Structuring a Sharded Image Retrieval Database

Eric Liang and Avideh Zakhor

Department of Electrical Engineering and Computer Science, University of California, Berkeley
{ekhliang, avz}@eecs.berkeley.edu

ABSTRACT

In previous work we described an approach to localization based in image retrieval. Specifically, we assume coarse localization based on GPS or cell tower and refine it by matching a user generated image query to a geotagged image database. We partition the image dataset into overlapping cells, each of which contains its own approximate nearest-neighbors search structure. By combining search results from multiple cells as specified by coarse localization, we have demonstrated superior retrieval accuracy on a large image database covering downtown Berkeley. In this paper, we investigate how to select the parameters of such a system e.g. size and spacing of the cells, and show how the combination of many cells outperforms a single search structure over a large region.

Keywords: image matching, image retrieval, visual landmark recognition

1. INTRODUCTION

Localization has many applications in search and visually immersive applications. With their incorporation into smart phones, GPS devices have become commonplace in recent years. Unfortunately this information is usually not reliable, especially in urban environments, where tall buildings block line-of-sight to GPS satellites. Combined with WiFi and cell-tower location services however, a user can usually be localized accurately to within a few hundred meters. An alternative approach to localization is to match a camera view against an image database and compute the user's relative pose. This can work in places where GPS is unreliable, but comes with its own set of challenges. In previous work,[1] we showed that by taking advantage of coarse location information, it is possible to dramatically improve the accuracy of a large scale image retrieval system to be used for a finer, image based localization.

The basic idea behind our system is to divide a large geographic area into overlapping circular "cells", with each cell having its own individual search structure. We assume each image query is tagged with approximate location information derived from GPS, cell tower triangulation, or WiFi specified via an "ambiguity circle" of a given radius. We then perform an image retrieval search in each of the individual cells with which the ambiguity circle overlaps, and combine the results of multiple cells in order to arrive at a rank ordered list of images matched to the query image. In this paper we further discuss the factors affecting retrieval performance of the system[1] and justify the choice of parameters for such a system.

The remainder of the paper is organized as follows. First, Section 2 describes the processing steps of the system in our previous work.[1] Given this description, Sections 3 and 4 present empirical results showing that sharding of the image database into overlapping partitions can provide improved retrieval performance. Section 5 concludes with an analysis of the reasons behind the performance characteristics of our system.

2. OVERVIEW OF PREVIOUS WORK

The block diagram of our system is shown in Figure 1. Using the estimated location and accuracy of the query image, we select cells that may contain features found in the query image. We efficiently pair SIFT features in the query image to features present in the database using approximate nearest-neighbor search. This process, called the *search step*, yields an arbitrarily long list of potential matches between database SIFT features and query SIFT features. Since the kd-trees are built offline to contain features from a fixed region, we discard matches that lie outside an ambiguity radius λ given for the particular query.

Next, in the *filter step*, as many false matches as possible are eliminated based on geometric constraints and other match criteria. Each database image is then scored based on the number of remaining features matches, and

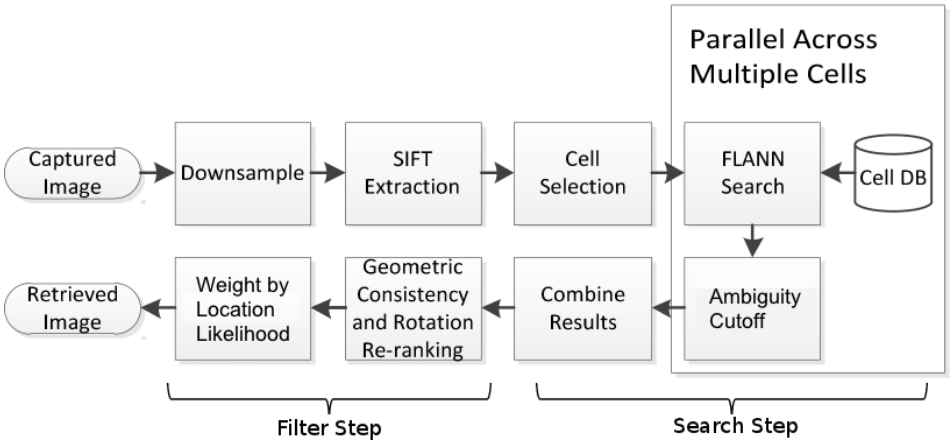

Figure 1. Block diagram of image-retrieval pipeline considered in this paper.

