Project no.:	**027657**
Project full title:	**Perception, Action & Cognition through learning of Object-Action Complexes**
Project Acronym:	**PACO-PLUS**
Deliverable no.:	**D4.1.4**
Title of the deliverable:	**Publication on fully grounded object-action complexes in terms of object shape and action affordances**

Contractual Date of Delivery to the CEC:	31.1.2010
Actual Date of Delivery to the CEC:	31.1.2010
Organisation name of lead contractor for this deliverable:	SDU
Author(s): Dirk Kraft, Norbert Krüger, Renaud Detry, Damir Omrčen, Aleš Ude, Christopher Geib, Ron Petrick, Mark Steedman, Justus Piater	
Participant(s): JSI, UEDIN, SDU, ULg	
Work package contributing to the deliverable:	WP1,WP2,WP4
Nature:	R
Version:	Final
Total number of pages:	10
Start date of project:	1st Feb. 2006 **Duration:** 48 month

Project co-funded by the European Commission within the Sixth Framework Programme (2002–2006)
Dissemination Level

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This report describes the work of the PACO-PLUS consortium on grounding through Object Action Complexes (OACs) done in WP4.1 in months 37–48. In particular we describe how a cognitive system grounds objects as well as their grasping and pushing affordances through exploration by means of OACs. It then uses these grounded representations for planning as well as for learning in discrete state spaces. This deliverable covers six publications [A, B, C, D, E, F] (some of them in the status submitted).

Keyword list: Vision, Object-Action Associations, Grasping, Pushing, Grounding

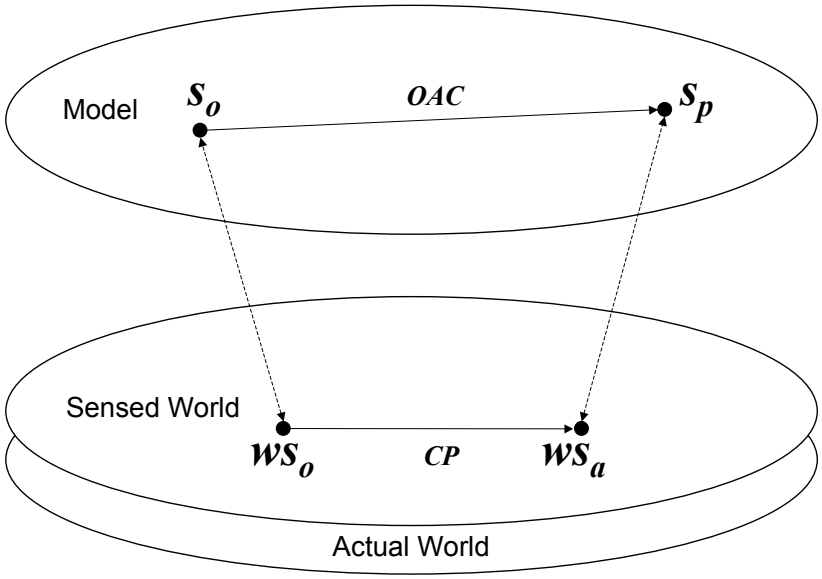
Table of Contents

1. Introduction

In this deliverable we present a formal definition of Object Action Complexes (OACs) (Section 2) and how to learn them, aiming at planning with these grounded representations. We describe the learning of a grasping OAC in Section 3 and how objects and grasping affordances become grounded in a cognitive system by pure exploration (Section 4). The grounding of an object independent pushing OAC and its use in a planning context is described in Section 5. Learning in the discrete state spaces of the planning level is described in Section 6.

This deliverable covers six publications [A, B, C, D, E, F] (some of them in the status submitted).

2. Formalisation of OACs

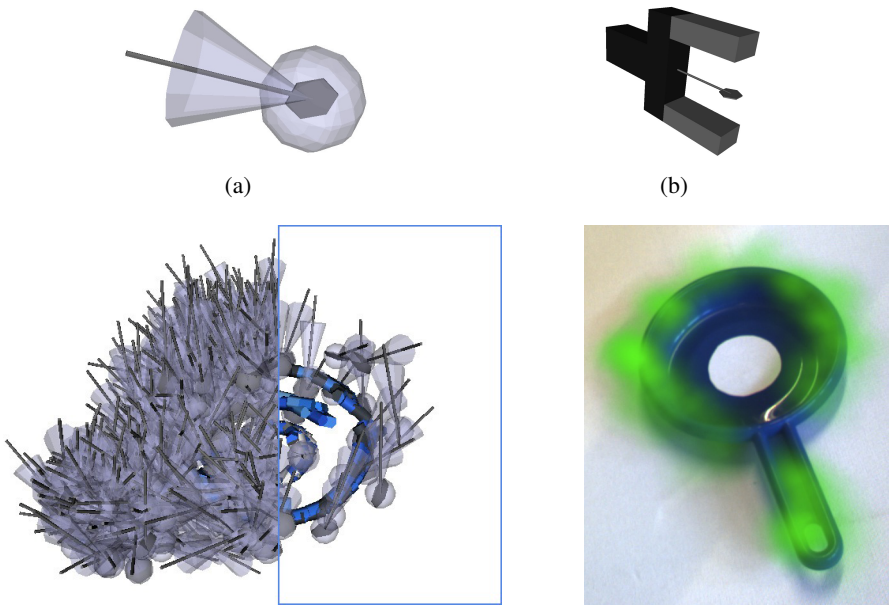


(a)

Figure 1: Graphical representation of an OAC and its relationship to the sensed world and a control program.

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. Object-Action Complexes are the basis for symbolic representations of sensorimotor experience. OACs are designed to capture the interaction between objects and associated actions in artificial cognitive systems.

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



(a) (b) (c) (d)

Figure 2: Grasp densities. Isotropic kernels (a) are used to represent a grasp success likely related to an actually executed two finger grasp (b). (c) By combining the information from multiple grasps the probability density function for grasp success over SO(3) can be represented by a set of these kernels. (d) Shows a different way to represent a grasp density. Here the green cloud indicates positions in which the likelihood of a successful grasp is high (orientation is disregarded in this image).

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.

The paper [D] 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.

3. Grasp Density Learning

An OAC representing object specific grasping affordances is connected to so called "grasp density learning". The publications [A, B] present this approach for learning object grasp affordance models in 3D from experience, and demonstrate its applicability through extensive testing and evaluation on a realistic and largely autonomous platform. Grasp affordance refers to relative object-gripper configurations that yield stable grasps. These affordances are represented probabilistically with grasp densities, i.e. continuous density functions defined on the space of 3D positions and orientations. A grasp density characterizes an object's grasp affordance; densities are linked to visual stimuli through registration with a visual model of the object they characterize. We explore a batch, experience-based learning paradigm where grasps sampled randomly from a density are performed, and an importance-sampling algorithm learns a refined density from the outcomes of these experiences. The first of such learning cycles is bootstrapped with a grasp density formed

from visual cues. The robot successfully exploits its actions into down weighting poor grasp solutions, which is reflected in the higher success rates achieved in subsequent learning cycles. The success rate of our method is quantified in a practical scenario where a robot needs to repeatedly grasp an object lying in an arbitrary pose, where each pose imposes a specific reaching constraint, and thus forces the robot to make use of the entire grasp density to select the most promising grasp within the set of achievable approaches. Figure 2 illustrates grasp densities and some of the achieved results.

4. Grounding objects and grasping affordances

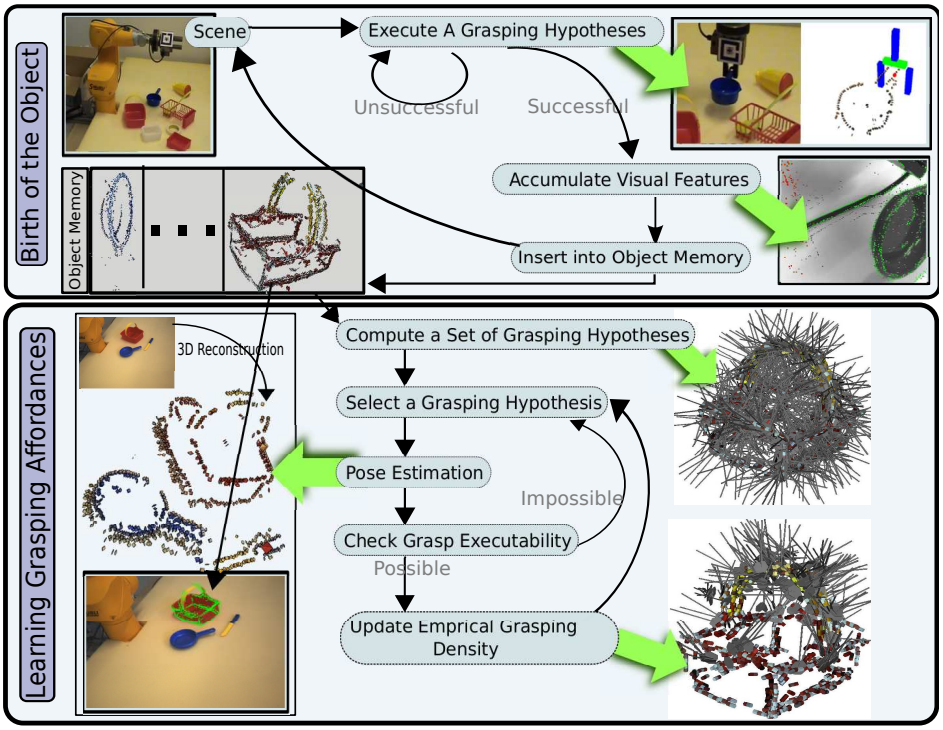


Figure 3: The two learning cycles. **(top)** In the first cycle visual object representations are learned. For this the grasping reflex OAC (to gain initial physical control over an unknown object) and the accumulation OAC (to view the object from different viewpoints and create a unified model) are combined. **(bottom)** In the second learning cycle we learn a grasp affordance model. The visual object models learned in the first cycle enable us to use the pose estimation to attach executed grasps to a common model. The grasp density OAC is used in this process and its inner model (an object specific grasp density) is learned.

The work [C] describes a system that uses a set of OACs for autonomous learning of visual object representations and their grasp affordances on a robot-vision system. It segments objects by grasping and moving 3D scene features, and creates probabilistic visual representations for object detection, recognition and pose estimation, which are then augmented by continuous characterizations of grasp affordances generated through biased, random exploration. The system uses a carefully balanced set of generic prior knowledge (1) about the embodiment of the system, (2) encoded in a vision system extracting structurally rich information from stereo image sequences as well as (3) contained in a number of built-in behavioural modules. This prior knowledge, applied in an autonomous exploration process, allows the system to generate object and grasping knowledge through interaction with its environment. Figure 3 shows the interaction of the used subcomponents from a technical point of view while Figures 4 and 5 shows how it can be interpreted in accordance with Baddeley's model of working memory [1].

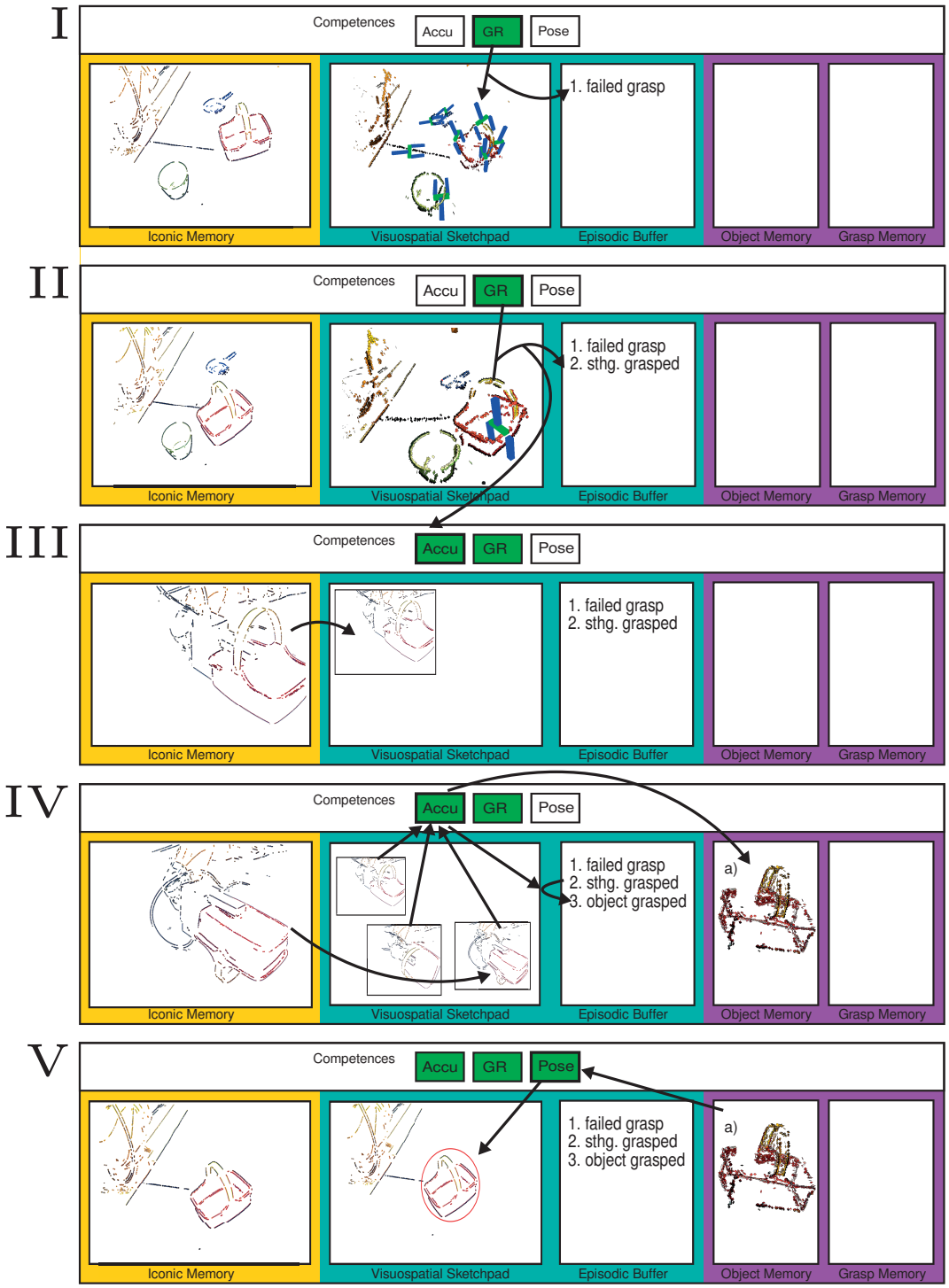


Figure 4: Illustration showing how the described grounding process can be interpreted in a accordance with human memory models (using Baddeley's model of working memory [1]). **I)** The grasping reflex is used. A collision between gripper and basket happens. The grasp is labelled as failed and stored in the episodic buffer. **II)** Successful application of the grasping reflex. The basket is grasped and the action labelled accordingly. **III)** The accumulation process is triggered by the successful grasp. **IV)** After a set of views have been collected they are incorporated into a common model. An object is born. The episodic buffer is updated (we now know that the previously perceived "something" corresponds to the new object) and the object model is stored in object memory. **V)** The new entry in the object memory enables a more complex representation in the visual sketchpad [1, p. 63] by the use of the pose estimation process.

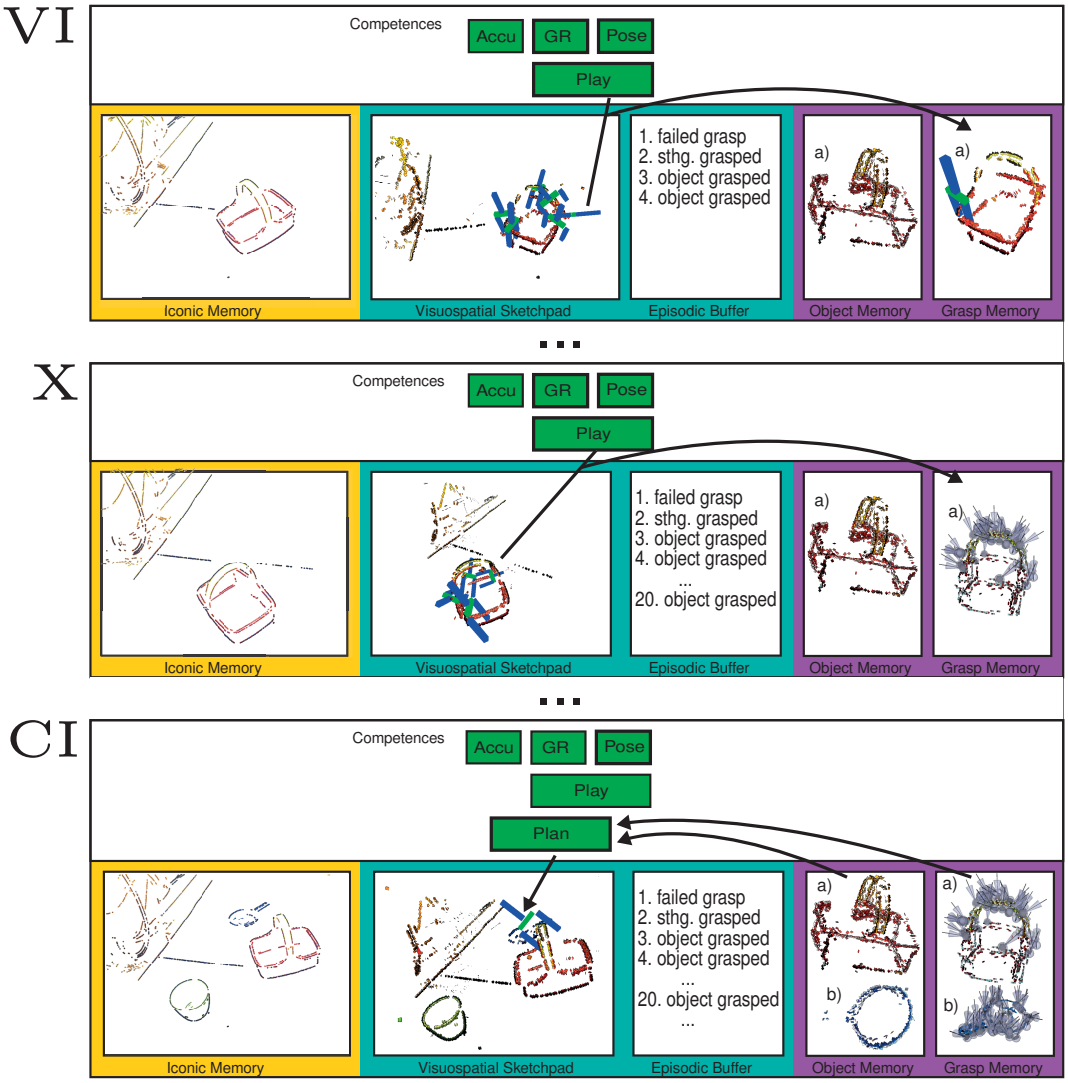


Figure 5: Illustration showing how the described grounding process can be interpreted in a accordance with human memory models (using Baddeley's model of working memory [1]). **VI)** A new competence combining grasping reflex and pose estimation, generating new entries in episodic buffer and grasp memory is activated (grasp affordance learning). **X)** After grasping the object multiple times the grasping model becomes sufficiently complex. **XI–C)** Additional objects are born and grasping models are learned. **CI)** Planning becomes possible with grounded object and grasp representations.

5. Relating motor knowledge to objects (Pushing)

We investigated in [E] how to relate robot motor knowledge to objects by exploration, thereby enabling the robot to autonomously acquire elementary object-action complexes. To this end they studied the learning of new manipulation OACs by observing the effects that the robot's exploratory movements have on objects. The proposed learning process results in a neural network that encodes the relationship between the robot's own manipulation actions and the resulting object behaviour. The learnt network can later be used for a goal-directed feedback control. The proposed approach is suitable for the acquisition of motor knowledge that is applicable not only to objects that the robot acted upon during learning, but to whole classes of objects. In a real-world experiment we showed that the developed system can be successfully applied to acquire a general pushing rule describing the relationship between the direction of push and the observed object motion for a

class of objects. In this way the robot acquires new sensorimotor knowledge without having any specialized prior model about the actions and/or objects. Figure 6 shows how this OAC has been used in a planning context on the humanoid robot ARMAR3 where the object needs to be pushed to the corner of the worktop before it can be grasped.

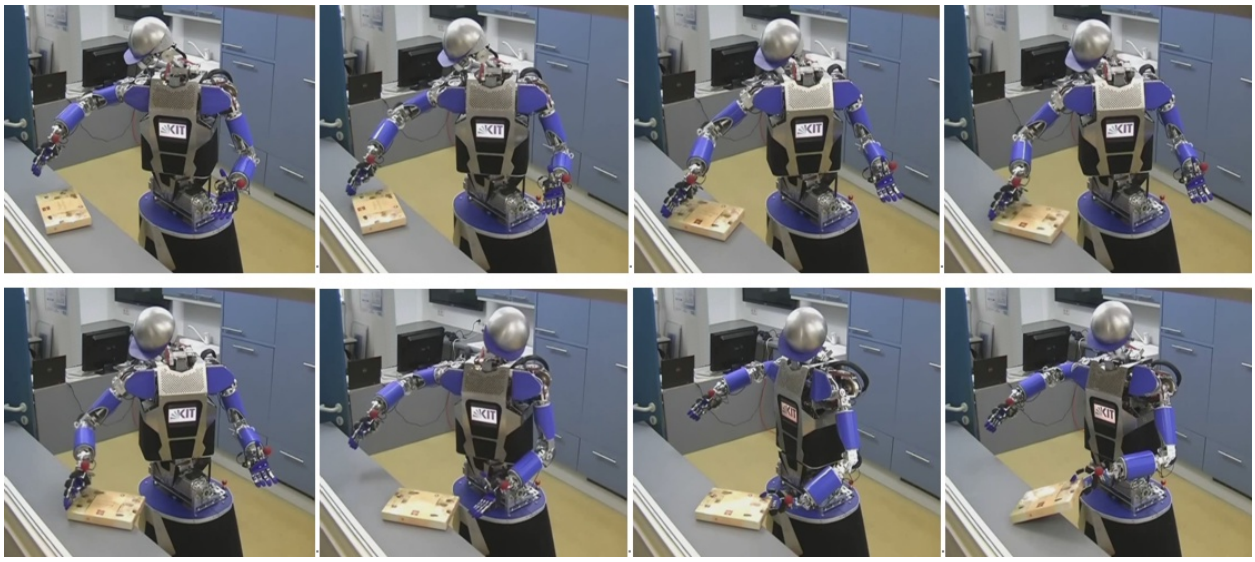
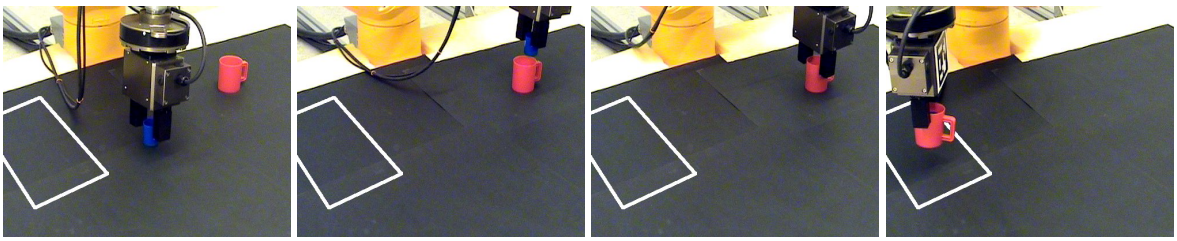


Figure 6: A sequence of robot pushes that bring the object to a graspable position.

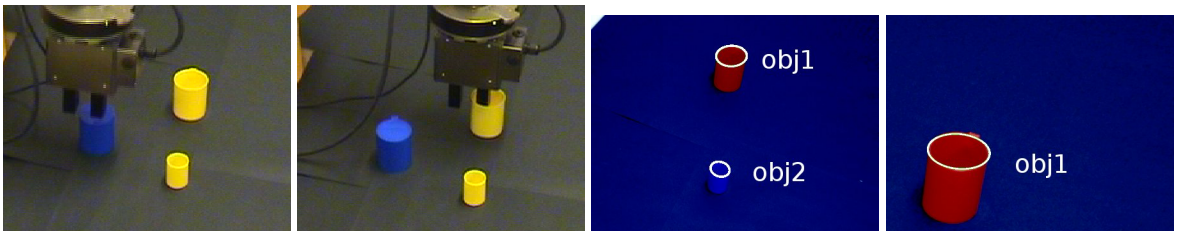
6. Planning

In [F] we describe an approach to robot control in real-world environments that integrates a cognitive vision system with a knowledge-level planner and plan execution monitor. This approach makes use of OACs to overcome some of the representational differences that arise between the low-level control mechanisms and high-level reasoning components of the system. We are here particularly interested in using OACs as a formalism that enables us to induce certain aspects of the representation, suitable for planning, through the robot's interaction with the world. We have implemented our ideas in a framework that supports object discovery, planning with sensing, action execution, and failure recovery, with the long term goal of designing a system that can be transferred to other robot platforms and planners. Figure 7 shows the execution of a simple plan to clean a table while Figure 8 shows sensing actions as well as resensing which is used when the plan monitor detects a problem.



(a) graspD-table(obj2) (b) putInto-object(obj2,obj1) (c) graspD-table(obj1) (d) putAway(obj1)

Figure 7: Executing a high level plan to clear a table.



(a) closed object (b) open object (c) normal view (d) "zoomed in"

Figure 8: (a-b) Poking into a potential opening as sensing action (sense-open). (c-d) Resensing as one possible solution to inconsistencies detected by the plan monitor. Resensing is realised here as focusing our attention on the relevant object.

7. Links to other Workpackages

Deliverable D4.1.4 is linked to and makes use of work made in a number of workpackages. It is linked to the software and hardware integration issues dealt with in WP1. In WP8, a number of sub-modules are used that have been developed in WP4, most notably the object specific grasping OAC as well as the pushing OAC. The planner and the acquisition of symbolic representations of OACs adapted to planning are developed elsewhere under WPs 4 and 5.

