Une approche programmation par contraintes pour la modélisation et la planification de mouvements en robotique humanoïde.

Mathias PAULIN

LIRMM - Univ. Montpellier 2, CNRS
161 rue Ada 34392 Montpellier
Mathias.Paulin@lirmm.fr

Résumé : Il existe de nombreuses démonstrations de comportements évolués pour les robots humanoïdes de dernière génération (montée de marches, préhension d'objet, *etc.*). Toutefois, il s'agit dans la plupart des cas de comportements préenregistrés qui nécessitent de nombreux calculs, et qui s'adaptent difficilement aux modifications de contexte. Dans cet article, nous présentons une approche innovante qui modélise les actions élémentaires d'un robot sous la forme de réseaux de contraintes, et combine ensuite ceux-ci à l'aide d'outils de planification. Lors de l'exécution d'une séquence d'actions, nous utilisons les bonnes propriétés des réseaux de contraintes pour contrôler et éventuellement corriger l'exécution d'une tâche. L'utilisation des contraintes fournit au robot un modèle de lui même et de son environnement, grâce auquel il peut raisonner sur ses comportements, ce qui lui confère une première forme d'autonomie. La fin de l'article présente des résultats expérimentaux préliminaires permettant de valider l'intérêt de notre approche, dont on peut désormais envisager le déploiement effectif sur un robot humanoïde tel que HOAP3 de FUJITSU.

Mots-clés : Programmation par contraintes, Planification de tâches.

1 Introduction

Les dernières générations de robots humanoïdes comme ASIMO (Honda) ou HRP-2 (Kawada) démontrent que des progrès colossaux ont été accomplis en termes d'habiletés physiques. Il existe désormais de nombreuses démonstrations de comportements évolués, telles que la préhension d'objets, la montée de marches, ou la danse. Toutefois, il s'agit dans la plupart des cas de comportements calculés à l'avance, qui sont ensuite déroulés de manière automatique, et qui s'adaptent difficilement aux modifications de contexte.

La difficulté à planifier des comportements est la conséquence de l'approche actuelle des roboticiens, qui s'attache à modéliser au plus près chaque action élémentaire en utilisant les lois physiques du monde (loi de conservation des énergies, moments cinétiques, *etc.*). Combiner plusieurs actions élémentaires se révèle être particulièrement

difficile et nécessite de nombreux calculs, pouvant aller jusqu'à plusieurs heures. Pour pallier cet écueil, les mécaniciens tentent d'utiliser des modèles cinématiques simplifiés, ainsi que la recherche de comportements optimaux par analyse des comportements humains. L'étude de la dynamique posturale chez l'homme a ainsi permis de définir des dynamiques de marche très intéressantes (Collins & Luca, 1993). Cependant, les comportements ainsi générés sont très difficiles à exploiter dans un contexte réel. En effet, la modélisation employée ne tient pas compte des perturbations extérieures qui peuvent se produire lors d'une exécution réelle. Un certain nombre de techniques sont alors utilisées pour permettre une exploitation plus large. De nombreux travaux ont ainsi été réalisés dans l'apprentissage, via des réseaux de neurones, d'actions réflexes automatiques susceptibles de tolérer des perturbations extérieures (Henaff, 2007).

Les liens entre l'intelligence artificielle et la robotique humanoïde se font de plus en plus étroits (réseaux de neurones, reconnaissance vocale, vision stéréoscopique, ...). Malgré ces liens, entre les robots actuels et le rêve de robots autonomes, capables de raisonner sur leurs comportements et de s'adapter à des modifications de contexte, l'écart reste encore important.

Dans cet article, nous proposons une amélioration de l'architecture logicielle proposée dans (Paulin, 2006), qui utilise le paradigme de la Programmation Par Contraintes (PPC) pour modéliser et planifier des comportements en robotique humanoïde. Notre approche consiste dans un premier temps à modéliser par apprentissage automatique les habiletés physiques d'un robot à l'aide de réseaux de contraintes (CSP), qui sont ensuite composés à l'aide de techniques de planification de tâches. Lors de l'exécution d'une séquence d'actions, nous utilisons les bonnes propriétés des CSP acquis, pour contrôler en temps réel la réalisation d'une tâche, en corrigeant les éventuels écarts existants entre les prévisions fournies par les CSP et les valeurs réelles.

