ABC : Animation Basée sur le Comportement

Ph. Preux*

F. Barbier† J-Cl. Darcheville ‡ S. Delepoulle§ F. Montagne§ Ph. Pudlo†

Ch. Renaud§ F. Rousselle§

15 janvier 2004

Résumé

Le projet ABC concerne la génération de comportements dans le cadre de l'animation d'entités virtuelles. Nous proposons d'aborder ce problème en combinant les approches basées sur la capture de mouvements et des algorithmes adaptatifs.

1 Introduction

Nous nous intéressons à la synthèse de mouvements effectués par des agents artificiels logiciels. Nous ne parlons pas ici des activités cognitives éventuelles entraînant un certain mouvement, mais du mouvement lui-même. Par exemple, dans ce contexte, nous voulons qu'un agent possédant une morphologie d'humanoïde, lorsqu'il effectue un mouvement du bras, ce mouvement soit le plus « réaliste » possible. Nous mettons ce terme entre « », sa définition précise risquant de nous entraîner dans des développements que nous ne voulons pas aborder ici. Disons simplement que l'on veut que l'agent effectue un mouvement « qui ne choque pas » notre perception : fluide, physiquement et biomécaniquement réaliste, ne traversant pas les objets, ... Ce problème n'est pas résolu à l'heure actuelle de manière totalement automatisée.

Pour synthétiser une séquence comportementale automatiquement, la technique standard est celle de la cinématique inverse. Étant donné un agent articulé, une position initiale et une position finale, on cherche la trajectoire reliant la première à la dernière. Si cette technique est adéquate pour contrôler le mouvement de robots industriels, elle ne permet pas de produire des mouvements réalistes. D'autres techniques sont également disponibles reposant sur la simulation de la dynamique d'un agent. Dans ce cas, on recherche les lois fondamentales du comportement de l'agent que l'on simule ensuite. On attend de la simulation de ces lois qu'elle produise un comportement intéressant. C'est ainsi que l'on obtient des dynamiques telles que celle de vols d'oiseaux (*boids*) [Rey87], de colonie d'insectes, du développement de la marche hexapode [Kod98], de nage par « nageur » virtuel [Cou02], d'un mouvement d'atteinte d'une cible par un bras [RB02, PDD04], ... Dans les travaux sur les *boids*, l'agent a un comportement purement réactif : en fonction de son environnement, il effectue une certaine action. Dans les trois derniers travaux mentionnés, l'agent n'est plus purement réactif dans le sens où son comportement évolue au cours du temps : dans un même état de son environnement, son comportement change au cours de son activité. Ces agents adaptent constamment leur comportement à leur environnement. Ils sont basés sur des algorithmes d'apprentissage par renforcement. S'appuyant sur les lois fondamentales du comportement animal, ces agents pourraient être utilisés pour simuler des mouvements réalistes si l'on disposait de modèles précis de la physique de leur environnement, ce dont nous ne disposons pas d'une manière générale. En outre, force est de constater que l'apprentissage par renforcement est lent, en tout cas initialement, lorsque l'on ne fournit aucune aide particulière à l'agent.

Aussi, à l'heure actuelle, la synthèse des mouvements des artefacts repose principalement sur des techniques de capture de mouvements [Don00]. Ces techniques sont lourdes, onéreuses et restreintes. Ainsi, toutes les séquences de mouvements doivent

*coordinateur du projet ; GRAPPA, EA 3588, Université de Lille 3, philippe.preux@univ-lille3.fr, http://www.grappa.univ-lille3.fr/~ppreux/abc

†LAMIH, UMR CNRS 8530, Université de Valenciennes

‡URECA, EA 1059, Université de Lille 3

§LIL, JE 2335, Univeristé du Littoral Côte d'Opale, Calais

être prévues à l'avance et les scénarii décrits exhaustivement et de manière précise. Un scénario est ensuite joué en mettant bout à bout des séquences comportementales des différents agents composant la scène ; les agents ont un ensemble de comportements fixé à l'avance ; l'adaptation de leur comportement face à des changements de l'environnement est une tâche difficile. Par exemple, voulant saisir un objet, il serait bon que l'agent puisse le saisir quelle que soit sa position, pas forcément envisagée lors de la capture ; se déplaçant d'un point à un autre, il serait bon que l'agent adapte sa démarche à des sols de différentes natures, pas forcément envisagés lors de la capture ; ou encore, adapte sa trajectoire si des obstacles surgissent ou d'autres agents la rencontre ; ...