Attached Papers

[A] Renaud Detry, Emre Başeski, Norbert Krüger, Mila Popović, Younes Touati, Oliver Kroemer, Jan Peters, and Justus Piater. Learning object-specific grasp affordance densities. In *International Conference on Development and Learning*, 2009.

[B] Renaud Detry, Dirk Kraft, Anders Glent Buch, Norbert Krüger, and Justus Piater. Refining grasp affordance models by experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010. (accepted).

[C] Dirk Kraft, Renaud Detry, Nicolas Pugeault, Emre Başeski, Justus Piater, and Norbert Krüger. Learning objects and grasp affordances through autonomous exploration. In *International Conference on Computer Vision Systems (ICVS)*, 2009.

[D] Norbert Krüger, Justus Piater, Christopher Geib, Ronald Petrick, Mark Steedman, Florentin Wörgötter, Aleš Ude, Tamim Asfour, Dirk Kraft, Damir Omrčen, Alejandro Agostini, and Rüdiger Dillmann. Object-action complexes: Grounded abstractions of sensorimotor processes. *Robotics and Autonomous Systems*. (submitted).

[E] D. Omrcen, C. Böge, T. Asfour, A. Ude, and R. Dillmann. Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Paris, France, 2009.

[F] Ronald Petrick, Dirk Kraft, Norbert Krüger, and Mark 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*, pages 58–65, Thessaloniki, Greece, September 2009.

References

[1] Alan Baddeley. *Working Memory, Thought, and Action*. Oxford University Press, 2007.

Learning Object-specific Grasp Affordance Densities

R. Detry, E. Başeski, M. Popović, Y. Touati, N. Krüger, O. Kroemer, J. Peters and J. Piater

***Abstract*—This paper addresses the issue of learning and representing *object grasp affordances*, i.e. object-gripper relative configurations that lead to successful grasps. The purpose of grasp affordances is to organize and store the whole knowledge that an agent has about the grasping of an object, in order to facilitate reasoning on grasping solutions and their achievability. The affordance representation consists in a continuous probability density function defined on the 6D gripper pose space – 3D position and orientation –, within an object-relative reference frame. Grasp affordances are initially learned from various sources, e.g. from imitation or from visual cues, leading to *grasp hypothesis densities*. Grasp densities are attached to a learned 3D visual object model, and pose estimation of the visual model allows a robotic agent to execute *samples* from a grasp hypothesis density under various object poses. Grasp outcomes are used to learn *grasp empirical densities*, i.e. grasps that have been confirmed through experience. We show the result of learning grasp hypothesis densities from both imitation and visual cues, and present grasp empirical densities learned from physical experience by a robot.**

I. Introduction

Grasping previously unknown objects is a fundamental skill of autonomous agents. Human grasping skills improve with growing experience with certain objects. In this paper, we describe a mechanism that allows a robot to learn grasp affordances [12] of objects described by learned visual models. Our first aim is to organize and memorize, independently of grasp information sources, the whole knowledge that an agent has about the grasping of an object, in order to facilitate reasoning on grasping solutions and their likelihood of success. We represent the affordance of an object for a certain grasp type through a continuous probability density function defined on the 6D gripper pose space $SE(3)$, within an object-relative reference frame. The computational encoding is *nonparametric*: A density is represented by a large number of weighted samples called *particles*. The probabilistic density in a region of space is given by the local density of the particles in that region. The underlying continuous density function is accessed through *kernel density estimation* [27].

The second contribution of this paper is a framework that allows an agent to learn initial affordances from various grasp cues, and enrich its grasping knowledge through experience. Affordances are initially constructed from human demonstration, or from a model-based method [1]. The grasp data produced by these *grasp sources* is used to build continuous *grasp hypothesis densities* (Section VI). These densities are attached to a 3D visual object model learned beforehand [9], which allows a robotic agent to execute *samples* from a grasp hypothesis density under arbitrary object poses, by using the visual model to estimate the 3D pose of the object.

The success rate of grasp samples depends on the source that is used to produce initial grasp data. However, no existing method can claim to be perfect. For example, data collected from imitation will suffer from the physical and mechanical difference between a human hand and a robotic gripper. In the case of grasps computed from a 3D model, results will be impeded by errors in the model, such as missing parts or imprecise geometry. In all cases, only a fraction of the hypothesis density samples will succeed; it thus seems necessary to also learn from experience. To this end, we use samples from grasp hypothesis densities that lead to a successful grasp to learn *grasp empirical densities*, i.e. grasps that have been confirmed through experience.

While we do not explicitly model human development, our learning-based approach loosely follows the biological example. In contrast to traditional robotics approaches that employ 3D scans or CAD models of the object and compute grasp parameters based on analytical physical models [2], [4], [22], we *learn* gripper poses that lead to stable grasps. We start with hypothesis densities, which may originate from a premature grasping mechanism providing only little bias towards stable grasp configurations. While this yields a rather low success rate, it is sufficient to bootstrap the acquisition of object-specific knowledge for skilled grasping. This procedure – feature-induced grasping refined by sensorimotor exploration – loosely resembles human acquisition of grasping skills during infancy, and constitutes a promising avenue towards viable robotic grasping, as it does for humans. Moreover, many of the employed methods (visual model and inference, vision-induced grasping, continuous affordances) resemble their biological counterparts, as explained in Section IV.

A unified representation of grasp affordances can potentially lead to many different applications. For instance, a grasp planner could combine a grasp density with hardware physical capabilities (robot reachability) and external constraints (obstacles) in order to select the grasp that has the largest chance of success within the subset of achievable grasps. Another possibility is the use of continuous grasp success likelihoods to infer robustness requirements on the execution particular

R. Detry and J. Piater are with the University of Liège, Belgium. Email: Renaud.Detry@ULg.ac.be.

E. Başeski, M. Popović, Y. Touati and N. Krüger are with the University of Southern Denmark.

O. Kroemer and J. Peters are with the MPI for Biological Cybernetics, Tübingen, Germany.

978-1-4244-4118-1/09/$25.00

grasp: if a grasp is centered on a narrow peak, pose estimation and servoing should be performed with more caution than when the grasp is placed in a wide region of high success likelihood.

II. Related Work

Object grasps can emerge in many different ways. A popular approach is to compute grasping solutions from the geometric properties of an object, typically obtained from a 3D object model. The most popular 3D model for grasping is probably the 3D mesh [17], [22], obtained e.g. from CAD or superquadrics fitting [3]. However, grasping has also successfully been achieved using models consisting of 3D surface patches [26], 3D edge segments [1], or 3D points [15].

When combined with an object pose estimation technique, the previous methods allow a robot to execute a grasp on a specific object. This involves object pose estimation, computation of a grasp on the aligned model, then servoing to the object and performing the grasp [17].

Means of representing grasp affordances probabilistically have been discussed in the work of de Granville et al. [7], which is quite closely related in spirit to ours. In this work, affordances correspond to object-relative hand approach orientations, although an extension where object-relative positions are also modeled is under way [6]. The aim of the authors is to build compact sets of canonical grasp approaches from human demonstration; they mean to compress a large number of examples provided by a human teacher into a small number of clusters. An affordance is expressed through a density represented as a mixture of position-orientation kernels; machine learning techniques are used to compute mixture and kernel parameters that best fit the data. This is quite different from our approach, where a density is represented with a much larger number of simpler kernels. Conceptually, using a larger number of kernels allows us to use significantly simpler learning methods (down to mere resampling of input data, see Section VI-A). Also, the representation of a grasp cluster through a single position-orientation kernel requires the assumption that hand position and orientation are independent within the cluster, which is generally not true. Representing a cluster with many particles can intrinsically capture more of the position-orientation correlation (see Section VII, and in particular Fig. 7). The affordance densities presented by de Granville et al. correspond to the hypothesis densities developed in this paper.

Learning grasp affordances from experience was demonstrated by Stoytchev [28], [29]. In this work, a robot discovers successful grasps through random exploratory actions on a given object. When subsequently confronted with the same object, the robot is able to generate a grasp that should present a high likelihood of success.

III. System Overview

The visual object model to which affordances are attached is the part-based model of Detry et al. [9] (Section IV-C). An object is modeled with a hierarchy of increasingly expressive object parts called *features*. The single top feature of a hierarchy represents the whole object. Features at the bottom of the hierarchy represent short 3D edge segments for which evidence is collected from stereo imagery via the Early-Cognitive-Vision (ECV) system of Krüger et al. [18], [25] (Section IV-A). In the following, we refer to these edge segments as *ECV descriptors*. The hierarchical model grounds its visual evidence in ECV reconstructions: a model is learned from segmented ECV descriptors, and the model can be used to recover the pose of the object within an ECV representation of a cluttered scene.

(a) ECV descriptors (b) Accumulated reconstructions

Fig. 1. ECV reconstructions

The mathematical representation of grasp densities and their association to hierarchical object models is discussed in Section V. In Section VI, we demonstrate the learning and refining of grasp densities from two grasp sources. The first source is imitation of human grasps. The second source uses a model-based algorithm which extracts grasping cues from an ECV reconstruction (Section IV-B).

IV. Methods

This section briefly describes the methods that are brought together for modeling the visual percepts of an object, and for bootstrapping hypothesis densities from visual cues. These sophisticated methods have proved essential for a robust execution of grasps on arbitrary objects in arbitrary poses.

A. Early Cognitive Vision

ECV descriptors [18], [25] represent short edge segments in 3D space, each ECV descriptor corresponding to a circular image patch with a 7-pixel diameter. To create an ECV reconstruction, pixel patches are extracted along image contours, within images captured with a calibrated stereo camera. The ECV descriptors are then computed with stereopsis across image pairs; each descriptor is thus defined by a 3D position and orientation. Descriptors may be tagged with color information, extracted from their corresponding 2D patches (Fig. 1a). The descriptors have been motivated by the concept of hypercolumns in the human visual system [14].

ECV reconstructions can further be improved by manipulating objects with a robot arm, and *accumulating* visual information across several views through structure-from-motion techniques [13]. Assuming that the motion adequately spans the object pose space, a complete 3D reconstruction of the object can be generated, eliminating self-occlusion issues [16] (see Fig. 1b).

B. Grasp Reflex From Co-planar ECV Descriptors

Pairs of ECV descriptors that are on the same plane and which have color information such that two similar colors are

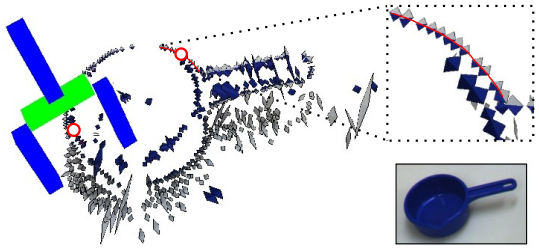


Fig. 2. Grasp reflex based on visual data.

pointing towards each other can be used to define grasps. Grasp position is defined by the location of one of the descriptors. Grasp orientation is calculated from the normal of the plane linking the two descriptors, and the orientation of the descriptor at which the grasp is located [16] (see Fig. 2). The grasps generated by this method will be referred to as *reflexes*. Since each pair of co-planar descriptors generates multiple reflexes, a large number of these are available. It has been shown that such a relatively simple mechanism can lead to success rates of around 30% [24] and hence can be used to bootstrap more sopisticated grasp representations as been described in this paper.

C. Feature Hierarchies For 3D Visual Object Representation

As explained in Section IV-A, an ECV reconstruction models a scene or an object with low-level descriptors. This section outlines a higher-level 3D object model [9] that grounds its visual evidence in ECV representations.

An object is modeled with a hierarchy of increasingly expressive object parts called *features*. Features at the bottom of the hierarchy (*primitive* features) represent ECV descriptors. Higher-level features (*meta*-features) represent geometric configurations of more elementary features. The single top feature of a hierarchy represents the object.

Unlike many part-based models, a hierarchy consists of features that may have several *instances* in a scene. To illustrate this, let us consider a part-based model of a bike, in which we assume a representation of wheels. Traditional part-based models [11], [5] would work by creating two wheel parts – one for each wheel. Our hierarchy however uses a single *generic* wheel feature; it stores the information on the existence of *two* wheels *within* the wheel feature. Likewise, a primitive feature represents a *generic* ECV descriptor, e.g. any descriptor that has a red-like color. While an object like the basket of Fig. 1 produces hundreds of red ECV descriptors, a hierarchy representing the basket will, in its simplest form, contain a single red-like primitive feature; it will encode internally that this feature has many instances within a basket object.

A hierarchy is implemented in a Markov tree. Features correspond to hidden nodes of the network; when a model is associated to a scene (during learning or detection), the pose distribution of feature i in the scene is represented through a random variable X_i. Random variables are thus defined over the pose space, which exactly corresponds to the Special Euclidean group $SE(3) = \mathbb{R}^3 \times SO(3)$. The random variable X_i associated to feature i effectively links that feature to its instances: X_i represents as one probability density function the pose distribution of all the instances of feature i, therefore avoiding specific model-to-scene correspondences.

The geometric relationship between two neighboring features i and j is encoded in a compatibility potential $\psi_{ij}(X_i, X_j)$. A compatibility potential represents the pose distribution of all the instances of the child feature in a reference frame defined by the parent feature; potentials are thus also defined on $SE(3)$.

The only observable features are primitive features, which receive evidence from the ECV system. Each primitive feature i is linked to an observed variable Y_i; the statistical dependency between a hidden variable X_i and its observed variable Y_i is encoded in an observation potential $\phi_i(X_i)$, which represents the pose distribution of ECV descriptors that have a color similar to the color of primitive feature i.

Density functions (random variables, compatibility potentials, observation potentials) are represented nonparametrically: a density is represented by a set of particles [9].

D. Pose Estimation

The hierarchical model presented above can be used to estimate the pose of a known object in a cluttered scene. Estimating the pose of an object amounts to deriving a posterior pose density for the top feature of its hierarchy, which involves two operations [9]:

1) Extract ECV descriptors, and transform them into observation potentials.
2) Propagate evidence through the graph using an applicable inference algorithm.

Each observation potential $\phi_i(X_i)$ is built from a subset of the early-vision observations. The subset that serves to build the potential $\phi_i(X_i)$ is the subset of ECV descriptors that have a color that is close enough to the color associated to primitive feature i.

Evidence is propagated through the hierarchy using a belief propagation (BP) algorithm [23], [30]. BP works by exchanging *messages* between neighboring nodes. Each message carries the belief that the sending node has about the pose of the receiving node. In other words, a message allows the sending feature to probabilistically vote for all the poses of the receiving feature that are consistent with its own pose – consistency being defined by the compatibility potential through which the message flows. Through message passing, BP propagates collected evidence from primitive features to the top of the hierarchy; each feature probabilistically votes for all possible object configurations consistent with its pose density. A consensus emerges among the available evidence, leading to one or more consistent scene interpretations. The pose likelihood for the whole object is eventually read out of the top feature; if the object is present twice in a scene, the top feature density should present two major modes. The global belief about the object pose may also be propagated from the top node down the hierarchy, reinforcing globally consistent evidence and permitting the inference of occluded features.

Within a biological system, cortical visual processing involves both bottom-up propagation of perceptual stimuli and

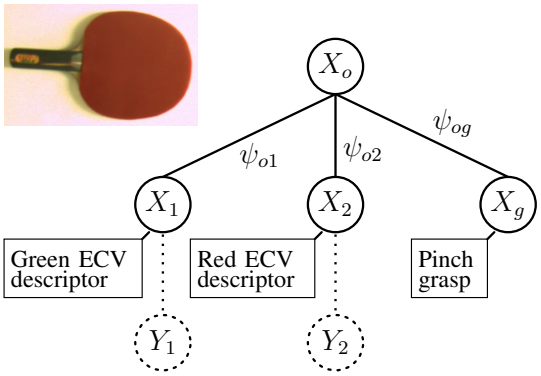


Fig. 3. Multi-sensory modeling of a table-tennis paddle with a 2-level hierarchy. The paddle is represented by feature o (top). Feature 1 represents a generic green ECV descriptor. The rectangular configuration of green edges around the handle of the paddle is encoded in ψ_{o1}. Y_1 and Y_2 are observed variables, which link features 1 and 2 to the visual evidence produced by ECV. X_g is a grasp feature, linked to the object feature through the pinch grasp affordance ψ_{og}.

modulation by top-down signals. Lee and Mumford [20] suggested that the visual processing stream might perform Bayesian inference within an undirected Markov chain, a crucial aspect of which is that ambiguities at low levels should persist and propagate upwards until they can be resolved by integrating larger-scale evidence or top down expectations. As a biologically plausible implementation of inference with arbitrary, possibly multimodal probability densities, Lee and Mumford suggest belief propagation using particle representations. The representation and methods presented above constitute a working computer implementation of central aspects of Lee and Mumford's model.

Algorithms that build hierarchies from accumulated ECV reconstructions are discussed in prior work [8].

V. Representing Grasp Densities

This section is focused on the probabilistic representation of grasp affordances, and on the integration of grasp affordances within the hierarchical object model. By *grasp affordance*, we refer to the different ways to place a hand or a gripper near an object so that closing the gripper will produce a stable grip. The grasps we consider are parametrized by a 6D gripper pose composed of a 3D position and a 3D orientation.

A. Grasp Features

Within our framework, a grasp affordance is represented with a probability density function defined on $SE(3)$ in an object-relative reference frame. Probabilistically speaking, we store an expression of the joint distribution $\mathbf{p}(X_o, X_g)$, where X_o is the pose distribution of the object, and X_g is the grasp affordance. This is done by adding a new "grasp" feature to the hierarchical Markov network, and linking it to the top feature (see Fig. 3). The statistical dependency of X_o and X_g is held in a compatibility potential $\psi_{og}(X_o, X_g)$, which exactly corresponds to the grasp density: $\psi_{og}(X_o, X_g)$ holds the relative configuration of grasp affordance and object pose, i.e. the grasp distribution into the reference frame of the top feature.

When an object model has been visually aligned to an object instance (i.e. when the marginal posterior of the top feature has been computed from visually-grounded bottom-up inference), the grasp affordance of the object *instance* is computed through top-down BP inference, by sending a message from X_o to X_g through $\psi_{og}(X_o, X_g)$. Intuitively, this corresponds to transforming the grasp density to align it to the current object pose, yet explicitly taking the uncertainty on object pose into account to produce a posterior grasp density that acknowledges visual noise.

B. Continuous Grasp Densities

From a mathematical point of view, grasp potentials are identical to visual potentials. They can thus be encoded with the same nonparametric density representation. Density evaluation is performed by assigning a kernel function to each particle supporting the density, and summing the evaluation of all kernels. Sampling from a distribution is performed by sampling from the kernel of a particle ℓ selected from $\mathbf{p}(\ell = i) \propto w^i$, where w^i is the weight of particle i.

Grasp densities (grasp potentials and grasp random variables) are defined on the Special Euclidean group $SE(3) = \mathbb{R}^3 \times SO(3)$, where $SO(3)$ is the Special Orthogonal group (the group of 3D rotations). We use a kernel that factorizes into two functions defined on $\mathbb{R}^3$ and $SO(3)$. Denoting the separation of an $SE(3)$ point x into a translation λ and a rotation θ by

$$x = (\lambda, \theta), \quad \mu = (\mu_t, \mu_r), \quad \sigma = (\sigma_t, \sigma_r),$$

we define our kernel with

$$\mathbf{K}(x; \mu, \sigma) = \mathbf{N}(\lambda; \mu_t, \sigma_t)\, \mathbf{\Theta}(\theta; \mu_r, \sigma_r) \tag{1}$$

where μ is the kernel mean point, σ is the kernel bandwidth, $\mathbf{N}(\cdot)$ is a trivariate isotropic Gaussian kernel, and $\mathbf{\Theta}(\cdot)$ is an orientation kernel defined on $SO(3)$. Denoting by θ' and μ_r' the quaternion representations of θ and μ_r [19], we define the orientation kernel with the Dimroth-Watson distribution [21]

$$\mathbf{\Theta}(\theta; \mu_r, \sigma_r) = \mathbf{W}(\theta'; \mu_r', \sigma_r) = C_w(\sigma_r) e^{\sigma_r (\mu_r'^{\top} \theta')^2} \tag{2}$$

where $C_w(\sigma_r)$ is a normalizing factor. This kernel corresponds to a Gaussian-like distribution on $SO(3)$. The Dimroth-Watson distribution inherently handles the double cover of $SO(3)$ by quaternions [7].

The bandwidth σ associated to a density should ideally be selected jointly in $\mathbb{R}^3$ and $SO(3)$. However, this is difficult to do. Instead, we set the orientation bandwidth σ_r to a constant allowing about 10° of deviation; the location bandwidth σ_t is then selected using a k-nearest neighbor technique [27].

The expressiveness of a single $SE(3)$ kernel (1) is rather limited: location and orientation components are both isotropic, and within a kernel, location and orientation are modeled independently. Nonparametric methods account for the simplicity of individual kernels by employing a large number of them: a grasp density will typically be supported by a thousand particles. Fig. 4a shows an intuitive rendering of an $SE(3)$ kernel from a grasp density. Fig. 4b and Fig. 4c illustrate continuous densities.

(a) (b) (c)

Fig. 4. Grasp density representation. The top image of Fig. (a) illustrates a particle from a nonparametric grasp density, and its associated kernel widths: the translucent sphere shows one position standard deviation, the cone shows the variance in orientation. The bottom image illustrates how the schematic rendering used in the top image relates to a physical gripper. Fig. (b) shows a 3D rendering of the kernels supporting a grasp density for a table-tennis paddle (for clarity, only 30 kernels are rendered). Fig. (c) indicates with a green mask of varying opacity the values of the location component of the same grasp density along the plane of the paddle (orientations were ignored to produce this last illustration).

VI. Learning Grasp Densities

This section explains how hypothesis densities are learned from source data (Section VI-A), and how empirical densities are learned from experience (Section VI-B).

A. Hypothesis Densities From Examples

Initial grasp knowledge, acquired for instance from imitation or reflex, is structured as a set of grasps parametrized by a 6D pose. Given the nonparametric representation, building a density from a set of grasps is straightforward – grasps can directly be used as particles representing the density. We typically limit the number of particles in a density to a thousand; if the number of grasps in a set is larger than 1000, the density is *resampled*: kernels are associated the particles, and 1000 samples are drawn and used as a representation replacement.

Since we wish to record object-relative information, densities have to be transformed to the reference frame of the object. Assuming that grasp poses are initially defined in the same reference frame as the visual ECV descriptors, this can be done by aligning the hierarchical model of the object by visual inference, and transforming the particles of each grasp density in the reference frame defined by the pose of the top feature of the aligned model.

A grasp density is integrated into the hierarchical object model through a new primitive feature i. The new feature is linked to the top model feature o through a potential $\psi_{io}(X_i, X_o)$ that corresponds to the object-relative density.

B. Empirical Densities Through Familiarization

As the name suggests, hypothesis densities do not pretend to reflect the true properties of an object. Their main defect is that they may strongly suggest grasps that might not be applicable at all, for instance because of gripper discrepancies in imitation-based hypotheses. A second, more subtle issue is that the grasp data used to learn hypothesis densities will generally be afflicted with a source-dependent spatial bias. A very good example can be made from the reflex computation of Section IV-B. Reflexes are computed from ECV visual descriptors. Therefore, parts of an object that have a denser visual resolution will yield more reflexes, incidentally biasing the corresponding region of the hypothesis density to a higher value. The next paragraph explains how grasping experience can be used to compute new densities (*empirical* densities) that better reflect gripper-object properties.

Empirical densities are leaned from the execution of *samples* from a hypothesis density, intuitively allowing the agent to familiarize itself with the object by discarding wrong hypotheses and refining good ones. Familiarization thus essentially consists in autonomously learning an *empirical* density from the outcomes of sample executions. A simple way to proceed is to build an empirical density directly from successful grasp samples. However, this approach would inevitably propagate the spatial bias mentioned above to empirical densities. Instead, we use importance sampling [10] to properly weight grasp outcomes, allowing us to draw samples from the physical grasp affordance of an object. The weight associated to a grasp sample x is computed as $\mathbf{a}(x) \,/\, \mathbf{q}(x)$, where $\mathbf{a}(x)$ is 1 if the execution of x has succeeded, 0 else, and $\mathbf{q}(x)$ corresponds to the value of the continuous hypothesis density at x. A set of these weighted samples directly forms a grasp empirical density that faithfully and uniformly reflects intrinsic gripper-object properties. Each empirical density is associated to the object model in the same way as hypothesis densities, through a new feature in the hierarchical network.

VII. Results

This section illustrates hypothesis densities learned from imitation and reflexes, and empirical densities are learned by grasping objects with a 3-finger Barrett hand. Densities are built for two objects: the table-tennis paddle of Fig. 3, and a toy plastic jug (Fig. 6). The experimental scenario is described below.

For each object, the experiment starts with a visual hierarchical model, and a set of grasps. For the paddle, grasps are generated with the method described in Section IV-B. Initial data for the jug was collected through human demonstration, using a motion capture system. From these data, a hypothesis density is built for each object. The particles supporting the hypothesis densities are rendered in Fig. 5.

In order to refine affordance knowledge, feedback on the execution of hypothesis density samples is needed. Grasps are executed with a Barrett hand mounted on an industrial arm. As illustrated in Fig. 6, the hand preshape is a parallel-fingers, opposing-thumb configuration. The reference pose of the hand is set for a pinch grasp, with the tool center point located in-between the tips of the fingers – similar to the reference pose illustrated in Fig. 4a. A grasp is considered successful if the robot is able to firmly lift up the object, success being asserted by raising the robotic hand while applying a constant, inward force to the fingers, and checking whether at least one finger is not fully closed.

As expected, the hypothesis densities led to a rather low success rate. We have observed approximate success rates of

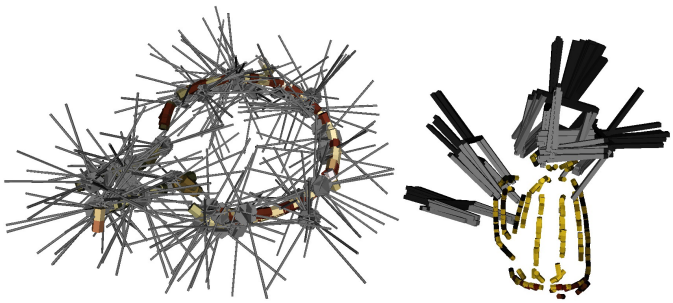


Fig. 5. Particles supporting grasp hypothesis densities.

