Apprentissage par Systèmes Multi-Agents Adaptatifs par feedbacks endogènes : prospectives et applications

Rapport de Master 2 Robotique : Décision et Commande (RODECO)
Parcours Électronique, Énergie électrique, Automatique (EEA)

Université Toulouse III
23 août 2017

Bruno DATO

Laboratoire d'accueil : Institut de Recherche en Informatique de Toulouse
Responsables de stage : Marie-Pierre GLEIZES, Nicolas VERSTAEVEL
Encadrants : Julien NIGON, Jérémy BOES, Pierre GLIZE
Équipe d'accueil : Systèmes Multi-Agents Coopératifs
Tuteur universitaire : Martin Cooper

Mots-clés : Apprentissage par feedback endogène, Systèmes multi-agents auto-adaptatifs, Auto-observation, Robotique

Résumé : Cette étude a pour objectif d'enrichir l'apprentissage par démonstrations et concevoir des méthodes d'auto-apprentissage en exploitant les actions et perceptions d'un système pour générer des feedbacks internes. Le verrou scientifique de cette étude est la génération de feedbacks endogènes lors d'un apprentissage. Nous explorerons notamment l'apprentissage d'un modèle de contrôle sans *a priori*. C'est-à-dire un modèle de contrôle construit à l'aide des actions et perceptions d'un système sans faire d'hypothèses sur la nature de celles-ci. Ce modèle lui permet d'évoluer dans la représentation qu'il a de son environnement, autrement dit, de ses perceptions. Pour aborder cette problématique, nous avons choisi de nous placer dans le cadre de l'apprentissage d'une tâche robotique.

Abstract : This work aims to enrich learning by demonstrations and to design self-learning methods by exploiting the actions and perceptions of a system to generate internal feedback. The scientific lock of this study is the generation of endogenous feedbacks during an apprenticeship. We will in particular explore the learning of a control model without *a priori*. That is, a learning process constructed using the actions and perceptions of a system without making assumptions about their nature. This model allows it to evolve in the representation it has of its environment, in other words, its perceptions. To address this problem, we have decided to consider the problem of learning robotic task.

Remerciements

Je remercie tout d'abord Marie-Pierre Gleizes qui m'a permis d'effectuer ce stage en me prenant dans son équipe pendant ces cinq derniers mois.

Je remercie ensuite Pierre Glize que j'avais rencontré il y a un an et demi maintenant et qui m'avait présenté les travaux de l'équipe. Je n'étais pas prêt à découvrir les système multi-agents à l'époque c'est pourquoi Marie-Pierre et lui même m'avaient conseillé de revenir l'année suivante, ce que j'ai fait avec plaisir.

Je tiens maintenant à remercier Nicolas Verstaevel pour son encadrement journalier, ses conseils, ses critiques et son côté roboticien.

J'aimerais également remercier Julien Nigon et Jérémy Boes qui m'ont guidé, aidé et encadré au quotidien. Ils m'ont eux aussi permis par leurs conseils et leurs critiques d'avancer dans mes travaux et de ne pas m'égarer dans la complexité.

Je remercie mes camarades de bureau Axel de Conti, Alexandre Perles et Tanguy Esteoule pour avoir répondu à mes interrogations spontanées.

Je remercie les chercheurs, doctorants et stagiaires qui de prêt ou de loin m'ont permis de faire avancer ma réflexion sur certains points de mes travaux.

Je voudrais aussi remercier tous les membres de l'équipe SMAC pour leur accueil, leur bonne humeur et les bons moments passés ensemble.

Je remercie enfin les enseignants et intervenants du parcours RODECO pour leur encadrement au cours de l'année et pour m'avoir permis de réaliser ce stage.

Table des matières

Introduction

Vers des systèmes autodidactes

Nous vivons dans un environnement qui regorge de systèmes artificiels dont le but est de nous assister dans notre quotidien. Toutes ces applications sont développées afin de servir un but précis défini avant leur conception. Cependant, de part la complexité de ces systèmes, il est impossible de prévoir à l'avance toutes les interactions qu'ils auront avec leur environnement. De plus, ces systèmes sont plongés dans un monde dynamique dans lequel divers dispositifs peuvent apparaître et disparaître. Face à ces besoins, il est légitime de penser que dans un futur proche, nous ne serons plus en mesure de concevoir et programmer tous ces systèmes, ils devront alors apprendre à être utiles de façon autonome et réactive. Autrement dit, ils devront agir et prendre des décisions sans l'intervention du concepteur.

Ce travail s'inscrit dans une problématique à long terme plus globale qui est de concevoir un système artificiel sans finalité *a priori* [d'Amico, 2016], c'est-à-dire que son utilité n'est pas explicitement définie à la création. Ce système doit être capable d'apprendre tout le long de sa vie afin de se rendre utile [Verstaevel et al., 2017].

Aujourd'hui, une approche fertile en robotique pour qu'un système puisse apprendre une tâche ou un service est l'apprentissage par démonstrations [Verstaevel, 2016]. Le système a alors besoin qu'un oracle lui dise ce qu'il est pertinent de retenir. Cet apprentissage par démonstrations se traduit par une personne qui pilote un robot pour lui montrer une tâche à effectuer. Qu'en est-il si le système se retrouve dans une situation qu'il ne connaît pas ou s'il n'y a personne pour faire cette démonstration ?

Objectifs et verrous scientifiques

Les travaux récents dans le domaine des systèmes multi-agents adaptatifs offrent des perspectives prometteuses au problème d'apprentissage par feedback endogène [d'Amico, 2016, Verstaevel, 2016]. Ces outils permettent d'obtenir des comportements collectifs fonctionnellement adéquats en temps réel tout en appréhendant des situations non prévues comportant des non linéarités, de la dynamique et des informations distribuées et bruitées.

Ce mémoire a pour objectif d'enrichir l'apprentissage par démonstrations et concevoir des méthodes d'auto-apprentissage en exploitant les actions et perceptions d'un système pour générer des feedbacks internes. Le verrou scientifique de cette étude est donc la génération de feedbacks endogènes lors d'un apprentissage. Nous explorerons notamment l'apprentissage d'un modèle de contrôle sans *a priori*. C'est-à-dire sans faire d'hypothèses sur la nature des actions et perceptions. Ce modèle lui permet alors d'évoluer dans la représentation qu'il a de son environnement, autrement dit de ses perceptions.

Un moyen de lever ce verrou est de doter le système de capacités d'auto-observation. Cette capacité est là pour lui permettre d'appréhender son évolution dans son environnement.

Ce système doit aussi posséder des propriétés de généricité et de passage à l'échelle. La généricité signifie que les méthodes d'auto-apprentissage mises en place doivent pouvoir s'appliquer à n'importe quel système doté d'actions et de perceptions. Enfin, en ce qui concerne le passage à l'échelle, cela veut dire que cet auto-apprentissage doit pouvoir s'adapter à de grandes quantités d'actions et de perceptions.

Cas d'application

Pour répondre à ces objectif et lever les verrous que nous venons de citer, nous avons travaillé sur un cas concret d'application qui est le contrôle de trajectoire d'un drone. Toutes les expérimentions ont été faites sur un simulateur de vol de drone mis en place à l'aide du moteur de jeu vidéo Unity.

Le problème du contrôle de trajectoire d'un drone dans un espace à trois dimensions peut être vu comme le déplacement de l'extrémité d'un bras robot dans un espace de même dimension. Ainsi, les méthodes mises en place au cours de cette étude pourraient aussi être appliquées à la commande d'un bras robotique.

Plan

Nous commencerons par faire un état de l'art des méthodes d'intelligence artificielles classiques et comment celles-ci ont été utilisées dans le domaine de la robotique. Cet état de l'art finira par la présentation du paradigme des systèmes multi-agents permettant d'introduire plus spécifiquement cette étude.

Ensuite, nous présenterons le cas d'application mis en place ainsi que les solutions apportées pour répondre à notre problème à l'aide des systèmes multi-agents.

Enfin, nous analyserons les résultats obtenus à l'aide de métriques que nous définirons dans ce dernier chapitre.

Chapitre 1

État de l'art

Ce chapitre propose un état de l'art des méthodes d'apprentissage existantes afin de situer cette étude et de s'inspirer des travaux déjà existants. Nous commencerons par présenter les familles d'intelligence artificielles classiques. Puis nous verrons comment elles ont été appliquées dans le domaine de la robotique. Enfin, nous présenterons l'approche des systèmes multi-agents qui servira de base aux contributions apportées par cette étude.

1.1 Apprentissage artificiel classique

Lorsque l'on parle d'apprentissage automatique en informatique, on distingue tout premièrement deux grande familles d'apprentissage : l'apprentissage supervisé et l'apprentissage par renforcement.

1.1.1 Apprentissage supervisé

L'apprentissage supervisé est une technique d'apprentissage automatique qui permet de construire des modèles à l'aide de données d'entrainement labellisées [Mohri et al., 2012]. Ces données d'entrainement sont des exemples de données pour lesquelles on connait déjà la nature. Généralement, chaque exemple de donnée est une paire constituée d'une entrée (un vecteur de données) et une sortie qui pour les données d'entrainement est connue, ce sont les labels. Un algorithme d'apprentissage supervisé construit alors un modèle à partir de ces données labellisées dans le but de généraliser et reconnaître de futures données inconnues. Ce type d'apprentissage permet de guider l'exploration de l'espace de recherche grâce à des données collectées hors ligne.

Parmi les techniques d'apprentissage supervisé, on distingue principalement des techniques communes telles que les k-Plus Proches Voisins (k-PPV), les mélanges de lois Gaussiennes (GMM) et les réseaux de neurones artificiels.

1.1.1.1 k-Plus Proches Voisins (k-PPV)

Cette technique de classification supervisée permet d'obtenir des performances très intéressantes à l'aide d'un algorithme très simple. À partir d'un échantillon d'observations et de leurs classes associées, c'est-à-dire leur label. Lorsqu'une nouvelle observation apparaît, la classe ou le label de cette observation est donnée par les k plus proches voisins de cette observation à l'aide d'une fonction de distance choisie au préalable (distance Euclidienne, de Minkowski ou de Mahalanobis par exemple) [Cover and Hart, 1967].

La valeur de k est choisie par l'utilisateur ($k \in \mathbb{N}$). On choisit généralement un k impair pour éviter une configuration telle que les plus proches voisins sont également répartis dans deux classes. Ainsi, le choix de k est très dépendant du jeu de données et il existe diverses heuristiques pour le choisir, par exemple en minimisant l'erreur de classification.

1.1.1.2 Les mélanges de lois Gaussiennes (GMM)

Cette technique fait partie des approches statistiques d'apprentissage supervisé. On se place ainsi dans le cadre Bayésien avec $o \in \mathbb{R}^n$ une observation et k la classe ou le label auquel elle appartient. On a alors la loi de réalisation de la classe $P(o/k)$ et sa distribution *a priori* $P(k)$. On souhaite connaître $P(k/o)$, la classe la plus probable produisant l'observation o. D'après la règle de Bayes, on a :

$$P(k/o) = \frac{P(o/k)P(k)}{P(o)} \tag{1.1}$$

On estime alors $\hat{k}$ par maximum de vraisemblance :

$$\hat{k} = \arg\max_k P(k/o) = \arg\max_k \frac{P(o/k)P(k)}{P(o)} = \arg\max_k P(o/k)P(k) \tag{1.2}$$

Il est nécessaire de connaître $P(o/k)$ et $P(k)$ pour tout k afin de prendre une décision sur $P(k/o)$.

On considère dans ce cas qu'un échantillon de données suit une loi de probabilité qui peut être décomposée comme un mélange de lois Gaussiennes de paramètres inconnus μ_k^i et Σ_k^i et pondérées par des coefficients π_k^i :

$$\begin{gathered} P(o/k) = \sum_{i=1}^{I} \pi_k^i N(o, \mu_k^i, \Sigma_k^i) \\ \text{avec} \quad N(o, \mu_k^i, \Sigma_k^i) = \frac{1}{\sqrt{(2\pi)^n \sqrt{det\Sigma_k^i}}} \exp(-\frac{1}{2}(o-\mu_k^i)^t(\Sigma_k^i)^{-1}(o-\mu_k^i)) \end{gathered} \tag{1.3}$$

Il existe ensuite plusieurs algorithmes EM ("Expectation Maximisation") qui permettent d'estimer π_k^i et $N(o, \mu_k^i, \Sigma_k^i)$ par des calculs itératifs de vraisemblance [Moon, 1996].

1.1.1.3 Les réseaux de neurones

Inspiré du neurone biologique, McCulloch et Pitts proposent en 1943 un modèle de neurone artificiel [McCulloch and Pitts, 1943].

Un neurone permet d'obtenir à partir de ses entrées x_i une sortie o qui partitionne l'espace du vecteur d'entrée $(x_1, x_2, ..., x_n)$ en 2 à l'aide d'une somme pondérée par les poids w_i, d'une fonction d'activation φ et d'un seuil θ tel que :

$$o = \varphi\left(\left(\sum_{i=1}^{2} x_i w_i\right) - \theta\right) \tag{1.4}$$

La fonction d'activation la plus commune est la fonction de seuil de Heaviside :

$$\forall x \in \mathbb{R}, \varphi(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases} \tag{1.5}$$

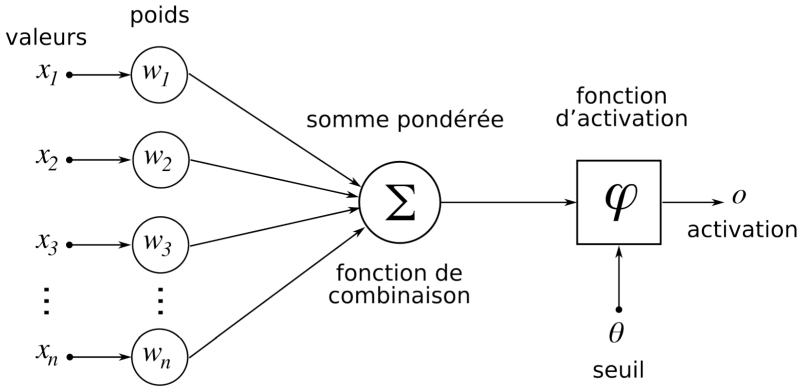


FIGURE 1.1 – Modèle de neurone artificiel formel

Il existe de nombreuses variantes à cette fonction telles que la fonction sigmoïde, la tangente hyperbolique, l'identité, le rectifieur...

On définit ainsi un perceptron ou réseau de neurones monocouche par p neurones tous connectés à l'ensemble des n entrées et ne contenant pas de cycles [Rosenblatt, 1958]. Plus le nombre de neurones est grand, plus la partition de l'espace d'entrée sera fine.

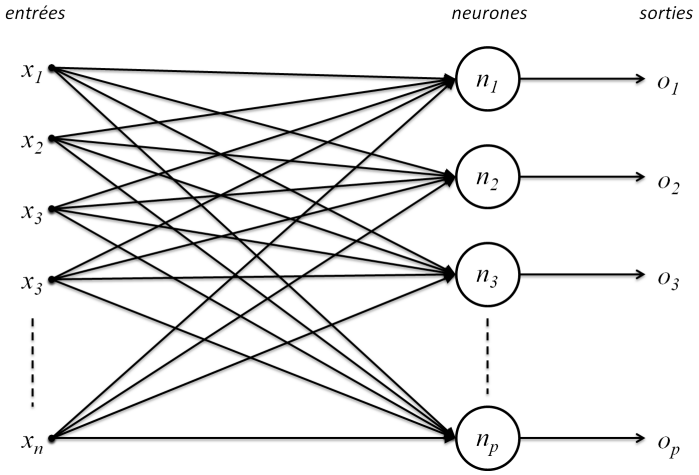


FIGURE 1.2 – Réseau de neurones monocouche

Pour que le réseau de neurones puisse converger vers les sorties attendues, on joue sur les valeurs des poids des différentes sommes pondérées qui sont initialisées aléatoirement. Ces poids sont ajustés à l'aide d'un algorithme de descente de gradient appliqué à chaque neurone. Pour que les poids convergent vers des valeurs "idéales", une grande quantité de données d'entrainement est nécessaire.

Afin d'améliorer les performance du perceptron originel, des variantes sont apparues comme les Perceptron multicouches (MLP) [Fiesler and Beale, 1996] qui possèdent des couches de neurones supplémentaires cachées. On trouve aussi les réseaux de neurones convolutionnels (CNN) [LeCun et al., 1989] qui sont très efficaces aujourd'hui pour la reconnaissance de formes sur des images.

1.1.2 Apprentissage par renforcement

1.1.2.1 Méthodes classiques

Un autre moyen d'explorer l'espace des états d'un système que l'apprentissage supervisé est l'apprentissage par renforcement.

L'apprentissage par renforcement consiste à apprendre une politique permettant de choisir quelle action effectuer en fonction de l'état du monde. Ces actions sont choisies de manière à maximiser une fonction de récompense définie *a priori*. Contrairement aux méthodes communes d'apprentissage automatique, les actions ne sont pas étiquetées en fonction des états du monde. L'entité apprenante doit explorer son environnement pour découvrir en fonction des états du monde dans lesquels il se trouve, quelles sont les actions qui permettent d'obtenir les plus grandes récompenses. Il est parfois possible qu'une action n'affecte pas seulement la récompense immédiate mais aussi les suivantes. L'apprentissage par renforcement se traduit alors par des jeux d'essais et d'erreurs mais aussi par des récompenses avec délais [Sutton and Barto, 1998].

Une variante de l'apprentissage par renforcement est l'apprentissage par renforcement inverse pour laquelle la fonction de récompense n'est pas connue mais doit être apprise.

1.1.2.2 Les algorithmes génétiques

Une approche dérivée de l'apprentissage par renforcement qui permet de répondre au problème de l'exploration de solutions est l'approche des algorithmes génétiques.

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes. Ces algorithmes s'inspirent de la théorie de l'évolution afin de résoudre des problèmes divers. Ils permettent d'obtenir une solution approchée à un problème d'optimisation lorsqu'il n'existe pas de solution exacte ou que la solution ne peut pas être trouvée dans un temps raisonnable. Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles [Holland and Reitman, 1977].

Afin de trouver de meilleures solutions que celles existantes dans la population, les solutions potentielles sont soumises à des mutations ou à des croisements. Les solutions apportant des améliorations sont mutées ou croisées dans le but de trouver des solutions plus performantes.

