Distributed Image Search in Camera Sensor Networks

Tingxin Yan, Deepak Ganesan, R. Manmatha
Department of Computer Science
University of Massachusetts, Amherst, MA 01003
{yan, dganesan, manmatha}@cs.umass.edu

Abstract

Recent advances in sensor networks permit the use of a large number of relatively inexpensive distributed computational nodes with camera sensors linked in a network and possibly linked to one or more central servers. We argue that the full potential of such a distributed system can be realized if it is designed as a distributed search engine where images from different sensors can be captured, stored, searched and queried. However, unlike traditional image search engines that are focused on resource-rich situations, the resource limitations of camera sensor networks in terms of energy, bandwidth, computational power, and memory capacity present significant challenges. In this paper, we describe the design and implementation of a distributed search system over a camera sensor network where each node is a search engine that senses, stores and searches information. Our work involves innovation at many levels including local storage, local search, and distributed search, all of which are designed to be efficient under the resource constraints of sensor networks. We present an implementation of the search engine on a network of iMote2 sensor nodes equipped with low-power cameras and extended flash storage. We evaluate our system for a dataset comprising book images, and demonstrate more than two orders of magnitude reduction in the amount of data communicated and up to 5x reduction in overall energy consumption over alternate techniques.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; H.3.3 [**Information Search and Retrieval**]: Search process

General Terms

Design, Management, Performance

SenSys'08, November 5–7, 2008, Raleigh, North Carolina, USA.

Keywords

camera sensor networks, visterms, vocabulary tree, distributed search

1. INTRODUCTION

Wireless camera sensor networks — networks comprising low-power camera sensors — have received considerable attention over recent years, as a result of rapid advances in camera sensor technologies, embedded platforms, and low-power wireless radios. The ability to easily deploy cheap, battery-powered cameras is valuable for a variety of applications including habitat monitoring [18], surveillance [8], security systems [10], monitoring old age homes [2], etc. In addition to camera sensor networks, the availability of cameras on almost all cellphones available today presents tremendous opportunities for "urban image sensing". Mobile phone-centric applications include microblogging ([7]), telemedicine (e.g. diet documentation [26]), and others [13].

While camera sensor networks present many exciting application opportunities, their design is challenging due to the size of images captured by a camera. In contrast to sensor network data management systems for low data-rate sensors such as temperature that can continually stream data from sensors to a central data gathering site, transmitting all but a small number of images is impractical due to energy constraints. For example, transmitting a single VGA-resolution image over a low-power CC2420 wireless radio takes up to a few minutes, thereby incurring significant energy cost. An alternate approach to designing camera sensor networks is to use image recognition techniques to identify specific entities of interest so that only images matching these entities are transmitted. Much of the work on camera sensor networks has employed such an approach (e.g.: [17]).

Instead of deploying such *application-specific* camera sensor networks, we argue that there is a need for a *general-purpose image search paradigm* that can be used to recognize a variety of objects, including new types of objects that might be detected. This can enable more flexible use of a camera sensor network across a wider range of users and applications. For example, in a habitat monitoring camera sensor network, many different end-users may be able to use a single deployment of camera sensors for their diverse goals including monitoring different types of birds or animals. Two recent technology trends make a compelling case for a search-based camera sensor network. The first trend is recent advances in image representation and retrieval makes it possible to efficiently compute and store compact image representations (referred to as visual terms or visterms [6,

30]), and efficiently search through them using techniques adapted from text retrieval [24, 25]. Second, flash memory storage is cheap, plentiful and extremely energy-efficient [21], hence images can be stored locally at sensor nodes for long durations and retrieved on-demand.

The ability to perform "search" over a sensor network also provides a natural and rich paradigm for querying sensor networks. Although there has been considerable work on energy-efficient query processing strategies, their focus has been on SQL-style equality, range, or predicate-based queries (e.g. range queries [5], min and max [29]; count and avg [19], median, and top-k [28]; and others [9]). The closest work to ours is on text search in a distributed sensor network [31, 32]. However, their work is specific to text search and assumes that users can annotate their data to generate searchable metadata. In contrast to these approaches, we seek to have a richer and automated methods for searching through complex data types such as images, and to enable post-facto analysis of archived sensing data in such sensor networks.

While distributed search in camera sensor networks opens up numerous new opportunities, it also presents many challenges. First, the resource constraints of sensor platforms necessitate efficient approaches to image search in contrast to traditional resource-intensive techniques. Second, designing an energy-efficient image search system at each sensor necessitates optimizing local computation and local storage on flash. Finally, distributed search across such a network of camera sensors requires ranking algorithm to be consistent across multiple sensors by merging query results. In addition, there is a need for techniques to minimize the amount of communication incurred to respond to a user query.

In this paper, we describe a novel general distributed image search architecture comprising a wireless camera sensor network where each node is a local search engine that senses, stores and searches images. The system is designed to efficiently (both in terms of computation and communication) merge scores from different local search engines to produce a unified global ranked list. Our search engine is made possible by the use of compact image representations called visterms for efficient communication and search, and the re-design of fundamental data structures for efficient flash-based storage and search. Our system is implemented on a network of iMote2 sensor nodes equipped with the Enalab cameras [1] and custom built SD card boards. Our work has the following key contributions:

- **Efficient Local Storage and Search**: The core of our system is an image search engine at each sensor node that can efficiently search and rank matching images, and efficiently store images and indexes of them on local flash memory. Our key contributions in this work include the use of an efficient image descriptor using "visterms" (see next section) and the re-design of two fundamental data structures in an image search engine — the vocabulary tree and the inverted index — to make it efficient for flash-based storage on resource-constrained sensor platforms. We show that our techniques improve the energy consumption and response time of performing local search by 5-6x over alternate techniques.

- **Distributed Image Search**: Our second contribution is a novel distributed image search engine that unifies the local search capabilities at individual nodes into a networked search engine. Our system enables seamless merging of scores from different local search engines across different sensors to generate a unified ranked list in response to queries. The compact image description in terms of visterms (see next section) minimizes communication overhead. We show that such a distributed search engine enables a user to query a sensor network in an energy-efficient manner using an iterative procedure involving the communication of local scores, representations and images to reduce energy consumption. Our results show that such a distributed image search is up to 5x more efficient than alternate approaches, while incurring reasonable increase in latency (less than four seconds in a four-hop network).

- **Application Case Studies**: We evaluate and demonstrate the performance of our distributed image search engine in the context of an indoor book monitoring application. We show that our system achieves up to 90% accuracy for search. We also show how system parameters can be tuned to tradeoff query accuracy for energy efficiency and response time.

2. BACKGROUND

Before presenting the design of our system, we first provide a concise background on the state of art image search techniques, and identify the major challenges in the design of an embedded image search engine for sensor networks. Image search involves three major steps: (a) extraction of distinguishing features from images, (b) clustering features to generate compact descriptors called visterms, (c) ranking matching results for a query, based on a weighted similarity measure called tf-idf ranking [3].

2.1 Image Search Overview

Image Features: A necessary pre-requisite for performing image search is the availability of distinguishing image features. While such features are not available for all kinds of image types and recognition tasks, several promising techniques have emerged in recent years. In particular, when one is interested in searching for images of the same object or scene, a good representation is obtained using the Scale-Invariant Feature Transform (SIFT) [16], which generates 128 dimensional vectors by essentially computing local orientation histograms. Such a SIFT vector is typically a good description of the local region. While a number of SIFT variants like GLOH and PCA-SIFT are available, a comparison [22] shows that GLOH and SIFT work best.

Visterms or Visual Words: While search can be performed by directly comparing SIFT vectors of two images, this approach is very inefficient. SIFT vectors are continuous 128 dimensional vectors and there are several hundred SIFT vectors for a VGA image. This makes it expensive to compute a distance measure for determining similarity between images. State-of-the-art image search techniques deal with this problem by clustering image features (e.g. SIFT vectors) using an efficient clustering algorithm such as hierarchical k-means [24], and by using each cluster as a visual word or visterm, analogous to a word in text retrieval [30].

The resulting hierarchical tree structure is referred to as the *vocabulary tree* for the images, where the leaf clusters form the "vocabulary" used to represent an image [24]. The

vocabulary tree contains both the hierarchical decomposition and the vectors specifying the center of each cluster. Since the number of bits needed to represent the vocabulary is far smaller than the number of bits needed to represent the SIFT vector, this representation is very efficient. We replace each 128 byte SIFT vector with a 4 byte visterm.

