MAP-SIM2

Planification de trajectoire

Antoine ALMOUSSA
Waren LONG
Othmane SAYEM
Promotion 2017

Maître de conférence :
Mr Nicolas
KIELBASIEWICZ

17 Mars 2016

Table des matières

Introduction

La planification de trajectoire est un problème souvent rencontrer dans des domaines tels que la robotique ou les jeux vidéos. L'objectif consiste à déterminer le chemin le plus court entre deux points situés dans un environnement 2D comptant de nombreux obstacles polygonaux. Dans notre démarche, nous commencerons d'abord par traiter le cas où l'objet mobile est un point se mouvant uniquement de manière rectiligne, pour ensuite envisager le cas où il s'agit d'une sphère. Néanmoins, les 2 modèles nous considérons un objet pouvant changer de direction instantanément.

Pour trouver la trajectoire de longueur minimale, nous utiliserons une méthode de type recherche opérationnelle. Nous determinerons grâce à l'algorithme de Dijkstra la trajectoire minimale parmi celles reliant les sommets des obstacles.

1 Stratégie de résolution du problème

1.1 Principe de construction du graphe des chemins

Afin de contourner les obstacles, la trajectoire minimale reliant un point A à un point B, passera par différents sommets de ces obstacles. Il s'agira donc de construire un graphe constitué des sommets des obstacles. Chaque arc du graphe, faisant la connexion entre 2 sommets, possèdera une valeur correspondant à la longueur du segement reliant ces 2 sommets s'il n'intersecte aucun obstacle ou à une valeur infinie dans le cas contraire. Nous ajouterons enfin à ce graphe les points de départ et d'arrivée que nous nous sommes donnés ainsi que les connections correspondantes à tous les sommets des obstacles.

1.2 Factorisation du graphe

La construction de ce graphe est une opération couteuse en calculs et donc en temps. Au lieu de traiter chaque arc de la même manière et dans un ordre quelconque, nous avans donc cherché à décomposer cette construction en plusieurs étapes.

L'objectif de ce travail préalable est simple. Si nous souhaitons réutiliser certains obstacles d'une carte ou même une carte entière, il semble inutile de reconstruire la partie du graphe correspondant aux arcs intra et/ou inter-obstables. Il suffit de compléter le graphe de la carte précédente avec les nouveaux arcs reliants les points de départ et d'arrivée avec les sommets des différents obstacles.

Ainsi, nous avons décider de factotiser la construction d'un graphe suivant 3 étapes. Sachant qu'à chaque étage de construction, l'objectif est d'isoler les arcs invalides qui ne peuvent constituer un choix de chemin.

• 1ère étape : Construction des graphes propres à chaque obstacle.

Pour un obstacle donné, il s'agira de conserver les arcs correspondant aux contours qui sont évidemment valables, et d'éliminer (au sens de lui donner une valeur infinie) les arcs traversants l'intérieur de l'obstacle. Il faudra faire attention aux obstacles non-convexes dont les arcs reliant 2 sommets non-consécutifs peuvent être valables ou non.

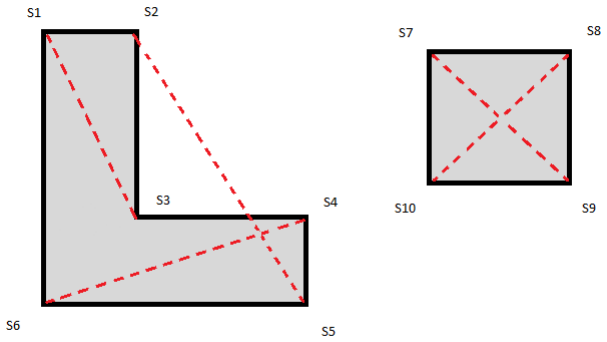

FIGURE 1 – 1ère étape : Tri intra-obstacle - construction des graphes propre à chaque obstacle

Le graphe ci-dessus illustre la 1ère étape du tri que nous opérons dans la construction de notre graphe. Ici, les arcs en pointillé rouge correspondent à des arcs de valeur infinie. Ces arcs traversent l'intérieur de l'obstacle. Nous ne les représenterons pas tous pour des raisons de lisibilité.

• 2ème étape : Construction du graphe de la carte entière.

Ici, on ajoute au graphe précédent, les arcs reliants les sommets de 2 obstacles différents et n'intersectant aucune arrête. En effet, à partir du moment où il y a une intersection, l'arc traverse un obstacle et n'est donc pas valable : on lui donne une valeur infinie.