Notre approche cherche à s'abstraire des lois physiques qui régissent la commande des robots humanoïdes. En effet, les réseaux de contraintes qu'elle manipule modélisent les conditions dans lesquelles chaque action élémentaire peut être exécutée, c'est-à-dire le *comment* et non le *pourquoi*. Cette modélisation est plus faible que ce qui est "mécaniquement" nécessaire, mais est en contrepartie beaucoup plus facile à manipuler car elle fournit, pour chaque action élémentaire, un *modèle* qui abstrait un espace de solutions possibles. La planification de séquences d'actions est en conséquence beaucoup plus aisée. Les réseaux de contraintes fournissent par ailleurs un modèle à même d'expliquer un état futur, la cause d'une replanification de comportement, ou bien encore les raisons d'un échec. Cette approche permet ainsi à un robot de raisonner sur ses aptitudes physiques, ce qui constitue un premier pas vers une autonomie comportementale non réflexe.

Le reste de cet article est organisé comme suit. Dans la section 2, nous introduisons les concepts utilisés dans l'article. La section 3 présente l'architecture générale de notre système de supervision, que nous définissons formellement dans la section 4. Avant de conclure en présentant nos perspectives de travail (section 6), la section 5 décrit quelques expérimentations préliminaires permettant de valider empiriquement l'intérêt de l'approche PPC pour la supervision de robots humanoïdes.

2 Préliminaires

2.1 Rappel sur la programmation par contraintes

Un réseau de contraintes $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est défini par un ensemble fini $\mathcal{X} = \{x_1, \ldots, x_n\}$ de n variables prenant chacune une valeur de leur domaine respectif $D_{x_1}, \ldots, D_{x_n}$, éléments de $\mathcal{D}$, et par $\mathcal{C} = \{c_1, \ldots, c_m\}$ 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 $(v_1, \ldots, v_n) \in D_{x_1} \times \ldots \times D_{x_n}$. 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 $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

La programmation par contraintes connaît un succès croissant et est désormais largement utilisée pour résoudre des problèmes réels difficiles hors de portée des autres approches (Wallace, 1996). 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 la recherche de solutions valides (le "Comment"). De nombreux travaux académiques et industriels ont permis la mise en place d'un large panel d'outils, tels que les contraintes globales (Bessiere & Hentenryck, 2003), les contraintes relaxées (Gaspin *et al.*, 2005) ou les QCSP (Benedetti *et al.*, 2006; Verger & Bessiere, 2006), qui permettent de modéliser une grande variété de problèmes. Par ailleurs, les dernières générations de résolveurs de contraintes comme Ilog Solveur ou Choco sont particulièrement performants et sont capables de déterminer efficacement si un problème est soluble et, le cas échéant, trouver une solution.

Forte de trente années de recherche et de progrès constants, la programmation par contraintes se révèle être un paradigme de choix, tant par son expressivité que par son efficacité à résoudre des problèmes d'envergure.

2.2 Le problème de planification de tâches

Dans sa définition la plus commune (Russell & Norvig, 2003), la planification classique 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 manipule les concepts d'état, de but et d'action.

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.

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.

3 Architecture générale du système de supervision

La modélisation des lois physiques actuellement utilisée par les roboticiens se révèle très difficile à manipuler lorsque l'on cherche à planifier une séquence d'actions. L'approche que nous proposons consiste à modéliser les actions élémentaires d'un robot à l'aide de réseaux de contraintes simples, ayant de bonnes propriétés calculatoires, puis à combiner et composer ces réseaux afin de planifier des comportements.

Nous utilisons la plate-forme CONACQ (Bessiere *et al.*, 2005) pour modéliser les actions élémentaires d'un robot par apprentissage automatique. Pour modéliser chaque action élémentaire a_i, il suffit de fournir un ensemble d'instances valides de a_i, ainsi qu'un ensemble d'instances ne correspondant pas à a_i. CONACQ modélise alors automatiquement a_i sous la forme d'un réseau de contraintes qui décrit les conditions dans lesquelles a_i peut être exécutée (*i.e.* sa précondition), ses effets, et enfin la manière dont les différents actionneurs du robot doivent réagir pour effectuer a_i. Les réseaux de contraintes ainsi acquis deviennent alors des unités élémentaires de contrôle que l'on peut composer à l'aide des outils de planification (figure 1).

