

Apprentissage de politiques efficaces avec XCSF et CEPS [★]

Didier Marin, Jérémie Decock, Lionel Rigoux et 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, jeremie.decock@gmail.com, rigoux@isir.upmc.fr, olivier.sigaud@upmc.fr

Résumé : Nous proposons dans cette contribution une méthode qui permet d'obtenir une politique efficace dans un cadre où l'état et l'action sont continus. Le système contrôlé est un bras à deux degrés de liberté actionné par six muscles. Nous apprenons par démonstration une politique paramétrique avec le système de classeurs XCSF à partir de trajectoires quasi-optimales et nous étudions la capacité d'XCSF à généraliser ce qu'il a appris le long de ces trajectoires sur l'ensemble de l'espace atteignable. De plus, nous montrons qu'une méthode d'optimisation stochastique appelée *Cross-Entropy Policy Search* permet d'améliorer encore la performance du contrôleur paramétrique.

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. L'approche la plus directe de ce problème consiste à adapter pour le continu les outils conceptuels de l'apprentissage par renforcement mis au point dans un premier temps pour le cadre discret. Cette approche engendre une activité de recherche importante et, en particulier, les approches de recherche directe d'une politique produisent des résultats de plus en plus satisfaisants (voir par exemple (Kober & Peters, 2008; Kormushev *et al.*, 2010)). Cependant, ces méthodes sont complexes, elles nécessitent un important effort d'ajustement de paramètres et sont sujettes à des problèmes de minima locaux. Dans (Lanzi, 2002), Lanzi présentait les systèmes de classeurs (LCS) comme des systèmes d'apprentissage par renforcement dotés d'une propriété de généralisation sur les variables non pertinentes. Dans la littérature récente sur ces systèmes, il y a eu quelques tentatives pour les appliquer au cas des actions continues, mais sur des problèmes jouets et avec des résultats très limités (Tran *et al.*, 2007; Wilson, 2006).

Dans le même temps, la recherche sur les LCS s'est significativement réorientée vers les problèmes d'apprentissage supervisé, qu'il s'agisse de classification ou de régression (voir par exemple (Bernadó-Mansilla *et al.*, 2001)). L'un des LCS les plus utilisés, XCSF, peut être vu comme le représentant ultime de cette évolution. En effet, il s'agit d'un outil générique et compétitif d'approximation de fonctions qui repose sur des méthodes de régression et dont tous les mécanismes de décision séquentielle que l'on trouve classiquement dans les LCS ont été supprimés.

Pourtant, on peut résoudre des problèmes de décision séquentielle ou de commande avec des méthodes d'apprentissage supervisés en utilisant le paradigme de l'apprentissage par démonstration (Argall *et al.*, 2009). Cette approche est plus simple que l'approche directe, sous réserve que l'on dispose d'une solution non adaptative au problème que l'on veut apprendre à résoudre.

Cet article présente une instanciation d'une telle approche, où les capacités de régression d'XCSF sont combinées avec un algorithme d'optimisation stochastique pour apprendre une politique par démonstration puis l'optimiser en ajustant ses paramètres.

Le contexte plus général de cette étude est la conception d'un modèle computationnel de la consolidation des habiletés motrices pour la réalisation de mouvements d'atteinte. Dans (Rigoux *et al.*,

★. Cet article est la traduction française d'un article accepté à GECCO 2011

2010), les auteurs présentent un modèle des mouvements d'atteinte qui repose sur un algorithme de calcul variationnel très coûteux. Une propriété cruciale du modèle est que les mouvements produits sont indépendants du temps. Cela se traduit par la possibilité d'apprendre des politiques stationnaires à partir du modèle.

Nous nous focalisons donc sur la capacité d'XCSF à apprendre des telles politiques stationnaires à partir de trajectoires fournies par le modèle préalable. En particulier, nous montrons que la tendance d'XCSF 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 XCSF peut être amélioré par un algorithme d'optimisation stochastique qui agit sur les paramètres définissant ce contrôleur. Ce processus supplémentaire peut être vu comme une façon de réintroduire de l'apprentissage par renforcement dans XCSF. Nous étudions expérimentalement 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 modèle lui-même et 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 ce travail.

2 Méthodes

Dans cette section, nous décrivons les méthodes utilisées pour réaliser notre modèle. Nous commençons par une rapide présentation de la méthode de commande optimale utilisée dans (Rigoux *et al.*, 2010), nous présentons l'algorithme *Cross-Entropy Policy Search* (CEPS) utilisé pour optimiser le contrôleur paramétrique, puis nous donnons un aperçu d'XCSF et de son utilisation pour apprendre le contrôleur optimisé par CEPS.

2.1 Commande optimale des mouvements d'atteinte

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 réaliser une tâche à un système tout en optimisant une fonction objectif. Au pas de temps t , l'état du système est un vecteur $\mathbf{x}_t \in \mathcal{X}$ et l'action est un vecteur $\mathbf{u}_t \in \mathcal{U}$. La fonction objectif $r : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$ représente l'intérêt de réaliser une action donnée dans un état donné, relativement au problème global à résoudre. 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.

La qualité d'une politique π pour un état \mathbf{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 :

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

où γ est le facteur d'actualisation relatif à l'incertitude sur le futur.