Face à ces différents points, nous voulons combiner ces deux approches : l'apprentissage par renforcement parce qu'il fournit un modèle intéressant du comportement animal et capture de mouvements pour aider l'agent lors de son apprentissage et lui fournir des exemples de ce qu'il doit faire, ce qui accélérera son apprentissage. On peut ainsi espérer réduire la quantité de mouvements capturés nécessaires pour obtenir les mouvements de l'agent (donc, réduction de la charge de travail par rapport aux captures de mouvements et réduction du temps d'apprentissage par l'agent) et utiliser les propriétés naturelles de l'apprentissage par renforcement pour obtenir des mouvements qui sont beaucoup moins figés et même, qui seront capables d'évoluer au cours du temps si l'environnement de l'agent change. Cela constitue l'esprit du projet ABC.

Dans la suite de cette présentation, nous décrivons d'abord l'apprentissage par renforcement puis l'architecture logicielle des agents utilisés ici. Nous présentons ensuite le type de mouvements sur lequel nous travaillons puis notre méthodologie et quelques résultats préliminaires avant de discuter des perspectives et de conclure.

2 L'apprentissage par renforcement

Les algorithmes basés sur la notion d'apprentissage de la différence temporelle résolvent le problème de l'apprentissage par renforcement qui est un problème de décision : dans un état donné, quelle action doit effectuer l'agent. Définissons ce problème plus précisèment. Considérons un agent évoluant dans un temps discret $t \in \mathbb{N}$. Soient :

- un ensemble d'états $\mathcal{S}$ décrivant les états possibles de l'agent. Parmi les états, certains peuvent être terminaux ;
- un ensemble d'actions possibles dans l'état s : $\mathcal{A}(s), \forall s \in \mathcal{S}$ et $\cup_{s \in \mathcal{S}} \mathcal{A}_s = \mathcal{A}$;
- une fonction de transition : $\mathcal{P}^a_{ss'}$ qui donne la probabilité de passer de l'état s dans l'état s' suite à l'émission de l'action a ;
- une fonction de retour immédiat : $\mathcal{R}^a_{ss'}$ qui donne le retour immédiat moyen perçu suite au passage de l'état s dans l'état s' résultant de l'émission de l'action a.

Dans la suite, on restreindra notre discussion au cas des problèmes de décision markoviens finis (PDMF), fini car l'ensemble des états et des actions est fini, markovien car $\mathcal{P}^a_{ss'}$ et $\mathcal{R}^a_{ss'}$ ne dépendent que de l'état courant et de l'action courante. Notons que l'apprentissage par renforcement s'applique également dans des situations de problèmes de décision non finis, non markoviens et non stationnaires.

Schématiquement, l'agent fonctionne comme suit. À l'instant t, il se trouve dans un état s_t qui synthétise sa perception de l'environnement courant ainsi que son état interne ; il choisit alors une action a_t à effectuer parmi $\mathcal{A}(s)$; l'émission de cette action provoque un changement d'état conformément à $\mathcal{P}$ et un retour immédiat (une conséquence) r_t conformément à $\mathcal{R}$. On passe à l'instant $t+1$.

On définit le retour : $R_t = \sum_{k=0}^{k=\infty} \gamma^t + k r_{t+k+1}$ où r_{t+k} est le retour immédiat obtenu à l'instant $t+k$. Ce retour est donc la somme des retours immédiats obtenus au cours de l'activité de l'agent ; le coefficient $\gamma \in [0,1]$ (taux de dépréciation) donne une plus ou moins grande importance aux retours à court terme par rapport aux retours à long terme.

On définit une politique (ou stratégie) $\pi(s,a)$ qui spécifie pour chaque état s la probabilité d'émettre l'action a. Le problème d'apprentissage par renforcement consiste à trouver une politique $\pi^*(s,a)$ qui maximise R. On sait que la stratégie optimale d'un PDMF existe et qu'elle est déterministe.

