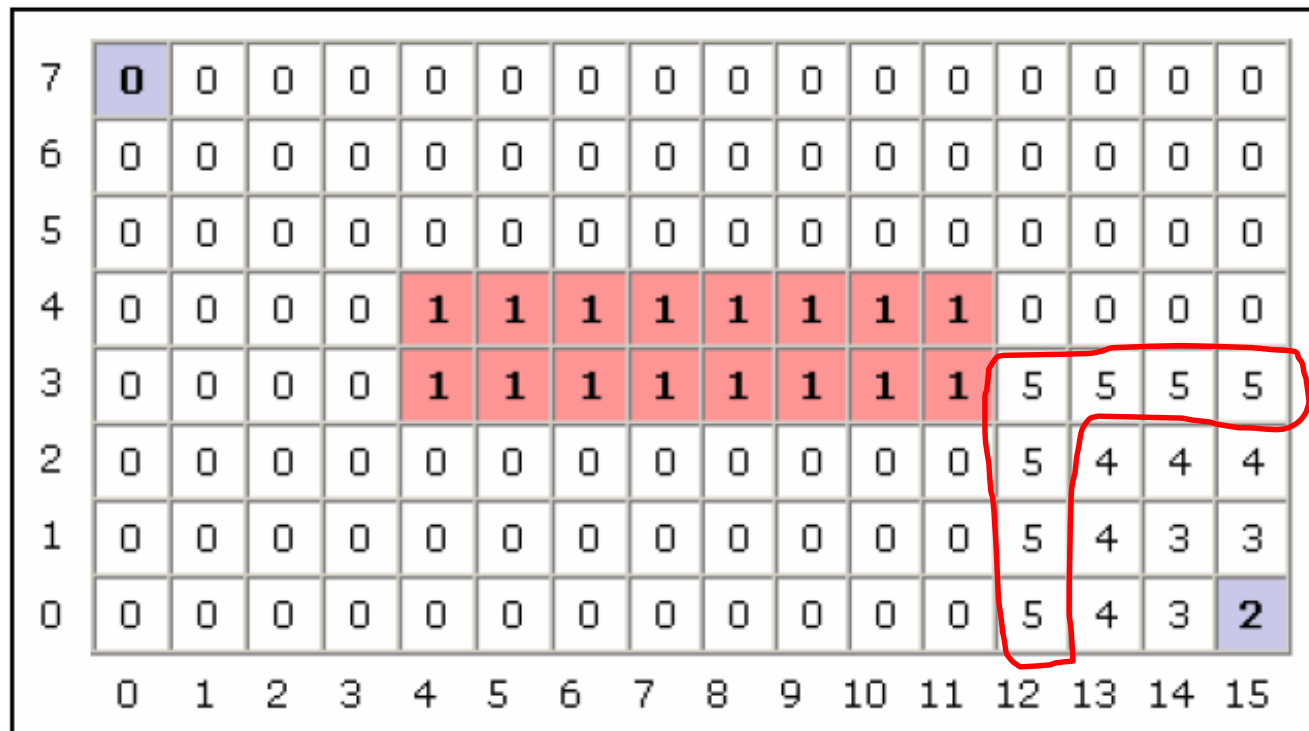
Planification par front d'onde

- *Wavefront planner*
- 0: non-visité 1: obstacle 2: indique le départ



Planification par front d'onde

- *Wavefront planner*
- 0: non-visité 1: obstacle 2: indique le départ



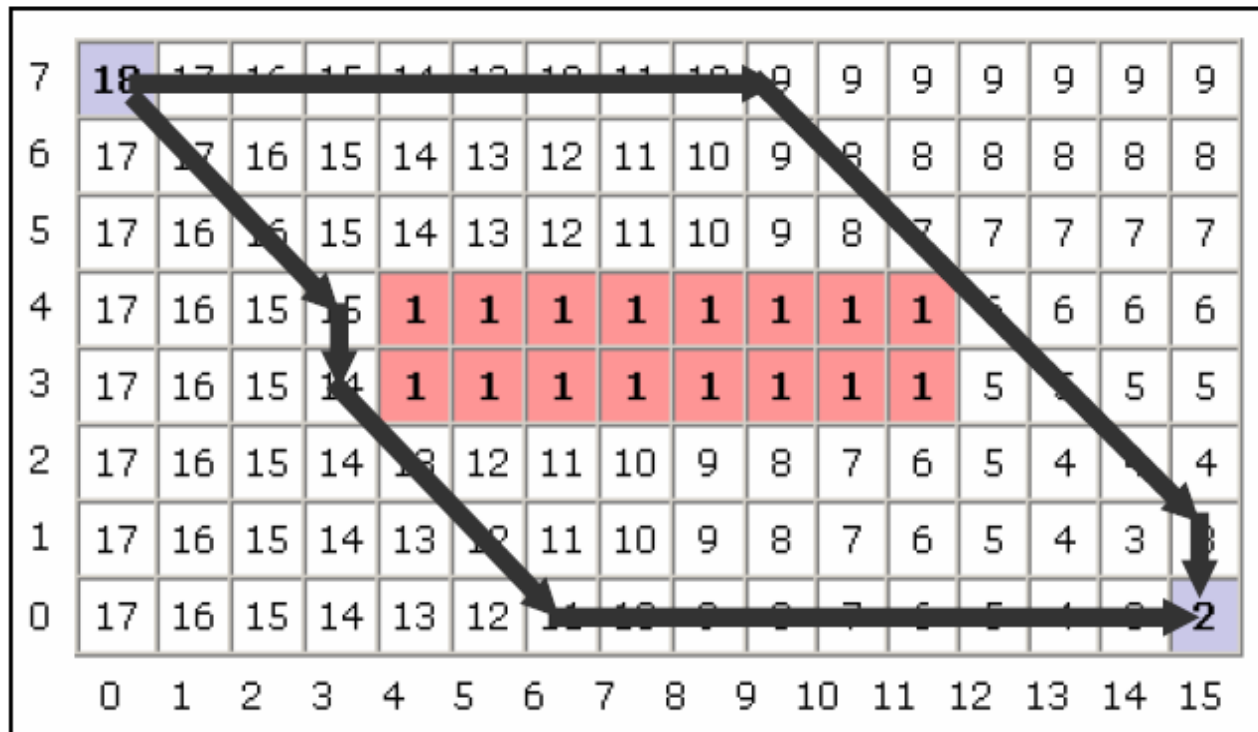
Planification par front d'onde

- Rempli jusqu'à plein
 - les 0 indiquent les endroits non-accessibles du point de départ.

7	18	17	16	15	14	13	12	11	10	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7
4	17	16	15	15	1	1	1	1	1	1	1	1	6	6	6
3	17	16	15	14	1	1	1	1	1	1	1	1	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Planification par front d'onde

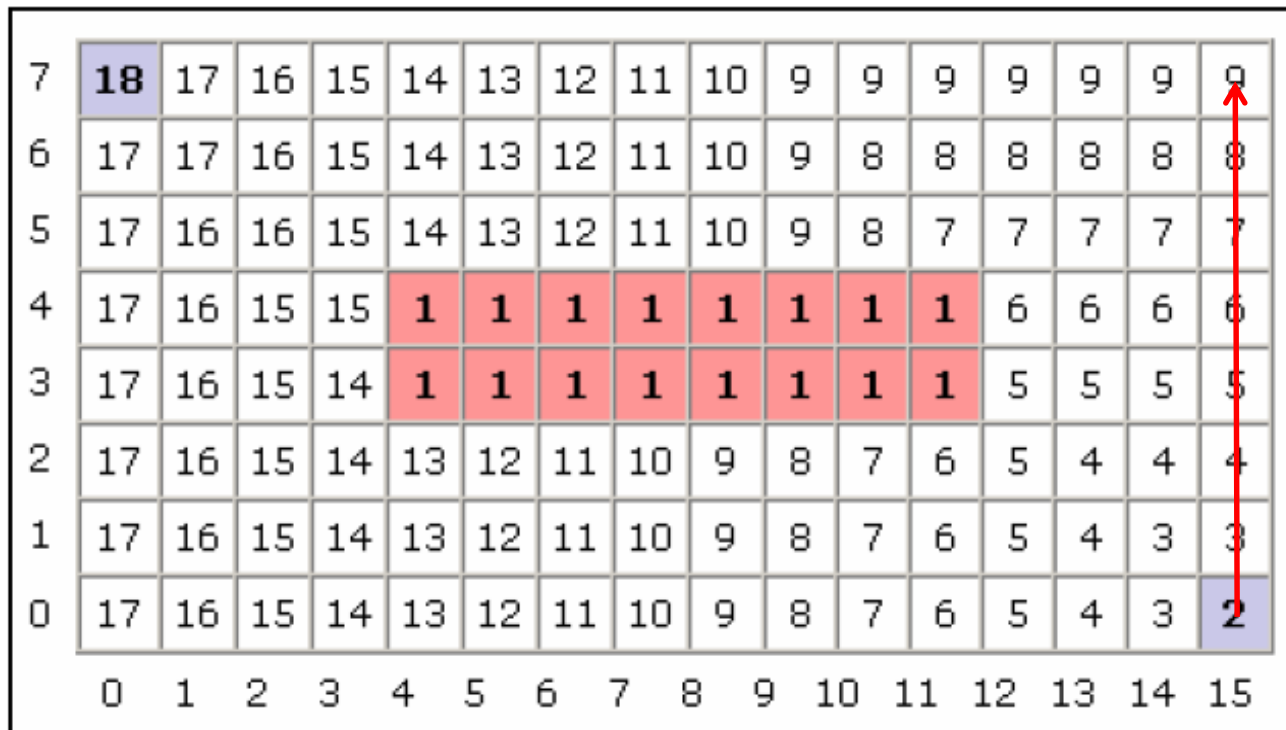
- La trajectoire consiste **à partir du but** de choisir de façon vorace les cases les plus petites.



2 des
nombreuses
trajectoires
possibles

Planification par front d'onde

- Si vous faites le choix vorace des nœuds **à partir du départ** ☹️, vous n'aurez pas nécessairement un chemin le plus court

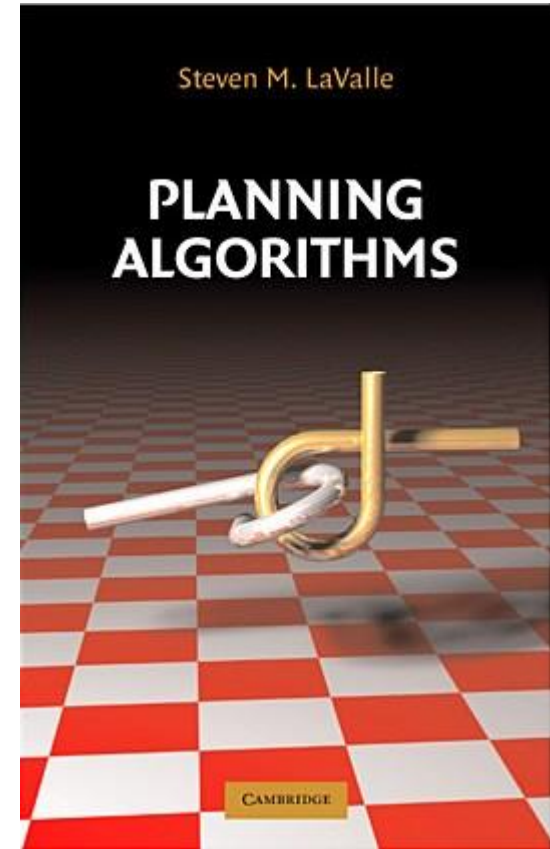


Méthodes probabilistes

- Basée sur l'échantillonnage au hasard de configurations dans $\mathcal{C}_{libre}$.
- Besoin d'une fonction de détection de collision
 - plus besoin de calculer le $\mathcal{C}_{libre}$ au complet
 - fonction très optimisée
- Approches
 - **Requête simple** : RRT-Connect
 - basée sur les Rapidely-exploring Random Trees (RRT)
 - **Requête multiple** : Probabilistic Road Maps (RPM)

Rapidly-exploring Random Trees (RRT)