Matching: Image matching is done by comparing visterms between two images. If two images have a large number of visterms in common they are likely to be similar. This comparison can be done more efficiently by using a data structure called the *inverted index* or inverted file [3], which provides a mapping between a visterm and all images that contain the visterm. Once the images to compare are looked up using the inverted index, a query image and a database image can be matched using visterms by scoring them. As is common in text retrieval, scoring is done by weighting more common visterms less than rare visterms [3, 24, 25]. The rationale is that if a visterm occurs in a large number of images, it is poor at discriminating between them. The weight for each visterm is obtained by computing the tf-idf score (term frequency - inverse document frequency) as follows:

$$\begin{aligned} tf_v &= \text{Freq. of visterm v in an image} \\ df_v &= \text{Num. of images in which visterm v occurs} \\ idf_v &= \frac{\text{Total num of images}}{\text{df}_v} \\ score &= \sum_i \log(tf_i + 1) \cdot \log(idf_i) \end{aligned} \quad (1)$$

where the index i is over visterms common to the query and database image and df_v denotes the document frequency of v. Once the matching score is computed for all images that have a visterm in common with the query image, the set of images can be ranked according to this score and presented to the user.

2.2 Problem Statement

There are many challenges in optimizing such an image search system for the energy, computation, and memory constraints of sensor networks. We focus on three key challenges in this paper:

- **Flash-based Vocabulary Tree:** The vocabulary tree data structure is typically very large in size (many tens of MB) and needs to be maintained on flash. While the data structure is static and is not modified once constructed, it is heavily accessed for lookups since every conversion of an image feature to visterm requires a lookup. Thus, our first challenge is: *How can we design a lookup-optimized flash-based vocabulary tree index structure for sensor platforms?*

- **Flash-based Inverted Index:** A second important data structure for search is the inverted index. As the number of images captured by a sensor grows, the inverted index will grow to be large and needs to be maintained on flash. Unlike the vocabulary tree, the inverted file is heavily updated since an insertion operation occurs for every visterm in every image. In contrast, the number of lookups on the inverted index depends on the query frequency, and can be expected to be less frequent. Thus, our second challenge is: *How can we design an update-optimized flash-based inverted index for sensor platforms?*

- **Distributed Search:** Existing image search engines are designed under the assumption that all data is available centrally. In a sensor network, each node has a unique and different local image database, therefore, we need to address questions about merging results from multiple nodes. In addition, sensor network users can pose different types of queries — continuous and ad-hoc — which need to be efficiently processed. Thus, our third challenge is: *How can we perform efficient and accurate distributed search across diverse types of user queries and diverse local sensor databases?*

The following sections discusses our overall architecture followed by techniques employed by our system to address the specific problems that we outlined in this section.

3. IMAGE SEARCH ARCHITECTURE

We now provide a broad overview of the operation of our distributed image search system, which we describe in the context of a bird monitoring sensor network. We assume that users of such a sensor network wish to pose archival queries on stored images as well as continuous queries on live images. An example of an archival query might be to retrieve the top five matches of a user-provided query image, say a hawk, that were detected by the sensor network in the last month. The user can also pose a continuous query, and request to be notified whenever a newly captured image matches the query image of the hawk. Such image search over a sensor network involves the following steps:

Image Capture and Feature Extraction: The first step involves capture of images by a camera sensor node, perhaps in a periodic or triggered manner. A simple motion detection algorithm may be used to filter images such that only potentially interesting images are retained for further processing. Once such an image is captured, each sensor extracts descriptive features using the SIFT algorithm and maps these features to visterms as described in Section 2. This process of mapping SIFT features to visterms involves looking up a vocabulary tree which can either be pre-loaded onto the sensor during deployment time, or dynamically downloaded from a server during in-situ operation.

Local Search: Search can proceed in two ways. One approach ("Query by Visterms") is to do local search at sensors *i.e.* to transmit visterms of query images to individual sensors, perform the search locally at the sensor nodes, and merge results from multiple sensors to generate a global search result. An alternate approach ("Collect Visterms") is to do *centralized search i.e.* to push visterms for all captured images from all sensors to a central proxy which indexes these visterms to search across images stored at sensors. While both modes of operation are supported by our system, their relative efficiencies depend on specific sensor platforms and infrastructure availability. Here, we restrict our discussion to the local search mechanism since it is a more general paradigm.

The local image search engine handles continuous and archival queries in different ways. For an archival query, the visterms of the image of interest to the query are matched against the visterms of all the images that are stored in

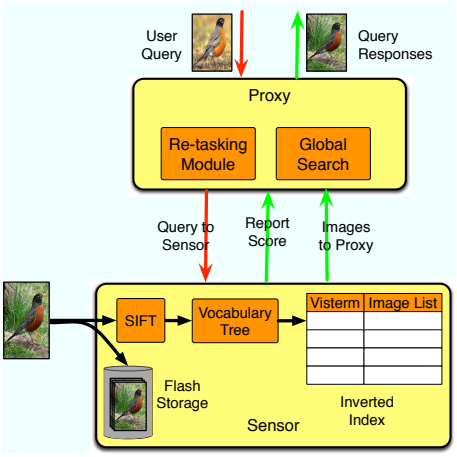


Figure 1: Search Engine Architecture

the local image store. The result of the local search is a top-k ranked list of best matches, which is transmitted to the proxy. For a continuous query, each captured image is matched against the specific images of interest to the query, and if the match is greater than a pre-defined threshold, the captured image is considered a match and transmitted to the proxy.

Global Search: Once the local search engine has searched for images matching the query, the proxy and the sensor interact to enable global search across a distributed sensor network. This interaction is shown in Figure 1. Global search involves combining results from multiple local search engines at different sensors to ensure that communication overhead is minimized across the network. For instance, it is wasteful if each sensor transmits images corresponding to its local top-k matches since the local top-k matches may not be the same as the global top-k matches to a query. Instead, the proxy gets the ranking scores corresponding to the top-k matches resulting from the local search procedure at each sensor. The proxy merges the scores obtained from different sensors to generate a global top-k list of images, which it communicates to the appropriate sensors. The sensors then transmit thumbnails or the full images of the requested images, which are presented to the user using a GUI.

4. BUFFERED VOCABULARY TREE

The vocabulary tree is the data structure used at each sensor to map image features (for example SIFT vectors) to visual terms or visterms. We now provide an overview of the operation of the vocabulary tree, and describe the design of a novel index structure, the Buffered Vocabulary Tree index structure, that is optimized for flash-based lookups.

4.1 Description

The vocabulary tree at each sensor is used to map SIFT vectors extracted from captured images to their corresponding visterms that are used for search. The vocabulary tree is typically created using a hierarchical k-means clustering of all the SIFT features in the image database [24]. Hierarchical k-means assumes that the data is first clustered into a small number of m clusters. Then at the next level, the points in each of the m clusters are clustered into a further m clusters so that this level has m^2 clusters. The process is repeated so that at a depth d the number of clusters is m^d. The process stops when the desired number of leaf clusters is reached. For example with $m = 10$ and $d = 6$ there are a million clusters at the leaf level. The clusters at the leaf level correspond to visual words or visterms. The ID of the leaf node is the ID of the visterm; for example, if a SIFT vector is mapped to the second cluster in the vocabulary tree, its visterm ID is 2. In the resulting vocabulary tree each level has a number of clusters, where each cluster is represented by the coordinates of the cluster center, as well as pointers from each node to its parent, children, and siblings.

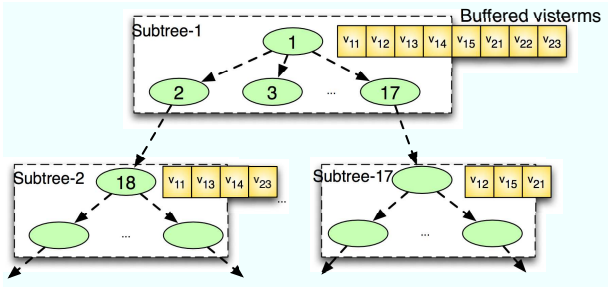


Figure 2: Vocabulary Tree

Lookup of the tree proceeds in a hierarchical manner where the SIFT vector is first compared with the m cluster centers at the top level and assigned to the cluster with the closest center at this level c_k. Then, the SIFT vector is compared with the centers of the siblings of c_k and assigned to the closest sibling c_{kj}. The process repeats until the SIFT vector is assigned to a leaf node or visterm.

