Localisation et navigation de robots

UPJV, Département EEA

M2 3EA, EC32, parcours RoVA

Année Universitaire 2024-2025

Fabio MORBIDI

Laboratoire MIS

Équipe Perception Robotique

E-mail: fabio.morbidi@u-picardie.fr

Mardi et jeudi 9h00-12h00,
salle CURI 305 : CM & TD
Mardi 9h00-12h00, salle TP204 : TP

Electronique
Energie Electrique
Automatique

UNIVERSITÉ de Picardie Jules Verne

Plan du chapitre

Partie 4 : Planification de trajectoire et évitement d'obstacles

3. Fenêtre dynamique

Fenêtre dynamique

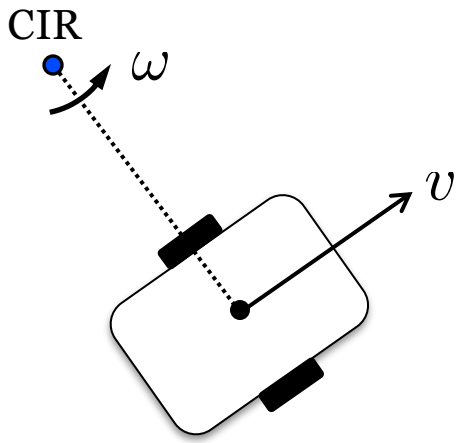
- Sélection d'un couple (v, ω)

  v : vitesse longitudinale du robot

  ω : vitesse angulaire du robot

  selon des contraintes :

  - Evitement d'obstacles
  - Atteindre un but
  - Modèle cinématique/dynamique du robot



Robot unicycle

"*The dynamic window approach to collision avoidance*", D. Fox, W. Burgard, S. Thrun, IEEE Robot. & Autom. Magazine, vol. 4, n. 1, pp. 23-33, 1997

"*High-speed navigation using the global dynamic window approach*", O. Brock, O. Khatib, in Proc. IEEE Int. Conf. Robotics and Automation, pp. 341-346, vol. 1, 1999

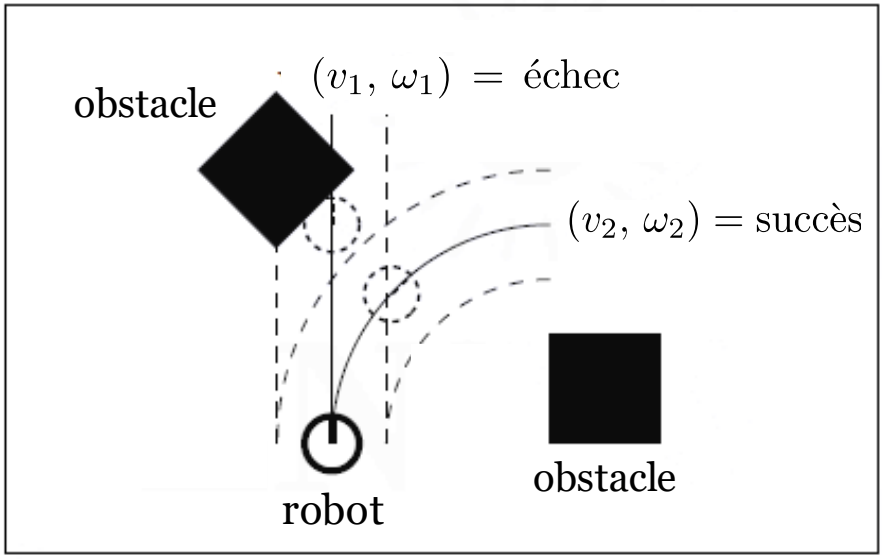
Fenêtre dynamique

- Contrainte principale : *évitement d'obstacles*
  - Contrainte dure
    - Binaire : succès/échec
    - Elle doit être obligatoirement satisfaite
  - Évaluation :
    - À partir de l'environnement perçu
    - À partir de la position future *estimée* du robot

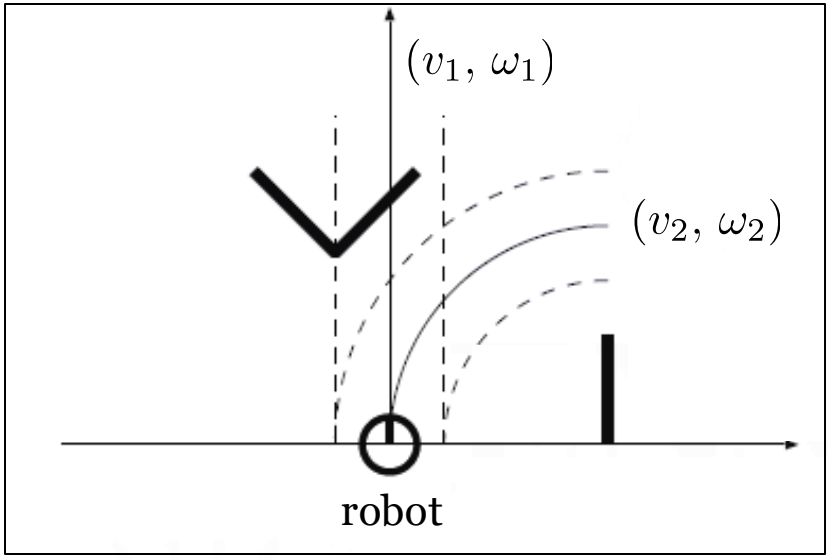
Fenêtre dynamique

- Illustration de la contrainte d'évitement d'obstacles

Environnement réel



Perceptions du robot



Fenêtre dynamique

- Construction du *graphe des vitesses*
  - Ensemble des couples de vitesse (et donc des trajectoires) possibles du robot

- Dans ce graphe, on peut tracer la *fenêtre des vitesses accessibles* au prochain pas de temps :

$$V_d = \{(v, \omega) \mid v \in [v_a - \dot{v}t,\ v_a + \dot{v}t] \text{ et } \omega \in [\omega_a - \dot{\omega}t,\ \omega_a + \dot{\omega}t]\}$$

où

t : intervalle de temps pendant lequel les accelerations linéaires et angulaires $(\dot{v}, \dot{\omega})$ du robot seront appliquées

(v_a, ω_a) : vitesse courante du robot

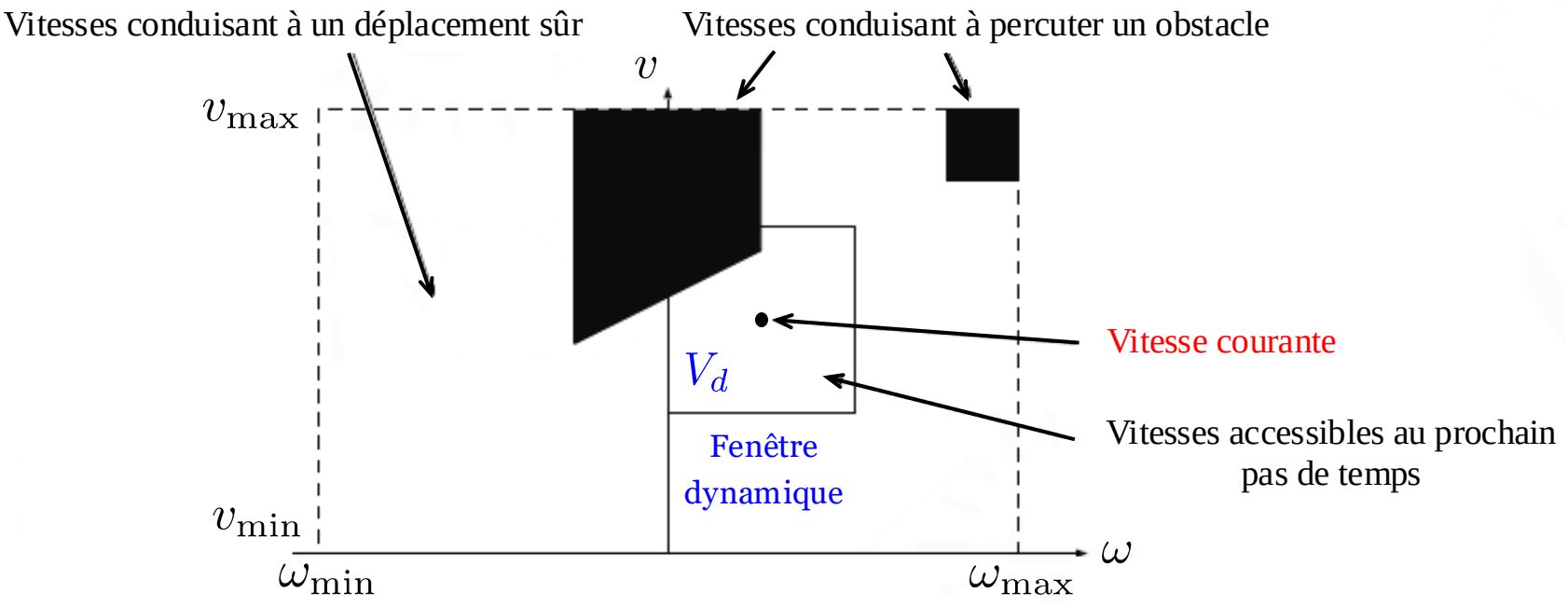
Remarque :
La fenêtre dynamique V_d est centrée sur la vitesse courante du robot et elle contient les vitesses du robot accessibles dans le prochain intervalle de temps

Fenêtre dynamique

- Fenêtre dynamique

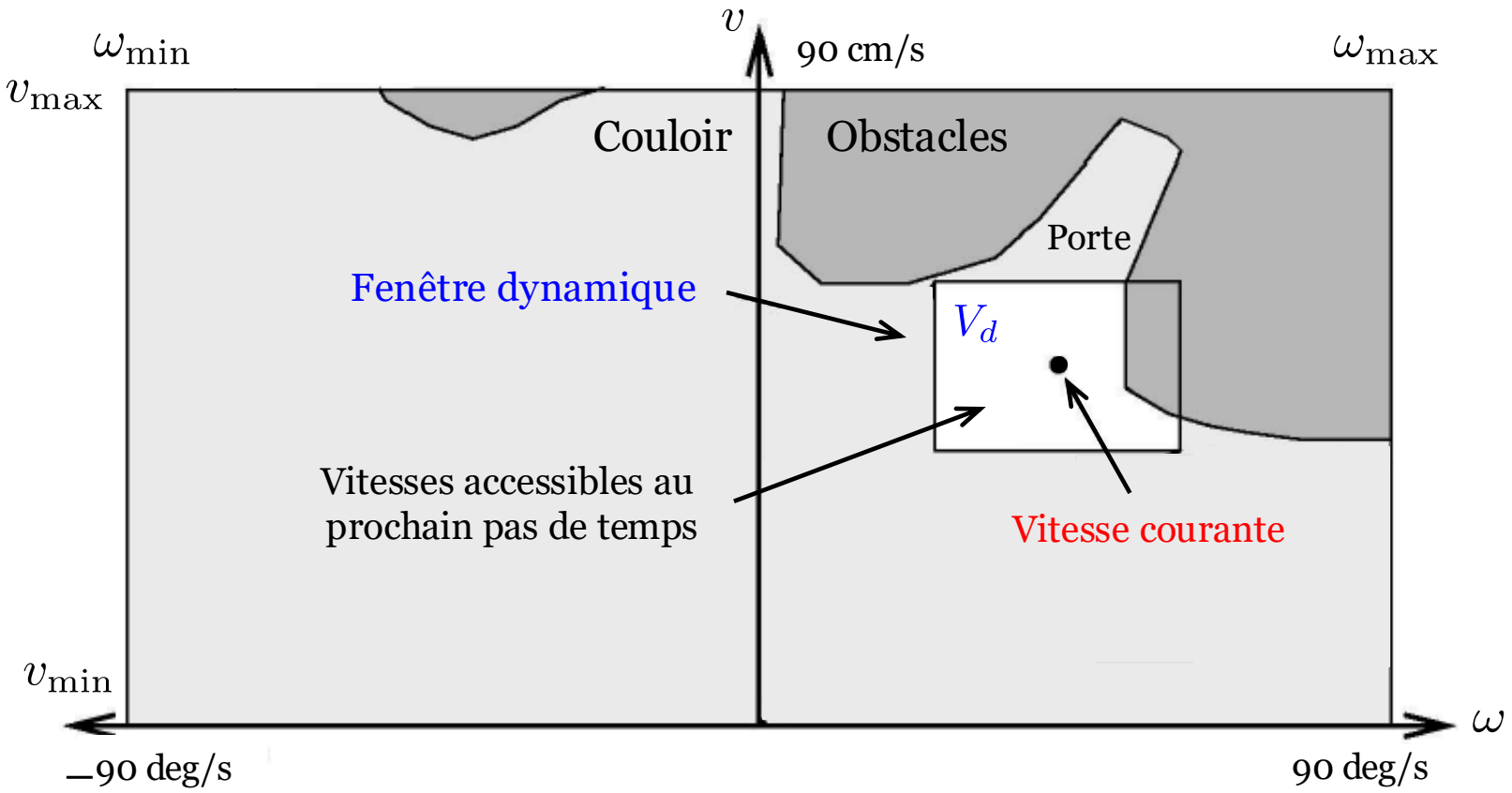
$$V_d = \{(v, \omega) \mid v \in [v_a - \dot{v}t, v_a + \dot{v}t] \text{ et } \omega \in [\omega_a - \dot{\omega}t, \omega_a + \dot{\omega}t]\}$$

Exemple:



Fenêtre dynamique

- Fenêtre dynamique pour le robot RHINO B21 de *Real World Interface*, avec un système synchro-drive à 3 roues [Fox *et al.*, RAM'97]
  - Max vitesse de translation du robot : 95 cm/s