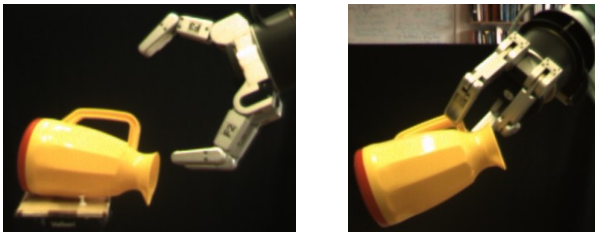


Fig. 6. Barrett hand grasping the toy jug.

14% for the paddle and 20% for the jug. Eventually, sets of 100 and 25 successful grasps were collected for the paddle and the jug respectively. This information was then used to build a grasp empirical density, following the procedure described in Section VI-B. Samples from the resulting empirical densities are shown in Fig. 7. For the paddle, the main evolution from hypothesis to empirical density is the removal of a large number of grasps for which the gripper wrist collides with the paddle body. Grasps presenting a steep approach relative to the plane of the paddle were also discarded, thereby preventing fingers from colliding with the object during hand servoing. None of the pinch-grasps at the paddle handle succeeded, hence their absence from the empirical density.

While grasping the top of the jug is easy for a human hand, it proved to be very difficult for the Barrett hand with parallel fingers and opposing thumb. Consequently, a large portion of the topside grasps suggested by the hypothesis density are not represented in the empirical density. The most reliable grasps approach the handle of the jug from above; these grasps are strongly supported in the empirical density.

The left image of Fig. 7 clearly illustrates the correlation between grasp positions and orientations: moving along the edge of the paddle, grasp approaches are most often roughly perpendicular to the local edge tangent. The nonparametric density representation successfully captures this correlation.

VIII. Conclusion And Future Work

We presented a framework for representing and learning object grasp affordances, and linking these to a visual object model. The affordance representation is probabilistic and nonparametric: an affordance is recorded in a continuous probability density function supported by a set of particles.

Grasp densities are initially learned from visual cues or imitation, leading to grasp hypothesis densities. Using the visual model for pose estimation, an agent is able to execute *samples*

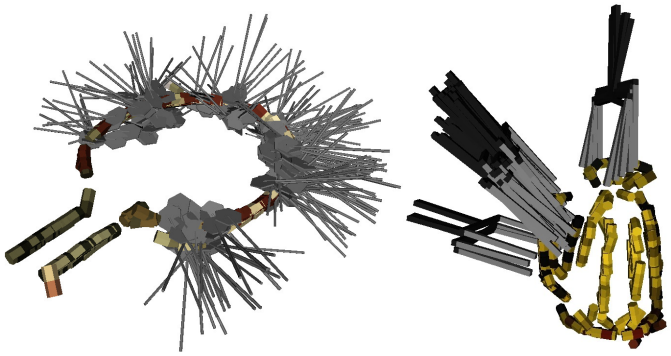


Fig. 7. Samples drawn from grasp empirical densities.

from a hypothesis density under arbitrary object poses. Observing the outcomes of these grasps allows the agent to learn from experience: an importance sampling algorithm is used to infer faithful object grasp properties from successful grasp samples. The resulting *grasp empirical densities* eventually allow for more robust grasping. The quantitative evaluation of this improvement will require large scale experiments.

Importance Sampling is a batch learning method, that requires the execution of a large number of grasps before an empirical density can be built. Learning empirical densities *on-line* would be very convenient, which is a path we plan to explore next.

We currently learn visual and grasp features independently, and connect them through a single top-level model feature. Yet, a part-based representation offers an elegant way to *locally* encode visuomotor descriptions. One of our goals is to learn visual parts that share the same grasp properties across different objects. This way, a grasp feature will be directly and exclusively connected to the visual evidence that predicts its applicability, allowing for its generalization across objects.

Acknowledgments

This work was supported by the Belgian National Fund for Scientific Research (FNRS) and the EU Cognitive Systems project PACO-PLUS (IST-FP6-IP-027657). We thank Volker Krüger and Dennis Herzog for their support during the recording of the human demonstration data.

References

[1] Daniel Aarno, Johan Sommerfeld, Danica Kragic, Nicolas Pugeault, Sinan Kalkan, Florentin Wörgötter, Dirk Kraft, and Norbert Krüger. Early reactive grasping with second order 3D feature relations. In *The IEEE International Conference on Advanced Robotics*, 2007.

[2] A. Bicchi and V. Kumar. Robotic grasping and contact: a review. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, 1, 2000.

[3] G. Biegelbauer and M. Vincze. Efficient 3D object detection by fitting superquadrics to range image data for robot's object manipulation. In *IEEE International Conference on Robotics and Automation*, 2007.

[4] Ch Borst, M. Fischer, and G. Hirzinger. Grasping the dice by dicing the grasp. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS 2003).*, volume 4, pages 3692–3697, 2003.

[5] G. Bouchard and B. Triggs. Hierarchical part-based visual object categorization. In *Computer Vision and Pattern Recognition*, volume 1, pages 710–715, 2005.

[6] C. de Granville and A. H. Fagg. Learning grasp affordances through human demonstration. *submitted to the Journal of Autonomous Robots*, 2009.

[7] Charles de Granville, Joshua Southerland, and Andrew H. Fagg. Learning grasp affordances through human demonstration. In *Proceedings of the International Conference on Development and Learning (ICDL'06)*, 2006.

[8] Renaud Detry and Justus H. Piater. Hierarchical integration of local 3D features for probabilistic pose recovery. In *Robot Manipulation: Sensing and Adapting to the Real World (Workshop at Robotics, Science and Systems)*, 2007.

[9] Renaud Detry, Nicolas Pugeault, and Justus H. Piater. Probabilistic pose recovery using learned hierarchical object models. In *International Cognitive Vision Workshop (Workshop at the 6th International Conference on Vision Systems)*, 2008.

[10] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[11] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient matching of pictorial structures. In *Conference on Computer Vision and Pattern Recognition (CVPR 2000)*, pages 2066–, 2000.

[12] James J. Gibson. *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates, 1979.

[13] R.I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, 2000.

[14] D.H. Hubel and T.N. Wiesel. Anatomical demonstration of columns in the monkey striate cortex. *Nature*, 221:747–750, 1969.

[15] Kai Huebner, Steffen Ruthotto, and Danica Kragic. Minimum volume bounding box decomposition for shape approximation in robot grasping. Technical report, KTH, 2007.

[16] D. Kraft, N. Pugeault, E. Başeski, M. Popović, D. Kragic, S. Kalkan, F. Wörgötter, and 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*, 2008. (accepted).

[17] Danica Kragic, Andrew T. Miller, and Peter K. Allen. Real-time tracking meets online grasp planning. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 2460–2465, 2001.

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

[19] James Kuffner. Effective sampling and distance metrics for 3D rigid body path planning. In *Proc. 2004 IEEE Int'l Conf. on Robotics and Automation (ICRA 2004)*. IEEE, May 2004.

[20] T. S. Lee and D. Mumford. Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America*, pages 1434–1448, 7 2003.

[21] K. V. Mardia and P. E. Jupp. *Directional Statistics*. Wiley Series in Probability and Statistics. Wiley, 1999.

[22] A. T. Miller, S. Knoop, H. Christensen, and P. K. Allen. Automatic grasp planning using shape primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2003*, volume 2, pages 1824–1829, 2003.

[23] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[24] Mila Popovic, Dirk Kraft, Leon Bodenhagen, Emre Baseski, Nicolas Pugeault, Danica Kragic, and Norbert Krüger. A strategy for grasping unknown objects based on co-planarity and colour information. Submitted to RAS.

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

[26] Mario Richtsfeld and Markus Vincze. Robotic grasping based on laser range and stereo data. In *International Conference on Robotics and Automation*, 2009.

[27] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, 1986.

[28] Alexander Stoytchev. Toward learning the binding affordances of objects: A behavior-grounded approach. In *Proceedings of AAAI Symposium on Developmental Robotics*, pages 17–22, Stanford University, Mar 21-23 2005.

[29] Alexander Stoytchev. Learning the affordances of tools using a behavior-grounded approach. In E. Rome et al., editors, *Affordance-Based Robot Control*, volume 4760 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 140–158. Springer, Berlin / Heidelberg, 2008.

[30] Erik B. Sudderth, Alexander T. Ihler, William T. Freeman, and Alan S. Willsky. Nonparametric belief propagation. In *Computer Vision and Pattern Recognition*, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

Refining Grasp Affordance Models by Experience

Renaud Detry, Dirk Kraft, Anders Glent Buch, Norbert Krüger, Justus Piater

***Abstract*— We present a method for learning object grasp affordance models in 3D from experience, and demonstrate its applicability through extensive testing and evaluation on a realistic and largely autonomous platform. *Grasp affordance* refers to relative object-gripper configurations that yield stable grasps. These affordances are represented probabilistically with *grasp densities*, i.e. continuous density functions defined on the space of 3D positions and orientations. A grasp density characterizes an *object*'s grasp affordance; densities are linked to visual stimuli through registration with a visual model of the object they characterize. We explore a batch, experience-based learning paradigm where grasps sampled randomly from a density are performed, and an importance-sampling algorithm learns a refined density from the outcomes of these experiences. The first of such learning cycles is bootstrapped with a grasp density formed from visual cues. We show that the robot successfully exploits its actions into downweighting poor grasp solutions, which is reflected in the higher success rates achieved in subsequent learning cycles. We quantify the success rate of our method in a practical scenario where a robot needs to repeatedly grasp an object lying in an arbitrary pose, where each pose imposes a specific reaching constraint, and thus forces the robot to make use of the entire grasp density to select the most promising grasp within the set of achievable approaches.**

I. Introduction

In cognitive robotics, the concept of affordances [6], [14] characterizes the relations between an agent and its environment through the effects of the agent's actions on the environment. Affordances have become a popular formalization for cognitive control processes, while bringing valuable insight on how cognitive control can be done. Within the field of robotic grasping, methods formalized as *grasp affordances* have recently emerged [2], [21], [3], [11]. Grasp affordances generally allow for an assessment of the success (effect) of a grasp solution (action) on a particular object (environment).

Grasping skills can be programmed into a robot in many different ways, starting with completely hard-wired kinematics, and ranging over a wide variety of methods of increasing autonomy and adaptivity. Amongst these, providing a robot with the means of learning grasping skills *from experience* appears particularly appealing – even beyond the conveniently autonomous aspect of the process: First, in performing manipulation tasks, a robot produces valuable information characterizing its body and its environment, and making use of that information seems only natural. Secondly, learning from experience directly involves the body of the robot, therefore producing a model intimately adapted to its morphology.

A generally accepted aspect of the theory of affordances is that it relates the opportunities provided by the environment to the abilities of the agent, instead of expressing a property of the environment alone [17], [14]. Learning from experience thus appears as a natural way of discovering grasp affordances. The main contribution of this paper is the application of a method for learning grasp affordances probabilistically from experience [3] and its thorough evaluation. Evaluation is conducted on a realistic, largely autonomous platform, through the collection of a large grasp dataset – more than 2000 grasps tested on a robot.

In this work, affordances express relative object-gripper configurations that yield stable grasps. They are represented probabilistically with *grasp densities* [3], i.e. continuous density functions defined on the space of 3D positions and orientations $SE(3)$. A grasp density characterizes an *object*'s grasp affordance; densities are linked to visual stimuli through registration with a visual model of the object they characterize.

Grasp densities are learned and refined through experience. Intuitively, the robot "plays" with an object in a sequence of grasp-and-drop actions. Grasps are selected randomly from the object grasp density. After each grasp, the object is dropped to the floor. When a satisfying quantity of data has been accumulated, an importance-sampling algorithm [5] produces a refined grasp density from the outcomes of the set of executed grasps. Learning is thus organized in *cycles* of batches of grasps.

In theory, the grasp density of an object that has never been grasped could be initialized to a uniform distribution. Unfortunately, the success rate of completely random grasps is extremely low and cannot allow for reasonable learning time. In the experiments presented in this paper, initial grasp densities are bootstrapped from visual cues. The process that produces grasp densities from a visual model of an object is kept simple, limiting the amount of bias introduced in the system. Throughout the paper, densities that are constructed from visual cues will be called *bootstrap densities*. Conversely, densities which are the result of experience will be referred to as *empirical densities*. Within each learning cycle, the density used by the robot to produce grasps will be called *hypothesis density*. In the first cycle, the hypothesis density is a bootstrap density. In subsequent cycles, the hypothesis density will typically correspond to the empirical density learned during the previous cycle.

Experiments are run on the robotic platform of Fig. 1. A simple control algorithm drives the grasp-and-drop protocol

R. Detry and J. Piater are with the University of Liège, Belgium. Email: Renaud.Detry@ULg.ac.be.

D. Kraft, A. Glent Buch and N. Krüger are with the University of Southern Denmark.

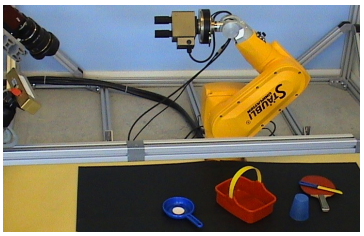


Fig. 1: Experiment platform (industrial arm, force-torque sensor, 2-finger gripper, stereo camera, foam floor).

on the robot. The pose of the object is recovered by visual pose estimation on the imagery provided by the camera, using a previously-learned visual model [4]. Path planning automatically excludes most of the grasps that would produce a collision with the ground, and some of the grasps that would collide with the object. Success is assessed by measuring the distance between the fingers of the gripper after the arm has been brought up. The resulting system is largely autonomous and forms a realistic setup. We show that the robot successfully exploits its actions to downweight poor grasp solutions, which is reflected in the higher success rates achieved in subsequent learning cycles. We finally quantify the success rate of our method in a practical scenario where a robot needs to repeatedly grasp an object lying in an arbitrary pose, where each pose imposes specific reaching constraints, and thus forces the robot to make use of the entire grasp density to select the most promising grasp within the achievable region.

II. Related Work

As discussed above, learning grasp affordances from experience has become an interesting and popular paradigm. In the work of A. Stoytchev [18], [19], a robot discovers successful ways of grasping tools through random exploratory actions. When subsequently confronted with the same object, the robot is able to generate a grasp that should present a high likelihood of success. Montesano et al. [11] learned 2D continuous and probabilistic grasp affordance models for a set of objects of varying shape and appearance, and developed means of qualifying the reliability of their grasp predictions. Detry et al. [3] presented a method for learning continuous and probabilistic grasp affordance models in 3D along with preliminary experimental results.

We note that the main problem with learning from experience is that it is usually slow. The main alternative to learning from experience is learning from a human teacher [2], [15], which is typically much faster.

A large body of literature on learning how to grasp focuses on methods that produce a number of *discrete* grasping solutions [15], [1]. A few recent methods instead explicitly aim at producing a *continuous*, probabilistic characterization of the grasping properties of an object [2], [3], [11]. The latter can naturally be used to produce grasping solutions; additionally, they allow for *ranking* grasps, i.e. provide a likelihood of success for an arbitrary grasp.

In learning a continuous characterization of object grasping properties probabilistically, one has a choice between learning success probabilities or learning success-conditional grasp densities. Denoting by O a random variable encoding grasp outcomes (success or failure), and by G a random variable encoding grasp poses, this translates to learning $\mathbf{p}(O|G)$ or learning $\mathbf{p}(G|O)$. The former allows one to directly compute a probability of success; it will generally be learned through discriminative methods from positive and negative examples (successful and failed grasps). The latter allows for grasp sampling, while still providing direct means of computing *relative* success probabilities – e.g. grasp a is twice as likely to succeed as grasp b. Success-conditional grasp densities are generative models computed from positive examples only. We note that one can theoretically be computed from the other using Bayes' rule. However, depending on the means of function representation, this process may prove either too costly or too noisy to be feasible in practice.

The learning of success-conditional grasp densities has been discussed in the work of de Granville et al. [2], where grasp densities are defined on hand approach orientations. Instead of considering success-conditional grasp probabilities, Montesano et al. [11] model grasp affordances as success probabilities. Formally, they learn a representation of $\mathbf{p}(O|I)$, where I is a local image patch. A grasp action consists in servoing the robot hand to a selected 2D position, approaching the object from the top until contact is made, and closing the hand. A robot thus learns a mapping from 2D image patches to grasp success probabilities, where a grasp is parametrized by its 2D hand position.

The most important application of grasping research is in generating a grasping solution from visual percepts of an object. Grasping research may thus be pertinently classified on the relationship a method entertains with visual perceptions. In the field of robotics, *visual perception* encompasses to a wide spectrum of representations: At the lower level, a scene may be described in terms of a large number of point elements, such as image pixels or depth maps. At the other end of the spectrum, a scene may be represented by instances of object models and their 2D or 3D poses. The gap between these two extremes is filled, bottom-up, by visual features of varying size and complexity, and, top-down, by object models that are recursively formed of visual parts of decreasing size and complexity. Intuitively, grasping methods that link to lower-level visual percepts can easily generalize across objects. These methods typically learn a continuous mapping from local visual descriptors to the probability of success of a grasp [11]. Grasp parameters are deduced from the visual descriptor, e.g. 2D grasping coordinates from the 2D position of the descriptor [11], or 3D grasp position from stereo matching of 2D grasping points [15]. On the other hand, methods that link to higher-level visual entities benefit from an increased geometric robustness. These will generally allow the encoding of richer grasp parameters such as 3D relative position and orientation. They typically learn a mapping from objects to grasp parameters and grasp probabilities [2], [3]; grasps are registered with the visual

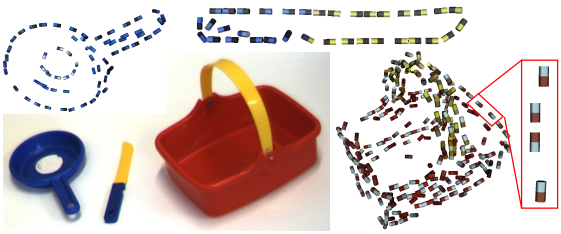


Fig. 2: ECV (accumulated) reconstructions. Each cylinder corresponds to an ECV descriptor. The axis of a cylinder is aligned with the direction of the modeled edge. Each cylinder bears the two colors found on both sides of the edge in 2D images.

object model. They are aligned to an object pose through visual pose estimation.

In this paper, we develop and evaluate a method for learning by experience continuous 3-dimensional success-conditional grasp densities, initially presented by Detry et al. [3]. Densities encode object grasps; they are linked to a high-level object model. The contribution of this paper include original means of bootstrapping densities, a technique for exploiting local density maxima, and a thorough evaluation of the resulting system through a realistic robot setup and scenario. By that, we can demonstrate the full potential of the concept of grasp densities in a setting on which a motion planner interacts with grasp densities. In particular we show that besides increasing the success rate for physically executed grasps, the amount of grasps to be tested by the motion planner reduces by a factor of 10.

III. Vision

This section briefly presents the vision methods used in this work. We first introduce 3D object-edge reconstructions which are used in Section V for bootstrapping densities. These reconstructions also serve as a basis for a hierarchical object model introduced next.

A. Early Cognitive Vision

ECV descriptors [9], [13] represent short edge segments in 3D space, each ECV descriptor corresponding to a circular image patch with a 7-pixel diameter. They are computed by combining 2D edge extraction in image pairs and stereopsis across the pairs. Each descriptor is defined by a 3D position and 3D edge orientation, therefore living in $\mathbb{R}^3 \times S^2_+$ where S^2_+ is a 2-hemisphere. Descriptors may be tagged with color information, extracted from their corresponding 2D patches (Fig. 2).

ECV reconstructions can further be improved by manipulating objects with a robot arm, and *accumulating* visual information across several views through structure-from-motion techniques [7]. Assuming that the motion adequately spans the object pose space, a complete 3D reconstruction of the object can be generated, eliminating self-occlusion issues [8] (see Fig. 2).

B. Pose Estimation

Visual models have the form of a hierarchy of increasingly expressive object parts [4], where bottom-level parts correspond to generic ECV descriptors. Visual inference of the hierarchical model is performed using a belief propagation algorithm (BP) [12], [20], [4]. BP derives a probabilistic estimate of the object pose, which in turn allows the alignment of the grasp model to the object. Means of autonomously learning of the hierarchical model and the underlying accumulated ECV reconstruction are presented in the work of Detry et al. [4] and Kraft et al. [8].

IV. Grasp Densities

This section explains how grasp densities probabilistically model grasp affordances, and how importance sampling is used to learn empirical densities.

A. Mathematical Representation

We are interested in modeling object-relative gripper configurations. The grasps we consider are thus parametrized by a 6D gripper pose composed of a 3D position and a 3D orientation, and grasp densities are continuous probability density functions defined on the 6D pose space $SE(3)$. Their computational representation is nonparametric: A density is represented by a large number of weighted samples called *particles*. The probabilistic density in a region of space is given by the local density of the particles in that region. The underlying continuous density function is accessed through *kernel density estimation* [16], by assigning a kernel function to each particle supporting the density. Density evaluation at a given pose x is performed by summing the evaluation of all kernels at x. Sampling from a distribution is performed by sampling from the kernel of a particle drawn at random.

Grasp densities are defined on the Special Euclidean group $SE(3) = \mathbb{R}^3 \times SO(3)$, where $SO(3)$ is the Special Orthogonal group (the group of 3D rotations). We use a kernel that factorizes into two functions defined on $\mathbb{R}^3$ and $SO(3)$. Denoting the separation of an $SE(3)$ point x into a translation λ and a rotation θ by

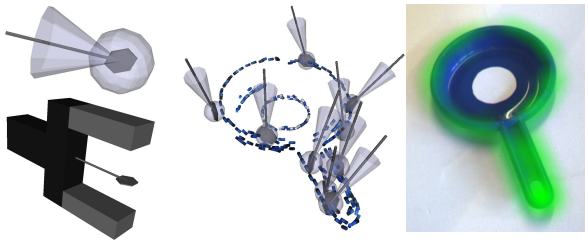
$$x = (\lambda, \theta), \quad \mu = (\mu_t, \mu_r), \quad \sigma = (\sigma_t, \sigma_r),$$

we define our kernel with

$$\mathbf{K}(x; \mu, \sigma) = \mathbf{N}(\lambda; \mu_t, \sigma_t)\, \mathbf{W}(\theta; \mu_r, \sigma_r) \qquad (1)$$

where μ is the kernel mean point, σ is the kernel bandwidth, $\mathbf{N}(\cdot)$ is a trivariate isotropic Gaussian kernel, and $\mathbf{W}(\cdot)$ is the Dimroth-Watson kernel [10], which corresponds to a Gaussian-like distribution on $SO(3)$. The position bandwidth σ_t is fixed to 10mm; the orientation bandwidth σ_r allows rotations of about $10°$. For a more detailed mathematical description and motivation of $SE(3)$ kernels and kernel density estimation, we refer the reader to the work of Sudderth et al. [20] and Detry et al. [4]. Fig. 3 illustrates $SE(3)$ kernels and continuous densities.

Grasp densities are defined in the same reference frame as visual features. Once visual features have been aligned to an object pose (Section III-B), the object grasp density can be



(a) (b) (c)

Fig. 3: Grasp density representation. The top image of Fig. (a) illustrates a particle from a nonparametric grasp density, and its associated kernel widths: the translucent sphere shows one position standard deviation, the cone shows the variance in orientation. The bottom image illustrates how the schematic rendering used in the top image relates to a physical gripper. Fig. (b) shows a 3D rendering of the kernels supporting a grasp density for a toy pan (for clarity, only 10 kernels are rendered). Fig. (c) indicates with a green mask of varying opacity the values of the location component of the same grasp density along the plane of the pan (orientations were ignored to produce this last illustration).

similarly aligned, and one can readily draw grasps from the density and execute them onto the object. The association of grasp densities with the visual model is covered in more detail in Detry et al. [3].

B. Learning Algorithm

Learning is organized in cycles, within each of which the robot exploits its current grasping knowledge through importance sampling [5] to produce a refined empirical density. Importance sampling is a technique that allows one to draw samples from an unknown *target* distribution by properly weighting samples from a preferably similar *proposal* distribution. The target distribution $\mathbf{t}(X)$ cannot be sampled from, but it can be evaluated. Therefore, samples are drawn from the known proposal distribution $\mathbf{p}(X)$, and the difference between the target and the proposal is accounted for by associating to each sample x a weight given by $\mathbf{t}(x) \,/\, \mathbf{p}(x)$.

Let us model with $g(x)$ the outcome of grasp x as

$$g(x) = \begin{cases} 1 & \text{if grasp at } x \text{ succeeds} \\ 0 & \text{if grasp at } x \text{ fails} \end{cases}$$

In the context of this paper, the target distribution corresponds to the object grasp affordance $\mathbf{a}(X)$. We would need to sample from $\mathbf{a}(X)$ in order to build a model of it, which unfortunately cannot be done. However, by approximating $\mathbf{a}(x) \simeq g(x)$, we can produce binary evaluations of an object grasp affordance by executing grasps. Importance sampling thus allows us to indirectly draw samples from $\mathbf{a}(X)$; the importance sampling proposal corresponds to the hypothesis density $\mathbf{h}(X)$, and the weight of each sample x is given by $g(x)/\,\mathbf{h}(x)$. Fig. 4 illustrates the effect of importance sampling in a simple 1-dimensional case.

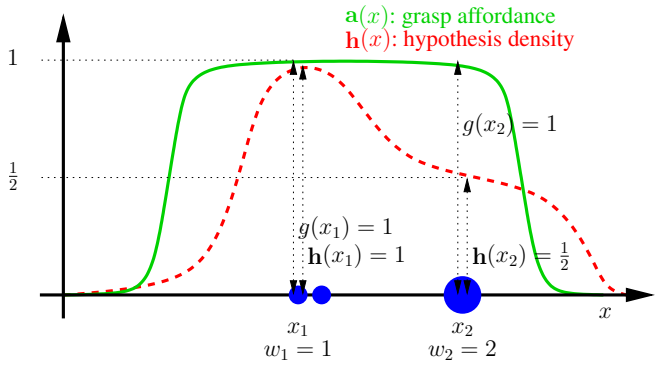


Fig. 4: Importance sampling weight computation. Although grasps such as x_2 are less likely to be executed than grasps like x_1, the weight associated to x_2 is twice as large as that associated to x_1.

