DREAM: Deferred Restructuring of Experience in Autonomous Machines
H2020-FETPROACT-2014

Deliverable D6.1

Experimental Protocols for the validation experiments

Due date of deliverable: July, 1st, 2015
Actual submission date: June, 26th, 2015
Partner responsible: UPMC
Name of participants:

- Stéphane Doncieux (UPMC, main author)
- Richard Duro (UDC)
- Abraham Prieto (UDC)
- Francisco Bellas (UDC)
- Freek Stulp (ARMINES)
- David Filliat (ARMINES)
- Tim Hospedales (QML)
- Jacqueline Heinerman (VU/VUmc)
- Evert Haasdijk (VU/VUmc)
- Gusz Eiben (VU/VUmc)

Grant Agreement Number: 640891
Contract Start Date: January, 1st, 2015
Duration: 48 months
Project coordinator: UPMC
Partners: UDC, QML, ARMINES, VU/VUmc

Document validated by the Management Committee on June, 22nd, 2015.

CONTENTS

I Introduction 9

1 NOMENCLATURE 13
- 1.1 Problem Definition 13
  - 1.1.1 World States and Observations 13
  - 1.1.2 Effector Commands 14
  - 1.1.3 Utility Functions 14
  - 1.1.4 Task, domain and context 15
  - 1.1.5 Time spent on the search for a solution 16
- 1.2 Internal Representations used by the robot 16
  - 1.2.1 Representation space and $\mathcal{KS}$ redescription function 17
  - 1.2.2 Policy 18
  - 1.2.3 Predictive models 19
- 1.3 Managing Internal Representations: Memory 19
  - 1.3.1 Representation memory 20
  - 1.3.2 Episodic memory 20
  - 1.3.3 Concept 20
  - 1.3.4 Relation memories 21
  - 1.3.5 Short term vs long term memory 21
  - 1.3.6 Cognitive strategy 22
- 1.4 Transfer and Adaptation 22
  - 1.4.1 Task Transfer and Domain Shift 23
  - 1.4.2 Visible versus Hidden Tasks and Domains 24
  - 1.4.3 Paramaterized versus Discrete Tasks and Domains 25
  - 1.4.4 Homogeneous and Heterogeneous Transfer 26

2 METHODOLOGY 27
- 2.1 Objective 27
- 2.2 Performance criteria 27
  - 2.2.1 Relative performance indicator 28
  - 2.2.2 TS^3 indicator 29
  - 2.2.3 Requirements for the experiments 30
- 2.3 Innate vs acquired 31
  - 2.3.1 Requirements for the experiments 31
- 2.4 Characterizing development 32
  - 2.4.1 Improvement over time 32

2.4.2 Generalization ability 32
2.4.3 Success of the developmental process 33
2.4.4 Requirements for the experiments 34
2.5 Project structure 35
2.5.1 WP1: Value Decomposition and restructuring 35
2.5.2 WP2: Representation restructuring (STM) 36
2.5.3 WP3: Knowledge consolidation and restructuring (LTM) 36
2.5.4 WP4: Collective sharing of knowledge 37

II Scenarios for scientific validations 39

3 SIMULATED FORAGING SCENARIO 43
3.1 Main principle and motivation 43
3.2 Detailed description 43
3.2.1 Experimental Setup 43
3.2.2 Tasks and performance criteria 44
3.2.3 Domains 44
3.2.4 Expert policy 47
3.2.5 Naive policy 50
3.3 Data to store 50
3.4 Goals of the experiments 51
3.4.1 With respect to WP1 51
3.4.2 With respect to WP2 51
3.4.3 With respect to WP3 52
3.4.4 With respect to WP4 52

4 FORAGING SCENARIO: HARDWARE AND COLLECTIVE 53
4.1 Main principle 53
4.2 Detailed description 53
4.2.1 Experimental Setup 53
4.2.2 Task and performance criteria 56
4.2.3 Domains 57
4.2.4 Data to store 58
4.3 Goals of the experiments 58
4.3.1 With respect to WP4 59

5 ROBOT WITH ARMS SCENARIO 61
5.1 Main principle and motivation 61
5.2 Detailed description 61
5.2.1 Experimental Setup 61
5.2.2 Tasks and utility functions 63
5.2.3 Domains 64
5.2.4 Expert policy 65
5.2.5 Naive policy 65

5.3 Data to store 66
5.4 Goals of the experiments 66
5.4.1 With respect to WP1 66
5.4.2 With respect to WP2 66
5.4.3 With respect to WP3 67
5.4.4 With respect to WP4 67

III Scenarios for a general public audience 69

6 ADOPT A ROBOT INITIATIVE 73
6.1 Main principle 73
6.2 Detailed description 74
6.2.1 Experimental Setup 74
6.2.2 Objectives 76
6.2.3 Tasks and performance criteria 76
6.2.4 Domains 77
6.2.5 Expert policy 78
6.2.6 Naive policy 79
6.3 Data to store 79
6.4 Contexts 79
6.5 Goals of the experiments 79
6.5.1 With respect to WP1 80
6.5.2 With respect to WP2 80
6.5.3 With respect to WP3 81
6.5.4 With respect to WP4 81

IV Conclusion 83

PREAMBLE

The DREAM project is dedicated to the development of methods and algorithms to bootstrap a developmental process. It is focused in particular on representational redescription, the process that allows to redescribe data coming from an individual's experience in a format that makes further treatments easier, including the ability to predict what will happen or to reach a desired effect, either directly or through a fast learning algorithm. DREAM aims at starting from the lowest level possible, with as few task-specific knowledge as possible, and progressively build this knowledge through robot interactions with the environment.

This document describes the methodological and experimental frameworks to be used during the DREAM project. The introduction part includes a nomenclature listing the important notions with respect to DREAM ideas with the terms we propose to use and what they mean for us. A methodological chapter follows to define the criteria we will use to quantify the progresses made during the developmental process. The next part describes the scenarios we will rely on for the validation experiment. The scenarios are described, including the elements required to apply the proposed methodology. The scenarios do not include research questions, but describe experimental setups in which relevant research questions may be addressed. At the end of each scenario description, the potential of each scenario to address the research questions of each work package is presented. Hypotheses and abilities to test will be identified during the project and dedicated experiments will then be proposed. They will, a priori, rely on the proposed scenarios, but new scenarios can be proposed, if it is justified. The document describes then a set of experiments that has a different goal: showing DREAM related ideas to a general public audience. These experiments do not aim at validating hypotheses or testing abilities and they have thus a different status.

Part I.

Introduction

The first two chapters define the nomenclature to be used during the project, in particular when describing scenarios and experiments and the methodology to be followed. Other chapters rely on the notions defined here.

1 NOMENCLATURE

This chapter aims at defining a nomenclature of terms to be used in the DREAM project. These terms will in particular be used to describe the scenarios.

1.1 PROBLEM DEFINITION

In this section, we formalize and give examples of

WORLD STATE: The configuration of the world; an abstract concept not accessible or used by the agent (that will build its own representation of it);

OBSERVATIONS: The observations the robot makes (e.g. camera images), which depend on the world state;

EFFECTOR COMMANDS: The commands sent to the robot's motors/effectors;

UTILITY FUNCTIONS The utility function that the robot needs to maximize. These functions may express task-specific goals or, for instance, encourage exploration. Utility functions cannot be modified by the robot, but it can be ignored at some moment during the development;

TASK, DOMAIN AND CONTEXT: Task, domain and context define an experiment;

TIME SPENT ON THE SEARCH FOR A SOLUTION (TS3): The time spent on the search for a solution corresponds to the time the robot spends trying to solve a task in a particular domain.

1.1.1 World States and Observations

The world has a state. Chairs will be at certain positions (even if there is nobody to observe them), and the moon is at a certain (variable) distance from the earth (even before any life on earth existed to observe the moon). What the state of our real world is easily leads to lengthy philosophical

discussions about Heisenberg. The idea is more obvious when using simulation, as the state of the world is very clearly defined (positions of objects, size of room, etc.), and can usually be saved to/loaded from a file.

The world state is the actual configuration of objects and robots in the world. To the robot, the world state is an abstract concept to which it does *not have access*, and which it can thus not use internally. Instead, it observes the world with a (non-linear) observation function (as in the Extended Kalman Filter).

$$H : \mathcal{S} \mapsto \mathcal{O}, \tag{1.1}$$

where $\mathcal{S}$ is the world state space and $\mathcal{O}$ is the space of observations. They are the raw sensor values coming from the robot sensors, e.g. pixel values in an image or laser range finder distances.

1.1.2 Effector Commands

An effector command is a set of vector of values $\mathbf{u} \subseteq \mathcal{U}$ to be sent to robot effectors, where $\mathcal{U}$ defines the space of possible commands. In case of n motors, it will be $\Pi_{i=0}^{N}[min_i, max_i]$, where $min_i \subseteq \mathbb{R}$ and $max_i \subseteq \mathbb{R}$ are respectively the minimum and maximum values for motor i. This is the output sent to the robot's actuators.

1.1.3 Utility Functions

The aim of a robot is to optimize utility functions. These are also known as reward functions in reinforcement learning[1], fitness functions in evolutionary robotics or cost functions in optimal control. Utility functions may express very task-specific goals (getting a high utility when the ball goes in the cup) or developmental goals (i.e. fostering curiosity by rewarding learning progress). The utility function *cannot be modified by the robot*; the robot may have internal representations of the utility function which it learns/modifies, we will call them *value functions* (Section 1.2.3), not utility functions:

- *Utility function*: user given and fixed reward function;
- *Value function*: self created reward function.

1 This problem formalization corresponds in essence to the formalization used in Markov Decision Processes (MDP) and Partially Observable Markov Decision Process (POMDP) with continuous state/action spaces. We avoid using the exact same terminology because problem formalizations (e.g. MDP or POMDP) are often associated with a certain class of algorithms to solve that problem (e.g. MDP/POMDP solvers). Even if the problem formalization fits, we want to distance ourselves from the typical solvers associated with this problem formalization.

We can distinguish between several types of utility functions, depending on the input they take:

- *External Utility Functions:* Utility functions need not be computed from the same observation space $\mathcal{O}$ that the robot has access to. For instance, a Thymio II robot may observe only its laser range finder values, whereas a ceiling camera (to which the Thymeo does not have access) is used to determine the positions of the robots to determine how far they are from a certain reward position.

- *Internal Utility Functions:* Although robots cannot modify utility functions, these utility function may depend on their internal states, like their battery level, for instance.

Utility functions either provide one single scalar (as in reinforcement learning) or composed of several utilities in a vector (as in multi-objective optimization). Utility functions will be noted $f()$ and utility values f.

1.1.4 Task, domain and context

A *task* T is something the robot needs to be able to do (whether intrinsically or extrinsically motivated). For example, an articulated robot may need to catch or throw a ball, a mobile robot may need to pick up and deliver an object. The task is defined through the corresponding utility functions, that evaluate the goodness of a particular policy at fulfilling a task (or set of tasks). E..g, for throwing, the utility may be squared distance of object from target; for object collection it may be number of objects delivered to goal. If you change the utility function, you are changing the task you are training the robot to do.

A *domain* D is a set of world conditions that may affect the efficacy of a policy at achieving a task, without changing the utility function itself. A good policy should be robust to changes in domain. For example: the strength of gravity, internal joint resistance and room lighting contribute to the operating domain of a vision-driven articulated robot. The ground-friction and room layout contribute to the operating domain of a mobile wheeled robot. Another way to understand domain using RL formalism is as defining the details of the world model $M : \mathcal{S}_t \times \mathcal{U}_t \rightarrow \mathcal{S}_{t+1}$ used to simulate the environment[2], or observation function $H : \mathcal{S} \rightarrow \mathcal{O}$ through which the robot experiences the world.

The *context* C of an experiment is the pair (D, T) describing the domain and task corresponding to the experiment.

2 In simulation we control the world model already, in reality we imagine this exists although neither the robot nor the experimenter has direct access to it.

1.1.5 Time spent on the search for a solution

The *time spent on the search for a solution* τ represents the time spent in a given context to try solving a task (including the eventual learning time).

Days and nights: Developmental time will be cut in a succession of active learning periods (*days*) and passive[3] introspection, consolidation and restructuring (*nights*). A history of a particular development can thus be based on events or features of the system occurring at particular days or nights that can be identified by an integer: d_1 is the first day after the developmental system started, d_2, the second day, etc. Days will have a maximum length l_d.

τ measures a time spent on trying to solve a task during the day. It implies that τ is bounded by l_d:

$$\tau < l_d$$

l_d is a parameter of the system.

For τ, the counter starts when the robot first perceives the situation. It is representative of the time used to dig into the memory for similar situations, to learn thanks to adapted representations or to explore when relevant representations are not available. This value is expected to change over several orders of magnitude from one task or domain to another (and from one moment during the development to another) depending on whether the robot uses zero-shot learning, rapid learning or slow learning. τ does not represent the total development time: it counts the time required on a particular learning experiment, i.e. when the cognitive architecture triggers a learning mechanism with an appropriate motivation, and stops when the goal is reached or when the cognitive architecture gives up trying to satisfy this motivation.

1.2 INTERNAL REPRESENTATIONS USED BY THE ROBOT

The previous section defined a problem, which has previously been addressed by for instance POMDP solvers or optimal control. Such approaches do not suffice to deal with complex problem instances within this general definition (learning to tidy a kitchen), but the problem formalisation is nevertheless useful. One of the key ideas in DREAM is that observation and effector commands should be redescribed in more abstract perception and action representations, to enhance generalization, adaptivity and robust-

3 At least from the perspective of the real robot, this period may include active learning, but in simulation only.

ness. In this section, we formalize the representations and models required for this approach. The notions introduced are the following:

REPRESENTATION SPACE AND KARMILOFF-SMITH FUNCTIONS: The representation spaces available to the cognitive architecture and the mapping functions to go from one space to another. This leads to the definition of *actions* and *percepts*;

POLICY: A policy is a specific mapping function that generates actions out of percepts;

PREDICTIVE MODELS: Predictive models predict the results of an action in a particular state (forward model) or predicts a reward out of a percept (value function).

1.2.1 Representation space and $\mathcal{KS}$ redescription function

Observations, or properties of the cognitive system or of effector commands can be captured by a representation function that transforms an input space into another space, to make further treatments easier or to connect to robot effectors. The robot itself is included in the definition of "world", so robot commands and robot internal states are susceptible of being sensed too.

We call such properties *attributes*. Attributes for observations can also be called *features*. Each attribute has an associated range of possible values. A representation space $\mathcal{R}_i$ is the set of all possible assignments of values to a set of attributes. A state denotes a (possibly partial) assignment of values to the attributes in the space. This definition is inspired from [5].

