Chapter 10
Mobile Image Search: Challenges and Methods

Xin Yang and K.T. Tim Cheng

Abstract The proliferation of camera-equipped mobile devices with enhanced mobile computing power and network connectivity results in a rising demand for mobile image search. Although image search has been studied extensively over the last few decades, most existing solutions, developed for desktops and server platforms, are not suitable for mobile devices. In this chapter, we provide an overview of challenging issues unique in mobile search scenarios and present several techniques addressing these challenges. Specifically, we focus the discussion on: (1) robust, distinctive, and fast feature extraction on mobile devices, (2) compact indexing structure for efficient feature matching, and (3) multimodel context-aware data fusion for improving performance of mobile image search.

10.1 Introduction

Mobile devices such as smartphones and tablets have experienced phenomenal growth. Their computing power has grown enormously and the connectivity of smartphones has also gone through rapid evolution. A wide range of radios including cellular broadband, Wi-Fi, Bluetooth, and NFC available in today's smartphones enable users to communicate with other devices, interact with the Internet, and exchange their data with and running their computing tasks in the clouds. The abundance of camera-equipped mobile devices and low-latency data networks has led to an increasing demand for mobile image search. A mobile image search system, which has the ability to identify objects in a picture and use the recognized object as a starting point for search (often referred to as 'query-by-image'), can support a wide range

X. Yang (✉)
Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
e-mail: xinyang@umail.ucsb.edu

K.T.T. Cheng
Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA
e-mail: timcheng@ece.ucsb.edu

G. Hua and X.-S. Hua (eds.), *Mobile Cloud Visual Media Computing*,
DOI 10.1007/978-3-319-24702-1_10

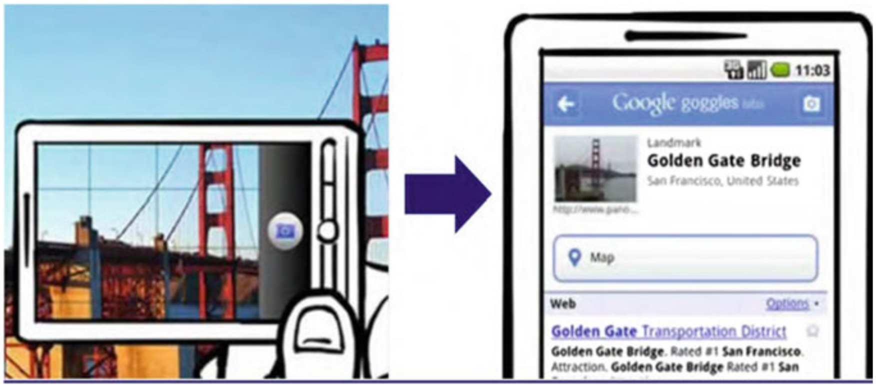
11:03
Google goggles
Landmark
Golden Gate Bridge
San Francisco, United States
Map
Web
Options
Golden Gate Transportation District
Golden Gate Bridge. Rated #1 San Francisco.
Attraction. Golden Gate Bridge Rated #1 San

Fig. 10.1 An example of a search of a famous landmark on mobile devices (*Source* Google Goggles.)

of mobile applications. For instance, a user can take a picture of a famous landmark to search for information about it (as shown in Fig. 10.1), a picture of a product's barcode or a book cover to search for online stores selling the product/book, a picture of a movie poster to view reviews or to find tickets at nearby theatres, or a picture of a restaurant menu in French for translation to English. Such a query-by-image capability allows users to search for items without typing any text. For its image-based translation capability, the app recognizes printed text and uses optical character recognition (OCR) to produce a snippet and then translate it into another language.

Image search has been studied extensively for several decades. To improve scalability, efficiency, and accuracy, the three key performance metrics of image search, a number of algorithms for image representation and indexing have been developed [1–3]. Most of the existing solutions are based on and optimized for the laptop, desktop, and server platforms and the unique challenges and opportunities presented by a mobile scenario have not been thoroughly analyzed. In the following, we first present a general pipeline for image search, and then we discuss the main challenges for image search on mobile devices. Then, we present some existing solutions and finally conclude the chapter by pointing out some promising directions for mobile image search.

10.2 Pipeline

A conventional image search pipeline (Fig. 10.2) consists of two phases: (1) offline database construction and (2) online image search. In the offline phase, feature extraction is performed for every database image. An indexing structure which encodes feature descriptors of all database images is constructed. Popular indexing methods for efficient and scalable image search include locality sensitive hashing (LSH) and

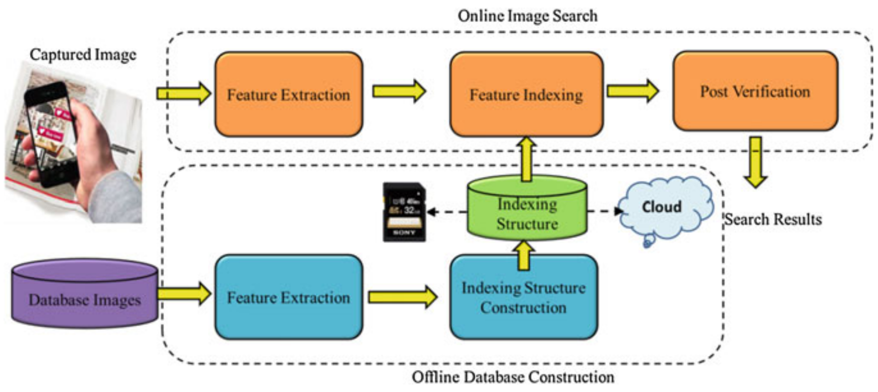

Fig. 10.2 A general pipeline for image search on mobile devices

bag-of-words (BOW) model. More details about these two methods will be provided in Sect. 10.4. In the online search phase, features of a captured image are first extracted, each of which is then used to query the database using an indexing structure for finding a matching feature in the database. The database image which has the most matching features with the capture image is considered as candidate targets. Postverification (using methods such as RANdom SAmple Consensus, RANSAC, [4]) and PROgressive Sample Consensus (PROSAC) [5] is then conducted among candidate images to find the most relevant images to the query image.

