Multi-View Vocabulary Trees for Mobile 3D Visual Search

DAVID EBRÍ MARS

KTH Electrical Engineering

Master's Degree Project
Stockholm, Sweden October 2014

XR-EE-KT 2014:013

Multi-View Vocabulary Trees for Mobile 3D Visual Search

David Ebrí Mars

Abstract

Mobile Visual Search (MVS) is a research field which focuses on the recognition of real-world objects by using mobile devices such as smart phones or robots. Current mobile visual search solutions achieve search results based on the appearance of the objects in images captured by mobile devices. It is suitable for planar structured objects such as CD cover images, magazines and art works. However, these solutions fail if different real objects appear similar in the captured images. To solve this problem, the novel solution captures not only the visual appearance of the query object, but uses also the underlying 3D geometry.

Vocabulary Tree (VT) methods have been widely used to efficiently find the match for a query in the database with a large volume of data. In this thesis, we study the vocabulary tree in the scenario of multi-view imagery for mobile visual search. We use hierarchically structured multi-view features to construct a multi-view vocabulary trees which represent the 3D geometric information of the objects. Relevant aspects of vocabulary trees such as the shaping of trees, tf-idf weighting and scoring functions have been studied and incorporated in the multi-view scenario. The experimental results show that our multi-view vocabulary trees improve the matching and ranking performance of mobile visual search.

Acknowledgements

This thesis has been developed at the Visual Computing and Communications group at the School of Electrical Engineering at KTH. It would never have been possible without the priceless help and dedication of my always comprehensive supervisor, Haopeng Li, who has guided me through the difficulties that I encountered during the thesis and offered me constructive advice.

Furthermore I would like to thank my coordinator Markus Flierl for introducing me to the topic and giving me the chance to participate in this exciting project. Also, I want to thank to the people at the department for their fellowship, especially to my project mate Hanwei Wu.

Last but not least, I would like to thank my family and friends, who have supported me throughout the entire process, particularly to my parents, even being 3000 km away from Stockholm. My heartfelt thanks.

Contents

List of Figures

Chapter 1

Introduction

1.1 What is Mobile Visual Search?

In the past decades, technology has been developed at high speed. The current computational capacity at our disposal was just a dream some time ago. An important field of the technological revolution in recent years has been mobile devices. They have moved from being big and heavy "bricks" with poor black and white screens to being powerful computers with big HD screens, high-resolution cameras and fast wireless connections. Furthermore, mobile phones have spread all around the world. The International Telecommunication Union estimated in May 2014 that there are nearly 7 billion mobile subscriptions worldwide, which represents a 95.5 percent of the world population[1]. All these properties make smartphones ideal devices to perform image processing algorithms and bring new possibilities to developers to create new uses for mobile phones. Additionally, considering the large number of users, technology for mobile devices can be a very profitable business.

A very interesting field that is emerging nowadays is object retrieval, which tries to identify objects from the real world in a similar way as humans do by applying techniques from computer vision and artificial intelligence. The new research area called mobile visual search (MVS) ([1],[2]) studies this field in the specific context of mobile devices. These systems try to recognise objects from images or video sequences taken by robots, smartphones, compact cameras, etc. The particularity of doing it from mobile devices adds some extra problems to the old ones. Some of these matters that need to be solved and optimized are client-server interaction, bad quality cameras, limited computational capacity and poor mobile connections. Thanks to advanced methods in signal processing and communication theory, MVS is facing these challenges with good results and promising perspectives.

[1] Link: `www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2014-e.pdf`

1.2 Applications

The great potential of mobile visual search relies on the fact that it doesn't need any additional element to perform the recognition than the objects themselves. In contrast to other systems like QR codes or barcodes, it recognises the objects directly from reality. The idea of identifying objects from the real world using a camera can be used in many different applications:

- Shopping. This is one of the most popular applications for mobile visual search. The idea is to take a picture of a product and recognise it at the moment, showing interesting information about it like description, price, shops where you can buy it, opinions of people that has already bought it, etc. These products can be DVDs, books, posters, computer games, clothes... There are already some commercial apps offering this service. One example is Flow Powered by Amazon[2], that can identify products from Amazon on-line shop and offer the possibility to buy them instantly.
- Games. This industry is very interested in developing good object recognizing algorithms, because the possibilities of playing with the reality in a virtual world are huge. There have been many approaches trough the years, from cameras that can identify the player's body (*Eyetoy* or *Kinect*), to smartphones that modify the world to create a battle scenario, like *Real Strike*[3].
- Geolocation. A complement to the GPS that uses the visual information of the mobile phone to locate the user with more accuracy.
- Tourism. The concept is to visit a city and discover it by taking pictures of buildings and monuments, so our smartphone can give us information about them, explain their history and curious facts.
- Robotics. Make robots to "understand" what they see. It can help to develop more sophisticated machines to be used innumerable fields: the military, the sanitary, humanitarian...

As we can see, the applications are countless. There are already some real applications available in the market that implement these ideas such as *Google Goggles*[4], *Camfind*[5] or *Layar*[6]. Although this technology is still in an early state, it is easy to find many different young companies and start-ups all around the world working on it, such as *Acrossair*[7] or *Mobile Acuity*[8]. There is no doubt that mobile visual search will have an important role in our future everyday lives.

[2]Flow: `www.amazon.com/A9-Innovations-LLC-Powered-Amazon/dp/B008G318PE`
[3]Real Strike: `https://itunes.apple.com/se/app/id507884100?mt=8`
[4]Google Goggles: `www.google.com/mobile/goggles`
[5]Camfind: `camfindapp.com`
[6]Layar: `www.layar.com`
[7]Acrossair: `www.acrossair.com`
[8]Mobile Acuity: `www.mobileacuity.com`

1.3 Motivation

Mobile visual search systems developed until now use just 2D image-based features. This means that the 3D geometric information of the objects is not taken into account. That's why current applications are restricted to recognise flat surfaces like CD covers, magazines, posters, books or artworks. Since the spatial structure of the objects is omitted, the performance of the current technology will be the same taking a photo of a real building and taking a photo of a picture of the same building. Our intentions are to use not only the visual appearance of the objects, but also their underlying 3D geometry [3]. This additional information will help us to improve the retrieval accuracy, because the system will be more robust against changes of perspective of the objects in the images.

To go beyond the 2D and know the geometry of an object it is imperative to have more than one image of it. We decided to use a multi-view approach, i.e., we use several images of the objects taken from different angles. With them it is possible to extract the spatial structure of the objects by applying an epipolar algorithm [3]. Also, from each of those images we extract visual information in the salient points and we create a database with them. These keypoints are called *feature descriptors*: small patches of the pictures with some local characteristics that are considered interesting and distinguishable from others. When we want to recognise an image, we use this database to compare the feature descriptors from the image taken by the client with the ones in the database, so we can identify to which object they belong.

One big problem in the retrieval algorithm is the huge data size. On the one side, we have the client which has taken two or more pictures of an object and extracted the descriptors. These will be denoted as query descriptors. On the other side, there is the server where we store a big database that can contain millions of objects with their respective descriptors. How do we compare the query descriptors with the database ones? Doing it one by one would be extremely slow and naive. A model adapted from text retrieval called *Bag of Words* (BoW) proposes to use vectors of occurrence counts of *visual words* for the recognition, similarly at how it is done in texts [4]. These visual words are contained in a visual vocabulary or codebook previously generated by clustering the image features. This solution works well with small databases, but it is infeasible when we have millions of descriptors. The clustering would be very complex and to compare descriptors one by one would be too slow.

To deal with this problem, David Nistér and Henrik Stewénius presented in 2006 a scalable and efficient way of organizing the data called Vocabulary Tree (VT) [5]. Basically, a VT is a hierarchical structure that vector quantizes the feature descriptors in a way that makes possible a fast and reliable recognition of the objects. It is constructed by applying recursively the clustering algorithm k-means on the dataset. Due to its efficiency and good performance, VT has been very popular and widely used in image retrieval since its publication. Our work in this Master Thesis has been to adapt the classic vocabulary tree to the particularities of our multi-view approach, evaluate its performance and extract conclusions from it.

1.4 Project Statements

The main tasks carried out in this Master Thesis have been:

1. Study carefully the previous literature related to mobile visual search putting emphasis on the vocabulary tree and the multi-view approach that has been developing the Communication Theory department of the Electrical Engineering School in KTH.
2. Implement the approach of single-view VT with the classical settings.
3. Apply the VT to the scenarios of multi-view imagery. Improve the performance of multi-view vocabulary trees.
4. Write the report detailing the theoretical framework, the work done, the results and the conclusions.

The implementation has been carried out using the tool MatLab. MatLab is a high-level language and interactive environment for numerical computation, visualization, and programming developed by MathWorks[9].

1.5 Outline

A little explanation of the content of each chapter of the thesis

- Chapter 2: Literature Review. In this section the fundamental background theory required to understand MVS is explained and reviewed. Also it is detailed the role that vocabulary tree performs in the system and why is it that important. Furthermore, our multi-view approach is introduced. All this section is supported by the publications taken on account in the thesis.
- Chapter 3: Multi-View Vocabulary Trees. The details of our implementation of the vocabulary tree and its adaptation to make it able to multi-view is explained here. Also some aspects and modifications that we have included in our work.
- Chapter 4: Experimental Results. We assess the performance of using multi-view vocabulary tree in mobile 3D visual search.
- Chapter 5: We summarize this thesis project.

[9]MathWorks: `www.mathworks.com`

Chapter 2

Literature Review

The vocabulary tree is a way of structuring the data used in image recognition systems to improve their efficiency. Thus, its role can only be understood when it is placed inside the whole system and the rest of the parts are well known. In this second chapter the fundamental theory required to get the whole picture of the MVS and understand the meaning of the VT will be introduced thoroughly.

2.1 Mobile Visual Search

The schedule followed by the recognition algorithm of MVS can be summarized in the following steps:

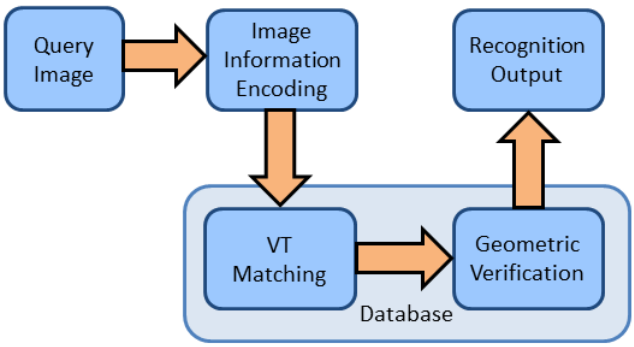

Figure 2.1: Mobile visual search generic scheme.

- Obtain the query image. This is the image that contains the object intended to be identified. It can be either a picture or a frame of a video where the object appears. In our case we will need more than one single image due to our multi-view approach.

- Detection and extraction of the features. There are many techniques employed for this purpose such as SIFT [6], SURF [7] or CHoG [8]. We will focus on SIFT due to its robustness against bad image conditions. Each interesting point of the image is codified in a 128 dimensional vector.
- Vocabulary Tree matching. This is the main point of our thesis. It is a crucial step to the right performance of the system. It handles to match the visual information of the query object with the correct one from the database as quickly as possible. Every descriptor passes through the VT and a candidates list is generated with the most probable objects to match the query in function of the output of the tree.
- Geometric verification. Each descriptor is extracted along with its location within the image, i.e., his 2D coordinates. This step checks that the spatial distribution of the descriptors assigned to an object are consistent. In this stage, the candidates list that we previously obtained will be corrected and the most probable object will be set as the output of the algorithm.

All these sub-systems will be explained in their respective sections below.

2.1.1 Client-Server Model

Because we are trying to create an image recognition system that works in mobile devices, there will be two main agents interacting: the client and the server. We need to decide a client-server structure to operate [2]. There are several options depending on whether the steps described in the previous section are executed by the server or the client. These are the different configurations:

1. The mobile device (client) only takes the pictures and sends them to the server, who analyzes them and runs all the algorithms. Only the capture of the images is executed in the client. The result of the computation will be sent back to the mobile device.
2. The client is the responsible of extracting the remarkable features from the images and sending them to the server. The server will use this information to recognise the object and, as in the option 1, send the results back to the client.
3. The mobile device contains the entire database and doesn't need to send anything to the server. All processes are performed locally in the client and the server doesn't take part on the recognition. In this case, the database can't be very large because mobile devices have short storage space. One implementation was carried out in [9].
4. Combinations of the options above. For example: the device contains a sub-set of the database with the most frequent objects identified. If the customer tries to recognise an object missing in the client, the mobile device can resort to the server.

