

STRUCTURAL INDEXING USING LOCAL  
IMAGE FEATURES

GERI GROLINGER

Professor:  
Sven Dickinson

April 29, 2008

University of Toronto  
Department of Computer Science

# Contents

<b>List of Figures</b>	<b>3</b>
<b>1 Introduction</b>	<b>4</b>
1.1 Motivation . . . . .	4
1.2 Problem definition . . . . .	5
<b>2 Related work</b>	<b>6</b>
<b>3 Perceptual grouping of local features</b>	<b>9</b>
3.1 Singletons . . . . .	9
3.2 Pairs . . . . .	9
3.2.1 Pairing criteria: scale . . . . .	10
3.2.2 Pairing criteria: spatial distance . . . . .	10
3.2.3 Pairing criteria: feature distance . . . . .	11
<b>4 Computing an index for feature-pairs</b>	<b>11</b>
4.1 Feature relation . . . . .	12
4.2 Feature dimensionality . . . . .	13
<b>5 Indexing and voting</b>	<b>13</b>
5.1 Index structure . . . . .	14
5.2 Indexing . . . . .	15
5.3 Voting and scoring . . . . .	15
<b>6 Experiments and results</b>	<b>16</b>
6.1 Image data . . . . .	16
6.2 Simulating occlusion and clutter . . . . .	17
6.3 Results . . . . .	17
<b>7 Conclusions and future work</b>	<b>21</b>

**8 Acknowledgments** **22**

**Bibliography** **24**

## List of Figures

1	Set of 4 images from image data . . . . .	16
2	Another example of 4 image-set from image data . . . . .	17
3	PCA-SIFT indexing results . . . . .	18
4	PCA-SIFT results for a range of bins in Nearest Neighbour search	18
5	Indexing using SIFT and PCA dimensionality reduction . . . . .	19
6	Regular split and sliding-split point rule . . . . .	19
7	Various voting methods . . . . .	20
8	Range of bins for SIFT indexing . . . . .	20
9	Indexing results for SIFT with clutter and occlusion . . . . .	20

# 1 Introduction

## 1.1 Motivation

The problem of object recognition is one of the most challenging and important problems in computer vision. Today's recognition systems typically extract a sparse set of local features, each of which characterizes a small patch of distinctive image data that is invariant to minor changes in lighting and viewpoint. The most popular such feature, called a SIFT (scale invariant feature transform) feature, specifies the position, scale, orientation, and histogram of the image gradient data contained in the patch. When such features are extracted from an image, they "vote" for objects that contain them (a process called *indexing*). Objects that receive a significant number of votes represent promising candidates for explaining the image.

However, since each such feature votes independently, two different objects consisting of the same set of features but in very different configurations will receive the same number of votes. Therefore, a costly consistency check must be applied to each model candidate in order to determine the best matching model. As the database grows to contain millions of object images, the ambiguity of a single image feature may grow to the point where each feature is a member of a large number of objects, leading to a potentially intractable number of candidates that must be verified. Ambiguity is further compounded when the number of object features is small or represents a small fraction of the total number of image features. This happens, for example, when the target object is occluded or embedded in a cluttered scene. Instead of invoking strong geometric constraints at verification time, we explore ways to incorporate these constraints at indexing time, leading to far fewer candidates that need to be verified.

## 1.2 Problem definition

In this project we try to address the above mentioned problems. One problem is the ambiguity of a single feature that grows when the feature is contained in a large number of objects. In this case, the ambiguous features vote for large number of objects, producing an intractable number of candidates that have to be verified by an expensive consistency check. The second problem with single features becomes apparent when the query object is occluded or embedded in a cluttered scene. In this case the number of features on the query object is small or it represents a small part of the total number of image features. In this project we attempt to solve the above problems by disambiguating image features at indexing time, so that later at verification time, smaller number of candidates have to be verified.

