International Journal of Advanced Robotic Systems

OPEN ACCESS ARTICLE

Self-Supervised Online Learning of Basic Object Push Affordances

Regular Paper

Barry Ridge[1]*, Aleš Leonardis[2,3], Aleš Ude[1], Miha Deniša[1] and Danijel Skočaj[2]

1 Humanoid and Cognitive Robotics Lab, Department for Automation, Biocybernetics and Robotics, Jožef Stefan Institute, Slovenia
2 Faculty of Computer and Information Science, University of Ljubljana, Slovenia
3 School of Computer Science, University of Birmingham, UK
*Corresponding author(s) E-mail: barry.ridge@ijs.si

Received 24 January 2013; Accepted 20 October 2014

DOI: 10.5772/59654

Abstract

Continuous learning of object affordances in a cognitive robot is a challenging problem, the solution to which arguably requires a developmental approach. In this paper, we describe scenarios where robotic systems interact with household objects by pushing them using robot arms while observing the scene with cameras, and which must incrementally learn, without external supervision, both the effect classes that emerge from these interactions as well as a discriminative model for predicting them from object properties. We formalize the scenario as a multi-view learning problem where data co-occur over two separate data views over time, and we present an online learning framework that uses a self-supervised form of learning vector quantization to build the discriminative model. In various experiments, we demonstrate the effectiveness of this approach in comparison with related supervised methods using data from experiments performed using two different robotic platforms.

Keywords Cognitive and Developmental Robotics, Affordances, Self-supervised Learning, Online Learning

1. Introduction

One of the fundamental enabling mechanisms of human and animal intelligence- and equally, one of the great challenges of modern-day robotics- is the ability to perceive and to exploit environmental affordances [13]. To recognize how to interact with objects in the world, that is to recognize what types of interactions they afford, is tantamount to understanding cause and effect relationships; moreover, from what we know of human and animal cognition, practice and experience help forge a path towards such understanding. This is clear from early childhood development. Through countless hours of motor babbling, children gain a wealth of experience from basic interactions with the world around them, from which they are able to learn basic affordances and, gradually, more complex ones. This is indicative of a continuous learning process involving the assimilation of novel concepts over time, though it is not yet evident precisely how this learning process proceeds. Consequently, implementing such capabilities in a robot is no trivial matter. This is an inherently multi-disciplinary challenge, drawing on such fields as computer vision, machine learning, artificial intelligence, psychology, neuroscience, and others.

Figure 1. An object affordance learning experimental setup using a Katana arm, a Bumblebee stereo camera and a Flea RGB camera

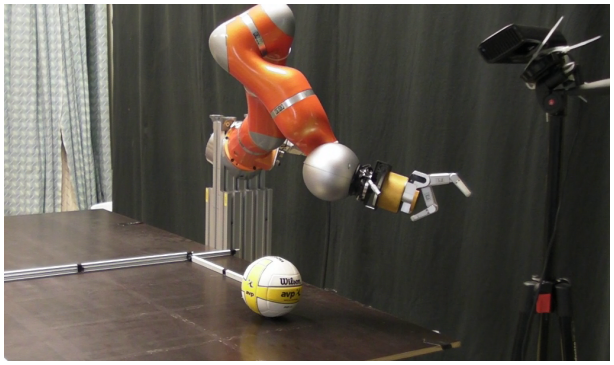

Figure 2. A second experimental setup using a KUKA-LWR arm, a BarrettHand and a Microsoft Kinect RGB-D sensor

For this paper, we approached the problem by using real robotic systems, as depicted in Figures 1 and 2, to perform experiments involving simple push manipulations on household objects. This allowed us to implement and explore the main idea behind our approach to object affordance learning, as visualized in Figure 3. Cameras record images, video and 3D data of these object interactions from which computer vision algorithms extract interesting features, separated into two data views: object features extracted prior to interaction and effect features extracted during and after interaction. These features are used as data for a machine learning algorithm; a self-supervised multi-view online discriminative learner that dynamically forms class clusters in one data view that are used to drive supervised learning in another.

Our main objective in designing this algorithm was that it would enable our robot, when presented with objects, to gradually learn how they behave from experiences interacting with them. When presented with a new object, the algorithm should be able to predict, from object features, how that object will behave in terms of possible effects of actions grounded in effect features. Given feature data from such interactions, we wanted the algorithm to be able to form its own affordance models online dynamically from a naive starting point. To that end, our main requirements were that it adhere to the following learning constraints:

1. The algorithm should be capable of *incremental learning* and should be able to commence learning from scratch without access to an initial batch of training data. This is desirable since, given the complexity and diversity of real-world environments, robots often encounter entirely novel objects and scenarios not contained in any prior training data.
2. Since the robot designer may not know in advance what types of objects the robot will encounter or how they will behave when the robot interacts with them, *task-specific prior object models should be avoided* in favour of providing the system with a sufficiently rich sensory feature set from which to *derive its own models*.
3. The algorithm should *learn autonomously/unsupervised*, such that, as the robot gains experience encountering different objects, acting upon them, and observing the effects, the exploration and exploitation aspects of learning would be interleaved and proceed automatically in the following ways:
   a. Affordance classes grounded in object effect features (e.g., motion features observed both during and after object interaction) would be derived based on qualitative differences between data clusters.
   b. Object features (e.g., shape features observed prior to interaction) that are most relevant for affordance class prediction, would be identified.
   c. Relationships between affordance classes and the predictive object features would be modeled.

Taken together, the above constraints may be regarded as guiding principles for developing a type of self-supervised online discriminative learning, an idea that forms the core of the learning approach in this paper.

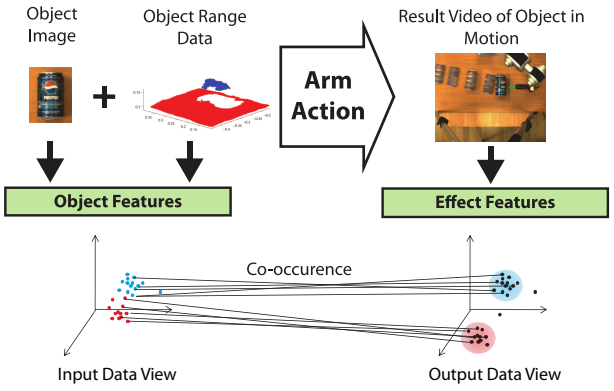

Figure 3. Main idea of our object affordance learning framework

Much of the research work related to the present work stems from the domain of cognitive robotics. Perhaps the most closely related work in the literature with respect to affordance learning to our work is by Fitzpatrick et al. [11]. The authors trained a humanoid robot to recognize "rolling" affordances of four household objects using a fixed set of actions to poke the objects in different direc-

tions, as well as simple visual descriptors for object recognition. There are two main differences between their method and ours. Firstly, in [11, 22], the feature associated with the rolling direction affordance was predetermined, whereas in our system, the learning algorithm is provided with a number of different output features and it must determine for itself the affordance classes within that feature space. Secondly, their system used object recognition to identify the affordances of individual objects, whereas our system determines the affordance class of objects (grounded in object effect features) based, not on their individual identity, but on input features representing a broad set of general object properties (e.g., shape).

Saxena et al. [31] used the same Katana robotic arm used in the present work (see Sec. 3.1) to attempt to grasp novel objects based on a probabilistic model trained on synthetic images of various other objects labelled with grasp points. Later, Detry et al. [7] tackled a similar problem and, rather than training their learning algorithm on synthetically generated objects, they enabled an autonomous robotic arm to grasp objects in an exploratory manner, trained on these interactions with real objects. They described a method for learning object grasp affordance densities, i.e., continuous probabilistic models of object grasp success over object-relative grasp poses. In both of these works the two possible affordances were specified in advance: graspable or non-graspable. The system presented in this paper, by comparison, generates its own affordance classes through interaction with objects. The authors of [35] worked with a robotic system consisting of a range scanner and a robotic arm that learned affordances of objects in a table-top setting using an unsupervised two-step approach of effect class discovery and discriminative learning for class prediction. They also applied similar techniques to a scenario involving self-discovery of motor primitives and learning grasp affordances [36].

Our learning approach is based on vector quantization via the use of self-organizing maps (SOMs) [17], which have been employed in past works on affordance learning in various different ways [6, 26, 32]. In [6], a SOM was applied to a simulated mobile robot learning how to prosper in an artificial environment by exploiting affordances of objects with survival values, such as nutrition and stamina. The SOM was used to cluster visual sensor data in the input space where nodes were assigned weights based on the success or failure of actions. In our case, by comparison, SOMs are used in two separate feature spaces in a multi-view learning configuration similarly to recent work by Sinapov et al. [32], where SOMs were used to cluster data in both the proprioceptive and auditory sensory streams of a humanoid robot while performing performing various exploratory behaviours on objects. In [24], the authors used a humanoid robot to push, grasp and tap objects on a table as well as a Bayesian network to form associations between actions, objects and effects. Though the goals were similar to those expressed in this paper, the learning method is less amenable to online learning, as a certain amount of data must be gathered initially to find categories before the network can be trained.

In the next section, we give an overview of our proposed learning algorithm, which was developed to adhere to the learning constraints outlined previously. The remainder of the paper consists of Section 3, where we review the robotic systems we used for performing experiments, Section 4, where we describe the experiments themselves, and Section 5, where we provide concluding thoughts.

2. Self-supervised Multi-view Online Learning

Multi-view learning [33], sometimes also referred to using the terms 'cross-modal learning', 'multi-modal learning' or 'co-clustering' [9, 2, 1, 4, 8] is an area of machine learning where, rather than having learning performed on data in a single feature space, it is instead performed over multiple separate feature spaces, otherwise known as 'data views' or 'modalities', in which data co-occur. Given this common theme, the learning goal may otherwise differ depending on the particular context [33]. In our scenario, object properties such as shape features define the feature space in one data view, the input space $X \subseteq \mathbb{R}^m$, whereas object effects under interaction, such as motion features and changes in shape features, define the feature space in another data view, the output space $Y \subseteq \mathbb{R}^n$. We assume that matching data co-occur in each of them.