The performance of feature extraction and indexing algorithms greatly affect the user experience of mobile image search apps. Ideally, we demand (1) highly robust and distinctive image features which can provide good search accuracy even for large databases and meanwhile can be extracted efficiently on mobile devices, and (2) a compact indexing structure which can be stored on mobile devices in order to avoid network latency for accessing data on a server. However, each of these goals remains challenging in mobile scenarios despite advances in image search algorithms as well as mobile hardware. For example, mobile CPUs are still not fast enough to achieve real-time performance for compute-intensive image processing operations, such as feature extraction. Popular feature extraction algorithms (e.g., SIFT [6], which is widely used for image search) require a large amount of floating point operations, which is slow to compute on mobile CPUs. In addition, limited memory space of mobile embedded system (i.e., 1–2 GB, shared by all apps and the OS, for today's smartphones and tablets) could be a limiting factor when extracting features for an image search. This is because feature extraction such as SIFT often requires large sets of intermediate data to be stored in memory as analysis is performed sequentially. The total amount of memory usage of each stage grows linearly with the size of the original image. For moderate- to high-resolution images, this process could easily exhaust memory resources. Limited storage space of mobile devices also prohibits indexing structure of a large database from being stored locally on a mobile device. As a result, most existing systems employ a client–server architecture. That is, sending the captured image or processed image data (e.g., image features) to a server (or a cloud) via Internet and performing feature indexing and post verification on the server side. The client–server mode may suffer from network latency and thus cannot meet the efficiency requirement for image search apps which demand real-time performance. In practice, according to the size of databases, available local storage and computing resources, and performance requirements, developers need to make decisions for a number of issues to optimize the user experience (e.g., choosing different algorithms, offloading different amount and which parts of the workload from the client to the server side, etc.). In the following, we first elaborate key challenges in mobile image search. In Sect. 10.4, we present some potential solutions addressing the challenges in feature extraction and indexing on mobile devices. We also present existing efforts for fusing multimodel context-aware information for mobile image search. In Sect. 10.5, we conclude the paper and discuss some future work.

10.3 Challenges

Mobile devices differ from general computing environments in several aspects. The design of a mobile image search system must take into account the following inherent limitations of mobile devices:

(a) **Lower Processing Power of CPU**. The design objectives of modern mobile application processors are more than just performance. Priority is often given to other factors such as low power consumption and a small form factor. Although the performance of mobile CPUs has achieved greater than 30X improvement within a short period of recent 5 years (e.g., ARM quad-core Cortex A-15 in 2014 vs. ARM 11 single-core in 2009), today's mobile CPU cores are still not powerful enough to achieve real-time performance for compute-intensive vision tasks such as sophisticated feature extraction and indexing algorithms. Graphics processing units (GPUs), which have been built into most application processors, can help speed up processing via parallel computing [7, 8], but most feature extraction and indexing algorithms are designed to be executed sequentially and cannot fully utilize the capability of GPU cores in a mobile application processor.

(b) **Less Memory Capacity**. Mobile devices have less memory and lower memory bandwidth than desktop systems. The memory of today's high-end smartphones, such as Samsung Galaxy S5, is limited to 2GB of SDRAM and the memory size of mid- and entry-level phones is even smaller. This level of memory sizes is not sufficient for performing local image search using a large database. In order to realize efficient image search, the entire indexing structure of a database needs to be loaded and reside in main memory. The total amount of memory usage for an indexing structure usually grows linearly with the number of database images. For a database of a moderate size (e.g., tens of thousands of images), or a large size (e.g., millions of images), the indexing structure itself could easily exhaust memory resources. Several scalable mobile image search systems [9, 10] employ the client–server model to handle large databases. That is, sending the captured image or processed image data (e.g., image features) to a server (or a cloud) via Internet, performing feature indexing and post verification on the server side, and then sending the search results and associated information back to the mobile device. While Wi-Fi is a built-in feature for almost all mobile devices, connection to high-bandwidth access points is still not available anyplace, neither anytime. For connection to data networks, today's mobile devices rely on a combination of mobile broadband networks including 3G, 3.5G, and 4G. These networks, while providing acceptable network access speed for most apps, cannot support real-time responses for apps demanding a large amount of data transfer. Moreover, advanced mobile broadband networks still have limited availability in areas not having dense populations.

(c) **Small Screen Size**. Modern high-end smartphones boast displays which measure slightly less than seven inches diagonally. However, this size is still much smaller than that of a common desktop or laptop. Smaller screens greatly limit the amount of information that can be presented to a user. As a result, it requires a more effective display of search results and higher search accuracy in order to achieve satisfactory user experience.

(d) **Noisy Query**. The search precision for content-based image search still has significant room for improvement. Particularly in the mobile scenario, a user's query photo can be noisy due to clutter, occlusions, and large viewpoint changes. Therefore, a visual search on a large-scale database with noisy images based on a noisy query cannot achieve high accuracy. Modern smartphones, equipped with various sensors (e.g., GPS, accelerometer gyroscope, magnetometer, etc.), can provide various forms of context information. Mobile search systems can incorporate such context information to improve image search's accuracy, efficiency, and scalability. However, context information captured by a mobile device's sensors is noisy as well. Integrating such noisy context information into vision-based image search methods that can robustly improve accuracy and efficiency is not a trivial task at all.

10.4 Methods

In this section, we introduce recent work addressing some of the challenges that face mobile image search. Specifically, we describe existing solutions for (1) extracting robust and distinctive features efficiently on mobile devices (Sect. 10.4.1); (2) constructing compact indexing structure which can be stored locally on mobile devices or can facilitate precise and fast matching feature retrieval from a large database in the cloud (Sect. 10.4.2); and (3) fusing multimodel context information to improve search accuracy and reduce computational complexity (Sect. 10.4.3).

10.4.1 Robust, Distinctive, and Fast Feature Extraction on Mobile Devices

Local features (an example shown in Fig. 10.3a) have been widely used in many computer vision and pattern recognition apps. In contrast to global feature extraction which generates a single feature vector for an entire image, local feature extraction generates a set of high-dimensional feature vectors for an image. Local feature extraction typically consists of two steps: (1) interest point detection, also referred to as local feature detection, which selects a set of salient points in an image, and (2) interest point description, also referred to as local feature description, which transforms a small image patch around a feature point into a vector representation suitable for further processing. In comparison with a global feature representation, local features are more robust to various geometric and photometric transformations, occlusion, and background clutters and thus more suitable for mobile image search.