On utilise une fonction d'évaluation ou fonction de fitness pour mesurer l'efficacité de chaque solution et sélectionner les solutions les plus performantes.

1.1.3 Synthèse

En ce qui concerne l'apprentissage supervisé, on constate qu'il est indispensable de disposer de données labellisées qui sont fournies par un oracle. Ces données labellisées permettent de diminuer l'espace de recherche au cours de l'apprentissage cependant l'espace de recherche en est limité. La majorité de ces approches sont hors ligne, les données labellisées sont collectées dans une premier temps, puis elles servent ensuite à la génération du ou des modèles. En ce qui concerne notre étude, le but final est de s'affranchir de ces données fournies par une entité extérieure au système et que cet apprentissage soit en ligne. Cette approche présente cependant l'avantage de réduire l'espace de recherche.

Un des points fort de l'apprentissage par renforcement est qu'il effectue une exploration plus vaste de l'espace de recherche et que cette exploration est en ligne. Cependant, la contrepartie est que cette exploration peut être couteuse en terme de temps d'exécution et elle peut s'avérer

dangereuse pour un robot par exemple. L'apprentissage par renforcement permet de s'affranchir de données étiquetées mais il nécessite une fonction de récompense définie *a priori*. Cette fonction définie à la conception est une information donnée au système par une entité extérieure ce que l'on souhaite éviter. Les algorithmes génétiques présentent des caractéristiques intéressantes de part leur apprentissage au cours de l'exécution et leur généricité. Cependant, leur fonction de fitness est dépendante de la tâche à accomplir comme pour l'apprentissage par renforcement classique. L'apprentissage par renforcement inverse en revanche est une approche envisageable car elle permet au système de construire sa propre fonction de récompense qu'on peut imaginer adaptative et non dépendante d'une seule tâche définie à la conception.

On pourrait penser que les méthodes d'apprentissage non supervisé que nous n'avons pas citées peuvent répondre au problème de l'apprentissage endogène. Cependant, ces méthodes s'appliquent à des problèmes de classification qui ne correspondent pas à notre problématique. C'est plutôt comme un problème d'apprentissage "sans superviseur" qu'il faut considérer dans cette problématique. Cet apprentissage ne doit reposer que sur lui même et sur ses expériences, aucun acteur extérieur ne doit intervenir.

Les apprentissages supervisé et par renforcement seuls ne peuvent donc pas répondre à ce problème, il serait intéressant de les coupler afin de disposer d'un espace de recherche réduit initialement grâce à l'apprentissage supervisé puis d'explorer de nouvelles situations à partir de cet espace grâce à l'apprentissage par renforcement.

1.2 Apprentissage artificiel en robotique

Dans le cadre de notre étude, on s'intéresse plus spécifiquement aux techniques utilisées en robotique. Dans ce domaine, on retrouve ces deux grandes familles d'apprentissage à travers l'apprentissage par démonstrations pour ce qui est de l'apprentissage supervisé et l'apprentissage constructiviste en ce qui concerne l'apprentissage par renforcement.

1.2.1 Apprentissage par démonstrations

Nous avons vu que l'apprentissage supervisé ne permet pas de répondre à notre problématique, cependant, inspiré de ce dernier, l'apprentissage par démonstrations permet de répondre au problème de l'exploration de l'espace dans lequel évolue un système robotique par exemple. En effet, une démonstration permet de régler le problème du bootstrap en apprentissage, c'est-à-dire que si un système est livré à une exploration aléatoire, son apprentissage sera dépendant de l'exploration initiale qu'il a effectué. Son apprentissage sera alors biaisé [Mazac et al., 2015].

1.2.1.1 Principes

L'apprentissage par démonstrations ou apprentissage par imitation est né dans les années 1980. Plutôt que de programmer à la main toutes les actions qu'un robot aurait à effectuer durant sa vie, les concepteurs ont doté les robots de capacités d'apprentissage leur permettant de se former à de nouvelles tâches pour pouvoir s'adapter aux besoins des utilisateurs. Les principaux travaux sur cette approche ont été réalisés par [Argall et al., 2009] et [Schaal et al., 2003]. Inspiré de l'apprentissage dans la nature et notamment chez l'homme, l'apprentissage par démonstrations comprend une phase d'observation d'un comportement puis une phase d'imitation de ce comportement. D'après [Dautenhahn and Nehaniv, 2002], quatre questions clés ont été dégagées :

- Que faut-il imiter ?
- Comment faut-il imiter ?
- Quand faut-il imiter ?
- Qui faut-il imiter ?

Parmi ces problématiques, seulement les deux premières ont été abordées à ce jour.

1.2.1.1.1 Que faut il-imiter ?

Il faut imiter en général une tâche qu'un robot est en charge d'apprendre. Pour cela, toutes les perceptions de l'environnement qu'a le robot ne sont pas nécessairement utiles pour l'effectuer. L'importance de ces perceptions dépend de la métrique qui est utilisée pour évaluer le comportement du robot. Ainsi, seulement certaines perceptions du robot sont utiles en fonction des tâches, il ne faut donc pas prendre en compte celles qui sont inutiles.

1.2.1.1.2 Comment faut-il imiter ?

Si on garde l'exemple du robot, une fois que nous savons ce qu'il faut apprendre, il faut maintenant savoir comment est-ce que le robot va exécuter les comportements qu'il a appris afin de maximiser la métrique fixée précédemment. Un robot ne possédant pas la même incarnation physique qu'un humain, il ne peut pas agir comme lui mais il peut tout de même aller d'un point à un autre si c'est ce que l'on lui a appris à faire. C'est le problème de la correspondance défini par [Nehaniv et al., 2002].

L'apprentissage par démonstrations consiste à apprendre une politique à partir d'exemples fournis par un tuteur, appelé aussi oracle. C'est de l'apprentissage supervisé. Une politique est une association entre un état du monde et une action. L'état du monde est défini par les perceptions que le robot a de son environnement et de lui même. Pendant la démonstration, le robot apprend alors des séquences de paires <état du monde, action>. Le robot se crée ainsi une politique pour connaître les actions à effectuer pour toutes les situations qu'il a déjà rencontrée lors d'une démonstration. Cependant, si le robot rencontre une situation qu'il n'a jamais vu auparavant, sa politique ne lui permet pas de trouver l'action à effectuer.

1.2.1.2 Apprentissage par démonstrations et systèmes multi-agents

Afin de concevoir un système ambiant s'adaptant aux besoins des utilisateurs qui interagissent avec lui, il est possible de coupler l'apprentissage par démonstrations à un système multi-agent [Verstaevel et al., 2016]. On parle de système ambiant lorsque celui-ci est capable de s'adapter aux besoins de l'utilisateur sans que celui-ci n'ait à les spécifier explicitement.

Le système multi-agent permet d'associer une action à l'état courant du monde tel qui est perçu par le système apprenant. Ainsi, pendant la démonstration, le tuteur ou l'oracle étiquette directement les situations rencontrées et leurs actions associées.

1.2.1.3 Apprentissage par intention

L'apprentissage par démonstrations peut s'étendre à l'apprentissage par intention. En plus des états du monde, l'apprentissage par intention utilise la dynamique de transition de ces états de l'environnement afin de chercher les objectifs qui motivent le comportement du tuteur. Grâce à la connaissance de ces objectifs, le robot se comporte correctement lorsque qu'il rencontre des nouvelles situations [MacGlashan and Littman, 2015].

L'approche souvent utilisée est la même que pour un problème d'apprentissage par renforcement inversé dans lequel l'intention de l'oracle est modélisée à l'aide d'une fonction de récompense que l'on chercher à maximiser [Ng et al., 2000]. Cet apprentissage est plus générique que l'apprentissage par démonstrations simple car il permet de s'adapter à de nouveaux états du monde non rencontrés pendant la phase de démonstration.

1.2.1.4 Analyse

L'apprentissage par démonstrations est une méthode intéressante car elle permet de fournir un premier comportement à un système ne contenant aucune donnée initialement. Ce comportement constitue une base à partir de laquelle il peut évoluer. Cependant, un tel système n'est toujours pas complètement autonome car une entité extérieure doit lui fournir cette démonstration. De plus, l'apprentissage est bridé par l'espace exploré lors de la démonstration.

L'apprentissage par intention est d'autant plus intéressant qu'il permet au système d'apprendre les motivations cachées derrière une démonstration. Ceci lui permet de s'adapter à de nouvelles situations là où l'apprentissage par démonstrations ne pourrait pas le faire. Cependant, il y a toujours un oracle qui fournit au système des motivations qui ne lui sont pas propres. Cette méthode ne permet donc pas de se passer de feedbacks extérieurs.

1.2.2 Robotique développementale

Nous avons donc vu que les méthodes "classiques" d'apprentissage ne sont pas adaptées à notre problématique qui est celle de l'apprentissage par feedback endogène ou autrement dit de la conception d'un système autodidacte. Hors, c'est à cette problématique que la robotique développementale s'attaque en partie.

La robotique développementale est une branche récente de la robotique
[Weng et al., 2001, Zlatev and Balkenius, 2001]. Elle est née du principe qu'une intelligence naturelle et possiblement artificielle est le résultat d'une entité possédant les trois propriétés suivantes :

- Elle est incorporée dans un corps capable de percevoir et d'agir ;
- Elle évolue dans un environnement social et/ou physique ;
- Les interactions avec cet environnement physique et/ou social font émerger des connaissances de plus en plus complexes qui sont le résultat d'un processus de développement épigénétique prolongé.

L'approche développementale s'inspire de la nature (plus particulièrement des enfants) et tend à créer un programme capable de développer ses propres connaissances et capacités en interagissant avec son environnement. Le challenge est d'apprendre en permanence à partir de connaissances déjà acquises pour atteindre des niveaux de connaissance plus sophistiqués [Guerin, 2011].

Plutôt que de programmer un adulte pouvant comprendre des concepts évolués, l'idée est de concevoir une intelligence artificielle capable d'apprendre des patterns de données et d'associer ces données pour en faire des concepts plus abstraits. Le but est alors à partir de ces concepts d'en créer d'autres de niveaux de plus en plus élevés afin d'apprendre des concepts plus complexes que l'on trouve généralement chez un être humain adulte. C'est ce que l'on appelle l'apprentissage constructiviste.

1.2.2.1 Apprentissage constructiviste

La théorie de l'apprentissage constructiviste a été développé par Piaget dès 1923. Elle met en avant l'activité d'un sujet qui se construit une représentation de la réalité qui l'entoure. Cette réalité se construit en interne chez le sujet ce qui correspond aux mécanismes que l'on recherche dans notre étude.

1.2.2.1.1 Apprentissage par *schema*

Selon [Piaget, 1976] les connaissances se construisent à partir de *schemas*, des unités de connaissances. Afin de créer une base de connaissances plus large, nous utilisons plusieurs de ces *schemas* pour en créer d'autres de plus haut niveau.

Les travaux de Piaget ont ainsi amené à plusieurs avancées dans le domaine de l'apprentissage. [Drescher, 1991] présenta en 1991 des *schemas* à trois parties : contexte, action et résultat. Ce type de *schema* peut être vu comme des prédictions de ce qu'il arrive si une action est exécutée dans un certain contexte.

[Holmes et al., 2005] généralisent et améliorent le mécanisme de Drescher en voyant le problème d'apprentissage comme une POMDP (Partially Observable Markov Decision Process).

[Chaput, 2004] va plus loin et développe une architecture pour l'apprentissage contructiviste (Constructivist Learning Architecture) basée sur la théorie du développement cognitif des nourrissons de [Cohen, 1998]. C'est une théorie un peu plus détaillée que celle de [Piaget, 1976] concernant le mécanisme de traitement de l'information requise. Les nourrissons apprennent à traiter l'information à des niveaux d'abstraction de plus en plus élevés en formant des unités ou *schema* de niveau supérieur à partir des relations entre les unités de niveau inférieur. Il existe un biais lors du traitement de l'information en utilisant les unités de haut niveau. Cependant, si l'entrée d'un *schema* devient trop complexe, le nourrisson descend à un niveau inférieur et tente d'affiner son abstraction afin de pouvoir gérer l'information complexe au plus haut niveau. Les principes de Cohen donnent une stratégie pour abstraire l'apprentissage. À partir des données des capteurs bruts, le nourrisson doit seulement prêter attention aux abstractions qu'il estime utiles. Il préfère trouver des relations entre des abstractions de haut niveau déjà créées plutôt que se concentrer à chaque fois sur celles de bas niveau.

[Stojanov et al., 1997] ont utilisé un mécanisme similaire aux modèles d'apprentissage de Drescher et Chaput. Ils confirment la pensée de [Brooks, 1991] que pour concevoir des agents intelligents, il est nécessaire de créer un système capable de survire dans un environnement dynamique. Cependant, il ne faut pas tomber dans la réactivité pure et se concentrer aussi sur la représentation interne des connaissances. Dans leur modèle, l'agent intelligent apprend des "attentes" grâce aux interactions avec son environnement. Ils ont aussi mis en place des liens associatifs au sein de la mémoire apprise, c'est-à-dire l'apprentissage de liens entre des séquences d'actions possibles. Ce mécanisme permet d'apprendre des cycles qui connectent des chaînes de *schemas*. Ces cycles modélisent de hauts niveaux de connaissances. D'autres mécanismes permettent de détecter ces cycles et de trouver les actions pour se déplacer d'un cycle à l'autre.

Les travaux de [Perotto and Álvares, 2006] introduisent la modification de *schema* avec trois méthodes : la différenciation, l'ajustement et l'intégration (fusion). La fusion pourrait s'appliquer aux méthodes vues précédemment afin de généraliser des prédictions au lieu de créer plusieurs *schemas* avec la même prédiction.

Plus récemment [Mazac et al., 2015] a proposé une approche multi-agent pour un apprentissage constructiviste en environnement continu appliqué à l'intelligence ambiante.

1.2.2.1.2 Apprentissage par motivation intrinsèque

Bien que ce point ne soit pas dans les objectifs principaux de notre problématique, l'existence de motivations intrinsèques est indispensable à la conception d'un système auto-apprenant. Seulement de telles motivations permettent de faire évoluer cet apprentissage.

En effet, un autre point important de l'apprentissage développemental ou constructiviste est la motivation intrinsèque, c'est-à-dire que l'agent apprenant puisse être capable d'évoluer à travers plusieurs niveaux d'intelligence en étant motivé par sa propre curiosité interne.

[Oudeyer et al., 2007] s'est attaqué à ce problème en réutilisant les *schemas* de Drescher et en statuant que l'apprentissage est motivé par une récompense interne proportionnelle à la diminution de l'erreur dans les prédictions. Ce mécanisme est appelé "Intelligent Adaptive Curiosity". C'est un apprentissage par renforcement dans lequel l'agent choisit les actions qui maximisent son processus d'apprentissage. Un point essentiel de ce mécanisme est qu'il stocke les *schemas* dans des régions. Lorsqu'une région possède un certain nombre de *schemas*, elle est divisée en deux de manière à minimiser la somme des variances de *schemas* de chaque région. Chaque région a alors son propre processus d'apprentissage. La découpe de régions fait apparaître des stades d'intelligence ou de comportement car il est possible qu'une nouvelle région conduise à une diminution nette de l'erreur de prédiction (et donc un gain important). Cette région sera ainsi favorisée pour le choix des actions. Cette méthode permet alors d'accélérer l'exploration de régions inconnues par rapport aux méthode utilisant des explorations aléatoires.

[Bondu and Lemaire, 2007] ont proposé une amélioration à ce mécanisme d'apprentissage, notamment à la façon dont sont sélectionnées les zones d'exploration afin d'améliorer les prédictions. C'est un parallèle entre le mécanisme d'apprentissage à curiosité interne et le mécanisme d'apprentissage actif. L'apprentissage actif est un type d'apprentissage supervisé pour lequel l'agent apprenant peut demander à son tuteur les exemples qui accéléreront selon lui le progrès de son apprentissage. [Bondu and Lemaire, 2007] ont proposé un critère afin d'équilibrer l'exploration et l'exploitation pour que l'agent ne se retrouve pas dans une situation où il ne peut rien apprendre ou bien dans une situation où il connaît déjà tout ce qu'il y a à savoir.

Une autre point de vue en ce qui concerne les stades d'apprentissage est celui de [Lee et al., 2007]. Ils proposent de fixer les stades d'apprentissage plutôt que de les voir émerger. Ces stades sont représentés comme des contraintes qui sont levées à chaque stade. Ces contraintes sont présentes pour faciliter l'apprentissage à ses débuts et l'affiner au fur et à mesure que l'apprentissage se développe et atteint ainsi les stades de plus haut niveau.

1.2.2.2 Principes de la robotique développementale

Afin de guider les recherches de cette nouvelle approche en pleine emergence, cinq principes ont été définis [Stoytchev, 2006].

1.2.2.2.1 Le principe de vérification

La robotique développementale est apparue comme une réaction à l'incapacité des architectures traditionnelles en robotique de s'adapter aux tâches nécessitant un niveau d'intelligence proche de celui de l'humain. Face à la complexité grandissante des nouveaux systèmes robotiques et aux tâches qui leur seront associées, il est évident que les programmeurs de ces systèmes seront un jour dépassés et ne pourrons prévoir à la conception toutes les interactions qu'un robot pourra avoir avec son environnement. En effet, il est naïf de penser que la complexité du monde qui nous entoure puisse être synthétisée à l'aide de programmes informatiques avant même que

le robot n'ai pu interagir avec son environnement à l'aide de ses capteurs et actionneurs. De plus, un autre problème concernant la conception telle qu'elle est faite aujourd'hui est qu'elle utilise trop d'hypothèses cachées que le robot n'a aucun moyen de vérifier ou de tester car elles ne sont pas explicites de son point de vue. Afin de s'adapter aux situations où ces hypothèses sont violées, le robot devrait être capable de tester et vérifier tout ce qu'il apprend. C'est ainsi qu'a été introduit le *principe de vérification* par Richard Sutton :

Principe de vérification : *Une intelligence artificielle peut créer et maintenir des connaissances uniquement dans la mesure où elle peut vérifier cette connaissance* [Sutton, 2001].

