Modélisation et Planification d'Actions Elémentaires Robotiques par Apprentissage de Réseaux de Contraintes

Mathias Paulin, Eric Bourreau, Christopher Dartnell, Sébastien Krut

Modélisation et planification d'actions élémentaires robotiques par apprentissage de réseaux de contraintes.

Article Jeune Chercheur

Mathias PAULIN[1], Eric BOURREAU[1],
Christopher DARTNELL[1], Sébastien KRUT[2].
LIRMM UMR 5506 Université Montpellier II - CNRS
[1] Département d'Informatique
[2] Département de Robotique
161 rue Ada 34392 Montpellier cedex 5 - France
{paulin | bourreau | dartnell | krut}@lirmm.fr

Résumé

Depuis près de dix ans, les progrès réalisés dans le domaine de la robotique sont colossaux et les liens avec l'Intelligence Artificielle se font de plus en plus étroits. Malgré ces progrès, les chercheurs et les ingénieurs de robotique se heurtent toujours à une difficulté de taille chaque fois qu'ils conçoivent un nouveau robot. Étant donné un robot capable d'effectuer un certain nombre d'actions élémentaires, la difficulté consiste à combiner ces différentes actions élémentaires pour que le robot réalise des tâches de plus haut niveau. À l'heure actuelle, les roboticiens modélisent les actions élementaires sous la forme de systèmes d'équations mathématiques complexes qu'il est ensuite très difficile de combiner entre eux pour planifier une tâche de haut niveau. Dans cet article, nous proposons d'expérimenter une approche CSP au processus de planification. Celle-ci consiste à modéliser les actions élémentaires sous la forme de réseaux de contraintes acquis par apprentissage, puis à déterminer des séquences d'actions de plus haut niveau en combinant les différents réseaux acquis à l'aide d'outils de planification de tâches. Ensuite, nous décrivons comment valider notre approche au travers de l'étude expérimentale d'un robot sauteur unijambiste baptisé Twig.

Abstract

In recent years, robotic, and more precisely humanoid robots, have made great improvements, especially in some close links with Artificial Intelligence. However, each time you have to design a new robot, it still exist a specific difficulty. You have, with some elementary actions, to combine them in a high-level vision to describe some complex tasks. In this article, we propose to interact with roboticians in the planning process. Our approach is based on elementary actions automatically modelled with constraint networks by Machine Learning. To describe high-level tasks we only have to sequence multiple networks, using planning tools. To validate this theoretical framework, we describe how we will experiment it on a one-leg jumping robot named Twig.

1 Introduction

Depuis près de dix ans, les progrès réalisés dans le domaine de la robotique sont colossaux et les liens avec l'Intelligence Artificielle se font de plus en plus étroits. Ainsi a-t-on vu la commercialisation du chien Aibo par Sony [1], et la réalisation de robots humanoïdes très aboutis, tels que les robots Asimo de Honda [3] ou HRP-2 de Kawada [8]. Les robots actuels disposent ainsi de larges panels d'actions élémentaires câblées[1] qui sont exécutées de manière automatique. La difficulté consiste cependant à combiner puis enchaîner ces différentes actions élémentaires dans le but de réa-

1. Pour les robots humanoïdes, les actions élémentaires peuvent être par exemple "*Fermer la main*", "*Tendre le bras*", etc.

liser une tâche plus complexe. À l'heure actuelle en robotique, chaque action élémentaire est modélisée sous la forme de systèmes d'équations mathématiques mettant en jeu des lois physiques particulièrement complexes.[2] Déterminer une loi de commande qui combine plusieurs actions élémentaires dans le but de réaliser une tâche plus complexe se révèle être particulièrement difficile et nécessite de nombreuses heures de calculs. Ainsi, certaines marches de robots humanoïdes se réduisent à dérouler automatiquement une séquence d'actions élémentaires, calculée à l'avance et limitée à des plages de fonctionnement parfaitement sécurisées afin de limiter les risques de chute dont le coût financier est parfois très important.

Dans cet article, nous proposons une approche totalement innovante qui utilise le paradigme de la programmation par contraintes et les méthodes de planification pour établir rapidement et facilement des séquences d'actions élémentaires d'un robot. Notre approche consiste à modéliser les actions élémentaires à l'aide de réseaux de contraintes (CSP). Afin d'automatiser le processus de modélisation, nous utilisons la plateforme d'apprentissage de réseaux de contraintes CONACQ [4], [5] pour acquérir automatiquement les CSP. Cette modélisation sous forme de réseaux a le double avantage d'abstraire un modèle déclaratif des actions élémentaires et d'avoir de bonnes propriétés computationnelles, capitales pour la planification. Ainsi, pour déterminer une séquence d'actions élémentaires permettant de réaliser une tâche plus complexe, nous comptons combiner les différents CSP précédemment acquis en utilisant un planificateur de tâches de type GRAPHPLAN[3] [7]. Grâce à notre approche, nous espérons simplifier le travail des roboticiens ainsi que contribuer à la constitution automatique de lois de commande.

Le reste de cet article est organisé comme suit. La section 2 rappelle quelques définitions préliminaires sur les réseaux de contraintes, l'apprentissage automatique de CSP ainsi que le problème de planification de tâches. Dans la section 3, nous présentons et justifions en détails notre approche, que nous validons partiellement ensuite dans la section 4 au travers de l'étude expérimentale d'un robot sauteur unijambiste baptisé TWIG. Avant de conclure (section 6), la section 5 décrit les perspectives de travail que nous comptons mettre en oeuvre afin d'enrichir notre approche.

2. Loi de conservation des énergies, moments cinétiques, équations aux dérivées partielles, etc.

3. ou tout autre solveur de planification compatible avec des CSP, CPlan[15] par exemple.

2 Préliminaires

Dans cette section, nous introduisons les concepts utilisés dans l'article. Nous rappelons quelques définitions de la programmation par contraintes avant de présenter succinctement l'apprentissage automatique de réseaux de contraintes puis le problème de planification de tâches.