4.2 Design Goals

The design of the vocabulary tree has two key objectives:

- *Minimize tree construction cost:* The process of constructing the vocabulary tree for a set of database images is computationally intensive. Thus, our first goal is to minimize the cost of tree construction.
- *Minimize Flash reads:* The vocabulary tree is a large data structure (few to many MB), hence it may not be possible to load the data structure completely to memory on a memory-constrained embedded platform (e.g. iMote2). Alternatively, we need to store it on flash and load it partially for visterm lookup. Since reading the vocabulary tree from flash incurs considerable latency, and consequently energy, our second goal is to minimize the number of reads from flash for every lookup of the vocabulary tree.

4.3 Proxy-based Tree Construction

Unlike conventional search engines where the vocabulary tree is created from all the images that need to be searched, our approach separates the images used for tree *construction* from those used for *search*. The proxy constructs the vocabulary tree from images similar (but not necessarily identical) to those we expect to capture within the sensor network. For example, in a book search application, a training set can comprise a variety of images of book covers for generating the vocabulary tree at the proxy. The images can even be captured using a different camera with different resolution

from the deployed nodes. Once constructed, the vocabulary tree can either be pre-loaded onto each sensor node prior to deployment (this can be done by physically plugging in the flash drive to the proxy and then copying it), or can be transmitted to the network during system operation.

One important consideration in constructing the vocabulary tree for sensor platforms is ensuring that its size is minimal. Previous work in the literature has shown that using larger vocabulary trees produces better results [24, 25]. This work is based on using trees with a million leaves or more which is many Gigabytes in size. Such a large vocabulary tree presents three problems in a sensor network context: (a) they are large and consume a significant fraction of the local flash storage capacity, (b) they incur greater access time to read, thereby greatly increasing the latency and energy consumption for search, and (c) they would be far too large to dynamically communicate to update sensor nodes. To reduce the size of the vocabulary tree, our design relies on the fact that typical sensor networks have a small number of entities of interest (e.g.: a few books, birds, or animals), hence, the vocabulary tree can be smaller and targeted towards searching for these entities. For example, in a book monitoring application, a few hundreds of books can be used as the training set to construct the vocabulary tree, thereby reducing the number of visterms and consequently the size of the tree.

4.4 Buffered Lookup

The key challenge in mapping a SIFT vector to a visterm on a sensor node is minimizing the overhead of reading the vocabulary tree from flash. A naive approach that reads the vocabulary tree from flash as and when required is extremely inefficient since it needs to read a large chunk of the vocabulary tree for practically every single lookup.

The main idea in our approach is to reduce the overhead of reads by performing batched reads of the vocabulary tree. Since the vocabulary tree is too large to be read into memory as a whole, it is split into smaller sub-trees as shown in Figure 2, and each subtree is stored as a separate file on flash. The subtree size is chosen such that it fits within the available memory in the system. Therefore, the entire subtree file can be read into memory. Second, we batch the SIFT vectors from a sequence of captured images into a large in-memory SIFT buffer. Once the SIFT buffer is full, the entire buffer is looked up using the vocabulary tree one level at a time. Thus, the root subtree (subtree 1 in Figure 2) is first read from flash, and the entire batch of SIFT vectors is looked up on the root subtree. This lookup determines which next level subtree needs to be read for each SIFT vector, and results in a smaller set of second-level buffers. The next level sub-trees are looked up one by one, and the batch processed on each of these sub-trees to generate third-level buffers, and so on. The process proceeds in a level by level manner, with a subtree file being read, and a buffer of SIFT vectors being looked up at each level. Such a buffered lookup on a segmented vocabulary tree ensures that the cost of reading an entire vocabulary tree is amortized over a batch of vectors, thereby reducing the amortized cost per lookup.

5. INVERTED INDEX

While the vocabulary tree is used to determine how to map from SIFT feature vectors to visterms, an inverted index (also known as the inverted file) as in text retrieval [3] is used to map a visterm to the set of images in the local database that contain the visterm. The inverted index is used in two situations in our system. First, when an image is inserted into the local database the visterms for the image are inserted into the inverted index; this enables search by visterms, wherein the visterms contained in the query image can be matched to the visterms contained in the stored locally captured images. Second, the inverted index is also used to determine which images to age when flash is filled up to make room for new images. Aging of images proceeds by first determining the images that are least likely to be useful for future search, and deleting them from flash.

5.1 Description

The inverted index is updated for every image that is stored in the database. Let the sequence of visterms contained in image I_i be $\mathbf{V_i} = v_1, v_2, ..., v_k$. Then, the entry I_i is inserted into the inverted index entry for each of these visterms contained in $\mathbf{V_i}$. Figure 3 shows an example of an inverted index that provides a mapping from the visterm to the set of images in the local database that contain the term, as well as the frequency of the appearance of the term across all images in the local database. Each entry is indexed by the Visterm ID, and contains the document frequency (df) score of the visterm (Equation 1), and a list of Image IDs. We use a modified version of the scoring function 2 which does not use the term frequency (tf) but uses only the inverse document frequency (idf). As we show later, the idf scores are sufficient for good performance in our system. Hence, we do not store the term frequency (tf) numbers per image - this also saves valuable memory space. Since the df and idf have to be updated when new images are added, it is more efficient to store the df and compute the idf at query time.

The inverted index facilitates querying by visterms. Let the set of visterms in the query image be $\mathbf{Q} = q_1, q_2, ..., q_n$. Each of these visterms is then used to look up the inverted index and the corresponding inverted list for the visterm returned. Thus for query visterm q_i, a list of image IDs $\mathbf{L_i} = I_{i1}, ..., I_{ik}$ is returned, where each element of the list is an image ID. The lists over all the query visterms are intersected and scored to obtain a rank ordering of the images with respect to the query.

5.2 Design Goals

Unlike the vocabulary tree which is a static data structure that is never updated, the inverted index is updated for every visterm in a captured image, hence it needs to be optimized for insertions. The design of the flash-based inverted index structure is influenced by the following characteristics of the flash layer below and from the search engine layer above, and has the following goals:

- *Minimize flash overwrites:* Flash writes are immutable and one-time—once written, a data page must be erased before it can be written again. The smallest unit that can be erased on flash, termed an *erase-block*, typically spans few tens of pages, which makes any in-place overwrite of a page extremely expensive since it incurs the cost of a block read, write and erase. Hence, it is important to minimize the number of overwrites to a location that has been previously written to on flash.

- *Exploit visterm frequency:* The frequency of words in documents typically follows a heavy tailed behavior,

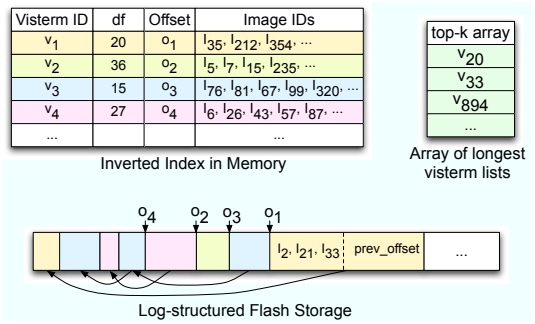


Figure 3: Inverted Index. DF denotes document frequency

referred to as a Zipf distribution, *i.e.* the frequency of a word is inversely proportional to its rank [3]. From a search perspective, the least frequent words are the most useful for discriminating between images, and have the highest idf score. Our third goal is to exploit this search engine characteristic to optimize the inverted index design.

5.3 Inverted Index Design

We discuss three aspects of the design of the inverted index in this section: (a) how the index is stored on flash, (b) how it is updated when a new image is captured, and (c) how entries are deleted when images are removed from flash.

Storage: The inverted index is maintained on flash in a *log-structured* manner, *i.e.* instead of updating a previously written location, data is continually stored as an append-only log. Within this log, the sequence of image IDs corresponding to each visterm is stored as a reverse linked list of chunks, where each chunk has a set of image IDs and a pointer to the previous chunk. For example, in Figure 3, the set of Image IDs corresponding to visterm v_1 is stored in flash as two chunks, with a pointer from the second chunk to the first chunk. The main benefit of this approach is that each write operation becomes significantly cheaper since it only involves an append operation on flash, and avoids expensive out-of-place rewrites. In addition, the writes of multiple chunks can be coalesced to reduce the number of writes to flash, thereby reducing the fixed cost of accessing flash for each write. Such a reverse linked list approach to storing files is also employed in a number of other flash-based data storage systems that have been designed for sensor platforms including MicroSearch [31], ELF [4], Capsule [20], and FlashDB [23].

