2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Action-Grounded Push Affordance Bootstrapping of Unknown Objects*

Barry Ridge and Aleš Ude†

Abstract— **When it comes to learning how to manipulate objects from experience with minimal prior knowledge, robots encounter significant challenges. When the objects are unknown to the robot, the lack of prior object models demands a robust feature descriptor such that the robot can reliably compare objects and the effects of their manipulation. In this paper, using an experimental platform that gathers 3-D data from the Kinect RGB-D sensor, as well as push action trajectories from a tracking system, we address these issues using an action-grounded 3-D feature descriptor. Rather than using pose-invariant visual features, as is often the case with object recognition, we ground the features of objects with respect to their manipulation, that is, by using shape features that describe the surface of an object relative to the push contact point and direction. Using this setup, object push affordance learning trials are performed by a human and both pre-push and post-push object features are gathered, as well as push action trajectories. A self-supervised multi-view online learning algorithm is employed to bootstrap both the discovery of affordance classes in the post-push view, as well as a discriminative model for predicting them in the pre-push view. Experimental results demonstrate the effectiveness of self-supervised class discovery, class prediction and feature relevance determination on a collection of unknown objects.**

I. INTRODUCTION

Endowing an autonomous robot with both the ability to learn about object affordances [1] from experience and the ability to use these learned affordances to make useful predictions and manipulations in its environment is no easy task, and simplifying assumptions are often made in order to make the problem more soluble. For example, in the case of object push affordance learning [2]–[6], if the desired result is to learn how the positions and orientations of objects change when pushed, the learning task can be simplified by selecting prior object models, using standard computer vision techniques to localise the object models within a scene, and inferring data such as end effector contact points on the objects using the models. However, when fewer assumptions are made about the shapes of objects or their affordances, such techniques may not be as feasible, and while this increases the complexity of the learning problem, it is an important research approach from a cognitive and developmental robotics perspective.

In this paper, we explore object push affordance learning by gathering data from human object push experimental trials (see Fig. 1). Objects on a table surface were pushed by a human, whose hand motion trajectories were tracked while 3-D

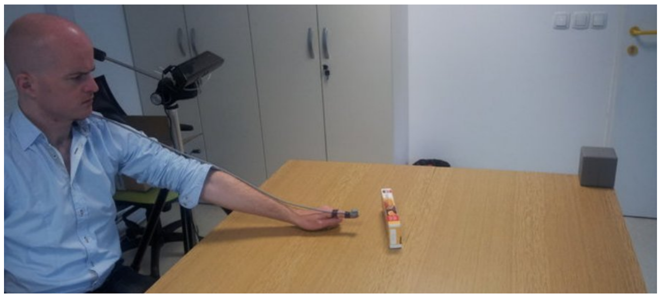
Fig. 1. Our setup for human object push affordance data gathering.

point clouds of the objects before and after interaction were recorded. The objects were pushed from various different positions on their surfaces and from various different directions exhibiting a number of different affordances such as forward translations, forward topples, left rotations and right rotations, depending broadly on the shapes of the objects, their orientations, and how they were pushed. Our goal was to extend a type of bootstrap learning whereby significant clusters are discovered in features extracted from the post-push point clouds that are used as affordance classes in order to train an affordance classifier using features extracted from the pre-push point clouds as input in a developmental multi-view online learning process. Note that a similar learning process could be realised on an autonomous robot. We use the above setup to ease the process of data gathering.

An idea for grounding the affordance learning task in the pushing actions informed our research. We argue that the approach of visual object recognition followed by object manipulation informed by a prior object model (see e.g. [6]). is, although quite useful when the main focus is accurate prediction, perhaps less important when the main focus is learning from experience. Instead, here we favour an approach where little or no prior information on the structure of the objects being pushed is assumed. To this end, we propose a features-based approach where, rather than using pose-invariant visual features, as is commonly the case with object recognition, we ground the visual features of objects with respect to their manipulation, that is, by using shape features that describe the surface of an object relative to the push contact point and direction.

A. Related Work

Past work on object affordance learning with robots has seen a varied succession of approaches, ranging from learning affordances for particular objects [2], to supervised discriminative learning of pre-defined affordance classes from object features [7], to unsupervised discovery and subsequent

*This research has been supported by EU FP7 projects Xperience (FP7-ICT-270273) and IntellAct (FP7-ICT-269959).

†B. Ridge and A. Ude are with the Laboratory of Humanoid and Cognitive Robotics, Department of Automation, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia. barry.ridge at ijs.si ales.ude at ijs.si

discriminative learning of affordance classes [5], [8]–[11] to probabilistic frameworks [3], [6], and others. One of the earliest works in the literature to deal with affordance learning in a robot was by Fitzpatrick *et al.* [2]. 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 directions as well as simple visual descriptors for object recognition.

Paletta *et al.* [7], [12] developed a mobile robotic platform equipped with a crane featuring a magnetic end-effector which was used to pick up metallic objects in its surroundings. Their affordance learning system used decision trees and reinforcement learning of predictive features (SIFT descriptors) to distinguish between two affordance classes of liftable and non-liftable metallic objects. Ugur *et al.* [11] 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. More recently they have applied similar techniques to a scenario involving self-discovery of motor primitives and learning grasp affordances [13].