Fenêtre dynamique

- Fenêtre
  - Prise en compte des obstacles
  - Choix d'un couple (v, ω) conduisant à un déplacement sûr
  - *Problème* : grande nombre de couples de vitesses possibles dans la fenêtre dynamique

- Solution possible
  - Ajout de *contraintes souples* : fonction de coût $G(v, \omega)$ à optimiser
  - Expression de *préférences* dans l'espace des vitesses accessibles

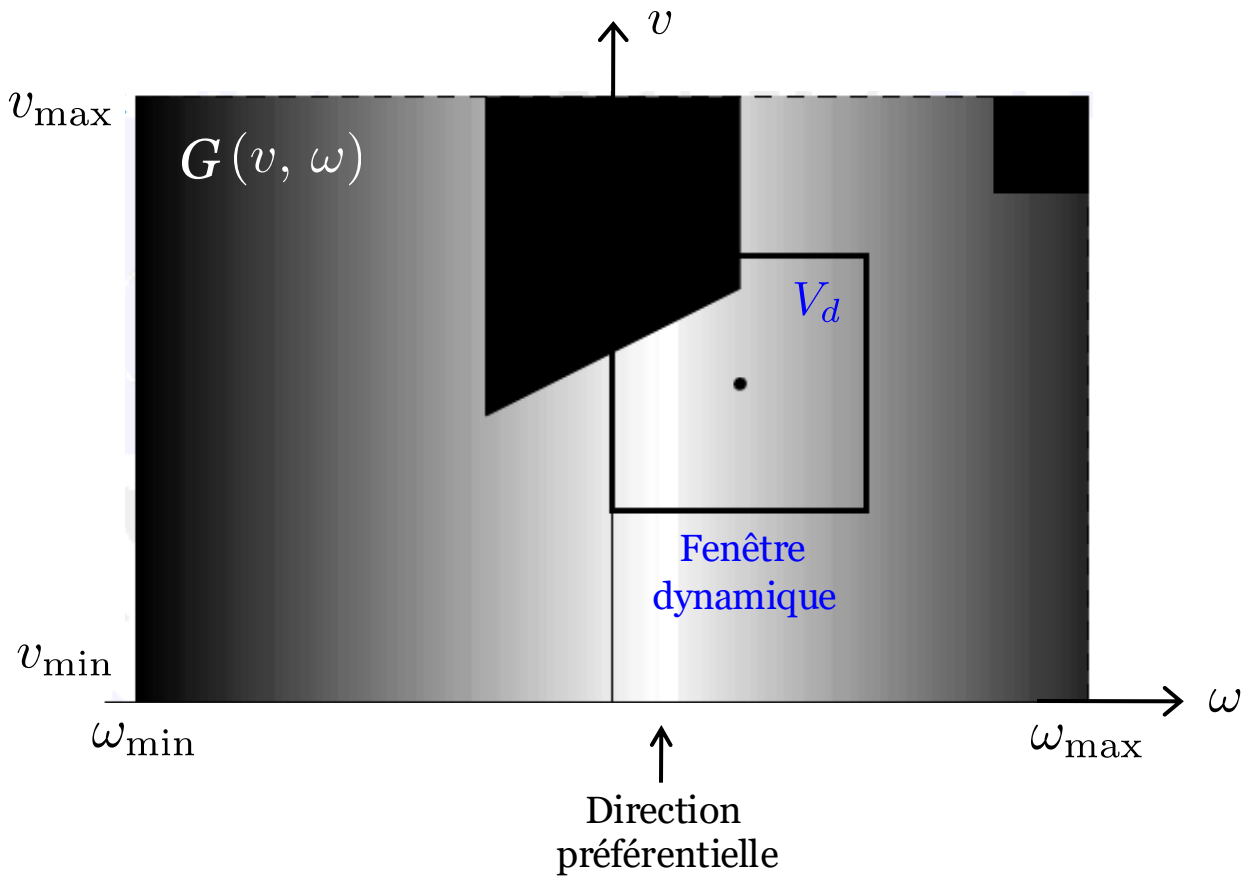
Fenêtre dynamique

- Coût : $G(v, \omega) = \sigma\left[\alpha\,\mathrm{heading}(v, \omega) + \beta\,\mathrm{dist}(v, \omega) + \gamma\,\mathrm{vel}(v, \omega)\right]$
  - Somme de plusieurs termes :
    1. Préférence de *direction* (« heading ») : utile si nous avons une estimation de la direction d'un but à long terme
    2. Préférence sur l'*éloignement maximal des obstacles*
    3. Préférence a priori sur les *vitesses* (longitudinales)

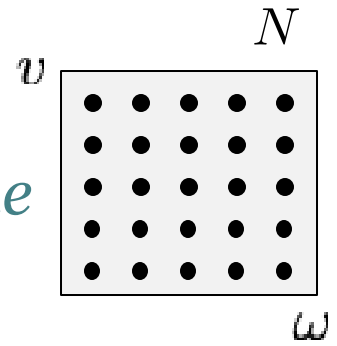
    La fonction $\sigma[\cdot]$ lisse la somme pondérée des trois termes (pour garantir plus d'espace latéral libre par rapport aux obstacles) et α, β, γ sont trois gains positifs
  - Le couple (v, ω) dans la fenêtre dynamique V_d qui *maximise* le coût $G(v, \omega)$ est alors choisi
  - Ce couple garantit l'évitement d'obstacles (contrainte dure) **et** les contraintes souples

Fenêtre dynamique

- Exemple de contrainte souple
  - Préférence directionnelle : $G(v, \omega) = \alpha \, \text{heading}(v, \omega)$



Fenêtre dynamique



- En pratique :
  - Evaluation des contraintes en N points du *graphe des vitesses* (c'est-à-dire, sur une grille)
    - La valeur de N dépend des ressources de calcul disponibles et de la complexité des contraintes
  - Utilisation intéressante pour:
    - Robots rapides
    - Robots à forte accélération/décélération

} Déplacement sûr et régulier

"*High-speed navigation using the global dynamic window approach*", O. Brock, O. Khatib, in Proc. IEEE Int. Conf. Robotics and Automation, pp. 341-346, vol. 1, 1999

- Carte de 30 m × 30 m de l'environnement avec une résolution de 5 cm
- Fréquence de commande : supérieure à 15 Hz
- Vitesse longitudinale moyenne du robot : supérieure à 1 m/s

Partie 4 : Planification de trajectoire et évitement d'obstacles

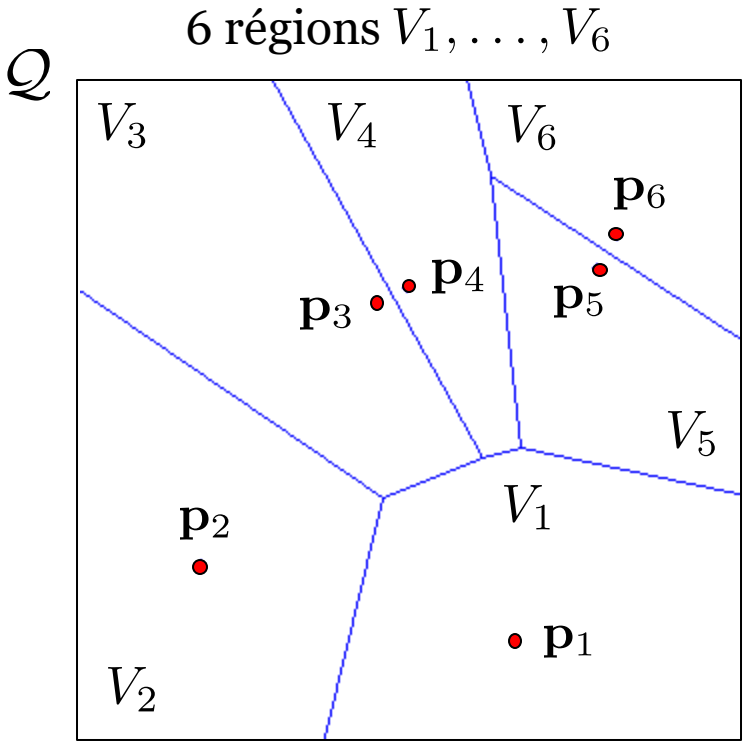
4. Graphe de Voronoï

Graphe de Voronoï

- Soit un ensemble de n points différents $\mathbf{p}_i$ (appelés *générateurs* ou *germes*) dans un environnement $\mathcal{Q} \subset \mathbb{R}^2$
- Le diagramme de Voronoï partitionne $\mathcal{Q}$ en régions

$$V_i,\ i \in \{1, \ldots, n\}$$

les plus proches de chaque point $\mathbf{p}_i$



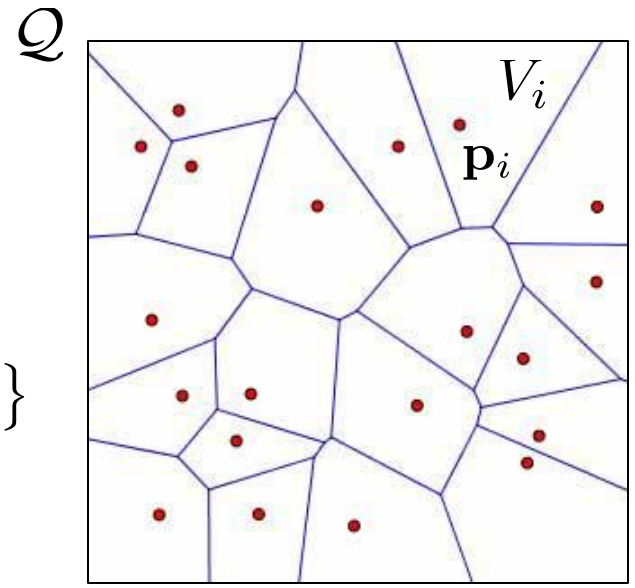
Georgy F. Voronoï (1868-1908)
mathématicien russe

Graphe de Voronoï

- Plus formellement, la région V_i est définie par :

$$V_i = \{\mathbf{z} \in \mathcal{Q} \mid \|\mathbf{z} - \mathbf{p}_i\|_2 \leq \|\mathbf{z} - \mathbf{p}_j\|_2, \forall j \neq i\}$$

où $\|\cdot\|_2$ indique la norme Euclidienne



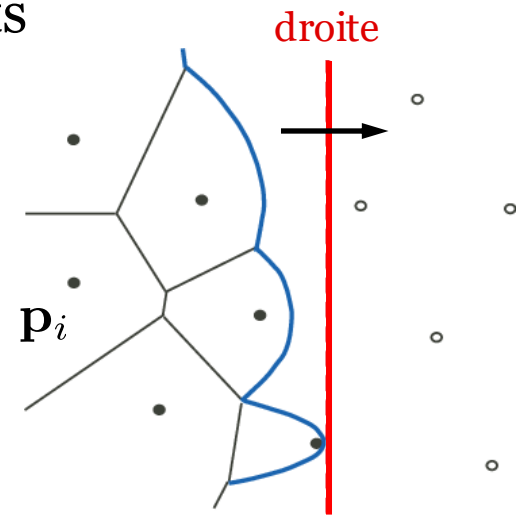
Exemple Matlab : *Plot_Voronoi.m*

- Le *diagramme de Voronoï* est l'ensemble des régions $\{V_1, \ldots, V_n\}$
- Le **graphe de Voronoï** est défini par la *frontière* des régions $V_1, \ldots, V_n$ (bleue dans la figure)

Graphe de Voronoï

- Plusieurs algorithmes existent pour le calcul du diagramme de Voronoï d'un ensemble de n points

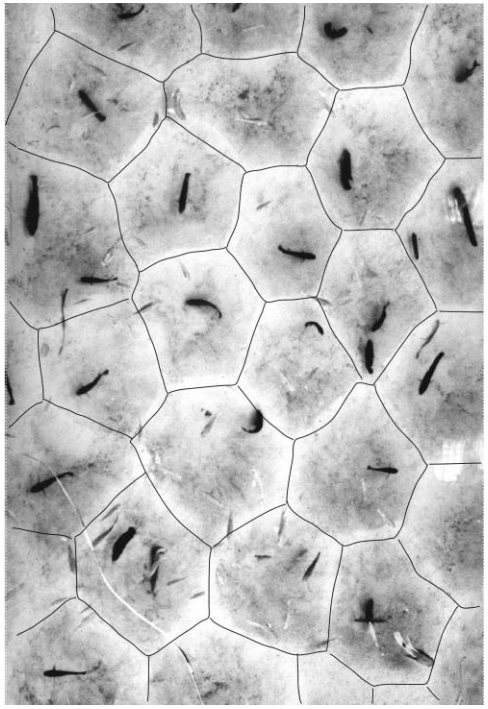
- L'**algorithme de Fortune** est un *algorithme de balayage* (« sweepline ») : une droite balaie les points dans une certaine direction, l'algorithme met à jour la construction, et lorsque tous les points ont été balayés, le diagramme est construit



- Complexité de l'algorithme de Fortune:
  - $O(n \log n)$ en temps
  - $O(n)$ en espace mémoire

"*Voronoi Diagrams and Delaunay Triangulations*", S. Fortune, in Ch. 27 of Handbook of Discrete and Computational Geometry, 3rd ed., CRC press, 2018

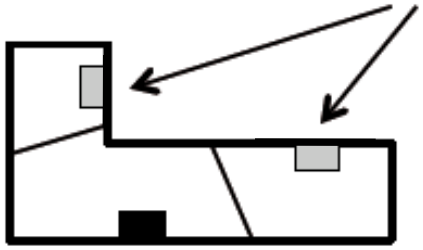
Graphe de Voronoï : exemples et applications