2.1 Rappel sur la programmation par contraintes

Définition 1 (Réseau de Contraintes) *Un réseau de contraintes $\mathcal{P}$ est un triplet $(\mathcal{X},\mathcal{D},\mathcal{C})$ où :*

- *$\mathcal{X} = \{x_1,...,x_n\}$ est un ensemble fini de n variables,*
- *$\mathcal{D} = \{D(x_1),\ldots,D(x_n)\}$ est l'ensemble de leurs domaines respectifs,*
- *$\mathcal{C} = \{c_1,...,c_m\}$ est une séquence de contraintes sur $\mathcal{X}$. Une contrainte c_i est définie par la séquence $var(c_i)$ des variables sur lesquelles elle porte, et par la relation $sol(c_i)$ qui spécifie les n-uplets autorisés sur $var(c_i)$.*

L'affectation de valeurs aux variables de $var(c_i)$ satisfait c_i si elle appartient à $sol(c_i)$. Une instance e sur $\mathcal{X}$ est un n-uplet $(v_1,...,v_n) \in D(x_1) \times ... \times D(x_n)$. On dit qu'une instance est une solution du réseau si elle satisfait toutes les contraintes du réseau. Sinon, c'est une non solution. On note $Sol(\mathcal{X},\mathcal{D},\mathcal{C})$ l'ensemble des solutions de $(\mathcal{X},\mathcal{D},\mathcal{C})$.

2.2 Apprentissage de réseau de contraintes

L'apprentissage de réseau de contraintes a pour but d'acquérir automatiquement un modèle à partir d'instances correspondant à des solutions et des non solutions au problème que l'on souhaite modéliser sous la forme d'un réseau de contraintes [4], [5], [9]. Comme point de départ, nous connaissons l'ensemble $\mathcal{X}$ des variables du problème, leur domaine $\mathcal{D}$ de valeurs possibles, et nous disposons de E^+, un sous-ensemble des solutions du problème, et E^- un ensemble de non solutions. L'objectif de l'apprentissage consiste à modéliser le problème dans un solveur de contraintes. Notre biais d'apprentissage noté $\mathcal{B}$ est en conséquence une librairie de contraintes issue de ce solveur.

On dit qu'une séquence de contraintes appartient au biais si et seulement si cette séquence n'utilise que des contraintes de la librairie du biais. Apprendre un réseau de contraintes consiste à rechercher une séquence de contraintes $\mathcal{C}$ appartenant à un biais d'apprentissage $\mathcal{B}$ donné, et dont l'ensemble des solutions est un sur-ensemble de E^+ ne contenant aucun élément de

E^-. Le Problème d'Acquisition de Contraintes se définit alors formellement comme suit :

Définition 2 (Pb. d'Acquisition de Contraintes) *Étant donnés un ensemble de variables $\mathcal{X}$, leur domaine $\mathcal{D}$, deux ensembles E^+ et E^- d'instances sur $\mathcal{X}$, et un biais d'apprentissage $\mathcal{B}$, le problème d'acquisition de contraintes consiste à trouver une séquence de contraintes $\mathcal{C}$ telle que :*

$$\begin{cases} \mathcal{C} \in \mathcal{B}, \\ \forall e^- \in E^-, e^- \text{ n'est pas solution de } (\mathcal{X},\mathcal{D},\mathcal{C}), \text{ et,} \\ \forall e^+ \in E^+, e^+ \text{ est solution de } (\mathcal{X},\mathcal{D},\mathcal{C}). \end{cases}$$

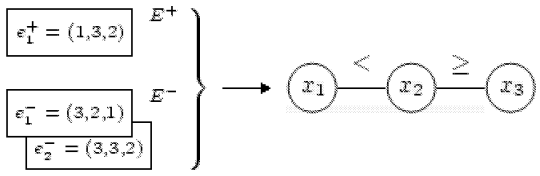

FIG. 1 – *Le principe d'apprentissage automatique de contraintes, illustré par un exemple.*

La figure 1 reprend l'exemple développé dans [10] afin d'illustrer le principe d'apprentissage automatique de contraintes. Les variables du problèmes sont $\mathcal{X} = \{x_1,x_2,x_3\}$, $D(x_i) = [1,2,3,4] \forall x_i \in \mathcal{X}$, et on utilise le biais d'apprentissage $\mathcal{B} = (\{x_1,x_2\},L),(\{x_2,x_3\},L))$ où $L = \{<, \leq, =, \geq, >, \neq\}$ pour acquérir un CSP à partir d'une instance positive et de deux instances négatives.

2.3 Le problème de planification de tâches

La planification[4] consiste à établir, avant leur exécution, un ordre d'application d'actions afin d'atteindre un objectif défini dans un environnement particulier appellé Monde. Le formalisme STRIPS communément utilisé pour exprimer des problèmes de planification de tâches utilise les concepts d'État, de But et d'Action définis de la manière suivante :

- Un ***État*** est une conjonction de littéraux positifs décrivant le Monde à un instant donné.
- Un ***But*** est un état particulier qui décrit de manière partielle le Monde dans un état final. On dit alors qu'un état s ***satisfait*** un but b si s contient tous les littéraux de b.
- Une ***Action*** A est caractérisée par son nom, ses paramètres, sa *pré-condition* et son *effet*. La ***pré-condition*** de A est une conjonction propositionnelle qui représente l'ensemble des conditions qui doivent être vérifiées pour que A puisse être exécutée. L'***effet*** de A est une conjonction propositionnelle qui décrit les changements qui surviennent lorsque A est exécutée.

Définition 3 (Problème de planification) *Étant donné E_i un état initial correspondant à une description complète du Monde, E_f un état final correspondant à un but, et $\mathcal{A}$ un ensemble d'actions, le problème de planification de tâches consiste à trouver une séquence d'actions issues de $\mathcal{A}$ telle que si ces actions sont exécutées à partir de E_i, elles permettent d'aboutir à un état qui satisfait E_f.*

Il existe de nombreux algorithmes permettant de résoudre le problème de planification. Il convient cependant de noter que les algorithmes les plus performants à l'heure actuelle sont pour la plupart des descendants de GRAPHPLAN [7].

3 Architecture proposée

L'approche que nous défendons dans cet article a pour but de déterminer rapidement et simplement des lois de commande pour un robot. Une loi de commande est une séquence d'actions élémentaires permettant de réaliser une tâche qui ne peut être accomplie au moyen d'une unique action élémentaire. Dans cette section, nous nous attachons à démontrer en quoi l'emploi des réseaux de contraintes et surtout l'utilisation que nous en avons constitue une bonne alternative à la démarche actuellement utilisée en robotique. De plus, nous montrons que notre démarche offre des perspectives intéressantes pour l'acquisition automatique de nouvelles lois de commande.

3.1 Principe général

Étant donné un robot capable d'effectuer un panel $\mathcal{A} = \{\mathcal{A}_1,..,\mathcal{A}_m\}$ d'actions élémentaires, et étant donnée une tâche ne pouvant être accomplie en une seule action élémentaire, le problème fourni par les roboticiens consiste à combiner différentes actions élémentaires afin que le robot puisse accomplir la tâche demandée. Si une telle combinaison d'actions existe, on parle alors de loi de commande.

