Planification de trajectoire par l'algorithme `rrt`

Jahmal BAPTISTE
Bunthet SAY

Janvier 2020

Contents

1 Introduction

Dans ce TP, nous allons utiliser l'algorithme `rrt` et ses variantes (`rrt_star` et `rrt_star_fn`) pour faire la planification de trajectoire dans 2 cartes avec des obstacles formant des passages de nature différente.

Dans la première question, nous allons analyser la longueur et le temps de calcul moyen du chemin de chaque algorithme pour la première carte pour trouver le meilleur algorithme correspondant.

Dans la deuxième question, nous allons analyser la performance de l'algorithme `rrt` pour une carte avec des passages étroits et nous allons améliorer sa performance en travaillant sur la technique de l'échantillonnage.

Le code de notre travail se trouve dans ce répositoire GitHub.

2 Question 1: Optimalité du chemin

Pour comparer les algorithmes `rrt`, `rrt_star` et `rrt_star_fn`, nous avons borné les itérations maximales entre 1000 et 10000 – il y avait trop d'échecs en-dessous de 1000 itérations et les calculs étaient trop longs au-dessus de 10000 pour un gain trop faible en performances.

Les résultats sont condensés dans les Figures 1, 2 et 3. Nous pouvons y voir que :

- le nombre d'échecs est pratiquement nul pour les trois algorithmes à partir de **5000 itérations**;
- l'algorithme `rrt` n'a pas l'air de gagner en qualité avec le nombre d'itérations, contrairement aux autres algorithmes (**24%** d'amélioration pour l'algorithme `rrt_star` et **15%** pour `rrt_star_fn`, entre 1000 et 10000 itérations);
- les temps de calculs augmentent beaucoup plus vite pour les algorithmes `rrt_star` et `rrt_star_fn` que pour l'algorithme `rrt`, qui reste bien plus rapide qu'eux.

A la lumière de ces graphiques il nous semble que l'algorithme présentant un meilleur compromis entre temps de calcul et l'optimalité du chemin est l'algorithme `rrt_star_fn` (à condition qu'on lui attribue 5000 itérations maximales pour ne pas trop s'exposer au risque d'échec).

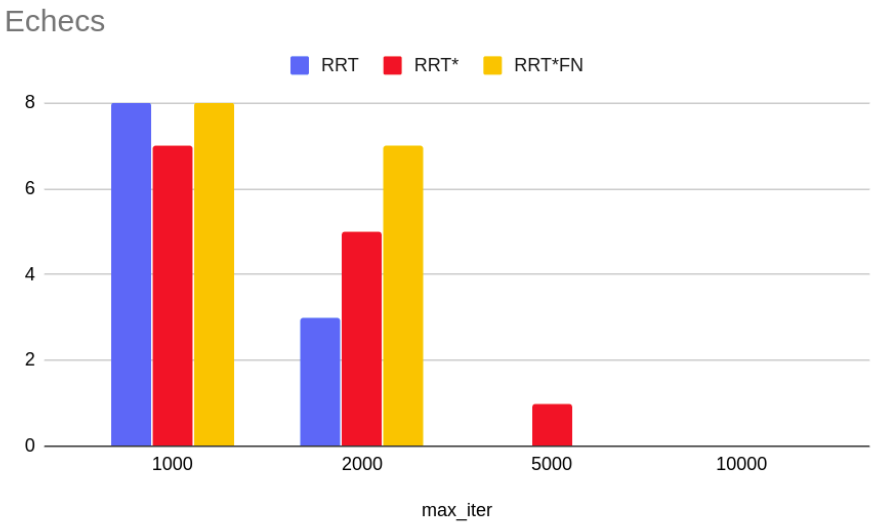

Figure 1: Nombre d'échecs sur 10 essais.

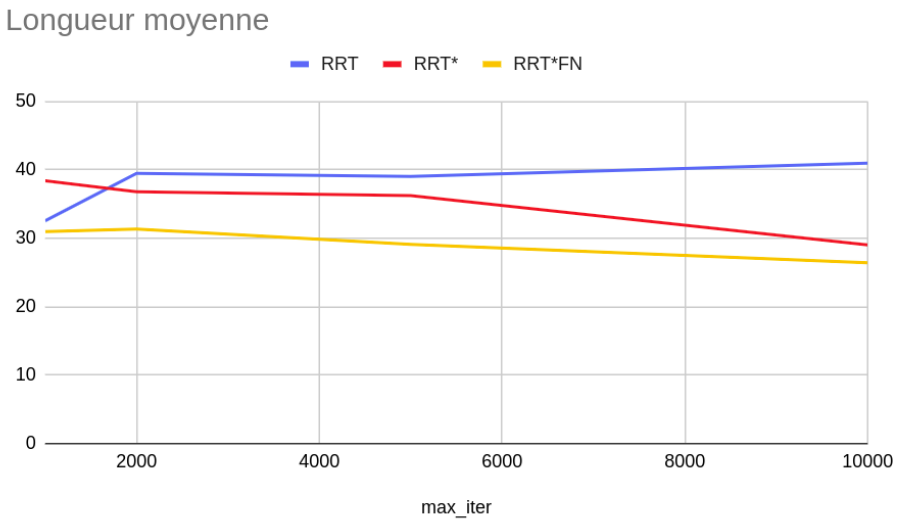

Figure 2: Longueurs moyennes du chemin sur 10 essais.

