Online Incremental Learning with Forgetting*

Authors Name/s per 1st Affiliation (*Author*)
line 1 (of *Affiliation*): dept. name of organization
line 2-name of organization, acronyms acceptable
line 3-City, Country
line 4-e-mail address if desired

Authors Name/s per 2st Affiliation (*Author*)
line 1 (of *Affiliation*): dept. name of organization
line 2-name of organization, acronyms acceptable
line 3-City, Country
line 4-e-mail address if desired

Authors Name/s per 3st Affiliation (*Author*)
line 1 (of *Affiliation*): dept. name of organization
line 2-name of organization, acronyms acceptable
line 3-City, Country
line 4-e-mail address if desired

***Abstract*— Agents such as service robots, operating in open-ended domains, must be able to continuously learn to recognize new objects. This continuous learning process however may pose some challenges, particularly in memory constrained systems, where it may become unfeasible to keep the representation of all known objects due to their large number. This work proposes an online incremental unsupervised and supervised learning approach comprising a weight-based 'forgetting' mechanism targeting open-ended learning scenarios. To achieve this, both the object categories as well as the visual codebooks used to encode their histogram representations are learned online and concurrently. The Bag of Words (BOW) model is used in order to obtain these sparse codebooks. Local and global weights associated to the features (i.e. key points extracted from the objects) are then used in order to encode the 'forgetting' mechanism, by which when a feature is forgotten its weight is redistributed over its nearest features according to their distances. A memory decay implementation based on these weights is also tested in order to assess its impact in the accuracy of the system. Preliminary results using this 'forgetting' mechanism show similar accuracy in the recognition of objects when compared to the use of all the learned features.**

Keywords—online learning; incremental learning; bag of words model; supervised learning; unsupervised learning; online k-means; open-ended learning; forgetting mechanism

I. INTRODUCTION

In order to properly perform their designated tasks, agents such as service robots, operating in open-ended domains must be able to continuously learn to recognize new objects. As part of this incremental learning process, the representation of already know objects should also be further refined over time [1]. Due to the large number of potentially different objects that the agent may encounter during its service lifetime it may not be possible to fully pre-train it. In such cases the agent should be able to learn on site in an online and incremental fashion [1].

The Bag of Words (BOW) model is a common approach used to generate compact representations of the object views in the form of codebooks. These visual codebooks can be [2] dense (e.g. texton representation) or sparse (e.g. key points) and can be constructed offline or online [1]. Using this compact and simplified representation of the objects, the object categories can be encoded by histograms constructed over the distribution of these visual words [3]. These histograms in turn may be used to perform visual object recognition (VOR).

The potentially large number of objects that the agent may encounter during its service lifetime however may present additional challenges, even when using these compact representations. An example of this may be in memory constrained agents, in which case memory exhaustion may prevent the system from learning to recognize new objects. One possible solution to overcome this issue is to implement a 'forgetting' mechanism by which the system 'forgets' features associated with unseen objects over time allowing for new objects to be learned. Ideally however this forgetting should not be immediate, but occur slowly over time allowing for some kind of memory of the forgotten features to be retained. This in turn would also allow the system to somewhat recall partially forgotten objects at a later time, possibly without having to learn them anew. This also means that some criterion should be in place in order to properly decide which features should be forgotten.

This work proposes an online incremental learning approach incorporating a 'forgetting' mechanism suitable for open-ended learning scenarios. In order to achieve this, the visual codebooks are learned using an unsupervised learning approach based on an online adaptation of K-Means. The object categories are then represented as histograms computed over the distribution of these visual words (i.e. the centroids of the clusters) using a supervised learning approach. Both the visual codebooks as well as the object categories are learned online and concurrently. Weights are then used to encode the 'forgetting' mechanism by which when a given feature is forgotten, its weight is redistributed by its neighboring features according to their distances.

The 'forgetting' mechanism is implemented using two mechanisms: *redundancy* and *memory decay*. The *redundancy* mechanism allows the system to detect redundant features (i.e. features similar to other features previously seen by the system) and therefore ideal candidates to be forgotten. The *memory decay* mechanism allows the system to decide which features should be forgotten first based on their weights. This mechanism is similar to simulated annealing in the sense that the weight of the features decays slowly over time until it reaches a 'temperature' that gives an indication that the feature is a candidate to be forgotten.

