BU CS TR97-005. Appears in **Proc. IEEE Workshop on Content-based Access of Image and Video Libraries**, 6/97.

ImageRover: A Content-Based Image Browser for the World Wide Web

Stan Sclaroff, Leonid Taycher, and Marco La Cascia
Image and Video Computing Group
Computer Science Department
Boston University
Boston, MA 02215

Abstract

ImageRover is a search by image content navigation tool for the world wide web. To gather images expediently, the image collection subsystem utilizes a distributed fleet of WWW robots running on different computers. The image robots gather information about the images they find, computing the appropriate image decompositions and indices, and store this extracted information in vector form for searches based on image content. At search time, users can iteratively guide the search through the selection of relevant examples. Search performance is made efficient through the use of an approximate, optimized k-d tree algorithm. The system employs a novel relevance feedback algorithm that selects the distance metrics appropriate for a particular query.

Keywords: Image databases, query by image content, content-based retrieval, world wide web search engines.

1 Introduction

For a while now there have been software "robots" roving the World Wide Web (WWW) collecting index information about the text documents they find. These robots extract indexing information that is later used to guide interactive searches. The scale of these databases is impressive. For instance, at this writing Hot Wired's *HotBot* provides a searchable index of over 50 million documents.

What is needed are equivalent web *image* search engines that crawl the web collecting information about the images they find, computing the appropriate image decompositions and indices, and storing this extracted information for searches based on image content [1].

Web image search engines could be applied profitably in many areas; *e.g.*, in searching on-line catalogs of consumer goods and services, museums, libraries, and medical or other scientific data collections. Such engines might also be useful in the areas of erotica-on-demand, image copyright enforcement, forensics and intelligence gathering. Lastly, such a web image robot would be useful to machine vision researchers studying image databases, since it could provide a very large testbed for image database indexing methods.

Given the number of unsolved problems in image understanding building a web image robot seems overly ambitious. Fortunately, just as providing moderately useful text search tools need not require understanding the text's meaning, searching images need not require solving the image understanding problem. Due to the scale and unstructured nature of the WWW, even the most basic indexing tools would be welcome.

Lycos, and many others in the first generation of text engines extracted keywords using standard algorithms that consider statistics like word placement, word frequencies, *etc*. These indices did not provide more advanced tools for detecting word sense, synonymns, or word meaning; nor did they offer guarrantees that all WWW documents matching the search criteria would be found. Despite these drawbacks, users eagerly flocked to use these indices to navigate and sift through a burgeoning growth of web documents. Such text engines provided an important service for the WWW that was roughly akin to what Unix "grep" provides for searching ASCII text files. Subsequent generations of text engines were built on this humble foundation.

Similarly, the first generation of image engines need not require solving the image understanding problem. Instead, the general approach should be to provide an arsenal of decompositions and discriminants that can be precomputed for images: color histograms, edge orientation histograms, texture measures, shape invariants, eigendecompositions, *etc*. The resulting information is stored in vector form. At search time, users can select a weighted subset of these decompositions to be used for computing image similarity measurements [2; 3; 4]. This approach is taken in ImageRover, the system described in this paper.

The resulting search tool provides a powerful method for *data exploration* or browsing. The user typically makes queries like "find more things like this." In exploration mode, interactivity is essential or the user looses interest. For large databases, we have found that greater interactivity can be achieved through the use of approximation algorithms for indexing. These approximate matches are acceptable because they yield improved interactivity with a predictable, graceful degredation of accuracy.

2 Approach

Based on current estimates of the number of images on the WWW, a complete index of web images would provide access to somewhere between 10 and 30 million images.

The total number of documents on the WWW is estimated currently at 50-100 million, with 20-30% of the documents being images[5; 6]. The gargantuan size of a complete web image index presents challenges on a number of fronts:

1. **Image Collection.** In our experiments it has been observed that a single-threaded robot can traverse the web gathering images at an average rate of one image every 82 seconds. Gathering 10-30 million images has the potential to take on the order of 25 years with a single-threaded robot on a single computer. We address this problem by employing a fleet of robots distributed across many machines. Experiments indicate that our framework can allow a modest fleet of 32 robots to collect over one million images monthly.

2. **Image Digestion.** As images are gathered, the robot then needs to digest each image, extract the needed image statistics and decompositions, and create a reduced resolution image thumbnail. While the computation time needed to accomplish this dependends on the processor and algorithms employed, it is clear that a single computer cannot digest 10-30 million images in a reasonable amount of time. This hurdle can be cleared using our multi-processor approach.