Formally, a representation space $\mathcal{R}_i$ comes with at least one $\mathcal{KS}$ redescription function (*Karmiloff-Smith* mapping function) that either maps observations or another representation space $\mathcal{R}_j$ to $\mathcal{R}_i$ or maps $\mathcal{R}_i$ to effector commands or another representation space $\mathcal{R}_k$:

- $\mathcal{KS} : \mathcal{O} \rightarrow \mathcal{R}_i$
- $\mathcal{KS} : \mathcal{R}_j \rightarrow \mathcal{R}_i$
- $\mathcal{KS} : \mathcal{R}_i \rightarrow \mathcal{R}_k$
- $\mathcal{KS} : \mathcal{R}_i \rightarrow \mathcal{U}$

where $\mathcal{R}_j$ and $\mathcal{R}_k$ are other representation spaces.

Actions are representations linked to effector commands. We will denote them $\mathcal{R}_u$. They are representations that directly affect effector actions without any other decision. These representations could typically be discovered

by unsupervised learning algorithms trained on data sets of effector commands. Formally, there exists a set of redescription functions that do not involve decision, such that $\mathcal{R}_u$ maps to $\mathcal{U}_r$:

$$\mathcal{R}_u \rightarrow \cdots \rightarrow \mathcal{U}_r$$

Percepts are representations linked to sensor values. We will denote them $\mathcal{R}_o$. They are representations that are deduced from robot sensor values without any intermediate decision. These representations could typically be discovered by unsupervised learning algorithms trained on data sets of observations. Formally, there exists a set of redescription functions that do not involve decision, such that $\mathcal{O}_r$ maps to $\mathcal{R}_o$:

$$\mathcal{O}_r \rightarrow \cdots \rightarrow \mathcal{R}_o$$

$\mathcal{KS}$ functions may resdescribe observations (from raw image to observed objects coordinates), effector commands (from an action description like 'move forward' to the effector commands to be sent to motors) or may correspond to a decision process (from 'obstacle in front' to 'turn left').

While a $\mathcal{O}_r \rightarrow \mathcal{R}_o$ or $\mathcal{R}_u \rightarrow \mathcal{U}_r$ mappings can be learned with unsupervised learning, a mapping from any $\mathcal{R}_o$ to any $\mathcal{R}_u$ is expected to be non-unsupervised, i.e. either supervised or relying on a reward signal.

WARNING: the notion of a decision process is actually hard to define formally. A $\mathcal{KS}$ function that maps an image to some object feature like coordinates may actually contain some decision (what pixels to consider, what pixels to ignore). This definition is then, somewhat, arbitrary and would need to be refined. It is anyway useful for practical reasons, in particular to define policies.

1.2.2 Policy

A policy is a $\mathcal{KS}$ redescription function (or an algorithm) that proposes actions as a function of perceptions (sensory values transformed through representation functions). It can be defined as follows:

$$\Pi : \mathcal{R}_o \rightarrow \mathcal{R}_u$$

A Π is usually learnt when trying to maximize a reward. The links between policies, rewards, context of use as well as any other relevant cognitive architecture component, will be managed in the relation memory part (section 1.3.4).

We will call π a policy mapping directly observations to effector commands:

$$\pi : \mathcal{O} \rightarrow \mathcal{A}$$

Procedural Strategy is a synonym of policy. A *behaviour* is what can be observed when a robot follows a particular policy.

1.2.3 Predictive models

We consider two different kinds of predictive models that depend on the input they take into account and on the output they generate: *forward models* and *value functions*.

Given a percept and an action, a forward model predicts a new state. The predicted state may be in a different representation space. It may, for instance, be a percept (what will the robot perceive after its action[4]) or an action (what will the robot do after that action[5]), or a reward (predict the instantaneous reward of a particular action in a particular state[6]). A forward model FM can then be defined as:

$$\mathrm{FM} : \mathcal{R}_o \times \mathcal{R}_u \rightarrow \mathcal{R}$$

A value function is a particular type of forward model that makes predictions about future utilities.

$$\mathrm{VF} : \mathcal{R}_o \rightarrow \mathbb{R} \tag{1.2}$$

Value functions may be considered as internal utility functions, which aim at modeling the true utility functions or at creating intermediate problems to solve. Value functions allow the system to select actions that lead to higher values, and thus higher utilities/rewards. Much of traditional model-free reinforcement learning is based on the assumption that value functions are the only predictive models that are required to solve a problem.

1.3 MANAGING INTERNAL REPRESENTATIONS: MEMORY

This section introduces the following notions:

REPRESENTATION MEMORY: The memory containing redescription functions;

CONCEPT abstraction of relationships;

4 This is like an internal estimate of the underlying world model, but mediated by the robot's imperfect perception of the world. May used conventionally for planning; or in this project for dreaming.

5 It would be a model of a policy.

6 Distinct from value function that is expected discounted future reward.

RELATION MEMORY: memory of the relations between the elements of the cognitive architecture;

SHORT TERM VS LONG TERM MEMORY: non consolidated vs consolidated memory;

COGNITIVE STRATEGY: Process designed to reach a particular goal while relying on other elements.

1.3.1 Representation memory

This memory contains all redescription functions.

1.3.2 Episodic memory

An *episodic memory* is a, usually size-limited, memory of relevant past events, i.e. of past perceptions and applied actions or re-representations of them. It is managed by a set of algorithms or rules (Episodic Memory Management Rules or EMMRs) that determine what, out of the data stream coming from its associated representation functions, can go into this memory and what is deleted from it when required (e.g. do not input a tuple that is already in the episodic memory or only input tuples that are hard to model by a given PM). A typical episodic memory will contain sets of tuples $(r_o^t, r_u, r_o^{t+1}, m) \in \mathcal{R}_o \times \mathcal{R}_u \times \mathcal{R}_o \times \mathcal{M}$.

An architecture may contain multiple episodic memories each providing a different view of the raw stream of data determined by its EMMR set. That is, what is stored in each one of these episodic memories will be determined by a set of representation functions linked to it that will provide a given representational format to each element in the tuples and its own EMMRs.

The most basic episodic memory is the one that preserves all the raw values of coming from all the sensors, including those related to motivations, as well as all the actions performed without any filtering or representation restructuring.

1.3.3 Concept

A *concept* is an abstraction of relationships between instances of architectural elements. It is aimed at simplifying and structuring the processing carried out by the cognitive architecture. It may involve actions, sensor values or any other part of the architecture and aims at abstracting a notion to make any related processing simpler and faster. It implies to go beyond the few observed examples and to find some generalization.

Concepts are related to the symbols of traditional AI. A symbol can be seen as a label for a concept. The goal here is to let concepts emerge during the developmental process. A concept may then appear without any associated symbol[7].

Discovering concepts autonomously is a challenging problem that will not be solved within the DREAM project. Demonstrating the discovery of at least one concept that is relevant for the robot will be considered as a significant achievement of the project.

1.3.4 Relation memories

The *relation memory* is a dynamic memory that preserves the links between different elements or instances of components present in a cognitive architecture whenever these are required. It is especially focused on long term memory due to the lower variability of the component instances at this level. These links can be activation links or inhibition links or they can be any other kind of relationship between elements of the architecture. This memory is related to a series of algorithms that permit establishing, eliminating and updating the links. Their main purpose is to provide a way for creating, updating and learning relationships between components based on experience.

If one thinks of the different knowledge nuggets a cognitive architecture can acquire in any form or representation (models, perceptions, re-represented perceptions, policies, motivations or goals, ...) as nodes in a graph, these links, which are acquired through experience, represent the edges that join these nodes and that establish interdependences, relationships or any other mutual influence that can lead to more complex forms of knowledge that could be further exploited. Typical mechanisms for knowledge scaffolding can rely on these relations.

As an example, they may be used to represent Object Action complexes [5] or any other Hybrid "knowledge nugget" in which several elements are closely interdependent. It can also be used to represent (and also discover) hierarchies of actions, models or representations.

This notion is relatively new and is expected to be important for the notion of representational redescription and the generalization ability we expect from it. Further studies of this notion with a more formal definition will then be proposed during the project.

1.3.5 Short term vs long term memory

All memories include a short term and a long term part.

7 It would get one if robots engage in a langage game or if humans "tell" the label associated to this concept (provided that the robot can demonstrate what the concept means for it).

The *short term memory* part (STM) contains the elements that are active at a particular moment or that have been memorized recently, typically during the last learning episodes.

The *long term memory* part (LTM) is a consolidated memory. It contains basically the same kind of elements as the STM, but the transfer from STM to LTM implies a selection process (what knowledge nuggets to transfer?) and may also imply a redescription process to make the corresponding knowledge nuggets more abstract and thus more robust to a change of context.

The LTM is the memory in which the cognitive architecture will look at when facing a new situation to see if it needs to start from scratch or if it can build upon previously acquired knowledge.

1.3.6 Cognitive strategy

A *cognitive strategy* should provide the structure that glues and links all the other elements of the architecture into a working cognitive mechanism. It basically contains code that implements a given cognitive mechanism making use of the necessary resources from the other parts of the cognitive architecture, and appropriately coordinates them. The objective of the cognitive strategy is to allow the system that implements it to fulfill its motivations. In other words, it is an algorithm designed with a particular goal or set of goals expressed in terms of motivation satisfaction or knowledge nugget acquisition. A cognitive strategy is to be used when a motivation cannot be satisfied with current policies. Skill optimization, motor babbling, knowledge extraction from learning traces (i.e. from an episodic memory) are examples of cognitive strategies goals.

Cognitive strategies can be connected to other cognitive strategies. A cognitive strategy selector would determine, for instance, which cognitive strategy to activate at a particular time. For debugging and test purposes, it can be no more than a graphical interface asking which cognitive strategy to activate, but in later versions, it can decide on its own depending on the current state of the architecture.

1.4 TRANSFER AND ADAPTATION

In DREAM we hypothesize that with long-term developmental experience, robots will learn new and better representations and policies. The key way to quantify the goodness of the developmental process and resulting representations/policies is speed and efficacy of generalization/adaptation to new situations. Here we formalize this idea in terms of the modeling

challenge and the evaluation of generalization. It leads to the following notions:

TASK TRANSFER AND DOMAIN SHIFT: Different ways of changing the context.

VISIBLE VS HIDDEN TASKS AND DOMAINS: Does the robot know that the task or domain has changed?

PARAMETERIZED VS DISCRETE TASKS AND DOMAINS Different kinds of context modifications, either a change of parameter or a completely new version

HOMOGENEOUS AND HETEROGENEOUS TRANSFER: Are $\mathcal{O}$ and $\mathcal{U}$ identical before and after the modification of context.

1.4.1 Task Transfer and Domain Shift

The stated holy grail of life long learning is "to design a machine that can accumulate adaptations on developmental timescales of months and years". What does this mean? This means it should face a series of tasks and domains, and eventually learn knowledge and skills which it can *abstract* over tasks and domains to become ever more capable.

Conventionally, one trains a different and independent policy per task and per domain. This is because typically applying a policy learned in one task/domain directly to another task/domain results in poor performance. In contrast, we aim for long term learning that can generalize over different domains and tasks.

Consider a particular learning context defined by the domain and task (D, T). In a normal learning session, the robot interacts with the environment and learns a policy/representation: $(D, T) \rightarrow (\Pi, \mathcal{R})$ that maximises its utility. For long term learning, we expect the robot will experience a set[8] of tasks and domains and may learn a set of policies/representations: $(D, T)_{i=1...I} \rightarrow (\Pi, \mathcal{R})_{j=1...J}$. The key success criteria of DREAM is that by consolidation of experience and abstraction, *the resulting set of policies/representations should be better than if each task/domain in the set* $(D, T)_{i=1...I}$ *had been trained independently*. By better we mean learn more quickly, or higher asymptotic performance.

Examples: The articulated robot should learn to throw to multiple positions in the room. By consolidation of experience it should learn to throw to any position better than if it had been trained to throw to that position independently (task transfer). The mobile robot should learn to collect and

8 These would typically be experienced in sequence for a particular robot, but could also be experienced in parallel in the case of collective robotics where each robot faces a different context but they share information.

deliver objects in multiple room layouts. By training on multiple room layouts it should eventually be able to do any given room better than if it had independently trained on that room (domain transfer).

Simplifications: In general, we hope that all learned tasks/domains could benefit from each other, but it may be convenient to simplify the setting initially by supposing that there is one task/domain that is well learned, and this should simply help to bootstrap a second task and domain. This is *one-way transfer*, as opposed to *multi-task/multi-domain* transfer. The ICDL'13 paper [4] is an example of the former, and the ICLR'15 paper [8] is an example of the latter.

Example Realisation 1: An example concrete realisation (UDC) of these ideas could be an evolutionary framework with a population of policies in STM and LTM. The population of policies in STM are being evolved to satisfy the current task/domain and good ones are stored in LTM. On changing task/domain, some relevant policies in LTM should be retrieved to populate the current STM.

Example Realisation 2: A second concrete realisation (QMUL) could be to train a structured meta-policy/model that relates all tasks/domains via mixture and/or factor analysis structure. The policy/model for any particular context (T, D) is derived from the meta-model [8].

1.4.2 Visible versus Hidden Tasks and Domains

An important distinction to make that will affect the challenge level of the posed transfer learning problem, is whether or not an indicator of each unique context (D, T) experienced is given to the robot, or if the robot must deduce any changes of context from interactions with the environment. These settings are termed *visible* versus *hidden* context.

Examples: A mobile robot facing multiple room layouts could have a sensor/input indicating which of a set of layouts (domains) it is currently experiencing. An articulated robot throwing objects of varying weight (domain) could be provided with an indicator about the current object type, or be required to deduce this from the outcomes of its throws.

In practice the hidden context setting means that the robot would have to (noisily) estimate the current task/domain from its sensors or interactions, e.g., $C : \mathcal{O} \rightarrow (\hat{D}, \hat{T})$. This is analogous to the noisy process of estimating world state through inverting of the observation function H in Eq. (1.1). Clearly transfer learning with hidden context is more challenging than with visible context. The IROS'14 paper of Stulp [7] is an example of learning with hidden context. Doncieux ICDL'13 [4] and Yang ICLR'15 [8] are examples of learning with visible contexts.

1.4.3 Paramaterized versus Discrete Tasks and Domains

Tasks and domains may be differentiated from each other at one extreme by a simple change in parameter, and at the other extreme they may be totally distinct.

Parameterized Tasks and Domains

In a *parameterized task*, the utility function (which defines the task) has a free parameter. For example, the task T of throwing or reaching to a target may have a corresponding utility function of $U(x, c) = (x - c)^2$, where x is the position of the object/end effector, and c is the position of the target. Here c is the task parameter. We denote a task with one specific parameterization $T(c)$ as a *task instance*. We hope that after experiencing a set of particular task instances $T(c_i)_{i=1...I}$, the robot will generalize/restructure and be able to solve *every* task instance $T(c)$.

