Apprentissage et optimisation de politiques pour un bras articulé actionné par des muscles

Didier Marin, Lionel Rigoux, Olivier Sigaud

Institut des Systèmes Intelligents et de Robotique
UPMC-Paris 6, CNRS UMR 7222
4 place Jussieu
75252 Paris Cedex 05, France
marin@isir.upmc.fr,lionel.rigoux@gmail.com,olivier.sigaud@upmc.fr

RÉSUMÉ. *De nombreux travaux présentent une combinaison d'apprentissage par démonstration et d'amélioration locale de politiques pour apprendre des contrôleurs pour des robots le long d'une trajectoire. Il manque à ces travaux une capacité de généralisation permettant d'apprendre sur tout l'espace atteignable par le robot. Dans cet article, nous présentons une méthode qui consiste à apprendre un tel contrôleur réactif en feedback et quasi optimal en deux étapes. Tout d'abord, un contrôleur paramétrique en feedback est appris par démonstration. On utilise pour cela des trajectoires réalisées par un contrôleur quasi optimal très coûteux. Ensuite, le contrôleur en feedback est optimisé par des méthodes de recherche directe sur les politiques. Nous obtenons alors un contrôleur quasi optimal qui s'exécute 20 000 fois plus vite que l'original, pour une performance proche. Ce travail est réalisé en simulation.*

ABSTRACT. *Many research works combine learning from demonstration and policy improvement methods to learn the controller of a robot along a specific trajectory. Nevertheless, a capability to learn in the whole reachable space of this robot is missing in these works. In this paper we propose a method that consists in learning a reactive near-optimal feedback controller in two steps. First, an efficient parametric feedback controller is obtained from learning from Demonstration based on the trajectories computed by a costly near-optimal controller. Second, the feedback controller is optimized further with direct Policy Search methods. As a result, we obtain a controller that is executed 20 000 times faster than the original controller for a similar performance. Our work is evaluated in simulation.*

MOTS-CLÉS : *commande optimale stochastique, apprentissage par démonstration, méthode de l'entropie croisée.*

KEYWORDS: *stochastic optimal control, learning from demonstration, cross-entropy methods.*

DOI:10.3166/RIA.27.205-225

1. Introduction

La conception de systèmes adaptatifs capables de traiter des problèmes de décision séquentielle dans des espaces d'état et d'action continus est un problème difficile qui engendre une activité de recherche importante. Une première approche face à ce type de problème réside dans les méthodes de recherche directe de politiques, qui ont connu un essort important ces dernières années (Mannor *et al.*, 2003 ; Peters, Schaal, 2008). Par ailleurs, on peut aussi résoudre des problèmes de décision séquentielle ou de commande avec des méthodes d'apprentissage supervisé en utilisant le paradigme de l'apprentissage par démonstration (Argall *et al.*, 2009). Il faut alors disposer d'une solution, éventuellement coûteuse, au problème que l'on veut apprendre à résoudre. Les travaux qui produisent les résultats les plus satisfaisants sont ceux qui combinent les deux approches. L'apprentissage supervisé est alors utilisé pour amorcer la recherche de politiques.

Dans la plupart des cas, ces méthodes sont appliquées dans un cadre robotique pour optimiser le comportement du robot le long d'une trajectoire, ce qui revient à réaliser l'optimisation sur un domaine de faible dimension, même quand la dimensionalité intrinsèque de l'état du système est importante. La capacité qui manque aux systèmes adaptatifs correspondants est une capacité de généralisation à tout l'espace de ce qui est appris localement.

Cet article présente une méthode combinant les deux approches décrites plus haut. Nous l'appliquons dans un cadre simulé où une politique est apprise sur tout l'espace atteignable d'un modèlé simulé de bras actionné par des muscles, auquel nous faisons réaliser des mouvements d'atteinte. Ce système est pertinent pour la modélisation de propriétés du contrôle moteur humain. Mais ce qui nous intéresse ici, avec un point de vue davantage tourné vers la robotique, est qu'écrire une loi de commande efficace pour un tel système avec des approches classiques de la commande ne va pas de soi, compte tenu de la redondance d'actionnement du système.

Pour l'apprentissage par démonstration, nous utilisons un algorithme de régression doté d'une bonne capacité de généralisation, ce qui permet d'apprendre un comportement adéquat sur un espace plus vaste que celui sur lequel le système a été entraîné. Cette méthode de régression est ensuite complétée par un algorithme d'optimisation stochastique qui est appliqué aux paramètres résultant de la régression. Le résultat final est un contrôleur en feedback qui s'exécute 20 000 fois plus vite que la solution originale coûteuse qui a servi à l'amorcer, tout en fournissant un comportement dont la performance est proche.

Pour évaluer notre méthode, nous commençons par montrer expérimentalement que la capacité de notre algorithme de régression à généraliser efficacement à partir d'un petit nombre d'exemples permet d'obtenir un contrôleur assez efficace sur l'ensemble des états atteignables par le système. Cependant, nous mettons en évidence le caractère généralement sous-optimal du comportement résultant.

Dans un second temps, nous montrons que le contrôleur appris par régression peut être amélioré par un algorithme d'optimisation stochastique qui agit sur les paramètres définissant ce contrôleur. Nous mettons alors en évidence la capacité de ce processus supplémentaire à améliorer la performance du contrôleur.

L'article est organisé de la façon suivante. Dans la section 2, nous présentons toutes les méthodes utilisées pour réaliser notre modèle. Dans la section 3, nous présentons le protocole expérimental. Les résultats sont exposés dans la section 4 et discutés dans la section 5. Enfin, la section 6 présente les perspectives de notre travail.

2. Méthodes

Dans cette section, nous décrivons les méthodes utilisées par notre modèle. Nous commençons par une brève présentation de la méthode de commande optimale utilisée dans (Rigoux *et al.*, 2010) pour modéliser des mouvements d'atteinte du bras. Nous présentons ensuite l'architecture globale sur laquelle repose notre approche, puis les algorithmes XCSF et *Cross-Entropy Policy Search* (CEPS) qui sont utilisés respectivement pour apprendre un contrôleur initial par démonstration et pour optimiser ce contrôleur.

2.1. Commande optimale des mouvements d'atteinte

Nous commençons par établir une relation entre le cadre formel des PDM (Sigaud, Buffet, 2008) et le cadre des travaux de (Rigoux *et al.*, 2010) qui s'intéresse à la modélisation des mouvements d'atteinte d'un bras actionné par des muscles.

2.1.1. Commande optimale et PDM

La commande optimale et les processus décisionnels de Markov (PDM) sont deux cadres formels similaires dès lors que les seconds considèrent des états et des actions continus (Bertsekas, 1995). Ces cadres permettent de faire atteindre un but à un système par l'optimisation d'une fonction objectif. Dans la littérature sur la commande optimale, la fonction objectif est exprimée comme une fonction de coût à minimiser. Dans le cadre des PDM, il s'agit d'une fonction de récompense à maximiser. Au pas de temps t, l'état du système est un vecteur $\boldsymbol{x}_t \in \mathcal{X}$ et l'action est un vecteur $\boldsymbol{u}_t \in \mathcal{U}$. La fonction de coût immédiat (ou de récompense immédiate) $r : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$ représente le coût immédiat (ou l'intérêt immédiat) engendré par une action donnée dans un état donné, relativement au problème global à résoudre.

La qualité d'une politique π pour un état $\boldsymbol{x}$ est exprimée par sa fonction de valeur, qui correspond à l'espérance de la somme actualisée des coûts (ou récompenses) au cours du temps :

$$V^{\pi}(\boldsymbol{x}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{x}_t, \boldsymbol{u}_t) \mid \boldsymbol{x}_0 = \boldsymbol{x}\right]$$

où γ est le facteur d'actualisation relatif à l'incertitude sur les coûts ou récompenses futurs. On cherche la politique π^* qui maximise la fonction V ci-dessus.

