Auto-organisation d'agents embarqués pour l'apprentissage par démonstration : principes et expérimentations

N. Verstaevel[a, b] verstaev@irit.fr
C. Régis[b] regis@irit.fr
M.P. Gleizes[b] gleizes@irit.fr
F. Robert[a] fabrice.robert@sogeti.com

[a] Sogeti High Tech,
3 Chemin de Laporte, Toulouse, France

[b] IRIT, Équipe SMAC,
Université Paul Sabatier, Toulouse, France

Résumé

L'adaptation des systèmes ambiants aux besoins spécifiques des utilisateurs est une tâche complexe. Pour rendre l'interaction humain-système aussi naturelle que possible pendant l'adaptation, nous proposons une approche basée sur l'apprentissage par démonstration. Cet apprentissage requiert des techniques d'apprentissage adaptatifs. Nous présentons Alex, un système multi-agent capable d'apprendre dynamiquement un comportement à partir de démonstrations réalisées par un tuteur. Les résultats d'expérimentations réalisées à la fois sur un robot réel et virtuel mettent en avant des propriétés intéressantes de notre technologie pour des applications ambiantes.

Mots-clés *: Robotique collective, Intelligence ambiante, Apprentissage, Auto-organisation*

Abstract

The adaptation of an ambient system to the specific needs of its users is a challenging task. Because human-system interaction has to be as natural as possible, we propose an approach based on Learning from Demonstration (LfD). However, using LfD in ambient systems needs adaptivity of the learning technic. We present Alex, a multi-agent system able to dynamically learn and reuse contexts from demonstration made by a tutor. Results of experiments performed on both a real and virtual robot show interesting properties of our technology for ambient applications.

Keywords*: Collective robotic, Ambient intelligence, Learning, Self-organization*

1 Introduction

Autrefois restreintes à une science du contrôle en environnements contraints, les recherches en robotique s'intéressent aujourd'hui à l'intégration de systèmes intelligents ou cyber-physiques autonomes dans des environnements où les tâches sont multiples, complexes et évolutives (Kaminka, 2012). La robotique de service diffère de sa version industrielle parce qu'elle fournit des services à des utilisateurs plutôt que de réaliser des tâches répétitives adaptées au processus de fabrication. C'est pourquoi les scientifiques s'intéressent à l'utilisation de dispositifs robotiques dans des applications ambiantes (Chung, 2014). Ces systèmes ambiants se caractérisent par leur dynamique et leur complexité. Un grand nombre de dispositifs (capteurs et effecteurs) hétérogènes peuvent apparaître et disparaître pour interagir de manière opportuniste. (Russel et al, 1995) décrivent l'environnement de ces dispositifs comme :

- **Inaccessible** : chaque entité n'a qu'une observation partielle de l'environnement.

- **Continu** : le nombre d'observations et

d'actions dans le monde réel n'est pas discret.

- **Non-déterministe** : les conséquences d'une action réalisée dans le monde réel ne peuvent pas être déterminées à l'avance.

- **Dynamique** : les actions du système, l'activité des utilisateurs, l'apparition et la disparition de dispositifs peuvent changer l'environnement.

Le caractère *distribué* des systèmes ambiants ne permet pas l'utilisation d'un processus de contrôle centralisé. À l'inverse, chaque dispositif robotique doit être doté de son propre contrôleur. La conception *ad hoc* d'un contrôleur pour un dispositif robotique dans un système ambiant est une tâche complexe qui nécessite une forte expertise et implique des coûts exponentiels de développement et de maintenance. En effet, elle nécessite de lister *a priori* tous les cas d'utilisation du dispositif robotique ainsi que ses différentes interactions. La tâche est d'autant plus complexe si l'on prend en compte le caractère spécifique, multiple et changeant des besoins des utilisateurs. Doter ces dispositifs de la capacité d'apprendre à s'adapter aux besoins de ses utilisateurs est une tâche particulièrement difficile (Hagras, 2004) qui présente un intérêt certain dans la réduction des coûts de développement et de maintenance. Dans cet article, nous proposons une approche nommée « *eXtreme Sensitive robotic*» (Verstaevel, 2015). L'*XS Robotic* considère chaque fonctionnalité du robot comme une entité autonome. Un robot est donc un agrégat de plusieurs *XS Function*, globalement une pour chaque capteur et effecteur. Une XS Function est soit sensitive (capteur température, ultrason,…) soit active (un moteur, une LED,…). Chaque *XS Function* est conçue comme *non finale*, c'est-à-dire que la fonctionnalité doit pouvoir s'adapter. En exploitant ses capacités d'auto-observation, le dispositif robotique observe des variations dans son environnement et déclenche les mécanismes d'adaptation adéquats. Avec cette approche, la production d'un comportement complexe par un ensemble de dispositifs émerge de l'interaction entre les différentes *XS Function* et l'environnement. Il n'y a donc pas de différence de conception entre un robot composé de plusieurs capteurs et effecteurs, une application multi-robot composée de plusieurs robots en interaction, ou un système ambiant. Tous ces systèmes sont composés de fonctionnalités distribuées au sein de différents dispositifs robotiques qui doivent s'adapter pour satisfaire leurs utilisateurs.

