Abstract

A set of modular algorithms for efficiently finding SIFT feature correspondences in images or image archives is presented. The basic algorithm, called SIFT-HHM (for SIFT Handed-Hierarchical Matching), finds shortcuts to the most likely feature point matches in a given image pair, converging in approximately $1/15$ the time it takes to perform an exhaustive search, and is at least four times faster than both PCA-SIFT and SURF, two descriptors designed partially to address the relatively slow matching speeds of SIFT. The phases of SIFT-HHM are based on a robust analysis of the SIFT keypoint descriptor, using data from an archive with over 5,000 images. As part of the analysis, we demonstrate a method for filtering out less-distinct SIFT vectors that improves overall matching performance.

An extension of SIFT-HHM, which we call SIFT-BHHM (for SIFT Binned Handed-Hierarchical Matching) is used to address the content-based image retrieval problem of finding feature correspondences in large databases. The algorithm sorts a keypoint database into bins using the principal components of the SIFT descriptor as the binning criteria. Given a query vector and a search window, the algorithm returns all of the bins within the desired range, and the bin contents are finally searched using SIFT-HHM. Our experiments show SIFT-BHHM to be preferable over current tree-based methods for a number of reasons. Most significantly, it will find feature correspondences approximately three times faster than the current state of the art, regardless of the size of the database (experiments were run on databases of up to 2 million SIFT vectors). Secondly, it takes less than $1/10$ the time to create our bin structure than it does to build a k-means tree. Finally, we note simplicity of storage, scalability, and suitability to distributed processing as incidental benefits.

The algorithms presented in this thesis both improve existing SIFT-based applications and create opportunities for more widespread use of SIFT, especially in the online and real-time application areas, where SIFT is typically regarded as too slow to compute and to match to be effective. Furthermore, applications that have been designed to use SURF or PCA-SIFT for matching speed reasons may benefit from being re-configured for SIFT to take advantage of our techniques.
Acknowledgements

The author would like to thank his supervisor, Dr. Anthony Whitehead, for providing much-needed guidance, invaluable expertise, and time out of his busy schedule to help construct and fine-tune the algorithms presented in this thesis, to proofread individual chapters and various conference submissions.

On a more personal level, the author wishes to thank his parents and siblings for not asking him “when is your thesis going to be finished?” too often, and, of course, his partner Shannon for her love and support during the thesis-writing period, especially when it takes the form of impromptu interpretive dances in the basement.
# Table of Contents

**Abstract**  
- ii

**Acknowledgements**  
- iii

**List of Tables**  
- vii

**List of Figures**  
- ix

**Introduction**  
- 1  
  1.1 Motivation.................................................................2  
  1.2 Problem Statement and Scope........................................2  
  1.3 Contributions............................................................3  
  1.4 Thesis Overview..........................................................3

**SIFT and Related Work**  
- 4  
  2.1 SIFT Algorithm...........................................................4  
    2.1.1 Scale-Space Extrema Detection..................................5  
    2.1.2 Keypoint Localization.............................................6  
    2.1.3 Orientation Assignment..........................................7  
    2.1.4 Descriptor Construction..........................................7  
    2.1.5 SIFT Matching.....................................................8  
  2.2 Principal Component Analysis........................................9  
  2.3 SIFT-Related Local Detectors and Descriptors......................10  
  2.4 Nearest-Neighbour Search Algorithms for High-Dimensional Spaces........................................12

**Keypoint Descriptor Properties**  
- 14
List of Tables

Table 3.1: Content summary for image set $S_A$..................................................................................14
Table 3.2: A ranking of SIFT descriptor indices in order of descending standard deviation ($\delta$)..........17
Table 3.3: Primary, secondary, tertiary and “remainder” descriptor index groupings..........................19
Table 3.4: Correlation matrix for primary elements, with pairings that cross the medial line
highlighted in grey, and the elements closest to the keypoint patch centre indicated with a bold border.........................................................................................................................21
Table 3.5: Separate groupings from Table 3.5: individual correlation matrices for LHS primaries
(a), RHS primaries (b), and LHS-RHS pairs (c). The inner elements are surrounded by a dark border.................................................................................................................................22
Table 3.6: Content summary for image set $S_B$..................................................................................26
Table 3.7: Correlation matrix for primary elements, using the Pearson coefficient. The entries are divided into two columns: the unshaded columns are data calculated using the $S_A$ SIFT vectors, while the shaded columns are data derived from the $S_B$ vectors. As in Table 3.4, the inner primary elements are indicated with a bold border..........................................
Table 4.1: Transformation types and ranges used for the random framework. For all trials, the transformation types are selected randomly from the six listed, and the extent of the transformation is chosen randomly from the associated range.................................................................30
Table 5.1: The percentage of the $S_A$ database keypoints with handedness score within $3\delta$ of the $h=0$ threshold, before and after the IPR filtering stage.................................................................................................44
Table 5.2: SIFT matching trial results using $S_t$ and the calibration framework. The handedness splitting strategy, combined with high-IPR keypoint filtering, is compared against a linear search.................................................................................................................................45
Table 5.3: SIFT matching trial results using $S_t$ and a pre-defined set of image transforms. The hierarchical approach to Euclidean-distance matching is compared against a linear search.................................................................................................................................49
Table 5.4: SIFT-HHM compared with PCA-SIFT, SURF and a linear search, using the PCA-SIFT training images (a) and using our trial image dataset, $S_t$ (b)........................................................................50
Table 6.1: Standard deviations for first 16 elements of $e_{PCV}$

Table 6.2: The number of bins for each index is determined by the size of the width elements. The thresholds 50 and 100 represent one and two standard deviations of the PCV element, respectively.

Table 6.3: The $\delta e_{PCV}$ values, bin widths and the number of bins at each PCV index $i$, computed from random transformation trials on our $S_A$ image archive.

Table 6.4: Algorithms (and associated parameters) chosen by FLANN given SIFT databases of 100,000, 1 million and 2 million keypoints and a range of target precision values (0.6 to 0.9).

Table 6.5: Interpolated speedup factors for FLANN and SIFT-BHHM on all three datasets, using 60% and 90% of the F1 score achieved by means of a linear search as performance points.

Table 6.6: Average query times for a 1000-feature search (the number of SIFT features in a typical image). For FLANN and SIFT-BHHM, the matching performance for the times given is 90% of that achieved by means of a linear search.

Table 6.7: Average build times for FLANN and SIFT-BHHM for the SIFT100K, SIFT1M and SIFT2M databases.
List of Figures

Figure 2.1: At left, creation of scale-space representation in multiple octaves; at right, subtraction between adjacent scales creates difference-of-Gaussian images from which extrema are detected........................................................................................................................................5

Figure 2.2: Target pixels (shown with an X) in difference-of-Gaussians images are compared with every neighbour at the current and adjacent scales to find minima and maxima............6

Figure 2.3: The SIFT keypoint descriptor is created from 16 4x4 pixel gradient sub-regions around the keypoint's center. Each gradient in the sub-region is assigned to an orientation histogram bin, at right (the keypoint's orientation is shown by the bold arrow leaving the center)........................................................................................................................................8

Figure 3.1: Inner and outer elements of the SIFT keypoint image patch. The medial line splits the image patch in half horizontally................................................................................................................................16

Figure 3.2: Mean (top) and standard deviation (bottom) of SIFT descriptor indices, taken from nearly 8.5 million keypoint samples.........................................................................................................................16

Figure 3.3: The eight greatest standard deviation peaks from the graphs in Figure 3.2 all correspond to indices aligned with the keypoint's orientation, as demonstrated by the bold arrow leaving the center........................................................................................................................................17

Figure 3.4: Secondary (a) and tertiary (b) descriptor elements........................................................................................................20

Figure 3.5: Primary elements from Chapter 4.2 shown for reference, with medial line indicated. .21

Figure 3.6: Histogram of $S_t$ keypoints, binned by handedness score (bin size = 10)..................23

Figure 3.7: Histogram of $S_t$ keypoints, binned by handedness score. Keypoint distributions with $IPR \leq 0.235$ and $IPR > 0.235$ are indicated.....................................................................................................................25

Figure 3.8: Comparison of mean (a) and standard deviation (b) of SIFT descriptor indices, using independent image sets $S_t$ and $S_b$........................................................................................................26

Figure 3.9: Histogram of both $S_t$ and $S_b$ keypoints, binned by handedness score (bin size is $h=10$): (a) all keypoints; (b) keypoints with $IPR > 0.235$..........................................................27

Figure 4.1: The ten trial dataset images ($S_t$)....................................................................................29

Figure 4.2: The four transformations that make up the calibration framework. Images are (a)
rotated clockwise by 45°, subject to a contrast increase by 10% of the pixel range (b), scaled by a factor of 1.2 (c), and compressed to a JPEG quality value of 50 (d).

Figure 4.3: The hypersphere around a target point (indicated by the X) is illustrated, with the radius of the hypersphere equivalent to the maximum magnitude of the error vector. Potential matches for X must reside inside the hypersphere boundary.

Figure 4.4: Distribution of correctly-matched $S_{t}$ keypoint vector pairs according to the magnitude of the error vector, after having applied transformations using the random framework. The $3\delta$ mark, which represents the threshold within which 99.5% of the vector pairs are found, is indicated by the dotted vertical line.

Figure 4.5: Trials to determine an appropriate threshold for the error vector: recall (a), precision (b), F1 score (c) and matching time speedup over linear search (d) are shown for a range of thresholds. The chosen limit (250) corresponds to the intersection of the F1 scores in (d) and is indicated by the dotted vertical line on all graphs.

Figure 5.1: Pseudo-code description for the handedness-splitting module.

Figure 5.2: Pseudo-code description for the hierarchical Euclidean-distance matching module.

Figure 5.3: The number of correctly-matched SIFT keys with respect to the total number of keys (a) and the incorrectly-matched keys (b), for a range of IPR values, from the $S_{t}$ random transformation trials (Chapter 4.5).

Figure 5.4: Matching performance (a) and the percentage of keypoints dropped (b) as a result of filtering out keypoints that exceed the given IPR threshold. When the IPR threshold is set to 0.235, matching performance is maximized (this point is indicated by the vertical dotted line in both graphs).

Figure 5.5: The absolute value of the handedness error ($e_{h}$) from all correctly matched keypoint vector pairs in the $S_{t}$ transformation trials. Mean and mean plus three standard deviations (signifying the threshold containing 99.5% of the matched pairs) values are indicated.

Figure 5.6: Calibration trials for finding an appropriate primary threshold for hierarchical Euclidean-distance matching (HEDM). Matching performance (a) and speedup factor (b) are shown for a range of threshold values. The point at which the F1 score is equivalent to a linear search ($d_{p}=75$, as indicated by the vertical dotted line on both graphs) is chosen as the best threshold.
Figure 5.7: Calibration trials for finding appropriate secondary (a) and tertiary (b) thresholds for hierarchical Euclidean-distance matching (HEDM). The point at which the F1 score is equivalent to a linear search ($d_s=95$ in (a) and $d_t=168$ in (b), as indicated by the vertical dotted lines on both graphs) is chosen as the best threshold.

Figure 6.1: The principal component vector elements are normalized to an approximately-Gaussian distribution with mean of 127.5 and $\delta = 50$. The distribution of the first principal component ($PCV_0$), is shown.

Figure 6.2: Distribution of the first four elements of $|e_{PCV}|$ ($e_{PCV0}$, $e_{PCV1}$, $e_{PCV2}$, $e_{PCV3}$), computed from the random $S_A$ transformation trials.

Figure 6.3: Bin boundaries for PCV element $i$ are determined by the chosen width element $w_i$. No bin maximum is allowed to be less than 27.5 and no bin minimum is allowed to be greater than 227.5 (these values correspond to the $2\delta$ range of all PCV elements, and are indicated by the vertical dotted lines).

Figure 6.4: SIFT Keypoint A fits within the boundaries of bin 2 for index $PCV_0$, bin 3 for $PCV_1$, and bin 1 for $PCV_2$. The bin numbers for each index are saved into the bin insertion array.

Figure 6.5: SIFT query vector $Q$ and search width $W$ are provided for a database search. The bins that fall within the search window at every PCV index are indicated in grey, and the bin retrieval arrays hold the possible bin combinations.

Figure 6.6: Search speed versus matching performance (a, c, e) and bin insertion (b, d, f) graphs for calibration trials using databases of 100,000 (a, b), 1 million (c, d) and 2 million (e, f) SIFT keys. On graphs (a), (c) and (e), the F1 score achieved by means of a linear search is indicated with a vertical dotted line.

Figure 6.7: SIFT-BHHM and FLANN algorithms are compared, on a logarithmic scale, in terms of matching performance (F1 score) and search speed, using the SIFT100K (a), SIFT1M (b) and SIFT2M (c) databases. The search width ($W$) value and the target precision ($pr$) are indicated for the SIFT-BHHM and FLANN trials, respectively. The linear-search F1 score is marked by the vertical dotted line on all three graphs.
Chapter 1

Introduction

Fundamental to most machine vision and image registration applications is the robust and distinct representation of salient image features. In recent years, local approaches, which do not require any image segmentation and are robust to occlusion, have gained prominence. Typically, such features consist of two parts: a location and a descriptor. The location precisely identifies the 2D or 3D coordinates at which the feature resides; the descriptor gives the feature a unique signature which can be used to find a match for it in another image. The challenge with feature extraction is thus twofold: first, to establish a method for finding image regions that is stable enough to be consistent across a broad range of image transformations; second, to describe these regions in a distinct enough fashion such that they may be reliably recognized.

The SIFT algorithm (for Scale-Invariant Feature Transform) has been used for content-based image retrieval [25, 27, 28, 33, 38, 39], video event classification [8, 31, 37], object recognition [1, 12, 36, 41], object tracking [29, 30], image classification [9, 26, 34, 35], markerless motion capture [42], building panoramas [40, 44], mobile surveillance [43, 45], and face authentication [32], among other applications. It stands out among local feature descriptors for its invariance to scale, rotation and linear illumination, and its partial invariance to 3D viewpoint change [5]. The widespread use of SIFT may be attributed to both its success at localizing invariant interest points in position and scale, using the difference-of-Gaussians detector, and the distinctiveness of its 128-element keypoint descriptor, which is derived from a 16 x 16 pixel gradient patch centered on the image feature location.

Despite its robustness, SIFT use is somewhat limited in online and real-time application spaces because the features take a relatively long time to compute (around 1 second for a typical image with 1,000 SIFT features), and finding correspondences for the 128-dimensional SIFT vectors is computationally expensive given that matches are determined from the Euclidean distance metric.

Our methods aim to improve SIFT matching times in two specific application areas. The first area concerns real-time applications such as object tracking and video event detection. For these applications, we propose a matching technique called SIFT-HHM (for Handed-Hierarchical Matching) that exploits certain repeatable properties of the SIFT descriptor vector to find shortcuts to the most likely
matches in a feature set. Secondly, we consider the content-based image retrieval scenario, which requires that feature matches be quickly found from a very large image archive. We propose in these cases to sort a SIFT keypoint archive into bins using the principal components of the SIFT descriptor vector as the binning criteria, and SIFT-HHM to find the matches to a query point once the closest bins have been retrieved.

1.1 Motivation

With the vast multitude of websites that use streaming video content, there is need for intelligent video processing that can be done in real time. At their lowest level, all such applications require a method for extracting information from image pixel data, and comparing this data for successive frames. SIFT is, for the most part, considered too slow to compute and to compare for these application spaces. Our SIFT-HHM technique allows an application to make use of the robustness of SIFT while matching the features in less than 7% of the time it takes for a linear search.

In recent years, the trend that has seen Internet users upload their personal digital photographs to online albums – rather than store them on a hard drive – has exploded. At the same time, Internet search engines are looking to improve the intelligence of their image-search capabilities by incorporating image content into the search. For these applications, it is not as necessary that image features be extracted quickly, as this can be done offline or upon upload. The challenge is, then, to match the content of a “query” image (or query feature set) to a very large image archive containing millions of image features, and to return the results within a reasonable amount of time (less than a second, or a few seconds, depending on the size of the archive and the precision and recall required of the search). Our PCA-based binning algorithm is able to find correspondences for all SIFT features in a typical image from a database with about 2 million features in approximately 0.13 seconds, which is about three times faster than the current state of the art.

1.2 Problem Statement and Scope

The intent of this thesis is to improve SIFT matching times for a wide range of applications without compromising matching performance as measured by recall and precision. As our approach is general, we do not concern ourselves with specific video processing or image-search applications; the parameters of our algorithms, however, may be tuned to suit individual uses.
1.3 Contributions

The main contributions of this thesis are:

- A modular technique, called SIFT-HHM, for finding SIFT correspondences in consecutive image frames in less than 7% of the time of a linear search. A conference paper describing SIFT-HHM has been accepted to the 2009 IEEE Workshop on Applications of Computer Vision (WACV) under the title “Efficient SIFT matching from keypoint descriptor properties.”