3. **Image Index and Search.** For data navigation, we consider it a requirement that user's queries are answered within a second. This places severe constraints on the methods and metrics employed in searching for matches in the high-dimensional vector space of extracted image statistics for millions of images. As pointed out by White and Jain [7], search time is actually dependent on the "intrinsic dimensionality" of the space. This intrinsic dimensionality can be approximated through a principal components analysis and a dimensionality reduction. Speed is further enhanced via use of an approximate k-nearest neighbors indexing scheme.

4. **User Interface.** The standard approach is query by example or query by keyword. Keywords have limited usefulness, in that it is difficult to assign keywords consistently and exhaustively. In query by example it is difficult to determine the appropriate combination of similarity measures for a particular search. Directly prompting users for weightings is problematic, since it may require that users grasp the technical details of the underlying representation. One way around this is to allow the users to provide example images; this keeps nettlesome image content parameters hidden from the user. ImageRover employs a novel approach to this relevance feedback problem.

2.1 Related Work

To date, there have been a number of query-by-image content (QBIC) demos available via the web; *e.g.,* Virage [3], IBM QBIC [2], Cypress [9], Photobook [10], VisualSeek [11], Jacob [12], *etc.* Nearly all systems include some form of color and texture-based image similarity measures. In addition, some systems provide search on image composition [3], shape [2; 10], faces [10], and/or groupings of colored blobs [13]. None of these systems provides a web search engine, in that each only operates on a local demo database of a few thousand images stored at the host web site. However, the algorithms developed in these and other QBIC systems serve as an excellent starting point for building WWW image search engines.

ImageRover is joined by others in the first wave of image search engines: Yahoo's Image Surfer, Lycos media search tool, WebSeer[6], and WebSeek[11].

Both Yahoo's Image Surfer (by Interpix Software) and WebSeek provide primarily keyword-based browsing tool for WWW images that are grouped together by category and subcategory. Example categories are: actors and actresses, animals, architecture, arts, comics, dance, rock, sports, supermodels, toys, *etc.* At search time, the user can either page through the images or deploy a simple color histogram-based search for similar images within the subcategory. To build the index, images are semi-automatically classified into a hierarchy of categories with associated text labels. Unfortunately, semi-automatic algorithms or entry of metadata is infeasible given that millions of images will populate an index that evolves daily.

Two systems, Lycos and WebSeer extract keywords automatically from the image URL and possibly from captions imbedded in the document that contains the image. The WebSeer system [6] supplements keyword extraction with cues about image content: grayscale vs. color, image dimensions, file type and size, file date. In addition, their system includes a face detector that stores the number of faces and largest face size. During search, the user can type keywords describing that which is sought, and choose from a preset toggle menu the desired image dimensions, file size range, and color. If pictures of people are desired, the user can additionally use a preset toggle menu to specify th number of faces desired per image, and the desired camera shot (close-up, wide-angle).

If carefully extracted, keywords may help guide the search to basic categories. Unfortunately, keywords may not accurately or completely describe image content. Cues about image content need to come directly from the image, especially once the search hones in on a basic category, *e.g.,* cats, cars. While the ImageRover approach is similar in spirit to that of WebSeer, it differs in that it allows searches of web images based direcly on image content. ImageRover's overall system architecture and underlying algorithms will now be described in detail. The Im-

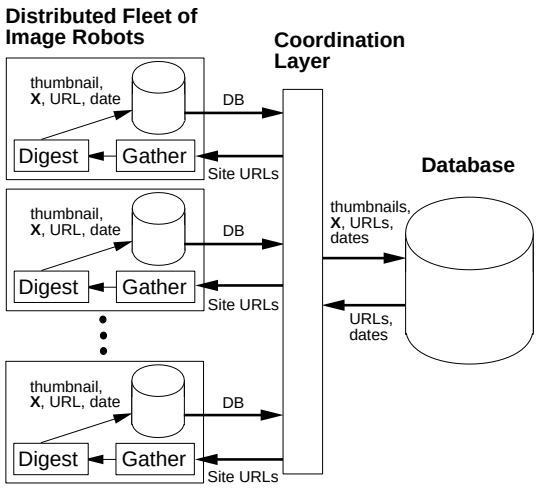

Figure 1: Image robot subsystem diagram. The image collection subsystem utilizes a distributed fleet of WWW robots running on different computers. Robots can contain *collection modules*, *digestion modules*, and a local database. The collection modules recursively parse and traverse WWW documents, gathering images. The digestion modules then process these images to extract needed image indexing information $\mathbf{X}$ and to compute a reduced resolution thumbnail image. The robots are dispatched and coordinated via a separate *coordination layer*, which also manages updates of the image index database.

ageRover system consists of two main components: an image collection subsystem, and an image search subsystem.