Pour être naturelle, cette adaptation doit reposer sur un procédé qui ne requiert pas d'expertise particulière pour l'utilisateur. De plus, elle nécessite à la fois d'être **générique**, pour être applicable à tout type de dispositif et pour tout utilisateur, ainsi que de respecter la propriété d'**ouverture** pour gérer l'apparition et la disparition de dispositifs. Le respect de la généricité et de l'ouverture requiert l'utilisation de techniques d'apprentissage *agnostiques* (Kearns, 1994) qui ne font que peu d'hypothèses sur leur environnement. Pour respecter ces propriétés, nous proposons l'utilisation de l'apprentissage par démonstration, un paradigme permettant l'apprentissage dynamique de nouveaux comportements.

2 Apprentissage par démonstration

L'apprentissage par démonstration, aussi appelé « apprentissage par imitation », est un paradigme principalement étudié en robotique qui permet à un système d'apprendre dynamiquement de nouveaux comportements (Argall, 2009). L'idée principale est qu'un contrôleur adéquat peut être appris de l'observation de l'activité d'une entité externe (humaine ou virtuelle) nommée *tuteur*. Le tuteur interagit avec le système au travers d'un processus de *démonstrations*, qui consiste en la réalisation d'une succession d'actions dans des situations particulières. Le système doit alors apprendre une fonction corrélant l'observation de son environnement avec les actions réalisées par le tuteur. L'avantage principal de cette technique est qu'elle ne nécessite aucune programmation explicite et aucune expertise sur le système de la part du tuteur. Des travaux

récents proposent une étude du domaine de l'apprentissage par démonstration (Argall, 2009) (Billard, 2008) et (Billing, 2010) fournit une formalisation complète. L'apprentissage par démonstration est un problème d'imitation où une entité doit produire un comportement similaire à une autre entité. Un *tuteur* évoluant dans un monde Ω peut réaliser un ensemble d'actions *A*. Le tuteur suit une politique $\Pi : \Omega \rightarrow A$ qui associe à chaque situation du monde une action. Il est supposé que cette politique est optimale pour satisfaire l'utilisateur. L'apprenant (aussi nommé *imitateur*) possède un ensemble d'observations *O* (nommé *espace d'observations*) sur le monde Ω et suit une politique $\Pi' : O \rightarrow A$ telle *que* $\Pi' \equiv \Pi$. L'apprenant doit donc trouver des corrélations entre la réalisation d'une action par son tuteur et des variations dans l'observation qu'il fait du monde. La prochaine section présente Alex, un système multi-agent pour l'apprentissage par démonstration conçu à partir de la vision *XS Robotic*.

L'application de ce paradigme aux systèmes ambiants repose sur la richesse des interactions avec leurs utilisateurs. L'utilisateur peut être vu comme tuteur du système et chacune de ses actions sur un dispositif comme une démonstration du comportement désiré. En effet, si l'utilisateur nécessite d'intervenir sur une fonctionnalité, c'est que le service fourni par cette fonctionnalité ne le satisfait plus. Le dispositif robotique contrôlant la fonctionnalité concernée peut alors exploiter cette information pour s'adapter. Le rôle d'un dispositif robotique est alors de corréler l'activité de l'utilisateur à l'observation qu'il fait de son environnement.

3 Alex: Adaptive Learner by EXperiments

L'utilisation du paradigme multi-agent (Ferber, 1999) dans l'XS Robotic apparaît naturelle pour sa capacité à faire face à la complexité et à apprendre en interaction avec son environnement (Panait, 2005). (Kaminka, 2012) montre l'apport des systèmes multi-agents pour la robotique. Plusieurs approches proposent dès le début des années 2000 l'utilisation de systèmes multi-agents pour permettre l'organisation d'un collectif de robots (Collinot, 1999) (Parker, 2000) (Picard, 2005) et encore de nos jours ce paradigme est utilisé pour des tâches complexes telles que l'exploration collective (Souliman, 2014) ou l'intervention en situation de crise (Lacouture, 2012). Ces approches considèrent souvent le robot comme un simple agent. Dans cet article, nous soutenons que la distribution du contrôle au sein de chaque fonctionnalité du robot permet une plus grande adaptation de celui-ci. Un robot devient alors un système multi-agent, composé d'un ensemble de fonctionnalités autonomes. Nous présentons Alex, un système multi-agent pour l'apprentissage par démonstration conçu à partir de la vision *XS Robotic*.