2.1.2 Un modèle des mouvements d'atteinte

Dans (Rigoux *et al.*, 2010), les auteurs proposent un modèle des mouvements d'atteinte chez l'homme 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 :

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

où α est le poids du terme d'effort, g est une fonction nulle partout sauf au but où elle vaut 1 et β est le poids sur le terme de récompense. La politique déterministe et quasi-optimale correspondante est obtenue par 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. Comme la politique ne prend pas en compte le 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.

Dans la suite, nous appelons *Quasi-Optimal Planning System* (QOPS) ce contrôleur. En effet, 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.

2.2 Méthode de recherche de politiques

Dans l'approche de commande décrite ci-dessus, les actions ne sont calculées que le long de la trajectoire effectivement réalisée par le système. Au contraire, 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 : $\pi(\mathbf{u}_t | \mathbf{x}_t) = P(\mathbf{u}_t | \mathbf{x}_t, \pi)$. Dans cet article, nous considérons exclusivement les politiques déterministes que nous notons $\mathbf{u}_t = \pi(\mathbf{x}_t)$ pour simplifier. Le nombre d'états et d'actions étant infini, on ne peut pas représenter de façon exacte une politique non-triviale dans ce contexte. C'est pourquoi on fait appel à des politiques paramétriques.

2.2.1 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 Θ . Nous notons π_θ la politique paramétrée par $\theta \in \Theta$ et nous écrivons simplement θ quand π_θ est utilisée comme paramètre.

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 :

$$J(\theta) = \mathbb{E} [V^\theta(\mathbf{x}) | \mathbf{x} \sim \mathcal{P}_0]$$

Par conséquent, la politique optimale est celle qui maximise cette fonction objectif. Comme nous nous restreignons à une famille de politique paramétrées, nous devons chercher un optimum relatif à cette famille

$$\theta^* = \operatorname{argmax}_\theta J(\theta)$$

où θ^* dénote l'ensemble de paramètres optimaux. Résoudre ce problème peut être vu comme un problème d'optimisation stochastique continue.

2.2.2 Recherche de politiques par descente de gradient

Une méthode classique pour optimiser une politique paramétrique consiste à réaliser une descente de gradient sur l'espace des paramètres. De nombreux travaux d'apprentissage par renforcement se sont penchés sur cette classe de problèmes (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 font l'hypothèse que la dérivée de la fonction objectif par rapport aux paramètres est connue, ce qui permet de calculer le gradient plus efficacement. C'est le cas de REINFORCE (Williams, 1992), des méthodes de « gradient naturel » (Peters & Schaal, 2008) ou de l'algorithme PoWER (Kober & Peters, 2008). Ce dernier est basé sur la méthode *Expectation-Maximization* qui, entre autres propriétés intéressantes, peut être utilisée pour combiner harmonieusement des problématiques d'apprentissage par renforcement et d'apprentissage par démonstration. Cependant, malgré des résultats convaincants en robotique (Kober & Peters, 2008; Kormushev *et al.*, 2010), ces méthodes sont complexes, sensibles à de nombreux hyper-paramètres et difficiles à appliquer à des problèmes continus de grande taille.

2.2.3 La méthode d'entropie croisée

La méthode d'entropie croisée (Rubinstein, 1997) est une approche générale de type Monte-Carlo qui peut être utilisée pour de l'optimisation continue (Rubinstein, 1999). Contrairement aux méthodes de gradient décrites ci-dessus, elle n'impose pas que la fonction objectif soit différentiable ni même continue.

Algorithme 1 Cross-Entropy Policy Search

Entrées: (μ_0, σ_0^2) : moyenne et écart-type initiaux de la distribution des paramètres

ρ : proportion d'échantillons dominants utilisés pour la mise à jour

$\bar{\sigma}^2$: terme de bruit additionnel

T : nombre d'itérations

N : nombre d'échantillons de paramètres à tirer au hasard

M : nombre d'essais pour chaque évaluation des politiques

H : horizon temporel des simulations

Sorties: paramètres optimisés $\theta = \mu_T$

pour $t = 1 \dots T - 1$ **faire**

$\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(N)} \leftarrow \mathcal{N}(\mu_t, \sigma_t^2)$

pour $i = 1 \dots N$ **faire**

Réalise M simulations utilisant la politique $\pi_{\theta^{(i)}}$ jusqu'à l'horizon H

Calcule la performance estimée de $\pi_{\theta^{(i)}}$

$J^{(i)} = \frac{1}{M} \sum_{j=1}^M \sum_{h=0}^{H-1} \gamma^h r_{j,h}$ où $r_{j,h}$ est la récompense au temps h du $j^{\text{ième}}$ épisode

fin pour

Trie les échantillons de paramètres $\theta^{(i)}$ selon $J^{(i)}$ et calcule l'ensemble \mathcal{S}_ρ des ρ meilleurs échantillons

