Outdoors Augmented Reality on Mobile Phone using Loxel-Based Visual Feature Organization

Gabriel Takacs
Stanford University
gtakacs@stanford.edu

Vijay Chandrasekhar
Stanford University
vijayc@stanford.edu

Natasha Gelfand
Nokia Reseach Center
natasha.gelfand@nokia.com

Yingen Xiong
Nokia Reseach Center
yingen.xiong@nokia.com

Wei-Chao Chen
Nokia Reseach Center
wei-chao.chen@nokia.com

Thanos Bismpigiannis
Stanford University
thanosb@stanford.edu

Radek Grzeszczuk
Nokia Reseach Center
radek.grzeszczuk@nokia.com

Kari Pulli
Nokia Reseach Center
kari.pulli@nokia.com

Bernd Girod
Stanford University
bgirod@stanford.edu

ABSTRACT

We have built an outdoors augmented reality system for mobile phones that matches camera-phone images against a large database of location-tagged images using a robust image retrieval algorithm. We avoid network latency by implementing the algorithm on the phone and deliver excellent performance by adapting a state-of-the-art image retrieval algorithm based on robust local descriptors. Matching is performed against a database of highly relevant features, which is continuously updated to reflect changes in the environment. We achieve fast updates and scalability by pruning of irrelevant features based on proximity to the user. By compressing and incrementally updating the features stored on the phone we make the system amenable to low-bandwidth wireless connections. We demonstrate system robustness on a dataset of location-tagged images and show a smart-phone implementation that achieves a high image matching rate while operating in near real-time.

Categories and Subject Descriptors

H.3.3 [**Information Technology and Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*

General Terms

Algorithms, Design, Performance

Keywords

local search, photo collections, geo-referenced photos, image search, mobile computing, augmented reality

1. INTRODUCTION

High-end mobile phones have developed into capable computational devices equipped with high-quality color displays, high-resolution digital cameras, and real-time hardware-accelerated 3D graphics. They can exchange information over broadband data connections, and sense location using GPS. This enables a new class

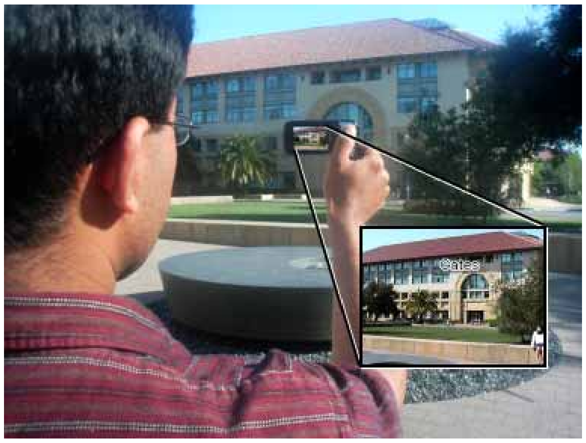

Figure 1: A snapshot of the outdoors augmented reality system being used. The system augments the viewfinder with information about the objects it recognizes in the image taken with a phone camera.

of augmented reality applications which use the phone camera to initiate search queries about objects in visual proximity to the user. Pointing with a camera provides a natural way of indicating one's interest and browsing information available at a particular location. Once the system recognizes the user's target it can augment the viewfinder with graphics and hyper-links that provide further information (the menu or customer ratings of a restaurant) or services (reserve a table and invite friends for a dinner). This paper presents the technologies we have developed for supporting these kinds of point-and-find applications.

1.1 Prior Work in Image Retrieval

Our work represents a special category of content-based image retrieval (CBIR) [1]. We want to recognize real-world images from a set of categories (buildings, landmarks, logos) but we want to do it under a variety of viewing and lighting conditions. We want the algorithms to work in the presence of transient clutter in the foreground, and changes in appearance. Object categorization algorithms [2, 3] typically require an expensive training step and are less discriminative among similar looking object categories than the algorithms based on robust local descriptors [4] on which we base our system.

Research on image search services using mobile devices includes work by Zhou *et al.* [5], which focuses on identifying matching robust local features in multiple query images. Only features that appear in many query images are used for querying a database. In this work, capturing a sequence of query images is done using a mobile phone, but the processing is done on a server.

Research on robust local descriptors is very active. Recent examples include, but are not limited to, SIFT by Lowe *et al.* [4], GLOH by Mikolajczyk and Schmid [6], and SURF by Bay *et al.* [7]. For our application, SURF features have the most favorable computational characteristics. Our image retrieval algorithm adapts the framework of Lowe *et al.* [4] to work with stringent bandwidth, memory and computational requirements. The modifications include clustering and pruning of features based on relevance ranking, and feature descriptor compression.

Recent work in object-based image retrieval uses a vocabulary of "visual words" to search for similar images in very large image collections [8]. In this formulation, a bag of visual words is used as an index during query and retrieval. Our application and algorithms also focus on fast retrieval from a database of features containing noisy data. We focus on minimizing the query time and exploiting the spatial structure of visual words within the image to develop techniques that are robust to noise.

1.2 Prior Work in Mobile Augmented Reality

A recent demonstration of an outdoor mobile augmented reality application running on a cell phone is Nokia's MARA project by Kähäri and Murphy [9]. The system does not perform any image analysis, instead it uses an external GPS for localization and an inertial sensor to provide orientation. PhoneGuide [10] is one of the first object recognition systems performing the computation on a mobile phone, instead of sending the images to a remote server. The system employs a neural network trained to recognize normalized color features and is used as a museum guide. Seifer *et al.* [11] use a mobile system based on a hand-held device, GPS sensor, and a camera for roadside sign detection and inventory. Their algorithm was efficient enough to ensure good quality results in mobile settings.