In [3], the authors used a humanoid robot to push objects on a table and used a Bayesian network to form associations between actions, objects and effects. In the work of Omrčen *et al.* [4], the robot first observes how an object moves when pushed in a certain direction. The collected data are used as input to a neural network which learns to predict the motion of pushed objects. Kopicki *et al.* [6] used a probabilistic framework to address prediction of rigid body transformations in an object pushing scenario both in simulation and using a real robotic system. Their work was similar to ours here in that it explicitly addressed the representation of object parts as well as the combination of knowledge from multiple models. However, their visual system relied on the use of prior object models for object localisation, something we explicitly avoid in this work.

Object shape features have been used in prior work on push affordance learning [5], [6], [11], [13], but grounding object shape features relative to pushing actions has not been studied as extensively. Recent work by Hermans *et al.* [14] used shape features encoded in a coordinate frame defined by object centres and push locations based on 2-D projections of object point clouds. In this paper, we develop a similar idea employing full 3-D shape features.

The remainder of the paper is structured as follows. In the following section, we describe how the action-grounded features are derived, including the object segmentation process, the transformation of both objects and action trajectories to the action coordinate frame, and the description of the features themselves. In Section III, we describe our learning approach. In Section IV we describe our experiments and results. Finally, in Section V, we conclude.

II. GROUNDING 3-D SHAPE FEATURES IN PUSH ACTIONS

In our experimental setup for human object push data gathering, shown in Figure 1, we employed a Microsoft Kinect™ RGB-D sensor for gathering 3-D point cloud data of scenes and objects, and a Polhemus Patriot™ electromagnetic tracking system for gathering trajectory data of human hand motions. A wooden table with a wooden frame was used as the work surface in order to avoid electromagnetic interference from metallic objects in the environment. A tracking sensor was placed at the end of the index finger of a human experimenter, while the tracking source was located at a corner of the table with the Kinect facing the table at a 45° angle as shown in Figure 1. Objects were placed at arbitrary locations on the table surface where they were pushed from various directions and at various contact points by the experimenter. 3-D point clouds of the scene were recorded both before and after each push interaction while hand trajectories were tracked during the interaction. Both the point clouds and the trajectories were processed offline where the objects were segmented from the table surface, object point clouds and push trajectories were transformed into the push action coordinate frame, and action-grounded shape features were extracted.

A. Object segmentation

We used tools from the Point Cloud Library (PCL)[1] to perform dominant plane segmentation on scene point clouds in order to acquire segmented point clouds of the objects lying on the table surface. This involved using a pass-through filter to subtract points in the scene cloud outside certain range limits, using RANSAC [15] to fit a plane model to the scene cloud, subtracting those scene points that were plane inliers, and clustering the remaining points to find the objects using Euclidean clustering [16].

B. Action coordinate frame transformation

We define the action frame to be the coordinate system with its origin at the contact point on the object, its positive y-axis pointing in the direction of the pushing motion parallel to the table surface, its positive z-axis pointing upward from the table surface, and its positive x-axis pointing perpendicularly to both of them such that a left-handed coordinate system is formed. In order to transform both the object point cloud and the push trajectory into the action frame, we perform the following procedure. Firstly, we transform the push trajectory from the Patriot tracker coordinate system to the Kinect coordinate system by using least-squares adjustment on a series of control points and calculating a rigid body transformation of the form $\mathbf{x}' = \mathbf{c} + \mathbf{R}\mathbf{x}$, where $\mathbf{x}'$ is the transformed vector, $\mathbf{x}$ is the initial vector, $\mathbf{c}$ is the translation vector, and $\mathbf{R}$ is a rotation matrix. The control points are gathered prior to performing pushing experiments by placing the tracking sensor at various positions in the workspace, recording the sensor position, recording the Kinect point

[1]http://pointclouds.org

cloud of the scene, then locating the sensor in the point cloud. Since the pushing motions performed in our experiments always follow an approximately linear trajectory, we proceed by using orthogonal distance regression via singular value decomposition to fit a 3-D line to the push trajectory. Finally, we find the point of intersection between this fitted line and the pre-push object point cloud, infer this to be the contact point, and finally transform both the pre-push and post-push object point clouds as well as the push trajectory to the action frame as defined by the contact point and the fitted line.

C. Action-grounded shape features

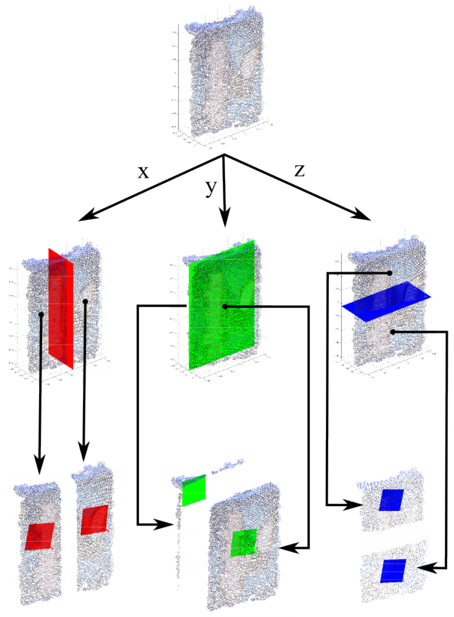

Fig. 2. Partitioning a sample object point cloud into sub-parts. Top row: original pre-push object point cloud. Middle row: partitioning planes divide the point cloud evenly in each dimension to create sub-parts. Bottom row: planes are fitted to each sub-part for feature extraction.