2.1.2. Un modèle des mouvements d'atteinte

Les mouvements d'atteinte chez l'homme sont des mouvements effectués avec le bras pour toucher une cible. Dans (Rigoux *et al.*, 2010), les auteurs proposent un modèle de ces mouvements qui repose sur un paradigme de commande optimale actuellement dominant dans la littérature sur le contrôle moteur humain (Todorov, Jordan, 2002 ; Guigon *et al.*, 2008). Le modèle suppose que le contrôle moteur humain est régi par une commande optimale en *feedback* calculée en chaque état visité sur la base d'une fonction de valeur V qui résulte d'un compromis entre un effort musculaire et une récompense liée à l'atteinte du but :

$$r(\boldsymbol{x}_t, \boldsymbol{u}_t) = \alpha \|\boldsymbol{u}_t\|^2 - \beta g(\boldsymbol{x}_t) \quad (1)$$

où α est le poids du terme d'effort, g est une fonction nulle partout sauf pour l'état but x^* où elle vaut 1 et β est le poids du terme de récompense. Pour trouver la politique déterministe et quasi optimale correspondante, les auteurs de (Rigoux *et al.*, 2010) utilisent une méthode de calcul variationnel très coûteuse.

Le contrôleur en *feedback* qui résulte du couplage de cette politique avec un estimateur d'état optimal amène le bras à la cible.

Dans la suite, nous appelons *Near-Optimal Planning System* (NOPS) le contrôleur issu des travaux de (Rigoux *et al.*, 2010). Les trajectoires ne sont pas optimales au sens strict, car elles ne prennent pas en compte la présence d'un bruit qui n'est pas modélisé. En particulier, l'état atteint après chaque pas n'étant pas l'état prédit à cause du bruit, une nouvelle séquence d'actions doit être calculée à chaque pas à partir de l'état effectivement atteint par le système, ce qui augmente encore le coût de la méthode.

2.2. Architecture globale

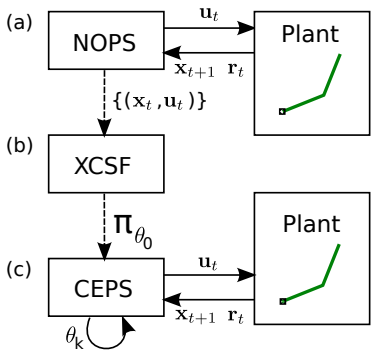

Figure 1. Architecture globale de notre approche

La figure 1 décrit l'architecture globale utilisée pour les expériences. Dans un premier temps, nous engendrons avec le NOPS (figure 1 (a)) un ensemble de trajectoires état-action quasi optimales. Dans un second temps, les couples état-action successifs de ces trajectoires sont utilisés comme échantillons pour la régression d'un modèle de la politique engendrée par le NOPS. Pour réaliser cette régression, nous avons choisi l'algorithme XCSF, qui offre une bonne efficacité computationnelle et une bonne capacité de généralisation par comparaison avec ses principaux concurrents comme LWPR (voir (Sigaud *et al.*, 2011) pour une telle comparaison). Pour réaliser la régression, on fournit l'état du système comme entrée et l'action comme sortie à prédire. Le modèle appris propose donc une action pour tout état.

En réalisant cet apprentissage avec XCSF (figure 1 (b)), on engendre un contrôleur qui réalise une politique déterministe. Nous appelons « politique XCSF » cette politique et nous la notons $\pi_{\boldsymbol{\theta_0}}$, avec $\boldsymbol{\theta_0}$ un ensemble de paramètres caractérisant le modèle appris par XCSF.

Cette politique $\pi_{\boldsymbol{\theta_0}}$ est ensuite optimisée en utilisant CEPS (figure 1 (c)). Plus précisément, CEPS fait varier les paramètres $\boldsymbol{\theta}$ de la politique XCSF par essais et erreurs à la recherche d'un contrôleur plus performant.

2.3. XCSF

Il existe dans l'état de l'art deux grandes catégories de systèmes pour la régression de fonctions non linéaires $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$: ceux qui réalisent une combinaison linéaire de modèles gaussiens et ceux qui réalisent une combinaison gaussienne de modèles linéaires (Sigaud *et al.*, 2011). Le système de classeurs XCSF (Wilson, 2001 ; 2002) appartient à la seconde catégorie. Il s'agit d'une évolution du système XCS (Butz *et al.*, 2005 ; Wilson, 1995), le plus utilisé des systèmes de classeurs, vers l'approximation de fonctions par des méthodes de régression. XCSF est un outil de régression générique, qui peut apprendre des fonctions reliant des points sans faire d'hypothèse sémantique sur les espaces auxquels appartiennent ces points.

Comme tous les systèmes de classeurs, XCSF gère une population de règles appelées *classeurs*. Ces classeurs contiennent une partie *condition* et une partie *prédiction*. Dans XCSF, la partie *condition* définit la région de validité d'un modèle local de la fonction à approcher tandis que la partie *prédiction* contient le modèle local lui-même. XCSF peut utiliser différents types de modèles de prédiction (linéaire, quadratique, etc.) et peut paver l'espace d'approximation avec différentes familles de régions (gaussiennes, hyper-rectangles, etc.). Dans cet article, nous ne considérons que le cas des modèles de prédiction linéaires et des régions gaussiennes.

En pratique, chaque classeur possède un modèle linéaire β_i utilisé pour prédire le vecteur de sortie local $\boldsymbol{y}_i \in \mathcal{Y}$ relatif à un sous-ensemble d'un vecteur d'entrées $\boldsymbol{x}_i \in \mathcal{X}$. Ce modèle linéaire est mis à jour par l'algorithme des moindres carrés récursifs. Mais ce modèle linéaire local ne vaut que pour un sous-espace de l'espace d'entrée délimité par un domaine $\phi_i(\boldsymbol{z}_i)$, où $\boldsymbol{z}_i \in \mathcal{Z}$ est un vecteur d'un autre sous-

ensemble de l'espace des entrées. Les domaines $\phi_i(\boldsymbol{z}_i)$ associés à chaque modèle linéaire, représentés ici par des gaussiennes, sont optimisés par un algorithme génétique. Par ailleurs, dans les cas simples, comme celui étudié dans cet article, on a $\mathcal{X} = \mathcal{Z}$ qui couvre la totalité de l'espace d'entrée.

Ainsi, les classeurs d'XCSF forment une population P qui décompose l'espace des conditions en un ensemble de modèles gaussiens partiellement superposés auxquels sont associés des modèles de prédiction linéaires. XCSF utilise seulement une partie de ces classeurs pour engendrer une approximation. En effet, à chaque itération d'apprentissage, XCSF engendre un ensemble M qui contient tous les classeurs fiables dans la population P dont la partie *condition* $\mathcal{Z}$ est compatible avec les entrées $\boldsymbol{z}$, c'est-à-dire pour lesquelles $\phi_i(\boldsymbol{z})$ est au dessus d'un seuil ϕ_0 [1].

Dans XCSF, la sortie $\hat{\boldsymbol{y}}$ est donnée pour un couple $(\boldsymbol{x}, \boldsymbol{z})$ par la somme des modèles linéaires β_i de tous les classeurs i de M pondérés par leur fitness F_i.

$$\hat{\boldsymbol{y}}\left(\boldsymbol{x}, \boldsymbol{z}\right) = \frac{1}{F\left(\boldsymbol{z}\right)} \sum_{i=1}^{n_M} F_i\left(\boldsymbol{z}\right) \boldsymbol{\beta}_i\left(\boldsymbol{x}\right)$$

où $F\left(\boldsymbol{z}\right) = \sum_{i=1}^{n_M} F_i\left(\boldsymbol{z}\right)$ et n_M est le nombre de classeurs dans M. Par ailleurs, les mécanismes qui sous-tendent l'évolution de la population de classeurs sont directement hérités du système de classeurs le plus standard, XCS. Une description plus complète d'XCSF est disponible dans (Butz, Herbort, 2008 ; Butz *et al.*, 2004).