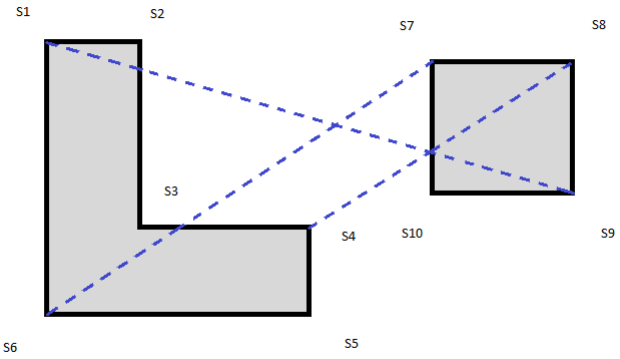

FIGURE 2 – 2ème étape : Tri inter-obstacle - construction du graphe de la carte

Le graphe ci-dessus illustre lui la 2ème étape du tri, toujours dans la construction de notre graphe. Les arcs en pointillé bleu représentent ceux auxquels nous avons attribuée une valeur infinie à l'issue de la 2ème étape du tri. Ces arcs traversent l'intérieur d'un des obstacles de la carte, peu importe lequel.

• 3ème étape : Ajout des points de départ et d'arrivée.

En ajoutant ces 2 derniers points, il ne nous reste plus qu'à traiter les arcs reliant chacun de ces 2 points avec tous les sommets des obstacles de la carte.

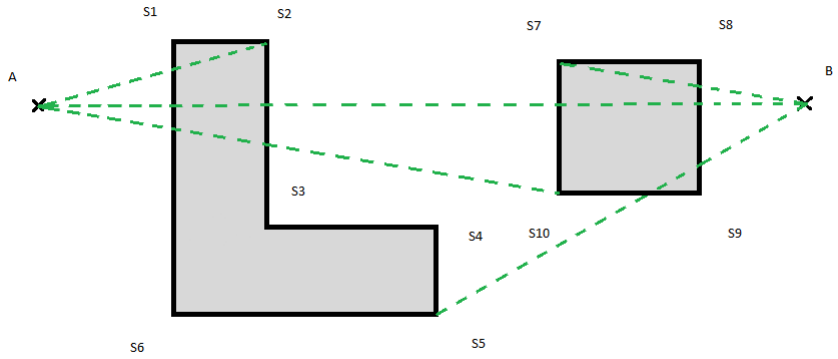

FIGURE 3 – 3ème étape : Tri des arcs issus des points de départ et d'arrivée - construction du graphe final

Sur le graphe ci-dessus illustrant cette 3ème étape de tri, on peut voir certains de ces arcs auxquels nous avons donné une valeur infinie car ils intersectent l'intérieur d'au moins un obstacle de la carte. Ils sont ici représentés en pointillé vert.

2 L'algorithme de Dijkstra

Nous disposons maintenant d'un graphe où chaque arc dispose d'une valeur : infinie si l'arc correspond à un chemin inenvisageable, ou égale à la longueur du segment si le chemin n'intersecte aucun obstacle. Il s'agit alors de calculer le plus court chemin entre un point de départ et un d'arrivée. Pour cela, nous avons fait appel à l'algorithme de Dijkstra, comme le suggerait l'énoncé.

La fonction prend en argument : le nombre d'obstacles, la matrice des coûts, deux vecteurs d'entiers qui listent les sommets et deux vecteurs qui vont contenir les parents de chaque sommet ainsi que le coût. Pour permettre le renvoi de deux objets, nous modifions ces deux vecteurs par référence. L'algorithme implémenté est insipiré du prototype de l'énoncé.

3 Les objets manipulés

Mise à part les classes demandées dans l'énoncé, notamment la classe Obstacle et la classe Arc, on a ajouté plusieurs fonctions membres qui ont été nécessaires pour l'établissement du Padding et la détéction des intersections.

Dans la classe Obstacle, on a défini :

— **Constructeur par vecteur** construit un obstacle à partir d'un vecteur de points, dans le sens trigonométrique. En effet, il copie les sommets au début, puis il crée le graphe correspondant tout en génerant un vecteur de normales aux arêtes.

— **Arcs_diag** est une fonction qui retourne un graphe avec tous les arcs physiquement possibles. Si l'obstacle est convexe, on conserve les arcs correspondant aux contours de l'obstacle, mais s'il ne l'est pas, **Arcs_diag** ajoute les arcs diagonaux qui manquent qui relient des sommets non-consécutifs.

