

PANDROIDE

Planification de mouvements par déformation de l'espace

Gaspard Ducamp et Yoann Taillé

Mai 2016

Table des matières

1	Introduction	3
2	Description des méthodes retenues	4
2.1	Approche sans prise en compte du seuil	4
2.2	Approche prenant en compte le seuil	5
3	D'autres façons de faire	6
3.1	Déformation utilisant une symétrie	6
3.2	Détection du blocage	6
3.3	Le cas de l'inclusion dans un obstacle	6
4	Exemple de déformations effectuées	7

1 Introduction

Ce document traite des différentes méthodes que nous avons imaginées et utilisées dans le cadre de notre projet. Le but étant de modéliser la traversée par un robot d'une pièce possiblement encombrée d'obstacles, nos algorithmes reposent sur un principe commun : identifier les zones de la pièce pouvant poser problème et les utiliser pour déterminer un trajet convenable. La notion de seuil abordée plus loin correspond à une distance minimale à respecter, le robot pouvant avoir une épaisseur.

Les parties de ce texte écrites en italiques permettent de relier la description de la méthode à notre code.

2 Description des méthodes retenues

2.1 Approche sans prise en compte du seuil

Dans cette version de notre algorithme, le seuil n'est utilisé que pour fusionner les objets. Il est donc possible que le chemin retourné longe de très près les obstacles. Notre algorithme (implémenté dans *algo.appel.appelPieceBloque*) est le suivant :

- On identifie les obstacles et les murs de la pièce (*outils.lectureFichier.lecturePieceObs*), et on leur ajoute un certain nombre de points (*outils.outils.redefineForme*).

- On fusionne les obstacles avec les murs ou les autres obstacles trop proches, de manière à respecter une distance égale au double du seuil (*outils.outils.fusion* pour les obstacles, *outils.outils.fusionMur* pour les murs). Si un obstacle est trop proche d'un mur, il est intégré aux murs de la pièce

- On dégage les fractions des murs à considérer comme des obstacles (*outils.outils.ramenerRect*). Pour cela, on inscrit la pièce dans un rectangle et on identifie les portions de la pièce qui s'écartent, complétées des sections du rectangle qui n'ont pas été respectées. On retient au passage les points de début et de fin "d'écartement", qui seront plus tard utilisés comme centres de réduction. Si la partie de mur contient un angle du rectangle, on ajoute cet angle à la liste des centres.

Ce qui suit est implémenté dans *algo.appel.calculerLigneBloque*

- On liste les obstacles et les morceaux de murs sur le chemin (*outils.outils.obsbloquants*).
- Tant qu'il en existe, on les parcourt :
 - Si l'obstacle est une partie de mur, on le réduit autour du point le plus proche parmi la liste des centres de réduction calculés précédemment, jusqu'à ce que le chemin ne passe plus par la forme réduite (*outils.outils.defomin* et *outils.outils.deformerPourc*).
 - Sinon, on inscrit l'obstacle dans un rectangle, dont on détermine le point le plus éloigné du chemin. On prend comme centre de réduction le point de l'obstacle le plus proche du point précédent, et l'on réduit l'obstacle jusqu'à ce que le chemin ne passe plus par la forme réduite (*outils.outils.genCible*).
 - On applique l'algorithme de déformation sur la ligne en prenant comme cible de déformation de l'obstacle sa réduction (*algo.deformAlgo.algo*).
 - On liste de nouveau les obstacles et les sections de murs sur le chemin.
 - Si l'on a déformé un obstacle plus d'un nombre de fois établi, on sort de la boucle.
- On retourne la ligne successivement déformée.

La terminaison de l'algorithme est donc garantie soit par le fait que le chemin ait bien été calculé, soit par le fait que l'on ait dépassé le nombre de déformations autorisé.

