Autonomous and Online Generation of Skills Inferring Actions Adapted to Low-Level and High-Level Contextual States

Carlos Maestre

Thèse de Doctorat
de la Sorbonne Université

Spécialité : **Informatique (EDITE)**

Présentée par : **M. Carlos Maestre**

Pour obtenir le grade de
Docteur en Informatique

Autonomous and Online Generation of Skills Inferring Actions Adapted to Low-Level and High-Level Contextual States

Rapporteurs	José Santos-Victor	-	IST, Universidade de Lisboa
	Verena V. Hafner	-	Humboldt-Universitat zu Berlin
Examinateurs	Raja Chatila	-	ISIR, Sorbonne Université, Paris
	Gianluca Baldasarre	-	ISTC, CNR, Rome
	Emre Ugur	-	Bogazici University, Istanbul
Directeur	Stéphane Doncieux	-	ISIR, Sorbonne Université
Directeur	Christophe Gonzales	-	LIP6, Sorbonne Université

To Leo

English Abstract

Robots are expected to assist us in our daily tasks. To that end, they may need to perform different tasks in changing scenarios. The number of dissimilar scenarios a robot can face is unlimited. Therefore, it is plausible to think that a robot must learn autonomously to perform tasks. A task consists in generating an expected change, i.e. an effect, in the environment, the robot configuration, or both. Therefore, the robot must learn to perform the right action on the environment to obtain the expected effect.

An approach to learning these actions is through a continuous interaction of the robot with its environment focusing on those actions producing effects on the environment. The acquired relation of applying an action on an object to obtain an effect is called *affordance*. During the last years many Research efforts were devoted to affordance learning. Related works cover from the learning of simple *push* actions on tabletop scenarios to the definition of complex cognitive architectures. These works rely on different *building blocks*, as vision methods to identify the position of the objects or predefined sensorimotor skills to generate effects on a constrained environment.

The use of predefined actions eases the learning of affordances, producing a rich and consistent information of the changes produced on an object. However, we claim that the use of these actions constrains the scalability of the available experiments to dynamic and noisy environments. The current work addresses the autonomous learning of a set of sensorimotor skills through interactions with an environment. Each skill must generate a continuous action to reproduce an effect on an object, adapted to the object position. Besides, each skill is simultaneously adapted to low-level perturbations, e.g. a change in the object position, and high-level contextual changes, e.g. a stove gets on.

Few questions arise while addressing the skill generation: first, how can a robot explore an environment gathering information with limited *a priori* information about it? We address this question through a babbling of the environment driven by an intrinsic motivation. We define a method, called Novelty-driven Evolutionary Babbling (NovEB), to explore possible robot's movements, while focusing on those that generate the highest novelty from the perception point of view. Perception relies on raw images gathered through the robot's cameras. A simulated PR2 robot, using this method, discovered on its own which regions of the workspace generate novel perceptions and focuses its exploration around them.

Second, how can a robot autonomously build a set of skills based on an initial information about the environment? We propose a method, named Adaptive Affordance Learning (A^2L), which endows a robot with the capacity to learn affordances associated to an object, both adapting the robot's skills to the object position, and increasing the robot's information about the object when needed. Two main contributions are presented: (1) an interaction process with the object adapting each movement to the fixed object position, decomposing each action into a sequence of discrete movements; (2) an iterative process to increase the information about the

object. These contributions are assessed in two experiments where a robot learns to push a box to different positions on a table. First, on a virtual setup on a simulated robotic arm. Finally, on a simulated Baxter robot.

Finally, we extend the previous skill generation to environments including both low-level and high-level perturbations. Initially, one or more kinaesthetic demonstrations of an action producing an effect on the object are provided to the robot, through a *Learning from Demonstration* approach. Then, a vector field is computed for each demonstration, generating information about the next movement to execute based on the robot context, composed of the relative positon of the object w.r.t. the robot's end-effector, and other high-level information. An action generator is learned, inferring in a closed-loop the next movement to reproduce an effect on the object based on the current robot context. In this work, a study is performed in order to select the best parametrization to build a *push to the right* and a *grasp* skill to reproduce an effect. Then, the selected parametrization is used to build a set of diverse skills, which are validated in several experiments performing tasks with different objects. The assessment of the built skills is directly performed on a physical Baxter.

French Abstract

Les robots sont censés nous aider dans nos tâches quotidiennes. À cette fin, ils peuvent devoir effectuer différentes tâches dans des scénarios changeants. Le nombre de scénarios dissemblables auxquels un robot peut faire face est illimité. Par conséquent, il est plausible de penser qu'un robot doit apprendre de manière autonome pour effectuer des tâches. Une tâche consiste à générer un changement attendu, c'est-à-dire un effet, dans l'environnement, la configuration du robot, ou les deux. Par conséquent, le robot doit apprendre à effectuer la bonne action sur l'environnement pour obtenir l'effet attendu.

Une approche de l'apprentissage de ces actions est à travers une interaction continue du robot avec son environnement en se concentrant sur ces actions produisant des effets sur l'environnement. La relation acquise de l'application d'une action sur un objet pour obtenir un effet est appelée *affordance*. Au cours des dernières années, de nombreux efforts de recherche ont été consacrés à l'apprentissage des affordances. Les travaux connexes couvrent l'apprentissage de simples actions *saissir* sur des scénarios de table à la définition d'architectures cognitives complexes. Ces travaux s'appuient sur différents *blocs de construction*, comme méthodes de vision pour identifier la position des objets ou des compétences sensorimotrices prédéfinies pour générer des effets sur un environnement contraint.

L'utilisation d'actions prédéfinies facilite l'apprentissage des affordances, produisant une information riche et cohérente des changements produits sur un objet. Cependant, nous affirmons que l'utilisation de ces actions limite l'évolutivité des expériences disponibles aux environnements dynamiques et bruyants. Le travail actuel porte sur l'apprentissage autonome d'un ensemble de compétences sensorimotrices à travers des interactions avec un environnement. Chaque compétence doit générer une action continue pour reproduire un effet sur un objet, adapté à la position de l'objet. En outre, chaque compétence est simultanément adaptée aux perturbations de bas niveau, par ex. un changement dans la position de l'objet, et des changements contextuels de haut niveau, par ex. un poêle s'allume.

Peu de questions se posent en abordant la génération de compétences: d'abord, comment un robot peut-il explorer un environnement rassemblant des informations avec des informations a priori a priori limitées à son sujet? Nous abordons cette question à travers un balbutiement de l'environnement animé par une motivation intrinsèque. Nous définissons une méthode, baptisée *Novelty-driven Evolutionary Babbling* (NovEB), pour explorer les mouvements possibles du robot, tout en mettant l'accent sur ceux qui génèrent la plus grande nouveauté du point de vue de la perception. La perception repose sur des images brutes recueillies à travers les caméras du robot. Un robot PR2 simulé, utilisant cette méthode, a découvert à lui seul quelles régions de l'espace de travail génèrent des perceptions nouvelles et concentre son exploration autour d'elles.

Deuxièmement, comment un robot peut-il construire de manière autonome un ensemble de compétences sur la base d'une information initiale sur l'environnement? Nous proposons une méthode, nommée *Adaptive Affordance Learning* (A^2L), qui

permet à un robot d'apprendre les affordances associées à un objet, en adaptant les compétences du robot à la position de l'objet et en augmentant les informations sur le robot. objet en cas de besoin. Deux contributions principales sont présentées: (1) un processus d'interaction avec l'objet adaptant chaque mouvement à la position de l'objet fixe, décomposant chaque action en une séquence de mouvements discrets; (2) un processus itératif pour augmenter les informations sur l'objet. Ces contributions sont évaluées dans deux expériences où un robot apprend à pousser une boîte à différentes positions sur une table. Tout d'abord, sur une configuration virtuelle sur un bras robotique simulé. Enfin, sur un robot Baxter simulé.

Enfin, nous étendons la génération de compétences précédente à des environnements comprenant à la fois des perturbations de bas niveau et de haut niveau. Initialement, une ou plusieurs démonstrations kinesthésiques d'une action produisant un effet sur l'objet sont fournies au robot, par le biais d'une approche *L'apprentissage par démonstration.* Ensuite, un champ de vecteur est calculé pour chaque démonstration, générant des informations sur le mouvement suivant à exécuter en fonction du contexte du robot, composé de la position relative de l'objet par rapport l'effecteur du robot, et d'autres informations de haut niveau. Un générateur d'action est appris, déduisant en boucle fermée le mouvement suivant pour reproduire un effet sur l'objet en fonction du contexte actuel du robot. Dans ce travail, une étude est effectuée afin de sélectionner la meilleure paramétrisation pour construire des compétences *pousser vers la droite* et *saissir* pour reproduire un effet. Ensuite, la paramétrisation sélectionnée est utilisée pour construire un ensemble de compétences diverses, qui sont validées dans plusieurs expériences exécutant des tâches avec différents objets. L'évaluation des compétences construites est directement réalisée sur un Baxter physique.

Acknowledgement

Four years ago I had a comfortable life in Madrid, close to my family and friends, a good job and a fancy apartment in the city center. I was in my early thirties, and I had a prosperous future in front of me. However, I needed a challenge in my life. And now I cannot be happier of the decision I chose one night at four o'clock in the morning: I was going to work with robots. Few months later I started an intership in robotics at the ISIR, in Paris, with very little knowledge about robotics and AI. That is the reason why this fantastic adventure would have not been able without the help of many people, of many friends. If I had the space to properly thank to each one of the people who helped me during my thesis, this section would become another chapter of the manuscript. However, I am going to briefly mention few people without who this manuscript would not have been possible.

Before starting with the acknowledgments I have to ask for forgiveness to the person I love more in the world, my son Leonardo: Leo, I did not spend enough time with you during more than one year to follow my personal goal working with robots. A goal that I hope someday we can share, and that it helps you in the future. I am sorry, son.

First, I would like to thank to my family for their support from the very beginning. I was able to face all the risks related to this adventure thanks to knowing they are always there for me, no matter what I do, when or where.

There is a person who always believed in me, who always supported me, who always guided my path, who corrected hundreds of times my writing, and to whom I will always be grateful: Stéphane. I did not only learn from him about robotics, but about how to make proper Research, among many other things. I also want to give special thanks to Christophe for his help to learn and apply Bayesian Networks, the core method of my work.

I believe when someone spends few years working in a place, projects come and go, but what it remains always with you is your workmates. In this sense, I am aware of how lucky I have been. I could have not find a better group of friends, with who I spent a wonderful time. I would like to specially mention Alex Vazques and Ryan Lober, much more than friends, my Parisian family. I would like also to thank Ghanim Mukhtar for all his support in my experiments. If Baxter is moving in the carried out experiments it is thanks to him. I also would like to mention to the group of friends who started at the same time that me as as inters, the excellent team of the famous J01, the Spanish corner guys at the lunch time, among others. And other wonderful people I met out of the lab, who helped me to chill, and who also helped in my French *evangelization*, as *faire du pique-nique, aller aux musées, manger* pheasant *au jouer à la petanque*.

When people ask me what I do in life I always say the French government pays me to play with robots. Although being a joke it is partially true, and that is why I also want to thank all the support from the ISIR, from the thesis grants to the

help provided by the administrative and technical staff.

Last, but not least, I am very glad that some of the most relevant researchers in the developmental robotics field accepted to be part of my jury. I hope we can work together in the future, for many years.

Contents

Chapter 1

Introduction

Contents

1.1 Autonomous Robots

An autonomous agent has been defined as *"any embodied system designed to satisfy internal or external goals by its own actions while in continuous long-term interaction with the environment in which it is situated"* (Beer, 1995, page 173). Autonomous robots are expected to help us in our daily tasks[1]. Depending on the complexity of the task goal to reach, the capacities of the robot need to be versatile enough to adapt to the situations it will be faced with. For instance, in order to clean an area a vacuum cleaner robot just executes one of its available behaviors reacting to the sensory information acquired through its sensors (Forlizzi and Disalvo, 2006). Conversely, in order to reach certain location an autonomous car must analyze the sensory information and combine it with previous knowledge to plan the next steps to perform, accordingly adapting its behavior to its environment (Thrun et al., 2007). Therefore, a robot performing a complex task should understand its environment to select and execute the next adapted action to reach the task goal.

A robot can be endowed with built-in capacities defined by a designer. In constrained environments a robot can reach a task goal using them. However, in unconstrained environments the number of dissimilar scenarios a robot can face is unlimited. Similarly, it is very complex to foresee all the situations in which a robot can be involved. For example, the *Spirit rover* behavior was programmed by a team of engineers to wander around a specific area in Mars gathering information (Sanderson, 2010). Nevertheless, during the wandering it became entrapped into a sandpit, and it was unable to release itself because of not being programmed for it. More recently, during the last DARPA Robotic Challenge (Atkeson et al., 2015)

[1]The nomenclature used in the current manuscript is inspired by the nomenclature described in the Deliverable 6.1 of the DREAM project (Doncieux et al., 2015b).

several robots failed to perform a trial due to the fact that the execution of built-in actions under incorrect circumstances. One trial consisted in turning a valve 360 degrees to the left. The robot of the NEDO-JSK team did not situate itself properly in front of the valve, and after the execution of a built-in grasping action not grasping anything the built-in turn action made the robot fell. Therefore, it is plausible to think that a robot must develop its own behavioral capacities with the minimum *a priori* knowledge, and learn when to use them, in order to perform a task.

During the last decades another approach has emerged concerning the generation by a robot of its own behavioral capacities through interactions with the environment, similarly as infants do, called Developmental Robotics (Asada et al., 2001, 2009; Lungarella et al., 2003; Weng, 2004; Meeden and Blank, 2006; Stoytchev, 2009; Cangelosi et al., 2015). The underlying idea is that through a developmental process a robot executes a trial-and-error approach learning from its failures to improve its performance. In the psychology literature there are different theories explaining infant development (Newcombe, 2013): *empiricism*, which suggests that babies are born with very little initial capabilities, and knowledge is based on the experiences acquired by the sensors; *nativism*, in which knowledge and skills are innate to newborns; and *contructivism*, proposing that newborns seek for knowledge to construct their own world model, developing abstract concepts through the interaction with the environment. Based on a constructivist approach Guerin et al. (2013) make a study of infant developmental process from basic actions to task planning using tools, aimed at providing insights about the developmental process to roboticists. Guerin defines two parallel tracks of development: the *abstract track*, composed of the abstract representations the infant uses, and the *concrete track*, representing the development of sensorimotor schemas (Piaget and Cook, 1952), i.e. progressively building complex behaviors from simple ones. *Sensorimotor schemas* are described as the minimal unit of knowledge connecting a *context*, an *action* and an *effect*. More precisely, given a context an agent selects and executes the next action to produce a desired effect on an object. Although in the developmental robotics literature there is not a clear definition of *context* explaining its content and boundaries, based on the available works we propose a definition:

Definition 1 *Context: A context represents all the circumstances related to a robot-object interaction at a certain instant of time.*

Definition 2 *Robot-object interaction: A robot-object interaction represents the trajectory executed by the robot's end-effector to change the features of the object, and these features during the robot's movement.*

A context is composed of *contextual states*, or just *states*:

Definition 3 *Contextual state: A contextual state represents a feature of a robot or an object related to the interaction between the robot and its environment.*

There are two types of contextual states:

- *low-level contextual states*, or just *low-level states*, related to the execution of an action, i.e. motor control, represented by continuous values, e.g. an object position (Calinon et al., 2010).
- *high-level contextual states*, or just *high-level states*, representing higher level concepts related to the objects, represented by continuous and discrete values, e.g. an object color or *circleness* (Montesano et al., 2008).

Guerin mentions that in psychology the terms *sensorimotor schema* (Piaget and Cook, 1952), *sensorimotor skill* (Fischer, 1980), *sensorimotor process* (Smith, 2009) and *perception-action routine* (Lockman, 2000) can be considered as equivalent. Another close terms used in the developmental robotics literature are *affordances* (Gibson, 1966) and *behavior* (Sahin et al., 2007). In the current manuscript, we use:

Definition 4 *Sensorimotor skill: A sensorimotor skill, or just skill, is the process transforming robot contextual states into robot motor commands*[2].

Although the learning of skills can lead to the development of high cognitive capabilities, as the sense of agency or self-awareness (Schillaci et al., 2013; Vernon, 2014), in the current manuscript skills are focused on generating actions to accomplish a task.

For a given context, it is supposed that a task requires the execution of a sequence of skills. When a skill produces the expected effect the next skill runs. This process continues until the task goal is reached. It is possible that a task plan cannot be executed, e.g. an effect cannot be produced or an object disappears from the environment, and thus a new task planning must be performed. Works in the robotics literature in which a robot learns to perform a task can be clustered in two groups:

- *Learning predictive models:* the predictive models learn the relation of the high-level states of the context. Task resolution is based on a planning process relying on these models, sometimes called affordances. An *affordance* is initially defined as the actions an agent can *afford* to execute through direct perception of an object (Gibson, 1966, 1986). In robotics, it has been defined as the acquired relation of applying an action on an object to obtain an effect (Sahin et al., 2007) (see Section 2.1.3 for an elaborated description discussion). In most of the experiments within the affordance literature a built-in repertoire of skills is available. In these works, given the contextual high-level states and an effect to produce, the predictive models based on affordance knowledge infer which skill among the available ones can reproduce the effect. Then, a task planner selects the skill related to the selected action, and the built-in skill executes the action based on the contextual low-level states (see top of Figure 1.1). A comparison of related works is available in Section 2.1.3.

[2]Our work gets inspiration from infant psychology, although the proposed methods do not directly model infant behavior.

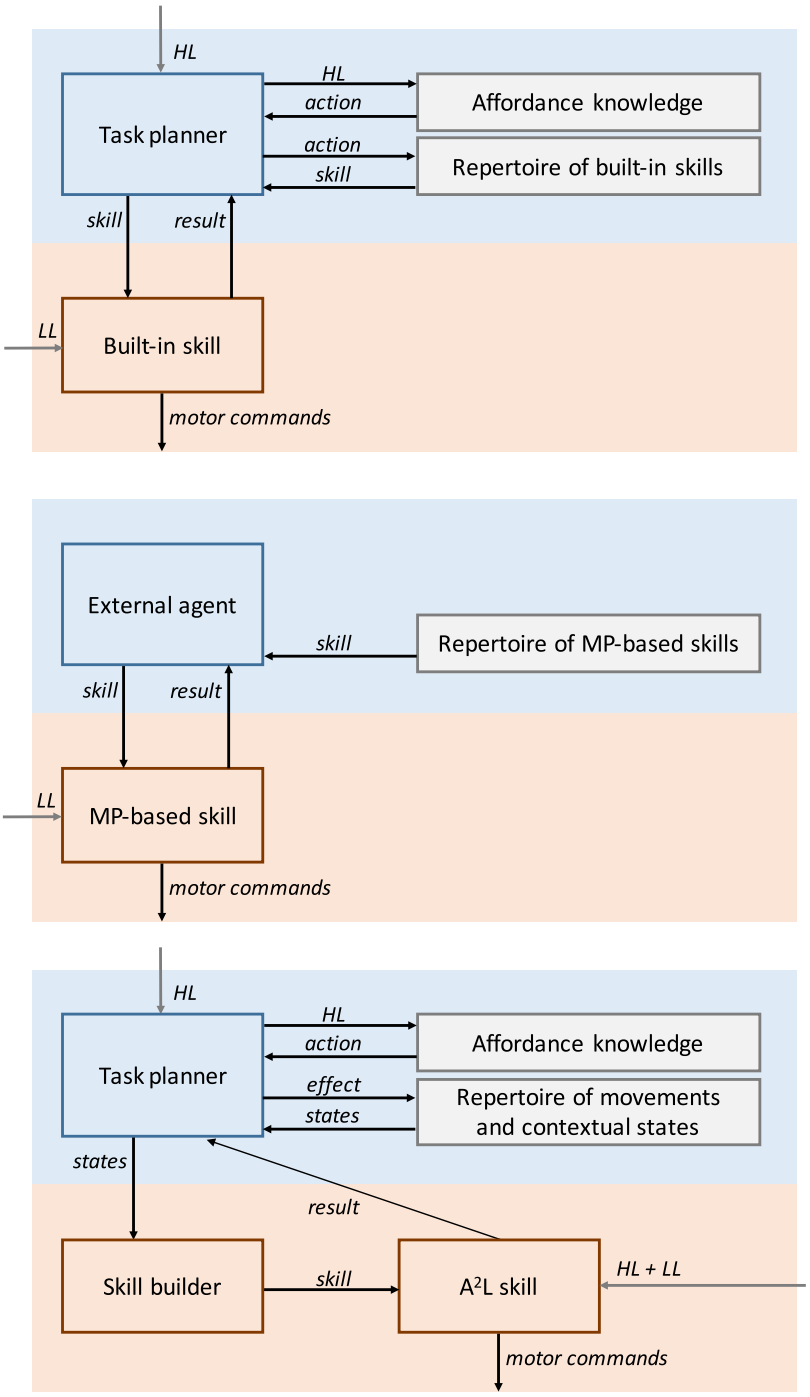


Figure 1.1: Workflow of different approaches performing a task. The light blue areas represent the selection of the next action to be executed by a robot. The light orange areas represent the execution of the action. *LL* and *HL* stand for low-level and high-level contextual states respectively, whereas *MP* stands for motion primitive. The white boxes represent available knowledge. At the top, workflow of works focused on the learning of the predictive models. At the middle, workflow of works focused on running a skill reproducing an action demonstrated by another agent, e.g. the experiment designer. At the bottom, workflow of our method, A^2L, in which motor commands are based on both HL and LL (see Section 1.2).

- *Building skills:* in these works an action is demonstrated to a robot by an external agent, based on an approach called Learning from Demonstration, *LfD* (further explained in Section 2.1.1). Then, a method learns a skill to reproduce the action. These works focus on the learning of one or few actions, and thus there is no need to perform a task planning, i.e. the designer in charge of the experiment directly selects the skill to run. The skill execution is driven by a motion primitive based on the low-level states of the context (see workflow at the middle of Figure 1.1). A *motion primitive* (MP) generates a continuous motion of a robot's end-effector reproducing a demonstrated action. The learned MPs are robust to changes, i.e. *perturbations*, during the execution of an action produced externally or by the lack of accuracy of robot sensors. Perturbations represent either *spatial* or *temporal changes* (Gribovskaya et al., 2011). Spatial perturbations are those related to a change of the spatial values of a state. For example, changes of the initial position of the robot's end-effector w.r.t. the object position before the execution of an action, or changes of the object position during the execution. Temporal perturbations are those related to a change of the duration of an action, i.e. if the robot's end-effector gets stuck or delayed during the execution of the action. A comparison of related works is available in Section 2.1.2.

We have identified several gaps in the current robotics literature w.r.t. the autonomous performance of tasks using objects: first, there is a lack of works combining task planning based on predictive models with motion control, i.e. combining high-level action selection with low-level adaptive action execution. To the best of the authors' knowledge, Kroemer et al. (2012) is the only work combining these features. In this work, a pouring task experiment is executed, in which a robotic arm *grasps* a watering can and *pours* water into a glass. The main objective of this experiment is to use affordance knowledge to learn predictive models mapping subparts of objects to MPs based on direct perception.

Second, in the same vein, the execution of a skill is disconnected from the robot cognitive architecture, i.e. it is a *black box* for the architecture. A skill is evaluated by the visual result obtained when executing an action, i.e. the designer of an experiment evaluates if the effect of the action is as expected. If the effect is not as expected the designer analyses the execution of the skill to improve the skill performance. We consider that providing a trace of the internal skill process to the cognitive architecture can be useful in higher-level stages to identify behavioral regularities, which can be exploited for transfer learning and generalization techniques.

Finally, in the experiments available in the developmental robotics literature, contexts are limited to high-level states *affording* similar actions on objects, e.g. *shape* and *dimension* to *push*, *grasp* and *stack* objects (Szedmak et al., 2014). Conversely, in daily environments object contextual states comprehend both high-level and low-level states. We suggest that in order to scale up the use of the learning methods used in the literature to our daily environments these methods should si-

multaneously address both types of states. Besides, high-level states should also represent different and less stable features. For instance, the context of a robot cooking a piece of meat can be composed of different types of high-level states, e.g. the meat size, quite stable, and the meat color, less stable, together to low-level states, e.g. the meat position. As a consequence of the constant stability of the high-level states in the literature, actions executed by skills are only reactive to low-level states, not reacting to changes of high-level states.

At this point a question arises: *how can a robot autonomously build skills*

- *in contexts with different levels of complexity,*
- *simultaneously adapting to both*
  - *low-level spatio-temporal perturbations*
  - *and high-level state changes,*
- *and generating a trace of its internal functioning.*

1.2 Adaptive Affordance Learning (A^2L)

We propose a method named Adaptive Affordance Learning (A^2L) that autonomously builds skills to reproduce effects on objects. Given a *dataset of interactions* the method builds *on-the-fly* an *ad hoc* skill to infer an action reproducing an effect (see workflow at the bottom of Figure 1.1). Actions are inferred based on the robot context. More precisely, actions are inferred adapted to the low-level and high-level contextual states representing features of the robot and its environment at each instant of time. In order to simultaneously adapt to both types of contextual states they are discretized, based on a *discretization configuration*. In the current manuscript, this configuration is empirically computed based on experience.

A^2L is composed of two complementary processes:

- Skill Building: given a dataset of interactions this process builds skills that infer actions to reproduce an effect on an object.
- Iterative Interaction Acquisition and Validation: an iterative process generating new interactions and validating them.

Given a dataset of interactions *the skill building process* generates one or more skills. A skill is an *action generator*.

Definition 5 *Action: An action is a sequence of movements to reproduce an effect on an object.*

Definition 6 *Action Generator: An action generator infers a movement to reproduce an effect given a context.*

Definition 7 *Movement: A movement is a displacement of the robot's end-effector between two subsequent instants of time to reproduce an effect.*

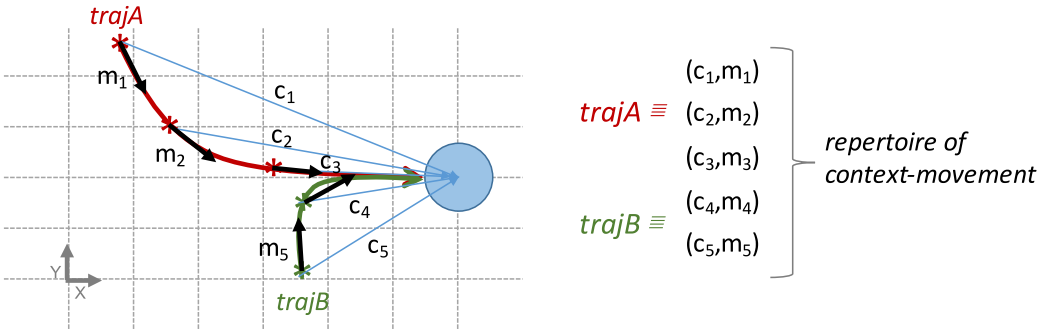


Figure 1.2: Example of transformation of two interactions of different size into a repertoire of movements and contextual states. The *blue circle* represents an object. The trajectory of two interactions A (red) and B (green) of the robot's end-effector reproduce the same effect, moving an object in certain direction (e.g. to the right). These trajectories are split up into tuples (context, movement), e.g. c_1, m_1. Thus, these two trajectories generate 5 robot-object relations, independent among them. Given a robot-object relation, e.g. the *distance* and *orientation* of c_1, the end-effector must execute the lcorresponding movement, e.g. m_1.

Each movement is obtained using a dynamical system (Perrin and Schlehuber-Caissier, 2016), and thus the skill is robust to low-level spatio-temporal perturbations. The dynamical system generates a *vector field*, in which a vector represents a velocity and orientation of the end-effector (Schaal, 1999; Ijspeert et al., 2002). Movements are vectors adapted to the object position.
This skill building process consists of two steps: a first step transforming the *dataset of interactions* into a *repertoire of movements and contextual states*. This step transforms interactions into movements and contextual states in order to learn the skill (see Figure 1.2).

Definition 8 *Repertoire of Movements and Contextual States: A repertoire of contextual movements is a dataset containing movements related to a context.*

The second step learns an action generator based on the transformed repertoire. In the current manuscript, an action generator is implemented as a Bayesian Network[3] (BN) (Pearl, 1988) similarly to the work of Montesano et al. (2008) among others. The BN handles random variables representing movements and low-level contextual states, represented with continuous values, and high-level contextual states, represented with both continuous and discrete values. Lauritzen (1992) introduced a BN handling continuous and discrete variables, and other works applied similar concepts to robotics (Osório et al., 2010; Stramandinoli et al., 2017). However, Osorio mentions that the performance of a BN combining continuous and discrete variables may be much slower than directly using discrete variables. As each cycle

[3]In the remaining of the manuscript we use indistinctly the terms *skill*, *action generator* and *BN*.

of the close loop performed by a skill is very short, A^{2}L uses discrete values to represent *effects*, *movements*, and *low-level and high-level states*. Effects are already discrete. Movements and contextual states are discretized in the previous step, and thus the available repertoire of movements and contextual states is already discretized. Therefore, the output of this step is an action generator that infers discrete movements to reproduce an effect in an object based on discrete low-level and high-level contextual states.

In order to build a skill, the repertoire must contain movements and low-level states representing the execution of the action. Besides, it can also represent high-level states defining the circumstances in which the action can be executed. For example, a repertoire can be composed of movements and low-level states representing how a robot can press a button from different relative positions of its end-effector. Therefore, A^{2}L would build a skill pressing the button from those positions. However, if the button can be only pressed under certain circumstances, e.g. a stove is off, the dataset would be extended with this high-level state. And thus A^{2}L would build a skill to press the button from the relative positions only when the stove is off; either if this is a consequence of a robot action, or if it was externally turned on. This feature provides to a higher-level stage with a high flexibility to decide the circumstances in which an action can be executed, i.e. it allows a robot to perform an action in different contexts.

The *iterative process* generates a dataset of interactions, used to build the skills, through interactions of the robot with its environment. The process consists of three phases, executed in an iterative fashion: in the first phase, called *Interaction Acquisition*, an exploration of the robot's environment is performed. The result of this exploration is a dataset of interactions. This dataset can be alternatively provided to the robot through a demonstration by an external agent (LfD). The second phase, called *Skill Generation*, executes the Skill building process for the current dataset of interactions. In the third phase, called *Interaction Validation*, the skill obtained in the previous phase is executed to reproduce a set of effects on an object. In a closed loop, the action generator analyses the context, both low-level and high-level states, and infers the next movement of the end-effector to execute, adapted to the object position. Thus, the dataset of interactions used to build the skill is validated. The iterative process stops after the phase 3 if all the effects are reproduced or after a preset number of iterations is reached. Otherwise, the phase 1 is executed again.

A^{2}L relies on a previous developmental stage, consisting in the identification of the relevant contextual states for a skill. E. J. Gibson calls to this process *differentiation* (Gibson, 2000, 2003), which is out of the scope of our work (a recent and relevant approach is available in Jonschkowski and Brock (2015); Jonschkowski et al. (2017)). Besides, although a robot endowed with A^{2}L can autonomously learn to interact with the environment to reproduce an effect, the execution of the method requires some *a priori* information, e.g. the discretization. This information is described in the next chapters while describing the method.

1.3 Contributions

The main contributions of A^2L and thus of the current manuscript are:

- The autonomous generation and validation of the dataset of interactions.
- The online generation of skills based on a repertoire of discrete movements and states.
- The execution of actions simultaneously adapted to spatio-temporal perturbations and high-level contextual changes.
- The generation by the skills of a discrete trace which can be exploited in higher-level stages, for example for transfer learning or generalization.
- Regarding the study of the state-of-the-art, a literature survey focused on action selection and execution is available in Section 2.1.2.

Table 1.1: Contents addressed in each chapter

			Chapter		
	Content	*Type*	*4*	*5*	*6*
1	Dataset of interactions		X	X	X
2	Available contextual states	Low-level states	X	X	X
		High-level states			X
3	Information acquisition	autonomous	X	X	
4		from demonstration			X
5	Actions adapted to pertutbations	in static environment		X	
6		in dynamic environment			X
7	Skill validation by a physical robot			X	X

1.4 Dissertation Outline

This section details the content of each chapter of the current manuscript (see table 1.1 for more details about chapters presenting this thesis contribution):

Chapter 1 The current chapter identifies relevant features for the resolution of tasks adapted to the current context. Then, we propose a new method to generate actions adapted to both low-level and high-level contextual states.

Chapter 2 This chapter covers the *state-of-the-art* of different areas of the developmental robotic literature related to the proposed method. First, some psychology insights explaining how infants interact with the environment are provided. In order to acquire contextual states relevant for a task, both the exploration of an environment driven by intrinsic motivations, and the skill learning from demonstration are detailed. The affordance theory is introduced and some relevant works are explained. Afterwards, a comparison of different works reproducing actions previously demonstrated by an external agent are described. Finally, some works focused on task planning are described.

Chapter 3 The methods on which A^2L relies are described in this chapter, i.e. Novelty Search for the exploration of an environment, *hill-climbing* and K2 to learn the action generator, a diffeomorphic matching algorithm to generate actions robust to low-level spatio-temporal perturbations, and PDDL (Planning Domain Definition Language) as task planner in experiments validating the generated repertoire of states.

Chapter 4 We present a method named Novelty-driven Evolutionary Babbling (NovEB), designed to perform a task-agnostic exploration of an unknown environment. Its main feature is to look for actions that maximize novelty in the raw sensorimotor space. It is based on Novelty Search (Lehman and Stanley, 2011), which relies on Evolutionary Algorithms driven by a behavior novelty criterion. The outcome of this approach is a raw dataset representing the interactions of a PR2 robot with its environment, which can be exploited by methods as A^2L. The content of this chapter has been published in Maestre et al. (2015).

Chapter 5 This chapter presents the two complementary processes of A^2L: the Iterative Repertoire Acquisition and Validation and the Skill Builder. A robot autonomously explores its environment using random actions building a repertoire of discrete movements and low-level states. The environment is static, i.e. the position of the objects only changes when the robot touches them. An experiment is executed in simulation, pushing a box in different directions on a table. Once the repertoire is available, the states are tested by the physical Baxter continuously pushing the box. The content of this chapter has been introduced in Maestre et al. (2016) and published in Maestre et al. (2017b).

Chapter 6 The Skill Builder available in Chapter 5 is updated to build skills robust to perturbations, using both low-level and high-level states. The states are built using raw data directly demonstrated to a physical Baxter robot by an external agent, i.e. learning from demonstration (Billard and Calinon (2016)). The assessment of the generated skills is directly performed on the Baxter through a set of experiments performing tasks of increasing complexity. The content of this chapter has been partially presented in Maestre et al. (2017a).

Chapter 7 This chapter discusses about the obtained results by the method. Also it identifies some drawbacks, and it proposes how to address them in future works.

Chapter 8 Finally, some general conclusions of the manuscript are presented.

1.5 Publications

In conference proceedings:

- Maestre, C., Mukhtar, G., Gonzales, C., and Doncieux, S. (2017, September). Iterative affordance learning with adaptive action generation. In IEEE International Conference on Developmental and Learning and on Epigenetic Robotics (ICDL-Epirob). Referenced in the bibliography as Maestre et al. (2017b).
- Maestre, C., Cully, A., Gonzales, C., and Doncieux, S. (2015, August). Bootstrapping interactions with objects from raw sensorimotor data: a Novelty Search based approach. In IEEE International Conference on Developmental and Learning and on Epigenetic Robotics (ICDL-Epirob). Referenced in the bibliography as Maestre et al. (2015).

In workshops:

- Maestre, C., Mukhtar, G., Gonzales, C., and Doncieux, S. (2017, October). Context-Based Generation of Continuous Actions to Reproduce Effects on Objects. In the Third International Workshop on Intrinsically Motivated Open-ended Learning (IMOL). Referenced in the bibliography as Maestre et al. (2017a).
- Maestre, C., Mukhtar, G., Gonzales, C., and Doncieux, S. (2016, September). Bootstraping manipulation skills to learn affordances in open-ended environments. In the workshop Autonomous Perception: Applying Sensorimotor Contingencies and Predictive Processing to Developmental Robotics (ICDL-Epirob). Referenced in the bibliography as Maestre et al. (2016).
- Ecarlat, P. and Cully, A. and Maestre, C. and Doncieux, S. (2015, September). Learning a high diversity of object manipulations though an evolutionary-based babbling. In the workshop Learning Object Affordances (IROS).
- Legoff, L. and Maestre, C. and Doncieux, S. (2015, September). Visual saliency-based babbling of unknown dynamic environments. In the workshop Learning Object Affordances (IROS).

To appear:

- Joel Lehman, Jeff Clune, Dusan Misevic3, Christoph Adami, Julie Beaulieu, Peter J Bentley, Samuel Bernard, Guillaume Beslon, David M Bryson, Frederic Carrere, Nick Cheney, Antoine Cully, Stephane Doncieux, Fred C Dyer, Andreas Ehinger, Kai Olav Ellefsen, Robert Feldt, Stephan Fischer, Dario

Floreano, Stephanie Forrest, Antoine Frenoy, Christian Gagne, Leni Le Goff, Laura M Grabowski, Babak Hodjat, Laurent Keller, Carole Knibbe, Peter Krcah, Richard E Lenski, Hod Lipson, Robert MacCurdy, Carlos Maestre, Frederic Mansanne, Risto Miikkulainen, Sara Mitri, David E Moriarty, Jean-Baptiste Mouret, Anh Nguyen, Charles Ofria, Marc Parizeau, David Parsons, Robert T Pennock, William F Punch, Thomas S Ray, Marc Schoenauer, Eric Schulte, Karl Sims, Kenneth O Stanley, Francois Taddei, Danesh Tarapore, Simon Thibault, Westley Weimer, Richard Watson, Jason Yosinski (2018). The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities. Trends in Ecology and Evolution.

Chapter 2

State-of-the-art

Contents

This section introduces works of the developmental robotics literature and the learning from demonstration literature related to A^2L. As explained in Chapter 1, our method (1) explores the robot's environment to acquire information of the interactions between the robot and an object, (2) builds skills and predictive models based on this information, and (3) validates the skills reproducing effects on objects. Therefore, this section presents works for each one of these steps. Similarly to our experiments, the works introduced in this section are performed by either anthropomorphic robots or robotics arms.

2.1 Interacting with the Environment

A robot can be endowed with built-in knowledge to interact with its environment defined by a designer. However, as discussed in Section 1.1 this approach constrains the number of situations and environments the robot can face. During the last decades an approach has emerged concerning the generation by a robot of its own skills and predictive models through interactions with the environment, similarly as infants do. This is called Developmental Robotics (Asada et al., 2001, 2009; Lungarella et al., 2003; Weng, 2004; Meeden and Blank, 2006; Stoytchev, 2009; Cangelosi et al., 2015). The underlying idea is that a robot improves its performance executing a trial-and-error approach, learning from its failures.

Neuroscience has revealed how action is coupled to perception (Kandel et al., 2014; Snyder, 2000; Buneo et al., 2002; Gallivan et al., 2013) and infant psychology has demonstrated that action is key in the development of cognition (Adolph and Berger, 2015; Von Hofsten, 2009, 2013). The capabilities of a robot endowed with A^2L are similar to those capabilities infants acquire at early stages (Jamone et al., 2016). By 7-8 months of age infants pose a repertoire of basic actions, such as

grasping, holding and shaking, allowing them to perform a goal-free exploration of their environment called *motor babbling* (Meltzoff and Moore, 1997). At this age, babbling involves performing actions on single objects. The result of this babbling is the identification of actions and its posterior correlation to effects (Elsner and Hommel, 2001). At around 9 months of age, infants use this knowledge to learn object affordances (Adolph and Kretch, 2015). They use the acquired affordance knowledge to perform simple tasks, i.e. achieving simple goals predicting the changes of the environment (Piaget and Cook, 1952). Infants of 12 months of age are able to learn more complex action-effect mappings by imitating other agents' actions and demonstrations (Want and Harris, 2002), and thus to extend their affordance knowledge.

2.1.1 Information Acquisition

A key step to learn to interact with the environment is the acquisition of information to build models to that end. In the analyzed works there are two main approaches:

- (i) the robot explores the environment in an *unsupervised* fashion using motor babbling, similarly as infant exploratory activities, e.g. mouthing, feeling, licking, and shaking. Exploration reduces uncertainty (Gibson, 1969) and improves predictability (Gibson, 1994). Motor babbling is performed either randomly or driven by an *intrinsic motivation* (Barto, 2004). The exploration is performed as an iterative process in which the acquired information is analyzed and used to define the next robot action.

- (ii) the robot acquires the information from a demonstration performed by an external agent. This is called Learning from Demonstration (LfD), or Programming by Demonstration (PbD). In the current manuscript, we focus on *supervised* kinesthetic demonstrations of interactions of a robot with an object, i.e. demonstrations in which an external agent performs an action moving a robot end-effector.

Unsupervised Exploration

Random motor babbling consists in arbitrarily modifying the values of a robot's end-effector in order to move it. Mugan and Kuipers (2012) and Demiris and Dearden (2005) are examples of works that use this type of explorations to bootstrap their systems. Random motor babbling provides the capacity to explore an environment with very little *a priori* information. For example, not providing information about the composition of the environment, only providing the capacity of executing actions, i.e. providing the robot's kinematic model. However, it also presents some limitations. On one hand, it can execute many movements that do not produce any contact with the objects composing the scene, repeatedly exploring regions not providing any information. On the other hand, when an interaction happens it has a small impact, if any, on the rest of the babbling process.

Baldassarre and Mirolli (2013) provides a complete study of intrinsic motivations, providing both a theoretical explanation and references to key related works. In this work three types of intrinsic motivations are defined: (i) prediction-based intrinsic motivations, exploring the less predictable areas of the environment state space; (ii) novelty-based intrinsic motivations, focused on the regions generating more novel results in the robot perception; and (iii) competence-based intrinsic motivations, prioritizing the exploration of regions where the learning of skills is higher.

A relevant work based on intrinsic motivations is Oudeyer et al. (2007), which proposes an exploration method called Intelligent Adaptive Curiosity. Endowed with this method a robot executes actions in its search space and it uses the corresponding data to train predictors, called experts, which progressively get specialized in different regions of the sensorimotor space. The next action to apply is then randomly chosen in the action space covered by the expert with the maximum learning progress. This method has been used in different works, for example to explore an environment by a robot (Mugan and Kuipers, 2012) or for phonetic learning (Moulin-Frier and Oudeyer, 2012). Recently, Baranes and Oudeyer (2013); Forestier and Oudeyer (2016, 2017) have extended the method allowing one to use it in more complex and challenging scenarios, for instance using tools.

Supervised Demonstrations

In Billard's own words: *"LfD is not a record and play technique. LfD implies learning, henceforth, generalization"* (Billard and Calinon, 2016, page 1995). Namely, given one or more demonstrations a robot can learn to perform an action that can be applied in contexts different from those in which the demonstrations were performed. Moreover, a robot can learn to reach a task goal learning from demonstrations all the required actions. This sequence of actions can be demonstrated to a robot in two different ways: on the one hand, performing all the task during a single demonstration. Then, this demonstration can be segmented into actions relevant for the task (Zimmer and Doncieux, 2017; Kulić et al., 2012; Niekum et al., 2012). On the other hand, different actions can be individually demonstrated to the robot. Once this repertoire of actions is available the objective would be learning the right sequence of action execution in order to reach the task goal. This sequence could be also demonstrated to the robot in a second stage, or it could be autonomously learned by the robot, for example using Reinforcement learning (Schaal, 1999). Relevant examples of works using LfD to build a skill generating an action are described in Section 2.1.2

2.1.2 Building skills

This section presents some of the most relevant works learning *motion primitives*, i.e. MPs, reproducing an action. These action demonstrations are usually performed by an external agent, and thus works reproducing actions are within the

Table 2.1: Comparison of methods generating adaptive skills, *DR* stands for *Discrete representation*, *IC* stands for *Inference capability*, *SP* stands for *Spatial perturbation*, *TP* stands for *Temporal perturbation*, *TD* stands for *Time-dependency*, *St* stands for *Stable*, *NE* stands for *Number of examples*, and *C* stands for *Combination of MPs*

Type	ID	Publication reference	MP learning method	DR	IC	SP	TP	TD	St	NE	C
Trajectory-based	1, 2	Ijspeert et al. (2002, 2013)	DMP	No	No	Final position	No	Yes	Yes	1	No
	3 4	Pastor et al. (2009), Kober et al. (2010)	DMP	No	No	Final position	No	Yes	Yes	1	No
	5	Kroemer et al. (2012)	DMP	No	No	Final position and velocity	No	Yes	Yes	1	Yes
	6	Muelling et al. (2013)	MoMP	No	No	Final position and velocity	No	Yes	Yes	1	Yes
	7, 8	Paraschos et al. (2013, 2017)	ProMP	No	No	All positions and velocities	Yes	No	Yes	M	Yes
State-based	9	Calinon et al. (2007)	GMR-DS	No	Yes	No	Yes	No	No	M	-
	10, 11	Calinon et al. (2010, 2011)	HMM + GMR	No	Yes	Final position	Yes	No	No	M	-
	12 13 14	Khansari-Zadeh and Billard (2011), Khansari-Zadeh and Billard (2014), Kim et al. (2014)	SEDS	No	Yes	Final position	Yes	No	Yes	M	-
	15	Calinon (2016)	TP-GMM	No	Yes	All positions	Yes	No	Yes	M	-
		The current work	**A^{2}L**	**Yes**	**Yes**	**All positions**	**Yes**	**No**	**No**	**M**	**Yes**

LfD literature (Section 2.1.1). In Table 2.1 there is a comparison of these works. The variables selected for the comparison represent the features identified in Section 1.1 for the execution of actions interacting with the environment to solve a task: discrete representations to handle high-level states changes, a strong inference capability to infer the next action to perform, and mechanisms to be robust to spatio-temporal low-level perturbations. Besides, other features studied within the motor control literature are added: the stability of a MP, the number of examples needed for the learning, and the combination of different MPs to reproduce an unseen action.