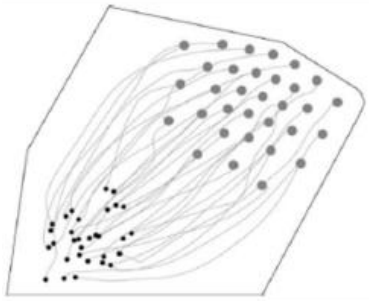


Salar d'Uyuni, désert en Bolivie (3700 m)

- Le poisson *Tilapia mossambica* crée des diagrammes de Voronoï au cours du processus d'élevage des petits (voir l'image à gauche)
- Les diagrammes de Voronoï sont utilisés dans plusieurs domaines (biologie, géographie, hydrologie, reconnaissance des formes, infographie, recherche opérationnelle, etc.) et pour de nombreuses applications:
  - Localisation de nouvelles installations
  - Couverture optimale d'une région limitée

Emplacement de nouveaux magasins dans un quartier

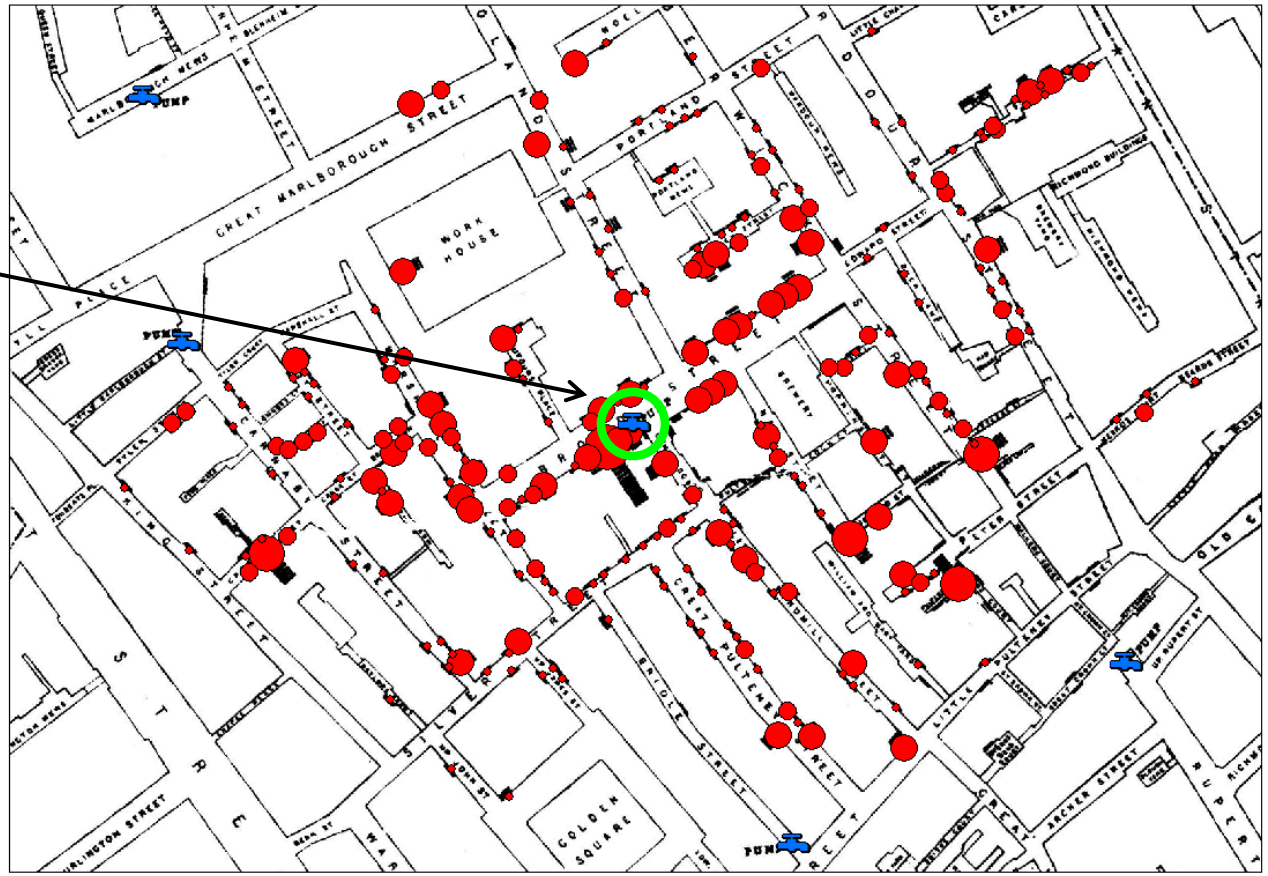




Couverture d'une région par une cohorte de robots [Cortés *et al.*, TRA'04]

Graphe de Voronoï : exemples et applications

Pompe d'eau infecte entre Broad Street et Cambridge Street à Londre



Le médecin John Snow utilisa un diagramme de Voronoï pour montrer que la majorité des personnes décédées à Londre suite à l'épidémie de choléra de 1854 (disques rouges), vivait plus près d'une pompe d'eau infecte (cercle vert) que de toutes les autres pompes (bleues)

Graphe de Voronoï : extensions

a) Graphe de Voronoï 3-D

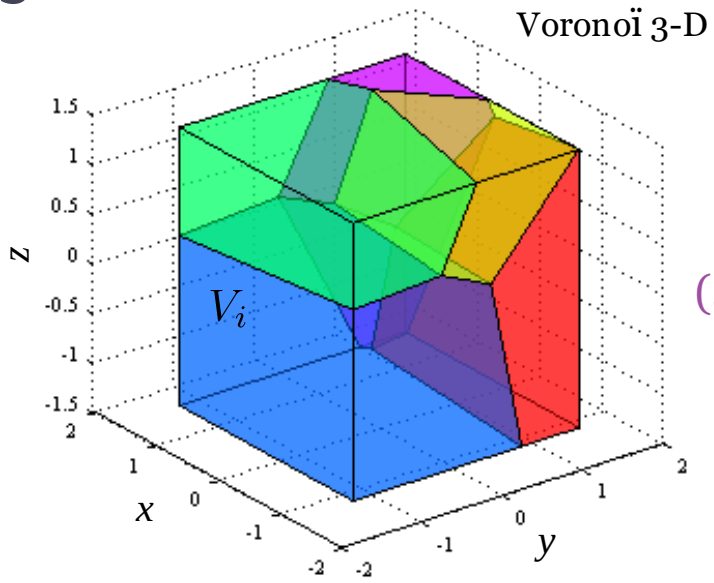
b) Graphe de Voronoï généralisé défini à partir de générateurs de dimension 1 ou 2 (*segments de droite* ou *surfaces*)

c) Graphe de Voronoï d'étendue r : chaque région de Voronoï est définie par:

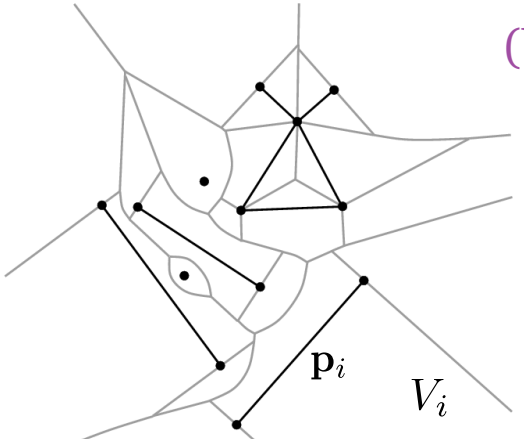
$$V_{i,r} = V_i \cap \overline{B}(\mathbf{p}_i, r)$$

où $\overline{B}(\mathbf{p}_i, r) = \{\mathbf{z} \in \mathcal{Q} \mid \|\mathbf{z} - \mathbf{p}_i\|_2 \leq r\}$ est un disque de rayon r centré sur $\mathbf{p}_i$

Graphe de Voronoï 3-D

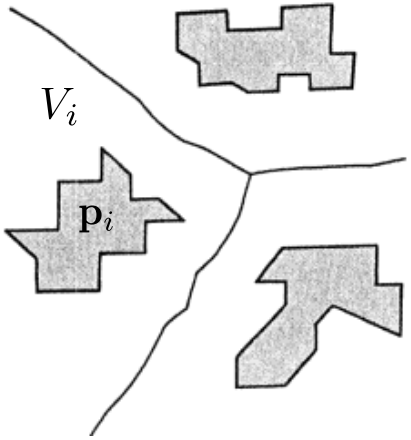


(a)



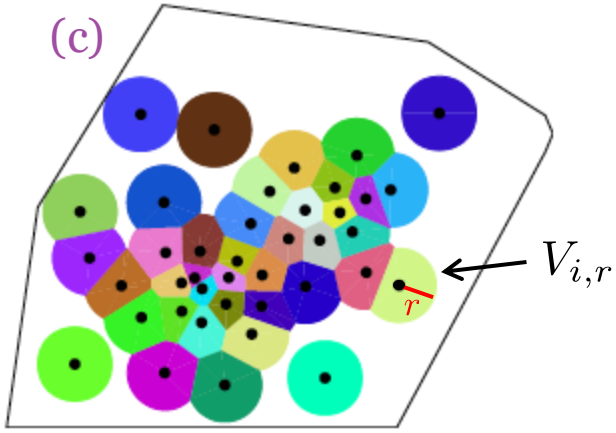
Graphe de Voronoï : Générateurs de dim. 1 (segments)

(b)



Graphe de Voronoï : Générateurs de dim. 2 (surfaces)

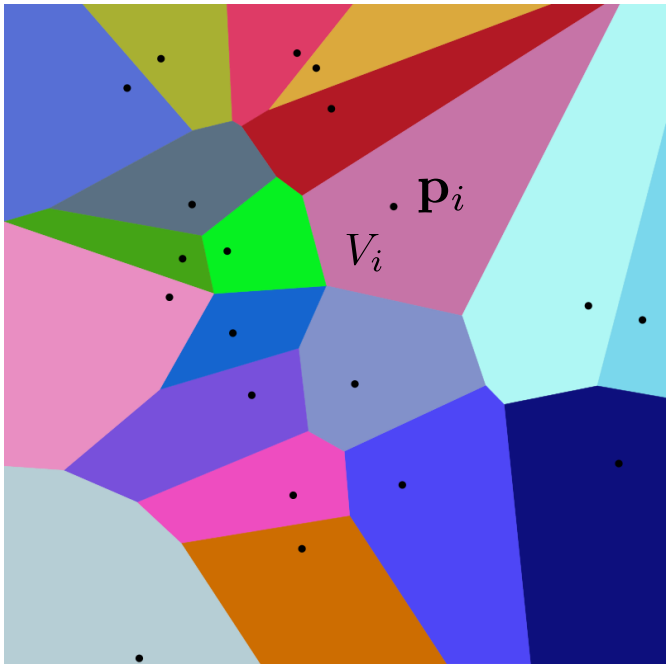
(c)



Graphe de Voronoï d'étendue r

Graphe de Voronoï : extensions

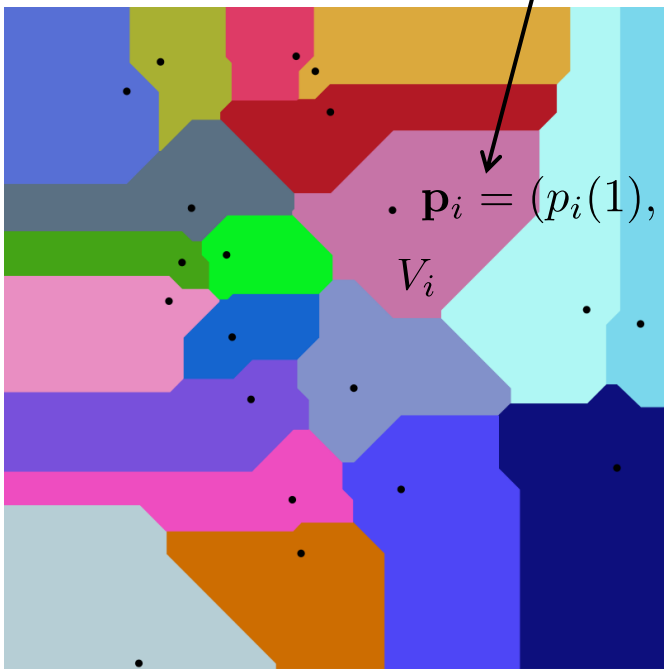
- Fonctions de distance autres que le *distance euclidienne*



Distance euclidienne

$$\mathrm{dist}(\mathbf{z}, \mathbf{p}_i) = \|\mathbf{z} - \mathbf{p}_i\|_2$$

Région *non convexe* !