With both pre-push and post-push object point clouds now grounded in the action coordinate frame, we turn to generating a feature descriptor that describes the shapes of the object point clouds with respect to the pushing action and that is rich enough to capture the resulting affordance effects. The main idea behind our approach is to divide the object point clouds into cells of sub-parts and use the properties of the sub-parts of the point clouds as a basis for the feature descriptor. More concretely, we divide each object point cloud evenly with respect to its minimum and maximum points along each coordinate axis such that there are seven cells that overlap for redundancy: one for the overall point cloud, two for the x-axis, two for the y-axis, and two for the z-axis. We then use two types of feature descriptors in each cell. To gauge the position of the sub-part in each cell relative to the action frame, we find the centroid of the points in the cell, which gives us three features. To gauge the shape of the sub-part in each cell relative to the action frame, we fit a planar surface to the points within the cell and the x and y components of the plane normal as features. We discard the z component to reduce the feature space dimensionality, the x and y components being sufficient to quantify the angle of the plane from the normal. Examples of these features being extracted from different point clouds are shown in Figure 5.

Using these five types of features, three for relative part position and two for planar surface fit orientation, we extract the five features for each part. This results in the following list of 35 features that are extracted before $\{O^1, \ldots, O^{35}\}$ and after $\{E^1, \ldots, E^{35}\}$ the push interaction:

- global object point cloud features.
- x-axis division, left part features.
- x-axis division, right part features.
- y-axis division, front part features.
- y-axis division, back part features.
- z-axis division, top part features.
- z-axis division, bottom part features.

III. BOOTSTRAPPING OBJECT PUSH AFFORDANCES

To enable the type of bootstrap learning discussed in the introduction, we frame our scenario as a multi-view online learning problem. Multi-view learning [17], as well as the related fields of cross-modal and multi-modal learning, [18]–[21], are machine learning areas which are concerned with the problem of learning from data represented by multiple distinct feature sets in different data views or modalities. Given that common theme, the learning objective may otherwise differ depending on the particular context [17]. In our scenario, object pre-push shape features $\mathbf{x}_i = [O_i^1, \ldots, O_i^{35}]^T$ define the feature space in one data view, the input space, whereas object post-push shape features $\mathbf{y}_i = [E_i^1, \ldots, E_i^{35}]^T$ define the feature space in another view, the output space. Our learning goal is to find significant clusters amongst the $\mathbf{y}_i$ feature vectors in the output space, then use these clusters as classes to train a classifier using the $\mathbf{x}_i$ feature vectors in the input space, that is to find a mapping $f : \mathbb{R}^n \to \mathbb{N}$ from input space feature vectors to class labels representing affordances grounded in output space feature clusters. We considered this as a multi-view learning problem since there is a natural separation between the two feature spaces under consideration, which model potential causes and potential effects respectively, and we wished to use information in the output view (effect) to influence learning in the input view (cause).

With that in mind, we extended the self-supervised multi-view online learning algorithm originally proposed in [5]. This algorithm is self-supervised in the sense that the data distribution in the output space coupled with the co-occurrence information, is used to form a supervision signal that directs learning in the input space. The algorithm achieves something similar to methods for self-discovery and prediction of affordance classes proposed in other work [13], but can be trained online and makes use of the class information for discriminative learning as it emerges dynamically during training. In the following, we summarise the relevant parts of the algorithm. Further detail can be found in [5].

A. *Self-supervised Multi-view Online Learning*

Assuming there are two datasets of co-occurring data, $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^m \mid i = 1, \ldots, D\}$ in the input space and $\mathcal{Y} = \{\mathbf{y}_i \in \mathbb{R}^n \mid i = 1, \ldots, D\}$ 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 arrive in an online data stream, we aim to represent each space via vector quantization [22] using *codebooks of prototype vectors* $\mathcal{W} = \{\mathbf{w}_i \in \mathbb{R}^m \mid i = 1, \ldots, M\}$ for the input space and $\mathcal{V} = \{\mathbf{v}_i \in \mathbb{R}^n \mid i = 1, \ldots, N\}$ for the output space respectively, approximating the data distributions in each view. We implement multi-view connectivity between the two codebooks via a weight matrix which we refer to as *co-occurrence mapping*, defined as follows:

$$\mathcal{H}(\mathcal{W}, \mathcal{V}) = \begin{bmatrix} \gamma_{1,1} & \gamma_{1,2} & \cdots & \gamma_{1,\mathrm{N}} \\ \gamma_{2,1} & \gamma_{2,2} & \cdots & \gamma_{2,\mathrm{N}} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{\mathrm{M},1} & \gamma_{\mathrm{M},2} & \cdots & \gamma_{\mathrm{M},\mathrm{N}} \end{bmatrix} \tag{1}$$

where the $\gamma_{i\,j}$ are weights that are used to record the level of data co-occurrence between nearest-neighbour prototypes in each of the codebooks and are adjusted by applying the Hebbian rule to cross-view prototype activations. We aim to find significant clusters of prototypes in the output space which we dub *class clusters* that we treat as classes to be used for discriminative learning in the input space. Codebook training within the input space uses two learning phases. The first phase involves unsupervised clustering of the prototypes in each data view using the self-organizing map (SOM) algorithm [22] such that the data distributions are roughly approximated. The second phase involves self-supervised discriminative learning using a modified form of learning vector quantization (LVQ) [5] such that the positions of the prototypes in the input space are refined for classification purposes using cross-view co-occurrence information. Throughout, the nearest-neighbour rule is employed using a weighted squared Euclidean distance,

$$d^2(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{n} \lambda_i (x_i - w_i)^2, \tag{2}$$

where x_i and w_i are feature components of $\mathbf{x}$ and $\mathbf{w}$ respectively, and the λ_i are weighting factors for each feature which facilitate feature relevance determination as described later in Section III-E.

