João Gabriel Santiago Mauricio de Abreu

PARAGRAPH: A LIBRARY FOR EFFICIENT DOCUMENT IMAGE RETRIEVAL

B.Sc. Dissertation

Federal University of Pernambuco
www.cin.ufpe.br

RECIFE
2017

Federal University of Pernambuco
Center for Informatics
Graduate in Computer Science

João Gabriel Santiago Mauricio de Abreu

PARAGRAPH: A LIBRARY FOR EFFICIENT DOCUMENT IMAGE RETRIEVAL

A B.Sc. Dissertation presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor in Computer Science.

Advisor: *Veronica Teichrieb*
Co-Advisor: *João Marcelo Teixeira*

RECIFE
2017

Acknowledgements

First and foremost, I would like to thank God, the Almighty, for giving me strength and knowledge during this journey; without His blessings, nothing would be possible.

I would also like to thank my family for their support and guidance throughout my life, and especially for teaching me the importance of learning and diligence. Similarly, I would like to thank my girlfriend, Rayana, for her encouragement, understanding and companionship during this journey.

Furthermore, I would like express my gratitude to my advisors, Veronica Teichrieb and João Marcelo Teixeira for their support and helpful guidance. Their knowledge and advice have helped me to keep on track and work at a smooth pace. In addition, I would like to thank all the professors in CIn for their advice and teachings; their efforts have allowed me to learn at a place of excellence. Likewise, I am indebted to my colleagues both at school and at Voxar Labs, who have helped make my learning an enjoyable and stimulating experience. Finally, I would like to specially thank Lucas Maggi for his help during the experiments performed in this work.

The best thing about a boolean is even if you are wrong, you are only off by a bit.

—UNKNOWN

Abstract

The task known as Camera-Based Document Image Retrieval has become increasingly relevant in the last decade due to the spread of mobile devices, which usually include digital cameras. A solution to this problem, particularly one suitable for use in the limited hardware of those platforms, is of great interest to a number of applications. For that reason, we have explored existent solutions to the problem, and implemented a modified version of the Locally Likely Arrangement Hashing (LLAH) method, in the form of a C++ library called pARagraph. Bearing in mind its usefulness for mobile devices, we adopted several optimizations in the implementation, focused in reducing execution time and, primarily, memory consumption. The pARagraph library was evaluated in regard to its accuracy, execution time and memory usage; from experimental results on a database comprised of 389 documents, we have obtained over 90% retrieval accuracy on most configurations, and retrievals required on average less than 100 ms and about 200 MB of memory.

Keywords: pARagraph, Camera-Based Document Image Retrieval, LLAH, SRIF, Augmented Reality

List of Figures

List of Tables

List of Acronyms

LLAH Locally Likely Arrangement Hashing 11

SRIF Scale and Rotation Invariant Features for Camera-Based Document Image Retrieval 11

RANSAC Random Sample Consensus 21

Contents

1

Introduction

The task of finding document images relevant to the user is known as Document Image Retrieval [1]. This is a very broad problem, however, and there are several more specific tasks, distinguished based on factors such as: needs of the user, input format and presence or absence of restrictions on it. The techniques utilized for recognizing the text contained in document images, for instance, are fundamentally different from the ones used for identifying identical documents. Furthermore, the techniques viable for document recognition are distinct when the input is a scanned image or a photograph captured by a camera. This work is focused on this latter version of the problem, in which the goal is to recognize, based on a photograph captured by a camera, which document is shown in the image. Formally, this process is known as Camera-Based Document Image Retrieval [1].

On the last decade, this has become an exceedingly relevant task, due to digital cameras becoming extremely accessible devices [3][4], mainly as a result of their inclusion in smartphones. Also, the quality of the cameras contained in these devices has improved considerably, so that, currently, most people always carry with themselves a camera with at least 1 megapixel [5]. Therefore, this dissemination and increase in resolution created a great opportunity for applications directed towards mobile devices.

There are several motivations for this sort of recognition, especially in mobile devices. Chief among those is the recovery of additional content that is relevant to the document, such as supplementary text, annotations, figures and videos. Moreover, the documents can be used as markers [6], functioning essentially as barcodes or QR codes. Finally, it is also possible to compute the projective transformation suffered by the document in the photographed image, in relation to its version in the database. This allows both the rendering of any relevant virtual content over the image, using the same transformation, and the identification in the database document of sections of the image selected by the user [7].

The restriction that input images are photographed by a camera implies several additional challenges that must be dealt with, mainly: low resolution, adjustment of camera focus, motion blur, uneven lighting, occlusion and, especially, projective distortion [3][1]. This last challenge is particularly important, because the existence of a projective distortion invalidates the majority

of the techniques that are not based on projective invariants.

In the literature, the technique called Locally Likely Arrangement Hashing (LLAH) [8][1] is largely utilized for the Camera-Based Document Image Retrieval problem. LLAH works well for images in low resolution, is robust to uneven lighting and severe occlusion and can recognize documents in spite of the presence of projective distortions. Furthermore, the required amortized run time by the algorithm for a particular search is constant in relation to the size of the database, allowing it to be executed in real time. Overall, LLAH is a good solution for the problem and several improvements were published for the algorithm over the years [2][4][9][10]. An adaptation of LLAH for smartphones was also published [4], which uses a client-server architecture and many optimizations, so that the algorithm could run in real time on the limited hardware.

The greatest restriction of LLAH is the considerable amount of memory that it requires, which limits its use in mobile devices without a client-server architecture. Even with the improvements proposed by the authors, mobile applications using the algorithm are limited to small databases.

As far as we know, there is no better alternative to LLAH for the Camera-Based Document Image Retrieval problem. Dang et al. published Scale and Rotation Invariant Features for Camera-Based Document Image Retrieval (SRIF) [11][3], a descriptor inspired by LLAH that can be used as a substitute for the technique's traditional one. SRIF allows the system to run faster, but it is only invariant to rotations and changes in scale; not to projective distortions in general.

1.1 Objective

The goal of this work is to investigate the Camera-Based Document Image Retrieval problem and implement a solution that can be used effectively in mobile devices. For that reason, the optimizations will be mainly focused on reducing the required amount of memory, which is the greatest limiting factor. The implementation will be in the format of a library called pARagraph; it can be used in various platforms, but it is outside the scope of this work to implement actual applications for them. Besides recognizing documents, the library will also be capable of computing the projective transformation between their query and database versions, which will make it useful for applications involving Augmented Reality [7]. Finally, the implementation will be done in C++, which was chosen both for its efficiency and compatibility with multiple platforms.

In this work, we will also perform several experiments in order to evaluate the performance of pARagraph according to some metrics: accuracy of the document retrieval process, number of frames per seconds (processing time) and memory consumption.

1.2 Structure

This work is divided into 5 chapters, including this introduction. Chapter 2 describes the LLAH technique and the SRIF descriptor, as well as the optimizations proposed for the former which were adopted in pARagraph. Next, Chapter 3 presents the pARagraph library, discussing its architecture, decisions taken during the project and implementation details in general. Then, Chapter 4 explains both the experiments conducted to measure the library's performance as well as its results. Finally, Chapter 5 discusses the conclusions taken from this work, how well the goals were reached, and possible future works.

2

Fundaments

In relation to the problem of Camera-Based Document Image Retrieval, the technique called LLAH is efficient in terms of accuracy, run time and scalability [3]. Particularly, its main advantages are that it executes in real time and the query time is amortized constant independently of the number of documents in the database. However, there are also limitations inherent to it, which are mainly the requirement of considerable amounts of memory and the fact that, because the recognition is based on portions of text, it performs poorly for documents comprised mostly of images [1].

As far as we know, there is no better alternative to LLAH for solving this problem. That said, we also have the option of using the SRIF descriptor in it, which reduces both runtime and accuracy; this trade-off might be beneficial for some applications.

In this work, we focused mainly on LLAH, but also implemented SRIF in order to offer flexibility to the pARagraph user. In the remainder of this chapter, we will describe both methods, as well as some of the optimizations for the former that were proposed by its authors.

2.1 LLAH

The LLAH method was proposed by Nakai, Kise and Iwamura in 2005 as a faster alternative to Geometric Hashing, besides having greater discrimination power than other existing techniques [8]. Roughly, LLAH works by storing data extracted from documents in a hash table, and then retrieving them through a voting scheme. The hash keys are computed from geometric invariants, which are themselves computed from the neighbourhood of estimated centroids of word regions in the document image.

In [1], the authors proposed the use of an affine invariant instead of a projective one. This is based on the fact that, for a small area such as the neighbourhood of a point, the projective distortion between the images can be approximated by an affine transformation. Therefore, LLAH works well with both projective and affine invariants, but is faster when the latter is used. The projective invariant used initially by LLAH is the cross ratio, computed from 5 coplanar

points a, b, c, d and e:

$$\frac{A(a,b,c)A(a,d,e)}{A(a,b,d)A(a,c,e)} \tag{2.1}$$

where $A(a,b,c)$ is the area of the triangle formed by points a, b and c. Also, the affine invariant which was proposed later is a ratio of triangle areas, computed from 4 coplanar points a, b, c and d:

$$\frac{A(a,c,d)}{A(a,b,c)} \tag{2.2}$$

Since its initial version, there have been several improvements and modifications proposed for LLAH relating to its processing time [2][4] and memory consumption [2][9][10], besides efficiency of the descriptors [9][10] used. In this work, we used only the improvements proposed in [2], but intend to also adapt the pipeline described in [4] in a future work. As for the modification to reduce memory consumption proposed in [9][10], we decided not to implement it, since it could reduce retrieval accuracy for images of relatively low resolution, which can be the case for photographs taken with smartphone cameras. Regarding the improvements to the discrimination power of the descriptors, they require larger vectors, and thus more memory for storing the hash table. Since we intend to reduce the memory consumption as much as possible, due to the restrictions of mobile devices, we decided to also not implement these modifications.

In the remainder of this section, we will present the original version of LLAH, which is the same for both projective and affine invariants. Then, we will describe the techniques proposed by its authors which were adopted in this work, regarding improvements to the running time and memory consumption.

2.1.1 Original LLAH

The processing performed by LLAH is divided into storage and retrieval, as can be seen in Figure 2.1. In both cases, feature points are extracted from the document image, then descriptors are computed and hash keys are obtained from them. This is where the storage and retrieval algorithms diverge, with the former storing data from the document in the hash table and the latter accessing this data to retrieve the correct document through a voting scheme.

In the original LLAH, the database consisted of a hash table where each position contains a list Figure 2.2. The elements stored in the table are then the necessary information for the voting: a document ID, a point ID and a vector r, which is the descriptor obtained from a feature point.

2.1.2 Feature Point Extraction

The feature points used in LLAH are the centroids of connected components corresponding to estimated word regions in the document image; their calculation is performed in 5 steps, which are as follows. First, an adaptive thresholding is applied to the image, which must be in

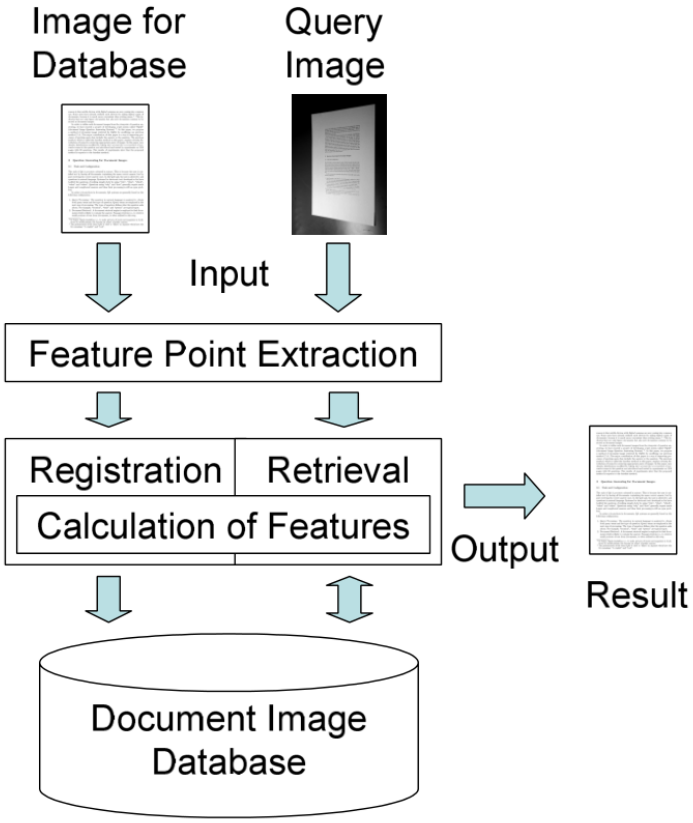

Figure 2.1: Overview of LLAH processing [1].

grayscale, generating a binary image. Next, the average size of a character is estimated, and then a Gaussian standard deviation, which is used in the following step, is computed from it. The average character size is estimated as the square root of the mode of connected components' areas in the image, and the standard deviation is calculated using the following equation [12]:

$$0.3 \cdot (\frac{c}{2} - 1) + 0.8 \tag{2.3}$$

where c is the estimated average character size.

