Discovering and exploiting the task hierarchy to learn sequences of motor policies for a strategic and interactive robot

Nicolas Duminy

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

Université Bretagne Sud

THÈSE DE DOCTORAT DE

L'UNIVERSITE BRETAGNE SUD
COMUE UNIVERSITÉ BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *INFORMATIQUE*

Par **Nicolas DUMINY**

« Découverte et exploitation de la hiérarchie des tâches pour apprendre des séquences de politiques motrices par un robot stratégique et interactif »

« Discovering and exploiting the task hierarchy to learn sequences of motor policies for a strategic and interactive robot"

Thèse présentée et soutenue à IMT ATLANTIQUE Campus de Brest le 18/12/2018
Unité de recherche : Lab-STICC UMR CNRS 6285
Thèse N° : 513

Rapporteurs avant soutenance :

François CHARPILLET	Directeur de Recherche	Inria	Nancy
Manuel LOPES	Maître de Conférences	Instituto Superior Tecnico	Lisboa

Composition du Jury :

Président :	Pierre DE LOOR	Professeur des Universités	ENIB	Brest
	François CHARPILLET	Directeur de Recherche	Inria	Nancy
	Manuel LOPES	Maître de Conférences	Instituto Superior Tecnico	Lisboa
Examinateurs :	Sylvie PESTY	Professeure des Universités	Université de Grenoble IMAG	Grenoble
	Sao Mai NGUYEN	Maître de Conférences	IMT Atlantique	Brest
	François CHARPILLET	Directeur de Recherche	Inria	Nancy
	Manuel LOPES	Maître de Conférences	Instituto Superior Tecnico	Lisboa
Dir. de thèse :	Dominique DUAHUT	Professeur des Universités	Université Bretagne Sud	Lorient

Abstract

Discovering and exploiting the task hierarchy to learn sequences of motor policies for a strategic and interactive robot

Efforts have been made to enable robots to operate more and more in complex unbounded ever-changing environments, alongside or even in cooperation with humans. Their tasks can be of various kinds, can be hierarchically organized, and can also change dramatically or be created, after the robot deployment. Therefore, those robots must be able to continuously learn new skills, in an unbounded, stochastic and high-dimensional space. Such environment is impossible to be completely explored during the robot's lifetime, therefore it must be able to organize its exploration and decide what is more important to learn and how to learn it. This becomes an even bigger challenge, when the robot is faced with tasks of various complexities, some requiring a simple action to be achieved, others needing a sequence of actions to be performed. How to learn is the question of which learning strategy the robot decides to use in order to learn a particular task. Those strategies can be of two different kinds: autonomous exploration of the environment, where the robot relies on itself and its own database to try achieving a task at best, and interactive strategies, where the robot relies on human experts to demonstrate how to achieve the task. As some strategies perform differently depending on the task at hand, the choice of both what task to learn and which data-collection strategy to use is connected, and a method used to make this choice is called intrinsic motivation. The learner is guided towards the interesting parts of the environment to learn the most interesting skills. It is capable to assess the complexity of the action needed to achieve a task. When faced with hierarchically organized tasks of different complexity which can be fulfilled by combinations of simplier tasks, the robot finds a new way to get knowledge by exploring the task hierarchy itself, combining skills in a goal-oriented way so as to build new more complex ones.

Starting from the study of a socially guided intrinsically motivated learner learning simple tasks, actively deciding what task to learn and which strategy to use between imitation of a human teacher and autonomous exploration, I extended this algorithm to enable it to learn sequences of actions, discover and exploit the task hierarchy. I ended up extending a more generic learning architecture, able to tackle this problem, called Socially Guided Intrinsic Motivation (SGIM), adapting it to this new challenge of learning complex hierarchical tasks using sequences of actions. I call the extended architecture **Socially Guided Intrinsic Motivation for Sequences of Actions through Hierarchical Tasks (SGIM-SAHT)**.

This SGIM-SAHT learner is able to actively choose which kind of strategy to use between autonomous exploration and interactive strategies and which task to focus on. It can also discover the task hierarchy, and decide when it is most appropriate to exploit this task hierarchy and combine previously learned skills together. It is also capable to adapt the size of its action sequence to the task at hand. In this manuscript, I will present different implementations of this architecture, which were developed incrementally up to the complete generic architecture.

This architecture is able to learn skills by mapping a motion it has done, known as an *action* or *policy*, to the consequence observed on the environment known as an *outcome*. Developing this architecture enabled to make contributions in:

- Learning sequences of motor actions: I propose a framework, called the *procedure framework*, developed to enable a strategic learner to discover and exploit the task hierarchy, by combining previously known actions in a task-oriented manner. I also enable the learner to perform sequences of actions of any size, which enables it to adapt their size to the task at hand;
- Interactive learning: I developed a new way for a human teacher to provide demonstrations to a robot, which is by using the task hierarchy via our procedure framework. I compared which kind of demonstrations, procedures or actions, is most appropriate for what kind of task, simple or complex and highly-hierarchical;
- Active learning: I introduced the procedure space, which can be explored and optimized by a strategic and intrinsically motivated learner. Now our learner can decide in function of the task at hand and depending on its own maturity, which space to use between procedure and actions;
- Strategic learning: using the same learning architecture, I tested its ability to handle a high variety of strategies and outcome spaces. The architecture was indeed able to organize its learning process despite such combined numbers of strategies and outcome spaces.

This thesis is organized as follows. In Chapter 1, I define our computational framework, taking the cognitive developmental perspective. This field allows the elaboration of very effective learning architecture by implementing theories taken from developmental psychology on a robotic platform, which also enables to test those theories. In this context, I formalize our learning architecture in Chapter 2, SGIM-SAHT, which extends the existing SGIM one to learning sequences of motor primitives for hierarchical tasks. In the next chapters, I develop new implementations of this architecture, tackling increasingly more complex problems. In Chapter 3, I present a basic implementation of this architecture, called Socially Guided Intrinsic Motivation with Active Choice of Teacher and Strategy for Cumulative Learning (SGIM-ACTSCL), and see how it can tackle the learning of multiple tasks hierarchically organized using simple actions only by testing it on the humanoid robot Poppy. It can actively decide what task to learn and how to learn it, either building actions autonomously or requesting a teacher for demonstrations. In Chapter 4, I tackle the learning of sequences of actions to achieve multiple tasks of various complexity, by discovering and exploiting this task hierarchy thanks to a new framework I introduced: the procedure framework, which allow the combination of previously known primitive motor sequences in a task-oriented way. This leads to the development of two algorithms. The former, called Intrinsically Motivated Procedure Babbling (IM-PB), enables to test if this task hierarchy can be autonomously explored alongside the autonomous exploration of actions. The latter, called Socially Guided Intrinsic Motivation with Procedure babbling (SGIM-PB), lets us test if this autonomous exploration of the task hierarchy and the action space can be bootstrapped by human teachers providing demonstrations. I test both implementations on a purely simulated environmental setup. Then in Chapter 5, I test the SGIM-PB algorithm on a physical setup featuring the Yumi industrial robot learning sequences of actions in a hierarchical environment. This test was first performed on simulation, then confirmed on the actual physical robot. I also tried to determine if the task hierarchy can be transferred between two different robots learning in the same environment. Finally, Chapter 6 concludes the thesis, focusing on the achievements, the limitations and perspectives of this study.

Résumé

Découverte et exploitation de la hiérarchie des tâches pour apprendre des séquences de politiques motrices par un robot stratégique et interactif

Des efforts sont réalisés pour permettre à des robots d'opérer dans des environnements complexes, non bornés, évoluant en permanence, au milieu ou même en coopération avec des humains. Leurs tâches peuvent être de types variés, hiérarchiques, et peuvent subir des changements radicaux ou même être créées après le déploiement du robot. Ainsi, ces robots doivent être capable d'apprendre en continu de nouvelles compétences, dans un espace non-borné, stochastique et à haute dimensionnalité. Ce type d'environnement ne peut pas être explorer en totalité durant la durée de fonctionnement du robot, il doit donc être capable d'organiser son exploration et de décider ce qui est le plus important à apprendre ainsi que la méthode d'apprentissage. Ceci devient encore plus difficile lorsque le robot est face à des tâches avec des complexités variables, certaines demandant une action simple pour être réalisée, d'autre demandant une séquence d'actions. Parler de méthode d'apprentissage signifie que le robot doit choisir une stratégie d'apprentissage adaptée à la tâche en cours. Ces stratégies peuvent être de deux catégories: autonomes, quand le robot se débrouille pour réaliser sa tâche au mieux en fonction de sa base de données collectées durant son apprentissage ou interactives, quand le robot demande des démonstrations. Comme certaines stratégies performent différemment en fonction de la tâche à apprendre, le choix de quelle tâche apprendre et quelle stratégie utiliser est fait de manière combinée. Une méthode permettant de guider ce choix se nomme la motivation intrinsèque. Le robot est guidé vers les zones les plus intéressantes de son environnement afin d'apprendre les compétences les plus intéressantes. Il est capable d'évaluer la complexité de l'action nécessaire pour réaliser une tâche. Quand il fait face à des tâches hiéarchiques de différrentes complexités, qui peuvent être réalisées par une combinaison de tâches plus simples, le robot utilise une nouvelle manière d'acquérir des compétences en explorant la hiérarchie des tâches elle-même, en combinant ses compétences via une combinaison de tâches afin d'acquérir des nouvelles et plus complexes.

Je suis parti de l'étude d'un algorithme stratégique et interactif apprenant des actions simples, décidant activement sur quelle tâche se concentrer et quelle stratégie utiliser entre imitation d'un expert humain et exploration autonome. Je l'ai étendu afin de lui permettre d'apprendre des séquences d'actions, découvrant et exploitant la hiérarchie des tâches. J'ai fini par étendre une architecture d'apprentissage plus génerique dans ce but, appelée Socially Guided Intrinsic Motivation (SGIM), en l'adaptant à ce nouveau problème d'apprentissage de tâches hiérarchiques par des séquences d'actions. J'appelle cette architecture étendue, **Socially Guided Intrinsic Motivation for Sequences of Actions through Hierarchical Tasks (SGIM-SAHT)**.

Cette architecture SGIM-SAHT est capable de choisir activement quelle type de stratégie utiliser entre exploration autonome et stratégies interactives, ainsi que sur quelle tâche se concentrer. Elle peut également découvrir la hiérarchie des tâches, et décider quand il est plus approprié de l'exploiter et combiner des compétences précédemment acquises. L'architecture SGIM-SAHT est également capable d'adapter la longueur de ses séquences d'actions à la tâche en cours. Dans ce manuscrit, je vais présenter différentes implémentations de cette architecture complète et générique développées de manière incrémentale.

Cette architecture est capable d'apprendre des compétences en reliant les mouvements réalisés, appelés actions ou politiques, aux conséquences observées sur son environnement. En développant cette architecture, je réalise des contributions aux domaines de :

- Apprentissage de séquences d'actions motrices : je propose une infrastructure algorithmique appelée *procédure*, developpée pour permettre à un apprenant stratégique et interactif de découvrir et exploiter la hiérarchie des tâches, en combinant des actions connues en fonction des tâches. Je lui ai également permis de construire des séquences d'actions de n'importe quelle taille, lui permettant d'adapter cette taille à la tâche en cours d'étude;

- Apprentissage interactif : j'ai développé une nouvelle manière de fournir des démonstrations à un robot pour un expert humain, en utilisant la hiérarchie des tâches via les procédures. J'ai analysé quel type de démonstrations, procédures ou actions, est plus adapté à quel type de tâche, simple ou complexe et hiérarchique;

- Apprentissage actif : j'ai introduit l'espace procédural, qui peut être exploré et optimisé par un apprenant stratégique et intrinséquement motivé. Cet apprenant peut maintenant décider en fonction de la tâche travaillée et de la maturité de son apprentissage, quel espace utiliser entre celui des procédures et celui des actions;

- Apprentissage stratégique : en utilisant la même architecture d'apprentisage, j'ai testé sa capacité à gérer une grande variété de stratégies et d'espaces de conséquences. Cette architecture a en effet été capable d'organiser son apprentissage malgré cette grande combinaison de stratégies et d'espaces de conséquences.

Cette thèse est organisée de la manière suivante. Dans le chapitre 1, je définis mon infrastructure algorithmique, en prenant l'approche du développement cognitif. Cette approche permet l'élaboration d'architectures d'apprentissage très efficaces en appliquant les théories de la psychologie développementale sur une plateforme robotique, ce qui permet dans le même temps de tester ces théories. Dans ce contexte, je formalise mon infrastructure algorithmique d'apprentissage dans le chapitre 2, SGIM-SAHT, étendant l'architecture SGIM à l'apprentissage de séquences d'actions motrices pour des tâches hiérarchiques. Dans les chapitres suivants, je développe des nouvelles implémentations de cette architecture attaquant des problèmes de plus en plus complexes de manière incrémentale. Dans le chapitre 3, je présente une implémentation basique de cette architecture, appelée Socially Guided Intrinsic Motivation with Active Choice of Teacher and Strategy for Cumulative Learning (SGIM-ACTSCL), et vois comment elle peut apprendre plusieurs tâches hiérarchiques en utilisant des actions simples en la testant sur le robot humanoïde Poppy. Il peut décider activement quelle tâche apprendre et comment, soit en construisant des actions tout seul, soit en demandant des démonstrations à un expert humain. Dans le chapitre 4, je m'intéresse à l'apprentissage de séquences d'actions pour réaliser de multiples tâches de complexités différentes, en apprenant et exploitant la hiérarchie des tâches grâce à la nouvelle infrastructure algorithmique que nous introduisons : les procédures, qui permet la combinaison de séquences d'actions connues en fonction des effets de ces actions. Ceci mène au développement de deux algorithmes. Le premier, appelé Intrinsically Motivated Procedure Babbling (IM-PB), permet de

tester si cette hiérarchie des tâches peut être exploré de manière autonome en même temps que l'espace des actions motrices. Le second, appelé Socially Guided Intrinsic Motivation with Procedure babbling (SGIM-PB), me permet de voir si cette exploration autonome de la hiérarchie des tâches et de l'espace des actions complexes peut être accéléré par des experts humains fournissant des démonstrations. Je teste les deux implémentations sur un environnement purement simulé. Puis dans le chapitre 5, je teste l'algorithme SGIM-PB sur le robot industriel Yumi apprenant des tâches hiérarchiques avec des séquences d'actions motrices dans un environnement physique. Ce test fut d'abord réalisé en simulation, puis confirmé sur le robot réel. J'ai également essayé de déterminer si la hiérarchie des tâches peut être transféré entre deux robots apprenant dans un même environnement. Finalement, le chapitre 6 conclut cette thèse, en se concentrant sur ses contributions, ses limitations et ses perspectives.

Acknowledgements

I would like to thank the many people, who one way or the other contributed to my phD.

I would like to thank my thesis supervisor, Dominique DUHAUT, which helped me a lot both on the administrative requirements of the phD, and also by providing advices and questionings from a non-developmental roboticist perspective. He also provided me with the interesting Yumi industrial robot, which I was able to use for my experiments. Sao Mai NGUYEN was my principal guide and mentor during those past three years, on a day-to-day basis. She helped me a lot both to learn the work of a researcher, and to understand the basic knowledge necessary to tackle my research field. She also provided insights and guidance, throughout my thesis.

I would like to thank the IMT-Atlantique and its staff, for allowing me to stay in their facility for my research activity, as well as allowing me to teach to their students for a year, providing me with an interesting and exciting experience. I would also like to thank more specific people from IMT-Atlantique. Especially André THEPAUT and Sylvie KEROUEDAN, which introduced me 4 years ago to the sphere of research. I would like to thank Jérôme KERDREUX for its technical assistance, which helped me develop my different experiments. For their different remarks, advices and emulative discussions throughout my thesis, I would like to thank Maxime DEVANNE and Panagiotis PAPADAKIS.

I would like to thank all members of the Centre Européen de Réalité virtuelle (Cerv) from ENIB, and especially its director Ronan QUERREC, for allowing me to work in their facility. For his technical assistance, while inside the Cerv facility, I would like to thank Frédéric DEVILLERS for helping me get settled and use some of their computers to run simulations. I also thank Sébastien KUBICKI for allowing me to use their tangible interactive table, and helping me to use it for my experiments.

Two other phD students, Alexandre MANOURY and Sébastien FORESTIER, were making study in the same field as I, and their work was a great inspiration. I thank Sébastien for its work on tool-based exploration, which was the greater inspiration to develop the procedure framework, described in this thesis. I thank Alexandre for the collaborative work we performed during the last year, and the mutual advice we gave each other while comparing our apporach and algorithmic architecture, one of which gave birth to a co-published material.

Finally, I would like to thank, all the undergraduate students which worked in our team. Especially Junshuai ZHU, an engineering student which made an internship in our team, and helped me a lot developing for the Yumi industrial robot, and even worked on introducing transfer learning methods into my algorithms. I would like to thank David RANAIVOTSIMBA, Paloma BRY, Liz Angélica RAMOS MEDINA and Morten STABENAU, for their work on clustering and regression techniques, which will for sure lead to some major improvements of my algorithmic architecture in the near future.

Contents

Acknowledgements **xi**

1 Life-long learning of hierarchical tasks using sequences of motor primitives **1**
1.1 Life-long learning problem . . . 2
1.2 Learning methods . . . 4
1.2.1 Learning motor action sequences . . . 5
1.2.2 Multi-task learning by a hierarchical representation . . . 6
1.2.3 Active motor learning in high-dimensional spaces . . . 6
Intrinsic motivation . . . 7
Social guidance . . . 8
Strategic learning . . . 8

2 A strategic intrinsically motivated architecture for life-long learning **11**
2.1 Formalization of the problem . . . 11
2.2 Example of experimental setup: rotating robotic arm drawing . . . 13
2.3 Strategic Intrinsically Motivated learner . . . 14
2.3.1 SAGG-RIAC . . . 14
2.3.2 SGIM-ACTS . . . 15
2.3.3 Extension to complex tasks . . . 15
2.4 Socially Guided Intrinsic Motivation for Sequence of Actions through Hierarchical Tasks . . . 15
2.5 Tackling the experiment of the rotating robot arm drawing . . . 17
2.6 Conclusion . . . 18

3 Poppy humanoid robot learning inter-related tasks on a tactile tablet **19**
3.1 SGIM-ACTSCL . . . 19
3.1.1 Strategies . . . 20
Mimicry of an action teacher . . . 20
Autonomous exploration of the primitive action space . . . 21
3.1.2 Interest Mapping . . . 22
3.2 Experiment . . . 24
Description of environment . . . 24
Dynamic Movement Primitives . . . 25
Action space . . . 25
Observable spaces . . . 26
Task spaces . . . 26
3.2.1 The teacher . . . 26
3.2.2 Evaluation Protocol . . . 27
Evaluation method . . . 27
Compared algorithms . . . 28
3.2.3 Results . . . 28
Evaluation performance . . . 28

Learning process organization . . . 30
3.2.4 Conclusions . . . 31

4 Using the task hierarchy to form sequences of motor actions 33
4.1 Experimental setup . . . 33
4.1.1 Environment . . . 34
4.1.2 Formalization of tasks and actions . . . 35
Action spaces . . . 35
Outcome subspaces . . . 36
4.2 Procedures framework . . . 36
4.3 Intrinsically Motivated Procedure Babbling . . . 38
4.3.1 Strategies . . . 38
Autonomous exploration of the action space . . . 38
Autonomous exploration of the procedure space . . . 40
4.3.2 Overview . . . 40
4.3.3 Experiment . . . 42
Evaluation Method . . . 42
4.3.4 Results . . . 42
Evaluation performance . . . 42
Lengths of action sequences used . . . 43
4.3.5 Conclusion . . . 44
4.4 Socially Guided Intrinsic Motivation with Procedure Babbling . . . 45
4.4.1 Interactive strategies . . . 45
Action teachers . . . 45
Procedural teachers . . . 45
4.4.2 Algorithm overview . . . 46
4.4.3 Experiment . . . 46
Teachers . . . 46
Evaluation method . . . 47
4.4.4 Results . . . 48
Distance to goals . . . 48
Analysis of the sampling strategy chosen for each goal . . . 50
Length of the sequence of primitive actions . . . 51
4.4.5 Conclusion . . . 52

5 Yumi industrial robot learning complex hierarchical tasks on a tangible interactive table 55
5.1 Simulated experiment . . . 55
5.1.1 Setup . . . 55
5.1.2 Experiment variables . . . 56
Action spaces . . . 56
Task spaces . . . 57
5.1.3 The teachers . . . 58
5.1.4 Evaluation method . . . 59
5.1.5 Results . . . 59
Evaluation performance . . . 59
Analysis of the sampling strategy chosen for each goal . . . 60
Length of the sequence of primitive actions . . . 61
5.1.6 Conclusion . . . 62
5.2 Physical experimental setup . . . 63
5.2.1 Description of the environment . . . 63

5.2.2 Formalization of tasks and actions . . . 63
5.2.3 Teachers . . . 65
5.2.4 Evaluation method . . . 66
5.2.5 Results . . . 66
Evaluation performance . . . 66
Analysis of the sampling strategy chosen for each goal . . . 67
Length of actions chosen and task hierarchy discovered . . . 67
5.2.6 Conclusion . . . 69
5.3 Transfer learning . . . 70
5.3.1 Experimental setup . . . 70
5.3.2 Definition of the problem . . . 70
5.3.3 Transfer Learning in SGIM-PB . . . 71
5.3.4 Teachers . . . 72
5.3.5 Evaluation method . . . 72
5.3.6 Results . . . 72
5.3.7 Conclusion . . . 73

6 Conclusion **77**
6.1 Conclusion of the manuscript . . . 77
6.2 Conclusions and limitations . . . 78
6.2.1 Conclusions of the approach . . . 78
6.2.2 Limitations of the approach . . . 79
6.2.3 Perspectives . . . 80
6.3 Contributions . . . 81
6.4 Takeaway message . . . 81
6.5 Impact . . . 82
6.6 Papers . . . 82

Bibliography **83**

List of Figures

1.1 Illustration of a task hierarchy. To make a drawing between points (x_a, y_a) and (x_b, y_b), a robot can recruit subtasks consisting in (ω_i) moving the pen to (x_a, y_a), then (ω_j) moving the pen to (x_b, y_b). These subtasks will be completed respectively with actions π_i and π_j. Therefore to complete the complete this drawing, the learning agent can use the sequence of actions (π_i, π_j) 2

2.1 Representation of the learning problem: spaces are shown as well as example points and mappings 12
2.2 Rotating robotic arm drawing: simple typical simulated experimental setup tackable by a SGIM-SAHT learner: a planar robotic arm able to move its tip, grab and move a pen, and make drawings with this pen . 13
2.3 The SGIM-SAHT algorithmic architecture 16

3.1 Architecture of the SGIM-ACTSCL algorithm: number between brackets link parts of the architecture with lines in Algo. 1, the arrows show data transfer between the different blocks 20
3.2 Experimental setup: the Poppy robot is in front of a tactile tablet it will learn to interact with. The red arrows indicate the motors used. The green arrows represent the axes of the surface of the tablet. 24
3.3 24 demonstrations in the teacher dataset (blue circles). For each demonstration, the robot repeats 20 times exactly the same demonstrated movement. The outcomes reached (red crosses) are stochastic. Overall the stylus did not touch the tablet 126 times. 27
3.4 Evaluation datasets: 441 points for Ω_1, 625 points for Ω_2 and Ω_3 27
3.5 Strategies of the compared algorithms 28
3.6 Mean and variance error for reaching goal averaged on all task subspaces 29
3.7 Points M_{start} reached and histogram of the line length l drawn by Imitation, SGIM-ACTSCL and SAGG-RIAC 29
3.8 Evolution of the choice of learning strategy of SGIM-ACTSCL: percentage of times each strategy is chosen across time 30
3.9 Evolution of the choice of tasks of SGIM-ACTSCL: percentage of times each task is chosen across time 30
3.10 Synergy between the choice of task space and the choice of learning strategy of SGIM-ACTSCL: percentage of times each strategy and task is chosen over all the learning process 31

4.1 Experimental setup: a robotic arm, can interact with the different objects in its environment (a pen and two joysticks). Both joysticks enable to control a video-game character (represented in top-right corner). A grey floor limits its motions and can be drawn upon using the pen (a possible drawing is represented). 34

4.2 Task hierarchy represented in this experimental setup 37
4.3 Architecture of the IM-PB algorithm: number between brackets link parts of the architecture with lines in Algo. 1, the arrows show data transfer between the different blocks 41
4.4 Evaluation of all algorithms (standard deviation shown in caption) . . 43
4.5 Evaluation of all algorithms per outcome space (for Ω_0, all evaluations are superposed) 43
4.6 Number of actions selected per action size for three increasingly more complex outcome spaces by the IM-PB learner 44
4.7 Architecture of the SGIM-PB algorithm: number between brackets link parts of the architecture with lines in Algo. 1, the arrows show data transfer between the different blocks 46
4.8 Evaluation of all algorithms (final standard deviation shown in caption) 48
4.9 Evaluation of all algorithms per outcome space (for Ω_0, all evaluations are superposed) 49
4.10 Choices of teachers and target outcomes of the SGIM-PB learner 50
4.11 Number of actions selected per action size for three increasingly more complex outcome spaces by the SGIM-PB (on the left) and IM-PB (on the right) learners 51
4.12 Task hierarchy discovered by the SGIM-PB (left side) and IM-PB (right side) learners: this represents for each complex outcome space the percentage of time each procedural space would be chosen 53

5.1 Experimental setup for the Yumi simulated experiment 56
5.2 Representation of the interactive table: the first object is in blue, the second one in green, the produced sound is also represented in top left corner 57
5.3 Representation of task hierarchy of the simulated Yumi experimental setup 58
5.4 Evaluation of all algorithms throughout learning process, final standard deviations are given in the legend 60
5.5 Evaluation of all algorithms per outcome space (RandomAction and IM-PB are superposed on all evaluations except for Ω_0) 61
5.6 Choices of strategy and goal outcome for the SGIM-PB learner 61
5.7 Percentage of actions chosen per action size by the SGIM-PB learner for each outcome space 62
5.8 Real Yumi setup 64
5.9 Representation of task hierarchy of the real physical Yumi experimental setup 64
5.10 Global evaluation of the physical Yumi experiment 66
5.11 Evaluation for each outcome space of the physical Yumi experiment . 67
5.12 Number of choices of each interactive strategy and goal outcome space during the learning process 68
5.13 Percentage of actions chosen per action size by the SGIM-PB learner for each outcome space 68
5.14 Task hierarchy discovered by the SGIM-PB learner: this represents for each complex outcome space the percentage of time each procedural space would be chosen for the physical yumi experiment 69
5.15 Global evaluation of both learners 72
5.16 Evaluation for each task of both learners 73

5.17 Task hierarchy discovered by the learners compared to the transferred dataset (Transfer dataset on the left column, SGIM-PB in center one, SGIM-TL on right one): this represents for each complex outcome space the percentage of time each procedural space would be chosen for the simulated yumi experiment with transfer learning 74
5.18 Task hierarchy used by the learners during their learning process compared to the hierarchy discovered in the transferred dataset (Transfer dataset on the left column, SGIM-PB in center one, SGIM-TL on right one): this represents for each complex outcome space the percentage of time each procedural space is chosen for the simulated yumi experiment with transfer learning 75

List of Symbols

π_θ	a primitive action
Π	primitive action space
π	sequence of primitive actions
$\Pi^{\mathbb{N}}$	action space
ω	outcome, consequence of an action
Ω	outcome space
Ω_i	a specific part of the outcome space containing outcomes of same type
L	forward learning model, mapping actions to outcomes
L^{-1}	inverse learning model, mapping outcomes to actions
F	feature space, space of all outcomes and actions
l_f	feature sequence
σ	learning strategy
ω_g	goal outcome
Σ	ensemble of all available strategies
R_i	partitioned region of the outcome space
θ	motor parameters of a primitive action
π_{θ_d}, ω_d	primitive action and corresponding outcome from a teacher demonstration
M	an ensemble of actions or procedures with their corresponding reached outcomes
p_1, p_2, p_3	exploration mod probabilities of being selected
$(\omega_1, \omega_2, ..., \omega_n)$	sequence of outcomes or procedure
γ	meta parameter limiting the built action complexity, used in performance metric
α	constant controlling the distribution of sizes of random action sequences built

Chapter 1

Life-long learning of hierarchical tasks using sequences of motor primitives

Nowadays, robots are expected to be able to perform more and more daily tasks such as manipulating objects, possibly in cooperation with humans in an ever changing environment. The expectations of the robot tasks and skills are also to vary. In such a context, the robots can not possibly possess all the useful information prior to its deployment. To compensate for this prior lack of knowledge, robots should be able to **continuously learn how to interact with their environment**. One way to achieve this is to inspire from the way humans learn. The robot would learn to perform more and more complicated tasks, some of them being combinations of tasks themselves, or related to each other according to a hierarchy which is a representation of the dependency between the tasks (i.e. complex tasks can be described as combinations between simplier tasks). An example of such a hierarchy is shown on Fig.1.1 , it shows a task hierarchy from the experimental setup described in Section 4.1. Discovering and exploiting this hierarchy can help a learner combine previous skills in a task-oriented way to build new more complex skills. In order to learn how to achieve those tasks, a robot would need to explore its environment, to observe the effects of its actions and the relationships between them. It would then have different methods at its disposal: it can either count on itself and explore its environment autonomously, or count on human experts to help it explore by providing advice or guidance. I believe that the combinations of these abilities to autonomously explore such an environment, getting advice from human experts, along with the discovery and exploitation of the task hierarchy simplifies the adaptation of the robot to this complex ever changing environment. Indeed, the ability to self-explore enables the robot to adapt its knowledge base directly in its deployed working area, without requiring an engineer to hand-craft the modifications. Getting advice from human experts which are experts on the tasks the robot has to learn but not necessarily robotic experts, can bootstrap the learning process by focusing the robot on interesting motions and tasks quicker. Also, in a complex tasks environment, where tasks could be interrelated, meaning they have potential dependencies between them, discovering and exploiting the task hierarchy can enable the reuse of the skills learned for the simplest tasks in order to learn the more complex ones, without restarting from scratch.

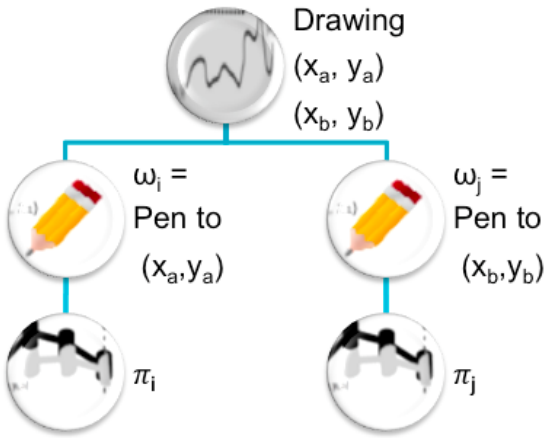

FIGURE 1.1: Illustration of a task hierarchy. To make a drawing between points (x_a, y_a) and (x_b, y_b), a robot can recruit subtasks consisting in (ω_i) moving the pen to (x_a, y_a), then (ω_j) moving the pen to (x_b, y_b). These subtasks will be completed respectively with actions π_i and π_j. Therefore to complete the complete this drawing, the learning agent can use the sequence of actions (π_i, π_j)

1.1 Life-long learning problem

Introducing robots into human environments to fulfil daily tasks makes their adaptivity necessary. As opposed to robots only environment, such as factories, human environments generally evolve and change all the time. So robots' actions can no longer be only programmed in advance prior to deployment. Also the prospect of increasingly deploying them among human users who are non-robotic experts, makes frequent engineer interventions to adapt the robots impractical. Thus, robots need to **continuously adapt to their changing and potentially open-ended environment, and also to the humans' ever changing needs, for its whole lifetime**. This corresponds to the definition of *continual life-long learning* (Thrun, 2012): a learning agent needs to continuously learn new skills, in an environment in potential different states, without reprogramming its behaviour by hand. Such an environment involves the learning of multiple tasks, which are impossible to test in totality because of the *curse of dimensionality*. This emphasizes the need to organize the learning process (i.e. actively decide which task to learn and how to learn it) in order to enable the robot to explore the tasks and learn as much of them as quickly as possible.

There are multiple challenges facing a robot tackling such a life-long learning problem:

- Stochasticity: when the robot performs an action on its environment, it isn't sure whether the same action repeated later will have the same consequences or outcomes. This is due to imprecisions in the robot's actuators or sensors, or even to changes occuring in the environment of the learner. As a consequence, the mapping between the actions and the outcomes produced can generally not be described by a simple function, but by a probability density.

- High-dimensionality: the sensorimotor space of those robots can contain a lot

of information, which overwhelms them at all time. The actions they can perform can be of potentially infinite dimensionality and the consequences observed can be of various types and dimensionality. The volume of those spaces grows exponentially as their dimensionality does. This problem is known as the curse of dimensionality as mentioned in (Baranes and Oudeyer, 2013).

- Unlearnability: there can be various regions in the sensorimotor space of the robot in which a predictive control model can not be learned by the robot, because of the environment and the robot's geometry. Those unlearnable regions are not known in advance and must be discovered by the learner.

- Unboundedness: the robot is in a situation, in which the number of associations between the possible outcomes and actions is infinite. As such, even during its whole lifetime, it would not have time to test any possible associations or even discover which associations are possible. This renders the existence of a mechanism to prioritize certain spaces instead of others necessary, as well as that of a metric to do this prioritization.

Life-long learning is also a problem that humans face, and particularly infants. Indeed, although humans possess certain skills at birth, they are nowhere near to what they will be able to perform in the later stages of their development. So human infants are faced with the same problems as a life-long learning robot. As such, **taking inspiration from the way human infants tackle their environment leads to the birth of *cognitive developmental robotics***. This approach uses the principles of *developmental learning* described in (Lungarella et al., 2003), the *action-perception loop* inferred in (Hari, 2006), *enactivism* as stated in (Varela, Thompson, and Rosch, 1991), and *trial and error* as observed in (Thorndike, 1898).

More precisely, I embrace the idea of a developmental approach, as described in (Asada et al., 2009) (Lungarella et al., 2003), indicating the learning process is progressive and incremental. As it is impossible to pre-program all the skills needed by a robot in a changeable environment, adaptation mechanisms are needed to continuously learn new skills. This approach is derived from observations on human infants in their early developmental stage in (Piaget, 1952). Indeed, newborn infants don't have the same level of abilities than adults, and only get those through a long and progressive period of maturation. Adults themselves are also able to adapt to changes in their environment or bodies, showing this developmental process is still ongoing.

I also consider the action-perception loop principle, that actions and perception are inter-related. The robot motions need to be guided by the robot perception. Also, the robot needs to move in order to perceive new situations. In my context, I consider an action-perception loop in which self-produced movements use perception information as a feedback to improve the learner knowledge. This principle is derived from studies on living beings, as (Held and Hein, 1963) showed on a cat that feeding it passive observation deprive it of its walking ability. (Hari, 2006) observed it by looking into human brain and concluded that we, humans, " shape our environment by our own actions and our environment shapes and stimulates us".