The co-occurrence mapping can be used to infer the relationship between the prototypes in the different feature spaces by projecting the weights for a given prototype in one space onto the codebook in the other space. Given prototype $\mathbf{w}^i \in \mathcal{W}$ we define

$$P(\mathbf{v}_j|\mathbf{w}_i) = \frac{\gamma_{i\,j}}{\sum_j^N \gamma_{i\,j}} \tag{3}$$

which describes the probability of prototype $\mathbf{v}_j \in \mathcal{V}$ matching prototype $\mathbf{w}_i \in \mathcal{W}$ based on the co-occurrence mapping, where $\gamma_{i\,j}$ is the connection weight from (1) between prototypes $\mathbf{w}_i$ and $\mathbf{v}_j$. Thus for a given $\mathbf{w}^i \in \mathcal{W}$, making

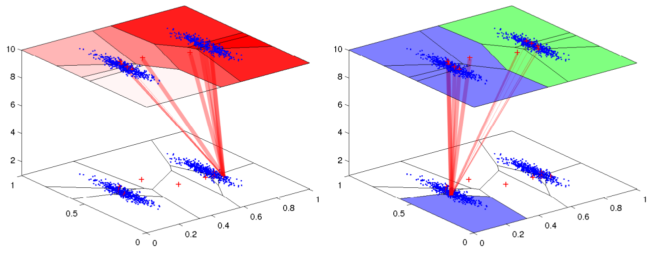

Fig. 3. Multi-view classifier construction. Left figure: Cross-view co-occurrence weight projection. The weights of the $\mathcal{H}(\mathcal{W}, \mathcal{V})$ mapping (red lines) from a prototype $\mathbf{w}_i$ in the $\mathcal{W}$ input space codebook (lower red crosses & Voronoi regions) are projected using (3) (upper red shaded regions) onto the prototypes of the $\mathcal{V}$ output space codebook (upper red crosses & Voronoi regions) Right figure: Class discovery and cross-view class projection. After both the codebooks and co-occurrence mapping are trained (cf. Section III-A), the upper codebook prototypes are clustered to form class labels (upper blue & green regions, cf. Section III-B). An appropriate class label is then projected onto a prototype $\mathbf{w}_j \in \mathcal{W}$ (lower blue region) using (5) (cf. Section III-C).

such projections for all $\mathbf{v}^i \in \mathcal{V}$ forms a spatial probability distribution over codebook $\mathcal{V}$ and is a useful tool that allows us to measure for measuring how one data view looks from the perspective of another in terms of past co-occurrences of data. A visualisation is provided on the left side of Fig. 3.

B. *Class Discovery*

In order to find class clusters in the output space, we treat the prototype vectors as data points and employ traditional unsupervised clustering to cluster the prototypes. Often, the k-means clustering algorithm is used for such purposes, but one issue with regular k-means is that k, that being the number of clusters, must be selected in advance. It is possible, however, to augment the algorithm such that k may be estimated automatically. We employ the X-means algorithm [23] for that purpose here to find both the optimal k^* number of clusters for the prototypes in output space codebook $\mathcal{V}$, as well as the actual clustering $C_{k^*}^{\mathcal{V}} = \{V_1, \ldots, V_{k^*}\}$, where the V_i are subsets of prototypes in $\mathcal{V}$. The V_i clusters that are discovered in this way are treated as classes grounded in output space features that can be mapped back onto the input space layer using Hebbian projection. This class discovery and projection process is visualised in Fig. 3.

C. *Cross-View Class Projection*

For cross-view classification, we require a mapping $f : \mathbb{R}^m \to L(\mathcal{V})$ that maps input space samples to class labels, where $L()$ is some labelling function. To realise this labelling function, the $C_{k^*}^{\mathcal{V}}$ class clusters found in output space codebook $\mathcal{V}$ via class discovery are projected back to the input space codebook $\mathcal{W}$. By summing the posterior probabilities $P(\mathbf{v}_j|\mathbf{w}_i)$ provided by such a projection, we can determine the posterior probability of class cluster V_l in output view codebook $\mathcal{V}$ given prototype $\mathbf{w}^i$ in input space codebook $\mathcal{W}$ as follows

$$P(V_l|\mathbf{w}_i) = \sum_{\mathbf{v}_j \in V_l} P(\mathbf{v}_j|\mathbf{w}_i) P(\mathbf{w}_i). \tag{4}$$

This allows us to assign an output space class cluster label to each of the prototypes in the input space codebook by maximizing the category cluster posterior probability for each of them. Thus, given $\mathbf{w}^i$, we define a labelling function

$$L(\mathbf{w}_i) = \arg\max_{l=1,\ldots,k^*} P(V_l|\mathbf{w}_i) \tag{5}$$

that labels the input space prototypes on that basis.