Selon Sutton, "la clé d'une IA réussie est qu'elle peut dire elle-même si elle fonctionne correctement". Ainsi, une IA doit être en charge de son propre apprentissage et si elle n'est pas en mesure de tester un concept, alors elle ne doit pas l'apprendre. Ce principe change considérablement les pratiques traditionnelles de la robotique autonome et force les programmeurs à repenser la façon dont les données sont représentées dans l'architecture du robot.

1.2.2.2.2 Le principe d'incorporation

Afin d'avoir les moyens de vérifier tout ce qu'il apprend, le robot doit pouvoir agir sur son environnement et donc avoir un corps physique. De plus, ce corps est d'autant plus important car il définit la partie de l'environnement la plus consistante, la plus prédictive et la plus vérifiable. Sinon, il ne devrait pas y avoir de différence entre le corps et l'environnement. Cependant, il ne doit pas y avoir de différence dans la vérification des connaissances que ce soit pour l'exploration de propriétés du corps ou l'exploration du monde bien que cette dernière vienne après. De plus, distinguer le corps de l'environnement externe devrait être chose aisée car certains évènements ne peuvent être expérimentés que par le propriétaire du corps. Rochat [Rochat, 2003] les appelle les évènements *auto-spécifiants* et les liste selon 3 catégories :

1) Les boucles efférentes-afférentes (e.g., bouger sa main et la voir bouger) ;
2) Le double touché (e.g., se faire toucher les deux index) ;
3) Les comportements de vocalisation suivis de l'audition de leurs résultats (e.g., crier et s'entendre crier).

Ces évènements sont multimodaux car ils impliquent plusieurs senseurs ou effecteurs et ils sont aussi auto-vérifiables car il est toujours possible de répéter une action et d'en observer le même résultat.

1.2.2.2.3 Le principe de subjectivité

Si un robot ne peut apprendre et conserver seulement la connaissance qu'il peut vérifier, ce que le robot apprend doit être une fonction de ce qu'il a expérimenté à travers ses propres capteurs et actionneurs, son apprentissage dépend donc de sa propre expérience. En effet, si tout apprentissage doit être vérifié à travers cette expérience individuelle, alors cet apprentissage ne peut être que subjectif [Ayer Alfred, 1952]. Ce principe introduit alors des limitations à ce qu'il est possible d'apprendre pour un agent spécifique. Il y a deux types de limitations : sensorimotrices et expérimentales.

1.2.2.2.3.1 Les limitations sensorimotrices

La première limitation imposée par le principe de subjectivité est que le robot est limité dans son apprentissage par ses capacité sensorimotrices, c'est-à-dire par ce qu'il est capable de faire.

Il en est de même pour l'homme c'est pourquoi il n'a cessé depuis qu'il existe de développer de nouvelles technologies et de nouveaux outils afin d'étendre les capacités de son propre corps.

1.2.2.2.3.2 Les limitations expérimentales

L'expérience contraint aussi ce que le robot peut apprendre étant donné que cet apprentissage se construit à partir de l'histoire des interactions entre le robot et son environnement. En effet, le temps est un facteur clé dans l'apprentissage développemental car cet apprentissage nécessite des interactions avec le monde extérieur au robot et il y a une limite à la vitesse que peuvent avoir ces interactions. Il est tout de même possible d'accélérer la vitesse d'apprentissage en se reposant sur l'expérience des autres. Le principe de subjectivité n'est alors pas violé pour autant car une vérification est tout de même faite. Nous les humains utilisons tout le temps l'expérience des autres à l'aide de l'écriture ou de la parole par exemple.

Une question est tout de même soulevée, si tout ce qui est appris est subjectif, comment est ce que deux personnes peuvent avoir une compréhension commune à propos de quelque chose ? Pour répondre à cette question, il est nécessaire que les représentations des connaissances des deux agents soient qualitativement les mêmes.

"*Nous définissons l'identité qualitative et la différence des expériences sensorielles de deux personnes en termes de similitude et de dissemblance de leurs réactions à des tests empiriques. Pour déterminer, par exemple, si deux personnes ont le même sens de la couleur, nous observons si elles classent les étendues de couleurs avec lesquelles elles sont confrontées de la même manière. Lorsque l'on dit qu'un homme est aveugle en terme de couleur, ce que nous affirmons est qu'il classe certaines étendues de couleurs différemment de comment elles seraient classées par la majorité des gens.*" [Ayer Alfred, 1952]

Le fait que les humains aient tous un corps semblable et évoluent dans le même milieu contribue aussi au fait qu'ils ont une compréhension partagée. Une autre façon de voir les choses est de dire que c'est notre corps qui dicte la façon dont nous pensons [Pfeifer and Bongard, 2006]. En effet, c'est le corps et les interactions avec l'environnement qui façonnent la façon dont nous apprenons et nous agissons à notre plus bas âge (développement individuel). Ensuite vient le côté social de notre apprentissage (développement social) [Asada et al., 2009].

Ainsi, les représentations internes des connaissances se doivent d'êtres flexibles au cours du temps afin de s'adapter aux nouvelles expériences.

1.2.2.2.4 Le principe de mise à terre

Le principe de mise à terre permet de définir ce qui constitue une vérification valide. Nous avons vu précédemment qu'il est nécessaire d'agir sur l'environnement pour effectuer toute vérification et donc la valider. Cependant, pour qu'une action soit utile pour une validation, il faut en connaître son résultat. La mise à terre consiste alors à définir des paires ACTION-RÉSULTAT (ou COMPORTEMENT-OBSERVATION). Ces paires doivent être associées à une confiance définie de manière probabiliste pour ne pas qu'une paire soit simplement une coïncidence. Cependant, une action ou son résultat peuvent être des processus impliquant plusieurs niveaux de détail. Ce que l'on identifie ce sont les caractéristiques persistantes d'une séquence de vérification qui sont constantes quelque soit le contexte, c'est-à-dire les invariants sensorimoteurs. Un de ces invariant est défini comme la contingence temporelle. C'est le temps relatif entre la co-occurrence d'une action et de son résultat. Sa détection permet de détecter quelles perceptions sont directement reliées au corps du robot [Stoytchev et al., 2007]. Il est alors pos-

sible de classifier le stimuli perçu comme “interne” ou “externe” et ainsi détecter quand quelque chose ne va pas.

1.2.2.2.5 Le principe de l’exploration graduelle

Le principe de l’exploration graduelle expose qu’il est impossible de tout apprendre en même temps. On apprend à ramper avant de marcher par exemple. Pour tout processus d’apprentissage développemental, il existe alors des étapes et des transitions dont leur définition est beaucoup discutée. En regardant plusieurs êtres vivants, on peut remarquer que le stade de la locomotion autonome varie de façon significative en fonction des espèces [Tomasello and Call, 1997, Power, 1999].

Pour ce qui concerne l’exploration, il est aussi indispensable de définir quel processus l’initie et quel processus la termine. D’après le principe de l’exploration graduelle, l’exploration est auto-régulée et commence par ce qui est le plus vérifiable pour finir par ce qui l’est le moins. On cherche alors à explorer en priorité ce qui vérifiable moyennement afin de pouvoir apprendre un peu sans pour étant être perdu complètement. La recherche est donc guidée non pas par une récompense externe comme dans l’apprentissage par renforcement mais par le besoin interne de réduire l’incertitude [Gibson, 1969].

1.2.2.3 Analyse

L’approche de la robotique développementale correspond aujourd’hui à ce que l’on pourrait aussi appeler l’apprentissage par feedback endogène car elle permet à un système d’apprendre son comportement interne puis de découvrir son environnement seulement à partir de ses actions et perceptions. Cette approche est destinée aux systèmes “vierges” qui apprennent à partir de rien et créent ainsi plusieurs niveaux d’apprentissage. Cette vision nouvelle de l’apprentissage en informatique qui fait appel à d’autres disciplines telles que la psychologie et l’étude de la cognition chez les êtres vivants est un domaine de recherche prometteur.

1.2.3 Synthèse

Pour notre étude, nous combinerons les avantages des apprentissages supervisés et par renforcement. Nous nous baserons sur un apprentissage par démonstrations afin d’apprendre un premier comportement et éviter le problème du bootstrap. Cet apprentissage permettra de disposer d’un espace de recherche réduit à partir duquel nous allons ajouter une exploration afin d’affiner la connaissance de cet espace. Cependant, pour mettre en place cette observation, il est nécessaire de doter notre système de capacités d’introspection.

1.3 Les Systèmes Multi-Agents

Nous proposons d’utiliser le paradigmes des systèmes multi-agents afin de mettre en place une auto-observation nécessaire à l’exploration de l’espace de notre système robotique.

Un Système Multi-Agent (SMA) est un système informatique décentralisé composé d’un ensemble d’entités autonomes en interaction entre elles et avec leur environnement. Chacune de ces entités que l’on nomme agents ont des caractéristiques et des objectifs propres. Ils respectent le principe de localité : un agent n’a qu’une connaissance partielle du monde. Tout agent composant le SMA possède ainsi ses propres perceptions, ses propres capacités d’acquisition et ses propres buts. Il est autonome c’est-à-dire capable de dire non. Dans un premier temps, nous

présenterons les agents. Ensuite nous détaillerons le fonctionnement d'un système multi-agent en explicitant ses propriétés principales et notamment comment un SMA est couplé avec son environnement. Enfin, nous montrerons quelles sont les propriétés additionnelles qu'un système multi-agent adaptatif apporte.

1.3.1 Les agents

Définition : "Un agent est une entité réelle ou virtuelle, évoluant dans un environnement, capable de le percevoir et d'agir dessus, qui peut communiquer avec d'autres agents, qui exhibe un comportement autonome, lequel peut être vu comme la conséquence de ses connaissances, de ses interactions avec d'autres agents et des buts qu'il poursuit" [Ferber, 1999].

Un agent est une entité autonome physique ou logicielle. Il ne possède qu'une représentation partielle de l'environnement dans lequel le système global et plongé. Cette représentation partielle dépend des perceptions de l'agent. Un agent est capable d'interagir avec son environnement grâce à ses compétences tout en suivant ses propres objectifs. Pour les satisfaire, il communique aussi avec d'autres agents.

Un agent effectue un cycle de type perception, décision et action. Durant l'étape de perception, l'agent reçoit des informations provenant de son environnement qui lui permettent de modéliser une représentation utile pour sa prise de décision. Ces informations sont obtenues à partir des autres agents avec lesquels il communique ou à partir d'une source de données à laquelle il est directement relié. Puis, durant l'étape de décision, l'agent exploite la représentation qu'il s'est forgée afin de choisir les actions lui permettant d'accomplir ses objectifs ou de s'en approcher. Enfin, durant l'étape d'action, l'agent exécute les actions qu'il a sélectionnées et agit ainsi directement sur son environnement ou sur les autres agents. Ces actions peuvent être de simples messages envoyés à d'autres agents ou des actions physiques accomplies grâces à des effecteurs dans sont environnement.

1.3.2 Le système multi-agent

Un système multi-agent est constitué d'un ensemble d'agents en interaction. Ces agents, de part leurs comportements individuels, peuvent d'accomplir une tâche plus globale qui est à l'échelle du système et dont ils n'ont pas conscience, c'est l'échelle macroscopique. Dotés de leurs propres objectifs et de leurs interaction coopératives ou conflictuelles à l'échelle microscopique du système, c'est-à-dire l'échelle au niveau local d'un agent, les agents peuvent sous certaines conditions faire émerger une organisation donnant naissance à un résultat à l'échelle macroscopique du système.

1.3.3 L'environnement

Un système multi-agent n'existe pas sans environnement car c'est au travers de lui que les agents sont capables d'exister et d'agir. Il n'existe pas de définition approuvée par la communauté des chercheurs pour cette notion [Weyns et al., 2004]. Cependant, plusieurs propriétés essentielles ont été dégagés par [Russell and Norvig, 2002]. L'environnement est :

- **Accessible ou inaccessible** : indique si les agents ont accès à l'environnement dans sa globalité ou non.
- **Déterministe ou non déterministe** : indique si un changement de l'état de l'environnement est dû à l'état précédent couplé aux actions choisies par les agents ou si ce n'est pas le cas.

- **Statique ou dynamique** : indique si l'environnement peut évoluer pendant qu'un agent est en train de prendre une décision ou s'il ne peut pas.
- **Discret ou continu** : indique si le nombre de perceptions et d'actions est limité ou non.

À l'échelle macroscopique, si on considère le système multi-agent dans sa globalité, l'environnent représente tout ce qui est extérieur au système (les agents font partie du système, ils ne sont donc pas dans l'environnement). Si on considère maintenant un agent seul, l'environnent est en plus constitué des autres agents du système.

Les agents en interaction avec cet environnement peuvent avoir un impact sur celui-ci et ils sont informés s'il change. On dit qu'un système multi-agent est couplé avec son environnement car il agit sur l'environnement et l'environnement agit sur lui.

1.3.3.1 L'autonomie

Définition : Un système multi-agent est un système autonome car il est constitué d'agents qui sont eux même autonomes. Chaque agent étant responsable des décision qu'il prend et de leur exécution, ils confèrent ainsi une autonomie au système multi-agent global. Un système multi-agent ne dépendant d'aucune entité qui lui est extérieure est alors autonome grâce aux agents qui le composent [Serugendo et al., 2011].

1.3.3.2 La distribution

Comme nous l'avons dit précédemment, un système multi-agent est un système distribué où les connaissances et le contrôle sont réparties entre les agents et la coopération entre ces agents peut permettre de réaliser une tâche à l'échelle macroscopique du système. Chaque agent possède une représentation de l'environnement qui dépend de ses perceptions et de ses compétences. Les systèmes multi-agents auxquels nous nous intéressons sont caractérisés par le fait qu'à l'échelle individuelle ou microscopique, les agents ne connaissent pas cette tâche globale mais tous réunis, ils constituent un ensemble disposant de toutes les connaissances et compétences nécessaire pour réaliser cette tâche.

1.3.3.3 L'émergence

L'émergence est une propriété que certains systèmes multi-agents possèdent. À travers cette propriété, un ensemble d'agents peut accomplir plus que la simple somme des compétences locales de chaque agent. Lorsque l'on parle d'émergence, on pense souvent aux mouvements collectifs que peuvent faire les bancs de poissons ou les nuées d'oiseaux. Cela va au-delà de la notion de calcul distribué car dans ce cas là, il est possible de déterminer une fonction macroscopique du système en analysant les agents à l'échelle microscopique. On parle d'émergence à partir du moment où la connaissance de la fonction locale d'un agent par un acteur extérieur au SMA ne permet pas d'inférer la fonction globale du système par ce même acteur [Gleizes et al., 2008].

1.3.4 Les systèmes multi-agents auto-adaptatifs

L'approche par système multi-agent propose un autre type de SMA avec des propriétés supplémentaires que sont les systèmes multi-agents auto-adaptatifs (*Adaptive Multi-Agent Systems*, AMAS). Ce sont des systèmes multi-agents qui ont la capacité de s'auto-adapter. Les agents qui constituent ces systèmes possèdent des comportements coopératifs qui sont à l'origine de cette auto-organisation.

1.3.4.1 La coopération

Nous avons vu précédemment qu'il existe un couplage entre un système multi-agent et son environnement par le biais d'interactions. Selon [Kalenka and Jennings, 1999], il existe trois types d'activité entre le système et son environnement :

- **Coopérative** : Les échanges entre le système et l'environnement apportent des bénéfices mutuels.
- **Neutre** : Les actions de l'un n'entravent ni ne favorisent l'activité de l'autre.
- **Antinomique** : Les actions de l'un empêchent l'autre d'accomplir son activité.

Il est nécessaire que le système soit dans un état coopératif pour permettre l'auto-organisation. On souhaite ainsi que toutes les interactions soient coopératives.

L'environnement étant complexe et dynamique, il est difficile d'atteindre et de demeurer dans cet état coopératif. Les agents qui composent le système doivent avoir la capacité de savoir s'ils sont dans un état de coopération ou non. Ainsi, si un agent n'est pas dans une situation de coopération, il doit être capable de revenir à un état coopératif à travers la résolution de cette situation de non coopération.

1.3.4.2 L'adéquation fonctionnelle

L'adéquation fonctionnelle est une notion qui permet de déterminer si un système fonctionne correctement ou non. En général, un système est considéré comme fonctionnellement adéquat par un observateur externe au système lorsqu'il exécute la fonction pour laquelle il a été conçu. Au sein d'un système multi-agent auto-adaptatif, le système lui-même doit pouvoir évaluer son adéquation fonctionnelle à l'aide des agents qui le composent. Cependant, les agents peuvent seulement évaluer des critères locaux. [Glize, 2001] a démontré qu'un système dont les agents sont tous dans un état coopératif est fonctionnellement adéquat. Cette démonstration repose sur la définition de l'adéquation fonctionnelle : *Un système fonctionnellement adéquat est un système qui n'a aucune activité antinomique sur avec environnement.*

Théorème de l'Adéquation Fonctionnelle : *Pour tout système fonctionnellement adéquat, il existe au moins un système à milieu intérieur coopératif qui réalise une fonction équivalente dans le même environnement.* L'expression *système à milieu intérieur coopératif* signifie que toutes les interactions entre les parties qui composent le système sont coopératives.

D'après ce théorème, pour tout problème possédant une solution calculable, il existe un système multi-agent adaptatif pour lequel si tous les agents sont en état de coopération, le système peut résoudre ce problème.

1.3.4.3 Les situations de non coopération

Lorsqu'un agent n'est pas en état coopératif, il est dans une situation de non coopération. Ces situations de non coopération arrivent lorsqu'il y a un défaut de perception, de décision ou d'action. [Georgé et al., 2011] en ont défini plusieurs types pour aider à leur identification :

- **Incompréhension** : l'agent ne peut pas extraire d'information du signal perçu.
- **Ambiguïté** : l'agent peut interpréter le signal perçu de plusieurs manières.
- **Incompétence** : l'agent ne peut parvenir à aucune décision à partir de ses connaissances actuelles.
- **Improductivité** : les décisions de l'agent n'aboutissent à aucune action.
- **Concurrence** : l'action d'un agent aura les mêmes conséquences que l'action d'un autre agent.

- **Conflit** : l'action sera incompatible avec celle d'un autre agent.
- **Inutilité** : l'action n'aura aucune conséquence sur son environnement.

