Coresets for Visual Summarization with Applications to Loop Closure

Mikhail Volkov[1], Guy Rosman[1], Dan Feldman[1], John W. Fisher III[1], and Daniela Rus[1]

***Abstract*— In continuously operating robotic systems, efficient representation of the previously seen camera feed is crucial. Using a highly efficient compression coreset method, we formulate a new method for hierarchical retrieval of frames from large video streams collected online by a moving robot. We demonstrate how to utilize the resulting structure for efficient loop-closure by a novel sampling approach that is adaptive to the structure of the video. The same structure also allows us to create a highly-effective search tool for large-scale videos, which we demonstrate in this paper. We show the efficiency of proposed approaches for retrieval and loop closure on standard datasets, and on a large-scale video from a mobile camera.**

I. Introduction

Life-long video streams are a crucial information source for robotics. Many uses of visual sensors involve retrieval tasks on the collected video with various contexts. Robots operating in a persistent manner are faced with several difficulties due to the variety of tasks and scenes, and the complexity of the data arriving from the sensors. While for some tasks we can define simplified world representations that form approximate sufficient statistics, real-world scenarios challange these approximations. For example, one important task is loop-closure detection for robotic navigation and localization. Under the assumption of a static environment, a location-based loop-closure can be defined in terms of locations and their visual appearance model. Clearly such a model would not be sufficient for a user operator trying to find a specific observed event in the robot's history.

Scene assumptions are also critical — breaking the assumption of a static environment would turn the relatively easily defined location-based mapping problem into the full life-long localization and reconstruction problems that are still open lines of research. On the other hand, keeping the full visual history of the robot in random access memory is also suboptimal and often is impossible. A more condensed form, summarizing the video and allowing various tasks to be performed is an important tool for persistent operation.

In this paper we demostrate how a method for high-dimensional stream compression based on the *k-segment mean coreset* [22] can be leveraged into efficient summarization, loop-closure detection and retrieval for arbitrary large videos. Informally, coresets are approximation algorithms for specific problems that offer linear construction complexity, and sublinear memory requirements. Running an algorithm for a specific type of coreset (such as segmentation or clustering) approximates the cost obtained by that algorithm on the original data, with a provably small error. More rigorously, coresets are described in Section II, where we also refer to extensive literature. Our proposed method trades off video complexity and length as as well accuracy and robustness requirements. We define a data-adaptive structure to retrieve and access keyframes from the video, and demonstrate how this structure can be used to probe for loop-closures over a video history of arbitrarily large size, while providing guarantees for the retrieval given enough computation time. The collection of the data requires a linear-size memory, and allows logarithmic-time retrieval, providing, in some sense, an approximate sufficient statistic for a large variety of tasks involving the visual history. The tasks we demonstrate in this paper can be considered representatives of the full range of possible tasks for a robotic system involving its visual history.

Contribution We develop an efficient feature-based coreset algorithm for summarizing video data with significantly lower memory requirements than existing methods. Additionally, we demonstrate a system implementation of the described algorithm that generates a visual summarization tree. We present a variety of experimental results that characterizes the efficiency and utility of the resulting system for place identification and loop closure.

We define and describe the relevant coresets, structures, and algorithms in Section II. In Section III we define the require notations for localization. This is followed by the details of the proposed closure detection and retrieval algorithms in Section IV. In Section V we demonstrate empirical results of our algorithm, both on existing and new datasets, followed by conclusions in Section VI.

A. Related Work

Large-scale place recognition in online robotic systems relates to several active fields of research. Studied intensely between the vision [26], [25], [15], [24] and the robotics [11], [4], [16] communities, attempts have been made to allow faster loop-closure processing of larger datasets [5] and to utilize 3D information in order to increase specificity [21]. Maddern et al. [18] propose a way of minimizing the effects of life-long collection of images in the mapping phase, especially with respect to location model creation.

We note that a key assumption of such algorithms is that loop-closure and location pruning are achieved at full video frame rate, thereby inducing both high computational costs and the need for significant retrieval efficiencies. When

*Support for this research has been provided by Hon Hai/Foxconn Technology Group, MIT Lincoln Laboratory, SMART Future Urban Mobility project, and ONR MURI grant N00014-09-1-1051. We are grateful for this support.

[1]Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, MA 02139, United States {mikhail,rosman,dannyf,fisher,rus} at csail.mit.edu

3D information is unreliable (e.g. indoor localization), when there is significant location uncertainty, or when loop-closure information is scarce, the number of locations may grow with the size of the video stream. In this sense, dealing with the inherent complexity of life-long videos in the area of localization is still quite limited.

Several works in the robotics community (see for example [4],[13],[1]) attempt to define the problem as large scale inference problem over a graphical model of observations conditioned on locations. One advantage of such methods is their natural integration with local filtering approaches for localization [7] and handling of outlier matches.

In the vision community, localization is highly related to place recognition works such as [15]. However in large scale location recognition, 2D-to-3D matching approaches (such as [17],[23]) attempt to address the retrieval problem associated with collections of large images and multiple locations. The main emphasis in these approaches is on obtaining high specificity and reducing false alarm rate – this is partially due to the fact that such systems are inherently looking for maximum probability solutions and hence attempt to prune multiple alternatives to the correct location. Several of these works obtain high efficiency by 3D-aided pruning, since they utilize the reconstructed map as well.