Distance de Manhattan ou distance ℓ_1

$$\mathrm{dist}(\mathbf{z}, \mathbf{p}_i) = \|\mathbf{z} - \mathbf{p}_i\|_1 \triangleq \sum_{j=1}^{2} |z(j) - p_i(j)|$$

"*Spatial tessellations: concepts and applications of Voronoi diagrams*", A. Okabe, B.N. Boots, K. Sugihara, S.N. Chiu, Wiley & Sons, 1992

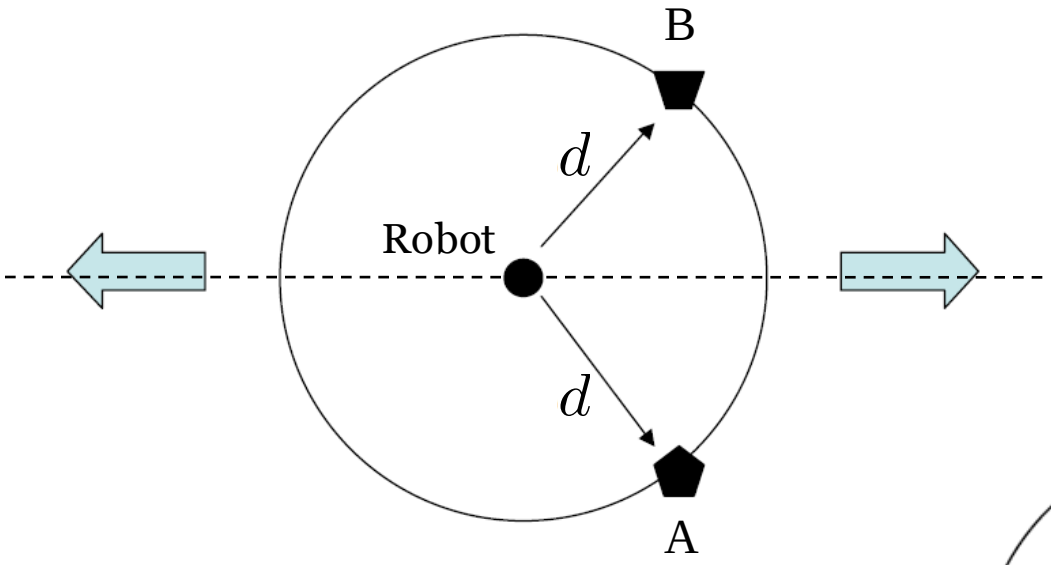
Graphe de Voronoï

Evitement d'obstacles

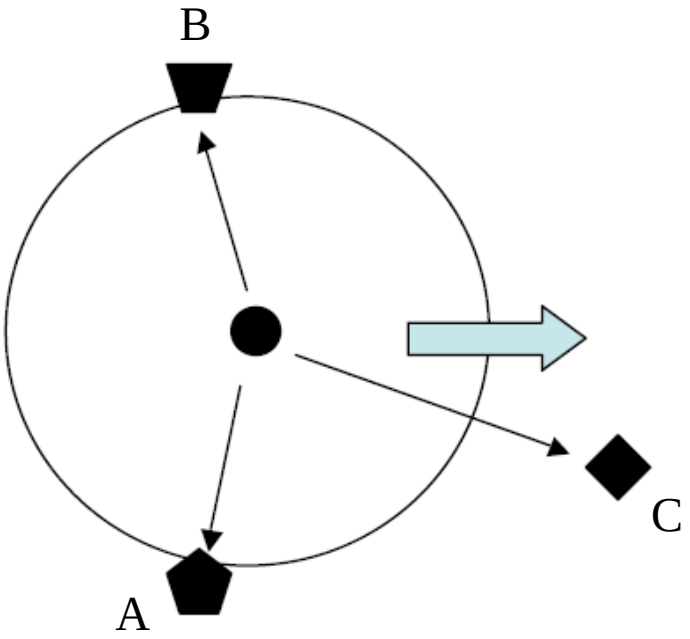
- On représente le robot par un point qui perçoit un ensemble d'obstacles (*ponctuels*) tout autour de lui
  - Distance des m obstacles les plus proches : $d_1, \ldots, d_m$
  - Angle vers les obstacles : $\alpha_1, \ldots, \alpha_m$
- Loi de contrôle du mouvement :
  - Quand **2 obstacles** sont détectés : se déplacer dans la direction $\gamma = (\alpha_1 + \alpha_2)/2$ (suivre la *ligne médiane*)
  - Quand il y a **3 obstacles** proches ou plus: définir un « *lieu* »

Graphe de Voronoï

- Mouvement selon une arête du graphe de Voronoï

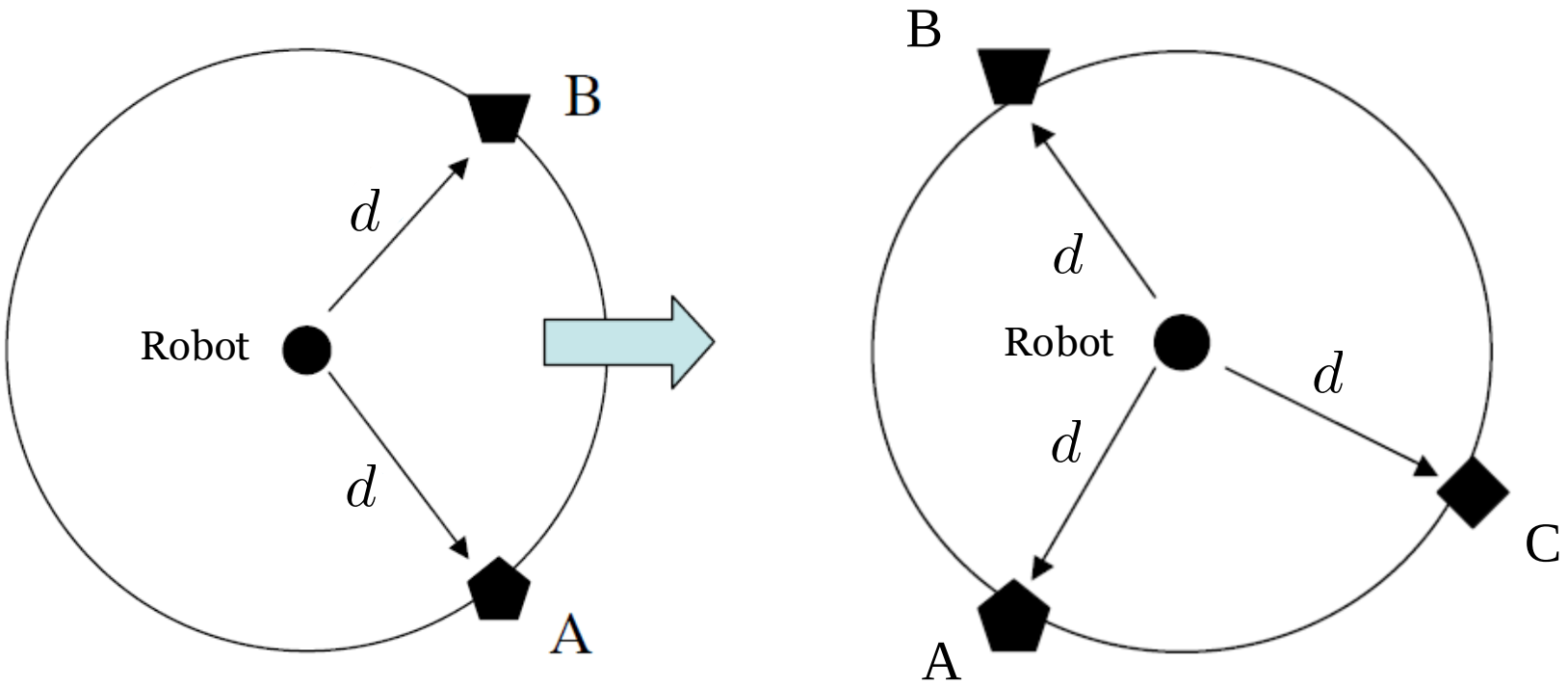


- Détection d'un troisième obstacle



Graphe de Voronoï

- Déplacement à équidistance des trois obstacles et définition d'un *lieu*



Graphe de Voronoï

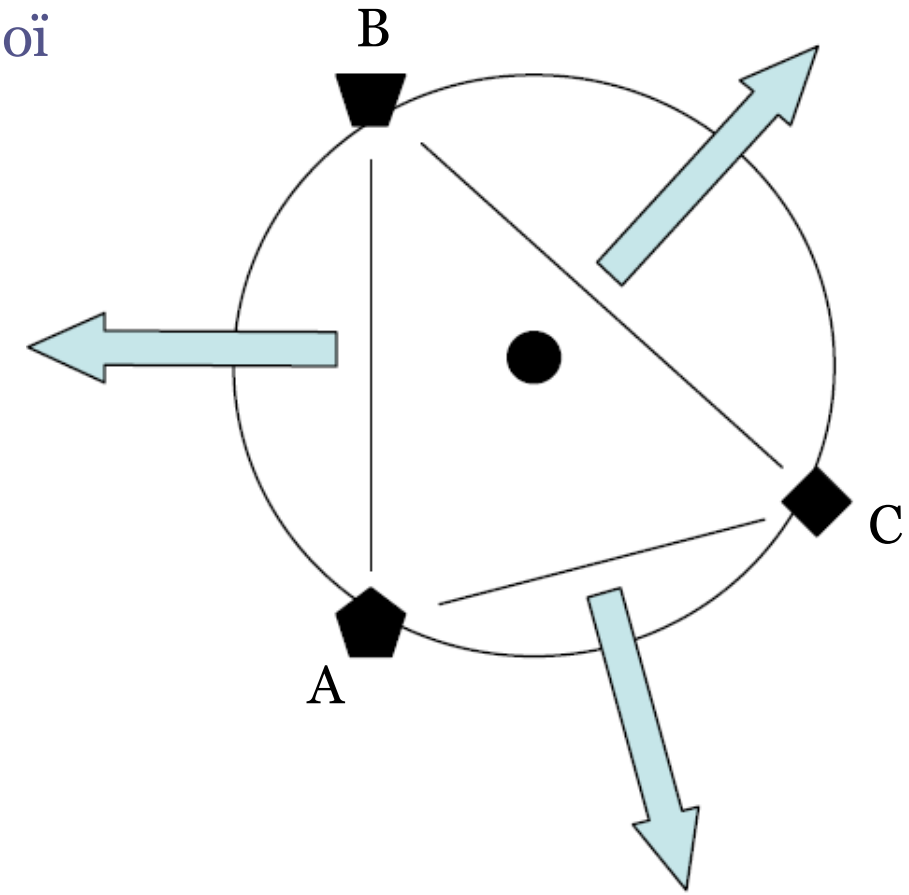
Algorithme

1. On part du point $\mathbf{q}_s$
2. Se déplacer en maintenant une distance égale entre les obstacles A et B : $d_A(t) = d_B(t)$
3. Sélectionner un obstacle C avec une distance $d_C(t)$ telle que : $d_C(t) = d_A(t) = d_B(t)$
   - Utiliser une *tolérance* sur l'égalité
   - Éviter les cas où ces distances ne sont jamais égales
   - Attention au cas où les (une des) distances sont égales à la *distance maximale perceptible* du robot
4. Arrêt et définition d'un *lieu*

Graphe de Voronoï

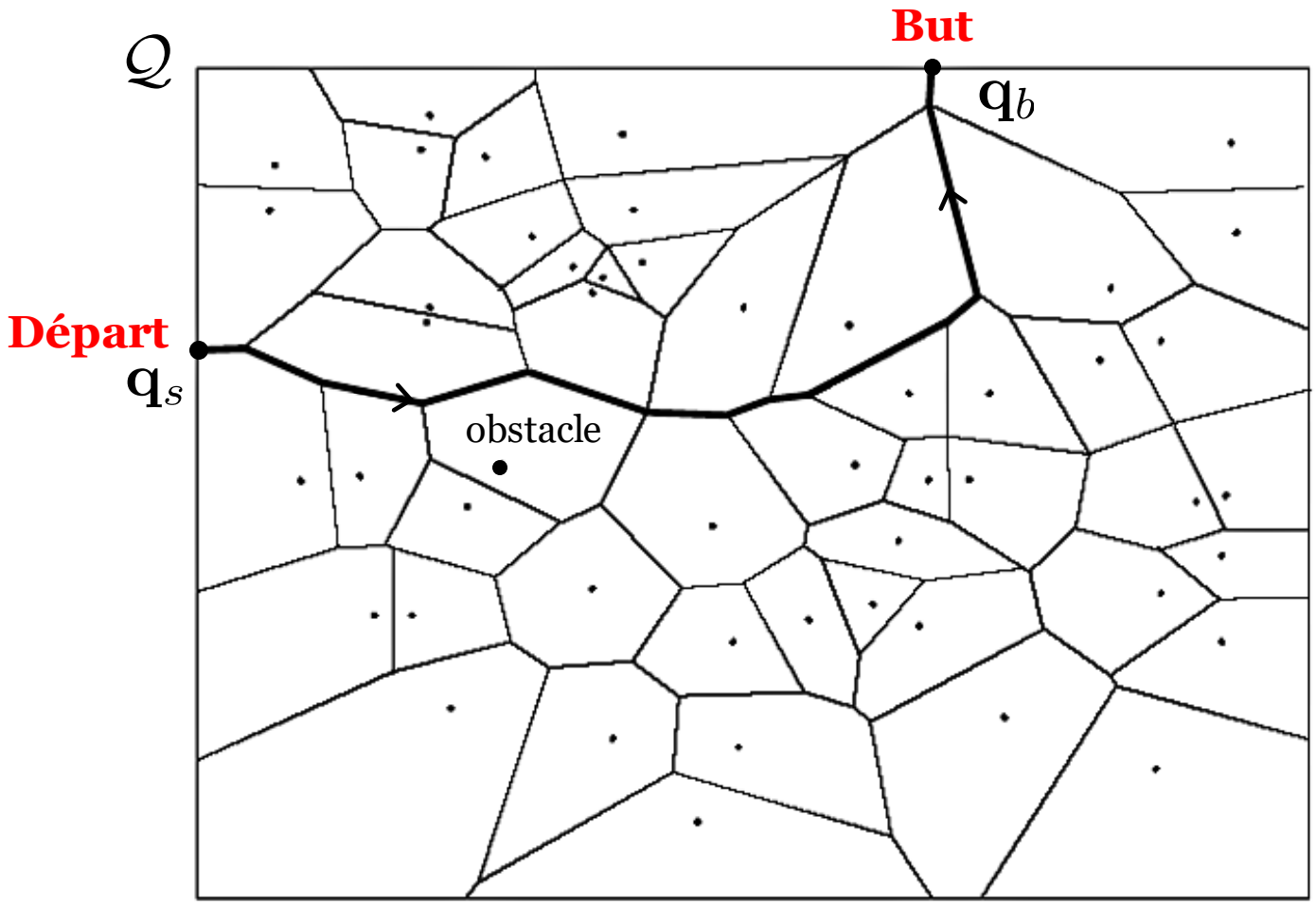
5. Dès que le robot est sur un *lieu*
   - Sélectionner une arête de sortie du graphe de Voronoï (selon un certain critère)
   - Effectuer une rotation pour faire face à l'arête