Our learning goal is to find significant clusters in Y that may be projected back to X and used as class labels to train a classifier, thus forming a mapping $f : \mathbb{R}^m \rightarrow \mathbb{N}$ from input space feature vectors to class labels representing affordances grounded in output space feature clusters. We consider this as a multi-view learning problem given that there is a natural separation between the two feature spaces under consideration, which model potential causes and potential effects respectively, and also as a self-supervised learning problem given that, if the online learning constraints described in the introduction are to be adhered to, the class clusters must be discovered and exploited for discriminative learning both dynamically and autonomously. To this end, we employed a similar form of an algorithmic framework for self-supervised multi-view learning originally proposed in [27][1] Assuming we have two different modalities or data views, each of them might yield two datasets of co-occurring data, $X = \{\mathbf{x}_i \in \mathbb{R}^m \mid i = 1, \ldots, D\}$ in the input space and $Y = \{\mathbf{y}_i \in \mathbb{R}^n \mid i = 1, \ldots, D\}$ in the output space, where we work under the assumption that the $\mathbf{x}_i$ and $\mathbf{y}_i$ data vectors are not all available at once and that they arrive in an online data stream. We aim to represent each of the data views via vector quantization [17, 21] using codebooks of

1 Source code has been released under the GNU/GPL license at https://github.com/barryridge/SSLVQ.

prototype vectors $W = \{\mathbf{w}_j \in \mathbb{R}^m \mid j = 1, \ldots, M\}$ for the input space and $V = \{\mathbf{v}_k \in \mathbb{R}^n \mid k = 1, \ldots, N\}$ for the output space, respectively, approximating the data distributions in each view, We adopt the same structural approach taken in [4], [23] in that we define full network connectivity between the two codebooks with what we refer to as *a Hebbian co-occurrence mapping* as follows:

$$H(W,V) = \left\{\gamma_{j,k} \in \mathsf{R} \middle| \mathbf{w}_j \in W, \mathbf{v}_k \in V, \forall j,k\right\}, \tag{1}$$

where the γ_{jk} are weights that are used to record the level of data co-occurrence between nearest-neighbour prototypes in each of the codebooks.

Thus, we aim to find significant clusters of prototypes in the output view (effect features) which we dub *class clusters* and which we treat as classes to be used for discriminative learning in the input-view (object features). However, these clusters are not fully-formed during online codebook training; thus, we use the information contained within the codebook network to infer their development and to guide the training process until the clusters are derived outside of training during classification. Codebook training within the input view, therefore, uses two learning phases. The first phase (cf. Section 2.1), involves unsupervised clustering of the prototypes such that the data distributions are roughly approximated. The second phase (cf. Section 2.2), involves self-supervised discriminative learning such that the positions of the prototypes are refined for classification purposes using cross-view co-occurrence information. The classification process, which may be applied at any given time-step during training and involves clustering the output view prototypes to discover the class clusters, is described below in Section 2.3. Finally, in Section 2.4, we describe feature relevance determination mechanisms that allow the algorithm to both infer and exploit the most relevant features in each view for both class discovery and class prediction.

2.1 Training Phase 1: Unsupervised Learning

For the first phase of codebook training, we employ the SOM algorithm [17] to perform unsupervised vector quantization of the prototypes in W and V. We describe this learning procedure with respect to input view codebook W only without loss of generality; the same algorithm is applied to output codebook V. The prototypes in W partition the input space into Voronoi regions or receptive fields such that each prototype represents those data vectors for which it is the nearest neighbour, and that thus fall within its receptive field as dictated by the squared Euclidean distance:

$$d^2(\mathbf{x},\mathbf{w}) = \sum_{i=1}^{m} \lambda^i (x^i - w^i)^2, \tag{2}$$

in which case the definition of the receptive field for prototype $\mathbf{w}_c$ would be

$$R_{\mathbf{w}_c} = \left\{\mathbf{x} \in X \middle| d^2(\mathbf{x},\mathbf{w}_c) \le d^2(\mathbf{x},\mathbf{w}_j) \forall \mathbf{w}_j \in W\right\}. \tag{3}$$

In (2) above, the λ^i are weighting factors for each dimension, which allows for the possibility of an adaptive distance function for the purposes of feature relevance determination, as discussed later in Section 2.4.

Given $\mathbf{x}$, the update rule for prototypes in W using the SOM algorithm is:

$$\mathbf{w}_j^{t+1} = \mathbf{w}_j^t + \alpha_W^t h_{\mathbf{w}_c}^t(\mathbf{w}_j^t)(\mathbf{x} - \mathbf{w}_j^t), \tag{4}$$

where h_t is a weighted neighbourhood function (cf. (5) below) for the nearest-neighbour prototype $\mathbf{w}_c$, that is, the closest prototype to data sample $\mathbf{x}$ at time t, and α_W^t is the learning rate at time t. The topology and neighbourhood function may in principle be defined almost arbitrarily, but often follow *de facto* definitions in the literature [17]. In our case, the prototypes are arranged in a rectangular sheet-shaped lattice such that, given prototype $\mathbf{w}_j$, it would have an associated location vector $\mathbf{r}_j \in \mathbb{R}^2$ governing its position within the lattice. We employ a Gaussian neighbourhood kernel function:

$$h_{\mathbf{w}_c}^t(\mathbf{w}_j) = \exp\left(-\frac{d(\mathbf{r}_c,\mathbf{r}_j)}{2\sigma_t^2}\right) \tag{5}$$

where the vectors $\mathbf{r}_c \in \mathbb{R}^2$ and $\mathbf{r}_j \in \mathbb{R}^2$ define 2D lattice locations of the nearest-neighbour prototype $\mathbf{w}_c$ and prototype $\mathbf{w}_j$, respectively, and the parameter σ_t specifies the width of the kernel at time t. We can take a probabilistic interpretation of the activation distribution over an entire codebook by exploiting the neighbourhood kernel function- as follows- to define:

$$P(\mathbf{w}_j \mid \mathbf{x}) = \frac{h_{\mathbf{w}_c}^t(\mathbf{w}_j)}{\sum_i^M h_{\mathbf{w}_c}^t(\mathbf{w}_i)}, \tag{6}$$

which is the conditional probability of the activation of prototype $\mathbf{w}_j \in W$ given data sample $\mathbf{x}$. We also define:

$$A_W(\mathbf{x}) = \{P(\mathbf{w}_1 \mid \mathbf{x}), P(\mathbf{w}_2 \mid \mathbf{x}), \ldots, P(\mathbf{w}_M \mid \mathbf{x})\} \tag{7}$$

to be the discrete spatial probability density function of the activation of codebook W given $\mathbf{x}$. Analogous rules and

definitions apply to the output view codebook V. A method of exploiting the topological structure of the SOM is provided in [23] such that, not only are the Hebbian weights between the winning prototypes in each view updated, but so too are those weights between the neighbours of the winning prototypes. We make use of this idea as well as (5) to apply the following update [23, 16] to all weights in $H(W,V)$ at each training step:

$$\gamma_{jk}^{t+1} = \gamma_{jk}^{t} + \alpha_H^t h_{\mathbf{w}_c}^t(\mathbf{w}_j) h_{\mathbf{v}_c}^t(\mathbf{v}_k) \tag{8}$$

where α_H^t refers to a learning rate for the co-occurrence mapping, which may be specified differently from the learning rates for the individual codebooks.

2.2 Training Phase 2: Self-supervised Learning

After the unsupervised training phase has proceeded for long enough to provide a robust Hebbian mapping, the self-supervised training phase may be initiated. In this phase the output view codebook continues to be trained with the usual SOM algorithm, while the input view codebook switches to a modified version of learning vector quantization (LVQ) [17] training that employs a cross-view probabilistic supervision signal to improve its discriminative model. To develop this idea, we make use of two additional theoretical concepts, cross-view Hebbian projection and the Hellinger distance, which we discuss next. Phase 2 training is illustrated in Figure 4.

2.2.1 Cross-view Hebbian Projection

The Hebbian co-occurrence mapping can be used to map the relationship between the prototypes in the different views, and one way to achieve this is through the use of Hebbian projection [4, 5]. A Hebbian projection is a spatial probability distribution over a codebook and is the result of selecting a prototype in one codebook and normalizing the Hebbian co-occurrence weights that map from it onto another codebook. This is a useful tool that allows us to measure how one data view looks from the perspective of another in terms of past co-occurrences of data. Taking our two data view codebooks W and V as before, given prototype $\mathbf{w}_j \in W$ we define:

$$P(\mathbf{v}_k \mid \mathbf{w}_j) = \frac{\gamma_{jk}}{\sum_j^N \gamma_{jk}} \tag{9}$$

to be the conditional probability of data co-occurring in the receptive field of prototype $\mathbf{v}_k \in V$ given data occurring in the receptive field of prototype $\mathbf{w}_j \in W$. We may now define a Hebbian projection from prototype $\mathbf{w}_j$ in codebook W to codebook V as:

$$\vec{H}_W^V(\mathbf{w}_j) = \{P(\mathbf{v}_1 \mid \mathbf{w}_j), P(\mathbf{v}_2 \mid \mathbf{w}_j), \ldots, P(\mathbf{v}_N \mid \mathbf{w}_j)\}. \tag{10}$$

2.2.2 Measuring Similarity Between Data Views

Using the definition from [12], for a countable state space Ω and given probability measures μ and ν :

$$d_H(\mu,\nu) := \left[\sum_{\omega\in\Omega}\left(\sqrt{\mu(\omega)} - \sqrt{\nu(\omega)}\right)^2\right]^{\frac{1}{2}}. \tag{11}$$

We use this to create a heuristic that allows us to measure the similarity between prototypes in the input view codebook and the output view codebook with respect to the Hebbian mapping. The Hellinger distance takes values in the bounded interval $[0,\sqrt{2}]$, making it amenable to statistical analysis (e.g., calculating the mean distance). Thus, given the input view training sample $\mathbf{x}$, if we measure $d_H\left(\vec{H}_W^V(\mathbf{w}_j), A_V(\mathbf{y})\right)$, the Hellinger distance between the Hebbian projection from a given prototype $\mathbf{w}_j \in W$ to V and activation distribution over V given output view sample $\mathbf{y}$, we can get an impression of how well $\mathbf{w}_j$ predicts the activity of the output view codebook given $\mathbf{x}$. Now that we have the necessary tools in place, we may proceed to present our modified LVQ algorithm.