We present a method for combining two SIFT features into a single compound feature. By combining two SIFT features, the indexing ambiguity declines as the index becomes richer and more specific. Our method for combining two SIFT features into a single compound feature, in addition to retaining the encoding of each patch, encodes the relations between the features, including relative scale, orientation, and proximity. Now, each feature contains encoding of two patches (instead of one) and is therefore more distinct. The added information about the features' relative scale, orientation and proximity incorporates into these new features geometric constraints before indexing (instead of just at verification time). After creating a compound feature, we reduce the dimensionality of the feature down to that of the original SIFT feature to make the indexing with pairs comparable to the indexing with the singletons. By combining two SIFT features, the indexing ambiguity clearly declines as the index becomes more specific. On the other hand, the process of combining the features into meaningful groups of two comes at a certain computational cost. Studying this trade off will help us find an optimal balance between bottom-up feature extraction and top-down model recovery as a function of database size.

In general terms, the project follows the path explained in the following para-

graph. First we perform the indexing with single features using images from a general purpose database (objects on these images are not specifically occluded nor embedded in cluttered scenes). After that we will create pairs of features and perform the indexing with them using the same images. In the next step we make the query images “harder” for the recognition. We do this by pasting the objects (in different proportions) from our original images onto unrelated background images. This way we simulate occlusion and clutter in the scene. As mentioned above, we expect the performance (in terms of recognition) of the single features to decline. However, the more distinct feature is expected to perform better in these situations since it is richer, more specific and more distinct than the singletons. Since the performance of feature pairs depends on many factors, we try different methods of pairing, indexing and scoring and present, compare and evaluate them all.

The remainder of this report is structured as follows. Section 2 gives overview of the related work. Section 3 describes methods for grouping of local features. Section 4 talks in detail about computing an index for feature-pairs. Section 5 describes indexing and voting. Section 6 covers the experiments and the results. Conclusions and future work are outlined in section 6.

## **2 Related work**

We first discuss several prominent indexing methods which inspired our work. We look into several problems and their solutions: choice of features and feature grouping with addition of relational information, data structures for indexing, indexing process, scoring methods and robustness to occlusion and clutter.

The choice of features varies largely in the work done in this area. Lamdan, Schwartz and Wolfson [9] simply use points of sharp concavities and deep convexities. In [14], Stein and Medioni use edges extracted from the image containing the model. However they go a step further and do line fitting and then combine the consecutive segments into a larger and more distinct entities. Costa and Shapiro

[5] choose to work with coaxial circular arcs, ellipses, triples of line segments, junctions and parallel line segments. They extend the features by using features' properties and relationships between them (such as enclosing and coaxial relations). They build graphs where the features represent nodes and relationships the arcs. In [7], Huet and Hancock also embed relational and structural information into their features. They build graphs of line segments extracted from an image and add to this descriptor two attributes: directed relative angle and directed relative position. Beis and Lowe [2] work with the assumption that a single feature will not be specific enough to make the difference between many different objects. Therefore, they use perceptually significant groupings of straight edge segments. They group segments based on parallelism, cotermination and symmetry. Further, they incorporate angle and edge-length relationships between the segments in a group into the multidimensional feature vectors. We base our work on a similar assumptions even though we use different type of features. The features used by Nister and Stewenius [12] are more similar to the features we use in our work. They work with MSERs (Maximally Stable Extremal Regions) [11] while some parts are implemented according to the SIFT (Scale Invariant Feature Transform) [10].