6. Lancer le robot sur la nouvelle arête, et répéter la procédure précédente (« go to 3. ») jusqu'au but q_b



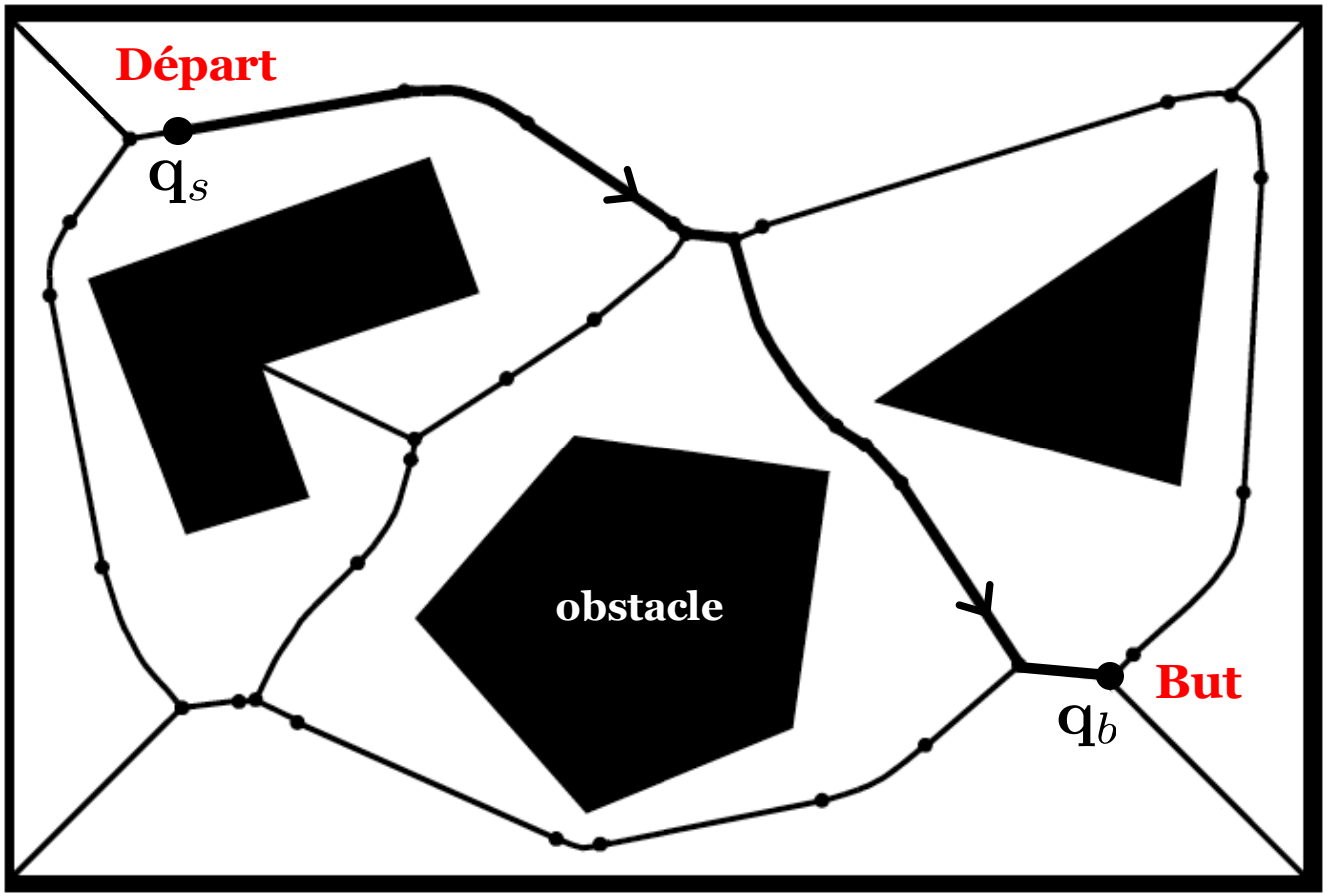
Graphe de Voronoï

- Exemple de chemin sans collisions (il n'est pas unique !)



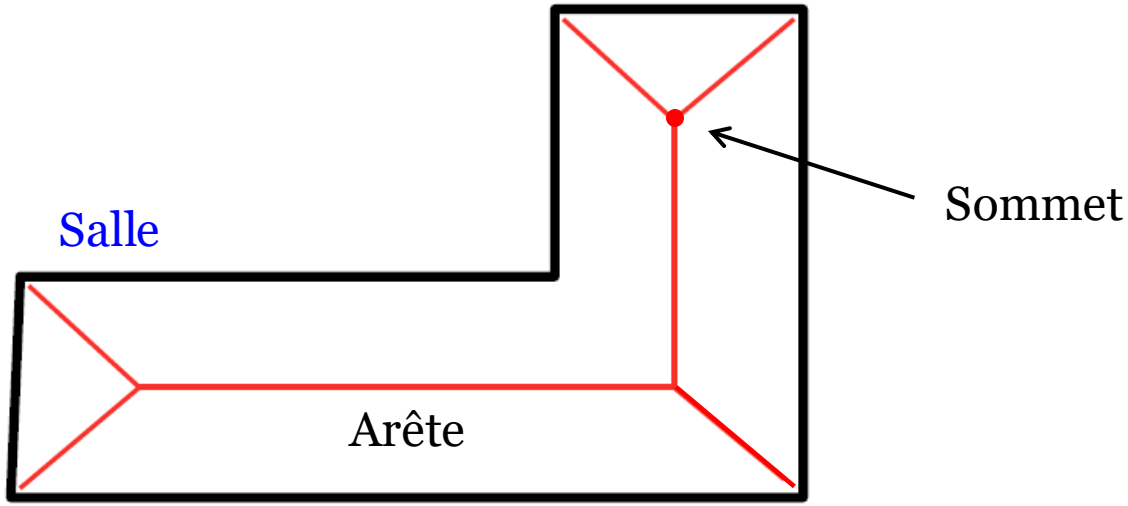
Graphe de Voronoï

- **Extension 1** : obstacles *non ponctuels* (polygones)



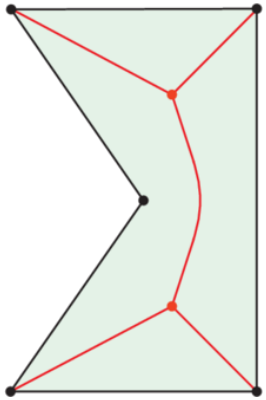
Graphe de Voronoï

- **Extension 2** : obstacles *non ponctuels* (par ex. les murs d'une salle)



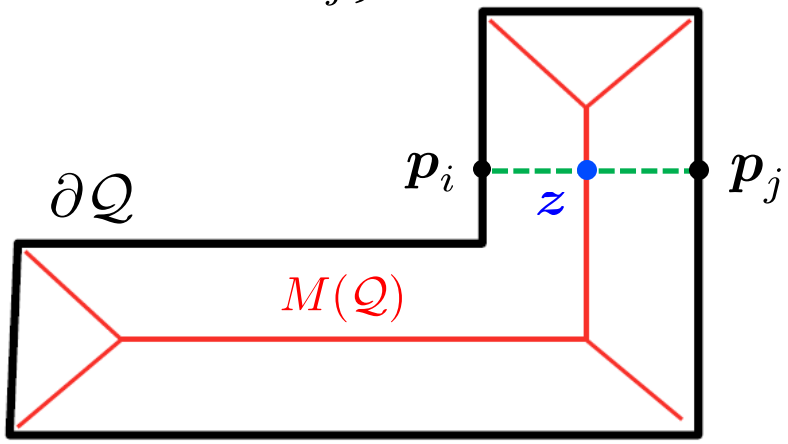
  - *Arête* du graphe de Voronoï :
    - Ensembles de points équidistants de deux bordures
  - *Sommet* du graphe de Voronoï :
    - Ensembles de points équidistants de trois bordures ou plus

Graphe de Voronoï



- L'**axe médian** (« *medial axis* ») permet de représenter la forme d'une région en trouvant son squelette topologique, c'est-à-dire un ensemble de courbes qui court le long du « milieu » de la région
- Soit $\partial\mathcal{Q}$ la frontière de l'environnement $\mathcal{Q}$. L'*axe médian* $M(\mathcal{Q})$ de $\mathcal{Q}$ est défini par :

$$M(\mathcal{Q}) = \{\boldsymbol{z} \in \mathcal{Q} \mid \|\boldsymbol{z} - \boldsymbol{p}_i\|_2 = \|\boldsymbol{z} - \boldsymbol{p}_j\|_2 = \min_{\boldsymbol{y} \in \partial\mathcal{Q}} \|\boldsymbol{z} - \boldsymbol{y}\|_2, \; \boldsymbol{p}_i, \boldsymbol{p}_j \in \partial\mathcal{Q}, \; \boldsymbol{p}_i \neq \boldsymbol{p}_j\}$$



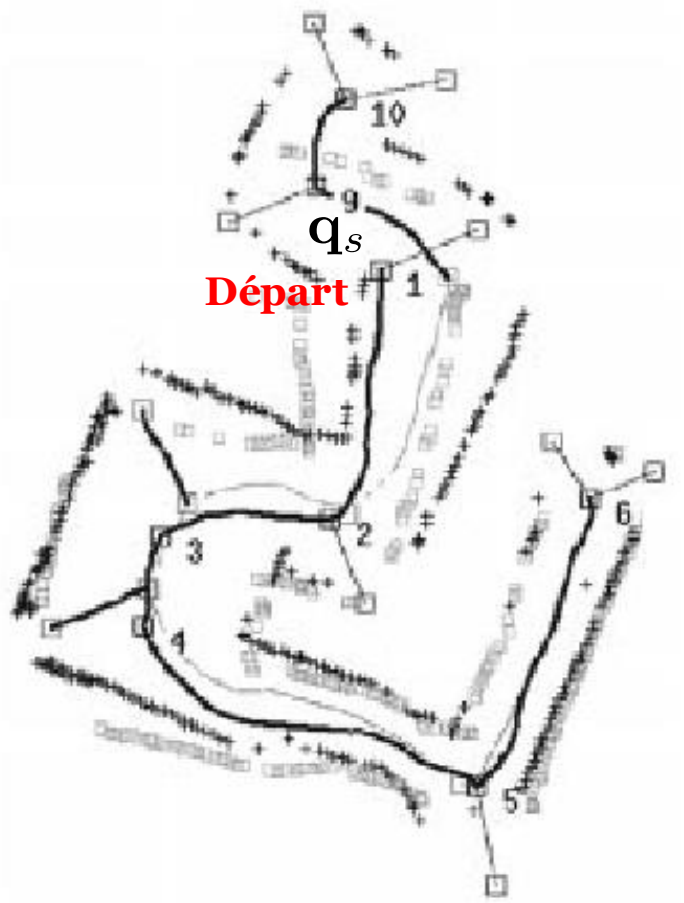
Graphe de Voronoï

Autre application du graphe de Voronoï en robotique mobile :

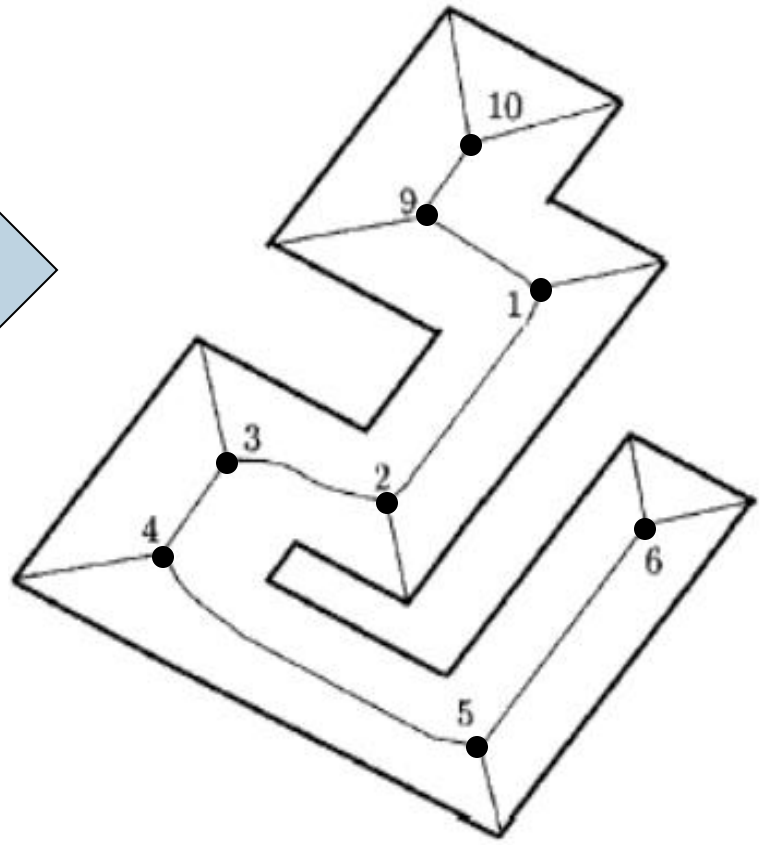
- Construction de **cartes topologiques** (cf. le Ch. 1.3)
  - Démarrage
    - Trouver l'objet le plus proche
    - Se déplacer jusqu'à trouver un second objet
    - Suivre *la ligne médiane* vers un troisième objet
    - Définition du *lieu* initial
  - Tant qu'il existe une arête inexplorée
    - Suivre cette arête vers le *lieu* situé à l'autre bout
  - Arrêt quand toutes les arêtes ont été explorées

