# Commande et estimation pour la robotique mobile

SYS5240

ESIEA 5A

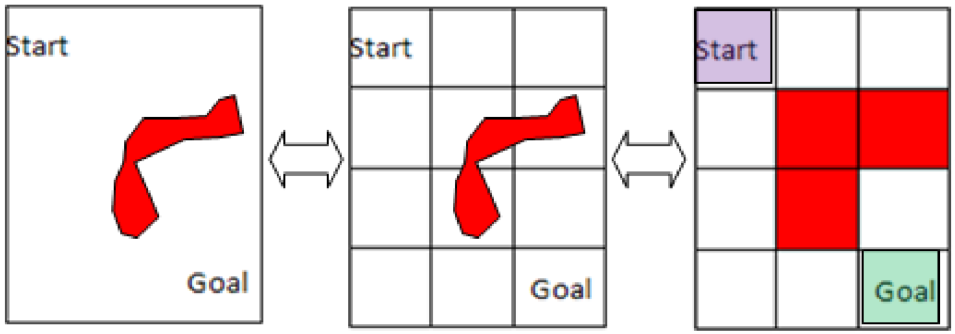
S. Bertrand
sbertrand@esiea.fr

# Génération de trajectoires

- Représentation de l'environnement
  - Carte, grille d'occupation et graphe
  - Matrice d'adjacence
- Recherche de chemin
  - Algorithme de Dijkstra
  - Algorithme A*
  - Autres variantes et propriétés
- Génération d'une trajectoire continue
  - Courbes de Bézier

# Représentation de l'environnement

- Modélisation de l'environnement

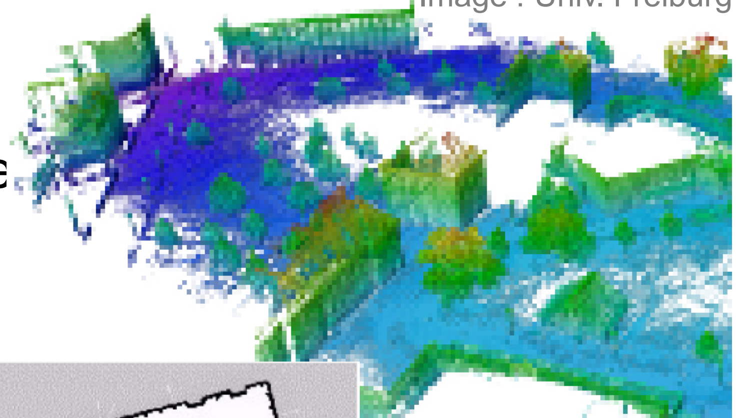



grille d'occupation

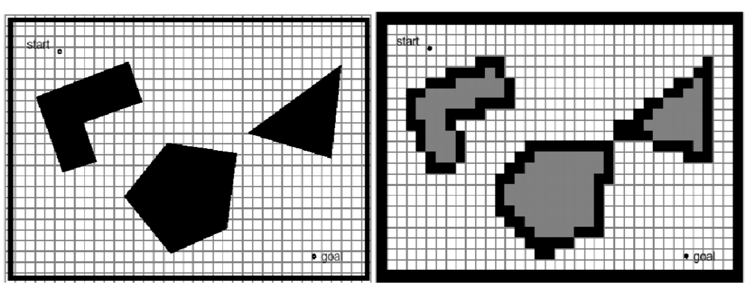
discrétisation

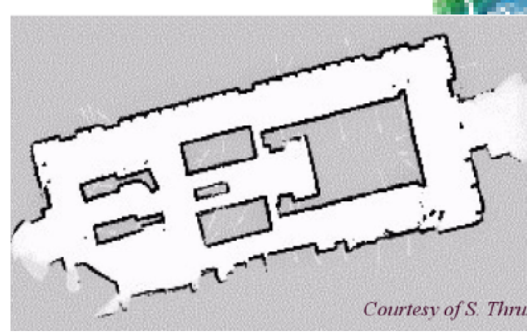
- Grille d'occupation :
  - Binaire : 0 -> libre, 1 -> occupée
  - Probabilité d'occupation
- Exemples en 2D ou 3D :