2.2.3 Self-Supervised Learning Vector Quantization

In traditional LVQ training [17], prototypes are given fixed class labels a priori. Subsequently, as training samples with accompanying class labels are encountered, the nearest neighbour prototypes are updated according to a set of update rules. If the prototype class label matches that of the training sample, the prototype vector is moved towards the sample. If the labels do not match, the prototype vector is moved away from the sample. In the modified form of LVQ we present here, we do not label the prototypes a priori. Given co-occurring ($\mathbf{x}$,$\mathbf{y}$) samples, input view prototypes are updated based on a supervision signal formed by $d_H\left(\vec{H}_W^V(\mathbf{w}_c), A_V(\mathbf{y})\right)$, the Hellinger distance between the Hebbian projection from the nearest neighbour prototype $\mathbf{w}_c \in W$ to V and the activation distribution over V given output view sample $\mathbf{y}$. If the distance is small, $\mathbf{w}_c$ is moved closer to $\mathbf{x}$, whereas if the distance is large, $\mathbf{w}_c$ is moved further away as follows:

$$\mathbf{w}_c^{t+1} := \begin{cases} \mathbf{w}_c^t + \alpha_L^t(\mathbf{x}^t - \mathbf{w}_c^t) \\ \textit{if } d_H(\vec{H}_W^V(\mathbf{w}_c), A_V(\mathbf{y})) < \varepsilon - \delta, \\ \\ \mathbf{w}_c^t - \alpha_L^t(\mathbf{x}^t - \mathbf{w}_c^t) \\ \textit{if } d_H(\vec{H}_W^V(\mathbf{w}_c), A_V(\mathbf{y})) \geq \varepsilon + \delta, \\ \\ \mathbf{w}_c^t \text{ otherwise.} \end{cases} \tag{12}$$

where, α_L^t is the LVQ learning rate and, assuming $d_H\left(\vec{H}_W^V(\mathbf{w}_c), A_V(\mathbf{y})\right)$ is normalised, ε is some decision threshold value with uncertainty window $\pm\delta$.

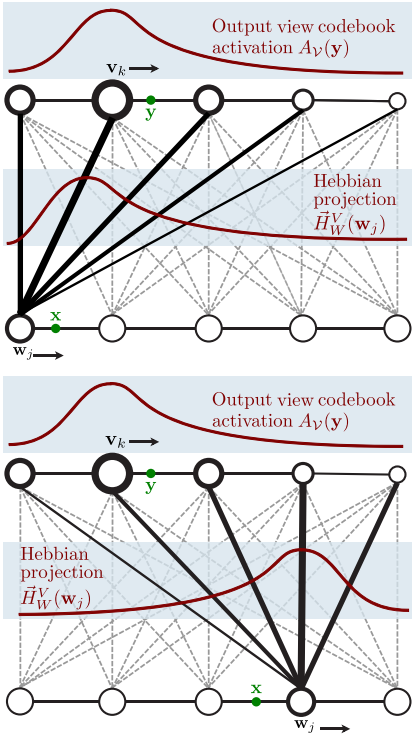

Figure 4. Visualisations of the self-supervised training rules from (12). The first figure shows the case when $d_H\left(\vec{H}_W^V(\mathbf{w}_c), A_V(\mathbf{y})\right) < \varepsilon - \delta$ and the second, when $d_H\left(\vec{H}_W^V(\mathbf{w}_c), A_V(\mathbf{y})\right) \geq \varepsilon + \delta$.

2.3 Cross-View Classification

For cross-view classification, we require a mapping $f : \mathbb{R}^m \to L\ (V)$ that maps input view samples to class labels, where $L\ ()$ is some labelling function. Our approach to this is as follows. After clustering the data via vector quantization using the prototypes in each of the separate data views, we wish to form class clusters by clustering the prototypes themselves in the output view which we treat as classes that may be projected back to the input view prototypes as class labels. The means for achieving this are described in the following sub-sections.

2.3.1 Class Discovery

In order to find the class clusters of prototypes, we treat the prototype vectors as data points and employ traditional unsupervised clustering, specifically, the k-means clustering algorithm. One issue with regular k-means is that k must be selected in advance. It is possible, however, to augment the algorithm such that k is selected automatically. We achieve this by running k-means for multiple different values of k, then evaluating each of the clusterings using multiple different cluster validity indices, and finally selecting the k^*-value which is most consistently selected as the best value by these. The validity indices we use are Davies-Bouldin [3]; Dunn [3]; Calinski-Harabasz [10]; Krzanowski-Lai [10]; and the silhouette index [3]. Thus, given multiple potential k-clusterings $K_1^V, \ldots, K_k^V, \ldots$ as generated by k-means, e.g. for $k = 1, \ldots, 10$, where $K_k^V = \{C_1^V, \ldots, C_k^V\}$ and the C_i^V are the clusters of prototypes $\mathbf{v} \in V$, the five validity indices are computed for each k and vote for an optimal k value. The k-value with the most votes, k^*, is selected, and $K_{k^*}^V$ becomes the selected clustering.

2.3.2 Cross-View Class Projection

For cross-view classification, we require a mapping $f : \mathbb{R}^m \to L\ (V)$ that maps input space samples to class labels, where $L\ ()$ is some labelling function. To realise this labelling function, the $K_{k^*}^V = \{C_1^V, \ldots, C_{k^*}^V\}$ class clusters found in V via class discovery are projected back to the input space codebook W. Given an input view test sample $\mathbf{x}$ to be classified, the nearest-neighbour prototype in the input view layer is found and its Hebbian weight links are mapped to the output view class clusters via Hebbian projection, as visualized in Figure 5.

By summing the posterior probabilities $P(\mathbf{v}_k \mid \mathbf{w}_j)$ provided by such a projection, we can determine the posterior probability of class cluster C_i^V in output view codebook V given prototype $\mathbf{w}_j$ in input view codebook W as follows

$$P(C_i^V \mid \mathbf{w}_j) = \int_{R_{C_i^V}} P(C_i^V \mid \mathbf{w}_j) P(\mathbf{w}_j) d\mathbf{v} \tag{13}$$

$$= \sum_{\mathbf{v}_k \in C_i^V} P(\mathbf{v}_k \mid \mathbf{w}_j) P(\mathbf{w}_j) \tag{14}$$

where $R_{C_i^V} = \cup_{\mathbf{v}_k \in C_i^V} R_{\mathbf{v}_k}$ is the receptive field for class cluster C_i^V. This allows us to assign an output view class cluster label to each of the prototypes in the input view codebook by maximizing the class cluster posterior probability for each of them. Thus, given $\mathbf{w}_j$, we define a labelling function

$$L(\mathbf{w}_j) = argmax_{i=1,\ldots,k^*} P(C_i^V \mid \mathbf{w}_j) \tag{15}$$

that labels the input view prototypes on that basis and we may then also define class clusters $K_{k^*}^W = \{C_1^W, \ldots, C_{k^*}^W\}$ in the input view where $C_i^W = \{\mathbf{w}_j \in W \mid L\ (\mathbf{w}_j) = i\}$.

2.3.3 Class Prediction

Given an input space test sample $\mathbf{x}$, its predicted output space class cluster may finally be determined using the labelling function from (15), the weighted squared Eucli-

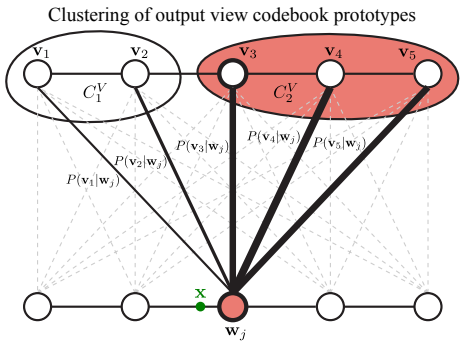

Figure 5. Cross-view classification

dean distance function from (2), and the nearest-neighbour rule as follows:

$$f(\mathbf{x}) = L\left(\arg\min_{\mathbf{w}_j \in W} d^2(\mathbf{x}, \mathbf{w}_j)\right). \tag{16}$$

2.4 Feature Relevance Determination

Some feature dimensions can prove to be more relevant than others, both for class discovery and class prediction, and determining the extent of their relevance and exploiting this information can improve accuracy. To this end, we make use of the Fisher criterion score, a concept that has been shown to be useful for feature relevance determination in learning vector quantization settings in previous work [25].

2.4.1 Input Feature Relevance

We exploit the positioning of the prototypes in the input feature space to estimate Fisher criterion scores for the input dimensions. These scores, once normalised, are used as the λ^i weighting factors in (2) for an adaptive distance function that accounts for dimensional relevance with respect to classifier output. To summarise this, the Fisher criterion score $F^i(X)$ may be estimated for the i-th dimension of X as

$$\lambda^i \propto \frac{S_B^i\left(K_{k^*}^W\right)}{S_W^i\left(K_{k^*}^W\right)} \approx F^i(X), \tag{17}$$

where the λ^i are assigned to normalized values of the fractional term, of which

$$S_B^i\left(K_{k^*}^W\right) = \sum_{c=1}^{k^*} \frac{M_c}{M}\left(\hat{w}_c^i - \hat{w}^i\right)^2 \tag{18}$$

is the estimated between-class variance over the i-th dimension summed over each class $c=1,\ldots,k^*$, where M and M_c are the cardinalities of the entire set of prototypes W and the c-class prototypes C_c^W respectively, and

$$\hat{w}^i = \sum_{\mathbf{w}_j \in W} p_j w_j^i \quad \text{and} \quad \hat{w}_c^i = \sum_{\mathbf{w}_j \in C_c^W} p_j w_j^i \tag{19}$$

are weighted means over the i-th dimension of the prototypes in W and C_c^W respectively, where p_j is a weighting factor that is proportional to the number of times prototype $\mathbf{w}_j$ was the nearest-neighbour during training, and

$$S_W^i\left(K_{k^*}^W\right) = \sum_{c=1}^{k^*} \frac{M_c}{M} \sum_{\mathbf{w}_j \in C_c^W} p_j\left(w_j^i - \hat{w}_c^i\right)^2 \tag{20}$$

is the estimated within-class variance over the i-th dimension.

