Elsevier Editorial System(tm) for Robotics and Autonomous Systems
Manuscript Draft

Manuscript Number:

Title: Object-Action Complexes: Grounded Abstractions of Sensorimotor Processes

Article Type: Full Length Article

Keywords: Object Action Complexes; Affordances; Cognitive Architecture; Grounding; Planning

Corresponding Author: Prof. Dr. Norbert Krueger, Ph.D.

Corresponding Author's Institution:

First Author: Norbert Krueger

Order of Authors: Norbert Krueger; Justus Piater; Christopher Geib; Ronald Petrick; Mark Steedman; Florentin Wörgötter; Aleš Ude; Tamim Asfour; Dirk Kraft; Damir Omrcen; Alejandro Agostini; Rüdiger Dillmann

Abstract: Autonomous cognitive robots must be able to interact with the world and reason about their interactions. On the one hand, physical interactions are inherently continuous, noisy, and require feedback. On the other hand, the knowledge needed for reasoning about high-level objectives and plans is more conveniently expressed as symbolic predictions about state changes. Bridging this gap between control knowledge and abstract reasoning has been a fundamental concern of autonomous robotics.

This paper proposes a formalism called an Object-Action Complex as the basis for symbolic representations of sensorimotor experience. OACs are designed to capture the interaction between objects and associated actions in artificial cognitive systems. This paper defines a formalism for describing object action relations and their use for autonomous cognitive robots, and describes how OACs can be learned. We also demonstrate how OACs interact across different levels of abstraction in the context of two tasks: the grounding of objects and grasping affordances, and the execution of plans using grounded representations.

Suggested Reviewers: Erol Sahin
Dept. of Computer Engineering, Middle East Technical University
erol@ceng.metu.edu.tr

Bernhard Nebel
Institut für Informatik, Albert-Ludwigs-Universität Freiburg
nebel@informatik.uni-freiburg.de

Michael Beetz
Computer Science Department, Chair IX, TU München
beetz@in.tum.de

Charlie Kemp
Health Systems Institute, Georgia Institute of Technology
charlie.kemp@hsi.gatech.edu

Benjamin Kuipers
Computer Science and Engineering, University of Michigan
kuipers@umich.edu

Object-Action Complexes: Grounded Abstractions of Sensorimotor Processes

Norbert Krüger[a], Justus Piater[b], Christopher Geib[c], Ronald Petrick[c], Mark Steedman[c], Florentin Wörgötter[d], Aleš Ude[e], Tamim Asfour[f], Dirk Kraft[a], Damir Omrčen[e], Alejandro Agostini[g], Rüdiger Dillmann[f]

[a]*Mærsk McKinney Møller Institute, University of Southern Denmark, Odense, Denmark*
[b]*Montefiore Institute, Université de Liège, Liège, Belgium*
[c]*School of Informatics, University of Edinburgh, Edinburgh, Scotland, UK*
[d]*Bernstein Center for Computational Neuroscience (BCCN), Göttingen, Germany*
[e]*Jožef Stefan Institute, Department of Automatics, Biocybernetics, and Robotics, Ljubljana, Slovenia*
[f]*Institute for Anthropomatics (IFA), Humanoids and Intelligence Systems Laboratories (HIS), Karlsruhe Institute of Technology, Karlsruhe, Germany*
[g]*Institut de Robotica i Informatica Industrial (CSIC-UPC), Barcelona, Spain*

Corresponding author:
Nobert Krüger
The Maersk Mc-Kinney Moller Institute
University of Southern Denmark
Campusvej 55
DK-5230 Odense M
Denmark
fax: (+45) 66 15 76 97
email: norbert@mmmi.sdu.dk

Object-Action Complexes: Grounded Abstractions of Sensorimotor Processes

Norbert Krüger[a], Justus Piater[b], Christopher Geib[c], Ronald Petrick[c], Mark Steedman[c], Florentin Wörgötter[d], Aleš Ude[e], Tamim Asfour[f], Dirk Kraft[a], Damir Omrčen[e], Alejandro Agostini[g], Rüdiger Dillmann[f]

[a]*Mærsk McKinney Møller Institute, University of Southern Denmark, Odense, Denmark*
[b]*Montefiore Institute, Université de Liège, Liège, Belgium*
[c]*School of Informatics, University of Edinburgh, Edinburgh, Scotland, UK*
[d]*Bernstein Center for Computational Neuroscience (BCCN), Göttingen, Germany*
[e]*Jožef Stefan Institute, Department of Automatics, Biocybernetics, and Robotics, Ljubljana, Slovenia*
[f]*Institute for Anthropomatics (IFA), Humanoids and Intelligence Systems Laboratories (HIS), Karlsruhe Institute of Technology, Karlsruhe, Germany*
[g]*Institut de Robotica i Informatica Industrial (CSIC-UPC), Barcelona, Spain*

Abstract

Autonomous cognitive robots must be able to interact with the world and reason about their interactions. On the one hand, physical interactions are inherently continuous, noisy, and require feedback. On the other hand, the knowledge needed for reasoning about high-level objectives and plans is more conveniently expressed as symbolic predictions about state changes. Bridging this gap between control knowledge and abstract reasoning has been a fundamental concern of autonomous robotics.

This paper proposes a formalism called an Object-Action Complex as the basis for symbolic representations of sensorimotor experience. OACs are designed to capture the interaction between objects and associated actions in artificial cognitive systems. This paper defines a formalism for describing object action relations and their use for autonomous cognitive robots, and describes how OACs can be learned. We also demonstrate how OACs interact across different levels of abstraction in the context of two tasks: the grounding of objects and grasping affordances, and the execution of plans using grounded representations.

Keywords: Object Action Complexes, Affordances, Cognitive Architecture, Grounding, Planning.

1. Introduction

Autonomous cognitive robots must be able to interact with the world and reason about the results of those interactions, a problem that requires overcoming a number of representational challenges. On the one hand, physical interactions are inherently continuous, noisy, and require feedback (e.g., move forward by 42.8 centimeters or until the forward pressure sensor is triggered). On the other hand, the knowledge needed for reasoning about high-level objectives and plans is more conveniently expressed as symbolic predictions about state changes (e.g., going into the kitchen enables retrieving the coffee pot). Bridging this gap between control knowledge and abstract reasoning has been a fundamental concern of autonomous robotics [1, 2, 3, 4]. However, the task of providing autonomous robots with the ability to build symbolic representations of continuous sensorimotor experience *de novo* has received much less attention, even though this capability is crucial if robots are ever to perform at levels comparable to humans.

This paper proposes a formalism called an *Object-Action Complex* (OAC, pronounced "oak") as the basis for symbolic representations of sensorimotor experience. OACs are designed to capture the interaction between objects and associated actions in artificial cognitive systems [5, 6]. In particular, this paper:

- defines a formalism for describing object-action relations and their use in autonomous cognitive robots,
- describes how OACs can be learned, and
- demonstrates how OACs interact across different levels of abstraction.

We will illustrate this concept using a number of example OACs. These OACs will interact in a learning system that uses exploration to autonomously acquire object-dependent grasp affordances, and create plans composed of action sequences.

2. Motivation

Object-Action Complexes (OACs) are a universal representation enabling efficient planning and the execution of purposeful action at all levels of a cognitive architecture. OACs combine the representational and computational

efficiency of STRIPS rules [7] and the object- and situation-oriented concept of affordance [8, 9] with the logical clarity of the event calculus [10, 11]. Affordance is the relation between a situation, usually including an object of a defined type, and the actions that it allows. While affordances have mostly been analyzed in their purely perceptual aspect, the OAC concept defines them more generally as state-transition functions suited to prediction. Such functions can be used for efficiently learning the multiple representations needed by an embodied agent for symbolic planning, execution, and sensorimotor control.

2.1. Representational Congruency

To achieve its goals in the real world, an embodied agent must develop predictive models that capture the dynamics of the world and describe how its actions affect the world. Building such models, by interacting with the world, requires overcoming certain representational challenges imposed by

- the continuous nature of the world itself,
- the limitations of the agent's sensors, and
- the stochastic nature of real world environments.

OACs are proposed as a framework for representing new actions and objects at all levels of abstraction, from the discrete high-level planning and reasoning processes all the way down to continuous low-level sensors and effectors. While multiple representations may be necessary, due to the diversity of components required in a complex system, building different representations for the same domain is only helpful if solving the problem at a higher level of abstraction also solves (or at least informs the solution of) the problem at a lower level of abstraction. We call this property *representational congruency*. That is, high-level plans must be interpreted in terms of low-level effector commands, and evidence of their success or failure must be recoverable in real time from the agent's sensors.

Figure 1 illustrates this idea with an OAC that predicts the behaviour of a low-level control program CP functioning in the real world to move an agent's end effectors. Since the agent's perception of the world is completely mediated by its sensors and effectors, any change in the world can only be observed by the agent through its (possibly faulty) sensors. Thus, executing

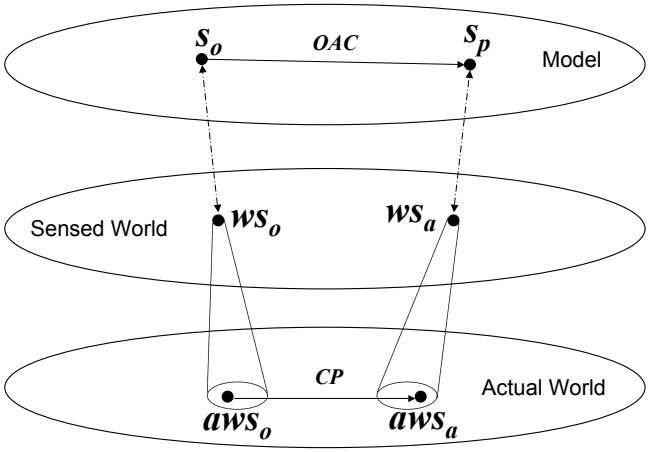

Figure 1: Graphical representation of an OAC and its relationship to a control program

CP causes the actual state of the world to move from an initial world state aws_o (sensed as ws_o) to some resulting state aws_a (sensed as ws_a).

For an OAC to be effective for planning, its higher level states must map to states that are equivalent to those the control program actually produces. For instance, if ws_o maps to state s_o and ws_a maps to state s_p then all OACs that model this particular CP must also map s_o to s_p to maintain representational congruency. Thus, we envision real-time cognitive systems as using OACs to solve a problem at one level of abstraction such that the resulting solution can be understood in terms of the lower levels of abstraction, even down to the level of the agent's sensors and effectors.

In practice, we can simplify this diagram slightly. Because the available sensor suite of a given agent is fixed, we can treat the actual world and the sensed world as a single level, as shown in Figure 2. We will make this assumption for the remainder of the paper.

2.2. Design Principles

Six design principles underlie and motivate our formalization of OACs. As motivation for our later formal definitions, we briefly introduce these principles here.

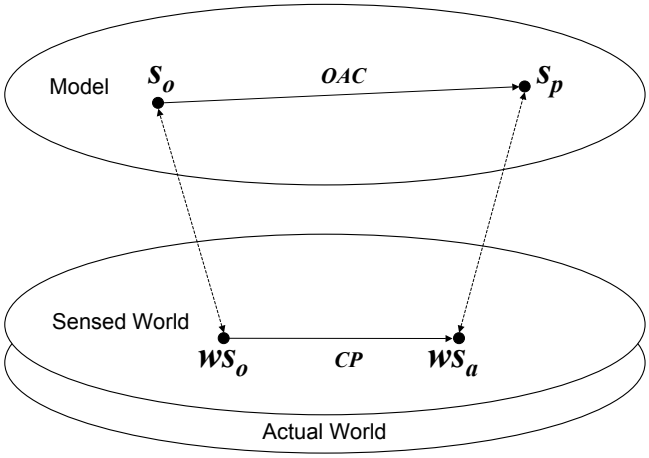

Figure 2: Graphical representation of an OAC and its relationship to the sensed world and a control program

P1 **Attributes:** Any formalization of actions, observations, and interactions with the world requires the specification of an attribute space and associated values that our definitions will operate over. An agent's expectations and predictions about how the world will change will be defined over subspaces of this attribute space.