Set Name	Set Size	Downsampled Image Size	Sensor Resolution	Capture Device
Berkeley	100	680x512	8 MP	HTC Droid Incredible
Oakland	112	683x512	5 MP	Google Nexus S

Table 1. Query sets used.

the top image is chosen based on the number of matches and distance from estimated location. We have evaluated the performance of the system based on correctness of the top image retrieved. Our criteria for correctness is that a human, observing the query and retrieved image, can visually confirm the query and retrieved image come from the same scene.

2.1 Search Step

The simplest conceivable implementation of search would be to throw all known database features into one kd-tree. This solution is infeasible in practice, because retrieval accuracy degrades with the size of the kd-tree,[2] and also due to the memory requirements of such a structure. Using moderately sized images taken in the commercial areas of Berkeley, we have generated about 24 million features per square kilometer, which is already too large for a single machine. We address both concerns by sharding our database into cells built around a hexagonal lattice, as discussed in Section 2.3.

To perform the search given an approximate location, each cell that overlaps with the ambiguity radius λ is searched in parallel. We use the FLANN library for fast retrieval of the k-nearest neighbors of each cell. To combine results from the query, we concatenate the list of matches from each cell in a process we call cell "combination".[1]

2.2 Filter Step

In practice the list of feature matches returned by the search step has a high proportion of false matches, most of which are unavoidable even if the search step were to be exact. To prune as many of these false matches, we use RANSAC[3] to enforce epipolar constraints on the geometry of the matches, which provides us with a reranked list of candidate images. In addition, we reject matches based on other attributes such as rotation mismatch and Euclidean distance between the SIFT feature vectors.

2.3 Ambiguity Cutoff

We have previously introduced the concept of an ambiguity radius λ, defined as the uncertainty in user location, which can vary with each query based on reported GPS or cell tower accuracy.[1] In prior work, we use λ to choose parameters for our cell structure such as r, the radius of each cell, and d, the distance between the cells. For example, for an expected λ of 75 meters, we can construct the lattice with r=d=236.6 meters based on geometric constraints of Single Cell Existence (SCE). This condition guarantees that the ambiguity circle is completely

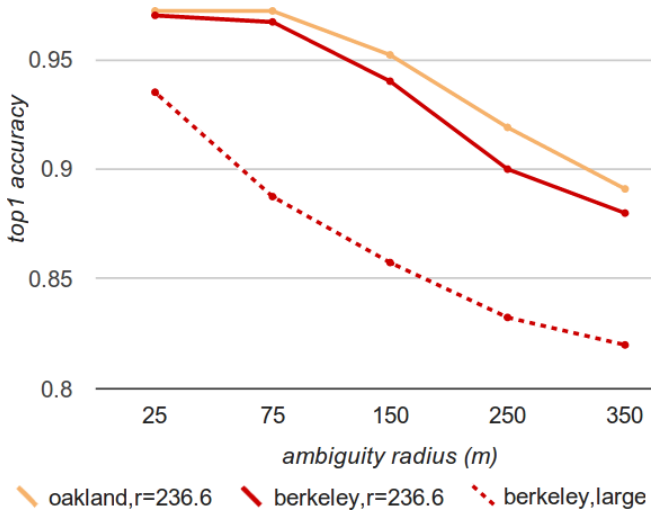

Figure 2. Overall retrieval accuracy dropoff with increasing ambiguity cutoff radius. Here cell combination for the search step is compared with a single large cell built over all the features.

contained in at least one cell containing the correct image match, and hence searching over that cell is ideally sufficient to retrieve the correct match.[1]

In this paper we introduce what we call *ambiguity cutoff* between the search and filter step by rejecting all images from the search step that are more than 25m outside this ambiguity radius.[1] This provides a coarse filter similar to the Naive Bayes postprocessing step in our previous work to weight retrieved matches by location. We use this concept to generate results in the remainder of this paper. Our primary motivation in introducing ambiguity cutoff is to allow the evaluation of search step performance without the lengthy training[1] more sophisticated methods of utilizing ambiguity information would require.

