Projet algorithme des graphes

Le robot ramasseur !

---

fevrier 2023-mai 2023

FIGURE 1 – Tennibot – Le robot ramasseur de balles de tennis

Projet réalisé par : JERIBI Ramy, Mahjoub Youssef, LARRAGETA César, OZANE Corentin.

Introduction

Ce projet vise à concevoir et programmer des algorithmes de ramassage. Pour cela nous allons considérer un robot qui est chargé de ramasser K déchets répartis dans un monde de manière arbitraire.

Définition des objets

Monde

L'espace de recherche du robot est modélisé par une grille carrée de coté N sur lequel le robot peut se déplacer en ligne droite (dans le cas d'un monde sans obstacles),les potentielles positions du robot sont donc comprises dans l'intervalle [1,N]x[1,N].

Déchets et Obstacles

Les positions des déchets et des obstacles sont encodées dans le fichier **test_world.txt** en utilisant le format suivant : une lettre (robot (R), déchet (un entier correspondant à ses coordonnées), un obstacle X), suivie des coordonnées correspondantes aux positions des objets (du coin inférieur droit et supérieur gauche pour les obstacles).

Algorithme

Le but de ce projet est de trouver un chemin dont la durée de parcours est la plus courte. Ce qui nous amène à considérer la vitesse de rotation du robot et la vitesse de translation qui sont tous les deux constantes. En effet le chemin varie en fonction de la prédominance de l'une de ces vitesses sur l'autre. En effet plus la vitesse de rotation est faible plus le robot sera poussé à limiter les détours et privilégier les lignes droites. Pour la suite nous avons choisis de considérer une vitesse de rotation infinie (instantané). Pour ce faire nous avons créer une classe *world* qui initialise la taille du monde, le nombre de déchets à ramasser, l'angle initiale du robot et le nombre de déchet à ramasser. Les attributs de cette classe sont :

1. **load_from_file** qui permet d'initialiser le monde à partir d'un fichier donnée
2. **world_graph** qui permet d'afficher le monde et les obstacles et il est de complexité O(max(nombre de déchets,nombre d'obstacle) avec n la taille du monde
3. **calcul_distances** Calcule la distance entre tous les points et il est de complexité O(n^2)
4. **tsp** Calcule le chemin optimale de parcours en parcourant tous les chemins possibles. La complexité de cette algorithme est de O($n!$).

   Le point positif de cette algorithme c'est qu'il est fonctionnel avec n'importe quelle monde (graphe) et qu'il renvoie souvent un schémas correct. Les points négatifs sont sa complexité temporelle qui est de O($n!$) et sa complexité spatiale car il stocke toute les valeurs d'angle et de distances disponibles.

5. un algorithme glouton a aussi été réalisé à l'aide de l'attribut **tsp_glouton**

Applications

Nous avons considéré le monde2 suivant qui est affiché à l'aide de l'attribut World_graph(). Le

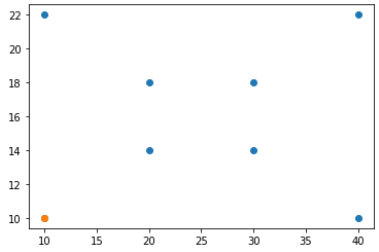

FIGURE 2 – Monde

point initial est marqué par la couleur orange. Vu que la vitesse de rotation est instantanée le chemin le plus court en termes de temps correspond à la distance la plus courte, donc en numérotant les sommets de en commençant de haut à gauche vers en bas à droite de 1 à 8. Intuitivement, on aura le parcours suivant (7,5,6,8,2,3,4,1). L'algorithme implémenté donne le parcours suivant4 :

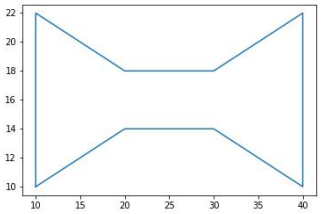

FIGURE 3 – parcours tsp

ce qui correspond à l'intuition initiale.

Le deuxième exemple est le suivant :

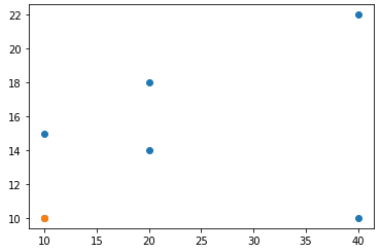

FIGURE 4 – Monde 2

En numérotant la figure comme précédemment on aura deux chemins (5,4,6,1,2,3) et (5,3,2,1,6,4). le résultat obtenue est correct car nous obtenant la figure suivante 5 :

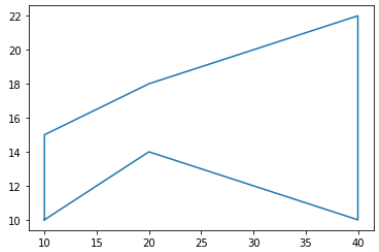

FIGURE 5 – parcours tsp deuxième exemple

Algorithme avec obstacle

L'objectif de cette partie du projet est de trouver la méthode la plus rentable dans un monde plein d'obstacles. Cela nous oblige non seulement à considérer tout ce qui a été dit jusqu'à présent, mais aussi à introduire de nouveaux outils pour créer de nouveaux algorithmes pour surmonter les obstacles.