3 Image Collection Subsystem

The image collection subsystem utilizes a distributed fleet of WWW robots running on different computers. These robots can be run on a number of computers at a single site (as has been the case in the development of our initial system) or across a number of geographically-distributed computers at volunteer sites.

As shown in the Figure 1, robots can contain *collection modules*, *digestion modules*, and a local database. The collection modules recursively parse and traverse WWW documents, gathering images. The digestion modules then process these images to extract needed image indexing information and to compute a reduced resolution thumbnail image. The robots are dispatched and coordinated via a separate *coordination layer*, which also manages updates of the image index database.

In general, robots can contain either a collection module, a digestion module, or both. This allows the coordination layer to dispatch robots that operate alone and/or robots that operate in symbiosis, depending on the capabilities of available computers. The components of the digestion modules are described in the next section.

3.1 Image Digestion

Each image digestion module processes an input stream of image URLs. Processing begins with translating the image file format (*e.g.*, GIF, TIFF, JPEG) to the internal format, and performing color transformations. A reduced resolution image thumbnail is computed for use as an icon during search.

With preprocessing completed, the digestor then executes a series of image analysis submodules that calculate information about the distributions of color, texture, orientation, faces, or other properties of the image. Each submodule computes distributions over N subimages. In the current implementation, N = 6; distributions are calculated over the whole image and over five image subregions: center, upper right, upper left, lower right, lower left.

The resulting distribution information is then stored in vector form, with each of the modules contributing subvectors to an image index vector $\mathbf{X}$. Given M modules and N subimages, the image index vector will have $n = M \times N$ subvectors:

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} \tag{1}$$

The dimensionality of these subvectors is reduced significantly via a principal components analysis, as will be described in Section 4.

3.2 Image Analysis Submodules

At this writing there are two image analysis submodules fully-implemented in our system: color analysis and orientation analysis. Thus $M = 2$ in the current system. Efforts are currently underway to expand the system to include additional texture measures of multi-resolution simultaneous autoregressive models [14] and shift-invariant eigenvector models [15], and face detection and description using eigenfaces [16; 17].

Color distributions are calculated as follows. Image color histograms are computed in the CIE $L^*u^*v^*$ color space, which has been shown to correspond closely to the human perception of color [18].

To transform a point from RGB to $L^*u^*v^*$ color space, it is first transformed into CIE XYZ space:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.607 & 0.174 & 0.200 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.116 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2}$$

where the conversion matrix is for CIE Illiminant C (overcast sky at noon).

The $L^*u^*v^*$ values are then calculated as:

$$L^* = \begin{cases} 25 * (100 * \frac{Y}{Y_0})^{\frac{1}{3}} - 16 & \text{if } \frac{Y}{Y_0} \geq 0.008856 \\ 903.3\frac{Y}{Y_0} & \text{otherwise} \end{cases} \quad (3)$$

$$u^* = 13L^*(u' - u'_0) \quad (4)$$

$$v^* = 13L^*(v' - v'_0) \quad (5)$$

$$u' = \frac{4X}{X + 15Y + 3Z} \quad (6)$$

$$v' = \frac{9Y}{X + 15Y + 3Z} \quad (7)$$

where the reference values are $(X_0, Y_0, Z_0) = (0.981, 1.000, 1.820)$, and $(u'_0, v'_0) = (0.1830, 0.4198)$, for white under CIE Illuminant C.

For each of the subimages, the color distribution is then calculated using the histogram method [19]. Each histogram quantizes the color space into 64 (4 for each axis) bins. The over all histogram is normalized to have unit sum and then blurred.

The texture direction distribution is calculated using steerable pyramids [20; 21]. For this application, a steerable pyramid of 4 levels was found to be sufficient. At each level, texture direction and strength at each pixel is calculated using the outputs of seven X-Y separable, steerable quadrature pair basis filters.