On définit :

- la valeur d'un état s (notée $V^\pi(s)$) est l'espérance de gain après avoir visité l'état s si l'agent suit la stratégie π :
  $V^\pi(s) = E_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, s_t = s)$;
- la qualité d'un couple état s, action a (notée $Q^\pi(s,a)$) est l'espérance de gain suite à l'émission du comportement a dans l'état s si l'agent suit la stratégie π : $Q^\pi(s,a) = E_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, s_t = s, a_t = a)$;

On note V^* et Q^* les valeurs optimales de ces quantités qui correspondent à une stratégie optimale π^*. V^* et Q^* peuvent être calculées par divers algorithmes itératifs. Une fois Q^* obtenu, π^* peut en être aisément déduite en adoptant un algorithme glouton : pour chaque état s, la stratégie optimale consiste à effectuer l'action (ou l'une des actions s'il y en a plusieurs) dont la qualité est maximale dans cet état.

Ainsi, dans le cas où ces 4 données $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ sont connues *a priori*, il existe un algorithme qui calcule π^* avec une complexité temporelle proportionnelle au nombre d'états au carré : c'est la méthode de programmation dynamique. Nous ne décrirons par cette méthode ici (voir par exemple [BT96]).

D'autres algorithmes ont été proposés, les méthodes à base de différence temporelle qui ont une certaine pertinence par rapport à la dynamique comportementale du vivant. Nécessitant moins d'hypothèses que la programmation dynamique pour leur mise en œuvre (il suffit de connaître $\mathcal{S}$ et $\mathcal{A}$), ces méthodes fournissent asymptotiquement Q^*, donc π^*, pour des environnements markoviens ; expérimentalement, on sait que l'on obtient des résultats intéressants au bout d'un nombre fini d'itérations et que ces méthodes fonctionnent aussi dans des environnements non markoviens et infinis. Dans ce cas, on ne dispose d'aucun résultat formel, seulement de preuves expérimentales ; un autre de leurs intérêts est de permettre un apprentissage en continu (en-ligne) : ceci est particulièrement intéressant pour les systèmes en constante interaction avec leur environnement : l'algorithme est en constante adaptation par rapport à son milieu, qui peut donc ne pas être stationnaire.

Les méthodes à base de différence temporelle (désormais abrégé en méthode ou algorithme TD) sont des algorithmes itératifs qui, à chaque itération, décident de l'action à effectuer en fonction de leur expérience passée, et corrigent leur estimation de l'intérêt d'effectuer cette action dans cet état (la qualité), en fonction du retour immédiat obtenu et de leur expérience passée. Contrairement à un algorithme d'apprentissage supervisé, la meilleure action à effectuer dans un état donné n'est jamais indiquée à un algorithme TD ; de même, l'algorithme ne reçoit aucune information lui indiquant s'il aurait pu effectuer une meilleure action ; il reçoit simplement un retour immédiat, ce retour immédiat étant la conséquence des actions précédentes de l'agent et étant non déterministe.

Un nombre croissant d'algorithmes TD ont été proposés : acteur-critique, Q-Learning, Q(λ), SARSA, SARSA(λ), TD (λ) pour n'en citer que les plus connus (voir par exemple [BT96, KLM96, SB98]). En réalité, il s'agit plutôt de schémas d'algorithmes : chacun peut être implanté de différentes manières ce qui peut faire grandement varier les performances pour un même algorithme.

Dans tous les cas cependant, l'idée de l'algorithme est à peu près la même : estimer V ou Q ; itérativement, en interagissant avec l'environnement, améliorer cette estimation ; utiliser l'estimation courante pour décider de l'action à effectuer. Un algorithme TD est principalement constitué de deux compsoants : l'un stocke l'estimation de la valeur des états (une structure de données : un tableau, un réseau de neurones, ...) ; l'autre sélectionne l'action à affectuer ; ces deux composants sont plus ou moins emmélés.