Image : Univ. Freiburg



octomap

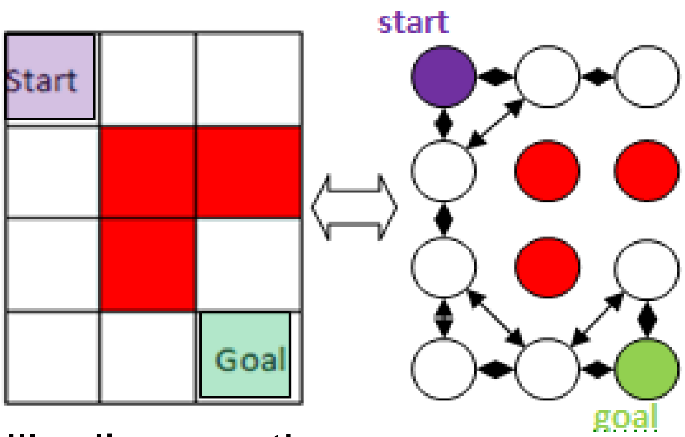



Source : R. Siegwart, M. Chli, M. Rufli & D. Scaramuzza, Autonomous Mobile Robots, Localization I, ETH

# Représentation de l'environnement

- Représentation par un graphe
  - Sommets : positions
  - Arcs : relation d'adjacence
  - Poids des arcs : adjacence ou distance

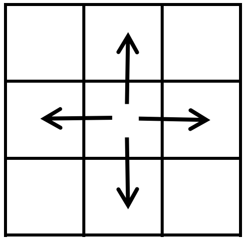



grille d'occupation graphe

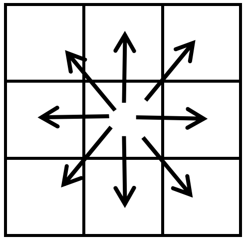
- Plusieurs possibilités de définir l'adjacence

(degré maximal d'adjacence des sommets)

4-adjacency



8-adjacency



etc…

=> Définit les mouvements possibles dans la recherche du chemin

# Représentation de l'environnement

- Matrice d'adjacence

$$A = [a_{ij}]$$

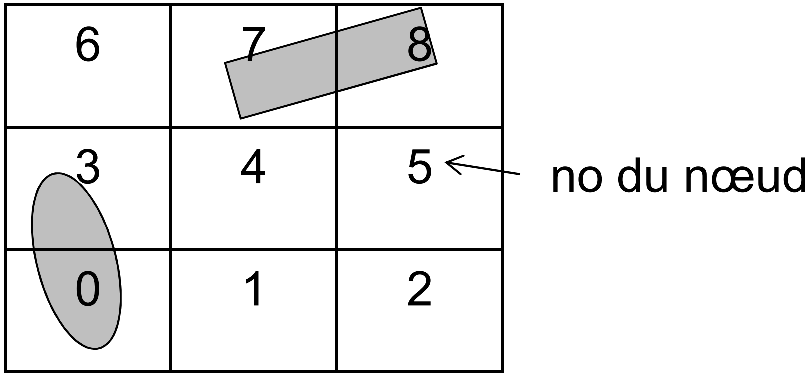
$a_{ij}$ = 1 si adjacence entre les nœuds *i* et *j* (*j*≠*i*)
= 0 sinon

$a_{ii}$ = 0

  - De taille $n$ x $n$, où $n$ est le nombre de nœuds
  - Graphe non orienté => matrice symétrique

$$\sum_i a_{ij} = \text{nombre de nœuds adjacents au nœud } i$$

- Exercice : donner la matrice d'adjacence de la carte suivante dans les cas *4* et *8-adjacency*



# Génération de trajectoires

- Représentation de l’environnement
  - Carte, grille d’occupation et graphe
  - Matrice d’adjacence
- Recherche de chemin
  - Algorithme de Dijkstra
  - Algorithme A*
  - Autres variantes et propriétés
- Génération d’une trajectoire continue
  - Courbes de Bézier