In the context of augmented reality, Fritz *et al.* [12] use a modified version of the SIFT algorithm for object detection and recognition in a relatively small database of mobile phone imagery of urban environments. The system uses a client-server architecture, where a mobile phone client captures an image of an urban environment and sends it to the server for analysis. The SURF algorithm has been used successfully in a variety of applications, including an interactive museum guide [13]. Local descriptors have also been used for tracking. Skrypnyk and Lowe [14] use the SIFT features for recognition, tracking, and virtual object placement. Camera tracking is done by extracting SIFT features from a video frame, matching them against features in a database, and using the correspondences to compute the camera pose. Takacs *et al.* [15] track SURF features using video coder motion vectors for mobile augmented reality applications.

Yeh *et al.* [16] propose a system for determining a user's location from a mobile device via image matching. The authors first build a "bootstrap database" of images of landmarks and train a CBIR algorithm on it. Since the images in the bootstrap database are tagged with keywords, when a query image is matched against the bootstrap database, the associated keywords can be used to find more textually related images through a web search. Finally, the CBIR algorithm is applied to the images returned from the web search to produce only those images that are visually relevant.

1.3 Contributions and Overview

In our own work, we have developed an outdoors augmented reality system for matching images taken with a GPS-equipped camera-phone against a database of location-tagged images. The system then provides the user links or services associated with the recognized object. If no match is found, the user has an option of associating the query image with a label from a list of nearby points of interest and submitting the data to the server. The system is fully implemented on the mobile device, and runs at close to real-time while maintaining excellent recognition performance.

To ensure that our image matching algorithm is robust against variations in illumination, viewpoint, and scale, we have adapted the SURF algorithm [7] to run on the mobile phone. We have optimized the performance and the memory usage of this already efficient algorithm. Our implementation runs an average of 35% faster than the original and uses only half as much memory. The matching quality of our implementation is comparable to that of the original.

We wanted the image matching to be done directly on a mobile client for several reasons. First, this significantly reduces the system latency. Second, distributing the computation among the users provides for better system scalability. Third, if a user takes several images at a single location then we save on bandwidth, since the phone has the data cached locally. Finally, we are currently extending the system described in this paper to work with real-time tracking [15], for which the option of uploading images to the server is clearly impractical.

This work deals with challenging data. Our datasets are captured in busy, outdoors settings. The images are full of occlusion by trees, cars, pedestrians, etc. In many cases, only a small percentage of image corresponds to the object of interest. The low resolution images are captured using low-end cameras with poor optics and noisy sensors. To maintain as high a recognition rate as possible, it is critical to limit the number of possible choices.

Using location information to limit irrelevant data has been critical to our system's performance. We do so by quantizing the user's position and only considering data from nearby location cells, or *loxels*. We develop a method for incrementally updating the local database of features on the handset as the user changes location. This enables us to maintain a large pool of relevant features on the client while keeping the bandwidth low. Another key aspect of the quantization of space into loxels is that it allows our system to scale to arbitrarily sized databases. This is because only a fixed, finite number of points of interest can occur within a loxel.

To further reduce bandwidth and memory usage on the client, we have developed a method to cluster features that appear in several images into *meta-features* and prune the others by selecting a subset of the most relevant features for a given geographic location. Feature clustering and pruning also help to limit the amount of data on the server by eliminating redundant information. We show that this operation does not degrade the quality of the image matching results.

After feature clustering and pruning, we achieve further savings in bandwidth and memory usage by improving the coding efficiency of the feature descriptors. By studying the entropy characteristics of the SURF descriptor, we devise a compression scheme that encodes the descriptors $7\times$ more efficiently than without compression. This is done without sacrificing matching performance.

Figure 2 gives an overview of the system which is divided into two major components, a mobile device and a server. These components communicate over a wireless network. The server groups images by location into loxels. Features are then extracted from all images in a loxel. These features are matched against each other, followed by a geometric consistency check which eliminates

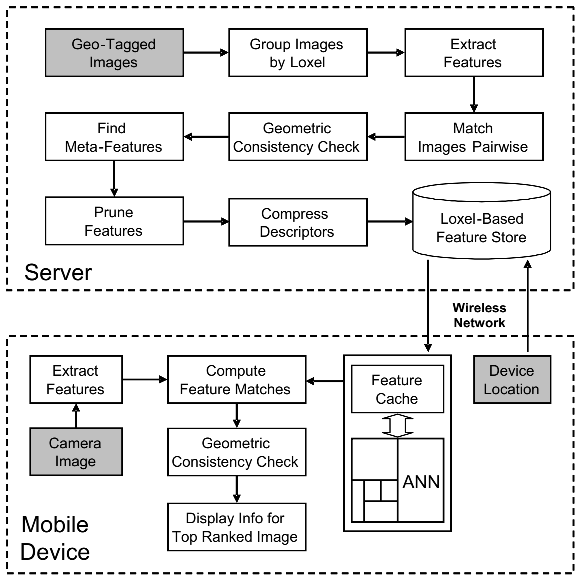

Figure 2: System block diagram. The system is divided into two major components, a mobile device and a server, which communicate over a wireless network.

incorrect matches. The server then groups the matching features into meta-features and prunes the remaining features to reduce the amount of data that needs to be stored on the server and sent to the client. This compact description of the loxel is then compressed and written into a *loxel file* and stored in a database for retrieval by the mobile device.