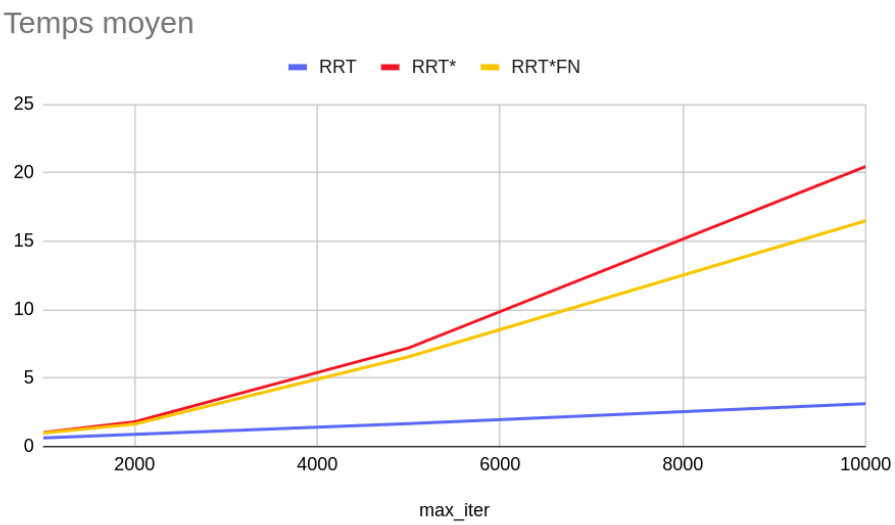

Figure 3: Temps moyens

algorithme	RRT			RRT*			RRT*FN		
max_iter	Echecs	Longueur moy	Temps moy	Echecs	Longueur moy	Temps moy	Echecs	Longueur moy	Temps moy
1000	8	32,56	0,637	7	38,42	1,03	8	30,97	0,98
2000	3	39,49	0,887	5	36,8	1,81	7	31,37	1,64
5000	0	39,06	1,684	1	36,25	7,21	0	29,11	6,57
10000	0	40,98	3,13	0	29,05	20,46	0	26,41	16,5

Figure 4: Données de 10 essais.

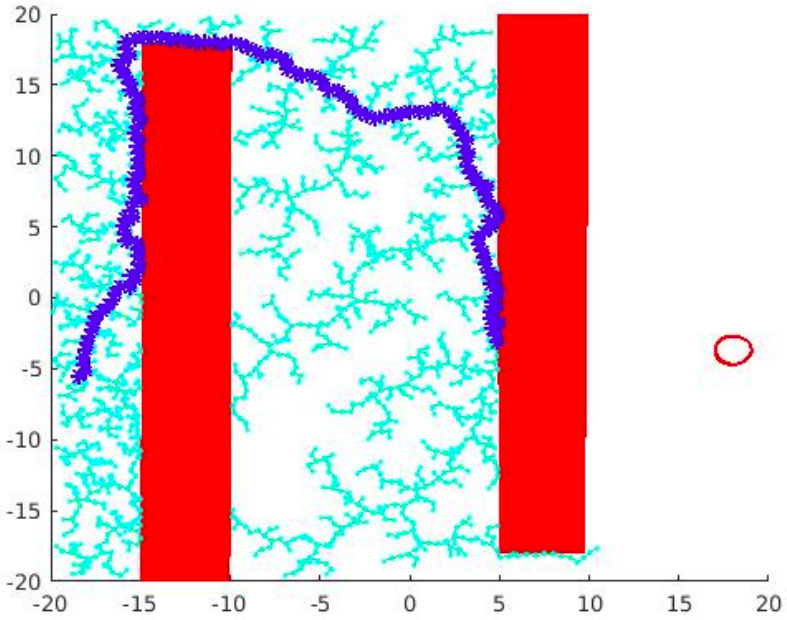

Figure 5: Résultat type de RRT sur une carte avec des passages étroits

3 Question 2: Planification pour des passage étroit

En utilisant l'algorithme `rrt` simple sur une carte présentant des passages étroits nous remarquons que le chemin menant à l'objectif est bien plus difficile à trouver (voir FIGURE 5) (nous n'avons réussi à trouver de solution qu'une poignée de fois sur les dizaines d'essais, donnant une probabilité de réussite ne dépassant pas 5%).

Notons que sur la carte précédente l'algorithme `rrt` trouvait (quasi-)systématiquement une solution avec le même nombre d'itérations.

Pour remédier à ce problème nous avons utilisé un algorithme de sélection des points objectifs inspiré de l'`OBRRT` : 5% des points objectifs sont tirés dans un voisinage des coins des obstacles. Nous avons rajouté à ce tirage la particularité de ne pas considérer les points qui sont sur le bord de la carte. En effet, les point sur les bords ne peuvent que faire perdre du temps au robot car la seule issue (une fois arrivé au niveau de ces points) est la voie d'entrée qui y a mené...

Ce nouvel algorithme performe beaucoup mieux avec une proportion de **19 réussites sur 50 essais**.

Cette modification a permis d'augmenter la proportion de réussite dans la recherche de trajectoire.

4 Conclusion

Nous avons comparé les variantes de l'algorithme `rrt` sur une carte relativement simple pour voir lequel offrait le meilleur compromis entre efficacité et temps de calcul.

Nous avons ensuite considéré un cas pathologique qui mettait à mal l'algorithme `rrt` (et très certainement ses variantes aussi) pour nous rendre compte de l'importance des connaissances a priori sur l'environnement dans lequel le robot évolue. Dans le cas considéré une simple disjonction de cas probabiliste suffisait à améliorer les performances mais il possible de tomber dans des cas plus difficiles à gérer encore...

Nous retiendrons que l'algorithme `rrt` permet de trouver des trajectoires dans plusieurs cas relativement simples à condition de lui donner la puissance de calcul et les heuristiques appropriées. L'amélioration de ses résultats sont disponibles à condition de vouloir payer avec un temps de calcul supplémentaire (d'un ordre de grandeur pour des résultats meilleurs d'une dizaine de pourcent).