THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *30 juin 2016* **par :**

Nicolas Verstaevel

Self-organization of robotic devices through demonstrations

JURY

Marie-Pierre GLEIZES	Professeur, Université de Toulouse	Directrice
Fabrice Robert	Responsable Innovation, Sogeti High Tech	Co-Encadrant
Christine RÉGIS	Maître de conférences, Université de Toulouse	Co-Encadrante
Stéphane DONCIEUX	Professeur, Université Pierre et Marie Curie	Rapporteur
Jean-Paul JAMONT	Maître de conférences HDR, Université Grenoble Alpes	Rapporteur
Patrick REIGNIER	Professeur, Institut National Polytechnique de Grenoble	Examinateur
Philippe RAVIX	Directeur Innovation, Sogeti High Tech	Invité

École doctorale et spécialité :
MITT : Domaine STIC : Intelligence Artificielle
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse
Directrices de Thèse :
Marie-Pierre GLEIZES et *Christine RÉGIS*
Rapporteurs :
Stéphane DONCIEUX et *Jean-Paul JAMONT*

THESIS

presented at

Université Paul Sabatier - Toulouse III

U.F.R. MATHÉMATIQUES, INFORMATIQUE ET GESTION

to obtain the title of

DOCTEUR DE L'UNIVERSITÉ DE TOULOUSE

delivered by

UNIVERSITÉ PAUL SABATIER - TOULOUSE III

Mention INTELLIGENCE ARTIFICIELLE

by

NICOLAS VERSTAEVEL

Doctoral school:	Informatique et Télécommunication
Laboratory:	Institut de Recherche en Informatique de Toulouse
Team:	Systèmes Multi-Agents Coopératifs

Self-Organization of Robotic Devices Through Demonstrations

JURY

Marie-Pierre GLEIZES	*Professor, Université de Toulouse*	(Supervisor)
Fabrice ROBERT	*R&D and Innovation Manager, Sogeti High Tech*	(Co-Supervisor)
Christine RÉGIS	*Assistant Professor, Université de Toulouse*	(Co-Supervisor)
Stéphane DONCIEUX	*Professor, Université Pierre et Marie Curie*	(Examiner)
Jean-Paul JAMONT	*Assistant Professor, Université Grenoble Alpes*	(Examiner)
Patrick REIGNIER	*Professor, Grenoble INP*	(Supervisor)
Philippe RAVIX	*R&D Director, Sogeti High Tech*	(Guest)

Nicolas Verstaevel

Auto-Organisation de Dispositifs Robotiques par Démonstrations

Directrice de thèse :	Marie-Pierre Gleizes, Professeur, Université de Toulouse
Co-Directeurs :	Christine Régis, Maître de conférences, Université de Toulouse
	Fabrice Robert, Responsable Innovation, Sogeti High Tech

Résumé

La théorie des AMAS (Adaptive Multi-Agent Systems) propose de résoudre des problèmes complexes par auto-organisation pour lesquels aucune solution algorithmique n'est connue. Le comportement auto-organisateur des agents coopératifs permet au système de s'adapter à un environnement dynamique pour maintenir le système dans un état de fonctionnement adéquat. Dans cette thèse, cette approche a été appliquée au contrôle dans les systèmes ambiants, et plus particulièrement à la robotique de service.

En effet, la robotique de service tend de plus en plus à s'intégrer à des environnements ambiants, on parle alors de robotique ambiante. Les systèmes ambiants présentent des caractéristiques, telles que l'ouverture et l'hétérogénéité, qui rendent la tâche de contrôle particulièrement complexe. Cette complexité est accrue si l'on prend en compte les besoins spécifiques, changeants et parfois contradictoires des utilisateurs. Les travaux de cette thèse proposent d'utiliser les principes de l'auto-organisation, pour concevoir un système multi-agent capable d'apprendre en temps réel à contrôler un système à partir des démonstrations faites par un tuteur. C'est l'apprentissage par démonstration. En observant l'activité de l'utilisateur et en apprenant le contexte dans lequel l'utilisateur agit, le système apprend une politique de contrôle pour satisfaire les utilisateurs.

Nous proposons un nouveau paradigme de conception des systèmes robotiques sous le nom d'*Extreme Sensitive Robotics*. L'idée de base de ce paradigme est de distribuer le contrôle au sein des différentes fonctionnalités qui composent un système et de doter chacune de ces fonctionnalités de la capacité à s'adapter de manière autonome à son environnement. Pour évaluer l'apport de ce paradigme, nous avons conçu ALEX (Adaptive Learner by EXperiments), un système multi-agent adaptatif dont la fonction est d'apprendre, en milieux ambiants, à contrôler un dispositif robotique à partir de démonstrations. L'approche par AMAS permet la conception de logiciels à fonctionnalités émergentes. La solution à un problème émerge des interactions coopératives entre un ensemble d'agents autonomes, chaque agent ne possédant qu'une vue partielle de l'environnement. L'application de cette approche nous conduit à isoler les différents agents impliqués dans le problème du contrôle et à décrire leurs comportements locaux. Ensuite, nous identifions un ensemble de situations de non coopération susceptibles de nuire à leurs comportements et proposons un ensemble de mécanismes pour résoudre et anticiper ces situations. Les différentes expérimentations ont montré la capacité du système à apprendre en temps réel à partir de l'observation de l'activité de l'utilisateur et ont mis en évidence les apports, les limitations et les perspectives offertes par notre approche à la problématique du contrôle de systèmes ambiants.

Nicolas Verstaevel

SELF-ORGANIZATION OF ROBOTIC DEVICES THROUGH DEMONSTRATIONS

Supervisor: Marie-Pierre Gleizes, Professor, Université de Toulouse
Co-Supervisors: Christine Régis, Assistant Professor, Université de Toulouse
Fabrice Robert, Innovation Manager, Sogeti High Tech

Abstract

The AMAS (Adaptive Multi-Agent Systems) theory proposes to solve complex problems for which there is no known algorithmic solution by self-organization. The self-organizing behaviour of the cooperative agents enables the system to self-adapt to a dynamical environment to maintain the system in a functionality adequate state. In this thesis, we apply the theory to the problematic of control in ambient systems, and more particularly to service robotics.

Service robotics is more and more taking part in ambient environment, we talk of ambient robotics. Ambient systems have challenging characteristics, such as openness and heterogeneity, which make the task of control particularly complex. This complexity is increased if we take into account the specific, changing and often contradictory needs of users. This thesis proposes to use the principle of self-organization to design a multi-agent system with the ability to learn in real-time to control a robotic device from demonstrations made by a tutor. We then talk of learning from demonstrations. By observing the activity of the users, and learning the context in which they act, the system learns a control policy allowing to satisfy users.

Firstly, we propose a new paradigm to design robotic systems under the name *Extreme Sensitive Robotics*. The main proposal of this paradigm is to distribute the control inside the different functionalities which compose a system, and to give to each functionality the capacity to self-adapt to its environment.

To evaluate the benefits of this paradigm, we designed ALEX (Adaptive Learner by Experiments), an Adaptive Multi-Agent System which learns to control a robotic device from demonstrations. The AMAS approach enables the design of software with emergent functionalities. The solution to a problem emerges from the cooperative interactions between a set of autonomous agents, each agent having only a partial perception of its environment. The application of this approach implies to isolate the different agents involved in the problem of control and to describe their local behaviour. Then, we identify a set of non-cooperative situations susceptible to disturb their normal behaviour, and propose a set of cooperation mechanisms to handle them.

The different experimentations have shown the capacity of our system to learn in real-time from the observation of the activity of the user and have enable to highlight the benefits, limitations and perspectives offered by our approach to the problematic of control in ambient systems.

To infinity and beyond...
Vers l'infini et au delà...

QUELLE aventure ! Parfois véritable chemin de croix, où tel un *Sisyphe* des temps modernes, il nous faut faire preuve d'une dévotion sans faille pour franchir les obstacles inhérents à tout travail scientifique, cette aventure tient souvent place d'*agora*, faite d'échanges et de rencontres, de débats technologiques et philosophiques, échanges réciproques de passions et de raison, transformant un périple en une quête initiatrice d'un profond changement de l'individu qui l'entreprend.

Mais un travail de thèse ne se résume pas à l'aventure d'un seul individu. Dans son mouvement, la thèse entraine tout une dynamique sociale, où elle se nourrit et s'enrichit des multiples interactions avec un environnement composé d'individus toujours plus singuliers. Après tout, que serait l'*agent* sans son *environnement* ?

Alors que ces mots sont les derniers que je couche, il me paraît naturel que ceux-ci le soient pour exprimer toute la reconnaissance que je porte à cet environnement, à sa richesse et sa complexité, notamment à ces individus qui le composent, et à l'importance que furent et que continueront d'être ces interactions, véritable vivier duquel ce manuscrit est peut être le résultat le plus facilement observable, mais n'en est pas la seule expression, tant ces trois années ont profondément changé l'homme que je suis devenu, et qui lira demain ces lignes avec la nostalgie d'un sincère amour.

Face à la complexité de cet environnement, il m'est impossible d'être exhaustif dans ces remerciements. Il me faudrait dresser la liste de toutes ces rencontres qui m'ont influencé, consciemment ou non, et ce, du petit Nicolas, au (presque) docteur d'aujourd'hui. Face à cette immensité, un mot, simple et sincère : "Merci", que je ne peux m'empêcher ici de décliner.

Aussi je tiens à remercier Stéphane Doncieux et Jean-Paul Jamont d'avoir pris le temps de lire et de commenter mes travaux. Au moment où je fixe ces mots, c'est avec beaucoup de joie que je me prépare à un moment d'échanges que nous partagerons avec Patrick Reignier que je remercie également.

Merci à Philippe Ravix de m'avoir fait confiance pour mener ces travaux. Sogeti High Tech fût aussi une composante importante de l'environnement de cette thèse et la source de nombreuses rencontres. C'est pourquoi j'adresse aussi un merci à mes collègues de Sogeti High Tech qui m'ont accompagné dans ce voyage.

Un voyage qui aurait put être une *Odyssée*, si ce n'était sans compter sur l'expérience, la patience, la pédagogie et souvent le réconfort que j'ai pu trouver auprès de mes encadrants. Les résultats de ces travaux sont tout autant vôtres. Je tiens tout d'abord à remercier Marie-Pierre Gleizes, qui aura su faire germer en moi la passion des *AMAS* et de m'avoir accordé la confiance et la liberté qui m'ont permis de m'épanouir, ainsi que pour l'énergie et la passion que tu mets chaque jour à oeuvrer pour l'équipe et pour l'université. Vient ensuite Fabrice Robert, dont le pragmatisme m'a amené à tout remettre en question. Je retiendrai nos discussions, parfois houleuses, toujours sincères, qui m'ont tant aidé à la construction de mon discours et m'ont amené, je l'avoue, à voir le monde différemment. Toi aussi, tu m'as fait confiance et je t'en remercie. Merci à toi Christine Régis, véritable phare dans l'océan des doutes que peut parfois être la thèse. J'admire ta quiétude qui m'a tant de fois réconforté. Tout ce voyage me m'aurait paru bien trouble sans tes conseils, toujours bienveillants, et ce sourire que tu as toujours arboré. Enfin, merci à toi, homme de l'ombre dont je tairais le nom

puisque tu n'aimes pas les honneurs. Ce fut un plaisir de débattre et d'échanger avec toi. Ton énergie et ta passion dévorante pour les AMAS est contagieuse, et en grand patriarche des AMAS, tu sais nous guider, nous interroger, et souvent nous bousculer et ce, afin que l'on puisse toujours offrir le meilleur de nous-même. Cette thèse aurait été parfaite s'il ne m'avait manqué quelques notes d'harmonica...

Évidemment, je ne vous oublie pas, vous les SMACkers. Permanents, docteurs, doctorants, stagiaires, vous êtes si nombreux qu'il me faudrait encore 180 pages pour dire tout le bien (et le mal) que vous méritez. Soyez sûrs que du sens des mots, aux "oui mais", en passant par les poneys et les idéaux révolutionnaires, j'emporte un peu de chacun, et beaucoup de tous. Mon seul regret (mais ainsi va la vie) est de ne pas pouvoir passer encore un peu de temps à vos côtés.

Alors qu'approche la fin de ces remerciements, j'ai une pensée particulière pour ma famille. Dans la pudeur qui me caractérise, je n'exprime que peu ce que je ressens, mais dans votre cas, les mots sont bien pauvres pour exprimer ma gratitude.

Et enfin à toi Flora, à qui il me faudra emprunter les mots d'un autre pour dire tout mon amour :

Parmi les étoiles admirées, mouillées
Par des fleuves différents et par la rosée,
J'ai seulement choisi l'étoile que j'aimais
et depuis ce temps-là je dors avec la nuit.

Parmi les vagues, une vague, une autre vague,
vague de verte mer, branche verte, froid vert,
j'ai seulement choisi l'unique et seule vague
et c'est la vague indivisible de ton corps.

Vers moi toutes les gouttes toutes les racines
et tous les fils de la lumière sont venus.

Je n'ai voulu que ta chevelure pour moi.
Et de toutes les offrandes de la patrie
Je n'ai choisi que celle de ton coeur sage.

[Pablo Neruda, La Centaine d'amour]

Nicolas.

Contents

General Introduction **1**
Contribution of the Thesis 1
Manuscript Organization 2

I Thesis context **5**

1 Ambient Robotics **7**
1.1 Ambient Systems 7
1.2 Control Theory 7
1.2.1 Dynamical Systems 8
1.2.2 Control Loops 8
1.3 Control in Ambient Systems 10
1.4 Towards Ambient Robotics 10
1.5 The Integrator Problem 11
1.6 Extreme Sensitive Robotics 12
1.7 Thesis Objectives 15

II State of the art **17**

2 Learning a Control Policy **19**
2.1 Behaviourism - Mind is a *Black-box* 21
2.1.1 Reinforcement Learning 23
2.1.1.1 Q-Learning 23
2.1.1.2 SARSA 23
2.1.2 Analysis 24
2.2 Cognitivism - Opening the *Black-box* 24
2.3 Connectionism 25

2.3.1 Artificial Neural Networks 26
2.3.1.1 Perceptron 26
2.3.1.2 Multilayer Perceptron 27
2.3.1.3 Learning through backpropagation 28
2.3.2 Self-Organizing Maps 28
2.3.3 Deep Learning 29
2.3.4 Analysis 30
2.4 Evolutionism 30
2.4.1 Genetic Algorithms 31
2.4.2 Analysis 32
2.5 Constructivism 33
2.5.1 Schema-Based Learning 35
2.5.1.1 Initial formalisation 35
2.5.1.2 Evolutions 36
2.5.2 Analysis 37
2.6 Other and Hybrid Approaches 37
2.6.1 Learning Classifier Systems 37
2.6.2 Case-Based Learning 39
2.7 Synthesis 40

3 Learning from Demonstration **41**
3.1 Imitation: From Natural to Artificial 42
3.1.1 Imitation in Nature 42
3.1.2 Defining Imitation 42
3.1.3 Big Five Questions 44
3.1.4 Imitation in Artificial Systems : Motivations 45
3.2 Imitation as a Learning Paradigm: Learning from Demonstration 45
3.2.1 Problem Statement 45
3.2.2 The Correspondence Problem 46
3.2.3 Designing Imitation 47
3.2.3.1 Gathering the Examples 47
3.2.3.2 Deriving a Policy 48
3.3 Some Applications of the LfD Paradigm 51
3.4 Imitation Learning and Ubiquitous Systems: Requirements and Proposals . . 52

III Contribution **55**

4 Designing Emergence **57**

4.1 Emerging Phenomena . . . 57
4.2 Multi-Agent Systems . . . 59
4.2.1 The Agent . . . 59
4.2.2 The Environment . . . 61
4.2.3 Properties of Multi-Agent Systems . . . 62
4.2.4 self-organization in MAS . . . 62
4.3 Designing the Emergence: the AMAS Approach . . . 63
4.3.1 Interaction and Cooperation . . . 63
4.3.2 Functional Adequacy . . . 64
4.3.3 Adapting the System Trough its Parts . . . 64
4.3.3.1 Non Cooperative Situations . . . 65
4.3.4 The ADELFE Methodology . . . 66
4.3.5 AMAS Applications . . . 66
4.4 Control and Learning with an AMAS . . . 67

5 ALEX, Show Me And I Learn **69**

5.1 Objectives . . . 69
5.2 Environment . . . 70
5.3 Nominal Behaviours . . . 71
5.3.1 ALEX Nominal Behaviour . . . 71
5.3.1.1 Observing the Environment . . . 72
5.3.1.2 Analysing the Environment State . . . 72
5.3.1.3 Applying the Adequate Control Action . . . 72
5.3.1.4 Synthesis . . . 72
5.3.2 Agents Nominal Behaviours . . . 73
5.3.2.1 Percept Agent Nominal Behaviour . . . 73
5.3.2.2 Context Agent Nominal Behaviour . . . 74
5.3.2.3 Input Agent Nominal Behaviour . . . 75
5.3.2.4 Synthesis . . . 76
5.4 Non Cooperative Situations . . . 77
5.4.1 Introducing the Tutor . . . 77
5.4.2 NCS 1: Incompetence of the Input Agent . . . 78
5.4.2.1 NCS 1.a: Action Available . . . 78

5.4.2.2 NCS 1.b: No Action Available 78
5.4.3 NCS 2: Conflict of the Input Agent 78
5.4.4 NCS 3: Conflict Between Context Agents and the Input Agent 79
5.4.4.1 NCS 3.a: Action Available 79
5.4.4.2 NCS 3.b: No Action Available 79
5.4.5 NCS 4: Concurrence of a Context Agent 80
5.4.5.1 NCS 4.a: Action Available 80
5.4.5.2 NCS 4.b: No Action Available 80
5.4.6 NCS 5: Incompetence of a Context Agent 80
5.4.7 NCS 6: Uselessness of a Context Agent 81
5.4.8 NCS 7: Uselessness of a Percept Agent 81
5.4.9 NCS 8: Incompetence of a Percept Agent 82
5.4.10 Synthesis 82
5.5 Implementation 84
5.5.1 Adaptive Value Trackers 84
5.5.2 Adaptive Value Range Trackers 85
5.5.3 Validity Ranges 87
5.5.4 Confidence Value 89
5.5.5 Context Agent Initialisation 91
5.5.6 Percept Agent Neighbourhood 91
5.5.7 Agent Scheduling 92
5.6 Algorithms 92
5.6.1 Percept Agent Behaviour 93
5.6.2 Input Agent Behaviour 93
5.6.3 Context Agent Behaviour 93
5.7 ALEX Differences with Other Approaches 95
5.7.1 Differences with Learning Classifiers 95
5.7.2 Differences with Cased-Based Reasoning 95
5.7.3 Differences with Artificial Neural Networks 96
5.7.4 Differences with Schema-Learning 96
5.8 Synthesis 96
5.8.1 Self-Organization and ALEX 97
5.8.2 And What About the Emergence? 98

6 Experimentations **99**

6.1 The Mountain Car Problem . . . 100
6.1.1 Problem Description . . . 100
6.1.2 Motivations and Objectives . . . 101
6.1.3 Experimental Process . . . 102
6.1.3.1 Introducing the Tutors . . . 102
6.1.3.2 Experiment Implementation . . . 103
6.1.4 Analysis . . . 104
6.1.4.1 Learning from a Virtual Tutor . . . 104
6.1.4.2 Learning from a Human Tutor . . . 107
6.1.5 Synthesis of the Experiment . . . 109
6.2 Teach Robots Yourselves Experiment . . . 111
6.2.1 General Description . . . 111
6.2.2 Motivations and Objectives . . . 112
6.2.3 Implementation Details . . . 113
6.2.4 Analysis . . . 114
6.2.4.1 First Experiment . . . 114
6.2.4.2 Second Experiment . . . 117
6.2.5 Synthesis of the Experiments . . . 118
6.3 Incremental Design Experiment . . . 120
6.3.1 Description of the Case Study . . . 121
6.3.2 The Case Study: Iterative Design Process . . . 122
6.3.2.1 Initial Architecture . . . 122
6.3.2.2 First Experiment: Navigation with a Distance Sensor . . . 123
6.3.2.3 Second Experiment: Adding Wheel Speed as an Observation 124
6.3.2.4 Third Experiment: Adding a Camera . . . 125
6.3.2.5 Final Experiment: Adding a Motor to the Door . . . 127
6.3.3 Synthesis of the Experiments . . . 128
6.4 Discussion on ALEX Usage in the Context of Industry 4.0 . . . 129
6.4.1 Requirements for Using ALEX . . . 129
6.4.2 ALEX and the Collaborative Robotic Arm : Proposals and Open Questions . . . 130
6.4.2.1 What Are the Actions? . . . 130
6.4.2.2 What to Observe? . . . 131
6.4.2.3 Who is the Tutor? . . . 132
6.4.3 Conclusion . . . 133

6.5 General Synthesis 133
6.5.1 Compliance with the Criteria 133
6.5.1.1 Task Independancy Criterion 133
6.5.1.2 User-Centered Criterion 134
6.5.1.3 On-Line Criterion 134
6.5.1.4 Openness 135
6.5.2 Discussions 135
6.5.2.1 Instantiation Cost and Parametrization 135
6.5.2.2 The Scalability 136
6.5.2.3 The Noise 137

IV Conclusion & Perspectives **139**

7 Conclusion and Perspectives **141**
General Conclusion 141
Contribution 143
Contribution to Robotics 143
Contribution to Machine Learning 143
Contribution to AMAS 144
Perspectives 144
Self-Observation 144
The Context-Learning Pattern 145
The Scientific Lock of Non Finality 146

Own Bibliography **149**

Bibliography **149**

List of Figures **161**

General Introduction

SINCE our very first breath, and until our last, at every moment of our life, we learn. During all our life, we interact with the world surrounding us, and, consciously or not, learn from those interactions. This capacity to learn is natural and allows each individual to adapt itself to its environment. While it is so natural, and often unconscious, for humans to constantly learn and adapt to their environment, transferring this capacity to learn to robots is a particularly complex task. However, this capacity is more and more becoming a requirement. Indeed, the rapid increase of hardware capacities, and their decreasing costs, have enable to populate our environment with heterogeneous intelligent robotic devices. Artificial Intelligence, once restrained in personal computers, is now distributed in our environment making those environments truly *ambient*. Those devices may adopt various forms, such as intelligent flower pots that follow lights, to more complex robots, such as humanoids, composed of many sensors and motors. However, the expectations generated by these devices are the same: we expect from them that they meet with our specific needs, facilitating our daily life. A such objective involves that all the devices, which are physically distributed in the environment, collaborate to satisfy us.

Those systems are truly complex: they are composed of a potentially huge number of devices, each device having an autonomous activity and providing services to multiple users, each user having its own and specific needs. Those factors make the *ad hoc* design of a controller supervising the activity of such system more and more prohibitive. A challenge and the main motivation of this thesis is then to give to each device the ability to learn and adapt to its environment and to human activity.

Contribution of the Thesis

In this thesis, we explore the problematic of control in *ambient systems*, with a particular interest for applications to service robotics. A first contribution is made with the *Extreme Sensitive Robotic* paradigm, an integrative approach of autonomous robotic components where each component dynamically learns its own control policy. This approach relies on a local self-adaptation of each robotic component to the environment and its users.

To enable each component to self-adapt to the specific and changing needs of its users, we propose to use the *Learning From Demonstration* paradigm. However, the usage of the *Learning From Demonstration* in the context of ambient systems implies to design an innovative learning approach to face the complexity inherent to these systems.

By applying the Adaptive Multi-Agent System approach, we designed ALEX (*Adaptive Learner by EXperiments*), an Adaptive Multi-Agent System designed to learn to control a robotic device from demonstrations in the context of ambient systems. The Adaptive Multi-Agent System approach is an approach to design systems with emergent functionalities. The solution to a problem emerges from the cooperative interactions between a set of agents, which only possess a partial view of the environment. In accordance with the Adaptive Multi-Agent Systems approach, we isolate the different agents involved in the problem and describe their local behaviour. Then, we identify a set of Non-Cooperative Situations (NCSs) susceptible to disturb their normal behaviour, and propose a set of cooperation mechanisms to handle them.

We evaluate our approach on a set of experiments, each experiment testing different hypothesis. From the analysis of the result, we highlight the advantages and limitations of our approach and point out the perspective offered by this work.

Manuscript Organization

This manuscript is structured as follows:

- ▷ Chapter 1: In this chapter, the **context** of the work and **motivations** are introduced: the problematic of control in ambient robotics is presented to formulate the problematic of the integration of robotic components. In a second part, we propose the *Extreme Sensitive Robotic* paradigm as a new paradigm to face those challenges and we outline its requirements. With this approach, each component is seen as autonomous and has to locally learn its own control policy. Lastly, the thesis objectives are exposed.
- ▷ Chapter 2: With this chapter, we explore the different approaches and hypothesis of the machine learning field, seeking for an approach with the capacity to **learn a control policy**. Such approach has to respect the requirements of the *Extreme Sensitive Robotics*. In order to do so, we explore the notion of learning in artificial systems, and present the main learning families. From this exploration we outline the proposal to learn from interactions with human.
- ▷ Chapter 3: This chapter describes the **Learning from Demonstration** paradigm, a paradigm which proposes to learn a policy from the demonstration performed by a human. In a first part, we highlight the inspirations and motivations of the Learning from Demonstration paradigm. Then, we present its usage from an engineering perspective, presenting the state of the art approaches. At last, we draw the requirement for enabling *Learning From Demonstration* in Ambient Robotics.
- ▷ Chapter 4: This chapter introduces the key concepts of emergence and self-organization to face complexity. It presents the Multi-Agent paradigm and the **Adaptive Multi-Agent Systems** approach. This chapter shows the adequacy of the AMAS approach for our application.
- ▷ Chapter 5: The chapter 5 describes our main contribution: **ALEX** (*Adaptive Learner by EXperiments*), an Adaptive Multi-Agent System designed to learn to control a robotic device from demonstrations. Its design and the behaviours of its agents are explained and the chapter concludes with a positioning in regards with other approaches.

▷ Chapter 6: This last chapter discusses of the **evaluations** of ALEX in three different experimentations. For each experimentation, we present the hypothesis to test, the motivations and we discuss the results we obtained. This chapter ends with a general synthesis pointing out ALEX properties and limitations.

Self-Organization of Robotic Devices Through Demonstrations

Thesis context

1 Ambient Robotics

In this chapter, the context of the work and motivations are introduced: the problematic of control in ambient robotics is presented to formulate the problematic of the integration of robotic components. In a second part, we propose the Extreme Sensitive Robotic paradigm as a new paradigm to face those challenges and we outline its requirements. With this approach, each component is seen as autonomous and has to locally learn its own control policy. Lastly, the thesis objectives are exposed.

1.1 Ambient Systems

We are living at a time where technologies evolve every day becoming cheaper and more powerful. As an illustration of such evolution, the price of a Raspberry Pi (a nano-computer) as switched from 25$ (Model A+) in 2012 to 5$ (Raspberry Pi Zero) in 2015. Intelligence, once restrained in personal computers, is now distributed in our environments under many forms encouraged by the *Do it yourself* revolution. Anyone can now afford to buy electronic components to equip and transform its environment. Those systems are *ambient* and work in concert to support people in carrying out their everyday life [Ramos et al., 2008]. Internet of things, wearable sensors, robotics, home automation are illustrations of the ubiquitous computing revolution [Weiser, 1991]. As software and hardware become ever more elaborated, intelligence is now embedded in objects. Those devices can recognize the situational context of users and provide adequate and personalised services in a transparent way. A consequence is that we now have at our disposal libraries of various components offering different kind of services. For example, smart cameras can produce data from image recognition algorithms or every day objects can play a role in the human-system interaction. Each of those components is autonomous and designed independently. Immersed in the environment, they have to work in collaboration to pro-actively satisfy users. The design of an intelligent system is then a matter of *integration* and a recurrent challenge is how to *control* all those intelligent things to collaborate to provide services to its users whereas they have an autonomous activity.

1.2 Control Theory

Control is a notion studied in many domains of application from mathematics to biology. From the most general point of view, the usual objective of control theory is to control dynamical systems refereed as *plants*. In this section, we present the general concepts of control theory and its application to ambient systems.

1.2.1 Dynamical Systems

Control theory sees a system by its functional aspect as a set of *inputs* on which a control can be applied through *effectors*, and a set of *outputs*, which can be measured through *sensors*. It focuses on the information flow from inputs to outputs and the processes of transformation that occurs between these two. A system with only one input and one output is a *SISO* system (Simple-Input and Simple-Output), and a system with several inputs and outputs is a *MIMO* (Multiple-Input and Multiple-Output) system.

The concept of *evolution in time* is central to the theory of dynamical systems. A system is said to have a linear evolution if it respects the *superposition property* which states that the net response at a given place and time caused by two or more stimuli is the sum of the responses which would have been caused by each stimulus individually. Systems with this property are governed by a function f that takes as argument of the system state x (the value of inputs and outputs) and the control inputs u:

$$\tfrac{dx}{dt} = f(x, u)$$

The system is said to be *linear*. Such system is relatively simple to control due to the absence of relative minimum on f.

Systems *affine in the control* evolve linearly regarding to the control input and non-linearly regarding to their current state [Sontag and Boyd, 1995]:

$$\tfrac{dx}{dt} = f(x) + g(x)u$$

where f and g are potentially non-linear functions. While being harder than linear system to control, the usage of mathematical methods of linearisation eases their control.

On contrary, *non-linear* systems do not follow the principle of superposition. Their evolution is not determined only by their current state and control inputs. Non linearity is one of the criterion defining the class of complex systems. In such systems, some mechanisms can be difficult or impossible to identify *a priori* and some parts of the system or the environment can be uncertain or inaccessible. Providing explanations or anticipation to non-linear events requires complex modelling and several hypothesis on the system and its environment.

1.2.2 Control Loops

The role of a *controller* is to transform an input *reference* into a *control signal*. The input reference corresponds to the desired output value of the *process* (the system to control) and the control signal corresponds to the action to perform on the controller to produce the adequate output.

The easiest way to control a process is to perform an *open-loop* control (Figure 1.1). The controller directly transforms an input reference into a control signal which is sent to the

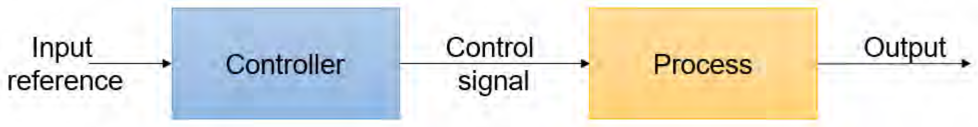

Figure 1.1 — Open-loop control.

process. A simple example of open-loop control is a TV remote control. When the user push a button (the input reference) on the remote, a control signal (basically an infra-red signal) is sent to the TV (which acts as the process) and the channel (the output value) is switched.

It results in a generic definition of *control* : the control of a system is the adequate adjustment of its inputs in accordance with the desired reference output. However, with an open-loop control, the controller does not have information about the process. For example, when you adjust the timer of your microwave to defrost food, the controller has no clue if your food is really unfrozen at the end of the timer and the microwave will stop itself no matter if your food is still frozen or not. The behaviour of the controlled system is influenced by the controller but the controller behaviour is independent of the controlled system.

To enable a system to adjust its control, the cybernetics (a theory of communication and control focusing on information flows) defined the notion of **feedback** [Norbert, 1948] as the action produced by a phenomenon routed back on its own causes. The system can then be said to feed back into itself as its outputs are routed back as inputs. A feedback is said *positive* if it increases a phenomenon, and *negative* otherwise. In closed-loop control, a feedback is added to evaluate the error to the desired objective. The controller is influenced by the difference between the desired output and the one observed by a sensor (Figure 1.2). Then, the behaviour of the controlled system influences the behaviour of the controller and reciprocally. The two systems are *coupled*.

The coupling between the controller and the process has been studied by the cybernetics. The term *variety* was introduced by W. Ross Ashby to denote the count of the total number of states of a system [Ashby et al., 1956]. This term is analogous to complexity: the more the system's variety is rich, the more complex the system is. The *law of Requisite Variety* states that *"if a system is to be stable, the number of states of its control mechanism must be greater than or equal to the number of states in the system being controlled"*. A controller has to drive and

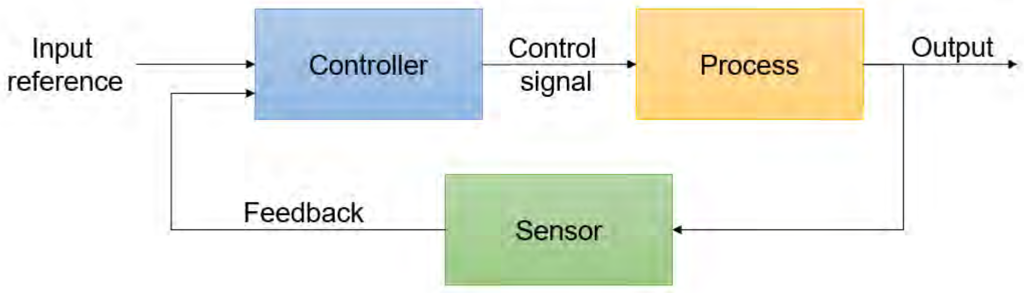

Figure 1.2 — Closed-loop control.

maintain a system into a desired state. In order to do so, each state of the controlled system needs to correspond to a state of the controller able to deal with it. If such condition is not respected, the controller could find itself in novel situations where it is unable to control the process, leading to an inversion of control. The law of requisite variety illustrates the complexity to design *ad hoc* controller in complex systems.

1.3 Control in Ambient Systems

Ambient systems have properties that make them particularly difficult to control. According to the definition of [Russell and Norvig, 1995], the environments of ambient systems are:

- **Inaccessible**: each device composing the system has a partial observation of the environment.
- **Continuous**: considering applications in the real world, the number of observations and actions is not discrete.
- **Non-deterministic**: consequences of performed actions in the real world could not be determined in advance with certainty.
- **Dynamic**: system's actions, user activity, appearance and disappearance of devices may change the environment.

Ambient systems are systems thought to be tailored to their users' needs. However, users have specific needs that may change, sometimes contradictorily. Then, **adaptivity** to users needs is a fundamental requirement of ambient systems: they have to change their behaviour to continuously satisfy their users.

Ambient systems are **open**, devices can appear and disappear, adding or removing sensors and effectors. To not affect user satisfaction, those appearance and disappearance must be treated at runtime.

Designing an *ad hoc* controller of an ambient system supervising the whole activity involves having a lot of knowledge on the system dynamics, including the anticipation of user needs and the appearance or disappearance of devices. As ambient systems are highly dynamic, any unanticipated change in their composition implies re-performing the whole design process meaning that sustainability of such systems is a challenging task.

1.4 Towards Ambient Robotics

The field of robotics has not escaped the ambient revolution. Since their beginning, robots become ever more elaborated both in terms of hardware and software. It may be considered that those systems are truly complex, complexity increased by their necessary adaptation to the high dynamics of their environment (including humans) and a dynamical coordination with other robots or artificial ambient systems. A robot for a particular application consists in the aggregation of the necessary components to satisfy its objectives. Those components are no longer restricted to be set up in the robot body but could also be distributed in its environment. The collective of components has to interact and collaborate to perform an adequate global activity.

The development of service robots is an upcoming area in robotic research with the idea to design robotic components providing services to users rather than mechanical automata achieving a particular task. Service robotics differs from its industrial version where robots are retained in steel cages in factories to perform automatic manipulation tasks. Those cages are both a protection for humans evolving inside factories and a way to ensure the stability of the robot environment. On contrary, service robotics deals with robot evolving in our every day life, among humans. In factories, Robots now become CoBots (Collaborative roBots), which means that robots are intended to physically interact with humans in a shared workspace, providing assistance and services to workers in a natural and transparent way. Those robots are parts of an ambient system providing a set of functionalities (such as mobility or prehension) and perceptions (using real-time image analysis, embedded sensors,...), working in collaboration with other robots, electronic devices and humans.

The emergence of those *cyber-physical systems* into factories is seen as the fourth industrial revolution, tagged with the term *Industry 4.0* [Jazdi, 2014]. The first industrial revolution was a period from 1760 to 1840 that mobilised the mechanization of production using water and steam power. The second industrial revolution was a period from the final third of the 19th century to the beginning of the 20th which introduced mass production with the help of electric power and division of labor. The third revolution was the digital revolution with the use of electronics and IT to further automate production. The fourth revolution will be the massive usage of cyber-physical systems, Internet of Things and Big Data. The expectations of this revolution are as great as its challenges are complicated. Decentralization, modularity, real-time capabilities and service orientation are some of these challenge.

This shift from robot to ambient robotics leads to a radical change in the way we design and control those robots putting users in the center of the design process. On this thesis, we propose to consider robots as part of ambient systems and to apply the same hypothesis than the one applied to ambient systems. We take the perspective of a robotic integrator. Robotic integrators are companies that analyse your robotic system needs, provide a plan for automation, and put the automation into production. They traditionally follow a design process that goes from the expression of needs to the design of the solution. However, the ambient paradigm puts these designers face to a problem we named *the integrator problem*.

1.5 The Integrator Problem

The evolution of technologies, both in terms of hardware and software, makes available libraries of various components realizing functions rather than objectives. Internet of things [Perera et al., 2014] is the perfect illustration of such a possibility. Designers of those systems have to aggregate different functions to build a system providing services to its users. Those functions are provided by electronic devices (basically by effectors) and each device controls

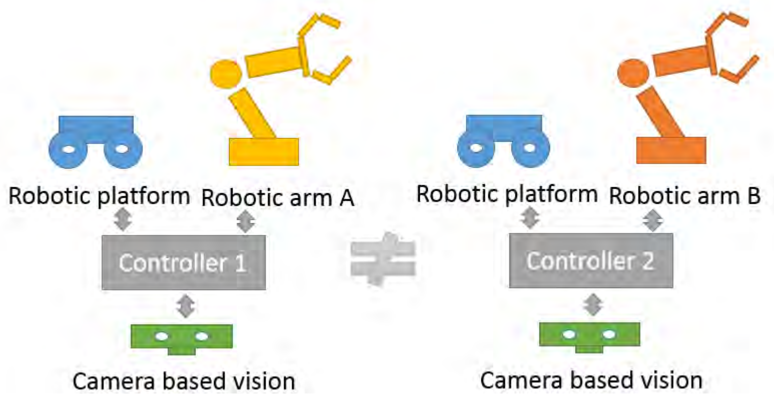

Figure 1.3 — The integrator problem. Each modification in the needs or the system's composition involves re-performing the whole design process

a particular functionality. For example, a particular device can control the activation of an electric shutter and another one can control lights. Robots are part of those systems and propose a set of functionality, which can include mobility. A mobile robotic platform equipped with a robotic arm then provides two functionalities: the ability to move and the ability to grab objects. The integration of those different robotic components in order to provide service to humans is a complex task. Let's consider the case of a designer who wants to integrate a robotic arm from one constructor and a mobile platform from another constructor, while using image processing algorithms from a third provider to perform a collecting task. The design of an *ad hoc* controller for such application is complex and requires a lot of expertise on each component, but also on its environment (which includes human activity). If for any reason, a component is replaced with another one (even if this new component provides the same functionality), the whole design process has to be performed again. This is time greedy and involves high cost of maintenance and evolution. However, the same system composition (one robotic arm, one camera based vision and a robotic platform) could be used in different kind of applications. For example, one could want to use it for turning valves in a factory or the other for cleaning a room in a nuclear power plant. Each new application involves designing a new controller (Figure 1.3).

The main motivation of this thesis is to propose to designers to profit from a system capable of self-adapting to both the environment and users' needs without requiring reprogramming any system's component. This is the postulate made by *Extreme Sensitive Robotics*.

1.6 Extreme Sensitive Robotics

The Extreme Sensitive Robotic paradigm [Verstaevel et al., 2015], studied in this thesis, proposes a change in the way we design robots by considering that a robot is the aggregation of the necessary functions to satisfy user's needs, but a group of robots has to be considered exactly in the same way: a set of macro-functions (each robot) working in coordination. An Extreme Sensitive Robot is made of simpler Extreme Sensitive Functions, each of these functions having the ability to self-adapt to what it can perceive from its environment.

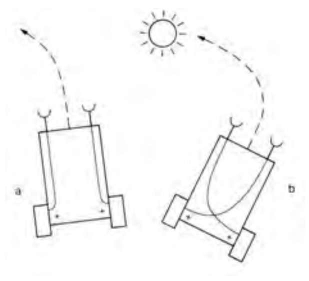

Figure 1.4 — An illustration of two braintenberg vehicles. A difference of connection produces a radically different behaviour.

Thereby, the overall activity results from local interactions between Extreme Sensitive Functions and the environment. Extreme Sensitive Robotics proposes that interactivity of ambient systems is more related to an autonomous observation of the dynamics of the surrounding environment (including the consequences of its own mobility) than the explicit communication between system entities. The absence of this explicit communication reduces the need for a priori knowledge on the system and allows each functionality to be designed separately. Self-observation capacities make the system extremely sensitive to its environment allowing it to integrate changes in its environment into its decision process.

The architecture of a robot is globally composed of functions of perception, action and decision. A robot that performs well has to permanently make cooperate these three functions in association with a loop-back correlating the consequences of its own actions with the observation of changes on its surrounding environment. This is what [Brooks, 1990] expressed as the Physical Grounding Hypothesis. In opposition to the classical reductionist approach, the Physical Grounding Hypothesis postulates that physical interactions with the environment are the primary source of constraints for the design of intelligent systems. Thus, there is no need of symbolic representation of the environment leading to a complex decision making. On the contrary, the system's behaviour is a reaction to a stimulus coming from its environment. Brooks's subsumption architecture [Brooks et al., 1986] is the origin of behaviour-based robotics. In Brooks's architecture, a robot controller is built layer by layer, each layer responsible for one behaviour. The subsumption architecture enables the robot to select the most adequate layer in reaction to what it perceives from its surrounding environment. The postulate that can be made of Brooks's work is that direct interactions with the environment have a strong influence on the robot's decision process. Making an Extreme Sensitive Robot consists in making it sensible to variations in the perception of its surrounding environment and not to an internal state representation. Extreme Sensitive Robotics is all about to sense, not to model.

Pioneering work of Walter Grey [Walter, 1951] in early 50s has shown that even without any form of computational intelligence, a machine can produce a behaviour that one can consider as a smart behaviour, even showing some learning skills. In Grey's robots, an active interaction between sensors and actuators allows a strong interaction with the surrounding environment and the emergence of a behaviour. Braitenberg [Braitenberg, 1986] proposed a set of vehicles where sensors are in direct interaction with actuators. The sensors could have an exciting or inhibitory influence on the actuators. With an exciting influence, more the sensor is excited more the actuator is excited. On the contrary, with an inhibitory influence, more the sensor is excited, less the actuator is excited. Depending of the type of influence and how sensors and actuators are connected, the same robot (same actuators and same sensors) could perform radically different behaviours (Figure 1.4). The only difference lies in how sensors and actuators are connected. It results that the robotic entity is not only influenced by its surrounding environment but also by the nature of the influence between sensors and actuators. Making an Extreme Sensitive Robot is then considering what occurs both outside and inside the robot's body.

Pfeifer [Pfeifer and Bongard, 2006] has named the relation between an entity and its environment as the *embodiment relation*. Pfeifer postulates that the behaviour of an entity is highly influenced by the environment in which it is immersed but also by its own body. To illustrate this phenomenon, Pfeifer proposed the following experiment: looking at the trajectory of an ant walking on rocks, one could say that the behaviour of the ant is smart. Indeed, as the ant is avoiding obstacles, the trajectory appears to be complex. However, if this ant was a thousand times larger, the ant would not be blocked by stones anymore and would then walk in a straight line. The same observer would then say that the ant behaviour is not smart any more. Whereas there has been no change on the ant's mind or on the environment, the observed behaviour differs. A change in the ant body has changed the effect its body produces on the environment. This philosophical experiment shows that any changes in the body could radically change the relation between an entity and its environment. The same idea has to be applied to robotics because some parts of a robot could disappear (for example, a sensor failure) or functionalities added during robot's activity. Even two robots with the same architecture can have electronic differences such as a motor rotating faster than the other does. These modifications of robot's body could have a strong impact on consequences of robot actions on the environment. Making an Extreme Sensitive Robot consists in making it sensitive to the effects that its actions have on the environment.

Thus, building an extremely sensitive robot is then to make it sensitive and adaptive to:

- How its environment evolves, including changes in user activity,
- How its functionalities interact,
- Appearance or disappearance of functionalities and their effects on the environment.

Unlike traditional robotic approach, which consists in building robust controllers for robotic platforms, the *Extreme Sensitive Robotic* approach deals with functionalities. Each functionality is thought and designed to be self-adaptive where the self-adaptation process is driven by a local observation of the environment. Thus, the control is distributed within each of these functionalities. A consequence is that problematic of self-organization is strictly local to each functionality, reducing its complexity. The initial Integrator Problem is thereby solved by the autonomous self-organization of its components (Figure 1.5).

To truly enable the *Extreme Sensitive Robotic* paradigm in the context of *Ambient Robotics*, we need a learning technique enabling the self-adaptation of its functionalities that respects the following properties:

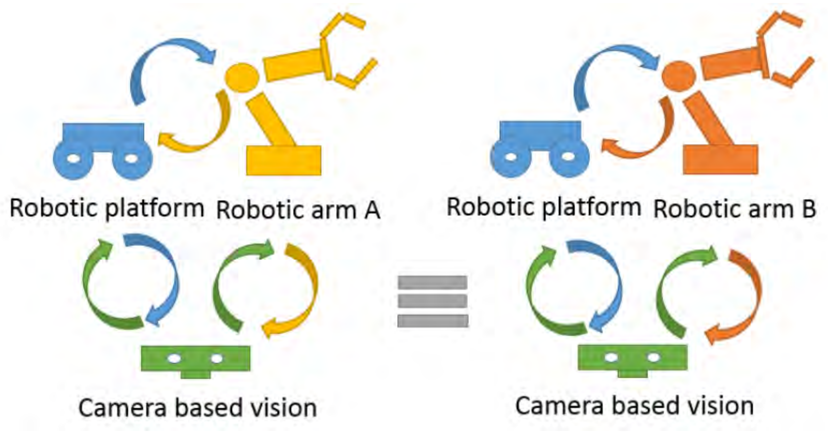

Figure 1.5 — The same problem through the scope of Extreme Sensitive Robotics. As each device is designed to be self-adaptive, the two systems are an equivalent problem.