The two most popular data structures for indexing are hash tables and KD-trees. The authors in [9], [14] and [5] use hash table to store extracted features for indexing. In this case the indexing is done simply by indexing with the query features (or feature sets/graphs) into the hash table and accumulating the votes for the features found in that bin. Huet and Hancock [7] use histogram-based indexing where they have 36 vertical bins for relative angles and 12 horizontal bins for relative positions between the features. Beis and Lowe [2] choose KD-trees and Nister and Stewenius [12] use hierarchical K-means (which are very similar to KD-trees) for the indexing structure since the tree structures are more efficient than hash-tables for retrieving nearest neighbours in higher-dimensional spaces. At indexing time, Beis and Lowe [2] index the KD-tree with features from the query image and use the Best-Bin-First (BBF) algorithm for retrieving nearest

neighbour feature vectors, indicating potential matches. The BBF algorithm first finds the bin containing the query point and then looks in the bins closest in Euclidean space to the query point. In our work, we use a similar method to BBF algorithm in the indexing stage. Nister and Stewenius [12] at indexing time propagate the feature descriptor vector down the tree by comparing at each level the descriptor vector to the candidates in the tree. For scoring, they use hierarchical voting scheme where they determine the similarity of the path down the tree for the database image descriptor and the query image descriptor. In one of the scoring methods that we tried, we apply a similar reasoning and score according to the density of the points in the tree (as explained in section 5) with the assumption that the more dense parts of the tree will be more ambiguous and contribute with less information to the recognition process.

The problem of occlusion and clutter that we address in our work is also explicitly tackled in some of the previous papers. Stein and Medioni [14] use flexible line fitting tolerances for polygon approximations to allow for feature robustness in presence of noise. Costa and Shapiro [5] introduce multi-level indexing that applies for multi-object scenes where features are typically missing or become different due to occlusion and clutter. This method starts indexing with larger subgraphs (the features they use are represented as relational graphs) and goes down toward smaller subgraphs until all objects in the scene are recognized. Large subgraphs are appropriate for scenes containing only one object or unoccluded scenes while small subgraphs are better for the case with occlusion and clutter since such scenes contain smaller number of features. Huet and Hancock [7] and Beis and Lowe [2] attempt to account for occlusion and clutter by adding relational information to their feature groups, making them in this way more specific and distinct. Nister and Stewenius [12] claim that their approach is efficient for larger databases and is robust to background clutter and occlusion. They count number of images with at least one path through a node as a clue of how ambiguous that node is.

## 3 Perceptual grouping of local features

We developed and explored several methods to incorporate strong geometric constraints at indexing time, leading to far fewer candidates that need to be verified. We present these methods for combining two SIFT features into a single compound feature with the assumption that the indexing ambiguity declines as the index becomes richer and more specific. In addition to retaining the encoding of each patch, we encode the relations between the features, including relative scale, orientation, and proximity. Then, we reduce the dimensionality of the compound feature down to that of the original SIFT feature.

### 3.1 Singletons

For feature extraction, we tried two different methods.

First we used the PCA-SIFT [8] which like SIFT, encodes the salient aspects of the image gradient in the feature point's neighborhood. However, instead of using SIFT's smoothed weighted histograms, this method applies Principal Components Analysis (PCA) to the normalized gradient patch. The dimensionality of the features obtained by the PCA-SIFT method can be adjusted.

As a second method (when the performance of the feature-pairs was not as expected, as explained in the next section), we used the original implementation of the SIFT extractor [10] to obtain the 128 dimensional feature vectors. Further we applied PCA to these features to reduce their dimensionality to the one we initially used (more detail on this is given in section 4).

### 3.2 Pairs

A feature-pair consists of two PCA-SIFT [8] or SIFT [10] features that are paired according to the specific rules that are explained in the next three subsections. The pair records the original location of both features in the pair but the feature descriptor vector of the pair consists of the feature vectors of both elements of the

pair. A pair is represented as a feature vector in a multi-dimensional space as the singletons and it has the same dimensionality as a single feature in the original framework.

To avoid pairing each feature with each other one (this would produce too many pairs), we use the following method: for each feature in an image, we take  $n$  (for example 30) of spatially closest points to it and pair it with each of them, producing  $n$  feature-pairs. We choose the spatially closest points because the assumption is that the features that are closer in space are more likely to belong to the same object and we want the two elements of a feature-pair to belong to the same object. Next, each of these feature-pairs is scored according to three criteria: scale, spatial distance and distance in feature space. Each scoring method produces a score between 0 and 1 and the three scores are multiplied to obtain the overall score. The feature-pair with the best overall score is chosen to form the final feature-pair. Once the pairs are formed in the described way, they are treated the same as the single features in the rest of the system.