- Pour recherche exploratoire dans des espaces à haute dimensionnalité
- Développé par Steven M. LaValle et James Kuffner
- Construire un arbre de façon incrémentale, pour minimiser la distance entre un point choisis au hasard et l'arbre



<http://planning.cs.uiuc.edu/>

Rapidly-exploring Random Trees (RRT)

```
for k = 1 to K  
   $q_{rand} \leftarrow \text{RAND\_CONF}()$   
   $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$   
   $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, \Delta q)$   
  G.add_vertex( $q_{new}$ )  
  G.add_edge( $q_{near}, q_{new}$ )  
end
```

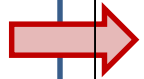
K : nombre d'itérations

G : graphe de l'arbre

- arbre au départ (1 seul sommet)

Rapidly-exploring Random Trees (RRT)

for $k = 1$ **to** K



$q_{\text{rand}} \leftarrow \text{RAND_CONF}()$

$q_{\text{near}} \leftarrow \text{NEAREST_VERTEX}(q_{\text{rand}}, G)$

$q_{\text{new}} \leftarrow \text{NEW_CONF}(q_{\text{near}}, \Delta q)$

$G.\text{add_vertex}(q_{\text{new}})$

$G.\text{add_edge}(q_{\text{near}}, q_{\text{new}})$

end

q_{rand}
 sommet tiré au hasard



Rapidly-exploring Random Trees (RRT)

```
for k = 1 to K  
   $q_{rand} \leftarrow \text{RAND\_CONF}()$   
   $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$   
   $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, \Delta q)$   
  G.add_vertex( $q_{new}$ )  
  G.add_edge( $q_{near}, q_{new}$ )  
end
```



 q_{near} sommet le plus proche de l'arbre G

Rapidly-exploring Random Trees (RRT)

```
for k = 1 to K
```

```
   $q_{rand} \leftarrow \text{RAND\_CONF}()$ 
```

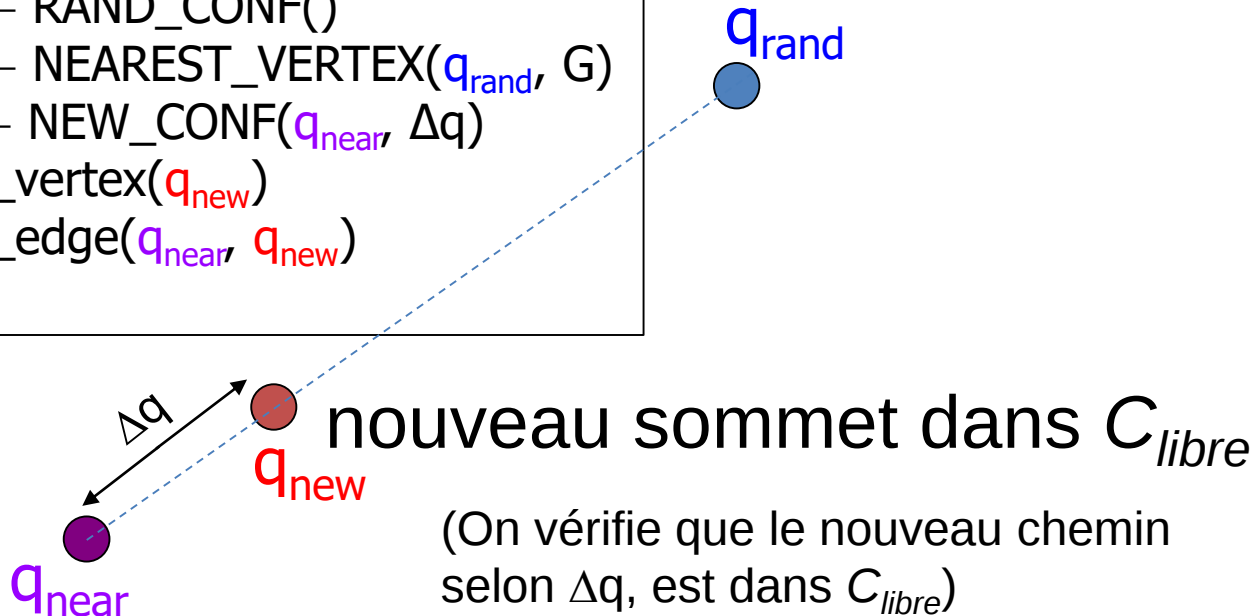
```
   $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$ 
```

```
   $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, \Delta q)$ 
```

```
  G.add_vertex( $q_{new}$ )
```

```
  G.add_edge( $q_{near}, q_{new}$ )
```

```
end
```



Rapidly-exploring Random Trees (RRT)

```
for k = 1 to K
```

```
   $q_{rand} \leftarrow \text{RAND\_CONF}()$ 
```

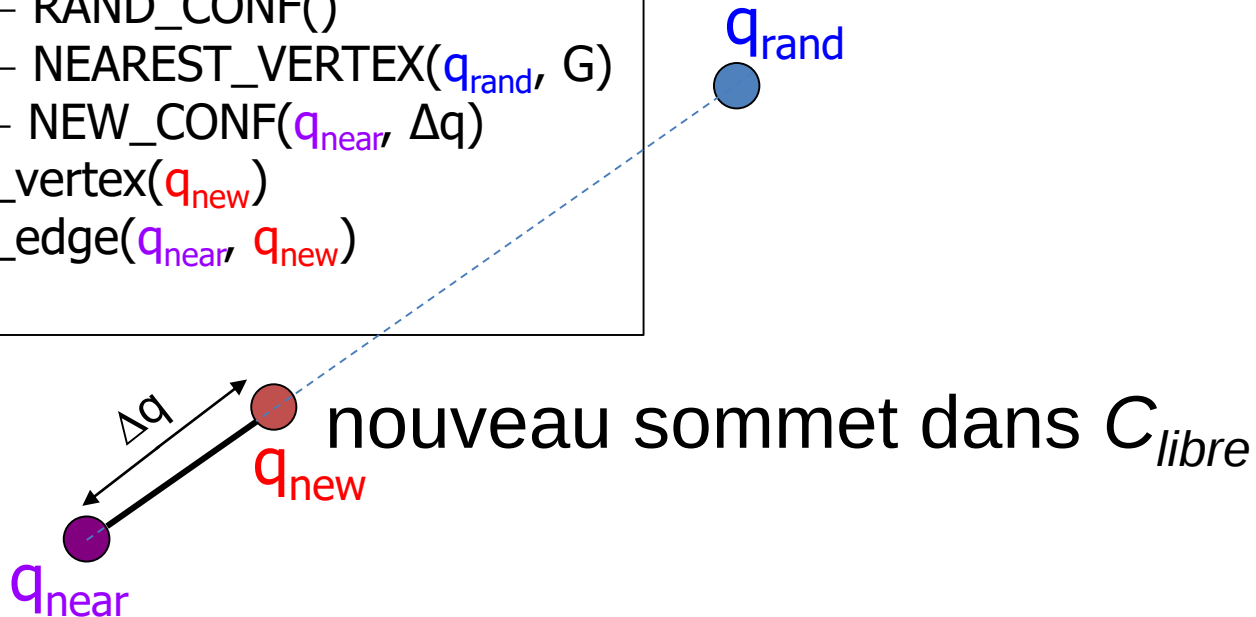
```
   $q_{near} \leftarrow \text{NEAREST\_VERTEX}(q_{rand}, G)$ 
```

```
   $q_{new} \leftarrow \text{NEW\_CONF}(q_{near}, \Delta q)$ 
```

```
  G.add_vertex( $q_{new}$ )
```

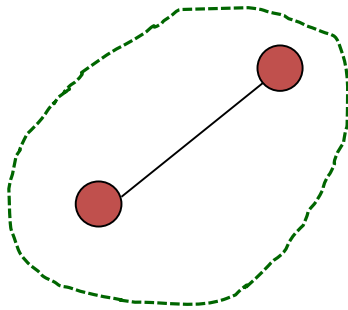
```
  G.add_edge( $q_{near}, q_{new}$ )
```

```
end
```



Rapidly-exploring Random Trees (RRT)

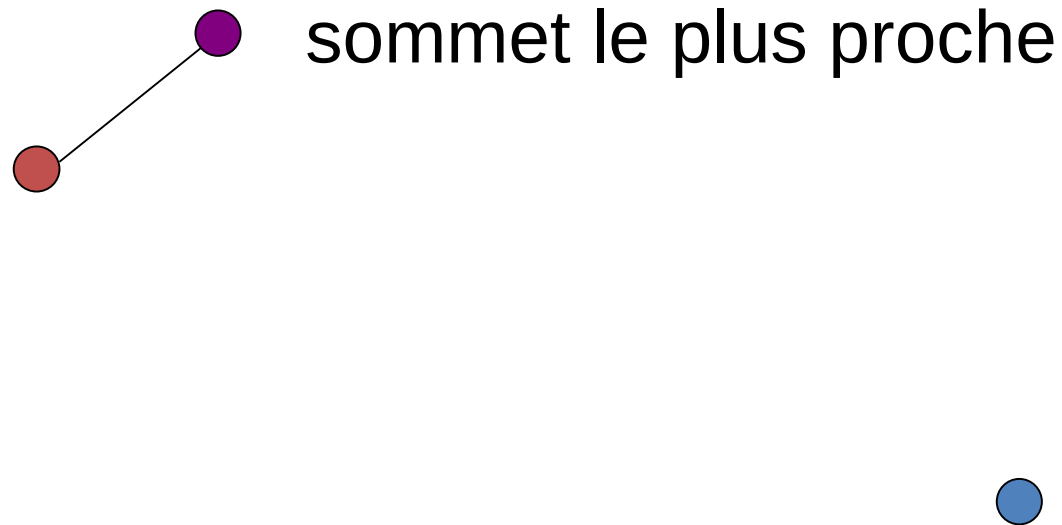
Arbre G



autre sommet tiré au hasard



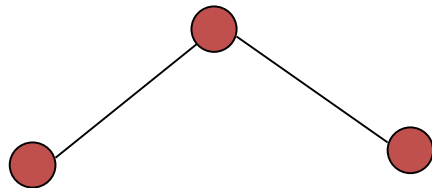
Rapidly-exploring Random Trees (RRT)



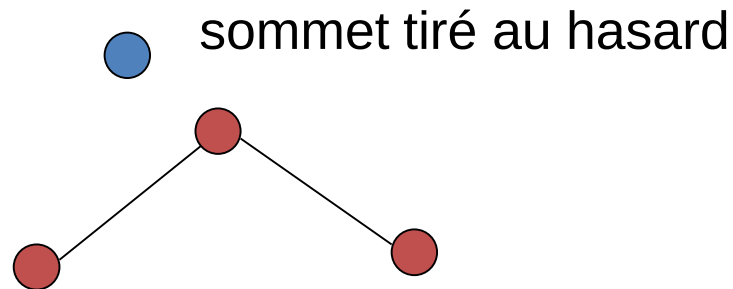
Rapidly-exploring Random Trees (RRT)



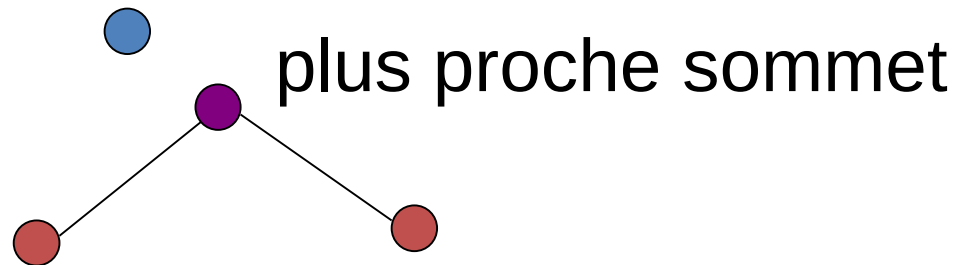
Rapidly-exploring Random Trees (RRT)



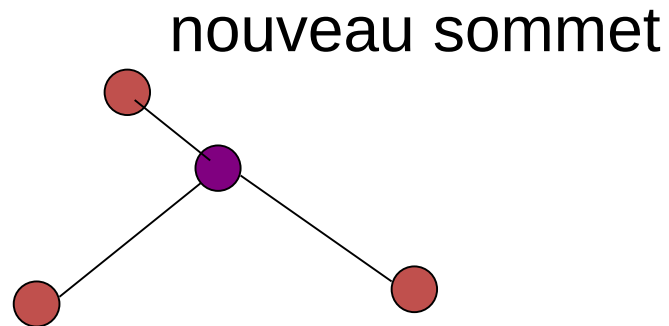
Rapidly-exploring Random Trees (RRT)



Rapidly-exploring Random Trees (RRT)



Rapidly-exploring Random Trees (RRT)



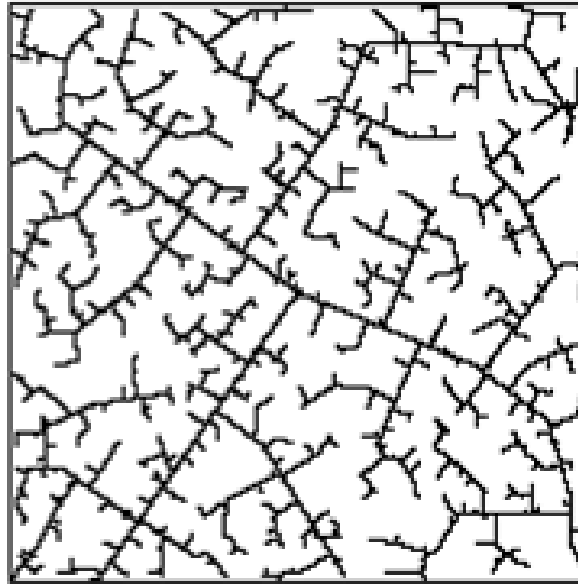
et ainsi de suite...