Then, a Gaussian smoothing filter is applied to the image, in order to connect characters in the same word. Finally, a second adaptive threshold is performed, so that each word becomes a single connected region.

2.1.3 Descriptors

The descriptor used in LLAH is a vector of discrete values, which describes a feature point. For a given feature point p, the computation of its descriptor r is based on the m nearest neighbours of p, and will be detailed next. First, the m neighbours p_i of p are sorted in relation to the angle between the vectors $\overrightarrow{pp_i}$ and $\overrightarrow{pp_0}$, for an arbitrarily chosen point p_0. Then, the C_f^m combinations of f ($f < m$) points are computed, with f being the number of points necessary

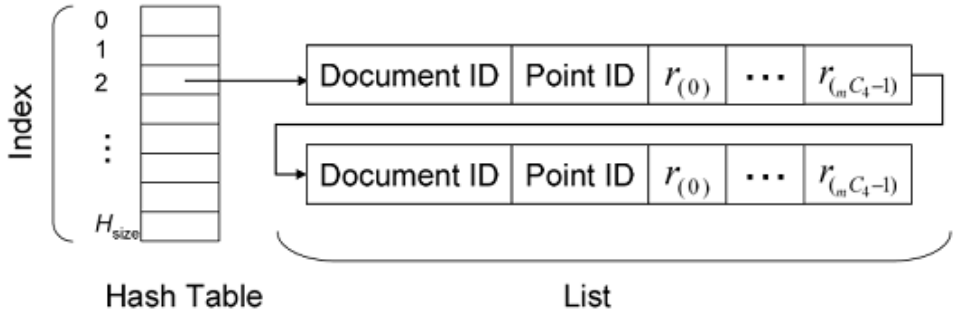

Figure 2.2: Hash table used in the original LLAH [2].

for computing the geometric invariant, i.e., 5 points for the cross ratio and 4 points for the ratio of triangle areas. The combinations are then sorted lexicographically, with point comparisons being determined according to the previous sorting by angle. Finally, each value $r_{(i)}$ from the r vector is computed by applying the geometric invariant to the i$^{\text{th}}$ combination of points in the lexicographical ordering, and discretizing the resulting value (Figure 2.3).

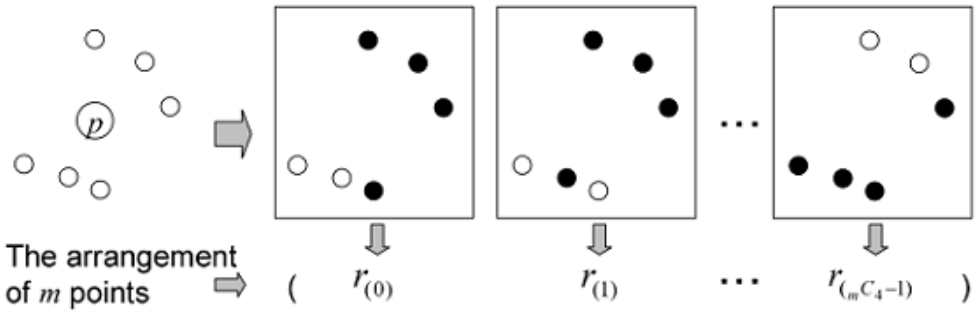

Figure 2.3: The r vector is computed based on the combinations of f points from among m neighbours, with each value $r_{(i)}$ being calculated from the i$^{\text{th}}$ combination following a lexicographical ordering. In the example, the initial point p_0 is the topmost one [2].

The discretization of values from the geometric invariant is performed through the use of t thresholds. These values are computed in a preprocessing step, and used to effectively partition the real line in $t+1$ intervals, numbered from 0 to t. The discretization function used in LLAH, then, simply corresponds to a mapping of all values in an interval to its number.

The procedure just described is not enough by itself, because the m nearest neighbours of a feature point p may be different after a projective distortion. LLAH deals with this problem by finding the n ($n > m$) nearest neighbours of p and then repeating the procedure for each combination of m elements from among them (Figure 2.4); consequently, C_m^n descriptors are found for each feature point. These extra steps are performed in the hope that at least one of the C_m^n combinations of neighbours will be found during both storage and retrieval.

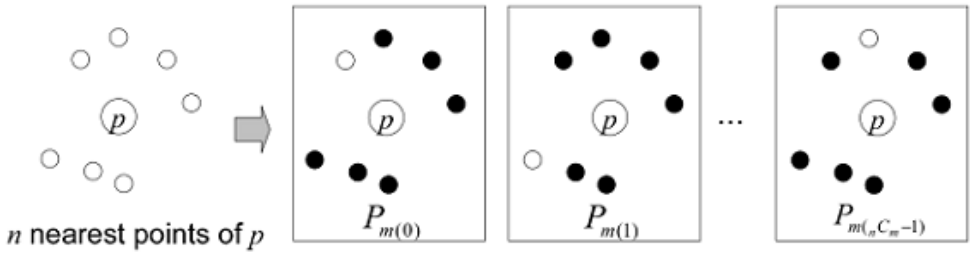

Figure 2.4: All the combinations of m points from among n nearest neighbours are examined. In the example, $m = 6$ and $n = 7$ [2].

2.1.4 Storage

The algorithm used in LLAH for storing a new document in the hash table is as follows. First, the feature points are extracted from the input image, as described previously. Next, for each of the C_m^n nearest neighbour combinations of each feature point, a descriptor is computed, as explained in the previous section. When one is found, a hash key H_{index} is then derived from it, through the equation:

$$H_{index} = \left(\sum_{i=0}^{C_f^m} r_{(i)} \cdot q^i \right) mod\ H_{size} \tag{2.4}$$

where q is the level of quantization of the invariant and H_{size} is the size of the hash table, both parameters to the method. Then, it is stored in the list found at the position H_{index} of the hash table: the document ID, the feature point ID and the descriptor r. Algorithm 2.1 describes the storage algorithm.

Algorithm 2.1 Storage algorithm used in the original LLAH [2].

for each $p \in \{$All feature points in a database image$\}$ **do**
  $P_n \leftarrow$ A set of the n nearest points of p
  for each $P_m \in \{$All combinations of m points from $P_n\}$ **do**
    $L_m \leftarrow (p_0, \cdots, p_i, \cdots, p_{m-1})$ where p_i is an ordered point of P_m based on the angle from p to p_i with an arbitrarily selected starting point p_0
    $(L_f(0), \cdots, L_f(i), \cdots, L_f(C_f^m - 1)) \leftarrow$ A lexicographically ordered list of all possible $L_f(i)$ that is a subsequence consisting of f points from L_m
    for $i = 0$ to $C_f^m - 1$ **do**
      $r_{(i)} \leftarrow$ a discretized affine invariant calculated from $L_f(i)$
    end for
    $H_{index} \leftarrow$ The hash index calculated by Equation (2.4)
    Store the item (document ID, point ID, $r_{(0)}, \cdots, r_{(C_f^m-1)}$) using H_{index}
  end for
end for

2.1.5 Retrieval

The retrieval algorithm used in LLAH is very similar to the storage one, and is as follows. Initially, as when storing a document, feature points are extracted from the image. Then, for each feature point, all C_m^n combinations of its nearest neighbours are computed. For a given combination, all of its m points are examined in turn as the initial point p_0, and used to generate a descriptor r. A hash key is then derived from this descriptor, also as in the storage algorithm. Next, the hash table is accessed at the position corresponding to the key, and the list contained there is retrieved. For each element in it, a vote is given to the corresponding document ID if three conditions are met. The conditions used by the authors of LLAH are meant to avoid some kinds of erroneous votes and, for an entry consisted of a descriptor r', a feature point ID i and a document ID j, they are [1]:

1. The vectors r and r' are equal.
2. It is the first time to vote for the document j with this feature point.
3. It is the first time to vote for the feature point i of the document j.

After all the C_m^n potential votes have been handled, the most voted document is returned. The retrieval algorithm can be seen in Algorithm 2.2.

Algorithm 2.2 Retrieval algorithm used in the original LLAH [2].

for each $p \in$ {All feature points in a query image} **do**
  $P_n \leftarrow$ A set of the n nearest points of p
  for each $P_m \in$ {All combinations of m points from P_n} **do**
    for each $p_0 \in P_m$ **do**
      $L_m \leftarrow (p_0, \cdots, p_i, \cdots, p_{m-1})$ where p_i is an ordered point of P_m based on the angle from p to p_i with a starting point p_0
      $(L_f(0), \cdots, L_f(i), \cdots, L_f(C_f^m - 1)) \leftarrow$ A lexicographically ordered list of all possible. $L_f(i)$ that is a subsequence consisting of f points from L_m
      for $i = 0$ to $C_f^m - 1$ **do**
        $r_{(i)} \leftarrow$ a discretized affine invariant calculated from $L_f(i)$
      end for
      $H_{index} \leftarrow$ The hash index calculated by Equation (2.4)
      Look up the hash table using H_{index} and obtain the list
      for each item of the list **do**
        if Conditions to prevent erroneous votes are satisfied **then**
          Vote for the document ID in the voting table
        end if
      end for
    end for
  end for
end for
Return the document image with the maximum votes

2.2 Run Time Improvements

We will now describe the improvement related to processing time that was adopted in this work. In the retrieval algorithm shown in Algorithm 2.2, all the points in a combination are examined as the initial point p_0. This is necessary due to the fact that L_m depends on the choice of p_0, so the same combination can generate a different list after a rotation. However, in [2] the authors have proposed a manner of selecting the initial point p_0 in a way that is invariant to rotation, eliminating the need to examine all m neighbours.

The selection strategy is shown in Figure 2.5. The m neighbours are enumerated from 0 to $m-1$, following a clockwise ordering around the feature point p. Then, for each neighbour i, the value $s_{(i)}$ is computed, which is the geometric invariant calculated from i and its $f-1$ successors. Finally, the point i corresponding to the greatest number $s_{(i)}$ is selected as p_0. In case of a tie between two or more points, the next values $s_{((i+1) \bmod m)}$ are used to choose one of them. If they are still tied, the values $s_{((i+2) \bmod m)}$ are used and so on. With this modification, the authors of LLAH estimated a reduction of 60% in retrieval run time.

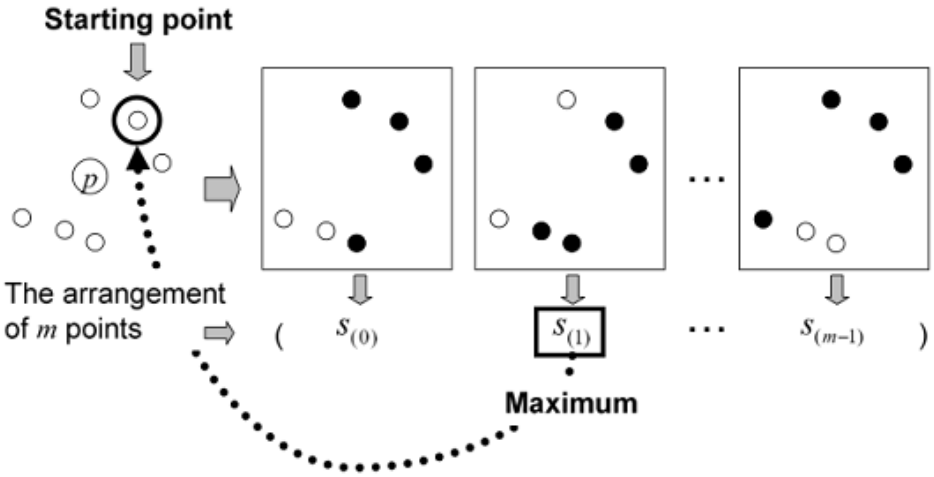

Figure 2.5: The feature point i corresponding to the greatest value $s_{(i)}$ is selected as the initial point [2].