### **3.2.1 Pairing criteria: scale**

We want the two elements of a pair to be close in scale. If the points' scales are far apart, there is a chance that, if we have another image of the same object, one of them will be lost before the other (this happens because of the way the SIFT features are extracted) and therefore, our pair will not "survive". Therefore, the closer the elements of the pair are in scale, the scale score will be higher.

### **3.2.2 Pairing criteria: spatial distance**

We do not want to combine two points that are located at the opposite ends of an image since they probably would not belong to the same object. On the other hand, we do not want them to be too close or completely overlapping either. If the encoded areas of the two points is partially or completely overlapping, then the patches of the image that they are encoding are too similar and the distinctiveness

of the feature-pair would be lessened. For the distance calculation, we use the Euclidean distance and we normalize it by the scale of the smaller point.

### **3.2.3 Pairing criteria: feature distance**

The distance between the two points in feature space is also calculated as Euclidean distance. The pair is scored with a higher score if the feature vectors of the two points are not similar. In this case the score is never zero since the pair containing two very similar feature points can also be useful. The score is lower for the more similar points to avoid repetitive textures.

## **4 Computing an index for feature-pairs**

We tried several methods for computing the index for feature-pairs.

The first method for index computation uses the PCA-SIFT [8] feature extractor for extracting singletons (single image features). The singletons extracted by this method are represented by 36 dimensional descriptor vector. To have a comparable indexing, we decrease the dimensionality of the feature-pairs to 36 dimensions. Since the PCA-SIFT [8] extractor generates the feature descriptor whose values are ordered by importance, we simply take the 18 most significant dimensions of each element of the pair and create a 36 dimensional feature-pair. This method did not produce satisfactory results. We are confident that the reason lies in the fact that the singleton feature's dimensionality is first reduced by the PCA-SIFT [8] extractor and then we pick the 18 from the remaining dimensions.

We try to handle this problem by first putting together the SIFT [10] feature in its full dimensionality ( $128 + 128$ ) and then applying the PCA dimensionality reduction to it to reduce it to 36 dimensions. This improved significantly the obtained results, as it can be seen in section 6.

## 4.1 Feature relation

To make the feature-pair even more distinct we also tried incorporating into the pair encoding the relational information between the elements of the feature-pair. The relational information that we add are relative scale, relative distance, and headings.

Relative scale between the two points A and B is defined as follows:

$$r_s = \frac{\min(s_A, s_B)}{\max(s_A, s_B)}$$

where  $r_s$  is relative scale between point A and point B, and  $s_A$  and  $s_B$  are scale of A and scale of B, respectively.

Relative distance between the points A and B is defined as follows:

$$r_d = \frac{|A, B|}{\min(s_A, s_B)}$$

where  $r_d$  is relative distance and  $|A, B|$  is the Euclidean distance between points A and B.

Heading between points A and B is defined in both directions, i.e. we have “heading AB” and “heading BA”. It represents the difference between the orientation of point A and angle to point B and it is defined as follows:

$$h_{AB} = \Delta_{\Theta}(\tan^{-1}(y_A - y_B, x_A - x_B) - o_A)$$

where  $\Delta_{\Theta}(\cdot) \in [-\Pi, +\Pi]$  denotes the principle angle,  $h_{AB}$  stands for “heading AB”,  $o_A$  is orientation of A and  $y_A, y_B, x_A, x_B$  are the coordinates of points A and B. The “heading BA” is defined in a similar way.