Un agent doit être capable de déterminer s'il est dans une de ces situations pour que la résolution puisse se faire localement par les agents. Ils modifient l'organisation afin que le système puissent revenir ou atteindre un état coopératif. Les agents disposent trois moyens pour modifier l'organisation du système :

- **Ajustement** : l'agent modifie son comportement interne en ajustant ses paramètres internes.
- **Réorganisation** : l'agent change ses relations avec les autres agents (il arrête d'interagir avec un agent donné, il prend contact avec un nouvel agent, ou il modifie l'importance des relations existantes).
- **Ouverture** : l'agent décide de s'auto-détruire ou bien de créer un nouvel agent.

Lors de la conception, le concepteur détermine quelle résolution doit être appliquée pour chaque situation de non coopération pour qu'un agent sache laquelle appliquer lorsqu'il les rencontre.

1.3.4.4 La méthode ADELFE

Un agent possède deux comportement distincts : le comportement nominal et le comportement coopératif. Lorsqu'il est dans le comportement nominal, il est guidé par son objectif local et il est dans un état de coopération. C'est cet état qui assure l'adéquation fonctionnelle du système. Le comportement coopératif est le comportement adopté par l'agent lorsqu'il est dans une situation de non coopération. Ce comportement n'est plus dirigé par un objectif local mais par la résolution de la situation de non coopération.

La méthode ADELFE permet de guider la conception d'un système multi-agent adaptatif en utilisant une conception ascendante [Bonjean et al., 2014]. Cette méthode permet de concevoir un système ayant la capacité de maintenir des interactions coopératives entre les agents qui le constituent.

1.3.5 Analyse

L'approche des systèmes multi-agents et notamment celle des systèmes multi-agents auto-adaptatifs est une approche intéressante car elle peut conférer à un système d'apprentissage des propriétés d'adaptation pour faire face à des situations imprévues. Ces situations sont inévitables dans le cadre d'un apprentissage interne à un système. Les systèmes multi-agents auto-adaptatifs étant une solution envisageable, il est nécessaire de disposer d'un formalisme pour effectuer un tel apprentissage.

1.4 Apprentissage de contextes par agents auto-adaptatifs

Afin de faire face à la complexité du monde qui nous entoure (non linéarité, dynamique, information distribuée, données bruités et imprédictibilité), les systèmes multi-agents auto-adaptatifs sont une des approches les plus prometteuses. Pour beaucoup d'applications telles que le contrôle de processus biologiques [Boes et al., 2015], le contrôle optimal de moteur [Boes et al., 2014] ou encore l'apprentissage en robotique [Verstaevel, 2016], on décompose le problème en plusieurs sous-problèmes : on mappe les situations rencontrées par le système, c'est-à-dire que l'on associe à un certain état du système que l'on appelle contexte, les actions qu'il est pertinent d'exécuter. Un contexte caractérise un état de l'environnement dans lequel le système se trouve, c'est-à-dire un jeu de perceptions car c'est à travers ses perceptions que le système voit son

environnement. À travers différentes applications, une architecture récurente est apparue et a été nommée *Self-Adaptive Context-Learning Pattern* (SACL) [Boes et al., 2015].

1.4.1 Self-Adaptive Context-Learning Pattern

Self-Adaptive Context-Learning Pattern est composé d'un *mécanisme d'adaptation* et d'un *mécanisme d'exploitation* (figure 1.3. Le *mécanisme d'adaptation* fournit au *mécanisme d'exploitation* des informations concernant les actions qu'il est possible d'effectuer pour la situation courante [Boes et al., 2015].

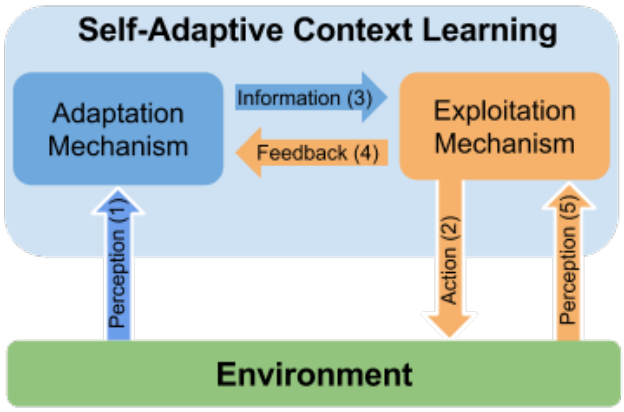


FIGURE 1.3 – Schéma du Self-Adaptive Context-Learning Pattern

1.4.1.1 Le mécanisme d'exploitation

Le *mécanisme d'exploitation* est donc un contrôleur, il choisit l'action qui est la plus adéquate parmi celles qui ont été proposées et il l'exécute. Le *mécanisme d'exploitation* communique aussi des informations au *mécanisme d'adaptation*, il lui envoie un feedback sur les données qu'il a reçues. Ce feedback est composé de l'action qui a été choisie par le *mécanisme d'exploitation* et une évaluation quantitative de l'utilité des dernières informations reçues.

1.4.1.2 Le mécanisme d'adaptation

Le *mécanisme d'adaptation* a pour but de construire, maintenir et fournir des connaissances fiables à propos du contexte courant et des actions possibles. Il est auto-adaptatif pour pouvoir se mettre à jour en fonction de la dynamique du *mécanisme d'exploitation* et de l'environnement.

Le *mécanisme d'adaptation* est construit à l'aide d'un système multi-agents adaptatifs composé de plusieurs agents différents : les agents percepts, les agents contextes et l'agent head.

Les Agents Percepts

Ils représentent une interface entre le système et son environnement. Chacun de ces agents est associé à une source de donnée ou une perception qui peut être un capteur par exemple. Chaque *Agent Percept* transmet à chaque *Agent Contexte* la valeur courante qu'il perçoit. Enfin, il y a autant d'*Agents Percepts* qu'il y a de sources de donnée dans le système.

Les Agents Contextes

Ce sont ces agents qui détiennent les connaissances du système. Leur nombre grandit avec la complexité de la tâche globale à effectuer par le système. Ces agents représentent, pour une situation donnée, une action et les effets que celle-ci est censée produire sur l'environnement. Le contexte, l'action et ses effets sont appris au cours de l'exécution. Chaque *Agent Contexte* possède des *plages de validité* qui permettent de connaître pour chaque perception les valeurs pour lesquelles un *Agent Contexte* est valide. Une *plage de validité* est une plage de valeurs possédant des bornes qui sont évolutives au cours de la vie de l'agent. Une *plage de validité* est valide lorsque la donnée associée se situe entre les bornes de celle-ci. Ces agents disposent aussi d'un indice de confiance qui leur permet de s'auto-juger sur la pertinence de l'action qu'ils proposent lorsqu'ils sont valides.

L'Agent Head

L'*Agent Head* est unique, c'est lui qui communique au *mécanisme d'exploitation* les propositions d'actions faites par les *Agents Contextes* valides. Cet agent reçoit aussi le feedback du *mécanisme d'exploitation* afin de communiquer à l'*Agent Contexte* dont l'action a été choisie si celle-ci était pertinente. L'*Agent Contexte* ainsi sollicité peut alors s'adapter en fonction du résultat de ce feedback.

Lorsque le système s'initialise, le *mécanisme d'adaptation* est seulement composé des *Agents Percepts* et d'un *Agent Head*. Il n'y a aucun *Agent Contexte*, ils sont crées en au cours de l'exécution à chaque fois que le système rencontre des situations qu'il ne connait pas.

1.4.2 Adaptive Multi-Agent Systems for Context Learning (AMAS4CL)

Un des points fort des systèmes multi-agents auto-adaptatifs est qu'ils permettent de doter un système de capacités d'apprentissage et d'auto-adaptation afin de résoudre divers problèmes dans un environnement dynamique. C'est pourquoi le *mécanisme d'adaptation* défini précédemment repose sur un système multi-agent auto-adaptatif et plus particulièrement sur une approche que l'on nomme *Adaptive Multi-Agent Systems for Context Learning* (AMAS4CL). Cette approche est née en réponse à la problématique de l'association d'une action à un état du monde à l'aide d'AMAS.

1.4.2.1 Les interactions

Dans AMAS4CL, il existe plusieurs interactions entre un *Agent Contexte* et son environnement que l'on peut classer en trois catégories :

- **Coopérative** : C'est une interaction utile pour l'environnement et l'agent. Autrement dit, la proposition de l'*Agent Contexte* est pertinente car elle a été sélectionnée par l'*Agent Head* et le résultat produit était celui qui était prévu.
- **Neutre** : Cette interaction n'a pas d'impact sur l'environnement ou l'agent. L'*Agent Head* n'a pas retenu l'action proposée par l'*Agent Contexte* car il l'a jugée non pertinente.
- **Antinomique** : C'est une interaction nuisible pour l'environnement et/ou l'agent. Bien que l'*Agent Head* ait choisi l'action proposée par l'*Agent Contexte*, elle n'a pas donné le résultat attendu. La proposition de l'*Agent Contexte* est donc erronée.

Pour que l'*Agent Head* puisse sélectionner des actions pertinentes, les *Agents Contextes* doivent tous mener à des interactions coopératives. Grâce à un feedback venant de l'*Agent Head*, les *Agents Contextes* reçoivent des informations concernant l'aboutissement de la proposition. Ils peuvent ainsi adapter leur comportement en fonction de la situation rencontrée.

1.4.2.2 Comportement des *Agents Contextes*

On distingue deux types de comportements pour un agent : le comportement nominal et le comportement coopératif [Bonjean et al., 2014]. Pour ces *Agents Contextes*, on décrit ces comportements de la manière suivante :

- **Le comportement nominal** : il s'applique lorsque l'agent est en état de coopération. Ce comportement lui permet d'accomplir son objectif personnel qui est de proposer l'action qui lui est associée lorsqu'il est valide.
- **Le comportement coopératif** : il s'applique lorsque l'agent est dans un autre état que celui de la coopération. Ce comportement permet à l'agent de s'adapter pour être ou revenir dans une état de coopération.

1.4.2.3 Les situations de non coopération

L'apprentissage de contexte par systèmes multi-agents auto-adaptatifs reposant sur l'approche AMAS, les agents peuvent rencontrer des situations de non coopération. [Boes et al., 2015] ont dégagé quatre types de situations de non coopération :

- **Mauvaise appréciation** : lorsqu'un *Agent Contexte* détermine grâce au feedback que son action n'a pas produit les effets escomptés. Dans ce cas, l'*Agent Contexte* réduit ses *plages de validité* afin d'éviter de faire des propositions dans une zone de l'espace des perceptions dans laquelle son appréciation est erronée.
- **Appréciation inexacte** : lorsqu'un *Agent Contexte* détermine grâce au feedback que sa proposition d'action est inexacte. Dans ce cas, l'agent ajuste son appréciation.
- **Inutilité de l'*Agent Contexte*** : suite à plusieurs ajustement de ses *plages de validité*, un *Agent Contexte* peut se retrouver avec des *plages de validité* trop réduites. Il se considère alors comme inutile et s'auto-détruit.
- **Improductivité de l'*Agent Head*** : lorsque l'*Agent Head* ne reçoit pas de proposition d'action ou lorsque le *mécanisme d'exploitation* n'exécute aucune des actions qui lui ont été proposées par l'*Agent Head*. Dans ce cas là, le *mécanisme d'exploitation* a effectué une nouvelle action. L'*Agent Head* demande alors aux *Agents Contextes* d'élargir leur *plages de validité* afin de pouvoir prendre en compte cette nouvelle action. Un nouvel *Agent Contexte* est créé si aucun des agents déjà présents ne peut élargir ses plages.

1.4.3 AMOEBA

Agnostic MOdEl Builder by self-Adaptation (AMOEBA) est un système d'apprentissage développé au sein de l'équipe SMAC qui repose sur l'approche AMAS et l'apprentissage de contextes [Nigon et al., 2016]. Ce système permet de générer des modèles de systèmes complexes en temps réel. C'est un système d'apprentissage agnostique car il est conçu pour s'adapter à des types de données très différentes, aucune hypothèse sur les données n'est faite pour générer les modèles.

Une instance d'AMOEBA peut posséder une ou plusieurs entrées et une seule sortie qui sont des valeurs numériques réelles. On associe souvent ces entrées aux perceptions d'un système. AMOEBA construit un mappage d'un espace de perceptions qui possède autant de dimensions qu'il y a d'entrées. Ce mappage correspond à la portion de l'espace qui sera connue. Et pour chacune des portions de l'espace qui sont mappées, une sortie désirée est générée à l'aide d'un modèle. Pour faire ce mappage et construire les modèles, AMOEBA utilise des exemples concrets de sorties désirées associés à leurs entrées. C'est le même principe que l'apprentissage supervisé (section 1.1.1). La source, appelé l'oracle, fournit les sorties désirées. Elle peut être activée ou désactivée pendant une exécution. Lorsque que l'oracle est activé, AMOEBA construit un modèle du comportement de cet oracle en utilisant ses perceptions, il est en mode adaptation.

Ces perceptions peuvent être des valeurs de capteurs, des modèles mathématiques, une base de données... Quand l'oracle est désactivé, AMOBA est en mode exploitation et n'apprend pas, il exploite ce qu'il a appris pour sélectionner la bonne sortie en fonction des perceptions qu'il a en entrée.

1.4.3.1 Les agents

AMOEBA étant basé sur l'approche AMAS et l'apprentissage de contextes, il est composé d'*Agents Percepts*, d'*Agents Contextes* et d'un *Agent Head*.

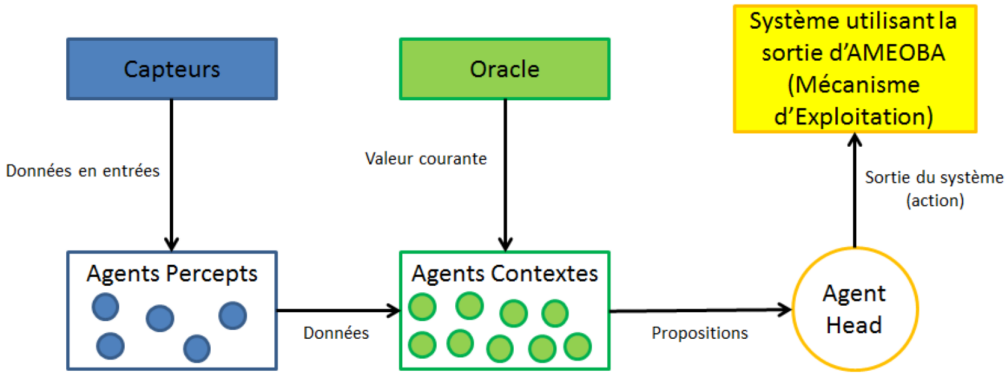


FIGURE 1.4 – Schéma de fonctionnement d'AMOEBA

Comme tout système multi-agent auto-adaptatif, les agents possèdent un comportement nominal que l'on modélise par un cycle *Perception-Décision-Action* et des comportements de coopération afin de maintenir ou atteindre l'état coopératif du système.

1.4.3.2 Les comportements nominaux des agents

Agent Percept. Pendant sa phase de perception, il récupère la valeur de la source de donnée à laquelle il est associé. Puis, pendant la phase de décision et action, il transmet cette valeur aux *Agents Contextes*.

Agent Contexte. Il commence son cycle en recevant les données des *Agents Percepts*. En fonction de ces données, il détermine s'il est valide ou non. Pour être valide (cf section 1.4.1.2), toutes ses *plages de validité* doivent être valides. Lorsqu'un *Agent Contexte* est valide, il envoie sa proposition et son indice de confiance à l'*Agent Head*.

Agent Head. Il commence son cycle en recevant les propositions de tous les *Agents Contextes* valides. Pendant la phase de décision, l'*Agent Head* choisit la proposition possédant l'indice de confiance le plus élevé. Puis, pendant la phase d'action, il envoie l'action ou sortie sélectionnée au système utilisant les sorties d'AMOEBA.

1.4.3.3 Les situations de non coopération

AMOEBA tire sa capacité d'auto-adaptation de la résolution des situations de non coopération que peuvent rencontrer les agents qui le composent. AMOEBA compte quatre cas de situations de non coopération :

- **Conflit d'un *Agent Contexte* - Mauvaise proposition** :
  Détection : Grâce à la valeur fournie par l'oracle, une *Agent Contexte* peut évaluer sa proposition. Si la différence entre sa proposition et la valeur de l'oracle est supérieure à une marge d'erreur (définie par l'utilisateur), l'*Agent Contexte* considère qu'il a donné

une mauvaise information à l'*Agent Head*.
Résolution : L'agent détermine qu'il n'aurait pas dû être valide et il réduit une *plage de validité* afin d'exclure la situation courante et les situations semblables pour ne plus fournir de propositions erronées. La *plage de validité* à réduire est choisie en minimisant la réduction de la taille de l'agent.

- **Conflit d'un *Agent Contexte* - Proposition inexacte** :
  Détection : Toujours grâce à la valeur fournie par l'oracle, un *Agent Contexte* peut évaluer si la proposition est inférieure à la marge d'erreur mais supérieure à la marge d'inexactitude (également définie par l'utilisateur). L'agent considère alors qu'il a donné une information inexacte à l'*Agent Head*.
  Résolution : Pour cette situation considérée comme moins grave que la précédente. L'agent adapte seulement sa proposition, c'est-à-dire le modèle qui lui permet de la calculer, afin d'être plus précis. Il diminue aussi son indice de confiance.
- **Inutilité d'un *Agent Contexte*** :
  Détection : Après avoir réduit plusieurs fois ses *plages de validité*, un *Agent Contexte* peut se retrouver avec certaines de ses plages trop réduites. Lorsque l'une d'entre elles est inférieure à une seuil critique (défini par le concepteur), l'*Agent Contexte* estime qu'il est inutile.
  Résolution : Dans cette situation, l'*Agent Contexte* s'auto-détruit car il considère qu'il ne pourra plus être valide.
- **Improductivité de l'*Agent Head*** :
  Détection : Lorsque l'*Agent Head* ne reçoit aucune proposition des *Agents Contexte*, il n'a alors aucune proposition à sélectionner. Cette situation arrive quand aucun des *Agents Contextes* n'est valide, c'est-à-dire que le système se trouve dans une situation qu'il n'a jamais rencontrée auparavant.
  Résolution : Dans le cas où la dernière proposition sélectionnée a été envoyée par le dernier *Agent Contexte* créé et que celui-ci a toujours fait de bonnes propositions, alors l'*Agent Head* demande à cet agent d'étendre ses *plages de validité* vers le contexte courant. Dans le cas contraire, l'*Agent Head* crée une nouvel *Agent Contexte* en initialisant ses *plages de validité* grâce aux valeurs du contexte courant, et sa proposition à l'aide de la valeur courante de l'oracle.