Paraschos categorizes MPs in *trajectory-based representations* and *state-based representations*: *"Trajectory-based primitives typically use time as the driving force of the movement. They require simple, typically linear, controllers, and scale well to a large number of DoFs. In contrast, state-based primitives do not require the knowledge of a time step but often need to use more complex, non-linear policies."* (Paraschos et al., 2017, page 2). On the one hand, trajectory-based MPs are based on dynamical systems (DS), which represent motion as time-independent functions. The principal disadvantage of DS is that they do not ensure the stability of the system. A relevant method to represent trajectory-based MPs is Dynamical Movement Primitives, i.e. DMPs (Ijspeert et al., 2002, 2013). This method adds an external stabilizer based on time to generate stable motion. In the current work, the term *stable* includes *global asymptotic stability*, i.e. motions converge towards a single position, where the velocity profile of the robot's end-effector tends to zero. A drawback of time-dependent DS is the generation of inappropriate accelerations of motion during execution if the end-effector gets delayed with regard to the expected execution. The delay can be externally produced, e.g. by another agent, or it can be the result of the unexpected interaction with the environment. Pastor et al. (2009) and Muelling et al. (2013), among others, introduce few improvements to the original DMPs. ProMP (Paraschos et al., 2013, 2017) represents a relevant enhancement improving most of the features in the same framework. For example ProMP are time-independent and stable, avoiding the previous drawback. Also, the action execution is consistent with regard to spatial perturbations at any position of the executed trajectory; and MPs are learned from multiple combined demonstrations. It is important to underline that trajectory-based MPs do not have inference capabilities.

On the other hand, state-based MPs are time-independent with inference capabilities by definition. Also, multiple demonstrations are provided, and therefore there is no need to combine MPs. States are represented by Gaussian functions, and computed based on the demonstrated trajectories of the robot's end-effector. For a specific position of the robot's end-effector, weights are computed using Hidden Markov Models (HMM) to identify the next state based on the current state. Once the state is available, the motion is computed using Gaussian Mixture Regression (GMR). The initial works (Calinon et al., 2007, 2010, 2011) do not generate stable actions. This drawback was solved in posterior studies by a method called Stable Estimator of Dynamical Systems (SEDS) (Khansari-Zadeh and Billard, 2011, 2014;

Kim et al., 2014). This method ensures stability computing Lyapunov candidates (Slotine and Li, 1991). However, SEDS can only handle spatial perturbations at the final position of the demonstrated trajectories. Calinon (2016) solves this problem through the generation of a set of waypoints around the trajectory, with different reference frames, adapting the perturbations independently if necessary.

Kroemer et al. (2012) is a work close to the features of the action generator of A^2L using DMPs, making direct relations between continuous actions and object affordances. In this work, a pouring task experiment is executed, where a robotic arm *grasps* a watering can and *pours* water into a glass. These actions are previously demonstrated to the robot by an external user (LfD), and they can be generalized to watering cans of different size and mugs. The pivotal idea behind this experiment is making a relation between subparts of the involved objects with the DMP. For grasping, the relevant subpart is the handle, whereas for pouring the relevant part is the lip of the glass.

2.1.3 Learning Predictive Models

In the developmental robotic literature, a robot uses predictive models to select the next action to execute while performing a task. These models are called *affordances* when they are directly perceived from the environment (Gibson, 1966). In this section we briefly explain the notion of affordances. For further details, Jamone et al. (2016) and Zech et al. (2017) are a couple of recent surveys providing a complete overview of the affordance literature. Besides, this section presents in Tables 2.2 and 2.3 a comparison of different works in the literature. The comparison is focused on the built-in actions provided in those works and relevant features: discrete representations to handle high-level states changes, a strong inference capability to infer the next action to perform, and mechanisms to be robust to spatio-temporal low-level perturbations.

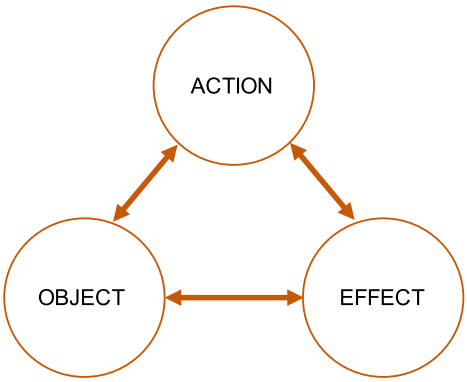


Figure 2.1: In the developmental robotics literature, an affordance represents the relation of an action, an object and an effect.

During the last decades different definitions of affordance have appeared. Initially, Gibson (1966) defined affordance as the direct perception of action possibil-

ities provided by objects to the agent. Jones (2003) explains that Gibson updated this definition during his lifetime. For instance, in his last work (Gibson, 1986) affordance is defined as not only being part of the environment, but it is related to the relation of an agent and its environment. Other ecological psychologists provide different interpretations of what an affordance is. For example, while Turvey (1992) and Stoffregen (2003) define affordances as *properties*, Chemero (2003) defines them as *relations* between particular aspects of the agent and particular aspects of the environment. These different interpretations have in common that *"an affordance manifests itself in relation to the action and perception capabilities of a particular actor"* (Jamone et al., 2016, page 4). Also, they are directly perceived by the agent.

Steedman (2002) makes the relation of a robot's action with its environment and the corresponding change, i.e. *effect*. In the same vein, from a robotics point of view Sahin et al. (2007) formalized affordances as the acquired relation of applying a behavior on an entity to obtain an effect:

$$(effect,\ (entity,\ behavior))$$

Montesano et al. (2008) provides a pragmatical definition, in which an affordance is the acquired relation of applying an action on an object to obtain an effect (see Figure 2.1). The Skill Builder presented in Chapter 5 and 6 gets inspiration from this formalization to create skills.

In this section predictive models either rely on affordance knowledge or are based on *deep learning* techniques. In the works learning predictive models, actions are usually considered as built-in knowledge, externally tailored by a designer. Therefore, the objective of a predictive model is to choose the right built-in action to perform, which is later executed in an open-loop. These works are only robust to spatial perturbations before the execution of an action, i.e. to the object position, not adapting the action to spatial and/or temporal perturbations during its execution. This offline spatial adaptation is usually externally hard-coded by the experiment designer. This low adaptation capability can result in the inability to scale up the executed experiments to realistic setups.

The works depicted in Tables 2.2 and 2.3 are categorized based on the classification available in Jamone et al. (2016). The relevant categories for the current manuscript are *Pioneering works* representing those first studies where the initial insights to learn the relation between objects and actions were identified; *Representing the effects* is the category with more related works, including A^2L, and extends the previous action-object relations to take into account the corresponding effect; *Multi-object interaction* represents affordances among several objects; and finally *Multi-step prediction* represents the use of affordances in high-level task planners to solve complex tasks.

The goal of most of the *pioneering works* (Krotkov, 1995; May et al., 2007; Metta and Fitzpatrick, 2003; Fitzpatrick and Metta, 2003) was the improvement of object perception through actions. More precisely, the identification of an affordance through the observation of the result performing actions on an object, e.g. *rollability*. Posterior works (Fitzpatrick et al., 2003; Stoytchev, 2005) made the first

Table 2.2: Comparison of action used within the affordance literature, where * represents ambiguous information, *DR* stands for *Discrete representation*, *IC* stands for *Inference capability*, *OffSP* stands for *Offline Spatial Perturbation*, *OnSP* stands for *Online Spatial Perturbation*, *TP* stands for *Temporal Perturbation*, *BA* stands for *Built-in actions*, and *RA* stands for *Repertoire of actions*. Regarding the learning methods *PI* stands for *Probabilistic Inference*, *DT* stands for *Decision Tree*, *BN* stands for *Bayesian Network*, *DRN* stands for *Relational Dependency Network*, *SVM* stands for *Support Vector Machine*, *NN* stands for *Neural Network*, *LWPR* stands for *Locally Weighted Projection Regression*, *DBN* stands for *Dynamic Bayesian Network*, *SVR* stands for *Support Vector Regression*, *LSTM* stands for *Long Short-term Memory*, *MMR* stands for *Maximum Margin Regression*, *GBN* stands for *Gaussian Bayesian Network*, and *DA* stands for *Denoisy autoencoder*.

Type	ID	Publication	Affordance learning method	DR	IC	OffSP	OnSP	TP	BA	RA
Pioneering works	16	Krotkov (1995)	-	-	-	No	No	No	Yes	Poke
	17	May et al. (2007)	-	-	-	No	No	No	No	Random
	18, 19	Metta and Fitzpatrick (2003), Fitzpatrick and Metta (2003)	-	-	-	Object position	No	No	Yes	Tap
	20	Fitzpatrick et al. (2003)	PI	No	Yes*	Object position	No	No	Yes	Tap
	21	Stoytchev (2005)	DT	No	No	Object position	No	No	No	Random
Representing the effects	22, 23	Demiris and Dearden (2005), Dearden and Demiris (2005)	BN	Yes	Yes	Object position	No	No	No	Random
	24	Hart et al. (2005)	DRN	Yes	No	Object position	No	No	Yes	Reach, Grasp
	25, 26, 27	Lopes et al. (2007), Montesano et al. (2008), Osório et al. (2010)	BN	Yes	Yes	Object position	No	No	Yes	Grasp, Tap, Touch
	28, 29	Ugur et al. (2009, 2011)	SVM	No	No	Object position	No	No	Yes	Push
	30	Ridge et al. (2010)	NN	No	No	No	No	No	Yes	Push
	31	Kopicki et al. (2011)	LWPR	No	Yes*	Object position	No	No	Yes	Push
	32, 33	Ugur et al. (2012, 2015b)	SVM	No	No	Object position	No	No	No	Grasp, Hit, Drop, Tap

Table 2.3: Continuation of Table 2.2

Type	ID	Publication	Affordance learning method	DR	IC	OffSP	OnSP	TP	BA	RA
	34	Mugan and Kuipers (2012)	DBN	Yes	Yes	Object position	No	No	Yes	Grasp
	35	Hermans et al. (2013)	SVR	No	Yes*	Object position and orientation	No	No	Yes	Push
	36 37	Finn et al. (2016), Finn and Levine (2017)	LSTM	No	No	Object position and orientation	No	No	No	Push
	38	Ebert et al. (2017)	LSTM	No	No	Object position and orientation	No	No	Yes, No	Lift, Push
	39	Hangl et al. (2016)	MMR	Yes	No	Object position and orientation	No	No	Yes	Push, Grasp, Flip
	40	Chavez-Garcia et al. (2017)	GBN	No	Yes	Object position	No	No	Yes	Push, Grasp
		The current work	**BN**	**Yes**	**Yes**	**Object position**	**Yes**	**Yes**	**No**	**Push, Grasp, Press**
Multi-object interaction	41 42	Jain and Inamura (2011) Jain and Inamura (2013)	BN	Yes	Yes	Object position	No	No	Yes	Push, Pull
	43, 44	Goncalves et al. (2014a,b)	BN	Yes	Yes	Object position	No	No	Yes	Tap, Push, Pull
	45	Szedmak et al. (2014)	SVM	No	No	Object position	No	No	Yes	Stack, Poke
	46, 47	Dehban et al. (2016, 2017)	DA	No	Yes	Object position	No	No	Yes	Push, Pull
Multi-step predictions	48 49	Omrčen et al. (2008), Krüger et al. (2011)	NN	Yes	No	Object position and orientation	No	No	Yes	Poke, Push, Grasp
	50, 51	Ugur and Piater (2015a,b)	SVM	Yes	Yes	Object position	No	No	Yes	Pick, Release, Poke
	52 53	Lang and Toussaint (2010) Antunes et al. (2016)	BN	Yes	Yes	Object position	No	No	Yes	Grasp, Release, Pull

attempts to learn the relation between the action and the obtained result, trying to choose the best action to reproduce it. These works identify simple changes on objects, and thus the complexity of the performed actions is low, i.e. from random actions to built-in *tap* or *poke* actions modifying the contact angle.

In contrast, the works *representing the effects* focus on the learning of an inverse model to reproduce a previously observed effect on an object. Dearden and Demiris (2005) is the first work to propose representing the forward model using Bayesian Networks (BN) in this context. The BN infers the expected visual perception of opening and closing a gripper. Afterwards, the same authors (Demiris and Dearden, 2005) transform the learned BN into an inverse model, inferring the motor commands to play imitation games. Because of the simplicity of the effects, these works use random motor commands as actions. Hart et al. (2005) uses a classical approach for built-in actions. The action is split up into an *approach* phase where the robot's end-effector moves to a predefined position with regard to the object, and a final phase where the effector actually performs the action, i.e. in this case *grasp*. This work also uses a probabilistic approach to represent the inverse model. Inspired by the previous works, Lopes et al. (2007); Montesano et al. (2008) define an affordance as a BN representing the relation between *action, object* and *effect* (see Figure 2.1). They provide built-in *grasp, tap,* and *touch* actions to also play imitation games. In these works, the execution of the actions depends on some free parameters, as the height of the robot's end-effector related to the object. Osório et al. (2010) extended these works improving the robustness to noisy environments, representing the perceptual information as GMM. Ugur et al. (2009, 2011) define a very complete framework for imitation games. They define a set of experiments, as cleaning a table or move an object to a specific position, where a robotic arm executes a sequence of built-in *push* and *grasp* actions. Ridge et al. (2010) is another example of affordances learning performed by a robotic arm. However, this is one of the few works using Neural Networks (NN) to represent the affordance knowledge. The objective of Kopicki et al. (2011) is to learn to predict effects on a set of different objects using regression techniques in a probabilistic framework using a built-in *push* action. Mugan and Kuipers (2012) is a *end-to-end* work endowing a robot to learn to autonomously perform tasks from continuous perception. Although Mugan does not specifically mention affordance learning, he defines a Dynamical Bayesian Network (DBN) to represent an inverse model to choose the right action to reproduce an effect. Notice that a DBN infers actions taking *time* into account. The proposed methodology is evaluated by a simulated robot grasping an object in a tabletop setup using a built-in *grasp* action. In Hermans et al. (2013) a PR2 robot uses a built-in *push* action on few objects to either displace them in straight line or to rotate them. In Hangl et al. (2016) a robot uses a built-in *push* action to rotate a book. Besides, the book is lifted using a *grasp* action. Also, a *flip* action is performed using both of the robot's end-effectors to open a small box. Chavez-Garcia et al. (2016) learns the affordances of composite objects using the affordance knowledge of the elementary objects. The authors use continuous values of the random variables relying on Gaussian Inference Diagrams(Shachter

and Kenley, 1989).

Multi-object interactions has gathered many research attention during the last years, mainly focused on the use of tools to reproduce effects on objects. Jain and Inamura (2011, 2013) use a BN to model affordances to *push* and *pull* objects using tools with different features. Similarly, Goncalves et al. (2014a,b) use this approach to extend the work performed by Montesano et al. (2008) to be applied using tool. The approach is validated on *tapping*, *pushing* and *pulling* objects. A different and promising approach is used in Dehban et al. (2016, 2017). These works use Denoising Autoencoders (Vincent et al., 2010) to model tool affordances using continuous values in order to *push* and *pull* different objects. Conversely to tool use, Szedmak et al. (2014) proposes to model the interactions of 83 objects with different features. A robotic arm is assisted by a human expert to build a dataset of interactions *poking* and *stacking* pairs of objects.

In order to perform complex tasks, affordance knowledge must be used to predict a sequence of actions. Although task planning is not directly related to A^2L it is relevant to mention few architectures that could use the skills built using our method (see Section 2.2).

In the previous works a repertoire of built-in actions was available for the affordance learning. Nevertheless, a couple of works by Ugur built this repertoire beforehand (Ugur et al., 2012, 2015b). And thus they are more suitable for learning in realistic environments. In these works a built-in *generic swipe action* is available, which executes a trajectory of a robot's end-effector from a fixed initial position to the position of a close object. Therefore, for different object positions different trajectories are built. Nevertheless, the shape of these trajectories does not differ much among them, because of the use of the same heuristic to generate them. The behavior of each instance of the swipe action is determined by five parameters: the initial, middle and final displacement regarding the center position of the object to be reached by the effector, the moment at which the hand is closed (it is assumed to be open), and the moment at which the hand reopens again. Whereas the displacement values are obligatory, the open/close values are optional. The duration of the swipe action is fixed to five seconds. The variation of the values of these parameters allow the robot to produce different actions on an object. The changes of object features produced by those actions are clustered using the X-Means algorithm (Pelleg and Moore, 2000) identifying a repertoire of skills generating the *push, no touch, release and grasp* actions. Other works in the same vein are Finn et al. (2016); Finn and Levine (2017); Ebert et al. (2017), which use a deep learning technique called convolutional LSTM (Hochreiter and Schmidhuber, 1997) in order to predict the visual output of an action. Finn builds a repertoire of continuous *push* actions based on an exploration performing thousands of interactions of a robotic arm with a set of objects. Ebert improves the results obtained by Finn adding a discrete *lift* action to move the end-effector away from the objects during the exploration (see Wong (2016) for a recent survey about applying deep learning techniques in robotics).

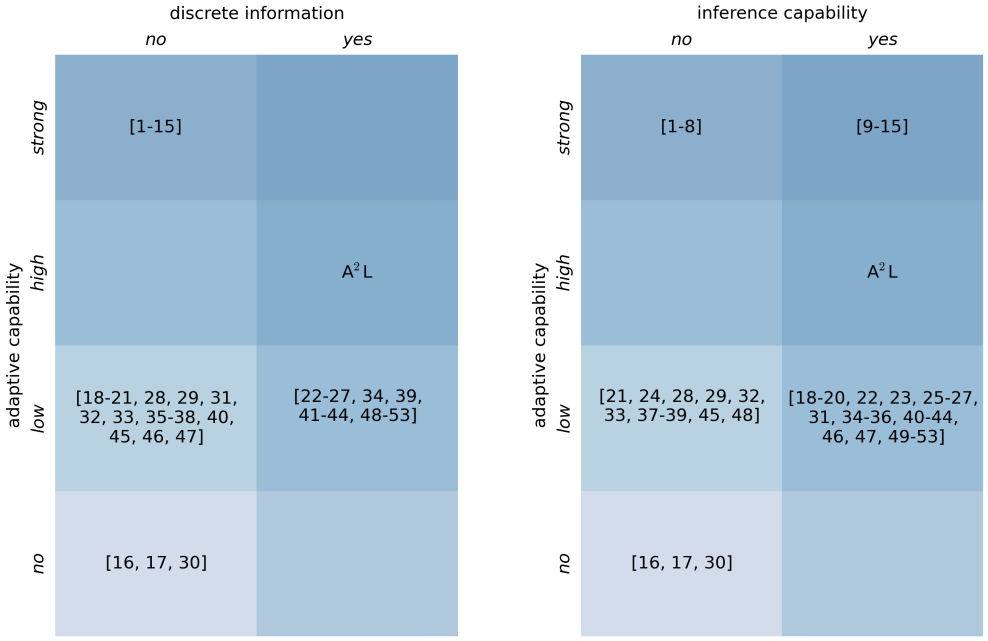


Figure 2.2: Comparison of works in the developmental robotics literature addressing the features proposed at the end of Section 1.1. The numbers represent works available in Tables 2.1, 2.2 and 2.3. On the left, comparison of the adaptation to the environment and the use of discrete representations. On the right, comparison of the adaptation to the environment and the inference capability. The *adaptation* values are as follows: *no* means the skills do not have any adaptation capabilities, *low* means the skills can only adapt to the object position before the execution of the action, *high* means the skills can adapt to both spatial and temporal perturbations before and during the execution of the action, and *strong* extends the *high* value being stable, i.e. always reproducing the effect.

2.2 Performing a Task

Mugan suggests that *"there are two broad planning frameworks within AI: STRIPS-based goal regression, and Markov Decision Process (MDP) planning."* (Mugan, 2011, page 7).

PDDL, Planning Domain Definition Language, (McDermott et al., 1998) is a common planner used in the literature using a STRIPS-like notation (Nilsson, 1981) (further details are available in Section 3.4). Some works using PDDL are Ugur and Piater (2015a,b), which propose building planning rules based on the affordance knowledge of a set of object and effect categories identified during the exploration of an environment. The exploration is performed using the built-in *grasp*, *release*, and *poke* actions. Also Konidaris et al. (2014, 2015) use a probabilistic PDDL planner to play computer games, called PPDDL. A recent paper extends these works using

PDDL to drive the actions of a physical robot performing a task consisting in moving a bottle from a cooler to a cupboard (Konidaris et al., 2018).

On the other hand, MDP planning (Puterman, 1990) is directly related to Reinforcement learning (Sutton and Barto, 1998), and thus it is widely used. Mugan (2011); Mugan and Kuipers (2012) learn predictive models as Dynamic BN, which are transformed into a MDP for task planning.

Another planner called PRADA (Lang and Toussaint, 2010) is used by Antunes et al. (2016) to connect human command with actions. In this work, an iCub robot prepares a hamburger, as a stack game, approaching with a tool different objects when needed.

2.3 Conclusions and Open Questions

As depicted in Figure 2.2, the skills generated by A^{2}L are the only skills that simultaneously adapt to perturbation in an online fashion, handle high-level and low-level information, and are able to infer actions based on uncertain information.

On the one hand, regarding the online adaptation to perturbations, the action generator of A^{2}L is time-independent and relies on low-level states and movements to infer motion. Similarly to the state-based works, in our method low-level states are used to determine the robot motion. However, whereas in our method the low-level states are common to different robot-object interactions, as *push* and *grasp*, in these works the states are only useful to reproduce some specific demonstrations from exactly the same initial positions.

On the other hand, the skills generated by A^{2}L can handle discrete representations and have a strong inference capability. And thus actions can be inferred based on movements, low-level information, e.g. object position, and high-level information, e.g. object color. And even abstract information, e.g. danger. However, handling discrete information comes at a cost: A^{2}L cannot ensure stable actions.

Chapter 3

Background

Contents

This chapter makes an introduction to the methods A^2L relies on:

- In Chapter 4, an environment exploration is executed by a robot driven by our method called NovEB. This method is based on Novelty Search, which relies on evolutionary algorithms driven by an intrinsic motivation based on a behavior novelty criterion. The concept of intrinsic motivation has been already addressed in 2.1.1. Therefore, we directly introduce the concept of evolutionary robotics and evolutionary algorithms, followed by an explanation of Novelty Search.

- The action generator of A^2L, used in Chapters 5 and 6, is implemented as a Bayesian Network (BN). BN outputs, adapted to low-level states, are translated to motor commands through a Dynamical System (DS), and to high-level states using predictive models based on affordance knowledge (already explained in Section 2.1.3). We first provide an introduction to the BN principle, and next to other BN features used by our method, e.g. d-separation (Pearl, 1986). Then, the main concepts of DS are presented. Later, an explanation of the Behavioral Dynamics approach (Warren, 2006) is available. Finally, we present the DS on which A^2L relies.

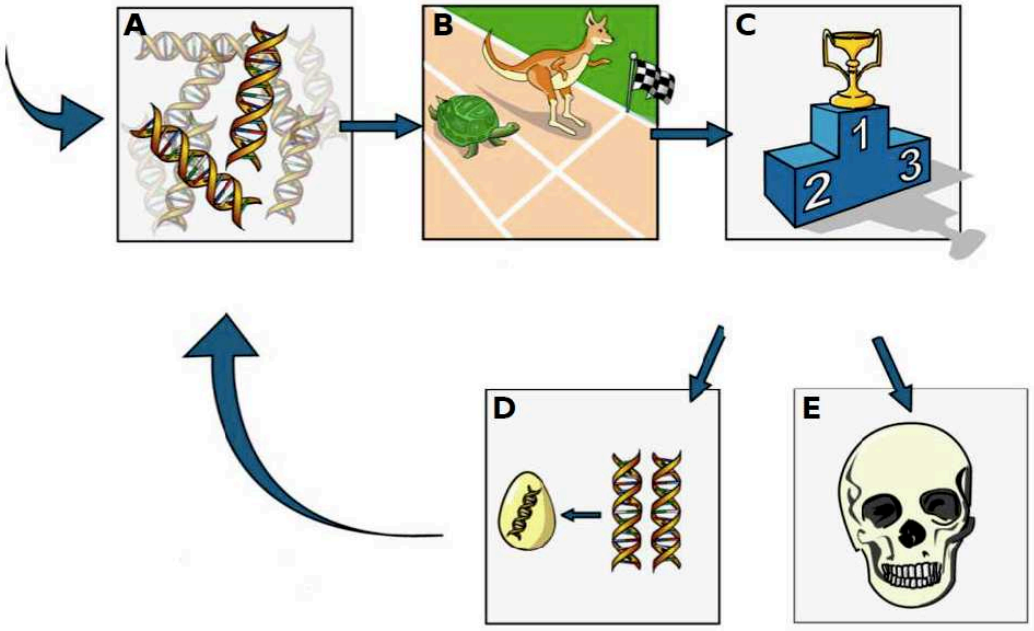


Figure 3.1: Steps of an evolutionary algorithm. (A) Generate (randomly) an initial population. (B) Calculate fitness value of each individual. (C) Select individuals based on this value. (E) The rest are discarded. (D) Apply genetic operators (*crossover* and *mutation*) to the selected individuals in order to generate the next population. From Cully and Mouret (2015).

- The system implemented for the validation experiments uses A^2L to learn and execute skills, and a higher-level planning method to identify the sequence of actions to perform. The corresponding state-of-the-art planning methods are then introduced at the end of this chapter.

3.1 Evolutionary Algorithms

3.1.1 Principle

Evolutionary algorithms (EA) rely on the variation and selection principles of natural selection in order to drive a search and optimization process (Eiben and Smith, 2008). Namely, evolutionary algorithms are an abstraction of the processes and principles established by Darwinism (Darwin, 1872). They perform a *black box* optimization process just driven by a cost function called *fitness function* by reference to biology. It is a blind search process that is robust to noisy and multi-modal fitness functions and versatile with respect to what is optimized (bit strings, vectors of float, graphs, trees, etc). A key feature of these algorithms comes from the fact that it is a robust technique, which can you deal with a great variety of problems coming from different areas, including those in which other methods encounter difficulties. While it is not guaranteed that the evolutionary algorithms find the optimal solu-

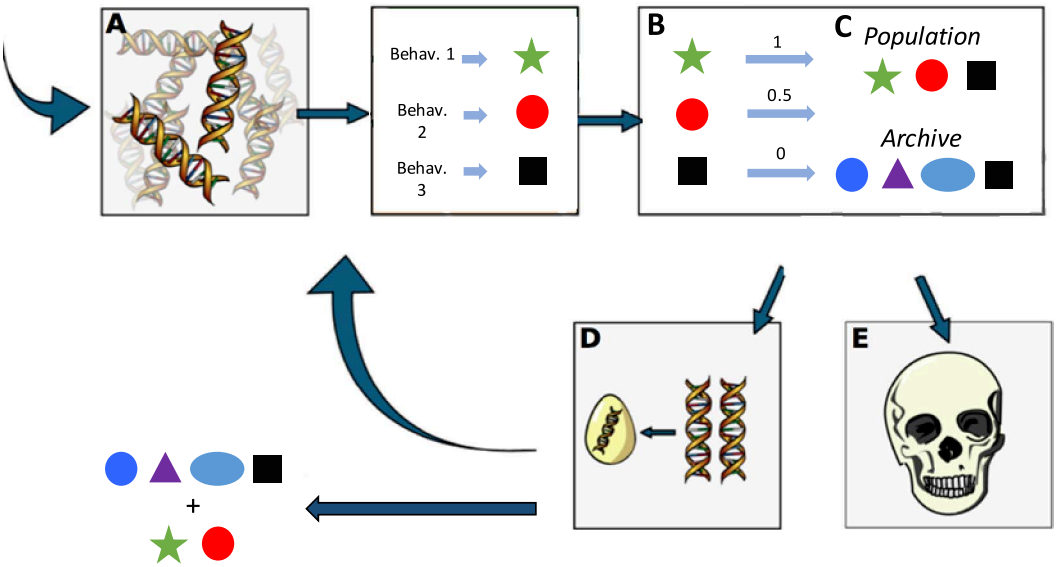


Figure 3.2: Steps of the Novelty Search algorithm. (A) Generate (randomly) an initial population. (B) Calculate fitness value of each individual, based on their *novelty* w.r.t. both the other individuals of the same population, and those within the *archive*. (C) Select individuals based on this value. (E) The rest are discarded. (D) Apply genetic operators (*crossover* and *mutation*) to the selected individuals in order to generate the next population. Besides, the individuals with highest novelty are stored into the archive for the computation of the *novelty* in the next iterations of the algorithm. In this example, the population is composed of three behaviors, represented as a green start, a red circle and a black square. The star gets the higher *novelty* because there are no other stars neither on the population nor on the archive. Conversely, the *novelty* of the other elements is lower due to the fact that there is already a circle, although of a different color, and another black square.

tion to the problem, there is empirical evidence that solutions of an acceptable level are found.

They are population-based algorithms in which many candidate solutions, called *individuals*, are considered in parallel (see Figure 3.1). Starting from a *random population*, an iterative process evaluates each individual using the fitness function, which describes what is expected from individuals, to be maximized by the evolutionary process. The process selects some of the individuals according to their fitness value and generates new solutions thanks to blind search operators, i.e. *mutation*, that makes small random modifications to an individual, or *crossover* that mixes several individuals. An individual contains a *genotype* and the corresponding fitness value. The *genotype* corresponds to the structure to be designed or optimized. Typical genotypes are vector of floating point values, strings of binary digits, trees or neural networks.

3.1.2 Novelty Search

Evolutionary robotics relies on evolutionary algorithms to generate robot controllers or morphology (Doncieux et al., 2015a). They have been used to generate controllers for locomotion, navigation or foraging tasks (Nelson et al., 2009). They have the specificity of not requiring the definition of a discrete set of actions and they can explore large sets of continuous variables, provided that they can make enough solution evaluations. These features are very interesting w.r.t the task-agnostic exploration of an environment because they can be exploited to generate actions of a high diversity avoiding premature convergence. It was recently shown that using task-independent behavior-based criteria to drive an exploration had a very significant impact on the generated results (Doncieux and Mouret, 2014).

Novelty Search is an evolutionary algorithm to search for novel behaviors (Lehman and Stanley, 2011). It is a task-independent method, driven by the seek of novelty (see Figure 3.2). The main features with respect to the previous algorithm are: (i) the use of a specific fitness function to compute the *novelty*, (ii) the comparison of individuals of different generations, stored in an *archive*. The *novelty* of an individual, i.e. a *behavior*, is defined as the average behavioral distance between this behavior and its k-nearest neighbors in the current population and in an archive of previously explored behaviors:

$$Novelty(i, p, a) = \frac{1}{k} \sum_{j=0}^{k} dist(i, neigh(i, p, a)_j) \tag{3.1}$$

where $neigh(i, p, a)_j$ is the jth-nearest neighbor of individual i, including the current *population*, p and the *archive*, a, with respect to the distance, *dist*, representing the distance between the corresponding behaviors. $Novelty(i, p, a)$ is then used as a fitness function in the evolutionary process. The method strongly relies on the *behavioral distance* used to compute novelty. This distance is typically defined in a space of behavior descriptors and is problem-specific. A related example is Cully and Mouret (2015), in which a hexapod robot quickly and autonomously learns to walk in any possible direction in its vicinity, using novelty to modify the robot's controllers.

3.2 Bayesian Networks

3.2.1 Principle

Bayesian Networks (BN) are a graphical representation of dependencies for probabilistic reasoning, in which the nodes represent *random variables* and the lack of arcs represent *conditional independence relationships* between the variables (Pearl, 1988). More precisely, a BN is a directed acyclic graph (DAG), i.e. a collection of nodes or vertices joined by directed edges without directed cycles. The topology or structure of the network provides information about the probabilistic dependencies between the variables as Conditional Probabilistic Distributions (CPDs). CPDs are

represented as tables, which grow exponentially with the number of parents of a node. This is the corresponding computational cost or complexity that is called *curse of dimensionality* (Bellman, 1966). Figure 3.3 shows both the topology and CPD of an example computing the probability of a grass being wet. The grass will be wet, W, if it rained that day, R, or if the sprinkler, S, run (below table). The sprinkler runs half of the times if it was not cloudy, C. Conversely, it rains almost each time it is cloudy. In this section upper case letters represent random variables and lower case letters represent possible values of the random variables. $P(X)$ represents the probability distribution of X, whereas $P(X=x)$ represents the probability that the value of X is x.

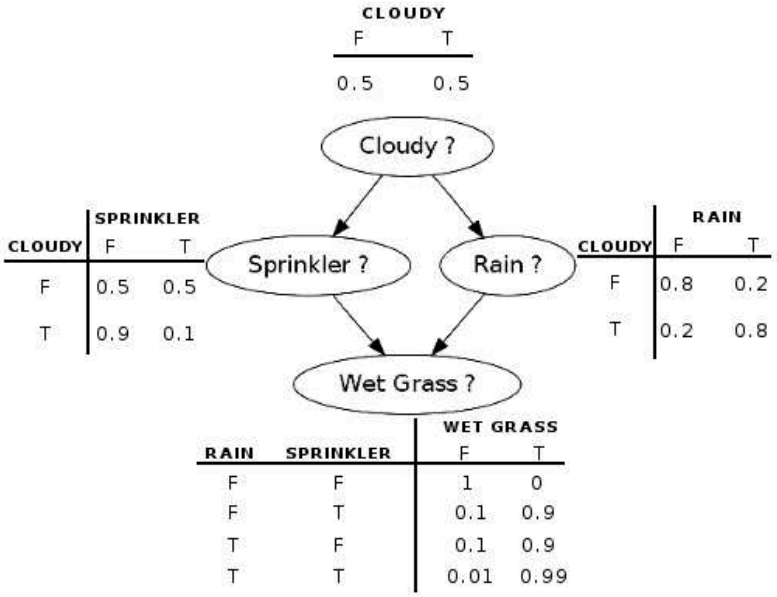


Figure 3.3: Example of a Bayesian Network and the related Conditional Probabilistic Distributions (from the aGrUM documentation).

If there is an arc linking a node X to another node Y, X is called a *father* of Y, and Y is called a *child* of X. The *parents* of a node X are all the fathers of the node. If node X has no parents, its local probability distribution is taken as unconditional, otherwise it is conditional. If the value of a node is observable - and therefore labeled as observed, that node is an *evidence* node.

In a BN the joint probability is specified by the product of the probabilities of each variable, X_i, given their parents, $Pa(X_i)$:

$$P(X_1, X_2, X_3, ..., X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i)) \tag{3.2}$$

In the previous example:

$$P(C, S, R, W) = P(C)P(S|C)P(R|C)P(W|S, R)$$

The conditional probability, or posterior probability, is the probability that $X{=}x$ given that $Y{=}y$:

$$P(x|y) = \frac{P(x \wedge y)}{P(y)} \tag{3.3}$$

And then the *Bayes theorem* can be used to compute $P(y|x)$ as:

$$P(x \wedge y) = P(x|y)P(y) = P(y|x)P(x) \tag{3.4}$$

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{3.5}$$

3.2.2 Inference Capability

The probabilistic reasoning or propagation of probabilities consists in propagating the effects of the evidence through the network to know the a posteriori probability of the variables. Namely, the values of certain variables are observed, called *evidence*, and the posterior probability of the other variables is obtained given the known variables. For example, in the wet grass example it is possible to compute the posterior probability of the sprinkler switched on if the grass is already wet, e.g. it is already raining:

$$\begin{aligned} P(S=1|W=1) &= \frac{P(W=1|S=1)P(S=1)}{P(W=1)} \\ &= \frac{P(W=1|S=1)P(S=1)}{P(W=1,S=1)+P(W=1,S=0)} \end{aligned}$$

where $P(W=1,S=1)$ and $P(W=1,S=0)$ can be computed using the joint probability.

In order to facilitate the inference computation, the DAG of a BN is transformed to another structure. In the current manuscript, the inference capability of a BN relies on a junction tree based inference called *lazy propagation* (for a detailed explanation see Madsen and Jensen (1999)). This approach allows inferring simultaneously more than one variable, if needed.

3.2.3 d-separation

Another relevant feature for A^{2}L is the concept of *d-separation* (Geiger et al., 1990), which represents the independence model of a BN. Being P a path between two nodes X and Y, i.e. a set of arcs linking X and Y, without taking into account their directions. P is said to be d-separated by a set of Z nodes if and only if (at least) one of the following conditions is true:

- P contains a directed chain, $X \ldots \leftarrow M \leftarrow \ldots Y$ or $X \ldots \rightarrow M \rightarrow \ldots Y$, in which the middle node M belongs to Z,
- P contains a divergence of M, $X \ldots \leftarrow M \rightarrow \ldots Y$, in which the node M belongs to Z,

- P contains a convergence to M, $X \ldots \rightarrow M \leftarrow \ldots Y$, in which neither the middle node M nor its descendants belong to Z.

Thus, X and Y are said to be d-separated by Z if all the paths between them are d-separated. More precisely, two nodes X and Y are d-separated given Z whenever they are conditionally independent given Z. If X and Y are not d-separated, they are called *d-connected*.

3.2.4 Structure Learning

In order to learn the probability distribution among the random variables, a BN must have a well-defined structure. This structure can be learned (as in Chapter 5) or it can be externally defined (as in Chapter 6). Structure learning consists in finding the structure best fitting a given dataset. Structure learning is a NP-hard problem (Chickering et al., 1994). There are three main approaches for structure learning: (i) *constraint-based approach*, which first uses statistical independence tests to identify a set of arc constraints for the graph and then finds the best DAG that satisfies the constraints (Pearl and Verma, 1995; Spirtes et al., 2003), (ii) *score-based searching approach*, which searches over the space of graphical structures for a structure with maximal score (Heckerman, 1996; Chickering, 2002; Buntine, 1991), and (iii) *hybrid approach*, which computes an initial DAG using the constraint-based approach, and refines it using a score-based searching approach (Tsamardinos et al., 2006; van Dijk et al., 2003). A^2L relies on a score-based searching approach for the structure learning.

There are two types of scoring functions: (i) Bayesian scoring functions, which compute the posterior probability distribution, starting from a prior probability distribution on the possible networks given a dataset D and on the possible parameters (CPD), and (ii) Information-theoretic scoring functions, which are based on the compression that can be achieved over D with an optimal code induced by the BN. In the current manuscript, two closely related information-theoretic scoring functions are used, penalizing a log-likelihood score, LL.

- *Akaike Information Criterion (AIC)* (Akaike, 1974). It is mainly used when there is a large number of models to evaluate. AIC provides a measure of the relative quality of the model. Its formula is:

$$AIC(B|T) = LL(B|D) - k$$

  where B represents a BN graphical structure, D represents the given data and k is the sum of the number of free parameters of B. AIC measures the fit with the likelihood and at the same time penalizes the use of many parameters.

- *Bayesian Information Criterion (BIC)* (Schwarz, 1978). This scoring criterion is very similar to AIC:

$$BIC(B|T) = LL(B|T) - \frac{1}{2}\, log(N) * k$$

where N represents the total number of instances in the data T. It is based on maximum likelihood as fitting measure, as AIC. The measure of complexity introduces both k and *log(N)*, penalizing more the inclusion of many variables than AIC does.

In the current manuscript two structure learning methods are used, with different *a priori* knowledge about the structure:

- Hill-climbing (Chickering et al., 1995). The method uses an iterative improvement technique. There is no initial information to drive the learning process. It starts with a BN structure, with or without arcs. At each step, it attempts to change the graph structure by a single operation of adding an arc, removing an arc or reversing an arc, preserving the acyclic property. If there are changes that increase the score, it is selected the change maximizing the score. Otherwise it makes another attempt. The method ends when there are no improvements, or when a preset number of iterations is reached.

- K2 (Cooper and Herskovits, 1992). A visual demonstration of its functioning is available in Ruiz (2005). It is one of the fastest methods for structure learning in BN. A topological order of the graph is needed. It starts with a structure, possibly empty. For each random variable, the method searches among its parent set the parent that most increases the score. The method stops when no improvement can be made.

3.3 Dynamical Systems

3.3.1 Principle

This section makes a brief description of the main features of dynamical systems (DS) relevant for A^2L. For a complete explanation see Katok and Hasselblatt (1995) and Siciliano and Khatib (2008). Dynamical systems are systems whose internal parameters, i.e. state variables, follow a series of temporary rules. They are called *systems* because they are described by a set of equations, and *dynamical* because their parameters vary with respect to some variable, usually time. A DS is autonomous if it is represented by an autonomous ordinary or unforced differential equation of the form:

$$\dot{x} = F(x) \tag{3.6}$$

whereas a non-autonomous DS has the form:

$$\dot{x} = F(x, t) \tag{3.7}$$

The difference between these systems lies in the fact that Equation 3.6 does not contain any external stimulus to the system dependent on the system that forces the natural behavior of the dynamics of the system, while 3.7 does.

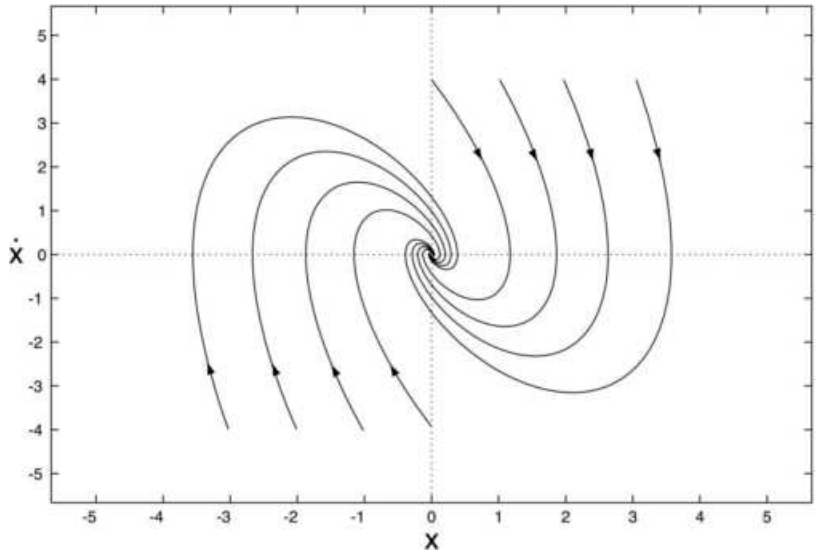


Figure 3.4: Example of an attractor from Warren (2006). The black point represents the position of an attractor, and the black arrows show the attraction directions to the point.

A system is *time-independent* if it does not depend explicitly on time. From the definition it can be concluded that every autonomous system is invariant in time. In general, a DS is invariant in time if:

$$x(0) = x(\delta) = x_0 \rightarrow x(t) = x(t+\delta) \forall t \tag{3.8}$$

Namely, for the system to be time-independent two trajectories passing through the same point in different times will have the same evolution. If Equation 3.8 does not comply, the DS is *time-dependent*.

The way of visualizing the behavior of the state variables of a dynamic system can be in the form of a time series (graph of a state variable versus time), or in the form of a phase space, used in the current manuscript. The phase space of an n-dimensional system as in Equation 3.6 is the space where all the possible states of a system are represented. Each system parameter is represented as an axis of a multidimensional space and each point of the space represents each possible status of system variables. In this type of representation, time becomes an implicit parameter (see Figure 3.4).

The phase space is described by a *vector field, F,* which governs the path of the system variables *x(t)* in time, called *path*. It is said that a singularity of the phase space is an *attractor* if every trajectory that starts close to it approaches it as time passes. On the other hand, a singularity of the phase space is *Lyapunov-stable* if all trajectories that start sufficiently close to it remain close to it during all time. The situation may arise that a singularity of the phase space is Lyapunov-stable but it is not an attractor. If this happens it is said to be *neutrally stable*. However, when the two types of stability usually occur at the same time it is called *asymptotically*

stable. Finally, a singularity is a repulsor when it is neither attractor nor Lyapunov-stable. That is, the trajectories that start close to it diverge as time passes. The importance of the stability of singularities lies in the fact that this determines the stability of the system in which the singularities are presented.

3.3.2 Behavioral Dynamics

During the 70's and the 80's different psychologists proposed theories to describe the interaction of an agent with its environment as DS. First, human and animal behavior was formalized by Kugler et al. (1980) as low-dimensional DS. More precisely, stable behaviors were described as *attractors*, behavior states to be avoided correspond to *repellers*, and changes in number or type of attractors and repellers are described as *bifurcations*.

Later, the *perception-action cycle* was introduced by Kugler and Turvey (1987) and Warren (1988). In these works, the interaction of an agent with its environment is described as a goal-oriented continuous loop, where the perceptual information acquired by the agent drives the generation and execution of an action, possibly interacting with the environment and modifying the perception of the agent.

Warren (2006) defines a theoretical framework to apply the previous theories in realistic use cases. To that end, he extends and complements the existing works as follows: first, both perception and action are ruled by different laws. On the one hand, the perceptual information follows the ecological laws defined by Gibson (1966), e.g. the *law of visual perception* (Gibson (1979)). On the other hand, Warren (1988) describes the *laws of control* by which to visually regulate the actions. Namely, the definition of a set of free parameters to tailor an action to the environment features. Second, Warren describes two levels of analysis for any interaction (Figure 3.5). The perception-action cycle, at the first level of analysis, represents the global behavior of the interaction. A lower level of analysis, i.e. second level, represents a low-dimensional description of the global behavior. This level describes the temporal evolution of a behavior, its *behavioral dynamics*. Warren claims the current state of a behavior, regarding a specific goal, is described based on the change of few variables, the *behavioral variables*. Therefore, at this level, observed trajectories are described based on these variables, and can be formalized as DS.

Both levels of analysis are tightly coupled. In a *bottom-up* approach, the behavioral dynamics emerge from the regularities identified during the interaction of the agent with its environment. Therefore, adapting a behavior to an environment involves both levels of analysis. The first level, gathering the environmental information, and the second level adapting the behavior to this information. In a *top-down* approach goals are defined as attractors, and therefore the low-level vector field affects the global success of the interaction. Warren claims that the correlation between both levels satisfies the features of emergent behavior and self-organization proposed by Bar-Yam (2004) and Haken (1978).