— **create_point** est la fonction clé du Padding. Elle prend comme arguments, le nombre d'itérations (finesse du padding) et le rayon de l'objet. Elle écrit sur un fichier à part, les nouvelles coordonnées des différents sommets de l'obstacle après application de la méthode du Padding.

D'autre part, dans le fichier "couts.hpp" on a défini :

— **intersection** elle prend en arguments deux points A et D, puis un vecteur d'osbtacles. Son rôle est de déterminer si l'arc (DA) coupe l'un des obstacles ou pas.

— **arc_arrete** Cette fonction sera nécessaire pour l'écriture de la matrice des couts. En effet, elle vérifie si un arc donné est physiquement possible dans notre graphe ou pas.

— **couts** c'est la fonction qui retourne la matrice des coûts nécessaire à l'application de l'algorithme Dijkstra.

Pour simplifier notre programme, on a rassemblé nos fonctions classes dans une seule fonction :

— **create_graphe** elle lit les coordonnées disponibles dans le fichier txt, crée un graphe avec les différents obstacles, applique l'algorithme de Dijkstra, puis renvoie le vecteur p, qui contient les prédécesseurs des sommets. Elle écrit ses coordonnées dans un autre fichier "Dijkstra.txt" qui sera réutilisé dans Matlab afin de dessiner les obstacles polygonaux et tracer la trajectoire optimale entre le point de départ et d'arrivée.

— **create_graphe_padding** elle fait le même travail que "create_graphe", mais en appliquant la méthode du padding, en considérant l'objet comme un disque. Elle lit les coordonnées, crée les obstacles, applique "create point" pour avoir les obstacles courbés, réecrit leurs coordonnées puis applique Dijkstra sur ses sommets et réecrit le vecteur p des prédécesseurs.

4 Cas d'un objet non ponctuel : Méthode du Padding

Nous avons effectué le padding comme conseillé dans l'énoncé : nous gardons un objet ponctuel et nous envisageons des obstacles agrandis d'un certain rayon choisi par l'utilisateur. Pour ceci, nous créons une liste de points de la manière suivante : nous créons le point qui se trouve sur la bissectrice sortante au sommet de l'obstacle, à une distance égale au rayon de l'objet. Nous effectuons la rotation du point créé autour du sommet de l'obstacle d'un angle pi/4 dans le sens horaire. L'utilisateur choisit un certain nombre d'itérations de rotation autour du sommet et donc le nombre de points qui vont être créés pour contourner l'obstacle. Ces points sont créés par des rotations autour du sommet de l'obstacle. Enfin, nous créeons un obstacle avec la liste de points. Voici 2 exemples sur 2 géométries différentes. L'obstacle est en gris, et le nouvel obstacle issu du padding défini par les contours rouges.

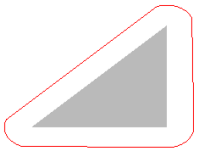

FIGURE 4 – Triangle avec Padding

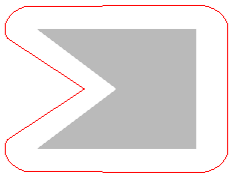

FIGURE 5 – Polygone avec Padding

On remarque que dans le cas d'un creux au sein du polygone, il n'est plus question de rotation autour du sommet. Le disque se coince dans le creux puis repart. Nous avons rencontré cette difficulté quuelques instants puisqu'il fallait déterminer ces angles creux.

Voici quelques exemplesde trajectoires avec du padding.

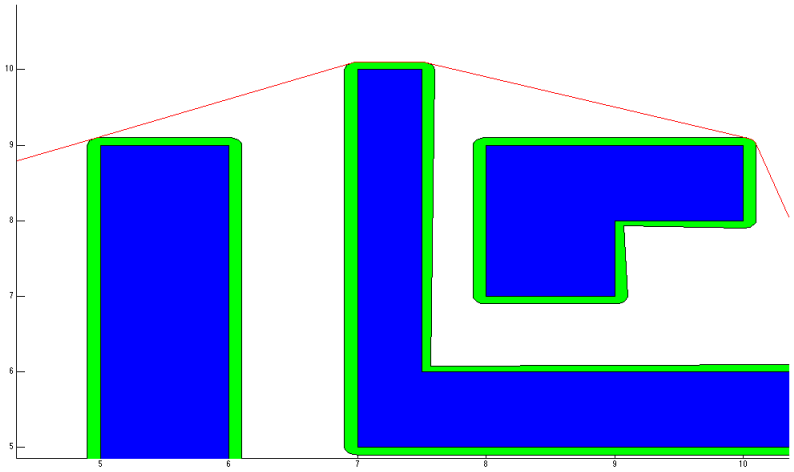

FIGURE 6 – Plus court chemin dans une carte où les nouveaux obstacles après padding sont en verts