# Recherche de chemin

- Algorithme de Dijkstra : principe
  - Recherche de chemin entre nœud *Start* et nœud *Goal*
  - Algorithme itératif
  - Génère l'arbre couvrant de taille minimum du graphe, à partir du nœud *Start*, jusqu'au nœud *Goal*
  - L'exploration se fait en passant par les nœuds de distance croissante à la racine
  - Avantage : le chemin trouvé est de longueur optimale
  - Inconvénient : nécessite potentiellement un grand nombre d'itérations

E.W.Dijkstra, *A Note on Two Problems in Connexion with Graphs, Numer*ische Mathematik 1, pp 267-271, 1959

# Recherche de chemin

- $g(s)$ : coût accumulé pour arriver au nœud $s$ en partant de $n_{start}$
- $c(n_c,s)$ : coût pour aller du nœud $n_c$ au nœud $s$
  - Exemples :
    - Distance euclidienne

$$c(n_c, s) = \sqrt{(x_{n_c} - x_s)^2 + (y_{n_c} - y_s)^2}$$

    - Distance de Manhattan

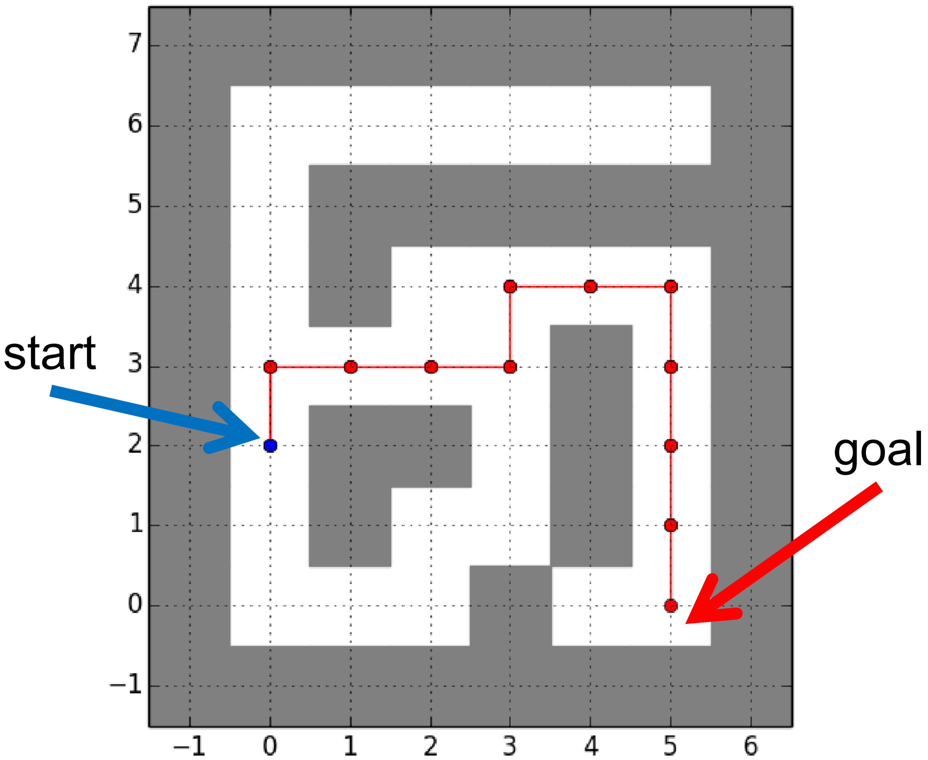
$$c(n_c, s) = |x_{n_c} - x_s| + |y_{n_c} - y_s|$$