Insertion: While a log-structured storage optimizes the write overhead of maintaining an inverted index, it increases the read overhead for each query since accessing the sequence for each visterm involves multiple reads in the file. To minimize read overhead, we exploit the Zipfian nature of the idf distribution in two ways during insertion of images to the index. First, we exploit the observation that the most frequent terms have a very low contribution to the idf score, and have least impact on search. Hence, these visual terms can be ignored and do not have to be updated for each image. In our system, we determine these non-discriminating visual terms during vocabulary tree-construction time, so that they do not need to be updated during system operation.

Second, we minimize the read overhead by only flushing the "longest chains" to flash i.e. the visterms that have the longest sequence of image IDs. Due to the Zipfian distribution of term frequency, a few chains are extremely long, and by writing these to flash, the number of flash read operations can be tremendously reduced. The Zipfian distribution also reduces the amount of state that needs to be maintained for determining the longest chains. We store a small top-k list of the longest chains as shown in Figure 3. This list is updated opportunistically — whenever a visterm is accessed that has a list longer than one of the entries in the top-k list, it is inserted into the list. When the size of the inverted index in memory exceeds the maximum threshold, the top-k list is used to determine which visterm sequences to flush to flash. Since the longer sequences are more frequently accessed, the top-k list contains the longest sequences with high probability, hence this technique writes the longest chains to flash, thereby minimizing the number of flash writes. The use of the frequency distribution of visterms to determine which entries to flush to flash is a key distinction between the inverted index that we use and the one that is proposed in [31].

Deletion: Image deletions or aging of images triggers deletion operations on the inverted index. Deletion of elements is expensive since it requires re-writing the entire visterm chain. To avoid this cost, we use a large image ID space (32 bits) such that rollover of the ID is practically impossible during the typical lifetime of a sensor node.

5.4 Using Inverted Index for Aging

In addition to search, our system also employs the inverted index is to determine what images should be aged when flash is filled up. Ideally, images that are less likely to contain objects of interest should be aged before images that are more likely to contain objects of interest. We use a combination of value-based and time-based aging of images. The "value" of an image is computed as the cumulative idf of all the visterms in the image normalized by the number of visterms in the image. Images with lower idf scores are more likely to be the background rather than objects of interest. This value measure can be combined with the time when an image was captured to determine a joint metric for aging. For example, images that are more than a week old, and have normalized idf score less than a pre-defined threshold can be selected for aging.

6. DISTRIBUTED SEARCH

A distributed search capability is essential to be able to obtain information that may be distributed over multiple nodes. In our system, the local search engines at individual sensors are networked into a single distributed search engine that is responsible for searching and ranking images in response to a query. In this section, we discuss how global ranking of images is performed over a sensor network.

6.1 Search Initiation

A user can initiate a search query by connecting to the proxy, and transmitting a search request. Two types of search queries are supported by our system: ad-hoc search and continuous search. An ad-hoc search query (also known as a snapshot query) is a one-time query, where the user

provides an image of interest and, perhaps a time period of interest, and initiates a search for all images captured during the specified time period that match the image. For example, in the case of a book monitoring sensor network, a user who is missing his or her copy of "TCP Illustrated" may issue an ad-hoc query together with a cover of the missing book, and request all images matching the book detected over the past few days. A continuous search query is one where the network is continually processing captured images to decide whether it matches a specific type of entity. For instance, in the above book example, the user can also issue a continuous query and request to be notified whenever the book is observed in the network over the next week.

In both cases, the proxy which receives the query image converts the image into its corresponding visterms. The visterm representation of a query is considerably smaller than the original image (approx 1600 bytes), hence, this makes the query considerably smaller to communicate over the sensor network. The proxy reliably transmits the query image visterms to the entire network using a reliable flooding protocol.

6.2 Local Ranking of Search Results

Once the query image visterms are received at each sensor, the sensors initiate the local search procedure. We first describe the process by which images are scored and ranked, and then describe how this technique can be used for answering ad-hoc and continuous queries.

Scoring and Ranking: The local search procedure involves two steps: (a) the inverted index is looked up to determine the set of images to search, and (b) the similarity score is computed for each of these images to determine a ranked list. Let V_Q be the set of visterms in the query image. The first step involved in performing the local search is to find all images in the local database that have at least one of the visterms in V_Q. This set of images can be determined by looking up the inverted index for each of the entries v in V_Q. Let L_v be the list of image IDs corresponding to visterm v in the inverted index. Assume that the first visterm in V_Q is v_1 and the corresponding inverted list of images is L_{v_1}. We maintain a list of documents D, which is initialized with the list of images L_{v_1}. For each of the other visterms v in V_Q, the corresponding inverted index L_v is scanned and any image not in the document list D is added to it.

Once the set of images is identified, matching is done by comparing visterms between each image in $V(i)$, where $i \in D$ and the visterms in the query image V_Q. If the two images have a large number of visterms in common they are likely to be similar. Visterms are weighted by their *idf*. Visterms which are very common have a lower idf score, and are weighted less than uncommon visterms. The idf is computed as described in Equation 1.

To obtain the score of an image i, we add up the idf scores for the visterms that match between the query image, V_Q, and the database image, V_i to obtain the total score as shown in Equation 2. Note that the equation does not use the *tf* term from equation 1. This is for two reasons: (a) some non-discriminating visterms occur very frequently (e.g. background visterms), leading to false matches, and (b) the *tf* term of every visterm in every image needs to be stored, which would significantly increase the size of inverted index.

Figure 4: Examples of book cover search results. The first column shows queries and the top results are shown in the second column. The technique is resilient to occlusions and viewpoint change (top left image) and specularities (bottom left image).

$$Score(V_i, V_Q) = \sum_{i \in V_Q and i \in V_i} \log(idf_i) \quad (2)$$

Any image with a score greater than a fixed pre-defined threshold ($Score(V_i, V_Q) > Th$) is considered a match, and is added to the list of query results. The final step is sorting these query results by score to generate a ranked list of results, where the higher ranked images have greater similarity to the query image.

Figure 4 shows example queries and the top search result for a book search example. Note the ability to handle viewpoint change, occlusion and specularities.

Ad-hoc vs Continuous Queries: The local search procedures for ad-hoc and continuous queries use the above scoring and ranking method but differ in the database images that they consider for the search. An ad-hoc query is processed over the set of images that were captured within the time period of interest to the query. To enable time-based querying, an additional index can be maintained that maps each stored image to a timestamp when it was captured. For a few thousand images, such a time index is a small table that can be maintained in memory. Once the list of images to match is retrieved from the inverted index lookup, the time index is looked up to prune the list and only considers images that were captured during the time period of interest. The rest of the scoring procedure is the same as the mechanism described above. In the case of a continuous query, the search procedure runs on each captured image as opposed to the stored images. In this case, a direct image-to-image comparison is performed between the visterms of the captured image and those of the query image. If the similarity between the visterms exceeds a pre-defined threshold, it is considered a positive match. If the number of continuous queries is high, the cost of direct matching can be further reduced by using an inverted file.

6.3 Global Ranking of Search Results

The search results produced by individual sensors are transmitted back to the proxy to enable global scoring of search results. The key problem in global ranking of search results is re-normalizing the scores across separate databases. As shown in Equation 1, the local scoring at each sensor de-

pends on the total number of images in the local database at each sensor. Different sensors could have different number of captured images, and hence, differently sized local databases. Hence, the scores need to be normalized before comparing them.

To facilitate global ranking, each sensor node transmits the count of the number of images in which each visterm from the set V_Q occurs, in addition to the total number of images in the local database. In other words, it transmits the numerator and denominator of Equation 1 separately to the proxy. Note that only the numbers for the visterms which occur in the query need to be updated not all the visterms. A typical query image has about 200 visterms so we need to only send on the order of 1.6 KB from each sensor node that has a valid result. Let S be the set of sensors in the network, and C_{ij} be the count of the number of images captured by sensor s_i that contain the visterm v_j. Let N_i be the total number of images at sensor s_i. Then, the global idf of visterm v is calculated as:

$$idf_v = \frac{\sum_{v_i \in S} N_i}{\sum_{v_i \in S} C_{iv}} \quad (3)$$

Once the normalized idfs are calculated for each visterm, the scores for each image are computed in the same manner as shown in Equation 2. Finally, the globally ranked scores are presented to the user. The user can request either a thumbnail of an image on the list, or the full image. Retrieving the thumbnail provides a cheaper option than retrieving the full image, hence it may be used as an intermediate step to visually prune images from the list.

