Estimation en ligne des métriques de diffusion en planification probabiliste de mouvement

Sébastien Dalibard
sous la direction de Jean-Paul Laumond
au sein de l'équipe Gepetto du LAAS

Stage réalisé du 1er mars au 15 août 2007

Le contexte général

Le contexte général de ce travail est celui des algorithmes probabilistes en planification de mouvement.

La planification de mouvement est un domaine de recherche très actif en robotique. Il s'agit de transformer un ordre donné à haut niveau à un robot en une série de déplacements élémentaires

Depuis quelques années, une part importante de ces recherches concerne les algorithmes probabilistes, et en particulier des méthodes de diffusion, qui permettent de résoudre des problèmes trop complexes pour les algorithmes déterministes traditionnels.

Le problème étudié

La question abordée concerne les limites de ces algorithmes probabilistes. Dans certains problèmes à géométrie compliquée, soit par une dimension élevée, soit par la présence de passages étroits, les méthodes de diffusion peuvent être fortement ralenties. Souvent pourtant, un choix pertinent de dimensions à favoriser permet de résoudre le problème rapidement.

Le sujet de ce travail est de décrire en ligne la forme des espaces dans lesquels évoluent des arbres de diffusion aléatoires. Répondre à cette question permettra d'améliorer les performances des algorithmes de planification de mouvement par diffusion dans les espaces contraints, par exemple dans les problèmes de désassemblage mécanique, où ils sont largement utilisés.

Les questions de description de passages étroits ou de réduction de dimension ont déjà été étudiées en planification de mouvement, sans être totalement résolues. L'originalité de cette contribution tient en la méthode proposée, qui utilise des outils statistiques connus, mais pas encore utilisés en planification de mouvement.

La contribution proposée

La solution proposée consiste à décrire les espaces dans lesquels évoluent des arbres de diffusion par une analyse statistique des nœuds de ces arbres. L'outil utilisé, l'analyse en composantes principales, est connu depuis longtemps, et utilisé dans de nombreux problèmes de réduction de dimension. Une fois l'espace de travail décrit, on biaise le processus de diffusion pour favoriser les directions obtenues par la description. On accélère ainsi notablement le remplissage de l'espace par les arbres de diffusion.

La démarche fut d'étudier la litttérature de planification de mouvement consacrée aux problèmes des passages étroits, puis de voir comment des problèmes similaires (réduction de la dimension, description concise de données complexes) étaient traités dans des domaines autres que la planification de mouvement. C'est ainsi qu'est venue l'idée d'utiliser l'analyse en composantes principales.

Les arguments en faveur de sa validité

Des travaux théoriques permettent d'étudier la convergence de la méthode proposée. Toutefois, la meilleure justification à notre solution sont les expériences. On présente à la fin de ce rapport des comparaisons entre les performances d'algorithmes connus de diffusion aléatoire et notre méthode, sur différents exemples, dont un industriel. Les résultats de ces expériences valident la solution proposée.

La méthode proposée ne modifie pas la structure des algorithmes de diffusion utilisés, et permet d'hériter de leurs propriétés de complétude. La robustesse de notre solution doit donc être analysée en des termes d'efficacité. On a essayé de tester notre algorithme sur des problèmes variés de planification de mouvement, et il a toujours montré de bons résultats.

Le bilan et les perspectives

Un des points positifs de la méthode proposée est qu'elle ne réimplémente qu'une partie des algorithmes de diffusion aléatoire. Ce n'est pas un nouvel algorithme de diffusion à proprement parler. On peut donc adapter n'importe quelle méthode de diffusion existante pour utiliser notre solution, en conservant les « bonnes » propriétés de l'ancien algorithme (complétude, rapidité). Cette généricité permet d'envisager un champ d'application aussi large que celui des algorithmes de diffusion aléatoire.

La contribution de ce travail au domaine de la planification de mouvement est donc, d'une part l'idée d'utiliser les outils statistiques de réduction de dimension pour contrôler les algorithmes de diffusion aléatoires, et d'autre part, une première étude sur les résultats de cette utilisation. Cette étude comprend un volet théorique sur l'analyse en composantes principales, et un sur les résultats pratiques observés lors de nos expériences.

La méthode proposée est utilisable en l'état, et la prochaine étape est de l'essayer sur un robot humanoïde, ce qui était un des buts du stage. Ces systèmes ont en effet de nombreux degrés de liberté, et sont donc particulièrement concernés par les problématiques de réduction de dimension.

À plus long terme, il peut être envisagé de poursuivre dans la voie du contrôle des algorithmes de diffusion par des méthodes statistiques en essayant des outils plus évolués que l'analyse en composantes principales. Il existe en effet un champ de recherche très actif sur la réduction de dimension non linéaire, et les domaines où l'analyse en composantes principales est traditionnelement utilisée semblent profiter de chaque nouvelle méthode proposée. Les raisons pour lesquelles on s'est dans un premier temps restreint à l'analyse en composantes principales sont détaillées dans le rapport.

Table des matières

Liste des algorithmes

Table des figures

Introduction

L'algorithmique de la planification de mouvement est un domaine de recherche très actif en robotique. Depuis quelques années, de nombreuses études portent sur les algorithmes basés sur des méthodes de recherche par arbre de diffusion aléatoires. Ces méthodes sont très efficaces dans les probèmes très contraints, comme les probèmes d'assemblage mécaniques.

Ces algorithmes présentent toutefois des limites. Lorsque la forme des espaces de travail est complexe, et que ces espaces présentent des « passages étroits », la progression des arbres de diffusion est fortement ralentie, alors même qu'un utilisateur humain serait capable de décrire la forme du passage et les directions à privilégier pour la diffusion.

L'objet de ce travail est d'automatiser cette description. Il s'agit d'implémenter une méthode d'estimation en ligne de la forme locale de l'espace de travail. Cette méthode sera ensuite utilisée pour favoriser certaines directions lors de la diffusion.

Le premier chapitre présente quelques généralités sur la planification de mouvement probabiliste, et décrit un algorithme largement utilisé : les *Rapidly exploring Random Trees*. Le deuxième chapitre fait le point sur les méthodes rencontrées dans la littérature qui s'attaquent aux problèmes des passages étroits et de la réduction de dimension en planification de mouvement. Le troisième chapitre présente les solutions envisagées et implémentées lors de ce stage pour répondre à ces problèmes, et le quatrième et dernier chapitre étudie en détail la solution retenue, qui utilise l'outil statistique d'analyse en composantes principales.

Les algorithmes développés ont été implémentés dans le cadre du projet HPP (*Humanoid Path Planner*) au LAAS. On a bénificié de la plateforme logicielle KineoWorks, qui comprend le logiciel KineoPathPlanner et une API pour développer des algorithmes de planification de mouvement. Toutes les captures d'écran et simulations de ce rapport en sont extraites.

Remerciements

Je tiens à remercier mon directeur de stage Jean-Paul Laumond, ainsi que toute l'équipe de recherche Gepetto, pour leur disponibilité durant tout ce stage.

Chapitre 1

Planification de mouvement par des méthodes de diffusion

Ce chapitre présente la cadre global dans lequel s'inscrit le travail réalisé. On présente dans un premier temps la planification de mouvement, puis le formalisme qu'on lui attache couramment. Dans une troisième partie, on décrit un algorithme particulier de planification de mouvement, qui repose sur des méthodes de diffusion.

1.1 La planification de mouvement en robotique

La robotique a pour but de concevoir des systèmes autonomes, capables de percevoir, raisonner, se déplacer et agir sur le monde qui les entoure. Un besoin fondamental pour un robot dans un environnement est sa capacité à traduire des ordres donnés par un humain et exprimés à un haut niveau, en une série de de mouvements et de tâches de bas niveau, qui décrivent de façon effective comment il va se déplacer. La planification de mouvement, ou de trajectoire, s'efforce de répondre à ce besoin.

Un probème souvent présenté comme la tâche canonique que doit savoir résoudre un planificateur de mouvement est celui du déménageur de piano. On imagine avoir une description précise de la géométrie d'une pièce dans laquelle on veut déplacer un piano. Un algorithme de planification de mouvement doit pouvoir spécifier une suite de mouvemements simples par lesquels on bouge le piano d'un endroit à un autre sans toucher aucun autre objet.

1.2 L'espace de configurations

Une notion importante avant d'aller plus loin est celle de degré de liberté. La position d'un robot dans son espace de travail, qu'on appelera plus volontiers sa configuration, peut être décrite par un vecteur de paramètres. Ces paramètres sont les degrés de liberté du robot. Un robot mobile sur roues par exemple, peut être situé dans l'espace au moyen de sa position dans $\mathbb{R}^2$ et de son angle par rapport à une direction de référence. La donnée de ces trois valeurs (x, y, θ) permet de déterminer comment le robot interagit avec son environnement.

L'espace de configuration est la formalisation de cette idée. Il a été introduit en 1980 par Tomas Lozano-Perez dans [13]. On appelle espace de configuration, et on note *CS* (Configuration Space), l'ensemble des configurations du robot. C'est une variété de dimension le nombre de degrés de liberté du robot. Les obstacles dans le monde réel ($\mathbb{R}^2$ ou $\mathbb{R}^3$) induisent

des obstacles dans CS. L'ensemble des configurations telles que le robot, dans cette configuration, n'est pas en collision avec son environnement est un sous-ensemble de CS qu'on appelle l'espace libre et qu'on note CS_{free}.

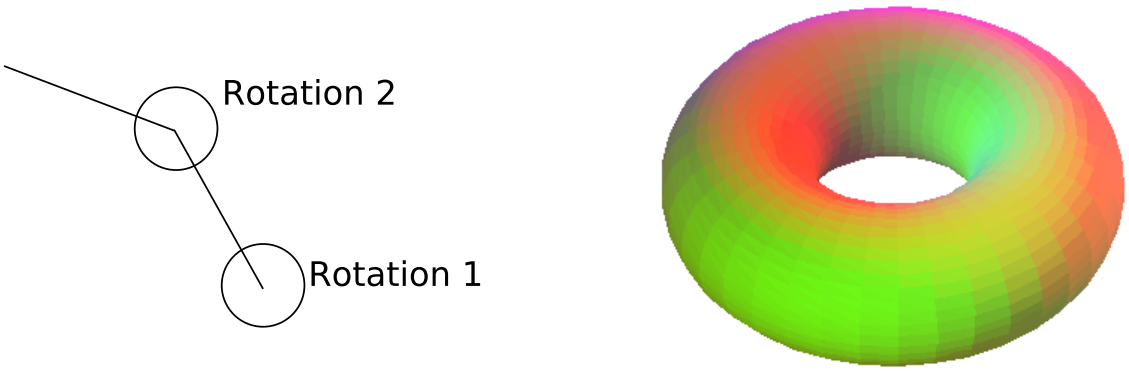

FIG. 1.1 – Bras à deux degrés de liberté, et son espace de configurations

Muni de ce formalisme, un problème de planification de mouvement se résume à ceci : étant donné un robot, son environnement, et deux points particuliers $q_{initial}$ et q_{final} de son espace libre, existe-t-il un chemin, dans CS_{free}, qui relie $q_{initial}$ et q_{final} ? La force de cette formalisation est de ne pas planifier le mouvement d'un objet rigide dans le monde réel ($\mathbb{R}^2$ ou $\mathbb{R}^3$), mais celui d'un point dans un espace de dimension supérieure.

1.3 Rapidly exploring Random Trees (RRT)

L'algorithme présenté ici appartient à la famille des algorithmes aléatoires de planification de mouvement, basés sur des méthodes d'échantillonage. L'idée derrière ces algorithmes est d'éviter la construction explicite de l'espace de configurations, ainsi que de l'espace libre. À la place, l'algorithme a accès à des « boîtes noires » qui lui fournissent les informations nécessaires sur l'environnement. On dispose ainsi d'une fonction qui prend en entrée une configuration et qui dit si elle est en collision, ou encore d'une fonction qui évalue la taille d'un voisinage inclu dans l'espace libre autour d'une configuration donnée.

La force des algorithmes aléatoires en planification de mouvement repose sur ce non-calcul de la structure de CS et CS_{free}. Ils sont en effet capables de traiter des problèmes dans des espaces hautement dimensionnés, où l'environnement est décrit par des structures de données compliquées, pouvant contenir des millions de primitives géométriques. La complexité de ces problèmes empêche toute approche déterministe qui représenterait explicitement CS.

Du fait de la partie aléatoire de ces algorithmes, il convient de définir des notions de complétudes différentes de celles utilisées traditionnellement. La deuxième partie de cette section présente ainsi quelques propriétés de l'algorithme étudié.

Cet algorithme a été présenté pour la première fois dans [11]. La description qu'on en fait ici reprend [12] et [9].

1.3.1 Algorithme

Un *Rapidly exploring Random Tree* (RRT) est un arbre de recherche aléatoires qu'on fait grossir dans l'espace libre. À la limite, cet arbre recouvre densément CS_{free}. C'est un algorithme très utilisé en planification de mouvement car il présente de bons résultats dans la pratique sans qu'il y ait besoin de régler aucun paramètre.