3 L'architecture logicielle de l'agent apprenant

Notre objectif est d'utiliser un algorithme TD en lui fournissant des exemples de trajectoires comportementales (suite d'actions). Ces exemples de trajectoire ne sont pas parfaits : ils permettent de dégrossir la recherche d'une stratégie optimale ; de plus,ce sont des « morceaux » de trajectoires qui ne relient pas un état initial à un état final mais deux états quelconques.

Après de premières tentatives d'utilisation des algorithmes d'apprentissage par renforcement classiques Q(λ), nous avons décidé de concevoir un nouvel algorithme, mieux adapté à la combinaison avec l'apprentissage à l'aide d'exemples et à des problèmes ayant de très grands espaces d'états [MDP03]. C'est un algorithme général dont le domaine d'applications n'est pas restreint au problème traité dans ABC et dont nous explorons les possibilités d'application à d'autres problèmes de contrôle.

Plus précisèment, l'algorithme de contrôle est de type TD(λ) [SB98]. Il est itératif : en entrée, il prend l'état courant et fournit en sortie l'action à effectuer pour se rapprocher de l'objectif. Ceci place l'agent dans un nouvel état qui provoque une nouvelle action de l'agent à l'itération suivante. Il s'agît donc d'apprendre des couples (état, action) permettant de déplacer l'agent vers la cible de manière naturelle.

Nous souhaitons que l'agent apprenne des trajec-

toires exemples et qu'ensuite, il puisse interagir librement avec son environnement. Les trajectoires exemples ne sont pas forcément des trajectoires optimales, loin s'en faut. Ce sont juste des aides pour que l'algorithme puisse réaliser son apprentissage plus rapidement à partir de ces ébauches de solution.

Un réseau de neurones est entraîné sur les trajectoires exemples en utilisant l'algorithme de rétro-propagation de l'erreur. Un problème se pose cependant concernant la définition de la notion d'état : quelles sont les données réellement pertinentes pour déterminer l'action à réaliser parmi toutes les données que nous fournissent les mouvements capturés ? Non seulement un mauvais choix alourdirait considérablement les temps d'apprentissage mais en plus, le fait d'utiliser trop d'informations pour spécifier un état dégraderait, éventuellement considérablement, les performances de l'algorithme (sur-apprentissage). Nous avons donc proposé d'utiliser ici une définition de l'état qui soit simplifiée pour éviter ce problème. Cet état est une projection dans un sous-espace de l'état original. Pour être tout à fait précis, l'algorithme apprend la valeur des états, c'est-à-dire, l'intérêt pour l'algorithme de se trouver dans cet état, vis-à-vis de l'objectif qu'il doit atteindre. Plus cette valeur est grande, meilleur est cet état. Disposant d'une notion de voisinage entre les états, pour déterminer l'action à réaliser, il suffit de considérer les états voisins de l'état courant et de choisir celui dont la valeur est la plus grande et d'émettre l'action qui va l'y faire passer, ceci jusqu'à atteindre un état terminal : la cible a été atteinte. On peut voir cela également comme l'apprentissage d'un gradient en direction de la cible.

Cet apprentissage à partir des exemples réalisé, l'agent doit pouvoir atteindre des cibles localisées à d'autres positions que celles qui ont été apprises. Nous voulons également que l'agent puisse adapter son comportement si son environnement change : la cible est mouvante, un obstacle est interposé entre la cible et lui, ... Pour cela, un apprentissage par renforcement est utilisé qui tend à apprendre, pour un état donné, l'action qui doit être réalisée. Le problème est que si l'on utilise le même réseau de neurones pour y stocker ces nouvelles informations, les données de l'apprentissage initial vont être perdues. Nous proposons donc une architecture constituée de deux composants (qui sont chacun un réseau de neurones) : le premier mémorise les trajectoires exemples dans un espace d'états simplifié ; dès lors, le second va apprendre pour chacun des véritables états si la valeur apprise par le premier réseau est digne de confiance ou non. Dans le premier cas, on choisira l'action principalement en fonction du premier réseau ; dans le second cas, on choisira une autre action que celle qui est indiquée par le premier réseau. Ainsi, le comportement peut s'adapter si l'environnement est tel que l'apprentissage initial n'est plus satisfaisant ; en même temps, cet apprentissage initial est conservé et donne une approximation des mouvements à réaliser.