Dans le cadre de notre architecture, XCSF est utilisé pour apprendre par régression un modèle de la politique engendrée par le NOPS. Le vecteur de sortie $\boldsymbol{y}$ contient donc les commandes motrices à envoyer au bras, le vecteur d'entrée contient l'état $\boldsymbol{x}$ du système (décrit dans la section 3.1) et le vecteur de conditions $\boldsymbol{z}$ contient lui aussi l'état $\boldsymbol{x}$ du système. Lorsqu'on optimise le vecteur $\boldsymbol{\theta}$ des paramètres d'un contrôleur XCSF, en pratique ce sont les poids des modèles linéaires β_i des classeurs qui sont optimisés.

2.4. Méthode de recherche de politiques

Dans le cadre des PDM, le choix d'une action est déterminé pour tout état par une politique stationnaire $\pi : \mathcal{X} \rightarrow \Pi(\mathcal{U})$ qui associe à chaque état une distribution sur les actions : $\pi(\boldsymbol{u}_t|\boldsymbol{x}_t) = P(\boldsymbol{u}_t|\boldsymbol{x}_t, \pi)$. Dans cet article, nous considérons exclusivement les politiques déterministes que nous notons $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t)$ pour simplifier. Le nombre d'états et d'actions étant infini, on ne peut pas représenter de façon exacte une politique quelconque dans ce contexte. On se restreint donc à des politiques paramétriques, qui constituent un sous-ensemble des politiques possibles défini par la forme de la fonction paramétrique employée.

1. Ce seuil est noté θ_m dans (Butz, Herbort, 2008).

2.4.1. Politiques paramétriques

Une politique paramétrique permet de parcourir une famille de politiques définies par une fonction $\pi : \Theta \times \mathcal{X} \rightarrow \Pi(\mathcal{U})$ choisie *a priori*. Différentes politiques sont obtenues pour différents vecteurs $\boldsymbol{\theta} \in \Theta$, qui représentent les paramètres d'une politique $\pi_{\boldsymbol{\theta}}$.

Étant donné une distribution $\mathcal{P}_0$ d'états initiaux d'un système, la performance globale du contrôleur peut être exprimée par la fonction objectif

$$J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}\left[V^{\boldsymbol{\theta}}(\boldsymbol{x}) \mid \boldsymbol{x} \sim \mathcal{P}_0\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(x_t, u_t) | \, \boldsymbol{x}_0 \sim \mathcal{P}_0\right], \qquad (2)$$

Par conséquent, la politique optimale est celle qui maximise cette fonction objectif. Comme on ne sait pas résoudre dans le cas général $\pi^* = \text{argmax}_\pi \, J(\pi)$, on cherche plus simplement $\boldsymbol{\theta}^* = \text{argmax}_{\boldsymbol{\theta}} \, J(\pi_{\boldsymbol{\theta}})$ où $\boldsymbol{\theta}^*$ dénote l'ensemble de paramètres optimaux pour la forme de politique paramétrique employée. Résoudre ce problème peut être vu comme un problème d'optimisation stochastique dans l'espace des paramètres de la politique.

2.4.2. Recherche de politiques par descente de gradient

Une méthode classique pour optimiser une politique paramétrique $\pi_{\boldsymbol{\theta}}$ consiste à réaliser une montée de gradient sur la performance $J(\pi_{\boldsymbol{\theta}})$ de la politique suivant ses paramètres $\boldsymbol{\theta}$. De nombreux travaux d'apprentissage par renforcement se sont penchés sur ces méthodes (voir (Peters, Schaal, 2008) pour une vue d'ensemble). L'approche qui implique le moins d'hypothèses est la méthode des différences finies. Elle consiste simplement à calculer la performance de la politique pour différentes valeurs des paramètres et à suivre le gradient de cette performance. Des méthodes plus puissantes exploitent une dérivation analytique de la fonction objectif pour calculer le gradient plus efficacement. C'est le cas de REINFORCE (Williams, 1992) et des méthodes de « gradient naturel » (Peters, Schaal, 2008). Cependant, ces méthodes sont sensibles à de nombreux hyper-paramètres et difficiles à appliquer à des problèmes continus de grande taille.

Ces méthodes sont à présent dépassées par une autre famille de méthodes qui, au lieu d'utiliser un gradient, comparent un ensemble de politiques et les pondèrent par leur performance. C'est le cas de l'algorithme PoWER (Kober, Peters, 2008 ; Kormushev *et al.*, 2010) qui est basé sur la méthode *Expectation-Maximization* dont le cadre permet de combiner harmonieusement des problématiques d'apprentissage par renforcement et d'apprentissage par démonstration. C'est aussi le cas de l'algorithme « *Policy Improvement with Path Integrals* » (PI^2) (Theodorou *et al.*, 2010a) qui est moins contraint que PoWER et se montre plus efficace d'au moins un ordre de grandeur sur des applications élémentaires de type robotique (Theodorou *et al.*, 2010b). Ces algorithmes sont en général appliqués dans le cadre de tâches robotiques à des

primitives motrices dynamiques (DMP) [2] qui dirigent l'effecteur terminal du robot le long d'une trajectoire spécifique.

Dans (Stulp, Sigaud, 2012), les auteurs comparent conceptuellement l'algorithme PI^2 avec les méthodes d'optimisation stochastique CMA-ES et la méthode de l'entropie croisée (CEM) que nous utilisons ici. Il se dégage de cette comparaison que les méthodes d'optimisation stochastique telles que CMA-ES et CEM sont conceptuellement très proches de PI^2, ce qui permet d'établir une perspective unifiée pour les méthodes de recherche directes de politiques paramétriques. Ces conclusions sont convergentes avec celles de (Heidrich-Meisner, Igel, 2008) qui montrent expérimentalement la supériorité de CMA-ES sur les méthodes de recherche de politiques basées sur l'estimation d'un gradient.

2.4.3. Méthode de l'entropie croisée

La méthode de l'entropie croisée (Rubinstein, 1997) est une approche générale de type Monte Carlo qui peut être utilisée pour l'optimisation continue (Rubinstein, 1999). Cette méthode a déjà été utilisée pour de l'optimisation de politiques, d'abord par (Szita, Lörincz, 2006) et plus récemment par (Busoniu *et al.*, 2011) pour optimiser un ensemble de fonctions de base représentant un contrôleur dont la fonction de valeur est apprise par renforcement. Contrairement aux méthodes de gradient décrites ci-dessus, la méthode de l'entropie croisée n'impose pas que la fonction objectif soit différentiable ni même continue.

Au lieu de mettre à jour un seul vecteur de paramètres de la politique $\boldsymbol{\theta}$, la méthode de l'entropie croisée améliore itérativement une distribution sur les paramètres de la politique au vu de leur performance. A l'itération t, elle évalue un ensemble de politiques dont les paramètres sont tirés suivant une distribution f_t. A partir de ces évaluations, une distribution « améliorée » g_t sur les paramètres de la politique est calculée, de telle sorte qu'une politique dont la performance empirique est meilleure aura une probabilité plus importante dans cette distribution. Dans la méthode d'entropie croisée, la distribution g_t correspond à une sélection des S meilleures politiques parmi celles échantillonnées, avec $g_t(\boldsymbol{\theta}) = 1/S$ pour celles-ci et 0 pour toute autre politique. Pour finir, on choisit la nouvelle distribution f_{t+1} la plus proche possible de g_t en minimisant une mesure de distance entre ces deux distributions, l'entropie croisée $H(f_{t+1}, g_t) = \mathbb{E}_{\boldsymbol{\theta}\sim f_{t+1}}\left[-\log g_t(\boldsymbol{\theta})\right]$.