Before the values for the spatial relations between the points are added to the feature-pair’s descriptor vector, they have to be scaled to appropriate size. Otherwise, these values could represent an insignificant part of the feature vector

and therefore, they could get lost in the PCA dimensionality reduction. We scale the spatial relations values so that the range of all four values together represent approximately 30% of the feature-pair range. After the spatial relations are added to a 256 dimensional (128+128) feature-pair vector, we reduce it's dimensionality by applying PCA dimensionality reduction to that of the singleton feature we used before (36).

## 4.2 Feature dimensionality

Our choice of dimensionality was motivated in several ways. The KD-tree that we use as the index structure poses a limit to the dimensionality of the indexing features. KD-tree is believed to be less effective with increasing dimensionality. Further, the dimensionality reduction helps make the computation tractable when the database is in order of millions of indexing features.

Since we use PCA-SIFT [8] in our first method, we use its default feature dimensionality (36). For feature-pairs we use the same dimensionality to factor out variation in performance of KD-tree.

The SIFT extractor [10] generates 128 dimensional features. However, we reduce them by applying PCA dimensionality reduction to 36 dimensions to be able to compare results to our first method. We do the same for feature-pairs, i.e. feature-pairs' dimensionality is also reduced to 36 dimensions.

## 5 Indexing and voting

The main goal of indexing is fast runtime recovery of the correct hypothesis. We present this as a nearest neighbour search in which, for a given feature vector, a number of closest points from the indexing space are retrieved. The KD-tree data structure that we choose is very efficient for obtaining nearest neighbours in a high-dimensional space.

## 5.1 Index structure

The KD-tree is a space-partitioning data structure. It is built as follows: Starting with a complete set of  $N$  high-dimensional points, the space is split on the dimension  $d$  in which the data exhibits the greatest variance (the difference between the largest and the smallest coordinate for this dimension is the largest). Then a split point  $p$  is chosen along the split dimension according to one of the following two rules: the cut is made at the mean value of the data in that dimension or the cut is made with a sliding-split point rule where the split is at the center, unless that would leave all the points on one side; in that case, the split is shifted so that there is at least one point on each side. This has the effect of generating trees with fewer long, thin bins. We have implemented and experimented with both methods and found that the later produces better results. Next, an internal node is created to store  $d$  and  $p$  (the split dimension and the split value) and the process iterates with both halves of the data until a bin contains fewer than a small fixed number of points (10, in this case). The leaves of a KD-tree form a complete partition of the data space, with the property that bins are smaller in higher-density regions and larger in lower density regions. This means that the nearest neighbour to any query should lie, with high probability, in the bin where the query falls or in an adjacent bin [2]. The points that are used for training the tree are arbitrary but they belong to the same set of images.

The KD-tree stores the features from all images in the dataset. The tree populating is done as follows: For each feature, we find the path down the tree based on the split information in the inner nodes. At the end of the populating process, we have the leaves of the tree containing the feature vectors of all data set images. The leaves represent the bins that completely partition the high-dimensional data space.

## 5.2 Indexing

At runtime, after the feature vectors are extracted from the query image, they are used to query the index. The KD-tree and the Nearest Neighbour algorithm [1] provides for fast access to the closest stored neighbours (that are the most similar in feature space). To retrieve the nearest neighbours of a feature vector, we first find the exact leaf node (bin) where that point belongs and then we use the Best Bin First (BBF) search algorithm [1] to retrieve the nearest neighbours to that point. The main idea behind the BBF search is to search first the bins closest in Euclidean distance to the query point rather than those closest in the KD-tree branch structure (which is the method used in the standard KD-tree search).

## 5.3 Voting and scoring

After finding the nearest neighbours with BBF search, votes are cast for each obtained neighbour feature vector for the image where that feature belongs. We examined and implemented several different voting methods. Each query feature can vote for all images where the features retrieved by nearest neighbour search come from. Alternatively, we can have a limit of one vote per query point for each image. The last option is to weight the votes according to the Euclidean distance of the query feature to the features obtained by the nearest neighbour search. To account for dense areas of the tree (which are less important since the points in these areas are more ambiguous due to their similarity) we normalize the weight by the density of the tree space considered by the nearest neighbour search. The last of the three described methods is the finest and accordingly it gives the best results among the three.