$\mu_{t+1} \leftarrow \text{moyenne}(\mathcal{S}_\rho)$

$\sigma_{t+1}^2 \leftarrow \text{ecart-type}(\mathcal{S}_\rho) + \bar{\sigma}^2$

fin pour

Au lieu de chercher une seule solution qui est mise à jour au fil des itérations, la méthode d'entropie croisée améliore une distribution $f \in \mathcal{F}$ sur des solutions $w \in \mathcal{W}$ au vu de leur performance. A l'itération t , elle calcule une nouvelle distribution f_{t+1} en rapprochant la distribution courante f_t autant que possible de la distribution $g_t = \{\forall w | S(w) > \gamma_t\}$ des solutions dont l'évaluation est au-dessus d'un seuil γ_t . La distance entre les distributions s'exprime en terme d'entropie croisée, elle minimise $H(f_{t+1}, g_t) = \mathbb{E}_{f_{t+1}} [-\log g_t]$. Ce calcul est réalisé en évaluant N échantillons de f_t , qui fournissent une estimation de $H(f_{t+1}, g_t)$ par un échantillonnage basé sur l'importance. Chaque seuil γ_t est calculé en utilisant ces échantillons, en conservant une proportion $\rho \in]0, 1[$ des meilleurs échantillons.

En faisant l'hypothèse que les distributions \mathcal{F} suivent une loi normale, on obtient f_{t+1} simplement en calculant la moyenne μ_t et l'écart-type σ_t^2 sur les ρ meilleurs échantillons de f_t . De plus, de Boer *et al.* (2005) suggère d'ajouter un terme de bruit $\bar{\sigma}^2$ à l'écart-type pour éviter une convergence prématurée. Cette méthode est illustrée sur la figure 1 empruntée à (Thiéry, 2010).

2.2.4 Cross-Entropy Policy Search

Cross-Entropy Policy Search (CEPS) est un algorithme d'entropie croisée que nous appliquons à la recherche d'une politique, utilisant les paramètres θ en tant que solutions w et le critère de performance J en tant que fonction objectif S . Le processus complet est décrit par l'algorithme 1.

Cette approche dite de *ranking* a été proposé dans (Szita & Lörincz, 2006) pour apprendre une politique optimale à Tetris. D'autre part, Heidrich-Meisner & Igel (2008) ont effectué une comparaison expérimentale de CMA-ES, une autre méthode de *ranking*, aux méthodes de l'art de recherche de politiques, basées sur l'estimation d'un gradient. Leur résultat indique que les méthodes de *ranking* sont compétitives tout en nécessitant moins de réglages. Plus récemment, Busoniu *et al.* (2011) ont proposé une approche similaire à CEPS pour des problèmes à états continus et actions discrètes.

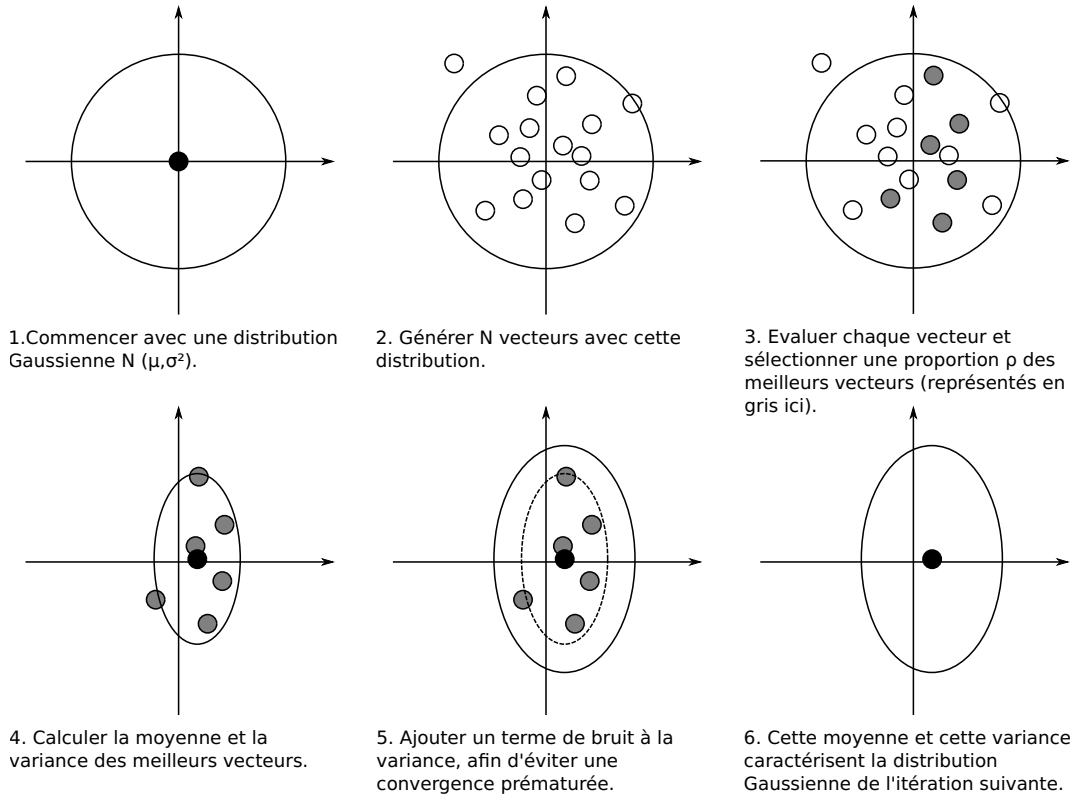


FIGURE 1 – Vue schématique de la méthode d'entropie croisée

2.3 XCSF

Le système de classeurs XCSF (Wilson, 2001, 2002) est une évolution du système XCS (Butz *et al.*, 2005; Wilson, 1995), le plus utilisé des systèmes de classeurs, vers l'approximation de fonctions. 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.