When activated, the phone continuously sends information to the server about its location. When the phone changes to a new location cell it receives any loxel files that have not been cached. Features from multiple loxel files are inserted into a single *k*d-tree for fast nearest-neighbor computation. The client matches features extracted from a query photo against the features in the loxel files, and performs a geometric consistency check to prevent false positive matches. Information or services for the highest ranked match are then displayed to the device's screen.

After the user is presented with the search results, data can be uploaded to the server. In this way the system grows and learns from user feedback. As seasons and buildings slowly change over time the system's database will adapt. By uploading the data after the matching, the system can maintain a low latency.

2. ROBUST IMAGE MATCHING

Our retrieval algorithm is based on robust local image descriptors. We compared the SIFT [4] and the SURF [7] algorithms for computing visual features and concluded that the SURF algorithm exhibits similar recognition results on our datasets while using less RAM and being significantly faster. Hence, we chose the SURF algorithm for implementation on a mobile phone.

2.1 Image Retrieval Pipeline

The algorithm assumes the presence of a database of labeled images. The goal is to assign labels to a test image by matching it with images in the database. Our algorithm starts by extracting SURF features from all labeled images and inserting them into a shared *k*d-tree that allows for fast approximate nearest neighbor (ANN) queries in high-dimensional spaces [17]. Next, the algorithm extracts features from the test image and matches them against the features in the ANN data structure using the *multiple ratio test* described below. The database images are then ranked based on the number of features that match a query feature. For the top ranked images, we eliminate matching features that do not pass the *geometric consistency check* and re-rank the images based on the new feature-match counts.

2.1.1 Multiple Ratio Test

We are trying to match the features of a test image I_t to the features from a set of images $\mathbf{I}_s = \{I_1, \ldots, I_m\}$. First, we input all features from $\mathbf{I}_s$ into an ANN data structure. For each feature F_t^i in I_t, we compute an ordered list of the $M+1$ nearest neighbors where M is the maximum number of images in $\mathbf{I}_s$ with the same label. Let $\mathbf{N}_t^i = \{N_1, \ldots, N_{M+1}\}$ be the list of neighbors and let $\mathbf{D}_t^i = \{D_1, \ldots, D_{M+1}\}$ be the list of their distances to F_t^i. We decide that N_j matches F_t^i if the ratio $r_j = D_j / D_{M+1} \leq \alpha$, for $j = 1, \ldots, M$. We refer to this method of selecting feature matches as a *multiple ratio test*.

2.1.2 Geometric Consistency Check

The multiple ratio test performs well, but occasionally produces incorrect feature pairings. A standard way of removing these outliers is to perform a geometric consistency check that tests if a group of features in one image can be transformed into the matching features in another image through a simple geometric operation [4]. We experimented with three different models, affine, homography, and epipolar. Of the three, the epipolar constraint most accurately captures possible feature transformations. However, epipolar model estimation requires more inliers and less noise than the other two models [4]. Therefore, we have chosen to use an affine model, which in practice performs as well as a homography [15] and is easier to compute.

2.2 Performance Evaluation

2.2.1 Experimental Datasets

We have used camera phones to collect a dataset of geotagged images for our experiments. We wanted to test if the quality of images acquired with these devices is sufficient for robust operation of the system. We also wanted to see how effective the camera phones are for collecting large image-based datasets. The data were acquired over multiple shooting sessions by several individuals, guaranteeing a diversity of views, lighting conditions, and occlusions. We labeled each photograph with a list of visible points of interest (POIs). Our dataset contains ∼2500 images from Stanford University, a shopping mall, and a commercial street. Note that because our system uses location information it is not the size, but the spatial density of the dataset that matters.

We collected test images separately, about 3-6 months after acquiring the original dataset. This was done to ensure that the test images are sufficiently different from the images in the dataset. The images were acquired using a variety of camera-phones, such as the Nokia N93, N95 and N5700. There are 72, 14, and 37 test images for Stanford, the shopping mall, and the commercial street, respectively. Additionally, to test the robustness of the system we have added 28 test images of objects that are not in our database, such as cars, trees, streets, etc. If one of these images produces a match then it is known to be erroneous. In total, there are 151 test images. [1]

[1]The readers can access, experiment with, and download our databases through a web-based interface available from this URI: `http://mar1.tnt.nokiaip.net/marwebapi/benchmarks`.

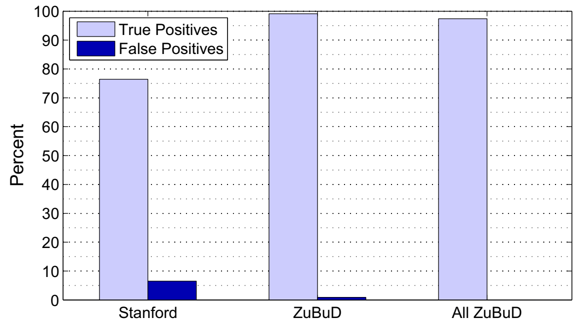

Figure 3: Image recognition rate for our dataset compared to *ZuBuD*. Good performance is indicated by a high true positive rate and low false positive rate. Results for *All ZuBuD* are for the *ZuBuD* dataset without geotagging. These results demonstrate that our dataset is significantly more challenging than *ZuBuD*.

