Proceedings of the 2008 International Conference on Cognitive Systems
University of Karlsruhe, Karlsruhe, Germany, April 2-4, 2008

A System for Learning Basic Object Affordances using a Self-Organizing Map

Barry Ridge, Danijel Skočaj, and Aleš Leonardis
Faculty of Computer and Information Science
University of Ljubljana, Slovenia
{barry.ridge, danijel.skocaj, ales.leonardis}@fri.uni-lj.si

***Abstract*— When a cognitive system encounters particular objects, it needs to know what effect each of its possible actions will have on the state of each of those objects in order to be able to make effective decisions and achieve its goals. Moreover, it should be able to generalize effectively so that when it encounters novel objects, it is able to estimate what effect its actions will have on them based on its experiences with previously encountered similar objects. This idea is encapsulated by the term "affordance", e.g. "a ball affords being rolled to the right when pushed from the left." In this paper, we discuss the development of a cognitive vision platform that uses a robotic arm to interact with household objects in an attempt to learn some of their basic affordance properties. We outline the various sensor and effector module competencies that were needed to achieve this and describe an experiment that uses a self-organizing map to integrate these modalities in a working affordance learning system.**

I. INTRODUCTION

Recent years have seen a surge of activity in the area of developmental robotics [1], a trend that can be seen to underscore the desire to move away from task-specific systems and towards more robust, adaptable platforms and architectures. Desirable traits of such systems include the ability to learn continuously during the course of a lifespan or deployment period, the capacity to construct new concepts from previously learned or known ones, the ability to actively learn via interaction with a tutor or another knowledgeable entity, etc. Naturally, these are difficult problems that are unlikely to be amenable to wholesale solutions, but many interesting, more tractable sub-problems can be identified, one of which is the issue of affordance learning.

The term *affordance* was coined by the psychologist J.J. Gibson [2] to describe the interactive possibilities of a particular object or environment, e.g., "a ball affords rolling" or "a lightswitch affords the illumination of a light bulb". For our purposes here, we will be framing the problem of affordance learning by considering "what will happen if action A_i is performed on object O_j".

In this paper we will present a cognitive vision system that learns basic object affordance properties by interacting with household objects on a table surface using a robotic arm, and observing the result using a camera system. The experimental environment is shown in Fig. 2 and Fig. 3. These devices are integrated over a distributed architecture, shown in Fig. 1. The main idea is to allow the system to perform a variety of simple push actions on objects that are placed on the work surface, record video footage of the result, harvest this data for appropriate features and attempt to learn the similarities inherent in the behaviour of those objects that are physically similar when affected by such actions.

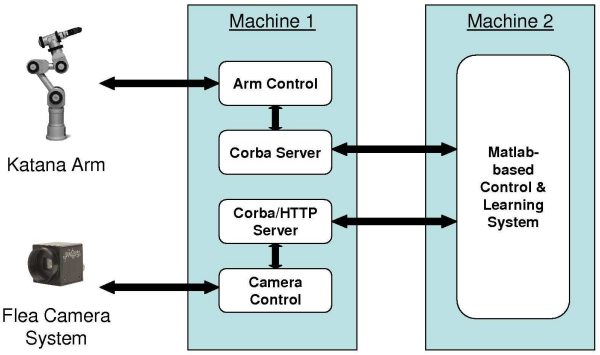


Fig. 1. System architecture.

A number of researchers have sought to develop systems for learning affordances in different settings [3], [4], [5], [6]. In [3], the author devised a similar experiment for affordance learning to the one listed here, where a robot was given a set of 4 actions to perform on 4 toy objects. However, the system only learned one affordance feature, a measurement of how likely it would be for an object to roll along its principal axis; in our case, 11 features are presented to a more generalized learning system. The author of [4] allowed a robotic arm to use a set of tools to manipulate a hockey puck on a work surface and considered two types of affordances; *binding affordances* for potential arm tool attachments, and *output affordances* for the effect that that tool would have on the puck. An object manipulation tool is also used by the robotic system presented here, as detailed in Section IV, but rather than learning the tool affordances, we focus on learning object affordances that become apparent when the tool is used to manipulate different types of objects. An architecture for action (mimicking) and program (gesture) level visual imitation in a robotic platform is presented in [5], where object affordance contexts are used to focus the attention of a gesture recognition system and reduce ambiguities. Gesture recognition in the motor space is performed using

This research has been supported by: Research program P2-0214 (Republic of Slovenia), EU FP6-004250-IP project CoSy and EU MRTN-CT-2004-005439 project VISIONTRAIN.

a Bayesian framework that relies on prior knowledge from object affordance contexts, which are provided. Though we do not perform recognition in this work, rather then providing affordances as a prior, we present a framework in which object affordances may be *learned* dynamically. This knowledge could be subsequently used to aid in various recognition tasks or, indeed, knowledge obtained from an object recognition process could be used as input to our affordance learning system.