- **Task Independent**: to be applicable on any kind of devices, the learning technique must not be task dependant.
- **User-Centered**: since an ambient system is thought to be tailored to its users, users have to be the center of the learning approach.
- **On-line**: in order to constantly self-adapt to its users, the learning technique must be performed in real-time.
- **Openness**: due to the high dynamics of ambient environments, the learning technique must deal with the appearance or disappearance of devices.

This set of criterion is going to be used to evaluate the different learning approaches found in the scientific literature (see chapter 2) and the contribution (see section 6.5.1).

1.7 Thesis Objectives

The main objective of this thesis is to assess the relevance and evaluate benefits of the *Extreme Sensitive Robotic* paradigm through the design of a system with the ability to control a functionality in the context of ambient robotics. The design of this system must be made in accordance with both the requirements imposed by the *Extreme Sensitive Robotic* paradigm and the properties of ambient systems.

Thus, the system has to satisfy two major criteria:

- Its control policy has to be continually updated in order to stay in accordance with the dynamic needs of its users. This implies that the system must be able to interact and adapt with its users.
- The system must be usable to control different kinds of robotic devices, in different kinds of environments. The heterogeneity of the control devices and environments involves to minimize the assumptions made on the system environment, and so, in order to maximize its usability.

The expected benefits are at two levels:

▷ Benefits for end-users by enabling end-users to express their needs and interact with the system in a natural way.

▷ Benefits for designers by facilitating the task of designing control systems in an ambient context and reducing the time to release and maintenance costs.

Self-Organization of Robotic Devices Through Demonstrations

State of the art

2 Learning a Control Policy

With this chapter, we explore the different approaches and hypothesis of the machine learning field, seeking for an approach with the capacity to ***learn a control policy****. Such approach has to respect the requirements of the Extreme Sensitive Robotics. In order to do so, we explore the notion of learning in artificial systems, and present the main learning families. From this exploration we outline the proposal to learn from interactions with human.*

LEARNING capacities are required to build intelligent robots with the ability to act with humans. *Learning* and *Intelligence* definitions are intrinsically linked as learning is often seen as a sufficient condition for the expression of intelligence. To give a universal definition of those terms and what makes a machine intelligent is a complex task [Legg and Hutter, 2007]. On this chapter, we highlight the general concepts from machine learning and draw a parallel with our application.

To give agents the capacity to learn is one of the main topic studied in Artificial Intelligence. Machine learning is used to solve tasks that are too complex for traditional algorithms in a variety of domain, with a variety of approaches. But defining learning is a tricky task that often depends on the field in which it is studied [Legg and Hutter, 2007]. From sociology to informatics, each domain came with its own view of what is learning and by extension, how to build a machine that learns. Here is a non exhaustive list of attempts to define learning:

- "Learning is any change in a system that allows it to perform better the second time on repetition of the same task or another task drawn from the same population" [Simon, 1983]
- "Learning is making useful changes in mind" [Minsky, 1983]
- "Learning is the organization of experience" [Fiol and Lyles, 1985]
- "Learning is constructing or modifying representations of what is being experience" [Michalski, 1993]

[Mitchell, 2006] defines the field of machine learning as the science seeking to answer to the questions "How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes". For [Mitchell, 2006], a machine is said to **learn** with respect to a particular task T, performance metric P, and type of experience E, if the system reliability improves its performance

P at task T, following experience E. This definition is interesting as it highlights three fundamental concepts.

The first one is that learning is learning to do *something*. A learning process is highly dependent of the task T to be achieved. A robot in a maze learns to navigate. A chess player learns to play chess. A financial algorithm learns to predict market state. Some examples of tasks could be identification, approximation, prediction or improvement of previously existing knowledge. Thus, learning can only be observed through the performance of a task.

Secondly, learning is not a static process. There is no learning without *evolution*. However, everything that evolves is not necessarily learning. For example, water evolves from ice to liquid, but can we say that ice is learning to melt? Learning means evolving in a *good way*, increasing the capacity of the system to perform its task. To determine if either or not the evolution leads to an amelioration, we need to define a performance metric P. This performance metric is intrinsically linked to the task to evaluate. A system *learns* if it improves its performance metric P between two observations, which means if the system is better at performing T. A system *adapts* if it managed to maintain a performance metric value while the environment has changed. This means that learning (and by extension adaptation) is only observable, measurable by an external observer in regard with the task T. The evolution of the process is called a "learning process". Those metrics can be based on error rate, on the consequences of error or an evaluation of the cost of treatment. Another implication of evolution is the need for memory. Learning and memory are closely related concepts. While learning is often seen as the acquisition process of skill or knowledge, memory can be seen as the expression of what have been learned. Learning is not only increasing a performance metric but also the capacity to reuse what have been learnt.

The last concept is the concept of experience E. How is the learner going to learn? With what kind of data? *Experience* is about describing the interactions between the learner and its *environment*.

Depending on how we specify T, P and E, the learning task may have different names such as data mining, predictive learning, programming by example, etc. The comparison of machine learning algorithms is only possible if T, P and E have the same nature.

The field of machine learning is usually distinguished in three categories [Russell and Norvig, 1995] [Cornuéjols and Miclet, 2011] based on how the experiment E is performed: **supervised learning**, **unsupervised learning** and **reinforcement learning**.

▷ **Supervised learning** algorithms infer a function from labelled data provided by an *oracle*. Each data is a pair of (*input*,*output*) values. The algorithm then has to infer a mapping function in order to be able to correctly associate to each new *input* its correct *output*. When the algorithm maps inputs to a discrete output space, we speak of a *classification* problem. When the output is continuous, it is a *regression* problem. The learning is traditionally ***off-line***, as those algorithms need to be trained on data example before being able to correctly label new data.

▷ On contrary, **unsupervised learning** algorithms try to find hidden structure in unlabelled data. They use no function of evaluation or labelling of a training data set.

Their goal is to learn to split data, establish correlations or to learn natural components of unlabelled data. Like supervised learning, this learning is traditionally performed ***off-line***.

- At last, **reinforcement learning** algorithms focus on the relation between the learner and its environment. Those algorithms learn from the interaction between the learner and its environment to optimise a utility function. It is an ***on-line*** learning based on the evaluation of the gain of the performance of an action on the current environment.

However, such categorisation is based on technical aspects and does not allow to highlight the fundamental differences between algorithms, and this categorisation can even be ambiguous. Indeed, [Russell and Norvig, 1995] express that supervised learning and reinforcement learning can be seen as a same form of learning, as those two approaches use an external entity to learn. Their difference is the expressiveness of the information provided by this external entity, which is richest in the case of supervised learning by providing directly the expected solution. As *learning* is studied in many sciences from psychology to informatics [Leonard, 2002], machine learning has been influenced by the different views of learning and how the learner and its environment interact. This interest in learning has led to many learning theories, each having its own hypothesis. Instead of studying machine learning algorithms only from their technical differences, we first propose a list of learning theories that have influenced machine learning. For each of those theories, we describes some of the algorithms that emerged from it. This list does not intend to be exhaustive but rather be an illustration of how our conception of what learning is influences our way to design artificial learning systems. This categorisation enables to position the work presented in this thesis in regard to the highlighted approaches. More precisely, we put a focus on the criterion that has been defined with the *Extreme Sensitive Robotics* approach in section 1.6.

2.1 Behaviourism - Mind is a *Black-box*

$\rightarrow$ *Learning is the modification of the observed behaviour of an individual due to the modification of an answer associated to an external stimuli*

Behaviourism is an approach to psychology that focuses on an observable individual behaviour [Skinner, 2011]. A *behaviourist* believes that, as human are biological machines, they do not have their actions determined by thoughts, feelings, intentions or mental processes. What is inside the mind is unobservable: the mind is a *black-box* [Friedenberg and Silverman, 2011]. This *black-box* is composed of *inputs* and *outputs*. *Inputs* describe what the entity perceives of its environment and *outputs* describe what it does. The *black-box* starts *tabula rasa* and the *behaviour* is a product of conditioning, a reaction to stimuli associating to a particular instance of *input* the adequate *output*. Then, *behaviourists* see learning has a modification of the observed behaviour due to the modification of the strength of an answer associated to an external stimuli (external environment) or to an internal stimuli (internal environment) of an organism. From the behaviourist point of view, three forms of learning can occur: *classical conditioning*, *operant conditioning* and *observational learning*.

Probably one of the most known behaviourist is Ivan Petrovitch Pavlov who introduced the concept of *classical conditioning* [Pavlov, 1941]. Classical conditioning is a learning process in which an unconditioned response to an unconditioned stimulus comes to be elicited to a previously neutral stimulus. Pavlov realised an experiment with dogs where the dog is presented with a neutral stimulus such as a light or a sound, and then food is placed in the dog's mouth. Whenever the dog see food, it salivates as an unconditioned response to the food unconditioned stimulus. After a few repetition of the sequence, the simple presence of the sound or light caused the dog to salivate even with the absence of actual food. The sound or light is now a conditioned stimulus leading the dog to salivate.

Operant Conditioning is based on Thorndike's "law of effect" which stipulates that responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation [Thorndike, 1927]. Learning occurs through *reinforcement* and *punishment* [Skinner, 1938]. *Reinforcement* is any event that *increases* the behaviour it follows. On the other hand, *punishment* is the presentation of an adverse event or outcome that causes a *decrease* in the behaviour it follows. Reinforcement and Punishment can be either positive or negative:

▹ A *positive reinforcer* is a favourable event or outcome that is presented after the behaviour. The behaviour is strengthened by the addition of something, such as praise or a direct reward.

▹ A *negative reinforcer* involves the removal of an unfavourable event or outcome after the display of a behaviour. The behaviour is strengthened by the removal of something considered unpleasant.

▹ *Positive punishment* involves the presentation of an unfavourable event or outcome in order to weaken the response it follows.

▹ *Negative punishment* occurs when an favourable event or outcome is removed after a behaviour occurs.

The last form of learning is *observational learning* which has been introduced by Bandura during its famous *Bobo Doll* experiment [Bandura et al., 1961]. Bandura proposed to study if social behaviours can be acquired by observation and imitation. For that, he realises an experiment in which children where individually put in a room with toys and a Bobo Doll. Children where demonstrated by an adult with two different kinds of behaviour. For some, the adult demonstrates an aggressive behaviour by hurting the Bobo doll with hands and objects. For other, the adult demonstrates a non-aggressive behaviour playing nicely with the Bobo doll. After 10 minutes, the adult leaves the room. Children who observed the aggressive model made far more imitative aggressive responses than those who were in the non-aggressive or control groups. Results of the experiment goes beyond this simple fact but the experiment clearly shows the importance of imitation in the learning process and introduces a social dimension to learning. This experiment leads to the proposal of a social learning theory [Bandura and McClelland, 1977].

2.1.1 Reinforcement Learning

Reinforcement learning is an area of machine learning inspired by the behaviourist psychology. The aim of a reinforcement algorithm is to maximize some notions of cumulative reward. Initially, the algorithm has no *a priori* knowledge on the effects of actions. In an exploratory phase, the algorithm has to determined, through experimentation, the gain of performing a particular action in a particular situation. Then, in an exploitation phase, the algorithm uses its previously acquired knowledge to choose actions that maximise gain expectancy. The model constructed by a reinforcement algorithm is called a *policy*. It determines which action has to be realised depending on the current state. As exploration and exploitation are performed in parallel, reinforcement learning algorithms are **on-line** learning approaches. On this section, we present two variants of reinforcement learning algorithm: Q-Learning and SARSA.

2.1.1.1 Q-Learning

The Q-Learning algorithm [Watkins, 1989] is based on the dependency between a utility function and actions to learn an optimal behaviour. The aim is to correctly evaluate from past experiments the utility function $Q(s,a)$ that computes the utility of an action a in the state s.

The utility function is evaluated from the immediate reward r obtained after the realisation of an action a through the following iterative process :

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)]$$

where $Q(s,a)$ is the estimated utility of the action a if this action is performed in the state s, A is the set of actions, s' is the reached state from s with the action a, $\max_{a' \in A} Q(s',a')$ is the estimated utility of the best action a' to perform in s', and α and γ two parameters defined by the designer.

The algorithm must alternate between phases of exploration (which means applying an action independently of its utility) and phases of exploitation (which means selecting the best action to perform). This is made with an ϵ-greedy policy. In this policy, the action is selected greedily with respect to the Q-value estimates a fraction $(1-\epsilon)$ of the time (where ϵ is a fraction between 0 and 1), and randomly selected among all actions a fraction ϵ of the time.

As the next action is selected among the set of all possible actions, the Q-Learning algorithm is said to be "off-policy". This ensures the convergence of the algorithm assuming that all system states are visited infinitely often.

2.1.1.2 SARSA

The SARSA (State-Action-Reward-State-Action) algorithm is an "on-politic" variant of the Q-Learning algorithm. Instead of selecting the action maximizing the utility, the action is selected in accordance with a policy π provided by the designer [Sutton, 1996]. This involves that the iterative update of the utility function is no more based on the best action to perform but on the next action recommended by the policy π:

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha[r + \gamma Q^{\pi}(s',a') - Q^{\pi}(s,a)]$$

where $Q^{\pi}(s,a)$ is the estimated utility of the action a if this action is performed in the current state s by following the policy π, r the immediate reward, s' is the reached state from s with the action a, $Q^{\pi}(s',a')$ is the estimated utility of the action a' recommended by the policy π, and α and γ two parameters defined by the user. The name of the algorithm comes from the parameters s,a,r,s',a'.

Similarly to the Q-Learning algorithm, an ϵ parameter allows to determine the probability of performing a random action.

2.1.2 Analysis

There have been a lot of works on reinforcement learning algorithm and they have shown interesting results in simulation and robotics [Kober and Peters, 2012]. Reinforcement learning promotes an approach where learning is the result of an active coupling between an agent and its environment. However, for our application, reinforcement learning presents some limitations. Reinforcement learning algorithms have an on-line learning property and the capacity to optimise a politic. In order to do so, they need a feedback function which enables the algorithm to evaluate the utility of the performance of an action in the current situation. The design of such function is a well-known complex task which has to be set up in regard of the desired objective to optimise. Q-learning and SARSA are limited to environments which can be modelled by a Markov decision process and needs a discrete model of environment states which could not be guaranteed in real world applications. Furthermore, the algorithm needs to explore the different actions to evaluate their utility. A large number of cycles is necessary to reach an adequate behaviour. This exploration can lead to the irreversible degradation of the controlled system.

2.2 Cognitivism - Opening the *Black-box*

$\rightarrow$ *Learning is making useful changes in mind*

One major critics to behaviourism came with Chomsky [Chomsky, 1959] which states that behaviourist models of language learning (through conditioning by "action-reaction-repetition") cannot explain various facts about language acquisition, such as the rapid acquisition of language by young children. In opposition to the "action-reaction-repetition" view of learning, Chomsky states that children have the capacity to produce sentences that they have never heard before. He answers that such capacity involves that the mind does not only "repeat" what they have heard but that they process information in their mind and extract rules which they then apply to create new sentences. Then, Chomsky argue that to study behaviour it is necessary to study the underlying mechanisms, which means open the box and study what is between *inputs* and *outputs*. This is the congnitivist postulate.

Criterion	Reinforcement learning	Comments
Task independent	- -	The feedback function has to be designed in accordance with the task to perform.
User-Centered	+	User preferences can be include in reinforcement learning algorithm through the feedback function.
On-line	++	Learning is performed in parallel with the exploitation.
Openness	- -	Reinforcement learning algorithms are not robust to changes in the problem description due to the dependence to the feedback function and the need to visit each environment state.

Figure 2.1 — Reinforcement learning assessment

In opposition to the behaviourist approach, congitivists consider the mind as an information-processing entity, a unit which manipulates knowledge. Cognitivist approach is based on the hypothesis that learning is not about what the learners do but what they know and how they acquire knowledge. [Newell and Simon, 1976] postulate that a physical symbol system has the necessary and sufficient means for general intelligent action. The solutions to problems are represented as symbol structures. A physical-symbol system applies its intelligence in problem-solving by search, that is, by generating progressively modifying symbol structures until it produces a solution structure.

This position has been quite influential in artificial intelligence from its very beginning. However, in regard of our application, such a position presents two major limitations.

As learning from the congnitivist point of view is seen as symbol manipulation, it involves to make an abstraction of the world. Then cognitivist models rest upon a predefined symbolic model. Such model is built by the human designer and includes its own knowledge. That is why such systems are named *expert systems*, because their model representation depends of the ability of a human expert. The more the problem to study is rich and complex and the more such model is difficult to build. Furthermore, such model needs to be complete a priori, which means that new symbols cannot be added in an expert system without affecting the model.

Without going in the details of the approach, it clearly appears that a cognitivist approach will not respect the openness property required by our application.

2.3 Connectionism

$\rightarrow$ *Learning is the emergent process of interconnected networks of simple units*

Connectionism is a set of approaches that models mental or behavioural phenomena as the emergent processes of interconnected networks of simple units [Medler, 1998]. Inspired by the human natural ability to learn, connectionists try to copy this ability by mimicking

Criterion	Reinforcement learning	Comments
Task independent	- -	Needs of human expertise.
User-Centered	- -	User needs to be modelled a priori.
On-line	- -	The reasoning is about the current system knowledge.
Openness	- -	The world needs to be modelled *a priori*.

Figure 2.2 — Cognitivist approach assessment

the physiology of brains, composed of interconnected neurons. In biology, a neuron is an excitable cell which composes a basic function of the nervous system of an animal. A neuron perceives synaptic signals from its dendrites which, if a threshold is reached, leads to an output nervous impulsion from its axon. Connected through their synapses, neurons form a complex network through which information is exchanged under the form of bio-electric stimulations. As such network is able to learn (especially neural network composing the human brain), researchers in the field of machine learning have proposed to study and create artificial networks of interconnected simple units to mimic the cognitive capacity of brains. The form of the connections and the units can vary from model to model. The most known approach is *artificial neural networks* [McCulloch and Pitts, 1943] where units in the network represent neurons and connections represent synapses.

2.3.1 Artificial Neural Networks

Introduced by [McCulloch and Pitts, 1943], Neural Networks are a set of learning algorithms widely studied and very popular in many domains of application from robotics to images identification. Inspired by the physiology of brains, they are composed of interconnected small units called *neurons*. We limits our presentation of Neural Networks to their utilisation in supervised learning under the form of *perceptrons*. Initially proposed by [Rosenblatt, 1958] and then extended to a multi-layer architecture by [Fiesler and Beale, 1996], *perceptrons* are linear classifiers that maps a set of input data to a set of appropriate outputs.

2.3.1.1 Perceptron

The basic element of the artificial neural network is an artificial *neuron*. An artificial neuron is a computation unit with inputs, outputs, internal state and parameters. The unit processes input data to generate an appropriate output. There are many variations of neurons, depending of the type of neural network. A generic type of artificial neuron, a formal neuron, is shown in Figure 2.3.

A formal neuron is modelled by its internal state σ_i, with i as the index of the neuron on the network, and an output function g allowing to determine its output value y_i:

$$y_i = g(\sigma)$$

Various function σ_i can be considered but most of the time a sigmoid function is used:

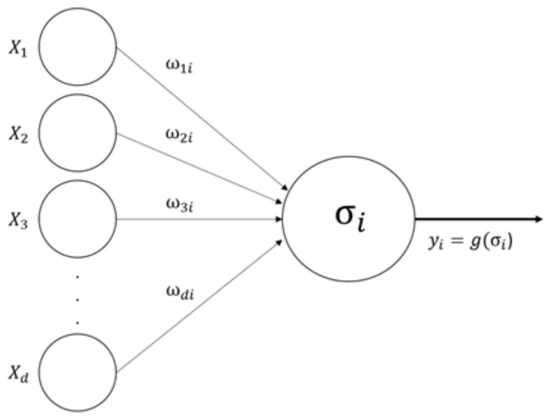

Figure 2.3 — A formal neuron

$$y_i = g(\sigma_i) = \frac{1}{1+\exp^{-\lambda\sigma_i}}$$

Each input j of a formal neuron i is associated with a weight ω_{ji}. The value of σ_i is generally computed from the d input values x_j and their associated weight ω_{ji} with the formula:

$$\sigma_i = \sum_{j=1}^{d} \omega_{ji} x_j$$

2.3.1.2 Multilayer Perceptron

A neural network is characterised by its architecture, which means by the way the formal neurons composing it are linked one to another. The architecture possibilities are basically infinite, depending on the task to perform. Some like the Boltzmann machines [Ackley et al., 1985] have a complete connectivity and each formal neuron is connected with every other units. We focus on the most known form: the *multilayer perceptron*. In a *multilayer perceptron* (MLP), formal neurons are classified in three category:

- *Input neurons* which composes the input layer are responsible of transmitting input data (which could be either example or data to labelled). In this case, the rule to determine the neuron state is $\sigma_i = x_i$ where x_i is the input value of index i.
- The last layer is composed of *output neurons* where each neuron corresponds to a class.
- *Hidden neurons* are the neurons that are neither in the input layer nor the output layer. They are intermediary computational units.

Multilayer perceptron contains no cycle, the information is *feed-forward*, which means that information only travels in one way, from input neurons to output neurons through hidden neurons. Neurons from the first layer are activated by received input values. They compute their output value σ_i and the result is sent to the first hidden layer. This layer computes its values and so on until the output layer is reached. The result is then the class of the output neurons with the biggest output value. The number of hidden neurons layers and the number of hidden neurons in each layer has to be determined by the designer. The ability of a neural network to generalise is dependent of the number of hidden layers, but increasing the number of hidden layers have an influence on the system's complexity, as more parameters has to be defined. An example of a multilayer perceptron is visible in Figure 2.4.

2.3.1.3 Learning through backpropagation

To enable learning in a multilayer perceptron we need to add a mechanism of backpropagation of the amount of error in the output compared to the expected result [Rumelhart et al., 1985]. The backpropagation of error enables the network to adjust weights of connections.

Sample data are sequentially provided to the neural network which adjusts in result the weights of its connections. This process is repeated a great number of time (generally a hundred times for each example) enabling the network to converge.

To adjust the weights of its connections, the network follows a *delta rule*, which is a gradient descent learning rule for updating the weights of neurons i and a neuron j:

$$\Delta\omega_{ij} = \alpha\delta_j y_i$$

where y_i is the output value of the neuron i (one of the input of the neuron j), $\alpha \in [0,1]$, and δ_j is a value based on the error. For an output neuron (with a sigmoid function g with parameter $\lambda = 1$), δ_j is expressed as follow:

$$\delta_j = (u_j - y_j)y_j(1 - y_j)$$

where u_j is the desired output (corresponding to the value of the example), y_j is the computed output. Recursively, the λ value for hidden neurons can be obtained with the formula:

$$\delta_j = y_j(1 - y_j) \sum_{k \in fol(j)} \delta_k \sigma_{jk}$$

where $fol(j)$ is the set of neurons receiving the output of the neurone j. As a matter of fact, the adjustment of the weight of connection of a layer is only possible if the lower layer has computed its own adjustments.

2.3.2 Self-Organizing Maps

Self-Organizing Maps (SOM), also called Kohonen's maps or network, [Kohonen, 2001] are a particular type of Artificial Neural Network (section 2.3.1) that aims to produce a low-dimensional discretized representation of a high-dimensional input space of training samples through unsupervised learning (see Figure 2.5). SOM are composed of a neurons grid (generally a two-dimensional grid). Each neuron is associated with a weight vector

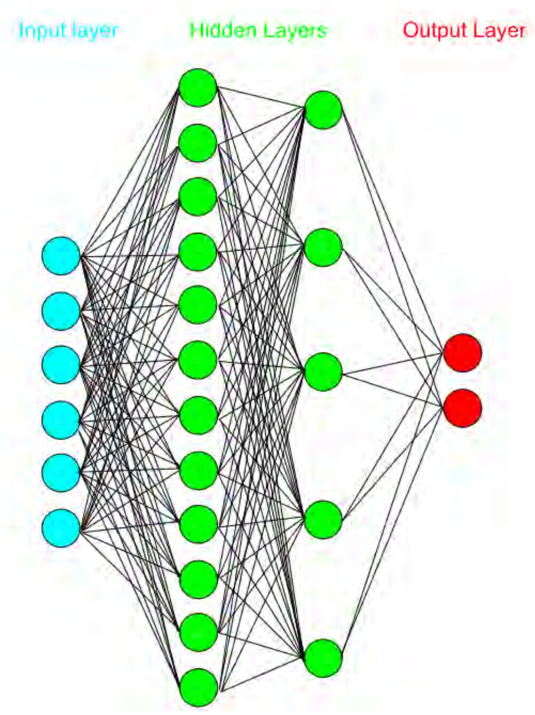

Figure 2.4 — A multilayer perception composed of two hidden layers and two output neurons

of the same dimension as the input data vectors, and a position in the map space. The weight of the SOM are initialised randomly and input data are randomly subjected to the SOM (each data is subjected many times). The node with the closest weight vector is selected and both its weight and position are updated to better match the example. Thus, neighbouring neurons are updated with a lower factor. The algorithm stops when a stopping criterion is reached. Such criteria can be the number of data subjections or a certain stability in the weights updates. Once the SOM has converged, the similarity between the input data as measured in the input data space is preserved as faithfully as possible within the representation space. There are numerous variants of the basic SOM [Príncipe and Miikkulainen, 2009], some proposing to use a multi-layered hierarchical SOMs [Dittenbach et al., 2000]. In their formal definition, SOMs do not allow classification or regression but variants have been proposed for regression learning where the input space is mapped to an output space [Hecht et al., 2015].

2.3.3 Deep Learning

Deep Learning denotes a set of algorithms that emerged from the usage of multi-layer Neural Networks. Those algorithms are said to be deep because the input is passed through several non-linearities before being output. Deep learning has many definition but [Deng and Yu, 2014] stipulates that these definitions have in common a multiple layers of nonlinear processing units and the supervised or unsupervised learning of feature representations in each layer, with the layers forming a hierarchy from low-level to high-level features. Deep

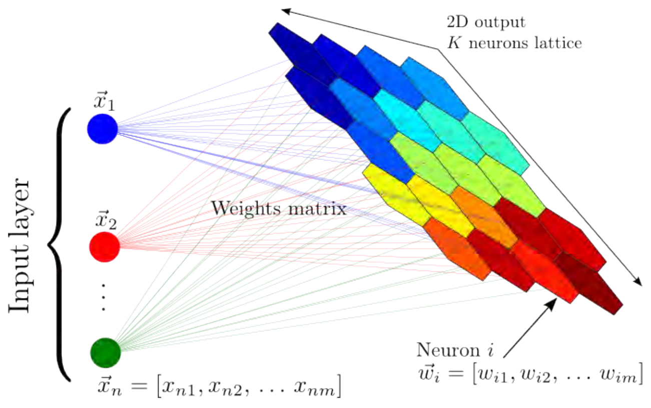
2D output
K neurons lattice
$\vec{x}_1$
$\vec{x}_2$
Input layer
Weights matrix
$\vec{x}_n = [x_{n1}, x_{n2}, \ldots x_{nm}]$
Neuron i
$\vec{w}_i = [w_{i1}, w_{i2}, \ldots w_{im}]$

Figure 2.5 — Illustration of a Self-Organizing map

learning algorithm have shown interesting results in pattern recognition [Schmidhuber, 2015]. Recently, [Mnih et al., 2013] used a deep learning approach to teach a convolutional neural network to control seven Atari 2600 games from high-dimensional sensory input using reinforcement learning.

2.3.4 Analysis

The number of neurons composing each layer, the activation function and the threshold of each neurons are parameter that need to be instantiated for each neural network. If the number of neurons is undervalued, the neural network could fail to learn a complex function whereas a number of neurons over-rated could have an impact on the neural network performances. Some approaches propose to help the designer to choose the good parameters by learning the network topology using other learning algorithms. For example, [Kant and Sangwan, 2015] combine neural network and genetic algorithms but even in this case, parameters have still to be hand-tune. More generally, neural networks have good results on classification tasks, but they lake of robustness in application where input or the task to perform can change. Another limitation of neural networks usage is the number of examples required to teach a neural network (which needs to be consequent) and the time to learn. Anyhow, they promote the seducing idea that learning is the emerging result of the interaction of simple units.

2.4 Evolutionism

$\rightarrow$ *Learning is the result of a long term adaptation of an individual to its environment due to selection pressure.*

Nature ability to self-adapt (as expressed in the popular proverb "Nature will find a way") has always been a great source of inspiration for scientists. Using the metaphor of *natural selection*, [Holland, 1992] proposes to build artificial systems that adapt to their environment through a process of variation and selection. Natural Selection is considered to be a key mechanism of evolution. The different animal species evolve from generations to generations inducing changes in their phenotypes. Those changes of phenotype lead to

Criterion	Neural networks	Comments
Task independent	+	The task to learn is unknown but the structure of the neural network influences it. However, once a task has been learned, the neural network has more difficulties to readjust
User-Centered	+	Labelled inputs can be provided by users to teach the neural network.
On-line	- -	Examples are provided offline, each example needing to be presented to the neural network a hundred times.
Openness	- -	The structure of the neural network has be instantiated relatively to the task to perform and is not robust to changes.

Figure 2.6 — Neural networks assessment

a differential survival and reproduction of individuals. Basically, the more a phenotype is adapted to its ecosystem, the greater is the capacity of the individual to survive and reproduce. The adaptation of a specie due to selection pressure of its environment can be considered as the result of a long term learning (phylogenetic learning [Pfeifer and Bongard, 2006]). This is the postulate made by *genetic algorithms*.

2.4.1 Genetic Algorithms

In genetic algorithms, a population of hypothesis is evolved toward better solutions. The hypothesis space is split in two spaces:

▷ A *phenotypic* space in which hypothesis can be evaluated and selected.

▷ A *genotypic* space in which hypothesis are transformable by specific operators.

An hypothesis is coded with a discrete alphabet (traditionally a binary alphabet). A genetic algorithm has then to cross individuals from an original population (generally a randomized population) P until a satisfying hypothesis is found. The evolution of populations is a four step process:

▷ 1: **Evaluation**: Each member of the current population is evaluated through a fitness function. The fitness function is provided by the designer and allows to compute a score which evaluates the performance of an individual in regards to the task to perform. In the case of supervised learning, such function compares each hypothesis with a set of provided examples. There are various fitness functions depending on the type of problem [Nelson et al., 2009].

▷ 2: **Selection**: The second step of the process is to select a portion of the population which will be used to breed a new generation at step 3. Analogically to selection

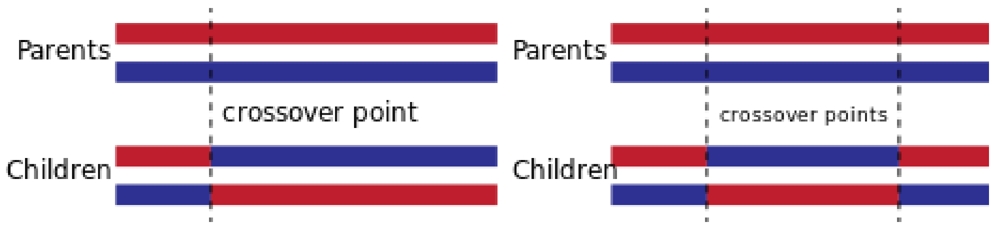

Figure 2.7 — One-point and Two-points crossover operators

pressure in Natural selection, this step determines those who survive and those who die. To assure diversity among the population while promoting individuals with the best fitness score, a probability of selection weighted by the fitness score is associated to each individual. So, each individual can be selected but individuals with the higher fitness score are favoured.

▷ 3: **Genetic operators**: Once the portion of population is selected for breeding, the new generation has to be created. Two mechanisms occurs: *crossover* and *mutation*. During crossovers, the genotypes of two individuals are mixed using a crossover operator to produce a new individual. Several operators exist for assuring the crossover. For example, the classical One-Point crossover operator randomly selects a point in the genotype and switches all the genes from this point to the end of the genotype while the Two-Points crossover operator performs the same action between two random points (see Figure 2.7). By crossing best individuals of a population, we hope to to build a new individual combining the strength of its both parents. If not, the selection process will make the new individual disappears at the next step. In complement with the *crossover* mechanism, a *mutation* rate m is associated with each gene which can randomly change its value. This allows the apparition of new phenotypic properties which, if they present advantages, will be kept during the selection process.

▷ 4: **Update**: In the last step, a fraction r of the current population is replaced with the new generation obtained in the third phase.

Those four steps are repeated, creating at each step a new generation in which the best individuals are more and more adapted to the task to perform (which means that the fitness value increases). The algorithm stops when the fitness value of an individual reaches a threshold fixed by the designer. A generic genetic algorithm is provided in Algorithm 2.1.

2.4.2 Analysis

Genetic algorithms have some interesting advantages. They can be used to address problem with no known solutions. Additionally, evolutionary algorithms are domain-friendly. They can handle highly heterogeneous constraints and objectives, as they do not need an analytical expression of the domain. For thus, they can easily be combined with other approaches to find the best parameters of an algorithm or the best controller architecture [Doncieux et al., 2015]. The sole definition of a fitness function can lead an evolutionary algorithm to find a satisfying solution. Furthermore, the paradigm has led to

Algorithm 2.1: A generic description of a genetic algorithm

```
Data: Fitness: a fitness function, Threshold: a termination criterion, $p$: the number of
      hypothesis in the population, $r$: the fraction to be replaced by crossover, $m$: the
      mutation rate
Result: The hypothesis with the highest fitness
Initialise population : $P \leftarrow$ Generate $p$ hypothesis at random;
Evaluate: for each $h \in P$, compute $Fitness(h)$;
while $max_h Fitness(h) < Threshold$ do
    Select : Probabilistically select $(1-r)p$ members of $P$ to add to $P_s$;
    Crossover: Probabilistically select $\frac{rp}{2}$ pair of hypothesis from $P$. For each pair,
    $< h_1, h_2 >$ produce two offspring and add top $P_s$;
    Mutate: Choose $m$ percent of the members of $P_s$ with uniform probability. For
    each, invert one randomly selected bit;
    Update: $P \leftarrow P_s$;
    Evaluate: for each $h \in P$, compute $Fitness(h)$;
end
Return the hypothesis $h \in P$ with the highest $Fitness(h)$;
```

the emergence of Evolutionary robotics, a subfield of robotics focusing on the robust and adaptive design of robots [Doncieux et al., 2015].

A first limitation of the approach is the difficulty to tune parameters. The choice of the size and composition of the initial population, the encoding of the genotype, and the choice of the operators, are not trivial. Some approaches propose to adapt those parameters at runtime [Kramer, 2010]. The design of the fitness function is also "notoriously difficult" [Bongard, 2013] and must be tuned for any new applications. Some works try to escape this limitation by studying task-agnostic approaches to genetic algorithm to enable the transfer to other tasks with limited or even no modification at all [Doncieux and Mouret, 2014]. However, the major limitation of genetic algorithms is that they require a large number of evaluations. Each produced population has to be evaluated in order to converge towards more effective individuals. Such need of evaluations restricts their usage to simulation.

2.5 Constructivism

$\rightarrow$ *Learning is an active process of knowledge construction between a learner and the environment.*

Constructivism is a theory of knowledge initiated by the work of Jean Piaget on the development of children [Piaget, 1954]. Constructivists believe that all humans have the ability to construct knowledge in their own minds through interactions with their environment. The basic components of this approach are *schemas* (or *schemata*). Schemas are considered to be the basic unit of knowledge which seeks for opportunities to repeat themselves [Guerin, 2011] (this notion is similar to the notion of pattern). [Piaget, 1954] sees learning as an active process of *accommodation* and *assimilation* where individuals construct

Criterion	Genetic algorithms	Comments
Task independent	+	Genetic algorithms are domain friendly. However, the expression of the fitness function could limit the usages. Some approaches allow the fitness function to be changed at runtime.
User-Centered	+	User needs can be expressed through the fitness function.
On-line	+	Each produced population has to be evaluated in order to converge towards more effective individuals.
Openness	+	Depending on some choices of model, genetic algorithms can theoretically include appearance or disappearance of data

Figure 2.8 — Genetic algorithms assessment

new knowledge from their experiences. On the one hand, *Assimilation* happens when an individual incorporates a new experience into existing schemas. The experience is fitted to the model. On the other hand, *accommodation* happens when existing schemas needs to be changed to deal with a new object or situation. The model is fitted to the experience. [Wadsworth, 1996] describes schemas as *index cards* filled in the brain. Each *card* tells to an individual how to react to incoming stimuli or information. [Piaget, 1954] defines the child period from birth to 2 year as the sensory-motor period of the intelligence in which children's intelligence is self-constructed though experiences. By exploring its environment, the child acquires the basic representation of its environment. The approach has led to the creation of a sub-field of robotics: the developmental approach [Lungarella et al., 2003]. [Weng et al., 2001] describes the differences between traditional manual programming and the developmental approach as follows:

- ▷ With the traditional manual development, the program is developed as follow:
  - Start with a task, understood by the human engineer (not the machine)
  - Design a task-specific representation
  - Program for the specific task using the representation
  - Run the program on the machine
- ▷ The developmental approach proposes the following process:
  - Design a body according to the agent's ecological working condition
  - Design a developmental program
  - At "Birth", the agent starts to run the developmental program
  - To develop its mind, humans mentally "raise" the developmental agent by interacting with it in real time

However, on this chapter, we only focus on the influence of the paradigm on machine learning. More precisely, we consider approaches based on the constructivist hypothesis named Schema-learning.

2.5.1 Schema-Based Learning

2.5.1.1 Initial formalisation

Schema-learning is a machine learning method inspired by the constructivist approach initially proposed by [Drescher, 1991]. It intends to model regularities in the interaction between a learning agent and its environment. [Holmes et al., 2004] proposes a formal description of Schema-Learning. A schema learner is fitted with a set of sensors $S = \{s_1, s_2, s_3, ...\}$ and a set of actions $A = \{a_1, a_2, ...\}$ through which it can perceive and manipulate the environment. Each sensor value s_i has a discrete value j such as s_i^j describes the value of the sensor i. By observing the effects of the action on the environment, the learner builds *schemas*. *Schemas* are tripartite structures <*Context*,*Action*,*Result*> and model the expected *results* of the performance of an *action* under a certain *context* (Figure 2.9). A schema is not a rule telling the agent what action to perform but a description of the consequences of the performance of the action under a certain context. A schema $C \xrightarrow{a_i} R$ is a description of the expected result R obtained by performing the action a_i in the situation C. The *context* $C = \{c_1, c2, ..., cn\}$ is a set of sensors condition $c_i \equiv s_j^k$ that are a precondition to the activation of a Schema and $R = \{r_1, r_2, ..., r_m\}$ is a set of sensors condition predicted to follow the action. A schema is said to be *applicable* if its context conditions are satisfied, *activated* if it is applicable and its action is taken, and to *succeed* if it is activated and its predicted result is obtained. Schema quality is evaluated by *reliability*, which is the probability that activation culminates in success $Rel(C \xrightarrow{a_i} R) = Prob(R_{t+1}|C_t, a_{i(t)})$. Two basic phases are involved in Schema-Learning : *discovery* in which context-free action/result schemas are found and *refinement* in which context is added to increase reliability. This process is named *marginal attribution*.

In discovery, statistics track the influence of each action a_i on each sensor condition s_r^j. Results of an action are determined by computing a ratio of the number of times a result is discovered after the performance of an action to the number of times the same result is discovered without the performance of the action. The result is associated with an action if the computed value is better than a threshold $\Theta_d > 1$.

$$Discovery\ rule : \frac{prob(s^j_{r(t+1)}|a_t)}{prob(s^j_{r(t+1)}|a_t)} > \Theta_d$$

When a schema is discovered, it has no context. The reliability of the schema may be low if the detected effect occurs only in particular situations. The refinement strategy allows to look for context conditions that increase the schema reliability. The criterion for adding a context condition s_c^j to a schema $C \xrightarrow{a_i} R$ is :

$$\textit{Refinement rule: } \frac{Rel(C \cup \{s_c^j\} \xrightarrow{a_i} R)}{Rel(C \xrightarrow{a_i} R)} > \Theta_c \text{ where } \Theta_c > 1.$$

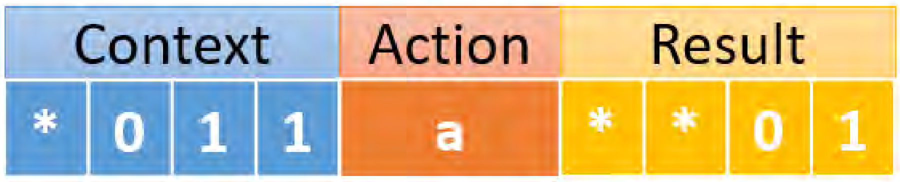

Figure 2.9 — The structure of a schema

Once the criterion is reached, a child schema $C' \xrightarrow{a_i} R$ is formed, where $C' = C \cup s^j_c$.

In addition with the discovery and refinement, the schema mechanism is able to discover hidden states through a process of *synthetic item* creation. In some cases, no context conditions are found to make a schema reliable. The schema learner can then create a new binary-valued virtual sensor, called a *synthetic item*, to represent the presence of unobservable conditions in the environment that have an influence the activation of the schema. [Drescher, 1991] postulates that two conditions are required for synthetic item creation: (1) a schema must be unreliable and (2) the schema must be locally consistent, meaning that if it succeeds once, it has a high probability of succeeding again if activated soon after the first success. The criterion for synthetic creation is $0 < Rel(C \xrightarrow{a_i} R) < \Theta_r$. When the criteria is met, a synthetic item is created and treated as a normal item. Because synthetic items are treated as a sensor, the schema learner can discover which previous action has led to each synthetic item state and the synthetic item can come to be included as a result condition in other schemas.

2.5.1.2 Evolutions

The initial work proposed by Drescher has been quite influential [Guerin, 2011] and evolutions of schema-learning have been proposed.

[Holmes et al., 2004] generalises and improves the Drescher's mechanism and proposes to view the problem as one of learning a Partially Observable Markov Decision Process.

[Chaput, 2004] proposed the *Constructivist Learning Architecture* based on Leslie Cohen's theory of infant cognitive development (an evolution of the initial Piagetian theory). The Constructivist Learning Architecture is based on self-organizing maps (2.5) where candidates schemas compete for a limited space via Self Organizing Maps.

[Perotto et al., 2007] proposed the *Constructivist Anticipatory Learning Mechanism* (CALM) System which adds adaptivity to schemas. The agent can modify its schema through differentiation, adjustment or integration. In differentiation, a general schema produces an unexpected result and leads to the creation of two new more specific schemas. Adjustment occurs when an erroneous prediction is made and differentiation is not possible. The schema structure is then adjusted to fit with the situation. Integration is differentiation in reverse, two schemas with the same expectation can be replaced by a single schema with a more general context.

In a more recent approach, [Mazac, 2015] proposed a multi-agent implementation of the constructivist approach applied to ambient intelligence to predict recurrent sensori-motor patterns on smart buildings.

2.5.2 Analysis

The schema-learning approach is probably the approach which has the most positive criteria. It focuses on the design of agents with the ability models regularities of interactions with its environment. However, by itself, the approach is not a control approach. Schema-learning interest is to build a model of the world and not necessary to control it. Sensibility to context and the ability to deal with missing data are two major advantages of the method.

Criterion	Schema learning	Comments
Task independent	+ +	The agent builds its own model of the world independently with its own objectives.
User-Centered	+ +	Users can "raise" the agent by interacting with it
On-line	+ +	Model construction is performed in parallel with the exploration
Openness	+ +	Schema-learning can deal with missing data with synthetic item and can handle the appearance of new data.

Figure 2.10 — Schema learning assessment

2.6 Other and Hybrid Approaches

The previously presented categorisation is not exhaustive and exclusive, as some of the approach can be combined. For example, [Heinen and Engel, 2010] combines neural network and reinforcement learning or [Kant and Sangwan, 2015] coupled neural networks and genetic algorithm. Thus, there is a wide variety of implementation of algorithms for each category, and even more combination possible. To illustrate such a possibility, we propose to focus on Learning Classifier Systems, which are a combination of reinforcement learning principle and genetic algorithms, and Case-Based reasoning.

2.6.1 Learning Classifier Systems

Learning Classifier Systems are a set of tools combining the principles of *reinforcement learning* and *genetic algorithms* to evolve distributed problem solutions [Butz, 2015]. These solutions are represented by a set of rules, called *population of classifiers*. Each rule is composed of a condition (the subspace of the problem description in which the rule is applicable) and a problem solution (also called an action). A Learning Classifier system receives from the environment a sensorial state describing the current situation. The current situation can lead to the activation of a subset of rules, each rule proposing its own action and a choice has to be made between the different proposed actions. This choice is based

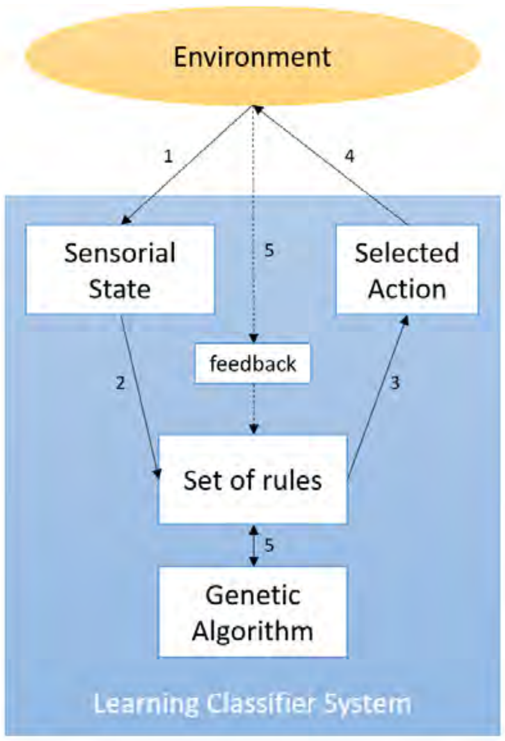

Figure 2.11 — Schematic view of an LCS system

on a *strength* value associated to each rule allowing to determine a probability of activation. Among the set of activated rules, the rule with the highest strength as the higher probability to be applied. This strength value is updated thanks to a feedback received from the environment while a genetic algorithm is responsible of creation and evolution of rules to only keep best rules. A Learning Classifier System then seeks to improve the reward that it receives from its environment. This improvement is based on both a genetic evolution and an history of its past interaction with the environment. The Figure 2.11 displays a schematic view of a Learning Classifier System.