6.4 Network Re-Tasking

An important component of our distributed infrastructure is the ability to re-task sensors by loading new vocabularies. This ability to re-task is important for two reasons. First, it enables the sensor proxy to be able to update the vocabulary tree at the remote sensors to reflect improvements in the structure, for instance, when new training images are used. Second, this allows the search engine to upload smaller vocabulary trees as and when needed on to the resource constrained sensors. One of the benefits of loading smaller vocabulary trees on-demand is that it is less expensive than searching through a large vocabulary tree locally at each sensor. Third, when it is necessary to use the sensor network to search for new kinds of objects, a new vocabulary tree may be loaded. For example, assume we had a vocabulary tree to detect certain kinds of animals such as deer but we now want to re-task it to find tigers, we can easily build a new vocabulary tree at the proxy and download it. Dissemination of the new vocabulary into a sensor network can be done using existing re-programming tools such as Deluge [11].

7. IMPLEMENTATION

Each sensor node comprises an iMote2 sensor [12], an Enalab camera [1], and a custom SD-card extension board that we designed, as shown in Figure 5. The Enalab camera module comprises an OV7649 Omnivision CMOS camera chip, which provides color VGA (640x480) resolution. The iMote2 comprises a Marvell PXA271 processor which runs

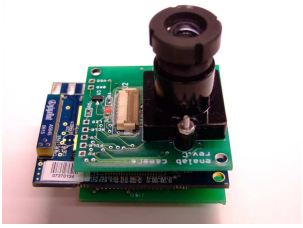


Figure 5: iMote2 with Enalab camera and custom SD card board

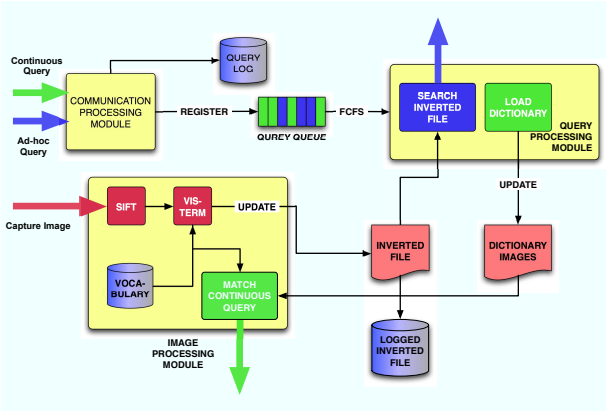


Figure 6: System Diagram

between 13-416 MHz, and has 32MB SDRAM[12]; a Chipcon CC2420 radio chip; and an Enalab camera. The camera is connected to the Quick Capture Interface (CIF) on iMote2. To support large image data storage, an 1GB external flash memory is attached to each sensor node.

Sensor Implementation: The overall block diagram of the search engine at each sensor is shown in Figure 6. The entire system is about 5000 lines of C code excluding the two image processing libraries that we used for SIFT and vocabulary tree construction. The system has three major modules: (a) Image Processing Module (IPM), (b) Query Processing Module (QPM), and (c) Communication Processing Module (CPM). The IPM captures an image using the camera periodically, reads the captured image and saves it into a PGM image file on flash. It then processes the PGM image file to extract SIFT features, and batches these features for a buffered lookup of the vocabulary tree.The QPM module processes ad-hoc queries and continuous queries from the proxy. For an ad-hoc query, it looks up the inverted file and find the best k matches from the locally stored images. If the value of k is not specified by the query, the default is set to five. For continuous queries, QPM loads the visterms of the corresponding dictionary images once a new type of query is received. Whenever a new image is captured, the IPM goes through all the dictionary images to find best k matches. CPM handles the communication to other sensors and the proxy.

SIFT Algorithm: The SIFT implementation we are using is derived from SIFT++ [27]. This implementation uses floating point for most calculations, which is exceedingly slow on the iMote2 (10 - 15 seconds to process a QVGA image) since it does not have a floating point unit. Ko

et al [14] present an optimized SIFT implementation which uses fixed point arithmetic and show much faster processing speed. However, their implementation is specific to the TI Blackfin processor family. For lack of time, we have been unable to port this implementation to our node.

Vocabulary Tree: Our implementation of the hierarchical k-means algorithm to generate the vocabulary tree is derived from the libpmk library [15]. Due to memory constraints on iMote2, we set the size of the vocabulary tree to have 64K visterms, *i*.e. the branching factor = 16, and depth = 5. By modifying libpmk, we can shrink the vocabulary tree to 3.65MB, but it is still too large to be loaded completely to memory in iMote2. As described in Section 4, we split the entire vocabulary tree into a number of smaller segments. The root segment contains the first three levels of the tree, and for each leaf node of the root subtree, i.e., $16^2 = 256$ in our case, there is a segment containing the last two levels of the tree. After splitting, each chunk is of size 14KB, which is small enough for iMote2 to read into memory.

Inverted Index: The inverted index is implemented as a single file on flash, which is used as an append-only log file. The inverted index is maintained in memory as long as sufficient memory is available, but once it exceeds the available memory, some image lists for visterms are written to the log file. Each of the logged entries is a list of image IDs, and the file offset to the previous logged entry corresponding to the same visterm. The offset of the head of this linked list is maintained in memory in order to access this flash chain. When the image list corresponding to a visterm needs to be loaded into memory, the linked list is traversed.

Image Storage and Aging: Each raw image is stored in a separate .pgm file, its SIFT features are stored in a .key file, and its visterms are stored in a .vis file. A four-byte image ID is used to uniquely identify an image. We expect the number of image that can be captured and stored on a sensor during its lifetime to be considerably less than the 2^{32}, therefore, we do not address the rollover problem in this implementation. Aging in our system is triggered when the flash becomes more than 75% full.

Wireless Communication Layer: The wireless communication in our system is based on the TinyOS MAC driver which is built upon the IEEE 802.15.4 radio protocol. This driver provides a simple open/close/read/write API for implementing wireless communication and is compatible with the TinyOS protocols. Since there is no readily available implementation of a reliable transport protocol for the iMote2, we implemented a reliable transport layer over the CC2420 MAC layer. The protocol is simple and has a fixed window (set to 2 in our implementation) and end-to-end retransmissions. We note that the contribution of our work is not a reliable transfer protocol, hence we were not focused on maximizing the performance of this component.

8. EXPERIMENTAL EVALUATION

In this section, we evaluate the efficiency of our distributed image search system on a testbed consisting of 6 iMote2 sensor nodes and a PC as a central proxy. For our trace-driven experiments, we use a dataset consisting of over 600 images for our training set. Among them, over three hundred images are technical book covers, and the other three hundred images contain random objects. The book cover images are collected from a digital camera with VGA format.

Table 1: Power and Energy breakdown

Component	State	Power(mW)	Per-byte Energy
PXA271 Processor	Active	192.3	
	Idle	137.0	
CC2420 radio	Active	214.9	46.39 μJ
SD Flash	Read	11.2	5.32 nJ
	Write	40.3	7.65 nJ
OV camera	Active	40.0	

The other images are collected from the Internet. Note that the database images we gathered do not need any further prossessing, such as cropping or alignment, thanks to the scale invariant property of SIFT features. During our live experiments, the iMote2 captures images periodically and different technical book covers were held up in front of the iMote2 camera to form the set of captured images.

8.1 Micro-benchmarks

Our first set of microbenchmarks are shown in Table 1, where we measure the power consumption of the PXA271 processor, the CC2420 radio, the SD card extension board, and the Enalab camera on our iMote2 platform. Since the data rates of different components vary, we also list the energy consumption for some of them. The results show that the processor consumes significant power on our platform, hence it is important to optimize the amount of processing. Another key observation is that the difference between the energy costs of communication and storage is significant. This is because the SD card consumes an order of magnitude less power than the CC2420 radio, and has a significantly higher data rate. The effective data rate of the CC2420 radio is roughly 46.6Kbps, whereas the data rate of the SD card is 12.5 MBps. Therefore, storing a byte is three orders of magnitude cheaper than transmitting a byte, thereby validating our extensive use of local storage.

Table 2 benchmarks the running time of the major image processing tasks in our system, including sensing, SIFT computation, and image compression. All of these tasks involve the processor in active mode. We find that the SIFT feature extraction from a QVGA image using the SIFT++ library [27] is prohibitively time consuming (roughly 12 seconds). A significant fraction of this overhead is because of floating point operations performed by the library, which consumed excessive time on a PXA271 since the processor does not have a floating point unit. This problem is solved in the fixed point implementation of SIFT described by Ko et al [14], who report a run time of roughly 2-2.5 seconds (shown in Table 2). Since the inflated SIFT processing time that we measure is an artifact of the implementation, we use the fixed point SIFT run-time numbers for the rest of this paper. The table also shows the the time required for lossy and lossless compression on the iMote2, which we perform before transmitting an image.

