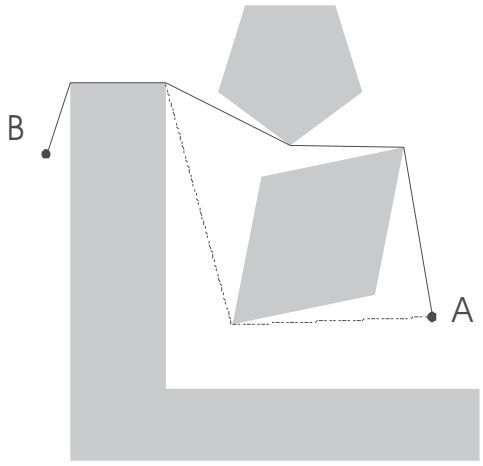
Planification de trajectoire

MAP-SIM2

2017

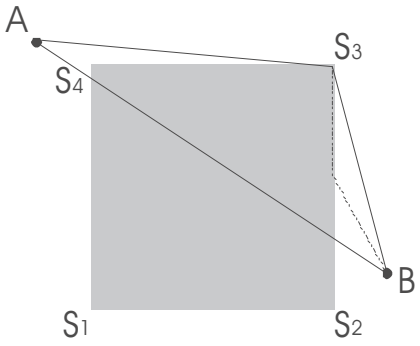
La planification de trajectoire consiste à déterminer une trajectoire, en générale la meilleure, que doit suivre un objet pour aller d'un point à un autre dans un environnement complexe (nombreux obstacles, terrain de nature différente).. On rencontre ce problème surtout en robotique mais également dans les jeux vidéo. On peut distinguer plusieurs cas de difficulté croissante : le cas où l'objet est assimilé à un point et peut changer de direction instantanément, le cas où l'objet à une forme donnée mais peut toujours changer de direction instantanément et enfin le cas où l'objet a une dynamique propre, c'est-à-dire que son mouvement n'est plus nécessairement rectiligne (une voiture par exemple). Pour simplifier, on se placera sur un terrain plat sur lequel des obstacles (polygones) sont présents.



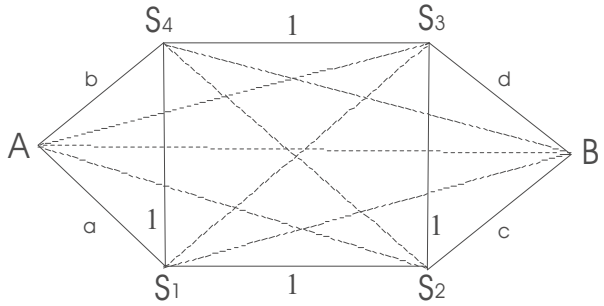
On s'intéresse à la recherche des trajectoires de longueur minimale. Il existe plusieurs méthodes pour la trouver. Nous proposons d'utiliser une méthode de type recherche opérationnelle basée sur la construction des trajectoires minimales reliant les sommets des obstacles.

1 Principe de la méthode

Lorsqu'il n'y a pas de dynamique, la trajectoire minimale reliant un point A à un point B est le segment AB si le segment n'intersecte pas d'obstacles. Il est assez clair que si le segment rencontre un obstacle la trajectoire minimale reliant A et B, passera par un sommet de l'obstacle.



On va donc construire un graphe constitué des sommets des obstacles, chaque arc du graphe (connection entre 2 sommets) aura une valeur : la longueur du segment reliant les deux sommets si le segment ne rencontre aucun obstacle et une valeur infinie sinon. On ajoutera à ce graphe les points de départ et d'arrivée que l'on s'est donné et les connections à tous les somets des obstacles.



Dans le graphe ci-dessus les arcs en pointillé correspondent à des arcs de valeur infinie.
On se ramène ainsi à un problème classique de recherche du plus court chemin dans un graphe pour lequel il existe des algorithmes performants, algorithme de Dijkstra par exemple (voir annexe).

2 Implémentation

2.1 Sommet

On écrira une classe **sommet** ainsi qu'une classe **segment**.

2.2 Obstacle

Afin de gérer les obstacles, on utilisera une classe **obstacle** décrivant un obstacle par ses sommets ordonnés. Il sera utile de disposer des normales de chaque arête de l'obstacle pour détecter l'intérieur de l'obstacle. Si vous créez les sommets d'un obstacle en tournant dans le sens trigonométrique, la normale sera donnée par la rotation de $\pi/2$ du vecteur directeur de l'arête $\overrightarrow{S_iS_{i+1}}$. Dans cette classe, on pourra écrire la fonction détectant l'intersection d'un segment (ouvert) avec l'obstacle.