The remainder of this paper is structured as follows. Section II reviews the related work. Section III presents the overall system overview and the concepts that support it. Section IV presents the implementation details. The tests performed and the results obtained are discussed in Section V and Section VI presents the conclusions and future work.

II. Related Work

There has been a lot of research in order to find methodologies that can produce optimal BOW based codebooks. In [4] the construction of discriminative codebooks is approached as an optimization problem. Category information of local image features is incorporated as an additional factor in the construction of these codebooks. In [5] image features are associated to the visual words in the codebook using soft assignment. This approach proposes the incorporation of ambiguity in the codebook model in order to improve its expressiveness. While tackling the codebook construction problem it is not clear how these approaches could be used in open-ended learning scenarios, specifically considering that the codebooks may change continuously over time.

In [6] the codebooks are constructed online using a modified version of agglomerative clustering. In this approach the clusters are obtained via incremental merging. The merging criterion used takes the global distribution of the data into account in order to optimize the efficiency of the codebook with respect to its repetitiveness and discriminative power.

In [7] a recursive least squares dictionary learning algorithm RLS-DLA, a generalization of the continuous K-Means algorithm, is proposed in order to continuously update the dictionary (i.e. the codebook). This approach seeks to converge towards a finite codebook by gradually increasing a forgetting factor λ. It is however not clear how the value of this factor could be adaptively changed in an open-ended learning domain. Also this factor is used mainly for purposes of convergence of the codebook (i.e. the clustering) and not as a real forgetting mechanism.

All of the approaches mentioned so far concentrate specifically in the construction of optimal codebooks. In [1] these codebooks are learned concurrently with the object categories in order to address the problem of online incremental learning in open-ended domains. This approach uses a Gaussian Mixture Model (GMM) to model the distribution of the observed features. In order to construct the codebook, a transient GMM composed of a large number of components is fitted using an Expectation Maximization (EM) framework and then simplified in order to collapse overlapping components in the mixture. A global GMM defines the codebook, with each of its components corresponding to a word in the codebook. While this work tries to overcome some of the issues related to GMMs, estimating the optimal number of components in the mixture as well as the parameters of the Gaussians can still be difficult and computationally costly.

None of the methodologies presented tackles the problem of forgetting features (and ultimately objects) over time in open-ended learning scenarios.

III. Conceptual Model

This section presents the main concepts that support this work, in particular, the memory mechanism and its components. That is, the conceptual model that establishes the basis for the implemented application. Implementation details are presented in Section IV.

A. Memory

The application's memory (the memory) contains M descriptors of local features (features) taken from object views seen by the application. These features are represented by vectors (e.g., spin images) [8], [9]. A feature f_i, $i = 1 \ldots M$, has a global weight, a category-level weight (local weight), a vector representing it in N dimensions and information about the category and the cluster to which it belongs. When a new feature f_j is added to the application's memory its global and local weights are set to 1.0. A category represents an object category and it is represented by a set of features, of previously seen object views, belonging to that category. The memory has a limit M^+, the maximum amount of features it can hold, that can be statically or dynamically established and a forgetting mechanism that includes feature redundancy, memory decay and reinforcement. When M^+ is reached some existing features are forgotten. These features can be selected randomly or based on cluster-level and category-level redundancy.

B. Codebook

Features in memory are organized in K clusters. The centroids of these clusters represent the visual words of the codebook. This codebook is constructed online in an incremental fashion, therefore K is not known apriori and may vary over time. A cluster's centroid, c_j, is computed using the formula presented in (1), where p_i is a feature that belongs to cluster j and w_i is its global weight.

$$c_j = \frac{\sum_i (w_i \times p_i)}{\sum_i w_i} \tag{1}$$

C. Weights as Memories of Forgotten Features

When a feature f_i is selected to be forgotten its global weight is distributed by its neighbors that can be determined as the features in a hypersphere with a specified diameter and center at f_i. Features can change from one cluster to another. Therefore, when a feature is forgotten, its global weight should be distributed by its neighbors, independently of the clusters to which they belong.