For the thesis, we have chosen the second option. Current smartphones have powerful enough CPUs to run the algorithms that extract the information needed from the pictures by themselves. This choice is useful when the features transmitted to the server are lighter than the images. One of the biggest challenges in MVS systems is to fulfil the restrictions of the network. The amount of data sent has to be as small as possible to provide a good service even with poor conditions. A calibration step can be introduced, so the query data is sent depending on the quality of the network. If the connection is excellent, the client will send all the feature descriptors and the recognition will be very good. Otherwise, if the quality is bad, just the most important information will be sent and obviously the recognition will not be that robust. With this approach it is possible to adapt to the conditions of the network. For this reason, it is a good idea to use this scheme:

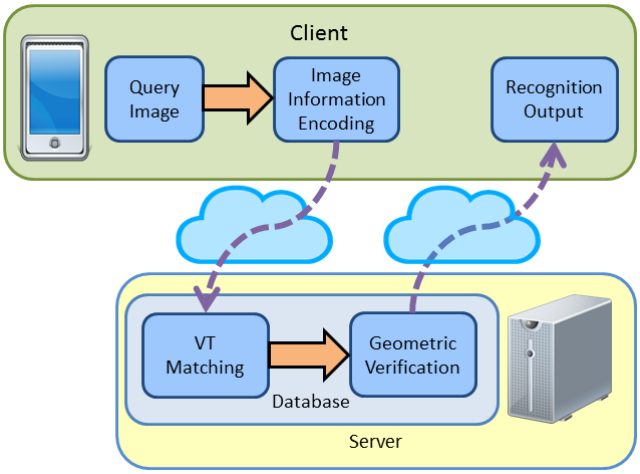

Figure 2.2: Diagram of the architecture of our MVS project

2.1.2 Image-based Feature Descriptors

In Chapter 1, we have introduced the feature descriptors. We could define a descriptor like a function that is applied to images or patches of images to describe their visual content and encode it in a way that allows us to perform comparisons with other images. The detection and encoding of salient characteristics in the images is a crucial problem. The descriptors have to fulfil a series of conditions as best as possible. They need to be:

- Precise: They need to be an accurate representation of the visual content. Also, we need to detect the same points in every image of the same object.
- Invariant to translation, rotation and scale changes.
- Invariant to presence of noise, blur, etc.
- Robust to occlusion and illumination change.

- Distinctive: The region should contain "interesting" information that allows us to differentiate among objects.
- Abundant: There should be enough points to represent the image.
- Compact: They need to be as light as possible in terms of bytes. Remember that we need to send them through the network.
- Time efficient: It can't take a long time to extract them.

There exists a trade off between the quality of a descriptor and its size. There are many approaches to face these problems and the literature is very wide. As it would take too long to explain deeply all the types of descriptors and feature detectors, we will focus on the most common strategy followed and review of the most popular algorithms. There are two main popular approaches that need to be explained: the global and the local descriptors.

2.1.2.1 Global and local descriptors

On the one hand there are the global descriptors. This approach conceives images as a whole element, without any kind of segmentation. It computes the color and texture features on the entire image. One example is FCTH (Fuzzy Color and Texture Histogram) [10], but probably the most popular global descriptor is GIST [11]. It is based on a very low dimensional representation of the scene. The authors propose a set of perceptual dimensions (naturalness, openness, roughness, expansion and ruggedness) that represent the dominant spatial structure of a scene. The model generates a multidimensional space in which scenes sharing membership in semantic categories are projected close.

On the other hand we have the local descriptors. This approach supports that images can be thought as a set of regions with interesting attributes which all together constitute the image. While a global descriptor encodes the whole picture, a local descriptor describes just a patch of the image. Thus, in order to represent an image by using local descriptors we will have a set of them, depending on how many interesting regions we have detected in the image. This conception is composed by two main steps: first, the detection of the *keypoints* of the image; and then, the feature extraction in that points. Some popular descriptors of this kind are the before mentioned SIFT (Scale Invariant Feature Transform) [6], SURF (Speeded Up Robust Features) [7], CHoG (Compressed Histogram of Graidents) [8] and also many others like FAST (Features from Accelerated Segment Test) [12], BRISK (Binary Robust Invariant Scalable Keypoints) [13] or ORB (Oriented FAST and Rotated BRIEF) [14].

The question now is: which approach is better? The answer seems to be: it depends for what. There are many studies regarding this inquiry. In [15] the authors compared both approaches for Web images retrieval and they found that the local descriptors significantly improve the performance of research in the Web domain. A similar study ([16]) concluded that although the GIST descriptor provides very high accuracy for near-duplicate detection, when it

comes to object recognition the local descriptors outperform the global ones. It makes sense because the global descriptors consider the images as a whole, and in object recognition we don't care about the image itself, but about the object contained in it. For example: Let's think about one object that appears in two quite different pictures (taken from different viewpoints, rotated, with scale changes, occlusion...). In theory, by using a global descriptor we couldn't establish any relation between them, but with a robust local descriptor we could. For this reason, local descriptors have gained popularity in object recognition applications and this is the approach used in the thesis.

2.1.2.2 Local detectors and descriptors

Now that we know which kind of descriptor fits us better, we need to choose one to perform our experiments. To discover it, we should know how they work. The steps to subtract local features from an image are [17]:

1. Detect the interesting points with distinctive information. They are usually placed in corners and edges, points where there is an abrupt change.
2. Define a region around each *keypoint* in an invariant manner. The points itself are not the only thing that matters, we need to determine small areas around the point that are reliable against scale changes.
3. Extract and normalize the region content. This normalization prevents from rotation. This is typically done by finding the region's dominant orientation and then rotating the content according to this angle, in order to bring it into a canonical orientation.
4. Encode the normalized region in a *descriptor*.

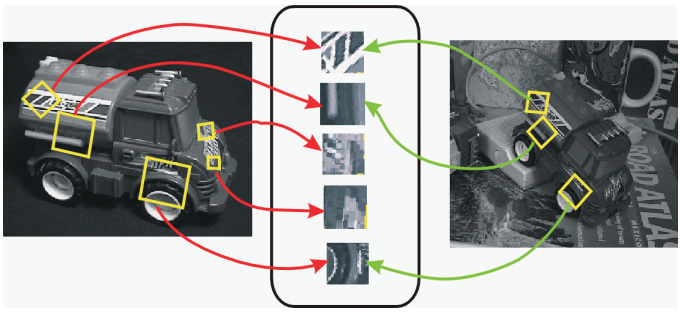

Figure 2.3: Example of local features that have been independently extracted, scaled and canonically oriented from two different images. Then, they have been matched among them. Figure extracted from [17]

One of the most popular local descriptors is SIFT. This is the one that we have used in the thesis. His popularity is due to his good performance and robustness against all the problems enumerated above, but there are many others. If the reader wishes to learn more about how local descriptors work and which different types exist, there is an exhaustive study carried out in [18].

2.1.2.3 SIFT: Scale Invariant Feature Transform

The Scale Invariant Feature Transform (SIFT) was published by David Lowe in 1999 in [6]. It is patented and its owner is the University of British Columbia. Over the years, it has demonstrated its effectiveness and robustness. There are many works explaining with high accuracy how SIFT works: [19], [20], [21]. We will look over the process briefly without going into the hard mathematics.

As David Lowe described in [19] there are 4 main steps followed by the algorithm to extract the local features from an image, and each of them has some sub-steps at the same time. Figure 2.4 shows them:

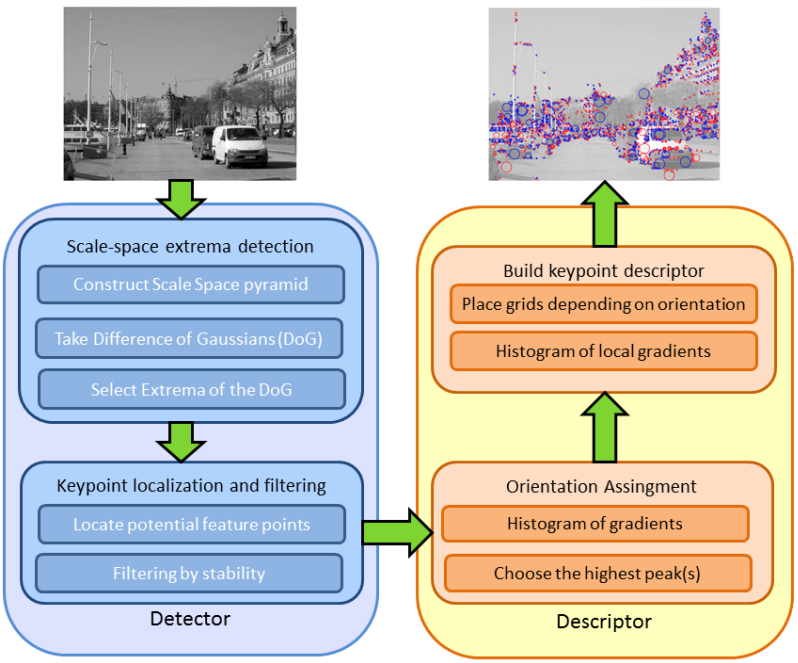

Figure 2.4: SIFT descriptor extraction process. We can see which stages are performed by the detector and the descriptor. Pictures extracted from [21]

1. Scale-space extrema detection. In this first step the potential interest points of the image that are invariant to scale and orientation are detected from scale-space extrema of differences-of-Gaussians (DoG) within a difference-of-Gaussians pyramid. The process begins with the construction of a Gaussian scale-space pyramid from the image. This is done by applying convolution to the image with Gaussian functions of different widths. Once we have it, we compute the difference-of-Gaussians between the levels in the scale-space pyramid to get another pyramid (Figure 2.5). Finally, the *keypoints* are obtained from the extrema values with respect to both the spatial coordinates in the image domain and the scale level in the pyramid.

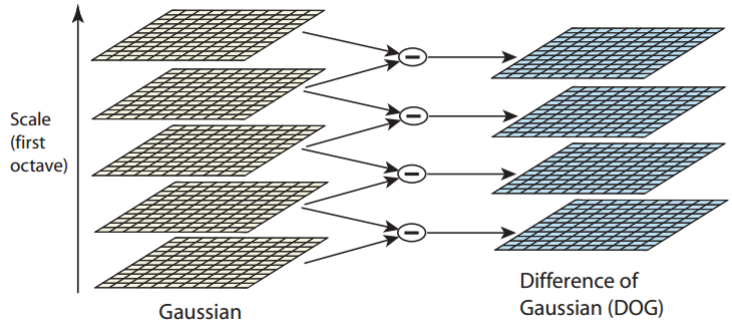

Figure 2.5: Gaussian and DoG pyramids. Extracted from [19]

2. *Keypoint* localization and filtering. Since we got the *keypoints* at different scales, we need to approximate the values to the truly points in the original image. This post-process is done by interpolation with Taylor expansion of the scale-space function. After this calculations there are still a lot of points, all of which are not good enough. Here, a *keypoint* filtering that rejects some low-contrast and edges points is carried out.

3. Orientation assignment. This step aims to assign a consistent orientation to each *keypoint*, so its representation can be relative to it and therefore achieve invariance against image rotation. In order to discover it, we compute the magnitudes and orientations of gradients of the Gaussian smoothed images over a neighbourhood around the interest points. Subsequently, a local histogram is formed by quantizing the orientations into 36 bins covering the 360 degrees range. Figure 2.6 represents an example. The gradient magnitude and orientation are computed using pixel differences.

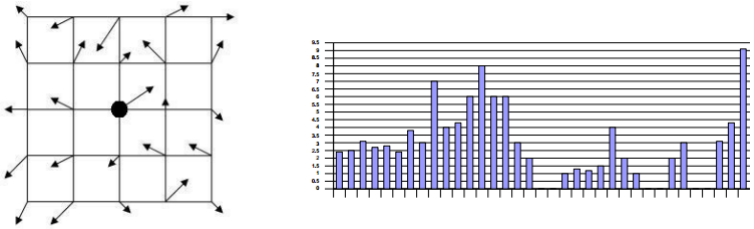

Figure 2.6: In the left, *keypoint* with gradients of orientations and magnitudes. In the right, histogram with 36 bins of such gradients. Extracted from [20]

To find the dominant orientation, peaks are detected in the histogram. To handle situations where there are more than one dominant orientation around the interest point, multiple peaks are accepted if the height of secondary peaks is above 80 percent of the highest peak height. In the case of multiple peaks, each peak is used for computing a new image descriptor for the corresponding orientation estimate.