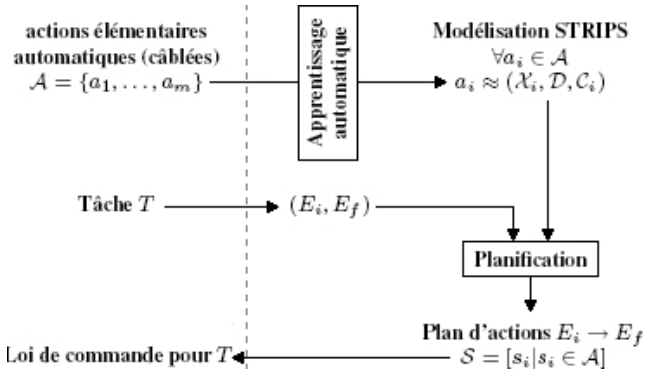

FIG. 1 – Acquisition des unités élémentaires de contrôle et planification.

La figure 2 illustre le principe de fonctionnement de notre système de supervision. Étant donnée une tâche T ne pouvant être réalisée au moyen d'une unique action élémentaire, un planificateur de tâches se charge de trouver une séquence S d'actions élémentaires dont l'exécution doit permettre de réaliser T. Pour réaliser ses calculs, le planificateur manipule et compose les unités élémentaires de contrôle modélisées par les réseaux de contraintes à l'étape précédente. Une fois calculée, la séquence S est transmise au module de contrôle qui se charge de faire exécuter cette séquence d'actions au robot ou à son simulateur. Cette exécution se fait alors pas à pas, c'est-à-dire une action

après l'autre. Après l'exécution de chaque action élémentaire, le module de contrôle confronte les mesures-capteurs réelles aux prévisions établies via la modélisation en contraintes. En cas d'écart mineur, un simple appel au solveur de contraintes permettra d'ajuster les prochaines commandes moteur à exécuter. En revanche, si l'écart entre les prévisions et la réalité est trop important, le module de supervision fera à nouveau appel au planificateur pour établir une nouvelle séquence d'actions permettant d'atteindre le but T depuis le nouvel état courant.

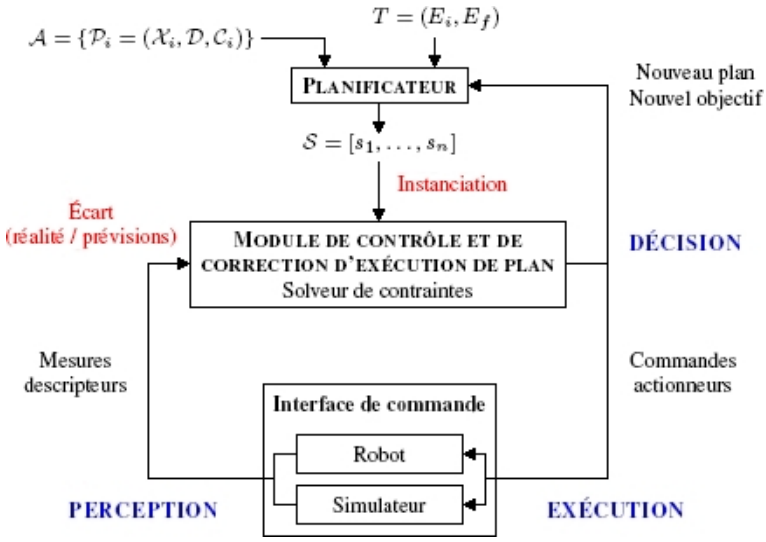

FIG. 2 – Contrôle et correction de l'exécution d'une séquence d'actions.

4 Formalisation du processus de supervision

Dans cette section, nous définissons de manière formelle le processus de supervision de notre plate-forme. Nous considérons pour cela un robot R constitué de p descripteurs (ses capteurs), qui décrivent son état courant, et de q actionneurs (ses moteurs). Soient $\delta = \{\delta_1, ..., \delta_p\}$ l'ensemble des descripteurs de R, $\alpha = \{\alpha_1, ..., \alpha_q\}$ l'ensemble de ses actionneurs et $\mathcal{A} = \{a_1, \ldots, a_m\}$ l'ensemble des actions élémentaires qu'il peut exécuter.

