Randomized Kinodynamic Planning

École Normale Supérieure Paris-Saclay

Olivier Lévêque & Aboubacar Tuo

24 mars 2018

Résumé

Ce rapport a pour étude l'article **Randomized Kinodynamic Planning** écrit par Steven M. LaValle et James J. Kuffner, Jr. en 1999. Un résumé de l'article et des simulations de l'algorithme RRT vous sont proposés.

`http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01b.pdf`

1 Introduction

Cet article [1] présente la première approche pour la planification probabiliste de trajectoires, connue aussi sous le nom de « kinodynamic planning ». Ce terme, introduit par B. Donald, P. Xavier, J. Canny et J. Rief en 1993 dans une de leurs publications **Kinodynamic motion planning** dans le *Journal of the ACM*, décrit une famille de problèmes fondamentaux cherchant à construire une trajectoire en boucle ouverte satisfaisant à la fois les contraintes imposées par des obstacles environnants mais aussi des contraintes différentielles locales imposées par la dynamique du système étudié.

Il est intéressant de noter que la planification de trajectoire peut être utilisée dans tous les domaines faisant appel à des modèles théoriques pour le contrôle de systèmes dynamiques. Ces domaines sont très variés, allant des préhenseurs en robotique aux modèles d'économétrie. De nombreux secteurs industriels font appel à la planification de trajectoire comme par exemple l'industrie cinématographique pour des effets visuels avancés (mouvements d'objets et de personnes obéissant aux lois physiques).

L'approche classique de la planification repose sur le découpage du problème général en trois parties.

1. Résoudre le problème de planification de base ;
2. Trouver une trajectoire et une commande satisfaisant la dynamique du système ;
3. Suivre cette trajectoire.

La majorité des algorithmes permettant de résoudre le problème de planification repose uniquement sur la cinématique du système et ignore complètement sa dynamique. C'est pourquoi le problème kinodynamique, qui prend compte de cette dynamique, est considéré comme une généralisation du problème de la planification holonome et non-holonome dans l'espace des variables d'état.

1. `http://msl.cs.uiuc.edu/~lavalle/papers/LavKuf01b.pdf`

2 Formulation du problème

Le problème de planification kinodynamique est formulé comme un problème de planification dans l'espace des variables d'état avec des contraintes différentielles d'ordre 1. Posons $\mathcal{C}$ l'espace des configurations pour un système rigide ou articulé pouvant se déplacer en 2D ou 3D, tel que chaque configuration $q \in \mathcal{C}$ représente une transformation appliquée au modèle géométrique associé au système. Posons aussi $\mathcal{X}$ l'espace d'état, tel que $x \in \mathcal{X}$ s'écrive $x = (q, \dot{q})$.

Sous des conditions appropriées [2], les contraintes différentielles peut-être exprimées sous la forme

$$\dot{x} = f(x, u) \text{ avec } u \in \mathcal{U} \tag{1}$$

où $\mathcal{U}$ représente l'ensemble des commandes admissibles.

Si nous considérons un environnement contenant des obstacles statiques. Il existe des différences notables entre trouver un chemin sans collision dans $\mathcal{C}$ ou dans $\mathcal{X}$. Dans $\mathcal{C}$, nous pouvons caractériser l'ensemble des configurations $\mathcal{C}_{obst}$ où le robot est en collision. La planification de trajectoire implique de trouver un chemin continu dans l'ensemble $\mathcal{C}_{free}$ tel que $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obst}$. Dans $\mathcal{X}$, nous pouvons définir $\mathcal{X}_{obst}$ tel que $x \in \mathcal{X}_{obst}$ si et seulement si $q_{obst} \in \mathcal{C}$ pour $x = (q, \dot{q})$, cependant en considérant la dynamique du système, une autre région interdite existe : *la région des collisions inévitables* que nous noterons $\mathcal{X}_{ric}$. Cette région correspond aux états où il n'existe aucune commande pour éviter la collision. Nous pouvons noter que $\mathcal{X}_{obst} \subseteq \mathcal{X}_{ric}$.
L'ensemble $\mathcal{X}_{ric}$ traduit bien la difficulté du problème de planification kinodynamique car si une trajectoire cinématique sans collision existe à un problème de planification, une trajectoire kinodynamique n'existe quant à elle pas forcément. Comme dans l'article, nous définirons $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obst}$ bien que la définition $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{ric}$ est aussi une alternative possible.

Avec cette définition de $\mathcal{X}_{free}$, résoudre le problème de planification kinodynamique revient à trouver une trajectoire d'un état initial, $x_{init} \in \mathcal{X}_{free}$, à un état final, $x_{goal} \in \mathcal{X}_{free}$ telle que cette trajectoire soit un chemin continu paramétré en temps, $\tau : [0, T] \longrightarrow \mathcal{X}_{free}$. L'état final peut aussi être considéré comme une région de l'espace d'état, tel que $\mathcal{X}_{goal} \subset \mathcal{X}_{free}$.