4. Build *keypoint* descriptors. A rectangular grid is placed centered at the interest point, with its orientation determined by the main peak in the histogram and with the spacing proportional to the detection scale of the interest point. Typically, grids with 4x4 subregions are used. For each of this subregions a local histogram of local gradient directions is computed. These histograms have 8 bins, which represent 8 quantized directions. Thus, an image descriptor will have 4x4x8=128 dimensions for each interest point.

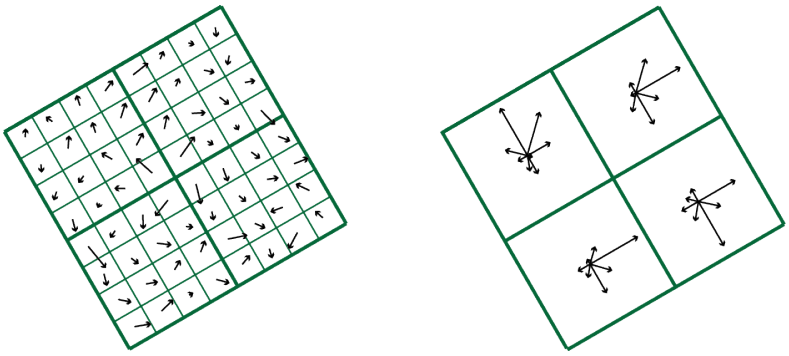

Figure 2.7: Computation of a SIFT descriptor. This figure shows an image descriptor computed over a 2x2 whereas the SIFT descriptor is usually computed over a 4x4 grid. Image taken from [21]

As we said, SIFT descriptors are a 128-dimensional vectors. They are quantized to 8 bits, which makes 1024 bits per descriptor. One image can have thousands of SIFT descriptors. They are transmitted along with their location in the image and with the meta-data of the mobile device. This means that it can happen that the data volume of the SIFT descriptors of one image is higher than the image itself with JPEG compression. This is the main disadvantage of SIFT: its high data size. To improve this characteristic, many approaches have been considered. One of them is the CHoG descriptors [8].

2.1.2.4 CHoG: Compressed Histogram of Gradients

CHoG aims to perform as good as SIFT descriptor (or even better) but with an important reduction of the bit rate. In [8], the authors claim that they have achieved a 20x reduction in the descriptor size, and also that they have outperformed other descriptors at lower or equivalent bit rates. There are many applications that require a low rate such as storage, latency and transmission. For this reason, CHoG is an interesting proposition.

CHoG is based on HoG (Histogram of Gradients) family of descriptors, as well as SIFT. For this reason, there are many similarities between SIFT and CHoG, and we will describe the CHoG steps briefly. The procedure is the following:

1. Compute interest points at different scales. As it happened with SIFT, these *keypoints* are usually corners and edges. Areas around the points are computed to subtract patches of different sizes from the image. The patches at different scales are oriented along the dominant gradient and scaled to create canonical patches.

2. The canonical patches are divided into localized spatial bins, which gives robustness to interest-point localization error. These bins are soft log-polar spatial bins, made using DAISY configurations proposed in [22]. Such configuration has shown to be more effective than the square-grid configuration used in SIFT.

3. A histogram of gradients within each spatial bin is computed and compressed. Also the bins of the histogram are important, because CHoG treats the information in each spatial bin as a distribution. This makes possible the use of quantization and compression techniques originally though for distributions, that help to get a very compact description of the patch. The number of spatial and gradient bins define the final dimension of the descriptors.

4. The descriptor is compressed by quantizing the gradient histogram in each spatial bin individually. The creators of CHoG tried with two different algorithms for compressing probability distributions: Huffman Tree Coding and Gagie Tree Coding. After the compression, there are two more steps carried out: the Tree Fixed Length Coding and the Tree Entropy Coding. The authors found that the Huffman compression gives better results.

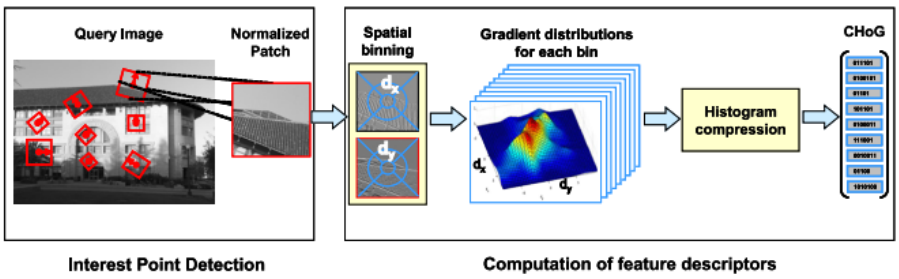

Figure 2.8: Computation of a CHoG descriptor. Image taken from [1]

The final results show a 20x reduction in the bit rate, low complexity and significant speed-up in the matching stage. Also, a very important point is that CHoG allows the distance computation between descriptors in the compressed representation, without the need of decoding.

Despite all those improvements, in this Master Thesis we have used the SIFT descriptors due to their easier management and their popularity. Also because the previous work carried out in the project in which the thesis is framed was using SIFT descriptors.

2.1.3 Vocabulary Trees

One critical problem in object recognition is how to deal in a reasonable time with huge databases composed by thousands of objects. Comparing all the query descriptors with all the database ones for every query image would be extremely slow. It is a naive and infeasible approach. The most famous approach to solve this problem is to cluster the descriptors into visual words.

2.1.3.1 Bag of Words

The comparison between texts and images is widely used in computer vision [23]. In text retrieval there is an approach that represents a document as a "bag" of important keywords. By using a set of well-chosen words as query, it is easy to find the best matching text. This philosophy is called bag of words (BoWs) and it is adapted successfully to image retrieval [4]. This text-image analogy considers images as documents where the words are feature descriptors that are considered interesting and distinguishable from others and that appear in different images. These feature descriptors are called *visual words*.

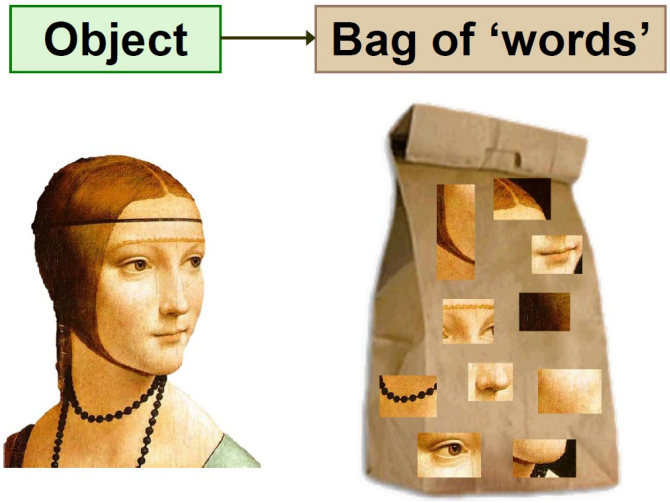

Figure 2.9: Bag of Words[1]

For using this idea it is necessary to create a visual vocabulary. Directly using the feature descriptors as visual words would not be efficient, and two descriptors almost identical would be considered different. Also, we should store millions of feature descriptors, which is infeasible. The solution is clustering the descriptors. Then, the very similar descriptors end in the same group and we can use the cluster center as a visual word. This process is a vector quantization and it makes possible to throw away the descriptors and use only the visual words.

[1]Image taken from: `http://gilscvblog.wordpress.com/`

In Figure 2.10 we can appreciate a scheme of the process of creating the visual vocabulary. In (a), feature descriptors (white ellipses) are extracted from a large set of images to populate the feature subspace. We can appreciate in (b) that the features are clustered in order to quantize the space into a discrete number of visual words. In the third step, (c), a query image comes and the nearest visual word is identified for each of its features. Notice that we got rid of the descriptors. Finally, in (d) a bag-of-visual-words histogram can be used to summarize the entire image and match it with the most similar one from the database.

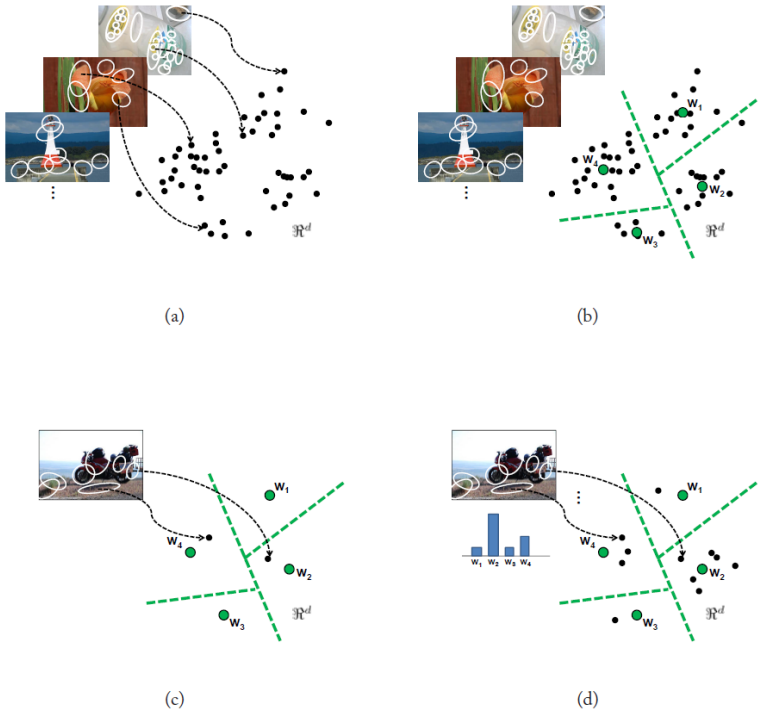

Figure 2.10: Creation of the Visual Vocabulary. The image belongs to [17]

2.1.3.2 *k*-means algorithm

Now the problem is how to cluster the feature descriptors in an efficient way. The original idea of using vector quantization for object recognition was suggested in [23]. The authors used the popular k-means algorithm to create vocabularies of 10k clusters. Due to k-means is widely used in feature descriptors clustering and also important to understand the vocabulary tree, we will explain it briefly.

It is difficult to assign a single author to k-means. Many persons contributed separately to different versions of this well-known algorithm. An historical approach is carried out in [24], where it is explained which authors worked in the different versions of the algorithm, and which were the applications.

Basically, k-means is a vector quantization method that partitions a set of d-dimensional N observations $(x_1, x_2, ..., x_N)$ in k different cells $c_i, i = 1...k$ such that the within-cluster sum of squares (WCSS) from the data points to the cluster centers are minimized. The algorithm computes:

$$\arg\min_{\mathbf{c}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in c_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \tag{2.1}$$

where $\mu_i, i = 1...k$ are the cluster centers obtained from the mean of all the points remaining in such clusters. Typically, the distance used for it is Euclidean, but there are also many other distance functions that can be used: the Manhattan distance, the Hamming distance, the correlation between points...

k-means is an iterative algorithm that keeps repeating some steps until it finds convergence. The stages are the following:

1. Initialize the starting centers of the clusters $\mu_i, i = 1...k$. These k points need to be chosen at the beginning as cluster centers to start the iterative algorithm that will refine their positions in each iteration until finding the convergence.
2. Determine the closest cluster to each observation based in the distance function chosen.
3. Update the positions of the clusters to the mean of all data points belonging to that cluster.
4. Repeat steps 2-3 until convergence. Convergence is defined as the situation in which the assignments do not change from one iteration to the next one.

To illustrate clearly these steps and understand better the algorithm, we have prepared a simple example in Figure 2.11[2]. In the figure there are six stages. First, we have the observations that we want to cluster. In this case we have chosen k=4 to make it easy. In 2), we choose the 4 starting points that will be our cluster centers. The different methods of choosing the initial conditions are explained later on. Here we have used the Forgy method. We can see in 3) how the observations are assigned to their closest centers. After that, it is time to update the positions of the centers by calculating the mean of the points within each cluster. Then, in 5), k-means rearranges the observations to the new cluster centers. Finally, the last update of the cluster centers is carried out. We have finished here the algorithm because we found the convergence, new iterations will neither change the positions of the centers or the assignments of the observations. Pay attention to how the cluster center of the green dots has started in the the upper-left cluster group of observations and has moved to the upper-right.