The previous concepts are formalized as DS. At the first level of analysis, on the one hand, an *ecological law* Φ (Equation 3.9) represents the change over time of

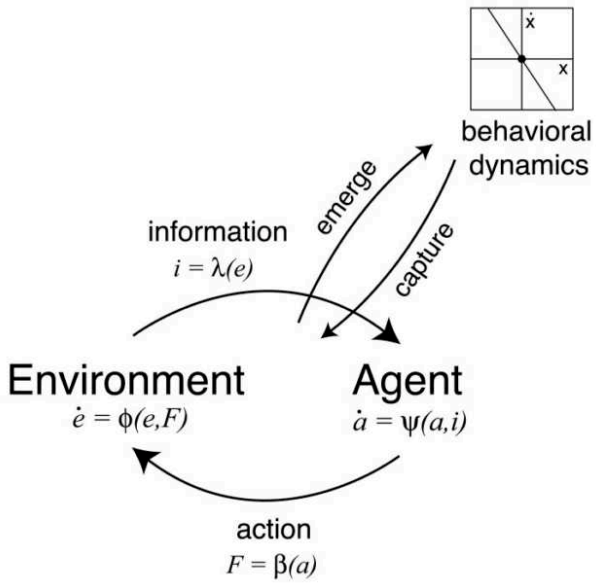


Figure 3.5: Description of the two levels of analysis of a robot interaction with its environment. From Warren (2006).

the environment state, e, after applying some forces, c, on it. Notice that forces are denoted as c instead of F, as in Figure 3.5, to avoid a nomenclature conflict with the formalization available in Chapters 5 and 6. On the other hand, a *law of control* Ψ (Equation 3.10) represents the change over time of the agent state, a, based on some information acquired about the environment, i.

$$\dot{e} = \Phi(e, c) \tag{3.9}$$

$$\dot{a} = \Psi(a, i) \tag{3.10}$$

Changes on both *ecological and control laws* rely on external information. The aforementioned forces, c, modifying the environment are the result of the execution of an action (Equation 3.11), called the *effector function*, β. From a robotics perspective this function corresponds to the computation of an action, based on the robot's inverse kinematic model, and its posterior execution.

$$c = \beta(a) \tag{3.11}$$

Similarly, an *information function* λ (Equation 3.12) transforms the state of the environment into the behavioral variables, adapting an action to the robot's environment.

$$i = \lambda(e) \tag{3.12}$$

At the second level of analysis, the DS corresponds to the change of value of

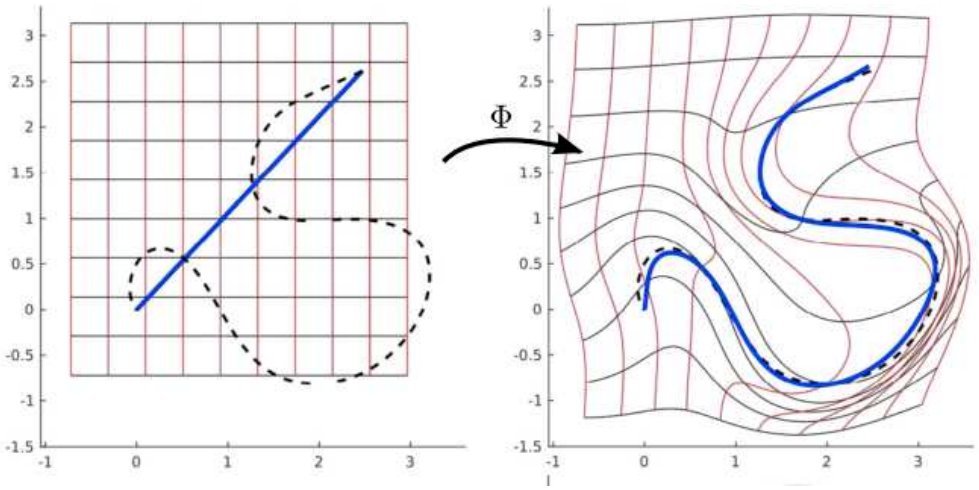


Figure 3.6: Example of a diffeomorphic matching, Φ, of a simple trajectory (blue line) to a demonstrated trajectory (dotted line). The space of the vector field is represented by the red grid. From Perrin and Schlehuber-Caissier (2016).

the behavioral variables, h, over time. Equation 3.13 represents a DS Ω where some behavioral systems change over time based on some parameters of the available velocity field, v.

$$\dot{h} = \Omega(h, v) \tag{3.13}$$

In summary, the relation of a robot with its environment is described using DS. This approach is utilized in Section 6.3 to endow a robot with the capacity to dynamically adapt to the position of an object while executing an action to reproduce an effect.

3.3.3 Building a Dynamical System in a Deformed Space

Perrin and Schlehuber-Caissier (2016) present a method to build a DS, i.e. a vector field, given the trajectory of a demonstrated action. A robot can use the vector field to reproduce the demonstrated action using one of its end-effectors. The method proposes applying a deformation to the motion space in order to fit a simple trajectory like $\hat{x} = -x$ to the demonstrated trajectory. More precisely, the authors aim to minimize a defined distance between both trajectories using a *diffeomorphic matching* algorithm (see Figure 3.6).

The proposed approach is complementary to the works presented in Section 2.1.2. The authors stress that the deformed space could improve the design of building asymptotically stable DS by methods like SEDS (Khansari-Zadeh and Billard, 2011, 2014) because the number of Lyapunov candidates is higher in this space. In Chapter 6, A^2L relies on this diffeomorphic matching to build vector fields of the demonstrated trajectories performed by an external agent.

3.4 Task planning

Mugan suggests that *"there are two broad planning frameworks within AI: STRIPS-based goal regression, and Markov Decision Process (MDP) planning."* (Mugan, 2011, page 7). On the one hand, goal regression techniques generate a sequence of actions to execute, given a task goal and information about the environment. On the other hand, MDP-based planning is related to Reinforcement learning (Schaal, 1999), using rewards to select the next action to perform. In the current manuscript, the skills performed by A^2L are validated through the performance of a task. We are thus interested on using for the validation a method that selects the next action to perform based on the changes produced on the environment. Therefore, we have selected a goal regression method called PDDL (Planning Domain Definition Language) (McDermott et al., 1998), because it is used on other works performing task planning within the developmental robotics literature, e.g. Konidaris et al. (2014) and Ugur and Piater (2015a).

PDDL is an action-centered language, inspired by the well-known STRIPS formulations of planning problems (Nilsson, 1981). It uses pre- and post-conditions to describe the applicability and effects of actions. It is partially inspired by the programming language called Lisp. PDDL has been recently extended[1], for instance to handle probabilities using MPDs, i.e. PPDDL, as in Konidaris et al. (2014). The components of PDDL are split up into two blocks, the *domain* and the *problem*. The domain defines some components available to solve different problems, whereas the problem defines a specific task. The components of the domain are:

Domain of the task It contains the predicates and the actions.

$$
\begin{aligned}
&(define\ (domain\ <domain\ name>)\\
&\quad <PDDL\ code\ for\ predicates>\\
&\quad <PDDL\ code\ for\ first\ action>\\
&\quad [...]\\
&\quad <PDDL\ code\ for\ last\ action>\\
&)
\end{aligned}
$$

Predicates Object properties defined as booleans.

$$
\begin{aligned}
&(:predicates\ (room\ ?x)\\
&\qquad\qquad (robot\ ?r)\\
&\qquad\qquad (robot-at\ ?x\ ?r)
\end{aligned}
$$

where *room* is true if and only if x is a room, where *robot* is true if and only if x is a robot, and *robot-at* is true if and only if x is a room, if r is a robot, and the robot is in the room.

[1] https://en.wikipedia.org/wiki/Planning_Domain_Definition_Language

Actions They change the task state.

$$\begin{aligned}(&: action\ move\ : parameters\ (?x\ ?y\ ?r)\\ &: precondition\ (and\ (room\ ?x)\ (room\ ?y)\ (robot\ ?r)\\ &\qquad (robot-at\ ?x\ ?r))\\ &: effect\ (and\ (robot-at\ ?y\ ?r)\\ &\qquad (not\ (robot-at\ ?x\ ?r))))\end{aligned}$$

where the precondition of the action *move* consists in having 2 rooms, x and y, a robot, r, and the robot is in room x; and the effect is that the robot moved from room x to room y.

The components of the task are:

Task It defines the task goal, initial state and objects for a specific domain.

$$\begin{aligned}(define\ problem\ &< problem\ name >)\\ (: domain\ &< domain\ name >)\\ &< goal\ specification >\\ &< initial\ state >\\ &< objects >\end{aligned}$$

Task goal It defines the goal to reach.

$$(: goal\ (robot-at\ kitchen\ roomba))$$

Initial state It defines the initial values for the task objects.

$$\begin{aligned}(: init\ &room\ kitchen\\ &room\ dinner\_room\\ &room\ living\_room\\ &robot\ roomba)\end{aligned}$$

Objects Entities of interest.

$$\begin{aligned}(: objects\ &kitchen\ dinner\_room\ living\_room\\ &roomba\ baxter)\end{aligned}$$

A *domain* can be used in many *problems*. And thus it should contain a wide range of actions and objects.

Chapter 4

Bootstrapping Interactions with the Environment

The results and text of this chapter have been published in the following article.

- **Maestre, C.**, Cully, A., Gonzales, C., and Doncieux, S. (2015, August). Bootstrapping interactions with objects from raw sensorimotor data: a Novelty Search based approach. In IEEE International Conference on Developmental and Learning and on Epigenetic Robotics (ICDL-Epirob). Referenced in the bibliography as Maestre et al. (2015).

Contents

4.1 Introduction

Learning models to predict a robot's actions and skills to execute them requires a substantial amount of data of robot-object interactions (see Definition 2). A robot with the capacity to move, i.e. endowed with a kinematic model, could acquire this data through an exploration of its environment, given a minimal *a priori* knowledge to drive the exploration. To that end, in this chapter, we present a method named Novelty-driven Evolutionary Babbling (NovEB), designed to perform a task-agnostic exploration of an environment. Its main feature is to look for actions that maximize novelty in the raw sensorimotor space. It is based on Novelty Search (Lehman and Stanley, 2011), which relies on Evolutionary Algorithms driven by a behavior novelty criterion. The outcome of this approach is a dataset of interactions composed of robot actions and their consequences, e.g. changes in the visual perception. The generated data are aimed at preparing a future developmental step

for a robot, which would consist in training classifiers or predictors, e.g. A^2L. This method uses as *a priori* knowledge:

- w.r.t. to the actions, a forward/inverse kinematic model is available[1].
- w.r.t. to the action consequences, a *distance* to compare them is predefined (further details are available in Section 3.1). Also, NovEB makes the assumption that new perceptions result from robot actions.

Besides, although the method does not need any information of the environment to run, in order to reach a good performance it is necessary that after each action the environment is reset, that is, objects move to their initial position.

4.2 Method

The method generates many different robot arm trajectories, $c^1, \ldots, c^S$, and it looks at the modifications they may create in the environment, as perceived from the robot's sensors (vision in particular). Based on the Novelty Search principles, a movement (see Definition 7) that generates perceptions that have never been encountered before has a higher chance to survive, namely to be selected to generate new close movements through the mutation operator. Movements that do not generate any perceptual novelty are discarded, thus focusing the search on movements generating new perceptions.

The main algorithm of NovEB to explore an environment is presented in Algorithm 1. *Generate_children_pop(pop)* is the function that creates a new population with mutation and crossover. *Novelty(i, p, a)* is defined as in Equation 3.1, on the basis of both the nearest neighbors in the current population, and in an archive of past behaviors with a distance defined in a behavioral space (an example is described in Section 4.3). At each generation, the individual with the highest novelty is added to the archive. In mono-objective problems, *Select(pop)* is an elitist algorithm that selects the best individuals among the parent and the children populations, using the function *Fitness(i)*. Both *Generate_children_pop(pop)* and *Select(pop)* are based on NSGA-II Deb et al. (2002), a *Pareto-based multi-objective evolutionary algorithm*, which is a state-of-the art algorithm for multi-objective problems; but it is also very efficient in mono-objective ones.

The *genotype* is a vector of waypoints in an Euclidean space, used to define a trajectory of the robot's end effector[2]. In the initial population, these values are randomly generated within a defined range. Afterwards, the values are modified by the mutation operator, without any restriction on the resulting values. Two mutation operators are defined. The first one adds a random Gaussian noise to a

[1] The robot's kinematic model is available in all the works presented in this manuscript. Henceforth, we omit it when describing the *a priori* knowledge provided.

[2] This type of environment exploration is inspired by the goal-directed exploration (Rolf et al., 2010). We do not follow the same terminology to avoid any misunderstanding with the use of the term goal as a synonym of a task to solve.

Algorithm 1 Novelty-driven Evolutionary Babbling (NovEB)

```
1: pop ← c^1, c^2, ..., c^S                          ▷ random population
2: a ← ∅                                   ▷ a stands for the novelty archive
3: g ← 0                                   ▷ number of generations in the population
4: while g < g_max do
5:     for i ∈ pop do
6:         Execute_trajectory (i)                    ▷ compute behaviour of i
7:     max_novelty ← 0
8:     for i ∈ pop do
9:         Fitness(i) ← Novelty(i, p, a)
10:        if Novelty(i, p, a) >max_novelty then
11:            max_novelty ← Novelty(i, p, a)
12:            best_candidate ← i
13:    archive ← archive ∪ {best_candidate}
14:    pop ← pop ∪ Generate_children_pop(pop)
15:    pop ←Select(pop)
16:    g ← g + 1
```

given trajectory. The second one can change the complexity of the trajectory by adding one waypoint. Added points are put in the middle of two other points of the trajectory. This ability to change the complexity of the genotype is inherited from NEAT (Stanley and Miikkulainen, 2002), and is also a feature of Novelty Search: the search starts with simple solutions, considers the behaviours they can generate and progressively considers solutions of higher complexities (Lehman and Stanley, 2011).

The behavior associated with an individual is an image of the scene as seen by the robot once its arm has come back to its initial position[3]. The robot's movements eventually change the scene by moving objects, being this reflected in the gathered images. In this work, a behavioral distance is predefined, used to compute novelty among the images. Namely, this distance drives the exploration of the environment.

4.3 Experimental Framework

The objective of this experiment is the generation of contacts of the robot's end effector with objects around the robot. The experiment is executed in a tabletop setup, i.e. a table with objects on top of it: on the left a gray box, on the front-center a can, on the right a blue box, and on the back-center a ball. All the objects

[3]This is meant to avoid to take into account robot's arm movements as a source of novelty. This could be useful to generate a self-model, but it would be misleading for building predictive models.

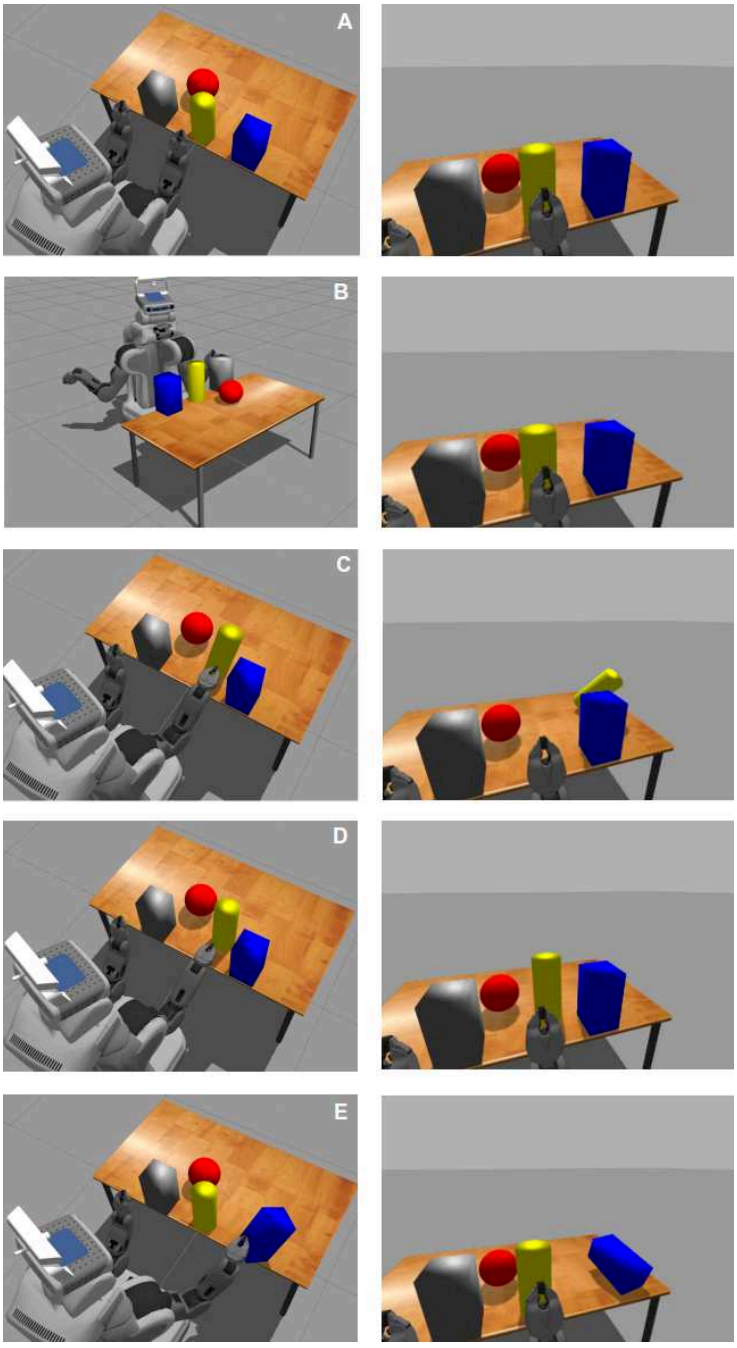


Figure 4.1: Examples of different moments during the experiments using NovEB. Each row represents the execution of a trajectory (except for column A that displays the initial state before any trajectory is executed). The right column represents the moment just after the execution of a trajectory, i.e., when the robot's arm has completed the trajectory. The left column shows the final image obtained once the arm has come back to its initial position. (Row B) A trajectory without any object on the table being touched does not produce any change in the environment. (Row C) A trajectory in which an object is touched, hence producing a change in its position. (Row D) A trajectory in which the contact with the object is slightly different can result in a very different output. (Row E) A trajectory in which several the blue box is touched. An illustrating video of the experiment is available: https://youtu.be/zCO7qOIvKOU

are within reach of the right arm of the robot, except for the ball, that can be moved only if it is pushed by another object. The two boxes and the can are located at the border of the table, and the ball is located behind them. The can is located just in front of the robot, in the middle of the table (see Fig. 4.1). The weight chosen for the objects on the table is low to facilitate their movement when the end effector of the robot makes contact with them. The experiment run for around 20000 iterations, as in Mugan and Kuipers (2012). At the end of each iteration, the objects are reinitialized to their original position.

A trajectory is initially composed of an initial position, common to all trajectories, and a final point to be reached. The randomly generated final coordinates are in the range [0,1], being able to go beyond these bounds afterwards. The robot relies on a motion planner, called OMPL[4], to plan a trajectory. It is not always possible to compute a trajectory given a final point, because either the trajectory is without reach of the robot, or the trajectory includes collisions with the table, the ground or the robot. Only the safe and feasible trajectories are executed.

The behavior descriptor associated to a trajectory is the final image of the scene, which is taken once the arm came back to the initial position. To compute the distances required for the novelty objective, images are encoded into a numeric string by using the pHash library[5], because perceptual hashes are close to one another if the features [of the two images] are similar (Zauner, 2010). The distance between two strings is then computed with the Hamming distance (Hamming, 1950).

To assess the performance of our approach, the final positions of the objects of the scene have also been recorded (objects and their positions are unknown to the robot).

4.3.1 Robotic Platform

The robot used in the experiments is called PR2[6]. The main feature of the PR2 robot w.r.t. the experiments is that the robot arm has 7 degrees of freedom, and another one for the gripper that has not been used in these experiments. Environment images are acquired using the right color stereo camera on the robot's head. ROS Hydro Medusa[7] was used to manage the robot. The simulation has been executed in Gazebo 1.9[8]. MoveIt[9] provides the robot with the capability of defining safe trajectories for the end effector, using OMPL, based on a set of points in the space. It also provides collision avoidance. The evolutionary algorithm, on which the execution of this method relies, is executed in Sferes$_{v2}$ (Mouret and Doncieux, 2010), a framework for evolutionary computation designed for multi-core parallelization.

[4] http://ompl.kavrakilab.org/
[5] https://www.phash.org/
[6] http://www.willowgarage.com/pages/pr2/specs
[7] http://wiki.ros.org/hydro
[8] http://gazebosim.org/
[9] http://moveit.ros.org/

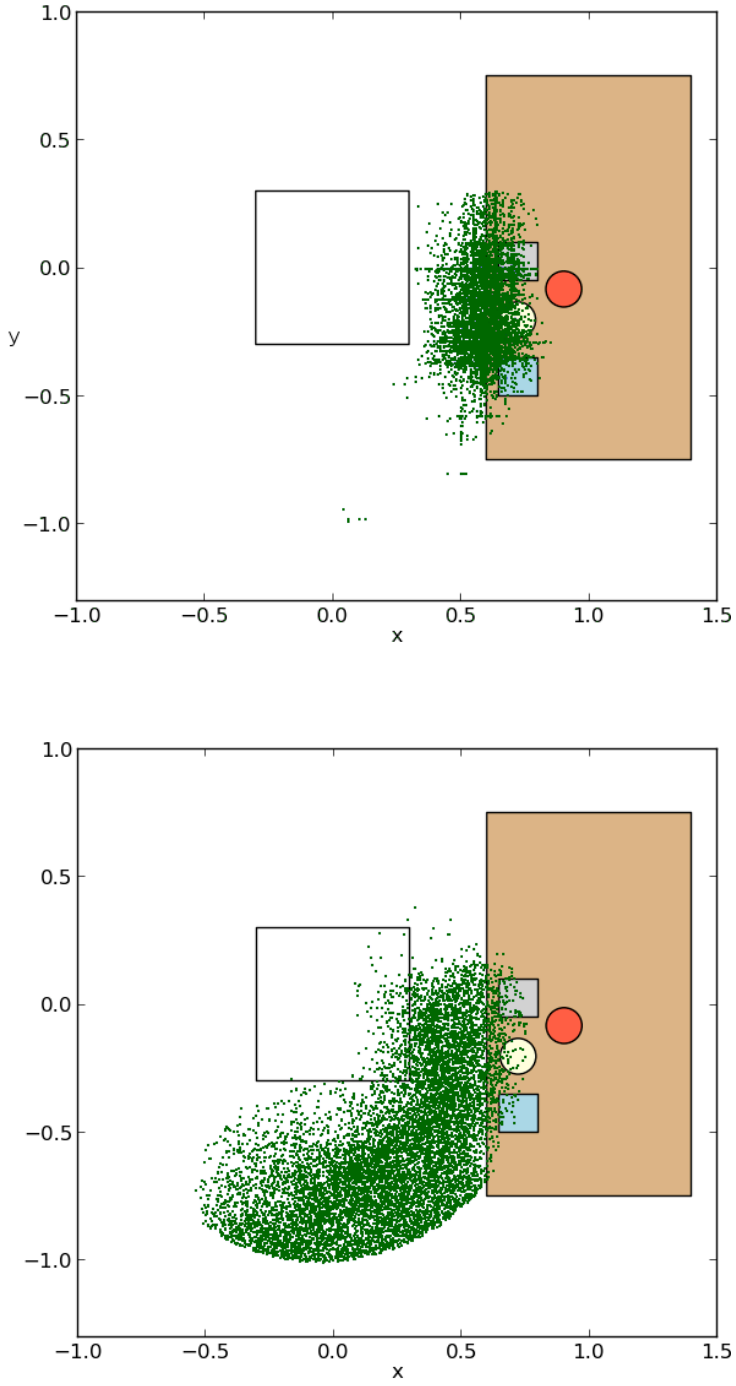


Figure 4.2: Top view showing the space covered during the execution of NovEB (top) and during that of the control experiment (bottom). The representation is composed of the PR2 (white box) in front of the table (brown box). The small circle and boxes represents the objects on the table. The green dots represent the final position of each trajectory executed by the robot. For the control experiment, dots show that a majority of the space within reach of the robot's right arm is searched, whereas NovEB clearly focuses on the interesting parts of the space.

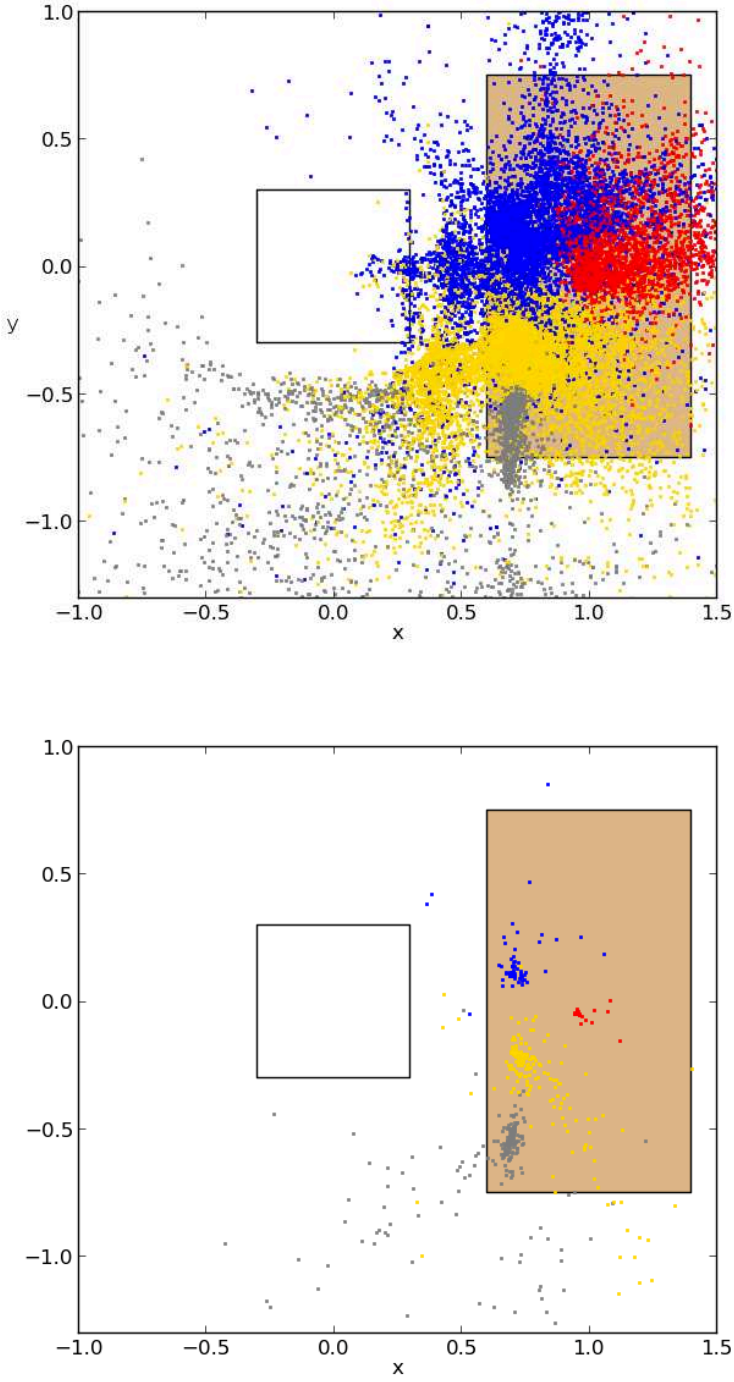


Figure 4.3: Top view showing the final positions of the objects during the execution of NovEB (top) and during that of the control experiment (bottom). The representation is similar to that of Fig. 4.2. The blue, gray, yellow and red dots represent the final positions of the blue box, of the gray box, of the can, and of the ball respectively. Some dots are located behind the robot due to the fact that the simulator's physics engine not always handling correctly the dynamics of the objects. However, this has no impact on the results of the experiments because such objects are out of the field of vision of the robot. Note again that NovEB produces much more changes than the control experiment.

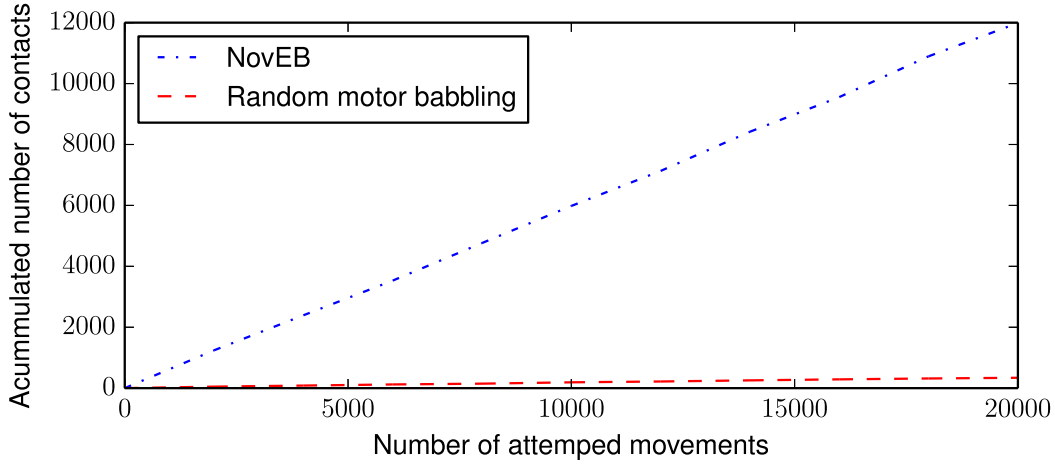


Figure 4.4: Comparison of the accumulated number of contacts produced in both experiments w.r.t. the number of attempted movements.

4.3.2 Experiments

4.3.2.1 Control experiment

A random motor babbling is defined as a control experiment, because of its wide use in different works to generate knowledge about the world, as in the work of Mugan and Kuipers (2012). In this experiment the motor babbling consists in the definition of different sets of joint values provided to the right arm of the PR2. These values are randomly generated, and used by the robot to perform a movement. The attempted movements are defined sequentially, until reaching the value of 20000. Only the safe and feasible movements are performed, called executed movements.

4.3.3 Experimental Results

The final positions of the robot's end effector reached during one run of the execution of two experiments are shown in Fig. 4.2. The space covered by NovEB is focused on the table, while the space explored by the random babbling covers a big part of the joint space of the right arm of the robot. During the exploration performed by NovEB, some final positions are located far from the table (isolated points at the bottom at the top of Fig. 4.2). These points correspond to the initial randomly generated solutions, and the exploration around them was stopped as they did not generate novelty.

In the current work, a contact is counted when the end effector of the right arm touches at least one object on the table. NovEB generated a high number of contacts with the objects of the scenario (see Fig. 4.4). When observing the final positions of the objects, it also appears that they spread over a larger portion of the space for NovEB than for the control experiment (see Fig.4.3). Therefore, the contacts generated by NovEB produced a high diversity of behaviors in the scenario.

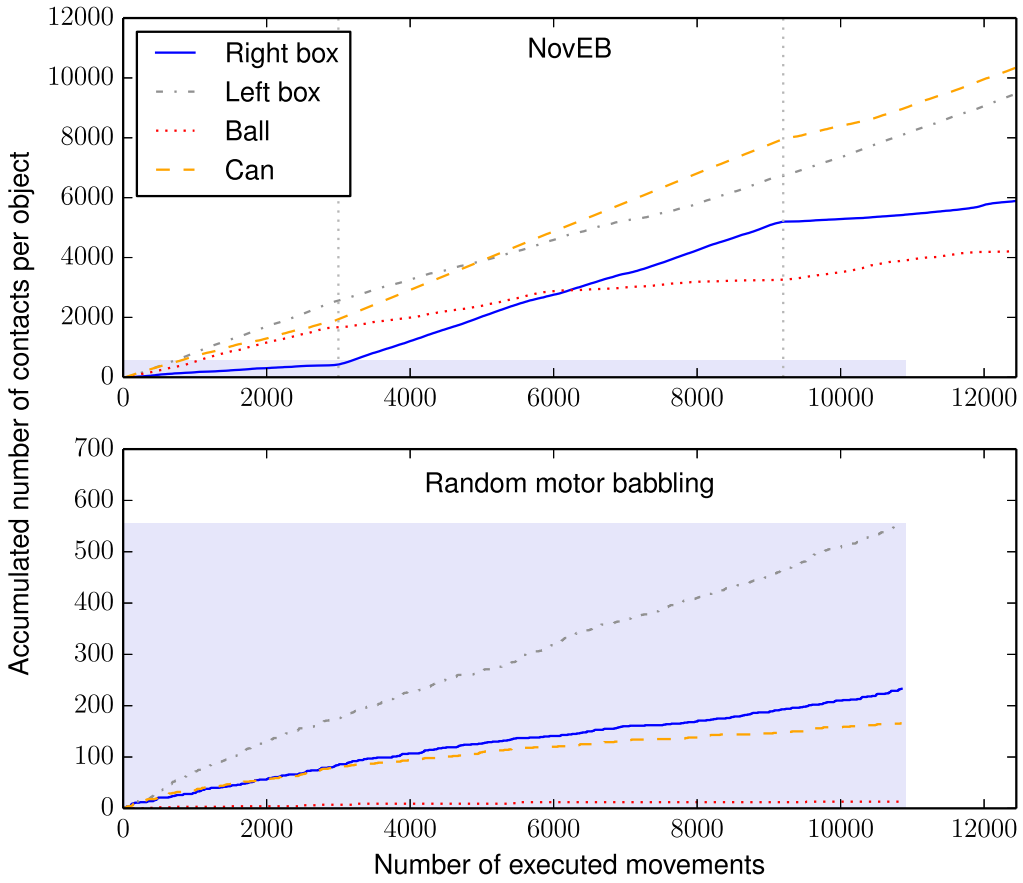


Figure 4.5: At the top, accumulated number of contacts created by NovEB for each object. Three regions can be distinguished, based on the growth of the number of contacts. At the bottom, accumulated number of contacts created during the control experiment for each object w.r.t the number of executed movements. The shaded areas represent the same area in both figures.

The random babbling generated a small number of contacts with each object (bottom of Fig. 4.5). In contrast, the number of contacts produced with each object is large in NovEB (top of Fig. 4.5). For instance, the number of changes produced to the can is over 10000, meaning that in almost each executed trajectory during the exploration the position of the can was modified. This result was expected as the can is located in the center of the table, and interactions with any of the boxes could modify its position. The babbling can be split up into three different phases: (1) At the beginning, the robot gets focused on the objects on the left side of the table. (2) When reaching 3000 executed movements, the robot changes its focus to the can. The growth of the number of interactions with the ball and the left box decreases and the contacts with the can and the right box increase. (3) After 9000

executed movements, the novelty found in the right part of the table decreases, and the robot comes back to search new behaviors in the left side.

4.4 Conclusions and Open Questions

In this chapter, we have proposed a novel method for generating interactions with the objects surrounding a robot through babbling. The approach has been applied on a simulated robot, which discovers on its own which regions of the workspace generate novel perceptions and focuses its exploration around them. The results show that NovEB is able to generate several thousand different interactions with this environment, an order of magnitude higher than the number of interactions produced with a random motor babbling approach. This difference is obtained thanks to the ability of NovEB to focus its exploration in regions that lead to novel visual perceptions. The outcome of this approach is a dataset of interactions composed of robot actions and their consequences, e.g. changes in the visual perception.

NovEB can be considered as an intrinsic motivation for exploration, like the Intelligence Adaptive Curiosity (IAC) defined by Oudeyer et al. (2007). One of the main differences between these two approaches lies in the assumptions on which these methods are based. The only requirement for NovEB is the robot's kinematic model, and the definition of a distance between two perceptions (in this chapter, between two images). This is specific to the robot's sensors and is independent from the task or the environment. Conversely, IAC, at least in its current implementation, requires to train predictors in order to estimate the learning progress. Training such algorithms to predict the consequences of an action only on the basis of raw perceptions is a challenge *per se*. Although recent works based on deep learning techniques are starting to predict the image that the robot's camera will capture based only on previously captured raw images and the executed actions, as Lesort and Filliat (2017), training these predictors is still an open question. Current implementations of IAC rely on higher level information, for example on the position of the objects in the scene (Oudeyer et al., 2007). However, providing the tools that extract these higher information from raw perceptions cannot be environment-agnostic. For example, when predicting the positions of the objects, the algorithm needs to know how many objects compose the scene, or how to extract these objects from the raw perceptions (using large object database, for instance). Based on this observation, these two approaches can be complementary. NovEB can be used to generate a large amount of data that can afterwards be used to extract information from the scene (number or shape of objects, for instance). Then, the high level information extracted can be used to run IAC for a detailed or goal-oriented exploration (Baranes and Oudeyer, 2013). NovEB can be also complementary with LeGoff et al. (2017), a work to segment the environment based on visual inputs. The trajectories generated by our method can be related to the object position using the inverse kinematic model, and thus they could be used as *push* primitive.

As NovEB is driven by novelty only, it suffers from some of the limitations that

have motivated the development of IAC: it should get focused on interactions that generate perceptions with a large variability. These interactions could constrain the exploration to an area of the setup, neglecting contacts in other areas producing less novelty, i.e. avoiding the exploration of possible relevant areas. IAC can avoid this phenomenon as the learning progress in such situations will remain low. This would be a strong limitation for the exploration ability of the system if NovEB was expected to handle the whole developmental process. But this is not an issue, as NovEB is aimed only at acquiring the data to bootstrap other developmental processes.

The direct comparison of the action consequences produces the necessity to run the method in static environments that reset after each action to obtain a good performance. This constraint makes very difficult the execution of the method by a physical robot. An alternative would be the direct execution of its output in simulation by the physical robot. However, this approach could not be satisfactory because of the *reality gap* (Doncieux and Mouret, 2014). Different approaches have been developed in the next chapters to avoid this constraint.

Chapter 5

Autonomous Generation of Interactions with the Environment

The results and text of this chapter have been published in the following article.

- **Maestre, C.**, Mukhtar, G., Gonzales, C., and Doncieux, S. (2017, September). Iterative affordance learning with adaptive action generation. In IEEE International Conference on Developmental and Learning and on Epigenetic Robotics (ICDL-Epirob). Referenced in the bibliography as Maestre et al. (2017b).

Contents

5.1 Introduction

This chapter focuses on the autonomous generation by a robot of a dataset of interactions (see Definitions 3, 7 and 8) used to build a skill reproducing an effect on an object. In this chapter, a list of effects to reproduce is provided. Interactions are represented by the position of the robot's end-effector and the object position during the interaction execution, i.e. they are represented by low-level states. In this work an object is represented by its position, not relying either on its orientation or its shape, as young infants (Rosenbaum, 2010).

On the one hand, in order to solve the drawbacks identified in the previous chapter we provide more *a priori* knowledge to the method about the environment. In this case, a skill is related to a known object, because the skill compares the relative position of the object to the end-effector position in order to infer the next movement of the robot. This relative position is used to compare changes in the object, making unnecessary the definition of a distance based on visual inputs. Therefore, the object can be placed in any location within reach of the robot, not needing to reset the object position after each action.

On the other hand, we reduce the *a priori* knowledge to explore the environment. In this case, the exploration is performed using random actions. The number of robot-object interactions generated by random actions is low (see Figure 4.5). As the goal of the exploration is to increase the number of interactions, this exploration is not completely random. The generated actions extend those trajectories of the available dataset of interactions touching the object with random movements (see Section 5.2.4). This approach, inspired by intrinsic motivations, allows the robot to progressively extend the dataset using the knowledge acquired in previous explorations, although in a naive fashion. However, even using this approach the exploration of the robot's environment would generate a low number of interactions, because of representing a big search space. Therefore, this search space is constrained to two dimensions, and the movements are discretized (more details are available in Section 5.3). This is a drawback w.r.t. methods driving the exploration using some knowledge, for example intrinsic motivations, as in the case of NovEB. But, it is the consequence of reducing the *a priori* information provided for the exploration using random actions.

In this chapter we present the two complementary processes of A^2L:

- Skill Building: given a dataset of interactions it builds skills that infer actions reproducing effects on objects, adapted to the object position.

- Iterative Interaction Acquisition and Validation: an iterative method generating and validating the dataset of interactions.

In order to execute A^2L, a dataset of interactions (see Definition 2) must have been generated in a previous developmental step (see Fig. 5.1), for example the result of executing NovEB. In this chapter, this initial dataset is the result of a random motor exploration performed by the robot's end-effector.

Once this dataset is available, the execution of the iterative process can start. At the beginning of each iteration of the process, a short exploration of the environment is executed. New robot-object interactions are identified, which are combined with the available dataset of interactions into a new extended dataset. Then, the skill builder creates a skill based on this dataset. The last step of an iteration consists in the validation of the skill trying to reproduce the set of given effects. If more effects are reproduced than in the previous iteration the extended dataset is consolidated as the working dataset, to be extended in the next iterations of the process. Otherwise,

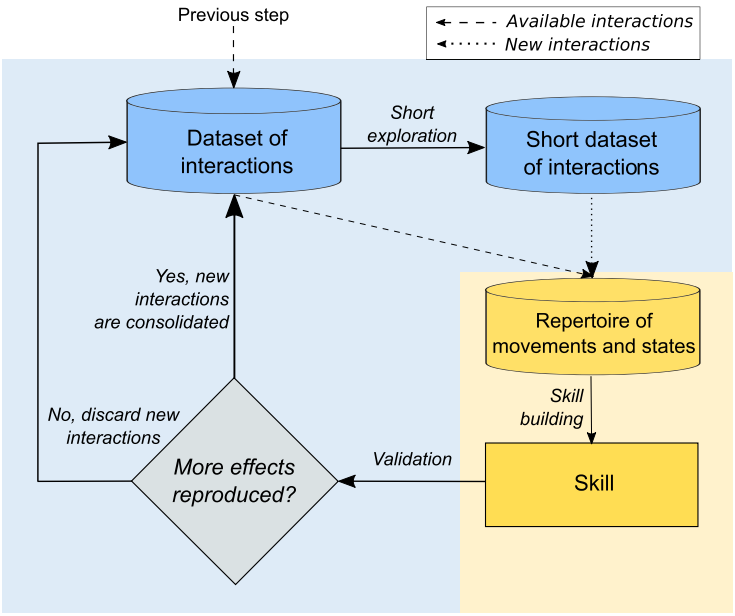


Figure 5.1: Workflow of A^{2}L, explained in Section 5.2.4. The blue area represents the Iterative Interaction Acquisition and Validation process. The yellow area represents the Skill Building process.

the extended dataset is discarded. The iterative process ends when all the effects are reproduced or after a preset number of iterations is reached.

Two sets of experiments are executed to assess A^{2}L. The goal for both experiments consists in the generation of a dataset of interactions allowing the method to build skills executed by the robot to *push* a box in different directions. The first experiment is performed in a *virtual setup*, implemented in Python without a physics engine (Section 5.3.1). In this setup a simulated robotic arm computes a mathematical approximation of the possible outcome of the execution of a trajectory. The effect produced by the action is also mathematically computed. In the second experiment, a simulated Baxter robot tries to build similar skills in a tabletop setup (Section 5.3.2). In this case, the setup is defined in Gazebo, including a physics engine. Finally, the skills built by the dataset of interactions are tested by the real Baxter in a task consisting in performing a continuous interaction with the box *pushing* it all over the table. Both experiments are executed in static environments, i.e. the position of the box can only change during the execution of a trajectory if it is touched by the robot. However, in the first experiment the position of the box is fixed, and in the second experiment the box can be located at different positions.

Algorithm 2 Initial random exploration

nb_it_p: number of current iteration executed
nb_it_{max}: maximum number of iterations
$\mathbb{D}_b$: dataset of interactions obtained from exploration
y_p: position of the object at the end of iteration p
$y_{p\text{-}1}$: position of the object at the end of iteration p-1

1: $nb\_int_p = 1$
2: $\mathbb{D}_b = \emptyset$
3: **while** $nb\_it_p \leq nb\_it_{max}$ **do**
4: $\quad \{x_p\} \leftarrow GenerateRandomTrajectory()$
5: $\quad y_p$, $\mathrm{XY_p} \leftarrow ExecuteTrajectory(\{x_p\})$
6: $\quad \Delta f_p \leftarrow y_p$ - $y_{p\text{-}1}$
7: $\quad$ **if** $\Delta f_p \neq 0$ **then**
8: $\quad\quad e_p \leftarrow IdentifyEffect(\Delta f_p)$
9: $\quad\quad \mathbb{D}_b \leftarrow AddInteractionsToDataset(XY_p,\ e_i)$
10: $\quad Increase(nb\_int_p)$

5.2 Iterative Developmental Framework

5.2.1 Initial Available Information

Before the execution of the method, two sets of information are required: (i) an initial dataset of interactions, representing robot-object interactions, to bootstrap the skill building process and to be increased in posterior iterations of the method; and (ii) a set of effects to be reproduced by the robot, to assess the generated states.

Dataset of Interactions

The dataset of interactions is environment-dependent, and thus it must be generated by the robot. It is initially composed of the interactions acquired by the robot during a random, goal-free, exploration of its environment (Demiris and Dearden, 2005), i.e. the execution of random actions (Algorithm 2). The interactions represent both the position of one of the robot's end-effectors and the object position at an instant of time:

$$x_t = \textit{end effector position}$$
$$y_t = \textit{object position}$$
$$XY^k = \{\ (x_0^{\mathrm{k}},\ y_0^{\mathrm{k}}),\ \ldots,\ (x_{\mathrm{T}}^{\mathrm{k}},\ y_{\mathrm{T}}^{\mathrm{k}})\ \}$$
$$e = \textit{label associated to a specific effect}$$
$$\mathbb{D} = \{\ (XY^k,\ \mathrm{e})\ \}$$

where XY^k represents an interaction, k, between the robot's end-effector and the object when executing a trajectory, and $\mathbb{D}$ represents a dataset of interactions. x_t

and y_t are acquired at different instants of the trajectory execution, $t \in [0, T]$, where T represents the total time of the execution of the trajectory.

Set of Effects

The set of effects to reproduce is externally provided:

$$f_t = y_t$$
$$\Delta f_t = y_t \text{ - } y_{t\text{-}1}$$
$$E = \{ (e_1, \widehat{\Delta} f_1), \ldots, (e_N, \widehat{\Delta} f_N) \}$$

where f_t represents the object position at an instant of time, Δf_t represents a variation of the object position between two instants of time, E represents the set of effects to reproduce, N is the number of effects, and $\widehat{\Delta} f_i$ represents the change of the object position associated to the effect e_i, which is also provided to the robot.

5.2.2 Method

This section details the two complementary processes of A^{2}L:

- *Skill Building*: given a dataset of interactions this process builds skills that infer actions to reproduce an effect on an object.
- *Iterative Interaction Acquisition and Validation*: an iterative process generating new interactions and validating them.