The separable basis set and interpolation functions for the second derivative of a Gaussian were implemented directly using the nine-tap formulation provided in Appendix H (tables IV and VI) of [20]. The resulting basis is comprised of three G_2 filters to steer the second derivative of a Gaussian, and four H_2 filters to steer the Hilbert transform of the second derivative of a Gaussian.

At each level in the pyramid, the output of these filters is combined to obtain a first order approximation to the Fourier series for oriented energy $E_{G_2H_2}$ as a function of angle θ:

$$E_{G_2H_2} = C_1 + C_2\cos(2\theta) + C_3\sin(2\theta), \quad (8)$$

where the terms C_1,C_2, C_3 are as prescribed in [20], Appendix I.

Dominant orientation angle θ_d and the orientation strength m at a given pixel are calculated via the following formulae:

$$\theta_d = \frac{1}{2}arg\,[C_2, C_3] \quad (9)$$

$$m = \sqrt{C_2^2 + C_3^2}. \quad (10)$$

Orientation histograms are then computed for each level in the pyramid. Each orientation histogram is quantized to 16 bins, thus the number of bins allocated for direction information in one region is 64 (4 levels * 16 bins/ level). The histogram is then normalized to have unit sum. Once computed, the histogram must be circularly blurred to obviate aliasing effects and to allow for "fuzzy" matching of histograms during image search [15].

In practice, there must be a lower bound placed on the accepted orientation strength allowed to contribute to the distribution. For the implementation described in this paper, all the points with the strength magnitude less than 0.005 were discarded and not counted in the overall direction histogram.

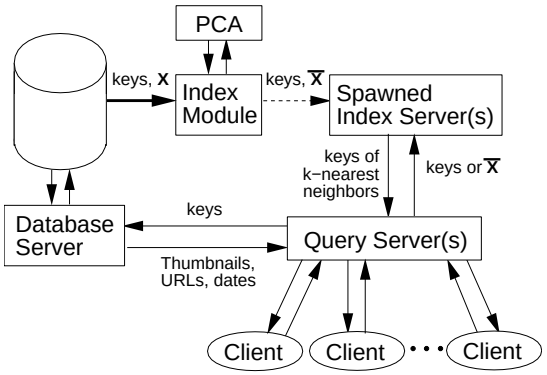

Figure 2: Interactive image query subsystem diagram.

4 Image Query Subsystem

The image indexing vectors $\mathbf{X}_i$ stored by the robots have rather high dimension. In the current system, the vectors have dimension 768. As pointed out by White and Jain [7], the data has *intrinsic dimension* that is significantly less than this. Furthermore, while it may be reasonable to assume a Gaussian distribution in the space, the distribution of samples may not be distributed uniformly across all dimensions. As a preliminary step, it is therefore useful to perform a dimensionality reduction via a principal components analysis (PCA) for each of the subvector spaces.

For each subvector space, we compute the eigenvectors and eigenvalues of their sample covariance matrix. The eigenvectors correspond to the principal axes of the subvector space, and the eigenvalues are the corresponding principal variances. For a very large database like that used for this application, it is only necessary to perform the PCA for a randomly selected subset of samples.

Although all eigenvectors are required to represent the distribution exactly, only a small number of vectors is generally needed to encode samples in the distribution within a specified tolerance. In practice, the first k eigenvectors are used, such that k is chosen to represent the variance in the dataset within some error threshold τ. In our experiments setting $\tau = 0.1$ (10% error), resulted in a dimension reduction of over 85%.

Using this truncated basis, each original image index vectors $\mathbf{X}$ undergo the dimensionality reducing transform,

producing a reduced vector $\bar{\mathbf{X}}$. This transform is performed once for all vectors in the database as a precomputation. It has the dual effect of 1.) concentrating the variance in a relatively small number of dimensions, and 2.) normalizing the principal directions by their inverse principal standard deviations, thus spreading samples more evenly within the k-dimensional space.

4.1 Query Server

The image query subsystem is based on a client-server architecture. At startup, the server first performs a dimensionality reduction, and then builds an optimized k-d tree[22], maintaining the data structure in main memory if possible. If it is not possible to store the complete data structure in memory, then the vectors are organized in a file using the same disk block for all the records belonging to the same bucket. With disk caching, the performance should be almost the same as storing the data in memory [7; 22].