Table 3 reports the memory usage of major components in our system. The arm-linux on an iMote2 takes about half of the total memory, and the image capture and SIFT processing components take about a quarter of the memory. As a result of our optimizations, the inverted index and vocabulary tree are highly compact and consume only a few hundred kilobytes of memory. The dictionary file for

Table 2: Image processing breakdown

Operation	Time(s)	Energy(J)
Sensing(Image Capture)	0.7	0.14
Sift (floating pt)	12.7	2.44
Sift(fixed pt) [14]	2.5	0.48
Compress to JPEG (ratio 0.33)	1.2	0.23
Compress to GZIP (ratio 0.61)	0.7	0.14

Table 3: Breakdown of memory usage

Task	Memory (MB)
Arm Linux	14.3
Image Capture	1.9
SIFT	7.6
Inverted Index	0.75
Vocabulary Tree	0.2
Dictionary File	0.1
Processing Modules	0.7

handling continuous queries corresponds to the images in memory.

8.2 Evaluation of Vocabulary Tree

In this section, we evaluate the performance of our vocabulary tree implementation and show the benefits of partitioning and batching to optimize lookup overhead.

8.2.1 Impact of Batched Lookup

We evaluate the impact of batched lookup on a vocabulary tree with 64K visterms, which has a depth of 4 and a branching factor of 16. The vocabulary tree is partitioned into 257 chunks including one root subtree and 256 second-level subtrees. Each of these chunks is around 20KB. We vary the batch size from a single image, which is a batch of roughly 200 SIFT features, to ten images, *i.e.* roughly 2000 SIFT features.

Figure 7 demonstrates the benefit of batched lookup. Note that the y-axis is in log scale and shows the time to lookup visterms for each image. The upper line shows the reference time consumed for lookup when no batching is used, *i.e.* when each SIFT feature is looked up independently, and the lower plot shows the effect of batching on per-image lookup time as the number of batched images increases. As can be seen, even batching the SIFT features of a single image reduces the lookup time by an order of magnitude.

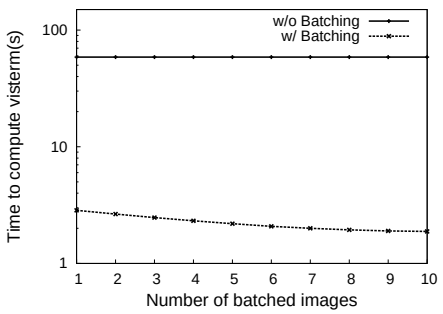


Figure 7: Impact of batching on lookup time.

Further batching reduces the lookup time even more, and when the batch size increases from 1 to 10 images, the time to lookup an image reduces from 2.8 secs to 1.8 secs. This shows that batched lookups, and the use of partitioned vocabulary trees has considerable performance benefit. The drawback of buffering is the delay since images are processed until a batch is full. It is not hard to observe that the average delay of each image is in proportion to the batch size, if the images are periodically captured. However, for the applications that are not in strictly realtime manner, the influence of delay can be overcome by choosing a proper buffer size.

8.2.2 Impact of Vocabulary Size

The size of the vocabulary tree has a significant impact on both the accuracy as well as the performance of search. From an accuracy perspective, the greater the number of visterms in a vocabulary tree, the better is the ability of the search engine to distinguish between different features. From a performance perspective, larger vocabularies mean greater time and energy consumed for both lookup from flash, as well as for dynamic reprogramming. To demonstrate this tradeoff between accuracy and performance, we evaluate three vocabulary trees of different sizes constructed from our training set. The accuracy and lookup numbers are averaged over twenty search queries.

Table 4 reports the impact of vocabulary size on three metrics: the lookup time per-image for converting SIFT feature to visterms, the search accuracy and the time to reprogram the vocabulary tree over the radio. In our evaluation, a matched result is defined as one of the top-k ranking list is correct, and accuracy is defined as the fraction of queries that have matched search results. We let k = 1,3,5 respectively in our experiment. The results show that the lookup time increases with increasing vocabulary size as expected, with over a second difference between lookup times for the vocabulary tree with 10K vs 83K nodes. The search accuracy increases with the size of the vocabulary tree as well. As the size of the vocabulary tree grows from 10K to 83K nodes, the accuracy increases by 15% from 0.73 to 0.88. In fact, the greatest gains in accuracy are made until the vocabulary tree becomes roughly 64K in size. Increasing the size of the vocabulary tree beyond 83K has a very small effect on the accuracy — a vocabulary tree with 100K nodes has an accuracy of 90% but larger trees give us no further improvement because of our small training set.

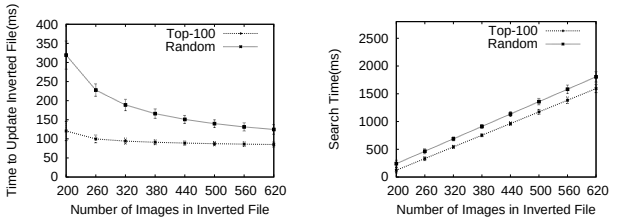
Table 4 also validates a key design goal — the ability to dynamically reprogram the sensor search engine with new vocabulary trees. The last column in the table shows the reprogramming time in the case of a one-hop network. A small sized vocabulary tree with 10K nodes is roughly 750 KB in size and can be reprogrammed in less than three minutes, which is not very expensive energy-wise and is feasible in most sensor network deployments. Even a larger vocabulary tree may be feasible for reprogramming, since trees with 64K and 83K nodes take about 20 minutes to reprogram.

8.3 Evaluation of Inverted Index

Having discussed the performance of the vocabulary tree, we turn to evaluating the core data structure used during search, the inverted index. In particular, we validate our use of the top-k list for determining which visterm sequences to save to flash. The inverted index is given approximately

Table 4: Impact of size of vocabulary tree.

#Visterms	Branching Factor	Depth	Size (MB)	Lookup Time(s)	top-most Accuracy	top-3 Accuracy	top-5 Accuracy	Response Time(min.)
10000	10	4	0.76	2.01	0.73	0.84	0.87	2.15
65536	16	4	4.92	2.85	0.82	0.86	0.87	14.08
83521	17	4	6.26	3.16	0.84	0.87	0.92	17.91



(a) Update time (b) Search time

Figure 8: Inverted index performance

1MB of memory, which is completely used once about 150 images are stored on the local sensor node. Beyond this point, further insertions result in writes of the inverted index to flash. The vocabulary tree size that we use in this experiment has 64K visterms.

We compare two methods of storing the inverted index in this experiment. The first technique, labeled "Random", is one where when the size of the inverted index increases beyond the memory threshold, a random set of 100 visterms are chosen, and their image lists are saved to flash. The second scheme, called "Top-100" is one where the indexes of the hundred longest visterm chains are maintained in a separate array in memory, and when the memory threshold is reached, the visterm lists with entries in this array are flushed.

Figure 8(a) reports the average time per image to update the inverted file using Random and top-100 methods. It can be seen that the update time using the top-100 method is less than half of the time for the Random scheme when there are roughly 200 images in the database and the gains reduce to about 25% when there are around 600 images. The diminishing gains with larger number of images is because the average length of the image list for each visterm increases with more images, therefore, even the Random list has a reasonable chance of writing a long list to flash.

Figure 8(b) presents the search time on the inverted file stored by Random and top-100 methods respectively. As the size of the inverted file grows, the search time also increases. However, the total time for search grows only up to a couple of seconds even for a reasonably large local image dataset with 600 images. The results also shows that the top-100 has less search time than random since it needs to read fewer times from flash during each search operation, although the benefits are only 10-15%.

8.4 Evaluation of Query Performance

We now evaluate the performance of our system for ad-hoc queries and continuous queries.

8.4.1 Benefits of Visterm Query

One of the optimizations in our search system is the ability to query by visterm *i.e.* instead of transmitting an entire image to a sensor, we only need to transmit visterms of the image. In this experiment, we compare the energy cost of visterm-based querying against two other variants. The first is an "Image query", where the entire image is transmitted to the sensor, which generates SIFT features and visterms from the query image locally, and performs local matching. The second scheme, labeled "SIFT query", corresponds to the case where the SIFT features for the query image are transmitted to the sensor, and the sensor generates visterms locally and performs matching. Here, we only measure the total energy cost of transmitting and interpreting the query, and do not include the cost of search to generate query results, which is common to all schemes. As shown in Table 5, transmitting only query visterms reduces the total cost of querying by roughly 20x and 10x in comparison with schemes that transmits the entire image or the SIFT features respectively. Much of this improvement is due to the fact that visterms are extremely small in comparison to transmitting the full image or SIFT features.