L'arbre $\mathcal{T}$ est enraciné en une configuration particulière q_0. À chaque étape, on tire une configuration aléatoire q_{rand} dans CS, on cherche le nœud de l'arbre q_{near} le plus proche de q_{rand}, et on étend l'arbre aussi loin que possible de q_{near} dans la direction de q_{rand}, en restant dans CS_{free}.

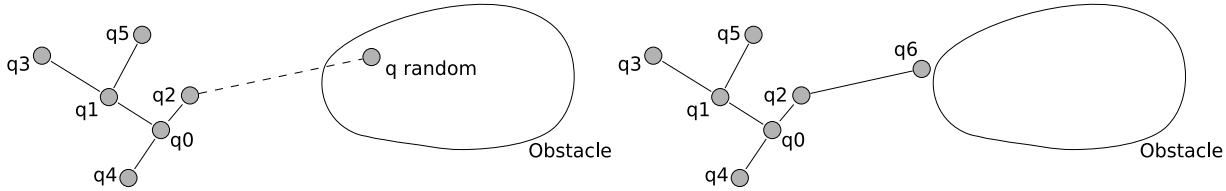

FIG. 1.2 – Une étape d'expansion de l'arbre

Algorithme 1 RRT(q_0)

$\mathcal{T}$.Init(q_0)
for $i = 1$ to K **do**
  $q_{rand} \leftarrow$ Rand(CS)
  $q_{near} \leftarrow$ Nearest($q_{rand}, \mathcal{T}$)
  $q_{new} \leftarrow$ Extend(q_{near}, q_{rand})
  $\mathcal{T}$.AddVertex(q_{new})
  $\mathcal{T}$.AddEdge(q_{near}, q_{new})
end for

Cette étape d'expansion est le cœur de l'algorithme RRT. Une fois qu'on sait construire incrémentalement un arbre, il existe de nombreuses options quant-à comment l'utiliser en planification de mouvement. La première idée est de faire grossir un arbre dont la racine est $q_{initial}$ en vérifiant à chaque étape si on peut connecter q_{final} au nouveau nœud de l'arbre. Une approche souvent plus efficace est de faire grossir conjointement deux arbres, enracinés l'un en $q_{initial}$, l'autre en q_{final}, en essayant régulièrement de les connecter.

Il existe de nombreuses variantes à l'algorithme RRT, qu'on ne présentera pas ici, car elles sont souvent des adaptations de l'algorithme à certains types de problèmes de planification de mouvement. Notre but est de rester le plus générique possible dans cette présentation.

1.3.2 Propriétés

Les propriétés de terminaison des RRT sont héritées des propriétés de la suite utilisée pour générer les configurations aléatoires. Si cette suite de configurations est dense dans *C-Space*, alors à la limite, l'arbre couvre densément la composante connexe de CS_{free} à laquelle il appartient.

Cette propriété permet d'énoncer la propriété de terminaison suivante : s'il existe un chemin dans CS_{free} qui relie $q_{initial}$ et q_{final}, alors l'algorithme RRT va le trouver en un temps fini

avec probabilité 1.

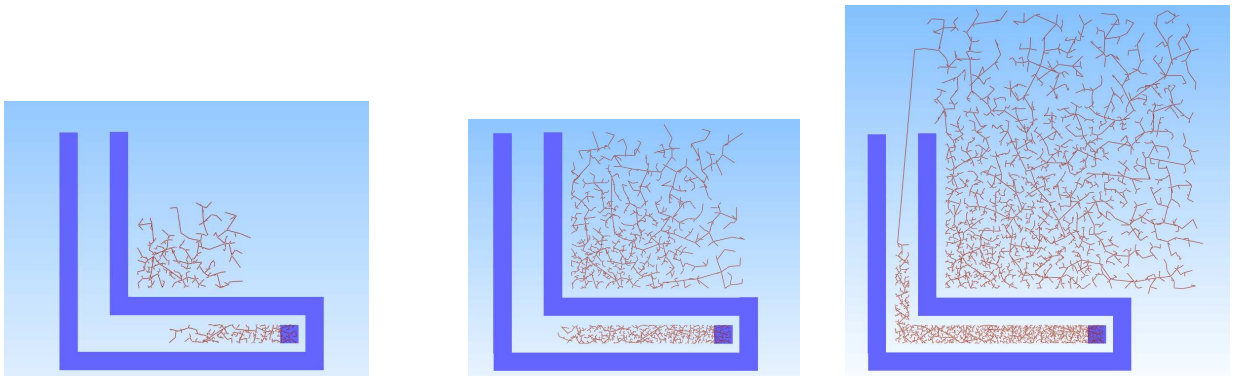

FIG. 1.3 – Exemple d'exécution de l'algorithme RRT. Planification du mouvement d'un carré en dimension 2.

Chapitre 2

Limites des algorithmes de diffusion

On a présenté les qualités des algorithmes de diffusion aléatoire au chapitre précédent. Il nous faut maintenant parler de leurs limites, qui ont motivé le travail réalisé lors de ce stage. Ce chapitre est destiné à présenter à la fois ces limites et les solutions ou adaptations qu'on peut trouver dans la littérature. C'est dans ce cadre précis que s'inscrit notre travail.

Les algorithmes de diffusion de type RRT sont souvent plus efficaces que des algorithmes déterministes, comme on l'a dit dans le chapitre précédent, lorsque l'espace de travail se complexifie. Toutefois, ils n'échappent pas à ce que Richard Bellman a appelé « the curse of dimensionality ». En effet, le temps mis par les algorithmes de diffusion à couvrir un espace varie de façon exponentielle avec la dimension de cet espace. Bien souvent, pourtant, un humain est capable d'identifier les dimensions pertinentes et de réduire la complexité de l'espace de travail. Ce travail de réduction de la dimension de l'espace dans lequel on applique les RRT est présenté dans une première section.

Par ailleurs, une notion plus difficile à quantifier, mais aisément compréhensible, est celle de la forme de l'espace libre, dans lequel l'arbre se développe. Quand un arbre doit passer par des « passages étroits », sa progression est fortement ralentie. Il va en effet buter contre les obstacles régulièrement, et de ce fait, la distance parcourue à chaque étape de diffusion est faible. La deuxième section de ce chapitre présente une caractérisation pertinente de ce qu'est un passage étroit dans CS_{free}, ainsi qu'une adaptation de l'algorithme RRT destinée à résoudre ce problème.

2.1 Réduction de la dimension

Le problème de devoir réduire la dimension de l'espace dans lequel on fait grossir des RRT se pose tout particulièrement dans deux cas précis :

- dans la planification de mouvement pour robot humanoïde (le robot HRP2 possède une trentaine de degrés de liberté)
- dans la planification de mouvement multi-robots.

2.1.1 Décomposition fonctionnelle

Dans le cas de la robotique humanoïde, on a affaire à un système anthropomorphe possédant de nombreux degrés de liberté. La solution présentée dans [3] pour simplifier les problèmes de planification de mouvement est une décomposition fonctionnelle. L'ensemble des degrés de

liberté du robot est partitionné en sous-ensembles selon le comportement attendu de chacun des degrés de liberté.

Concrêtement, on découple la planification en disant que les degrés de liberté des jambes servent à la locomotion, les degrés de liberté des bras servent à la manipulation et les degrés de liberté du tronc servent à l'adaptabilité du système global.

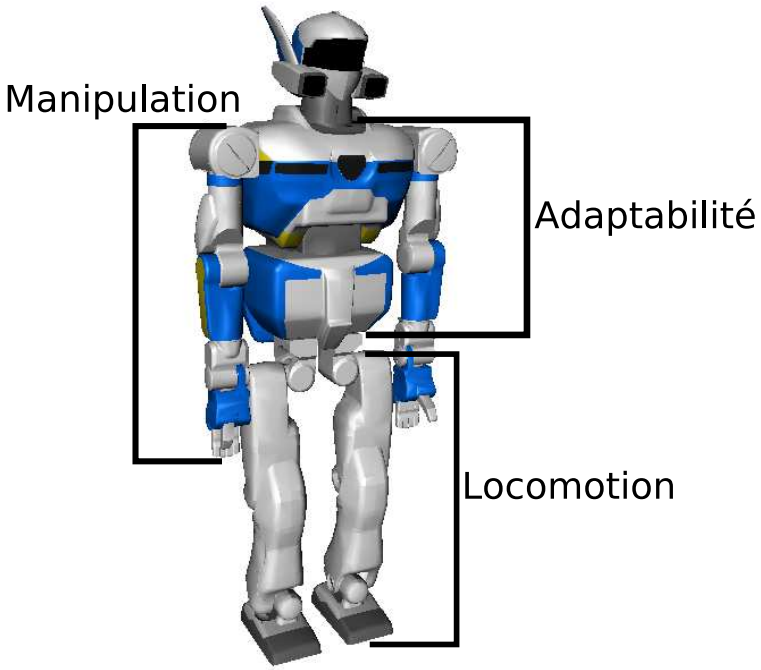

FIG. 2.1 – Le robot humanoïde HRP2 et la décomposition fonctionnelle.

Cette partition, inspirée évidemment de ce qu'on attend d'un robot humanoïde, permet de résoudre de nombreux problèmes efficacement, et - ce qui est important - les mouvements générés semblent naturels. Toutefois, certains cas précis viennent à bout de ce découplage. Ainsi, la tâche d'attraper une balle par terre est un problème de manipulation que le robot ne pourra exécuter sans planifier un mouvement de ses jambes. On peut imaginer de nombreux autres exemples qui appellent une stratégie de planification prenant en compte tous les degrés de liberté à la fois.

2.1.2 Manhattan-Like RRT

L'algorithme présenté maintenant est une adaptation de l'algorithme RRT, qui montre de bons résultats dans certains problèmes de planification multi-robots. Il a été conçu pour les problèmes de desassemblage pour des objets avec des parties articulées. Il a été introduit dans [1].

L'idée est ici de partitionner l'ensemble des degrés de liberté du système global en deux : les degrés actifs L_{act} et les degrés passifs L_{pas}. S'ensuit une hiérarchisation de la planification, on commence par calculer le mouvement des parties actives comme dans une étape de RRT standard. On analyse ensuite les parties du système qui ont empêché d'aller plus loin dans la diffusion. Si des parties actives sont en collision avec des parties passives mobiles, on note

les degrés de liberté passifs concernés L_{pas}^{col}, et on les prend en compte dans la deuxième partie de l'algorithme, qui planifie le mouvement des parties passives. Cette deuxième partie reprend le mécanisme de diffusion des RRT, la seule différence étant la façon dont on génère la configuration aléatoire : q_{rand}^{pas} est tirée dans un voisingae de q_{near} et ne diffère de q_{near} que pour les degrés de liberté qui appartiennent à L_{pas}^{col} .

Le nom doné à l'algorihtme vient de la forme des chemins obtenus dans CS_{free}. Ils ressemblent à des trajectoires de Manhattan, on avance par étape en utilisant successivement L_{act}, puis, orthogonalement, L_{pas}.

Algorithme 2 Manhattan-Like RRT

$\mathcal{T}$.INIT(q_0)
for $i = 1$ to K **do**
  $q_{rand}^{act} \leftarrow \text{Rand}(CS, L_{act})$
  $q_{near} \leftarrow \text{Nearest}(q_{rand}^{act}, \mathcal{T}, L_{act})$
  $(q_{new}, L_{pas}^{col}) \leftarrow ML - \text{Extend}(q_{near}, q_{rand}^{act})$
  $\mathcal{T}$.AddVertex(q_{new})
  $\mathcal{T}$.AddEdge(q_{near}, q_{new})
  $q_{near} \leftarrow q_{new}$
  while $L_{pas}^{col} \neq \varnothing$ **do**
    $q_{rand}^{pas} \leftarrow \text{Rand}(CS, q_{near}, L_{pas}^{col})$
    $(q_{new}, L_{pas}^{col'}) \leftarrow ML - \text{Extend}(q_{near}, q_{rand}^{pas})$
    $\mathcal{T}$.AddVertex(q_{new})
    $\mathcal{T}$.AddEdge(q_{near}, q_{new})
    $q_{near} \leftarrow q_{new}$
    $L_{pas}^{col} \leftarrow L_{pas}^{col'} \backslash L_{pas}^{col}$
  end while
end for

Les limites de cet algorithme sont du même ordre que celles de la décomposition fonctionnelle. La partition degrés actifs / degrés passifs se fait en entrée, par un utilisateur humain. Les gains en termes de complexité que ces algorithmes montrent par rapport aux RRT classiques donnent envie de les utiliser largement, mais cela ne pourra être fait qu'à condition de remplacer la connaissance de l'utilisateur sur la hiérarchie des degrés de liberté par une détection automatique des directions de CS dans lesquelles on doit explorer.