La trajectoire, $x(t)$ pour $t \in [0, T]$, est déterminée en intégrant l'équation (1). Il peut être approprié de sélectionner la trajectoire qui optimise un certain critère tel que la distance nécessaire pour atteindre x_{goal}.

Une approximation numérique de l'équation (1) pour calculer l'état suivant $x(t + \delta t)$ à partir de l'état actuel $x(t)$ et de la commande $u(t')$ avec $t' \in [t, t + \delta t]$ est, par exemple, de choisir une commande u constante sur l'intervalle d'intégration $[t, t + \delta t]$ et d'utiliser la méthode d'intégration de Runge-Kutta d'ordre 4.

$$x(t + \delta t) \approx x(t) + \frac{\delta t}{6}(f(x(t), u) + 2x' + 2x'' + x''') \text{ avec } \begin{cases} x' & = f(x(t) + \frac{\delta t}{2}x, u) \\ x'' & = f(x(t) + \frac{\delta t}{2}x', u) \\ x''' & = f(x(t) + \frac{\delta t}{2}x'', u) \end{cases}$$

2. Conditions du théorème des fonctions implicites

3 Rapidly-Exploring Random Trees (RRT)

Les auteurs de l'article proposent pour résoudre le problème de plannification kinodynamique un algorithme baptisé « Rapidly-Exploring Random Trees » (RRT). Il s'agit d'une méthode simple permettant d'explorer de manière rapide, aléatoire et uniforme l'espace d'état.

```
BUILD_RRT(x_init)
1  T.init(x_init);
2  for k = 1 to K do
3      x_rand ← RANDOM_STATE();
4      EXTEND(T, x_rand);
5  Return T
```

```
EXTEND(T, x)
1  x_near ← NEAREST_NEIGHBOR(x, T);
2  if NEW_STATE(x, x_near, x_new, u_new) then
3      T.add_vertex(x_new);
4      T.add_edge(x_near, x_new, u_new);
5      if x_new = x then
6          Return Reached;
7      else
8          Return Advanced;
9  Return Trapped;
```

FIGURE 1 – Structure de l'algorithme RRT

Pour cette exploration, l'algorithme RRT construit un arbre de la manière suivante. Il commence par définir l'état initial x_{init} comme un sommet de l'arbre puis répétitivement tire aléatoirement un échantillon x dans l'espace d'état, cherche selon une métrique ρ le sommet voisin le plus proche, x_k^{near} (fonction EXTEND – figure 2) et trouve un contrôle $u_k \in \mathcal{U}$ permettant, à partir de x_k^{near}, de se rapprocher de x (fonction NEW_STATE). Alors $x_{k+1}^{new} = f(x_k^{near}, u_k)$ est ajouté à l'arbre s'il respecte bien les contraintes globales [3]. La structure de l'algorithme RRT est rappelée figure 1.

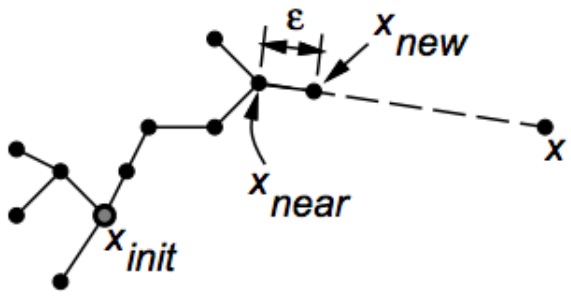

FIGURE 2 – L'opération EXTEND

La commande $u_k \in \mathcal{U}$ est appliquée durant un temps Δt qui peut-être choisi fixe ou aléatoire à chaque itération dans un intervalle $]0, \Delta t_{max}]$. En général, Δt doit être bien plus grand que le temps d'intégration δt nécessaire pour calculer (1). La commande u_k peut-être choisi aléatoirement ou en essayant toutes les possibilités et en gardant celle qui rapproche

3. L'état x_{k+1}^{new} doit appartenir à $\mathcal{X}_{free}$

le plus possible le nouveau sommet x_{k+1}^{new} de l'échantillon x. Si $\mathcal{U}$ est infini, l'ensemble peut être discrétisé. L'échantillon x est considéré comme atteint si $||x_{k+1}^{new} - x|| < \epsilon$ avec $\epsilon > 0$.