Due to the computational scaling properties of k-d search in high-dimensional spaces, expected performance of search in the optimized k-d tree is not better than brute-force nearest neighbor search if k is significantly greater than log(number of records in database). Therefore, ImageRover employs an approximation factor in the optimized k-d search algorithm along the lines of [8]. The k-d tree "bounds overlap ball" test is modified to include an approximation factor ϵ. The output of the approximate algorithm is a set of data points. It has been proven that each of these output points is at a distance from the query point that is at most a factor $(1 + \epsilon)$ greater than the true nearest neighbor distances[8].

Once initialized, the index server runs as a process separate from the database query server, possibly on a different computer. For each query, a client connects to the server to send the query data and then waits for the resulting k nearest neighbors. The server performs the query and returns the results to the client.

4.2 User Interface

As depicted in Figure 2, queries are serviced by a separate query server running at the ImageRover WWW site. The interface is presented via a Web browser as an HTML document. ImageRover employs a query by example paradigm along the lines of [2; 3; 10].

To get the search going, a set of randomly selected images are shown to the user. The user can ask the system for another set of random images, or he/she can mark example "relevant" images from those presented.

Once the user finds and marks one or more images to guide the search, the user can initiate a query with a click on the search button. An example ImageRover search is shown in Figure 3. The query image(s) are also shown in the top of the screen. Similar images (the number of returned images is a user chosen value) are then retrieved and shown to the user in decreasing similarity order. This gives users an opportunity to see the collection of example images used so far. The user can then select other relevant images to guide next search and/or deselect one or more of the query images and iterate the query. There is no limit to the number of iterations in providing relevance feedback, nor in the number of example images.

Each thumbnail image is a hypertext link to the original image. By clicking on any thumbnail, the user can retrieve the desired image from its corresponding home WWW site.

Through the "search accuracy"' button, the user can specify the desired level of approximation factor employed in the k-nearest neighbors algorithm. Settings of "high," "medium," and "low" allow the user to control of the trade-off between search speed/accuracy. In our experiments these settings correspond with $\epsilon = 0.0$, 5.0, and 10.0 respectively.

In a typical search, a user starts the session browsing the database in a random way. When the user finds something similar to what he/she is looking for, he/she checks the corresponding check-box and asks the system for similar images. The system usually retrieves relevant images and false matches, the user checks the images that are more relevant with respect to what he/she is looking for and re-iterates the query until desired images are found. During a search, the user can deselect one or more of the example images and check some other images he/she finds more relevant.

4.3 Relevance Feedback

The ImageRover system employs a relevance feedback algorithm that selects appropriate L_m Minkowski distance metrics on the fly. The formulation of this algorithm is as follows.

Let $\bar{\mathbf{X}}$ and $\bar{\mathbf{Y}}$ denote image index vectors in a database. Let $\bar{\mathbf{x}}_i$ and $\bar{\mathbf{y}}_i$ denote subvectors corresponding to the output of a particular image analysis module for a particular region in the image (as described in Section 3.1).

We define the normalized L_m distance between two subvectors:

$$\tilde{L}_m(\bar{\mathbf{x}}_i, \bar{\mathbf{y}}_i) = \frac{L_m(\bar{\mathbf{x}}_i, \bar{\mathbf{y}}_i)}{\mu_m^{(i)}} \tag{11}$$

where the normalization factor is computed based on the probability distribution of the images contained in the database,

$$\mu_m^{(i)} = E[L_m(\bar{\mathbf{x}}_i, \bar{\mathbf{y}}_i)]. \tag{12}$$

The expected value $\mu_m^{(i)}$ can be computed off-line over an entire database or a statistically significant subset of it. Moreover, if the database is reasonably large, we don't need to recompute this factor when new images are added to the archive.