V. Experiments

This section demonstrates the applicability of our method to learning empirical densities, quantifies the efficacy of the various learning cycles, and estimates the efficacy of empirical densities in a typical grasping scenario.

Section V-A explains the process of executing a set of grasp *trials*, and details the nature of recorded data. Section V-B presents the application of this process for both learning empirical densities and estimating their efficacy in practical scenarios. Results are discussed in Section V-C.

A. Setup

Our robotic platform is composed of an industrial robotic arm, a force-torque sensor, a two-finger gripper, and a stereo camera. The force-torque sensor is mounted between the arm and the gripper. The arm and the camera are calibrated to a common world reference frame. The execution of a set of grasp trials is driven by a finite state machine (FSM), which instructs the robot to grasp and lift an object, then drop the object to the floor and start again. The floor around the robot is covered with foam, which allows objects to lightly bounce during drop-off. This also allows the gripper to push slightly into the floor and grasp thin objects lying on the foam surface (e.g. the knife of Fig. 2). The FSM is initially provided with an object model, which includes a visual model as described in Section III-B, and a grasp density registered with the visual model. The FSM then performs a set of grasp trials, which involve the following operations:

i. Estimate the pose of the object and align the grasp density;
ii. Produce a grasp from the aligned grasp density;
iii. Submit the grasp to the path planner;
iv. Servo the gripper to the grasp pose;
v. Close the gripper fingers;
vi. Lift the object;
vii. Drop the object.

Pose estimation (i) is performed by means detailed in Section III-A and Section III-B. Depending on the purpose,

grasps (ii) are drawn either randomly, or from a local maximum of the density.

The path planner has a built-in representation of the floor and robot body. Its representation of the floor is defined a few centimeters below the foam surface, to allow the gripper to grasp thin objects as explained above. The planner is provided with a gripper pose (ii) and the 3D scene reconstruction extracted during pose estimation (i). Because the 3D scene reconstruction doesn't cover self-occluded parts of the object, a 3D *accumulated* reconstruction of the object is also provided (Section III-A). The path planner computes a collision-free path to the target gripper configuration. It can avoid self-collisions and most ground-collisions from its built-in knowledge of the arm and workspace; it can also avoid some object collisions from the 3D-edge scene reconstruction and the aligned object reconstruction. When no path can be found, the path planner is able to produce a detailed error report.

During servoing (iv) and grasping (v), measures from the force-torque sensor are compared to a model of the arm dynamics, allowing for automatic collision detection. Closure success is verified after grasping (v) by measuring the gap between the fingers, and after lifting (vi) by checking that the fingers cannot be brought closer to each other. The object is finally dropped to the floor from a height of about 50cm and bounces to an arbitrary pose.

Robot assessments are monitored by a human supervisor. Pose estimation will sometime fail, e.g. because the object fell out of the field of view of the camera, or because the stereo signal from the object is too poor to produce an ECV reconstruction. Pose estimates are thus visualized in 3D; if pose estimation fails, the trial is aborted and the supervisor moves the object to another arbitrary pose. After path planning, the supervisor has a chance to abort a grasp that should clearly fail. During servo, grasp and lift, he can notify undetected collisions. Despite this supervision, the resulting system is largely autonomous: The role of the supervisor is limited to notifying wrong robot assessments; pose estimates and grasps are never tuned by hand.

If the robot properly executes the operations mentioned above and lifts the object, the trial is a success. When an operation produces an error, the trial is a failure, and the FSM starts over at step ii, or at step i if the error involved an object displacement. Errors can come from a pose estimation failure, no found path, supervisor notification of bound-to-fail grasp, collision (notified either from the force-torque sensor or from the supervisor), or empty gripper (v and vi). We define two mutually-exclusive error classes. The first class, denoted by E_p, includes errors arising from a path-planner–predicted collision with the ground or the object. The second class, E_r, correspond to object collisions, ground collisions, or void grasps, either asserted by the supervisor, or physically occurring on the robot. Errors E_r also include cases where the object drops off the gripper during lift-up. The FSM keeps track of errors by counting the number of occurrences e_r and e_p of errors of class E_r and E_p. Pose estimation failures and cases where the path planner cannot find an inverse-kinematics solution at all (e.g. object out of reach) are ignored because these are not intrinsically part of the concept of grasp densities. Naturally, the number s of successful grasps is also recorded.

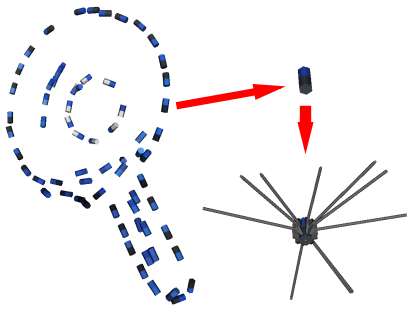


Fig. 5: Bootstrapping grasp densities from ECV descriptors.

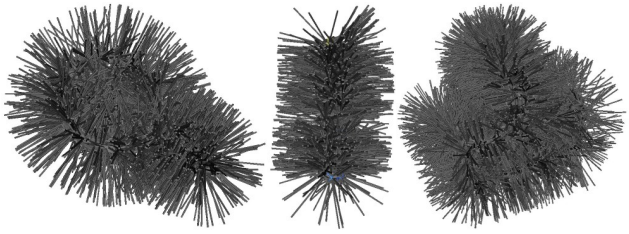


Fig. 6: Particles supporting bootstrap densities.

The execution of a complete grasp trial typically takes 40 to 60 seconds. Through the process described above, the robot will effectively learn grasp affordances offered by an object lying on a flat surface in a natural pose.

B. Protocol

This section applies the process of the previous section to learn and evaluate empirical densities. Experiments were run on the three objects of Fig. 2, selected for their differences in shape and structure, which offer a large variety of grasping possibilities. Visual models were acquired by performing a 3D reconstruction of the object edges (Section III-A), and organizing the resulting ECV descriptors in a hierarchy (Section III-B).

Grasp densities were bootstrapped from the ECV reconstructions of the objects. As explained in Section III-A, an ECV reconstruction represents object edges with short 3D segments. Object edges appeared as natural candidates for grasping, and an interesting way to bias grasp learning. We thus define bootstrap densities as functions yielding a high value around object edges. Bootstrap densities are, just like other densities, represented nonparametrically. They are formed by generating sets of $SE(3)$ particles from ECV descriptors. Mathematically, ECV descriptors live in $\mathbb{R}^3 \times S^2_+$; an ECV descriptor thus cannot fully define an $SE(3)$ grasp. We thus create a *set* of $SE(3)$ particles for each ECV descriptor, effectively covering the ECV orientation degree of freedom (See Fig. 5 and Fig. 6).

To each object of Fig. 2, we applied two learning cycles. In the first cycle, the robot uses the object bootstrap density b as hypothesis to learn an empirical density g_1. In the second

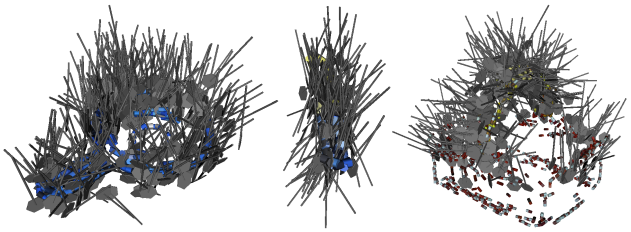


Fig. 7: Samples from empirical densities learned during the first cycle.

Batch		s	e_r	e_p	r_{rp}	r_r
Pan	*cycle 1*	200	370	1631	0.091	0.351
	cycle 2	100	86	114	0.333	0.538
	test	75	39	24	0.543	0.658
Knife	*cycle 1*	100	131	751	0.102	0.433
	cycle 2	100	153	157	0.244	0.395
	test	63	71	89	0.283	0.470
Basket	*cycle 1*	151	173	1121	0.104	0.466
	cycle 2	100	62	77	0.418	0.617
	test	64	26	22	0.571	0.711

TABLE I: Success/error counts and success rates. (See also Fig. 8.)

cycle, the hypothesis density corresponds to g_1, and the robot learns a second empirical density g_2. The purpose of the second cycle is to provide a quantitative comparison of the grasping knowledge expressed by bootstrap and empirical densities through the error statistics of both processes; g_2 is not used thereafter.

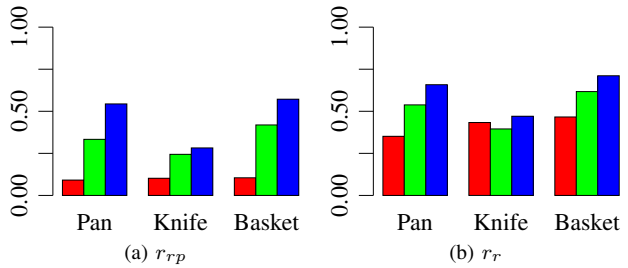
We tested the performance of our method in a usage scenario in which it has to successively allow a robot to perform the grasp that has the highest likelihood of success within the robot's region of reachability. However, expressing the region of $SE(3)$ that the robot can reach to is not trivial, and goes beyond the scope of this paper. Our usage scenario thus implements each grasp trial by randomly drawing a set of grasps from an empirical density, and sorting these grasps in decreasing order of likelihood according to that empirical density. The grasps are sequentially submitted to the path planner and the first feasible grasp is selected. The empirical density used in the usage scenario is g_1, in order to provide a direct comparison with the statistics collected during the second learning cycle.

C. Results and Discussion

The empirical densities produced during the first learning cycle are shown in Fig. 7. Comprehensive quantitative results are displayed in Table I. Columns s, e_r, and e_p correspond to the statistics collected during the experiment. The last two columns show success rates computed as

$$r_{rp} = \frac{s}{s + e_r + e_p}, \qquad r_r = \frac{s}{s + e_r}.$$

Rows titled *cycle 1* and *cycle 2* correspond to the first and second learning cycles. Rows titled *test* correspond to the



(a) r_{rp} (b) r_r

Fig. 8: Success rates. Red, green, and blue bars respectively illustrate rates for the first cycle, second cycle, and test. Numerical rates can be read from Table I.

usage scenario defined above. Fig. 8 shows success rates graphically.

Fig. 7 shows that the empirical densities learned in the first cycle are a much better model of grasp affordances than the bootstrap densities of Fig. 6. The global success rate r_{rp} (Fig. 8a) for the two learning cycles provides a quantitative comparison of the grasping knowledge expressed by bootstrap and empirical densities. The empirical densities produced during the first cycle allow the robot to collect, during the second cycle, a similar amount of positive examples with a much smaller number of trials. The red bars in Fig. 8a show that grasps generated from modes of an empirical indeed have a higher chance of success than randomly sampled grasps.

Fig. 8b shows success rates in which planner-detected errors E_p are ignored. Since planner-detected errors largely amount to ground-collisions, Fig. 8b shows that a large portion of the knowledge acquired by the robot models which side of the object most often faces up, hence encouraging the robot to produce grasps approaching to that side. This situation is pushed to the limit with the knife: All grasps suggested by its bootstrap density would effectively work for a free-floating knife, i.e. all grasps that do not collide with the ground have the same chance of success. When ignoring E_p errors, the success rate for the first and second cycles of the knife are almost identical.

Our results make a number of issues explicit. For all objects we can reduce the number of grasps that need to be considered by the motion planner by an average factor of 10. This is an important result, since path planning is generally slow, and ground plane information may not always be available to the planner. The average success rate of grasps performed by the robot grows from 42% to 52%. In test scenarios, the success rate of robot grasps is in average 61%. These numbers are quite encouraging, given that we tested our system under realistic settings: Visual models, which are learned autonomously [4], [8], do not exhaustively encode relevant object features. During pose estimation, estimates that are considered successful are nevertheless affected by errors of the order of 5–10mm in position and a few degrees in orientation. The path planner approximates obstacles with box constellations that may often be imprecise and over-restrictive. Inverse kinematics can perform only up to the

precision of robot-camera calibration. When grasping near the floor, the force-torque sensor may issue a collision detection for a grasp that has worked before, because of a different approach dynamic. For the pan, and in particular for the knife, we have a very difficult grasping situation, given the short distance between the object and the ground. As a consequence, small errors in pose estimates can lead to collisions even with an optimal grasp. Therefore, the error counts in Table I do not exclusively reflect issues related to grasp densities. We showed that complete sets of grasp affordances can be acquired by largely autonomous learning. The concept of grasp densities served as a powerful tool to represent these affordances which can be used to find an optimal grasp in a concrete context.

VI. Conclusion And Future Work

We have presented a method for learning *3-dimensional* probabilistic grasp affordance models in the form of grasp densities, and demonstrated their applicability through extensive testing on a largely autonomous platform.

Grasp densities are learned from experience with an importance-sampling algorithm: samples from an object affordance are indirectly drawn by properly weighting samples from an approximating hypothesis density. Densities are bootstrapped from a 3D object-edge reconstructions, yielding bias towards edge grasps.

We assembled an experiment setup which efficiently implements a realistic learning environment: The robot handles objects appearing in arbitrary poses, and deals with the noise inherent to autonomous processes. We have collected a large amount of data quantifying the progress made from bootstrap to empirical densities. We also evaluated empirical densities in a realistic usage scenario, where the robot effectively selects the grasp with the highest success likelihood amongst the grasps that are within its reach. Result are particularly convincing given the low level of control on the overall experiment process.

In this paper, grasp densities characterize *object* grasps; they are registered with a visual model of the object. Yet, affordances generally characterize object-robot relations through a minimal set of properties, which means that object properties not essential to a relation should be left out. This in turn allows e.g. for generalization of affordances between objects that share the same grasp-relevant features. Ultimately, instead of registering densities with a whole object, we aim to relate them to visual object *parts* that predict their applicability. The part-based model of Section III-B offers an elegant way of *locally* encoding visuomotor descriptions, allowing for generalization of grasps across objects that share the same parts.

Acknowledgments

This work was supported by the Belgian National Fund for Scientific Research (FNRS) and the EU Cognitive Systems project PACO-PLUS (IST-FP6-IP-027657).

References

[1] A. Bicchi and V. Kumar. Robotic grasping and contact: a review. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, 1, 2000.

[2] Charles de Granville, Joshua Southerland, and Andrew H. Fagg. Learning grasp affordances through human demonstration. In *International Conference on Development and Learning*, 2006.

[3] Renaud Detry, Emre Başeski, Norbert Krüger, Mila Popović, Younes Touati, Oliver Kroemer, Jan Peters, and Justus Piater. Learning object-specific grasp affordance densities. In *International Conference on Development and Learning*, 2009.

[4] Renaud Detry, Nicolas Pugeault, and Justus Piater. A probabilistic framework for 3D visual object representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2009.

[5] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[6] James J. Gibson. *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates, 1979.

[7] R.I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[8] D. Kraft, N. Pugeault, E. Başeski, M. Popović, D. Kragic, S. Kalkan, F. Wörgötter, and N. Krüger. Birth of the object: Detection of objectness and extraction of object shape through object action complexes. *International Journal of Humanoid Robotics*, 5:247–265, 2009.

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

[10] K. V. Mardia and P. E. Jupp. *Directional Statistics*. Wiley Series in Probability and Statistics. Wiley, 1999.

[11] L. Montesano and M. Lopes. Learning grasping affordances from local visual descriptors. In *International Conference on Development and Learning*, 2009.

[12] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

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

[14] Erol Sahin, Maya Cakmak, Mehmet R. Dogar, Emre Ugur, and Gokturk Ucoluk. To afford or not to afford: A new formalization of affordances towards affordance-based robot control. *Adaptive Behavior*, 2007.

[15] A. Saxena, J. Driemeyer, and A. Y. Ng. Robotic Grasping of Novel Objects using Vision. *The International Journal of Robotics Research*, 27(2):157, 2008.

[16] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, 1986.

[17] T. Stoffregen. Affordances as properties of the animal environment system. *Ecological Psychology*, 15(2):115–134, 2003.

[18] Alexander Stoytchev. Toward learning the binding affordances of objects: A behavior-grounded approach. In *Proceedings of AAAI Symposium on Developmental Robotics*, pages 17–22, Stanford University, Mar 21-23 2005.

[19] Alexander Stoytchev. Learning the affordances of tools using a behavior-grounded approach. In E. Rome et al., editors, *Affordance-Based Robot Control*, volume 4760 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 140–158. Springer, Berlin / Heidelberg, 2008.

[20] Erik B. Sudderth. *Graphical models for visual object recognition and tracking*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006.

[21] J.D. Sweeney and R. Grupen. A model of shared grasp affordances from demonstration. In *International Conference on Humanoid Robots*, 2007.

Learning Objects and Grasp Affordances through Autonomous Exploration

Dirk Kraft[1], Renaud Detry[2], Nicolas Pugeault[1], Emre Başeski[1], Justus Piater[2], and Norbert Krüger[1]

[1] University of Southern Denmark, Denmark.
{kraft,nicolas,emre,norbert}@mmmi.sdu.dk
[2] University of Liège, Belgium.
{Renaud.Detry,Justus.Piater}@ULg.ac.be

Abstract. We describe a system for autonomous learning of visual object representations and their grasp affordances on a robot-vision system. It segments objects by grasping and moving 3D scene features, and creates probabilistic visual representations for object detection, recognition and pose estimation, which are then augmented by continuous characterizations of grasp affordances generated through biased, random exploration. Thus, based on a careful balance of generic prior knowledge encoded in (1) the embodiment of the system, (2) a vision system extracting structurally rich information from stereo image sequences as well as (3) a number of built-in behavioral modules on the one hand, and autonomous exploration on the other hand, the system is able to generate object and grasping knowledge through interaction with its environment.

1 Introduction

We describe a robot vision system that is able to autonomously learn visual object representations and their grasp affordances. Learning takes place without external supervision; rather, the combination of a number of behaviors implements a bootstrapping process that results in the generation of object and grasping knowledge.

Learning of objects and affordances has to address a number of sub-aspects related to the object aspect of the problem (**O1**–**O3**) and to the action aspect (**A1**, **A2**):

O1 What is an object, i.e., what is "objectness"?
O2 How to compute relevant attributes (shape and appearance) to be memorized?
O3 How can the object be recognized and how can its pose determined?
A1 What is the (preferably complete) set of actions it affords?
A2 What action is triggered in a concrete situation?

A satisfactory answer to **O1** is given by Gibson [1] as temporal permanence, manipulability and constrained size in comparison to the agent. Note that manipulability can only be tested by acting on the potential object, and hence requires an agent with at least minimal abilities to act upon objects. For **O2** there are requirements discussed in the vision literature. In many systems, in particular in the context of robotics, the object shape is given a priori by a CAD representation and is then used for object identification and pose estimation (see, e.g., Lowe [2]). However, CAD representations are not

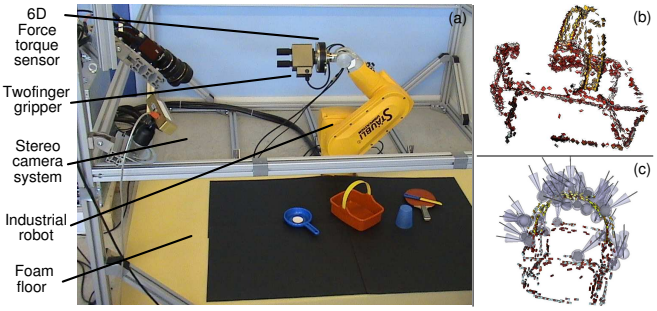


Fig. 1. (a) Hardware setup. **(b, c)** Outcome of the learning process in form of a geometric object model **(b)** and a grasp density **(c)**.

available in a general context and hence for a cognitive system, it is important that it is able to learn object representations from experience. Important issues to be considered when coding objects are that the information memorized (1) is useful for the tasks to be performed with the representations (e.g., for matching or grasping), (2) is efficiently accessible internally, and (3) requires little storage space. **O3** has been addressed widely in the computer vision literature. In particular in the context of grasping, besides the actual recognition, the determination of the pose is of importance since it allows the system to associate learned grasps in object coordinates to an observes object instance.

In cognitive robotics, the automatic association of grasping actions to objects (**A1**, **A2**) is referred as learning *affordances* [3]. For maximum flexibility, it is desirable to represent the set of grasp affordances to the most complete extent possible **A1**. There are attempts to compute such a complete set by analytic means [4] which however in general require a pre-existing 3D surface model. In addition, analytic modeling of the interaction between a gripper and an object surface, besides being very time consuming, is very complex since it involves for example friction parameters that are difficult to estimate. Hence we decided to achieve such knowledge by letting the robot experiment in the real world. The decision on the grasp to be performed in a given situation **A2** involves additional considerations, in particular work-space constraints.

This paper describes a system that approaches the above problems in a way that does not require any explicit prior object or affordance knowledge. Instead, the system generates object and grasping knowledge by pure exploration (see Fig. 1). We present a robot-vision system driven by a number of basic behaviors that generate object and grasp-affordance knowledge within two learning cycles. In the first cycle, a multi-modal visual representation covering geometric as well as appearance information (see Fig. 2) is extracted by actively manipulating a potential object. In the second cycle, the robot "plays" with the object by trying out various grasping options. Successful grasp parameters are associated to the object model, leading to an increasingly complete description of the object's grasp affordance. This is done by largely autonomous exploration with only very little interaction between robot and humans. Only interaction that puts the system into a state from which learning can continue is permitted (e.g., putting the ob-

ject back after playing has pushed it out of the workspace). No high level information such as object identities or demonstrations of ways to grasp it is given to the system.

However, this is not to imply that the system does not make use of any prior knowledge. Quite to the contrary, the system can only perform the complex learning tasks by utilizing a large degree of innate knowledge about the world with which it interacts. However, this knowledge is of rather generic structure. Specifically, the system

- has knowledge about its embodiment and the consequences of its movements in the three-dimensional world (kinematics and the ability to plan motions),
- has a sophisticated early cognitive vision (ECV) system [5–7] that provides semantically rich and structured 2D and 3D information about the world. This system contains prior knowledge about image features and their relations.
- has a set of procedural prior knowledge about how to: *a)* grasp unknown objects based on visual features, *b)* create visual object models based on object motion, *c)* evaluate a grasping attempt, *d)* estimate object pose based on a learned visual model and *e)* generalize from individual grasps to grasping densities.

This paper describes the various sub-modules and their interaction that lead to the autonomous learning of objects and associated grasp affordances. We show that, based on a careful balance of generic prior knowledge and exploratory learning, the system is able to generate object and grasping knowledge while exploring the world it acts upon. While the sub-modules have already been described [7–12], the novel part of this work is the integration of these components into an autonomously learning system.

2 State of the Art

Concerning the aspects **O1**–**O3**, the work of Fitzpatrick and Metta [13] is closely related to our object learning approach since the overall goal as well as the hardware setup are similar: finding out about the relations of actions and objects by exploration using a stereo system combined with a grasping device. We see the main distinguishing feature of this work to our approach in the amount of prior structure we use. For example, we assume a much more sophisticated vision system. Also, the use of an industrial robot allows for a precise generation of scene changes exploited for the extraction of the 3D shape of the object. Similar to this work, we initially assume "reflex-like" actions that trigger exploration. However, since in our system the robot knows about its body and about the 3D geometry of the world and since the arm can be controlled more precisely, we can infer more information from having physical control over the object in terms of an exact association of visual entities based on proprioceptive information. Therefore, we can learn a complete 3D representation of the object (instead of 2D appearance models) that can then be linked to pose estimation. Modayil and Kuipers [14] addressed the problem of detection of objectness and the extraction of object shape in the context of a mobile robot using laser. Here also motion information (in terms of the odometry of the mobile robot) is used to formulate predictions. In this way, they can to extract a 2D cross section of the 3D environment, albeit only in terms of geometric information.

Object grasp affordances (**A1**, **A2**) can emerge in different ways. A popular approach is to compute grasping solutions from the geometric properties of an object,

typically obtained from a 3D object model. The most popular 3D model for grasping is probably the 3D mesh [4], obtained e.g. from CAD or superquadric fitting [15]. However, grasping has also successfully been achieved using models consisting of 3D surface patches [16], 3D edge segments [8,12], or 3D points [17]. When combined with pose estimation, such methods allow a robot to execute a grasp on a specific object. In our system, we start with edge-based triggering of grasping actions [8,12] which is then verified by empirical exploration. This requires a system that is able to perform a large number of actions (of which many will likely fail) in a relatively unconstrained environment, this requires a representation of grasp affordances that translate the grasping attempts into a probabilistic statement about grasp success likelihoods.

Learning grasp affordances from experience was demonstrated by Stoytchev [18, 19]. In this work, a robot discovers successful grasps through random exploratory actions on a given object. When subsequently confronted with the same object, the robot is able to generate a grasp that should present a high likelihood of success.

Means of representing continuous grasp affordances have been discussed by de Granville et al. [20]. In their work, affordances correspond to object-relative hand approach orientations, although an extension is underway where object-relative positions are also modeled [21].

3 The Robot-Vision System

Hardware setup: The hardware setup (see Fig. 1) used for this work consists of a six-degree-of-freedom industrial robot arm (Stäubli RX60) with a force/torque (FT) sensor (Schunk FTACL 50-80) and a two-finger-parallel gripper (Schunk PG 70) attached. The FT sensor is mounted between robot arm and gripper and is used to compute to detect collision. Together with the foam ground, this permits graceful reactions to collision situations which might occur because of limited knowledge about the objects in the scene. In addition, a calibrated stereo camera system is mounted in a fixed position in the scene. The system also makes use of a path-planning module which allows it to verify the feasibility of grasps with respect to workspace constraints and 3D structure discovered by the vision system.

Early cognitive vision system: In this work, we make use of the visual representation delivered by an early cognitive vision system [5–7]. Sparse 2D and 3D features, so-called *multi-modal primitives*, are created along image contours. 2D features represent a small image patch in terms of position, orientation, phase. These are matched across two stereo views, and pairs of corresponding 2D features permit the reconstruction of a 3D equivalent. 2D and 3D primitives are organized into perceptual groups in 2D and 3D (called 2D and 3D contours in the following). The procedure to create visual representations is illustrated in Fig. 2 on an example stereo image pair. Note that the resultant representation not only contains appearance (e.g., color and phase) but also geometrical information (i.e., 2D and 3D position and orientation).