Rapidly-exploring Random Trees (RRT)

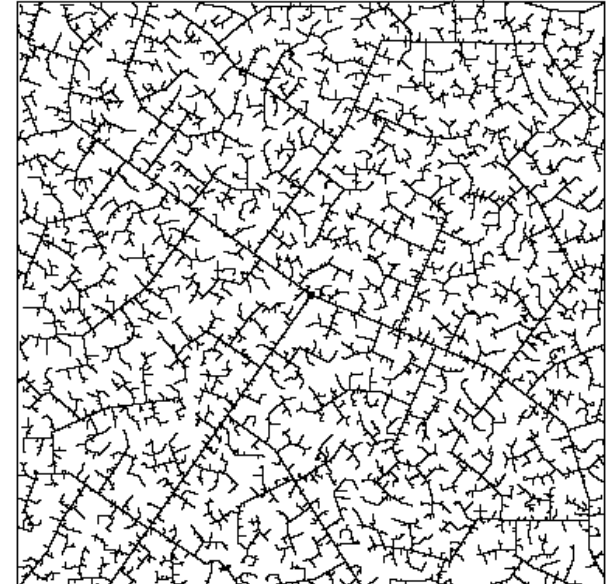
- Tend à bien remplir l'espace vide



45 itérations



390 itérations



2345 itérations

RRT-Connect

- Croître deux arbres, l'un partant de la configuration de départ et l'autre d'arrivée
- À chaque tirage, on essaie de connecter un des arbres avec l'autre, de façon vorace

RRT-Connect: exemple

Phase croissance de l'arbre

● Départ



○ Arrivée

RRT-Connect: exemple

Phase croissance de l'arbre

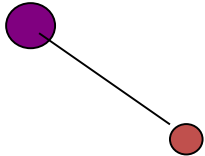


● sommet tiré au hasard

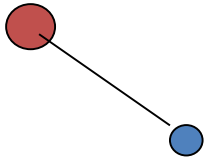


RRT-Connect: exemple

Phase croissance de l'arbre

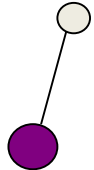
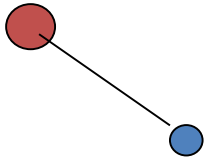


RRT-Connect: exemple



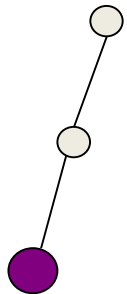
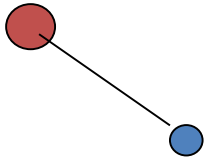
On essaie maintenant de
connecter l'arbre du bas vers
l'arbre du haut, de façon
vorace (greedy)

RRT-Connect: exemple



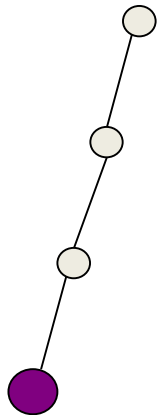
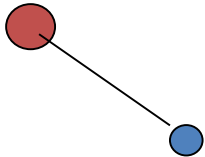
Phase connexion des arbres

RRT-Connect: exemple



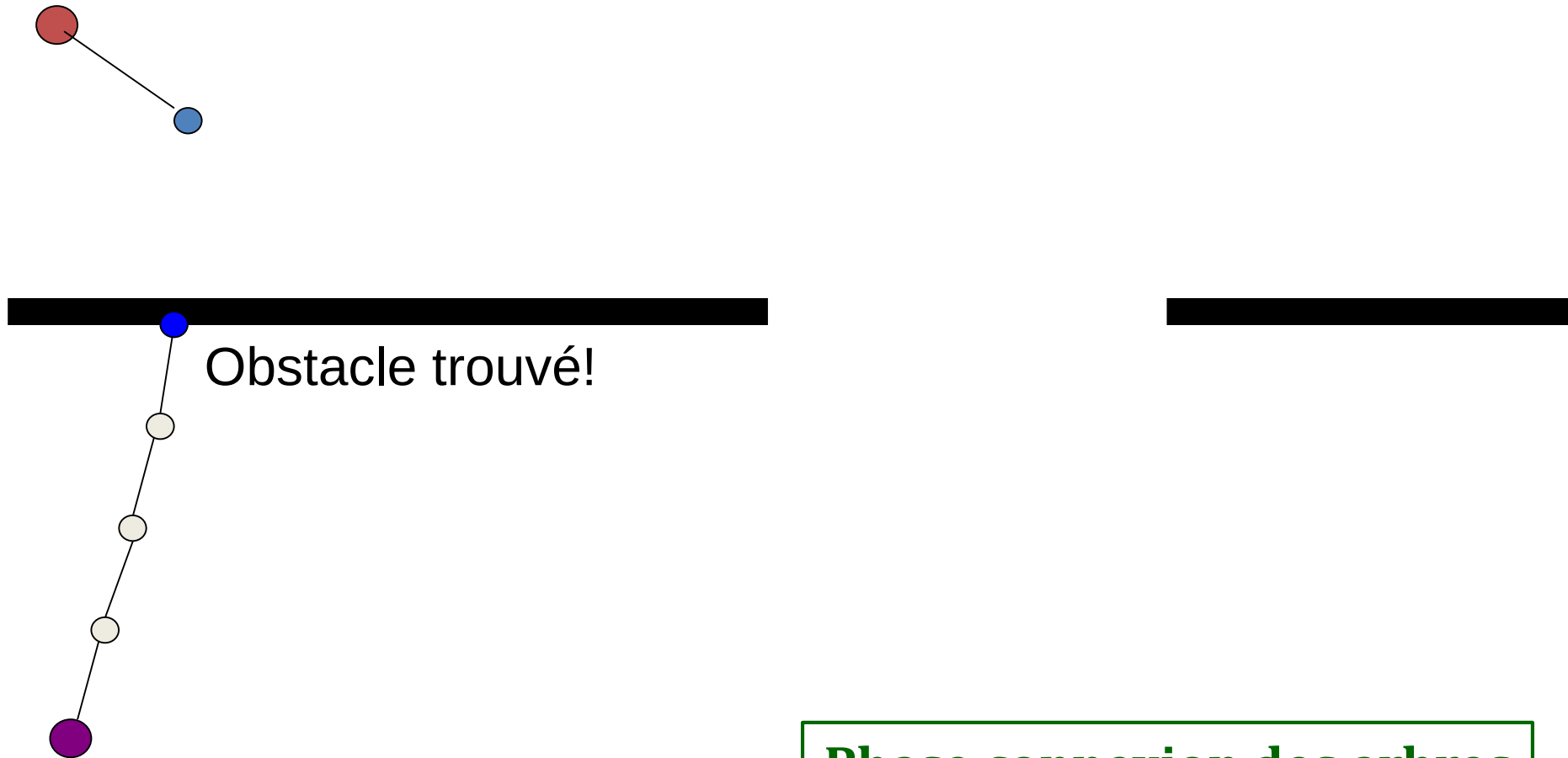
Phase connexion des arbres

RRT-Connect: exemple



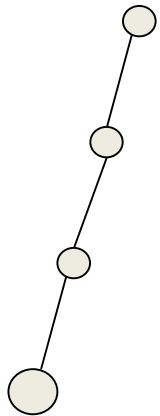
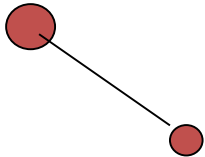
Phase connexion des arbres

RRT-Connect: exemple



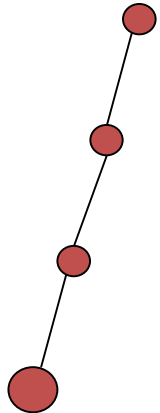
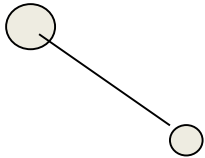
Phase connexion des arbres

RRT-Connect: exemple



On inverse les rôles!

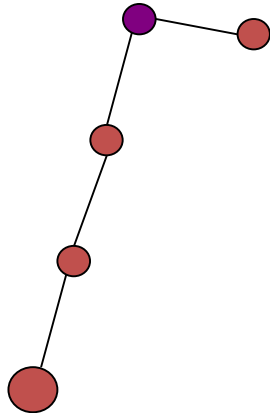
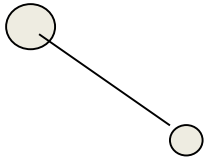
RRT-Connect: exemple



On inverse les rôles!

Phase croissance de l'arbre

RRT-Connect: exemple

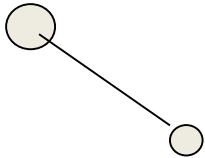


on tire un point au hasard, ajoute
un sommet à l'arbre du bas

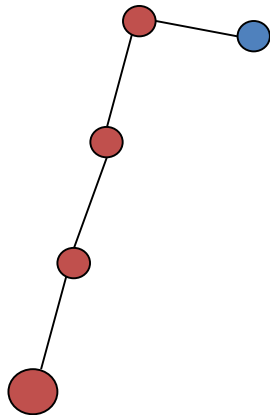
Phase croissance de l'arbre

RRT-Connect: exemple

Phase connexion des arbres

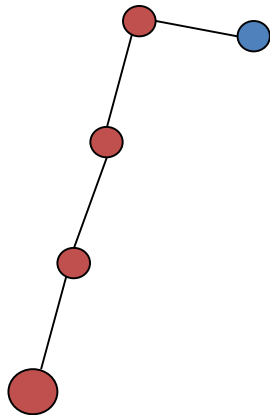
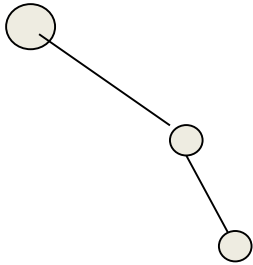


Et de façon vorace on croît l'arbre du haut



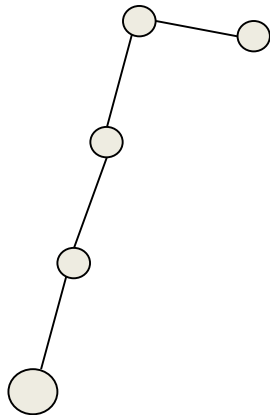
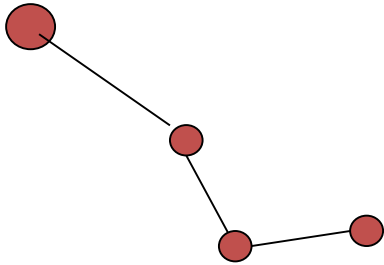
RRT-Connect: exemple