While the attribute spaces for different levels of action representation may differ, all levels of representation must be downwardly congruent. That is, higher-level (more abstract) attribute spaces must be related to lower-level (less abstract) attribute spaces by a (possibly partial) functional relation that establishes corresponding states. This allows low-level state information to be used by higher-level OACs and guarantees that higher-level OACs reflect actual changes from the lower levels.

P2 **Prediction:** A cognitive agent performing an action to achieve some effect must be able to predict how the world will change as a result of its actions. That is, it must know which attributes of the world must hold for an action to be possible (which will typically include the presence of an object), which attributes will change, and how they will change as a result of the action. Such representations will typically be *partial*,

i.e., defined over a subspace of the attribute space. Again, predictions at all levels must be congruent, so that high-level action predictions can be interpreted at lower levels, and low-level changes in the world can be captured by high-level features.

P3 **Execution:** Many previous efforts to produce fully autonomous robotic agents have been limited by simplifying sensor, action, and effector models. We instead take the approach that complete robotic systems must be built with the means to actually perform actions in the world and evaluate their success. This requires agents to be embodied within physical systems interacting with the physical world.

P4 **Evaluation:** In order to improve its performance in a nondeterministic physical world, an agent must be able to evaluate the effectiveness of its actions, by recognizing the difference between the states it predicted would arise from its actions, and the states that actually resulted from action execution.

P5 **Learning:** State and action representations are dynamic entities that can be extended by learning in a number of ways: continuous parameters can be optimized, attribute spaces can be refined or extended, new control programs can be added, and prediction functions can be improved. Embodied physical experiences with actions, predictions, and outcomes deliver the input to such processes at all levels of the system.

P6 **Reliability:** It is not sufficient for an agent merely to have a model of the changing world. It must also learn the reliability of this model. Thus, OACs must maintain measurements that capture the accuracy of their predictions over past executions.

The rest of this paper is organised as follows. Section 3 provides a formal definition of the OAC concept, based on the above design principles. Section 4 characterizes how OACs learn. Section 5 describes how OACs are executed within a physical robot system. Section 6 gives examples of OACs represented at different levels of a cognitive architecture. Section 7 extends our prior examples to demonstrate the interaction between OACs within the same system. Section 8 discusses the relationship between OACs and other existing representations in the literature. Finally, we conclude in Section 9.

3. Definitions

Our OAC definition is split into two parts: a *symbolic description* consisting of a prediction function [P2] that operates on a mental model (i.e., attribute space [P1]) of the world, and an *execution specification* [P3] that defines how the OAC is executed by the embodied system. This separation is intended to capture the difference between the knowledge needed for action applicability and effect reasoning (represented in the symbolic description), and the procedural knowledge required for execution (encapsulated in the execution specification). Furthermore, OACs are not limited to continuous or discrete representations of actions. Instead, our definitions are flexible enough to accommodate both kinds of representations, as we will see in Section 6. In the remainder of this section we will describe an OAC's formal description. The execution specification will be discussed in Section 5.

We begin with a set of definitions.

Definition 3.1. *An **attribute space** $\mathcal{S}$ is the set of all possible configurations of a world model. A point $s \in \mathcal{S}$ denotes a **state** within the space.*

Definition 3.2. *An **Object-Action Complex (OAC)** is a triple*

$$(\texttt{id}, T, M) \tag{1}$$

where:

- $\texttt{id}$ *is a unique identifier,*
- $T : \mathcal{S} \to \mathcal{S}$ *is a prediction function encoding the system's beliefs as to how the world (and the robot) will change if the OAC is executed [P2], and*
- M *is a statistical measure representing the success of the OAC within a window over the past [P6].*

As notation, we will use range(T) and domain(T) to denote the range and domain of T respectively. In general, much of $\mathcal{S}$ will be irrelevant for many OACs. Thus, we anticipate that both the range and domain of T will typically be subsets of $\mathcal{S}$. Since observations are costly in real world systems, we can often use range(T) and domain(T) to more efficiently allocate system resources for verifying OAC execution, thereby reducing sensor load.

Different OACs within the same agent may be defined on very different state spaces. For example, consider an OAC defined on an attribute space that includes an end-effector's joint space and the location of a ball. Such an OAC might make predictions, given a particular torque, of the final position of the end-effector and the trajectory of the ball. In contrast, the same agent might also have a more abstract OAC that describes the game of basketball. In this case, the OAC might predict that exerting the same torque will result in the ball scoring two points.

Given the diversity of state spaces that an OAC can be defined on, M must be flexible enough to capture the reliability of the OAC's prediction function. As a result, we allow each OAC to define M as an appropriate statistical measure for its needs. Thus, different OACs in a single system might define M in very different ways. For example:

1. In a simple domain where an OAC is used until it fails and then is never used again, we might define M as a Boolean flag that indicates whether the OAC has failed.
2. In a more complex domain where M tracks the accuracy of an OAC's prediction function over time, we might also want to know how reliable the estimate of that accuracy is. If $M^{\langle\rangle}$ indicates the expectated value of the OAC's performance, and N specifies the reliability of these estimates in terms of the number of past experiences, then we could define M as a pair that contains these two values.
3. In even more complex domains it might be convenient to store statistical data beyond $M^{\langle\rangle}$ and N, e.g., lower-level OACs might maintain differences in specific attributes.

We will give further examples of OACs and their reliability measures in Section 6.

In order to discuss how an OAC's T and M are learned we provide the following two definitions.

Definition 3.3. *Given an attribute space $\mathcal{S}$ and an OAC with identifier $\mathtt{id}$ defined on $\mathcal{S}$, an **experiment** is a tuple*

$$(s_o, \mathtt{id}, s_p, s_a) \tag{2}$$

where:

- $s_o \in \mathcal{S}$,

- $s_p \in \mathcal{S}$ *such that OAC* $\boldsymbol{id}$ *predicts state* s_p *will result from its execution in* s_o*, i.e.,* $s_p = T_{\boldsymbol{id}}(s_o)$*, and*
- $s_a \in \mathcal{S}$ *such that* s_a *is observed as a result of actually executing OAC* $\boldsymbol{id}$ *in state* s_o*.*

Thus, an experiment is an *empirical event* grounded in sensory experience. As such, experiments can be used to update OACs in cycles of execution and learning (see Section 6) based on evaluations of their success [P4].

Definition 3.4. *Let* `execute` *be a function that maps an OAC id to an experiment, i.e.,*

$$\texttt{execute} : \boldsymbol{id} \rightarrow (s_o, \boldsymbol{id}, s_p, s_a). \tag{3}$$

The `execute` function should be interpreted as an operation that executes the control program specified by the OAC in the current world state, returning an experiment containing: the state s_o in which execution began, the OAC `id` that was executed, the state s_p the OAC predicted would result from its execution in state s_o, and the state s_a that actually resulted from the OAC's execution.

We note that there can be significant differences between s_p and s_a. In fact, there is no reason why s_a must even fall within range($T_{\texttt{id}}$). (E.g., range($T_{\texttt{id}}$) may be incorrect and not include attributes that are relevant to the OAC instance that we want to learn.) More generally, there is no requirement that the features of an attribute space be relevant to the OAC's prediction function. The prediction function need not be defined over the whole attribute space or make use of all of the attributes in the space. This means the attribute space can (and in our examples will) contain things that are not part of the domain or range of the prediction function. However, it is important to keep in mind that all the features changed by executing the OAC are reflected in the states returned by an experiment, even if those states are not within the domain and range of the prediction function.

In the next section we will discuss how an agent's OACs are learned [P5] from the information provided by its day to day experiences in the form of experiments.

4. Learning

Recall from Figure 2 that OACs are symbolic models of control programs that are executable by an agent in the real world. This characterization

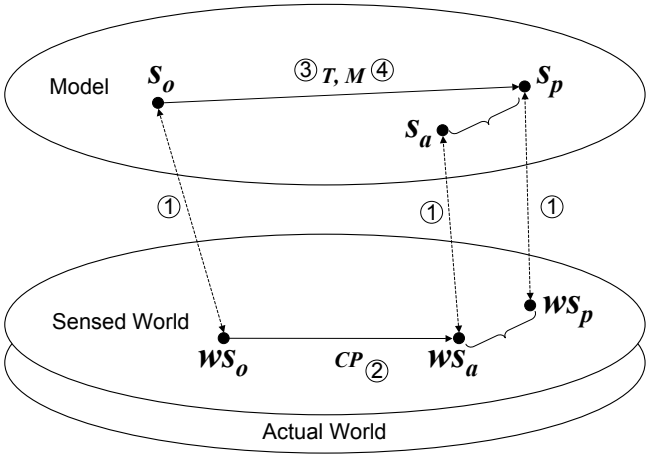

Figure 3: Graphical representation of the OAC learning problems

immediately gives rise to a number of learning questions that must be addressed for OACs to be effective. Figure 3 shows the OAC from Figure 2 and indicates portions of the model that are related to specific learning tasks. In particular, we consider four main types of learning:

1. **Translation: Learning the mapping from states of the real world to states of the model** (labeled 1) Learning the mapping from sensed world states to model states goes beyond simply learning the mapping between existing attributes. It also involves identifying those properties of the world that are key to effectively predicting state transitions and, when necessary, building new attributes that define the domain and range of an OAC's prediction function. We define the following procedure to perform this type of learning:

   Definition 4.1. *Let* `updateModel` *be a procedure that takes an experiment on a particular OAC*

   $$\texttt{updateModel} : \text{experiment} \rightarrow \textit{void}. \tag{4}$$

   `updateModel` should be interpreted as a procedure that updates an OAC's model of the world on the basis of an experiment: the experiment's outcome is inspected and a decision is made as to whether or

not the OAC's model needs to change. If so, the procedure modifies the model. Every OAC that addresses this learning problem should define this procedure. For instance, such learning might be used to create new attributes for high-level actions from more low-level sensorial (visual and haptic) information, e.g., the categories "open" and "closed" used as preconditions for certain grasping or filling actions.

2. **Assimilation: Learning the low-level control program** (labeled 2) This learning task modifies an OAC's control program to minimize the distance between the world state ws_p predicted by the OAC and the actual world state ws_a. We define the following procedure to perform this type of learning:

   Definition 4.2. *Let* `updateCP` *be a procedure that takes an experiment on a particular OAC and returns true or false,*

   $$\texttt{updateCP} : \text{experiment} \rightarrow \textit{void}. \tag{5}$$

   `updateCP` should be interpreted as a procedure that updates an OAC's control program on the basis of an experiment, to bring its resulting states in line with a given OAC. This function considers the experiment's outcome and modifies the OAC's control program appropriately. Every OAC that addresses this learning problem should define this procedure (see Sections 6.1 and 6.2). For example, suppose an agent knows it wants to throw a ball into a basket. If the OAC modelling the act of throwing a ball into a basket is known then the control program must be modified in order to ensure this effect can be repeatedly caused in the world.

3. **Accommodation: Learning the prediction function** (labeled 3) This learning task modifies the prediction function to minimize the distance between a predicted model state s_p, and the actual resulting model state s_a. We define the following procedure to perform this type of learning:

   Definition 4.3. *Let* `updateT` *be a procedure that takes an experiment on a particular OAC*

   $$\texttt{updateT} : \text{experiment} \rightarrow \textit{void}. \tag{6}$$

   `updateT` should be interpreted as a procedure that updates an OAC's prediction function on the basis of an experiment. It considers the

experiment's outcome and modifies the OAC's prediction function appropriately. Every OAC that addresses this learning problem should define this procedure (see, e.g., Section 6.3).