The use of coresets of data approximation has been considered previously with the work of [20] and [22] being the most closely related. While [20] considers the use of coresets for appearance-based mapping, navigation, and localization they do not exploit temporal consistency across frames as we do here. Additionally, the computational complexity and associated memory requirements of the proposed approach represent an exponential improvement when compared to that work. Lastly, the proposed coreset formulation allows for an unbounded set of locations while supporting location retrieval and loop-closure. While in [22] we adopt a descriptor vector representation of frames and demonstrate segmentation using a derived coreset, we do not consider localization or loop-closure. Furthermore, in contrast to the current formulation, the number of segments is assumed to be known *a priori*.

II. Coresets And Stream Compression

We now turn to describe the coreset used in this paper and the specific properties that make it useful for video retrieval and summarization. In the problem statement we query an observed image from a static set of observed locations. However, in practice we are given a possible unbounded video stream of multiple frames per second (50/60 is the HD standard). To this end, we select a representative over-segmentation of the video stream (along with a compact representation of each segment) called a *coreset*. The coreset approximates the original data in a provable way, that guarantees a good trade-off between the size of the coreset and the approximation of each scene in the video stream. More precisely, we embed images into $\mathbb{R}^d$ based on a naive Bayes approximation, representing n images as a set n points in $\mathbb{R}^d$. The algorithm outputs a set, called ε-coreset, of roughly k/ε weighted segments approximating the data, such that the sum of squared distances over the original points and approximating segments to every k piecewise linear function is the same, up to a factor of $1 \pm \varepsilon$. The existence and construction of coresets has been investigated for a number of problems in computational geometry and machine learning in many recent papers (cf. surveys in [8]). The assumption in this model is that similar images corresponds to points on approximately the same time segment.

In this work we leverage our most recent results for high-dimensional data segmentation [22] and present algorithms to perform efficient loop-closure detection and retrieval for arbitrary large videos. The core of our system is the k-segment coreset, which provides flexibility with respect to varying dimensionalities, multiple sensors, and different cost functions. For a single segment $(k = 1)$ no segmentation is necessary, and the data is represented using SVD. For multiple segments we want to know how many segments are in the partition, how to divide them, and how to approximate each segment. We present a two-part coreset construction: first we estimate the complexity of the data using a *bicriteria* algorithm; second we define a fine partition of the data into coreset segments using a *balanced partition* algorithm, and we approximate each segment by SVD. The algorithm guarantees a segmentation that is close to k-segments with a cost that is close to the optimal cost.

A. Streaming and Parallelization

One major advantage of coresets is that they can be constructed in parallel, as well as in a streaming setting where data points arrive one by one. This is important in scenarios where it is impossible to keep the entire data in random access memory. The key insight is that coresets satisfy certain composition properties, which have first been used by [10] for streaming and parallel construction of coresets for k-median and k-means clustering. These properties are:

1) The union of two ε-coresets is an ε-coreset.
2) An ε-coreset of such a union is an $\varepsilon(1+\varepsilon)$-coreset.

We note that while the first property may seem trivial by concatenating the elements of the coresets, the second property relates to the desired compactness of the representation, and states that we can further compactness the unified coreset so that it scales nicely as more data is added.

Streaming In the streaming setting, we assume that points arrive one-by-one, but we do not have enough memory to remember the entire data set. Thus, we wish to maintain a coreset over time, while keeping only a small subset of $O(\log n)$ coresets in memory. Since each coreset is small, the overall memory consumption is also small. Using the properties above, we can construct a coreset for every block of consecutive points arriving in a stream. When we have two coresets in memory, we can merge them (by property 1), and re-compress them (by property 2) to avoid increase in the coreset size. An important subtlety arises: while merging two coresets does not increase the approximation error, compressing a coreset does increase the error. However, using a binary tree as shown in Fig. 1, the final approximation

in the root is roughly $O(\varepsilon \log n)$ for an original stream of n points, which can be reduced to ε by using a little smaller value for ε (cf. [22] for a discussion of this point in the context of k-segmentation).

We call the tree resulting from the coreset merges the *coreset streaming tree*. We denote the coresets created directly from data as *streaming leaves*. An additional, useful structure can be defined by looking at segment merges in the balanced partition algorithm in [22]. This structure is the *coreset segment tree*.

Parallelization Using the same ideas from the streaming model, a (nonparallel) coreset construction can be transformed into a parallel one. We partition the data into sets, and compute coresets for each set, independently, on different computers in a cluster. We then merge two coresets (by property 1), and compute a single coreset for every pair of such coresets (by property 2). Continuing in this manner yields a process that takes $O(\log n)$ iterations of parallel computation. This computation is also naturally suited for map-reduce [6] style computations, where the map tasks compute coresets for disjoint parts of the data, and the reduce tasks perform the merge-and-compress operations. We note that unlike coresets for clustering such as the one used in [21], parallel computation requires us to keep track of the time associated with the datapoints sent to each processor.

New Approaches We now discuss how our work differs from, and builds on, existing state of the art in this respect. Using the above techniques, existing coreset construction algorithms allow us to handle streaming and parallel data but not both. This is because the parallel approach assumes that we can partition the data in advance and split it between the machines. However, we cannot split an unbounded stream in such a way when not all the data is available in advance. In our problem we wish to process streaming video on the cloud, i.e., computing it for streaming data *and* in parallel. In other words, we want to compress the data in a distributive manner while it is uploaded to the cloud.