Un classeur définit un domaine $\phi_i(\mathbf{z})$ et possède un modèle linéaire correspondant β_i utilisé pour prédire un vecteur de sortie local \mathbf{y}_i relatif à un vecteur d'entrée \mathbf{x}_i . Le modèle linéaire est mis à jour par l'algorithme des moindres carrés récursifs.

Les classeurs d'XCSF forment une population P qui décompose l'espace des conditions en un ensemble de modèles de prédiction partiellement superposés. XCSF utilise seulement une partie de ses 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 \mathbf{z} , c'est-à-dire pour lesquelles $\phi_i(\mathbf{z})$ est au dessus d'un seuil ϕ_0 ¹.

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

$$\hat{\mathbf{y}}(\mathbf{x}, \mathbf{z}) = \frac{1}{F(\mathbf{z})} \sum_{i=1}^{n_M} F_i(\mathbf{z}) \hat{\mathbf{y}}_i(\mathbf{x}) \quad (2)$$

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

où $F(\mathbf{z}) = \sum_{i=1}^{n_M} F_i(\mathbf{z})$ 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 d'XCS. Une description plus complète d'XCSF est disponible dans (Butz & Herbort, 2008; Butz *et al.*, 2004).

2.4 Architecture globale

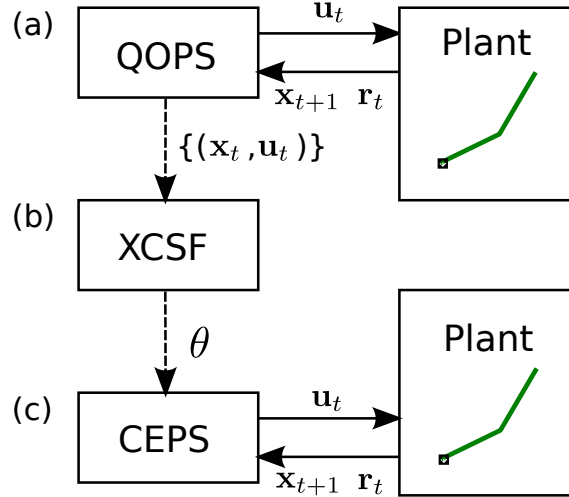


FIGURE 2 – Architecture globale pour les expériences.

La figure 2 décrit l'architecture globale utilisée pour les expériences. Un ensemble de trajectoires état-action quasi-optimales engendrées par le QOPS (figure 2 (a)) fournit des échantillons pour l'apprentissage supervisé avec XCSF, auquel on fournit l'état du système comme entrée pour l'espace des conditions et des prédictions, et l'action comme sortie sur laquelle on applique la régression.

En fournissant de tels échantillons à XCSF (figure 2 (b)), on engendre une action pour tout état qui se situe dans le domaine couvert par la population de classeurs. En utilisant une action par défaut $\mathbf{u}_{default}$ pour les états qui ne sont couverts par la population, donc pour lesquels XCSF ne prédit rien, on obtient une politique déterministe. Nous appelons « politique XCSF » cette politique. Elle est paramétrique puisque chaque classeur possède des paramètres dans ses parties *condition* et *prédiction*.

Cette politique XCSF est ensuite optimisée en utilisant CEPS (figure 2 (c)). On engendre un ensemble de politiques XCSF en faisant varier légèrement les paramètres de la politique initiale. Chaque élément de cet ensemble correspond à un point sur la figure 1. En pratique, on ne règle que les paramètres de la partie *prédiction* des classeurs : la forme des domaines associés à chaque modèle reste inchangé. En d'autres termes, les paramètres de la politique, donc les échantillons θ pour CEPS, sont des vecteurs composés des poids de chaque modèle local.

3 Protocole expérimental

Nous illustrons à présent l'utilisation d'XCSF pour apprendre à contrôler un bras de façon à ce qu'il atteigne une cible avec son effecteur terminal.

3.1 Modèle de bras

Le système est un bras à deux degrés de liberté contrôlé par 6 muscles, illustré 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 \mathbf{q}^* , la position articulaire courante du bras \mathbf{q} et sa vitesse courante $\dot{\mathbf{q}}$. L'état $(\mathbf{q}^*, \mathbf{q}, \dot{\mathbf{q}})$ est de dimension 6. L'état initial

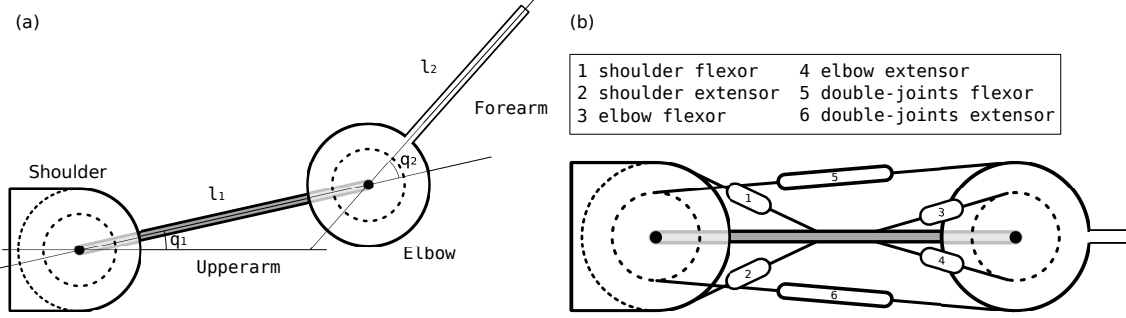


FIGURE 3 – Modèle de bras. (a) Vue schématique de la mécanique du bras. (b) Vue schématique de l’actionnement musculaire, où chaque nombre représente un muscle dont le nom est dans la boîte.