À l'heure actuelle, pour exprimer les lois de commande des robots, les roboticiens s'attachent à modéliser au plus près chaque action élémentaire en utilisant les lois physiques. Pour cela, ils font intervenir des lois physiques particulièrement complexes, telles que la loi de conservation des énergies, des équations aux

4. Dans cette section, nous présentons le problème de planification dans sa définition la plus commune telle que définie dans [2].

dérivées partielles, des théorèmes liés aux moments cinétiques, etc. En procédant ainsi, chaque action élémentaire s'exprime à l'aide de systèmes d'équations mathématiques caractérisant les capacités mécaniques du robot relativement aux lois du monde. Ces systèmes d'équations sont par leur définition très complexes, et composer une séquence valide faisant intervenir plusieurs actions élémentaires se révèle très difficile.

Depuis plusieurs années, la programmation par contraintes connaît un succès croissant et est largement utilisée dans de nombreuses applications industrielles. La principale raison de ce succès réside dans son aspect déclaratif et dans la dissociation inhérente entre un modèle décrivant le problème (le "Quoi") et les techniques de résolution utilisées pour le résoudre (le "Comment"). Comme nous l'avons présenté précédemment, les lois de commande des robots actuels se révèlent être modélisées de manière complexe et difficilement résolvables. Nous avons donc pensé à modéliser ces lois de commande à l'aide de réseaux de contraintes simples ayant de bonnes propriétés computationnelles, puis à combiner et composer ces CSP pour établir les lois de commande du robot. [5]

L'objectif de notre travail consistant à proposer une alternative aux chercheurs en robotique qui, sauf exceptions, ne sont pas des experts en programmation par contraintes, nous avons décidé de modéliser les actions élémentaires d'un robot par apprentissage automatique à l'aide de la plateforme CONACQ [4] [5]. Pour modéliser automatiquement chaque action élémentaire $\mathcal{A}_i$, les roboticiens n'auront qu'à fournir un ensemble d'instances valides de $\mathcal{A}_i$, ainsi qu'un ensemble d'instances ne correspondant pas à $\mathcal{A}_i$. CONACQ exprimera alors $\mathcal{A}_i$ dans le formalisme STRIPS (*c.f.* section 2.3) à l'aide de $\mathcal{P}_p$ et $\mathcal{P}_e$, deux réseaux de contraintes modélisant respectivement la précondition de $\mathcal{A}_i$ et les effets de $\mathcal{A}_i$, ainsi qu'un troisième réseau $\mathcal{P}_a$ exprimant comment les différents actionneurs [6] du robot doivent réagir pour effectuer $\mathcal{A}_i$.

Comme l'illustre la figure 2, l'approche que nous proposons consiste à exprimer chaque action élémentaire $\mathcal{A}_i$ d'un robot dans le formalisme STRIPS à l'aide de réseaux de contraintes. Pour exprimer une loi de commande permettant d'accomplir une tâche non élémentaire T, nous faisons alors appel à un planificateur de tâches afin de trouver un plan permettant d'accomplir T, qui est alors exprimée à l'aide du couple d'états

5. Notre objectif étant de répondre à des impératifs de simplicité et de rapidité, nous utiliserons un jeu très simplifié de contraintes répondant à la problématique de la planification et non à des équations mathématiques complexes.

6. En robotique, les différents moteurs du robot sont communément appellés actionneurs.

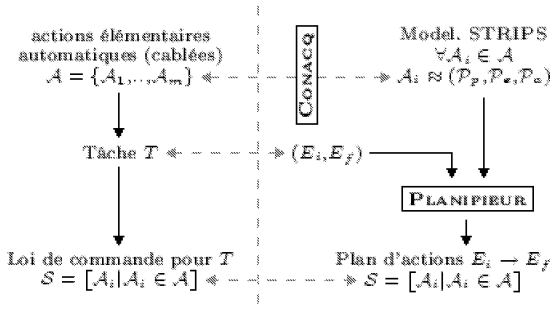

FIG. 2 – *Principe général de notre approche.*

(E_i,E_f). Si un plan est trouvé, la séquence d'actions élémentaires $\mathcal{S} = [\mathcal{A}_i|\mathcal{A}_i \in \mathcal{A}]$ correspondra alors à la loi de commande que devra exécuter le robot pour accomplir T.

3.2 Architecture détaillée

3.2.1 Modélisation par réseaux de contraintes

Nous présentons ici comment caractériser chaque action élémentaire $\mathcal{A}_i$ à l'aide de trois réseaux de contraintes modélisant respectivement la *pré-condition* de $\mathcal{A}_i$, son *effet*, ainsi que la manière dont doivent réagir les différents actionneurs du robot pour réaliser $\mathcal{A}_i$.

L'approche que nous proposons dans cet article se veut générique. Ainsi, nous étudions les robots dans leur définition la plus générale : Un robot est un système mécanique poly-articulé, mû par des actionneurs, commandé par un calculateur, et qui est destiné à effectuer une grande variété de tâches. Notre approche consiste à modéliser facilement les lois de commande que le calculateur d'un robot devra manipuler pour commander les actionneurs afin de mener à bien une tâche pour laquelle il est dédié. Qu'il s'agisse de robots mobiles à roues [6], de robots sous-marins [13] [14], ou bien encore de robots humanoïdes [8], un robot évolue toujours dans un environnement déterminé. Pour décrire complètement l'évolution d'un robot, un ensemble de descripteurs de l'état du robot et de l'environnement est nécessaire. Ces descripteurs sont en fait des capteurs qui décrivent différentes *dimensions* du monde, et peuvent être aussi divers qu'un capteur de pression, un accéléromètre (généralement placé sur le robot), la position d'un point particulier du robot ou de l'environnement, [7] etc.

7. Un capteur indiquant la position du centre de gravité du robot est ainsi généralement utilisé.

Soit un robot R tel que $\alpha = \{a_1,...,a_k\}$ est l'ensemble des variables qui modélisent les actionneurs de R, et tel que $\delta = \{d_1,...,d_n\}$ est l'ensemble des descripteurs de R et de son environemment. Étants donnés $\mathcal{A}_i$ une action élémentaire de R et $\mathcal{L}$ une librairie de contraintes, modéliser $\mathcal{A}_i$ à l'aide de réseaux de contraintes sous le formalisme STRIPS revient à modéliser trois CSP $\mathcal{P}_p$, $\mathcal{P}_a$ et $\mathcal{P}_e$ respectant la définition 4.