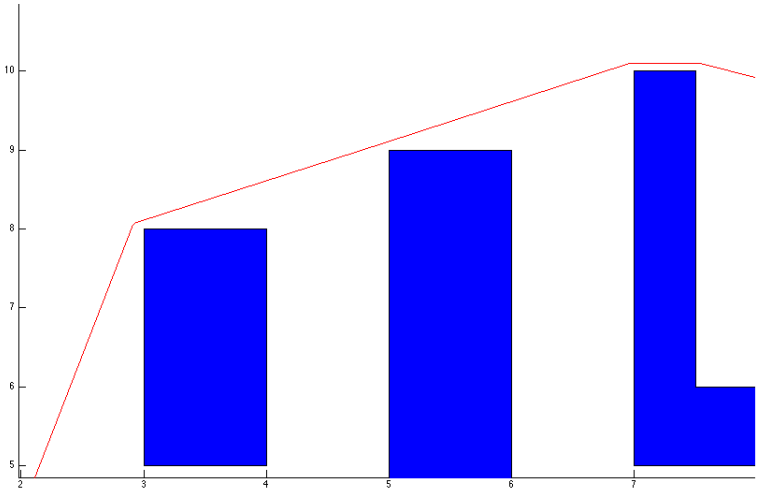

FIGURE 7 – Plus court chemin dans une carte où les nouveaux obstacles après padding ne sont pas visibles

5 Répartition des tâches au sein du groupe

Après réception des différents documents nécessaires pour le projet, Nous avons d'abord essayer de comprendre ce qui nous était demandé. Nous avons alors discuté des différentes difficultés que présentaient le sujet et nous avons effectué quelques recherches sur Internet à propos de l'algorithme de Djikstra afin de mieux le comprendre.
Ensuite, nous nous sommes répartis les premières tâches nécessaires pour le lancement du sujet :

— **Othmane** s'est principalement occupé des classes OBSTACLE, ARC, GRAPHE ;

— **Antoine** s'est occupé de coder l'algorithme Dijkstra en se basant notamment sur l'exemple donné dans l'énoncé. Il a également commencé à comprendre la technique du Padding.

— **Waren** a revu en premier les classes POINT, MATRICE, VECTEUR. Ensuite, il a codé la fonction clé de notre projet "intersect" qui détecte l'intersection entre différentes droites.

En avançant dans le code, et durant les scéances du Jeudi matin, on se réunissait pour voir à quel point chacun de nous avait progressé. Nous présentions également les problèmes rencontrés lors de la compilation, et essayions de les résoudre ensemble.
Après cette première partie, où nous avons établi les principales classes du projet, nous nous sommes accordés sur une seconde organisation du travail :

— **Antoine** Après avoir terminé l'algorithme de Djikstra, il s'est occupé de l'entière mise en place du Padding.

— **Waren** s'est intéressé à la problématique des obstacles non convexes. Il a donc codé une fonction arcs_diag qui retourne un graphe avec tous les arcs possibles, y compris les arcs diagonnaux possibles.

— **Othmane** s'est intéressé à la finalisation de la fonction "couts" qui retourne la matrice des coûts.

6 Les difficultés rencontrées

Vu que c'était un travail de groupe, les principales difficutés rencontrées durant la programmation étaient liées à la combinaison et l'utilisation des résultats individuels de chacun d'entre nous afin de les tester dans l'ensemble. À plusieurs reprises, les fonctions écrites par Antoine n'étaient pas compatibles avec les classes codées par Othmane et Waren, notamment pour les classes de base telles que Point et Obstacle. Nous avons donc été amené à compléter ces classes tout au long du projet dès que cela était nécessaire.

On a également perdu beaucoup de temps en travaillant sur les obstacles non convexes : l'utilisation des normales des points ne nous permettait pas toujours de détecter les points intérieurs aux obstacles. Nous avons donc utilisé le produit vectoriel pour bien démontrer leur existence, mais également pour déterminer l'intersection physique et réelle entre deux arcs du graphe.

Nous avons rencontré une autre difficulté en traitant la méthode du Padding : encore une fois, les obstacles concaves posaient problème. En effet, au niveau des points intérieurs, creux, il ne fallait pas faire tourner le sommet mais les laisser comme ils étaient, tout en agrandissant les bords.