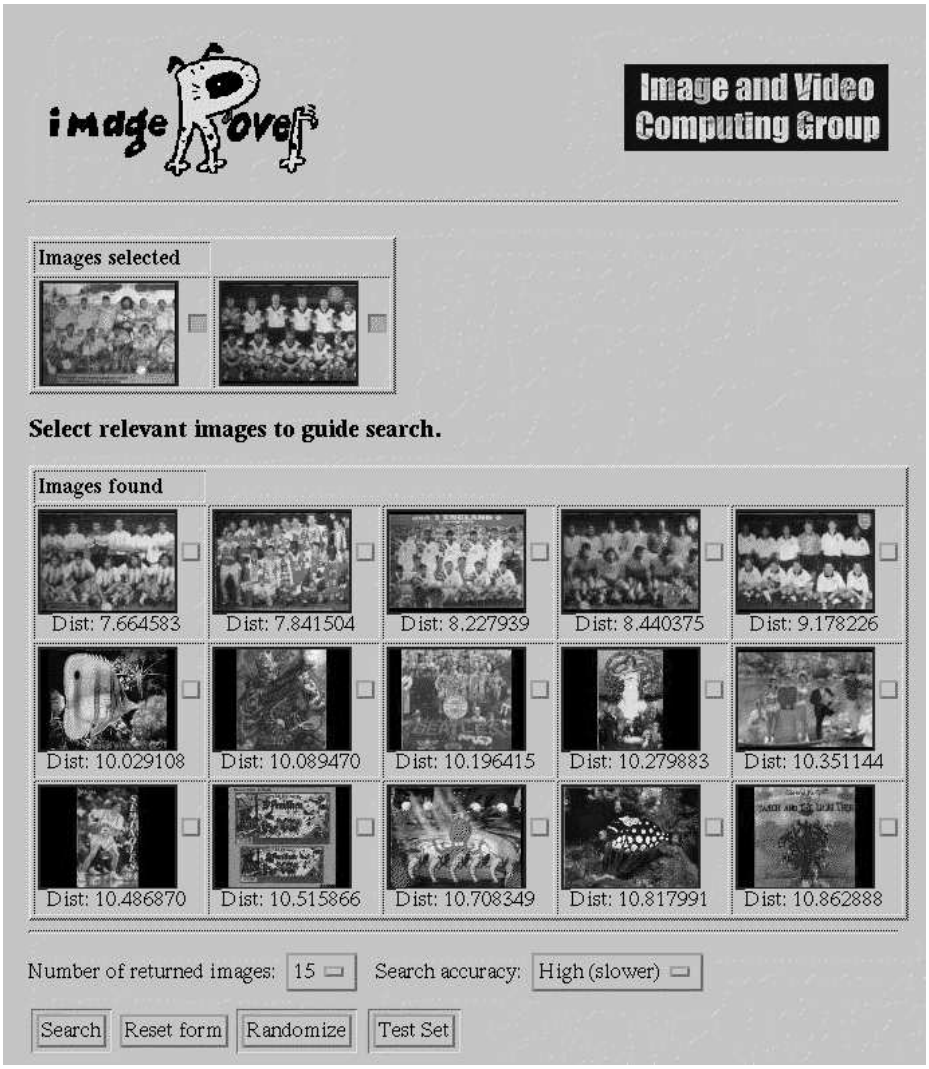

Figure 3: Example search for pictures of sports teams, based on relevance feedback from the user. The user-selected relevant images appear in the upper row (two example images of soccer team photos). The next three rows contain the retrieved 15 nearest neighbors. Images are displayed in similarity rank order, right to left, top to bottom. In this particular example, ImageRover ranked five more sport team photographs as closest to the user-provided examples. The other returned images share similar color and orientation distributions.

It is difficult to determine in advance which $\tilde{L}_m$ distance metric is best suited for a particular similarity detection task [23]. Therefore, our system selects the appropriate $\tilde{L}_m$ metric each time a query is made, based on relevance feedback from the user. Furthermore, instead of using the same metric for each image region and image measure, we allow selection of appropriate metrics for each of the various image index subvectors.

Assume that the user has specified a set S of *relevant* images. The appropriate value of m for the i^{th} subvector should minimize the mean distance between the relevant images. The dimension of the distance metric is determined as follows:

$$m_i = \arg\min_m \eta_m^{(i)} \tag{13}$$

where

$$\eta_m^{(i)} = E[\tilde{L}_m(\bar{\mathbf{p}}_i, \bar{\mathbf{q}}_i)], \quad \bar{\mathbf{P}}, \bar{\mathbf{Q}} \in S. \tag{14}$$

Queries by multiple examples are implemented in the following way. First, the mean query vector is computed for S. A k-nearest neighbor search of the image index then utilizes the following weighted distance metric:

$$\delta(\mathbf{X}, \mathbf{Y}) = \begin{pmatrix} w_1, w_2, \ldots w_n \end{pmatrix} \cdot \begin{pmatrix} \tilde{L}_{m_1}(\mathbf{x}_1, \mathbf{y}_1) \\ \tilde{L}_{m_2}(\mathbf{x}_2, \mathbf{y}_2) \\ \vdots \\ \tilde{L}_{m_n}(\mathbf{x}_n, \mathbf{y}_n) \end{pmatrix}, \tag{15}$$

where the w_i are *relevance weights*:

$$w_i = \frac{1}{\epsilon + \eta_m^{(i)}} \tag{16}$$

The constant ϵ is included to prevent a particular characteristic or a particular region from giving too strong a bias to the query.