D. 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 (5), the weighted squared Euclidean distance function from (2), and the nearest-neighbour rule as follows:

$$f(\mathbf{x}) = L\left(\arg\min_{\mathbf{w}_i \in \mathcal{W}} d^2(\mathbf{x}, \mathbf{w}_i)\right). \tag{6}$$

E. Feature Relevance Determination

Some features can prove to be more relevant than others for class prediction, and determining the extent of their relevance and exploiting this information can improve classification accuracy. To this end, we make use of an algorithm developed in previous work for feature relevance determination, specifically Algorithm 1 from [24]. It exploits the positioning of the prototypes in the input feature space to estimate Fisher criterion scores for the input dimensions, and subsequently, to estimate the λ_i weighting factors in (2) for an adaptive distance function that accounts for feature relevance with respect to classifier output. Due to the bootstrapped nature of the learning algorithm described in this paper, class information may not be fully formed at a given time step during training. Therefore, to avoid corrupting the online learning process we do not apply the feature weights during training, but apply them at classification time instead. Further details on this method may be found in [24].

IV. EXPERIMENTS

To test our affordance learning system, the experimental environment was set up as shown in Figure 1. We selected 5 household objects (cf. Fig. 4) for the experiments: 4 flat-surfaced objects; a book, a marshmallow box, a cookie packet, and a biscuit box, and 1 curved-surfaced object; a yoghurt bottle. A dataset was collected as follows. A number of object push tests were carried out for each of the 5 objects listed previously and the resulting data was processed, leaving 134 data samples. Objects were placed at random start locations within the workspace and within view of the Kinect sensor, and the human experimenter would perform straight-line pushes on the objects, attempting to keep the pushes within reasonable limits of 5 different categories: pushing through the top, bottom, left, right and centre of the objects respectively, from the direction of the field of view of the Kinect. For evaluation the samples were hand-labelled with four ground truth labels: left rotation, right rotation, forward translation and forward topple, but this information was of course not used by our self-supervised learning algorithm, and was used strictly for evaluation and for training the supervised classifiers outlined below in Section IV-A. Sample object interactions are shown in Fig. 5. The results presented below examine three different aspects of our learning framework: class discovery, class prediction and feature relevance determination.

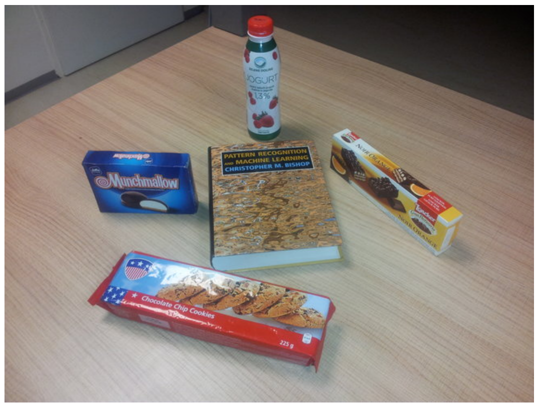

Fig. 4. Test objects used in our experiments.

A. Evaluation Procedure

To test our self-supervised learning paradigm on the dataset described above, we performed 10-fold cross-validation, evaluating performance online at regular intervals over the training period. Our self-supervised learning vector quantization algorithm (SSLVQ) [5], as well as a variation employing feature relevance determination at classification time (SSLVQ (FRC)) [5], [24], were compared alongside the supervised LVQ algorithms GLVQ [25], GRLVQ [26] and SRNG [27] in this online evaluation. Two main experiments were performed, the first one performing 10-fold cross-validation for 1 epoch over the training data to test short-term training performance, and the second one performing 10-fold cross-validation for 10 epochs over the training data over 10 trials to test more long-term training performance. In each case, codebooks in both the input space and output space consisted of 64 prototypes arranged in a 8×8 hexagonal lattice with a sheet-shaped topology [5], [22]. The feature weights of the codebook prototype vectors were randomly initialized to test the abilities of the algorithms to learn from scratch. The 10-fold cross-validation was therefore performed in 10 trials and results were averaged in order to account for the variation in codebook initialization between trials. The learning phase was switched from unsupervised learning to self-supervised learning one tenth of the way through training. Batch SVM was also performed outside of the online evaluation for reference. Batch methods, as opposed to online methods that are trained sample-by-sample, have access to the entire training set during training, and therefore usually provide superior results. SVM parameters were optimized using cross validation over the training data prior to training.

Given the self-supervision aspect, the evaluation criteria, by necessity, differed from the traditional match-counting