4. **Reliability measurement: Learning the prediction function's long term statistics** (labeled 4) This learning task updates the OAC's reliability measure M to reflect the long term success of the OAC. We define the following procedure to perform this type of learning:

   Definition 4.4. *Let* `updateM` *be a procedure that takes an experiment on a particular OAC*

   $$\texttt{updateM} : \text{experiment} \rightarrow \textit{void}. \tag{7}$$

   `updateM` should be interpreted as a procedure that updates an OAC's long term statistics on the basis of an experiment. This procedure considers the experiment's outcome and modifies the OAC's statistics appropriately. `updateM` is normally defined for every OAC.

We note that all of these learning problems can be addressed by recognizing the differences between predicted states and those states actually achieved, as captured by experiments. We will see further applications of these learning tasks in Section 6.

5. Execution

In Section 3 we defined an OAC as comprising two components: a *symbolic description* and an *execution specification*. In this section we define an OAC's execution specification by describing how OACs are anchored to specific control programs.

5.1. One-to-One OAC Execution

In our discussion up to this point, we have only considered single OACs modelling single control programs. This simplification makes execution relatively straightforward: the execution specification is the mapping of the OAC to the control program. Given such a mapping, the OAC's `execute` function can then invoke the specified control program and allow it to run until it terminates. The control program can then report the result of the experiment.

For instance, consider an autonomous system equipped with sensors and reflexive control programs for discovering objects in the world. In such a

system, an object may first be recognized through the repeatable and predictable action of a control program on the object. This "Birth of an Object" [12] can be extended to a "Birth of an OAC" [13, 14]: a grasping OAC is acquired for a particular object by incrementally extending grasping affordences associated to the object by playing with it (see Section 7.1). Thus, the grasping OAC forms a one-to-one execution relationship with the control program that performs the actual grasping in the world. We will see examples of this type of execution control in Section 6.1.

We can also consider higher-level OACs that stand in one-to-one correspondence with lower-level OACs. Rather than modelling control programs, these higher-level OACs model lower-level OACs defined on congruent attribute spaces. In terms of execution, we define the execution specification of a high-level OAC as simply calling the `execute` function of the corresponding lower-level OAC. We can also imagine more general "towers" of OACs where each OAC stands in one-to-one relation with an OAC (or a control program in the base case) that is beneath it in the tower. In such cases, the execution specification of each OAC is just the invocation of the next lower OAC in the tower. Thus, calling `execute` for the highest OAC results in a stack of calls to `execute`, one for each level of the tower, where each OAC invokes the OAC at the next level down until the process grounds out in the execution of a single motor program. The experiment that results from this execution is then returned (and appropriately translated for each attribute space) as the result of each call. Section 6.4 will discuss the use of high-level OACs for planning that stand in one-to-one correspondence to lower-level OACs (see Section 6.2) defined on a different attribute space.

5.2. One-to-Many Execution

The one-to-one mappings we previously discussed are not the only kind of relationship we can envision for OACs. We can also imagine more complex scenarios, where an OAC is mapped to a sequence of OACs or motor programs, or has an execution specification that uses iteration and conditional invocation of the kind found in dynamic logic [15]. For example, an OAC for opening a door might be comprised of a sequence of lower-level OACs modelling actions like: approach the door, grasp the door knob, twist the door knob, pull on the door knob, etc. In order to execute such a higher-level OAC, each of these actions must be successfully executed in the specified sequence. Furthermore, a formal definition of this kind of one-to-many execution specification requires ordering constraints and success criteria for each

of the sub-OACs.

This document will not provide a detailed example of such complex one-to-many OACs. We leave their learning and specification as an area for future work. However, we note that the correct understanding of the execution specification for such OACs must, like the one-to-one cases, rest on recursively calling the `execute` function and continually monitoring the congruency between the attribute spaces of the underlying OACs. It may also be necessary for an OAC to interrupt execution and replan its activities in order to restore congruency lost through error and non-determinism (see, e.g., [13]). It is the abstraction provided by `execute` and the congruency relation between attribute spaces that makes OACs a powerful reasoning tool in these situations.

6. Examples of OACs

In this section we give a number of concrete examples of OACs. These OACs will be situated within a three-level architecture, as illustrated in Figure 4 [16]. In this architecture, the lower sensorimotor level provides multisensory percepts and motor and sensing actions. The mid level stores the robot's sensorimotor experiences, makes them available to various learning processes, and serves as a link between raw sensorimotor and abstract symbolic processing. The high level is responsible for symbolic reasoning, such as planning. Each level defines its own set of OACs. We also assume there is an object memory component $\mathcal{M}^{\mathrm{O}}$ that stores object knowledge, as generated by the `update` functions and required by various `execute` functions.

The OACs discussed in the following sections include low-level actions for object-agnostic grasping (Section 6.1), mid-level actions for grasping an object based on previously-learned object models (Section 6.2), and high-level actions supporting planning (Section 6.4). To demonstrate that object-action associations beyond grasping can be formalized, we also give an example of object pushing (Section 6.3).

In each case we provide an informal description of the OAC, followed by a formal definition and an example of how the OAC can be embedded within a procedural structure to produce more complex behavioural patterns. In Section 7 we give examples of how grounding and planning can be realised by a set of interacting OACs.

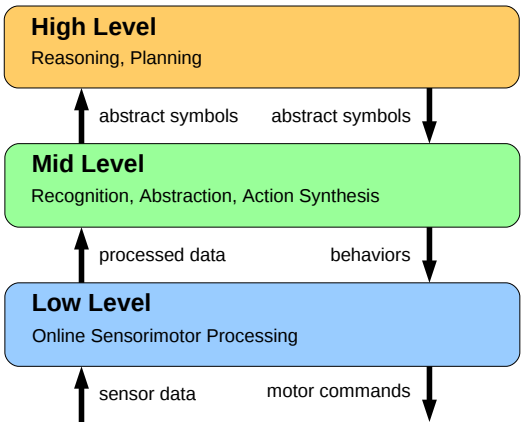

Figure 4: The three-level architecture supporting the example OACs in Section 6.

6.1. Grasping without Object Knowledge: $\texttt{oac}^{GenGrasp}$

6.1.1. Description

In the first example we consider an OAC $\texttt{oac}^{\texttt{GenGrasp}}$ ("`GenGrasp`" stands for "grasp generic") that associates grasping hypotheses to co-planar contour pairs (see Figure 5). This OAC can be applied to any visual structure containing (1) 3D contours and (2) a co-planarity relation.

$\texttt{oac}^{\texttt{GenGrasp}}$ is a low-level OAC constituting a visual feature/grasp association that can trigger a grasping action on an unknown "something" (see Figure 5b). Within the Early Cognitive Vision (ECV) system [17], whch provides ECV features in terms of local multi-modal symbolic visual descriptors, this OAC can be applied to scenes as well as learned visual object representations (see [12, 18] for details). It associates to any pair of co-planar contours $(C_i, C_j) \in \mathcal{C} \times \mathcal{C}$ (where $\mathcal{C}$ is the space of 3D contours) certain grasping hypotheses $G^{\mathrm{H}}(C_i, C_j)$ which can then be executed by the system.

6.1.2. Definition

The symbolic description of $\texttt{oac}^{\texttt{GenGrasp}}$ is formally defined by the triple

$$(\texttt{GenGrasp}, T, M)$$

where the relevant aspects of T are characterized by domain(T) and range(T). The domain of the predication function, domain(T), is defined by:

$$\{\Omega \neq \emptyset, \text{status}(\texttt{gripper}) = \texttt{empty}, \mathcal{C} \times \mathcal{C}\} \tag{8}$$

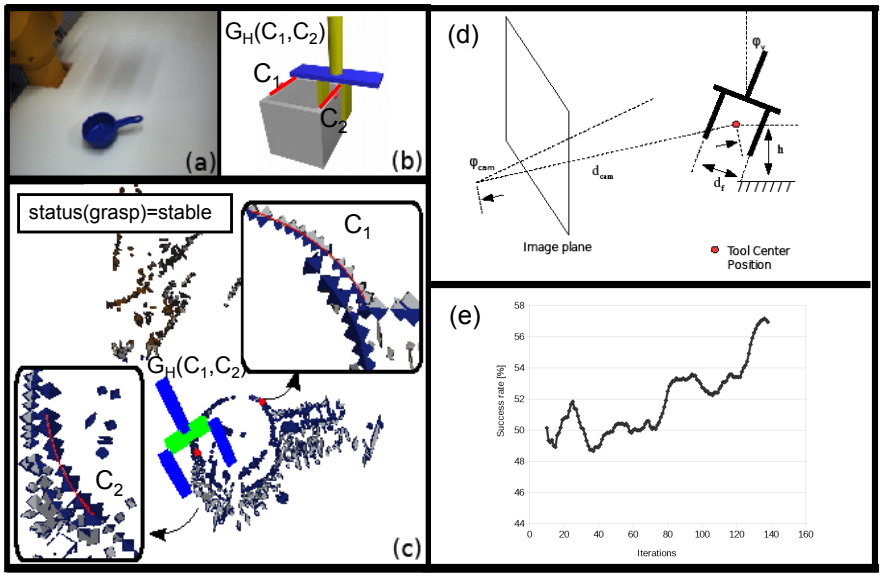

Figure 5: (a) The image of the scene captured by the left camera. (b) A possible grasping action type defined by using the two coplanar contours C_1 and C_2 shown in red. (c) A successful grasping hypothesis. The 3D contours from which the grasp was calculated are shown. Note that the information displayed is the core of an experiment "experiment". (d) Features used in learning process (e.g., distance from the camera, distance between fingers, etc.). (e) Change of performance as a result of the learning process.

which contains two preconditions and placeholders for two specifically chosen 3D contours. In particular, it requires that (1) there are co-planar contours C_i, C_j in the scene or object representation, i.e., the set of co-planar contours

$$\Omega = \{(C_i, C_j) \in \mathcal{C} \times \mathcal{C} \mid \mathrm{cop}(C_i, C_j) > s\}$$

is not empty, (2) the gripper is empty, and (3) a pair of contours C_i, C_j is concretely chosen with $\mathrm{cop}(C_i, C_j)$ being a coplanarity relation defined on two 3D contours C_i, C_j (see, e.g., [19] for details).

The range of the prediction function, $\mathrm{range}(T)$, is characterized by specific values of a state attribute status(`grasp`):

$$\mathrm{range}(T) = \left\{ \mathrm{status}(\texttt{grasp}) \in \left\{ \begin{array}{l} \texttt{noplan}, \texttt{collision}, \\ \texttt{void}, \texttt{unstable}, \texttt{stable} \end{array} \right\} \right\}. \qquad (9)$$

Before the execution of $\texttt{oac}^{\texttt{GenGrasp}}$, a large number of grasping hypotheses are computed, since in general there are many co-planar contours in a typical

scene. After selecting a specific grasping hypothesis, a motion planner tries to find a collision-free path that allows the arm to reach the pregrasping pose associated to the grasping hypothesis, which may result in a number of possible outcomes. If the planner fails to find a suitable trajectory or decides there is none, execution stops, and the result is `noplan`. If the hand unexpectedly enters into a collision, execution stops at that point, and the result is `collision`. If the closed gripper is determined to be empty, the result is `void`. If the gripper closes further while lifting the object, the result is `unstable`. Otherwise, the grasp is deemed successful, and the result is `stable`. Thus, the prediction function T for the OAC is simply the attribute status(`grasp`) = `stable`.

During execution, grasping hypotheses from co-planar contour pairs are computed.[1] Thus, the arguments of the OAC's `execute` function are given by

$$(|\Omega| > 0, \text{status}(\texttt{gripper}) = \texttt{empty}, (C_1, C_2)),$$

where $|\Omega|$ is the number of elements in the set Ω, and (C_1, C_2) is the concrete pair of extracted contours that was picked earlier.