The resulting relevance feedback mechanism allows the user to perform queries by example based on more than one sample image. The user can collect the images he or she finds during the search, refining the result at each iteration. The main idea consists of giving more importance to the elements of the feature vectors with the lowest variances. These elements very likely represent the main features the user is interested in. Experimental results have confirmed this behavior.

4.4 Search Example

Figure 3 shows an example search in the ImageRover system using relevance feedback. The user was searching for images of sports teams. The user first selected one picture of a soccer team. The search iterated twice, with the user providing relevance feedback by marking another sports team image. The user-selected relevant images appear in the upper row of the image in the figure. The next three rows contain the retrieved 15 nearest neighbors. Images are displayed in similarity rank order, right to left, top to bottom. In this particular example, ImageRover ranked five more sports team photographs as closest to the user-provided examples. As expected, the other returned images share similar color and orientation distributions.

Due to space limitations, it is difficult to include more than one example of image search in the ImageRover system. Readers are therefore invited to visit the ImageRover WWW site to try the system: `http://www.cs.bu.edu/groups/ivc/ImageRover/`.

5 Performance Experiments

In our experiments, the image collection subsystem has had the following performance characteristics. For all single-threaded robots, the average time needed to retrieve an image over the network from a remote server was 82 seconds. This number includes traversing the many text documents in search of images, and it includes waiting politely between document requests. Adherence to robot etiquette protocols dictates that a robot cannot flood a site with requests.

The average CPU time to compute $\mathbf{X}$ on Indigo2 R10000: 12 seconds. Performance is therefore not limited by computation time, but instead on time needed to request and receive image files.

On average, each single-threaded robot can collect 1044 images daily. It is therefore reasonable to expect that a modest fleet of 32 single-threaded robots can collect approximately 1 million images monthly. Multi-threaded robots should achieve significantly greater throughput.

The average space needed to store image thumbnail and $\mathbf{X}$ was 3K bytes. Based on this estimate, 28GB are needed to store an index for ten million images.

In simulation we have tested the performance of the approximate k-nearest neighbors search. Experimental searches were conducted on an SGI Indigo2 R10K with 128MB of main memory, for a data set of size $N = 500,000$ and dimension $k = 78$.

In searches for 20 nearest neighbors with approximation factor $\epsilon = 5.0$, search averaged 1.02 CPU seconds per query in 1000 random trials. With the approximation level set at $\epsilon = 10.0$, nearest neighbor queries averaged 0.11 seconds. For comparison, brute-force nearest neighbors search averaged 1.82 seconds in the same experiment. Thus the approximation yielded a significant speed-up: up to 16 times faster, depending on the specified approximation factor.

The deviation of the approximate nearest neighbors from the true nearest neighbors was also measured. The mean deviation e from true nearest was computed using the following equation:

$$e = \frac{\sum_i^k \parallel \mathbf{q} - \mathbf{a}_i \parallel_2 - \sum_i^k \parallel \mathbf{q} - \mathbf{p}_i \parallel_2}{\sum_i^k \parallel \mathbf{q} - \mathbf{p}_i \parallel_2} \qquad (17)$$

where $\mathbf{q}$ is the query vector, $\mathbf{a}_i$ and $\mathbf{p}_i$ are the approximate and true nearest neighbors respectively.

Based on this equation, the mean deviation from true nearest neighbors was 1.4% for approximation level $\epsilon = 5.0$, and 8.6% for approximation level $\epsilon = 10.0$. In practice, it is expected that a user will set the approximation level high early in a search, decreasing ϵ as the search "closes in" on images of interest.

6 Summary

ImageRover is a content-based image browser for the world wide web. Technical challenges associated with this project are due in part to the staggering scale and unstructured nature of the world wide web, and to the problem of developing fast and effective image indexing methods for fast image database queries.

Images are gathered via a system of distributed robots. This allows efficient and expedient collection, digestion, indexing, and storage of images on the world wide web. ImageRover's distributed robot framework can enable a modest fleet of 32 robots to collect and index over one million images monthly. Image statistics are extracted and stored as vectors in a high-dimensional space.