[2]It has been done with the help of the tool available in this webpage, where there is also an explanation of k-means: `http://www.onmyphd.com/?p=k-means.clustering`

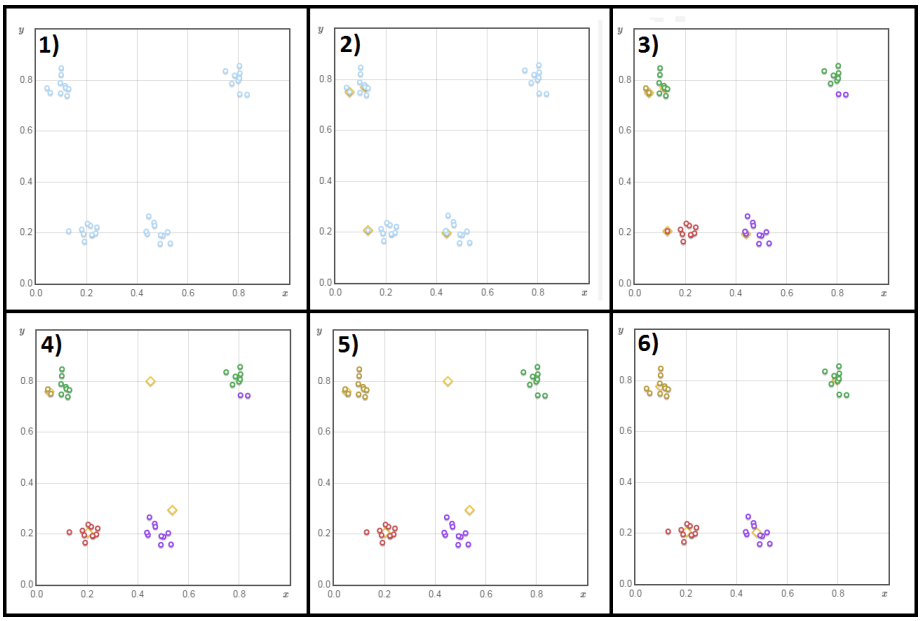

Figure 2.11: Example of a simple clustering in 6 steps by using k-means

The convergence is not necessarily the global minimum of the sum of squares, because the problem is non-convex. For this reason, the most usual is that the algorithm converges to a local minimum. There is no guarantee that the global optimum is found using k-means.

Since the algorithm stops in local minimum, the initial conditions (starting points of the clusters) are very important and define the results of the clusters, but there isn't any method that guarantees to find the global optimum. There are many options to choose the starting points. The Forgy method consists on choosing randomly k observations from the dataset. The Random Partition method assigns a cluster randomly to each observation and then computes the means as in the step 3. Gonzalez's algorithm [25] initializes the cluster centers with points that have the maximum distance possible between them. A very common way is to run the algorithm many times with different starting conditions and get the one that gives better results.

k-means is widely used in many different applications because it is relatively efficient and fast. Its complexity is $O(kNdI)$, where k is the number of clusters, N is the number of d-dimensional observations and I is number of iterations. Its applications are very diverse, it is used in fields like machine learning, image processing, pattern recognition, data mining, geostatistics, astronomy or even in agriculture.

2.1.3.3 Hierarchical *k*-means: Vocabulary Trees

This popular clustering algorithm described in the previous section was the first used to group feature descriptors into visual vocabularies, like for example in [23]. It results efficient when we apply it to small databases made of a few thousands of visual words, but it is difficult to extend its use to larger vocabularies. It becomes very slow to create the visual words when the number of clusters is high.

There have been approaches to make k-means more efficient in clustering big volumes of data, like the one carried out in [26]. The authors use an alteration of the original k-means called *Approximate k-means*. Although they get an efficient clustering, there is still the problem of matching. Imagine that we can cluster in an effective way a huge database into millions of groups. In the matching process we would need to compare each of the query descriptors to all those millions of visual words to find the closest similarity, which is very slow.

To make this process efficient, David Nistér and Henrik Stewénius proposed in [5] a scheme to structure the data called Vocabulary Tree (VT) that offers good performance and is scalable to big databases. A vocabulary tree is a product of a hierarchical k-means clustering that quantizes the feature descriptors by using the k-means algorithm recursively. It needs an offline training phase in which the tree is created from the data set.

This training stage begins with the division of the data in k different clusters by using k-means. The k cluster-centers will be the nodes of the first level of the tree. The same process is performed in each of the groups to get other k clusters within each of the first-level clusters, i.e., we apply the k-means algorithm recursively,dividing each cluster in k other clusters. This is done repeatedly until we reach a maximum depth D. At the end of the procedure we will get a vocabulary tree with branch factor k and D levels of depth[3]. Each node of the tree is a visual word and the nodes at bottom level are called leaf nodes. Our visual vocabulary will be composed by the k^D visual words in the leaf nodes.

The values of k and D describe the shape of the VT and how big will the vocabulary be. They are defined experimentally, depending on the data and the size of the database. There is a trade-off: if it is too small, the performance of classification will be very poor; if it is too large, it will perform well because the clusters will be small and the quantization error small, but it will be inefficient and difficult to store. Regarding the value of k, on the one hand, it is stated in [5] that while the computational complexity of increasing the size of the vocabulary in a non-hierarchical manner (conventional k-means) would be very high, the cost in the hierarchical approach is logarithmic in the number of leaf nodes. That brings us to choose a small k to be efficient. On the other hand, some experiments show that a higher branching factor offer better quality in the search, but not a dramatically improvement. A typical value often chosen is k=10, although other similar values can be also a good choice.

[3]Don't confuse the VT with branch factor k and depth D with a k-d tree. In other works the notation used for depth is L, but we decided to use D in our code for practical reasons.

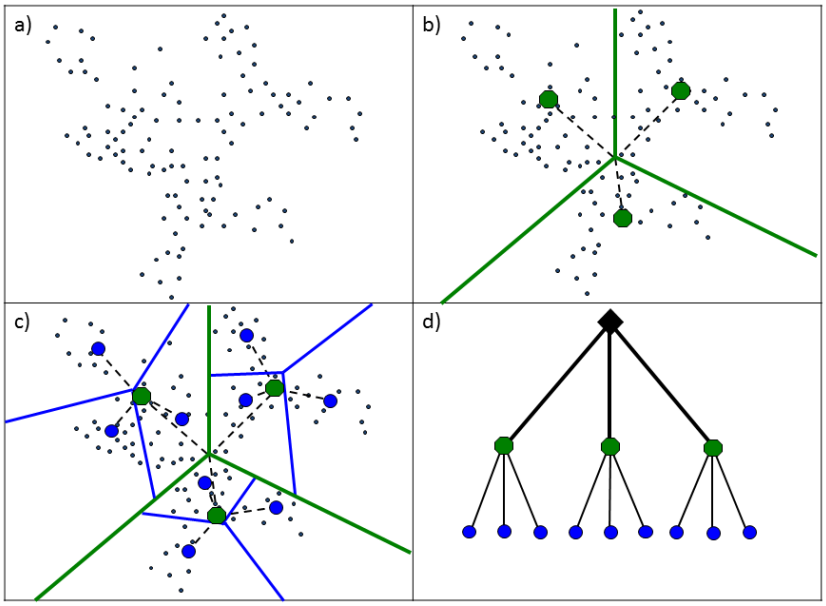

Figure 2.12: Creation of a Vocabulary Tree with k=3 and D=2

We can observe an example of the training step in Figure 2.11. In (a) there is a set of points. Each of the points represents a SIFT descriptor from the database. Although SIFT descriptors are 128-dimensional, we represent them in this figure as 2D points to describe appreciate the process with clarity. In (b), the first level clustering has been executed with k=3. The green lines define the Voronoi cells and the dots represent the centers of those cells (made as the mean of all the points within a cluster). We can see in (c) that a second level clustering has been carried out in each of the first level cells. This time the lines and the cluster centers are painted in blue. In (d), we get rid of the SIFT descriptors and keep only the structure obtained from applying hierarchical k-means. In this simple case the vocabulary tree has a configuration of k=3 and D=2, what makes a total of 9 leaf nodes, the visual words that compose our visual vocabulary.

The purpose of the vocabulary tree is, besides to save storage space, to make the matching process much more efficient by comparing the query descriptors only with the nodes of the tree instead of doing it with all the database visual words. Each query descriptor is compared with the k first level nodes using Euclidean distance to find the closest one. Then, we repeat the operation with the k nodes born from the chosen one at the first place. The procedure is performed until a leaf node is reached. With this method, each query descriptor is propagated down the tree and it is only necessary to make k comparisons at each level. This structured matching substantially reduces the computation needed to find the closest visual word (now it is only kD dot products) and enables the creation of large visual vocabularies.

2.1.3.4 TF-IDF scheme

To recognise a query image the most common procedure is to assign to every database image a similarity score that describes how similar it is to the query image. A candidates list of the most probable images will be the output, sorted in function of this similarity score. The question now is how to assign this similarity score.

As explained before, the Bag of Words approach makes possible to use text retrieval methods in image retrieval. One of the most popular is the Term Frequency-Inverse Document Frequency (TF-IDF) weighting scheme [27]. This technique is key in text retrieval because it measures how important are the words within a set of documents and assigns them a weight in consequence. It makes possible an easy matching of documents by using as query a few well selected words.

It is easy to understand TF-IDF when thinking about text. Let's imagine a set of text documents about different topics with different lengths. We will assign a score to every word, in every document, that will represent how important this word is. This score is the result of the multiplication of two terms: *tf* and *idf*. The first term, *tf*, reflects the frequency of a word within a particular document, because we assume that a word is relevant in a document when it appears very often. Mathematically it is defined by:

$$tf_{i,k} = \frac{n_{i,k}}{n_k}, \tag{2.2}$$

where $n_{i,k}$ is the number of occurrences of the word i in the document k, and n_k is the total number of words in the document k. The term n_k is used as a normalization due to the different lengths of the documents. Without it, a longer text would have a much higher probability of containing the words. The second term, *idf*, characterizes the importance of a word over the whole set of documents, the global weight of the word. This is important because there are some words that are more frequent than others, like "that" or "for". The common words are penalized by this term. It is defined as:

$$idf_i = log\frac{N}{N_i}, \tag{2.3}$$

where N is the total number of documents in our corpus, and N_i is the number of documents where the word i appears. Finally, the weight assigned to a word in a document is the product of both terms. Thus, the highest weights belong to the words that appear many times in the document but that are rare in general. The resulting formula is:

$$w_{i,k} = tf_{i,k} * idf_i = \frac{n_{i,k}}{n_k} log\frac{N}{N_i}, \tag{2.4}$$

In image recognition it is used exactly the same procedure that we just described for text retrieval, with the only differences that now the words are visual words (the cluster centers of the leaf nodes of the tree) and the documents are images. The rest remains the same. The matching process begins with the initialization to 0 of all the scores related to the images. When the server starts receiving the query descriptors of an image, it analyses them one by one using the VT until they reach the bottom of the tree. Let the set of leaf nodes matched with the query for the k-th database image be denoted as $\mathcal{I}_k$. When all the query descriptors have passed through the tree, the final similarity score for the k-th image is:

$$s_k = \sum_{\mathbf{i} \in \mathcal{I}_k} w_{i,k}, \tag{2.5}$$

At this point all the scores are sorted in descending order so the first image in this list is the most similar to the query and the last one represents the most different. Then, it is cut keeping only a small number of objects from the top, dismissing the rest of images. This small list of the most probable images is the candidates list that will be further analyzed in the following steps of MVS.

The first usage of TF-IDF in image retrieval with visual words instead of normal text words was in [23] and it worked so well that it has been widely used in many other works ([1],[28]). It is important to state that the formulas above exposed to calculate the *tf* and *idf* are not the only ones. There are many different implementations and modifications of TF-IDF. The one proposed here is very common due to its simplicity, but there exist others like the one proposed by Robertson *et al.* in [29], which is more complex but efficient. There are also modifications or adaptations of this method for different purposes, like in [30], where the authors merge together different vocabularies that weight the visual words by using properties of the patches such as color, texture or shape.

2.1.4 Geometric Verification

GV is, as you can see in Figure 2.2, the name of the stage that follows the vocabulary tree matching. Until now we have extracted feature descriptors from the images, clustered them hierarchically in a VT with k-means and defined a weighting method (TF-IDF) that allows us to get a candidates list of the most similar images from the database to match with a query. In all this process we have omitted completely the spatial structure of the objects. The only information used in the retrieval algorithm until now has been related to the visual appearance (SIFT). The next step is to take profit of the locations of the feature descriptors within the images to rearrange the candidates list in function of the geometric consistency with the query. The first image in this reorganized candidates list will be our output.

The GV step is in general computationally complex. It is important to state two different problems related to the geometry of the features locations. The first one is how to compare the query spatial pattern with the ones in the server

in a robust way and efficient. The second one is the possibility of mismatching, i.e., query descriptors that are matched wrong with database descriptors. Thus, they shouldn't be taken on account. There are different methods to deal with these problems. We will briefly explain some of them.