En contraignant les distributions f à suivre une loi normale, on obtient chaque nouvelle distribution f_{t+1} simplement en calculant la moyenne $\boldsymbol{\mu}_{t+1}$ et l'écart type $\boldsymbol{\sigma}^2_{t+1}$ sur l'ensemble des politiques sélectionnées à l'itération t (i.e., $\{\forall\boldsymbol{\theta}|g_t(\boldsymbol{\theta}) > 0\}$). De plus, (Boer *et al.*, 2005) suggère d'ajouter un terme de bruit σ^2_{noise} à l'écart type pour éviter une convergence prématurée. Cette méthode est illustrée sur la figure 2 inspirée de (Thiéry, 2010).

2. Pour *Dynamic Motor Primitives*

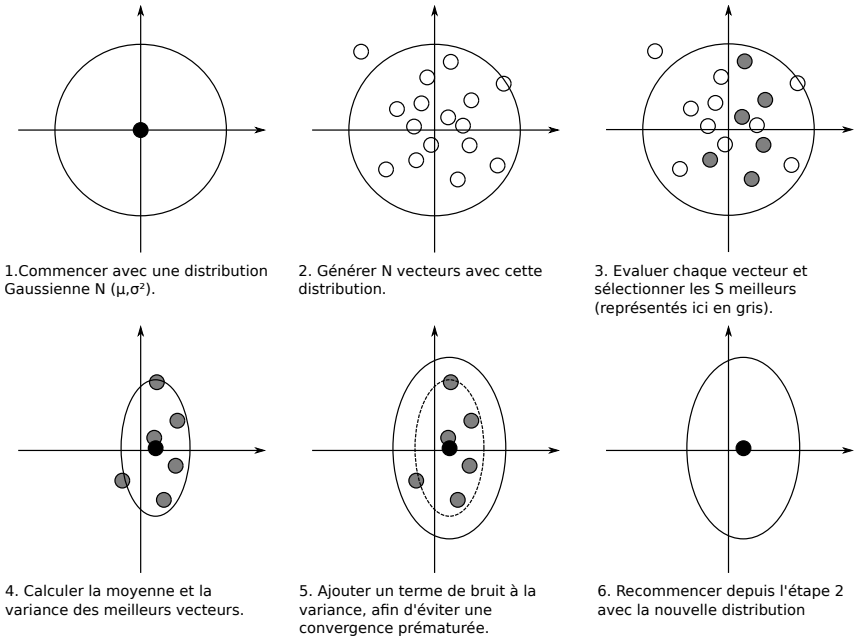

Figure 2. Vue schématique de la méthode de l'entropie croisée

2.4.4. Cross-Entropy Policy Search

Cross-Entropy Policy Search (CEPS, cf. Alg. 1) est un algorithme basé sur la méthode de l'entropie croisée et adapté à l'apprentissage de mouvements d'atteinte : la performance de chaque politique $\pi_{\boldsymbol{\theta}}$ est évaluée en moyenne pour un ensemble de cibles $\{\boldsymbol{x}^*_{(j)}\}_{j=1}^{M}$, correspondant à des états initiaux ou terminaux du système.

3. Etude expérimentale

Nous illustrons à présent l'utilisation d'XCSF et de CEPS pour apprendre à contrôler un bras actionné par 6 muscles de façon à atteindre diverses cibles avec son effecteur terminal.

3.1. Modèle de bras

Le système qui sert de support à nos expériences est un bras à deux degrés de liberté actionné par 6 muscles représenté sur la figure 3. Il s'agit d'une version simplifiée du modèle décrit dans (Li, 2006). Tous les angles sont exprimés en radians. Les équations de la dynamique figurent dans l'annexe A. L'espace d'état est constitué de la position articulaire de la cible $\boldsymbol{q}^*$, la position articulaire courante du bras $\boldsymbol{q}$ et sa vitesse courante $\dot{\boldsymbol{q}}$. L'état $\boldsymbol{x} = (\boldsymbol{q}^*; \boldsymbol{q}; \dot{\boldsymbol{q}})$ est donc de dimension 6.

Algorithme 1 Cross-Entropy Policy Search (CEPS)

ENTRÉES: $\{\boldsymbol{x}^*_{(j)}\}_{j=1}^{M}$: ensemble d'états initiaux ou cibles
$(\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0^2)$: moyenne et écart type de la distribution initiale sur les paramètres de la politique
σ^2_{noise}: terme de bruit additionnel
N : nombre de politiques à comparer à chaque itération
S : nombre de politiques à sélectionner pour chaque mise à jour
K : nombre d'itérations

pour $k = 1 \cdots K$ **faire**
  pour $i = 1 \cdots N$ **faire**
    Tire un vecteur de paramètres $\boldsymbol{\theta}_{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2)$
    pour $j = 1 \cdots M$ **faire**
      Réalise un épisode τ_j *pour l'état* $\boldsymbol{x}^*_{(j)}$ *suivant* $\pi_{\boldsymbol{\theta}_{(i)}}$
    fin pour
    Calcule la performance estimée de $\pi_{\boldsymbol{\theta}^{(i)}}$
    $J_{(i)} = \frac{1}{M} \sum_{j=1}^{M} \sum_{t=0}^{|\tau_j|-1} \gamma^t r_{j,t}$ *où* $r_{j,t}$ *est la récompense au temps* t *du* j^{ieme} *épisode*
  fin pour
  Déterminer l'ensemble $\mathcal{M}$ *des* S *meilleures politiques parmi* $\{\boldsymbol{\theta}_{(i)}\}_{i=0}^{N}$ *d'après leur performance respective* $\{J_{(i)}\}_{i=0}^{N}$
  $\boldsymbol{\mu}_{k+1} \leftarrow$ moyenne$(\mathcal{M})$
  $\boldsymbol{\sigma}^2_{k+1} \leftarrow$ écart type$(\mathcal{M}) + \sigma^2_{noise}$
fin pour
renvoyer les paramètres de la politique optimisée $\boldsymbol{\theta} = \boldsymbol{\mu}_{K+1}$

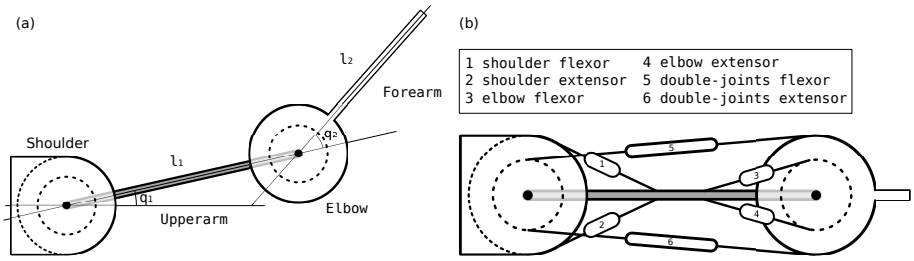

Figure 3. Modèle de bras. (a) Vue schématique de la mécanique du bras. (b) Vue schématique de l'actionnement musculaire, où chaque nombre représente un muscle dont le nom est donné en légende

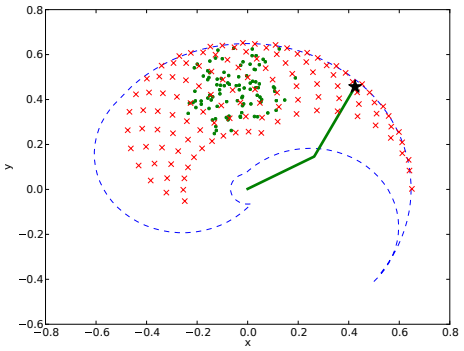

Figure 4. L'espace atteignable du bras est délimité par une enveloppe en forme de spirale. Les deux segments du bras sont représentés par deux lignes en gras. Le point $\mathcal{P}$ est représenté par une étoile, les points de $\mathcal{L}$ sont en vert et les points de $\mathcal{T}$ sont des croix rouges (voir section 3.2 pour la définition de ces ensembles)

Chacun des états initiaux $\boldsymbol{x}_0$ décrit donc une configuration particulière de la tâche par le point de départ $\boldsymbol{q}_{init}$ et la cible $\boldsymbol{q}^*$ à atteindre, ainsi qu'une vitesse initiale nulle pour toutes les articulations.