Moreover, I take the enactivist approach, introduced in (Varela, Thompson, and Rosch, 1991) which considers that cognition is based on situated and embodied agents. As such, its knowledge is gained and organized by interacting with its environment, and are thus dependent of the robot's body. Therefore, enactivism is based on the notion of embodiment Brooks (1991). In this approach, the robot must

perform actions with its own body in order to learn as its cognition is grounded on self-experience.

In this concept, I also take the trial-and-error approach, which states that the robot learns through repeated attempts of actions on its environment, until success or quitting. Both failures and successes help the robot improving its behaviour using its gained personal experience. This principle is directly derived from observations on animal behaviors, such as those performed on cats in (Thorndike, 1898), which when placed in a maze get better with experience for finding a way out.

Also, I aimed at a robot capable to perform various ranges of tasks of different complexities. This means that actions of various length or duration are needed to achieve those tasks. Therefore, I considered the definition of *primitive motor actions*, as the smallest quantity of motion doable by the robot. When combining multiple primitive actions together, the robot can perform a complex motor action, which is defined as a succession of primitive actions. Therefore, concerning the tasks, I consider complexity as the underlying complexity of the motor actions able to achieve them. Enabling the learner to build actions of various complexities emphasizes the problem of unlearnability, unboundedness and the curse of dimensionality. Indeed I would like the robot to be able to associate with a task a sequence of action of unbounded size. This means that I consider that the action space is of infinite dimensionality, rendering the number of possible actions also infinite. However, this approach enables the learner to **adapt the complexity of its actions to the task at hand**, which in a real world environment leads to a robot learning to be efficient in its actions. This also leads to a learning process, prioritizing the easiest and simplest tasks at first, before exploiting the task hierarchy and combine them to learn new ones.

Grounding my work on these principles and in order to tackle those challenges of stochasticity, high-dimensionality, unboundedness and unlearnability, I take inspiration from the approaches to the problem of **learning sequences of motor actions, multi-task learning by a hierarchical representation, active motor skill learning in high-dimensional spaces. For this latter, I focus on the concepts of *intrinsic motivation*, *social guidance* and *strategic learning*.** In the next section, I discuss such methods.

1.2 Learning methods

In this section, I discuss different methods which I got my inspiration from for tackling the life-long learning of complex motor actions problem. I also need to first formalize my view of the sensorimotor space, for this thesis. This view, is goal-oriented and derived from the action-perception loop principle. It differs from the state-action view traditionally used in reinforcement learning Sutton and Barto (1998), and is described in (Nguyen and Oudeyer, 2012).

The robot is faced with 3 different spaces, describing parts of the whole sensorimotor space:

- the *context space* describes all the possible states of the environment, prior to an action execution by the robot.
- the *action space* contains all actions the robot can attempt. Those actions are a parametrized encoding of the robot movement.

- the *outcome space* contains all effects on the environment observable by the robot. An outcome describes the change of state of the environment after a robot motion.

As I am tackling the learning of sequences of motor actions instead of only single primitives, I am increasing the exploration range of the learner. In this case, the context would need to be defined as the initial state of the environment prior to a primitive action execution. This introduces the idea of intermediate contexts which can be encountered during a motor sequence execution. However, as I already increase the dimensions to explore through enabling action sequences, and even combinations of tasks with the procedure framework described in section 4.2, **I decided to simplify my learning problem by ignoring the context.**

1.2.1 Learning motor action sequences

In this thesis, I tackle the learning of complex actions to complete high-level tasks. More concretely, in this study, I define the actions as a sequence of primitive actions. As I wish to get rid of any a priori on the maximum complexity of the action needed to complete any task, **the sequence of primitive actions can be unbounded**. The learning agent thus learns to associate to any outcome or effect on the world, an a priori unbounded sequence of primitive actions. I review in this paragraph works in compositionally of primitives from the robot learning perspective. The principle that motions are divided in motor primitives, that can be composed together after a maturation phase is derived from such observations in many species including primates and human beings (Giszter, 2015), showing evidences for the existence of motor primitives in skilled behaviours of childs and their re-use throughout adulthood.

A first approach to learning motor actions is to use *via-points* such as in (Stulp and Schaal, 2011; Reinhart, 2017) or *parametrised skills* such as in Silva, Konidaris, and Barto, 2012. The number of via-points or parameters is a way to define the level of complexity of the actions, but these works use a fixed and finite number of via-points. A small number of via-points can limit the complexity of the actions available to the learning agent, while a high number can increase the number of parameters to be learned. Another approach is to chain primitive actions into sequences of actions. However, this would increase the difficulty for the learner to tackle simpler tasks which would be reachable using less complex actions. Enabling the learner to decide autonomously the complexity of the action necessary to solve a task would allow the approach to be adaptive, and suitable to a greater number of problems.

Options (Sutton, Precup, and Singh, 1999; Machado, Bellemare, and Bowling, 2017) introduced in the reinforcement learning framework Sutton and Barto, 1998 offer temporally abstract actions to the learner. These options represent a temporal abstraction of actions as explained in Sutton, 2006. Chains of options have been proposed as extensions in order to reach a given target event. Learning simple skills and planning sequences of actions instead of learning a sequence directly has been shown to simplify the learning problem in Konidaris and Barto, 2009. They are a way to represent action probability density in a goal-oriented way. However, each option is built to reach one particular task and they have only been tested for discrete tasks and actions, in which a bounded number of options were used. I would like to **reuse this idea of temporal abstraction and goal-oriented representation to create unbounded action sequences**.

1.2.2 Multi-task learning by a hierarchical representation

Indeed, an essential component of autonomous, flexible and adaptive robots will be to exploit temporal abstractions, i.e. to treat complex tasks of extended duration, that is to treat complex tasks of extended duration (e.g. making a drawing) not as a single skill, but rather as a sequential combination of skills (e.g. grasping the pen, moving the pen to the initial position of the drawing, etc.) Such task decompositions drastically reduce the search space for planning and control, and are fundamental to making complex tasks amenable to learning. This idea can be traced back to the hypothesis posed in Elman, 1993 that the learning needs to be progressive and develop, *starting small*. It has been reintroduced as *curriculum learning* in Bengio et al., 2009, as formalised in terms of the order of the training dataset: the examples should not be randomly presented but organized in a meaningful order which illustrates gradually more concepts, and gradually more complex ones. For multi-task learning in the reinforcement learning framework, it has been studied as *hierarchical reinforcement learning* as introduced in Barto and Mahadevan, 2003, relying on task decomposition or task hierarchy.

Indeed, the relationships between tasks in task hierarchy Forestier and Oudeyer (2016) and Reinhart (2017) have been successfully exploited for learning tool use or learning inverse models for parameterized motion primitives, allowing the robot to reuse previously learned tasks to build more complex ones. As opposed to classical methods enabling robots to learn tool-use, as (Brown and Sammut, 2012) or (Schillaci, Hafner, and Lara, 2012), which consider tools as objects with affordances to learn using a symbolic representation, (Forestier and Oudeyer, 2016) does not necessitate this formalism and learns tool-use using simply parametrized skills, leveraging on a pre-defined task hierarchy. Barto, Konidaris, and Vigorito (2013) showed that building complex actions made of lower-level actions according to the task hierarchy can bootstrap exploration by reaching interesting outcomes more rapidly. Temporal abstraction has also proven to enhance the learning efficiency of a deep reinforcement learner in Kulkarni et al. (2016).

On a different approach (Arie et al., 2012) also showed composing primitive actions through observation of a human teacher enables a robot to build sequences of actions in order to perform object manipulation tasks. This approach relies on neuroscience modelling of mirror neuron systems. From the computational neuroscience point of view for sequence-learning task with trial-and- error, Hikosaka et al. (1999) suggested that procedural learning proceeds as a gradual transition from a spatial sequence to a motor, based on observations that the brain uses two parallel learning processes to learn action sequences: spatial sequence (goal-oriented, task space) mechanism and motor sequence (action space) mechanism. Each of the acquired motor sequences can also be used as an element of a more complex sequence.

I would like to extend these ideas of representations of tasks as temporal abstraction and as hierarchies, and to exploit the dual representation of tasks and actions sequences in this thesis. Instead of a pre-defined task hierarchy given by the programmer, my robot learner should be able to **learn hierarchical representations of its task space to more easily use acquired skills for higher-level tasks**.

1.2.3 Active motor learning in high-dimensional spaces

In order to learn sequences of primitive actions for multi-task learning, beyond the specific methods for learning sequences of actions and multi-task learning, I would

like to review the methods for learning high-dimensional mappings. More specifically, while the cited works above have outlined the importance of the organisation and order of the training data, I would like to examine how this organisation can be decided online by the robot learner during its learning process, instead of being left to the designer or programmer.

To address the challenge of multi-task motor learning, I will take the point of view of continual learning, also named life-long or curriculum Bengio et al., 2009 learning, that constructs a sophisticated understanding of the world from its own experience to apply previously learned knowledge and skills to new situation with more complex skills and knowledge. Humans and other biological species have this ability to learn continuously from experience and use these as the foundation for later learning. Reinforcement learning, as described in Sutton and Barto, 1998, has introduced in a framework for learning motor actions from experience by autonomous data sampling through exploration. However, classical techniques based on reinforcement learning such as Peters and Schaal, 2008; Stulp and Schaal, 2011 still need an engineer to manually design a reward function for each particular task, limiting their capability for multi-task learning.

Intrinsic motivation

More recent algorithms have tried to replace this manually defined reward function, and have proposed algorithms using intrinsic reward, using inspiration from *intrinsic motivation*, which is first described in developmental psychology as triggering curiosity in human beings Deci and Ryan, 1985 and has more recently been described in terms of neural mechanisms for information-seeking behaviours Gottlieb et al., 2013. This theory tries to explain our ability to learn continuously, although I do not have a clear tangible goal other than survival and reproduction, intrinsically motivated agents are still able to learn a wide variety of tasks and specialise in some tasks influenced by their environment and development, even in some tasks that are not directly useful for survival and reproduction. Psychological theories such as intrinsic motivation have tried to explain these apparently non-rewarding behaviours and have successfully inspired learning algorithms Oudeyer, Kaplan, and Hafner, 2007; Schmidhuber, 2010. More recently, **these algorithms have been applied for multi-task learning and have successfully driven the learner's exploration through goal-oriented exploration** as illustrated in Baranes and Oudeyer, 2010; Rolf, Steil, and Gienger, 2010. Santucci, Baldassarre, and Mirolli (2016) has also proposed a goal-discovering robotic architecture for intrisically-motivated learning to discover goals and learn corresponding actions, providing the number of goals is preset. Intrinsic motivation has also been coupled with deep reinforcement learning in (Colas, Sigaud, and Oudeyer, 2018) to solve sparse or deceptive reward problems to reach a single goal.

However for multi-task learning, especially when the dimension of the outcome space increases, these methods become less efficient (Baranes and Oudeyer, 2013) due to the curse of dimensionality, or when the reachable space of the robot is small compared to its environment. To enable robots to learn a wide range of tasks, and even an infinite number of tasks defined in a continuous space, heuristics such as social guidance can help by driving its exploration towards interesting and reachable space fast. Also, another approach combining introspection with intrinsic motivation described in (Merrick, 2012) enables a Reinforcement Learner to learn more complex goals by altering its strategy during its learning process.

Social guidance

Indeed, *imitation learning* Argall et al. (2009), Billard et al. (2007), and Schaal, Ijspeert, and Billard (2003) has proven very efficient for learning in high-dimensional space as demonstration can orient the learner towards efficient subspaces. Information could be provided to the robot using external reinforcement signals (Thomaz and Breazeal, 2008), actions (Grollman and Jenkins, 2010), advice operators (Argall, Browning, and Veloso, 2008), or disambiguation among actions (Chernova and Veloso, 2009). Furthermore, tutors' demonstrations can be combined with autonomous robot learning for more efficient exploration in the sensori-motor space. Initial human demonstrations have successfully initiated reinforcement learning in Muelling, Kober, and Peters, 2010; Reinhart, 2017. Nguyen, Baranes, and Oudeyer (2011) has combined demonstrations with intrinsic motivation throughout the learning process and shown that **autonomous exploration is bootstrapped by demonstrations**, enabling an agent to learn mappings in higher-dimensional spaces. Another advantage of introducing imitation learning techniques is to include non-robotic experts in the learning process (Chernova and Veloso, 2009).

Furthermore, tutor's guidance has been shown to be more efficient if the learner can actively request a human for help when needed instead of being passive, both from the learner or the teacher perspective (Cakmak, Chao, and Thomaz, 2010). (Melo, Guerra, and Lopes, 2018) showed that having a human teacher adapt to its student instead of imposing its demonstrations increases the teaching benefit for the learner. This approach is called interactive learning and it enables a learner to benefit from both local exploration and learning from demonstration. One of the key elements of these hybrid approaches is to choose when to request human information or learn in autonomy so as to diminish the teacher's attendance. The need for reducing the learner's calls of the teachers was identified in (Billard et al., 2007).

Strategic learning

This principle of a learner deciding on its learning process is generalised as *strategic learning*, as formalised in Lopes and Oudeyer (2012). Simple versions have enabled the learner to choose which task space to focus on (Baranes and Oudeyer, 2010), or change its strategy online (Baram, El-Yaniv, and Luz, 2004). In (Nguyen and Oudeyer, 2012), the algorithm SGIM-ACTS enabled the robot learner to **both choose its strategy and target outcome**. Owing to its ability to organize its learning process, by choosing actively both which strategy to use and which outcome to focus on. They have introduced the notion of *strategy* as a method of generating actions and outcome samples. This study considered 2 kinds of strategy: autonomous exploration driven by intrinsic motivation and imitation of one of the available human teachers. The SGIM-ACTS algorithm relies on the empirical evaluation of its learning progress. It showed its potential to learn on a real high dimensional robot a set of hierarchically organized tasks in (Duminy, Nguyen, and Duhaut, 2016). This is why I consider to extend SGIM-ACTS to learn to associate a large number of tasks to motor action sequences.

However, these works have considered an action space at fixed dimensionality, thus actions of bounded complexity. I would like to **extend these methods for unbounded sequences of motor primitives** and for larger outcome spaces.

Thus, in the following of this thesis, I propose a learning architecture to tackle the learning of multiple hierarchically organized continuous tasks in a real-life stochastic world, using motor primitive sequences of unconstrained size. Such approach

clearly needs to overcome the challenges of stochasticity, unlearnability and unboundedness of the robot environment. The ability to combine without limitations motor actions together will increase the challenge of high-dimensionality facing the robot. I ground my work on the field of cognitive developmental robotics and consider its principles when designing my learning architecture. I take a developmental approach, I consider actions and perceptions as linked, I take the enactivist approach and develop a learning agent increasing its knowledge through trial-and-error. More precisely, I combine multiple learning methods in my approach. I reuse the idea of temporal abstraction, and consider a learner able to form unbounded sequences of motor primitives. I implement a framework, called the procedures and described in Chapter 4, which represent the task hierarchy of the environment through sequences of tasks, leading to a combination of reused motor actions in a task-oriented way. I am using intrinsic motivation as a mean to guide the learning process of a strategic architecture, combining socially guided interactive strategies with autonomous ones, to tackle the learning of a set of multiple hierarchically organized tasks of various complexities. The teacher attendance is taken into account as my learning architecture can request help from human experts but is encouraged to rely on itself as much as possible.

In the Chapter 2, I formalize the learning problem and describe the **Socially Guided Intrinsic Motivation for Sequence of Actions through Hierarchical Tasks (SGIM-SAHT)** learning architecture, which I propose for solving it. In the next chapters, I describe various experiments in which I implemented different versions of this architecture. I start in Chapter 3 by testing the SGIM-SAHT architecture on an experiment with only simple motor actions but with a set of hierarchical tasks, designed to put my architecture to the test, before considering the learning of sequences of motor actions. In Chapter 4, I designed an experiment with a set of hierarchical tasks achieved using sequences of motor actions on which I test an autonomous learner which also implements the SGIM-SAHT architecture, and a more complete version adding social guidance. In Chapter 5, I designed another complex environmental setup using a physical real industrial robot. The SGIM-SAHT architecture is tested on a simulated version of the environment, and then on a physical real version of this environment. At the end of the chapter, I tested if procedures can be transferred to bootstrap the learning of another learning agent on the same environmental setup. Finally, I conclude this manuscript in Chapter 6.

Chapter 2

A strategic intrinsically motivated architecture for life-long learning

In this chapter, I formalize the learning problem within the developmental robotics scope and I describe my learning architecture, called the SGIM-SAHT architecture. This architecture is proposed for learning sequences of motor actions instead of single primitive actions. This architecture self-organizes its learning process using intrinsic motivation to decide at any time which outcome to focus on and which data-collection strategy to use. Those data-collection strategies include specific strategies classified as either social guidance or autonomous exploration and described in detail in the following chapters. However, this architecture does not in any way limit the field of available strategies to those two domains. It is to **tackle the learning of a field of hierarchically organized tasks using unbounded sequences of motor primitives in a continuous environment by discovering and exploiting the task hierarchy**.

2.1 Formalization of the problem

In my thesis, I am tackling the problem of enabling a robot to actively learn how to interact with its environment. Those interactions are made through the robot's sequences of actions and can have various observable consequences. The robot is to learn actively how to generate an ensemble of consequences as broad as possible, as fast as possible. It has access to sensory information, and know which features to attend to, corresponding to the possible outcomes it can observe. It initially only knows the dimensionalities and extended boundaries of the spaces of parametrized actions it can execute, and those of the multiple types of consequences it can observe. It knows neither its own geometry, nor any a-priori relationship between those spaces, nor the degree of difficulty of learning in each space.

So, in my approach, I consider that the learning agent is a robot, able to perform motions through the use of *primitive actions* $\pi_\theta \in \Pi$. We suppose that the primitive actions are parametrised functions with parameters of dimension n : we note the parameters $\theta \in \mathbb{R}^n$. Those primitive actions therefore correspond to the smallest unit of motion available to the learner. The robot is also able to perform *sequences of primitive actions* of any size $i \in \mathbb{N}$, by chaining multiple primitive actions together. Let us note the action π. Therefore, the complete space of actions available to the learner, is the ensemble of all sequence of action of any size. The space of complex action is thus $\Pi^{\mathbb{N}}$.

When the robot performs motions, the environment can change as a consequence of those motions. *Let us call outcomes* $\omega \in \Omega$ the consequences that are perceived by the robot. Those outcomes can be of various types and dimensionalities, and are

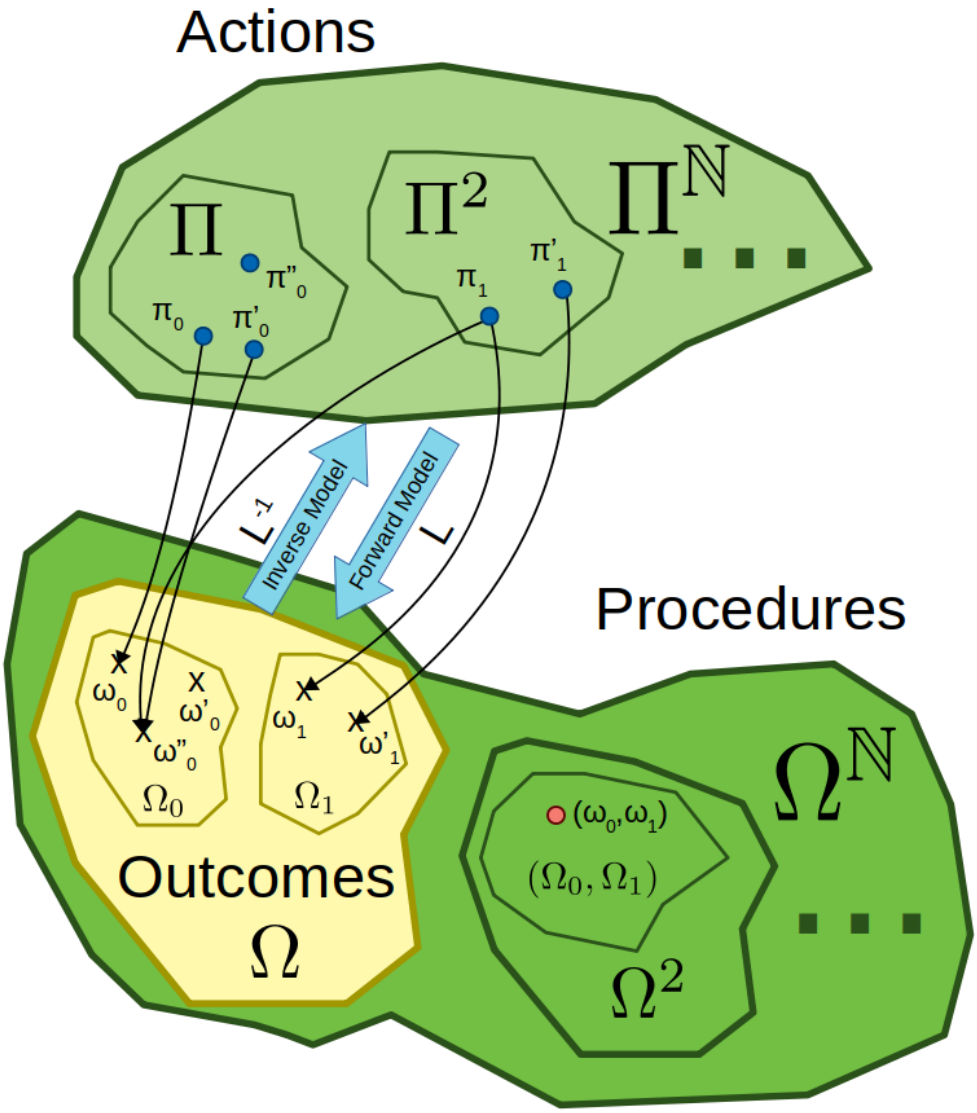

FIGURE 2.1: Representation of the learning problem: spaces are shown as well as example points and mappings

therefore split in outcome subspaces $\Omega_i \subset \Omega$. Those outcomes can also be of different complexities, meaning that the actions generating these outcomes may require different numbers of primitive actions to chain. The robot aims at knowing how to generate a range of outcomes as broad as possible, as fast as possible. It learns the mapping between the actions $\pi \in \Pi^{\mathbb{N}}$ it can perform and their outcomes $\omega \in \Omega$. This is known as the *forward model* L. More importantly, it learns which action to perform depending on the outcomes to generate. This is known as the *inverse model* L^{-1}.

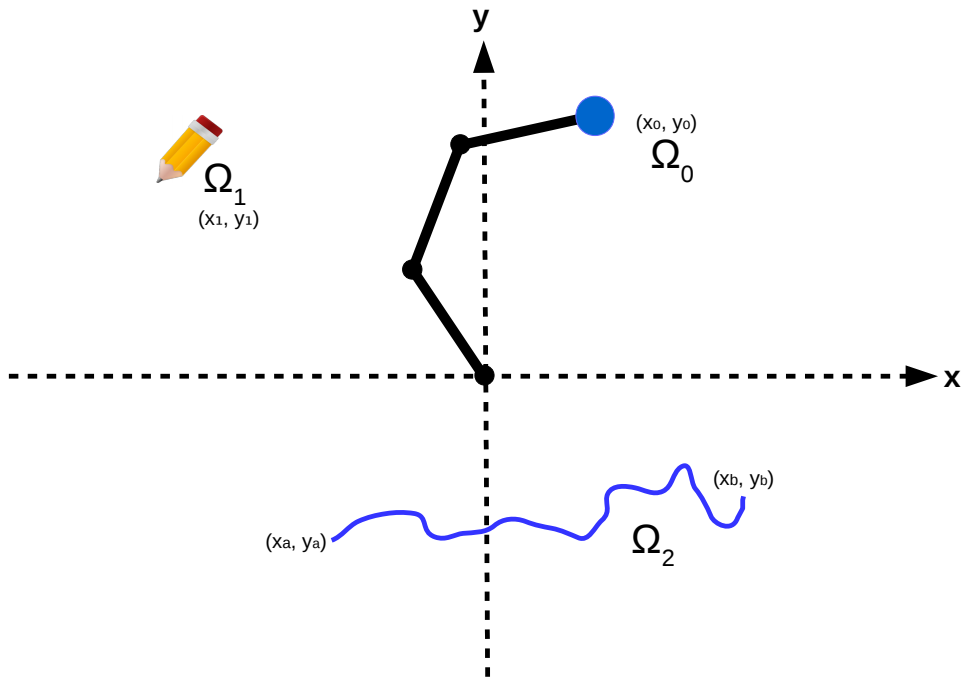

FIGURE 2.2: Rotating robotic arm drawing: simple typical simulated experimental setup tackable by a SGIM-SAHT learner: a planar robotic arm able to move its tip, grab and move a pen, and make drawings with this pen

2.2 Example of experimental setup: rotating robotic arm drawing

To illustrate the type of problems I have formalised, I describe in this section a theoretical setup featuring a multiple DOF robot learning multiple complex tasks hierarchically organized using sequences of motor actions.

Let us consider a planar robotic arm, consisting of 3 joints able to rotate without physical constraints. The robot controls its motions using sequences of motion primitives of any size. Around the robot lays a pen that can be grabbed by hovering the tip of the robotic arm close to it. After grabbing the pen by hovering close to it, the robot is able to draw on the 2D-surface. The drawing corresponds to the trajectory of the grabbed pen on the surface. The previous drawing is deleted when starting a new motion primitive. A representation of the complete setup is shown on Fig. 2.2.

Let us define actions as the trajectories of the robot's joint angles for a duration of 3s: α^0, α^1 and α^2 which are the respective angles of each joints, ordered from base to tip. Then, I define a primitive action as π_θ parametrized by $\theta = (\alpha^0_{t=1s}, \alpha^1_{t=1s}, \alpha^2_{t=1s}, \alpha^0_{t=2s}, \alpha^1_{t=2s}, \alpha^2_{t=2s}, \alpha^0_{t=3s}, \alpha^1_{t=3s}, \alpha^2_{t=3s}) \in \mathbb{R}^9$. Those actions are executed by interpolating the trajectories of each angle α^i linearly, from the pose at the beginning of the action to the each of the given poses. The robot is free to chain as many primitive actions as it wants. This beginning pose $(\alpha^0_{t=0s}, \alpha^1_{t=0s}, \alpha^2_{t=0s})$ is reset to the initial position of the robot after each action sequence tried, along with the whole environment (pen repositioned to initial position and drawing cleared). For the purpose of my thesis, this initial pose can be fixed to any position depending

on the experiment (not to any position resulting to the pen starting already in the robot's grasp though).

There are various observables available to the learner. First, the learner can detect the position of its arm's tip (x_0, y_0). When grabbed, the robot also detects the pen's position (x_1, y_1). In addition, the drawing is sensed by the robot through the following features: the first (x_a, y_a) and last (x_b, y_b) points of the drawing. There are 3 outcome subspaces in this setup: $\Omega_0 = \{(x_0, y_0)\}$, $\Omega_1 = \{(x_1, y_1)\}$ and $\Omega_2 = \{(x_a, y_a, x_b, y_b)\}$. The outcome space to tackle is therefore $\Omega = \Omega_0 \cup \Omega_1 \cup \Omega_2$.

The outcome subspaces are set this way as to be increasingly more difficult to learn and complex. They are also hierarchical, as moving the pen (Ω_1) means being able to move the arm's tip (Ω_0) from the pen initial pose to its desired end position, while making drawings (Ω_2) means moving the pen (Ω_1) to the first point of the desired drawing and moving the arm's tip (Ω_0) to the last point.

In this setup, the robot only knows the dimensionalities and boundaries of each outcome space alongside with that of the primitive action space. The robot is then to learn how to reach a range of outcomes as broad as possible, as quickly as possible, by building and testing sequences of actions.

In the next section, I describe the approach I followed to tackle such example of a learning problem. The architecture I developed, enables a robot to self-organize its learning process through the selection of the most adapted learning strategy for each task in a developmental manner. This architecture enables a learner to discover and exploit the hierarchical relationships between the tasks by combining skills in order to achieve more complex tasks.

2.3 Strategic Intrinsically Motivated learner

To tackle this problem of learning to reach fields of outcomes using sequences of motor actions, I consider the family of strategic learning algorithm. These strategic learners propose active learning architecture able to decide when, what and how to learn at any given time. When to learn refers to the time at which the agent will learn, what to learn refers to the outcomes to focus on, and how to learn refers to the method used to learn reaching that outcome called strategy. More particularly, I focused on the branch of intrinsically motivated algorithms that started by the Self-Adaptive Goal Generation - Robust Intelligent Adaptive Curiosity (SAGG-RIAC) algorithm.

2.3.1 SAGG-RIAC

This algorithm presented in Baranes and Oudeyer, 2010 focuses on the problem to help a learning agent to decide what outcome to focus on at any given time. It learns by episodes, where a goal outcome is generated based on the competence improvement recorded during the learning process. This goal outcome tends to be generated in areas where this competence improvement or progress is maximal. Then the algorithm performs the autonomous exploration of the action space which generates an action to reach that goal, based on its inverse model.

This algorithm was successfully used on high-dimensional robots that learned reaching whole outcome spaces, using primitive actions. It was also extended by the Socially Guided Intrinsic Motivation architecture (SGIM), to add new imitation strategies that the robot can use to bootstrap its learning process thanks to demonstrations provided by human teachers.

2.3.2 SGIM-ACTS

There are different variants of this SGIM architecture that have been tested, giving more and more control to the learning agent of its own learning process. Socially-Guided Intrinsic Motivation with Active Choice of Teacher and Strategies (SGIM-ACTS), developed in (Nguyen and Oudeyer, 2012), is the most advanced of them, in which the learning agent is able to actively choose both what goal outcome to learn and what strategy to use for that, between the autonomous exploration strategy developed by SAGG-RIAC to the strategies of imitating a specific teacher. This choice of both strategy and goal outcome is done at the same time depending on a measurement derived from the competence progress used by SAGG-RIAC called the interest metric. This metric introduces strategy costs so to encourage the learner to rely on autonomous exploration as much as possible.

This algorithm proved to be able to learn reaching whole outcome spaces, using primitive actions, more quickly and broadly than the SAGG-RIAC algorithm. It showed imitation strategies bootstrapped the early learning process of a robotic agent. Also, this agent was proved able to be able to organize its learning process, focusing on the easiest outcomes first, and identifying quickly the teachers' area of expertise. This is why I decided to extend this architecture to the problem of learning complex tasks hierarchically organized with sequences of actions.

2.3.3 Extension to complex tasks

As I tackle the learning of multiple tasks, of potential various types and complexity, I consider the use of sequences of motor actions and procedures, so as to profit from the underlying task hierarchy of the environment. This task hierarchy is unknown from the learner at the beginning and will be discovered during learning.

The actions and outcomes are grouped in a common ensemble called *features* $F = \Pi \cup \Omega$. I call feature sequences l_f action sequences if $l_f \in \Pi^{\mathbb{N}}$, and *procedures* if $l_f \in \Omega^{\mathbb{N}}$. Action sequences correspond to a succession of primitive actions that are chained together and executed, one after the other. Procedures are successions of outcomes, which represent how the learner exploits the task hierarchy to combine known skills. This idea of procedures, and how a specific SGIM-SAHT implementation uses them, is explained in section 4.2. Let us note that while procedures or other sequences combining outcomes can be used as internal representations by the learning agent, only action sequences can be executed on the environment.

Confined in those principles, I developed a generic learning architecture, using **strategic multi-task learning, guided by intrinsic motivation, to learn sequences of motor actions by potentially exploiting the task hierarchy** as I show in section 4.2. This architecture is called Socially Guided Intrinsic Motivation for Sequence of Actions through Hierarchical Tasks (SGIM-SAHT), and is described in the next section. This architecture is an extension and generalization of the SGIM-ACTS algorithm.

2.4 Socially Guided Intrinsic Motivation for Sequence of Actions through Hierarchical Tasks

The SGIM-SAHT architecture learns by episodes in which a goal outcome $\omega_g \in \Omega$ and a strategy $\sigma \in \Sigma$ have been selected.

The selected strategy σ is applied for the chosen goal outcome ω_g, and a feature sequence l_f is built to try reaching the goal.

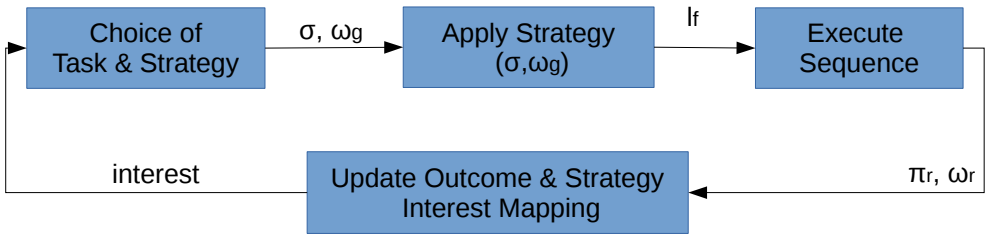

FIGURE 2.3: The SGIM-SAHT algorithmic architecture

This feature sequence l_f is broken down to a sequence of motor actions $\pi \in \Pi^{\mathbb{N}}$, before being executed by the robot to see its outcomes. The outcomes ω_r are then recorded, along with the tried actions and built observable sequence. This is important to note that the breakdown process is potentially recursive, and that each step of it is also recorded in the robot memory.

After each episode, the learner stores the executed actions and feature sequences, along with their reached outcomes in its episodic memory. Then, it computes its competence $competence(\omega_g)$ at reaching the goal ω_g, which depends on the euclidean distance between ω_g and the reached outcome ω_r. Its exact definition depends on the implementation. More importantly, the learner updates its interest map, by computing the interest $interest(\omega_g, \sigma)$ of the goal outcome for the used strategy. This interest depends on the progress measure $p(\omega_g)$ which is the derivative of the competence.

The learner then uses these interest measures to partition the outcome space Ω in regions R_i of high and low progress. In the beginning of the next episode, the learner chooses the strategy and goal outcome according to the updated interest map. The forward and inverse models consist local nearest neighbours based of the collection of all the actions attempted by the learner along with their reached outcomes. They are subsequently learned by adding more data, through more trials made by the learner on future episodes.

Algorithm 1 SGIM-SAHT architecture

Input: the different strategies $\sigma_1, ..., \sigma_n$
Initialization: partition of outcome spaces $R \leftarrow \bigsqcup_i \{\Omega_i\}$
Initialization: episodic memory $Memory \leftarrow \emptyset$
1: **loop**
2: $\omega_g, \sigma \leftarrow$ Select Goal Outcome and Strategy (R, H)
3: $l_f \leftarrow$ Execute Strategy(σ, ω_g)
4: $Memory \leftarrow$ Execute Sequence(l_f)
5: $R \leftarrow$ Update Outcome and Strategy Interest Mapping(R,$Memory$,ω_g)
6: **end loop**

The learner is provided with the outcome spaces boundaries (possibly larger than what is actually possible), the primitive action space boundaries, and distance metrics for both outcome spaces and action spaces. The strategies are also provided to the learner, although it does not know what they consist of. The robot will then learn the mapping between the outcome space and the action space, potentially relying on the unprovided task hierarchy which will also have to be discovered.