2.1.4.1 RANSAC

Fischler and Bolles proposed in 1981 a robust method for fitting a model to experimental data called RANSAC (RANdom SAmple Consensus) [31] that has been very popular and widely used since then. This iterative technique assumes that the data contains two types of observations: *inliers* and *outliers*. *Inliers* are the samples that follow a pattern or model and *outliers* are the observations that are not consistent with the model. Inliers can be noisy but still follow the pattern, while outliers come from erroneous measurements or too noisy observations. RANSAC takes all the data points as inputs along with some parameters like the tolerance of how much noise will we allow in the observations. Then, it outputs the data indicating which observations have been considered inliers and which outliers.

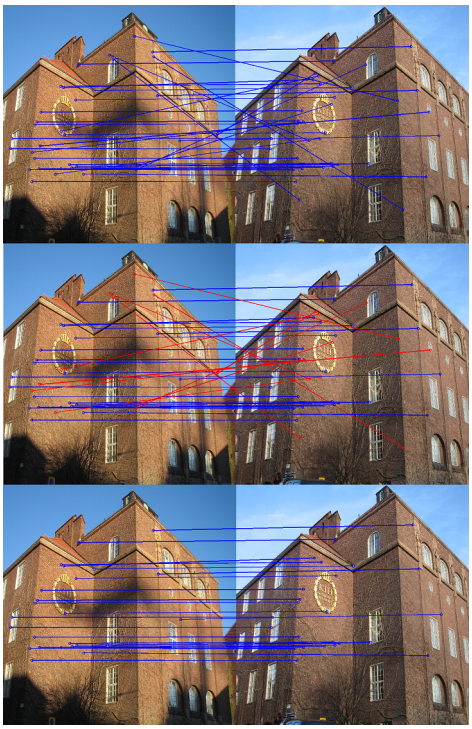

Figure 2.13: Example of RANSAC. In this example we have matched a query image of KTH with the database one, but there are some wrong matches that we would like to remove. After running RANSAC, we can differentiate between inliers (red) and outliers (blue). Finally, we throw away all the mismatches and keep the correct ones.

RANSAC is considered very robust. Its estimations of thee models are accurate, so it results very useful to discern between inliers and outliers. The problem is that it takes large amounts of time. It is an iterative algorithm: as more iterations are done, more exact is the model. If we limit the iterations, the solution can be not optimal. Another disadvantage is that it is not guaranteed that some outliers will be defined as inliers and the other way around.

2.1.4.2 Fast Geometric Re-ranking

Due to the slowness of RANSAC, it is infeasible to perform it in a long list of candidates. Some authors have proposed to introduce an intermediate step called *Fast Geometric Re-ranking* (or just *Re-ranking*) that can help to speed up the GV [32]. The idea is to run a much faster but less accurate algorithm that can use the geometric information to reorder the candidates list and make it much shorter, and then run RANSAC or some other similar method on it.

In [32] Tsai *et al.* propose a re-ranking that uses the *(x,y)* locations to ensure a consistency between the query and the database images. The procedure is represented at Figure 2.13. To achieve it, first they generate a set of pairwise correspondences of features using the VT (a). After that, they calculate geometric distances between the points within the images separately (b), and compute log-distance ratios of the pairs of distances as we see painted in colors in (c). These ratios represent the scale change of the objects. Then, a histogram of the ratios is calculated (d) and the final score of this fast geometric re-ranking is the maximum value of the histogram, because a peak on it can reveal a high similarity between the query and the database images. We have implemented a *Fast 3D Geometric Verification* algorithm for the multi-view approach based on this work [33]. It is explained in Section 3.2.2.

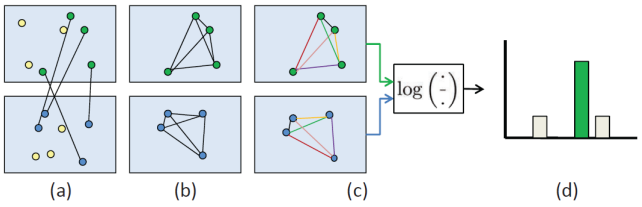

Figure 2.14: Fast geometric re-ranking. Image credit: [32]

Another work proposes a similar but not identical way of performing the re-ranking stage [28]. The main difference is that while [32] assumes a single global affine transform between the query and the database images, the authors of [28] do not. They claim that this global transform may not hold under some circumstances. For example, when modifying the point of view in 3D objects, the structure formed by the feature locations may change in a different way than the scale or the orientation. Thus, Wang *et al.* develop a re-ranking method that compares the spatial local neighborhoods of the features in the query and the matched images. The features with larger common neighborhood contribute more to the final re-ranking similarity score.

2.2 Mobile 3D Visual Search

As we introduced in Chapter 1, the purpose of this thesis is to adapt the previously described vocabulary tree to our multi-view approach. Our idea consists on improving the mobile visual search by using multiple images of the objects taken from different points of view instead of using single images. Remember that current approaches use 2D image-based features, so they can't take on account the geometry of the objects. The advantages of using multi-view features are:

- Features with established correspondences among the views are more robust to changes of perspective and varying lighting conditions.
- The spatial structure of the objects can be exploited for a more accurate geometric verification.
- Using the representative descriptors reduces the redundancy of memory occupation significantly.

With this new approach, it is necessary to adapt all MVS stages. Although the thesis is focused on the changes done regarding VT, we will briefly explain in this section how have we conducted the multi-view approach in the other parts as well.

It is important not to confuse the Geometric Verification step with the use of the objects 3D geometry, they are different things. One advantage of using multiple views is to extract the 3D information. Then, this 3D information will be used in the GV step. This 3D Geometric Verification step will have the same mission as the classic GV, but using 3 dimensions instead of planar objects.

2.2.1 Hierarchically Structured Multi-View Features

To extract the underlying 3D geometry of the objects it is necessary to use more than one image of every object. The point now is how to get feature descriptors across the views of multiple images. The strategy followed solves this question efficiently and provides a hierarchy among the descriptors that allow us to select the most robust of them [34].

The first step consists on extracting the SIFT features from all the images individually. There is a high redundancy in the feature space because many descriptors appear in multiple images of the same object. These are the descriptors that we are interested in. The features that appear in many views of the same object usually represent very characteristic points from the foreground, which means that they are relevant and robust against changes of perspective and lighting conditions, while the ones that appear just in a few or a single image probably belong to background objects that have a negligible discriminative power.

Figure 2.15: Multi-View correspondences

Let $f_i \leftrightarrow f_j$ be the multi-view feature correspondence where f_i represents a feature in the i-th image and f_j represents the same feature in j-th image. Then, we can define a set of feature correspondences among l images as

$$C^l_{i,j,...,k} = \{(f_i, f_j, ..., f_k) | f_i \leftrightarrow f_j \leftrightarrow ... \leftrightarrow f_k\}, \tag{2.6}$$

We can organize the set $C^l_{i,j,...,k}$ in a hierarchical manner in function of the number of views where the features appear. Usually, the distribution of a set forms a pyramid structure with a small number of robust features on the top, while the ones at the bottom are less reliable and more abundant, as we can observe in the Figure 2.15.

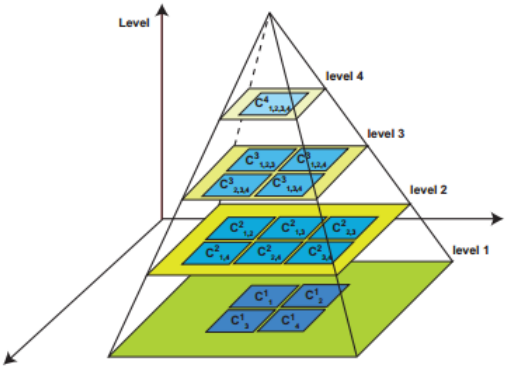

Figure 2.16: Pyramidal set of features

Every correspondence $f_i \leftrightarrow f_j \leftrightarrow ... \leftrightarrow f_k$ has l feature descriptors (one per view), which results very redundant. We follow the strategy of finding a common feature descriptor for all of them by taking the median as a robust estimate:

$$\widehat{d^l(u)} = Median\left\{d^l_h(u) : h = i, j, ..., k\right\}, u = 1, ..., 128 \tag{2.7}$$

where $\widehat{d^l}$ is the new descriptor, d^l_h is the feature descriptor of the h-th view and u is the descriptor dimension. This hierarchical pyramid makes possible a good feature selection from the top to the bottom. By truncating the pyramid, we can create a vocabulary tree that offers a higher performance than the conventional approach.

2.2.2 Multi-View Geometric Verification: A 3D approach

As in conventional mobile visual search, in our 3D approach it is necessary a step that can rearrange the candidates list obtained from the vocabulary tree to discard outliers and improve the results. Now this stage is even more challenging than before due to the third dimension that we need to take on account.

2.2.2.1 Helmert transformation of stereo features

To extract the 3D geometric information of the objects, the client needs to take more than one picture of the object intended to be recognised. As explained in [3], a combination of the features locations along with intrinsic camera parameters (such as the focal length or the image width in pixels), makes possible to extract the set of 3D world coordinates of the query object. Also a self-calibration is done to align the features from all the different views and the RANSAC algorithm discards the possible outliers.

After the VT matching is done, it is time to compare the 3D world coordinates of the query (W^c) and server (W^s) objects. The seven-parameter Helmert transformation is used for this purpose, which takes on account changes on scale and rotation:

$$\vec{w^c} = s\,\Phi\,\vec{w^s} + \vec{t}, \tag{2.8}$$

where $\vec{w^c}\,\epsilon\,W^c$, $\vec{w^s}\,\epsilon\,W^s$, s is the scale parameter in $\mathbb{R}^+$, Φ is the rotation matrix in $\mathbb{R}^3$ and $\vec{t}$ is the translation parameter in $\mathbb{R}^3$. Due to misalignments caused by erroneous calibrations, the RANSAC algorithm is used here to get rid of the outliers. By applying this method it is possible to compare the query and database spatial structures and find which object from the candidates list is more similar to the query.

2.2.2.2 Fast 3D Geometric Verification

Although the technique presented above offers a good performance, the usage of the RANSAC algorithm to distinguish the outliers from the inliers makes it very slow. A fast geometric re-ranking method similar to the one described in Section 2.1.4.2 can be a better choice. With the 3D geometric information of the object, it is possible to efficiently extend this method to the 3D space which is more robust against a perspective transformation in the object.

With this idea on mind and keeping the use of the Helmert transformation, we have developed a 3D GV step called *Fast 3D Geometric Verification* that can be executed much faster than other RANSAC-based methods [33] and offers a high performance. It is explained in Section 3.2.2.

Chapter 3

Multi-View Vocabulary Trees

This chapter presents the multi-view vocabulary tree for mobile 3D visual search. To construct it, we use the hierarchically structured multi-view features extracted from multiple images per object. The new object based TF-IDF weighting scheme and scoring function have been incorporated in the multi-view scenario to improve the recognition performance. Also, we utilize the 3D geometric information associated with the multi-view vocabulary tree to implement a fast 3D geometric verification. We include details of the MatLab implementation.

3.1 Multi-View Visual Vocabulary

A vital issue in the performance of the VT is its codebook, i.e., the visual vocabulary composed by the visual words of the leaf nodes[1]. In this section, three main aspects about it are considered. The first one is the database used to build it, the *Multi-View Features*. After that, we explore the options regarding the *Codebooks Universality*. Finally, we expose the *Memory-Constrained Multi-View Vocabulary Trees*.

3.1.1 Multi-View Features

The main improvement of mobile 3D visual search comes from the usage of multiple images taken from different perspectives for each object. To handle them, we use multi-view features, as explained in Section 2.2.1, and we organize them in a hierarchical pyramid in function of the number of views where they appear. The descriptors from the top are very relevant, while the ones in the bottom levels are usually from background objects that are less helpful [34]. Thus, we decided to truncate the database from top to down, using a small set of high quality descriptors and discarding the less discriminative ones.

[1]We use *codebook* and *visual vocabulary* as synonyms

3.1.2 Codebook Universality

Codebook universality is a current research topic [35]. The idea of a universal codebook consists on creating a general visual vocabulary from a huge number of feature descriptors of a big variety of objects that can fill all the descriptor subspace. Thus, it would be suitable for applications with objects of different nature. From the VT perspective, once a general tree (a VT with a universal codebook, the leaf nodes of the tree) is built from a big enough database, it could be used with others after a training. This training would consist on passing the descriptors from our database through this VT and just set the TF-IDF weights. Any clustering would be needed here because the structure of the tree would have been already constructed. It is much simpler than building the whole tree from the beginning.