# Recherche de chemin

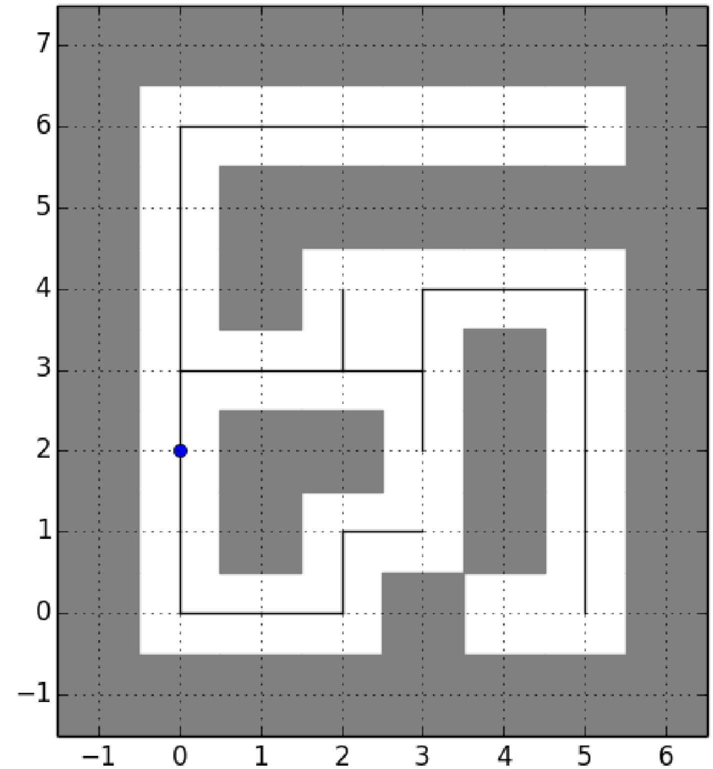
- Algorithme de Dijkstra : pseudo-code
  - Entrée : nœuds $n_{start}$ et $n_{goal}$
  - Initialisation de deux listes : *Open*:= $\{n_{start}\}$, *Closed*:={ }
  - Pour chq nœud $n$ : $g(n) := +\infty$, $g(n_{start}) := 0$
  - Tant que *Open* non vide :
    - Enlever *Open* le nœud de cout $g$ minimal -> noeud $n_c$
    - Ajouter $n_c$ à *Closed*
    - Si $n_c = n_{goal}$ : fin de l'algorithme : succès
    - Pour chaque successeur $s$ de $n_c$ :
      - Si $g(s)>g(n_c)+c(n_c, s)$
        - Indiquer $n_c$ comme le parent de $s$
        - $g(s):=g(n_c)+c(n_c,s)$ et ajouter s à Open
  - Fin de l'algorithme : échec
  - Sortie : liste *Closed* permettant de trouver le chemin

# Recherche de chemin

- Algorithme de Dijkstra : exemple
  - 4-adjacency, distance de Manhattan




Chemin optimal




Arbre couvrant : 27 nœuds évalués

# Recherche de chemin

- Algorithme A* : principe
  - Diriger la recherche vers l'objectif à atteindre, à l'aide d'un coût $h(n_c, n_{goal})$ (« heuristique »)
  - Si la fonction $h$ est admissible, cad qu'elle ne surestime pas le coût minimum du chemin entre $n_c$ et $n_{goal}$, alors l'algorithme A* garantit l'obtention d'un chemin optimal
  - Coût d'un chemin testé passant par le nœud $n_c$ :

$$f(n_c) = \underbrace{g(n_c)}_{\text{Coût pour arriver à } n_c} + \underbrace{h(n_c, n_{goal})}_{\text{Coût pour aller de } n_c \text{ de à } n_{goal}}$$