Définition 4 (Modélisation par contraintes) *Soient R un robot, α l'ensemble des actionneurs de R et δ l'ensemble des descripteurs de R et de son environemment, $\mathcal{A}_i$ une action élémentaire de R et $\mathcal{L}$ une librairie de contraintes. Étant donnés $\mathcal{P}_p = (X_p,\mathcal{D}_p,\mathcal{C}_p)$, $\mathcal{P}_a = (X_a,\mathcal{D}_a,\mathcal{C}_a)$ et $\mathcal{P}_e = (X_e,\mathcal{D}_e,\mathcal{C}_e)$ trois réseaux de contraintes tels que $X_p \subseteq \delta$, $X_a \subseteq \alpha$, $X_e \subseteq \delta$, $\mathcal{C}_p \subseteq \mathcal{L}$, $\mathcal{C}_a \subseteq \mathcal{L}$, $\mathcal{C}_e \subseteq \mathcal{L}$, et tels que $\mathcal{D}_p$ (resp. $\mathcal{D}_a$, $\mathcal{D}_e$) est l'ensemble des domaines des variables de X_p (resp. X_a, X_e).*

Le triplet $(\mathcal{P}_p,\mathcal{P}_a,\mathcal{P}_e)$ modélise correctement $\mathcal{A}_i$ dans l'optique d'une planification si et seulement si :

$$\begin{cases} \mathcal{P}_p \text{ exprime les conditions dans lesquelles } \mathcal{A}_i \text{ peut être exécutée,} \\ \mathcal{P}_a \text{ modélise le comportement des actionneurs lors de l'exécution de } \mathcal{A}_i, \\ \mathcal{P}_e \text{ exprime l'état de } R \text{ et de son environnement après l'exécution de } \mathcal{A}_i. \end{cases}$$

Il est à noter que cette modélisation sous forme de réseaux de contraintes permet d'acquérir, pour chaque action élémentaire $\mathcal{A}_i$, un modèle qui abstrait un ensemble d'instances possibles pour $\mathcal{A}_i$. Il s'agit donc d'un *espace* de solutions possibles (et non une solution unique) qui devrait pouvoir permettre de combiner plus facilement plusieurs actions élémentaires entre elles.

3.2.2 Modélisation par apprentissage

Afin de modéliser *automatiquement* une action élémentaire $\mathcal{A}_i$, nous avons choisi d'utiliser la plateforme d'apprentissage automatique de réseaux de contraintes CONACQ [4] [5]. Pour cela, nous devons disposer d'une librairie $\mathcal{L}$ de contraintes, appellé biais d'apprentissage, et de deux ensembles disjoints E^+ et E^- contenant respectivement différentes instances valides de $\mathcal{A}_i$ et des instances ne correspondant pas à $\mathcal{A}_i$. Ce jeu de données sera fourni simplement par les roboticiens. À partir de ces données, CONACQ modélisera un réseau de contraintes $\mathcal{P}$ répondant au problème d'acquisition de contraintes défini en section 2.2 mais aussi décomposable tel que décrit dans la définition 4.

Pour que le réseau de contraintes acquis par CONACQ caractérise complètement l'action élémentaire $\mathcal{A}_i$, il convient que les instances d'entraînement de E^+ et E^- rendent pleinement compte de l'évolution de l'état du robot et de l'environnement durant $\mathcal{A}_i$. Si $\alpha = \{a_1,..,a_k\}$ est l'ensemble des actionneurs du robot, et $\delta = \{d_1,..,d_n\}$ est l'ensemble des descripteurs du robot et de l'environnement, alors les instances d'entraînement doivent être de la forme $(d^i_1,..,d^i_n, a_1,..,a_k,d^f_1,..,d^f_n)$ où d^i_j $\forall j \in [1..n]$ (*resp.* d^f_j) correspond à l'état du descripteur d_j au début (*resp.* à la fin) de l'action élémentaire, et a_j correspond à la valeur prise par l'actionneur a_j durant $\mathcal{A}_i$. Pour modéliser $\mathcal{A}_i$ en respectant la définition 4, nous allons chercher à acquérir un réseau de contraintes en plaçant un biais d'apprentissage portant sur les actionneurs $\{a_1,..,a_k\}$, un autre sur l'état des descripteurs $\{d^i_1,..,d^i_n\}$ et un dernier placé entre $\{d^i_1,..,d^i_n\}$ et $\{d^f_1,..,d^f_n\}$. Ce dernier biais permet de caractériser les changements opérés durant $\mathcal{A}_i$.

À partir du biais d'apprentissage précédemment mentionné, de E^+ et E^-, CONACQ modélise l'action élémentaire $\mathcal{A}_i$ au moyen d'un réseau de contraintes $\mathcal{P}$. Il est alors immédiat de remarquer que le sous réseau de $\mathcal{P}$ issu du biais impliquant les descripteurs $\{d^i_1,..,d^i_n\}$ exprime la *pré-condition* de $\mathcal{A}_i$, et correspond donc à $\mathcal{P}_p$. De manière analogue, le sous réseau de $\mathcal{P}$ issu du biais plaçé entre $\{d^i_1,..,d^i_n\}$ et $\{d^f_1,..,d^f_n\}$ exprime les *effets* de $\mathcal{A}_i$, et modélise par conséquent $\mathcal{P}_e$. Enfin, $\mathcal{P}_a$ est modélisé à l'aide du sous réseau de contraintes issu de $\mathcal{P}$ impliquant les actionneurs $\{a_1,..,a_k\}$ du robot.

3.2.3 Planification de tâches

Soit $\mathcal{A} = \{\mathcal{A}_1,..,\mathcal{A}_m\}$ l'ensemble des m actions élémentaires que peut réaliser un robot. L'apprentissage automatique permet de modéliser chaque action élémentaire $\mathcal{A}_i \in \mathcal{A}$ sous la forme de trois réseaux de contraintes $(\mathcal{P}^i_p,\mathcal{P}^i_e,\mathcal{P}^i_a)$. Il nous reste désormais à réussir à combiner ces différentes actions élémentaires afin d'établir une séquence $\mathcal{S}$ d'actions élémentaires que le robot devra effectuer afin d'accomplir une tâche T qui ne peut être accomplie en une seule action élémentaire.

Pour chaque action élémentaire $\mathcal{A}_i \in \mathcal{A}$, restreignons la modélisation produite par CONACQ au couple $(\mathcal{P}^i_p,\mathcal{P}^i_e)$, et exprimons la tâche T à accomplir dans le formalisme STRIPS au moyen de l'état initial E_i et de l'état final E_f (*i.e.* le but de T) tels que définis dans la section 2.3.