Additionally, for comparison to other published algorithms, we have included results for the Zürich Building Database (*ZuBuD*) [18]. This dataset has $\sim$1000 photographs of buildings around Zürich, and 115 separate test images. Since the publisher did not geotag the database, we use this dataset by with synthetic geotags. The geotags are randomly placed with a density corresponding to 5 POIs per loxel.

For each test dataset we report a percentage of correct matches (true positives) and incorrect matches (false positives). A match is considered correct if the top ranked image has the same label as the query image. Since some of the images in the database have multiple labels, and some images are entered into the database with different labels, we decide that a match is correct if at least one of the labels of matching images is the same as the label of the test image.

2.2.2 *Experimental Results*

To tune the performance of our image retrieval pipeline we have performed experiments with the geometric consistency check and image resolution. We vary these parameters while querying with our test sets and evaluating the resulting true-positive and false-positive rates.

Since smaller images lead to faster processing, we desire the smallest image possible without significant degredation in recognition performance. We have found images of resolution 640$\times$480 to provide robust matching with sufficient speed. Additionally, this is the smallest native resolution of many cameras. We show in Section 4.3.1 that at this image resolution we can get the system running at close to real-time on a mobile phone and that we can meet the stringent bandwidth limits set by data transfer rates over wireless networks.

As discussed in Section 2.1.2, we have chosen an affine consistency check. However, the parameters of this check can still be varied to trade-off true and false positive rates. By increasing the minimum number of good matches, or inliers, we are able to decrease the false positive rate. However, the associated true positive rate also decreases as some correct, but week matches are discarded. We choose a minimum of four matching features and require the point-wise model error to be within 20 pixels for a match to be considered correct.

As seen in Figure 3, these parameters provide an average match rate of 77% true positives with 7% false positives. The matching performance depends on the difficulty of the dataset with our data yielding the lowest true positive rates and the highest false positive rates. The cleaner *ZuBuD* datasets correctly identify almost every query with very few false positives. The *All ZuBuD* results are with the entire dataset in a single loxel. The high performance of this dataset shows that our image retrieval algorithm scales to very dense datasets. Figure 4 shows sample image matching results of our test images against our database.

3. LOXEL-BASED ORGANIZATION

In this section we discuss how to adapt our local robust descriptors algorithms so that they can run on mobile phones with limited computational resources over wireless networks with low bandwidth data connections. First, we examine using location information to reduce the amount of data that need to be transferred to and processed by the phone. Next, we present a new feature clustering algorithm which reduces the size of the feature set without degrading the recognition results. We also show how to use the natural ranking of features produced by the clustering algorithm to prune unimportant features. We end by describing our feature compression scheme, which uses a combination of lossy and lossless compression of feature descriptors to obtain additional bandwidth and memory savings.

3.1 Location Grid

A simple but effective method of reducing the size of the feature set is to consider only features that originate from objects directly visible from the current location. We estimate the visibility by laying down a uniform grid over a geographic area of interest. We refer to a grid cell as a location cell, or simply *loxel*,[2] and we will refer to the area visible from a given loxel as a visibility kernel, or simply *kernel*. A kernel will typically span multiple adjacent loxels. Each loxel $\mathbf{L}^i_j$ has an associated kernel $\mathbf{K}^i_j$. Note that while the loxels cover disjoint areas, the kernel areas overlap. This affects the way we represent and transmit data to avoid redundancy.

To determine a kernel size that yields the best recognition rate, we have experimented with kernels of size 1$\times$1, 3$\times$3, 5$\times$5, 7$\times$7, etc. For our dataset, a kernel of size 1$\times$1 is too small to gaurantee that visible POIs fall within the kernel. A larger kernel of size 3$\times$3 yields the best results. Kernel sizes larger than 3$\times$3 include POIs which are not visible, thus cluttering the search space and decreasing the match rate. Even where POIs are further apart, a kernel of size 3$\times$3 performs no worse than larger kernels.

Having determined the best kernel size, we can estimate how many features are contained in each kernel to estimate the size of the matching problem to be performed on the mobile device. For the our dataset, the maximum is 40K and the mean is 8K. For practical bandwidth and memory reasons, we currently allow a maximum of 10K features per kernel of size 3$\times$3. Since unprocessed kernels contain more than 10K features, we must compress the data to stay below the limit.

3.2 Feature Clustering and Pruning

Since we envision an environment where many users contribute images using a variety of camera-phones, we purposely collected our datasets in a non-methodical fashion. As a result, the density of images varies greatly within each loxel and for each POI. Some loxels may contain many tens of images, while other loxels contain very few image. This makes sending the correct feature descriptors to the phone challenging. Additionally, as users contribute images to the database the amount of data must be kept small and highly relevant.

[2]Throughout the paper we use a loxel size of 30m$\times$30m.