Fig. 2. Side view of the workspace. The robotic arm holds a black plastic tool that is used to push objects on the work surface.

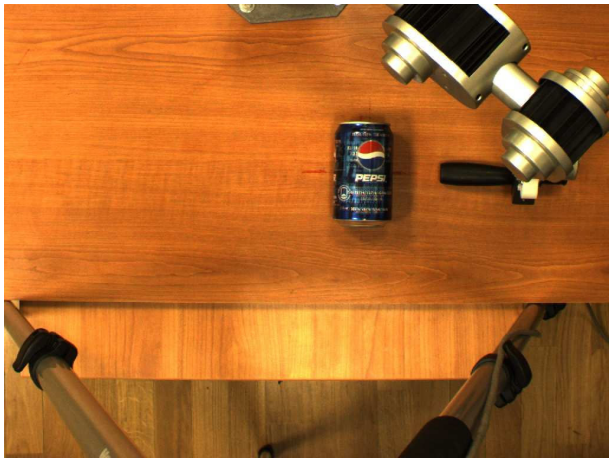


Fig. 3. Workspace as seen by the tripod-mounted camera system.

For the particular experiments presented here, we provide the system with information on what object is present in the scene, as well as with a fixed repertoire of possible arm 'push' actions. Recognition and/or classification of the particular objects involved was not the primary focus of this work, but could be dealt with in a seperate module and integrated into the system. Thus, the task in the experiments is framed as a regression problem where the system receives a feature vector when a particular action is performed on a given object, and tries to both estimate and adjust a target function for that action/object combination. Naturally, a hugely important aspect of such an experimental architecture is that the actions performed by the effector and the features garnered from the sensors be reasonably consistent in nature; otherwise it would be impossible to learn such target functions. We discuss how this is achieved later in Section II.

In principle, many different learning algorithms could be used in the system described above to solve the problem of predicting the resulting feature vectors for particular action/object combinations. We chose to use the *self-organizing map (SOM)* or *Kohenen map* [7] for three main reasons.

Firstly, during training, SOMs form clusters that not only group similar exemplars, but are also related to each other topologically. A SOM is made up of nodes that are linked to other nodes by a map topology and a neighbourhood function. If the node at the center of a data cluster is topologically close to another cluster node then concepts captured by those respective clusters can be said to be similar, whereas clusters that are topologically distant from each other are more likely to represent concepts that are dissimilar.

The second reason why SOMs were chosen is because they can be randomly initialized and trained incrementally without a batch training procedure; the maps self-organize and form clusters as data arrives sample-by-sample. This is an essential requirement for any cognitive system that needs to learn in a continuous, life-long manner.

Thirdly, SOMs have been previously shown to work well for learning affordances, albeit in a simulated robotic system [6]. The authors of [6] used a SOM with a Hebbian mechanism called a *Growing When Required (GWR)* network to aid a simulated Khepera robot in learning affordances of objects with survival values such as nutrition and stamina so that it could prosper over time in its environment.

The paper is organized as follows. In the next section, we outline the system architecture and implementation details. This will aid the discussion in Section III where we highlight how a SOM was used as the learning mechanism for our system. In Section IV we describe an experiment devised to evaluate the system, and finally in Section V we conclude and state our goals for future work.

II. System Architecture & Implementation

A. Robotic Arm

In our system, we use a Neuronics Katana 6M robotic arm which features 5 DC motors for main arm movement, as well as a 6th motor to power a 2 fingered gripper that houses both infrared and haptic sensors (note: these sensors are not used in the experiment presented here). The base of the arm is mounted on a flat table with a wooden laminate surface, and the arm is allowed to move freely in the area above the table surface, avoiding collisions with the table through the use of specialized control software.