Alex est un système multi-agent construit pour apprendre à contrôler une fonctionnalité à partir de démonstrations réalisées par un tuteur. Sa conception est basée sur l'approche des systèmes multi-agents adaptatifs (AMAS) (Gleizes, 2012) qui adresse la problématique des systèmes complexes par une approche « bottom-up » où le concept de coopération agit comme moteur de l'auto-organisation. Le théorème de l'adéquation fonctionnelle (Capera, 2003) stipule que « pour tout système fonctionnellement adéquat, il existe au moins un système à milieu interne coopératif qui réalise la même fonction dans le même environnement ». Le rôle d'un AMAS est alors d'automatiquement détecter et réparer des situations de non-coopération en s'auto-organisant pour atteindre un état fonctionnellement adéquat. La conception d'Alex suit la méthodologie ADELFE2.0 (Bonjean, 2014) qui guide le concepteur d'un système adaptatif.

3.1 Context-Learning

La partie 2 illustre l'impossibilité de lister *a priori* toutes les situations qu'un robot peut rencontrer. Cette incapacité à spécifier

complètement le comportement du dispositif rend obligatoire son adaptation aux différents contextes d'utilisations (Makkonen, 2009). Le terme « contexte » n'a pas de définition communément admise, aussi fait-il référence dans cet article à toute information externe à l'activité d'une entité qui affecte son activité. Cet ensemble d'informations décrit l'environnement tel que le perçoit l'entité (Guivarch, 2014). Un système capable d'exploiter des informations de son contexte est alors nommé « context-aware ». Alex est conçu pour interagir continuellement avec son environnement et apprendre dynamiquement ses différents contextes. Ainsi, il construit une politique Π qui associe à chaque contexte l'action adéquate à réaliser. Pour ce faire, il dispose de capacités *d'auto-observation* pour construire dynamiquement des corrélations entre la réalisation d'une action et l'effet de cette action sur l'environnement. Une instance d'Alex est composée d'un ensemble non fini d'agents contextes. Chaque agent contexte est porteur d'une corrélation associant à un état de l'environnement une action. Cet ensemble, vide au départ, est dynamiquement peuplé au fur et à mesure qu'Alex détecte de nouvelles situations. Les agents contextes s'auto-organisent (voire disparaissent) pour effectuer un contrôle sur une fonctionnalité.

Les agents contextes sont décrits en décomposant leur comportement en deux types. Le comportement *nominal* de l'agent correspond à son comportement lorsque le dispositif robotique se trouve dans un état fonctionnellement adéquat qui ne nécessite pas d'adaptation. Le comportement *coopératif* est une subsomption du comportement nominal qui s'exécute lorsqu'une situation non coopérative est détectée. Il conduit alors à l'auto-organisation de l'entité en interaction avec les autres entités du système afin de retourner dans un état fonctionnellement adéquat.

3.2.1 Comportement nominal

Un ***agent contexte*** associe une description du contexte noté *V* (voir section 3.2.2) à une action unique dont il ne connait pas la sémantique. Ces actions peuvent être interprétées par un observateur humain comme « Avance », « Tourne à droite » ou « vitesse à 42% ». Un agent contexte reçoit un ensemble *O* de signaux de son environnement qu'il utilise pour caractériser le contexte courant du système. Quand l'observation qu'il fait de l'environnement correspond à sa description *V*, l'agent contexte déclenche son action $a \in A$ ou décide de ne rien faire (θ). Le comportement nominal N d'un agent contexte se résume à la fonction $N : O \times V \rightarrow a||\theta$. Le comportement Π du dispositif est donc résultant de l'interaction entre les agents contextes.

3.2.2 Description du contexte

Pour déclencher son action, l'agent contexte a besoin de comparer la situation courante avec sa représentation *V*. Cette représentation du contexte est réalisée grâce à des plages de valeurs (nommées ***plages de validité***). Un agent contexte reçoit un ensemble d'observations *O* de son environnement. Chaque $o \in O$ est une valeur continue incluse dans un espace de définition $o \in [o_{min}, o_{max}]$. Par exemple, une observation émanant d'un capteur de température $temp \in [-20,40]$ où -20 et 40 sont les températures minimales et maximales du capteur. Ces valeurs *min* et *max* ne sont pas connues *a priori* par Alex.

Une plage de validité *v* est associée à chaque observation telle que v_o correspond à la plage de validité associée à l'observation *o*. L'ensemble des plages de validité *V* forme le ***domaine de validité***. Une plage de validité est composée de deux valeurs v_{min} et v_{max} telle que $[v_{min}, v_{max}] \subseteq [o_{min}, o_{max}]$. Le rôle d'une plage de validité est d'adapter dynamiquement la valeur des deux bornes v_{min} et v_{max}. Une plage de validité est ***valide*** si et seulement si $o \in [v_{min}, v_{max}]$, c'est-à-dire si la valeur courante de l'observation est comprise dans les bornes de la plage de validité. Le domaine de validité sera *valide* si toutes ses plages de validité sont *valides*. Ainsi, un agent contexte dont le

domaine de validité est *valide* déclenche son action, considérant que le contexte courant du système correspond à sa représentation. La gestion des plages de validité se fait grâce à des trackers adaptatifs nommés Adaptive Value Trackers (AVT) (Lemouzy, 2011). Cet outil permet de trouver automatiquement une valeur à partir de feedbacks successifs. Il est à noter qu'avec cette représentation, aucune sémantique n'est associée au signal et que seuls des tests d'inclusion ensembliste sont réalisés.