7 Résultats obtenus et Commentaires

Après traitement des difficultés que nous avons rencontrées, nous avons pu commencer à jouer avec nos résultats. Par exemple, nous avons créé une carte un minimum complexe : un labyrinthe composé d'obstacles convexes et non-convexes. Nous avons ensuite placé les points de départ et d'arrivée de part et d'autre du labyrinthe. Nous nous sommes alors amusé à comparer les plus courts chemins obtenus dans le cas de l'objet ponctuel (labyrinthe noir) et dans le cas du padding (labyrinthe bleu).

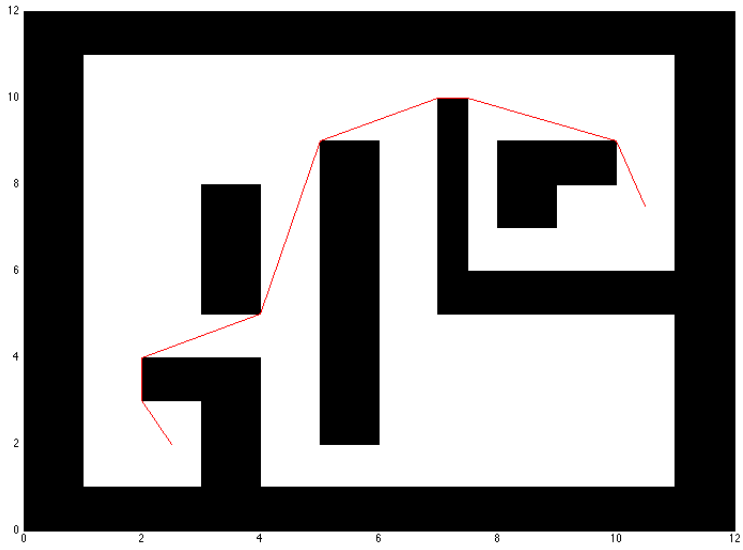

FIGURE 8 – Plus court chemin obtenu dans le cas d'un objet ponctuel

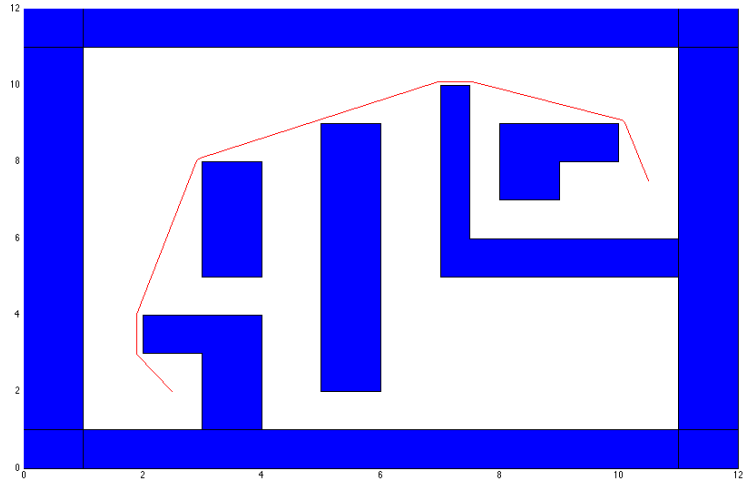

FIGURE 9 – Plus court chemin obtenu dans le cas d'un objet non ponctuel en forme de disque

On remarque immédiatemment que le plus court chemin obtenu n'est pas le même dans les 2 cas. Pourtant il s'agit bien de la même carte, du même labyrinthe avec les mêmes points de départ et d'arrivée. Le padding est donc une technique suffisamment sensible pour changer un plus court chemin. En effet, on comprend bien qu'avec la méthode du padding, le passage de l'objet par un sommet d'un obstacle est une opération plus coûteuse en distance. L'algorithme cherchera donc à limiter le passage de l'objet par des sommets. Lacomparaison des 2 chemins ci-dessus valide cette hypothèse.

Conclusion

Le problème de planification de trajectoire est un problème qui a suscité notre intérêt dès les premières réflexions de par son caractère concret et très visuel. Tout au long de ce projet, il nous a fallu faire des allers retours entre le code et des raisonnements géométriques. De plus, nous avons apprécié la certaine liberté que nous disposions dans la résolution du sujet. En effet, plusieurs méthodes semblaient possibles pour traiter ce problème de planification de trajectoire. Nous avons choisi de construire le graphe par échelle croissante afin de faciliter la réutilisation du code et de limiter le temps de calcul.