1.4.4 Analyse

Les méthodes d'apprentissage à l'aide de l'approche AMAS sont très prometteuses grâce à leurs capacités d'adaptation en temps réel. En effet, le système d'apprentissage AMOEBA met en place un apprentissage par démonstrations en ligne. Il est possible grâce à ses capacité d'auto-adaptation de constamment apporter de nouvelles connaissances étiquetés par un oracle sans avoir à recommencer la démonstration depuis le début.

1.5 Synthèse

Parmi les deux grandes familles d'apprentissage artificiel que sont l'apprentissage supervisé et l'apprentissage par renforcement, chacune d'entre elles possèdent des avantages différents. L'apprentissage supervisé permet de guider l'exploration de l'espace de recherche et l'apprentissage par renforcement permet de faire une exploration plus globale et en ligne.

Appliqués à la robotique, ces avantages se retrouvent respectivement pour les techniques d'apprentissage par démonstrations et d'apprentissage constructiviste. Pour notre étude, nous ne mettrons pas en place d'apprentissage constructiviste mais nous utiliserons comme base

un apprentissage par démonstrations qui sera effectué à l'aide du mécanisme d'apprentissage AMOEBA.

Nous proposerons alors un mécanisme complémentaire à AMOEBA qui reposera sur l'approche des systèmes multi-agents auto-adaptatifs. Ce mécanisme permettra de doter notre système robotique de capacités d'auto-observation lui permettant d'affiner l'espace de recherche découvert par la démonstration.

Chapitre 2

AMALOM - Modèle de contrôle par système multi-agent auto-adaptatif

Dans ce chapitre, nous présentons le fonctionnement du cas d'application qui est le simulateur de vol d'un drone. Ensuite, nous détaillons l'architecture d'AMALOM (Adaptive Multi-Agent Linear cOntrol Model). Cette architecture comprend un apprentissage par démonstrations sur lequel vient s'ajouter l'apprentissage par feedback endogène conçu pendant le stage.

2.1 Simulateur de vol de drone

Le cas d'application de cette étude est une simulation de vol de drone mise en place dans le cadre de cette étude à l'aide du moteur de jeu vidéo Unity. Unity est un moteur de initialement de jeu mais aussi de simulation multi-plateforme. Ce moteur propose un licence gratuite sans limitation au niveau du moteur. Il a déjà été utilisé dans le cadre de travaux précédents [d'Amico, 2016].

Le drone peut être commandé manuellement en translation et en rotation. Il peut aussi être piloté de manière automatique à l'aide de commandes lui permettant de s'asservir en position sur des zones d'intérêt. Soit $X(t) = (x(t), y(t), z(t), \psi(t))^T$ le vecteur d'état du drone, comportant les position $(x(t), y(t), z(t))$ et le lacet $\psi(t)$, est modifié à l'aide du vecteur de commande $C(t) = (C_1(t), C_2(t), C_3(t), C_4(t))^T$. Chacune des trois premières commandes permet d'exercer une force sur le drone dans les directions respectives x, y et z. C'est alors le moteur physique de Unity qui calcule les positions et vitesses du drone. La commande $C_4(t)$ permet d'incrémenter ou de décrémenter l'angle $\psi(t)$. Toutes ces commandes sont bornées entre -1 et 1.

Les zones d'intérêt sont choisies manuellement ou de façon aléatoire à l'intérieur d'une sphère. Il est aussi possible d'ajouter des "courants d'air" afin de venir perturber le vol du drone. La simulation dispose d'une interface utilisateur afin de pouvoir interagir avec les mécanismes d'apprentissage utilisés et la simulation elle même (figure 2.1).

2.2 Une architecture pour des systèmes auto-apprenants

La solution mise en place au cours de cette étude est constituée d'un apprentissage par démonstrations et d'un apprentissage par feedback endogène (figure 2.2).

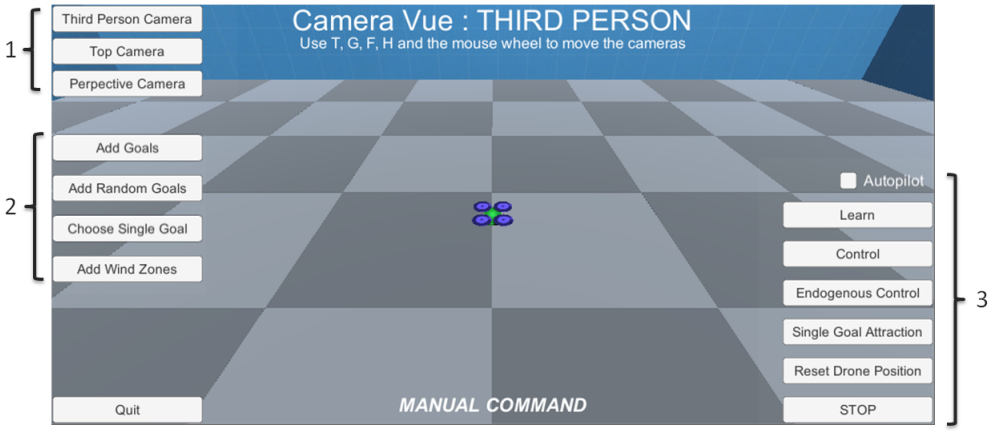


FIGURE 2.1 – Capture de la simulation de vol de drone sous le moteur de jeu Unity - (1) Options caméras ; (2) Ajout de zones d'intérêt ou zones de perturbations ; (3) Options d'apprentissage et de commande du drone

L'apprentissage par démonstrations permet de construire un premier jeu de modèles à partir des perceptions de notre système robotique qui est le drone et à partir des commandes qui sont effectués pendant la démonstration. Le drone apprendra ainsi dans notre cas, une trajectoire. Nous verrons par le suite que cet apprentissage lui permet déjà suivre la trajectoire qu'il a apprise. Cependant, s'il vient à s'écarter de celle-ci à cause d'une perturbation comme un coup de vent par exemple, il sera amené dans une situation de l'espace qu'il ne connait pas.

L'apprentissage par feedback endogène permet d'apprendre un modèle de contrôle du drone à partir de l'exploitation des perceptions et des actions expérimentées par le drone. À partir de ce modèle, il est alors possible d'évoluer dans l'espace des perceptions à l'aide des actions donc dispose le système robotique. Ainsi, pour éviter que le drone ne s'écarte de l'espace des perceptions qu'il connait, il est asservi sur ces espaces à l'aide du modèle de contrôle.

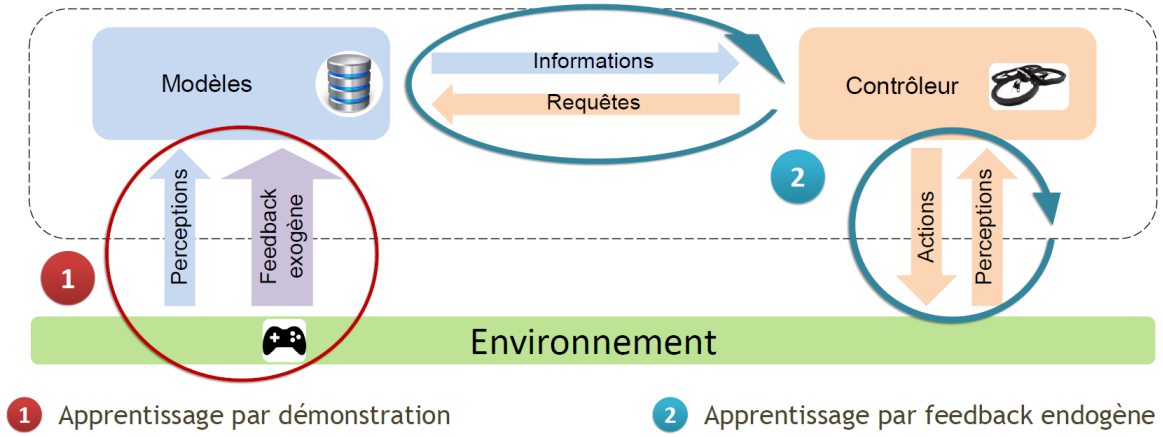


FIGURE 2.2 – Schéma de l'architecture d'AMALOM, mécanisme couplant apprentissage par démonstrations et apprentissage par feedback endogène

2.2.1 Apprentissage par démonstrations

Comme nous l'avons dit précédemment, dans le cadre de notre étude appliquée à la simulation de vol d'un drone, nous avons mis en place un apprentissage par démonstrations de

trajectoires. Cette démonstration peut être faite manuellement mais cela dévient vite fastidieux c'est pourquoi nous avons choisi de générer et d'asservir le drone sur trajectoire de manière automatique.

2.2.1.1 Génération et suivi de trajectoires

La génération des trajectoires est faite par la définition de zones d'intérêt par lesquelles le drone doit passer. On peut imaginer plusieurs zones qu'un drone doit visiter pour prendre des mesures ou bien surveiller des zones à risque. Ensuite, la commande du drone pour le besoin de la démonstration a été faite grâce à un asservissement des commandes sur les positions de ces zones d'intérêt. Ainsi, le drone suit un cycle pendant lequel il s'asservit successivement sur chacune de zones à visiter.

2.2.1.2 Apprentissage des commandes par démonstrations

Pour apprendre à notre système les trajectoires ainsi suivies par le drone pendant la démonstration, nous utilisons trois instances d'AMOEBA. Comme nous l'avons vu section 1.4.3, une instance d'AMOEBA permet de générer des modèles qui à partir de plusieurs entrées fournissent une sortie. Ces sorties sont les données labellisées fournies pendant la démonstration. Notre étude se penche sur un problème de commande, les entrées sont donc certaines des perceptions de notre système robotique (position, vitesse, accélération ...) et les sorties certaines de ses actions. On se place ici dans un cas simplifié où le drone dispose de sa position (x, y, z) absolue dans son environnement et qu'il est commandable suivant ces trois axes.

Chacune des trois instances d'AMOEBA possède alors en entrée les positions absolues (P_x, P_y, P_z) du drone que l'on notera dorénavant (P_0, P_1, P_2). En sortie, elles possèdent chacune une des trois commandes suivant les axes (X, Y, Z) que l'on notera (C_0, C_1, C_2). Durant la démonstration, chaque instance apprend une des trois commandes en fonction des positions (P_0, P_1, P_2) du drone.

Afin d'exploiter cet apprentissage, il faut alors envoyer des requêtes avec les positions courantes du drone vers chacune des instances d'AMOEBA qui retournent les commandes qu'elles pensent pertinentes à effectuer en fonction du contexte dans lequel se trouve le drone. Lorsque le drone ne se situe pas dans un contexte, AMOEBA calcule la sortie à partir du contexte le plus proche. Ce calcul est fait à l'aide d'une distance euclidienne.

2.2.2 Apprentissage par feedback endogène

Afin de forcer le drone à rester dans un espace connu, nous avons mis en place un apprentissage par feedback endogène permettant d'apprendre un modèle de contrôle du drone.

2.2.2.1 Définition du modèle

Afin d'apprendre un modèle de contrôle sans *a priori*, on utilise les interactions qu'a le système avec son environnement, c'est-à-dire les actions ou commandes $C_{i,t}$ qu'il peut effectuer (valeurs réelles des commandes) et les perception $P_{j,t}$ qu'il a de son environnement (valeurs réelles de ses capteurs) avec t les cycles de temps discrets des instants considérés. Afin d'apprendre au système comment évoluer dans son environnement avec la seule connaissance de ces informations, il est nécessaire d'apprendre un modèle de contrôle permettant d'associer à une ou plusieurs variations de perception une ou plusieurs actions pour chaque instant t. Ceci

aura pour but de donner au système la capacité d'agir sur ses perceptions car il connaîtra les conséquences de ses actions.

On suppose que chaque valeur de commande $C_{i,t}$ du système peut être calculée pour chaque situation à l'aide d'un modèle linéaire dépendant de ses n variations de perception $\delta P_{j,t}$ souhaitées tel que :

$$C_{i,t} = \sum_{j=1}^{n} \omega_{ij} \delta P_{j,t} \tag{2.1}$$

avec ω_{ij} des poids associés à chaque variation de perception et $\delta P_{j,t} = P^*_{j,t} - P_{j,t}$ l'écart entre la position souhaitée au temps suivant et la position courante.

Pour notre cas où il y a 3 perceptions et 3 actions, on considère le modèle de contrôle linéaire suivant :

$$\begin{pmatrix} C_{1,t} \\ C_{2,t} \\ C_{3,t} \end{pmatrix} = \begin{pmatrix} \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{21} & \omega_{22} & \omega_{23} \\ \omega_{31} & \omega_{32} & \omega_{33} \end{pmatrix} . \begin{pmatrix} \delta P_{1,t} \\ \delta P_{2,t} \\ \delta P_{3,t} \end{pmatrix} = \begin{matrix} \omega_{11}\delta P_{1,t} + \omega_{12}\delta P_{2,t} + \omega_{13}\delta P_{3,t} \\ \omega_{21}\delta P_{1,t} + \omega_{22}\delta P_{2,t} + \omega_{23}\delta P_{3,t} \\ \omega_{31}\delta P_{1,t} + \omega_{32}\delta P_{2,t} + \omega_{33}\delta P_{3,t} \end{matrix} \tag{2.2}$$

Afin de pouvoir contrôler notre système, il faut connaître les valeurs des poids ω_{ij}. On se place dans le cas simplifié que pour une commande, un seul des poids est différent de 0. Pour ajuster ces valeurs, nous allons introduire deux nouvelles grandeurs, la criticité et la sensibilité.

La criticité représente la différence entre une commande effectuée par le système robotique et la valeur qu'aurait fourni le modèle pour effectuer ce même déplacement :

$$\begin{aligned} \mathcal{C}_{C_{i,t}} &= |C_{i,t_{reel}} - C_{i,t_{modele}}| \\ &= |C_{i,t_{reel}} - \textstyle\sum_{j=1}^{n} \omega_{ij} \delta P_{j,t}| \end{aligned} \tag{2.3}$$

On définit maintenant la sensibilité s_{ij} avec laquelle une perception varie par rapport à une certaine commande :

	$\delta P_{1,t}$	$\delta P_{2,t}$	$\delta P_{3,t}$
$C_{1,t}$	s_{11}	s_{12}	s_{13}
$C_{2,t}$	s_{21}	s_{22}	s_{23}
$C_{3,t}$	s_{31}	s_{32}	s_{33}

On calcule la sensibilité de la façon suivante :

$$\begin{aligned} s_{ij} &= \frac{\sum_t \beta_{i,t} C_{i,t} \delta P_{j,t}}{\sum_{i,j,t} \beta_{i,t} C_{i,t} \delta P_{j,t}} \\ \text{avec} \quad \beta_{i,t} &= \begin{cases} \frac{|C_{i,t}|}{\sum_k |C_{k,t}|} & \text{si } \sum_k |C_{k,t}| \neq 0 \\ 0 & \text{sinon} \end{cases} \end{aligned} \tag{2.4}$$

On suppose ici que chaque commande $C_{i,t}$ a un impact sur une seule des variations $\delta P_{j,t}$. Chaque produit $C_{i,t}\delta P_{j,t}$ permet de savoir à un instant t si la commande $C_{i,t}$ est corrélée à la variation de perception $\delta P_{j,t}$. Plus ce produit sera grand, plus la variation sera sensible à cette commande. Cependant, il est possible que plusieurs commandes soient effectuées au même instant entrainant ainsi plusieurs variations. Le coefficient de pondération $\beta_{i,t}$ permet de donner plus d'importance au produit $C_{i,t}\delta P_{j,t}$ lorsque la commande considérée est grande devant les

autres commandes actives. Le terme $\sum_{i,j,t} \beta_{i,t} C_{i,t} \delta P_{j,t}$ permet de normaliser toutes les valeurs de sensibilité qui seront mises à jour à chaque instant. Pour accorder plus de réactivité et de poids aux expériences récentes du système, les sensibilités seront calculées à partir d'une quantité finie d'instants t.

2.2.2.2 Les agents

Du point de vue système multi-agent, nous avons agentifié notre problème à l'aide des agents suivants :

- **Les agents *modèle***. Ces agent sont chacun associés à une des commandes du drone, il possèdent une criticité définie à l'équation 2.3. Ces agents permettent de construire la partie du modèle leur correspondant. Un tel agent questionne l'agent *sensibilité* lui correspondant afin de savoir quelle perception est la plus sensible à la commande de son modèle. À partir de cette information, il ajuste le poids du modèle correspondant à cette perception de manière à minimiser sa criticité. Si la perception la plus sensible venait à changer, l'agent sauvegarde la valeur courante du poids non null avant de le mettre à zéro et d'ajuster le poids correspondant à la nouvelle perception la plus sensible. Ainsi, si la perception la plus sensible redevient une de celles enregistrées, l'agent n'a pas à recommencer son ajustement depuis le début.
- **Les agents *sensibilité***. Ces agents sont associés à une des commandes du drone. Chacun possède autant de sensibilités qu'il y a de perceptions (équation 2.4). Le rôle de ces agents et de trier continuellement ces sensibilités afin de savoir laquelle est la plus forte pour donner cette information à l'agent modèle correspondant. Ces sensibilités sont calculées à partir d'une moyenne glissante dont le nombre de valeurs peut être choisi par l'utilisateur.

2.2.2.3 Utilisation du modèle

Le modèle ainsi généré est prévu pour venir compléter l'exploitation de l'apprentissage par démonstrations effectué à l'aide des instances d'AMOEBA. Nous avons vu précédemment que pendant l'exploitation de cet apprentissage, il est possible que le drone sorte des zones connues de l'espace par les instances AMOEBA qui sont les contextes qui auront été créés pendant l'apprentissage. Pour éviter cela, nous utilisons le modèle de contrôle afin d'asservir le drone sur les positions de l'espace qu'il connait. Dans le cas où il s'écarte de ces zones, il y est alors ramené directement vers elles grâce à cet asservissement sans que l'apprentissage par démonstrations ne lui ait appris à le faire. Cela rappelle le suivi de trajectoire en robotique.