3.2.3 Comportement coopératif

À chaque cycle, le dispositif robotique ne peut réaliser qu'une unique action dans un seul état à la fois. Les agents contextes *valides* doivent alors coopérer pour déterminer si leur action est la plus appropriée dans le contexte courant. Pour être dans un état fonctionnellement adéquat, deux agents contextes *valides* ne devraient pas proposer d'actions antagonistes (par exemple, deux contextes proposant chacun d'allumer et d'éteindre une lumière). Cependant, en ne conservant que le comportement nominal, des situations de ***non coopération*** doivent être résolues pour amener le système dans un état fonctionnellement adéquat. On distingue trois situations de non-coopération. La première (a) se produit lorsqu'au moins deux agents contextes proposent de réaliser une action (*concurrence*). La seconde (b) se produit si l'action proposée par un agent contexte est en contradiction avec l'action réalisée par le tuteur (*conflit*). La dernière (c) se produit lorsqu'aucun agent contexte ne propose d'action (*inutilité*). Pour résoudre ces situations, les agents contextes suivent un comportement coopératif.

3.2.3.1 Confiance

Un agent contexte détermine un niveau de *confiance* qui représente la pertinence de la réalisation de son action. Cette confiance permet aux agents contextes de coopérer. Si un agent contexte se trouvant *valide* veut réaliser une action alors que sa confiance est inférieure à celle d'un autre agent contexte *valide*, il doit modifier son domaine de validité en ajustant les bornes de ses plages de validité afin d'exclure la situation courante. Ainsi, la prochaine fois que la même situation se reproduira, cet agent contexte ne proposera pas son action. La confiance *c* d'un agent contexte est régie par la fonction suivante :

$$c_{t+1} = c_t * (1 - \alpha) + F_t * \alpha.$$

c_{t+1} est la confiance de l'agent contexte à l'instant $t+1$. $F_t \in 0|1$ est la valeur de feedback et $\alpha \in [0,1]$ est un paramètre fixé qui modélise l'importance du feeback. Chaque fois qu'un agent contexte a raison de proposer son action, $F_t = 1$. À l'inverse, chaque fois que cet agent fait une mauvaise proposition, $F_t = 0$. Dans notre implémentation, α est empiriquement fixé à 0,8 et la valeur de confiance est initialisée à 0,5. La confiance des agents contextes permet de lever l'ambiguïté soulevée par la situation de non coopération (a). Plus l'agent contexte fait de bonnes propositions, plus sa confiance est élevée. À l'inverse, plus l'agent a fait de mauvaises propositions, plus sa confiance est faible. La valeur minimale de confiance est fixée à 0,1, ce qui signifie que tout contexte avec une confiance inférieure à 0,1 s'autodétruira.

3.2.3.2 Adéquation avec le tuteur

Lorsque le tuteur agit sur un dispositif, son action est perçue par les agents contextes. Si un agent contexte est *valide* dans la situation courante, mais propose une action différente de celle de l'utilisateur, il va alors ajuster les bornes de ses plages de validité afin d'exclure la situation courante en ajustant les bornes v_{min} et v_{max}. Si l'agent contexte perçoit une donnée de l'environnement qui n'est pas encore incluse dans ses plages de validité (suite par exemple à l'ajout d'un capteur dans l'environnement), il ajoute une nouvelle plage de valeur associée à cette observation. À l'inverse, si l'agent contexte propose une action en adéquation avec le tuteur, il renforce sa confiance. Dans tous les cas, quand le tuteur

agit, son action subsume celle des agents contextes. Ces mécanismes permettent la résolution de la situation de coopération (b). Le processus d'adaptation des bornes des plages de validité peut conduire à une situation où $v_{max} < v_{min}$. Si une telle situation se produit, la plage de valeur concernée est plus valide. L'agent contexte s'autodétruit.

3.2.3.3 Anticipation et création d'agent contexte

L'état fonctionnellement adéquat du dispositif robotique nécessite qu'il existe un unique agent contexte dont l'action est activée par cycle de décision. Cependant, le dispositif peut se retrouver dans des situations entièrement nouvelles où aucun agent contexte n'est *valide* et par conséquent être dans une situation d'incompétence. Deux mécanismes peuvent résoudre cette situation : étendre les bornes de validité de l'agent contexte existant ou créer un nouvel agent contexte pour représenter cette nouvelle situation.