The sparse and symbolic nature of the multi-modal primitives allows for the coding of relevant perceptual structures that express relevant spatial relations in 2D and 3D [22]. Similar relations are also defined for 2D and 3D contours to enable more

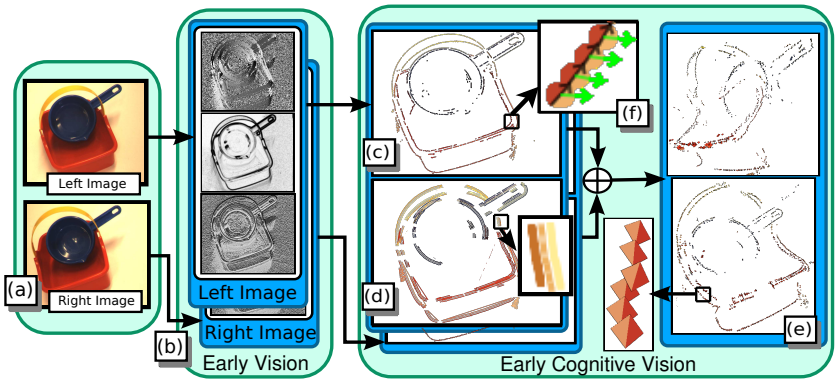


Fig. 2. An overview of the visual representation. **(a)** Stereo image pair, **(b)** Filter responses, **(c)** 2D primitives, **(d)** 2D contours, **(e)** 3D primitives, **(f)** close-up of (c).

global reasoning processes. In our context, the coplanarity and co-colority relations (i.e., sharing similar color structure) permit the association of grasps to pairs of contours. Figure 3(c) illustrates the association of grasp affordances to an unknown object by using appearance and geometrical properties of the visual entities. The formalization of the visual change of a primitive under a rigid-body motion allows for the accumulation of the primitives belonging to the object (see Sect. 4).

4 The First Learning Cycle: Birth of the Object

Within the first learning cycle, the "objectness" of visually-detected structure in the scene **O1** is first tested by trying to obtain physical control over such detected structure and then manipulating it. In case the structure changes according to the movement of the robot arm, a 3D object representation is extracted.

Initial grasping behavior: To gain physical control over unknown objects a heuristic grasp computation mechanism based on [8, 12] is used. Pairs of 3D contours that share a common plane and have similar colors suggest a possible grasp; see Fig. 3(a–c). The grasp location is defined by the position of one of the contours. Grasp orientation is calculated from the common plane defined by the two features and the orientation of the contour at the grasp location. Every contour pair fulfilling this criteria generates multiple possible grasps (see Fig. 3(b) for one such possible grasp definition).

Accumulation: Once the object has been successfully grasped, the system moves it to present it to the camera from a variety of perspectives to accumulate a full 3D symbolic model of the object [7]. This process is based on the combination of three components. First, all primitives are tracked over time and filtered using an Unscented Kalman Filter based on the combination of prediction, observation and update stages. The prediction stage uses the system's knowledge of the arm motion to calculate the poses of all accumulated primitives at the next time step. The observation stage matches the predicted

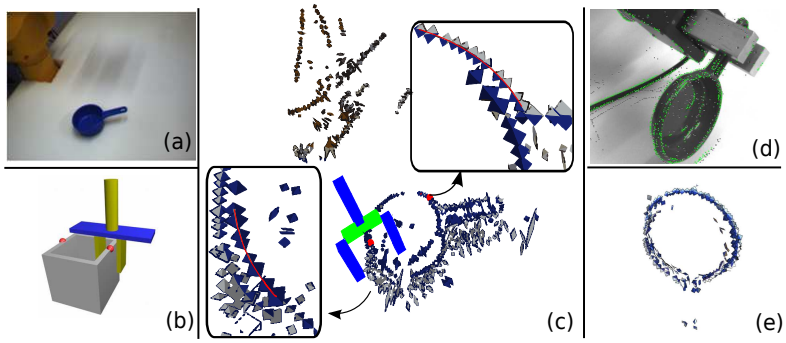


Fig. 3. (a–c) Initial grasping behavior: **(a)** A Scene, **(b)** Definition of a possible grasp based on two contours, **(c)** Representation of the scene with contours generating a grasp. **(d)** A step in the accumulation process where features from the previous scene get matched to the new scene. **(e)** Model extracted by the accumulation process.

primitives with their newly observed counterparts. The update stage corrects the accumulated primitives according to the associated observations. This allows the encoding and update of the feature vector. Second, the confidence in each tracked primitive is updated at each time step according to how precisely the accumulated primitive was matched with a new observation. The third process takes care of preserving primitives once their confidences exceed a threshold, even if they later become occluded for a long period of time. It also ensures that primitives are discarded if their confidence falls below a threshold. New primitives that were not associated to any accumulated primitive are added to the accumulated representation, allowing the progressive construction of a full 3D model. Note that the sparse nature of primitives yields a condensed description.

The learning cycle: Figure 4 (top) shows how the two sub-modules described above interact to generate object models for previously unknown objects. The initial grasping behavior is used to gain physical control over an unknown object. In case no object has been grasped in the process (this is determined using haptic feedback i.e. the distance of the fingers after grasping) another grasping option is executed. After the object has been grasped, the accumulation process is used to generate an object model which is then stored in memory. This process can be repeated until all objects in the scene have been discovered (a naive approach here can be to assume that we have learned all objects if grasping fails for a certain amount of trials). Results of the first learning cycle can be seen in Figs. 1(b), 3(e) and [11].

5 The Second Learning Cycle: Learning Grasp Affordances

In the second learning cycle, 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 **O3**. A mechanism such as that triggering the grasps in the first learning cycle generates a large number of grasping options (see Fig. 4 bottom). A random sample of these are then

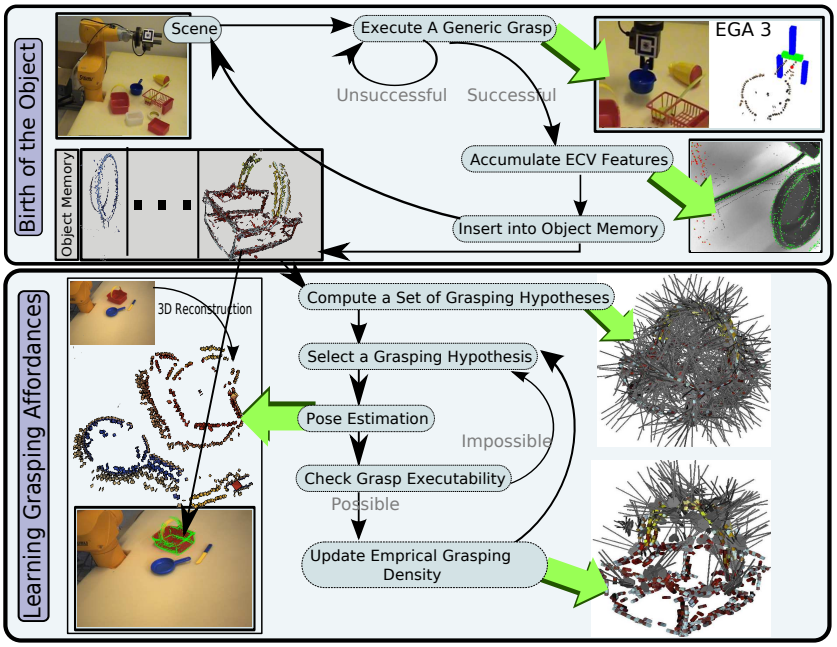


Fig. 4. The two learning cycles and the interaction between them. See text.

tested individually. Successful grasps are then turned into a probability density function that represents a the grasp affordances associated to the object **A1** in the form of the success likelihood of grasp parameters. This grasp density can then be used to compute the optimal grasp in a specific situation **A2** [10]. The second learning cycle is invoked after the first learning cycle has successfully establish the presence and the shape of an object.

Pose estimation: In preparation for pose estimation, a structural object model is built on top of the set of ECV primitives that has been accumulated in the first learning cycle. An object is modeled with a hierarchy of increasingly expressive object parts [9]. Parts at the bottom of the hierarchy represent ECV primitives. Higher-level parts represent geometric configurations of more elementary parts. The single top part of a hierarchy represents the object. A hierarchy is implemented as a Markov tree, where parts correspond to hidden nodes, and relative spatial relationships between parts define compatibility potentials.

An object model can be autonomously built from a segmented ECV reconstruction [9] as produced by the first learning cycle (Sect. 4). Visual inference of the hierarchical model is performed using a belief propagation algorithm (BP; see, e.g., [23]). BP derives a posterior pose density for the top part of the hierarchy, thereby producing a probabilistic estimate of the object pose.

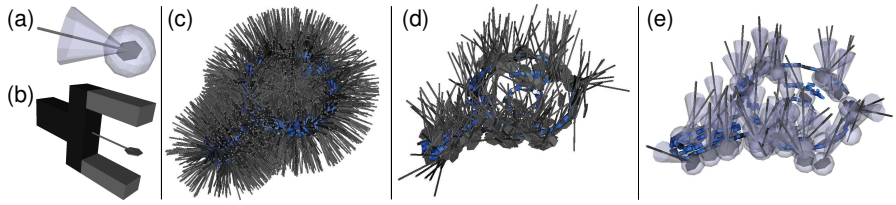


Fig. 5. Grasp density representation. **(a)** illustrates a particle from a nonparametric grasp density, and its associated kernel widths: the translucent sphere shows one position standard deviation, the cone shows the variance in orientation. **(b)** illustrates how the schematic rendering used in the top image relates to a physical gripper. **(c)** Samples from a grasp hypothesis density. **(d)** Samples from an empirical density learned from the hypothesis density in (c). **(e)** A 3D rendering of the kernels supporting the empirical grasp density in (d).

Grasp densities: When formalizing object grasp affordances, we mean to organize and memorize, independently of grasp information sources, the whole knowledge that an agent has about the grasping of an object. By *grasp affordance*, we refer to the different ways of placing a hand or a gripper near an object so that closing the gripper will produce a stable grip. The grasps we consider are parametrized by a 6D gripper pose and a grasp (preshape) type. The gripper pose is composed of a 3D position and a 3D orientation, defined within an object-relative reference frame.

We represent the grasp affordance of an object through a continuous probability density function defined on the 6D object-relative gripper pose space $SE(3)$ [10]. The computational encoding is *nonparametric*: A density is simply represented by the samples we see from it. The samples supporting a density are called *particles* and the probabilistic density in a region of space is given by the local density of the particles in that region. The underlying continuous density is accessed by assigning a kernel function to each particle – a technique generally known as *kernel density estimation* [24]. The kernel functions capture Gaussian-like shapes on the 6D pose space $SE(3)$ (see Fig. 5).

A grasp affordance is attached to the hierarchical model as a new *grasp node* linked to the top node of the network. The potential between grasp node and top node is defined by the grasp density. When an object model is visually aligned to an object instance, the grasp affordance of the object *instance* is computed through the same BP process as used for visual inference. Intuitively, this corresponds to transforming the grasp density to align it to the current object pose, yet explicitly taking the uncertainty on object pose into account to produce a posterior grasp density that acknowledges visual noise.

The learning cycle: Affordances can initially be constructed from a grasp generation method that produces a minimum proportion of successful grasps (e.g., the initial grasping behavior in Sect. 4). In this work we used an approach where we initially use grasp hypotheses at random orientations at the position of the ECV primitives of the object model. We call affordance representations built with any of these weak priors *grasp hypothesis densities* [10]. These are attached to the object hierarchical model, which will allow a robotic agent to execute *random samples* from a grasp hypothesis density under arbitrary object poses, by using the visual model to estimate the 3D pose of the object.

Although grasp hypothesis densities already allow grasping, it is clear that physical experience with an object will add valuable information. We thus use samples from grasp hypothesis densities that lead to a successful grasp to learn *grasp empirical densities*, i.e. grasps that have been confirmed through experience [10]. In this way, we increase grasping performance for the blue pan from 46% to 81%. The process of computing hypothesis densities, pose estimation and execution of random samples from the grasp hypothesis density through which a empirical density is generated is shown in Fig. 4 (bottom).

6 Discussion and Conclusions

The descriptions of the presented sub-modules [7–12] include an evaluation. We therefore only want to reiterate here a few important points that influence the performance and restrict the system. Because of the limitations of the robot system, the objects are limited by size (ca. 5–40 cm) and weight (up to 3 kg). Further restrictions are introduced by the vision system. The objects need to be describable by line-segment features and therefore can not be heavily textured. In addition the used stereopsis process can not reconstruct features on epipolar lines. This can lead to problems for the initial grasping behavior and the pose estimation process, but not the accumulation process.

Besides the blue pan object shown throughout this work we have successfully tested the full system on a toy knife and on a toy basket. The individual sub-components have been tested on more objects.

Autonomous systems benefit from an ability to acquire object and affordance knowledge without external supervision. We have brought together 3D stereo vision, heuristic grasping, structure from motion, and probabilistic representations combining visual features and gripper pose to autonomously segment objects from cluttered scenes and learn visual and affordance models through exploration. This enables an autonomous robot — initially equipped only with some generic knowledge about the world and about itself — to learn about objects and subsequently to detect, recognize and grasp them.

Acknowledgments

This work was supported by the Belgian National Fund for Scientific Research (FNRS) and the EU Cognitive Systems project PACO-PLUS (IST-FP6-IP-027657).

References

1. Gibson, J.: The Ecological Approach to Visual Perception. Houghton Mifflin (1979)
2. Lowe, D.G.: Fitting parametrized 3D-models to images. IEEE Transactions on Pattern Analysis and Machine Intelligence **13**(5) (1991) 441–450
3. Sahin, E., Cakmak, M., Dogar, M., Ugur, E., Ucoluk, G.: To afford or not to afford: A new formalization of affordances toward affordance-based robot control. Adaptive Behavior **15**(4) (December 2007) 447–472
4. Bicchi, A., Kumar, V.: Robotic grasping and contact: A review. In: IEEE Int. Conf on Robotics and Automation. (2000) 348–353

5. Krüger, N., Lappe, M., Wörgötter, F.: Biologically Motivated Multi-modal Processing of Visual Primitives. The Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour **1**(5) (2004) 417–428
6. Pugeault, N.: Early Cognitive Vision: Feedback Mechanisms for the Disambiguation of Early Visual Representation. PhD thesis, Informatics Institute, University of Göttingen (2008)
7. Pugeault, N., Wörgötter, F., Krüger, N.: Accumulated Visual Representation for Cognitive Vision. In Proceedings of the British Machine Vision Conference (BMVC) (2008)
8. Aarno, D., Sommerfeld, J., Kragic, D., Pugeault, N., Kalkan, S., Wörgötter, F., Kraft, D., Krüger, N.: Early reactive grasping with second order 3d feature relations. In: Recent Progress in Robotics: Viable Robotic Service to Human. Springer Berlin / Heidelberg (2008)
9. Detry, R., Pugeault, N., Piater, J.: A probabilistic framework for 3D visual object representation. IEEE Transactions on Pattern Analysis and Machine Intelligence (2009) (accepted).
10. Detry, R., Başeski, E., Krüger, N., Popović, M., Touati, Y., Kroemer, O., Peters, J., Piater, J.: Learning object-specific grasp affordance densities. In: International Conference on Development and Learning. (2009) (to appear).
11. Kraft, D., Pugeault, N., Başeski, E., Popović, M., Kragic, D., Kalkan, S., Wörgötter, F., Krüger, N.: 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
12. Popović, M.: An early grasping reflex in a cognitive robot vision system. Master's thesis, The Maersk Mc-Kinney Moller Institute, University of Southern Denmark (2008)
13. Fitzpatrick, P., Metta, G.: Grounding Vision Through Experimental Manipulation. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **361** (2003) 2165 – 2185
14. Modayil, J., Kuipers, B.: Bootstrap learning for object discovery. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) **1** (2004) 742–747
15. Biegelbauer, G., Vincze, M.: Efficient 3D object detection by fitting superquadrics to range image data for robot's object manipulation. In: IEEE International Conference on Robotics and Automation. (2007)
16. Richtsfeld, M., Vincze, M.: Robotic grasping based on laser range and stereo data. In: International Conference on Robotics and Automation. (2009)
17. Huebner, K., Ruthotto, S., Kragic, D.: Minimum volume bounding box decomposition for shape approximation in robot grasping. Technical report, KTH (2007)
18. Stoytchev, A.: Toward learning the binding affordances of objects: A behavior-grounded approach. In: Proceedings of AAAI Symposium on Developmental Robotics. (2005) 17–22
19. Stoytchev, A.: Learning the affordances of tools using a behavior-grounded approach. In: Affordance-Based Robot Control. Volume 4760 of Lecture Notes in Artificial Intelligence (LNAI). Springer (2008) 140–158
20. de Granville, C., Southerland, J., Fagg, A.H.: Learning grasp affordances through human demonstration. In: Proceedings of the International Conference on Development and Learning (ICDL'06). (2006)
21. de Granville, C., Fagg, A.H.: Learning grasp affordances through human demonstration. submitted to the Journal of Autonomous Robots (2009)
22. Baseski, E., Pugeault, N., Kalkan, S., Kraft, D., Wörgötter, F., Krüger., N.: A scene representation based on multi-modal 2D and 3D features. ICCV Workshop on 3D Representation for Recognition 3dRR-07 (2007)
23. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
24. Silverman, B.W.: Density Estimation for Statistics and Data Analysis. Chapman & Hall/CRC (1986)

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

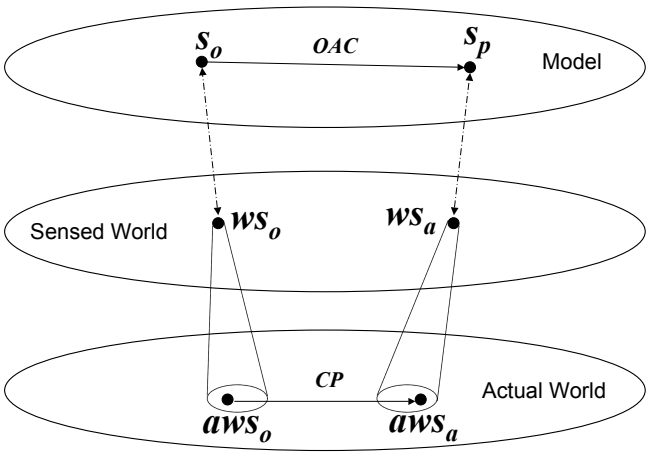


Figure 1: Graphical representation of an OAC and its relationship to a control program

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.

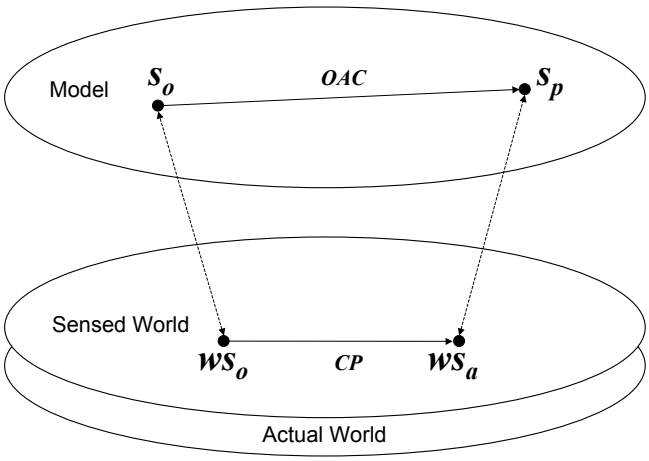


Figure 2: Graphical representation of an OAC and its relationship to the sensed world and a control program

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

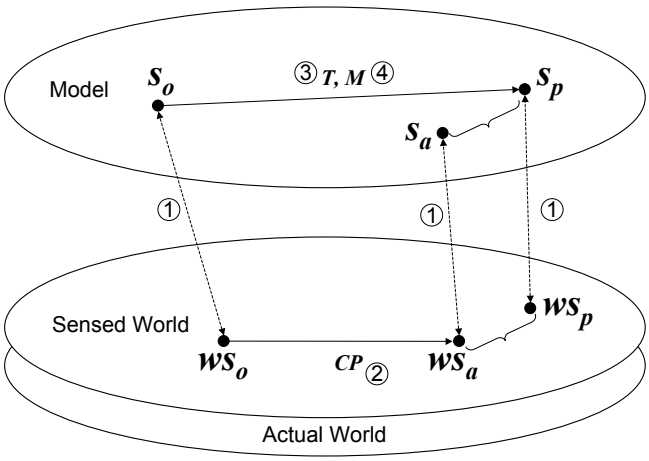


Figure 3: Graphical representation of the OAC learning problems

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.

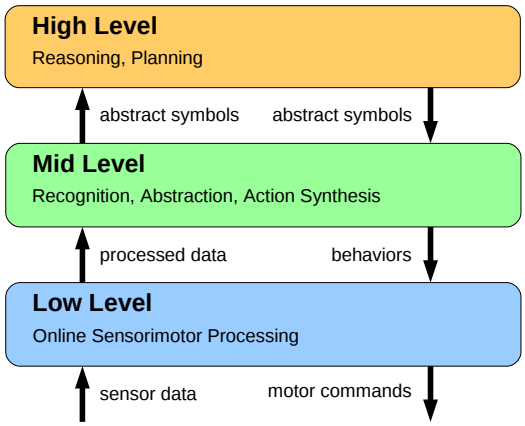


Figure 4: The three-level architecture supporting the example OACs in Section 6.

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:

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

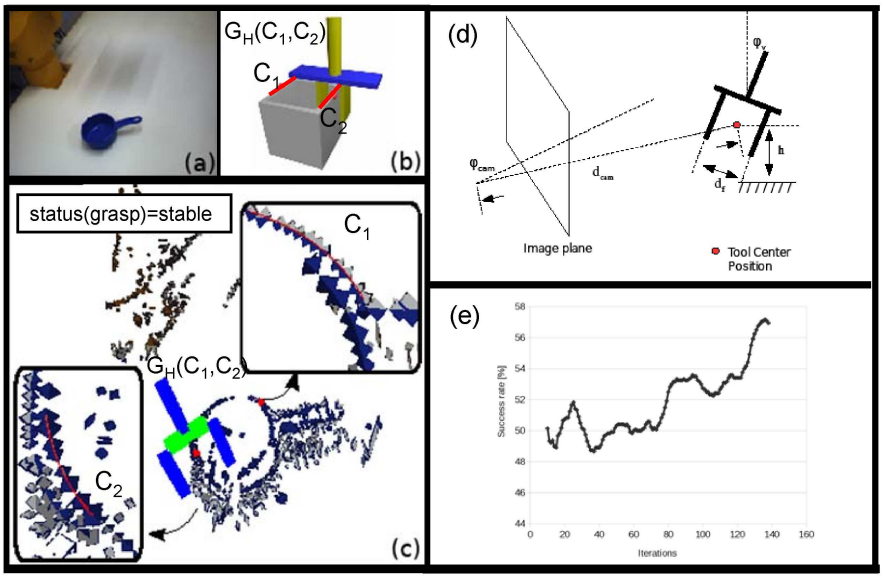


Figure 5: (a) The image of the scene captured by the left camera. (b) A possible grasping action type defined by using the two coplanar contours C_1 and C_2 shown in red. (c) A successful grasping hypothesis. The 3D contours from which the grasp was calculated are shown. Note that the information displayed 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.

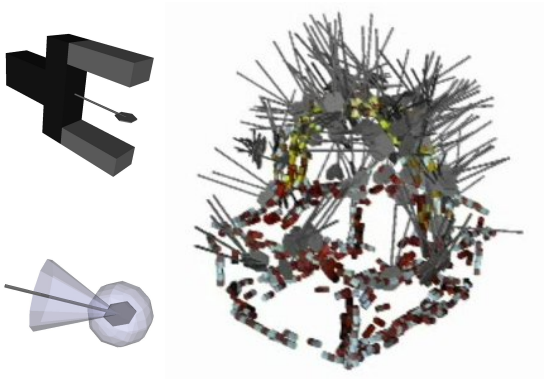


Figure 6: Visualization of grasp densities as used in Figure 7. A (continuous) grasp density is represented nonparametrically by particles. Each particle represents a 6D gripper pose (top left). The set of particles is interpreted as a continuous density via kernel density estimation, using a combination 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.

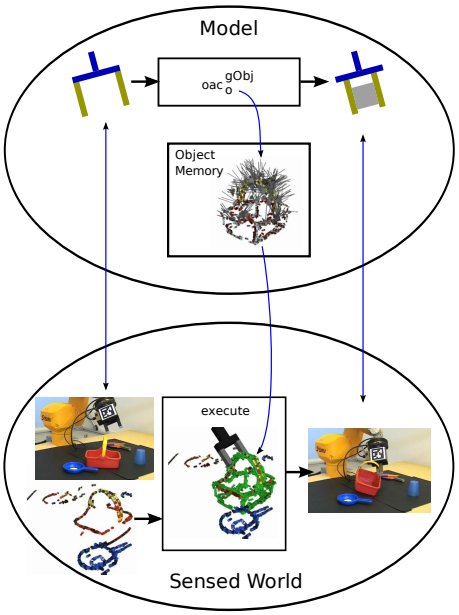


Figure 7: $\texttt{oac}_o^{\texttt{graspObj}}$ and its interaction with the environment (cf. Figure 2). The precondition (gripper empty) and predicted result (gripper holding an object) are mapped onto corresponding sensor states. The OAC refers to a specific object model (stored in the object memory $\mathcal{M}^{\mathrm{O}}$). 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.

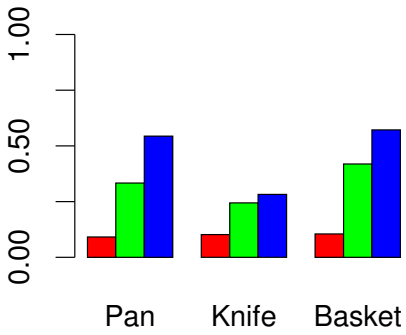


Figure 8: Evolving statistics of status(`grasp`) = `stable` for the OACs $\texttt{oac}_{\text{Pan}}^{\texttt{graspObj}}$, $\texttt{oac}_{\text{Knife}}^{\texttt{graspObj}}$, and $\texttt{oac}_{\text{Basket}}^{\texttt{graspObj}}$ over cumulative rounds of grasping trials [25].