Ainsi, lorsqu'on s'écarte des contextes les plus proches, le modèle de contrôle nous donne le vecteur de commande C_{ASS} permettant d'asservir le drone sur la moyenne des centres des contextes les plus proches. En effet, comme nous avons plusieurs instances d'AMOEBA, il y a plusieurs contextes à prendre en compte. Cependant, si le drone a déjà dépassé ces contextes, on ne peut prendre en compte entièrement cette commande car elle empêche d'effectuer la commande C_{AMOEBA} apprise par démonstration qui est le vecteur de commande fourni en regroupant toutes les instances d'AMOEBA. On effectue alors une projection du vecteur C_{ASS} sur le vecteur C_{AMOEBA} afin de ne garder que la partie orthogonale au vecteur C_{AMOEBA} que l'on nomme $C_{\perp}$ (voir figure 2.3). On a alors le vecteur de commande final C_{FINAL} tel que :

$$
\begin{aligned}
C_{FINAL} &= C_{AMOEBA} + C_{\perp} \\
&= C_{AMOEBA} + C_{ASS} - P_{C_{AMOEBA}}(C_{ASS}) \qquad (2.5) \\
\text{avec} \quad & P_{C_{AMOEBA}}(C_{ASS}) = \frac{C_{ASS}.C_{AMOEBA}}{\|C_{AMOEBA}\|^2}.C_{AMOEBA}
\end{aligned}
$$

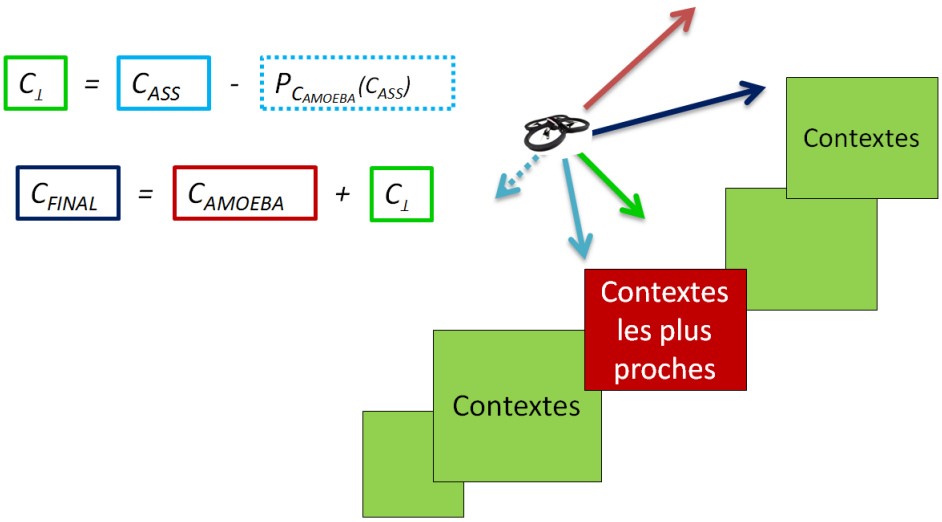


FIGURE 2.3 – Schéma des vecteurs de commandes pour le suivi des contextes, pour des raisons de visualisation, les contextes sont représentés à deux dimensions et on considère que les contextes des trois instances d'AMOEBA sont identiques

2.3 Synthèse

Pour répondre au problème de l'apprentissage par feedback endogène, nous avons tout d'abord conçu une simulation de vol de drone à l'aide du moteur de jeu Unity afin de disposer d'un cas concret d'étude.

Ensuite, nous avons mis en place un apprentissage par démonstrations à l'aide du mécanisme d'apprentissage AMOEBA qui est basé sur l'approche de l'apprentissage de contextes par agents auto-adaptatifs. Cet apprentissage permet de guider l'exploration de l'espace dans lequel évolue le système robotique qu'est le drone.

Il se traduit dans notre cas par l'apprentissage des commandes permettant au drone de suivre une trajectoire. Cette trajectoire est définie à partir de zones d'intérêt sur lesquelles le drone s'asservit à l'aide d'un pilote automatique conçu pour les besoins de la démonstration. Les commandes nécessaires au pilotage du drone sont alors fournies par ce pilote automatique qui joue le rôle d'oracle. Cependant, lors de l'exploitation de apprentissage, le drone peut se trouver dans une zone de l'espace qu'il ne connait pas.

Pour remédier à ce problème, en addition à cet apprentissage par démonstrations, un mécanisme d'apprentissage endogène nous permet d'apprendre un modèle de contrôle du drone afin d'apprendre comment le contrôler en exploitant seulement ses actions et perceptions. Ce modèle permet alors d'asservir le drone sur les zones de l'espace qu'il connait et éviter ainsi les zones pour lesquelles la démonstration initiale n'est pas valable.

Chapitre 3

Analyse des résultats

Dans ce chapitre, nous commençons par présenter les résultat obtenus à partir d'apprentissage par démonstrations seul. Ensuite, nous présentons en quoi l'apprentissage endogène permet d'apporter une amélioration à l'apprentissage par démonstrations pour notre cas d'application.

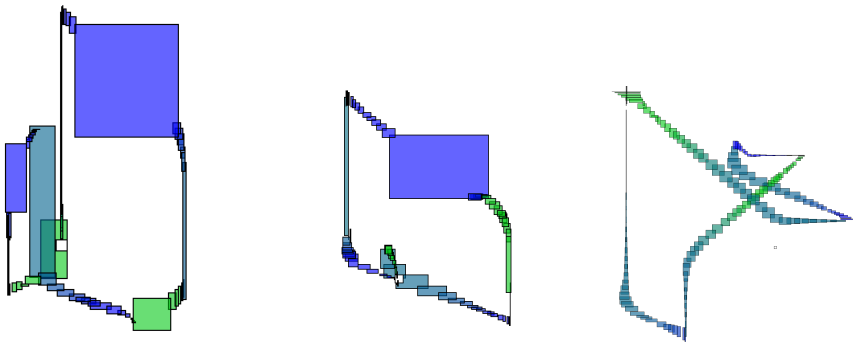
3.1 Apprentissage par démonstrations

Comme nous l'avons dit précédemment, grâce à la génération de zones d'intérêt et à l'asservissement sur celles-ci, nous avons mis en place un apprentissage par démonstrations. Pour l'ensemble des résultats que nous allons présenter, il y aura pour chaque trajectoire, cinq zones générées aléatoirement qui seront visitées par le drone.

3.1.1 Étude de sensibilité pour AMOEBA

Les instances d'AMOEBA possèdent plusieurs paramètres qu'il est possible de changer, dans le cadre de cette étude, nous nous somme concentrés sur un des ces paramètres qui est la taille initiale des contextes qui sont créés lors de l'apprentissage. De gros contextes vont généraliser les connaissances ce qui diminuera le nombre de contextes tandis que de petits contextes généraliseront peu. Ce qui entrainement un grande quantité de contextes.

On peut voir figure 3.1c que plus les contextes sont petits, plus la trajectoire se dessine sur l'espace de recherche mais il y a très peu de généralisation. En effet, figure 3.1a, les contextes ont tendance à plus recouvrir l'espace mais la trajectoire est moins bien visible.

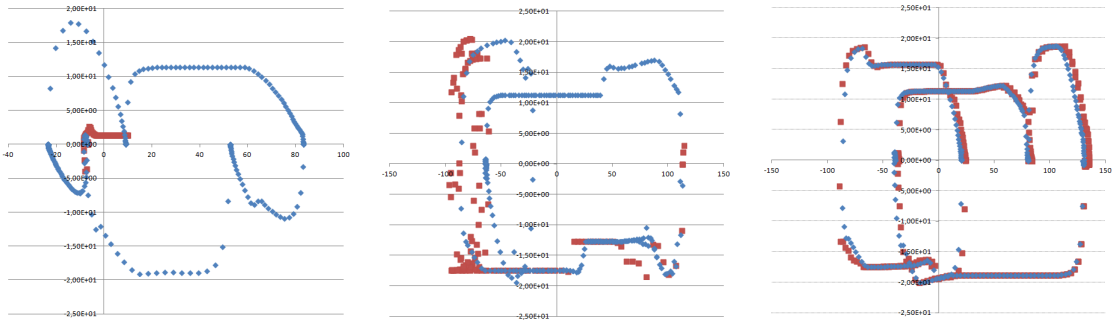


(a) (b) (c)

FIGURE 3.1 – Coupe à deux dimensions des contextes pour une des commandes du drone pour plusieurs paramètres de taille initiale des contextes

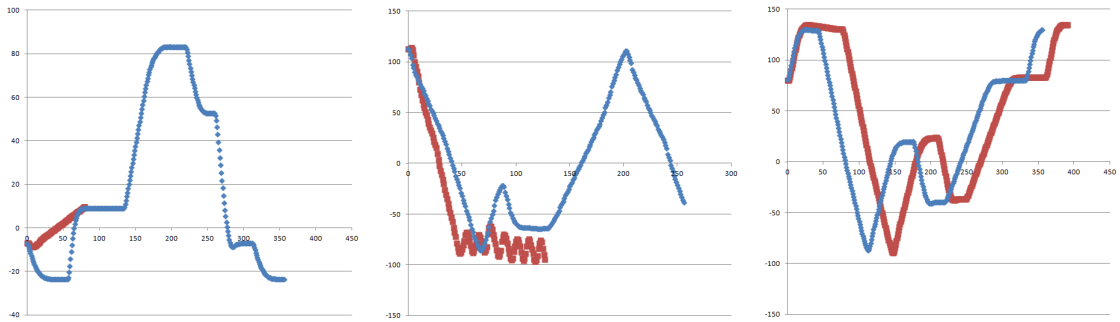
Afin de représenter visuellement l'écart entre les trajectoires faites par le démonstrateur et celles faites par le mécanisme d'apprentissage AMOEBA, nous utilisons les portraits de phase. Les portrait de phase permettent de tracer une variable x en fonction de sa dérivée $\dot{x}$ tel que $f(x) = \dot{x}$. Dans notre cas, étant donné que l'on considère les positions parcourues par le drone, nous tracerons les vitesses en fonction des position pour les trois axes X, Y et Z. Ceci nous permet de nous abstraire de la variable du temps et de nous concentrer sur les formes des trajectoires. Dans notre cas, on trace alors la trajectoire initiale faite grâce à l'asservissement puis celle obtenue grâce à l'apprentissage par démonstrations. On compare ici l'efficacité de l'apprentissage en fonction de la tailles des contextes. On trace seulement une des trois perceptions. Ainsi, on remarque figure 3.2 que plus les contextes sont petits, plus le portrait de phase de la trajectoire faite grâce à l'exploitation de l'apprentissage par démonstrations est proche de la trajectoire initiale. En traçant une des perceptions en fonction du temps pour les deux trajectoires (figure 3.3), on peut voir que la trajectoire apprise décroche bien plus tôt lorsque les contextes sont de grande taille.

Pour notre problème, on préfère alors utiliser de petits contextes en plus grande quantité car il permettent de mieux suivre la trajectoire initialement montrée.



(a) AVT 0.5 (b) AVT 0.25 (c) AVT 0.1

FIGURE 3.2 – Portraits de phase des trajectoires effectuées par l'asservissement/démonstrateur (bleu) et par AMOEBA (rouge) pour une des trois dimensions de l'espace des perceptions, pour plusieurs paramètres de taille initiale des contextes



(a) AVT 0.5 (b) AVT 0.25 (c) AVT 0.1

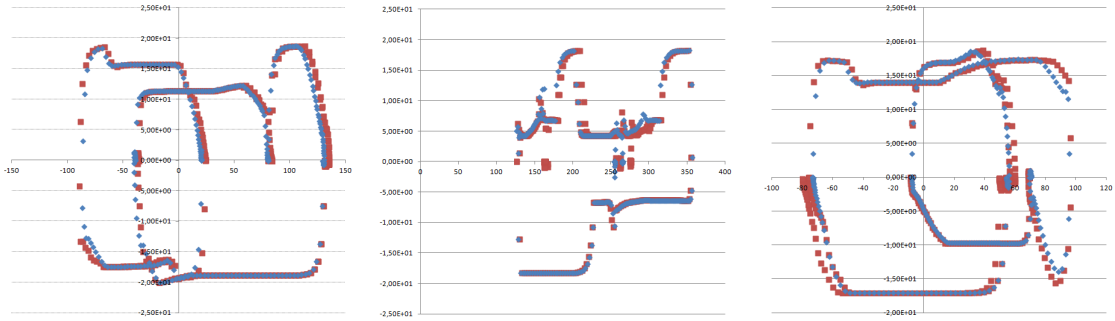
FIGURE 3.3 – Position selon un seul axe du drone en fonction du temps de trajectoires faites par l'asservissement/démonstrateur (bleu) et par AMOEBA (rouge), pour plusieurs paramètres de taille initiale des contextes

3.1.2 Asservissement vs AMOEBA

Maintenant que nous avons fixé le paramètre concernant la taille initiale des contextes, nous allons comparer l'exploitation de l'apprentissage par démonstrations avec la trajectoire utilisée pour la démonstration. Dans un premier temps, nous nous intéresserons au cas sans perturbations. Ensuite, nous introduirons des perturbations pour voir comment réagit AMOEBA.

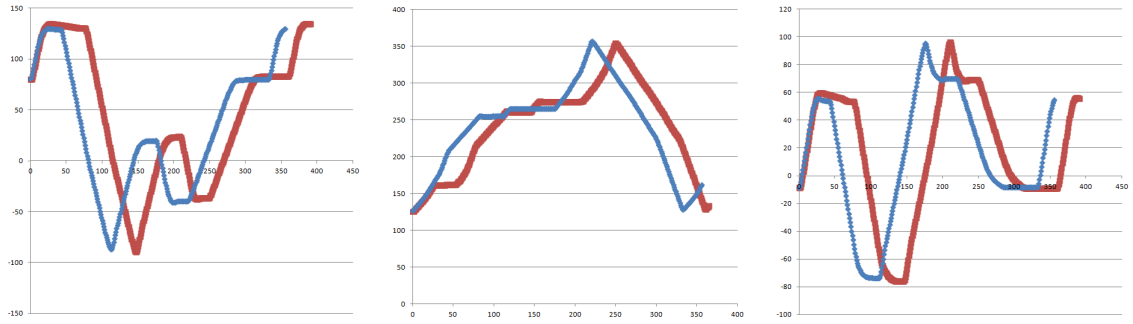
3.1.2.1 Exploitation sans perturbations

Afin de comparer les deux trajectoires, on utilise à nouveaux les portraits de phase de celles-ci ainsi que l'évolution des perceptions en fonction du temps. On remarque alors figure 3.4 que les portraits de phase sont relativement semblables. Cependant, figure 3.5, on remarque un certain retard entre la trajectoire initiale et celle d'AMOEBA. Ce retard peut s'expliquer par une certaine latence dans la communication entre le mécanisme possédant les connaissances et le contrôleur.



(a) P0 (b) P1 (c) P2

FIGURE 3.4 – Portraits de phase d'une trajectoire effectuée par l'asservissement/démonstrateur (bleu) et par AMOEBA (rouge) pour les trois dimensions de l'espace des perceptions, sans perturbations



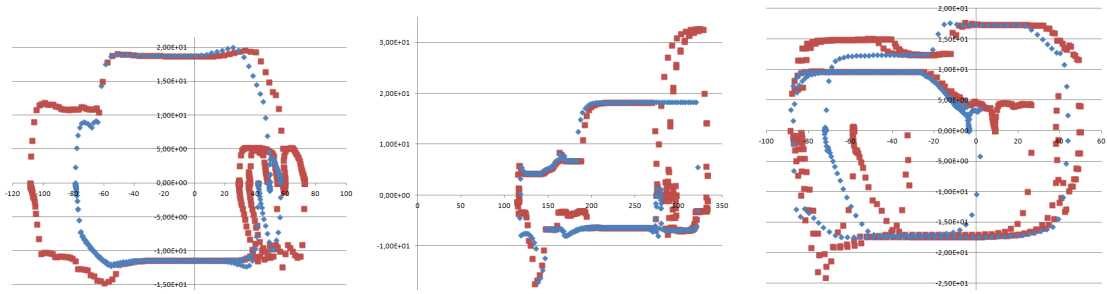
(a) P0 (b) P1 (c) P2

FIGURE 3.5 – Positions selon toutes les perceptions du drone en fonction du temps d'une trajectoire faite par l'asservissement/démonstrateur (bleu) et par AMOEBA (rouge), sans perturbations

3.1.2.2 Exploitation avec perturbations

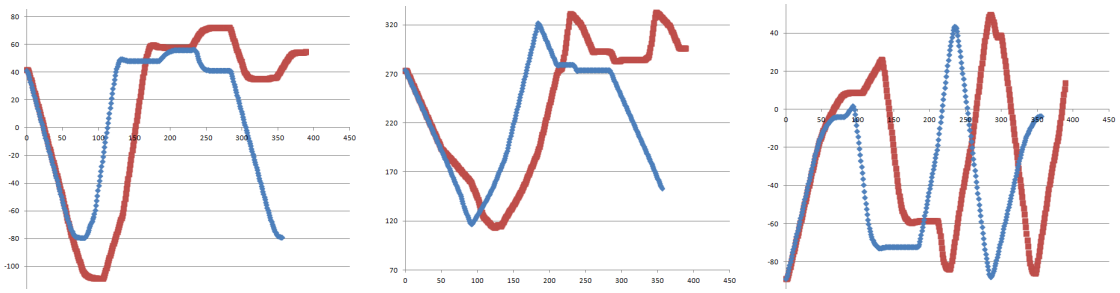
On introduit maintenant des perturbations. On peut alors voir sur la figure 3.6 que les portraits de phases sont moins semblables que lorsqu'il n'y avait pas de perturbations. En regardant

la figure 3.7, on se rend compte que l'exploitation de l'apprentissage par démonstrations n'arrive plus à reproduire des perceptions semblables à la trajectoire initiale.