2.2 Caractérisation des passages étroits

Cette section est consacrée à l'étude du problème des passages étroits dans la littérature de la planification de mouvement. La première partie reprend un formalisme présenté par David Hsu dans sa thèse [6]. L'intérêt est surtout descriptif. Dans la deuxième partie, on présente un algorithme, dérivé de RRT, implémenté dans la solution KineoWorks, qui atténue les effets des passages étroits sur la vitesse de progression d'un RRT.

2.2.1 Espaces expansifs

L'étude qui suit n'a pas été faite pour caractériser les performances des RRT (l'article qui introduit les espaces expansifs [7] est antérieur à l'article qui a présenté les RRT tels qu'on les

utilise aujourd'hui [9]). Les algorithmes de planification de mouvement étudiés appartiennent à la famille des PRM (*Probabilistic Roadmaps*). Il s'agit de faire évoluer un graphe, et non un arbre, dans CS_{free}. Les sommets du graphe sont des configurations, et on ajoute une arête entre deux sommets si on peut les relier en ligne droite en restant dans CS_{free}. On voit que les RRT peuvent être vus comme une sous-famille des PRM. La différence principale est la notion de diffusion, absente des algorithmes PRM.

Toutefois, le problème posé par la présence de passages étroits dans CS_{free} touche pareillement PRM et RRT. La progression de ces algorithmes est ralentie par la faible probabilité de placer suffisament de nœuds dans un espace de volume faible pour passer de l'autre côté du passage. C'est pour cette raison qu'il nous a paru pertinent de présenter les résultats suivants, même s'ils ne concernent pas directement la famille d'algorithme sur lesquels nous avons travaillé.

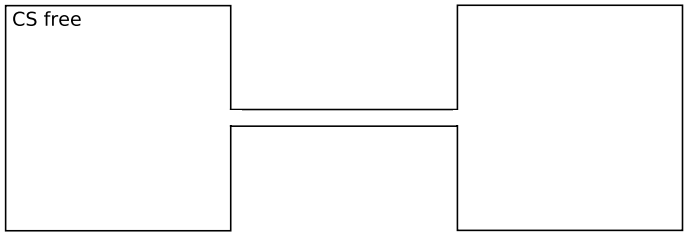

FIG. 2.2 – Un espace libre comprenant un passage étroit.

Notations. Pour tout sous-ensemble S de CS_{free}, $\mu(S)$ est le volume de S. On normalise en supposant $\mu(CS_{free}) = 1$. On dira qu'une configuration est visible depuis une autre s'il existe un chemin en ligne droite dans CS_{free} qui les relie et on notera $\nu(p)$ le sous-ensemble de $CSfree$ visible depuis p.

Définition 1. *On dira qu'un espace libre CS_{free} est $(\alpha, \beta, \varepsilon)$-expansible si et seulement si, pour toute composante connexe F de CS_{free}, on a :*

- $\forall p \in F$, $\mu(\nu(p)) > \varepsilon$
- *Pour tout S sous-ensemble connexe de F, l'ensemble*

$$LOOKOUT(S) = \{q \in S | \mu(\nu(q) \backslash S) \geq \beta \mu(F \backslash S)\}$$

est de volume $\mu(LOOKOUT(S)) \geq \alpha\mu(S)$.

La première condition est simple et traduit juste le fait que de tout point, on puisse voir au moins une fraction ε de l'espace libre.

En ce qui concerne la deuxième condition, $LOOKOUT(S)$ est le sous-ensemble de S dont on peut voir une fraction β du complémentaire de S. La condition est que ce sous-ensemble représente une fraction au moins α du volume de S. Les paramètres α et β traduisent donc le volume des points qui peuvent contribuer à de nouvelles zones de visibilité, lorsqu'on tire des configurations aléatoires à ajouter à la roadmap.

Ainsi, si on pense à S comme à la région de visibilité de l'ensemble des nœuds de la roadmap à un instant donné, dans un espace $(\alpha, \beta, \varepsilon)$-expansible, avec α et β grands, on a beaucoup de chances de tirer de nouvelles configurations qui augmenteront S de manière significative.

À l'inverse, des espaces comprenant des passages étroits ne pourront avoir que des α et β petits. Seule une faible fraction du volume de la zone qui est d'un côté du passage voit une fraction importante du complémentaire de cette zone à travers le passage.

Notons enfin que les choix d'α et β sont corrélés et ne sont pas uniques. Dans un espace avec des passages étroits, si on veut un grand β, on voit qu'α sera plus petit.

Muni de cette définition, on peut énoncer le théroème suivant :

Théorème 1. *Soit $\gamma \in]\,0,1\,[$ Soit S un ensemble de configurations, de cardinal $2n$ avec $n = \lceil 8\ln(8/\varepsilon\alpha\gamma)/\varepsilon\alpha + 3/\beta + 2 \rceil$. Les configurations sont choisies indépendemment et uniformément dans CS_{free}. On construit le graphe G de sommets ces configurations, en ajoutant des arêtes entre les configurations visibles l'une depuis l'autre. Alors avec une probabilité au moins $1-\gamma$, la topologie de G capture celle de CS_{free}, i.e. tout sous-graphe de G constitué de configurations appartenant à la même composante connexe de CS_{free} est connexe.*

Note. La démonstration de ce résultat peut être trouvée dans [7]. On commence par calculer la probabilité de pouvoir construire une suite de configurations de S de longueur n, telle que toute configuration de la suite voit son prédecesseur. Le résultat est ensuite obtenu en calculant l'espérance du volume de la zone de visibilité d'une telle suite.

La propriété obtenue (la topologie de G capture celle de CS_{free}) est très importante, car elle permet de répondre au problème de décision : existe-t-il un chemin de $q_{initial}$ à q_{final} ? En ce sens, on peut dire que la valeur de n présentée dans le théorème est la complexité du problème de planification de mouvement considéré.

La première remarque à faire est positive, il s'agit de la dépendance de n en γ : elle est logarithmique. Ce résultat était ainsi destiné à expliquer la puissance des algorithmes aléatoires en planification de mouvement : la probabilité d'erreur décroit exponentiellement avec le nombre de configurations tirées.

Toutefois, on voit que ce résultat est vrai à α et β fixés. La complexité en (α, β) est en $O(-\ln(\alpha)/\alpha + 1/\beta)$ qui tend vers $+\infty$ quand α ou β tendent vers 0. Il est difficile d'interpréter plus en avant ce résultat car, même si le travail de formalisation des passages étroits est intéressant, il ne donne aucune piste pour mesurer α et β dans un espace de configurations donné.

Décrivons quand même un exemple simple, pour lequel on peut évaluer α, β et ε.

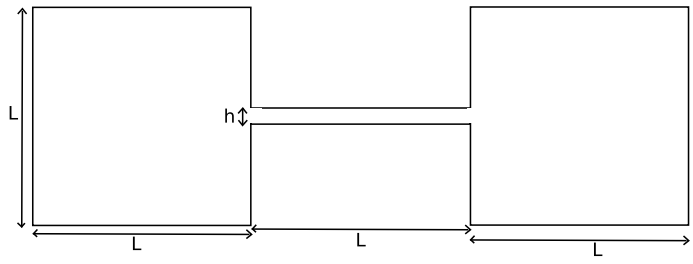

FIG. 2.3 – Un espace libre simple, où $\alpha, \beta, \varepsilon \sim (h/L)$.

Sur cet exemple, en dimension 2, on peut donner les ordres de grandeur de α, β et ε. Les points qui ont la plus faible visibilité sont dans le passage étroit. Ils voient un espace de volume $3Lh$, rapporté au volume total qui est en L^2. D'où $\varepsilon \sim (h/L)$. De plus, si on considère

le carré de gauche, seuls une fraction de volume $\sim Lh$ voit des points extérieurs au carré, et ces points voient un volume $\sim Lh$ du complémentaire du carré. D'où $\alpha \sim h/L$ et $\beta \sim h/L$. Si on fait le calcul avec le même exemple en dimension d, les deux hypercubes ont un volume L^d et le passage étroit est contraint dans k directions ($1 \leq k \leq d-1$), on trouve $\alpha, \beta, \varepsilon \sim (h/L)^k$. La complexité du problème est exponentielle en k.

Cette présentation permet de mesurer à quel point les passages étroits peuvent ralentir les algorithmes probabilistes de planification de mouvement, en particulier en dimension élevée. La section suivante présente un algorithme destiné à accélérer les RRT dans des espaces présentant des passages étroits.

2.2.2 Iterative Path Planning (IPP)

L'algorithme suivant a été présenté dans [4], et est implémenté dans la plateforme Kineo. Il est souvent plus efficace qu'un RRT simple, et pour cette raison, est l'algorithme utilisé par défaut par KineoPathPlanner.

L'idée de l'algorithme IPP est de tolérer au début de la planification une certaine « pénétration » du robot dans les obstacles. Les passages étroits de l'espace libre vont ainsi être élargis et le RRT va trouver un chemin entre $q_{initial}$ et q_{final} plus rapidement. Une fois ce premier chemin trouvé, on diminue la tolérance, en gardant les parties du chemin trouvé qui ne sont pas en collision. Ces parties vont aider le nouveau RRT à trouver un chemin plus vite. On continue jusqu'á ce que la tolérance soit nulle.

Algorithme 3 Iterative Path Planner

roadmap $\mathcal{R} \leftarrow \varnothing$
dynamic penetration $\delta \leftarrow$ Distance$(q_{initial}, q_{final})$
while $\delta > \varepsilon$ **do**
  path $\Gamma \leftarrow$ RRT$(q_{initial}, q_{final}, \mathcal{R})$
  $\delta \leftarrow \delta/\alpha$
  $\mathcal{R} \leftarrow$ RemoveCollidingParts(Γ)
end while
return $\mathcal{R}$

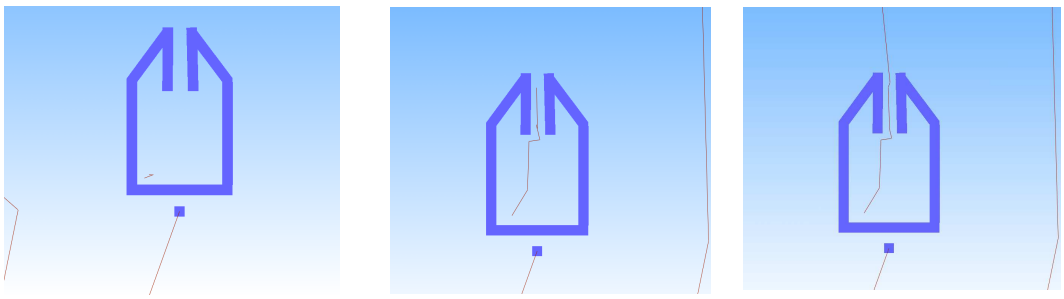

FIG. 2.4 – Exécution d'IPP pour un carré en dimension 2.

On voit ici une réalisation d'IPP sur un exemple en deux dimensions. En rouge, on voit la partie du chemin qui est conservée entre deux étapes, c'est-à-dire, les configurations du chemin trouvé qui ne sont pas en collision lorsqu'on diminue la pénétration. Le premier chemin trouvé

n'est pas intéressant, la pénétration a autorisé le cube à passer au travers de l'obstacle. Le deuxième chemin passe par le passage étroit, tout en étant en collision. Il permet de trouver le troisième chemin beaucoup plus rapidement qu'un RRT classique. On trouvera dans [4] une étude précise des performances comparées de RRT et IPP.

Cet algorithme est efficace. Il accélère notablement les RRT. Néanmoins, il profiterait, comme les autres algorithmes de diffusion, d'une méthode pour décrire l'espace libre qui controlerait la diffusion. C'est notre contribution au cadre des algorithmes de diffusion en planification de mouvement, et le sujet du chapitre suivant.

Chapitre 3

Mesure et contrôle de la diffusion

Ce chapitre présente les méthodes envisagées pour répondre au problème de la diffusion dans les passages étroits. Il s'agit d'une part de mesurer en ligne la « forme » de l'espace libre, à partir des résultats du processus de diffusion, et d'autre part de biaiser le processus de diffusion en fonction de cette mesure.

La première méthode envisagée repose sur une analyse unidimensionnelle de la diffusion, et définit, pour tout point de l'espace, une direction de diffusion privilégiée. La seconde méthode est multidimensionnelle. À tout point, elle associe une base ordonnée de l'espace, des directions à privilégier aux directions à éviter.

C'est cette deuxième méthode qui fera l'objet d'une étude précise dans le chapitre suivant.

3.1 Vitesse de dérive

3.1.1 Mesure de la vitesse de dérive

Cette présentation ne prétend pas être une démonstration mathématique, qui exhiberait certaines propriétés des RRT, mais sert à donner l'intuition qui a guidé l'élaboration de la méthode.

Une étape du processus aléatoire de création d'un chemin de l'arbre de recherche peut se réduire à ceci :

- On choisit un noeud aléatoirement dans l'arbre existant,
- On choisit une direction aléatoire dans $\mathbb{R}^n$,
- On avance, depuis le nœud choisi, dans la direction choisie, d'une distance aléatoire.