Table 5: Energy cost of querying (J)

Query Type	Communication	Computation	Total
Image query	3.5	0.52	4.02
SIFT query	2.45	0.04	2.49
Visterm query	0.01	0	0.01

8.4.2 Ad-hoc vs Continuous Query

Ad-hoc queries and continuous queries are handled differently as discussed in Section 6. Both queries require image capture, SIFT feature extraction, and visterm computation. After this step, the two schemes diverge. Ad-hoc query processing requires that an inverted file is updated when an image is captured, and a search is performed over the database images when a query is received. A continuous query is an image-to-image match where the captured image is matched against the images of interest for the active continuous queries.

Table 6 provides a breakdown of the energy cost of these components. The batch size used for the vocabulary tree is 10 images. As can be seen, our optimizations result in tremendously reduced visterm computation and search costs. Both continuous and ad-hoc queries consume less than 0.25 Joules per image. In fact, the cost is dominated by SIFT features computation (we address this in Section 8.6). Thus, both types of queries can be cheaply handled in our system.

8.5 Distributed Search Performance

Having evaluated several individual components of our search engine, we turn to an end-to-end performance evaluation of search in this section. In each experiment, the search system runs on the iMote2 testbed for an hour, and

Table 6: Energy cost of capturing and searching an image (J)

Component	Task	Energy (J)
Image Capture	Capture Image	0.04
Image Representation	Compute SIFT Feature	0.48
	Compute Visterm	0.04
Ad-hoc Querying	Update Inverted File	0.02
	Search	0.23
Continuous Querying	Match Visterm Histogram	0.16

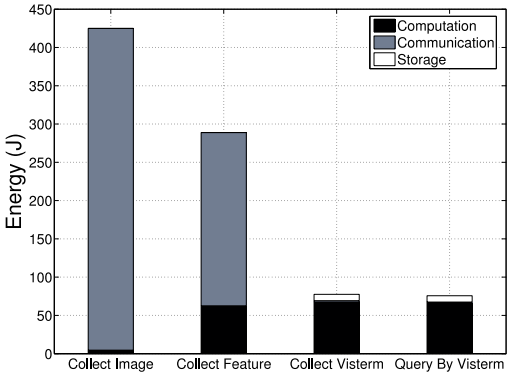


Figure 9: Total Energy cost of four mechanisms

the image capture rate is set to 30 seconds. The query rate is varied for different experiments as specified. The results show the aggregate numbers for different operations over this duration.

8.5.1 Push vs Pull

We compare two approaches to design a distributed search engine for sensor networks — a push-based approach vs a pull-based approach. There are three types of push-based approaches: (a) all captured images are transmitted to the proxy, in which case there is no need for any computation or storage at the local sensor, (b) when SIFT features are transmitted, and only the SIFT processing is performed at the sensor, and (c) when visterms are transmitted, therefore both SIFT processing and vocabulary tree lookup are performed locally. In a pull-based approach, the visterms

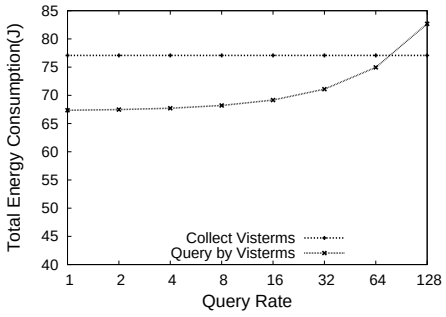


Figure 10: A comparison of collect visterms and query by visterms

corresponding to the query are transmitted to the sensors, and the query is locally processed at each sensor in the manner described in Section 6.

Figure 9 provides a breakdown of the communication, computation, and storage overhead for the four schemes — "Collect image", "Collect features" and "Collect Visterms" are three push-based schemes, and "Query by Visterms" is the in-network search technique that we use. The query rate is fixed to four queries an hour and the sampling rate is one image per 30 seconds. As can be seen, the computation overhead is highest for the query-by-visterms scheme, but the communication overhead is tremendously reduced; storage consumes only a small fraction of the overall cost. A query-by-visterms scheme consumes only a fifth of the energy consumed by an approach that sends all images to the proxy, and only a third of a scheme that sends SIFT features.

Figure 9 also shows that the "Collect Visterms" and "Query by Visterms" schemes have roughly equivalent performance. We now provide a more fine-grained comparison between the two schemes in Figure 10. A "Collect Visterms" scheme consumes more communication overhead for transmitting visterms of each captured image but does not incur the computation overhead to maintain, update, or lookup the inverted index for a query. The results show that unless the query rate is extremely high (more than one query/min), the query-by-visterms approach is better than a collect visterms approach. However, we also see that the difference between the two schemes is only roughly 15% since visterms are very compact and not very expensive to communicate. Since both schemes are extremely efficient, the choice between transmitting visterms to the proxy and searching at a proxy vs transmitting query visterms to the sensor and searching at the sensor depends on the needs of the application. Our system provides the capability to design a sensor network search engine using either of these methods.

Notice that our evaluation is carried out on the iMote2 platform where the CPU consumes almost the same power as the radio(see Table 1). If we apply our paradigm to a computation cheap platform, we expect that the benefit of querying by visterms would give us even higher benefit. Meanwhile, querying by visterms is a more flexible paradigm than collecting visterms since it doesn't need extra infrastructure to provide query processing and data storage functionality. For instance, sensor nodes can directly search with each other using a "querying by visterms" paradigm without the need for a central proxy.

8.5.2 Multi-hop Latency

So far, our evaluation has only considered a single hop sensor network. We now evaluate the latency incurred for search in a multi-hop sensor network. We place five iMote2 nodes along a linear multi-hop chain in this experiment, and configure the topology by using a static forwarding table at each node. The total round trip latency for a user search includes: (a) the time taken by the proxy to process the query image and generate visterms, (b) latency to transmit query visterms to the destination mote via a multihop network, (c) local search on the mote, (d) transmission of the local ranked list of query results, (e) merging the individual ranked lists at the proxy, and finally (f) transmission of the global ranked list to the sensors so that they can transmit thumbnails or full images. We do not include the time

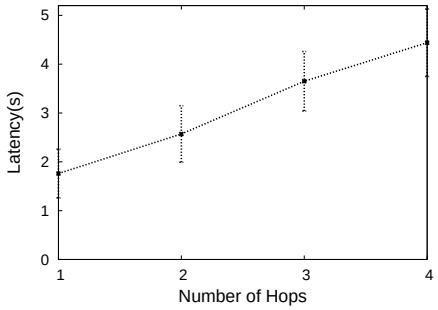
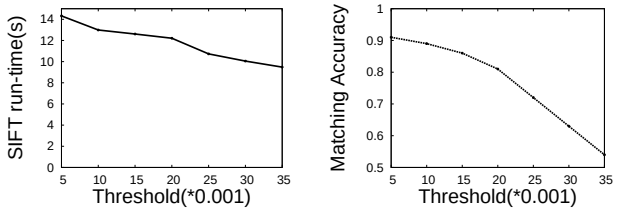


Figure 11: Round trip latency in multihop environment



(a) Run-time(s) (b) Accuracy

Figure 12: Impact of tuning SIFT threshold

taken to transmit the thumbnails or images in this experiment, and only take into account the additional overhead resulting from performing our iterative search procedure involving multiple rounds between the proxy and the sensor. The maximum size of a ranked list that can be transmitted by any sensor is set to five.

Figure 11 shows the round trip latency over multiple hops for a single query. As expected, the round trip latency increases as the number of hop grows, however, even for a four hop network, the overall search latency is only around 4 seconds, which we believe is acceptable given the increased energy efficiency.

In our evaluation, we broadcast queries to all the six nodes and send back visterms to proxy. In a large scale network, queries usually have other constraints, for instance, they may be restricted to specific geometric regions, or to a specific time frame. By adopting query processing algorithms, we can direct queries to a subset of nodes thus reduce the overhead. Also we can adopt in-network aggregation algorithm to reduce the energy cost of sending back visterms from sensors to proxy. We leave this aspect for future work.

8.6 Tuning SIFT Performance

As shown in Table 6, our optimizations of visterm extraction and search dramatically reduce the energy cost of these components, leaving SIFT processing as the primary energy consumer in our system. A key parameter in SIFT is the threshold that controls the sensitivity of SIFT features and we explore how to optimize it. Larger thresholds lead to fewer extracted features and vice-versa. In practice, matching accuracy increases with the number of features. The default threshold value in SIFT is 0.007, which corresponding to approximately 200 features for a QVGA image of a typical book cover.