2.4.2 Output Feature Relevance

The obvious application of the Fisher criterion score lies in the input space where, after a given point in training when class prediction is attempted, the class clusters discovered in the output space are projected onto the input space prototypes as class labels, thereby making the class information available a priori. However, we also make use of it in the output space to augment the class discovery process. In the output space, even though the class cluster structure is initially unknown, it is still possible to select features that are more likely to be relevant to forming good cluster hypotheses. Firstly, given i, we split the i-th dimension of V into multiple histogram partitions $Q_1^V,\ldots,Q_k^V,\ldots$, where each $Q_k^V = \{B_1^{V_i},\ldots,B_k^{V_i}\}$ subdivides V along the range of i into k evenly-sized bins containing V-prototypes. Then using Q_k^V in place of $K_{k^*}^W$ in (17), calculating (17) for a range of k values (we used k=2,4,6,8,10 in our experiments), averaging over k, and assigning a λ^i weight to the k-averaged value, we may select features over a certain threshold (we used the λ^i-average in our experiments) that are likely to contribute most to good clustering over a range of k-values. We use then these features as input for the clustering algorithm described in Section 2.3.1.

3. Robot and Vision Systems

We used two different experimental platforms for our experiments. Section 3.1 below gives an overview of each of these setups, after which Section 3.2 discusses the object push actions, while Sections 3.3 and 3.4 describe our visual feature extraction implementations for object features and effect features respectively. The feature choices were motivated by the types of affordances we intended on studying in our experiments: rolling and translating affordances with the first platform using both flat and

curved-surfaced objects, and rolling, translating and toppling affordances in experiments with the second platform using both flat-surfaced and spherical objects.

3.1 Experimental Platforms

In the first setup we used a 5-DOF Neuronics Katana 6M robotic arm for manipulation, as well as two Point Gray Research cameras for vision- the Flea monocular camera and the Bumblebee 2 grayscale stereo camera, which recorded images, video and 3D point clouds from range data. The arm was mounted on a flat wooden table and was made to produce linear pushing motions via the use of a modified version of the Golem [18], control software for the Katana arm. Further details are described in [27]. In the second setup, as depicted in Figure 2, a 7-DOF KUKA-LWR with an attached 3-fingered Barrett Hand was used for object push interactions, while a Microsoft Kinect RGB-D sensor was used to gather 3D point clouds of the scene. The Point Cloud Library (PCL)[2] was used to extract and manipulate object clouds from the resulting data.

3.2 Object Push Actions

Fixed robot arm pushing actions were used in both of the experimental setups, with objects being placed at predefined start positions through which the push action trajectories would intersect. Theoretical models of object affordances are often approached in the literature [29, 24] as being ternary relationships between objects, actions and effects, and although we restricted the action component in this way, our focus in this paper was on learning the relations between the other two components. Even when operating under such experimental assumptions, the use of a variety of objects in various configurations in the experiments presented in this paper nevertheless resulted in reasonably diverse object effects, as illustrated in Figure 17. In a separate study [28], we relaxed this restriction and placed the emphasis on how information from varied push actions may be incorporated into the learning framework, something that is discussed later in Section 5.

3.3 Object Feature Extraction

With regard to the object features, for our particular scenario we were primarily interested in extracting features that describe the global shape of an object, as they were likely to be more relevant in determining how the object would behave when pushed than the types of local invariant visual features used predominantly in object recognition scenarios.

3.3.1 Object Detection and Segmentation

To extract such global shape features, we required a means of detecting and segmenting the objects from the scene. In the case of the Katana/Camera setup, objects were segmented both from the image data and from the 3D point clouds derived from the range data produced by the stereo camera. To this end, we used the algorithm visually outlined in Figure 6, for multi-modally segmenting the objects from both regular images and their corresponding 3D point clouds, that leverages a notable feature of our affordance learning environment, i.e., that the objects always lie on a flat table surface (cf. [27] for further details). In the case of the KUKA-LWR/Kinect setup, once again the objects always lie on a flat table surface, so we were able to make use of the *Dominant Plane Segmentation* module from the PCL library to segment the objects from the scene point cloud captured by the Kinect.

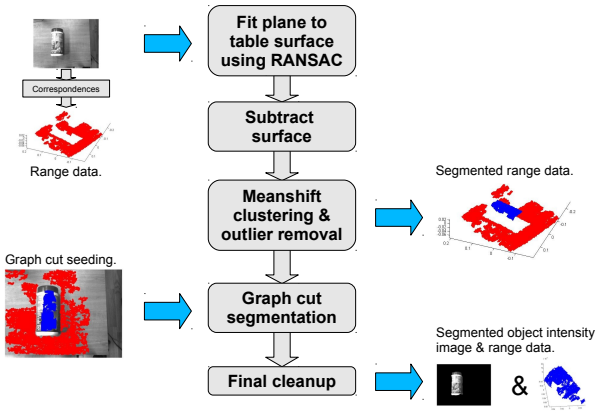

Figure 6. Katana/Camera setup object segmentation pipeline

3.3.2 Visual Shape Features from Object Images

In the case of the Katana/Camera setup, visual features were extracted from objects segmented from the greyscale scene images produced by the Bumblebee camera. The segmentation technique outlined in Figure 6 yielded reasonably robust image silhouettes of objects, and these were then used to calculate the following nine shape features:

O_1^a: Area: number of pixels in the object region.

O_2^a: Convex area: number of pixels in the convex hull of the object region.

O_3^a: Eccentricity: ratio of the distance between the foci of the ellipse that has the same second-moments as the object region and its major axis length.

O_4^a: Equivalent circular diameter: diameter of a circle with the same area as the object region.

O_5^a: Extent: ratio of pixels in the object region to the pixels in its bounding box.

O_6^a: Filled area: number of pixels in the filled object region.

2 http://pointclouds.org/

O_7^a: Major axis length: length of the major axis of the ellipse that has the same normalized second central moments as the object region.

O_8^a: Minor axis length: length of the minor axis of the ellipse that has the same normalized second central moments as the object region.

O_9^a: Perimeter: of the boundary of the object region.

3.3.3 3D Shape Features from Object Point Clouds

In order to capture object surface properties, 3D shape features were extracted from segmented object point clouds derived from both the Katana/Camera and the KUKA-LWR/Kinect experimental setups. in both cases we adopted a strategy of surface fitting. For instance, in the case of the Katana/Camera setup, a quadratic surface was fitted to an object point cloud, or to a part of an object point cloud, in order to derive curvature features from the object surface. Given the points for an object or part of an object, we fit the following quadratic polynomial function:

$$Z = aX^2 + bXY + cY^2 + dX + eY + f \tag{21}$$

solving for the coefficients a,b,c,d,e and f. The principal curvatures of the surface are then given by taking *eigenvalues* of the matrix: $\begin{pmatrix} a & b \\ b & c \end{pmatrix}$, providing two features which form a good description of the curvature of the surface. As well as this, we fitted planes to objects or object-part point clouds and used the *normal to the plane* to indicate the orientation of the points, which provided two features (we discard the z-coordinate, since the point clouds were normalized with respect to the workspace coordinate frame).

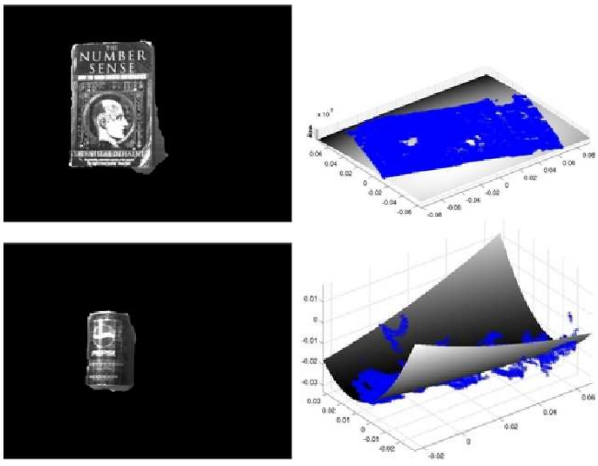

Figure 7. Examples of segmented images and 3D point clouds with fitted quadratic surfaces taken with the stereo camera for two different types of objects: a book which slides when pushed by the robotic arm, and a Pepsi can which rolls when pushed by the arm

Using these four surface features (two for planar orientation and two for curvature) in the Katana/Camera setup, we divided the object cloud into parts and extracted the four features for each part. These parts consisted of the global object cloud itself, as well as eight other parts found by dividing the object cloud evenly along two of its axes in various ways. This yielded the following list of features:

$O_{1\ldots4}^b$: Global object point cloud plane/curve features.

$O_{5\ldots8}^b$: x-axis split, left plane/curve features.

$O_{9\ldots12}^b$: x-axis split, right plane/curve features.

$O_{13\ldots16}^b$: y-axis split, front plane/curve features.

$O_{17\ldots20}^b$: y-axis split, back plane/curve features.

$O_{21\ldots24}^b$: x-y-axis split, front-left plane/curve features.

$O_{25\ldots28}^b$: x-y-axis split, front-right plane/curve features.

$O_{29\ldots32}^b$: x-y-axis split, back-left plane/curve features.

$O_{33\ldots26}^b$: x-y-axis split, back-right plane/curve features.

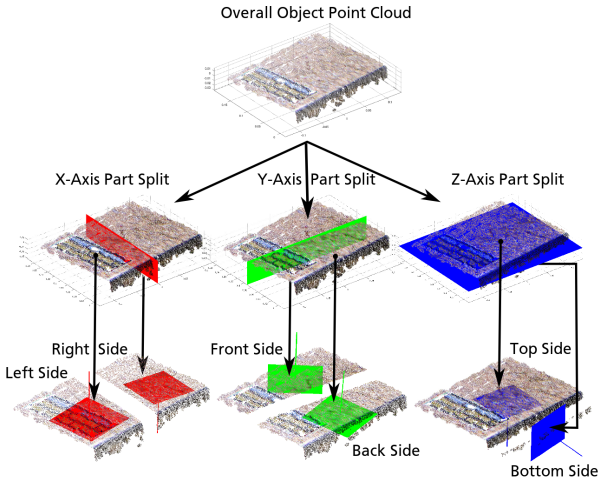

Figure 8. Top row: original object point cloud. Middle row: partitioning planes divide the point cloud evenly in each dimension to create sub-parts. Bottom row: planes are fitted to each sub-part for feature extraction.

