TP4 - Planification de trajectoire par l'algorithme RRT

Auteurs du CR

Zhi Zhou, zhi.zhou@ensta-paris.fr Simon Queyrut, simon.queyrut@ensta-paris.fr

@zroykhi, @queyrusi

Résumé de cours

Pour planifier une trajectoire il faut éviter des obstacles ; ce peut être fait avec Vector Field Histogram (considérations angulaires), fenêtre dynamique (contraintes) et champs de potentiels. Pour faire face aux imprévus, la planification réactive peut utiliser les Bug. La recherche de chemin stochastique est assurée par la construction d'un arbre avec échantillonage aléatoire pour les branches (plusieurs RRT). Dans le cas d'une carte inconnue, l'exploration pourra être planifiée avec A* ou à partir de grilles d'occupation.

Description du TP

Sous Matlab, nous constatons l'efficacité de plusieurs algorithmes de type RRT en faisant varier un paramètre et nous implémentons OBRRT pour permettre à un RRT de terminer sa course malgré les obstacles dans le nombre d'itérations imparti.

Résultats

Question 1

Nous considérons la carte `bench_june1.mat` . En réglant le nombre maximal d'itérations de 5000 à 10000 pour `rrt` , `rrt_star` et `rrt_star_fn` , nous relevons pour les nombres d'échecs

	5000	6000	7000	8000	9000	10000
`rrt`	9	8	5	3	2	1
`rrt_star`	9	7	5	2	1	1
`rrt_star_fn`	8	7	4	2	1	0

puis pour la longueur moyenne des chemins :

	5000	6000	7000	8000	9000	10000
`rrt`	116.7437	112.6842	114.7131	113.1237	115.51	113.4749
`rrt_star`	108.2845	103.7718	103.8191	103.2836	99.8623	98.9981
`rrt_star_fn`	97.0298	94.5601	93.3796	93.1107	91.7861	90.6379

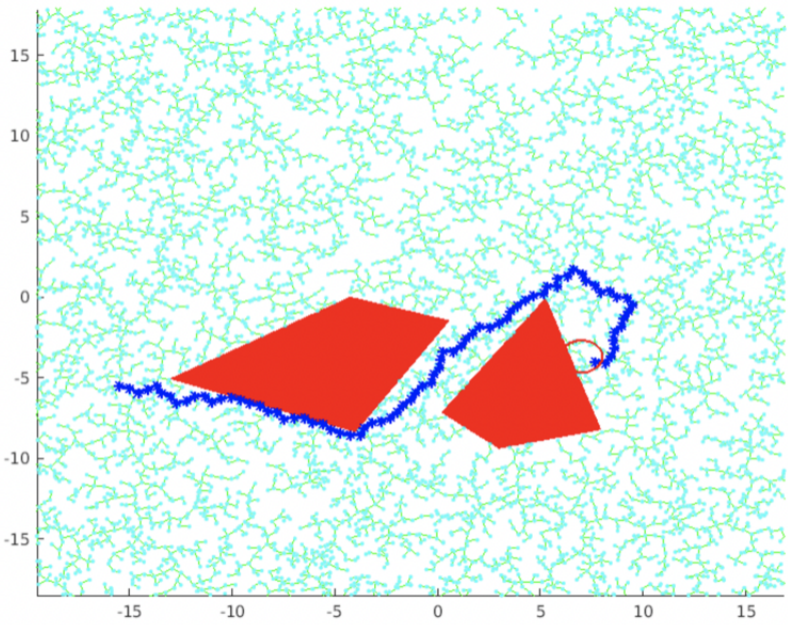
et les temps d'exécution :

	5000	6000	7000	8000	9000	10000
`rrt`	4541	3944.5	5555.4	5983.1429	6688	5892.67
`rrt_star`	4969	5498.66	5951.8	5802.25	5751.89	6210.33
`rrt_star_fn`	4600.5	5460.33	5581.1667	3678.125	5645.3	5317.88

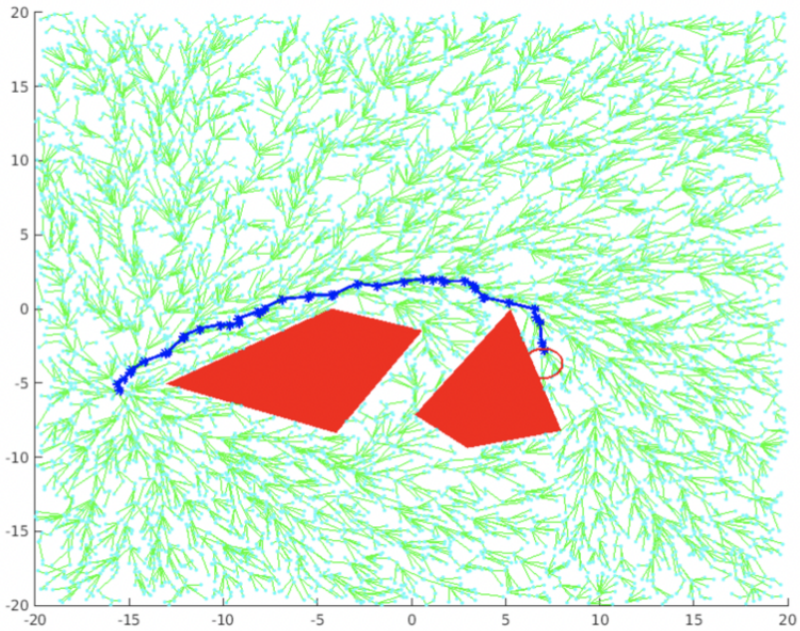
- Nous constatons pour que **la longueur des chemins diminue légèrement selon le nombre d'itérations pour les trois types de RRT** puisque l'arbre, ayant trouvé éventuellement un chemin satisfaisant, continue de s'étendre dans la réserve d'itérations restantes et trouvera probablement un chemin plus optimal avant la fin de la réserve.