2.3 Memory Consumption Improvements

In this work we also adopted the improvement proposed in [2] to reduce the amount of memory required by LLAH. In the original method, all extracted descriptors are added to the hash table, but those inserted in positions in which collisions occur are less important, since they have less discrimination power. Also, Nakai et al. estimated that the positions in the table which contain collisions are only 28% of the nonempty positions. Therefore, it is possible to remove from the table the data in those positions, without greatly affecting the accuracy. With this removal, the hash table will contain at most one element per position, and it becomes possible to

simplify the data structure used to represent it (Figure 2.6). Because each position contains either one or no elements, the data can be store directly in the table, without using a list. Furthermore, the descriptors r are no longer needed, since they were kept in order to identify the correct entry in the list of elements. Consequently, the stored elements are simplified to a tuple of IDs, for the document and the feature point, and specific values can be used to indicate that a given position is empty or that a collision occurred in it. With these modifications, the authors estimated roughly an 80% reduction in the required amount of memory.

An additional consequence of this modification in the hash table is that, since the descriptors are no longer stored, the Condition 1 from the retrieval algorithm no longer applies, and it is removed.

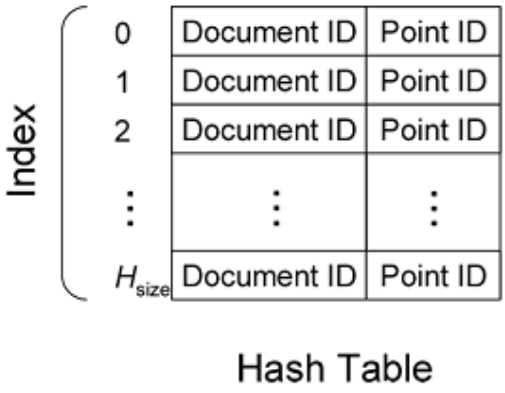

Figure 2.6: Simplified hash table [2].

2.4 SRIF

In 2015, Dang et al. published SRIF [3][11], a new descriptor for feature points based on a geometric invariant different from the one used in LLAH. SRIF is a vector of discrete values analogous to the r vector used in LLAH as a descriptor, and they have only two differences: the geometric invariant and discretization function which are used.

The discretization function used in SRIF is simpler than its counterpart in LLAH. For a real number x, computed with the geometric invariant, it is discretized with the formula:

$$2 \cdot trunc(x) + round(x - trunc(x)) \tag{2.5}$$

The geometric invariant used in SRIF is computed from only a feature point P and two other points, P_i and P_j, with the restriction that all three of them must be coplanar (Figure 2.7). Let $|\overrightarrow{PP_i}|$ and $|\overrightarrow{PP_j}|$ be the magnitudes of the two vectors $\overrightarrow{PP_i}$ and $\overrightarrow{PP_j}$, respectively, and θ_{ij} the angle between these vectors. $L_{min_{ij}}$ and $L_{max_{ij}}$ are then defined as $min(|\overrightarrow{PP_i}|/|\overrightarrow{PP_j}|, |\overrightarrow{PP_j}|/|\overrightarrow{PP_i}|)$ and $max(|\overrightarrow{PP_i}|/|\overrightarrow{PP_j}|, |\overrightarrow{PP_j}|/|\overrightarrow{PP_i}|)$, respectively. Thus, SRIF can be used with one of two geometric

invariants:

$$\theta_{ij} \cdot L_{min_{ij}} \tag{2.6}$$

$$\theta_{ij} \cdot L_{max_{ij}} \tag{2.7}$$

Unfortunately, these values are not invariant to projective or affine transformations, but only to rotations and changes of scale. This makes SRIF a less suitable descriptor than its counterpart in LLAH for photographs captured by cameras in arbitrary positions. However, as only two neighbouring points are necessary, the system can use smaller values for the n and m parameters. Consequently, the use of this descriptor accelerates considerably the LLAH method, because less combinations are examined.

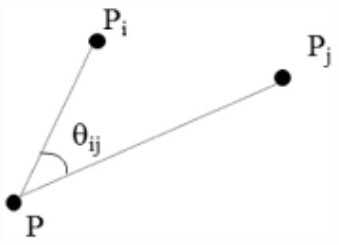

Figure 2.7: Points used for computing the invariant used in SRIF, as well as the angle between the vectors $\overrightarrow{PP_i}$ and $\overrightarrow{PP_j}$ [3].

Finally, Dang et al. proposed the use of SRIF in a system identical to the original LLAH's, except in two aspects: the descriptor used is SRIF and the system also computes the projective transformation suffered by the document. The latter is performed at the end of processing, after the voting has been completed, and also serves as an additional validation step, because it will not be possible to find one if the document retrieved by the voting is incorrect. In that case, the system ignores the most voted document and examines the second most voted, and so on. In the proposed system, the computation of homographies is performed by the Random Sample Consensus (RANSAC) technique [13].

3

pARagraph

pARagraph is an implementation of LLAH, created bearing in mind the restrictions of mobile devices. It is provided as a library written in the C++ language, which was chosen both for its efficiency and its compatibility with multiple platforms. For the same reasons, we used the OpenCV [14] library for image processing and some geometrical calculations. In the future, pARagraph will be available for commercial use as a Voxar Labs technology.

Due to the intent of running on mobile devices, the library was implemented with optimizations to reduce both the run time and the amount of memory required. The latter, however, were prioritized in this work, due to the fact that the LLAH technique requires a considerable amount of memory for its database, and this is a greatly restricted resource on mobile devices. Some of the adopted optimizations were already described in Chapter 2, as well as the reasons why other improvements proposed for LLAH were not used. The remaining, such as the use of parallelism, paging of the hash table, and modifications on its data structure, will be presented in this chapter.

An overview of the processing executed in pARagraph can be seen in Figure 3.1. As in LLAH, there are two workflows, one for storing and another for retrieving documents, and the steps of extracting feature points and calculating hash keys are shared among them. However, there are three modifications in relation to the processing performed by the original LLAH technique. Firstly, the homography is found after the voting and returned by the retrieval algorithm, together with the ID of the most voted document. Secondly, all hash keys are computed before the table is accessed. Thirdly, the keys are grouped by their page number of the hash table, also before the table is accessed.

These last two modifications are necessary because the library uses a paging scheme to reduce the amount of memory occupied by the hash table. As a result of this, all accesses to a particular page of the table must be dealt with in sequence, so as to avoid unnecessary page switching.

From among the different steps in the workflow, the computation of hash keys is the only one that can be parallelized. In our implementation, this step's workload is divided among 4 threads, and each is responsible for handling a subset of the centroids.

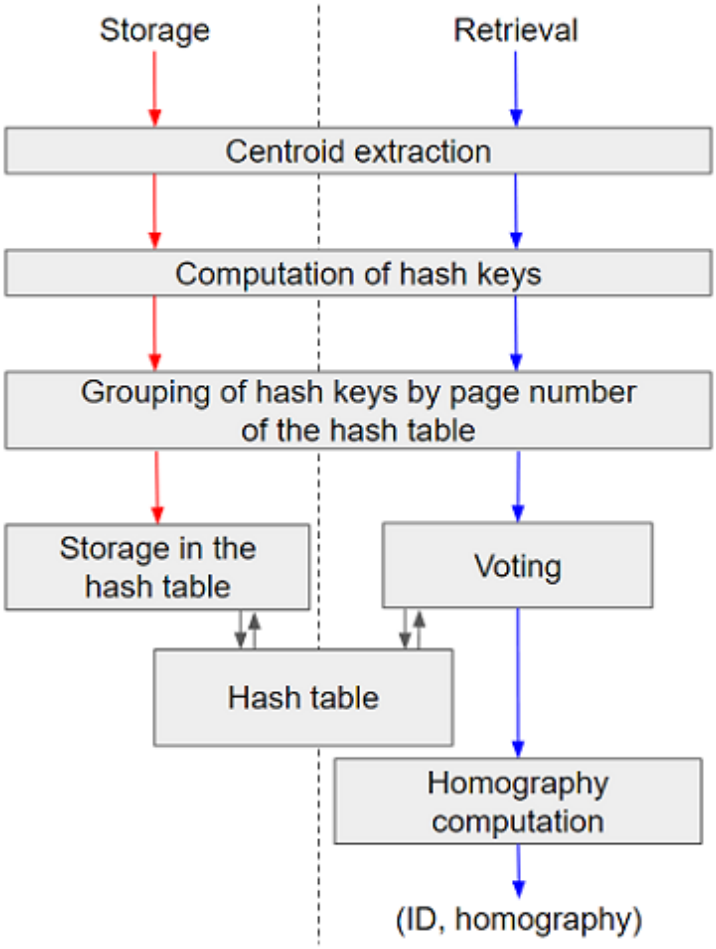

Figure 3.1: Overview of processing for storage, on the left, and retrieval, on the right.

In the remainder of this chapter, we will examine the various modules that comprise the pARagraph library and detail its differences to the original LLAH method. During this process, we will explain the workings of each step of the workflow and present both the storage and retrieval algorithms.

3.1 Centroid Extraction

The feature points used in pARagraph are still the centroids of the word regions, and the algorithm for their extraction can be seen in Algorithm 3.1. In its implementation, we used OpenCV for the functions pertaining to image resizing, adaptive thresholding, Gaussian smoothing and computation of connected components.

The processing performed by the algorithm is as follows. Initially, the input image is resized in such a way that its smaller dimension has 1080 pixels, without modifying the aspect ratio. This resizing is necessary due to the fact that both the formula for the standard deviation used in the Gaussian blur and the heuristics to filter noise assume that there is little variation in the input images' resolutions. The current values were thus empirically fine-tuned for images of

approximately 1080p.

This choice of resolution was the result of a trade-off; it is, at the same time, not too low and not much greater than the smallest expected resolutions for photographs captured by mobile devices. When an image is resized to a smaller resolution, the algorithm is essentially unaffected. However, when an image is enlarged that way, it is somewhat blurred. This additional blurring, when added to the Gaussian smoothing which is part of the algorithm, affects considerably the performance of retrieval. At the same time, performance is also reduced for images with too low resolutions, because the error inherent to the discretization in pixel coordinates affects the results of the invariants. We could somewhat compensate for this error, in order to resize the images to a lower resolution and avoid enlarging them as much as possible, by using less thresholds in the discretization function and thus increasing each interval. However, this would also reduce performance, by decreasing the discrimination power of the descriptors.

After the resizing, an adaptive thresholding is applied to the image, and the average length c of a character is estimated as the square root of the mode of connected components' area values. In this estimation, components with bounding boxes of area at most 30 pixels are considered noise and ignored. After that, a Gaussian smoothing filter is applied to the image, in order to merge the characters of each word. The smoothing is done with a kernel of size c and a standard deviation of $c/4$. Following the smoothing, a second adaptive thresholding is applied to the image, so that each word becomes a single connected component. After that, the connected components are computed once more, and their centroids are stored together with the width and height of their bounding boxes. Then, a second noise filtering step occurs, and feature points with bounding boxes of area at most 50 are removed. Finally, a grid object is instantiated, using as its cells' width and height values the medians of the stored widths and heights, respectively. Figure 3.2 shows an image during the different steps of feature point extraction.

3.2 Geometric Invariants and Discretization Function

We implemented three geometric invariants for pARagraph: the cross ratio, the ratio of triangle areas and the SRIF invariant. For this last one, we used the version given by Equation 2.7, since Dang et al. claimed in [3] that it could produce better results.

The values found by computing the invariants are discretized as normal for LLAH. We use 9 threshold values, computed in a preprocessing phase, and thus divide the real line in 10 intervals, which are numbered from 0 to 9. The discretization of a value therefore consists simply in finding the number of the interval which contains it. The thresholds are stored sorted in an array, which allows us to perform this search efficiently with a binary search.

Computing the thresholds for a given database is done in a preprocessing step and will be described next. Initially, each document has its centroids and corresponding descriptors computed as normal, but the values found as results of calculating the geometric invariant are stored in an array. Next, this data is sorted, and its deciles are then chosen as the thresholds.

Algorithm 3.1 Centroid extraction algorithm.