Local features' efficiency, robustness, and distinctiveness significantly affect the user experience and performance of a mobile image search system. In this section, we give an overview of mobile interest point detection and description. Due to space limitation, we only review some most representative methods, which do not represent a comprehensive survey.

Interest Point Detection

An interest point detector is an operator which attributes a saliency score to each pixel of an image and then chooses a subset of pixels with local maximum scores. A good detector should provide points that have the following properties: (1) *repeatability (or robustness)*, i.e., given two images of the same object under different image conditions, a high percentage of points on the object in both images can be chosen, (2) *distinctiveness*, i.e., the neighborhood of a detected point should be sufficiently informative so that the point can be easily distinguished from other detected points, (3) *efficiency*, i.e., the detection in a new image should be sufficiently fast to support real-time applications, and (4) *quantity*, i.e., a typical image should contain a

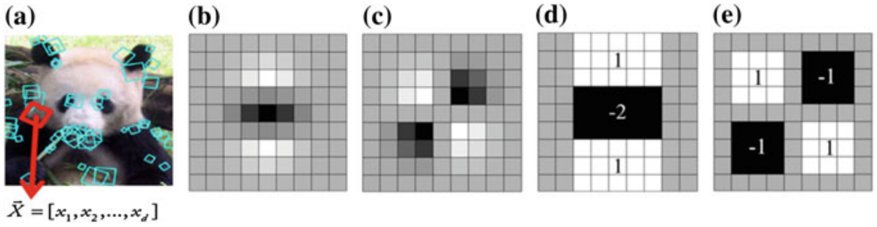

Fig. 10.3 **a** An exemplar image overlaid with detected local features. **b** and **c** are the discretized and cropped Gaussian second-order partial derivative in the y-direction and the xy-direction, respectively; **d** and **e** are SURF box filter approximation for L_{yy} and L_{xy}, respectively

sufficient number of detected points to cover a target object, so that it can be recognized using these detected points even under partial occlusion.

There exist a wide variety of interest point detectors. Some lightweight detectors [11] aim at high efficiency to target applications which demand real-time performance and/or mobile hardware platforms with limited computing resources. However, the performance of these detectors is relatively poor. As a result, it requires postverification to exclude false matches in the subsequent matching phase which often incurs a nontrivial runtime. Therefore, their overall runtime efficiency is not necessary high. On the other hand, several high-quality feature detectors [6, 12, 13] have been developed with a primary focus on robustness and distinctiveness. These detectors' ability to accurately localize correct targets from a large database makes them suitable for large-scale image search. However, the computational complexity of these detectors is usually very high, making them inefficient on a mobile device. Some recent efforts, e.g., [14], adapt these feature detection algorithms with respect to mobile platforms and optimize their performance and efficiency for mobile image search. In the following, we review the most representative methods for the lightweight detector, the high-quality detector, and algorithm adaptation. A thorough survey on local feature-based detectors can be found in [15].

a. Lightweight Detector: FAST

The FAST (Features from Accelerated Segmented Test) detector, proposed by Rosten and Drummond [11], is popular due to its highly efficient processing pipeline. The basic idea of FAST is to compare 16 pixels located on the boundary of a circle (radius is 3) around a central point, each of which is numbered from 1 to 16 clockwise. If the intensities of n consecutive pixels are all higher or all lower than that of the central pixel and n is greater than a predefined minimum threshold, then the central pixel is labeled as a potential feature point and n is defined as the response value for the central pixel. The final set of feature points is determined after applying a *nonmaximum suppression* step, which selects a potential point as a feature point if its response value is the local maximum within a small region. Because the FAST detector only involves a set of intensity comparisons with few arithmetic operations, it is highly efficient.

The FAST detector is not invariant to scale changes. To achieve scale invariance, Rublee et al. [16] employed a scale pyramid to an image and detected FAST feature points at each level in the pyramid. FAST could produce large responses along edges, leading to lower repeatability and distinctiveness compared to high-quality detectors such as SIFT [6] and SURF [12, 13]. To address this limitation, Rublee et al. further employed a Harris corner measure to order the FAST feature points and discard those with small responses to the Harris measure.

b. High-Quality Detector: SURF

The SURF (Speeded Up Robust Feature) detector, proposed by Bay et al. [12, 13], is one of the most popular high-quality point detectors in the literature. It is scale-invariant and based on the determinant of the *Hessian* matrix $H(X, \sigma)$:

$$H(X, \sigma) = \begin{bmatrix} L_{xx}(X, \sigma) & L_{xy}(X, \sigma) \\ L_{xy}(X, \sigma) & L_{yy}(X, \sigma) \end{bmatrix} \tag{10.1}$$

where $X = (x, y)$ is a pixel location in an Image I, σ is a scale factor, $L_{xx}(X, \sigma)$ is the convolution of the Gaussian second-order derivative in the x direction, similarly for L_{yy} and L_{xy} (see Fig. 10.3b, c).

To speed up the computation, a SURF detector approximates the Gaussian second-order partial derivatives with a combination of box filter responses (see Fig. 10.3d, e), computed using the integral image technique [17]. Denoting the approximated derivatives as D_{xx}, D_{xy} and D_{yy}, the approximate Hessian determinant can be expressed as:

$$\det\left(H_{approx}\right) = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{10.2}$$

A SURF detector computes Hessian determinant values for every image pixel i over scales using box filters of a successively larger size, yielding a determinant pyramid for the entire image. Then it applies a $3 \times 3 \times 3$ local maximum extraction over the determinant pyramid to select interest points' locations and corresponding salient scales.

To achieve rotation invariance, SURF relies on gradient histograms to identify a dominant orientation for each detected point. An image patch around each point is rotated to its dominant orientation before computing a feature descriptor. Specially, the dominant orientation of a SURF detector is computed as follows. First, the entire orientation space is quantized into N histogram bins, each of which represents a sliding orientation window covering an angle of $\pi/3$. Then SURF computes gradient responses of every pixel in a circular neighborhood of an interest point. Based on the gradient orientation of a pixel, SURF maps it to the corresponding histogram bins and adds its gradient response to these bins. Finally, the bin with the largest responses is utilized to calculate the dominant orientations of interest points.