A universal visual vocabulary has some advantages and also disadvantages. On the one hand, the off-line process of generating a VT for our application would be much faster with a general codebook, because the clustering is already done. It also allows an on-the-fly insertion of new image descriptors. On the other hand, the universal codebook approach comes from text processing and while in a language there is a limited number of words with which all texts can be constructed, in image processing most researchers do not think there is a universal codebook for all image databases. Furthermore, a universal vocabulary would probably occupy a large storage space that would result useless in some applications, specially when these are very specific. For example, in an application about the artworks of a concrete museum, why should we use a VT that contemplates all possible objects when just a limited number of visual words will appear in the paintings?

The usual tendency is to build the codebooks for given databases and not use them in others. In [35], the authors study the possibility of building general visual vocabularies. Their conclusion is that it is possible, but the database that creates the codebooks has to be very large. They also state that the behavior of the codebook is more complicated than expected and further works are required. Due to the above mentioned reasons and the particularities of our database (we use multiple images per object instead of single images), we chose to build a specific VT, as many other works do [36].

3.1.3 Memory-Constrained Multi-View Vocabulary Trees

The next step regarding the visual vocabulary is to decide its size, the number of visual words (given by k and D) that we will create from our database. There is a trade-off between the size of the vocabulary tree and its retrieval performance. If it is too small, the performance of the matching will be very poor; if it is too large, it will perform well but it will be harder to store and will consume more time in the matching process. It has been shown in [5] that the memory occupation of the vocabulary tree is linear in the number of leaf nodes K^D. For N-dimensional descriptors represented with **single** precision, the size of the tree is approximately $4NK^D$ bytes.

On the one hand, it is desirable to build a tree with a high number of visual words for a given database, because the clusters will be small and the quantization error will be reduced. The problem is that to build such VT with a high number of visual words in a big database composed by millions of descriptors would take a very long clustering time and would increase the storage space in the sever end. Depending on the application and the hardware specifications of the server, the authors of each work decide experimentally how precise can they allow their visual vocabulary to be.

In this situation, our solution is, as explained above, the database truncation. It is desirable to choose more discriminative and representative descriptors to generate a tree with a very precise codebook, with a large number of visual words, but still light enough to store it and use efficiently. Without this feature selection we couldn't build such accurate visual vocabulary because the size of the VT would have been far too big to store it. Instead of building a very large VT for all the database, our strategy consists on keeping a reasonable size of codebook but with well-chosen descriptors. In other words, with our selection method we can build vocabulary trees that offer a very high performance occupying small storage space.

With the idea of constructing a big codebook, we have configured our implementation of the hierarchical k-means algorithm to keep growing until the clusters can't be further partitioned, instead of setting a depth. While it is possible to keep clustering, we do it. In this new scenario, the number of descriptors that form the visual words is very small, always less than k. We call the trees generated with this idea, *Memory-Constrained Vocabulary Trees*.

Under these circumstances, the TF-IDF weighting scheme can be improved to be more discriminative. On the one hand, the weights of the visual words have now a smaller variance, this limits their discriminative power. On the other hand, the tiny size of the clusters makes the system more sensible to the descriptor noise, it is possible that some descriptors fall in wrong cells. For these reasons, we have decided to adapt the scoring scheme.

3.2 Construction of Vocabulary Trees

This section focuses on the details of implementing such VT in MatLab. It is divided in two main points. Section 3.2.1, *Hierarchical k-means Algorithm*, talks about the problems we faced when writing the files that create VT and how we dealt with them. There is a deep explanation, including pseudocode, of how we decided to implement this recursive algorithm. Section 3.2.2, *Types of nodes*, explains the three different kinds of nodes we have in our implementation of the vocabulary tree.

3.2.1 Hierarchical *k*-means Algorithm

The main part of the so-called Hierarchical k-means is, obviously, the popular k-means algorithm. Due to its wide usage, MatLab includes an efficient implementation of the algorithm[2] and it hasn't been necessary to write it. However, we had to configure it to obtain a properly behaviour that could be used recursively in the creation of the VT.

3.2.1.1 *k*-means Set-Up

Although k-means is already implemented in MatLab, it is very important to configure the options of the algorithm to make it work as we wish. An important parameter is how to choose the seeds, i.e., the initial points from where the algorithm starts clustering. Depending on this choice, the performance of the algorithm can vary significantly. As described in Section 2.1.3.2 there are many ways of choosing the starting conditions, and no one assures the optimal performance. The Forgy method chooses k random observations from the data. The Random Partition assigns random clusters to the observations and then calculates their means. The MatLab option *uniform* selects k points randomly from the subspace of the data. Another option called *cluster* performs a preliminary clustering on a random 10% of the data. There also algorithms focused on finding starting points, such as previously explained Gonzalez's algorithm [25], or k-means++ [37], which claims to improve the running time of classic k-means, and the quality of the final solution.

We tried most of these starting methods to choose the most appropiate. Unfortunately, MatLab incorporated the k-means++ algorithm in the 2014 version, but we didn't dispose of that version when experimenting with them, so we couldn't try it. We found that the most interesting ways to choose the seeds in our application were two: the Forgy method and Gonzalez's algorithm[3]. The solution we decided to use to avoid the local minima when using the Forgy method is to run k-means several times with different random initial points and take the realization wich gives the better results. This is not necessary in Gonzalez's algorithm. Due to its deterministic behaviour, we always get the same clusters when we run it with the same parameters, because it always extracts the same starting points from the dataset. Thus, just one iteration is needed. However, we finally chose the Forgy method (in MatLab it is called *sample*) because after some realizations we usually get a higher performance than with Gonzalez's algorithm.

It is something usual in k-means that at some point a cluster loses all its observations and becomes an *empty cluster*. There is a limited number of options here. The first one is considering it an error and stop the algorithm. The second is to remove the *empty cluster* and continue with the clustering but with one cluster less (K-1 clusters). This option doesn't suit us because we need

[2]MatLab k-means page: `http://es.mathworks.com/help/stats/kmeans.html`

[3]We experimented with the implemented version in MatLab carried out by Yuan Yao, available in `http://math.stanford.edu/~yuany/pku/matlab/kcenter.m`

to have exactly k clusters at each level of the VT. The option we chose is called *singleton* in MatLab. It consists on creating a new cluster formed by the point which is further from any other cluster and continue the process as usually. It has demonstrated to be a good solution.

Regarding the distance function used to measure the distance between the points, there are several choices: Euclidean, Hamming, cityblock, cosine... Although in [5] the authors state that they get better results with L_1-norm distance, we observed a better performance when using L_2-norm distance, i.e., Euclidean distance, which is also a common choice [1].

The last parameter we modified is the maximum number of iterations. k-means is an iterative algorithm, it starts clustering with k initial points and each iteration refines this clustering. It stops when there isn't any change from one iteration to another, i.e., when it finds convergence. The problem is that the algorithm can get stuck trying to converge without achieving it, consuming large amounts of time. In order to avoid an excessive time consumption, we need to limit the number of iterations. This limitation has to be large enough to assure a proper clustering (achieving convergence or at least to get close to it) but it also has to be small to avoid unnecessary iterations. There isn't a right choice in this issue, it depends on the data, its distribution, the starting points, the branching factor... MatLab uses a limit of 100 iterations, but we increased it to 1000 due to the difficulty of clustering SIFT descriptors and to ensure a good clustering.

3.2.1.2 Structure of Hierarchical *k*-means

Now that it is clear how we configured k-means, it is time to apply it recursively to create the different levels of the VT. To do it, the way we have structured the program of Hierarchical k-means is the following (in pseudocode):

```
function HIERARCHICAL k-MEANS(data,k,D)
    Initialization of the tree
    Recurse(data,k,D)
    function RECURSE(data,k,D)
        L=size(data)
        if current depth=D OR L<k then
            Create leaf nodes and add them to the VT
            Break
        end if
        clusters=k-means(data,k)
        Add cluster centers to the VT
        for i=1:k do
            Recurse(clusters(i),k,D)
        end for
    end function
end function
```

The main program gets as inputs the data that we are going to cluster and the two main parameters of the VT: the branching factor k and the depth D. The first we do is to initialize the tree: we create the root node (the variable that holds the whole VT) and initialize other parameters. Then, we call the function *Recurse*, which is in charge of applying k-means recursively. Within *Recurse*, the first step is to check if we have reached D and thus we need to stop growing. The tree doesn't grow uniformly, the number of descriptors that are contained in the resulting cells after a partition are usually different. For this reason, we have configured our implementation of the hierarchical k-means algorithm to stop growing under two circumstances:

- The depth D is reached.
- The depth D is not reached but a cluster can't be further partitioned because it contains less than k descriptors.

In both situations we can't continue clustering and it is necessary to "cut the branch". At this point, we create the leaf nodes, add them to the VT and exit the program. If the algorithm didn't get into the *if*, it means that we have to continue executing k-means on the data that *Recurse* received. After doing it, we have k resulting clusters with their respective k cluster centers. We add the centers to the VT corpus in the corresponding level and use a *for* loop to send separately to *Recurse* all the observations that belong to the different clusters. All the process is performed again until every *branch* of the VT reaches D or gets less than k descriptors within a cluster.

3.2.2 Types of Nodes

Now that we have shown how to build the VT, we will explain the different types of nodes that compose it. There are three different types of nodes in our implementation: the *root node*, the *inner nodes* and the *leaf nodes*. Each of them has different characteristics and purposes. We explain them in this subsection.

3.2.2.1 Root node

There is only one root node in the VT and it contains the whole tree. It has some parameters that give information about the tree:

- K: Branch factor. It is the number of branches that "born" from the root and inner nodes. This means that every node of the VT, except for the leaf nodes, have k "sons".
- *Depth*: The maximum level that can achieve the tree. It defines, besides K, the shape of the tree. As explained above, we can't guarantee that all the branches will grow until reach this depth. Some of them will stop growing before if a cell can't be further partitioned due to the lack of data.

- *Leaf nodes*: Number of leaf nodes that have been created in the VT.
- w: This is a vector needed in the matching process. It contains the parameter n_k of the TF-IDF weighting scheme, i.e., it indicates how many descriptors has each object of the database.
- *Quality*: It is the average distance between the leaf nodes centroids and the data-points. We could think about it as the average quantization error committed in the clustering. We called this parameter *Quality* because it gives a general idea about the performance of the tree. If we build a small tree compared to the amount of data, this parameter will be high and the matching process will perform poorly. The same happens in the other way around. We can't define it as the main factor that defines the performance of the VT because many other are also important.
- *sub*: A structure with the first level of k inner nodes. It contains all the VT.

3.2.2.2 Inner nodes

These nodes are situated between the root node and the leaf nodes. They form the "body" of the tree. Each of them is a structure with three parameters:

- *Center*: The centroids of the clusters. They have the same dimension as the feature descriptors, in this case 128. They are visual words, computed by the k-means algorithm as the mean of all the observations assigned to the cluster.
- *sub*: The k "sons" of the node. They can be either inner or leaf nodes, depending on which level they are.
- *idx*: It is a vector containing the path of the tree until reaching the node. The length of this vector is the level of its node, and their components describe which node composes the path in every level.

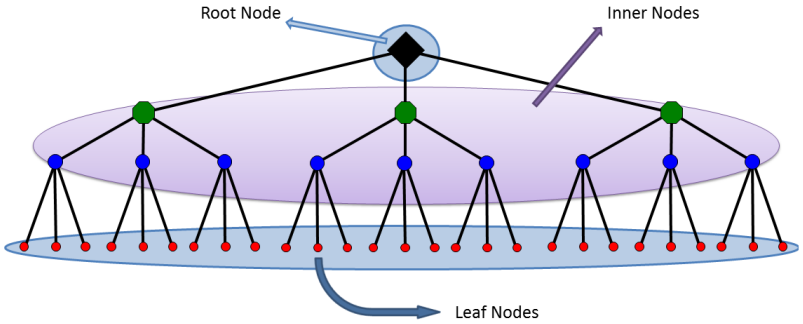

Figure 3.1: Example of Vocabulary Tree showing the different kinds of nodes

3.2.2.3 Leaf nodes

The last kind of nodes are the ones that don't have "sons", because they are in the last level of the branch. They can be created either because the depth D has been reached or because there weren't enough descriptors in the cluster to perform another division. There are up to K^D leaf nodes in a tree. They are very important because they are the visual words that compose our visual vocabulary. Each leaf node stores the following information:

- *Center*: The centroids of the clusters. They have the same dimension as the feature descriptors, in this case 128. They are visual words, computed by the k-means algorithm as the mean of all the observations assigned to the cluster. In this case, they compose the visual vocabulary of our database.
- *sub*: The inverted index. It is a table made of two rows. The first row contains the IDs of the objects that have some descriptors in this cluster. The second one indicates the $n_{i,k}$ parameter from the TF-IDF weighting scheme, i.e., the number of occurrences of the visual word i in the k-th object. We could also define it as the count of how many descriptors has each object in this cluster. In the i-th leaf node:

Obj(i,1)	Obj(i,2)	Obj(i,3)	...	Obj(i,k)
n(i,1)	n(i,2)	n(i,3)	...	n(i,k)

Figure 3.2: Inverted Index

- *w*: The weight of the visual word. It is the *idf* parameter explained in the Section 2.1.3.4. This parameter quantifies the importance of the visual word depending on the number of objects where it appears. It is necessary in the matching process.
- *des*: The locations of the descriptors that belong to this cluster in their respective images. We store them because we need them in the Geometric Verification step.
- *idx*: It is a vector containing the path of the tree until reaching the node. The length of this vector is the level of its node, and their components describe which node composes the path in every level.