2.3 Graphe des chemins

Un arc du graphe est un triplet $(S_1, S_2, longueur)$ où S_1, S_2 sont deux sommets. On écrira donc une classe **arc** ainsi qu'une classe **graphe** qui sera essentiellement constituée d'une liste d'arc. Il sera judicieux de ne pas incorporé les arcs de longueur infinie dans cette liste ce qui limitera les calculs dans l'algorithme de recherche du chemin minimal. Afin de développer un algorithme "générique", à savoir qui travaille avec des arcs et des graphes quelconques indépendants du problèmes que l'on traite, on utilisera le polymorphisme dans le cadre de l'héritage de classe qui permet de manipuler une classe **arc_planification** (dérivant de la classe **arc**) comme une classe **arc**.

2.4 Validation

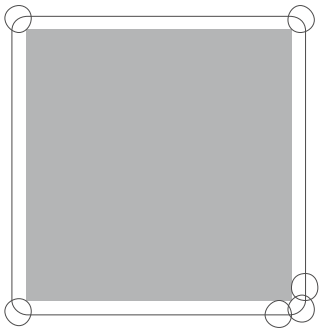
On validera chaque fonctionnalités élémentaires, on réalisera des tests globaux simples (chemin autour d'un obtacle carré, ...). Prévoir de tester le cas où il n'y a pas de chemin possible. Enfin on réalisera des tests sur des configurations complexes (plusieurs obstacles de formes différentes).

2.5 Représentation graphique

Afin de visualiser les trajectoires obtenues, le plus simple consiste à générer un fichier contenant les obstacles et la trajectoire calculée, à le relire dans Matlab et à utiliser les fonctionnalités graphiques simples de Matlab pour réaliser le graphique.

3 Objet non ponctuel

On étendra la méthode précédente au cas où l'objet n'est plus ponctuel mais est un disque. La technique la plus simple consiste à utiliser le "padding" : cette métode consiste à modifier les obstacles en considérant comme nouvel obstacle l'ancien obstacle épaissi de la couche générée par le glissement de l'objet autour de l'obstacle.



On notera que le nouvel obstacle n'est plus un polygone (les coins sont "devenus" des quarts de cercle de rayon le rayon de l'objet). En première approximation, on discrétisera ces quarts de cercle en une ligne polygonale (8 segments par exemple) et on se ramènera ainsi au cas d'un obstacle polygonal avec un objet ponctuel (centre du disque).
L'extension au cas d'objet de forme quelconque peut-être traitée de la même façon si on suppose que l'objet ne tourne pas sur lui même. Lorsque que l'objet peut tourner sur lui même, le problème se complique singulièrement car alors la trajectoire de l'objet est décrite par la position et l'orientation de l'objet (un angle). On se place alors dans $\mathbb{R}^2 \times [0, 2\pi[$, espace (point, direction), les obstacles devenant des domaines de $\mathbb{R}^3$, chaque "tranche" du domaine correspondant à une direction donnée de l'objet. L'obstacle volumique n'est pas un polyèdre, mais en discrétisant les directions suivant un nombre donné d'angle on se ramène au cas d'un polyèdre. Pour trouver la trajectoire minimale on peut encore appliquer la méthode précédente mais dans l'espace Un changement de direction sans déplacement correspond à un "déplacement" spatial dans la direction verticale, on peut éventuellement attribuer une valeur à tel déplacement (par exemple proportionnel à la différence angulaire).
On traitera le cas où l'objet est un disque ou un objet quelconque (ne tournant pas sur lui même) en écrivant une fonction transformant un obstacle à l'aide de la méthode décrite précédemment.

4 Organisation du travail (3p)

On pourra adopter le partage suivant du travail après s'être concerté sur la définition des classes générales (point, segment, arc, graphe). Un élève se consacrera au développement de ces classes, travail à effectuer de façon rapide puis au développement des outils d'initialisation et de représentation graphique. Pendant ce temps, un autre élève s'interessera à la mise en oeuvre de l'algorithme de Dijkstra, tandis que le dernier pourra s'atteler au développement de la technique de padding. Une fois chaque partie développée, on pourra les réunir pour réaliser l'outil de planification dans le cas d'objet qui ne tournent pas.

5 Extension possible

Le cas d'un objet tournant sur lui même.

6 Annexe : algorithme de Dijkstra

Cette annexe est tirée du site http://www.apprendre-en-ligne.net/graphes/dijkstra/algorithme.html. Il existe de nombreux liens sur Internet relatifs à cet algorithme.

Edgser Wybe Dijkstra (1930-2002) a proposé en 1959 un algorithme qui permet de calculer le plus court chemin entre un sommet particulier et tous les autres. Le résultat est une arborescence.

