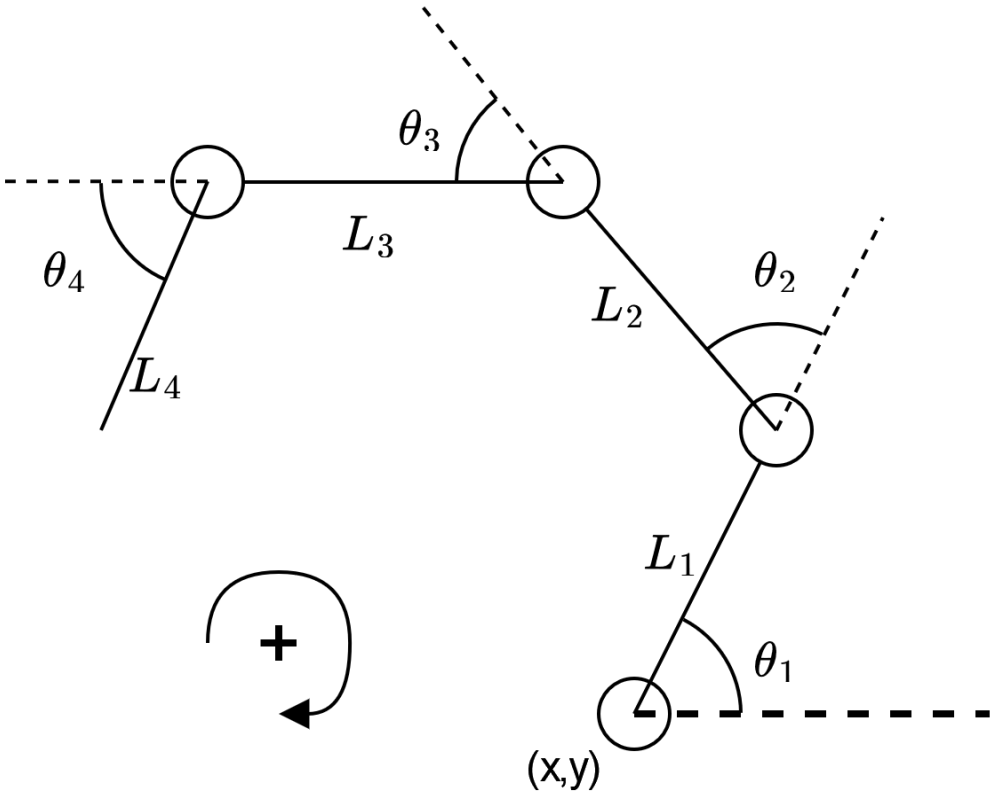
Compte rendu de PLANI

Sommaire

I. Objectif du TP

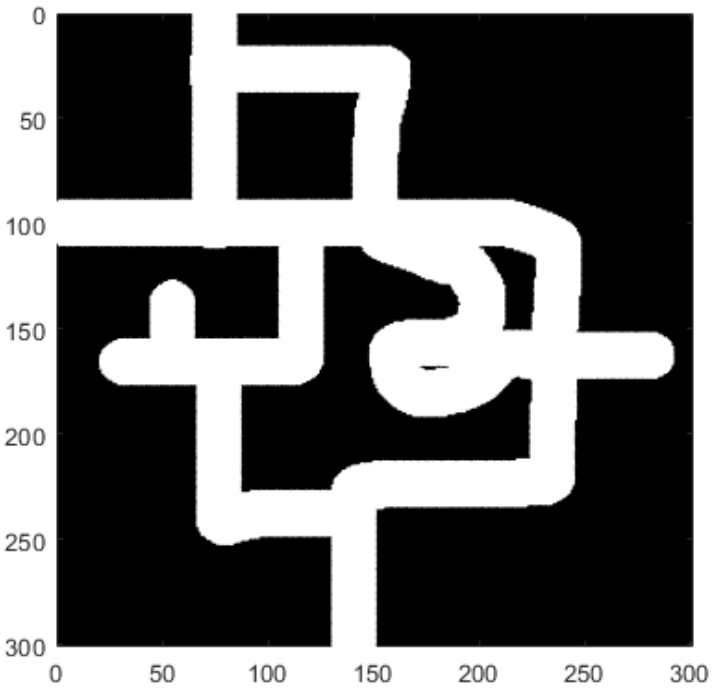
Le but de ce TP est de résoudre un problème de planification en espace hautement encombré. En l'occurrence, il s'agit de faire passer un robot dans un réseau de tuyau, représenté en 2D, donc dans le cas où tous les robots sont dans le même plan.
Le robot est un robot 4R dont la base peut se déplacer, et on ne se préoccupe pas de la façon dont le mouvement est généré.