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

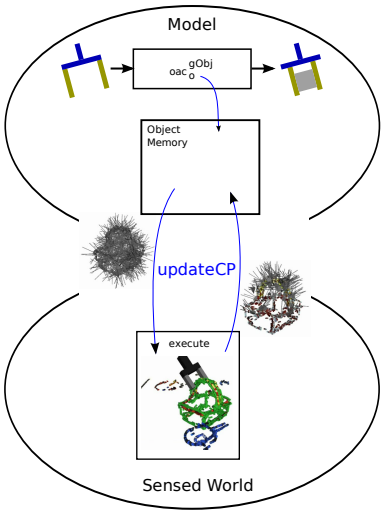


Figure 9: The principal learning capability of $\texttt{oac}_o^{\texttt{graspObj}}$ (cf. Figure 3). A grasp density used for execution can be updated by incorporating new experiments.

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 $\text{range}(T) = \{\text{loc}(o)\}$.

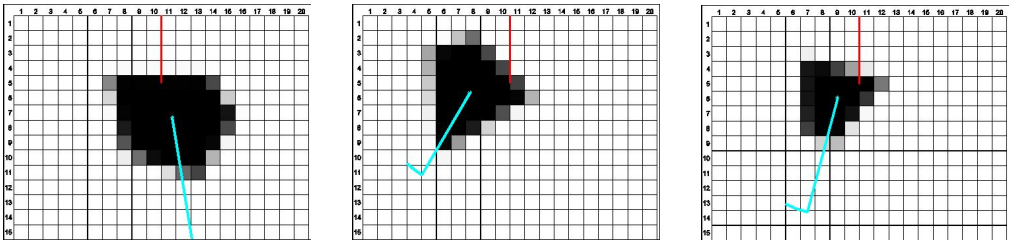


Figure 10: Samples of low resolution object images used as input to the neural network.

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

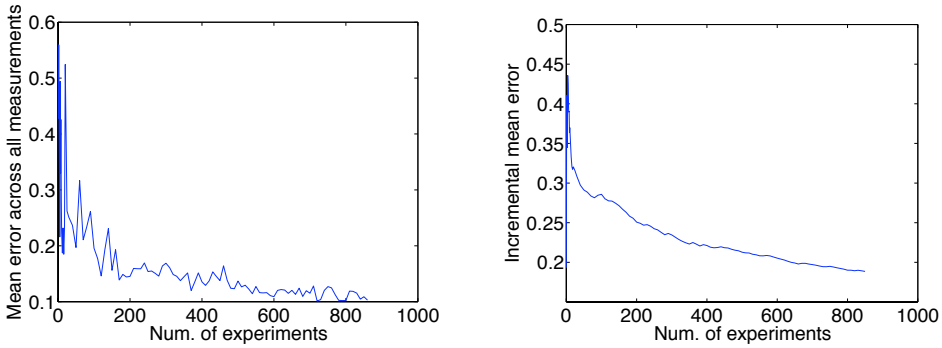


Figure 12: Mean error of robot pushing. The left figure shows the mean error calculated using Eqn. (14) and all measurements. The right figure shows the incremental statistical evaluation as realized by `updateM`. Four different objects were used in the experiment.

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`; $\text{bin}(o)$; $\text{loc}(o)$;
  experiment = `execute`(`push`);
  if $\text{d}(\text{loc}(o), \text{loc}_a(o)) > \epsilon$ **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.

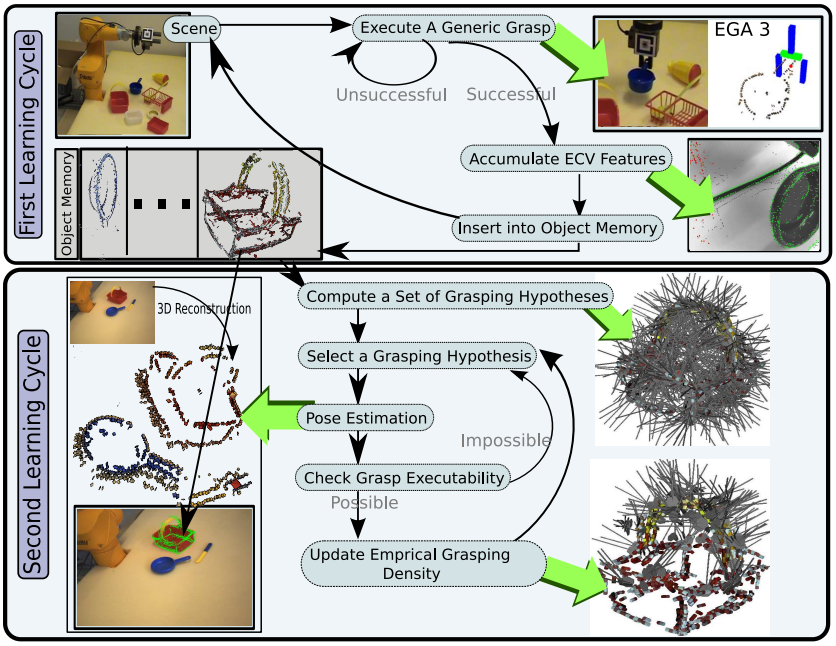


Figure 13: Grounding the OAC $\mathtt{oac}_o^{\mathtt{graspObj}}$ in two learning cycles. Within the first learning cycle, physical control over a potential object is obtained by the use of $\mathtt{oac}^{\mathtt{GenGrasp}}$. Once control over the object is achieved and the visual structure changes according to the movement of the robot arm, a 3D object representation is extracted and stored in the memory. In the second learning cycle $\mathtt{oac}_o^{\mathtt{graspObj}}$ 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 $\mathtt{inGripper}(o)$ in a world initially described by the predicate set:

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

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]

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

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($\texttt{grasp}$) = $\texttt{stable}$ 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 $\texttt{oac}_o^{\texttt{graspObj}}$: $\texttt{updateM}$ revises the long-term statistics M, and $\texttt{updateCP}$ 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).

[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, 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.

9th IEEE-RAS International Conference on Humanoid Robots
December 7-10, 2009 Paris, France

Autonomous acquisition of pushing actions to support object grasping with a humanoid robot

Damir Omrčen, Christian Böge, Tamim Asfour, Aleš Ude, and Rüdiger Dillmann

Jožef Stefan Institute, Slovenia, damir.omrcen@ijs.si, ales.ude@ijs.si

University of Karlsruhe, Germany, asfour@ira.uka.de, c.boege@gmx.de, dillmann@ira.uka.de

***Abstract*— There are many situations in which an object that needs to be grasped is not graspable, but could be grasped if it was situated at a different location. By applying nonprehensile manipulation actions such as poking, the object can be moved to a new location without first being grasped. We consider these issues in the context of an artificial cognitive system. The goal of the paper is twofold; firstly, we study how the robot can acquire nonprehensile manipulation knowledge by observing the outcomes of exploratory movements on objects. We propose a learning process that enables the robot to acquire a general pushing rule describing the relationship between the direction of poke and the observed object motion for a class of objects. In this way the robot acquires new action knowledge without having any specialized prior model about the action. Secondly, we investigate how the acquired action knowledge can be used to realize grasping in complex situations where the robot could not grasp the object without moving it to a new location. Here the learned poking behavior serves as a support action for robot grasping. The proposed approach has been implemented and tested on a humanoid robot Armar III.**

I. INTRODUCTION

Autonomous robots should explore their environment actively, persistently, and systematically as most animals and humans do [16]. To study and develop intelligent humanoid robots, we therefore pursue a developmental approach. Cognitive abilities should develop by refining the knowledge acquired in previous stages of the development [8]. In the context of this paper, the first stage involves learning to distinguish the robot body from the rest of the world. Afterwards, the robot can move on to the second stage, that is interaction with external objects. The next stage involves interpreting object-object interactions. In this paper we focus on the second stage – interaction with external objects.

In a cognitive system objects and actions cannot be separated because objects can induce actions (cup → drink), while actions can redefine objects. Objects and actions are inseparably intertwined and higher-level categories are therefore determined (and also limited) by the action an agent can perform and by the attributes of the world it can perceive; the resulting, so-called Object-Action Complexes (OACs) are the entities on which cognition develops (action-centered cognition) [17]. While this paper is concerned with OACs at the level of early perception-action events, our research strives to provide a continuous path from such events to complex cognitive processes, where OACs are used as basic building blocks.

One approach towards the acquisition of new object-action knowledge is that such information can be obtained by performing exploratory primitive actions on a number of different objects. By observing the changes in the environment caused by the applied actions, the robot can associate the applied actions with the resulting object behavior and thus gain understanding of causes and effects. As a representative example we study the acquisition of nonprehensile manipulation knowledge, i.e. object manipulation without a grasp. This kind of manipulation is used when it is difficult or impossible to grasp an object, e.g. when an object is too wide, too large or too heavy. As an example of nonprehensile manipulation, we focus on poking, which is defined as a short term pushing action. Poking can also be used in order to identify and segment the objects from the background [4].

Our aim is to obtain a general **pushing rule** rather than an object specific one. Humans are good at generalization, especially when their experiences are very diverse. The same generalization capabilities need to be achieved in autonomous robots. To achieve a reasonably high level of generalization, some authors use recurrent neural networks with parametric bias [10][14], where static images of objects are linked to dynamic features of objects. In this paper we achieve generalization of the pushing rule by using object images as input to the system, which provides appropriate data to extrapolate the pushing rule.

When poking an object, the object motion depends on the object's shape, weight distribution and on the support friction forces. A lot of work has already been done in the field of mechanics on the controllability and planning for poking [7] [6]. Obviously, poking could easily be implemented by assuming a proper representation for the physics of the task, but such an approach relies on a priori knowledge about the action and therefore does not solve the complete learning problem. Additionally, it is sometimes difficult to obtain the model parameters using available sensors (e.g. it is very difficult to obtain friction between the object and the pusher using vision). If the physical model of the object and the action is not available like in our work, the robot has to experiment with different poking actions on the object. In this way the robot acquires new knowledge from exploration and human demonstration in the same way as infants learn their actions – performing actions on objects means playing with toys.

While poking has been used to study cognitive processes before [5], our work focuses on different issue, which is learning generalized pushing rules with the application to grasping. After learning, the robot can use the newly acquired knowledge to push an object towards a specified location. On the other hand, Fitzpatrick et al. [5] were primarily concerned with using poking actions to extract the associated object properties. Pushing has often been used as an example when learning affordances [13], but this type of research normally focuses on higher-level knowledge such as "this ball affords pushing". Instead, our work focuses on acquiring fine-grained controller that can be used to move a range of different objects in any specified direction.

Only objects in proximal space are graspable and only objects of a certain size and shape can be grasped. There are many studies on graspability of objects [18] and how to generate grasp hypothesis. Even if the object is graspable in free space, it might not be possible to grasp it when other objects, e.g a surface on which it lies, are taken into consideration. In this paper we study how such problems can be resolved by pushing.

II. SCENARIO DESCRIPTION

The experiment has been designed as follows (see Fig. 1). The robot stands at a table and needs to grasp an object that lies on the table. After detecting that the object is not graspable at a given configuration because it is too wide, the robot has to bring the object to the table boundary, where the object becomes graspable due to its flatness. The robot uses its hand to push the object. The part of the hand that has been used for pushing will be termed as a *pusher*. Once the object is pushed to the table boundary, it becomes graspable and the robot grasps it (see also the attached video).

The work is conducted under the following hypothesizes: objects are flat with homogenous mass distribution, the surface is flat, the friction is uniform, maximal size of object is limited.

To learn a general pushing rule, the robot starts by experimenting with different primitive poking actions[1] applied to different objects and at different locations on the objects' boundaries. With this process the robot builds a knowledge base, which describes the relationship between the point and angle of push on one side and the actual object movement on the other side. Based on this data, a neural network is learned, which maps the performed pushing action and shape of the pushed object to the resulting object movement.

Afterwards the robot can use the acquired knowledge to find the right poking action in order to move the object as desired (i.e. the robot should push an object to a graspable position). The final goal of the object is defined by a higher-level motion planer, which is not described in this work. The objects used in experiments are planar polygonal objects as shown in Fig. 7.

[1] We implemented the primitive poking actions are straight line movements of a pusher in a given direction. Regarding pushing, this is the only prior knowledge available to the system.

For object pose estimation, a stereo-based approach presented in [3] has been applied. Textured object are recognized and localized using 2-D feature point correspondences between the current object snapshots and the off-line learned views, which are stored as part of the object representation in an object database. The pose is computed on the basis of triangulated subpixel-accurate stereo correspondences within the estimated 2-D area of the object, yielding 3-D to 3-D point correspondences with a training view. On the other hand, recognition and localization of single-colored objects combines model-based view generation with stereo-based position estimation. Orientation information is retrieved from the matched views and an accurate pose is calculated by a pose correction procedure, as presented in [3].

The proposed techniques were implemented on a humanoid robot Armar III [1] (see Fig. 1) and industrial arm Mitsubishi Pa-10. Grasping has been implemented only on the humanoid robot using visual servoing techniques as presented in [15]. Given the object position, a collision-free motion is generated to reach a pre-grasp position of the arm using inverse kinematics. To estimate the hand position, an artificial marker is attached to the wrist of the robot and tracked visually while moving the hand to the grasp pose. The required orientation of the hand is computed using the forward kinematics.

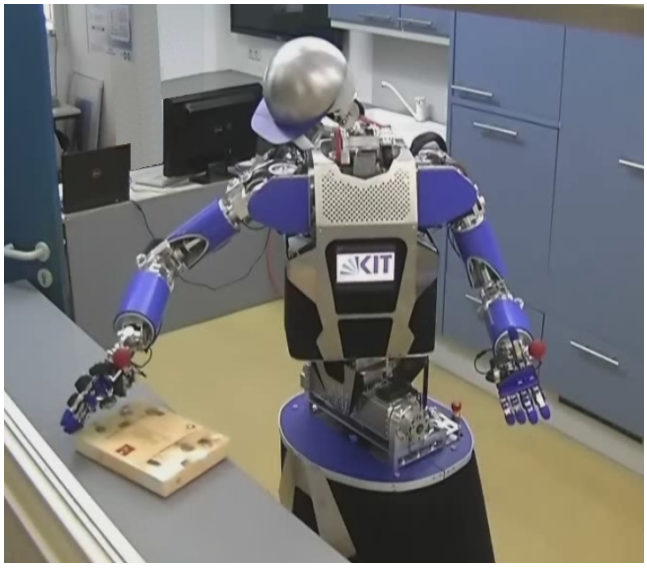


Fig. 1: Humanoid robot Armar III during pushing action

III. LEARNING OF THE PUSHING RULE

The pushing rule is learned using an exploratory process introduced in the previous section. The task of the robot is to learn the relationship between the point and angle of push on the object's boundary and the actual object movement after the pushing action is performed (see Fig. 2). We call the problem of learning the movement of the object being pushed the **direct** pushing problem.

In our previous work [11], the robot learned the response of only one object after the push has been performed. Thus for each new object, the robot had to learn everything from scratch. There was no prediction and no generalization to

other objects. Here we propose an approach that can generalize the pushing rule to objects that differ in shape and size, including objects whose response to pushes was not observed during learning. The set of objects used in the experiment is shown in Fig. 7.

In the learning phase the robot experiments with a number of different poking actions. The robot has to poke an object from different sides and at various angles (see Fig. 2). Additionally, it has to experiment with different objects to achieve generalization. In the beginning of the process the robot has no knowledge about how objects respond to the primitive poking actions, thus initially the robot experiments with different poking actions randomly.

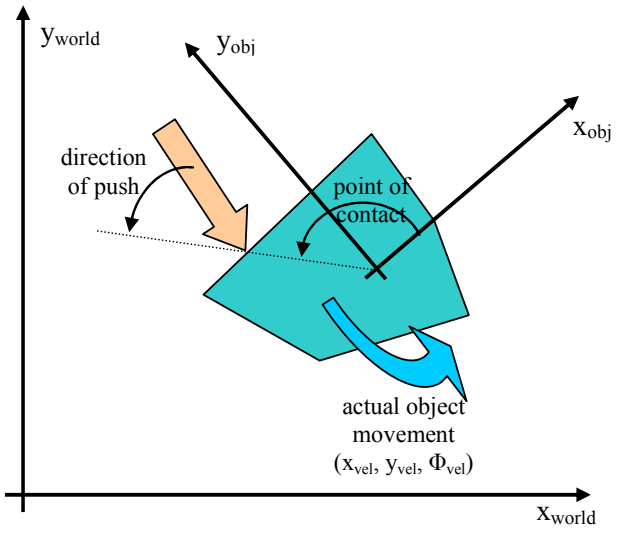


Fig. 2: Schematics of a poking action

After a poking action has been applied to an object, the object accelerates and changes its position and orientation. Since the objects are relatively light and the friction between the object and the table is relatively high, we can neglect the dynamic properties of motion. Typical response of the object is shown in Fig. 3. The object velocity settles in less that 200 ms. The object velocity estimated by vision is noisy, but since learning takes place after the push is completed, the data can be filtered and processed well before the use.

Since the object settles its motion in a very short time, the response of the object to the poking action is determined with sufficient accuracy by simply observing the displacements of the pusher and the object. The displacement of the pusher is expressed by two parameters: the point and the angle of contact on the object boundary, which define the direction of push. The primitive poking actions keep the velocity of the pusher constant while performing the action. The point of contact is expressed as the angle between the line segment connecting the point of contact and the centre of the object and the x-axis of the object's coordinate system. Similarly, the angle of a contact is expressed as the angle between the pushing direction and the tangent at the point of contact (see Fig. 2).

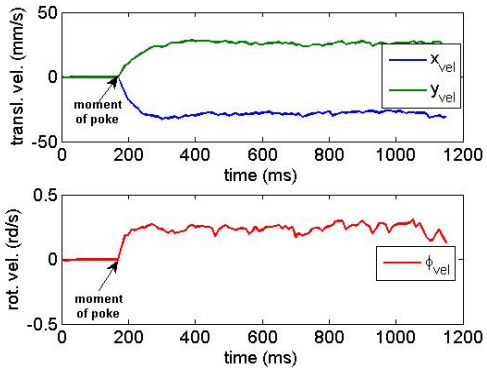


Fig. 3: Typical response (velocity) of an object after applying a poking action

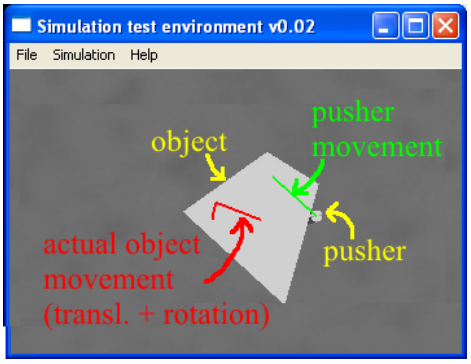


Fig. 4: Agent view of a scene during learning

The response of an object is represented by three parameters, i.e. the planar velocity of the object centre and the rotational velocity about the centre point on the object. The abstract robot's view of the experiment is shown in Fig. 4.

Humans can predict the motion of an object to be pushed based on the acquired object images, which are used as input to the neural networks in the brain. The application of retina images as an input to control the robot behavior has already been studied in robotics. For example, Oztop et al. have used retina images and Hopfield networks to realize hand posture imitation [12]. Inspired by these findings, we utilized the binarized object images as an input to the system. To limit the search space that needs to be explored, we normalized the binarized images with respect to the pushing direction in the retina image. Before each primitive poking action is applied, the observed scene is mapped in such a way that the pushing direction is always at the same position on the retina (see Fig. 5). This normalization process ensures that the acquired knowledge is invariant against object position and orientation as long as the pusher is able to apply the primitive poking actions at the given configuration.

The original resolution of the camera images was 640 x 480 pixels. Due to the computational complexity and to achieve faster convergence and better generalization, we reduced the resolution of the input image to 20 x 15 pixels. Translated and rotated retina images of reduced resolution served as input to the neural network, which is used to represent the pushing rule. The network has three outputs to represent the predicted velocity of an object in all three directions.

We applied a two-layer backpropagation network with 300 input neurons, 10 neurons in the hidden layer, and 3 output neurons. Each input neuron corresponds to a pixel of the object's reduced resolution image. The value of the pixel is in the range from 0 to 1, where 0 means that no object is present at the pixel, while 1 means that the pixel is covered by the object. The three output neurons correspond to the object's linear and rotational velocity on the support surface. The output velocities have been normalized to the range from -1 to 1. The hidden layer as well as the output layer use the tan-sigmoid transfer function. To train the network we employed the Levenberg-Marquardt training function.

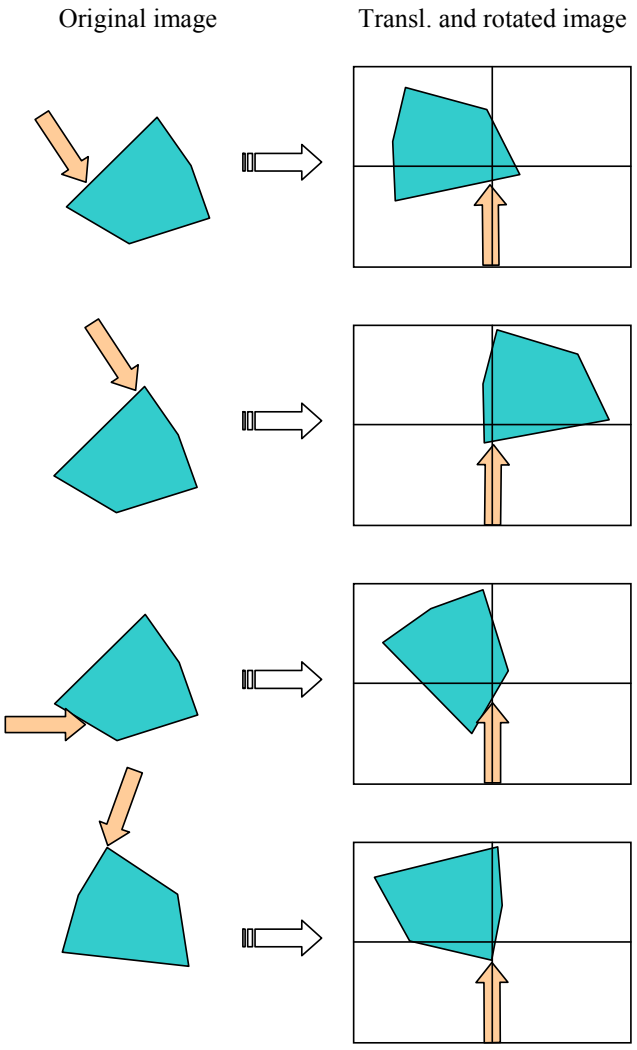


Fig. 5: The binarized object image is rotated and translated to ensure the same pushing position and direction on the retina

IV. Applying The Pushing Rule

After the learning phase is completed, the robot can generate a poking action to move an object in the desired direction. We call this process the **inverse** pushing problem. The inverse problem deals with where and how the object has to be pushed to achieve motion close to the desired one.

The aim of the robot in this phase is to perform a set of poking actions in order to bring the object to the desired location. Here, a higher-level motion planner should provide the desired movement of the object, whereas the lower-level controller needs to solve the inverse problem. The agent view of the poking scene is shown in Fig. 6.

Note that the robot cannot always achieve the desired velocity due to the physical limitations of the action (this is still a nonprehensile action). In many cases it happens that an object cannot be moved in the desired direction because the pusher slides from the object boundary or it even moves away from the boundary. Such events cannot happen when the object is firmly grasped.

To solve the inverse pushing problem, i.e. to achieve an optimal pusher motion for the desired pushing direction, the agent needs to optimize a criterion function with respect to the point and angle of push, e. g. the weighted square error between the desired motion and the predicted one. Thus we need to find a global minimum of the following function:

$$e = (\mathbf{W}(X_{des} - X_{pred}))^2, \qquad (1)$$

where X_{des} represents the desired motion in all three DOFs and X_{pred} represents the motion of the object which is predicted by the neural network, respectively. **W** is the weight specifying the importance of each direction.

To solve the inverse problem by finding the minimum of Eq. (1), we use classical optimization techniques. For an initially selected point and angle of push on the object boundary, the acquired binarized image is transformed as described in the previous section. Based on the generated object image, we predict the movement of the object using the learned neural network. The predicted velocity is compared to the desired one and a new point and angle of push are determined by the optimization method. The process is repeated until an appropriate point and angle on the object's boundary are found. We believe that this process is similar to how humans visualize their action before doing it.

Pushing is a nonprehensile action and it is therefore difficult to ensure that an object will move exactly in the desired direction, both because of the inaccuracies in the learned model and because the optimization process might not find an optimal solution, e.g. because it is stuck in a local minimum. We therefore realized the "pushing to a desired location" behavior as a feedback process, where the robot repeatedly pushes the object until the final position is reached.

The computational complexity of the optimization process is relatively high. However, the pushing point and angle has to be updated with a relatively low frequency (every second in the current implementation). And the computational complexity is still low enough to be easily calculated in the available time. If this is not the case the robot can wait with action execution until the update is available. On the other hand the low-level robot joint control loop runs at much higher sample rate.

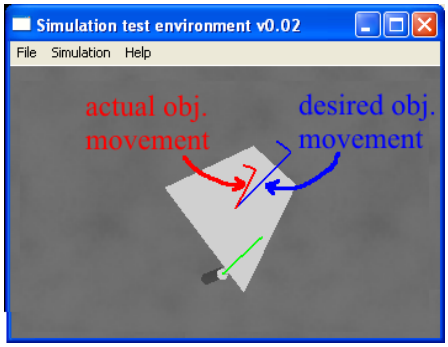


Fig. 6: Agent view of a scene while controlling object movement

V. Pushing for Grasping

In our previous work [2], we presented a framework for object grasping and manipulation, which incorporates the described vision system for object localization, a path planner for the generation of collision-free trajectories and an offline grasp analyzer that provides the most feasible grasp configurations for each object. The results provided by the system's components are stored and used by the control system of the robot for the execution of a grasp of a particular object. The central assumption of this framework is the existence of a database with 3-D models of all objects encountered in the robot's workspace and a 3-D model of the robot hand. Grasp hypotheses for each object are generated in simulation using the grasping simulation environment GraspIt! (see [9]) and stored as part of the object representation in the database.