It is important to notice that the feature descriptors are not stored in the leaf nodes. We only keep their locations in the different views of the objects and the inverted index. Consequently, we save an important storage space.

3.3 Matching and Multi-View Vocabulary Trees

As explained above, we are now building vocabulary trees that don't stop growing until they get less than k within a cluster. This causes that the clusters are very small and thus their centers are very close to the descriptors. In this scenario, the leaf nodes cells in the feature space are very concrete. Now we have a larger number of leaf nodes than before but made with fewer descriptors. Thus, there are only a few possible weights for leaf nodes, its variance has been significantly reduced. Also the effect of the descriptor noise is more important now, because it may cause a wrong clustering when the descriptors lie in the borders of the cells. With these new settings, we observed that the TF-IDF weighting scheme described in Section 2.1.3.4 can be more discriminative than before. This motivates us to adapt the weighting scheme to this new situation and mitigate these negative effects.

3.3.1 Object-based TF-IDF

We call the new TF-IDF weighting scheme as *Object-based TF-IDF* because our database doesn't consist anymore just on *images*, now we are talking about *objects*. We have several images taken from different points of view for each object. It is a new conception.

First, we have introduced a counter. In this new situation there are much more visual words than before, but they are more specific, which makes more important where these visual words appear than their weights. This parameter compensates the poorer power of discrimination of the leaf nodes weights by giving more importance to which objects are matched with the query. It is a counter of how many query descriptors have been matched with each database object. In practice, it is implemented as a counter initialized to 0 for all objects, and each time that a query descriptor is associated to a leaf node, we increase by one the count for all the database objects that have some descriptor composing this visual word. At the end of the matching progress, we multiply this accumulated count to the scores of the objects. In Section 2.1.3.4 we defined $\mathcal{I}_k$ as the set of leaf nodes matched with the query for the k-th database object. Its size, $|\mathcal{I}_k|$, will be the factor introduced in equation 2.5.

The second factor we introduced helps us to compensate the descriptor noise. We observed that in most descriptor mismatch cases, the query descriptors lie in the border of two clusters due to the query descriptor noise, which is caused by scale, rotation variations and occlusion. Therefore, we decided to use the ratio of the two closest distances between the incoming query descriptor and cluster centers as the uncertainty of visual words. We assign credibility values to each visual word of vocabulary tree. The credibility values can be calculated as

$$c_{i,k} = 1 - d_l(1)/d_l(2), \tag{3.1}$$

where $d_l(1)$ is the Euclidean distance between the query descriptor and the centroid of its closet leaf node and $d_l(2)$ is the same but related to the second closest leaf node. The credibility value reflects the confidence of the query matching. It is close to 1 when the query feature can be clearly distinguished between first and second closest leaf nodes. It is close to 0 when the query feature lies close to the border of two leaf nodes. Therefore, it is reasonable to assign the credibility values to visual words to punish the visual words uncertainty. And based on test result, we further take square of the credibility value to intensify its effect. Thus, the final scoring scheme, the *Object-based TF-IDF*, has this expression:

$$s_k = |\mathcal{I}_k| \sum_{\mathbf{i} \in |\mathcal{I}_k|} w_{i,k} c_{i,k}^2. \tag{3.2}$$

3.3.2 Fast 3D Geometric Verification

To perform the 3D GV in an efficient way, we have created a method called *Fast 3D Geometric Verification* [33]. Although it is not the topic of this thesis, we will explain it briefly.

As explained in Section 2.1.4.1, RANSAC works well but it is too slow. It takes too much time to execute it in a real-time application. To avoid it, we use an idea from [32] to discern between inliers and outliers. We sort the feature correspondences between query and database by counting the number of the descriptors under the associated leaf-nodes. Usually, the leaf node which contains less descriptors is more unique and reliable. We choose the ones with small number of descriptors.

In our earlier work [3], explained in Section 2.2.2.1, we use the equation (2.8) to combine the 3D world coordinates of the feature locations in the client (W^c) and the server (W^s). We used the Helmert-constrained RANSAC to estimate the parameters, but again it requires high computational complexity that makes it too slow.

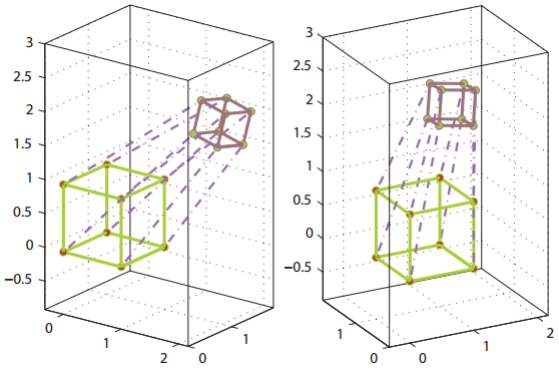

Figure 3.3: Misalignments between 3D objects

For this reason, instead of estimating the parameters in (2.8), it is enough to only calculate the misalignment between 3D world coordinates of query (q) and database (p) features:

$$g(q,p) = \log_2[1 + \left\|\vec{w^c} - \vec{w^s}\right\|] \tag{3.3}$$

where g is the function of calculating the 3D misalignment, $\vec{w^c} \epsilon W^c$, $\vec{w^s} \epsilon W^s$. The 3D misalignment only depends on the transformation between two coordinate systems, it remains a constant value for all correct matches. With the variance of the 3D misalignment we can know the inconsistency of 3D geometry:

$$J_k = varg(q,p), \tag{3.4}$$

For a set of correspondences which contains consistent matches, the variance J is usually small. Then, the variance of the 3D misalignments allows us to rearrange the candidates list that we obtained in the previous step.

Chapter 4

Experimental Results

In order to evaluate the performance of the vocabulary tree, several experiments have been carried out, evaluating different parameters involved in the VT construction. The results are presented in this chapter. Firstly, in Section 4.1, we explain the details of our multi-view database used in all the experiments. After that, in Section 4.2, *Performance*, we study the behaviour of the VT. Finally, the complexity in the object matching process is presented in Section 4.3.

4.1 Database

The multi-view dataset used to evaluate our work is called *Stockholm Buildings*[1]. It consists on 254 images of 50 different buildings of Stockholm. For the client side we have 100 more images of the same 50 buildings, two per each. The images have been taken at different times and from different point-views. Thus, there are changes in lighting conditions, scale and perspective. The hardware used was a Cannon IXUS50 digital camera that has a resolution of 2592 x 1944 pixels.

As explained in Section 2.2.1, the feature descriptors have been structured in hierarchical pyramids in function of the number of views where they appear. The most reliable descriptors are situated at the top of the pyramid, while the ones at the bottom usually belong to background objects and have a poorer discriminative quality. By truncating this pyramid, we perform a feature selection from the top of the pyramid. The number of views per object is not constant, it varies between 2 and 10, depending on the object, what represents an additional difficulty in the feature selection. The query features are selected and encoded with the rate-constrained feature selection method from our earlier work [3].

[1]Database: `http://www.ee.kth.se/~haopeng/sthlmbuildings`

4.2 Performance

In this section, we present, as introduced at the beginning of the chapter, the results of several experiments studying the performance of the vocabulary tree regarding different factors.

In the figures, the Y axis represents the *Recall*, defined as the percentage of the 50 query objects that are correctly placed in the first position of the candidates list after the matching process. The X axis shows the *Datarate* in KB/query, the size of the query packets of descriptors sent to the server to recognise the objects. When the datarate increases, more descriptors are sent and thus the recognition is usually better. We have tested a wide range of datarates: from a small quantity of descriptors, to a set of descriptors with a comparable size to a JPEG compressed image.

Note that there are some irregularities in the performance curves of some trees. At some points, the recall goes down when the datarate increases, and later it goes up again. We think it happens due to the randomness of the initial points selection in k-means algorithm and the fact that the global optimum is rarely achieved. Anyways, the tendency of increasing the recall when using a higher datarate is clear.

4.2.1 Adding Views

This first has been designed to demonstrate the power of the multi-view approach. Similarly as in [34], we wanted to confirm that the increase in the recall when adding new views to the database still occurs when using a VT. With this purpose, four trees with exactly the same specifications have been constructed with the only difference in the origin of the descriptors used to construct it. We have used four different databases, all of them composed by multi-view images, but the number of views per object is different in each of them.

The first database has been constructed using only 2 views per object. Only the descriptors that appeared in both views have been taken on account. We always discard the descriptors that appear just in a single view. Thus, there isn't any hierarchy between the descriptors in this database, all of them appear in two views. After that, the second database has been generated using 3 views per object. In this case there is a two levels hierarchy: the descriptors that appear in 3 views and the others that appear only in 2. The same process has been applied to the third database, but using 4 views. The discrimination among the features is clearer now, with three different possible levels. Finally, the last database has been constructed with all the views available, a different number for each object.

To make a fair comparison between the databases, we decided not to truncate the pyramids by views, because the first database has single level of hierarchy. We decided to load a constant number of descriptors from the top of the pyramid. We chose to select the 500 top descriptors of every in all databases.

Thus, the number of descriptors stays constant all the time, the only change is their robustness due to the quality of the classification. 25000 descriptors have been used in this experiment. The tree configuration chosen in this case has been $k = 12$, and $D = 3$, which makes a total of 1728 leaf nodes. This size of tree is shallow compared to the memory-constrained vocabulary tree, but we decided to experiment with the conventional approach before testing our new way of building trees. Thus, it also uses the original TF-IDF weighting scheme explained in 2.1.3.4. This have been our results:

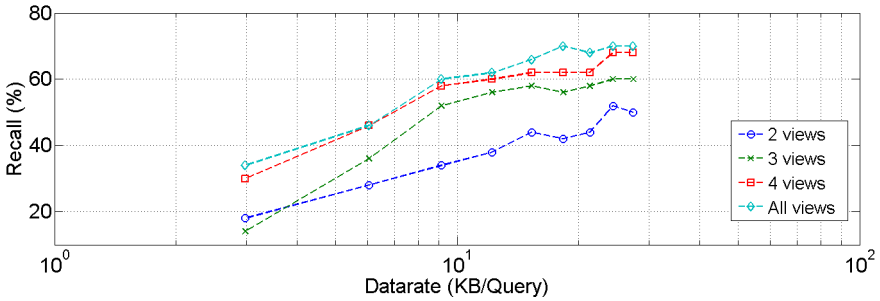

Figure 4.1: Adding views experiment

The results show an important increase of the recall-rate when we include more views per object. The change is specially significant when passing from 2 views to 3. It is also remarkable when using 4 views. The difference between the 4 views database and the one with all the views is not as wide as the others, but still show an increase. The conclusion is that the multi-view approach offers an important improvement in the VT performance, confirming the results obtained in [34].

4.2.2 Codebook Size

In this second experiment, we have decided to study the performance of the tree in function of the size of the visual vocabulary. As described in Section 3.1.3, the size of the visual vocabulary for a given database strongly influences the VT performance, mainly due to the quantization error. In this case, we have used the whole database with all the views, without truncating it, which consists on a total of 508061 descriptors. We have configured 4 vocabulary trees with different number of leaf nodes to study the evolution in their performances. We use the ratio between the number of descriptors and leaf nodes as an orientation parameter of the size of the tree compared to the database. We still use the TF-IDF without modifications. The configurations of the trees are the following:

- VT 1: $k = 8$, and $D = 4$. Ratio $\approx$ 125 descriptors per leaf node
- VT 2: $k = 10$, and $D = 4$. Ratio $\approx$ 50 descriptors per leaf node
- VT 3: $k = 12$, and $D = 4$. Ratio $\approx$ 25 descriptors per leaf node
- VT 4: $k = 15$, and $D = 4$. Ratio $\approx$ 10 descriptors per leaf node

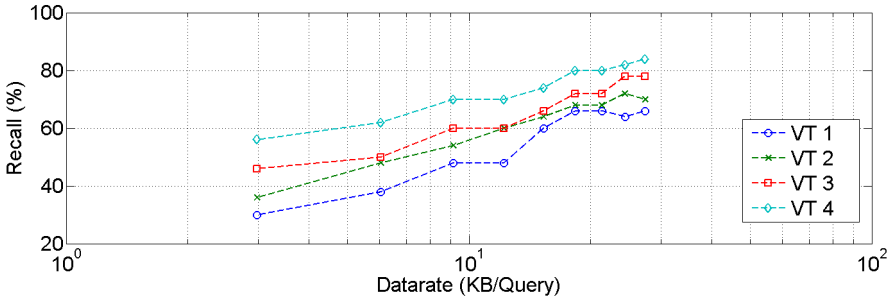

Figure 4.2: Comparison between codebook sizes

We demonstrate that given a database, the performance increases significantly when the number of leaf nodes is higher, due to a more accurate quantization.