The system employs dimensionality reduction via a PCA on the original higher-dimensional vector space and then stores the result in an optimized k-d tree. An approximate k-d search algorithm [8], can allow the user to specify an "approximation" level for the nearest neighbors. The user can specify the level of approximation desired, allowing control of the tradeoff between speed and accuracy. In experiments it has been shown that this indexing structure enables database search of 500,000 database records in less than one CPU second on standard Unix workstation.

A query by example interface has been developed and is accessible via a world wide web interface. The system employs a novel relevance feedback algorithm that selects the Minkowski L_m distance metrics appropriate for a particular query. The resulting search tool provides a powerful method for *data exploration* or browsing of WWW images.

Acknowledgments

This work is sponsored in part by through grants from the National Science Foundation (Faculty Early Career Award #IRI-9624168, and CISE Research Infrastructure Awards #CDA-9623865 and #CDA-89202936). Marco La Cascia is sponsored in major part by a doctorate scholarship from the Italian Ministry for University and Scientific Research.

References

[1] S. Sclaroff. World wide web image search engines. position paper presented at The NSF/ARPA Visual Information Management Workshop TR95-016, Boston University, 1995.

[2] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, D. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *IEEE Computer*, pages 23–30, 1995.

[3] A. Gupta. Visual information retrieval technology: A virage perspective. TR 3A, Virage Inc., 177 Bovet Road, Suite 540, San Mateo, CA 94403, 1996.

[4] R. Picard, T. P. Minka, and M. Szummer. Modeling user subjectivity in image libraries. In *Proc. ICIP*, 1996.

[5] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of www client-based traces. TR-95-010, Boston U., 1995.

[6] C. Frankel, M. Swain, and V. Athitsos. Webseer: An image search engine for the world wide web. TR 96-14, U. Chicago, 1996.

[7] D. White and R. Jain. Algorithms and strategies for similarity retrieval. TR VCL-96-101, UCSD, 1996.

[8] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor seaching in fixed dimensions. In *Proc. ACM-SIAM Symp. on Discrete Alg.*, pages 573–582, 1994.

[9] D. A. Forsyth, J. Malik, M. M. Fleck, H. Greenspan, T. Leung, S. Belongie, C. Carson, and C. Bregler. Finding pictures of objects in large collections of images. In *Proc. ECCV 96 Workshop on Object Rep.*, 1996.

[10] A. Pentland, R. Picard, and S. Sclaroff. Photobook: Tools for content-based manipulation of image databases. *IJCV*, 18(3):233–254, 1996.

[11] J. R. Smith and S.-F. Chang. Visualseek: a fully automated content-based image query system. In *Proc. ACM Multimedia '96*, 1996.

[12] E. Ardizzone and M. La Cascia. Automatic video database indexing and retrieval. *Multimedia Tools and Applications*, Jan. 1997.

[13] V. Ogle and Stonebreaker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, 28(2):49–56, 1995.

[14] J. Mao and A. K. Jain. Texture classification and segmentation using multiresolution simultaneous autoregressive models. *Pattern Recognition*, 25(2):173–188, 1992.

[15] R. Picard and T. Minka. Vision texture for annotation. *Multimedia Systems*, 3(3):3–14, 1995.

[16] A. Pentland, B. Moghaddam, T. Starner, O. Oliyide, and M. Turk. View-based and modular eigenspaces for face recognition. In *Proc. CVPR*, pp. 84–91, 1994.

[17] M. Turk and A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86, 1991.

[18] U. Gargi and R. Kasturi. An evaluation of color histogram based methods in video indexing. In *Proc. of Int. Workshop on Image Databases and Multi-media Search*, 1996.

[19] J. Hafner, Harpreet Sawney, W. Equitz, M. Flickner, and W. Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE T-PAMI*, 1(7):729–736, 1995.

[20] W. Freeman and E. H. Adelson. The Design and Use of Steerable Filters. *IEEE T-PAMI*, 13(9):891–906, 1991.

[21] M. Gorkani and R. Picard. Texture orientation for sorting photos at a glance. In *Proc. IEEE CVPR*, 1994.

[22] J. H. Friedman, J. H. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. on Math. Software*, 3(3):209–226, 1977.

[23] R. O. Duda and P. E. Hart. *Pattern Recognition and Scene Analysis*. John Wiley, New York, 1973.