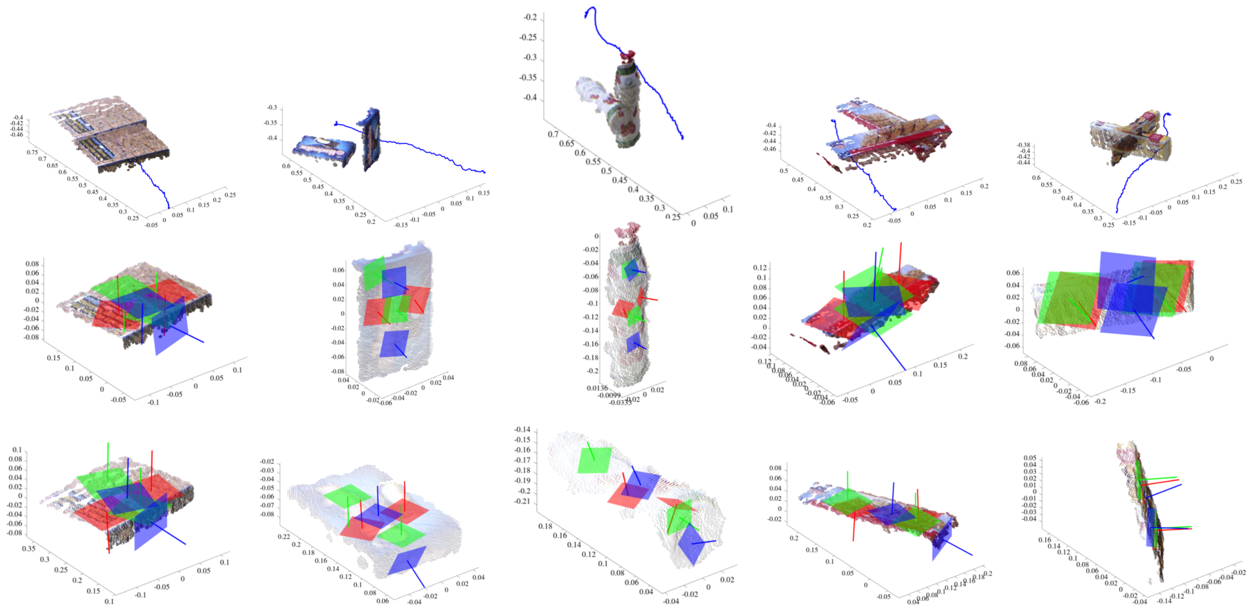

Fig. 5. Action-grounded shape feature extraction. Top row: pre-push and post-push 3-D point clouds and action trajectories for the five test objects being pushed in various different ways. Second & third rows: action-grounded shape feature extraction (cf. Section II-C) for the pre-push and post-push point clouds, respectively. Plane fits are shown in red for the x-axis divisions of the point clouds, in green for the y-axis divisions, and in blue for the z-axis divisions. Four different affordances are visible in the columns from left to right: forward translation, forward topple, right rotation, and left rotation.

utilized to evaluate fully-supervised classifiers. Clusters of prototypes were found in the output space codebook as described in Section III-B and subsequently matched to the ground truth classes by first matching all ground truth labelled training data to nearest-neighbour output space prototypes, then assigning each class cluster the ground truth label which their respective prototypes matched to most frequently. Then, given a test sample consisting of an input space test vector $\mathbf{x}_i$ and an output space test vector $\mathbf{y}_i$, the input space codebook was tasked with predicting an output space class cluster V_j for $\mathbf{x}_i$ using the process described in Section III-D. The output space test vector $\mathbf{y}_i$ was then matched to a class cluster V_k in the output modality via the nearest-neighbour rule. If the V_j class 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 correct classification.

B. *Results: Full Feature Set*

1) Class Discovery: An important consideration in evaluating whether or not our algorithm is capable of self-supervised multi-view learning is to examine if it is capable of successfully finding class clusters in the output space, without which self-supervised discriminative learning in the input space would not be possible. Recall that this is achieved by clustering prototypes in the output space at classification time using X-means clustering (cf. Section III-B). But how quickly do the prototypes position themselves such that this clustering may happen successfully? The leftmost graphs of Figures 6 and 7 answer this question by showing the rate at which ground truth labelled output space test samples fall within output space class clusters with matching ground truth labels (cf. Section IV-A) over time. As is evident from the graph for short-term training over 1 epoch, performance starts to peak around half-way through training and reaches near-optimal performance by the end, in which case output view test samples were correctly matched to the four ground-truth affordance classes over 94% of the time by the end of training. In the case of long-term training over 10 epochs, near-optimal performance is achieved early in the training process and is maintained throughout, meaning that cross-view prediction should at least have the opportunity to reach optimal ground truth prediction rates early on.

2) Class Prediction: With regard to class prediction, batch SVM scores 92% classification accuracy using ground truth labels, and although the other learners were not expected to perform at this level given the fact that they were trained on-line from a random initialization, this serves as a good reference point. Turning to the middle graphs on class prediction in Figs. 6 and 7, most of the learners perform poorly in short-term training, most of them scoring below 50% classification accuracy, likely due to there being insufficient training time to tackle the complexity of the problem. The self-supervised learners, reaching a rate of 42%, out-perform all of the supervised classifiers apart from GRLVQ, which reaches just under 50% accuracy. The self-supervised learners capitalise slightly in this case from the dynamic class labelling process that occurs when classifying as described in Section III-D, while the class labels for the prototypes of the supervised classifiers are randomly distributed, which means that it takes time for the labelled prototypes to cluster appropriately. Long-term training sees all of the learners performing better,

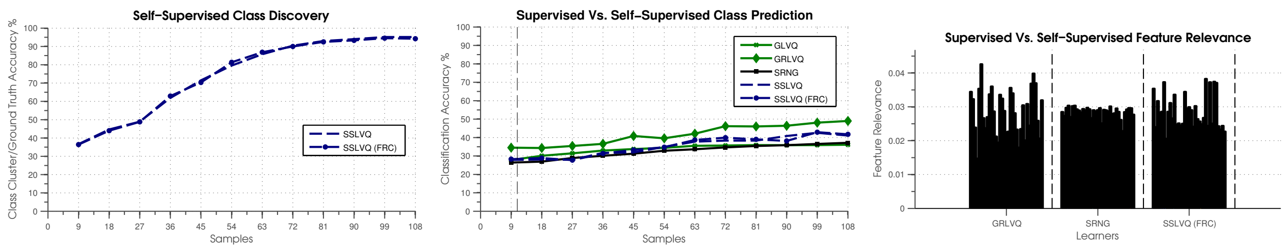

Fig. 6. Results for 1 epoch of online training over the full feature set. From left to right: class discovery, class prediction and feature relevance results are shown for 10-fold cross-validation averaged over 10 trials with random prototype initialization (cf. Section IV-A). Bold vertical dashed lines in the class prediction graphs indicate training phase shifts from unsupervised to self-supervised learning (cf. Section III-A). Comparitive batch SVM class prediction score: 92%.