There are two contexts in which our system allows simultaneous streaming and parallelization. The first is streaming of buffered data. For example (a robotic scenario), in the case of an autonomous exploration vehicle or UAV collecting a high volume of video, it is still useful and sometimes necessary to stream the data to a collection point at a later time. Another example is a wearable device with an intermittent connection to a data server that will buffer data at times when it is unable to upload it in real time. In both cases, the context will dictate a sufficient leaf size beyond which temporal continuity is not expected, and continuous data blocks of this size can be streamed in parallel.

Another context that is prevalent in robotics is a multi-source video stream (such as the FAB-MAP dataset [4], which is considered an industry standard). In this case, we can parallelize the coreset construction by streaming each view separately, multiplexing video streams where necessary. Temporal continuity is naturally preserved.

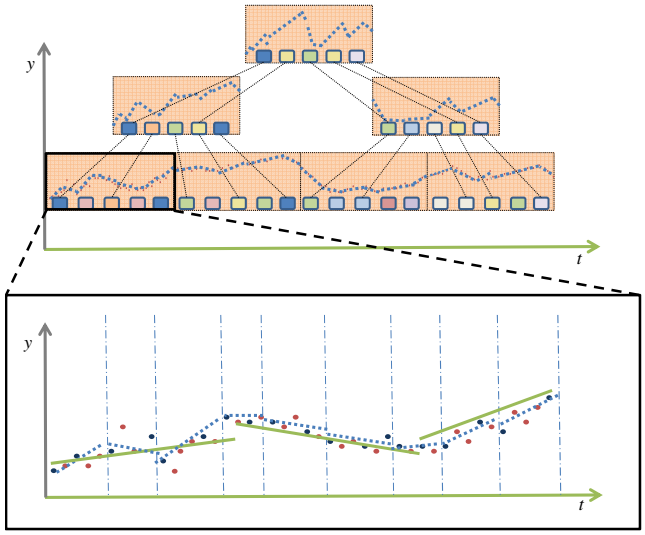


Fig. 1: Streaming coreset construction of a data stream. The bottom figure illustrates the online construction and segmentation of a block from an incoming data stream. Coreset segments are shown with dashed blue lines. The top figure illustrates the continuous compression of the data stream through progressive merge/reduce computation of the coresets from lower level coresets.

B. Summarization and Retrieval

Roughly, in video summarization the goal is to get an image representative from each scene in a given time interval, for a partition that captures the structure of the video. Using the right feature space, we assume that images in the same scene are generated by a simple model. We use a linear approximation to allow short-scale temporal variations. Our goal is to select a representative image from each such segment. However, the coreset for this problem only guarantees that the k-segment of the coreset will approximate the k-segment of the original data, which may not be a fine enough segmentation. In addition, the assumption above holds usually but not always – thein practice images that are intuitively similar appear on different segments, due to appearance of new objects or viewing of different scene areas with a small FOV camera. In this paper we demonstrate how the coresets streaming framework allows us to overcome these difficulties with little effort.

In is paper we differentiate between several hierarchical structures that are created during the stream processing. The traditional two are the coreset streaming tree defined by the merging of batches in the data stream, and the coreset segment tree, which depicts the merging of coreset segments during the streaming. The former has a fixed topology regardless of the data, while the latter is adaptive to transitions in the data, yet does not prioritize different aspects of the data points themselves. We add on top of these a third structure, which allows us to get this unary adaptivity.

Specifically, we add a layer that stores a carefully chosen set of images along with each node of the coresets tree, called *key frames*. Besides accomodating possible partitions of the data, these key frames are expected to capture the variability

of the observed scenes in the video, and provide for various applicative needs. We detail the selection of the key frames in Section IV-A. These key frames are merged both according to the images quality and their ability to represent other images in the video, and according their representation of video transitions, as captured by segment merges in the coreset algorithm. We call the tree formed by selection of prominent key frames the *keyframe merge tree*.

In the context of life-long processing and retrieval, the coreset streaming tree is usually used to support the streaming and parallel model as explained in Section II-A, where only a small subset of the coresets in the tree exists at any given moment.

III. Loop-Closure Problem Formulation

We now describe the loop-closure problem following the notation of [4]. Denoting the set of observations at time k, usually extracted from an observed image by Z_k, we wish to associate it with a unique location L_i. This is done by evaluation of the set under a location-dependent probabilistic model

$$p(Z_k \mid L_i) = p(z_1, \ldots, z_{|v|} \mid L_i), \tag{1}$$

where $|v|$ denotes the number of features, and z_i denotes the indicator for appearance of feature i in the image k. While some methods consider the variables z_i as binary variables, in many cases, counts of feature appearance may be multiple (especially if the visual vocabulary used is small).

We are looking for the location that maximizes the conditional probability

$$p(L_i \mid Z_k) = \frac{p(Z_k \mid L_i)p(L_i)}{p(Z_k)}, \tag{2}$$

and for the purpose of this work, we ignore the temporal dependency that is often sought [4]

$$p(L_i \mid Z_k, Z^{k-1}) = \frac{p(Z_k \mid L_i, Z^{k-1})p(L_i, Z^{k-1})}{p(Z_k, Z^{k-1})}, \tag{3}$$

where Z^{k-1} denotes the observation history up to time k.