2.5 Tackling the experiment of the rotating robot arm drawing

If we test the SGIM-SAHT architecture on the experimental setup of the rotating robot arm drawing introduced in section 2.2, we can expect to see interesting patterns in the results showing its capabilities. This architecture shall learn faster and more broadly than simplier strategic algorithms like SAGG-RIAC or SGIM-ACTS.

If provided with different teachers and autonomous strategies, it shall be able to identify the most adapted combinations of strategies and tasks that optimize its learning accuracy and speed. It shall also be able to organize its learning process, so as to learn the easiest tasks first (in this case Ω_0), then tackling more and more complex tasks (Ω_1 and finally Ω_2) when maturing. It shall rely more heavily on interactive strategies early on to benefit from their observed bootstrapping effect on learning, while relying more on autonomous strategies on the long run. The evolution of the learner strategical decisions of what task and which strategy to use shall also be greatly influenced by the task hierarchy.

SGIM-SAHT shall be able to discover and use the task hierarchy wisely to learn faster. For example, using the task hierarchy would be useless to learn to move its tip (Ω_0), as it corresponds to the simplest task of the setup. Also, this task being the base of the task hierarchy, its learning shall be prioritary for the robot. However, hopefully, when desiring to move the pen (Ω_1), the robot shall use its skill at moving its arm's tip, as an intermediary to more easily learn, this being possible only when maturing the learning of the tip motion task (Ω_0). Finally when able to move the pen reliably, making drawings (Ω_2) shall be decomposed into a first skill corresponding to moving the pen position towards the first position of the drawing, then a second one moving the arm's tip towards the last position of the drawing. It is to note that the robot might also decide to combine two displacements of the pen as skills to do this same drawing, although it might lead to a less efficient learning. Indeed, after the first displacement of the pen, this latter is still in the robot's grasp, so it won't need to grab it again to move it afterwards, a simple arm motion will suffice. This could lead to two potential problems: the former is that when chaining two skills of pen displacement together the second one might have an unnecessary hovering by the initial pose of the pen, leading to more complex actions than needed, the latter is that the robot will necessary have learned more skills to displace its tip than to move the pen, so it would be able to tune a arm's tip motion to reach the final drawing position more accurately than a pen's one. This is an example of a potential suboptimal use of the task hierarchy by the learner. More extreme ones could be to use drawing skills to move the arm's tip.

Finally, SGIM-SAHT, while chaining multiple primitive actions together to move the pen or make drawings, shall be able to limit the complexity of this action sequences. Indeed, it would be suboptimal to use more complex actions than primitives to move the arm's tip around. Moving the pen's position shall be possible using primitives also or at least less than 2 primitives. Indeed, using the task hierarchy by reusing its skills in moving the arm's tip could ease the learning of the pen's motion one, at the result of more complex actions of 2 primitive actions. However, making a drawing shall not require more than the number of primitives used for moving the pen plus an additional one. The tradeoff between accuracy of the skills built and their efficiency in terms of number of actions chained can be tune by the γ parameter.

2.6 Conclusion

In this Chapter, I described a learning architecture, called SGIM-SAHT, able to self-organize its learning process by actively choosing which task to focus on and which data-collection strategy to use, along with a simulated experimental setup built to show the qualities of the architecture. **SGIM-SAHT is driven by intrinsic motivation towards the tasks leading to a maximal progress, and can build unbounded sequences of primitive actions**. Those action sequences can be built directly, according to the target task, or can be the result of the recursive process of replacing combinations of tasks by such an action sequence, this combination of tasks resulting of an exploitation of the observed task hierarchy of the environment. The strategies available to the learning architecture can be socially guided or autonomous, and their determination depend on the specific implementation. The next chapters describe 3 main implementations of this architecture, from the most simple one to the more complete version: SGIM-ACTSCL, described in Chapter 3, IM-PB, described in Chapter 4, and SGIM-PB, described in Chapter 4 and tested on real-life robot in Chapter 5. Their specific implementation details, along with the results they yield when experimented, are also contained in thir respective chapters.

Chapter 3

Poppy humanoid robot learning inter-related tasks on a tactile tablet

I present a first implementation of the SGIM-SAHT architecture in this chapter. The idea is to find out whether the SGIM-SAHT architecture can effectively learn in a continuous environment with a set of hierarchically organized tasks. Developing my architecture incrementally, I propose in this chapter a first version using simple actions only. The adaptation of the learning architecture to complex motor actions is the focus of the entire Chapter 4.

This implementation was built to tackle the **learning of a set of multiple inter-related and hierarchically organized tasks using only simple motor actions**. Those kinds of inter-related tasks can be triggered at the same time or even share parameters in their definition, which means they cannot be learned one at a time in isolation. The fact that they are hierarchically organized also means that one or more tasks consist in a combination of other tasks. This means that it is difficult for my SGIM-SAHT learner to organize its learning process in terms on which outcomes to focus on, as they are inter-related. Also the fact they are hierarchically organized means my learner shall be able to identify the basic tasks from the more complex ones, and start by learning the simplest ones to later tackles the most complex ones. To summarize, in this chapter, I want to see if the developed SGIM-SAHT implementation, called **SGIM-ACTSCL** described in section 3.1 can self-organize its learning process despite this challenge, and if it can learn better than each of its strategies taken alone. I conducted an experiment, designed to identify whether my SGIM-SAHT architecture seems appropriate for learning multiple hierarchically organized tasks due to its self-organizing learning process. The chapter is organized as follows: first I describe the algorithm I designed to learn the setup and then I show and analyse its results on my experimental setup. Both the developed algorithm and the experiment presented in this chapter, were published in (Duminy, Nguyen, and Duhaut, 2016).

3.1 SGIM-ACTSCL

SGIM-ACTSCL is a hierarchical algorithmic architecture that merges intrinsically motivated active exploration and interactive learning. The agent learns to achieve different types of outcomes by actively choosing which outcomes to focus on and set as goals, which data collection strategy to adopt and to which teacher to ask for help. It learns local inverse and forward models in complex, redundant and continuous spaces.

The SGIM-ACTSCL learner starts from scractch, it is only provided with the primitive action space and the outcome subspaces boundaries and dimensionalities. Its aim is to learn how to reach a set of outcomes as broad as possible, as fast as possible. It has therefore to **learn both what are the possible outcomes to reach and the primitive actions to use for that**. In order to learn, the agent can count on different learning strategies, which are methods to build a primitive action for any given target outcome. It also need to map the different regions of the outcome subspaces with the best suited strategies to learn them. The forward and inverse models consist only of the data collected during the learning process, which are the mappings between primitive actions and reached outcomes. So these are learned by adding new data in the memory of the learner.

This algorithm is an adaptation of SGIM-ACTS Nguyen and Oudeyer, 2012 for cumulative learning by sharing the observables produced during an episode between all task spaces to enhance the learning process. This enables other task spaces which have been reached too to take the most of the attempt (which is particularly useful when task spaces have dimension overlaps). The teachers were modified to enable them to give a demonstration close to the requested goal for each task space. Details about each module can be read in Nguyen and Oudeyer, 2012. The complete architecture is shown on Fig. 3.1.

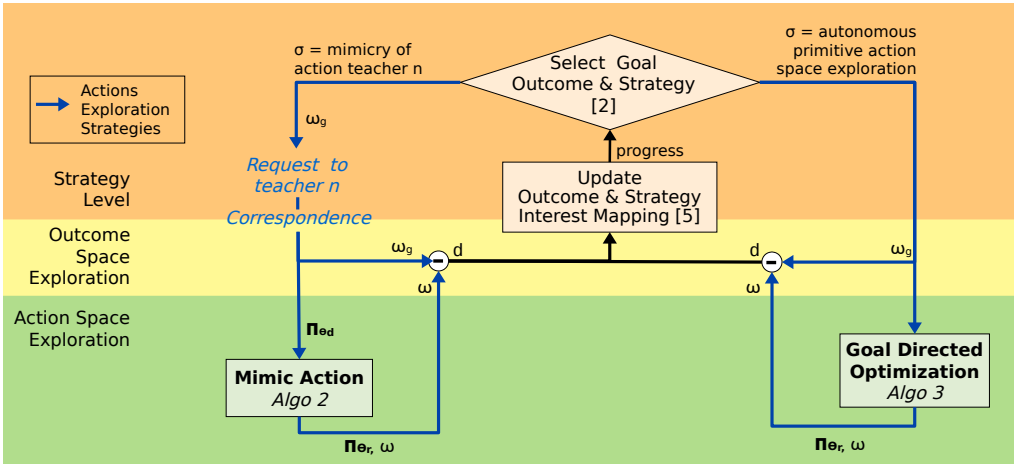

FIGURE 3.1: Architecture of the SGIM-ACTSCL algorithm: number between brackets link parts of the architecture with lines in Algo. 1, the arrows show data transfer between the different blocks

3.1.1 Strategies

SGIM-ACTSCL learns by episodes during which it actively chooses simultaneously an outcome $\omega_g \in \Omega$ to reach and a learning strategy. Its choice of strategy σ is selected between **autonomous exploration of the action space** and **mimicry of a specific action teacher**.

Mimicry of an action teacher

In an episode under the mimicry of an action teacher strategy (see Algo. 2), my SGIM-ACTSCL learner actively self-generates a goal ω_g where its competence improvement is maximal. The SGIM-ACTSCL learner explores preferentially goal outcomes easy to reach and where it makes progress the fastest. The selected teacher

answers its request with a demonstration $[\pi_{\theta_d}, \omega_d]$ to produce an outcome ω_d that is closest to ω_g (line 1 in Algo. 2). In the case of the present study, ω_d and ω_g can belong to different subspaces of the outcome space, and can be of different dimensionality. The robot mimics the teacher to reproduce π_{θ_d}, first exactly (line 2 in Algo. 2), then for a fixed duration, by performing actions π_θ which are small variations of an approximation of π_{θ_d} (lines 3-7 in Algo. 2). Indeed, the demonstration trajectory might be impossible for the learner to re-execute, because of correspondence problems and of the encoding of motor primitives. The variations on the demonstrated action are built by adding a uniform noise of maximum range ϵ (line 4 in Algo. 2).

Algorithm 2 Mimicry of an action teacher

Input: number of repetitions of the strategy $nbIm$
Input: the teacher demonstration repertoire $D = \{\pi_\theta, \omega\}$
Input: a target outcome ω_g
Input: noise added during repetitions ϵ
1: $[\pi_{\theta_d}, \omega_d] \leftarrow$ Nearest Neighbour(ω_g, D)
2: Execute action(π_{θ_d})
3: **for** $nbIm$ times **do**
4: $\quad \pi_{\theta_{rand}} \leftarrow$ random vector with $\mid \pi_{\theta_{rand}} \mid < \epsilon$
5: $\quad \pi_\theta \leftarrow \pi_{\theta_d} + \pi_{\theta_{rand}}$
6: $\quad$ Execute action(π_θ)
7: **end for**

The teacher's demonstrations repertoire are built in advance in practice for my experiments, by recording actions and their reached outcomes.

Autonomous exploration of the primitive action space

In an episode under the autonomous exploration of the primitive action space strategy, it explores autonomously following a method inspired by the SAGG-RIAC algorithm , which I call Goal-Directed Optimization in Fig. 3.1. This method works by iteration in which the learner first chooses how it explores the action space:

- Global exploration: the learner performs a random action without any regards to the actual goal outcome;
- Local exploration: the learner builds an action optimized for the specific goal outcome using the best local model.

The choice between both methods is done according to the learner's knowledge in the neighbourhood of the goal outcome. It tends to use global optimization when no close outcome is known (i.e. in the beginning of the learning process and when it explores remote region according to its current skill set) and use local exploration when more mature. The probability used to bias this choice is based on the sigmoid function, applied to the distance between the target outcome ω_g and its nearest-neighbours in the learner's dataset. This choice is referred in the algorithm as the choice of mode (line 3 in Algo. 3). The complete algorithm is written in Algo. 3.

The local exploration method (line 8 in Algo. 3) starts by determining the best local model for the outcome at hand. A function called best locality is used, as to select an ensemble of pairs of actions and outcomes $M = \{\pi_\theta, \omega\}$ (each action with its corresponding reached outcome). The pseudo-code for this method is shown in Algo. 4. Four different constants are needed when applying this algorithm: nbY

Algorithm 3 Autonomous exploration of the primitive action space

Input: a target outcome ω_g
Input: number of repetitions of the strategy $nbAuto$
for $nbAuto$ times **do**
  $Y \leftarrow$Nearest-Neighbours(ω_g)
  $mode \leftarrow$ mode *global-exploration* or *local-exploration*(Y, ω_g)
  if $mode =$Global exploration **then**
    $\pi_\theta \leftarrow$ Random action parameters
    Execute action(π_θ)
  else if $mode =$Local exploration **then**
    Local-optimization(ω_g)
  end if
end for

which gives the minimum number of nearest-neighbours of the target outcome ω_g to check, nbA which is the minimum number of nearest-neighbours checked for a target action, $distA$ and $distY$ corresponds to maximal distances accepted before rejecting respectively actions and outcomes. Multivariate linear regression is used inside this algorithm, computed using the normal equation method. Once a local model is selected, the learner uses the Nelder-Mead simplex algorithm to optimize the action. When this method is chosen by the learner, it is applied up to the end of the learning episode, as opposed to the global exploration which is only applied for one iteration (making the choice again afterwards).

An extensive study of the role of these different learning strategies can be found in Nguyen and Oudeyer, 2012. Thus the imitation exploration **increases** the learner's actions repertoire on which to build up self-exploration, while **biasing** the action space exploration to interesting subspaces, that allow the robot to **overcome** high-dimensionality and redundancy issues and **interpolate** to **generalise** in continuous outcome spaces. Self-exploration is essential to build up on these demonstrations to overcome correspondence problems and collect more data to acquire better precision according to the embodiment of the robot.

3.1.2 Interest Mapping

After each episode, the learner stores the actions executed along with their reached outcomes in its episodic memory. It computes its *competence* in reaching the goal outcome ω_g by computing the distance $d(\omega_r, \omega_g)$ with the outcome ω_r it actually reached. Then it updates its interest model by computing the interest $interest(\omega, \sigma)$ of the goal outcome and each outcome reached (including the outcome spaces reached but not targeted): $interest(\omega, \sigma) = p(\omega)/K(\sigma)$, where $K(\sigma)$ is the cost of the strategy used and the empirical progress $p(\omega)$ is the difference between the best competence before the attempt and the competence for the current attempt.

The learning agent then uses these interest measures to partition the outcome space Ω into regions of high and low interest (line 5 in Algo. 1). For each strategy σ, the outcomes reached and the goal are added to their partition region. Over a fixed number of measures of interest in the region, it is then partitioned into 2 subregions so as maximise the difference in interest between the 2 subregions. Each partition is done according to one dimension only. This dimension and the exact frontier value between both partitions is determined between all possible one-dimensional cuts

Algorithm 4 Best Locality

Input: constants $distA, distY, nbA, nbY$
Input: a target outcome ω_g
1: $H = \{\pi_\theta, \omega\} \leftarrow$ Nearest Neighbours(ω_g)
2: $K \leftarrow \varnothing$
3: $V \leftarrow \varnothing$
4: **for** each element H_i of H **do**
5: **if** $d(\omega_i, \omega_g) > distY$ **and** $i > nbY$ **then**
6: break
7: **end if**
8: $K_i = \{\pi_\theta, \omega\} \leftarrow$ Nearest Neighbours(π_{θ_i})
9: **for** each element K_{ij} of K_i **do**
10: **if** $d(\pi_{\theta_i}, \pi_{\theta_{ij}}) > distA$ **and** $j > nbA$ **then**
11: Remove K_{ij} and the following K_{ik} from K_i
12: **end if**
13: **end for**
14: $v \leftarrow 0$
15: **for** each element K_{ij} of K_i **do**
16: $M \leftarrow K_i \setminus K_ij$
17: $\pi_{\theta_g} \leftarrow$Linear-Regression($M, \omega_j$)
18: $v \leftarrow v + d(\pi_{\theta_g}, \pi_{\theta_{ij}})$
19: **end for**
20: $V_i \leftarrow v/size(K_i)$
21: **end for**
22: $k \leftarrow argmin(V)$
23: $M \leftarrow K_k$
24: **return** M

between two consecutive outcomes of the region (consecutive after sorting them according to the dimension currently studied). The method used is detailed in Nguyen and Oudeyer, 2014. Thus, **the learning agent discovers by itself how to organise its learning process and partition its task space into unreachable regions, easy regions and difficult regions, based on empirical measures of interest**.

The choice of strategy and goal outcome is based on the empirical progress measured in each region R_n of the outcome space Ω. ω_g, σ are chosen stochastically (with respectively probabilities p_1, p_2, p_3), by one of the sampling modes (line 2 in Algo. 1):

- mode 1: choose σ and $\omega_g \in \Omega$ at random;
- mode 2: choose an outcome region R_n and a strategy σ with a probability proportional to its interest value. Then generate $\omega_g \in R_n$ at random;
- mode 3: choose σ and R_n like in mode 2, but generate a goal $\omega_g \in R_n$ close to the outcome with the highest measure of progress.

In the beginning of the learning process , as the robot has no outcome and interest measure to guide this choice, the first mode doing random exploration is automatically selected. At this state, the partition regions consist of the whole outcome subspaces.

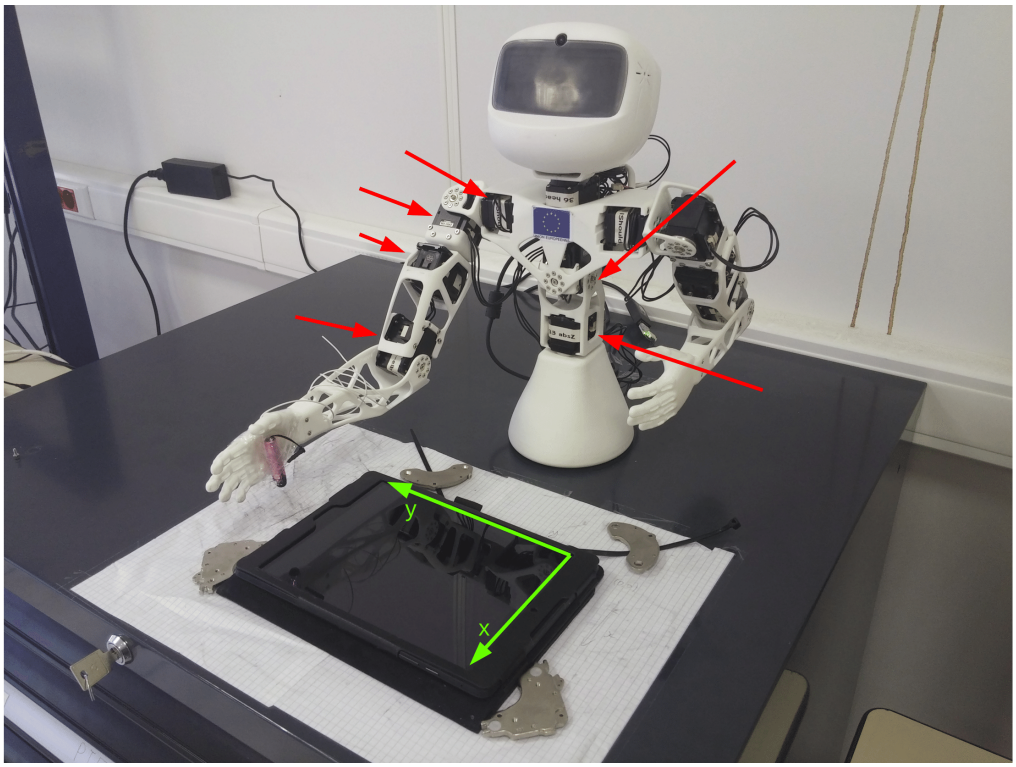

FIGURE 3.2: Experimental setup: the Poppy robot is in front of a tactile tablet it will learn to interact with. The red arrows indicate the motors used. The green arrows represent the axes of the surface of the tablet.

3.2 Experiment

I designed an experiment for a robot to learn to use a tactile tablet, namely to learn an infinite number of tasks, organised as 3 interrelated types of tasks.

I carried out my experiment on a real robot with a high number of dimensions for action and observable spaces. Testing the algorithm on a real platform adds the problem of stochasticity as the control of a real robot and the use of a real sensor (the tablet) add uncertainty. Fig. 3.3 shows that when repeating several times the same movement, the teacher's demonstration, the points sensed by the tablet are stochastic. I also decided to use the bio-inspired *Dynamic Movement Primitives* as my robot motion encoders.

This setup is designed to test the robustness of my SGIM-ACTSCL algorithm to self-organize its learning process when facing multiple inter-related set of tasks: correctly assessing the difficulty of each task and learn them incrementally, adapt its strategy to the task at hand despite their inter-relations, performs better than each strategy taken alone which proves this active choice of task and strategy is beneficial to the learner.

Description of environment

The learning agent of this experiment is a **Poppy torso robot** designed by the flowers team of INRIA Bordeaux and described in (Lapeyre, Rouanet, and Oudeyer, 2013). It is equipped with a tactile stylus on its right hand. Before him lays a 10" tactile

tablet, which it will learn to interact with, through the learning of 3 interrelated types of tasks described in subsection 3.2. Each of its actions produces observables of 5 dimensions (section 3.2).

The robot always starts an episode from the same position, as shown in Fig. 3.2. The learning algorithm gives an action to the robot controller to execute. Then the tablet senses the list of points touched and returns to the robot the observables.

In the next subsections, I formalize how I encoded the tasks and actions for this experiment.

Dynamic Movement Primitives

I encode my actions as discrete joint space motions using the dynamic movement primitives (DMP) framework Ijspeert, Nakanishi, and Schaal, 2002. This framework offers many advantages (robustness, temporal and spatial invariance, and guaranteed convergence to the goal) and is thus widely used in robotics.I here use the formulation developed in Pastor et al., 2009. Each one dimensional DMP is defined by the system:

$$\tau \dot{v} = K(g - x) - Dv - K(g - x_0)s + Kf \tag{3.1}$$

$$\tau \dot{x} = v \tag{3.2}$$

where x and v are the position and velocity of the system; x_0 and g are the starting and end position; τ is a factor used to temporally scale the system; K is like a spring constant; D is the damping term and f is a non-linear term used to shape the trajectory of the motion called the forcing term. It can be learned to fit a given trajectory using learning from demonstrations techniques Schaal, Atkeson, and Vijayakumar, 2002 and is defined as:

$$f(s) = \frac{\sum_i \omega_i \psi_i(s) s}{\sum_i \psi_i(s)} \tag{3.3}$$

where $\psi_i(s) = \exp(-h_i(s - c_i)^2)$, with centres c_i; widths h_i, and weights w_i. The function f does not depend directly on time but uses a phase variable s, which will start at 1 and decrease monotonically to 0 through the motion duration following the canonical system:

$$\tau \dot{s} = -\alpha s \tag{3.4}$$

The realization of multi-dimensional DMPs is feasible by using one transformation system per degree of freedom (DOF) which share a common canonical system, ensuring henceforth the synchronization of the different DOF throughout the motion. The learning of their forcing term can be done successively.

Action space

I selected 6 joints on the whole robot: the right arm, one joint to rotate the spine and one to bend forward (Fig. 3.2).

A 6-dimensional DMP is used to encode an action. The K, D and α parameters of eq. 3.1 are fixed for the whole experiment. The temporal scaling term τ of the DMP is shared for all the dimensions. The forcing term f_i of each transformation system is coded with 5 basis functions, which locations and widths are fixed for the whole experiment, leaving only their corresponding weights w_i to be parametrized. The end angle g_i of each joint is also a parameter but the starting pose is fixed, the robot

always starting from the same pose. Therefore an action π_θ is parametrized by:

$$\theta = (\tau, a_0, a_1, a_2, a_3, a_4, a_5) \in [0,1]^{37} \tag{3.5}$$

where $a_i = (g_i, w_{i,0}, w_{i,1}, w_{i,2}, w_{i,3}, w_{i,4})$ represents the parameters of joint i. The action space is thus $[0,1]^{37}$.

Observable spaces

The effects of the robot's actions are observed by the tablet which acts here as a sensor. The tablet sends the list of all points (x, y) touched by the robot at the end of the movement. Using this list, I considered the following observables:

- $M_{start} = (x_{start}, y_{start})$: the first position touched on the tablet by the learner during its attempt.
- $M_{end} = (x_{end}, y_{end})$: the last position touched on the tablet during its attempt.
- l: the length of the drawing on its whole attempt.

Task spaces

The tasks the agent will learn to master are normalised combinations of the previously defined observables: $\Omega_1 = \{M_{start}\} = [0,1]^2$, $\Omega_2 = \{M_{start}, M_{end}\} = [0,1]^4$ and $\Omega_3 = \{M_{start}, M_{end}, l\} = [0.1]^5$. I defined the task space as $\Omega = \Omega_1 \cup \Omega_2 \cup \Omega_3$.

These tasks have **various degrees of difficulty and some will depend on each other**. The idea beyond this choice of interdependent task spaces is to use tasks representing different levels of complexity (different combinations of observables) that the robot could explore progressively. The observables produced by an action are shared to improve the skill of the robot in all the tasks at once, without restricting them to the task space initially targeted by the action.

3.2.1 The teacher

For the experiment, I designed a teacher who has a demonstration dataset, recorded by kinaesthetic on the robot. The dataset consists of 24 demonstrations to touch points regularly distributed on the surface of the tablet which don't coincide with any test point of the evaluation testbench so to not give the learners using this teacher an unfair advantage. Each demonstration corresponds to outcomes where $M_{start} = M_{end}$. So he is an expert in tasks Ω_1 only. The points contained in the dataset of the teacher are shown by the blue circles in Fig. 3.3. The teacher gives a demonstration when requested for an outcome $\omega_g \in \Omega$ by the robot. For any ω_g in any subspace Ω_1, he chooses the demonstration (π_d, ω_d) which outcome ω_d is the closest to ω_g. His dataset has been built by kinaesthetic on the robot as to be capable of reaching a big proportion of the tablet.

Moreover, due to problems during the experiment, the dataset was built using a Poppy robot different from the one used in the learning phase. The differences in the joints offsets and robot's position introduce a correspondence problem. Fig. 3.3 shows a shift between demonstrations and repetitions by the robot of the demonstrated action.

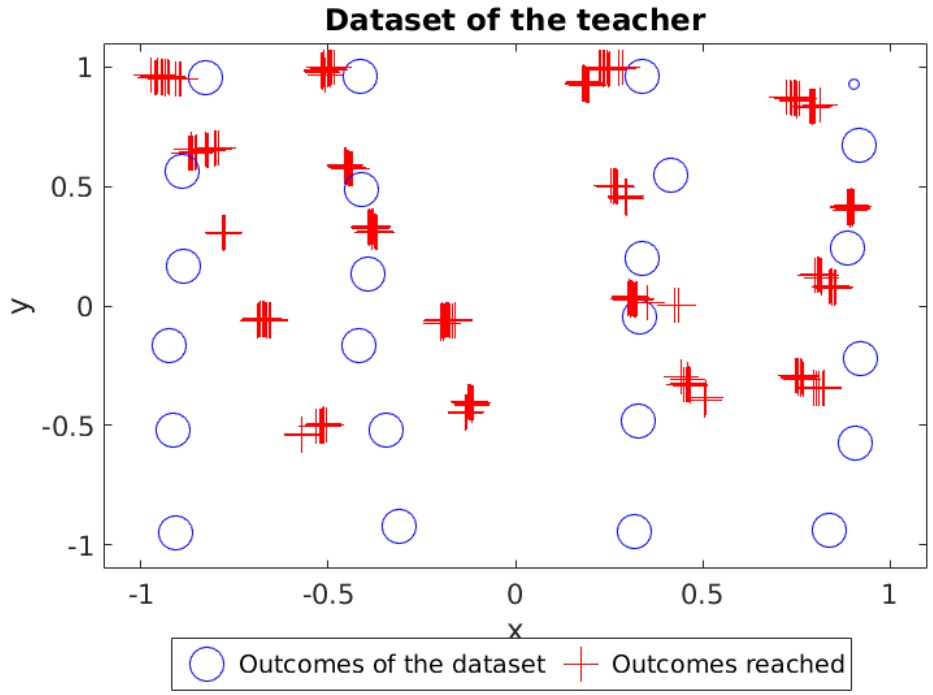

FIGURE 3.3: 24 demonstrations in the teacher dataset (blue circles). For each demonstration, the robot repeats 20 times exactly the same demonstrated movement. The outcomes reached (red crosses) are stochastic. Overall the stylus did not touch the tablet 126 times.

3.2.2 Evaluation Protocol

To assess the capacity of my SGIM-ACTSCL learner, I first need an evaluation method, and more importantly a metric, and I need other algorithms to compare my evaluation results with.

Evaluation method

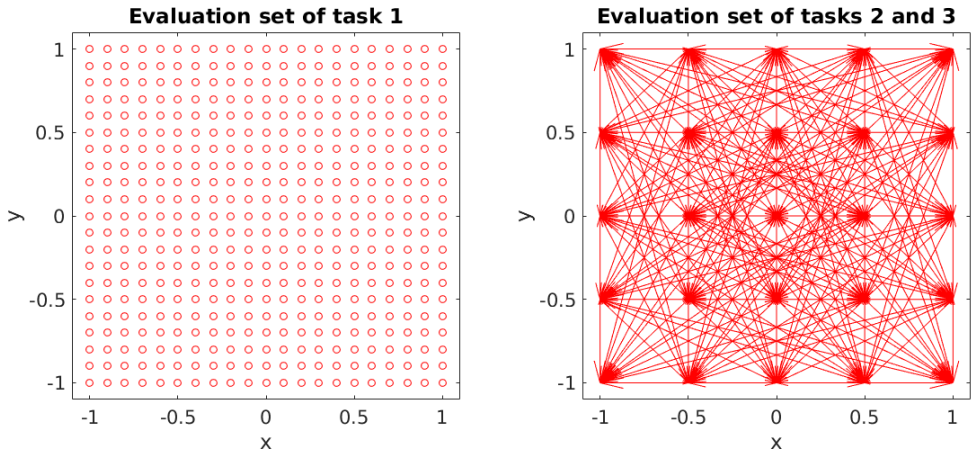

FIGURE 3.4: Evaluation datasets: 441 points for Ω_1, 625 points for Ω_2 and Ω_3

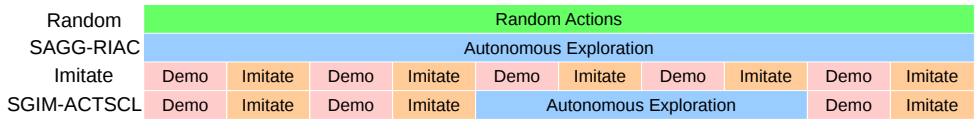

FIGURE 3.5: Strategies of the compared algorithms

In order to evaluate my algorithm, I define beforehand a benchmark dataset of outcomes: one set per outcome space or a total of 1691 points (Fig. 3.4). For the tesbench of Ω_1, a grid with cells of 0.1 was used. For Ω_2 and Ω_3, 25 points regularly distributed on the tablet surface were chosen and the tesbenches correspond to each possible straight lines between two of those points. The task space Ω_3 uses the same lines than Ω_2, except the line length l is added. This evaluation dataset is different from the teacher demonstrations, sharing no common outcomes.

To assess how well the robot can reach each of the outcomes of the evaluation dataset, I compute the closest reached outcomes. I plot the mean distance for predefined and regularly distributed timestamps. The evaluation is carried out while freezing the learning system. Its results have no impact on the learning process.

Compared algorithms

To check the efficiency of my SGIM-ACTSCL algorithm in this experimental setup, I compared with 3 other learning algorithms:

- Random exploration: the robot learns by executing random actions π from the action space.
- SAGG-RIAC: the learner autonomously explores its environment using goal-babbling without any teacher demonstrations and is driven by intrinsic motivation.
- Imitation: the learner requests a demonstration at a regular frequency, the demonstration given is among the less chosen ones. It is executed and repeated with small variations.
- SGIM-ACTSCL: interactive learning where the learner driven by intrinsic motivation chooses between autonomous exploration or imitation of the teacher.

Each run took an average of 3 days. The code for those algorithms is available at `https://bitbucket.org/smartan117/sgim_example`.

3.2.3 Results

Evaluation performance

Fig. 3.6 plots for the 4 exploration algorithms, the mean distance to outcomes of the evaluation set, through time obtained on those four experiments. It shows that **SGIM-ACTSCL outperforms the three others**. SAGIM-ACTSCL outperforms Random and SAGG-RIAC from the beginning. From $t > 1000$, it outperforms imitation, owing to goal-oriented self-exploration.