Comparing to FAST, SURF point detection involves much more complex computations and, thus, is much slower than FAST. The runtime limitation of SURF is further exacerbated when running a SURF detector on a mobile platform. Table 10.1 compares the runtime performance of a FAST detector and a SURF detector running on a single CPU core in a mobile device (Motorola Xoom1) and a laptop (Thinkpad T420) respectively. Running a FAST detector takes 170 ms on a Motorola Xoom1 (whose application processor consists of dual-core ARM Cortex-A9) and 40 ms on

Table 10.1 Comparison of FAST and SURF detectors on mobile device and PC

Time detector	Mobile device (ms)	PC (ms)	Speedup
FAST detector	170	40	4x
SURF detector	2156	143	15x

an i5-based Thinkpad, yielding a 4x speed gap. However, running a SURF detector on them takes 2156 and 143 ms respectively, indicating a 15x speed gap.

FAST is more efficient, but less robust and distinctive than SURF. As a result, FAST usually fails to achieve satisfactory performance for mobile image search apps which still demand sufficiently high search accuracy from a large database and the ability of handling content with large photometric/geometric changes.

c. Algorithm Adaptation: Accelerating SURF on Mobile Devices

There are several techniques aiming at improving SURF's efficiency. They include exploiting coherency between consecutive frames [18], employing graphics processing units (GPUs) for parallel computing, and optimizing various aspects of the implementation [8]. A solution proposed in [14] analyzes the causes for a SURF detector's poor efficiency and large overhead on a mobile platform, and propose a set of techniques to adapt the SURF algorithm to a mobile platform. Specially, two mismatches between the computations used in the SURF algorithm and common mobile hardware platforms are identified as the sources for its significant performance degradation:

- *Mismatch between SURF's data access pattern and a mobile platform's small cache size*. A SURF detector relies on an integral image and accesses it using a sliding window of successively larger size for different scales. But a 2D array is stored in a row-based fashion in memory (cache and DRAM), not in a window-based fashion; pixels in a single sliding window reside in multiple memory rows (illustrated in Fig. 10.4a). The data cache size of a mobile application processor (AP), typically 32KB for today's devices, is too small to cache all memory rows

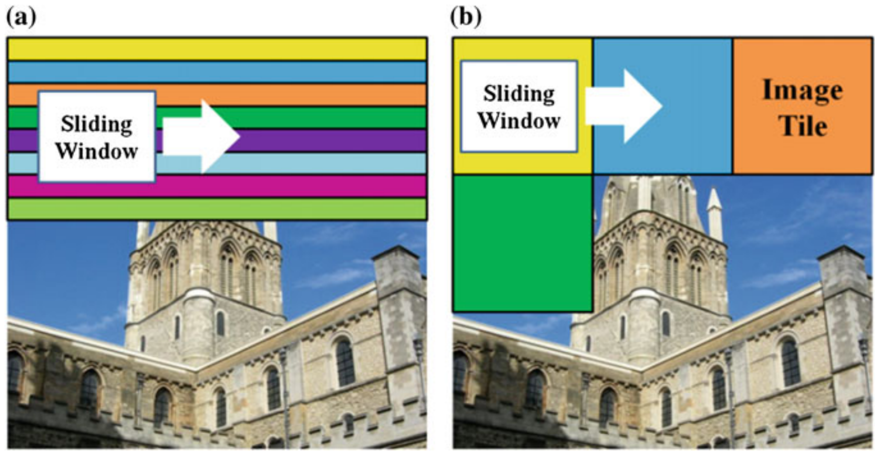

Fig. 10.4 Illustration of data locality and access pattern in **a** the original SURF detector, and **b** the tiled SURF. Each color represents data stored in a unique DRAM row. In the original SURF, a sliding window needs to access multiple DRAM rows, leading to frequent cache misses, while in tiled SURF, all required data within a sliding window can be cached

for pixels involved in one sliding window, leading to cache misses and cache line replacements and, in turn, incurring expensive memory access.

- *Mismatch between a huge amount of data-dependent branches in the SURF algorithm and high pipeline hazard penalty of the mobile platform.* To identify a dominant orientation, a SURF detector analyzes a gradient histogram. During this analysis, every pixel around an interesting point is mapped to corresponding histogram bins via a set of branch operations, i.e., "If-then-Else" expressions. The total number of pixels involved in this analysis is huge and thus the entire process involves an enormous amount of data-dependent branch operations. However, the branch predictor and the speculation of out-of-order execution of an ARM-based mobile CPU core are less sophisticated that of a laptop, desktop, or server processor. Consequently, it incurs higher pipeline hazard penalties, yielding significant performance degradation.

To address the problem caused by the mismatch between the data access pattern of SURF and the small cache size of a mobile CPU, a tiled SURF was proposed in [14] which divides an image into tiles (illustrated in Fig. 10.4b) and performs point detection for each tile individually to exploit local spatial coherences and reduce external memory traffic. To avoid pipeline hazards penalties, two solutions were proposed in [14] to remove data-dependent branch operations. The first solution is to use an alternative implementation: instead of using "If-then-Else" expressions, a lookup table is used to store the correlations between each orientation and the corresponding histogram bins. This alternative does not change the functionality and other computations, but trades memory for speed. The second solution is to replace the original gradient histogram method with a branching-free orientation operator based on gradient moments (i.e., GMoment) [19]. The gradient-moment-based method may slightly degrade the robustness of a SURF detector, but can greatly improve its runtime on mobile platforms.

Tables 10.2 and 10.3 compare the runtime cost and the Phone-to-PC runtime ratio between the original and adapted SURF, respectively [14]. The Phone-to-PC ratio, defined in Eq. (10.3), is the runtime of a program running on a mobile CPU divided by that on a desktop CPU, which reflects the speed gap between them.

$$\textit{Phone-to-PC ratio} = \frac{\text{runtime on a mobile platform}}{\text{runtime on an x86-based PC}} \tag{10.3}$$

Table 10.2 Runtime cost comparison on three mobile platforms

Time (ms)	Droid	Thunderbolt	Xoom1
U-SURF	1310	525	461
U-SURF tiling	930	356	243
O-SURF	7700	2495	2156
O-SURF lookup table	4264	1820	1178
O-SURF GMoment	1516	613	519
O-SURF tiling + GMoment	1053	404	269

Table 10.3 Speed ratio comparison on three mobile platforms

Phone-to-PC ratio (x)	Droid	Thunderbolt	Xoom1
U-SURF	20	8	7
U-SURF tiling	14	7	4
O-SURF	54	17	15
O-SURF lookup table	18	7	6
O-SURF GMoment	19	8	7
O-SURF tiling + GMoment	13	7	3