5.2.3 Skill Building

In the current manuscript, an action is a sequence of movements to reproduce an effect on an object (see Definition 5). Given a context, a skill infers and executes an action adapted to the object position reproducing an effect on the object. Therefore, a skill is an action generator ϕ (see Definition 6) inferring a movement reproducing an effect on the object given a context:

$$\phi(e, context) = \Delta x_t$$

The action generator is implemented as a Bayesian Network (BN) (Pearl, 1988) (introduced in Section 3.2), similarly to (Montesano et al., 2008), among others. A BN is a graphical representation of dependencies for probabilistic reasoning, in which the nodes represent *random variables* and the lack of arcs represent *independence relationships* between the variables (further details are available in Section 3.2.1). The reasons for using a BN as an action generator are twofold: (i) it has a strong inference capability, and (ii) its representation of the contextual state relations as probabilistic dependencies allows one to analyze and understand the outcomes of learning. Lauritzen (1992) introduced a BN combining variables using continuous and discrete values, and there are works applying similar concepts to robotics (Osório et al., 2010; Stramandinoli et al., 2017). However, Osorio mentions

that the performance of a BN combining continuous and discrete variables is much slower than directly using discrete variables. As each cycle of the close loop performed by a skill is very short, A^{2}L uses discrete values to represent movements, low-level and high-level states. The movements and contextual states are discretized when the dataset of interactions is transformed into the repertoire of movements and contextual states. Although the discretization used for each variable is unique, many states are common to different contexts, e.g. object position.

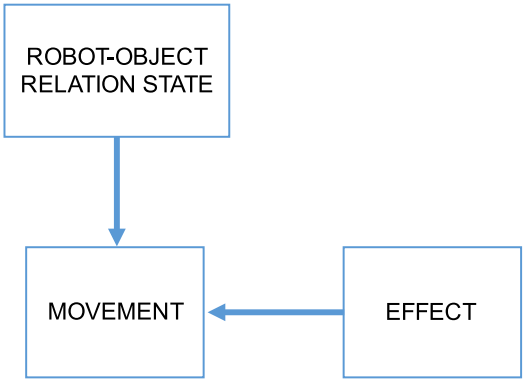


Figure 5.2: Conditional dependencies of the *hard-coded* structure allowing the action generator to infer the next *movement* based on the *effect* and the *low-level relation states*.

Step 1: Transformation of the Dataset of Interactions into a Repertoire of Movements and Contextual States

A BN is learned based on a dataset containing information about the random variables. In this dataset the information must follow a similar format, i.e. the same number of variables with similar type of values. Therefore, a BN that generates trajectories of actions can be learned with a dataset composed of trajectories of similar length. Other works generating trajectories follow this approach (Calinon et al., 2010, 2011). However, this generates a very strong constraint for the use of the method. In order to avoid this constraint, in the current work trajectories can have any length because each interaction, composed of a trajectory and the related object positions, is transformed into a set of tuples (see Figure 1.2). Each tuple is composed of a movement and a low-level contextual state at an instant of time (depicted in Figure 1.2):

$$(XY^k, \mathrm{e}) = (\ \{\ \Delta x_t,\ \delta_\mathrm{t}\ \},\ \mathrm{e})$$
$$\delta_\mathrm{t} \equiv \delta_\mathrm{t}(f_t,\ x_t)$$

where δ_t represents the *robot-object relation state*. As each tuple has the same variables and size, the BN learns probability distributions of tuples, which can be exploited to infer back the probabilities of whole trajectories (see Equation 5.2).

Definition 9 *Robot-Object Relation State: A robot-object relation state, or just relation state, is a physical relation between the robot's end-effector and the object at an instant of time.*

An example of robot-object relation state is the *distance* and *orientation* between the robot's end-effector position and the object position.

In other words, the dataset of interactions, $\mathbb{D}$, is transformed into a repertoire of movements and states, R:

$$\mathrm{R} = \{\ (\mathrm{e},\ \Delta \mathrm{x_t}\ ,\ \delta_\mathrm{t})\ \}$$

As already discussed in Section 1.2, the performance of a BN combining continuous and discrete variables is often much slower than directly using discrete variables. As $\mathrm{A^2L}$ is aimed to generate actions adapted to the object position, the action generator needs to generate movements as quickly as possible. And thus, in the current work, the action generator infers *discrete movements* based on *discrete robot-object relation states* and *discrete effects*. Their discretization is performed when the dataset of interactions, $\mathbb{D}$, is transformed into the repertoire of movements and states, R, performed based on a *discretization configuration*. This configuration may have a deep impact in the results obtained using the method. A suitable configuration must entail a trade-off between being generic to be suitable for different sets of effects, and specific enough for the current set of effects. In the current work this configuration has been empirically designed for each experiment. Future work may include the selection of the discretization configuration in a developmental process in which discretizations are generated and tested using an iterative approach.

Step 2: Learning the Action Generator

Once the discrete repertoire of movements and states, R, is available the BN can be learned. Learning a BN consists of two steps: (i) identifying the structure representing the conditional dependencies among the random variables, and (ii) learning their conditional probability distributions (CPD). The structure can be either learned combining some of the structure learning methods available in the literature to a score (see Section 3.2) or it can be externally provided by a designer. However, the CPDs are always learned from the available repertoire.

In order to complete the information of the skill building process, let us define it based on the new available definitions. Given an effect to reproduce and the robot-object relation state, an action generator, ϕ, infers the next movement to execute to reproduce the effect on the object:

$$\phi(e, \delta_\mathrm{t}) = \arg\max_{\widehat{\Delta} x_\mathrm{t}} P(\widehat{\Delta} x_\mathrm{t} \mid e, \delta_\mathrm{t}) = \Delta x_\mathrm{t} \tag{5.1}$$

Algorithm 3 Iterative Interaction Acquisition and Validation

nb_it_p: number of current iteration
nb_it_{max}: maximum number of iterations
x_1: initial position of the end-effector
ψ_{p}: score computed in an iteration
$\psi_{\max}$: maximum score obtained
E: set of available effects
$\mathbb{D}_p$: available dataset on interactions
$\mathbb{D}_s$: dataset on interactions from short exploration
$\mathbb{D}_p^*$: extended dataset on interactions
R_{p}: repertoire of discrete states
ϕ_{p}: action generator

1: $nb\_it_p = 1$
2: $\psi_{\max} = 0$
3: **while** $nb\_it_p \leq nb\_it_{max}$ **do**
4: $\quad \mathbb{D}_{\mathrm{s}} \leftarrow$ *ExecuteShortExploration*($\mathbb{D}_p$)
5: $\quad \mathbb{D}_p^* \leftarrow$ *ComputeExtendedDataset*($\mathbb{D}_p$, $\mathbb{D}_s$)
6: $\quad R_{\mathrm{p}} \leftarrow$ *DiscretizeAcquiredStates*($\mathbb{D}_p^*$)
7: $\quad \phi_{\mathrm{p}} \leftarrow$ *BuildSkill*(R_p)
8: $\quad \psi_{\mathrm{p}} \leftarrow$ *ValidateRepertoire*(E, ϕ_p, x_0)
9: $\quad$ **if** $\psi_{\mathrm{p}} \geq \psi_{\mathrm{p\text{-}1}}$ **then**
10: $\quad\quad \psi_{\max} = \psi_{\mathrm{p}}$
11: $\quad\quad \mathbb{D}_{\mathrm{p}} = \mathbb{D}_{\mathrm{p}}^*$
12: $\quad$ *Increase*(nb_it_p)

5.2.4 Iterative Interaction Acquisition and Validation

Phase 1: Interaction Acquisition

The process presented in Section 5.2.3, i.e. the skill building, creates a skill based on a dataset of robot-object interactions. However, given a set of effects to reproduce, it is possible that some of them could not be generated at an instant of time. In order to build a skill reproducing those effects the dataset has to be extended with new interactions. To that end, the robot executes at the beginning of each iteration of the process a short exploration around the object, i.e. it executes a small number of trajectories, possibly generating new interactions with the object. These new interactions are combined with the available dataset, generating an extended dataset of interactions, $\mathbb{D}_{\mathrm{i}}^*$. This extended dataset is later used in the skill building.

The short exploration is based on the interactions within the dataset touching the object. A new trajectory is generated randomly selecting one of those trajectories, and modifying it. This modification consists in selecting one waypoint of

Algorithm 4 Validation of the repertoire of states

E: set of identified effects
ϕ_{p}: action generator
a_{e}: action inferred for effect e
x_0: initial position of the end-effector
y_0: reference position of the object
x_{pq}: current position of the end-effector
y_{pq}: current position of the object
N: number of available effects
nb_mov_{q}: current number of movement
nb_mov_{max}: maximum number of movements to execute
θ: available discretization configuration
δ_{pq}: relation state based on environment and robot states
$\Delta \mathrm{x}_{\mathrm{pq}}$: discrete movement
$\hat{e}$: expected effect
e_i: obtained effect
res_{e_i}: result of an effect
$w_{res_{e_i}}$: weight associated to a result
ψ_{p}: score obtained

1: **function** ValidateRepertoire(E, ϕ_{p}, x_0)
2: $\quad \psi_{\mathrm{p}} = 0$
3: $\quad$ **for** $\hat{e} \in E$ **do**
4: $\quad\quad y_0 \leftarrow$ *GetInitialObjectPosition()*
5: $\quad\quad x_{pq} = x_0$
6: $\quad\quad nb\_mov_{\mathrm{q}} = 0$
7: $\quad\quad a_{\hat{e}} = \emptyset$
8: $\quad\quad$ **while** $\neg contact \cap nb\_mov_{\mathrm{q}} < nb\_mov_{\mathrm{max}}$ **do**
9: $\quad\quad\quad y_{pq} \leftarrow$ *GetObjectPosition()*
10: $\quad\quad\quad \delta_{\mathrm{pq}} \leftarrow$ *ComputeInteractionFeatures*(x_{pq}, y_{pq})
11: $\quad\quad\quad \Delta \mathrm{x}_{\mathrm{pq}} \leftarrow$ *InfereMovement*(ϕ_p, $\hat{e}$, δ_{pq}) ▷ Equation 5.1
12: $\quad\quad\quad x_{pq} \leftarrow$ *MovementExecution*(x_{pq}, Δx_{pq})
13: $\quad\quad\quad contact \leftarrow$ *CheckObjectContact()*
14: $\quad\quad\quad$ *Increase*(nb_mov_q)
15: $\quad\quad$ **if** *contact* **then**
16: $\quad\quad\quad e_{a_{\hat{e}}} \leftarrow$ *IdentifyObtainedEffect*(y_{pq}, y_0)
17: $\quad\quad\quad$ **if** $e_{a_{\hat{e}}} = \hat{e}$ **then**:

18: $res_{\hat{e}} = success$
19: **else**
20: $res_{\hat{e}} = false\_positive$
21: **else**
22: $res_{\hat{e}} = failure$
23: $\psi_{\mathrm{p}} = \psi_{\mathrm{p}} + res_{\hat{e}} * w_{res_{\hat{e}}}$ ▷ Update score
24: **return** $\psi_{\mathrm{p}} * \mathrm{N}$

the trajectory and modify it moving it the distance and orientation of one of the discrete movements used by the robot in the current work. For example, the final waypoint of a vertical straight trajectory can be moved to the right transforming the trajectory into a curve. This approach, inspired by intrinsic motivations, allows the robot to progressively extend the dataset of interactions using the knowledge acquired in previous explorations, although in a naive fashion.

The execution of a trajectory by a robot is very time consuming, possibly taking few seconds to complete it. The execution by the robot of all the trajectories defined in the short exploration could entail the execution of hundreds of trajectories, most of them not having any impact on the object. In the current chapter, this issue has been addressed making a mathematical estimation of the possible contact of a trajectory with the object, based on its position. Only if the Euclidean distance of a waypoint of the trajectory is under a predefined threshold, i.e. if it is *close* to the object, the action is executed by the robot.

Phase 2: Skill Generation Phase

This phase just consists in executing the complementary skill building process providing the extended dataset of interactions, $\mathbb{D}_{\mathrm{i}}^{*}$, to generate a skill to validate the datatset in the next phase.

Phase 3: Interaction Validation

The skill validation of the extended dataset of interactions consists in the comparison of the number of effects reproduced before and after the extension of the dataset of interactions. More precisely, if more effects are generated using the extended dataset than before extending it, this is consolidated as the working dataset, to be extended in the next iterations of the process. Otherwise, it is discarded. A score is computed to measure the number of effects reproduced. This score relies on the result of the actions inferred by the robot trying to reproduce the effects. These actions produce changes in the object position:

$$\Delta f_t^{a_e} = y_t^{a_e} - y_{t\text{-}p}^{a_e} \in \mathrm{S_f}$$

where a^e represents an action to reproduce an effect, S_f represents the set of changes of the object position produced by the robot during the exploration, and t and p represent different instants of time, such that t is posterior to p. An action is considered a *success* if after its execution the change of the object position produced in the object is similar to the change expected from the desired effect, n:

$$\exists\ n \in [1,N],\ \exists\ t \in [0,T\text{-}1],\ \Delta f_t^{a_n} \in S_f,\ \Delta f_t^{a_n} \approx \widehat{\Delta f}_n,$$

a *false positive*, if the change is similar to the change expected from another effect:

$$\exists\ n \in [1,N],\ \exists\ t \in [0,T\text{-}1],\ \Delta f_t^{a_n} \in S_f,\ \Delta f_t^{a_n} \neq \widehat{\Delta f}_n$$

or a *failure*, if there is no change in the position of the object:

$$\forall\ t \in [0,T\text{-}1],\ \Delta f_t^{a_n} \notin S_f$$

The score, ψ_i, is computed as:

$$\psi_i = (\sum_{e \in E} res_e * W_{res_e}) * N$$

where res_e is the result obtained after the execution of an action reproducing an effect, W represents the predefined weight associated to a result (based on experience), and N represents the total number of given effects to reproduce.

The actions are inferred using the skill obtained in the previous phase. Running a skill, α, to reproduce an effect, e, on an object, f, consists in inferring a sequence of movements:

$$\alpha(e, f) = \{\phi(e, \delta_t)\} = \{\Delta x_t\}, \tag{5.2}$$

5.3 Experimental Framework

Two sets of experiments are executed to asses A^{2}L, using a robotic arm (Section 5.3.1) and a Baxter robot (Section 5.3.2), respectively. The objective of these experiments is the acquisition by a robot of a dataset of interactions to build skills *pushing* a box in different directions. Although a robot endowed with A^{2}L can autonomously learn to interact with the environment to reproduce an effect, the execution of the method requires some *a priori* knowledge. Each set is composed of two experiments: In Experiment 1 a predefined dataset of interactions, reproducing all the given effects, is available. Therefore, only the *skill building process* is executed. Conversely, in Experiment 2 the dataset of interactions is built in different iterations of the method, and thus both the *iterative process* and the *skill building process* are executed. Table 5.1 shows the *a priori* knowldge available in each experiment.

- *List of effects to reproduce:* the experiments are validated based on the reproduction of the *pushing* effects.

- *Constrained movements:* the exploration of the environment is performed using random actions. In order to increase the number of contacts the space to be explored by the robot is constrained to two dimensions. Besides, the movements are discrete, with a fix distance of 5 centimeters and as possible orientations moving to the *right*, to the *left*, *far* from the robot, *close* to it, and the diagonals, i.e. *right-far*, *right-close*, *left-far* and *left-close*. Thus, the action generator infers one of these movements, which are vectors with fix *distance* and *orientation*.

- *Discretization configuration:* the movement and contextual states are discretized because of the capacity of A^{2}L to generate actions based on low-level contextual states and high-level contextual states. To that end, in all the experiments a discretization configuration is available. This configuration is empirically designed, composed of the *distance* and *orientation* between one of the robot's end-effector and the object to interact with. In this case, the *distance* has a range of 0.5 meters, and it is divided in 8 sections of the same size. The *orientation* has a range of 360 degrees, and it is also divided in 8 sections of the same size.

- *Predefined dataset of interactions:* Experiment 1 is aimed at validating the approach with an ideal dataset of interactions reproducing all the effects, whereas Experiment 2 is aimed at validating the approach with a self generated dataset.

- *Predefined BN structure:* There are two versions of each experiment: (i) a version in which a *hard-coded* structure of the BN is available, representing the conditional dependencies of the random variables of the action generator, i.e. the relation between, effect, movement and relation state; (ii) another one in which structure learning methods try to identify the structure. In both cases the corresponding CPDs are learned using a Maximum Likelihood estimator, i.e. without *a priori*. The Bayesian learning is computed using the aGrUM library[1] (Gonzales et al., 2017). In these experiments the robotic arm can only *push* a box. Therefore, the relevant robot-object relation states for this action are the *distance* and the *orientation* between the arm's end-effector and the box.

5.3.1 Simulated Robotic Arm

Experiment 1 and *Experiment 2* have been carried out in order to assess A^{2}L in a virtual setup, implemented in Python. The experiments are executed by a simulated robotic arm. The results obtained in each experiment are analyzed in Section 5.3.1.1.

[1] https://forge.lip6.fr/projects/agrum/wiki.

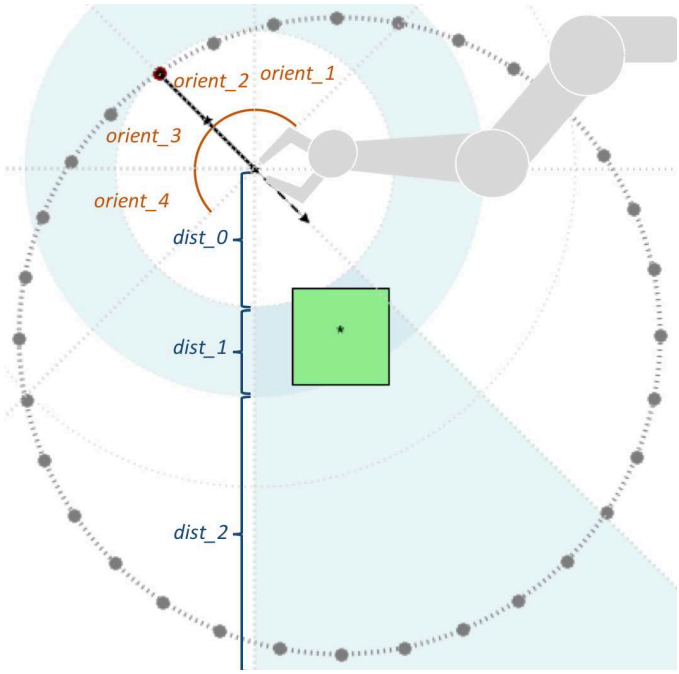


Figure 5.3: Example of the setup used in the experiments. The box is also represented with a position, i.e. its center. The grey points represent initial positions of the arm. The clear blue areas represent both the orientation, i.e. the triangle, and the distance, i.e. the circle, of the box w.r.t. the end-effector. An example of distance and orientation discretizations are also represented, in blue and brown respectively.

Table 5.1: *A priori* knowledge available in each experiment.

	List of effects	Constrained movements	Discretization configuration	Predefined interactions	Predefined structure
Experim. 1	X	X	X	X	X
	X	X	X	X	
Experim. 2	X	X	X		X
	X	X	X		

Experimental Setup

All the experiments are carried out in the same setup. This setup is a two-dimensional (2D) squared working space of 2 x 2 units, with range [-1, 1] in each axis (Fig. 5.3). In the center of the space there is a box, of side size 0.3 units. A simulated camera is located on top of the box, capturing the whole working space. 32 positions around the box, describing a circle, with a distance of 11.25 degrees among them, represent the initial positions of the virtual robotic end-effector during

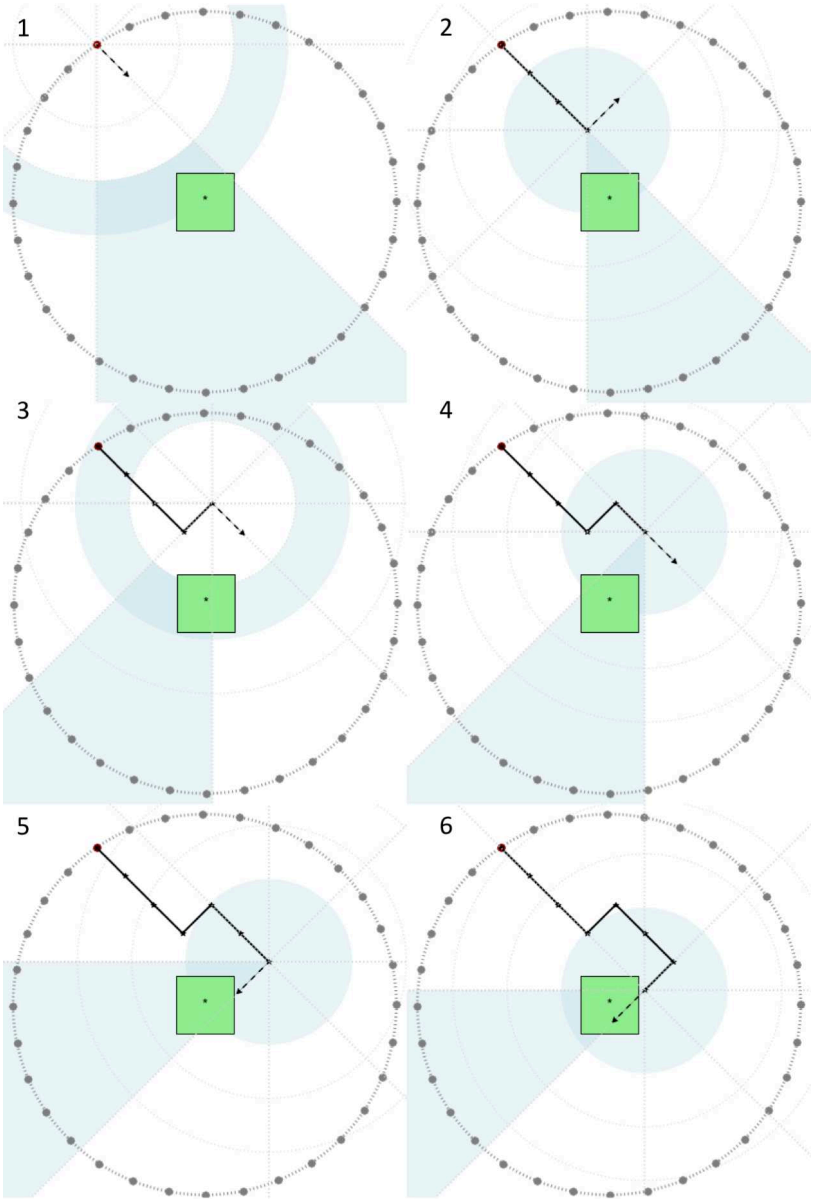


Figure 5.4: Some sequential steps of an inferred trajectory performing the *pushed_left* effect on the box. At the beginning, the end-effector is located at an initial position (in red). The next movement to execute is computed by the action generator based on the robot-object relation state. In this case, the *distance* and *orientation* between them (darker blue area), e.g. *dist_2* and *orient_7* in step 1, and *dist_1* and *orient_6* in step 2. This process repeats (from step 1 to 6) until the end-effector touches the box. A related video is available online: https://www.youtube.com/playlist?list=PL2drYAFCMtzcZ_RlfiFr2AWtcvgc1qR8o.

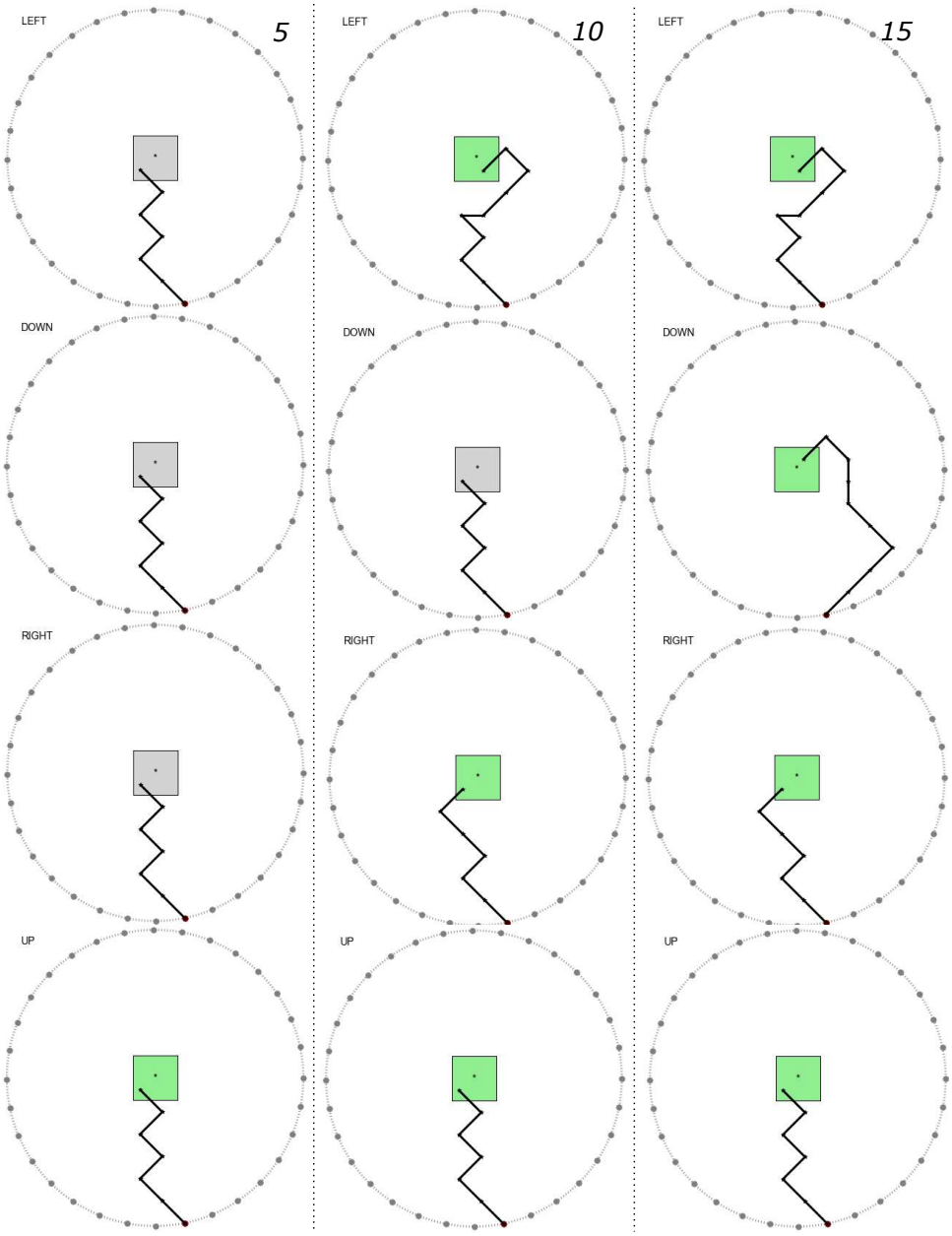


Figure 5.5: Example of skill building along the iterative process of the method to reproduce the four available effects from an initial position. The actions in the figure have been inferred with an action generator learned based on a self-generated dataset of interactions, using *hill climbing* and AIC to learn the conditional dependencies among effect, movement and relation state. After five iterations (left) of the process only one trajectory can be inferred, because the dependencies have not been correctly identified, mostly producing false-positive results (grey box). In the tenth iteration (middle) new actions have been inferred, reproducing successful results (green box) for the *pushed_left* and *pushed_right* effects. Finally, in the fifteenth iteration (right), also the most complex *pushed_down* effect has been reproduced.

the experiments.

The robotic arm is represented as a 2D coordinate in the working space, and thus neither its kinematics nor its DoF are relevant for the experiment. The box is represented as the coordinate (0,0), with a side size of 0.3 units. The only perceptual representation of the setup available for the robot is the position of the box. At the beginning of each iteration of the method, the box is relocated at its initial position.

Discrete effects

A contact between the arm and the box occurs when the position representing the arm is within the area represented by the box. A contact produces a displacement, of a fix distance, of the position of the box. Only four displacements are possible: the box moves to the *left* if it the trajectory executed by the arm intersects with the right side of the box; to the *right* if it intersects with the left side; *up* if it intersects with the bottom side; and *down* if it intersects with the top side. Therefore, only four effects are available: pushed_left, pushed_right, pushed_up and pushed_down, respectively.

Validation

The weights chosen for the skill validation must reflect than a *successful* result of a trajectory is better than a *false_positive*, and much better than a *failure*. Choosing different sets of values based on this principle did not seem to have a relevant impact in the behavior of the method. Based on experience we have selected a value of 8 for $W_{success}$, 2 for $W_{false\_positive}$, and 1 for $W_{failure}$. As the number of effects does not change along the experiments, the computation of the score (Equation 5.2.4) only depends on the results of the trajectories. In each iteration of the method the robot infers 128 trajectories (32 initial positions * 4 effects) trying to reproduce the effects. Therefore, the maximum possible score is 128 * 8. However, the value of the score is normalized between 0 and 100.

Experiment 1

A predefined dataset of interactions is available, generated by trajectories reproducing all the effects, interacting with the four sides of the box from each initial position of the robotic arm. This dataset is used to learn an action generator by the skill builder, which generates trajectories from the 32 initial positions of the arm. It is expected to obtain very high success ratios, and thus very high validation scores.

Experiment 2

In this experiment both processes of A^2L, the iterative process and the skill building, are executed. The robotic arm explores the working space executing random

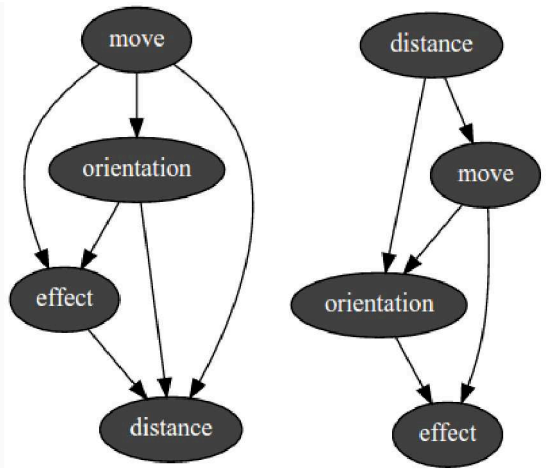


Figure 5.6: Examples of BN learned using *hill climbing* and the AIC score, for Experiment 1 (left) and Experiment 2 (right). In both cases the conditional dependencies between the movement and the other random variables have been properly identified.

Table 5.2: Results of the two versions of Experiment 1 and 2. On the left, using the *hard-coded BN structure*, and on the right, learning the structure using the *hill climbing* method. S stands for trajectories producing *successful* results, FP stands for trajectories producing *false positives*, F stands for trajectories producing *failures*, Sc stands for the normalized score, and ΔSc stands for the variation of the score between the initial and the final iteration. As there it no iterative process in Experiment 1, this variation does not exist, being represented as -.

	Hard-coded					Hill climbing				
	S	FP	F	Sc	ΔSc	S	FP	P	Sc	ΔSc
Experiment 1	115	13	0	92.38	-	112	16	0	90.06	-
Experiment 2	97	31	0	83.57	40.07	87	41	0	78.32	36.6

trajectories, generating a dataset of interactions. The structure of the action generator learned based on this dataset is inferred using *hill climbing* (Chickering et al., 1995) with the AIC score, without a priori information about the structure.

5.3.1.1 Experimental Results

The results obtained for the experiments are available in Table 5.2. For both experiments, around 100000 tuples composed of an effect, a movement and a robot-object relation state have been generated.

Experiment 1: as expected, the action generator built based on the predefined dataset of interactions reproduced most of the effects for both learning methods. The version using the *hard-coded* structure obtained slightly better results than the

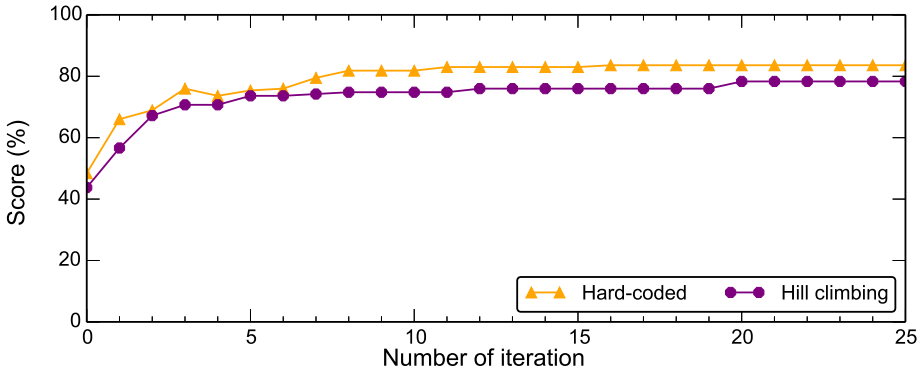


Figure 5.7: Evolution of the results obtained in the *Experiment 2* along the iterations. The scores are computed as the mean values of 5 runs of the experiment, where each run executes 25 iterations.

version learning the structure using *hill climbing*. These results show the capacity of A^{2}L to build a skill given interactions properly representing the reproduction of the effects. The *false positive* results identified in Table 5.2 are the result of an involuntary contact of the arm's end-effector with the box while executing a move turning around it, because of the use of discrete movements. The learned BN structure is depicted at the left of Figure 5.6.

Experiment 2: using A^{2}L the robotic arm has reproduced most of the effects (see Figure 5.7). In this case, the results obtained by the version of the experiment using the *hard-coded* structure are better than those obtained by *hill climbing*. Both methods are able to reproduce half of the effects using the interactions result of the initial exploration before the execution of the method. Later, through the iterative process, these methods generate new interactions allowing the robot to reproduce other effects, improving the score around a 40 per cent, i.e. ΔSc. The improvement of the score grows rapidly during the first iterations, and reaches a *plateau* afterwards. These plateaus result from the size of the dataset of interactions. After several iterations the dataset has thousands of interactions. It would be necessary to generate many new interactions to modify the inferred trajectory to reproduce the effect from an initial position. This is also the reason of the generated *false positives*, because of the high complexity of generating randomly some example trajectories from each initial position, e.g. the trajectory around the box needed for the *pushed_right* effect from the initial position at the right side of the circle. Therefore, the number of examples related to those complex trajectories is low, being poorly reproduced by the action generator. The BN structure learned by *hill climbing* is depicted at the right side of Figure 5.6.

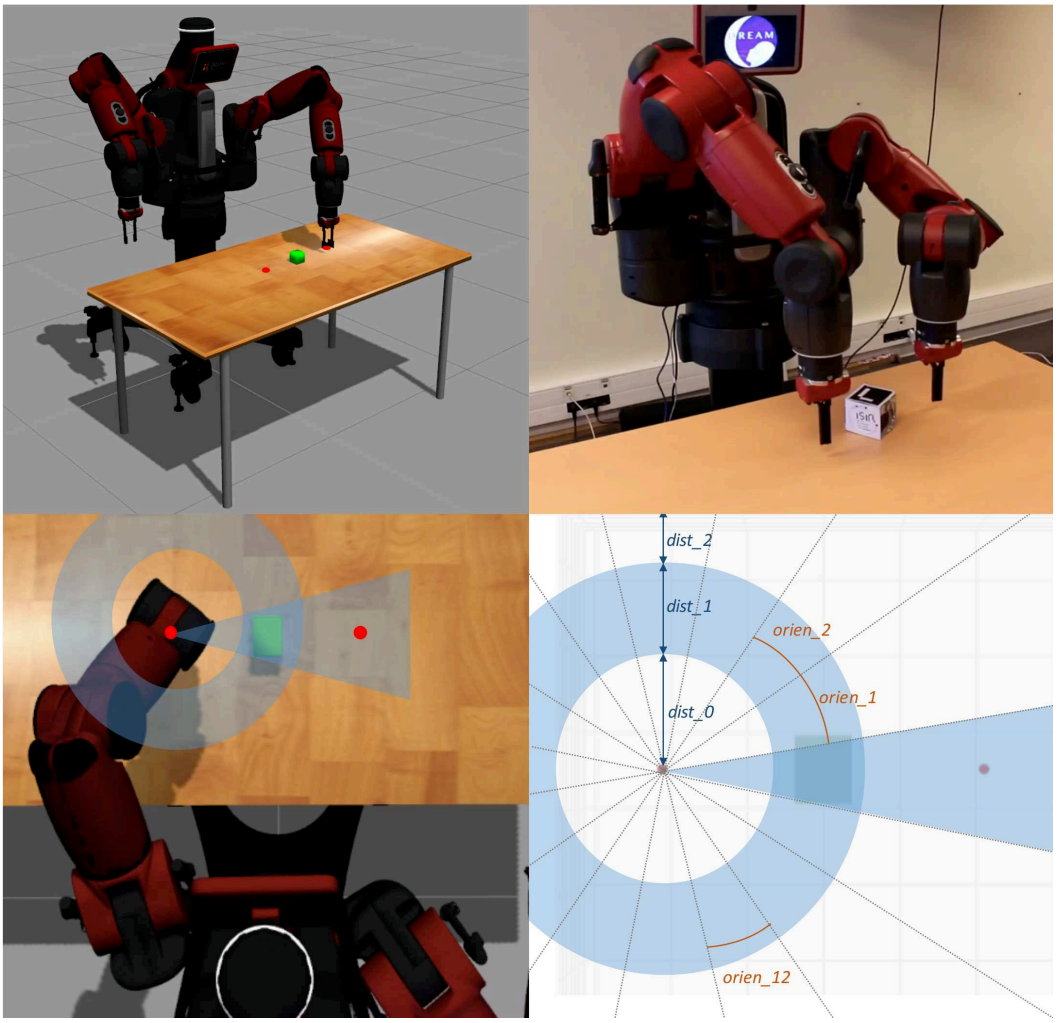


Figure 5.8: In the top-left corner, a simulated Baxter robot in the tabletop setup used for the experiments. This setup is composed of a box on a table in front of the robot. In the top-right corner, a physical Baxter robot in a similar tabletop setup used for to asses the generated repertoire of states performing a simple test. In the bottom-left corner, top view of the simulated setup. In the bottom-right corner, the *virtual setup* emulating the top view of the simulated setup. The red circles represent both initial positions of the end-effector from where the actions are inferred and executed. The blue triangle and torus represent, respectively, the *orientation* and *distance* from the left end-effector to the box. In the current work, the *distance* is discretized in three sections (dark blue text), and the *orientation* in 16 sections (brown text). For instance, in the example the box is at *distance 1* and *orientation 0* of the end-effector.

5.3.2 Simulated Baxter Robot

Experiment 1 and Experiment 2 are executed to assess the generation and validation of a dataset of interactions by a simulated Baxter robot[2] in a setup with ODE as physics engine[3]. The main features of the robot w.r.t the experiments are that the robot has two arms of 7 DoF each one of them. Similarly to the robotic arm experiments, the goal is pushing a box in different positions. In this case, the box can be located in different initial positions, and thus the actions generated by the skills must be adapted to its position. Besides, we show that the generated dataset of interactions generated in the Experiment 2 can be used by a physical Baxter, in a similar setup, continuously *pushing* a box on a table. This section finishes with an analysis of the obtained results.

ROS Indigo Igloo[4] was used to manage the robot. The simulation has been executed in Gazebo 1.9[5]. We developed a library to manage the kinematics of the Baxter robot[6], based on MoveIt[7] and the own robot's kinematics library. During the execution of the task, the position of the physical box is acquired using a QR code (see top-right corner of Figure 5.8).

5.3.2.1 Experimental Framework

Experimental Setup

The scenario of the experiment simulates a three-dimensional (3D) Cartesian table-top setup composed by a table and a box. The robot is located in front of the table (Fig. 5.8). The dimensions of the box are 7 x 8.5 x 8 centimeters (cm) of width, length and height, respectively. The position of the box at the beginning of the experiment is at 65 cm in front of the robot, and 10 cm to the left. The reference frame of the setup is located at the base of the robot, and thus the perceptions of the robot are relative to itself. The robot only perceives the position of the box[8]. The position of the box is located at the center of the object, which can change during the experiment. However, in order to allow the method to infer actions that the robot can execute, if the box is moved more than 10 centimeters away from its initial position it is automatically relocated around the initial position.

In simulation, the position of the box is directly provided by Gazebo. In the physical robot, the location of the box is provided by a QR code obtained by a RGB-D camera.

The robot moves its left end-effector to interact with the box starting from two initial positions. These initial positions are located at 20 cm to the left of the box

[2] http://www.rethinkrobotics.com/baxter/
[3] http://www.ode.org/
[4] http://wiki.ros.org/indigo
[5] http://gazebosim.org/
[6] https://github.com/cmaestre/baxter_kinematics
[7] http://moveit.ros.org/
[8] In the current work, the low-level states of an object are only represented by its position, not relying either on its orientation or its shape, as in young infants (Rosenbaum, 2010).

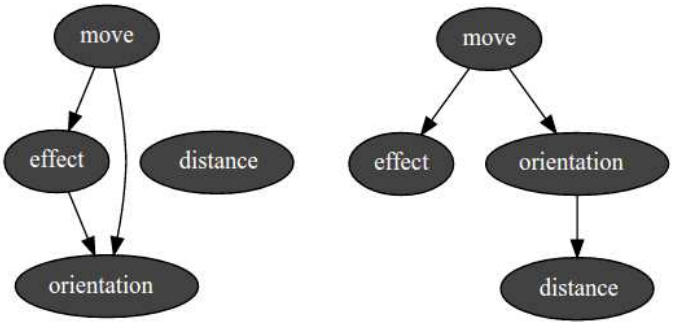


Figure 5.9: Results of the BN structure learning of the K2 method, for Experiment 1 (left) and Experiment 2 (right). In both cases K2 has identified the direct dependency of the *movement* with the *effect* and the *orientation*, but it is missing the direct dependency with the *distance*.

position and 20 cm to the right, respectively. Because of the exploration of the environment performed using random actions, these are constrained to a 2D space, i.e. with a constant Z value. Examples of movements are shown on Fig. 5.10.

Discrete effects

The robot can reproduce eight effects categorized in two types: effects in which the box is mainly moved in one axis, i.e. far from the robot (*pushed_far*), close to it (*pushed_close*), to the left (*pushed_left*), to the right (*pushed_right*; and effects in which the box is moved in both axis, i.e. *pushed_far_left*, *pushed_close_left*, *pushed_close_right* and *pushed_far_right*. A displacement of the box is identified as an effect, or it is discarded otherwise. A displacement is identified as an effect of the first type if the displacement in one axis is at least three times bigger than in the other axis. To be identified as an effect of the second group the displacement in one axis is at least bigger than the half of the displacement in the other axis. The robot can execute a maximum of 16 actions (2 initial positions * 8 effects) during each iteration of the experiment.

Validation

In this case, the weights chosen have a value of 4 for $W_{success}$, 2 for $W_{false\_positive}$, and 1 for $W_{failure}$. The only difference w.r.t. the weights chosen in the robotic arm experiments is the selection of a lower value for the result of a *successful* action. This lower value gives more relevance to the *false positive* results in the final score. Nevertheless, the selection of different values does not provide significant changes in the results. Successfully inferring the 16 possible trajectories would produce the highest possible score with value 64. The score is normalized between 0 and 100.

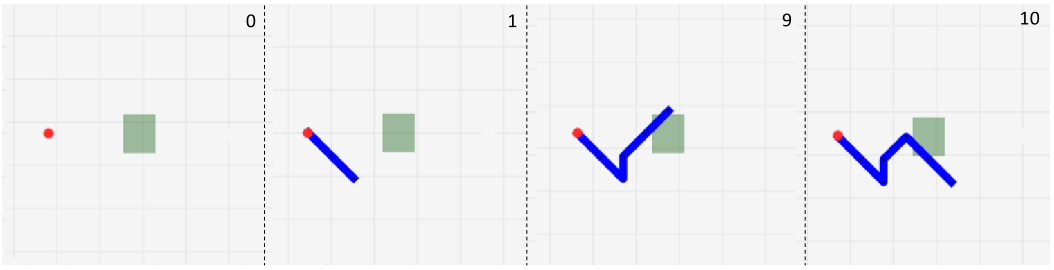


Figure 5.10: Simulation of the evolution of an action inferred by the action generator (blue line) reproducing the *pushed_close_right* effect from an initial position (red circle). In this case, the action generator is learned using the *hard-coded* structure. In each figure the box is represented in the center of it as a green square, the numbers show the current iteration of the method, and the robot is situated at the bottom of each figure. The action generator evolves from not inferring any movement (iteration 0) to infer an action producing the desired effect (iteration 10), going through intermediary step, as inferring an action not touching the box (iteration 1) or reproducing another effect (iteration 9).

Experiment 1

A predefined dataset of interactions is available, representing trajectories reproducing the effects from all the initial positions of the end-effector. For instance, to reproduce the *pushed_left* effect the robot's end-effector, from each initial position, moves at a position 15 centimeters to the right of the center of the box, and then it moves 20 centimeters to the left). A skill is built based on the dataset. It is expected to obtain very high success ratios, and thus very high scores.

Experiment 2

In this experiment the dataset of interactions is generated running the iterative process. An initial goal-free exploration is executed by the Baxter robot, generating several interactions between the robot and the box, stored into an initial repertoire of states. This dataset is increased and evaluated at each iteration of the iterative process. An illustrating video of the experiment is available[9].

In the current experiment, first, a *hard-coded* structure of the BN representing the conditional dependencies of the *movement*, with the *effect* and the *relation state* is given. Second, in this case the structure is learned using *K2* (Cooper and Herskovits, 1992) with a K2 score. This method needs as input an order of the random variables of the BN for the structure learning, in this case selected based on experience (further details are available in Section 3.2). *Hill climbing* was discarded as structure learning method because the number of interactions generated is much lower than in the robotic arm experiments, and thus it is necessary to drive the structure learning with some *a priori* knowledge. The CPDs are also learned using

[9] https://youtu.be/5a02TkaaaRk

Table 5.3: Results of the experiments. *S* stands for trajectories producing *successful* results, *FP* stands for trajectories producing *false positives*, *F* stands for trajectories producing *failures*, *Sc* stands for the normalized score, and ΔSc stands for the variation of the score between the initial and the final iteration.

	Hard-coded					K2				
	S	FP	F	Sc	ΔSc	S	FP	P	Sc	ΔSc
Experiment 1	16	0	0	100	-	15	1	0	95.3	-
Experiment 2	8	2	6	53.1	33.1	4	12	0	43.57	43.57

a Maximum Likelihood estimator.

Pushing a Box with a Physical Baxter Robot

In this simple test a physical Baxter robot builds a skill to continuously interact with a box, based on the dataset of interactions generated in Experiment 2 using K2. The interaction is performed using both end-effectors of the robot, located randomly at each side of the box, respectively (see top-right corner of Figure 5.8). In each run of the test the action generator infers a trajectory for each one of the given effects, using each one of the end-effectors (see Figure 5.12). These trajectories are computed based on a mathematical approximation of the execution of each movement. If a trajectory gets *close* to the object position, it is considered as touching it. For each of the trajectories *touching* the object, the mean value of the posterior probability of each movement is computed. That trajectory with higher probability is executed.