- A method for filtering out less-distinct features (and thus, reducing the likelihood of mismatches) by examining the contributions of four of the 128 SIFT descriptor elements (IPR filtering).

- A PCA-based binning algorithm that also uses SIFT-HHM at the lowest level and finds SIFT correspondences in large databases, at about a three-fold speed improvement over the existing state of the art for SIFT matching, without compromising recall and precision. The binning approach is generic, and is likely to produce similar results using other feature descriptors (i.e. SURF), provided that they exhibit similar statistical properties to SIFT.

1.4 Thesis Overview

This thesis is divided into seven chapters. In Chapter 2, we provide a thorough background on SIFT, principal component analysis (PCA), and the current state of the art for local feature detection/description and approximate nearest-neighbour search in high-dimensional spaces. Chapter 3 discusses the salient aspects of the SIFT descriptor vector that are used to construct our SIFT-HHM scheme. In Chapter 4, we present our experimental framework and introduce the error vector, which is used to quantify the distortion of feature vectors due to image transformations. The SIFT-HHM algorithm is described in Chapter 5, and the PCA-based binning technique for matching to large databases is covered in Chapter 6. In Chapter 7, we present our conclusions and suggest directions for future work on this topic.
Chapter 2

SIFT and Related Work

In this chapter we first provide a thorough overview of the scale-invariant feature transform [1]: its method for finding scale-invariant interest points (Chapters 2.1.1 and 2.1.2), the construction of the keypoint descriptor vector from local gradient orientation histograms (Chapters 2.1.3 and 2.1.4), and the method by which SIFT vectors are compared against one another to find correspondence between images (Chapter 2.1.5). We then describe principal component analysis (PCA), a statistical technique for reducing the dimensionality of a vector space. PCA has been used effectively to construct local descriptors [2, 5], and forms the basis of our binning strategy for large keypoint databases (Chapter 6). Finally, we discuss previously published work in two main areas with which our efforts have the most in common. The first such area concerns newer, SIFT-derived (or SIFT-inspired) local descriptors that claim to approach or exceed the matching performance of SIFT while at the same time being faster to compute and to match. While we do not create a new descriptor, but rather improve the matching step for SIFT, we demonstrate in Chapter 5 that our algorithm will find correspondences faster than the fastest state-of-the-art local descriptors. The second area of related work deals with finding nearest neighbours (or approximate nearest-neighbours) in high-dimensional spaces, such as the 128-element SIFT vector. As a general problem, there is a great deal of literature on the subject; most methods involve constructing a hierarchical data structure like a kd-tree. We focus mainly on the methods applied specifically to SIFT. The results from the trials using our PCA-based binning algorithm (Chapter 6) show that our method is faster at comparable levels of accuracy than all previously published methods for matching SIFT keys large databases.

2.1 SIFT Algorithm

SIFT is implemented as a four-stage cascade-filtering algorithm: the first two stages deal with the localization of scale-invariant interest points; the last two stages construct a rotation-invariant descriptor from local image gradients. The keypoint-matching algorithm uses the Euclidean distance between SIFT descriptor vectors to find a match. While it is important to note that many parameters of SIFT can be customized (such as the gradient magnitude threshold for creating separate features for the
same location, the size of the gradient patch from which the descriptor is generated, or the size of the descriptor vector itself) to suit different applications, we focus here on the standard SIFT implementation as described in [1], and the corresponding source code\(^1\), which is by far the most popular version.

### 2.1.1 Scale-Space Extrema Detection

The goal of keypoint detection is to identify locations and scales that can be repeatably found in different views of the same object or scene. For scale-invariant points, it is important to search not only every possible location in the image, but also across all possible scales, using a continuous function known as scale space [14]. Lindeberg [15] showed that scale space must be created by a Gaussian function. SIFT scale space, then, is created by repeatedly convolving an input image with a Gaussian kernel of increasing \( \delta \). After every octave, or doubling of \( \delta \), the image is downscaled by a factor of two and the blurring iterations are re-started. This process is illustrated on the left side of Figure 2.1.

![Scale-Space Representation](image)

**Figure 2.1**: At left, creation of scale-space representation in multiple octaves; at right, subtraction between adjacent scales creates difference-of-Gaussian images from which extrema are detected.

Once the scale space has been constructed, adjacent scale representations are subtracted (as shown on the right side of Figure 2.1), effectively implementing the difference-of-Gaussians (DoG) function

---

\(^1\) Available at [http://www.cs.ubc.ca/~lowe/keypoints/]
that approximates the Laplacian-of-Gaussian used to find interest points in blob detection algorithms [15].

Scale space is then searched for local minima and maxima, which serve as initial feature point locations. To qualify as a possible feature point, a given DoG pixel needs to be either greater than or less than all pixels in its immediate neighbourhood as well as all pixels in corresponding neighbourhoods in adjacent DoG images (as shown in Figure 2.2).

![Figure 2.2](image)

Figure 2.2: Target pixels (shown with an X) in difference-of-Gaussians images are compared with every neighbour at the current and adjacent scales to find minima and maxima.

### 2.1.2 Keypoint Localization

The second stage of SIFT refines the results obtained from finding the scale-space minima and maxima. Keypoints, defined by 3-dimensional coordinates \((x, y, \delta)\) representing their physical pixel location and the scale at which they were found, are fitted to a 3D quadratic function to determine the interpolated sub-pixel location of the extreme point. This provides a substantial improvement in stability and matching, according to [1]. The value of the quadratic at the extremum is also useful for rejecting unstable points with low contrast.

Finally, it is necessary to remove keypoints located along edges, as these points have poorly-defined peaks – and thus are hard to localize – in the difference-of-Gaussians function. A ratio of principal curvatures, computed from the Hessian matrix in a method borrowed from Harris and Stephens [16], is used to threshold the points located along edges. In Chapter 3, we show how this filtering operation effectively creates the “handedness” characteristic distribution that allows us to split keypoint databases.
into left-handed and right-handed bins.

2.1.3 Orientation Assignment

After a keypoint has been accurately located in pixel (x-y coordinates) and scale space, it is assigned an orientation based on the most dominant direction of the gradients in the local image patch. An orientation histogram is formed from the gradient orientations of sample points within a circular region around the keypoint centre at the closest scale (using the discrete scale-space images from the first stage in the algorithm). The histogram consists of 36 bins, representing the 360° range of angles in 10° increments. Each sample is weighted both by its gradient magnitude and its distance to the keypoint centre, expressed as a circular Gaussian window centred on the keypoint with δ equal to 1.5 times the keypoint's scale. The largest bin in the histogram is chosen as the keypoint orientation; however, if other bins come within 80% of this peak value, separate keypoints are created using these next-most dominant orientations at the same location in pixel and scale space. Finally, a parabola is used to fit to the histogram values surrounding the peak to interpolate the orientation with better accuracy.

Since the keypoint's descriptor vector is constructed from local gradients that are defined with respect to the keypoint orientation, the orientation assignment step serves to anchor the keypoint, providing stability against image rotation.

2.1.4 Descriptor Construction

The fourth, and final, stage of SIFT uses the precise location (in x-y and scale space) and orientation of the keypoint to construct its unique signature vector. The gradient samples from a 16 x 16-pixel rectangular area centred on each keypoint at the closest scale are rotated with respect to the keypoint orientation, to achieve rotation invariance. Furthermore, gradient magnitudes are weighted using a Gaussian window centred on the keypoint with δ equal to half the width of the rectangular area, to give greater emphasis to those gradients that are located closer to the keypoint centre.

The local gradient patch is then divided into 4 x 4 sub-regions, consisting of 16 pixels each, and each sub-region is subsequently characterized by the gradient contributions to an 8-bin orientation histogram, as shown in Figure 2.3. To avoid boundary effects, weighted gradients are also distributed into adjacent sub-region bins through linear interpolation.

The concatenation of the 16 orientation histograms creates SIFT's 128-element (16 sub-regions x 8 orientation bins) descriptor vector.
Following this, two important post-processing techniques are carried out to address specific robustness issues: first, the entire descriptor is normalized to unit length, which has the effect of rendering it invariant to linear illumination (uniform contrast) changes; second, any descriptor elements constituting more than 20% of the total descriptor length are cut off at the threshold and the descriptor is re-normalized. The latter helps to reduce the importance of large gradient magnitudes resulting from nonlinear illumination changes on the keypoint's matching performance. Finally, the descriptor is multiplied by 512 to put its elements within a byte (0-255) range.

2.1.5 SIFT Matching

Matching SIFT keypoints between images or databases involves computing the Euclidean distance between descriptors for every keypoint and every potential match. The algorithm uses a nearest-neighbor distance ratio metric that declares a match only if the closest keypoint distance is less than 0.6 times the next-closest keypoint distance. This scheme effectively identifies correct matches while minimizing false positives [5].

To find a match for a single keypoint, this algorithm must compute 128 multiplications, additions and subtractions for every potential match in the database. This can become prohibitively time consuming for larger keypoint repositories. To address this, many fast approximate nearest-neighbour algorithms have been proposed, such as Best-Bin First proposed by Lowe in [1], that significantly speed up matching times. These algorithms will be discussed more fully in Chapter 2.4.
2.2 Principal Component Analysis

Principal component analysis [17] is a standard dimensionality-reduction technique that has been applied to a number of varied computer vision problems, including object detection [19] and tracking [18], and background modeling [20]. Although it has a number of documented limitations [21], such as the assumption of linearity (that vectors can be properly expressed as linear combinations of a certain basis) and a Gaussian distribution among vector elements, it is still popular on account of its simplicity. It has also been used successfully to create SIFT-based local feature descriptors PCA-SIFT [2] and the Gradient Location and Orientation Histogram (GLOH) [5]. In Chapter 6, we present a novel method for finding SIFT vector nearest-neighbour correspondences in large keypoint databases. Our method consists of creating a series of hierarchical bins, and the SIFT principal component vector is used as the binning criteria. Here, we give an overview of the technique and its implementation via the covariance method.

The goal of PCA is to reduce the dimensionality of a feature space into a linearly orthogonal set of basis vectors. The basis vectors are ordered in terms of their contribution to the variance of the data set: the first vector (the principal component) represents the axis along which the data shows the greatest variance, the second vector represents the second-most varying axis, and so on. The ordered basis vectors are then used to project the original data into the new orthogonal coordinate system. Given that high-dimensional spaces create problems for many computer vision applications (nearest-neighbour search being but one example), there is great motivation for reducing the size of feature vectors.

Expressed mathematically, PCA projection transforms a data set $X$ of dimension $M$ into a data set $Y$ of dimension $L$, where $L < M$, using the Karhunen–Loeve transform:

$$ Y = KLT \{ X \} $$

The KLT transformation matrix can be created by finding the $L$ most important eigenvectors of $X$, as follows. First, the mean and standard deviation of each element in $X$ are found and used to create the $M \times M$ covariance matrix $C$, according to the derivation:

$$ C_{i,j} = \text{cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)], i, j \in (1, M) $$

(2.2)

where $\mu_i = E[X_i]$ and $\mu_j = E[X_j]$ are the means of vector indices $X_i$ and $X_j$, respectively. Next, the covariance matrix is diagonalized – thus removing the linear dependence among the matrix elements –
by the eigenvector matrix $V$:

$$V^{-1} C V = D \quad (2.3)$$

where $D$ is the diagonal matrix containing the eigenvalues of $C$. The matrix $V$ is then ordered by decreasing eigenvalue (according to the rows of $D$), and matrix $W$ is created from the first $L$ rows of $V$. $W$ contains the linear projections for the $L$ basis vectors that show the most variation in the data set $X$. Before computing the projection, however, the data set must be shifted (to have zero mean) and normalized by standard deviation, as follows:

$$Z = \frac{(X - \mu_X)}{\sigma_X} \quad (2.4)$$

Finally, the KLT transform can be computed using the adjusted data set $Z$ and the projection matrix $W$:

$$Y = W \cdot Z = KLT \{X\} \quad (2.5)$$

### 2.3 SIFT-Related Local Detectors and Descriptors

The popularity of SIFT has inspired many refinements on its basic detection and description scheme. The SIFT-related algorithms attempt to improve detection repeatability, distinctiveness, computation and matching times, or some combination thereof. Here we provide a summary of these techniques.

Ke and Sukthankar [2] focused on improving the description step while retaining the first three stages of SIFT. Their method, termed PCA-SIFT, samples from a much larger gradient patch and uses principal component analysis to reduce the descriptor to a 36-element vector. Their results show an improvement in recall-precision over standard SIFT on a small number of artificial image transformation trials. However, a second comparative study by Mikolajczyk [5] showed PCA-SIFT to be less distinctive than SIFT. Furthermore, while the lower-dimensional vector improves matching speed approximately by a factor of three, additional computation is required to project the vectors into orthogonal PCA-SIFT space.

Also in [5], Mikolajczyk proposed a similar descriptor called GLOH that added granularity to the orientation histogram bins and again used PCA for data compression. The GLOH descriptor proved to
be more distinctive than PCA-SIFT with the same number of dimensions, but the more complicated processing, combined with the use of PCA, meant that it also took longer to compute than the standard SIFT descriptor.

Grabner et al. [7], claim to achieve an 8-fold speedup in computation time over the standard SIFT binaries. Their detector uses integral images and a mean-filtering approach for the localization step, and integral histograms [11] to compute the descriptor from gradient orientation bins. Scale-space is thus approximated by mean – as opposed to Gaussian – blurring, and the feature point locations are found using the difference-of-means operator. SIFT’s localization post-processing techniques (Chapter 2.1.2) are also omitted to save computation time. While the descriptor achieves comparable results to SIFT on a small image dataset, it is noticeably less rotation-invariant. Also, because the dimensionality of the descriptor is unchanged, keypoint matching times remain a bottleneck.

Speeded-Up Robust Features (SURF) by Bay et al. [4] also uses integral images for the keypoint localization and description stages. Scale-space maxima are found using the determinant of an approximated Hessian matrix constructed from box filters representing Gaussian second-order partial derivatives. The descriptor's 64 constituent elements are made from the sums of the horizontal and vertical Haar wavelet responses from the image patch around the keypoint. To find keypoint correspondences, sign of the trace of the Hessian matrix is used to split the databases in half, and the same-sign keypoints are compared using the Euclidean distance metric on the descriptor vector, effectively doubling the convergence speed. While clearly faster, independent analyses on both indoor [12] and outdoor [13] vision applications show poorer matching performance when compared to SIFT.

Finally, Mikolajczyk and Matas [10] showed that using the Mahalanobis distance, rather than the Euclidean distance, for local descriptor nearest-neighbour matching resulted in improved recall-precision performance and matching speed in tree data structures. Their descriptor, termed M-SIFT (for Mahalanobis-SIFT), transforms original SIFT descriptors into an orthogonal, truncated vector space. Euclidean distance calculations in the transformed vector space correspond to the Mahalanobis distance in the original SIFT vector space. As with PCA-SIFT, however, the projection step makes M-SIFT slower to compute than standard SIFT.

In summary, the SIFT-based local interest point detector and descriptor schemes typically represent a compromise between improved detection repeatability and distinctiveness at the expense of computation speed, or vice-versa. So far, none have emerged as univerally superior to SIFT for a wide range of applications.
2.4 Nearest-Neighbour Search Algorithms for High-Dimensional Spaces

Many SIFT applications, such as content-based image retrieval [25, 26, 27], require finding correspondences against images from a very large archive. In such cases, the time required to compute the SIFT vectors from the test image is not nearly as significant as the time required to find the correspondences in the database. For these applications, a number of tree-based methods for nearest-neighbour binary search have been proposed.

The most widely used method for a nearest-neighbour search involves building a kd-tree [22]. The kd-tree (for $k$-dimensional tree) is a binary space-partitioning scheme that splits a database first along the dimension that shows the greatest variance, then, recursively, along the dimension with the second-most variance, and so on until the data has been split along every axis. If further branching is required, the splitting starts again along the most variant axis. It is well-known that the kd-tree works well for exact nearest-neighbour searches in low-dimensional data, but quickly becomes less effective as the number of dimensions is pushed beyond 10. For the 128-dimensional SIFT vector space, exact nearest-neighbour matching using the kd-tree is clearly impractical.

For a number of applications, however, it may not be necessary to return an exact nearest-neighbour match; an approximate match will suffice. To this end, Beis and Lowe [1, 6] use a priority queue to speed up the kd-tree search by visiting nodes in order of their distance from the query point. Their algorithm, called Best-Bin First (BBF), controls the degree of approximation by placing a limit on the number of nodes examined before returning the best match found. Approximate nearest-neighbour queries against a 100,000 SIFT keypoint database using BBF were reported to achieve a 95% precision rate while reaching a speedup factor of two orders of magnitude over a linear search [1].

Silpa-Anan and Hartley [23] have recently proposed a parallel implementation using multiple randomized kd-trees. Rather than being split along the axes in descending order of variance, as in the classical kd-tree, a random axis is chosen from the $D$ dimensions that show the greatest variance. As in the BBF algorithm, a single priority queue is maintained across all trees, and the search is terminated when a pre-defined maximum number of nodes have been visited.