Phase connexion des arbres



RRT-Connect: exemple

Phase croissance de l'arbre



RRT-Connect: exemple



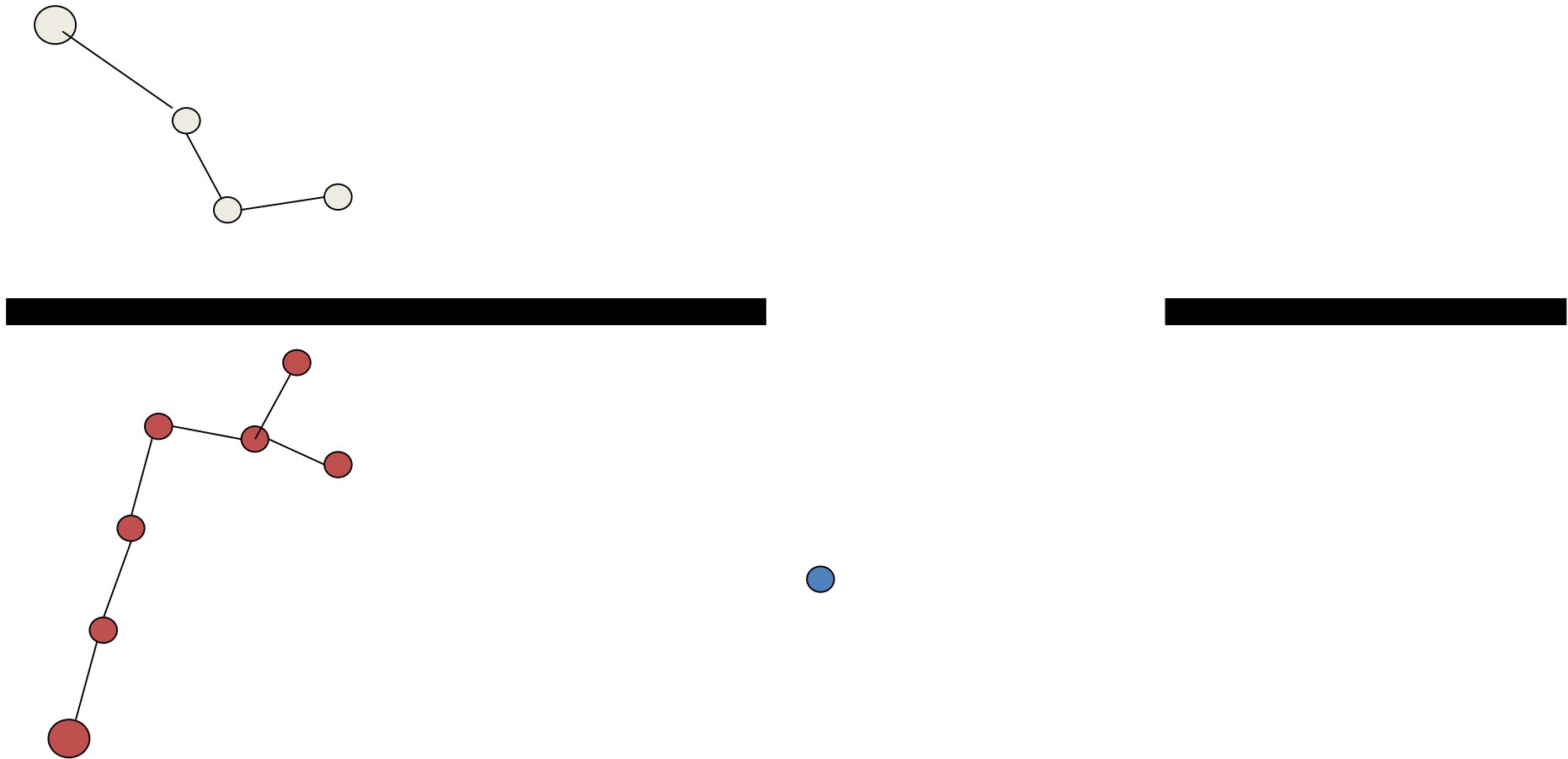
Phase connexion des arbres

RRT-Connect: exemple



Phase croissance de l'arbre

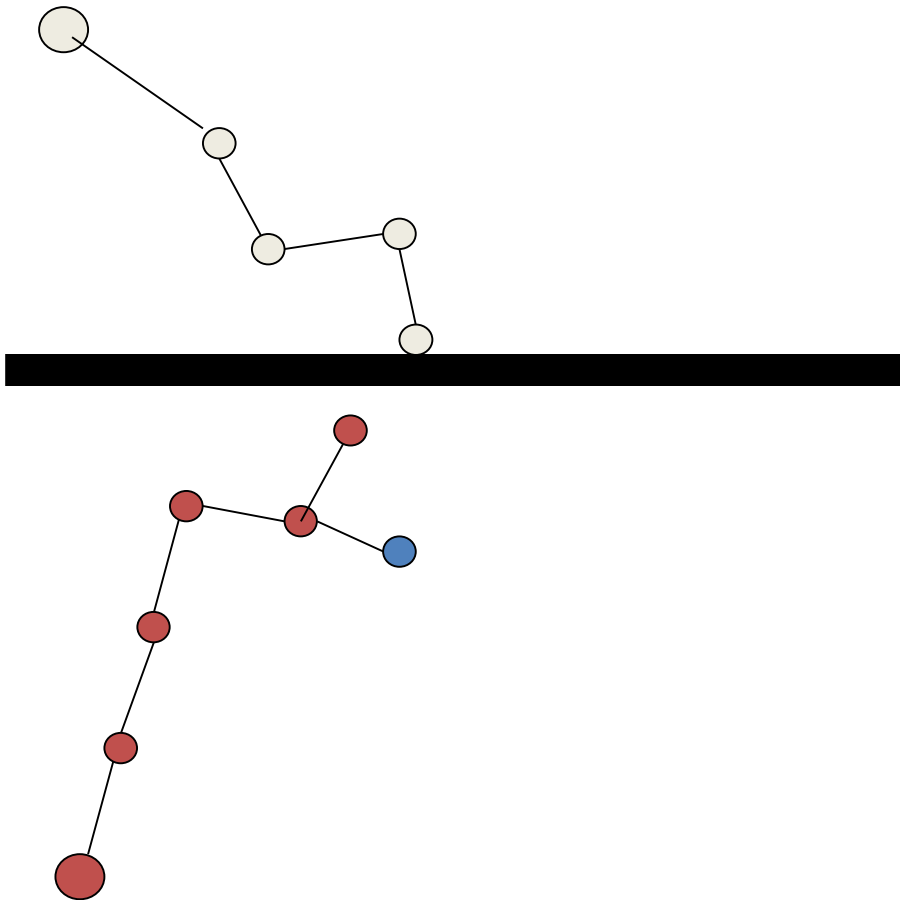
RRT-Connect: exemple



Phase croissance de l'arbre

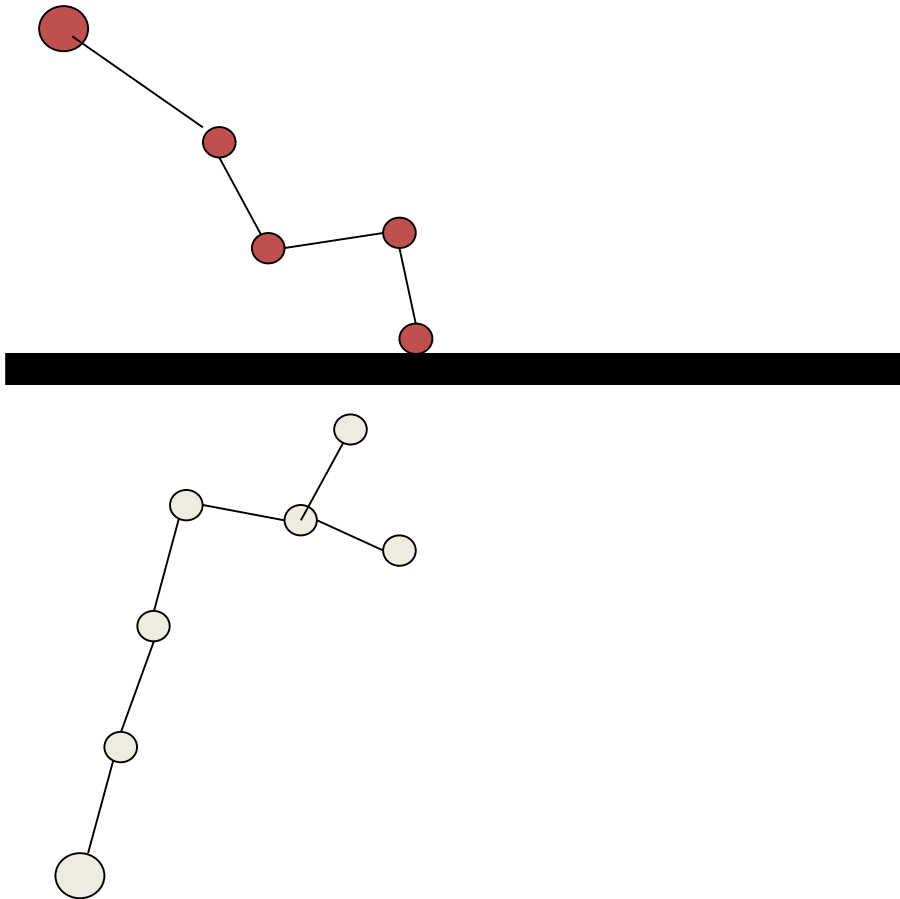
RRT-Connect: exemple

Phase connexion des arbres



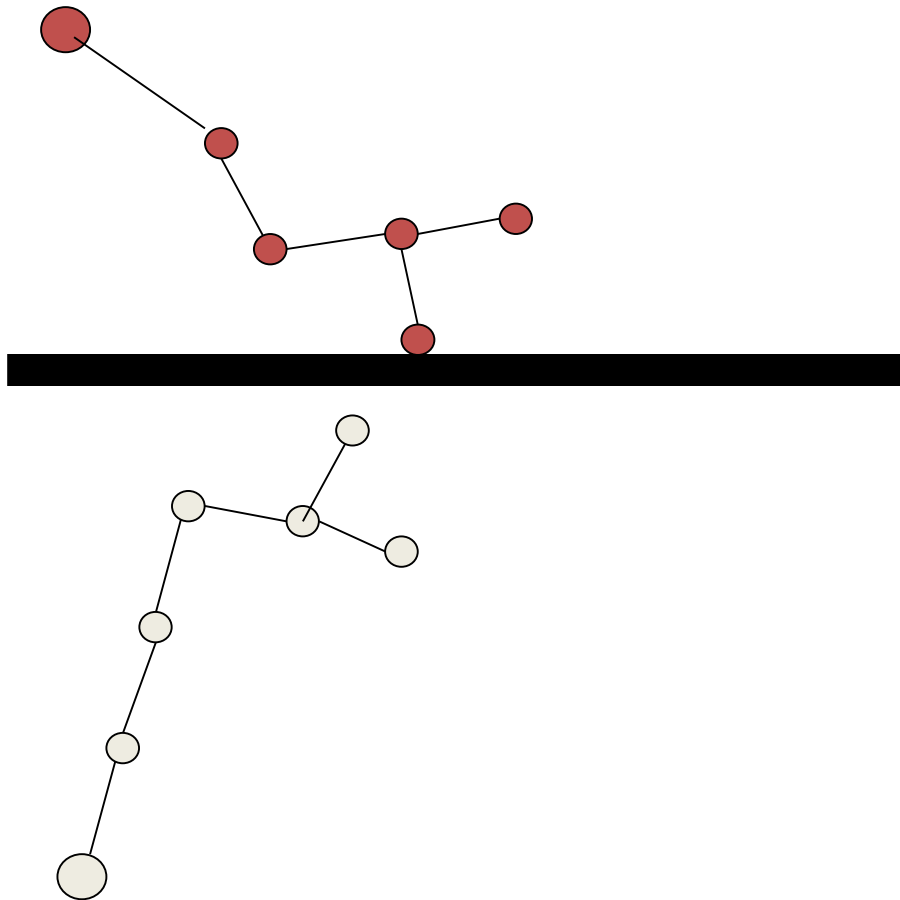
RRT-Connect: exemple

Phase croissance de l'arbre