Pour anticiper une situation d'incompétence, le concept de ***validable*** est introduit. Une plage de validité v_o est validable si et seulement si $o \notin [v_{min}, v_{max}]$ et $o \in [v_{min} - \Delta_{min}, v_{max} + \Delta_{max}]$. Δ_{min} et Δ_{max} sont deux valeurs (une pour chaque borne) gérées dynamiquement par les AVTs pour contrôler l'évolution de chaque borne. Δ peut être interprété comme le prochain incrément de la borne. Avec le concept de *validable*, les agents contextes peuvent proposer leur action dans des situations suffisamment leur plage de validité. Si l'action proposée par l'agent est la plus adéquate, il adapte alors ses bornes pour inclure la situation courante. À l'inverse, si la proposition n'était pas pertinente, les AVT diminueront leur Δ pour ne pas se proposer la prochaine fois.

Lorsqu'aucun agent contexte n'est *valide* ou *validable*, le dispositif robotique est face à une situation d'incompétence, il ne peut décider de l'action à réaliser. L'agent contexte *valide* à l'étape précédente (mais qui n'est plus *valide* dans la situation courante) peut alors créer un nouvel agent contexte destiné à le remplacer en lui associant l'action effectuée par le tuteur. L'agent contexte ainsi créé initialise ses plages de validité pour représenter la situation courante.

Si le tuteur n'a pas effectué d'action, deux mécanismes peuvent se produire, selon l'autonomie désirée par le tuteur. Avec une approche *collaborative*, le dispositif peut interroger le tuteur sur l'action à réaliser. Avec une approche *autonome*, le dispositif maintient la dernière action qu'il a su réaliser, considérant l'inaction du tuteur comme une demande de maintien de l'action courante. Ce mode peut être choisi à l'initialisation du dispositif et être changé dynamiquement.

Ces deux mécanismes, le concept de validabilité et la création d'agents contexte, permettent de résoudre la situation de non coopération (c). Dans la suite de cet article, nous proposons d'illustrer le comportement d'Alex sur une application concrète.

4 Expérimentations

Un robot muni de deux roues sans aucun capteur est immergé dans une arène de 2mx2m composée de deux artefacts bleu et vert. Une caméra située 2m au-dessus du centre de l'arène observe la scène et peut analyser les pixels pour capturer la position des artefacts bleu et vert par rapport à la position du robot (Figure 1). La caméra peut produire quatre observations sur la scène correspondant à la distance, et l'angle pour chaque artefact. Pour montrer la capacité à gérer l'apparition et la disparition de signaux, la camera envoie les

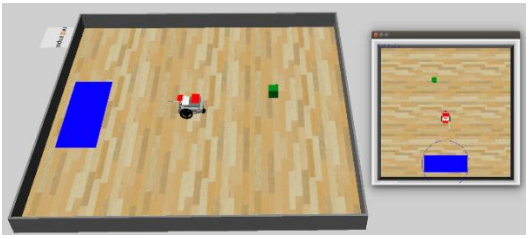

Figure 1- L'expérimentation sous Webots™.

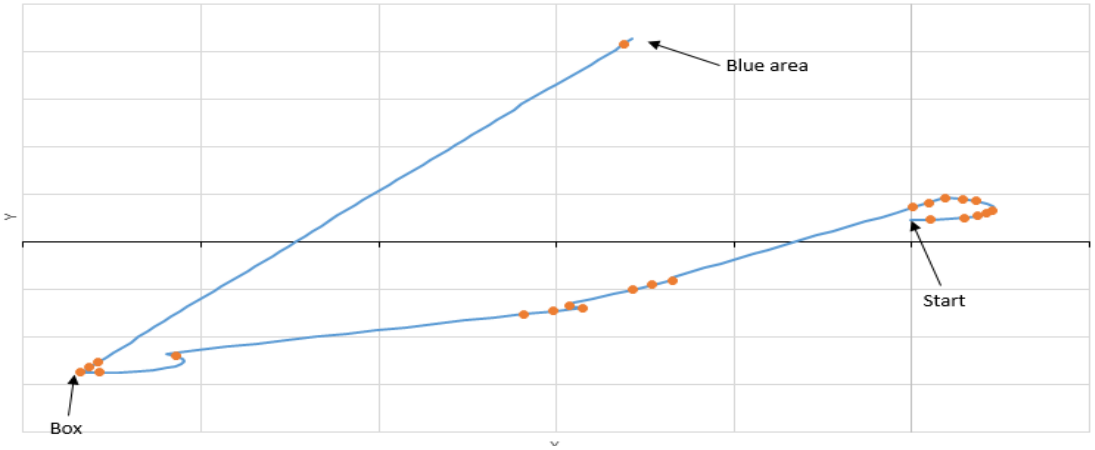

Figure 2 – Trajectoire du robot et zones de création d'agents contextes.

informations de positions au robot uniquement si les artefacts se situent face au robot. À chaque étape, la caméra peut alors produire zéro, deux ou quatre observations.