Cette vision est très floue, puisqu'elle ne précise pas à chaque étape les lois choisies pour le tirage aléatoire. Cependant, elle suffit à donner une intuition importante : les chemins de l'arbre sont des réalisations de marches aléatoires, dont les lois dépendent de la façon dont l'espace est couvert. À la limite, quand l'espace est couvert densément par le RRT, on peut considérer ces chemins comme des réalisations de mouvemement brownnien.

Il est donc pertinent de considérer ces chemins de l'arbre comme des trajectoires de particules dans une situation d'écoulement, de la configuration initiale vers la configuration finale. Et ainsi, en tout nœud de l'arbre, on peut définir une vitesse de dérive, qui est la moyenne des chemins qui partent de ce nœud.

Algorithmiquement, on stocke la vitesses de dérive associée à chaque nœud, et à chaque étape de l'algorithme, on met à jour les vitesses de tous les nœuds ancêtres du nœud ajouté.

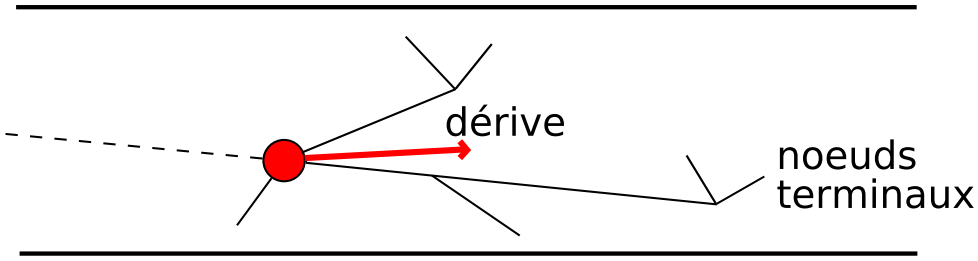

FIG. 3.1 – Mesure de la vitesse de dérive

Algorithme 4 Update Vector Field $(q_{new}, q_{near}, \mathcal{T})$

vector $newSpeed \leftarrow (q_{new} - q_{near})$
VectorField$[q_{new}] \leftarrow newSpeed$
double $weight \leftarrow 1$
node $currentNode \leftarrow q_{near}$
while $currentNode \neq \mathcal{T}$.root **do**
  int $M \leftarrow currentNode$.CountChildren()
  $weight \leftarrow weight/M$
  VectorField$[currentNode] \leftarrow (1-weight)$.VectorField$[currentNode]+weight.newSpeed$
  $currentNode \leftarrow$ father$[currentNode]$
end while

3.1.2 Contrôle de la diffusion

Le contrôle de la diffusion se fait à l'étape de l'algorithme RRT qui étend l'arbre. On a tiré une configuration aléatoire q_{rand} et on a choisi le nœud q_{near} depuis lequel on allait étendre l'arbre . Par ailleurs, on a stocké la vitesse de dérive en q_{near}, $\mathbf{v_{q_{near}}}$.

On va changer la direction vers laquelle on prolonge l'arbre en fonction de cette vitesse. q'_{rand}, la configuration aléatoire modifiée, est un barycentre de q_{rand} et de sa projection $P_\Delta(q_{rand})$ sur la droite Δ passant par q_{near} et de direction $\mathbf{v_{q_{near}}}$. Les poids respectifs de q_{rand} et de sa projection sont fonctions de $||\mathbf{v_{q_{near}}}||$. Quand la vitesse est nulle, $q'_{rand} = q_{rand}$. Quand $||\mathbf{v_{q_{near}}}|| \to \infty$, $q'_{rand} \to P_\Delta(q_{rand})$.

Plus précisément, si on note v_0 la moyenne des normes des vitesses de dérive dans l'arbre, on a :

$$q'_{rand} = (1-\lambda).q_{rand} + \lambda.P_\Delta(q_{rand}) \text{ où } \lambda = \frac{2}{\pi}\arctan(\frac{||\mathbf{v_{q_{near}}}||}{v_0})$$

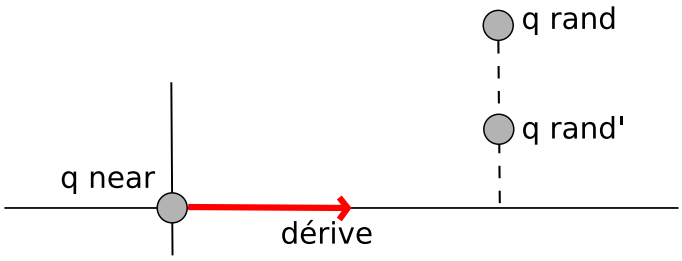

FIG. 3.2 – Projection de la configuration aléatoire vers l'axe de la vitesse de dérive

Ainsi, une sphère centrée en q_{near}, qui représente un tirage isotrope de la direction choisie pour étendre l'arbre, est transformée en une ellipsoïde, d'axe de révolution Δ, et de forme plus où moins allongée en fonction de la valeur de $||\mathbf{v_{q_{near}}}||$.

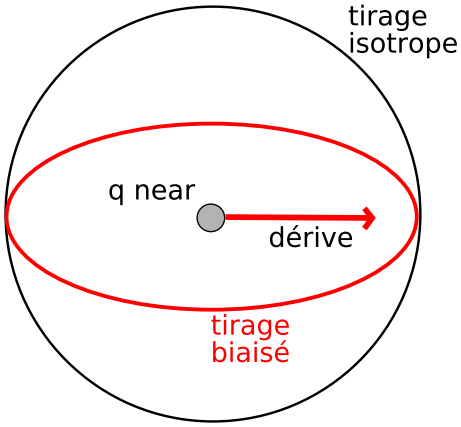

FIG. 3.3 – Transformation du tirage autour d'une configuration donnée

3.1.3 Limites

Cet algorithme a été implémenté, et fonctionne bien en pratique sur de nombreux exemples. Cependant, il souffre de certaines limites, dues principalement à la faiblesse d'une description unidimensionnelle.

Tout d'abord, dans de nombreux cas, une description unidimensionnelle est insuffisante, et ne peut traduire correctement la forme de l'espace libre. Ainsi, en dimension 3, on peut très bien imaginer une situation où deux dimensions sont peu contraintes et la troisième très contrainte. La direction mesurée par l'algorithme sera inclue dans le plan libre, mais sa seule donnée ne reflètera pas la liberté selon une autre direction. Les deux directions orthogonales à la vitesse de dérive seront traitées de la même manière par l'algorithme, alors qu'une des deux est contrainte et l'autre non.

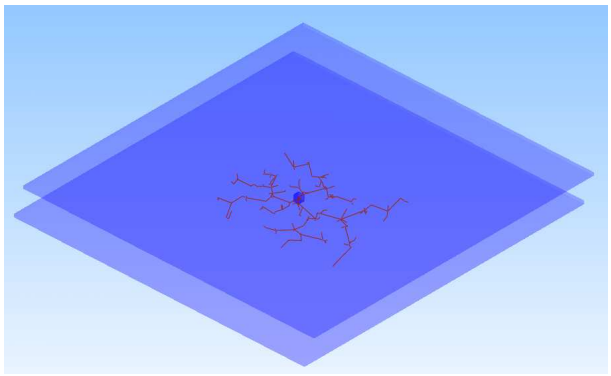

FIG. 3.4 – Cube coincé entre deux plans en dimension 3.

Une autre limite à cet algorithme est plus problématique. Il s'agit d'une situation de dégénéressence qui fait perdre certaines « bonnes » propriétés de recouvrement de l'espace des RRT. Dans un espace peu contraint, on va mesurer parfois des vitesses de dérive sensiblement plus élevées que la vitesse moyenne de dérive dans l'arbre. La situation est alors instable, car l'erreur d'appréciation de la forme de l'espace est accentuée aux étapes suivantes, du fait qu'on projette de plus en plus vers la direction trouvée. L'arbre ne s'étend alors que dans certaines directions favorisées, arbitraires, et ne couvre pas densément l'espace.

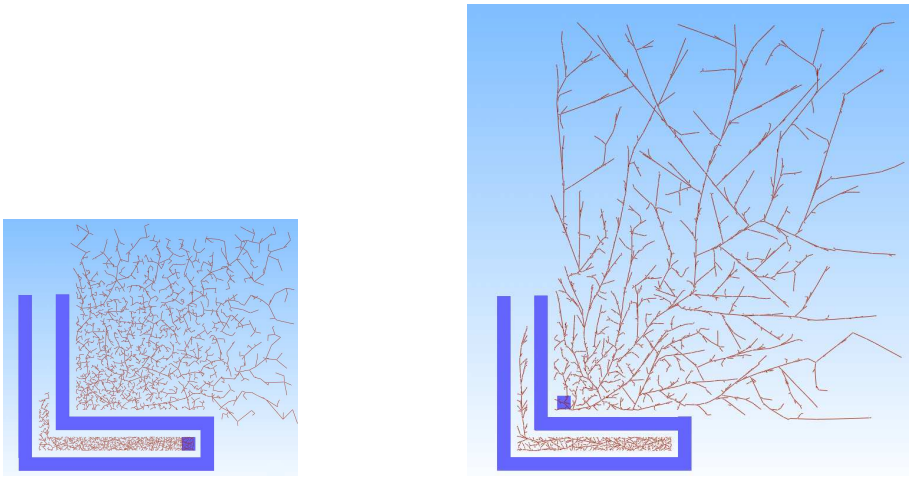

FIG. 3.5 – Comparaison RRT / Vitesse de dérive sur un exemple en dimension 2.

On voit sur cette capture d'écran que l'arbre contrôlé par notre algorithme se comporte de façon satisfaisante dans l'espace contraint, où il est plus rapide que le RRT. Cependant, dans un espace libre, on perd les propriétés de remplissage dense de l'espace propres aux RRT.

3.2 Description multidimensionnelle de la diffusion

Les limites de l'approche de la partie précédente appellent une analyse multidimensionnelle des processus de diffusion. La première idée fut de dériver le champ de vecteurs vitesse, pour favoriser, outre les directions des vitesses de dérive, celles selon lesquelles la différentielle du champ de vecteurs est positive. Cette propriété permet de caractériser les régions de l'espace où les vecteurs vitesses divergent, qui sont des zones peu contraintes, et celles où ils convergent, qui sont des passages étroits.

Cette idée a éé implémentée. Elle présente de bons résultats sous certaines conditions. Remarquons qu'elle necessite de nombreuses étapes de calcul : estimation des vecteurs vitesses, évalutation de la différentielle du champs de vecteur obtenue par des régressions linéaires. Le nombre de nœuds de l'arbre nécessaire à une estimation fiable des directions à favoriser croît avec chacun de ces calculs. On a donc préferé une autre méthode, très courante en traitement statistique de données hautement dimensionnées, et qui donne des résultats similaires pour moins de calculs numériques : l'analyse en composantes principales.

3.2.1 L'Analyse en Composantes Principales (ACP)

L'ACP est un outil statistique qui permet de décrire un nuage de p points dans un espace de dimension n, en cherchant les directions de l'espace selon lesquelles la variance des p points est maximisée. Concrêtement, l'ACP est un changement de base orthogonal tel que la plus grande variance des projections des p points soit obtenue en projetant sur la première coordonnée de la nouvelle base, puis sur la seconde, et ainsi de suite. Pour une description plus précise, on pourra se référer à [8].

On l'utilise souvent pour représenter des données d'un espace de grande dimension sur un espace de dimension plus petite, par exemple deux ou trois, afin de les visualiser.

Description de l'algorithme. On commence par calculer la matrice de covariance des p points dans la base canonique : C. La variance selon une direction donnée par un vecteur unitaire $\mathbf{u}$ est donnée par $\mathbf{u}^T C\mathbf{u}$. C'est une forme quadratique en $\mathbf{u}$. La matrice symétrique de la forme bilinéaire associée est C. C est diagonalisable en base orthonormée. Le maximum de $\mathbf{u}^T C\mathbf{u}$ avec la contrainte $\mathbf{u}^T\mathbf{u} = 1$ est donné par le vecteur propre unitaire $\mathbf{u_1}$ associé à la plus grande valeur propre λ_1. Le maximum de $\mathbf{u}^T C\mathbf{u}$ avec les contraintes $\mathbf{u}^T\mathbf{u} = 1$ et $\mathbf{u}^T\mathbf{u_1} = 0$ est donné par le vecteur propre unitaire $\mathbf{u_2}$ associé à la deuxième plus grande valeur propre λ_2, et ainsi de suite.

L'algorithme présenté prend donc en entrée un tableau de p vecteurs de dimension n, et renvoie les matrices Λ et Q contenant respectivement les valeurs propres de la matrice de covariance des p vecteurs et ses vecteurs propres associés.