3. SENSITIVITY TO AMBIGUITY

For a query where λ is similar in magnitude to the radius of each search cell, cell combination is expected to outperform a single cell containing features over a larger region containing the combination. This is because kd-tree performance degrades with size. However, it is less evident that combination can perform well in cases where λ is much greater than the radius of the search cell, where instead of combining 6-8 cells, we are forced to combine 20-30. Such a situation may arise, for example, if GPS reception is particularly poor. Since the number of cells to combine increases with the square of the search radius when $\lambda \gg r$, the fraction of total true matches generated by the search stage is likely to shrink quadratically with λ. However, the number of true matches in the correct database image(s) stays constant regardless of λ, since they are only present in a constant number of cells. In essence, all that is required for retrieval to succeed is for the correct image to score a high number of matches, letting it "stand out" well. This allows combination to work well even when combining many cells.

Figure 2 shows the accuracy of top 1 retrieved images as a function of ambiguity cutoff for Berkeley and Oakland datasets shown in Table 1 using the "combination" approach.[1] The "large cell" curve corresponds to a single large kd-tree built over the entire query region which contains 21 million features for the "Berkeley" dataset. The cell parameters for the other two lines are r=d=236.6m. FLANN parameters[4] for the large cell are, nn=4, trees=4, checks=1024, and for the other lines, nn=1, trees=1, checks=1024.

As seen in Figure 2, the combination approach of[1] outperforms "largecell" particularly for large ambiguity circles. Furthermore, for a fixed cell radius, a 14-fold increase in ambiguity radius degrades top one retrieval performance by only 10%. This suggests that it is not necessary to increase cell radius in the presence of large ambiguity and that the combination approach works well with moderate size cells. In Section 4.2 we further examine the significance of cell radius in the combination step.

parameters	density	end-to-end accuracy	accuracy without filter step
r=236.6, d=167.3	approx. 6x	0.94	0.86
r=236.6, d=193.1	approx. 4.5x	0.96	0.86
r=236.6, d=236.6	approx. 3x	0.96	0.86
r=236.6, d=275	approx. 2x	0.95	0.85
r=236.6, d=334.6	approx. 1.5x	0.86	0.73
r=236.6, d=409.8	approx. 1x	0.84	0.71

Table 2. Performance of our system as a function of cell density.

4. CHOICE OF CELL GEOMETRY

Thus far we have observed that combination provides superior performance to single-cell approaches.[1] One remaining question is how to choose the parameters for constructing the cell structure given expected values of λ. We approach the problem by first finding the optimal cell density, which we define as the average number of cells a feature appears in. Holding density fixed, we then determine the cell radius for the hexagonal cell structure.[1]

4.1 Cell Density

Table 2 shows the performance gains of combination at a cell density of approximately 3x, which means that each feature is present in three or four cells. Since higher densities increase memory and CPU requirements, the natural question is whether further increases in cell density yield any performance gain. In Table 2 we evaluate combination performance at a range of densities, keeping r fixed and changing d. As seen, higher cell densities provide diminishing gains. Since a density of ˜3x appears performant and in a hexagonal lattice has the convenient property that r=d, we select this density for further testing.

4.2 Cell Size

We now investigate the effect of cell size on performance assuming r=d. There are two considerations for cell size - the performance of the search structure in question with respect to the number of features in each cell, and the size of the cell with respect to the expected λ. From benchmarks,[2] it is undesirable for cells to have more than millions of features per cell. Hence there is a soft upper bound on cell size.

At the lower limit, when cell radius is similar to or smaller than expected λ, many cells have to be combined. When this happens the search step ranks subregions rather than the entire ambiguity circle; this pushes more responsibility to the filter step to globally rank candidate images. In Section 3 we argued that combining many cells could perform well, and showed in Figure 2 that many-cell combination is indeed preferable to results from a single large, inaccurate kd-tree. However, it is less clear whether combining many small kd-trees outperforms a few moderate sized kd-trees, since both sized kd-trees have similar accuracies.