est défini par $\mathbf{q} = [q_1, q_2] = [0.5, 0.59]$, une vitesse nulle et une position de cible variable. 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 $\sigma_u^2 = .02$. Le simulateur utilise la méthode d’Euler avec un pas de temps $\Delta t = 2$ ms et il est stoppé après $H = 1000$ pas de temps.

La tâche est accomplie quand la distance euclidienne entre l’effecteur terminal et la cible est inférieure à $d = 0.01$ m. Il n’y a pas de condition explicite sur la vitesse, mais la méthode de Rigoux *et al.* (2010) fait en sorte que la vitesse soit très faible à proximité de la cible. La fonction de récompense, donnée par l’équation (1), est paramétrée par $\alpha = 200\Delta t$ et $\beta = 40\gamma$ avec un facteur d’actualisation γ fixé à $e^{-\Delta t}$ (ces paramètres ont été choisis pour correspondre à ceux du critère utilisé par QOPS).

3.2 Méthodes expérimentales

Le QOPS est codé en C++ avec la bibliothèque Eigen. Il écrit les trajectoires engendrées dans un fichier. Nous utilisons l’implémentation JavaXCSF (Stalph & Butz, 2009) d’XCSF, et le code principal ainsi que l’algorithme CEPS sont codés en Java. Pour afficher les résultats, nous utilisons Python avec la bibliothèque Matplotlib (Hunter, 2007). Les expériences tournent sur un processeur Intel Core 2 Duo E8400 @ 3 GHz avec 4 Go de RAM. Nous réalisons deux expériences.

3.2.1 Généralisation avec XCSF

Tout d’abord, nous créons deux ensembles de positions cibles, une pour l’apprentissage et l’autre pour tester la performance du contrôleur. Un ensemble de 100 cibles d’apprentissage sont tirées selon une loi normale centrée sur le centre de l’espace atteignable (moyenne $(x = -0.059, y = 0.44)$ et écart-type 0.1). Les cibles de test sont réparties sur une grille 11×11 dans l’espace atteignable. Toutes les trajectoires de toutes les expériences partent de la même position initiale (voir la figure 4).

Le QOPS est utilisé pour engendrer des trajectoires pour toutes les cibles. Ensuite, XCSF apprend à prédire les actions du QOPS pour les cibles d’apprentissage en utilisant ces trajectoires, puis sa performance est établie sur les cibles de test. Ces performances et les trajectoires sont comparées avec celles obtenues avec le QOPS, pour voir comment le contrôleur engendré par XCSF se comporte par rapport à une référence quasi-optimale.

3.2.2 Adaptation avec CEPS

Dans un deuxième temps, la politique XCSF obtenue à l’issue de l’expérience précédente 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 fait qu’un épisode pour évaluer la performance sur une cible donnée.

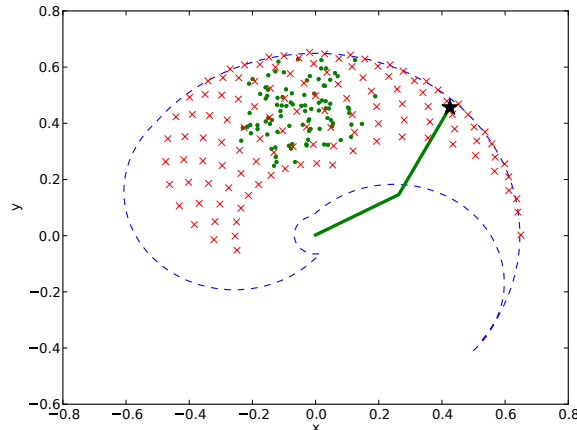


FIGURE 4 – L’espace atteignable du bras est délimité par une enveloppe en forme de spirale. Les deux segments du bras sont représentés par deux lignes en gras. La configuration initiale est celle qui apparaît sur la figure. Les cibles d’apprentissage sont représentées par des poids et les cibles de test par des croix.

3.3 Paramètres

Les paramètres d’XCSF sont les suivants. Le nombre d’itérations est fixé à 100000 et la taille maximale de la population à 6400. Les entrées sont normalisées : les positions courante et cible sont bornées par l’espace atteignable et la vitesse est bornée par $[-100, +100] \text{ rad.s}^{-1}$. L’action par défaut $\mathbf{u}_{default}$ est un vecteur nul, donc il n’y a pas d’activation musculaire. Après un réglage empirique des paramètres, le taux d’apprentissage α (appelé **beta** dans JavaXCSF) est réglé à 1.0, et la compaction est désactivée². On désactive le *multithreading* pour favoriser la reproductibilité des résultats.