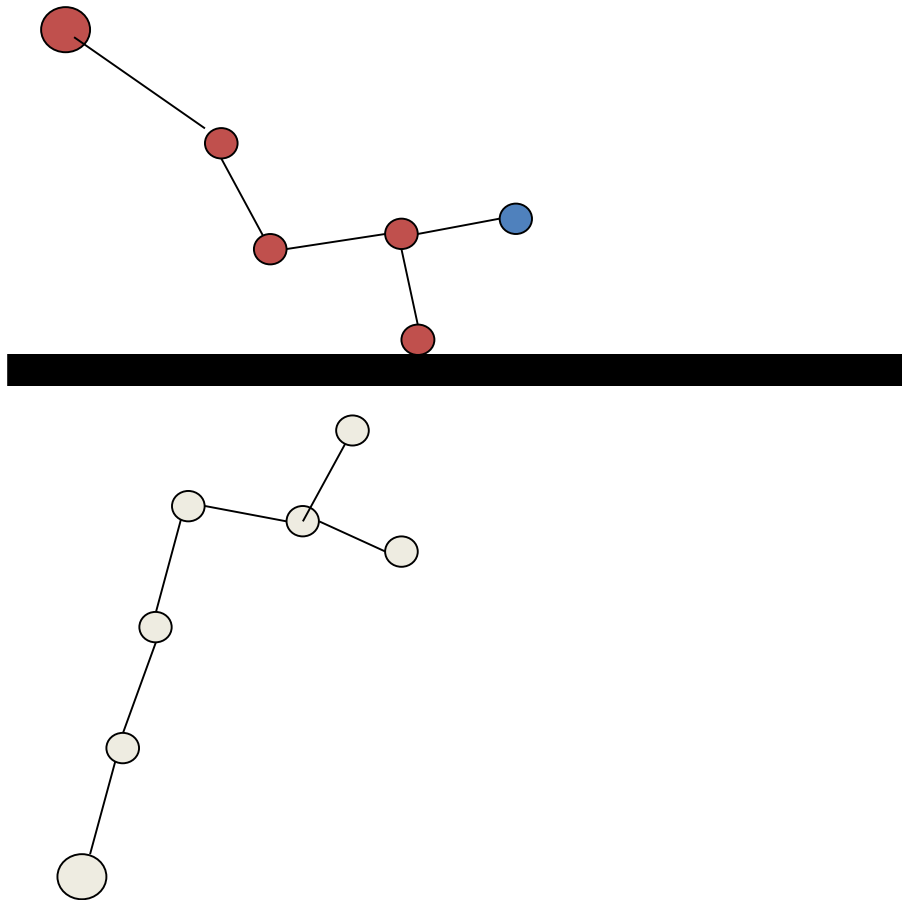


RRT-Connect: exemple

Phase croissance de l'arbre

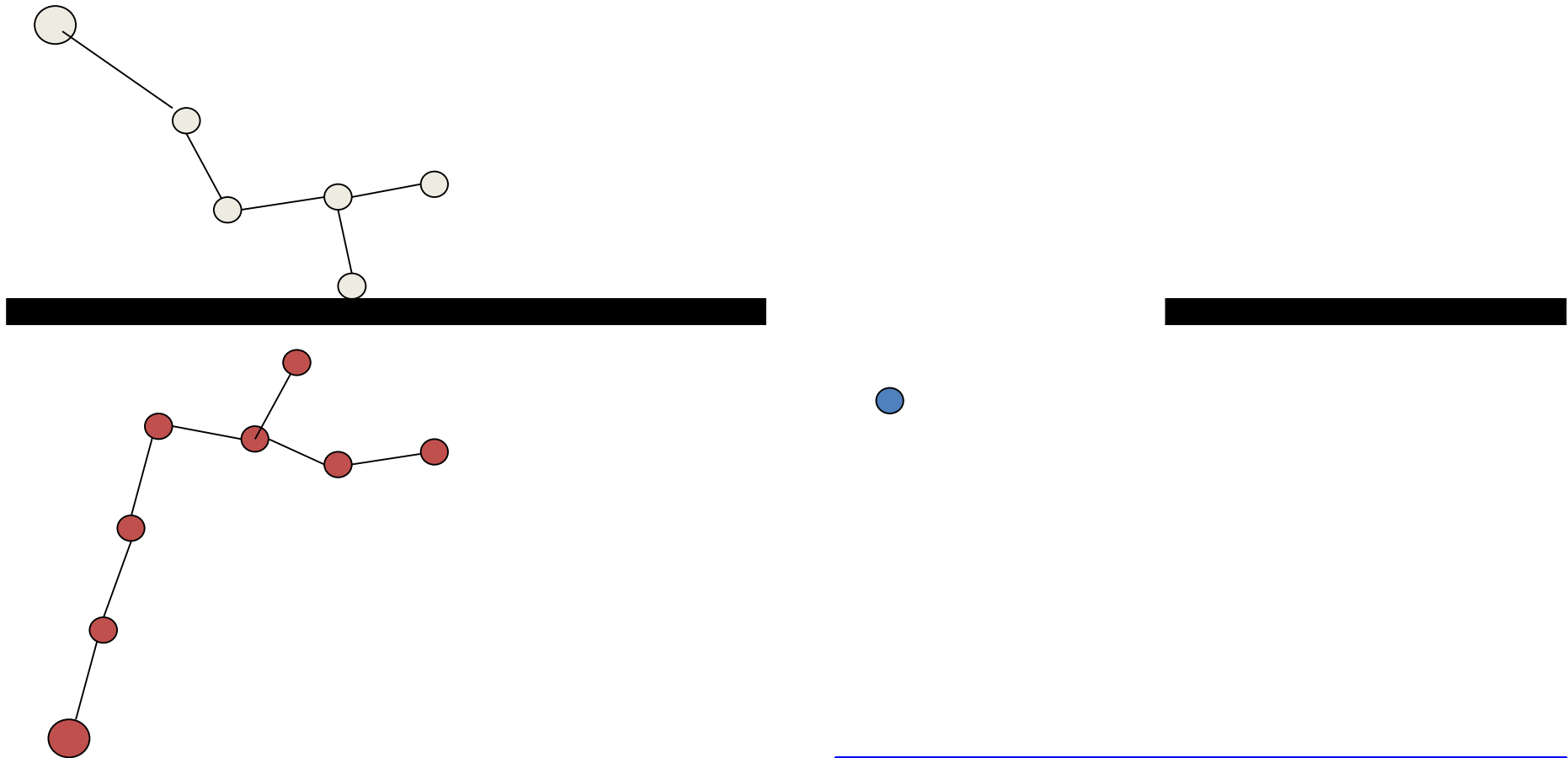


RRT-Connect: exemple



Phase connexion des arbres

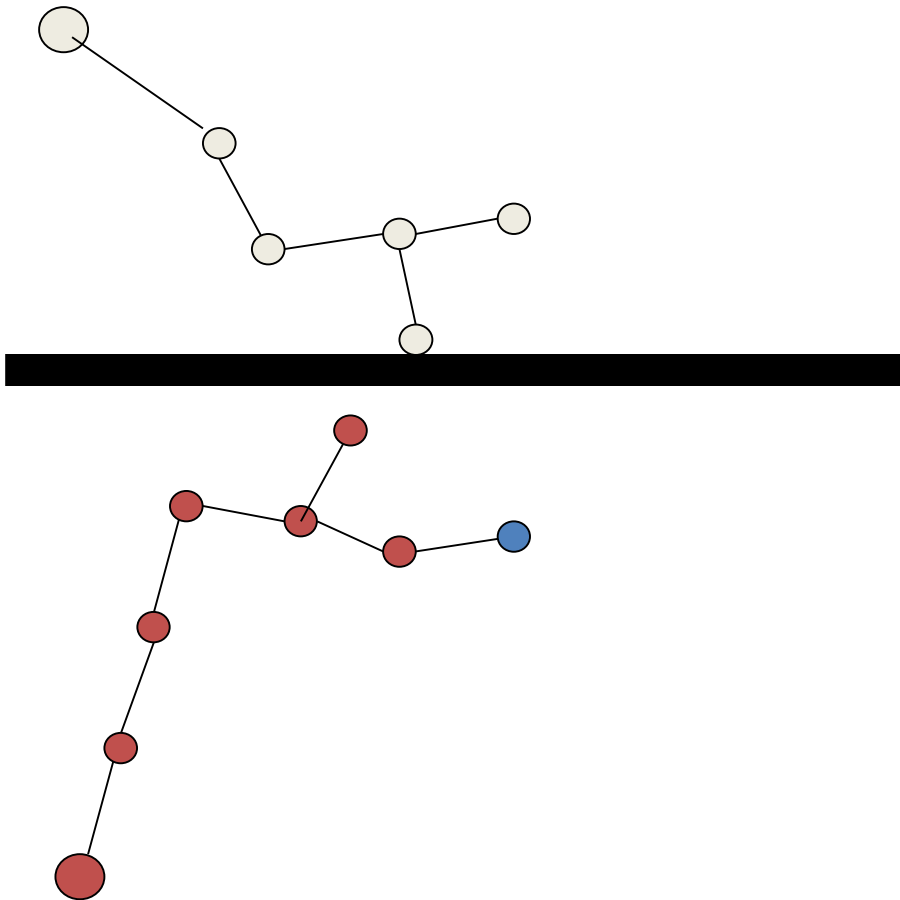
RRT-Connect: exemple



Phase croissance de l'arbre

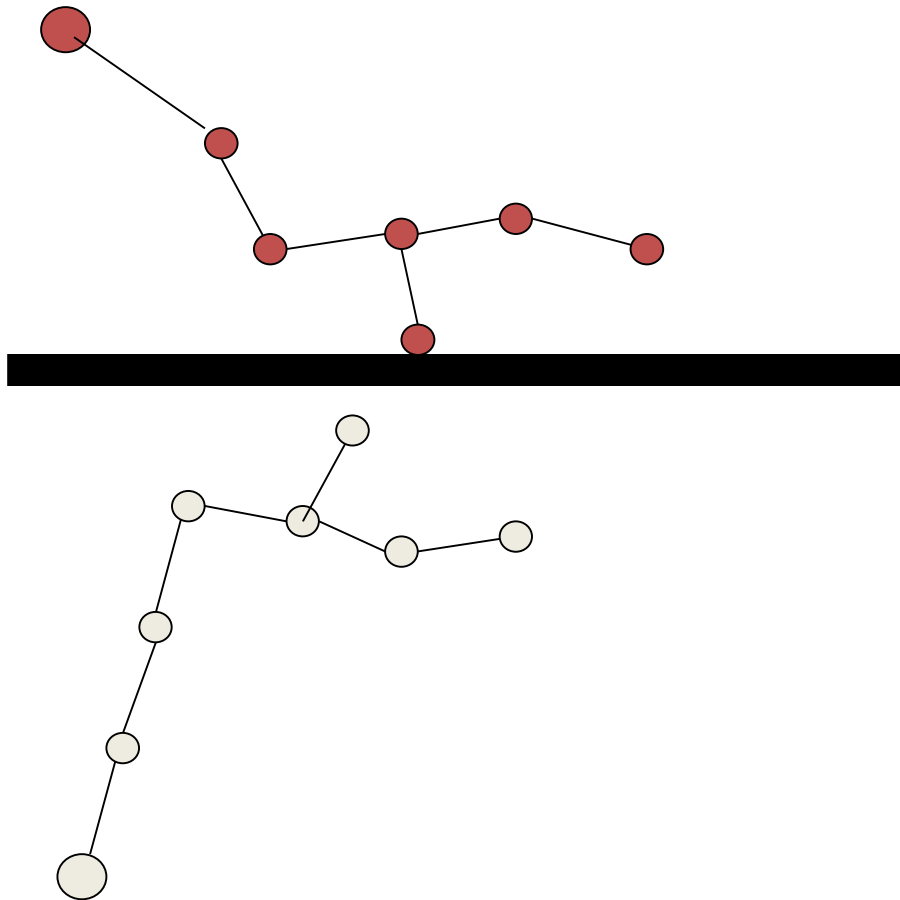
RRT-Connect: exemple

Phase connexion des arbres

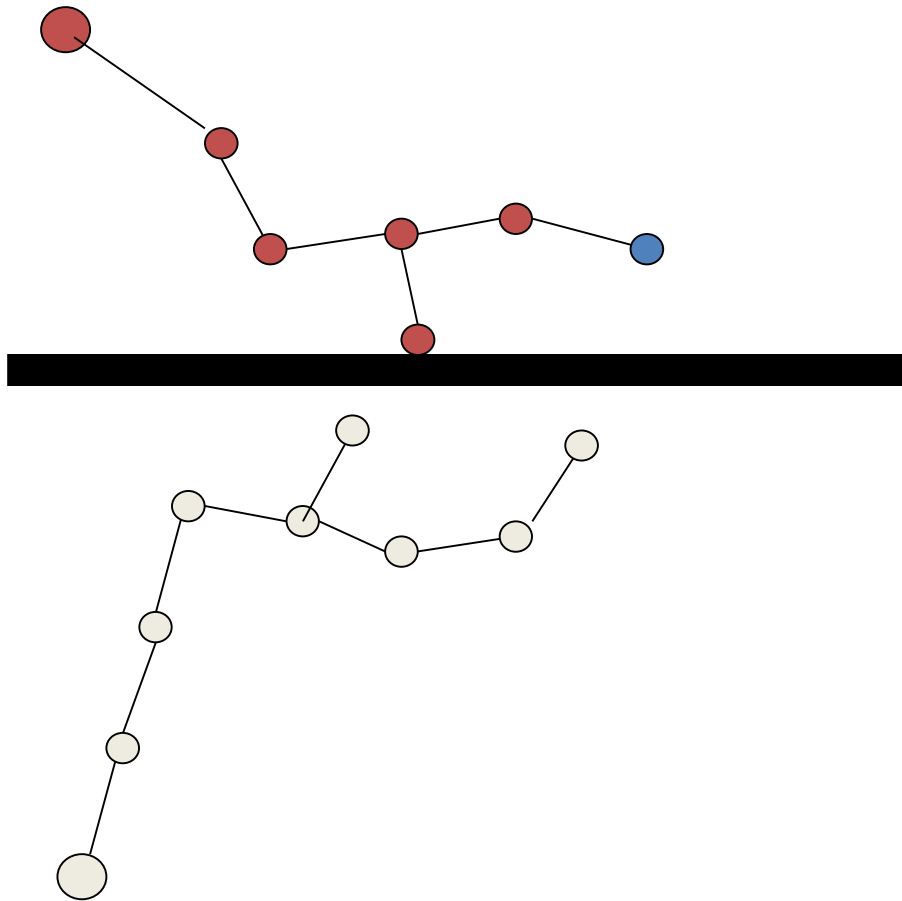


RRT-Connect: exemple

Phase croissance de l'arbre

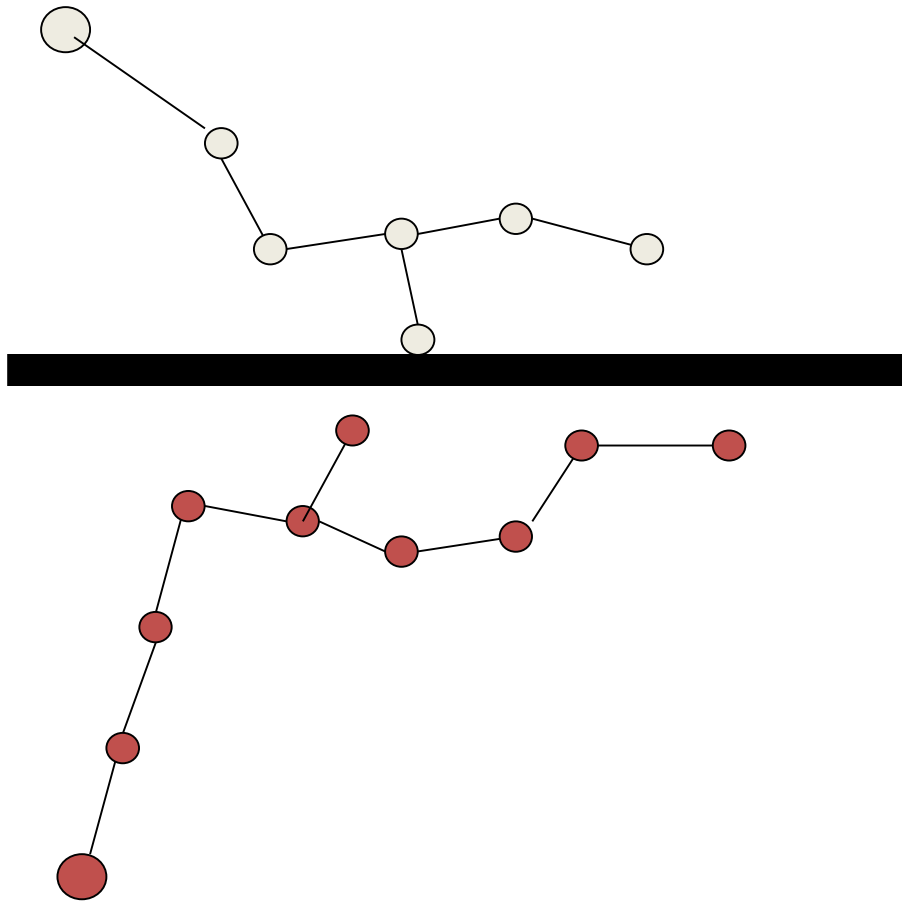


RRT-Connect: exemple