Input: Image *Img* in grayscale
Output: List of centroids; grid containing all centroids

function COMPUTECENTROIDS(*Img*)
    Resize *Img* such that its smallest dimension becomes 1080p, while maintaining its aspect ratio
    $Img \leftarrow adaptiveThreshold(Img)$
    $ModeAreas \leftarrow$ mode of area values from all connected components with bounding boxes of area greater than 30
    $CharSize \leftarrow sqrt(ModeAreas)$
    $\sigma \leftarrow 0.25 \cdot CharSize$
    $Img \leftarrow gaussianBlur(Img, CharSize, \sigma)$
    $Img \leftarrow adaptiveThreshold(Img)$
    $Centroids, Widths, Heights \leftarrow$ empty lists
    for each connected component *ConComp* with a bounding box of area greater than 50 **in** *Img* **do**
        $Centroids.append(ConComp.centroid())$
        $Widths.append(ConComp.boundingBox.width())$
        $Heights.append(ConComp.boundingBox.height())$
    end for
    $MedianWidth \leftarrow$ median of the values in *Widths*
    $MedianHeight \leftarrow$ median of the values in *Heights*
    Grid $\leftarrow$ grid created on the area occupied by the image and using $MedianWidth$ and $MedianHeight$ for width and height of cells, respectively
    Insert in *Grid* all points contained in *Centroids*
    Return *Centroids* and *Grid*
end function

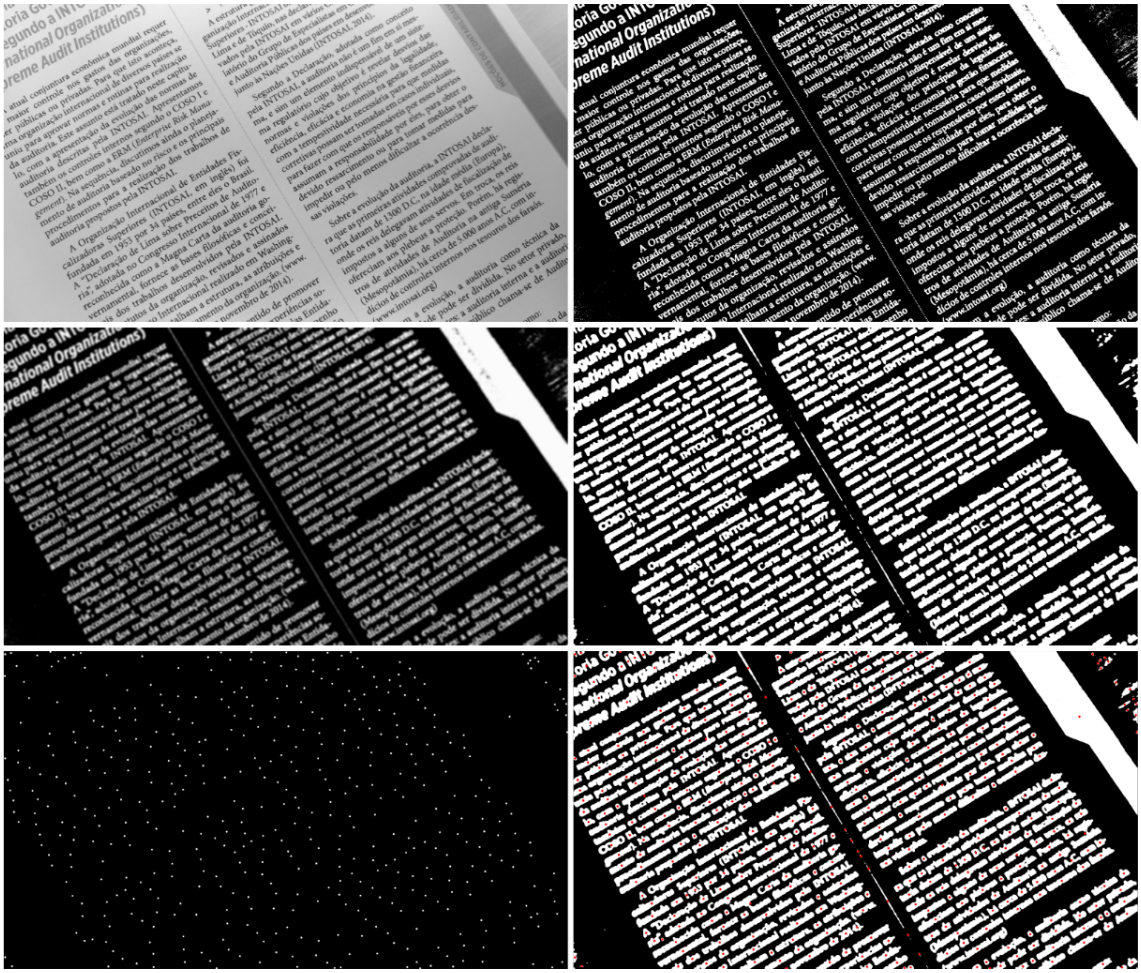

Figure 3.2: Centroid extraction procedure. From left to right and top to bottom: original image, after first adaptive thresholding, after Gaussian smoothing, after second adaptive thresholding, extracted centroids, word regions overlaid with the centroids.

Finally, these values can be reused for other databases, but this might affect retrieval accuracy.

In the particular case of the SRIF geometric invariant, we implemented two approaches for discretizing values: Equation (2.5) and the technique that uses thresholds. We also computed the thresholds for a database of 10 documents, in order to compare both approaches. The results were similar, and the values obtained as thresholds were close to multiples of 0.5, which suggests that both techniques are approximately equivalent for use with this invariant.

3.3 Data Structures

There are several auxiliary data structures used during pARagraph's processing, such as classes for points and vectors in two and three dimensions and classes for representing graphs. There are, however, two data structures that deserve special attention for being more complex, and those will be presented in this section.

3.3.1 Hash Table

The original LLAH uses a hash table implemented as a vector, in which every position contains a linked list. There is also a modified version, described in Section 2.3, which uses a simplified table; this is made possible by emptying positions in which collisions occurred. As a result, there is no longer the need to keep a list at every position, which reduces the stored data to a pair of IDs. However, though this modification greatly reduces memory consumption, it is still necessary to allocate enough memory for the entire vector, even when the table is sparse. Because of this, this modified table is used in pARagraph, but it is represented in a virtual fashion, with only the nonempty positions being stored. This data is stored in a map, implemented as an instance of the class `std::unordered_map` from C++'s standard library, instead of in an array of size H_{size}.

The benefit of this strategy is that the `std::unordered_map` class stores the data in an internal array which is proportional to the number of nonempty positions in the hash table, which avoids the considerable amount of unused memory if the table is sparse. However, it becomes necessary to also store the computed hash keys, which could cause a greater amount of memory to be consumed if the table is not sparse. Fortunately, that is expected to almost never be the case, for two reasons. Firstly, since the computed hash keys are very large values, H_{size} must also be a very large number. For a database containing 10,000 documents, for instance, Nakai et al. have chosen $H_{size} = 1.28 \cdot 10^8$ [1]. Secondly, the maximum database size which is feasible for use in mobile devices is limited, due to the restricted amount of memory and disk space of these platforms.

The `Hash_Table` class in the library implementation represents the virtual hash table by storing not only a mapping of its nonempty positions, but also a set of its forbidden hash keys. These keys correspond to positions where collisions occurred, and the set containing them is implemented as an instance of the `std::unordered_set` class, which is also from C++'s standard library. Every time a collision is caused by an insertion, the key is added to this set and both elements associated with it are removed from the table. Furthermore, insertions in which the key associated with the new element is a forbidden one are not allowed, and return failure. This architecture comprised of a map and a set was chosen in order to further reduce memory consumption, by avoiding storing a reserved pair of IDs in the map in order to signal a forbidden position in the table.

There are two more implementation differences between pARagraph and LLAH. In the latter, the data stored in the table is comprised of tuples containing a document ID and a feature point ID. However, the point IDs are used only in the condition checks for erroneous votes in the retrieval algorithm, and they serve only to distinguish between feature points of the same document. Because of this, we replaced them with the corresponding point coordinates, which are necessary in order to compute the homography and equally capable of distinguishing one feature point from another.

The other difference between implementations is a paging scheme for the virtual hash table used in pARagraph. It is divided into a number of pages specified by the user, and every page contains its respective set and map, which contain the data pertaining to its subset of the table. For a table divided into 3 pages, for instance, pages 0, 1 and 2 contain the data corresponding to the keys in the intervals $[0, \frac{H_{size}}{3})$, $[\frac{H_{size}}{3}, \frac{2 \cdot H_{size}}{3})$ and $[\frac{2 \cdot H_{size}}{3}, H_{size})$, respectively.

3.3.2 2D Grid

The computation of a given centroid's descriptors requires finding its nearest neighbours, which in turn demands an acceleration structure to be done efficiently. To that end, there are various alternatives that could be used [15], such as: some BSP (Binary Space Partitioning) tree, such as a kd-tree; some BVH (Bounding Volume Hierarchy); a quadtree; and a grid. That said, because the searches are performed in a space with known dimensions (the image dimensions) and the centroids are well spread over it, a grid performs searches efficiently [16]. Due to both this efficiency and its implementation simplicity, we chose a grid as the acceleration data structure used in pARagraph.

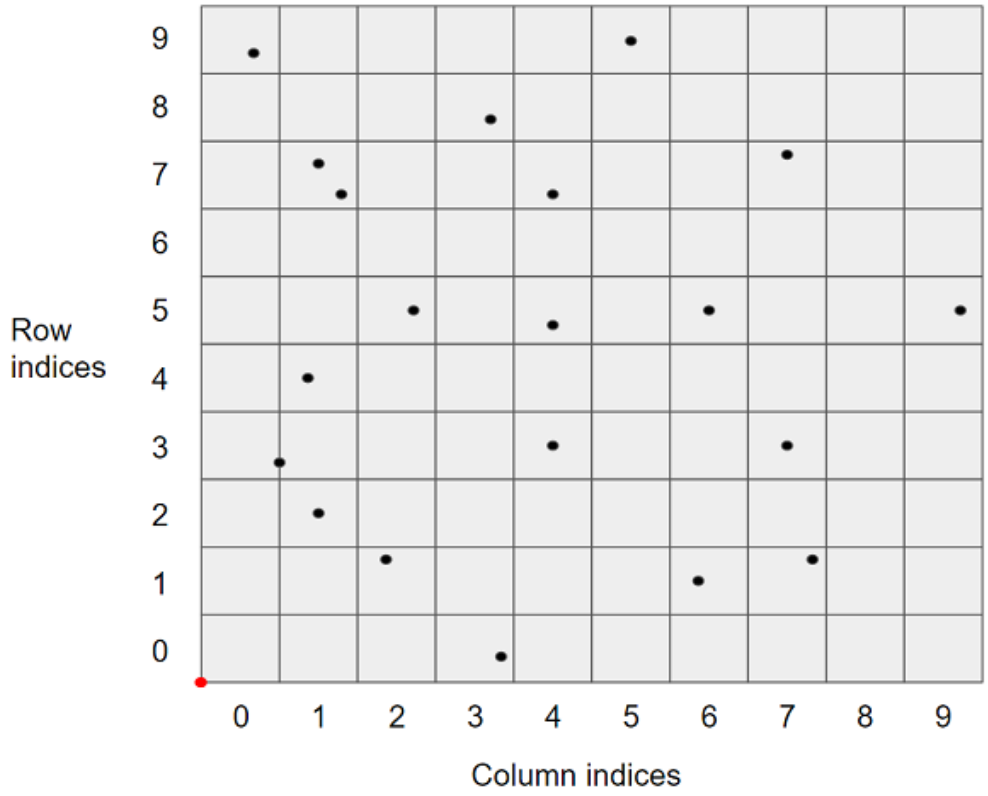

Figure 3.3: Example of a region divided by a grid, with each cell storing zero or more points. The grid origin is marked in red.

A two dimensional grid divides a region in rectangular cells of fixed, but not necessarily equal, width and height (Figure 3.3). Each cell is identified by two indices and stores points in some data structure, usually a list. A grid also contains an origin, which is a vertex of its

bounding box, from which the numberings of rows and columns begin.

Depending on the data, there could be several empty cells, which would result in a considerable amount of wasted memory if cells are stored in a matrix. If that is the case, a common optimization is to keep the cells in a hash table instead of a matrix [16], so that only the nonempty ones need to be stored. This strategy is used in our implementation due to the fact that the centroids are only spread on the portion of the image occupied by the document, which is not necessarily its entirety. Furthermore, they are reasonably far apart, so there might be empty cells between them if cell dimensions are small. The formula we used to generate a hash key from an index (i, j), with i being the row index and j being the column index, is given by:

$$j + (i \cdot N_x) \tag{3.1}$$

where N_x is the number of grid columns, i.e., the number of divisions in the X axis.