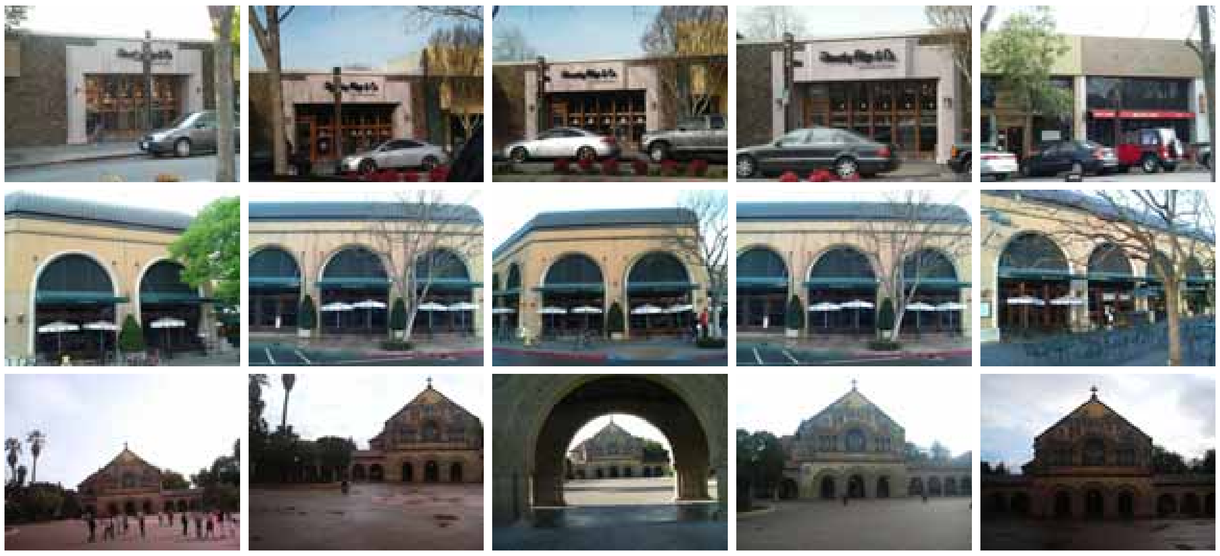

Figure 4: Examples of image matches. For each row, we show the test image on the left, followed by the 4 top ranked images. These results demonstrate the difficulty of the dataset and the robustness of matching.

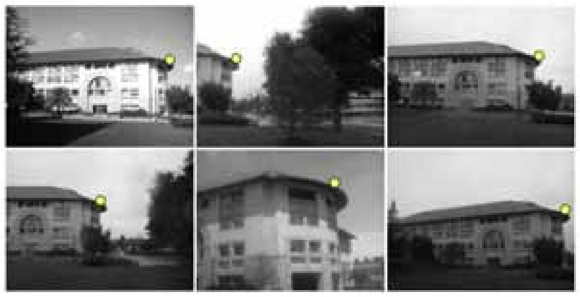

Figure 5: An example of a meta-feature, a feature point which occurs in 6 different views of the same point of interest in a loxel.

To reduce the bandwidth and memory requirements of our application we use a carefully selected subset of all features for each loxel. We require that this subset of features has a small amount of data, high quality features, and a wide coverage. For practical reasons, we currently set a limit of 10K features per kernel. Hence, since we typically use a kernel of size 3×3, this gives us a budget of $\sim$1K features per loxel. For this reason, only high quality features that are more likely to match potential query features are sent to the phone. For example, we should not be sending features that come from transient occluders such as cars and people. For coverage, The features budgeted for a loxel must cover all of its POIs as equally as possible.

3.2.1 Feature Clustering

As the first step, we reduce the number of features per loxel by clustering features. If a loxel has many images, it is very likely that some of them cover the same POI from similar views. In this case, the corresponding feature descriptors would be similar. The clustering algorithm tries to identify groups of features that correspond to the same physical location portrayed in different images. Once we find such features, we can cluster them into a single *meta-feature*. An example of a meta-feature seen in several images is shown in Fig. 5.

The meta-feature computation is done on the server using four steps. First, for each image in the loxel, find all matching images using the ratio test described in Section 2.1.1. Second, build a graph, G, on the features in the loxel. The nodes in the graph are the features in the loxel, there is an edge between any two features that matched in the previous step. In ideal conditions, groups of features coming from the same world point would form cliques in the graph, G. However, due to noise in the descriptors, viewpoint variation, and the non-transitivity of the ratio test, full cliques do not usually form. Therefore, in the next step we extract connected components of G as our feature clusters. Finally, each set of features, $F_1^i \dots F_k^i$, forming the i^{th} connected component becomes a *meta-feature*, whose descriptor value is obtained by averaging and renormalizing the descriptors of $F_1^i \dots F_k^i$.

This algorithm removes redundancy from the features in each loxel. In addition, it produces a confidence weight for each meta-feature, which is the size of its corresponding cluster. Features with higher weights have been seen in more images, and therefore are good candidates to be sent to the phone, since they are likely to be seen again. Meanwhile, *singleton* features, from clusters of size 1, often come from transient foreground occluders such as cars.

Image matching with meta-features requires a modification to the multiple-ratio test described in Section 2.1.1. Using meta-features, the $M+1$ nearest neighbor is an over-estimate of the first outlier. To determine the correct outlier we compute the cumulative sum of the meta-features' cluster size. The first meta-feature whose cumulative sum exceeds M is considered an outlier.

The addition of feature clustering enhances the matching performance of the system, since the averaged descriptor values are more robust to noise, and a match against a meta-feature is automatically counted as a match against several images. However, the compression rate achieved by feature clustering alone is relatively low, about $10-20\%$. Even for dense loxels, most of the features are singletons. Thus meta-features alone are not sufficient to reduce the number of features. In addition, if some POI has only a single image in the database, all of its features are singletons and would be thrown out by this method. Thus, we propose a feature budgeting method that ensures that each POI in the loxel has a sufficient number of features representing it.