4.2.3 Memory-Constrained Vocabulary Trees

From the two previous experiments we can extract two conclusions: the VT performance is increased with a good selection of multi-view features and also when the size of the visual vocabulary is big related to the size of the database. In this third experiment, we show the improvement in the recall rate that we obtain when we combine these two ideas in the *Memory Constrained Vocabulary Tree* with the *Object-based TF-IDF*.

We have built two trees with the same shape: $k = 8$, and $D = 5$. The difference between them is the number of descriptors from the database that we have used in the process. In blue we can see the tree made using the whole database, without truncating it at any point (508061). In red we can see the behaviour of the *Memory Constrained Vocabulary Tree*, that has been built using a truncation to get the top features from the pyramidal structure. The number of descriptors used in this tree has been 116517. The improvement in the recall rate is significant.

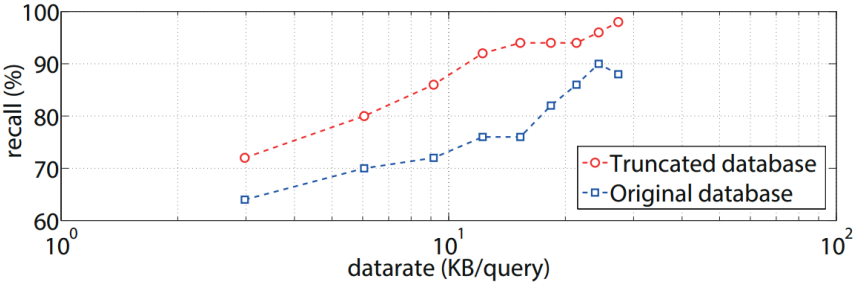

Figure 4.3: Memory-constrained vocabulary tree

The truncation has been done between groups of descriptors, using complete levels of the pyramid. In other words, we don't split the levels, we load them completely. In this particular VT the process of selecting the features has consisted on loading level per level, for every object, starting from the top, until we have at least a 15% of the total descriptors of the object. Some objects have more descriptors than others in the construction of the VT, but the TF-IDF weighting scheme corrects that detail.

4.2.4 3D Geometric Verification

Finally, the last experiment has been designed to show the different performances of some Geometric Verification methods. We have used the 2D RANSAC (very used in conventional MVS), 3D RANSAC, and our new creation, *Fast 3D Geometric Verification*. The VT tested is the same memory-constrained VT used in the previous experiment. Although the recalls can seem similar (especially for RANSAC 3D and the Fast 3D), there is an important improvement on complexity in the Fast 3D. For this reason, it is more suitable for a practical implementation.

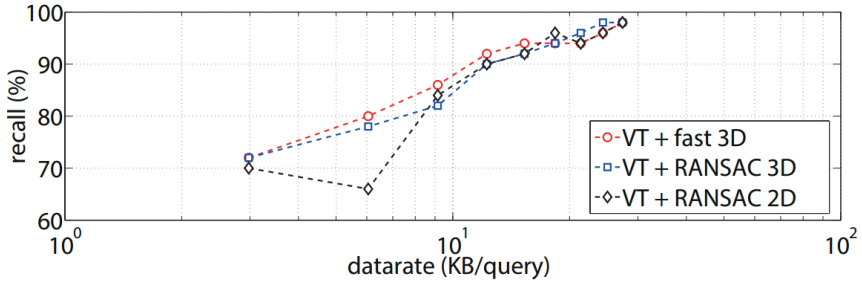

Figure 4.4: Comparison between GV methods

4.3 Computational Complexity of Matching

One of the most important characteristics of the vocabulary tree is its efficiency. It is only needed to make K dot products per level, to calculate the Euclidean distance between the query descriptor and the nodes of the VT. In total, KD dot products are enough to find the leaf node that best suits a query feature descriptor. This is, along its reduced storage space, the reason for its usage in MVS.

About the GV, the *Fast 3D Geometric Verification* is also improving the efficiency. To obtain a top five ranked images, the fast 3D geometric verification method needs only 0.16 seconds to achieve an average recall of 90%. The 2D RANSAC algorithm needs 3 seconds and the 3D RANSAC algorithm needs 13 seconds to achieve the same recall level.

Chapter 5

Conclusions

In this chapter, the last one of the thesis, a short summary of the work done in the thesis is presented, along with some possible ways to affront the future work regarding the multi-view vocabulary trees in mobile 3D visual search.

5.1 Summary of Results

The most important conclusion from our work is that we have constructed different from conventional vocabulary trees based on a multi-view approach for real-time mobile 3D visual search that has demonstrated to outperform significantly the current recall-rate of MVS. Also, it doesn't need a higher storage space in the server, even less space is needed to handle it. We call this new kind of VT *Memory Constrained Vocabulary Tree*.

Its construction has been possible thanks to the multi-view features extracted using a multiple images for each object. By structuring them in a hierarchical pyramid, we are able to select a small number of high quality features to construct a memory-efficient vocabulary trees for low computation complexity requirements.

We have designed a new scoring function to improve the recognition performance of multi-view vocabulary tree. We modified the TF-IDF weighting scheme to create the *Object-based TF-IDF*, that can give better results for our trees.

On the other hand, as the 3D geometry information is incorporated in the multi-view vocabulary tree, it allows us to design an algorithm for fast 3D geometric verification that results more efficient than the RANSAC-based models. The experimental results show that our multi-view vocabulary trees improve the recall-datarate performance significantly.

5.2 Future Work

In this last section, we comment some possible ways of improving the multi-view vocabulary tree and improve the work.

- Something interesting would be to try our method in bigger databases. Our database *Stockholm Buildings* contains only 254 images for 50 different objects, which is quite small compared to the thousands of objects that can form databases used in real applications. To know the performance of our work in a more realistic application much more images are needed.
- Another possible improvement could be to experiment with new TF-IDF weighting schemes that maybe could give us a better performance. We only use the leaf nodes in the weighting, but some approaches apply a hierarchical scoring, i.e., they use all the nodes of the path across the tree from the root to the leaf node.
- An exhaustive study about different ways of selecting the features from the top of the hierarchical pyramid could be also an interesting option.

Bibliography

[1] B. Girod, V. Chandrasekhar, D. Chen, N.-M. Cheung, R. Grzeszczuk, Y. Reznik, G. Takacs, S. Tsai, and R. Vedantham, "Mobile visual search," *Signal Processing Magazine, IEEE*, vol. 28, no. 4, pp. 61–76, July 2011.

[2] B. Girod, V. Chandrasekhar, R. Grzeszczuk, and Y. Reznik, "Mobile visual search: Architectures, technologies, and the emerging mpeg standard," *MultiMedia, IEEE*, vol. 18, no. 3, pp. 86–94, March 2011.

[3] H. Li and M. Flierl, "Mobile 3d visual search using the helmert transformation of stereo features," in *ICIP'13*, 2013, pp. 3470–3474.

[4] L. Fei-Fei and P. Perona, "A bayesian hierarchical model for learning natural scene categories," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, June 2005, pp. 524–531 vol. 2.

[5] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2, 2006, pp. 2161–2168.

[6] D. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1150–1157 vol.2.

[7] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in *In ECCV*, 2006, pp. 404–417.

[8] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and B. Girod, "Chog: Compressed histogram of gradients a low bit-rate feature descriptor," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, June 2009, pp. 2504–2511.

[9] G. Takacs, Y. Xiong, R. Grzeszczuk, V. Chandrasekhar, W. chao Chen, K. Pulli, N. Gelfand, T. Bismpigiannis, and B. Girod, "Outdoors augmented reality on mobile phone using loxel-based visual feature organization," in *In Proceeding of ACM international conference on Multimedia Information Retrieval*, 2008, pp. 427–434.

[10] S. Chatzichristofis and Y. Boutalis, "Fcth: Fuzzy color and texture histogram - a low level feature for accurate image retrieval," in *Image Analysis for Multimedia Interactive Services, 2008. WIAMIS '08. Ninth International Workshop on*, May 2008, pp. 191–196.

[11] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, pp. 145–175, 2001.

[12] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 1, pp. 105–119, Jan 2010.

[13] S. Leutenegger, M. Chli, and R. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, Nov 2011, pp. 2548–2555.

[14] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, Nov 2011, pp. 2564–2571.

[15] H. Bannour, L. Hlaoua, and B. el Ayeb, "Survey of the adequate descriptor for content-based image retrieval on the web: Global versus local features," in *(CORIA 2009) COnférence en Recherche d'Infomations et Applications, 6th French Information Retrieval Conference, Presqu'île de Giens, France*, 2009, pp. 445–456.

[16] M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid, "Evaluation of gist descriptors for web-scale image search," in *International Conference on Image and Video Retrieval*. ACM, july 2009. [Online]. Available: http://lear.inrialpes.fr/pubs/2009/DJSAS09

[17] K. Grauman and B. Leibe, *Visual Object Recognition*. Morgan and Claypool Publishers, 2011.

[18] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: A survey," *FnT Comp. Graphics and Vision*, pp. 177–280, 2008.

[19] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, 2004.

[20] Y. Meng and D. B. Tiddeman(supervisor), "Implementing the scale invariant feature transform(sift) method."

[21] T. Lindeberg, "Scale Invariant Feature Transform," vol. 7, no. 5, p. 10491, 2012, revision 142692.

[22] S. Winder, G. Hua, and M. Brown, "Picking the best daisy," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, June 2009, pp. 178–185.

[23] J. Sivic and A. Zisserman, "Video google: a text retrieval approach to object matching in videos," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, Oct 2003, pp. 1470–1477 vol.2.

[24] H.-H. Bock, "Clustering methods: A history of k-means algorithms," in *Selected Contributions in Data Analysis and Classification*, ser. Studies in Classification, Data Analysis, and Knowledge Organization, P. Brito, G. Cucumel, P. Bertrand, and F. de Carvalho, Eds. Springer Berlin Heidelberg, 2007, pp. 161–172. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73560-1_15

[25] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theor. Comput. Sci.*, vol. 38, pp. 293–306, 1985. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(85)90224-5

[26] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[27] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[28] X. Wang, M. Yang, and K. Yu, "Efficient re-ranking in vocabulary tree based image retrieval," in *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*, Nov 2011, pp. 855–859.

[29] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford, "Okapi at trec-3," 1996, pp. 109–126.

[30] C. Moulin, C. Barat, and C. Ducottet, "Fusion of tf.idf weighted bag of visual features for image classification," in *Content-Based Multimedia Indexing (CBMI), 2010 International Workshop on*, June 2010, pp. 1–6.

[31] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. [Online]. Available: http://doi.acm.org/10.1145/358669.358692

[32] S. S. Tsai, D. M. Chen, G. Takacs, V. Chandrasekhar, R. Vedantham, R. Grzeszczuk, and B. Girod, "Fast geometric re-ranking for image-based retrieval," in *Proceedings of the International Conference on Image Processing, ICIP 2010, September 26-29, Hong Kong, China*, 2010, pp. 1029–1032.

[33] D. E. Mars, H. Wu, H. Li, and M. Flierl, "Multi-view vocabulary tree using fast 3d geometric verification embedded matching and ranking," in *Submitted to IEEE Data Compression Conference (DCC), 2014*, October 2014.

[34] X. Lyu, H. Li, and M. Flierl, "Hierarchically structured multi-view features for mobile visual search," in *Data Compression Conference (DCC), 2014*, March 2014, pp. 23–32.

[35] J. H. Wei-Xue Liu and H. R. Karimi, "Research on vocabulary sizes and codebook universality," *Abstract and Applied Analysis*, vol. 2014, 2014.

[36] G. Schindler, M. Brown, and R. Szeliski, "City-scale location recognition," in *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*, 2007.

[37] D. Arthur and S. Vassilvitskii, "K-means++: the advantages of careful seeding," in *In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.