The graspability of the object depends on the support surface, object properties and the available hand. In order to determine whether an object can be grasped with the available robotic hand, we use the simulator to validate the grasp hypotheses associated with the object. A hypothesis is rejected if its execution (in simulation) causes collision with the surface on which the object is placed. If the system generates some hypotheses but none of them leads to a successful grasp, the object must be first relocated before being grasped. One possible way is to push the object to the rim of the table to make it graspable. For this purpose we select one of the hypotheses and compute the pose of the object at the edge of the table so that the object surface associated with the grasp hypothesis lies beyond the rim of the table.

While the grasping part of the system is implemented in a classic way, we are currently working towards a system that can learn to generate a set of grasp hypotheses based on the general properties of the object and test grasp executions.

VI. Results

Our humanoid robot uses whole body manipulation to manipulate the object. Here, the mobile platform has three DOFs, the hip has 1 DOF and there are additional 7 DOFs in the arms. Depending on the reachability of the desired point, the robot can use the right or the left arm. Technically, to achieve a pushing action with a cylinder shaped pusher, five DOFs are necessary. Three DOFs are needed to control the position of the pusher and two DOF are needed to control the rotations. One DOF of rotation about the cylinder axis is not important and therefore does not need to be considered in the controller. To control the robot we used a velocity based task controller with null space joint limit avoidance.

A poke or a short time push is a pushing action which last 2 sec in the current implementation. The controller is defined is such a way that the pusher moves at a constant predefined speed to a pushing direction which is defined in the object frame.

We performed the learning process on a set of different planar objects shown in Fig. 7. The white polygonal objects were used for learning while the chocolate box was used to validate the learning process, i.e. to validate the generality of the pushing rule.

The real robot generated a hand-guided movement around the object (see Fig. 8, green line), which resulted in several randomly distributed pushes of the object. The experiment took about 2 minutes for each object. After the experimentation the data was filtered, velocities and positions of the object were calculated, and learning instances were generated. Among all the measurement samples we used only those that resulted in significant object movements. For all five objects we collected 867 instances, which were used for training of the neural network. The sample learning instances are shown in Fig. 9.

Fig. 7: A set of objects that were used for learning

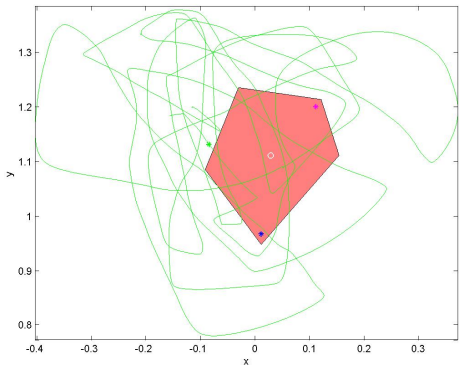


Fig. 8: Green line shows the movement of the pusher during the 2 minute experiment. Red patch represents an object at the initial position.

To validate the proposed approach, we compared the predicted and the actual velocity of the object. We evaluated the differences in the direction of object's movement. Experiments showed that the object movement prediction gets better as the number of instances gets larger. Fig. 10 shows the mean error of all measurements. The x-axis represents the number of instances used for training of the neural network. When the number of instances used for training is very small, e.g. up to 50 instances, the mean error is about 0.5 rad. However, already when using a set of 200 instances, the mean error in prediction gets better and is about 0.2 rad. The error drops further as more data is collected. Considering unknown friction, low resolution of the image, and the generalization property of the pushing rule, we consider this as a very good result.

The learning process would converge faster and the error would be smaller if we provided more initial knowledge to the system. However, our goal was to develop a system which acquires new knowledge by its own actions, so that a robot could evolve into a more intelligent machine. Therefore, as little as possible was hardcoded to learn the pushing rule. Note that the primitive poking movements could also be learned.

The acquired pushing rule has been applied to push an object to a graspable position. Fig. 11 (see also the attached video) shows the robot during pushing and grasping. The object was not graspable at its initial location. The robot thus generated a plan to move the object to a position where the robot could grasp it. After a few pushes, the object was brought to a graspable position, where the robot could successfully grasp it.

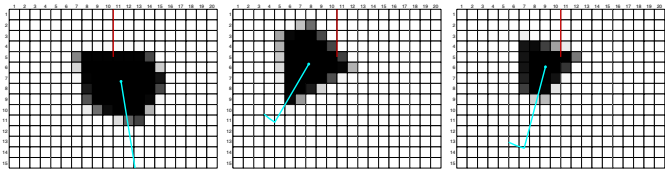


Fig. 9: Sample learning instances. The cyan line indicates the actual object movement, while the red line indicates the pushing direction. The object is shown in black.

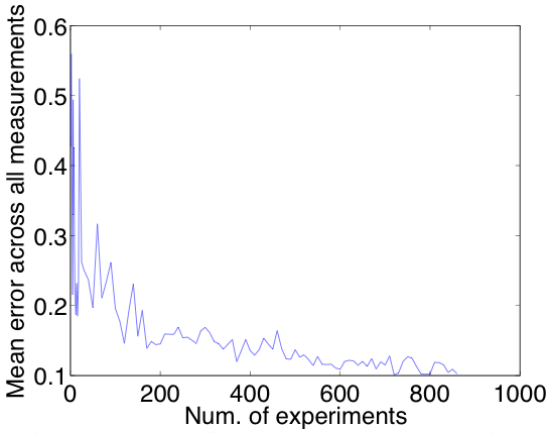


Fig. 10: The error in velocity direction prediction decreases as the number of learning samples increases

VII. DISCUSSION AND FUTURE WORK

In summary, we realized the process of associating object-action cause-effects through an explorative, self-emergent process. Such processes are of great importance for the early cognition. No specialized knowledge about the pushing action was provided to the robot. We only provided rules about how to explore the environment and the robot associated the applied actions to object responses independently. We believe that such an explorative learning process, possibly combined with imitation, is one of the keys to natural sensorimotor learning.

While precise learning of pushing actions can take a long time, the agent can learn a rough but reasonable approximation of the behavior already after a few explorative pushes. This initial knowledge can already be used for a rather rough control of the object movement.

While controlling the motion, the robot can update its knowledge base by observing the actual movement of the object. Thus the relationship between the desired and the actual object motion gradually becomes more accurate and the control of the object movement direction improves. Additionally, to make the learning of poking actions more optimal, human instructor can demonstrate the most representative pokes (e.g. perpendicular pokes from a few different sides). Incremental learning combined with imitation is the next important topics of our research.

One could argue that the experiments on the real robot do not guaranty the success of the proposed approach. Reproducibility could be a problem due to inaccuracies in pushing point and direction and the strong friction nonlinearity between the object and table. The same problem occurs in human learning, but we still use the pushing action very successfully. In the robot case the learning should only provide a rough model of the action. However, the action execution should be strongly supported by online closed loop control, which solves the problem of incomplete model. And that is our future work.

Additionally it is crucial to update the robot's knowledge base by observing the actual movement of the object. That would improve the accuracy of the pushing model.

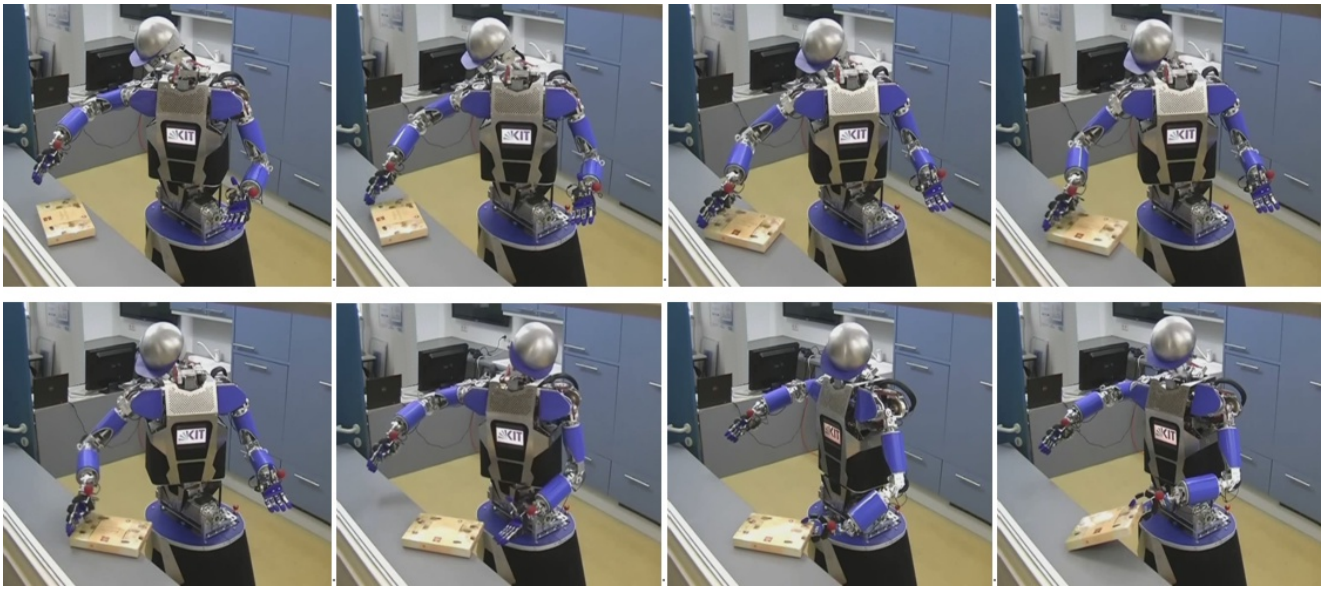


Fig. 11: A sequence of robot pushes that bring the object to a graspable position

REFERENCES

[1] T. Asfour, K. Regenstein, P. Azad, J. Schröder, A. Bierbaum, N. Vahrenkamp, and R. Dillmann. ARMAR-III: An integrated humanoid platform for sensory-motor control In *Proc. IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 169-175, Genoa, Italy, 2006.

[2] T. Asfour, P. Azad, N. Vahrenkamp, K. Regenstein, A. Bierbaum, K. Welke, J. Schröder, R. Dillmann, Toward humanoid manipulation in human-centred environments. *Robotics and Autonomous Systems*, 56(1):54-65, 2008.

[3] P. Azad, T. Asfour, and R. Dillmann, Stereo-based 6D object localization for grasping with humanoid robot systems. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 919–924, San Diego, USA, 2007.

[4] P. Fitzpatrick and G. Metta. Grounding vision through experimental manipulation. *Royal Society of London Transactions Series A*, 361(1811):2165–2185, 2003.

[5] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini. Learning about objects through action - initial steps towards artificial cognition. In *Proc. IEEE Int. Conf. Robotics and Automation*, pp. 3140–3145, Taipei, Taiwan, 2003.

[6] Q. Li and S. Payandeh. Manipulation of convex objects via two-agent point-contact push. *The International Journal of Robotics Research*, 26(4):377–403, 2007.

[7] K. M. Lynch and M. T. Mason. Stable pushing: mechanics, controllability, and planning. *The International Journal of Robotics Research*, 15(6):533–556, 1996.

[8] G. Metta, G. Sandini, L. Natale, Manzotti R., and F. Panerai. Development in artificial systems. In *Proc. EDEC Symposium at the Int. Conf. on Cognitive Science*, Beijing, China, 2001.

[9] A. T. Miller and P. K. Allen. GraspIt! a versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine,* 11(4):110-122, 2004.

[10] S. Nishide, T. Ogata, J. Tani, K. Komatani, and H. G. Okuno. Predicting object dynamics from visual images through active sensing experiences. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2501–2506, Rome, Italy, 2007.

[11] D. Omrčen, A. Ude, and A. Kos. Learning primitive actions through object exploration. In *Proc. IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 306-311, Daejeon, Korea, 2008.

[12] E. Oztop, T. Chaminade, G. Cheng, and M. Kawato. Imitation bootstrapping: Experiments on a robotic hand. In *Proc. IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 189–195, Tsukuba, Japan, 2005.

[13] E. Sahin, M. Cakmak, M. R. Dogar, E. Ugur, and G. Ucoluk. To afford or not to afford: A new formalization of affordances towards affordance-based robot control. *Adaptive Behavior*, 15(4):447-472, 2007.

[14] J. Tani and M. Ito. Self-organization of behavioral primitives as multiple attractor dynamics: A robot experiment. *IEEE Trans. on SMC, Part A*, 33(4):481–488, 2003.

[15] N. Vahrenkamp, S. Wieland, P. Azad, D. Gonzalez, T. Asfour and R. Dillmann. Visual servoing for humanoid grasping skills. In *Proc. IEEE-RAS Int. Conf. on Humanoid Robots,* pp. 406-412, Daejeon, Korea, 2008.

[16] W. G. Walter. An imitation of life. *Scientific American*, 182(5):42–45, 1950.

[17] F. Wörgötter, A. Agostini, N. Krüger, N. Shylo, B. Porr. Cognitive agents — a procedural perspective relying on the predictability of Object-Action-Complexes (OACs), *Robotics and Autonomous Systems*, 57:420-432, 2009.

[18] X. Zhu, H. Ding, and M. Y. Wang, A numerical test for the closure properties of 3-D grasps, *IEEE Trans. Robotics and Automation*, 20(3):543-549, 2004.

Combining Cognitive Vision, Knowledge-Level Planning with Sensing, and Execution Monitoring for Effective Robot Control

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Dirk Kraft **Norbert Krüger**
The Maersk Mc-Kinney Moller Institute
University of Southern Denmark
DK-5230 Odense M, Denmark
{kraft,norbert}@mmmi.sdu.dk

Mark Steedman
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
steedman@inf.ed.ac.uk

Abstract

We describe an approach to robot control in real-world environments that integrates a cognitive vision system with a knowledge-level planner and plan execution monitor. Our approach makes use of a formalism called an Object-Action Complex (OAC) to overcome some of the representational differences that arise between the low-level control mechanisms and high-level reasoning components of the system. We are particularly interested in using OACs as a formalism that enables us to induce certain aspects of the representation, suitable for planning, through the robot's interaction with the world. Although this work is at a preliminary stage, we have implemented our ideas in a framework that supports object discovery, planning with sensing, action execution, and failure recovery, with the long term goal of designing a system that can be transferred to other robot platforms and planners.

Introduction and Motivation

A robot operating in a real-world domain must typically rely on a range of mechanisms that combine both reactive and planned behaviour, and operate at different levels of representational abstraction. Building a system that can effectively perform these tasks requires overcoming a number of theoretical and practical challenges that arise from integrating such diverse components within a single framework.

One of the crucial aspects of the integration task is representation: the requirements of robot controllers differ from those of traditional planning systems, and neither representation is usually sufficient to accommodate the needs of an integrated system. For instance, robot systems often use real-valued representations to model features like 3D spatial coordinates and joint angles, allowing robot behaviours to be specified as continuous transforms of vectors over time (Murray, Li, and Sastry 1994). On the other hand, planning systems tend to use representations based on discrete, symbolic models of objects, properties, and actions, described in languages like STRIPS (Fikes and Nilsson 1971) or PDDL (McDermott 1998). Overcoming these differences is essential for building a system that can act in the real world.

In this paper we describe an approach that combines a *cognitive vision* system with a *knowledge-level planner* and *plan execution monitor*, on a robot platform that can manipulate objects in a restricted, but uncertain, environment. Our system uses a multi-level architecture that mixes a low-level robot/vision controller for object manipulation and scene interpretation, with high-level components for reasoning, planning, and action failure recovery. To overcome the modelling differences between the different system components, we use a representational unit called an *Object-Action Complex (OAC)* (Geib et al. 2006; Krüger et al. 2009), which arises naturally from the robot's interaction with the world. OACs provide an object/situation-oriented notion of affordance in a universal formalism for describing state change.

Although the idea of combining a robot/vision system with an automated planner is not new, the particular components we use each bring their own strengths to this work. For instance, the cognitive vision system (Krüger, Lappe, and Wörgötter 2004; Pugeault 2008) provides a powerful object discovery mechanism that lets us induce certain aspects of the representation, suitable for planning, from the robot's basic "reflex" actions. The high-level planner, PKS (Petrick and Bacchus 2002; 2004), is effective at constructing plans under conditions of incomplete information, with both ordinary physical actions and *sensing* actions. Moreover, OACs occur at all levels of the system and, we believe, provide a novel solution to some of the integration problems that arise in our architecture.

This paper reports on work currently in progress, centred around OACs and their role in object discovery, planning with sensing, action execution, and failure recovery in uncertain domains. This work also forms part of a larger project investigating perception, action, and cognition, and combines multiple robot platforms with symbolic representations and reasoning mechanisms. We have therefore approached this work with a great deal of generality, in order to facilitate the transfer of our ideas to robot platforms and planners with capabilities other than those we describe here.

Hardware Setup and Testing Domain

The hardware setup used in this work (see Figure 1) consists of a six-degree-of-freedom industrial robot arm (Stäubli RX60) with a force/torque (FT) sensor (Schunk FTACL 50-80) and a two-finger-parallel gripper (Schunk PG 70) attached. The FT sensor is mounted between the robot arm and gripper and is used to detect collisions which might occur due to limited knowledge about the objects in the world. In addition, a calibrated stereo camera system is mounted in a fixed position. The AVT Pike cameras have a resolution

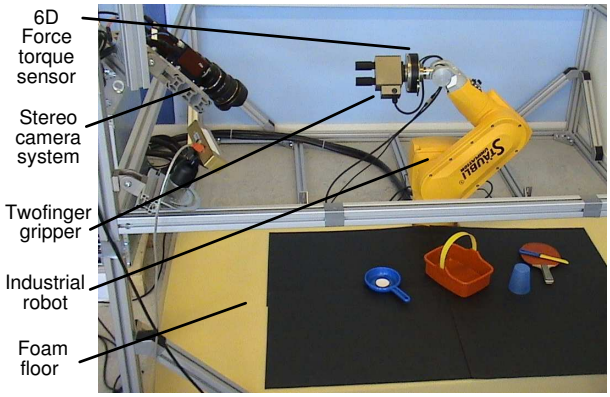


Figure 1: Hardware setup.

of up to 2048x2048 pixels and can produce high-resolution images for particular regions of interest.

To test our approach, we use a Blocksworld-like object manipulation scenario. This domain consists of a table with a number of objects on it and a "shelf" (a special region of the table). The robot can view the objects in the world but, initially, does not have any knowledge about those objects. Instead, world knowledge must be provided by the vision system, the robot's sensors, and the primitive actions built into the robot controller. The robot is given the task of clearing the objects from the table by placing them onto the shelf. The shelf has limited space so the objects must be stacked in order to successfully complete the task. For simplicity, each object has a radius which provides an estimate of its size. An object A can be stacked into an object B provided the radius of A is less than that of B, and B is "open." Unlike standard Blocksworld, the robot does not have complete information about the state of the world. Instead, we consider scenarios where the robot does not know whether an object is open or not and must perform a test to determine an object's "openness". The robot also has a choice of four different grasping types for manipulating objects in the world. Not all grasp types can be used on every object, and certain grasp types are further restricted by the position of an object relative to other objects in the world. Finally, actions can fail during execution and the robot's sensors may return noisy data.

Basic Representations and OACs

At the robot/vision level, the system has a set Σ of sensors, $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where each sensor σ_i returns an observation $obs(\sigma_i)$ about some feature of the world, represented as a real-valued vector. The execution of a robot-level action, called a *motor program*, may cause changes to the world which can be observed through subsequent sensing. Each motor program is typically executed with respect to particular *objects* in the world. We assume that initially the robot/vision system does not know about any objects and, therefore, can't execute many motor programs. Instead, the robot has a set of object-independent *basic reflex actions* which it can use in conjunction with the vision system for early exploration and object discovery.

At the planning level, the underlying representation is based on a set of *fluents*, $f_1, f_2, \ldots, f_m$: first-order predicates and functions that denote particular qualities of the world, robot, and objects. Fluents typically represent high-level versions of some of the world-level properties the robot is capable of sensing, where the value of a fluent is a function Γ_i of a set of observations returned by the sensor set, i.e., $f_i = \Gamma_i(\Sigma)$. However, in general, not every sensor need map to some fluent, and we allow for the possibility of fluents with no direct mapping to robot-level sensors.

Fluents may be parametrized and instantiated by high-level counterparts of the objects discovered at the robot level. In particular, for each robot-level object obj^r we denote a corresponding high-level object by obj^p. A *state* is a snapshot of the values of all instantiated fluents at some point during the execution of the system, i.e., $\{f_1, f_2, \ldots, f_m\}$. States represent an intersection between the low-level and high-level representations and are *induced* from the sensor observations (the Γ_i functions) and the object set.

The planning level representation also includes a set of high-level *actions*, $\alpha_1, \alpha_2, \ldots, \alpha_p$, which are viewed as abstract versions of some of the robot's motor programs. Since all actions must ultimately be executed by the robot, each action is decomposable to a fixed set of motor programs $\Pi(\alpha_i)$, where $\Pi(\alpha_i) = \{mp_1, mp_2, \ldots, mp_l\}$, and each mp_j is a motor program. As with fluents, not every robot-level motor program need map to a high-level action.

Although the robot/vision and planning levels use quite different representations (i.e., real-valued vectors versus logical fluents), the notions of "action" and "state change" are common among these components. To capture these similarities, we model our actions and motor programs using a structure called an *Object-Action Complex (OAC)* (Geib et al. 2006; Krüger et al. 2009). Formally, an OAC is a tuple $\left\langle I, T^S, M \right\rangle$, where I is an identifier label for the OAC, $T : S \rightarrow S$ is a transition function over a state space S, and M is a statistic measure of the accuracy of the transition. OACs provide a universal "container" for encapsulating the relationship between actions (operating over objects) and the changes they make to their state spaces. Each OAC also has an identical set of predefined operations (e.g., composition, update, etc.), providing a common interface to these structures. Since robot systems may have many components, OACs are meant to provide a standard language for describing action-like processes (including continuous processes) within these components, and to simplify the exchange of information between different components.

OACs exist at each level of our system. We encode each motor program on the robot/vision level and each action at the planning level as a separate OAC, with OACs at each level having a different underlying state space. By assigning an accuracy metric to each OAC we also capture the nondeterministic nature of our actions in the real world. Furthermore, since every interaction of the robot with the world provides the robot with an opportunity to observe a small portion of the world's state space (interpreted with respect to the state space of a particular OAC), we can make use of this information to refine or improve the accuracy of the OACs at all levels of our system.

Typically, we consider OACs that are formed from *partial* state descriptions, which may have low reliability. Such descriptions arise since the robot cannot always sense the status of all objects and properties in the world (e.g., occluded or undiscovered objects). Furthermore, the robot's sensors may be noisy and, thus, there is no guarantee that sensor observations are always correct. Certain sensors also have associated resource costs (e.g., time, energy, etc.) which limit their execution. For instance, our robot can perform a test to determine whether an object is open by "poking" the object to check its concavity. Such operations are only initiated on demand at the discretion of the high-level planning system.

Finally, our system includes a middle level component that mediates between the robot and planning levels. This component is responsible for mapping between OACs at different levels of the system (i.e., implementing the Γ_i and Π functions) in order to ensure that observation/state and motor program/action information passing between levels is translated into a form that the destination level understands.

In the remainder of this paper we will look at the main components of our system in greater detail, and describe the current (and future) role of OACs in our framework.

Vision-Based Object Discovery

The visual representation used by the lower level of our system is delivered by an early cognitive vision system (Krüger, Lappe, and Wörgötter 2004; Pugeault 2008) which creates sparse 2D and 3D features, so-called *multi-modal primitives*, along image contours from stereo images. 2D features represent a small image patch in terms of position, orientation, phase, colour and optical flow. These are matched across two stereo views, and pairs of corresponding 2D features permit the reconstruction of an equivalent 3D feature. 2D and 3D primitives are then organized into perceptual groups in 2D and 3D. The procedure to create visual representations is illustrated in Figure 2. We note that the resulting representation not only contains appearance information (e.g., colour and phase) but also geometrical information (i.e., 2D and 3D position and orientation).

Initially, the system lacks knowledge of the objects in a scene and so the visual representation is unsegmented: descriptors that belong to one object are not explicitly distinct from the ones that belong to other objects, or the background. To aid in the discovery of new objects, the robot is equipped with a basic reflex action (Aarno et al. 2007) that is elicited by specific visual feature combinations in the unsegmented world representation (e.g., see Figure 3(a)–(c)). The outcome of these reflexes allows the system to gather knowledge about the scene, which is used to segment the visual world into objects and identify basic affordances. We consider a reflex where the robot tries to grasp a planar surface in the scene. Each time the robot executes such a reflex, haptic information allows the system to evaluate the outcome: either the grasp was successful and the gripper is holding something, or it failed and the gripper simply closed.

With physical control, the system visually inspects an object from a variety of viewpoints and builds a 3D representation (Kraft et al. 2008). Features on the object are tracked over multiple frames, between which the object moves with

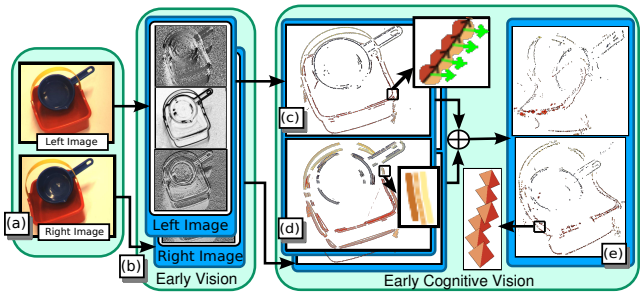


Figure 2: An overview of the visual representation. (a) Stereo image pair, (b) Filter responses, (c) 2D primitives, (d) 2D contours, (e) 3D primitives.

a known motion. If features are constant over a series of frames they become included in the object's representation; otherwise they are assumed to not belong to the object. (See Figure 3(d)–(f) and (Kraft et al. 2008) for a more detailed explanation.) The final description is labelled and recorded as an identifier for a new object class, along with the successful reflex (now a motor program). Using this new knowledge, the system then reconsiders its interpretation of the scene: using a representation-specific pose estimation algorithm (Detry, Pugeault, and Piater 2009) all other instances of the same object class are identified and labelled. By repeating this process, the system constructs a representation of the world objects, as instances of symbolic classes that carry basic affordances, i.e., particular reflex actions that have been successfully applied to objects of this class.[1] This relationship can also be interpreted as a new low-level OAC.