In the case of the KUKA-LWR/Kinect setup, we partitioned the object point clouds slightly differently, as depicted in Figure 8, this time dividing each of the x, y and z axes evenly into two parts each and fitting planes to each of their respective part point clouds as well as to the entire object point cloud. The part division along the third dimension was motivated by the need to detect the additional object toppling affordance that resulted from the experiments performed with this setup. This time, object part centroids as well as plane normals and curvature features were used, resulting in seven features per part, three for part centroids,

as well as two for the plane normals and two for the curvature features as before, resulting in the following list of features:

$O^c_{1\ldots7}$: Global point cloud centroid/plane/curve features.

$O^c_{8\ldots14}$: x-split, left centroid/plane/curve features.

$O^c_{15\ldots21}$: x-split, right centroid/plane/curve features.

$O^c_{22\ldots28}$: y-split, front centroid/plane/curve features.

$O^c_{29\ldots35}$: y-split, back centroid/plane/curve features.

$O^c_{36\ldots42}$: z-split, top centroid/plane/curve features.

$O^c_{43\ldots49}$: z-split, bottom centroid/plane/curve features.

3.4 Effect Feature Extraction

When it came to the effect features, we chose to both track the objects in motion globally in the workspace and to compare changes in object properties locally, deriving three main sets of features based on the global motion of the object, changes in 3D shape, and local appearance changes of the object, respectively, depending on the platform.

3.4.1 Tracking Object Motion

In the Katana/Camera setup, after an arm action was performed on an object, the resulting videos of the interaction gathered from the Flea camera were processed for tracked object motion features. This was primarily achieved using a probabilistic tracker from [19], which is in essence a colour-based particle filter that also makes use of background subtraction using a pre-learned background image. Object shapes were approximated by elliptical regions, while their colour was encoded using colour histograms and their motion was modelled using a dynamic model from [20]. In order to compensate for tracking overshoots and thereby produce more accurate local object appearance change features (cf. Figure 9 and Section 3.4.4), the estimates of object position from this tracker were then further refined using colour histogram back-projection [34]. See [27] for further details on this method.

The KUKA-LWR/Kinect setup, on the other hand, made use of the *libpcl_tracking* library from the PCL, which also uses a particle filter to estimate 3D object poses using Monte Carlo sampling techniques and calculates the likelihood using combined weighted metrics for hyper-dimensional spaces including Cartesian data, colors, and surface normals. These 3D object models were gathered from scene point clouds using the same dominant surface segmentation method discussed in Section 3.3.1. The real-time 3D object tracking provided by this method is illustrated in Figure 10.

Figure 9. Object tracking in the Katana/Camera setup. Upper row: Outer rectangle is a likelihood window around the object from the particle filter. Inner rectangle is the result histogram back-projection further localizing the object. Lower row: Close-ups show how the appearance of the object changes during motion.

3.4.2 Object Motion Features

Using the output of the visually-based particle filter tracker of the Katana/Camera setup, the following nine features were calculated:

$E^a_{1\ldots2}$: Total distance travelled in x & y dimensions,

E^a_3: Total Euclidean distance travelled,

$E^a_{4\ldots5}$: Mean velocity in x & y dimensions,

$E^a_{6\ldots7}$: Velocity variance in x & y dimensions,

$E^a_{8\ldots9}$: Final x & y positions.

In the case of the KUKA-LWR/Kinect setup, since the 3D point cloud-based particle filter tracker from the PCL library allowed for tracking objects in three dimensions, we were able to extend some of the 2D features used on the previous setup into the z dimension. Some alternative features were also designed- in addition- to suit the experiments performed with this platform. The 17 resulting features are listed as follows:

$E^b_{1\ldots2}$: Total distance travelled in x, y & z dimensions.

E^b_4: Total Euclidean distance travelled in $\mathbb{R}^2$.

E^b_5: Total Euclidean distance travelled in $\mathbb{R}^3$.

$E^b_{6\ldots8}$: Final x, y & z positions.

E^b_9: Time in motion.

E^b_{10}: Trajectory extent volume.

E^b_{11}: Trajectory convex hull volume.

E^b_{12}: Trajectory convex hull surface area.

E^b_{13}: Summed trajectory point distance from start position.

E_{14}^{b}: Mean trajectory point distance from start position.

E_{15}^{b}: Trajectory point distance variance from start position.

E_{16}^{b}: E_{14}^{b}, weighted to favour points near end of trajectory.

E_{71}^{b}: E_{15}^{b}, weighted to favour points near end of trajectory.

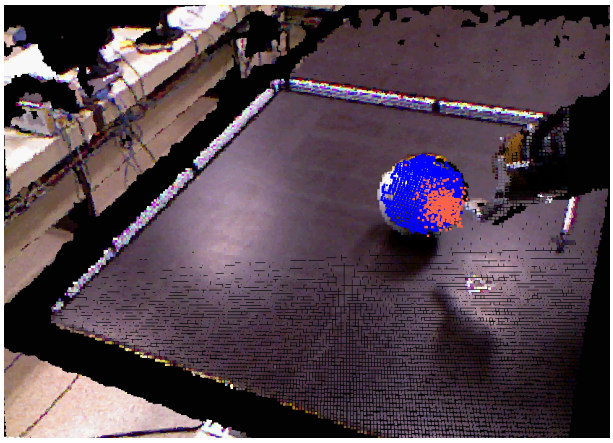

Figure 10. PCL-based particle filter tracking from the KUKA-LWR/Kinect experiment. Blue points: model of the ball object derived from segmenting the ball from the table prior to interaction. Orange points: particle filter points used to track the object during interaction and update the model.

3.4.3 Object Shape-change Features

In the case of the experiments performed using the KUKA-LWR/Kinect setup, since we intended to study toppling affordances, we required features capable of detecting the types of changes in object shape in three dimensions that would result from such affordances. Thus, using a similar methodology to that of Ugur et al. [36], we derived a set of 3D shape change features by taking the difference between the $O_{1\ldots49}^{c}$ 3D shape features from Section 3.3.3 recorded from the object pre-interaction and the same features recorded from the object post-interaction. These new features, $E_{1\ldots49}^{c}$, were therefore derived as follows:

$E_{1\ldots49}^{c}$: $O_i^c(t_e) - O_i^c(t_s)$ for $i = 1\ldots49$, where $O_i^c(t_s)$ and $O_i^c(t_e)$ are the object shape features O_i^c extracted from the object at time t_s, when it is at its start position, and at time t_e, when at its end position, respectively.

3.4.4 Object Appearance Change Features

In the experiments involving the Katana/Camera setup, in order to estimate appearance changes of objects during motion (cf. Figure 9), and thus potentially capture some of the differences in visual effects between rolling and non-rolling objects, we calculated the average difference of both colour and edge histograms between video frames of the objects, the aim being to detect both motion blur and the texture changes characteristic of many rotating objects. Histogram difference averages were then calculated from the start of object motion until the end. We derived three effect features from this procedure:

E_1^d: Average colour histogram difference.

E_2^d: Average edge histogram difference.

E_3^d: Product of E_1^d and E_2^d.

4. Experiments

We performed experiments using both of the robotic platforms described in Section 3, gathering real world data from affordance learning trials, and comparing our proposed self-supervised algorithm to well-known supervised vector quantization algorithms. The learning trials were divided into a number of sub-experiments, using different combinations of the feature sets described in Sections 3.3 and 3.4 depending on the particular platform being used and the goals of the experiment. The Katana/Camera experiment involved objects and pushes that resulted in two different affordance ground-truth effect classes being produced- rolling and non-rolling classes-whereas in the KUKA-LWR/Kinect experiment, more complex effects were produced from the objects used, their poses and the scenario, resulting in three different ground-truth classes: rolling, non-rolling and toppling. The supervised algorithms that were compared were generalized learning vector quantization (GLVQ) [30], generalized relevance learning vector quantization (GRLVQ) [15] and supervised relevance neural gas (SRNG) [14], the latter two of which also provide feature relevance estimates. We compared their performance relative to our self-supervised learning approach by applying the evaluation procedure described below.

4.1 Evaluation Procedure

In the following, a modified form of leave-one-out cross-validation, which we call leave-one-object-out cross-validation (LOOOCV) was employed. In order to evaluate our learning approach, given an $\{X, Y\}$ dataset of features extracted from interactions with various objects from a given experiment, LOOOCV involved splitting the dataset into a test set consisting of all of the samples for a given object, and a training set of all of the samples for the remaining objects. The learning task was then to train learners using the training set, find the affordance classes in the output view and try to classify the test set samples of the left-out object on that basis. Cross-validation was performed by using each of the objects in the full dataset in turn as the test object to form multiple test and training sets, performing the learning task using each of these, and averaging class discovery and class prediction scores across all of them. Performing the evaluation in this way, using LOOOCV, allowed us to test the performance of affordance prediction for novel objects that the algorithms do not encounter during training, a more stringent evaluation

than would otherwise be provided by LOOCV or k-folds cross validation.

Our self-supervised learning vector quantization algorithm (SSLVQ) was compared to a self-supervised self-organizing map (SSSOM), as well as variations employing feature relevance determination at classification time (SSLVQ (FRC) and SSSOM (FRC)) alongside the supervised algorithms GLVQ, GRLVQ and SRNG. SSSOM differed from SSLVQ in that it only made use of the update in (4) in both input and output views, whereas SSLVQ also employed the self-supervised update of (12) in the input view. In the case of the experiments with the Katana/Camera setup presented below in Section 4.2, codebooks in each data view consisted of 49 prototypes arranged in a 7×7 hexagonal lattice with a sheet-shaped topology [17], whereas in the experiments using the KUKA-LWR/Kinect setup presented in Section 4.3, larger 10×10 codebooks were used. The feature weights of the codebook prototype vectors were randomly initialized to test the abilities of the algorithms to learn from scratch. LOOOCV was therefore performed in 10 trials and results were averaged in order to account for the variation in codebook initialization between trials. Constant learning rates of α_W^t=0.1, α_V^t=0.1, were used for the prototype update of (4) in codebooks W and V, respectively, in the unsupervised learning phase, and α_L^t=0.1 was used for (12) in the self-supervised learning phase. Learning phases were switched from unsupervised to self-supervised halfway through training in the Katana/Camera experiments, and one tenth of the way through training in the KUKA-LWR/Kinect experiments where a longer training period was employed over multiple epochs.