1) Interface: The system is designed to be controlled from the Matlab software environment. Matlab was chosen as it allows for rapid prototyping of high-level control programs and provides extensive functionality for computer vision

manipulations as well as other procedures. In order for experiments involving the robot arm to be performed via Matlab and to aid swift cross-platform integration of the arm in future projects, a CORBA interface was developed to sit between the low-level arm control software and high-level Java control client which can easily be called from Matlab. This allows for swift arm/work-space calibration from within Matlab and provides simple *moveTo*(x, y, z) style functionality for moving the end-effector to a localized position (x, y, z) in the workspace.

2) Arm Control Software: At the heart of the arm control architecture lies Golem[1], control and planning software developed specifically for the Katana arm. The software works by transforming desired positions expressed in workspace (x, y, z) coordinate trajectories, to jointspace trajectories, i.e. suitable control signals for each of the 6 arm motors. Golem was originally developed for experiments that involve navigating the arm around workspace obstacles in order to reach objects for grasping. Since our intended use of the arm involved *collision* with objects rather than avoidance, the software had to be modified in two important ways. Firstly, planned movement trajectories had to be constrained to be as linear as possible and secondly, the orientation of the forearm and end-effector had to be made as consistent as possible over repeated movements.

B. Camera System

A Point Gray Research Flea monocular camera (640x480 @ 60FPS or 1024x768 @ 30FPS) was used to gather images and video for the experiment listed in Section IV.

1) Interface: The camera system is operated using a similar interface to that of the robotic arm. A Java client is called from Matlab to interface with a CORBA server that implements the low-level camera functionality. During experiments, after an action command is issued to the robotic arm, the camera system starts recording images and continues recording until movement in the scene has ceased. These images are then used to create a video, which is passed to a compression module, after which it may be gathered from a web server.

2) Tracking System: After video processing, objects are tracked using a probabilistic tracker [8]. This tracker is in essence a color-based particle filter, which makes use of background subtraction using a pre-learned background image. Object shapes are approximated by elliptical regions, while their colour is encoded using colour histograms. The dynamics of objects are modeled using a dynamic model [9], which allows for tracking with a smaller number of particles, and consequently, near real-time tracking performance.

3) Feature Extraction: The following 11 features are extracted from the video data: total distance traveled in x-axis, total distance traveled in y-axis, total Euclidean distance traveled, mean velocity in x-axis, mean velocity in y-axis, velocity variance in x-axis, velocity variance in y-axis, final x position, final y position, final orientation, orientation difference between start orientation and final orientation.

[1]http://www.cs.bham.ac.uk/~msk/

III. Learning with a Self-Organizing Map

A. SOM Description

A SOM [7] is a set of n nodes that are connected to each other via a neighbourhood relation on a low-dimensional (usually 2D) grid. Each ith node contains a d-dimensional weight vector, $\mathbf{m}_i = [m_{i1}, \ldots, m_{id}]$ whose dimension d is equal to the dimension of the data vectors that are provided as input to the SOM during training. The neighbourhood relation may be configured in any number of ways. In the map used for the experiment detailed in Section IV, we used a hexagonal neighbourhood function where each node is connected to six neighbouring nodes, as shown in Fig. 5. Various topologies, e.g., sheet-shaped, cylindrical, toroidal, may also be used to connect map nodes to each other. Depending on the type of neighbourhood relation and the topology imposed, as the SOM is trained weight vectors that are similar with respect to a distance metric, usually Euclidean, will move closer to each other topologically.

B. Incremental Training Algorithm

The weight vectors in each of the n nodes of the SOM are usually randomly initialized before training begins. At each training step, a data vector $\mathbf{x} = [x_1, \ldots, x_d]$ is fed to the SOM and is measured against each node in the SOM using the Euclidean distance metric, as follows:

$$||\mathbf{x} - \mathbf{m}_i||^2 = \sum_{j=1}^{d} w_j (x_j - m_{ij})^2, \quad (1)$$

where w_j is an element of mask vector $\mathbf{w} = [w_1, \ldots, w_d]$ which can be used to block out or discount individual features during training depending on requirements. The node that is closest to the input data vector based on this metric is called the *best matching unit (BMU)* and both it and its neighbouring nodes are updated using the following update rule

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \alpha(t) h_{ci}(t)[\mathbf{x}(t) - \mathbf{m}_i(t)], \quad (2)$$

operating over all $i \in [1, n]$, where $\alpha(t)$ is the learning rate at time t and $h_{ci}(t)$ is the neighbourhood kernel around the BMU c. The neighbourhood kernel is a non-increasing function of time and of the grid-wise distance (distance between nodes on the grid, as opposed to Euclidean distance between their constituent weight-vectors) of node i from the winning node c.