Given this voting scheme we calculate the score for the query image. The scoring method is tightly related to our image data set. The image set that we use contains images in groups of four that belong together. For score calculation, we take into consideration only the three images from the same group as the query image. We find its score by sorting the votes and looking at the rank of the three

images from the same set as the query image. From the rank of these three images, we obtain the precision which is  $\frac{i}{rank}$ , where  $i = 1, 2, 3$  stands for first, second and third image. We calculate the average precision of the three images across all the query images and the overall average precision which is used to evaluate our method.

## 6 Experiments and results

### 6.1 Image data

The image set that we are using comes from Center for Visualization and Virtual Environments of the University of Kentucky. It consists of 10200 images where the images are grouped into sets of 4 images of the same object taken from different point of view, from different distance or with different lighting. Two examples of four images that belong to the same set can be seen in Figures 1 and 2.



Figure 1: Set of 4 images from image data



Figure 2: Another example of 4 image-set from image data

## 6.2 Simulating occlusion and clutter

To simulate occlusion and clutter, we produce “harder” images where each one contains a proportion of one of our original images and a part from a random background image. We use these images for querying in the same way as the images from our image set.

## 6.3 Results

In this section we present and discuss the results obtained by different methods and techniques described in this report. Even though we did not manage to improve the results to the level where the pairs perform better than the single features, we show how we improved the method and explain why we did not yet succeed in our intention.

The results shown in Figure 3 are obtained by using PCA-SIFT [8] and by retrieving 25 nearest bins in the Nearest Neighbour search. The first column shows the average precision for indexing with single features, feature-pairs and feature-pairs in case when we choose 10 pairs per each singleton. The second column shows the ratio between the average precision of singletons and pairs. It is clear that the pairs perform better when there is only one pair per each singleton. We are confident that the reason behind this is that introducing multiple pairs introduces noise in indexing.

	Average precision	Singletons to pairs ratio
Singletons	0.2262	
Pairs	0.0846	2.6738
Multiple pairs (10)	0.0535	4.2280

Figure 3: PCA-SIFT indexing results

Figure 4 shows the results obtained by the same method but for a range of bins used for Nearest Neighbour search. We can see that decrease in the number of bins improves the results for both singletons and pairs. However, the ratio in the last row indicates that the results for the singletons improve faster than the results for pairs.

Average precision for different number of bins								
	35	30	25	20	15	10	5	2
Singletons	0.1962	0.2104	0.2261	0.2450	0.2696	0.3002	0.3398	0.3641
Pairs	0.0798	0.0820	0.0846	0.0870	0.0903	0.0937	0.0969	0.0976
Singletons to pairs ratio	2.4586	2.5659	2.6726	2.8161	2.9856	3.2038	3.5067	3.7305

Figure 4: PCA-SIFT results for a range of bins in Nearest Neighbour search

The results obtained by using SIFT [10] features after the PCA dimensionality reduction was applied to them are presented in Figure 5. The top part of the first column shows the results of indexing with singletons, pairs, and pairs with added spatial relations, while the bottom part shows the results of indexing with pairs

where we choose 5 or 10 pairs per each singleton. The second column shows the ratio of singletons' and pairs' results. From the ratios, we can see that addition of spatial relations improves the performance of the pairs. However, as before, using multiple pairs does not help improve the results.

	Average precision	Singletons to pairs ratio
Singletons	0.2901	
Pairs	0.1338	2.1682
Pairs with spatial relations	0.2475	1.1721
Multiple pairs (5)	0.1014	2.8609
Multiple pairs (10)	0.073	3.9740

Figure 5: Indexing using SIFT and PCA dimensionality reduction

Figures 6 and 7 show the results obtained by the same method but with different splitting rules (Figure 6) and with different voting techniques (Figure 7). As it can be seen, sliding-split point rule performs better for both singletons and pairs. From the voting methods that we tried, weighted voting gives the best results individually for singletons and pairs and for their ratio.