Exprimé de la sorte, le problème de départ consistant à trouver une séquence d'actions élémentaires per-

mettant de réaliser T revient désormais à résoudre le problème de planification de tâches (E_i,E_f,$\mathcal{A}$). Pour résoudre ce problème de planification, nous faisons alors appel à un planificateur tel que GRAPHPLAN que nous utilisons comme une boîte noire. Il convient cependant de noter d'une part que la modélisation des préconditions et des effets des actions de $\mathcal{A}$ sous la forme de réseaux de contraintes nous assure des propriétés de propagation fortes et nous permet d'envisager un temps de réponse limité de l'algorithme de planification. Dans un premier temps, cette planification est envisagée offline. Par la suite, des tentatives de planification temps réel restent envisageables (sous conditions). D'autre part, il convient de remarquer que l'on exprimera une relation d'exclusion mutuelle (ou mutex) de la manière suivante. Soient $\mathcal{A}_i$ et $\mathcal{A}_j$ deux actions élémentaires, et soient les réseaux de contraintes $\mathcal{P}_1$, $\mathcal{P}_2$ et $\mathcal{P}_3$ tels que $\mathcal{P}_1 = \mathcal{P}_e^i \cup \mathcal{P}_p^j$, $\mathcal{P}_2 = \mathcal{P}_e^i \cup \mathcal{P}_e^j$, et $\mathcal{P}_3 = \mathcal{P}_p^i \cup \mathcal{P}_p^j$. $\mathcal{A}_i$ et $\mathcal{A}_j$ seront mutex si au moins l'un des trois réseaux $\mathcal{P}_1$, $\mathcal{P}_2$ ou $\mathcal{P}_3$ n'a pas de solution.[8]

Si il existe un plan permettant d'atteindre E_f à partir de E_i, alors le planificateur renverra une séquence $\mathcal{S} = [\mathcal{A}_i | \mathcal{A}_i \in \mathcal{A}]$ qui renfermera les différentes actions élémentaires $\mathcal{A}_i$ à exécuter. Ainsi, pour accomplir la tâche T, il suffira alors au robot d'effectuer la séquence d'actions $\mathcal{S}$. Il convient de noter que les séquences ainsi produites correspondent bien à des lois de commande, au sens roboticien du terme, puisque pour chaque action élémentaire $\mathcal{A}_i$ à effectuer, le réseau $\mathcal{P}_a^i$ correspondant nous indique les valeurs qui doivent être appliquées sur les différents actionneurs du robot pour que ce dernier effectue $\mathcal{A}_i$. De plus, pour une tâche T donnée, le planificateur peut renvoyer plusieurs séquences d'actions élémentaires permettant d'accomplir T.

L'abstraction des actions élémentaires sous la forme de réseaux de contraintes modélisés par apprentissage automatique, puis l'utilisation des outils de planification de tâches nous permet d'établir facilement, rapidement et automatiquement des lois de commande pour des robots poly-articulés. De plus, contrairement aux démarches habituellement utilisées en robotique, la modélisation en CSP nous garantie de très bonnes propriétés computationnelles qui pourraient permettre d'envisager une planification en temps réel, qui rendrait les robots autonomes.

8. Nous suivons en cela la définition des relations d'exclusion mutuelle de GRAPHPLAN.

3.3 Exemple théorique

Dans cette section, nous illustrons notre démarche au travers de l'étude théorique d'un robot mobile à roues schématiquement représenté sur la figure 3. Notre robot est capable d'accomplir deux actions élémentaires : $\mathcal{A}_{av}$ permet d'avancer de deux mètres vers l'avant le long de l'axe $\overrightarrow{Ox}$, et $\mathcal{A}_{ar}$ permet de reculer d'un mètre selon $\overrightarrow{Ox}$. Dans cette illustration théorique, nous cherchons à exprimer la loi de commande permettant de se déplacer vers l'avant de trois mètres à partir d'une position x_0.

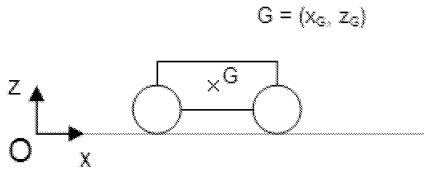

FIG. 3 – *Le robot utilisé pour l'illustration théorique.*

Notre robot est muni de deux actionneurs : un moteur électrique sur lequel on peut appliquer une brève tension U_M et un inverseur de poussée Inv dont la valeur doit être mise à 0 pour avancer, 1 pour reculer. Deux descripteurs permettent de connaître l'état du monde et du robot à tout instant t : x_G^t donne la position de l'abscisse du centre de gravité du robot dans un repère bi-dimensionnel (O,x,z), et s^t est un booléen permettant d'indiquer si les roues touchent le sol.

Pour réaliser $\mathcal{A}_{av}$, il faut que les roues touchent le sol et il faut appliquer une tension de 20 volts sur le moteur du robot, ce qui a pour effet de faire avancer le robot de deux mètres vers l'avant. Pour pouvoir réaliser l'action élémentaire $\mathcal{A}_{ar}$, les roues doivent toucher le sol et il faut appliquer une tension de 10 volts sur le moteur, tout en fixant la valeur de l'inverseur à 1. Le robot recule alors d'un mètre.

Pour chacune des deux actions élémentaires $\mathcal{A}_{av}$ et $\mathcal{A}_{ar}$, nous fournissons à CONACQ des exemples d'instances valides et non valides de $\mathcal{A}_{av}$ (*resp.* $\mathcal{A}_{ar}$). Ces instances sont exprimées sous la forme de tuples étiquetés[9] (s^i,x_G^i,U_M,Inv,s^f,x_G^f). L'apprentissage automatique de CSP permet alors de modéliser les deux actions élémentaires $\mathcal{A}_{av}$ et $\mathcal{A}_{ar}$ à l'aide des réseaux de contraintes suivants :

9. Un tuple est étiqueté positivement si il caractérise une instance valide, négativement sinon.

$$\mathcal{A}_{Av} = \begin{cases} \mathcal{P}_p = \{(s^i = 1)\} \\ \mathcal{P}_a = \{(U_M = 20,\, Inv = 0)\} \\ \mathcal{P}_e = \{(s^f = s^i,\, x_R^f = x_R^i + 2)\} \end{cases}$$

$$\mathcal{A}_{Ar} = \begin{cases} \mathcal{P}_p = \{(s^i = 1)\} \\ \mathcal{P}_a = \{(U_M = 10,\, Inv = 1)\} \\ \mathcal{P}_e = \{(s^f = s^i,\, x_R^f = x_R^i - 1)\} \end{cases}$$

On cherche désormais à établir la loi de commande permettant au robot de se déplacer de trois mètres vers l'avant en partant de l'origine du repère. Dans le formalisme STRIPS, cette tâche s'exprime à l'aide de l'état initial $E_i = \{s^i = 1, x_G^i = 0\}$ et du but $E_f = \{s^f = 1, x_G^f = 3\}$. Comme le montre la figure 4, il existe trois plans minimaux en nombre d'actions permettant d'atteindre E_f à partir de E_i au moyen des actions élémentaires issues de $\mathcal{A} = \{\mathcal{A}_{av}, \mathcal{A}_{ar}\}$.