Additionally, Muja and Lowe [3] implement a variant on the k-means tree analogous to the BBF approximation of the kd-tree. The classical k-means tree [24] is constructed by clustering data points with the k-means algorithm into $k$ distinct groups, then recursively doing the same for each of the
groups. The recursion is stopped when the number of points in a region is smaller than $k$. In [3], the authors propose first performing a single traversal through the tree while maintaining a priority queue of all the unexplored branches in each node along the path. The algorithm then searches the unexplored branches in order of their distance from the query point (as measured by the mean value of the clustered data points at the branch). A limit is set on the total number of nodes to search before returning a match.

The Fast Library for Approximate Nearest-Neighbours (FLANN) [3], a library of approximate nearest-neighbour search algorithms, is publicly-available software that will automatically choose the optimal algorithm for a given dataset and desired precision. Experiments on large SIFT keypoint databases reveal that the best speedup factors can be achieved by using either randomized kd-trees or the hierarchical k-means tree. The query trials indicate that, on a database of 31 million SIFT keys, a speedup factor of better than three orders of magnitude over a linear search is possible while finding 90% of the closest matches. On smaller datasets (for example, 100,000 keypoints), however, such a speedup is only possible with a much lower precision rate (60% or less). In Chapter 6, we demonstrate that our PCA-based binning algorithm substantially outperforms the tree-based algorithms chosen by FLANN, achieving much higher speedup in matching time at equivalent recall and precision levels for databases of 100,000, 1 million and 2 million SIFT keypoints.
Chapter 3

Keypoint Descriptor Properties

As all SIFT keypoints are subject to the same localization technique – as defined by the first three stages of the algorithm reviewed in Chapter 2 – we might expect that the resulting 128-byte descriptor vector, built from the local image patch, has some distinct and repeatable properties. In this chapter, we identify some of these salient characteristics using the support of a very large keypoint database. In particular, all data discussed in this chapter was extracted from a database of nearly 8.5 million keypoints pulled from over 5,000 images (image set A, denoted $S_A$). $S_A$ contains a diversity of complexity, size and content, as summarized in Table 3.1. In Chapter 3.2, we show that there is a great deal of variance among the 128 SIFT descriptor indices in terms of their means and standard deviations. This leads us to suggest a hierarchical grouping into primary, secondary, tertiary and remainder indices in Chapter 3.3. In Chapter 3.4, we introduce the handedness score, a metric created to capture the correlation of four of the eight primary elements (the inner primaries, using the notation of Chapter 3.1), and we further demonstrate that, by filtering out keypoints that do not pass a ratio (the inner primary ratio) test, we are able to create an approximately bimodal distribution suitable for splitting a database in half. Finally, Chapter 3.5 we provide a cross-verification for all metrics using a second large image archive. This chapter sets the foundation for Chapter 5, where we explain how the keypoint descriptor properties revealed here can be exploited to greatly improve SIFT keypoint matching speeds.

<table>
<thead>
<tr>
<th>Image Set</th>
<th>Total Images</th>
<th>Total SIFT Keys</th>
<th>Content (key words)</th>
</tr>
</thead>
</table>

Table 3.1: Content summary for image set $S_A$. 
3.1 Descriptor Notation and Geography

Given that this chapter is an exploration of the useful properties of the SIFT keypoint descriptor, we first establish a simple notation for referring to the vector itself and its 128 constituent indices\(^2\). We use \( \mathbf{v} \) to refer to the SIFT descriptor vector, then, and \( v_i \) to identify its \( i \)th component scalar, as shown in Equation 3.1.

\[
\mathbf{v} = \{v_0, v_1, v_2, \ldots, v_{125}, v_{126}, v_{127}\}
\]  

(3.1)

It is also useful, in the discussion that follows, to define a few terms that refer to the geography of the descriptor, so as to identify regions and descriptor elements with respect to their physical location in the local image patch. We thus define the following terms:

**Inner area:** Those image patch gradient sub-regions (and corresponding descriptor elements) that are closest to the keypoint centre. There are four inner regions and 32 inner descriptor elements. The inner area is highlighted in the shaded area of Figure 3.1.

**Outer area:** Those image patch gradient sub-regions (and corresponding descriptor elements) that are furthest from the keypoint centre. There are 12 outer regions and 96 outer descriptor elements. The outer area is left white in Figure 3.1.

**Medial line:** The line that divides the local image patch in half vertically. It is coincident with the keypoint orientation, as indicated in Figure 3.1.

---

\(^2\) Technically, a SIFT descriptor can contain more – or less – than 128 elements, but for the purposes of this thesis, we concern ourselves only with the most commonly-used, standard 128-byte vector.
3.2 Mean and Standard Deviation of Descriptor Indices

Using $S_A$, we computed the mean and standard deviation of each index in the SIFT descriptor vector, and the results are shown in Figure 3.2. The standard deviation metric is important because, for any nearest-neighbour matching algorithm based on the Euclidean distance between keypoint descriptors, those elements that vary most will carry disproportionately more weight.

![Figure 3.1: Inner and outer elements of the SIFT keypoint image patch. The medial line splits the image patch in half horizontally.](diagram)

![Figure 3.2: Mean (top) and standard deviation (bottom) of SIFT descriptor indices, taken from nearly 8.5 million keypoint samples.](graph)
What is striking about these graphs is the eight very large, symmetric peaks in mean values at certain descriptor indices. The vertical dotted line through the graphs shows that the mean-value peaks also correspond to the indices with the greatest standard deviations. The eight highest standard deviation values are also highlighted in bold at the top of Table 3.2. Note that the greatest $\delta$ (for index $v_{104}$) is more than twice that for the lowest value (index $v_6$). To understand these results, recall that a SIFT keypoint descriptor is made up of 16 8-bin orientation histograms, as described in Chapter 2. Figure 3.3 highlights the 4 x 4 gradient pixel sub-regions and orientation bins corresponding to the descriptor indices with the greatest standard deviations. Evidently, all eight of the elements at the top of Table 3.2 correspond to sub-region gradients aligned to the dominant keypoint orientation. This means that the elements that are in line with the keypoint's orientation contribute more to the distinctiveness of a descriptor vector than the others.
What is perhaps surprising about the eight more important elements is that while the means for the elements that are located closer to the centre of the keypoint (\(v_{40}, v_{48}, v_{72}\) and \(v_{80}\)) are significantly greater than those on the periphery (\(v_{8}, v_{16}, v_{104}\) and \(v_{112}\)), their standard deviations are slightly smaller. The difference in the mean values is likely a result of the Gaussian weighting function that is applied to the gradient patch around the keypoint to give emphasis to those gradients closer to the center. The inner element values are also more likely to be affected by the post-processing technique of limiting all descriptor index values to 20% of the total vector length, which reduces both the mean and standard deviation of these indices.

It is not altogether unexpected that, for a keypoint that is by definition aligned to the most dominant local gradient direction, the most salient descriptor indices, themselves chosen from that same gradient patch, are those whose orientations align with this direction. What is clear, however, is that there is certainly a hierarchy of importance within the SIFT descriptor vector. A corollary to this is that to treat all descriptor indices equally when matching keypoints is sub-optimal.

### 3.3 Hierarchical Descriptor Index Groupings by Standard Deviation

We now attempt to divide the descriptor vector into hierarchical groupings based on the standard deviation ranking of Table 3.3. In Chapter 5, we use these groupings to perform Euclidean-distance calculations hierarchically, filtering out a large majority of potential matches while only examining a small subset of the vector. The delineation of the groupings is somewhat arbitrary, but corresponds approximately to clusters of indices with progressively smaller standard deviations. We refer to the groupings as the primary, secondary and tertiary elements, which are defined in Table 3.4. For completeness, the table also includes the 68 descriptor elements that do not form a part of any of these groups, which we call the remainder indices.
Table 3.3: Primary, secondary, tertiary and “remainder” descriptor index groupings.

<table>
<thead>
<tr>
<th>Grouping</th>
<th># Elements</th>
<th>Constituent Descriptor Index Values</th>
<th>Std. Deviation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary</td>
<td>8</td>
<td>8, 16, 40, 48, 72, 80, 104, 112</td>
<td>53.84 – 48.63</td>
</tr>
<tr>
<td>Secondary</td>
<td>12</td>
<td>9, 23, 36, 44, 52, 60, 68, 76, 84, 92, 111, 113</td>
<td>42.71 – 38.23</td>
</tr>
<tr>
<td>Tertiary</td>
<td>40</td>
<td>0, 4, 15, 17, 24, 28, 32, 34, 35, 41, 42, 43, 47, 49, 53, 54, 55, 56, 61, 62, 64, 69, 70, 73, 77, 78, 79, 81, 82, 83, 87, 88, 90, 91, 96, 100, 105, 119, 120, 124</td>
<td>37.11 – 32.52</td>
</tr>
<tr>
<td>Remainder</td>
<td>68</td>
<td>1, 2, 3, 5, 6, 7, 10, 11, 12, 13, 14, 18, 19, 20, 21, 22, 25, 26, 27, 29, 30, 31, 33, 37, 38, 39, 45, 46, 50, 51, 57, 58, 59, 63, 65, 66, 67, 71, 74, 75, 85, 86, 89, 93, 94, 95, 97, 98, 99, 101, 102, 103, 106, 107, 108, 109, 110, 114, 115, 116, 117, 118, 121, 122, 123, 125, 126, 127</td>
<td>32.5 – 22.6</td>
</tr>
</tbody>
</table>

The primary elements are simply the set of eight indices highlighted in Chapter 3.2 and illustrated in Figure 3.3; they correspond to local image patch gradients aligned to the keypoint orientation, straddling the medial line of the patch. They have (by a clear margin) the greatest standard deviation values in the descriptor vector. The secondary elements are a symmetrical grouping of the 12 indices whose standard deviations are the next-highest, after the primary elements. Their locations within the local image patch are shown in Figure 3.4(a). While none of them are aligned with the keypoint orientation, four of them (elements \(v_9, v_{23}, v_{111}\) and \(v_{113}\)) are adjacent to it and eight (elements \(v_{36}, v_{44}, v_{52}, v_{60}, v_{68}, v_{76}, v_{84}, \) and \(v_{92}\)) oppose it. After the secondary elements, the standard deviation values drop off more slowly and thus more indices are contained within a smaller \(\delta\) range. Choosing a third grouping, then, is a great deal more arbitrary. However, given that there exists a significant disparity in \(\delta\) values between the bottom of the secondary grouping (38.23) and the 128\(^{th}\)-ranked element (22.6), we hypothesize that a third hierarchy level could prove useful in reducing the computational demand of Euclidean-distance calculations. Our tertiary grouping of 40 elements is illustrated in Figure 3.4(b). Together, the primary, secondary and tertiary elements comprise nearly half (60 elements in total) the SIFT descriptor vector. The matching trials of Chapter 5 demonstrate that most potential keypoint matches can be ruled out after having examined only one, two or all three of the tiers shown in Table
3.4. This results in a speedup factor of at least an order of magnitude over an exhaustive search for all trials.

![Figure 3.4: Secondary (a) and tertiary (b) descriptor elements.](image)

**3.4 Handedness and the Inner Primary Ratio**

Although the results from Figure 3.2 might have been expected given the nature of SIFT keypoint localization, another, significantly less obvious, characteristic of the descriptor becomes apparent when considering the linear relationships between the primary elements. To measure the correlation between two elements X and Y, we use Pearson's product-moment coefficient, $\rho_{(X,Y)}$, defined as follows:

$$
\rho_{(X,Y)} = \frac{\text{cov}(X, Y)}{(\sigma_X \sigma_Y)} = \frac{E(((X - \mu_X)(Y - \mu_Y)))}{(\sigma_X \sigma_Y)} \tag{3.2}
$$

where $\text{cov}(X, Y)$ represents the covariance of elements X and Y, and where $\mu_X$, $\sigma_X$, $\mu_Y$ and $\sigma_Y$ represent their means and standard deviations, respectively. The Pearson coefficient ranges from -1.0 (complete inverse linear dependence) to 1.0 (complete positive linear dependence). A result of 0 indicates that the elements are linearly independent.

A correlation matrix for the eight primary elements, using the nearly 8.5 million keys in $S_d$, is displayed as Table 3.4. Note that the correlation matrix is equivalent to the covariance matrix normalized to the standard deviation of the row and column elements.
Table 3.4: Correlation matrix for primary elements, with pairings that cross the medial line highlighted in grey, and the elements closest to the keypoint patch centre indicated with a bold border.

<table>
<thead>
<tr>
<th></th>
<th>V8</th>
<th>V16</th>
<th>V40</th>
<th>V48</th>
<th>V72</th>
<th>V80</th>
<th>V104</th>
<th>V112</th>
</tr>
</thead>
<tbody>
<tr>
<td>V8</td>
<td>1.000</td>
<td>0.395</td>
<td>0.677</td>
<td>0.177</td>
<td>0.481</td>
<td>0.055</td>
<td>0.425</td>
<td>0.150</td>
</tr>
<tr>
<td>V16</td>
<td>0.395</td>
<td>1.000</td>
<td>0.206</td>
<td>0.677</td>
<td>0.080</td>
<td>0.492</td>
<td>0.165</td>
<td>0.437</td>
</tr>
<tr>
<td>V40</td>
<td>0.677</td>
<td>0.206</td>
<td>1.000</td>
<td>-0.123</td>
<td>0.786</td>
<td>-0.186</td>
<td>0.480</td>
<td>0.071</td>
</tr>
<tr>
<td>V48</td>
<td>0.177</td>
<td>0.677</td>
<td>-0.123</td>
<td>1.000</td>
<td>-0.180</td>
<td>0.802</td>
<td>0.065</td>
<td>0.491</td>
</tr>
<tr>
<td>V72</td>
<td>0.481</td>
<td>0.080</td>
<td>0.786</td>
<td>-0.180</td>
<td>1.000</td>
<td>-0.123</td>
<td>0.676</td>
<td>0.205</td>
</tr>
<tr>
<td>V80</td>
<td>0.055</td>
<td>0.492</td>
<td>-0.186</td>
<td>0.802</td>
<td>-0.123</td>
<td>1.000</td>
<td>0.181</td>
<td>0.676</td>
</tr>
<tr>
<td>V104</td>
<td>0.425</td>
<td>0.165</td>
<td>0.480</td>
<td>0.065</td>
<td>0.676</td>
<td>0.181</td>
<td>1.000</td>
<td>0.396</td>
</tr>
<tr>
<td>V112</td>
<td>0.150</td>
<td>0.437</td>
<td>0.071</td>
<td>0.491</td>
<td>0.205</td>
<td>0.676</td>
<td>0.396</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Figure 3.5: Primary elements from Chapter 4.2 shown for reference, with medial line indicated.

The primary element locations from Figure 3.3 have been reproduced here as Figure 3.5 with the medial line indicated, for easy reference. Since all of the primary elements represent gradients aligned to the same direction (the assigned keypoint orientation), it is unsurprising that there is, generally, positive correlation between them. An interesting pattern appears, however, when we highlight in grey those element pairs whose sub-regions lie on opposite sides of the medial line of the image patch. It appears that these particular groupings are the least-correlated of the primary elements. To illustrate this further, Table 3.5 separates Table 3.4 into three smaller tables, identifying the correlation between those...
elements that lie to the left (a) and right (b) of the medial line, respectively. We refer to these groupings as the LHS and RHS primaries. The third table (Table 3.5(c)) shows the correlation between the LHS-RHS pairs. It is clear from these tables that the greatest linear dependence occurs between primary elements on the same side of the medial line, with surprisingly less correlation – and in some cases, inverse correlation – between elements that cross it. This observation is particularly acute when considering the quartet of primary elements that are closest to the centre of the image patch (v_{40}, v_{48}, v_{72} and v_{80}). We call these elements the inner primaries, according to the definition provided in Chapter 3.1, and they are indicated with a bold border in Tables 3.4 and 3.5. Interestingly, v_{40} varies almost completely with v_{72} (ρ = 0.786), but is inversely related to v_{48} (ρ = -0.123), which is equidistant and represents the same gradient orientation, but lies across the medial line. The same is true of the other three elements forming this inner group.

![Table 3.5](image)

(a)  (b)  (c)

Table 3.5: Separate groupings from Table 3.5: individual correlation matrices for LHS primaries (a), RHS primaries (b), and LHS-RHS pairs (c). The inner elements are surrounded by a dark border.

We next present a metric we refer to as the keypoint handedness score (h), that yields a surprising distribution when applied to a large database of SIFT keys:

\[
    h = (v_{48} + v_{80}) - (v_{40} + v_{72})
\]

Simply put, the handedness score is the difference between the sums of the like-handed inner primaries. A negative handedness score indicates that the left-handed elements (v_{40} and v_{72}) dominate, whereas the converse shows a right-hand side dominance. We computed the handedness score on the full set of keypoints in S_a, and observed the distribution shown in Figure 3.6.
Two things immediately stand out from this graph: first, the anomalous large density spike around $h=0$, which consumes over 5% of the database keypoints; and second, the approximately symmetrical distribution with two distinct local maxima at around $h = \pm 160$. Excluding the one large spike, there also appears to be a density valley that extends from about $h=-100$ to $h=100$.