2.2 Approche prenant en compte le seuil

Dans cette version, le seuil est considéré comme une distance à respecter entre les objets et le chemin. Le robot ne pourra donc pas passer entre certains encombrants trop proches. Elle repose sur l'agrandissement des obstacles :

- On identifie les obstacles et les murs de la pièce (*outils.lectureFichier.lecturePieceObs*).
- On dégage les fractions des murs à considérer comme des obstacles. Pour cela, on inscrit la pièce dans un rectangle et on identifie les portions de la pièce qui s'en écartent, complétées des sections du rectangle qui n'ont pas été respectées. On retient le rectangle circonscrit à la pièce (*outils.outils.ramenerRectAgr*).
- On agrandit tous les obstacles selon le seuil. Pour ce faire, on détermine pour chaque point de l'obstacle un carré dont il est le centre, et ayant le double du seuil pour longueur de côté. On sépare ce carré en quatre carrés inscrits de longueur de côté égale au seuil. On évalue lesquels parmi les quatre ont le plus de coins dans l'obstacle, et on ajoute à ce dernier les coins des carrés retenus diagonalement opposés au centre (*outils.outils.agrandiradd*).
- On ajoute aux obstacles un nombre de points calculé de manière à ce qu'ils soient séparés d'une distance égale au seuil (*outils.outils.redefineseuil*).
- On fusionne les obstacles trop proches entre eux. Les polygones étant agrandis, on vérifie pour cela si des points d'un obstacle sont inclus dans un autre, et l'on les fusionne formes le cas échéant, en ne retenant que les points qui ne sont pas partagés (*outils.outils.fusionAgr*).
- On parcourt la liste des obstacles agrandis, en cherchant pour chacun la liste de leurs points extérieurs au rectangle circonscrit à la pièce (*outils.outils.pointsdehors*). Si celle-ci n'est pas vide, l'obstacle est donc considéré comme faisant partie de l'un des murs. Cette liste de points sera utilisée comme potentiels centres de réduction par la suite.

Ce qui suit est implémenté dans *algo.appel.calculerLigneAgr*

- On liste les obstacles agrandis, dont font donc partie les portions de murs, bloquant le chemin (*outils.outils.obsbloquants*).
- Tant qu'il en existe, on les parcourt :
 - Si l'obstacle est une partie de mur, on le réduit autour du point le plus proche parmi la liste des centres de réduction calculés précédemment, jusqu'à ce que le chemin ne passe plus par la forme réduite (*outils.outils.defomin* et *outils.outils.deformerPourc*).
 - Sinon, on inscrit l'obstacle dans un rectangle, dont on détermine le point le plus éloigné du chemin. On prend comme centre de réduction le point de l'obstacle le plus proche du point précédent, et l'on réduit l'obstacle jusqu'à ce que le chemin ne passe plus par la forme réduite (*outils.outils.genCible*).
 - On applique l'algorithme de déformation sur la ligne en prenant comme cible de déformation de l'obstacle sa réduction (*algo.deformAlgo.algo*).
 - On liste de nouveau les obstacles et les sections de murs sur le chemin.
 - Si le nombre de déformations d'un obstacle dépasse une certaine limite, on sort de la boucle.
- On retourne la ligne successivement déformée.

La terminaison de l'algorithme est encore une fois garantie soit par le fait que le chemin ait bien été calculé, soit par le fait que l'on ait dépassé le nombre de déformations autorisé.

3 D'autres façons de faire

Nous avons aussi implémenté et testé des procédés alternatifs ayant pour but d'obtenir un chemin ne passant pas par les obstacles.

3.1 Déformation utilisant une symétrie

L'une des approches testées pour déformer la ligne consiste à prendre comme cible de déformation le symétrique d'un obstacle, dans le cas où celui-ci fait partie d'un mur. L'axe de symétrie choisi est le segment entre le premier et le dernier des points s'écartant du rectangle circonscrit à la pièce. Ainsi, on obtient une déformation généralement assez forte pour ne pas avoir à déformer de nouveau l'obstacle.

Cependant, la déformation obtenue appliquée au chemin peut être trop importante et n'est donc pas adaptée au déplacement dans un environnement étroit. De plus, la nature trop spécifique (car seulement applicable aux obstacles tirés des murs) de cette approche nous a poussé à ne pas la retenir.

3.2 Détection du blocage

Dans l'optique de la détection du blocage du chemin et de la fusion entre obstacles, nous avons choisi de tester l'inclusion d'un point dans un polygone. Toutefois, une fois implémenté en Python, ce test dépend de l'ordre des points de la forme. Une approche parfois plus fiable consiste à évaluer la distance entre les points. Elle nécessite néanmoins un grand nombre de points définissant à la fois le chemin et les obstacles, rendant ainsi les calculs plus coûteux (il faut, pour tous les points de la ligne, tester leurs distances à tous les points de l'obstacle).