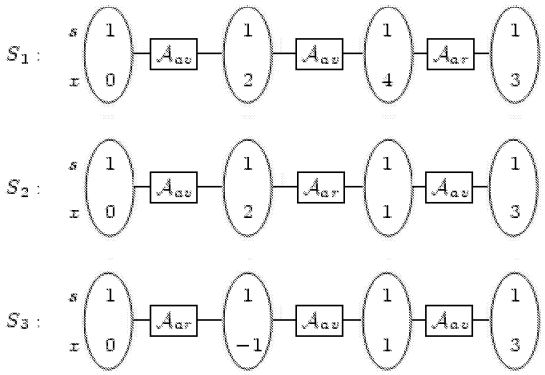

FIG. 4 – *Trois plans possibles pour* $(E_i, E_f, \mathcal{A})$.

La réponse au problème de planification $(E_i, E_f, \mathcal{A})$ renvoyée par GRAPHPLAN sera la donnée des séquences d'actions élémentaires $\mathcal{S}_1 = [\mathcal{A}_{av}, \mathcal{A}_{av}, \mathcal{A}_{ar}]$, $\mathcal{S}_2 = [\mathcal{A}_{av}, \mathcal{A}_{ar}, \mathcal{A}_{av}]$ et $\mathcal{S}_3 = [\mathcal{A}_{ar}, \mathcal{A}_{av}, \mathcal{A}_{av}]$. $\mathcal{S}_1$, $\mathcal{S}_2$ et $\mathcal{S}_3$ expriment bien trois lois de commandes distinctes générées automatiquement permettant de réaliser une même tâche.

4 Expérimentation

Dans le cadre d'une collaboration transversale entre roboticiens et informaticiens au sein du Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), nous avons réalisé une première expérimentation afin de valider empiriquement l'approche que nous présentons dans cet article. Nous avons pour cela chercher à modéliser les mouvements élémentaires d'un robot sauteur unijambiste nommé TWIG.

4.1 Le robot Twig

Pour cette première expérimentation, nous avons souhaité étudier un robot mobile ayant une architecture minimale mais dont les lois de commande sont difficiles à établir malgré sa mécanique simple. Afin de répondre à ces impératifs, le choix des roboticiens s'est porté sur l'étude d'un *vieux* robot unijambiste baptisé TWIG défini dans [11]. Comme le montre la figure 5, TWIG est constitué d'un volant d'inertie, d'un ressort hélicoïdal et d'un vérin hydraulique. Le volant d'inertie permet à TWIG de rester dans un état stable[10] (*i.e.* en position verticale) et lui permet aussi de se pencher dans une direction. Le verin hydraulique sert à effectuer une poussée longitudinale sur la jambe, alors que le ressort est utilisé pour amortir la réception d'un saut.

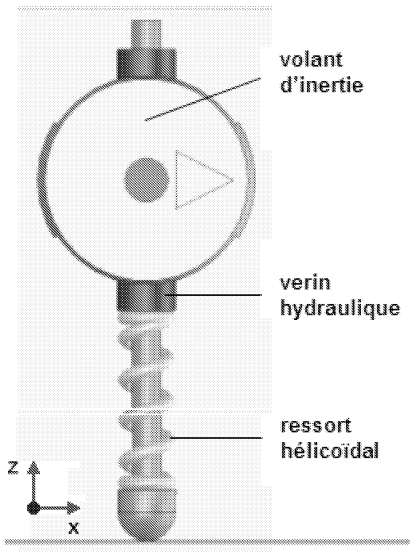

FIG. 5 – *Le robot* TWIG.

Nous étudions TWIG dans l'espace bidimensionnel (xOz) dans lequel il est capable d'effectuer les actions élémentaires suivantes : Sauter verticalement selon l'axe $\overrightarrow{Oz}$, atterrir verticalement, sauter vers l'avant (*i.e.* axe $\overrightarrow{Ox}$ positif) et rester dans un état vertical stable.

4.2 Environnement d'expérimentation

Lors de sa conception, du fait du coût humain et matériel, un robot est généralement modélisé à l'aide

10. Si le robot penche d'un côté, le volant d'inertie tourne dans le sens inverse afin de stabiliser le robot en position verticale.

d'un logiciel de Conception Assistée par Ordinateur (CAO). La figure 5 est ainsi le résultat de la modélisation de TWIG au moyen du logiciel de CAO SOLIDWORKS [12]. C'est à partir de cette modélisation virtuelle que les concepteurs cherchent ensuite à établir les lois de commande de leur robot. Ils utilisent pour cela des outils de prototypage qui servent à simuler le comportement du robot face aux lois physiques du monde réel. Dans le cas de TWIG, nous avons utilisé COSMOSMOTION, le simulateur de SOLIDWORKS.

Pour modéliser les lois de commande de TWIG, nous disposons des descripteurs suivants :

- U_θ et U_R correspondent aux tensions qui sont respectivement appliquées au volant d'inertie et au vérin hydraulique.
- R représente la distance de déploiement de la partie mobile du vérin par rapport à sa partie fixe.
- θ caractérise l'angle d'orientation du volant d'inertie par rapport au corps du robot.
- G est le point de gravité du robot, sur lequel est fixé un accéléromètre. Les descripteurs $\ddot{x}_G$ et $\ddot{z}_G$ caractérisent les accélérations de G, et les descripteurs x_G et z_G indiquent la position de G.[11]
- s est un capteur booléen qui vaut 1 quand TWIG touche le sol, 0 sinon.

Ces variables d'états nous ont été recommandées par les roboticiens.