Parameterized Domains: Analogously, a domain D may also be parameterized $D(c)$. For example, the parameter c may be the weight of the object to throw. After learning to throw objects of a set of specific weights, we hope the robot will be able to restructure its policy/representation and throw an object of any weight.

Note the reason tasks and domains are broken up in the way they are above: if the goal is to get an object to a particular position, the target position is a task parameter that modulates the utility function. In contrast, the weight of the object is a "nuisance" condition that the robot should be robust to, thus it is a domain.

Zero-Shot Learning: In the case of parameterized tasks/domains, we expect that a robot can eventually 'solve' the entire set of parameterized tasks/domains. In particular, this means that once 'solved', the robot can be asked to throw / reach to a novel location (task instance) that it has never tried before and achieve it on the first try with zero practice. (This 'no-practice' condition is known as *zero-shot learning* in computer vision.)

Discrete Tasks and Domains

Discrete tasks are not simple re-parameterizations like task instances in parameterized tasks. For example reaching versus throwing versus catching. The more ambitious goal in DREAM is to achieve transfer between discrete tasks rather than simply parameterized tasks. This may require more profound restructuring of representations/policy, but should be achievable to some extent (E.g., throwing and catching both require grasping skill that could be shared).

Analogously, *discrete domains* also exist. Transferring a skill across very different robot types (e.g., NAO→PR2) or putting a mobile robot in a totally different maze would correspond to a discrete domain shift.

We do not expect discrete task/domain transfer to be achievable in 'zero-shot' condition. But knowledge/skill accumulation could still boost the asymptotic performance or learning rate of the individual tasks/domains.

1.4.4 Homogeneous and Heterogeneous Transfer

A final distinction to make is whether the perceptual space $\mathcal{O}$ and/or effector space $\mathcal{U}$ correspond (E.g., have same number of dimensions) across a task/domain shift. The correspondence case is known as *homogeneous* transfer and the non-correspondence case as *heterogeneous* transfer. The latter heterogeneous case is more challenging.

Examples: An example of discrete domain shift would be transferring a policy across a change of robotic platform. Note that this is a (discrete) domain shift and not a task shift. The task may remain the same (E.g., collect and deliver balls), so no change to reward function. But changing robot Thymio $\rightarrow$ Robobo would clearly influence the perceptual function $H : \mathcal{S} \rightarrow \mathcal{O}$ and the outcome of a effector command $\mathcal{S}_t \times \mathcal{U}_t \rightarrow \mathcal{S}_{t+1}$, thus changing the domain. Since the sensors and actuators are heterogeneous (may have different dimensionality), this would also be an example of a heterogeneous transfer task. One way this could happen is to transfer the high-level abstract policy Π, but adapt the low level KS re-description functions $\mathcal{O} \rightarrow \mathcal{R}_o$ and $\mathcal{R}_u \rightarrow \mathcal{U}$ to suit the new robot.

2 METHODOLOGY

2.1 OBJECTIVE

Once their task and environment are known, robots can be explicitly programmed. If the environment shows some variability, learning algorithms can help the robot explore different possibilities to find out how to solve the task. Current learning algorithms anyway require to define appropriate representations. They are considered as problem-specific and cannot be modified by the algorithms.

The DREAM project aims at going beyond imposed representations through the design of algorithms able to restructure the internal representations needed for any of its learning or decision algorithm. An adapted representation is expected to lead to a fast and efficient problem solving. It will typically be task-specific and thus require a specific and careful preparation with the help of a human expert for each task the robot is supposed to fulfill. DREAM cognitive architecture is expected to replace human intervention by an automatic developmental process that will let the robot discover what it needs to know. This developmental step may include interactions with humans to guide it, but no direct programming. DREAM cognitive architecture is expected to start with task-agnostic learning and exploration algorithms whose behavior will generate the data required to bootstrap higher-level learning and decision algorithms. DREAM's ultimate goal is to allow a robot to face an environment that was unknown to its human designer and to reach the level of efficiency of control algorithms relying on task specific representations. DREAM project will be a success if it results in a significant step in that direction and the validation experiments are designed to highlight it.

2.2 PERFORMANCE CRITERIA

Each experiment should have its own utility functions that will be used as performance criteria. Their evaluation may rely on robot's sensors, on an external device, or on feedback from a human experimenter (this last possibility should be avoided, except if the experiments is explicitly targeted towards human interaction). Performance criteria can be used to compare algorithms or developing robots. Typical experiments on curiosity do not

include such criteria, it is then difficult to draw any conclusion about what an individual can do after its development phase.

Performance criteria will be the only experiment-specific aspect of a DREAM setup. They describe the main motivation of the robot (just as survival and reproduction are the main motivations of living beings). In the following, we will use the term *utility value* to refer to the performance criterion value.

During the development phase, it is expected that value functions are created on the fly and may override utility functions, at least for a while.

Once a utility function is defined, at least three control experiments need to be made:

- random policies: what is the best performance of a set of random policies? Let's call f^*_{Rand} the corresponding utility value.
- naive policy: what is the performance of a learning algorithm with task-agnostic representations (typically an evolutionary run with novelty/diversity objectives). Let's call f^*_{naive} the best utility value obtained with such an approach and τ^*_{naive} the corresponding learning speed. This experiment shows the baseline of what is possible with a task agnostic learning process;
- expert policy: what performance can we reach with a policy that exploits all the knowledge we may have on the task? It may either be hand-made controller or a controller tuned by learning or optimization algorithms exploiting dedicated representations. Let's call f^*_{expert} the best utility value thus obtained and τ^*_{expert} the corresponding learning speed.

These definitions allow to define indicators that are easy to interpret and that will highlight properties of our algorithms:

- a relative performance indicator
- a task resolution time indicator

2.2.1 Relative performance indicator

In the following, we make the assumption that $f^*_{expert} > f^*_{naive}$. If it is not the case, it means that an agnostic search is better than or equal to a well informed approach. It may happen if the knowledge used to build the hand-made controller is not appropriate. The corresponding experiment would be a good example of the potential of embodied intelligence, but it would be useless for demonstrating the impact of development. In such a case, we won't be able to measure the progress related to development with the proposed indicator.

The relative performance of a developing individual of performance f is defined as follows:

$$\eta = \frac{f - f^*_{naive}}{f^*_{expert} - f^*_{naive}} \times 100$$

η can be interpreted as follows:

- $\eta < 0$: the performance is below a naive approach. Development has then completely failed.
- $\eta = 0$: the performance is the same as that of a naive approach.
- $0 < \eta \leqslant 100$: this is the expected result of the developmental process. The closer η is to 100, the more successful the experiment.
- $\eta > 100$: the developmental process generates more efficient results than a hand made controller. This is the holy grail of developmental robotics that would highlight the gains associated to embodied intelligence. Nevertheless, it should not be forgotten that it depends on the performance of the expert controller that may not be the most efficient controller possible. If controllers with such η values were to be found, the relevance of the expert controller would need to be carefully checked and debated.

WARNING: this indicator will not be defined for experiments in which $f^*_{naive} = f^*_{expert}$. They should not be used then to demonstrate any progress in performance. Such experiments can be useful to bootstrap the developmental process, and may also be used to demonstrate a progress in the task resolution time.

2.2.2 TS^3 indicator

The time spent on the search for a solution (TS^3) is an indicator of the relevance of the knowledge available. If all necessary knowledge is available for a task, it should be solved rapidly. τ^*_{expert} is representative of the fastest task resolution, typically a zero-shot learning. If no knowledge at all is available, a long task agnostic search is to be performed. τ^*_{naive} is representative of this time.

The TS^3 indicator ζ is defined as follows:

$$\zeta = \frac{\log_{10}(\frac{\tau}{\tau^*_{expert}})}{\log_{10}(\frac{\tau^*_{naive}}{\tau^*_{expert}})} \times 100 = \frac{\log_{10}(\tau) - \log_{10}(\tau^*_{expert})}{\log_{10}(\tau^*_{naive}) - \log_{10}(\tau^*_{expert})} \times 100$$

ζ can be interpreted as follows:

- $\zeta < 0$: the resolution time for the developing robot is faster than the hand made solution. This is also a holy grail that would demonstrate the potential of embodied intelligence. As for $\eta > 100$, the relevance of the hand made solution in this case should be carefully checked and debated.

- $0 \leqslant \zeta < \epsilon$ (with ϵ a small value): the resolution is almost as fast as that of the expert solution, i.e. the robot is able to solve the task without learning.

- $\zeta \leqslant 50$ (i.e. $\frac{\tau}{\tau^*_{expert}} < \sqrt{\frac{\tau^*_{naive}}{\tau^*_{expert}}}$): the resolution time is in between zero-shot and fast learning. The value of 50 (corresponding to a resolution time in square root of the slow learning speed) is arbitrary chosen to set fast learning in between one-shot resolution and slow learning on a log scale.

- $50 < \zeta \leqslant 100$: the resolution time is in between rapid learning and slow learning.

- $\zeta > 100$: the resolution speed is slower than the slowest agnostic learning process. The development has failed.

What is expected is a ζ that progressively moves from 100, at the beginning of the developmental process, to 0. The closer the learning speed gets to 0, the more successful the developmental process.

2.2.3 Requirements for the experiments

Summary of the performance-related requirements on the design of the validation experiments:

- A performance criterion measuring the level of achievement of the task: it will also be used as a utility function by the robot;

- An expert policy must be available for each task and domains. It can be built independently for each task and domains. There is no need for a single policy to cover the whole set of tasks and domains.

- The performance of the naive policy should be below that of the expert policy, otherwise, the gain expected to come from development will be limited.

2.3 INNATE VS ACQUIRED

The morphology of the robot is given a priori. Any morphology-specific knowledge can be considered as available to the development process. It includes in particular:

- perception-related knowledge. Any representation or algorithm that is specific to a particular modality (e.g. vision) can be provided. SIFT or SURF features, SuperVoxels or whatever other vision algorithm can be provided as long as it does not include any task related knowledge (e.g. features of the environment or of objects);

- motor-based knowledge. Neural networks, DMP (Dynamic Movement Primitives) or any other representation providing control over robot motors can be provided as long as they do not include task specific knowledge (the definition of task specific actions must be avoided, for instance).

The philosophy of DREAM is to discover any environment and task specific knowledge and to provide, if required, any knowledge related to robot features. It includes in particular any potentially relevant representation. The goal of the developmental process is to start from an empty task-related knowledge base and to progressively fill it while exploiting robot experience. Representations may also be discovered, starting from a versatile formalism, but this goal goes beyond DREAM objectives and thus will not be a priority in the project.

It should be noted that "acquired" covers here autonomous acquisition and also learning by imitation. The line between innate and acquired is not with respect to decisional autonomy, but with respect to what was put in the robot program: any knowledge that is explicitly coded (and thus requires some programming effort) is considered as innate and any knowledge that is learned (either autonomously or through demonstration) is considered as acquired.

2.3.1 Requirements for the experiments

No particular requirements follow this aspect, except that relevant representations are provided to the developmental process (representations dealing with vision if the robot has a camera, representations to handle redundancy for a robotic arm, etc).

2.4 CHARACTERIZING DEVELOPMENT

The indicators based on the performance criteria allow to quantify the performance over a single learning instance. The developmental process is expected to change these indicators over time and also over the situations the robot will face. The profile of change of these indicators will need to be observed and characterized. This will highlight the life-long learning abilities of the system and may result in requirements on the validation experiments.

2.4.1 Improvement over time

The improvement over time is an expected feature of the developmental process: solving a task that has already been solved before is expected to require less time than when solving it for the first time. It creates no strong requirements over the experimental protocols, except the ability to put the robot in the same situation several times during its developmental process to check any progress in the performance indicators.

2.4.2 Generalization ability

The generalization ability is the ability to transfer the knowledge acquired in one task or domain to another task or domain. With the indicators defined above, a generalization ability can be attributed to the developing robot if, on a new task or domain that has not been encountered before:

- $\eta > 0$, the closer η is to 100, the more successful the generalization;
- $\zeta < 100$, the closer ζ is to 0, the more successful the generalization.

Testing the generalization ability has a significant impact on the definition of an experiment. It implies that the task, defined by a specific utility function, is associated to domains that can be changed. For a robot supposed to move objects to a basket, for instance, the domain corresponds to different objects, at different positions, and different basket positions. Transferring knowledge from one task to another needs to be possible and to help, at least for some couple of tasks. Identifying what knowledge to transfer (or not) is part of the problems to be solved by the cognitive architecture.

There should be a way for the robot to transfer knowledge from one task or domain to another, otherwise, nothing more than agnostic learning can be expected on each new task and domain. This ability to transfer knowledge from one context (task or domain) A to another context B can be stated as the existence of common "knowledge nuggets" between the two contexts and the fact that knowing it helps solving B. This can be explicitly

tested by looking at the expert policies: if they share some features, then transfer of knowledge will be considered as possible. The more they share, the easier the transfer should be.

The level of difficulty of an experiment in a context c can be estimated with the difference in performance between the expert policy for this particular context and the naive policy, i.e. with the value:

$$\Delta_c = f^*_{expert,c} - f^*_{naive,c}$$

Validation experiments are expected to come with a set of contexts with the following features:

- low difficulty (not mandatory): easy to learn experiments ($\Delta_c < \epsilon$). These contexts can be used to bootstrap the developmental process. They are not mandatory, utility functions or self-built value functions can help bootstrapping the process (like artificial curiosity). It can anyway be useful for some proposed developmental algorithms and does not need to be prohibited. They are anyway not mandatory and should be considered only if they are explicitly required. Relying on such contexts will be considered as a limitation of the approach, but not as a failure, if it allows to solve more complex tasks and domains;

- medium difficulty (mandatory): $f^*_{naive} < f^*_{expert}$. An experiment should include a set of such contexts $c_1, c_2, \ldots, c_n$with an increasing level of difficulty: $\Delta_{c_1} < \Delta_{c_2} < \cdots < \Delta_{c_n}$. Ideally, contexts of any arbitrary difficulty should be possible to define;

- high difficulty (recommended): contexts with $\tau^*_{naive} = l_d$[1]. These contexts are challenging tests for the developmental process. Succeeding to solve such a task will be considered as a very significant achievement of DREAM's goals;

- extreme difficulty (not mandatory): contexts in which no expert solution has been found (with a straight-forward approach). These contexts are extremely challenging tests for the developmental process. Succeeding to such an experiment will be considered as a major achievement of DREAM's goals. Please note that the absence of expert solution will need to be carefully justified.