Several approximation have been considered for computing $p(Z_k \mid L_i)$. The simplest approximation is the naive Bayes approximation

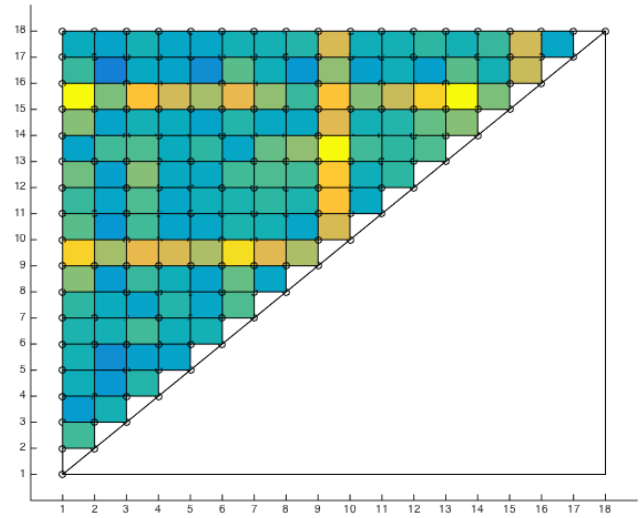
$$p(Z_k \mid L_i) = \prod_{i=1}^{|v|} p(z_i \mid L_i), \tag{4}$$

which leads to L_2 distance measure between observations and location distributions, while assuming

$$p(z_j \mid L_i) \propto exp\left\{-\left(z_j - \mu_j(L_i)\right)^2/\sigma^2\right\}, \tag{5}$$

where we do not assume a binary appearance vector. We note that the log-probability of the observation given a location naive Bayes model is the L_2 distance in feature-space. This distance is the distortion measure approximated by k-segment mean coreset for k-segment models.

Another often used approximation is the Chow-Liu tree [2]. As proposed in [4], the naive Bayes PDF model can be improved upon by an optimal tree (in the KL-divergence sense) with respect to the empirical distribution of the location in a way that is still tractable (i.e by solving a max-weight spanning tree problem).



(a) Dissimilarity matrix

(b) Relevance score

Fig. 2: Examples showing the dissimilarity matrix d and relevance score f^* associated with the two terms of (9). The dissimilarity matrix (Fig. 2a) shows visually how different the candidate frames are in descriptor space. The relevance score (Fig. 2b) given by (7) is shown in black. The 9 out of 18 selected keyframes during a merge of two nodes are shown with black dots.

IV. Retrieval Algorithms

We now detail the construction required for the summarization and retrieval tasks described above, in terms of the augmentation of the coreset structures, and the algorithms running on top of the coresets.

A. Incorporating Keyframes into a Coresets Tree

We first describe the incorporation of keyframes into the coreset streaming tree construction. This allows us to demonstrate the use of keyframes stored in the coreset tree in localization, keeping only a fraction of the frames for localization, At each streaming leaf SL we keep a set of K keyframes, for some fixed K. With each node merge in the tree, we select the new set of keyframes from the ones existing in the children nodes by running a modified *farthest-point-sampling* (FPS) algorithm [12], [9] on the feature vectors. The FPS chooses the frame with feature vector x from the data, that is farthest from the existing set S_{j-1}

$$x_j = \underset{x}{\operatorname{argmax}}\, d(x, S_{j-1}), \tag{6}$$

with S_{j-1} marking the set of previously chosen frames, in terms of their feature vectors, and $d(x_i,x_j)$ is the L_2 distance between keyframes x_i,x_j in the feature space. Here, we modify the FPS selection rule by adding an image relevance score term that can include image quality (sharpness and saliency), temporal information (time span and number of represented segments), and other quality and importance measures. The relevance score is defined as

$$f^*(x) = \alpha_T f_T(x) + \alpha_S f_S(x) + \alpha_B f_B(x) \tag{7}$$

where positive $f_i(x_j)$ indicates higher relevance. The relevance score $f_B(x_j)$ is the the blur measure for image j based on [3] (negated for consistency with our positive relevance convention). The relevance scores $f_T(x_j), f_S(x_j)$ denote video time and number of coreset segments associated with the keyframe x_j, respectively. More generally

$$f^*(x_j) = \textstyle\sum_{i=1}^{N} \alpha_i \; f_i(x_j) \tag{8}$$

for any set of metrics $1 \ldots N$, such as the example in Fig. 2b. The weights α allow us to fine-tune the system, for example to give more weight to image quality vs temporal span. This allows us to get a rich set of representative keyframes that are meaningful in the video in terms of time span and complexity, Given a starting point x_0 we modify the FPS algorithm to include the relevance score. The new point is then given by

$$\begin{aligned} x_j &= \operatorname*{argmax}_x \left\{ \hat{d}(x, S_{j-1}) \right\} \\ &= \operatorname*{argmax}_x \left\{ d(x, S_{j-1}) + f^*(x) \right\}. \end{aligned} \tag{9}$$