Nous définissons un état $\mathcal{P}$ qui correspond au point $\boldsymbol{q} = [q_1; q_2] = [0,5; 0,59]$, une vitesse nulle et une position de cible variable. Cet état sert alternativement d'état initial ou de cible selon les expériences.

Les positions sont bornées par l'espace atteignable par un bras humain moyen, avec $q_1 \in [-0,6; 2,6]$ et $q_2 \in [-0,2; 3,0]$, comme il apparaît sur la figure 4. L'espace d'action est défini par un signal d'activation pour chaque muscle, ce qui fait aussi 6 dimensions. L'action est perturbée par un bruit multiplicatif d'écart type $\sigma_u^2 = 0,02$. Le simulateur utilise la méthode d'Euler avec un pas de temps $\Delta t = 2$ ms et il est stoppé après H =1 000 pas de temps.

La tâche est accomplie quand la distance euclidienne entre l'effecteur terminal et la cible est inférieure à $0,01\ m$ et la vitesse euclidienne de l'effecteur terminal est inférieure à $0,1\ m.s^{-1}$. Ces seuils sont assez élevés pour éviter que l'exploration ne fasse perdre la cible, ce qui arrive fréquemment durant l'apprentissage. Pour le NOPS, ces seuils ne sont pas nécessaires car il est capable d'une précision arbitraire.

La fonction de coût, donnée par l'équation (1), est paramétrée par $\alpha = 200$, $\beta = 40$ et le facteur d'actualisation $\gamma = 0,998$.

A noter que les figures sont présentées dans l'espace cartésien $(x; y)$ alors que notre représentation d'état est calculée dans l'espace articulaire.

3.2. Expériences

Nous réalisons trois expériences.

Dans la première expérience, nous créons deux ensembles de cibles, $\mathcal{L}$ pour l'apprentissage et $\mathcal{T}$ pour le test, afin de mesurer la capacité de généralisation d'XCSF d'un ensemble à l'autre. L'ensemble $\mathcal{L}$ contient 50 cibles tirées selon une distribution normale centrée sur $x = -0,059$ et $y = 0,44$ avec un écart type de $0,1$ dans chaque dimension. Ces cibles correspondent à des positions moyennes où le bras se situe loin des butées articulaires. L'ensemble $\mathcal{T}$ contient 94 cibles dans une grille 10×10 qui couvre tout l'espace articulaire (sur les 100 trajectoires prévues initialement, le NOPS diverge pour 6 d'entre elles, qui sont exclues). On mesure donc la généralisation à des trajectoires extrêmes à partir de trajectoires moyennes.

Toutes les trajectoires partent du point $\mathcal{P}$ (voir figure 4). Le NOPS est utilisé pour engendrer une trajectoire pour chaque cible de $\mathcal{L}$. Ensuite, XCSF apprend les actions du NOPS pour ces cibles de $\mathcal{L}$ en utilisant les trajectoires engendrées. Les espaces des conditions et des prédictions correspondent aux états $\boldsymbol{x}$. La performance et les trajectoires obtenues avec $\pi_{\boldsymbol{\theta}_0}$ sont comparées à celles engendrées par le NOPS, pour voir comment le contrôleur $\pi_{\boldsymbol{\theta}_0}$ engendré par XCSF se comporte par rapport à une référence quasi optimale et pour mesurer la capacité de généralisation de $\pi_{\boldsymbol{\theta}_0}$ à d'autres cibles.

Dans la seconde expérience, les rôles des points de départ et des cibles sont inversés. Le bras démarre de $\mathcal{L}$ et $\mathcal{T}$ tandis que $\mathcal{P}$ est utilisé comme cible unique. Le reste du protocole est identique à celui de la première expérience.

La politique $\pi_{\boldsymbol{\theta}_0}$ est paramétrique puisque tous les classeurs ont des paramètres dans leurs parties *condition* et *prédiction*. On peut donc lui appliquer l'algorithme CEPS décrit plus haut pour optimiser sa performance. Chaque politique $\pi_{\boldsymbol{\theta}}$ qui résulte de cette application est un point sur la figure 2. En pratique, $\boldsymbol{\theta}$ ne contient que les poids des modèles locaux β_i. Dans une troisième expérience, la politique $\pi_{\boldsymbol{\theta}_0}$ obtenue à l'issue des expériences précédentes est optimisée en utilisant CEPS. Chaque politique est évaluée sur l'ensemble complet des cibles de test. La performance globale d'une politique est la moyenne des performances obtenues pour chaque cible. On ne réalise qu'un seul épisode pour évaluer la performance pour une configuration (point de départ et cible) donnée.

3.3. Paramétrage

Nous utilisons l'implémentation JavaXCSF (P. O. Stalph, Butz, 2009) de XCSF et tous les algorithmes sont codés en Java. Les expériences tournent sur un processeur Intel Core 2 Duo E8400 @ 3 GHz avec 4 Go de RAM.

Les paramètres d'XCSF sont les suivants. Le nombre d'itérations est fixé à un million et la taille maximale de la population à 500. Les entrées sont normalisées : les positions courante et cible sont bornées par l'espace atteignable et la vitesse est bornée par $[-100; +100]\ rad.s^{-1}$. L'action par défaut $\boldsymbol{u}_{default}$ est un vecteur nul qui ne génère pas d'activation musculaire. Après un réglage empirique des paramètres, le taux d'apprentissage α (appelé `beta` dans JavaXCSF) est réglé à 1, et la compaction

est désactivée [3]. On désactive le *multithreading* pour favoriser la reproductibilité des résultats.

Pour CEPS, le nombre d'itérations K est fixé de telle façon que l'algorithme tourne pendant deux heures sur le processeur décrit plus haut. Chaque itération consiste en l'évaluation de $N = 100$ politiques. Chaque configuration étant évaluée une fois seulement, le nombre d'épisodes M utilisé pour évaluer chaque politique est égal au nombre de configurations testées, à savoir $|\mathcal{T}| = 94$. La proportion des politiques sélectionnées par CEPS est fixée à $80\ \%$ des N politiques évaluées, soit $S = 80$ politiques. La variance initiale $\boldsymbol{\sigma}^2$ est fixée à $0,1$ et un bruit supplémentaire $\sigma^2_{noise} = 10^{-3}$ est ajouté à chaque itération.

4. Résultats

Nous examinons tout d'abord si la politique apprise avec XCSF est similaire à celle obtenue avec le NOPS pour les mêmes points de départ et cibles et nous mesurons la capacité de généralisation d'XCSF sur des points de départ et cibles différentes. Ensuite, nous mettons en évidence la capacité de CEPS à améliorer la performance de la politique apprise là où la généralisation n'a pas fourni une performance satisfaisante.

4.1. Performance de la politique XCSF

Les résultats présentés dans cette section sont similaires à ceux présentés dans (Marin *et al.*, 2011), bien que nous n'ayons utilisé que 500 classeurs au lieu de 6 400 dans l'article original. Le temps moyen d'exécution d'une trajectoire avec le NOPS est d'environ 10 minutes. Avec $\pi_{\boldsymbol{\theta}_0}$, il est d'environ 30 millisecondes.