6.1 Principe de l'algorithme

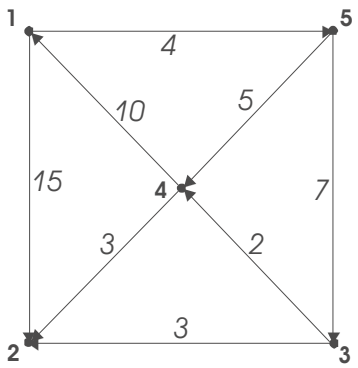
Numérotons les sommets du graphe $G = (V, E)$ de 1 à n. Supposons que l'on s'intéresse aux chemins partant du sommet 1. On construit un vecteur $l = (l(1); l(2); ...; l(n))$ ayant n composantes tel que $l(j)$ soit égal à la longueur du plus court chemin allant de 1 au sommet j. On initialise ce vecteur à $c_{1,j}$, c'est-à-dire à la première ligne de la matrice des coûts du graphe, définie comme indiqué ci-dessous:
où $d(i, j)$ est le poids (la longueur) de l'arc (i, j). Les c_{ij} doivent être strictement positifs.
On construit un autre vecteur p pour mémoriser le chemin pour aller du sommet 1 au sommet voulu. La valeur $p(i)$ donne le sommet qui précède i dans le chemin.
On considère ensuite deux ensembles de sommets, S initialisé à $\{1\}$ et T initialisé à $\{2, 3, ..., n\}$. À chaque pas de l'algorithme, on ajoute à S un sommet jusqu'à ce que $S = V$ de telle sorte que le vecteur l donne à chaque étape le coût minimal des chemins de 1 aux sommets de S.

6.2 Description de l'algorithme

On suppose ici que le sommet de départ (qui sera la racine de l'arborescence) est le sommet 1. Notons qu'on peut toujours renuméroter les sommets pour que ce soit le cas.

```
Initialisations : l(j) = c1,j et p(j) = NIL, pour 1 j n
                    Pour 2 j n faire
                      Si c1,j < alors p(j) = 1.
                    S = {1} ; T = {2, 3, ..., n}.
Iterations      : Tant que T n'est pas vide faire
                    Choisir i dans T tel que l(i) est minimum
                    Retirer i de T et l'ajouter a S
                    Pour chaque successeur j de i, avec j dans T, faire
                       Si l(j) > l(i) + d(i,j) alors
                       l(j) = l(i) + d(i,j)
                       p(j) = i
```

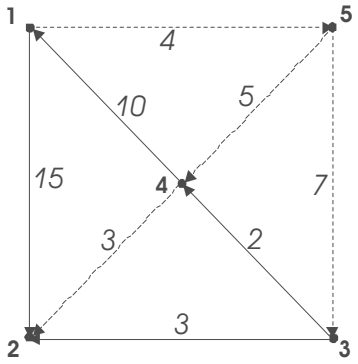
6.3 Exemple



- **Initialisation** : S = {1} ; T = {2, 3, 4, 5} ; l = (0, 15, , , 4); p = (NIL, 1, NIL, NIL, 1)

- **1ère itération** : i = 5 car l(5) = min(15, , , 4) = 4; S = {1, 5}; T = {2, 3, 4};
  les successeurs de 5 dans T sont 3 et 4;
  l(3) prend la nouvelle valeur min(; l(5) + d(5; 3)) = min(; 4 + 7) = 11; p(3) = 5;
  l(4) prend la nouvelle valeur min(; l(5) + d(5; 4)) = 9; p(4) = 5;
  d'où les nouveaux vecteurs l = (0, 15, 11, 9, 4) et p = (NIL, 1, 5, 5, 1)
- **2e itération** : i = 4; l(4) = 9;S = {1, 5, 4}; T = {2, 3};
  le seul successeur de 4 dans T est 2;
  l(2) prend la nouvelle valeur min(15; l(4) + d(4; 2)) = min(15; 9 + 3) = 12; p(2) = 4;
  d'ou les nouveaux vecteurs l = (0, 12, 11, 9, 4) et p = (NIL, 4, 5, 5, 1)
- **3e itération :**i = 3; l(3) = 11;S = {1, 5, 4, 3}; T = {2};
  le seul successeur de 3 dans T est 2;
  l(2) garde sa valeur car min(12; l(3) + d(3; 2)) = min(12; 11 + 3) = 12;
  d'ou les vecteurs inchangés l = (0, 12, 11, 9, 4) et p = (NIL, 4, 5, 5, 1)
- **4e itération** ; i = 2; l(2) = 12;S = {1, 5, 4, 3, 2}; T={};
- **FIN.** l = (0, 12, 11, 9, 4); p = (NIL, 4, 5, 5, 1).

Le chemin minimal de 1 à 4 par exemple est de coût 9. C'est le chemin 1-5-4, car p(4) = 5 et p(5) = 1.



Plus courts chemins partant
du point 1