Aside from the $h=0$ spike, it appears that the data is fairly evenly split between left-side dominant (left-handed) and right-side dominant (right-handed) keypoints. The dual maxima at $h = \pm 160$ and central valley confirms the inverse relationship between the LHS and RHS elements highlighted in Table 3.4. This seems counter-intuitive: why would these elements, created from adjacent regions of the image patch and representing the same gradient orientations, not vary together? The answer lies in the second stage of the SIFT algorithm (Chapter 2.1.2), where local scale-space extrema are rejected due to poor contrast or if they are located along edges. An edge manifests itself as a consistent gradient that spans the width of the image patch. If patches containing edges were to bypass the second SIFT stage, the keypoint would be oriented to the edge gradient and there would be a high degree of correlation between the LHS and RHS primary elements. Because of the second-stage filtering, however, the dominant gradient cannot span the width of the image patch, and most often it is very unevenly split along the medial line, resulting in the distribution shown in Figure 3.6.

To explain the central spike, we refer again to the post-processing technique of limiting all descriptor components to 20% of the vector magnitude. The effect is to reduce the emphasis on the most dominant gradients in the image patch and to give, consequently, more influence to the smaller
gradients. By construction, the most dominant gradients are aligned with the keypoint orientation, as we saw in Figure 3.2. The primary elements closest to the keypoint centre (the inner primaries), from which the handedness score is calculated, are the elements most frequently limited by this 20% threshold. We hypothesize, then, that the majority of the keypoints that constitute the central spike in Figure 3.4 are those whose inner primary elements are all very large and have, consequently, been limited by the threshold. The inner primaries thus cancel each other out, which results in a handedness score of near 0.

To verify this hypothesis, we introduce another metric, the *inner primary ratio* (IPR), that represents the magnitude of the inner primary elements in relation to the magnitude of the entire SIFT vector.

\[
IPR = \frac{(v_{40}^2 + v_{48}^2 + v_{72}^2 + v_{80}^2)}{\sum_{0}^{127} v_i^2}
\]  

Larger IPR values indicate a greater influence of the inner primary elements on the descriptor vector, while smaller values indicate the converse. We choose a threshold value of IPR=0.235 to separate the dataset into those vectors that are highly dependent on the inner primaries (IPR>0.235), and those that are less dependent (IPR<=0.235). This threshold corresponds to a matching performance peak for filtering trials that are discussed in Chapter 5.2.3. For now, though, it is simply useful for better understanding the concentrations of keypoints in the handedness score bins. Figure 3.7 is a reproduction of Figure 3.6, but with each bin concentration delineated based on the density of vectors that fall on either side of the IPR threshold.
Clearly, our hypothesis regarding the large number of keypoints near $h=0$ was correct. Nearly 90% of the vectors that constitute the central spike have IPR values that exceed the threshold. In Chapter 5, we demonstrate that there are two specific advantages to be gained by filtering out the high-IPR keys: firstly, these vectors tend to be less distinct in general and therefore match poorly; second, by eliminating them, we are left with an approximately bimodal distribution that can be easily divided in half at the $h=0$ mark. The latter process enables us to approximately double the matching convergence speed.

### 3.5 Cross-verification of metrics using $S_B$

All of the properties of the SIFT descriptor vector discussed in this chapter (and metrics used to describe them) have been established using data from the very large $S_A$ image archive. In order to show that our results are not dependent on the data set, we provide, in this space, a cross-verification of all significant metrics using a second, independent image archive. This archive, which we call Image Set $B$, or $S_B$, contains over 1,000 images and more than 680,000 keypoints, and – like $S_A$ – represents a diversity of complexity and content (the archive's contents are summarized in Table 3.6).
Table 3.6: Content summary for image set $S_B$.

In Figure 3.8, we have plotted the means and standard deviations of the descriptor elements for all SIFT vectors contained in $S_B$, and overlaid them against the $S_A$ results from Figure 3.2. While it appears as though the $S_B$ results show slightly more variance among the indices on both graphs, the pattern of peaks is consistent. In observing the standard deviation graph of Figure 3.8 (b), it appears that our hierarchical groupings from Chapter 3.3 will hold true for this data set as well, with perhaps some minor variation in rank order. In particular, the eight primary element peaks are visible, and indeed even more exaggerated, on the $S_B$ data set.

Table 3.7 displays the correlation between primary elements using the vectors from the $S_B$ image archive (grey columns) alongside the corresponding Pearson coefficients for the $S_A$ data, from Table 3.4. Again, there is some minor variation, but the fact that the matrix entries are largely aligned indicates strongly that our observations regarding RHS and LHS primaries from Chapter 3.4 will prove true regardless of the data set.

![Figure 3.8](image-url)
Table 3.7: Correlation matrix for primary elements, using the Pearson coefficient. The entries are divided into two columns: the unshaded columns are data calculated using the $S_A$ SIFT vectors, while the shaded columns are data derived from the $S_B$ vectors. As in Table 3.4, the inner primary elements are indicated with a bold border.

Finally, Figure 3.9 lays side-by-side the histograms of the $S_A$ and $S_B$ keypoints, binned according to handedness score. In Figure 3.9 (a), all keypoints are binned, while Figure 3.9 (b) plots only those that have high IPR values (IPR > 0.235). The distributions are similar in shape, with the $S_B$ archive clearly containing more high-IPR vectors that contribute to a larger density spike at the $h=0$ mark on both graphs. In Chapter 5, we show that filtering out the high-IPR keys results in improved matching performance. We hypothesize that such an approach would be even more effective on the $S_B$ images than on the $S_A$ images, given the greater number of high-IPR vectors contained in $S_B$.
Chapter 4

Experimental Framework and the Error Vector

This chapter sets up the experimental basis for Chapters 5 and 6, which compare our SIFT matching techniques to the current state-of-the-art methods for fast keypoint matching. We begin by stating our model of the matching process, which is derived from that used by Mikolajczyk and Matas in [10]. In Chapter 4.2, we explain our experimental method with respect to two transformation trial frameworks which we refer to as the calibration framework and the random framework. Chapter 4.3 discusses the performance metrics used to objectively compare methods and parameter settings, and how they are calculated. In Chapter 4.4, we introduce the error vector, which – analogous to the noise signal in telecommunications – represents the distortion of the original SIFT descriptor caused by an image transformation. Finally, we use the calibration framework to determine a practical threshold for the error vector in Chapter 4.5.

4.1 Model of the Matching Process

Since our aim is to evaluate the performance of various techniques for finding corresponding interest points between images under a variety of transformations, we adopt the following model of the matching process, which is similar to that described in [10]. Given an image \( I \) that contains a set \( F_I \) of feature vectors, we apply a transformation to \( I \) to create \( I' \), and from the new image extract feature set \( F'_I \). The problem of finding correspondences, then, is defined as a search for a function \( B: F_I \rightarrow F'_I \cup \nu_0' \) that assigns every \( \nu_i \in F_I \) either a \( \nu_j' \) from \( F'_I \) or \( \nu_0' \), representing no match.

With SIFT matching, the keypoint-pair assignments are based on the closest nearest neighbours by Euclidean distance, according to the method described in Chapter 2.1.5. Implied in this notation is that the matching is performed in one direction only: features in \( F'_I \) are searched to find the best match for a given feature in \( F_I \), and not vice-versa. To constrain the matching process to a true one-to-one correspondence, symmetrical mapping would need to be enforced. However, we focus on one-way matching for two reasons. First, for many applications, like object recognition, the unidirectional mapping is sufficient. Second, the performance of most published
methods for finding nearest-neighbour correspondences [1, 3, 5, 7, 10, 22, 23] are evaluated with respect to one-way matching; it therefore makes sense to employ a similar approach for comparative purposes.

4.2 Transformation Trial Frameworks

For most experiments in this thesis, we have used a set of 10 images which we call \( S \), our trial image dataset. The \( S \) images are shown in thumbnail form in Figure 4.1. They were chosen to reflect a variety of complexity and content: a selection of nature scenes, urban scenes, and fine art are included in the set. We use the \( S \) images to both find the best parameter selections for our matching algorithm, and to compare our approach with existing methods. For the latter, we also run separate analyses using a set of publicly-available images\(^3\). The transformations are designed to simulate real-world phenomena: scale change mimics camera zoom, for example, while contrast changes imitate lighting variance introduced by manipulating the camera aperture. The transformations themselves are artificial to facilitate the homographical mapping of interest points from the original to the transformed images. If the homography can be easily and rapidly calculated, it allows us to perform a greater number of trials on a greater number of images.

For parameterization trials, we put each \( S \) image through the fixed sequence of transformations shown in Figure 4.2. The four transformations (which we call the calibration framework) are applied independently to each image, and the resulting image is matched against the original in each case, for a total of 40 trials for every analysis. Having a pre-defined set of transformations allows us to easily

---

\(^3\) The PCA-SIFT training images, which are available at [http://www.cs.cmu.edu/~yke/pcasift/](http://www.cs.cmu.edu/~yke/pcasift/)
compare the matching results obtained from two different parameter settings (as in trials to find a suitable threshold, for example).

![Figure 4.2: The four transformations that make up the calibration framework. Images are (a) rotated clockwise by 45º, subject to a contrast increase by 10% of the pixel range (b), scaled by a factor of 1.2 (c), and compressed to a JPEG quality value of 50 (d).](image)

While the calibration framework is useful for parameter tuning, we want to ensure that our algorithm performs well regardless of the type of image transformation applied. Our random framework, then, is an attempt to decouple the matching results from the specific transformations used in the calibration process. The random framework uses six different types of transformations, and the type and extent of the transformation is randomized according to the ranges specified in Table 4.1. Each image is typically subject to two successive transformations, and the resulting image is matched against the original in each case. By choosing the transformations randomly, and applying them in succession, there is in general much more distortion of the image in these trials. We use this more rigorous framework to compare our matching algorithm with existing approaches.

<table>
<thead>
<tr>
<th>Transformation Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotation</td>
<td>Any angle</td>
</tr>
<tr>
<td>Scale adjustment</td>
<td>Scaling factor of 0.7 to 1.5</td>
</tr>
<tr>
<td>Contrast adjustment</td>
<td>-20% to 20% of the pixel range</td>
</tr>
<tr>
<td>Affine warp (horizontal shear)</td>
<td>5% to 25% of the image width</td>
</tr>
<tr>
<td>Gaussian noise addition</td>
<td>$\delta=1$ to 10 pixels</td>
</tr>
<tr>
<td>JPEG compression</td>
<td>Quality values of 40 to 80</td>
</tr>
</tbody>
</table>

Table 4.1: Transformation types and ranges used for the random framework. For all trials, the transformation types are selected randomly from the six listed, and the extent of the transformation is chosen randomly from the associated range.
4.3 Metrics

As in [2] and [5], we use recall and precision as the main evaluation metrics, as we want to quantify the adeptness of various approaches at both finding the correct SIFT keypoint matches and reducing the number of false positives. Thus, we define \textit{recall} and \textit{precision} as follows:

\[
\text{recall} = \frac{\text{number of correct matches}}{\text{number of correspondences}} \quad (4.1)
\]

\[
\text{precision} = \frac{\text{number of correct matches}}{\text{total number of matches reported}} \quad (4.2)
\]

A \textit{correct match} is defined as a match returned where the two keypoints correspond to the same image patch, as determined by the homographical mapping between the initial and the transformed images. A 2-pixel tolerance in x-y coordinates is deemed sufficient for identifying a correct match. The \textit{number of correspondences} is pre-determined by examining the x-y coordinates of each SIFT keypoint found in both images. If, for a given keypoint in the original image, there exists a keypoint in the transformed image at the same homographically-mapped location, then a correspondence is registered. In a sense, the number of correspondences can be intuitively thought of as the number of \textit{potential} correct matches, based on the results from the interest-point detection process.

In some cases, we will be examining two different algorithms and one will exhibit better recall but poorer precision than the other, making objective comparison difficult. In such cases we have found it useful to combine recall and precision into a scalar value, the F1 score, that represents a linear combination of the two, where both are evenly weighted. The F1 score is defined as follows:

\[
\text{F1 score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (4.3)
\]

It is noted that some applications are more concerned with recall than precision, or vice-versa. In this thesis, however, we gear our approach towards the general case where they are assumed to be equally important. The F1 score, therefore, is a generic measure of the correctness and thoroughness of the matching results achieved by a certain algorithm. We assume a higher F1 score to represent better results, regardless of the recall-precision balance.
4.4 The Error Vector

It is useful in the discussion that follows to not only count the number of correct matches resulting from a given transformation trial, but also to measure the distortion of the matched feature pairs. We thus introduce the *error vector*, which is analogous to the noise signal in telecommunications processes. If mapping function \( B \) (from our model of the matching process) returns a matched vector pair \((\mathbf{v}, \mathbf{v}')\) according to the nearest-neighbour distance ratio method described in Chapter 2.1.2, and it is confirmed that the vectors represent the same image patch (using the homographical mapping between the initial and transformed images), the error vector, then, is defined as follows:

\[
\mathbf{e} = \mathbf{v}' - \mathbf{v}
\]  

(4.4)

Vector \( \mathbf{e} \) thus represents the resulting distortion of the SIFT descriptor vector incurred by an image transformation. It follows, then, that the magnitude \( \|\mathbf{e}\| \) is the Euclidean distance between the matched feature vectors:

\[
\|\mathbf{e}\| = \sqrt{\sum_{i=0}^{127} e_i^2}
\]  

(4.5)

In the next section, we show that it is possible to find a practical limit to the magnitude of the error vector, beyond which it is highly improbable that keypoints \( \mathbf{v} \) and \( \mathbf{v}' \) represent the same image patch. We also use a similar concept in Chapter 5 (the *handedness error*) to justify splitting a SIFT feature set into left-handed and right-handed bins according to the handedness score, and to determine appropriate bin widths (the *principal component vector error*) for our PCA-based binning algorithm presented in Chapter 6.

4.5 Thresholding the Error Vector

The most successful nearest-neighbour search methods used for finding SIFT matches involve the construction of either kd-trees or k-means trees [5]. Both of these structures are space-partitioning schemes that are designed for quickly isolating the subset of the feature space most likely to contain the closest neighbour to the target point. Similarly, the bulk of our matching algorithm rests on the assumption that there exists only a small 128-dimensional sub-region around a target point within which a nearest neighbour must be found, and an optimal algorithm consists of quickly finding and searching this sub-region for a match.
Here, we assume, for simplicity, that the sub-region is spherical (although the inequality of the descriptor vector elements, as demonstrated by the standard deviation graph of Figure 3.2, indicates strongly that this is not the case) as shown in Figure 4.3. The radius of the sphere in this figure is equivalent to the maximum Euclidean distance between two SIFT keypoint vectors that correspond to the same image patch. From Equation 4.5, this value is equal to the upper limit of the magnitude of the error vector.

Figure 4.3: The hypersphere around a target point (indicated by the X) is illustrated, with the radius of the hypersphere equivalent to the maximum magnitude of the error vector. Potential matches for X must reside inside the hypersphere boundary.

To arrive at a possible maximum for the error vector, we applied the random transformation framework to the entire set of $S_A$ images and saved the correctly-matched SIFT keypoint vectors (a total of nearly 3.8 million keypoint pairs). For each of the correctly-matched vectors, the error vector magnitude was computed, resulting in the distribution shown in Figure 4.4. The mean is zero, since the error vector represents the difference between two identically-distributed sets of vectors. The $3\delta$ mark ($||e|| = 194.9$) indicates the threshold that contains approximately 99.5% of the keypoint pairs examined.
It is apparent from Figure 4.4 that the distance between correctly-matched keypoint pairs is confined to a very small range around the target point, regardless of the transformation. The $3\delta$ value represents less than a single byte range across a 128-byte vector, meaning that the hypersphere illustrated in Figure 4.3 consists of a tiny fraction of the total feature space. We infer from this distribution that SIFT will tolerate only a finite amount of distortion of the image patch, beyond which a keypoint will either not be detected or its descriptor vector will be too different from the original keypoint to match properly. If it is possible to limit the search for a SIFT vector match to the 128-dimensional hypersphere around a target point defined by the magnitude of the error vector, we stand to not only find the correct nearest neighbour with very high probability, but also to greatly reduce the search time.