Algorithme 5 Analyse en Composantes Principales(Points[p][n])

matrix $Covariance$
for $i = 1$ to n **do**
  $Covariance_{ii} \leftarrow$ Variance(Points[1][i], Points[2][i], ..., Points[p][i])
end for
for $i, j = 1$ to n, $i \neq j$ **do**
  $Covariance_{ij} \leftarrow$ Covariance ((Points[1][i], Points[2][i], ...), (Points[1][j], Points[2][j], ...))
  $Covariance_{ji} \leftarrow$ Covariance ((Points[1][i], Points[2][i], ...), (Points[1][j], Points[2][j], ...))
end for
matrix Λ, matrix Q
$\Lambda, Q \leftarrow$ EigenSolve($Covariance$)
return Λ, Q

3.2.2 L'ACP en planification de mouvement

L'ACP est fréquemment utilisée dans certains domaines comme la vision algorithmique, mais relativement peu en planification de mouvement. Il convient toutefois de signaler les travaux de Lydia Kavraki, et en particulier [16]. Cet article présente une application de l'analyse en composantes principales à un domaine qui ressort de la planification de mouvement. Il s'agit plus précisément de la modélisation de la flexibilité des protéines. Le probème est de décrire efficacement, en termes de planification de mouvement, le mouvement de protéines, c'est-à-dire de systèmes possédant plusieurs milliers de degrés de liberté.

Cet article montre que l'analyse en composantes principales permet de réduire la dimension de l'espace considéré à moins de quelques dizaines. Ce travail permet de mieux comprendre et décire les mouvements de protéines, ce qui permet, d'un point de vue biologique, de mieux comprendre leur fonction.

Il nous a semblé important de mentionner ces travaux parce qu'ils sont le seul exemple dans la littérature d'analyse en composantes principales appliquée à la planification de mouvement. Toutefois, on voit le point de vue différent que nous avons adopté : alors qu'il s'agit ici d'application uniquement descriptive, l'utilisation que nous faisons de l'ACP a pour but de contrôler un mouvement. Par ailleurs, les systèmes considérés n'ont rien à voir d'un point de vue calculatoire, nous étudions des robots possédant au plus une trentaine de degrés de liberté, alors que les molécules décrites ici ont entre quelques centaines et quelques milliers de degrés de liberté.

3.2.3 L'ACP appliquée aux méthodes de diffusion

On n'a pas trouvé dans la littérature d'application de l'ACP à des algorithmes de planification de mouvement par diffusion. La méthode présentée ici est donc une contribution nouvelle à la planification de mouvement.

L'ACP nous permet de décrire localement la forme de l'espace libre. La façon dont on utilise l'ACP ne modifie pas le mécanisme des algorithmes de diffusion (par exemple RRT). On ne fait que réimplémenter la fonction Extend, qui étend l'arbre depuis q_{near} vers q_{rand}, en modifiant la direction dans laquelle on progresse en fonction des résultats locaux de l'ACP sur les nœuds de l'arbre.

Description de l'algorithme

- On a choisi le nœud à étendre q_{near}.
- On effectue un parcours en largeur de l'arbre de recherche en partant de ce nœud jusqu'à avoir suffisament de points (voir la discussion au chapitre suivant).
- On effectue une ACP sur ces points, pour obtenir une base formée des vecteurs propres de la matrice de covariance des nœuds de l'arbre, et les valeurs propres associées.
- Dans le repére centré en q_{near} et muni de cette base, on obtient les coordonnées de q'_{rand} à partir de celles de q_{rand} ainsi : si on pose $\lambda_1 > \lambda_2 > \cdots > \lambda_n > 0$ valeurs propres de C matrice de covariance, et $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_n}$ les vecteurs propres associés (vecteurs de base du repère considéré), la i$^{\text{ème}}$ coordonnée de q'_{rand} est

$$q'^{(i)}_{rand} = \frac{\lambda_i}{\lambda_1} q^{(i)}_{rand}$$

D'où, dans le repère canonique :

$$q'_{rand} = q_{near} + \sum_{i=1}^{n} \left(\frac{\lambda_i}{\lambda_1} (q_{rand} - q_{near}).\mathbf{u_i} \right) \mathbf{u_i}$$

On privilégie donc les directions selon lesquelles la variance observée est la plus grande, et on projette les directions selon lesquelles la variance est faible vers q_{near}. La dessin de la section précédente (fig. 3.3) est encore valable, sauf que chaque dimension de l'ellipsoïde est modifiée selon la variance selon cette dimension. Ainsi, on dépasse les limites descriptives de la méthode précédente, qui ne pouvait pas favoriser identiquement deux directions orthogonales.

Algorithme 6 ACP-extend($q_{near}, q_{random}, \mathcal{T}$)

vector table Points[p][n]
Points← BreadthFirstSearch($\mathcal{T}, q_{near}, p$)
matrix Λ, matrix Q
$\Lambda, Q \leftarrow$ ACP(Points)
Sort(Λ, Q)
$q_{random} \leftarrow q_{near} + \sum_{i=1}^{n} \left(\frac{\Lambda_{ii}}{\Lambda_{11}} (q_{random} - q_{near}).Q_i \right) Q_i$
$q_{new} \leftarrow$ RRT-extend(q_{near}, q_{random})
return q_{new}

Remarque. On remarque que l'algorithme présenté ci-dessus possède un paramètre à régler : le nombre p de voisins utilisés pour l'ACP. Grossièrement, le choix pour ce nombre dépend de la dimension de l'espace de configurations, et de la forme de l'espace libre autour du point considéré. Dans une première implémentation, on lui a donné une valeur qui dépend linéairement de la dimension de CS. Une partie du chapitre suivant est consacrée à une étude qui permet de supprimer ce paramètre de l'algorithme.

Adaptation de l'algorithme. Implémenté tel quel, l'algorithme présente des limites semblables à celles décrites à propos de l'algorithme de la vitesse de dérive. La diffusion contrôlée par l'ACP donne au RRT une inertie dans sa progression. Si l'espace libre a une forme de tunnel, par exemple, avec deux possibilités, aller tout droit ou prendre un virage, l'ACP va favoriser le fait d'aller tout droit. Le RRT classique, dans ce cas, explore les deux possibilités avec même probabilité. La solution implémentée au cours du stage consiste à utiliser avec probabilité 1/2 le RRT et avec probabilité 1/2 l'ACP à chaque étape de diffusion. L'idée est qu'on profite des propriétés de terminaison des RRT, en accélérant la diffusion dans les passages contraints grâce à l'ACP.

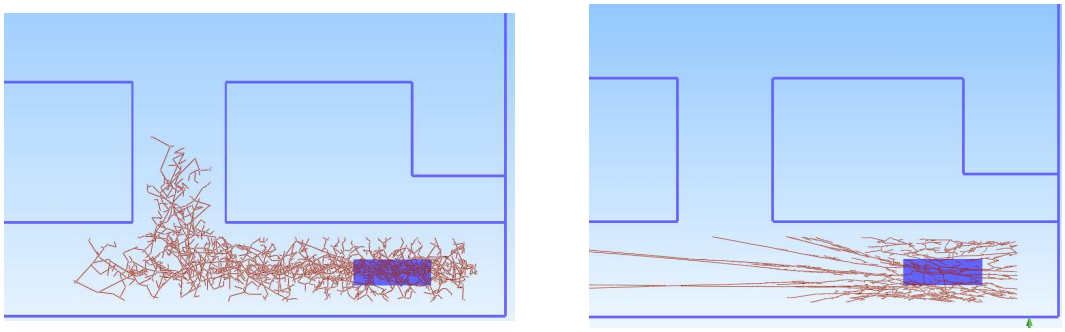

FIG. 3.6 – Comparaison RRT / ACP pour un rectangle en dimension 2.

On voit sur la capture d'écran ci-dessus que l'algorithme ACP va plus vite dans les lignes droites, mais qu'il ne peut pas prendre un virage dans une direction orthogonale à sa direction privilégiée. Ce problème est réglé en utilisant l'algorithme de diffusion traditionnel la moitié du temps.

3.2.4 Discussion sur les extensions de l'ACP

L'analyse en composantes principales est un outil statistique connu et largement utilisé. De nombreux travaux détaillent ses propriétés de convergence et d'optimalité. Ce sont ces raisons qui nous ont poussé à l'utiliser. Par ailleurs, un avantage certain de l'ACP est l'interprétation directe qu'on peut faire de ses résultats.

Toutefois, il convient de signaler l'existence d'un champ de recherche très actif sur des méthodes non linéaires de réduction de dimension. Les articles [15] et [14] présentent respectivement les méthodes ISOMAP et LLE. Ces méthodes semblent mieux adaptées que l'ACP pour des problèmes de réduction de dimension tirés de la vision algorithmique, où les espaces auxquels appartiennent les données ne sont pas nécessairement linéaires.

Nous avons fait le choix de ne pas utiliser ces méthodes pour plusieurs raisons. Tout d'abord, utiliser ACP donne un premier cadre de travail, et permet d'étudier l'efficacité de méthodes de réduction de dimension dans la planification de mouvement probabiliste. Des adaptations pour utiliser d'autres méthodes plus récentes pourront être faites plus tard. Il se trouve par ailleurs que dans beaucoup d'exemples étudiés, le sous-espace à découvrir est effectivement linéaire. En effet, si par exemple on considère un robot dont certains degrés de liberté sont fortement contraints, l'espace libre est une intersection d'hyperplans de l'espace des configurations, et est donc linéaire. Enfin, et c'est la remarque la plus importante, l'application qu'on fait de l'ACP ici est locale. À chaque étape, on n'utilise que les plus proches voisins du point considéré, et on découvre en fait, plutôt que la sous-variété à laquelle appartiennent les points, l'espace tangent à cette sous-variété. C'est essentiellement l'esprit de certains algorithmes de réduction de dimension non linéaires, par exemple LLE (*Locally Linear Embedding*) [14], et notre méthode nous permet donc de caractériser des espaces libres non linéaires. La finesse de l'analyse dépend évidemment de la densité de points dont on dispose, mais comme on hérite des propriétés de terminaison des RRT, on sait qu'à la limite, on obtient une couverture dense de l'espace, et donc une description suffisament fine.

Chapitre 4

Analyse de l'algorithme

L'analyse de l'algorithme présenté au chapitre précédent se fait à plusieurs niveaux. Tout d'abord, comme on l'a dit précédemment, l'ACP, parce qu'elle est très répendue, est un objet d'études important. Il nous a paru intéressant de consulter des résultats publiés au sujet de la convergence des résultats donnés par l'ACP, et de les confronter aux résultats empiriques qu'on a pu obtenir sur des simulations simples.

Une des raisons qui ont poussé à faire ces études est la volonté de supprimer tous les réglages de paramètres de l'algorithme. On a vu dans l'implémentation de notre algorithme qu'on avait besoin du nombre de points sur lesquels on applique l'ACP. Les études théoriques permettent de justifier un choix numérique, mais on propose mieux : l'option d'une implémentation incrémentale de l'ACP, qui permet de s'affranchir totalement de ce paramètre.

Nous détaillons par ailleurs des résultats sur l'effet théorique du contrôle de la diffusion, sous l'hypothèse que la description de l'espace libre par l'ACP est exacte. Nous présentons aussi une étude rapide de la complexité de l'algorithme implémenté.

Enfin, et il s'agit de la partie la plus importante d'un point de vue pratique, on présente les résultats obtenus par notre algorithme sur quelques exemples. On les comparera d'une part aux RRT classiques, mais aussi, puisqu'on en avait la possibilité sur la plateforme Kineo, à l'algorithme IPP.

4.1 Convergence de l'ACP

4.1.1 Résultats théoriques

Les résultats présentés ici sont tirés de la thèse de Laurent Zwald [17]. Ils ont été présentés pour la première fois dans [18]. Le but n'est pas de présenter les démonstrations ni même les idées mathématiques qui les guident, mais plutôt quelques résultats pour forger l'intuition sur la vitesse de convergence de l'ACP. On a retenu des résultats faciles à exprimer, en vulgarisant, en dehors d'un cadre formel strict.

On considère une variable aléatoire X, qui prend ses valeurs dans un espace mesurable $\mathcal{X}$ suivant une distribution P. On a accès à n réalisations de X, sur lesquelles on effectue une ACP. On note C la matrice de covariance de X et C_n la covariance empirique : mesurée après n tirages. On s'intéresse par ailleurs au sous-espace de dimension D engendré par les D vecteurs propres associés aux D plus grandes valeurs propres de C. On note ce sous-espace S_D, et la projection sur ce sous-espace P_{S_D}. Son équivalent empirique (obtenu à partir de C_n)

est noté $\widehat{S_D}$ (respectivement $P_{\widehat{S_D}}$).

Les deux résultats présentés dans cette partie concernent :

– la vitesse de convergence de C_n vers C,

– la vitesse de convergence de $P_{\widehat{S_D}}$ vers P_{S_D}.