	Average precision	
	Regular split	Sliding split point
Singletons	0.2826	0.2901
Pairs	0.1290	0.1338
Singletons to pairs ratio	2.1907	2.1682

Figure 6: Regular split and sliding-split point rule

Figure 8 shows the results from indexing with SIFT [10] after PCA dimensionality reduction was applied to it for a range of bins used to Nearest Neighbour search. The ratio of singletons' to pairs' values is shown in the bottom row of the table. As before, smaller number of bins produces better results for singletons and

	Average precision		
	One vote	More votes	Weighted votes
Singletons	0.2901	0.2128	0.3933
Pairs	0.1338	0.1089	0.2541
Singletons to pairs ratio	2.1682	1.9541	1.5478

Figure 7: Various voting methods

for pairs. However, now the pairs' results improve faster (as it can be seen from the bottom row with ratios).

	Average precision for different number of bins				
	20	15	10	5	3
Singletons	0.3075	0.3279	0.3545	0.3860	0.3995
Pairs	0.1481	0.1659	0.1896	0.2237	0.2397
Singletons to pairs ratio	2.0763	1.9765	1.8697	1.7255	1.6667

Figure 8: Range of bins for SIFT indexing

The results of indexing with “harder” images where we simulated clutter and occlusion can be seen in Figure 9. As previously, the pairs with added spatial relations perform better than the pairs without them. Also, as expected, the results for pairs are better for “harder” images (with 50% of the background image - column 0.5).

	Average precision		Singletons to pairs ratio	
	0.25	0.5	0.25	0.5
Singletons	0.2601	0.3152		
Pairs	0.1448	0.1905	1.7963	1.6546
Pairs with spatial relations	0.1561	0.2065	1.6662	1.5264

Figure 9: Indexing results for SIFT with clutter and occlusion

## 7 Conclusions and future work

As a part of this project, I got an opportunity to review a significant amount of literature related to object recognition and structural indexing. Surveying the methods tried so far, gave me a better idea of what the goal of this project should be. Even though the idea of perceptual grouping is not new, it has not been tried yet on SIFT features since they are very distinctive and work very well for recognition. However, as the database sizes grew and the recognition tasks became harder (clutter and occlusion), the need for even more specific and distinct feature became apparent. The methods I studied and implemented are partially based on the work done so far but also we introduced some new ideas since the previously developed methods do not apply to SIFT. The techniques for feature grouping and the addition of relational information (between the single SIFT features) to feature-pairs are completely new. Further, the methods for voting as well as the methods for challenging the recognition process (by artificially adding clutter and occlusion) are new in this application too. As the results have shown, the approaches that we have tried so far are not as successful as we would want them to be. However, we were able to narrow down the reasons due to which our results are still limited.

Feature-pairs have a huge advantage of being much more distinctive than the single features. However, by reducing their dimensionality we lose some of this distinctiveness and this shows in our results. The dimensionality reduction seems to be hurting our results but on the other hand dimensionality reduction is necessary in order to avoid the limitations of the KD-tree. The KD-tree is believed to be less effective with increasing dimensionality. In our future work, we would like to look into the ways to go around this problem. Perhaps we can allow the dimensionality to be a bit higher than it is right now, but we believe that this would not change the results significantly. Maybe a better solution would be an improved or new indexing data structure.

Another cause for the insufficient improvement of the results is the survivability of the feature-pairs. Choosing the pairs that guarantee that both of their elements will survive from one image of the same object to another is very hard.