(a) P0 (b) P1 (c) P2

FIGURE 3.6 – Portraits de phase d'une trajectoire effectuée par l'asservissement/démonstrateur (bleu) et par AMOEBA (rouge) pour les trois dimensions de l'espace des perceptions, avec perturbations



(a) P0 (b) P1 (c) P2

FIGURE 3.7 – Positions selon toutes les perceptions du drone en fonction du temps d'une trajectoire faite par l'asservissement/démonstrateur (bleu) et par AMOEBA (rouge), avec perturbations

Le mécanisme AMOEBA permet de bien reproduire une trajectoire qu'il a apprise mais lorsque de nouvelle situations apparaissent à cause des perturbation, la reproduction de la trajectoire est dégradée.

3.2 Apprentissage d'un modèle de contrôle linéaire

Puisque que qu'AMOEBA ne peut pas gérer l'imprévu, il faut introduire un modèle de contrôle permettant d'apprendre de nouveaux comportements pour les situations inconnues. Nous commencerons ici par tester l'apprentissage d'un modèle de contrôle par feeback endogène sur un modèle connu afin de valider notre mécanisme. Ensuite, nous comparerons une exploration manuelle de l'espace à une exploration aléatoire pour apprendre cette fois-ci le modèle de contrôle du drone. Enfin, nous verrons comment l'apprentissage endogène permet d'améliorer les résultats obtenus à la section 3.1.2.

3.2.1 Apprentissage de modèles connus

Nous avons testé le mécanisme d'apprentissage par feedback endogène sur plusieurs modèles connus afin de le valider. Nous nous sommes basés sur plusieurs modèles de tailles différentes

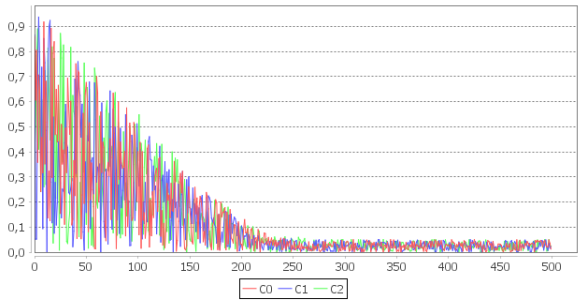
respectant les hypothèses énoncées à la section 2.2.2.1. Un exemple d'un tel modèle est le suivant :

$$\begin{pmatrix} \delta P_0 \\ \delta P_1 \\ \delta P_2 \end{pmatrix} = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{pmatrix} . \begin{pmatrix} C_0 \\ C_1 \\ C_2 \end{pmatrix} \Leftrightarrow \begin{pmatrix} C_0 \\ C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{pmatrix} . \begin{pmatrix} \delta P_0 \\ \delta P_1 \\ \delta P_2 \end{pmatrix} \tag{3.1}$$

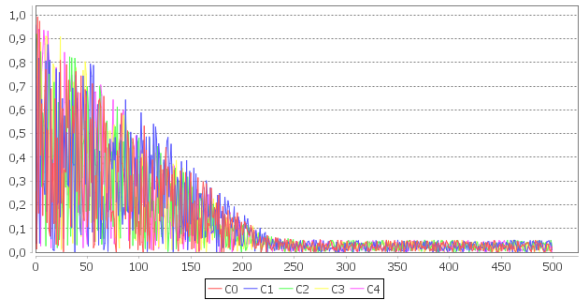
Avec un seuil de criticité autorisée à 0.05, le mécanisme d'apprentissage trouve, après 300 cycles d'apprentissage, le modèle suivant :

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} 9.5 & 0 & 0 \\ 0 & 9.5 & 0 \\ 0 & 0 & 9.5 \end{pmatrix} . \begin{pmatrix} \delta P_0 \\ \delta P_1 \\ \delta P_2 \end{pmatrix} \tag{3.2}$$

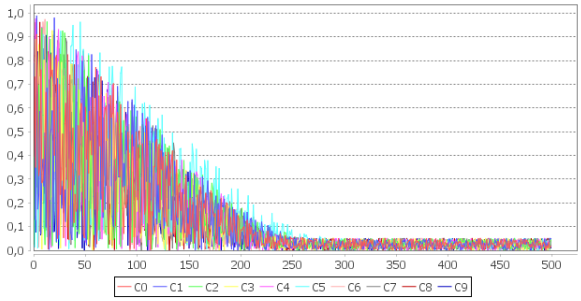
On peut observer que pour des tailles de modèles allant jusqu'à 10x10, le mécanisme d'apprentissage parvient à minimiser sa criticité au seuil demandé toujours en un nombre de cycles d'apprentissage d'environs 300 (figure 3.8).



(a) 3x3



(b) 5x5



(c) 10x10

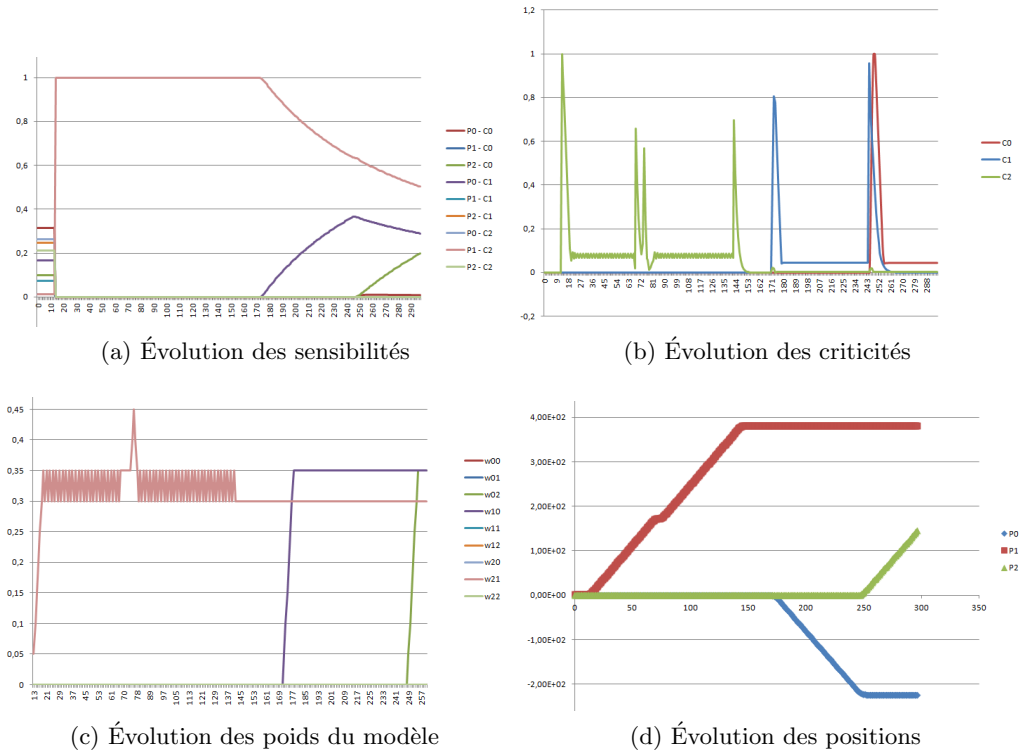
FIGURE 3.8 – Criticités sur des modèles de tailles différents obtenus par le mécanisme d'apprentissage endogène à partir d'explorations aléatoires sur des modèles connus

Pour des modèles connus respectant les hypothèses énoncées à la section 2.2.2.1, le mécanisme d'apprentissage par feedback endogène permet de générer un modèle fidèle.

3.2.2 Apprentissage du modèle de contrôle du drone

Après avoir validé la génération de modèles à partir de modèles connus, on s'intéresse maintenant au mécanisme qui permet de classer les sensibilités afin de générer modèle de contrôle du drone dans la simulation. Nous présentons ici deux types d'exploration, une exploration manuelle et une exploration aléatoire.

Lors de l'exploration manuelle, il est possible d'accélérer le processus de classement des sensibilités en demandant au drone de n'évoluer que dans une seule direction à la fois. Ainsi, figure 3.9a, on peut observer qu'au début de l'exploration une seule perception est sensible à une seule commande car c'est la seule que le système a expérimenté à cet instant. Cette expérience permet alors au poids w_{21} de se stabiliser entre les valeurs 0.3 et 0.35 (figure 3.9c). On remarque aussi que la criticité correspondant au modèle de la commande C_2 se stabilise en même temps que w_{21} (figure 3.9b). Ainsi, en commandant au drone de se déplacer seulement dans une direction à la fois (figure 3.9d), il est possible de construire itérativement le modèle de contrôle sans que d'autres informations superflues viennent biaiser l'apprentissage.



(a) Évolution des sensibilités

(b) Évolution des criticités

(c) Évolution des poids du modèle

(d) Évolution des positions

FIGURE 3.9 – Exploration manuelle

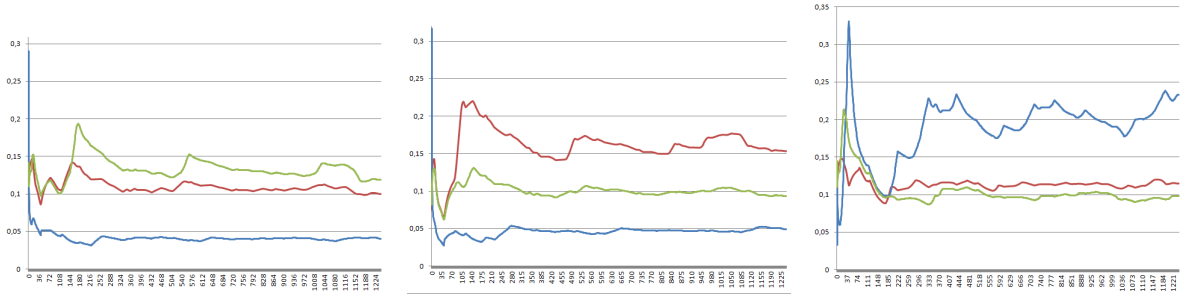
Lors d'une exploration aléatoire de cinq zones d'intérêt, nous ne sommes pas maîtres des commandes appliquées au drone. Le calcul des sensibilités est alors dépendant des directions que le drone prend, et s'il les prend de manière indépendante ou non des autres. En effet, si le drone se déplace selon deux axes de l'espace des perceptions en même temps, le mécanisme d'apprentissage n'a aucun moyen de savoir quelle variation de perception est reliée à quelle commande. C'est pourquoi figures 3.10a, 3.10b et 3.10c, les sensibilités sont plus chaotiques que pendant l'exploration manuelle. Dans ce cas là, les sensibilités ne seront bien classées qui si l'exploration aléatoire permet au drone d'évoluer dans le plus de directions possibles en favorisant les directions à une seule commande active.

Si l'on fait une moyenne des poids obtenus pour les figures 3.10d, 3.10e et 3.10f, on obtient le modèle moyen suivant :

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} 0.044861111 & 0 & 0.248194444 \\ 0.23415493 & 0 & 0 \\ 0.009099265 & 0.474816176 & 0.013051471 \end{pmatrix} . \begin{pmatrix} \delta P_0 \\ \delta P_1 \\ \delta P_2 \end{pmatrix} \tag{3.3}$$

On trouve alors des valeurs qui se rapprochent des poids obtenus pendant l'exploration ma-

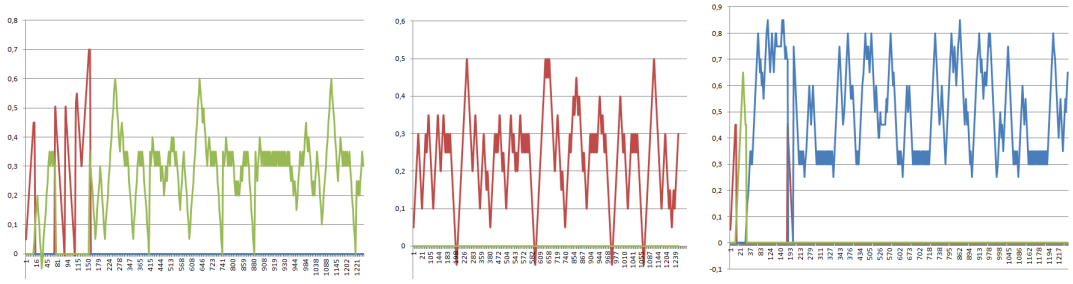
nuelle qui étaient d'environ 0.3 et 0.35.



(a) Évolution des sensibilités à C0 de P0 (rouge), P1 (bleu) et P2 (vert)

(b) Évolution des sensibilités à C1 de P0 (rouge), P1 (bleu) et P2 (vert)

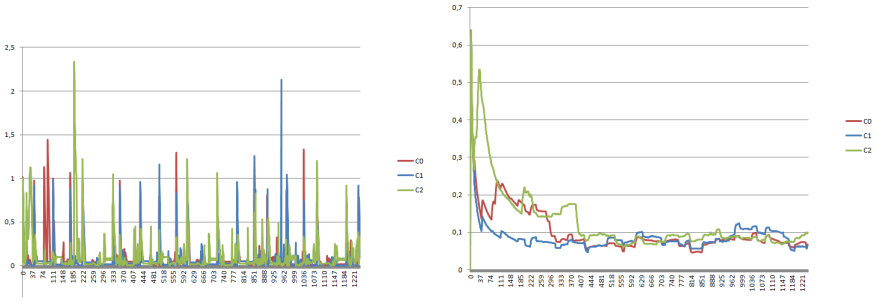
(c) Évolution des sensibilités à C2 de P0 (rouge), P1 (bleu) et P2 (vert)



(d) Évolution des poids w00 (rouge), w01 (bleu) et w02 (vert) du modèle de C0

(e) Évolution des poids w10 (rouge), w11 (bleu) et w12 (vert) du modèle de C1

(f) Évolution des poids w20 (rouge), w21 (bleu) et w22 (vert) du modèle de C2



(g) Évolution des criticités

(h) Évolution des criticités moyennes

FIGURE 3.10 – Exploration automatique par un asservissement successif sur cinq zones d'intérêt

Une exploration manuelle de l'espace de recherche permet au mécanisme d'apprentissage de trouver les liens de sensibilité entre les commandes et les perceptions plus rapidement qu'un exploration aléatoire. Lors, de l'exploration aléatoire, la génération de ces liens est entièrement dépendante de la trajectoire effectuée. Il se peut alors que le mécanisme d'apprentissage ne parviennent pas générer le modèle si le drone ne rencontre pas certaines variations pour ses perceptions.

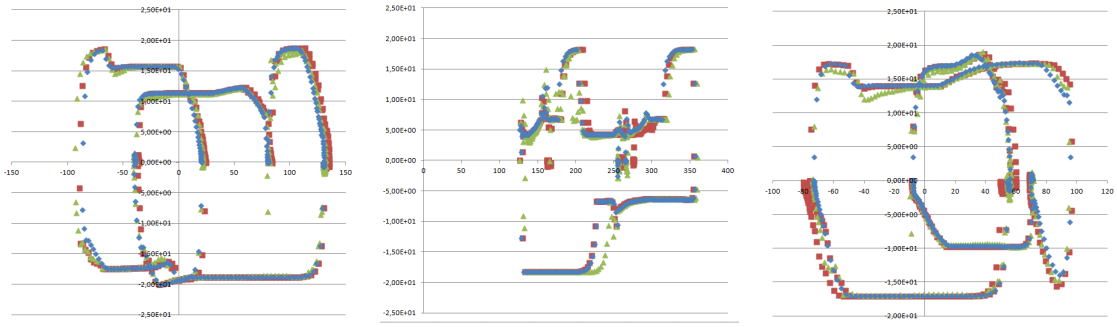
3.2.3 Asservissement vs AMOEBA vs AMALOM

Nous avons vu que l'exploitation d'AMOEBA après l'apprentissage par démonstrations permet d'effectuer les trajectoires initiales avec un léger retard dans le temps. Dans le cas où l'on introduit des perturbations, il est possible que l'exploitation ne permette pas de suivre comme il faut la trajectoire initiale. Nous allons voir dans cette partie comment le mécanisme AMALOM

permet d'améliorer l'exploitation de la trajectoire initiale.

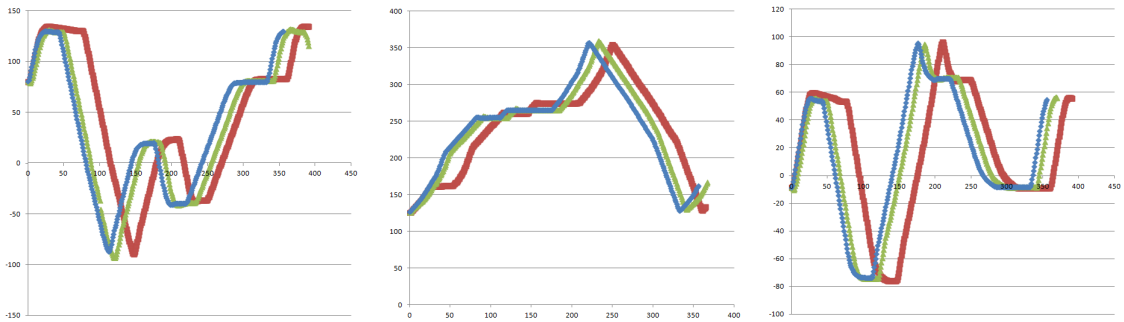
3.2.3.1 Exploitation sans perturbations

Lorsqu'il n'y pas de perturbations, on peut voir sur les portraits de phase de trajectoire figure 3.11, que les trajectoires ainsi effectuées par l'asservissement initial, AMOEBA et AMALOM sont semblables. Puis, lorsque l'on trace les positions prises pour ces trajectoires au cours du temps figure 3.12, on remarque que la trajectoire effectuée par AMALOM est moins retardée que celle effectuée par AMOEBA.