The results examine three different aspects of our learning framework: class discovery, class prediction and feature relevance determination. An important consideration in evaluating whether or not our algorithm is capable of self-supervised multi-view learning is to examine whether it is capable of successfully finding class clusters in the output-view, without which self-supervised discriminative learning in the input-view would not be possible. But how quickly do the prototypes position themselves, such that this clustering can happen successfully? To answer this question, clusters of prototypes were found in the output view as described in Section 2.3.1 and subsequently matched to the ground truth classes by first matching all ground truth labelled training data to nearest-neighbour output view prototypes, then assigning each cluster the ground truth label which their respective prototypes matched to most frequently. Then, to evaluate prediction, given a test sample consisting of an input view test vector $\mathbf{x}^i$ and an output view test vector $\mathbf{y}^i$, the input view codebook was tasked with predicting an output view cluster V^j for $\mathbf{x}^i$ using the process described in Section 2.3. The output view test vector $\mathbf{y}^i$ was then matched to a cluster V^k in the output view via the nearest-neighbour rule. If the V^j cluster predicted by the input codebook matched the V^k cluster and that cluster also matched the ground-truth label for the test sample, this was deemed to be a true positive. We evaluated the feature relevance components by averaging their λ_i weights after training.

4.2 Experiments using Katana/Camera Platform

To test our affordance learning system with the Katana/Camera platform, the experimental environment was set up as shown in Figure 1. During the experiments, objects were placed at a fixed starting position prior to interaction. The two camera systems were used to provide both sufficiently detailed close-up range data of the object surfaces and a sufficiently wide field of view to capture object motion over the entire work area. To achieve this, the stereo camera was positioned above the start position, while the monocular camera was positioned at a higher position in front of the workspace, giving both cameras a top-down viewpoint of the work surface.

Figure 11. Object image segmentations (not to scale) from the Katana/Cameras experiment

We selected eight household objects (cf. Figure 11) for the experiments: four flat-surfaced objects (a book, a CD box, a box of tea and a drink carton) and four curved-surfaced objects (a box of cleaning wipes, a cola can, a soda can and a tennis ball box. A dataset was gathered consisting of 20 object push tests for each of the eight objects and the resulting data was processed, leaving 160 data samples. During tests, the curved objects would tend to roll after being pushed, whereas the flat objects would stop suddenly; accordingly, the samples were then hand-labelled with two ground-truth labels: rolling and non-rolling. These ground-truth labels were not used to train the self-supervised learners but they were required for the performance evaluation. This data was processed to produce the visual shape features $\{O_1^a,\ldots,O_9^a\}$ and 3D shape features $\{O_1^b,\ldots,O_{36}^b\}$ (cf. Sections 3.3.2 & 3.3.3), as well as the global motion effect features $\{E_1^a,\ldots,E_9^a\}$ and the local appearance change features $\{E_1^d,\ldots,E_3^d\}$ (cf. Sections 3.4.2 & 3.4.4). In the first of the two sub-experiments (the results of which are described below), the features $\{O_1^a,\ldots,O_9^a\}$ were paired with the two 3D global object curvature features $\{O_3^b,O_4^b\}$ to produce input view space $X^{a,b}$ consisting of vectors of the form $\mathbf{x}=\{O_1^a,\ldots,O_9^a,O_3^b,O_4^b\}^T$, while in the second, vectors of

the form $\mathbf{x}=\{O_1^b,\ldots,O_{36}^b\}^T$ using all of the 3D shape features formed the basis for input-view space X^b. In both sub-experiments, both the global motion effect features and the local appearance change effect features were grouped, defining the output view space $Y^{a,d}$ composed of vectors of the form $\mathbf{y}=\{E_1^a,\ldots,E_9^a,E_1^d,\ldots,E_3^d\}^T$.

4.2.1 Results: 3D + 2D Object Features

In this experiment, the self-supervised learners were trained using data from $\{X^{a,b},Y^{a,d}\}$, while the supervised learners were trained with $\{X^{a,b},L_{GT}(X^{a,b})\}$, where L_{GT} applies ground-truth labels to the input view samples. Online evaluation and comparison of the learners was performed using LOOOCV over one epoch of training.

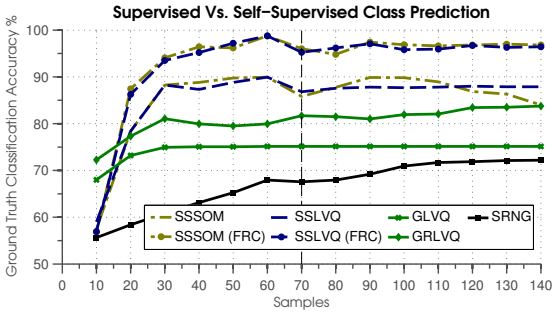

Figure 12. LOOOCV (cf. Section 4.1) class prediction results for the Katana/Camera 3D + 2D object feature experiment (cf. Section 4.2.1). The vertical dashed line indicates the transition from unsupervised to self-supervised learning (cf. Section 2).

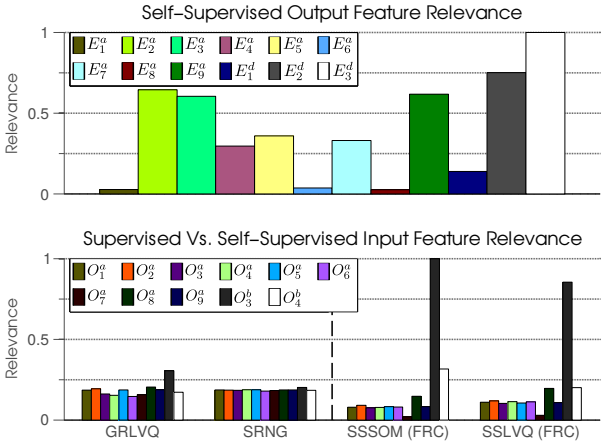

Figure 13. Feature relevance results for the Katana/Camera 3D + 2D object features experiment (cf. Section 4.2.1). The vertical dashed line in the lower figure separates the input-view feature histograms for two supervised and two self-supervised learners. The upper figure shows output view results which were applicable to self-supervised learners only and were the same in both cases (SSSOM & SSLVQ). Features are colour-coded (see legends).

Class Discovery: Optimal performance was achieved very early in training, after 10 samples, with class-cluster-to-ground-truth match accuracy being maintained at 100% throughout, meaning that cross-view prediction from input view test samples should at least have the opportunity to reach optimal ground truth prediction rates within the training period.

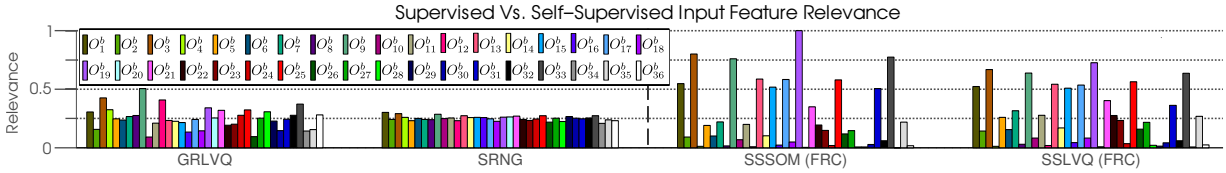

Figure 14. Input view feature relevance results from the Katana/Camera experiment using the 3-D object features (cf. Section 4.2.2)

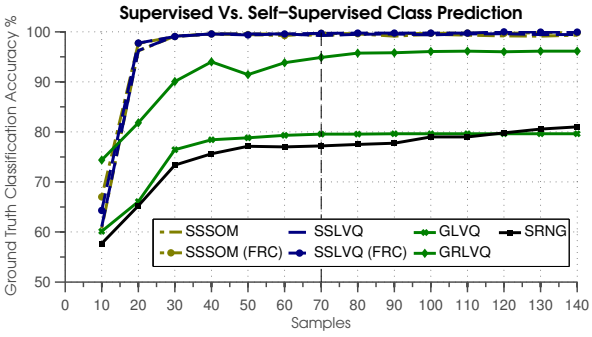

Figure 15. LOOOCV class prediction results for the Katana/Camera 3-D object feature experiment (cf. Section 4.2.2)

Class Prediction: When it comes to predicting these newly discovered classes, Figure 12 shows how the various self-supervised learners perform and how they compare to supervised classifiers predicting ground-truth labels using input view features. In this test, only a small subset of the object features (curvature features $\{O_3^b,O_4^b\}$) is relevant for class prediction, and the most significant result is that there is a prominent difference between those self-supervised learners with feature relevance mechanisms and those without, most likely due to this reason. Interestingly, all of the self-supervised classifiers out-perform the supervised classifiers throughout training. This initially surprising result is likely due to the fact that, owing to the randomized prototype initialization, the prototype class labels of the supervised classifiers are not optimally distributed with

respect to the class distributions at the outset, whereas the self-supervised classifiers undergo dynamic labelling of their input view prototypes as described in Section 2.3.2. This means that the supervised classifier prototypes potentially take more time to adjust their positions with respect to the class distributions. It is also worth noting that SSLVQ appears to maintain more predictive stability over this short training period than SSSOM, which lacks the self-supervised discriminative learning mechanism of SSLVQ (cf. Section 2.2.3).

Feature Relevance Determination: Figure 13 illustrates the mean results of feature relevance determination at the end of training for both the input and output views. The input view graph features comparisons with the supervised learners GRLVQ and SRNG that feature feature relevance determination mechanisms. The features E_2^a,E_3^a,E_9^a,E_2^d and E_3^d, those being total distance traveled in y, total Euclidean distance traveled, final y position, the average edge histogram difference and the product of it and its colour counterpart, are highlighted as being the most relevant in the output view for class discovery, which makes sense given their assumed potential for distinguishing rolling versus non-rolling behaviours in the context of this experiment. In the input view, most of the learners correctly select the O_3^b feature, or the object curvature along the x-dimension (along the rolling direction), as being the most relevant for class prediction. Both GRLVQ and SRNG make the least prominent distinction because- in the short training time tested- their gradient descent-based feature relevance mechanisms most likely do not have sufficient training time in which to function optimally.