Le tuteur humain peut contrôler le robot au travers d'une manette. Chaque joystick contrôle la vitesse d'une roue (le joystick gauche contrôle la roue gauche et réciproquement). Cette valeur, comprise entre [-100,100], correspond au pourcentage de la vitesse maximale à appliquer ainsi qu'au sens de rotation. Le but de l'expérimentation est de montrer la capacité d'Alex d'apprendre des comportements complexes à partir de l'observation des actions du tuteur. Pour ce faire, le tuteur peut montrer un ensemble de comportements dans l'arène qu'Alex doit imiter. L'expérience est réalisée à la fois sur un robot virtuel et sur un robot réel.

En accord avec la vision *XS Robotic*, une instance d'Alex est associée à chaque roue leur permettant d'agir de manière autonome. Il n'existe aucune communication explicite entre les deux roues. Le rôle d'une instance d'Alex est alors d'apprendre à contrôler une roue en corrélant l'activité du tuteur aux perceptions qu'il reçoit de la caméra. À chaque étape de décision, Alex peut recevoir les observations de la caméra ainsi que l'action réalisée par le tuteur. Un cycle de décision intervient tous les 250 ms. La même implémentation d'Alex est utilisée à la fois en simulation et dans l'application réelle. OpenCV est utilisé sur les deux applications pour effectuer le traitement d'images. L'expérience est simulée sous Webots™. L'expérience réelle a été réalisée grâce à un robot Boe-Bot basé sur une plateforme Arduino et l'utilisation d'un protocole de communication Xbee. Une des activités que le tuteur réalise dans l'arène est une tâche de collecte d'artefacts. Le robot est muni de deux branches solides permettant la capture de petits objets. Ces branches sont fixes et ne comportent aucun capteur. Le but de l'activité est alors de capturer les artefacts verts et de les transporter vers l'artefact bleu. À chaque fois qu'un artefact est déposé dans la zone bleue, il est soit, dans la simulation, déplacé aléatoirement dans l'arène, soit déplacé par le tuteur dans l'application réelle. Les résultats présentés ont été obtenus à partir des simulations et constatés sur l'expérimentation réelle. L'expérimentation se propose d'illustrer le processus d'adaptation durant les phases de démonstrations pour l'apprentissage et les phases de fonctionnement en autonomie du robot.

4.1 L'activité du tuteur permet la création d'agent contextes

La figure 2 montre la trajectoire suivie par le robot pendant une phase de démonstration du comportement de collecte. Dans cette démonstration, le tuteur prend le contrôle du robot et le conduit afin de capturer l'artefact vert puis le ramener vers l'artefact bleu. Chaque point correspond à la création d'un

agent contexte par l'une des deux instances d'Alex. En observant la figure, on constate que les agents contextes ne sont créés qu'au cours de changements de direction commandés par le tuteur. À l'inverse, quand le tuteur maintient sa direction, aucun agent contexte n'est créé. Alex observe chaque action du tuteur et détermine si celle-ci appartient à un contexte existant ou à un nouveau contexte. La création de nouveaux agents contextes résulte donc de l'observation de l'activité du tuteur.

4.2 L'activité du tuteur permet l'adaptation des agents contextes

La figure 3 montre la structure d'un agent contexte particulier à sa création (3.a) et à la fin d'une phase démonstration (3.b). Chaque ligne correspond à la structure d'une plage de validité associée à une observation. Les plages les plus claires correspondent à la zone *valide*, les plus foncées à la zone *validable*. Le curseur blanc représente la valeur courante du signal. À sa création (3.a), l'agent contexte initialise ses plages de validité pour représenter la situation courante et est associé à l'action courante du tuteur. Tant que l'action proposée par cet agent contexte est en adéquation avec celle du tuteur, l'agent contexte adapte ses plages de validité. Les plages de validité à la fin de la démonstration (3.b) sont la résultante d'un processus d'auto-organisation entre les agents contextes. En observant plus précisément l'évolution des plages de validité, on se rend compte que les plages étiquetées *BlueA* et *BlueD* sont plus larges que celles étiquetées *GreenA* et *GreenD*. Cela signifie que ce contexte particulier est bien plus sensible aux variations des données liées à l'artefact vert qu'aux données liées à l'artefact bleu. Ce contexte particulier est valide lorsque l'artefact vert est proche de l'avant du robot et que le robot est face à la zone bleue. Il est donc impliqué dans la phase de l'activité où le robot ramène l'artefact capturé dans la zone bleue.