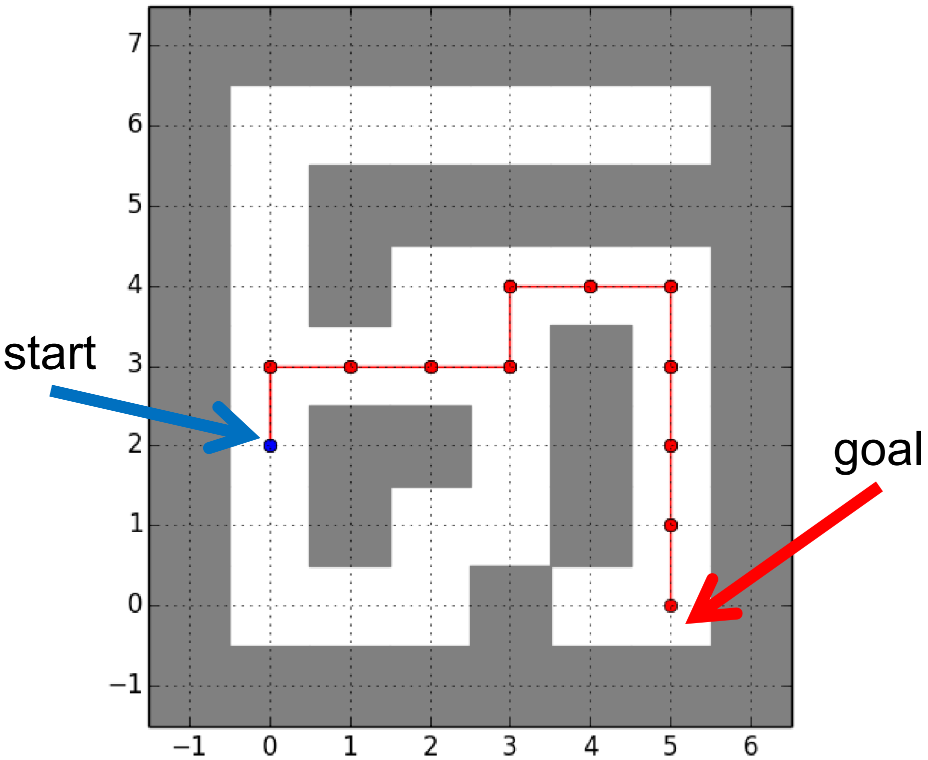
  - Exemple de fonctions $h$ : distance Euclidienne, distance de Manhattan

# Recherche de chemin

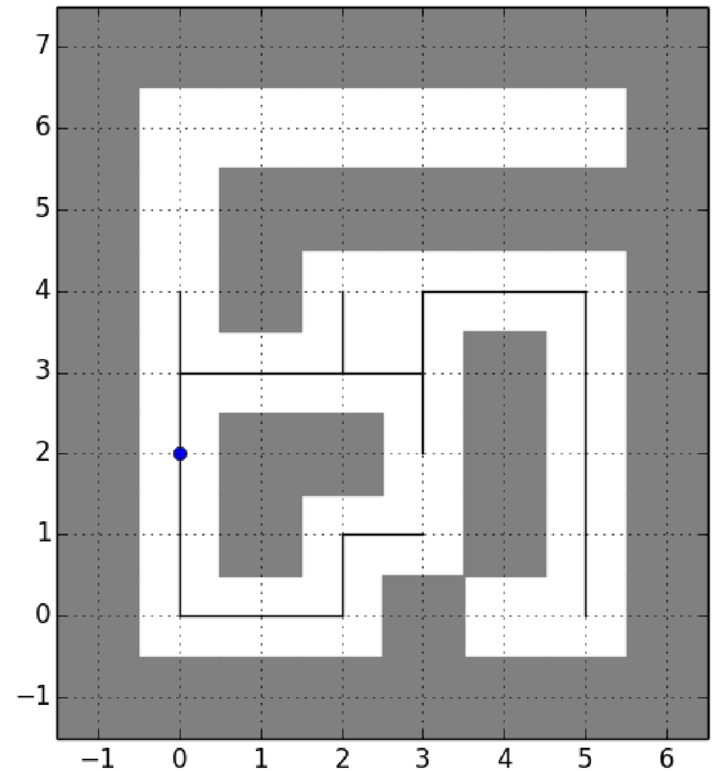
- Algorithme A*: pseudo-code
  - Entrée : nœuds $n_{start}$ et $n_{goal}$
  - Initialisation de deux listes : *Open*:= {$n_{start}$}, *Closed*:={ }
  - Pour chq nœud $n$ : $g(n) := +\infty$, $g(n_{start}) := 0$, $f(n_{start}) := g(n_{start}) + h(n_{start}, n_{goal})$
  - Tant que *Open* non vide :
    - Enlever *Open* le nœud de cout $f$ minimal -> noeud $n_c$
    - Ajouter $n_c$ à *Closed*
    - Si $n_c = n_{goal}$ : fin de l'algorithme : succès
    - Pour chaque successeur $s$ de $n_c$ :
      - Si $g(s) > g(n_c) + c(n_c, s)$
        - Indiquer $n_c$ comme le parent de $s$
        - $g(s) := g(n_c) + c(n_c, s)$ , $f(s) := g(s) + h(s, n_{goal})$
        - ajouter s à Open (ou le mettre à jour si déjà présent)
  - Fin de l'algorithme : échec
  - Sortie : liste Closed permettant de trouver le chemin