Graphe de Voronoï

- Exemple



Construction de la carte topologique d'un bâtiment (8 lieux)



Partie 4 : Planification de trajectoire et évitement d'obstacles

5. Planification probabiliste

Méthodes déterministes et probabilistes

Méthodes déterministes (champs de potentiel, fenêtre dynamique, graphe de Voronoï)

- Elles permettent de retrouver le même chemin sans collisions *à chaque exécution*, sous réserve d'avoir des conditions initiales équivalentes
- Les méthodes déterministes sont dites *complètes en résolution* car elles garantissent de trouver une solution ou d'indiquer *s'il n'y a pas de solution*

Méthodes probabilistes

- Ces méthodes ne trouveront pas forcément le même chemin sans collisions *à chaque exécution*, même avec les mêmes conditions initiales
- Ces méthodes ne sont pas complètes en résolution, mais elles garantissent de trouver un chemin sans collisions s'il en existe un. On dit qu'elles sont *complètes en probabilité*

Planification probabiliste

Les *planificateurs probabilistes* font partie de la grande famille des *méthodes basées sur l'échantillonnage* (« sampling-based methods », en anglais)

- Très efficaces, spécialement pour problèmes définis dans un espace des configurations $\mathcal{C}$ de *très haute dimensionalité*

Idée de base des méthodes basées sur l'échantillonnage :
Déterminer un ensemble fini de configurations sans collisions qui représentent adéquatement l'*espace libre* dans l'environnement et utiliser ces configurations pour construire une "*roadmap*" (carte de route) entre deux poses données $\mathbf{q}_s$ et $\mathbf{q}_b$ du robot

- À chaque itération on choisit une *configuration candidate* et on vérifie qu'elle ne comporte pas de collisions entre le robot et les obstacles

- Si il y a une collision, la configuration candidate est rejetée. Sinon, elle est rajoutée à la roadmap et elle est connectée, si possible, aux autres configurations déjà mémorisées

Planification probabiliste

Cette procédure est assez générale. Il y a deux façons de choisir les candidats :

- *Approche déterministe* : les configurations candidates sont choisies à travers une *grille régulière* appliquée à l'environnement (voir, par ex. [Janson *et al.*, IJRR'18])
- *Approche stochastique* : les configurations candidates sont choisies selon une certaine *fonction de densité de probabilité* (pdf)

Deux méthodes importantes qui utilisent l'*approche stochastique* sont :

1. **PRM** (Probabilistic Roadmap)
2. **RRT** (Rapidly-exploring Random Tree)

Remarque: l'étude de l'*optimalité* de ces méthodes est assez récente (elle s'appuie sur la théorie des graphes aléatoires et de la percolation), voir les articles:

"*Sampling-based algorithms for optimal motion planning*", S. Karaman, E. Frazzoli, Int. Journal of Robotics Research, vol. 30, no. 7, pp. 846-894, 2011

"*Exploring implicit spaces for constrained sampling-based planning*", Z. Kingston, M. Moll, L.E. Kavraki, Int. Journal of Robotics Research, vol. 38, no. 10-11, pp. 1151–1178, 2019

Planification probabiliste : PRM

1. Probabilistic Roadmap (PRM)

"*Probabilistic roadmaps for path planning in high-dimensional configuration spaces*", L.E. Kavraki, P. Svestka, J.-C. Latombe, M.H. Overmars, IEEE Trans. Robotics and Automation, vol. 12, n. 4, pp. 566-580, 1996

Algorithme :

- On génère un échantillon aléatoire de l'espace des configurations $\mathbf{q}_{\text{rand}}$ en utilisant une *pdf uniforme*
- $\mathbf{q}_{\text{rand}}$ est testé pour les collisions
- Si $\mathbf{q}_{\text{rand}}$ ne génère pas de collisions, il est ajouté à la *roadmap* (PRM) et connecté (si possible) à travers des *chemins locaux sans collisions*, à des configurations 'suffisamment' proches déjà présentes dans la roadmap

Planification probabiliste : PRM

- La génération d'un chemin local sans collisions entre $\mathbf{q}_{\text{rand}}$ et une configuration proche $\mathbf{q}_{\text{near}}$ est effectuée par un *planificateur local*. Typiquement, on considère un chemin droit entre $\mathbf{q}_{\text{rand}}$ et $\mathbf{q}_{\text{near}}$, et on vérifie s'il y a des collisions (par ex. on vérifie que les échantillons individuels du segment ne génèrent pas de collisions)

- Si le chemin droit provoque une collision, il est rejeté et $\mathbf{q}_{\text{rand}}$ et $\mathbf{q}_{\text{near}}$ ne sont pas connectés dans la roadmap

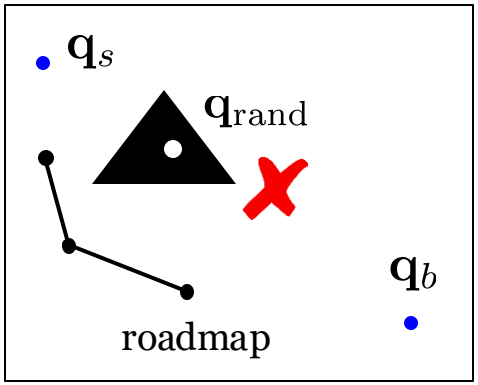
- La procédure incrémentale de la méthode PRM s'arrête lorsque un *nombre maximal d'itérations* a été atteint ou le *nombre de composantes connexes* dans la roadmap dévient inférieur à un seuil-limite

- On vérifie si il est possible de connecter $\mathbf{q}_s$ et $\mathbf{q}_b$ à la *même* composante connexe de la PRM en utilisant des chemins locaux sans collisions

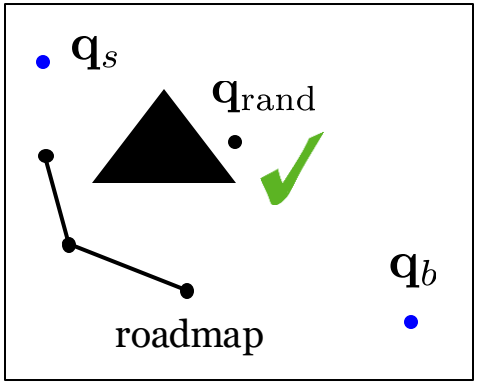
Remarque : si une solution n'a pas été trouvée, la PRM peut être améliorée en effectuant *plus d'itérations* ou en utilisant des stratégies pour *réduire le nombre de composantes connexes*

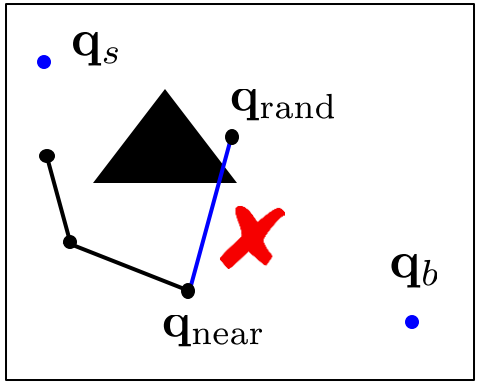
PRM : illustration de l'algorithme (3 cas possibles: A, B et C)

A

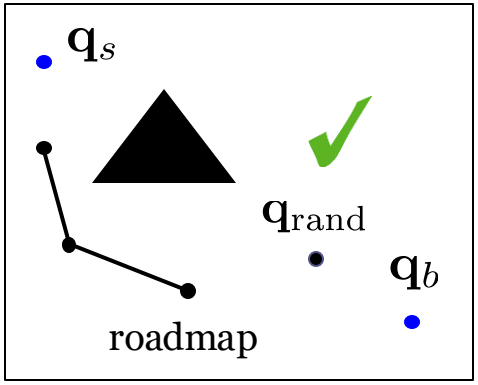


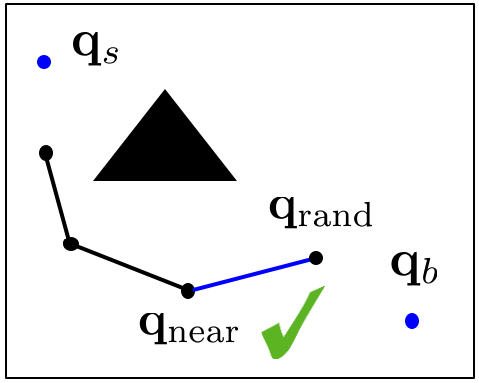
B





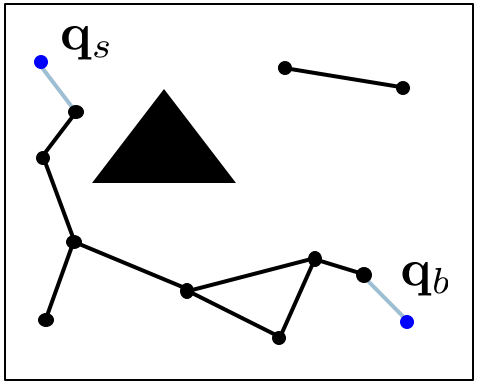
C





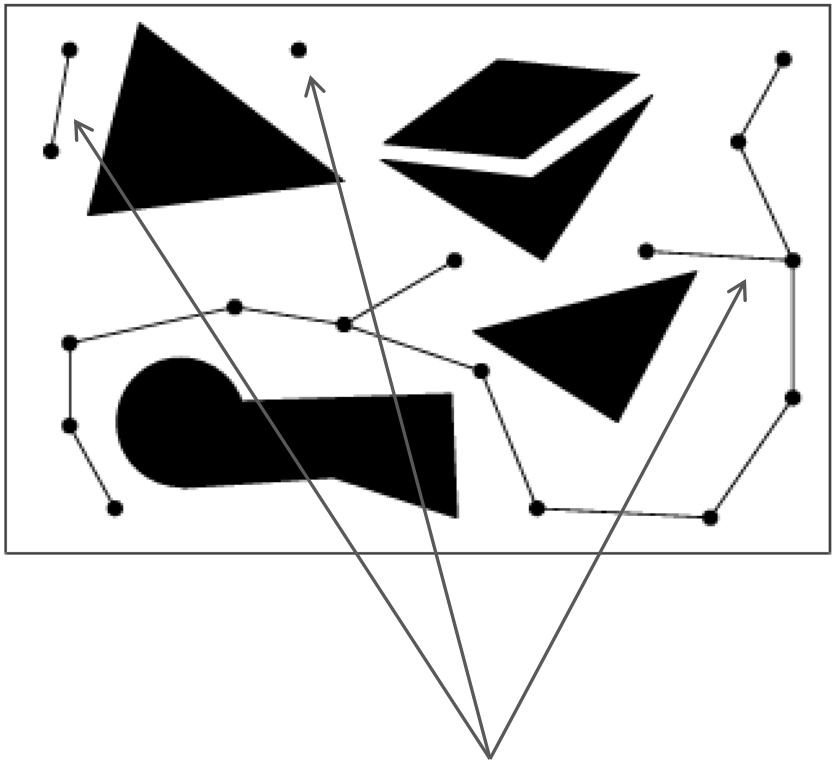
• • •

2 composantes connexes de la PRM

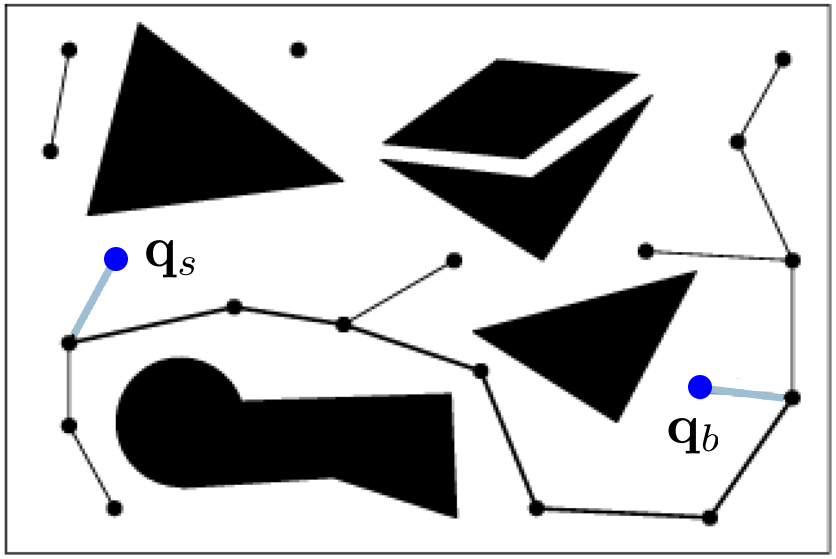


Planification probabiliste : PRM

Exemple



3 composantes
connexes de la PRM



Utilisation de la PRM
pour trouver un chemin
sans collisions entre $\mathbf{q}_s$ et $\mathbf{q}_b$

Planification probabiliste : PRM

Avantages de la méthode PRM :

- Elle trouve un chemin sans collisions *très rapidement*
  (à condition que la PRM ait été bien développée)

- Nouvelles instances du même problème produisent une amélioration potentielle de la PRM. La PRM améliore, en termes de connexité et d'efficacité temporelle, avec l'utilisation (la méthode PRM est donc intrinsèquement « multiple-query » ou à *demande multiple*)