Remarque. Les résultats présentés dans la thèse sont plus généraux, et considèrent des opérateurs agissant dans un espace de Hilbert. La distance entre les opérateurs mesurés et réels utilise la norme de Hilbert-Schmidt. Nous sommes quant-à-nous toujours en dimension finie et dans la suite, nous ne préciserons donc pas la norme utilisée dans les théorémes. Les opérateurs de covariance ou de projection peuvent être assimilés à leur matrice dans la base canonique.

Le premier résultat est un lemme sur la convergence de la covariance empirique.

Lemme 1. *On suppose qu'il existe $M \in \mathbb{R}$ tel que pour tout $x \in \mathcal{X}$, $||x||^2 \leq M$. Alors, pour tout $\xi > 0$, avec probabilité supérieure à $1 - e^{-\xi}$,*

$$||C_n - C|| \leq \frac{2M}{\sqrt{n}} \left(1 + \sqrt{\frac{\xi}{2}}\right)$$

La condition de compacité de l'espace dans lequel on tire les points est toujours respectée dans les situations où on utilise l'ACP. On observe une décroissance en $\frac{1}{\sqrt{(n)}}$, qui n'est pas surprenante si on pense, par exemple, au théorème central limite.

Ce résultat est surtout présenté pour amener le théorème suivant. On donne ici une borne sur la vitesse de convergence des projecteurs sur les sous-espaces propres que retourne l'ACP.

Théorème 2. *On suppose qu'il existe $M \in \mathbb{R}$ tel que pour tout $x \in \mathcal{X}$, $||x||^2 \leq M$. Soient S_D et $\widehat{S_D}$ les sous-espaces engendrés par les D premiers vecteurs propres de C, respectivement C_n, comme défini précédemment. On note $\lambda_1 > \lambda_2 > \dots$ les valeurs propres de C, si $D > 0$ est tel que $\lambda_D > 0$, on pose $\delta_D = \frac{1}{2}(\lambda_D - \lambda_{D+1})$ et*

$$B_D = \frac{2M}{\delta_D} \left(1 + \sqrt{\frac{\xi}{2}}\right)$$

Alors, pour $n \geq B_D^2$, pour tout $\xi > 0$, avec probabilité supérieure à $1 - e^{-\xi}$,

$$\left|\left|P_{\widehat{S_D}} - P_{S_D}\right|\right| \leq \frac{B_D}{\sqrt{n}}$$

Ce résultat appelle plusieurs remarques. Tout d'abord, on retrouve la même dépendance en $\frac{1}{\sqrt{n}}$ que pour le lemme précédent. Notons ensuite que B_D ne dépend que de l'écart entre la $D^{\text{ème}}$ et la $(D+1)^{\text{ème}}$ valeur propre. La valeur de D elle-même, la dimension de l'espace total, et les autres relations entre les valeurs propres n'interviennent pas. C'est ce qui fait la force de ce résultat, et la simplicité avec laquelle on peut l'interprêter.

Enfin une remarque pour revenir au cadre de la planification de mouvement. La dépendance en $\frac{1}{\delta_D}$ est naturelle : plus l'espace est étroit (en fait, plus l'écart entre les « grandes » et les

« petites » valeurs propres est grand), moins il faut de points pour que la description donnée par l'ACP converge. C'est un résultat encourageant pour l'utilisation qu'on fait de l'ACP. Les espaces où la diffusion doit être contrôlée pour améliorer les résultats sont précisément ceux où l'ACP donnera les meilleurs résultats. Si δ_D est faible, c'est-à-dire, si toutes les valeurs propres sont proches les unes des autres, alors l'espace est localement aussi contraint dans toutes les directions, et les algorithmes de diffusion classiques donnent de bons résultats.

4.1.2 Expériences sur l'ACP

On a réalisé des expériences simples pour vérifier la vitesse de convergence de notre implémentation de l'analyse en composantes principales et la confronter aux résultats théoriques énoncés ci-dessus.

On se place dans un espace de dimension d, où k dimensions sont contraintes ($1 \leq k \leq d-1$), et on tire n points uniformément sur lesquels on réalise une ACP. On répète cette simulation un certain nombre de fois pour chaque valeur de n, et on fait varier n. On a ensuite mesuré la variance de la distance entre le sous-espace obtenu par l'ACP et le sous-espace formé par les dimensions « libres ». On montre ici deux exemples, en dimension 16 et 30, sur lesquels on fait apparaître un modèle théorique (en rouge) qui varie en $1/n$, puisqu'il s'agit de la variance. L'écart-type, qui peut être assimilé à l'erreur moyenne de l'ACP, varie donc avec ce modèle en $1/\sqrt{n}$, comme dans les résultats des théorèmes précédents.

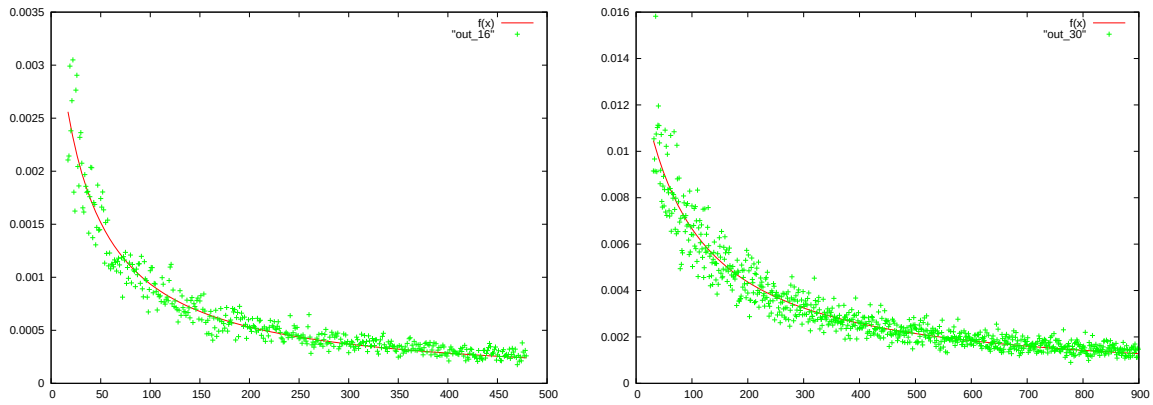

FIG. 4.1 – Variance des résultats donnés par l'ACP en fonction du nombre de points utilisés.

L'espace dans lequel on a tiré les points est un hyper-parallélépipède. Il est trois fois plus épais selon les dimensions libres que selon les dimensions contraintes. k vaut $d/4$.

On a fait varier ces valeurs dans les expériences, pour voir si elles correspondaient aux bornes indiquées dans le théorème. Les dépendances se font dans le bon sens (plus l'espace est étroit, plus l'ACP converge vite), mais les valeurs trouvées ne sont pas suffisament précises pour pouvoir corroborer la dépendance quantitativement, comme il est possible de le faire pour la dépendance en le nombre de points.

4.2 Implémentation incrémentale de l'ACP

La partie précédente a présenté des résultats qui permettent de justifier le choix d'un nombre de points sur lesquels appliquer l'ACP, qui permet de garantir un résultat avec une erreur majorée, avec une certaine probabilité. On présente ici une autre approche, inspirée par des résultats présentés dans [10], à propos d'algorithmes de planification de mouvement par visibilité. L'idée est d'arrêter un processus aléatoire lorsqu'une certaine quantité converge. Dans le cas d'algorithme par visibilité, il s'agit de la fraction d'espace visible depuis les points de la roadmap.

Ici, on mesure la dimension de la sous-variété mise en évidence par l'ACP. Lorsque cette dimension se stabilise, on estime que l'ACP a suffisament de points pour bien décrire l'espace libre, et on l'arrête. Évidemment, l'implémentation de cette méthode nécessite une version incrémentale de l'ACP. La version classique de l'ACP présentée plus haut nécessiterait de refaire les calculs de diagonalisation à chaque fois qu'on rajoute un point. Ce n'est pas envisageable pour des raisons de temps de calcul.

Il existe de nombreuses versions incrémentales de l'ACP. La plupart diminuent les calculs nécessaires à chaque étape en faisant évoluer un sous-espace de dimension fixe, dont on estime qu'il représente bien les données. Nous ne pouvons pas utiliser ces méthodes ici, car notre utilisation de l'ACP prend en compte toutes les dimensions de l'espace. Même celles avec une faible variance sont conservées et servent au moment de contrôler la diffusion de l'arbre.

La version incrémentale de l'ACP que nous avons implémentée a été présentée dans [2]. Elle repose sur une analyse des perturbations des matrices diagonales, et l'implémentation a montré de bons résultats en pratique.

4.2.1 Perturbations de matrices diagonales

On suppose pour le calcul qui vient que les données utilisées pour l'ACP sont centrées. Le calcul dans le cas de données non-centrées est similaire, mais un peu plus long. L'esprit de ce calcul, qui réside dans l'analyse de perturbation de matrices diagonales au rang 1, est le même.

Mise à jour de la covariance

On suppose qu'on a réalisé une ACP sur k vecteurs $(\mathbf{x_i})_{i=1..k}$. On veut mettre à jour ce calcul après la mesure de $\mathbf{x_{k+1}}$. La nouvelle covariance C_{k+1} vaut :

$$C_{k+1} = \frac{1}{k+1}\sum_{i=1}^{k+1} \mathbf{x_i}\mathbf{x_i}^T = \frac{k}{k+1}C_k + \frac{1}{k+1}\mathbf{x_{k+1}}\mathbf{x_{k+1}}^T \tag{4.1}$$

On a déjà diagonalisé C_k en base orthonormée, et on veut diagonaliser C_{k+1}. On note $C_k = Q_k\Lambda_k Q_k^T$ et $C_{k+1} = Q_{k+1}\Lambda_{k+1}Q_{k+1}^T$ où Q est la matrice composée des vecteurs propres et Λ la matrice diagonale des valeurs propres. De plus, on note $\mathbf{y_{k+1}} = Q_k^T\mathbf{x_{k+1}}$. C'est le nouveau vecteur exprimé dans la base de vecteurs propres de l'ancienne covariance.

En réinjectant ces notations dans (4.1), on obtient :

$$Q_{k+1}\Lambda_{k+1}Q_{k+1}^T = Q_k\left[\frac{k}{k+1}\Lambda_k + \frac{1}{k+1}\mathbf{y_{k+1}}\mathbf{y_{k+1}}^T\right]Q_k^T \tag{4.2}$$

Ainsi, il nous suffit de diagonaliser $\frac{k}{k+1}\Lambda_k + \frac{1}{k+1}\mathbf{y_{k+1}}\mathbf{y_{k+1}}^T$, pour obtenir Q_{k+1} et Λ_{k+1}. Si on pose

$$\frac{k}{k+1}\Lambda_k + \frac{1}{k+1}\mathbf{y_{k+1}}\mathbf{y_{k+1}}^T = VDV^T$$

avec D diagonale et V orthonormale, on obtient :

$$\begin{cases} Q_{k+1}=Q_kV \\ \Lambda_{k+1}= D \end{cases}$$

La matrice à diagonaliser a une structure très particulière : c'est la somme d'une matrice diagonale et d'une matrice de norme très petite devant celle de la matrice diagonale. Le calcul qui suit montre comment la diagonaliser en faisant des approximations au rang 1 sur $\mathbf{y_{k+1}}\mathbf{y_{k+1}}^T$.

Analyse de perturbations sur une matrice diagonale

On veut diagonaliser $(\Lambda + \alpha\alpha^T)$ avec $\alpha\alpha^T$ petit devant Λ. On note $(\Lambda + \alpha\alpha^T) = VDV^T$. Comme la matrice à diagonaliser est proche de Λ, on pose $D = \Lambda + P_\Lambda$ et $V = I + P_V$: les valeurs propres de $(\Lambda + \alpha\alpha^T)$ sont proches des valeurs propres de Λ et ses vecteurs propres aussi. P_Λ et P_V sont petites devant Λ, respectivement I. En développant VDV^T, on obtient :

$$\begin{aligned} VDV^T= & (I + P_V)(\Lambda + P_\Lambda)(I + P_V)^T \\ =&\Lambda + P_\Lambda + DP_V^T + P_VD + P_V\Lambda P_V^T + P_VP_\Lambda P_\Lambda^T \end{aligned}$$

Les termes $P_V\Lambda P_\Lambda^T$ et $P_VP_\Lambda P_V^T$ sont négligeables (ordre 2 en (P_V, P_Λ)). En identifiant $VDV^T = \Lambda + \alpha\alpha^T$, on obtient :

$$\alpha\alpha^T = P_\Lambda + DP_V^T + P_VD \tag{4.3}$$

Par ailleurs, V est orthogonale, ce qui se traduit, pour P_V, par : $(I + P_V)(I + P_V)^T = I$. Le terme $P_VP_V^T$ est négligeable, ce qui donne : $P_V = -P_V^T$.