5.3.2.2 Experimental Results

The results obtained for the experiments are available in the Table 5.3

Experiment 1: based on the predefined interactions the robot reproduces all the effects for the hand-coded method and most of them for K2. These results reflect that, given a dataset of interactions representing the reproduction of the effects, the method is able to build a skill reproducing most of the effects.

Experiment 2: similarly to the results obtained in the robotic arm experiment the robot has increased around 40 per cent the number of effects reproduced throughout the running of the iterative process with both available learning methods (Fig. 5.11) showing the capability of the method to generate the dataset of interactions. However, in this case the number of effects reproduced after the initial exploration of the environment before the use of A^2L is very low for the *hard-coded* method, and no effect is identified for K2. Therefore, the size of the initial dataset is very low (see blue lines) and after 25 iterations of the method is still low. Using the *hard-coded* structure half of the effects are reproduced, meaning that interactions reproducing the other half of the effects are missing or there is a low number of

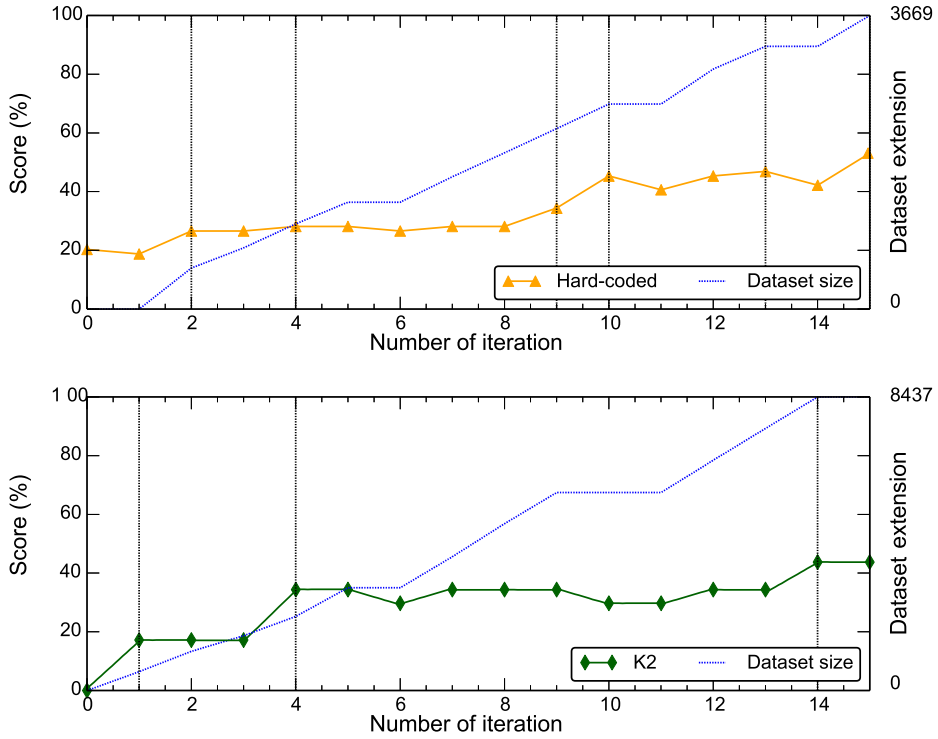


Figure 5.11: Results of the *Experiment 2* using the *hard-coded* structure (at the top) and learning the structure with K2 (at the bottom). These results show the evolution of the number of effects reproduced along 15 iterations. The score is represented as yellow and green lines using the *hard-coded* and the K2 structures, respectively. The size of the repertoire of movements and contextual states is represented as a blue line. The dotted vertical lines represent iterations in which the score improved w.r.t. the previous iteration.

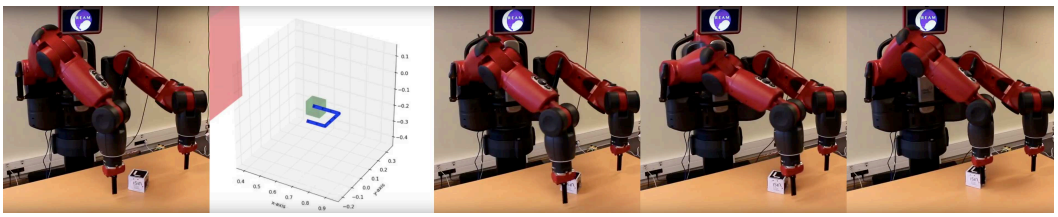


Figure 5.12: Sequence (from left to right) of the steps executed in an iteration of the test with the physical Baxter. First, the position of the object and of each end-effector is obtained. Then, the trajectory with a higher probability is selected (represented in a virtual setup). Finally, the movements composing the trajectory are executed.

them. Using the structure learned using K2 only 4 effects are reproduced, although it produces a high number of *false positive* results. This means that the structure of the BN has not identified the right dependencies among the movement, effect and the relation state (on the right of Fig. 5.9). Therefore, more interactions are needed to both, identify the right structure of the BN and to reproduce the complete set of effects.

Test with the physical Baxter robot: the robot interacts with the box using the repertoire of states generated in the previous experiment until the robot pushes the box out of the table. This result proves that the repertoire of states generated by the simulated Baxter can be directly executed by a physical Baxter if their surrounding environments share the same features. The execution of a trajectory is shown in Figure 5.11. An online video is available[10] showing the interactions of the robot with the box.

5.4 Conclusions and Open Questions

In this chapter we have analyzed the capability of A^{2}L to generate and validate interactions, and to build skills used by a robot to reproduce effects on an object. With respect to NovEB (see Chapter 4) more *a priori* knowledge about the environment is provided, i.e. the object position, but the method explores the environment with very low *a priori* knowledge using random variables. Moreover, explorations are constrained to two Cartesian dimensions using discrete movements to increase the numbers in robot-object interactions generated.

In order to assess the method, two sets of experiments have been carried out. In the first set, a simulated robotic arm in a virtual setup performs the experiments. The execution of the actions and their effects in the virtual setup are based on mathematical approximations because of the lack of a physics engine. The second set of experiments is performed by a simulated Baxter robot in a setup with physics engine. Besides, interactions generated in these experiments are directly used by a physical Baxter robot to interact with a box. In both sets of experiments the environments are static, that is, only the robot can move the box. However, in the robotic arm experiments the position of the box is reinitialized after the execution of an action, whereas in the experiments with the Baxter it is enough if the box is within reach of the robot.

Our method mainly relies on a discretization configuration and in the conditional dependencies, i.e. the BN structure, among an *effect*, a *movement* and the robot-object relation state, to run. On the one hand, the obtained results show that given a dataset of interactions properly representing effects on an object, a correct discretization and the proper BN structure the method is able to reproduce most of the effects. The better the trajectories of the interactions produce the effects are, the better the inferred actions are reproducing the effects. On the other hand, it requires a very high number of interactions in order to identify the conditional

[10] https://youtu.be/RKJRXmRTHDc

dependencies represented in the BN of the action generator. In the simulated Baxter experiment the use of a structure learning method without *a priori* knowledge about these dependencies was directly discarded, because of not being able to reproduce any effect. The method was able to reproduce half of the possible effects using K2 with some insights about the conditional dependencies. Possibly with a higher number of interactions K2 could improve its results, although the generation of those interactions would last for a very long period of time.

A similar temporal constraint is the reason to use discrete movements in a 2D environment. This approach can limit the reuse of the inferred actions to execute some tasks in posterior higher-level stages. Chapter 6 extends the skill builder to create skills generating continuous actions adapting in a close loop to the object position, i.e. actions are adapted the object position even it this changes during the execution of the action. Moreover, the use of random actions to drive robot motion generates poor actions. For example, while exploring an environment. We suggest the use of intrinsic motivations to drive the unsupervised explorations, as described in Section 2.1.1 and implemented in Chapter 4; or building the repertoire of skills directly demonstrating to the robot the right actions to reproduce an effect, as also described in Section 2.1.1 and implemented in Chapter 6.

It is relevant to mention that the use of relative positions for the object, i.e. the robot-object relation states, make the method very robust to different object positions.

The use of more realistic scenarios, with daily objects represented with both low-level and high-level states, and different actions, as *grasp*, would challenge the capabilities of the method. Also, the setup of the experiments could be extended to more than one object. These challenges have been also addressed in Chapter 6.

Another improvement to A^2L is related to the autonomous generation of *a priori* information needed to execute, currently externally provided, i.e. the set of effects to reproduce and the discretization configuration. This information could have been computed in an unsupervised fashion within the iterative process.

The use of more realistic scenarios, with daily objects represented with both low-level and high-level states, and different actions, as grasp, would challenge the capabilities of the method. Also, the setup of the experiments could be extended to more than one object. These challenges have been also addressed in Chapter 6.

Chapter 6

Online Generation of Actions Adapted to Contextual States

The results and text of this chapter have been presented in the following article.

- **Maestre, C.**, Mukhtar, G., Gonzales, C., and Doncieux, S. (2017, October). Context-Based Generation of Continuous Actions to Reproduce Effects on Objects. In the Third International Workshop on Intrinsically Motivated Open-ended Learning (IMOL). Referenced in the bibliography as Maestre et al. (2017a).

Contents

6.1 Introduction

In Chapter 5 we demonstrated how a robot can build skills through interactions with its environment to infer actions reproducing effects on objects (see Definitions 2, 4 and 5). To that end, we proposed a method, named Adaptive Affordance Learning (A^2L), which endows a robot with the capacity to generate a dataset of robot-object interactions used to build the skill. The method presented two main processes: (i) the acquisition of interactions when needed, called *Iterative Interaction Acquisition*

and Validation, and (ii) the adaption of the robot's actions to the object position, called *Skill Building*. The method was validated on a physical Baxter robot in a tabletop setup, generating a dataset of interactions *pushing* a box in different directions. However, in those experiments the actions generated were discrete and constrained to a 2D space; and the setup was static, i.e. the position of the box only changed when the robot touched it.

This chapter is focused on improving the generation of skills presented in Chapter 5, addressing the different identified constraints, in order to scale up their use to our daily environments. In these environments, the skills must be robust to spatio-temporal perturbations. Namely, the position of an object can change independently from the robot's actions, and an action can be modified during its execution. In order to address this feature, in the current work A^2L builds adaptive skills using a Dynamical System (DS) called *diffeomorphism* (explained in Section 3). The DS generates a *vector field*, in which a vector represents a *movement* of the end-effector (Schaal, 1999; Ijspeert et al., 2002) (see Definition 7). Therefore, in a closed loop, the action generator of the skill infers movements adapted to the object position, i.e. its low-level contextual states (see Definitions 6, 1 and 3). Moreover, the inference of movements by a vector field removes the necessity to use discrete movements, and the constraints applied to the robot environment, which is now a 3D Cartesian space.

In Chapter 5 the dataset of interactions was generated through an iterative process. Although the method was able to reproduce most of the effects in a virtual setup, it was not able to reach a high performance in a simulated tabletop setup. Besides, the acquired information was very noisy because of the use of random actions. This chapter is focused on building skills directly in physical environments, which are already noisy due to the lack of accuracy of the robot's sensors. In order to avoid adding more noisy information through an exploration of the environment, different robot-object interactions are directly demonstrated to the robot by an external agent. For example, the experiment designer shows how to produce an effect on an object through a kinesthetic demonstration moving the robot's end-effector, i.e. learning from demonstration (LfD) (Billard and Calinon (2016)). Moreover, in this chapter each skill is built from an individual dataset of interactions, including one or more demonstrations of interactions with an object producing the same effect, i.e. a skill is built using *batch learning*.

Our method mainly relies on the conditional dependencies among the *effect* to reproduce, a *context* at an instant of time and the next *movement* to perform, and on a discretization of the context. Regarding the conditional dependencies, the action generator is a Bayesian Network (BN) (Pearl, 1988). Therefore, the structure of the BN represents these dependencies. In the previous chapter, the context was represented by the end-effector and object positions, and a *robot-object relation state* was computed based on these positions. However, in order to better represent our daily environments, in this chapter the context is composed of low-level and high-level states (see Description 9). Therefore, the structure of the BN must represent the conditional dependencies among an *effect*, a *robot-object relation*

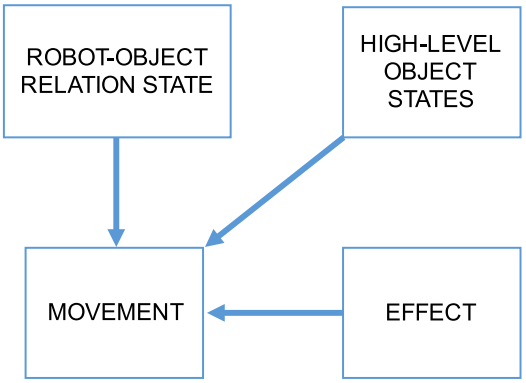


Figure 6.1: Conditional dependencies of the *hard-coded* structure allowing the action generator to infer the next *movement* based on the *effect* and the *low-level and high-level relation states*.

state and *high-level context states* at an instant of time, and the next *movement* to perform (see Figure 6.1).

Moreover, as the environments are not constrained, the robot-object relation state is extended to represent the *distance*, *orientation* and *inclination* of the position of an object w.r.t. the end-effector position (depicted in Figure 6.5). The object position represents the center of gravity of the object, usually located at the center of the object; whereas the position of the end-effector represents the position of the tip of their fingers when the end-effector is closed. The selection of a different end-effector position would not affect the performance of A^2L, because the method learns to infer movements based on distances acquired during the demonstrations. For example, from a demonstration *grasping* an object the method learns to close the fingers when a distance from the end-effector position to the object position is reached. This distance would be smaller if the end-effector position is located at the tip of the fingers than if its located at the base of the end-effector.

Low-level contextual states have continuous values, e.g. an object Euclidean position. High-level contextual states may have continuous values , e.g. an object shape (Ugur and Piater, 2015a), but they usually have discrete values, e.g. an object color (Montesano et al., 2008). A BN needs to handle both low-level and high-level contextual states, that is, it needs to handle continuous and discrete values. Unfortunately, as mentioned in Chapter 1, BN can only handle variables with discrete values. *Effects* are already discrete. Therefore, *robot-object relation states*, *high-level context states* and *movements* are discretized. In the previous chapter, a discretization configuration was available, empirically created based on experience. In the current work, a study is performed in order to select the best parametrization to build a *push to the right* and a *grasp* skill to reproduce an effect. This study is composed of the identification of the most adequate discretization configuration and the structure of the BN to use. Then, the selected parametrization is used to build a set of diverse skills, which are validated in several experiments performing tasks with different objects. The assessment of the built skills is directly

performed by the physical Baxter in Section 6.5.

6.2 Extended formalization

In Chapter 5 a detailed formalization of A^{2}L was provided. This section redefines basic aspects common to the previous formalization, required to explain the structure of a skill (Section 6.3) and the upgraded method to learn skills and execute them (Section 6.4) .

Skills are based on interactions of the robot with an object to reproduce an effect. An interaction represents (i) a trajectory of the robot's end-effector and (ii) the robot's context during the execution of the trajectory. A trajectory is stored as a sequence of Cartesian positions, i.e. *waypoints*, representing continuous positions of the end-effector, x_t, and gripper states, g_t, at certain instant of time. The robot's context is composed of low-level and high-level contextual states of the object at certain instant of time, f_t. In the current work, the low-level object states correspond to the continuous object's position, y_t. These states and the end-effector position are acquired at certain instants of time during the execution of the trajectory. Therefore, an interaction between the robot's end-effector and the object when executing a trajectory, Υ_{xgyc}, is represented as:

$$\begin{aligned} x_t &= \text{end effector position} \\ g_t &= \text{gripper state} \\ y_t &= \text{object position} \\ h_t &= \text{high-level object states} \\ \Upsilon_{xgyc} &= \{(x_0, g_0, y_0, h_0), ..., (x_T, g_T, y_T, h_T)\} \end{aligned} \tag{6.1}$$

An effect is defined as an expected variation of the object contextual states, $\widehat{\Lambda f}$, and it is associated to a label, e. In the current work the expected variation can be related to either a variation of the object position or a variation of the high-level object states:

$$e \equiv \widehat{\Lambda f} = y_t - y_{t-1} \vee h_t - h_{t-1}$$

where the subscript t represents an instant of time.

The dataset of interactions containing the demonstrations performed by an external agent, $\mathbb{D}$, is composed of an effect and one or more interactions producing the effect:

$$\mathbb{D} = (e, \{\Upsilon^k_{xgyh}\}) \tag{6.2}$$

where k represents one of the K interactions available.

Once the skill is available it can be used to generate actions to reproduce effects on objects. An action, a_e, is a sequence of movements reproducing an effect. A movement, $(\Lambda x_t, \Lambda g_t)$, consists in a displacement of the robot's end-effector and a change of the gripper state between two subsequent instants of time (see Definition

7):

$$\begin{aligned} \Lambda x_t &= x_t - x_{t-1} \\ \Lambda g_t &= g_t - g_{t-1} \\ a_e &= \{(\Lambda x_t, \Lambda g_t)\} \end{aligned}$$

Actions produce changes in the object states (see Definition 5):

$$\Lambda f_t^{a_e} = y_t^{a_e} - y_p^{a_e}$$

where a^e represents an action to reproduce an effect, and t and p represent different instants of time, such that t is posterior to p. An action is a *success*, if after its execution the change produced in the object is equivalent to the desired effect:

$$\Lambda f_t^{a_e} \approx \widehat{\Lambda f_e}$$

or a *failure*, if the change is equivalent to another effect:

$$\Lambda f_t^{a_e} \neq \widehat{\Lambda f_e}$$

or if there is no change in the position of the object:

$$\Lambda f_t^{a_e} = 0$$

6.3 Context-based and Adaptive Skills

6.3.1 Theoretical framework

The skills generated by A^2L must be robust to spatio-temporal perturbations. Therefore, it is necessary to add a dynamical framework providing this capability of adaptation. Warren (2006) defines a theoretical framework describing the interaction of an agent with its environment as a DS, applied in realistic use cases (a detailed explanation is available in Section 3.3.2). We use this framework to upgrade our method to add the adaptation capability.

The most relevant features from this framework, regarding our work, are twofold: first, the description of two levels of analysis for any interaction of the agent with its environment (Figure 6.2). At the *first level of analysis* a *perception-action cycle* represents the global behavior of the interaction (Kugler and Turvey, 1987; Warren, 1988). In a constant loop, the perceptual information gathered by the agent drives the generation of an action. Its execution modifies the environment, generating novel perceptual information. The *second level of analysis* represents a low-dimensional description of the global behavior. This level describes the temporal evolution of a behavior, i.e. an action, called *behavioral dynamics*. The status of an action is described based on the change of few variables, the *behavioral variables*. At this level, a vector field is generated based on one or more effect-oriented trajectories of a demonstrated interaction. The vector field is described by the behavioral variables. The perceptual information of the first level is described using

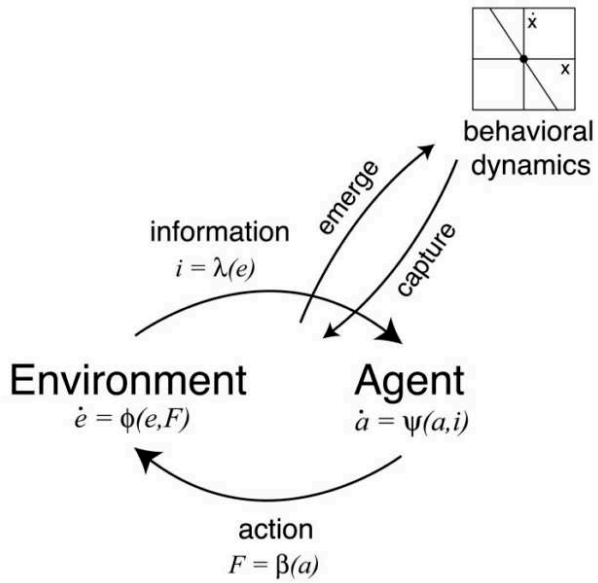


Figure 6.2: Description of the two levels of analysis of a robot interaction with its environment. From Warren (2006). A full explanation is available in Section 3.3.2.

the same variables. Then the perceptual information at a certain instant defines the current action state in the low-level vector field, and the action is computed from that state. Second, at the second level, the effects produced by the demonstrated interactions are *attractors* within the vector fields. Therefore, the generation of an action at the first level is driven by the effect-oriented vector fields, from the current action state. Namely, the vector fields drive the global interactions of the agent. This is important to easily adapt the actions to the environment. For example, if the perceptual information identifies a new object to be avoided during the action to execute, a *repeller* could be added to the vector field to avoid that area of the environment.

Warren applies the theoretical framework to a steering and obstacle avoidance problem, with the aim of predicting routes through complex scenes. Based on his insights, different problems must be addressed at each level of analysis: *"The problem at this level [first level] is to identify the informational variables that are used to guide behavior and to formalize the control laws by which they regulate action. [...] the problem at the second level of analysis is to identify a system of differential equations (i.e., a dynamical system) whose solutions capture the observed behavior"* (Warren et al., 2010, page 1). In order to apply the theoretical framework to the building of skills by A^2L we must also address these problems. Regarding the problem at the first level of analysis, a skill generates continuous actions adapted to an object position to reproduce an effect. As aforementioned, generating an action entails the inference of a sequence of movements of the robot's end-effector. Therefore, the states used to guide behavior are those related to the generation of a movement, i.e. low-level contextual states. In the current work, these states

correspond to the robot-object relation states, i.e. the *distance*, *orientation* and *inclination* between the end-effector and the object. Regarding the problem at the second level of analysis, we must define a dynamical system that, based on one or more trajectories, produces a vector field representing the temporal evolution of the trajectories. In the current chapter a continuous vector field is generated using a *diffeomorphism*, explained in Section 6.4.2. Information from this vector field is extracted and discretized to learn the action generator (a detailed explanation is available in Section 6.4).

6.3.2 Skill structure

Once the previously identified problems have been properly addressed, the theoretical framework can be applied to A^2L in order to build adaptive skills. The structure of a skill is depicted in Figure 6.3. The skill receives information to reproduce an *effect* on an *object*. In Section 3.3.2 the theoretical framework was described for a general interaction of a robot with its environment. Conversely, a skill reproduces a specific effect. And thus its execution has to stop at certain instant of time, providing a *result*, together to a detailed *trace* of the execution. The running of the skill stops either producing a *failure* after a maximum number of iterations, or producing a *successful* result when a condition representing the expected effect is reached:

$$\hat{\Lambda} f = \sum_{i=1}^{t} \hat{\Lambda} f_i$$

where $\hat{\Lambda} f$ represents the variation of the object features representing the effect and the summation represents the accumulated variation of the object features after t iterations.

This condition is integrated into the *perception-action cycle* within the higher level of analysis. At the beginning of the cycle, the condition is evaluated. If the effect has been reproduced the skill execution stops. If not, the object position, y_t, and high-level contextual states, H_t, of the object are acquired (equivalent to the information function, Equation 3.12):

$$y_t, h_t = GetContextStates() \tag{6.3}$$

Then, together to the available robot's state, the robot-object relation state is computed and discretized:

$$\delta_t = ComputeRelationState(x_t, f_t) \tag{6.4}$$

where δ_t represents the discrete robot-object relation state at a certain moment, , i.e. the position of the object with respect to the end-effector, x_t represents the continuous position of the robot's end-effector at a certain moment, f_t represents the object features at a certain moment, and $ComputeRelationState$ is a function computing the continuous robot-object relation state.

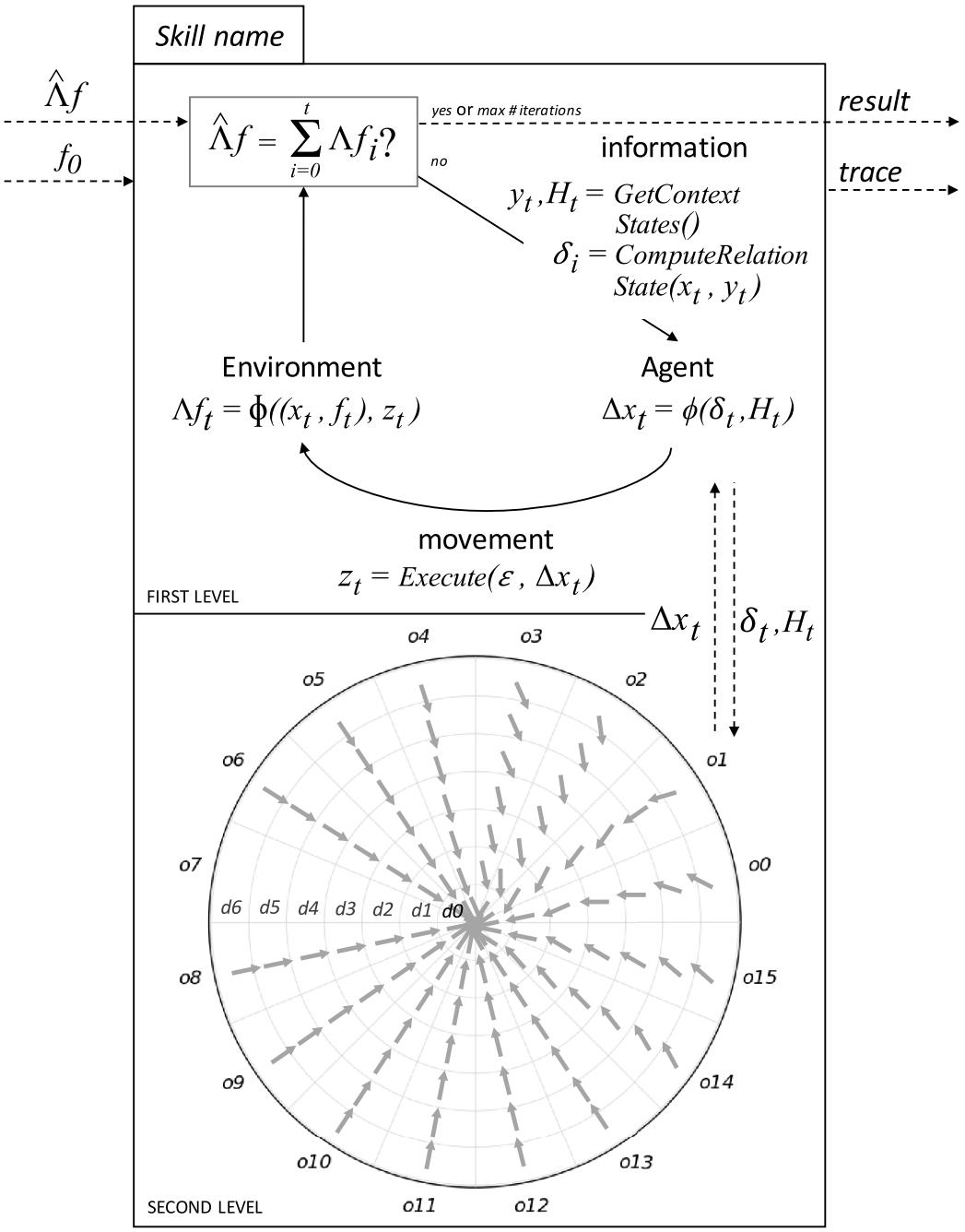


Figure 6.3: Structure of a sensorimotor skill generated by A^{2}L. In order to ease the reading of the figure we remove the step becoming continuous values into discrete ones, and vice versa. Therefore, the functions *GetContextStates* and *ComputeRelationState* return discrete high-level and low-level contextual states, respectively. Besides, the function *Execute* is able to execute a discrete movement.

Once the previous information is available, the continuous next movement to reproduce the effect, Λx_t, is computed. As aforementioned, the movement is not directly computed by a dynamical system, as in Equation 3.10. An action generator, ϕ, computes it based on the available relation state and high-level object states:

$$\Delta x_t = \phi(\delta_t, H_t) \tag{6.5}$$

Then the robot executes the movement with its end-effector, ε, generating a set of forces, z (equivalent to Equation 3.11):

$$z_t = Execute(\varepsilon, \Delta x_t) \tag{6.6}$$

The execution of the movement modifies the position of the end-effector, Λx_t. The forces associated to the movement may modify the object states, Λf_t following the Physical laws of the environment, Φ, if the robot touches the object (equivalent to Equation 3.9):

$$\Lambda f_t = \Phi((x_t, f_t), z_t) \tag{6.7}$$

At the second level of Figure 6.3, an example is depicted of the movements (grey arrows) inferred by the action generator (Equation 6.5) for different discrete values of the *behavioral variables*. In this example these variables only correspond to the relation state, i.e. the position of the object with respect to the end-effector, composed of a *distance*, d_i, and an *orientation*, o_i. The states are represented in polar coordinates used in the discretization process explained in Section 6.4.2.

6.4 Method

This section explains the improvement in the process of A^{2}L to learn and execute skills adapted to the contextual states. A skill is learned from one or more demonstrations of interactions with an object, producing the same effect on it, e.g. *pushing* it to the right. Therefore, a dataset of interactions must be available. These interactions are acquired by an external agent through one or more demonstrations (LfD), and stored into a dataset of interactions, $\mathbb{D}$ (Section 6.4.1).

6.4.1 Initial Interaction Acquisition by Demonstration

The external agent performs the demonstrations manipulating the Baxter robot (LfD). In each demonstration, the external agent grasps the wrist of one of the robot's end-effectors, presses a button located in the end-effector to start the recording of the demonstration, moves the end-effector to perform an action (usually interacting with an object) and releases the button once the trajectory has finished. The external agent presses another button during the execution of the action whether the gripper has to open or close.

The acquired information is continuous, composed of the trajectory, i.e. a sequence of position of the end-effector, and for each position of the end-effector (i)

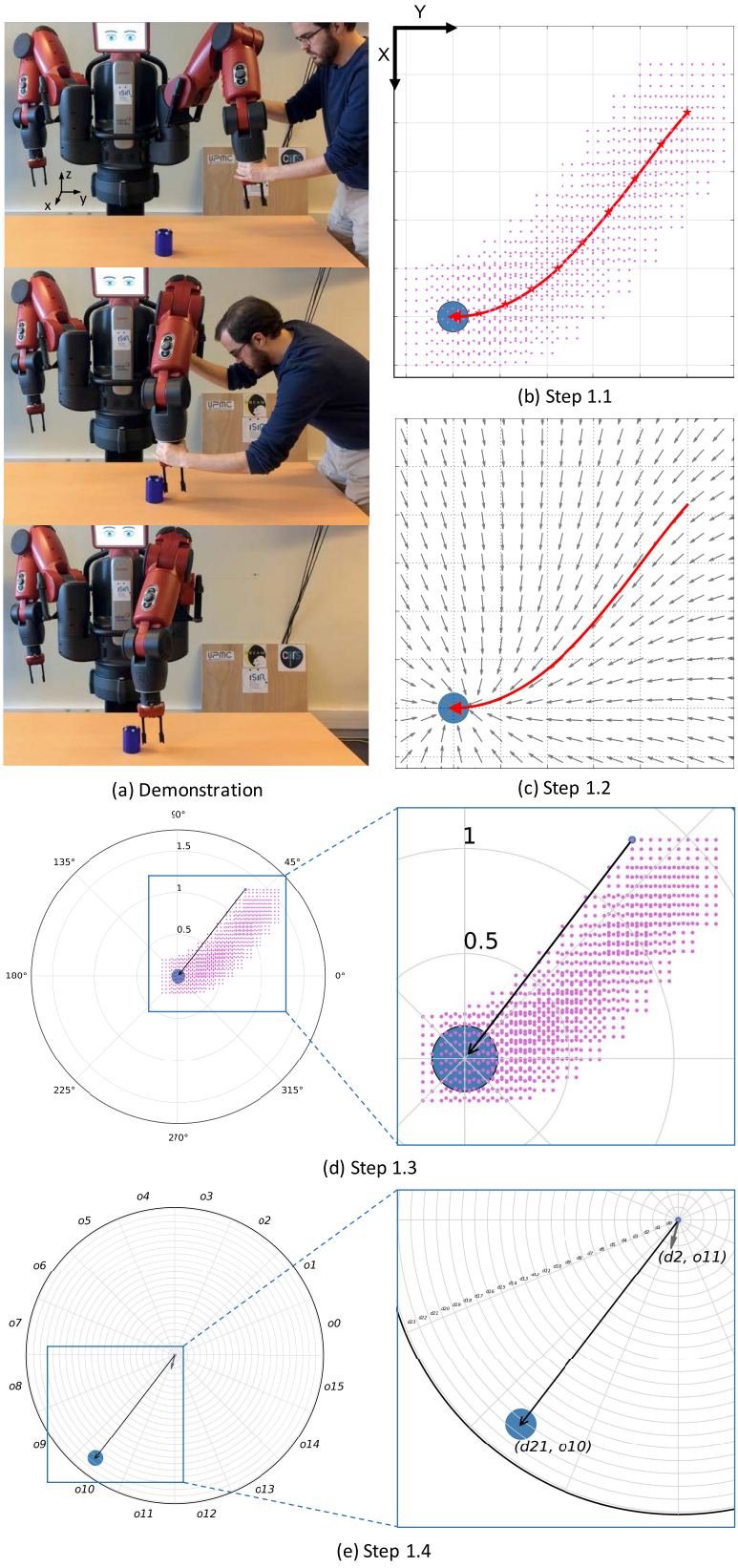
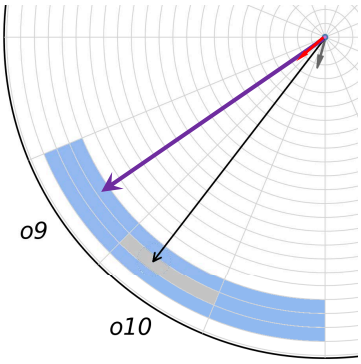
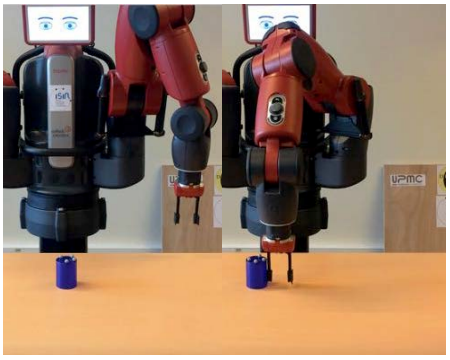

Y
X
(a) Demonstration
(b) Step 1.1
(c) Step 1.2
1.5
1
0.5
135°
45°
180°
0°
225°
315°
270°
1
0.5
(d) Step 1.3
o4
o3
o5
o2
o6
o1
o7
o0
o8
o15
o9
o14
o10
o13
o11
o12
(d2, o11)
(d21, o10)
(e) Step 1.4

Figure 6.4



(f) Step 2.2



(g) Reproduced effect

Figure 6.4: (continued) Example of the inference of an action to push an object to the right. (a) An initial kinesthetic demonstration of a trajectory to *push* the object is performed. At the top, the setup of the experiment. At the middle, the demonstration performed by an external agent. At the bottom, the result of the demonstration. Although the reference frame of the setup is located in the base of the robot, in order to facilitate the visual comprehension of the setup the reference frame is located in another place in this figure. (b) Step 1.1: Example of the computation of a vicinity. The trajectory is performed in the Cartesian X-Y plane, and the figure represents the top view of the setup. The trajectory is represented with a red arrow, and the object with a blue circle. The red stars represent the waypoints selected to compute the vicinity of the trajectory. For each of them, a set of end-effector positions is generated, represented by the pink points. (c) Step 1.2: Example of the computation of movements from new positions of the end-effector. Each grey arrow represents the next movement, from a position, to be executed by the end-effector in order to reproduce the demonstrated effect. (d) Step 1.3: Example of the computation of a block of information from a position of the vicinity of the trajectory selected with a blue circle. In this example, a block of information is a tuple composed of a continuous movement of the end-effector and a continuous robot-object relation state. The grey vector represents the movement, obtained from the vector field from the current position. The black vector represents the position of the object with respect to the end-effector position, i.e. the relation state. (e) Step 1.4: Example of the discretization of a block of information. Each vector is discretized. As a vector is composed of a *distance* and an *orientation*, represented in continuous spherical coordinates, the block of information is discretized into a tuple represented by a discrete movement, i.e. (*d2*, *o11*), and a discrete relation state, i.e. (*d22*, *o10*). The origin of coordinates correspond to the position of the end-effector, because the position of the object is relative to it. (f) Step 2.2: Example of the inference of a movement by the action generator. In this case, at an instant of time during the generation of an action the relation between the end-effector and the object is similar to the discrete relation state computed in the previous steps, e.g. (*d21*, *o11*). For the current position of the end-effector a set of nearest neighbors is computed: for each blue neighbor*bin* of the discretization, with a *distance* and *orientation*, a neighbor block of information is computed. However, in the figure only 1 neighbor is depicted. Each neighbor is composed of a relation state, e.g. the purple vector with values (*d21*, *o9*), and a movement, e.g. the red vector with values (*d2*, *o9*). The final movement of the end-effector is computed as a mean of all the movements of the current relation state and the related neighbors. (g) Different instants during the reproduction of the demonstrated effect (from left to right).

Algorithm 5 Initial interaction acquisition by demonstration

S: skill name
$\mathbb{D}$: continuous dataset of interactions acquired from demonstration
$\mathbb{T}_e$: trajectory composed of end-effector positions
$\mathbb{T}_o$: object position during the trajectory
$\mathbb{T}_g$: openness of the gripper during the trajectory
$\mathbb{T}_c$: high-level object states during the trajectory

```
S ← GetSkillName()
D ← CreateEmptyDataset(S)
while WaitForDemonstration() do
    while demonstration do
        T_e, T_o, T_g, T_c ← RecordInteraction()
    D ← SaveInteraction(T_e, T_o, T_g, T_c)
```

the position of the object, (ii) the openness of the gripper, and (iii) the high-level object states. Once all the demonstrations are finished to reproduce the same effect, i.e. to build the same skill, the acquired information is stored into a repertoire of states, $\mathbb{D}$ (Equation 6.2). Algorithm 5 shows the pseudo-code of this process.

Each skill is used to perform an action reproducing an effect. Therefore, before the demonstrations, a name is provided to the skill, representative of the action, e.g. *grasp* or $push_{right}$.

Once the dataset of interactions is available the method to build skills starts. This process entails two steps (see Figure 6.4):

Step 1 The dataset of interactions, $\mathbb{D}$, is transformed into a repertoire of movements and contextual states, R, in order to build skills that can generate actions adapted to changes of both the object position and the high-level object states (Section 6.4.2).

Step 2 Based on the discrete repertoire created during the previous step, the action generator is built (Section 6.4.3). The action generator can be used to infer movements to reproduce an effect on an object.

6.4.2 Step 1: Adapting the Interactions to Learn Adaptive Skills

The dataset of interactions, $\mathbb{D}$, represents an action producing an effect on an object. Namely, these interactions can be used to generate an action reproducing the effect under the *same* context, i.e. given the same robot-object relation states, and with similar high-level states. However, the movements inferred by the action generator must be adapted to (i) changes of the high-level object states and/or (ii) changes of the position of an object, i.e. changes of the robot-object relation state. Therefore, the objective of this section is to generalize the knowledge to reproduce the effect provided by an interaction in a specific context to *different but close* contexts. Close

Algorithm 6 Step 1: Adapting the interactions to learn adaptive skills

S: skill name
$\mathbb{D}$: continuous dataset from demonstration
R discrete repertoire of movements and context states
N_{dist}: number of discrete bins used for the distance
N_{orien}: number of discrete bins used for the orientation
N_{inclin}: number of discrete bins used for the inclination
θ_{s}: discretization configuration associated to the skill
τ: a demonstrated trajectory
ω_τ: information associated to a waypoint of a trajectory
ϑ_τ: vector field associated to a trajectory
ϱ: sampling area
ς_ϱ: size of the sampling area
ρ: information associated to an element of the sampling area
x_ρ: position of the end-effector associated to a sampling element
Λx_ρ : continuous displacement of the end-effector associated to a sampling element
y_{ω_τ}: position of the object associated to a waypoint
γ_ρ: continuous relation state associated to a sampling element
g_{ω_τ} : continuous gripper state associated to a waypoint
h_{ω_τ} : continuous high-level object states associated to a waypoint
Δx_ρ : discrete displacement of the end-effector associated to a sampling element
δ_ρ: discrete relation state associated to a sampling element
G_{ω_τ} : discrete gripper state associated to a waypoint
H_{ω_τ} : discrete high-level object states associated to a waypoint
B : block of discrete information

```
1:  function DatasetAdaptation(S, ς_ϱ, N_orien, N_inclin)
2:      N_dist ← ComputeDistanceBins(ς_ϱ)
3:      θ_s ← ComputeDiscretization(N_dist, N_orien, N_inclin)
4:      𝔻 ← LoadContinuousDataset(S)
5:      R← CreateEmptyDiscreteRepertoire(S)
6:      for τ ∈ 𝔻 do
7:          ϑ_τ ← ComputeVectorField(τ)
8:          for ω_τ ∈ τ do
9:              y_{ω_τ} ← GetObjectPosition(ω_τ)
10:             ϱ ← SamplingArea(ω_τ)
11:             for ρ ∈ ϱ do
12:                 x_ρ ← GetEndEffectorPosition(ρ)
```

Algorithm 6 Step 1: Adapting the Acquired Repertoire of States (continuation)

```
        Λx_ρ ← GetEndEffectorDisplacement(ϑ_τ, x_ρ)
        γ_ρ ← ComputeRelationState(x_ρ, y_ω_τ)
        g_ω_τ ← GetGripperState(ω_τ)
        h_ω_τ ← GetContextStates(ω_τ)
        Δx_ρ, δ_ρ, G_ω_τ, H_ω_τ ← DiscretizeInfo(θ_s, Λx_ρ, γ_ρ, g_ω_τ, h_ω_τ)
        B ← CreateBlock(Δx_ρ, δ_ρ, G_ω_τ, H_ω_τ)
        R ← StoreInfo(R_s, B)
    return R, θ_s
```

contexts are composed of different robot-object relation states. It would be highly expensive to record examples of the robot reproducing an effect encompassing all the possible positions of the object with respect to the robot's end-effector. In Calinon et al. (2010) a set of Gaussians is computed given a trajectory, representing position states of the robot's end-effector around the trajectory (Section 2.1.2). Inspired by this approach, a sampling of positions of the end-effector around each demonstrated trajectory is computed. This sampling is called the *vicinity* of a trajectory, and it is explained in Step 1.1.

For each position of the end-effector within the vicinity a *episodic block* of information related to a movement at an instant of time, B, is acquired (Step 1.3). Each block is a triple composed of (i) the movement to execute to reproduce the effect from a position of the end-effector, $(\Lambda x_t, \Lambda g_t)$, (ii) the robot-object relation state at that moment, δ, and (iii) the high-level object state at that moment, H:

$$B = ((\Lambda x_t, \Lambda g_t), \delta, H)$$

In order to compute the movement needed to reproduce the effect from the current position of the vicinity, previously a*vector field* is generated based on the demonstrations (Step 1.2). Finally, in the Step 1.4, the blocks from the different interactions are discretized and stored together into a discrete repertoire of states, R:

$$R = \{B\}$$

The pseudo-code of the adaptation process is available in Algorithm 6.

Step 1.1: Computing the Vicinity

A vicinity is computed for each demonstrated trajectory. First, the trajectory is reduced to a set of equidistant waypoints (represented as red stars in the Figure 6.4). The number of waypoints is computed based on the length of the trajectory. The higher the number of waypoints, the more precise the representation of the

demonstration[1]. For each waypoint the function *SamplingArea* creates a vicinity of positions of the end-effector around the waypoint. The vicinity is represented as a cubic grid centered in the waypoint with side size Q, and composed of P x P x P equidistant positions, P and Q being preset values.

Step 1.2: Computing Movements of the End-effector from New Positions

The objective of using a vector field is computing the next movement of the end-effector to reproduce the effect. In the current manuscript a vector field is obtained from a demonstration using a DS called *diffeomorphism* (explained in Section 3). This DS has a parameter to compute the tendency to reproduce the demonstrated trajectory, defined based on experience.

The vector field is computed, by the function *ComputeVectorField*, based on the trajectory of each demonstration using a *diffeomorphism* (see Section 3.3.3 for further details). An example is depicted in Figure 6.4, c. If the end-effector is in a position close to the demonstrated trajectory the vector field provides a vector converging to the trajectory, possibly reproducing the expected effect. The vector corresponds to a movement of the end-effector. However, even with the right parameter, if the position of the end-effector is far from the demonstrated trajectory, e.g. at the left corner in the Figure, the movements directly converge to the attractor, not reproducing the effect.

Step 1.3: Creating Blocks of Information

This process entails the following stages:

Acquiring the Movement: the movement of the end-effector from a position to reproduce an effect is a vector in Cartesian coordinates. It is directly provided by the vector field for a position of the end-effector by the function *GetEndEffectorDisplacement* from the available vector field related to the trajectory. Similarly, the continuous state of the gripper is a value with range [0, 100], acquired by the function *GetGripperState*. The value of the gripper state for positions of the same vicinity is similar to the value of the corresponding waypoint.

Acquiring the Robot-Object Relation State: as aforementioned, the robot-object relation state represents the position of the end-effector, always available, with respect to the object position, acquired by the function *GetContextStates*. It is a vector from the end-effector to the object in Cartesian coordinates, computed by the function *ComputeRelationState*.

Acquiring the High-level Object States: in the current work the high-level states are always discrete, and they are directly acquired by the function *GetContextStates*. Similarly to the gripper state, positions of the same vicinity have the contextual information of the corresponding waypoint.

[1] A too high number of waypoints can affect the velocity in which the action generator infers a movement, because of the size of the BN.