There are many variations of Learning Classifier Systems but they can be split into two categories depending upon on either the genetic algorithm acts on a population of separate set of rules or a population of single set of rules:

▷ A *Pittsburgh-type* Learning Classifier System has a population of separate rule sets, where the genetic algorithm recombines and reproduces the best of these rule sets.

▷ In a *Michigan-style* Learning Classifier System there is only a single set of rules in a population and the algorithm's action focuses on selecting the best classifiers within that set. Michigan-style Learning Classifier Systems have two main types of fitness definitions: strength-based (e.g. ZCS [Wilson, 1994]) and accuracy-based (e.g. XCS [Wilson, 1995]).

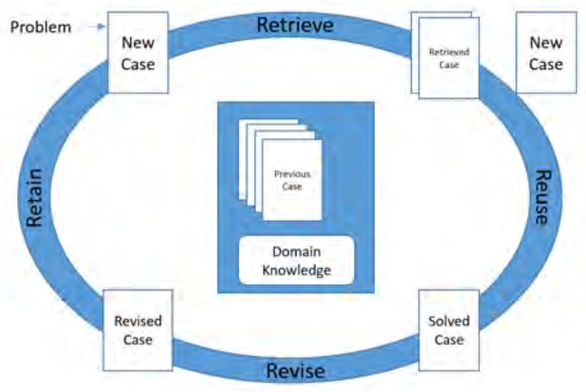

Figure 2.12 — Steps of a Case-based reasoning algorithms

2.6.2 Case-Based Learning

Case-Based Reasoning [Aamodt and Plaza, 1994] is a solving process of new problems based on the solutions of analogous past problems. A Case-Based Reasoning system is composed of a base of previous cases. A case is the association of a problem description and a representation of a solution. The process of case-based reasoning has been formalized as a four-step process:

- 1: **Retrieve** an analogous case based on a similarity function that has to be instantiated by the designer.
- 2: **Reuse** the previous case by applying its solution. This eventually could lead to the adaptation of the proposed solution thanks to an application function that has to be provided by the designer.
- 3: **Revise** the proposed solution. Once the solution has been applied, the Case-Based Reasoning system has to determine if either or not this solution is correct. This is performed by an expert (a human expert or another system able to evaluate the quality of the solution). If the solution is not correct, the Case-Based Reasoning system has to infer the reasons of this failure and eventually to modify its treatment. This functionality has to be instantiated during the conception of the system.
- 4: **Retain** the solution. Once the solution has been validated, the new case is created ans stored in the base.

By itself, the Case-Based Reasoning is not a learning algorithm as a huge part of its behaviour has to be instantiated by the designer. However, it proposes a general framework to conceive a system with the ability to enrich its knowledge at runtime.

2.7 Synthesis

In this chapter, we propose an overview of the field of machine learning highlighting different approaches, each with its own pros and cons. The literature is rich of proposals combining those different approaches into new ones. This results in a large quantity of variations of the same basic algorithm and a certain difficulty of implementation to a concrete application. From this first overview emerges some interesting avenues of research for our application.

Supervised learning algorithms, such as artificial neural networks, need to learn from examples, consisting in a set of labelled data. Then, they include in their definition a practical way to deal with user needs. Indeed those labelled data can directly refers to user preferences. A first axis of research is then to look toward applications that learns a mapping function from human examples. However, those algorithms are traditionally off-line, which means that their behaviour has two steps, first gathering the data, and then building a mapping function. This off-line behaviour prevents their usage in highly dynamic applications which is the case of our targeted systems, where user needs are supposed to be dynamic. A proposition could then consist in proposing an approach enabling on-line learning from human examples.

On the other side, reinforcement algorithms have naturally the ability to learn in real-time from the interaction with their environment. Their main drawback is the design of the feedback function, which is a known complex task, and the large number of cycles is necessary to reach an adequate behaviour. The feedback function should be designed in accordance with user needs. As those needs are dynamical, a proposition could consist in the real-time adjustment of a learning function under the guidance of a human user. This guidance could then reduce the need of exploration of such algorithms.

The last axis of research is the one proposed by the constructivist approach, which proposes to design systems that progressively build knowledge through interactions with its environment. The question left by this approach is what is the motor of these interactions. A proposition could then consist to use the interactions with users as a source of experiences.

All these ideas converge to a proposal: the design of a system that learns by interaction with humans. On the next chapter, we propose to study learning from demonstration, a paradigm to learn a control policy from demonstration performed by a human user.

3 Learning from Demonstration

*This chapter describes the **Learning from Demonstration** paradigm, a paradigm which proposes to learn a policy from the demonstrations performed by a human. In a first part, we highlight the inspirations and motivations of the Learning from Demonstration paradigm. Then, we present its usage from an engineering perspective, presenting the state of the art approaches. At last, we draw the requirement for enabling Learning From Demonstration in Ambient Robotics.*

THE design by hand of a control policy is a well-known complex task. To do so, the designer has to build a control policy that maps the world state to actions. This mapping policy involves a lot of knowledge on the world and its dynamics. Thus the *ad-hoc* design of a controller is restricted to highly skilled people. Furthermore, an *ad-hoc* policy restrained the system to an *a priori* set of skills and behaviours. Extending robot behaviour involves the same complexity as designing a new controller.

On the previous chapter, we presented an overview of the field of machine learning with the aim to learn such policy. From this overview, we propose to use human demonstrations as source of examples to guide the learning algorithm. Such approach has one major advantage, it is a bio-inspired approach that is natural for users. Indeed, imitation and learning from demonstrations are a natural way for human to learn and learning from other plays an important role in the development of our societies.

The study of the underlying mechanisms that enable humans and animals to acquire information or skills from another individual is a topic studied in various domains, from ethology [Heyes and Galef Jr, 1996] to cognitive sciences [Nadel and Butterworth, 1999]. This capacity to extend skills with a form social of learning has inspired roboticists [Schaal, 1999] trying to break the gap between robots and end-users.

Learning from Demonstration (LfD) [Argall et al., 2009] (also known as *imitation learning* or *programming by demonstration* [Billard et al., 2008]) is a paradigm that proposes to extract this policy from demonstrations performed by a teacher (also named *tutor*). The system acquires its autonomy by mimicking the demonstrated behaviours.

On this chapter, we study ***imitation*** as a learning paradigm to enable skills acquisition in human-robot applications. In the first section, we discuss of the nature of imitation and its study in natural and artificial systems. The second section focuses on *Learning from Demonstration* as a learning paradigm in robotics and discusses of key challenges in this field. The last section discusses of its application to ambient robotics.

3.1 Imitation: From Natural to Artificial

3.1.1 Imitation in Nature

The ability to *learn from others* is a powerful mechanism enabling individuals to share experiences which has a long history of research. Social learning (which includes imitation) enables the development and spreading of culture, ideas, belief and behaviour among a population from generations to generations [Dautenhahn, 2003].

Humans are natural born imitators and imitation plays an important role in our development. [Piaget, 1945], who studied the child's development, sees *imitation* as a step of the child development by which the child acquires models of the world. Here imitation is defined by *learning by seeing* which means that imitation is a tool allowing the acquisition of higher skills. As Piaget considers that intelligence is the capacity to abstract and model the world, *imitation* is then seen as the process allowing the transition between sensori-motor intelligence and imaged representation [Nadel, 1986]. [Blackrnore, 1999] sees imitation as a crucial media allowing the transfer of *memes*. *Memes* are cultural units of knowledge which play a role similar to genes in biological evolution. They are transferred from one individual to one other individual allowing the culture to be spread and evolve.

Nevertheless, humans are not the only animals to have the capacity to mimic. [Ferrari et al., 2006] show evidences of imitation in neonatal rhesus macaques. Facial expressions were demonstrated to a new born macaque which reproduces the observe behaviour (see figure 3.1). [Herman, 2002] demonstrates dolphin capacity to imitates a variety of behaviour and notably to perform vocal mimicry.

Tomasello suggests that while non-human primate can *emulate*, which means learning something about the environment from the observation of a demonstrator manipulating it, they are not capable of *true imitation* which requires not only to understand that the demonstrator is an animated agent interacting with the environment, but that this agent has goal and a *mind* [Tomasello, 1999]. For Tomasello, *true imitation* is a sign of a uniquely human mind.

This debate among scientists if either or not a particular species is capable of imitation is still prolific and illustrates the complexity of defining and comparing *imitation*.

3.1.2 Defining Imitation

Imitation is a ill-defined term with a particular definition in each domain in which it is studied. [Dautenhahn, 2003] discusses of what is the nature of imitation and what are the differences with other forms of social learning.

[Zentall, 1996] [Zentall, 2001] differs imitation from:

- ▷ **Contagion** where unlearned species-typical behaviour spreads among a group of individual, for example yawning behaviour.
- ▷ **Social facilitation** or **social enhancement** where the presence of conspecifics encourages similar behaviour.

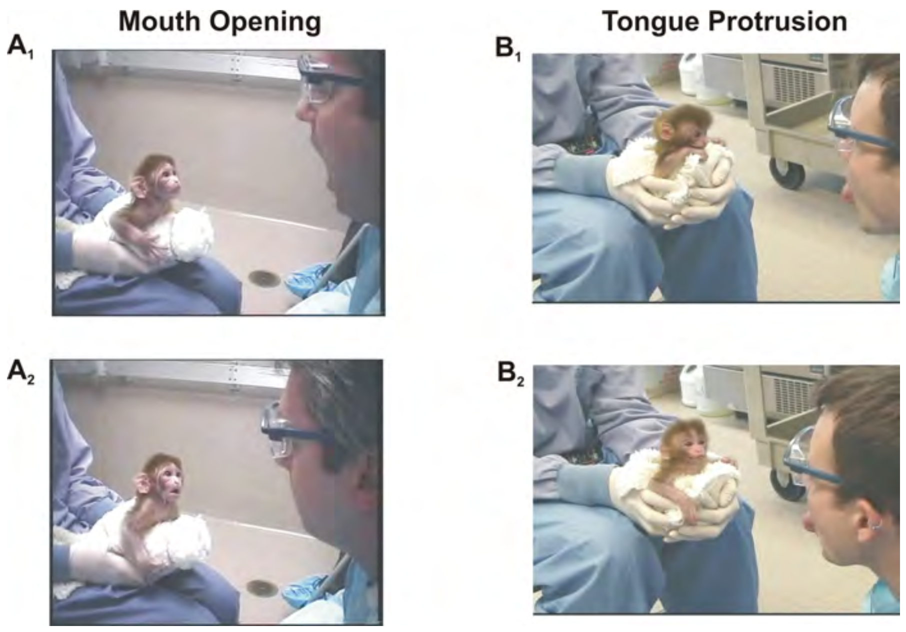

Figure 3.1 — Neonatal imitation in Rhesus Macaques [Ferrari et al., 2006]. In both situations, the macaque imitates the observed behaviours

- ▷ **Local enhancement** where the attention of the learner is drawn to a place or a location due to activities of the demonstrator.
- ▷ **Observational conditioning** (Pavlovian association) where the demonstrator gives only social clue of the task to perform, for example with positive or negative feedbacks.

[Thorndike, 1898] defined imitation as *any situation in which animals from an act witnessed learn to do an act*. Thorndike definition does not imply novelty of the learnt behaviour.

[Thorpe, 1956] defined *true imitation* as the *copying of a novel or otherwise improbable act or utterance, or some act for which there is clearly no instinctive tendency*.

According to [Mitchell, 1987] imitations occur when : something C (the copy) is produced by an organism and/or machine, where

- ▷ C is similar to something else M (the model)
- ▷ The registration (or perception) of M is necessary for the production of C, and
- ▷ C is designed to be ***similar*** to M.

The similarity depends on what part of the demonstration is mimicked. [Call and Carpenter, 2002] makes two kinds of distinction level for evaluating similarity.

The first distinction level is either or not the learner understands and adopts the goal of the observed behaviour. On the one hand, if the imitator understands the goal and copies the action, it will reproduce the results (*imitation*) or not reproduce it (*failed imitation*). On the other hand, if the imitator understands the goal but does not copies the action, it can still try to reproduce the result (*goal emulation*). At last if the imitator does not understand or adopt the goal, it has again the subsequent choice of copying the action or not, and producing the result or not. Then, they distinguish:

- *Mimicry*: copying the action with or without producing the same result.
- *Emulation*: not copying the action but reproducing the result.
- *Failed emulation*: (or other social or non-social learning) otherwise.

[Call and Carpenter, 2002] makes a second distinction between goals, actions and results which can be explained through the coconut problem. Let's assume that two humans A and B are under a palm tree. A demonstrates the following behaviour: climbing up the palm tree, picking a coconut, climbing down the tree and eating the coconuts. What will be a successful imitation for B depend on whether success is judged on the level of ***actions***, ***results*** or ***goals***:

- Imitation based on ***actions*** means that B will perform exactly the same sequence of actions: climbing up the palm tree, picking a coconut, climbing down the tree and eating the coconut.
- Imitation based on ***results*** means that B will do anything to make the coconut down the palm tree, including shaking or cutting it, resulting in catching a coconut from the palm tree and eat it.
- Imitation based on ***goals*** means that B has to *eat a coconut*, then he can use his car to go to the supermarket, buy a coconut and eat it.

Each one of those imitations is either a success or a failure, depending on which criterion is used for evaluating good imitation.

3.1.3 Big Five Questions

Five central questions (aslo called *Big Five*) have been identified that need to be addressed by scientists interested in designing experiments on imitation [Dautenhahn, 2003] [Dautenhahn and Nehaniv, 2002] :

- **Who** to imitate: first is the choice of the model (the one to be imitated).
- **When** to imitate: the second is determining when imitation has to be done. There are typically two types in literature if whether the imitation is immediate leading to synchronous behaviours, or deferred which means that the imitated behaviour might occur even in the absence of the model.
- **What** to imitate: the third question directly refers to [Call and Carpenter, 2002] distinction between goals, actions and results and which part of the demonstration has to be replicated.
- **How** to imitate: the ***how*** question addresses the problematic of generating an appropriate mapping between the model behaviour and the imitator's one.

- ▷ What is a **successful** imitation : at last, but certainly not least, one need to be able to distinguish good imitation from bad imitation. Then, good metrics must be clearly identified.

3.1.4 Imitation in Artificial Systems : Motivations

This work about the study of imitation has inspired AI researchers. The problem of learning a mapping between world states and actions lies in the heart of many control applications. This mapping policy is hard to hand-craft. Thus, they propose that rather than requiring users to analytically decompose and manually program a desired behaviour, an appropriate robot controller can be derived from observations of a human's own performance thereof. This offers some advantages:

- ▷ The system is more easily **extensible** and **adaptable** to novel situations.
- ▷ Users can **share their expertise** through the natural process of demonstration.
- ▷ The approach **doesn't require** users to have programming abilities.

Another argument in favour of learning from demonstration came with the Moravec's paradox. [Moravec, 1988] proposes that while "*it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility*". Some humans skills are complex to reverse engineer. *Learning from Demonstration* appears to be an easier way to mimic human natural skills. On the rest of this chapter, we present Learning from Demonstration in an engineering perspective.

3.2 Imitation as a Learning Paradigm: Learning from Demonstration

Learning from Demonstration (LfD) is a well established technique in robotics and the field has been reviewed in two surveys from [Billard et al., 2008]) and [Argall et al., 2009]. [Billing and Hellström, 2010] also proposed a complete formalism for Learning from Demonstration. On this section, we highlight the key concepts of Learning from Demonstration and introduce the formalism that is going to be used on this thesis.

3.2.1 Problem Statement

Learning from Demonstration is a subset of Supervised Learning where a learner is presented with labelled dataset and learns an approximation of the function which produces the data. We use elements of formalisation from [Argall et al., 2009] to describe the problem. *Learning from Demonstration* is a process involving two entities: a ***tutor*** and a ***learner***.

A *tutor* evolves in a world in which he realises an observation Ω. The tutor have available of a set of actions A (A could be empty). The tutor follows a politic π_{tutor} that associates to any world state Ω a particular action $a \in A$ (definition 1).

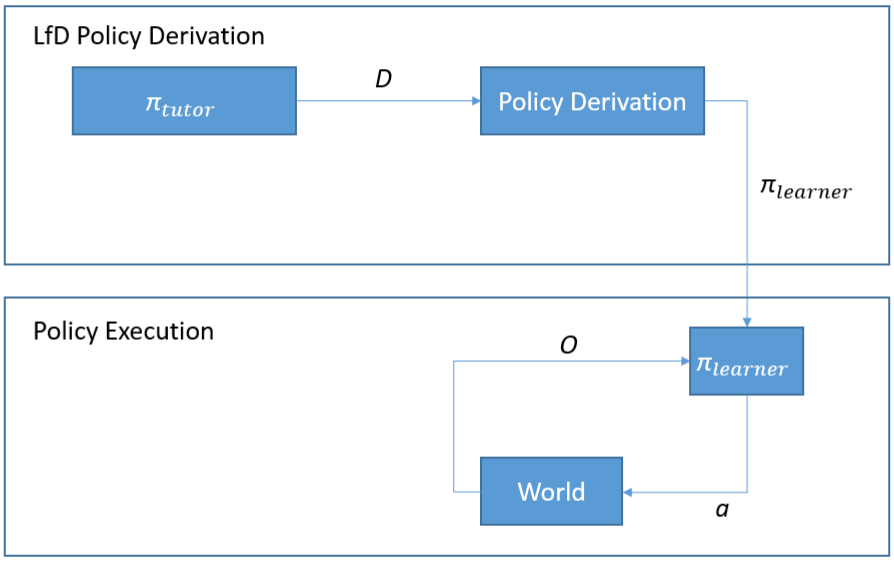

Figure 3.2 — Control policy derivation and execution [Argall et al., 2009]

Definition 1. $\pi_{tutor} : \Omega \to a$

A *learner* possesses a set of observations O (named *observation space*) on the space states Ω and its own set of actions B. The learner follows a politic $\pi_{learner}$ that associates to any observation space O a particular action $b \in B$ in order to produce a behaviour that is similar to the observed one (definition 2). The actions are either low-level motion command or high-level behaviours depending on the desired level of control to practice by the learner. It may also be a set of actions, depending on the application domain.

Definition 2. $\pi_{learner} : O \to b$ *such as* $\pi_{learner} \cong \pi_{tutor}$

To construct its policy, the learner has at his disposal a set of **demonstrations** (also called training dataset in literature) D which are recorded from the tutor activity. A demonstration $d_j \in D$ consists in k_j pairs of observation ω_j and action a_j (definition 3). Those sequences of state-action allow the learner to both construct and validate its policy.

Definition 3. $d_j : \{(\omega_j^i, a_j^i)\}, \omega_j^i \in \Omega, a_j^i \in A, i = 0....k_j$

Given a particular set of demonstrations D, the learner derives a new policy $\pi_{learner}$ which enables the learner to select an action a based on the current state O (figure 3.2).

3.2.2 The Correspondence Problem

In cases where $O = \Omega$, the identity mapping, the tutor and the learner possess the same observations on the world. The same remark has to be made on the two sets of actions A and B. If $A = B$, both the learner and the tutor have identical skills.

Nevertheless, in many cases, such assumptions cannot be made. It is particularly true in real world problems where the world is observed through sensors. The tutor and the

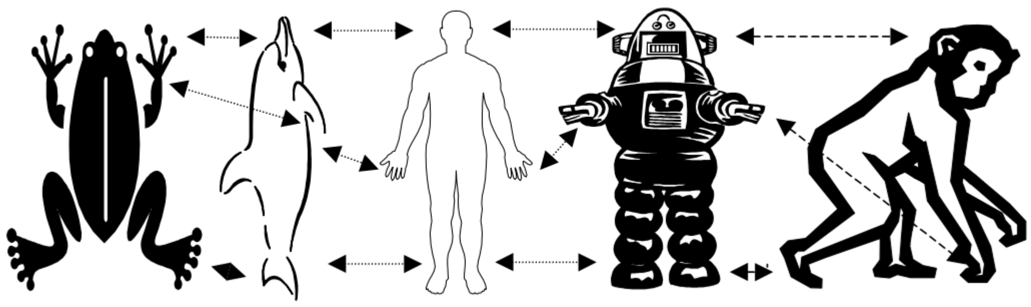

Figure 3.3 — Structural homologies among tetrapod animals/artifacts [Dautenhahn and Nehaniv, 2002]

learner could have a different sensory-motor apparatus. This is what is expressed as *the correspondence problem*. [Dautenhahn and Nehaniv, 2002] define the correspondence problem as follow :

Given an imitator (a biological or artificial system) trying to imitate a model (the biological or artificial system to be imitated), how can the imitator identify, generate and evaluate appropriate mappings (perceptual, behavioural, cognitive) between its own behaviour and the behaviour of the model?

An illustration of structural homologies can be found in figure 3.3. For example, the grip of a robot could grab objects. Then it presents some similarities to the human hand. However, the grip has no sense of touch and thus cannot make a difference between a soft object and a rough object.

The dissimilarity in the sensory-motor apparatus could result in failures in imitation, has the learner is unable to sense a particular part of the demonstration or perform a particular action.

Solving the correspondence problem in a human-robot application is a non trivial problem. The solutions often result in particular design choice in the design of an LfD application. The rest of this section discusses of those different design choices.

3.2.3 Designing Imitation

3.2.3.1 Gathering the Examples

On the previous section, we have identified the correspondence problem as the issue with the identification of a mapping between the teacher and the learner that allows the transfer of information from one to another. [Argall et al., 2009] split the correspondence problem in two mapping functions (Figure 3.4):

- ▷ The *Record mapping* function refers to the mapping between the states/actions experienced by the teacher during the demonstration execution and those that are recorded.
- ▷ The *Embodiment mapping* function refers to the mapping between the states/actions

recorded within the data set and those that the learner would observe/execute.

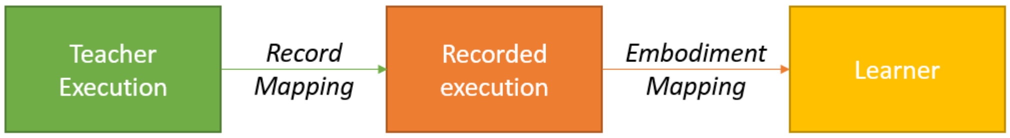

Figure 3.4 — Mapping a teacher execution to the learner [Argall et al., 2009]

For each of those mapping, we can consider two cases, if either a mapping function is effectively needed or not. In the case of the record mapping, if no mapping function is needed, the states/actions are directly recorded within the dataset without transformation. Otherwise, a transformation function will be used to encoded states/actions into the dataset. The same classification can be made with embodiment mapping, if either or not a mapping function is required to map states/actions. If no embodiment mapping is required, the gathering process is called *Demonstration*. On the other side, if an embodiment mapping is required, the gathering process is called *Imitation*. Depending on either the learner and the teacher possess a similar sensory-motor apparatus, the choice of the approach to gather the example differs:

- ▷ **Teleoperation** is a demonstration technique in which the teacher directly acts on the learner and the demonstration is recorded through the learner observations. The record mapping and embodiment mapping are direct.
- ▷ **Shadowing** is a demonstration technique in which the learner records the execution using its own observations to mimic the teacher as the teacher executes the task. Here, a record mapping function is needed to map teacher actions to learner actions, but no embodiment mapping is required as the learner uses its own observations.
- ▷ **Sensor on Teacher** is an imitation technique in which the teacher's body is equipped with sensors to record its activity. Here, the record mapping is direct, but the embodiment mapping requires a mapping function.
- ▷ **External Observation** is an imitation technique in which the teacher activity is observed through external sensors. With such approach, there is a need for a non-direct record mapping function and an embodiment mapping function.

This classification is sum-up in the Figure 3.5.

3.2.3.2 Deriving a Policy

Once demonstrations are collectable, the learner has to infer a policy from those examples. [Argall et al., 2009] categorize approaches to learn from demonstration in three classes (figure 3.6). In this section, we provide a description of each class with some applications and we present their pros and cons.

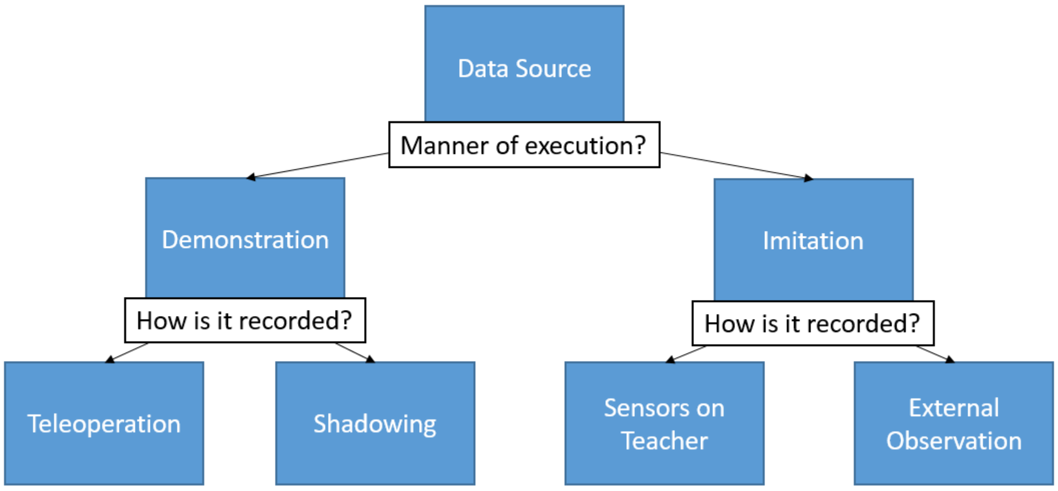

Figure 3.5 — Categorization of approaches to gather the demonstrations [Argall et al., 2009]

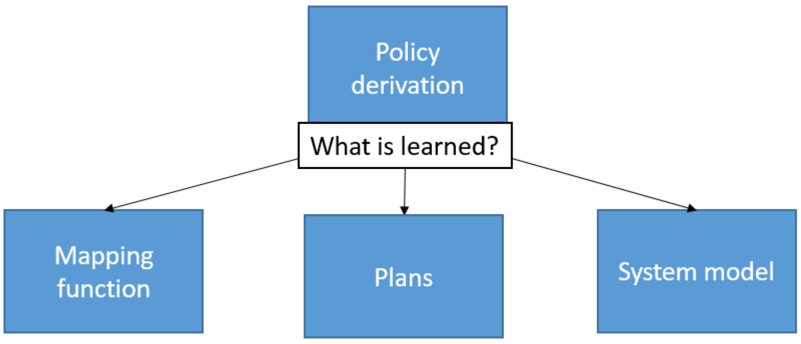

Figure 3.6 — Policy derivation categorisation according to [Argall et al., 2009]

▷ Learning a ***Mapping function***: those techniques calculate a function that associates the current state to action $f() : O \to a \in A$. Its goal is to reproduce the underlying teacher policy while generalising to unseen situations. Those techniques use *classification* and/or *regression*. For example, [Mitić and Miljković, 2014] use a combination of *Neural Networks* and *Learning from Demonstration* to teach visual control of a nonholonomic mobile robot.

- **pros**: with those approaches, the tutor directly labels the different situations during the demonstration process with the adequate actions to perform. As we intend to use the interactivity of ambient system, learning a mapping function appears to be adequate. An example is [Chernova and Veloso, 2007] who use a set of Gaussian Mixture Models to teach a car how to drive in a busy road. Each Gaussian Mixture Model is incrementally trained with demonstrations gathered from the observation of a human performance driving the virtual car through a keyboard.

  - **cons**: traditional *classification* and *regression* techniques, such as k-Nearest Neighbors (kNN) [Saunders et al., 2006] are separated in two phases: first, gathering the examples, and then production of the mapping function. Any change in the user behaviour involves re-performing the whole training process which could be time-greedy and unsatisfying for end-users. Moreover, those techniques are parametrized, and then need to be tuned for each application.

- ▷ Learning ***System model***: those techniques use a state transition model of the world from which they derive a policy. *Reinforcement learning* techniques are a typical case of system modelling where any state is associated with a reward function. While the usage of reinforcement learning is traditionally limited by the need of large number of environment samples to reach a desirable behaviour, its combination with the LfD paradigm allows to reduce the space search improving its performances. [Brys et al., 2015] illustrate the gain of performance by comparing performance of reinforcement learning algorithm with or without usage of the LfD paradigm.

  - **pros**: the main advantage of *system models*, and more precisely of using *reinforcement learning* techniques is their on-line capacity to learn and their capacity to find optimal behaviours.
  - **cons**: while their on-line capacity to learn is highly desirable in our context, the design of a feedback function is a well-known complex task that has to be hand-crafted for each application. Some approaches, called *inverse reinforcement learning*, propose to use LfD not only to reduce the number of samples but also to infer the feedback function. For example, Knox et al [Knox et al., 2013] taught to a robot different kinds of behaviour such as keeping conversational distance or aiming towards the human. They create a predictive model of human reinforcement feedbacks and use this model to increase reinforcement learning algorithms. While they show interesting results in learning a particular task, those approaches are not robust to changes in the task to perform or the system composition and any of those changes involve re-performing the whole learning process.

- ▷ Learning ***Plans***: instead of directly mapping states to actions, those approaches represent the desired behaviour as a plan. The *plan* is a sequence of actions that leads from an initial state to the final goal state. For example [Mollard et al., 2015] propose an approach using plans for robot instruction for assembly task. They use low-level demonstrations to learn high-level relational plan of the task. Then, they use a user graphical interface through which the user can interact with the plan to correct both high-level plan or low-level geometrical knowledge of the task.

  - **pros**: plans offer the possibility to draw a readable map of the controller behaviour and then enable intelligibility of the learnt behaviour. Furthermore, they allow to identify goals and to propose different behaviours to reach them.
  - **cons**: the main drawback in using planification algorithms is the complexity of re-planification. Some approaches, such as the one of [Mollard et al., 2015], propose tools to assist the user in the re-planning process. However, such approach prevent their online usage.

Each of those approaches have pros and cons which must be considered during the design of an LfD application. For the approaches that learn a mapping function, the LfD paradigm offers a natural way to gather the example required by learning algorithms. To those which learn a system model, the LfD paradigm allows to reduce the search space increasing the convergence speed of algorithms. On the next section, we present some recent applications of the LfD paradigm.

3.3 Some Applications of the LfD Paradigm

In this section, we propose an overview of some recent applications to illustrate the variety of tasks and applications that are designed using the LfD paradigm. For a more complete overview of past applications, the reader can refer to [Billard et al., 2008] and [Argall et al., 2009].

[DeVautl et al., 2015] use a combination of the LfD paradigm with genetic algorithms to train a set of autonomous robots in parallel in order to perform a navigation task. The LfD is here used to reduce the time required to produce sufficiently rich training sets for learning.

[Pagliuca and Nolfi, 2015] propose an approach combining LfD paradigm with learning from experiences. In this approach, the LfD paradigm allows the agent to drive the learning process toward similar solutions to the demonstration, while the agent stays free to differentiate to maximize its performance.

[Knox et al., 2013] proposes a framework called TAMER (Training an Agent Manually via Evaluative Reinforcement) for learning from numeric human feedbacks. An agent learns through quantitative feedbacks demonstrated by a human user to approximate a feedback function which is then used by a reinforcement algorithm. The results tend to show that the usage of the LfD approach reduce the time for reinforcement learning algorithms to converge toward a satisfying solution.

[Carrera et al., 2015] use the LfD approach to teach an autonomous underwater vehicle in an underwater valve turning task. The LfD paradigm is here used to infer a model of positional and force profiles encoded as a mixture of dynamical systems, which is used to reproduce the task satisfying both the positional and force profiles.

[Bruno et al., 2014] use LfD to transfer sensory-motor skills to a robotic platforms. The combination of imitation and exploration strategies is used to transfer sensory-motor skills to a robotic platforms. The approach is tested on a reaching task performed with a Barrett WAM manipulator.

[Fonooni et al., 2015] apply ant colony optimization algorithms for high-level behaviour learning and reproduction from demonstrations. They use a combination of ant colony optimization algorithms, Semantic Networks and Spreading Activation mechanisms to teach to a robot to collect objects with particular shapes, and then to place them in the designated baskets regardless of their color and size.

The LfD paradigm is used to learn a variety of task for various robots, mainly in combination with other learning approaches to reduces the need of exploration and examples. However, to our knowledge, it has never been applied to the context of Ambient

Systems. In the next section, we discuss of imitation learning and ubiquitous systems.

3.4 Imitation Learning and Ubiquitous Systems: Requirements and Proposals

This section ends our overview of the *Learning from Demonstration* paradigm. *Learning from Demonstration* offers undeniable advantages for those who took interest in designing interactive applications requiring few user skills. The paradigm has been used on many applications and is still studied nowadays.

Now that the LfD field has been presented, we want to study the application of the LfD paradigm to Ambient Systems. A first thing to do is to answer to the five main questions (see section 3.1.3). While initially those questions were thought for designing experiments in imitation with animals, they are a good way to start designing the requirements for a LfD applications. Thus, we propose to answer to those questions in the context of Extreme Sensitive Robotics.

- ▻ The first question to answer is **Who** to imitate. Ambient systems are used by many users and must provide services adapted to them. Then, the system has to imitate every user who interact with it. This imitation should be *proactive*, which mean that users needs must be anticipated.
- ▻ Now that we know who we want to imitate, we need to consider **What** has to be imitated. As we want to design a control system, we need to imitate users actions. In ambient systems, users can interact with the system through a variety of *medium*, from switches to most advanced user machine interfaces. However, no matter what is the medium used, the control action is the message that is sent to the effector to change its current state. This message is what we want to imitate.
- ▻ The third question is **When** to imitate. Due to the hyper-interactivity of ambient systems, users can have day to day interactions with the system and users can act at any time. If an ambient system uses imitation to adapt its policy, this adaptation has to be performed in real time. Then, imitation has to occur each time a user realises an action that was not anticipated by the system.
- ▻ When a user perform an action, it expresses two things: that the current system state is not satisfying him and what is the correct action to change the system's state. Then, to solve the **How** question, we propose to correlate the performance of an action to the current environmental state. This correlation results in building a mapping function associating to a state description the correct control action to perform. Then, when a similar situation occur, the system can pro-actively apply the correct action.
- ▻ The last question is what makes a **successful** imitation. The notion of metric to evaluate imitation is not trivial. Generally, it depends on what task has to be imitated. In our system, we want to imitate any kind of user. Then, no *a priori* can be made on the task to perform. However, we need to evaluate if either or not the system is in a satisfactory

state. As the aim of the control system is to pro-actively perform control action on behalf of users, then the system is performing well if no user is acting on it to change its state.

The answer to those questions allows to highlight two hypothesis. One first hypothesis is that whenever a user has to act on a system, the service provided by the system was not satisfying him any more. The action performed by the user can be seen as a demonstration, associating to the current system state the adequate action to perform. This demonstration is a qualitative feedback expressing that the previous action performed or maintained by the system was not the desired one and also provides the adequate action to perform. A second hypothesis is that when a user performs an action in a certain context, the same action is desired in a similar context. Then building a mapping function associating the context to actions appears to be a good control policy.

The building of this mapping function should respect the properties of ambient systems, and then must deal with the openness properties and provides real-time learning. Each action from the tutor should be dynamically and autonomously integrated in the learning process allowing our system to be truly self-adaptive. The openness property is particularly challenging because it has to be considered that the observation space O (as described in section 3.2.1) is non finite. The system can pass from situations where data is plentiful to situations where data are missing.

To be truly applicable with any kind of devices, another requirement in our application is genericity. Genericity has two impacts on requirement. The first is that we have to consider that the action space A (section 3.2.1) is unknown and has to be learnt. Thus, any device can dynamically learn its possible actions. The other impact is on the technique to gather demonstrations (see section 3.2.3.1). To be truly generic, the learning technique must not be dependent of a particular approach to gather example. Then, we consider *teleoperation* has the most generic and natural way to perform demonstrations. Indeed, teleoperation involves a direct control of users over the learning device (no matter which medium is used to perform this control) and a direct mapping with the device sensors. Teleoperation does not prevent the use of other mapping techniques, like shadowing or sensor on teacher, as those techniques operate a mapping to the action space.

The next chapter introduces our contribution by presenting the methodology adopted to design a system enabling realt-time learning from demonstration in ambient systems.

Self-Organization of Robotic Devices Through Demonstrations

Contribution

4 Designing Emergence

This chapter introduces the key concepts of emergence and self-organization to face complexity. It presents the Multi-Agent paradigm and the ***Adaptive Multi-Agent Systems*** *approach. This chapter shows the adequacy of the AMAS approach for our application.*

THE complexity of the ambient systems induces severe limitations on the design process of a new system. However, these limitations are not restrictive to the problem addressed in this thesis, but can be seen as a reflection of the barrier that complexity imposes to our design abilities.

Nowadays, artificial systems tend to be more and more dynamical, open, distributed and multi-users. One just has to look at the revolutions of the Internet of Things and cloud computing (to name only these two examples) in order to perceive the challenges of tomorrow. Just like technologies are evolving fast, the whole society is progressing and it is becoming harder and harder to anticipate what will be the requirements of tomorrow.

It is difficult to characterize the functionality offered by a system and the task to perform may be incompletely specified, or even dynamical. Anticipating all the system's interactions and evolutions is challenging, particularly for systems evolving in the real world with humans.

Traditional approaches are frontally facing complexity, trying to divide the problem recursively into smaller sub-problems. Thus, each new problem and sub-problem bring into movement a set of experts to design an *ad hoc* solution, at the expense of the system genericity. Any change in the problem involves re-performing the whole design process.

Another way of thinking has emerged to face complexity, offering a conceptual break in our way to design artificial systems. As complexity is out of reach, no component of the system should have to face the problem in its whole. Hence, the design of an artificial system has to focus to the local micro-level (where complexity is lowered) in order to induce the emergence of the desired phenomenon at the macro-level.

4.1 Emerging Phenomena

Emergence is a non-trivial term subject to an active debate. While this notion is central to the study of complex systems, it has no formal definition that is unanimous. We commonly qualify as emergent a phenomenon where global behaviour arises from the interactions between the local parts of the system [De Wolf and Holvoet, 2005].

A definition of emergence from the computer science perspective is proposed by [Di Marzo Serugendo et al., 2011a]. They define emergence as "the process that cause a software system to produce an *emergent phenomenon*". "An *emergent phenomenon* produced by a software is an interpretation of an attractor the system has converged into, which is practically unpredictable given the functionality of system component".

The phenomena of emergence is easily observable in nature. Its best-known representatives are surely ants. The behaviour of a single ant is rather simple to describe as a reactive behaviour that consists in moving randomly until a food spot or pheromones are found. Ants behaviour does not involve any advanced cognitive abilities, advanced social skills or planning capacities. However, regrouped in colonies of many individuals, the colony, as a whole, exhibit complex strategies such as a resilient path-finding strategy to find the shortest path to food [Goss et al., 1989].

Lift your head in autumn and you will probably see a ballet of birds dancing complex choreographies. Flocking is another notable example of emergent behaviour found in nature. Such as ants, birds do not have individually a sophisticated behaviour. However, evolving in large groups, remarkable patterns are easily observable which cannot be deduced by the study of individuals [Reynolds, 1987].

Those phenomena are difficult to design with the traditional centralized approach of problem decomposition, but constructing those phenomena through a bottom-up design focusing on the behaviour and interaction of individuals often result in simple rules. For example, flocking can be obtained with only three simple rules:

- 1 - **Separation** - a bird turns to avoid another bird which gets too close.
- 2 - **Alignment** - a bird tends to turn so that it is moving in the same direction that nearby birds are moving.
- 3 - **Cohesion** - a bird moves towards other nearby birds (unless another bird is too close).

However, limiting the emergence to a separation between the macro-level and the micro-level is not sufficient. Indeed, each global activity of any system is the result of the interaction of its parts, such as the movement of watch hands is the result of the interaction between the different wheels and other components of the watch. [Goldstein, 1999] claims that other criteria have to be taken into account. Firstly, the *dynamical* aspect of the phenomenon, *emergent phenomena are not pre-given wholes but arise as a complex system evolves over time*. Then, emerging phenomena are *coherent*, they tend to maintain some sense of identity over time. An emergent phenomenon appears during the evolution of the system, and is maintained enough over time to have its own identity.

To those criteria is added the *radical novelty* of the phenomenon, which is probably the heart of emergence. *Emergents have features that are not previously observed in the complex system under observation*. This novelty is not predictable from the micro-level which has no explicit representation of the global behaviour.

The differentiation between the macro-level and micro-level induces another criterion for emergence: the *decentralization* of the control. The macro-level is intangible and then not directly controllable by an external entity (such as a supervisor), the control is only possible by the entities at the micro-level. But no entity at the micro-level has a global control on the macro-level. This criterion is the source of the popular expression to describe the emergence, "the whole is more than the sum of its parts".

All those characteristics describing the emergence highlight a fundamental (and highly discussed) aspect of the emergence. The emergence is an ostensible phenomenon, which means that it is only recognizable by showing itself. Thus, the emergence is dependent of the knowledge of its observer. An expert which acquires enough knowledge on the behaviour of a complex system to a point where he can unroll with exactitude the chain of causality between the micro and macro levels, would not see a phenomenon as emergent as the radical novelty would be missing. Emergence is a matter of observation and observer.

However, the emergence can be seen as an answer to our own limitation to understand and track the multitude of interactions that occur in complex systems by isolating and studying the entity at the micro-level. Consequently, there has been a growing interest about emergence as a design paradigm in artificial systems. A key component of the approach is that, although a system can be simple to design, it can exhibit complex functionalities that emerge from the interactions between its parts. On the rest of this chapter, we present the tool and concept to build systems with emerging functionalities, beginning with Multi-Agent Systems.

4.2 Multi-Agent Systems

Multi-Agent Systems (MAS) are computerized systems composed of multiple interacting and autonomous entities, the agents, within a common environment. Each agent has only a partial view of its environment. MASs offer a methodological way to study complex systems with a bottom-up approach. MASs are used in many different domains, from collective problems solving to the study of collective behaviours. The MAS paradigm proposes to focus on the design of agents and their collective behaviours leading to the realisation of a particular task. This distribution of tasks inside a MASs makes them highly suitable to overcome a greater complexity than the complexity apprehended by conventional methods. In this section, we present the key concepts behind Multi-Agent Systems.

4.2.1 The Agent

There are many definitions of the term *agent* as well as many paradigms using it. A commonly accepted definition is that *"an agent is a computer system that is situated in some environment, and that is capable of autonomous actions in this environment in order to meet its design objectives"* [Weiss, 1999].

The definition has been enriched by [Ferber, 1999], notably by adding a locality criterion. Ferber defines an agent as *"an autonomous physical or virtual entity able to act (or communicate) in a given environment given local perceptions and partial knowledge. An agent acts in order to reach*

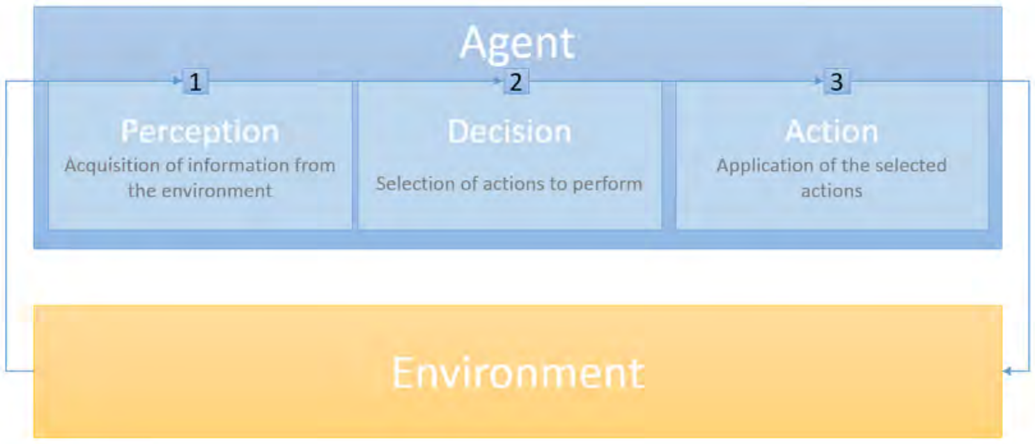
Agent
1
Perception
Acquisition of information from the environment
2
Decision
Selection of actions to perform
3
Action
Application of the selected actions
Environment

Figure 4.1 — An agent's lifecyle

a local objective given its local competence".

This definition highlights the fundamental properties of an agent:

▷ An agent is **autonomous**, which means that it is the only one to control its behaviour. This implies that the choice to act or not is only driven by the agent's own behaviour. The agent's capacity to say "no" (to choose not to act) makes a concrete differentiation between an agent and a sub-program.

▷ An agent evolves in an **environment** (physical or virtual) on which it is able to locally perceive information and locally act. An intuitive definition would be that the environment of an agent is everything that is external to the agent and which can be perceived by the agent, including the other agents (the environment is described in section 5.2). This environment acts as the interaction medium.

▷ An agent is able to **interact** and **communicate** with other agents either directly or through the environment.

▷ An agent possesses a **partial** knowledge of this environment.

▷ An agent possesses its own **resources** and **skills**.

The agent's behaviour is ruled by a three steps looping lifecycle of *Perception-Decision-Action* (see Figure 4.1):

▷ 1: *Perception* is the process during which the agent acquires information from its environment and updates its internal state.

▷ 2: *Decision* is the process during which the agent decides of actions to perform. This decision is based on its local perceptions, its internal knowledge and its own objectives.

▷ 3: *Action* is the process during which the agent applies the actions.

Beside those common properties, agents possess other characteristics that enable their differentiation [Di Marzo Serugendo et al., 2011b]:

- ▷ **Reactive/Proactive**: An agent is said **reactive** when its actions are triggered by events that occur in its environment as a *reflex behaviour*. The trigger rules are dependent of the agent's perceptions and its internal state. Those kinds of agents have generally few or no memory. On the opposite there are **proactive** agents. Those agents are able to modify their objectives and create new ones. They are also refereed as *cognitive* agents as they often involve complex learning algorithms. There is no concrete frontier between reactive and proactive agents. They are the two extremes of a linear axis to categorise agents from their granularity. The reactive agents are less complex, and so they are usually numerous, each agent focusing on a simple task. The system is said to have a *fine-grained* granularity. On contrary, proactive agents can process more complex tasks, involving that the system needs less agents to reach its objectives. The system is said to have a *coarse-grained* granularity.