(a) P0 (b) P1 (c) P2

FIGURE 3.11 – Portraits de phase d'une trajectoire effectuée par l'asservissement/démonstrateur (bleu), par AMOEBA (rouge) et par AMALOM (vert) pour les trois dimensions de l'espace des perceptions, sans perturbations



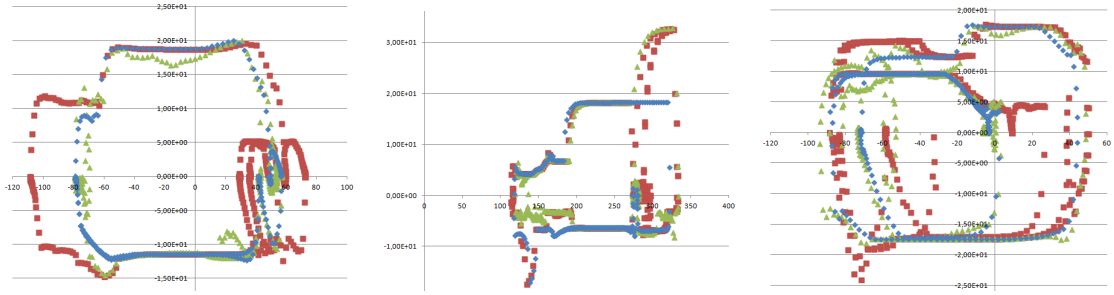
(a) P0 (b) P1 (c) P2

FIGURE 3.12 – Positions selon toutes les perceptions du drone en fonction du temps d'une trajectoire faite par l'asservissement/démonstrateur (bleu), par AMOEBA (rouge) et par AMALOM (vert), sans perturbations

3.2.3.2 Exploitation avec perturbations

En ajoutant des perturbations pendant l'exploitation des trajectoires apprises, on observe que les portraits de phase figure 3.13 sont plus chaotiques. Par exemple, figure 3.13a, on voit apparaître une oscillation pour la trajectoire effectuée par AMOEBA qui n'était pas présente initialement et qui n'est pas présente lors de l'utilisation d'AMALOM. Toujours sur cette figure, on peut observer un écart important pour la trajectoire effectuée par AMOEBA qui n'est pas présent pour l'exploitation d'AMALOM. C'est le résultat positif que nous attendions, AMALOM permet au drone de rester plus proche des zones connues et ainsi appliquer les connaissances

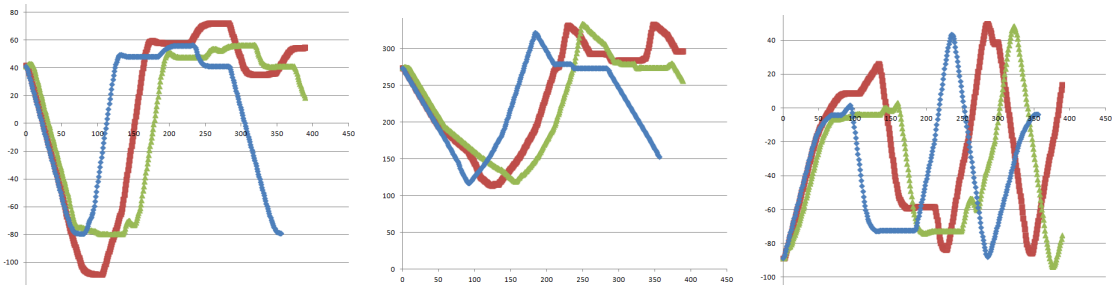
qui ont un sens dans ces zones là.



(a) P0 (b) P1 (c) P2

FIGURE 3.13 – Portraits de phase d'une trajectoire effectuée par l'asservissement/démonstrateur (bleu), par AMOEBA (rouge) et par AMALOM (vert) pour les trois dimensions de l'espace des perceptions, avec perturbations

En regardant les trajectoires en fonction du temps figure 3.14, on peut observer que même si les trajectoires faites par AMALOM prennent plus de retard que celle commandées par AMOEBA, les formes prises par les positions sont plus proches des formes initiales que ne le sont celles effectuées par AMOEBA. Par exemple, figure 3.14a, les positions prises lors de l'exploitation d'AMOEBA vont au delà des positions prises initialement. L'exploitation d'AMALOM prend plus de retard mais elle arrive à respecter les positions de la trajectoire initiale.



(a) P0 (b) P1 (c) P2

FIGURE 3.14 – Positions selon toutes les perceptions du drone en fonction du temps d'une trajectoire faite par l'asservissement/démonstrateur (bleu), par AMOEBA (rouge) et par AMALOM (vert), avec perturbations

3.3 Synthèse

Nous avons vu dans cette partie comment l'exploitation d'AMOEBA et AMALOM permettent d'effectuer des trajectoires apprises lors d'un apprentissage par démonstrations comportant cinq zones d'intérêt. Les résultats que nous avons présentés permettent de voir en quoi AMALOM permet d'améliorer l'exploitation de la trajectoire initiale.

Nous avons vu premièrement que des contextes de trop grande taille ne permettent pas d'effectuer à nouveau une trajectoire apprise par démonstration. On privilégie alors une grande quantité de contextes de petite taille.

Lorsqu'il n y pas de perturbations lors de l'exploitation de la trajectoire apprise par AMOEBA, la trajectoire est semblable à celle de l'asservissement initial. Cependant, l'introduction de perturbations ne permet pas d'exploiter l'apprentissage convenablement.

C'est pourquoi nous avons introduit l'apprentissage par feedback endogène d'un modèle de contrôle afin de mieux exploiter les modèles appris par AMOEBA. Cet apprentissage a pu être validé à partir de tests sur des modèles connus de plusieurs dimensions.

Cependant, ce mécanisme est dépendant des liens de sensibilité qu'il arrive à construire entre les entrées et les sorties du modèle qu'il génère. Ces liens sont les sensibilités des perceptions aux actions. Suivant l'exploration de l'espace qu'il effectue, ces liens sont plus ou moins explicites. Lors d'une exploration manuelle ou chacune des actions est effectuée de manière individuelle successivement, c'est facile pour le mécanisme de trouver ces liens et donc les coefficients du modèle. Lors d'une exploration aléatoire, la génération de ces liens sera dépendante de l'exploration et pourra ne pas aboutir dans certains cas.

Enfin, nous avons vu qu'AMALOM permet de mieux s'adapter aux perturbations que ne le fait AMOEBA. Ces perturbations entraînent un retard plus conséquent lors de la commande des trajectoires mais celle-ci est plus fidèle dans le domaine spatial.

Les résultats présentés ici sont des résultats préliminaires qui concernent un cas particulier d'expérimentation. Il faudrait par la suite mettre en place une métrique permettant de comparer les exploitations faites par AMOEBA et AMALOM pour d'autres types de trajectoires comportant plus de cinq zones d'intérêt.

Conclusion

Bilan du travail réalisé

L'objectif de cette étude était d'enrichir un apprentissage par démonstrations en concevant des méthodes d'auto-apprentissage capables d'exploiter les actions et perceptions d'un système interagissant avec son environnement. Le verrou à lever était la génération de feedbacks endogènes lors d'un apprentissage.

La mise en place d'un apprentissage par démonstrations à l'aide du mécanisme d'apprentissage AMOEBA et de la simulation de système robotique sous Unity a permis de confirmer les avantages comme les inconvénients de cette approche. Un tel apprentissage permet de guider l'exploration de l'espace de recherche mais ne permet pas de faire face à des situations imprévues jamais rencontrées lors des démonstrations.

Ce premier constat a confirmé le besoin d'un mécanisme d'apprentissage par feedback endogène. Basé sur l'approche des systèmes multi-agents auto-adaptatifs, le mécanisme AMALOM a permis de venir compléter un apprentissage par démonstrations en apportant au système robotique des capacités d'auto-observation. Grâce à ces capacités, AMALOM peut générer un modèle de contrôle du système robotique à partir de ses actions et perceptions afin d'apprendre à évoluer dans l'espace de ses perceptions. D'après les résultats préliminaires, l'apprentissage par démonstrations couplé à l'apprentissage du modèle de contrôle a rendu plus robuste aux perturbations l'exploitation des trajectoires apprises.

Ce premier prototype d'apprentissage par système multi-agent auto-adaptatif par feedback endogène dresse un bilan positif en ce qui concerne le verrou scientifique qui était la génération de feedbacks endogènes lors d'un apprentissage.

Perspectives

Les expérimentations présentées dans cette étude ne sont que des résultats préliminaires. Il faudrait réaliser d'autres mesures avec des trajectoires plus complexes et plus de zones d'intérêt pour notre cas d'application par exemple. Des expérimentations concernant des tâches robotiques autres que la réplication d'une trajectoire seraient intéressantes.

Afin de valider les propriétés de généricité, il serait nécessaire d'effectuer des expérimentations sur d'autres systèmes possédant des perceptions et des actions, robotiques ou non. Ceci permettrait de prouver que le mécanisme d'apprentissage par feedback endogène peut être utilisé sur plusieurs cas d'application.

Concernant la propriété de passage à l'échelle, nous avons déjà commencé à tester les performances du mécanisme d'apprentissage lorsque les modèles ont des tailles plus grandes. Cependant, ces tests doivent prendre en compte des modèles bien plus grands de dimensions telles que 1000x1000 pour valider le passage à l'échelle.

Pour pouvoir évaluer proprement les performances de la solution mise en place, il faut réfléchir à la mise en place d'une métrique. Elle permettrait de comparer l'efficacité avec laquelle le mécanisme d'apprentissage reproduit ce qu'il a appris avec et sans l'ajout du modèle de contrôle permettant au système de demeurer dans l'espace qu'il connait.

De plus, une autre voie à approfondir est celle de l'exploration de l'espace de recherche. L'exploration aléatoire ne permettant pas à 100% de générer les liens de sensibilité entre les actions et les perceptions, il est nécessaire de mettre en place des méthodes d'exploration permettant de minimiser le temps nécessaire à la génération de ces liens et de fournir avec certitude un modèle de contrôle.

Un autre point sur lequel nous ne nous sommes pas concentrés est la génération de motivations intrinsèques. Un système doté de motivations propres et d'un modèle de contrôle généré par feedbacks endogènes pourrait effectuer des déplacements dans l'espace de ses perceptions à partir de ses propres volontés.

Enfin, au cours de cette étude, nous sommes restés au niveau expérimental de la simulation. Il faudrait par la suite mettre en place un tel apprentissage sur un dispositif réel. Pour le cas d'application du drone, une suite intéressante serait d'apprendre directement le comportement des rotors du drone.

Bibliographie

[Argall et al., 2009] Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5) :469–483.

[Asada et al., 2009] Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., and Yoshida, C. (2009). Cognitive developmental robotics : A survey. *IEEE Transactions on Autonomous Mental Development*, 1(1) :12–34.

[Ayer Alfred, 1952] Ayer Alfred, J. (1952). Language, truth, and logic.

[Boes et al., 2014] Boes, J., Migeon, F., Glize, P., and Salvy, E. (2014). Model-free optimization of an engine control unit thanks to self-adaptive multi-agent systems. In *International Conference on Embedded Real Time Software and Systems-ERTS^2 2014*, pages pp–350.

[Boes et al., 2015] Boes, J., Nigon, J., Verstaevel, N., Gleizes, M.-P., and Migeon, F. (2015). The self-adaptive context learning pattern : Overview and proposal. In *International and Interdisciplinary Conference on Modeling and Using Context*, pages 91–104. Springer.

[Bondu and Lemaire, 2007] Bondu, A. and Lemaire, V. (2007). Active learning using adaptive curiosity. In *International Conference on Epigenetic Robotics : Modeling Cognitive Development in Robotic Systems*.

[Bonjean et al., 2014] Bonjean, N., Mefteh, W., Gleizes, M. P., Maurel, C., and Migeon, F. (2014). Adelfe 2.0. In *Handbook on agent-oriented design processes*, pages 19–63. Springer.

[Brooks, 1991] Brooks, R. A. (1991). Intelligence without representation. *Artificial intelligence*, 47(1-3) :139–159.

[Chaput, 2004] Chaput, H. H. (2004). *The constructivist learning architecture : A model of cognitive development for robust autonomous robots*. PhD thesis.

[Cohen, 1998] Cohen, L. B. (1998). An information-processing approach to infant perception and cognition. *The development of sensory, motor, and cognitive capacities in early infancy*, pages 277–300.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1) :21–27.

[d'Amico, 2016] d'Amico, D. (2016). Concevoir un système artificiel sans finalité a priori.

[Dautenhahn and Nehaniv, 2002] Dautenhahn, K. and Nehaniv, C. L. (2002). *Imitation in animals and artifacts*. MIT Press Cambridge, MA.

[Drescher, 1991] Drescher, G. L. (1991). *Made-up minds : a constructivist approach to artificial intelligence*. MIT press.

[Ferber, 1999] Ferber, J. (1999). *Multi-agent systems : an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading.

[Fiesler and Beale, 1996] Fiesler, E. and Beale, R. (1996). *Handbook of neural computation*. Oxford University Press.

[Georgé et al., 2011] Georgé, J.-P., Gleizes, M.-P., and Camps, V. (2011). Cooperation. In *Self-organising Software*, pages 193–226. Springer.

[Gibson, 1969] Gibson, E. J. (1969). Principles of perceptual learning and development.

[Gleizes et al., 2008] Gleizes, M.-P., Camps, V., Georgé, J.-P., and Capera, D. (2008). Engineering systems which generate emergent functionalities. In *Engineering Environment-Mediated Multi-Agent Systems*, pages 58–75. Springer.

[Glize, 2001] Glize, P. (2001). L'adaptation des systèmes à fonctionnalité émergente par auto-organisation coopérative. *Hdr, Université Paul Sabatier, Toulouse III*.

[Guerin, 2011] Guerin, F. (2011). Learning like a baby : a survey of artificial intelligence approaches. *The Knowledge Engineering Review*, 26(2) :209–236.

[Holland and Reitman, 1977] Holland, J. H. and Reitman, J. S. (1977). Cognitive systems based on adaptive algorithms. *Acm Sigart Bulletin*, (63) :49–49.

[Holmes et al., 2005] Holmes, M. P. et al. (2005). Schema learning : Experience-based construction of predictive action models. In *Advances in Neural Information Processing Systems*, pages 585–592.

[Kalenka and Jennings, 1999] Kalenka, S. and Jennings, N. R. (1999). Socially responsible decision making by autonomous agents. In *Cognition, Agency and Rationality*, pages 135–149. Springer.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4) :541–551.

[Lee et al., 2007] Lee, M. H., Meng, Q., and Chao, F. (2007). Staged competence learning in developmental robotics. *Adaptive Behavior*, 15(3) :241–255.

[MacGlashan and Littman, 2015] MacGlashan, J. and Littman, M. L. (2015). Between imitation and intention learning. In *IJCAI*, pages 3692–3698.

[Mazac et al., 2015] Mazac, S., Armetta, F., and Hassas, S. (2015). Approche décentralisée pour un apprentissage constructiviste en environnement continu : application à l'intelligence ambiante. In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA)*. CÉPADUÈS ÉDITIONS.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4) :115–133.

[Mohri et al., 2012] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of machine learning*. MIT press.

[Moon, 1996] Moon, T. K. (1996). The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6) :47–60.

[Nehaniv et al., 2002] Nehaniv, C. L., Dautenhahn, K., et al. (2002). The correspondence problem. *Imitation in animals and artifacts*, 41.

[Ng et al., 2000] Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670.

[Nigon et al., 2016] Nigon, J., Gleizes, M.-P., and Migeon, F. (2016). Self-adaptive model generation for ambient systems. *Procedia Computer Science*, 83 :675–679.

[Oudeyer et al., 2007] Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2) :265–286.

[Perotto and Álvares, 2006] Perotto, F. S. and Álvares, L. O. (2006). Learning regularities with a constructivist agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 807–809. ACM.

[Pfeifer and Bongard, 2006] Pfeifer, R. and Bongard, J. (2006). *How the body shapes the way we think : a new view of intelligence*. MIT press.

[Piaget, 1976] Piaget, J. (1976). Piaget's theory. In *Piaget and his school*, pages 11–23. Springer.

[Power, 1999] Power, T. G. (1999). *Play and exploration in children and animals*. Psychology Press.

[Rochat, 2003] Rochat, P. (2003). Five levels of self-awareness as they unfold early in life. *Consciousness and cognition*, 12(4) :717–731.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386.

[Russell and Norvig, 2002] Russell, S. J. and Norvig, P. (2002). Artificial intelligence : a modern approach (international edition).

[Schaal et al., 2003] Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London B : Biological Sciences*, 358(1431) :537–547.

[Serugendo et al., 2011] Serugendo, G. D. M., Gleizes, M.-P., and Karageorgos, A. (2011). *Self-organising software : From natural to artificial adaptation*. Springer Science & Business Media.

[Stojanov et al., 1997] Stojanov, G., Bozinovski, S., and Trajkovski, G. (1997). Interactionist-expectative view on agency and learning. *Mathematics and Computers in Simulation*, 44(3) :295–310.

[Stoytchev, 2006] Stoytchev, A. (2006). Five basic principles of developmental robotics. In *NIPS Workshop on Grounding, Perception, Knowledge, and Cognition*.

[Stoytchev et al., 2007] Stoytchev, A., Berthouze, L., Prince, C., Littman, M., Kozima, H., and Balkenius, C. (2007). Toward video-guided robot behaviors. In *Proceedings of the Seventh International Conference on Epigenetic Robotics (EpiRob)*, volume 135, pages 165–172.

[Sutton, 2001] Sutton, R. S. (2001). Verification, the key to ai. *on-line essay.[Online]. Available : http ://www. cs. ualberta. ca/sutton/IncIdeas/KeytoAI. html*.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning : An introduction*, volume 1. MIT press Cambridge.

[Tomasello and Call, 1997] Tomasello, M. and Call, J. (1997). *Primate cognition*. Oxford University Press, USA.

[Verstaevel, 2016] Verstaevel, N. (2016). *Self-organization of robotic devices through demonstrations*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier.

[Verstaevel et al., 2017] Verstaevel, N., Boes, J., Nigon, J., d'Amico, D., and Gleizes, M. P. (2017). Lifelong machine learning with adaptive multi-agent systems. In *ICAART (2)*, pages 275–286.

[Verstaevel et al., 2016] Verstaevel, N., Régis, C., Gleizes, M.-P., and Robert, F. (2016). Principles and experimentations of self-organizing embedded agents allowing learning from demonstration in ambient robotics. *Future Generation Computer Systems*, 64 :78–87.

[Weng et al., 2001] Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., and Thelen, E. (2001). Autonomous mental development by robots and animals. *Science*, 291(5504) :599–600.

[Weyns et al., 2004] Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., and Ferber, J. (2004). Environments for multiagent systems state-of-the-art and research challenges. *E4MAS*, 3374 :1–47.

[Zlatev and Balkenius, 2001] Zlatev, J. and Balkenius, C. (2001). Introduction : Why òepigenetic roboticsó ?