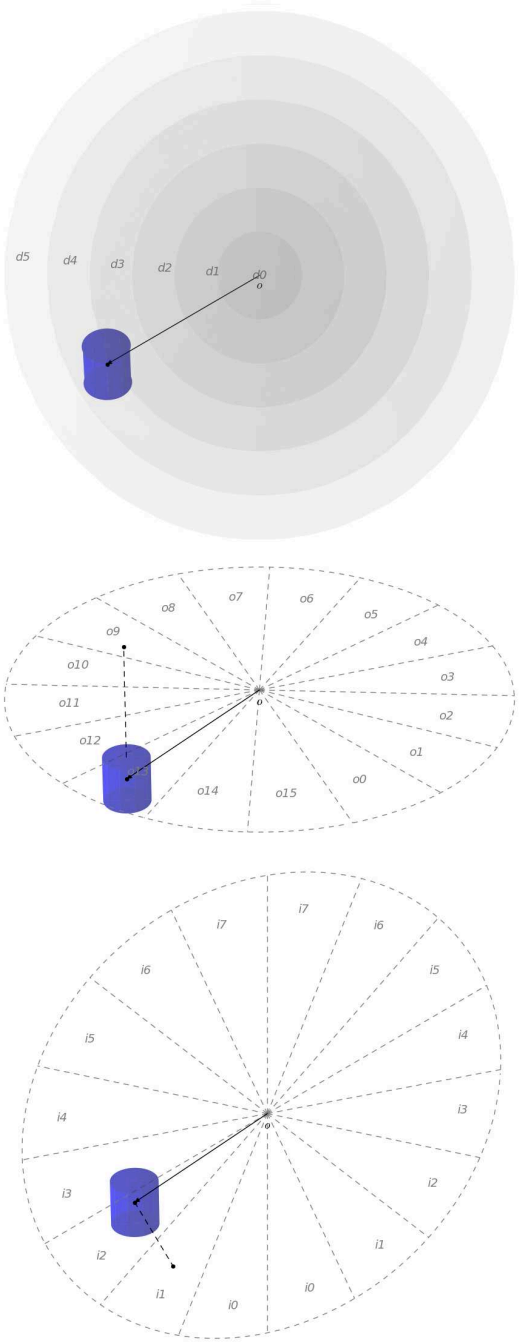


Figure 6.5: Example of the discretization configuration used in experiments (see Section 6.5). The black arrow represents a robot-object relation state, i.e. the position of the object with respect to the robot's end-effector (at the origin). At the top, representation of the *distance* bins, with value *d4* for the vector in the example. At the center, representation of the *orientation* bins, with value *o10* for the vector in the example. At the bottom, representation of the *inclination* bins, with value *i2* for the vector in the example.

Step 1.4: From Continuous to Discrete Information

The action generator is a BN, learned using a discrete dataset. Therefore, each block of information is discretized before being stored into R. To that end, a *discretization configuration* must be available. The values for the *orientation* and *inclination* are predefined, whereas the values for the distance is computed. The length of the minimal size of a *distance bin* is related to the distance between two positions in the vicinity:

$$minimal\ distance\ bin\ size = \frac{Q}{P-1} \tag{6.8}$$

This bin size is used to compute the whole distance discretization, e.g. every bin can have the same size or the size can grow following some heuristic (see Experiment 1 for more details). Therefore, the accuracy of the actions inferred by the action generator depends on the values of P and Q.

This configuration is applied to discretize the displacement of the end-effector, the relation state and the gripper state. On the one hand, the definition of a gripper state directly depends on the accuracy of the task to perform, e.g. for some tasks *open* and *close* can be enough, whereas for others degree of openness can be relevant. On the other hand, both a movement of the end-effector and the relation state are vectors defined in the Cartesian coordinates. In these coordinates the range of each axis is $[-\infty, \infty]$, which makes very difficult to find a proper discretization. For this reason, the displacement of the end-effector and the relation state are transformed to *spherical coordinates* before being discretized. A vector in spherical coordinates is composed of a *distance*, with range $[0, \infty]$, an *orientation*, with range, $[-\pi, \pi)$ and and *inclination*, with range $[0, \pi]$. In the current work, the range of the distance is constrained to the range $[0, 0.5)$ because any object farther than one meter is considered as *far* from the robot. And thus several movements are needed to reach it, becoming *close* eventually. Each of these ranges is divided into a preset number of *bins* of the same size. Figure 6.5 shows an example of this discretization. In Section 6.5 an experiment shows the impact of using different discretization configurations.

6.4.3 Step 2: Running a Skill to Reproduce an Effect on an Object

An action generator, ϕ, is a BN that infers discrete movements, Δx, to reproduce an effect, e, on an object. Each movement is adapted to both the position of the object with respect to the end-effector, i.e. the discrete relation state, δ, and the discrete high-level object states, H, at certain instant of time (see Figure 6.1). A movement is composed of the end-effector displacement and the gripper state, which are independently inferred:

$$\begin{aligned}(\Delta x_t, \Delta g_t) &= \phi(e, \delta, H) \\ \Delta x_t &= \arg\max_{\widehat{\Delta} x_t} P(\widehat{\Delta} x_t \mid e, \delta, H) \\ \Delta g_t &= \arg\max_{\widehat{\Delta} g_t} P(\widehat{\Delta} g_t \mid e, \delta, H)\end{aligned} \tag{6.9}$$

As described in Section 6.4.2, a discrete vector is composed of a *distance*, an *orientation* and an *inclination*. Therefore, a discrete movement is described using three discrete values:

$$\Delta x_t = (\Delta_{dist} x_t, \Delta_{orien} x_t, \Delta_{inclin} x_t)$$

Although it is possible that there is a weak dependency among these values, in order to speed up the computation of a movement we consider tha these values are independent. And thus the inference of a movement consist in the individual inference of each one of them (see d-separation in Section 3.2.3):

$$\begin{aligned} \Delta x_t = (& \arg\max_{\Delta_{dist} x_t} P(\Delta_{dist} x_t \mid e, \delta, H), \\ & \arg\max_{\Delta_{orien} x_t} P(\Delta_{orien} x_t \mid e, \delta, H), \\ & \arg\max_{\Delta_{inclin} x_t} P(\Delta_{inclin} x_t \mid e, \delta, H)) \end{aligned} \tag{6.10}$$

Examples of learned BN are available in Figures 6.14b and 6.14c.

In the current work, an action generator is learned each time the skill runs, based on the dataset of discrete blocks, D, computed in the previous section (Step 2.1).

The inference and execution of each movement is performed within the *perception-action cycle* explained in Section 6.3.2 and depicted in Figure 6.3. The perceptual information of the robot is transformed and provided to the action generator, which infers the movement (Step 2.2). Then, the movement is executed by the robot using its inverse kinematic model. This execution generates a displacement of the position of the robot's end-effector, which can modify the robot's environment. If the effect has not been reproduced, or a maximum number of movements executed, a new iteration of the cycle is executed.

The pseudo-code to run a skill is available in Algorithm 7.

Step 2.1: Learning the Action Generator

Learning a BN consists in two steps: (i) the generation of a structure representing the causal relations of the components of a block, and (ii) the computation of their conditional probability distributions (CPDs).

The discrete dataset of blocks can contain information of one ore more interactions, i.e. they have been computed based on different trajectories. And these trajectories can suggest different movements for the same robot-object relation state and contextual information. The uncertainty generated in these cases is directly handled by the probability distributions of the BN, computing different probabilities for each movement observed under the same circumstances.

Algorithm 7 Step 2: Running a skill to reproduce an effect

S: skill name
e_s: effect associated to the skill
ReproducedEffect(): boolean function indicating if the effect has been reproduced
ε: end-effector selected to run the skill
R: discrete repertoire of movements and states
$\mathbb{T}_E$: sequence of continuous waypoints representing the inferred trajectory
$\mathbb{P}_E$: set of probabilities for each waypoint
$\mathbb{T}_O$: set of continuous object positions acquired at the same instant of time that the waypoints
T_G: sequence of values of the gripper openness acquired at the same instant of time that the waypoints
θ_s: discretization configuration associated to the skill
ϕ_{D_s}: action generator
nb_mov: number of movements executed
nb_mov_{max}: maximum number of movements that can be executed
$know_{mov}$: indicates the current knowledge to infer a movement
x: current position of the end-effector
y: current position of the object
g: current gripper openness
H: current discrete high-level contextual states
$\widetilde{\Lambda}x$: inferred continuous Cartesian movement vector
$\widetilde{p}_{\Lambda x}$: inferred probability associated to a movement
$\widetilde{G}$: inferred discrete gripper action state
ExecuteMov(): function executing a movement

1: **function** ReproduceEffect(S, ε, e_s, *ReproducedEffect()*, nb_mov_{max})
2: $R \leftarrow$ *LoadDiscreteRepertoire(S)*
3: $\phi_R \leftarrow$ *CreateActionGenerator(R)*
4: $\theta_s \leftarrow$ *LoadDiscretizationConfiguration(S)*
5: *nb_mov* is initially set to 0
6: *mov_knowledge* is initially set to *True*
7: $\mathbb{T}_E$, $\mathbb{P}_E$, T_G, $\mathbb{T}_O \leftarrow \emptyset$
8: **while** $\neg$*ReproducedEffect*$(e_s) \cap know_{\text{mov}} \cap nb\_mov < nb\_mov_{\text{max}}$ **do**
9: *x, g* $\leftarrow$ *GetProprioceptiveInformation*(ε)
10: *y, H* $\leftarrow$ *GetContextualStates()*
11: $\widetilde{\Lambda}x$, $\widetilde{p}_{\Lambda x}$, $\widetilde{G}$ = *InfereMov*$(\theta_s, \phi_R, x, y, g, H)$
12: **if** $\widetilde{\Lambda}x \neq \emptyset$ **then**

Algorithm 7 Step 2: Running a skill to reproduce an effect (continuation)

```
            x = x + Λ̃x
            Add x to 𝕋_E
            Add p̃_Lambdax to ℙ_E
            Add G̃ to 𝕋_G
            y = ExecuteMov(Λ̃x, G̃)
            Add y to 𝕋_O
        else
            know_mov = False
        Add 1 to nb_mov
    return ReproducedEffect(e_s), 𝕋_E, ℙ_E, T_G, 𝕋_O
```

Algorithm 8 Step 2.2: Inferring a movement

Ξ: set of discrete relation states
Θ: set of inferred values
Note: Refer to Algorithm 7 for more definitions

```
function InfereMov(θ_s, φ_R, x, y, g, H)
    Θ ← ∅
    γ ← ComputeRelationState(x, y)
    δ, G ← DiscretizeInfo(θ_s, γ, g)
    Ξ ← ComputeNearestNeighbours(δ)
    for δ ∈ Ξ do
        Δx, p_Δx, G ← φ_R(e_s, δ, H)
        if Δx ≠ ∅ then
            Λx ← ToContinuous(Δx)
            Add (Λx, p_Λx, G) to Θ
    Λ̃x ← ComputeMeanMov(Θ)
    p̃_Λx ← ComputeMeanProb(Θ)
    G̃ ← ComputeMeanGripperOpenness(Θ)
    return Λ̃x, p̃_Λx, G̃
```

Step 2.2: Inferring a Movement

The process to infer a movement (Equation 6.9) is detailed in this section. The pseudo-code is available in Algorithm 8.

While computing the vicinity of a trajectory some robot-object relation states may not have been covered by the positions of the grid. Covering all the positions would entail a complex parametrization of the vicinity grid and the discretization

configuration. In order to avoid the identification of this parametrization, the movement inferred for a relation state and contextual information to reproduce an effect is computed as the mean value of a set of relation states, Ξ. This set consists of the *nearest neighbors* relation states of the current relation state, including itself. In this case, for each dimension of the current state, the previous and the next values. For example, for *dimension 3*, the neighbors are *d2* and *d4*. And thus, the size of each dimension is always 3, except for the maximum and minimum distance, which is 2, e.g. the extreme distance only have 1 neighbor. The set of states, including the current state, i, is $[d_{i-1}, d_i, d_{i+1}]$ for each dimension of the relation state:

$$length(nearest\ neighbors) = \prod_{d=1}^{M} N_d$$

where M represents the number of dimensions, and N represents the size of the current dimension d. An example of this computation is available in Figure 6.4. In this example the relation state, the black arrow, is described using a *distance* and an *orientation*, e.g. *d21* and *o10*. The ranges of nearest neighbors would be [*d20, d21, d22*] for the *distance*, and [*o9, o10, o11*] for the *orientation*. Therefore, the set of nearest neighbors would have 9 elements, 3 x 3. Conversely, for the current state *d0*, with range [*d0, d1*], and *o10*, the set of nearest neighbors would have 6 elements, and thus 2 x 3 elements. If another dimension with 3 elements is added, e.g. the *inclination*, the set of nearest neighbors would have 18 elements, 2 x 3 x 3.

The action generator infers discrete movements of the end-effector. Therefore, it is necessary to transform these movements to continuous values. The function *ToContinuous()* selects the mid value of the range corresponding to each value composing the movement. For example, for the movement (*d2, o11*) of the Step 2.4 in Figure 6.4 the function computes the mid value for the ranges of *d2* and *o11*.

6.5 Experimental Framework

6.5.1 Experiments

Two sets of experiments have been executed to validate the building of skills adapted to the contextual states (see Table 6.1). For each skill one or more demonstrations of trajectories reproducing the corresponding effect are previously performed, i.e. a repertoire of contextual states is available to build each skill. Videos of the experiments are available online.[2]

As aforementioned in Section 6.4.2, the accuracy of an action generator is based on the number of positions, P, and the size of the vicinity, Q, selected to transform the repertoire of states representing the demonstrated trajectories. In these experiments two actions generators with different levels of accuracy are learned (see Figure 6.7): a fine-grained action generator inferring small movements (around 2.5 cm) and a coarse-grained action generator inferring small movements (around 6

[2] https://www.youtube.com/playlist?list=PL2drYAFCMtzcZ_RlfiFr2AWtcvgc1qR8o

Table 6.1: Details of the experiments

ID	*Type*	*Objective of the study*	*A priori structure*	*A priori discretization*	*Skills*	*Objects*
1	Skill building	Discretization impact	X		Push	Cylinder
			X		Grasp	Cylinder
2		Structure learning		X	Push	Cylinder
				X	Grasp	Cylinder
3		Generalization	X	X	Push	Cylinder
			X	X	Push	Tea box
4	Task planning and execution	Solving a maze	X	X	Push Set	Cylinder Cake
5		Heating a croissant	X	X	Grasp Release Press	Croissant Pan Dish Button

cm). Although just the fine-grained action generator would be enough to accurately reproduce the effects, the use of the coarse-grain generator provides more realistic trajectories, with bigger movements far from the object and smaller ones close to it. The corresponding P and Q values are 7 positions and 20 cm for the fine-grained one, and 40cm and 8 positions for the coarse-grained one, respectively. The fine-grained generator is used if the end-effector is close to an object (arbitrarily preset to 10 cm), whereas the coarse-grained generator is used in any other case. For these experiments the gripper state is simply discretized into two states: *open* if its continuous state is bigger than 50, or *closed* otherwise.

Robotic Platform

Similarly to the experiments carried out in the previous chapter (see Section 5.3.2), the validation of the method is performed on a Baxter robot. Nevertheless, in the current chapter only the physical robot is used. Each gripper of the robot has a different configuration: on the left gripper, the fingers of the gripper are in the farthest position, in order to grasp big objects. On each finger there are adapters to facilitate the pushing and the grasping. On the right gripper, the fingers are in a intermediate position, in order to grasp smaller objects. And there are only adapters to grasp.

The execution of the robot relies on ROS Indigo Igloo and our kinematic library[3].

[3] https://github.com/cmaestre/baxter_kinematics

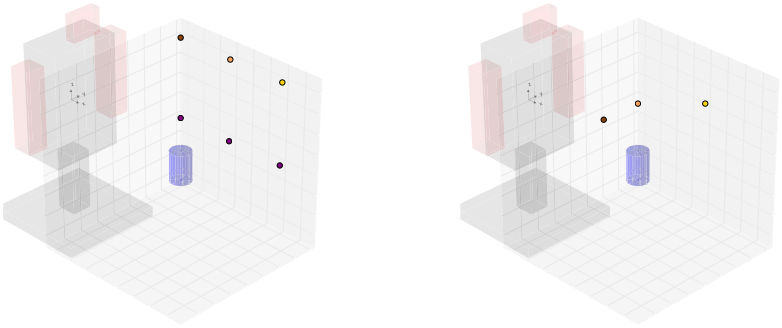


Figure 6.6: Virtual reproduction of the setup, in which the robot is composed of red and gray boxes, and the cylinder is represented in blue. On the left, initial positions to *push*, and on the right to *grasp*. The yellow, orange, and brown points represent the initial positions of the demonstrated trajectories, whereas the purple points represent the initial positions used in the generalization experiment.

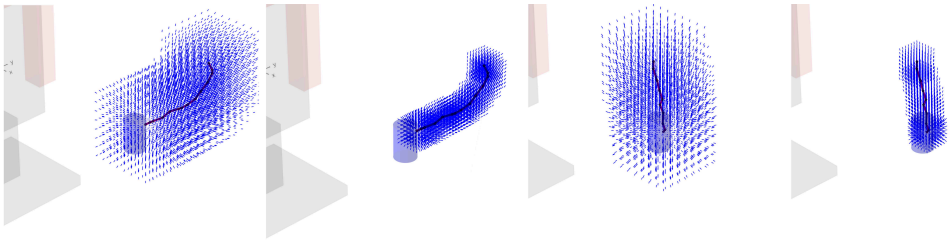


Figure 6.7: Examples of vicinities computed to build the *push* and *grasp* skills at the top and bottom, respectively. On the left, to learn a coarse-grained action generator, and on the right to learn a fine-grained one.

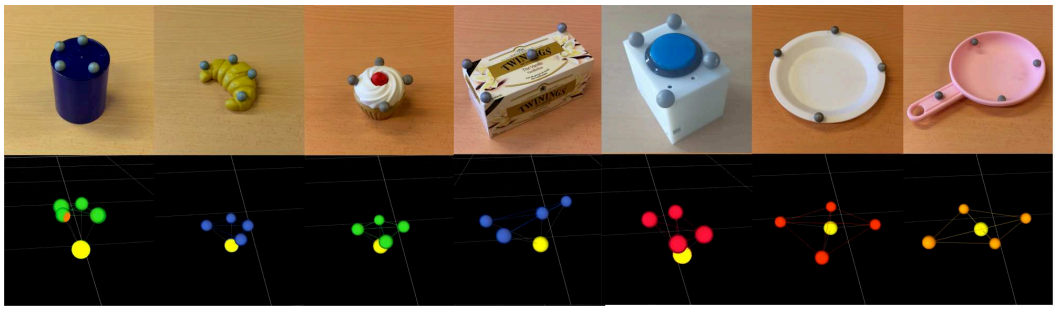


Figure 6.8: Set of objects used in the experiments. The top row shows pictures of a cylinder, a croissant, a cake, a tea box, a button, a dish and a pan, respectively. Each structure of the second row shows the representation captured by the OptiTrack system of the object upon it. The light yellow ball in each structure represents the position of the object gathered by the robot, always in the center of the object except for the hook, located at the handle.

Skill Building Experiments

The first set of experiments makes a study of the results obtained executing A^2L by a Baxter robot in order to reproduce an effect on an object, located on the table in front of the robot. In these experiments only the low-level object states are used, that is, the perception of the robot is constrained to the position of the object.

In these experiments, the robot tries to *grasp* a cylinder and *push it to the right* 5 centimeters with its left end-effector from different initial positions (see Figure 6.12). The initial positions for the *push* action are located at 30 cm in X, Y and Z from the object (yellow point), at 30 cm in Y and Z from the object (orange point), at 30 cm in Y and Z and -30 cm in X from the object (brown point), and the purple points share the same X and Y than these points, except that Z has the same value than the object. The initial positions for the *grasp* action are located at 30 cm in X, Y and Z from the object (yellow point), at 30 cm in Z from the object (orange point), and at 30 cm in X and Y from the object (brown point). The initial positions have been selected close to the initial positions of the demonstrations. All the trajectories inferred by A^2L in these experiments are available in Appendix A.

Experimental Setup

A table of 180 x 80 x 75 centimeters (cm) of width, length and height, respectively, is located in front of the Baxer robot (see Step 1 of Figure 6.4).

Figure 6.8 shows the set of objects used for the experiments. In this set of experiments only the cylinder and the tea box are used. The number of objects used in an experiment is small, based on the idea that a human being cannot handle simultaneously more than 3 or 4 objects (Spelke and Kinzler, 2007). The positions of the objects are acquired using an OptiTrack motion capture system[4]. This system is composed of 4 cameras located at the ceiling, over the setup, and it generates a virtual structure of the markers located on the objects, providing the position of each object defined in the center of it, for instance.

The reference frame of the setup is located at the base of the robot, and thus the perceptions perceived by the robot are relative to itself, e.g. an object position.

Experiment 1: Study of the Discretization Impact The objective of this experiment is to analyze the impact of different discretization configurations in the reproduction of an effect on an object. In this case, the discretization for the *orientation* and *inclination* have bins of the same size, because the range of both is $[0, 2\pi]$. And thus, the first and last bins are connected. In this experiment, the values used for the *orientation* and *inclination* are 4, 8 and 16, selected based on experience. Conversely, the *distance* has a range of [0, M], where M represents the longest distance of a movement of the robot, in this case 50 cm. Two different types of discretizations for the *distance* are compared: on the one hand, a *linear* discretization, in which M is split up in bins of the same size. On the other hand, a

[4] http://optitrack.com/

progressive discretization is computed inspired on the Fibonacci sequence, in which each number is the result of the addition of the previous two numbers, being the first two numbers 0 and 1, respectively. In our case, the distance of each bin is the addition of the size of the previous two bins:

$$\begin{aligned} dist_0 &= minimal\ distance\ bin\ size \\ dist_1 &= dist_0 * 2 \\ dist_n &= dist_{n-1} + dist_{n-2} \end{aligned} \tag{6.11}$$

In both cases the minimal size of each bin is computed as in Equation 6.8.

In order to test different discretization configurations a *hard-coded* structure of the BN representing the action generator is provided, in the same vein that the structure used in Chapter 5. This structure represents relations of dependency of the low-level states to be inferred by the BN, i.e. the movement and gripper state, w.r.t. any other state (see Figure 6.14b).

The cylinder has been selected because of the complexity to interact with it. More precisely, in order to push it in a specific direction, e.g. to the right, the end-effector has to touch it exactly in the center of the left side of the object. Otherwise it will be moved in a different direction. Similarly, while executing a grasping action the end-effector must be on the center of the object from the top, or it will slip from the gripper.

Experiment 2: Study of the Structure Learning This experiment builds the *push* and *grasp* skills using the *hard-coded* structure, and compare their performance when these skills are built using structures created combining structure learning methods and scores available in the literature. To that end, a fix discretization configuration is provided, based on the results obtained in Experiment 1.

As in the Chapter 5, we have selected *hill climbing* (Chickering et al., 1995) , with no a priori information about the structure, and *K2* (Cooper and Herskovits, 1992), which needs as input an order of the random variables of the BN for the structure learning. In this experiment, the random variables are:

- the *effect*,
- *grasped, the object state representing if the object is grasped*,
- the *distance*, *orientation*, and *inclination* of the robot-object relation state,
- the *distance*, *orientation*, and *inclination* of the movement,
- the *openness* of the gripper

In this experiment, two versions of the K2 algorithm are available, with the current *increasing* order, and with the opposite *decreasing* order.

These methods are combined with the score methods *AIC*, *BIC*, *Likelihood*, *K2* and *BDeu* to learn the structure. These scores are described in Section 3.2. Once the structure is available the correspondings CPDs are learned using Maximum a

posteriori (MAP) with a smooth *a priori*. Except for K2 and BDeu, which have their own a prioris.

Experiment 3: Skill Generalization In this experiment the *push* skill is built using the best discretization and structure identified in the previous experiments. Then, the generalization capabilities of the skill is analyzed in two different situations: *pushing* the cylinder from positions not observed during the demonstrations; and *pushing* the tea box, with different size and weight than the cylinder.

Validation

Similarly to Calinon et al. (2010, 2011), we use metrics to evaluate each reproduction attempt w.r.t. the available demonstrations. However, our goal is not reproducing the demonstrations, but rather reproducing their effect. Therefore, instead of using techniques as the *root-mean-square* to compare the error of the inferred trajectory, we define two metrics to measure the error in the effect depending on the type of skill used:

- For *pushing* an object, the mean-square-error of the Euclidean distance between the final position of the object and the expected one is computed, with a precision threshold of 1 cm.
- For *grasping* an object, the number of times the object is grasped, in the range 0, if all the attempts fail, and 3 if all are successful.

The quality of the performed action is measured based on two values:

- The number of movements inferred, with a maximum of 15 movements.
- The mean *smoothness value* among the movements. The smoothness value between two movements, m_1 and m_2, is computed as:

$$smoothness\ value = cos^{-1}(\mid m_1 \mid * \mid m_2 \mid)$$

As the Baxter robot does not provide tactile feedback, an object is *considered* as grasped if the robot-object relation state is under certain preset distance, and if the value of the gripper state is in the lower half range (under 50). The object is considered as *pushed* certain distance if it is under an Euclidean distance of 1.5 cm from that position.

Task Experiments

In the second set of experiments, the best discretizaton parameters and structure used to build skills identified the first set of experiments are validated. To that end, two experiments are defined with different objectives:

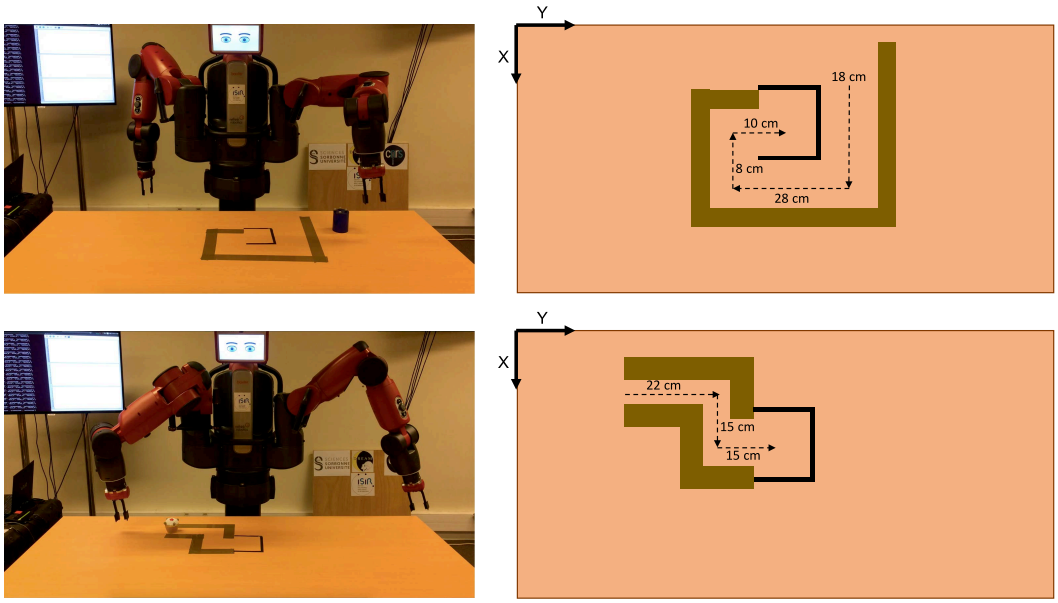


Figure 6.9: Setup of the mazes used in Experiment 4. On the top, for the first maze, and on the bottom, for the second maze. In both cases, from left to right, the physical setup and the expected distances to *push* the corresponding object in order to solve the maze. At the right side of the robot there is a screen showing the *distance*, *orientation* and *inclination* of the object w.r.t. the robot end-effector.

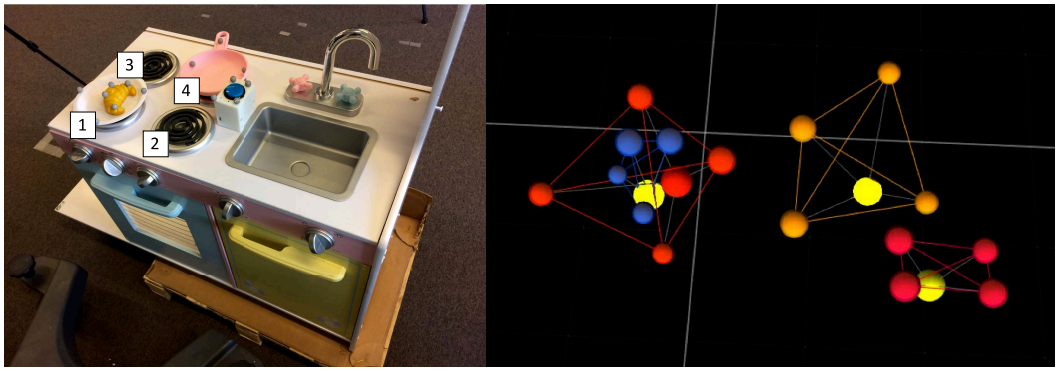


Figure 6.10: Setup used in Experiment 5. On the left, an image of the setup from the robot's point of view. A kitchen is located in front of the robot. The elements of the kitchen used in the experiment are four stoves, a dish, a pan, a croissant and a switch button. On the right, example of the setup acquired by the motion capture system from the same point of view used for the image. The yellow markers represent the position of each object.

Experiment 4: Solving a Maze

The goal of this experiment is twofold: (i) to validate the reproduction of different results for the same skill, e.g. *push to the right* an object different distances; and (ii) to combine different skills to solve a task.

The setup of this experiment consist of two mazes of different configurations. The robot must reproduce a sequence of actions *pushing* an object specific distances to reach the goal (see Figure 6.9). The objects to *push* have different sizes, shapes and weights. These are the cyilinder for the first maze, and the cake for the second maze.

In order to reproduce the sequence of actions different skills have been demonstrated to the robot. First, a set of demonstrations are executed to *push* an object to the *left*, to the *right*, *close* to the robot, and *far* from the robot. Before executing each *push* action it is necessary to set the robot's end-effector on one side of the object, e.g. to *push* it to the right the end-effector must be located at the left of the object. Therefore, a set of demonstrations are executed to move the end-effector from the object to on of its sides (see Figures 6.15 and 6.16).

In these experiments the experiment designer chooses the next actions to execute and the expected effect, including the distance to move an object or the robot's end-effector.

Experiment 5: Heating a Croissant

The objective of this experiment is to show (i) that skills built by A^2L, based on demonstrations, can be used to perform a multi-step task in a realistic scenario, adapting to both low-level and high-level contextual states; and (ii) that the skill behavior can be traced. The possibility to understand the behavior of a skill can be exploited in higher-level stages to identify behavioral regularities, for instance to perform transfer learning and skill generalization.

A scenario is defined, comprising a toy-like kitchen and other objects on it (see Figure 6.10). Two tests have been initially carried out in order to show the robustness of the skills w.r.t. spatio-temporal perturbations. In these tests the robot context is composed of low-level states. First, a *pick-and-place* experiment has been executed. The robot has to *grasp* the croissant and *release* it inside the pan. The position of both objects changes and the ongoing robot action has to adapt to these changes. Second, the robot has to *grasp* the croissant. In this case, during the execution of the *grasping* action the position of the end-effector is externally modified. Besides, the croissant position changes.

Th same scenario is used to assess the adaptability of the system to changes of the high-level states. Therefore, the robot context is composed of low-level and high-level states. The task consists in heating a croissant until reaching a specific temperature. The high-level states of the objects are:

- Stove number 4 : *on* (red) or *off* (black).

- Croissant: *cold* (yellow), *mid temperature* (salmon), *high temperature* (brown) or grasped (green).
- Button: *pressed* or *not pressed*.

These state colors are visually represented during the experiment in a screen next to the robot (see Figure 6.17). Initially, the value of the high-level states are the following: the stove is *off*, the button is *not pressed*, the *croissant* is *cold* and it is located in the dish, over the stove 1 (which is always off). If the croissant is in the pan, the pan is over the stove 4, and the stove is *on*, the temperature of the croissant changes from *cold* to *mid temperature* after few seconds; and from *mid temperature* to *high temperature* again after few seconds.

The sequence of actions to reach the task goal are:

1. *Push* the button to turn the stoves on.
2. *Grasp* the croissant.
3. *Release* it into the pan.
4. When the croissant has reached the *mid temperature*, *grasp* it again.
5. *Release* it back into the dish.
6. *Turn* the stove *off*.

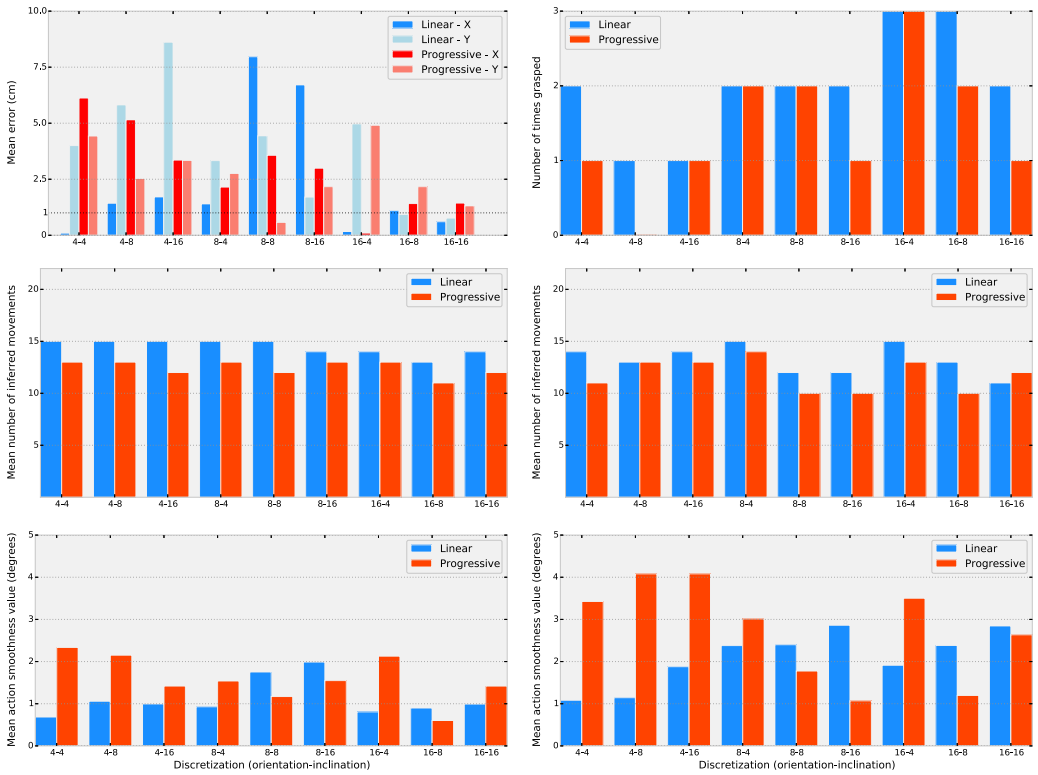
Therefore, the actions demonstrated to the robot are *pressing* the button, *grasping* the croissant, and *releasing* the croissant from the dish to the pan, and vice versa. The demonstrations are performed with the left robot end-effector, and actions executed with the right end-effector. Before the *grasp* and *press* actions the end-effector is randomly located over the setup, in a range of 20 to 40 cm of height, in order to show that actions can be inferred from different initial positions of the end-effector.

The multi-step experiment is executed using a STRIPS planner with PDDL-like problem specification (see Section 3.4). It is written in Python, and called PyDDL[5]. The problem, domain and planned sequence of actions actions are available in Annex B. The task planner perceives the state of the objects using ROS. Initially, the sequence of required actions is computed. Then, each time a state changes the corresponding action is executed. The task planner is also in charge of changing the colors of the screen representing the object states.

6.5.2 Experimental Results

Experiment 1: Study of the Discretization Impact Results are similar using both *linear* and *progressive* discretizations for *pushing* and *grasping* the cylinder (see Figure 6.11). However, the *linear* discretization reaches slightly better results (see top of Figure 6.11, a).

[5] https://github.com/garydoranjr/pyddl



(a) Push to the right (b) Grasp

Figure 6.11: Results of Experiment 1.

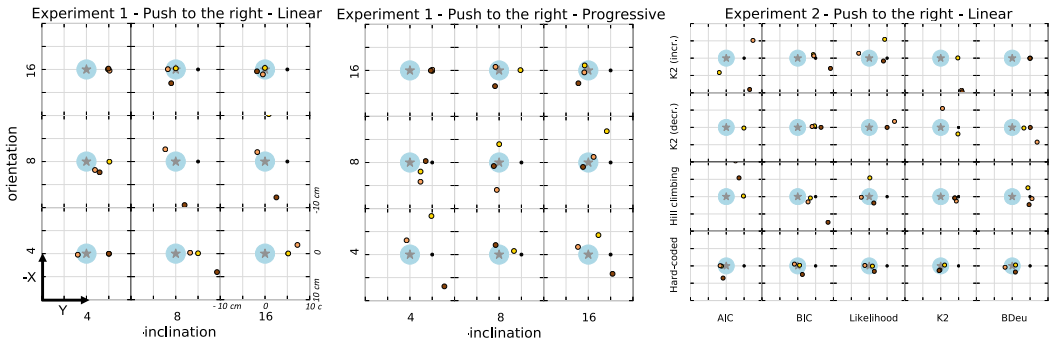
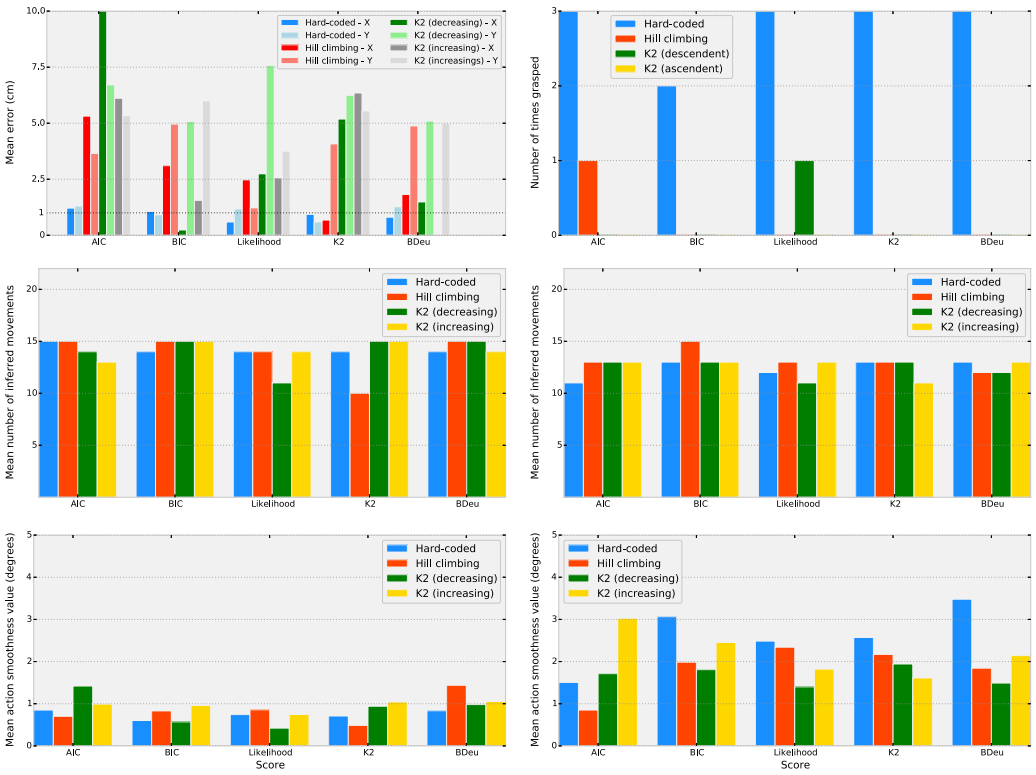


Figure 6.12: Displacement of the cylinder in Experiments 1 and 2 using the *linear* and *progressive* discretizations for the *distance*. A cell shows the result of *pushing* the cylinder from the 3 initial positions, using a discretization or structure learning configuration. In each cell, the grey star in the center represents the expected final position, the black point represents the initial position of the object (5 cm at the *left* of the star).

In the *pushing* experiments the final position of the cylinder is closer to the expected one, i.e. the mean error decreases, for a higher number of bins representing



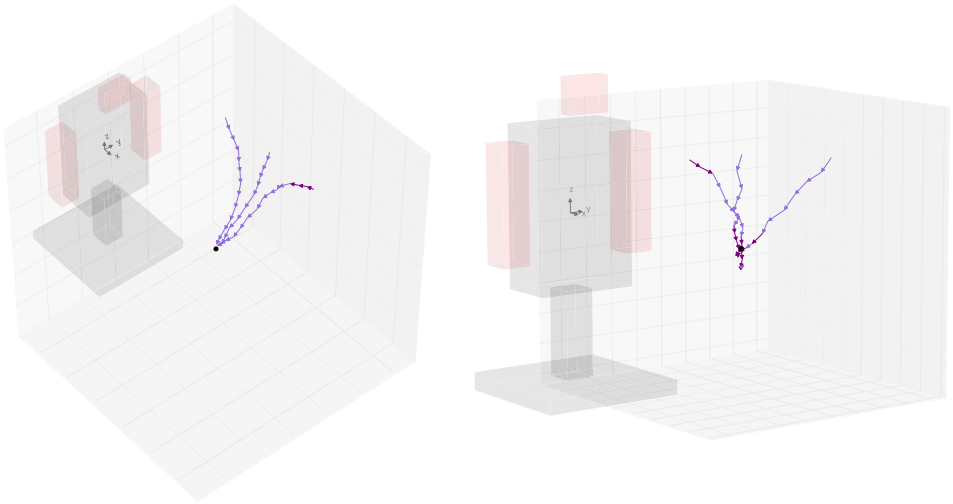
(a) Push to the right (b) Grasp

Figure 6.13: Results of Experiment 2.

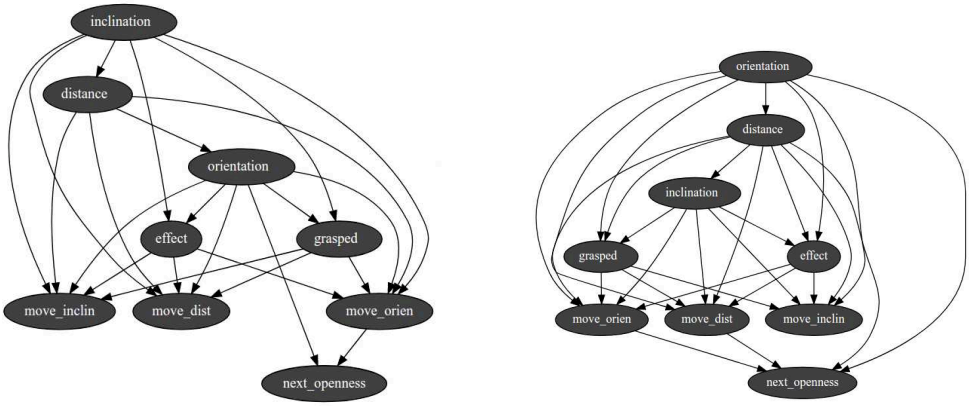
the *orientation* and *inclination*. For the *linear* discretization using 4 bins for the *orientation* the cylinder is barely moved (the error in Y is high), whereas using 8 bins it mainly pushes it far and close to the robot. In the *progressive* discretization the results improve for a higher number of bins for the *orientation* for any *distance* value. In both cases using the configuration 8-4 the robot starts to *push* the cylinder in the right direction. For the configurations 16-8 and 16-16 is the error so small that it can be disregarded. These results are confirmed by the Figure 6.12. Regarding the action, the *progressive* discretization uses less number of movements but it is less smooth, whereas the movements of the *linear* discretization are smoother, thus using a higher number of them. However, in both cases the number of mean movements used is very close.

Similarly, in the *grasping* experiment the results improve for configurations with a higher number of bins. In this case, the best values are reached for the configurations 16-4 and 16-8. Again, the actions inferred using the *progressive* configuration are less smooth using a lower number of movements, than those actions inferred using the *linear* discretization.

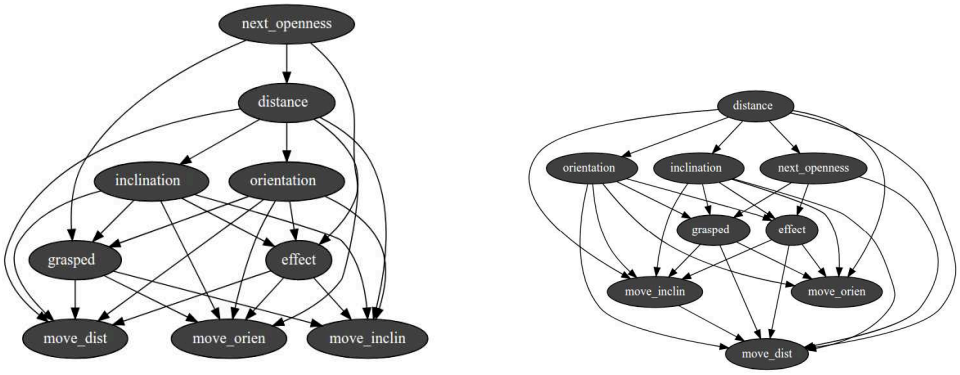
Experiment 2: Study of the Structure Learning In this experiment, the discretization configuration is fixed composed of the best result obtained in the



(a) Actions inferred for *push* and *grasp*, respectively.



(b) *S*tructure of the coarse- and fine-grained action generators for *push*, respectively.



(c) *S*tructure of the coarse- and fine-grained generators for *grasp*, respectively.

Figure 6.14: Actions inferred and structure of the actions generators whose discretization is composed of *linear* distance, 16 orientations and 8 inclinations; and built using the *hard-coded* structure with the AIC score.

Table 6.2: Results of Experiment 3.

Object	*Initial positions*	*Mean error X*	*Mean error Y*	*Mean number of movements*	*Mean action smoothness value*
Cylinder	Demonstrations	1	1.1	13	1
	Generalization	8	6.7	13	0.9
Tea box	Demonstrations	1	1.4	10	1.1
	Generalization	0.7	1.5	9	1.03

previous Experiment, i.e. *linear* distance, with 16 bins for the *orientation* and 16 bins for the *inclination*. The obtained results show a clear difference between using the *hard-coded* structure and the structure learning methods.

For the *pushing* experiment, the final position of the cylinder is very close to the expected position using the *hard-coded* structure combined with any of the scores. This is depicted in Figures 6.13, a and 6.12. However, the structure learning methods have not been able to identify the proper causal relations among the random variables (see Annex A). And thus the mean error of the final position of the cylinder using these methods is high.

The mean smoothness value of the *hard-coded* approach generates very smooth trajectories and a constant number of movements using any of the scores. The trajectories generated by the learning methods are also smooth with a constant number of movements because they are mainly straight lines from the initial positions to the cylinder.

Experiment 3: Skill Generalization A^2L shows a very good generalization capability to *push* the tea box, similar to those obtained from the initial positions used in the demonstrations. Conversely, the results obtained *pushing* the cylinder are poor. The obtained results are clearly connected to the precision needed to *push* the object, because the method looses some accuracy while executing the action.