4.1 Apprentissage des unités élémentaires de contrôle

Comme nous l'avons décrit dans la section précédente, notre approche consiste à modéliser chaque action élémentaire $a_i \in \mathcal{A}$ à l'aide d'un réseau de contraintes. Afin d'acquérir ces unités élémentaires de contrôle, nous définissons les fonctions $varI$, $varA$ et $varF$ définies sur $\mathcal{A}$ de la manière suivante : $varI(a_i)$ renvoie l'ensemble

$\{\delta_1^I, \ldots, \delta_p^I\}$ des variables qui modélisent les descripteurs de R **avant** l'exécution de a_i, $varA(a_i)$ renvoie l'ensemble $\{\alpha_1, \ldots, \alpha_q\}$ des variables qui modélisent les actionneurs de R **pendant** l'exécution de a_i et $varF(a_i)$ renvoie l'ensemble $\{\delta_1^F, \ldots, \delta_p^F\}$ des variables qui modélisent les descripteurs de R **après** l'exécution de a_i.

Chaque action élémentaire $a_i \in \mathcal{A}$ est alors modélisée en suivant la définition 1.

Définition 1 (Modélisation par contraintes)
Une action élémentaire $a_i \in \mathcal{A}$ est modélisée par le réseau de contraintes $\mathcal{P}_i = (X_i, D, C_i)$ tel que :

$$\begin{cases} X_i = varI(a_i) \cup varA(a_i) \cup varF(a_i), \\ D = \{D(x_j) | x_j \in X_i \text{ et } D(x_j) \text{ est le domaine de valeurs possibles pour } x_j\}, \\ C_i = Pre(a_i) \cup Action(a_i) \cup Post(a_i). \end{cases}$$

où $Pre(a_i)$ est l'ensemble des contraintes qui modélisent les conditions dans lesquelles a_i peut être exécutée, $Action(a_i)$ est l'ensemble des contraintes qui expriment comment a_i est exécutée, et $Post(a_i)$ est l'ensemble des contraintes qui modélisent les effets de a_i sur le robot et son environnement.

4.2 Planification d'une séquence d'actions

Après acquisition des unités élémentaires de contrôle, chaque action élémentaire $a_i \in \mathcal{A}$ est modélisée par le réseau $\mathcal{P}_i$. Il reste alors à combiner ces différentes unités de contrôle pour établir une séquence $\mathcal{S}$ d'actions élémentaires que le robot devra exécuter pour accomplir une tâche T.

Nous exprimons chaque action élémentaire a_i dans le formalisme STRIPS. La précondition de a_i est donnée par restriction du réseau $\mathcal{P}_i$ à $Pre(a_i)$, et son effet par $Post(a_i)$. La tâche T est quant à elle exprimée au moyen de l'état initial E_i et de l'état final E_f (*i.e.* le but de T) tels que définis à la section 2.2. Exprimé de la sorte, le problème de départ consistant à trouver une séquence d'actions élémentaires permettant de réaliser T revient désormais à résoudre le problème de planification de tâches $(E_i, E_f, \mathcal{A})$. Pour le résoudre, nous faisons appel à un planificateur de tâches utilisant le formalisme STRIPS.

S'il existe un plan permettant d'atteindre E_f à partir de E_i, le planificateur renvoie une séquence $\mathcal{S} = [s_i | s_i \in \mathcal{A}]$ qui renferme les différentes actions élémentaires a_i à exécuter. $\mathcal{S}$ correspond dans ce cas à une séquence respectant les conditions d'enchaînements des actions élémentaires permettant d'accomplir T. En l'état, elle ne permet cependant pas de déterminer quelles tensions doivent être successivement appliquées aux moteurs du robot pour que ce dernier réalise effectivement T.