Pour CEPS, le nombre d’itérations T est fixé à 40, chacune consistant en l’évaluation de $N = 100$ politiques. Chaque cible étant évaluée une fois seulement, le nombre d’essais M est égal au nombre de cibles, à savoir 121. La proportion d’épisodes retenus ρ est fixé 0.01, donc seule la meilleure des 100 politiques est retenue comme nouvelle moyenne de la distribution sur les paramètres des politiques. Un bruit supplémentaire $\bar{\sigma}^2 = 0.01$ est utilisé en tant que nouvelle variance σ^2 sur la distribution des paramètres de politiques.

4 Résultats

Dans cette section, nous présentons les résultats obtenus. Nous examinons tout d’abord si la politique apprise avec XCSF est similaire à celle obtenue avec le QOPS pour les mêmes cibles et nous mesurons la capacité de généralisation de XCSF sur des 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 et généralisation

Le temps d’exécution moyen pour réaliser une trajectoire avec le QOPS est d’environ 10 minutes. Avec la politique XCSF, cela prend environ 2 secondes. Optimiser la politique XCSF avec CEPS n’augmente pas le temps de réalisation d’une trajectoire.

La figure 5(a) montre l’ensemble des trajectoires utilisées pour apprendre la politique XCSF. La figure 5(b) montre l’ensemble des trajectoires obtenues avec le QOPS pour quelques cibles choisies dans l’ensemble de test. La figure 5(c) montre les trajectoires engendrées par la politique XCSF pour les mêmes cibles. La forme générale de ces trajectoires est similaire, même si le QOPS engendre des trajectoires un peu plus recourbées pour les cibles les plus lointaines.

2. `startCompaction=resetRLSPredictionsAfterSteps=2`

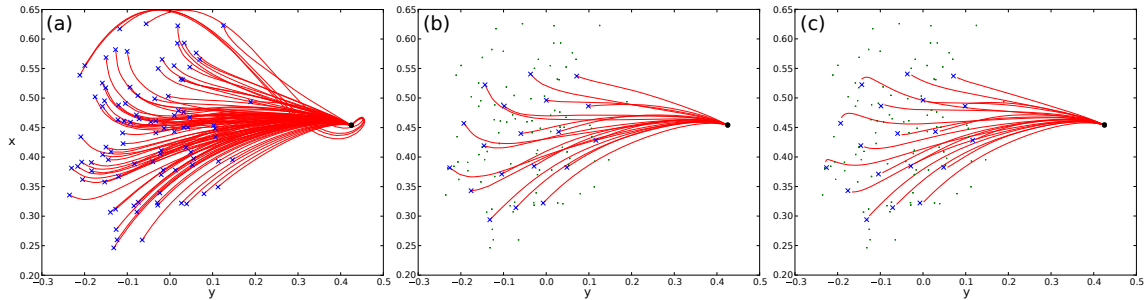


FIGURE 5 – Trajectoires obtenues avec le QOPS pour les cibles d’apprentissage (a), les cibles de test (b) et avec la politique XCSF pour les cibles de test (c). La position de départ est représentée par un point, les cibles sont représentées par des croix. Dans (b) et (c), les points représentent les cibles d’apprentissage.

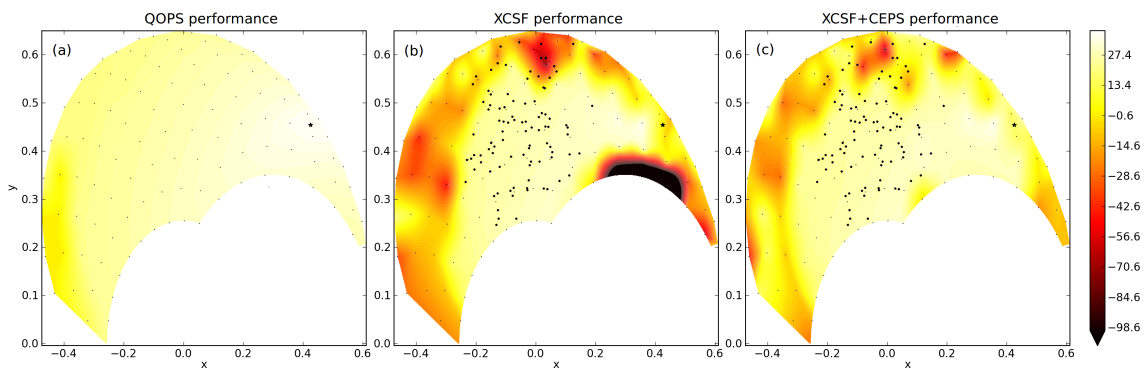


FIGURE 6 – Performance du QOPS (a), de la politique XCSF (b) et de la politique XCSF optimisée par CEPS (c), obtenue en interpolant les performances sur les cibles de test représentées par des petits points. Les positions des cibles d’apprentissage sont indiquées par des gros points et la position initiale par une étoile. La performance est calculée conformément à l’équation (1) et représentée par une couleur définie par l’échelle à droite.

La figure 6(a) montre la performance du QOPS en fonction de la position des cibles, obtenue en interpolant la performance des trajectoires du QOPS sur les cibles de test. La figure 6(b) montre la performance d’une politique XCSF représentative. Cette performance est très proche de celle du QOPS pour les cibles proches des cibles d’apprentissage. La performance pour des cibles plus éloignées est généralement plus faible, mais elle est équivalente à celle du QOPS pour certaines cibles. La discontinuité de performance dans la partie supérieure de la région des cibles d’apprentissage peut s’expliquer par la bifurcation des trajectoires (voir la figure 5(a)).