C. SOM Usage in Experiments

For the experiment outlined in the following section, we made use of the publicly available SOM Toolbox 2.0 [2] for Matlab. Feature vectors were collected, each of which had unique text labels associated with them detailing the action/object pairing that produced the feature vector. The text labels that were used in the experiment are shown in Table I. During training, when the BMU for the input data vector is found, the label or labels that are associated with

[2]http://www.cis.hut.fi/projects/somtoolbox/

TABLE I

ACTION/OBJECT PAIRINGS.

Push top / blue cube	Push middle / blue cube	Push bottom / blue cube
Push top / ladybird	Push middle / ladybird	Push bottom / ladybird
Push top / Pepsi can	Push middle / Pepsi can	Push bottom / Pepsi can
Push top / phone	Push middle / phone	Push bottom / phone

that data vector are attached to the weight vector of the BMU. The labels do not affect the weighting of the nodes or the structure of the map, but are useful for classifying the clusters that are formed in the map during training. For our experiments, we use the labels to identify action/object pairings, e.g., "Push top of pepsican" or "Push bottom of phone". As the SOM is being trained, more and more labels get attached to BMU's in the map. In order to predict what feature outcomes are afforded by a particular action/object pairing, the system may search for the node in the SOM that has had the label for that action/object pairing attached to it most frequently. The weight vector at that node forms the affordance prediction for that action/object pairing.

The true power of the SOM in affordance learning lies with its capacity for topological self-organization. If two action/object pairings are sufficiently dissimilar with respect to the feature vectors they produce, the text labels that are associated with them should cluster on nodes in the map that are topologically distant. This allows for different affordance concepts to be neatly captured in an unsupervised way. For example, if a ball is pushed on a table surface, it is likely to roll across the table and produce quite a different feature vector to a box-shaped object that is pushed on the table in a similar way. These feature vectors would likely cluster at topologically distant locations in the SOM, thus capturing the concept of rolling versus non-rolling objects. In the next section we describe an experiment to demonstrate this idea and we evaluate its performance.

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

Fig. 4. Test Objects: (a) Blue cube (b) Ladybird rattle (c) Pepsi can (d) Mobile phone.

IV. EXPERIMENTAL RESULTS

A. Description of Experiment

To test the efficacy of the system for learning basic object affordance properties, the experimental environment was set up as shown in Fig. 2 and Fig. 3, where the Flea camera was positioned roughly one metre above the work surface on a tripod, giving it a top-down viewpoint of the scene. To help avoid arm/object occlusions, which would have posed some difficulties for the tracking system, an black plastic pushing tool was placed in the Katana arm's gripper as shown.

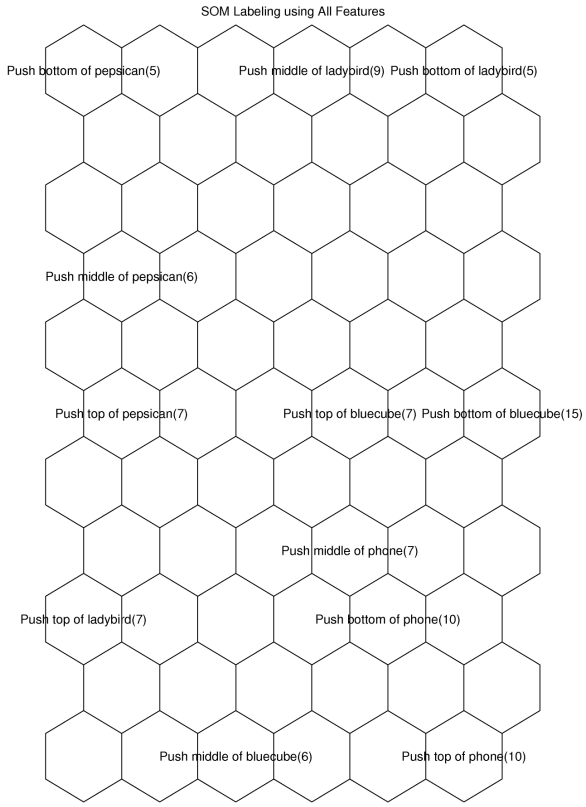


Fig. 5. Example of SOM labeling after 180 training steps using the full dataset. As data vectors with associated text labels are incrementally fed to the SOM, their best-matching node in the map is found, updated, and the text label is added to that node. Over time, labels will gather on specific points in the map and labels that represent action/object pairings with similar properties will be found close to each other. Only the most frequent occurences of each label are shown in this visualization.