The computed grasping hypothesis is then performed and the grasp status status(`grasp`)$_{t+1}$ is sensed after picking up the object, resulting in an experiment (see Figure 5c):

$$\text{experiment} = \{(1, 1, (C_1, C_2)), \texttt{GenGrasp}, \text{status}(\texttt{grasp})_{t+1} = \texttt{stable}, \text{status}(\texttt{grasp})_{t+1}\}.$$

Each experiment can either be used directly for on-line learning, as in the learning cycle in Section 6.1.3, or stored in an episodic memory for off-line learning at a later stage (see [20] for details).

Learning affects the execution of the control program through `updateCP`, and the updating of long-term statistics via `updateM` (see Figure 5e). The OAC's prediction function always remains constant. Learning is applied in the selecting the most promising grasping hypothes. The optimal choice of grasps depends on certain parameters (e.g., contour distance, object position in working space, see Figure 5d). Based on an RBF network (see [20] for details), a function that estimates the success likelihood for a certain grasp has been learned in a cycle of experimentation and learning (see Section 6.1.3).

[1] In practice, multiple hypotheses are computed from each co-planar pair of contours and one is chosen according to a ranking criterion (see [18, 20] for further details).

(In practice, we showed an increase in the success rate from 42% to 51% by such learning; see [20] for details).[2]

6.1.3. Simple exploration behaviour

Finally, $\texttt{oac}^{\texttt{GenGrasp}}$ can be applied multiple times to different contour pairs. Using this OAC, we can easily demonstrate explorative behaviour by the following loop which realises a simple learning cycle:

while *true* **do**
  choose pair of contours C_1, C_2
  experiment=`execute`(`GenGrasp`);
  `updateCP`(experiment);
  `updateM`(experiment);
  drop object
end

This loop also demonstrates how OACs can be embedded in procedural structures. We will see more examples of such structures in the following sections.

6.2. Grasping Based on Object Knowledge: $\texttt{oac}_o^{graspObj}$

6.2.1. Description

In this example we consider an OAC $\texttt{oac}_o^{\texttt{graspObj}}$ ("graspObj" stands for "grasp Object") which represents a specific object or class of objects o together with a set of associated grasp affordances specified with respect to the robot (see figure 7). Object models o_i are stored in object memory $\mathcal{M}^{\mathrm{O}}$. (See Section 7.1 for more information about learning such models.) An object model includes a learned, structural object model that represents geometric relations between 3D visual patches (ECV features [21, 22]) as Markov networks [23]. In addition, it contains a continuous representation of object-relative gripper positions that lead to successful grasps (grasp densities [24]). Object detection, pose estimation and the determination of useful gripper positions for grasping the object are all done simultaneously using probabilistic inference within the Markov network, given a scene reconstruction in terms of ECV features.

[2]Note that since $\texttt{oac}^{\texttt{GenGrasp}}$ uses very little prior knowledge, a high performance cannot be expected except in trivial scenarios.

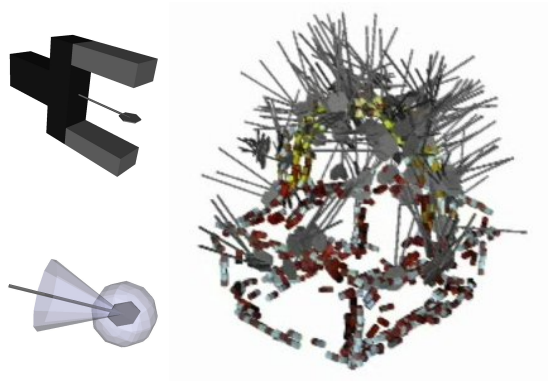

Figure 6: Visualization of grasp densities as used in Figure 7. A (continuous) grasp density is represented nonparametrically by particles. Each particle represents a 6D gripper pose (top left). The set of particles is interpreted as a continuous density via kernel density estimation, using a combination of a 3D isotropic Gaussian kernel for position and a toroidal isotropic Dimroth-Watson kernel for orientation (bottom left, showing unit-variance isosurfaces for both kernels). For visualization of grasp densities, gripper poses are represented as "spatulas" (top left) to reduce clutter (right).

6.2.2. Definition

The symbolic description of $\texttt{oac}_o^{\texttt{graspObj}}$ is formally defined by the triple

$$(\texttt{graspObj}, T, M)$$

where the relevant aspects of T are characterized by domain(T) and range(T). $\texttt{oac}_o^{\texttt{graspObj}}$ is potentially applicable whenever the gripper is empty and an instance of object o is present in the scene. Thus, domain(T) is defined as a set of assertions on the attribute space $\mathcal{S}$:

$$\text{domain}(T) = \{\text{status}(\texttt{gripper}) = \texttt{empty}, \text{targetObj} = o, o \in \mathcal{M}^{\text{O}}\}. \quad (10)$$

Here, the state description includes an attribute targetObj that specifies which object model o is to be applied by the `execute` function.

The `execute` function performs the following steps (Fig. 7):

1. Access or request a reconstruction of the current scene in terms of ECV features.
2. Retrieve the object model o from $\mathcal{M}^{\text{O}}$, and use it to locate the object and determine a gripper position.

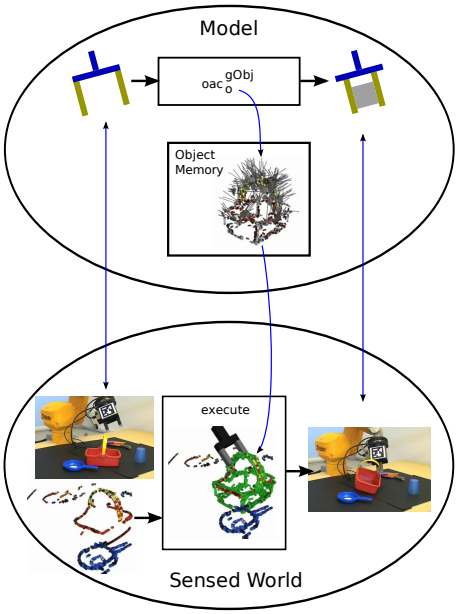

Figure 7: $\texttt{oac}_o^{\texttt{graspObj}}$ and its interaction with the environment (cf. Figure 2). The precondition (gripper empty) and predicted result (gripper holding an object) are mapped onto corresponding sensor states. The OAC refers to a specific object model (stored in the object memory $\mathcal{M}^{\mathrm{O}}$). The `execute` function instantiates this model within an ECV scene reconstruction, chooses a grasp according to the object-aligned grasp density (Figure 6), and triggers its execution.

3. Ask a path planner to generate a plan for maneuvering the gripper to the intended position.
4. If such a plan is found, execute the computed trajectory, and close the gripper to grasp the object.

This procedure yields a new state that is characterized by an attribute status(`grasp`) that can be assigned specific values similar to those in the state space of the OAC $\texttt{oac}^{\texttt{GenGrasp}}$:

$$\mathrm{range}(T) = \left\{ \mathrm{status}(\texttt{grasp}) \in \left\{ \begin{array}{l} \texttt{nopose}, \texttt{noplan}, \texttt{collision}, \\ \texttt{void}, \texttt{unstable}, \texttt{stable} \end{array} \right\} \right\}. \quad (11)$$

The only addition with respect to $\texttt{oac}^{\texttt{GenGrasp}}$ is the value `nopose`, which represents the case where no object instance can be reliably located.

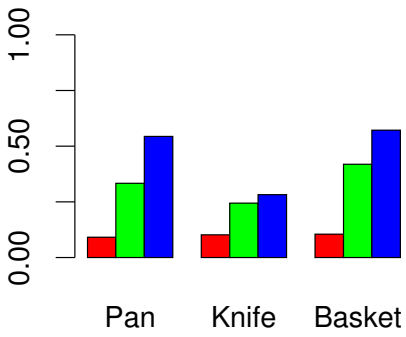

Figure 8: Evolving statistics of status(`grasp`) = `stable` for the OACs $\texttt{oac}^{\texttt{graspObj}}_{\text{Pan}}$, $\texttt{oac}^{\texttt{graspObj}}_{\text{Knife}}$, and $\texttt{oac}^{\texttt{graspObj}}_{\text{Basket}}$ over cumulative rounds of grasping trials [25].

The `execute` function is defined in such a way as to return an experiment

$$\text{experiment} = (s_o, \texttt{gObj}, s_p, s_a),$$

where s_p typically contains status(`grasp`) = `stable`, and in s_a, status(`grasp`) takes one of the values listed in Eqn. (11). In addition, the data structures representing s_o, s_p and s_a include further state information such as the object model o as well as object and gripper poses. Such information is used, in particular, by `updateCP` to update the grasp density by integrating new experiments, which lead to increasingly reliable performance (Figure 9).

Objects are always located within the currently-sensed part of the scene. Thus, it is up to other parts of the system to make sure that the scene reconstruction available to `execute` contains one and only one instance of the object o, e.g., by directing sensors accordingly.

As in the previous example, the prediction function T always returns status(`grasp`) = `stable`. M is defined in such a way as to maintain cumulative outcome statistics of executions of this OAC, updated via `updateM` (see Figure 8).

6.2.3. Usage Example

The following procedure outlines how a higher-level process might acquire and refine grasping skills on a variety of objects. In this scenario, the scene contains up to one instance of each object of interest. The robot "plays" with the object by repeatingly grasping and dropping the object. This leads to a learning cycle similar to the one in Section 6.1.3, in which the system

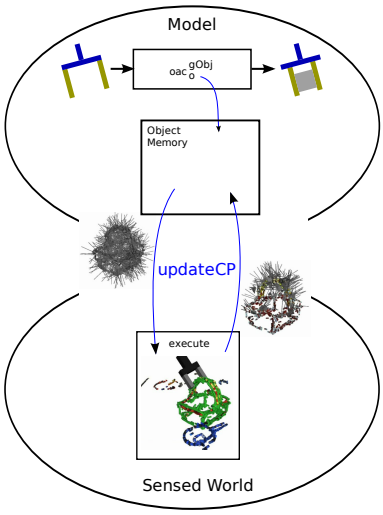

Figure 9: The principal learning capability of $\texttt{oac}_o^{\texttt{graspObj}}$ (cf. Figure 3). A grasp density used for execution can be updated by incorporating new experiments.

generates knowledge about the grasp affordances associated to the object:

while *true* **do**
  experiment = `execute`(`gObj`);
  `updateCP`(experiment);
  `updateM`(experiment);
  drop object
end

6.3. Acquiring Pushing Behaviours Based on Simple Motor Primitives: $\texttt{oac}^{GenPush}$

6.3.1. Description

In this example we define an OAC $\texttt{oac}^{\texttt{GenPush}}$ which encodes how to push objects in different directions on a planar surface without grasping. Pushing as a nonprehensile action cannot be learned with sufficient accuracy to ensure that a given object moves to the desired target in one step, i.e., by applying one pushing movement. If a planner specifies that an object o should be pushed to a certain target, $\texttt{oac}^{\texttt{GenPush}}$ needs to be applied iteratively in a feedback loop until the target location is eventually reached. To achieve this, the system needs to know how objects move when short pushing actions are applied (such actions are also called poking). To apply such actions, the

object to be pushed needs to be localisable within the workspace of the robot. Besides the object location, the resulting motion depends on properties such as shape, mass distribution and friction. (We will focus here on shape.)

Some prior motor knowledge needs to be available before this OAC can be learned. In particular, we assume that the robot knows how to move the pusher, e.g., the robot hand or a tool held in its hand, along a straight line in Cartesian space. The central issue for learning $\texttt{oac}^{\texttt{GenPush}}$ is to acquire a prediction function that can estimate the object movement in response to the pusher movement. The resulting control policy encoded by $\texttt{oac}^{\texttt{GenPush}}$ is neither object nor target dependent. A detailed description of technical aspects of an earlier realization of the pushing OAC can be found in [26].