4.2.2 Results: 3D Object Features

Class Discovery: In this experiment, both the output view data and the learning parameters were the same as in the previous experiment; thus class discovery results were indistinguishable, with optimal performance again being attained early in training and maintained throughout.

Class Prediction: In this experiment, the 2D visual object features were removed in favour of purely 3D surface features of feature space X^b, and since many more of these features were relevant for class prediction than in the previous case, there were correlated improvements in the class prediction capabilities of all learners, as illustrated in Figure 15. This time, all of the self-supervised learners performed comparably well, reaching 100% performance early in training, as well as beating the supervised classifiers over one epoch of training.

Feature Relevance Determination: Since both the features that were used in the output view- as well as the output view learning parameters- were identical to those used in the previous experiment, the results for output feature relevance determination were almost identical, and so we omit those results here. The input-view feature relevances, on the other hand, as shown in Figure 14, reveal that many of the features of X^b are relevant for class prediction. Features O_3^b,O_9^b,O_{19}^b, and O_{33}^b are favoured prominently by both the supervised and self-supervised learners. The selection of O_3^b matches with the results from the previous experiment, where it was one of two curvature features included in input space $X^{a,b}$, and was selected as the most relevant feature. The right side of the x-axis split that generated the O_9^b plane normal feature was the closest side to the stereo camera, and so produced more 3D points (thus yielding more reliable surface fits and resulting features). Similarly, O_{19}^b is generated from the back side of the y-axis split, which is the closest object part to the camera along that dimension. Finally, feature O_{33}^b is generated from the part at the intersection between the previous two, and so benefits from the increased reliability of both.

Figure 16. Object clouds from the KUKA-LWR/Kinect experiment

4.3 Experiment using the KUKA-LWR/Kinect Platform

In the KUKA-LWR/Kinect platform experiment, we set out to apply our learning framework to a slightly more complex pushing scenario that would yield more than two affordance classes: rolling, toppling and translating. The environment was set up in similar fashion to the experiments with the Katana/Camera system (cf. Figure 2). This time, we selected 10 household objects (cf. Figure 16) for the experiments: six flat-surfaced objects and four curved-surfaced objects. During the experiment, objects were again placed at a fixed starting position prior to interaction. This time, however, objects were placed in more varied poses when possible in order to generate more varied affordance effects: flat, sideways or upright, and either with the major axis of the object perpendicular to or parallel to the push direction vector if that was well-defined, i.e., for the non-ball objects. These descriptions are provided here for the readers' benefit and were not used for training (except for the pose information potentially being encoded in the input feature vectors). During these trials, the flat-surfaced objects would either tend to translate forward or topple over after being pushed, depending on their pose, and the balls would tend to roll, though along much more complex and varied trajectories than in the Katana/Camera experiments, as illustrated in Figure 17.

The samples were again hand-labelled with the ground-truth labels: rolling, translating and toppling. 126 samples

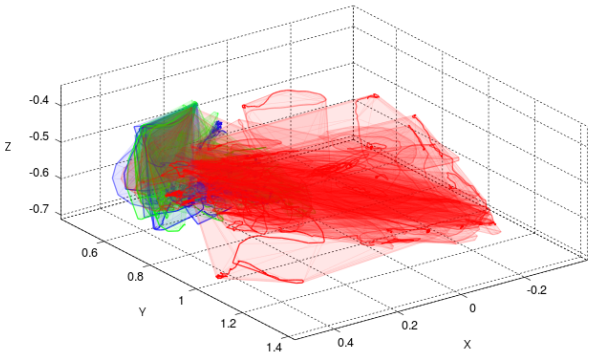

Figure 17. Object trajectories and trajectory convex hulls from the KUKA-LWR/Kinect experiment, colour-coded by ground-truth; red: rolling; green: toppling; blue: translating

were collected, in total, of a biscuit box (12 samples: eight toppling, four translating), a coffee box (20 samples: five toppling, 15 translating), a cookie packet (eight samples: all translating), a contact lens solution box (16 samples: all toppling), a marshmallow box (23 samples: nine toppling, 14 translating), a book (14 samples: six toppling, eight translating), a handball (eight samples), a small football (six samples), and both lightly coloured (eight samples) and darkly coloured (11 samples) larger plastic toy balls. This time, the data was used to generate the 3D shape features described in Section 3.3.3 for the input view space X^c consisting of feature vectors of the form $\mathbf{x}=\{O_1^c,\ldots,O_{49}^c\}^T$, as well as the 3-D shape change features and the global motion effect features described in Sections 3.4.2 and 3.4.3, respectively, to form the output-view space $Y^{c,b}$ consisting of feature vectors of the form $\mathbf{y}=\{E_1^c,\ldots,E_{49}^c,E_1^b,\ldots,E_{17}^b\}^T$.

4.3.1 Staged Feature Relevance Determination

Initial learning trials with the full $\{X^c,Y^{a,d}\}$ feature set proved inconsistent, so we opted for a more advanced learning strategy: staged self-supervised learning, in which the most relevant features are determined via thresholding feature relevance at each stage, and where the there remaining features were discarded before retraining with the reduced feature set at the next stage. This emulates an aspect of developmental learning where concepts are refined over multiple learning stages. We thus performed three rounds of self-supervised learning using LOOOCV on the SSLVQ (FRC) learner over 10 epochs for the first two stages, before performing a final run of LOOOCV on all of the learners over 50 epochs using the remaining features. After the first stage, only the features from the output view were reduced, whereas after the second stage, both the input and output view feature sets were reduced. When reducing feature sets, thresholds were set to the mean feature relevance value plus one standard deviation. Output feature relevances were calculated separately for the shape difference features and for the motion features.

4.3.2 Results

Feature Relevance Determination: The input vectors used in the first stage were of the form $\mathbf{x}^{s_1}=\{O_1^c,\ldots,O_{49}^c\}^T$, with output vectors $\mathbf{y}^{s_1}=\{E_1^c,\ldots,E_{49}^c,E_1^b,\ldots,E_{17}^b\}^T$. After the first stage, the input vectors remained as $\mathbf{x}^{s_2}=\mathbf{x}^{s_1}$ for the second stage, while the output vectors were reduced to the form $\mathbf{y}^{s_2}=\{E_9^c,E_{11}^c,E_{23}^c,E_{24}^c,E_{38}^c,E_{39}^c,E_{45}^c,E_{14}^b,E_{16}^b\}^T$ (cf. upper part of Figure 18). After the second stage, the input vectors were reduced to $\mathbf{x}^{s_3}=\{O_4^c,O_{11}^c,O_{18}^c,O_{23}^c,O_{24}^c,O_{25}^c,O_{28}^c,O_{31}^c,O_{32}^c,O_{39}^c,O_{41}^c,O_{45}^c\}^T$ (cf. lower part of Figure 18), while the output vectors were reduced to $\mathbf{y}^{s_3}=\{E_{45}^c,E_{14}^b\}^T$. By the end of the final third stage, in the output view, only the E_{45}^c (*z-axis split top side part centroid z-coordinate difference*) and E_{14}^b (*mean trajectory point distance from the start position*) features remain the most relevant for class discovery, whereas the input-view features O_{28}^c (*y-axis split front-side part y-curvature*) and O_{45}^c (*z-axis split top-side part centroid z-coordinate*) are the most relevant for class prediction. This demonstrates that affordance concepts which associate curvature in the direction of the push (O_{28}^c) and object height (O_{45}^c) with affordance effect classes defined primarily by object height difference (E_{45}^c) and the distance travelled by the object in motion (E_{14}^b), having been formed autonomously using staged self-supervised learning. This appears to match with human intuition for this scenario, although this has not been tested empirically. It is also worth noting that the SSLVQ-based method proposed in this paper proves to be more effective than SSSOM at discerning relevant discriminative features in the input view. This is not too surprising given that the SSSOM lacks a cross-view discriminative learning mechanism (cf. Section 4.1).

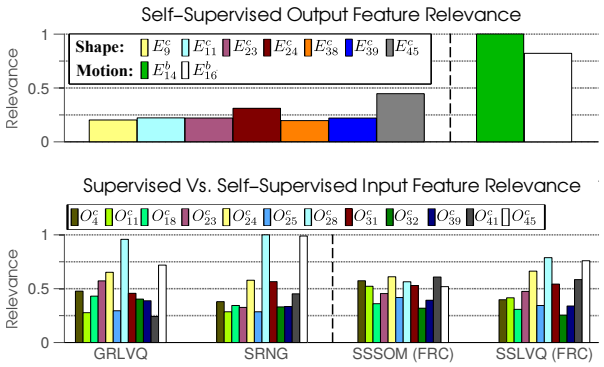

Figure 18. Mean feature relevance results for both input and output views in staged feature relevance learning for the KUKA-LWR/Kinect experiment as described in Section 4.3.1. The upper figure shows the relevance histogram of the reduced output feature set after the second learning stage with a vertical line separating the shape and motion features, which were clustered and reduced separately at each stage. The lower figure shows reduced feature relevance histograms for the input view after the third stage where the vertical dashed line separates supervised and self-supervised learners.

*Class Discovery:*Figure 19 shows the average class discovery results for each of the self-supervised learners over 10 trials of the 50-epoch third stage LOOOCV evaluation. The learning parameters were the same in the output-view in all cases, so the results are similar for each of the learners. The learners discover the ground-truth classes with a mean accuracy of 90% after 50 epochs of training, which sets a limit on their potential results for class prediction.