4.3 Instanciation d'une séquence d'actions élémentaires

Soit T une tâche non élémentaire et $\mathcal{S} = [s_1, \ldots, s_n]$ une séquence d'actions élémentaires planifiée selon le processus décrit précédemment. Pour déterminer quelles sont les tensions à appliquer pour réaliser T, il faut dans un premier temps créer le réseau de contraintes, appelé dans la suite CSP global, correspondant à $\mathcal{S}$. Ce CSP global est

défini par instanciation (au sens *Objet* du terme) des réseaux de contraintes modélisés lors du processus d'apprentissage, et subit une phase de propagation pour éliminer les valeurs inconsistantes. Dans un second temps, il convient de résoudre ce CSP global afin d'obtenir une séquence d'actions exécutable par le robot.

L'instanciation d'une action élémentaire $a_i \in \mathcal{A}$ à un instant t se définit à l'aide des définitions 2 et 3.

Définition 2 (Instanciation des variables)
Soit $a_i \in \mathcal{A}$ une action élémentaire modélisée par le réseau de contraintes $\mathcal{P}_i = (X_i, D, C_i)$ où $X_i = \{\delta_1^I, \ldots, \delta_p^I, \alpha_1, \ldots, \alpha_q, \delta_1^F, \ldots, \delta_p^F\}$. Une variable $x \in X_i$ est instanciée à un instant t au moyen de la fonction $inst$ définie sur $X_i \times \mathbb{N}$ comme suit :

$$inst(x,t) = \begin{cases} \delta_j^{I,t} & \exists j \in [1..p] \text{ tel que } x = \delta_j^I \\ \alpha_j^t & \exists j \in [1..q] \text{ tel que } x = \alpha_j \\ \delta_j^{F,t} & \exists j \in [1..p] \text{ tel que } x = \delta_j^F \end{cases}$$

On note alors $inst(X_i, t) = \{inst(x,t) | x \in X_i\}$ l'instanciation de l'ensemble X_i à l'instant t.

Définition 3 (Instanciation d'une action)
L'instanciation de $a_i \in \mathcal{A}$ à un instant t, notée $a_{i,t}$ est définie par le triplet $(X_{i,t}, D, C_{i,t})$ tel que :

$$\begin{cases} X_{i,t} = inst(X_i, t), \\ C_{i,t} = \{c_{i,t} = (scope, sol(c_i)) | c_i \in C_i \text{ et } scope = inst(var(c_i), t)\}. \end{cases}$$

Afin d'associer à chaque élément s_t de $\mathcal{S}$ l'action élémentaire $a_i \in \mathcal{A}$ que l'on doit effectuer à l'étape t, on utilise la fonction d'association φ, définie sur $\{1..n\}$, et à valeurs dans $\{1..m\}$, qui à t associe $\varphi(t) = i$ si et seulement si $a_i \in \mathcal{A}$ est l'action à exécuter à l'instant t. La séquence $\mathcal{S}$ s'écrit alors $\mathcal{S} = [a_{\varphi(1),1}, \ldots, a_{\varphi(n),n}]$. Afin de déterminer quelles tensions doivent être successivement appliquées aux actionneurs du robot, nous devons créer le réseau de contraintes correspondant à l'exécution de la séquence $\mathcal{S} = [a_{\varphi(1),1}, \ldots, a_{\varphi(n),n}]$, qui est défini comme suit :

Définition 4 (CSP global)
Le réseau de contraintes permettant d'instancier $\mathcal{S}$ est le réseau de contraintes $\mathcal{P} = (X, D, C)$ où :

$$\begin{cases} X = \bigcup_{t=1}^{n} X_{\varphi(t),t}, \\ C = \bigcup_{t=1}^{n} C_{\varphi(t),t} \ \cup \ \bigcup_{t=1}^{n-1} \{(\delta_j^{F,t} = \delta_j^{I,t+1}) | j \in [1..p]\}. \end{cases}$$

Soient T une tâche que l'on souhaite faire réaliser par le robot R et (E_i, E_f) modélisant T. Soient alors $\mathcal{S}$ la séquence d'actions élémentaires permettant de réaliser T et $\mathcal{P} = (X, D, C)$ le CSP global permettant d'instancier $\mathcal{S}$ au sens de la définition 4.

Afin que $\mathcal{P}$ modélise correctement la tâche T, il convient d'ajouter à C un ensemble de contraintes d'égalité : $\forall \delta_j^{I,1} \in X$ (resp. $\delta_j^{F,n}$) modélisant un descripteur de R apparaissant dans E_i (resp. E_f), il faut ajouter à C la contrainte $(\delta_j^{I,1} = v)$ (resp. $(\delta_j^{F,n} = v)$) où v est la valeur de ce descripteur dans E_i (resp. E_f).