6.3.2. Definition

The symbolic description of $\texttt{oac}^{\texttt{GenPush}}$ is formally defined by the triple

$$(\texttt{GenPush}, T, M).$$

The prediction function T associated with $\texttt{oac}^{\texttt{GenPush}}$ should say how an object moves in response to the applied pushing movement. To this end, the system must have at its disposal information about the object's shape, its current location on the planar surface, the duration of the pushing movement, and its direction. We represent the shape by 2D binarized object images, such as those shown in Figure 10. Such images are sufficient as shape models (as opposed to full 3D shape models) because this OAC only encodes the pushing behavior for objects that do not roll on planar surfaces. We can then predict the next object location using the transformation

$$T(\text{bin}(o), \text{loc}(o), \tau, a) = T'(\text{bin}(o), \text{loc}(o), a)\tau + \text{loc}(o). \tag{12}$$

Here, $\text{loc}(o)$ denotes the location of the object o before the application of the pushing movement, $\text{bin}(o)$ is the shape model in the form of a binary image of the object to be pushed, a denotes the parameters describing the direction of the movement of the pusher (as realized by the control policy), τ is the duration of the push, and T' is the function predicting the outcome of the push in terms of the object's linear and angular velocity. The prediction function T is thus defined as

$$T : \{\text{bin}(o), \text{loc}(o), \tau, a\} \longrightarrow \{loc(o)\}, \tag{13}$$

where $\text{domain}(T) = \{\text{bin}(o), \text{loc}(o), \tau, a\}$ and $\text{range}(T) = \{\text{loc}(o)\}$.

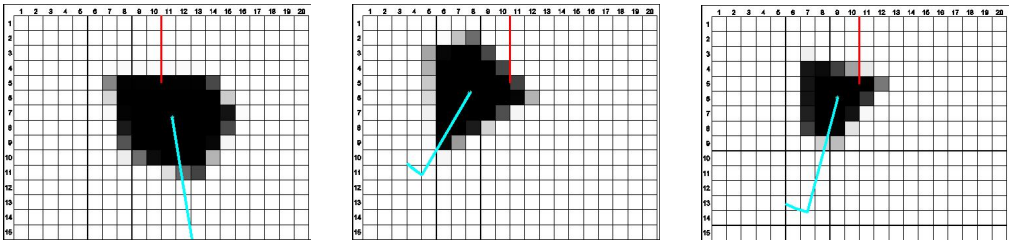

Figure 10: Samples of low resolution object images used as input to the neural network.

Figure 11: Pushing behaviour realized by $\texttt{oac}^{\texttt{GenPush}}$ after learning prediction function T.

The prediction function T returns the expected position and orientation of the object after being pushed at a given point and angle on the boundary with constant velocity for a certain amount of time. The angle of push is defined with respect to the boundary tangent. These two parameters are fully determined by the object's binary image and the pusher's Cartesian motion, which must therefore be included in domain(T).

An impulse to push an object in a certain direction must be provided by a higher level cognitive process. The appropriate robot control policy can be determined based on the available prediction function T. Two possibilities will be discussed in Section 6.3.3. The execution process `execute` works in the following steps: 1) extract the binary image of the object $\text{bin}(o)$ and its location $\text{loc}(o)$, 2) acquire the pushing movement parameters a, 3) predict the outcome of the pushing action by calculating $T(\text{bin}(o), \text{loc}(o), a, \tau)$, 4) execute the pushing movement by calling the pushing movement primitive initialized by (a, τ), and 5) localise the object after the push. The result of a call to the `execute` function is therefore an experiment of the form

$$\text{experiment} \;=\; ((\text{loc}(o), \text{bin}(o));\ \texttt{push};\ T(\text{bin}(o), \text{loc}(o), a, \tau);\ \text{loc}_a(o)).$$

Here $\text{loc}_a(o)$ is the location of the object after the push. When the task is to push an object towards a given target location, the robot can solve it by

successively applying `execute` in a feedback loop until the goal is reached. Note that lower-level motor primitives that realize straight-line motion of the pusher in Cartesian space are constant and do not need to change while learning $\texttt{oac}^{\texttt{GenPush}}$.

The statistical evaluation M measures how close the predicted object movement is to the real object movement. Here and in what follows we use $\mathrm{loc}(o) = (\mathbf{u}, \theta)$, $\mathrm{loc}_a(o) = (\mathbf{u}_a, \theta_a)$, $\mathrm{loc}_p(o) = (\mathbf{u}_p, \theta_p) = T(\mathrm{bin}(o), \mathrm{loc}(o), a, \tau)$ to respectively denote the current object position and orientation, the position and orientation after the push, and the predicted object position and orientation. We define the following metrics to measure the difference between the expected and actual object movement on the planar surface

$$\mathrm{d}(\mathrm{loc}_a(o), \mathrm{loc}_p(o)) \;\; = \;\; w_1 \left\| \mathbf{u}_a - \mathbf{u}_p \right\| + w_2 \left| \theta_a - \theta_p \right|, \tag{14}$$

where $w_1, w_2 > 0$. The expectation of the $\texttt{oac}^{\texttt{GenPush}}$ performance after N experiments is thus given by

$$M = \frac{1}{N} \sum_{i=1}^{N} \mathrm{d}(\mathrm{loc}_a(o)_i, \mathrm{loc}_p(o)_i). \tag{15}$$

The learning in $\texttt{oac}^{\texttt{GenPush}}$ affects the prediction function through `updateT`, and the long-term statistics via `updateM`. This learning is realized using a feedforward neural network with backpropagation. This network represents a forward model for object movements that have been recorded with each pushing action. To ensure that $\texttt{oac}^{\texttt{GenPush}}$ can be applied to different objects, the shape parameters specified in the form of a low resolution binary image are used as input to the neural network. Function T is updated incrementally based on the observed object movements. Statistical evaluation is also done incrementally as experiments are performed. Note, however, that since the prediction function T changes during learning, the statistical evaluation `updateM` only converges to the true accuracy of the behaviour once T becomes stable (see Figure 12).

6.3.3. Incremental learning by exploration

There are two modes of operation in which we consider $\texttt{oac}^{\texttt{GenPush}}$:

A. Initial learning of the prediction function T, where the pushing directions encoded by a are randomly selected, and

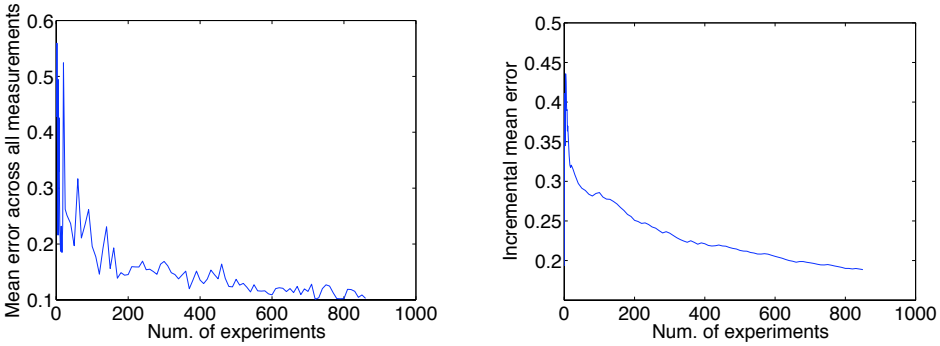

Figure 12: Mean error of robot pushing. The left figure shows the mean error calculated using Eqn. (14) and all measurements. The right figure shows the incremental statistical evaluation as realized by `updateM`. Four different objects were used in the experiment.

B. Pushing the object towards a given target, where the current pusher movement a is determined based on the previously learned prediction function and the given target location.

As described above, the prediction function T is essentially encoded by a neural network with the binary image of an object and the direction of the pusher movement a used as input values, and the predicted final position and orientation of the pushed object as output. In mode B, we calculate the optimal pusher movement a by first determining the desired Cartesian movement of o from its current location towards the target location and then inverting the neural network using nonlinear optimisation. The resulting behaviour is presented in Figure 11.

The learning process has been implemented using the following exploration behavior:

while *true* **do**
  $a =$ `SelectRandomMotion`; $\mathrm{bin}(o)$; $\mathrm{loc}(o)$;
  experiment = `execute`(`push`);
  if $\mathrm{d}(\mathrm{loc}(o), \mathrm{loc}_a(o)) > \epsilon$ **then**
    `updateM`(experiment);
    `updateT`(experiment);
  end
end

where constant $\epsilon > 0$ is used to determine whether the object has moved or not. In this context, `updateT` estimates the weights of the neural network. Note that `updateM` is always applied to data before it has been used for

Properties	
`clear(X)`	A predicate indicating no object is stacked in `X`.
`focusOfAttn(X)`	A predicate indicating that object `X` is the agent's focus of attention.
`gripperEmpty`	A predicate describing whether the robot's gripper is empty or not.
`inGripper(X)`	A predicate indicating that the robot is holding object `X` in its gripper.
`inStack(X,Y)`	A predicate indicating that object `X` is in a stack with object `Y` at its base.
`isIn(X,Y)`	A predicate indicating that object `X` is stacked in object `Y`.
`onShelf(X)`	A predicate indicating that object `X` is on the shelf.
`onTable(X)`	A predicate indicating that object `X` is on the table.
`open(X)`	A predicate indicating that object `X` is open.
`pushable(X)`	A predicate indicating that object `X` is pushable by the robot.
`radius(X) = Y`	A function indicating that the radius of object `X` is `Y`.
`reachable(X)`	A predicate indicating that object `X` is reachable for grasping by the gripper.
`shelfSpace = X`	A function indicating that there are `X` empty shelf spaces.

Table 1: Attribute space for planning-level OACs

learning.

6.4. Planning with OACs

6.4.1. Description

We now turn our attention to high-level OACs usable for planning, and consider an OAC $\texttt{oac}^{\texttt{graspObjPlan}}$ that models a grasping action [27]. At an abstract level, $\texttt{oac}^{\texttt{graspObjPlan}}$ can be thought of as an action that attempts to pick up an object from a table. This OAC operates on a discrete attribute space defined in terms of a set of logical predicate and function symbols that denote certain properties of the world. Such representations are standard in AI planning systems and we will structure our OAC so that we can it for building and executing plans.

6.4.2. Definition

The symbolic descrition of $\texttt{oac}^{\texttt{graspObjPlan}}$ is formally defined by the triple

$$(\texttt{graspObjPlan}, T, M).$$

Table 1 shows the attribute space $\mathcal{S}$ for our OAC, defined as a set of logical symbols. Given this attribute space, we can define the prediction function T

Name	Initial Conditions	Prediction
$\texttt{oac}^{\texttt{graspObjPlan}}$	`focusOfAttn(X)`	`inGripper(X)`
	`reachable(X)`	`not(gripperEmpty)`
	`clear(X)`	`not(onTable(X))`
	`gripperEmpty`	
	`onTable(X)`	
$\texttt{oac}^{\texttt{pushObjPlan}}$	`focusOfAttn(X)`	`reachable(X)`
	`not(reachable(X))`	
	`pushable(X)`	
	`clear(X)`	
	`gripperEmpty`	
	`onTable(X)`	

Table 2: Prediction function T for planning-level grasping and pushing OACs.

as the STRIPS-style rule [7, 28] given at the top of Table 2. Such rules require a description of the initial conditions that must hold for the action to be applied, and the predicted conditions that result from performing the action. In this case, both the initial conditions and the predictions are assumed to be conjunctions of specific attributes, i.e., all of the initial conditions must be true in the world for the prediction function to be defined, and all of the predictions are expected to be true in any state that results from the execution of the OAC. In terms of $\texttt{oac}^{\texttt{graspObjPlan}}$, this means that if an object is the focus of attention, is on the table, is clear, is reachable, and the agent's gripper is empty, then after executing this OAC we predict the object will be in the gripper and not on the table, and the gripper will no longer be empty. In any other case, the prediction function is undefined.

We must also provide a statistical measure M of the reliability of T. Taking the simplest possible approach, we define M as the long term probability of T correctly predicting the resulting state, assuming the OAC's execution began from a state for which the OAC was defined. We note that in classical AI planning systems, the reliability measure for all OACs would be fixed at 1. Such planners assume a deterministic and totally observable world, thereby removing all uncertainty from their prediction functions.