- ▷ **Situated/Social**: An agent is said **situated** if its perceptions and its communications skills are conditioned by its localisation in the environment. On the opposite, the agent is said **social** if its perceptions and communications skills are not dependent of its localisation. Agents can directly interact without requiring localisation condition. Here too, there is no concrete frontier between situated and social agent.

Those characteristics are not exclusive but they illustrate the expressiveness of the agent based modelling. A key component of agent based modelling is the environment, which will be discussed in the next section.

4.2.2 The Environment

The notion of environment is crucial in MASs. Indeed, the environment is not only a source of information, but also the medium by which those agents act and interact. While being a key component of MASs, the environment lacks of formal definition which reaches a consensus inside the MAS community [Weyns et al., 2005]. Intuitively, the environment of an entity can be described as everything which is not this entity. Depending of the adopted point of view, different environments can be identified. In a MAS, we can either adopt the MAS's point of view (the system is viewed at its *macro-level*) or an agent's point of view (the system is viewed at its *micro-level*).

From the system's point of view, the environment is everything that is outside of the system.

From the agent's point of view, the environment is not only a part of the MAS's environment, but also the other agents. The agent's environment can be split in two components : the *physical* component of the agent's environment, describing what the agent can perceive and how it can act and the *social* component of the agent's environment, describing with which agents it can interact. Then, a MAS can be either *homogeneous*, composed of agents with the same capacities, or *heterogeneous*, composed of agents with different skills.

[Russell and Norvig, 1995] propose four characteristics to describe the environment of an agent:

- **Accessible/Inaccessible**: the agent's environment is accessible by an agent if the agent is able to perceive all the information required for its task.
- **Discrete/Continuous**: the agent's environment is discrete if it possesses a finite number of distinct states.
- **Deterministic/Non deterministic**: the agent's environment is deterministic if its evolution consecutive to an action is only dependent of its current state.
- **Dynamic/Static**: the agent's environment is dynamic if it evolves despite the agent's inactivity or during its deliberation.

4.2.3 Properties of Multi-Agent Systems

In MASs, each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint. However, all the required knowledge and competences required for solving the problem are still present, distributed among the system. Thanks to this distribution, the MAS paradigm seems particularly suited to problems with a natural distribution such as ambient systems.

A MAS is *open* if agents can appear and disappear during system's lifetime. On the opposite case, the system is said to be *closed*. The appearance of an agent is most of the time the result of the decision of an existing agent while its disappearance can be the decision of an agent (which then commits a form of suicide), or initiated by the environment.

Another property is the absence of external or global control system. The control is distributed inside each agent, and each agent is the only one to be responsible of its behaviour.

Some approaches from the literature can claim to be MASs without respecting the previous properties. For example, one could decompose a problem into a set of sub-problems (each sub-problem named an agent). Once each "agent" has solved its own problem, the results are lift to a controller entity (also named an agent) which recomposes a solution from the sub-solutions. From our point of view, such approach, which does not respect the decentralization of control and knowledge among the agents, is not a MAS approach but rather an agent based modelling approach, the two approaches are not focusing on the same issues.

4.2.4 self-organization in MAS

The MAS and its environment are coupled. The actions of MAS alter its environment and the perception of a modification by an agent acts as a feedback signal on the MAS. The activity of the MAS influences its environment, and the MAS activity is influenced by its environment. Consequently, the inner organization of the MAS (its structures and its interactions) is dynamically altered as each agent locally self-adapts to changes in its environment: the system is self-organizing.

The concept of self-organization is not inherent to MAS, as it has been studied since the ancient Greek in a variety of domain. [Di Marzo Serugendo et al., 2011a] propose to define this concept from a software engineering point of view.

Definition: *self-organization* is the process enabling a software to dynamically alter its internal organization (structure and functionality) during its execution time without any explicit external directing mechanism.

A difference is made between weak self-organization, where the control of the inner organization is centralized by an internal entity, and strong self-organization, where this control is decentralized.

Definition: *Strong self-organizing* systems are systems where self-organization process decision is distributed locally among the system components without involving any centralized point of control (either internal or external).

Definition: *Weak self-organizing systems* are systems where, from an internal point of view, self-organization is internally administrated by a centralized point of planning and control.

Many mechanisms to enable self-organization can be found in literatures. Stigmergy, for example, is a mechanism for indirect coordination between agents trough modification of the environment [Bourjot et al., 2011]. Here there is no centralized control of the self-organization process. On the opposite, the holonic approach [Calabrese et al., 2010] proposes to build hierarchical layers of agent where agents of the higher level practice a direct control on the sub-layer.

From those definitions, we can identify two processes in a MAS: the one which operates the self-organization and the one which realises the function for which the system has been designed to. However, these two processes are intricated in a way that it is difficult to tell if one interaction belongs to the fist one or to the other one.

self-organization enables adaptation in MAS. Any change in the organization involves a change in the global function. Then, self-organization can be assimilated to a form of learning, as the system learns to interact with its environment. But most of all, self-organization is a key concept to control the emergence of desired properties. In the next section, we propose an approach to design artificial system with emergent functionalities: the Adaptive Multi-Agent System approach.

4.3 Designing the Emergence: the AMAS Approach

This section presents the Adaptive Multi-Agent System (AMAS) approach. It proposes a methodological approach to study and build artificial systems with emergent functionalities. The capacity of adaptation of an AMAS comes from its capacity to self-organize thanks to the cooperative attitude of the agents composing the system [Georgé et al., 2003]. The foundations of the approach are presented on the rest of this chapter.

4.3.1 Interaction and Cooperation

On the previous section, we have discussed of the coupling between a MAS and its environment. The activity of the MAS influences its environment, and the MAS activity is influenced by its environment. [Kalenka and Jennings, 1999] categorise the interaction between a system and its environment in three types:

- ▷ **Cooperative**: the action of an entity promotes the activity of the other providing to both entity individual benefits.
- ▷ **Neutral**: the action of an entity neither hinders or promotes the activity of the other.
- ▷ **Antinomic**: the action of an entity hinders the activity of the other.

A system is in a cooperative state if all its interaction are cooperative. A system is in a non cooperative state if there is at least one interaction that is neutral or antinomic.

4.3.2 Functional Adequacy

A key notion of the AMAS approach is the functional adequacy. An artificial system is designed to perform a function and intuitively, the system is functionally adequate when it executes the function which it has been designed to. Usually, the evaluation of the functional adequacy is determined by an external entity which observes the system activity. However, with a MAS, this evaluation has to be performed by the inner agents which have no clue on the global task. This must rest on self-observation capacities, evaluating only local criterion. The AMAS approach stipulates that a system in which all the agents are in a cooperative state is functionally adequate [Georgé et al., 2003]. The AMAS approach proposes a definition of the functional adequacy based on the categorization of the interaction between a system and its environment.

Definition: A system is functionally adequate if it has no antinomic activity on its environment.

Reciprocally, a cooperative system, which has only beneficial activities with its environment, is functionally adequate.

Given this definition, [Glize, 2001] expresses the theorem of functional adequacy as :

Theorem of functional adequacy : *Given a functionally adequate system, there exists at least one cooperative internal medium system that fulfils an equivalent function in the same environment*.

For more information on the demonstration of the theorem, the reader can refer to [Georgé et al., 2004]. A *cooperative internal medium* is a system in which all the interactions between its constituting parts are *cooperative*.

Thus, for each problem where a solution is effectively calculable, there exists a MAS where all the agents are in a cooperative state that solves this problem. The design of a functionally adequate system can be made with a focus on the design of local cooperative interactions between the constituting parts.

4.3.3 Adapting the System Trough its Parts

As it has been said earlier, the MAS is coupled with its environment. As soon as a change in the environment occurs, the system's functionality may not be in functional adequacy anymore. The more the system is complex, the more difficult it is to reach and maintain a functionally adequate state. As expressed by the AMAS approach, the non adequacy of the system comes from the existence of non cooperative interactions within the system. In order to repair the system functionality and reach a functionally adequate state, agents within the system need to locally detect the failures in cooperation and change their behaviour in result. Self-organization of an AMAS rests on the self-observation capacities of its agents to detect, anticipate and repair *non cooperative situations*.

4.3.3.1 Non Cooperative Situations

An agent is in a Non Cooperative Situation (NCS) when there is a failure in its perception, decision or action process resulting in non cooperative interactions. Seven types of NCS have been identified:

- ▷ **Incomprehension**: the agent cannot extract the semantic contents of a received stimulus.
- ▷ **Ambiguity**: the agent extracts several interpretations from a same stimulus.
- ▷ **Incompetency**: the agent cannot benefits from the current knowledge state during the decision.
- ▷ **Unproductiveness**: the agent cannot propose an action to do during the decision.
- ▷ **Concurrency**: the agent perceives another agent which is acting to reach the same world state.
- ▷ **Conflict**: the agent believes that the transformation it is going to operate on the world is incompatible with the activity of another agent.
- ▷ **Uselessness**: the agent believes that its action cannot change the world state or it believes that the results for its action are not interesting for the other agents.

To solve a NCS, an agent has to locally adjust its behaviour. In order to do so, the agent disposes of three means [Capera, 2005]:

- ▷ **Tuning**: the agent adjusts its internal parameters.
- ▷ **Reorganization**: the agent changes the way it interacts with its neighbourhood,i.e it stops interacting with a given neighbour, or it starts interacting with a new neighbour, or it updates the confidence given to its existing neighbours.
- ▷ **Openness**: the agent creates one or several other agents, or deletes itself.

The behaviour of an agent can be split in two parts:

- ▷ the *nominal* behaviour which ensures the functional adequacy when the agent is in a cooperative state.

- the *cooperative* behaviour, which is a subsumption of the nominal behaviour, enables the agent to reach its nominal behaviour.

The cooperation in an AMAS is assured by mechanisms which either anticipate or resolve NCS. This task is devolved to the system designer who has to identify the NCS and to propose the adequate mechanisms. In the next section, we present a methodology to build Adaptive Multi-Agent Systems.

4.3.4 The ADELFE Methodology

The AMAS approach differs from traditional MAS engineering by its focus on local cooperative behaviours. The designer must describe the system's environment, specify the agents composing the system, characterize their interactions and failures in cooperation and propose mechanisms to restore a cooperative state if needed.

ADELFE [Bernon et al., 2002][Picard and Gleizes, 2004] is the french acronym for "*Atelier de Développement de Logiciel à Fonctionnalité Emergente*" which can be translated by *Toolkit for Designing Software with Emergent Functionalities*. The ADELFE methodology is based on the well-known software development methodology *Rational Unified Process* in which some work products specific to the AMAS approach are added [Bonjean et al., 2014]. ADELFE is composed of 21 work products split in 5 phases:

- **WD1 - Preliminary requirements:** this phase represents a consensus description of specifications between customers, users and designers on what must be and what must give the system, its limitations and constraints.
- **WD2 - Final requirements:** in this work definition, the system achieved with the preliminary requirements is transformed in a use case model, and the requirements (functional or not) and their priorities are organized and managed.
- **WD3 - Analysis:** the analysis begins with a study or analysis of the domain. Then, identification and definition of agents are processed. The analysis phase defines an understanding view of the system, its structure in terms of components and identifies if the AMAS theory is required.
- **WD4 - Design:** this phase details the system architecture in terms of modules, subsystems, objects and agents. These activities are important from a multi-agent point of view as a recursive characterization of multi-agent systems is achieved at this point.
- **WD5 - Implementation:** implementation of the framework and agent behaviours.

The ADELFE process is not a simple waterfall process as some loops and increments are included. A complete description of the ADELFE approach can be found in [Bonjean et al., 2014].

4.3.5 AMAS Applications

Currently, there is no theoretical tool powerful enough to model a dynamic system such as an AMAS in the general case. Hence, application of the theory play an important role in the validation of the approach. Since its conceptualisation, the AMAS approach has been applied to various systems:

- ▷ Real-time simulation for flood forecast [Georgé et al., 2003].
- ▷ Manufacturing control [Kaddoum, 2011].
- ▷ Continuous optimisation [Jorquera, 2013].
- ▷ Abnormal behaviour detection and alert triggering in maritime surveillance [Brax et al., 2013].
- ▷ Self-tuning of Game Scenarios [Pons, 2014].

This non exhaustive list illustrates the variety of domains in which the AMAS approach has been, and is still, validated.

4.4 Control and Learning with an AMAS

The AMAS approach focuses on the design of autonomous agents that have to collectively solve a common task or reach a common objective. Thus, the AMAS approach cannot fulfil with all the adaptive complex systems and all kinds of simulations of those systems. [Georgé et al., 2011] identify some characteristics of the applications in which the AMAS approach is useful:

- ▷ The application is complex in the sense of complex systems.
- ▷ The control and knowledge can (and often has to) be distributed.
- ▷ There is a problem to solve. The problem can be expressed as a task or a function to realise, a structure to observe, ...
- ▷ The application objective can be very precise such as the optimisation of a function or more diffuse such as the satisfaction of the system end-user.
- ▷ The system has to adapt to an endogenous dynamic (with the appearance or disappearance of parts of the system) or an exogenous (with the interaction with its environment).
- ▷ The system is underspecified. In this case, the adaptation is a mean to design it.

In many aspects, our desired application in Ambient Systems matches with these criteria:

- ▷ The control is a task to realise.
- ▷ The law of requisite variety ([Ashby et al., 1956]) involves that the controller complexity must be at least equal to the system's complexity.
- ▷ The different functionalities (as expressed by the Extreme Sensitive Robotics in section 1.6) are distributed.

- The task to perform is *a priori* unknown.
- The system has to adapt itself to different kind of user with different kind of needs.
- The system has to deal with the appearance and disappearance of other devices.

Among the different AMAS applications, the results of two recent contributions tend to strengthen the AMAS adequacy to our problem. [Guivarch, 2014] has proposed and tested in simulation an AMAS to learn user preferences in the context of ambient systems. [Boes, 2014] designed an AMAS to dynamically learn to control heat engines. These two approaches have validated the pertinence of the AMAS for the design of control systems in complex environments. Thus, they have highly inspired the system that we are presenting in the next chapter.

5 ALEX, Show Me And I Learn

*This chapter describes our main contribution: **ALEX** (Adaptive Learner by EXperiments), an Adaptive Multi-Agent System designed to learn to control a robotic device from demonstrations. Its design and the behaviours of its agents are explained and the chapter concludes with a positioning in regards with other approaches.*

FROM the analysis made in the previous chapters, we have identified the need to design a software component that is able to dynamically build a mapping function from demonstrations. This chapter describes the design and behaviour of **ALEX**, an acronym for **A**daptive **L**earner by **EX**periments, which is a multi-agent system designed to learn to control a system from demonstrations performed by external (human or virtual) entities. Its design has been made in accordance with the ADELFE methodology.

5.1 Objectives

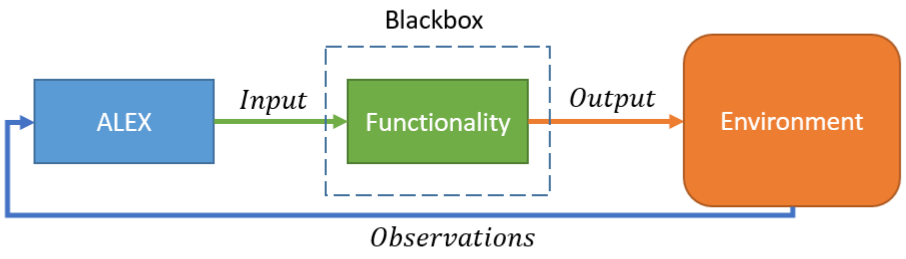

Figure 5.1 — The objective of ALEX is to control the input of a functionality in accordance with user needs.

In accordance with the Extreme Sensitive Paradigm (see section 1.6), the objective of an ALEX instance is to control a functionality, seen as a slaved Single-Input Single-Output blackbox (figure 1.2). ALEX must control the input of the functionality in accordance with what has been demonstrated in order to produce the desired effects on the environment. In order to do so, ALEX receives from the environment updates on a set of observations. This set of observations describes a vector of integer values, each value corresponding to a particular observation. On the previous chapters, we have established that the control has to be made through the building of a mapping function associating to each state of the set of observations the adequate control action to perform.

One first requirement for ALEX is to be easily usable with any kind of functionality. This involves that the usage of this software component must require as few settings as possible and no expertise on how ALEX behaves. Another implication is that ALEX must not require any *a priori* information on the controlled functionality, which means that it must be able to learn all the different inputs required to perform its control.

A second requirement for ALEX is the possibility for users (known as *tutors*) to demonstrate a behaviour at any moment by tele-operating the functionality, which means by providing to ALEX the desired control action to perform over. This involves that ALEX must continuously observe the tutoring input to evaluate and adapt its own policy.

The function of ALEX is to learn and exploit a mapping policy which associates the current state of ALEX environment to the adequate control action on the functionality. In order to do so, ALEX must observe in real-time its environment, including users activity.

5.2 Environment

Once the objectives of ALEX have been highlighted, we can describe the environment in which ALEX evolves by describing the entities composing it. A differentiation is made between *active* entities, which have their own dynamics and can initiate the activity even in the absence of external stimuli, and *passive* entities, which have no dynamics of their own and can only be perceived and potentially altered by active entities and the system itself. We identify two types of active entities and one passive entity in ALEX's environment.

Active entities

- *Observation*: ALEX must observe in real time the environment in which the functionality evolves. This environment is described by a set of observations (basically signals coming from sensors and other software). Each observation is an active entity with its own dynamics.
- *Tutors activity*: ALEX must interact with tutors which can act at any time to demonstrate a behaviour. Those tutors, which can be either human or virtual, have their own dynamics.

Passive entities

- *Functionality input*: The objective of ALEX is to control the input of a functionality. The value of this input is only controlled by ALEX and has no dynamics of its own.

From this description of the environment we can now propose a component description of ALEX (figure 5.2). ALEX requires a set of observations and the tutors activity and provides the functionality input.

The description of ALEX's environment enables its characterisation using the characteristics defined by [Russell and Norvig, 1995]:

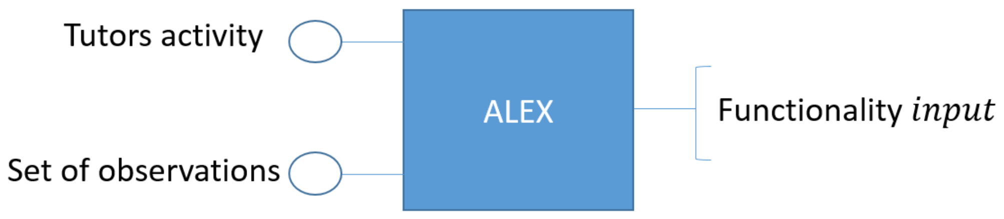

Figure 5.2 — The component view of ALEX. ALEX requires to perceive tutors activity and a set of observations to provide the adequate input corresponding to the current situation.

- ▹ The environment is **dynamic**: the active entities present in ALEX's environment are not only modified by ALEX but have their own dynamics. Those modifications can occur even without any activity from ALEX. For example, a signal describing the luminosity of a room can have its own dynamics, due to variation of luminosity during a day, even if no effector is controlling a light.
- ▹ The environment is **continuous**: regarding applications in real world, the number of observations and actions is not discrete.
- ▹ The environment appears to be **non-deterministic**: The environment is partially observable, the consequences of performed actions in the real world could not be determined in advance with certainty.
- ▹ The environment is **non-accessible**: not all information that could be used is available to the system. For example, user satisfaction is not directly accessible, only its activity is observable.

This analysis reinforces the idea that the problem is difficult and that the Adaptive Multi-Agent System approach is adequate. On the rest of this chapter, we describe how ALEX operates.

5.3 Nominal Behaviours

In the AMAS approach, the *nominal behaviour* is the behaviour enabling an entity to perform its tasks in the absence of non-cooperative situations [Bonjean et al., 2014]. ALEX is in its nominal behaviour when it has acquired enough knowledge to control the functionality in accordance with what has been demonstrated. ALEX nominal behaviour is then to exploit what it has learned in order to control the functionality. To introduce the different agents composing ALEX, we first describe the nominal behaviour of ALEX. Then, we provide for each agent a complete description of their nominal behaviours.

5.3.1 ALEX Nominal Behaviour

The nominal behaviour of ALEX is the behaviour that the system follows when it is not in a non cooperative situation, which means when the system successfully learned to perform the demonstrated task. The behaviour can be split in three activities, each activity carried out by a specific type of agent.

5.3.1.1 Observing the Environment

A first activity is to observe the environment and its evolutions. In order to do so, ALEX has to build a representation of each observation. This task is devolved to **Percept Agents**. The term *percept* refers to its usage in philosophy where the term describes an object of perception; something that is perceived. In our context, the word *percept* is an echo to the functionality of the *Percept Agent*, which is to build and share an internal representation of an external stimuli. Each incoming observation is associated with a unique *Percept Agent*. During its life cycle, a *Percept Agent* perceives the observation, updates its knowledge on this observation, and shares this knowledge to those who need it.

5.3.1.2 Analysing the Environment State

Thanks to *Percept Agents*, ALEX has an internal and distributed representation of its environment. This environment needs to be analysed in order to be able to decide if an action has to be performed, based on the experience acquired by the system. This task is devolved to **Context Agents**. The term *context* may appear ambiguous as it is used in many domains [Bazire and Brézillon, 2005]. Here, the term context refers to all information external to the activity of an entity that affects its activity. This set of information describes the environment as the entity sees it [Guivarch et al., 2012].

The function of a *Context Agent* is to propose to change the current functionality input with a particular *action* when it is appropriate. In order to do so, the *Context Agent* disposes of a description of the environment states in which its *action* is appropriate. This description is named *context* as it describes the context in which the action is applicable. Thanks to the information provided by *Percept Agents*, the *Context Agent* can compare its own *context* description to the current state of the environment and decide or not to propose its *action*.

5.3.1.3 Applying the Adequate Control Action

The objective of ALEX is to control the input of a functionality in accordance with the demonstrations made by tutors. In order to do so, ALEX needs to build an internal representation of this input and to own the skills enabling it to change the current input. This task is devolved to the **Input Agent**. *Context Agents*, during their life-cycles, can ask the *Input Agent* to apply their actions. The function of the **Input Agent** is to apply, when it is necessary, the action proposed by a *Context Agent*.

5.3.1.4 Synthesis

The behaviour of ALEX is the emerging result of the interactions between three kinds of agents:

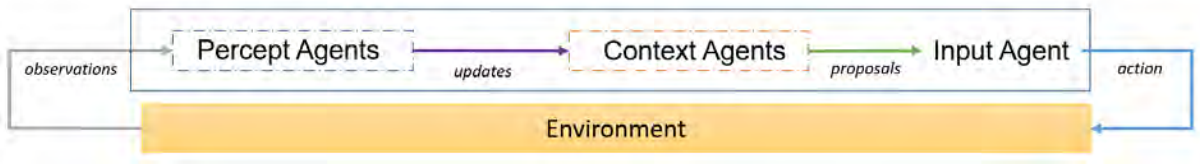

Figure 5.3 — A schematic view of interactions within ALEX during the nominal behaviour. *Percept Agents* receive observations from the environment and send updates to *Context Agents*. A Context Agent makes a proposal to the *Input Agents* and the *Input Agents* applies the *action*.

- ▷ **Percept Agents** are associated to each observation. They build and share knowledge about this observation to agents requiring it.
- ▷ **Context Agents** are associated with a unique *action* and a *context* description. They propose their *actions* when the current environment matches with their *context* descriptions.
- ▷ A unique **Input Agent** applies *actions* coming from a Context Agent.

Those interactions are illustrated in figure 5.3.

This system decomposition is quite similar to the traditional Perception-Decision-Action decomposition, where *Percept Agents* can be seen as the perceptive part of the system, *Context Agent* as the decisional part, and the *Input Agent* as its active part. However, each agent has its own autonomy and its own Perception-Decision-Action life-cycle. On the rest of this section, we provide more details on the behaviour of each agent.

5.3.2 Agents Nominal Behaviours

On the previous section, we have highlighted the function of each type of agent composing ALEX. On this section, we describe the structure of each agent and their nominal behaviours through a Perception-Decision-Action process.

5.3.2.1 Percept Agent Nominal Behaviour

Function The function of a *Percept Agent* is to build and share knowledge about an observation to agents that require this knowledge.

Each observation from the environment is associated with a unique *Percept Agent*. This *Percept Agent* is composed of an internal representation of this observation and a neighbourhood of agents interested by updates on this observation.

The internal representation of an observation is composed of:

- ▷ The current value of the observation.
- ▷ The previous value of the observation.
- ▷ The minimal and maximal values observed.

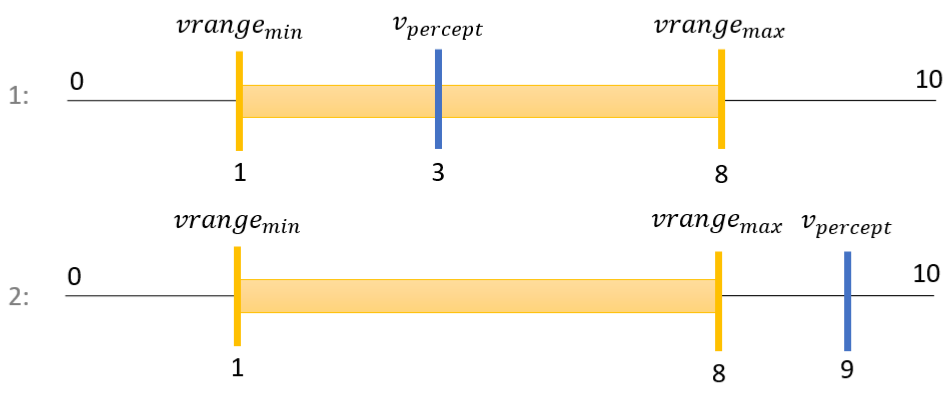

Figure 5.4 — An illustration of a validity range with two different $v_{percept}$ values. In (1), $v_{percept} \in [vrange_{min}, vrange_{max}]$. In (2), $v_{percept} \notin [vrange_{min}, vrange_{max}]$.

The *Percept Agent* receives updates about its observation from the environment and can send messages to any member of its neighbourhood to share its knowledge.

The nominal behaviour of a *Percept Agent* then consists in:

- **1 - Perception:** The *Percept Agent* receives updates on an observation from its environment.
- **2 - Decision:** The *Percept Agent* updates its internal representation.
- **3 - Action:** The *Percept Agent* sends updates to the agents in its neighbourhood.

5.3.2.2 Context Agent Nominal Behaviour

Function The function of a *Context Agent* is to propose its *action* when this *action* is appropriate.

A *Context Agent* is composed of a *context* description and a unique *action*.

The *context* description is composed of a set of *validity ranges*, one associated with each *Percept Agent*. A validity range is the description of a validity interval $[vrange_{min}, vrange_{max}]$ and an internal representation of the current value $v_{percept}$ of the associated *Percept Agent* (see figure 5.4). Then, the *Context Agent* disposes for each *Percept Agent* of an interval describing the values in which the *action* is appropriate and can compare the current value of the *Percept Agent* to this interval.

The *action* is a textual description of the action to apply. It can be the description of a high-level command (such as "go Forward", "go Left") or a low-level command ("speed at 42%", "x=2.1",..) depending on the functionality to control. It is provided to the agent at birth and never changes.

By analysing its context description, the *Context Agent* can determine if its action is appropriate and send an *action* proposal to the *Input Agent*.

Then, the nominal behaviour of a *Context Agent* consists in:

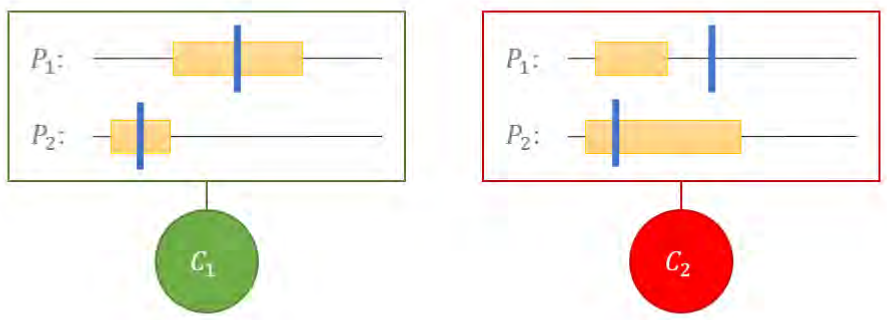

Figure 5.5 — An illustration of a *valid Context Agent* C_1 and an *invalid Context Agent* C_2. Each *Context Agent* has a validity range associated with the *Percept Agents* P_1 and P_2. The current value of P_1 (the blue line) is out of the validity range of C_2.

- ▷ **1 - Perception:** The *Context Agent* receives updates from *Percept Agents* and updates the internal representation of each validity range.
- ▷ **2 - Decision:** The *Context Agent* determines its internal state:
  - If the current value of all *Percept Agents* is included in the interval described by the validity ranges of the *Context Agent*, the *Context Agent* is said *valid* (see figure 5.5).
  - If at least one value of a *Percept Agent* is not included in the interval described by the validity ranges of the *Context Agent*, the *Context Agent* is said *invalid*.
- ▷ **3 - Action:**
  - If the *Context Agent* is *valid*, the *Context Agent* proposes its *action* to the *Input Agent*.
  - If the *Context Agent* is *invalid*, the *Context Agent* does not propose its *action*.

5.3.2.3 Input Agent Nominal Behaviour

Function The function of the *Input Agent* is to apply, when it is necessary, the action proposed by a *Context Agent*.

The *Input Agent* is composed of an internal representation of the functionality input. The *Input Agent* can perceive and change the current state of the functionality input.

The nominal behaviour of the *Input Agent* is:

- ▷ **1 - Perception:** The *Input Agent* receives an *action* proposal from a *Context Agent*.
- ▷ **2 - Decision:** The *Input Agent* compares the *action* proposed by the *Context Agent* to the current functionality input.
- ▷ **3 - Action:** If the action proposed by the *Context Agent* is different from the current functionality input, the *Input Agent* applies the *Context Agent action* and changes the current functionality input state.

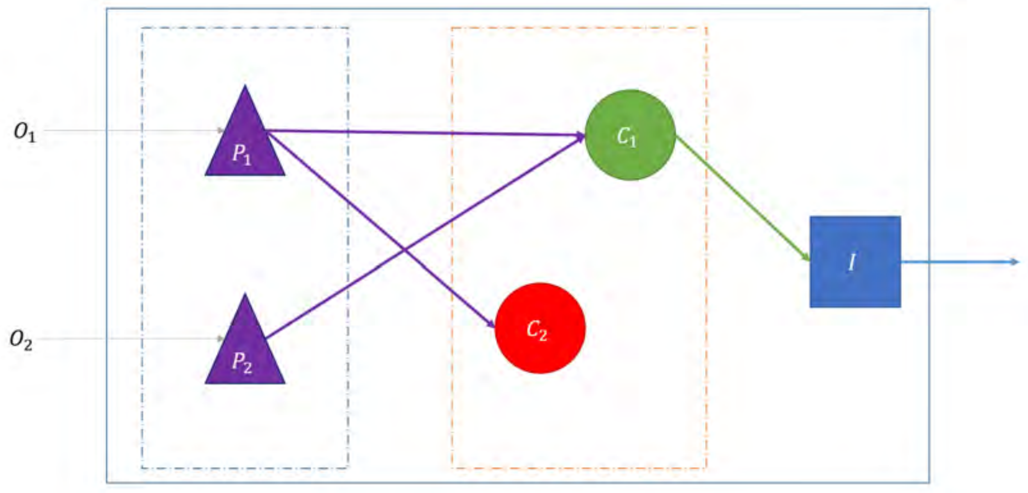

Figure 5.6 — An example of ALEX architecture with two *Percept Agents* (purple triangles), a *valid Context Agent* (green circle), an *invalid Context Agent* (red circle) and the *Input Agent* (blue square). Each *Percept Agent* receives updates on an observation from the environment and sends updates to *Context Agents*. *valid Context Agents* make proposals to the *Input Agent* which applies the *action*.

5.3.2.4 Synthesis

At this point, we have outlined the general behaviour of ALEX and described the different agents composing it. ALEX is composed of a set of *Percept Agents*, a set of *Context Agents* and an *Input Agent*. Each agent has its own objectives and interactions among the system. *Percept Agents* build an internal representation of an observation and interact with the environment and *Context Agents*. *Context Agents* act as an analysis unit by comparing the current state of the environment to their context description. They interact with *Percept Agents* and the *Input Agent*. At last, the *Input Agent* is responsible of applying the different *actions*. The *Input Agent* interacts with *Context Agents* and the functionality. The nominal behaviour is illustrated in the figure 5.3 and an example of ALEX architecture is proposed in figure 5.6.

In this section, we have described the nominal behaviour of each agent. This behaviour is the one that occurs when a system managed to find an organization that enables it to perform its task. However, there are many situations in which the nominal behaviour may fail. In the Adaptive Multi-Agent Systems approach, failures of the nominal behaviour occur when there are Non Cooperative Situations. Agents among the system must reorganize to reach back the nominal behaviour. The questions are now how to enable such re-organization, how to populate ALEX with enough agents and how those different agents can self-organize to find an organization that enables the system to perform its task? This can only be enabled by the design of local mechanisms allowing agents to anticipate, detect and repair Non Cooperative Situations. The next section lists all the Non Cooperative Situations and the mechanisms implemented to repair them.

5.4 Non Cooperative Situations

In this section, we list the different Non Cooperative Situations that could occur within the system. Those situations are numbered. However, there is no hierarchy among them, the numbering is just used in a didactic way so that every situations are easily searchable. Non Cooperative Situations are solved by applying local rules enabling each agent to adjust its behaviour and regain its nominal behaviour.

5.4.1 Introducing the Tutor

The reader could have noticed from the expression of the nominal behaviours that there are no mention to the Tutor. While being absent of the nominal behaviour, the Tutor is a key component of ALEX ability to self-organize. Before describing all the Non Cooperative Situations and the mechanisms that we have implemented to repair or anticipate them, we first propose to put a focus on the tutor and to introduce the interactions between ALEX and its Tutor.

ALEX is built on the hypothesis that there exists at least one entity (there may be several tutors) among the environment that is able to demonstrate an adequate behaviour. This entity (or those entities) is referred in this thesis as the *Tutor*. To demonstrate a behaviour, the Tutor can act on ALEX by providing the adequate *action* to perform. At each time step, ALEX only perceives the Tutor activity, which means that ALEX perceives the action performed by the Tutor. There is no assumption on the number of actual entities doing the demonstration, ALEX only perceives which is the adequate *action* to perform on the current situation, no matter who is at the origin of this *action*.

The Tutor can act at any time on ALEX, which means that the *action* could be empty if no one is performing a demonstration on the current situation. When the *action* is empty, ALEX uses the knowledge it has previously acquired to control autonomously its functionality. However, when the *action* is not empty, ALEX applies the *action* proposed by the Tutor.

The observation of the Tutor activity is a rich feedback to enable *Context Agents* to evaluate the adequacy of their own activity. Indeed, the simple comparison of the Tutor *action* to the one proposed by *Context Agents* enables not only to evaluate the adequacy of *Context Agents* behaviour, but it can also be the source to build new *Context Agents*. Thus, the Tutor activity can be seen as a perturbation implying self-organization.

On the rest of this section, we list the different Non Cooperative Situations. For each Non Cooperative Situation, we first describe the problem that leads to a failure in the cooperation. Some NCS are split in two different situations in which the Tutor *action* is empty (labelled as "No action available") and situations in which an *action* is present (labelled as "Action available"). Then we propose mechanisms to solve or anticipate these situations. The implementation of those mechanisms are detailed in the next section.

5.4.2 NCS 1: Incompetence of the Input Agent

Problem description: during its life-cycle, the *Input Agent* decides or not to change the functionality input. Its decision is based on the proposition made by a *Context Agent*. If no *Context Agent* has made a proposal, the *Input Agent* is then unable to decide which *action* to apply and is in a situation of incompetence.

5.4.2.1 NCS 1.a: Action Available

Detection: this NCS is detected by the *Input Agent* during its perception phase when no *action* proposal have been received from *Context Agents* and a Tutor's *action* is available.

Resolution: to resolve this situation, the *Input Agent* perceives the Tutor's *action* and when this *action* is available, applies it. Using the Tutor's *action* enables the *Input Agent* to resolve a situation of incompetence.

However, the next time a similar situation occurs, the *Input Agent* will be once again in a situation of incompetence. Indeed, the absence of *action* proposal from a *Context Agent* involves that no *Context Agent* is *valid* in the current situation. In order to anticipate such situations, the *Input Agent* creates a new *Context Agent*. The *action* of this new *Context Agent* is the one of the Tutor and its *context* is initialised to describe the current situation. Then, the next time a similar situation will occur, the newly created *Context Agent* would be *valid* and propose the action. The creation of *Context Agents* is discussed in section 5.5.5.

5.4.2.2 NCS 1.b: No Action Available

Detection: this NCS is detected by the *Input Agent* during its perception phase when no *action* proposal has been received from *Context Agents* and no *action* from the Tutor is available.

Resolution: in the resolution of the previous NCS (NCS 1.a), the *Input Agent* used the Tutor *action* to resolve its incompetence. However, in some situations this action is not available as no Tutor is performing a demonstration. To resolve those situations, the *Input Agent* maintains the last known *action*. If this *action* is not the one that satisfies the Tutor, the Tutor will, on the next cycle, act on the system to correct it.

5.4.3 NCS 2: Conflict of the Input Agent

Problem description: the *Input Agent* must apply the action proposed by a *Context Agent* if this action is in adequacy with the Tutor. If the *Input Agent* applies an inadequate action, the *Input Agent* will be in conflict with its environment.

Detection: this situation can occur if the *Input Agent* applies an action that is not in adequacy with the Tutor. It is only detectable by comparing the *action* proposed by a *Context*

Agent and the Tutor *action*. If the action of the *Context Agent* is different from the action proposed by the Tutor *action*, applying the *action* of the *Context Agent* could lead to a conflict between the *Input Agent* and its environment.

Anticipation: this situation is solved by anticipation. Whenever an *action* from the Tutor is available, the *Input Agent*applies this *action*, no matter what have been proposed by the *Context Agents*.

5.4.4 NCS 3: Conflict Between Context Agents and the Input Agent

Problem description: if two (or more) *Context Agents* propose at the same time different *actions*, the agents are in conflict. Indeed, only one *action* is applicable at a time, so they try to operate *actions* that are incompatible.

Detection: during its perception phase, the *Input Agent* receives action proposals from *Context Agents* which enable it to detect situations of conflict.

5.4.4.1 NCS 3.a: Action Available

Resolution: if a Tutor's *action* is available, the *Input Agent* is not altered by the conflict between *Context Agents* as the mechanism presented in the NCS 2 (section 5.4.3) enables it to decide which is the action to apply. However, *Context Agents* needs to adapt their behaviours. Two situations are differentiable: if the *action* proposed by a *Context Agent* is identical to the one of the Tutor or if the *action* proposed by a *Context Agent* is different. In the first case, *Context Agents* will be in concurrence and this situation is treated with the NCS 4 (section 5.4.5). In the second case, *Context Agents* that propose an *action* that is different to the one of the Tutor fail to propose their *action* when it is adequate. They need to modify their *context* representation to exclude the current situation. Then, the next time a similar situation occurs, the *Context Agent* would not be *valid*.

5.4.4.2 NCS 3.b: No Action Available

Resolution: if no Tutor's *action* is available, the *Input Agent* is unable to decide which is the *action* to perform. *Context Agents* proposing different actions are not only in conflict, but this conflict leads to a situation of incompetence for the *Input Agent*. In order to solve those situations, *Context Agents* must help the *Input Agent* to choose which is the best *action*. In order to do so, we add to the *action* proposal of *Context Agents* a *confidence* value. This *confidence* value is self-managed by each *Context Agent* by comparing their *action* proposals to the Tutor's action. The more their proposals are in adequacy with the Tutor action, the more their *confidence* value is high. Reciprocally, the more their proposals are not in adequacy with the Tutor's *action*, the more their *confidence* is low. The *Input Agent* which now receives *actions* proposals associated with a *confidence* value can select the *action* with the higher *confidence*. The implementation of this *confidence* value and its management are presented in the section 5.5.4.

Anticipation: to anticipate situations of conflicts, *Context Agents* self-manage their *confidence* value. Each time the *Input Agent* applies an *action*, a message is sent to each *valid Context Agents* to inform which is the *action* being applied and what is the *confidence* value associated with this action. Using this information, each Context Agent can compare its proposal to the *action* performed by the *Input Agents* and adapt its *confidence* value consequently.

5.4.5 NCS 4: Concurrence of a Context Agent

Problem description: if two (or more) *Context Agents* propose at the same time identical *actions*, *Context Agents* would be in a concurrency situation in which they act to reach the same world state.

Detection: during its perception phase, the *Input Agent* receives action proposals from *Context Agents* which enable it to detect situations of concurrence.

5.4.5.1 NCS 4.a: Action Available

Resolution: if two or more *Context Agents* propose the same *action* and this *action* is not in adequacy with the Tutor's *action*, those *Context Agents* must exclude the current situation from their *context* as they are in conflict with the *Input Agent*. The resolution of this situation will activate the anticipation of the NCS 3 (see section 5.4.4) and the *Context Agent* will decrease its *confidence* value.

If two or more *Context Agents* propose the same *action* and this *action* is in adequacy with the Tutor's *action*, those *Context Agents* are in concurrency. This concurrency does not affect the *Input Agent* as the good *action* is proposed. Using the mechanism of *confidence* introduced in the NCS 3 (see section 5.4.4), the *Input Agent* is able to select the *action* with the higher *confidence* value. Those concurrency situations do not affect ALEX capacity to control its functionality but could have an impact on its computational resources, as there is a redundancy in the information carried by *Context Agents*. *Context Agents* with the lower *confidence* values will try to exclude the current situation from their current *context* description.

5.4.5.2 NCS 4.b: No Action Available

This NCS is detected and solved using the mechanism introduced in the NCS 3 (see section 5.4.4). The *Input Agent* selects the *action* proposal with the higher *confidence* value.

5.4.6 NCS 5: Incompetence of a Context Agent

Problem description: if a previously *valid Context Agent* becomes *invalid* but its action is still applied by the *Input Agent*, the *Context Agent* is in a situation of incompetence where it fails to propose its *action* whereas the *action* is the one to be applied.

Detection: this situation is detected by a *Context Agent* which becomes *invalid* while its *action* is still applied by the *Input Agent*.

Resolution: if a *Context Agent* was previously *valid* and its *action* is still applied by the *Input Agent*, this Context Agent will adapt its *context* description to include the current situation. If the agent cannot adapt its *context* description (see section 5.5.3), the situation could lead to an incompetence of the *Input Agent* which is solved by the mechanism of the NCS 1 (see section 5.4.2).

Anticipation: to anticipate situations of incompetence, *Context Agents* propose their actions in situations that are closed to their *context* description but are not included in it. *Context Agents* will then said to be *validable*. A *Context Agent* that is *validable* will propose its *action* to the *Input Agent*. The *Input Agent*, during its selection process will only consider *validable* proposals if no other *Context Agent* is making a *valid* proposal. The *validable* mechanism is explained in details in section 5.5.3. If the action of a *validable Context Agent* is selected, this *Context Agent* will adapt its *context* description to include the current situation.

5.4.7 NCS 6: Uselessness of a Context Agent

Problem description: it happens that a *Context Agent* is led to a progressive reduction of one or more of its *validity range* so that the amplitude of the validity range is smaller than the smallest amplitude observable. The *Context Agent* would be permanently *invalid* and then be in a situation of uselessness. While this NCS is not crucial to ALEX good behaviour, maintaining useless agents among the system is a waste of computational resource.

Detection: this NCS is detected by *Context Agents* after the update their *validity ranges*.

Resolution: if the *Context Agent* is useless, the only solution to solve this NCS is for the *Context Agent* to self-suppress. By this operation, the *Context Agent* prevents the usage of computational resources that could be useful to other agents.

Anticipation: the NCS 3 (see section 5.4.4) introduced a mechanism of *confidence*. This *confidence* value enable to detect *Context Agents* that have a tendency to make bad proposal. A threshold parameter enables to filter those agents and agents with a confidence value that is lower than this parameter will self-suppress.

5.4.8 NCS 7: Uselessness of a Percept Agent

Problem description: from the description of its nominal behaviour, a *Context Agent* is *valid if and only if* the current value of all *Percept Agents* are included in the interval described by its validity ranges. If there exists at least one value from a *Percept Agent* that is out of bound, the *Context Agent* is *invalid*. If a *Percept Agent A* sends updates to a *Context Agent* that is already invalidated by another *Percept Agent B*, the *Percept Agent A* is in a uselessness situation. The information of the *Percept Agent A* does not help the *Context Agent* to decide.

Detection: this situation is detected by a *Context Agent* after the reception of an update message from a Percept Agent. If after the reception of an update from a *Percept Agent*, the current value of the validity range associated with this *Percept Agent* is out of bounds, the *Context Agent* detects that the value from the other *Percept Agents* is not useful any-more to determine its current state.

Resolution: the *Context Agent* which detects this situation informs all the other *Percept Agents* that their updates are not required and those *Percept Agents* remove the *Context Agent* from their neighbourhood. The *Context Agent* will only receives updates from the last *Percept Agent* that has made it *invalid*.

5.4.9 NCS 8: Incompetence of a Percept Agent

Problem description: from the previous NCS resolution, if a *Context Agent* is *invalid*, the reason is that the value of a particular *Percept Agent C* is not included by the *validity range* of the *Context Agent*. If the *Percept Agent C* changes of value and is now included by the *validity range* of the *Context Agent*, the other *Percept Agents*, which has stopped to send their updates to the *Context Agent*, would be in a situation of incompetence, as the *Context Agent* now requires those updates to determine its state.

Detection: this situation is detected by a *Context Agent* when an update from a *Percept Agent* makes this *Context Agent valid*.