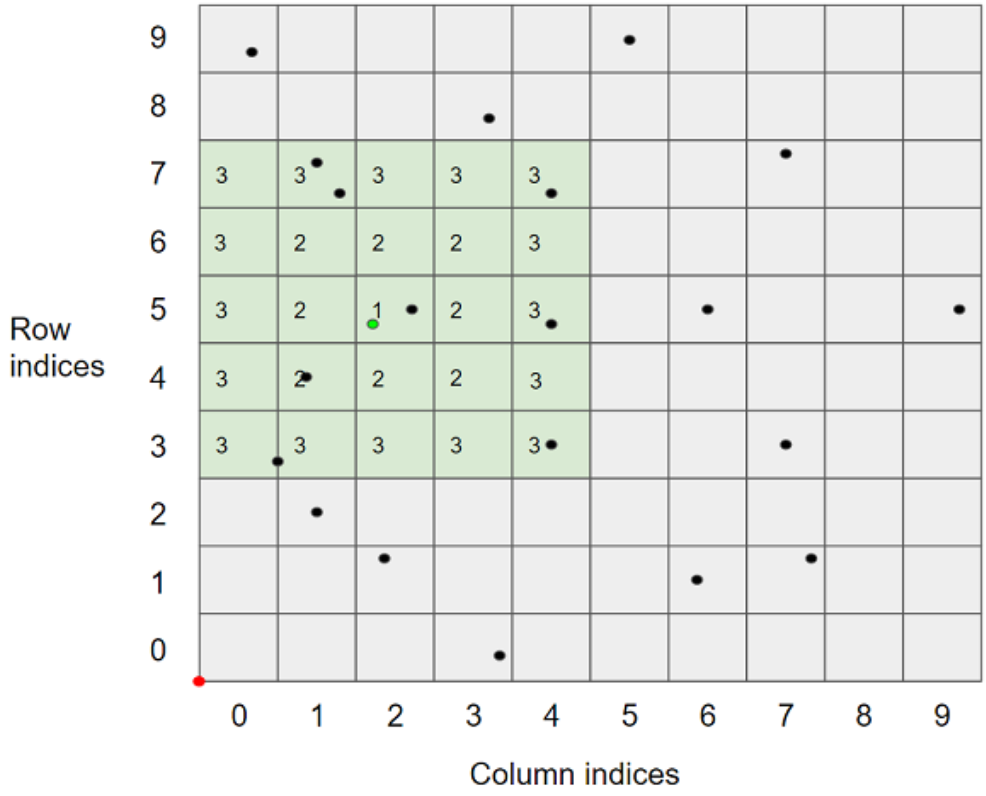

Figure 3.4: Example search for the 8 nearest neighbours of an input point (in light green). Visited cells (in dark green) are numbered by the iteration in which they are visited.

The procedure for searching for the nearest neighbours of a given centroid (x, y) can be seen in Algorithm 3.2 and is as follows. Firstly, the cell containing the centroid is found, with its index (i, j) being calculated by:

$$j = \frac{x - x_o}{S_x} \tag{3.2}$$

$$i = \frac{y - y_o}{S_y} \tag{3.3}$$

where (x_o, y_o) is the origin of the grid and S_x and S_y are the width and height of a cell, respectively. Then, neighbouring cells at a fixed distance of cell (i, j) are examined, and all points contained in them are stored. This process is repeated for search distances increasingly larger, until the correct number of neighbouring points have been found. An example search, for 8 nearest neighbours, is shown in Figure 3.4.

Algorithm 3.2 Function for finding the nearest neighbours of a point.

Input: Point P; number of nearest neighbours N
Output: List of neighbours

```
1:  function NEARESTNEIGHBOURS(P, N)
2:      Find the index (i, j) of the cell which contains P, using Equations (3.2) and (3.3)
3:      Result ← empty list
4:      Dist ← 0
5:      while Result.size() < N do
6:          Neighbours ← empty list
7:          for each cell coordinate Coord at a distance Dist from (i, j) do
8:              if Coord is within the grid then
9:                  Add all points in cell Coord to Neighbours
10:             end if
11:         end for
12:         Remaining ← min(Neighbours.size(), N − Result.size())
13:         if Remaining < Neighbours.size() then
14:             Order the points in Neighbours by their distance to P
15:         end if
16:         Add the Remaining first points in Neighbours to Result
17:         Dist++
18:     end while
19: end function
```

3.4 Storage

The algorithm to store a document used in pARagraph is very similar to the one used by LLAH, and can be seen in Algorithm 3.3. Firstly, centroids are extracted from the input image and then all hash values are computed from them. This second step uses the Grid created previously to accelerate the many nearest neighbour searches that are required, and can be parallelized. Therefore, we share the work in this step between 4 threads, each computing the hash values of a quarter of the feature points; however, this optimization is omitted in Algorithm 3.3 for simplicity. Next, these values are then grouped by their hash table's page number and stored together with a pointer to the corresponding centroid. Finally, the data is stored in the hash table with the groups being processed one at a time, thus avoiding page changes.

In the code shown, the choice for lists instead of arrays in lines 3 and 4 is so that old entries can be deleted (lines 11 and 18) whenever data is moved around (lines 9 and 17), which keeps the memory usage constant. This seemingly unnecessary copying is done so that the step

in line 3, which is a processing bottleneck, can be processed completely in parallel. If the tuples had been grouped as soon as they were computed, all threads would be accessing the same memory and synchronization techniques would be necessary to avoid errors.

Algorithm 3.3 Storage algorithm.

Input: Image *Img* in grayscale; document ID *DocId* of the new document
Output: No output

```
1:  function STOREDOCUMENT(Img, DocId)
2:      Centroids, G ← computeCentroids(Img)
3:      HashValues ← list of the same size as Centroids, in which the element in the i^th position
        is an array containing the hash values of the i^th centroid
4:      PageHashValues ← array with size equal to the number of pages in the hash table. Each
        element is an empty list
5:      for each point P in Centroids do
6:          Ptr ← pointer pointing to P
7:          for each Key in HashValues.front() do
8:              PageNumber ← page number of Key in the hash table
9:              PageHashValues[PageNumber].append((Key, Ptr))
10:         end for
11:         HashValues.popFront()
12:     end for
13:     for each list GroupedHashValues in PageHashValues do
14:         while GroupedHashValues.size() > 0 do
15:             Key, Ptr ← GroupedHashValues.front()
16:             P ← point pointed to by Ptr
17:             Insert (DocId, P) in the hash table using key Key
18:             GroupedHashValues.popFront()
19:         end while
20:     end for
21: end function
```

3.5 Retrieval

The retrieval algorithm used in pARagraph (Algorithm 3.4) has some key differences from the one used in LLAH, primarily the computation of the homography and its use to validate the retrieved document. The first steps of processing are the same as in the storage algorithm: the centroids and grid are computed, all hash values are found (with the use of 4 threads) and these values are grouped by their hash table's page number. Next, the voting is done, during which only the votes that satisfy Conditions 2 and 3 in Section 2.1.5 are cast. Also, all the point correspondences between the query points and the original database points in the hash table are stored during this step; they will be used to compute the homography. Then, the most voted document is found, and its number of votes is checked to be greater than 10. This value was chosen in order to provide some minimum robustness to noise (erroneous votes) to the

homography computation, which requires at least 5 votes. Finally, the point correspondences pertaining to other documents are removed, and the homography is computed.

3.6 Homography Computation

The computation of the homography from the original database image to the query image is the main difference between the retrieval algorithms used in pARagraph and LLAH, and it is shown in Algorithm 3.5. The systems proposed by Dang et al. in [11] and Takeda et al. in [4][10] also perform this computation, but we execute a novel filtering of erroneous correspondences and a different validation of the computed homography.

The first step of the processing is to remove false matches in the point correspondences, which are the result of erroneous votes. This is necessary because when the number of correspondences is small, the homography that is found is sometimes incorrect, even when the correct document is retrieved. The reason for this is the presence of mismatches in the correspondences, i.e., noise, which occur due to erroneous votes. This is not a problem when there is a reasonable amount of data, because RANSAC is robust to noise, but for a small number of correspondences (e.g., between 10 and 15), noise can be significant.

In order to filter existing noise, we use a modification of the technique described in [17]. In their work, Zhao et al. propose the use of a Delaunay triangulation computed with the feature points in the correct image, which is then used to create a graph with the corresponding query points. This second graph is built by inserting an edge joining two query points if and only if their corresponding original points are also joined by an edge in the triangulation. This graph is then used to find possible mismatches, because its topology should not change with the projective distortion, i.e., it should remain planar. Therefore, all vertices whose majority of edges contain edge crossings are considered mismatches and removed from the graph. This approach does not work well for our intended filtering, however, because a mismatched query point is usually very far from its intended position, which means that its edges cross many others. Overall, this causes most, if not all, of the graph's edges to have at least one crossing (Figure 3.5).

To solve this problem, we use a greedy heuristic. Let the crossing number of an edge be the number of other edges it crosses, and the crossing number of a vertex be the sum of its incident edges' crossing numbers. We find the vertex with the greatest crossing number in the graph and remove it, as well as its incident edges. We then repeat this process for as long as there are edge crossings (Figure 3.6). Empirically, we found 0 to be a good value for the maximum allowed vertex crossing number (which applies the most conservative filtering) but this could be increased depending on the application. Greater values would allow more correspondences to pass the filtering step, but some of them could be noise. However, since RANSAC is fairly robust to noise, this might not be a problem. This increase of the acceptable crossing number could be beneficial in the later validation step, helping decrease the occurrences of false negatives in the retrieval, but it could also allow the returning of incorrect homographies, which fit the noise

Algorithm 3.4 Retrieval algorithm.

Input: Image *Img* in grayscale; ratio *MinClosePoints* of correctly mapped centroids by the computed homography

Output: Boolean indicating success or failure; document ID *DocId* of the retrieved document; homography *H*

function RETRIEVEDOCUMENT(*Img*, *MinClosePoints*)
  $Centroids, G \leftarrow computeCentroids(Img)$
  $HashValues \leftarrow$ list of the same size as Centroids, in which the element in the i^{th} position is an array containing the hash values of the i^{th} centroid
  $PageHashValues \leftarrow$ array with size equal to the number of pages in the hash table. Each element is an empty list
  for each point P **in** $Centroids$ **do**
    $Ptr \leftarrow$ pointer pointing to P
    for each Key **in** $HashValues.front()$ **do**
      $PageNumber \leftarrow$ page number of Key in the hash table
      $PageHashValues[PageNumber].append((Key, Ptr))$
    end for
    $HashValues.popFront()$
  end for
  $VotingMap \leftarrow$ empty mapping
  $PointCorrespondences \leftarrow$ empty list of point tuples
  for each list $GroupedHashValues$ **in** $PageHashValues$ **do**
    while $GroupedHashValues.size() > 0$ **do**
      $Key, Ptr \leftarrow GroupedHashValues.front()$
      if there is an entry in the hash table with key Key **then**
        $(DocId, OriginalPoint) \leftarrow$ entry with key Key
        if conditions to prevent erroneous votes are satisfied **then**
          $VotingMap[DocId]$++
          $QueryPoint \leftarrow$ point pointed to by Ptr
          Append $(DocId, QueryPoint, OriginalPoint)$ to $PointCorrespondences$
        end if
      end if
      $GroupedHashValues.popFront()$
    end while
  end for
  $(RetrievedDocId, NumVotes) \leftarrow$ entry in $VotingMap$ with the most votes
  if there were no votes **or** $NumVotes < 10$ **then**
    Return failure
  end if
  Remove the entries in $PointCorrespondences$ in which the $DocId$ is not $RetrievedDocId$
  $(ValidH, H) \leftarrow computeHomography(PointCorrespondences, MinClosePoints)$
  if not $ValidH$ **then**
    Return failure
  end if
  Return $true$, $RetrievedDocId$ and H
end function

Algorithm 3.5 Algorithm to compute the homography.

Input: list *PointCorrespondences* containing triples of a document ID and two points; ratio *MinClosePoints* of correctly mapped centroids by the computed homography
Output: Boolean indicating success or failure; homography H

1: **function** COMPUTEHOMOGRAPHY(*PointCorrespondences*, *MinClosePoints*)
2: *DelaunayQuery* ← Delaunay triangulation of the original points in *PointCorrespondences*
3: *DelaunayQuery* ← graph with no edges in which the vertices are the query points in *PointCorrespondences*
4: **for each** edge E **in** *DelaunayOriginal* **do**
5: Insert an edge in *DelaunayQuery* connecting the two query points whose original points are the two vertices of E
6: **end for**
7: $V \leftarrow$ vertex from *DelaunayQuery* whose edges have the greatest sum of crossings
8: **while** $V.sumCrossings() > 0$ **do**
9: Remove V from the graph
10: $V \leftarrow$ vertex from *DelaunayQuery* whose edges have the greatest sum of crossings
11: **end while**
12: Remove from *PointCorrespondences* the entries whose query points were removed from *DelaunayQuery*
13: Compute the homography H from the original points in *PointCorrespondences* to their query points, utilizing RANSAC
14: **if** no homography was found **then**
15: Return failure
16: **end if**
17: *ClosePoints* ← ratio of entries in *PointCorrespondences* in which the original point, when transformed by H, is within a Manhattan distance of 10 from the query point
18: **if** $ClosePoints < MinClosePoints$ **then**
19: Return failure
20: **end if**
21: Return *true* and H
22: **end function**

instead of the correct matches.