The evaluation experiments were performed on three mobile devices: a Motorola Droid which features an ARM Cortex-A8 processor, an HTC Thunderbolt which uses a Scorpion processor, and a Motorola Xoom1 which uses a dual-core ARM Cortex-A9 processor. The first two rows of Tables 10.2 and 10.3 compare the runtime cost and the Phone-to-PC ratio of upright SURF (U-SURF) without and with tiling. As expected, tiling can greatly reduce runtime cost by 29–47 %. It reduces the Phone-to-PC ratio by 12.5–42.9 % on these three devices. The reduction in Phone-to-PC ratio indicates that the mismatch between the data access pattern and a small cache size of a mobile CPU causes more severe runtime degradation on mobile CPUs than desktop CPUs. So alleviating this problem is critical for performance optimization when porting algorithms to a mobile CPU. The 3rd–5th rows of Tables 10.2 and 10.3 compare the results of oriented SURF (O-SURF) with branch operations, O-SURF using a lookup table and using *GMoment* [19], respectively, which show that using a lookup table or using the *GMoment* method can greatly reduce the overall runtime and the Phone-to-PC ratio on three platforms. The reduction in the Phone-to-PC ratio further confirms that branch hazard penalty has a much greater runtime impact on a mobile CPU than on a desktop CPU. Choosing proper implementations or algorithms to avoid such penalties is critical for a mobile task. The last rows of Tables 10.2 and 10.3 show the results of applying both adaptation ideas to O-SURF: the runtime of SURF on mobile platforms can be reduced by 6X–8X.

Local Feature Description

Once a set of interest points has been extracted from an image, their content needs to be encoded in descriptors that are suitable for matching. In the past decade, the most popular choices for this step are the SIFT and the SURF descriptors. SIFT and SURF have successfully demonstrated their high robustness and distinctiveness in a variety of computer vision applications. However, the computational complexity of the SIFT descriptor is too high for real-time applications with tight runtime constraints. While SURF accelerates SIFT by 2X–3X, it is still not sufficiently fast for real-time applications running on a mobile device. In addition, SIFT and SURF are high-dimensional real-value vectors which demand large storage space and high computing power for matching. The booming development of real-time mobile apps has stimulated significant advances in binary descriptors that are more compact, and faster to compute than SURF-like features while maintaining a satisfactory feature quality. Notable work includes BRIEF [8] and its variants rBRIEF [16], BRISK [20], FREAK [21], and LDB [22–24]. In the following, we review three representative descriptors: SURF, BIREF, and LDB.

a. SURF: Speed Up Robust Features

The SURF descriptor aims to achieve robustness to lighting variations and small positional shifts by encoding the image information in a localized set of gradient statistics. Specifically, each image patch is divided into 4×4 grid cells. In each cell, SURF computes a set of summary statistics $\sum d_x$, $\sum |d_x|$, $\sum d_y$, and $\sum |d_y|$, resulting in a 64-dimensional descriptor. The first-order derivatives d_x and d_y can be calculated very efficiently using box filters and integral images.

Motivated by the success of SURF, a further optimized version proposed in [8] takes advantage of the computational power available in CUDA [25]-enabled graphics cards. This GPUSURF implementation has been reported to perform feature extraction for a 600×480 image at a frame rate up to 20 Hz, thus making feature extraction an affordable processing step. However, to date, most mobile GPU cores do not support CUDA. Furthermore, mobile GPU cores, in addition to being much less powerful than desktop GPU chips, share the same external memory and memory buses with CPU cores and other heterogeneous cores in the application processor. Thus porting an implementation from desktop-based GPUs to mobile GPUs remains a tedious task with unpredictable performance gain [26, 27].

b. BRIEF: Binary Robust Independent Elementary Features

The BRIEF descriptor, proposed in [28], primarily aims at high-computational efficiency for construction and matching, and a small footprint for storage. The basic idea of BRIEF is to directly generate bit strings by simple binary tests comparing pixel intensities in an image patch. More specifically, a binary test τ is defined and performed on a patch p of size $S \times S$ as

$$\tau(p; x, y) = \begin{cases} 1 & \text{if } \; I(p,\ x) \ < \ I(p,\ y) \\ 0 & \text{otherwise} \end{cases} \tag{10.4}$$

where $I(p, x)$ is the pixel intensity at location $x = (u, v)^T$. Choosing a set of $n_d(x, y)$-location pairs uniquely defines the binary test set and consequently leads to an n_d -dimensional bit string that corresponds to the decimal counterpart of

$$\sum_{1 \le i \le n_d} 2^{i-1} \tau(p; x_i, y_i) \tag{10.5}$$

By construction, the tests of Eq. (10.5) consider only the information at single pixels; therefore, the resulting BRIEF descriptors are very sensitive to noises. To increase

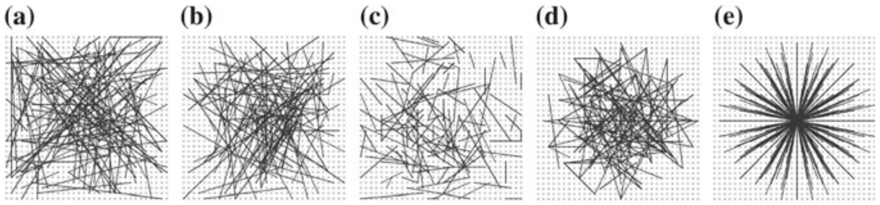

Fig. 10.5 Different approaches to choose the test locations. Sampling (X, Y) locations from a patch of size $S \times S$ according to **a** a uniform distribution $(-\frac{S}{2}, \frac{S}{2})$, **b** an isotropic Gaussian distribution $(0, \frac{1}{25}S^2)$, **c** a nonisotropic Gaussian distribution $X \sim (0, \frac{1}{25}S^2)$, $Y \sim (x_i, \frac{1}{25}S^2)$, **d** random distribution, **e** from a coarse polar grids. Courtesy of [28]

the stability and repeatability, the authors proposed to smooth pixels of every pixel pairs using Gaussian or box filters before performing the binary tests.

The spatial arrangement of binary tests greatly affects the performance of the BRIEF descriptor. In [28], the authors experimented with five sampling geometries for determining the spatial arrangement, as shown in Fig. 10.5a–e. Experimental results demonstrate that the tests which are randomly sampled from an isotropic Gaussian distribution—Gaussian $(0, \frac{1}{25}S^2)$ where the origin of the coordinate system is the center of a patch and S is the patch size—give the highest recognition rate.