Phase connexion des arbres

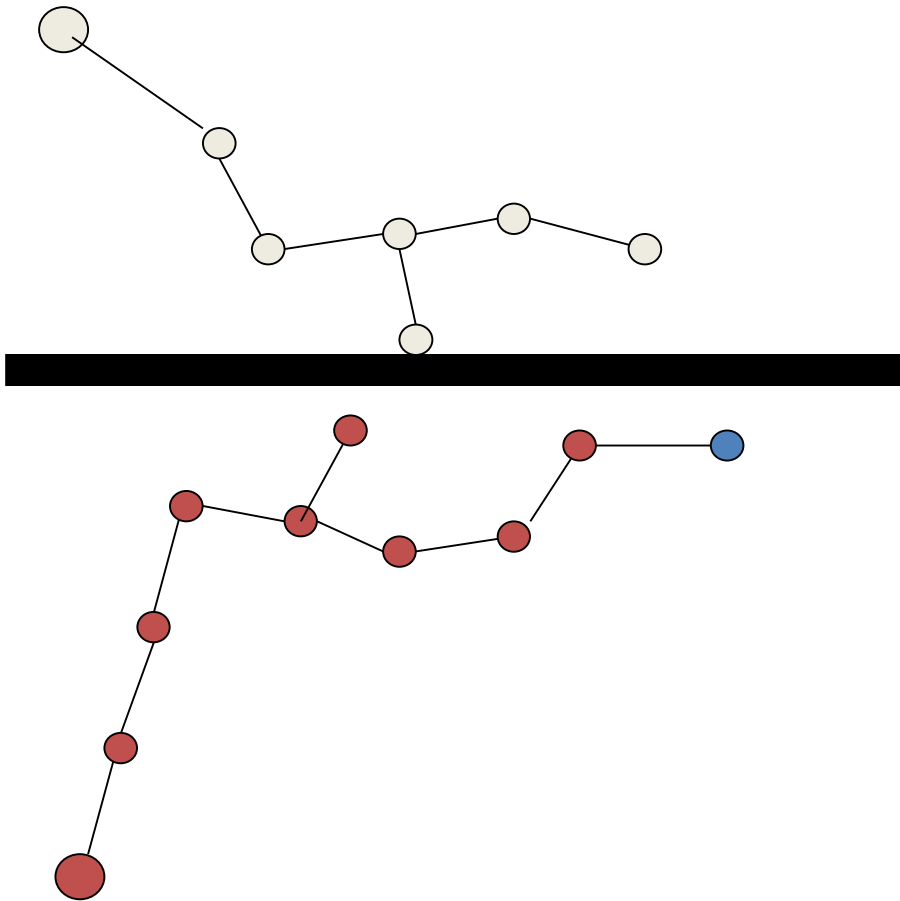
RRT-Connect: exemple



Phase croissance de l'arbre

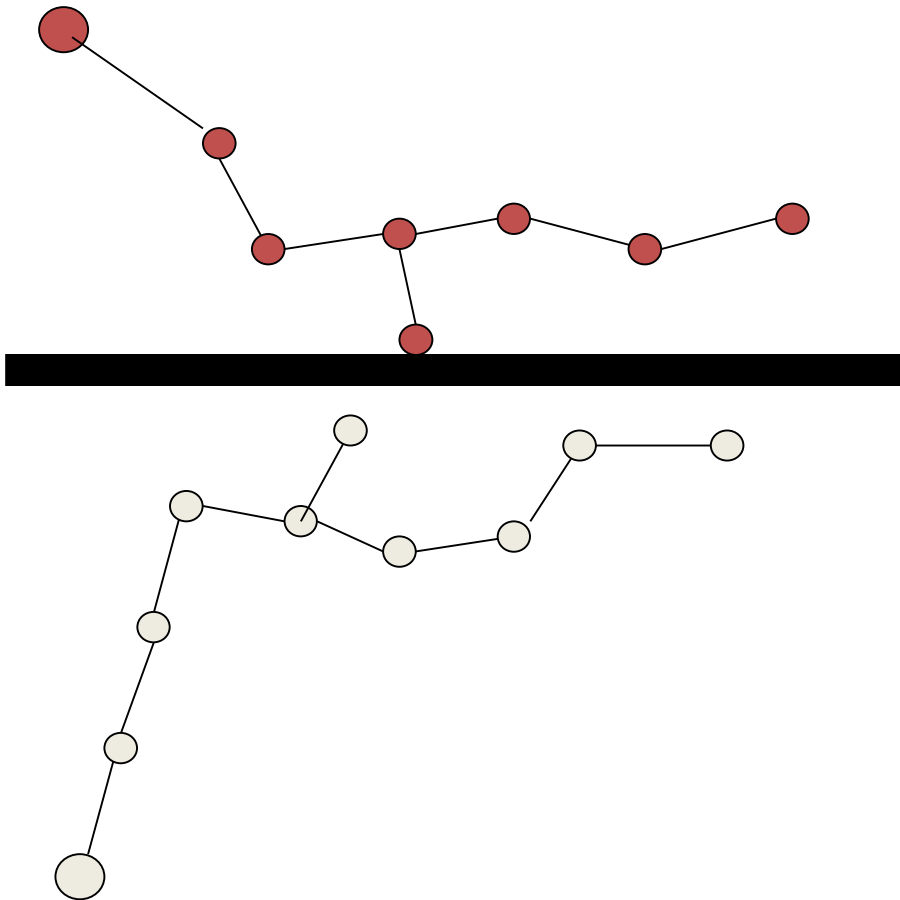
RRT-Connect: exemple

Phase connexion des arbres

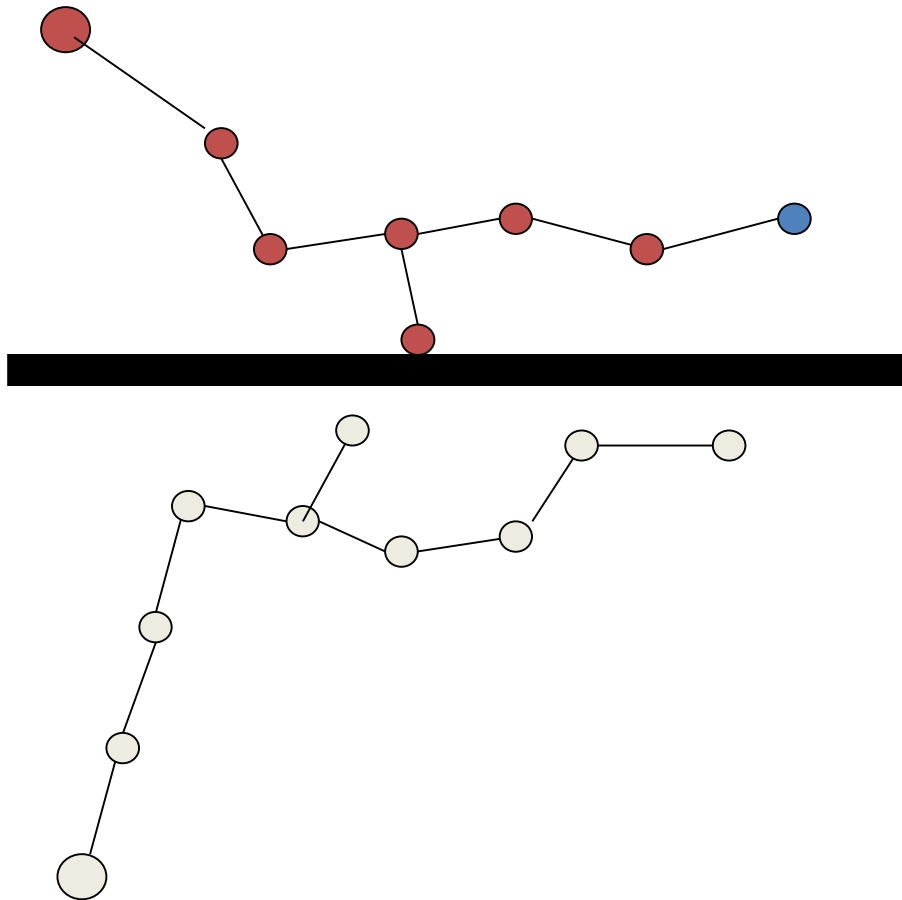


RRT-Connect: exemple

Phase croissance de l'arbre

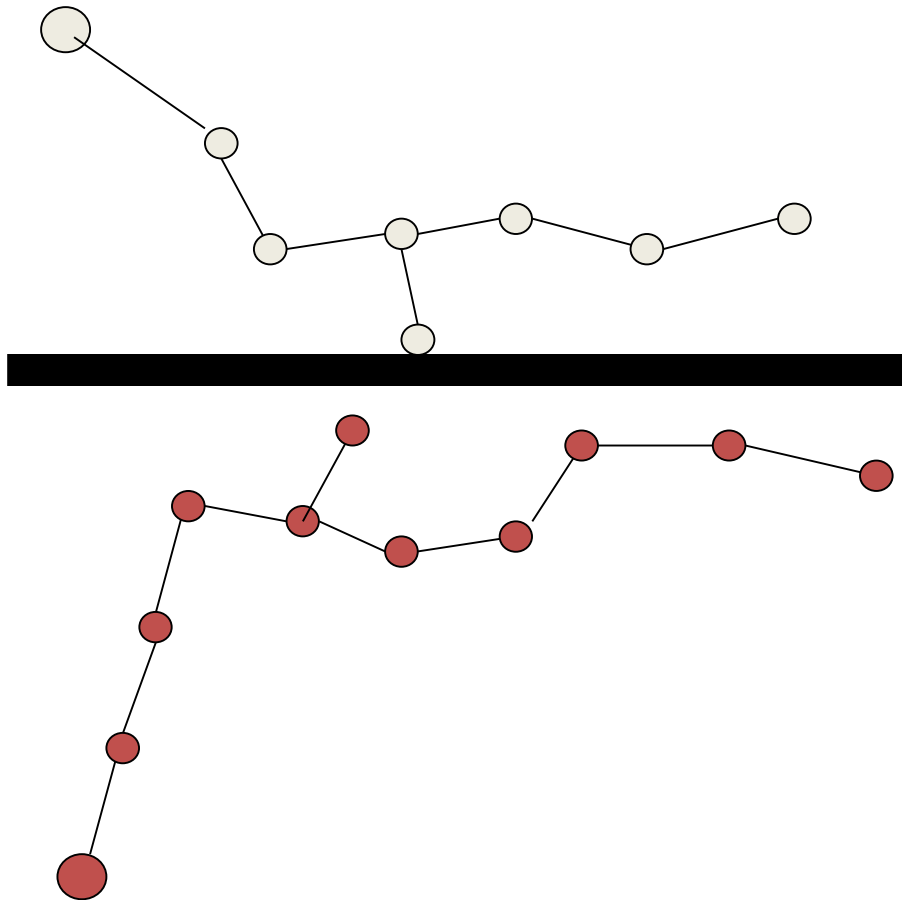


RRT-Connect: exemple



Phase connexion des arbres

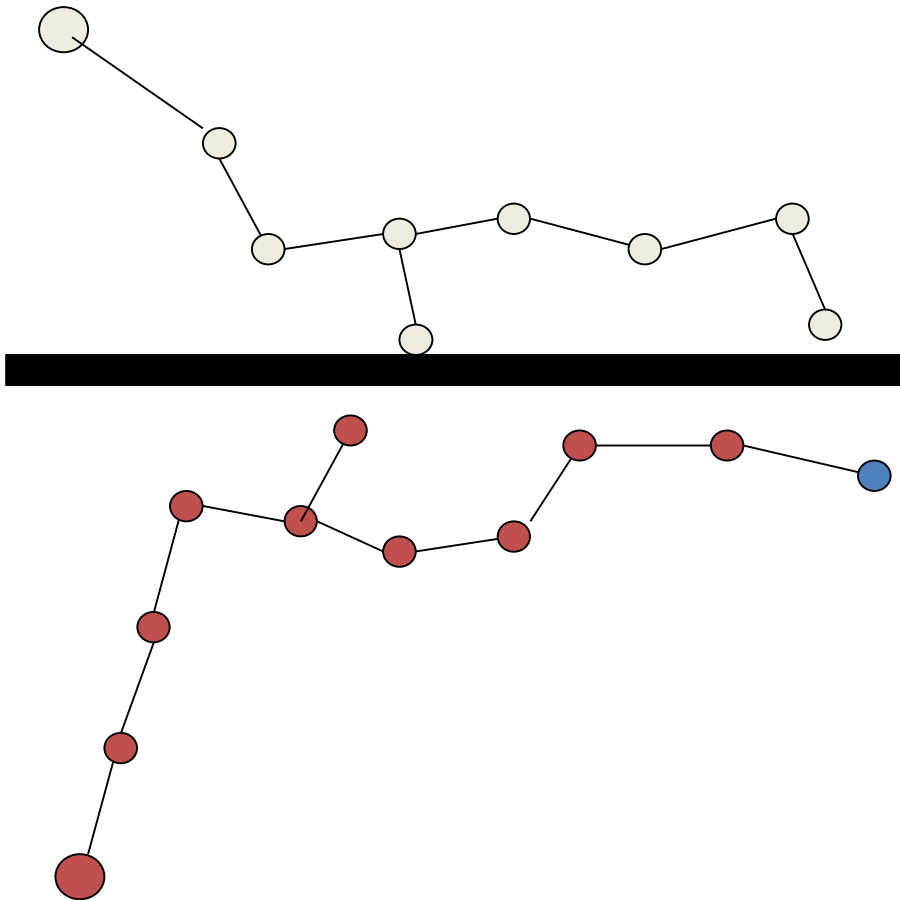
RRT-Connect: exemple



Phase croissance de l'arbre