4.3 Modélisation des actions élémentaires de Twig

La fréquence du simulateur COSMOSMOTION a été fixée à 100 Hertz, c'est à dire que pendant une durée de 1 seconde, le simulateur réalise 100 mesures successives de chacun des descripteurs de TWIG. Pour modéliser les différentes actions élémentaires dans le formalisme STRIPS, nous nous sommes cependant intéressés à l'état des descripteurs au début et à la fin de chaque action élémentaire, et nous avons pris soin de prendre en compte le temps nécessaire à la réalisation de chaque action élémentaire. Chacune des actions élémentaires de TWIG a été décrite sous la forme de tuples étiquetés $(t^i,s^i,x^i_G,z^i_G,U_R,U_\theta,t^f,s^f,x^f_G,z^f_G)$, où la variable t référence le temps, et où les exposants i et f indiquent respectivement l'état des variables au temps initial et final de chaque action élémentaire. Par souci de simplification des réseaux de contraintes, nous avons choisi de ne pas étudier les descripteurs θ et R.[12]

11. Pour notre étude, x_G et z_G nous sont donnés directement par le simulateur, mais peuvent être calculés en intégrant deux fois les accélérations de G.

12. Ces descripteurs seront cependant pris en compte dans de futures expérimentations, comme nous le verrons dans la suite de l'article.

À l'aide de COSMOSMOTION, nous avons effectué une série de simulations afin de constituer les données d'entraînement nécessaires à l'apprentissage automatique, à raison de 3 instances positives et 4 instances négatives en moyenne pour chaque action élémentaire. À partir de ces données d'entraînement, CONACQ a **automatiquement** modélisé les quatre actions élémentaires de TWIG à l'aide des réseaux de contraintes suivants :

Saut Vertical

$$\begin{cases} \mathcal{P}_p = \{(s^i = 1)\} \\ \mathcal{P}_a = \{(U_R = 500,\, U_\theta = 0)\} \\ \mathcal{P}_e = \{(s^f = 0,\, x^f_G = x^i_G,\, z^f_G = z^i_G + 3,\, t^f = t^i + 2)\} \end{cases}$$

Atterrissage Vertical

$$\begin{cases} \mathcal{P}_p = \{(s^i = 0)\} \\ \mathcal{P}_a = \{(U_R = 0,\, U_\theta = 0)\} \\ \mathcal{P}_e = \{(s^f = 1,\, x^f_G = x^i_G,\, z^f_G = z^i_G - 3,\, t^f = t^i + 20)\} \end{cases}$$

Saut Horizontal

$$\begin{cases} \mathcal{P}_p = \{(s^i = 1)\} \\ \mathcal{P}_a = \{(U_R = 300,\, U_\theta = 450)\} \\ \mathcal{P}_e = \{(s^f = 0,\, x^f_G = x^i_G + 3,\, z^f_G = z^i_G + 2,\, t^f = t^i + 6)\} \end{cases}$$

Rester stable

$$\begin{cases} \mathcal{P}_p = \{(s^i = 1)\} \\ \mathcal{P}_a = \{(U_R = 0,\, U_\theta = 0)\} \\ \mathcal{P}_e = \{(s^f = 1,\, x^f_G = x^i_G,\, z^f_G = z^i_G,\, t^f = t^i + 1)\} \end{cases}$$

Si le temps moyen d'exécution des sauts et de l'état stable est relativement court (3 sec), il convient de noter que le réseau de contraintes modélisant l'atterrissage nous indique une durée totale de 20 secondes. Les actionneurs de TWIG n'étant pas sollicités lors de la phase d'atterrissage, seul le ressort hélicoïdal permet de retrouver l'altitude nominale de TWIG, par extensions et compressions successives. Le temps nécessaire à l'atterrissage est par conséquent plus long que celui du *décollage*.

Dans cette expérimentation, une différence notable par rapport au framework décrit précédemment réside dans le fait que pour l'instant nous avons ici des CSP instanciés en partie.

4.4 Planification d'actions pour Twig

Dans le formalisme STRIPS, les pré-conditions et les effets sont exprimés sous la forme de conjonctions propositionnelles. Dans l'approche que nous proposons, ces mêmes pré-conditions et effets sont modélisés sous forme de réseaux de contraintes. Pour établir des lois de commande à partir de ces différentes actions élémentaires, nous devons disposer d'un planificateur de tâches capable de manipuler ces mêmes réseaux de contraintes pour construire ses plans. Nous n'avons malheureusement pas eu le temps d'adapter l'algorithme GRAPHPLAN afin d'implémenter un tel planificateur de tâches.

Cependant, au regard des réseaux de contraintes que nous avons acquis, et dans la mesure où nous ne nous situons pas dans le cadre d'une planification de haut niveau,[13] nous pouvons légitimement espérer que l'utilisation d'un planificateur de tâches issu de GRAPHPLAN nous permettra d'obtenir des réponses des lois de commandes avec des temps de réponses très raisonnables.

5 Discussions et perspectives

L'approche que nous proposons dans cet article présente une réelle alternative aux techniques employées par les roboticiens à l'heure actuelle pour combiner plusieurs actions élémentaires dans le but de réaliser une tâche de plus haut niveau. Si l'expérimentation réalisée sur TWIG présentée dans la section précédente a permis de valider l'intérêt de notre approche, il convient cependant de remarquer qu'elle a volontairement été biaisée pour concentrer nos propos sur l'application de notre approche à TWIG. Dans cette section, nous nous attachons donc à décrire les perspectives de travail que nous comptons mener afin d'étendre facilement notre approche à des robots plus complexes que TWIG.

Discrétisation des variables

Les différents descripteurs utilisés en robotique prennent majoritairement leur valeur dans des espaces continus. CONACQ manipulant des réseaux de contraintes discrets, il est nécessaire d'effectuer un pré-traitement sur les données issues des capteurs afin de les discrétiser. Dans le cas de TWIG, nous avons arbitrairement discrétisé chacun des descripteurs du robot afin que CONACQ puisse effectuer correctement le processus de modélisation.

13. *i.e.* Nombre restreint d'actions, buts à atteindre raisonnables, etc.

Une discrétisation arbitraire contraint parfois trop l'apprentissage. En conséquence, dans l'optique de projets plus complexes, il serait souhaitable de mettre au point un module permettant d'effectuer la discrétisation la mieux adaptée en fonction du robot étudié.

Biais d'apprentissage

L'objectif de l'expérimentation n'étant pas d'étudier avec précision tout le panel des actions élémentaires de TWIG, nous avons utilisé un biais d'apprentissage mettant en jeu une librairie limitée de contraintes. Pour une utilisation de notre approche dans une modélisation d'actions élémentaires plus complexes, telle que retrouver la position d'équilibre par bonds successifs (lorsque TWIG est penché), la librairie de contraintes utilisée devra être étoffée. Pour ce type d'actions élémentaires, les contraintes devront permettre d'exprimer la tension à appliquer au moteur pour rétablir l'équilibre en fonction de l'angle d'inclinaison θ de TWIG par rapport au sol, de la distance R de déploiement du vérin, de sa vitesse de chute, etc.