After the execution of the Experiments 1, 2 and 3 there is a combination of discretization and structure learning approach which we have selected to validate building other skills and using them to perform tasks:

- *Linear* distance
- 16 bins for the *orientation*
- 16 bins for the *inclination*
- *Hard-coded* approach
- AIC score

Figures 6.14a, 6.14b and 6.14c show more details about this combination. The actions inferred by the coarse- and fine-grained action generators are easy to observe based on their length, i.e. longer and shorter, respectively. The different colors make reference to the probability values computed by the action generators: *light violet* (not present) means that the generator has very little knowledge to infer the movement, *violet* means that the generator has a good knowledge to infer the movement (as most of the *push* movements), *magenta* means that the generator has a high knowledge to infer the movement (as the movements close to the object while *grasping*, and *black* that the generator only has one possibility result inferring the next movement. The *pushing* actions are smooth, first, aligning to the object position, and then approaching in a straight line. Conversely, the *grasping* actions are less smooth, although the fine-grained action generator has a high knowledge of the approximation to infer, producing a high success ratio. Regarding the generated structures, it is relevant to underline that they handle the same contextual states, even when they are not necessary, as *grasped* and *next_openness* for *pushing*.

Experiment 4: Solving a Maze The results obtained for both mazes are depicted in Figures 6.15 and 6.16. In both cases, the robot was able to solve the maze, showing precision on the *pushing* actions, confirming the results obtained in the Experiments 1 and 2. A^2L has built skills that can reproduce different effects, and thus they can be used in different tasks. Besides, the reproduced effects can be combined.

In Figure 6.15, the figure in J shows the actions executed between two screenshots. These actions are: (A-B) the robot *sets* the end-effector *behind* the cylinder, (B-C) the robot *pushes* the cylinder *far*, (C-D) the robot *sets* the end-effector *at the left* of the cylinder, (D-E) the robot *pushes* the cylinder to the *right*, (E-F) the robot *sets* the end-effector *in front* of the cylinder, (F-G) the robot *pushes* the cylinder *close*, (G-H) the robot *sets* the end-effector *at the right* of the cylinder, (F-G) the robot *pushes* the cylinder to the *left*. All the actions are accurate, except setting the arm *at the back* (B) and *in front* of the cylinder (F), due to reaching the kinematic limits of the right arm of the robot.

In Figure 6.16, the figure in F shows the actions executed between two screenshots. These actions are: (A) the robot *pushes* the cylinder to the *right*, (A-B) the robot *sets* the end-effector *at the back* of the cylinder, (B-C) the robot *pushes* the cylinder *far*, (C-D) the robot *sets* the end-effector *at the right* of the cylinder, (D-E) the robot *pushes* the cylinder to the *right*, a different distance than A. Similarly, in B the actions is not very precise due to reaching the kinematic limits of the arm of the robot.

Experiment 5: Heating a Croissant Figures 6.18 and 6.19 show the trajectories obtained in the tests, i.e. executing actions with spatio-temporal perturbations. In both cases the skills are robust, reproducing the expected effect. In Figure 6.18, the action A-C shows a curve of the action adapting to the change in the croissant position. Similarly, at the beginning of the action C-F there is an abrupt change in

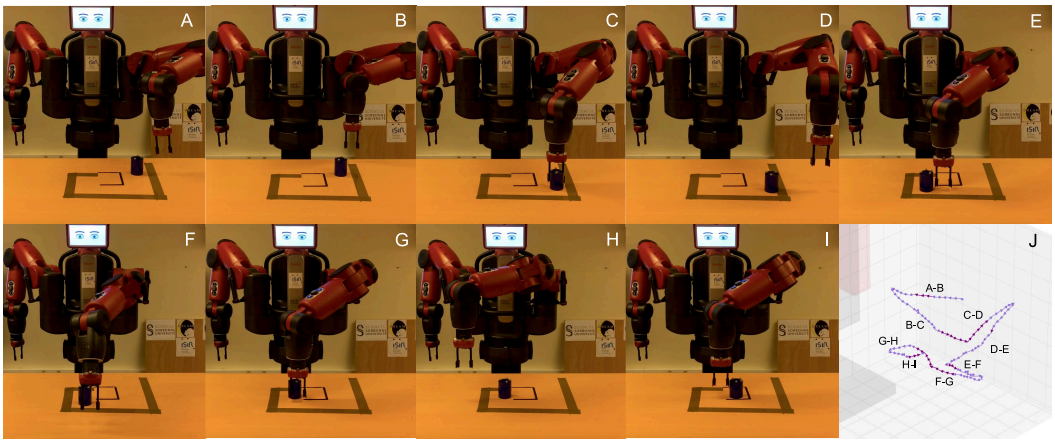


Figure 6.15: Actions solving the first maze. From A to I, screenshots of the execution of the task. Finally, in J, virtual representation of the actions executed.

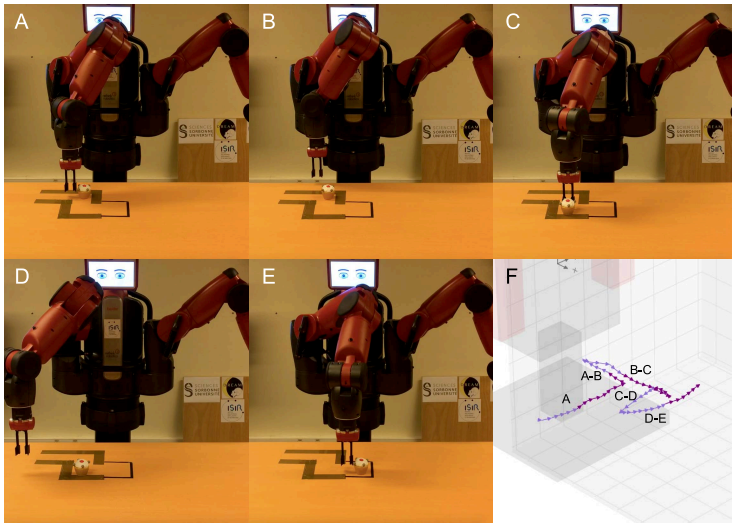


Figure 6.16: Actions solving the second maze. From A to E, screenshots of the execution of the task. Finally, in F, virtual representation of the actions executed.

Figure 6.17: Actions of a successful execution of the task heating the croissant. From A to E, screenshots *pushing* the button, *grasping* the croissant, and *putting it* into the pan. In F, virtual representation of these actions. From G to H, screenshots *grasping* again the croissant, *putting it* back into the dish, and pressing the button. In L, virtual representation of these actions.

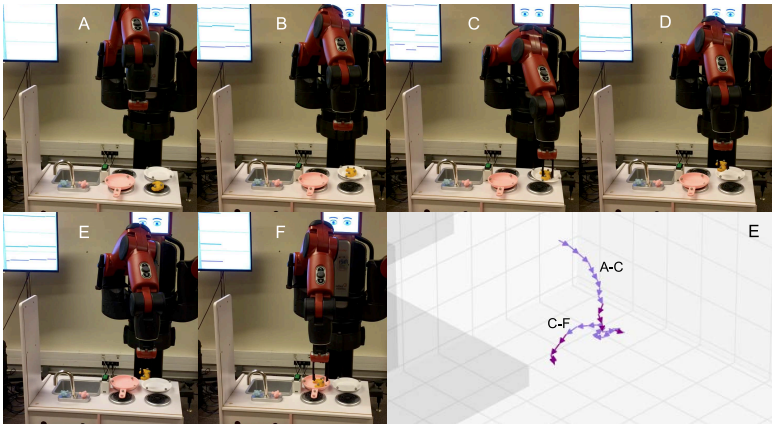


Figure 6.18: Actions of the first spatio-temporal test, composed of *grasping* the croissant and *releasing* it into the pan. In this case, the spatial perturbation consists in changing the position of the pan during the *release* action.

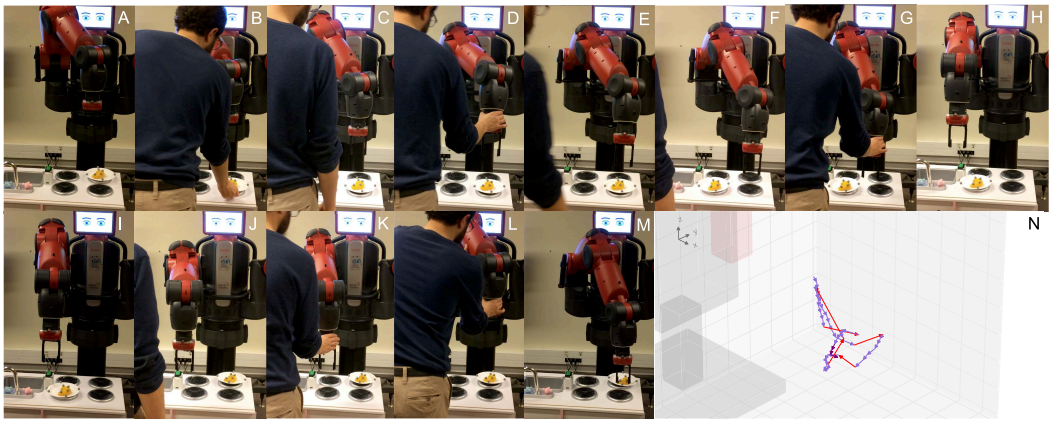


Figure 6.19: Actions of the second spatio-temporal test, in which the robot tries to *grasp* the croissant. Both spatial and temporal perturbations are present, changing the position of the croissant, and externally moving the robot's end-effector, respectivelly. The orange arrows represent externally generated long movements of the end-effector.

the action direction when the pan position is changed. In Figure 6.19, the external changes in the position of the end-effector are identified as orange arrows. After these changes the action generator continuous inferring movements adapted to the next low-level contextual state, i.e. the robot-object relation state.

The results obtained for the multi-step experiment are the following: on the one hand, Figure 6.20 shows the number of effects reproduced in 10 runs of the experiment. 6 runs completely reproduced all the effects, whereas 3 times the robot was not able to properly release the croissant from the pan to the dish. The *release* actions mainly failed because these actions did not move the end-effector high enough and the markers of the croissant and the pan touched, displacing the pan and the dish, and/or making the croissant fall from the end-effector. On the other

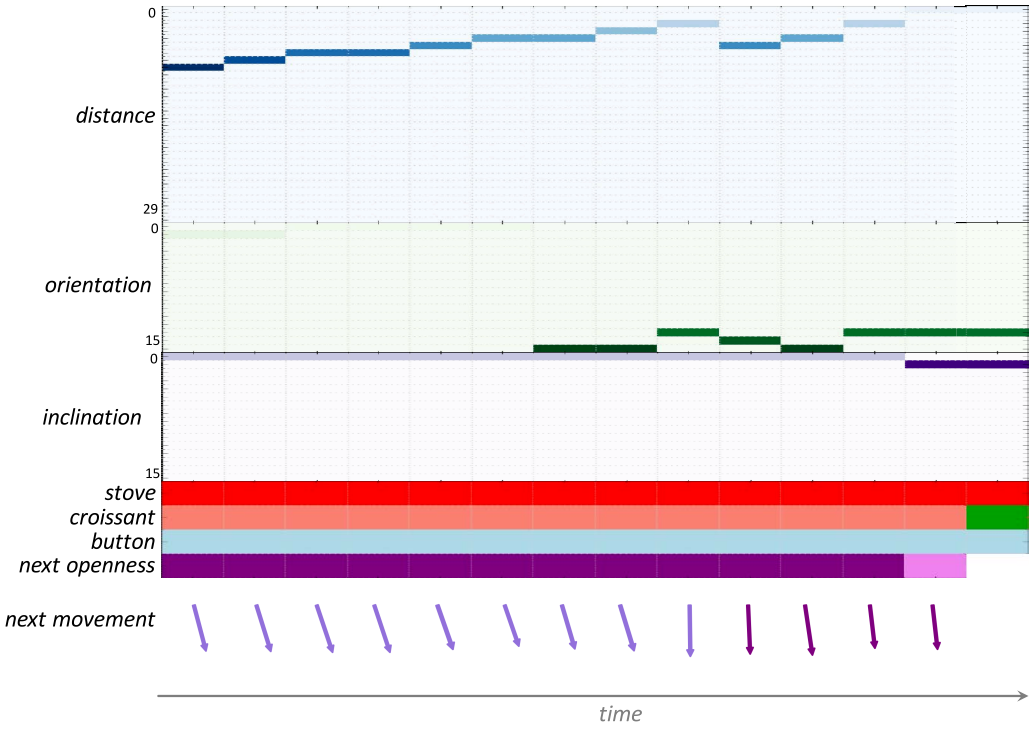


Figure 6.20: Trace of the action G-H of Figure 6.17, composed of context states at consecutive instants of time during the action (from left to right). An instant of time represents an iteration of the *perception-action cycle* of the skill. The trace is composed of both the low-level and high-level contextual states used by the skill during the execution of the action. The low-level states are composed of the *distance*, *orientation* and *inclination* of the robot-object relation state. In these states the change of values when the end-effector approaches the croissant is clearly perceptible. Also, it is straightforward to identify the movement in which the fine-grained action generator starts inferring the next movement observing the change of probability, i.e. color, in the next movement to execute. The high-level states represent the states of the stove, the gripper and the button. These states remain stable during the execution of the action. Finally, it is also available the continuous next movement and the next openness value of the gripper, i.e. *open* in purple and *closed* in light purple. The gripper value changes at the last movement, when the action generator infers that the croissant must be grasped. The movements show an initial curvature, and get straight when approaching the object. At the last instant of time (on the right) neither a movement nor a next gripper value are inferred because the stop condition is reached, i.e. the croissant is grasped, in green.

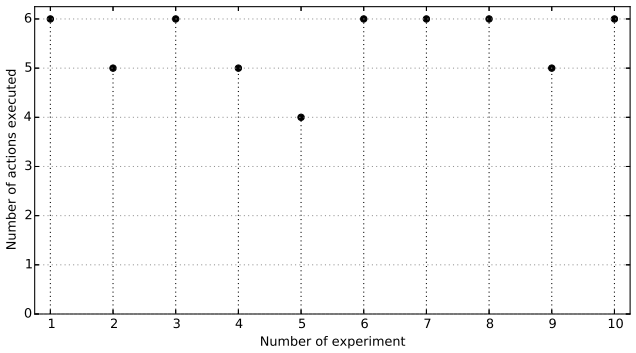


Figure 6.21: Results of the Experiment 5, heating croissant. The Y axis represents the number of action successfully executed of those listed in Experiment 5 (see Section 6.5). Ten runs have been executed. A run is successful if the 6 actions are executed.

hand, Figure 6.17 shows the actions of a successful execution of the task heating the croissant. In general, the actions *pressing* the button (A-B, and J-K) are quite accurate. Also, the *grasping* actions are quite robust, although once the action failed due to the orientation of the croissant. The *release* action from the dish to the pan is composed of very high initial movements, avoiding touching the dish. However, from the pan to the dish, the height is lower, producing the objects to touch each other.

It is relevant to mention that after E the croissant is cold. The task planner tries to execute the *grasping* action, G-H, but the action generator is not able to infer a movement, because the corresponding skill has been built with the croissant state as *mid temperature*. Once this level is reached, *salmon* in the Figure, the *grasp* action starts. The trace of the *grasp* action G-H is available in Figure 6.21.

6.6 Conclusions and Open Questions

This chapter focused on extending the skill building capacities of A^2L in order to scale up their use to our daily environments. The action generator of a skill infers an action to reproduce an effect on an object given a context. An action is a sequence of movements. Contexts are composed of low-level and high-level states. Robot actions must be adapted to both types of context states simultaneously. Adapting to low-level states means that actions must be robust to spatio-temporal perturbations. To that end, movements are generated by a vector field created using a DS. More precisely, the method relies on a *diffeomorphism* to generate the vector field. Movements are continuous vectors representing the next displacement of the robot's end-effector to reproduce an effect on an object. When running the skill, the corresponding action generator infers, in a closed loop, movements adapted to the object position. This position is relative to the robot's end-effector. Adapting

to high-level contextual states means that the robot has to change its behavior based on the values of these states. High-level states can be continuous, but they are commonly discrete. In order to handle discrete values *low-level and high-level context states*, and *movements* are discretized. *Effects* are already discrete. Thus, a discretization configuration is necessary.

Contexts are acquired during the interaction of a robot with an object. Therefore, a dataset of interactions must be generated in order to learn skills. In this chapter, different robot-object interactions are directly demonstrated to the robot by an external agent. In this case, the agent moves the robot's end-effector touching an object reproducing an effect (LfD).

In the current work, a study is performed in order to select the best parametrization to build a *push to the right* and a *grasp* skill to reproduce an effect. This study is composed of the identification of the most adequate discretization configuration and the structure of the BN to use. Then, the selected parametrization is used to build a set of diverse skills, which are validated in several experiments performing tasks with different objects. The assessment of the built skills is directly performed by the physical Baxter.

The obtained results show that our method builds skills that can be used to perform tasks using objects. Besides, the skills built using the method are *transparent*, i.e. its behavior can be traced. Regarding the low-level contextual states, the inference of a movement at each instant of time computed by a vector field allows the robot to adapt its actions to changes in the position of an object, and to perform these actions for relative object positions unobserved during the demonstrations. Given an interaction reproducing an effect, using the best skill parametrization the precision of the inferred actions allows the robot to reproduce the expect effect under similar relative object position w.r.t. the end-effector. The precision decreases from unobserved relative positions, but the robot is able to reproduce the effect in most of the cases. A key feature of the method is the capability to combine the knowledge acquired from different interactions reproducing the same effect. This capability allows the execution of a developmental approach, as the iterative process of A2L, in which the robot through trial-and-error acquires the required knowledge to reproduce an action under different contexts. For example, the robot can learn to *push* the cylinder from different initial positions. Regarding the high-level contextual states, the robot performs actions only under certain high-level contextual states. For example, the robot turns a stove *on* only if it is *off*.

In our experiments we built two actions planners with different levels of accuracy. For both of them a unique discretization configuration was needed to *push* and *grasp* the cylinder. These skills have only in common the robot's end-effector approaches to the object without a specific velocity. The same configuration was later used for other skills sharing this feature, as *press* or *release*. This means that it is not necessary to identify a configuration discretization for each skill, but for a group of skills sharing global features, as approaching or moving away the end-effector from an object. Therefore, for a single discretization configuration different tasks can be performed.

Similarly to the simulated Baxter experiment in Chapter 5, the different structure learning methods and scores have not been able to properly identify the conditional dependencies needed to learn action generators reproducing the effects. In this case, the size of the repertoire of movements and contextual states used for learning the generators has around 1000 examples. It is clearly insufficient for these algorithms. Fortunately, the actions generators learned based on the *hard-coded* structure are able to reproduce different effects, as *pushing* or *grasping*.

The output of the execution of the skills by our method is a trace of the contextual states provided to the action generator, the inferred movement and the result of the action at different instants of time of the execution. This trace makes transparent the internal skill behavior. The possibility to understand the behavior of a skill can be exploited in higher-level stages to identify behavioral regularities, for instance to perform transfer learning and skill generalization. Besides, it could be used in a developmental process to generate and to update the provided *a priori* information needed for the model to execute, mainly the discretization configuration and the conditional dependencies.

It is also relevant to mention that the obtained results confirm the consideration that we did suggesting that *distance*, the *orientation* and the *inclination* can be considered are independent.

Chapter 7

Concluding Remarks

7.1 Discussion and Perspectives

The main objective of this manuscript was to endow a robot with the capacity to perform tasks in environments with features similar to those of our daily environments. Through an exploration of the environment the robot should autonomously identify the interactions to learn to reproduce effects on objects. We have demonstrated that using A^2L a robot is able to autonomously generate a dataset of interactions through interactions with the environment. These interactions allowed the robot to perform tasks adapted to its context.

We have tried to provide the less possible *a priori* knowledge to the method, to increase its adaptability and generalization capabilities. However, during the development of the method we have identified several constraints. The acquisition of interactions to build skill through an exploration of the environment requires a minimal *a priori* knowledge to drive the robot actions. The use of random actions generates a lot of noise that complicates the skill generation. Directly providing to the robot the ideal interactions, for example through demonstrations, allows us to build methods that generate good results. However, this approach requires a supervisor to show the gestures to reproduce, i.e. *a priori* knowledge. A combination of both techniques could be a better solution, as in Ugur et al. (2015a). In this work, a caretaker helps a robot to grasp a mug handle physically modifying the on-going end-effector trajectory. Although it is not included in this manuscript, this is a clear next step to combine the iterative process generating interactions with the skill building presented in Chapter 6.

The learning of the action generators of the skills relies on (i) the conditional dependencies among the *effect* to reproduce, the *context* and the next *movement* to perform, and on (ii) a discretization of the context. On the one hand, the different structure learning methods and scores used to learn the structure of the BN representing the action generators need a very high number of interactions to identify the corresponding dependencies. Generating such a number of interactions with a robot is very complex. In this case, the use of a *hard-coded* structure has been useful to reproduce different effects, as *pushing* and *grasping*. Fortunately, the *a priori knowledge* used to build this structure is very low, because it only refers to connect the action inferred by the skill, e.g. a movement or a gripper open/close action, to the remaining available information. Therefore, this structure can be used in many different situations the robot has to face. On the other hand, the carried out experiments have demonstrated that a unique discretization configuration can

be used to perform different actions. The objects used for the experiments are available in our daily environments, and have different shapes and sizes. However, they also have features in common, as being rigid objects. Running the skills to reproduce the created skills in other types of objects, e.g. paper tissue, could require a different discretization configuration. Besides, this configuration could be learned in a developmental fashion, and adapted to the type of object the robot has to interact with. A possible improvement to the method would be using BNs that can handle continuous and discrete variables, as Lauritzen (1992). However, these techniques should have a very fast inference capacity, contrary to what Osorio suggests (Osório et al., 2010).

Also, *a priori* knowledge has been added in order to acquire the position of an object. This has been one of the most challenging aspects of the execution of the experiments. We have tested different techniques with different *a priori* knowledge, e.g. blob, SIFT, QR codes, point cloud, and found positive features and drawbacks in all of them. In the current manuscript, we have added markers to the objects, and use a motion capture system to track their position. We have to define the exact part of the object provided to the robot as its position. In this case, we have always provided the center of mass of the object. The use of this approach is a significant limitation for the context acquisition in our daily environments. Besides, during the task performance the markers produce unexpected and unrealistic object contacts. A desired method would allow the generation of object models from visual perception, and their tracking robust to occlusions, as our visual library[1].

The current status of the method provides a lot of room for improvements. For example, the DS can be used to add more capacities to the method. In the current implementation only the *distance* and *orientation* are used from the vector provided by the DS. A necessary improvement to the method is using the *velocities* and *accelerations* provided by the vector field. This would allow the robot to execute actions at different velocities, to execute actions as *poke*. Also w.r.t. the DS, the use of *repellers* in the vector fields generated would provide to the method an obstacle avoidance capacity, allowing the use of the method in more realistic environments.

The execution of the adaptive skills built by A^2L relies on the *perception-action cycle* presented in Figure 6.3.2. In this cycle, after the execution of a movement the contextual states are acquired and the next movement to execute is inferred. This approach generates reactive actions, adapting to the current robot's context. In the executed experiments, both action generators compute a movement in between 0.4 and 0.6 seconds. There is evidence that this computation lasts at least 0.2 seconds in adults, and much longer in infants (Von Hofsten, 2013). Von Hofsten suggests that humans use *predictive control* to predict the next changes in the context. And thus infer the next action based on these predictions, rather than on the actual context. This approach can be applied to the skill building presented in this manuscript.

In the same vein, another possible improvement to the method would be the capability to connect a sequence of contexts in a period of time. This would give to

[1] https://github.com/cmaestre/pcl_tracking

the method the possibility of inferring a sequence of movements, and it would also reduce the time of computation. A possible method providing this feature would be the use of Dynamic Bayesian Networks (Mugan and Kuipers, 2012).

Also, the number of context states that a BN can handle are limited, because of the *curse of dimensionality* related to the generation of the tables representing the conditional probabilities. This constraint has been already solved in Goncalves et al. (2014a) reducing the dimensionality of the information provided to the BN using the Principal Component Analysis (PCA) technique. This approach would allow the BN to handle more information. For example, the orientation of both the end-effector and the object, or higher-level abstract information, as the idea of *danger*. Also, the BN could handle contextual states of more than one object at a time, building skills inferring actions using several objects, e.g. multi-object interactions. However, the use of PCA or other dimensionality reduction techniques could complicate the definition of a *hard-coded* structure.

7.2 Conclusion

In this manuscript, we have proposed a method named Adaptive Affordance Learning (A^2L) that autonomously builds skills to reproduce effects on objects. Given a context, a skill infers and executes an action adapted to the object position reproducing an effect on an object. A context is composed of low-level contextual states, related to the execution of an action, e.g. an object position; and high-level contextual states representing higher level concepts of the objects, e.g. an object color.

Skills are built based on a dataset of interactions of the robot with an object. In order to acquire these interactions, we have presented in Chapter 4 a method named Novelty-driven Evolutionary Babbling (NovEB), designed to perform a task-agnostic exploration of an environment. Its main feature is to look for actions that maximize novelty in the raw sensorimotor space. It is based on Novelty Search, which relies on Evolutionary Algorithms driven by a behavior novelty criterion. Although the method has focused its exploration in regions that lead to the generation of interactions, the method runs in static environments that reset after each action to obtain a good performance. This constraint makes very difficult the execution of the method by a physical robot. Later, in Chapter 5 we have introduced the iterative process of A^2L, which generates a dataset of interactions, used to build the skills, through interactions of the robot with its environment. The process consists of three phases, executed in an iterative fashion: first, an exploration of the robot's environment has been performed based on random actions. The result of this exploration was a dataset of interactions. Then, a skill was built based on these interactions, which has been later validated reproducing a set of effects on an object.

The building of skills has been introduced in Chapter 5. The skills were action generators implemented as Bayesian Networks. In that chapter, because of the use

of random actions during the exploration of the environment, the actions inferred by the skills were discrete and constrained to a 2D Cartesian space. Besides, they were executed in an open loop, and thus the object position could not change during the actions execution. These constraints have been addressed in Chapter 6 building skills using a Dynamical System. Actions inferred by these skills were robust to spatio-temporal perturbations because each movement of the end-effector is generated as a continuous vector. In the carried out experiments the position of an object could change independently from the robot's actions.

These actions are also adapted to high-level contextual changes. High-level states can be continuous, but they are commonly discrete. In order to adapt to low-level and high-level contextual states these are discretized. Therefore, the Bayesian Networks representing the actions generators receive the next effect to reproduce and discrete contextual states, and infer discrete movements.

The output of the execution of the skills by our method is a trace of the contextual states provided to the action generator, the inferred movement and the result of the action at different instants of time of the execution. This trace makes transparent the internal skill behavior, which can be useful in higher-level stages to identify behavioral regularities to exploit transfer learning and generalization.

Acronyms

A^{2}L Adaptive Affordance Learning

BN Bayesian Network

CPD Conditional Probability Distribution

DoF Degrees of Freedom

DS Dynamical System

HL Contextual High-Level Skills

LfD Learning From Demonstration

LL Contextual Low-Level Skills

MP Motion Primitive

NovEB Novelty-driven Evolutionary Babbling

PDDL Planning Domain Definition Language

Bibliography

Adolph, K. E. and Berger, S. E. (2015). Physical and Motor Development. (Cité en page 13.)

Adolph, K. E. and Kretch, K. S. (2015). Gibson's Theory of Perceptual Learning. *International Encyclopedia of the Social & Behavioral Sciences*, 10:127–134. (Cité en page 14.)

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactionson Automatic Control*, 19(6):716–723. (Cité en page 33.)

Antunes, A., Jamone, L., Saponaro, G., Bernardino, A., and Ventura, R. (2016). From human instructions to robot actions: Formulation of goals, affordances and probabilistic planning. *Proceedings - IEEE International Conference on Robotics and Automation*, 2016-June(May):5449–5454. (Cité en pages 21 et 25.)

Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., and Yoshida, C. (2009). Cognitive Developmental Robotics : A Survey. *IEEE Transactions on Autonomous Mental Development*, 1(1):12–34. (Cité en pages 2 et 13.)

Asada, M., Macdorman, K. F., Ishiguro, H., and Kuniyoshi, Y. (2001). Cognitive developmental robotics as a new paradigm for the design of humanoid robots. 37:185–193. (Cité en pages 2 et 13.)

Atkeson, C. G., Babu, B. P. W., Banerjee, N., Berenson, D., Bove, C. P., Cui, X., DeDonato, M., Du, R., Feng, S., Franklin, P., Gennert, M. A., Graff, J. P., He, P., Jaeger, A., Kim, J., Knoedler, K., Li, L., Liu, C., Long, X., Padir, T., Polido, F., Tighe, G. G., and Xinjilefu, X. (2015). What Happened at the DARPA Robotics Challenge and Why. *DRC Finals Special Issue of the Journal of Field Robotics*. (Cité en page 1.)

Baldassarre, G. and Mirolli, M. (2013). *Intrinsically Motivated Learning in Natural and Artificial Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg. (Cité en page 14.)

Bar-Yam, Y. (2004). A mathematical theory of strong emergence using multiscale variety. *Complexity*, 9(6):15–24. (Cité en page 36.)

Baranes, A. and Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73. (Cité en pages 15 et 50.)

Barto, A. G. (2004). Intrinsically motivated learning of hierarchical collections of skills. In *3rd International Conference Development Learning*, pages 112–119. (Cité en page 14.)

Beer, R. D. (1995). A dynamical systems perspective interaction on agent-environment. *Artificial Intellegence*, 72:173–215. (Cité en page 1.)

Bellman, R. (1966). Dynamic Programming. *Science*, 153(3731):34–37. (Cité en page 31.)

Billard, A. G. and Calinon, S. (2016). Handbook of Robotics Chapter 59 : Robot Programming by Demonstration. *Robotics*, 48:1371–1394. (Cité en pages 10, 15 et 80.)

Buneo, C. A., Jarvis, M. R., Batista, A. P., and Andersen, R. A. (2002). Direct visuomotor transformations for reaching. *Nature*, 416(6881):632–636. (Cité en page 13.)

Buntine, W. (1991). Theory refinement on Bayesian networks. *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 52–60. (Cité en page 33.)

Calinon, S. (2016). A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29. (Cité en pages 16 et 18.)

Calinon, S., D'halluin, F., Sauser, E. L., Caldwell, D. G., and Billard, A. G. (2010). A probabilistic approach based on dynamical systems to learn and reproduce gestures by imitation. *IEEE Robotics and Automation Magazine*, 17(January 2016):44–54. (Cité en pages 3, 16, 17, 58, 92 et 104.)

Calinon, S., Guenter, F., and Billard, A. G. (2007). On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(2):286–298. (Cité en pages 16 et 17.)

Calinon, S., Pistillo, A., and Caldwell, D. G. (2011). Encoding the time and space constraints of a task in explicit-duration hidden Markov model. In *IEEE International Conference on Intelligent Robots and Systems*, pages 3413–3418. (Cité en pages 16, 17, 58 et 104.)

Cangelosi, A., Schlesinger, M., and Smith, L. B. (2015). *Developmental Robotics: From Babies to Robots*. MIT Press. (Cité en pages 2 et 13.)

Chavez-Garcia, R. O., Andries, M., Luce-Vayrac, P., and Chatila, R. (2017). Discovering and Manipulating Affordances. *International Symposium on Experimental Robotics*, (Iser):679–691. (Cité en page 21.)

Chavez-Garcia, R. O., Luce-Vayrac, P., and Chatila, R. (2016). Discovering affordances through perception and manipulation. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem:3959–3964. (Cité en page 22.)

Chemero, A. (2003). An Outline of a Theory of Affordances. *Ecological Psychology*, 15(2):181–195. (Cité en page 19.)

Chickering, D., Geiger, D., and Heckerman, D. (1995). Learning Bayesian networks: Search methods and experimental results. *Proceedings of Fifth Conference on Artificial Intelligence and Statistics*, pages 112–128. (Cité en pages 34, 69 et 103.)

Chickering, D. M. (2002). Optimal structure identification with greedy search. *The Journal of Machine Learning Research*, 3:507–554. (Cité en page 33.)

Chickering, D. M., Geiger, D., and Heckerman, D. (1994). Learning Bayesian networks is NP-hard. *Microsoft Research*, MSR-TR-94-(1999):17–94. (Cité en page 33.)

Cooper, G. F. and Herskovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347. (Cité en pages 34, 74 et 103.)

Cully, A. and Mouret, J.-B. (2015). Evolving a Behavioral Repertoire for a Walking Robot. *Evolutionary computation*, (2005). (Cité en pages 28 et 30.)

Darwin, C. (1872). *The origin of species by means of natural selection; or, The preservation of favored races in the struggle for life*, volume 78. (Cité en page 28.)

Dearden, A. and Demiris, Y. (2005). Learning forward models for robots. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1440–1445. (Cité en pages 20 et 22.)

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197. (Cité en page 42.)

Dehban, A., Jamone, L., and Kampff, A. R. (2016). Denoising Auto-encoders for Learning of Objects and Tools Affordances in Continuous Space. pages 1–6. (Cité en pages 21 et 23.)

Dehban, A., Jamone, L., and Kampff, A. R. (2017). A Deep Probabilistic Framework for Heterogeneous Self-Supervised Learning of Affordances. *Humanoids 2017*. (Cité en pages 21 et 23.)

Demiris, Y. and Dearden, A. (2005). From motor babbling to hierarchical learning by imitation: a robot developmental pathway. (Cité en pages 14, 20, 22 et 56.)

Doncieux, S., Bredeche, N., Mouret, J.-B., and Eiben, A. E. (2015a). Evolutionary robotics: what, why, and where to. *Frontiers in Robotics and AI*, 2. (Cité en page 30.)

Doncieux, S., Duro, R., Prieto, A., Bellas, F., Stulp, F., Filliat, D., Hospedales, T., Heinerman, J., Haasdijk, E., and Eiben, G. (2015b). Deliverable D6.1 - DREAM project. Technical report. (Cité en page 1.)

Doncieux, S. and Mouret, J.-B. (2014). Beyond black-box optimization: A review of selective pressures for evolutionary robotics. *Evolutionary Intelligence*, 7(2):71–93. (Cité en pages 30 et 51.)

Ebert, F., Finn, C., Lee, A. X., and Levine, S. (2017). Self-Supervised Visual Planning with Temporal Skip Connections. (CoRL):1–13. (Cité en pages 21 et 23.)

Eiben, A. E. and Smith, J. E. (2008). *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer. (Cité en page 28.)

Elsner, B. and Hommel, B. (2001). Effect anticipation and action control. *Journal of Experimental Psychology: Human Perception and Performance*, 27(1):229–240. (Cité en page 14.)

Finn, C., Goodfellow, I., and Levine, S. (2016). Unsupervised Learning for Physical Interaction through Video Prediction. (Cité en pages 21 et 23.)

Finn, C. and Levine, S. (2017). Deep visual foresight for planning robot motion. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2786–2793. (Cité en pages 21 et 23.)

Fischer, K. W. (1980). A theory of cognitive development: The control and construction of hierarchies of skills. *Psychological Review*, 87(6):477–531. (Cité en page 3.)

Fitzpatrick, P. and Metta, G. (2003). Grounding vision through experimental manipulation. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 361(1811):2165–2185. (Cité en pages 19 et 20.)

Fitzpatrick, P., Metta, G., Natale, L., Rao, S., and Sandini, G. (2003). Learning about objects through action-initial steps towards artificial cognition. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3140–3145. (Cité en pages 19 et 20.)

Forestier, S. and Oudeyer, P. Y. (2016). Modular active curiosity-driven discovery of tool use. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem:3965–3972. (Cité en page 15.)

Forestier, S. and Oudeyer, P. Y. (2017). Overlapping waves in tool use development: A curiosity-driven computational model. *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2016*, pages 238–245. (Cité en page 15.)

Forlizzi, J. and Disalvo, C. (2006). Service Robots in the Domestic Environment: A Study of the Roomba Vacuum in the Home. *Design*, 2006:258–265. (Cité en page 1.)

Gallivan, J. P., Adam McLean, D., Valyear, K. F., and Culham, J. C. (2013). Decoding the neural mechanisms of human tool use. *eLife*, 2013(2). (Cité en page 13.)

Geiger, D., Verma, T., and Pearl, J. (1990). Identifying independence in bayesian networks. (Cité en page 32.)

Gibson, E. J. (1969). Principles of perceptual learning and development. *Principles of perceptual learning and development.*, page 353. (Cité en page 14.)

Gibson, E. J. (1994). Has psychology a future? *Psychological Science*, 5(2):69–76. (Cité en page 14.)

Gibson, E. J. (2000). Perceptual Learning in Development: Some Basic Concepts. *Ecological Psychology*, 12(4):295–302. (Cité en page 8.)

Gibson, E. J. (2003). The World Is So Full of a Number of Things: On Specification and Perceptual Learning. *Ecological Psychology*, 15(4):283–287. (Cité en page 8.)

Gibson, J. J. (1966). The senses considered as perceptual systems. Houghton Mifflin, Boston. (Cité en pages 3, 18 et 36.)

Gibson, J. J. (1979). The Ecological Approach to Visual Perception. (Cité en page 36.)

Gibson, J. J. (1986). The Ecological Approach to Visual Perception. In *Chapter Eight The Theory of Affordances*, pages 127–136. (Cité en pages 3 et 19.)

Goncalves, A., Abrantes, J., Saponaro, G., Jamone, L., and Bernardino, A. (2014a). Learning intermediate object affordances: Towards the development of a tool concept. *IEEE ICDL-EPIROB 2014 - 4th Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics*, (October):482–488. (Cité en pages 21, 23 et 121.)

Goncalves, A., Saponaro, G., Jamone, L., and Bernardino, A. (2014b). Learning visual affordances of objects and tools through autonomous robot exploration. *IEEE ICARSC*, (April 2016):128–133. (Cité en pages 21 et 23.)

Gonzales, C., Torti, L., and Wuillemin, P. H. (2017). aGrUM: A graphical universal model framework. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10351 LNCS, pages 171–177. (Cité en page 64.)

Gribovskaya, E., Khansari-Zadeh, S. M., and Billard, A. G. (2011). Learning Non-linear Multivariate Dynamics of Motion in Robotic Manipulators. *The International Journal of Robotics Research*, 30(1):80–117. (Cité en page 5.)

Guerin, F., Krüger, N., and Kraft, D. (2013). A Survey of the Ontogeny of Tool Use : from Sensorimotor Experience to Planning. *Autonomous Mental Development, IEEE Transactions*, 5(1):18 – 45. (Cité en page 2.)

Haken, H. (1978). *Synergetics: An introduction.* (Cité en page 36.)

Hamming, R. (1950). Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160. (Cité en page 45.)

Hangl, S., Ugur, E., Szedmak, S., and Piater, J. (2016). Robotic playing for hierarchical complex skill learning. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem:2799–2804. (Cité en pages 21 et 22.)

Hart, S., Grupen, R., and Jensen, D. (2005). A Relational Representation for Procedural Task Knowledge. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 1280–1285. (Cité en pages 20 et 22.)

Heckerman, D. (1996). A Tutorial on Learning With Bayesian Networks. *Innovations in Bayesian Networks*, 1995(November):33–82. (Cité en page 33.)

Hermans, T., Li, F., Rehg, J. M., and Bobick, A. F. (2013). Learning contact locations for pushing and orienting unknown objects. *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 435–442. (Cité en pages 21 et 22.)

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. (Cité en page 23.)

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–73. (Cité en pages 16 et 17.)

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). Learning Attractor Landscapes for Learning Motor Primitives. *Advances in Neural Information Processing Systems 15 (NIPS2002)*, pages 1547–1554. (Cité en pages 7, 16, 17 et 80.)

Jain, R. and Inamura, T. (2011). Learning of Tool Affordances for autonomous tool manipulation. *2011 IEEE/SICE International Symposium on System Integration, SII 2011*, pages 814–819. (Cité en pages 21 et 23.)

Jain, R. and Inamura, T. (2013). Bayesian learning of tool affordances based on generalization of functional feature to estimate effects of unseen tools. *Artificial Life and Robotics*, 18(1-2):95–103. (Cité en pages 21 et 23.)

Jamone, L., Ugur, E., Cangelosi, A., Fadiga, L., Bernardino, A., Piater, J., and Santos-victor, J. (2016). Affordances in psychology, neuroscience and robotics: a survey. *IEEE Transactions on Cognitive and Developmental Systems*, (August):1–1. (Cité en pages 13, 18 et 19.)

Jones, K. S. (2003). What Is an Affordance? *Ecological Psychology*, 15(2):107–114. (Cité en page 19.)

Jonschkowski, R. and Brock, O. (2015). Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428. (Cité en page 8.)

Jonschkowski, R., Hafner, R., Scholz, J., and Riedmiller, M. (2017). PVEs: Position-Velocity Encoders for Unsupervised Learning of Structured State Representations. *arXiv preprint arXiv:1705.09805*. (Cité en page 8.)

Kandel, E. R., Schwartz, J. H., Jessell, T. M., Siegelbaum, S. A., and Hudspeth, A. (2014). *Principles of Neural Science, Fifth Edition*, volume 3. (Cité en page 13.)

Katok, A. and Hasselblatt, B. (1995). Introduction to the modern theory of dynamical systems. *Cambridge, Cambridge*. (Cité en page 34.)

Khansari-Zadeh, S. M. and Billard, A. G. (2011). Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957. (Cité en pages 16, 17 et 38.)

Khansari-Zadeh, S. M. and Billard, A. G. (2014). Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics and Autonomous Systems*, 62(6):752–765. (Cité en pages 16, 17 et 38.)

Kim, S., Shukla, A., and Billard, A. G. (2014). Catching objects in flight. *IEEE Transactions on Robotics*, 30(5):1049–1065. (Cité en pages 16 et 18.)

Kober, J., Muelling, K., Kroemer, O., Lampert, C. H., Scholkopf, B., and Peters, J. (2010). Movement templates for learning of hitting and batting. *2010 IEEE International Conference on Robotics and Automation*, pages 853–858. (Cité en page 16.)

Konidaris, G., Kaelbling, L., and Lozano-Perez, T. (2014). Constructing Symbolic Representations for High-Level Planning. In *Conference on Artificial Intelligence*, pages 1932–1940. (Cité en pages 24 et 39.)

Konidaris, G., Kaelbling, L. P., and Lozano-Perez, T. (2015). Symbol acquisition for probabilistic high-level planning. *IJCAI International Joint Conference on Artificial Intelligence*, 2015-Janua(Ijcai):3619–3627. (Cité en page 24.)

Konidaris, G., Kaelbling, L. P., and Lozano-perez, T. (2018). From Skills to Symbols : Learning Symbolic Representations for Abstract High-Level Planning. 61:1–72. (Cité en page 25.)

Kopicki, M., Zurek, S., Stolkin, R., Mörwald, T., and Wyatt, J. (2011). Learning to predict how rigid objects behave under simple manipulation. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5722–5729. (Cité en pages 20 et 22.)

Kroemer, O., Ugur, E., Oztop, E., and Peters, J. (2012). A Kernel-based approach to Direct Action Perception. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2605–2610. (Cité en pages 5, 16 et 18.)

Krotkov, E. (1995). Robotic Perception of Material. *Proceedings of the IJCAI*, (March):88–94. (Cité en pages 19 et 20.)

Krüger, N., Geib, C., Piater, J., Petrick, R., Steedman, M., Wörgötter, F., Ude, A., Asfour, T., Kraft, D., Omrčen, D., Agostini, A., and Dillmann, R. (2011). ObjectAction Complexes: Grounded abstractions of sensorymotor processes. *Robotics and Autonomous Systems*, 59(10):740–757. (Cité en page 21.)

Kugler, P. N., Scott Kelso, J. A., and Turvey, M. T. (1980). On The Concept of Coordinative Structures as Dissipative Structures. *Advances in Psychology*, 1(C):3–47. (Cité en page 36.)

Kugler, P. N. and Turvey, M. T. (1987). Information, natural law, and the self-assembly of rhythmic movement. *Resources for ecological psychology*, page 481. (Cité en pages 36 et 83.)

Kulić, D., Ott, C., Lee, D., Ishikawa, J., and Nakamura, Y. (2012). Incremental learning of full body motion primitives and their sequencing through human motion observation. *International Journal of Robotics Research*, 31(3):330–345. (Cité en page 15.)

Lang, T. and Toussaint, M. (2010). Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39:1–49. (Cité en pages 21 et 25.)

Lauritzen, S. L. (1992). Propagation of Probabilities, Means, and Variances in Mixed Graphical Association Models. *Journal of the American Statistical Association*, 87(420):1098–1108. (Cité en pages 7, 57 et 120.)

LeGoff, L. K., Mukhtar, G., Fur, P. H. L., and Doncieux, S. (2017). Segmenting objects through an autonomous agnostic exploration conducted by a robot. *Proceedings - 2017 1st IEEE International Conference on Robotic Computing, IRC 2017*, pages 284–291. (Cité en page 50.)

Lehman, J. and Stanley, K. O. (2011). Abandoning objectives: evolution through the search for novelty alone. *Evolutionary computation*, 19:189–223. (Cité en pages 10, 30, 41 et 43.)

Lesort, T. and Filliat, D. (2017). Unsupervised Deep Learning of State Representation using Robotic Priors. *International Conference on Learning Representation*, pages 1–9. (Cité en page 50.)

Lockman, J. (2000). A perception–action perspective on tool use development. *Child development*, 71(1):137–144. (Cité en page 3.)

Lopes, M., Melo, F. S., and Montesano, L. (2007). Affordance-based imitation learning in robots. In *IEEE International Conference on Intelligent Robots and Systems*, number August, pages 1015–1021. (Cité en pages 20 et 22.)

Lungarella, M., Metta, G., Pfeifer, R., and Sandini, G. (2003). Developmental robotics: a survey. *Connection Science*, 15(4):151–190. (Cité en pages 2 et 13.)

Madsen, A. L. and Jensen, F. V. (1999). Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1):203–245. (Cité en page 32.)

Maestre, C., Cully, A., Gonzales, C., and Doncieux, S. (2015). Bootstrapping interactions with objects from raw sensorimotor data: a Novelty Search based approach. In *ICDL-Epirob - International Conference on Development and Learning, Epirob*, page 6. (Cité en pages 10, 11 et 41.)

Maestre, C., Gonzales, C., and Doncieux, S. (2016). Bootstraping manipulation skills to learn affordances in open-ended environments. *Proceedings of the workshop Autonomous Perception: Applying Sensorimotor Contingencies and Predictive Processing to Developmental Robotics (ICDL-Epirob)*, (i):1–2. (Cité en pages 10 et 11.)