- Dans des espaces à *très haute dimensionnalité* (dimension > 4)
  la méthode PRM est très efficace pour trouver rapidement une solution

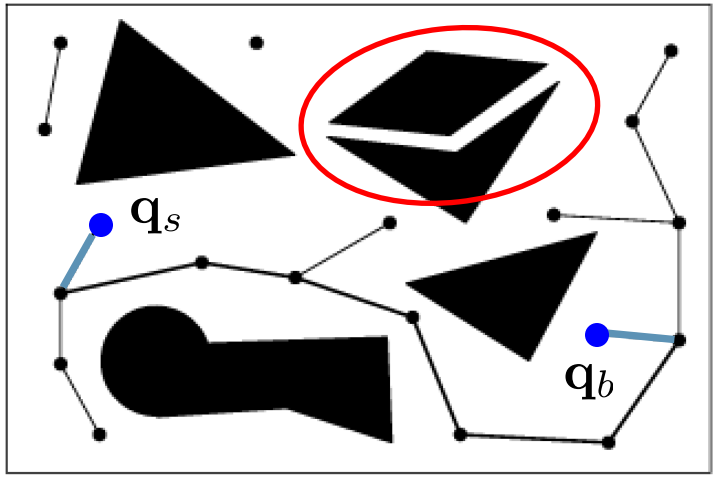
- *Simplicité* de mise en œuvre : il n'est pas nécessaire d'avoir
  une représentation géométrique des obstacles

Planification probabiliste : PRM

Inconvénients de la méthode PRM

- PRM est seulement *complet en probabilité* : la probabilité de trouver une solution (s'il y a une) tend vers 1 comme le temps d'exécution tend vers l'infini. Donc, s'il n'y a pas de solutions, l'algorithme ne terminera *jamais* : en pratique, un *nombre maximum d'itérations* est fixé pour garantir la terminaison
- Les *passages étroits* dans l'environnement sont critiques. En utilisant une *pdf uniforme* pour générer q_{rand}, la probabilité de placer un candidat dans une certaine région de l'espace libre est proportionnelle à son *volume*. Par conséquent, il serait peu probable d'avoir un chemin à travers un *passage étroit* dans un délai raisonnable

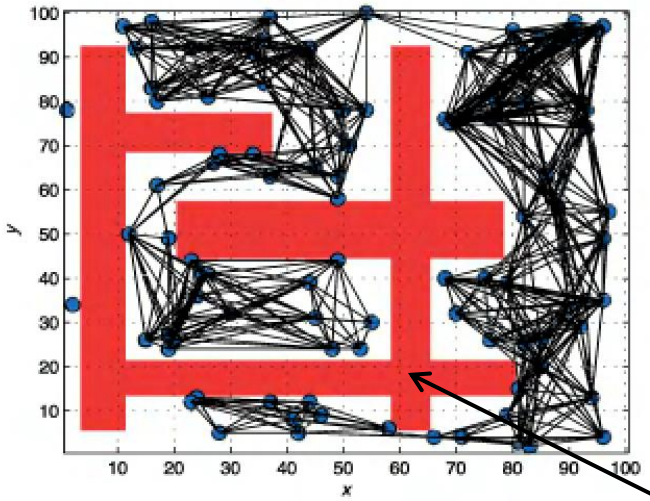
  Solution possible : utiliser une pdf *non uniforme*



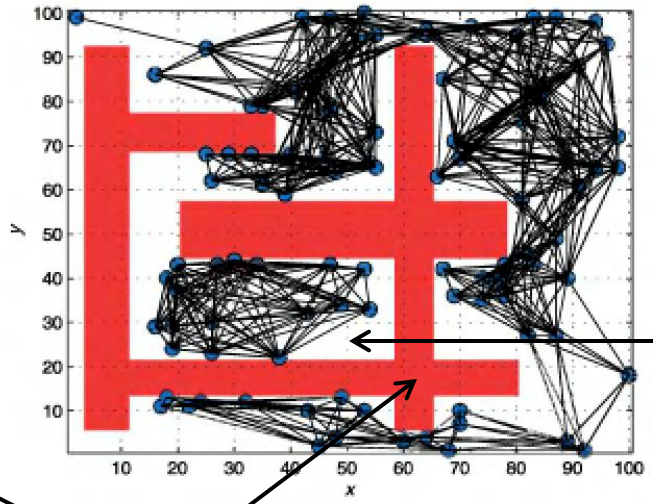
- *Contrôle de collision : chronophage*. Avec Lazy PRM [Bohlin & Kavraki, ICRA'00], on a une réduction du nombre de contrôles pendant la planification et par conséquent une minimisation du temps d'exécution

Planification probabiliste : PRM

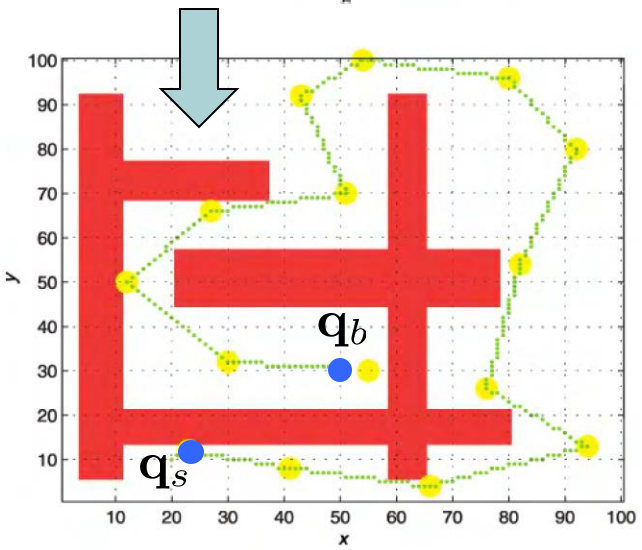
Exécution 1



Exécution 2



Deux composantes connexe de la PRM



Chemin sans collisions du robot choisi (vert)

Obstacle

"*Robotics, Vision and Control: Fundamental Algorithms in MATLAB*", P. Corke, Springer, 2011

Robotics Toolbox de P. Corke :

```
>> prm.plan()
```

Dans la *Robotics System Toolbox* de Matlab (R2017) :

```
>> PathPlanningExample.m
```

Planification probabiliste : RRT

2. Rapidly-exploring Random Tree (RRT)

"Randomized Kinodynamic Planning", S.M. LaValle, J.J. Kuffner Jr., Int. Journal of Robotics Research, vol. 20, n. 5, pp. 378-400, 2001

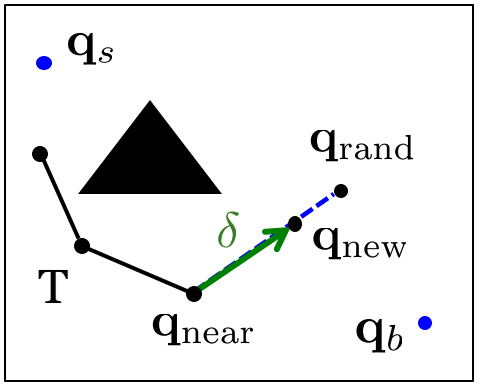
- RRT est un algorithme de planification probabiliste « single-query »
  - L'algorithme ne génère pas une roadmap qui représente d'une façon exhaustive la connexité de l'espace libre dans tout l'environnement. On explore uniquement une *portion de l'espace libre* qui est pertinente à la solution du problème (ce qui engendre une forte réduction du *temps de calcul*)

- La méthode RRT utilise une structure de données qui s'appelle *Rapidly-exploring Random Tree* (RRT)
  - L'expansion incrémentale de l'RRT (on appelle cet *arbre* « T ») est basée sur une simple procédure stochastique répétée à chaque itération

Planification probabiliste : RRT

Algorithme

- On génère un échantillon aléatoire de l'espace des configurations $\mathbf{q}_{\text{rand}}$ en utilisant une *pdf uniforme* (comme pour la méthode PRM)
- La configuration $\mathbf{q}_{\text{near}}$ en T la plus proche à $\mathbf{q}_{\text{rand}}$ est déterminée, et une nouvelle configuration candidate $\mathbf{q}_{\text{new}}$ est produite sur le *segment joignant* $\mathbf{q}_{\text{near}}$ à $\mathbf{q}_{\text{rand}}$, à une distance préfixée δ de $\mathbf{q}_{\text{near}}$
- On vérifie que soit $\mathbf{q}_{\text{new}}$ soit le segment de $\mathbf{q}_{\text{near}}$ à $\mathbf{q}_{\text{new}}$ n'engendre pas de collisions. Si tel est le cas, T est élargi en incluant $\mathbf{q}_{\text{new}}$ et le segment le joignant à $\mathbf{q}_{\text{near}}$ ($\mathbf{q}_{\text{rand}}$ n'est pas rajouté à T, donc il n'est pas nécessaire de vérifier s'il engendre des collisions : sa seule fonction est d'indiquer la direction d'expansion de l'arbre T)



Planification probabiliste : RRT

- Pour *accélérer* la recherche d'un chemin entre $\mathbf{q}_s$ et $\mathbf{q}_b$, la méthode RRT bidirectionnelle utilise deux arbres T_s et T_b ancrés en $\mathbf{q}_s$ et $\mathbf{q}_b$, respectivement. À chaque itération, les deux arbres sont élargis avec la méthode décrite dans la slide précédente

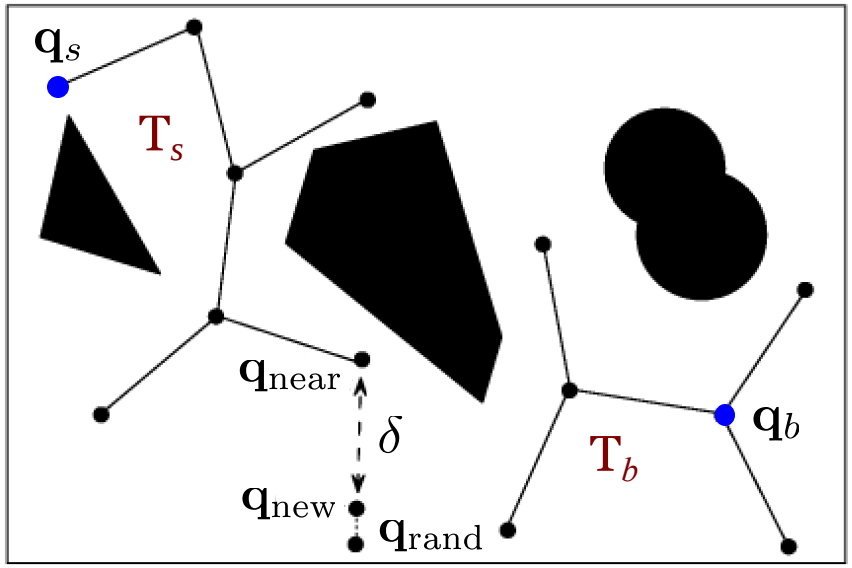
- Après un certain nombre de pas d'expansion, l'algorithme entre dans une phase où il essaie de connecter T_s et T_b en étendant chaque arbre vers l'autre. Ceci est fait en générant un $\mathbf{q}_{\text{new}}$ comme une expansion de T_s et en essayant de connecter T_b à $\mathbf{q}_{\text{new}}$

- Si le segment joignant T_s et T_b est sans collisions, l'expansion est *complète* et les deux arbres sont raccordés

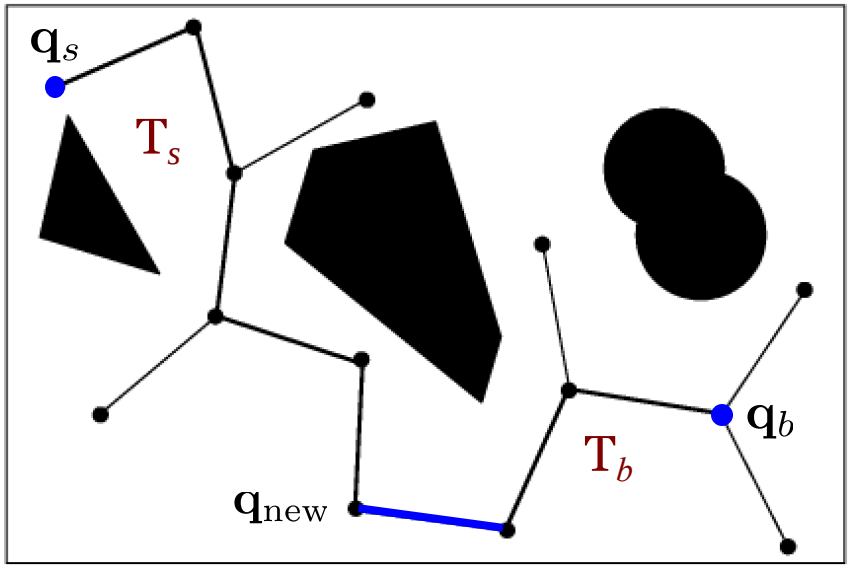
- Si la procédure de connexion n'aboutisse pas à une solution après un certain nombre d'itérations, on peut conclure que les deux arbres sont encore *distants* et la phase d'expansion peut recommencer

Planification probabiliste : RRT

- Illustration de la méthode RRT bidirectionnelle ou « *RRTConnect* »



Procédure stochastique pour l'expansion d'un arbre (T_s)



Les arbres T_s et T_b ont été raccordés grâce au arc bleu

Planification probabiliste : RRT

Avantages de la méthode RRT

- La procédure d'expansion de RRT est *simple* et *très efficace* pour « explorer » l'espace libre dans un environnement (même non convexe). Elle est biaisée vers les parties de l'environnement qui n'ont pas été encore visitées
- La probabilité que une configuration générique de l'espace libre soit rajoutée à le RRT tend vers 1 comme le temps d'exécution tend vers l'infini (à condition que la configuration se trouve dans la même composante connexe de l'espace libre où le RRT est ancré)
- Comme la méthode PRM, la méthode RRT est **complète en probabilité**