2.4.3 Success of the developmental process

Development is expected to last over long periods. We can denote $\eta_c(d)$ the relative performance of the developing individual in a context c and

1 l_d is the length of the day, i.e. the maximum amount of time spent on a particular experimental context.

at the end of day d and $\zeta_c(d)$ the corresponding TS[3] indicator. A typical developmental process is expected to be successful iif:

$$\forall c \in \mathcal{C}, \exists n \in \mathbb{N}^* \wedge [d_1, \ldots, d_n] \in \mathbb{N}^n \quad | \quad \eta_c(0) < \eta_c(d_1) < \cdots < \eta_c(d_n)$$

$$\forall c \in \mathcal{C}, \exists n \in \mathbb{N}^* \wedge [d_1, \ldots, d_n] \in \mathbb{N}^n \quad | \quad \zeta_c(0) > \zeta_c(d_1) > \cdots > \zeta_c(d_n)$$

where $\mathcal{C}$ is the set of contexts for the experiment.

The larger n, the more progressive the development will be. The closer $\eta_c(0)$ to 0 and $\eta_c(d_n)$ to 100 (resp. $\zeta_c(0)$ to 100 and $\zeta_c(d_n)$ to 0), the more successful the development will be.

Improvement over time will be demonstrated if, after learning in context c during day d, either $\eta_c(d-1) < \eta_c(d)$ or $\zeta(d-1) > \zeta(d)$ holds.

A generalization ability will be demonstrated if either $\eta_c(d-1) < \eta_c(d)$ or $\zeta(d-1) > \zeta(d)$ holds and no learning happened on context c between $d-1$ and d.

Testing it has no particular constraint on the experiment definition, except that there should be enough different contexts and that they should be tested on a regular basis, once after each day.

2.4.4 Requirements for the experiments

Summary of the development-related requirements on the design of the validation experiments:

- definition of contexts for the experiments (tasks and/or domains). Contexts should share "knowledge nuggets" to allow transfer and for the development to be of any use;
- contexts of increasing difficulty should be defined;
- high or extreme difficulty should be possible to define (at least for one validation experiment);
- some contexts will be used during learning, others only for testing generalization abilities.

To test the generalization ability, it should be possible to check the relative performance and the resolution speed without any knowledge acquisition. It implies, for the cognitive architecture, to save its state, i.e. all knowledge discovered up to now, and to restore it after the test so that it is guaranteed that no knowledge has been acquired during the test, as it would alter future tests on the same context.

2.5 PROJECT STRUCTURE

The validation experiments aim at demonstrating the level of achievement of project goals. It implies in particular that the goals of each work packages are achieved. A small reminder of WP objectives is then provided below. These objectives will be considered also as requirements for experiment definition. Among the scientific and technic WP, WP5 is dedicated to study the potential cross-fertilization between neuroscience and robotics. It results then in no strong requirement on the validation experiments, but it can, of course, be used as a source of inspiration. WP6 is dedicated to the integration of WP1 to 4 contributions. It has no specific requirement, except including WP objectives in a same experiment and thus avoid to have exclusively WP specific experiments.

2.5.1 WP1: Value Decomposition and restructuring

The objective of WP1 is to create a motivational system for the robots within the DREAM architecture. This implies two important goals. On the one hand, to study and decide what type and which utility functions the robot should be endowed with innately so that it can derive the value functions that will allow it to operate coherently depending on its environment. Part of these functions (mainly heterostatic) will provide the drives that will make the robot explore and thus allow it to learn efficiently in complex dynamic environments. This will determine the robot's curiosity or its ability to speedily learn from its environment. Another part will provide the basic or main self-preservation and intention generating (mainly homeostatic) utility functions, such as pain or pleasure. These will allow for the generation of instrumental goals and for interaction abilities.

On the other hand, a very important topic that must be addressed here is that of how to produce and relate intermediate or instrumental value functions so that some order and focus can be established in the robot's interaction with the environment thus facilitating its operation in complex environments.

Both of these aspects will be addressed in the experiments that will address several research topics.

STUDY AND DETERMINATION OF APPROPRIATE INNATE MOTIVATIONS: Every experiment will be carried out considering different sets of innate motivations so that the best general set can be determined and its dependencies on tasks and domains studied.

AUTONOMOUS DISCOVERY INSTRUMENTAL VALUE FUNCTIONS: In addition, a decomposition into instrumental value functions will be sought in experiments that involve multiple and complex tasks as well as those

that are more "life-long". Different approaches and clues to discover appropriate instrumental functions will be tested.

2.5.2 WP2: Representation restructuring (STM)

The goal of WP2 is to determine, out of the recent experience of a single individual, typically during a given day, the representations that are the most adapted to store and manipulate policies or predictive models. It implies to find out the relevant representation spaces and redescription functions and how to take them into account either to determine robot actions and movements (policy), or to determine the result of robot actions (predictive models). These new representations will be part of the Representation Memory, they are typically built separately for each day and not compared to previously acquired knowledge (this consolidation is done in WP3). Reaching these goals will require several processes:

DISCOVERY OF RELEVANT REPRESENTATION SPACES: For example the definition of the visual features space from a set of raw visual observation made during a set of experiments on similar tasks/domains or the search of relevance following (Doncieux, IEEE-ICDL 2013): relevant features are those specific to behaviors with high values.

RESTRUCTURING OF POLICIES: For example using regression on recent learning traces in order to define policy spaces or restructure them.

VALIDATION OF NEW REPRESENTATIONS: For example using policy optimization on related tasks/domains to validate the potential of new representation spaces.

2.5.3 WP3: Knowledge consolidation and restructuring (LTM)

This work package seeks to define a long-term memory aimed at storing consolidated knowledge and making it available for further learning and decision processes. In fact, this long term memory structure is contemplated as the main knowledge repository of the DREAM architecture. As such, this work package is concerned with studying different mechanisms that can be incorporated into this LTM structure so that real consolidation and restructuring can be achieved.

Consolidation is taken as the process of linking and organizing the knowledge that is preserved in this memory. This knowledge is represented in the form of components such as policies, forward models, value functions, episodic events, etc. It will be very important for the information regarding the concurrent use of any of these components to be preserved through some type of relationship parameters so that use contexts can be identified and organized. On the other hand, it will also be very important to provide

some type of order and neighborhood information on the different types of elements so that contextual classes can be extracted and thus generalization made possible.

Regarding representational restructuring, this work package is concerned with creating processes that allow DREAM to produce better representations for the models or policies it stores from the knowledge and information acquired by the robot. These representations will involve two main aspects. Initially, it will consider how to rerepresent policies, forward models, etc so that they correspond to "primitives" in order to make it much simpler to progressively construct more complex representations through aggregation, thus allowing for effective scaffolding. It will also consider how to generate and/or use new transformation functions (representation functions) for the information and data coming from the perceptual system all the way to the actuation system so that creating models and/or policies is made simpler (for instance, going from pixel values to blob position information on an image). As indicated, the objective is to do this during the DREAM phase through internal simulations based on the information present in LTM.

As a consequence of all of this, the main research topics that will be addressed and tested throughout the experiments are related to:

STRATEGIES FOR POPULATING THE LONG TERM MEMORY: It is important to determine how to manage long term memory in terms of what goes in and what is forgotten. The effect of these processes on the capabilities of the final system must be evaluated.

CONSOLIDATION AND RELATIONAL MEMORY MANAGMENT ALGORITHMS: It is also necessary to determine what is the best structure or mechanism to establish and preserve valid versions of the relationships among components of the LTM and how to organize these elements into neighborhood based representations so that generalization can be achieved.

RESTRUCTURING STRATEGIES AND REPRESENTATIONAL REDESCRIPTION: This involves the determination of the appropriate approaches to the modularization of the representations of policies, models, etc. so as to make them more accessible to scaffolding in LTM and studying how to incorporate representational redescriptions within the DREAM processes without making the system unstable.

2.5.4 WP4: Collective sharing of knowledge

WP4's objective to share individually acquired knowledge between robots is met when a robot can generalize over an unforeseen context through the experience of other robots.

This objective raises a number of research questions that will be investigated through experiments:

SPEED OF GENERALIZED LEARNING We expect social learning to improve the speed at which generalized policies are discovered, the quality (in terms of measurable robot behavior) of these policies, or both. The quality of robot behaviour (expressed in an appropriate metric) is compared between instances with and without social learning enabled;

MORE THAN PARALLELIZATION We will investigate under which circumstances the benefits of social learning exceed those of merely parallelizing the trials. We hypothesize that social learning can be particularly useful when sharing generalized policies between physically different robots or when sharing experiences of events that occur infrequently. We will also investigate to what extent the benefits of social learning depend on the environment – for instance, if that is stable or dynamic;

LEVEL OF KNOWLEDGE EXCHANGE Social learning can be implemented at the level of the sensori-motor stream, allowing the robots to collectively restructure their joint experiences (i.e., to dream collectively). Robots can also exchange generalized policies (i.e., after restructuring individually). This could allow for robots to exchange knowledge even if they are physically different: an arm robot can then pass knowledge about the effect of pressing some button to a wheeled robot. We will compare these two levels of knowledge exchange in terms of metrics such as efficacy and diversity of resulting behaviour, rate of learning, etc.;

SELF-ADAPTATION The parameters of the social learning implementation can be designed to be self-adapting on the basis of the same interactions that implement the exchange of knowledge. We want to investigate which parameters benefit from such an approach and to design algorithms that implement parameter control. Similar metrics as for the previous research question apply.

Part II.

Scenarios for scientific validations

This part describes the scenarios to be used in DREAM for scientific publications and for the final demonstrations. These scenarios aim at providing a general framework for DREAM experiments. DREAM experiments have two different goals:

1. allow to test DREAM related hypotheses;
2. demonstrate the abilities gained through the algorithms and approaches developed in DREAM.

The scenarios are defined to provide the context for the definition of both kinds of experiments and on all DREAM related topics.

In each chapter, the main principles and motivations for the scenario are presented. In the "detailed description" section, the scenario is described together with a proposed implementation of the methodology described in chapter 2, including:

- tasks and domains
- expert and naive policies

The data to store for an experiment following this scenario is mentioned in another section.

An experiment will be built either with a precise hypothesis to test or to demonstrate a particular ability. Detailed hypotheses and abilities are not provided in this document, but each chapter will end with a description of how the scenario can address the goals of each work package.

3 SIMULATED FORAGING SCENARIO

3.1 MAIN PRINCIPLE AND MOTIVATION

The simulated foraging scenario is a scenario in which a robot has to collect balls and put them into a basket. It relies on a simplified simulation, aimed at being fast more than realistic. It is intended to be a test bed for DREAM ideas on a simple and easily replicable setup.

To get a reward, the robot needs to exhibit a non-trivial sequence of elementary behaviors. Having a good generalization ability and a fast task resolution time requires then, a priori, to go beyond a dedicated open-loop trajectory to abstract these behaviors and find a way to combine them.

3.2 DETAILED DESCRIPTION

3.2.1 Experimental Setup

The environment contains balls that the robot can collect. The environment also contains walls that can create several different rooms the robot needs to explore to find the basket and the balls. Some other objects can be added like doors or switches. Switches can have an impact on the environment like opening a door or making new balls appear. They are associated with a light that is turned on when the switch is active and off once the robot has activated it. The robot has a sensor that can detect these lights when they are turned on. The robot cannot see the switch when the light is turned off. The basket is also associated to some lights that the robot can detect. See Figure 1 for a typical example of collectball environment.

The robot has three effectors: two motors that drive the robot wheels and a "collectball" effector that tells whether the robot is in a collect mode or not. When this effector is above a threshold, the robot collects a nearby ball; when it is below, it throws the ball it carries, if any. The robot can handle only one ball at a time. The simulation does not handle collisions with balls, it acts as if the robot can move on balls (just as if balls were actually coins the robot can move on without altering its trajectory). The robot has three laser sensors that tell the distance towards obstacles, two binary ball detectors, two binary basket detectors and two binary switch

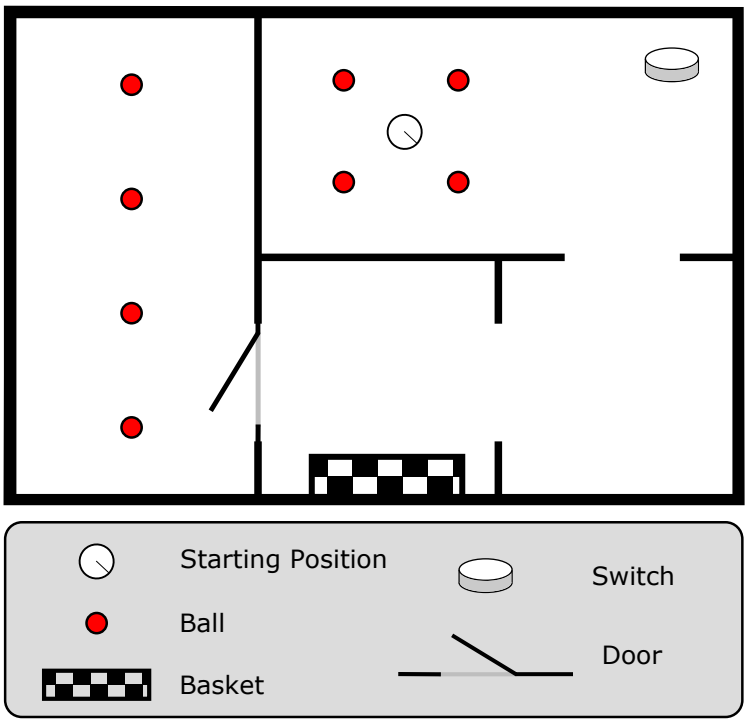

Figure 1.: Robot environment for the collectball scenarios.

detectors (front left and front right), see Figure 2. The robot also has a sensor that tells whether it is carrying a ball or not.

As these are scenarios running in simulation there is no need of specific sensors for the utility functions. The robot trajectory and environment states are directly available.

3.2.2 Tasks and performance criteria

There is a single task here with a single performance criterion that is the number of balls put into the basket.

The experiment is a success if all balls have been collected. A maximum resolution time will be defined in each experiment depending on experiment goal.