Dans une version plus élaborée de cet algorithme, les deux phases d'apprentissage (avec des exemples et en-ligne) seront intimement mélées. Ainsi, l'apprentissage initial perdra cet aspect figé : si l'environnement se modifie vraiment de manière non temporaire, il n'est pas besoin de conserver les trajectoires initiales ; donc, le premier réseau doit lui-aussi s'adapter à ces changements lents de l'environnement. On obtient donc un algorithme qui intègre deux niveaux d'adaptation : l'un pour les variations fréquentes et « superficielles », l'autre pour les variations plus structurelles et plus rares. Par ailleurs, l'architecture à base de ddeux réseaux de neurones n'est pas une obligation. D'autres modes de représentations sont envisageables et sont actuellement à l'étude.

Nous dénommons cet algorithme « architecture critique-critique » par référence à l'architecture « acteur-critique » [Wit77, BSA83]. Chacun des deux réseaux de neurones est un critique qui, en fonction de l'état courant, produit un avis sur l'intérêt d'être dans cet état. La dimension « acteur » de l'algorithme est bien entendu présente (sinon, il n'émettrait aucune action) mais laissée dans l'ombre puisque nous ne proposons rien là d'original. Cette architecture met l'accent sur la phase de sélection de l'action et plus généralement, l'équilibrage entre l'exploration et l'exploitation des connaissances déjà acquises. À l'heure actuelle, l'exploration est essentiellement aléatoire ; notre algorithme essaie d'avoir une stratégie d'exploration qui soit meilleure que l'aléatoire. Notons aussi que l'utilisation de deux espaces d'états, l'espace « complet » avec des informations éventuellement inutiles, l'autre élagué, est une originalité de notre approche. L'objectif est de travailler sur l'espace d'états élagué pour réduire au maximum la charge en temps de calcul et de n'utiliser l'espace complet que pour des corrections. Un autre avantage d'utiliser un espace élagué est d'augmenter la généralisation de l'apprentissage réalisé. Il n'est pas toujours évident, loin de là, de déterminer les attributs réellement utiles de l'état.

4 Méthodologie, application et résultats

Pour l'heure, nous nous intéressons au mouvement d'atteinte d'une cible par le bras d'un humanoïde. Pour cela, nous utilisons des trajectoires d'un bras capturées lors de la réalisation de ce même mouvement. Ces trajectoires sont injectées dans l'architecture critique-critique. Ci-dessous, nous détaillons successivement les aspects capture de mouvements et apprentissage des exemples.

4.1 Capture de mouvements

Nous capturons le mouvement d'atteinte d'un bras d'un sujet dont le reste du corps est immobile (aux petits mouvements involontaires inévitables près). La cible est placée devant le sujet à une distance telle que le sujet l'atteint le bras légérement flêchi et de telle manière que la cible soit située au-dessus du tronc du sujet. 13 positions différentes de la cible sont utilisées, ainsi qu'une position de départ.

Plus précisèment, la position d'un certain nombre de points de référence du sujet humain est mesurée ; ces points sont indiqués à la figure 1. Il y a 39 marqueurs disposés sur le corps du sujet : 2×8 pour les deux bras, 3 pour la tête, 5 pour le torse + 2×6 pour les deux jambes et 3 pour les hanches. Les mesures sont effectuées avec une fréquence de 120 Hz. Chaque mouvement consiste à atteindre un point cible à partir de la position de repos. Les 13 points cibles sont situés à la hauteur du tiers supérieur du sujet, face à lui (voir figure 1). Trois protocoles ont été utilisés :

- les points cibles étant numérotés de 1 à 13, atteinte de chacune des cibles, à partir du point de repos, dans l'ordre ;
- atteinte des 13 positions dans un ordre aléatoire, chacune étant atteinte depuis la position de repos ;
- atteinte des 13 positions dans un second ordre aléatoire, chacune étant atteinte depuis la position de repos.