Fig. 3.7 analyses this difference, by plotting the outcomes reached by imitation, SGIM-ACTSCL and SAGG-RIAC. The first column shows that the outcomes in Ω_1

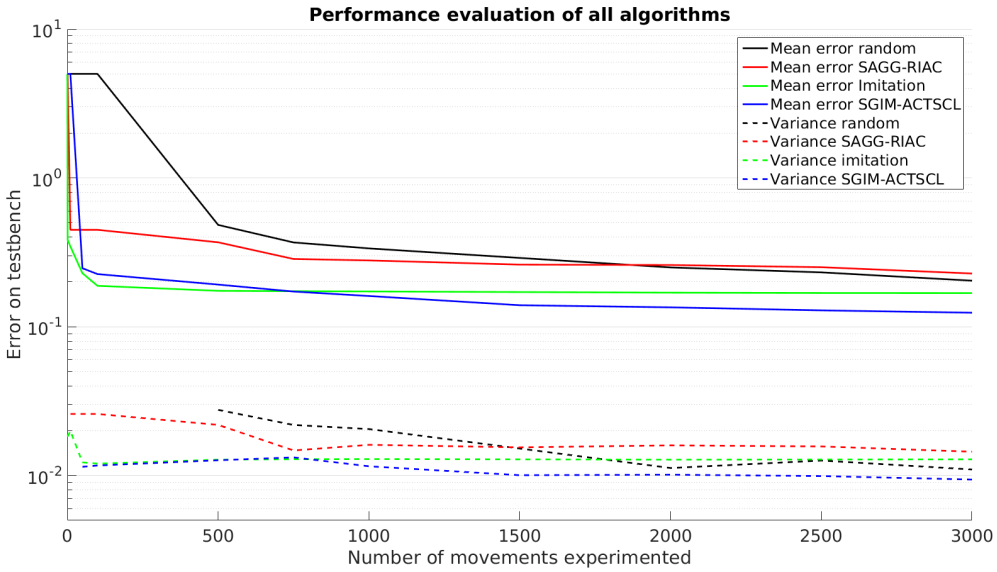

FIGURE 3.6: Mean and variance error for reaching goal averaged on all task subspaces

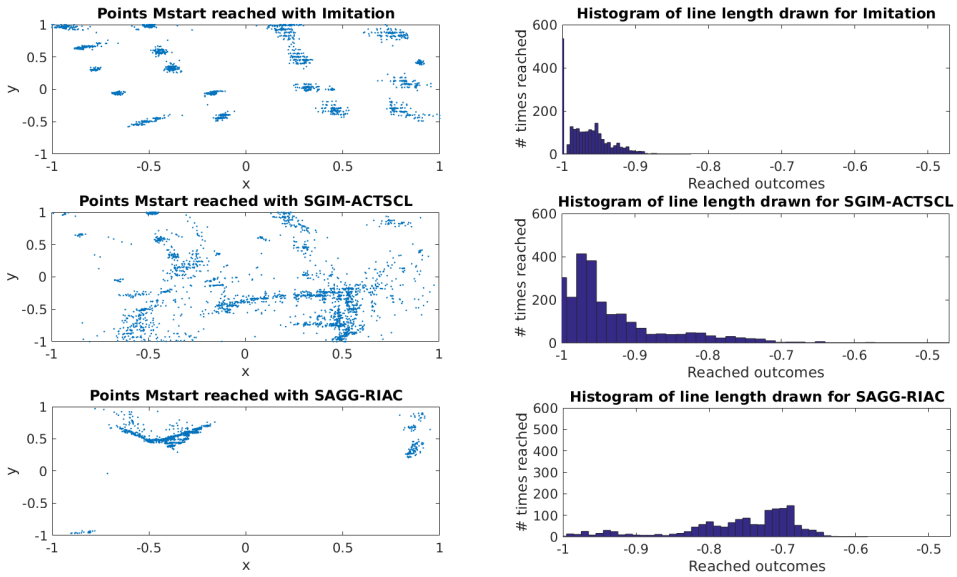

FIGURE 3.7: Points M_{start} reached and histogram of the line length l drawn by Imitation, SGIM-ACTSCL and SAGG-RIAC

reached by imitation are close to the demonstrations, whereas **SGIM-ACTSCL extended its exploration to cover a wider range of outcomes**. SAGG-RIAC explored intensively a smaller part of the tablet. Likewise, while demonstrations correspond to outcomes in Ω_3 with only length $l = 0$, the histograms in the second column shows that imitation could increase the length of its drawings a bit, while SGIM-ACTSCL and SAGG-RIAC could draw longer lines.

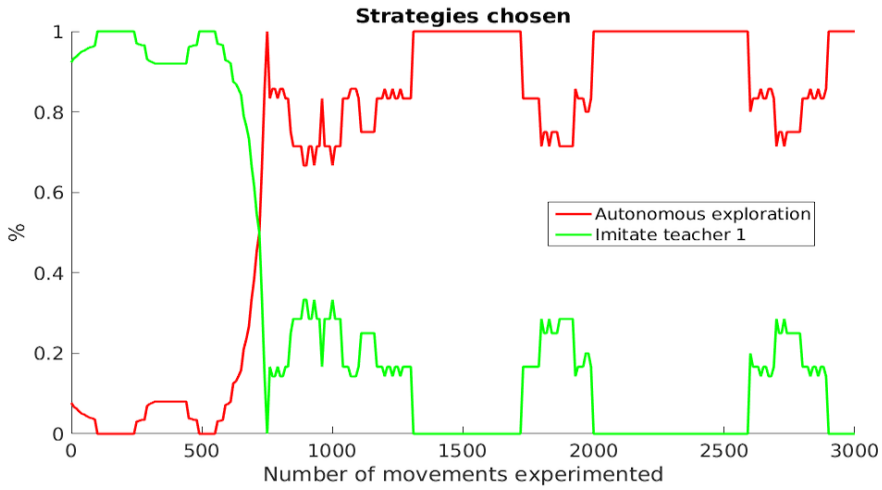

FIGURE 3.8: Evolution of the choice of learning strategy of SGIM-ACTSCL: percentage of times each strategy is chosen across time

Learning process organization

While SGIM-ACTSCL outperforms each of its strategy taken alone, I analyse how the SAGG-RIAC and imitation strategies were used by SGIM-ACTSCL through time. Fig 3.8 shows that in the beginning the robot takes advantage of the imitation strategy which overcomes the difficulty to reach the tablet at first. This difficulty is well shown by the Random algorithm results which only touched the tablet 14 times on the 3000 attempts. Imitation strategy enables it to outperform the self-exploration algorithms, but not the imitation algorithm as the latter was repeating each demonstration equally while the former was not (the demonstrations were chosen according to the robot curiosity). After more than 700 attempts using the imitation strategy, the robot had reproduced most of the teacher demonstrations and changed its strategy to keep progressing. As the teacher was only able to produce points, the learner chose the autonomous exploration strategy which enabled him to reach points farther and farther from the initial points it reached through demonstrations.

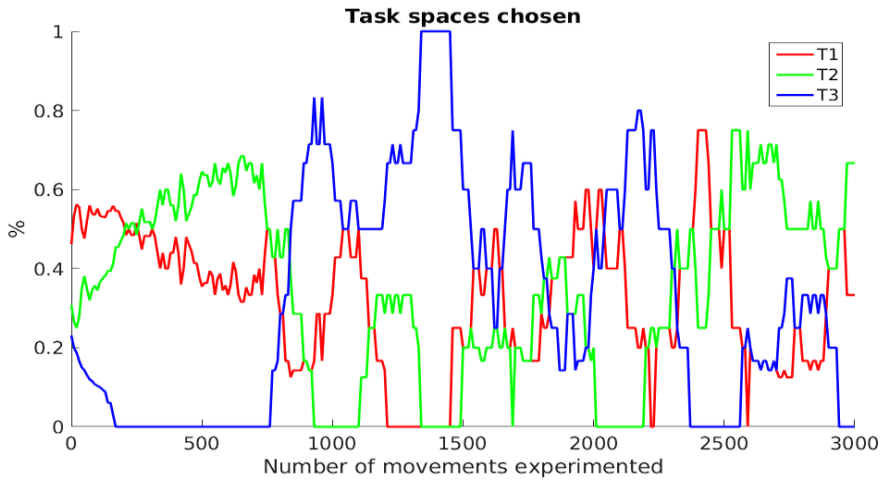

FIGURE 3.9: Evolution of the choice of tasks of SGIM-ACTSCL: percentage of times each task is chosen across time

Detailing the different types of outcomes, Fig. 3.9 shows 4 phases in time. The learner focuses on type of outcomes Ω_1 in the beginning. The combined choice of the imitation strategy with the task space Ω_1 enabled the robot to progress quickly and starts choosing the task space Ω_2 in the second phase for $250 < t < 800$. After accumulating skills, from $t = 800$, it became able to tackle the most complex task space Ω_3. Finally for $t > 1500$, the SGIM-ACTSCL learner kept using autonomous exploration with the three task spaces, focusing more on the difficult tasks Ω_3 and Ω_2. This finally enabled him to overtake the imitation algorithm by keeping progressing when the latter stagnates.

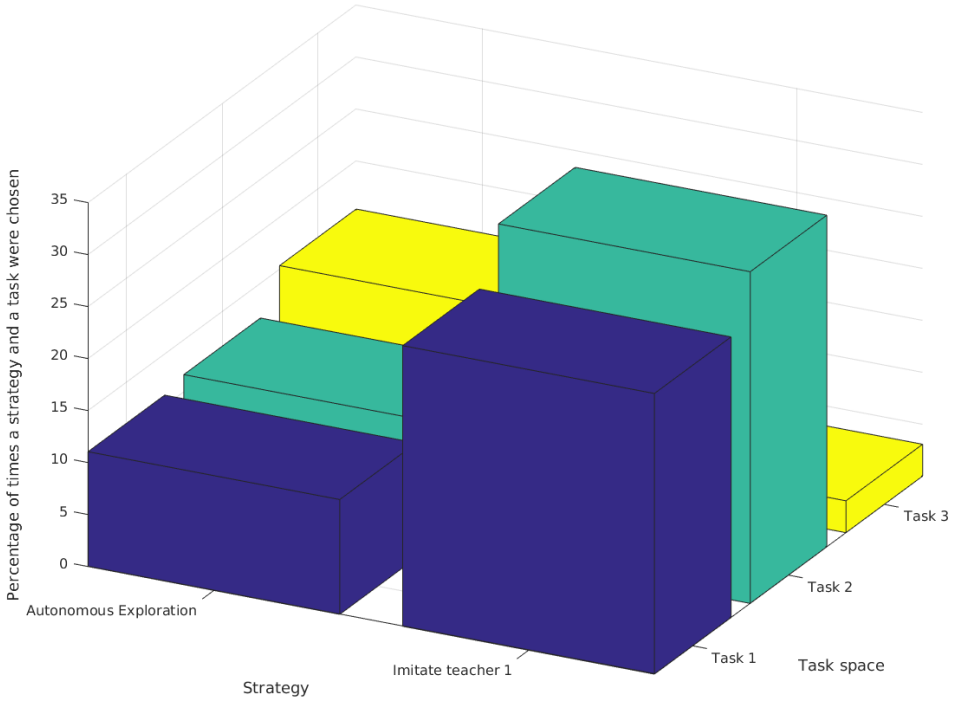

FIGURE 3.10: Synergy between the choice of task space and the choice of learning strategy of SGIM-ACTSCL: percentage of times each strategy and task is chosen over all the learning process

Fig. 3.10 shows that the task space Ω_3 was mostly combined with autonomous exploration on the overall learning process when the imitation was more associated with Ω_1 and Ω_2. The learner could **coherently choose the adequate exploration method for each task**.

The learner showed it was capable to make wise strategic decisions regarding the outcome spaces to tackle and the best strategy to use for it. It started with the simplest task space Ω_1 and also tried quickly Ω_2 and chose to imitate the teacher for enabling it to make the quickest progress. It was then capable to tackle the most difficult task space Ω_3 and noticed the teacher was less adapted for it. The learner successfully used its first acquired dataset of task spaces Ω_1 and Ω_2 to autonomously explore the more complex task space Ω_3.

3.2.4 Conclusions

The SGIM-SAHT architecture appears to be able to **self-organize its learning process to learn a set of hierarchically organized tasks in a continuous environment**.

It is capable of correctly assessing which strategy is more appropriate with each outcome space. It was able to learn on a developmental manner, focusing on the easiest task first, before tackling increasingly more difficult ones, requesting demonstrations from the teacher at the beginning when it knew barely nothing, before relying on autonomous exploration to extend its range of skills. All of these reasons explain why the SGIM-ACTSCL learner outperforms its competitors on this setup.

The learning architecture therefore seems a viable candidate for learning a set of hierarchical organized tasks using complex motor actions. However, as this transition from simple actions to possibly infinite successions of primitive actions can increase the curse of dimensionality, I need to design a mechanism to enable my learner to discover and exploit the task hierarchy to ease its learning process. It needs to be able to combine previously learned skills to learn more complex ones iteratively. In the next chapter, I describe an experimental setup with a hierarchical set of tasks, tackled by a learner performing sequences of motor actions at will. I also describe the method I found for easing this learning using the task hierarchy: the procedure framework.

Chapter 4

Using the task hierarchy to form sequences of motor actions

In this chapter, I introduce the learning of sequences of motor primitives. While the previous chapter considered, as a majority of works in motor learning, an action space at fixed dimensionality, thus actions of bounded complexity, I would like to extend these methods for unbounded sequences of motor primitives and for larger outcome spaces. *How can a robot learn to achieve a set of hierarchically organized tasks using unbounded sequences of primitive actions?*

To enable a robot to learn the mapping between unbounded and high-dimensional outcome and action spaces, I introduce in this chapter a goal-oriented representation of sequences of actions, and propose two versions of the algorithm for multi-task learning. The first version of the algorithm introduces the **procedural framework, built to discover and exploit the task hierarchy**. It performs autonomous exploration only, and its results are shown and analysed in the second section. Then, a second version of the algorithm adds interactive strategies to built a socially guided intrinsically motivated learner, so as to study whether its bootstrapping effect extend to sequences of motor actions learning. This latter algorithm is described in the section 4.4. I show that both algorithms are capable of determining a task hierarchy representation to learn a set of complex interrelated tasks using adapted action sequences, and that the performance of my algorithm is bootstrapped by a tutor's demonstrations.

To illustrate the multi-task learning problems that I am considering, I first describe an experimental setup. The code for the algorithms developed in this chapter as well as the experimental setup described is available at `https://bitbucket.org/smartan117/sgim_iclr`.

4.1 Experimental setup

In this study, I designed an experiment with a simulated robotic arm, which can move in its environment and interact with objects in it. I considered a setup with multiple tasks to learn, with tasks independent of each other and tasks that are interdependent. For interdependent tasks, I was inspired by tool use examples such as the setup proposed in (Forestier, Mollard, and Oudeyer, 2017). The robot can learn an infinite number of tasks, grouped as 6 hierarchically organized types of tasks. The robot is capable of performing **action sequences of unrestricted size** (i.e. consisting of any number of primitives), with primitive actions highly redundant and of high dimensionality. The experimental setup was first introduced in (Duminy, Nguyen, and Duhaut, 2018b).

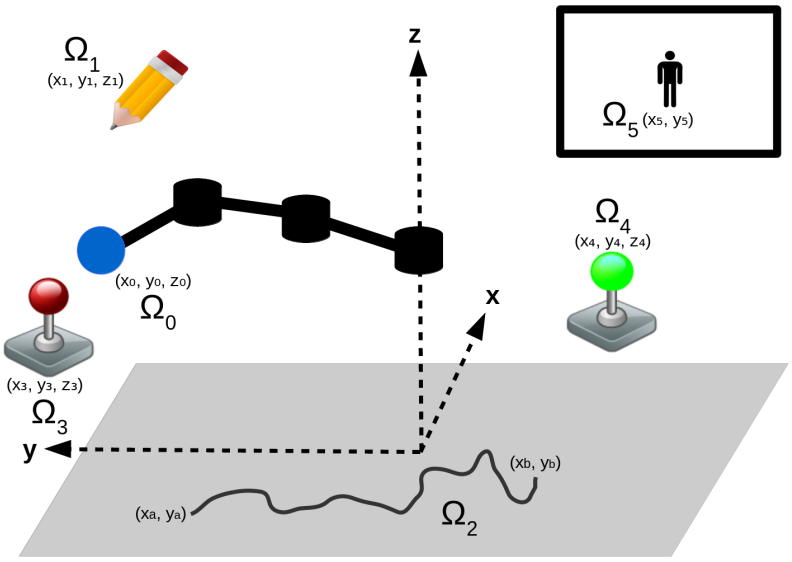

FIGURE 4.1: Experimental setup: a robotic arm, can interact with the different objects in its environment (a pen and two joysticks). Both joysticks enable to control a video-game character (represented in top-right corner). A grey floor limits its motions and can be drawn upon using the pen (a possible drawing is represented).

4.1.1 Environment

The Fig. 4.1 shows environmental setup (contained in a cube delimited by $(x, y, z) \in [-1; 1]^3$). The learning agent is a **planar robotic arm of 3 joints** with the base centred on the horizontal plane, able to rotate freely around the vertical axis (each link has a length of 0.33) and change its position on the z-axis. The robot can grab objects in this environment, by hovering its arm tip (blue in the Fig. 4.1) close to them, which position is noted (x_0, y_0, z_0). The robot can interact with:

- Floor (below $z = 0.0$): limits the motions of the robot, slightly elastic which enable the robot to go down to $z = -0.2$ by forcing on it;
- Pen: can be moved around and draw on the floor, broken if forcing too much on the floor (when $z <= -0.3$);
- Joystick 1 (the red one on the figure): can be moved inside a cube-shaped area (automatically released otherwise, position normalized for this area), its x-axis position control a video-game character x position on the screen when grabbed by the robot;
- Joystick 2 (the green one on the figure): can be moved inside a cube-shaped area (automatically released otherwise, position normalized for this area), its y-axis position control a video-game character y position on the screen when grabbed by the robot;
- Video-game character: can be moved on the screen by using the two joysticks, its position is refreshed only at the end of a primitive action execution for the manipulated joystick.

The robot grabber can only handle one object. When it touches a second object, it breaks, releasing both objects.

The robot always starts from the same position before executing an action, and primitives are executed sequentially without getting back to this initial position. Whole action sequences are recorded with their outcomes, but each step of the action sequence execution is also recorded. This is done so as to enable the robot to select parts of action sequences when it can, thus helping it to optimize the size of action sequences it executes with respect to the outcomes at hand.

4.1.2 Formalization of tasks and actions

The distance used to compare two actions or outcomes together is the normalized euclidean distance.

Action spaces

The motions of each of the three joints of the robot are encoded by one-dimensional Dynamic Movement Primitive (DMP) (Pastor et al., 2009), defined by the system:

$$\tau \dot{v} = K(g - x) - Dv + (g - x_0)f(s) \tag{4.1}$$
$$\tau \dot{x} = v \tag{4.2}$$
$$\tau \dot{s} = -\alpha s \tag{4.3}$$

where x and v are the position and velocity of the system; s is the phase of the motion; x_0 and g are the starting and end position of the motion; τ is a factor used to temporally scale the system (set to fix the length of a primitive execution); K and D are the spring constant and damping term fixed for the whole experiment; α is also a constant fixed for the experiment; and f is a non-linear term used to shape the trajectory called the forcing term. This forcing term is defined as:

$$f(s) = \frac{\sum_i w_i \psi_i(s) s}{\sum_i \psi_i(s)} \tag{4.4}$$

where $\psi_i(s) = \exp(-h_i(s - c_i)^2)$ with centers c_i and widths h_i fixed for all primitives. There are 3 weights w_i per DMP.

The weights of the forcing term and the end positions are the only parameters of the DMP used by the robot. The starting position of a primitive is set by either the initial position of the robot (if it is starting a new action sequence) or the end position of the preceding primitive. The robot can also set its position on the vertical axis z for every primitive. Therefore a primitive action π_θ is parametrized by:

$$\theta = (a_0, a_1, a_2, z) \tag{4.5}$$

where $a_i = (w_0^{(i)}, w_1^{(i)}, w_2^{(i)}, g^{(i)})$ corresponds to the DMP parameters of the joint i, ordered from base to tip, and z is the fixed vertical position. Thus, the primitive action space is $\Pi = \mathbb{R}^{13}$. When combining two or more primitive actions $(\pi_{\theta_0}, \pi_{\theta_1}, ...)$, in an action sequence π_θ, the parameters $(\theta_0, \theta_1, ...)$ are simply concatenated together from the first primitive to the last. The total action space, $(\mathbb{R}^{13})^{\mathbb{N}}$ is of unbounded dimension.

Outcome subspaces

The outcome subspaces the robot learns to reach are hierarchically organized and defined as:

- Ω_0: the position (x_0, y_0, z_0) of the end effector of the robot in Cartesian coordinates at the end of an action execution;
- Ω_1: the position (x_1, y_1, z_1) of the pen at the end of an action execution if the pen is grabbed by the robot;
- Ω_2: the first (x_a, y_a) and last (x_b, y_b) points of the last drawn continuous line on the floor if the pen is functional (x_a, y_a, x_b, y_b);
- Ω_3: the position (x_3, y_3, z_3) of the first joystick at the end of an action execution if it is grabbed by the robot;
- Ω_4: the position (x_4, y_4, z_4) of the second joystick at the end of an action execution if it is grabbed by the robot;
- Ω_5: the position (x_5, y_5) of the video-game character at the end of an action execution if moved.

The outcome space is a composite and continuous space $\Omega = \cup_{i=0}^{5} \Omega_i$, with subspaces of 3 to 4 dimensions. A quick analysis of this setup highlights interdependencies between tasks: controlling the position of the pen comes after controlling the position of the end effector; and controlling the position of the video-game character comes after controlling the positions of both joysticks, which in turn comes after controlling the position of the end effector. In my setup, the most complex task is controlling the position of the video-game character. This task should require a sequence of 4 actions : move the end-effector to initial position of the joystick 1, move joystick 1, then move the end-effector to the initial position of joystick 2, and move joystick 2. Besides, there are independent tasks: the position of the pen does not really depend on the position of the video-game character. Therefore, the interdependencies can be grouped into 2 dependency graphs, these are shown in Fig. 4.2. With this setup, I test if the robot can **distinguish task hierarchies between dependent and independent tasks, and can compose tools uses**.

In this setup, my intuition is that a learning agent should start by making good progress in the easy tasks in Ω_0 then $\Omega_1, \Omega_3, \Omega_4$. Once it has a good mastery of the easy tasks, it can reuse this knowledge to learn to achieve higher-level tasks.

In my multi-task learning perspective, I will examine how well the robot performs for each of the tasks in these subspaces. I will particularly examine its performance for the tasks of Ω_5, which I consider the most complex tasks.

In the next section, I formalize my learning problem by introducing a goal- oriented representation of sequences of actions, named the *procedures*.

4.2 Procedures framework

As this algorithm tackles the learning of complex hierarchically organized tasks, exploring and exploiting this hierarchy could ease the learning of the more complex tasks. I define procedures as a way to encourage the robot to reuse previously learned tasks, and chain them to build more complex ones. More formally, **a procedure is defined as a succession of previously known outcomes** $(\omega_1, \omega_2, ..., \omega_n \in \Omega)$

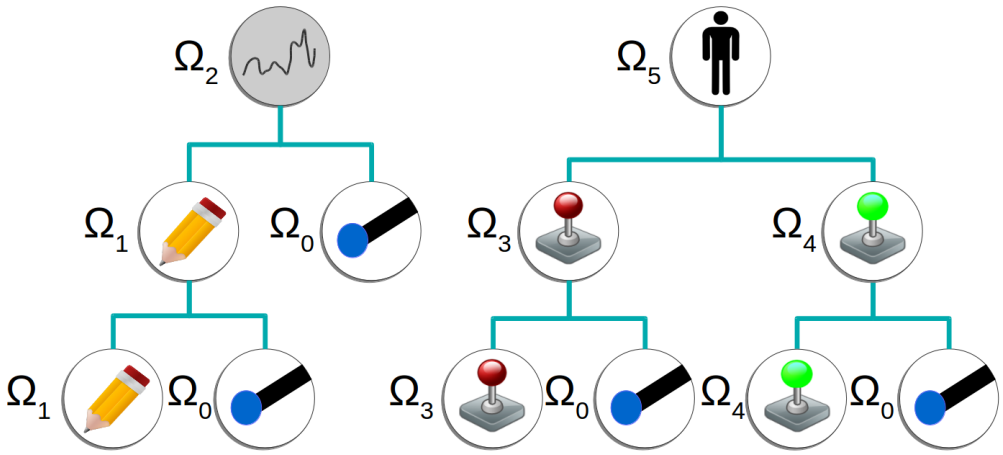

FIGURE 4.2: Task hierarchy represented in this experimental setup

and is noted $(\omega_1, \omega_1, ..., \omega_n)$. The procedure space is thus simply $\Omega^{\mathbb{N}}$. The definition of the procedure space only depends on the outcome space. But the valid procedures, representing the real dependencies between tasks, depend on each application case. Thus the learning agent can explore the procedure space to test which procedures are valid.

Executing a procedure $(\omega_1, \omega_1, ..., \omega_n)$ means building the action sequence π corresponding to the succession of actions $\pi_i, i \in [\![1, n]\!]$ (potentially action sequences as well) and execute it (where the π_i reach best the ω_i $\forall i \in [\![1, n]\!]$ respectively). An example illustrates this idea of task hierarchy in Fig. 1.1. As the subtasks ω_i are generally unknown from the learner, the procedure is updated before execution (see Algo. 5) to subtasks ω'_i which are the closest tasks reached by the learner (by executing respectively π'_1 to π'_n). When the agent selects a procedure to be executed, this latter is only a way to build the action sequence which will actually be executed. So the agent does not check if the subtasks are actually reached when executing a procedure.

Algorithm 5 Procedure adaptation

Input: $(\omega_1, ..., \omega_n) \in \Omega^n$
Input: inverse model L
1: **for** $i \in [\![1, n]\!]$ **do**
2: $\omega'_i \leftarrow$Nearest-Neighbour(ω_i) // *get the nearest outcome known from* ω_i
3: $\pi'_i \leftarrow L(\omega'_i)$ // *get the known action sequence that reached* ω'_i
4: **end for**
5: **return** $\pi = \pi'_1 ... \pi'_n$

If the procedure given can not be executed by the robot, because at least one of the subtasks space is not reachable, then the procedure is abandoned and replaced by a random action sequence.

4.3 Intrinsically Motivated Procedure Babbling

In this section, I describe an intrinsically motivated learner, able to learn action sequences through autonomous exploration. **This algorithm, called Intrinsically Motivated Procedure Babbling (IM-PB), learns action sequences to complete multiple tasks by exploring autonomously, driven by intrinsic motivation**. It uses the procedure framework, to expand its learning capabilities on complex tasks by combining previously learned simpler tasks. This algorithm and the experimental results were first published in (Duminy, Nguyen, and Duhaut, 2018b).

Contrarily to the definiiton of procedures, for the algorithm, I limited my study to the case of procedures of size 2 (sequences of 2 outcomes only) as I wish to prove the bootstrapping effect of the representation via procedures, before tackling the challenges of exploring a high-dimensional space of procedures $\Omega^{\mathbb{N}}$. This still allows the learning agent to use a high number of subtasks because of the recursivity of the definition of procedures.

My learning algorithm, called IM-PB, starts from scratch, it is only provided with the primitive action space and outcome subspaces dimensionalities and boundaries. The procedural spaces are also predefined, as all the possible composition of outcome subspaces (Ω_i, Ω_j) with $\Omega_i, \Omega_j \in \Omega$. Then its aim is to learn how to reach a set of outcomes as broad as possible, as fast as possible. This means it has to learn both what are the possible outcomes to reach and the action sequences or procedures to use for that. In order to learn, the agent can count on different learning strategies, which are methods to build an action or procedure from any given target outcome. It also needs to map the outcome subspaces and even regions to the best suited strategies to learn them. In this algorithm, the forward and inverse models are memory based and consist only of the cumulative data, mappings of actions, procedures and their respective reached outcomes obtained through all the attempts of the learner. So they are learned by adding new data in the learner's memory.

The IM-PB algorithm learns by episode, each of which starts by the learner choosing a goal outcome ω_g to target and a strategy σ to use, based on its progress, as detailed in section 3.1.2 with SGIM-ACTSCL. In each episode, the robot starts from the same position before executing an action, and primitives are executed sequentially without getting back to this initial position. Whole action sequences are recorded with their outcomes, but each step of the action sequence execution is also recorded. These data enable the robot to select parts of the action sequences, thus helping it to optimize the size of action sequences it executes with respect to the outcomes at hand. The way these data are generated depend on the strategy chosen. The strategies available for the learner are the autonomous exploration of the action space and that of the procedure space.

4.3.1 Strategies

Autonomous exploration of the action space

In an episode under the autonomous exploration of the action space strategy, the learner tries to optimize the action sequence π to produce ω_g using on of these methods:

- Global exploration: the learner performs a random action sequence of unconstrained size;
- Local exploration: the learner optimizes a action sequence for the specific goal outcome using the best local inverse model.

The metric used to make this choice is the same than in SGIM-ACTSCL in section 3.1.1.

The global exploration builds recursively an action sequence following Algo. 6. It starts by a single random primitive action, and it chains it with other random primitive actions with a probability of $1/\alpha^n$, where $\alpha = 2$ is a constant controlling the distribution of the size of produced actions and n is the current size of the built action.

Algorithm 6 Random Action Sequence

Input: constant α
Initialization: $\pi \leftarrow \varnothing$
Initialization: $n \leftarrow 0$
loop
  $r \leftarrow 1/\alpha^n$
  $p \leftarrow$ Random number between 0 and 1
  if $p > r$ **then**
    Break
  end if
  $\pi_{\theta_n} \leftarrow$ Random Primitive Action
  $\pi \leftarrow$ Concatenate π and π_{θ_n}
  $n \leftarrow n + 1$
end loop
return π

The local exploration optimizes an action sequence to reach at best the target outcome. The best locality function is used to determine the local inverse model used. Then multivariate linear regression is used to build an action sequence. This action sequence is modified by adding a uniform noise with a range inversely proportional to the standard deviation among the actions in the local inverse model used. This noise is added to increase exploration in case of a too homogeneous local model. The complete algorithm for this strategy is described in Algo. 7.

Algorithm 7 Autonomous exploration of the action space

Input: a target outcome ω_g
$Y \leftarrow$Nearest-Neighbours(ω_g)
mode $\leftarrow$ mode *global-exploration* or *local-exploration*(Y, ω_g)
if *mode* =Global exploration **then**
  $\pi \leftarrow$ Random Action Sequence
  Execute action(π)
else if *mode* =Local exploration **then**
  $M \leftarrow$ Best Locality(ω_g)
  $\pi \leftarrow$ Linear Regression(M, ω_g)
  $\epsilon \leftarrow$ Maximum noise proportional to standard deviation of actions in M
  $\pi_{rand} \leftarrow$ random vector with $\mid \pi_{rand} \mid < \epsilon$
  $\pi \leftarrow \pi + \pi_{rand}$
  Execute action(π)
end if

Autonomous exploration of the procedure space

In an episode under the autonomous exploration of the procedure space strategy, the learner builds a size 2 procedure (ω_i, ω_j) such as to reproduce the goal outcome ω_g the best using one of these methods:

- Global exploration: the learner performs a random procedure;
- Local exploration: the learner optimizes a procedure for the specific goal outcome using the best local inverse model.

The metric used to make this choice is the same than in SGIM-ACTSCL in section 3.1.1. When no outcome space has been discovered, the execution of any procedure is not feasible, therefore in this case the learner produces a random action sequence following the global exploration method of the autonomous exploration of the action space strategy.

The global exploration method builds a random procedure by selecting two outcome spaces Ω_i and Ω_j already known to the learner (i.e. reached at least once by the learner), and then select both random components of the procedures among them. The algorithm is described in Algo. 8.

Algorithm 8 Random Procedure

$l \leftarrow$ List of outcome spaces Ω_i reached at least once
if $l = \varnothing$ **then**
  return $\varnothing$
else
  $(\Omega_i, \Omega_j) \leftarrow$ Choose two random outcome spaces from l
  $\omega_i \leftarrow$ Random vector from Ω_i
  $\omega_j \leftarrow$ Random vector from Ω_j
  return (ω_i, ω_j)
end if

The local exploration optimizes a procedure to reach the target at best. The best locality function is also used to determine the local inverse model used (in this case the inverse model is a subpart of $\Omega^2 \rightarrow \Omega$). The procedure obtained is modified by adding a uniform noise proportional to the standard deviation among the procedures of the local model similarly to the local exploration of the action space in 4.3.1. The complete algorithm of the strategy is described in Algo. 9.

4.3.2 Overview

The IM-PB algorithm learns by episodes. It starts each episode by selecting a goal outcome ω_g and a strategy to use.

Its available strategies are autonomous exploration of the action space, and that of the procedure space.

The strategy used for the episode, builds a feature sequence l_f (either a sequence of motor actions or a procedure), which is then executed by the learner. The reached outcomes ω, along with the executed feature sequence l_f are recorded.

The interest model is then updated, according to the data acquired during the episode.

The complete algorithm is shown on Fig. 4.3.

Algorithm 9 Autonomous exploration of the procedure space

Input: a target outcome ω_g
Initialization: $p \leftarrow \varnothing$

$Y \leftarrow$Nearest-Neighbours(ω_g)
$mode \leftarrow$ mode *global-exploration* or *local-exploration*(Y, ω_g)
if $mode$ =Global exploration **then**
  $p = (\omega_i, \omega_j) \leftarrow$ Random Procedure
  if $p = \varnothing$ **then**
    $\pi \leftarrow$ Random Action Sequence
    Execute action(π)
  else
    Execute procedure(p)
  end if
else if $mode$ =Local exploration **then**
  $M \leftarrow$ Best Locality Procedure(ω_g)
  $p = (\omega_i, \omega_j) \leftarrow$ Linear Regression(M, ω_g)
  $\epsilon \leftarrow$ Maximum noise proportional to standard deviation of procedures in M
  $p_{rand} \leftarrow$ random vector with $\mid p_{rand} \mid < \epsilon$
  $p \leftarrow p + p_{rand}$
  Execute procedure(p)
end if

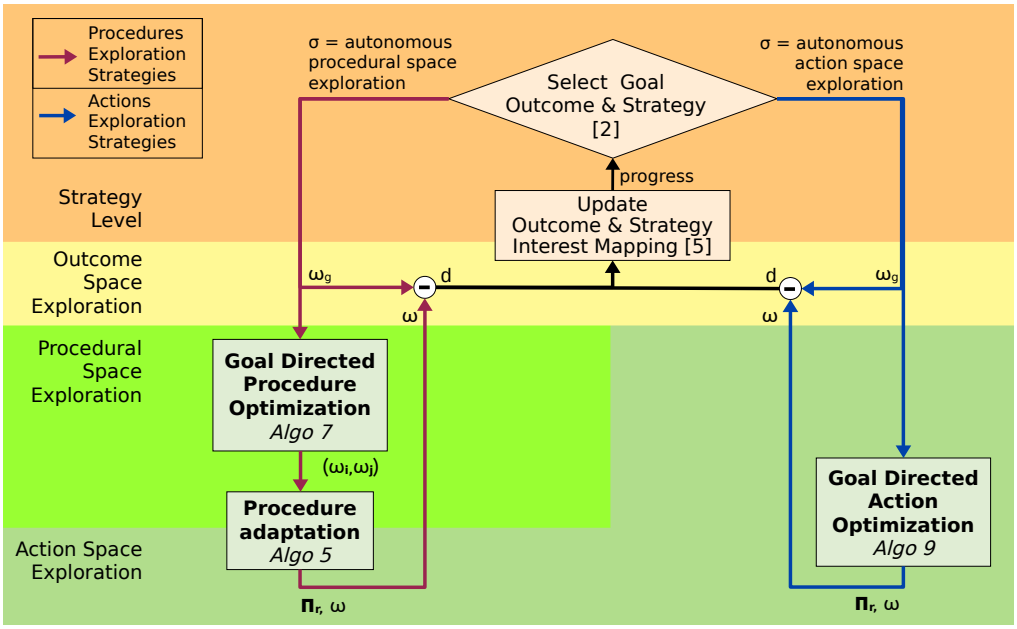

FIGURE 4.3: Architecture of the IM-PB algorithm: number between brackets link parts of the architecture with lines in Algo. 1, the arrows show data transfer between the different blocks

An important change from the SGIM-ACTSCL algorithm, is the development of a new metric called the performance, which adds the action cost to the competence measure. The learner can compute nearest neighbours to select actions or procedures to optimize (when choosing local optimization in any of both autonomous

exploration strategies and when refining procedures) or when computing the competence to reach a specific goal, it actually uses a performance metric (4.6) which also takes into account the complexity of the action chosen:

$$perf(\omega_g) = d(\omega, \omega_g)\gamma^n \tag{4.6}$$

where $d(\omega, \omega_g)$ is the normalized Euclidean distance between the target outcome ω_g and the outcome ω reached by the action, γ is a constant and n is equal to the size of the action (the number of primitives chained).