On obtient donc les coefficients de P_Λ et P_V à partir de ceux de α grâce à (5.3). Si on note λ_i le $i^{\text{ème}}$ élément diagonal de Λ, et α_i la $i^{\text{ème}}$ coordonnée de α, on a :

$$\begin{cases} (P_\Lambda)_{ij}= & 0 & (\text{si } i \neq j) \\ (P_\Lambda)_{ii}= & \alpha_i^2 & \\ (P_V)_{ij}= & \frac{\alpha_i\alpha_j}{\lambda_j+\alpha_j^2-\lambda_i-\alpha_i^2} & (\text{si } i \neq j) \\ (P_V)_{ii}= & 0 & \end{cases}$$

4.2.2 Adaptation de l'algorithme

Le calcul précédent montre comment calculer de manière incrémentale l'ACP. Notre algorithme peut donc être adapté. À chaque étape, on ajoute un nouveau point, voisin du noeud de l'arbre à étendre, on met à jour le calcul de l'ACP, on évalue la dimension de la sous-variété découverte. Quand cette valeur converge, on arrête et on estime avoir correctement évalué la forme de l'espace libre au voisinage du point considéré.

Algorithme 7 incremental ACP(q_{near})

vector table Points$[n+1][n]$
Points$\leftarrow$ BreadthFirstSearch$(\mathcal{T}, q_{near}, n+1)$
matrix Λ, matrix Q
$\Lambda, Q \leftarrow$ ACP(Points)
int $d \leftarrow$ MeasureDimension(Λ)
list $L \leftarrow [d]$
int $k \leftarrow (n+1)$
while L has not converged **do**
  get new point $\mathbf{x}$ by continuing the BFS
  vector $\mathbf{y} \leftarrow Q^T\mathbf{x}$
  find perturbation matrices P_V and P_Λ corresponding to $\frac{k}{k+1}\Lambda + \frac{1}{k+1}\mathbf{y}\mathbf{y}^T$
  $Q \leftarrow Q \times (I + P_V)$
  $\Lambda \leftarrow \Lambda + P_\Lambda$
  $d \leftarrow$ MeasureDimension(Λ)
  $L \leftarrow L :: d$
  $k \leftarrow (k+1)$
end while
return Λ, Q

Remarque. L'évaluation de la dimension de la sous-variété trouvée se fait ainsi : on trie les valeurs propres de la matrice de covariance, puis on cherche les deux valeurs propres consécutives dont l'écart est maximal (écart mesuré comme la différence ou le rapport entre les deux valeurs propres). Les « grandes » valeurs propres correspondent aux directions de la sous-variété. Leur nombre donne la dimension recherchée.

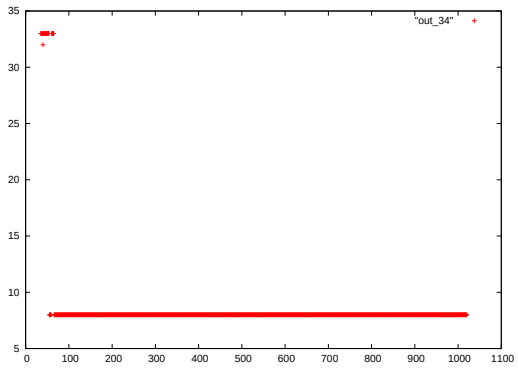

FIG. 4.2 – Convergence de la dimension mesurée.

On voit ci-dessus un exemple de convergence de la dimension mesurée par l'ACP. En abscisse, le nombre de voisins utilisés et en ordonnée la dimension obtenue. L'espace est de dimension totale 34 et la sous-variété à identifier de dimension 8. Après une soixantaine d'itérations, la courbe se stabilise sur 8.

4.3 Analyse du contrôle de la diffusion

Toutes les analyses précédentes présentent des résultats sur la qualité et la convergence de la description de l'espace libre. L'autre partie à analyser de l'algorithme est celle qui prend en entrée cette description et qui transforme le processus de diffusion. On présentera dans la suite du chapitre des comparaisons entre notre algorithme et les algorithmes de diffusion traditionnels sur des exemples réels.

On compare ici théoriquement une étape de diffusion d'un RRT et de notre algorithme. Voici la situation : on se place dans un espace de dimension 2, localement rectangulaire, et dont une des deux dimensions varie. On s'interesse à la distance moyenne parcourue par les algorithmes considérés en une étape de diffusion. On suppose que l'ACP donne une description locale correcte de l'espace (les directions des vecteurs propres sont celles du rectangle, et les valeurs propres varient comme le carré des dimensions du rectangle). On tire une configuration aléatoire sur un cercle centré sur la configuration à étendre, et de rayon la grande dimension du rectangle. C'est la distance maximale que peuvent parcourir chacun des algorithmes en une étape.

Toutes ces hypothèses ont pour but de permettre un calcul explicite de la distance moyenne parcourue.

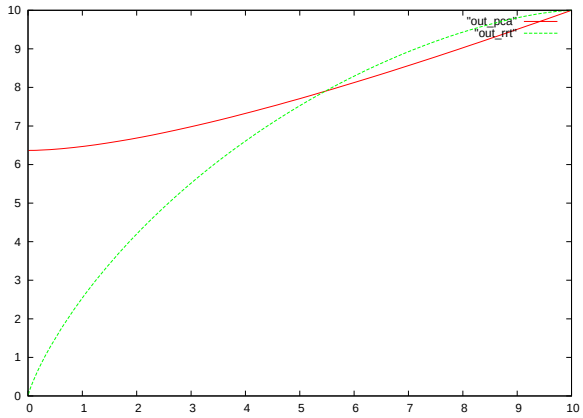

Fig. 4.3 – Comoparaison théorique de la diffusion classique et contrôlée par l'ACP.

En rouge la distance moyenne parcourue par l'ACP à chaque étape, et en vert par un RRT classique. En abscisse l'épaisseur du petit coté du rectangle entre 0 et 10. Le grand coté est constant et vaut 10.

Quelques remarques simples tout d'abord : si le rectangle est un carré, les deux algorithmes donnent le même résultat. C'est normal, car l'algorithme ACP ne change pas la configuration tirée. Quand le rectangle a une forme peu allongée, le RRT non modifié donne de meilleurs résultats. On peut l'expliquer en disant qu'alors, l'ACP projette « trop » sur la direction principale, ce qui fait perdre de la distance parcourue, alors que l'espace est peu contraint.

Quand l'épaisseur de la dimension contrainte tend vers 0, on voit que la distance moyenne parcourue par le RRT tend vers 0 aussi, ce qui est compréhensible. La distance moyenne parcourue par l'ACP tend vers une constante strictement positive. Expliquons ce qu'il se passe. L'ACP appliquée à la loi uniforme sur le cercle sur lequel on tire la configuration aléatoire donne une ellipse, sur laquelle la mesure est concentrée sur les extrémitées. Le schéma suivant

montre le cercle initial et l'ellipse, dans le cas où la dimension contrainte est trois fois moins épaisse que la dimension libre. On a indiqué en rouge où se trouve 50% de la mesure.

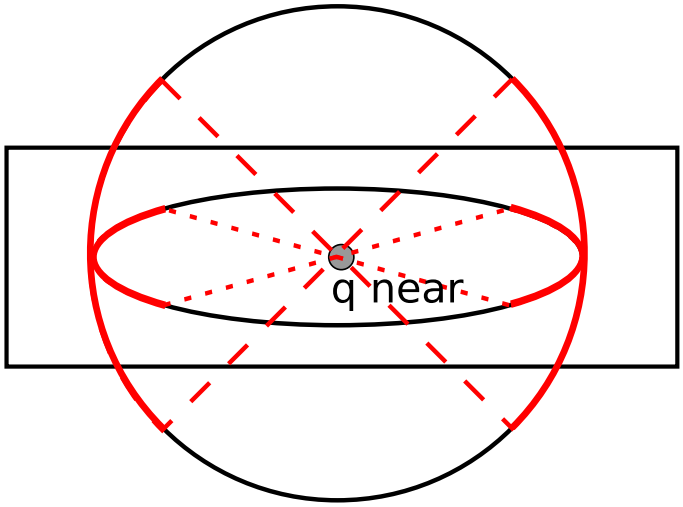

FIG. 4.4 – L'effet du contrôle de la diffusion pour un espace de dimension 2 de rapport 1/3.

On voit sur cette figure l'effet du contrôle de la diffusion. La projection « écrase » la mesure sur les bouts de l'ellipse dès que le passage est étroit.

4.4 Complexité

Commençons par étudier la complexité de l'algorithme avec l'implémentation classique de l'ACP.

La complexité de la méthode proposée dépend, à chaque étape de diffusion :

– du nombre p de voisins utilisés pour réaliser l'ACP,

– de la dimension d de CS.

Le calcul des moyennes et variances pour remplir la matrice de covariance se fait en $O(d^2 \times p)$: il y a d^2 cases de la matrice à remplir, et chacune nécessite $O(p)$ additions. Une fois la matrice obtenue, il faut la diagonaliser. La diagonalisation d'une matrice symétrique se fait en $O(d^3)$.

Dans la première implémentation de l'ACP (non incrémentale), on utilise un nombre de voisins $p = O(d)$. La complexité de la méthode, à chaque étape de diffusion, est donc $O(d^3)$.

Le calcul est similaire dans le cas de l'ACP incrémentale, sauf qu'on a pas a priori, de bornes sur p. Dans les faits, on a observé que jusqu'en dimension 50, p variait linéairement avec la dimension de l'espace total, ou du moins, pouvait être majoré par une fonction linéaire de d.

Ce résultat appelle plusieurs remarques. Tout d'abord, on a vu dans l'étude de l'effet des passages étroits sur les algorithmes de diffusion que la complexité de ces algorithmes, dans des espaces compliqués, était exponentielle en d. Si notre méthode permet de réduire cette dépendance, ce qui semble être le cas dans les cas pratiques étudiés, les calculs supplémentaires qu'elle induit, polynomiaux en d, sont raisonnables.

Une autre remarque : l'implémentation incrémentale de l'ACP permettrait une implémentation de la méthode qui garderait en mémoire pour chaque nœud les résultats de l'ACP, pour avoir moins de calcul à faire à chaque étape. Nous avons choisi de ne pas le faire pour des questions de rapport de complexité temps/espace. Notre méthode a l'avantage d'avoir une complexité constante pour un problème donné. En effet, une fois la dimension de CS fixé, la complexité en temps est constante à chaque itération, et la complexité en espace est constante aussi (en $O(d^2)$). La complexité en espace d'une implémentation qui garderait les résultats de l'ACP pour tous les nœuds de l'arbre dépendrait du nombre de nœuds nécessaires pour résoudre le problème, qu'on ne peut pas évaluer a priori, et qui peut prendre des valeurs élevées comme le montre les expériences. Ce nombre n'intervient nulle part dans notre méthode, et c'est pour nous un point positif. Ainsi, nous avons pensé qu'il était plus efficace de conserver cette implémentation où à chaque étape on effectue une nouvelle ACP.

4.5 Résultats pratiques

Les tests suivants ont été réalisés sur un PC Core2Duo 2.14 GHz avec 2 Go de RAM. L'implémentation utilise la librairie C Gnu Scientific Library [5] pour tout ce qui est calcul statistique et algèbre linéaire.

L'ACP telle qu'on l'a présentée ne modifie que l'étape de diffusion des algorithmes utilisés, on peut donc comparer les résultats qu'on obtient avec n'importe quel algorithme de diffusion. On a choisi ici de comparer dans un premier temps les RRT classiques avec les RRT modifiés par l'ACP. On ne présente toutefois qu'un seul test concernant les RRT classiques. La raison est qu'on profitait de la plateforme Kineo, qui contient l'algorithme IPP, qui est beaucoup plus rapide que RRT. Pour pouvoir faire les tests plus rapidement, on a donc utilisé IPP, qu'on compare à IPP modifié par l'ACP.

Comparaison RRT / ACP

Tout d'abord, voici une comparaison entre l'algorithme RRT traditionnel, tel qu'il est implémenté sur la plateforme KineoWorks et l'algorithme RRT où on a modifié la fonction Extend en ACP-Extend.

L'exemple utilisé est celui du cube qui doit s'extraire de deux plans parallèles proches l'un de l'autre. Le cube a six degrés de liberté : trois en rotation et trois en translation.

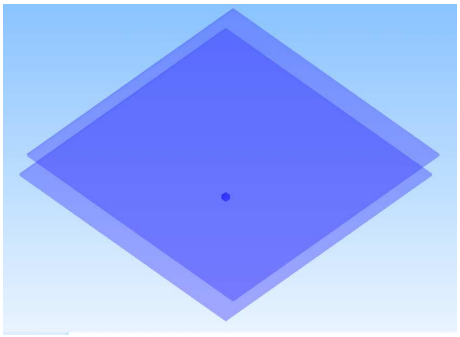

Fig. 4.5 – Comparaison RRT / ACP.