*Class Prediction:*Figure 20 shows both average class prediction results for all learners and the average SSLVQ (FRC) class prediction for specific test objects. While GRLVQ and SRNG achieve just over 85% accuracy on average during most of the 50-epoch training period, the self-supervised leaners start out quite poorly, before reaching 75% accuracy by the end of training. SSLVQ (FRC) appears to give slightly better performance over the other self-supervised learners, but the results are inconclusive. This might be explained by the fact that the feature sets have been significantly refined by this final learning stage. The lower graph of Figure 20 shows that SSLVQ (FRC) appears to struggle most when either the dark- or the light-coloured large balls are left out of the training set. This may be due to the fact that when either of these objects are left out, there are far fewer training samples that are both curved in the direction of the push (O^c_{28} feature) *and* are relatively tall (O^c_{45} feature). For example, when the dark ball is left out, there are 115 total training samples, 22 of which are rolling samples, and only eight of which are samples of relatively tall ball-shaped rolling objects. By comparison, when the small football is left out, there are 120 total training samples, 27 of which are rolling samples and 19 of which are samples of relatively tall rolling balls. Without adequate numbers of samples of tall rolling balls in the training set, the self-supervised learner struggles when it encounters them as novel samples in the test set.

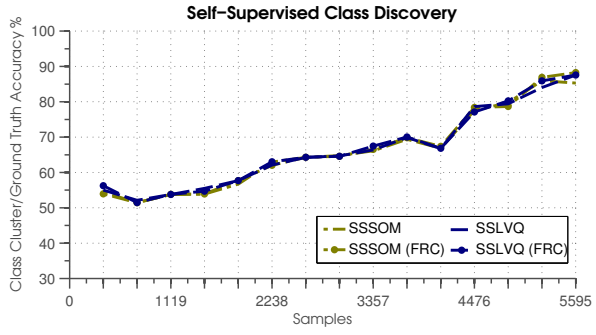

Figure 19. Mean class discovery results for the third stage of LOOOCV (cf. Section 4.1) run over 50 training epochs in the KUKA-LWR/Kinect experiment as described in Section 4.3

5. Conclusion

In this paper, we presented a self-supervised multi-view online learning algorithm along with two robotic systems used for performing object push-affordance learning experiments and demonstrated how the algorithm could be used to enable the autonomous acquisition of novel affordance concepts. We formalized a multi-view learning scenario where data co-occur over two separate data views over time, and where the task is to exploit significant clusters that emerge in one view as classes that can be used to perform online discriminative learning in the other view. Our proposed algorithm uses a self-supervised form of learning vector quantization to build the discriminative model by mapping information between the data views using the co-occurrence information gained from online learning experience. We tested our approach using data from real-world experiments by comparing it with related supervised methods and showed how the inclusion of a feature relevance determination mechanism can boost predictive accuracy when many redundant features are present in the data. In particular, we demonstrated how a such self-supervised learning process can be applied in developmental stages where the feature set is refined at each stage in order to enhance the learning process.

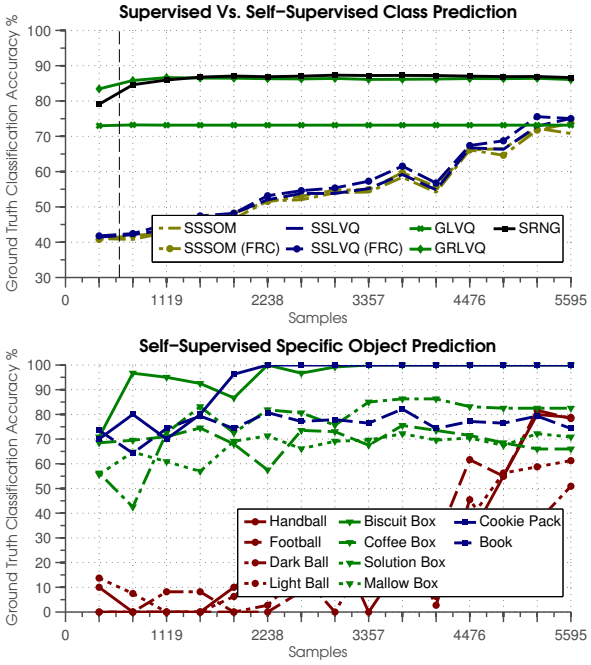

Figure 20. Mean class prediction results for all learners (upper graph) & for SSLVQ (FRC) with specific test objects (lower graph) for the third stage of LOOOCV (cf. Section 4.1) run over training 50 epochs in the KUKA-LWR/Kinect experiment (cf. Section 4.3)

In future work, we aim to expand on the number of classes and objects used, perhaps by training multiple learners under a mixture of experts model. We would also extend the model so that more data views are included, which could be useful when other sensory modalities, e.g., haptic, are used alongside the visual ones. Actions are a crucial component of affordance learning, and although a more thorough investigation of their potential in our learning setup went beyond the scope of this work, there are many possible ways in which they could be exploited, one avenue

of which we have explored elsewhere [28]. Otherwise, learning by imitation could help select the initial feature set and guide exploratory behaviours in a goal-directed manner. Reinforcement learning could also help the robot guide its own exploration once the goals are defined, thus improving the learning-rate. In this sense, our approach is complementary to such approaches, and we hope to explore these connections in future.

6. Acknowledgements

This research was supported by both the EU FP7 project CogX (ICT-215181) and the EU FP7 project Xperience (ICT-270273).

7. . References

[1] S. Bickel and T. Scheffer. Multi-view clustering. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 19–26, Washington D.C., USA, November 2004.

[2] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, page 92–100, 1998.

[3] N. Bolshakova and F. Azuaje. Cluster validation techniques for genome expression data. *Signal Processing*, 83(4):825–833, April 2003.

[4] Michael H. Coen. Cross-modal clustering. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, volume 2, page 932–937, Pittsburgh, PA, USA, July 2005. AAAI Press.

[5] Michael H. Coen. Self-supervised acquisition of vowels in american english. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)*, volume 2, page 1451–1456, Boston, MA, USA, July 2006. AAAI Press.

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

[7] Renaud Detry, Dirk Kraft, Oliver Kroemer, Leon Bodenhagen, Jan Peters, Norbert Krüger, and Justus Piater. Learning grasp affordance densities. *Paladyn*, 2(1):1–17, 2011.

[8] Virginia de Sa, Patrick Gallagher, Joshua Lewis, and Vicente Malave. Multi-view kernel construction. *Machine Learning*, 79(1):47–71, 2010.

[9] V. R. de Sa. Learning classification with unlabeled data. In *Advances in Neural Information Processing Systems 6*, pages 112–119, Denver, CO, USA, 1994. Morgan Kaufmann.

[10] Sandrine Dudoit and Jane Fridlyand. A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome Biology*, 3(7):research0036, June 2002.

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

[12] Alison L Gibbs and Francis Edward Su. On choosing and bounding probability metrics. *International Statistical Review*, 70(3):419–435, December 2002.

[13] J. J Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.

[14] B. Hammer, M. Strickert, and T. Villmann. Supervised neural gas with general similarity measure. *Neural Processing Letters*, 21(1):21–44, 2005.

[15] B. Hammer and T. Villmann. Generalized relevance learning vector quantization. *Neural Networks*, 15(8-9):1059–1068, 2002.

[16] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. New York: Wiley, 1949.

[17] T. Kohonen. *Self-organizing maps*. Springer, 1997.

[18] Marek Kopicki. *Prediction learning in robotic manipulation*. Ph.D. thesis, University of Birmingham, April 2010.

[19] M. Kristan, J. Perš, A. Leonardis, and S. Kovačič. A hierarchical dynamic model for tracking in sports. In *Proceedings of the Sixteenth Electrotechnical and Computer Science Conference*, September 2007.

[20] M. Kristan, J. Perš, M. Perše, and S. Kovačič. Closed-world tracking of multiple interacting targets for indoor-sports applications. *Computer Vision and Image Understanding*, 113(5):598–611, 2009.

[21] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, 1980.

[22] G. Metta and P. Fitzpatrick. Early integration of vision and manipulation. *Adaptive Behavior*, 11(2): 109–128, 2003.

[23] R. Miikkulainen. Dyslexic and category-specific aphasic impairments in a self-organizing feature map model of the lexicon. *Brain and Language*, 59(2): 334–366, 1997.

[24] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor. Learning object affordances: From sensory-motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15–26, 2008.

[25] Barry Ridge, Aleš Leonardis, and Danijel Skočaj. Relevance determination for learning vector quantization using the fisher criterion score. In *Proceedings of the Seventeenth Computer Vision Winter Workshop (CVWW)*, Mala Nedelja, Slovenia, February 2012.

[26] Barry Ridge, Danijel Skočaj, and Aleš Leonardis. A system for learning basic object affordances using a self-organizing map. In *Proceedings of the First*

International Conference on Cognitive Systems, pages 65–70, Karlsruhe, Germany, 2008.

[27] Barry Ridge, Danijel Skočaj, and Aleš Leonardis. Self-supervised cross-modal online learning of basic object affordances for developmental robotic systems. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5047–5054, Anchorage, USA, May 2010. IEEE.

[28] Barry Ridge and Aleš Ude. Action-grounded push affordance bootstrapping of unknown objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, Tokyo, Japan, November 2013.

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

[30] A. Sato and K. Yamada. Generalized learning vector quantization. In *Advances in Neural Information Processing Systems 8*, page 423–429, Denver, CO, USA, 1996. MIT Press.

[31] A. Saxena, J. Driemeyer, and A. Y Ng. Robotic grasping of novel objects using vision. *International Journal of Robotics Research*, 27(2):157, 2008.

[32] Jivko Sinapov, Taylor Bergquist, Connor Schenck, Ugonna Ohiri, Shane Griffith, and Alexander Stoytchev. Interactive object recognition using proprioceptive and auditory feedback. *The International Journal of Robotics Research*, 30(10):1250–1262, September 2011.

[33] Shiliang Sun. A survey of multi-view machine learning. *Neural Computing and Applications*, page 1–8, 2013.

[34] M. J Swain and D. H Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[35] E. Ugur, E. Oztop, and E. Sahin. Goal emulation and planning in perceptual space using learned affordances. *Robotics and Autonomous Systems*, 2011.

[36] E. Ugur, E. Sahin, and E. Oztop. Self-discovery of motor primitives and learning grasp affordances. In *Proceedings of the 2012 IEEE/RSJ International Conference on Inte*lligent Robots and Systems (IROS), pages 3260 –3267, October 2012.