Tableau 1. Moyenne et écart type de la performance donnée par (1) sur les cibles de $\mathcal{L}$ et $\mathcal{T}$. Une performance plus élevée et de moindre écart type est meilleure

départ / cible	$\mathcal{P}/\mathcal{L}$	$\mathcal{P}/\mathcal{T}$
NOPS	$28,22 \pm 2,06$	$27,46 \pm 3,73$
XCSF	$27,45 \pm 2,35$	$2,44 \pm 46,03$
CEPS	$22,96 \pm 11,50$	$12,76 \pm 22,28$

4.1.1. Un seul point de départ pour plusieurs cibles

La figure 5(a) montre la performance du NOPS en utilisant $\mathcal{P}$ comme position de départ et en fonction de la position des cibles, obtenue en interpolant la performance des trajectoires du NOPS sur $\mathcal{T}$. La figure 5(b) montre la performance d'une politique $\pi_{\boldsymbol{\theta}_0}$ représentative. Cette performance est très proche de celle du NOPS pour les cibles proches des cibles d'apprentissage. La performance pour des cibles plus éloignées est

3. `startCompaction=resetRLSPredictionsAfterSteps=2`

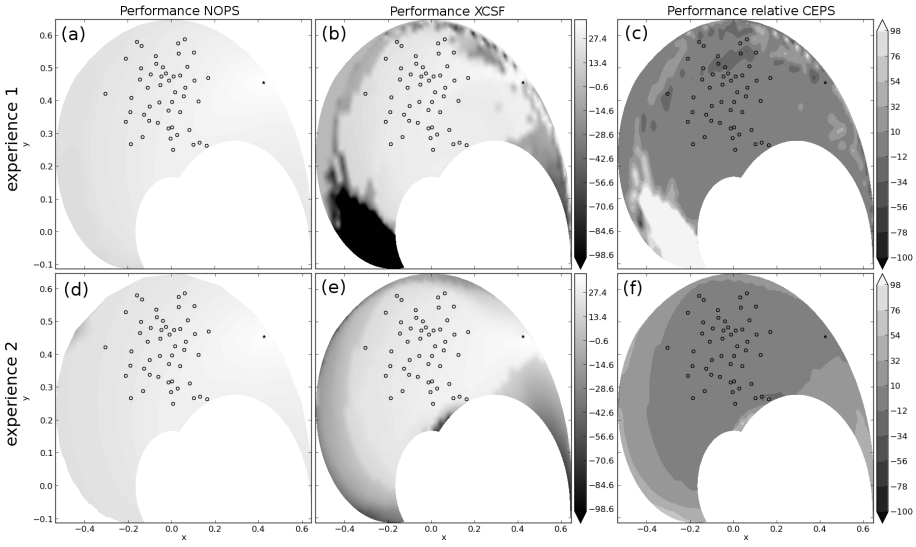

Figure 5. Performance du NOPS (a) et de π_{θ_0} (b), et performance relative par rapport à (b) de π_{θ_0} optimisée par CEPS *(c). La deuxième ligne correspond à la seconde expérience, où la cible est $\mathcal{P}$ et la performance varie en fonction de la position de départ. Ces graphiques sont obtenus en interpolant les performances à partir d'une grille de points de départ ou de cibles. L'ensemble $\mathcal{L}$ est représenté par des cercles blancs et $\mathcal{P}$ par une étoile. La performance est calculée conformément à (1) et (2), voir l'algorithme 1*

Tableau 2. Moyenne et écart type de la performance donnée par (1) sur les points de départ de l'expérience 2. Une performance plus élevée et de moindre écart type est meilleure

départ / cible	$\mathcal{L}/\mathcal{P}$	$\mathcal{T}/\mathcal{P}$
NOPS	$27,68 \pm 2,16$	$26,37 \pm 5,11$
XCSF	$27,51 \pm 2,67$	$3,99 \pm 28,84$
CEPS	$18,98 \pm 11,84$	$16,39 \pm 16,36$

généralement plus faible, mais elle est équivalente à celle du NOPS pour une partie significative des cibles.

4.1.2. Plusieurs points de départ pour une seule cible

La figure 5(d) montre la performance du NOPS en utilisant $\mathcal{P}$ comme cible et en fonction de la position des points de départ. La figure 5(e) montre la performance d'une politique π_{θ_0} représentative. On peut voir que la performance en généralisation est beaucoup plus régulière dans cette expérience que dans la précédente (figure 5(b)). Cette comparaison visuelle est prolongée dans la section suivante.

4.2. Performance après optimisation par CEPS

La figure 5(c) montre la performance de $\pi_{\boldsymbol{\theta}}$ après optimisation par CEPS. La performance a été globalement améliorée sur l'ensemble de l'espace atteignable. En particulier, la performance a été nettement améliorée dans la région autour du point $(x = -0,3; y = 0)$ où la politique $\pi_{\boldsymbol{\theta}_0}$ était très mauvaise. La figure 5(f) montre la performance de $\pi_{\boldsymbol{\theta}}$ après optimisation de $\boldsymbol{\theta}$ par CEPS. La performance est globalement améliorée sur tout l'espace atteignable. C'est le cas en particulier pour les régions extrêmes, où la performance était très mauvaise sur la figure 5(e).

Les résultats des tableaux 1 et 2 confirment quantitativement cette impression visuelle. La comparaison entre la colonne de gauche et la colonne de droite de ces tableaux montre que l'écart important de performance sur la totalité de l'espace atteignable $(\mathcal{P}/\mathcal{T})$ entre le NOPS et la politique XCSF est due principalement à quelques zones dans laquelle la performance de la politique XCSF est très faible. En effet, quand on ne considère que la région où se trouvent les cibles d'apprentissage, $(\mathcal{P}/\mathcal{L})$, on peut voir que la performance est beaucoup plus proche de l'optimum que lorsqu'on considère la performance du contrôleur XCSF sur tout l'espace. Par ailleurs, on voit que le recours à CEPS, bien que limité en raison du coût computationnel, permet de redresser significativement la performance sur tout l'espace. Mais, de façon intéressante, cette performance locale décroît dans la zone où XCSF avait bien appris. Ces points sont discutés dans la section 5.

Enfin, on peut constater que la performance du contrôleur XCSF et du contrôleur optimisé par CEPS est meilleure sur tout l'espace lorsqu'on travaille de tout point de départ vers une cible unique que lorsqu'on part d'un point de départ unique vers une multitude de cibles. En revanche, elle est plutôt moins bonne sur la zone d'apprentissage, si bien qu'il est difficile de conclure.

5. Discussion

Le principal bénéfice de l'utilisation de $\pi_{\boldsymbol{\theta}}$ comme contrôleur est qu'il est environ 20 000 fois plus rapide que le NOPS. Cela permet d'envisager l'exécution en temps-réel de mouvements quasi-optimaux pour des robots, ce qui est loin d'être le cas avec le NOPS. Par ailleurs, la capacité de généralisation d'XCSF, qui a été au centre de notre étude expérimentale, joue aussi un rôle très important pour l'applicabilité de notre méthode à la robotique. En effet, apprendre la politique XCSF serait extrêmement coûteux s'il fallait engendrer un grand nombre de trajectoires avec le NOPS pour arriver à une performance acceptable. Comme le montre la figure 5(b), quelques trajectoires suffisent pour obtenir une performance correcte. Cependant, comme nous l'avons mis en évidence dans la section 4.2, même si la performance est satisfaisante dans une région plus large que celle sur laquelle XCSF a appris, elle peut chuter considérablement à l'extérieur de la région d'apprentissage et sa variance est élevée. Disposer de davantage de trajectoires permettrait d'obtenir une meilleure généralisation, mais engendrerait un coût computationnel plus important. On peut donc faire un compromis

entre temps de calcul et qualité de la généralisation en fonction des contraintes de l'application.

Le fait que la performance soit plus faible et que la variance soit élevée en dehors de la région d'apprentissage n'est pas surprenant. En effet, XCSF doit extrapoler à partir de sa connaissance locale, ce qui peut être très difficile dans des régions où le comportement du système est très différent de celui de la région d'apprentissage. De ce point de vue, utiliser CEPS pour améliorer la performance a des conséquences intéressantes. Tout d'abord, pour optimiser les paramètres, CEPS engendre des trajectoires supplémentaires réparties sur la totalité de l'espace atteignable, donc il intègre de la connaissance du comportement du système sur des régions où XCSF n'a pas appris. Cette exploration supplémentaire est réalisée en utilisant la politique XCSF, donc le coût supplémentaire est limité par rapport à un usage plus large du NOPS. Cependant, comme CEPS optimise un critère global sur la totalité de l'espace atteignable, cette optimisation en moyenne se traduit aussi par une perte locale de performance dans la région d'apprentissage où XCSF avait produit une politique très efficace.