	Itérations	Nœuds	Temps
RRT	7170	3584	61 min
ACP	3321	2718	45 s

FIG. 4.6 – Résultats de la comparaison RRT / ACP

Le temps mis par les deux algorithmes n'est pas très pertinent, car l'implémentation de KineoWorks rafraichit l'interface graphique régulièrement, ce que notre implémentation ne fait pas. Dans les comparaisons suivantes, on ne l'indiquera plus.

Les deux faits notables sont :

- L'ACP a résolu le problème avec moins de nœuds que le RRT, ce qui suggère une meilleure qualité de nœuds. En d'autres termes une plus grande distance parcourue à chaque itération.
- La probabilité d'ajouter un nœud à chaque étape de l'algorithme est bien meilleure pour l'ACP (0,82) que pour le RRT (0,50).

Ces deux observations corroborent le fait que l'ACP permet d'aller plus loin à chaque étape, et de prendre des directions qui ne rencontrent pas d'obstacle immédiatement. C'est en effet lorsque q_{new} est trop proche de q_{near} que l'implémentation Kineo des RRT n'ajoute pas le nouveau nœud à l'arbre.

Comparaison IPP / ACP

Les quatre comparaisons suivantes concernent l'algorithme IPP, comparé à IPP où on a réimplémenté la fonction Extend. Lorsqu'on fait tourner l'algorithme natif de Kineo, on n'a pas accès à toutes les informations nécessaires pour une comparaison en profondeur des deux algorithmes. On indiquera donc, dans les comparaisons qui viennent :

- le nombre d'itérations de chacun des algorithmes, auquel on peut accéder même pour l'implémentation de Kineo. La comparaison de ces deux nombres tient lieu de comparaison de temps de calcul, puisqu'on a vu que le temps ajouté par notre algorithme est constant par itération (pour un problème donné).
- la probabilité de créer un nouveau noeud à chaque itération.
- la distance moyenne parcourue à chaque étape. On n'a pas accès à cette donnée quand on fait tourner l'algorithme Kineo, donc on indiquera celle qu'on a obtenue aux étapes de notre algorithme où l'on a utilisé une extension traditionnelle de l'arbre.

La figure (4.7) montre les quatre exemples sur lesquels on a comparé IPP à IPP modifié par l'ACP. De gauche à droite, et de haut en bas :

- un cube libre de tout mouvement en trois dimensions, qui doit passer par deux passages étroits où deux dimensions en translation sont libres et la troisième non. C'est donc un système à six degrés de liberté : trois en translation et trois en rotation.
- un bras (en vert) à quatre degrés de liberté qui doit éviter un obstacle en se repliant. Deux degrés sont fortement contraints et n'ont pas à bouger pour exécuter la tâche, les deux autres sont peu contraints et doivent servir à éviter l'obstacle.
- un système à dix degrés de liberté, constitué de plusieurs bras articulés. La forme de l'espace libre est plus compliquée, de nombreux obstacles limitent les mouvements des bras.

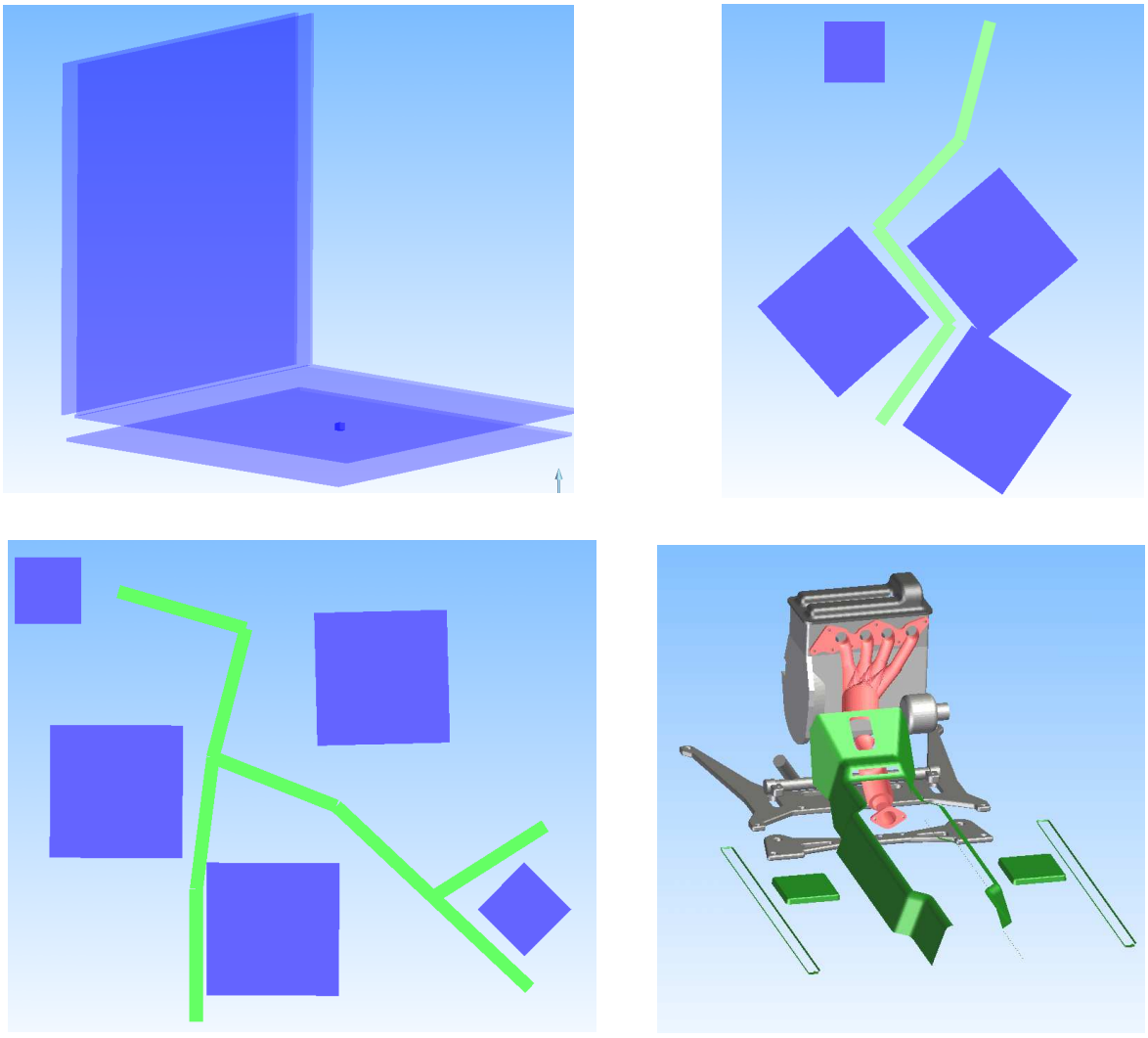

FIG. 4.7 – Comparaison IPP / ACP.

– enfin, un exemple industriel, fourni avec la plateforme Kineo. Il s'agit d'extraire un pot d'échappement (en rouge) de l'arrière d'une voiture.

Les deux premiers exemples ont été construits pour tester l'ACP. Les sous-variétés à découvrir sont linéaires, certains degrés de liberté sont bloqués. Il est attendu que notre algorithme se comporte bien sur ces exemples. Le troisième exemple correspond à un espace de configurations plus compliqué, qu'on ne saurait décrire, mais dans lequel il y a nécessairement beaucoup de contraintes et de passages étroits. Enfin le quatrième exemple n'est pas de nous, c'est un cas typique d'utilisation des algorithmes de planification de mouvement, et les résultats de notre algorithme sur ce type d'exemple permettent de le valider.

	Itérations	Probabilité de créer un nœud	Distance moyenne
Cube en trois dimensions			
IPP	2 318	0,23	13,4
ACP	324	0,62	24,4
Bras à quatre degrés de liberté			
IPP	1 867	0,18	6,67
ACP	715	0,34	13,4
Bras à dix degrés de liberté			
IPP	21 289	0,092	8,10
ACP	3 125	0,13	21,0
Pot d'échappement			
IPP	1 761	0,58	342
ACP	58	0,74	328

FIG. 4.8 – Résultats des comparaisons IPP / ACP

La première remarque est simple : dans tous ces exemples, notre algorithme se comporte mieux que IPP, il nécessite moins d'itérations. On voit que c'est dû au fait qu'on crée plus de nœuds par itération, et que les nœuds créés sont plus espacés les uns des autres.

Notons que dans l'exemple du pot d'échappement, la distance moyenne parcourue à chaque étape est légèrement plus faible avec ACP qu'avec IPP. C'est en fait une situation particulière : il s'agit d'un problème de désassemblage. Un des deux arbres grossit dans un espace très peu contraint (la situation désassemblée) dans lequel un RRT classique grossit plus vite qu'ACP comme on l'a vu précédemment. La distance affichée, qui est la moyenne des distances de tous les arbres qu'on a fait grossir dans ce problème, ne permet pas à elle seule de comparer ACP et IPP dans un espace contraint.

Il est aussi intéressant de comparer les deux exemples de bras articulés, car ce sont des situations très similaires. Un environnement très contraint, et un système qui ne présente que des degrés de liberté en rotation. C'est la situation la plus proche de la robotique humanoïde. On voit que les performances de l'ACP, rapportées à celles d'IPP, croissent avec la dimension. Le gain en rapport du nombre d'itérations en dimension 4 est de 2,6, en dimension 10 il est de 6,8.

On n'a malheureusement pas d'exemple à présenter de problème de robotique humanoïde car ce sont des simulations plus longues à mettre en place. On espère toutefois en obtenir prochainement.

Conclusion et ouvertures

L'apport de ce travail est l'application d'une méthode connue : l'analyse en composantes principales, dans un domaine où on ne l'avait jamais fait : les algorithmes de planification de mouvement par diffusion. La principale justification de l'intérêt de cette application, au-delà des résultats théoriques sur l'optimalité ou la vitesse de convergence de l'ACP, est la qualité des résultats obtenus sur des cas pratiques.

C'est la principale satisfaction à l'issue de ce travail, l'algorithme présenté est implémenté, fonctionne et est prêt à être utilisé sur de nombreux problèmes de planification de mouvement sur lesquels on a vérifié qu'il donnait de bons résultats.

Le travail de test et de comparaison avec les algorithmes existants doit être prolongé, en particulier par des expériences sur des systèmes antropomorphes. C'était un des buts poursuivis en étudiant des méthodes de réduction de dimension. De plus, du travail pourrait être fait pour étudier d'autres méthodes de réduction de dimension que l'ACP, et déterminer les méthodes les mieux adaptées aux problèmes de planification de mouvement.

Bibliographie

[1] J. Cortes and T. Simeon. Disassembly path planning for objects with articulated parts. In *IFAC International Workshop on Intelligent Assembly and Disassembly (IAD'07)*, pages 34–39, Alicante (Espagne), 2007.

[2] Deniz Erdogmus, Yadunandana N. Rao, Hemanth Peddaneni, Anant Hegde, and Jose C. Principe. Recursive principal components analysis using eigenvector matrix perturbation. *EURASIP Journal on Applied Signal Processing*, 2004(13) :2034–2041, 2004. doi :10.1155/S1110865704404120.

[3] C. Esteves Jaramillo. *Motion planning : from digital actors to humanoid robots*. PhD thesis, Institut National Polytechnique, Toulouse, 100p., 2007. Doctorat.

[4] E. Ferre and J.P. Laumond. An iterative diffusion algorithm for part disassembly. In *2004 International Conference on Robotics and Automation (ICRA'2004)*, pages 3149–3154, New Orleans (USA), 2004.

[5] Gnu scientific library. http ://www.gnu.org/software/gsl/.

[6] D. Hsu. *Randomized Single-query Motion Planning in Expansive Spaces*. PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, May 2000.

[7] D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. Computational Geometry & Applications*, 9(4-5) :495–512, 1999.

[8] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[9] J. Kuffner and S. LaValle. RRT-connect : An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'2000), San Francisco, CA*, April 2000.

[10] J. Laumond and T. Simeon. Notes on visibility roadmaps and path planning, 2000.

[11] S. M. LaValle. Rapidly-exploring random trees : A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, October 1998.

[12] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http ://planning.cs.uiuc.edu/.

[13] Tomas Lozano-Perez. Spatial planning : A configuration space approach. *IEEE Transactions on Computers*, 32(2) :108–120, 1983.

[14] Sam T. Roweis and Lawrence K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500) :2323–2326, 2000.

[15] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500) :2319–2323, 2000.

[16] Miguel L. Teodoro, Jr. George N. Phillips, and Lydia E. Kavraki. A dimensionality reduction approach to modeling protein flexibility. In *RECOMB '02 : Proceedings of the sixth annual international conference on Computational biology*, pages 299–308, New York, NY, USA, 2002. ACM Press.

[17] L. Zwald. *Performances statistiques d'algorithmes d'apprentissage : « kernel projection machine » et analyse en composantes principales à noyau*. PhD thesis, Université Paris Sud - Paris XI, 2005.

[18] Laurent Zwald and Gilles Blanchard. On the convergence of eigenspaces in kernel principal component analysis. In *NIPS*, 2005.