4.3.3 Experiment

Evaluation Method

To evaluate my algorithm, I created a **benchmark** linearly distributed across the Ω_i, of 27,600 points. The evaluation consists in computing mean Euclidean distance between each of the benchmark outcomes and their nearest neighbour in the learner dataset. This evaluation is repeated regularly.

Then to asses my algorithm efficiency, I compare its results to those algorithms:

- RandomAction: performs random exploration of the action space $\Pi^{\mathbb{N}}$;
- SAGG-RIAC: performs autonomous exploration of the action space $\Pi^{\mathbb{N}}$ guided by intrinsic motivation;
- Random-PB: performs both random exploration of actions and procedures;
- IM-PB: performs both autonomous exploration of the procedural space and the action space, guided by intrinsic motivation.

Each algorithm was run 5 times for 25,000 iterations (complete action sequences executions). The meta parameter was: $\gamma = 1.2$.

4.3.4 Results

Evaluation performance

Fig. 4.4 shows the global evaluation of all tested algorithms, which is the mean error made by each algorithm to reproduce the benchmarks with respect to the number of complete action sequences tried. **Random-PB and IM-PB owing to procedures have lower errors than the others even since the beginning**. Indeed, they perform better than the downgrades without procedures, RandomAction and SAGG-RIAC. We can also see, through the final standard deviation given in the legend for each algorithm, that those results are consistent.

On **each individual outcome space** (Fig. 4.5), **IM-PB outperforms the other algorithms**. The comparison of the learners without procedures (RandomAction and SAGG-RIAC) with the others shows they learn less on any outcome space but Ω_0 (reachable using single primitives, with no subtask) and especially for Ω_1, Ω_2 and Ω_5 which were the most hierarchical in this setup. So the **procedures helped when learning any potentially hierarchical task** in this experiment.

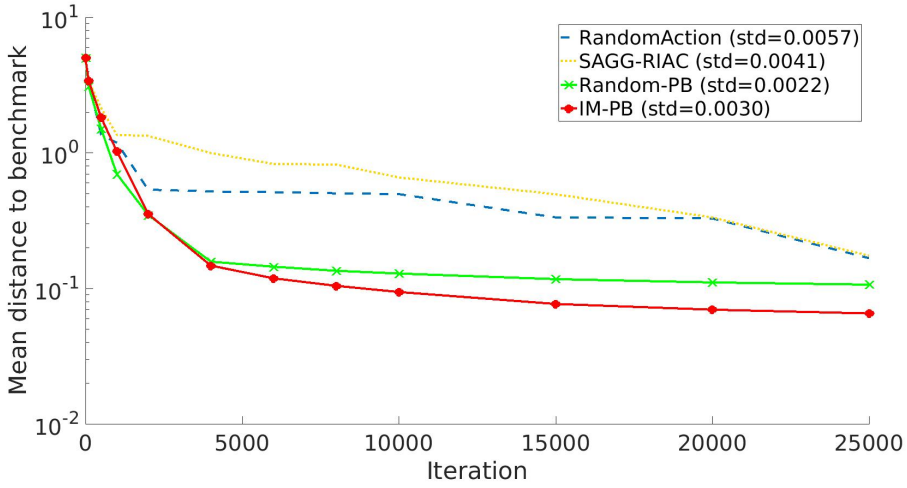

FIGURE 4.4: Evaluation of all algorithms (standard deviation shown in caption)

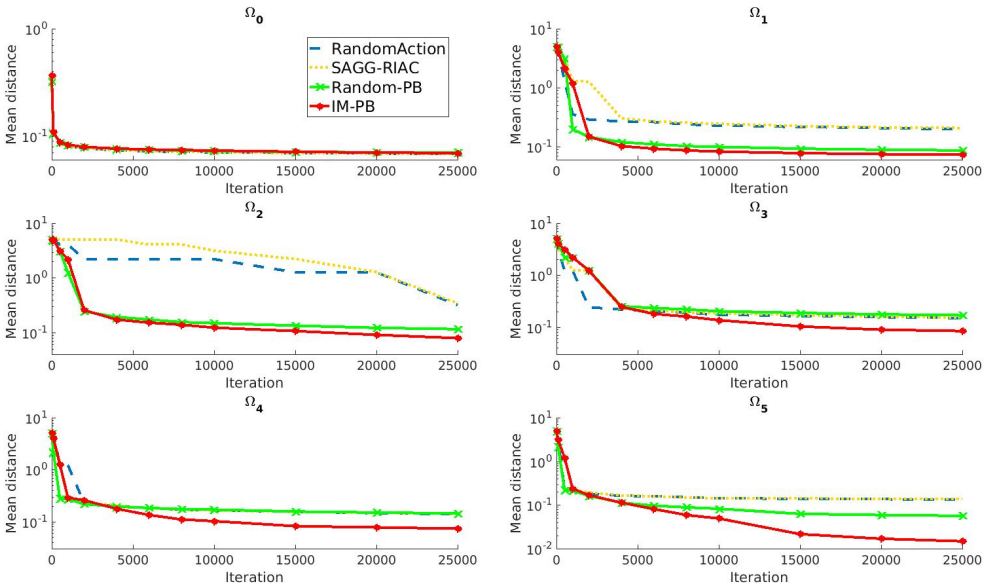

FIGURE 4.5: Evaluation of all algorithms per outcome space (for Ω_0, all evaluations are superposed)

Lengths of action sequences used

I wanted to see if my IM-PB learner adapts the complexity of its actions to the working task. So I looked which action space would be chosen by the local optimization function (used inside the action space exploration strategy) for the Ω_0, Ω_1 and Ω_2 subspaces (chosen because they are increasingly complex) on their respective evaluation benchmarks. Fig. 4.6 shows the results of this analysis.

As we can see on those three interrelated outcome subspaces (Fig. 4.6), the learner is **capable to adapt the complexity of its action sequences to the outcome at hand**. It chooses longer actions for Ω_1 and Ω_2 (size 3 and 4 compared to size 1 for Ω_0). My learner is capable to correctly limit the complexity of its action sequences

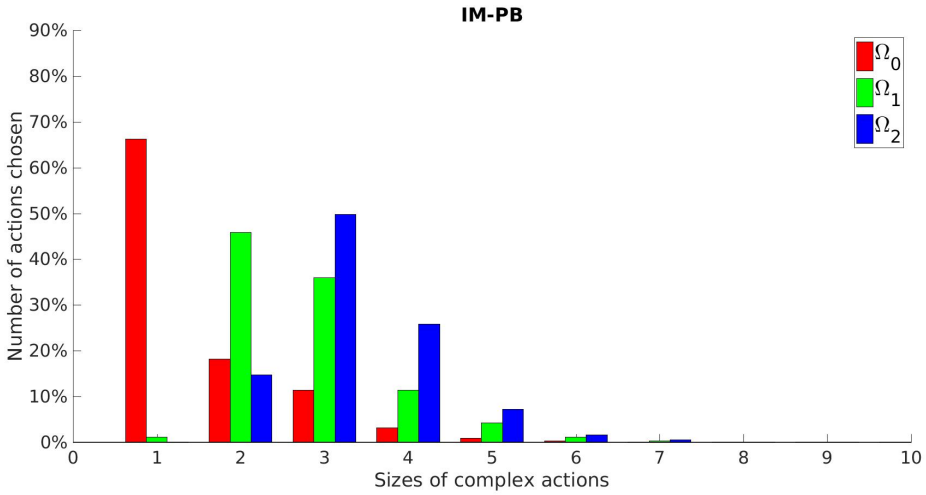

FIGURE 4.6: Number of actions selected per action size for three increasingly more complex outcome spaces by the IM-PB learner

instead of being stuck into always trying longer and longer actions. However, the learner did not increase its action sequences complexity from Ω_1 to Ω_2, as I hoped.

4.3.5 Conclusion

The results show, an intrinsically motivated learner is capable to learn sequences of motor actions. Intrinsic Motivation seems indeed to guide the learning process towards interesting regions. It also shows that my procedure framework is highly relevant to learn hierarchical set of tasks. In the next section, I finish the implementation of a complete socially guided learner intrinsically motivated, and analyze its performance on the experimental setup.

To summarize, I have introduced the framework of *procedures* as a goal-directed representation of sequences of primitive actions. To show that procedures can bootstrap the learning of action sequences, I have proposed IM-PB as a learning algorithm that leverages two data collection *strategies*: autonomous exploration of actions, and exploration of procedures. IM-PB learns to reach an ensemble of outcomes, by mapping them to actions. IM-PB takes advantage of the dependencies between tasks. It explores the procedure space to learn these dependencies. **Combining these procedures with the learning of simple actions to complete simple tasks, it can build sequences of actions to achieve complex tasks**.

I showed that the robot can take advantage of the procedures representation to improve its performance, especially on high-level tasks. It can also adapt the complexity of its action sequence to the task to achieve.

Nevertheless, this adaptation is limited to the first two levels of task hierarchy, and the learner can not well adapt this complexity to a deeper hierarchy of tasks. To help the robot improve its understanding of task dependencies, I explore in the next section how supplementary information from tutors can help the robot to learn task hierarchies.

4.4 Socially Guided Intrinsic Motivation with Procedure Babbling

In this section, I want to extend IM-PB, by providing human teachers to my learner. I show in this section, that as social guidance does in a simple action setup in Chapter 3, adding human action teachers help the learner by bootstrapping its early learning process by focusing on the most interesting parts of the action space. I also show that **adding human procedural teachers has a similar effect on its ability to focus on the most useful procedural spaces and adapt them to the task at hand**. To enable my socially guided learner to perform interactive strategies, I developed two different strategies. Both can provide demonstrations from human experts. These strategies, added to the IM-PB algorithm, transforms it to a new algorithm using social guidance as well as autonomous exploration, called Socially Guided Intrinsic Motivation with Procedure Babbling (SGIM-PB). I analyze in the results of this section, what are the advantages of both types of demonstrations (procedures or actions) and I show that they are complementary, as action demonstrations are well suited for the simplest outcome spaces while procedure demonstrations are better suited for the most complex and hierarchical tasks.

This algorithm is built on the same prerequisites and hypothesis than IM-PB. The only difference being the strategies available to both learners, SGIM-PB able to count on interactive strategies unavailable to IM-PB. The implementations and experimental results were presented in (Duminy, Nguyen, and Duhaut, 2019).

4.4.1 Interactive strategies

When implementing interactive learning, we need to think about two aspects: **what** human expertise will provide (what kind of data), and **when** it will provide them. For the second aspect, I am considering an active learner, and therefore consider only strategies in which help is providing to the learner's request. For the first aspect, we need to look at what the learner can do. It can execute sequences of motor actions, but thanks to the procedural framework, it can also perform procedures. Therefore, I design two types of teachers that can interact with the learner at the learner's request: *action teachers* and *procedural teachers*.

Action teachers

This first type of strategy enables a teacher to provide demonstrations of sequences of motor actions to the learner on the learner's request. It functions exactly like for the SGIM-ACTSCL algorithm, except the actions might here be sequences. This strategy is also called mimicry of a action teacher.

Procedural teachers

This second type of strategy enables a teacher to provide demonstrations of procedures to the learner according to a **preset function** which depends on the goal outcome ω_g. The procedures are computed on the fly when the learner requests them and don't need to be recorded first as the action demonstrations from a action teacher do. In this case, another factor can prevent the learner from using the provided demonstration well and it is its current skill set. Indeed the procedure is adapted to the learner's skill set before being executed (using Algo. 5) and can

thus be quite different from the one provided by the teacher. This strategy is called mimicry of a procedural teacher.

4.4.2 Algorithm overview

The SGIM-PB algorithm learns by episode. It starts each episode by selecting a goal outcome ω_g and a strategy to use.

Its available strategies are autonomous exploration of the action space, autonomous exploration of the procedure space, mimicry of one of the available action or procedural teachers (N.B. each teacher is considered a specific strategy by the learner).

The strategy used for the episode, builds a feature sequence l_f (either a motor action sequence or a procedure), which is then executed by the learner. The reached outcomes ω, along with the executed feature sequence l_f are recorded.

The interest model is then updated, according to the data acquired during the episode.

The complete algorithm is shown on Fig. 4.7.

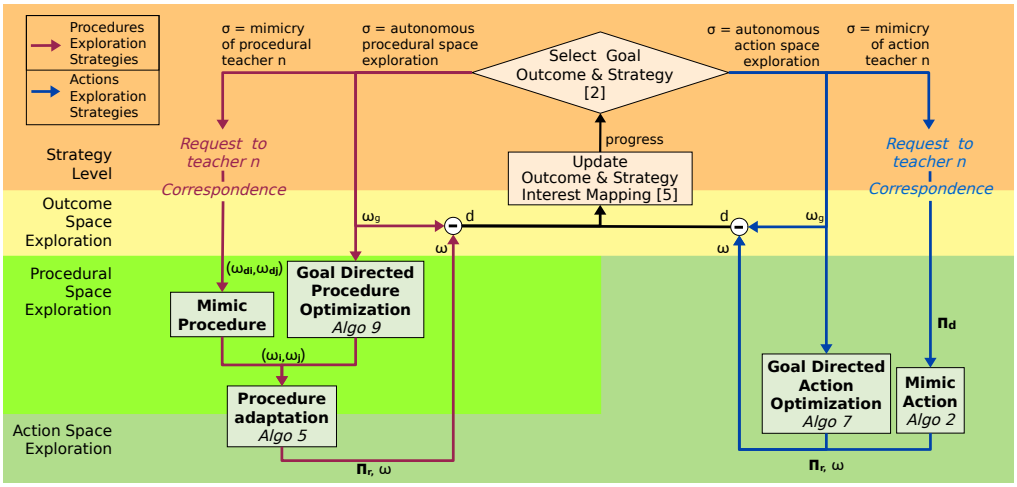

FIGURE 4.7: Architecture of the SGIM-PB algorithm: number between brackets link parts of the architecture with lines in Algo. 1, the arrows show data transfer between the different blocks

4.4.3 Experiment

Teachers

My SGIM-PB learner can actively learn by asking teachers to give demonstrations of procedures or actions (strategies *Mimic procedural teacher* and *Mimic action teacher*).

To help the SGIM-PB learner, procedural teachers were available so as to provide procedures for every complex outcome subspaces Ω_1, Ω_2, Ω_3, Ω_4 and Ω_5. As Ω_0 is the simplest outcome space in my setup, the base of its task hierarchy, I decided to build the preset functions for these procedural teachers up from Ω_0. Each teacher was only giving procedures useful for its own outcome space, and was aware of its task representation. When presented with an outcome outside its outcome space of expertise, it provides a demonstration for a newly drawn random target outcome in its outcome space of expertise. They all had a cost of 5. The rules used to provide procedures are the following:

- ProceduralTeacher1 ($\omega_{1_g} \in \Omega_1$): (ω_1, ω_0) with $\omega_1 \in \Omega_1$ equals to the pen initial position and $\omega_0 \in \Omega_0$ equals to the desired final pen position ω_{1_g};
- ProceduralTeacher2 ($\omega_{2_g} = (x_a, y_a, x_b, y_b) \in \Omega_2$): (ω_1, ω_0) with $\omega_1 \in \Omega_1$ equals to the point on the $z = 1.0$ plane above the first point of the desired drawing $\omega_1 = (x_a, y_a, 1)$, and $\omega_0 \in \Omega_0$ equals to the desired final drawing point, $\omega_0 = (x_b, y_b, 0)$;
- ProceduralTeacher3 ($\omega_{3_g} \in \Omega_3$): (ω_3, ω_0) with $\omega_3 = (0, 0, 0), \omega_3 \in \Omega_3$ and $\omega_0 \in \Omega_0$ equals to the end effector position leading to the desired final position of the first joystick ω_{3_g};
- ProceduralTeacher4 ($\omega_{4_g} \in \Omega_4$): (ω_4, ω_0) with $\omega_4 = (0, 0, 0), \omega_4 \in \Omega_4$ and $\omega_0 \in \Omega_0$ equals to the end effector position leading to the desired final position of the second joystick ω_{4_g};
- ProceduralTeacher5 ($\omega_{5_g} = (x, y) \in \Omega_5$): (ω_3, ω_4) with $\omega_3 = (x, 0, 0), \omega_3 \in \Omega_3$ with x corresponding to the desired x-position of the video-game character, $\omega_4 = (0, y, 0), \omega_4 \in \Omega_4$ with y corresponding to the desired y-position of the video-game character.

I also added action teachers corresponding to the same outcome spaces to bootstrap the robot early learning process. The strategy attached to each teacher has a cost of 10. Each teacher was capable to provide demonstrations (as actions executable by the robot) linearly distributed in its outcome space. All those teachers consist of demonstrations repertoires built by drawing sparse demonstrations from a random action learner trained a huge amount of time (1,000,000 iterations):

- MimicryTeacher1 (Ω_1): 15 demonstrations;
- MimicryTeacher2 (Ω_2): 25 demonstrations;
- MimicryTeacher3 (Ω_3): 18 demonstrations;
- MimicryTeacher4 (Ω_4): 18 demonstrations;
- MimicryTeacher5 (Ω_5): 9 demonstrations;

These costs were chosen so as to encourage the robot to rely on itself as much as possible to reduce the teacher load. The costs of 10 for an action teacher strategy and 5 for a procedural teacher are arbitrary. Their difference comes from my belief that giving a procedure takes less time to the teacher than providing it with a detailed demonstrated motor action.

Evaluation method

The method used to evaluate an algorithm on the setup is the same than in 4.3.3. To assess my algorithm efficiency, I compare its results with 3 other algorithms:

- SAGG-RIAC: performs autonomous exploration of the action space $\Pi^{\mathbb{N}}$ guided by intrinsic motivation;
- SGIM-ACTS: interactive learner driven by intrinsic motivation. Choosing between autonomous exploration of the action space $\Pi^{\mathbb{N}}$ and mimicry of one of the available action teachers;

- IM-PB: performs both autonomous exploration of the procedural space and the action space, guided by intrinsic motivation;
- SGIM-PB: interactive learner driven by intrinsic motivation. Choosing between autonomous exploration strategies (either of the action space or the procedural space) and mimicry of one of the available teachers (either action or procedural teachers).

Each algorithm was run 10 times on this setup. Each run, I let the algorithm performs 25,000 iterations (complete action sequences executions). The value of γ for this experiment is 1.2. The probabilities to choose either of the sampling mode of SGIM-PB are $p_1 = 0.15$, $p_2 = 0.65$, $p_3 = 0.2$.

4.4.4 Results

Distance to goals

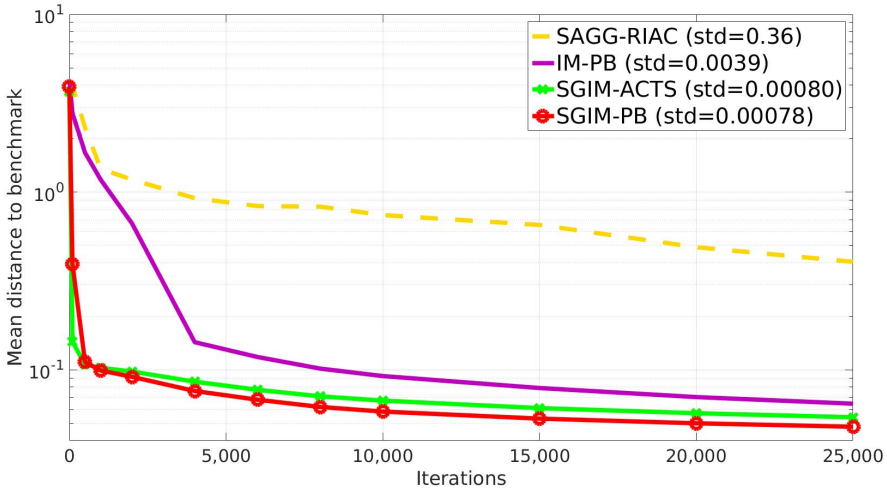

FIGURE 4.8: Evaluation of all algorithms (final standard deviation shown in caption)

The Fig. 4.8 shows the global evaluation of all the tested algorithms, which corresponds to the mean error made by each algorithm to reproduce the benchmarks with respect to the number of complete action sequences tried. Random, SGIM-ACTS, SGIM-PB were run 20 times while IM-PB and SAGG-RIAC was run 10 times on this setup so as to obtain statistically significant differences between SGIM-PB and the other algorithms, according to the Student's t-test on two algorithms : $p = 3 * 10^{-16} < 0.1$ when compared with random, p=0.01 for SAGG-RIAC, $p = 1 * 10^{-9}$ for SGIM-ACTS. The complete results for Student's t-test are reported in Table 4.1. **The algorithms capable of performing procedures (IM-PB and SGIM-PB) have errors that drop to levels lower than the their non-procedure equivalents (SAGG-RIAC and SGIM-ACTS)**. The t-test comparing the final errors of IM-PB and SGIM-PB vs SAGG-RIAC and SGIM-ACTS gives a strong difference with $p = 9e - 4 < 0.1$. Moreover, this difference starts since the beginning of the learning process (shown on Fig. 4.8). It seems that the procedures bootstrap the exploration, enabling the learner to progress further. Indeed, the autonomous learner

IM-PB learner, the upgraded version of SAGG-RIAC by the use of procedures, has significantly better performance.

We can also see that the SGIM-PB algorithm has a very quick improvement in global evaluation owing to the bootstrapping effect of the different teachers. It goes lower to the final evaluation of SAGG-RIAC (0.17) after only 500 iterations. This bootstrapping effect comes from the mimicry teachers, as it is also observed for SGIM-ACTS which shares the same mimicry teachers.

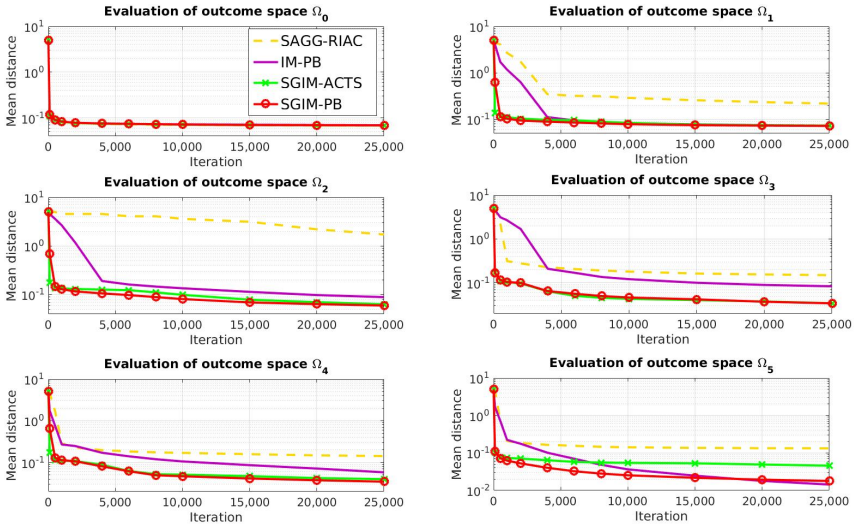

FIGURE 4.9: Evaluation of all algorithms per outcome space (for Ω_0, all evaluations are superposed)

If we look at the evaluation on each individual outcome space (Fig. 4.9), we can see that the **learners with demonstrations (SGIM-PB and SGIM-ACTS) outperform the other algorithms, except for the most simple outcome space** Ω_0, which does not require sequences of actions, and the outcome space Ω_5. In the case of Ω_5, the difference with IM-PB is not significative (IM-PB seems a bit better but the difference is not significative with $p > 0.1$). The results for Student's t-test are reported in Table 4.1. This exception for Ω_5 is due to the fact that IM-PB practiced much more on this outcome space (1500 iterations where it chose goals in Ω_5 against 160 for SGIM-PB). SGIM-PB and SGIM-ACTS are much better than the other algorithms on the two joysticks outcome spaces (Ω_3 and Ω_4) (with respectively p=7e-4 and 1e-5). This is not surprising given the fact that those outcome spaces require precise actions. Indeed, if the end-effector gets out of the area where it can control the joystick, the latter is released, thus potentially ruining the attempt. So on these outcome spaces working directly on carefully crafted actions can alleviate this problem, while using procedures might be tricky, as the outcomes used don't take into account the motion trajectory but merely its final state. SGIM-PB was provided with such actions by the action teachers. Also if we compare the results of the autonomous learner without procedures (SAGG-RIAC) with the one with procedures (IM-PB), we can see that it learns less on any outcome space but Ω_0 (which was the only outcome space reachable using only single primitive actions and that could not benefit from using the task hierarchy to be learned) and especially for Ω_1, Ω_2 and Ω_5 which were the most hierarchical in this setup. More generally, it seems than on this highly hierarchical Ω_5, the learners with procedures were better. So the **procedures helped when learning any potentially hierarchical task** in this experiment.

		global	task0	task 1	task 2	task 3	task 4	task 5
SGIM-PB vs random	t	-33	9	-27	-15	-32	-50	-57
	p	3e-16	5e-8	9e-15	4e-11	4e-16	6e-19	5e-20
SGIM-PB vs SAGG-RIAC	t	-3	9	-10	-2	-44	-46	-84
	p	1e-2	6e-8	1e-8	3e-2	4e-18	2e-18	1e-22
SGIM-PB vs IM-PB	t	-11	-4	-4	-5	-5	-3	1
	p	3e-9	4e-4	1e-3	1e-4	9e-5	3e-3	0.2
SGIM-PB vs SGIM-ACTS	t	-12	5	-3	-3	-0.5	-3	-18
	p	1e-9	2e-4	2e-3	1e-2	6e-2	1e-2	2e-12
(SGIM-PB, IM-PB) vs	t	-2.5	9	-5	-2	-4	-5	-8
(random, SAGG-RIAC, SGIM-ACTS)	p	2e-2	1e-12	3e-6	7e-2	6e-4	3e-6	4e-11

TABLE 4.1: Student's t-test on two samples for comparing SGIM-PB with each of the algorithms and for comparing the procedure algorithms (SGIM-PB and IM-PB) to algorithms without the procedure framework (SGIM-ACTS, SAGG-RIAC and random). I tested the difference of the distances to goal at the end of the learning (t=25,000) for the global evaluation and for each task type. Negative values for t mean that SGIM-PB makes lower error. The non-significative results ($p > 0.1$) are hightlighted.

Analysis of the sampling strategy chosen for each goal

I further analyzed the results of my SGIM-PB learner. I looked in its learning process to see which pairs of teachers and target outcomes it has chosen (Fig. 4.10). It was capable to **request demonstrations from the relevant teachers** depending on the task at hand, except for the outcome space Ω_0 which had no human teachers and therefore could not find a better teacher to help it. Indeed, for the outcome space Ω_2, the procedural teacher (ProceduralTeacher2) specially built for this outcome space was greatly chosen.

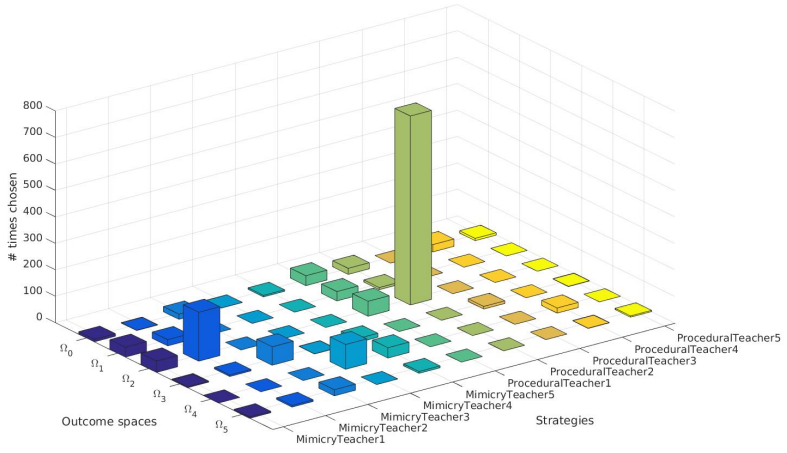

FIGURE 4.10: Choices of teachers and target outcomes of the SGIM-PB learner

I wanted to see if my SGIM-PB learner adapts the complexity of its action sequences to the working task. So I looked which action space would be chosen by the local optimization function (used inside the action space exploration strategy) for the Ω_0, Ω_1 and Ω_2 subspaces (chosen because they are increasingly complex) on their respective evaluation benchmarks. I compared those results with the same obtained

by the IM-PB learner in Fig. 4.6 to see if the teachers had an effect on the complexity of action sequences produced. Fig. 4.11 shows the results of this analysis.

Length of the sequence of primitive actions

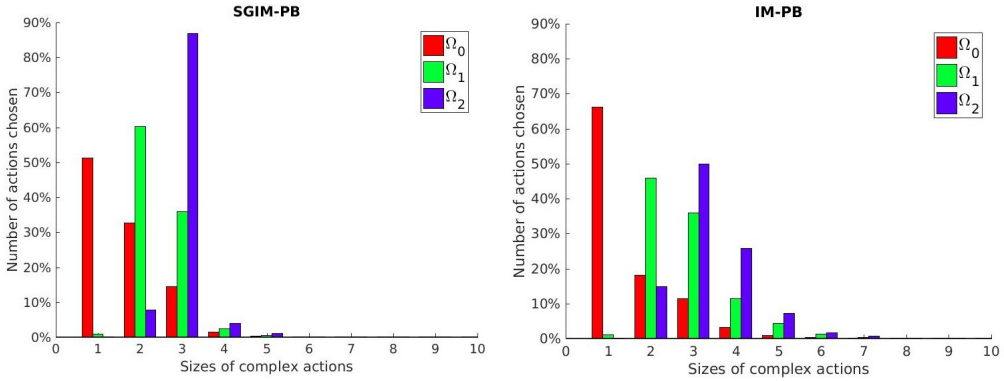

FIGURE 4.11: Number of actions selected per action size for three increasingly more complex outcome spaces by the SGIM-PB (on the left) and IM-PB (on the right) learners

As we can see on those three interrelated outcome subspaces (Fig. 4.11), **the learner is capable to adapt the complexity of its action sequences to the outcome at hand**. It chooses longer actions for the Ω_1 subspace (actions of size 2 and 3 while using mostly actions of size 1 and 2 for Ω_0) and even longer for the Ω_2 subspace (using far more actions of size 3 than for the others). It shows that my learner is capable to correctly limit the complexity of its action sequences instead of being stuck into always trying longer and longer actions. Also, if we look at the action sequence complexity of the IM-PB learner, we see it was also capable to correctly limit its complexity (especially on Ω_0 where it used even more single-primitive actions than SGIM-PB). However, we can see that the SGIM-PB learner, owing to the teacher strategies available to it, had a smaller spread on the size of action sequences distribution for each of the three outcome spaces.

I also wanted to see if my SGIM-PB algorithm had discovered the task hierarchy of this experiment. I hoped it would correctly assess which procedural space is adapted to each of the complex outcome subspaces (all subspaces except Ω_0 as it cannot benefit from procedures to be reached). So I looked which procedural space was selected by the local optimization function (used inside the procedural space exploration strategy) for each of the outcome subspaces on their respective evaluation benchmarks. For assessing those results, I compared them with those obtained by the IM-PB learner on the same process.

As we can see on left column of Fig. 4.12, the **SGIM-PB learner successfully chooses the procedural spaces most adapted for each complex outcome subspace** (the same as those I used to build the procedural teachers). For instance, to move the video character (task Ω_5), the robot mainly uses subtasks Ω_4 (position of the second joystick) and Ω_3 (position of the first joystick). To move the position of the first joystick (task Ω_3), subtasks Ω_0 (position of the end-effector) and Ω_3 (position of the first joystick) are used. The same way, task Ω_4 recruits subtasks Ω_0 and Ω_4. Thus by recursively, the robot has built a hierarchical representation that task Ω_5 depends on subtasks $(\Omega_0, \Omega_4, \Omega_0, \Omega_3)$. This means it was successfully able to discover and exploit it. By comparison, the IM-PB learner was only capable to identify

useful procedural spaces for the Ω_1 and Ω_2 outcome subspaces. For both those outcome subspaces, it identified the one procedural space mainly used by SGIM-PB learner and another one (Ω_2, Ω_0) which can also be useful to learn to reach those outcome subspaces, though arguably less efficient. Indeed, using a action moving the pen (in Ω_1) is enough for the first component of procedures used to reach Ω_1 and Ω_2, and it can lead to less complex action sequences than using one drawing on the floor (in Ω_2). If we look at the result for the outcome subspaces Ω_3 and Ω_4, the IM-PB learner was incapable to identify adapted procedural spaces. The absence of a teacher to guide it could explain the IM-PB learner poor results on those outcome subspaces. Also, compared to the great focus of the SGIM-PB learner on this outcome subspaces, IM-PB results were more dispersed, indicating its difficulty to select an adapted procedural space. As those outcome subspaces require precise actions and are less adapted to procedures, this difficulty is understandable. By looking at the results of both learners, we can see that the **procedural teachers had a profound impact on the choice of adapted procedures for each outcome subspaces**, and clearly guided its whole learning process by helping it discover the task hierarchy of the experimental setup.

4.4.5 Conclusion

Both IM-PB and SGIM-PB learner proved they could tackle the learning of a set of multiple hierarchically organized tasks using sequences of motor actions. These results prove my procedural framework enables the discovery and exploitation of the task hierarchy, and helps the learner to explore further its environment. Both algorithms were able to outperform their non-procedural respective competitors (SGIM-ACTS for SGIM-PB and SAGG-RIAC for IM-PB) on this setup. My SGIM-PB learner, also benefited from its interactive strategies, which by making it focus on a subset of the action space or the procedural space, bootstrapped its learning process and made it learn the task hierarchy better than IM-PB did on its own.

However, my setup was only a simulation and did not consider a real physical robot in a realistic setup. Therefore, in the next chapter, I propose such a realistic setup, and test my algorithm SGIM-PB on it.