Pour atténuer cet effet, nous avons défini dans (Marin, Sigaud, 2012) une méthode locale dérivée de CEPS (nommée LCEPS) qui réalise l'adaptation à raison d'une cible à la fois et qui définit un critère local améliorant la performance relativement à cette unique cible. Pour cela, LCEPS agit préférentiellement sur les classeurs qui ont le plus besoin d'être corrigés, sans modifier les classeurs efficaces. Ces travaux ne sont pas décrits davantage ici, nous renvoyons le lecteur à (Marin, Sigaud, 2012) pour plus de détails. Il faut toutefois souligner la différence de fond entre l'application de LCEPS à la politique XCSF et l'application beaucoup plus classique dans la littérature des algorithmes PoWER et PI^2 à une ou plusieurs DMP. Les méthodes d'amélioration de politiques appliquées à des DMP sont conçues pour améliorer le comportement d'un robot le long d'une trajectoire spécifique à partir d'un unique état initial. A ce titre, elles ne disposent a priori que d'une capacité de généralisation très limitée à des trajectoires et des cibles plus ou moins proches. Au contraire, LCEPS peut être utilisé pour optimiser (localement) une politique qui définit un comportement sur tout l'espace atteignable du système considéré.

Enfin, on peut regretter que les méthodes présentées dans cet article soient assujetties au réglage empirique d'un grand nombre de paramètres. Il s'agit d'un inconvénient commun à la plupart de ces méthodes d'apprentissage et d'optimisation pour des systèmes continus, dont notre approche fait partie.

6. Conclusion

Dans cet article, nous avons montré sur un problème simulé de dimension modeste comment on pouvait apprendre une politique continue en états et en actions qui soit efficace en termes de coût, sur la base de démonstrations réalisées par un système de planification quasi optimal beaucoup plus coûteux à mettre en œuvre. Dans le cadre des expériences que nous avons réalisées, le contrôleur qui en résulte est 20 000 fois plus rapide que l'original et la capacité de généralisation d'XCSF simplifie notable-

ment le processus d'apprentissage. De son côté, l'algorithme CEPS bénéficie de la robustesse des méthodes d'optimisation stochastique et peut adapter le contrôleur à des changements dans l'environnement. Enfin, l'extension LCEPS permet de réaliser cette adaptation localement, en fonction de trajectoires réalisées par le système.

Cette démarche pourrait être appliquée à des systèmes robotiques de dimension équivalente et éventuellement étendue à des systèmes de plus grande dimension, dans les limites des capacités d'XCSF et CEPS à couvrir correctement des espaces de très grande taille. Une dizaine de dimensions pour l'état et pour l'action semblent largement atteignables, ce qui permet de résoudre des problèmes robotiques tout à fait significatifs.

Cependant, notre étude expérimentale a montré que, même dans le cadre d'une application simulée, la performance obtenue n'est pas encore assez proche de celle du NOPS utilisé pour apprendre. Pour améliorer encore cette performance, les options les plus évidentes sont les suivantes.

Tout d'abord, pour obtenir une amélioration significative, CEPS a besoin d'engendrer un grand nombre de trajectoires. Au lieu de les engendrer à partir du système lui-même, ce qui peut être très coûteux si le système est un robot réel, on peut plutôt apprendre un modèle de ce système, tel que cela a été fait dans (Salaün *et al.*, 2010 ; P. Stalph *et al.*, 2010), et utiliser ce modèle pour améliorer la politique.

Ensuite, dans les travaux présentés ici, on effectue une phase d'apprentissage par démonstration avec XCSF suivie d'une phase d'optimisation de la politique en utilisant CEPS. A présent que nous disposons de la variante locale de CEPS nommée LCEPS, il serait intéressant de mettre en place une méthode d'apprentissage actif qui permette de déterminer quelle méthode d'apprentissage appliquer (apprentissage par démonstration ou optimisation) et sur quelle cible s'entraîner en fonction de l'état courant du contrôleur et de ses progrès.

A plus long terme, nous envisageons d'appliquer à des problèmes robotiques les méthodes présentées ici sur un système simulé. Notamment, dans le cadre du projet AAL DOMEO, nous étudions s'il est possible d'appliquer ces méthodes au problème consistant à adapter à un sujet particulier la trajectoire d'un robot d'assistance à la verticalisation (Pasqui *et al.*, 2010). Par ailleurs, nous aimerions mettre en œuvre les méthodes décrites ici pour commander le robot humanoïde iCub de façon à lui faire décrire des mouvements ressemblant à ceux des humains.

Remerciements

Ce travail est financé par le « Ambient Assisted Living Joint Programme of the European Union and the National Innovation Office (DOMEO-AAL-2008-1-159) », voir http://www.aal-domeo.eu.

Bibliographie

Argall B. D., Chernova S., Veloso M., Browning B. (2009, mai). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, vol. 57, n° 5, p. 469–483.

Bertsekas D. P. (1995). *Dynamic Programming and Optimal Control*. Belmont, MA, Athena Scientific.

Boer P.-T. de, Kroese D. P., Mannor S., Rubinstein R. Y. (2005). A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, vol. 134, n° 1, p. 19–67. Consulté sur http://www.springerlink.com/index/10.1007/s10479-005-5724-z

Busoniu L., Ernst D., De Schutter B., Babuska R. (2011). Cross-entropy optimization of control policies with adaptive basis functions. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, n° 1, p. 196–209.

Butz M. V., Goldberg D., Lanzi P. (2005). Computational Complexity of the XCS Classifier System. *Foundations of Learning Classifier Systems*, vol. 51, p. 91–125.

Butz M. V., Herbort O. (2008). Context-dependent predictions and cognitive arm control with XCSF. In *Proceedings of the 10th annual conference on genetic and evolutionary computation*, p. 1357-1364. ACM New York, NY, USA.

Butz M. V., Kovacs T., Lanzi P. L., Wilson S. W. (2004). Toward a theory of generalization and learning in xcs. *IEEE Transactions on Evolutionary Computation*, vol. 8, n° 1, p. 28–46.

Guigon E., Baraduc P., Desmurget M. (2008). Optimality, stochasticity and variability in motor behavior. *Journal of Computational Neuroscience*, vol. 24, n° 1, p. 57-68.

Heidrich-Meisner V., Igel C. (2008). Similarities and differences between policy gradient methods and evolution strategies. In *Proceedings of the 16th european symposium on artificial neural networks (esann)*. Citeseer.

Kober J., Peters J. (2008). Policy search for motor primitives in robotics. *Advances in Neural Information Processing Systems (NIPS)*, p. 1–8. Consulté sur http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.148.7472\&rep=rep1\&type=pdf

Kormushev P., Calinon S., Caldwell D. G. (2010). Robot Motor Skill Coordination with EM-based Reinforcement Learning. In *Proc. of ieee/rsj intl conf. on intelligent robots and systems (iros-2010)*.

Li W. (2006). *Optimal control for biological movement systems*. Thèse de doctorat non publiée, University of California, San Diego.

Mannor S., Rubinstein R., Gat Y. (2003). The cross entropy method for fast policy search. In *Proceedings icml*, vol. 20, p. 512.

Marin D., Decock J., Rigoux L., Sigaud O. (2011). Learning cost-efficient control policies with xcsf: generalization capabilities and further improvement. In *Proceedings of the 13th annual conference on genetic and evolutionary computation*, p. 1235–1242.