Pour construire une trajectoire à partir de l'exploration précédente, nous devons construire deux arbres RRT l'un partant de l'état initial, x_{init} et l'autre de l'état final, x_{goal}. Cette algorithme, baptisé « RRT bidirectionnel », cherche les sommets communs aux deux arbres. Deux sommets, x et x', sont considérés en commun si $\rho(x, x') < \epsilon$ avec $\epsilon > 0$. L'algorithme peut s'arrêter au premier couple de sommets communs trouvé ou peut continuer pour choisir la meilleure trajectoire parmis la collection de sommets communs trouvés. La structure de l'algorithme RRT bidirectionnel est rappelée figure 3.

```
RRT_BIDIRECTIONAL(x_init, x_goal)
1  T_a.init(x_init); T_b.init(x_goal);
2  for k = 1 to K do
3      x_rand ← RANDOM_STATE();
4      if not (EXTEND(T_a, x_rand) = Trapped) then
5          if (EXTEND(T_b, x_new) = Reached) then
6              Return PATH(T_a, T_b);
7      SWAP(T_a, T_b);
8  Return Failure
```

FIGURE 3 – Structure de l'algorithme RRT bidirectionnel

4 Implémentations & Simulations

Plusieurs simulations de planification kinodynamique ont été menées par nos soins sous MATLAB, et vous sont présentées dans cette section. La figure 4 illustre l'exploration d'un espace d'état à l'aide de l'algorithme RRT détaillé précédemment et la figure 5 illustre le fonctionnement de l'algorithme RRT bidirectonnel pour la recherche d'une trajectoire kinodynamique entre un état initial $x_{init} \in \mathcal{X}_{free}$ et un état final $x_{goal} \in \mathcal{X}_{free}$.

Dynamique du modèle Le système rigide considéré ici est une voiture se déplaçant en 2D, ne pouvant qu'avancer (pas de marche arrière) et tourner. Sa dynamique est fournie par les équations suivantes.

$$x = \begin{pmatrix} p_1 \\ p_2 \\ \theta \end{pmatrix} \in \mathcal{X}$$

$$\dot{x} = f(x(t), u(t)) = \begin{pmatrix} \dot{p_1} \\ \dot{p_2} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} u_1 \cos(u_2) \\ u_1 \sin(u_2) \\ u_2 \end{pmatrix} \in \mathcal{C}$$

avec

$$\mathcal{X} = [0, 100] \times [0, 100] \times [-\pi, \pi[$$

Métrique Pour pouvoir calculer une distance entre différents états, nous nous munissons de la métrique suivante correspondant à la norme euclidienne.

$$\rho(x_1, x_2) = \sqrt{\alpha(p_{1,1} - p_{1,2})^2 + \beta(p_{2,1} - p_{2,2})^2 + \gamma(\theta_1 - \theta_2)^2}$$

où α, β et γ sont respectivement les poids des composantes $p_{1,i}$ (position horizontal), $p_{2,i}$ (position verticale) et θ_i (position angulaire) des états x_i. Nous avons choisi ici $\alpha = \beta = 1$ et $\gamma = 0$ pour calculer la distance séparant deux positions spatiales de la voiture.

Commande appliquée Pour calculer la commande $u = (u_1, u_2) \in \mathcal{U}$ connaissant le sommet voisin x^{near} et l'échantillon x que nous cherchons à atteindre, nous calculons la commande u comme recommandé dans la section 5.5 du livre de Steven M. LaValle [4].

$$u_1 = \frac{\rho(x^{near}, x)}{M} \text{ avec } M = 100$$
$$u_2 = \arctan\left(\frac{p_{2,x} - p_{2,x_k^{near}}}{p_{1,x} - p_{1,x_k^{near}}}\right)$$

M est ici un facteur de normalisation choisi pour que u_1 appartient à l'intervalle $[0, 1]$. Nous constatons qu'avec cette approche, au fur et à mesure que l'échantillon x tiré est proche d'un des sommets de l'arbre, $\rho(x^{near}, x)$ devient petit. Ainsi u_1 impose de petits déplacements à la voiture permettant une meilleur exploration de l'espace d'état $\mathcal{X}$.