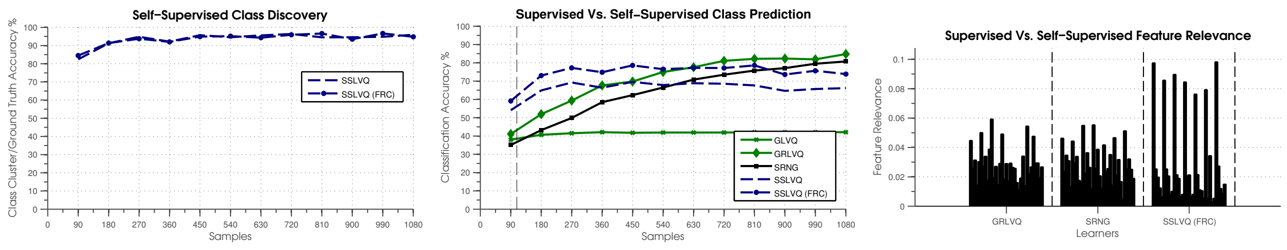

Fig. 7. Results for 10 epochs of online training over the full feature set. Results are shown for 10-fold cross-validation averaged over 10 trials with random prototype initialization (cf. Section IV-A). Comparative batch SVM class prediction score: 92%.

with GRLVQ scoring 85%, SRNG scoring 81%, SSLVQ (FRC) scoring 74% SSLVQ scoring 66%, and GLVQ scoring 42%. by the end of training. In this instance it is possible to see the benefit of feature relevance determination, with its addition boosting the performance of self-supervised learning by 8%. GLVQ is known to suffer issues with poor prototype initialization, hence its relatively poor performance. SRNG has a slower update rule than GRLVQ, which likely accounts for its relatively slow adaptation here.

3) Feature Relevance Determination: The rightmost graphs of Figures 6 and 7 show average feature relevances at the end of training as determined by the three learners GRLVQ, SRNG and SSLVQ (FRC) which have feature relevance determination capabilities. In the 10-epoch case, all three of them highlight the following list of features as being most significant for class prediction:

- Overall point cloud, centroid x-coordinate.
- x-axis, left side part, centroid x-coordinate.
- x-axis, right side part, centroid x-coordinate.
- y-axis, front side part, centroid x-coordinate.
- z-axis, top side part, centroid x-coordinate.

In general, the object part centroid features were determined to be the most discriminative for affordance class prediction.

C. Results: Reduced Feature Set

Using this knowledge, we performed an additional set of experiments on a reduced feature set over the same experimental data, this time using only the centroids of the sub-parts to form the feature vectors in both the input and output spaces. The experimental parameters were kept the same as in Section IV-B and the results for these experiments are shown in Figures 8 and 9. This refinement of the feature spaces boosts the predictive performance of SSLVQ (FRC) up to 87% over 10 epochs of training, a significant improvement over the full feature set.

V. CONCLUSIONS

In summary, the main contribution of this paper was the proposal of an action-grounded 3-D visual feature descriptor to be used for bootstrapping object affordances in autonomous robots when little prior information is available about the objects. We have demonstrated through experimental results how, when used in combination with a self-supervised learning algorithm, this feature descriptor is effective at facilitating both affordance class discovery and prediction in an online learning setting with a number of initially unknown objects and object affordances. A feature relevance determination extension to the self-supervised algorithm was also shown to boost affordance class prediction results by emphasizing the discriminative contribution of particular features within the descriptor.

With regard to future work, firstly, we aim to migrate the affordance learning techniques presented here to a humanoid robotic system. The design of the shape features could potentially be improved by matching 2-D invariant image features to 3-D surface features and thereby directly tracking object part motion during interaction. Looking towards improving the affordance learning aspects, we aim to implement regression capabilities that would allow for continuous prediction of object and object part positions. The bootstrapping of significant affordance classes as presented in this paper would mitigate this task, constraining the problem by guiding the development of multiple continuous models.

REFERENCES

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

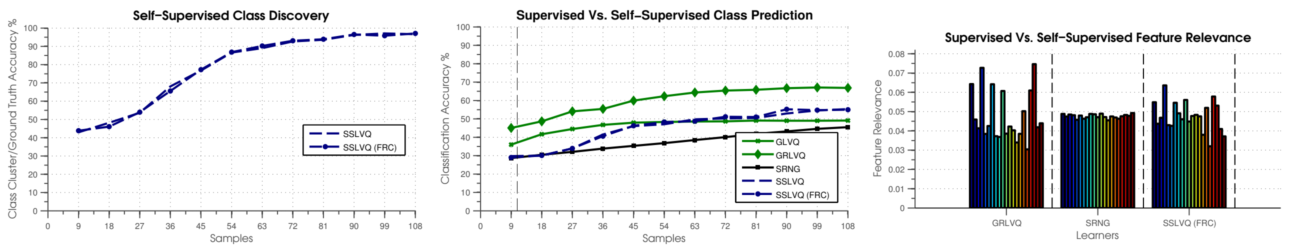

Fig. 8. Results for 1 epoch of training over the reduced feature set. Class discovery, class prediction and feature relevance results are shown for 10-fold cross-validation averaged over 10 trials with random prototype initialization (cf. Section IV-A). Comparative batch SVM class prediction score: 96%.