Figure 3 shows system performance under the r=d constraint with several selections of r from 150m to 500m. We use the same parameters as in Figure 2, which this figure extends. The average number of features for r=150, 236, 350, and 500m are 2, 4, 10, 20 million respectively. The more extreme choices of r such as 150m perform poorly across all λ chosen. Interestingly, the r=350 line suggests that there is no unique r that is optimal for all λ. For the largest value of λ, i.e. 350, the performance peaks at r=236 which corresponds to 4 million features per cell. In this case, the SCE condition would have set r=830m which results in a prohibitively large number of features per cell and hence low performance. Therefore the SCE condition should not be chosen to design cell structures for large values of ambiguity radius.

5. ANALYSIS OF COMBINATION STEP

In previous work we have shown that overlapping cell combination boosts overall retrieval performance even for small λ, but have not speculated on the reasons. Here we speculate that there are two factors at work in combination, the averaging of answers from multiple search structures, and the recall of more candidate matches in the search step to be pruned by the outlier-tolerant filter step.

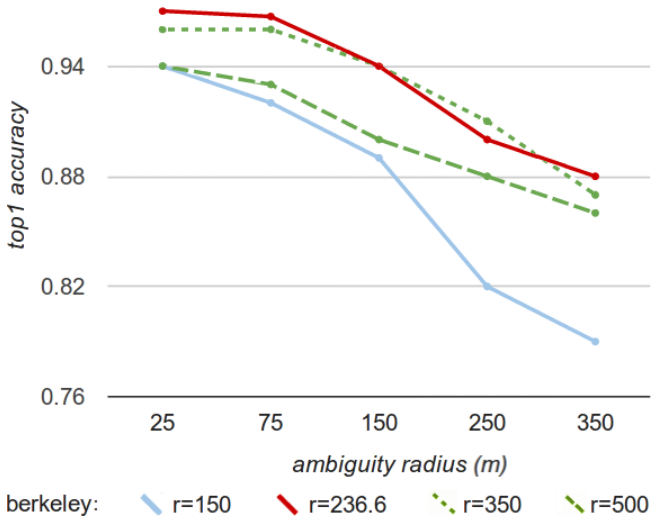

Figure 3. Comparison of overall system performance when combining cells built with different r. Each step up in radius roughly corresponds to a factor of two increase in cell size. At r=236.6m, each cell contains around 4.2 million features for our dataset.

5.1 Relation to Bagging

Bootstrap aggregation or bagging[5] is a machine learning meta-algorithm for improving the performance of classifiers by reducing variance and overfit. Multiple classifiers are trained on datasets sampled from a training set, and the output of the ensemble of classifiers are averaged during classification. Bagging is most commonly used for unstable classifiers such as decision trees, where small changes in the training set can significantly alter outputs.

While bagging is not directly applicable to the retrieval problem at hand, in principal it is similar to the combination step. Since our search structures are built to overlap each other, each kd-tree index is constructed over a different subset of the available data. Though nearest-neighbor classification should produce very stable results, we are limited to approximate methods because of retrieval speed constraints. Furthermore, the true nearest neighbor of a SIFT feature is not necessarily the "correct" match in practice.

To investigate whether overlapping cells do indeed provide performance gains as described, in Table 3 we compare combination and single-cell performance, omitting the filter step. We leave out the filter step because it includes a RANSAC component sensitive to the number of candidate matches it takes as input. Without the filter step, combination is truly averaging the output of each kd-tree when it concatenates their outputs. The query sets used to generate the results in Table 3 are described in Table 1. We built cells conforming to the Single Cell existence constraint as described in previous work.[1] An ambiguity cutoff radius of 75 meters is used, and the query locations are sampled from and averaged over a 75-meter radius uniform distribution around the true location to smooth out the results. The kdtree4 parameter indicates the use of four randomized kdtrees per

Retrieval accuracy, Berkeley set, no filter step.

	kdtree1	kdtree4
combination	0.857	0.863
single cell	0.830	0.833

Retrieval accuracy, Oakland set, no filter step.

	kdtree1	kdtree4
combination	0.912	0.928
single cell	0.875	0.875

Table 3. Average performance of combination vs other search methods.

	nn=1	nn=2	nn=4	nn=8	nn=16
ncells=1	.896	.891	**.946**	.929	.932
ncells=2	.936	.943	**.961**	.930	.942
ncells=4	**.956**	.956	**.**952	.944	.926
ncells=8	**.961**	.950	**.961**	.947	.935

Table 4. Mean retrieval accuracy varying with the number of nearest neighbors considered in FLANN search and number of cells combined.

cell by FLANN to perform the query, which has been shown to improve performance.[2,6] As seen in Table 3, the combination approach outperforms the single cell approach for both Oakland and Berkeley datasets. This is true regardless of whether one or more kd-trees are used per cell. In essence, the superior performance of multiple cell combination in retrieval is reminiscent of bagging in classification.