Let G be the set of neighbors of f_i, in a hypersphere with a specified diameter and center at f_i, $\dim(G) = N$, d_{ij} the distance of f_i to $g_j \in G$, w_i the global weight of f_i and w_j the global weight of g_j. Then, w_j is updated using the rule presented in (2).

$$w_j = w_j + \frac{\frac{1}{d_{ij}}}{\sum_{k=1}^{N} \frac{1}{d_{ik}}} * w_i \tag{2}$$

In order to represent categories as histograms using the words in the computed codebook, we must keep a memory of which features of a category were forgotton. Therefore, when a feature f_i is forgotten, its category-level weight (local weight) is distributed by all the features that belong to the category to which f_i belongs.

Let H be the set of all features that belong to the category to which f_i is assigned to, $\dim(H) = N$, d_{ij} the distance of f_i to $h_j \in H$, lw_i the local weight of f_i and lw_j the local weight of h_j. Then, lw_j is updated using the rule presented in (3).

$$lw_j = lw_j + \frac{\frac{1}{d_{ij}}}{\sum_{k=1}^{N} \frac{1}{d_{ik}}} * lw_i \quad (3)$$

D. Forgetting Features

Let C be the set of categories and F the set of features in the application's memory. Then, the rule φ presented in (4), where $GW(f_i)$ represents the global weight of f_i and $|f_i, f_j|$ the distance between f_i and f_j, establishes the condition for feature redundancy using the feature redundancy threshold, ε, (a) and the condition to forget features using the memory decay threshold, δ, (b). In both cases, f_i will be forgotten.

$$\varphi = \begin{cases} |f_i, f_j| \leq \varepsilon, f_i \wedge f_j \in F, i \neq j & (a) \\ GW(f_i) \leq \delta & (b) \end{cases} \quad (4)$$

Note that, in this work we do not consider the condition where the application's memory limit M^+ is reached. This was left as future work.

IV. Implementation

We implemented an application that follows the conceptual model presented in Section III. For simplification purposes, the application introduces some changes to the original conceptual model. They will be discussed in the remainder of this section.

A. Technologies Used

The proposed application was implemented using C++ and uses the PCL[1] (Point Cloud Library) to manipulate the object views. **TABLE I.** presents the PCL classes used by the application's modules and the values considered for their main configuration parameters. We use a voxel grid filter for point cloud downsampling and key points extraction. We assume that the object views seen by the application (i.e., stored in the views repository) are already properly segmented and cleaned up. Therefore, we do not use any other filters on the object views (e.g., a conditional removal filter to remove outliers). The object views are represented using spin image feature descriptors (features) and the distance between features is computed using k-d trees. This includes the *k* nearest neighbors and the neighbors within a specified radius of a specific feature.

B. Architecture

The application has five main modules: the Object Viewer, the Feature Extractor, the Category Recognizer, K-Means and the Memory modules. They are represented by the rectangles and the cylinder (with solid lines) shown in Fig. 1. The arrows represent the exchange of data and information between the modules. Arrows with a solid line depict send operations and arrows with a dashed line depict get operations. Each module corresponds to (was implemented as) a C++ class. The Memory module is a special module that plays a central role in the application.

The Object Viewer reads object views stored on disk and shuffles the categories and object views to be used during the training and the testing phases. The Feature Extractor extracts key points from the object views and computes their spin image feature descriptors. The Category Recognizer assigns the features that represent an object view to a known category. K-Means assigns features to clusters and manages the overall clustering by creating new clusters according to a split criterion. The Memory module manages the memory of the application. This includes: the application's forgetting mechanism, memory decay, feature redundancy and weights redistribution. Fig. 1 presents a high level overview of the architecture of the application.

TABLE I. PCL CLASSES USED BY THE MAIN APPLICATION MODULES.

PCL Class|Paremeters	Usage	Application Module
VoxelGrid voxel size = 0.015	Point cloud downsampling and key points extraction.	FeatureExtractor
SpinImageEstimation *window width = 8* *support length = 0.1m* *support angle = 90º* *min points neighbors = 0*	Object views representation.	FeatureExtractor
KdTreeFLANN *nearestKSearch*(...) *radiusSearch*(...) search radius = 0.065	Computation of distances between spin images.	Searcher

C. Application Main Execution Flows

The application has two main execution flows presented in Fig. 2 and Fig. 3. When the application 'sees' a new object it will extract key points from the object view and compute spin image feature descriptors for those key points. These feature descriptors (or features) represent the object view. Then, the new features will be 1) added to memory, 2) assigned to a cluster, 3) assigned to a category and 4) checked for redundancy. New features will always be assigned to the correct category during the testing phase, independently of the predicted category result. During this execution flow, the clusters will be updated as needed. This includes updating their centroids and performing cluster splitting operations. See Fig. 2.