3.2.3 Domains

Domains are defined by the following attributes:

- environment map with walls
- robot initial position
- number of balls and ball positions

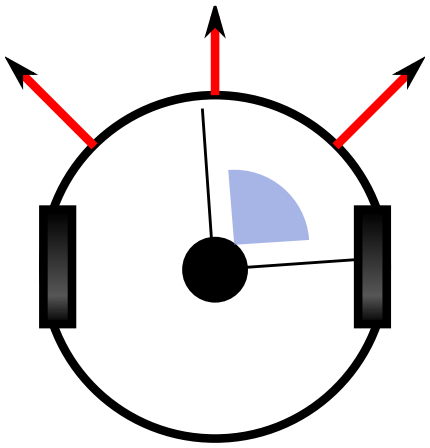

Figure 2.: Robot for the collectball scenarios. The robot has 3 laser-like sensors measuring distances to walls and pie-slice sensors telling whether a particular object is present or not in its field of view (a single right pie-slice sensor is represented on the figure). Such sensors can detect balls, switches or baskets.

- basket position(s)
- **option:** switch position and switch action (opening a door, making a ball appear, etc)
- **option:** door presence, if yes, door position and initial state (opened/-closed)

Low difficulty domains

Arena with no walls will be considered as low difficulty domains. Examples of such domains are shown on Figure 3.

Medium difficulty domains

Setup used in the collectball part of [3]. The arena contains walls separating it into different rooms. Balls and baskets are not in the same room. The robot starts within a room in which there are balls (or in a neighboring one). A switch may be used to give access to another room in which there are other balls to collect. Examples of such domains are shown on Figure 4.

High difficulty domains

Setup inspired from the target task described in [2]. The arena contains several rooms and the robot starts from a room that is very distant from the ones containing the ball. The robot will then need to explore for a

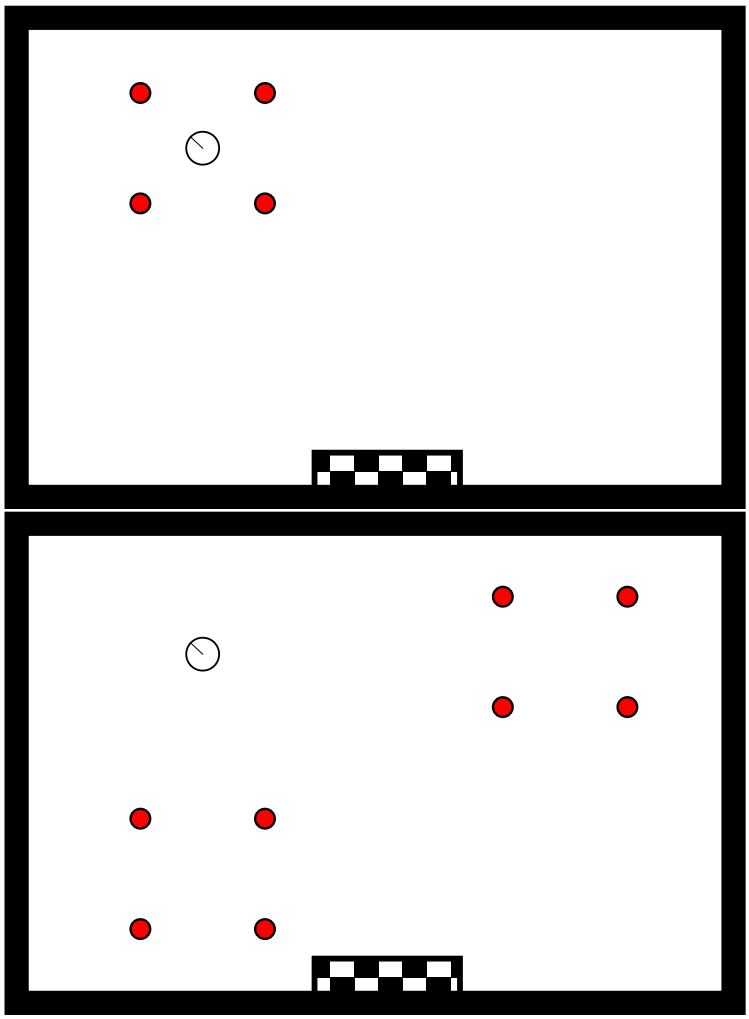

Figure 3.: Examples of low difficulty domains (same legend as in Figure 1).

longer period to discover the balls. The switch may be used, for instance, to give access to the room in which the balls are or to make balls appear, thus increasing the complexity of the behavior to exhibit to collect a ball. Examples of such domains are shown on Figure 5.

Extreme difficulty domains

Extreme difficulty domains are domains in which the sequence to put a ball into the basket is much longer. It may involve to activate switches in particular order, to have doors that cannot be opened at the same time, etc. Examples of such domains are shown on Figure 6.

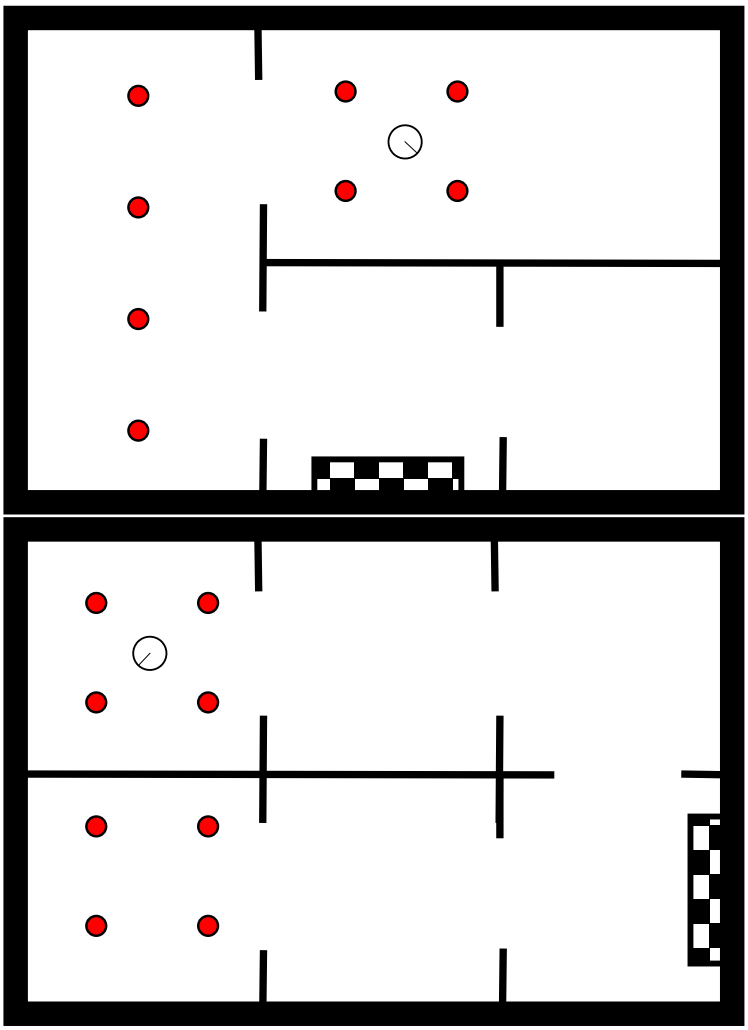

Figure 4.: Examples of medium difficulty domains (same legend as in Figure 1).

3.2.4 Expert policy

The expert policy is based on a subsumption architecture with the following behavioral modules:

1. wall follower (default behavior);

2. move towards a ball (when the robot carries no ball and a ball is perceived):
   - if a ball is perceived on left or right only: move towards it while trying to center this perception (oscillate between seeing the ball on the left and seeing the ball on the right)
   - if two balls are perceived: move forward

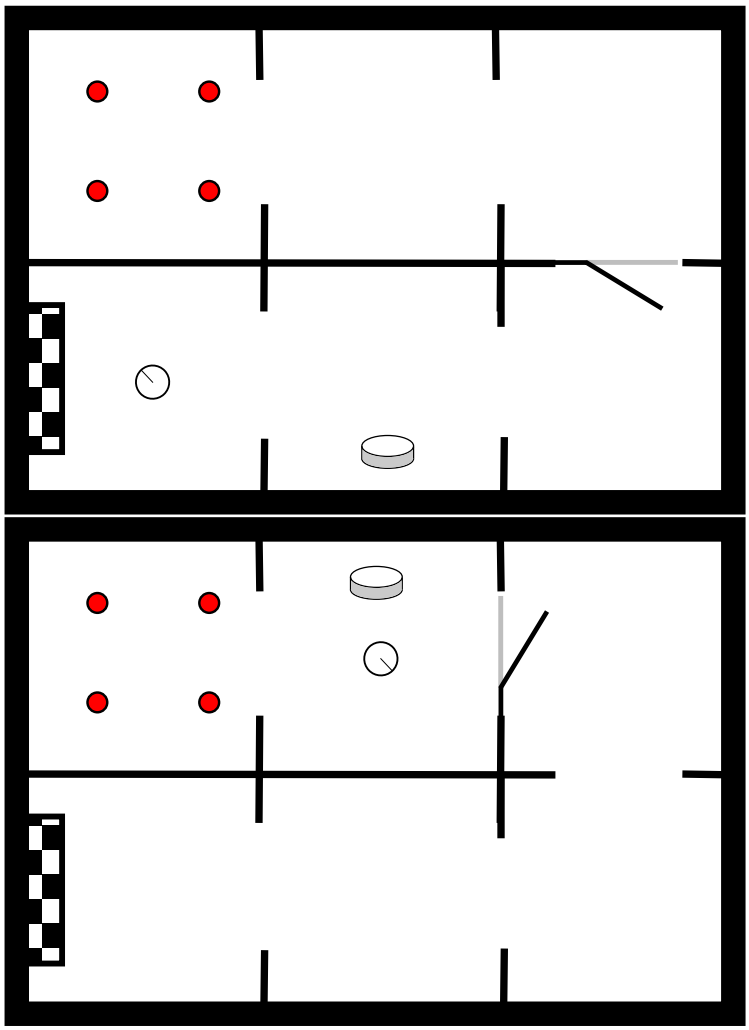

Figure 5.: Examples of medium difficulty domains (same legend as in Figure 1). The switch opens the door.

3. avoid walls to go towards a ball (when there is an obstacle on the way towards the ball): the robot turns left or right 90°, move forward for a while and turns back right or left 90°;

4. go backwards (when the robot bumpers are active): go backwards for a random and short period and turn also for a random and short period to escape from collisions and dead-ends;

5. move towards basket (when the robot carries a ball and sees the basket): similar behavior as for the move towards ball behavior, except that the robot throws the ball when it perceives a basket on both sensors and when bumpers are active

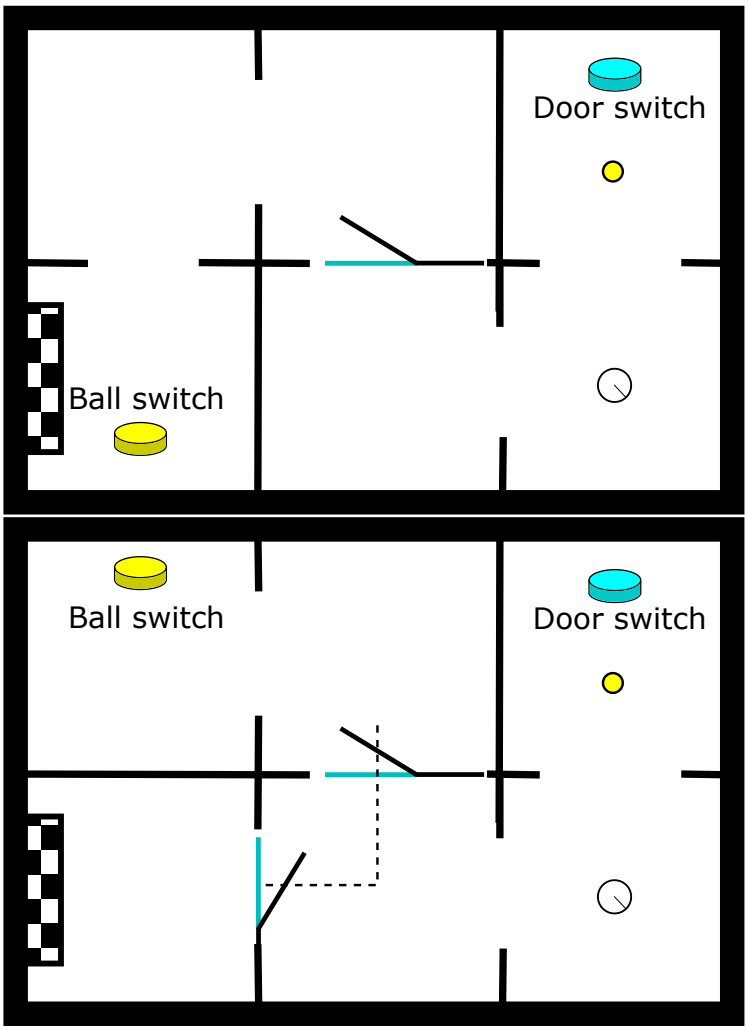

Figure 6.: Examples of extreme difficulty domains (same legend as in Figure 1). The switch opens the door. On the bottom arena, the two doors cannot be opened at the same time.

6. avoid walls to go towards the basket (when the robot carries a ball, sees the basket and has an obstacle in front): similar behavior than for the avoid walls to go towards a ball, except that it is not activated when both basket detectors are active (as it is supposed to mean that the robot is in the final approach of the basket).

As this is a subsumption architecture, the active behavioral module with the highest index is the one controlling the robot.

3.2.5 Naive policy

The naive policy will be the state of the art approach in evolutionary robotics on this task. It aims at showing what is possible without representational restructuring.

We will rely on dynamic behavioral diversity, as described in [3]. The robot is controlled with a neural network whose structure and parameters are explored with the evolutionary algorithm (DNN encoding). The evolutionary algorithm is NSGA-II with two objectives to maximize: the number of collected balls and a dynamic behavioral diversity. The dynamic behavioral diversity measures the mean distance of an individual towards the rest of the population. This objective is to be maximized by evolution, thus pushing towards diverse behaviors. "Dynamic" means that several behavioral distances are used and changed along the run, as described in [3]. Task agnostic behavioral distances will be used:

- edit distance on discretized trajectories
- Euclidean distance between vectors of the sensorimotor values entropy
- Hamming distance between discretized sensorimotor values.

We will switch between them at every generation.

The adhoc distance used in [3] (euclidean distances between the vectors of ball positions at the end of the experiments) will not be used here as it correspond to a task specific knowledge.

3.3 DATA TO STORE

Representation restructuring algorithms will consider learning traces to extract relevant knowledge. To test such algorithms, there is a need to store all data that may be of help without the need to launch a learning algorithm again.

To this end, the data to be stored for a collectball experiment are:

- description of the domain
- for each time step of each policy evaluation:
  - robot perception
  - environment state (ball still present in the environment, switch and door state, etc)
  - robot position

3.4 GOALS OF THE EXPERIMENTS

3.4.1 With respect to WP1

In this scenario, the reward associated to putting a ball into the basket comes after a sequence of actions (going towards a ball, collecting it, going towards the basket, putting the ball into the basket). Driving the learning and developmental process with this reward only leads then to poor results [6]. This scenario can then be a testbed for value functions oriented towards exploration (diversity, novelty, curiosity) or towards intermediate goals that the system should discover by itself (good idea to go towards a ball, to collect balls, to find the basket, etc).