Resolution: the *Context Agent* which detects this situation informs all the other *Percept Agents* that their updates are required and those *Percept Agents* add the *Context Agent* to their neighbourhood. The *Context Agent* will receive updates from the all the *Percept Agents*.

5.4.10 Synthesis

A synthesis of the different NCS is presented in figure 5.7.

NCS	**Detection**	**Resolution / Anticipation**
NCS1.a	The *Input Agent* receives no action proposal. A Tutor's action is available.	Use the Tutor's action to create a new *Context Agent* and apply Tutor's action.
NCS1.b	The *Input Agent* receives no action proposal. A Tutor's action is not available.	Apply the last known *action*.
NCS2	The *Input Agent* receives an action proposal that is not in adequacy with the Tutor's action.	Whenever a Tutor's action is available, applies the Tutor's action.
NCS3.a	The *Input Agent* receives more than one different action proposal and a Tutor's action is available.	Each *Context Agent* which proposes an action that is not in adequacy with the Tutor's action excludes the current situation from its *context* description.
NCS3.b	The *Input Agent* receives more than one different action proposal but no Tutor's action is available.	Every *Context Agent* builds a *confidence* value which is associated with the *action* proposal. The *Input Agent* selects the *action* with the higher *confidence* value.
NCS4.a	The *Input Agent* receives more than one identical *action* proposal and a Tutor's action is available.	The *Input Agent* selects the action with the higher *confidence* value. *Context Agents* with the lowers *confidence* values try to exclude the current situation from their *context* description.
NCS4.b	The *Input Agent* receives more than one identical *action* proposal and no Tutor's action is available.	Every *Context Agent* builds a *confidence* value which is associated with the *action* proposal. The *Input Agent* selects the *action* with the higher *confidence* value.
NCS5	A *Context Agent* becomes invalid but its action is still applied by the *Input Agent*.	The *Context Agent* tries to include the current situation. If it fails, the *Input Agent* uses the Tutor's action to create a new *Context Agent*. To anticipate those situations, *Context Agents* propose their *action* in situation that are close to their *context* description.
NCS6	A validity range amplitude or the *confidence* value is lower than a threshold.	The *Context Agent* self-suppresses.
NCS7	A *Percept Agent* receives a removal message from a *Context Agent*.	The *Percept Agent* removes the *Context Agent* from its neighborhood.
NCS8	A *Percept Agent* receives an addition message from a *Context Agent*.	The *Percept Agent* adds the *Context Agent* from its neighborhood.

Figure 5.7 — Synthesis of NCS

5.5 Implementation

The previous section highlighted the different Non Cooperative Situations that can occur and proposed mechanisms to solve them. Synthetically, four mechanisms enable self-organization in ALEX:

- ▷ The *Input Agent* ability to create *Context Agents*.
- ▷ *Context Agents* ability to manage their Validity Ranges.
- ▷ *Context Agents* self-management of a *confidence* value.
- ▷ *Percept Agents* neighbourhood management.

In this section, we provide details on how those mechanisms are effectively implemented.

Before describing the mechanisms, we need to introduce some tools that will be used by these mechanisms.

5.5.1 Adaptive Value Trackers

The *Adaptive Value Tracker* (*AVT*) is a tool introduced by [Lemouzy, 2011]. An *Adaptive Value Tracker* enables the discovery of a dynamic real value through successive feedbacks of three kinds: higher (↑), lower (↓) or equal (∼). An *Adaptive Value Tracker* is described by five components:

- ▷ $v_t \in [v_{min}, v_{max}]$ the current value of the *AVT* at step t.
- ▷ $\Delta_t \in [\Delta_{min}, \Delta_{max}]$ the current variation value of v_t.
- ▷ $Fb_t \in \{\uparrow, \downarrow, \sim\}$ the value of the feedback at step t.
- ▷ λ_a the coefficient of acceleration ($\lambda_a > 1$).
- ▷ λ_d the coefficient of deceleration ($0 < \lambda_a < 1$).

Each time an *Adaptive Value Tracker* receives a feedback, it adjusts its value v_t and its variation Δ_t following the behaviour described in figure 5.9. Basically, two successive feedbacks in the same direction increase the value of Δ to accelerate the evolution of v. Two successive feedback of different directions decrease the value of Δ to decelerate the evolution of v. At last, a $\sim$ feedback decreases Δ to decelerate the evolution of v. The coefficient of acceleration λ_a and the coefficient of λ_b can differ depending on the dynamic of the value to track. A coefficient $\lambda_a = 2$ and $\lambda_b = 1/3$ offers good results when no hypothesis on this dynamic can be made [Lemouzy, 2011]. An illustration of the behaviour of the AVT is shown in figure 5.8.

Adaptive Value Trackers have the ability to converge quickly to a value and to maintain in it, while being still able to converge as quickly to a new value. This ability to quickly and dynamically track values makes them ideal in cases where the parameters of an agent can change frequently.

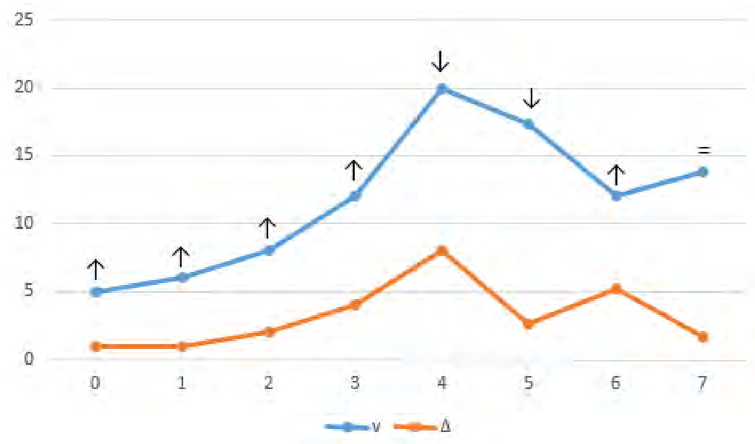

Figure 5.8 — An illustration of the behaviour of an AVT starting at $v_0 = 5$ and $\Delta_0 = 1$ seeking to reach the value 14 with a precision of $+/-0.2$.

		Fb_t		
		$\uparrow$	$\downarrow$	$\sim$
Fb_{t-1}	$\uparrow$	$\Delta_t = \Delta_{t-1}.\lambda_a$ $v_{t+1} = v_t + \Delta_t$	$\Delta_t = \Delta_{t-1}.\lambda_d$ $v_{t+1} = v_t - \Delta_t$	$\Delta_t = \Delta_{t-1}.\lambda_d$ $v_{t+1} = v_t$
	$\downarrow$	$\Delta_t = \Delta_{t-1}.\lambda_d$ $v_{t+1} = v_t + \Delta_t$	$\Delta_t = \Delta_{t-1}.\lambda_a$ $v_{t+1} = v_t - \Delta_t$	$\Delta_t = \Delta_{t-1}.\lambda_d$ $v_{t+1} = v_t$
	$\sim$	$\Delta_t = \Delta_{t-1}$ $v_{t+1} = v_t + \Delta_t$	$\Delta_t = \Delta_{t-1}$ $v_{t+1} = v_t - \Delta_t$	$\Delta_t = \Delta_{t-1}.\lambda_d$ $v_{t+1} = v_t$

Figure 5.9 — Tuning behaviour of v_t and Δ_t of the AVT according to Fb_t and Fb_{t-1}.

5.5.2 Adaptive Value Range Trackers

An *Adaptive Value Range Tracker* (AVRT) is an extension of the *Adaptive Value Tracker* [Guivarch, 2014]. While the objective of an *Adaptive Value Trackers* is to seek for a value, an *Adaptive Value Range Tracker* models a value range composed of two bounds, each bound being an *Adaptive Value Tracker*. AVT_{lower} describes the *Adaptive Value Tracker* that models the lower bound of the range and AVT_{upper} describes the *Adaptive Value Tracker* that models the upper bound. Thus, an *Adaptive Value Range Trackers* is made of two values v_{lower}, which is the current value of AVT_{lower}, and v_{upper}, which is the current value of AVT_{upper}.

An *Adaptive Value Range Tracker* receives a set of examples $E = E_{in} \cup E_{out}$ where E_{in} describes values that must be included by the sought range and E_{out} describes values that must be excluded. Each $e \in E$ is then composed of a value $v_e \in R$ and an information $IN|OUT$ describing if the value must be included by the sought range or not. The role of an *Adaptive Value Range Tracker* is to dynamically adapt its bounds v_{lower} and v_{upper} to each example. Three situations occurs:

▷ The example (v_e, IN) is not included in $[v_{lower}, v_{upper}]$. The *Adaptive Value Range Tracker* has to adapt its bounds to include the example. If $v_e > AVT_{upper}$, a feedback $\uparrow$ is sent to AVT_{upper}. If $v_e < AVT_{lower}$, a feedback $\downarrow$ is sent to AVT_{lower}.

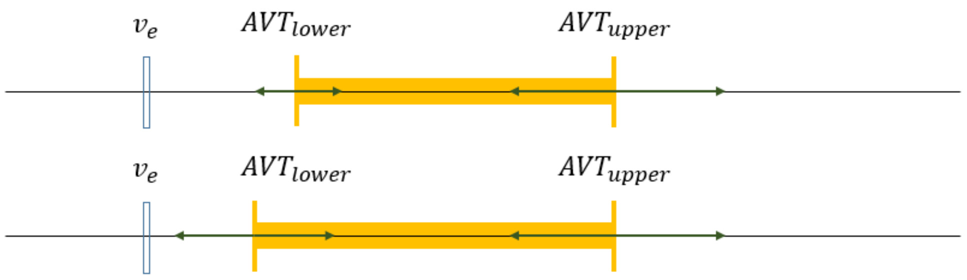

Figure 5.10 — An illustration of the behaviour of an AVRT with an example (v_e, IN). Here, a feedback $\downarrow$ is sent to the AVR_{lower} to integrate v_e to the range.

▷ The example (v_e, OUT) is included in $[v_{lower}, v_{upper}]$. The *Adaptive Value Range Tracker* has to adapt its bounds to exclude the example. If $|v_{upper} - v_e| >= |v_{lower} - ve|$, a feedback $\downarrow$ is sent to AVT_{upper}. Else, a feedback $\uparrow$ is sent to AVT_{lower}.

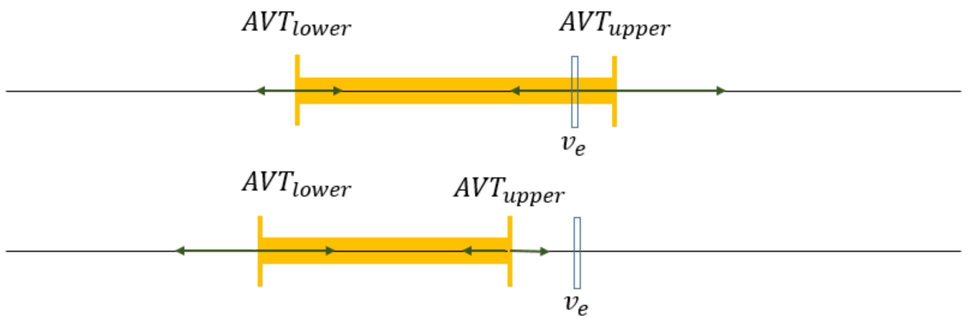

Figure 5.11 — An illustration of the behaviour of an AVRT with an example (v_e, OUT). Here, a feedback $\downarrow$ is sent to the AVR_{upper} to integrate v_e to the range.

▷ The example (v_e, IN) is included in $[v_{lower}, v_{upper}]$ or the example (v_e, OUT) is not included in $[v_{lower}, v_{upper}]$. The current bounds fits with the example. A feedback $\sim$ is sent to AVT_{upper} and AVT_{lower}.

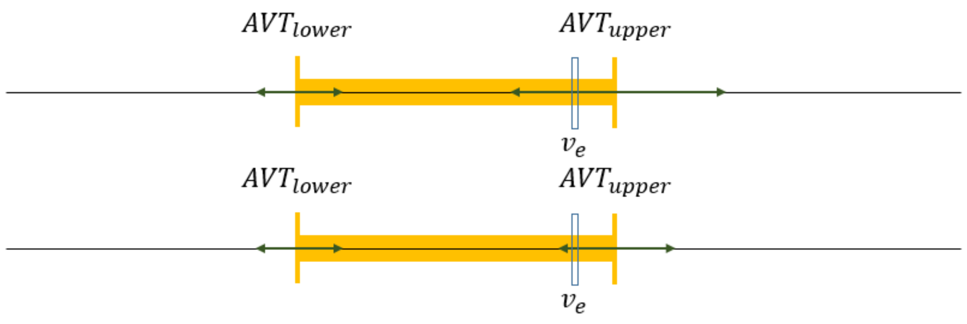

Figure 5.12 — An illustration of the behaviour of an AVRT with an example (v_e, IN). Here, a feedback $\sim$ is sent to the AVR_{upper}.

The integration of a value into the range of an *Adaptive Value Range Tracker* is not necessary instantaneous. If the Δ of an *Adaptive Value Tracker* is lower than the distance between the current value of this *Adaptive Value Tracker* and the value to include or exclude, one feedback is not enough to include or exclude this value.

The usage of *Adaptive Value Range Trackers* presents some interests for our application:

▷ As *Adaptive Value Range Trackers* are based on *Adaptive Value Trackers*, they keep the ability to model a range even if the sought range to model changes with time.

▷ Due to the fact that a value is not directly integrated to the range and the inertial behaviour of AVT, *Adaptive Value Range Trackers* have some statistical resiliency to incorrect signals.

5.5.3 Validity Ranges

In section 5.3.2.2 we describe the nominal behaviour of Context Agents and introduce the concept of validity ranges. A *Context Agent* is composed of a set of validity ranges, one for each Percept Agent. This set of validity ranges describes the *context* in which the *action* of the *Context Agent* is applicable. Validity ranges are composed of two bounds $[vrange_{min}, vrange_{max}]$ and an internal representation of the current value $v_{percept}$ of its associated Percept Agent. This set of validity ranges enables the *Context Agent* to determine its validity and to propose or not its *action*.

Modelling the *context* description with validity range is choosing a structure that does not require any kind of semantic. However, it requires that there exists an order relation among the different values of an observation. As we intend to use our algorithm on real-world applications using numeric data, this assumption is not a strong one. However, in other kind of applications, such assumption could present limitations and the mechanisms introduced in this section could seem inappropriate. The model presented in this section is not the only one application to solve the non cooperative situation and can be easily changed with others that respond to the non cooperative situations listed in the previous section.

A key component of *Context Agents* is their ability to manage their *context* description, and by extension their validity ranges. Indeed, some non cooperative situations are solved by the inclusion or the exclusion of the current situation from the *context* of *Context Agents*. They must tune their validity ranges in order to be *valid* when their *action* has to be applied and *invalid* otherwise.

Validity Ranges are modelled using *Adaptive Value Range Trackers* (section 5.5.2). Then, each validity range is composed of an *Adaptive Value Range Tracker* and the current value $v_{percept}$ of the associated Percept Agent. Such as *Adaptive Value Range Trackers*, validity ranges adaptation is made through three kinds of signal:

▹ An *exclusion* signal, which asks for the validity range to exclude the current *Percept Agent* $v_{percept}$ value from its interval. An example $(v_{percept}, OUT)$ is then sent to the *Adaptive Value Range Tracker*.

▹ An *inclusion* signal, which asks for the validity range to include the current *Percept Agent* $v_{percept}$ value to its interval. An example $(v_{percept}, IN)$ is then sent to the *Adaptive Value Range Tracker*.

▹ A *confirmation* signal, which validates that the current interval is satisfying. It corresponds to the feedback $\sim$ of the *Adaptive Value Range Tracker*.

A validity range which includes the current value of $v_{percept}$ is said to be *valid*. A validity range which does not include the current value of $v_{percept}$ is said to be *invalid*. A validity range which can include the value of $v_{percept}$ by sending only one feedback to the *AVRT* is said to be *validable*. A validity range which can exclude the current value of $v_{percept}$ by sending only one feedback to the *AVRT* is said to be *invalidable*. In parallel, a *Context Agent* is said *valid* if all of its validity ranges are *valid*, *validable* if all its validity ranges are either *valid* or *validable*, and *invalid* if there is at least one *invalid* validity range.

When a *Context Agent* seeks to adapt its *context* description, it has to send the adequate signal to each of its validity range. According to NCS description (see section 5.4), four cases can occur depending on the state of the Context Agent, the desired state of the agent and the current state of each validity range:

▹ The *Context Agent* is *valid* and seeks to be *invalid*:

In NCS3 (see section 5.4.4), a *valid Context Agent* proposes an *action* that is not in adequacy with the Tutor *action*. The *Context Agent* must adapt its *context* description to exclude the current situation in order to be *invalid* the next time a similar situation will occur. In order to be *invalid* the *Context Agent* requires that at least one of its validity range is *invalid*. Two situations have to be differentiated depending on if there is at least one validity range that is *invalidable* or if no validity range is *invalidable*. If their is only one validity range that is *invalidable*, the *Context Agent* sends to this validity range an exclusion signal. Then, the next time the same situation will occur, the *Context Agent* will be *invalid*. However, if there are more than one validity range that is *invalidable*, the *Context Agent* is unable to decide which validity range has to exclude the current situation. Then, the *Context Agent* asks to every *invalidable* validity range to exclude the current situation. In the same way, if there is no *invalidable* validity range, the *Context Agent* sends to all validity range an *exclusion* signal.

In NCS4 (see section 5.4.5), a *Context Agent* is in concurrency with another Context Agent. If there is at least one validity range that is not *valid*, the *Context Agent* sends an exclusion signal to every *invalidable* validity ranges. By doing so, the *Context Agent* makes sure not to be *valid* the next time a similar situation will occur. However, if no validity range is not *valid*, the resolution of the situation does not pass through an adaptation of the *context* but is made through the *confidence* value (see section 5.5.4).

▷ The *Context Agent* is *validable* and seeks to be *valid*:

A mechanism has been introduced in NCS5 to anticipate situations of incompetence. This mechanism states that *Context Agents* propose their actions in situations that are closed to their *context* description but are not included in it (see section 5.4.6). The concept of *validable* enables the *Context Agent* to determine if a situation is close enough to its *context* description. Then, *Context Agents* that are *validable* propose their *action*. If the proposed *action* is applied, the *Context Agent* has to include the current situation to its *context* description. Then, for each *validable* validity ranges, the *Context Agent* sends an *inclusion* signal.

▷ The *Context Agent* is *validable* and seeks to be not *validable*:

This situation is the opposite of the previous one. A *validable Context Agent* has proposed its *action*, but this *action* is not in adequacy with the Tutor one. The *Context Agent* has to adapt its *context* description in order to not be *validable* the next time a similar situation occurs. Then, for each *validable* validity ranges, the *Context Agent* sends a *confirmation* signal. The effect of the confirmation signal is to reduce the Δ of each *AVT*, and by extension to reduce the area in which the *Context Agent* is *validable*.

▷ The *Context Agent* is *invalid* and seeks to be *valid*:

Whenever a *Context Agent* is *invalid* and seeks to be *valid*, the *Context Agent* sends an *inclusion* signal to every *invalid* validity ranges. However, it does not ensure that the *Context Agent* will effectively be *valid* the next time a similar situation will occur. So, this situation only happens when there are no other solutions for the Context Agent.

5.5.4 Confidence Value

The NCS3.b (see section 5.4.4) introduced the concept of a *confidence* value. The idea behind the *confidence* value is that an information about the utility the *action* of a *Context Agent* would help the *Input Agents* to realise its nominal behaviour. In our application, a notion of utility is obscure as what is useful could only be evaluated by the Tutor. Indeed, a useful action is the one that satisfies the Tutor. But the problem is how to model or observe Tutor satisfaction without making some hypothesis on this Tutor?

Instead, we propose to use a *confidence* value that is only updated through a local evaluation of the *Context Agent* activity. The idea is not to evaluate the utility of an *action* from its functional aspect, which means by evaluating how the action would impact the Tutor satisfaction, but to evaluate the activity of the *Context Agent* among the system, by observing if its proposal are leading or not to non cooperative situations. The *Context Agent* can then self-build a *confidence* value which is only based on an evaluation of the quality of its interactions.

The functions to implement this *confidence* value are numerous and the only requirement is that every *Context Agents* use the same process to build the *confidence* value. Then, we propose that the *confidence* value c is an integer such as $c \in [0;1]$. To update the *confidence* value, we propose to use the lambda function $c_{t+1} = c_t * (1 - \lambda) + R * \lambda$ where $\lambda \in [0;1]$ and $R \in [0;1]$ is a feedback. The parameter λ moderates the impact of a feedback on the *confidence* value. A R value close to 1 increases the *confidence* value whereas a value close to 0 decreases the confidence value. The more λ is high, the more the *confidence* value evolves rapidly. For example, if $\lambda = 0.1$, the confidence value will be increased by 10%. One feedback will have a low influence on the *confidence* value whereas many successive feedbacks will have a significant impact. The more a *confidence* value receives feedback $R = 1$, the more its value converges toward 1 without reaching it. In parallel, the more a *confidence* value receives feedback $R = 0$, the more its value converges toward 0 without reaching it. Then, when a *Context Agent* increases its *confidence* value, the new *confidence* value is calculated with the previous formula and a parameter R equals to 1. When a *Context Agent* decreases its *confidence* value, the new *confidence* value is calculated with the previous formula and a parameter R equals to 0. We fixed empirically λ at 0.1. This parameter has few impact on *Context Agents* behaviour as long as each agent uses the same function to calculate its *confidence* value.

Now that the *Context Agent* have the ability to increase or decrease its *confidence* value, we now have to describe the situation in which the *Context Agent* updates its *confidence* value on how this value is effectively used.

Each time the *Input Agents* applies an *action* coming from the Tutor, the *Input Agents* sends a feedback to each *Context Agent* that made a proposal describing the *action* $action_{input}$ that was performed and the *confidence* value c_{input} associated to this *action*. The *confidence* value c_{input} corresponds to the highest confidence value of *Context Agents* proposals that are in adequacy with the Tutor action, and 0 if no *Context Agent* was proposing this *action*. Each *Context Agents* use this information to both manage their context description (see section 5.4) and update their *confidence* values. Three situations involves the modification of the *confidence* value:

- ▷ The proposal of a *Context Agent* corresponds to the feedback of the *Input Agent*. The *Context Agent* increases its *confidence* value as its *context* description is in adequacy with the Tutor.
- ▷ The *action* proposed by a *Context Agent* does not correspond to the one applied by the *Input Agent*. This situation triggers the mechanism introduced by the non cooperative situation NCS3.a (see section 5.4.4). The *Context Agent* tries to exclude the current situation from its Context description. If there are no *invalidable* validity range, the *Context Agent* will not succeed to exclude the current situation. Then, the *Context Agent* decreases its confidence value, as it fails to manage its *context* description.
- ▷ At last, when a *validable Context Agent* has made a proposal that is not in adequacy with the *action* applied by the *Input Agent*, the *Context Agent* confirms its validity ranges

in order to not be *validable* the next time the same situation will happen (see section 5.4). If after sending the confirmation signal, the *Context Agent* is still *validable* (due to the fact that values are not directly excluded from *Adaptive Value Range Trackers*), the *Context Agent* decreases its *confidence* value as it failed to manage its *context* description.

By increasing its *confidence* value when its *context* description leads to cooperative interactions and decreasing it when its context description leads to non cooperative interactions, *Context Agents* self-manage a *confidence* value describing the adequacy of their proposals. By using this value, the *Input Agents* can then select the most confident *Context Agents*, which correspond to *Context Agents* that have had the most *cooperative* interactions.

5.5.5 Context Agent Initialisation

The NCS1.a (see section 5.4.2) is resolved by the creation a new *Context Agent* associated with the Tutor *action* and initialised to describe the current situation. The initialisation of a *Context Agent* consists in the initialisation of its validity range and its *confidence* value.

▷ **Validity Range initialisation**: a validity range is created and associated with each Percept Agent. Each bound of the *Adaptive Value Tracker* is initialised around the current value of the Percept Agent. In order to do so, a parameter σ enables the initialisation of each bound of the *Adaptive Value Range Tracker* such as $v_{lower} = v_{percept} - \sigma$ and $v_{upper} = v_{percept} + \sigma$. Thus, the AVT_{upper} is initialised to a value of $v_{percept} + \sigma$ and an initial Δ_{lower} of σ and the AVT_{lower} is initialised to a value of $v_{percept} - \sigma$ and an initial Δ_{upper} of σ. σ is empirically fixed at half of the last variation such as $\sigma = (v_t - v_{t-1})/2$. This enables the *Context Agent* to be *valid* on the current value of the *Percept Agent* and *validable* to the previous value. The figure 5.13 illustrates the initialisation of a validity range.

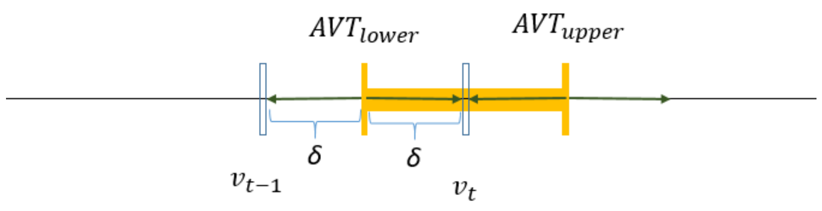

Figure 5.13 — Initialisation of a validity range around the current value v_t. Each bound is placed at a distance $\sigma = (v_t - v_{t-1})/2$ of the value v_t

▷ **Confidence initialisation**: the *confidence* is arbitrary initialised to a value of 0.5. The value itself does not matter as long as the initialisation value is the same for every Context Agent.

5.5.6 Percept Agent Neighbourhood

In NCS7 (see section 5.4.8) and NCS8 (see section 5.4.9), we extend *Percept Agent* abilities by adding the ability to manage their neighbourhoods. When a *Percept Agent* is created, its neighbourhood is empty as no *Context Agent* requires its updates. Whenever a new *Context Agent* is created, the *Context Agent* is initialised to be *valid* in the current situation (see section 5.5.5) and then requires the updates of every Percept Agent. Thus, the *Context Agent* is added to the neighbourhood of every Percept Agent.

Every time a *Percept Agent* sends an *update*, the recipient *Context Agent* updates the status of the associated validity range. If the validity range was *valid* and becomes *invalid* after the update, the *Context Agent* informs the other *Percept Agents* that their updates is not necessary. When a *Percept Agent* receives a removal message from a Context Agent, the *Percept Agent* removes the *Context Agent* from its neighbourhood. Thus, the *Context Agent* will now only receive update from the *Percept Agent* that has made its validity range *invalid*. If, after another update, this validity range becomes *valid* again, the *Context Agent* sends a message to every other Percept Agents. When a *Percept Agent* receives this message, it adds the *Context Agent* to its neighbourhood.

5.5.7 Agent Scheduling

The last item to be addressed in this section is agent's life-cycle scheduling. In the previous sections, we have described the behaviour of each agent as a three steps life-cycle of Perception-Decision-Action. In this section, we address the issue of how those life-cycles are activated. This involves to go back to the macro-level and to consider ALEX as a software component.

In section 5.2, we have described ALEX environment and proposed a component view of ALEX. This component view has three interfaces which correspond to the three functions proposed by ALEX: a function which enables to send Tutor actions, a function to send updates on observations, and a function to receive the action proposed by ALEX.

The reception of an update triggers the life-cycle of the *Percept Agent* associated with this observation. During its life-cycle, the Percept-Agent sends updates to the *Context Agent* and partly activates its life-cycle. A *Context Agent* which becomes *invalid* after an update sends a message to the *Percept Agents* in order to be removed from the *Percept Agents* neighbourhoods (see section 5.4.8). However, this does not trigger the action proposal.

Whenever an external entity asks for ALEX which is the action to perform, the decision life-cycle of each *valid* or *validable Context Agent* is triggered. When all *Context Agents* have sent their proposals, the life-cycle of the *Input Agents* is then triggered, resulting in the output action proposed by ALEX.

By doing so, cycle management is externalised enabling ALEX to be used in various systems with different dynamics. Indeed, the frequency of the decision cycle is dependent of the domain in which ALEX is used, but does not affect the way ALEX behaves.

5.6 Algorithms

Using the description of nominal behaviour (see section 5.3), the description of non cooperative situations (see section 5.4) and the proposed mechanisms to solve those situations (see section 5.5), we propose for each Agent a pseudo-code algorithm describing its behaviour.

5.6.1 Percept Agent Behaviour

Algorithm 5.1: Life cycle of a Percept Agent

```
Receive observation signal;
Receive Context Agents messages;
Update neighbourhood;
Update knowledge;
Send updates to neighbourhood;
```

Percept Agents are the one responsible of the observation of the environment. The algorithm 5.1 illustrates their behaviour. They receive updates from the environment, manage their neighbourhood and send updates to *Context Agents*.

5.6.2 Input Agent Behaviour

Algorithm 5.2: Life cycle of the Input Agent

```
Receive $action_{tutor}$ ;
Receive Context Agents proposals ;
if Tutor action available then
    if No proposal is in adequacy with $action_{tutor}$ then
        Create a new Context Agent associated with $action_{tutor}$;
    end
    Apply $action_{tutor}$;
else
    if Proposals have been received then
        Select the action with the highest confidence from proposals;
    else
        Maintain previous action;
    end
    Apply selected action;
end
Inform Context Agents ;
```

The *Input Agents* is the one responsible of the application of *actions*. The algorithm 5.2 illustrates its behaviour. The *Input Agents* receives the Tutor activity and proposals from *Context Agents*, applies the adequate *action* and sends feedback to *Context Agents*.

5.6.3 Context Agent Behaviour

Context Agents are the one with the most non cooperative situations. The resolution of those non cooperative situations lead to the tuning of the parameter of *Context Agents*. The algorithm 5.3 illustrates their behaviour.

Algorithm 5.3: Life cycle of the Context Agent

```
forall the updates from Percept Agents $p$ do
    Update validity range $v_p$ associated with $p$;
    if $v_p$ is now invalid then
        NCS7: Send removal message to all other Percept Agents ;
    end
    if $v_p$ is now valid then
        NCS8: Send inscription message to all other Percept Agents ;
    end
end
if The Context Agent was valid or validable at previous step then
    NCS3,4,5: Receive feedback ($action_{input}, confidence_{input}$ from Input Agent ;
    if $action != action_{input}$ then
        if It exists at least one invalidable validity range then
            NCS3: For each invalidable validity ranges send a feedback to exclude the
            current situation ;
        else
            NCS3: For each validity ranges, send a feedback to exclude the current
            situation ;
            NCS3,4,5: Decrease $confidence$ ;
        end
    else
        if $confidence != confidence_{input}$ then
            if It exists at least one invalidable validity range then
                NCS4: For each invalidable validity ranges send a feedback to exclude
                the current situation ;
            else
                NCS3,4,5: Decrease $confidence$ ;
            end
        end
        if The Context Agent is validable then
            NCS5: For each validable validity range send a feedback to include the
            current situation ;
        else
            NCS3,4,5: Increase $confidence$ ;
        end
    end
end
if The Context Agent is validable then
    NCS5: Send proposal ($action, confidence$);
end
if The Context Agent is valid then
    Send proposal ($action, confidence$);
end
```

5.7 ALEX Differences with Other Approaches

The bottom-up design of ALEX and the locality of its behaviour are in complete rupture with the traditional engineering approach presented in chapters 2 and 3. On this section, we propose to highlight the points that makes ALEX different.

5.7.1 Differences with Learning Classifiers

A *Learning Classifier System* (see section 2.6.1) is a reinforcement learning system composed of a set of rules, a matching system linking the state of the environment with the conditions of the rules, a mechanism to select rules among the activated ones, and an algorithm for rules evolution.

A parallel is easily drawn between ALEX and Learning Classifier Systems. Indeed, *Context Agents* can be seen as the combination of the matching function (with the validity ranges) and the set of rules (by proposing their actions), and the *Input Agents* can be seen as the mechanism of rule selection.

However, with ALEX, each *Context Agent* is autonomous and learns by itself, whereas, in Learning Classifier Systems, a genetic algorithm is used to make the set of rules evolve. The fitness function used by the genetic algorithm of an Learning Classifier System is a feedback perceived from the environment and is relative to the task to perform. The difficulty with a Learning Classifier System is both the instantiation of this feedback function, which is dependent to the task to perform, and its repartition among the different rules. Such difficulty does not exist with ALEX as each *Context Agent* generates is own feedback through local self-observation capacities. Each *Context Agent* locally evaluates its adequacy and self-adapts in response. Then, the "feedbacks" used by ALEX are completely *agnostic* to the task to perform.

A second level of differentiation is the selection mechanism. In Learning Classifier Systems, each rule is associated with a probability of activation, and the selection mechanism is based on those probability of activation. But the selected rule is not obligatory the rule with the higher probability. In ALEX, the *Input Agent* behaviour does not truly operate a selection of the rule to apply as its decision is always to select the proposal with the highest *confidence* value. This *confidence* value is self-managed by each Agent Context. Then, the selection mechanism is distributed among each *Context Agent* and the *Input Agents* just operates the last phase of the selection, which is the application of the action.

5.7.2 Differences with Cased-Based Reasoning

Case-Based Reasoning (see section 2.6.2) is a solving process of new problems based on the solutions of analogous past problems. Each case of a Case-Based Reasoning system is the association of a problem description and a representation of a solution. It might be tempting to draw an analogy between ALEX and a Cased-Base reasoning systems, where *Context Agents* can be seen as the different solutions to reuse. However, a *Context Agent* is not a the solution of a problem but an actor of its resolution (this point is a fundamental part of the AMAS approach and the bottom-up design of an artificial system). Indeed, a Context Agent, by itself, is just the expression of the local adequacy of an action. The fact that a *Context Agent* is *valid* does not involve that its action is going to be applied. The solution is the one that results from the interactions between all the agents of an ALEX instance. This is why ALEX is not a Cased-Based Reasoning system.

5.7.3 Differences with Artificial Neural Networks

Artificial Neural Networks are composed of interconnected neurons, where each neuron is a small computational unit with inputs, outputs, an internal state and parameters (see section 2.3.1). One could see *Context Agents* as "advanced" neurons, and the architecture of ALEX, where messages navigate from *Percept Agents* to the *Input Agents* through the different *Context Agents*, as a form of multilayer perceptron. However, we have seen that information within ALEX is not feed-forward (contrary to neural networks). For example, when a *Percept Agent* sends an update message to a *Context Agent*, this *Context Agent* can directly reply to the *Percept Agent* with a *removal* message. The main difference between ALEX and Artificial Networks is the organization of the interaction between the entities. With artificial Neural Network, the topology of the network has to be fixed *a priori* in regards with the task to learn whereas within ALEX, agents self-organize and the topology (the number of agents and the way they interact) evolves dynamically to resolve situations of non cooperation.

5.7.4 Differences with Schema-Learning

ALEX possesses undeniable similarities with Schema-Learning, the most notable being the tripartite structure of $< Context, Action, Result >$ of schemas and *Context Agents* (see section 2.5.1). But while Schema-Learning seeks to model the regularities of interactions of a system with its environment, ALEX seeks to retain only its cooperative ones (cooperative in the sense of the AMAS approach see section 4.3). Furthermore, Schema-Learning does not include the notion of control.

However, the similarity in the structures is interesting. Indeed, Schema-Learning is inspired by the Piaget's theory of cognitive development. Schema-Learning is the result of a design process which goes from the study of human behaviour towards artificial systems. On the opposite, ALEX, which is based on the AMAS approach, is built from the cybernetic towards artificial systems. The fact that these two approaches, which come from different horizons with different motivations converge toward a similar structure is probably an interesting axis for future researches.

5.8 Synthesis

In this chapter, we have presented ALEX, our contribution to enable real-time learning of a control policy from demonstrations performed by an external entity. ALEX is designed to control a functionality, seen as a Simple-Input Simple-Output system, by applying the adequate control action, which means by modifying the current input of the functionality. The structure of ALEX, from the description of its nominal behaviour to the proposition of mechanisms to solve non cooperative situations, has been presented in details in this chapter. The behaviour of ALEX is the result of interactions between the different kinds of agents composing it: *Percept Agents*, which build and share a representation of the environment, *Context Agents*, which make action proposals, and a unique *Input Agent*, which is responsible of the effective control. Before the presentation of the evaluation of ALEX on different use-cases, and now that the reader has acquired an expertise on how ALEX behaves, we propose to a focus on self-organization with ALEX.

5.8.1 Self-Organization and ALEX

ALEX is an *Adaptive Multi-Agent System*. Within an *Adaptive Multi-Agent System*, self-organization is driven by cooperation. By solving non cooperative situations, agents adjust their internal parameters, change their interactions or create and suppress other agents. Those different actions lead to change in the structural organization of the agent and in the organization of the interaction between the agents. Thus we can consider self-organization from different points of views.

At the level of Percept Agents *Percept Agents* are agents which observe the external environment of ALEX to build and share an internal representation of it. By interacting with *Context Agents*, *Percept Agents* adapt their neighbourhood in order to only send updates to *Context Agents* that require them (see section 5.5.6). This can be seen as a processes of self-organization where *Percept Agents* and *Context Agents* adapt the way they interact in reaction to the environment in order to reach an organization that ensures to the system to realise its nominal behaviour.

At the level of Context Agents At start, ALEX is empty of any *Context Agent*. They are dynamically created by the *Input Agents* in order to solve non cooperative situations. Each *Context Agent* follows simple rules to adjust its behaviour. The combination of these rules and *Context Agent* creation leads to the creation of a collective of *Context Agents*, each *Context Agent* modelling, with its *context* description, a portion of the observation space where its *action* is applicable. Thus, each *Context Agent* can be seen as a tile, whose boundaries are represented by the bounds of its validity ranges. *Context Agents* are continuously adapting those boundaries to solve non cooperative situations. From this adaptation results a change in the way *Context Agents* interact with the *Input Agent*. Collectively, *Context Agents* pave the observation space with tiles (which can overlap) to build a mapping policy enabling ALEX to control its functionality. Thus, they continuously and autonomously change of organization (the paving) in reaction to changes in the environment. The group composed by *Context Agents* and the *Input Agents* is then self-organized.

At the macro level An instance of ALEX takes local decisions in order to control its functionality. Those decisions are the result of a self-organization process of the inner agents in reaction to changes in the environment. From the macro level, we then observe that

ALEX adapts its behaviour to its environment. But another level of self-organization can be considered at this level with multiple ALEX instances. Indeed, an instance of ALEX controls a unique functionality, but systems are composed of many functionalities immersed in the same environment. While each ALEX is autonomous and takes only local decisions, it is required that the collective of ALEX collaborate in order to perform a coherent activity. The behaviour of an ALEX instance is conditioned by its environment, modelled by the set of observations. As each ALEX observes and evolves in the same environment (considering application to robotics, this environment is the real-world), the set of observations of an ALEX instance is partly influenced by the activity of the other ALEX (this is particularly true if one of those observation is a direct observation of the current state of an ALEX instance). Then, the proposals of *Context Agents* inside an ALEX instance are partly conditioned by the activity of the other ALEX instances. The behaviour of an ALEX instance is then influenced by the behaviour of the other ALEX, and reciprocally. So, the whole collective of ALEX is self-organizing, each ALEX taking only local decisions, to reach a coherent global activity. This self-organization at the macro-level is the ultimate goal of the Extreme Sensitive Robotic paradigm, and the one sought by this thesis (see section 1.6).

5.8.2 And What About the Emergence?

Is the behaviour of a *Context Agent* emergent? And what about the one of an ALEX instance? And the collective? Those questions are far from trivial. We have seen in section 4.1 that defining the emergence is delicate, and it is something that depends on the knowledge of the observer and its position among the system. The direct answer to these questions in this document would be biased by theoretical and empirical knowledge acquired by the redactor of this thesis on his own system. Instead, we provided through the focus on self-organization, elements of answers and let freedom to the reader to ponder the emergent nature of ALEX.

6 Experimentations

This last chapter discusses of the ***evaluations*** *of ALEX in three different experimentations. For each experimentations, we present the hypothesis to test, the motivations and we discuss of the results we obtained. This chapter concludes with a general synthesis pointing out ALEX properties and limitations.*

IN this chapter, we propose to study ALEX behaviour through a set of experimentations. The first experiment, entitled "The Mountain Car Problem", is a study of a toy problem used to both illustrate ALEX behaviour and to set up pointers to compare our approach to others. The second experiment is entitled "Teach Robot Yourself" and proposes to teach to a two-wheeled rover a task of collect. The last experiment proposes to study the usage of ALEX in an iterative design process of a navigation task. Before concluding, we discuss of ALEX applicability in the context of *Industries 4.0*. At last, the chapter concludes with a discussion on the ongoing experiments.

For each experiment, we first introduce the context of the experiment and draw the objectives and motivations. Then, we present the protocol and discuss the results we obtained.

6.1 The Mountain Car Problem

Objectives:

- ▷ Illustrate and explain ALEX behaviour.
- ▷ Show ALEX capacity to imitate a policy.
- ▷ Show ALEX capacity to learn from human demonstrations.
- ▷ Position the learning with Context Agents to other learning techniques.

Figure 6.1 — Objectives of the Mountain Car Experiment

6.1.1 Problem Description

Initially introduced by [Moore, 1990] and lately defined by [Sutton and Barto, 1999], the Mountain Car is a well-known toy problem mainly studied in the field of reinforcement learning (figure 6.2). An under-powered car situated in a valley must drive up a steep hill. The gravity is too strong for the car's engine so that the car cannot only speed up to climb the hill. The car has to learn to leverage potential energy by driving up to the opposite hill in order to gain enough energy to climb the other hill.

The problem is described by [Sutton and Barto, 1999] as follows:

- ▷ A two dimensional continuous state space:

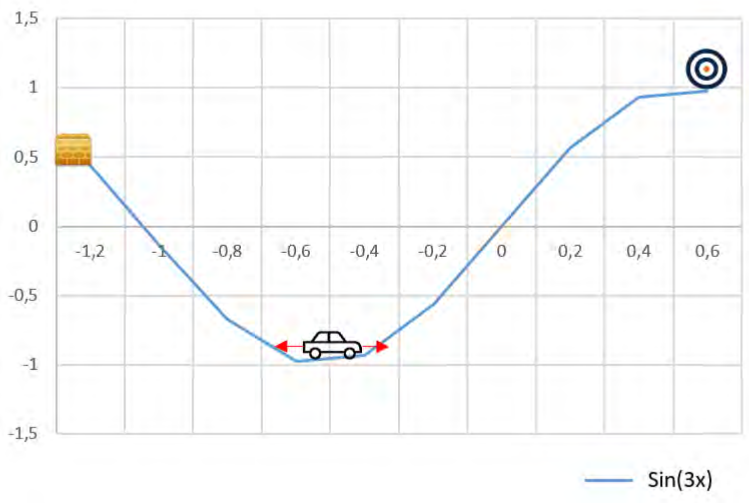

Figure 6.2 — An illustration of the mountain car. An under-powered car situated in a valley must drive up a steep hill and reach the target.

  - $Velocity \in [-0.07, 0.07]$
  - $Position \in [-1.2, 0.6]$

- ▷ A one-dimensional discrete action space:
  - $Motor \in [left, neutral, right]$
- ▷ An update function performed at every step:
  - $Action \in [-1; 0; 1]$ each value corresponding to one element of the action space.
  - $Velocity \leftarrow Velocity + (Action) * 0.001 + cos(3 * Position) * (-0.0025)$
  - $Position \leftarrow Position + Velocity$
- ▷ An instant reward function:
  - $Reward \leftarrow -1 + height$ with height 0 being the lowest point in the valley.
- ▷ An initial condition:
  - $Position \leftarrow -0.5$, which corresponds to the hollow of the hill.
  - $Velocity \leftarrow 0.0$
- ▷ A termination condition:
  - $Position >= 0.6$, which corresponds to the top of the opposite hill.

The initial problem is then to build a policy that enables the car to reach its target while minimizing the number of steps.

6.1.2 Motivations and Objectives

The Mountain Car problem is a well-known benchmark for reinforcement learning problem used to evaluate learning algorithms [Gatti, 2015]. As a matter of fact, the scientific literature is rich of results which can be compared. However, the initial problem is a problem of optimisation where an optimal control policy has to be found through the exploitation of a reward function. Thus, it does not include any notion of tutoring and may first appear to be irrelevant to test our approach. But adding a tutor to this problem is not a complex task. Indeed, we can easily hand-craft a policy (optimal or not) to control the car, or we can even use the policy learned by any reinforcement algorithm, and use those policies to perform the demonstrations. The imitation task will then be to learn a new policy that mimics the one used during the demonstration. Using a *virtual* tutor presents the interest that the likeness of the learnt policy is easily measurable. Furthermore, the task is easily feasible by a human tutor as the strategy which consists in gaining momentum to climb the hill is highly intuitive.

Thereby, the problem has been recently used as a benchmark to compare approaches using inverse reinforcement learning, a subclass of the LfD approach, where demonstrations of human tutors are used to infer a feedback function which is then used by a reinforcement

Algorithm 6.1: Virtual tutor policy.

```
if $Velocity > 0$ then
    return 1 ;
else
    return $-1$ ;
end
```

learning algorithm to optimize a control policy [Celemin and Ruiz-del Solar, 2015],[Knox et al., 2011]).

Another interest in the Mountain Car problem is its two dimensional states space which allows to draw a 2D projection of a policy and to easily compare policies. By this mean, we intend to illustrate how Context Agents inside ALEX populate and map cooperatively the states space.

ALEX objective is not to learn an optimal policy, but to mimic the behaviour of its tutor. Then, the evaluation will address the functional adequacy (does ALEX manage to learn a policy to control the car and reach the top of the hill), the likeness of this policy (does ALEX perform the task as well as what has been demonstrated), and the number of demonstrations required to learn a functionally adequate policy.

6.1.3 Experimental Process

6.1.3.1 Introducing the Tutors

The mountain car is initially a problem for reinforcement learning techniques and does not include in its initial description the notion of tutoring. Thus, we propose to add to the initial problem description a tutor which will act as an *oracle* during the demonstration. The Tutor follows a policy $\pi_{tutor} : (Velocity, Position) \rightarrow Action$ that associates to any situation of the states space, described by the current *Velocity* and *Position* values, an action to perform. As we are not interested in finding an optimal policy to control the car in the fewest steps possible but to mimic the demonstrated behaviour, Tutor's policy is considered to be the optimal behaviour to satisfy the Tutor. We want to consider two types of situations with two different Tutors: a *virtual* tutor and a *human* tutor.