A feature will be forgotten if 1) it is redundant, 2) its global weight is less or equal to the memory decay threshold or 3) the number of features in the application's memory is bigger or equal to M^+, where M^+ represents the maximum number of features that the application's memory can hold. In this implementation we did not consider the condition described in 3). See Fig. 3. New features are assigned to their respective closest cluster.

D. Categories and Object Views Histograms

Let V be the set of the object views and C the set of the object categories. The mapping $V \rightarrow C$ is obtained by computing the Euclidean distance between the histograms of each $v \in V$ and each $c \in C$ and choosing $min(distance(histogram_v, histogram_c))$ for every v (i.e., a histogram is computed for every c and every v). A histogram h_i (of a view or category) is based on the local weights of the features of that view/category

[1] http://pointclouds.org/

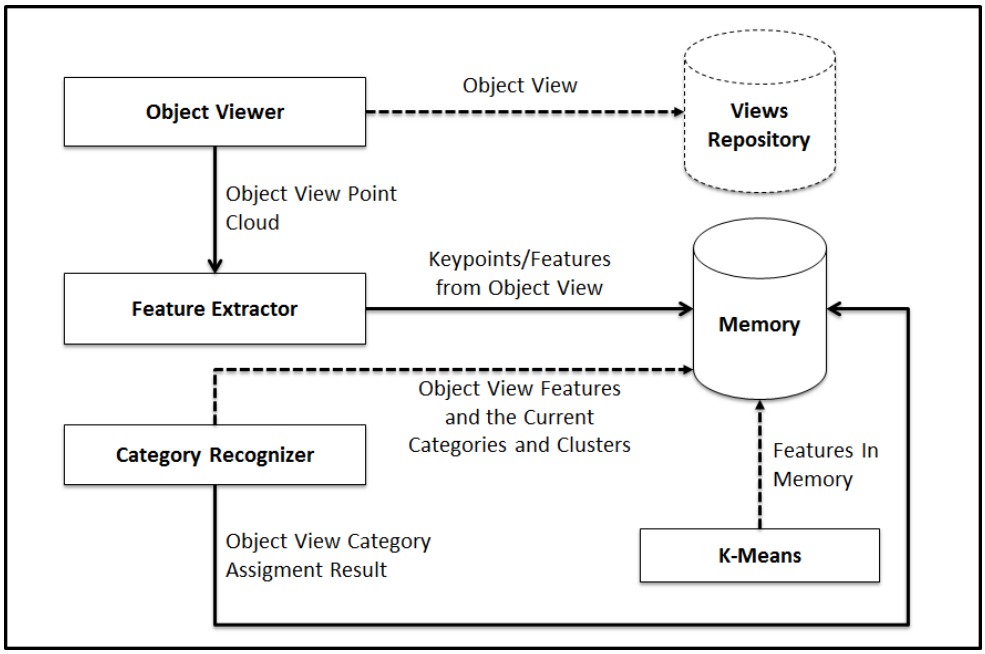

Fig. 1. Overview of the architecture of the developed application.

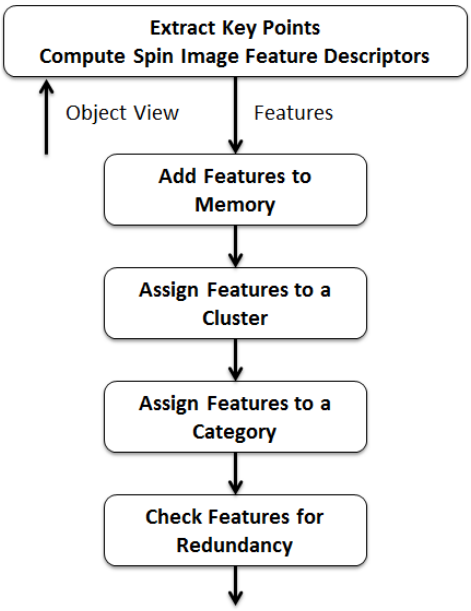

Fig. 2. Processing new seen features.

and $dim(h_i) = N$, N is the number of clusters, more specifically, h_i is a vector in R^N. To avoid problems related to dominance the histograms are normalized using $\sqrt{\sum_{i=1}^{N} lw_i^2}$, where lw_i is the sum of the local weights of all the features associated with the respective histogram that belong to cluster i, $i = 1...N$.