Figure 12(a) shows the SIFT run-time (we use the floating point version since we do not have the optimized version) for different thresholds, and Figure 12(b) shows the corresponding image matching accuracy. The graphs show that as the threshold increases to 0.35, the running time of the algorithm drops by a third but the matching accuracy drops by around 30% as well. A reasonable operating region for the algorithm is to use a threshold between 0.005 and 0.02, which reduces SIFT processing time by about 2 seconds, while the accuracy is above 80%. Since these benefits are solely due to the reduction in the number of features, we believe that similar gains can be obtained with the fixed point version of SIFT.

9. RELATED WORK

In this section, we review closely related prior work on flash-based storage structures, camera sensor networks and distributed search and image recognition.

Flash-based Index Structures: There have been a number of recent efforts at designing flash-based storage systems and data structures including FlashDB [23], Capsule [20], ELF [4], MicroHash [33], and others. Among these, the closest are FlashDB, MicroHash and Capsule: FlashDB presents an optimized B-tree for flash, MicroHash is an optimized Hash Table for flash, and Capsule provides a library of common storage objects (stack, queue, list, array, file) for NAND flash. The similarities between our techniques and the approaches used by prior work is limited to the use of log-structured storage. Other aspects of our data structures such as the sorted and batched access of the vocabulary tree, and exploiting the Zipf distribution of the visterms are specific to our system.

Camera Sensor Networks: Much research on camera sensor networks has focused on the problem of image recognition, activity recognition (e.g.: [17]), tracking and surveillance (e.g. [10]). These systems are designed with specific applications in mind. In our design, a distributed camera search engine provides the ability for users to pose a broad set of queries thereby enabling the design of more general-purpose sensor networks. While our prototype focuses on any type of books, one can easily change this to other kinds of similar object and scenes provided appropriate features are available for that object or scene. SIFT features, are good for approximately planar surfaces like books and buildings [25] and may be even appropriate for many objects where a portion of the image is locally planar. SIFT features are robust to factors like viewpoint variation, lighting, shadows, sensor variation and sensor resolution. Further, robustness is achieved using a ranking framework in our case. There has also been work on optimizing SIFT performance in the context of sensor platforms [14]. However, their focus is on a specialized image recognition problem rather than the design of a search engine.

Search and Recognition: While there has been an enormous amount of work on image search in resource-rich server-class systems, image search on resource-constrained embedded systems has received very limited attention. The closest work is on text search in a distributed sensor network [31, 32]. However, their work assumes that users can annotate their data to generate searchable metadata. In contrast, our system is completely automated based on embedded object recognition techniques and does not require human endeavor in the loop.

10. DISCUSSION AND CONCLUSION

In this paper, we presented the design and implementation of a distributed search engine for wireless sensor networks, and showed that such a design is energy-efficient, and accurate. Our key contributions were five-fold. First, we designed a distributed image search system which represents images using a compact and efficient vocabulary of visterms. Second, we designed a buffered vocabulary tree index structure for flash memory that uses batched lookups together with a segmented tree to minimize lookup time. Third, we designed a log-based inverted index that optimizes for insertion by storing data in a log on flash, and optimizes lookup by writing longest sequences in flash. Fourth, we designed a distributed merging scheme that can merge scores across multiple sensor nodes. Finally, we showed using a full implementation of our system on a network of iMote2 camera sensor nodes that our system is up to five times more efficient than alternate designs for camera sensor networks.

Our work on distributed search engines opens up a number of new opportunities for sensor network research. We seek to extend our work to design a more general multimodal sensor search engine that can enable search across acoustic, image, vibration, weather or other sensor modalities. We also seek to explore new image representations that are suitable for habitat monitoring applications of sensor networks. One of the limitations of the SIFT features that we use in our work is that it works best when the scene or object is approximately planar. For example, the use of SIFT for extracting features is harder when there is less variation in image intensities - as for example on the surface of a uniformly colored bird. One of our areas of future work will involve new features that can be used for habitat monitoring in sensor networks.

Acknowledgments

This research was supported, in part, by NSF grants CNS-0626873, CNS-0546177, CNS-052072 and CNS-0619337. R. Manmatha is also supported by the Center for Intelligent Information Retrieval. We thank Richard Han for shepherding and the anonymous reviewers for their comments.

11. REFERENCES

[1] Enalab imote2 camera. http://enaweb.eng.yale.edu/drupal/. 2007.

[2] H. Aghajan, J. Augusto, C. Wu, P. McCullagh, and J. Walkden. Distributed vision-based accident management for assisted living. In *ICOST*, pages 196–205, 2007.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.

[4] H. Dai, M. Neufeld, and R. Han. ELF: an efficient log-structured flash file system for micro sensor nodes. In *ACM SenSys '04*, pages 176–187, 2004.

[5] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[6] S. Feng, R. Manmatha, and V. Lavrenko. Multiple Bernoulli relevance models for image and video annotation. In *IEEE CVPR*, pages 1002–1009, 2004.

[7] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-blog: Sharing and querying content through mobile phones and social participation. In *ACM MobiSys*, pages 174–186, 2008.

[8] R. Goshorn, J. Goshorn, D. Goshorn, and H. Aghajan. Architecture for cluster-based automated surveillance network for detecting and tracking multiple persons. In *ICDSC*, 2007.

[9] J. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Towards sophisticated sensing with queries. In *IPSN '03*, pages 63–79, 2003.

[10] S. Hengstler, D. Prashanth, S. Fong, and H. Aghajan. Mesheye: A hybrid-resolution smart camera mote for applications in distributed intelligent surveillance. In *IPSN-SPOTS*, pages 360–369, 2007.

[11] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *ACM SenSys*, pages 81–94, 2004.

[12] `http://www.intel.com/research/exploratory/motes.htm`. Intel imote2.

[13] A. Kansal, M. Goraczko, and F. Zhao. Building a sensor network of mobile phones. In *IPSN*, pages 547–548, 2007.

[14] T. Ko, Z. M. Charbiwala, S. Ahmadian, M. Rahimi, M. B. Srivastava, S. Soatto, and D. Estrin. Exploring tradeoffs in accuracy, energy and latency of scale invariant feature transform in wireless camera networks. In *ICDSC*, 2007.

[15] `http://people.csail.mit.edu/kkl/libpmk/`. LIBPMK: A Pyramid Match Toolkit.

[16] D. G. Lowe. Distinctive image features from scale-invariant keypoints. In *in* International Journal of Computer Vision, 60, *2004*, pages 91–110, 2004.

[17] D. Lymberopoulos, A. S. Ogale, A. Savvides, and Y. Aloimonos. A sensory grammar for inferring behaviors in sensor networks. In *IPSN*, pages 251–259, 2006.

[18] M. Rahimi and R. Baer and O. I. Iroezi and J. C. Garcia and J. Warrior and D. Estrin and M. Srivastava. Cyclops: In situ Image Sensing and Interpretation in Wireless Sensor Networks. In *ACM Sensys*, pages 192–204, 2005.

[19] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, Boston, MA, 2002.

[20] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Capsule: An energy-optimized object storage system for memory-constrained sensor devices. In *SenSys*, pages 195–208, 2006.

[21] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Ultra-low power data storage for sensor networks. In *IPSN-SPOTS*, pages 374–381, 2006.

[22] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE PAMI*, 27(10):1615–1630, October 2005.

[23] S. Nath and A. Kansal. Flashdb: dynamic self-tuning database for nand flash. In *IPSN*, pages 410–419, 2007.

[24] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, pages 2161–2168, 2006.

[25] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, 2007.

[26] S. Reddy, A. Parker, J. Hyman, J. Burke, D. Estrin, and M. Hansen. Image browsing, processing, and clustering for participatory sensing: Lessons from a dietsense prototype. In *EmNets*, 2007.

[27] `http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html`. SIFT++: A lightweight C++ implementation of SIFT.

[28] A. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *ICDE*, page 68, 2006.

[29] A. Silberstein, K. Munagala, and J. Yang. Energy-efficient monitoring of extreme values in sensor networks. In *ACM SIGMOD*, pages 157–168, 2006.

[30] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, pages 1470–1477, 2003.

[31] C. C. Tan, B. Sheng, H. Wang, and Q. Li. Microsearch: When search engines meet small devices. In *Pervasive*, pages 93–110, 2008.

[32] H. Wang, C. C. Tan, and Q. Li. Snoogle: A search engine for physical world. In *IEEE Infocom*, 2008.

[33] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W. Najjar. MicroHash: An efficient index structure for flash-based sensor devices. In *USENIX FAST*, 2005.