En moyenne, chaque trajectoire du point de repos à la cible et retour consiste en 220 points de mesure et la mesure est donc fournie pour chacun des points de référence du sujet humain. La position des points est donnée dans un système de coordonnées cartésiennes centré sur la salle (et non sur le sujet). Le mouvement de retour de la cible vers la position de repos est également capturé. De ce fait, pour chaque position de la cible, on dispose de 3 trajectoires dans chaque

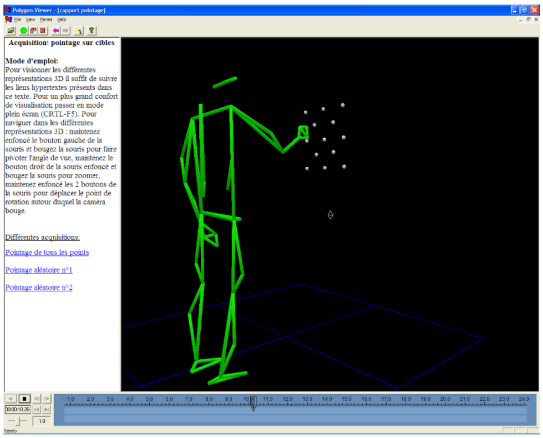

FIG. 1 – Une image de synthèse représentant un instant d'un mouvement capturé. Le sujet est stylisé. Les 13 points blancs indiquent les 13 positions de la cible : à partir d'une position de repos du bras verticale, le long du corps, le sujet humain pointe chacune de ces 13 positions. On voit le bras en cours de mouvement d'atteinte de l'une des cibles.

sens (vers et depuis la cible). Chaque mesure consiste en un triplet (x, y, z). On n'utilise que la position de l'épaule, du coude et du poignet, soit 9 nombres réels. On considère que les autres marqueurs sont fixes. On note que la position de l'épaule varie très peu au cours des différents mouvements.

4.2 Apprentissage des exemples

Les exemples sont constitués des positions (x, y, z) de l'épaule, du coude et du poignet, ainsi que de la cible du mouvement, soit 12 réels. Un état est donc un 12-uplet auquel on associe sa valeur. Celle-ci est assignée de manière à créer un gradient le long de la trajectoire afin que l'agent ait naturellement tendance à amener sa main vers la cible. Cette valeur doit donc être maximale aux cibles, minimale à la position de repos du bras, et croissante de cette position initiale à la cible. Parmi toutes les possibilités, nous avons choisi pour l'instant une fonction linéaire le long de la trajectoire, dont la valeur $\in [0, 1]$. Notons que le choix de cette fonction est assez arbitraire. Le but est d'obtenir le meilleur apprentissage possible ; aussi, d'autres fonctions non linéaires sont à l'étude.

On utilise uniquement les trajectoires de la position initiale à la cible et les 13 trajectoires capturées dans l'ordre. L'utilisation des trajectoires dans un ordre aléatoire n'apporte rien à l'apprentissage. Le nombre

d'exemples est de 13 trajectoires constituée chacune d'une centaine de points, soit 1554 trajectoires au total.

Cex exemples sont appris par un réseau de neurones possédant 12 entrées, 2 couches cachées et un neurone en sortie. La fonction d'activation des neurones est la fonction sigmoïde. Naturellement, les données sont bruitées.

4.3 Évaluation des performances de l'apprentissage des exemples

Nous souhaitons obtenir l'apprentissage des exemples le plus précis possible tout en évitant le sur-apprentissage et en minimisant les temps de calcul. Pour cela, nous avons du chercher l'architecture d'un réseau de neurones qui fournissent les meilleures performances. C'est également pour cette raison que nous poursuivons nos études concernant des solutions alternatives à la représentation de ces exemples : l'utilisation d'un réseau de neurones n'est pas forcément la meilleure solution.

Pour l'évaluation de l'apprentissage des exemples, nous avons décomposé l'ensemble des points des trajectoires en un jeu d'apprentissage (2/3 des exemples pris au hasard parmi les 1554) et un jeu de test (le 1/3 restant). Ceci nous a permis de choisir l'architecture du réseau de neurones.