E. Learning the codebook

The codebook is learned online using agglomerative clustering alongside an adaptation of the K-means algorithm. The agglomerative clustering approach is used in order to derive the initial K clusters. More formally, given two features f_i and f_j with distance d_{ij}, f_i and f_j should be clustered together if $d_{ij} < \mu - \sigma$, where μ and σ represent respectively the mean and standard deviation of the pairwise distances between all the features seen so far.

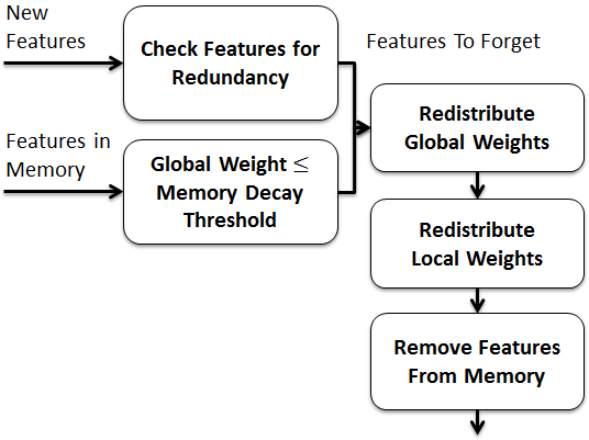

Fig. 3. Forgetting mechanism.

After computing the initial K clusters an adaptation of the K-means algorithm is used. This adaptation implements a splitting criterion that allows the creation of new clusters over time. More formally given cluster c_i, c_i should be split into two new clusters c_j and c_k if $radius(c_i) > meanDist(c_i) + 3.5 * stdDist(c_i)$, where $radius(c_i)$ represents the radius of c_i (i.e. the largest distance from a point in the cluster to the centroid) and $meanDist(c_i)$ and $stdDist(c_i)$ represent respectively the mean and the standard deviation of the distances of all the points in the cluster to the centroid. No mechanism was implemented in order to allow clusters to be merged.

F. Memory Management

Let C be the set of categories, K the set of clusters and F the set of features in the application's memory. Then, the application's memory main data structures are C, K and F.

Forgetting feature f_i follows the algorithm given below.

Algorithm 1

1. Redistribute Global Weights.
2. Redistribute Local Weights.
3. Update the cluster's centroid to which f_i is assigned to.
4. Remove f_i from memory.

1) Global Weight Redistribution

Let $f_i \in F$ be a feature to forget and θ a specified search radius. Then, global weight redistribution follows the algorithm presented below.

Algorithm 2

$S_i = Snapshot(F)$
$D = \{f \in S_i \mid |f, f_i| \leq \theta, f \neq f_i\}$
For $f \in D$
    $AssignNewGlobalWeight(f)$
    $UpdateCentroid(k_i)$, $k_i \in K$, $Assigned(f, k_i)$

Snapshot S_i is used for global weight redistribution for every feature being forgotten at a specific point during the application's execution. *UpdateCentroid* uses (1) and *AssignNewGlobalWeight* uses (2), presented in Section III. The search radius, θ, computation rules are given by the rule presented below.

$$\theta = \begin{cases} \text{avg}(|f_i, k_m|, |f_i, k_j|) & , |\text{K}| > 1, \text{m} \neq \text{j} \\ |f_i, \text{k}| & , |\text{K}| = 1 \end{cases}$$

Where avg refers to the arithmetic mean and k_m and k_j are the two closest clusters to f_i. For performance reasons we use a fixed search radius, $\theta = 0.065$. This value has no theoretical support behind it. It is just a value we choose by looking at the distances computed by KdTreeFLANN.

2) Local Weight Redistribution

Let $f_i \in F$ be a feature to forget and $c \in C$ a category. Local weight redistribution follows the algorithm presented below.

Algorithm 3

$A = \{x \mid Assigned(x, c), Assigned(f_i, c), x \in F, x \neq f_i\}$
For $x \in A$
    $AssignNewLocalWeight(x)$

Set A can be obtained from a snapshot of F or from F. In the case of new features a snapshot of F will be used since all new features from an object view belong to the same category. *AssignNewLocalWeight* uses (3), presented in Section III.