A set of 3 pushing actions was provided to the system, each of which involved keeping the forearm part of the arm orthogonal to the work surface and pushing from the right side of the workspace to the left side, through a fixed object start position. One of the actions pushed through the middle of the fixed start position, a second pushed through a point above the start position ("above" with respect to the start location in the image space, not in the arm space) and a third action pushed through a point below the start position. We selected 4 household objects to be used in the experiments as shown in Fig. 4; a blue toy cube, a toy ladybird rattle that is capable of rolling, a Pepsi can, and a mobile phone. During trials, each of these objects was placed centred at the start position with a consistent orientation, as in Fig. 4, and the Katana arm pushed the object at a fixed speed using one of the 3 actions. See Table I for the full range of twelve action/object pairings that were tested.

After an action was performed on an object, the images were gathered from the Flea camera, converted to video, compressed, and passed to the tracking system, as well as the other feature extractors. The 11 extracted features discussed in Section II-B.3 along with the action/object label were then used as input for the SOM during training.

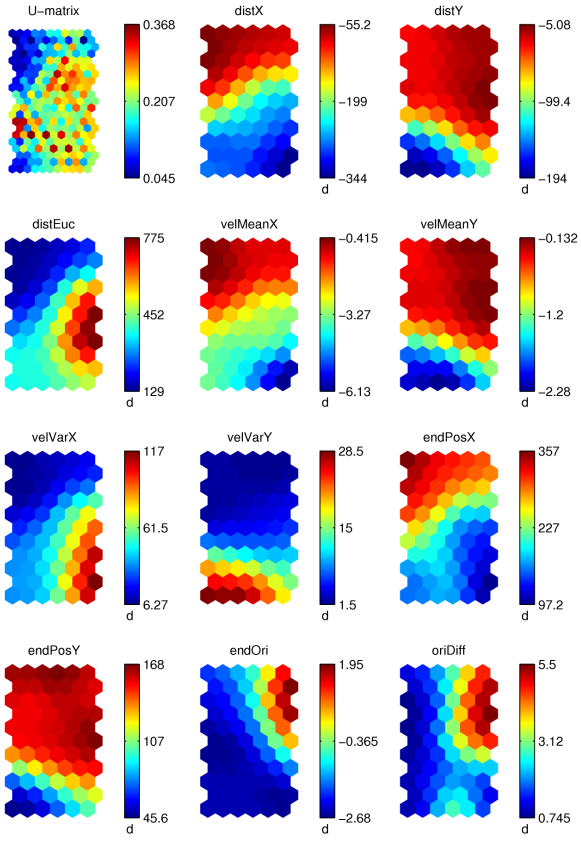


Fig. 6. Example of SOM self-organization after 180 training steps using the full dataset. Cross-sections of the map are shown detailing the learned feature weight values and topological organization for each of the 11 features. A unified distance matrix is shown in the top-left and is useful for visualizing the overall Euclidean distance between adjacent map nodes when considering all features together.

B. Results

In order to evaluate the SOM, 15 object push tests were carried out for each of the 12 action/object pairings listed in Table I and the resulting data was processed, leaving 180 data samples. This dataset was then normalized and an effective SOM size of 66 nodes was determined based on the size of the dataset. A sheet-shaped hexagonal lattice (see Fig. 5 and Fig. 6) was chosen for the SOM and this determined both the neighbourhood function and the topology of the map.