Expressed quantitatively, if we choose the radius of the hypersphere to be equal to the $3\delta$ value, and noting that the volume of an $n$-dimensional sphere (with $n$ an even number) is given by:

$$V_n = C_n r^n$$

(4.6)

Where

$$C_n = \frac{\pi^{k}}{(k!)} , \text{ for even } n = 2k.$$  

(4.7)
then the volume of the 128-dimensional hypersphere around a keypoint is

\[ V_{128} = \pi^{64} \left( \frac{194.9^{128}}{(64!)} \right) = 9.829 \times 10^{203} \]  

(4.8)

Which seems very large, but when contrasted with the full volume of the SIFT vector space \( 255^{128} = 1.089 \times 10^{308} \), this represents a reduction by a factor of \( 9.026 \times 10^{105} \) of the search space. In practice, the reduction factor is much smaller, given that SIFT vectors are typically unevenly clustered into a much smaller volume of the total SIFT vector space. However, considering that existing methods [1, 5] do not make use of a maximum-distance threshold and are therefore wasting computation cycles evaluating keypoints that are too far from the target to match, we stand to improve matching convergence speed by introducing one.

From the SIFT matching paradigm presented in Chapter 2.1.5, it is not sufficient to simply find the closest nearest neighbour to a given target point: the two closest neighbours must be found, and the ratio between their distances is used to declare a match. We therefore use the calibration framework and the S_t images to find the lowest possible error vector threshold that is still high enough to find the closest two matches to the target point. The keypoints in the transformed image are searched linearly for a match to a given target vector from the original image. Distance calculations are short-circuited if intermediate distances exceed the threshold before all 128 vector elements have been compared. If no vectors are under the distance threshold, a null match is returned. If only one vector passes the threshold test, it is returned as a match if its distance to the target point is less than 80% of the maximum distance. In the case where multiple vectors pass the threshold test, a match is determined by the distance ratio threshold as described in Chapter 2.1.5. The average results from the trials are shown in Figure 4.5. Figures 4.5 (a) and 4.5 (b) show the effect of the threshold on recall and precision, respectively. The inverse relationship between these metrics is clear: lower thresholds favour higher precision at the expense of recall; higher thresholds favour the converse. The F1 score (Figure 4.5 (c)) shows gradually better performance as the threshold is increased. We take the intersection of the Figure 4.5 (c) curves (between the thresholded and non-thresholded trials) as the practical limit of the error vector, as it is the point at which matching performance – as measured by the F1 score – is equivalent to the results obtained without a threshold. This crossing point occurs near \( \|e\| = 250 \), and is indicated by the dotted vertical line on all graphs. According to the results of Figure 4.5, at this threshold, equivalent SIFT matching performance can be retained while convergence times are sped up
by a factor greater than 3 (Figure 4.5 (d)). This speed-up alone is approximately equivalent to that achieved by PCA-SIFT [2] – a result we validate in Chapter 5.

Finally, it is worth noting that, in limiting error vectors to a length of 250, we have essentially reduced the necessary size of the accumulator for storing the closest Euclidean distances. Given that Euclidean distances are typically stored as sum-of-squares (SSD) values, the maximum SSD for \( \|e\| \) is then \( 250^2 = 62,500 \). Thus, the accumulator can be implemented as a 16-bit unsigned register.

Figure 4.5: Trials to determine an appropriate threshold for the error vector: recall (a), precision (b), F1 score (c) and matching time speedup over linear search (d) are shown for a range of thresholds. The chosen limit (250) corresponds to the intersection of the F1 scores in (d) and is indicated by the dotted vertical line on all graphs.

Finally, it is worth noting that, in limiting error vectors to a length of 250, we have essentially reduced the necessary size of the accumulator for storing the closest Euclidean distances. Given that Euclidean distances are typically stored as sum-of-squares (SSD) values, the maximum SSD for \( \|e\| \) is then \( 250^2 = 62,500 \). Thus, the accumulator can be implemented as a 16-bit unsigned register.
Chapter 5

Handed-Hierarchical Matching

In Chapter 3, we highlighted some repeatable characteristics of the SIFT descriptor vector, supported by evidence from millions of keypoint samples. We showed that some vector elements tend to vary far more than others, and are therefore more distinct, as SIFT matching is based on the Euclidean distance metric. We also introduced the “handedness” characteristic, which is a tendency of the inner primary SIFT elements to be strongly weighted to one side of the medial line.

In this chapter, we show how these properties can be used to greatly improve SIFT matching speed while retaining near-equivalent recall-precision performance of a linear search. Our technique, which we refer to as SIFT Handed-Hierarchical Matching (or SIFT-HHM) combines a database split based on the handedness score and a hierarchical approach to Euclidean-distance calculation that makes use of the descriptor groupings from Chapter 3. An overview of the algorithm is presented in Chapter 5.1. The handedness split (Chapter 5.2) and the hierarchical Euclidean-distance matching (Chapter 5.3) modules are then treated in isolation, in order to fine-tune thresholding parameters and examine their effect on matching performance and speed. The evaluation of Chapter 5.4 demonstrates how well the full SIFT-HHM algorithm performs in relation not only to SIFT linear search-based matching, but also when compared to two other popular local descriptor alternatives – PCA-SIFT [2] and SURF [4] – that are designed specifically to be faster and more robust.

5.1 Overview of the Algorithm

Given two SIFT feature sets \( F_I \) and \( F_{I'} \), corresponding to different images \( I \) and \( I' \) of the same scene, the fastest known method for finding all of the correspondences between \( F_I \) and \( F_{I'} \), without building a search tree-type data structure (which is impractical for single-image matching applications), is to compare each vector in \( F_I \) with each vector in \( F_{I'} \) and return the best match (or no match) in every case.

Our modular Handed-Hierarchical Matching technique can be conceived of as a discrete set of methods for short-circuiting the tedious linear search. As each comparison between keypoint vectors requires calculating the Euclidean distance – a computationally expensive process that requires 128 subtractions, additions and multiplications per vector pair – we focus on limiting the necessary
comparisons to a select few high-probability potential matches.

### 5.1.1 Handedness Splitting Module

The handedness splitting module – summarized by the pseudo-code in Figure 5.1 – essentially divides the databases $F_I$ and $F_R$ in two, using the handedness score from Chapter 3 as the binning criteria. The split is preceded by a filtering step that discards all keypoints whose inner primary elements constitute too high a percentage of the entire vector length (this is expressed by the *inner primary ratio*, as defined in Chapter 3.4). The filtering step removes approximately 12% of the keypoints from any SIFT database, and results in overall improved matching performance. If they pass through the filter, the vectors in $F_I$ will be put into either the left-handed ($F_{I\_left}$) or right-handed ($F_{I\_right}$) bin based on individual keypoint handedness scores. Similarly, the $F_R$ vectors will be sent to either $F_{R\_left}$ or $F_{R\_right}$. The matching program will only compare vectors in $F_{I\_left}$ with vectors in $F_{R\_left}$, and $F_{I\_right}$ with $F_{R\_right}$, thus reducing the search space – and, consequently, matching times – by approximately half.

![Handedness Splitting Module](image)

**Figure 5.1:** Pseudo-code description for the handedness-splitting module.

### 5.1.2 Hierarchical Euclidean-Distance Matching Module

The second module makes the vector comparisons between the SIFT keys in first the left-handed, then the right-handed bins. As in the original SIFT matching program, Euclidean distance is used to find the closest, and second-closest, matches. In our algorithm, however, we treat the descriptor vector hierarchically, according to the standard deviation ranking of Table 3.3. The algorithm works in a
cascading fashion, as illustrated in the pseudo-code of Figure 5.2. The Euclidean distance is first computed for only the eight primary elements (denoted $||e_p||$), and keypoints with $||e_p||$ greater than a threshold are rejected without comparing the rest of the vector.

The keypoints that remain from the threshold test are stored, along with the distance $||e_p||$, into a retention cache. After all possible matches have been explored, the retention cache is searched linearly. The full Euclidean distance calculations are completed using the rest of the vector and adding the total to the $||e_p||$ value. As in Lowe's algorithm, the distance ratio between the closest and second-closest keypoint vectors is used to declare a match or no match.

![Hierarchical Euclidean-Distance Matching Module](image)

**Figure 5.2**: Pseudo-code description for the hierarchical Euclidean-distance matching module.

The hierarchical approach effectively short-circuits Euclidean distance calculations, filtering out a great number of unlikely matches using minimal computation. The $||e_p||$ threshold is tuned to deliver approximately equivalent matching performance and a speedup factor of better than one order of magnitude over a linear search.

### 5.2 Handedness Splitting

In our analysis from Chapter 3.4, we introduced the handedness score ($h$), which, when applied to a large keypoint database such as $S_d$, leads to an almost-bimodal distribution – except for the large
density spike near \( h=0 \) – with local maxima near \( h=\pm 160 \) (Figure 3.6). We also showed that the anomalous, large concentration of keypoints near the \( h=0 \) mark consisted primarily of vectors whose inner primary elements represented a disproportionately high percentage of the total descriptor length (a ratio we termed the *inner primary ratio*, or IPR). The distribution of \( S_A \) keys, separated into “high” (above-threshold) IPR and “low” (below-threshold) IPR vectors is shown in Figure 3.7. Finally, we suggested that by removing the high-IPR vectors, we would be left with a purely bimodal distribution with a local minimum near \( h=0 \) – in other words, a nearly-ideal distribution for splitting the database in half at the \( h=0 \) mark. The database-splitting operation would effectively halve the search space for a given keypoint match, and theoretically double the search speed. There are concerns with this strategy, however. Firstly, we will need to show that the high-IPR vectors are less distinct than the rest, otherwise we stand to gain no benefit in matching performance from eliminating them. Secondly, we will need to ensure the boundary effects from splitting the database in half are insignificant enough to be practical: if too many keypoints have matches on the opposite side of the \( h=0 \) threshold, the splitting operation will result in poorer performance, as the correct matches will not be found. We expect that some loss of recall and precision will inevitably occur due to the boundary effects; however, we hope to compensate for this loss (and in fact demonstrate a net performance gain) by eliminating the high-IPR vectors, which represent approximately 12% of all SIFT keys.

5.2.1 IPR Thresholding

To better understand the relationship between the IPR value and matching performance, we again make use of the stored vectors from the comprehensive set of trials performed on the \( S_A \) image database using the random framework (these vectors were initially used to create a distribution for the magnitude of the error vector in Chapter 4.5). The complete set of vectors (about 8.5 million vectors), the correctly matched vectors (nearly 3.8 million), and the incorrectly matched vectors (about 1.4 million) are all sorted and binned with respect to their IPR value. The graphs of Figure 5.3 show the ratio between the number of correctly-matched keypoints with respect to the total number of keypoints (Figure 5.3 (a)) and the incorrectly-matched keypoints (Figure 5.3 (b)), respectively, for a range of IPR values. The ratios are quite high for the low-IPR vectors, but degrade rapidly when the IPR is greater than about 0.15 (corresponding to vectors whose inner primary elements form 15% of the total descriptor length).

The reason for the poor performance of the high-IPR vectors is straightforward. A high IPR is an
indication that the gradients that form the inner primary elements – in other words, the image-patch
gradients aligned to the dominant keypoint orientation – are disproportionately stronger than the rest of
the gradients in the image patch. The fact that they are much stronger increases the likelihood that they
were subject to the SIFT post-processing technique of thresholding all descriptor elements to 20% of
the total vector magnitude, as we hypothesized in Chapter 3.4. If four elements in the vector need to be
so limited, it is an indication that the rest of the vector contains very small values. This thresholding,
followed by the practice of normalizing the vector to unit length, thus serves to magnify otherwise
insignificant and unstable parts of the vector. It is unsurprising, then, that these particular vectors would
demonstrate poor matching performance.

The ratios of Figure 5.3 give a strong indication that there is a matching performance benefit to be
realized by rejecting keypoints with IPR above a threshold. We attempt, then, to find this threshold
experimentally, using the calibration framework and the small set of $S_t$ random transformation trials (Chapter 4.5).

![Figure 5.3: The number of correctly-matched SIFT keys with respect to the total number of keys (a) and the incorrectly-matched keys (b), for a range of IPR values, from the $S_t$ random transformation trials (Chapter 4.5).](image)

The ratios of Figure 5.3 give a strong indication that there is a matching performance benefit to be
realized by rejecting keypoints with IPR above a threshold. We attempt, then, to find this threshold
experimentally, using the calibration framework and the small set of $S_t$ trial images. We tested a range
of thresholds, from $IPR=0.125$ to $IPR=0.5$, rejecting the keypoints with IPR above the threshold in each
case; the rest of the keypoints were matched linearly. We computed the average F1 score for all the
calibration trials and plotted it against the results obtained with a linear search in Figure 5.4 (a). From
this graph, it is clear that we are able to realize an – albeit small – matching performance gain by
excluding keypoints with high IPR, with a maximum gain observed when the IPR threshold is set to
0.235 (as indicated by the vertical dotted line). The IPR=0.235 threshold seems to represent a tipping point: if the threshold is set lower than this value, the number of “good” (more robust) keypoints rejected surpasses the number of “bad” ones and matching performance suffers. Using this threshold, approximately 12% of the SIFT vectors are discarded before matching (as indicated by the vertical dotted line in Figure 5.4 (b)).

![Figure 5.4: Matching performance (a) and the percentage of keypoints dropped (b) as a result of filtering out keypoints that exceed the given IPR threshold. When the IPR threshold is set to 0.235, matching performance is maximized (this point is indicated by the vertical dotted line in both graphs).](image)

5.2.2 Handedness Error

After filtering out the SIFT keypoints with high IPR values, binning the remaining database keypoints based on their handedness scores yields a symmetrical, bimodal distribution as shown in Figure 3.7. The filtering step improves matching performance while making a database split at the $h=0$
mark more feasible by eliminating most of the keypoints near the dividing line. It is still important, however, to quantify the boundary effects: specifically, how close do the handedness scores need to be for two SIFT keypoints representing the same image patch to match? If matching keypoint pairs are regularly found on either side of the $h=0$ line, the database split will result in much poorer matching performance. We thus introduce the scalar handedness error ($e_h$), that simply represents the difference in the handedness scores from two correctly-matched SIFT vectors:

$$e_h = h_v - h_{v'}$$

(5.1)

where $v$ and $v'$ are SIFT keypoint vectors that represent the same image patch in two different images, and $h_v$ and $h_{v'}$ are their handedness scores, respectively.

We computed the handedness error for the large set of correctly-matched SIFT vector pairs from the $S_4$ random transformation trials, and the resulting distribution is shown in Figure 5.5. The mean of the distribution is zero, since $e_h$ is symmetrical. The dotted vertical line through the graph indicates the $3\sigma$ mark (73.02), within which approximately 99.5% of the keypoint pairs are found.

![Figure 5.5: The absolute value of the handedness error ($e_h$) from all correctly matched keypoint vector pairs in the $S_4$ transformation trials. Mean and mean plus three standard deviations (signifying the threshold containing 99.5% of the matched pairs) values are indicated.](image)

Since the handedness score distribution ranges from approximately -250 to 250, the $3\sigma$ mark indicated represents just under one-seventh the total range. Furthermore, due to the density valley from
about $h=-100$ to $h=100$, the vast majority of keypoints are too far from the $h=0$ cutoff to be adversely affected by the boundary. Table 5.1 lists the percentage of the $S_a$ vectors with handedness scores within $3\delta$ of the $h=0$ threshold (in other words, with $h$ between -73.02 and 73.02) before and after the IPR filtering stage. The benefit of filtering is evident, as approximately half the keypoints that are vulnerable to boundary effects are removed. Only 8.5% of the post-filter vectors in the $S_a$ database are within $3\delta$ of the handedness error from the dividing line. It is reasonable, then, to expect that the matching performance loss due to the database split will be minimal.

<table>
<thead>
<tr>
<th></th>
<th>Keypoints within $3\delta$ of $h=0$ mark (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before IPR filtering</td>
<td>17.1</td>
</tr>
<tr>
<td>After IPR filtering</td>
<td>8.5</td>
</tr>
</tbody>
</table>

Table 5.1: The percentage of the $S_a$ database keypoints with handedness score within $3\delta$ of the $h=0$ threshold, before and after the IPR filtering stage.

### 5.2.3 Comparison with Linear Search

Here, we attempt to quantify the effect of our handedness-splitting module with respect to matching performance and convergence speed. As stated in the algorithm description of Chapter 5.1.1, we conduct our transformation trials as follows. First, SIFT keypoints with IPR values greater than 0.235 are discarded, which has the double effect of both eliminating keys that are known to match poorly and creating a bimodal distribution with local maxima on either side (and equidistant from) the $h=0$ axis. Second, they are each separated into left-handed and right-handed bins based on the sign of the handedness score (with the $h=0$ keys arbitrarily grouped into the right-handed bin; grouping them into the left-handed bin yields nearly identical results). Finally, we search (linearly) for keypoint matches between like-handed bins. Because we are searching approximately half the database for keypoint matches, we expect a speedup factor of two over a full linear search.