# Recherche de chemin

- Algorithme A*: exemple
  - 4-adjacency, distance de Manhattan




Chemin optimal

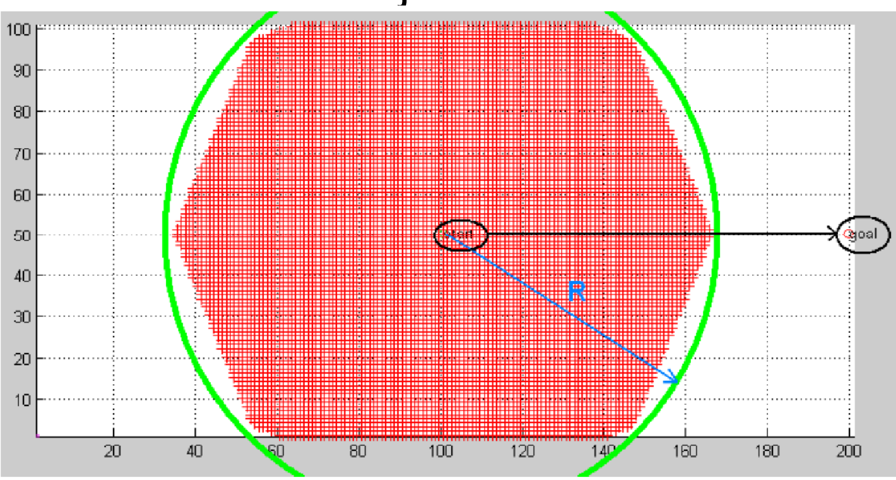



Arbre couvrant : 20 nœuds évalués

# Recherche de chemin

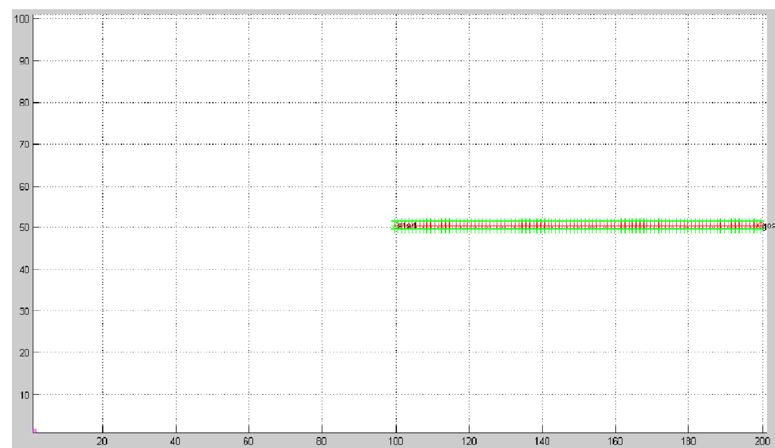
- Un exemple (dégénéré) de grande dimension pour bien comprendre
  - Carte de taille 200 x 100, sans obstacle
  - $n_{start}$ : ($x$=100,$y$=50) $n_{goal}$ : ($x$=200,$y$=50)

Dijkstra




11000 itérations, et $n_{goal}$ n'est toujours pas atteint !