In this noise filtering step, we utilize OpenCV for the computation of the Delaunay triangulation, and we store the graphs using our implementations of `Graph`, `Edge` and `Vertex` data structures. These were created to perform efficiently the greedy heuristic that was described, through the storage in each `Edge` object of a set of pointers to all edges it crosses, which allows us to efficiently compute and update crossing numbers after a vertex removal.

After the noise filtering step, the correspondences considered mismatches are removed and then the homography from the original points to the query points is computed using OpenCV's implementation of RANSAC. Afterward, the validation step is performed, which is intended to confirm that the computed transformation is satisfactory. For this to be true, the original points, when transformed by the homography, must be close to the query points. Therefore, the ratio of point correspondences for which this is true can be used to validate the transformation. For that reason, the validation performed in the algorithm consists in computing this ratio and then checking it to be greater than a minimum value passed as a parameter. Empirically, we found 0.6 to be a good number for this parameter, and this is used as its default value in pARagraph. Finally, we consider the points to be close if their Manhattan distance [18] is less than 10.

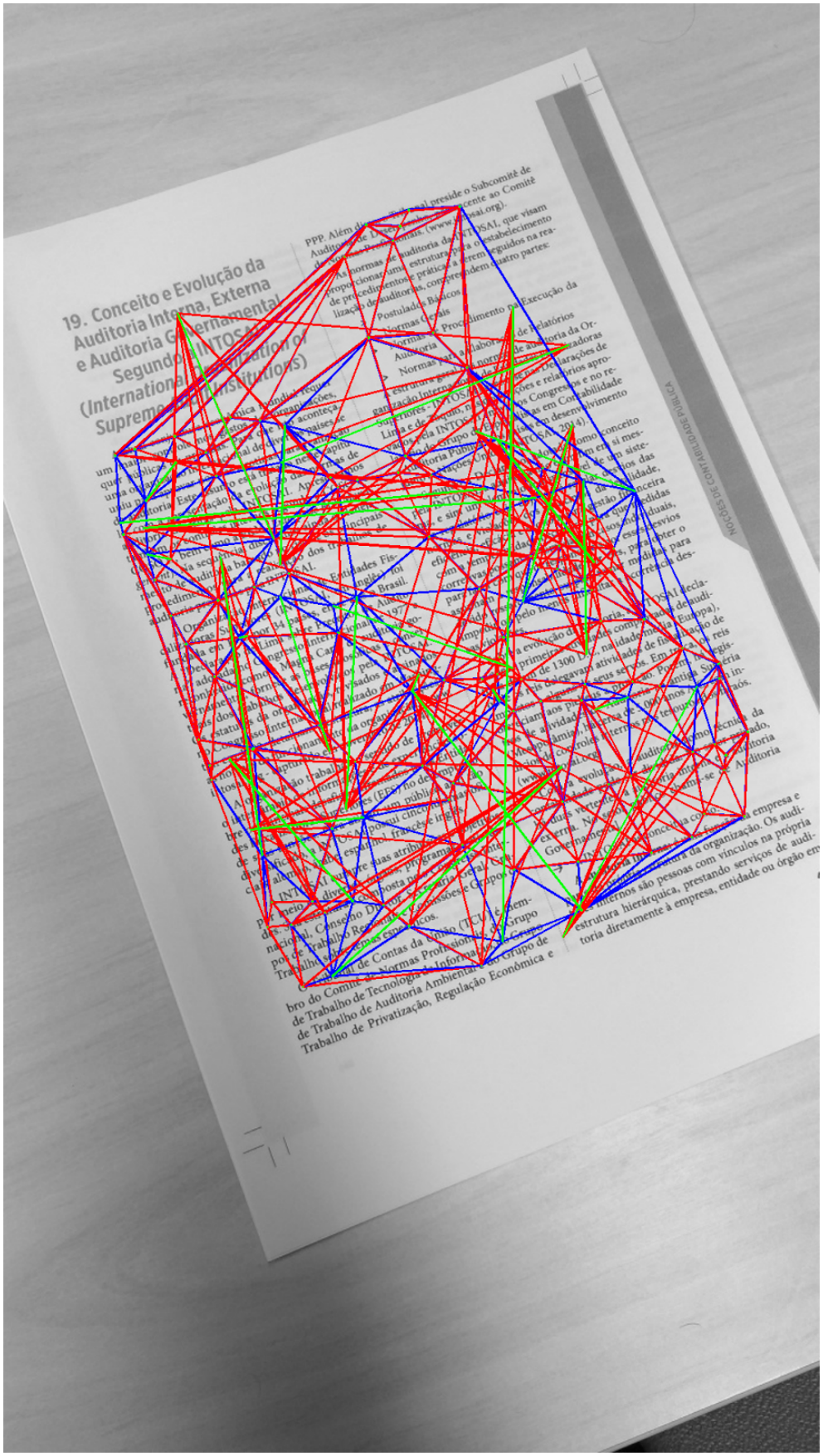

Figure 3.5: Delaunay triangulation of the original points (in blue) generates a graph from the query points (in red). The original points have been transformed by the correct homography to be shown in the image. Because the displacement between corresponding query and original points (in green) is very large for mismatched points, almost all edges in the red graph have at least one edge crossing.

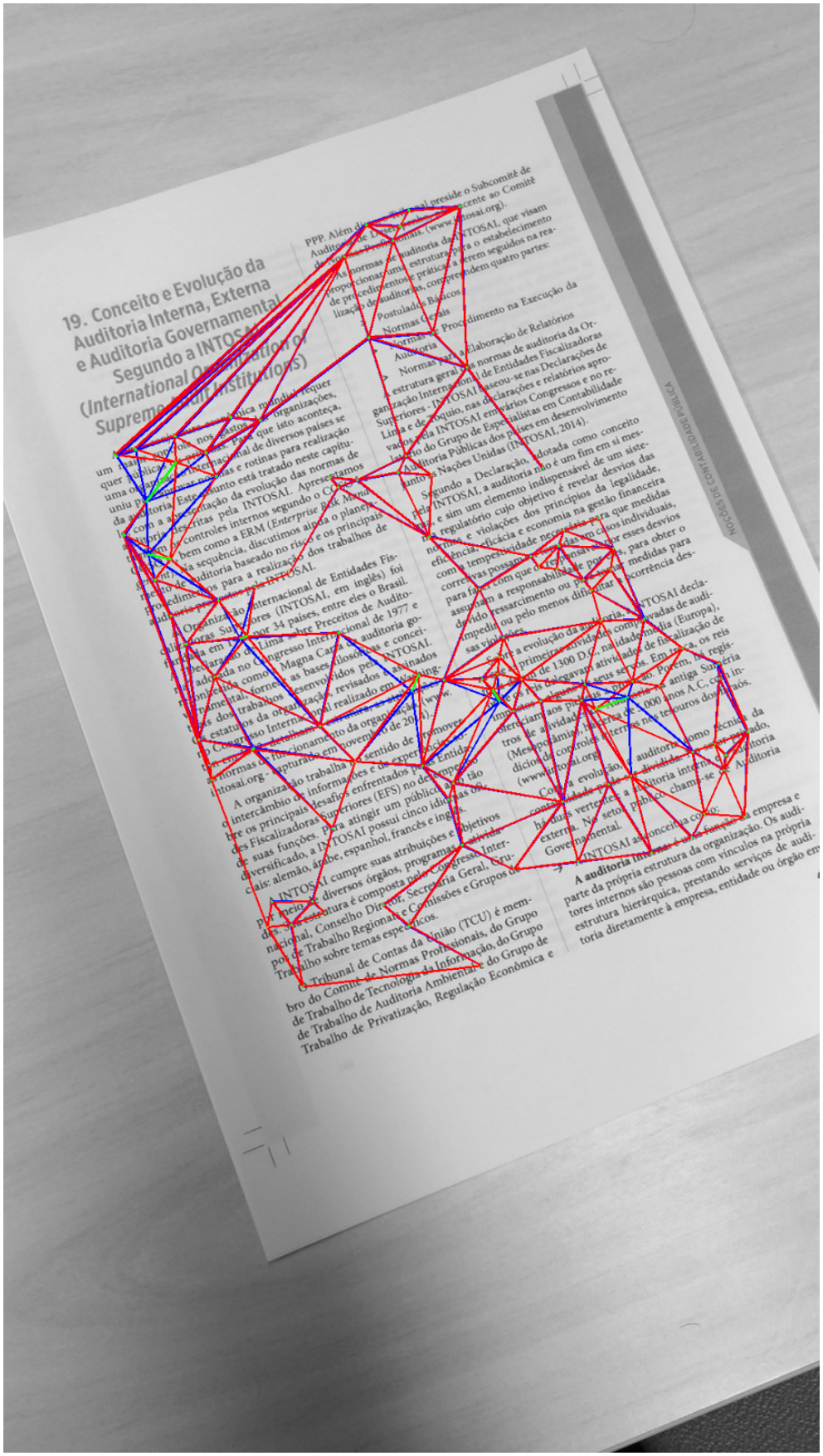

Figure 3.6: Delaunay triangulation of the original points (in blue) is used to generate a graph from the corresponding query points, which is shown (in red) after filtering. The original points have been transformed by the correct homography to be shown in the image, and the displacement from them to the corresponding query points is shown in green.

4

Experiments and Results

In order to evaluate pARagraph's performance, we constructed a database and established several testing cases. The selected database was comprised of 389 images, which are the pages of the full and short papers published in the proceedings of the IX Symposium on Virtual and Augmented Reality (SVR 2007). We used PDF files of the papers to acquire the document images, each page being a PNG file with 1785×2523 pixels resolution (Figure 4.1).

The test images were taken with an ASUS Zenfone 3 smartphone from pages of a physical copy of the proceedings, and each photograph was stored as a JPEG file with 3492×4656 pixels resolution.

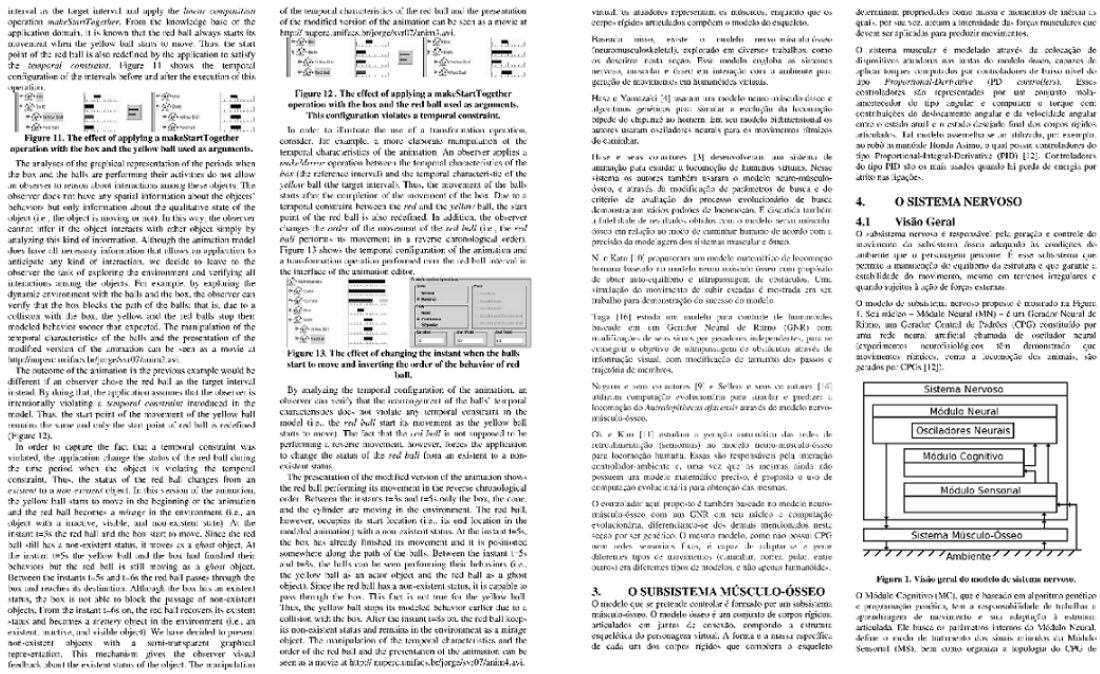

Figure 4.1: Two sample pages of the database used in the experiments.