It can be shown that the resulting selected set is close to the optimal set if the values of the score function are bounded and sufficiently close to 0, converging in the limit to the 2-optimality guarantee of FPS. Let S be the set chosen by the modified FPS, and let

$$\rho(S) = \max_x \hat{d}(x,S). \tag{10}$$

Lemma 1 *Let S^* be an optimal representative set given by $S^* = argmin_{S'} \rho(S')$. Then*

$$\rho(S) \leq 2\rho(S^*) - \min_x f^*(x) \tag{11}$$

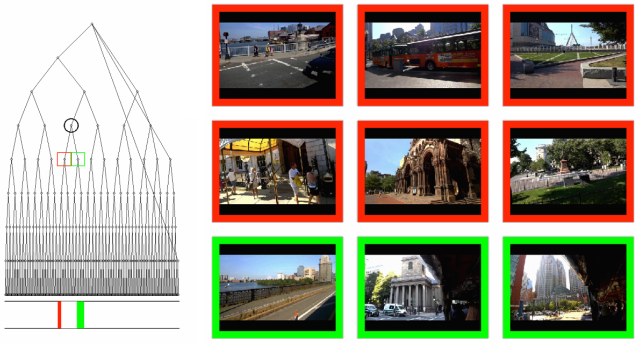
Proof: The proof for a specific k and S_k is done similar to the FPS proof [19], by looking at x_{max}, the maximizer of $\hat{d}(x,S_k)$, along with S_k. By comparing this set to S_k^* using

$$\hat{d}(x,y), \;\; x \in \{x_{max}\} \cup S_k, \;\; y \in S_k^* \tag{12}$$

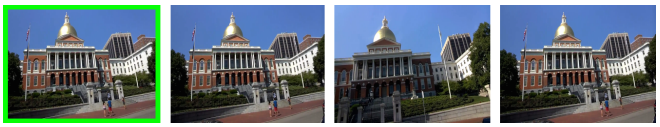
we have two elements x_1,x_2 with $\hat{d}$ minimal to the same element $s^* \in S_k^*$, by the pigeonhole principle. Let us assume x_1 to be the latest of the two. It can be shown that $\hat{d}(x_{max},S_k) \leq \hat{d}(x_1,x_2)$, due to the order of selection of x_{max},x_1,x_2,

$$\begin{aligned} \hat{d}(x_1,x_2) &\leq \hat{d}(x_1,s^*) + \hat{d}(x_2,s^*) - f(x_2) \\ &\leq 2\rho(S_k^*) - \min_x(f^*(x)), \end{aligned} \tag{13}$$

by triangle inequality over d and the definition of $\hat{d}$. ■



(a) The interactive UI described in Section IV-B showing the coreset tree for the Boston Trolley Tour (left). The image collage shows 9 keyframes captured from this data (right). Observe that the keyframes with the red/green margins propagated from the left/right child nodes; the corresponding represented time interval is shown at the bottom of the tree.



(b) An image retrieval example from the Boston dataset, using the tree-sampling method in Algorithms 1,2. The green frame marks the query image corresponding to x_{ref} in Algorithm 2. The other images are the maximum-probability match found by sampling.

Fig. 3: Boston tour loop-closure results

The function f^* makes the coreset tree more adapted to the data. By storing this relevance score for each node in the coreset tree, the relative importance of individual keyframes is propagated along the binary tree structure, allowing us to query the tree for this information at each level. In general, our coreset tree system facilitates the addition of other metrics, such as object detection, optic flow, etc. that allow us to emphasize the semantic content that is most appropriate to the problem.

B. User-Interface for Retrieval

The proposed key-frames allow easy and useful search and retrieval for a human user. We now describe user interface that allows the user to browse through the video stream and see the summarization and representative images from each interval using a visual tree, as shown for example in Fig. 3a. We note that our user interface, summarization and time search approach can be useful for any other type of coresets on images, such as k-means clustering, or for trajectories summarization. The interface demonstrate a non-standard retrieval task and could be relevant for robotic tasks, such as searching for people and objects in a subsequence of the video, change detections, and so forth.

In the proposed UI, the user can browse through the binary tree on the left. The node with the white circle corresponds to the selected time interval. Its left child is always marked by a red circle, and its right child by a green one. The red and green rectangulars on the bottom of Fig. 3a mark the relevant leaves of the red and green nodes respectively. In

Algorithm 1 $v = \text{SAMPLEOLDNODE}(v_{end})$ – Sample node v from the tree recorded before v_{end}

$v \leftarrow v_{end}$
$t_{max} \leftarrow \infty$
$started\_descent \leftarrow 0$
$\alpha < 1$ is a fixed parameter
while $started\_descent \neq 1$ **do**
  sample p uniformly at random from $[0,1]$
  if v is not the root node **and** $p < \alpha$ **then**
    $t_{max} \leftarrow t_{start}(v)$
    $v \leftarrow \text{PARENT}(v)$
  else
    $started\_descent \leftarrow 1$
  end if
end while
while v is not a leaf node **do**
  $v \leftarrow \text{OLDCHILD}(v,\ t_{max},\ \text{KEYFRAMES}(v))$
  $started\_descent \leftarrow 1$
end while
return v

Algorithm 2 $X = \text{UPDATECLOSURECACHE}(x_{ref})$ – Find matching candidates X for image descriptor x_{ref}

$WM = \emptyset$
while 1 **do**
  $v \leftarrow \text{SAMPLEOLDNODE}(v_{end})$
  if $v \notin WM$ **and** $\text{FULL}(WM)$ **then**
    select node v_{rem} from WM at random
    $WM \leftarrow WM \setminus \{v_{rem}\}$
  end if
  $WM \leftarrow WM \cup \{v\}$.
  compute loop-closure probabilities for x_{ref} using WM
  compute matching candidates X
end while
return X

the right side of the figure we show the selected key frames in the white nodes. The key frames that were chosen from the red note are marked by a red border, and similarly the other key frames are marked by a green border.