3.4.2 With respect to WP2

Percepts

Perceptions are pre-processed in this task as the robot as a direct access to the presence (or not) of the relevant objects in its field of view (ball, basket and another switch object used to trigger different actions in the environment). From this perspective, the only perception based representation that may make sense here is the concept of the presence of an obstacle in front of the robot. It is a simple concept that would probably be no more than a threshold on the laser sensors.

Policies

The collectball scenario is a perfect playground for policy restructuring. Policies could in particular be decomposed to increase generalization ("go towards a ball" or "find basket" behaviors, for instance). It would in particular imply to find out what information to take into account and what to ignore to make these behaviors robust to domain change.

Forward models

Observations are noiseless in the collectball scenarios and the physical model of the robot is perfectly known, as the simulation actually implements it. It allows then to test, in a simplified setup, the acquisition of a forward model that predicts how actions will influence future perceptions. Concerning WP2, forward models will be sought for a limited set of similar domains. The robustness to more radical changes of domains will be studied in WP3.

3.4.3 With respect to WP3

As for the policies, finding out the relationships between observations and actions may be a key to get models that are robust to a radical change of domain. Such relations are known: wheel motors will have a clear influence on the laser sensors, the "collectball" effector has an impact on the carrying ball sensor, etc.

3.4.4 With respect to WP4

The collectball scenario can be a testbed for social learning. Robots put in different domains may share the knowledge they have acquired. The initial scenario is not adapted for collective learning, but can inspire new experiments dedicated to this kind of approaches, see chapter 4 for an adaptation of this scenario for collective learning.

4 FORAGING SCENARIO: HARDWARE AND COLLECTIVE

4.1 MAIN PRINCIPLE

The previous chapter explained the foraging scenario for simulation purposes. Simulation is a (usually) fast way to test new ideas. The foraging scenario is, for the same reasons explained in the previous chapter, also suitable in a hardware scenario.
In this chapter we explain how we implement the scenario outlined in the previous chapter in a physical environment with mobile robots. As this chapter describes a real-world implementation of the scenario outlined in the previous chapter, we refrain from a repeat of the detailed description of the scenario at a more conceptual level – for that, we refer to chapter 3.

We also describe the collective aspect of this foraging task. "Collective" here is meant in the sense of *learning* collectively, not in the sense of having a task that requires some level of collaboration, task decomposition or coordination between individual robots. In this sense, the considerations regarding relevance to work packages and objectives are the same for the simulated and hardware foraging experiments. The relevance of the experiments vis a vis social learning (the collective aspect, WP4) is emphasized in this chapter, that of other work packages is detailed in chapter 3.

4.2 DETAILED DESCRIPTION

4.2.1 Experimental Setup

The experimental setup consists of the robot, environment, task and performance criteria and data to store. We discuss them one by one.

Robot

The robot platform can be any wheeled robot where we choose the Thymio II robot. For the purpose of the project we need to expand the robot with extra equipment to use a camera and run longer experiments.

THYMIO II Thymio II is an educational robot developed within the framework of the Swiss National Center for Competence in Research Robotics, by a team of the Ecole Polytechnique Fédérale de Lausanne (EPFL), the Ecole d'Art de Lausanne (écal), the Eidgenössische Technische Hochschule Zürich (ETHZ) and the Mobsya association.

The robot's hardware is shown in Figure 7. The Thymio II robot includes seven proximity sensors to detect obstacles, five of which are positioned on the front and two on the back. These sensors can have values in a range between `0` and about `4500`, where the higher value the closer to an obstacle is the corresponding sensor. Two ground sensors are positioned on the bottom of the robot and can have values between `0` and about `1000`, where the lower value the darker is the ground. The robot can move thanks to two differential wheels, meaning that two different speeds in a range between `-500` and `500` can be set in each wheel. For the other sensors, actuators and support, we refer to the Figure 7.

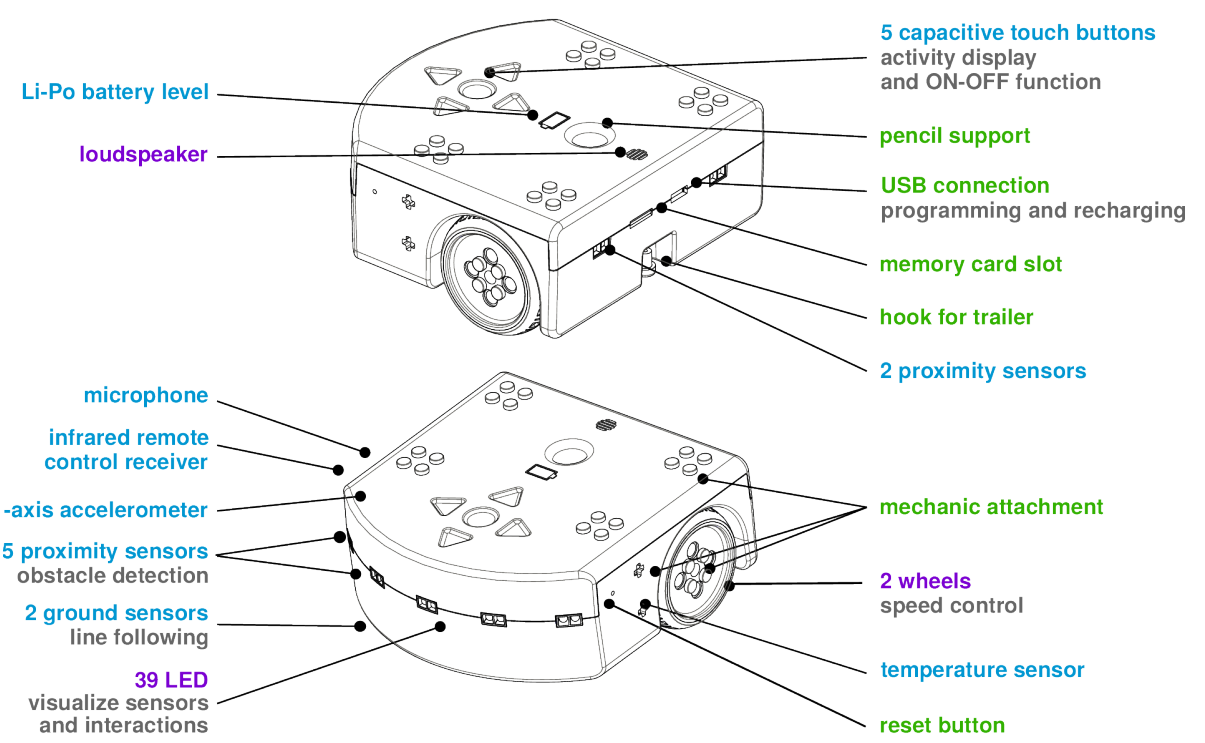

Figure 7.: Specifications of Thymio II robot developed by EPFL and écal. Image is taken from the Thymio website.

EXTENSIONS For the purpose of our research, we extend the standard setup with a more powerful logic board, wireless communication, and a high capacity battery. We intend to use a Raspberry Pi B+ (credit card-sized single-board computer developed in the UK by the Raspberry Pi Foundation) that interacts with the Thymio sensors and actuators. A WiFi dongle attached to the Raspberry Pi ensures communication between the robots. The power is given by a USB battery that allows for a total experimental time of multiple hours.
A 3D printed case developed by ISIR/UPMC for this project holds the necessary equipment. It is shown in Figure 8 and possible extensions are shown in Figure 9.

Figure 8.: Thymio II robot with 3D printed case and possible hardware extensions.

Figure 9.: Extensions of mobile robot includes Raspberry Pi 2, USB battery and WiFi dongle.

Environment

The environment is similar to the simulated environment but needs some modifications for hardware implementation:

- In the simulated environment the balls do not interact with the environment. This is not possible for the hardware scenario. The balls will have a flat bottom so that they cannot roll away from the robot after interaction. We call the balls in this scenario *pucks*.
- The pucks have to be pushed towards a goal because the robot does not have a gripper.
- The goal in the simulated environments is a basket. In the hardware scenario the goal has to be on the same level as the floor because the robot cannot lift the puck. This can be modelled with a hole in the arena or a light source. In the first prototype of the environment shown in Figure 10 the latter is implemented.
- The corners have to be round. This is necessary for the robots so that they cannot push a puck stuck into a corner.
- The walls are white because the Thymio II proximity sensors are more responsive to bright colors than to dark ones.
- In the hardware scenario we will also make use of switches to open a certain room. We will create switches using lego mindstorm robots as shown in Figure 11 where pushing the red button results in opening the door.

Figures 10 and 11 show the first prototype of the smallest environment that will be used for experiments. A tracking system will also be part of this arena to track the robot(s) in the environment.
Multiple environments will be created for the hardware scenario. The main principles of the pucks, target area and switches will stay the same. In this prototype, the pucks are attached to a black circle so that the less advanced Thymio II robots can detect the puck with ground sensors.

4.2.2 Task and performance criteria

The task that the robot has to perform is pushing pucks to the target area. When a puck is successfully pushed to the target area we have to make an extra decision whether the puck is placed back into the environment. This has impact on the performance criteria. When we place the puck back into the arena we can give the robot a fixed time and define the performance as the number of pucks collected. When we do not place the puck back

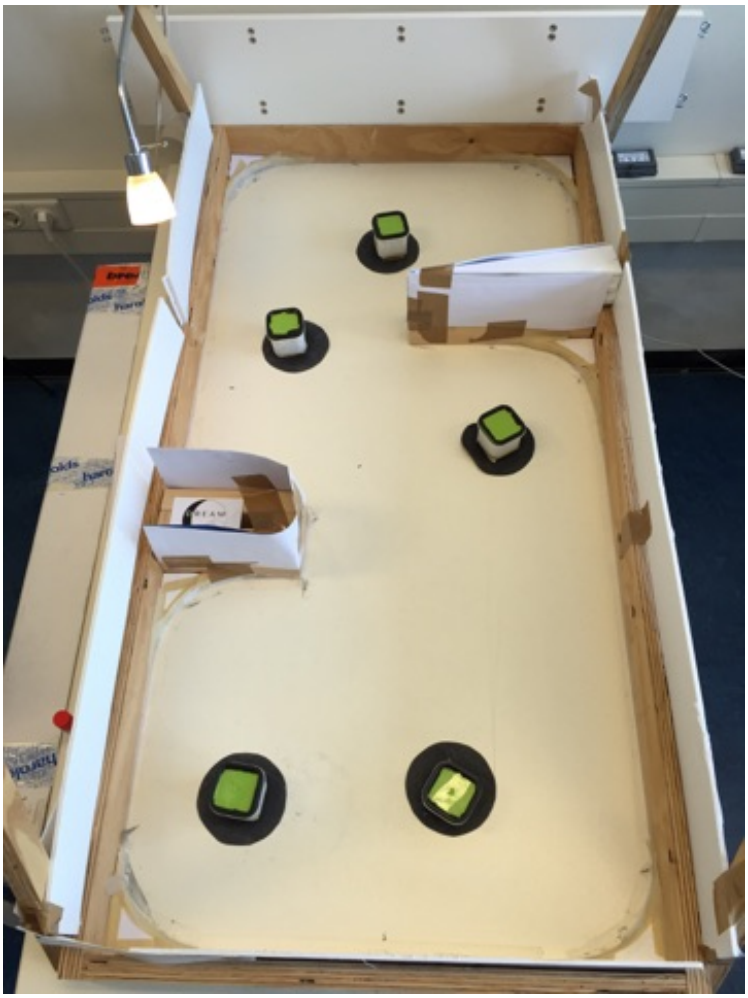

Figure 10.: First prototype of the environment that is used for the experiments.

into the environment we can have the same performance indicator but we can also set the performance to the time that it takes for the robot to collect all pucks. Because of battery restrictions of the robot this is a less desired situation.

4.2.3 Domains

The same attributes as in the simulation scenario can be changed to create low to high difficulty domains.
The expert and naive controllers from the simulation scenario will be reimplemented in the hardware scenario. They are equally relevant for social learning as the task of the robots is individual, not collective. The collective aspect of this scenario lies in *learning* socially, not acting together.

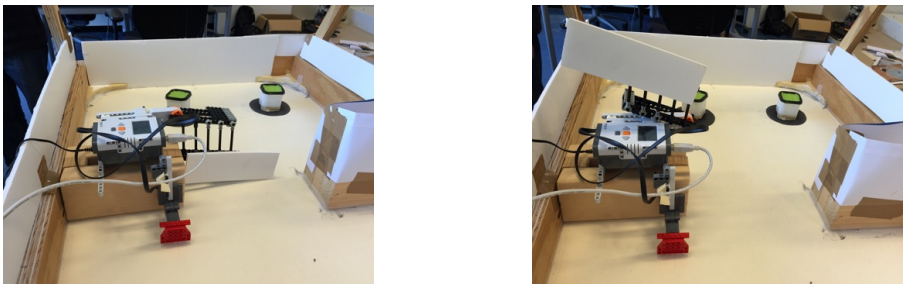

Figure 11.: Switch built with a lego mindstorm robot that will open when pushing the red button

4.2.4 Data to store

Representation restructuring algorithms will consider learning traces to extract relevant knowledge. To test such algorithms, there is a need to store all data that may be of help without the need to launch a learning algorithm again.
The complication we have when using hardware is the maximum possible data to store in one day. We prefer not to use an external computer for storing the data because we do not want the robot to be dependent from communication with other devices.
For every experiment we prefer to store the following data:

- environment setup
- configuration parameters
- trajectory of the robot captured by the tracking system
- time steps when a puck is successfully brought to the target location
- when memory capacity allows, per time step:
  - robot perception
  - environment state (how many pucks are presents and where, state of switch and door)
  - robot position

4.3 GOALS OF THE EXPERIMENTS

With respect to the relevance for WPs 1, 2 and 3, we refer to the previous chapter: the scenario is the same, but now implemented in hardware rather than in simulation.

4.3.1 With respect to WP4

The simulation environment is only designed for one robot but this scenario will have multiple robots executing the foraging task. Multiple robots in the same or different environments results in more experience on a global level. This experiment helps in answering all the research questions listed in Section 2.5.4.