Une fois l'apprentissage effectué, nous avons voulu mesurer la généralisation de l'apprentissage obtenu. Pour cela, on fait atteindre une cible dans une position qui n'a pas été apprise par le réseau et on meusre la distance entre l'extrémité du bras et la position de la cible à la fin de la trajectoire. La figure 2 du haut présente ces résultats. L'abcisse représente la position de la cible ; la barre horizontale en bas de la figure indique la zone contenant les positions apprises dans les exemples. L'ordonnée indique la distance de l'extrémité du bras avec la cible. Deux courbes sont représentées pour deux architectures de réseau de neurones différentes. On constate plusieurs choses :

- pour les positions de cible correspondant aux exemples mais aussi aux positions intermédiaires, l'agent approche son bras très près de la cible. Il a donc bien généralisé son apprentissage aux positions intermédiaires à celles apprises ;
- pour des positions de la cible extérieures aux exemples, les performances sont encore tout à fait intéressantes : le rséeau de neurones généralise donc bien son apprentissage en dehors de cette zone (dans une certaine limite bien entendu) ;
- les performances les meilleures sont obtenues par le réseau le plus petit parmi les 2 représentés ici (courbe en rouge, la plus proche de l'ordonnée 1). Ce réseau plus petit évite le sur-apprentissage et fournit donc une meilleure généralisation.

Ces résultats étant tout à fait satisfaisants, nous avons voulu essayer de diminuer le coût de l'apprentissage en terme de temps de calcul. Pour cela, on veut utiliser moins de trajectoires exemples tout en conservant les mêmes performances. La figure 2 du bas représente les résultats obtenus en n'utilisant que les 4 trajectoires extrêmes (les 4 coins). On voit à nouveau que les performances sont très bonnes et on observe bien à nouveau la généralisation de l'apprentissage à des positions de cibles non apprises. Les gains en temps de calcul sont appréciables : d'une quinzaine de minutes avec 13 trajectoires, on passe à 7 minutes avec 4 trajectoires.

5 Discussion et conclusion

À ce stade d'avancement du projet, l'architecture critique-critique est en cours de mise au point. Si le composant d'apprentissage des exemples est fonctionnel, le composant d'adaptation de l'apprentissage ne l'est pas encore tout à fait. La mise au point en est assez délicate. Nous pensons que ces problèmes seront rapidement résolus. Après une phase de test et de validation, nous envisageons d'appliquer la même méthodologie à la simulation de la marche bipède. Dans ce cas, nous voulons obtenir une simulation de différents régimes : marche lente, rapide, course, en ligne droite, en courbe, sur le plat, en plan incliné, sur des sols plus ou moins mous, ...

Parmi le travail à venir lorsque nous aurons une simulation du comportement de pointage, nous effectuerons un certain de tests de validité psychologique et bio-mécanique des mouvements obtenus. Sur un autre plan, nous « habillerons » le bras pour obtenir des images.

L'utilisation d'un algorithme généralisant son apprentissage à partir d'exemples ouvre un certain nombre de possibilités de manière à réduire les temps d'apprentissage d'une part, et surtout, le nombre d'exemples nécessaires pour obtenir un apprentissage de bonne qualité. Ce dernier point est important puisque le temps (et le coût) de réalisations des exemples (captures de mouvements) est crucial. Nous sommes actuellement en train de mener une