3.3 Le cas de l'inclusion dans un obstacle

Le fait que le point de départ ou d'arrivée du trajet soit enfermé dans un obstacle est un cas rare mais vraisemblable et particulièrement gênant dans le cadre de la recherche d'un chemin. En effet, si la seule sortie possible d'un obstacle est orientée dans la direction opposée à la destination, il devient compliqué de déterminer un passage par déformation de l'espace.

L'une des solutions possibles repose sur la décomposition de l'obstacle en portions exploitables (*outils.outils.decoupons*), et à la déformation successive de ses différentes parties.

Notre expérimentation a consisté à diviser récursivement en quatre le rectangle contenant le point bloqué jusqu'à ce qu'il ne contienne plus de partie de l'obstacle (et qu'il ne contienne donc plus que ledit point). En envoyant l'une après l'autre une réduction d'une section d'obstacle dans la suivante, tout en conservant les déformations effectuées, et en appliquant ces déformations à la ligne dans l'ordre inverse, nous avons obtenu des débuts de résultats prometteurs. Toutefois, n'ayant pas trouvé de moyen de généraliser l'approche, nous avons décidé de laisser ce problème de côté.

4 Exemple de déformations effectuées

Ci-dessous nous pouvons observer les différentes déformations qui ont été faites lors d'une exécution. Les formes rouges (resp. jaunes) correspondent aux formes cibles lors des déformations (resp. déformations inverses).

D'autres exemples sont visibles dans le manuel utilisateur.

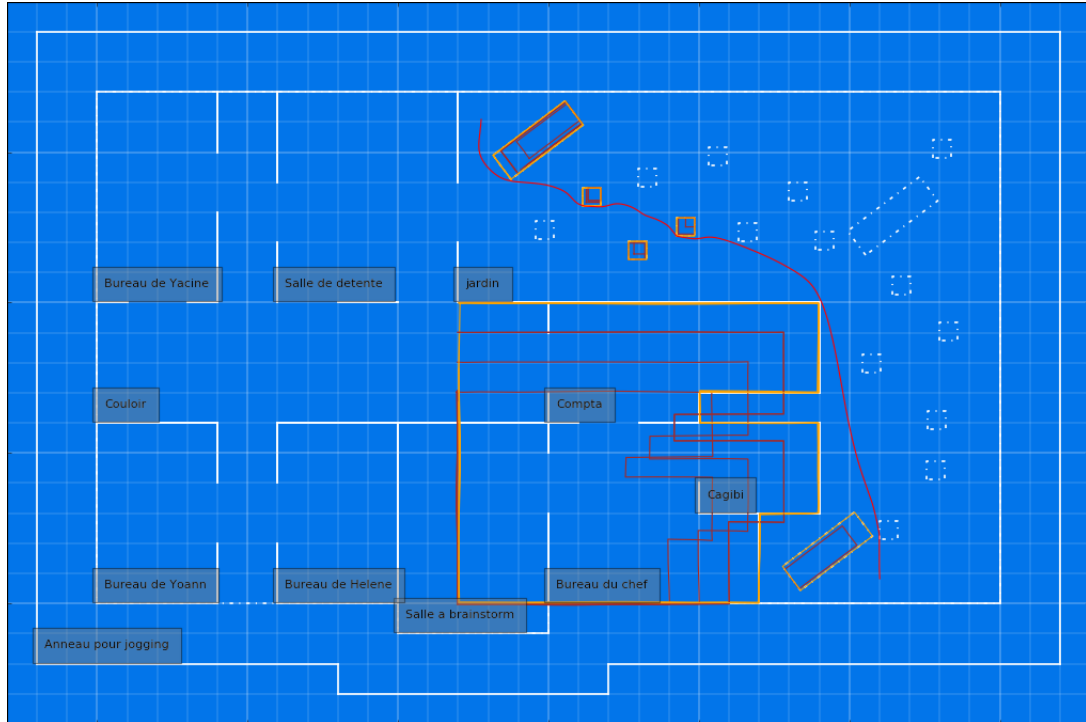


FIGURE 1 – Résultat obtenu lors d'un parcours sans seuil