BRIEF and its enhanced versions of BRIEF [16, 20, 21] are very efficient to compute, store, and to match (simply computing the Hamming distance between descriptors via XOR and bit count operations). These runtime advantages make these binary descriptors attractive for real-time applications and handheld devices. However, they often utilize overly simplified information, i.e., only intensities of a subset of pixels within an image patch, and thus have low discriminative ability. Lack of distinctiveness results in a huge number of false matches when matching against a large database. Expensive postverification methods (e.g., RANdom SAmple Consensus (RANSAC) [4]) are usually required to discover and validate matching consensus, increasing the runtime of the entire process.

c. LDB: Local Difference Binary

LDB (Local Difference Binary), a binary descriptor, achieves similar computational speed and robustness as BRIEF and other state-of-the-art binary descriptors, yet offering greater distinctiveness. The better quality of LDB is achieved through three schemes. First, LDB utilizes average intensity I_{avg} and first-order gradients, d_x and d_y, of grid cells within an image patch. Specifically, the internal patterns of the image patch is captured through a set of binary tests, each of which compares the I_{avg}, d_x and d_y of a pair of grid cells (illustrated in Fig. 10.6a, b). The average intensity and gradients capture both the DC and AC components of a patch, thus they provide a more complete description than other binary descriptors. Second, LDB employs a multiple gridding strategy to encode the structure at different spatial granularities (Fig. 10.6c). Coarse-level grids can cancel out high-frequency noise while fine-level grids can capture detailed local patterns, thus enhancing distinctiveness. Third, LDB

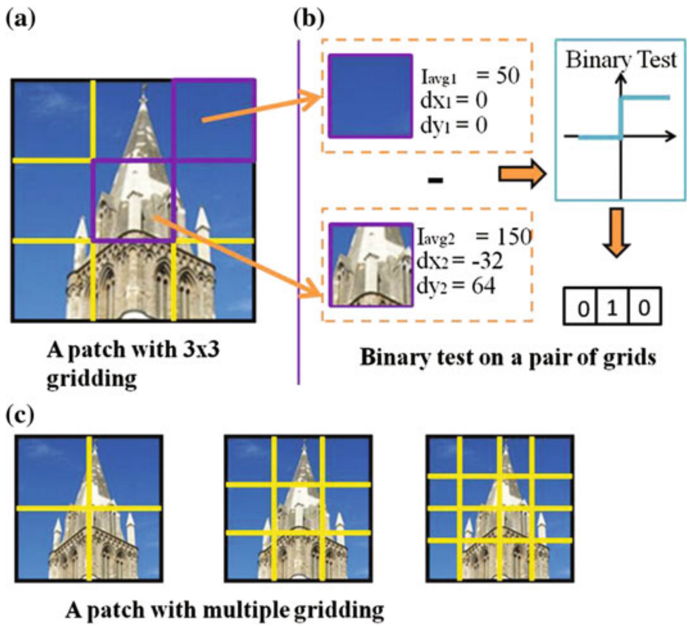
(a)
(b)
(c)
Iavg1 = 50
dx1 = 0
dy1 = 0
-
Iavg2 = 150
dx2 = -32
dy2 = 64
Binary Test
0 1 0
A patch with 3x3 gridding
Binary test on a pair of grids
A patch with multiple gridding

Fig. 10.6 Illustration of LDB extraction. **a** An image patch is divided into 3 × 3 equal-sized grids. **b** Compute the intensity summation (I), gradient in x and y directions (d_x and d_y) of each patch, and compare I, d_x and d_y between every unique pair of grids. **c** 3-level gridding (with 2 × 2, 3 × 3, and 4 × 4 grids) is applied to capture information at different granularities

leverages a modified AdaBoost method [23] to select a set of salient bits. The modified AdaBoost targets the fundamental goal of idea binary descriptors: minimizing distance between matches while maximizing them between mismatches, optimizing the performance of LDB for a given descriptor length. Computing LDB is highly efficient: relying on integral images, the average intensity, and first-order gradients of each grid cell can be obtained by only 4–8 add/subtract operations.

Accelerating Feature Extraction via Mobile GPU Cores

Mobile application processor includes embedded graphics processing unit (GPU) cores and other hardware accelerators in addition to the CPU cores. GPUs allow for large quantities of instructions to be executed in parallel and efficient for floating point operations. While originally intended for rendering 2D and 3D graphics, GPUs have been at the core of a branch of study known as general-purpose computation on graphics processing units (GPGPU) [29]. GPGPU technology extends the programmability of GPUs to enable nongraphics applications with high parallelizability to run more efficiently than on a CPU. In the context of mobile image search, where sequential feature extraction algorithms are often used, In order to employ GPGPU, feature extraction algorithms need to be broken up into smaller subtasks which can be executed in parallel. Several efforts have been made to improve the parallelization of feature extraction. For example, in [30], a number of stages in the SIFT algorithm are parallelized to run on consumer desktop GPUs, decreasing runtime by a factor of 10. However, it should be pointed out that mobile GPUs have very different characteristics compared to desktop GPUs: a mobile GPU has fewer cores, smaller graphics memory, lower GPU bus bandwidth, sharing the same memory bus with mobile CPUs, and variant architecture when compared to a desktop GPU. To fully utilize the mobile GPUs, new feature extraction algorithms must be devised with the aim to be executed concurrently. It is also necessary to characterize the computing capability of the target mobile CPU-GPU platform in order to identify the condition that offloading tasks to GPU cores leads to an optimized performance [7].

10.4.2 Compact Indexing Structure for Fast Matching on Mobile Devices

To search relevant database images for a captured image, an image search system matches each feature descriptor in the captured image to database features to find the query feature's nearest neighbor (NN). If the similarity between a feature and its NN being above a predetermined threshold and they comply with a geometric model, this feature pair is considered a matched pair. The database object which has most matched features to the captured image is considered as the recognized object.

Fast and accurately retrieving the NN of a local feature from a large database is the key to efficient and accurate image search, ensuring a satisfactory user experience and scalability for mobile image search apps. Two popular techniques commonly used for large-scale NN matching are Locality Sensitive Hashing (LSH) and bag-of-words (BOW) matching.