4. 3 De la démonstration du tuteur naît l'autonomie

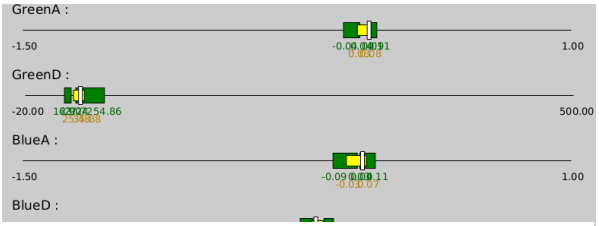

Figure 3.a – Domaine de validité d'un agent contexte à sa création

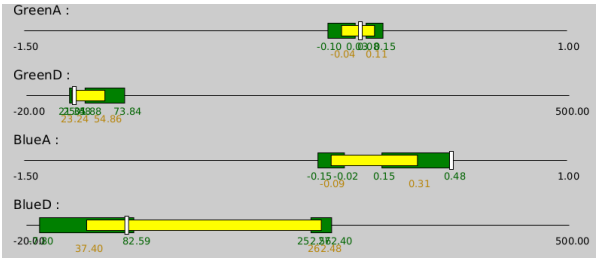

Figure 3.b – Domaine de validité du même agent contexte à la fin de la démonstration

Pour observer la capacité d'Alex à imiter les performances du tuteur, celui-ci réalise une démonstration de cinq minutes au cours de laquelle un certain nombre d'artefacts (nommés *boîtes*) sont collectés. Le nombre de boîtes collectées par le tuteur sert alors de métrique pour observer la capacité d'Alex à proposer des performances semblables à celle du tuteur. Au bout de cinq minutes, les dispositifs agissent de manière autonome, c'est-à-dire que le tuteur n'intervient plus sur leur fonctionnalité. Le nombre de boîtes collectées est calculé toutes les cinq minutes. La figure 4 montre le résultat obtenu au cours d'une de ces expérimentations. Au pire cas, les dispositifs autonomes réalisent la tâche aussi bien que le tuteur : *12* boîtes sont collectées. En moyenne, le système parvient à collecter *14* boîtes. Deux facteurs influencent ce résultat. Le premier vient du caractère aléatoire du déplacement de la boîte, la distance à parcourir pour collecter une boîte variant. L'autre facteur réside dans le fait que le tuteur suit un processus décisionnel plus lent que le système et peut aussi se contredire. Ce phénomène est observable sur la figure 2. À la moitié du chemin entre la position de départ et l'artefact vert, on peut observer un changement de trajectoire. Ce changement est une erreur de

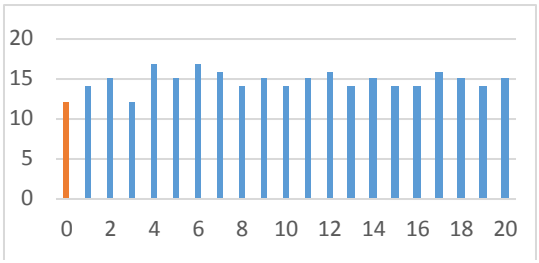

Figure 4 – Nombre de boîtes collectées toutes les 5 minutes. Le pas 0 correspond au score de référence.

télé-opération du tuteur. Les agents contextes créés pour représenter cette situation possèdent une valeur de confiance faible car leur action n'est jamais reproduite par le tuteur. Ainsi, le comportement appris par Alex ne retenant que les propositions d'action des agents les plus confiants, les agents contextes issus de ces erreurs ne seront jamais sélectionnés. Ceci permet à Alex de réaliser la tâche de manière plus efficace.

4. 4 Synthèse

Cette expérimentation permet d'illustrer le mécanisme d'auto-organisation des agents contextes au niveau d'un dispositif robotique qui conduit à l'observation d'un comportement complexe au niveau du robot. Le processus d'auto-organisation est la résultante de l'interaction entre les instances d'Alex et le tuteur. Les expérimentations réalisées à la fois en simulation et sur un robot réel ont permis de mettre en avant des propriétés qui différencient Alex des approches traditionnelles:

Généricité : le processus d'apprentissage est indépendant de la tâche à réaliser et ne nécessite pas de sémantique sur les signaux.

Ouverture : de nouveaux signaux peuvent être dynamiquement intégrés dans le processus d'apprentissage.

Distribution : l'auto-observation permet à chaque dispositif d'être autonome dans son apprentissage tout en restant collaboratif.

Temps réel : l'auto-organisation est réalisée sans interrompre l'activité du système en collaboration avec le tuteur.

5 Conclusion

Le travail présenté traite de l'apprentissage par démonstration en milieu ambiant. Il présente Alex, un système multi-agent conçu pour apprendre un comportement à partir des démonstrations réalisées par un tuteur. Les expériences réalisées montrent la capacité d'Alex à apprendre de l'interaction avec son tuteur. Cela signifie concrètement que la simple démonstration d'une activité permet à chaque système multi-agent associé à chacun des effecteurs du système de comprendre de manière autonome quelles sont les données pertinentes pour imiter collectivement le comportement de son tuteur sans aucune forme de contrôle centralisé.

L'expérimentation montre que deux instances d'Alex (ici deux roues) peuvent coopérer sans communication directe. Le fait que chacune des instances d'Alex corrèle son activité avec les variations de son environnement suffit pour que les deux SMA Alex agissent de manière coopérative. Cet article présente une preuve de concept illustrant qu'un système multi-agent capable d'apprendre à partir de l'auto-observation de son contexte (*context-learning*) est une réponse adéquate à la complexité inhérente aux systèmes ambiants. Cette approche présente un intérêt particulier pour des applications dont la spécification ne peut être complètement définie *a priori*.