Additionally, the user can specify a time span that he or she is interested in summarizing. The user interface computes the minimum subtree that summarizes the activity in that time region. The relevant images are extracted from the hard drive according to the user clicks. Theoretically, the retrieval time of the relevant coreset is poly-logarithmic in the size of the related video stream using the properties and structures of the coreset tree.

C. Life-long Loop-Closure

We now proceed to describe how life-long loop-closure can be performed based on the coreset streaming segment tree. While the segments in the coreset streaming tree provide an adaptive tradeoff between temporal resolution and efficiency, different nodes in the coreset streaming tree are still of equal time span. However, we expect the data-adaptivity of the coreset to allow efficient retrieval for arbitrarily long videos. To this end, we define a method for random caching of frames for loop-closure detection, based on the graph distance in the streaming segment tree. Similar to RTAB-MAP [14], we assume the system to include a *working memory* (*WM* in the notation of [14]) and an index based on the coreset streaming tree in memory. The coreset streaming tree nodes point to a database of frames (or their corresponding locations, in the case of a location-based retrieval), we denote as *long-term memory* (or LTM), where each leaf is a page of frames to be swapped in and out. We define retrieval and discard rules for pages containing previous frames, between a working memory and the database of the robot's visual history. We note that a similar approach could be incorporated with location-based mapping by replacing the frames descriptors with pointers to locations observation models.

The retrieval rule we employ randomly selects pages based according to the procedure SAMPLEOLDNODE presented in Algorithm 1. The procedure uses the function $\text{OLDCHILD}(v,t,keyframes)$, that returns a child of v recorded at t_{end} older than t at random. We adapt the sampling so that each child has a weight proportional to the number of keyframes the parent node drew from it, plus 1 (to ensure a non-zero sampling probability). This allows us to take into account quality metrics based on the coreset as well as image quality/saliency, as described in Subsection IV-A. The effect of the weighted traversal can be seen in Fig. 4b.

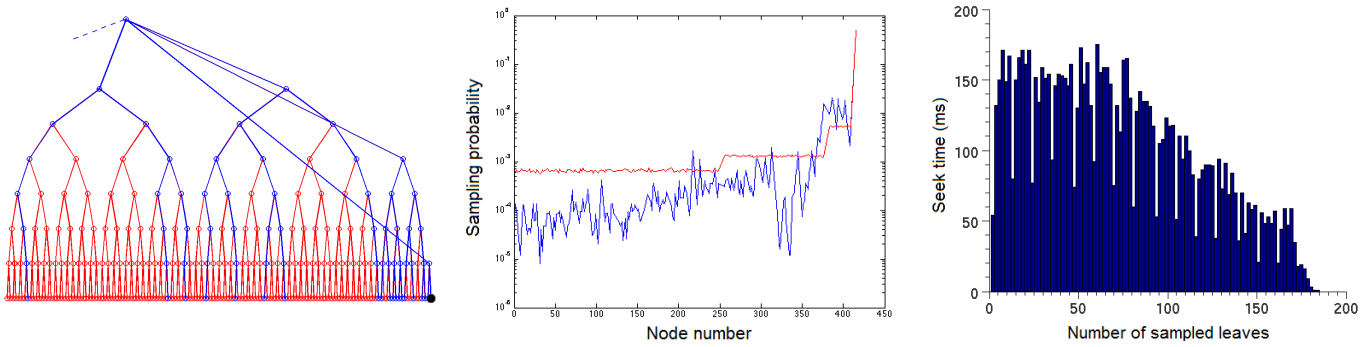
It can be seen that the probability of reaching leaf ℓ, by traversing from the last leaf in the tree v_{end}, is non-zero. Since reaching each leaf from v_{end} has exactly one path, and this corresponds to a single set of choices in the conditionals we can bound this probability from below by looking at the direct route from v_{end} to the node ℓ,

$$\begin{aligned} & p\left(\text{SAMPLEOLDNODE}(v_{end}) = \ell\right) \\ & \geq \alpha^{d_U(v_{end},\ell)} \left(\frac{1}{d_{max}}\right)^{d_D(v_{end},\ell)} \\ & \geq \left(\min\left(\alpha, \frac{1}{d_{max}}\right)\right)^{d(v_{end},\ell)}, \end{aligned} \tag{14}$$

where $d_D(v_{end},\ell)$ denotes the path length from the common root of v_{end} and ℓ to ℓ, and $d_U(v_{end},\ell)$ denotes the path length from the common root to v_{end}. It is easy to see that

$$p(\text{SAMPLEOLDNODE}(v_{end})) = \ell) \leq \alpha^{d_U(v_{end},\ell)}. \tag{15}$$