4.1 Primary Tests

We constructed three primary experiments to evaluate the library, each using images with different levels of distortion, and performed them using 4 parameter configurations:

1. $n = 7$, $m = 6$ and ratio of triangle areas invariant.
2. $n = 8$, $m = 7$ and ratio of triangle areas invariant.
3. $n = 8$, $m = 7$ and cross ratio invariant.
4. $n = 6$, $m = 5$ and SRIF invariant.

Of the chosen configurations, the first three were chosen because they presented good results in [1], and the fourth was chosen because it was the fastest shown in [3].

In all cases, the q parameter was set to 25; the hash table had size $H_{size} = 1.28 \cdot 10^8$ and only 1 page; and the threshold values were computed in a preprocessing step, with the exception of the fourth configuration, in which values were discretized according to Equation 2.5. Because implementing an application for mobile devices which uses the library is a future work, the experiments were performed in a computer with an Intel Core i7-4700HQ 2.40 GHz processor and 12 GB of memory.

The metrics used to measure the library's performance were: accuracy of retrieval, time to store a document in the database, time to retrieve a document and peak memory consumption. Of those, the storage and retrieval time were measured as the average of all such operations performed during a given experiment. Also, in order to obtain more reliable results, each experiment was done three times, and the time and memory metrics were averaged among all executions, resulting in the values presented in this chapter.

The first experiment performed was with "well-behaved" queries, i.e., images in which there are approximately no similar, affine or projective distortion, and roughly entirely occupied by the document (Figure 4.2). Following those restrictions, we randomly selected 50 pages from the book, which were then presented to the algorithm. The results, which can be seen in Table 4.1, show that all four configurations had a satisfactory performance, with the best accuracy being achieved while using the SRIF invariant.

The second experiment was performed with images obtained after rotating the camera by 45, 90, 135 and 180 degrees (Figure 4.3). We initially selected 10 images randomly from among the 50 chosen for the first experiment. Then, we photographed all of them with the camera having been rotated by each of the 4 aforementioned angles; therefore, a total of 40 queries were searched. The results, presented in Table 4.2, show that all four configurations displayed once again very satisfactory results, with some reaching 100% accuracy.

The third experiment was performed with images obtained with the camera in poses that would generate projective distortions. Initially, we chose 4 arbitrary poses that were capable of

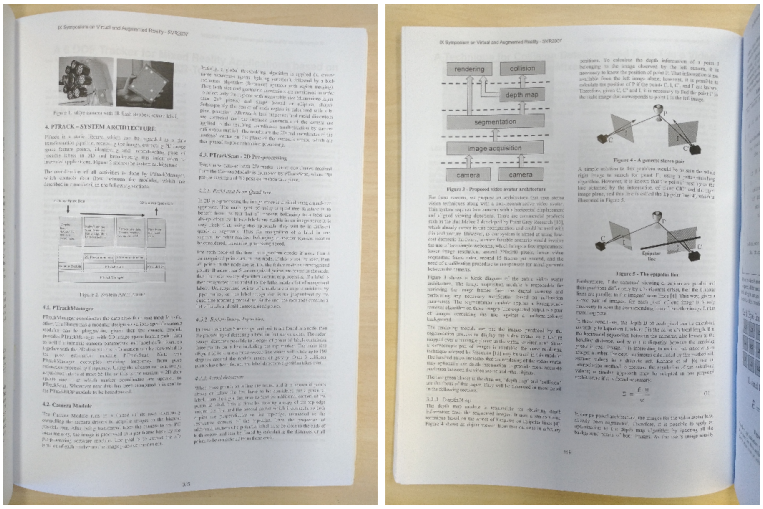

Figure 4.2: Examples of pages used in the first experiment. The images are "well-behaved", having approximately no similar, affine or projective distortions.

Table 4.1: Results of the first experiment, performed with "well-behaved" images.

n	*m*	**Invariant**	**Accuracy**	**Avg. Storage Time**	**Avg. Retrieval Time**	**Peak Memory Consumption**
7	6	ratio of triangle areas	96%	62.86 ms	65.39 ms	162.55 MB
8	7	ratio of triangle areas	96%	90.63 ms	96.81 ms	288.84 MB
8	7	cross ratio	92%	81.62 ms	84.73 ms	201.30 MB
6	5	SRIF invariant	98%	54.79 ms	55.92 ms	244.14 MB

Table 4.2: Results of the second experiment, performed with rotated images.

n	*m*	**Invariant**	**Accuracy**	**Avg. Storage Time**	**Avg. Retrieval Time**	**Peak Memory Consumption**
7	6	ratio of triangle areas	92.5%	66.41 ms	73.95 ms	170.76 MB
8	7	ratio of triangle areas	100%	90.26 ms	101.97 ms	296.98 MB
8	7	cross ratio	92.5%	80.03 ms	94.15 ms	209.44 MB
6	5	SRIF invariant	100%	54.88 ms	61.23 ms	251.74 MB

generating this level of distortion (Figure 4.4). Then, we photographed the same 10 documents chosen for the second experiment while the camera was being held in each of the 4 poses. Therefore, a total of 40 queries were presented to the library; the results are shown in Table 4.3. Again, the configurations had satisfactory performance, with the exception of the one using the SRIF invariant. This was expected, however, as the SRIF descriptor is not invariant to projective distortion, but only to rotations and changes of scale.

The experiments show that pARagraph's accuracy is very high, with the lowest among all configurations being 92% in the first two experiments. In the third experiment, the lowest

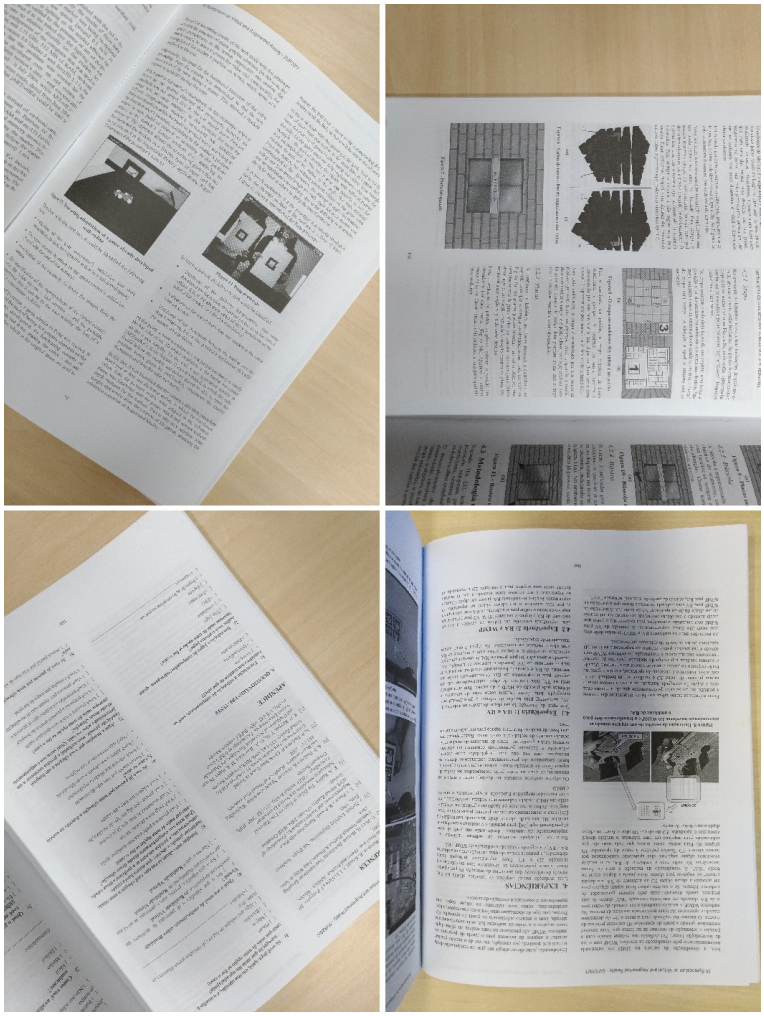

Figure 4.3: Examples of pages used in the second experiment. The images show the four rotations in which photographs were taken for this experiment; they are roughly, from left to right and top to bottom: 45, 90, 135 and 180 degrees.

Table 4.3: Results of the third experiment, performed with projectively distorted images.

n	m	**Invariant**	**Accuracy**	**Avg. Storage Time**	**Avg. Retrieval Time**	**Peak Memory Consumption**
7	6	ratio of triangle areas	92.5%	63.88 ms	72.45 ms	162.75 MB
8	7	ratio of triangle areas	95%	87.64 ms	115.92 ms	289.33 MB
8	7	cross ratio	95%	84.29 ms	96.82 ms	201.54 MB
6	5	SRIF invariant	80%	56.09 ms	59.95 ms	243.9 MB

accuracy was 80%, which was expected due to the SRIF invariant, but this value was 92.5% among the remaining configurations. It is important to point out that in the second and third experiments, some accuracies were higher than in the first, even though the images had distortions. This is because the former experiments used only 10 pages from the 50 used in the latter, and some of the difficult cases were not present in this subset, such as pages with very little text or

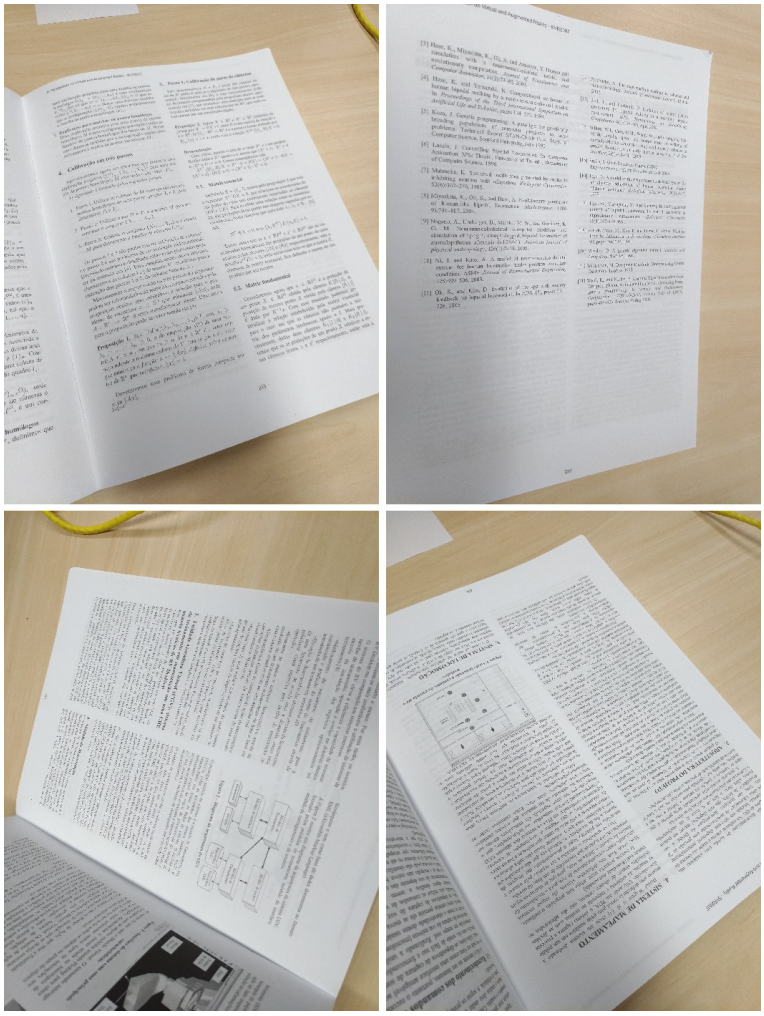

Figure 4.4: Examples of pages used in the third experiment. The images shown demonstrate the four poses in which photographs were taken for this experiment.

many images.