To cater for the collective aspect, the hardware variant of the experiments can extend the single-robot experiment described in the previous chapter in two ways:

- Adding more robots in the same environment. In a large enough arena, this is a very straightforward option. It requires additional tracking capabilities compared to a single robot implementation to be able to track and identify multiple robots simultaneously. Also, it impacts the complexity of the environment as robots can now interfere with each other by bumping into each other, blocking passages or stealing each other's pucks;
- Having two or more (different) environments in parallel, each with a single robot (or possibly a small number of robots). The robots in parallel environments can communicate for the purpose of social learning. Multiple low difficulty arenas can then possibly speed up the learning process in higher difficulty arenas because the lower difficulty instances allow the robots to solve subtasks that generalise to the higher difficulty arena. This variant can also be straightforwardly adopted with the simulated ball-collecting scenario.

As yet, we have not decided in favour of either of these variants and possibly will implement both.

5 ROBOT WITH ARMS SCENARIO

5.1 MAIN PRINCIPLE AND MOTIVATION

The Robot with arms scenario is a scenario in which an armed robot has to interact with objects in various setups. Humans may participate to the learning process. The main setup will be a robot observing a table through a camera and interacting with objects that lie on the table using its arms. It relies on both simulation and various physical robots, from low cost manipulation arms, to complex humanoid robot arms. These experiments will involve complex actuation (several degrees of freedom arms) and perception (using vision among others).

The utility functions will be related mostly to object position and, in some instances, with satisfying the wishes of humans. Depending on the particular domain the corresponding policies will involve several levels of difficulty, from single actions in mostly free spaces to sequences of actions in spaces with obstacles.

5.2 DETAILED DESCRIPTION

5.2.1 Experimental Setup

The environment is made of a planar surface on which various objects will be placed. In the most simple environment, only one object is placed at a pre-defined position. The environment may also contain several objects with different features (e.g. different colors, shapes, weights) and different initial positions. The environment may also contain obstacles that will block the objects motion, or baskets that could receive the objects. The setup can be arranged so that in order to be able to put a given object in a basket, another object has to be moved first (see figure 12).

The robot is placed in front or on the table. It has an arm with varying degrees of freedom. The reachable space of the arm can be only a part of the complete environment. The control can be performed either in the joint space, or in the operational space using hand-coded forward/inverse kinematic. The arm is equipped with a simple gripper or a hand that can be used to push or grasp objects with simple grasping strategies. Precise manipulation is out of the scope of the project. The robots we plan to use

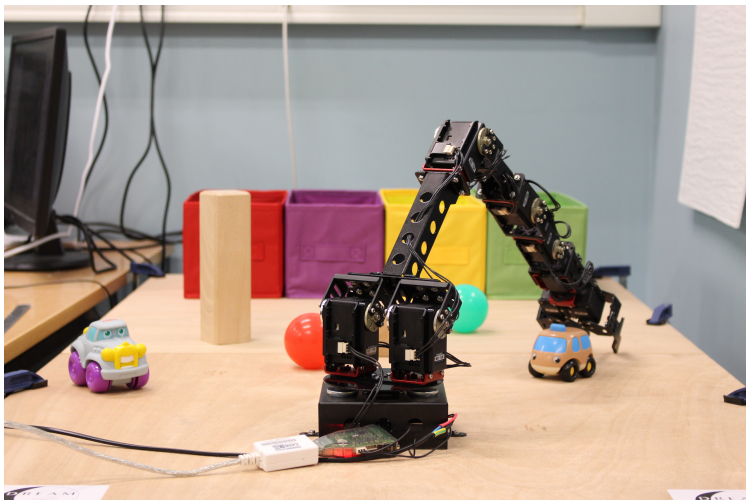

Figure 12.: CrustCrawler arm robot in action.

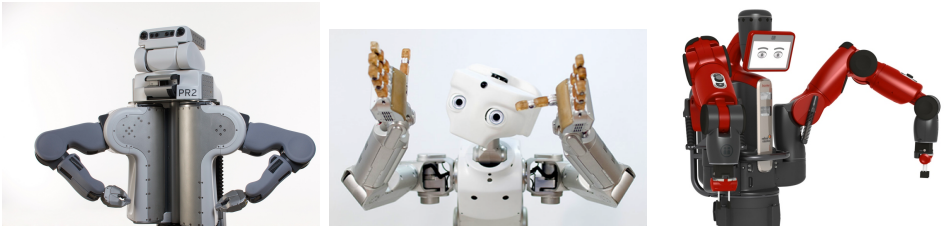

Figure 13.: Left to right : PR2, Meka robot and Baxter robot.

in this setup are the CrustCrawler robotic arm, the PR2, the Meka and the Baxter (see figure 13).

Besides proprioceptive information on the robot joints, the main sensor is a RGB or RGB-D camera. The camera can be positioned for example above the planar surface and observe the scene, including part of the robot arm (see figure 14). The raw images can be used as observation. Task independent pre-processing can also be used, such as computing superpixels, supervoxels or computing local features in order to work in a intermediate level feature space. Beside the camera, other sensors such as touch or pressure sensors on the arm or gripper can be considered in order to take advantage of multi-modal information.

When the experiment is performed in simulation, the utility function (see next section) will be computed using the simulator data. For experiments with the real robot, a camera associated to a specific object detector will be used to monitor the relevant objects positions, including humans when this is the case.

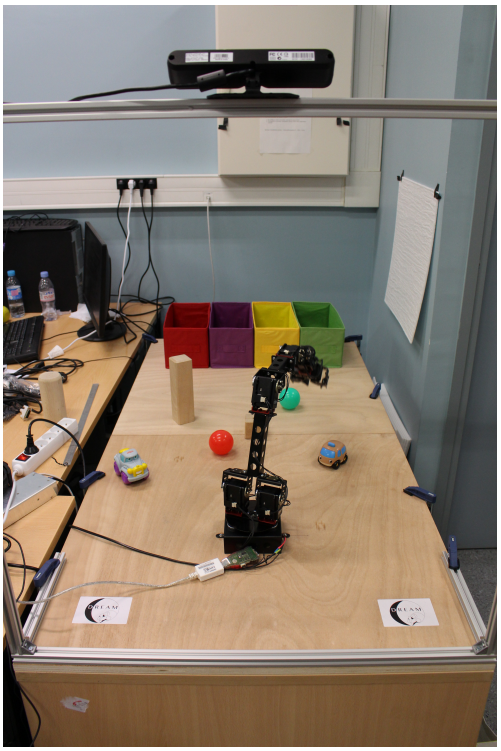

Figure 14.: An example of experimental setup with a CrustCrawler arm and a fixed RGB-D camera.

5.2.2 Tasks and utility functions

Several utility functions, corresponding to tasks with varying degrees of difficulty will be used in these experiments. The initial list of tasks include:

- Touching an object : the utility function will measure the distance between the arm and an object,
- Putting an object to a specific position : the utility function will measure the distance between the objects of interest and the goal positions.
- Putting objects to specific positions according to the objects features (e.g., putting each object in a basket of the same color): the utility function will measure the sum of distance between each object and its goal positions.
- Moving and interacting with objects supervised by humans: the utility function will given by the human in the form of a pain or pleasure signal depending on the robot actions.

The task will be considered a success if utility functions based on distances to the target goes to zero, corresponding for example to the objects

reaching their goal positions. the task involving interaction with humans will be a success if pleasure is maximized and pain minimized.

5.2.3 Domains

Domains covering a large range of difficulties can be defined by varying several parameters of the environment:

- Object initial position, either fixed or random,
- Object type, either always identical, or having different features (color, size, shape, weight...),
- Number of objects to be placed to a goal position : either a single object with a single goal, or several objects and several goals with an association strategy (e.g., put red object in red basket, or put small objects on the left and big objects on the right),
- presence or absence of distractor objects, beside the ones considered in the utility functions,
- necessity or not to move the distractor objets to reach the goal position, in case these objects block the path,
- presence or absence of obstacles to the object motion.

Low difficulty domains

In the most simple domain, the planar surface contains only one object, at a fixed initial position, and a unique goal position that the object should reach.

Medium difficulty domains

Several domains may be defined by varying different parameters:

- The planar surface contains one object but the object initial position is random, and it should be put at a given goal position.
- The planar surface contains one object at a fixed position, but random obstacles should be avoided to reach the goal.
- The planar surface contains one object, but some object feature is random and one out of several goals should be reached according to this feature (e.g., if object is red, put it in the red basket).

High difficulty domains

High difficulty domains can be obtained by imposing intermediate manipulations to reach the goal. For example, the goal is obstructed by a disctractor object that should be removed before putting the target object to the goal position.

Extreme difficulty domains

Even more difficult domains may be defined by combining several constraints together: put several objects to their associated goals (depending on an object feature), in scenarios where obstacle constrain the object displacement order and distractor objects can block the paths, including humans. The inspiration for these scenarios can be found in puzzle solving tasks used to test animal puzzle solving skills.

5.2.4 Expert policy

The expert policies will be hand-designed for each task/domain based on several generic modules that will be arranged to solve the task. These modules include:

- A specific object detector that can detect a given object and output its features (e.g., detect red objet and output its position, size, shape) using the robot camera,
- A PID controller in joint space or in operational space using forward/inverse kinematic,
- A high-level finite state machine controller that will sequence the robot behavior to perform the policy.

Using these modules, the experimenter will easily design a controller that can solve the task.

5.2.5 Naive policy

The naive policy will be the state of the art approach in evolutionary robotics on this task. It aims at showing what is possible without representational restructuring.

We will follow the same approach as described in section 3.2.5. We will rely on evolutionary algorithm, taking a fixed, task-agnostic image feature representation as an input and exploiting the behavioral diversity approach based on novelty in the image feature space in order to explore the space of controllers.

5.3 DATA TO STORE

Representation restructuring algorithms will consider learning traces to extract relevant knowledge. To this end, the data to be stored for a robot with arms experiment are:

- description of the domain
- for each time step:
  - robot perception : images (subsampled in resolution or frequency to reduce storage requirements), joint positions
  - environment state (position of objects)
  - value of utility function

5.4 GOALS OF THE EXPERIMENTS

5.4.1 With respect to WP1

In some of the experiments of this scenario, the utility associated to object interactions will come after some intermediate actions (such as removing an obstacle) that can be used as a testbed to discover useful intermediate value functions. In addition, this scenario will also be used to test the problem of appropriate utility functions for interaction with humans (initially extrinsic pain and pleasure signals, but some intrinsic functions may also be considered)

This experiment can also be a testbed for value functions oriented towards exploration (diversity, novelty, curiosity).

5.4.2 With respect to WP2

Percepts

Observations are potentially complex in this scenario and will rely mainly on vision, using an RGB-D sensor, thus containing color and depth information. Depending on the particular task/domain, these images can be used directly, or can be pre-processed in order to extract information such as superpixels, supervoxels or local image features. Proprioception (joint values) will also be used.

Based on these data, and sets of learning experiments performed during a day, we expect representation restructuring to discover representation spaces that encode feature useful to learn policies or forward models such as particular object types (e.g., cube, ball), object characteristics (e.g., size,color) or object location.

Policies

The robot with arms experiment is a good playground for policy restructuring. Policies could in particular be decomposed to increase generalization ("reach an object" or "grasp an object" behaviors, for instance). It would in particular imply to find out what visual information to take into account and what to ignore to make these behaviors robust to domain change.

Forward models

Learning forward models in these experiments will make it possible to validate the utility of observation representation. Conversely, detecting the perceptual features that are useful for a given forward model (for example in order to predict the utility function) will be useful to drive representation restructuring of perception toward interesting features.

5.4.3 With respect to WP3

Using a larger set of tasks and domains than for WP2, the robot with arms experiments can also be used in the frame of WP3 to discover higher-level representation that generalize to larger variations than WP2.

5.4.4 With respect to WP4

This scenario is not originally designed for collective experiments, but extensions can be designed in order to produce representation restructuring that could be useful to different robots with different sensory setup and in different but related domains.

Part III.

Scenarios for a general public audience

This part describes a set of scenarios demonstrating DREAM related ideas and targeting the general public. The goal is to have robots moving around and demonstrating DREAM related abilities in a context that is appealing for a non expert (typically a child). It creates some constraints that are absent from the former scenarios. People should be willing to engage in the interaction with the robot, and their interest in robot's behavior should be maintained over time. It implies that the capabilities demonstrated by the robot should come up rapidly, without requiring a large set of interactions. Some shortcuts will then be allowed in the developmental process to make it as fast as possible while still demonstrating, to some extent, the targeted abilities.

6 ADOPT A ROBOT INITIATIVE

6.1 MAIN PRINCIPLE

The idea behind the "adopt a robot initiative" is to take the results of the DREAM project into real life scenarios with real humans by means of a low-cost and highly interactive robot platform which can be used for non-expert users (like children) to start dabbling with robots. From the DREAM architecture point of view, this initiative will provide an application scenario implying a real robot and humans interacting with it. It will allow to carry out several different human-robot learning experiences (even implying the different cultural aspects of the users) that can be used to improve the cognitive architecture as well as to gain insights on how humans interact with robots and how this can affect their life-long developmental learning processes.

A noteworthy aspect of this experimental setup and that makes it different from the previous ones, whether simulated or on real robots, is that it relies on human interactions with several users and in different contexts to define useful representations, utility functions, different domains, etc. and this will provide real life-long learning and intrinsically collective problems.

It is also worth indicating here that the "adopt a robot initiative" is not really a scientific experiment in the strict sense of the word, it is rather an experience with the cognitive architecture in real life scenarios. Consequently, there are no real precise experimental objectives in terms of testing specific algorithms with respect to benchmarks. It is more of an observational experience where we seek to gather information on how people respond to the architecture and on how co-learning experiences can take place between humans and robots that use DREAM. As was indicated, it is a dissemination task, but out of which we expect to glean valuable information towards honing in on human dependent factors of the interactions of DREAM based robots with humans that can lead to new projects. Anyway, the experiments that will be defined here will make use of the DREAM architecture as a whole, so this section has been structured mainly as the previous ones.

6.2 DETAILED DESCRIPTION

6.2.1 Experimental Setup

This scenario requires robots and humans interacting in a completely free environment, like a house, a school, or some outdoor location (the human is a master and they cannot be considered as a part of the scenario, so they must not be modeled). Moreover, and in order to speed up learning, different easy to track objects could be used (like colored toys) that the humans could carry in their hand or that can be placed in the same surface as the robot (typically a table or the floor). The remaining elements of the environment are not relevant for this scenario. The robot that will be used in this scenario is the Robobo. The specific type of tasks that could be defined here are grouped in what is called *interactive Robobo games*.

Experimental robotic platform

The Robobo is a mobile robotic platform with an orientable stand to hold the smartphone which controls it. The platform is equipped with two wheels and a set of basic sensors. The smartphone is connected to the platform through bluetooth and it is capable of receiving the sensorial information and sending commands to the motors (platform displacement and orientation of the stand).