Plusieurs variantes possibles :

- Au lieu d'utiliser une δ constante pour générer $\mathbf{q}_{\text{new}}$, on peut définir un pas qui est une *fonction de l'espace libre disponible* (on peut aller jusqu'à $\mathbf{q}_{\text{rand}}$, comme dans la phase expansion de la méthode PRM). Cette version *gloutonne* (« greedy ») de la méthode peut être beaucoup plus efficace, si il y a des vastes zones vides
- RRT* garantie la convergence vers un chemin qui est optimal par rapport à une certaine fonction de coût $c : \Sigma_{\mathcal{C}} \rightarrow \mathbb{R}_{>0}$ [Karaman & Frazzoli, IJRR'11]
- On peut facilement utiliser la méthode RRT avec des *contraintes différentielles* (non-holonômes et cinématiques/dynamiques) sur le robot

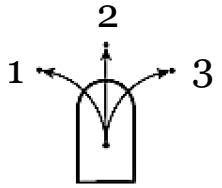
Planification probabiliste : RRT

Extension pour robots non-holonômes

Pour un robot de type unicycle, les chemins en ligne droite pour aller de $\mathbf{q}_s$ à $\mathbf{q}_b$ générés par la méthode RRT, *ne sont pas admissibles*, en général

- **Solution simple et générale** : utiliser des primitives de mouvement
  - Elles sont un ensemble fini et admissible de chemins locaux dans l'espace des configurations
  - Chaque primitive est produite par un *choix spécifique des vitesses* dans le modèle cinématique
  - Les chemins admissibles sont générés par *concaténation* des primitives de mouvement
- Par exemple, pour un *robot unicycle*, l'ensemble de vitesses suivant :

$$v(t) = \overline{v}, \quad \omega(t) \in \{-\overline{\omega}, 0, \overline{\omega}\}, \quad t \in [t_s, t_s + \Delta], \quad \overline{v}, \overline{\omega} > 0$$

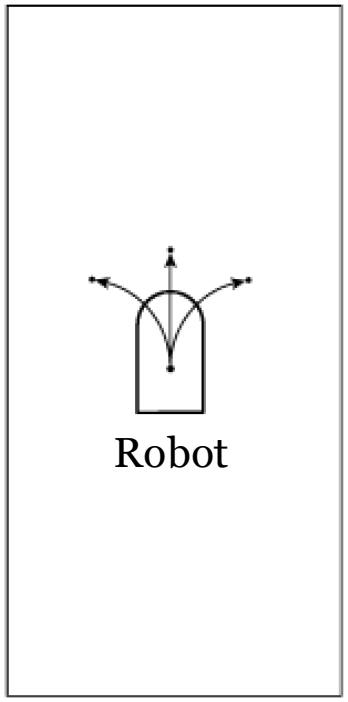


produit trois chemins locaux admissibles : 1) virage à gauche le long d'un arc de cercle, 2) trajet en ligne droit, 3) virage à droite le long d'un arc de cercle

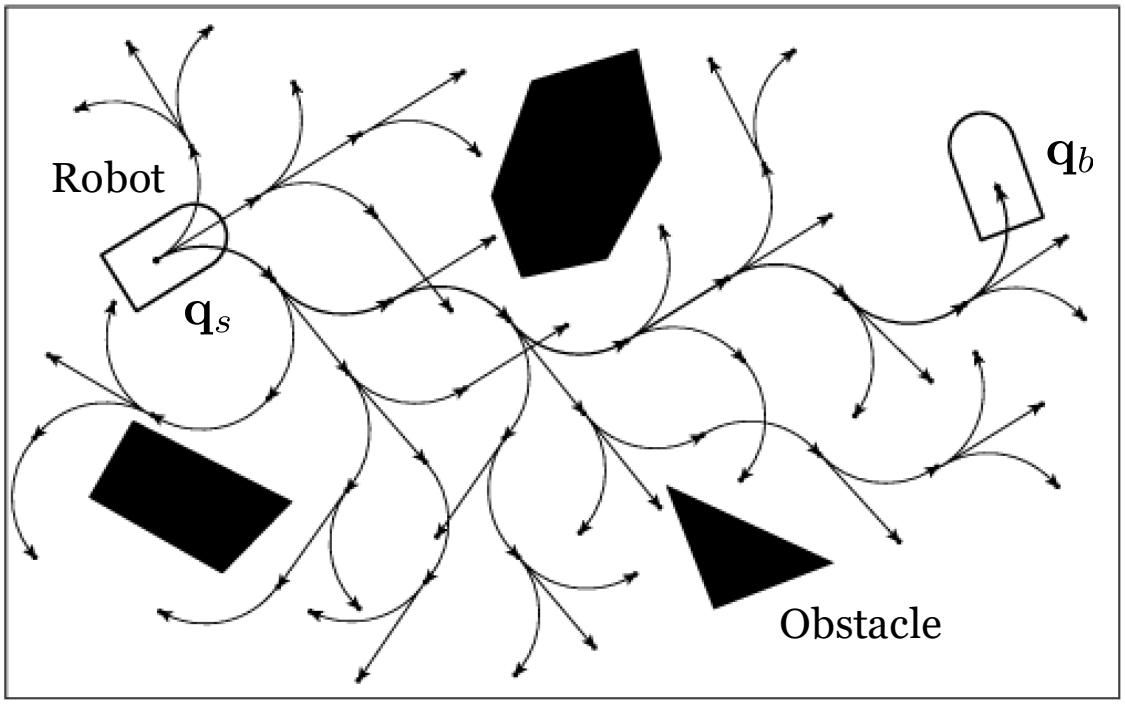
Remarque : ces primitives de mouvement n'autorisent pas la *marche arrière* ou une *rotation sur place* (en effet, $\overline{v} > 0$)

Planification probabiliste : RRT

Extension pour robots non-holonômes



Les 3 primitives de mouvement pour un robot unicycle



Un exemple de RRT (sa projection dans l'espace opérationnel du robot)

Planification probabiliste : RRT

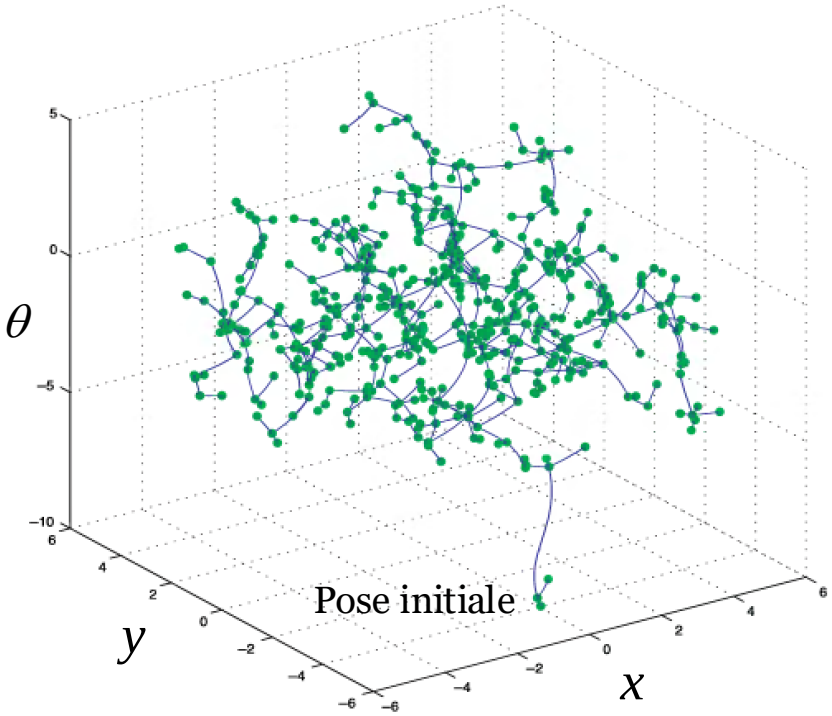
Extension pour robots non-holonômes

- L'expansion d'un RRT pour un robot non-holonôme est similaire à la procédure vue précédemment (seulement la génération de $\mathbf{q}_{\text{new}}$ est différente)

- Sous des hypothèses appropriées, on peut montrer que si la configuration finale peut être atteinte de la configuration initiale à travers une concaténation sans collisions de primitives, la probabilité que $\mathbf{q}_b$ soit rajouté à l'arbre T tend vers 1 comme le temps d'exécution tend vers l'infini

- Pour augmenter l'efficacité de la méthode, une *version bidirectionnelle* peut être envisagée

Espace des poses du robot ($\mathcal{C} = \mathrm{SO}(2)$)



RRT pour un robot unicycle avec configuration initiale $[0,\ 0,\ 0]^T$. Chaque sommet (vert) représente une pose admissible du robot dans l'espace libre

Robotics Toolbox de P. Corke :

```
>> rrt.plan()
```

Planificateurs probabilistes : logiciel et variantes

- Implémentation efficace des méthodes basées sur l'échantillonnage (C++/Python): The Open Motion Planning Library (OMPL), https://ompl.kavrakilab.org
- Nombreuses variantes et améliorations de PRM et RRT dans la littérature :

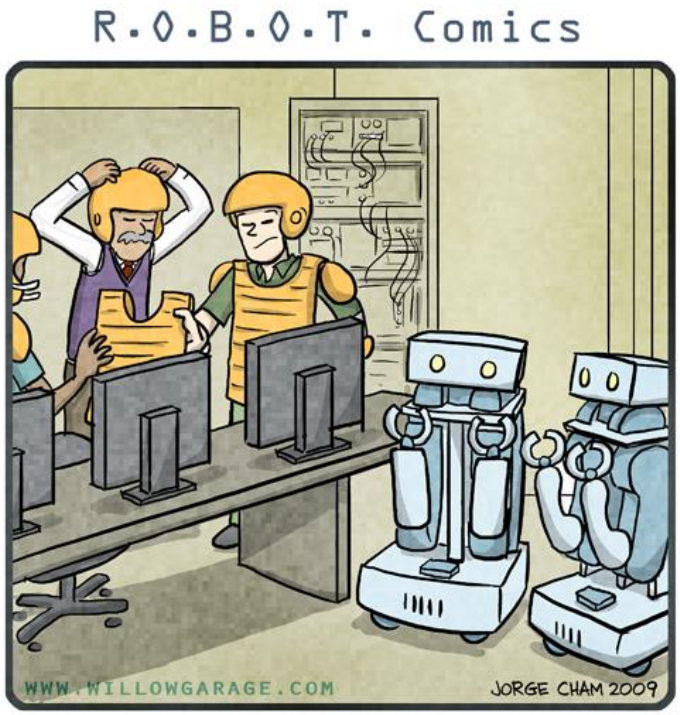
Planificateurs *multiple-query* :

- PRM
  - Lazy PRM
  - PRM* : version asymptot. optimale de PRM
  - Lazy PRM*
- SPARS (*SPArse Roadmap Spanner algorithm*) : planificateur asymptot. quasi-optimal
- SPARS2

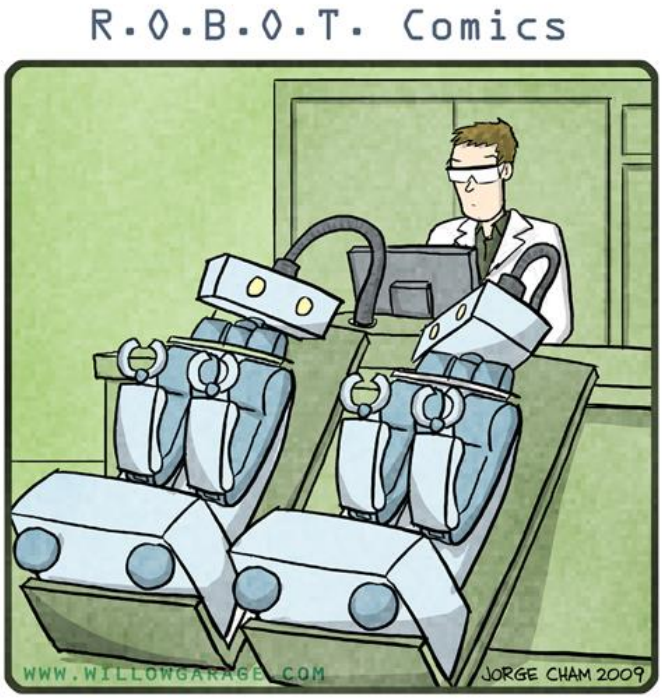
Planificateurs *single-query* :

- RRT
  - Lazy RRT
  - RRT*: version asymptot. optimale de RRT
  - RRT# et RRTX: deux variantes de RRT* avec une vitesse de convergence plus rapide
  - Informed RRT* et neural RRT*
  - SST (*Sparse Stable RRT*): planificateur ciné-dynamique asymptot. quasi-optimal
  - VF-RRT (*Vector Field RRT*)
  - pRRT (*Parallel RRT*)
  - TSRRT (*Task-Space RRT*)
- EST (*Expansive Space Trees*)
- KPIECE (*Kinematic Planning by Interior-Exterior Cell Exploration*)
- STRIDE (*Seach Tree with Resolution Independent Density Estimation*)
- PDST (*Path-Directed Subdivision Trees*)
- FMT* (*Fast Marching Tree*) : plus rapide que RRT*
- QRRT (*Quotient-Space RRT*)

Fin du CM



"I HAVE A BAD FEELING ABOUT THIS DEMO."



"DO YOU EVER FEEL LIKE YOU'RE IN THE MATRIX?"