Let us mark by t_{start} and t_{end} the beginning and end of the time span associated with each node or leaf of the tree. It is furthermore easy to see that going up to the parent node, t_{start} is non-increasing and t_{end} is non-decreasing. Going down to an earlier child, t_{start} is non-increasing. This can be summed up in the following lemma:



(a) Sampling paths along the coreset tree (b) Comparison of tree-based sampling (c) Seek time histogram

Fig. 4: Loop-closure sampling results. Fig. 4a shows the sampling paths (blue) starting from the query node v_{end} (black) along the coreset streaming tree of the Boston data in Fig. 3a. Fig. 4b shows the comparison of leaf sample frequency without usage of keyframes computed on the data in Fig. 3a for $500k$ trials (red) against sampled nodes using keyframes (blue). Intuitively, high-probability nodes correspond to segments in the video that had salient and clear sections in the video. Fig. 4c shows the seek time histogram for the example shown in Fig. 3b, using the tree-sampling method in Algorithms 1,2. As expected, the tree-sampling seek times drop as the number of sampled leaves increases.

Lemma 2 SAMPLEOLDNODE(v) *samples a leaf whose span ends before* $t_{end}(v)$*, and all previous leaves have a non-zero probability of being sampled, as described in equation* (14)*.*

Based on this sampling procedure we present the loop-closure detection algorithm described in Algorithm 2, which operates according to a maximum time allotted per turn, in order to fit a real-time regime. We define FULL(WM) to be a function indicating whether the working memory is full. The sampling probability of leaves according to Algorithm 1 can be seen in Fig. 4a, showing the adaptiveness of the method.

It can be shown that the pages in WM are distributed exponentially decreasing with respect to the tree distance from start node v_{end}, assuming pages are kept in a FIFO order in the cache. The probability of a leaf to be added to WM is bounded from zero, thus ensuring that every leaf (and the locations pointed by it) has a chance of being sampled.

V. RESULTS

The primary dataset used in this study is a recording of a Boston Trolley Tour. The video was captured using Google Glass and spans 3 hours. Doing away with an obvious redundancy for these application [5], we conservatively subsample by a factor of 5, to around 75k frames. Fig. 3a (left) shows the coreset tree generated by processing the entire tour.

A preliminary set of experiments serve as a hard ground-truth demonstration of the correctness of the coreset tree for the purposes of video segmentation. For this demonstration we select 15 still frames capturing scenes of interest during the tour. The stills were duplicated for random periods of time into a synthetic video file. The purpose of these experiments is to demonstrate (a) the coreset tree's utility in successfully segmenting data, and (b) the capability of the coreset tree to adaptively propagate important keyframes to the higher nodes without repetition. Fig. 3a (right) shows the results of these experiments. We observe that the coreset tree has successfully segmented the data and captured all representative frames, and that each video still was captured by a keyframe, regardless of how long the original video segment was. This demonstrates the coreset's capacity to capture information. Secondly, we observe that as the keyframes propagate up to the node of the tree, the modified FPS algorithm described by equation (9) favors few repetitions of similar keyframes, by definition of the FPS algorithm. This highlights our coreset tree's capacity to summarize information in an adaptive manner that can be tailored to the problem domain.

A. Loop-Closure Experiments

Loop-closure experiments were conducted on a short video of 6000 frames with 2 known ground-truth loops. A coreset tree was created using a set of 5000 VQ representatives trained on 100k SURF descriptors. In general, the choice of leaf size depends on the problem domain, and will reflect the typical temporal resolution of the sequence. We used leaf sizes of $100, 150, 200$. With 9 keyframes per leaf, a leaf size of 200 represents a subsampling of the input data by a factor of more than 22. With larger leaf sizes, results were too sparse for this short video sequence, however for larger videos such as the Boston data, a leaf size that is an order of magnitude on par with the size of the test data is appropriate. An L_2 distance map was calculated based on the descriptors of the leaf-node keyframes, and thresholded to produce a loop-closure map.

The loop-closure map produced by our coreset was compared against an equivalent loop-closure using uniform sampling of frames from the test video. Keyframe descriptors were primed for a number of thresholds that give meaningful loop-closure patterns. Results were evaluated objectively by computing the precision/recall trends for our coreset tree against uniform sampling (Fig. 5). We see a typical trend of precision decreasing with recall, as the true positives get outweighed by false negatives with increasing threshold. For all values of recall, we achieve a higher precision by using

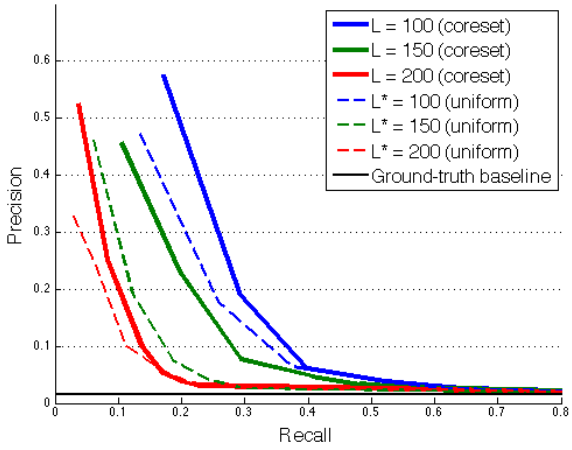


Fig. 5: Precision/recall plot comparing keyframes from the coreset tree (solid lines) against uniformly sampled keyframes from the video (dashed lines), as inputs for L_2 loop-closure. The ground-truth line represents the asymptotic precision score, i.e. $TP/(P+N)$ for a $P = 1, N = 0$ classifier.

descriptors from the keyframes of the coreset tree compared against uniform sampling. These results demonstrate the ability the coreset tree to capture information that is useful for state of the art loop-closure algorithms such as [5].