More recent work in AI planning has moved beyond these assumptions (see, e.g., [29]). For instance, there are now a number of planning algorithms that use probabilistic statements about an action's long term success to build plans with probabilistic bounds on the likelihood of achieving their goals. Our definition of M makes our OACs suitable for use by such planners.

In related work, we have also focused on the problem of implementing

an updateT function for learning such representations. To do so we use a training set of example actions in the world, and corresponding observations of the world before and after each action. For each example, a reduced world state consisting of a subset of the propositional features that make up the entire state is computed and considered by the learning model. The reduced state is provided as input to the learning model in the form of a vector where each bit corresponds to the value of a single feature in the world. The learning problem is then treated as a set of binary classification problems, with one classifier for each feature, and the model learning the changes to each feature in the reduced state. Our particular approach uses a kernelised voted perceptron classifier [30, 31], which is computationally efficient and can handle noise and partial observability. We refer the reader to [32] for a detailed account of how this kind of OAC (both the symbolic prediction function and the associated reliability measure M) can be learned.

The execution specification for $\texttt{oac}^{\texttt{graspObjPlan}}$ is straightforward. Given the previous examples in this section, we simply define the execution of $\texttt{oac}^{\texttt{graspObjPlan}}$ in terms of the execution of $\texttt{oac}^{\texttt{graspObj}}$. In other words, invoking the execution function $\texttt{execute}(\texttt{oac}^{\texttt{graspObjPlan}})$ invokes $\texttt{execute}(\texttt{oac}^{\texttt{graspObj}})$.

Table 2 also shows an example of a second planning level OAC, $\texttt{oac}^{\texttt{pushObjPlan}}$. In this case, $\texttt{oac}^{\texttt{pushObjPlan}}$ models an action that pushes an object into a position so that it can be grasped using $\texttt{oac}^{\texttt{graspObjPlan}}$. $\texttt{oac}^{\texttt{pushObjPlan}}$ operates over the same attribute space $\mathcal{S}$ as $\texttt{oac}^{\texttt{graspObjPlan}}$, and is defined in a similar way. The OAC's execution specification is also defined in a likewise manner: $\texttt{execute}(\texttt{oac}^{\texttt{pushObjPlan}})$ is defined as $\texttt{execute}(\texttt{oac}^{\texttt{GenPush}})$. In the next section we will use these two planning level OACs together in a single architecture.

7. Interacting OACs

In this section, we describe two examples of OACs interacting in a single architecture. The first example, described in Section 7.1, addresses the grounding of objects and object-related grasp affordances. The second example, in Section 7.2, describes how such grounded representations can be used to execute plans.

7.1. Grounding Grasping OACs

The grounding of objects and object-related grasping affordances is based on two learning cycles involving the OACs $\texttt{oac}^{\texttt{GenGrasp}}$ and $\texttt{oac}_o^{\texttt{graspObj}}$ (see

Figure 13). This process has been previously described in [14], however, we give a brief description of it here in a procedural OAC notation:

First learning cycle
while status(`grasp`) $\neq$ `stable` **do**
  experiment = `execute`(`GenGrasp`);
  `updateCP`(experiment);
  `updateM`(experiment);
  open gripper
end
Accumulate object representation o_i
if *accumulation successful* **then**
  transfer o_i into object memory $\mathcal{M}^{\mathrm{O}}$
  initialise $\mathtt{oac}_{o_i}^{\mathtt{graspObj}}$ in $\mathcal{M}^{\mathrm{OAC}}$
  Second learning cycle
  while *instance of object* o_i *in scene* **do**
    state.targetObj = o_i
    experiment = `execute`($\mathtt{graspObj}_{o_i}$);
    `updateCP`(experiment);
    `updateM`(experiment);
    open gripper
  end
end

In this process, object knowledge and grasp knowledge is built up and stored as part of the internal representation (i.e., the object and grasp memory). Furthermore, certain characteristics of our OACs play an important role in this process:

- Although the purpose of the first learning cycle is not to learn the OAC $\mathtt{oac}^{\mathtt{GenGrasp}}$ (the aim is to attain physical control over an object), learning is nevertheless taking place by calls to the `updateCP`(experiment) and `updateM`(experiment) functions, as a process parallel to those processes steered by, e.g., intentions or automised behaviours.

- OACs can be chained to create complex behaviours that are not necessarily driven by planning. For instance, innate processes such as those used for tasks like bootstrapping a system can also modelled by interacting OACs.

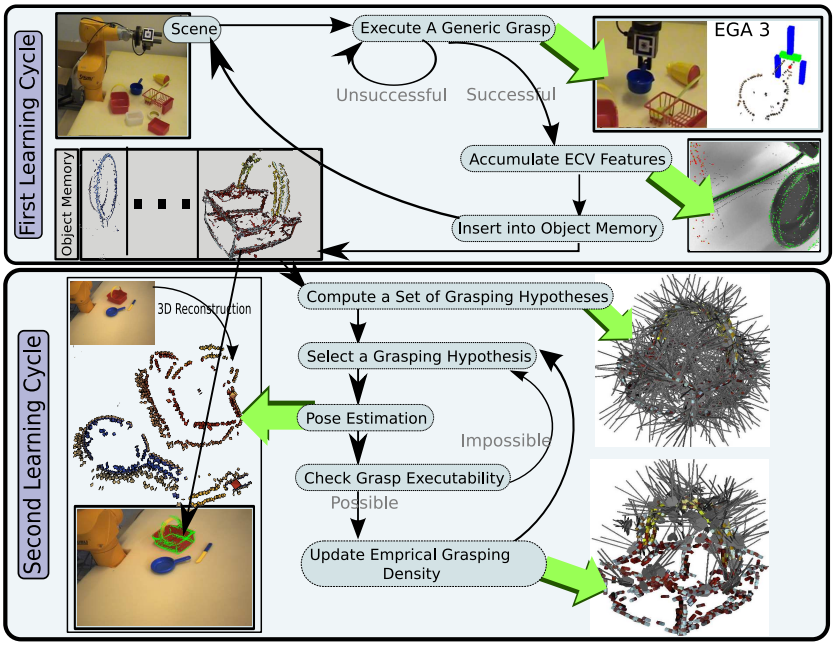

Figure 13: Grounding the OAC $\texttt{oac}_o^{\texttt{graspObj}}$ in two learning cycles. Within the first learning cycle, physical control over a potential object is obtained by the use of $\texttt{oac}^{\texttt{GenGrasp}}$. Once control over the object is achieved and the visual structure changes according to the movement of the robot arm, a 3D object representation is extracted and stored in the memory. In the second learning cycle $\texttt{oac}_o^{\texttt{graspObj}}$ is established and refined. First, the object representation extracted in the first learning cycle is used to determine the pose of the object in case it is present in the scene. Random samples of these are then tested individually. Successful grasps are turned into a probability density function that represents the grasp affordances associated to the object, in the form of the success likelihood of the grasp parameters.

- The interaction of multiple OACs, as demonstrated in the two learning cycles, can result in the grounding of symbolic entities usable for planning (see Section 7.2).

7.2. Performing Plans

We now demonstrate how higher-level OACs can be executed by calling lower-level OACs, in the context of performing a plan. To do this, we consider an agent that is given the high-level goal of achieving $\texttt{inGripper}(o)$ in a world initially described by the predicate set:

$$\{\texttt{focusOfAttn}(o), \texttt{gripperEmpty}, \neg\texttt{reachable}(o), \\ \texttt{pushable}(o), \texttt{onTable}(o), \texttt{clear}(o)\}.$$

Given the fact that o is not reachable in the initial state, a high-level planner might build a plan that first involves pushing o in order to make it `reachable`, followed by an action that picks o up. This results in the following plan consisting of two high-level OACs:[3]

$\texttt{oac}^{\texttt{pushObjPlan}}$
$\texttt{oac}^{\texttt{graspObjPlan}}$.

Recall from Section 6.4 that the execution of our higher-level OACs rests on the execution of lower-level OACs, with one OAC effectively calling another OAC as a subroutine, i.e.,

$$\begin{array}{lcl} \texttt{execute}(\texttt{oac}^{\texttt{graspObjPlan}}) & \longrightarrow & \texttt{execute}(\texttt{oac}^{\texttt{graspObj}}), \\ \texttt{execute}(\texttt{oac}^{\texttt{pushObjPlan}}) & \longrightarrow & \texttt{execute}(\texttt{oac}^{\texttt{GenPush}}). \end{array}$$

To understand the execution of the above plan, we must consider the ordering relation of the respective execution calls and returns of the component OACs in the plan. If we assume that the world and the agent act as predicted and planned, without plan or execution failures, then the following is an example of what a hypothetical "call stack" of an OAC-based agent would look like when executing this plan:[4]

$\text{experiment}_{topLev}$=`execute(`$\texttt{oac}^{\texttt{pushObjPlan}}$`)`
    $\text{experiment}_{midLev}$=`execute(`$\texttt{oac}^{\texttt{GenPush}}$`)`
    `updateT`($\text{experiment}_{midLev}$)
    `updateM`($\text{experiment}_{midLev}$)
`updateT`($\text{experiment}_{topLev}$)
`updateM`($\text{experiment}_{topLev}$)
$\text{experiment}_{topLev}$=`execute(`$\texttt{oac}^{\texttt{graspObjPlan}}$`)`
    $\text{experiment}_{midLev}$=`execute(`$\texttt{oac}_{o}^{\texttt{graspObj}}$`)`
    `updateCP`($\text{experiment}_{midLev}$)
    `updateM`($\text{experiment}_{midLev}$)
`updateT`($\text{experiment}_{topLev}$)
`updateM`($\text{experiment}_{topLev}$)

This simple procedure hides many of the processes actually required for

[3] We refer the reader to [27, 29] for more details on how such a plan is built.

[4] We refer the reader to [28] for an initial discussion of plan execution in the face of plan failure, which is beyond the scope of this paper.

plan execution. For instance, many additional steps are performed during the execution of the above plan:

1. The execution of $\texttt{oac}^{\texttt{pushObjPlan}}$ is defined in terms of the execution of $\texttt{oac}^{\texttt{GenPush}}$. Thus, information must be translated from the high-level representation into $\texttt{oac}^{\texttt{GenPush}}$'s model. Based on $\texttt{focusOfAttn}(o)$, a process must be invoked to acquire $\text{bin}(o)$ and extract $\text{loc}(o)$ from the environment. Second, a process must identify τ and a for the desired push operation.
2. As we described in Section 6.3, executing $\texttt{oac}^{\texttt{GenPush}}$ invokes a low-level control program that performs the actual pushing of o, making use of the agent's end effector.
3. Executing $\texttt{oac}^{\texttt{GenPush}}$ returns the experiment $\text{experiment}_{midLev}$:

   $$(\{(\text{loc}(o), \text{bin}(o))\},\ \texttt{push},\ \{T(\text{bin}(o), \text{loc}(o), a, \tau)\}, \{\text{loc}(o)'\})$$

   (see Section 6.3). $\text{loc}(o)'$ can then be translated to determine the truth value of the high-level predicate $\texttt{reachable}(o)$ which is used in the experiment returned by $\texttt{oac}^{\texttt{pushObjPlan}}$. In addition, $\texttt{updateT}(\text{experiment}_{midLev})$ and $\texttt{updateM}(\text{experiment}_{midLev})$ use $\text{experiment}_{midLev}$ to update the prediction function and long-term statistics M of $\texttt{oac}^{\texttt{GenPush}}$.
4. Executing $\texttt{oac}^{\texttt{pushObjPlan}}$ returns the experiment $\text{experiment}_{topLev}$:

   $$\begin{array}{l}(\ \{\neg\texttt{reachable}(o), \texttt{pushable}(o), \texttt{clear}(o), \texttt{gripperEmpty}, \texttt{onTable}(o)\}, \\ \quad \texttt{oac}^{\texttt{pushObjPlan}}, \\ \quad \{\texttt{reachable}(o)\}, \\ \quad \{\texttt{reachable}(o)\}? \ )\end{array}$$

   indicating that $\texttt{reachable}(o)$ is now true in the actual world, and the agent can update its model with this information. Additional learning is performed by the procedures $\texttt{updateT}(\text{experiment}_{topLev})$ and $\texttt{updateM}(\text{experiment}_{topLev})$.
5. A plan execution monitor of the kind described in [27] can now verify at the high level that pushing the object has in fact resulted in a state where it can now be grasped, i.e., $\texttt{reachable}(o)$ is now true. This is indicated by '?' in the expression '$\{\texttt{reachable}(o)\}$?'.
6. The execution of $\texttt{oac}^{\texttt{graspObjPlan}}$ is defined in terms of the execution of $\texttt{oac}_o^{\texttt{graspObj}}$. However, as with $\texttt{oac}^{\texttt{pushObjPlan}}$, information must be translated from the high-level representation into $\texttt{oac}^{\texttt{GenPush}}$'s model. Since

$\texttt{focusOfAttn}(o)$ is true in the world, the translation process ensures that targetObj $= o$.