Virtual Tutor In the first situation, the tutor is *virtual*, which means that the demonstration is performed by an handcrafted algorithm. The *virtual* tutor acts as an *oracle* providing at any step the action to perform by following the algorithm 6.1. With this policy, the car can pass the hill in 117 steps. The virtual tutor strategy is interesting because it is only based on the current *velocity* of the car. Thus, the current *position* of the car is a useless data.

Human Tutor In the second situation, the demonstration is made by a human tutor through a keyboard interface. *Left* and *Right* keys enable the Tutor to control the *right* and *left* movement of the car. If no key is tapped, the action of the Tutor is considered to be

Algorithm 6.2: The mountain car problem pseudo-code of one episode.

```
$Velocity \leftarrow 0.0$ ;
$Position \leftarrow -0.5$ ;
while $Position < 0.6$ do
    $ALEX.updateObservations(Velocity, Position)$ ;
    if $isTutoring$ then
        $ALEX.newTutorAction(TutorAction())$ ;
    end
    $Action \leftarrow ALEX.getAction()$ ;
    $Velocity \leftarrow Velocity + (Action) * 0.001 + \cos 3 * Position * (-0.0025)$;
    $Position \leftarrow Position + Velocity$ ;
end
```

neutral. A step occurs every $250ms$. The human tutor disposes of a graphical representation of the current situation (see figure 6.3) where the blue ball models the car and the green block the goal to reach.

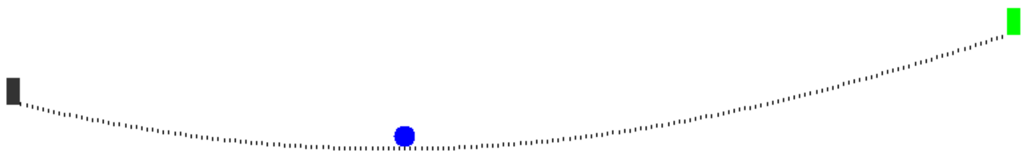

Figure 6.3 — The graphical representation of the Mountain Car for human tutoring.

6.1.3.2 Experiment Implementation

Experiments are composed of one instance of ALEX which has to learn to control the car. Each experiment is performed by alternating an episode of *demonstration* with an episode of *autonomy* (one episode refers to the completion of the task, ie to the fact that the car has successfully climbed the top of the hill):

▷ During an episode of *demonstration*, the car is controlled by a tutor (either virtual or human) while ALEX is observing tutor's actions. The ALEX instance controlling the car receives three signals: the current *Velocity*, the current *Position* and the action of the tutor $Action_{tutor}$.

▷ During an episode of *autonomy*, the car is controlled by ALEX and the tutor makes no more demonstration. The ALEX instance now receives two signals, the current *Velocity* and the current *Position*, and must provide the action to perform $Action_{ALEX}$.

The algorithm 6.2 illustrates the course of one episode.

An experiment is set-up for both types of tutor, starting with the virtual tutor. An experiment then consists in the achievement of many episodes. *Context Agents* are reset between each experiment, but kept between each episode. The same implementation of ALEX is used in every experiment without any tuning of parameter.

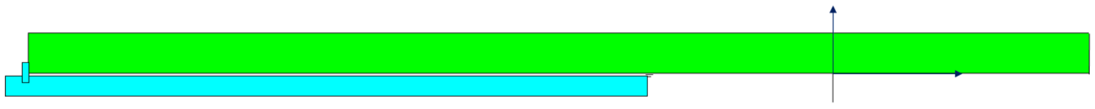

Figure 6.4 — The 2D projection of the policy learnt by ALEX with a *virtual* tutor. The space is split similarly to the policy of the *virtual* tutor.

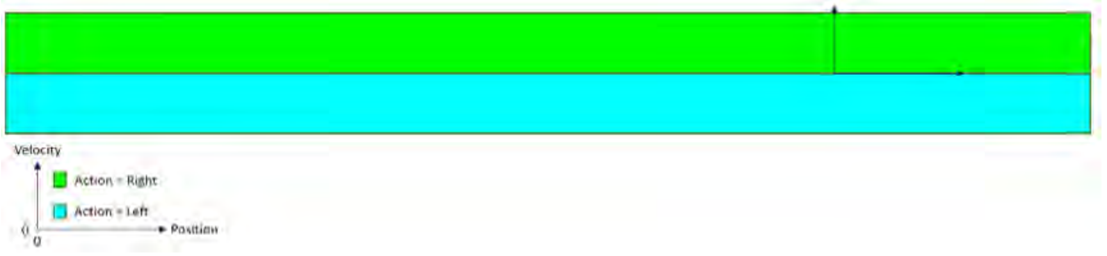

Figure 6.5 — The 2D projection of the *virtual* tutor policy. The space is split in two areas depending on the *velocity* value.

In each experiment, the functional adequacy (the ability of the system to perform its task) is the main criterion of evaluation of the episodes of autonomy.

The evaluation of the behaviour of ALEX is made through the analysis of the evolution of the confidence value of *Context Agents* and the 2D projections of the learnt policy.

6.1.4 Analysis

6.1.4.1 Learning from a Virtual Tutor

The first criterion of the analysis is the functional adequacy. In a unique episode of demonstration, ALEX succeeds to map the observation space with enough *Context Agents* in order to be able to reach the top of the hill.

A 2D projection of the learnt policy is visible in the figure 6.4 to illustrate how the Context Agents occupy the space of the problem. The horizontal axis is for the *Position* value and the vertical axis for the *Velocity* value. The different squares model the subspace in which a particular Context Agent is *valid*. Green squares are *Context Agents* proposing the *right* action whereas blue squares are Context Agents proposing the $left$ action. In cases of overlapping *Context Agents*, *Context Agents* with the higher confidence value are put in front of the others. Then, the figure describes for each point of the space which will be the action proposed by ALEX. The same 2D projection is made for the *virtual* tutor policy and is visible in the figure 6.5.

First, we observe that ALEX did not completely maps the observation space. Only the space observed during the demonstration is mapped, leaving the unvisited subspace empty of *Context Agents*. If we compare the learnt policy (Figure 6.4) to the virtual tutor policy (Figure 6.5), we observe a similar separation of the space. *Context Agents* proposing the

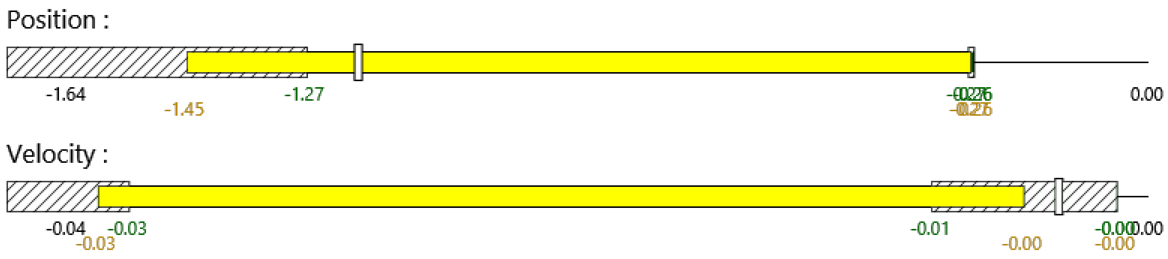

Figure 6.6 — The structure of a particular *Context Agent* involved in the Mountain Car problem. The yellow areas are the area in wich the *Context Agent* is *valid*. The stripped areas are the area in which the Context Agent is *validable*. The white area is the current value of the *Percept Agent*.

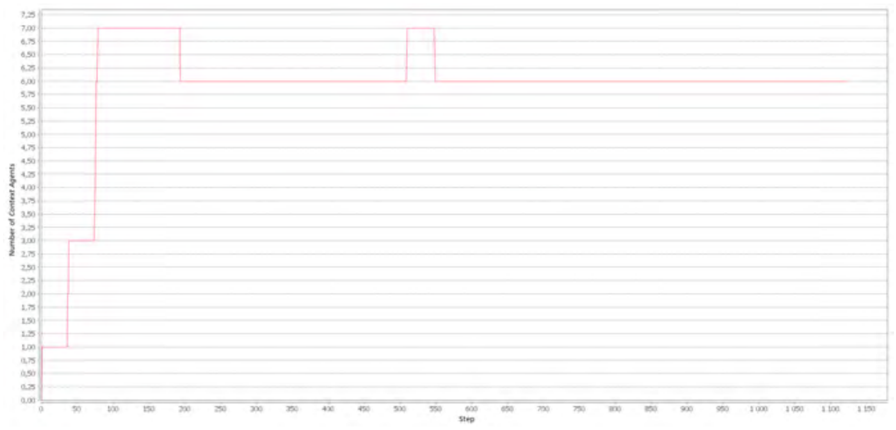

Figure 6.7 — The evolution of the number of the Context Agents during an experiment with a *virtual* tutor.

action −1 occupy the bottom space whereas *Context Agents* proposing the action 1 occupy the top space. This similar separation of the space enables ALEX to perform an effective control of the car.

If we observe the structure of particular *Context Agents* (see the figure 6.6), we find that validity ranges are a lot smaller on the *Velocity* than they are on *Position*. As *Position* is a useless datum (regarding to the *virtual* tutor strategy), *Context Agents* have learnt to be more sensitive to *Velocity* than *Position*.

The figure 6.7 shows the evolution of the number of Context Agents during the episodes of demonstrations. The horizontal axis corresponds to the number of cumulative steps demonstrated since the beginning of the experiment (it does not take into account phases of autonomy where Context Agents do not evolve). The vertical axis corresponds to the number of Context Agents.

The first episode of demonstration leads to the creation of seven Context Agents and reaches six at the end of the demonstration. Indeed, we can see that one *Context Agent* disappears during the experiment. At the end of the experiment, the space is split between six *Context Agents* and those six *Context Agents* are able to collectively control the car in its hill climbing task.

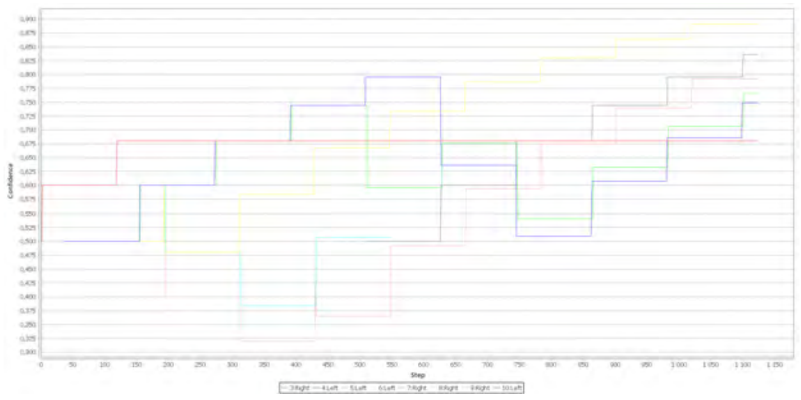

Figure 6.8 — The evolution of the confidence of the Context Agents.

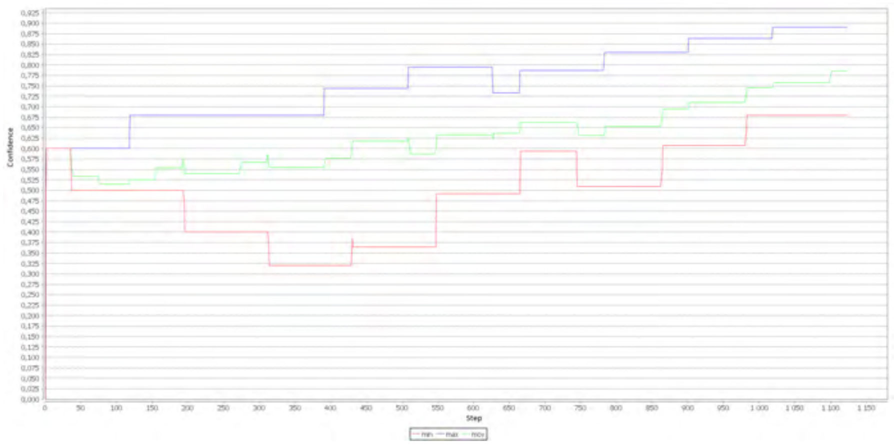

Figure 6.9 — The evolution of the minimum, the maximum and the average confidence of the Context Agents.

The figure 6.8 shows the evolution of the *confidence* of each *Context Agent* during the episodes of the demonstration and the figure 6.9 shows the evolution of the minimum, the maximum and the average confidence of the *Context Agents* during those episodes. The horizontal axis corresponds to the number of cumulative steps demonstrated since the beginning of the experiment. The vertical axis corresponds to the *confidence* value.

While it takes only one episode (117 steps) to be able to control the car, it takes approximatively six episodes for the *Context Agents* to stabilise. This stabilisation is visible around the step 750 where the evolution of the average *confidence* is converging toward 1. This result can be explained by the usage of *Adaptive Value Range Trackers*. The inertial behaviour of the AVRT (see section 5.5.2), by which two consecutive feedbacks increase the value of the variation, enables AVRTs to quickly approximate their ranges. But to fix its boundaries, the AVRT needs to oscillate around the adequate value of the boundary (this behaviour is similar to the binary search). This oscillation, which is not symmetric (see section 5.5.2), enables the boundaries to converge toward the adequate values. The phenomenon is visible in figure 6.10 which shows the evolution of the size of the validity ranges associated with the *Velocity* value for each Context Agent. Thus, the inertial behaviour of the AVRT has an oscillatory effect on its bounds. The perturbation at

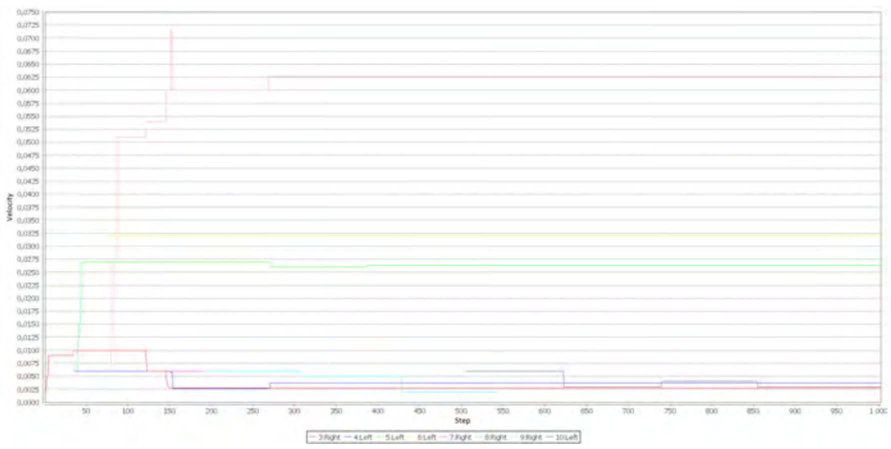

Figure 6.10 — The evolution of the size of the validity range associated to the *Velocity* value for each Context Agent.

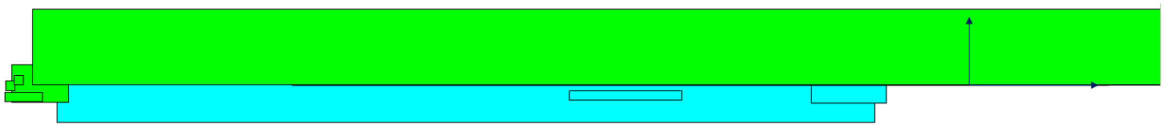

Figure 6.11 — The 2D projection of the policy learnt by ALEX with a human tutor.

boundaries is also influenced by the number of *Context Agents* that are in concurrency at those boundaries. Indeed, the more a *Context Agent* has *neighbours* at its boundaries, the more the Context Agent needs to fix its boundaries in order to avoid Non Cooperative Situations. The concept of *neighbours* in a n-dimensional space is not trivial. Here, we consider that two *Context Agents* are neighbours if there is an overlap in their *context* description that includes at least a boundary of one of the *Context Agent*.

While it may appear to be a disadvantage for the current problem, where the boundaries to model are fixed (because the tutor policy is non dynamic), we postulate that it is an advantage in ambient systems where the boundaries are supposed to be dynamic (due to the openness property and the dynamic nature of the users) and it provides a statistical resilience to noise and manipulation mistakes. We now propose to study the behaviour of ALEX on the same problem but with a human tutor.

6.1.4.2 Learning from a Human Tutor

The figure 6.11 shows the 2D projection of a policy learnt by ALEX after four demonstrations made by a human tutor (here, the author of the thesis). The strategy used by the tutor is similar to the one of the virtual tutor. Thus, we can see that the space is globally split in two parts, depending on the velocity value. An exception to this separation is visible on the left part of the figure where Context Agents proposing the action *right* overlap a Context Agent proposing the action $left$. For those Context Agents, the *Position* value is as important as the *Velocity* value. This is due to the fact that the tutor anticipates the presence of the inelastic wall, and pro-actively changes of action when the car reaches

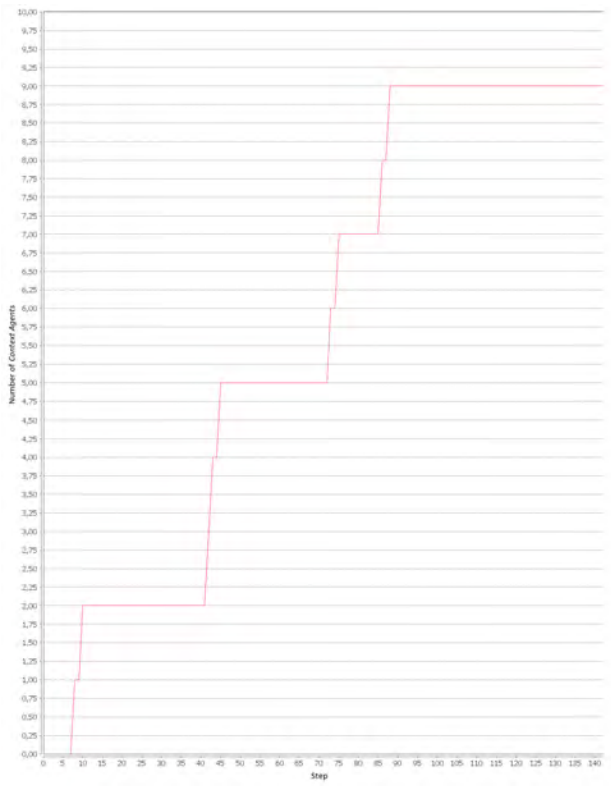

Figure 6.12 — The evolution of the number of Context Agents during an experiment with a *human* tutor.

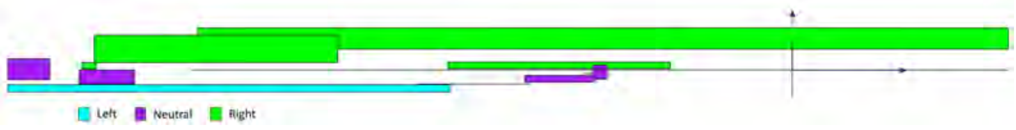

Figure 6.13 — The 2D projection of another policy learnt by ALEX with the same human tutor following a different strategy.

the top of the left hill. The mapping differs from the one learnt with a virtual tutor (figure 6.4), in particular by the number of Context Agents created (see figure 6.12) but globally has the same shape.

The figure 6.13 shows the 2D projection of another policy learnt by ALEX after four other demonstrations made by the same human tutor following a different strategy. Here, the tutor uses the action *Neutral* (squares in purple in the figure). When the *Velocity* variation changes direction (eg when the car stops accelerating), the tutor triggers the neutral action that has the effect of terminating the thrust. The car is then subjected to gravity. When the velocity passes through zero, the tutor resets the push in the opposite direction.

Those two experiments with human demonstrations illustrate that ALEX manages not only to learn from human demonstrations, but also that the learnt strategies differ from those demonstrations, each Tutor having its own policy. Then, ALEX successfully self-adapts its control policy to its Tutor.

6.1.5 Synthesis of the Experiment

The Mountain Car Experiment intends to illustrate ALEX behaviour. In the first part of the experiment, we illustrate how the *Context Agents* self-adapt to collectively build a mapping function to imitate a virtual tutor. The two dimensions of the problem enable to draw a two dimensional projection of *Context Agents* structure and to compare it with the policy to imitate. This graphical representation of the learnt policy shows that the *Context Agents* manage to split the space similarly to the policy to imitate.

In the second part of the experiment, we illustrate ALEX behaviour with a human tutor. Two different strategies were adopted and each strategy leads to a different organization of *Context Agents*. This result is illustrative of ALEX behaviour: what it can learn is dependent of how it experiences it.

The experiment enables to set up comparison pointers with other approaches. Indeed, the Mountain Car problem serves as a toy-problem in the reinforcement learning field [Gatti, 2015]. Those reinforcement learning algorithms try to find an optimal policy to control the car in as few steps as possible. For its part, ALEX intends to learn a policy that mimics the one demonstrated by its tutor.

Knox et al. [Knox et al., 2013] recently proposed an approach to reduce the space of research using Learning from Demonstration. The authors proposed an algorithm named *TAMER* which creates a predictive model of human reinforcement feedback and uses this model to increase a reinforcement learning algorithm. Demonstration in TAMER consists in a succession of quantitative feedbacks (for example : -1, 0, +5) given by a tutor which is only an observer of the system. The experiment they performed on the mountain car problem compared *TAMER* to the traditional Sarsa [Sutton and Barto, 1999] algorithm. They show that TAMER reduces the time needed to arrive at a "good" policy while needing more time to find optimality. However, each TAMER agents needs to be shaped for three runs of twenty episodes.

Celemin et al. [Celemin and Ruiz-del Solar, 2015] proposes a more recent approach named *COACH* to learn continuous actions from corrective advices communicated by humans. *COACH* lets the trainer to shape the policy of an agent through occasional feedbacks. Similarly to *TAMER*, they use those feedbacks to construct a model of the human feedbacks which is then exploited by a policy supervised learner. The performance of *COACH* is compared with the one of TAMER on the Cart-Pole problem (another well-known toy problem) and they show that they significantly reduced the number of episodes to converge to a good policy. However, they suffer from the same limitation than *TAMER*.

Our own experiment has shown that only one demonstration is required for ALEX to learn a good control policy. However, our approach imposes the presence of a tutor (human or virtual) which is an actor of the system which exhibits a policy that is supposed to be the optimal policy satisfying its needs. Whereas the approaches mentioned previously propose to infer a model of user satisfaction from a succession of quantitative feedbacks, our own approach proposes to directly use the control action of the tutor as a feedback. This can be either an advantage or a disadvantage depending on the target system. Indeed, using user action requires that the system is controllable by the tutor, which is not trivial in the case of a distributed system (this problem is partly addressed with the Incremental Design Experiment and is discussed with the limits and perspectives in conclusion).

6.2 Teach Robots Yourselves Experiment

Objectives:

- ▷ Illustrate the Extreme Sensitive Robotic paradigm.
- ▷ Show ALEX capacity to learn in real-time to control a two-wheeled rover.
- ▷ Show ALEX capacity to deal with unseen situations.

Figure 6.14 — Objectives of the Teach Robots Yourselves Experiment

The Teach Robots Yourselves is an experiment of imitation learning that we designed to illustrate ALEX capacity to learn in real time from demonstrations performed by a human Tutor.

6.2.1 General Description

A two-wheeled rover with no sensor on it is immersed in a $2mx2m$ arena composed of a blue area and a green block. An intelligent camera located perpendicularly to the arena at $2m$ of its center can analyse pixels to capture the position of the center of the green and blue colours components relatively to the rover orientation (determined by two red markers on the rover). The figure 6.15 shows an illustration of the experiment.

A human Tutor can perform a direct control over the rover at any time through a 2-joysticks gamepad. Each joystick controls the speed of one wheel (left joystick for left wheel

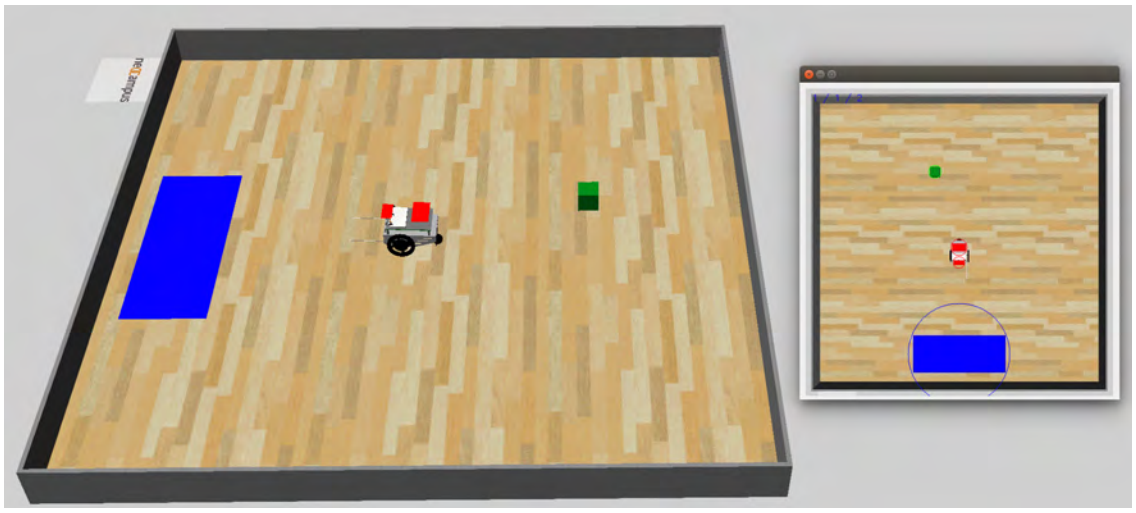

Figure 6.15 — The experiment in the Webots simulator. On the left, the rover inside the arena. On the right, the camera detection.

and reciprocally). The Tutor can perform a range of activities in the arena and one particular activity is a collecting task. This task consists in catching the green block and transporting it to the blue area. In order to do so, the rover is suited up with solid whiskers to allow boxes capture. Whiskers are not movable and there is no sensor on them. They are just a physical part of the robot.

Whenever a green block reaches the blue area, the box is removed from the arena and another one is placed at a random location. The experiment intends to use ALEX to correlate the Tutor activity to the environmental state, and use those correlations to perform the collecting task autonomously.

At each time step, the camera produces four observations about the scene, describing the distance and the angle to the center of the green and blue components relatively to the red component position and orientation on the camera image (the image has a resolution of $1360x1360$). Wheel speed is described by a real value which belongs to $[-100; 100]$ and corresponds to the percentage of the maximum speed to be applied and the sign of the rotation. The set of observations produced by the camera, in addition with the current *speed* of each wheel, forms a vector describing the current situation (figure 6.16).

Observation	**Symbol**	**Domain**
Distance in pixel to the center of the green component	D_g	$[-680; 680]$
Angle between the rover front and the green component in radian	A_g	$[-2\pi, 2\pi]$
Distance in pixel to the center of the blue component	D_b	$[-680; 680]$
Angle between the rover front and the blue component in radian	A_b	$[-2\pi, 2\pi]$
Speed of the left wheel	$Speed_{left}$	$[-100; 100]$
Speed of the right wheel	$Speed_{right}$	$[-100; 100]$

Figure 6.16 — Description of the vector of observations

The problem to handle is to build a mapping policy that associates to any situation described by this vector the adequate *speed* for each wheel.

To induce more dynamics to the vector of observation, the observations of the camera are set available only if an artefact is in front of the rover (basically if the angle to the artefact is include between $\pi/2$ and $-\pi/2$). If the artefact is not in front of the rover, the values of the observations associated to this artefact are arbitrary set to an out of bounds value (which is always the same value). This enables to simulate the appearance and disappearance of observations. At each time step, the vector is then composed by the six observations, but some of these values could be missing.

6.2.2 Motivations and Objectives

The problematic of indoor localisation is an active field of research [Adler et al., 2015]. The complexity to build a map of the environment and to be able to locate yourself on this map is particularly complex. This complexity is increased by the dynamic nature of ambient environments, where obstacles can be added and removed. In this experiment, instead of relying on elements of physical localisation, we propose to only use a monocular camera. This camera only builds information in its own referential by pixel analysis. Thus, that information is not directly information of localisation of the rover and no model of the environment is provided a priori. However, the activity of the rover, through its movements, produces an effect on what is observed by the camera and those effects lead to changes in the data provided by the camera. We hypothesise that correlating the Tutor actions to those data enables to build a mapping policy that is sufficient to control the rover movements. The rover then learns to be sensitive to the variation of its environment (here described by the data from the camera), and the environment itself is influenced by the activity of the rover. This concept is at the core of the *Extreme Sensitive Robotic* paradigm (see section 1.6).

Another motivation of the experiment is the combinatory of the observation space and the randomness of the movement of the artefact which denotes of a certain level of complexity, a key component of ambient systems. Furthermore, due to the random movement of the box, the demonstrations only enable to explore a subspace of the problem. A consequence is that ALEX may be faced to situations which have not been explored during the demonstrations. The experiment is then demonstrative of ALEX capacity to generalise what it has learned to face unseen situations.

In this experiment, two functionalities (the left and right wheels) have to be controlled. Thus, in accordance with the Extreme Sensitive Robotic paradigm, each functionality is considered as an autonomous system, with its own controller. So, aach wheel is controlled by one instance of ALEX. The experiment then enables to test collaboration between two instances of ALEX.

6.2.3 Implementation Details

The experiment is performed in simulation. The experiment has been developed on the *Webots* simulator [Michel, 1998].

The experiment involves five components: the rover to control, the camera, the tutor, and the two ALEX instances. The figure 6.17 summarizes the interactions between the different experimental components. In adequacy with the *Extreme Sensitive Robotic* paradigm, each of those entities is autonomous.

$ALEX_{left}$ and $ALEX_{right}$ are the two ALEX instances respectively responsible of controlling the speed of the left and right wheels of the rover. In each ALEX, one-step of decision occurs every $250ms$.

Every $125ms$, the rover checks if a new speed change request has been received and if so, applies the new speeds. Each time a wheel changes of speed, the rover sends a message $(Speed_{left}, Speed_{right})$ to inform the ALEX instances of a change in the current speed of the wheels.

The exact same implementation of ALEX (made in java) is used to control each wheel. Communication between the entities in simulation is made through a TCP/IP protocol.

The Open Source Computer Vision (OpenCV), an open source computer vision, is used for color detection. The camera provides the data (D_g, A_g, D_b, A_b) every $125ms$.

The Algorithm 6.3 describes in pseudo-code one step of an ALEX instance.

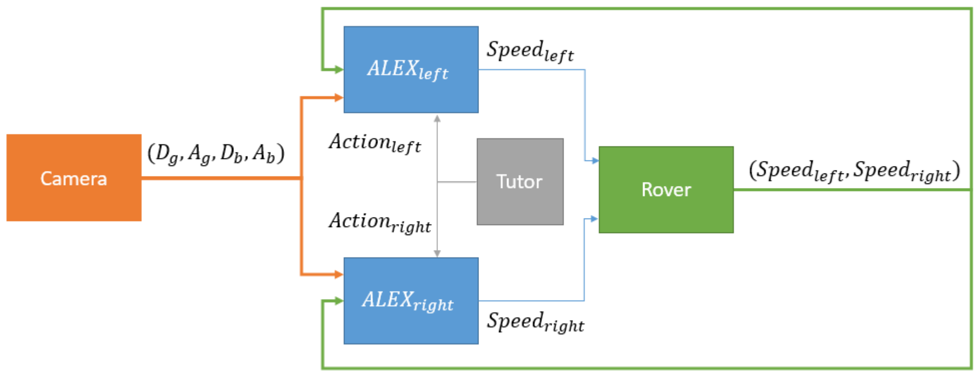

Figure 6.17 — Interaction between the entities involved in the Teach Robot Yourselves experiment.

6.2.4 Analysis

We propose to consider two experiments. In the first one, the Tutor realises a continuous demonstration of the task, and the behaviour of ALEX is analyse a *posteriori* of this demonstration. In the second, the Tutor realises only punctual demonstrations, by showing the adequate behaviour to the system only if the current system behaviour is not satisfying.

6.2.4.1 First Experiment

In a first experiment, we propose to study the behaviour of the ALEX instances with a continuous demonstration. In order to do so, the tutor is asked to control the rover during a certain amount of time and has to perform the collecting task. During this phase, ALEX is

Algorithm 6.3: Pseudo-code of one step of an ALEX instance involved in the *Teach Robot Yourselves Experiment*.

```
Receiving data from the Camera:
$D_g \leftarrow getDataDg()$ ;
$A_g \leftarrow getDataAg()$ ;
$A_b \leftarrow getDataAb()$ ;
$D_b \leftarrow getDataDb()$ ;
Receiving the Tutor action: $Action_{tutor} \leftarrow getTutorAction()$ ;
Updating ALEX:
$ALEX.updateObservations(D_g, A_g, A_b, D_b)$ ;
**if** $Action! = \emptyset$ **then**
    $ALEX.newTutorAction(Action_{tutor})$ ;
**end**
Sending the actions action to apply:
$sendAction(ALEX.getAction())$ ;
```

6

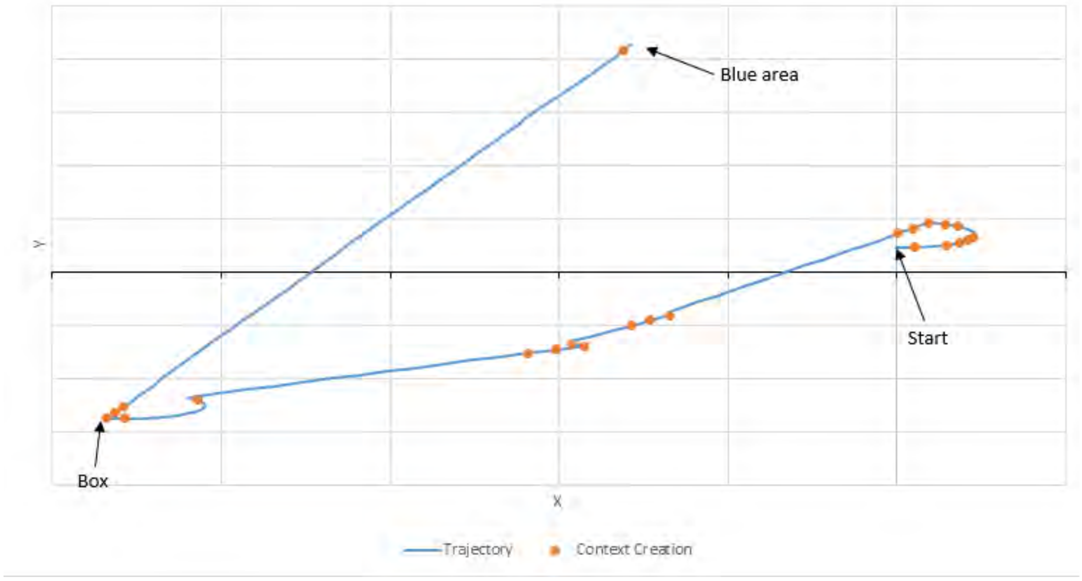

Figure 6.18 — Rover's trajectory and *Context Agents* creation during a demonstration

fed with demonstrations at each time step. In a second phase, the performance of the ALEX instances are observed while no further demonstrations are provided.

This modality of demonstration is similar to the one presented in the previous experiment (see section 6.2). The experiment is split in two phases, the demonstration phase, where the demonstrations are actually performed, and the exploitation phase, where what has been learned is reused in order to ensure the autonomy of the rover.

In the analysis of this experiment, we propose to highlight three aspects of ALEX behaviour: *Context Agent* creation, *Context Agent* evolution and the global (and emergent?) behaviour.

First, we propose to study *Context Agents* creation during a demonstration of a box collection. In such demonstration, the Tutor takes control over the rover and drives it to collect and transport the box to the blue area. Basically, the sequence of actions performed by the Tutor can be summarized as:

- 1 - Turns until the rover is in front of the box (until A_g is near zero).
- 2 - Goes forward until the box is inside the rover whiskers (until D_g is near zero).
- 3 - Turns while the rover is not in front of the blue area and the box is still in the whisker (while A_b is not near zero and D_g is near zero).
- 4 - Goes forward while the box is inside the whiskers and the rover is in front of the blue area (while A_b and D_g are near zero).

The ALEX instances associated to the wheels start empty of any *Context Agents*. We propose to observe, during this demonstration, in which points of the trajectory the *Context Agents* are created. The Figure 6.18 illustrates the creation of *Context Agents* in the two ALEX

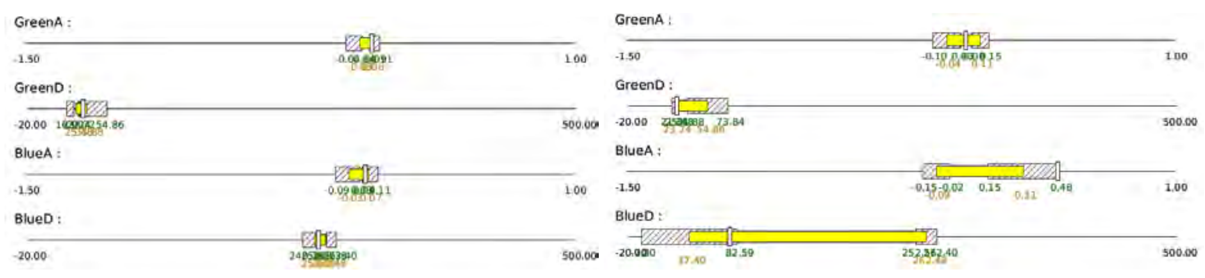

Figure 6.19 — On the top, the validity domain structure at the creation of a *Context agent*. On the bottom, the same *Context agent* at the end of the demonstration.

instances. It shows the trajectory performed by the rover during the first demonstration of the collecting task (in which the first box is collected). The horizontal and vertical axis correspond to the coordinates of the rover in the simulated arena and orange spots to situations where a *Context Agent* has been created.

We can see that the trajectory is not filled with orange spots. Basically, the ALEX instances create new *Context agents* when the Tutor changes the direction of the rover. On contrary, when the Tutor maintains the direction of the rover, no *Context Agents* are created. Each Tutor action is observed and each *Context Agent* determines if the current situation belongs to its own context description. If there is no such *Context Agent*, a new one is created. But if there is an existing *Context Agent*, this means that the situation is already handled by a *valid* or *validable* Context Agent. A consequence for a *validable Context Agent* is that its context description will be adapted to integrate the current situation.

The Figure 6.19 shows the structure of a particular *Context Agent* involved in this experiment, at its creation and at the end of a demonstration. Each line corresponds to the structure of a validity range associated to an observation. The filled range corresponds to the valid range whereas the striped area corresponds to the *validable* range. White boxes correspond to the current value of the signal. We observe that each validity range has its own evolution. This evolution is the result of the self-organization process. More precisely, the validity ranges associated to the perception of the green block (*GreenA* and *GreenD*) are smaller than the one associated to the perception of the blue area (*BlueA* and *BlueD*). This particular *Context Agent* is valid when the block is close to the front of the rover and the rover is in front of the blue area. It is involved in the part of the activity where the rover brings back the block to the blue area.

To observe the capacity of the system to imitate the Tutor performance, the Tutor realized a 5 minutes demonstration in which 12 boxes were collected. The number of collected boxes by the Tutor serves as a metric for performance comparison. The system is then let in autonomy and each 5 minutes the score is computed. During the autonomy phase, the *Context Agents* have to find the most adequate action. The figure 6.20 shows results we obtained. In the worst case, the system performs the task as well as the Tutor does: 12 boxes are collected. However, the number of collected boxes is often better than the Tutor's ones. Two factors influence this result. The first one comes with the randomness of the box movements. In some case, boxes are moved farther away from the blue area and it takes more time to reach and bring back the box. The other one, more interesting, lies in the fact

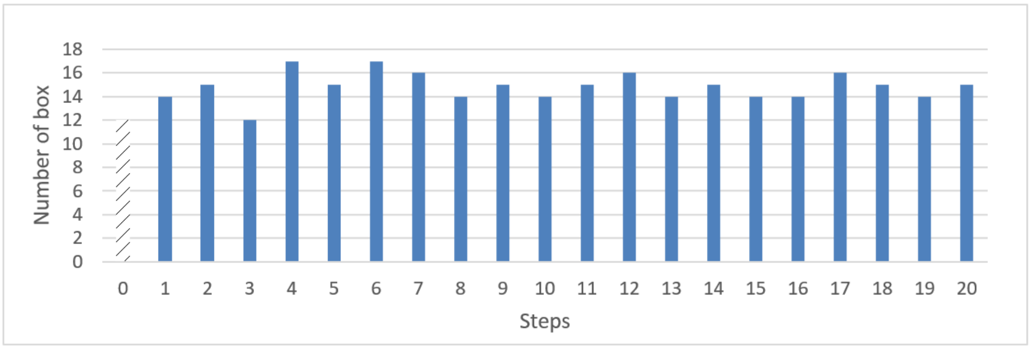

Figure 6.20 — Number of collected boxes each 5 minutes. The step 0 corresponds to the reference score.

that the Tutor needs more time to take a decision than ALEX does. Moreover, the Tutor can contradict itself. This phenomenon is observable in the Figure 6.18. At midpoint between the start position and the box position, we can observe a change in the trajectory. This change is in fact a Tutor tele-operation mistake. The *Context Agents* corresponding to this situation will never be reselected as they correspond to a non-desired action and will then self-destroy. The learnt behaviour is then "filtered" of Tutor mistakes allowing it to perform the task more efficiently.

6.2.4.2 Second Experiment

In a second experiment, the tutor is asked to make a demonstration (which consists in taking temporary the control over the rover) only if the current behaviour of ALEX is not satisfying. Then, a demonstration is made only in situation where the ALEX instances failed to show a behaviour that is satisfying.

The ALEX instances start completely empty of Context Agents (and then, the first action has to be the one of the Tutor), and have to learn over the time to correlate the Tutor activity to the environment. But when the Tutor is not making a demonstration, the ALEX instances must exploit what they have learnt to control the rover, even if the current situation is unmapped by the Context Agent (see the Non Cooperative Situation 5.4.2). Thus, contrary to the previous experiments, the learning and the exploitation of the policy are not separated but performed alternatively in a same experiment.

At each time step, the action applied is either the result of the ALEX instances or the result of the Tutor activity. We propose to compare the number of steps where the Tutor acted, to the number of steps where the situation is only handled by the ALEX instances, for each box collected.

The figure 6.21 shows in orange the number of actions performed by the Tutor for each box collected, and in blue the number of actions which are the result of the ALEX instances during a session where 30 boxes have been collected.

First, if we look at the results obtained after the first box collection, we can see that the ALEX instances has been acting during 137 steps, and the Tutor has been acting during

23. Whereas the ALEX instances start completely empty of Context Agents, and the first box collection is not ended, the ALEX instances manages to perform most of the actions. This is a consequence of the resolution process of the Non Cooperative Situations 1.b (see section 5.4.2), which consists to maintain the last known action in situation where no Context Agents are *valid* or *validable*. Thus, only situations where the action needs to be changed are required to be mapped.

The variation of the number of total steps to collect a box (minimum: 160, maximum: 406, average: 280.7) is a consequence of the randomness of the box movement. However, in general, the number of actions performed by the ALEX instances (minimum: 137, maximum: 404, average: 268.8) exceeds the number of actions performed by the Tutor (minimum: 0, maximum: 57, average: 11.9).

If we look at the general tendency of the Tutor actions, we can observe that a consequence of the ALEX instances activity is that the Tutor actions are minimized over time. The more ALEX manages to build its mapping function, the less the Tutor have to act on it. However, the fact that after 30 boxes the number of Tutor actions falls to zero does not guaranty that all the situations required to control the rover have been mapped by the *Context Agents*. But, as the tutor can act at any time over the rover, a failure in its behaviour can be corrected by a demonstration of the adequate action, this demonstration leading to a reorganization of the *Context Agents* inside the ALEX instances.

6.2.5 Synthesis of the Experiments

With those experiments, we explore ALEX capacity to learn to control a two wheeled-rover from demonstrations performed by a human user. While the experiment may appear to be simpler than a realistic situation, the combinatory of the observation space and the randomness of the movement of the artefact illustrate a certain level of complexity, which is a crucial property of ambient systems.

In the first experiment, we have illustrated that ALEX can be used to imitate (and thus replace) the tutor behaviour with three underlined hypothesis:

- ▷ Hypothesis 1 - There exists an entity in the environment which can perform demonstration of the task.
- ▷ Hypothesis 2 - There are sufficient observations to learn the task.
- ▷ Hypothesis 3 - There have been enough demonstrations to learn the task.

The results show that ALEX manages to imitate the Tutor behaviour but can also perform the task even more effectively than the Tutor itself. This means concretely that the simple process of demonstration allows each multi-agent system associated to all effectors to understand autonomously what the relevant data are in order to mimic collectively what the tutor does, without any central control. The *Context Agents* inside each ALEX instance self-organize to produce collectively a behaviour that is user satisfying.

In the second experiment of the Teach Robot Yourselves experiment, we have illustrated that ALEX can be used even without the hypothesis 3, which means used even if there has not been yet enough demonstration to perform the task. This real-time learning capacity of ALEX enables learning and exploitation to be performed during the same lifecyle. A natural consequence is that tutor actions are minimized over time. The more the ALEX instances manage to learn a satisfying policy, the less the tutor needs to act on the system to demonstrate the adequate behaviour. Whereas the previous experiments show how ALEX can be substituted to the tutor, with this second experiment we show that ALEX can be used to assist a tutor by pro-actively performing actions on its behalf. However, the tutor still keeps its control over the system and can act at any time to change ALEX behaviour. This point is differentiating ALEX from the other learning from demonstration techniques where learning and exploitation are two separated phases. With ALEX, learning and exploitation are performed in a same lifecyle, as the result of a self-organization process.