We again used the $S_t$ images and the calibration framework to compare the handedness splitting technique against a linear search. The results of these trials are shown in Table 5.2. Our technique achieves both better recall and precision, while converging slightly better than twice as fast. It should be noted that because the IPR filtering step removes, on average, more than 12% of the keypoints from the feature set, we normalized the matching times to the average amount of time required to match two 1,000-keypoint datasets, counting only the keypoints that pass the filtering stage.
Table 5.2: SIFT matching trial results using $S_i$ and the calibration framework. The handedness splitting strategy, combined with high-IPR keypoint filtering, is compared against a linear search.

### 5.3 Hierarchical Euclidean-Distance Matching

Given the descriptor index analysis from Chapter 3.2, some parts of the SIFT descriptor vector vary much more – and are hence more important to a Euclidean distance-based matching algorithm – than others, it would make sense to take advantage of this to improve matching convergence times. In Chapter 3.3, we laid the framework for a hierarchical matching strategy by dividing the descriptor vector into primary (eight elements), secondary (12 elements), and tertiary (40 elements) groupings based on a ranking by standard deviation. The specific vector indices that correspond to these groupings are summarized in Table 3.4. We propose to match SIFT vectors in stages and, after each stage, retain only the vectors most likely to contain the closest match. We expect that, by eliminating a vast majority of the least likely matches after having only examined a small subset of the vector, we will be able to approach the recall and precision performance of a linear search while converging in a fraction of the time.

#### 5.3.1 Finding Primary, Secondary and Tertiary Thresholds

Assuming that the size of the keypoint retention cache is large enough (thousands) to accommodate an image of any size, we can eliminate it as a design parameter and use Euclidean distance thresholds to determine whether a keypoint is retained or dropped. In keeping with the notation used for our discussion of the error vector in Chapter 3, we introduce the primary ($e_p$), secondary ($e_s$) and tertiary ($e_t$) error vectors as follows:

$$
e_p = (v_i - v_i')$$

$$
e_s = (v_s - v_s')$$

4 The average matching time is the approximate time required to find correspondences in two 1,000-keypoint images, with filtered keypoints not included.
\[ e_i = (v_i - v'_i) \]  \hspace{1cm} (5.5)\]

where \( v_p, v_s, v_t, v'_p, \) and \( v'_t \) are the primary, secondary and tertiary subsets of SIFT vectors \( v \) and \( v' \), which represent the same image patch. In determining thresholds for the hierarchical matching stages, we will be concerned with the magnitude of vectors \( e_p, e_s \) and \( e_t \). We thus define the primary \( (d_p) \), secondary \( (d_s) \) and tertiary \( (d_t) \) distance as follows:

\[ d_p = \|e_p\| = \sqrt{\sum_{i=1}^{7} e_p(i)^2} \] \hspace{1cm} (5.6)

\[ d_s = \|e_s\| = \sqrt{\sum_{i=1}^{11} e_s(i)^2} \] \hspace{1cm} (5.7)

\[ d_t = \|e_t\| = \sqrt{\sum_{i=1}^{39} e_t(i)^2} \] \hspace{1cm} (5.8)

In thresholding \( d_p, d_s \) and \( d_t \), we aim to achieve the same (or as close as possible) matching performance of a linear search while minimizing the convergence time. We again use the \( S_i \) image dataset and the calibration framework to derive the thresholds experimentally. Figure 5.8 illustrates the calibration trials used to find the best limit for \( d_p \).

![Figure 5.6: Calibration trials for finding an appropriate primary threshold for hierarchical Euclidean-distance matching (HEDM). Matching performance (a) and speedup factor (b) are shown for a range of threshold values. The point at which the F1 score is equivalent to a linear search \((d_p=75\), as indicated by the vertical dotted line on both graphs\) is chosen as the best threshold.](image)

46
The process used to find an appropriate primary threshold is described as follows. SIFT features are extracted from the original and transformed images, and are matched against one another. For each keypoint pair, the primary distance is calculated. If \( d_p \) exceeds the chosen threshold, the search point is rejected as a possible match. If \( d_p \) is lower than the threshold, the search point, along with the \( d_p \) value, is saved into the retention cache. Once all of the possible matches have been examined, the retention cache is searched linearly to find the closest and next-closest matches using the full Euclidean distance calculation on the rest of the vector. As in the original algorithm proposed by Lowe, the distance ratio of 0.6 between the two closest keypoints to the target is used to either return a match or a null match. We experimented with a range of primary distance thresholds and achieved the matching performance results (represented by the F1 score) shown in Figure 5.6 (a). Obviously, increasing the threshold leads to a greater number of keypoints being retained in the cache and better matching performance, at the cost of longer computation times (Figure 5.6 (b)). Conversely, smaller thresholds show faster convergence but at a lower recall-precision performance. As with the error vector thresholding trials from Chapter 4.5, we chose our primary threshold as the point at which the F1 score using the hierarchical algorithm was equivalent to that achieved by means of a linear search. From Figure 5.6 (a), this crossing point occurs when \( d_p = 75 \), which results in a speedup factor of approximately 11 over a linear search.

We determined the secondary threshold in a similar fashion. We fixed the primary threshold at 75, and forced the SIFT keys with \( d_p \leq 75 \) to pass through a second filtering stage. The secondary distance, \( d_s \), is used to determine which keys are retained in the keypoint cache for full Euclidean distance calculation. Figure 5.7 (a) shows our experiments across a range of secondary threshold values. We again use the crossing point of the F1 score curves, when comparing our method against a linear search, to find the threshold that achieves equivalent matching performance. This point occurs when the secondary threshold is set at \( d_s = 95 \), and is indicated by the vertical dotted line in Figure 5.7 (a).

The tertiary threshold was likewise determined by fixing the primary and secondary thresholds at \( d_p = 75 \) and \( d_s = 95 \), respectively, and forcing the remaining keypoints through a third filtering stage using the tertiary descriptor grouping. The tertiary threshold that met the matching performance for a linear search was found at \( d_t = 168 \), as shown in Figure 5.7 (b).
5.3.2 Comparison with Linear Search

The average recall, precision and the matching time speedup factor for all hierarchy trials is compared with a linear search in Table 5.3. The F1 scores for all trials line up as a result of the calibration process. Recall and precision metrics are nearly identical, with the hierarchical approach trading off slightly better recall for a precision loss in all cases. The matching time benefit in treating the SIFT descriptor hierarchically is evident from the last column in the table, which shows that we can expect approximately an order of magnitude improvement regardless of the number of hierarchy levels chosen. This speedup is possible due to the perhaps surprisingly high percentage of keypoints that are discarded after computing the Euclidean distance for only a subset of the descriptor vector. In fact, 96.8% of the potential matches to any given keypoint are rejected after calculating the Euclidean distance for only the eight primary elements. Adding subsequent tiers, while eliminating incrementally more potential matches, do not benefit the matching time, as the gain not substantial enough to make up for the additional processing required to handle the extra levels. We thus choose to limit the hierarchical matching algorithm to one level, using the \( d_p = 75 \) threshold, which should deliver the equivalent matching performance of a linear search while converging approximately 11 times faster.

Figure 5.7: Calibration trials for finding appropriate secondary (a) and tertiary (b) thresholds for hierarchical Euclidean-distance matching (HEDM). The point at which the F1 score is equivalent to a linear search \( d_s = 95 \) in (a) and \( d_t = 168 \) in (b), as indicated by the vertical dotted lines on both graphs) is chosen as the best threshold.
### Hierarchy Levels

<table>
<thead>
<tr>
<th>Levels</th>
<th>Threshold(s)</th>
<th>Recall</th>
<th>Precision</th>
<th>F1 Score</th>
<th>Keypoints Dropped (%)</th>
<th>Speedup Factor over Linear Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>--</td>
<td>0.862</td>
<td>0.990</td>
<td>0.921</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$d_p = 75$</td>
<td>0.863</td>
<td>0.987</td>
<td>0.921</td>
<td>96.80</td>
<td>11.04</td>
</tr>
<tr>
<td>2</td>
<td>$d_p = 75$, $d_s = 95$</td>
<td>0.867</td>
<td>0.983</td>
<td>0.921</td>
<td>99.45</td>
<td>10.88</td>
</tr>
<tr>
<td>3</td>
<td>$d_p = 75$, $d_s = 95$, $d_t = 168$</td>
<td>0.864</td>
<td>0.988</td>
<td>0.921</td>
<td>99.85</td>
<td>9.28</td>
</tr>
</tbody>
</table>

Table 5.3: SIFT matching trial results using $S$, and a pre-defined set of image transforms. The hierarchical approach to Euclidean-distance matching is compared against a linear search.

### 5.4 Evaluation

Here, we present an evaluation of our complete matching algorithm, which combines the handedness splitting technique from Chapter 5.2 and hierarchical Euclidean-distance matching from Chapter 5.3. In addition, we place a maximum value on the magnitude of the error vector as determined by the thresholding trials of Chapter 4.5. We want to compare the full algorithm, which we call SIFT-HHM (for SIFT-Handed-Hierarchical Matching), against not only SIFT matching using a linear search, but also against two other popular local descriptors – PCA-SIFT and SURF – that claim to be both faster and more robust than SIFT.

For the evaluation, we used both our trial image dataset, $S$, and the PCA-SIFT training images\(^5\). We conducted transformation trials according to the random framework specified in Chapter 4.2. The PCA-SIFT matching program was thresholded (to a maximum Euclidean distance value of 3750) to deliver a similar recall as SIFT, based on a calibration process using a separate set of images and a few simple transformations. Like SIFT, SURF keys were matched using a nearest-neighbour distance ratio metric of 0.6. The results from the evaluation are shown in Table 5.4.

---

\(^5\) PCA-SIFT training images can be found at [http://www.cs.cmu.edu/~yke/pcasift/](http://www.cs.cmu.edu/~yke/pcasift/)
Table 5.4: SIFT-HHM compared with PCA-SIFT, SURF and a linear search, using the PCA-SIFT training images (a) and using our trial image dataset, S, (b).

We recorded recall, precision and the aggregate F1 score as matching performance metrics. Since the number of features extracted by each algorithm is not consistent (PCA-SIFT extracts the same number of features as SIFT, but SIFT-HHM matches 12% less than SIFT due to the filtering step and SURF finds about two-thirds the number of feature points as SIFT), the matching times are normalized to the amount of time it takes to find all of the correspondences between two 1,000-feature datasets. This number represents the average amount of SIFT keys extracted from a typical image, and the number of SURF features found in a very large, or detailed, image.

As the PCA-SIFT eigenspace is trained on the first set of images, it is perhaps not surprising that PCA-SIFT performs slightly better on it than the other local descriptors (Table 5.4(a)). The roughly threefold speedup achieved using PCA-SIFT over regular SIFT can be attributed to the fact that its descriptor vector (36 elements long) is just under one-third the length of the SIFT vector. SURF was the worst performer – judging by the F1 score – on both datasets. SURF matching is clearly faster than

---

6 The normalized matching time is the amount of time it takes to find all of the correspondences between two feature sets with 1000 features each.
either SIFT or PCA-SIFT matching, however, because it uses a 64-element vector and the database is split in two based on the sign of the Laplacian, which can be considered a 65th element.

Our algorithm showed a slight matching performance degradation over SIFT while reaching convergence about an average of 16 times faster, regardless of the image dataset used. SIFT-HHM is also comparable, performance-wise, to PCA-SIFT and SURF, at an approximate speedup factor of five times and four times, respectively.

5.5 Discussion

In this chapter, we constructed an efficient SIFT matching algorithm that exploits the descriptor properties described in Chapter 3. Our algorithm is tuned to deliver nearly identical performance as SIFT matching while converging in approximately one-sixteenth the time. When compared with two more recent local descriptors – PCA-SIFT and SURF – that use far fewer dimensions, our algorithm achieves comparable recall and precision performance and finds feature point correspondences five and four times faster, respectively. These results suggest that, for single-image matching applications such as real-time object recognition and tracking, SIFT, in combination with our Handed-Hierarchical Matching program, should be the local descriptor of choice.
A PCA-Based Binning Approach for Matching to Large Keypoint Databases

In Chapter 5, we presented a modular approach, which we call SIFT-HHM (for SIFT-Handed-Hierarchical Matching), for quickly finding SIFT keypoint correspondences in image pairs. This algorithm is well-suited to applications like object-tracking, where the features need to be computed and compared from consecutive frames in real time. Some applications, however, require quickly finding correspondences to a very large archive. In these applications, like content-based image retrieval, it is more practical to construct a searchable data structure from the archive. For fast SIFT matching to large databases, the most successful such techniques have found approximate nearest-neighbours from randomized kd-trees or k-means trees [3]. The latter has been shown to have improved query search speed by a factor of three orders of magnitude over a linear search, at a reduced (60%) precision rate verified by our evaluation in Chapter 6.3.

For these types of applications, we propose to divide a SIFT keypoint database into a series of bins, using the principal components of the SIFT descriptor vector as the binning criteria. In the discussion that follows, we use the value $N$ to describe the number of times the database is divided, using the first $N$ principal components. The optimal choice of $N$ is dependent on the size of the database, as we show in Chapter 6.2. The bin contents are stored in a hash table, and retrieved using a unique integer key. The database query process is described as follows. The principal components for the query vector are first computed, and a desired search range is provided by the user. All of the bins within the given range from the query vector are retrieved; their contents are searched linearly using the SIFT-HHM technique from Chapter 5. Our experiments (Chapter 6.3) show that our binning method, which we refer to as SIFT-BHHM (for Binned Handed-Hierarchical Matching), is approximately three times faster than the fastest tree-based structures for the same matching performance on databases of 100,000, 1 million and 2 million SIFT vectors. Furthermore, its simple construction allows for fast build times and it is easily scalable: additional keypoints can be inserted by simply applying a PCA projection to their descriptor vectors and appending them to the appropriate bins. If the database grows to the point where further sub-division is necessary, this is also a trivial operation. By comparison, tree structures require time-consuming tree-traversals and re-balancing when new vectors are added.
6.1 PCA Descriptor and PCA Error Vector

Our algorithm uses the PCA projection of SIFT vectors to associate SIFT keys with specific bins. The PCA projection, which we will call the principal component vector (PCV), creates an orthogonal and truncated vector space, using the principal components of the SIFT space as a basis. The PCA projection is inherently hierarchical, with the first element representing the axis along which the data shows the greatest variance, the second element the second-most variance, and so on (a more thorough review of PCA is provided in Chapter 2.2). As such, it is an ideal transformation for dimensionality reduction, as the highest-order elements typically represent data variance in the noise range and can be eliminated. PCA projection has been used most notably in [2] to reduce a 3042-element SIFT input vector to 36 dimensions. Our approach differs from PCA-SIFT in that we are using PCA primarily to reorganize a large dataset into bins, and to retrieve the closest bins to the query point through fast indexing. The retrieved keypoints are matched to the query point using their standard 128-element SIFT descriptor vectors, by the method described in Chapter 5. The strategy for sorting a database into hierarchical overlapping bins is covered in detail in Chapter 6.2. Here, we introduce the principal component vector and its associated error vector, $\mathbf{e}_{\text{PCV}}$. The standard deviation of $\mathbf{e}_{\text{PCV}}$ is used to define the width of the bins at each level of the hierarchy.

To construct the PCA projection matrix, the SIFT eigenspace was calculated offline from the large $S_A$ image archive (we used public-domain software for this task\footnote{http://astro.u-strasbg.fr/~fmurtagh/mda-sw/}). The $N$-dimensional PCV (where $N < 128$), then, is the vector created from the $N$ principal component projections of the SIFT vector space, and is defined as follows:

$$ \text{PCV} = \text{Norm}\{\text{KLT}\{\mathbf{v}\}\} = \{PCV_0, PCV_1, PCV_2, PCV_3, \ldots, PCV_N\} $$

(6.1)

where $\text{KLT}$ is the Karhunen–Loeve transform described in Chapter 2.2, and $\mathbf{v}$ is a SIFT descriptor vector. The $\text{Norm}$ function simply scales and shifts the PCA projection of $\mathbf{v}$ such that every element in $\text{PCV}$ has an approximately Gaussian distribution with a mean of 127.5 (half the byte range) and a standard deviation of 50, as shown in Figure 6.1. Furthermore, the PCV elements are thresholded at 0 and 255 to contain their values within a byte range in order to reduce overhead. This normalization step allows us to construct a general binning algorithm that expects the same distribution of keypoints at every level in the hierarchy. We refer to the individual principal components as $PCV_0$ (for the first
principal component), $PCV_1$ (for the second principal component), $PCV_2$, and so on.

Analagous to our discussion of the SIFT error vector in Chapter 4.4, our binning algorithm is an attempt to reduce the search space around a target vector as much as possible while still returning the closest nearest neighbour. We conceived of the SIFT search space as a 128-dimensional hypersphere, with a radius equal to the maximum magnitude of the error vector. Here we use the orthogonal PCA space to partition the data, grouping keypoints with close principal components into the same bins. The width of the bins (in terms of the range of PCV values held by a given bin) is determined experimentally, by looking at the difference in the PCV elements for correctly-matched keypoint pairs over a large set of transformation trials. This difference vector, which we call the PCV error vector ($e_{PCV}$), is defined as follows:

$$e_{PCV} = PCV - PCV' = \{ e_{PCV_0}, e_{PCV_1}, e_{PCV_2}, ... e_{PCV_N} \}$$  \hspace{1cm} (6.2)$$

where $PCV$ and $PCV'$ are principal component vectors corresponding to different views of the same image patch. Like the PCV, the PCV error vector's constituent scalars are denoted $e_{PCV_0}$, $e_{PCV_1}$, $e_{PCV_2}$ and so on.