Storage and retrieval time are also satisfactory, and most configurations are able to perform the latter in less than 100 ms, which allows retrievals to be performed at roughly 10 fps. Concerning memory consumption, the required amounts are well within the limits of mobile devices, and the virtualization of the hash table reduced it considerably. If a vector had been used to store the hash table, it would have required $1.28 \cdot 10^8 \cdot 24 = 2.86$ GB of memory, where the 24 bytes occupied by each entry are due to the document ID and the point coordinates being stored in 8-byte integer and double-precision floating-point variables, respectively. However, the consumption could still be substantially reduced with little loss of accuracy by changing these to 4-byte integer and single-precision floating-point variables; due to time restrictions, we leave this optimization to a future work.

Comparing the four configurations, we see that the first and third presented very similar accuracy overall, but the former uses less memory and executes faster. This is due to the cross ratio needing 5 feature points, while the ratio of triangle areas needs only 4. This requires the choice of greater values for both the n and m parameters, which increases the number of hash

values computed from each document (i.e., $C_m^n \cdot C_f^m$), and therefore the execution time. Also, the use of the cross ratio invariant causes the descriptors to be vectors of length 21, as opposed to 15 when using the ratio of triangle areas. This is the reason for the greater memory consumption, as longer descriptors are more unlikely to be equal, which reduces the number of collisions in the hash table. Higher values of $C_m^n \cdot C_f^m$ and greater descriptor length are also the reason why the second configuration requires more time and memory than all others, as well as why it has the highest overall accuracy. This configuration has the longest descriptors, which grants them greater discrimination power, not only reducing the number of collisions in the table, but also helping improve accuracy. As for the fourth configuration, it presented very good accuracy in the first two experiments, but it was noticeably worse than the others on the third experiment. This was expected, however, due to the invariant used not being able to handle projective distortions, and the 80% accuracy was surprisingly good. We believe that the high memory usage is due to the occurrence of few collisions in the hash table, which would also explain why the accuracy was so high in the first two experiments, even though the descriptor size is smallest when this configuration is used.

Overall, we believe that the first and second configurations are the best ones, and there is a trade-off between them: the former executes faster and using less memory, while the latter is more accurate. The third configuration is about as accurate as the first, but also slower and more memory-consuming, so it is not as viable. The fourth configuration, however, might be the best for applications in which little or no projective distortion is expected. While it requires more memory than the first and third configurations, it is also the fastest and had particularly good accuracy on the first two experiments.

4.2 Stress Tests

In order to evaluate the limits of the library's retrieval accuracy, we performed two stress tests; the queries used can be seen in Figure 4.5, and the results obtained are shown in Table 4.4.

The first test was intended to evaluate the retrieval with images affected by severe projective distortion. Photographs (Figure 4.5, *a*-*e*) were taken of the first five documents from the 10 used in the second and third experiments performed previously, and the 5 images were then presented to pARagraph while using each configuration. The fourth of those failed to recognize almost all queries, as expected because of its use of the SRIF invariant, but the remaining three had no problems with most of them. The second configuration retrieved all documents, while the first and third only failed to retrieve the query Figure 4.5 *a*.

The second test was made to evaluate the retrieval in the presence of severe occlusion. The 10 images used in the previous experiments were photographed, while the documents were occluded in different ways or zoomed in (Figure 4.5, *f*-*o*). Among the queries, Figure 4.5, *k*-*l* were not recognized with any of the configurations, and those were the only two images the second and fourth configurations failed to retrieve. The first and third had worse results, with

Table 4.4: Results of the four configurations on the stress tests.

n	m	**Invariant**	**Accuracy in Test 1**	**Accuracy in Test 2**
7	6	ratio of triangle areas	4 / 5	6 / 10
8	7	ratio of triangle areas	5 / 5	8 / 10
8	7	cross ratio	4 / 5	7 / 10
6	5	SRIF invariant	1 / 5	8 / 10

both of them failing to retrieve Figure 4.5 *g* besides the two, and the former also not recognizing Figure 4.5 *m*. Still, all of them could retrieve more than half of the queries, which shows significant robustness.

Overall, the implementation proved to be very robust to projective distortions, due to the first test posing no problems to most configurations, and capable of retrieving documents even when a large portion of the page is occluded. Particularly, in the case of zoomed in images, a few paragraphs were enough to successfully retrieve the document, which is especially important due to the fact that zooming in on the text is a good way to compensate for low resolutions.

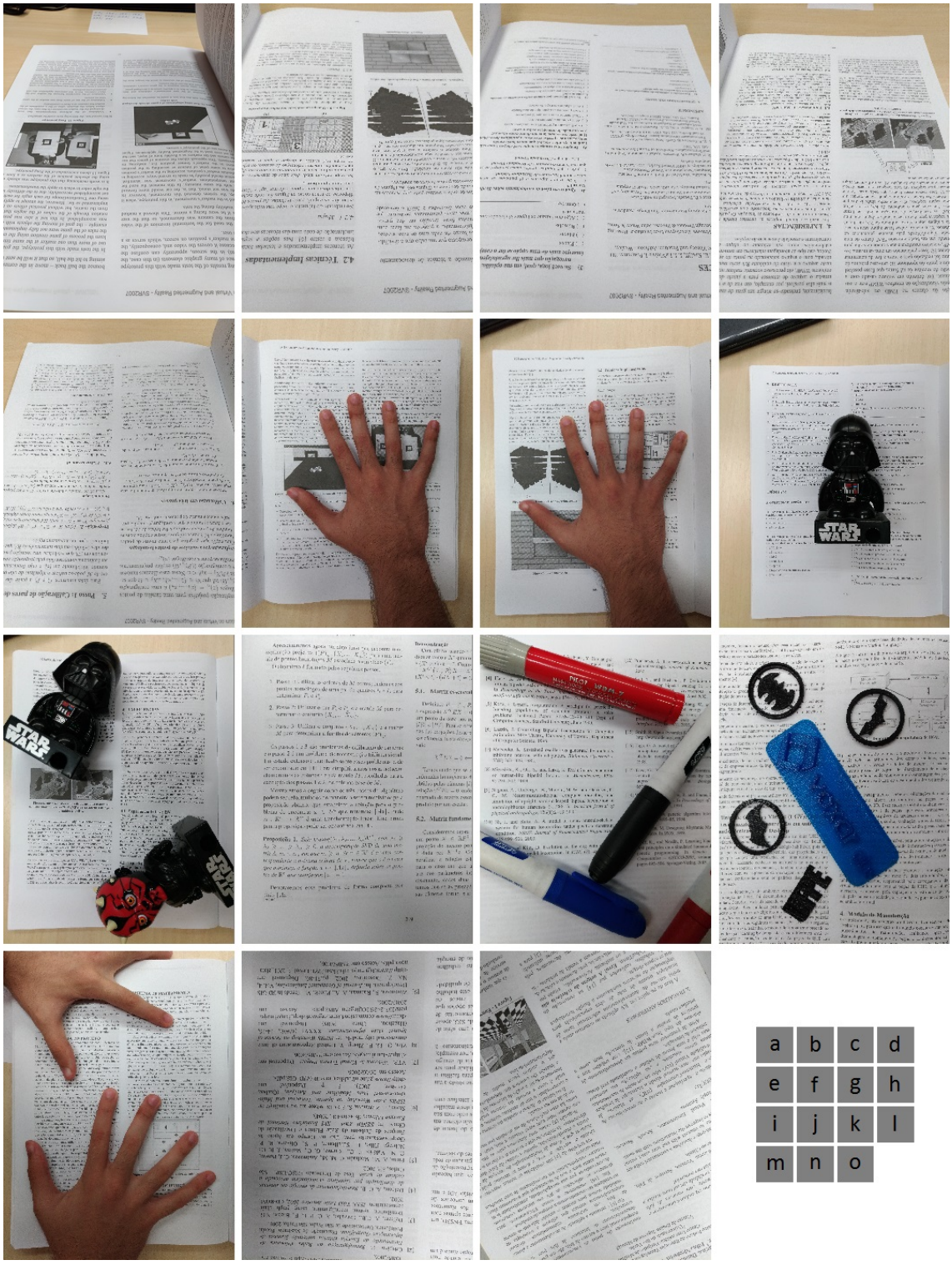

Figure 4.5: Photographs used in the stress tests. Queries *a-e* were used in the first, while *f-o* were used in the second.

5

Conclusion

In this work we have investigated the Camera-Based Document Image Retrieval problem and implemented the pARagraph library, a C++ implementation of the LLAH technique. In order to make pARagraph viable for use in mobile devices, which have limited hardware, improvements for the run time and memory consumption were necessary. Therefore, we have adopted two modifications that were proposed for LLAH in the past, which significantly decrease both retrieval time and amount of memory necessary for the hash table. Furthermore, we have implemented several other optimizations for execution time (parallelism), memory consumption (paging scheme, hash table virtualization) and accuracy of retrieval (noise filtering with Delaunay triangulation).

We evaluated the library's performance through three experiments with increasing levels of distortion in the input images. The results were very satisfactory, with all parameter configurations showing very high retrieval accuracy and executing at roughly 10 fps. Also, two configurations produced the best results, which offers some versatility for applications: one is faster and uses less memory, which makes it best for speed-oriented or memory-oriented applications; the other is more accurate, which makes it best for precision-oriented applications.

Finally, future works include: the adaptation of the pipeline described in [4], which could greatly increase the number of frames per second; the development of applications for mobile devices that use the library to perform Augmented Reality; the adoption of the strategy used in [3][11] of analyzing the t most voted documents; and the reduction in precision of the data stored in the hash table, in order to further diminish memory consumption.

References

[1] Tomohiro Nakai, Koichi Kise, and Masakazu Iwamura. Use of affine invariants in locally likely arrangement hashing for camera-based document image retrieval. *Document Analysis Systems VII*, pages 541–552, 2006.

[2] Tomohiro Nakai, Koichi Kise, and Masakazu Iwamura. Camera based document image retrieval with more time and memory efficient LLAH. *Proceedings of the Second International Workshop on Camera-Based Document Analysis and Recognition (CBDAR)*, pages 21–28, 2007.

[3] Quoc Bao Dang, Muhammad Muzzamil Luqman, Mickaël Coustaty, CD Tran, and Jean-Marc Ogier. SRIF: Scale and rotation invariant features for camera-based document image retrieval. In *13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 601–605. IEEE, 2015.

[4] Kazutaka Takeda, Koichi Kise, and Masakazu Iwamura. Real-time document image retrieval on a smartphone. In *10th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 225–229. IEEE, 2012.

[5] Neil Shah. Changing trends in camera & display resolutions in smartphones, 2016. `http://www.counterpointresearch.com/pulse-premium-july-2016/`.

[6] Hideaki Uchiyama and Hideo Saito. Random dot markers. In *Virtual Reality Conference (VR), 2011 IEEE*, pages 35–38. IEEE, 2011.

[7] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997.

[8] Tomohiro Nakai, Koichi Kise, and Masakazu Iwamura. Hashing with local combinations of feature points and its application to camera-based document image retrieval. *Proceedings of the 1st International Workshop on Camera-Based Document Analysis and Recognition (CBDAR)*, pages 87–94, 2005.

[9] Kazutaka Takeda, Koichi Kise, and Masakazu Iwamura. Real-time document image retrieval for a 10 million pages database with a memory efficient and stability improved LLAH. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1054–1058. IEEE, 2011.

[10] Kazutaka Takeda, Koichi Kise, and Masakazu Iwamura. Memory reduction for real-time document image retrieval with a 20 million pages database. *Proceedings of the Fourth International Workshop on Camera-Based Document Analysis and Recognition (CBDAR)*, 1, 2011.

[11] Quoc Bao Dang, Viet Phuong Le, Muhammad Muzzamil Luqman, Mickaël Coustaty, CD Tran, and Jean-Marc Ogier. Camera-based document image retrieval system using local features-comparing SRIF with LLAH, SIFT, SURF and ORB. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 1211–1215. IEEE, 2015.

[12] Tomohiro Nakai, Koichi Kise, and Masakazu Iwamura. Document/image searching method and program, and document/image recording and searching device. World Intellectual Property Organization, 2006. Publication number WO/2006/092957.

[13] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[14] Gary Bradski. The OpenCV library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11):120–123, 2000.

[15] Hanan Samet. *An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures*, pages 51–68. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[16] John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. *Computer graphics: principles and practice (3rd ed.)*. Addison-Wesley Professional, Boston, MA, USA, July 2013.

[17] Xinyue Zhao, Zaixing He, and Shuyou Zhang. Improved keypoint descriptors based on Delaunay triangulation for image matching. *Optik-International Journal for Light and Electron Optics*, 125(13):3121–3123, 2014.

[18] Paul E Black. Manhattan distance. in *Dictionary of Algorithms and Data Structures* [online], Vreda Pieterse and Paul E. Black eds., 31 May.