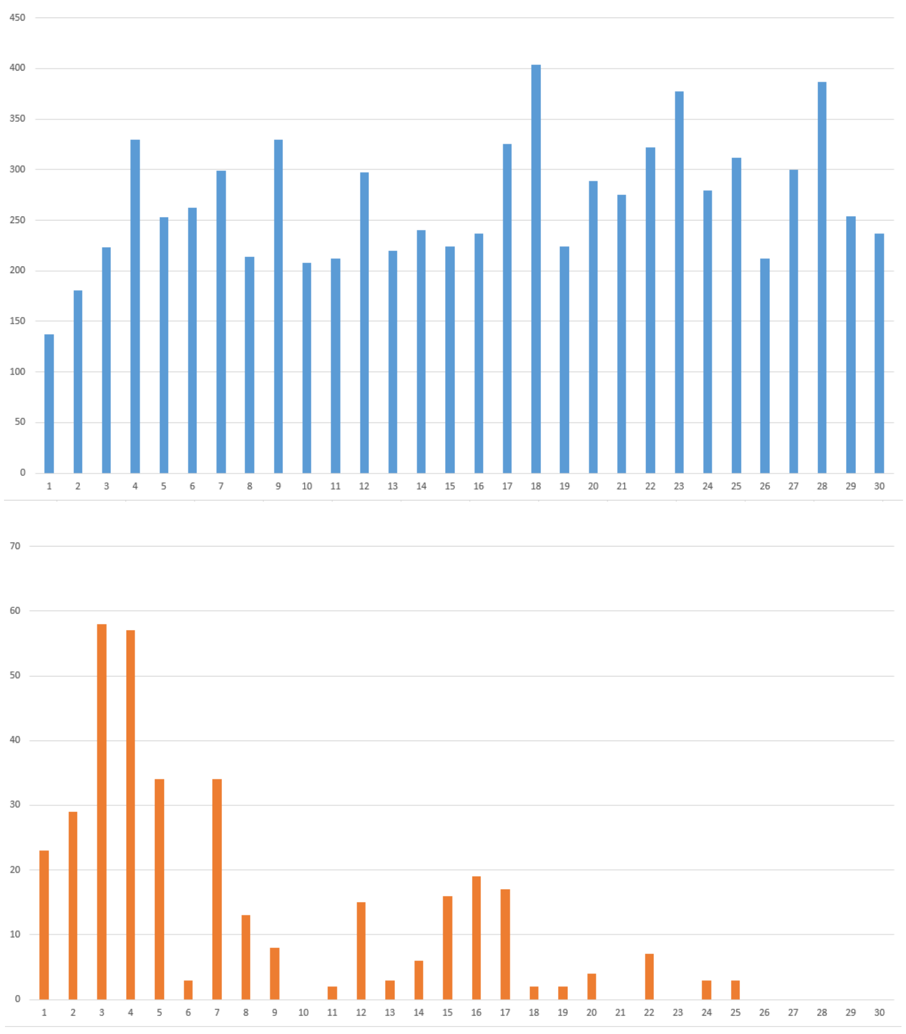

Figure 6.21 — The blue histogram shows the number of steps where the action is handled by the ALEX instances for each collected boxes. The orange histogram shows the number of steps where the Tutor has acted for each collected boxes.

6.3 Incremental Design Experiment

Objectives:

- ▷ Illustrate the usage of ALEX as a development paradigm
- ▷ Show the benefits of using self-organized learning techniques in the design process.
- ▷ Focus on a point of differentiation: the interpretability of the model
- ▷ Highlight perspectives of the approach.

Figure 6.22 — Objectives of the Incremental Design Experiment

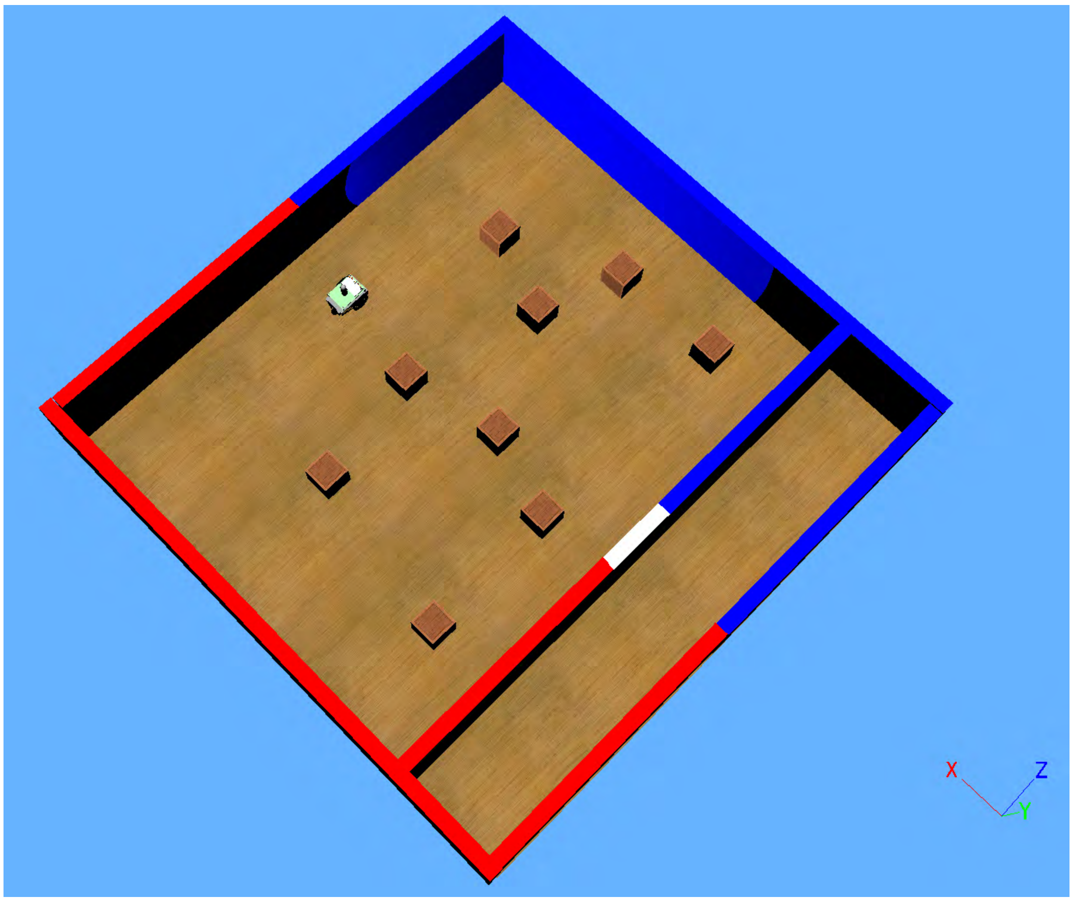

Figure 6.23 — A view of the experiment. The rover evolves in an arena and has to reach the white door.

On the previous experiments, we have illustrated ALEX capacity to learn a control policy from continuous and punctual demonstrations.

With the first experiment, we made the assumptions that (1) there exists an entity in the environment that is able to perform demonstrations of the task, (2) the necessary data to learn the task are available and, (3) there have been enough demonstrations to learn the task.

In the first part of the Teach Robot Yourselves, those assumptions were kept. But in the second part, we relaxed the assumption (3) to illustrate that ALEX can learn and use what it has learned in the same lifecyle. Unlike traditional supervised learning algorithms, learning and exploitation of the learnt policy are not sequential.

On this section, we propose to relax the assumptions (2) and (3) and illustrate how ALEX can be used even if all the necessary data are not present.

The *Extreme Sensitive Robotic* paradigm (see section 1.6) postulates that we now have at our disposal libraries of various hardware and software components. A particular robotic application is then seen as the aggregate of the adequate components to realise the function(s) for which it has been designed. This aggregation still requires an expertise for determining which components have to be put together to perform a particular task.

The challenge to equip the system with the adequate components is far for trivial. An under-equipped system may fail to achieve its task, and a system over-equipped may be more complex to control and induces additional costs.

We propose to study through a case study in which a robotic application is designed, how the combination of the Extreme Sensitive Robotic paradigm and the usage of ALEX, and the analysis of ALEX behaviour can help the designer in finding the adequate composition.

6.3.1 Description of the Case Study

We propose as case study the design of the following experiment:

▷ *Lets consider an experiment where a two-wheeled rover is locked in an unknown room. The rover must find a way to exit the room. The only way out is through a door. The room may (or may not) be populated with obstacles. The number of obstacles and their positions are a priori unknown and may vary over time. The rover then has to navigate through the room to reach the gate and navigate through it. The experiment is a success if the rover is outside the room and the door is closed. On contrary, the experiments fails if the rover is unable to go out of the room.*

In this case study, we propose to adopt the role of an engineer (hereafter referred as the designer of the system) who is asked to design this experiment in simulation. The simulation may be equipped with all the sensors and effectors that the designer thinks to be required. The role of the designer is to propose the adequate system composition (which means the different sensors and effectors and their associated behaviours) to complete the task. The design process is incremental, which means that the designer will propose successive system compositions and test each composition for the task (or the subtask) for which the composition has been thought. In case of a failure in learning the task, the designer will proceed to an analysis of the components behaviour and propose a new system composition. This whole process ends with success in performing the complete task.

The simulation is developed using the *Webots* simulator [Michel, 1998] and the same implementation of ALEX. A rover and an arena equipped with a door are provided to the designer. The designer can add many sensors and effectors to the system but cannot change the physical behaviour of the rover and the door.

6.3.2 The Case Study: Iterative Design Process

6.3.2.1 Initial Architecture

In accordance with the *Extreme Sensitive Robotic* paradigm, the first step consists in identifying which are the functionalities involved in the experiment. Each of those functionalities will then have to be considered as an autonomous system with its own controller. As the experiment is about analysing ALEX behaviour, each functionality is controlled by an ALEX instance.

Designer's thought: *The rover is composed of two wheels, each wheel controlling its own speed value from -100 to 100. Each wheel is associated with an unique ALEX controller. An ALEX controller must build and exploit a mapping function associating to the current context the adequate speed value.*

The figure 6.24 summarizes the initial rover architecture.

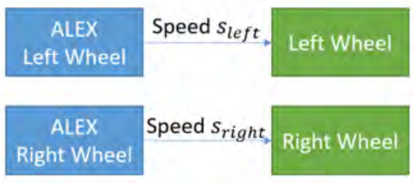

Figure 6.24 — The core of the rover architecture. Each ALEX controller is responsible of the speed of one wheel.

Once the different functionalities have been highlighted, we need to identify the contextual perceptions that are going to be used by the ALEX instances to determine the current context of the system. The challenge is to find the adequate set of perceptions.

ALEX makes no assumption about the nature of the transmitted data and their number. Its ability to learn is based on a self-observation of its context and its tutor activity, and not on any form of *a priori* knowledge about the environment. This enables an ALEX instance to be used in any kind of system composition. The only requirement is that perceptions are differentiable (basically, each perception disposes of its own identifier) and there exists an order relation between the values of the perception.

During the design process, the designer proposes a set of perceptions that he thinks to be appropriate for the task, and validates this composition through a demonstration of the activity. During this demonstration, the ALEX instances self-organize and correlate those perceptions to the Tutor's activity. If those perceptions are sufficiently discriminating to perform the task, the ALEX instances will succeed to build and exploit their mapping function which will enable them to realise their task autonomously. If not, the ALEX instances will fail to act autonomously.

6.3.2.2 First Experiment: Navigation with a Distance Sensor

The designer proposes a first architecture that is going to be experimented.

Designer's thought: *A first task extracted from the analysis of the objectives of the experiment is the ability to navigate through the room. Navigation requires a perception of the different obstacles. Thus, the designer identifies a distance sensor as required to navigate through the room. The distance sensor is thought as an autonomous entity which provides an observation about the environment. It autonomously sends updates to the two ALEX instances. This sensor is set up on the middle of the two wheels and enables the rover to detect the distance to obstacles that are located in front of it.*

This first architecture is shown in figure 6.25.

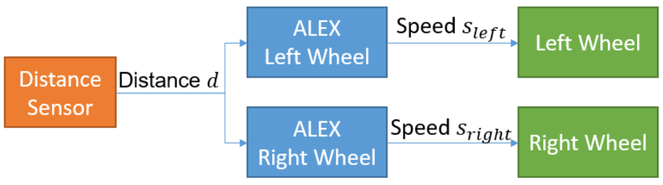

Figure 6.25 — Architecture of the first experiment. Each ALEX receives the distance value and has to associate to this value the adequate speed.

Now that the designer has agreed on a first architecture, this architecture needs to be tested. In order to do so, the global target behaviour is split in four subtasks: (1) navigate in the room, (2) reach the door, (3) open the door, (4) pass the door. A failure in one of these tasks causes the failure of the following tasks.

The designer proceeds to the demonstration of the task (1) and, secondly, observes the system capacity to perform the task autonomously. Unfortunately, the rover fails in its navigation task. We propose to analyse the *Context Agents* inside each ALEX instance to understand the reasons of this failure. Thanks to this analyse, we found out that using only a distance sensor value leads to ambiguities in the demonstration. This phenomenon is visible in figure 6.26.

The figure 6.26 shows the structure of two *Context Agents* after the demonstration of the task (1). The yellow areas correspond to the values of the distance sensor where the *Context Agent* is *valid*. The green area to the values where the *Context Agent* is *validable*. Those two *Context Agents* are extracted from the ALEX instance controlling the right wheel. They propose different actions. The first one proposes to go at a speed of 100 whereas the second one proposes to go at a speed of 0. If we observe the two validity ranges of the *Context Agents*, we observe an overlap. This overlap means that the two *Context Agents* propose

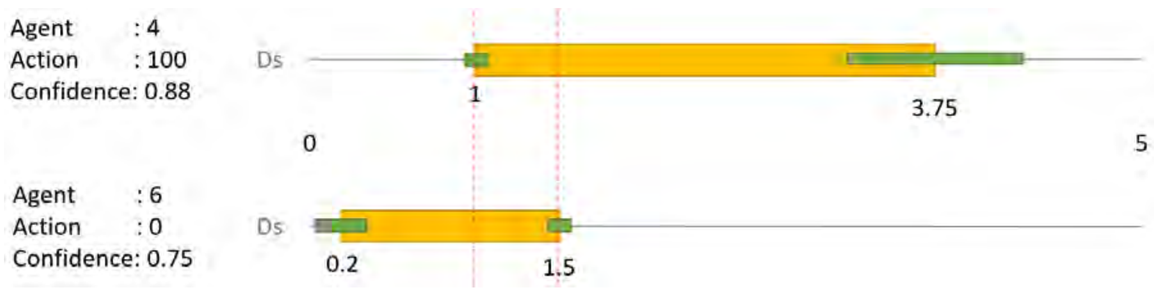

Figure 6.26 — A comparison of two *Context Agents* extracted from the first experiment. The two *Context Agents* propose a different action under the same context leading to ambiguity.

their actions in similar situation, leading to ambiguity. The reason is that using only one distance sensor is not well enough to discriminate all situations. For example, when the rover is at a particular distance, the distance can either express that the rover is approaching an obstacle or moving away.

In this first experiment, there is not enough data to build the necessary correlations. It results some ambiguities in the demonstration, where different actions where shown in similar contexts. These ambiguities are visible by the fact that many *Context Agents* proposing different actions are *valid* under the same situations. The observation of the *Context Agents* activity enables the manual detection of those situations. From this analysis, the designer disposes of some clues to improve the system's activity.

	(1) Navigation	**(2) Reach the door**	**(3) Open the door**	**(4) Pass the door**
1	Failure	-	-	-

Figure 6.27 — Synthesis of the results after the first experiment.

6.3.2.3 Second Experiment: Adding Wheel Speed as an Observation

From the analysis of the first experiment, the designer now proposes and tests a modification of the architecture.

Designer's thought: *The first experiment leads to ambiguous situations. Those situations do not allow a differentiation between situations where the rover is approaching an obstacle and situations where the rover is moving away from it. To disambiguate those situations, the designer proposes to use the current speed value of both wheels as an input to the ALEX instances.*

The revised architecture is visible in figure 6.28. Here, the choice of the designer is not to add a new sensor but to use the current states of the different effectors as a contextual information. Thus, at each time step, the two wheels autonomously send updates about their current speed to the two ALEX instances. The two instances now dispose of the current value of the distance sensor d, and the two speed values S_{left} and S_{right}.

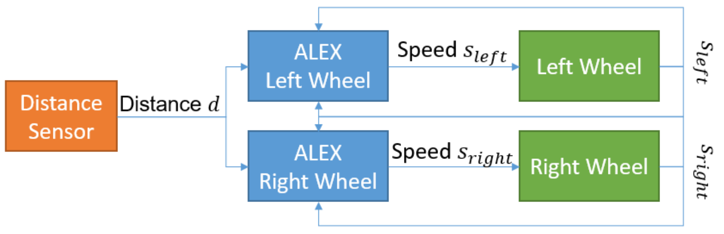

Figure 6.28 — The revised architecture of the first experiment used in the second experiment. Now, each ALEX instance receives the distance and the current speed of both wheels.

With this new architecture, the designer proceeds to a new demonstration of the task (1) and, once again, observes the behaviour of ALEX instances acting autonomously.

The rover succeeds to navigate through the room, avoiding the obstacles. The task (1) is now a success. Consequently, the designer now proceeds to the demonstration of the combined tasks (1) and (2). However, as no information allows to differentiate the door from a wall or an obstacle, the rover fails to complete the task (2).

	(1) Navigation	(2) Reach the door	(3) Open the door	(4) Pass the door
1	Failure	-	-	-
2	Success	Failure	-	-

Figure 6.29 — Synthesis of the results after the second experiment.

6.3.2.4 Third Experiment: Adding a Camera

Designer's thought: *As the rover needs to differentiate the walls and the door, the designer proposes to add a Camera on the rover to recognize characteristics of the detected objects. Using a detection algorithm, the camera can provide visual information about the environment of the rover. As the walls and the door have different colors (see figure 6.23), the camera identifies the coordinate (x,y) of the center of each of the three colors (blue, white and red).*

A camera is added to the simulation to provide new data to the ALEX instances. Each ALEX instance now receives, in complement with the data of the second experiment, the coordinates (x,y) of the center of each color component. If no artefact of one color is detected, the coordinates provided are (-1,-1) (an out-of-bounds value). The addition of the camera does not involve any modification on the ALEX instances.

The new architecture is visible in figure 6.30.

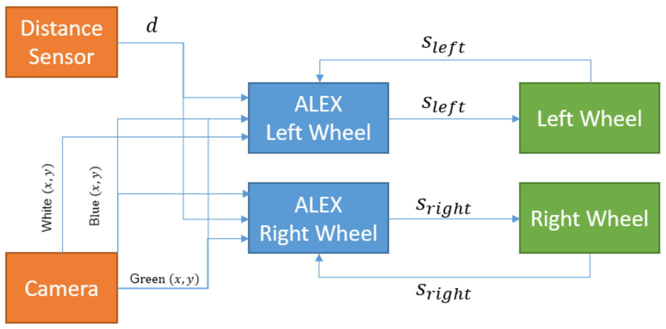

Figure 6.30 — The third experiment architecture. A camera is added providing three new couples of values (x,y) for each detected color.

The Tutor then performs another demonstration of the combined tasks (1) and (2).

The rover successfully manages to navigate inside the room and reaches the door. By observing the structure of the *Context Agents* involved in this experiment, the designer found that the agents involved in the task (2) have learnt to be less sensible to the *blue* and *red* coordinates. One example of those agents is visible in figure 6.31. The validity ranges associated to the signals *WhiteX* and *WhiteY* are smaller than the one associated to the signals *BlueX* and *BlueY*, and *RedX* and *RedY*. As the activity only involves identifying the white door, the other data are unrelated. Our designer might exploit this information to remove unused data from the system.

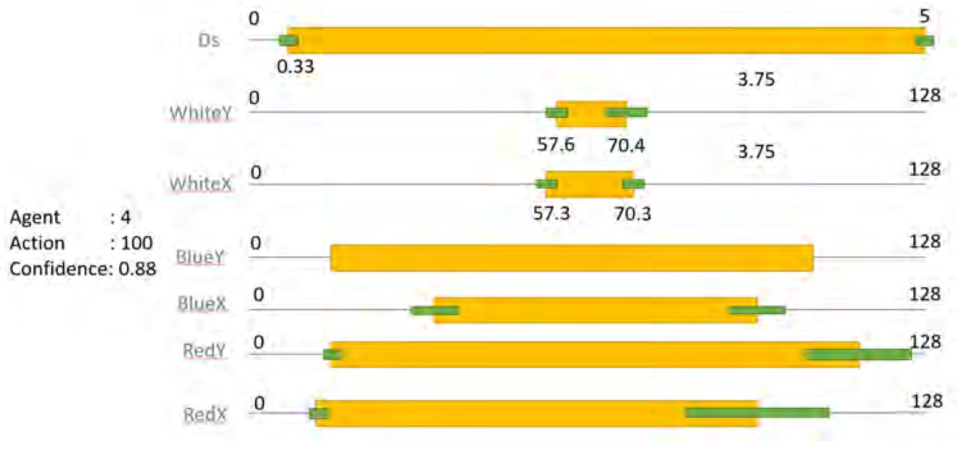

Figure 6.31 — The structure of a particular *Context Agent* involved in the third experiment. This *Context Agent* is a lot more sensitive to the White (x,y) value than the other.

As the tasks (1) and (2) are now successfully learnt, the designer now proceeds to a demonstration of the combined tasks (1),(2) and (3).

The observation of the behaviour of the ALEX instances shows that the rover fails to complete the task (3). While it managed to reach the door, the rover failed to open it as its engines were not powerful enough to push the door.

	(1) Navigation	(2) Reach the door	(3) Open the door	(4) Pass the door
1	Failure	-	-	-
2	Success	Failure	-	-
3	Success	Success	Failure	-

Figure 6.32 — Synthesis of the results after the third experiment.

6.3.2.5 Final Experiment: Adding a Motor to the Door

Designer's thought: *In the previous experiment, the rover fails to open the door as its engines were not powerful enough. There are two ways to solve this situation. One consists in increasing the power of the rover engines. The other consists in adding a motor to the door to enable the door to self-open. Adding a motor to the door is identified as less costly than changing the rover engines. Then, a new effector with its associated ALEX controller is added to the simulation. This new ALEX instance receives the same data than the previous ones.*

For the last experiment, the door is equipped with a motor. An ALEX instance is associated to the door and must learn when it has to be open and when it has to be closed. This new ALEX instance receives the same data than the two ALEX instances controlling the wheels. Adding this new effector does not involve any action on the pre-existing devices and previously learnt *Context Agents* may be kept.

The new architecture is visible in figure 6.33.

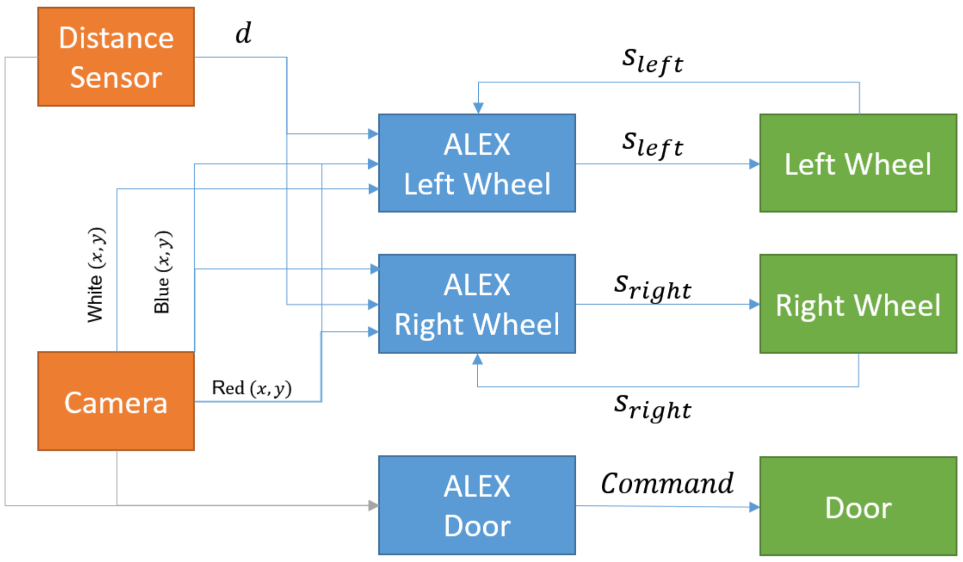

Figure 6.33 — The architecture of the last experiment. The door is now controlled by an ALEX instance and receives the same information than the other ALEX.

The designer performs a final demonstration of the combined tasks (1), (2), (3) and (4), demonstrating to each component the desired behaviour. The designer now controls the two wheels and the opening of the door.

Finally, the rover manages to exit the room by passing through the door. The door correlated the action of opening to a low value of the distance sensor signal with the coordinates *WhiteX* and *WhiteY* near the center of the screen. The rover and the door manage to collaborate without direct communication. As long as they share perceptions, they have enough information to coordinate their activities.

	(1) Navigation	**(2) Reach the door**	**(3) Open the door**	**(4) Pass the door**
1	Failure	-	-	-
2	Success	Failure	-	-
3	Success	Success	Failure	-
4	Success	Success	Success	Success

Figure 6.34 — Synthesis of the results after the last experiment.

6.3.3 Synthesis of the Experiments

In this experiment, we illustrate the benefits of using ALEX as an accelerator during the design process. In order to do so, we took the point of view of a designer who was assigned the task of designing a robotic system.

Through this case study, we show how the combination of the Extreme Sensitive Robotic paradigm and ALEX can help and unload the designer in its task of design. With this approach, the design of a system is made incrementally and bottom-up. Thus, the designer can focus on the functionality provided by the system delegating the control to the different instances of ALEX. Different system compositions can be tested and evaluated without requiring any strong modification of the pre-existing ALEX instances.

In the synthesis of the Mountain Car experiment, we expressed that a limitation of our approach is that the process of demonstration requires that the system is controllable by a tutor, which is not trivial in distributed environment composed of many devices. The incremental design process offers an interesting solution to this problem by enabling each component to be added incrementally. Then, the demonstration can focus on sub-part of the system facilitating the control task by demonstrating only the behaviour of one component at a time (another solution to this limitation is discussed in conclusion).

This experiment enables to highlight a differentiating point of the Context-Learning approach, the interpretability of the learnt model. Indeed, the structure of the validity ranges by which the *context* is modelled is easily readable. As ALEX learning capacity is based on self-observation and self-organization of *Context Agents*, analysing this organization and their behaviours provides some clues on system's adequacy. In this experiment, we show that this analysis enables the designer to point out situations of ambiguity where a required data is missing and situations where data are useless for the task to learn.

However, this analysis is hand-made by the designer, and thus requires a certain form of expertise to understand and extract useful information from the *Context Agent* self-organization. Nevertheless, we think that this analysis can be made automatically by the *Context Agents* themselves and we are currently working on the extension of the self-observation capacities of both the Context Agents and the Percept Agents, notably to detect missing or useless data (this point is discussed in perspectives).

6.4 Discussion on ALEX Usage in the Context of Industry 4.0

Objectives:

- ▻ Discuss of ALEX usage in the context of *Industry 4.0*.
- ▻ Highlight ALEX requirements.
- ▻ Set the framework of using ALEX with a collaborative robotic arm.

Figure 6.35 — Objectives of the Collaborative Robotic Arm Thought Experiment

Before making a synthesis of the experiments, we propose to synthesize the behaviour of ALEX and discuss of its applicability in the context of *Industries 4.0*. As an illustration, we propose to think about the usage of ALEX with a collaborative robotic arm and a human operator, with a focus on the how and the why of using ALEX. First, we remind the requirements imposed by ALEX usage on the nature of the environment. Then, we address the questions of the nature of the actions, the nature of the observations and the Tutor's role.

6.4.1 Requirements for Using ALEX

ALEX is designed to be coupled with an environment in which it autonomously performs actions. When ALEX performs an action, it changes the current state of an effector. By switching to its new state, the effector produces an effect on the environment. ALEX observes this effect through the different *observations* which are sent to it and will decide either to perform a new action or to maintain the current one. Thus, the actions of ALEX modify the environment and changes in the environment influence ALEX decisions (Figure 6.36).

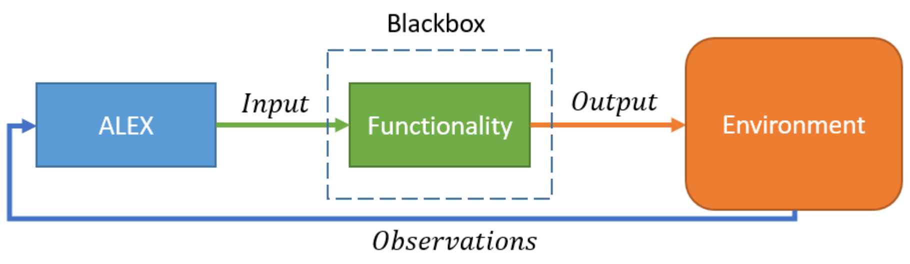

Figure 6.36 — ALEX is coupled with its environment from which it receives observations and on which it acts.

This coupling is fundamental for ALEX, considering that its generalisation capacity is based on the observation of the environment, and imposes requirements on ALEX usage. Indeed, when a tutor performs an action on ALEX, the agents inside ALEX observe the variations that occur in the environment throughout the application of the action, and those variations enable each agent to generate its own feedbacks. In the particular case of *Context Agents*, this observation enables the validity ranges of the *Context Agents* to self-adapt and through successive feedbacks to generalize (see section 5.5.2). In order for those variations to be observable, the action requires to have a certain time continuity. Indeed, an action that is maintained only during one decision cycle would lead to the creation of a *Context Agent* with validity ranges initialised around the current situation, since this initialisation is based on the observation of the variations in the environment that occur since the last decision cycle (see section 5.5.5).

The usage of ALEX then requires that observations and the actions that are sent to ALEX are sequential. Using ALEX without this requirement, for example by randomly providing sequences of $(observations, action)$ without taking their sequential aspect into account, is clearly not what ALEX has been designed for and its ability to learn a control policy in such environment is not guaranteed.

6.4.2 ALEX and the Collaborative Robotic Arm : Proposals and Open Questions

Now that we have highlighted some requirements, how can ALEX be used to enable collaboration between a robotic arm and a human operator? Three points have to be determined in order to use ALEX: what is the system to control and what are the **actions**, what are the **observations** on the environment, and who is the **tutor** and how the demonstrations are performed.

6.4.2.1 What Are the Actions?

A first step in using ALEX is identifying the functionalities to control and what are the **actions** to control them. In ALEX, actions are discrete. The system (external to ALEX) which recovers ALEX actions, associates to any discrete action a command to send to the functionality to control. This implies that the functionality controlled by ALEX has to be seen like an automata, where every action is a transition to a new state. Due to the discrete nature of the actions in ALEX, *Context Agents* do not modify their action, and thus ALEX in its current form is not able to perform regression. For example, in the case of the control of an increment (for example, the temperature control), if the demonstration consists in three successive incrementations of $+1$, ALEX would not learn that the action to perform is $+3$ but ALEX will learn that the action $+1$ has to be performed and maintained it in a certain context.

So, how can we control a robotic arm with ALEX? Here, we propose three modalities, depending on the level of control you want to have.

The first one is to consider the robotic arm as a set of primitive movements, such has *grasp*, *open hand*, *close two fingers*, etc. To each primitive corresponds a sequence of pre-programmed actions. With this level of control, the learning is more about when each action must be performed than on the what and how to do.

The second modality is to control the movement of the arm in its own referential by successive translations of the position of the hand. Actions such as *left*, *right*, *higher*, or *lower* enable to move the hand position, while the kinematic of the movement (the way the arm will move to reach the new position) is performed by the robot itself, with *inverse kinematic*. With this modality, the end-position of the hand is the one being controlled.

At last, the modality which offers the highest level of control on the robotic arm is to see it through the scope of the *Extreme Sensitive Robotics* as an aggregate of different motors, and to control each motor independently. Each motor can thus possess a set of primitive actions, such has *turn clockwise* or *stop* which ALEX has to learn to control to perform an adequate collective behaviour. The result of the control of the different motors then enables to completely control the robotic arm movement.

This classification of modalities of control is non exhaustive and non exclusive. New modalities can be found or modalities can be combined, depending on what is the task to perform and what we want to learn. Due to the discrete nature of its action, ALEX is completely agnostic of this modality, enabling its usage with any kind of system which can be controlled through state automatons. However, the modality may affect the observations required to perform the control.

6.4.2.2 What to Observe?

The second step in using ALEX is to identify the **observations** that may be useful to extract the context in which the actions are performed. Indeed, *Context Agents* inside ALEX learn the context in which an action is applied in order to pro-actively propose this action in similar situations. They learn this context from a self-observation of the environment, characterized by the observations that are sent to ALEX. Therefore, the observations depend on what the actions are, and what are the variations in the environment in which we want ALEX to react.

ALEX is designed to learn a control policy in a dynamic environment through demonstrations performed by a human tutor. An underlined hypothesis is that the actions that are demonstrated to ALEX by the Tutor are contextual. It postulates that if the tutor changes of action, it is because a part of the environment has changed. Through the self-organization process, the *Context Agents* isolate what is the part of the environment that leads to the production of an action and they use this information to pro-actively propose their actions.

Putting the adequate observations to evaluate and control a task is not trivial as it is highly dependant of which is the system to control and what is the task to perform. The different experiments presented in this thesis make the assumption that their are enough observations to learn the demonstrated task. With the Incremental Design Experiment, we have shown that the absence of required observation may be detected from the analysis of the activity of *Context Agents* and highlighted that the automatic detection of missing data is in perspectives. However, the problematic of what makes an observation a good observation to learn a task is clearly an open question. The problematic of what are the good observations in order to learn a high variety of task is even a more important question, which could be the work of another (or many others) thesis.

By its design, ALEX reveals some features on the observations that seems to be the most appropriate for it:

- ▷ The coupling between ALEX and its environment involves that the observations are put on the task to learn and the consequences of the different actions that are performed.
- ▷ The observation are continuous (or at least there exists an order relation between the different values of an observation).
- ▷ Those observations are provided sequentially.

Our initial problem of a collaborative robotic arm can then be seen from two angles, depending on what we want to learn.

The first one is to design a robotic arm that self-adapts to its environment. Thus, the observations are set on the activity of the robotic arm and the different elements composing its environment. For example, we can put a camera which observes the current position of the arm relatively to the position of some artefacts. The task to learn may be how to interact with the different artefacts. A tutor may then teach to the arm to drag and drop some artefacts in some parts of the environment (this proposal shares some similitude with the Teach Robot Yourselves Experiment). The observations in such application take the form of metrics of distances and positions. The activity of the robotic arm directly affects the observations, and reciprocally, the observations affects the robotic arm decisions.

A second approach consists to see the operator as the system to be controlled and the robotic arm as a service provider. In such context, the observations are put on the operator activity and the action performed by the robotic arm intends to modify the current activity of the operator. Such application involves observations that enable the characterization of the activity of the operator in order to associate to a particular situation the adequate service. The task to learn is here the identification of situations and which are the adequate services to offer in those situations.

Of course, the two approaches can be combined in order to design a robotic arm with the ability to self-adapt to both its environment and its operator. ALEX has no *a priori* knowledge on the task to perform, and only seeks to self-adapt by establishing correlations between the actions performed by its Tutor and its perception of the environment. That is why the problematic of the nature of actions and observations discussed in this section were not directly addressed during the design of ALEX and the experiments. However, like any control system, the capacity of ALEX to control a system is highly dependant of what are the observations on this system and which level of control is possible on this system.

6.4.2.3 Who is the Tutor?

A final point to address, which is not restrictive to ALEX, is the role of **Tutor** in such context. The usage of the *Learning From Demonstrations* paradigm involves that there exists an entity that is able to perform a demonstration of the task to achieve. The process of demonstration then requires an interface to actually perform the demonstration. From the easiness of this interface depends the requirement of expertise for end-users. However, this interface is also dependant of the system to control, the task to perform and the users which will use it. This problematic is a topic by itself. Once again, ALEX abstracts the notion of medium for the demonstration process. Indeed, the action performed by the Tutor is sent by its environment. But, in a more global view, this medium has to be taken into account relatively to the system to control, the task to learn, and the end-users. There exist many methods to perform a demonstration, from tele-operation to shadowing (see section 3.2.3.1), each method having its pro and cons, and ALEX could be used with any of those methods.

However, there is a requirement that the tutoring process does not affect the observation of the system. In the previous section, we have identified two approaches: one consisting in a robotic arm that adapts to its environment and the second consisting in providing service to its operator. With the first approach, the objective of using ALEX is to replace the operator by an autonomous system. As a matter of fact, the operator can act as the Tutor of ALEX since its activity (the control of the robotic arm) is not the one observed by ALEX. The operator is external the ALEX environment. But, with the second approach, the operator is the one being observed by ALEX, the operator is a part of ALEX environment. Thus, this operator cannot be the Tutor of the system, since its activity may be disrupted by the modality of demonstration. A third person is then required to perform the demonstration.

6.4.3 Conclusion

In this section, we questioned the usage of ALEX in the context of the *Industry 4.0* by studying the applicability of ALEX in the design of a collaborative robotic arm. With this discussion, we intended to highlight the requirements of using ALEX. This discussion also enabled to point out some strengths, limitations, perspectives and open questions of our approach by contextualizing it in an application domain. The next section proposes to synthesize and discuss ALEX properties and its limitations in details.

6.5 General Synthesis

The different experiments that we realised now enable to highlight ALEX properties and to discuss of its limitations and perspectives. In the section **??**, we seek for an approach with the four following criteria: task independancy, user-centered approach, on-line learning and openness. First, we propose to analyse ALEX for each of those criteria.

6.5.1 Compliance with the Criteria

6.5.1.1 Task Independancy Criterion

This criterion is related to the variety of tasks that a system is able to perform without requiring modification. To evaluate this property in ALEX, we need to look at what are the hypothesis of the observation space and the action space which draw the requirement for ALEX usage.

Actions in ALEX are discrete. The underlined hypothesis is that the system to control disposes of various states and a particular action corresponds to a transition toward a particular state. The role of ALEX is to pro-actively change the current state of the system in accordance with its Tutor by sending the adequate action. ALEX dynamically learns those actions from the imitation of the Tutor's action. The action is then interpreted by the controlled system which applies the adequate transition. This enables ALEX to control a high variety of systems which respect this hypothesis. While discrete actions enable ALEX to be used on many devices without requiring *a priori* knowledge on it, it is also a limitation, preventing ALEX usage to systems with continuous actions. Using Context-Learning with continuous action is a perspective of this work.

Observations in ALEX are continuous and sequential. This is in complete accordance with the characteristics of the environment of ambient systems and respects the Physically Grounding Hypothesis expressed by [Brooks, 1990]. Two choices of design are based on this hypothesis. The first one is the choice to model the *context* with validity ranges. The adjustment of this context description through bound management involves that there exists an order relation between the different observation values. The second comes with the usage of Adaptive Value Ranges Trackers, which uses this order relation to generalize. Integrating discrete values to the context description is possible and is a perspective of the approach. However, those discrete values have to possess an order relation in order for the mechanism of bounds adjustment to work.

The results that we obtained tend to show that these assumptions are consistent with the scope of ambient robotics, but they can be limiting for environments with different assumptions. In regard with those assumptions, ALEX offers the possibility to learn a large variety of tasks and then respects the task independancy criterion.

6.5.1.2 User-Centered Criterion

The choice to use *Learning From Demonstration* and to learn from the observation of the Tutor's activity makes of ALEX a user-centered approach. As ALEX possesses no internal representation or model of the Tutor, it can be used with any kind of user, and even with multi-users (as long as the demonstration made by the users are not in contradiction). The experiments that we performed on the Mountain Car problem has illustrated that ALEX manages to learn different policies and then can self-adapt to different kinds of Tutor.

6.5.1.3 On-Line Criterion

The action made by the Tutor acts as the motor of ALEX self-organization. At each time step, ALEX observes if the Tutor has acted. The activity of the Tutor enables each agent to generate their own feedbacks and to resolve the different Non Cooperative Situations. As this self-observation is made at each time step, we can say that ALEX has an On-Line learning capacity. Each time the Tutor acts, its action is observed and the the *Context-Agents* self-adapt their organization in consequence. However, in the absence of Tutor's activity, the organization of the agents inside ALEX is not modified, their decision process only leads to the selection of the action to perform.

6.5.1.4 Openness

The openness with ALEX has to be considered at three levels: the action space, the observation space, and with multiple ALEX instances at the macro level.

The actions in ALEX are dynamically learned from the observation of the Tutor's activity. Those actions are associated to *Context Agents* which can appear and disappear. A consequence is that the number of actions mapped with ALEX may vary over time. Thus, the openness property is respected for the action space as action can be dynamically added or removed.

Each observations in ALEX is linked to a unique *Percept Agents*. Each time ALEX receives an update value from a new observation, a new *Percept Agent* is created. A consequence is that the *Context Agents* which will be created after the appearance of the Percept Agent will integrate this new value to their organization. Appearance of new observations is then handled by ALEX. However, ALEX does not include any process to detect that a data is useless or that a data has stopped sending updates. In the experiments we performed, the disappearance of a data was simulated by a particular value, describing that for this particular value the observation was not present. In perspective, ALEX must possess the ability to dynamically detect the absence of update or that a data is useless.

At the macro level, the experiments have shown that the locality of the decision of each ALEX instance enables to easily add or remove other ALEX instances. The requirement for two ALEX instances to collaborate is either that one is the observation of the other and reciprocally, or that they partly make the observation of the environment. However, the macro-level suffers from the same limitations than those of the observation space.

A limitation to the openness in ALEX is that the self-organization process requires that the Tutor performs a demonstration. A perspective is to enable self-organization even in the absence of Tutor's activity.

From the analysis of the result, we can express that ALEX is compliant with the criteria expressed during the analysis of the state of the art. However, ALEX possesses other properties that are interesting to discuss.

6.5.2 Discussions

In this section, we propose to discuss of three aspects of ALEX: its low instantiation cost and its parametrization, its scalability and the notion of noise. This discussion enables to point out some requirements to use ALEX.

6.5.2.1 Instantiation Cost and Parametrization

Our will to make the fewest assumption as possible has another consequence in the instantiation cost of ALEX. Indeed, all the experiments that we shown in this thesis are obtained with the same identical implementation of ALEX (without any tuning of parameter). Those results tend to illustrate ALEX capacity to learn to control an *a priori* unknown system. As the components of the control policy (the actions and the observation set), and the control policy itself are dynamically learned during the demonstrations, ALEX is easy to instantiate. However, someone with some knowledge about the environment could seek to tune ALEX in order to increase its convergence speed or refine its policy. Two parameters are actually empirically fixed: the confidence threshold and the σ value to initialise a validity range.

The first one, the confidence value, is used to remove *Context Agents* with a low confidence level. This parameter is an optimization of ALEX behaviour, optimizing the memory used by ALEX by anticipating *Context Agents* uselessness. A low value (near 0) may offer some resilience, by keeping old *Context Agents* which can be re-used in case of a change in the Tutor's policy. On contrary, a high value (near the initial confidence value of 0.5), may impose to keep only the most recent *Context Agents*, reducing the number of *Context Agents*.

The second one, the σ value, has a stronger impact on ALEX behaviour. This parameter is involved in *Context-Agents* initialisation. This parameter has an influence on a *Context Agent* capacity to grow. Indeed, a value that is too low may impede the *Context Agents* to grow, leading to a mapping policy composed of many small *Context-Agents*. We have illustrated in the Teach Robot Yourself experiment that, as long as the situations in which an action must be changed are mapped, the paving of the space made by ALEX could have some holes and still be in functional adequacy. At the opposite, a value that is too high may lead the *Context Agents* to generalize too much. A consequence is that the system may have to face more Non Cooperative Situations, having an impact on its time to converge. Hand-tuning the σ value is complex as it depends of the variations of the observation associated to the validity range to initialize. Those variations are also contextual: an observation may have low variation in a particular context and high one in another. In ALEX, σ is fixed as a percentage of the last variation of the associated observation and thus σ is contextual. However, by doing so, the σ value is sensitive to the observation variation, which may have an impact on time to converge in case of highly noisy environments and demonstration mistakes. A perspective of the work is then to study the *Context Agent* creation and extends the self-observation capacities of the *Percept Agents*.

6.5.2.2 The Scalability

While it has not been tested in the experiments presented in this thesis, many factors make ALEX scalable.

First is the openness property of ALEX which allows actions and observations to be added dynamically during the process of demonstration. The Incremental Design Experiment illustrated such possibility.

Secondly is the *Context Agent* behaviour. One may think that complexity in ALEX may arise from the increasing of the number of Context-Agents. During the self-organization process, *Context Agents* generalize to unseen situations. A consequence is that not every Tutor's action lead to a Context-Agent creation (this is illustrated in the Teach Robot Yourself Experiment). *Context Agents*, through self-organization, seek to resolve the overlaps of *Context Agents* which can lead to situations where more than one *Context Agent* is *valid* at the same time. A consequence is that few *Context Agents* are *valid* at the same time (ideally, the current situation is mapped by only one *Context Agent*). The mechanism that allows *Percept Agent* to send updates only to *Context Agents* that needs it allows to reduce the number of messages, and then the complexity of ALEX behaviour. In perspective, the detection of useless data, and then the deactivation of a *Percept Agent*, can further reduce this complexity.

Lastly, the distribution of the control inside the different functionalities is also a factor of scalability. An instance of ALEX seeks to establish the correlations that only concern the effector that it controls. Its decisions are purely local, based on a self-observation of its environment. Then, at the macro-level (composed of many ALEX instances), there is no combinatorial, as each ALEX is autonomous. This was the hypothesis of the Extreme Sensitive Robotic paradigm and the experiments we performed, while not testing hundred of parallel components, are reassuring indicators of its truth.

6.5.2.3 The Noise

The notion of noise in ALEX is particular. ALEX uses no *a priori* model to interpret signal. On contrary, ALEX builds its own models through the *Context Agents*. The semantic interpretation of signals is in fact performed by the Tutor during the demonstrations. The noise is then to be sought in the Tutor's activity. The Teach Robot Yourself illustrates that, in a certain extent, ALEX manages to filter demonstration mistakes. But the Incremental Design Experiment shows that, if the observations are not contextual enough, ALEX fails in its imitation task. The question of noise then becomes: are the data contextual enough in regard of the policy demonstrated by the Tutor? This is still an open question. We tried to provide some elements of answer in the Incremental Design Experiment. However, the analysis made in this experiment is hand-crafted by a human expert. A perspective is then to automate this process, notably the discovery of missing and useless data. This could be performed by the formalisation of new Non Cooperative Situations and new mechanisms of cooperation (this point is discussed in perspective).