On se retrouve alors dans un problème de planification de chemin d'un espace articulaire 6D ($Q_{free} = \Re \times [0,1]^5$), avec des contraintes statiques.

II. Méthode de résolution choisie

Nous avons choisi de discrétiser l'espace cartésien 2D, et de le représenter par une image matricielle binaire : un 1 (ou blanc) représente un espace libre et un 0 (ou noir) représente un espace innacessible. Voilà à quoi ressemble une carte : (ici de 300x300px)



A partir de cette représentation, nous avons décidé d'utiliser un PRM pour résoudre ce problème. Voici les points changeant d'un PRM classique que nous avons adapté pour ce problème :

- La génération aléatoire : elle a été modifiée pour être bien plus efficace qu'une méthode complètement aléatoire.
    - L'algorithme commence par choisir un *(x,y)* dans les pixels blancs (espace libre). La base ne posera donc pas de problème.
    - Ensuite, l'algorithme essaie au maximum 100 fois de trouver un angle θ_1 (aléatoirement) tel que le premier segment ne rentre pas en collision avec les obstacles.
    - On répète la tentative précédente jusqu'à avoir tous les segments, mais les thêta ne peuvent plus prendre n'importe quelle valeur, ils ont un angle maximum (réglable) qu'ils peuvent prendre (en valeur absolu).
    - Si jamais au bout de 100 tentatives pour n'importe quel theta on arrive pas à placer un segment, alors on recommence du tout début avec (x,y) différents.
    - Sinon on a une configuration valide : on peut l'ajouter au graphe.
- Le test de collision : il s'effectue en réalisant un échantillonnage le long de chaque bras et en vérifiant que le pixel le plus proche du point échantillonné est bien libre

- Distance entre deux configurations : pour économiser du temps de calcul nous avons décidé de ne pas utiliser les distances euclidiennes entre chaque point d'intersection des segments, mais une formule sur mesure bien plus rapide :

$$d = \alpha_1\sqrt{x^2 + y^2} + \sqrt{\alpha_2 \min\{\theta_1{}^2; (\theta_1 - 2\pi)^2; (\theta_1 + 2\pi)^2\} + \alpha_3\theta_2{}^2 + \alpha_4\theta_3{}^2 + \alpha_5\theta_4{}^2}$$

Pour essayer de correspondre au mieux à la distance réelle, nous avons établi

$$1 > \alpha_i > \alpha_{i+1} \; \forall i$$

En effet, la distance xy et la plus importante, puis les premiers angles sont plus importants que les derniers (car leur effet s'ajoute en bout de chaîne). De plus les coefficients sont inférieurs à 1 pour ne jamais sur-évaluer la distance, ce qui pourrait dérégler l'algorithme A*

- Planificateur local : Pour vérifier qu'il n'y a pas d'obstacle entre les configurations on utilise un échantillonnage le long d'une fonction linéaire entre les deux configurations (l'implémentation prend en charge le modulo 2π pour θ_1 afin d'éviter de passer par un chemin plus long que nécessaire) et on vérifie pour chaque configuration intermédiaire qu'il n'y ai pas collision.