4.2 Performance après optimisation par CEPS

La figure 6(c) montre la performance de la politique XCSF 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 de $(x = 0.4, y = 0.35)$ où elle était très mauvaise.

Les résultats de la Table 1 confirment quantitativement cette impression visuelle. En comparant la colonne 2 et la colonne 4, on peut voir que l’écart important de performance sur la totalité de l’espace atteignable entre le QOPS et la politique XCSF est due principalement à de petites 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, (« région d’apprentissage » dans la Table 1), on peut voir que la performance est beaucoup plus proche de l’optimum. Mais, de façon intéressante, cette performance locale décroît après l’optimisation par CEPS. Ces points sont discutés dans la section 5.

TABLE 1 – Performance sur les cibles de test mesurée par l’équation (1)). Les deux premières colonnes montrent la performance et l’écart type ainsi que le pourcentage de performance par comparaison à QOPS pour la totalité de l’espace atteignable. Les deux dernières colonnes montrent la même information pour la région d’apprentissage seulement (régions des croix sur la figure 5 (b) et (c)).

	espace complet	%QOPS	région d’apprentissage	%QOPS
QOPS	25.85 ± 7.5	100%	27.27 ± 1.8	100%
XCSF	-3.59 ± 45.8	12%	26.95 ± 1.9	99%
+CEPS	9.32 ± 22.0	32%	24.06 ± 10.6	88%

5 Discussion

Le principal bénéfice de l’utilisation de la politique XCSF comme contrôleur est qu’il est environ 300 fois plus rapide que le QOPS. Cela permet de rendre compte de l’exécution en temps-réel d’un mouvement, ce qui est loin d’être le cas avec le QOPS. Par ailleurs, la capacité de généralisation d’XCSF joue aussi un rôle très important. En effet, apprendre la politique XCSF serait extrêmement coûteux s’il fallait engendrer un grand nombre de trajectoires avec le QOPS pour arriver à une performance acceptable. Comme le montre la figure 6(b), quelques trajectoires suffisent. 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.

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 QOPS. 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 pourrions utiliser CEPS à raison d’une cible à la fois et définir un critère local qui améliorerait la performance relativement à cette unique cible. De plus, il serait intéressant de mettre au point une méthode plus locale qui agirait préférentiellement sur les classeurs qui ont le plus besoin d’être corrigés, sans modifier les classeurs efficaces. L’approche la plus convaincante pour réaliser cette nouvelle conception consisterait à introduire le processus d’optimisation stochastique au sein des mécanismes d’XCSF.

6 Conclusion

Dans cet article, nous avons montré que XCSF pouvait être utilisé pour apprendre une politique continue en états et en actions 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. Le contrôleur qui en résulte est rapide et la capacité de généralisation d’XCSF simplifie le processus d’apprentissage. De plus, nous avons appliqué au contrôleur appris un processus d’optimisation stochastique qui améliore encore sa performance. Cependant, notre étude expérimentale a montré que la performance obtenue n’est pas encore assez proche de celle du QOPS utilisé pour apprendre. Pour améliorer encore cette performance, les options les plus évidentes sont les suivantes :

- On pourrait considérer d’autres modèles de prédiction (par exemple quadratique au lieu de linéaire) ou définitions des régions (par exemple des hyper-rectangles plutôt que des gaussiennes).

- On pourrait appliquer CEPS sur un plus grand nombre de paramètres de la population d’XCSF. Par exemple, la partie *condition* pourrait être exploitée en optimisant la forme du domaine des classeurs. De plus, la population elle-même pourrait être améliorée en ajoutant ou en supprimant des classeurs via des modifications du jeu de paramètres. La contrepartie de l’optimisation sur une plus grand nombre de paramètres est qu’elle prend généralement plus de temps.

- De même que les algorithmes génétiques, CEPS est un algorithme très générique qui n’utilise pas le gradient de la performance du système en fonction des paramètres pour améliorer la politique. Pourtant, dans la mesure où le modèle de XCSF est différentiable, nous pourrions lui appliquer des méthodes de gradient plus sophistiquées telle que (Kormushev *et al.*, 2010; Peters & Schaal, 2008). Ceci dit, les travaux de Thiéry (2010) nous conduisent à considérer que, bien qu’elles s’appuient sur davantage d’information pertinente, il ne va pas de soi que ces méthodes permettent d’obtenir de meilleures performances.

- Finalement, 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 complexe, on peut plutôt apprendre un modèle de ce système, tel que cela a été fait dans (Salaün *et al.*, 2009; Stalph *et al.*, 2010) par exemple, et utiliser ce modèle pour améliorer la politique.

Remerciements

Ce travail est financé par le « Ambient Assisted Living Joint Programme of the European Union and the National Innovation Office (DOME0-AAL-2008-1-159) », voir <http://www.aal-domeo.eu>.