The object-centric nature of the robot's world exploration process has immediate consequences for the high-level representation. First, newly discovered objects are reported to the planning level and added to its representation. At this level, objects are simply labels that act as indices to the object information stored at the robot level. Such a representation means that the planner can avoid reasoning about certain types of real-valued information (e.g., 3D coordinates, orientation vectors, etc.) and instead refer to objects by their labels (e.g., $obj1^p$ may denote a particular red cup on the table). Second, the planner can immediately use such objects during plan generation. Since we assume that object names do not change over time, plans with object references will be understandable to the lower system levels. Finally, the identification of new objects will cause the robot/vision system to start sending regular updates about the state of objects and their properties to the planning level. In particular, low-level observations resulting from subsequent interactions with the world will contain state information about these objects, pro-

[1] We have recently completed the technical implementation of the pose estimation algorithm. Prior to this, a circle detection algorithm was developed (Başeski, Kraft, and Krüger 2009) to recognise cylindrical objects. Four grasp templates were used to define the primitive reflex actions in an object-centric way (where concrete grasps were generated based on the object pose). Although this approach negates the need for the general pose estimation algorithm, the conclusions drawn from experiments in this limited scenario are still easily transferable to the general case.

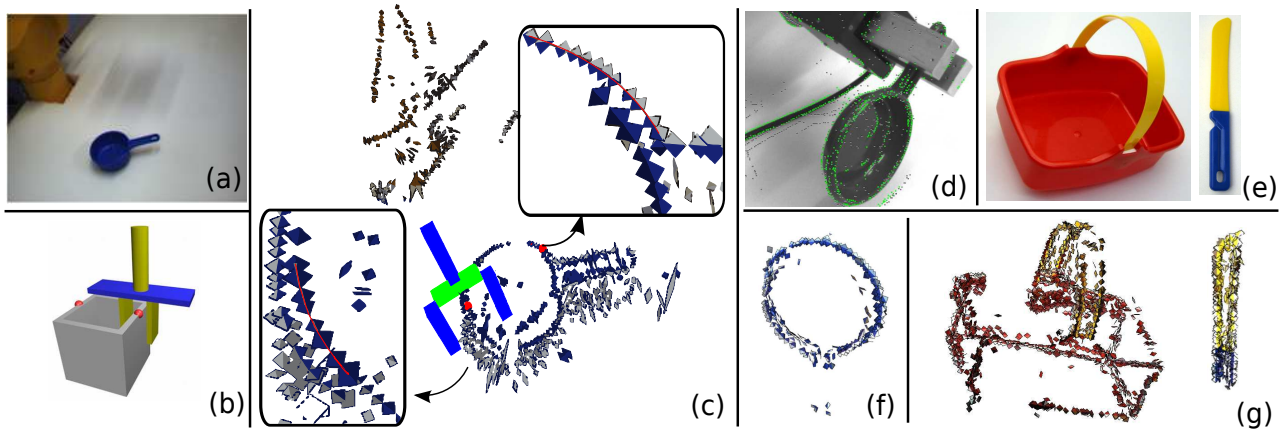


Figure 3: (a)–(c) Initial grasping behaviour: (a) A Scene, (b) Definition of a possible grasp based on two contours, (c) Representation of the scene with contours generating a grasp. (d)–(f) Accumulation process ("birth of the object"): (d) One step in the process. The dots on the image show the predicted structures. Both spurious primitives, parts of the background that are not confirmed by the image, and the confirmed predictions are shown, (e) Images of objects, (f),(g) Extracted models.

vided they can be sensed by the robot.

Knowledge-Level Planning with Sensing

The high-level planner constructs plans that direct the behaviour of the robot to achieve a set of goals. Plans are built using PKS ("Planning with Knowledge and Sensing") (Petrick and Bacchus 2002; 2004), a conditional planner that can operate with incomplete information and sensing actions. Like other symbolic planners, PKS requires a goal, a description of the initial state, and a list of the available actions. Unlike classical planners, PKS operates at the *knowledge level* by explicitly modelling what the planner knows and does not know about the state of the world. PKS can reason efficiently about certain restricted types of knowledge, and make effective use of features like functions, which often arise in real-world scenarios.

PKS is based on a generalization of STRIPS (Fikes and Nilsson 1971). In STRIPS, a single database represents the world state; actions update this database in a way that corresponds to their effects on the world. In PKS, the planner's knowledge state is represented by five databases, each of which stores a particular type of knowledge. Actions describe the changes they make to the database set and, thus, to the underlying knowledge state. PKS also supports ADL-style conditional action effects (Pednault 1989), numerical reasoning, and a set of program-like control structures.

Table 1 shows an example of some of the PKS actions available in the testing domain. As in standard planning representations, like PDDL, actions in PKS are described by their preconditions and effects. Actions may be parametrized (e.g., $graspA(x)$), with an action's parameters replaced with references to specific world objects when an action is instantiated in a plan. As we described above, objects at the planning level are labels to actual objects identified by the robot/vision system.

Preconditions and effects are specified in terms of a set of high-level predicates and functions, i.e., fluents that model particular qualities of the world, robot, and objects. For instance, the actions in Table 1 include references to fluents:

- $open(x)$: object x is open,
- *gripperEmpty*: the robot's gripper is empty,
- $onTable(x)$: object x is on the table,
- $isIn(x, y)$: object x is stacked in object y,
- $radius(x) = y$: the radius of object x is y, and
- $reachableX(x)$: object x is reachable using grasp type X,

among others. While most high-level properties abstract the information returned by the robot-level sensors (e.g., *onTable* requires data from a set of visual sensors concerning object positions), some properties correspond more closely to individual sensors (e.g., *gripperEmpty* closely models a low-level sensor that detects whether the robot's gripper can be closed without contact).

One significant difference between PKS and other planners is that all actions in PKS are modelled at the knowledge level: preconditions denote conditions that must be true of the planner's knowledge state while effects describe changes to what the planner knows. For instance, precondition expressions of the form $K(\phi)$ denote a knowledge-level query that asks "does the planner know ϕ to be true?" while an expression like $K_w(\phi)$ asks "does the planner know whether ϕ is true or not?" Effect expressions of the form $add(D, \phi)$ assert that ϕ should be added to database D, while $del(D, \phi)$ means that ϕ should be removed from database D. In Table 1, K_f refers to a database that models the planner's definite knowledge of facts, while K_w is a specialized database that stores the results of sensing actions that return binary information.

In our robot scenario, high-level actions represent counterparts to some of the motor programs available at the robot level. For instance, the planner has access to actions like:

- $graspA(x)$: grasp x from the table using grasp type A,
- $graspD(x)$: grasp x from the table using grasp type D,
- $putInto(x, y)$: put x into y on the table,

Action	Preconditions	Effects
$graspA(x)$	$K(reachableA(x))$	$add(K_f, inGripper(x))$
	$K(gripperEmpty)$	$add(K_f, \neg gripperEmpty)$
	$K(onTable(x))$	$add(K_f, \neg onTable(x))$
	$K(clear(x))$	
	$K(radius(x) \geq minA)$	
	$K(radius(x) \leq maxA)$	
$graspD(x)$	$K(reachableD(x))$	$add(K_f, inGripper(x))$
	$K(gripperEmpty)$	$add(K_f, \neg gripperEmpty)$
	$K(onTable(x))$	$add(K_f, \neg onTable(x))$
	$K(radius(x) \leq maxD)$	
$putInto(x, y)$	$K(x \neq y)$	$add(K_f, gripperEmpty)$
	$K(inGripper(x))$	$add(K_f, isIn(x, y))$
	$K(open(y))$	$add(K_f, clear(y))$
	$K(clear(y))$	$add(K_f, \neg inGripper(x))$
	$K(onTable(y))$	
	$K(radius(y) > radius(x))$	
$putAway(x)$	$K(inGripper(x))$	$add(K_f, onShelf(x))$
	$K(shelfSpace > 0)$	$add(K_f, gripperEmpty)$
		$add(K_f, \neg inGripper(x))$
		$add(K_f, shelfSpace -= 1)$
$findout\text{-}open(x)$	$\neg K_w(open(x))$	$add(K_w, open(x))$
	$K(onTable(x))$	

Table 1: PKS actions in the testing domain.

- *putAway*(x): put x away onto a shelf space, and
- *findout-open*(x): determine whether x is open or not,

among others. Some actions like "grasp" are divided into multiple actions (e.g., *graspA*, *graspD*, plus actions for grasp types B and C). The object-centric nature of these actions means they do not require 3D coordinates, joint angles, or similar real values but, instead, include parameters that can be instantiated with specific objects. Actions like *putInto* and *putAway* account for different object/location configurations, although the motor programs that implement these actions do not necessarily make such distinctions. (The complete action list has a larger set of such actions.) The *findout-open* action is an example of a high-level *sensing action* that directs the robot to gather information about the world state that is not normally provided as part of its regular sensing cycle. From the planner's point of view, an action's sensory effects are assumed to only change the planner's knowledge state, while leaving the world state unchanged.

Each planning level action is treated as an individual OAC with its own identifier and transition function corresponding to the action's preconditions and effects. All planning level OACs share a common state space consisting of the high-level predicates and functions. Each OAC also maintains a measure, M, of its reliability, which is updated by the plan execution monitor (see below). Currently, PKS does not use this information (or any probabilistic measures) during plan generation, but instead relies on its ability to reason about incomplete information and replan from action failure.

As an example, consider the situation in the testing domain where two unstacked and open objects $obj1^p$ and $obj2^p$ are on a table, the planner can construct the following plan for clearing all *open* objects from the table:

$$\begin{array}{l} graspD(obj2^p), \\ putInto(obj2^p, obj1^p), \\ graspD(obj1^p), \\ putAway(obj1^p). \end{array}$$

In this plan, $obj2^p$ is grasped from the table using grasp type D (an overhand grasp) and put into $obj1^p$, before the stacked objects are grasped and removed to the shelf.

The planner can also build more complex plans using sensing actions. For instance, if the planner is given the goal of removing the *open* objects from the table in the example scenario, but does not know whether object $obj3^p$ is open or not, then it might construct the conditional plan:

$$\begin{array}{l} findout\text{-}open(obj3^p), \\ branch(open(obj3^p)) \\ K^+ : \\ \quad graspA(obj3^p), \\ \quad putAway(obj3^p) \\ K^- : \\ \quad nil. \end{array}$$

This plan senses the truth value of the predicate $open(obj3^p)$ using *findout-open* and reasons about the possible outcome of this action. As a result, two branches are included in the plan denoting potential execution paths: if $open(obj3^p)$ is true (the K^+ branch) then $obj3^p$ is grasped and put away; if $open(obj3^p)$ is false (the K^- branch) then no action is taken.

State Generation and OAC Interaction

From an integration point of view, the robot/vision system is linked to the planning level through a component which mediates between the state spaces and OACs used by the two levels of the system. Since the planner is not able to handle raw sensor data as a state description, or directly control the robot, the low-level observations generated by the robot/vision system must be abstracted into a language the planner understands, and planned actions must be converted into appropriate robot-level motor programs.

For state space information, sensor data is "wrapped" and reported to the planner in the form of a fluent-based symbolic state representation that includes predicates and functions. Currently, the mappings between certain sensor combinations and the corresponding high-level fluents (i.e., the Γ_i functions) are simply hardcoded. For example:

- *inGripper*, *gripperEmpty*: Initially the gripper is empty and the predicate *gripperEmpty* is formed. As soon as the robot grasps an object ($objX^r$), and confirms that the grasp is successful by means of the gripper not closing up to mechanical limits, the system knows that it has the object in its hand and can form a predicate $inGripper(objX^p)$. Releasing the object returns the gripper to an empty state.
- *reachableX*: Based on the position of a circle forming the top of a cylindrical object in the scene we can compute possible grasp positions (for the different grasp types) for each object. Using standard robotics path planning methods we then compute whether or not there is a collision-free path between the start position and the gripper pose needed to reach the object for a particular grasp.

- *open*: Objects are not assumed to be "open." Unlike the above properties which are determined directly from ordinary sensor data, the robot must perform an explicit test to determine an object's openness. In this case, the robot attempts to use its gripper to "poke" inside the potential opening of an object. If the robot encounters a collision (determined by the FT sensor), the object is assumed to be closed. Otherwise, we assume the object is open.

To compute these predicates, the mediator interacts with the robot/vision system to maintain a snapshot of the current world state which, besides the state information necessary for the planner, also contains information needed for consistency and action computations. In particular, object positions are represented here. To cope with sensor noise (especially the vision-based information about the number and location of circles) a simple mechanism to avoid spurious object disappearance and appearance is employed.

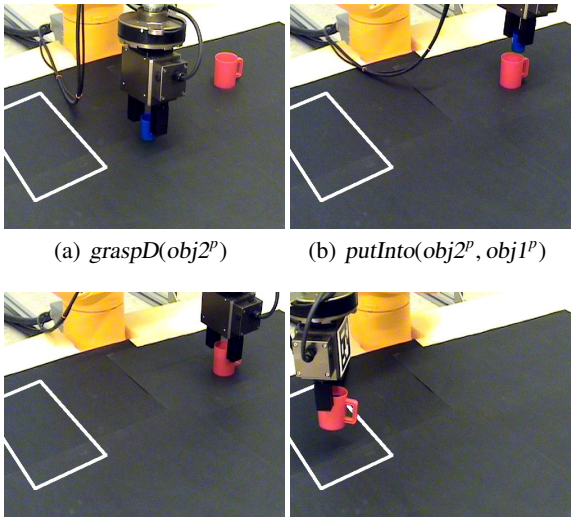
From the planner's point of view, it begins operation without any information about the state of the world. After an initial exploration of the environment, the robot/vision system begins to gather observations and generate (partial) state reports about the current set of objects it believes to be in the world, along with the properties it senses for those objects. This observation set (converted into a fluent-based representation) is then sent to the planner and used as its initial (incomplete) knowledge state: the predicate and function instances are treated as *known* state information, with all other state information considered to be unknown. Subsequent state reports are interpreted by the plan monitor (see below) and used to update the reliability of high-level OACs.

High-level planning actions, in the form of OACs, must also be mapped to their appropriate low-level counterparts, for execution by the robot system in the real world. We currently assume that the set of action schema is supplied to the planner as part of its input, as are the mappings from planning actions to robot motor programs (the Π function).

For instance, the high-level OAC *graspD* is realised on the lowest level as a mapping to an object-independent OAC, $graspD^r$.[2] This low-level OAC requires the object position (retrieved using the object label as an index) as an input to computing suitable grasping positions. The preconditions of this OAC require that there be a grasping position on the brim of the object for which a collision free path from the current position to the grasp position exists. The motor program associated with this OAC is a motion sequence that first completely opens the gripper's fingers, followed by a movement of the arm along the joint trajectory and, lastly, closes the fingers and lifts the arm. After the motor program has been executed the expected outcome state expresses that the fingers should no longer be totally open nor totally closed. In this case, closed fingers indicate that the action failed and no object has been grasped.

Plan Execution and Failure Recovery

Once a plan is generated, the planning level interacts with the robot/vision level (through the mid-level mediator) to execute the plan. Actions are sent to the robot one at a time, where they are converted into motor programs and executed in the world. A stream of observations is also generated, arising from the executed motor programs, and processed into high-level state information. Upon action completion the robot/vision level returns this information to the higher reasoning levels, along with an indication of the success or failure of the action which are used to update the reliability measure M of the high-level OACs. The execution cycle then continues. For instance, Figure 4 shows the execution of the four step plan described above for clearing a table.



(a) $graspD(obj2^p)$ (b) $putInto(obj2^p, obj1^p)$

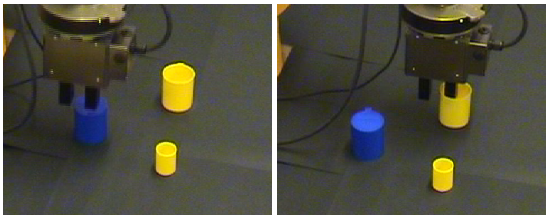
(c) $graspD(obj1^p)$ (d) $putAway(obj1^p)$

Figure 4: Executing a high-level plan to clear a table.

An essential component in this process is the *plan execution monitor*, which assesses action failure and unexpected state information resulting from feedback provided to the planner from the execution of planned actions at the robot level. The execution monitor operates in conjunction with the planner and mid-level mediator, and is responsible for controlling replanning and resensing activities in the system. In particular, the difference between predicted and observed states are used to decide between (i) continuing the execution of an existing plan, (ii) asking the vision system to resense a portion of a scene at a higher resolution in the hope of producing a more detailed state report, and (iii) replanning from an unexpected state using the current state report as a new initial planning state. The plan execution monitor also has the important task of managing the execution of plans with conditional branches, resulting from the inclusion of high-level sensing actions. In each case, the decision of the monitor depends on the type of action being processed and the state information returned by the robot.

Continuing a plan's execution During plan execution, actions are delivered to the lower control levels for execution on the robot. After the execution of each action, a state report representing the *observed* state of the world is returned to the plan monitor and compared against the planner's *predicted* state as constructed during planning, to determine if plan execution should continue or resensing/replanning

[2] In general, a high-level OAC may be realised by multiple robot-level OACs.



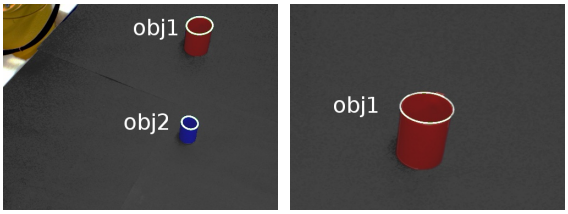
(a) Object is not open (b) Object is open

Figure 5: Testing the openness of an object.

should be activated. Since states in our testing domain tend to be partial, we currently use a limited horizon lookahead method, that attempts to verify that the preconditions for the next n actions in the plan are satisfied in the current (partial) state, and the states that follow when the predicted effects of those actions are applied. (In our testing domain $n = 1$ is often sufficient to ensure good performance.) This means that it is possible for an action to only achieve some of its effects and for the plan to continue, provided the action did not report that it outright failed, and the state is sufficiently correct to ensure the execution of the next action in the plan. (Thus, we defer possible replanning over plan continuation if possible.) If a state match is successful, the monitor then proceeds with the current plan. Otherwise, resensing is considered as a secondary test before replanning (see below).

Sensing actions and conditional plan execution The plan execution monitor also has the added task of managing the execution of plans with sensing actions and associated conditional plan branches. When a high-level sensing action is encountered in a plan it is sent to the robot/vision level like any other action and executed on the robot (as determined by the Π mappings). The actual execution of a sensing action is left to the lower control level which can make more informed decisions about motor program execution. For instance, the *findout-open* action in our example domain is executed at the robot level as a combination of "physical" action (e.g., "poking" an object to determine its openness) and "observational" action (i.e., observing the result); as far as the planner is concerned, the action is executed under the assumption that it is *knowledge producing* and will return an expected piece of information. (Figure 5 shows the execution of *findout-open* by the robot in the case where (a) an object is not open and (b) an object is open.) The sensing result will subsequently be observed by the robot system and returned to the planner as part of the state update cycle.

Plans may also have conditional branch points resulting from sensing actions. When faced with a branch in a plan, the plan execution monitor makes a decision as to the correct plan branch it should execute, based on its current knowledge state. If only partial state information is available, but the required information needed for branch determination is missing (e.g., due to a failure at the robot/vision level), resensing or replanning is triggered. For instance, the example conditional plan given above includes the branch point $branch(open(obj3^p))$, i.e., branch on the truth of the fluent $open(obj3^p)$. If $open(obj3^p)$ is true according to the planner's knowledge state then the "positive" (K^+) branch of the plan is followed and the next action is considered; if



(a) (b)

Figure 6: Resensing the scene using the region of interest capabilities of the high resolution cameras.

$\neg open(obj3^p)$ is true then the "negative" (K^-) branch is followed. If the planner has no information about $open(obj3^p)$ then replanning or resensing is activated. It is important to note that the robot/vision system will never be aware of the conditional nature of a plan, and will never receive a "branch" action. From the point of view of the robot, it will only receive a sequential stream of actions.

Resensing at the monitoring level Sensing also plays a role during plan monitoring as a strategy for improving the monitor's accuracy. When the monitor has determined an action's predicted effects do not match the observed state, resensing is considered. At this point, the accuracy of the action's predictions are checked by comparing the M component of the high-level OAC, weighted together with the M components of the OACs of the underlying motor programs which implement this action (the Π mapping), against a threshold value. If the accuracy measure falls below the threshold (i.e., the predictions are considered too spurious), then replanning is activated; otherwise, resensing is performed.

When resensing is required, the plan monitor provides the vision system with a list of the objects considered relevant to the execution of the action that is reported to have failed, based on the parameters in the high-level action description. This information lets the vision system use its high resolution camera to target particular regions of interest in the scene with greater resolution, to reevaluate the sensors that provide information about these objects. New state information returned by this operation may help the monitor decide between continuing a plan's execution and replanning.

For instance, Figure 6(a) shows the state of the world before the $graspD(obj2^p)$ action in our example plan for clearing a table is executed and $obj2^p$ is grasped; both objects in the scene are correctly detected and identified. After executing $graspD(obj2^p)$, however, it is possible that $obj1^p$ may no longer be detected, leading the monitor to resense both $obj1^p$ and $obj2^p$ since the next action in the plan, $putInto(obj2^p, obj1^p)$, depends on these two objects. In Figure 6(b), the old position of $obj1^p$ is resensed, leading to a rediscovery of the object. The old position of $obj2^p$ is also resensed to confirm that it is no longer on the table. In this case, the conditions in the state are sufficient for the monitor to decide that the next action in the plan can be executed.

Replanning When the monitor determines that an action has failed based on the available (resensed) state information, a new plan is constructed for the given goal using the current state as the planner's new initial knowledge state. We use rapid replanning techniques, rather than plan repair, due

to the success of planners like FF-Replan (Yoon, Fern, and Givan 2007). This technique also provides a way of overcoming PKS's inability to work with probabilistic representations: if a plan fails we direct PKS to construct an alternate plan for achieving the goal. So far this technique has proven to be effective during testing in our example domain.

Discussion and Conclusions

We believe OACs provide a useful tool for overcoming some of the challenges surrounding the representation of affordances, actions, and state change in real-world robot systems: OACs facilitate the description of different system components in terms of a common representation and common set of interfaces. Although we have grounded many of our system components in terms of the OAC concept, and can describe processes like object discovery and action execution in terms of OACs, our work is preliminary and we have not used this representation to its full potential.

For instance, while our OACs maintain a measure of reliability (i.e., the M measure), this property is not significantly used in our system. We are currently exploring how to improve the reliability of lower-level OACs based on state observations, which could in turn "refine" related higher-level OACs. Closely related to OAC update is the idea of learning completely new OACs. To this end, we are investigating how high-level action schema (i.e., planning level OACs) can be learned directly from (partial) state snapshots provided by the robot level (Mourão, Petrick, and Steedman 2008). Furthermore, we would also like to automatically induce the mapping between OACs at different levels. Thus, the OACs in this paper are not as fully featured as those of (Krüger et al. 2009) and implementing the full set of OAC properties remains a future goal of this work.

The robot/vision components of our system are also being improved. After a recent significant increase in the frequency at which the robot/vision level can provide state updates, we are exploring a more sophisticated mechanism to cope with the sensor noise using multiple consecutive updates. In the future we will also investigate whether a probabilistic framework can increase the reliability of the information provided to the planning level. More work is also needed to properly compare our approach to other existing architectures in the literature.

Although this work is preliminary, we have implemented a framework with all the control mechanisms described here. This has enabled us to test our system in a domain similar to the one described in the paper, but with more actions, more objects, and more complex plans. While the results of our initial experiments look promising, we are also in the process of transferring some of our ideas to a humanoid robot that can operate in a real-world kitchen with real-world objects and appliances. This will provide us with a challenging environment to test the scalability of our system and, in particular, our approach to planning and plan execution.

Acknowledgements

This work was partly funded by the European Commission through the PACO-PLUS project (FP6-2004-IST-4-27657).

References

Aarno, D.; Sommerfeld, J.; Kragic, D.; Pugeault, N.; Kalkan, S.; Wörgötter, F.; Kraft, D.; and Krüger, N. 2007. Early reactive grasping with second order 3D feature relations. In *The IEEE International Conference on Advanced Robotics*.

Başeski, E.; Kraft, D.; and Krüger, N. 2009. A hierarchical 3d circle detection algorithm applied in a grasping scenario. In *Proc. of VISAPP-09*, 496–502.

Detry, R.; Pugeault, N.; and Piater, J. 2009. A probabilistic framework for 3D visual object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. To appear.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Geib, C.; Mourão, K.; Petrick, R.; Pugeault, N.; Steedman, M.; Krueger, N.; and Wörgötter, F. 2006. Object action complexes as an interface for planning and robot control. In *IEEE-RAS Humanoids-06 Workshop: Towards Cognitive Humanoid Robots*.

Kraft, D.; Pugeault, N.; Başeski, E.; Popović, M.; Kragic, D.; Kalkan, S.; Wörgötter, F.; and Krüger, N. 2008. 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* 5:247–265.

Krüger, N.; Piater, J.; Wörgötter, F.; Geib, C.; Petrick, R.; Steedman, M.; Ude, A.; Asfour, T.; Kraft, D.; Omrčen, D.; Hommel, B.; Agostini, A.; Kragic, D.; Eklundh, J.-O.; Krüger, V.; Torras, C.; and Dillmann, R. 2009. A formal definition of object-action complexes and examples at different levels of the processing hierarchy. PACO-PLUS Technical Report, available from `http://www.paco-plus.org/`.

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

McDermott, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Mourão, K.; Petrick, R. P. A.; and Steedman, M. 2008. Using kernel perceptrons to learn action effects for planning. In *Proc. of CogSys 2008*, 45–50.

Murray, R.; Li, Z.; and Sastry, S. 1994. *A mathematical introduction to Robotic Manipulation*. CRC Press.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. of KR-89*, 324–332. Morgan Kaufmann.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of AIPS-2002*, 212–221.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proc. of ICAPS-04*, 2–11.

Pugeault, N. 2008. *Early Cognitive Vision: Feedback Mechanisms for the Disambiguation of Early Visual Representation*. Ph.D. Dissertation, Informatics Institute, University of Göttingen.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *Proc. of ICAPS-07*, 352–359.