Self-Organization of Robotic Devices Through Demonstrations

Conclusion & Perspectives

7 Conclusion and Perspectives

*This last chapter discusses of the **evaluations** of ALEX in three different experimentations. For each experimentation, we present the hypothesis to test, the motivations and we discuss the results we obtained. This chapter ends with a general synthesis pointing out ALEX properties and limitations.*

General Conclusion

This thesis started from the assessment that robotics is more and more becoming *ambient*. The drastic reduction of the cost of electronic components and the augmentation of their computational capacities make available libraries of various components, which once distributed in the environment, should provide various services and functionalities to users in a transparent way. A consequence is that the problem of designing a robot is becoming a problem of integration which consists in aggregating the adequate components to achieve a particular goal. In the context of service robotics, this particular goal is to satisfy the users by pro-actively offering services.

Several factors make the *ad hoc* design of a controller supervising the activity of such system more and more complex: factors such as the dynamical aspect of the system, the nature of user needs, and the maintenance and evolutions costs of such a system.

Through this thesis, we explore a design approach named Extreme Sensitive Robotics, which is an integrative approach of autonomous robotic components where each component dynamically learns its own control policy. It proposes a bottom-up design focusing on the functionality and services offered by the different components of the system instead of a top-down approach focusing on a sub-division of the tasks to achieve. To achieve such vision, each component needs to be able to dynamically learn and adapt a control policy to what it perceives from its environment (including the human activity).

In the chapter 2, we seek in the scientific literature an approach to learn a control policy with the capacities to deal with properties imposed by ambient systems. Such an approach must be independent of the task to learn, user-centered, have an on-line learning capacity and respect the openness property. This state of the art enables us to highlight the *Learning from Demonstration* paradigm as a good candidate.

The chapter 3 draws a complete overview of the *Learning from Demonstration* paradigm, presenting the whys and wherefores. The different techniques to learn a control policy from demonstrations are discussed and the chapter is concluded with a list of requirements and proposals to enable the usage of the *Learning From Demonstration* paradigm in a an ambient context.

However, the complexity of ambient systems induces severe limitations on the design process of a new system. Facing this complexity requires a conceptual break in the way we design artificial systems. The chapter 4 introduces the fundamental concepts of the *Adaptive Multi-Agent Systems* approach. The AMAS approach focuses on the design of cooperative interactions between autonomous agents that have to collectively solve a common task or reach a common objective. From agent self-organization and their cooperative interactions emerge a global behaviour that is more than the sum of its parts. We conclude the chapter with the adequacy of the AMAS approach with the target objective to learn a control policy.

The chapter 5 introduces ALEX, an acronym for Adaptive Learner by EXperiments, our main contribution. ALEX is an adaptive multi-agent system that we designed to learn a control policy from demonstrations performed by a tutor. In ALEX, each agent is unaware of the global task to achieve. An agent only focuses on its local objectives and the resolution of Non Cooperative Situations. Those resolutions of Non Cooperative Situations lead to changes in agents organization, and those changes impact the global behaviour of ALEX, guiding it toward a functionally adequate state.

ALEX behaviour is the result of a coupling between a set of *Context Agents* and an *Input Agent*. On the one hand, the *Input Agent* is the entity responsible of sending the adequate actions to the controlled system. At any step, the *Input Agent* observes the actions performed by the Tutor and receives an action proposal from the set of *Context Agents*. Those two actions enable both the *Input Agent* to maintain a functionally adequate behaviour and to generate a feedback about the activity of the set of *Context Agents*. On the other hand, the set of *Context Agents* are at the heart of the learning process. Each *Context Agent* models the subspace of the environmental state in which an action is performable. However, a *Context Agent*, observed lonely, is not an absolute or probabilistic activation rule. It is both the result and a participant of a self-organization process where a paving of the space is made by *Context Agents*. Thanks to the feedback from the *Input Agent* and a self-observation of the environment, they dynamically self-organize to collectively build a mapping policy enabling the *Input Agent* to always dispose of adequate information about the action to perform. Thus, the activity of the *Input Agent* influences the activity of *Context Agents* and reciprocally. But the activity of one *Context Agent* has also an impact on the activity of the other *Context Agents*. Those coupling are the motors guiding the system toward a functionally adequate state.

At last the chapter 6 introduces and discusses the different experiments that we performed. The first experiment, named Mountain Car problem, is the illustration of ALEX behaviour on a well-known toy-problem which both illustrates ALEX capacity to learn and to set up pointers of comparison. The second experiment, named Teach Robot Yourself, illustrates ALEX capacity to learn to control a two wheeled rover on a task of collect. The last experiment, that we named Iterative Design Process, intends to show how ALEX can be used to ease and accelerate the design of a robotic application.

Contribution

The contribution offered by this thesis is dual with both the *Extreme Sensitive Robotic* paradigm and ALEX. Hence, we can consider this contribution at three levels: to the robotic field, to machine learning and to the Multi-Agent and AMAS approach.

Contribution to Robotics

With the *Extreme Sensitive Robotic* paradigm, this thesis offers an iterative and bottom-up approach to design a robotic application which rests on the distribution of the control inside the different components of the system. Then, the design of components with self-adaptive properties offers the promises of robotic applications which are easily extensible, reconfigurable and reusable. This conceptual breakthrough in the way we think and design robotic applications is a first answer to the need to integrate as quickly as possible the technological developments while answering to the more and more dynamic needs of its users.

This thesis also contributes, with ALEX, to the challenges of collaborative robotics, an emerging trend in the robotic field which proposes to enable to humans and robots to have natural interactions and to collaborate. By the usage of the *Learning From Demonstration* paradigm, ALEX enables end-users to *tele-operate* each component to demonstrate a behaviour. By observing the environmental state and correlating it to the user actions, ALEX can pro-actively perform actions on behalf on the user. Its on-line learning capacity enables the users to act at any moment to correct the system behaviour or to demonstrate new activities.

Contribution to Machine Learning

The main contribution to the machine learning field is ALEX. From its functional aspect, ALEX is an on-line supervised classification algorithm of continuous and sequential data. An hypothesis that differentiates ALEX from other approaches is that the vector of data and the actions space are considered to be *a priori* unknown and dynamic (some of them can appear and disappear).

However, the contribution offered by this thesis with ALEX is not restricted to this functional aspect and contributes to a bigger challenge. ALEX learning capacity emerges from self-organization of its inner agents, offering a higher adaptivity to the dynamic changes of its environment. This thesis is another demonstration that self-organization is a powerful approach to face complexity. A mantra that we followed during the design of self-organization in ALEX, which was imposed by the dynamic nature of the environment and the task dependency requirement, was to make the fewest assumptions possible, seeking to obtain a certain form of *agnostic learning* through self-organization. The objective was to provide a generic learning technique in which the hypothesis of application are based on the characterisation of the environment and not on a model of this environment or on any form of *a priori* knowledge. With ALEX, we achieved a certain level of genericness by not making strong assumption on the nature of the action to perform and the vector of data.

To allow this, the design of ALEX focused on the micro-level, giving agents the capacity to self-observe. From this self-observation, combined with self-organization and cooperation, the agents generate their own feedbacks and motivations, producing at the macro-level in a coherent global activity. This small step toward agnostic learning is modest but is very encouraging about the possibility at long term to design a system capable of generating its own feedbacks (this point is discussed in perspectives).

Contribution to AMAS

This thesis offers another validation of the AMAS approach and its capacity to deal with complexity. More precisely, the development of ALEX participates in the maturity of the approach on the issues of control and learning, especially on the notions of self-observation and self-organization. The work presented is in continuity with the work on Context-Learning with Adaptive Multi-Agent Systems initiated by [Videau, 2011], and enriched by [Boes, 2014],[Guivarch, 2014], and [Gatto et al., 2013]. By applying *Context Agents* to another domain, this thesis participates to the formalization and the validation of the Context-Learning approach, where learning is the emerging result of the self-organization of a set of *Context Agents* (this point is discussed in perspective in the section 7). More globally, this thesis illustrates and promotes a fundamental concept at the very heart of the AMAS approach: facing complexity by the distribution of the control and the locality of the decisions.

Perspectives

This thesis provides contributions but also leads to numerous perspectives of work. The work performed in this thesis enables to highlight three area of research: the extension of self-observation capacities, the formalisation and comparison of the Context-Learning approach and the ultimate quest of non finality.

Self-Observation

The experiments that we performed enable to highlight some interesting perspectives of evolutions for ALEX. Those perspectives globally focus on the extension of ALEX self-observation capacities.

Two parameters in ALEX are still empirically fixed: one for the confidence threshold and the other one for the initialisation of validity ranges. The confidence threshold is enabling memory optimisation in ALEX and has a low influence on its behaviour. Meanwhile, the second parameter may have more or less an impact on system's convergence speed and giving the ability to self-adapt those values is then an interesting perspective. A track to allow this adaptation is to study the interaction between the *Percept Agents* and the *Context Agents*. Indeed, the initialization of validity ranges is made thanks to the information provided by the *Percept Agents*. By observing the impact of the activity of the *Percept Agents* on the system, which means by observing if the value provided by a *Percept Agent* is leading to Non Cooperative Situations or not, the *Percept Agent* may dispose of enough information to adjust the way this information is provided. Using the same source of information, the *Percept Agents* may then be able to locally determine if their own activity is harmful for the system, which means leading to Non Cooperative Situations, and decide to suppress themselves. The system will then be able to detect and filtrate useless data. However, such mechanism has to enable the capacity to create or re-create *Percept Agents* if they become later needed.

Another perspective is to study the problematic of hidden variables. The Incremental Design Experiment has shown that we can manually detect that a data is missing through the observation of *Context Agent* validity ranges. We believe that such detection is feasible by the agents themselves, from the observation of their own activity and its impact on system's functional adequacy. A track to investigate is to detect in which context an ambiguity arises and in which context this ambiguity disappears. Then, by creating a synthetic *Percept Agent* that switches of value between those two contexts, the system may solve by itself this ambiguity (this idea is really close of the notion of synthetic items that we can find in *schema learning*).

At last, ALEX learns only when a Tutor is acting. However, even without any activity from the Tutor, the system may self-observe trying to resolve or anticipate Non Cooperative Situations. This problematic of self-observation goes far beyond this thesis work. The work of this thesis, however, leads us to believe that this is a subject of exciting researches.

The Context-Learning Pattern

This work takes part in a more comprehensive approach that proposes to build Adaptive Multi-Agent Systems with the ability to learn to manage the context in which they are plugged-in. The usage of *Context Agents* in an Adaptive Multi-Agent System was first introduced by [Videau, 2011], which applied them to the control of bio-process. This pioneering work gave birth to three other works:

▷ ESCHER [Boes, 2014], an Adaptive Multi-Agent System designed to learn to optimize the control of complex systems. ESCHER has been validated in the context of combustion engines optimization. ESCHER uses *Context Agents* to learn the local influence of a continuous action on the variation of the inputs. With this information, ESCHER can learn the consequences of the performance of an action under a certain context and use those information to guide the system toward an optimum.

▷ SAVER [Gatto et al., 2013], an Adaptive Multi-Agent System designed to optimize the energy consumption of buildings. SAVER shares some similitude with ESCHER, the main difference residing in the nature of the action to perform.

▷ [Guivarch, 2014] proposed AMADEUS, an Adaptive Multi-Agent System to learn a user's recurring action in ambient systems. AMADEUS is the system which shares the more similitude with ALEX. Here, the main difference lies in the role of users. In AMADEUS, the user is passive whereas ALEX requires an active interaction.

With this thesis, ALEX joins the Context-Learning family. ALEX shares similitude with those systems, reusing partly the same structure of agents. Those similitude have led us to study what are the key features of using the *Context Agents* and how to abstract and reuse them. This work led to the proposition of the *Self-Adaptive Context Learning Pattern* [Boes et al., 2015], which presents the heart and the differences of the usage of *Context Agents*. The design of ALEX, as presented in this thesis, is built on this pattern.

The next step of this more global study of the Context-Learning approach is to make a comparative study to highlight the pro and cons, not only by comparing our own approaches one with the other, but also to other state-of-the-art approaches. We are currently applying the different Context-Learning systems to the Mario AI benchmark [Karakovskiy and Togelius, 2012] in order to compare them to other methods issued from the state-of-the-art.

The Context-Learning approach still needs to gain in formalization and enriched, notably by increasing the self-observation capacities of the *Percept* and *Context Agents*. Works are currently being done on the subject, offering a large field of applications, from the optimization of users comfort to the automatic generation of models. ALEX is a milestone in this work, asking as many questions as it answers.

The Scientific Lock of Non Finality

At last, this thesis modestly participates to a long term quest toward the *non finality*. The scientific lock associated with the *non finality* is that the information available on the usages of a software are not sufficient or completely specifiable during its design. We are unable to clearly specify how the systems have to behave. This is a more general description of the integrator problem (see section 1.5).

The lack of explicit finality also expresses the inability to evaluate their activity in regards to the objectives to achieve, since those objectives are not clearly defined. Thus, they do not possess any cost or evaluation function that are dependant of the application. This postulate involves to seek for new paradigms and novel approaches to face the unknown. The key feature is then to enable those systems to self-adapt to their context in order to maintain a coherent activity. The *non finality* is then ensuring their adaptation. Ideas similar to what we express under the term *non finality* may be found in the scientific literature such as Intrinsically Motivated Learning [Baldassarre and Mirolli, 2013] or Task-Agnostic Learning [Doncieux and Mouret, 2014].

Such an objective involves to refrain from making assumptions that are external to the system, focusing only on its internal medium. The system has to only use its own activity and its interaction with its environment as the only source of feedback, generating its own motivations and objectives. The fact that the designers have no control on what is external to the system implies that the design should be done by a bottom-up approach.

By proposing a bottom-up focusing on the design of cooperative local behaviours, the AMAS approach appears to be in total adequacy with this quest. In a certain extent, the work of this thesis tried, with the design of ALEX, to participate to this quest. However with ALEX, there is still an external source of information through the Tutor's activity. But the work that we have made with the Context-Learning approach, and the perspectives offered by self-observation capacities, make us one step closer to this ultimate goal. What was once unthinkable, now seems reachable, yet the price to reach it still requires few years of researches.

Own Bibliography

International Journal

▷ Verstaevel Nicolas, Régis Christine, Gleizes Marie-Pierre, Robert Fabrice. *Principles and Experimentations of Self-Organizing Embedded Agents Allowing Learning From Demonstration in Ambient Robotics.*, Future Generation Computer Systems, Elsevier, 2016.

International Conference

▷ Verstaevel Nicolas, Régis Christine, Guivarch Valérian, Gleizes Marie-Pierre, Robert Fabrice. *A Distributed User-Centered Approach For Control In Ambient Robotics*, Embeded Real-Time Software and Systems (ERTS2016), 2016.

▷ Verstaevel Nicolas, Régis Christine, Guivarch Valérian, Gleizes Marie-Pierre, Robert Fabrice. *Extreme Sensitive Robotics - A Context-Aware Ubiquitous Learning*, International Conference on Agents and Artificial Intelligence (ICAART 2015), 1, 242-248, INSTINC, 2015.

▷ Verstaevel Nicolas, Régis Christine, Gleizes Marie-Pierre, Robert Fabrice. *Principles and experimentations of self-organizing embedded agents allowing learning from demonstration in ambient robotics*, The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), Procedia Computer Science, 52, 194-201, Elsevier, 2015.

▷ Boes Jérémy, Nigon Julien, Verstaevel Nicolas, Gleizes Marie-Pierre, Migeon Frédéric. *The Self-Adaptive Context Learning Pattern: Overview and Proposal*, Modelling and Using Context,vol. 9405 ,91-104 , Springer International Publishing, 2015.

National Conference

▷ Verstaevel Nicolas, Régis Christine, Gleizes Marie-Pierre, Robert Fabrice. *Auto-organisation d'agents embarqués pour l'apprentissage par démonstration: principes et expérimentations*, 23es Journées Francophones sur les Systèmes Multi-Agents (JFSMA'15), 91-100, Cépaduès, 2015.

Bibliography

AAMODT, A. AND PLAZA, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications 7,* 1, 39–59.

ACKLEY, D. H., HINTON, G. E., AND SEJNOWSKI, T. J. 1985. A learning algorithm for boltzmann machines*. *Cognitive science 9,* 1, 147–169.

ADLER, S., SCHMITT, S., WOLTER, K., AND KYAS, M. 2015. A survey of experimental evaluation in indoor localization research. In *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*. 1–10.

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.

ASHBY, W. R. ET AL. 1956. An introduction to cybernetics. *An introduction to cybernetics.*.

BALDASSARRE, G. AND MIROLLI, M. 2013. *Intrinsically motivated learning systems: an overview*. Springer.

BANDURA, A. AND MCCLELLAND, D. C. 1977. Social learning theory.

BANDURA, A., ROSS, D., AND ROSS, S. A. 1961. Transmission of aggression through imitation of aggressive models. *Journal of Abnormal and Social Psychology*, 63, 575–82.

BAZIRE, M. AND BRÉZILLON, P. 2005. Understanding context before using it. In *Modeling and using context*. Springer, 29–40.

BERNON, C., GLEIZES, M.-P., PEYRUQUEOU, S., AND PICARD, G. 2002. Adelfe: a methodology for adaptive multi-agent systems engineering. In *Engineering Societies in the Agents World III*. Springer, 156–169.

BILLARD, A., CALINON, S., DILLMANN, R., AND SCHAAL, S. 2008. Robot programming by demonstration. In *Springer handbook of robotics*. Springer, 1371–1394.

BILLING, E. A. AND HELLSTRÖM, T. 2010. A formalism for learning from demonstration. *Paladyn, Journal of Behavioral Robotics 1,* 1, 1–13.

BLACKRNORE, S. 1999. The meme machine.

BOES, J. 2014. Apprentissage du contrôle de systèmes complexes par l'auto-organisation coopérative d'un système multi-agent. Ph.D. thesis, Université de Toulouse III-Paul Sabatier.

BOES, J., NIGON, J., VERSTAEVEL, N., GLEIZES, M.-P., AND MIGEON, F. 2015. The self-adaptive context learning pattern: Overview and proposal. In *Modeling and Using Context*. Springer, 91–104.

BONGARD, J. C. 2013. Evolutionary robotics. *Communications of the ACM 56,* 8, 74–83.

BONJEAN, N., MEFTEH, W., GLEIZES, M.-P., MAUREL, C., AND MIGEON, F. 2014. Adelfe 2.0. In *Handbook on Agent-Oriented Design Processes*. Springer, 19–63.

BOURJOT, C., DESOR, D., AND CHEVRIER, V. 2011. Stigmergy. In *Self-organising Software*, G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, Eds. Natural Computing Series. Springer Berlin Heidelberg, 123–138.

BRAITENBERG, V. 1986. *Vehicles: Experiments in synthetic psychology.* MIT press.

BRAX, N., ANDONOFF, E., GEORGÉ, J.-P., GLEIZES, M.-P., AND MANO, J.-P. 2013. Mas4at, un sma auto-adaptatif pour le déclenchement d'alertes dans le cadre de la surveillance maritime. *Revue d'intelligence artificielle 27,* 3, 371–395.

BROOKS, R. ET AL. 1986. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of 2,* 1, 14–23.

BROOKS, R. A. 1990. Elephants don't play chess. *Robotics and autonomous systems 6,* 1, 3–15.

BROOKS, R. A. 1995. Intelligence without reason. *The artificial life route to artificial intelligence: Building embodied, situated agents*, 25–81.

BRUNO, D., CALINON, S., AND CALDWELL, D. 2014. Learning adaptive movements from demonstration and self-guided exploration. In *Development and Learning and Epigenetic Robotics (ICDL-Epirob), 2014 Joint IEEE International Conferences on*. 101–106.

BRYS, T., HARUTYUNYAN, A., SUAY, H. B., CHERNOVA, S., TAYLOR, M. E., AND NOWÉ, A. 2015. Reinforcement learning from demonstration through shaping. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*

BUTZ, M. V. 2015. Learning classifier systems. In *Springer Handbook of Computational Intelligence*. Springer, 961–981.

CALABRESE, M., AMATO, A., DI LECCE, V., AND PIURI, V. 2010. Hierarchical-granularity holonic modelling. *Journal of Ambient Intelligence and Humanized Computing 1,* 3, 199–209.

CALL, J. AND CARPENTER, M. 2002. Three sources of information in social learning. *Imitation in animals and artifacts*, 211–228.

CAPERA, D. 2005. ThÃ¨se de doctorat. Ph.D. thesis, Université Paul Sabatier, Toulouse, France. (Soutenance le 23/06/2005).

CARRERA, A., PALOMERAS, N., HURTÓS, N., KORMUSHEV, P., AND CARRERAS, M. 2015. Cognitive system for autonomous underwater intervention. *Pattern Recognition Letters 67, Part 1*, 91 – 99. Cognitive Systems for Knowledge Discovery.

CELEMIN, C. AND RUIZ-DEL SOLAR, J. 2015. Coach: Learning continuous actions from corrective advice communicated by humans. In *Advanced Robotics (ICAR), 2015 International Conference on*. IEEE, 581–586.

CHAPUT, H. H. 2004. *The constructivist learning architecture: A model of cognitive development for robust autonomous robots*. Computer Science Department, University of Texas at Austin.

CHERNOVA, S. AND VELOSO, M. 2007. Confidence-based policy learning from demonstration using gaussian mixture models. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 233.

CHOMSKY, N. 1959. A review of bf skinner's verbal behavior. *Language 35,* 1, 26–58.

CIOARGA, R., NALATAN, I., TURA-BOB, S., MICEA, M., CRETU, V., BIRIESCU, M., AND GROZA, V. 2008. Emergent exploration and resource gathering in collaborative robotic environments. In *Robotic and Sensors Environments, 2008. ROSE 2008. International Workshop on*. 13–18.

CORNUÉJOLS, A. AND MICLET, L. 2011. *Apprentissage artificiel: concepts et algorithmes*. Editions Eyrolles.

CURRAN, W. 2015. Developing learning from demonstration techniques for individuals with physical disabilities. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts*. HRI'15 Extended Abstracts. ACM, New York, NY, USA, 233–234.

DAUTENHAHN, K.; NEHANIV, C. A. A. 2003. Learning by experience from others : Social learning and imitation in animals and robots. In *Adaptivity and Learning*. Springer Berlin Heidelberg, 217–241.

DAUTENHAHN, K. AND NEHANIV, C. L. 2002. Imitation in animals and artifacts. MIT Press, Cambridge, MA, USA, Chapter The Agent-based Perspective on Imitation, 1–40.

DE WOLF, T. AND HOLVOET, T. 2005. Emergence versus self-organisation: Different concepts but promising when combined. In *Engineering self-organising systems*. Springer, 1–15.

DENG, L. AND YU, D. 2014. Deep learning: methods and applications. *Foundations and Trends in Signal Processing 7,* 3–4, 197–387.

DEVAUTL, T., FORREST, S., TANIMOTO, I., SOULE, T., AND HECKENDORN, R. 2015. Learning from demonstration for distributed, encapsulated evolution of autonomous outdoor robots. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. GECCO Companion 2015. ACM, New York, NY, USA, 1381–1382.

Di Marzo Serugendo, G., Gleizes, M.-P., and Karageorgos, A. 2011a. History and definitions. In *Self-organising Software*, G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, Eds. Natural Computing Series. Springer Berlin Heidelberg, 33–74.

Di Marzo Serugendo, G., Gleizes, M.-P., and Karageorgos, A. 2011b. *Self-organising systems*. Springer.

Dittenbach, M., Merkl, D., and Rauber, A. 2000. The growing hierarchical self-organizing map. In *ijcnn*. IEEE, 6015.

Doncieux, S., Bredeche, N., Mouret, J.-B., and Eiben, A. E. G. 2015. Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI 2*, 4.

Doncieux, S. and Mouret, J.-B. 2014. Beyond black-box optimization: a review of selective pressures for evolutionary robotics. *Evolutionary Intelligence 7,* 2, 71–93.

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

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

Ferrari, P. F., Visalberghi, E., Paukner, A., Fogassi, L., Ruggiero, A., and Suomi, S. J. 2006. Neonatal imitation in rhesus macaques. *PLoS biology 4,* 9, 1501.

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

Fiol, C. M. and Lyles, M. A. 1985. Organizational learning. *Academy of management review 10,* 4, 803–813.

Fonooni, B., Jevtić, A., Hellstrom, T., and Janlert, L.-E. 2015. Applying ant colony optimization algorithms for high-level behavior learning and reproduction from demonstrations. *Robotics and Autonomous Systems 65*, 24 – 39.

Friedenberg, J. and Silverman, G. 2011. *Cognitive science: An introduction to the study of mind*. Sage.

Gatti, C. 2015. The mountain car problem. In *Design of Experiments for Reinforcement Learning*. Springer, 95–109.

Gatto, F., Gleizes, M.-P., and Elicegui, L. 2013. Saver: Self-adaptive energy saver. In *European Workshop on Multi-Agent Systems (EUMAS), Toulouse, France*.

Georgé, J.-P., Edmonds, B., and Glize, P. 2004. Making self-organising adaptive multiagent systems work. In *Methodologies and Software Engineering for Agent Systems*. Springer, 321–340.

Georgé, J.-P., Gleizes, M.-P., and Camps, V. 2011. Cooperation. In *Self-organising Software*, G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, Eds. Natural Computing Series. Springer, http://www.springerlink.com, 193–226.

GEORGÉ, J.-P., GLEIZES, M. P., AND GLIZE, P. 2003. Conception de systèmes adaptatifs à fonctionnalité émergente: la théorie amas. *Revue d'intelligence artificielle 17,* 4, 591–626.

GEORGÉ, J.-P., GLEIZES, M.-P., GLIZE, P., AND RÉGIS, C. 2003. Real-time simulation for flood forecast: an adaptive multi-agent system staff. In *Proceedings of the AISB*. Vol. 3. 109–114.

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*. Springer, 58–75.

GLIZE, P. 2001. L'adaptation des systèmes à fonctionnalité émergente par auto-organisation coopérative. Ph.D. thesis, Université Paul Sabatier.

GOLDSTEIN, J. 1999. Emergence as a construct: History and issues. *Emergence 1,* 1, 49–72.

GOSS, S., ARON, S., DENEUBOURG, J.-L., AND PASTEELS, J. M. 1989. Self-organized shortcuts in the argentine ant. *Naturwissenschaften 76,* 12, 579–581.

GUERIN, F. 2011. Learning like a baby: a survey of artificial intelligence approaches. *The Knowledge Engineering Review 26,* 02, 209–236.

GUIVARCH, V. 2014. Prise en compte de la dynamique du contexte pour les systèmes ambiants par systèmes multi-agents adaptatifs. Ph.D. thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier.

GUIVARCH, V., CAMPS, V., AND PÉNINOU, A. 2012. Context awareness in ambient systems by an adaptive multi-agent approach. In *Ambient Intelligence*. Springer Berlin Heidelberg, 129–144.

HECHT, T., LEFORT, M., AND GEPPERTH, A. 2015. Using self-organizing maps for regression: the importance of the output function. In *European Symposium On Artificial Neural Networks (ESANN)*.

HEIDRICH-MEISNER, V. AND IGEL, C. 2008. Variable metric reinforcement learning methods applied to the noisy mountain car problem. In *Recent advances in reinforcement learning*. Springer, 136–150.

HEINEN, M. R. AND ENGEL, P. M. 2010. An incremental probabilistic neural network for regression and reinforcement learning tasks. In *Artificial Neural Networks–ICANN 2010*. Springer, 170–179.

HERMAN, L. M. 2002. Vocal, social, and self-imitation by bottlenosed dolphins.

HEYES, C. M. AND GALEF JR, B. G. 1996. *Social learning in animals: the roots of culture*. Elsevier.

HOLLAND, J. H. 1975 Reprinted in 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA.

HOLMES, M. P. ET AL. 2004. Schema learning: Experience-based construction of predictive action models. In *Advances in Neural Information Processing Systems*. 585–592.

HUTCHINS, E. 1995. *Cognition in the Wild*. MIT press.

JAZDI, N. 2014. Cyber physical systems in the context of industry 4.0. In *Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on*. IEEE, 1–4.

JORQUERA, T. 2013. An adaptive multi-agent system for self-organizing continuous optimization. Ph.D. thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier.

KADDOUM, E. 2011. Optimization under constraints of distributed complex problems using cooperative self-organization. Ph.D. thesis, Université de Toulouse.

KALENKA, S. AND JENNINGS, N. R. 1999. Socially responsible decision making by autonomous agents. In *Cognition, Agency and Rationality*. Springer, 135–149.

KANT, G. AND SANGWAN, K. S. 2015. Predictive modelling and optimization of machining parameters to minimize surface roughness using artificial neural network coupled with genetic algorithm. *Procedia CIRP 31*, 453–458.

KARAKOVSKIY, S. AND TOGELIUS, J. 2012. The mario ai benchmark and competitions. *Computational Intelligence and AI in Games, IEEE Transactions on 4,* 1, 55–67.

KNOX, W. B., SETAPEN, A. B., AND STONE, P. 2011. Reinforcement learning with human feedback in mountain car. In *AAAI Spring Symposium: Help Me Help You: Bridging the Gaps in Human-Agent Collaboration*.

KNOX, W. B. AND STONE, P. 2009. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*. ACM, 9–16.

KNOX, W. B., STONE, P., AND BREAZEAL, C. 2013. Training a robot via human feedback: A case study. In *Social Robotics*. Springer International Publishing, 460–470.

KOBER, J. AND PETERS, J. 2012. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*. Springer, 579–610.

KOHONEN, T. 2001. Self-organizing maps, vol. 30 of springer series in information sciences.

KOLODNER, J. 2014. *Case-based reasoning*. Morgan Kaufmann.

KRAMER, O. 2010. Evolutionary self-adaptation: a survey of operators and strategy parameters. *Evolutionary Intelligence 3,* 2, 51–65.

LEGG, S. AND HUTTER, M. 2007. Universal intelligence: A definition of machine intelligence. *Minds and Machines 17,* 4, 391–444.

LEMOUZY, S. 2011. Systèmes interactifs auto-adaptatifs par systèmes multi-agents auto-organisateurs: application à la personnalisation de l'accès à l'information. Ph.D. thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier.

LEONARD, D. C. 2002. *Learning Theories: A to Z: A to Z*. ABC-CLIO.

LUNGARELLA, M., METTA, G., PFEIFER, R., AND SANDINI, G. 2003. Developmental robotics: a survey. *Connection Science 15,* 4, 151–190.

MACGLASHAN, J. AND LITTMAN, M. L. 2015. Between imitation and intention learning. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, 3692–3698.

MAZAC, S. 2015. Approche décentralisée de l'apprentissage constructiviste et modélisation multi-agent du probléme d'amorçage de l'apprentissage sensorimoteur en environnement continu. application à l'intelligence ambiante. Ph.D. thesis, Université de Lyon.

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.

MEDLER, D. A. 1998. A brief history of connectionism. *Neural Computing Surveys 1,* 18–72.

MICHALSKI, R. S. 1993. Toward a unified theory of learning: Multistrategy task-adaptive learning. In *IN: READINGS IN KNOWLEDGE ACQUISITION AND*. Citeseer.

MICHEL, O. 1998. Webots: Symbiosis between virtual and real mobile robots. In *Virtual Worlds*. Springer, 254–263.

MINSKY, M. 1988. *Society of mind*. Simon and Schuster.

MINSKY, M. L. 1983. *Learning meaning*. Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

MITCHELL, R. W. 1987. A comparative-developmental approach to understanding imitation. In *Perspectives in ethology*. Springer, 183–215.

MITCHELL, T. M. 2006. *The discipline of machine learning*. Vol. 17. Carnegie Mellon University, School of Computer Science, Machine Learning Department.

MITIĆ, M. AND MILJKOVIĆ, Z. 2014. Neural network learning from demonstration and epipolar geometry for visual control of a nonholonomic mobile robot. *Soft Computing 18,* 5, 1011–1025.

MNIH, V., KAVUKCUOGLU, K., SILVER, D., GRAVES, A., ANTONOGLOU, I., WIERSTRA, D., AND RIEDMILLER, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

MOLLARD, Y., MUNZER, T., BAISERO, A., TOUSSAINT, M., AND LOPES, M. 2015. Robot programming from demonstration, feedback and transfer. In *Intelligent Robots and Systems (IROS)*.

MOORE, A. W. 1990. Efficient memory-based learning for robot control, Thesis, University of Cambridge, Computer Laboratory, 1990.

MORAVEC, H. 1988. *Mind children: The future of robot and human intelligence*. Harvard University Press.

MORIN, E. 1990. *Introduction à la pensée complexe*. Vol. 96. Esf Paris.

NADEL, J. 1986. *Imitation et communication entre jeunes enfants*. Vol. 13. Presses Universitaires de France-PUF.

NADEL, J. E. AND BUTTERWORTH, G. E. 1999. *Imitation in infancy.* Cambridge University Press.

NELSON, A. L., BARLOW, G. J., AND DOITSIDIS, L. 2009. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems 57,* 4, 345–370.

NEWELL, A. AND SIMON, H. A. 1976. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM 19,* 3, 113–126.

NORBERT, W. 1948. Cybernetics or control and communication in the animal and the machine. *Hertnann, Paris*.

PAGLIUCA, P. AND NOLFI, S. 2015. Integrating learning by experience and demonstration in autonomous robots. *Adaptive Behavior*.

PAVLOV, I. P. 1941. Lectures on conditioned reflexes. vol. ii. conditioned reflexes and psychiatry.

PERERA, C., ZASLAVSKY, A., CHRISTEN, P., AND GEORGAKOPOULOS, D. 2014. Context aware computing for the internet of things: A survey. *Communications Surveys & Tutorials, IEEE 16,* 1, 414–454.

PEROTTO, F. S., BUISSON, J.-C., AND ALVARES, L. O. 2007. Constructivist anticipatory learning mechanism (calm)–dealing with partially deterministic and partially observable environments. In *COGNITIVE DEVELOPMENT IN ROBOTIC SYSTEMS. LUND UNIVERSITY COGNITIVE STUDIES, 135.* Citeseer.

PFEIFER, R. AND BONGARD, J. 2006. *How the body shapes the way we think: a new view of intelligence*. MIT press.

PIAGET, J. 1945. *La formation du symbole chez l'enfant: imitation, jeu et rêve, image et représentation*. Delachaux et Niestlé Paris.

PIAGET, J. 1954. *The construction of reality in the child*. Basic Books.

PICARD, G. AND GLEIZES, M.-P. 2004. The adelfe methodology. In *Methodologies and Software Engineering for Agent Systems*. Springer, 157–175.

PONS, L. 2014. Self-tuning of game scenarios through self-adaptative multi-agent systems. Ph.D. thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier.

PRÍNCIPE, J. C. AND MIIKKULAINEN, R. 2009. Advances in self-organizing maps. *Lecture Notes in Computer Science 5629*.

RAMOS, C., AUGUSTO, J. C., AND SHAPIRO, D. 2008. Ambient intelligence: the next step for artificial intelligence. *Intelligent Systems, IEEE 23,* 2 (March), 15–18.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. In *ACM Siggraph Computer Graphics*. Vol. 21. ACM, 25–34.

ROSENBLATT, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review 65,* 6, 386.

RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. 1985. Learning internal representations by error propagation. Tech. rep., DTIC Document.

RUSSELL, S. AND NORVIG, P. 1995. Artificial intelligence: a modern approach.

SAUNDERS, J., NEHANIV, C. L., AND DAUTENHAHN, K. 2006. Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM, 118–125.

SCHAAL, S. 1999. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences 3,* 6, 233–242.

SCHMIDHUBER, J. 2015. Deep learning in neural networks: An overview. *Neural Networks 61,* 85–117.

SEARLE, J. R. 1980. Minds, brains, and programs. *Behavioral and brain sciences 3,* 03, 417–424.

SEARLE, J. R. 1998. *Mind, language and society: Philosophy in the real world*. Cambridge Univ Press.

SIMON, H. A. 1983. Why should machines learn? In *Machine learning*. Springer, 25–37.

SKINNER, B. F. 1938. The behavior of organisms: An experimental analysis.

SKINNER, B. F. 2011. *About behaviorism*. Vintage.

SONTAG, E. D. AND BOYD, S. 1995. Mathematical control theory: Deterministic finite-dimensional systems. *IEEE Transactions on Automatic Control 40,* 3, 563–563.

SOUZA, R. D., EL-KHOURY, S., SANTOS-VICTOR, J., AND BILLARD, A. 2015. Recognizing the grasp intention from human demonstration. *Robotics and Autonomous Systems 74, Part A*, 108 – 121.

STARZYK, J. A. 2008. *Motivation in embodied intelligence, Frontiers in Robotics, Automation and Control*. INTECH Open Access Publisher.

SUTTON, R. S. 1996. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 1038–1044.

SUTTON, R. S. AND BARTO, A. G. 1999. Reinforcement learning: An introduction. *Robotica 17,* 2, 229–235.

THORNDIKE, E. L. 1898. Animal intelligence: an experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements 2,* 4, i.

THORNDIKE, E. L. 1927. The law of effect. *The American Journal of Psychology*, 212–222.

THORPE, W. H. 1956. Learning and instinct in animals.

TOMASELLO, M. 1999. *The cultural origins of human cognition*. Harvard University Press.

VERNON, D. 2014. *Artificial cognitive systems: A primer*. MIT Press.

VERSTAEVEL, N., RÉGIS, C., GUIVARCH, V., GLEIZES, M.-P., AND ROBERT, F. 2015. Extreme sensitive robotic: A context-aware ubiquitous learning. In *Proceedings of the 2015 International Conference on Agents and Artificial Intelligence*. Vol. 1. 242–248.

VIDEAU, S. 2011. Contrôle de processus dynamiques par systèmes multi-agents adaptatifs: application au contrôle de bioprocédés. Ph.D. thesis, Toulouse, INSA.

WADSWORTH, B. J. 1996. *Piaget's theory of cognitive and affective development: Foundations of constructivism* . Longman Publishing.

WALTER, W. G. 1951. A machine that learns. *Scientific American 185,* 2, 60–63.

WATKINS, C. J. C. H. 1989. Learning from delayed rewards. Ph.D. thesis, University of Cambridge England.

WEISER, M. 1991. The computer for the 21st century. *Scientific american 265,* 3, 94–104.

WEISS, G. 1999. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

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, D., PARUNAK, H. V. D., MICHEL, F., HOLVOET, T., AND FERBER, J. 2005. Environments for multiagent systems state-of-the-art and research challenges. In *Environments for multi-agent systems*. Springer, 1–47.

WIENER, N. 1948. Cybernetics or control and communication in the animal and the machine.

WILSON, S. W. 1994. Zcs: A zeroth level classifier system. *Evolutionary computation 2,* 1, 1–18.

WILSON, S. W. 1995. Classifier fitness based on accuracy. *Evolutionary computation 3,* 2, 149–175.

ZENTALL, T. R. 1996. An analysis of imitative learning in animals. *Social learning in animals: The roots of culture*, 221–243.

ZENTALL, T. R. 2001. Imitation in animals: evidence, function, and mechanisms. *Cybernetics & Systems 32,* 1-2, 53–96.

List of Figures

1.1 Open-loop control. 9
1.2 Closed-loop control. 9
1.3 The integrator problem. Each modification in the needs or the system's composition involves re-performing the whole design process 12
1.4 An illustration of two braintenberg vehicles. A difference of connection produces a radically different behaviour. 13
1.5 The same problem through the scope of Extreme Sensitive Robotics. As each device is designed to be self-adaptive, the two systems are an equivalent problem. 15

2.1 Reinforcement learning assessment 25
2.2 Cognitivist approach assessment 26
2.3 A formal neuron 27
2.4 A multilayer perception composed of two hidden layers and two output neurons 29
2.5 Illustration of a Self-Organizing map 30
2.6 Neural networks assessment 31
2.7 One-point and Two-points crossover operators 32
2.8 Genetic algorithms assessment 34
2.9 The structure of a schema 36
2.10 Schema learning assessment 37
2.11 Schematic view of an LCS system 38
2.12 Steps of a Case-based reasoning algorithms 39

3.1 Neonatal imitation in Rhesus Macaques [50]. In both situations, the macaque imitates the observed behaviours 43
3.2 Control policy derivation and execution [4] 46
3.3 Structural homologies among tetrapod animals/artifacts [39] 47

3.4 Mapping a teacher execution to the learner [4] 48
3.5 Categorization of approaches to gather the demonstrations [4] 49
3.6 Policy derivation categorisation according to [4] 49

4.1 An agent's lifecyle 60

5.1 The objective of ALEX is to control the input of a functionality in accordance with user needs. 69
5.2 The component view of ALEX. ALEX requires to perceive tutors activity and a set of observations to provide the adequate input corresponding to the current situation. 71
5.3 A schematic view of interactions within ALEX during the nominal behaviour. *Percept Agents* receive observations from the environment and send updates to *Context Agents*. A Context Agent makes a proposal to the *Input Agents* and the *Input Agents* applies the *action*. 73
5.4 An illustration of a validity range with two different $v_{percept}$ values. In (1), $v_{percept} \in [vrange_{min}, vrange_{max}]$. In (2), $v_{percept} \notin [vrange_{min}, vrange_{max}]$. 74
5.5 An illustration of a *valid Context Agent* C_1 and an *invalid Context Agent* C_2. Each *Context Agent* has a validity range associated with the *Percept Agents* P_1 and P_2. The current value of P_1 (the blue line) is out of the validity range of C_2. 75
5.6 An example of ALEX architecture with two *Percept Agents* (purple triangles), a *valid Context Agent* (green circle), an *invalid Context Agent* (red circle) and the *Input Agent* (blue square). Each *Percept Agent* receives updates on an observation from the environment and sends updates to *Context Agents*. *valid Context Agents* make proposals to the *Input Agent* which applies the *action*. . . 76
5.7 Synthesis of NCS 83
5.8 An illustration of the behaviour of an AVT starting at $v_0 = 5$ and $\Delta_0 = 1$ seeking to reach the value 14 with a precision of $+/-0.2$. 85
5.9 Tuning behaviour of v_t and Δ_t of the AVT according to Fb_t and Fb_{t-1}. 85
5.10 An illustration of the behaviour of an AVRT with an example (v_e, IN). Here, a feedback $\downarrow$ is sent to the AVR_{lower} to integrate v_e to the range. 86
5.11 An illustration of the behaviour of an AVRT with an example (v_e, OUT). Here, a feedback $\downarrow$ is sent to the AVR_{upper} to integrate v_e to the range. 86
5.12 An illustration of the behaviour of an AVRT with an example (v_e, IN). Here, a feedback $\sim$ is sent to the AVR_{upper}. 87
5.13 Initialisation of a validity range around the current value v_t. Each bound is placed at a distance $\sigma = (v_t - v_{t-1})/2$ of the value v_t 91

6.1 Objectives of the Mountain Car Experiment 100

6.2 An illustration of the mountain car. An under-powered car situated in a valley must drive up a steep hill and reach the target. 100

6.3 The graphical representation of the Mountain Car for human tutoring. 103

6.4 The 2D projection of the policy learnt by ALEX with a *virtual* tutor. The space is split similarly to the policy of the *virtual* tutor. 104

6.5 The 2D projection of the *virtual* tutor policy. The space is split in two areas depending on the *velocity* value. 104

6.6 The structure of a particular *Context Agent* involved in the Mountain Car problem. The yellow areas are the area in wich the *Context Agent* is *valid*. The stripped areas are the area in which the Context Agent is *validable*. The white area is the current value of the *Percept Agent*. 105

6.7 The evolution of the number of the Context Agents during an experiment with a *virtual* tutor. 105

6.8 The evolution of the confidence of the Context Agents. 106

6.9 The evolution of the minimum, the maximum and the average confidence of the Context Agents. 106

6.10 The evolution of the size of the validity range associated to the *Velocity* value for each Context Agent. 107

6.11 The 2D projection of the policy learnt by ALEX with a human tutor. 107

6.12 The evolution of the number of Context Agents during an experiment with a *human* tutor. 108

6.13 The 2D projection of another policy learnt by ALEX with the same human tutor following a different strategy. 108

6.14 Objectives of the Teach Robots Yourselves Experiment 111

6.15 The experiment in the Webots simulator. On the left, the rover inside the arena. On the right, the camera detection. 111

6.16 Description of the vector of observations 112

6.17 Interaction between the entities involved in the Teach Robot Yourselves experiment. 114

6.18 Rover's trajectory and *Context Agents* creation during a demonstration 115

6.19 On the top, the validity domain structure at the creation of a *Context agent*. On the bottom, the same *Context agent* at the end of the demonstration. 116

6.20 Number of collected boxes each 5 minutes. The step 0 corresponds to the reference score. 117

6.21 The blue histogram shows the number of steps where the action is handled by the ALEX instances for each collected boxes. The orange histogram shows the number of steps where the Tutor has acted for each collected boxes. 119

6.22 Objectives of the Incremental Design Experiment 120

6.23 A view of the experiment. The rover evolves in an arena and has to reach the white door. 120
6.24 The core of the rover architecture. Each ALEX controller is responsible of the speed of one wheel. 122
6.25 Architecture of the first experiment. Each ALEX receives the distance value and has to associate to this value the adequate speed. 123
6.26 A comparison of two *Context Agents* extracted from the first experiment. The two *Context Agents* propose a different action under the same context leading to ambiguity. 124
6.27 Synthesis of the results after the first experiment. 124
6.28 The revised architecture of the first experiment used in the second experiment. Now, each ALEX instance receives the distance and the current speed of both wheels. 125
6.29 Synthesis of the results after the second experiment. 125
6.30 The third experiment architecture. A camera is added providing three new couples of values (x,y) for each detected color. 126
6.31 The structure of a particular *Context Agent* involved in the third experiment. This *Context Agent* is a lot more sensitive to the White (x,y) value than the other. 126
6.32 Synthesis of the results after the third experiment. 127
6.33 The architecture of the last experiment. The door is now controlled by an ALEX instance and receives the same information than the other ALEX. . . . 127
6.34 Synthesis of the results after the last experiment. 128
6.35 Objectives of the Collaborative Robotic Arm Thought Experiment 129
6.36 ALEX is coupled with its environment from which it receives observations and on which it acts. 129