Marin D., Sigaud O. (2012). Towards fast and adaptive optimal control policies for robots: A direct policy search approach. In *Proceedings robotica*, p. 21-26. Guimaraes, Portugal.

Pasqui V., Saint-Bauzel L., Sigaud O. (2010, june). Characterization of a least effort user-centered trajectory for sit-to-stand assistance user-centered trajectory for sit-to-stand assistance. In *Proceedings iutam*.

Peters J., Schaal S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks : the official journal of the International Neural Network Society*, vol. 21, n° 4, p. 682–97. Consulté sur http://www.ncbi.nlm.nih.gov/pubmed/18482830

Rigoux L., Sigaud O., Terekhov A., Guigon E. (2010). Movement duration as an emergent property of reward directed motor control. In *Proceedings of the annual symposium advances in computational motor control.*

Rubinstein R. Y. (1997). Optimization of computer simulation models with rare events. *European Journal of Operational Research*, vol. 99, n° 1, p. 89–112. Consulté sur http://www.sciencedirect.com/science/article/B6VCT-3SWY0P1-X/2/9268866f182f6800f26b042ca64421f5

Rubinstein R. Y. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, vol. 1, n° 2, p. 127–190.

Salaün C., Padois V., Sigaud O. (2010). Learning forward models for the operational space control of redundant robots. In J. Peters, O. Sigaud (Eds.), *From motor learning to interaction learning in robots*, vol. 264, p. 169-192. Heidelberg, Springer.

Sigaud O., Buffet O. (2008). *Processus décisionnels de markov en intelligence artificielle*. Paris, Lavoisier.

Sigaud O., Salaün C., Padois V. (2011). On-line regression algorithms for learning mechanical models of robots: a survey. *Robotics and Autonomous Systems*, vol. 51, p. 1117-1125.

Stalph P., Rubinsztajn J., Sigaud O., Butz M. (2010). A Comparative Study: Function Approximation with LWPR and XCSF. In *Proceedings of the 13th international workshop on advances in learning classifier systems.*

Stalph P. O., Butz M. V. (2009). *Documentation of JavaXCSF*. Rapport technique. COBOSLAB.

Stulp F., Sigaud O. (2012). Path integral policy improvement with covariance matrix adaptation. In *Proceedings icml*, p. 1-8. Edinburgh, Scotland.

Szita I., Lörincz A. (2006). Learning Tetris using the noisy cross-entropy method. *Neural computation*, vol. 18, n° 12, p. 2936–41. Consulté sur http://www.ncbi.nlm.nih.gov/pubmed/17052153

Theodorou E., Buchli J., Schaal S. (2010a). A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, vol. 11, p. 3137-3181.

Theodorou E., Buchli J., Schaal S. (2010b). Reinforcement learning of motor skills in high dimensions: A path integral approach. In *Proceedings of the ieee international conference on robotics and automation.*

Thiéry C. (2010). *Itération sur les Politiques Optimiste et Apprentissage du Jeu de Tetris*. Thèse de doctorat non publiée, Université Henri Poincaré - Nancy 1. Consulté sur http://tel.archives-ouvertes.fr/tel-00550081/

Todorov E., Jordan M. I. (2002). Optimal feedback control as a theory of motor coordination. *Nature Neurosciences*, vol. 5, n° 11, p. 1226-1235.

Williams R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, vol. 8, n° 3-4, p. 229–256. Consulté sur http://www.springerlink.com/index/10.1007/BF00992696

Wilson S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, vol. 3, n° 2, p. 149–175.

Wilson S. W. (2001). Function approximation with a classifier system. In *Proceedings of the genetic and evolutionary computation conference (gecco-2001)*, p. 974–981. San Francisco, California, USA, Morgan Kaufmann.

Wilson S. W. (2002). Classifiers that Approximate Functions. *Natural Computing*, vol. 1, n° 2-3, p. 211-234.

Annexe. Dynamique du bras

Le tableau 3 fournit la nomenclature des paramètres et variables du modèle.

Tableau 3. Nomenclature : paramètres du modèle de bras

m_i	masse du segment i (kg)
l_i	longueur du segment i (m)
s_i	inertie du segment i $(kg.m^2)$
d_i	distance entre le centre du segment i et son centre de masse (m)
κ	paramètre du filtre de Heaviside
$\boldsymbol{A}$	matrice des moments du bras
$\boldsymbol{T}$	tension musculaire
$\boldsymbol{M}$	matrice d'inertie
$\boldsymbol{J}$	matrice Jacobienne
$\boldsymbol{C}$	force de Coriolis
$\boldsymbol{\tau}$	couples aux segments $(N.m)$
$\boldsymbol{B}$	amortissement
$\boldsymbol{u}$	activation musculaire (action)
σ_u^2	bruit musculaire multiplicatif
$\tilde{\boldsymbol{u}}$	activation musculaire filtrée
$\boldsymbol{q}^*$	position articulaire visée (rad)
$\boldsymbol{q}$	position articulaire courante (rad)
$\dot{\boldsymbol{q}}$	vitesse articulaire courante $(rad.s^{-1})$

Étant donné l'état courant $\boldsymbol{x}_t = (\boldsymbol{q}^*, \boldsymbol{q}, \dot{\boldsymbol{q}})$ et l'action $\boldsymbol{u}_t$, la dynamique du bras est calculée conformément aux équations suivantes :

$m_1 = 1,4$, $m_2 = 1,1$, $l_1 = 0,30$, $l_2 = 0,35$, $s_1 = 0,11$, $s_2 = 0,16$, $d_1 = 0,025$, $d_2 = 0,045$, $\kappa = 25$, $k_1 = d_1 + d_2 + m_2 l_1^2$, $k_2 = m_2 l_1 s_2$, $k_3 = d_2$,

$$\boldsymbol{A} = \begin{bmatrix} 0,04 & -0,04 & 0 & 0 & 0,028 & -0,035 \\ 0 & 0 & 0,025 & -0,025 & 0,028 & -0,35 \end{bmatrix}^\top,$$

$$\boldsymbol{T} = \begin{bmatrix} 700 & 382 & 572 & 445 & 159 & 318 \end{bmatrix},$$

$$\boldsymbol{M} = \begin{bmatrix} k_1 + 2k_2\cos(q_2) & k_3 + k_2\cos(q_2) \\ k_3 + k_2\cos(q_2) & k_3 \end{bmatrix},$$

$$\boldsymbol{J} = \begin{bmatrix} -l_1 \sin(q_1) - l_2 \sin(q_1 + q_2) & -l_2 \sin(q_1 + q_2) \\ l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) & l_2 \cos(q_1 + q_2) \end{bmatrix},$$

$$\boldsymbol{C} = \begin{bmatrix} -\dot{q}_2(2\dot{q}_1 + \dot{q}_2)k_2 \sin(q_2) & \dot{q}_1^2 k_2 \sin(q_2) \end{bmatrix},$$

$$\boldsymbol{B} = \begin{bmatrix} .05 & .025 \\ .025 & .05 \end{bmatrix} \dot{\boldsymbol{q}}_t,$$

$$\tilde{\boldsymbol{u}} = \log(\exp(\kappa \times \boldsymbol{u}_t \times (1 + \mathcal{N}(0, \boldsymbol{I}\sigma_u^2))) + 1)/\kappa,$$

$$\boldsymbol{\tau} = \boldsymbol{A}^\top (\boldsymbol{T} \times \tilde{\boldsymbol{u}}),$$

$$\partial \boldsymbol{q}^*/\partial t = 0, \qquad \partial \dot{\boldsymbol{q}}/\partial t = \boldsymbol{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{C} - \boldsymbol{B}), \qquad \partial \boldsymbol{q}/\partial t = \dot{\boldsymbol{q}},$$

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \partial \boldsymbol{x}_t/\partial t \times \Delta t,$$

où $\times$ représente la multiplication terme à terme et $\boldsymbol{I}$ est la matrice identité 6×6.