un run pour `rrt` :

un run pour `rrt` :



un run pour `rrt_star` :



- **Les temps de calcul augmentent puisque les échecs ne sont pas comptabilisés dans la moyenne**. Comme la recherche de chemins implique un processus stochastique, certains run de la recherche peuvent demander un plus grand nombre d'itérations pour aboutir à un chemin donc moins la limite d'itérations est contraignante, plus on est amenés à comptabiliser des chemin trouvés avec beaucoup de temps.
- `rrt_star_fn` offre une optimalité remarquable tant dans son nombre d'échecs, sa longueur moyenne des chemins trouvés et son temps d'exécution.

Question 2

Nous considérons à présent la carte `RSE12.mat` et lançons la commandee suivante :

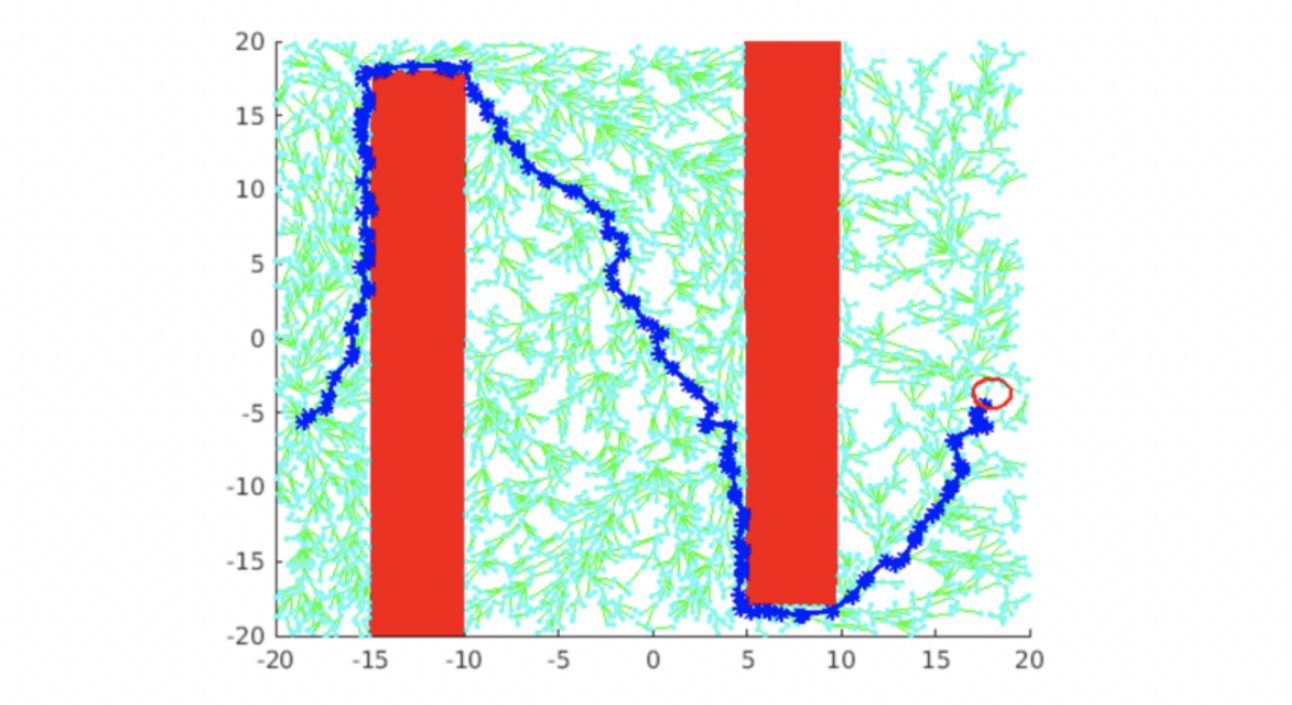
```
benchmark2D('RSE12.mat',[-18.5 -5.5],[18 -3.65],'rrt','FNSimple2D',5000)
```

Nous constatons un échec dans les 10 exécutions. L'algorithme ne trouve pas de chemin du start au goal. Nous modifions `FNSimple2D_Obst.m` afin d'implémenter une variante de l'algorithme OBRRT.

Un jeton aléatoire servira à viser 5% du temps un des coins dees obstacles. Nous choisissons un des "coins" (les obstacles sont plygonaux donc ce sont plutôt des sommets) d'un des deux obstacles de manière aléatoire.

```
x = rand;
if x < 0.05
    obs_ind = randi([1 this.obstacle.num], 1);
    corners = this.obstacle.output{obs_ind};
    col_ind = randi([1 size(corners, 1)], 1);
    position = corners(col_ind,:)';
end
```

Nous obtenons un arbre de recherche à l'allure suivante :



et les résultats sur le benchmark sont un peu meilleurs : seulement 6 échecs. Avec un peu plus de 9%, nous arrivons parfois à 3 échecs de moyenne.

Nous ne sommes pas allés plus loin dans ce TP.