In Chapter 4.5, we found the distribution for the magnitude of the SIFT error vector, $e$, by examining the millions of correctly-matched vector pairs resulting from the random transformation trials on the $S_i$ image database. Here, we use the same vector pairs to find distributions for the individual elements
in $\mathbf{e}_{PCV}$. The principal component vectors for all of the matched pairs were computed using the PCA projection matrix, and $\mathbf{e}_{PCV}$ computed for each pair. The distributions for the first four elements of $|\mathbf{e}_{PCV}|$ (we use the absolute value because $\mathbf{e}_{PCV}$ is symmetrical in the negative range) are shown in Figure 6.2. Evidently, there is very little difference between the distributions, and most data points are contained within a small value, when compared with the full byte range of the PCV elements. This is an indication that, for correctly-matched keypoint pairs, the first few principal components are very closely aligned. We might expect, however, that the higher-order principal components will not be as synchronized. This is because higher-order elements represent axes about which the data shows less variance; these elements, therefore, do not contribute as much to a match. The standard deviations ($\delta$) and $3\delta$ mark (representing the threshold within which 99.5% of the correctly-matched keypoint pairs are found) for the first 16 elements of $\mathbf{e}_{PCV}$ are shown in Table 6.1. There seems to be a gradual, if inconsistent, upward trend in the $\delta$ value with increasing index of $\mathbf{e}_{PCV}$. The first six elements have $3\delta$ values less than 30, which corresponds to less than one-eighth the PCV byte range.

In Chapter 6.2.5, we use the standard deviation of the PCV error vector as a basis for determining the width of the bins, and for determining the search range for database queries. It is useful, then, to refer to it by the following vector notation:

$$\delta_{PCV} = \{ \delta_{ePCV0}, \delta_{ePCV1}, \delta_{ePCV2}, \ldots, \delta_{ePCVN} \} \quad (6.3)$$

where $\delta_{ePCVi}$ represents the standard deviation of the $i^{th}$ element of the PCV error vector (basically the median difference between two SIFT vectors representing the same image patch).

Figure 6.2: Distribution of the first four elements of $|\mathbf{e}_{PCV}|$ ($\mathbf{e}_{PCV0}, \mathbf{e}_{PCV1}, \mathbf{e}_{PCV2}, \mathbf{e}_{PCV3}$), computed from the random $S_4$ transformation trials.
### 6.2 Binning Algorithm

In this section, we describe in detail our binning algorithm for finding fast matches in large SIFT keypoint databases. Chapter 6.2.1 covers the method by which keypoints are sorted and stored in bins, using their first N principal components. Chapter 6.2.2 discusses the database query process, whereby the bins closest to the query vector are retrieved and their contents searched using the SIFT-HHM technique from Chapter 5. In Chapter 6.2.3, we attempt to optimize N for databases of 100,000 and 1 million SIFT keys.

#### 6.2.1 Database Sorting

In order to sort SIFT keys into bins, we must first define boundaries with which to divide the database. Because the PCV elements all have approximately the same distribution (an approximated Gaussian distribution with a mean of 127.5 and standard deviation of 50, as shown in Figure 6.1), it is possible to create a generic method for assigning bin boundaries, regardless of the PCV index used. We use the $\delta_{PCV}$ vector (Table 6.1), which varies depending on the index, to define the bin widths. The index value is first rounded up to the nearest integer, and bin widths are nominally assigned to be four times this value, as follows:

$$ w_i = 4 \times \text{roundup}(\delta_{PCV}) \tag{6.4} $$

<table>
<thead>
<tr>
<th>$\delta_{PCV}$ Index</th>
<th>$\delta$</th>
<th>$3\delta$</th>
<th>$\delta_{PCV}$ Index</th>
<th>$\delta$</th>
<th>$3\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7.70</td>
<td>23.10</td>
<td>8</td>
<td>11.09</td>
<td>33.27</td>
</tr>
<tr>
<td>1</td>
<td>6.72</td>
<td>20.16</td>
<td>9</td>
<td>12.65</td>
<td>37.95</td>
</tr>
<tr>
<td>2</td>
<td>7.60</td>
<td>22.80</td>
<td>10</td>
<td>12.71</td>
<td>38.13</td>
</tr>
<tr>
<td>3</td>
<td>9.03</td>
<td>27.09</td>
<td>11</td>
<td>11.84</td>
<td>35.52</td>
</tr>
<tr>
<td>4</td>
<td>7.95</td>
<td>23.85</td>
<td>12</td>
<td>13.19</td>
<td>39.57</td>
</tr>
<tr>
<td>5</td>
<td>9.72</td>
<td>29.16</td>
<td>13</td>
<td>11.28</td>
<td>33.84</td>
</tr>
<tr>
<td>6</td>
<td>12.10</td>
<td>36.30</td>
<td>14</td>
<td>12.64</td>
<td>37.92</td>
</tr>
<tr>
<td>7</td>
<td>10.24</td>
<td>30.72</td>
<td>15</td>
<td>12.16</td>
<td>36.48</td>
</tr>
</tbody>
</table>

Table 6.1: Standard deviations for first 16 elements of $\delta_{PCV}$. 

56
where $w_i$ represents the width of the bins for PCV element $i$ ($i < N$). The four-times multiplier represents a compromise between granularity and a desire to minimize the number of hash table retrievals, which can negatively impact the search times. Figure 6.3 illustrates the delineation of the bins for PCV element $i$. Since the PCV elements are contained within a byte range, the values 0 (minimum), 255 (maximum) and 127 (mean) represent natural boundaries. The $w_i$ indices create the remaining thresholds; however, the size of the $w_i$ values dictates the number of bins used in the hierarchy level, according to the ranges specified in Table 6.2. This was a design decision to ensure that no maximum bin boundary could be less than the mean minus $2\delta$ (27.5) of the PCV element; likewise, no minimum boundary could be more than the mean plus $2\delta$ (227.5) of the PCV element (these points are shown by the dotted vertical lines in Figure 6.3). Allowing boundaries outside this range would result in bins with very few keypoint entries. Finally, Table 6.3 summarizes the bin structure used on our datasets. The first 16 PCV indices, with their corresponding $d_{PCV}$ value (from Table 6.1) are listed. The remaining columns show the calculation of the bin width value, $w_i$, and number of bins at each index $i$. All of the indices in the table separate the database into either 6 or 4 bins, according to the formula laid out in Table 6.2. In Chapter 6.2.3, we show that a choice of $N=6$ will maximize performance on a database with 100,000 SIFT keys. According to Table 6.3, this means that we will utilise PCV indices 0-5 to divide the database. Since these indices correspond to a division into 6 bins each, the total database division is $6^6 = 46,656$, which allows for, on average, just over two SIFT vectors per bin. In practice, because the PCV elements have Gaussian (and not uniform) distributions, some bins will have far more entries than others, and a large percentage will be empty. For these reasons, the bin structure is implemented as a sparse hash table within which only bins containing vectors are stored.

![Figure 6.3](image)

Figure 6.3: Bin boundaries for PCV element $i$ are determined by the chosen width element $w_i$. No bin maximum is allowed to be less than 27.5 and no bin minimum is allowed to be greater than 227.5 (these values correspond to the $2\delta$ range of all PCV elements, and are indicated by the vertical dotted lines).
Bin Width (for the $i^{th}$ index) | Number of bins for PCV index $i$
---|---
$w_i \leq 50$ | 6
$w_i > 50 \land w_i \leq 100$ | 4
$w_i > 100$ | 2

Table 6.2: The number of bins for each index is determined by the size of the width elements. The thresholds 50 and 100 represent one and two standard deviations of the PCV element, respectively.

<table>
<thead>
<tr>
<th>PCV index $i$</th>
<th>$\delta_{PCV}$</th>
<th>$w_i$</th>
<th>#bins at index $i$</th>
<th>PCV index $i$</th>
<th>$\delta_{PCV}$</th>
<th>$w_i$</th>
<th>#bins at index $i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7.70</td>
<td>4*8= 32</td>
<td>6</td>
<td>8</td>
<td>11.09</td>
<td>4*12= 48</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>6.72</td>
<td>4*7= 28</td>
<td>6</td>
<td>9</td>
<td>12.65</td>
<td>4*13= 52</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>7.60</td>
<td>4*8= 32</td>
<td>6</td>
<td>10</td>
<td>12.71</td>
<td>4*13= 52</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9.03</td>
<td>4*10= 40</td>
<td>6</td>
<td>11</td>
<td>11.84</td>
<td>4*12= 48</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>7.95</td>
<td>4*8= 32</td>
<td>6</td>
<td>12</td>
<td>13.19</td>
<td>4*14= 56</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>9.72</td>
<td>4*10= 40</td>
<td>6</td>
<td>13</td>
<td>11.28</td>
<td>4*12= 48</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>12.10</td>
<td>4*13= 52</td>
<td>4</td>
<td>14</td>
<td>12.64</td>
<td>4*13= 52</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>10.24</td>
<td>4*11= 44</td>
<td>6</td>
<td>15</td>
<td>12.16</td>
<td>4*13= 52</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 6.3: The $\delta_{PCV}$ values, bin widths and the number of bins at each PCV index $i$, computed from random transformation trials on our $S_d$ image archive.

A SIFT vector is inserted into the binning structure as follows. First, the first $N$ principal components are computed. Each PCV index value is then compared against the bin boundaries for that particular index, and an array keeps track of the bin number within which the keypoint falls for each index. Figure 6.4 demonstrates this process using a simplified bin structure with $N=3$. SIFT vector A's first principal component ($PCV_0$) falls within the range for bin number 2, its second principal component within the $PCV_1$ bin 3 range, and its third principal component within the $PCV_2$ bin 1 range. The bin insertion array holds the bin numbers for each index.
The last step is to take the bin insertion array values and construct a unique integer key with which to identify the bin, so that it can be inserted into a hash table structure. The key must be unique so as to avoid hash collisions. We use the following formula to generate this unique bin key:

$$unique\ bin\ key = \sum_{i=0}^{N-1} b_i \prod_{j=i+1}^{N-1} n_j$$

(6.5)

where $b_i$ represents the bin number at the $i^{th}$ index (from the bin insertion array), and $n_j$ represents the number of bins at index $j$, where $j=i+1$. From the example illustrated in Figure 6.4, keypoint A's unique integer key is $2*(6*6)+3*(6)+1 = 91$. Note that, in this example, because all three indices use 6 bins, the integer key is effectively a three-digit base-6 value. Keypoint A is inserted into the hash table structure, using integer key 91 to reference a linked list element containing the keypoint. We have used the hash_map class from the C++ Standard Template Library to implement the hash table\textsuperscript{8}. Upon insertion, the algorithm first checks to see if there already are entries in the desired bin, and if so,

---

\textsuperscript{8} STL documentation and source can be found at \url{http://www.sgi.com/tech/stl/}. Other hash table implementations, such as Google's sparsehash and densehash, may be used in place of hash_map, to optimize for memory usage and search speed, respectively.

---

59
prepends the current SIFT key to the list; otherwise, it creates the list with the current key as the only entry.

Of final note, in our implementation, we combine the IPR filtering and handedness splitting stages from SIFT-HHM into the database sort: the former by excluding the high-IPR keys from the database, and the latter by multiplying the unique bin integer keys by -1 for all left-handed SIFT vectors (so as to distinguish them from the right-handed vectors with the same integer key).

### 6.2.2 Database Querying

The database query process is implemented as a range search: bins within the user-specified range from the query vector are retrieved from the hash table, and their contents are searched using the method from Chapter 5. The *search width* input parameter is expressed as a multiple of the $\Delta_{PCV}$ vector, and from it the boundaries of the search are defined at every PCV index. Once the PCV is computed for the query vector, we use the search width to retrieve all of the bins that fall within the given range. The bins themselves are identified by computing their unique bin keys (according to the formula of Equation 6.5), and a cache holds a pointer to each bin retrieved (if no hash table entries exist for a given bin key, no pointer is stored). Finally, each bin is iteratively searched (in the order of retrieval) using SIFT-HHM. Figure 6.5 provides an illustrated example, again using a simplified bin structure with $N=3$. A search range $W$ is given along with SIFT query vector $Q$, and the bins that fall within the search window are indicated in grey. A set of *bin retrieval arrays* keeps track of the bin combinations as the PCV index is incremented. The contents of the bin retrieval arrays are finally used to compute the unique bin integer keys. In this example, four bins are retrieved from the query.
In this section we examine the performance of SIFT-BHHM relative to a linear search, for a range of $N$ values and on databases of 100,000, 1 million and 2 million SIFT vectors. The specific performance metrics we wish to track are precision and recall (represented by the aggregate F1 score) and search speed (represented as a speedup factor over a linear search). We base our choice of $N$ as that which delivers the highest search time improvement for a given F1 score level. For these trials, we used the calibration framework on $S_t$, our trial image set. All of the keypoints from the $S_t$ images were appended to a single keypoint file (this totaled approximately 12,000 SIFT keys). To bring our databases up to 100,000, 1 million and 2 million vectors, we randomly selected images from the $S_A$ and $S_B$ archives and appended their keypoints to the file. For the trials, we performed transformations on the $S_t$ images, and used the SIFT vectors extracted from the transformed images as query vectors for the database, effectively modeling an image-search application. Since the original $S_t$ images are part of the database, the number of potential correct matches is known.

The results of our trials on all databases are shown in Figure 6.6. The first two graphs (Figure 6.6 (a) and (b)) capture the matching versus speed performance and the number of created bins, respectively, for varying values of $N$ on the 100,000-keypoint database. Likewise, Figures 6.6 (c) and (d) capture the
same metrics on the 1-million database trials, and Figures 6.6 (e) and (f) refer to the 2-million database trials. We will first discuss the matching versus speed performance graphs (Figures 6.6 (a), (c) and (e)) for all three databases. To create the curves, we experimented with a variety of $N$ values and varied the query search width ($W$) variable from 1.0 to 4.0. On all three of these graphs, the vertical dotted line indicates the F1 score achieved by means of a linear search. In examining Figure 6.6 (a), it is apparent that, to achieve a matching performance level equivalent to a linear search, a choice of $N=5$ or $N=6$ delivers the fastest search speeds (a speedup factor of nearly three orders of magnitude). If it is not necessary to achieve equivalent recall-precision to a linear search (F1 scores from approximately 0.65 to 0.8), it is clear that $N=6$ is the best choice. At even lower matching performance levels (F1 score under 0.6), $N=7$ appears to slightly outperform. On the 1-million feature set (Figure 6.6 (c)), it appears as though $N=7$ and $N=8$ provide the best search speeds for a given matching performance level, with $N=8$ performing slightly better for most recall-precision levels. Finally, matching trials on the 2-million feature set (Figure 6.6 (e)) also show the best overall performance with $N=8$.

The bin insertion graphs (Figures 6.6 (b), (d) and (f)) all show a similar S-curve pattern. There appears to be an exponential increase in the number of bins for lower values of $N$; this increase eventually becomes linear before leveling off. It appears from these curves that recursively dividing the database into progressively smaller bins reaches a saturation point, where, presumably, the bins are small enough and cannot be divided further. As we might expect, the saturation point, or leveling-off phase, occurs earlier on the smaller databases (approximately at $N=9$ for the 100,000-keypoint trials) than on the larger databases (about $N=11$ on the 2-million keypoint dataset). Interestingly, the midpoint of the S-curves seems to approximately coincide with the optimal matching and speed performance on all three databases ($N=6$ or $N=7$ on the 100,000-keypoint database, and $N=8$ on the 1-million and 2-million keypoint databases). Intuitively, this is the point at which the database has been divided enough times, and, consequently, the bins are small enough to significantly reduce the search space. Beyond this point, the additional (and smaller) bins do not justify the increased time cost involved in computing exponentially more unique bin integer keys for retrieval.
Figure 6.6: Search speed versus matching performance (a, c, e) and bin insertion (b, d, f) graphs for calibration trials using databases of 100,000 (a, b), 1 million (c, d) and 2 million (e, f) SIFT keys. On graphs (a), (c) and (e), the F1 score achieved by means of a linear search is indicated with a vertical dotted line.
We note finally that Figures 6.6 (a), (c) and (e) show that SIFT-BHHM is able to exceed the matching performance of a linear search for all values of $N$, using high search width values of either 3.0 or 4.0. While it may seem unlikely that a search algorithm that examines only a small fraction of a database might achieve better recall and precision than one that examines every potential match, we hypothesize that the reasons for this improvement are two-fold. First, the IPR filtering stage in the HHM algorithm (Chapter 5), which is used to find the correct match once a bin has been retrieved, may have a greater effect on larger databases. We showed in Chapter 5 that the high-IPR SIFT keypoints are less distinct and therefore match more poorly than the low-IPR keys on single-image trials. Placed within a large database, the high-IPR keys become even less distinct, given the number of additional keypoints that will have very similar descriptor vectors. A second factor in the improved matching performance may be that the nearest-neighbour distance ratio used to declare a match (the closest vector to the query point must be less than 60% of the distance to the second-closest vector) is not suitable for large databases, where many more vectors could come very close to the query point. Our technique of isolating a small subset of potential matches could introduce an incidental benefit of filtering out keys that might have been the second-closest vectors to the query points, thus allowing the closest match to pass the distance ratio test.