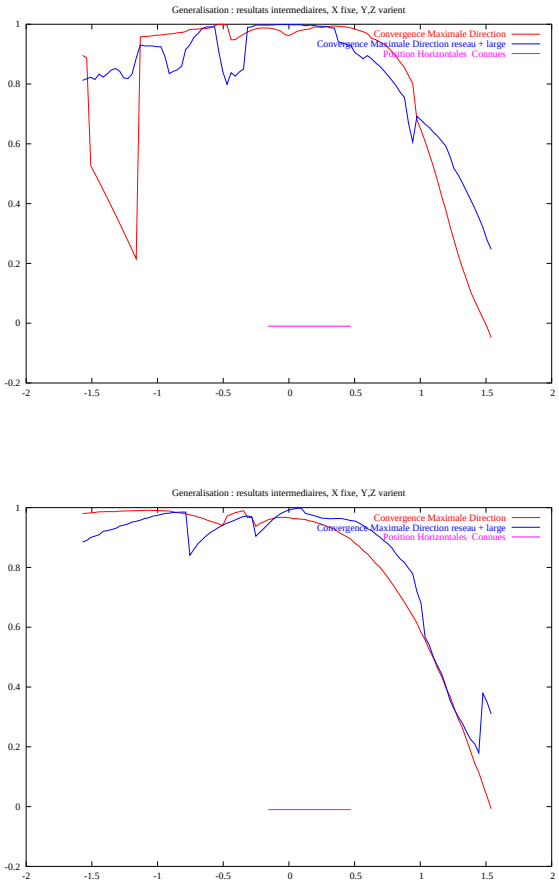

FIG. 2 – Courbes de généralisation de l'apprentissage obtenues avec deux architecture sde réseaux de neurones différentes, en utilisant tous les exemples disponibles (en haut) et en utilisant que 4 trajectoires (en bas). Voir le texte pour plus de précisions.

étude dans cette direction visant à déterminer le meilleur rapport qualité d'apprentissage par rapport à son coût. De manière connexe à ce point, d'autres méthodes de représentation des exemples sont envisageables à la place, ou en complément, à un réseau de neurones. Nous menons également une étude sur cette question. Par ailleurs, l'utilisation d'algorithme adaptatif permet de fournir des exemples approchés du comportement à réaliser et non de devoir fournir des exemples parfaits. Bien au contraire, à partir d'ébauches, l'algorithme adaptatif optimise petit à petit son comportement.

Il peut sembler curieux que nous n'abordions pas la notion de planification de l'action dans notre travail. En fait, notre travail n'est pas incompatible avec cette notion, mais il se place simplement à une autre échelle. Nous voulu simuler des comportements de « bas niveau » ; une fois ces comportements simulés, nous pouvons parfaitement les animer et les organiser à l'aide d'un planificateur d'actions. Ce planificateur pourrait tout à faut être lui-même adaptatif.

Enfin, pour conclure, nous aimerions ajouter que, pour nous, dans le comportement (même humain), tout n'est pas cognitif. Aussi, une saine analyse du comportement, sans *a priori* cognitiviste, sous un angle comportementale et mécanique, peut permettre d'avancer dans la simulation de comportements « de base », comportements omni-présents dans nos activités, bien plus fréquents que ceux qualifiés de cognitifs. Pour l'instant, l'absence de réalisme se situe beaucoup au niveau de ces comportements.

Références

[BSA83] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuron-like elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13 :835–846, 1983.

[BT96] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-dynamic programming.* Athena Scientific, 1996.

[Cou02] Rémi Coulom. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control.* PhD thesis, Institut National Polytechnique de Grenoble, 2002.

[Don00] S. Donikian. L'animation comportementale. *Bulletin de l'Association Française d'Informatique Graphique*,

December 2000. disponible à l'url `http://iic0e.univ-littoral.fr/~afig/`.

[KLM96] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning : a survey. *Journal of Artificial Intelligence Research*, 4 :237–285, 1996.

[Kod98] J. Kodjabachian. *Développement et évolution de réseaux de neurones artificiels*. PhD thesis, Paris 6, 1998.

[MDP03] F. Montagne, S. Delepoulle, and Ph. Preux. A critic-critic architecture to combine reinforcement and supervised learnings. In *Proc. European Workshop on Reinforcement Learning*, Nancy, 2003.

[PDD04] Ph. Preux, S. Delepoulle, and J-Cl. Darcheville. A generic architecture for adaptive agents based on reinforcement learning. *Information Sciences Journal*, 2004. (to appear).

[RB02] M.T. Rosenstein and A.G. Barto. Supervised learning combined with an actor-critic architecture. Technical report, Dpt. of Computer Science, Univ. of Massachussets, Amherst, MA, USA, 2002.

[Rey87] C.W. Reynolds. Flocks, herds and schools : a distributed behavioral model. *Computer Graphics*, 21(4) :25–34, 1987.

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement learning : an introduction*. MIT Press, 1998.

[Wit77] I.H. Witten. An adaptive optimal controller for discrete-time markov environment. *Information and control*, 34 :286–295, 1977.