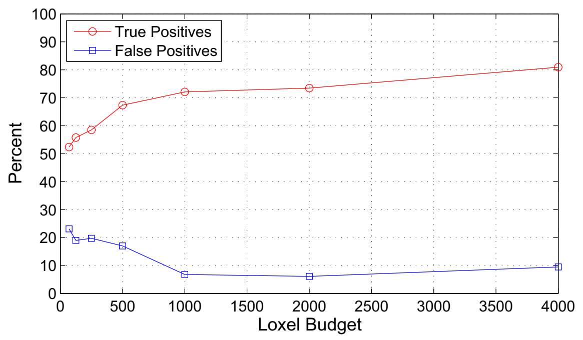

Figure 6: Matching performance as a function of feature clustering and pruning. The matching rate is relativly unchanged until the budget drops below 1000 features.

3.2.2 Feature Budgeting

Feature budgeting selects a pre-determined number of features, N, for each loxel, such that all POIs are covered by roughly equal number of features. The best features are then selected for each POI. This motivates the need for ranking features and images.

Image ranking: All images in a loxel need to be ranked by their content and the features that they contain. The rank of an image is given as the number of meta-features weighted by each meta-feature's cluster size. Therefore, images which have many features that belong to large clusters are moved up in the ranked list.

Feature allocation: We want to make sure each POI in a loxel is equally well covered by features. Therefore, we allocate to each POI a budget of N/P feature points, where P is the number of POIs in the given loxel. The allocation of features to POIs works as follows.

First, we allocate meta-features by going through the images in order of decreasing rank. As long as the feature budget is not reached, we add all meta-features from the next image to the set of allocated features. Next, the meta-features within each image are ranked and picked according to decreasing cluster size. Finally, if we have not reached the feature budget for the POI and all meta-features have been picked, we make a second pass through the ranked images adding all singleton features from each subsequent image until the feature budget is reached.

In both budgeting passes, we set a minimum threshold of the number of features per image we must pick (currently set to 30). That is, either we pick at least 30 point per image, or none at all. This ensures that each image has enough features picked for the success of the geometric consistency check.

The budgeting algorithm maintains all the requirements outlined in the beginning of the section. Because the images are ranked by the number of meta-features, and the meta-features are ranked by their size, features that are present in many images are picked first.

This method scales well and becomes increasingly robust as more images are added to the loxel, since we are able to better distinguish transient foreground objects and discard their features, focusing our attention on features belonging to objects of interest. As the number of images in a loxel becomes very large, new images need only be matched against highly ranked images.

3.2.3 Feature Clustering Results

Figure 6 illustrates how feature clustering and pruning impacts the recognition performance. We see that clustering features into meta-features and pruning slightly degrades the match rate but can provide a $4\times$ reduction in data.

3.3 Feature Compression

Further savings in memory and bandwidth are possible by considering the information redundancy within each feature descriptor. Traditionally, this has been done by applying dimensionality reduction techniques to the space of feature descriptors [19]. Instead, we take an information theoretic approach and show that the coding efficiency of the SURF descriptors can be significantly improved. By studying the entropy characteristics of the SURF descriptor we were able to produce a very efficient encoding for the descriptor that is $7\times$ smaller than the original at the same feature matching performance. (We are able to compress a 256 byte SURF descriptor to 36.8 bytes.) We first use lossy compression to discard as many bits as possible through quantization and then we losslessly compress the remaining data with entropy coding.

3.4 Data Organization and Transmission

Our system operates as a client-server architecture, where a client indicates its current location to a server by specifying that it is in loxel $\mathbf{L}_j^i$ and the server ensures that the client has all the relevant data for kernel $\mathbf{K}_j^i$. We organize the data for neighboring kernels so that duplication of information is minimized by associating all the feature data and all the point of interest data with the loxels. Only the nearest-neighbor data structure is associated with the kernels, but since they are very fast to build, we simply do that on the phone. See the timing results in Section 4.

Loxel-centric organization of data allows for incremental updates of the working set of features stored on the client. If a client moves from loxel $\mathbf{L}_j^i$ to loxel $\mathbf{L}_j^{i+1}$, the server only needs to send to the client the data associated with the three new loxels from column $i+2$. The client then rebuilds a new approximate nearest-neighbor data structure for kernel $\mathbf{L}_j^{i+1}$.

We use four data structures for a single loxel. A *feature block* stores the feature data such as its descriptor, weight, and a pointer to a list of feature identifiers. For a singleton feature, the list has one entry. For a meta-feature, this list has a length equal to the weight of the meta-feature. A *feature identifier* has a pair of records, one specifies the image to which the feature belongs, the other stores the image coordinates of the feature. The coordinates are used for the geometric consistency check. An *image label* database stores the list of labels associated with individualimages.

A *POI database* stores the information associated with the points of interest inside the loxel, such as its name, address, URL, etc. This database is indexed with the image labels.

At any time, the phone client will have these data structures available for all the loxels of the kernel it is in. The approximate nearest-neighbor (ANN) data structure is the only data structure accessing the information from the data structures of the different loxels. The ANN structure does not differentiate between the meta-features and the regular features in any way.

For a typical query, a feature match is computed by looking up the query feature in the ANN data structure. The information about the feature is then identified by following a link to the feature block database. The identifiers of the images associated with that feature are extracted from the feature identifier array and the feature counts for all these images are incremented.

The image information database is used to look up the labels for a given image. Additionally, if we must conduct a geometric consistency check, we can use the image information database to look up the image coordinates of a feature associated with a given image.