3) Memory Decay Factor

The memory decay factor, say π, is applied to $F \setminus \{$features added for the first time$\}$ (i.e., is applied to the features' weights, local and global), such that: a weight ω will have the new value $\omega = \omega \times \pi$. We assume $\pi = 0.99985$. There is no solid ground theoretical support for the value of π. The memory decay factor is applied before adding new seen features to the application's memory.

4) Feature Redundancy

The feature redundancy check follows the algorithm presented below.

Algorithm 4

$A = Snapshot(F)$
$F^* = \{$new seen features$\}$
For f in F^*
    $B = \{f_j \mid f_i \in A, f_j \in MinK(|f, f_i|)\}, B \subsetneq A$
    For f_j in B
        $|f, f_j| \leq \varepsilon$
            $ForgetFeature(f)$

Where B has the k-nearest features to feature f, $MinK()$ finds such set B. Then if the distance between f and a feature in the set B is smaller than a threshold ε feature f is redundant and will be forgotten. Because we are using a snapshot a condition of mutual redundancy may occur (i.e., a feature is made redundant by another feature that has been forgotten but still has a reference in the snapshot). This condition is checked for up to k (i.e., if a mutual redundancy condition occurs the application can detect and avoid it up to k features, the number of features in B).

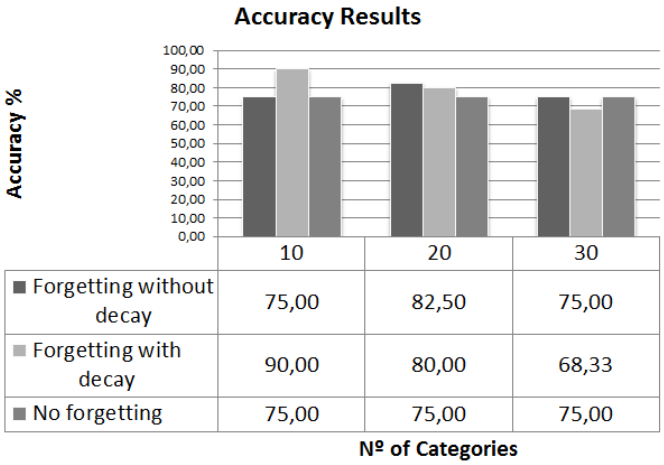

Fig. 4. Accuracy results obtained when using (not using) the forgetting mechanism with and without memory decay when seeing objects from 10, 20 and 30 different categories.

V. Experimental Results

This section discusses and presents the tests performed and the results obtained.

A. Data Set

The tests were performed using the Washington RGB-D Object Dataset [10], [11]. This dataset contains views of 300 physically distinct everyday objects, organized into 51 categories. Each view is a 3D point cloud in the PCD format. Each point in this cloud is stored with 6 fields: its 3D coordinates (x, y, z), its color, packed into 24 bits with 8 bits per channel (RGB) and its x and y coordinates on the image.

B. Tests

The tests were performed using the following two phase procedure. During the training phase the codebook was incrementally constructed using the adaptation to the K-means algorithm already mentioned. The object categories were learned concurrently. During the testing phase the system used the constructed codebook to identify new yet unseen objects. In order to simulate an open-ended scenario, the codebook

continued to be updated during this testing phase. The categories and the object views selected to perform each test were shuffled and separated into a training set (65%) and a testing set (35%) in order to perform the test. An equal number of object views was used for each category in order to prevent any bias related to the characteristics of the different categories (e.g. using more objects of a category that due to its characteristics can be better identified by the system).

Fig. 4 presents the accuracy results obtained when using (not using) the forgetting mechanism with and without memory decay when seeing objects from 10, 20 and 30 different categories and Fig. 5 presents the codebook size.

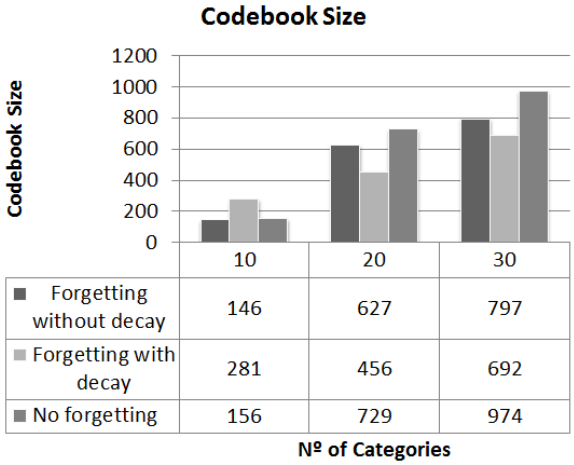

Fig. 5. Codebook size obtained when using (not using) the forgetting mechanism with and without memory decay when seeing object from 10, 20 and 30 different categories.