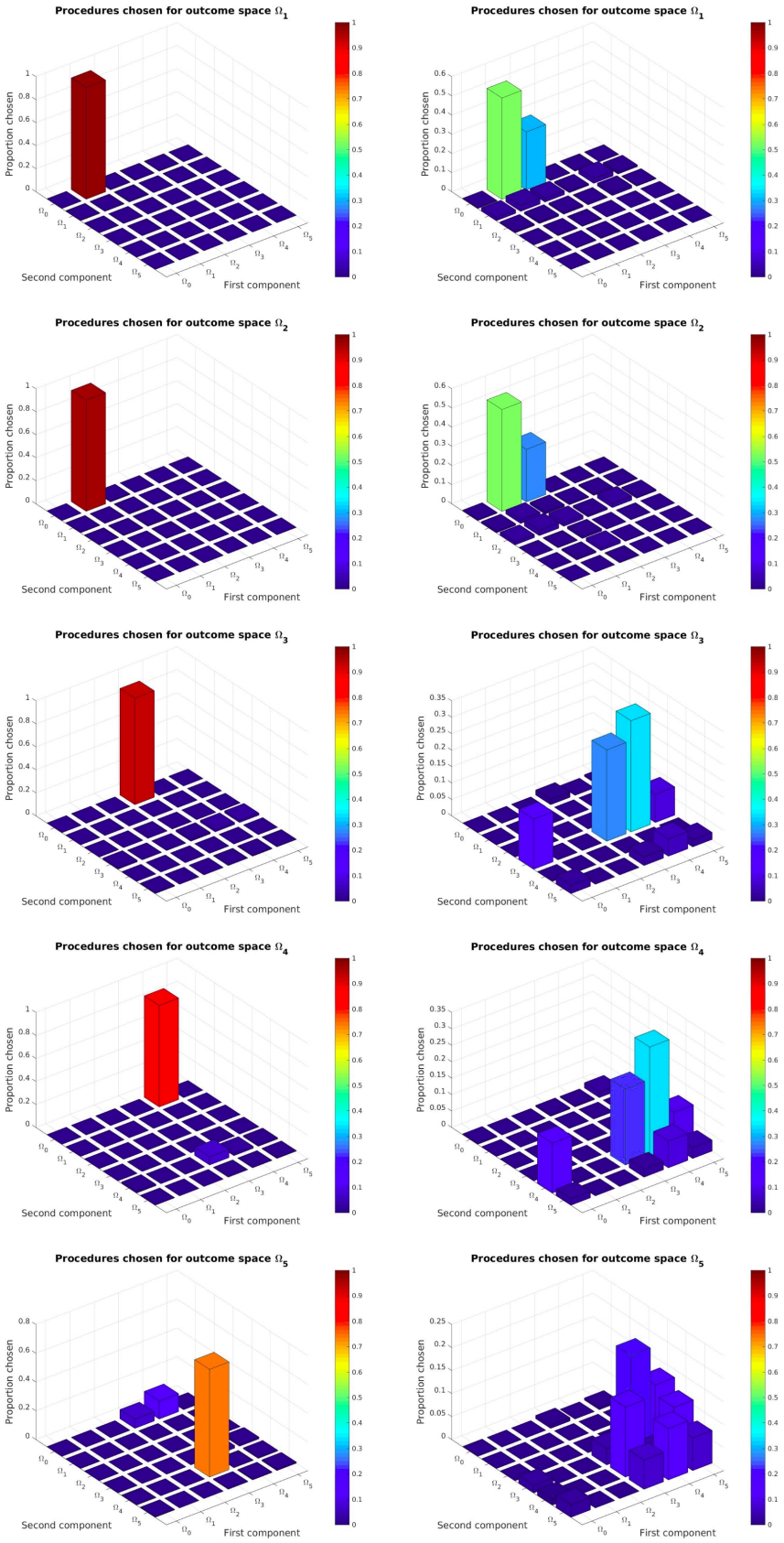

FIGURE 4.12: Task hierarchy discovered by the SGIM-PB (left side) and IM-PB (right side) learners: this represents for each complex outcome space the percentage of time each procedural space would be chosen

Chapter 5

Yumi industrial robot learning complex hierarchical tasks on a tangible interactive table

In this chapter, I am interested in replicating the results I observed in the previous chapter, but this time using a **real-world robot, the Yumi robot from ABB, in a realistic setup**. I study first a simulated version of this real-world setup. I want to see if my algorithm is still relevant in the more realistic context on an industrial robot. Then I study the real physical version of this setup. The testing on the real physical setup being very long (more than one month for one run of one learning algorithm), I decided to emphasize on the simulation results, which were faster to obtain (less than 10 days per run). However, I wanted to see if using those simulation results are relevant by performing one run for the two best learning algorithms in simulation on the actual real setup. Afterwords, I describe a way of introducing transfer learning in the simulated setup, so as to bootstrap the learning of my SGIM-PB agent. The idea is to transfer the usable and relevant knowledge, from a previous learning process, to a new learner with a different learning context. In this exploratory work, I looked at a small change where a two-arms robot, previously trained on its right arm has to start again only with its left arm.

5.1 Simulated experiment

In this part, I describe the experimental setup using a **real-life industrial robot**, on which I compare my different algorithms. I want to confirm the results obtained in Chapter 4 on a realistic setup.

I designed an experimental setup, in which the 7 DOF right arm of an industrial Yumi robot by ABB can interact with an interactive table and its virtual objects. It can learn an infinite number of hierarchically organized tasks regrouped in 5 types of tasks, using sequences of motor actions of unrestricted size. This experimental setup was first introduced in (Duminy, Nguyen, and Duhaut, 2018a).

5.1.1 Setup

Fig. 5.1 shows the robot facing an interactive table. The robot learns to interact with it using the tip of its arm (the tip of the vacuum pump below its hand). The position of the arm's tip on the table is noted (x_0, y_0). Two virtual objects (disks of radius $R = 4cm$) can be picked and placed, by placing the arm's tip on them and moving it at another position on the table. Once interacted with, the final positions of the two objects are given to the robot by the table, respectively (x_1, y_1) and (x_2, y_2).

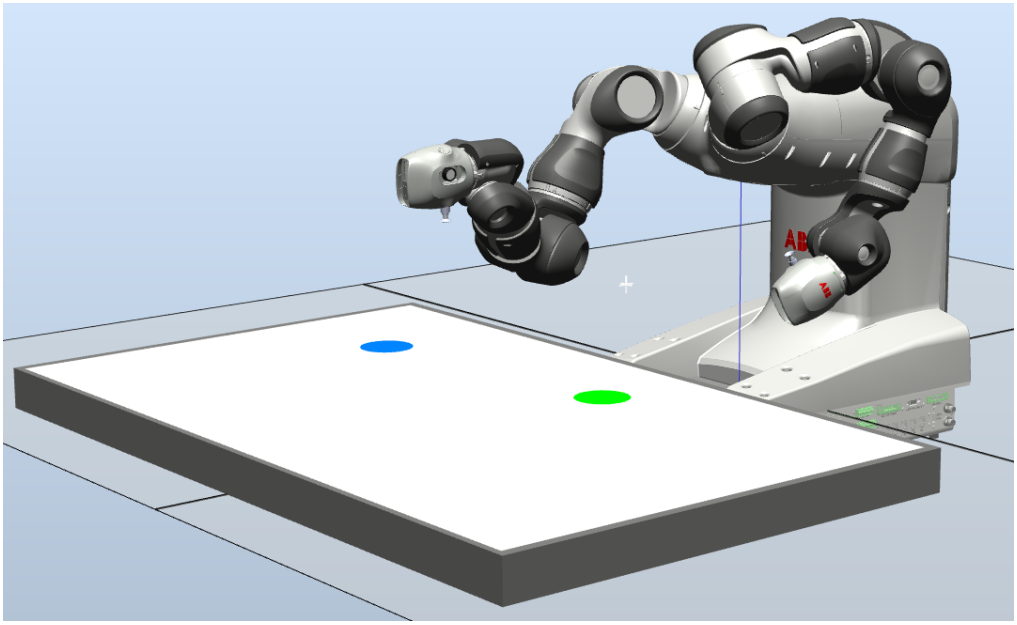

FIGURE 5.1: Experimental setup for the Yumi simulated experiment

Only one object can be moved at a time, otherwise the setup is blocked and the robot's motion cancelled. If both objects have been moved, a sound is emitted by the interactive table, parametrised by its frequency f, its intensity level l and its rhythm b. The emitted sound depends on the relative position of both objects and the absolute position of the first object. The sound parameters are computed as follow:

$$f = (D/4 - d_{min})4/D \tag{5.1}$$
$$l = 1 - 2(log(r) - log(r_{min}))/(log(D) - log(r_{min})) \tag{5.2}$$
$$b = (|\varphi|/\pi) * 0.95 + 0.05 \tag{5.3}$$

where D is the diagonal of the interactive table, $r_{min} = 2R$, (r, φ) the polar coordinate of the second object in the system centred on the first one, and d_{min} is the distance between the first object and the closest table corner (see Fig. 5.3).

The motions of the Yumi robot are executed using a physical simulation (using the Robotstudio software by ABB). The interactive table and its behaviour is simulated and its state is refreshed after each primitive motor action executed. The robot is not allowed to collide with the interactive table. In this case, the motor action is cancelled and reaches no outcomes. The arm itself has 7 DOF. Before each attempt, the robot is set to its initial position and the environment is reset.

5.1.2 Experiment variables

Action spaces

The motions of each joint are controlled by Dynamic Movement Primitives (DMP). To each joint is attached a one dimensional DMP a_i controlling it, parametrised by the end joint angle $g^{(i)}$, and one basis functions for the forcing term, parametrized by its weight $w^{(i)}$. I am using the original form of the DMP from Pastor et al., 2009

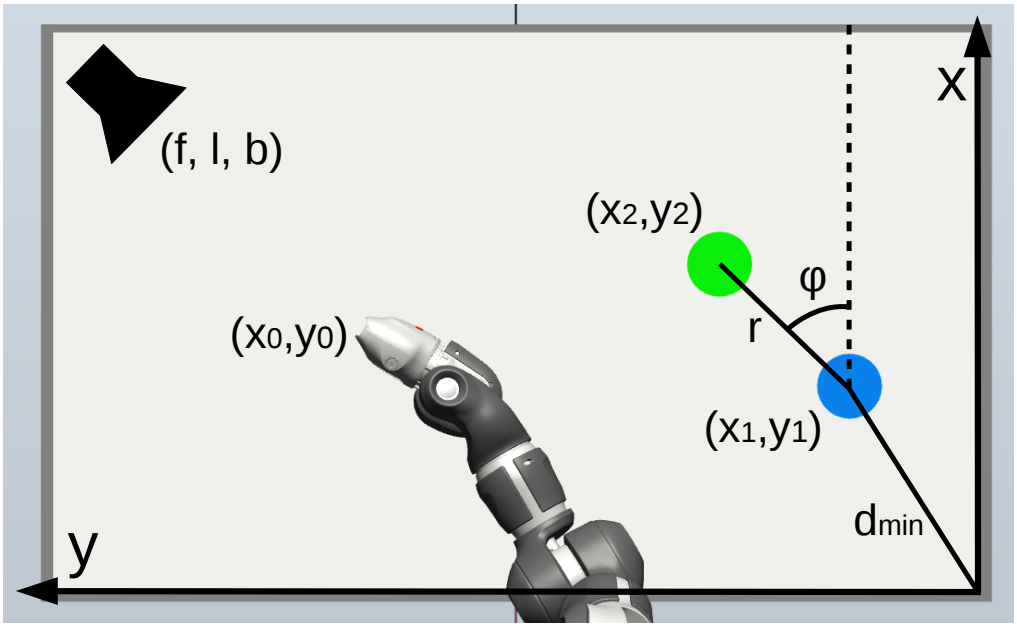

FIGURE 5.2: Representation of the interactive table: the first object is in blue, the second one in green, the produced sound is also represented in top left corner

and I keep the same notations. A primitive motor action is simply the concatenation of those DMP parameters for all joints:

$$\theta = (a_0, a_1, a_2, a_3, a_4, a_5, a_6) \tag{5.4}$$

$$where\ a_i = (w^{(i)}, g^{(i)}) \tag{5.5}$$

Two or more primitive actions $(\pi_{\theta_0}, \pi_{\theta_1}, ...)$ can be combined in an action sequence π.

Task spaces

The task spaces the robot learns are hierarchically organized:

- $\Omega_0 = \{(x_0, y_0)\}$: the positions touched by the robot on the table;
- $\Omega_1 = \{(x_1, y_1)\}$: the positions where the robot placed the first object on the table;
- $\Omega_2 = \{(x_2, y_2)\}$: the positions where the robot placed the second object on the table;
- $\Omega_3 = \{(x_1, y_1, x_2, y_2)\}$: the positions where the robot placed both objects;
- $\Omega_4 = \{(f, l, b)\}$: the sounds produced by the table;

The outcome space is a composite and continuous space $\Omega = \bigcup_{i=0}^{5} \Omega_i$, containing subspaces of 2 to 4 dimensions. Multiple interdependencies are present between tasks: controlling the position of either the blue object (Ω_1) or the green object (Ω_2) comes after being able to touch the table at a given position (Ω_0); moving both objects

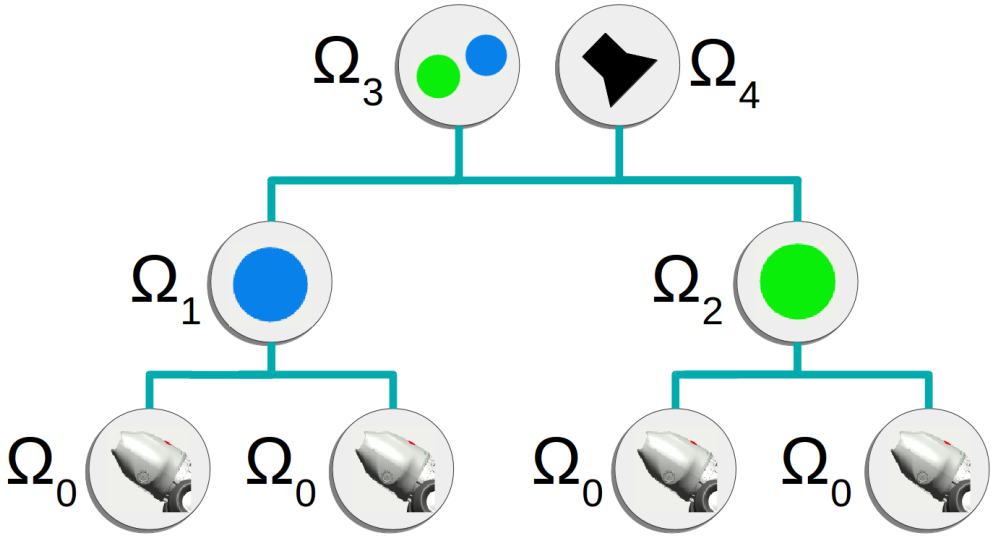

FIGURE 5.3: Representation of task hierarchy of the simulated Yumi experimental setup

(Ω_3) or making a sound (Ω_4) comes after being able to move the blue (Ω_1) and the green object (Ω_2). This task hierarchy is represented on Fig.**??**.

In this setup, my intuition is that a learning agent should start by making a good progress in the easiest task Ω_0, then Ω_1, Ω_2. Once it mastered those easy tasks, it can reuse that knowledge to learn to achieve the most complex tasks Ω_3 and Ω_4. I will particularly focus on the learning of the Ω_4 task space and the use of the procedure framework for it. Indeed in this setup, the relationship between a goal outcome in Ω_4 and the necessary positions of both objects (Ω_1, Ω_2) to reach that goal are not linear. So with this setup, I test if the robot can **learn a non-linear mapping between a complex task and a procedural space**.

5.1.3 The teachers

To help the SGIM-PB learner, procedural teachers (with a strategical cost $K(\sigma) = 5$) were available for every outcome space except Ω_0. Each teacher is only capable to give procedures according to its outcome space of expertise, knows the task hierarchy and indicate procedures according to a construction rule:

- ProceduralTeacher1 ($\omega_1 \in \Omega_1$): (ω_0, ω_0') where $\omega_0 \in \Omega_0$ is equal to the initial position of the first object on the table, and $\omega_0' = \omega_1 \in \Omega_0$ to its desired final position;
- ProceduralTeacher2 ($\omega_2 \in \Omega_2$): (ω_0, ω_0') where $\omega_0 \in \Omega_0$ is equal to the initial position of the second object on the table, and $\omega_0' = \omega_2 \in \Omega_0$ to its desired final position;
- ProceduralTeacher3 ($\omega_3 = (x_1, y_1, x_2, y_2) \in \Omega_3$): (ω_1, ω_2) where $\omega_1 = (x_1, y_1) \in \Omega_1$ is equal to the first object desired final position on the table, and $\omega_2 = (x_2, y_2) \in \Omega_2$ to that of the second one;
- ProceduralTeacher4 ($\omega_4 \in \Omega_4$): (ω_1, ω_2), where $\omega_1 \in \Omega_1$ is the final position of the first object, chosen as to both be on the semi-diagonal going from bottom-right corner to the centre of the table and corresponding to the desired sound

frequency, and $\omega_2 \in \Omega_2$ is the final position of the second object which relative position to first one corresponds to the desired sound level and rhythm.

I also added different configurations of action teachers (with a strategical cost of $K(\sigma) = 10$), each expert of one outcome space:

- ActionTeacher0 (Ω_0): 11 demos of primitive actions;
- ActionTeacher1 (Ω_1): 10 demos of size 2 actions;
- ActionTeacher2 (Ω_2): 8 demos of size 2 actions;
- ActionTeacher34 ($\Omega_3 \times \Omega_4$): 73 demos of size 4 actions.

5.1.4 Evaluation method

To evaluate my algorithm, I created a benchmark linearly distributed across the Ω_i, of 19,200 points. The evaluation consists in computing mean Euclidean distance between each of the benchmark outcomes and their nearest neighbour in the learner dataset. When the learner is incapable to at least reach the outcome space, the evaluation is set to 5. The evaluation is repeated regularly across the learning process.

Then to assess the efficiency of my algorithm, I am comparing the results of the following algorithms:

- RandomAction: random exploration of the action space $\Pi^{\mathbb{N}}$;
- IM-PB: autonomous exploration of the action space $\Pi^{\mathbb{N}}$ and the procedural space Ω^2 driven by intrinsic motivation;
- SGIM-ACTS: interactive learner driven by intrinsic motivation. Choosing between autonomous exploration of the action space $\Pi^{\mathbb{N}}$ and mimicry of one of the action teachers;
- SGIM-PB: interactive learner driven by intrinsic motivation. Choosing between autonomous exploration strategies (either of the action space or the procedural space) and mimicry of one of the available teachers procedural teachers and ActionTeacher0.

Each algorithm was run 10 times (results averaged on all runs). I also added another result as a threshold corresponding to the evaluation of a learner knowing only the combined skills of every action teachers for the whole learning process, called Teachers. Each run takes a total average of 7 days to complete the 25,000 learning iterations. The code used for this experiment is available at `https://bitbucket.org/smartan117/sgim-yumi-simu`.

5.1.5 Results

Evaluation performance

The Fig. 5.4 shows the global evaluation of all tested algorithms, which corresponds to the mean error made by each algorithm to reproduce the benchmarks with respect to the number of complete sequences of motor actions tried during the learning. We can see that both autonomous learners (RandomAction and IM-PB) have higher final levels of error than the others, which shows this setup was tough to learn without

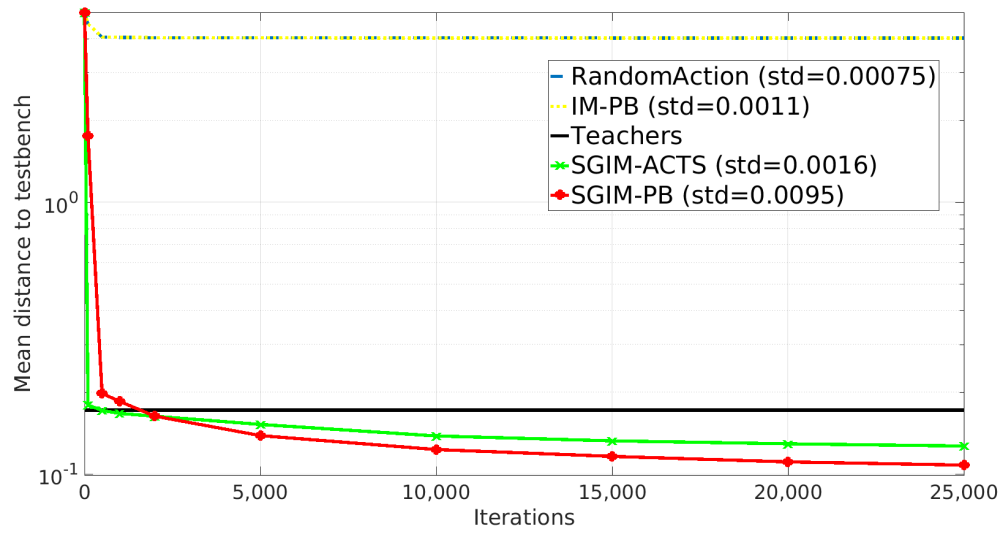

FIGURE 5.4: Evaluation of all algorithms throughout learning process, final standard deviations are given in the legend

demonstrations. I also show that both the SGIM-PB and the SGIM-ACTS learners have errors dropping lower than the Teachers result (in black), showing they went further than the provided action demonstrations. And if we look at the values of final standard deviation for each algorithm, we can see that these results are pretty consistent among all runs. Also both have about the same final evaluation, SGIM-PB even slightly outperforming SGIM-ACTS, showing that procedural teachers can replace action teachers for helping learning complex tasks. If we look at the evaluation per outcome space, on Fig. 5.5, we also see that both autonomous learners were not able to move any of the objects as they did not reach any of the complex outcome spaces $\Omega_1, \Omega_2, \Omega_3, \Omega_4$. Moreover, both SGIM learners have similar final evaluation measures for the $\Omega_0, \Omega_1, \Omega_2$ spaces and SGIM-PB outperforms SGIM-ACTS on the most complex tasks Ω_3, Ω_4. Thus, **procedural teachers are well adapted to tackle the most complex and hierarchical outcome spaces**.

Analysis of the sampling strategy chosen for each goal

If we look at the learning process of the SGIM-PB learner, we can see the proportion of strategical choices made by the learner at the beginning of each episode. Fig. 5.6 shows those choices per outcome space and strategy, and we see that the **SGIM-PB learner was capable to organize its learning process**. We can see that the learner spent most of its time learning the most complex outcome spaces Ω_3, Ω_4 and especially the highest dimension space Ω_3. Also the learner spent most time using autonomous exploration strategies, which reduces the need for the teachers attendance. We also see that the learner explored mostly the procedural space for the most complex outcome spaces Ω_3, Ω_4, while more relying on action exploration for the least complex outcome space Ω_0. We can also see that the learner figured on the overall which teacher was more appropriate for each outcome space. Even though it used almost equally ProceduralTeacher 3 and 4 for the Ω_4 space as those spaces are related.

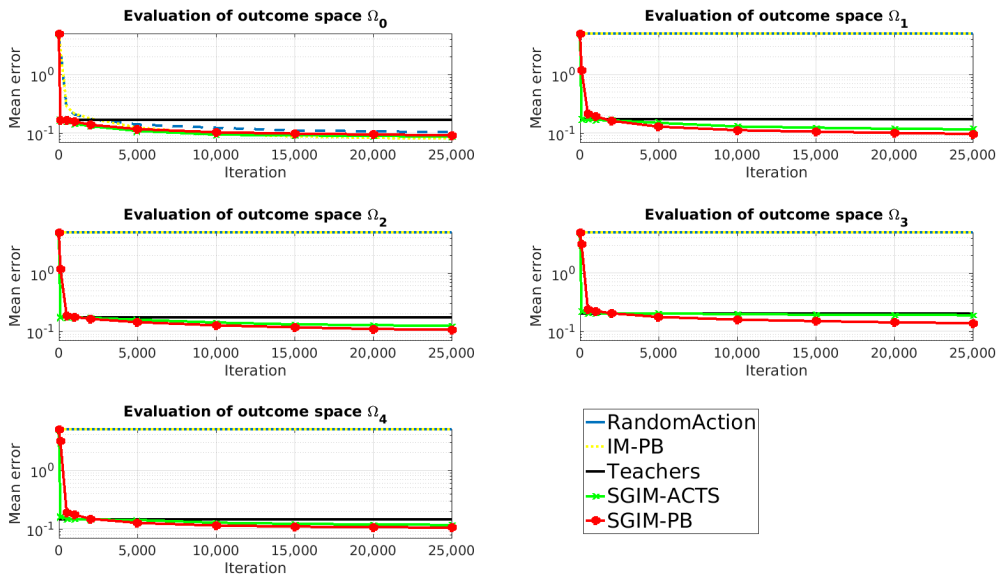

FIGURE 5.5: Evaluation of all algorithms per outcome space (RandomAction and IM-PB are superposed on all evaluations except for Ω_0)

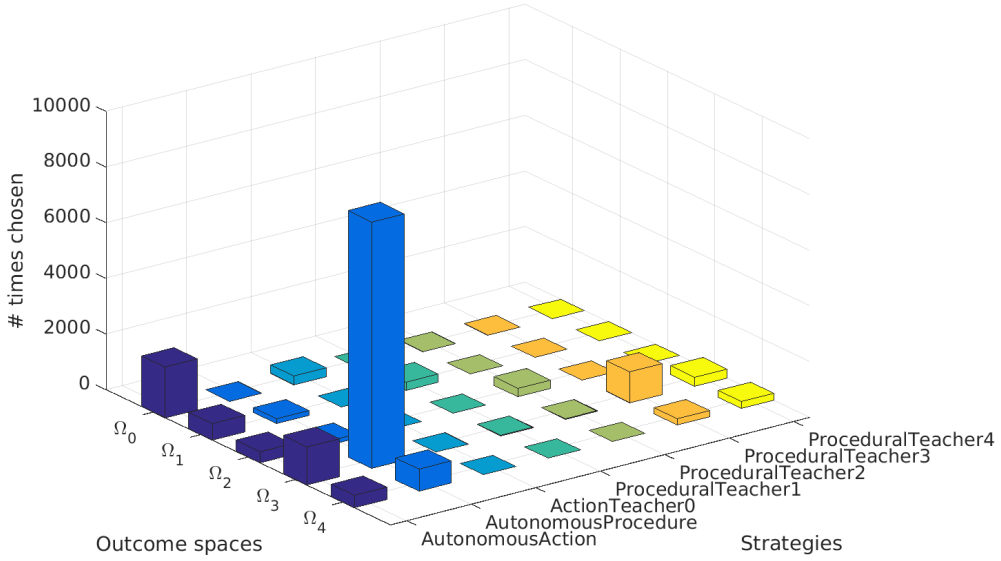

FIGURE 5.6: Choices of strategy and goal outcome for the SGIM-PB learner

Length of the sequence of primitive actions

To see if the SGIM-PB learner was capable to adapt the complexity of its actions to the task at hand, I analysed which action size would be chosen by the local action optimization function, for each point of the evaluation testbench. I computed this percentage for three outcome spaces of increasing complexity : Ω_0, Ω_1 and Ω_4. I showed it on Fig. 5.7. We can see that **SGIM-PB is able to limit the size of its actions**: using mostly primitive actions and 2-primitive actions for Ω_0, 2-primitive actions for Ω_1, and 4-primitive actions for Ω_4. Although, it could be wondered why

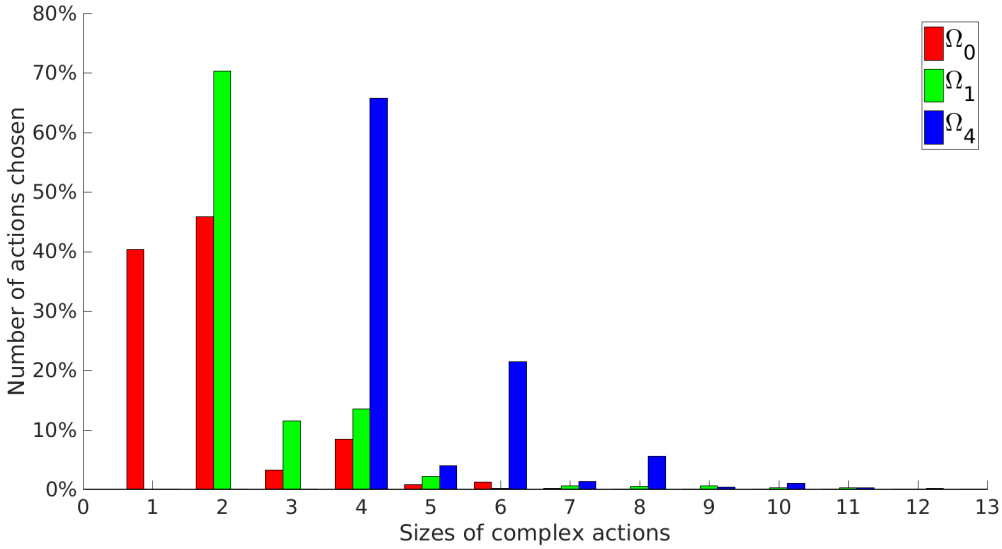

FIGURE 5.7: Percentage of actions chosen per action size by the SGIM-PB learner for each outcome space

the Ω_0 outcome space had been associated with size 2 actions, and not only primitives. This is certainly due to the fact that SGIM-PB set goals in the Ω_0 outcome space far fewer times than on the more complex outcome spaces (2000 times against more than 18,000 times for Ω_3 and Ω_4). So it tried a lot of action sequences which reached Ω_0 as any action that moves any object or makes sound ($\Omega_1, \Omega_2, \Omega_3, \Omega_4$) also touches the table.

5.1.6 Conclusion

This experiment showed the SGIM-PB learner is adapted to learn a set of multiple hierarchical tasks on such a real-world setup. It is capable to learn better than the other algorithms, but also it learns quickly, owing to the bootstrapping effect of the teachers. The interesting observation we can do is that, even deprived of action teachers for the complex tasks Ω_1, Ω_2, Ω_3 and Ω_4, the SGIM-PB learner was capable to outperforms the other algorithms on these outcome subspaces. This shows that in a setup with a hierarchical set of tasks, the procedural teachers are sufficient to bootstrap the learning process. They even enable SGIM-PB to outperform the learner SGIM-ACTS which had action teachers for such tasks. **This confirms the results I obtained on Chapter 4 indicating that both teachers are complementary and procedural teachers are particularly good with the most complex tasks**.

Without surprise for us, the SGIM-SAHT architecture is still able to self-organize its learning process on this realistic setup. The SGIM-PB learner correctly assessing the teachers' domains of expertise, and switching from exploring actions mainly for the simplest tasks Ω_0, Ω_1 and Ω_2 to exploring mainly procedures for the most complex ones Ω_3 and Ω_4.

It is also capable of adapting the complexity of its action sequences to that of the task at hand. Although it is not perfect at it, it still figures out the optimal action sequence size to use depending on the task in average. In next section, I compare both SGIM-PB and SGIM-ACTS (the second better learning algorithm in this experiment) on a physical version of this setup. The setup has been slightly modified with the addition of a more complex outcome subspace, for which both learners are deprived

of human advice. This so to confirm my belief that the procedural framework, owing to the possibility to combine previous knowledge, is well suited to explore such a space.

5.2 Physical experimental setup

In this experiment, I want to compare both SGIM-ACTS and SGIM-PB on a physical setup. As mentioned earlier, the setup was slightly modified with the addition of a new task more complex, for which no teacher repertoire is built. This modification was added to see whether my SGIM-PB learner is able to tackle such a highly hierarchical space autonomously better thanks to the autonomous exploration of the procedure space strategy. Another change I made was to alter the procedural teacher strategy so as to make it more similar to the action teachers, with a repertoire of procedural demonstrations, which I compare with the repertoires of actions for the teachers of SGIM-ACTS. The outcomes of the repertoires of both kinds of teachers are identical to see if **SGIM-PB still performs better on the complex tasks, with a limited number of procedure demonstrations**.

5.2.1 Description of the environment

This experimental setup is quite similar to the one with a simulated Yumi robot described in section 5.1. Only this time a real Yumi robot is used in conjunction with a real interactive table. The interactive table used for this experiment is described in (Kubicki, Lepreux, and Kolski, 2012) (Kubicki et al., 2016). The dimensions of the table are the same than the simulated one and the objects, also virtual and managed by the interactive table, are positioned at the same spots. The only addition to the setup was a **sixth type of outcome in the form of a maintained sound**. After moving both objects on the table, the same sound (f, l, b) is emitted in a burst, but if the arm's tip is detected by the table to a new position on the table, the sound is now maintained for a duration t proportional to the distance between the arm's tip detected position and the second object current position: $t = d_2/D$ where d_2 is the distance between the arm's tip and object 2 on the table and D is the table diagonal. A picture of this setup is shown on Fig. 5.8.

5.2.2 Formalization of tasks and actions

The actions are encoded using Dynamic Movement Primitives, as described in 5.1.2 with the notations used in 5.4.

The task spaces the robot learns are hierarchically organized:

- $\Omega_0 = \{(x_0, y_0)\}$: the positions touched by the robot on the table;
- $\Omega_1 = \{(x_1, y_1)\}$: the positions where the robot placed the first object on the table;
- $\Omega_2 = \{(x_2, y_2)\}$: the positions where the robot placed the second object on the table;
- $\Omega_3 = \{(x_1, y_1, x_2, y_2)\}$: the positions where the robot placed both objects;
- $\Omega_4 = \{(f, l, b)\}$: the burst sounds produced by the table;
- $\Omega_5 = \{(f, l, b, t)\}$: the maintained sound produced by the table.