Maestre, C., Mukhtar, G., Gonzales, C., and Doncieux, S. (2017a). Context-Based Generation of Continuous Actions to Reproduce Effects on Objects. In *The Third International Workshop on Intrinsically Motivated Open-ended Learning (IMOL)*. (Cité en pages 10, 11 et 79.)

Maestre, C., Mukhtar, G., Gonzales, C., and Doncieux, S. (2017b). Iterative affordance learning with adaptive action generation. In *ICDL-Epirob - International Conference on Development and Learning, Epirob*, number 2. (Cité en pages 10, 11 et 53.)

May, S., Klodt, M., Rome, E., and Breithaupt, R. (2007). GPU-accelerated affordance cueing based on visual attention. *IEEE International Conference on Intelligent Robots and Systems*, pages 3385–3390. (Cité en pages 19 et 20.)

McDermott, D., Ghallab, M., Howe, A., and C (1998). PDDL - The Planning Domain Definition Language. *The AIPS-98 Planning Competition Committee*, page 27. (Cité en pages 24 et 39.)

Meeden, L. A. and Blank, D. S. (2006). Introduction to developmental robotics. *Connection Science*, 18(2):93–96. (Cité en pages 2 et 13.)

Meltzoff, A. N. and Moore, M. K. (1997). Explaining Facial Imitation: A Theoretical Model. *Early development & parenting*, 6(3-4):179–192. (Cité en page 14.)

Metta, G. and Fitzpatrick, P. (2003). Better Vision through Manipulation. *Adaptive Behavior*, 11(2):109–128. (Cité en pages 19 et 20.)

Montesano, L., Lopes, M., Bernardino, A., and Santos-victor, J. (2008). Learning Object Affordances: From Sensory–Motor Coordination to Imitation. *IEEE Transactions on Robotics*, 24(1):15–26. (Cité en pages 3, 7, 19, 20, 22, 23, 57 et 81.)

Moulin-Frier, C. and Oudeyer, P. Y. (2012). Curiosity-driven phonetic learning. *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL 2012*. (Cité en page 15.)

Mouret, J.-B. and Doncieux, S. (2010). Sferes v2: Evolvin' in the multi-core world. *IEEE Congress on Evolutionary Computation*, (2):1–8. (Cité en page 45.)

Muelling, K., Kober, J., Kroemer, O., and Peters, J. (2013). Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279. (Cité en pages 16 et 17.)

Mugan, J. (2011). *Autonomous Qualitative Learning of Distinctions and Actions in a Developing Agent*. PhD thesis. (Cité en pages 24, 25 et 39.)

Mugan, J. and Kuipers, B. J. (2012). Autonomous Learning of High-Level States and Actions in Continuous Environments. *IEEE Transactions on Autonomous Mental Development*, 4(1):70–86. (Cité en pages 14, 15, 21, 22, 25, 45, 48 et 121.)

Nelson, A. L., Barlow, G. J., and Doitsidis, L. (2009). Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4):345–370. (Cité en page 30.)

Newcombe, N. S. (2013). Cognitive development: Changing views of cognitive change. *Wiley Interdisciplinary Reviews: Cognitive Science*, 4(5):479–491. (Cité en page 2.)

Niekum, S., Osentoski, S., Konidaris, G., and Barto, A. G. (2012). Learning and generalization of complex tasks from unstructured demonstrations. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5239–5246. (Cité en page 15.)

Nilsson, N. J. (1981). Principles of Artificial Intelligence. (Cité en pages 24 et 39.)

Omrčen, D., Ude, A., and Kos, A. (2008). Learning primitive actions through object exploration. *2008 8th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2008*, pages 306–311. (Cité en page 21.)

Osório, P., Bernardino, A., Martinez-Cantin, R., and Santos-Victor, J. (2010). Gaussian Mixture Models for Affordance Learning using Bayesian Networks. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4432–4437. (Cité en pages 7, 20, 22, 57 et 120.)

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286. (Cité en pages 15 et 50.)

Paraschos, A., Daniel, C., Peters, J., and Neumann, G. (2013). Probabilistic Movement Primitives. *Neural Information Processing Systems*, pages 1–9. (Cité en pages 16 et 17.)

Paraschos, A., Daniel, C., Peters, J., and Neumann, G. (2017). Using probabilistic movement primitives in robotics. *Autonomous Robots*, pages 1–23. (Cité en pages 16 et 17.)

Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. (2009). Learning and Generalization of Motor Skills by Learning from Demonstration. *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, pages 1293–1298. (Cité en pages 16 et 17.)

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. (Cité en page 27.)

Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems. (Cité en pages 7, 30, 57 et 80.)

Pearl, J. and Verma, T. S. (1995). A theory of inferred causation. *Studies in Logic and the Foundations of Mathematics*, 134(C):789–811. (Cité en page 33.)

Pelleg, D. and Moore, A. (2000). X-means: Extending K-means with efficient estimation of the number of clusters. *Proceedings of the Seventeenth ICML*, pages 727–734. (Cité en page 23.)

Perrin, N. and Schlehuber-Caissier, P. (2016). Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems. *Systems and Control Letters*, 96:51–59. (Cité en pages 7 et 38.)

Piaget, J. and Cook, M. (1952). The origins of intelligence in children. page 419. (Cité en pages 2, 3 et 14.)

Puterman, M. (1990). *Markov decision processes*. (Cité en page 25.)

Ridge, B., Skočaj, D., and Leonardis, A. (2010). Self-supervised cross-modal online learning of basic object affordances for developmental robotic systems. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5047–5054. (Cité en pages 20 et 22.)

Rolf, M., Steil, J. J., and Gienger, M. (2010). Goal babbling permits direct learning of inverse kinematics. *IEEE Transactions on Autonomous Mental Development*, 2(3):216–229. (Cité en page 42.)

Rosenbaum, D. A. (2010). *Human Motor Control*. (Cité en pages 53 et 72.)

Ruiz, C. (2005). Illustration of the K2 algorithm for learning Bayes net structures. *Department of Computer Science*, pages 1–7. (Cité en page 34.)

Sahin, E., Cakmak, M., Dogar, M. R., Ugur, E., and Ucoluk, G. (2007). To Afford or Not to Afford: A New Formalization of Affordances Toward Affordance-Based Robot Control. *Adaptive Behavior*, 15(4):447–472. (Cité en pages 3 et 19.)

Sanderson, K. (2010). Mars rover Spirit (2003–10). *Nature*, 463(February):2010. (Cité en page 1.)

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242. (Cité en pages 7, 15, 39 et 80.)

Schillaci, G., Hafner, V. V., Lara, B., and Grosjean, M. (2013). Is that me? Sensorimotor learning and self-other distinction in robotics. In *ACM/IEEE International Conference on Human-Robot Interaction*, pages 223–224. (Cité en page 3.)

Schwarz, G. (1978). Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464. (Cité en page 33.)

Shachter, R. D. and Kenley, C. R. (1989). Gaussian Influence Diagrams. (Cité en page 22.)

Siciliano, B. and Khatib, O. (2008). *Springer Handbook of Robotics*. (Cité en page 34.)

Slotine, J.-J. E. and Li, W. (1991). *Applied Nonlinear Control*, volume 62. (Cité en page 18.)

Smith, L. B. (2009). Dynamic Systems, Sensorimotor Processes, and the Origins of Stability and Flexibility. In *Toward a Unified Theory of Development Connectionism and Dynamic System Theory Re-Consider*. (Cité en page 3.)

Snyder, L. H. (2000). Coordinate transformations for eye and arm movements in the brain. (Cité en page 13.)

Spelke, E. S. and Kinzler, K. D. (2007). Core knowledge. (Cité en page 102.)

Spirtes, P., Glymour, C., and Scheines, R. (2003). *Causation, Prediction, and Search*, volume 45. (Cité en page 33.)

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10:99–127. (Cité en page 43.)

Steedman, M. (2002). Plans, Affordances, And Combinatory Grammar. *Linguistics and Philosophy*, 25(5):723–753. (Cité en page 19.)

Stoffregen, T. A. (2003). Affordances as Properties of the Animal-Environment System. *Ecological Psychology*, 15(2):115–134. (Cité en page 19.)

Stoytchev, A. (2005). Toward Learning the Binding Affordances of Objects: A Behavior-Grounded Approach. *AAAI Symposium on Developmental Robotics*, pages 21–23. (Cité en pages 19 et 20.)

Stoytchev, A. (2009). Some Basic Principles of Developmental Robotics. 1(2):1–9. (Cité en pages 2 et 13.)

Stramandinoli, F., Tikhanoff, V., Pattacini, U., and Nori, F. (2017). Heteroscedastic Regression and Active Learning for Modeling Affordances in Humanoids. *IEEE Transactions on Cognitive and Developmental Systems*, X(X):1–1. (Cité en pages 7 et 57.)

Sutton, R. and Barto, A. G. (1998). Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054. (Cité en page 25.)

Szedmak, S., Ugur, E., and Piater, J. (2014). Knowledge propagation and relation learning for predicting action effects. In *IEEE International Conference on Intelligent Robots and Systems*, pages 623–629. Institute of Electrical and Electronics Engineers Inc. (Cité en pages 5, 21 et 23.)

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L. E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney, P. (2007). Stanley: The robot that won the DARPA Grand Challenge. *Springer Tracts in Advanced Robotics*, 36:1–43. (Cité en page 1.)

Tsamardinos, I., Brown, L. E., and Aliferis, C. F. (2006). The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78. (Cité en page 33.)

Turvey, M. T. (1992). Affordances and Prospective Control: An Outline of the Ontology. *Ecological Psychology*, 4(3):173–187. (Cité en page 19.)

Ugur, E., Nagai, Y., Celikkanat, H., and Oztop, E. (2015a). Parental scaffolding as a bootstrapping mechanism for learning grasp affordances and imitation skills. *Robotica*, 33(05):1163–1180. (Cité en page 119.)

Ugur, E., Nagai, Y., Sahin, E., and Oztop, E. (2015b). Staged development of robot skills: Behavior formation, affordance learning and imitation with motionese. *IEEE Transactions on Autonomous Mental Development*, 7(2):119–139. (Cité en pages 20 et 23.)

Ugur, E., Oztop, E., and Sahin, E. (2011). Goal emulation and planning in perceptual space using learned affordances. *Robotics and Autonomous Systems*, 59(7-8):580–595. (Cité en pages 20 et 22.)

Ugur, E. and Piater, J. (2015a). Bottom-Up Learning of Object Categories , Action Effects and Logical Rules : From Continuous Manipulative Exploration to Symbolic Planning. *IEEE International Conference on Robotics and Automation*, 4(4):1–1. (Cité en pages 21, 24, 39 et 81.)

Ugur, E. and Piater, J. (2015b). Refining discovered symbols with multi-step interaction experience. *IEEE-RAS International Conference on Humanoid Robots*, 2015-Decem:1007–1012. (Cité en pages 21 et 24.)

Ugur, E., Sahin, E., and Oztop, E. (2009). Affordance learning from range data for multi-step planning. *Perception*, pages 177–184. (Cité en pages 20 et 22.)

Ugur, E., Sahin, E., and Oztop, E. (2012). Self-discovery of motor primitives and learning grasp affordances. *IEEE International Conference on Intelligent Robots and Systems*, pages 3260–3267. (Cité en pages 20 et 23.)

van Dijk, S., van der Gaag, L. C., and Thierens, D. (2003). A Skeleton-Based Approach to Learning Bayesian Networks from Data. *Lecture Notes in Computer Science*, 2838:132–143. (Cité en page 33.)

Vernon, D. (2014). *Artificial Cognitive Systems. A Primer*, volume 53. (Cité en page 3.)

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion Pierre-Antoine Manzagol. *Journal of Machine Learning Research*, 11:3371–3408. (Cité en page 23.)

Von Hofsten, C. (2009). Action, the foundation for cognitive development. *Scandinavian Journal of Psychology*, 50(6):617–623. (Cité en page 13.)

Von Hofsten, C. (2013). *Action in Infancy: A Foundation for Cognitive Development*. Number February 2013. (Cité en pages 13 et 120.)

Want, S. C. and Harris, P. L. (2002). How do children ape? Applying concepts from the study of non-human primates to the developmental study of 'imitation' in children. (Cité en page 14.)

Warren, W. H. (1988). Action modes and laws of control for the visual guidance of action. *Advances in Psychology*, 50(C):339–379. (Cité en pages 36 et 83.)

Warren, W. H. (2006). The dynamics of perception and action. *Psychological Review*, 113(2):358–389. (Cité en pages 27, 35, 36, 37, 83 et 84.)

Warren, W. H., Fajen, B., and Belcher, D. (2010). Behavioral dynamics of steering, obstacle avoidance, and route selection. *Journal of Vision*, 1(3):184–184. (Cité en page 84.)

Weng, J. (2004). Developmental robotics: Theory and experiments. *International Journal of Humanoid Robotics*, 01(02):199–236. (Cité en pages 2 et 13.)

Wong, J. M. (2016). Towards Lifelong Self-Supervision: A Deep Learning Direction for Robotics. (Cité en page 23.)

Zauner, C. (2010). *Implementation and benchmarking of perceptual image hash functions*. PhD thesis, Upper Austria University of Applied Sciences. (Cité en page 45.)

Zech, P., Haller, S., Lakani, S. R., Ridge, B., Ugur, E., and Piater, J. (2017). Computationalmodels of affordance in robotics: a taxonomy and systematic classification. *Adaptive Behavior*, 25(5):235–271. (Cité en page 18.)

Zimmer, M. and Doncieux, S. (2017). Bootstrapping Q-Learning for Robotics from Neuro-Evolution Results. *IEEE Transactions on Cognitive and Developmental Systems*, X(JANUARY):1–1. (Cité en page 15.)

Appendix A

Annex A - Extra results of the Experiments of Chapter 6

Inferred actions

This section shows the trajectories inferred by A^2L in the Experiments 1, 2 and 3.

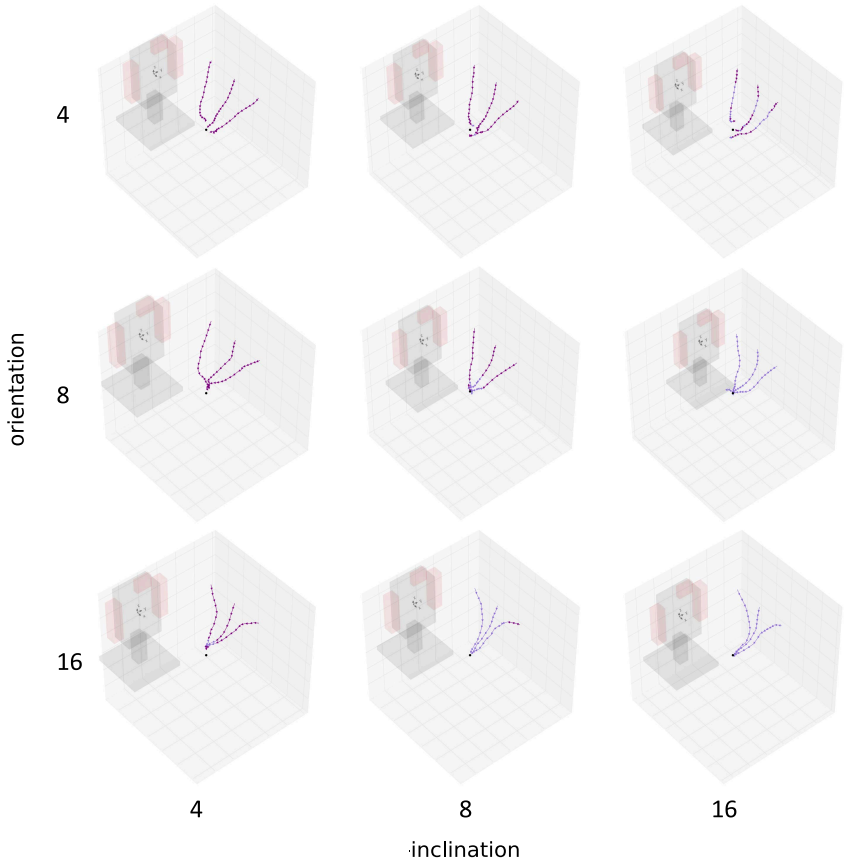


Figure A.1: Experiment 1.1 - Linear discretization

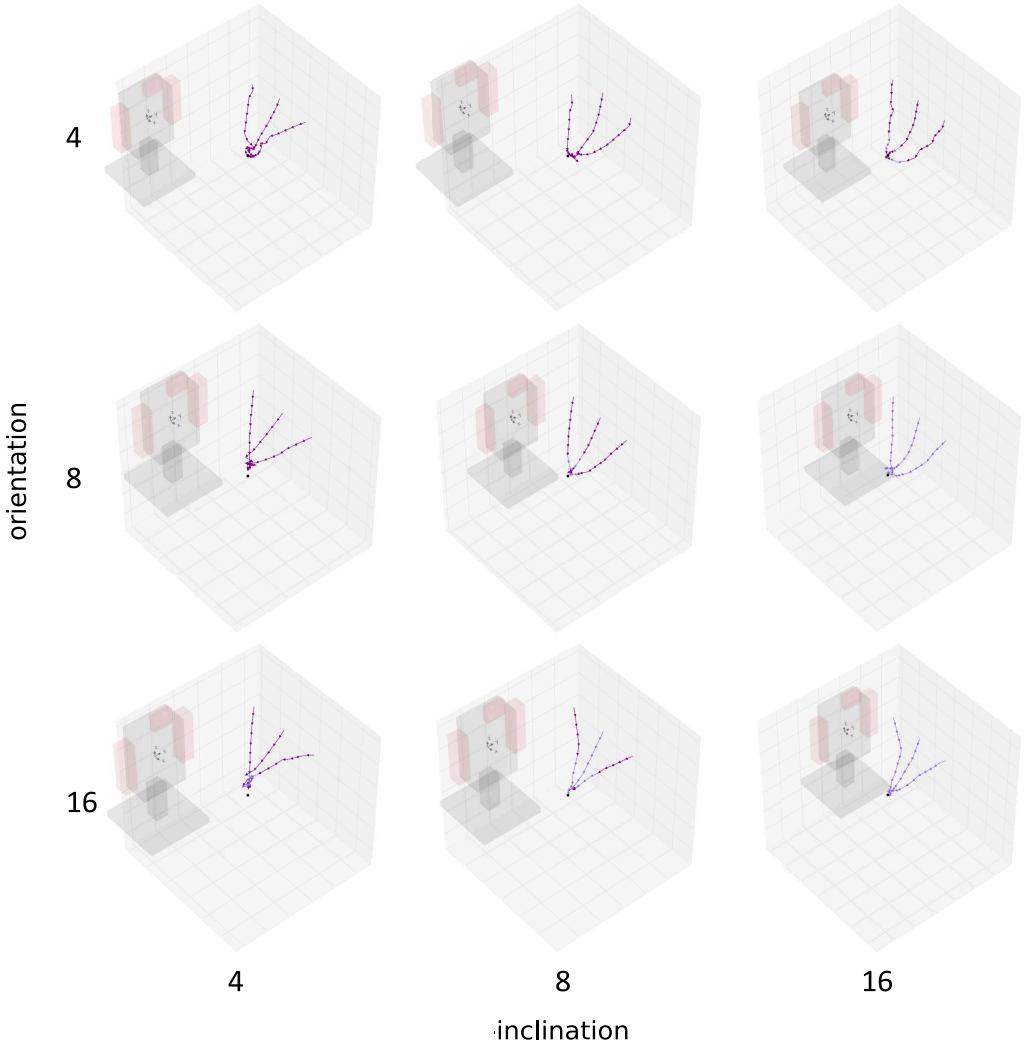


Figure A.2: Experiment 1.1 - Progressive discretization

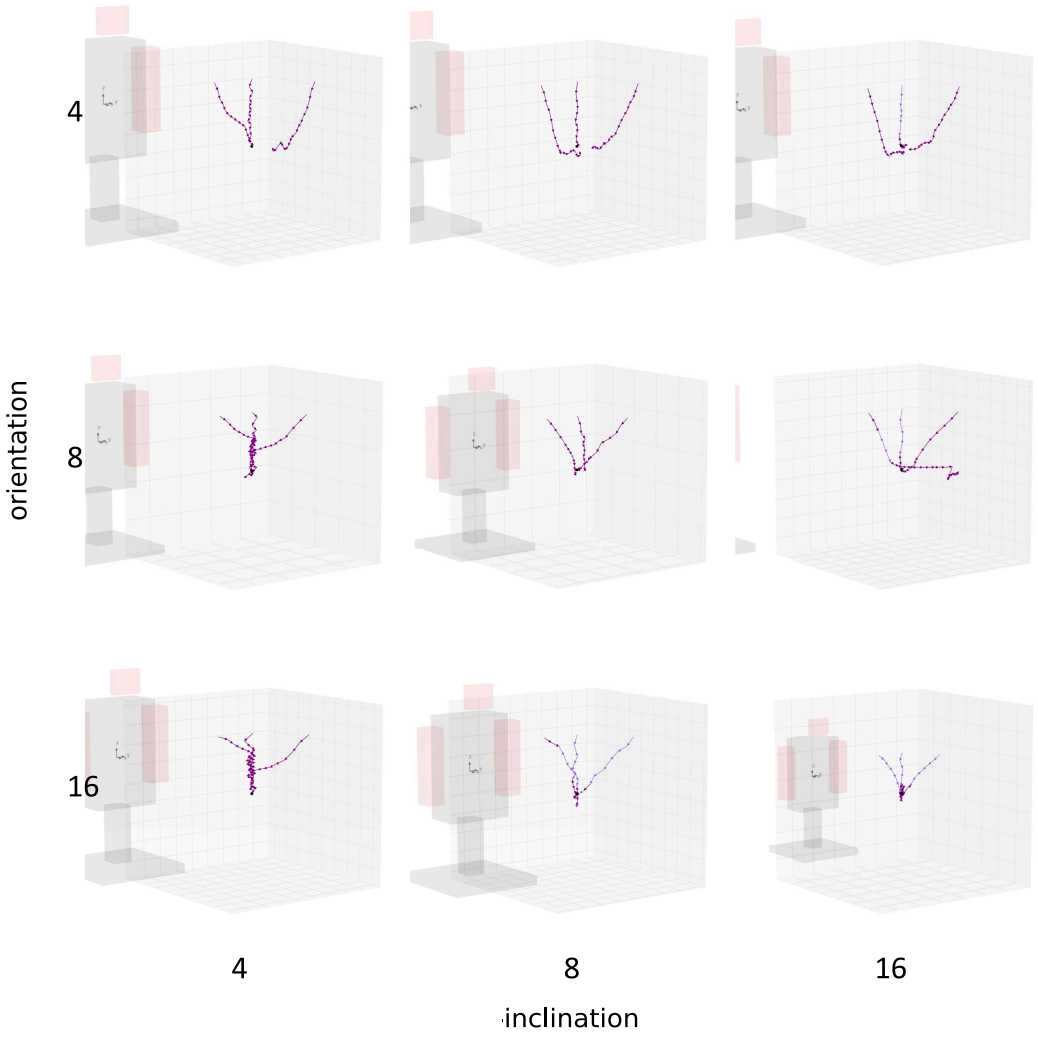


Figure A.3: Experiment 1.2 - Linear discretization

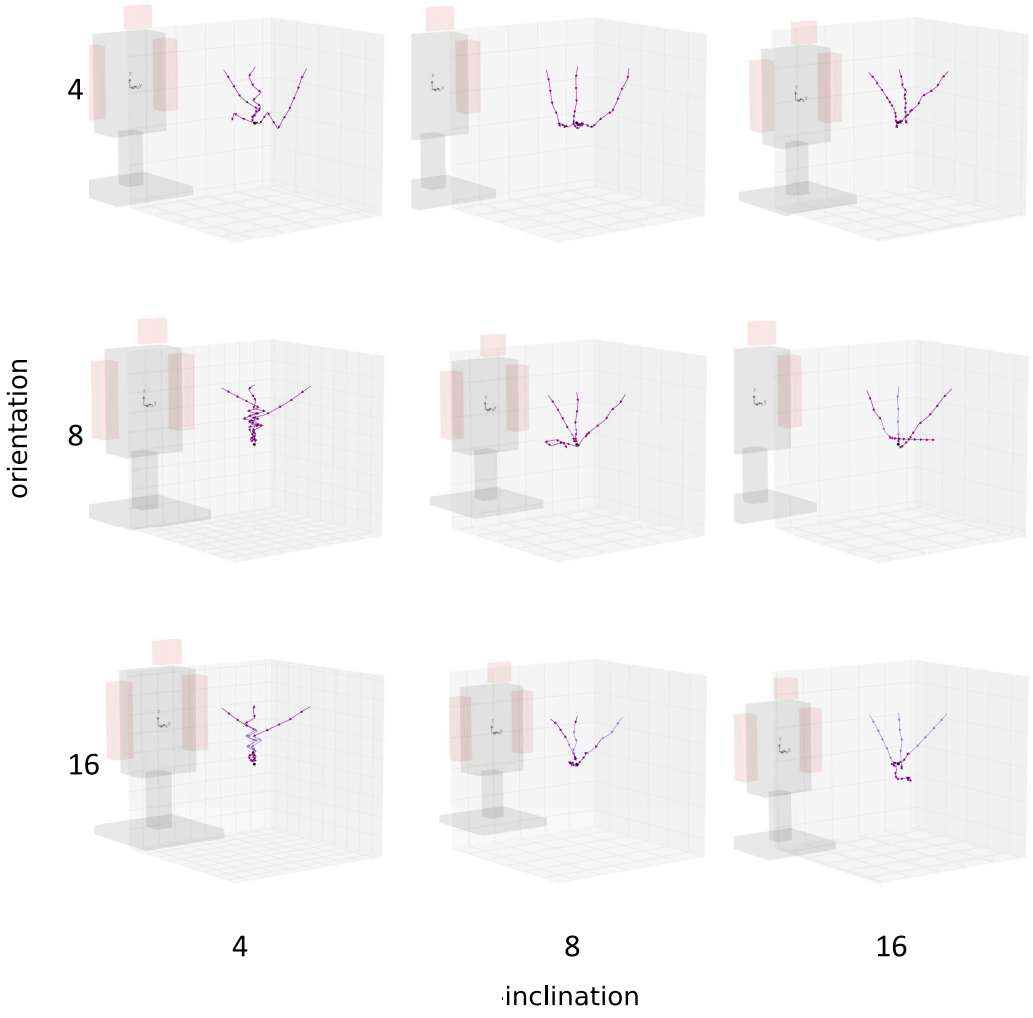


Figure A.4: Experiment 1.2 - Progressive discretization

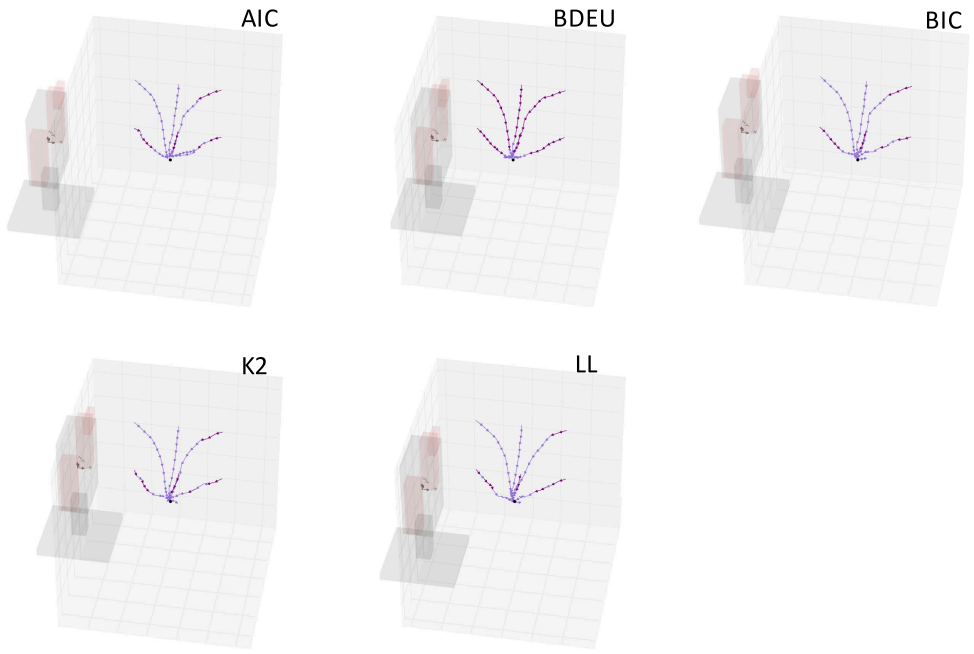


Figure A.5: Experiments 2.1 and 3.1 - Hard-coded

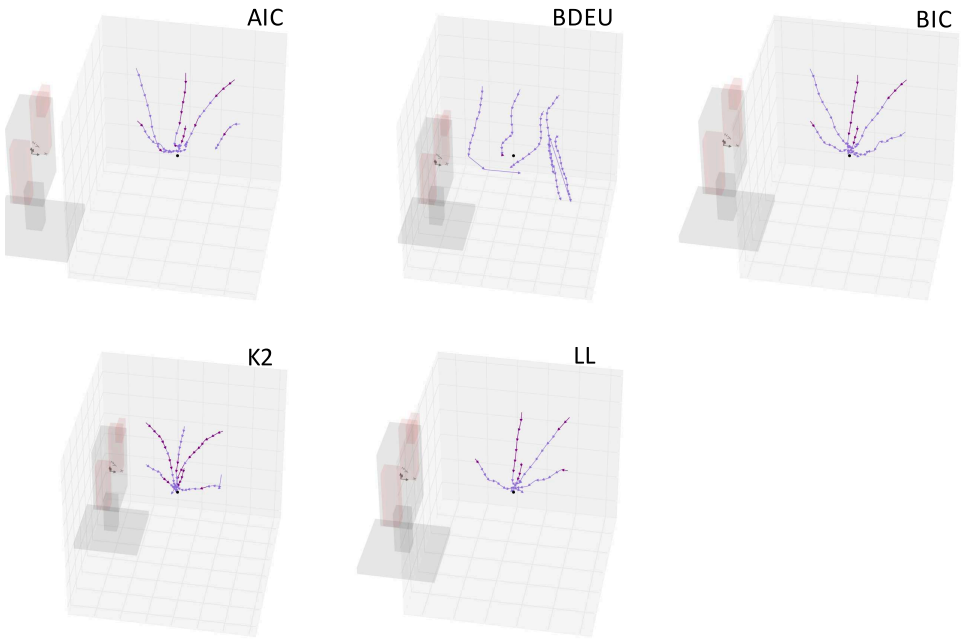


Figure A.6: Experiments 2.1 and 3.1 - Hill climbing

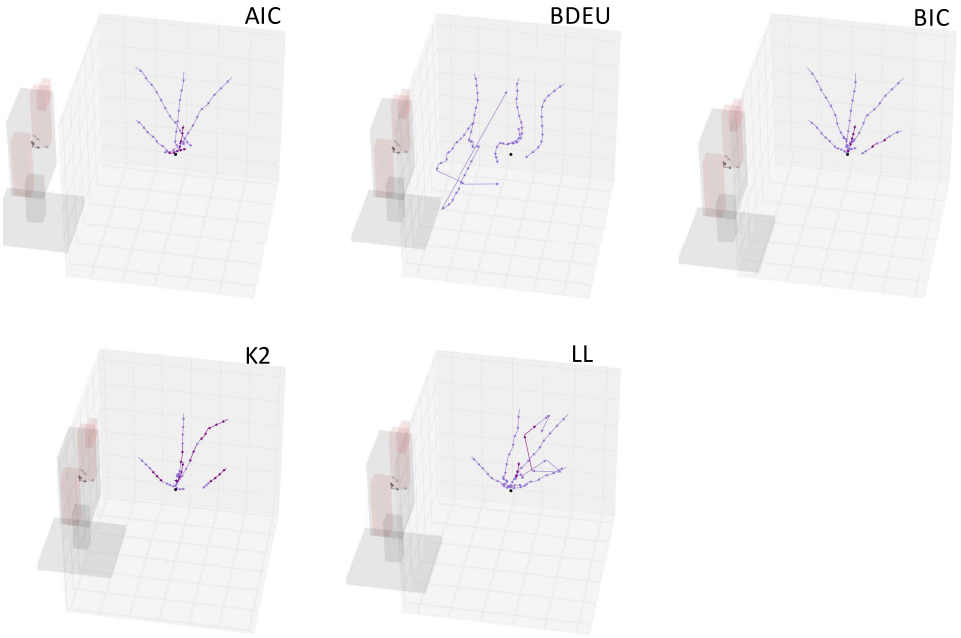


Figure A.7: Experiments 2.1 and 3.1 - K2 descendent

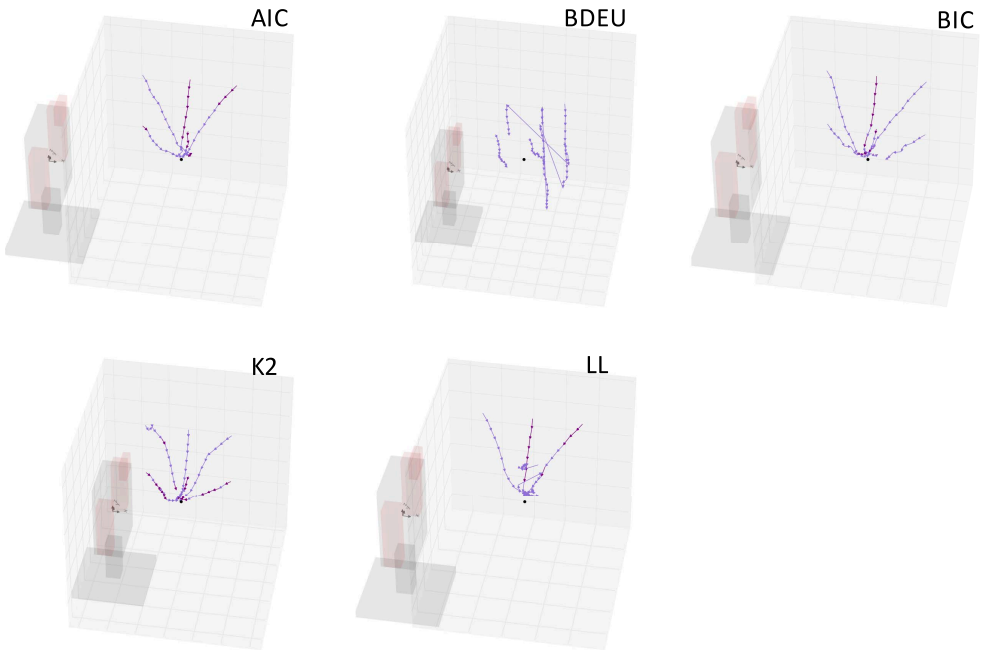


Figure A.8: Experiments 2.1 and 3.1 - K2 ascendent

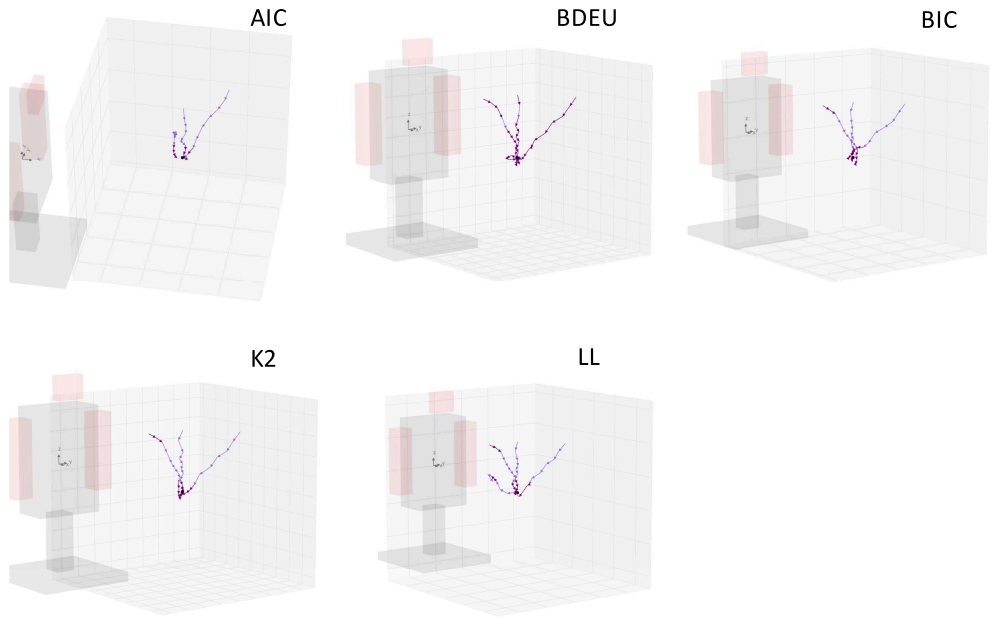


Figure A.9: Experiment 2.2 - Hard-coded

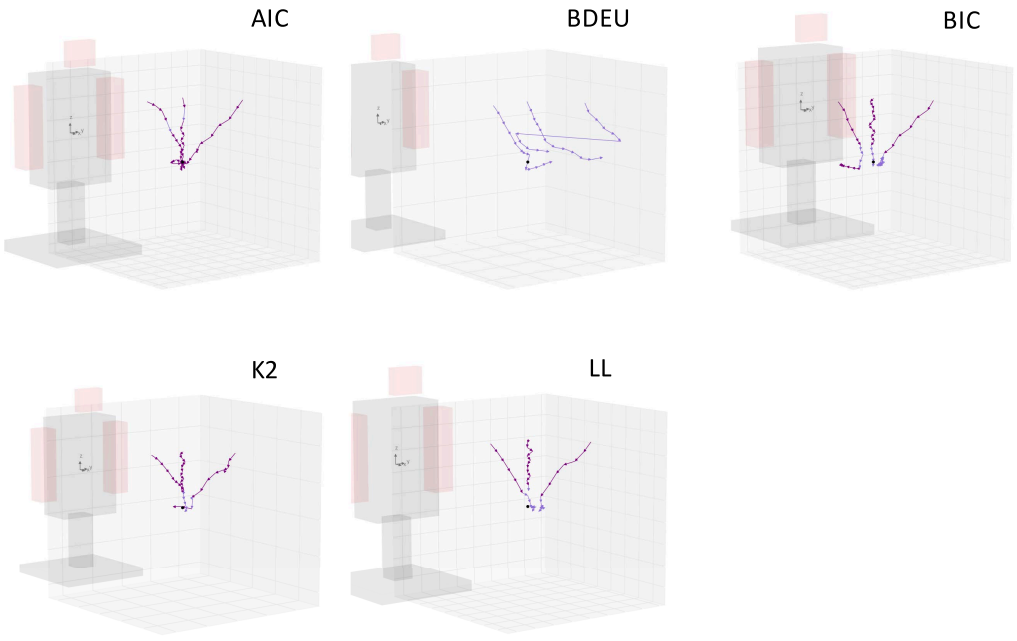


Figure A.10: Experiment 2.2 - Hill climbing

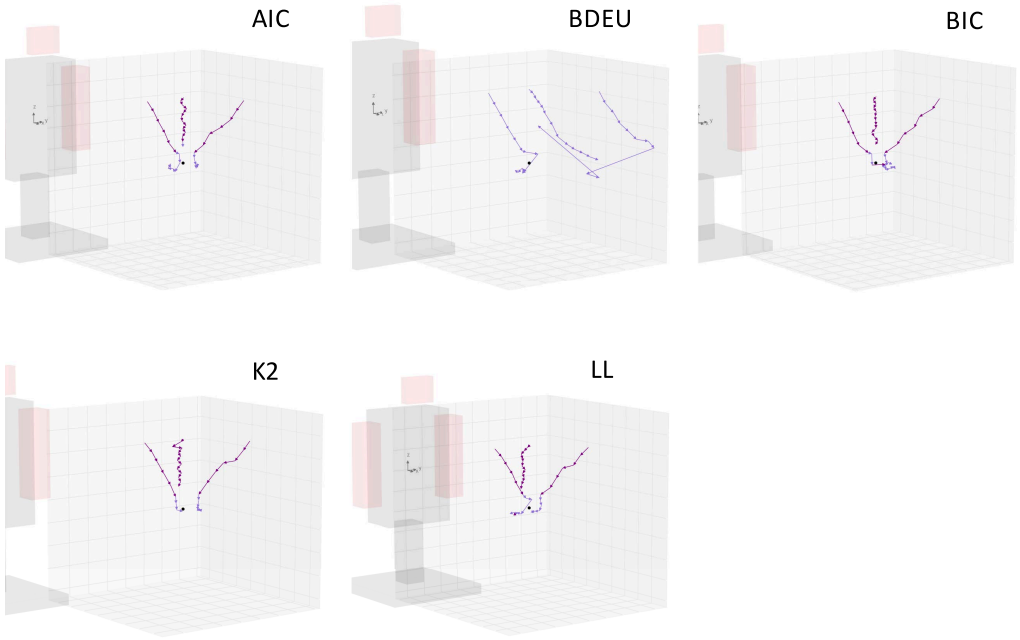


Figure A.11: Experiment 2.2 - K2 descendent

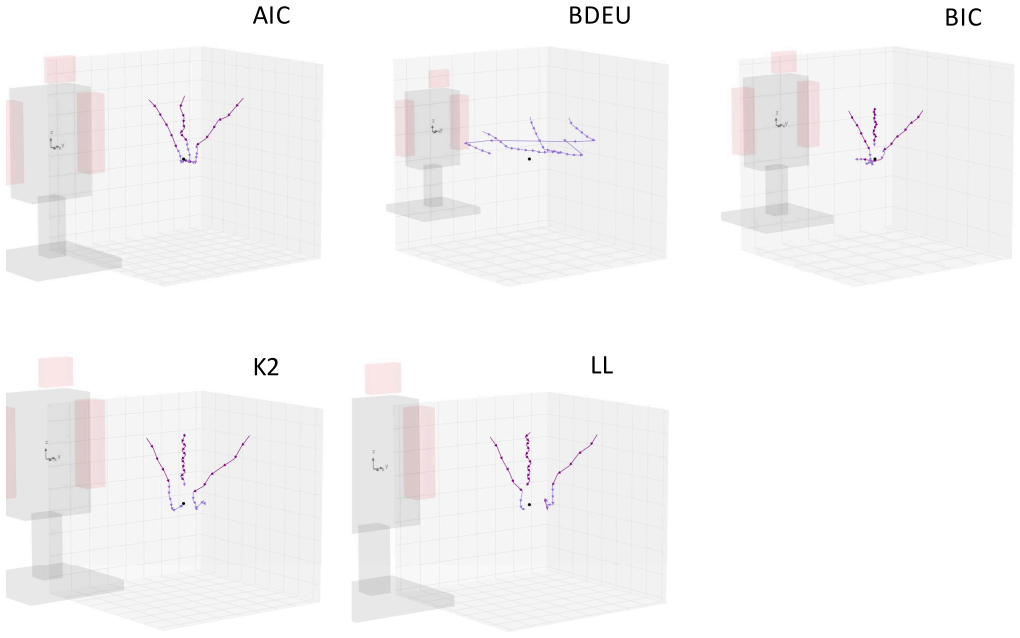


Figure A.12: Experiment 2.2- K2 ascendent

Appendix B

Annex B - Task Planning

This section shows the task definition and planned action sequence obtained using a PDDL-inspired task planner.

Domain and Problem

```
domain = Domain((
    Action(
        'turn-stove-on',
        preconditions=(
            ('at', 'stove-power', 'off'),
            ('at', 'food-coocking', 'none'),
            ('at', 'food-grasped', 'no'),
        ),
        effects=(
            ('at', 'stove-power', 'on'),
            ('at', 'food-location', 'dish'),
        ),
    ),
    Action(
        'grasp-food-dish',
        preconditions=(
            ('at', 'stove-power', 'on'),
            ('at', 'food-grasped', 'no'),
            ('at', 'food-location', 'dish'),
        ),
        effects=(
            ('at', 'food-grasped', 'yes'),
        ),
    ),
    Action(
        'set-food-pan',
        preconditions=(
            ('at', 'stove-power', 'on'),
            ('at', 'food-coocking', 'none'),
            ('at', 'food-location', 'dish'),
            ('at', 'food-grasped', 'yes'),
        ),
        effects=(
```

```
            ('at', 'food-location', 'pan'),
            ('at', 'food-grasped', 'no'),
        ),
    ),
    Action(
        'cook-food',
        preconditions=(
            ('at', 'stove-power', 'on'),
            ('at', 'food-coocking', 'none'),
            ('at', 'food-location', 'pan'),
            ('at', 'food-grasped', 'no'),
        ),
        effects=(
            ('at', 'food-coocking', 'mid'),
        ),
    ),
     Action(
        'grasp-food-pan',
        preconditions=(
            ('at', 'stove-power', 'on'),
            ('at', 'food-coocking', 'mid'),
            ('at', 'food-location', 'pan'),
            ('at', 'food-grasped', 'no'),
        ),
        effects=(
            ('at', 'food-grasped', 'yes'),
        ),
    ),
    Action(
        'set-food-dish',
        preconditions=(
            ('at', 'stove-power', 'on'),
            ('at', 'food-coocking', 'mid'),
            ('at', 'food-location', 'pan'),
            ('at', 'food-grasped', 'yes'),
        ),
        effects=(
            ('at', 'food-location', 'dish'),
            ('at', 'food-grasped', 'no'),
        ),
    ),
    Action(
        'turn-stove-off',
        preconditions=(
            ('at', 'stove-power', 'on'),
            ('at', 'food-coocking', 'mid'),
            ('at', 'food-grasped', 'no'),
        ),
        effects=(
```

```
            ('at', 'stove-power', 'off'),
        ),
    ),
))

problem = Problem(
    domain,
    {
        'food-coocking': ('none', 'rare', 'mid'),
        'food-grasped': ('yes', 'no'),
        'food-location': ('dish', 'pan'),
        'stove-power': ('on', 'off'),
    },
    init=(
        ('at', 'food-location', 'dish'),
        ('at', 'stove-power', 'off'),
        ('at', 'food-coocking', 'none'),
        ('at', 'food-grasped', 'no'),
    ),
    goal=(
        ('at', 'stove-power', 'off'),
        ('at', 'food-coocking', 'mid'),
        ('at', 'food-location', 'dish'),
    )
)
```

Planned Sequence of Actions

1. turn-stove-on()
2. grasp-food-dish()
3. set-food-pan()
4. cook-food()
5. grasp-food-pan()
6. set-food-dish()
7. turn-stove-off()