4.4 Exécution d'une séquence d'actions

Soit s une solution du CSP global, la propriété 1 nous permet d'établir que la restriction de s aux variables α_j^t $\forall (j,t) \in [1..q] \times [1..n]$ nous indique quelles tensions doivent être successivement appliquées aux actionneurs du robot pour réaliser T.

Propriété 1 (Solutions du CSP global)
Soient (E_i, E_f) le couple d'états initial et final caractérisant une tâche T que l'on souhaite exécuter et $\mathcal{S} = [s_1, \ldots, s_n]$ une séquence d'actions élémentaires permettant de réaliser T. Soit $\mathcal{P} = (X, D, C)$ le CSP global modélisant $\mathcal{S}$ et le couple (E_i, E_f). Si $\mathcal{P}$ admet des solutions, alors toute instanciation valide (solution) de $\mathcal{P}$ est un plan valide pour T.

5 Expérimentations préliminaires

Pour valider expérimentalement l'intérêt de notre approche, nous avons réalisé des expérimentations préliminaires sur les robots TWIG et PIRI respectivement représentés par les figures 3 et 4.

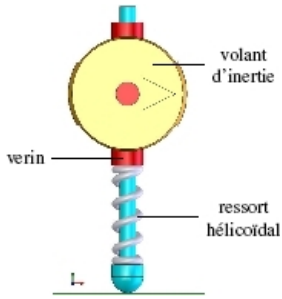

FIG. 3 – Le robot TWIG.

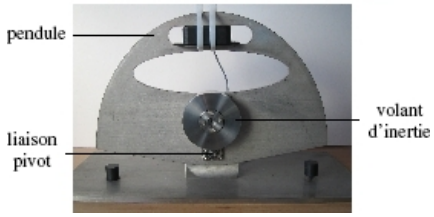

FIG. 4 – Le robot PIRI.

Dans un premier temps, nous avons poursuivi l'étude du robot TWIG initiée pour (Paulin *et al.*, 2006). Cette étude amont avait permis de valider l'utilisation de CONACQ pour modéliser automatiquement en contraintes les actions élémentaires de ce robot sauteur unijambiste modélisé sur simulateur. Le travail réalisé dans cette deuxième étude a consisté à combiner ces actions élémentaires (saut vertical, atterrissage vertical, saut horizontal, rester stable) à l'aide d'un planificateur de tâches capable de manipuler des réseaux de contraintes. Nous avons pour cela implémenté un planificateur STRIPS inspiré de GRAPHPLAN, à l'aide du résolveur de contraintes CHOCO,[1] auquel nous

[1] *http ://choco.sourceforge.net/* The CHOCO constraint programming system.

avons fait appel pour planifier des séquences d'actions pour TWIG. Cette série d'expérimentations a ainsi permis de valider l'intérêt de notre approche *contraintes* pour une planification de tâches élégante et performante.

Au travers de l'étude d'un Pendule Inversé stabilisé par une Roue d'Inertie (PIRI), nous avons cherché à montrer la capacité des contraintes à établir une loi de commande (robuste aux perturbations) de recherche d'équilibre pour un robot cadencé à 100Hz. Nous avons pour cela implémenté en langage CHOCO la loi de commande bas niveau[2] utilisée par les roboticiens et avons ensuite commandé PIRI à l'aide de ce réseau de contraintes. Cette étude expérimentale a mis en lumière la capacité de notre approche PPC à répondre aux impératifs du réel (fréquence élevée, perturbations extérieures) au moyen du cycle *perception / décision / action* sur lequel est basé notre système de supervision.

6 Conclusion et perspectives

Dans cet article, nous avons proposé un système de supervision qui utilise les réseaux de contraintes pour modéliser les actions élémentaires d'un robot humanoïde. Notre approche permet de modéliser de manière élégante et performante les habiletés physiques du robot sous la forme d'unités élémentaires de contrôle, qui sont ensuite combinées à l'aide d'outils de planification. L'utilisation des contraintes permet de fournir un *modèle* du robot et de son environnement, à partir duquel nous sommes capables d'établir des prévisions sur son état futur, puis de confronter ces prévisions à la réalité de l'exécution d'une action.