III. Résultats de notre résolution

On a un algorithme qui permet de bien générer une trajectoire, elle n'est pas optimale mais c'est ce qu'on attend d'un PRM. On a par contre des chemins qui sont possibles dans le sens qu'ils ne rentrent en collision avec aucun obstacle. Ci joint au rapport, il y a deux animations obtenus avec cet algorithme (disponibles dans le dossier **out/**) :

- ***Non-optimal-path.mp4*** donne un exemple où le PRM n'est absolument pas la manière la plus optimale d'arriver au but,
- ***Quite-optimal-path.mp4*** donne un exemple, pour le coup, le chemin semble assez proche de ce que l'on pourrait attendre d'une résolution optimale.

IV. Architecture du code

Le code que l'on a réalisé cherche autant que possible à résoudre ce problème pour un robot N-R plan. On a alors décomposé le programme en plusieurs fichiers que l'on détaille ci-dessous.

- ***Main.m*** le seul programme qu'il est nécessaire de lancer (plus de détail dans la partie suivante),
- ***Astar/*** contient tous les programmes nécessaire pour utiliser l'algorithme A* sur un graphe représenté par une matrice creuse,
- ***Display/*** contient tous les programmes d'affichage,
    - ***get_map*** lit une image et la renvoie en matrice creuse binaire avec sa taille,
    - ***make_movie*** affiche une liste de configuration en faisant une animation,
    - ***save_movie*** enregistre l'animation créée par *make_movie* dans un fichier mp4 horodaté dans le dossier **out/**,
    - ***plot_robot*** affiche une configuration du robot, sur une figure déjà existante lorsqu'elle est passée en argument,
    - ***show_sparse*** affiche une matrice creuse en mettant les 1 en blanc et les 0 en noir en renvoyant la figure,
- ***Maps/*** contient les cartes (images) pour tester le PRM ,
- ***PRM/*** contient les algorithmes relatifs au calcul du PRM et des générations de positions aléatoires,
    - ***addConfiguration*** ajoute une configuration à un graphe,
    - ***collision*** détecte si une configuration donnée est en collision avec les obstacles,
    - ***createConfigurationArray*** génère un nombre donné de configuration et les renvoie dans un tableau,
    - ***createConfigurationGraph*** génère la totalité du graphe,
    - ***delta*** vérifie si un chemin est possible entre deux configurations par le biais d'une interpolation linéaire discrétisé,
    - ***generateRandomConfiguration*** génère une configuration aléatoire,
    - ***interpolation*** génère des configurations intermédiaires entre 2 autres configurations,
- ***Robot/*** contient les fonctions relatives aux calculs géométriques et/ou dépendantes du robot,
    - ***dgm*** calcule la position d'un point du robot par le numéro du bras et la longueur sur ce dernier,
    - ***distance*** calcule la distance entre deux configurations (voir II),
    - ***pos*** retourne la position en pixel d'un point du robot (paramétré par l'indice du bras auquel il est rattaché et de la longueur sur celui-ci),

V. Notice d’utilisation

Pour utiliser le programme, il suffit de lancer le script main, qui va s’occuper de lancer tous les autres scripts dans l’ordre. Point d'intérêt :

- *L* fixe à la fois les longueurs de chaque segment et le nombre de segment en fonction de sa taille
- On peut définir l’*angle_max* de chaque Θ_i, $i > 1$ pour éviter que des segments soient proche d’être les uns sur les autres
- *image_file* au début du programme permet de choisir l’image, à modifier éventuellement
- *density* permet de modifier la taille réelle que représente la carte (en pixel/m)
- *start* et *goal* sont à modifier selon la carte (s' ils ne sont pas possible atteignable l’algorithme ne marchera a priori jamais)
- *seg_sampling* et *conf_sampling* peuvent être intéressante à modifier pour etre plus precis / aller plus vite (échantillonnage respectivement pour les segments et les configuratiosn)
- Des propriétés d’animations peuvent aussi être changé, pour la rendre plus ou moins fluide

Après toutes les définitions, les algorithmes du PRM s'exécutent, et sont assez explicites, au besoin se référer à la partie précédente ou au commentaires inclus pour comprendre ce que fait chaque programme.