A*

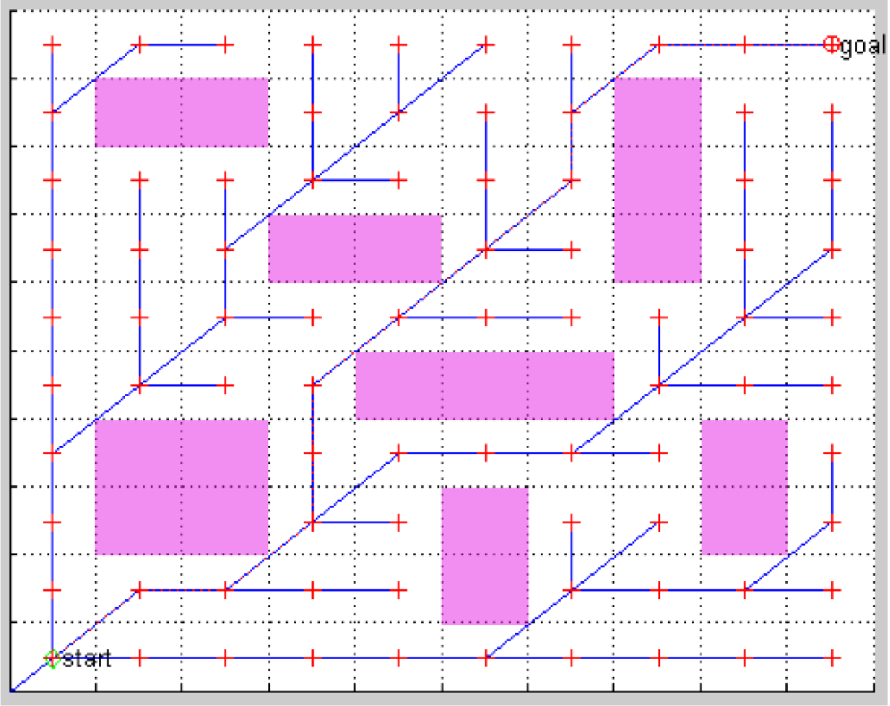



100 itérations, et $n_{goal}$ est atteint !

# Recherche de chemin

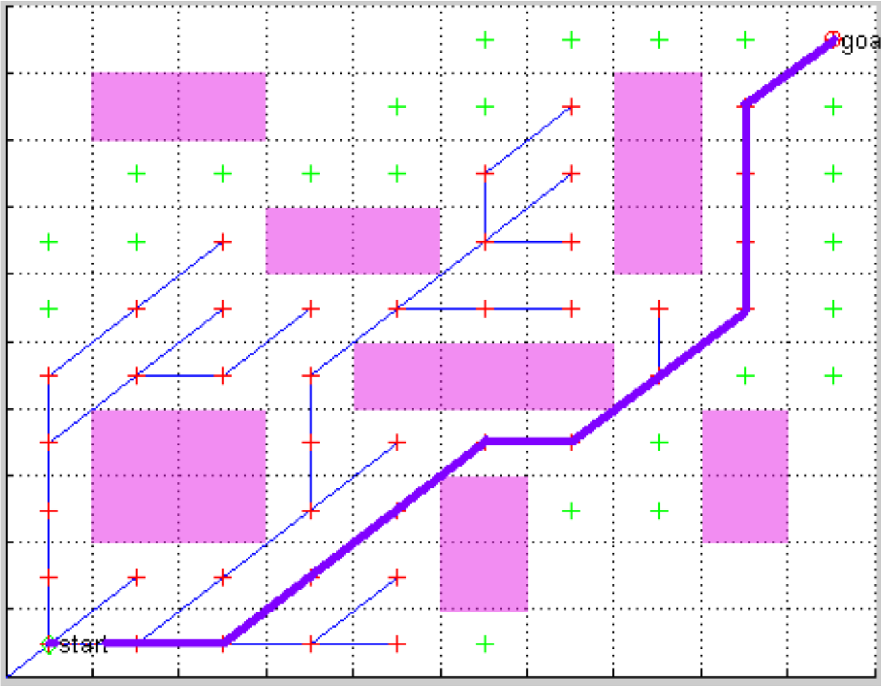
- Autre exemple : 8-adjacency, distance euclidienne

Dijkstra




82 itérations

A*




41 itérations

# Recherche de chemin

- Nombreuses variantes de ces algorithmes
  - ARA*, D*, D* Lite, etc..