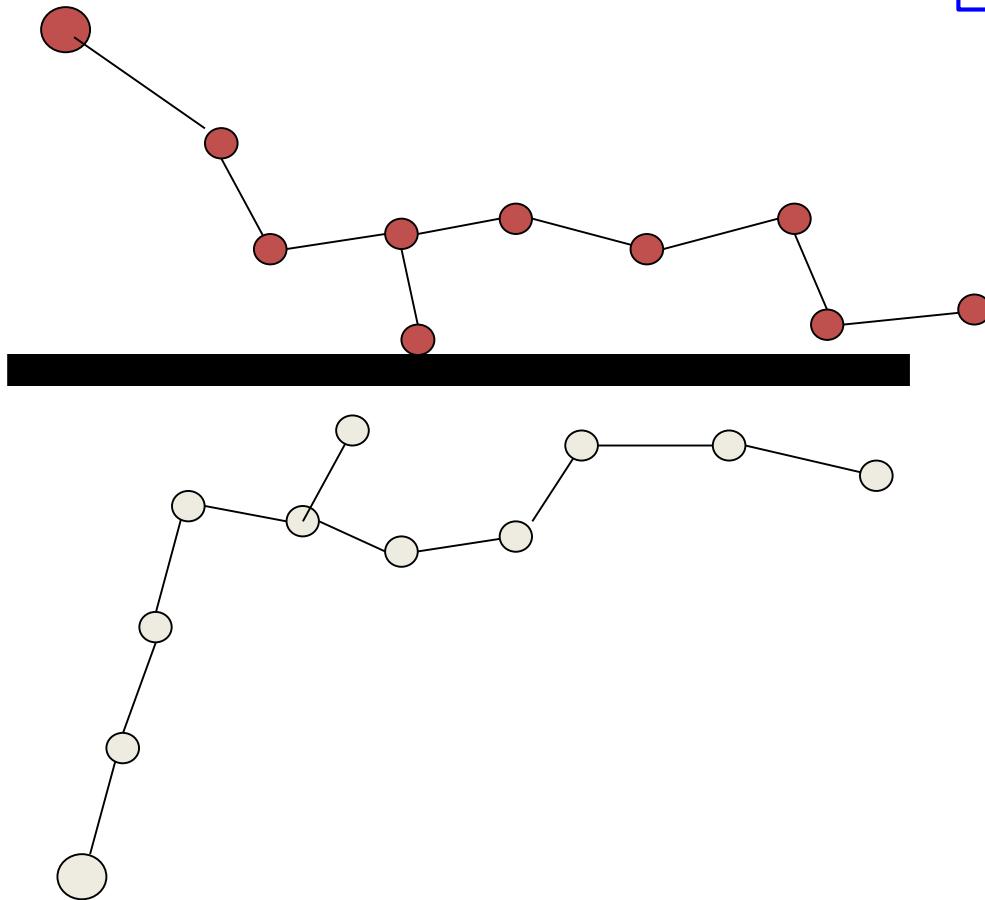
RRT-Connect: exemple

Phase connexion des arbres

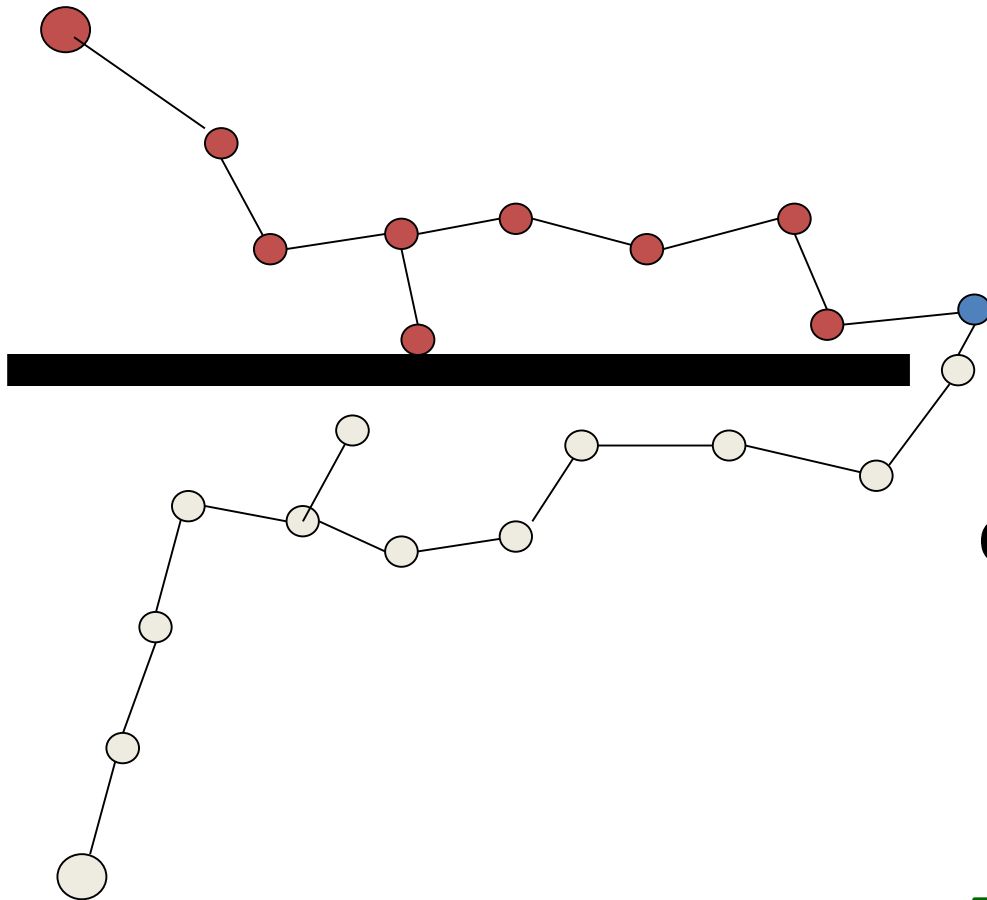


RRT-Connect: exemple

Phase croissance de l'arbre



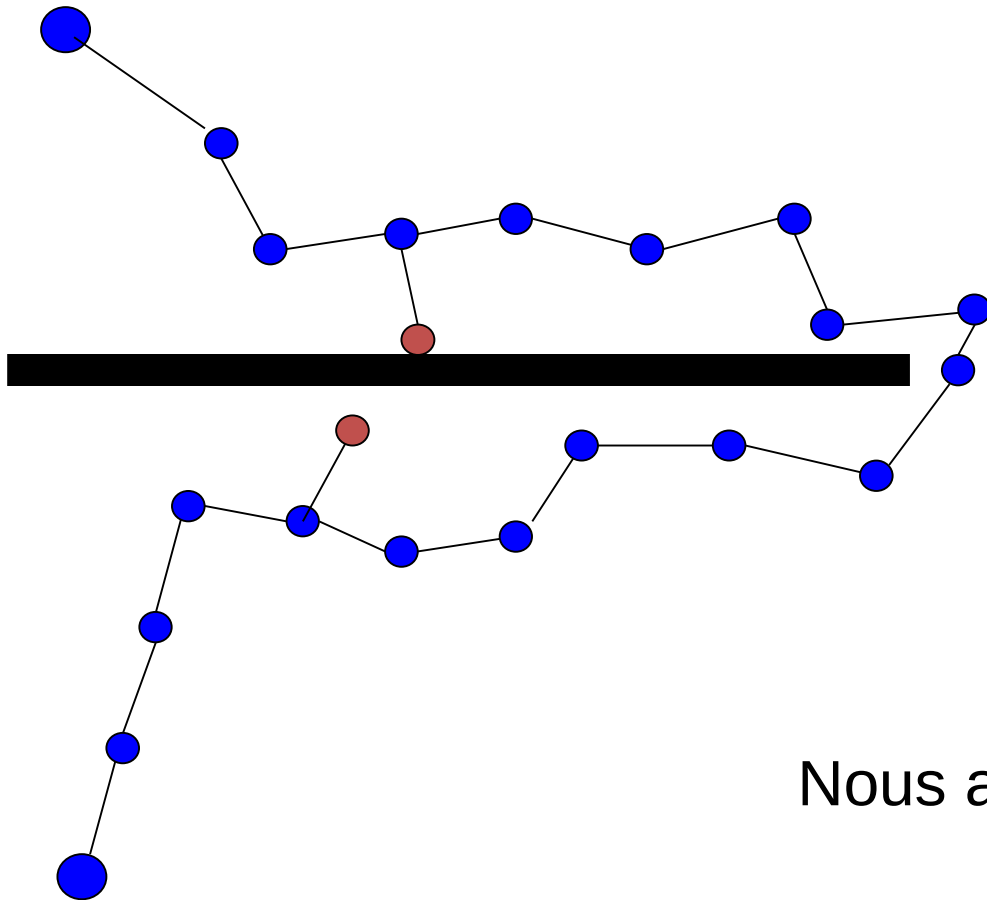
RRT-Connect: exemple



Connexion réussie!

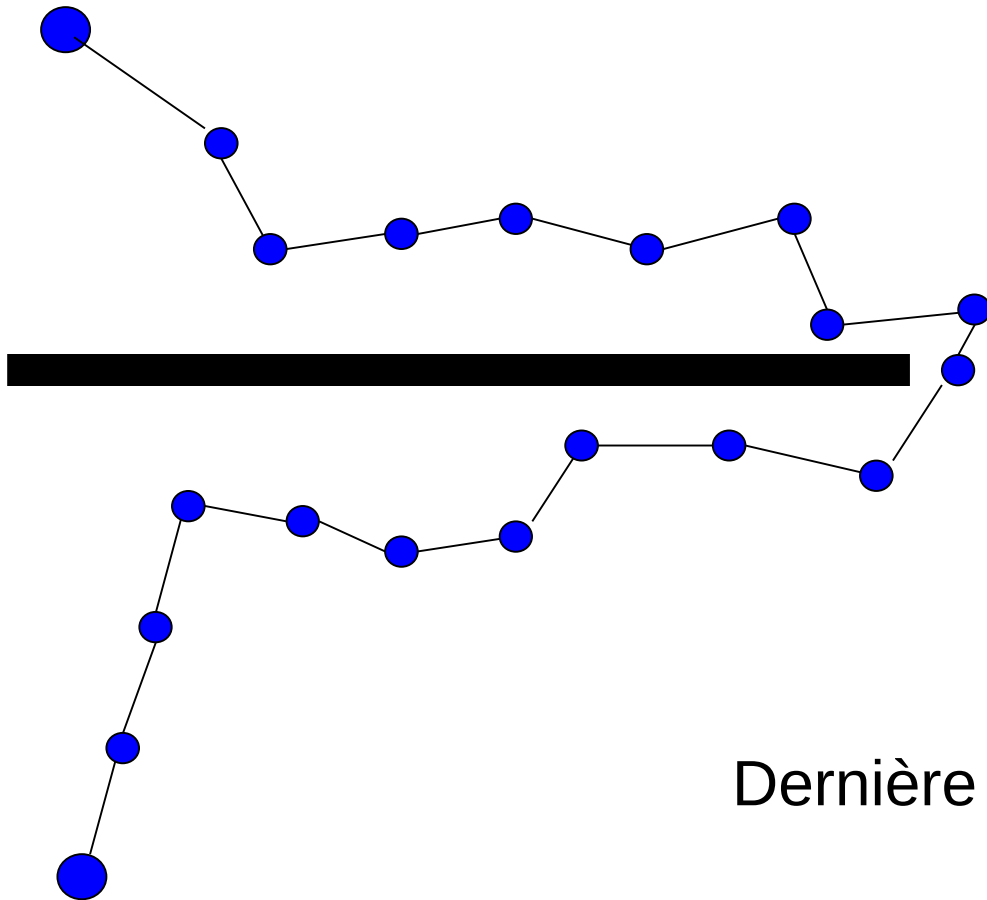
Phase connexion des arbres

RRT-Connect: exemple



Nous avons une solution !

RRT-Connect: exemple

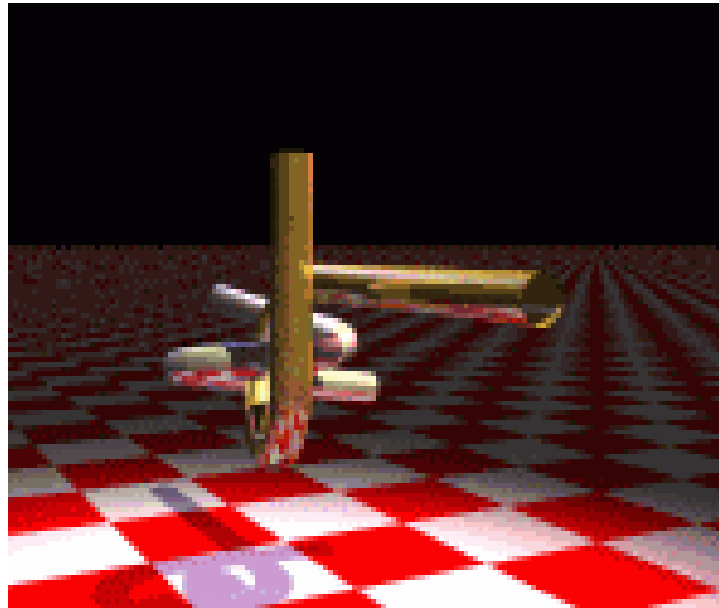


Dernière étape : lissage



Exemple de problème résolu avec RRT