The main features of Robobo are:

- Simplicity
- Latest on-board technology (smartphones are frequently updated)
- Capabilities/price high ratio
- Continuous "free" upgrading (very powerful operating systems)
- Human friendly interfaces
- Active open software development community

The smartphone provides the following sensors, actuators and communications capabilities to the robotic system:

- Sensors
  - Vision: Two high-res cameras (3D in the near future)
  - Ambient: proximity, light, temperature...
  - IMU: gyroscopes, accelerometers, magnetometers
  - Audio: microphone
  - Location: GPS / WI-FI / Cell sites

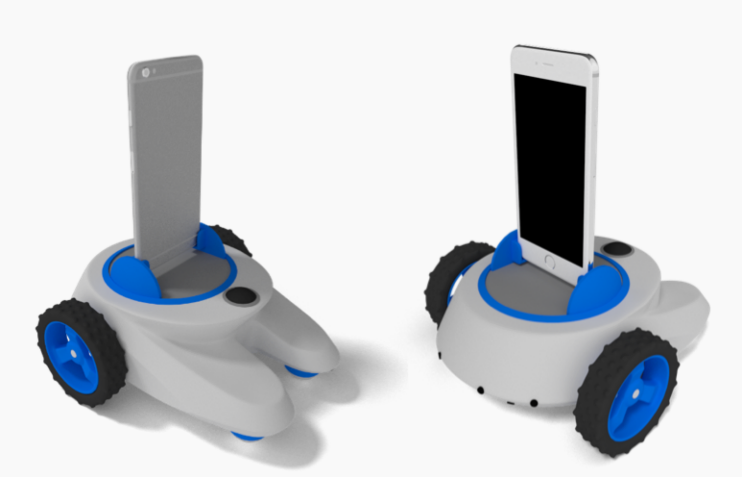

Figure 15.: Robobo prototype design

- Actuators
  - LCD Screen
  - Speaker
  - Communications (mail alarm, text message, phone call...)
- Communications
  - 3G-4G
  - WI-FI
  - Bluetooth
  - USB
  - NFC

The robotic platform is equipped with the following sensors and actuators:

- Sensors
  - Frontal obstacles detection: IR
  - Ground detection: sd-IR
  - Ambient light: ALS
- Actuators
  - Two wheels
  - Pan-tilt unit
  - LED lights

6.2.2 Objectives

The *interactive Robobo games* will be used to reach three main goals:

1. Entertainment: the first goal is related with creating a learning experience that is realistic, amusing and interesting for non-expert users (like occasional users, children...).

2. Educational robotics: a second goal is focused in making non-expert users (both children or adults) familiar with robotics and with the challenges associated with achieving a proper learning. In this case, it is not only the behaviour of the robot but the cooperation between robot and user what will be studied and evaluated.

3. Human interaction tasks: the third goal is to provide a new type of scenario to study and improve the learning processes in the DREAM architecture since this experimental setup will constitute a task where humans are part of the scenario and an active element in the evaluation of the behaviour of the robot.

6.2.3 Tasks and performance criteria

As a first approach, we will present with detail here a first task or game that has been called the *Robobian language*. In this game, there is a goal, which, in an abstract level, implies that the user and the robot should gradually converge to a common language to communicate between them.

More specifically, to reach the entertainment goal established above in this task the user should perceive that he is reaching an agreement with the Robobo in a distracted fashion. Regarding the educational robotics goal, this is enabled by the fact that the user will also adapt their initial idea of the language to some instances of the temporary non-optimal mappings, and this will affect the definitive language. This human adaptation will be guided by an expected improvement in language reliability and/or learning speed, and its effectiveness will be an indication of the intuition of the human about the capabilities and response of the robot. So we expect to obtain a co-adaptive process in model and policy learning. Finally, the DREAM learning goal will be achieved if the different techniques are robust enough for the real time learning in open environments.

According to three goals established in the previous section, three categories of performance can be defined:

1. *Entertainment*: it will be evaluated using satisfaction surveys that will be presented in the smartphone screen during and after the course of the interactive games.

2. *Educational robotics*: its performance will be associated to the learning speed and the efficiency of the language. The learning speed can be simply measured as the real time used to finish the learning (learn a fixed number or commands) and the efficiency of the language will measure its reliability and time-efficiency and it will be evaluated using a language test in which the Robobo will show on the screen a set of commands that the user has to communicate to the Robobo. The time required to send and execute the set of commands will be correlated with the efficiency of the language.

3. *Human interaction experiences*: the reward provided by the user must be maximized and this reward can be an explicit one which will be defined based on a predefined simple rewarding communication code (*for instance some fixed voice commands: good/bad, yes/no*) or an implicit one which will extract the reward based on indirect measurements, such as the number of times the same command is repeated, the intensity of a voice command, the speed of the production of the command (a slower gesture, voice or tactile command will indicate a probable failed previous communication command). Based on these metrics the final performance will be calculated using the criteria established in section 2 based on expert and naive policies and comparing them with those obtained by the DREAM architecture

6.2.4 Domains

Low difficulty domains

A simple initial language game could be an *imitation game*, where the robot will try to associate the movements made by a user carrying a particular object that is easy to track to an specific action (move forward, turn right, turn on the leds...) This game will provide one value function that represents the language for a given user. This task is inspired in that explained in section IV.B of the paper [1]

More specifically, the user can choose from a set of predefined movements of the object to present to the robot (circle, line, zig-zag). The value function will relate the detected movement performed by the user (human related percept), the action taken (self action percept) with the user feedback (utility), that can be obtained as presented in the previous section. In this example, no forward models will be required because the language only indicates actions and not goal states which could require sequences of actions.

Once the value function is learned, the action-selection process is simply an exhaustive search in the predefined action space (learning a policy is not required here).

Medium difficulty domains

This domain is covered by an extended version of the imitation game, in this case the commands will not only indicate specific actions on the robot, but also goal states. For example, one specific command could indicate that the robot has to look towards the face of the human, or to move to a certain position on the floor. In this case, the complexity of the value function will increase since it will also include information from the physical scenario (scenario percepts) and it will require of the creation of forward models (how to reach that spot, how to face a face...) so that those goal states can be found.

High difficulty domains

In this case, the language will increase the complexity of the commands. The percepts which trigger a command could be combinations or sequences of percepts and also the effect of the commands could be sequences of actions and states. It implies the restructuring of the value function to produce efficient policies. For example, in the case of commands that imply a complex sequence of actions (pick the ball that matches the color of the object in my hand and take it to the box) the creation of different value functions which correspond to subgoals will be essential to produce a satisfactory performance.

Extremely difficulty domains

In this case, apart from what was presented in the high difficulty domains, the innate representations of percepts and actions will be reduced. Therefore the architecture will need to find appropiate representations to identify for example movements of the hands (circular, zig-zag, approaching). The restructuring of forward models according to those new representations may also be convenient and definitely the scenario will imply dependent learning processes working at several levels and affected by the particularities of the interactions with a human.

6.2.5 Expert policy

The expert policy is generated in a straightforward manner since it is directly the encoding of the mapping between actions of the user and actions of the robot that the user has decided to use in a set of rules (or any transformation operator which may be required).

6.2.6 Naive policy

The naive policy will be a neuroevolutionary algorithm like NEAT that is used to learn the models from the same level of innate representations that the robot is using. It aims at showing what is possible to achieve with one the state of the art approximation operators without using any cognitive architecture.

6.3 DATA TO STORE

Apart from the general data which is stored in any experimental setup of the cognitive architecture, which will consider all information captured by the robot, this particular experiment will require to store the response of the scenario since in this case this scenario is neither stationary nor known a priori or predefined but dependent on the dynamic response of the users. This additional scenario data will be stored by means of video and audio recordings of the experiments.

Moreover, the satisfaction surveys imply storing the results to analyze the entertainment experience.

6.4 CONTEXTS

We plan to perform the experiments explained on the following contexts:

1. Initial tests during a hackademia performed by expert users
2. Experiments carried by groups of master students as part of a robotic course.
3. Interaction with the general public (non-expert users: class of young children, etc.)

6.5 GOALS OF THE EXPERIMENTS

Apart from the aims of the experiments which will fit in the following subsections according to the distinct WP's and which correspond to the adaptive interaction experiments, the adopt a robot initiative also pursues the following contributions in a social level, as stated previously:

- Make general users aware of what is behind the interaction with a robot in order to involve them in the specifics of the learning process.

This will arise automatically from the observation of the response of the robots. Nonetheless it will be also convenient to provide certain feedback to the user regarding the internal processes of the robot to help them understand what works or fails and how (i.e. a robot may not be detecting a blue ball either because the lighting changed and target color is not correct, or because the ball isn't in its field of view, or because it's simply looking for another object...).

- Help to grow an interest for robotics in children. Promote active games and an association between leisure and learning.

6.5.1 With respect to WP1

The adaptable interactive experiments, like in general learning tasks, will make use of a reward to drive the learning (adaptation) process. This reward will be in this case a particular one since it will be generated based on an online feedback from the user. This positive (or non-negative) feedback may be either an implicit evaluation (namely: the amount of attention the robot captures from the user, how precise the communication is between them, how far the user is, how much the user moves or any other...) or an explicit evaluation where the user knows and uses a reward/penalty code. Apart from the specific reward that this experiment will have, the *interactive games* show no difference with the rest of the learning processes considered in the project and as for them, the task and domain should include a complexity and richness of possibilities which allow the learning algorithm to significantly benefit from producing intermediate estimated rewards (value decomposition) associated to the emergence of intermediate or instrumental goals and expected values associated to non-rewarded states. These features are presented more in detail along with the specific experiments proposed.

6.5.2 With respect to WP2

Since from the very first episodes of the interactive experiments the robots are expected to produce acceptable behaviors and after a reasonable number of episodes the behavior should be good enough as to grab the users attention, some innate medium-level representations are required here and from there the cognitive architecture should develop higher level, more efficient representations.

Percepts

The robot will be provided with several real time computer vision and speech recognition algorithms. From those representations it will build

more complex representations as we already mentioned (circular movement, approaching hand, ...). Since we deal with a real life problem, there is no doubt that countless useful high level perceptual representations may arise.

Policies

One of the features that may trigger the restructuring of a policy is the creation of new perceptual or action representations. The fact that the experiment will easily produce new perceptual representations is directly implying that new policies based on those new representations will also arise.

Another possibility is the emergence of new policies to decompose current ones so that they are more accurate and more general. Adaptive interactive games are also very appropriate to make use of combinations of policies since they work in changing environments which may trigger context based policies.

Forward models

The presence and great involvement of human actions in the experiment will produce forward models that will have to deal with very noisy and dynamical scenarios. Therefore, the representations should be adequate and the models generic so that they can capture not the whole response of the environment to the actions of the robot but rather some general regularities which are useful for the cognitive mechanism.

6.5.3 With respect to WP3

Apart from the usual considerations of the process of consolidation in a life-long learning process, consolidation in this case operates at two levels, local and global consolidation. The first one will strengthen some parts of the forward models and value functions which have to do with the particular scenario (specific user or users and physical location) and the second one will consolidate pieces of knowledge which may be of interest for all the robots in all the environments by means of a global centralized server to all the robots.

6.5.4 With respect to WP4

This is, in this case, connected to the consolidation process as mentioned before. In a way, the adaptive interactive games will be considered as a big collective experiment in which each robot in each environment will be providing interaction information and adding knowledge to the central

long term memory which will serve as a reservoir to enrich and improve the learning process with a huge set of episodes coming from different domains. The way knowledge is collectively shared and reused will be fundamental to the success of the experiment.

Specifically, different languages can arise in different robots, so in the sleeping-scale processes we could analyse the different policies and forward models and try to generalize them. These more general models and policies could make up a sort of basic language for the robots that could be used with users in the first stages of learning as seeds or in those cases where the learning is complex. So, the collective aspects that can be studied in this experiment are very relevant for the DREAM architecture.

Part IV.

Conclusion

This document has been written during the first six months of the DREAM project. As the ideas explored during the project are new, at least for project members, they may evolve during time and this document is not more than a reflect of our current thoughts after the first months of the project. After the first experiments, new bottlenecks can be identified that will raise new questions and new hypotheses.

The document does not deal with validation experiments integrating each partner and work package contribution. The scenarios have been designed to have the capability to demonstrate such an integration, but as long as the exact abilities we will get are not yet precisely identified, it has revealed to be premature to design a precise validation experiment. A typical experiment validating the integration of each contribution would take the form of a robot with arms experiments that discovers objects in its surrounding environment, discovers what it can do with it and learns how to use them to reach a desired effect. This knowledge would be shared between different robots with arms, may they have an identical morphology or not. Beyond this short description, the precise definition of these experiments will require to identify more precisely the ability of our algorithms to deal with the issues that are raised by such a scenario.

BIBLIOGRAPHY

[1] F. Bellas, R.J. Duro, A. Faina, and D. Souto. Multilevel darwinist brain (mdb): Artificial evolution in a cognitive architecture for real robots. *IEEE Transactions on Autonomous Mental Development*, 2(4), 2010. (Cited page 77.)

[2] S. Doncieux. Knowledge extraction from learning traces in continuous domains. In *AAAI 2014 fall Symposium "Knowledge, Skill, and Behavior Transfer in Autonomous Robots"*., 2014. (Cited page 45.)

[3] S. Doncieux and J.B. Mouret. Behavioral diversity with multiple behavioral distances. In *Proc. of IEEE Congress on Evolutionary Computation, 2013 (CEC 2013)*, 2013. (Cited pages 45 and 50.)

[4] Stephane Doncieux. Transfer learning for direct policy search: A reward shaping approach. In *IEEE International Conference on Development and Learning and on Epigenetic Robots*, 2013. (Cited page 24.)

[5] N Krüger, C Geib, J Piater, R. Petrick, M. Sttedman, Wörgötter F., and R. Dillmann. ObjectâĂŞAction Complexes: Grounded abstractions of sensoryâĂŞmotor processes. *Robotics and Autonomous Systems*, 59(10):740–757, 2011. (Cited pages 17 and 21.)

[6] J.-B. Mouret and S. Doncieux. Encouraging behavioral diversity in evolutionary robotics: an empirical study. *Evolutionary Computation*, 20(1):91–133, 2012. (Cited page 51.)

[7] Freek Stulp, Laura Herlant, Antoine Hoarau, and Gennaro Raiola. Simultaneous on-line discovery and improvement of robotic skill options. In *IROS*, 2014. (Cited page 24.)

[8] Yongxin Yang and Timothy Hospedales. A unified perspective on multi-domain and multi-task learning. In *International Conference on Learning Representations*, 2015. (Cited page 24.)