ROB316
TP3: Planification de chemin pour la méthode A^*

Jahmal BAPTISTE
Bunthet SAY

Janvier 2020

Contents

1 Introduction et résumé du cours

La planification de trajectoire est la recherche d'un chemin entre la position courante et un but en minimisant une sorte de coût. Elle est composé de 3 étapes principaux:

- modélisation de la carte:
  Dans la pratique, l'environnement est modélisé soit une carte topologique soit une carte métrique. Le choix est fait selon l'application.
- Discrétisation des espaces de recherche:
  On représente l'environnement sous forme d'un graphe discret et le problème devient donc la recherche de chemin dans un graphe.
- Recherche de chemin dans un graphe
  Il existe plusieurs algorithmes pour estimer le coût de chemin le plus court entre chaque noeud et le but et elles sont classées dans 2 types: algorithme avec et sans information.A^*, une extension de $Dijktra$, est une algorithme informé par l'ajout d'une heuristique de la distance d'un point vers le but. Cela permet à l'algorithme de chercher les noeuds qui semble le plus intéressante et donc pourrait réduire le temps de la recherche.

Quand le robot a une configuration, on l'en prend en compte et on faire la planification dans l'espace de configuration.

Dans ce TP, nous utilisons la méthode A^* pour la planification de trajectoire en présentant des obstacles.On va analyser l'effet de l'heuristique et le rôle de poids associés au noeuds.

Notre travail se trouve dans ce répositoire GitHub.

2 Fonction fournie

2.1 Question 1

L'algorithme de A^* est une extension de l'algorithme $Dijktra$ adaptée pour la planification trajectoire. La $Dijktra$ fait la recherche dans une ensemble des noeuds pas encore regardés tous les noeud dans un rayon circulaire fixe en augmentant graduellement ce cercle pour rechercher des intersections de plus en plus loin de notre point de départ jusqu'à arriver au but ou jusqu'à tous les noeuds sont visités. D'une itération à la prochaine, cette algorithme cherche un noeud pas encore visité qui minimise la fonction $f(n) = g(n)$ tel que g(n) est la distance entre le noeud de départ et noeud n.

La A^* ajoute une information (heuristique) $h(n)$ à tous les noeuds permettant de faire la recherche dans la direction le plus intéressante d'abord. Dans le code, $h(n)$ est la distance euclidienne entre le noeud n et le but et la fonction

f à minimiser est $f(n) = g(n) + h(n)$. Donc, la A^* cherche les noeuds dans la direction vers le but d'abord. Cela permet la recherche de s'arrêter plus vide.

3 Rôle de l'heuristique

3.1 Question 2

Avec le paramètre $weight = 0$, l'algorithme exécuté est l'algorithme de $Dijktra$ car $H = 0$ (il n'y a pas d'heuristique).

3.2 Question 3

Lorsque l'on modifie le paramètre `weight` l'ordre de recherche des points est modifié; plus le paramètre est élevé plus l'algorithme privilégiera des points proches de l'objectif (selon son heuristique) - il donnera alors beaucoup d'importance à sa fonction heuristique et fera moins d'exploration.

- Pour la valeur 1 (cf. FIGURE 2)
  L'algorithme commence à prendre en compte la direction de l'objectif dans sa recherche. Il ignore les points qui sont trop dans la direction opposée et n'explore pas spécialement latéralement.
- Pour la valeur 3 (cf. FIGURE 3)
  L'algorithme va en ligne droite jusqu'à ce qu'il heurte un obstacle. Une fois un obstacle heurté, il se met à chercher presque uniquement latéralement, jusqu'à ce qu'il puisse repartir dans la direction que lui suggère son heuristique.
- Pour la valeur 1.4 (cf. FIGURE 4)
  La recherche sans obstacle est toujours aussi franche dans la direction suggérée par l'heuristique mais l'exploration après avoir rencontré un obstacle est plus ouverte (il recherche plus en arrière qu'avec la valeur 3).

La valeur qui permet une obtention de résultat la plus rapide dans ce cas est la 3 car l'algorithme trouve l'objectif avec le moins de points de recherche. Cependant la valeur 1.4 permet de trouver une trajectoire plus souple en aval.

La valeur la plus adaptée dépend donc de la question de la qualité de trajectoire que l'on veut. Si l'on veut une trajectoire plus souple la valeur la plus adaptée pour le paramètre `weight` est 1.4.

4 Rôle de poids associé au noeuds

4.1 Question 4

En fixant le paramètre `weight` à une valeur de 1.4 on prend ici la distance aux obstacles en compte lors du choix de trajectoire (une fois l'objectif trouvé).