We tried to ensure this by using our three pairing criteria but it is quiet possible that they have to be thought through and improved. For a feature-pair to be useful, both of its feature elements have to resist the view, lighting and scale variations. To verify the “survivability problem” we have run some brute-force tests that check the proportion of the single features that survive between the 4 images of the same object (from our data set) versus the proportion of the feature-pairs that survive. We have found out that (for a certain threshold) for singletons, the mean fraction of the features that survive in the images that do not match is 2%, whereas in the images that do match it is 17%. For pairs, the mean fraction of the features that survive in the images that do not match is 0.2%, and in the images that do match it is 3%. For this brute-force matching purposes we were matching both elements in their original dimensionality of each feature-pair. This result tells us that the survivability of the pairs is smaller than the survivability of the singletons but the pairs rely on the low false positives. However that hurts the performance of the feature-pairs in the KD-tree because the search done with KD-tree adds roughly the same amount of noise (false positives) to both singletons and pairs. This added noise hurts the feature-pairs more than the singletons since the ratio between matches in images that match and those that do not match deteriorates faster for pairs (for example, if the KD-tree search adds 1% of noise to both singletons and pairs, the the numbers for singletons become  $17 + 1 = 18\%$  and  $2 + 1 = 3\%$  and the numbers for pairs become  $3 + 1 = 4\%$  and  $0.2 + 1 = 1.2\%$ ).

The trade offs between the distinctiveness of the feature-pairs, dimensionality reduction, their survivability and their performance limitations due to the KD-tree are still to be studied.

## 8 Acknowledgments

I would like to thank my supervisor, Professor Sven Dickinson, for his guidance and very useful comments and input. I appreciate very much that I had to prepare this write up because it gave me a chance to look back and think over everything I

did so far. It allowed me to evaluate the implemented ideas, to summarize, explain and understand the obtained results and plan the further path of the project.

I would also like to give many thanks to Mike Jamieson for his mentorship and time invested to discuss the methods and ideas.

## References

- [1] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 1000, Washington, DC, USA, 1997. IEEE Computer Society.
- [2] Jeffrey S. Beis and David G. Lowe. Indexing without invariants in 3d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):1000–1015, 1999.
- [3] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *PAMI*, 24(4):509–522, April 2002.
- [4] Mauro S. Costa and Linda G. Shapiro. 3d object recognition and pose with relational indexing. *Comput. Vis. Image Underst.*, 79(3):364–407, 2000.
- [5] M.S. Costa and L.G. Shapiro. Scene analysis using appearance-based models and relational indexing. *iscv*, 00:103, 1995.
- [6] David Forsyth, Joseph L. Mundy, Andrew Zisserman, Chris Coelho, Aaron Heller, and Charles Rothwell. Invariant descriptors for 3d object recognition and pose. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(10):971–991, 1991.
- [7] Benoit Huet and Edwin R. Hancock. Relational histograms for shape indexing. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 563, Washington, DC, USA, 1998. IEEE Computer Society.
- [8] Y. Ke and R. Sukthankar. Pca-sift: A more distinctive representation for local image descriptors, 2004.
- [9] Y. Lamdan, J.T. Schwartz, and H.J. Wolfson. Affine invariant model-based object recognition. *RA*, 6:578–589, 1990.

- [10] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.
- [11] J. Matas, O. Chum, U. Martin, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *Proceedings of the British Machine Vision Conference*, volume 1, pages 384–393, London, 2002.
- [12] D. Nistér and H. Stewénus. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2161–2168, June 2006.
- [13] Kaleem Siddiqi. Indexing hierarchical structures using graph spectra. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1125–1140, 2005. Member-Ali Shokoufandeh and Student Member-Diego Macrini and Member-Sven Dickinson and Fellow-Steven W. Zucker.
- [14] Fridtjof Stein and Gérard G. Medioni. Structural indexing: Efficient 2d object recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(12):1198–1204, 1992.
- [15] Haim J. Wolfson and Isidore Rigoutsos. Geometric hashing: An overview. *IEEE Comput. Sci. Eng.*, 4(4):10–21, 1997.