LSH: Locality Sensitive Hashing

LSH [31] is widely used for approximate NN search. The key of LSH is a hash function, which maps similar descriptors into the same bucket of a hash table and distinct descriptors into different buckets. To find the NN of a query descriptor, we first retrieve its matching bucket and then check all the descriptors within the matched bucket using a brute-force search.

For binary features, the hash function can simply be a subset of bits from the original bit string; descriptors with a common sub-bit-string are casted to the same table bucket. The size of the subset, i.e., the hash key size, determines the upper bound of the Hamming distance among descriptors within the same buckets. To improve the detection rate of NN search based on LSH, two techniques, namely *multitable* and *multiprobe*, are usually used. The multitable technique stores the database descriptors in several hash tables, each of which leverages a different hash function. In the query phase, the query descriptor is hashed into a bucket of every hash table and all descriptors in each of these buckets are then further checked for matching. Multitable improves the detection rate of NN search at the cost of higher memory usage and longer matching time, which is linearly proportional to the number of hash tables used. Multiprobe examines both the bucket in which the query descriptor falls and its neighboring buckets. While multiprobe would result in more matching checks of database descriptors, it actually requires fewer hash tables than multitable and thus incurs lower memory usage. In addition, it allows a larger key size and in turn smaller buckets and fewer matches to check per bucket.

Bag-of-Words Matching

Bag-of-Words (BoW) matching [3] is an effective strategy to reduce memory usage and support fast matching via a scalable indexing scheme such as an inverted file. Typically, BoW matching quantizes local image descriptors into visual words and then computes the image similarity by counting the frequency of words co-occurrences. However, it completely ignores the spatial information which may degrade the accuracy. To address this limitation of BoW matching, several approaches have been proposed to compensate the loss of spatial information. For example, geometric verification [32], designed for general image-matching applications, verifies local correspondences by checking their homography consistency. Wu et al. presented a bundling feature matching scheme [33] for partial-duplicate image detection. In their approach, sets of local features are bundled into groups by maximally stable extremal regions (MSER) [34] detected regions, and robust geometric constraints are then enforced within each group. Spatial pyramid matching [35], which considers approximate global geometric correspondences, is another scheme to enforce geometric constraints for more accurate BoW matching. This scheme partitions the image into increasingly finer sub-regions and computes histograms of local features detected within each sub-region. To compute the similarity between two images, the distance between histograms at each spatial level is weighted and summed together. These above-mentioned schemes yield more reliable local-region matches by enforcing various geometric constraints. However, these schemes are very compute-expensive. Thus, for real-time mobile image search, the indexing procedure based on these methods must be conducted on the server side or in the cloud.

10.4.3 Fusing Multimodel Context-Aware Information for Mobile Image Search

At present, the processing power and memory capacity of mobile devices are still too limited for image search apps solely relying on sophisticated visual feature extraction and matching methods. Modern smartphones have equipped a wide range of sensors, e.g., compass, accelerometer, gyroscope, GPS, etc. greatly enrich the devices' functionalities and provide various forms of context information to facilitate image search. For instance, Global Position System (GPS) location is important information for landmark images. In the meanwhile, a growing fraction of images in image databases are tagged with geographical information. As of February 2009, there are more than 100 million geotagged images on Flickr [36]. By leveraging GPS to identify the location of a mobile devices and utilizing a compass (or in combination with other sensors) to determine the direction that the device is heading to, an image search system could retrieve related images which have similar geotagged and direction information as the query image.

The problem, however, is that built-in sensors usually lack sufficient accuracy, thus cannot provide satisfactory performance for search tasks. For instance, since GPS drift can be as much as 100 meters, in densely built areas or using a noisy and large-scale database, more false positive images from surrounding locations will be included. Several studies proposed to combine these vision-based and sensor-based methods. For example, in [10], authors proposed two modes, parallel and serial, to integrate location information in a mobile landmark image search system. In parallel mode, query data from content and location is processed independently, and then the results are combined together through a linear combination approach. In serial mode, location information is first applied to narrow down the search space, and then results will be refined and re-ranked based on visual information. Serial integration can significantly reduce the search scope for the captured landmark, which in turn will greatly improve search precision and speed. However, it may also incur the risk of losing some true positives, i.e., a worse recall, due to the absence of location tags. Another work fuse visual and GPS information is presented in [37]. In this work, the authors proposed to combine visual tracking and GPS for outdoor building visualization. The user can place virtual models on Google Earth and the app can retrieve and visualize them based on the user's GPS location.

The trend of integrating more sensors into mobile devices has not stopped yet. For example, Google has just released a new mobile platform, Tango, which integrates six Degree-of-Freedom motion sensors, depth sensors, and high-quality cameras. Amazon has announced their new Fire phone which includes four cameras tucked into the front corners of the phone, in additional to other motion sensors. Advances in mobile hardware offer the opportunities to gain richer contextual information surrounding a mobile device and in turn open a door for new approaches to best utilizing all available multimodel information.

10.5 Conclusions

The advancement of mobile technology, in terms of hardware computing power, seamless connectivity to the cloud and fast computer vision algorithms, have raised image search into the mainstream of mobile apps. Following the widespread popularity of a handful of killer image search applications already commercially available, it is believed that mobile image search will expand exponentially in the next few years. The advent of mobile image search will have a profound and lasting impact on the way people use their smartphones and tablets. These emerging mobile image search apps will turn our everyday world into a fully interactive digital experience, from which we can see, hear, feel and even smell the information in a different way. This emerging direction will push the industry toward truly ubiquitous computing and a technologically converged paradigm.

The scalability, accuracy, and efficiency of the underlying techniques (i.e., feature extraction and indexing) are key factors influencing user experience of mobile image search apps. New algorithms in computer vision and pattern recognition, such as lightweight feature extraction, have been developed to provide efficiency, compactness on low-power mobile devices, and meanwhile maintain sufficiently good accuracy. Several efforts are also made to analyze particular hardware limitations for executing existing feature extraction and indexing algorithms on mobile devices and explore adaption techniques to address these limitations. In addition to advances in the development of lightweight computer vision algorithm, a variety of sensors have been integrated into modern smartphones, enabling location recognition (e.g., via GPS) and device tracking (e.g., via gyroscope, accelerometer, and magnetometer) at little computational cost. However, due to large noise of low-cost sensors equipped in today's smartphones, the accuracy of location recognition is usually low and cannot meet the requirement for apps which demand high accuracy. Fusing visual information with sensor data is a promising direction to achieve both high accuracy and efficiency, and we shall see an increasing amount of research work along this direction in the near future.