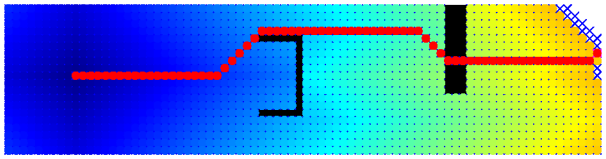

Figure 1: `weight = 0`

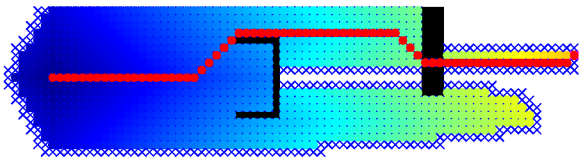

Figure 2: `weight = 1`

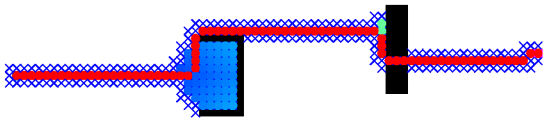

Figure 3: `weight = 3`

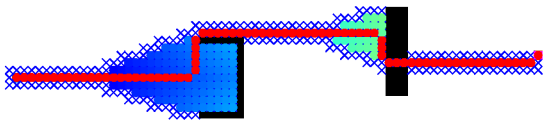

Figure 4: `weight = 1.4`

La fonction `bwdist` de *MATLAB* permet de calculer la distance minimale de chaque point à l'obstacle le plus proche et c'est grâce à cette fonction que nous avons pu obtenir des trajectoires qui s'éloignent des obstacles.

Nous avons spécifié le type de distance prise en compte à `'euclidean'` (et `'quasi-euclidean'` pour comparer). Il ne faut néanmoins pas utiliser ces valeurs telles quelles en tant que poids car cela signifierait qu'il faut attribuer plus d'importance aux points le plus proches des obstacles. Il a donc fallu inverser les valeurs pour remédier à cette subtilité dans l'interprétation de ces valeurs.

Les trajectoires obtenues sont présentées en FIGURES 5 et 6.
Nous y voyons que la trajectoire s'éloigne effectivement des obstacles (et qu'il n'y a pas de différence en pratique entre les deux méthodes de calcul de la distance).

5 Influence de l'environnement

5.1 Question 5

En ajoutant des obstacles nous avons moyenné les résultats sur 5 essais pour chaque jeu de valeurs, vu que les obstacles sont générés aléatoirement.

Nous avons testé l'algorithme avec 2, 5, 10, 20 et 50 obstacles pour comparer leur vitesse d'exécution.

Nous avons observé que l'algorithme trouvait l'objectif plus rapidement en moyenne avec le paramètre `weight` à 1.4 qu'à 0 quel que soit le nombre d'obstacles.
Les FIGURES 7 et 8 montrent dans quel cas de figure l'algorithme A* apporte le plus de gain : lorsque l'environnement présente une occasion de rebrousser chemin, l'algorithme A* aura tendance à ne pas le faire, contrairement à l'algorithme *Dijkstra*.

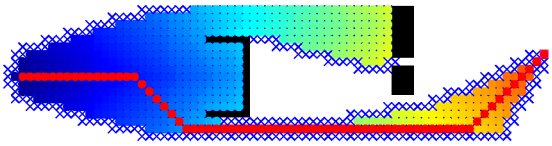

Figure 5: `W = euclidean`

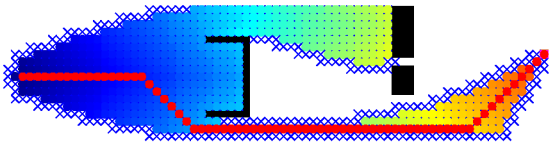

Figure 6: `W = quasi-euclidean`

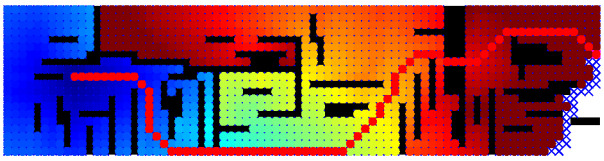

Figure 7: `weight = 0` et obstacles = 50

Figure 8: `weight = 1.4` et obstacles = 50

6 Conclusion

Nous avons pu comparer l'algorithme A* à l'algorithme *Dijkstra* dans le cas de la planification de trajectoire.

Nous avons pu voir que la connaissance du milieu via une fonction heuristique permettait d'accélérer le calcul de la trajectoire mais qu'il faut aussi calibrer la fonction heuristique pour en tirer d'autres profits (qualité de la trajectoire).

Ces calculs s'appliquent dans le cas d'une connaissance qui dépasse les capteurs qui équipent le robot (lorsqu'il est dans l'environnement pour le moins) donc ces calculs ne peuvent se faire qu'avec l'aide d'une tierce entité (satellite par exemple) ou en amont.