B. Retrieval Experiments

We now demonstrate a retrieval application based on the keyframes defined in Subsection IV-A. For a larger scale retrieval experiement, we demonstrate retrieval for a given query image in Fig. 3b. Given a query image, we show the results of 3 runs of the a search for a match (with some threshold on the L_2 distance) that includes the query image and similar results. We resample tree leaves until a match is found, and in each leaf retrieved, we look for the minimum L_2 distance match, starting with an empty *WM*.

We note that processing the overall video of 75k frames can be done at 3 frames per second on an *i*7 CPU. The histogram of leaf retrievals until match is shown in Fig. 4c. The seek access attempts average of 70.5 is significantly lower that the uniform expected number of attempts of 108.5 because we expect caching according to recent hits and more adaptive keyframe-based sampling to further improve results.

VI. Conclusions

In the paper we demonstrate how coresets for the *k*-segment means can form a basis for summarization of visual histories, for retrieval and localization tasks. We show how the coreset streaming tree can be used for visual loop-closure and image retrieval from a video sequence over large videos, essentially summarizing the video at real-time speeds. In future work we expect to extend the use of coresets into additional retrieval and understanding tasks, explore the use of additional coresets, and consider the more challenging case of indexing dynamic environments.

References

[1] R. Anati and K. Daniilidis. Constructing topological maps using markov random fields and loop-closure detection. In *NIPS*, pages 37–45, 2009.

[2] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Inf. Theor.*, 14(3):462–467, May 1968.

[3] F. Crete, T. Dolmiere, P. Ladret, and M. Nicolas. The blur effect: perception and estimation with a new no-reference perceptual blur metric. In *Proc. SPIE*, volume 6492, 2007.

[4] M. Cummins and P. Newman. FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *International Journal of Robotics Research*, 27(6):647–665, 2008.

[5] M. Cummins and P. Newman. Appearance-only slam at large scale with fab-map 2.0. *International Journal of Robotics Research*, 30(9):1100–1123, Aug. 2011.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[7] F. Dellaert and M. Kaess. Square root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1203, Dec. 2006.

[8] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *STOC*, 2010. Manuscript available at arXiv.org.

[9] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[10] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proc. 36th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 291–300, 2004.

[11] K. Ho and P. Newman. SLAM-Loop Closing with Visually Salient Features. In *ICRA*, Apr. 2005.

[12] D. Hochbaum and D. Shmoys. A best possible approximation for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[13] H. Johannsson, M. Kaess, M. Fallon, and J. Leonard. Temporally scalable visual SLAM using a reduced pose graph. In *ICRA*, Karlsruhe, Germany, May 2013.

[14] M. Labbe and F. Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Trans. Robotics and Automation*, 29(3):734–745, June 2013.

[15] A. Levin and R. Szeliski. Visual odometry and map correlation. In *CVPR*, volume I, pages 611–618, Washington, DC, June 2004. IEEE Computer Society.

[16] S. Li and S. Tsuji. Selecting distinctive scene features for landmarks. In *ICRA*, pages 53–59 vol.1, May 1992.

[17] Y. Li, D. J. Crandall, and D. P. Huttenlocher. Landmark classification in large-scale image collections. In *ICCV*, pages 1957–1964, 2009.

[18] W. P. Maddern, M. Milford, and G. Wyeth. Towards persistent indoor appearance-based localization, mapping and navigation using cat-graph. In *IROS*, pages 4224–4230, 2012.

[19] D. M. Mount. Design and analysis of computer algorithms. Lecture notes, University of Maryland, 2008.

[20] R. Paul, D. Feldman, D. Rus, and P. Newman. Visual precis generation using coresets. In *ICRA*, pages 1304–1311, May 2014.

[21] R. Paul and P. Newman. FAB-MAP 3D: Topological mapping with spatial and visual appearance. In *ICRA*, pages 2649–2656, Anchorage, Alaska, May 2010.

[22] G. Rosman, M. Volkov, D. Feldman, J. W. Fisher III, and D. Rus. Coresets for k-segmentation of streaming data. In *NIPS*, pages 559–567. 2014.

[23] T. Sattler, B. Leibe, and L. Kobbelt. Improving image-based localization by active correspondence search. In *ECCV*, pages 752–765, Berlin, Heidelberg, 2012. Springer-Verlag.

[24] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *CVPR*, 2007.

[25] J. Sivic and A. Zisserman. Video Google: Efficient visual search of videos. In J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, editors, *Toward Category-Level Object Recognition*, volume 4170 of *LNCS*, pages 127–144. Springer, 2006.

[26] J. Zheng, M. Barth, and S. Tsuji. Qualitative route scene description using autonomous landmark detection. In *ICCV*, pages 558–562, Dec 1990.