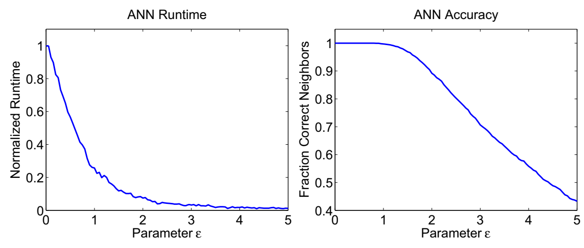

Figure 7: Impact of search accuracy on the quality and the speed of feature matching. The time is measured in milliseconds and the accuracy is measured as a fraction of correctly label points.

4. MOBILE PHONE IMPLEMENTATION

In this section, we present our implementation of the SURF algorithm and its adaptation to the mobile phone. Next, we discuss the impact that accuracy has on the speed of the nearest-neighbor search and show that we can achieve an order of magnitude speedup with minimal impact on matching accuracy. Finally, we discuss the details of the phone implementation of the image matching pipeline. We study the performance, memory use, and bandwidth consumption on the phone.

4.1 SURF Implementation

Our mobile phone implementation of the SURF algorithm [20] focuses on using memory efficiently and on reducing computation. The algorithm consists of three major steps: interest-point extraction, repeatable angle computation, and descriptor computation. In the first two steps of the algorithm, we use the integral image [21] for efficient Haar transform computation as described in the original paper. For memory efficiency, we only store either the original image or the integral image, and convert between them as needed. This is possible because the conversion takes minimal time compared to the rest of the algorithm. After tuning, our implementation runs an average of 35% faster than the original and uses only half as much memory with comparable descriptor quality.

4.2 Image Query Time

The accuracy of the ANN search is controlled by a parameter ε. If $\varepsilon = 0$, the returned result is identical to the exact result, otherwise it is at most $(1+\varepsilon)$-times the distance to the closest point [17].

We measure the query time for different values of ε using 10 independent tests sets, each with ∼300 query points and an ANN tree with ∼40K features. Fig. 7 shows an average of the 10 trials. The left graph shows the impact of ε on the query speed, and the right graph shows the percentage of feature match errors, with respect to an exact search ($\varepsilon = 0$).

For $\varepsilon = 1.5$ we get an order of magnitude improvement in performance and a minimal decrease in accuracy. We verified that this result is consistent across different datasets and at different problem sizes. Additionally, we achieve an additional 4× speedup by vectorizing the ANN algorithm.

4.3 Performance Analysis

4.3.1 Execution Profile

We have implemented our system on a Nokia N95 smart-phone. Here we report the timing of the algorithm pipeline stages executing on a 332MHz ARM11 CPU with a VFP unit capable of issuing 4 single-precision floating-point instructions in one cycle.

We timed our system using the settings discussed in the paper: image resolution of 640×480, 250 query features, and 7000 features in the ANN data structures. For feature matching, we use the multiple ratio test with $\alpha = 0.8$ and the accuracy of the ANN search set to $\varepsilon = 1.5$. The ANN query routine has been vectorized. This gives the following execution profile: SURF computation—2.4 sec, ANN query—0.3 sec, geometric consistency check—0.1 sec. The total runtime for image matching on the phone is 2.8 seconds.

In an application where real-time performance is critical, we can use images of resolution 320×240 as queries. This yields ∼ 4× reduction in the time to compute features and feature matches. The system then runs at about 1 frame per second while maintaining a very high image retrieval quality.

The time to build the ANN tree is 0.5 sec. Since we need to rebuild the ANN tree only once per loxel change, it is much more efficient to do this task on the phone. Thus we never send the ANN data structures from the server, instead we build them on the phone.

4.3.2 Memory Usage

In the current implementation, the feature descriptors get decompressed right when they arrive at the client and before they are inserted into the ANN data structure. Assuming an ANN tree with 7000 descriptors, this translates to 7K×256B=1.8MB. In the future, we plan to modify the ANN tree to operate on the quantized descriptors. This would reduce our memory requirement for this data structure by 4×.

Our implementation of the SURF algorithm uses memory very efficiently. At any given time, it needs at most 1.75× the original image size. For a 640×480 grayscale image, we can represent its intensity using 307KB. Hence, the most memory we will ever use for the SURF feature calculation is 538KB.

4.3.3 Bandwidth Consumption

We would like our system to operate within the bandwidth constraints of existing wireless networks. A typical 3G network supports download speeds of up to ∼2 Mbps and upload speeds of ∼150 kbps. More than 95% of the data sent to the phone is devoted to feature descriptors. We can ignore the nearest-neighbor data structures since it is more efficient to build these on the client than to transfer them from the server. Hence the communication of feature descriptors to the client dominates the bandwidth consumption of our system.

Let's assume a kernel of size 3×3 loxels, a loxel of size 30m × 30m, and an average time to traverse a loxel (at walking speed) of 20 sec. The size of the feature descriptor after compression is 36.8 bytes and we have a maximum of 1000 features per loxel after feature budgeting. At each loxel change, the data for 3 new loxels need to be transmitted. This corresponds to 110 KB of data, which will take negligible time to download over a 3G network. For all practical purposes, a person using the system while walking will have the data on the phone updated right after entering a new loxel.