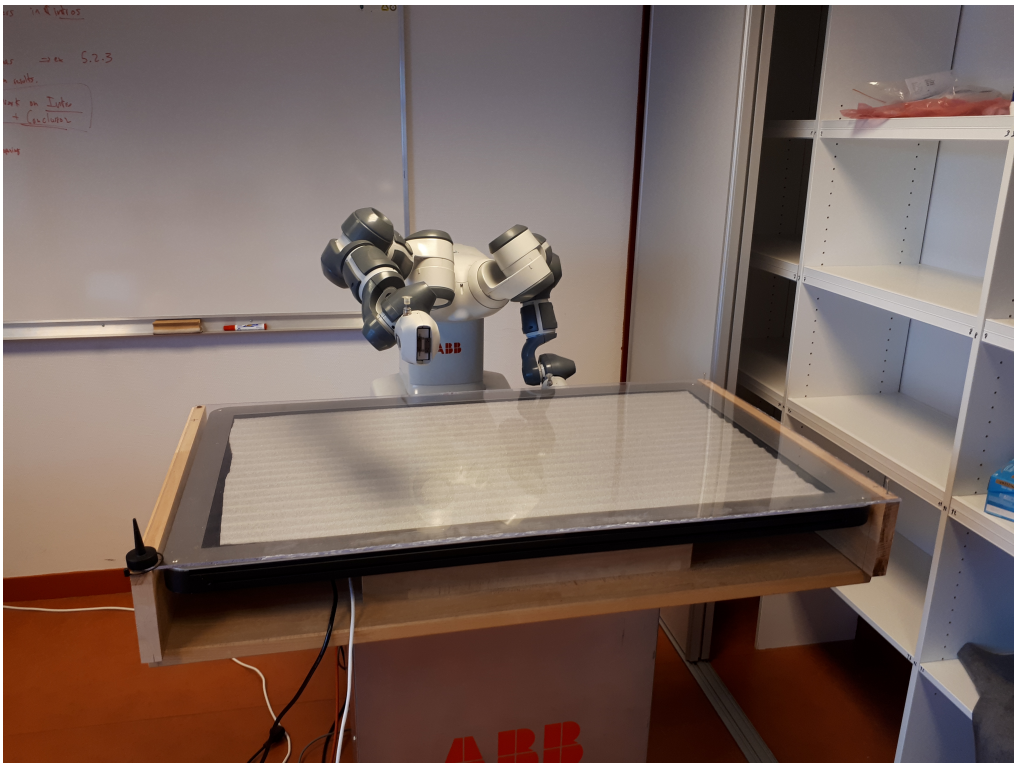

FIGURE 5.8: Real Yumi setup

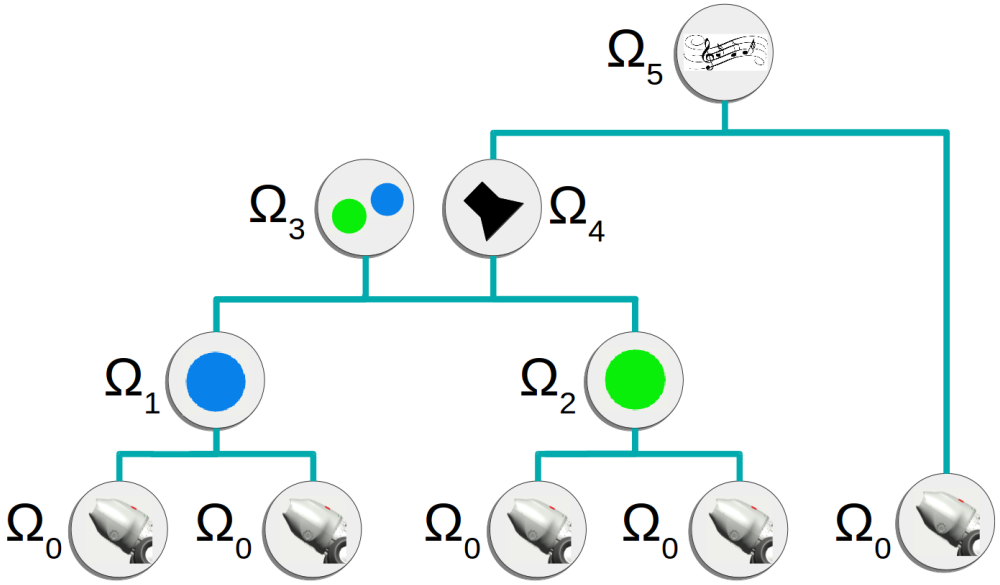

FIGURE 5.9: Representation of task hierarchy of the real physical Yumi experimental setup

The outcome space is composite and continuous: $\Omega = \bigcup_{i=0}^{5} \Omega_i$. This task hierarchy is similar to the one presented in section 5.1.2, with the addition of a higher hierarchical task, which is the maintained sound Ω_5. It is is represented on Fig. 5.9.

5.2.3 Teachers

In this experiment, I wanted to delve into the differences between SGIM-ACTS and SGIM-PB in terms of learning ability. So as to put them on an equal footing, I changed the procedural teacher strategy, which now works the same way that the action teachers do. Instead as a function building an adapted procedure on the fly to the learner's request, the **procedural teacher now has a demonstration dataset**, and provides the procedure reaching the closest outcome to the one asked. To maximize their similarity in knowledge, I built the teachers dataset for 4 of the most complex tasks ($\Omega_1, \Omega_2, \Omega_3$ and Ω_4) for both action and procedural teachers the same way. So each of those teachers have the demonstrations reaching the same outcomes respectively. An extra action teacher was added to provide demonstration for the simplest outcome space Ω_0:

- ActionTeacher0 (Ω_0): 9 demonstrations of primitive actions;
- ActionTeacher1 (Ω_1): 7 demonstrations of size 2 actions;
- ActionTeacher2 (Ω_2): 7 demonstrations of size 2 actions;
- ActionTeacher3 (Ω_3): 32 demonstrations of size 4 actions;
- ActionTeacher4 (Ω_4): 7 demonstrations of size 4 actions;

The corresponding demonstrations for the procedural teachers, correspond to the way the primitive actions from ActionTeacher0 were composed together to build the demonstration repertoires for the teachers of the complex tasks:

- ProceduralTeacher1 ($\omega_1 \in \Omega_1$): (ω_0, ω_0') where $\omega_0 \in \Omega_0$ is equal to the initial position of the first object on the table, and $\omega_0' = \omega_1 \in \Omega_0$ to its desired final position;
- ProceduralTeacher2 ($\omega_2 \in \Omega_2$): (ω_0, ω_0') where $\omega_0 \in \Omega_0$ is equal to the initial position of the second object on the table, and $\omega_0' = \omega_2 \in \Omega_0$ to its desired final position;
- ProceduralTeacher3 ($\omega_3 = (x_1, y_1, x_2, y_2) \in \Omega_3$): (ω_1, ω_2) where $\omega_1 = (x_1, y_1) \in \Omega_1$ is equal to the first object desired final position on the table, and $\omega_2 = (x_2, y_2) \in \Omega_2$ to that of the second one;
- ProceduralTeacher4 ($\omega_4 \in \Omega_4$): (ω_1, ω_2), where $\omega_1 \in \Omega_1$ is the final position of the first object, chosen as to both be on the semi-diagonal going from bottom-right corner to the centre of the table and corresponding to the desired sound frequency, and $\omega_2 \in \Omega_2$ is the final position of the second object which relative position to first one corresponds to the desired sound level and rhythm.

The action teachers were provided to the SGIM-ACTS learner, while the SGIM-PB algorithm had all procedural teachers and the action teacher for Ω_0. No teacher was provided to both learners for the most complex outcome space Ω_5, as to compare the autonomous exploration capability of both learners.

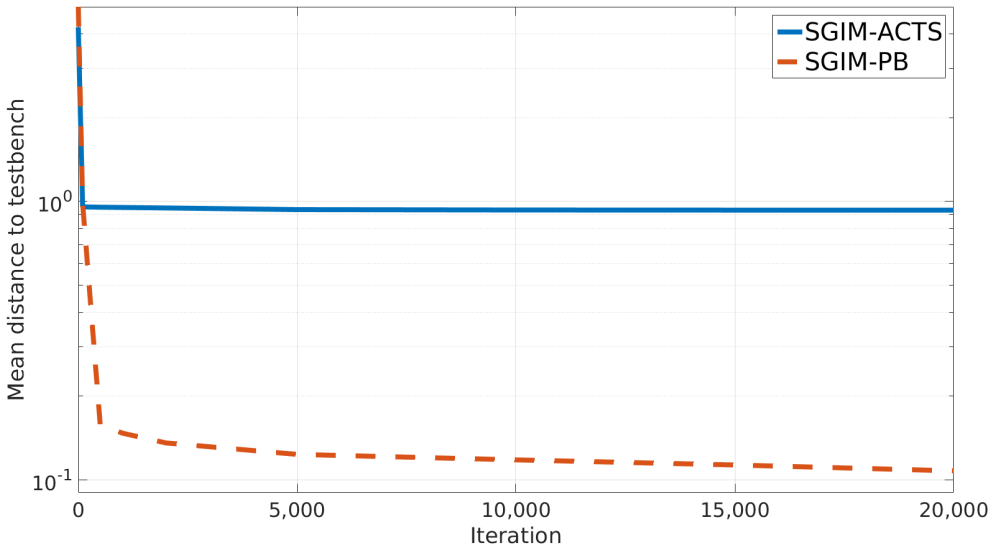

FIGURE 5.10: Global evaluation of the physical Yumi experiment

5.2.4 Evaluation method

For this experiment, the same evaluation method as in 5.1.4 is used, except that a new evaluation testbench of 10,000 points has been added for the Ω_5 outcome space. So now the testbench for the whole evaluation totalizes 29,200 points.

For this experiment, only two algorithms have been compared, due to a lack of time for the experiment. The SGIM-ACTS algorithm as well as the SGIM-PB one. Both algorithms were run once for the experiment, so the results provided in the next subsection are advanced results. Each run takes a total of 30 days to complete the 20,000 learning iterations. The code used for this experiment is available at `https://bitbucket.org/smartan117/sgim-yumi-real`.

5.2.5 Results

Evaluation performance

Fig. 5.10 shows the comparative global evaluation results of SGIM-PB and SGIM-ACTS. **SGIM-PB is capable of learning much further than SGIM-ACTS**, and keeps on progressing throughout the learning process. In order to understand this huge gap between both algorithms, we need to look at the evaluation measure evolution for each individual outcome space.

Fig. 5.11 shows that excepting the first two outcome spaces Ω_0 and Ω_1, SGIM-PB outclasses SGIM-ACTS in terms of final learning capabilities. More generally, both algorithms are capable of converging very quickly, owing to the bootstrapping effect of the interactive strategies. SGIM-PB teachers for the complex outcomes $(\Omega_1, \Omega_2, \Omega_3, \Omega_4)$ only gives procedures, which necessitates to first learn the components simplier outcomes before being able to exploit them for reaching more complex ones, whereas SGIM-ACTS can directly request actions from its teachers, leading to more immediate results. However, while converging slightly slower than SGIM-PB, it still takes less than 1,000 iterations for it to catch up with SGIM-ACTS in terms of evaluation convergence. Furthermore, SGIM-PB is capable of progressing throughout the experiment while SGIM-ACTS quickly stagnates after only 5,000

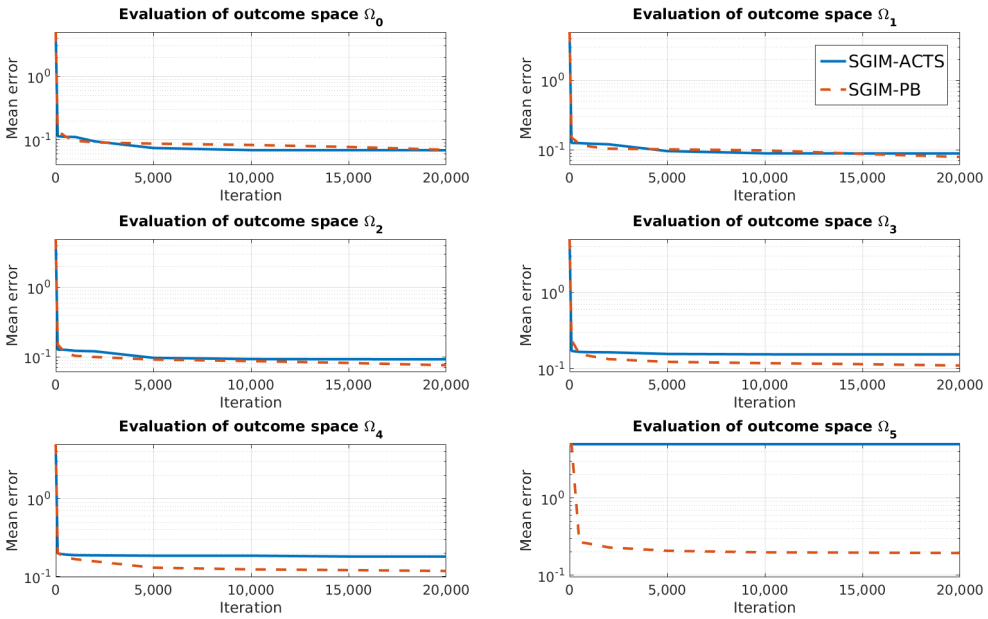

FIGURE 5.11: Evaluation for each outcome space of the physical Yumi experiment

iterations. For the Ω_5 outcome space, both algorithms could **only rely on their autonomous exploration strategies**, as no teacher was provided. **SGIM-PB, owing to the procedure framework to reuse the knowledge acquired for the other outcome spaces, was able to explore in this outcome space while SGIM-ACTS was not capable to at least reach it once**.

Analysis of the sampling strategy chosen for each goal

If we focus on the **SGIM-PB** results on this experiment, we can see it was capable of correctly self-organize its learning process, by **correctly peak the teacher most adapted to the goal at hand** (see Fig. 5.12). There is however one exception, as it appears that the learner chose the ProceduralTeacher4, giving procedures for the Ω_4 outcome space, with target outcome in Ω_3. Though suboptimal, it can be explained by the fact producing sound (Ω_4) induces moving both objects (Ω_3), so this teacher can indirectly provide demonstrations for this outcome space.

Length of actions chosen and task hierarchy discovered

I wanted to see whether SGIM-PB was able to learn both the task hierarchy of this setup, and the complexity of the different outcome spaces.

That is why, after the learning process, I subjected my learner to the evaluation testbench, recording which procedural space and which size of actions the local exploration method would use to build the procedure and action reaching each outcome test point. The results of this analyzes are presented as histograms, Fig. 5.13 for the action sequence complexity on 4 incrementally more complex outcome spaces ($\Omega_0, \Omega_1, \Omega_4, \Omega_5$), and Fig. 5.14 for the procedural spaces chosen for each outcome space.

We can see on Fig. 5.13, that the **SGIM-PB** learner is capable to **adapt the complexity of its action sequences to the targeted outcome**. It chooses short actions of

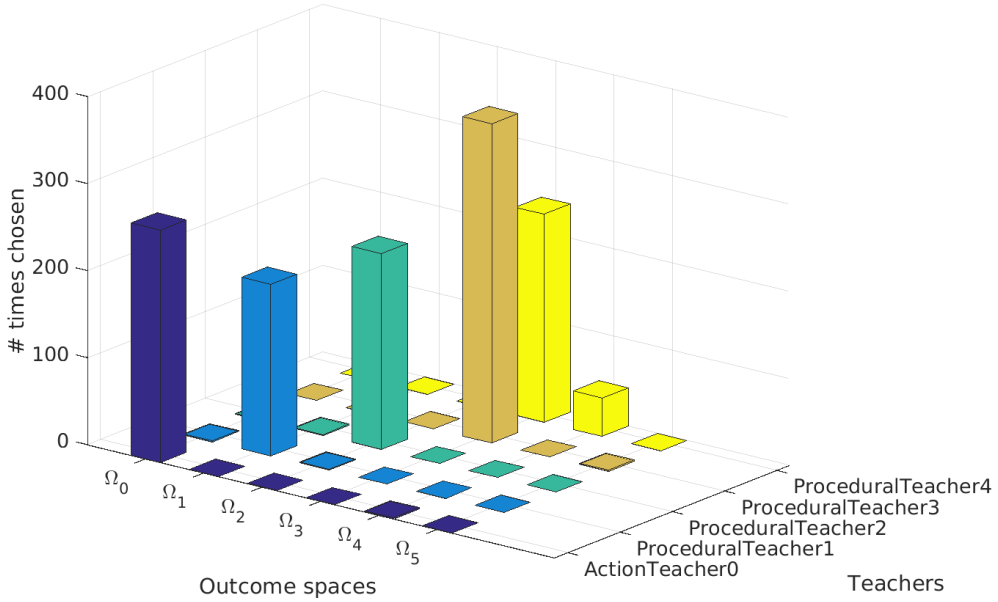

FIGURE 5.12: Number of choices of each interactive strategy and goal outcome space during the learning process

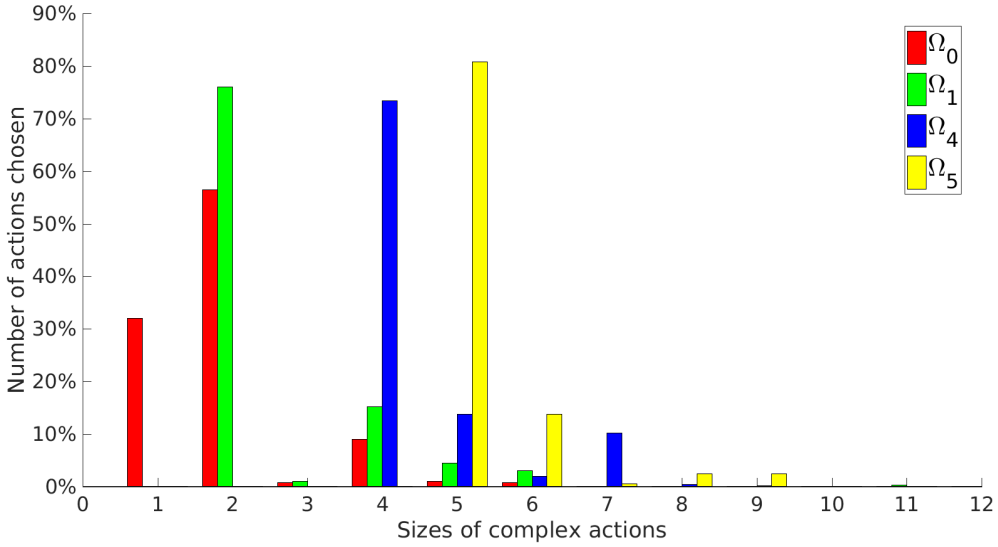

FIGURE 5.13: Percentage of actions chosen per action size by the SGIM-PB learner for each outcome space

size 1 or 2 for the simplest outcome space Ω_0, size 2 actions mostly for Ω_1, while using longer actions of size 4 for producing sound (Ω_4), and it uses size 5 actions for the most complex outcome space (Ω_5).

Fig. 5.14 shows **SGIM-PB was able to learn the task hierarchy**. For each outcome space is choosing mostly the same procedural space than the one used by the procedural teacher expert of this outcome space. It is even capable to **learn the task hierarchy in the absence of provided teacher** for the Ω_5 outcome space. The learner found that to produce a maintained sound (Ω_5), it simply has to produce a sound (Ω_4) and then put its arm's tip (Ω_0) at a new position on the interactive table.

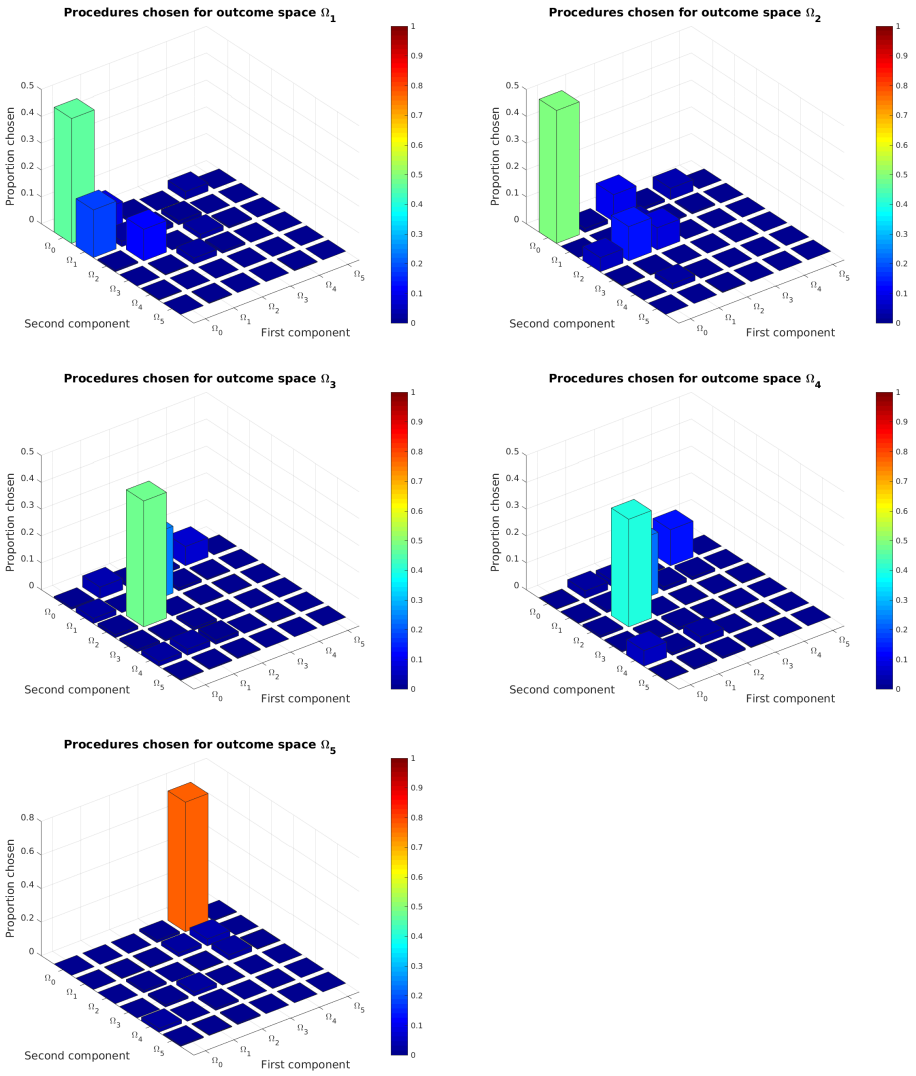

FIGURE 5.14: Task hierarchy discovered by the SGIM-PB learner: this represents for each complex outcome space the percentage of time each procedural space would be chosen for the physical yumi experiment

5.2.6 Conclusion

The results of this experiment confirm the simulation results observed in section 5.1. **SGIM-PB is able to outperform SGIM-ACTS on a real physical setup**, SGIM-ACTS which is the best action-only developmental learner of the simulated experiment. It also proves the ability of the SGIM-SAHT architecture to self-organizes the learning process. The SGIM-PB algorithm can also discover and exploit the task hierarchy to learn further, thanks to the procedure framework. It is also capable to correctly assess the complexity of each outcome space at the end of its learning process.

Another conclusion of this experiment is the **apparent superiority of demonstrated procedures for complex outcomes over demonstrated actions**. At the cost of a slightly slower convergence, the learner seems to learn better in the end. And

this, while the time needed to record the demonstrated procedures, which only requires to select the two procedure component for each demonstration, is far shorter than the time needed to record a demonstrated action. Indeed, because of the correspondence problem, showing the robot a action is done using kinaesthetic.

In the next section, I investigate another potential advantage of the procedure framework. I study if the procedural knowledge can be easily transferred between two learning agents on similar setups.

5.3 Transfer learning

Transfer learning (Pan and Yang, 2010) (Taylor and Stone, 2009) (Croonenborghs, Driessens, and Bruynooghe, 2008) describes an ensemble of techniques that address the problem of adapting a learning agent to changes in the environment he has learned, without restarting from scratch the learning process. This is especially true in cases where building a training dataset is expensive or impossible. These techniques instead, focuses on giving the learner the capability to **reuse the old training data and adapt them to the new circumstances**.

I wanted to see if my SGIM-PB learner could, owing to transfer learning techniques, have an advantage when its environment changes during the learning process (i.e. robot displacement or even change of robot type altogether) so that it doesn't have to restart from scratch. In particular, I argue the **procedures framework is a good way to transfer data** after such a change.

5.3.1 Experimental setup

I reused the simulated Yumi setup, described in 5.1. I let the SGIM-PB learner exercises on the setup for a complete run duration of 25,000 iterations. Then, I stopped the learning process, and forced the robot to use its left arm instead of its right arm (it used the right arm for the whole learning process up to this point). Then the robot has to adapt to the change as rapidly as possible in order to learn in this new configuration. The primitive action space Π and outcome space Ω are the same, but the individual actions have all to be relearned. A new revised action teacher is provided to the learner for the Ω_0 outcome space, and the same procedural teachers are available to it.

The goal for the learner is through the use of transfer learning, to learn faster thanks to reusing parts of its old knowledge base which are not outdated. And the parts that are unchanged are the procedures which it had learned as the outcome spaces and their relationships which were left untouched.

5.3.2 Definition of the problem

To understand transfer learning (Pan and Yang, 2010), we should define two important notions first: "domain" and "task".

A "domain" $D = \{\chi, P(X)\}$ consists of two components: a feature space χ and a marginal distribution $P(X)$ where $X = \{x_1, ..., x_n\} \in \chi$. An example in a document classification task is taking each term as binary feature, χ is the space of all term vectors, and x_i corresponds to the i^{th} term vector in some documents, and X is a particular learning sample. Two domains are different if they have a different feature space of a different marginal probability distribution.

Given a specific domain D, a "task" $t = \{Y, f(.)\}$ consists of two components: a label space Y and an objective predictive function $f(.)$ which is learned from the

training data, consisting of pairs $\{x_i, y_i\}$ and is used to predict the label $f(x)$ of new instances x. In the same example of a document binary classification task, Y can be *True* or *False*.

Given a target domain D_T and task T_T, and a source domain D_S and task T_S, transfer learning aims at improving the learning of the target predictive function $f_T(.)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$ or $T_S \neq T_T$.

Transfer learning techniques are categorized in 3 types according to Pan and Yang (2010):

- Inductive transfer learning: $T_S \neq T_T$. Some labelled data in D_T are required to induce $f_T(.)$;
- Transductive transfer learning: $D_S \neq D_T$ and $T_S = T_T$. No labelled data in D_T are available while a lot are in D_S;
- Unsupervised transfer learning: $T_S \neq T_T$ but are related and Y_S and Y_T are not observable. It focuses on solving unsupervised tasks.

5.3.3 Transfer Learning in SGIM-PB

In this experiment, I want the SGIM-PB learner to **reuse the procedures previously learned on the new modified setup**. So the outcomes correspond to the features in the transfer learning notations, while the actions and procedures are viewed as the labels. This can be counter-intuitive as the actions are the variables the learner controls to produce the outcomes. However, it must be noted that the SGIM-PB algorithm is trying to learn the inverse model L^{-1}, as it tries to predict which procedure is more adapted to which outcome. Then $P(X)$ represents the distribution of outcomes, while $f(.)$ corresponds to the inverse model itself L^{-1}. So in my particular problem, We can make the following transfer learning assumptions:

- The outcome spaces are the same;
- The action space Π is not the same so $Y_S \neq Y_T$;
- $P(X)$ is not the same between the target and source domain;
- The predictive function $f(.)$ is different but related by the procedures;
- Labelled data (both procedures and actions) are available for D_S, while only partial labelled data (procedures only) are available for D_T.

From these assumptions, we can deduce that D_S and D_T, and T_S and T_T are respectively different but related for SGIM-PB. We also have access to labelled data in D_T. Therefore my problem lies in the **inductive transfer learning** category.

For this first attempt at using transfer learning for the SGIM-PB learner, I am using an offline approach. When the environment of the learner changes, when it has to switch its controlled arm, a transfer function is applied so as to keep all old reached outcome reached via a procedure, along with the procedure in question. So these old transferred data, will only be used when the robot is using the local exploration of the procedural space substrategy.

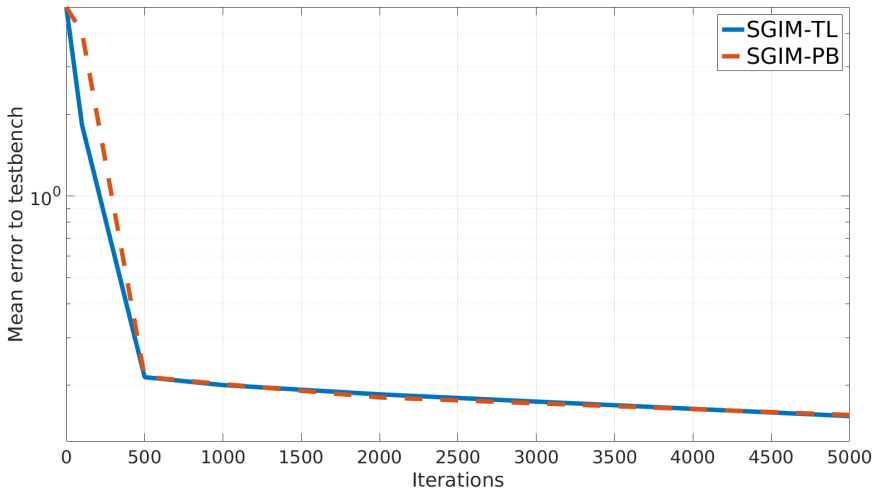

FIGURE 5.15: Global evaluation of both learners

5.3.4 Teachers

As the dataset of actions for the Ω_0 outcome space, used by ActionTeacher0 in section 5.1 was recorded for a Yumi robot right arm, I need to replace it by a new dataset adapted to its left arm. So I built a new teacher dataset of 11 demonstrations of primitive actions reaching various positions on the interactive table, including both objects initial positions.

5.3.5 Evaluation method

To assess my method ability, I used the same evaluation method as in 5.1.4.

I trained a SGIM-PB learner on its right arm for 25,000 iterations, corresponding to a rather mature learning process. Then I compared two SGIM-PB runs on the left arm, one starting from scratch, and the other having transferred the procedures from the mature right-arm SGIM-PB learner. In the following subsection, I refer at **SGIM-PB** for the learner **starting from scratch** on the left arm, and **SGIM-TL** for the one **using transfer learning**. Each was run 3 times and the results given are the average of those runs.

5.3.6 Results

Fig. 5.15 shows the results of the global evaluation of both learners. We can see that the learner with the procedures transferred, SGIM-TL, has an initial boost compared to the regular SGIM-PB algorithm. Then after around 500 iterations, both algorithms have almost the same evaluation measures up to the end. This seems to show that **my transfer learning method is sufficient to bootstrap the early learning of a SGIM-PB learner, but this advantage does not endure for the full learning process**.

If we look in more details, we can see the evaluation for each particular outcome space of both algorithms on Fig. 5.16.

The results are really astonishing, as I had expected the transfer learning bootstrap to be more visible on the most complex tasks, however it appears to be the opposite. Indeed, SGIM-TL has a significant head start for the outcome space Ω_0,

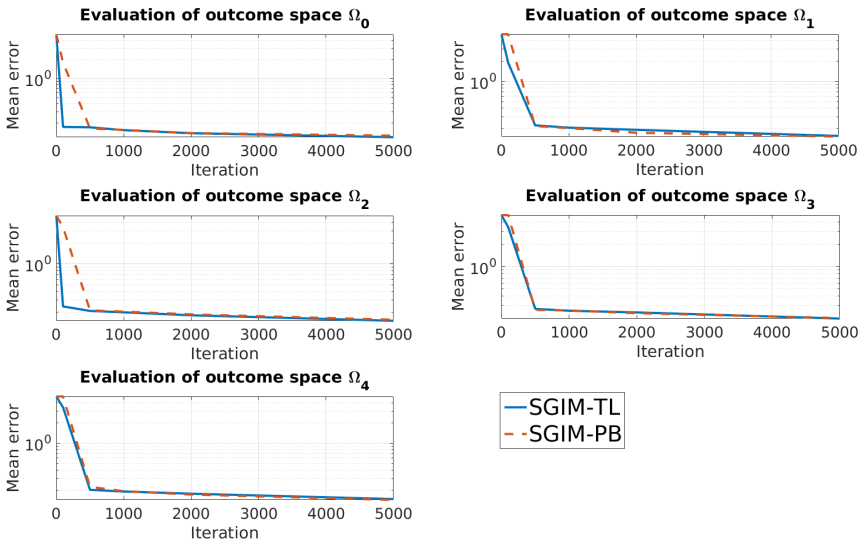

FIGURE 5.16: Evaluation for each task of both learners

Ω_2 and Ω_1 to a limited extent. SGIM-TL is also slightly faster in the beginning to learn on all the other outcome spaces, but this difference is small.

If we look at the procedures the learners are choosing, in order to see if the transferred procedures have any effect on the learner SGIM-TL, we obtain underwhelming results. Indeed if I compute at the end of the learning process the procedures used for each point of the evaluation testbench, which gives us the task hierarchy discovered at the end of the learning process, we can see that **SGIM-TL discovers the same hierarchy than its model the transfer dataset**. This is shown on Fig. 5.17 But this can easily be explained as SGIM-TL is still in the beginning of its learning process after the experiment (5,000 iteration versus the 25,000 iterations of learning data in the transfer dataset), so the majority of its procedures comes from this transferred dataset. However if we look at what **both learners have actually used during their learning process on Fig. 5.18, we don't see much of a difference between both learners**.

5.3.7 Conclusion

Although I still believe that procedures can be transferred from a mature SGIM-PB learner to a beginner SGIM-PB learner with a change in its motor control (in this case using the left arm instead of the right one), this experiment proved that my method is too simple, and should be modified significantly if I want to reach better results.

However, the early bootstrap which we observe on the learner with the transferred dataset compared to the regular one makes us hopeful that this method could be improved to yield better results. I still need to understand why **the transfer of such a huge dataset of procedures seems not to alter the online choices of procedures made by the learner during its learning process**.

Implementing a real transfer function, such as to analyze the data before transferring them, could be a lead to follow in order to develop a reliable transfer learning mechanism for a SGIM-PB learner. Such a method could use clustering methods to filter the procedures most helpful to the learner. Also, prolonging the experiment and examining the results at the end of a longer learning process of 25,000 iterations would be interesting to see if the transfer has some long term learning benefits.