Dans le cas de systèmes robotisés beaucoup plus complexes, tels que les robots humanoïdes, il faudra là encore étoffer de manière conséquente la librairie de contraintes utilisée. Cependant, dans ce cas, toute la difficulté consistera à trouver le bon compromis entre l'expressivité des réseaux de contraintes acquis (afin de modéliser au plus près les actions élémentaires), et les propriétés computationnelles de ces mêmes réseaux de contraintes (nécessaires pour garantir une planification de tâches rapide). C'est ce que nous testerons dans le futur au LIRMM, sur des experimentations liées au robot humanoïde HRP-2 [8].

Module de gestion de la planification

Dans le cas de robots mobiles de *haut niveau* conçus pour réaliser des tâches très complexes et ayant beaucoup d'interactions avec leur environnement, il est parfaitement possible d'envisager qu'il existe différentes séquences d'actions permettant d'accomplir un même objectif. Dans ce cas, il nous parait important d'envisager dès maintenant un module de gestion de la planification qui, étant données plusieurs séquences d'actions permettant d'accomplir un même but, permette de préférer une séquence aux autres en fonction de certains critères, tels que la consommation en énergie, la dangerosité de la séquence pour le robot ou son environnement, l'impact de la séquence d'actions sur l'environnement, etc.

Un robot pourvu d'un tel module de gestion de la planification pourrait alors être capable d'écarter une séquence d'actions au profit d'une autre séquence

conceptuellement meilleure. La réalisation de tels robots constituerait un pas non négligeable vers la conception de robots de plus en plus autonomes, capables d'interagir dynamiquement avec leur environnement, tout en considérant des critères propres d'optimalité.

6 Conclusion

Dans cet article, nous avons proposé une approche qui permet de modéliser sous la forme de réseaux de contraintes les lois de commande qui régissent les actions élémentaires d'un robot mobile. À l'aide de la plateforme d'acquisition de contraintes CONACQ, nous modélisons chaque action élémentaire sous le formalisme STRIPS au moyen de trois réseaux de contraintes exprimant respectivement les conditions dans lesquelles l'action peut-être effectuée, les effets de l'action sur le robot et son environnement, ainsi que la loi de commande régissant les actionneurs du robot. Combiner entre elles plusieurs actions élémentaires revient alors à rechercher un plan à l'aide d'un planificateur de tâches tel que GRAPHPLAN. La technique que nous avons présentée dans cet article se révèle ainsi être une bonne alternative aux techniques actuelles des roboticiens. Là où les roboticiens modélisent chaque action élémentaire à l'aide de systèmes d'équations complexes qu'il est ensuite difficile de combiner avec d'autres actions élémentaires, notre approche permet de modéliser de manière élégante et performante chaque action élémentaire puis permet ensuite de combiner plusieurs actions élémentaires dans le but d'accomplir des tâches de plus haut niveau.

Enfin, dans le cadre d'une collaboration transversale entre roboticiens et informaticiens, des premiers résultats expérimentaux ont permis de valider l'intérêt de notre approche sur l'acquisition de modèles CSP pour les actions élémentaires et ont mis en lumière quelques aspects à renforcer tant pour une utilisation complète de la boucle de séquencement que pour son utilisation dans le cadre de robots plus complexes.

Remerciements

Nous tenons vivement à remercier Jean SALLANTIN, qui a su déceler avec TWIG une opportunité de travail permettant aux deux départements du LIRMM de s'enrichir mutuellement sur deux domaines connexes. De même, nous souhaitons remercier Rémi COLETTA tant pour ses précieux conseils sur les aspects théoriques de notre approche que pour son aide concernant l'utilisation de CONACQ.

Références

[1] AIBO *http://www.eu.aibo.com/* Sony Aibo Europe - Official Website.

[2] S. Russell, P. Norvig *Artificial Intelligence - A Modern Approach.* Second Edition, Prentice Hall, ISBN:0-13-080302-2, 2003.

[3] ASIMO *http://asimo.honda.com/* ASIMO Humanoïd Robot - Honda Robotic Technology.

[4] C. Bessiere, R. Coletta, F. Koriche, B. O'Sullivan *A SAT-Based Version Space Algorithm for Acquiring Constraint Satisfaction Problems.* Proceedings of ECML'05, Porto, Portugal, pages 747-751, Octobre 2005.

[5] R. Coletta, C. Bessière, B. O'Sullivan, E.C. Freuder, S. O'Connell, J. Quinqueton *Semi-automatic modeling by constraint acquisition.* Proceedings of CP-2003, Short paper, LNCS 2833, Springer Kinsale, Cork, Ireland., pages 812-816, Septembre 2003.

[6] FIRA *http://www.fira.net* The Federation of International Robot-soccer Association.

[7] A. L. Blum, M. L. Furst *Fast Planning Through Planning Graph Analysis.* Proceedings of Artificial Inteligence, pages 281-300, 1997.

[8] HRP-2 *http://www.kawada.co.jp/global/ams/hrp_2.html* The humanoïd robot project of Kawada Industries, Inc.

[9] A. Legtchenko, A. Lallouet *Acquisition de Contraintes Ouvertes par Apprentissage de Solveurs.* Proceedings of CAP'05, Francois Denis ed, Nice, France, Juin 2005.

[10] M. Paulin *Apprentissage interactif de réseau de contraintes.* Proceedings of RJCIA'05, pages 225-238, Nice, France, Juin 2005.

[11] Marc H. Raibert *Legged robots that balance.* MIT Press Series In Artificial Intelligence, Cambrige, Massachussetts, ISBN:0-262-18117-7, 1986.

[12] SolidWorks *http://www.solidworks.fr* Logiciels de Conception Assistée par Ordinateur.

[13] TAIPAN *http://www.lirmm.fr/taipan/* The Taipan web site.

[14] B. Jouvencel, P. Lepinay, J. Vaganay, R. Zapata *TAIPAN, an AUV for very shallow water applications* Session invitée à WAC'98, : 3nd World Automation Congress, Anchorange, Alaska USA. May 1998.

[15] P. van Beek, X. Chen *CPlan: A Constraint Programming approach to Planning* Proceedings of AAAI-99, Orlando, Florida., pages 585-590, 1999.