Forts de ces résultats, les travaux en cours portent sur l'application de cette technologie dans un contexte industriel. Dans cette perspective et en partenariat avec plusieurs industriels, nous utilisons Alex en robotique collaborative pour le contrôle de bras robotisés dans un contexte industriel, mais aussi pour des applications grand-public.

Enfin, nous désirons à la fois formaliser notre approche pour l'inclure dans une théorie de l'apprentissage multi-agent contextuelle et comparer les performances de notre approche avec d'autres techniques d'apprentissage à la

pointe de l'état de l'art.

Remerciements

Les auteurs souhaitent remercier Sogeti High Tech et l'ANRT pour le soutien apporté à ces travaux.

Références

Argall, Brenna D., Chernova, Sonia, Veloso, Manuela, et al. A survey of robot learning from demonstration. *Robotics and autonomous systems*, vol. 57, no 5, p. 469-483, 2009.

Billard, Aude, et al. "Robot programming by demonstration." *Springer handbook of robotics*, p. 1371-1394, 2008.

Billing, Erik A. et Hellström, Thomas. A formalism for learning from demonstration. *Paladyn*, vol. 1, no 1, p. 1-13, 2010.

Bonjean, Noelie, et al. "ADELFE 2.0." *Handbook on Agent-Oriented Design Processes*. Springer Berlin Heidelberg, p. 19-63, 2014.

Capera, Davy, et al. "The AMAS theory for complex problem solving based on self-organizing cooperative agents." *Proceedings of Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* IEEE, 2003.

Collinot, A., Drogoul, A., & Benhamou, P. Agent oriented design of a soccer robot team. *In Proceedings of the Second International Conference on Multi-Agent Systems*, pp. 41-47, 1999.

Chung, Joohyun. Ambient robotics for meaningful life of the elderly. *Advanced Construction and Building Technology for Society*, 2014.

Ferber, J. *Multi-agent systems: an introduction to distributed artificial intelligence*, Vol. 1, 1999.

Gleizes, Marie-Pierre. Self-adaptive complex systems. *Multi-Agent Systems*. Springer Berlin Heidelberg, p 114-128, 2012.

Guivarch, Valérian, et al. Self-Adaptation of a Learnt Behaviour by Detecting and by Managing User's Implicit Contradictions. *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, vol. 3. IEEE Computer Society, 2014.

Hagras, H., Callaghan, V., Colley, M., Clarke, G., Pounds-Cornish, A., & Duman, H. Creating an ambient-intelligence environment using embedded agents. *Intelligent Systems*, IEEE, vol. 19, no 6, p.12-20, 2004.

Kaminka, G. A. Autonomous agents research in robotics: A report from the trenches. *AAAI Spring Symposium Series*, 2012.

Kearns, M. J., Schapire, R. E., & Sellie, L. M. Toward efficient agnostic learning. *Machine Learning*, vol. 17, no 2-3, p. 115-141, 1994.

Lacouture, Jérôme, et al. Rosace: Agent-based systems for dynamic task allocation in crisis management. *Advances on Practical Applications of Agents and Multi-Agent Systems*. Springer Berlin Heidelberg. p. 255-259, 2012.

Lemouzy, S., Camps, V., & Glize, P. Principles and properties of a mas learning algorithm: A comparison with standard learning algorithms applied to implicit feedback assessment. *Web Intelligence and Intelligent Agent Technology*. IEEE, vol. 2, p. 228-235, 2011.

Makkonen Jarmo, Avdouevski Ivan, Kerminen Riitta and Visa Ari. Context Awareness. *Human-Computer Interaction*, Inaki Maurtua (Ed.), ISBN: 978-953-307-022-3, InTech, DOI: 10.5772/7743, 2009.

Panait, L., & Luke, S. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, vol. 11, no 3, 387-434, 2005.

Parker, Lynne E. Current state of the art in distributed autonomous mobile robotics. *Distributed Autonomous Robotic Systems 4*. Springer Japan, p. 3-12, 2000.

Picard, Gauthier, and Marie Pierre Gleizes. Cooperative self-organization to design robust and adaptive collectives. *ICINCO*. 2005.

Russell, S., Norvig, P., & Intelligence, A. A modern approach. *Artificial Intelligence*. Prentice-Hall, Egnlewood Cliffs, vol. 25, 1995

Souliman, A., Joukhadar, A., Alturbeh, H., & Whidborne, J. F. Intelligent Collision Avoidance for Multi Agent Mobile Robots. *In Intelligent Systems for Science and Information*. *Springer International Publishing*, p. 297-315, 2014

Verstaevel, N., Régis C., Guivarch, V., Gleizes, M.P., Robert, F. Extreme Sensitive Robotic: A Context-Aware Ubiquitous Learning. *Proceedings of the 2015 International Conference on Agents and Artificial Intelligence*, vol. 1, p. 242-248, 2015.