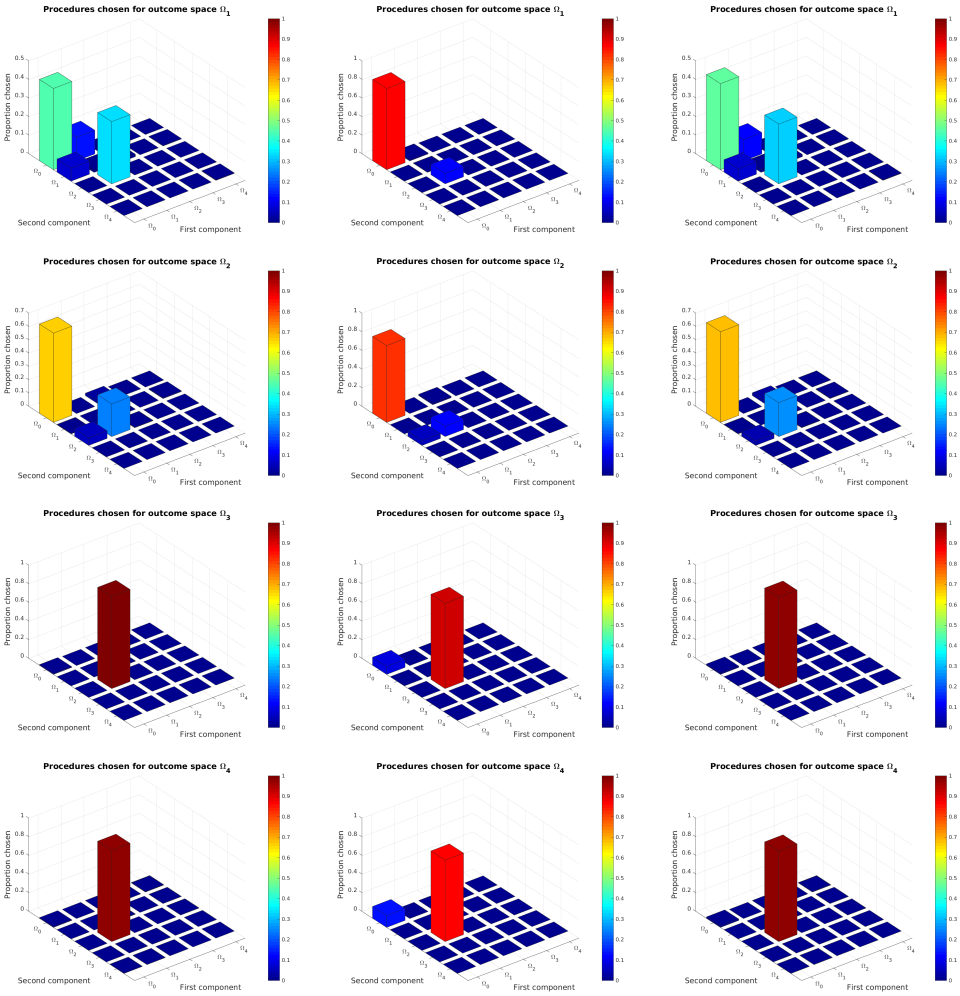

FIGURE 5.17: Task hierarchy discovered by the learners compared to the transferred dataset (Transfer dataset on the left column, SGIM-PB in center one, SGIM-TL on right one): this represents for each complex outcome space the percentage of time each procedural space would be chosen for the simulated yumi experiment with transfer learning

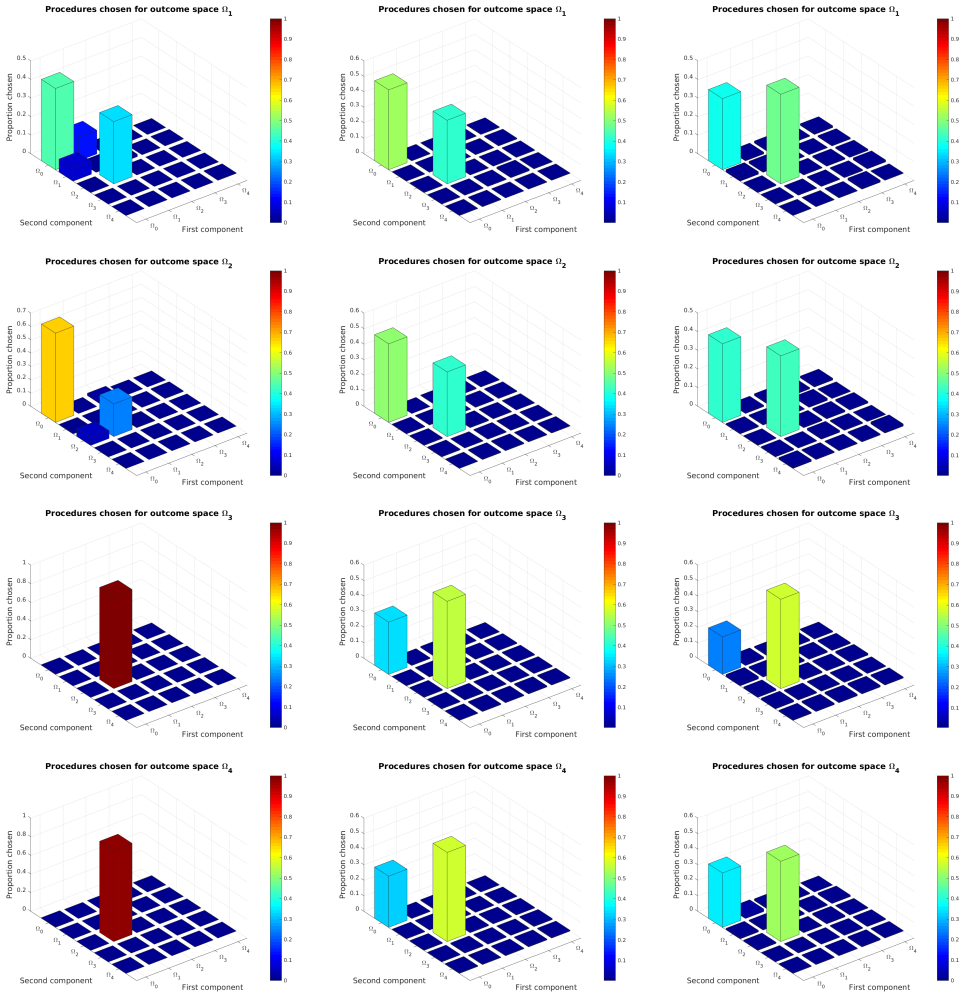

FIGURE 5.18: Task hierarchy used by the learners during their learning process compared to the hierarchy discovered in the transferred dataset (Transfer dataset on the left column, SGIM-PB in center one, SGIM-TL on right one): this represents for each complex outcome space the percentage of time each procedural space is chosen for the simulated yumi experiment with transfer learning

Chapter 6

Conclusion

6.1 Conclusion of the manuscript

In this thesis, I tackled the **learning of a set of multiple hierarchically organized tasks using action sequences**. In Chapter 1, I described the *life-long learning* problem, in which scope I fall, and its challenges a learning agent has to overcome. Those challenges are the stochasticity of the agent's environment, the high-dimensionality of its sensorimotor space, the unlearnability of some regions of this sensorimotor space and its unboundedness. To tackle these challenges, I use the principles of cognitive developmental robotics: a developmental approach, the action-perception loop, enactivism and trial-and-error. More precisely, I described the different methods from which I inspire: using temporal abstraction and goal-oriented representation to learn unbounded motor action sequences, exploiting the dual representation of tasks and action sequences to discover and exploit the task hierarchy in an environment, using intrinsic motivation as a guidance mechanism for the learning process, using interactive strategies to bootstrap this learning process, choosing between multiple strategies the most appropriate one depending on the task at hand. In Chapter 2, I developed both the formalization of the learning problem and that of the **SGIM-SAHT learning architecture**. This architecture combines intrinsic motivation as a guidance mechanism with multi-task learning, and proposes to use both autonomous strategies and interactive ones, bootstrapped by a framework built to discover and exploit the task hierarchy of the environment, by combining skills in a task-oriented way: the procedure framework. In the previous chapters, different implementations of this architecture were proposed. Their features are shown in Table. 6.1.

I tested my architecture first using simple primitive actions only on a real physical setup. The first version of the SGIM-SAHT architecture, developped for this case of simple primitive actions, proved able to self-organize its learning process. It was also capable to learn more tasks than other learning algorithms on this setup, owing to the combined strength of the interactive strategy to bootstrap the early learning process and autonomous exploration strategy to extend its skills. This ability to learn a set of hierarchically organized tasks proved the SGIM-SAHT architecture as potentially adapted to learn sequences of motor actions.

	Intrinsic Motivation	Action Size	Procedure Framework	Social Guidance
SGIM-ACTSCL	Yes	Primitives only	No	Yes
IM-PB	Yes	Any	Size-2	No
SGIM-PB	Yes	Any	Size-2	Yes
SGIM-SAHT	Yes	Any	Any	Yes

TABLE 6.1: Features implemented by all the implementations of the SGIM-SAHT architecture presented in this thesis

Then I developed the **procedure framework, which by allowing the combination of previously learned skills through the composition of outcomes, enabled the discovery and exploitation of the task hierarchy**. In this framework, we define procedures as sequences of previously known outcomes, which are replaced by the succession of actions reaching those. This framework was made to tackle the learning of sequences of motor actions in a task-oriented way. This was tested on an experimental setup with a set of hierarchically organized tasks. It was first tested on a learner performing autonomous exploration strategies, called IM-PB. Then it was tested on a more complete implementation of the SGIM-SAHT architecture which uses interactive strategies, called SGIM-PB. **The procedural framework, eased the learning of the hierarchical set of tasks**. The learners having this framework, were capable to learn more tasks than their respective counterparts. Such learners were also capable of organizing their learning process, adapting the complexity of actions used to that of the tasks at hand, and discovering the task hierarchy of the setup. This framework seemed particularly useful on the most complex and hierarchical tasks, whereas the use of actions seemed logically preferred on the most simple tasks. For SGIM-PB, a new way to provide demonstrations as procedures was designed in the *mimicry of a procedure teacher* strategy. This strategy proved to enable the robot to focus more on the most useful procedural spaces according to the task at hand.

An experimental setup was designed, using a **real physical robot** to learn a set of hierarchical tasks using sequences of motor actions. The test of the SGIM-PB learner on this setup in simulation, **yielded the same results** than on the previous one. The tests on a real physical version of this setup, although not statistically significant for now, seemed to comfort my results. Also, in this physical test, I delved more deeply in the comparison of procedure demonstrations and action ones. The results confirmed my theory that both are complementary: actions more useful for the simplest tasks while procedures are better on the complex most hierarchical tasks. Lastly, I wanted to see how my procedural framework could enable the transfer of the hierarchical information (i.e. the learned procedures) from a trained SGIM-PB learner to an untrained different one. This transfer of the procedures proved to bootstrap the early learning process of a different SGIM-PB robot, though it did not seem to influence on the long run, and the tasks affected were not the most complex ones as I anticipated. Moreover, **the transfer of procedures did not seem to alter the procedures used by the SGIM-PB learner**. This proved that the method I used to transfer procedures must be refined before producing any significant predictable results.

6.2 Conclusions and limitations

6.2.1 Conclusions of the approach

I showed in this work that **a developmental approach, and more precisely a strategical and intrinsically motivated one, can effectively enable a robot to learn multiple hierarchically organized and inter-related tasks in a complex environment**. This is due to the learner ability to organize its learning process and tackle multiple tasks using the most appropriate strategies. The ability to exploit the task hierarchy after discovering it, also contributes in a learner ability to reuse its previous knowledge to tackle the most complex tasks.

In both the cases of a simple action setup like the Poppy experiment and an action sequence one like the Yumi experiment, the SGIM-SAHT architecture enables the learner to adapt its learning strategy to the task at hand.

The use of the procedure framework, gives the SGIM-PB learner an advantage in learning speed, as well as in learning more tasks in the end. I showed that although action teachers directly providing sequences of motor actions for complex outcomes bootstraps the early learning process, **the use of procedures, demonstrated or not, gives an edge to a strategic intrinsically motivated learner on the long run**.

And in the absence of teachers, an autonomous intrinsically motivated learner using the procedural framework showed it outmatched its action-only counterpart.

Both the IM-PB and the SGIM-PB algorithms, showed to be able to **adapt the length of their actions to the task at hand**.

The procedural teachers do not only help the SGIM-PB learner to start its learning process for complex outcome spaces. Once simpler outcome spaces needed have become mature, they also learn it to focus on the most useful procedural spaces. This showed **the ability of my algorithmic architecture to discover and exploit the task hierarchy**. Also the very existence of the procedure framework, gives human teachers another mean to provide demonstrations to a learning agent. I hypothesize that this way of providing demonstrations is easier for the teacher, especially for the most complex tasks and if they are non-robotic but only task experts and would have a hard time handling the robot to provide it actions.

The procedure framework also seems to enable the transfer of knowledge when the environment of the learner changes. That transferred knowledge is the ensemble of all procedures tested along with their reached outcomes. This transfer is however not significant and I believe that refining the transfer method will yield better results.

6.2.2 Limitations of the approach

However, some work still needs to be done on the subject. Indeed some preparatory work has been presented in this thesis, and they still need to be confirmed by a more statistical analysis. But more importantly, the current approach suffers from different drawbacks.

While the procedures do enable a learner to combine previously learned skills to build new more complex ones, not a great focus has been given to the efficiency of the action sequences built. Indeed while the algorithm does adapt the length of its actions to the task at hand, **it still combine way more primitive actions than necessary**. This is due to the fact that the main factors to build procedures is the accuracy of the attempt, rather than some efficiency, or energy spent in the process. A way of tackling this issue could be to integrate the energy or time spent during each attempt into the performance metric more aggressively to balance the ratio between accuracy and efficiency, instead of just adding a limitation factor as it is now the case. Another method would be to formalize a 2-variables progress, one being the accuracy progress already in use, the other being a progress in efficiency. This would enable the learner to come back to the study of an already known outcome space region, when a breakthrough shows it could be reached using shorter actions. And a third method would be to add an analysis process using for example clustering techniques, so as to form classes of actions supposedly able to cover whole outcomes regions, and prioritize the use of the most efficient ones.

Also, the initial state of the environment, before performing an action (whether from rest state or between two primitive actions execution) is not taken into account. So when combining actions together, the first one was actually recorded from the same starting point, while the second one will necessary start from a different position, and even a different environment configuration (i.e. context). This could generate situations in which a first action is executed (according to a procedure) and

would reach the given outcome (the first component of the procedure) and possibly some other outcome as well (in a different outcome space), then the second action could undo what the first one did, or been unable to do its part because of another outcome reached by the first action. This problem is amplified by the fact the learner does not know when an outcome is produced, if it comes from the action it just did, or if this is just a consequence already observed before that has not been modified by the last action executed.

Moreover, it can sometimes generate suboptimal behaviours where a procedure outcome is only due to one of its components. More precisely, it can reach the target outcome then move on and do something else which does not invalidate the first step. Then the robot will consider this procedure reached the correct outcome, and its procedural space, though suboptimal, would have more chances of being reused later for the same target outcome space. For example, in the Yumi simulated experiment, the robot could decide to move the first object, by applying a procedure moving both objects. This would lead to **more complex procedures and actions for relatively simple target outcomes**. This latter problem being due both to not using the contexts, and amplified by the quasi-absence of a measure of the robot efficiency from the algorithm decisions. This problem could become intractable, if we authorize the learner to combine an unconstrained number of outcomes in a procedures.

Lastly, while the low-level models and functions used in this work are voluntarily simple, using more complex models, such are neural networks instead of linear regression, or simply taking time to optimize certain hyperparameters (like the γ factor used to take action size into account in NN-search), could make these algorithms more powerful. This is especially true for the exploitation of all combined knowledge by the inverse model, for which a better method of generalizing its knowledge could unlock new possibilities for our architecture in real-life applications.

6.2.3 Perspectives

I see different ways to enhance this work. They are based on problems encountered and identified during this thesis.

First, the learner is only able to reuse actions and procedures starting from the initial rest pose, as they could lead to various different outcomes if starting from another context. This prevent the learner to reuse parts of actions and procedures, not starting from rest pose. Enabling the learner to do so, would multiply possibilities for the learner to reuse its knowledge. It could be accomplished by **introducing the contexts in the SGIM-SAHT architecture, as a new space to be exploited**. Then the robot could record all encountered contexts, and identified which are close so that it could more easily combine actions based on their final and initial context. However, this could **dramatically increase the complexity** of the learning process, and would also lead to the problem of balancing between contexts and target outcomes to determine the action or procedure to apply. This problem could tackled by allowing the learner to **extract itself the features** (parameters of actions, contexts or outcomes) that influences a specific task. This could, for example, enable the learner to ignore the context in situations where it does not influence the outcome.

Second, actions are combined using procedures in my work, and those procedures are built using demonstrations or exploration of the procedural space. This exploration can be quite slow, and could become intractable when enabling the formation of procedures of unbounded size. **Combining this approach with planning could help explore this infinite dimensional space of actions more effectively**. A planning process could build unbounded sequences of outcomes or actions, from the

learner knowledge base, therefore speeding up the exploration. In order to do that, we would need to rely more on the forward model learned by the robot. However, this planning process should also be optimized so as not to slow done the whole learning process. This could be done by **discretizing the environment** in classes of actions and outcomes.

Third, whenever the learner is trying to autonomously reach a specific outcome in the environment, whether in the beginning of its learning process when its knowledge base is really sparse or at the end when it is denser, it starts by searching in its whole database and build a neighbour set of actions or procedures, and outcomes. This process is quite slow and is growing slower when the dataset grows. **Working at a representational level and extract global and local rules from its database and apply them to similar situations instead of always looking in its whole database, could speed up the process while allowing the learner to reflect on its knowledge and optimize them**. Clustering and tree- or graph-based representation techniques could be used for such purpose. These would also add the benefit of providing a human expert observer with a better understanding of the learner's knowledge, and so allowing him to help it more accordingly.

Last, when a human expert is teaching the robot in my approach, it is only at the learner's request. So, if the teacher observes the learner is performing very poorly, by spending a lot of time exploring uninteresting spaces (known as such by the teacher but not by the learner), or overoptimizing actions to reach specific outcomes in an already vastly known region because it did not discover other regions, it must wait for the learner to ask for help. This could take some time, **giving teachers the ability to intervene when they saw fit, could enable them to bias the learner's exploration towards what they deem more important faster**. This could be done by allowing them to use *scaffolding* or provide external rewards, leading to a more complete and unified implementation of a developmental robotic approach. Scaffolding is done by a teacher to place its student in a state that eases its learning, for example a parent holding its infant hands or hips to help the learning of walk. I could also introduce other social guidance notions such as *emulation*, which could show to the learner what tasks are more useful and feasible in the environment.

6.3 Contributions

In this thesis, I have focused on the learning of a set of complex hierarchically organized tasks. I showed that a strategical intrinsically motivated learner is well equipped to learn in such an environment, owing to its ability to select the right task to learn, the right method to learn it, at the right time. The ability to combine both autonomous exploration and interactive strategies is particularly useful, as it enables the learner to combine those methods strengths while alleviating their weaknesses. I also showed that **taking a task-oriented approach to enable a learner to compose known skills together is well indicated into enhancing the learning in such environment**. I also showed why this framework inside a strategical intrinsically motivated learner is efficient, due to the ability for such learner to adapt its strategy and the complexity of its actions to the task at hand.

6.4 Takeaway message

One of the key aspects in learning complex tasks with sequences of motor actions, is the ability to combine simplier skills together in a task-oriented way so as to

build new more complex ones. Without this way of combining and reusing simplier skills, a learning agent would be overwhelmed by the vastness of its environment, and that of its own effectors. Also, combining interactive learning and autonomous exploration into a single strategical intrinsically motivated learner clearly showed its potency in such environment, by both bootstrapping early development via human advice, then decreasing the human load by relying on self-exploration on the long run. Teaching a learner how to combine its skills rather than teaching it new complex skills from scratch extends the communication tools of the teacher with the robot, by offering a new promising way for it to help a robot in its learning process.

6.5 Impact

This work could be used outside the sphere of developmental robotics. Indeed, the algorithms developed in this thesis, could be applied in **any machine learning problem, where an agent has to learn in as few trials as possible how to perform its tasks, for multi-task learning in a static environment**. But also, a dual-representation of sequences of actions and outcomes as used in this thesis could also be **observed in human infants learning complex tasks**. Experiments could be conducted so as to see how infants learn those complex tasks and how they tend to reuse their previous knowledge to solve increasingly more difficult problems relating to motor skills development in sport practice. Also a sociological study could suggest if this approach of teaching how to combine simple skills rather than teaching the new sportive motor skills from scratch is actually easier and more relevant for a human teacher.

6.6 Papers

- N. Duminy, S. M. Nguyen, and D. Duhaut, "Strategic and interactive learning a a hierarchical set of tasks by the Poppy humanoid robot", in *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, Sept. 2016, pp. 204-209.
- N. Duminy, S. M. Nguyen, and D. Duhaut, "Learning a set of interrelated tasks by using sequences of motor policies for a strategic intrinsically motivated learner", in *2018 Second IEEE International Conference on Robotic Computing (IRC)*, 2018, pp. 288-291.
- N. Duminy, S. M. Nguyen, and D. Duhaut, "Effects of social guidance on a robot learning sequences of policies in hierarchical learning", in *IEEE International Conference on Systems, Man and Cybernetics (SMC2018)*, 2018, pp. 3755-3760.
- N. Duminy, A. Manoury, S. M. Nguyen, C. Buche, and D. Duhaut, "Learning Sequences of Policies by using an Intrinsically Motivated Learner and a Task Hierarchy", in the Workshop on Continual Unsupervised Sensorimotor Learning in *2018 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, 2018, accepted.
- N. Duminy, S. M. Nguyen, and D. Duhaut, "Learning a Set of Interrelated Tasks by Using a Succession of Motor Policies for a Socially Guided Intrinsically Motivated Learner". In: *Frontiers in Neurorobotics* 12, p. 87.

Bibliography

Argall, Brenna D., B. Browning, and Manuela Veloso (2008). "Learning robot motion control with demonstration and advice-operators". In: *In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 399–404.

Argall, Brenna D. et al. (2009). "A survey of robot learning from demonstration". In: *Robotics and Autonomous Systems* 57.5, pp. 469 –483.

Arie, Hiroaki et al. (2012). "Imitating others by composition of primitive actions: A neuro-dynamic model". In: *Robotics and Autonomous Systems* 60.5, pp. 729–741.

Asada, Minoru et al. (2009). "Cognitive developmental robotics: A survey". In: *IEEE transactions on autonomous mental development* 1.1, pp. 12–34.

Baram, Y., R. El-Yaniv, and K. Luz (2004). "Online choice of active learning algorithms". In: *The Journal of Machine Learning Research*, 5, pp. 255–291.

Baranes, Adrien and Pierre-Yves Oudeyer (2010). "Intrinsically motivated goal exploration for active motor learning in robots: A case study". In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, pp. 1766–1773.

Baranes, Adrien and Pierre-Yves Oudeyer (2013). "Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots". In: *Robotics and Autonomous Systems* 61.1, pp. 49–73.

Barto, Andrew G, George Konidaris, and Christopher Vigorito (2013). "Behavioral hierarchy: exploration and representation". In: *Computational and Robotic Models of the Hierarchical Organization of Behavior*. Springer, pp. 13–46.

Barto, Andrew G and Sridhar Mahadevan (2003). "Recent advances in hierarchical reinforcement learning". In: *Discrete event dynamic systems* 13.1-2, pp. 41–77.

Bengio, Yoshua et al. (2009). "Curriculum Learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. New York, NY, USA: ACM, pp. 41–48.

Billard, Aude et al. (2007). "Handbook of Robotics". In: 59. Chap. Robot Programming by Demonstration.

Brooks, Rodney A (1991). "Intelligence without representation". In: *Artificial intelligence* 47.1-3, pp. 139–159.

Brown, Solly and Claude Sammut (2012). "A relational approach to tool-use learning in robots". In: *International Conference on Inductive Logic Programming*. Springer, pp. 1–15.

Cakmak, Maya, C. Chao, and Andrea L. Thomaz (2010). "Designing interactions for robot active learners". In: *Autonomous Mental Development, IEEE Transactions on* 2.2, pp. 108–118.

Chernova, Sonia and Manuela Veloso (2009). "Interactive Policy Learning through Confidence-Based Autonomy". In: *Journal of Artificial Intelligence Research* 34.1, p. 1.

Colas, Cédric, Olivier Sigaud, and Pierre-Yves Oudeyer (2018). "GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms". In: *arXiv preprint arXiv:1802.05054*.

Croonenborghs, Tom, Kurt Driessens, and Maurice Bruynooghe (2008). "Learning Relational Options for Inductive Transfer in Relational Reinforcement Learning". In: *Inductive Logic Programming*. Ed. by Hendrik Blockeel et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 88–97.

Deci, E.L. and Richard M. Ryan (1985). *Intrinsic Motivation and self-determination in human behavior*. New York: Plenum Press.

Duminy, N., S. M. Nguyen, and D. Duhaut (Sept. 2016). "Strategic and interactive learning of a hierarchical set of tasks by the Poppy humanoid robot". In: *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 204–209.

Duminy, Nicolas, Sao Mai Nguyen, and Dominique Duhaut (2018a). "Effects of social guidance on a robot learning sequences of policies in hierarchical learning". In: *IEEE International Conference on Systems, Man and Cybernetics (SMC2018)*, pp. 3755–3760.

Duminy, Nicolas, Sao Mai Nguyen, and Dominique Duhaut (2018b). "Learning a set of interrelated tasks by using sequences of motor policies for a strategic intrinsically motivated learner". In: *IEEE International Robotics Conference*, pp. 288–291.

Duminy, Nicolas, Sao Mai Nguyen, and Dominique Duhaut (2019). "Learning a Set of Interrelated Tasks by Using a Succession of Motor Policies for a Socially Guided Intrinsically Motivated Learner". In: *Frontiers in Neurorobotics* 12, p. 87.

Elman, J. (1993). "Learning and development in neural networks: The importance of starting small". In: *Cognition* 48, pp. 71–99.

Forestier, Sébastien, Yoan Mollard, and Pierre-Yves Oudeyer (2017). "Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning". In: *CoRR* abs/1708.02190.

Forestier, Sébastien and Pierre-Yves Oudeyer (2016). "Curiosity-driven development of tool use precursors: a computational model". In: *38th Annual Conference of the Cognitive Science Society (CogSci 2016)*, pp. 1859–1864.

Giszter, Simon F (2015). "Motor primitives—new data and future questions". In: *Current opinion in neurobiology* 33, pp. 156–165.

Gottlieb, Jacqueline et al. (2013). "Information-seeking, curiosity, and attention: computational and neural mechanisms". In: *Trends in Cognitive Sciences* 17.11, pp. 585–593.

Grollman, Daniel H and Odest Chadwicke Jenkins (2010). "Incremental learning of subtasks from unsegmented demonstration". In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, pp. 261–266.

Hari, Riitta (2006). "Action–perception connection and the cortical mu rhythm". In: *Progress in brain research* 159, pp. 253–260.

Held, Richard and Alan Hein (1963). "Movement-produced stimulation in the development of visually guided behaviour". In: *Journal of comparative and physiological psychology* 56.5, pp. 872–876.

Hikosaka, Okihide et al. (1999). "Parallel neural networks for learning sequential procedures". In: *Trends in neurosciences* 22.10, pp. 464–471.

Ijspeert, Auke Jan, Jun Nakanishi, and Stefan Schaal (2002). *Learning attractor landscapes for learning motor primitives*. Tech. rep.

Konidaris, G.D. and Andrew G. Barto (2009). "Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining." In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 1015–1023.

Kubicki, Sébastien, Sophie Lepreux, and Christophe Kolski (2012). "RFID-driven situation awareness on TangiSense, a table interacting with tangible objects". In: *Personal and Ubiquitous Computing* 16.8, pp. 1079–1094.

Kubicki, Sébastien et al. (2016). "Using a tangible interactive tabletop to learn at school: empirical studies in the wild". In: *Actes de la 28ième conférence francophone sur l'Interaction Homme-Machine*. ACM, pp. 155–166.

Kulkarni, Tejas D et al. (2016). "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in neural information processing systems*, pp. 3675–3683.

Lapeyre, Matthieu, Pierre Rouanet, and Pierre-Yves Oudeyer (Oct. 2013). "Poppy Humanoid Platform: Experimental Evaluation of the Role of a Bio-inspired Thigh Shape". In: *Humanoids*. Atlanta, United States.

Lopes, Manuel and Pierre-Yves Oudeyer (2012). "The strategic student approach for life-long exploration and learning". In: *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*. IEEE, pp. 1–8.

Lungarella, Max et al. (2003). "Developmental robotics: a survey". In: *Connection Science* 15.4, pp. 151–190.

Machado, Marlos C, Marc G Bellemare, and Michael Bowling (2017). "A laplacian framework for option discovery in reinforcement learning". In: *arXiv preprint arXiv:1703.00956*.

Melo, Francisco S, Carla Guerra, and Manuel Lopes (2018). "Interactive Optimal Teaching with Unknown Learners." In: *IJCAI*, pp. 2567–2573.

Merrick, Kathryn E (2012). "Intrinsic motivation and introspection in reinforcement learning". In: *IEEE Transactions on Autonomous Mental Development* 4.4, pp. 315–329.

Muelling, Katharina, Jens Kober, and Jan Peters (2010). "Learning table tennis with a mixture of motor primitives". In: *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*. IEEE, pp. 411–416.

Nguyen, Sao Mai, Adrien Baranes, and Pierre-Yves Oudeyer (2011). "Bootstrapping intrinsically motivated learning with human demonstrations". In: *IEEE International Conference on Development and Learning*. Vol. 2. IEEE, pp. 1–8.

Nguyen, Sao Mai and Pierre-Yves Oudeyer (2012). "Active choice of teachers, learning strategies and goals for a socially guided intrinsic motivation learner". In: *Paladyn Journal of Behavioural Robotics* 3.3, pp. 136–146.

Nguyen, Sao Mai and Pierre-Yves Oudeyer (2014). "Socially Guided Intrinsic Motivation for Robot Learning of Motor Skills". In: *Autonomous Robots* 36.3, pp. 273–294.

Oudeyer, Pierre-Yves, Frederic Kaplan, and V. Hafner (2007). "Intrinsic Motivation Systems for Autonomous Mental Development". In: *IEEE Transactions on Evolutionary Computation* 11.2, pp. 265–286.

Pan, S. J. and Q. Yang (Oct. 2010). "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10, pp. 1345–1359.

Pastor, Peter et al. (2009). "Learning and generalization of motor skills by learning from demonstration". In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, pp. 763–768.

Peters, Jan and Stefan Schaal (2008). "Natural Actor Critic". In: *Neurocomputing* 7-9, pp. 1180–1190.

Piaget, J. (1952). *The origins of intelligence in children (M. Cook, Trans.)* New York: WW Norton & Co.

Reinhart, René Felix (2017). "Autonomous exploration of motor skills by skill babbling". In: *Autonomous Robots* 41.7, pp. 1521–1537.

Rolf, M., J. Steil, and M. Gienger (Sept. 2010). "Goal Babbling permits Direct Learning of Inverse Kinematics". In: *IEEE Trans. Autonomous Mental Development* 2.3, pp. 216–229.

Santucci, V. G., G. Baldassarre, and M. Mirolli (2016). "GRAIL: A Goal-Discovering Robotic Architecture for Intrinsically-Motivated Learning". In: *IEEE Transactions on Cognitive and Developmental Systems* 8.3, pp. 214–231.

Schaal, S., A. Ijspeert, and A. Billard (2003). "Computational approaches to motor learning by imitation". In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 358.1431, p. 537.

Schaal, Stefan, Christopher G Atkeson, and Sethu Vijayakumar (2002). "Scalable techniques from nonparametric statistics for real time robot learning". In: *Applied Intelligence* 17.1, pp. 49–60.

Schillaci, Guido, Verena Vanessa Hafner, and Bruno Lara (2012). "Coupled inverse-forward models for action execution leading to tool-use in a humanoid robot". In: *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*. ACM, pp. 231–232.

Schmidhuber, J. (2010). "Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990-2010)". In: *IEEE Transactions on Autonomous Mental Development* 2.3, pp. 230–247.

Silva, B.C. da, G. Konidaris, and Andrew G. Barto (2012). "Learning Parameterized Skills". In: *29th International Conference on Machine Learning (ICML 2012)*.

Stulp, Freek and Stefan Schaal (2011). "Hierarchical reinforcement learning with movement primitives". In: *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*. IEEE, pp. 231–238.

Sutton R. S., Rafols E. J. Koop A. (2006). "Temporal abstraction in temporal-difference networks". In: *Advances in Neural Information Processing Systems 18 (NIPS*05)*.

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning: an introduction*. MIT Press.

Sutton, Richard S, Doina Precup, and Satinder Singh (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2, pp. 181–211.

Taylor, Matthew E. and Peter Stone (Dec. 2009). "Transfer Learning for Reinforcement Learning Domains: A Survey". In: *J. Mach. Learn. Res.* 10, pp. 1633–1685.

Thomaz, Andrea L. and Cynthia Breazeal (2008). "Experiments in Socially Guided Exploration: Lessons learned in building robots that learn with and without human teachers". In: *Connection Science* 20 Special Issue on Social Learning in Embodied Agents.2,3, pp. 91–110.

Thorndike, E. L. (1898). "Animal intelligence: An experimental study of the associative processes in animals". In: *The Psychological Review: Monograph Supplements* 2.4, pp. i–109.

Thrun, Sebastian (2012). *Explanation-based neural network learning: A lifelong learning approach*. Vol. 357. Springer Science & Business Media.

Varela, F., E. Thompson, and E. Rosch (1991). *The embodied mind : cognitive science and human experience*. Cambridge, MA: MIT Press.

Titre : **Découverte et exploitation de la hiérarchie des tâches pour apprendre des séquences de politiques motrices par un robot stratégique et interactif**

Mots clés : *Motivation intrinsèque, Babillage de buts, Apprentissage de tâches multiples, Apprentissage interactif, Apprentissage hiérarchique, Apprentissage stratégique*

Résumé : Il y a actuellement des efforts pour faire opérer des robots dans des environnements complexes, non bornés, évoluant en permanence, au milieu ou même en coopération avec des humains. Leurs tâches peuvent être de types variés, hiérarchiques, et peuvent subir des changements radicaux ou même être créées après le déploiement du robot. Ainsi, ces robots doivent être capable d'apprendre en continu de nouvelles compétences, dans un espace non-borné, stochastique et à haute dimensionnalité. Ce type d'environnement ne peut pas être exploré en totalité, le robot va devoir organiser son exploration et décider ce qui est le plus important à apprendre ainsi que la méthode l'apprentissage. Ceci devient encore plus difficile lorsque le robot est face à des tâches à complexités variables, demandant soit une action simple ou une séquence d'actions pour être réalisées. Nous avons développé une infrastructure algorithmique d'apprentissage stratégique intrinsèquement motivé, appelée Socially Guided Intrinsic Motivation for Sequences of Actions through Hierarchical Tasks (SGIM-SAHT), apprenant la relation entre ses actions et leurs conséquences sur l'environnement. Elle organise son apprentissage, en décidant activement sur quelle tâche se concentrer, et quelle stratégie employer entre autonomes et interactives. Afin d'apprendre des tâches hiérarchiques, une architecture algorithmique appelée procédures fut développée pour découvrir et exploiter la hiérarchie des tâches, afin de combiner des compétences en fonction des tâches. L'utilisation de séquences d'actions a permis à cette architecture d'apprentissage d'adapter la complexité de ses actions à celle de la tâche étudiée.

Title: **Discovering and exploiting the task hierarchy to learn sequences of motor policies for a strategic and interactive robot**

Keywords : *Intrinsic Motivation, Goal-Babbling, Multi-task learning, Interactive learning, Hierarchical learning, Strategic learning*

Abstract : Efforts are made to make robots operate more and more in complex unbounded ever-changing environments, alongside or even in cooperation with humans. Their tasks can be of various kinds, can be hierarchically organized, and can also change dramatically or be created, after the robot deployment. Therefore, those robots must be able to continuously learn new skills, in an unbounded, stochastic and highdimensional space. Such environment is impossible to be completely explored during the robot's lifetime, therefore it must be able to organize its exploration and decide what is more important to learn and how to learn it, using metrics such as intrinsic motivation guiding it towards the most interesting tasks and strategies. This becomes an even bigger challenge, when the robot is faced with tasks of various complexity, some requiring a simple action to be achieved, other needing a sequence of actions to be performed. We developed a strategic intrinsically motivated learning architecture, called Socially Guided Intrinsic Motivation for Sequences of Actions through Hierarchical Tasks (SGIM-SAHT), able to learn the mapping between its actions and their outcomes on the environment. This architecture is capable to organize its learning process, by deciding which outcome to focus on, and which strategy to use among autonomous and interactive ones. For learning hierarchical set of tasks, the architecture was provided with a framework, called procedure framework, to discover and exploit the task hierarchy and combine skills together in a task-oriented way. The use of sequences of actions enabled such a learner to adapt the complexity of its actions to that of the task at hand.