A Dynamique du bras

La table 2 fournit la nomenclature des paramètres et variables du modèle. La dynamique du bras est calculée de la façon suivante, étant donné l’état courant $\mathbf{x}_t = (\mathbf{q}^*, \mathbf{q}, \dot{\mathbf{q}})$ et l’action \mathbf{u}_t :

$$m_1 = 1.4, m_2 = 1.1, l_1 = .30, l_2 = .35$$

$$s_1 = .11, s_2 = .16, d_1 = .025, d_2 = .045, \kappa = 25$$

$$k_1 = d_1 + d_2 + m_2 l_1^2, k_2 = m_2 l_1 s_2, k_3 = d_2$$

$$\mathbf{A} = \begin{bmatrix} .04 & -.04 & 0 & 0 & .028 & -.035 \\ 0 & 0 & .025 & -.025 & .028 & -0.35 \end{bmatrix}^\top$$

$$\mathbf{T} = [700 \quad 382 \quad 572 \quad 445 \quad 159 \quad 318]$$

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

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

$$\mathbf{C} = [-\dot{q}_2(2\dot{q}_1 + \dot{q}_2)k_2 \sin(q_2) \quad \dot{q}_1^2 k_2 \sin(q_2)]$$

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

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

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

$$\partial \mathbf{q}^* / \partial t = 0$$

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

$$\partial \mathbf{q} / \partial t = \dot{\mathbf{q}}$$

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

TABLE 2 – Paramètres du modèle du 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
\mathbf{A}	matrice des moments du bras
\mathbf{T}	tension musculaire
\mathbf{M}	matrice d’inertie
\mathbf{J}	matrice Jacobienne
\mathbf{C}	force de Coriolis
$\boldsymbol{\tau}$	couples aux segments ($N.m$)
\mathbf{B}	amortissement
\mathbf{u}	activation musculaire (action)
σ_u^2	bruit musculaire multiplicatif
$\tilde{\mathbf{u}}$	activation musculaire filtrée
\mathbf{q}^*	position articulaire visée (rad)
\mathbf{q}	position articulaire courante (rad)
$\dot{\mathbf{q}}$	vitesse articulaire courante ($rad.s^{-1}$)

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

Références

- ARGALL B. D., CHERNOVA S., VELOSO M. & BROWNING B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, **57**(5), 469–483.
- BERNADÓ-MANSILLA E., LLORÀ X. & GARREL J. M. (2001). XCS and GALE : a comparative study of two Learning Classifier Systems with six other learning algorithms on classification tasks. In P.-L. LANZI, W. STOLZMANN & S. W. WILSON, Eds., *Proceedings of the fourth international workshop on Learning Classifier Systems*.
- BERTSEKAS D. P. (1995). *Dynamic Programming and Optimal Control*. Belmont, MA : Athena Scientific.
- BUSONI 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*, **41**(1), 196–209.
- BUTZ M. V., GOLDBERG D. & LANZI P. (2005). Computational Complexity of the XCS Classifier System. *Foundations of Learning Classifier Systems*, **51**, 91–125.
- BUTZ M. V. & HERBERT 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*, **8**(1), 28–46.
- DE BOER P.-T., KROESE D. P., MANNOR S. & RUBINSTEIN R. Y. (2005). A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, **134**(1), 19–67.
- GUIGON E., BARADUC P. & DESMURGET M. (2008). Optimality, stochasticity and variability in motor behavior. *Journal of Computational Neuroscience*, **24**(1), 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.
- HUNTER J. (2007). Matplotlib : A 2D graphics environment. *Computing in Science & Engineering*, p. 90–95.
- KOBER J. & PETERS J. (2008). Policy search for motor primitives in robotics. *Advances in Neural Information Processing Systems (NIPS)*, p. 1–8.
- 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)*.
- LANZI P.-L. (2002). Learning Classifier Systems from a Reinforcement Learning Perspective. *Journal of Soft Computing*, **6**(3-4), 162–170.
- LI W. (2006). *Optimal control for biological movement systems*. PhD thesis, University of California, San Diego.
- PETERS J. & SCHAAL S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks : the official journal of the International Neural Network Society*, **21**(4), 682–97.
- 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*, **99**(1), 89–112.
- RUBINSTEIN R. Y. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, **1**(2), 127–190.
- SALAÜN C., PADOIS V. & SIGAUD O. (2009). Control of redundant robots using learned models : an operational space control approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, p. 878–885.
- 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 interne, COBOSLAB.
- SZITA I. & LÖRINCZ A. (2006). Learning Tetris using the noisy cross-entropy method. *Neural computation*, **18**(12), 2936–41.
- THIÉRY C. (2010). *Itération sur les Politiques Optimiste et Apprentissage du Jeu de Tetris*. PhD thesis, Université Henri Poincaré - Nancy 1.
- TODOROV E. & JORDAN M. I. (2002). Optimal feedback control as a theory of motor coordination. *Nature Neurosciences*, **5**(11), 1226–1235.
- TRAN T. H., SANZA C., DUTHEN Y. & NGUYEN D. T. (2007). XCSF with computed continuous action. In *Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO'07)*, p. 1861–1869 : ACM New York, NY, USA.
- WILLIAMS R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, **8**(3-4), 229–256.
- WILSON S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, **3**(2), 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*, **1**(2-3), 211–234.
- WILSON S. W. (2006). *Three Architectures for Continuous Action*. Rapport interne, No. 2006019, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.