Pour cela, nous utilisons la nouvelle fonction *path_with_obstacle*. Cela vous permet de calculer le chemin le plus court entre deux emplacements tout en évitant les obstacles. Les entrées sont l'obstacle à éviter, la position de départ *src*, la position de fin *dst*, l'angle actuel *current_angle*, la vitesse de déplacement et la vitesse de rotation. Elle renvoie le chemin le plus court, la distance parcourue et l'angle final entre *src* et *dst* tout en évitant les obstacles. Dans le même temps, une nouvelle fonction *tsp_obstacles* a été ajoutée qui prend en compte les obstacles et résout le problème du robot. Elle prend la vitesse de déplacement et la vitesse de rotation en entrée et renvoie le chemin le plus court pouvant atteindre toutes les positions spécifiées tout en évitant les obstacles.

La complexité temporelle de *tsp_obstacles* est généralement très élevée. Puisque nous devons tester toutes les permutations possibles de positions, la complexité temporelle est de l'ordre de $O(n!)$ (où n est un nombre). Visitez un poste. De plus, la fonction *path_with_obstacle* est appelée pour chaque paire d'emplacements connectés par des segments qui traversent des obstacles, de sorte que sa complexité temporelle est multipliée par le nombre de ces paires. Donc en pratique cette fonction est très chronophage lorsqu'il y a beaucoup de positions et d'obstacles.

Applications

Afin d'appliquer cet algortihme en conditions réelles, lui ont été données en entrée le monde avec obstacles de la **Figure 2**, mais un obstacle X a été rajouté en position : $X : (24, 8); (26, 24)$. Le monde de test est donc celui de la **Figure 6**.

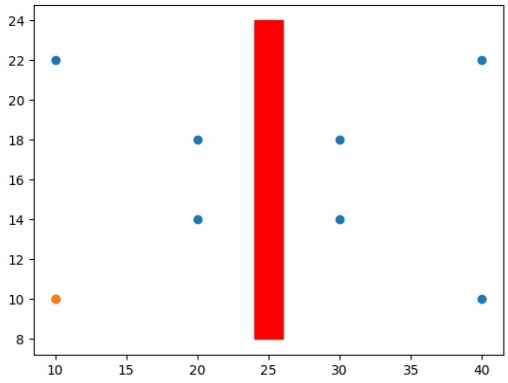

FIGURE 6 – Monde avec obstacles

On fixe les vitesses de déplacement et de rotation du robot à 1, le point de départ du robot étant toujours marqué de la couleur orange, il retourne le chemin visible en **Figure7**, correspondant en effet au chemin intuitivement le plus court. l'algorithme naif utilisé garanti effectivement de retourner le chemin le plus court possible, mais sa correction se fait au détriment de sa complexité puisque toutes les possibilités sont examinées avant de ne sélectionner que la plus optimale, ce qui nous empêchera de traiter des jeux de données trop importants.

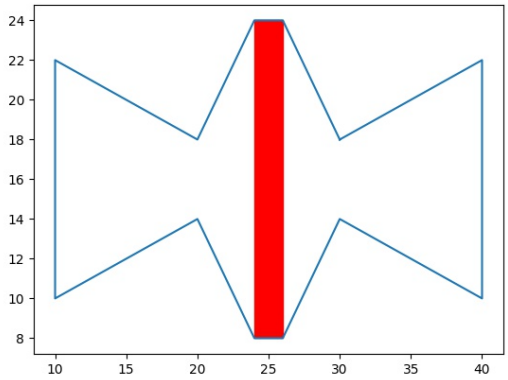

FIGURE 7 – Chemin avec obstacles

Conclusion

En conclusion, les algorithmes mis au point dans ce projet sont efficaces pour résoudre le problème du robot ramasseur de déchets dans un monde avec et sans obstacles, mais seulement dans le cas d'un faible nombre de déchets. En effet, il est important de noter que les algorithmes proposés ont une complexité temporelle élevée, ce qui les rend inefficaces pour traiter de grands jeux de données. Des algorithmes plus performants tels que Dijkstra ou A* pourraient être envisagés pour améliorer l'efficacité et la scalabilité de la solution. Ils offriraient une meilleure exploration de l'espace de recherche, réduisant ainsi le temps de calcul et permettant de résoudre des problèmes de plus grande envergure.