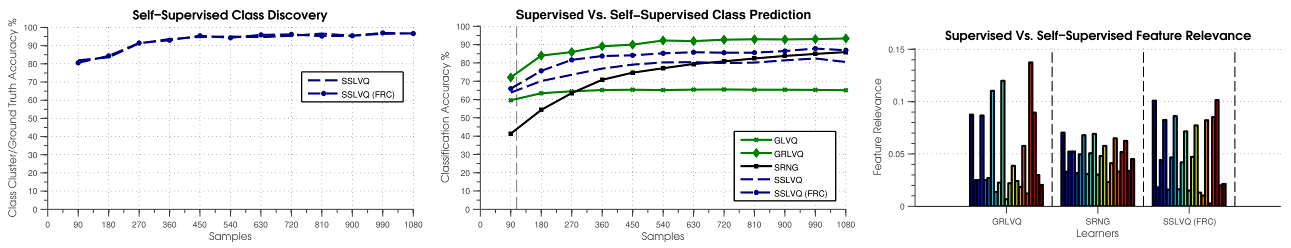

Fig. 9. Results for 10 epochs of online training over the restricted feature set using only part centroids. Again, results are shown for 10-fold cross-validation averaged over 10 trials with random prototype initialization (cf. Section IV-A). Batch SVM class prediction score: 96%.

[2] 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)*, vol. 3, 2003.

[3] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Learning object affordances: From sensory-motor coordination to imitation," vol. 24, no. 1, p. 15–26, 2008.

[4] D. Omrčen, C. Boge, T. Asfour, A. Ude, and R. Dillmann, "Autonomous acquisition of pushing actions to support object grasping with a humanoid robot," in *Proceedings of the 9th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Dec. 2009, pp. 277 –283.

[5] B. Ridge, D. Skočaj, and A. 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)*. Anchorage, USA: IEEE, May 2010, pp. 5047–5054.

[6] M. Kopicki, S. Zurek, R. Stolkin, T. Morwald, and J. Wyatt, "Learning to predict how rigid objects behave under simple manipulation," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 5722 –5729.

[7] G. Fritz, L. Paletta, M. Kumar, G. Dorffner, R. Breithaupt, and E. Rome, "Visual learning of affordance based cues," vol. 9, p. 52–64, 2006.

[8] I. Cos-Aguilera, L. Canamero, and G. Hayes, "Using a SOFM to learn object affordances," in *Proceedings of the 5th Workshop of Physical Agents (WAF'04), Girona, Spain*, 2004.

[9] J. Sinapov and A. Stoytchev, "Detecting the functional similarities between tools using a hierarchical representation of outcomes," in *Proceedings of the 7th IEEE International Conference on Development and Learning (ICDL)*, Aug. 2008, pp. 91 –96.

[10] S. Griffith, J. Sinapov, M. Miller, and A. Stoytchev, "Toward interactive learning of object categories by a robot: A case study with container and non-container objects," in *Proceedings of the 8th IEEE International Conference on Development and Learning (ICDL)*. IEEE, June 2009, pp. 1–6.

[11] E. Ugur, E. Oztop, and E. Sahin, "Goal emulation and planning in perceptual space using learned affordances," 2011.

[12] L. Paletta and G. Fritz, "Reinforcement learning of predictive features," in *Proceedings of 31st Workshop of the Austrian Association for Pattern Recognition (AAPR/OAPR)*, 2007, p. 105–112.

[13] 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 Intelligent Robots and Systems (IROS)*, Oct. 2012, pp. 3260 –3267.

[14] T. Hermans, F. Li, J. M. Rehg, and A. F. Bobick, "Learning stable pushing locations," in *Proceedings of the Third Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics (ICDL-EpiRob)*, Osaka, Japan, Aug. 2013.

[15] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," vol. 24, no. 6, p. 381–395, 1981.

[16] R. B. Rusu, "Semantic 3D object maps for everyday manipulation in human living environments," Ph.D. dissertation, Computer Science department, Technische Universitaet Muenchen, Germany, Oct. 2009.

[17] S. Sun, "A survey of multi-view machine learning," p. 1–8, 2013.

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

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

[20] S. Bickel and T. Scheffer, "Multi-view clustering," in *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, Washington D.C., USA, Nov. 2004, pp. 19–26.

[21] V. de Sa, P. Gallagher, J. Lewis, and V. Malave, "Multi-view kernel construction," vol. 79, no. 1, pp. 47–71, 2010.

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

[23] D. Pelleg and A. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proceedings of the Seventeenth International Conference on Machine Learning*, vol. 1, 2000, p. 727–734.

[24] B. Ridge, A. Leonardis, and D. 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, Feb. 2012.

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

[26] B. Hammer and T. Villmann, "Generalized relevance learning vector quantization," vol. 15, no. 8-9, pp. 1059–1068, 2002.

[27] B. Hammer, M. Strickert, and T. Villmann, "Supervised neural gas with general similarity measure," vol. 21, no. 1, p. 21–44, 2005.