- Propriétés recherchées :
  - Aspect « Dynamic » :
    - prise en compte de modifications en ligne de la carte (obstacles, mises à jour)
    - en réutilisant les informations de planification réalisées précédemment

  - Aspect « Anytime » :
    - possibilité d'interrompre le calcul n'importe quand en obtenant tout de même une solution faisable

# Génération de trajectoires

- Représentation de l'environnement
  - Carte, grille d'occupation et graphe
  - Matrice d'adjacence
- Recherche de chemin
  - Algorithme de Dijkstra
  - Algorithme A*
  - Autres variantes et propriétés
- Génération d'une trajectoire continue
  - Courbes de Bézier

# Génération d'une trajectoire continue

- Algorithmes de recherche de chemin : donne une liste de points de passage (way points)

- Selon la commande réalisée (cf. suite du cours) on peut avoir besoin d'un profil de position, vitesse (et accélération) continu au cours du temps

$p(t)$ de classe $C^1$ => $p(t)$ continue, $v(t)$ continue
$p(t)$ de classe $C^2$ => $p(t)$ continue, $v(t)$ continue, $a(t)$ continue

# Génération d'une trajectoire de référence

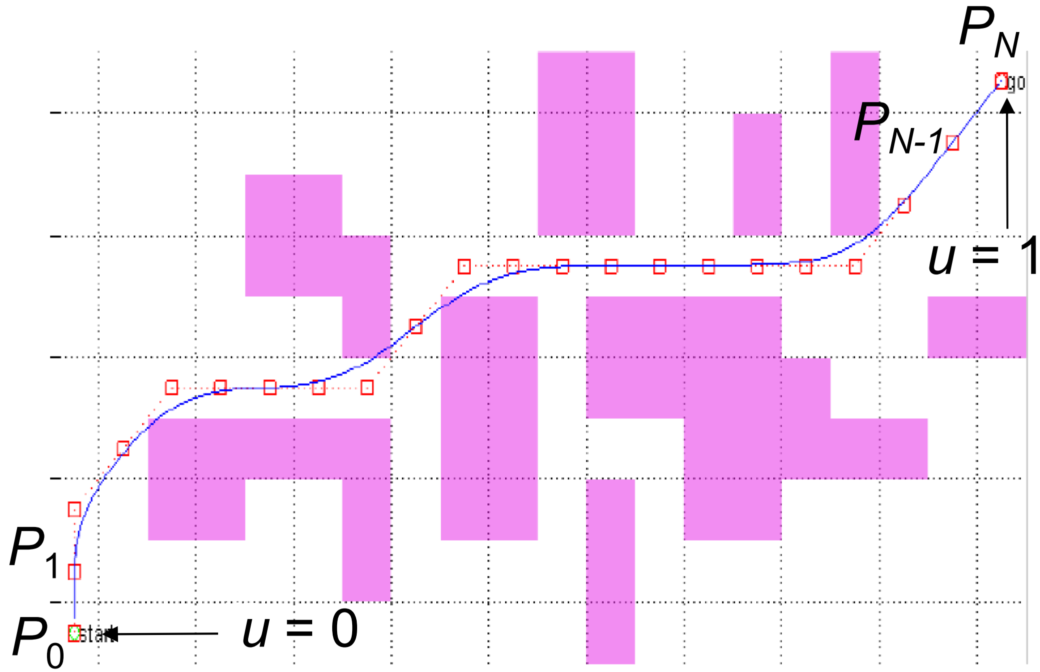
## •Courbes de Bézier

- •Définie à partir d'un ensemble de $N$ « points de contrôle » $P_i$
- •Courbe polynomiale de classe $C^N$ dépendant d'un paramètre
- •Passe par le 1er et le dernier point de contrôle
- •Est tangente à $P_0P_1$ et $P_{N-1}P_N$

$$p(u) = \sum_{i=0}^{N} B_i^N(u).P_i$$

$$\text{pour} \quad u \in [0,1]$$

$$B_i^N(u) = \frac{N!}{i!(N-i)!} u^i (1-u)^{N-i}$$



# Conclusion

On sait donc:

- modéliser l'environnement du robot
    - carte, grille d'occupation, graphe
- chercher dans cette carte un chemin pour le robot
- générer une trajectoire sous forme de points de passage ou d'une référence suffisamment « lisse »

On voudrait maintenant :

- que le robot se localise
- que le robot suive la trajectoire souhaitée