Par ailleurs, la réalité opérationnelle d'un robot autonome nécessite une réponse efficace et symbolique, que la programmation par contraintes est à même de fournir. Le système de supervision que nous proposons permet ainsi d'entrevoir une première génération de robots humanoïdes capables de raisonner minimalement sur leur comportement.

Les expérimentations préliminaires présentées dans la section 5 semblent démontrer la capacité de notre système de supervision à répondre aux impératifs d'expressivité et d'efficacité liés à la robotique humanoïde. Notre objectif est désormais de déployer et d'adapter ce système aux spécificités du robot humanoïde HOAP3 de FUJITSU (figure 5) récemment arrivé au Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM).

Fig. 5 - Le robot HOAP3.

Avec HOAP3, nous allons nous confronter à un cahier des charges d'envergure : 28 articulations motorisées, des descripteurs nombreux et de nature diverses (accéléromètres, télémètres, caméras stéréoscopiques, *etc.*) ou bien encore la nécessité de garan-

[2]La loi de commande utilisée est en effet la somme pondérée de la vitesse du volant d'inertie, l'angle du pendule par rapport à la verticale, et sa vitesse.

tir un fonctionnement sécurisé pour éviter chutes et casses mécaniques. En partenariat avec les roboticiens du laboratoire, nous souhaitons réaliser une intégration de notre approche cohérente et respectueuse des techniques de commande existantes, afin d'exploiter au mieux les habiletés physiques de HOAP3.

Remerciements

L'auteur tient à remercier Jean SALLANTIN qui a soutenu ce travail avec le plus vif intérêt. Certaines idées ont émergé suites à d'agréables et fructueuses discussions avec Eric BOURREAU, dont les remarques ont par la suite contribué à améliorer la lisibilité de cet article. Enfin l'auteur tient à remercier vivement Sébastien ANDARY pour les heures de travail passées à la programmation du robot PIRI.

Références

BENEDETTI M., LALLOUET A. & VAUTARD J. (2006). Reusing csp propagators for qcsps. In *Proceedings of the Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming (CSCLP-06)*, Lisbonne, Portugal.

BESSIERE C., COLETTA R., KORICHE F. & O'SULLIVAN B. (2005). A sat-based version space algorithm for acquiring constraint satisfaction problems. In *Proceedings of ECML'05*, p. 747–751, Porto, Portugal.

BESSIERE C. & HENTENRYCK P. V. (2003). To be or not to be...a global constraint. In L. 2833, Ed., *Proceedings of CP-2003*, p. 789–794, Springer Kinsale, Cork, Ireland.

COLLINS J. & LUCA C. D. (1993). Open-loop and closed-loop control of posture : A random-walk analysis of center-of-pressure trajectories. In *Experimental Brain Research*, p. 308–318.

GASPIN C., SCHIEX T. & ZYTNICKI M. (2005). Bound arc consistency for weighted csps. In *7th International CP-2005 Workshop on Preferences and Soft Constraints*.

HENAFF P. (2007). Genèse des mouvements rythmiques et réflexes inspirée de la biologie : Application à la locomotion et au contrôle d'équilibre des robots humanoïdes. In *Journées Nationales de la Robotique Humanoïde*, Montpellier, France.

PAULIN M. (2006). Supervision of robot tasks planning through constraint networks acquisition. In *CP-2006 Doctoral Programme*, p. 180–185, Nantes, France.

PAULIN M., BOURREAU E., DARTNELL C. & KRUT S. (2006). Modélisation et planification d'actions élémentaires robotiques par apprentissage de réseaux de contraintes. In *Proceedings of JFPC'2006*, p. 405–414, Nîmes, France.

RUSSELL S. & NORVIG P. (2003). *Artificial Intelligence - A Modern approach*. Second Edition. Prentice Hall.

VERGER G. & BESSIERE C. (2006). Blocksolve : a bottom-up approach for solving quantified csps. In *Proceedings of CP-2006*, p. 635–649, Nantes, France.

WALLACE M. (1996). Practical applications of constraint programming. *Constraints*, **1**(1/2), 139–168.