6.3 Evaluation

In this section, we compare SIFT-BHHM with the fastest available approximate nearest-neighbour search software, FLANN (Fast Library for Approximate Nearest Neighbours) [3], which reportedly can achieve up to 1,000 times search speed improvement on a single nearest-neighbour search, while returning 95% of the closest matches to a query vector (this result is achieved on a dataset with 31 million SIFT features). In Chapter 6.3.1, we describe FLANN in further detail, and Chapter 6.3.2 discusses our experimental results using the random framework on databases of 100,000 (which we call SIFT100K), 1 million (SIFT1M) and 2 million (SIFT2M) SIFT keys.

6.3.1 FLANN

FLANN is a configurable nearest-neighbour search library that is intended for use with a variety of high-dimensional vector spaces. Given a desired search precision (a positive floating-point number less than 1), and other parameters to optimize for memory use or build time, the algorithm examines the
dataset and chooses the best search algorithm to apply. According to [3], the best results for SIFT vectors use either randomized kd-trees or k-means trees (these algorithms are described in Chapter 2).

To find the optimal FLANN search algorithms for our datasets, we initialized the program using the default parameters (so as not to state a preference for either build time or memory use) and varied the target precision value from 0.6 to 0.9. The algorithms (and associated parameters) chosen by FLANN to achieve the fastest-possible query search results after examining all three (SIFT100K, SIFT1M and SIFT2M) databases are summarized in Table 6.4. For every configuration, the k-means tree was chosen as the most appropriate algorithm for our datasets. Higher search precision is achieved by increasing either the branching factor, number of search iterations, the number of leaf nodes examined per iteration, or some combination thereof. These settings are saved and used in the next section to configure FLANN for the comparison trials against SIFT-BHHM.

<table>
<thead>
<tr>
<th>Target Precision</th>
<th>Algorithm chosen by FLANN (100K SIFT database)</th>
<th>Algorithm chosen by FLANN (1M SIFT database)</th>
<th>Algorithm chosen by FLANN (2M SIFT database)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>k-means tree, branching factor = 32, maximum 5 search iterations, max. 1 leaf node searched / iteration</td>
<td>k-means tree, branching factor = 16, maximum 1 search iteration, max. 1 leaf node searched / iteration</td>
<td>k-means tree, branching factor = 32, maximum 1 search iteration, max. 1 leaf node searched / iteration</td>
</tr>
<tr>
<td>0.7</td>
<td>k-means tree, branching factor = 32, maximum 5 search iterations, max. 16 leaf nodes searched / iteration</td>
<td>k-means tree, branching factor = 32, maximum 1 search iteration, max. 1 leaf node searched / iteration</td>
<td>k-means tree, branching factor = 32, maximum 5 search iterations, max. 16 leaf nodes searched / iteration</td>
</tr>
<tr>
<td>0.8</td>
<td>k-means tree, branching factor = 32, maximum 10 search iterations, max. 57 leaf nodes searched / iteration</td>
<td>k-means tree, branching factor = 64, maximum 1 search iteration, max. 1 leaf node searched / iteration</td>
<td>k-means tree, branching factor = 32, maximum 10 search iterations, max. 32 leaf nodes searched / iteration</td>
</tr>
<tr>
<td>0.9</td>
<td>k-means tree, branching factor = 32, maximum 10 search iterations, max. 193 leaf nodes searched / iteration</td>
<td>k-means tree, branching factor = 16, maximum 5 search iterations, max. 230 leaf nodes searched / iteration</td>
<td>k-means tree, branching factor = 16, maximum 5 search iterations, max. 240 leaf nodes searched / iteration</td>
</tr>
</tbody>
</table>

Table 6.4: Algorithms (and associated parameters) chosen by FLANN given SIFT databases of 100,000, 1 million and 2 million keypoints and a range of target precision values (0.6 to 0.9).
6.3.2 SIFT-BHHM versus FLANN

In this section we compare our binning algorithm with FLANN using the random framework. The publicly-available images used to train PCA-SIFT\(^9\) are used here as our query images. We formed our databases by first extracting the SIFT keys from the query images, then combining them with a random sampling of SIFT vectors from the \(S_A\) and \(S_B\) image archives. For SIFT100K, we have chosen to run the binning algorithm with \(N=6\) (although the case could be made for \(N=7\) or \(N=5\) based on the results from Figure 6.6 (a), depending on the desired level of matching performance, we chose \(N=6\) as it was the best configuration for the widest range of F1 scores). Similarly, we chose \(N=8\) for both SIFT1M and SIFT2M. On all three databases, we varied the search width variable from 0.5 to 2.5 to get the speedup-versus-F1 score curves. FLANN was configured using the optimal algorithm parameters summarized in Table 6.4. We re-compiled the FLANN source\(^10\) (rather than simply making use of the existing binaries) to ensure that the compilation settings were consistent for SIFT-BHHM, the linear search and the FLANN matching program. For every query vector, FLANN returns the two nearest neighbours, and the distance ratio test (from Chapter 2.1.5) is applied to declare a match.

Figure 6.7 compares SIFT-BHHM and FLANN on search speedup versus matching performance graphs (shown on a logarithmic scale). Our binning algorithm clearly outperform FLANN on all three databases. There are some areas where the two algorithms are closer or further apart, but our algorithm typically achieves a three-fold speedup factor over FLANN for the same level of matching performance. The F1 score achieved by means of a linear search is again indicated by the vertical dotted line on all graphs. Our algorithm is able to deliver a nearly 1,000-fold speed improvement over a linear search – while maintaining an equivalent F1 score – on the SIFT100K database (Figure 6.7(a)), and about 10,000-fold speed improvement at a similar recall-precision level on both the SIFT1M and SIFT2M databases (Figures 6.7 (b-c)). In Table 6.5, we have used linear interpolation on the graphs of Figure 6.7 to estimate the speedup factor achieved at two matching performance points (60% and 90% of the linear search F1 score) on all three databases. The approximate factor-of-three improvement over FLANN is clearly evident from this table. Table 6.6 shows the actual search times for a 1,000-feature query (the number of SIFT features in a typical image) using FLANN, SIFT-BHHM and a linear search on all three databases.

---

9 http://www.cs.cmu.edu/~yke/pcasift/
10 FLANN source is available at http://people.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN.
The database build times indicated in Table 6.7 show SIFT-BHHM to be markedly faster on all data sets, which is perhaps to be expected, given the relative simplicity of the binning algorithm compared to a k-means tree.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>SIFT100K</th>
<th>SIFT1M</th>
<th>SIFT2M</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLANN speedup factor</td>
<td>1744</td>
<td>602</td>
<td>12569</td>
</tr>
<tr>
<td>SIFT-BHHM speedup factor</td>
<td>4703</td>
<td>1954</td>
<td>39483</td>
</tr>
</tbody>
</table>

Table 6.5: Interpolated speedup factors for FLANN and SIFT-BHHM on all three datasets, using 60% and 90% of the F1 score achieved by means of a linear search as performance points.

Figure 6.7: SIFT-BHHM and FLANN algorithms are compared, on a logarithmic scale, in terms of matching performance (F1 score) and search speed, using the SIFT100K (a), SIFT1M (b) and SIFT2M (c) databases. The search width (W) value and the target precision (pr) are indicated for the SIFT-BHHM and FLANN trials, respectively. The linear-search F1 score is marked by the vertical dotted line on all three graphs.
Table 6.6: Average query times for a 1000-feature search (the number of SIFT features in a typical image). For FLANN and SIFT-BHHM, the matching performance for the times given is 90% of that achieved by means of a linear search.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>SIFT100K</th>
<th>SIFT1M</th>
<th>SIFT2M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Search</td>
<td>123.01</td>
<td>1233.71</td>
<td>2549.95</td>
</tr>
<tr>
<td>FLANN</td>
<td>0.21</td>
<td>0.27</td>
<td>0.42</td>
</tr>
<tr>
<td>SIFT-BHHM</td>
<td>0.06</td>
<td>0.06</td>
<td>0.13</td>
</tr>
</tbody>
</table>

Table 6.7: Average build times for FLANN and SIFT-BHHM for the SIFT100K, SIFT1M and SIFT2M databases.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Average SIFT100K Build Time (s)</th>
<th>Average SIFT1M Build Time (s)</th>
<th>Average SIFT2M Build Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLANN</td>
<td>55.38</td>
<td>883.02</td>
<td>1116.44</td>
</tr>
<tr>
<td>SIFT-BHHM</td>
<td>2.95</td>
<td>70.65</td>
<td>90.2</td>
</tr>
</tbody>
</table>

6.4 Discussion

In this chapter, we built upon the SIFT-HHM technique introduced in Chapter 5 and constructed a binning algorithm that addresses the content-based image retrieval problem. Our technique decomposes a SIFT keypoint database into bins using the principal components of the SIFT descriptor vector as the binning criteria. The bin widths are designed to reflect the typical displacement ($\delta_{ePCV}$) along the principal component axes by two SIFT vectors representing the same image patch. For querying a database, the user specifies a search width variable that is a multiple of this typical displacement value. All of the bins contained within the given range are retrieved by the query process and, finally, searched linearly using SIFT-HHM. Our experiments from Chapter 6.3 show our modular binning algorithm to outperform FLANN search speeds – the current state-of-the-art for SIFT nearest-neighbour search – by an approximate factor of three for the same matching performance. Furthermore, due to its simplicity, our database sort process takes less than one-tenth the time to build a k-means tree, the optimal algorithm chosen by FLANN for SIFT matching.

The faster search speeds achieved by our binning algorithm may be attributed to its effectiveness at isolating a very small subset of potential keypoint matches from a large keypoint database: our analysis of the PCV error vector from Chapter 6.1 shows that two vectors representing the same image patch
tend to have very close alignment in their principal components, regardless of the type and extent of the distortion applied (too much distortion will render the vector unmatchable using any algorithm). Furthermore, we hypothesize that SIFT-HHM introduces a moderating effect on the search speeds. If the number of keypoints retrieved from the query process is low, SIFT-HHM will have little effect as the search space has already been significantly reduced. If the number of retrieved keypoints is high, however, SIFT-HHM will find the closest match in approximately $1/15^{th}$ the time it would take to search the retrieved bins linearly, as we showed in Chapter 5. Thus, our search algorithm does not suffer abnormally long search times when the query vector's principal components occupy a dense space within the database.

Additionally, we note our algorithm's ease of storage and scalability as advantages. Tree structures, such as the k-means tree, must retain relationships between nodes, and adding new elements sometimes involves costly tree re-balancing. Our database bins can be stored in any fashion, so long as the vectors that belong to the same bin are kept together. Introducing new vectors is simply a matter of finding the correct bin and appending to a list. If the database grows to the point where subdivision is necessary, it is simply a matter of dividing the existing bins along the next available principal component, a relatively trivial process. We suggest, finally, that our algorithm is well-suited to external memory-based distributed systems, where, for example, multiple bins could be searched in parallel from disparate repositories. Such an arrangement is likely to produce even better results than those we have demonstrated in this chapter, and on keypoint databases that are far larger.
Chapter 7

Conclusions and Future Work

This thesis has targeted one of the main limitations of the popular SIFT algorithm: its feature vector matching speeds. The methods presented here are based solely on an evaluation of the SIFT descriptor vector and its construction, and demonstrate discernible improvements over the current state of the art.

In Chapter 3, we showed that the vector indices aligned with the dominant keypoint orientation vary considerably more than the rest of the vector, and are thus more important from the perspective of a Euclidean distance-based matching algorithm. Furthermore, we demonstrated that a subset of these more important indices can be used to both identify less-distinct features for exclusion and to split a database in half using a metric we call the handedness score. In Chapter 5, we made use of these characteristics to construct our SIFT-HHM algorithm, which reduces the average search for a feature match in a typical image to approximately $1/15$th of the time it takes to perform an exhaustive linear comparison. This result also represents a matching time improvement over SURF and PCA-SIFT – two algorithms that use far fewer dimensions than SIFT – of five times and four times, respectively. This matching algorithm creates opportunities for applications needing very fast, real-time feature matching to make use of the well-documented robustness of SIFT.

In Chapter 6, we added a PCA-based sorting module to SIFT-HHM as a solution for content-based image retrieval applications. Our algorithm sorts a SIFT keypoint archive into bins based on the first few principal components of the descriptor vector. Bins are stored in a hash table, and retrieved based on proximity to a query vector. At the lowest level, the retrieved feature vectors are evaluated as potential matches using SIFT-HHM. Our experiments show this modular technique to outperform k-means tree search speeds (using FLANN [3], a nearest-neighbour search library that implements the current state-of-the-art in SIFT matching) by an approximate factor of three, for the same level of matching performance. Also worth noting is the fact that our data structure creation times are less than $1/10$th the (k-means) tree build times for databases with 100,000, 1 million and 2 million SIFT keys. Because of the discrete nature of the bins, our solution simplifies storage, as relationships between nodes do not need to be maintained. For the same reasons, our solution is one that lends itself well to distributed processing, where, for example, multiple bins could be searched in parallel rather than
consecutively (as they are in our implementation). Such an implementation would likely show even
greater search speed improvement than the results achieved in Chapter 6. Finally, we note that our
solution is easily scaled if an image archive grows to the point where the average bin size is too high
and search speeds are negatively impacted. In Chapter 6, we showed that the number of principal
components used to divide the database was dependent on the size of the database itself (we found that
\(N=6\) was optimal on the 100,000-keypoint database, and \(N=8\) a better choice on the 1 million and 2
million keypoint databases). Once a database reaches a certain threshold size, the bins should be
subdivided based on the next available principal component, which is a fairly trivial process. In
comparison, the kd-tree and k-means tree structures occasionally need expensive re-balancing when the
insertion of a new node disrupts the tree symmetry.

There are a number of opportunities to improve upon or extend the work done in this thesis. For one,
it remains to be seen whether an analysis like the one we have provided in Chapter 3 would be useful
and revealing for other feature descriptors (like SURF, for example). If it is possible to improve upon
linear search speeds by an order of magnitude (as we have done with SIFT-HHM), then the binning
algorithm from Chapter 6 is also likely to be more effective than a kd-tree or a k-means tree structure
for large image archives.

We mentioned in Chapter 5 that the IPR-filtering stage of SIFT-HHM could be embedded into the
SIFT algorithm itself, whether as a post-processing technique or – if these “less-robust” vectors could
be detected at earlier stages – as a means of short-circuiting the description phase. As we showed from
the thresholding trials of Chapter 5.2.1, such an approach would both strengthen the overall matching
results and significantly reduce the number of SIFT keys extracted from a given image, which impacts
storage and matching times.

The binning algorithm from Chapter 6 could be improved upon in a number of ways. We have
already mentioned using distributed processing to search retrieved bins in parallel, rather than in series
(it should be noted that to make use of distributed systems for kd-tree or k-means tree-based searches is
a more complex problem). Also, the choice of hash table implementation is certain to affect search
speeds. While we have used the `hash_map` class from the Standard Template Library, certain publicly-
available algorithms (including Google's `sparsehash` and `densehash`) will optimize for memory or
retrieval time. Finally, it is worth considering whether principal components are the best values to use
as binning criteria. As we stated in Chapter 2.2, PCA suffers from a few notable shortcomings; namely,
the assumption of linearity (that vectors can be properly expressed as linear combinations of a different
basis) and that vector elements have Gaussian distribution. It is worth investigating whether nonlinear methods, such as Kernel PCA [28] allow for greater database division and fewer bin retrievals, resulting in faster search speeds.

In summary, the main contributions of this thesis are:

- A modular technique, called SIFT-HHM, for finding SIFT correspondences in consecutive image frames in less than 7% of the time of a linear search.

- A method for filtering out less-distinct features (and thus, reducing the likelihood of mismatches) by examining the contributions of four of the 128 SIFT descriptor elements (IPR filtering).

- A PCA-based binning algorithm that also uses SIFT-HHM at the lowest level and finds SIFT correspondences in large databases, at about a three-fold speed improvement over the existing state of the art for SIFT matching, without compromising recall and precision. The binning approach is generic, and is likely to produce similar results using other feature descriptors (i.e. SURF), provided that they exhibit similar statistical properties to SIFT.
Bibliography