It is also possible to interpret the superior performance of combination in terms of redundancy and error correction codes. In combining the results from multiple cells, we ensure that poor performance by one cell does not dominate the final results and its effect is mitigated by the retrieval results from neighboring cells.

5.2 Multiple Nearest Neighbors

When running the system without the filter step, we have empirically found that admitting more than the first nearest neighbor from FLANN search is detrimental to overall performance. Using multiple nearest neighbors trades some precision for greater recall of correct feature matches, which is undesirable by itself. Once the filter step is added though, admitting more than the first nearest neighbor could be beneficial because a high proportion of false matches are pruned. Because our approach to combination concatenates the list of feature matches from each cell together, it also increases the recall of matches, though, as discussed in Section 5.1, not necessarily at a cost of precision.

To better understand the interaction between combination, multiple nearest neighbors, and the filter step, we run experiments varying the maximum number of cells combined and the number of neighboring features used. Table 4 shows mean retrieval accuracy as a function of the number of nearest neighbors considered in FLANN search, and number of cells combined using a single randomized kd-tree per cell. The filter step is activated for the results shown in this table. A total of 212 query images from Berkeley and Oakland are used to generate this table. The cell parameters satisfy the SCE constraint[1] and are chosen to be d=r=236.6m, λ=75m. The results are generated by sampling from and averaging over locations within the ambiguity circle to smooth out irregularities.

From Table 4 we conclude that system performance is somewhat sensitive to the quantity as well as quality of candidate matches processed by the filter step. With a single cell, using multiple neighbors boosts performance because the filter step eliminates enough false matches that overall precision increases. Even though the best performance occurs for ncells=8 and nn=1, the gap between that and ncells=1, nn=4 is minimal. In many scenarios this 2% drop in performance might be well worth the 8-fold decrease in CPU consumption and memory requirement.

6. CONCLUSIONS

From Figures 2 and 3, we conclude that cell radius should be chosen such that the number of features is a few million rather than a few tens of million. Specifically, for large ambiguity radii, applying the SCE condition in designing our cell structure results in poor performance. Comparing accuracy for λ = 350m in Figures 2 and 3, we conclude that even when cells of 20 million features are used, it is preferable to combine their results rather than use a single large cell. From Figure 2, we further conclude that the performance of a fixed cell radius system is fairly robust to changes in ambiguity radius, since a 14-fold increase in ambiguity radius only degrades retrieval performance by 10%. From Table 2, we conclude that a cell density of about 3x resulting in r=d offers the best accuracy and complexity performance. Finally, from Table 4, we conclude that if we use more than one nearest neighbor at the output of each kd-tree, the performance loss due to using a single cell is minimal as compared to multiple cells. This indicates interesting tradeoffs between retrieval cost and accuracy.

REFERENCES

[1] Zhang, J., Hallquist, A., Liang, E., and Zakhor, A., "Location-based image retrieval for urban environments," in [*ICIP*], (2011).

[2] Mohamed, A., Welinder, P., Munich, M., and Perona, P., "Scaling object recognition: Benchmark of current state of the art techniques," in [*ICCV*], (2009).

[3] Fischler, M. A. and Bolles, R. C., "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Comm. of the ACM* **24**(6), 381395.

[4] Muja, M. and Lowe, D. G., *FLANN - Fast Library for Approximate Nearest Neighbors.*

[5] Russell, S. J. and Norvig, P., [*Artificial Intelligence: A Modern Approach*], Prentice Hall, Upper Saddle River, N.J (2010).

[6] Muja, M. and Lowe, D. G., "Fast approximate nearest neighbors with automatic algorithm configuration," in [*VISAPP*], (2009).