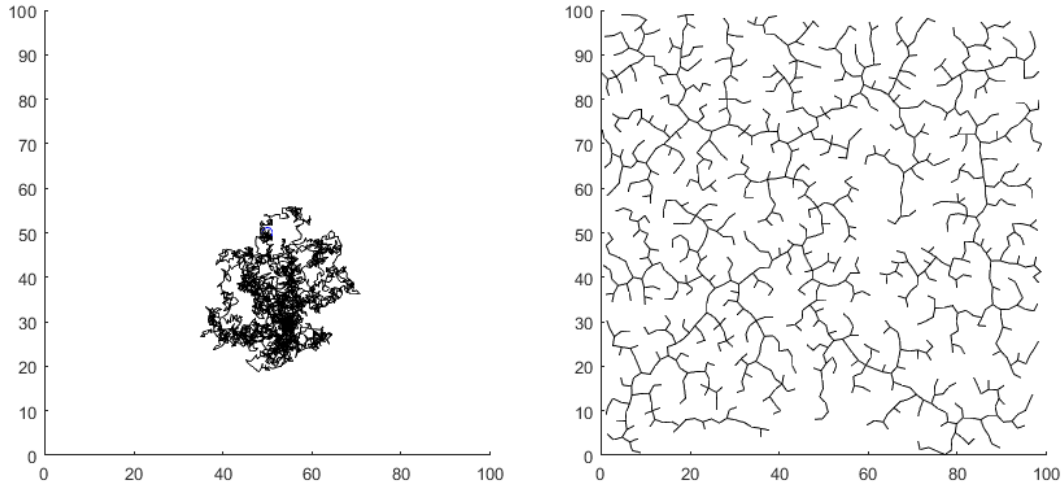

FIGURE 4 – Comparaison entre un algorithme d'exploration naïf (à gauche) et l'algorithme d'exploration RRT (à droite) – L'algorithme d'exploration naïf consiste à explorer l'espace d'état avec un pas fixe et une direction aléatoire recalculée à chaque itération. Nous constatons qu'avec le même nombre d'itérations (3000) et le même état initial (50,50), l'algorithme RRT explore plus rapidement et de manière uniforme l'espace d'état.

4. `http://planning.cs.uiuc.edu/`

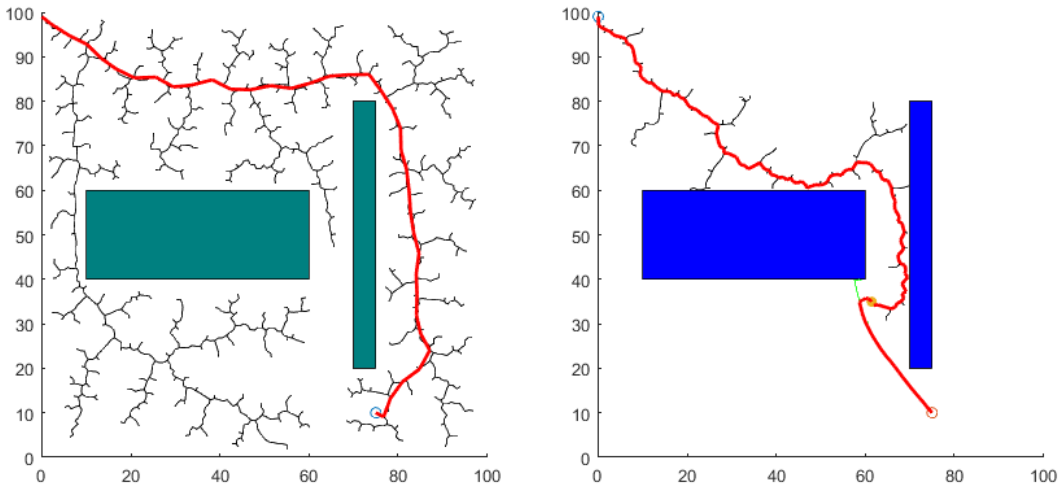

FIGURE 5 – Comparaison entre une trajectoire kinodynamique déterminée à l'aide de l'algorithme RRT (à gauche) et RTT bidirectionnel (à droite) partant du même état initial (1,99) et arrivant au même état final (75,10) – L'algorithme RTT bidirectionnel trouve une trajectoire plus rapidement et avec une exploration moins approfondie que l'algorithme RRT.

Lors de l'exploration figure 5 (à gauche), nous associons à chaque sommet un coût correspondant à la distance parcourue sur l'arbre depuis le sommet initial. Pour calculer la trajectoire de coût minimal, nous définissons pour chaque sommet un « parent » correspondant au sommet voisin de coût minimal. Nous pouvons ainsi construire la trajectoire optimale[5] en partant de l'état final et en remontant de parent en parent jusqu'à l'état initial.

5 Conclusion

Ce rapport, résumé de l'article **Randomized Kinodynamic Planning** écrit par Steven M. LaValle et James J. Kuffner, aura permis de rappeler l'intérêt et la formulation du problème de planification kinodynamique en décrivant l'algorithme « Rapidly-Exploring Random Trees » (RRT) permettant une exploration rapide, aléatoire et uniforme de l'espace d'état et l'algorithme RRT bidirectionnel permettant de trouver une trajectoire dans l'espace d'état issue de cette exploration.

5. Trajectoire de coût minimal