A first test was carried out to measure the generalization capability of the system. The dataset was seperated into a training set of 135 samples with one of the object types omitted and a test set of 45 samples associated with the remaining object. A SOM was trained incrementally using random permutations of the training set and at each training step, the entire test set was used to evaluate the map. For each test sample, the BMU was found in the map and its topological node distance was measured against other nodes in the map whose labels matched the action label associated with the test sample. The labeled nodes with the highest frequencies for each of the training objects were chosen and the distances were recorded and averaged over 20 trials for each test object. Results are shown in Fig. 7. It is clear from the Figure that the SOM quickly captures the difference between rolling and non-rolling object affordances and is able to generalize this knowledge for novel objects. Training objects that generate the smallest map distances to the test objects in this evaluation are considered to be best-matching object types for the respective test objects.

A second test was used to measure the classification ability of the system based on the results of the previous test. The experiment was set up as before, but this time, at each training step the BMU for each test sample was found in the map and the closest labeled map node with the corresponding action label was used to classify the object of the test sample. At each step and for each trial, the number of times this object matched the relevant best-matching training object selected in the previous test was counted and averaged across all 20 trials for each test object. This meant that over time, 9 possible classes emerged in the maps based on the action/object pairings of the training sets. Results are shown in Fig. 8.

V. Conclusion and Future Work

In this paper we discussed the development of a cognitive system equipped with a robotic arm and a camera system that is capable of learning basic object affordance properties. We demonstrated how a self-organizing map may be used as an unsupervised mechanism for classifying action/object pairings with similar affordance properties and we presented experimental results proving the efficacy of this approach.

In future, we hope to replace the text-labeling of objects with a system module that gathers visual features of the objects and uses them as input to the affordance learning system. We would also like to explore the idea of dynamically constructing more complex affordance concepts from basic ones in a developmental learning framework.

References

[1] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini, "Developmental robotics: a survey," *Connection Science*, vol. 15, no. 4, pp. 151–190, 2003.

[2] J. Gibson, *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates, 1986.

[3] P. Fitzpatrick, G. Metta, L. Natale, S. Rao, and G. Sandini, "Learning about objects through action-initial steps towards artificial cognition," *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, 2003.

[4] A. Stoytchev, "Behavior-grounded representation of tool affordances," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 3060–3065.

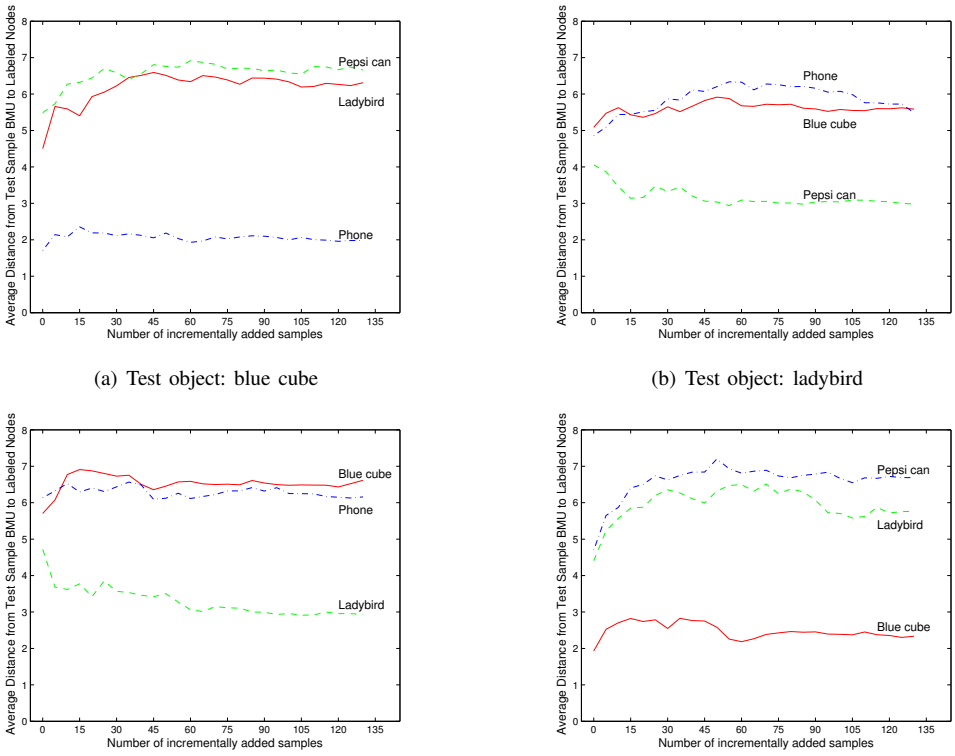
[5] M. Lopes and J. Santos-Victor, "Visual learning by imitation with motor representations," *IEEE Transactions on Systems, Man, and Cybernetics-sPart B: Cybernetics*, vol. 35, no. 3, 2005.