7. As we described in Section 6.2, executing $\texttt{oac}_o^{\texttt{graspObj}}$ invokes a low-level control program that performs the actual grasping of o, making use of the agent's end effector.
8. Executing $\texttt{oac}_o^{\texttt{graspObj}}$ returns the experiment $\text{experiment}_{midLev}$:

$$\begin{array}{ll} (& \{\text{status}(\texttt{gripper}) = \texttt{empty}, \text{targetObj} = o\}, \\ & \texttt{gObj}, \\ & \{\text{status}(\texttt{grasp}) = \texttt{stable}\}, \\ & \{\text{status}(\texttt{grasp}) = \texttt{stable}\}? \ ) \end{array}$$

(see Section 6.2). status(grasp) = stable can then be translated to determine the truth value of the high-level predicate $\texttt{inGripper}(o)$, which is used in the experiment returned by $\texttt{oac}^{\texttt{graspObjPlan}}$. In addition, learning based on $\text{experiment}_{midLev}$ is performed for $\texttt{oac}_o^{\texttt{graspObj}}$: updateM revises the long-term statistics M, and updateCP updates the control program associated with $\texttt{oac}_o^{\texttt{graspObj}}$.

9. Executing $\texttt{oac}^{\texttt{graspObjPlan}}$ returns the experiment $\text{experiment}_{topLev}$:

$$\begin{array}{ll} (& \{\texttt{reachable}(o), \texttt{clear}(o), \texttt{gripperEmpty}, \texttt{onTable}(o)\}, \\ & \texttt{oac}^{\texttt{graspObjPlan}}, \\ & \{\texttt{inGripper}(o), \neg\texttt{gripperEmpty}, \neg\texttt{onTable}(o)\}, \\ & \{\texttt{inGripper}(o), \neg\texttt{gripperEmpty}, \neg\texttt{onTable}(o)\}? \ ) \end{array}$$

indicating that $\texttt{inGripper}(o)$ is now true in the world and, as before, the agent can update its high-level model to reflect this fact. Again, learning based on $\text{experiment}_{topLev}$ takes place at the high level.

10. The plan execution monitor can now verify that $\texttt{inGripper}(o)$ is now true, and end plan execution.

Thus, as illustrated in the above example, the successful execution of a plan may typically require invoking OACs at multiple levels of abstraction, translating the calls between different models, and monitoring the results to confirm the success of the actions involved.

8. Relation to Other Approaches

In this section we briefly discuss the relationship between OACs and other existing representations in the literature. In particular, OACs combine several known and novel concepts into one conjoint formalism.

Attributes and Expected Change: The representation of world states in terms of discrete attribute spaces, and the representation of actions as expected changes to the values of these attributes, can be directly linked to STRIPS [7] and other classical formalisms, including [33, 34, 35]. However, OACs go beyond such classical representations in permitting both continuous and discrete attribute spaces, making it possible to use OACs at different levels of a processing hierarchy: from low-level sensory-motor processes for robot perception and control, to high-level symbolic units for planning and language. Thus, OACs can be viewed as containers enabling sub-symbolic as well as symbolic representations, and models of both symbolic cognition and emergent cognition can be formalized using OACs (see [36]). For example, the Birth of the Object process [12, 37]—whereby a rich object description and a representation of grasping affordances emerges through interaction with the world—can be understood as the concatenation of several low-level perception-action interactions that are formulated in terms of OACs (see section 7.1), leading to processes in which symbolic entities emerge on the planning level.

Grounding and Situatedness: OACs reflect a growing consensus concerning the importance of grounding behavior in sensory-motor experience. Such grounding has been stressed in the context of embodied cognition research (see, e.g., [38, 39, 40, 41]). To build a truly cognitive system, it is necessary to have the system's representations grounded by interacting with the physical world in a closed perception-action loop [40]. OACs are necessarily grounded by their execution functions (Section 5), and are learned from the sensory-motor experiences of the robot (Section 4).

Modularity: The principle of modularity is widespread in cognitive process modelling (e.g., vision [42] and motor control [43, 44, 45]). As we demonstrated in Section 6, this concept is also inherent in the structure of OACs: OACs often operate at increasing levels of abstraction, each with a particular representation of situations and contexts. For instance, consider our three examples of OACs for grasping objects. On the lowest level, continuous grasp affordance densities code individual end-effectors poses for grasping completely unknown objects. At the mid level, these affordance densities are used to hypothesize possible grasps when the agent has some object knowledge. Finally, the highest level plans effective grasps to move objects.

Learning is also modularised through the OAC concept, and in our example OACs: the lowest level learns the difference between successful and

unsuccessful grasps, the mid level learns alternative object-specific ways of posing the hand, and the highest level learns the abstract preconditions and effects of grasping. Maintaining representational congruency between the attribute spaces of the different OACs allows systems to benefit from the modularity of the information learned for each OAC.

Predictivity: Predictability of cause and effect (or the lack of it) is important for cognitive agents and has been treated in a large body of work [46, 47, 48, 49, 50, 6]. OACs go beyond existing action representations by describing a common predictive formalism for cognitive processes, usable at multiple levels of abstraction. The prediction function itself can be seen as a dynamic entity, changing under the influence of ongoing learning processes in the cognitive system.

Learning, Evaluation, and Memorization: Cognitive agents must learn from past experiences in order to improve their own development, a task that typically requires a form of memory as a means of tracking prior interactions. While memory itself is not often a problem, such processes must ensure efficient representation, with properties like associative completion and content addressability, to enable machine learning from stored instances presented over a period of time.

We have seen numerous examples of OAC learning throughout this paper. Since our OAC definition allows various types of learning algorithms to be applied, individual OACs can tailor such learning to their specific needs. Most notably, OACs can learn their prediction functions, an idea which is closely related to statistical structure learning as discussed in [51, 52, 53, 54, 55, 32, 48].

OACs can also learn how successful their executions are over particular time windows. In particular in early development, when actions are likely to be unsuccessful, it is important to ensure that such execution uncertainties can be reasoned about. The storage of statistical data concerning execution reliability also has important applications to probabilistic planning [56], where an OAC's probability of success can be utilized to compute optimal plans. Consistently successful plans can then be memorized for future reference.

9. Conclusion

OACs are a dynamic, learnable, refinable, and grounded representation that binds objects, actions, and attributes in a causal model. OACs have

the ability to carry low-level (sensory-motor) as well as high-level (symbolic) information and can therefore be used to join the perception-action space of an agent with its planning-reasoning space. In addition, OACs can be combined to produce more complex behaviours, and sequenced as part of a plan generation process. As a consequence, the OAC concept can be used to bridge the gap between low-level sensory-motor representations, required for robot perception and control, and high-level representations supporting abstract reasoning and planning.

10. Acknowledgments

This work was supported by the EU Cognitive Systems project PACO-PLUS (IST-FP6-IP-027657).

References

[1] R. A. Brooks, A robust layered control system for a mobile robot, IEEE Journal of Robotics and Automation 2 (1986) 14–23.

[2] R. A. Brooks, C. Breazeal, M. Marjanovic, B. Scassellati, M. M. Williamson, The Cog project: Building a humanoid robot, Lecture Notes in Computer Science 1562 (1999) 52–87.

[3] V. Braitenberg, Vehicles: Experiments in Synthetic Psychology, The MIT Press, 1986.
URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262521121

[4] M. Huber, A hybrid architecture for adaptive robot control.
URL http://scholarworks.umass.edu/dissertations/AAI9988799

[5] C. Geib, K. Mourão, R. Petrick, N. Pugeault, M. Steedman, N. Krüger, F. Wörgötter, Object action complexes as an interface for planning and robot control, in: Workshop 'Toward Cognitive Humanoid Robots' at IEEE-RAS International Conference on Humanoid Robots, 2006.

[6] F. Wörgötter, A. Agostini, N. Krüger, N. Shyloa, B. Porr, Cognitive agents — a procedural perspective relying on the predictability of object-action-complexes (OACs), Robotics and Autonomous Systems 57 (4) (2009) 420–432.

[7] R. E. Fikes, N. J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (3-4) (1971) 189–208.

[8] J. J. Gibson, The Perception of the Visual World, Houghton Mifflin, Boston, 1950.

[9] E. Sahin, M. Çakmak, M. R. Doğar, E. Uğur, G. Ücoluk, To afford or not to afford: A new formalization of affordances toward affordance-based robot control, Adaptive Behavior 15 (4) (2007) 447–472.

[10] R. Kowalski, M. Sergot, A logic-based calculus of events, New Generation Computing 4 (1986) 67–95.

[11] M. Steedman, Plans, affordances, and combinatory grammar, Linguistics and Philosophy 25 (5-6) (2002) 723–53.

[12] D. Kraft, N. Pugeault, E. Başeski, M. Popović, D. Kragic, S. Kalkan, F. Wörgötter, N. Krüger, Birth of the Object: Detection of Objectness and Extraction of Object Shape through Object Action Complexes, Special Issue on "Cognitive Humanoid Robots" of the International Journal of Humanoid Robotics 5 (2009) 247–265.

[13] R. P. A. Petrick, D. Kraft, N. Krüger, M. Steedman, Combining cognitive vision, knowledge-level planning with sensing, and execution monitoring for effective robot control, in: Proceedings of the Fourth Workshop on Planning and Plan Execution for Real-World Systems at ICAPS 2009, Thessaloniki, Greece, 2009, pp. 58–65.

[14] D. Kraft, R. Detry, N. Pugeault, E. Başeski, J. Piater, N. Krüger, Learning objects and grasp affordances through autonomous exploration, in: International Conference on Computer Vision Systems (ICVS), 2009.

[15] D. Harel, Dynamic logic, in: D. Gabbay, F. Guenthner (Eds.), Handbook of Philosophical Logic, Vol. II, Reidel, Dordrecht, 1984, pp. 497–604.

[16] D. Kraft, E. Başeski, M. Popović, A. M. Batog, A. Kjær-Nielsen, N. Krüger, R. Petrick, C. Geib, N. Pugeault, M. Steedman, T. Asfour, R. Dillmann, S. Kalkan, F. Wörgötter, B. Hommel, R. Detry, J. Piater, Exploration and planning in a three level cognitive architecture, in: International Conference on Cognitive Systems (CogSys), 2008.

[17] N. Krüger, M. Lappe, F. Wörgötter, Biologically Motivated Multi-modal Processing of Visual Primitives, Interdisciplinary Journal of Artificial Intelligence & the Simulation of Behaviour, AISB Journal 1 (5) (2004) 417–427.

[18] M. Popović, D. Kraft, L. Bodenhagen, E. Başeski, N. Pugeault, D. Kragic, N. Krüger, A strategy for grasping unknown objects based on co-planarity and colour information, submitted to RAS.