To compare the latency, we have implemented the same system in both a *client-side* and *server-side* configuration. The *client-side* configuration does feature computation and image matching directly on the phone, as previously discussed. The *server-side* configuration uploads the query image to the server and does all the computation there. For the this configuration, the upload time dominates the computation time. Using typical 75 KB JPEG images over a 3G wireless network, our experiments show that it takes 5-6 seconds for the *server-side* configuration and 2-3 seconds for the *client-side* configuration. The latency advantage of the *client-side* configuration will continue to grow as the speed of mobile devices continues to grow faster than the upload rate of wireless networks. Additionally, as we move towards video frame-rates transmitting frames over the network will become infeasible.

5. CONCLUSION

Simply pointing your camera at objects around you and finding information about them provides a very intuitive user interface for accessing information and services around your current location. Such a system provides a bridge between the digital and physical worlds.

In this paper we have studied the image processing aspects of such a system, and how the image and location database should be organized on the server, and updated on the mobile client. A modified SURF is a good choice for robust image recognition that can be implemented on modern smart-phones. Our database organization and update mechanism can work with the constraints of the current data bandwidth. SURF and other similar systems yield many features per image, which would overwhelm a naïve implementation. We have addressed that by pruning features that are likely to be noise, thus concentrating on features that are repeatedly found in several images, and aggressively compressing the features without degrading the recognition rates. By including user feedback, our system can adapt to changing environments.

We have tested the system with real data, obtained by camera phones with a GPS sensor. The data were collected from several locations, by several people, at several times of the year and day. The recognition system has been implemented and tested on mobile phones. Our preliminary results indicate the system works fast enough for a large set of applications, such as a tourist guide for pedestrians.

6. ACKNOWLEDGMENTS

The authors would like to thank Ramakrishna Vedantham and Harlan Hile for designing and developing a web-based interface to the image database, Jiang Gao and Philipp Schloter for useful discussions, and John Shen for support and encouragement.

7. REFERENCES

[1] R. Datta, J. Li, and J. Z. Wang, "Content-based image retrieval: approaches and trends of the new age," in *MIR '05: Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*. New York, NY, USA: ACM, 2005, pp. 253–262.

[2] R. Fergus, P. Perona, and A. Zisserman, "Object class recognition by unsupervised scale-invariant learning," in *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 02. IEEE Computer Society, 2003, p. 264.

[3] A. Ferencz, E. Learned-Miller, and J. Malik, "Learning Hyper-Features for Visual Identification," in *Neural Information Processing Systems*, vol. 18, 2004.

[4] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[5] Y. Zhou, X. Fan, X. Xie, Y. Gong, and W.-Y. Ma, "Inquiring of the Sights from the Web via Camera Mobiles," *2006 IEEE International Conference on Multimedia and Expo*, pp. 661–664, July 2006.

[6] K. Mikolajczyk and C. Schmid, "Performance Evaluation of Local Descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 10, pp. 1615–1630, 2005.

[7] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded Up Robust Features," in *ECCV (1)*, 2006, pp. 404–417.

[8] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object Retrieval with Large Vocabularies and Fast Spatial Matching," in *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[9] K. Greene, "Hyperlinking Reality via Phones," MIT Technology Review, Nov/Dec 2006.

[10] E. Bruns and O. Bimber, "Adaptive Training of Video Sets for Image Recognition on Mobile Phones," *Journal of Personal and Ubiquitous Computing*, 2008, to appear.

[11] C. Seifert, L. Paletta, A. Jeitler, E. Hödl, J.-P. Andreu, P. M. Luley, and A. Almer, "Visual Object Detection for Mobile Road Sign Inventory," in *Proc. of Mobile Human-Computer Interaction - Mobile HCI 2004, 6th International Symposium*, 2004, pp. 491–495.

[12] G. Fritz, C. Seifert, and L. Paletta, "A Mobile Vision System for Urban Detection with Informative Local Descriptors," in *ICVS '06: Proceedings of the Fourth IEEE International Conference on Computer Vision Systems*, 2006, p. 30.

[13] H. Bay, B. Fasel, and L. V. Gool, "Interactive Museum Guide: Fast and Robust Recognition of Museum Objects," in *Proceedings of the First International Workshop on Mobile Vision*, May 2006.

[14] I. Skrypnyk and D. G. Lowe, "Scene Modelling, Recognition and Tracking with Invariant Image Features," in *ISMAR '04: Proceedings of the Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, 2004, pp. 110–119.

[15] G. Takacs, V. Chandrasekhar, B. Girod, and R. Grzeszczuk, "Feature Tracking for Mobile Augmented Reality Using Video Coder Motion Vectors," in *ISMAR '07: Proceedings of the Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, 2007.

[16] T. Yeh, K. Tollmar, and T. Darrell, "Searching the Web with Mobile Images for Location Recognition," in *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2. IEEE Computer Society, 2004, pp. 76–81.

[17] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions," *Journal of the ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[18] L. V. G. H.Shao, T. Svoboda, "Zubud-Zürich buildings database for image based recognition." ETH Zürich, Tech. Rep. 260, 2003.

[19] Y. Ke and R. Sukthankar, "PCA-SIFT: A More Distinctive Representation for Local Image Descriptors," in *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 02. IEEE Computer Society, 2004, pp. 506–513.

[20] W.-C. Chen, Y. Xiong, J. Gao, N. Gelfand, and R. Grzeszczuk, "Efficient Extraction of Robust Image Features on Mobile Devices," in *ISMAR '07: Proceedings of the Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, 2007.

[21] P. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2001, pp. I:511–518.