References

1. Yang, X., Zhu, Q., Cheng, K.-T.: Near-duplicate detection for images and videos. In: ACM Workshop on Large Scale Multimedia Retrieval and Mining, Beijing, October 2009
2. Jegou, H., Douze, M., Schmid, C.: Packing bag-of-features. In: International Conference on Computer Vision, September 2009
3. Sivic, J., Zisserman, A.: Video Google: a text retrieval approach to object matching in videos. In: Proceedings of International Conference on Computer Vision, vol. 2, pp. 1470–1477 (2003)
4. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**, 381–395 (1981)
5. Chum, O., Matas, J.: Matching with PROSAC—progressive sample consensus. Proc. Comput. Vis. Pattern Recognit. **1**, 220–226 (2005)
6. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60**(2), 91–110 (2004)
7. Cheng, K.T., Yang, X., Wang, Y.-C.: Performance optimization of vision apps on mobile application processor. In: International Conference on Systems, Signals and Image Processing (IWSSIP), Bucharest, Romania, 7–9 July 2013
8. Terriberry, T.B., French, L.M., Helmsen, J.: GPU accelerating speeded-up robust features. In: Proceedings of the 3D Data Processing, Visualization and Transmission (2008)

9. Xie, X., Lu, L., Jia, M.L., Li, H., Seide, F., Ma, W.Y.: Mobile search with multimodel queries. Proc. IEEE **96**(4), 589–601 (2008)
10. Yang, X., Pang, S., Cheng, K.-T.: Mobile image search with multimodel context-aware queries. In: ACM International Workshop on Mobile Vision, June 2010
11. Rosten, E., Drummond, T.: Machine learning for high speed corner detection. In: Proceedings of the European Conference on Computer Vision (2006)
12. Bay, H., Ess, A., Tuytelaars, T., Gool, L.V.: SURF: speeded-up robust features. In: Proceedings of the European Conference on Computer Vision (2006)
13. Bay, H., Ess, A., Tuytelaars, T., Gool, L.V.: Speeded-up robust features. In: Proceedings of the Conference on Vision and Image Understanding, vol. 110(3), June 2008
14. Yang, X., Cheng, K.T.: Accelerating SURF detector on mobile devices. In: ACM International Conference on Multimedia, Nara, Japan, October 2012
15. Tuytelaars, T., Mikolajczyk, K.: Local invariant feature detectors: a survey. J. Found. Trends Comput. Graph. Vis. **3**, 177–280 (2008)
16. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: an efficient alternative to SIFT or SURF. In: Proceedings of the International Conference on Computer Vision (2011)
17. Simard, P., Bottou, L., Haffner, P., LeCun, Y.: Boxlets: a fast convolution algorithm for signal processing and neural networks. In: Proceedings of the Neural Information Processing Systems (NIPS) (1998)
18. Ta, D.N., Chen, W.C., Gelfand, N., Pulli, K.: SURFTrac: efficient tracking and continuous object recognition using local feature descriptors. In: Proceedings of the Conference on Vision and Pattern Recognition (2009)
19. Rosin, P.L.: Measuring corner properties. J. Comput. Vis. Image Underst. **73**(2), 291–307 (1999)
20. Leutenegger, S., Chli, M., Siegwart, R.: BRISK: binary robust invariant scalable keypoints. In: Proceedins of the Computer Vision on Pattern Recognition (2011)
21. Alahi, A., Ortiz, R., Vandergheynst, P.: FREAK: fast retinal keypoint. In: Proceedings of the Computer Vision on Pattern Recognition (2012)
22. Yang, X., Cheng, K.T.: Local difference binary for ultrafast distinctive feature description. IEEE Trans. Pattern Anal. Mach. Intell. **36**(1), 188–194 (2014)
23. Yang, X., Cheng, K.T.: Learning optimized local difference binaries for scalable augmented reality on mobile devices. IEEE Trans. Vis. Comput. Graph. **20**(6), 852–865 (2014)
24. Yang, X., Cheng, K.T.: LDB: an ultrafast feature for scalable augmented reality on mobile device. In: Proceedings of International Symposium on Mixed and Augmented Reality, pp. 49–57 (2012)
25. CUDA: http://www.nvidia.com/object/cuda_home_new.html
26. Wang, Y.-C., Cheng, K.-T.: Energy and performance characterization of mobile heterogeneous computing. In: IEEE Workshop on Signal Processing System, Canada, October 2012
27. Wang, Y.-C., Cheng, K.-T.: Energy-optimized mapping of application to smartphone platform—a case study of mobile face recognition. In: IEEE Workshop of Embedded Computer Vision, USA (2011)
28. Calonder, M., Lepetit, V., Strecha, C., Fua, P.: BRIEF: binary robust independent elementary features. In: Proceedings of the European Conference on Computer Vision (2010)
29. GPGPU: http://gpgpu.org/developer
30. Sinha, S., Frahm, J., Pollefeys, M., Genc, Y.: GPU-based video feature tracking and matching. In: Workshop on Edge Computing Using New Commodity Architectures (2006)
31. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of International Conference on Very Large Databases, pp. 518–529 (1999)
32. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: Proceedings of Computer Vision and Pattern Recognition, pp. 1–8 (2007)
33. Wu, Z., Ke, Q.F., Isard, M., Sun, J.: Bundling features for large scale partial-duplicate web image search. In: Proceedings of Computer Vision and Pattern Recognition, pp. 25–32 (2009)

34. Matas, J., Chum, O., Urban, M., Pajdla, T.: Robust wide baseline stereo from maximally stable extremal regions. In: Proceedings of British Machine Vision Conference, pp. 384–396 (2002)
35. Lazebnik, S., Schmid, C., Ponce, J.: Beyond bags of features: spatial pyramid matching for recognizing natural scene categories. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 2169–2178 (2006)
36. Kleban, J., Moxley, E., Xu, J.J., Manjunath, B.S.: Global annotation on georeference photographs. In: ACM International Conference on Image and Video Retrieval, Greece, July 2009
37. Naimark, L., Foxlin, E.: Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. In: Proceedings of the International Symposium on Mixed and Augmented Reality, pp. 27–36 (2002)