[6] I. Cos-Aguilera, L. Canamero, and G. Hayes, "Using a sofm to learn object affordances," *Proceedings of the 5th Workshop of Physical Agents (WAF), Girona, Catalonia, Spain*, 2004.

[7] T. Kohonen, "Self-organizing maps," *Springer Series In Information Sciences; Vol. 30*, p. 426, 1997.

[8] M. Kristan, J. Pers, M. Perse, and S. Kovacic, "Towards fast and efficient methods for tracking players in sports," in *Proceedings of the ECCV Workshop on Computer Vision Based Analysis in Sport Environments*, May 2006, pp. 14–25.

[9] M. Kristan, J. Pers, A. Leonardis, and S. Kovacic, "A hierarchical dynamic model for tracking in sports," in *Proceedings of the sixteen Electrotechnical and Computer Science Conference (ERK)*, 2007.



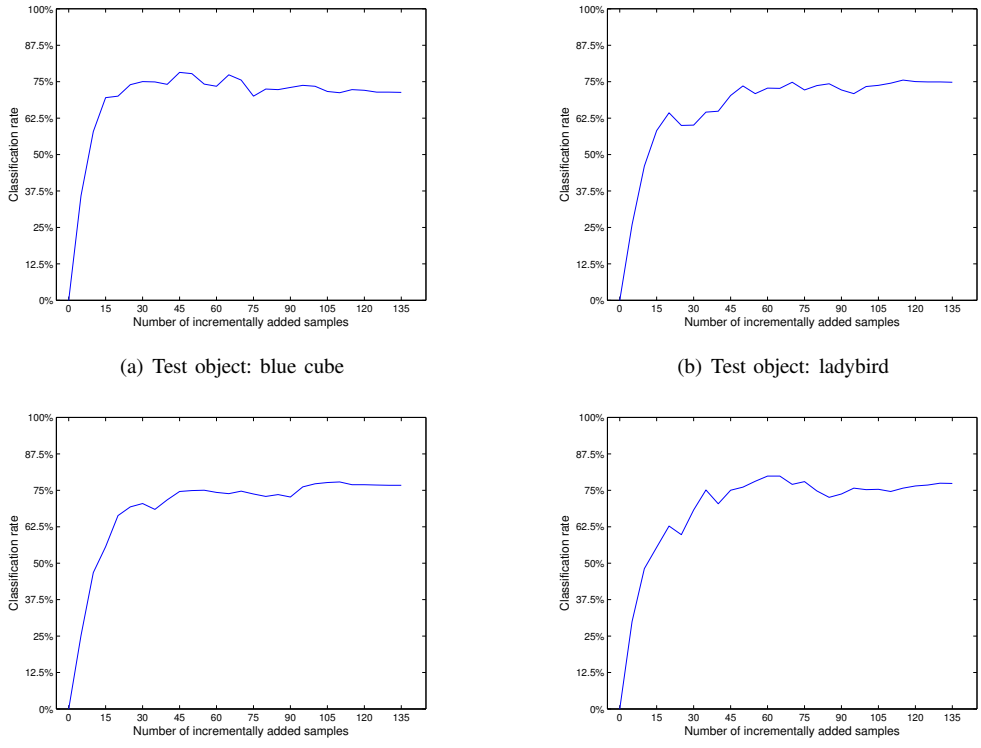
(a) Test object: blue cube

(b) Test object: ladybird

(c) Test object: pepsi can

(d) Test object: phone

Fig. 7. Generalization results for the first test detailed in Section IV-B. The best-matching object type in each of the test cases (a), (b), (c) and (d) was used as the basis for the second test listed in Section IV-B and Fig. 8, e.g., "phone" is the best-matching object for the "blue cube" test object in (a).



(a) Test object: blue cube

(b) Test object: ladybird

(c) Test object: pepsi can

(d) Test object: phone

Fig. 8. Classification results. This evaluation was set up as described in Section IV-B. The y-axes show the percentage of correct classifications for all 45 test samples in each case based on the best-matching objects derived from the the first test evaluation described in Section IV-B and Fig. 7.