- Extractions de deux tiges courbées



Problème proposé par Boris Yamrom, GE comme exemple de problème difficile. Repris par Nancy Amato, Texas A&M University. Solution en 2001 par James Kuffner et Steve LaValle. Calcul sur un PC de 2003 : quelques minutes.

RRT-Connect si non-holonome

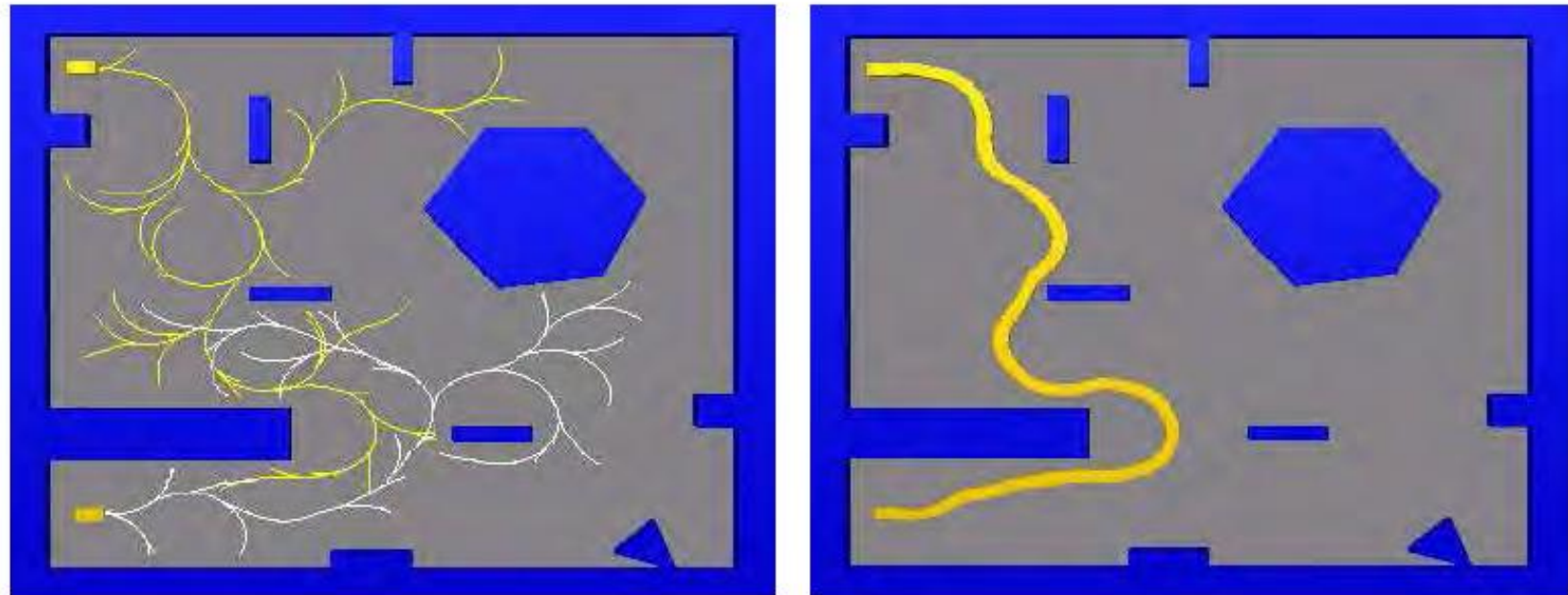


Fig. 4. The RRT obtained with a bidirectional planner (RRT-ExtExt) in 22.583 secs and 1376 nodes.

Exploring unknown environments with RRT-based strategies

Abraham Sánchez L. and René Zapata