C. Discussion

As depicted in Fig. 4 the preliminary tests seem to indicate similar (and in some cases better) accuracy results obtained when using the 'forgetting' mechanism when compared with the usage of the complete set of observed features. In the case of the use of the 'forgetting' mechanism some variability in the results seem to occur. This may be due to the fact that this mechanism may introduce higher volatility in the clustering (the codebook). Although these results also seem to indicate that the memory decay mechanism plays a negligible penalty effect in the results, more tests must be performed using several different levels of varying decay in order to fully assess the real impact of this mechanism.

With regard to the codebook size, depicted in Fig. 5 it can be seen that the size of the codebook increases with the number of categories tested. This growing effect may be due to the fact that no merging operation was implemented, which in turn means that the number of cluster can only increase.

VI. Conclusions and Future Work

This work proposed an incremental online learning methodology comprising a 'forgetting' mechanism, targeting open-ended learning domains. The methodology used proposes the concurrent learning of the codebooks and the object categories using supervised and unsupervised learning techniques. The construction of the codebook as well as its continued updating during the testing phase is achieved via an adaptation of the K-means algorithm. Weights are also proposed in order to encode the 'forgetting' mechanism. Redundancy and memory decay are the main mechanisms by which forgetting is achieved. The system was then tested using the Washington RGB-D Object Dataset. The preliminary results seem to indicate that this 'forgetting' mechanism does not impact negatively the accuracy results obtained with regard to object recognition when compared to a non-forgetting approach, showing some improvement in some cases. The memory decay mechanism also seems to have a negligible impact on these results although further testing should be performed.

Future work should focus on further assessing the impact of the 'forgetting' mechanism with respect to accuracy and computational and memory cost. The extent to which the system is able to recall past seen objects whose features have been partially forgotten should also be assessed. Lastly, the incremental online learning methodology proposed should be thoroughly tested. Special emphasis should also be given to the introduction of an error correction mechanism in order to prevent the accumulation of errors in the weights due to wrong classifications obtained during the testing phase. Future tests should include natural images.

References

[1] M. Oliveira, L. Seabra Lopes, G. H. Lim, S. H. Kasaei, A. D. Sappa, and A. M. Tome, "Concurrent learning of visual codebooks and object categories in open-ended domains," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 2488–2495.

[2] F. Jurie and B. Triggs, "Creating efficient codebooks for visual recognition," in *Tenth IEEE International Conference on Computer Vision, 2005. ICCV 2005.*, 2005, vol. I, pp. 604–610.

[3] F. Perronnin, "Universal and adapted vocabularies for generic visual categorization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 7, pp. 1243–1256, 2008.

[4] C. Zhang, J. Liu, Y. Ouyang, Q. Tian, H. Lu, and S. Ma, "Category sensitive codebook construction for object category recognition," in *16th IEEE International Conference on Image Processing (ICIP)*, 2009, pp. 329–332.

[5] J. C. Van Gemert, C. J. Veenman, A. W. M. Smeulders, and J. M. Geusebroek, "Visual word ambiguity," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 7, pp. 1271–1283, 2010.

[6] T. Nicosevici and R. Garcia, "Automatic visual bag-of-words for online navigation and mapping," *IEEE Trans. Robot.*, vol. 28, no. 4, pp. 886–898, 2012.

[7] K. Skretting and K. Engan, "Recursive least squares dictionary learning algorithm," *IEEE Trans. Signal Process.*, vol. 58, no. 4, pp. 2121–2130, 2010.

[8] A. E. Johnson, "Spin-images: A representation for 3-D surface matching," 1997.

[9] A. E. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3D scenes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 5, pp. 433–449, 1999.

[10] "RGB-D Object Dataset." [Online]. Available: http://rgbd-dataset.cs.washington.edu/index.html. [Accessed: 21-Jan-2018].

[11] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view RGB-D object dataset," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 1817–1824.