[19] D. Aarno, J. Sommerfeld, D. Kragic, N. Pugeault, S. Kalkan, F. Wörgötter, D. Kraft, N. Krüger, Early reactive grasping with second order 3d feature relations, in: The IEEE International Conference on Advanced Robotics, Jeju Island, Korea, 2007.

[20] L. Bodenhagen, D. Kraft, M. Popović, E. Başeski, P. E. Hotz, N. Krüger, Learning to grasp unknown objects based on 3d edge information, in: IEEE International Symposium on Computational Intellegience in Robotics and Automation, 2009.

[21] N. Krüger, M. Lappe, F. Wörgötter, Biologically Motivated Multi-modal Processing of Visual Primitives, The Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour 1 (5) (2004) 417–428.

[22] N. Pugeault, Early Cognitive Vision: Feedback Mechanisms for the Disambiguation of Early Visual Representation, Vdm Verlag Dr. Müller, 2008.

[23] R. Detry, N. Pugeault, J. Piater, A probabilistic framework for 3d visual object representation, IEEE transactions on Pattern Analysis and Machine Intelligence 31 (10) (2009) 1790–1803.

[24] R. Detry, E. Başeski, N. Krüger, M. Popović, Y. Touati, O. Kroemer, J. Peters, J. Piater, Learning object-specific grasp affordance densities, in: International Conference on Development and Learning, 2009.

[25] R. Detry, D. Kraft, A. G. Buch, N. Krüger, J. Piater, Refining grasp affordance models by experience, submitted to the International Conference on Robotics and Automation (2010).

[26] D. Omrčen, A. Ude, , A. Kos, Learning primitive actions through object exploration, in: International Conference on Humanoid Robots, Daejeon, Korea, 2008, pp. 306–311.

[27] R. Petrick, D. Kraft, K. Mourão, C. Geib, N. Pugeault, N. Krüger, M. Steedman, Representation and integration: Combining robot control, high-level planning, and action learning, in: Proceedings of the 6th International Cognitive Robotics Workshop (CogRob 2008), Patras, Greece, July 21-22, 2008.

[28] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 2nd Edition, Prentice Hall, Upper Saddle River, NJ, 2003.

[29] R. P. A. Petrick, F. Bacchus, A knowledge-based approach to planning with incomplete information and sensing, in: Proceedings of the International Conference on Artificial Intelligence Planning Systems (AIPS-02), 2002, pp. 212–221.

[30] Y. Freund, R. Schapire, Large margin classification using the perceptron algorithm, Machine Learning 37 (1999) 277–96. doi:10.1023/A:1007662407062.
URL http://dx.doi.org/10.1023/A:1007662407062

[31] R. Khardon, G. M. Wachman, Noise tolerant variants of the perceptron algorithm, Journal of Machine Learning Research 8 (2007) 227–248.
URL http://www.cs.tufts.edu/tr/techreps/TR-2005-8

[32] K. Mourão, R. P. A. Petrick, M. Steedman, Using kernel perceptrons to learn action effects for planning, in: International Conference on Cognitive Systems (CogSys 2008), 2008, pp. 45–50.

[33] A. Newell, H. Simon, GPS, a program that simulates human thought, in: E. A. Feigenbaum, J. Feldman (Eds.), Computers and Thought, McGraw-Hill, NY, 1963, pp. 279–293.

[34] C. Green, Application of theorem proving to problem solving, in: Proceedings of the First International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1969, pp. 741–747.

[35] E. D. Sacerdoti, The nonlinear nature of plans, in: Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1975, pp. 206–214.

[36] D. Vernon, G. G. Metta, G. Sandini, A survey of artificial cognitive systems: implications for the autonomous development of mental capabilities in computational agents, IEEE Transactions on Evolutionary Computation 11 (2007) 151–180.

[37] R. Detry, M. Popović, Y. P. Touati, E. Başeski, N. Krüger, J. Piater, Autonomuous learning of object-specific grasp affordance densities, submitted to the ICRA Workshop on Approaches to Sensorimotor Learning on Humanoid Robots (2009).

[38] S. Harnad, The symbol grounding problem, Physica D (42) (1990) 335–346.

[39] R. Brooks, Intelligence without reason, in: Proceedings of the International Joint Conference on Artificial Intelligence, 1991, pp. 569–595.

[40] R. Brooks, Elephants don't play chess, Robotics and Autonomous Systems (1990) 3–15.

[41] R. Pfeifer, M. Lungarella, F. Iida, Self-organization, embodiment, and biologically inspired robotics, Science 318 (2007) 1088–1093.

[42] R. Jacobs, M. Jordan, A. Barto, Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks, Cognitive Science 15 (2) (1991) 219–250.

[43] K. Narendra, J. Balakrishnan, Adaptive control using multiple models, IEEE Transaction on Automatic Control 42 (2) (1997) 171–187.

[44] M. Haruno, D. Wolpert, M. Kawato, MOSAIC model for sensorimotor learning and control, Neural Computation 13 (2001) 2201–2220.

[45] C. Miall, Modular motor learning, Trends in Cognitive Sciences 6 (1) (2002) 1–3.

[46] A. Samuel, Some studies in machine learning using the game of checkers, IBM Journal of Research and Development 3 (3) (1959) 210–229.

[47] N. J. Nilsson, Learning Machines, McGraw-Hill, 1965.

[48] L. P. Kaelbling, Learning functions in k-DNF from reinforcement, in: Proceedings of the Seventh International Workshop on Machine Learning, Morgan Kaufmann, 1990, pp. 162–169.

[49] T. Mitchel, Machine Learning, WCB McGraw Hill, 1997.

[50] H. Pasula, L. Zettlemoyer, L. P. Kaelbling, Learning symbolic models of stochastic domains, Journal of Artificial Intelligence 29 (2007) 309–352.

[51] J. Pearl, Probabilistic Reaasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.

[52] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum likehood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society (Series B) 39 (1) (1977) 1–38.

[53] D. Spiegelhalter, P. Dawid, S. Lauritzen, R. Cowell, Bayesian analysis in expert systems, Statistical Science 8 (1993) 219–283.

[54] S. Kok, P. Domingos, Learning the structure of Markov logic networks, in: Proceedings of the Twenty-Second International Conference on Machine Learing, ACM Press, 2005, pp. 441–448.

[55] E. Amir, A. Chang, Learning partially observable deterministic action models, Journal of Artificial Intelligence Research 33 (2008) 349–402.

[56] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufman, San Francisco, CA, 2004.

Norbert Krüger is a Professor at the Mærsk McKinney Møller Institute, University of Southern Denmark. He holds a M.Sc. from the Ruhr-Universität Bochum, Germany and his Ph.D. from the University of Bielefeld. He is a partner in several EU and national projects: PACO-PLUS, Drivsco, NISA, Handyman.

Norbert Krüger is leading the Cognitive Vision Lab which is focussing on computer vision and cognitive systems, in particular the learning of object representations in the context of grasping. He has also been working in the areas of computational neuroscience and machine learning.

Justus H. Piater is a professor of computer science at the University of Liège, Belgium, where he directs the Computer Vision research group. His research interests include computer vision and machine learning, with a focus on visual learning, closed-loop interaction of sensorimotor systems, and video analysis. He has published more than 80 technical papers in scientific journals and at international conferences. He earned his Ph.D. degree at the University of Massachusetts Amherst, USA, in 2001, where he held a Fulbright graduate student fellowship, and subsequently spent two years on a European Marie-Curie Individual Fellowship at INRIA Rhône-Alpes, France, before joining the University of Liège.

Christopher Geib is a Research Fellow at the University of Edinburgh School of Informatics. He holds a M.S. and Ph.D. from the University of Pennsylvania. His research focuses broadly on decision making and reasoning about actions under conditions of uncertainty including planning, scheduling, constraint based reasoning, human computer and robot interaction, and probabilistic reasoning. His recent research focus has been on probabilistic intent recognition through weighted model counting and planning based on grammatical formalisms.

Ronald Petrick is currently a Research Fellow in the School of Informatics at the University of Edinburgh. He received a Master of Mathematics degree in computer science from the University of Waterloo and a Ph.D. in computer science from the University of Toronto. His research interests include automated planning, knowledge representation and reasoning, and cognitive robotics.

Florentin Wörgötter has studied Biology and Mathematics in Düsseldorf. He received his PhD in 1988 in Essen working experimentally on the visual cortex before he turned to computational issues at the Caltech, USA (1988-1990). After 1990 he was researcher at the University of Bochumconcerned withexperimental and computational neuroscience of the visual system. Between 2000 and 2005 he had been Professor for Computational Neuroscience at the Psychology Department of the University of Stirling, Scotland where his interests strongly turned towards "Learning in Neurons". Since July 2005 he leads the Department for Computational Neuroscience of the Bernstein Center at the University of Göttingen. His main research interest is information processing in closed-loop perception-action systems, whichincludes aspects of sensory processing, motor control and learning/plasticity. These approaches are tested in walking as well as driving robotic implementations. His group has developed the RunBot a fast and adaptive biped walking robot.

Aleš Ude studied applied mathematics at the University of Ljubljana, Slovenia, and received his doctoral degree from the Faculty of Informatics, University of Karlsruhe, Germany. He was awarded the STA fellowship for postdoctoral studies in ERATO Kawato Dynamic Brain Project, Japan. He has been a visiting researcher at ATR Computational Neuroscience Laboratories, Kyoto, Japan, for a number of years and is still associated with this group. Currently he is a senior researcher at the Department of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia. His research focuses on imitation and action learning, perception of human activity, humanoid robot vision, and humanoid cognition.

Tamim Asfour received his diploma degree in electrical engineering and his Ph.D. degree in computer science from the University of Karlsruhe, Germany in 1994 and 2003, respectively. He is leader of the humanoid robotics research group at the institute for Anthromopmatics at the Karlsruhe Institute of Technology (KIT). His research interests include humanoid robotics, grasping and manipulation, imitation learning, system integration and mechatronics.

Dirk Kraft obtained a diploma degree in computer science from the University of Karlsruhe (TH), Germany in 2006 and a Ph.D. degree from the University of Southern Denmark in 2009. He is currently a research assistant in the Mærsk McKinney Møller Institute, University of Southern Denmark where he is working within the EU-project PACO-PLUS. His research interests include cognitive systems, robotics and computer vision.

Damir Omrčen received his PhD in robotics from the University of Ljubljana, Slovenia, in 2005. He is employed as a research assistant at the Department of Automation, Biocybernetics and Robotics at "Jozef Stefan" Institute in Ljubljana. His fields of interest include vision and robot control where he combines classical model based approaches and more advanced approaches based on exploration and learning.

Alejandro Agostini received the B.S. degrees in Bioengineering with honors from the National University of Entre Ríos, Argentina, and in Electronic Engineering from the National University of Catalonia (UPC), Spain. He is currently a senior Ph.D. student in Artificial Intelligence at the UPC, and is working at the Institut de Robòtica Industrial (IRI) under a work contract drawn up by the Spanish Research Council (CSIC). He has been involved in several national research projects related to biomedical signal processing (HRV, Biomechanics), health care informatics (QUIRON), multi-agent systems (CARREL), machine learning and robotics (SIRVENT), and has the partner (CSIC) major role in the EU projects PACO-PLUS and GARNICS,

developing cognitive systems for decision making. He performed several research stays at the Bernstein Centre for Computational Neuroscience, Göttingen, Germany, and at the University of Karlsruhe, Germany.

Rüdiger Dillmann is Professor at the Computer Science Faculty, Karlsruhe Institute of Technology (KIT), Germany. He is director of the Research Center for Information Science (FZI), Karlsruhe. He is scientific leader of German collaborative research center Humanoid Robots (SFB 588). His research interests include humanoid robotics, technical cognitive systems, machine learning, computer vision, robot programming by demonstration and medical applications of informatics.